THÔNG TIN SINH VIÊN

Mã số sinh viên: 23127447
Họ và tên: Nguyễn Thanh Owen
Lớp: 23CLC06
Đồ án: Final Project
Email: ntowen23@clc.fitus.edu.vn

Bài 1

Câu a

Trước khi tiến hành phân tích và xây dựng mô hình, cần kiểm tra dữ liệu để đảm bảo không có giá trị bị thiếu hoặc dòng trống gây ảnh hưởng đến kết quả.

Các bước thực hiện:

Bước 1: Đọc và xem thông tin dữ liệu

- Đọc dữ liệu từ file `stockton4.csv`.
- Xem kích thước dữ liệu, 5 dòng đầu tiên và mô tả tổng quan (`info()`, `describe()`).

Bước 2: Xử lý dữ liệu

- Kiểm tra giá trị null (NaN) trong từng cột.
- Loại bỏ các dòng có giá trị bị thiếu.
- Kiểm tra và loại bỏ các dòng trống (nếu có).

Bước 3: Phân tích tương quan

- Vẽ biểu đồ để phân tích tương quan giữa giá bán nhà (`sprice`) và các đặc trưng độc lập:
  - Các biến số (`livarea` , `beds` , `baths` , `age`) được vẽ dưới dạng **biểu đồ phân tán** bình thường để quan sát xu hướng và mức độ phân tán dữ liệu.
  - Các biến nhị phân (`lgelot` , `pool`) cũng được vẽ bằng **biểu đồ phân tán**, trong đó trục X chỉ hiển thị hai giá trị **0** và **1**

Bước 1: Đọc và xem thông tin dữ liệu

In [157…

```
import pandas as pd

# Đọc dữ liệu
df = pd.read_csv("stockton4.csv")

# Dữ liệu ban đầu
print("Kích thước dữ liệu:", df.shape)
print(df)

# Thông tin tổng quát về dữ liệu
print("\nThông tin dữ liệu:")
print(df.info())
```

```
Kích thước dữ liệu: (1500, 7)
      sprice  livarea  beds  baths  lgelot  age  pool
0     138000       17     3    2.0       1   97     0
1     105700       21     4    2.5       0   18     0
2      22000        7     2    1.0       0   49     0
3     255000       30     3    3.0       1   23     0
4     203000       21     4    2.0       1   18     0
...      ...      ...   ...    ...     ...  ...   ...
1495  170000       21     4    2.5       0   20     0
1496  149000       25     5    3.0       0   21     0
1497  187000       20     3    2.5       0   40     0
1498  186635       16     3    2.0       0   35     0
1499  385000       43     3    3.0       1   38     0

[1500 rows x 7 columns]

Thông tin dữ liệu:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   sprice   1500 non-null   int64
 1   livarea  1500 non-null   int64
 2   beds     1500 non-null   int64
 3   baths    1500 non-null   float64
 4   lgelot   1500 non-null   int64
 5   age      1500 non-null   int64
 6   pool     1500 non-null   int64
dtypes: float64(1), int64(6)
memory usage: 82.2 KB
None
```

Bước 2: Xử lý dữ liệu

- Kiểm tra giá trị null (NaN) và loại bỏ các dòng bị thiếu dữ liệu.
- Kiểm tra và loại bỏ các dòng trống (nếu có).

In [158…

```python
# Kiểm tra số lượng giá trị null ở mỗi cột
print("\nSố lượng giá trị null ở mỗi cột:")
```

```
print(df.isnull().sum())

# Loại bỏ các dòng có giá trị null
df = df.dropna()

# Kiểm tra lại sau khi xử lý
print("\nKích thước dữ liệu sau khi loại bỏ null:", df.shape)

# Loại bỏ các dòng dữ liệu trống (nếu có)
df = df[~df.astype(str).apply(lambda row: row.str.strip().eq('').any(), axis=1)]

print("\nKích thước dữ liệu sau khi loại bỏ dòng trống:", df.shape)

# Dữ liệu sau khi xử lý
print("\nKích thước dữ liệu:", df.shape)
print(df)
```

```
Số lượng giá trị null ở mỗi cột:
sprice     0
livarea    0
beds       0
baths      0
lgelot     0
age        0
pool       0
dtype: int64

Kích thước dữ liệu sau khi loại bỏ null: (1500, 7)

Kích thước dữ liệu sau khi loại bỏ dòng trống: (1500, 7)

Kích thước dữ liệu: (1500, 7)
      sprice  livarea  beds  baths  lgelot  age  pool
0     138000       17     3    2.0       1   97     0
1     105700       21     4    2.5       0   18     0
2      22000        7     2    1.0       0   49     0
3     255000       30     3    3.0       1   23     0
4     203000       21     4    2.0       1   18     0
...      ...      ...   ...    ...     ...  ...   ...
1495  170000       21     4    2.5       0   20     0
1496  149000       25     5    3.0       0   21     0
1497  187000       20     3    2.5       0   40     0
1498  186635       16     3    2.0       0   35     0
1499  385000       43     3    3.0       1   38     0

[1500 rows x 7 columns]
```

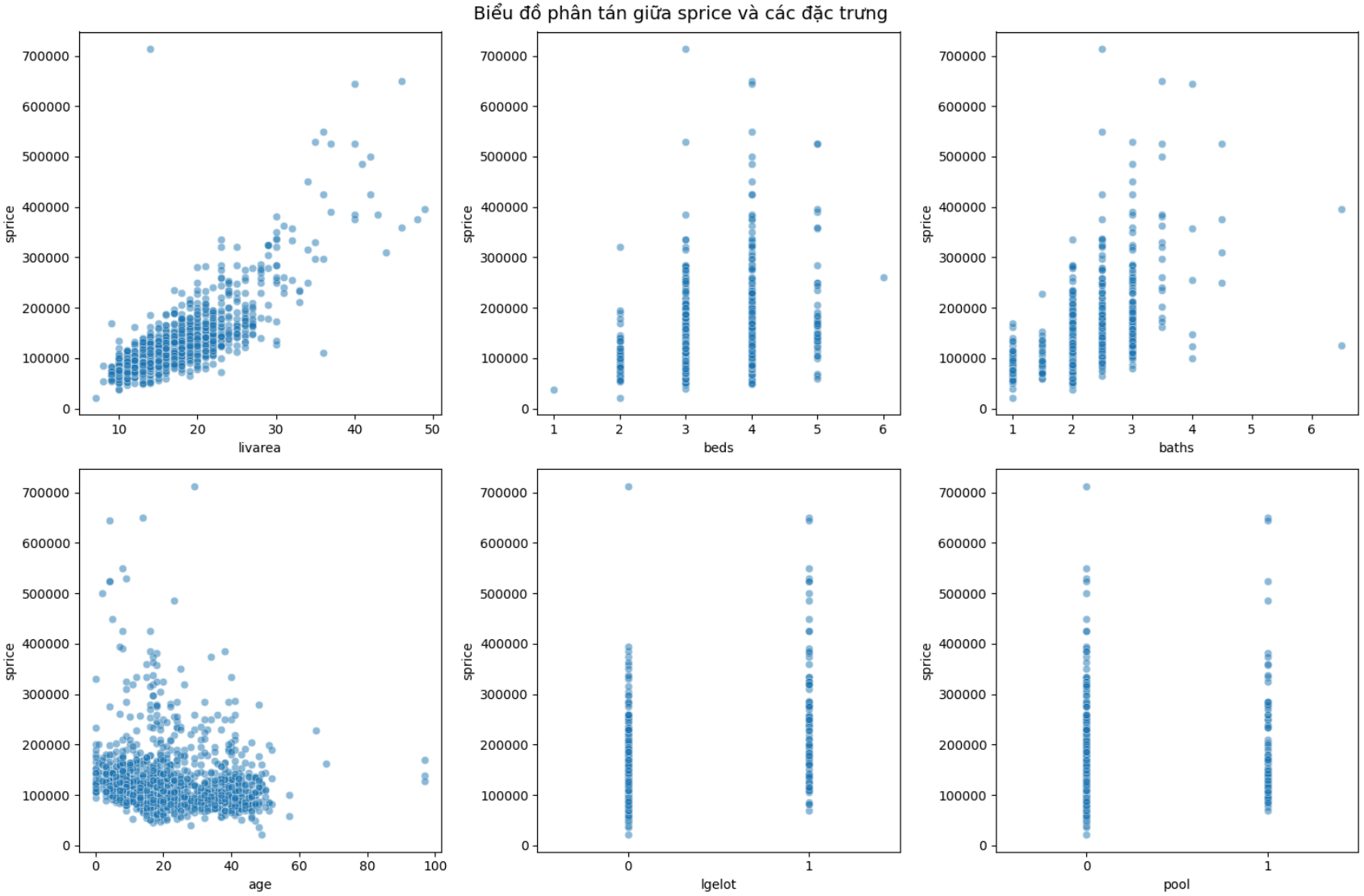
Bước 3: Phân tích tương quan

Vẽ biểu đồ để phân tích tương quan giữa giá bán nhà (`sprice`) và các đặc trưng độc lập

In [159…

```python
import matplotlib.pyplot as plt
import seaborn as sns

features = ['livarea', 'beds', 'baths', 'age', 'lgelot', 'pool']

plt.figure(figsize=(15,10))
for i, col in enumerate(features, 1):
```

```
    plt.subplot(2, 3, i)
    sns.scatterplot(x=df[col], y=df['sprice'], alpha=0.5)

    # nếu là biến nhị phân thì ép nhãn 0/1 và thu hẹp khoảng cách
    if col in ['lgelot', 'pool']:
        plt.xticks([0, 1], ['0','1'])
        plt.xlim(-0.5, 1.5)   # giảm khoảng cách giữa 0 và 1 cho các biểu đồ có biến nhị phân

    plt.xlabel(col)
    plt.ylabel("sprice")

plt.suptitle("Biểu đồ phân tán giữa sprice và các đặc trưng", fontsize=14)
plt.tight_layout()
plt.show()
```



Câu b

- Xây dựng **mô hình hồi quy tuyến tính đơn biến**:
  - Biến phụ thuộc: `sprice` (giá bán nhà).
  - Biến độc lập: lần lượt từng đặc trưng (`livarea` , `beds` , `baths` , `age` , `lgelot` , `pool`).

Các bước thực hiện

Bước 1: Chuẩn bị dữ liệu

- Biến phụ thuộc: `sprice` (giá bán nhà).
- Biến độc lập: lần lượt chọn từng đặc trưng trong tập dữ liệu (`livarea` , `beds` , `baths` , `age` , `lgelot` , `pool`).

Bước 2: Huấn luyện mô hình bằng OLS

- Tạo ma trận X gồm một cột hằng số (bias = 1) và cột đặc trưng được chọn.
- Sử dụng công thức:
  $\hat{\beta} = (X^T X)^{-1} X^T y$
- Tính ra hệ số β_0 (intercept) và β_1 (slope) cho từng mô hình hồi quy đơn biến.
- Tạo giá trị dự đoán $\hat{y}$ từ các hệ số này.

Bước 3: Đánh giá mô hình bằng chuẩn L2 của vector phần dư

- Tính **độ dài (chuẩn L2)** của vector phần dư giữa giá trị thực tế và dự đoán:

$$\|y - \hat{y}\| = \sqrt{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

- Trong đó:
  - y là vector giá trị thực tế.
  - $\hat{y}$ là vector giá trị dự đoán từ mô hình.
  - Vector phần dư là $y - \hat{y}$.
- Ý nghĩa:
  - Chuẩn L2 càng nhỏ → mô hình dự đoán càng chính xác.
  - Chuẩn L2 càng lớn → sai số tổng thể càng cao.

Bước 4: Trực quan hóa kết quả

- Vẽ biểu đồ scatter so sánh giữa giá trị thực tế `y` và giá trị dự đoán $\hat{y}$.

- Thêm đường chéo $y = x$ để đối chiếu.
- Các điểm càng nằm gần đường chéo thì mô hình dự đoán càng chính xác.

Bước 1: Chuẩn bị dữ liệu

- Xác định biến phụ thuộc: `sprice` (giá bán nhà).
- Xác định các biến độc lập: lần lượt chọn từng đặc trưng (`livarea` , `beds` , `baths` , `age` , `lgelot` , `pool`).
- Với mỗi biến độc lập, tạo ma trận X gồm một cột toàn 1 (để biểu diễn hệ số chặn β_0) và cột dữ liệu của biến đó.
- Tập biến phụ thuộc y được lấy từ cột `sprice` .

In [160…

```
# Biến phụ thuộc
y = df['sprice'].tolist()

# Biến độc lập
features = ['livarea', 'beds', 'baths', 'age', 'lgelot', 'pool']
```

Bước 2: Huấn luyện mô hình bằng OLS

- Với mỗi đặc trưng được chọn, xây dựng ma trận X gồm một cột toàn 1 (bias = 1) và cột đặc trưng.
- Sử dụng công thức:
  $\hat{\beta} = (X^T X)^{-1} X^T y$
- Trong đó:
  - β_0: hệ số chặn (intercept).
  - β_1: hệ số góc (slope) ứng với đặc trưng.
- Sau khi tính được $\hat{\beta}$, tạo giá trị dự đoán $\hat{y}$ cho từng mẫu dữ liệu.

Cài đặt các hàm cần thiết:

- `mat_transpose(matrix)` : chuyển vị ma trận.
- `mat_mult(A, B)` : nhân hai ma trận.
- `mat_inverse(matrix)` : nghịch đảo ma trận vuông bằng phương pháp khử Gauss-Jordan.
- `train_ols(X, y)` : thực hiện việc huấn luyện hồi quy tuyến tính theo công thức:
- `predict(X, beta)` : tính giá trị dự đoán $\hat{y}$ từ ma trận đặc trưng X và vector hệ số β.
  $\beta = (X^T X)^{-1} X^T y$

Trong đó:

- X: ma trận đặc trưng (có thêm cột 1.0).
- y: vector giá trị chất lượng rượu.
- β: vector trọng số (hệ số hồi quy).

In [161…

```
def mat_transpose(matrix):
    return [list(row) for row in zip(*matrix)]

def mat_mult(A, B):
    result = [[0 for _ in range(len(B[0]))] for _ in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            s = 0
            for k in range(len(B)):
                s += A[i][k] * B[k][j]
            result[i][j] = s
    return result

def mat_inverse(matrix):
    n = len(matrix)
    mat = [row[:] for row in matrix]
    identity = [[float(i == j) for j in range(n)] for i in range(n)]

    for i in range(n):
        diag = mat[i][i]
        for j in range(n):
            mat[i][j] /= diag
            identity[i][j] /= diag
        for k in range(n):
            if i != k:
                factor = mat[k][i]
                for j in range(n):
                    mat[k][j] -= factor * mat[i][j]
                    identity[k][j] -= factor * identity[i][j]
    return identity

def train_ols(X, y):
    X_T = mat_transpose(X)
    XTX = mat_mult(X_T, X)
```

```
    XTy = mat_mult(X_T, [[v] for v in y])
    XTX_inv = mat_inverse(XTX)
    beta = mat_mult(XTX_inv, XTy)
    return beta

def predict(X, beta):
    return [sum(beta[j][0] * row[j] for j in range(len(beta))) for row in X]
```

Huấn luyện mô hình:

In [162…

```
# Huấn luyện mô hình đơn biến cho từng đặc trưng
results = {}

for col in features:
    # Tạo ma trận X: [1, giá trị đặc trưng]
    X = [[1, val] for val in df[col].tolist()]

    # Huấn luyện bằng OLS
    beta = train_ols(X, y)  # [[beta0], [beta1]]

    # Dự đoán y_hat
    y_pred = predict(X, beta)

    # Lưu kết quả
    results[col] = {
        "beta": beta,
        "y_pred": y_pred
    }

    print(f"Đặc trưng: {col:8} --> beta0 = {beta[0][0]:.4f}, beta1 = {beta[1][0]:.4f}")
```

```
Đặc trưng: livarea  --> beta0 = -30069.1996, beta1 = 9181.7108
Đặc trưng: beds     --> beta0 = 7392.9485, beta1 = 35400.0313
Đặc trưng: baths    --> beta0 = -11555.8809, beta1 = 63408.2195
Đặc trưng: age      --> beta0 = 137403.5904, beta1 = -627.1610
Đặc trưng: lgelot   --> beta0 = 115220.0192, beta1 = 133797.3492
Đặc trưng: pool     --> beta0 = 119318.6933, beta1 = 66966.7047
```

Phương trình hồi quy:

In [163…

```python
for col in features:
    X = [[1, val] for val in df[col].tolist()]
    beta = train_ols(X, y)

    beta0, beta1 = beta[0][0], beta[1][0]
    print(f"Mô hình hồi quy với {col}:  ŷ = {beta0:.2f} + {beta1:.2f} * {col}")
```

```
Mô hình hồi quy với livarea:  ŷ = -30069.20 + 9181.71 * livarea
Mô hình hồi quy với beds:  ŷ = 7392.95 + 35400.03 * beds
Mô hình hồi quy với baths:  ŷ = -11555.88 + 63408.22 * baths
Mô hình hồi quy với age:  ŷ = 137403.59 + -627.16 * age
Mô hình hồi quy với lgelot:  ŷ = 115220.02 + 133797.35 * lgelot
Mô hình hồi quy với pool:  ŷ = 119318.69 + 66966.70 * pool
```

Bước 3: Đánh giá mô hình bằng chuẩn L2 của vector phần dư

Hàm `residual_norm_loss(y_true, X, beta)` được sử dụng để tính **độ dài (chuẩn L2)** của vector phần dư theo công thức:

$$\|y - \hat{y}\| = \sqrt{\sum (y_i - \hat{y}_i)^2}$$

Trong đó:

- y là vector giá trị thực tế.
- $\hat{y}$ là vector giá trị dự đoán, được tính bằng hàm `predict(X, beta)`.
- Vector phần dư là $y - \hat{y}$.

Chuẩn L2 này phản ánh **mức độ sai lệch tổng thể** giữa dự đoán và thực tế:

- Chuẩn càng nhỏ → mô hình dự đoán càng chính xác.
- Chuẩn càng lớn → sai số tổng thể càng cao.

In [164…

```python
def residual_norm_loss(y_true, X, beta):
    y_pred = predict(X, beta)
    residuals = [(yt - yp) for yt, yp in zip(y_true, y_pred)]
    norm = (sum(r ** 2 for r in residuals)) ** 0.5
    return norm
```

In [165…

```python
best_feature = None
best_norm = float('inf')
worst_feature = None
worst_norm = -float('inf')

for col in features:
    # Tạo ma trận X: [1, giá trị đặc trưng]
    X = [[1, val] for val in df[col].tolist()]

    beta = results[col]["beta"]
    norm = residual_norm_loss(y, X, beta)

    results[col]["residual_norm"] = norm
    print(f"Đặc trưng: {col:8} --> ||residual|| = {norm:.4f}")

    if norm < best_norm:
        best_norm = norm
        best_feature = col

    if norm > worst_norm:
        worst_norm = norm
        worst_feature = col

print(f"\nĐặc trưng tốt nhất (chuẩn nhỏ nhất): {best_feature} với ||residual|| = {best_norm:.4f}")
print(f"Đặc trưng tệ nhất (chuẩn lớn nhất): {worst_feature} với ||residual|| = {worst_norm:.4f}")
```

```
Đặc trưng: livarea  --> ||residual|| = 1492302.8600
Đặc trưng: beds     --> ||residual|| = 2296811.7452
Đặc trưng: baths    --> ||residual|| = 2081735.4287
Đặc trưng: age      --> ||residual|| = 2428086.4662
Đặc trưng: lgelot   --> ||residual|| = 2098584.3425
Đặc trưng: pool     --> ||residual|| = 2363522.8042

Đặc trưng tốt nhất (chuẩn nhỏ nhất): livarea với ||residual|| = 1492302.8600
Đặc trưng tệ nhất (chuẩn lớn nhất): age với ||residual|| = 2428086.4662
```

Bước 4: Trực quan hóa kết quả

Trong bước này, chúng ta tiến hành **đánh giá trực quan hiệu quả dự đoán của mô hình** bằng cách so sánh giữa **giá trị thực tế** `y` và **giá trị dự đoán** $\hat{y}$ thông qua biểu đồ phân tán.

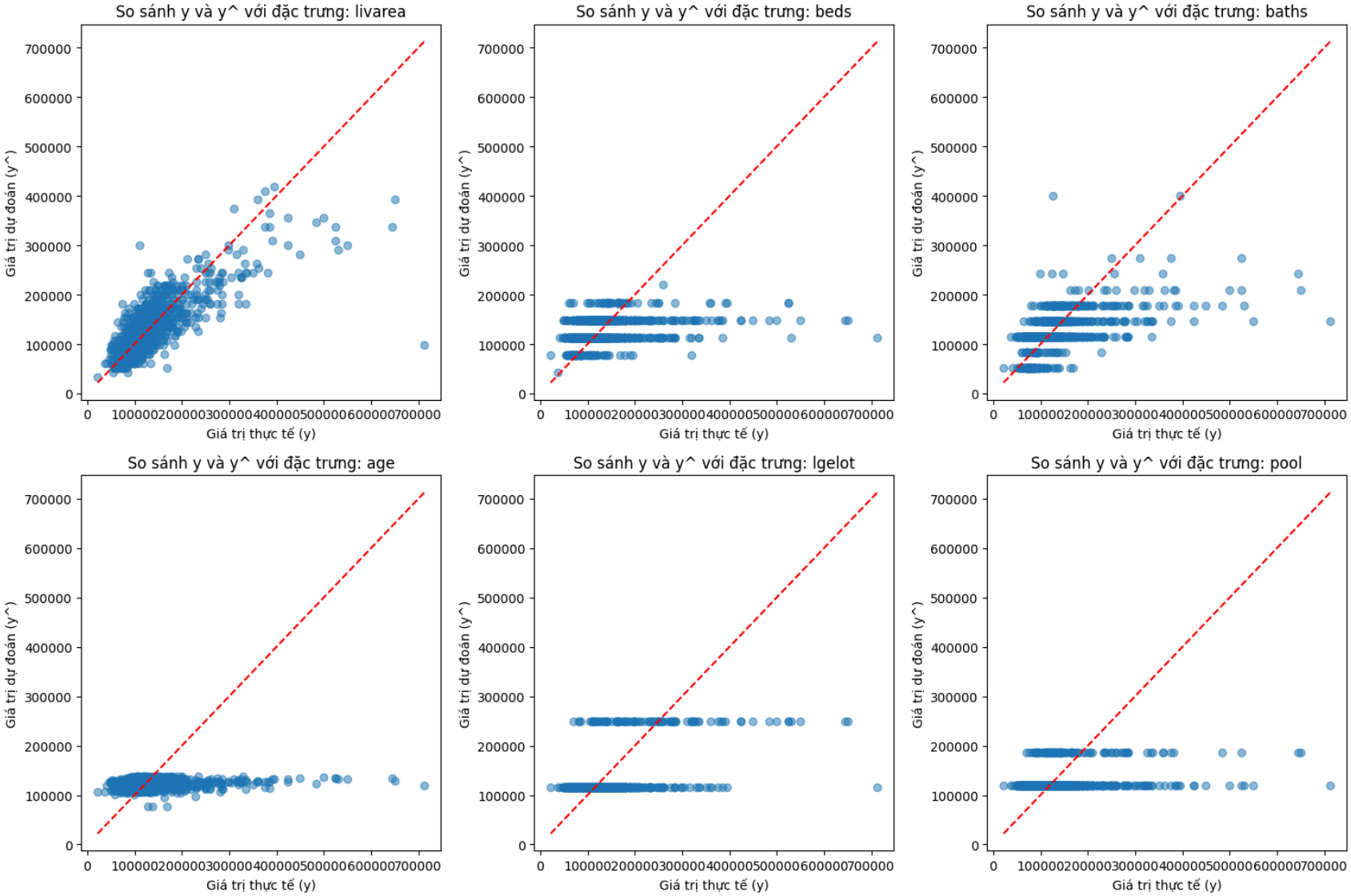
- Xem xét mức độ chính xác của mô hình khi dự đoán `y` từ từng đặc trưng riêng biệt.
- Đánh giá trực quan bằng cách so sánh khoảng cách giữa các điểm dữ liệu và đường chéo $y = \hat{y}$.

Giải thích biểu đồ:

- Mỗi biểu đồ biểu diễn mối quan hệ giữa `y` và $\hat{y}$ khi chỉ dùng **một đặc trưng** trong mô hình tuyến tính đơn.
- **Trục hoành (x-axis)**: Giá trị thực tế `y` trong tập dữ liệu.
- **Trục tung (y-axis)**: Giá trị dự đoán $\hat{y}$ từ mô hình tuyến tính với 1 đặc trưng tương ứng.
- **Đường chéo đỏ ($y = \hat{y}$)**: Thể hiện kết quả dự đoán lý tưởng – nếu mô hình dự đoán hoàn hảo, tất cả điểm sẽ nằm đúng trên đường này.
- **Các điểm gần đường chéo**: Dự đoán chính xác.
- **Các điểm lệch xa đường chéo**: Dự đoán kém chính xác.

In [166…

```python
import matplotlib.pyplot as plt

# Vẽ biểu đồ phân tán y vs y^ cho từng đặc trưng
plt.figure(figsize=(15, 10))

for i, col in enumerate(features, 1):
    # Lấy dữ liệu dự đoán
    X = [[1, val] for val in df[col].tolist()]
    beta = results[col]["beta"]
    y_pred = predict(X, beta)

    # Vẽ biểu đồ
    plt.subplot(2, 3, i)
    plt.scatter(y, y_pred, alpha=0.5)
    plt.plot([min(y), max(y)], [min(y), max(y)], color="red", linestyle="--")  # đường chéo y=x
    plt.xlabel("Giá trị thực tế (y)")
    plt.ylabel("Giá trị dự đoán (y^)")
    plt.title(f"So sánh y và y^ với đặc trưng: {col}")

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



Câu c

Xây dựng mô hình hồi quy tuyến tính **đa biến**

- **Biến phụ thuộc**: `sprice` (giá bán nhà).

- **Biến độc lập**: Tất cả các đặc trưng còn lại: `livarea`, `beds`, `baths`, `age`, `lgelot`, `pool`.

Mô hình hồi quy tuyến tính đa biến được biểu diễn dưới dạng:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

trong đó:

- y là biến phụ thuộc (ở đây là `sprice`),
- $x_1, x_2, \ldots, x_p$ là các biến độc lập (`livarea`, `beds`, `baths`, `age`, `lgelot`, `pool`),
- β_0 là hệ số chặn (intercept),
- $\beta_1, \beta_2, \ldots, \beta_p$ là các hệ số hồi quy,

Các bước thực hiện

Bước 1: Chuẩn bị dữ liệu

- Chọn biến phụ thuộc: `y = sprice`.
- Tạo ma trận đặc trưng X gồm:
  - Một cột giá trị hằng số (bias/intercept).
  - Các cột đặc trưng độc lập: `livarea`, `beds`, `baths`, `age`, `lgelot`, `pool`.

Bước 2: Huấn luyện mô hình bằng OLS

- Áp dụng công thức hồi quy tuyến tính:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- Kết quả là một vector hệ số β tương ứng với từng đặc trưng, cùng với hệ số chặn (intercept).

Bước 3: Tính giá trị dự đoán và đánh giá mô hình bằng chuẩn L2

- Tính giá trị dự đoán từ mô hình:

$$\hat{y} = X\hat{\beta}$$

- Tính **độ lớn vector phần dư** (chuẩn L2), dùng để đánh giá sai số tổng thể của mô hình:

$$\|y - \hat{y}\| = \sqrt{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

- Ý nghĩa:
  - Chuẩn L2 càng nhỏ → mô hình dự đoán càng chính xác.
  - Chuẩn L2 càng lớn → sai số tổng thể càng cao.

Bước 4: Trực quan hóa kết quả dự đoán

- Vẽ biểu đồ phân tán giữa `y` (giá trị thực tế) và $\hat{y}$ (giá trị dự đoán từ mô hình đa biến).
- Thêm đường chéo $y = x$ để đối chiếu.
- Nếu các điểm nằm gần đường chéo, mô hình có độ chính xác cao.

Bước 1: Chuẩn bị dữ liệu

In [167…

```
attributes = df.drop(columns='sprice').values.tolist()
y = df['sprice'].values.tolist()
X_multi = [[1.0] + row for row in attributes]
```

Bước 2: Huấn luyện mô hình bằng OLS

- Sử dụng các hàm xử lý ma trận ở **câu b**

In [ ]:

```
beta_multi_list = train_ols(X_multi, y)
beta_multi = [b[0] if isinstance(b, list) else b for b in beta_multi_list]

print("Vector hệ số beta (gồm intercept và các hệ số):")
for i, b in enumerate(beta_multi):
    print(f"beta_{i} = {b:.4f}")

# In mô hình hồi quy dưới dạng phương trình
features = ['Intercept', 'livarea', 'beds', 'baths', 'age', 'lgelot', 'pool']

model_eq = "sprice = "
for i, b in enumerate(beta_multi):
    sign = " + "
```

```
    if b >= 0 and i > 0:
        model_eq += f"{sign}{b:.4f}*{features[i]}" if i > 0 else f"{b:.4f}"
    else:
        model_eq += f"{sign}({b:.4f})*{features[i]}" if i > 0 else f"{b:.4f}"

print("\nPhương trình hồi quy đa biến:")
print(model_eq)
```

```
Vector hệ số beta (gồm intercept và các hệ số):
beta_0 = 16031.1133
beta_1 = 8884.4814
beta_2 = -10571.8634
beta_3 = -3539.3181
beta_4 = 59598.0218
beta_5 = -162.9355
beta_6 = 14479.2586

Phương trình hồi quy đa biến:
y = 16031.1133 + 8884.4814*livarea + (-10571.8634)*beds + (-3539.3181)*baths + 59598.0218*age + (-162.9355)*lgelot +
14479.2586*pool
```

Bước 3: Tính giá trị dự đoán và đánh giá mô hình

In [169…

```
y_hat = predict(X_multi, beta_multi_list)

# Tính độ lớn vector phần dư (chuẩn L2)
residual_norm = 0
for y_true, y_pred in zip(y, y_hat):
    residual_norm += (y_true - y_pred) ** 2

residual_norm = residual_norm ** 0.5

print(f"Độ lớn vector phần dư (chuẩn L2): {residual_norm:.4f}")
```

```
Độ lớn vector phần dư (chuẩn L2): 1353756.9712
```

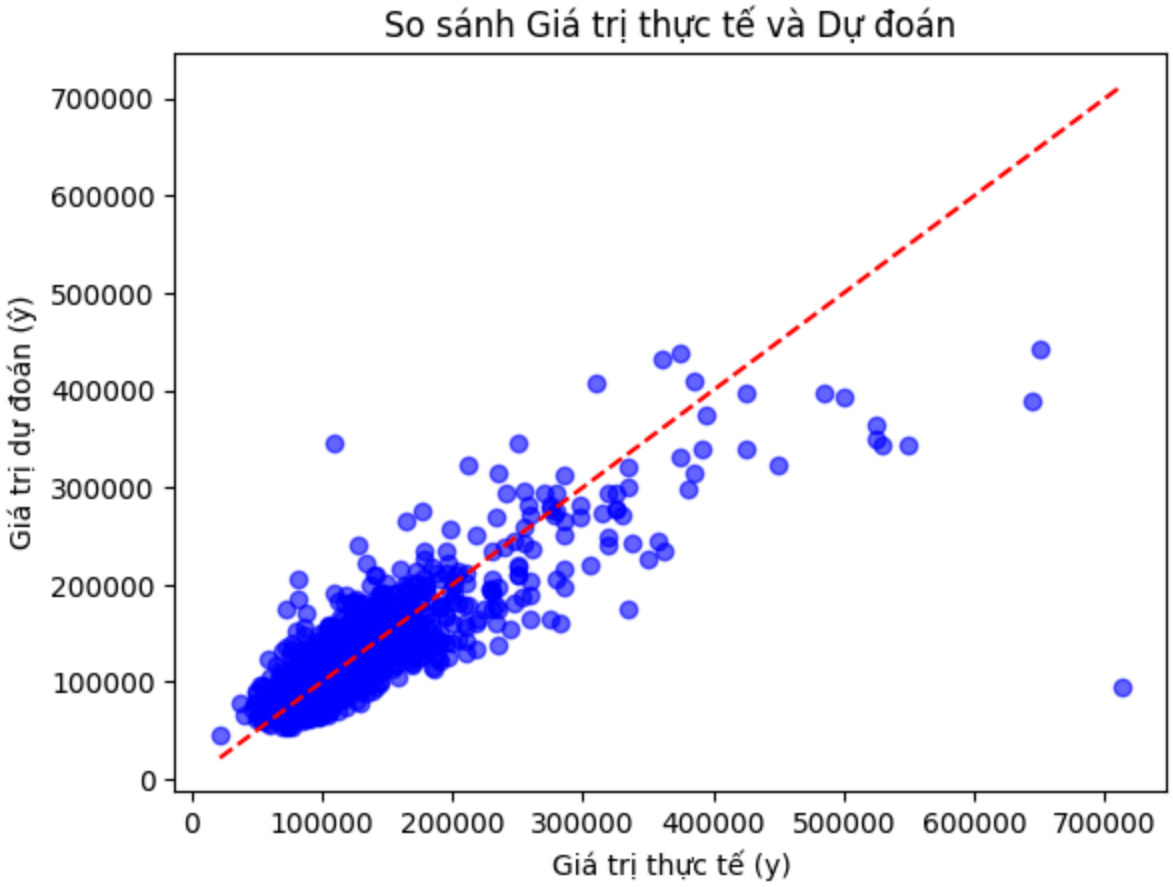
Bước 4: Trực quan hóa kết quả dự đoán

- Vẽ biểu đồ phân tán giữa giá trị thực tế `y` (giá bán nhà) và giá trị dự đoán $\hat{y}$ từ mô hình hồi quy đa biến.
- Trục hoành (x-axis) là giá trị thực tế `y`.

- Trục tung (y-axis) là giá trị dự đoán $\hat{y}$.
- Thêm đường chéo $y = x$ (màu đỏ, nét đứt) làm tham chiếu, thể hiện trường hợp dự đoán hoàn toàn chính xác.
- Nếu các điểm dữ liệu nằm gần đường chéo, chứng tỏ mô hình dự đoán tốt và có độ chính xác cao.
- Nếu các điểm phân tán xa đường chéo, mô hình có độ chính xác kém.

In [170…

```python
import matplotlib.pyplot as plt

# Vẽ biểu đồ phân tán giữa y (giá thực tế) và y_hat (giá dự đoán)
plt.scatter(y, y_hat, color='blue', alpha=0.6)
plt.xlabel('Giá trị thực tế (y)')
plt.ylabel('Giá trị dự đoán (ŷ)')
plt.title('So sánh Giá trị thực tế và Dự đoán')

# Vẽ đường chéo y = x để đối chiếu
min_val = min(min(y), min(y_hat))
max_val = max(max(y), max(y_hat))
plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--')

plt.show()
```



Nhận xét

- Dự đoán từ mô hình đa biến có chuẩn L2 nhỏ hơn nhiều so với các mô hình đơn biến, chứng tỏ sai số dự đoán giảm đáng kể.
- Biểu đồ so sánh giữa giá trị thực tế và giá trị dự đoán thể hiện các điểm dữ liệu tập trung gần đường y = x hơn, cho thấy mô hình đa biến có độ chính xác và khả năng dự đoán cao hơn.

Câu d

Xây dựng mô hình hồi quy tuyến tính đa biến với biến gốc và tất cả các biến tương tác

- **Biến phụ thuộc**: `sprice` (giá bán nhà).
- **Biến độc lập**:
  - Các biến gốc: `livarea`, `beds`, `baths`, `age`, `lgelot`, `pool`.
  - Tất cả các biến tương tác có thể tạo ra từ biến gốc (tổ hợp 2 biến trở lên).

Mô hình hồi quy tuyến tính đa biến được biểu diễn dưới dạng:

$$y = \beta_0 + \sum_{i=1}^{p} \beta_i x_i + \sum_{j=1}^{q} \beta_j z_j$$

trong đó:

- y là biến phụ thuộc (`sprice`),
- x_i là các biến gốc,
- z_j là các biến tương tác được tạo ra từ tích các biến gốc,
- β_0 là hệ số chặn,
- β_i, β_j là hệ số hồi quy tương ứng.

Các bước thực hiện

Bước 1: Chuẩn bị dữ liệu

- Tạo biến tương tác cho mọi tổ hợp biến gốc từ 2 biến trở lên bằng cách nhân từng cặp, bộ 3,... các biến lại với nhau.
- Tập hợp đặc trưng cuối cùng gồm: các biến gốc + các biến tương tác.

Bước 2: Tạo ma trận đặc trưng và vector mục tiêu

- Ma trận X gồm:
  - Một cột intercept (hằng số 1).
  - Các biến gốc: `livarea`, `beds`, `baths`, `age`, `lgelot`, `pool`.
  - Tất cả các biến tương tác giữa các biến gốc (tổ hợp từ 2 biến trở lên).
- Vector mục tiêu y là giá trị `sprice` (giá bán nhà).

Bước 3: Huấn luyện mô hình

- Sử dụng phương pháp bình phương nhỏ nhất OLS để tìm hệ số $\hat{\beta}$:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- In hệ số hồi quy và phương trình hồi quy để hiểu và diễn giải mô hình. **Bước 4: Đánh giá mô hình**
- Tính giá trị dự đoán từ mô hình:

$$\hat{y} = X\hat{\beta}$$

- Tính **chuẩn L2** của vector phần dư (độ lớn vector sai số) để đánh giá độ chính xác của mô hình:

$$\|y - \hat{y}\| = \sqrt{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

- Chuẩn L2 càng nhỏ → mô hình dự đoán càng chính xác.

Bước 5: Trực quan hóa

- Vẽ biểu đồ phân tán giữa giá trị thực tế `sprice` và giá trị dự đoán $\hat{y}$.
- Thêm đường chéo $y = x$ để đánh giá trực quan độ chính xác mô hình.
- Nếu các điểm gần đường chéo, mô hình dự đoán tốt.

Bước 1: Chuẩn bị dữ liệu

`itertools.combinations` là một hàm trong thư viện `itertools` của Python, được dùng để tạo ra tất cả các tổ hợp (combinations) không lặp lại của các phần tử trong một tập hợp với kích thước cho trước.

Hàm `combinations` trong thư viện này được dùng để sinh ra tất cả các tổ hợp không lặp lại của các phần tử trong một tập hợp với kích thước cố định.
Ví dụ:
`combinations(['a', 'b', 'c'], 2)` sẽ tạo ra các tổ hợp 2 phần tử như `('a', 'b')`, `('a', 'c')`, `('b', 'c')`.

In [171…

```
from itertools import combinations
# Tạo biến tương tác từ các biến gốc
```

```
features = ['livarea', 'beds', 'baths', 'age', 'lgelot', 'pool']

interaction_features = []
for r in range(2, len(features) + 1):
    combs = list(combinations(features, r))
    for comb in combs:
        new_col = "_x_".join(comb)
        df[new_col] = 1
        for c in comb:
            df[new_col] = df[new_col] * df[c]
        interaction_features.append(new_col)

# Tập hợp đặc trưng cuối cùng: biến gốc + biến tương tác
features_new = features + interaction_features
```

Bước 2: Tạo ma trận đặc trưng (X) và vector mục tiêu (y)

In [172…

```
X = [[1] + [df[col].iloc[i] for col in features_new] for i in range(len(df))]
y = df['sprice'].tolist()
```

Bước 3: Huấn luyện mô hình

In [173…

```
beta_new = train_ols(X, y)


print("\nHệ số hồi quy với biến gốc và biến tương tác:")
for i, col in enumerate(['Intercept'] + features_new):
    print(f"{col:40}: {beta_new[i][0]:.4f}")


equation = f"sprice = {beta_new[0][0]:.4f}"  # Intercept
for i, col in enumerate(features_new, start=1):
    equation += f" + ({beta_new[i][0]:.4f})*{col}"
print("\nPhương trình hồi quy:")
print(equation)
```

Hệ số hồi quy với biến gốc và biến tương tác:

Intercept	-30291.1993
livarea	899.0028
beds	31137.5685
baths	4906.4691
age	-1638.3184
lgelot	446250.1726
pool	-3908516.6488
livarea_x_beds	718.6295
livarea_x_baths	3796.9741
livarea_x_age	216.5736
livarea_x_lgelot	61173.1678
livarea_x_pool	176370.3415
beds_x_baths	-12544.2559
beds_x_age	110.3480
beds_x_lgelot	-313312.1970
beds_x_pool	1188321.0579
baths_x_age	2699.8735
baths_x_lgelot	-436675.7159
baths_x_pool	1719662.7271
age_x_lgelot	-3777.6821
age_x_pool	156511.3034
lgelot_x_pool	351112.7215
livarea_x_beds_x_baths	-539.8437
livarea_x_beds_x_age	-42.6188
livarea_x_beds_x_lgelot	-9338.6248
livarea_x_beds_x_pool	-54068.2281
livarea_x_baths_x_age	-202.4565
livarea_x_baths_x_lgelot	-12630.5696
livarea_x_baths_x_pool	-76939.9070
livarea_x_age_x_lgelot	-1752.3866
livarea_x_age_x_pool	-6538.4314
livarea_x_lgelot_x_pool	-108517.7665
beds_x_baths_x_age	-677.0011
beds_x_baths_x_lgelot	197356.6937
beds_x_baths_x_pool	-520527.0025
beds_x_age_x_lgelot	5625.8253
beds_x_age_x_pool	-52265.7597
beds_x_lgelot_x_pool	407475.5830
baths_x_age_x_lgelot	4470.4691
baths_x_age_x_pool	-65004.4833
baths_x_lgelot_x_pool	-481321.7643

```
age_x_lgelot_x_pool                     : 95995.5554
livarea_x_beds_x_baths_x_age            : 51.4433
livarea_x_beds_x_baths_x_lgelot         : 930.7935
livarea_x_beds_x_baths_x_pool           : 23411.0366
livarea_x_beds_x_age_x_lgelot           : 287.4656
livarea_x_beds_x_age_x_pool             : 2296.3400
livarea_x_beds_x_lgelot_x_pool          : 15005.1536
livarea_x_baths_x_age_x_lgelot          : 556.7094
livarea_x_baths_x_age_x_pool            : 2659.0623
livarea_x_baths_x_lgelot_x_pool         : 60140.8282
livarea_x_age_x_lgelot_x_pool           : -1490.3759
beds_x_baths_x_age_x_lgelot             : -2654.8685
beds_x_baths_x_age_x_pool               : 21688.5848
beds_x_baths_x_lgelot_x_pool            : -52620.2056
beds_x_age_x_lgelot_x_pool              : -37235.7159
baths_x_age_x_lgelot_x_pool             : -16408.8513
livarea_x_beds_x_baths_x_age_x_lgelot   : -99.3031
livarea_x_beds_x_baths_x_age_x_pool     : -930.7872
livarea_x_beds_x_baths_x_lgelot_x_pool  : -11129.5942
livarea_x_beds_x_age_x_lgelot_x_pool    : 581.6176
livarea_x_baths_x_age_x_lgelot_x_pool   : -263.4558
beds_x_baths_x_age_x_lgelot_x_pool      : 7623.8436
livarea_x_beds_x_baths_x_age_x_lgelot_x_pool: 45.3274

Phương trình hồi quy:
sprice = -30291.1993 + (899.0028)*livarea + (31137.5685)*beds + (4906.4691)*baths + (-1638.3184)*age + (446250.1726)*
lgelot + (-3908516.6488)*pool + (718.6295)*livarea_x_beds + (3796.9741)*livarea_x_baths + (216.5736)*livarea_x_age +
(61173.1678)*livarea_x_lgelot + (176370.3415)*livarea_x_pool + (-12544.2559)*beds_x_baths + (110.3480)*beds_x_age +
(-313312.1970)*beds_x_lgelot + (1188321.0579)*beds_x_pool + (2699.8735)*baths_x_age + (-436675.7159)*baths_x_lgelot +
(1719662.7271)*baths_x_pool + (-3777.6821)*age_x_lgelot + (156511.3034)*age_x_pool + (351112.7215)*lgelot_x_pool + (-
539.8437)*livarea_x_beds_x_baths + (-42.6188)*livarea_x_beds_x_age + (-9338.6248)*livarea_x_beds_x_lgelot + (-54068.2
281)*livarea_x_beds_x_pool + (-202.4565)*livarea_x_baths_x_age + (-12630.5696)*livarea_x_baths_x_lgelot + (-76939.907
0)*livarea_x_baths_x_pool + (-1752.3866)*livarea_x_age_x_lgelot + (-6538.4314)*livarea_x_age_x_pool + (-108517.7665)*
livarea_x_lgelot_x_pool + (-677.0011)*beds_x_baths_x_age + (197356.6937)*beds_x_baths_x_lgelot + (-520527.0025)*beds_
x_baths_x_pool + (5625.8253)*beds_x_age_x_lgelot + (-52265.7597)*beds_x_age_x_pool + (407475.5830)*beds_x_lgelot_x_po
ol + (4470.4691)*baths_x_age_x_lgelot + (-65004.4833)*baths_x_age_x_pool + (-481321.7643)*baths_x_lgelot_x_pool + (95
995.5554)*age_x_lgelot_x_pool + (51.4433)*livarea_x_beds_x_baths_x_age + (930.7935)*livarea_x_beds_x_baths_x_lgelot +
(23411.0366)*livarea_x_beds_x_baths_x_pool + (287.4656)*livarea_x_beds_x_age_x_lgelot + (2296.3400)*livarea_x_beds_x_
age_x_pool + (15005.1536)*livarea_x_beds_x_lgelot_x_pool + (556.7094)*livarea_x_baths_x_age_x_lgelot + (2659.0623)*li
varea_x_baths_x_age_x_pool + (60140.8282)*livarea_x_baths_x_lgelot_x_pool + (-1490.3759)*livarea_x_age_x_lgelot_x_poo
l + (-2654.8685)*beds_x_baths_x_age_x_lgelot + (21688.5848)*beds_x_baths_x_age_x_pool + (-52620.2056)*beds_x_baths_x_
lgelot_x_pool + (-37235.7159)*beds_x_age_x_lgelot_x_pool + (-16408.8513)*baths_x_age_x_lgelot_x_pool + (-99.3031)*liv
```

```
area_x_beds_x_baths_x_age_x_lgelot + (-930.7872)*livarea_x_beds_x_baths_x_age_x_pool + (-11129.5942)*livarea_x_beds_x
_baths_x_lgelot_x_pool + (581.6176)*livarea_x_beds_x_age_x_lgelot_x_pool + (-263.4558)*livarea_x_baths_x_age_x_lgelot
_x_pool + (7623.8436)*beds_x_baths_x_age_x_lgelot_x_pool + (45.3274)*livarea_x_beds_x_baths_x_age_x_lgelot_x_pool
```

Bước 4: Tính độ lớn vector phần dư

In [174…

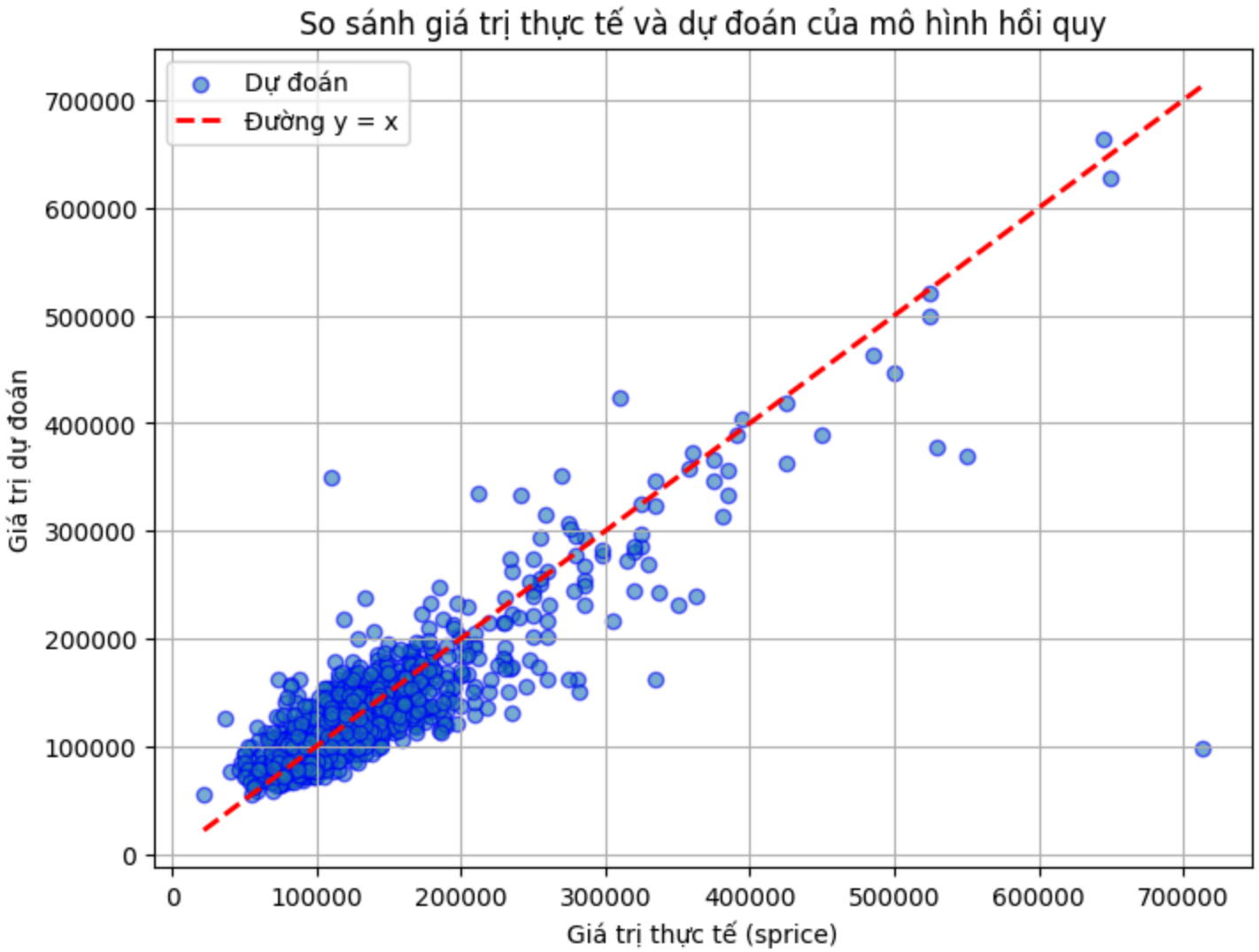
```python
residual_norm_c = residual_norm_loss(y, X_multi, beta_multi_list)  # X_multi, beta_multi_list từ câu c

# Chuẩn L2 của mô hình thêm biến tương tác
residual_norm_new = residual_norm_loss(y, X, beta_new)

# Tính phần trăm giảm chuẩn L2
reduction_percent = ((residual_norm_c - residual_norm_new) / residual_norm_c) * 100

print(f"Chuẩn L2 câu c (chỉ biến gốc): {residual_norm_c:.4f}")
print(f"Chuẩn L2 sau khi thêm biến tương tác: {residual_norm_new:.4f}")
print(f"Giảm chuẩn L2: {reduction_percent:.2f}% so với mô hình câu c")
```

```
Chuẩn L2 câu c (chỉ biến gốc): 1353756.9712
Chuẩn L2 sau khi thêm biến tương tác: 1209242.5834
Giảm chuẩn L2: 10.68% so với mô hình câu c
```

Bước 5: Trực quan hóa

In [175…

```python
y_pred = [sum(X[i][j] * beta_new[j][0] for j in range(len(beta_new))) for i in range(len(X))]

plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, alpha=0.6, edgecolors='b', label='Dự đoán')
plt.plot([min(y), max(y)], [min(y), max(y)], 'r--', lw=2, label='Đường y = x')

plt.xlabel('Giá trị thực tế (sprice)')
plt.ylabel('Giá trị dự đoán')
plt.title('So sánh giá trị thực tế và dự đoán của mô hình hồi quy')
plt.legend()
plt.grid(True)
plt.show()
```



Lý do thêm các biến tương tác vào mô hình

Việc bổ sung các biến tương tác giúp mô hình không chỉ học được ảnh hưởng riêng lẻ của từng biến mà còn nắm bắt được cách các biến kết hợp tác động đến biến mục tiêu.

Lý do chọn cách này là vì trong thực tế, các yếu tố không luôn tác động độc lập mà thường có sự tương tác phức tạp. Ví dụ, diện tích nhà (`livarea`) có thể ảnh hưởng khác khi kết hợp với số phòng ngủ (`beds`) hoặc có hồ bơi (`pool`).

Do đó, thêm biến tương tác giúp cải thiện khả năng dự đoán và độ chính xác của mô hình bằng cách mở rộng không gian đặc trưng.

Bài 2

Câu a

Dữ liệu trạng thái giao thông 60 ngày

- Dữ liệu mô phỏng trạng thái giao thông trong 60 ngày được lưu dưới dạng danh sách `traffic_data`.
- Mỗi phần tử trong danh sách biểu diễn trạng thái giao thông của một ngày với các giá trị có thể là:
  - `K`: Kẹt xe
  - `B`: Bình thường
  - `T`: Thoáng
- Các trạng thái này thể hiện tình trạng giao thông theo từng ngày.
- Danh sách `states` chứa tất cả các trạng thái có thể xuất hiện.

In [176…

```
# Dữ liệu trạng thái đã cho (60 ngày)
traffic_data = ['T', 'B', 'T', 'T', 'K', 'K', 'K', 'K', 'B', 'K', 'K', 'T', 'B', 'B', 'T', 'T', 'K', 'B', 'K', 'T',
                'K', 'B', 'K', 'B', 'K', 'K', 'B', 'B', 'B', 'K', 'K', 'K', 'T', 'K', 'B', 'B', 'B', 'K', 'B', 'B',
                'T', 'T', 'B', 'T', 'T', 'B', 'K', 'K', 'B', 'K', 'T', 'B', 'K', 'K', 'T', 'B', 'B', 'T', 'B', 'B']
states = ['K', 'B', 'T']
print("Dữ liệu giao thông 60 ngày:")
print(traffic_data)
```

```
Dữ liệu giao thông 60 ngày:
['T', 'B', 'T', 'T', 'K', 'K', 'K', 'K', 'B', 'K', 'K', 'T', 'B', 'B', 'T', 'T', 'K', 'B', 'K', 'T', 'K', 'B', 'K',
'B', 'K', 'K', 'B', 'B', 'B', 'K', 'K', 'K', 'T', 'K', 'B', 'B', 'B', 'K', 'B', 'B', 'T', 'T', 'B', 'T', 'T', 'B',
'K', 'K', 'B', 'K', 'T', 'B', 'K', 'K', 'T', 'B', 'B', 'T', 'B', 'B']
```

Câu b

Xây dựng ma trận đếm số lần chuyển trạng thái

- Từ dữ liệu trạng thái giao thông trong 60 ngày, xây dựng ma trận đếm số lần chuyển trạng thái giữa các trạng thái.
- Ma trận có kích thước 3×3 với 3 là số trạng thái (`K` , `B` , `T`).
- Hàng thứ i và cột thứ j của ma trận biểu diễn số lần chuyển từ trạng thái j (xuất phát) sang trạng thái i (đến).
- Ví dụ: Giá trị tại hàng `K` , cột `B` là số lần trạng thái chuyển từ `B` sang `K` .

In [177…

```python
def build_count_matrix(data, states=['K','B','T']):
    n = len(states)
    count_matrix = [[0]*n for _ in range(n)]
    state_to_index = {s:i for i,s in enumerate(states)}

    for idx in range(len(data)-1):
        start = state_to_index[data[idx]]     # trạng thái xuất phát (j)
        end = state_to_index[data[idx+1]]     # trạng thái đến (i)
        count_matrix[end][start] += 1         # tăng count cho hàng i, cột j

    return count_matrix

count_matrix = build_count_matrix(traffic_data, states)

print("Ma trận đếm chuyển trạng thái (i=đến, j=xuất phát):")
print("    K  B  T (xuất phát j)")
for i, row in enumerate(count_matrix):
    print(f"{states[i]}: {row} (đến i)")
```

```
Ma trận đếm chuyển trạng thái (i=đến, j=xuất phát):
    K  B  T (xuất phát j)
K: [9, 9, 4] (đến i)
B: [8, 8, 7] (đến i)
T: [5, 5, 4] (đến i)
```

Câu c

Tính ma trận xác suất chuyển trạng thái

- Dựa trên ma trận đếm chuyển trạng thái, tính ma trận xác suất chuyển trạng thái P.

- Ma trận P có kích thước 3×3 với 3 là số trạng thái.
- Mỗi phần tử P_{ij} là xác suất chuyển từ trạng thái j (xuất phát) sang trạng thái i (đến).
- Cách tính: chia số lần chuyển từ trạng thái j sang i cho tổng số lần xuất phát từ trạng thái j.

In [178…

```python
def build_transition_matrix_col_norm(count_matrix):
    n = len(count_matrix)
    col_sums = [0]*n
    for j in range(n):
        s = 0
        for i in range(n):
            s += count_matrix[i][j]
        col_sums[j] = s

    P = []
    for i in range(n):
        row = []
        for j in range(n):
            if col_sums[j] == 0:
                row.append(1/n)  # giả sử đều nếu cột rỗng
            else:
                row.append(count_matrix[i][j]/col_sums[j])
        P.append(row)
    return P

P = build_transition_matrix_col_norm(count_matrix)

print("\nMa trận xác suất chuyển trạng thái (chuẩn hóa tổng cột = 1):")
print("      K      B      T    (xuất phát j)")
for i, row in enumerate(P):
    print(f"{states[i]}: {[round(x,4) for x in row]} (đến i)")
```

```
Ma trận xác suất chuyển trạng thái (chuẩn hóa tổng cột = 1):
      K      B      T    (xuất phát j)
K: [0.4091, 0.4091, 0.2667] (đến i)
B: [0.3636, 0.3636, 0.4667] (đến i)
T: [0.2273, 0.2273, 0.2667] (đến i)
```

Câu d

Dự đoán phân phối trạng thái giao thông cho các ngày tiếp theo

- Hàm `prop_distribution` : tính phân phối xác suất trạng thái giao thông vào ngày thứ k tiếp theo.
- Giả định phân phối ban đầu (ngày 0) là đồng đều, tức mỗi trạng thái có xác suất bằng nhau (1/3).
- Sử dụng phép nhân ma trận với vector phân phối để tính phân phối ngày tiếp theo:

$$\pi(k) = P^k \cdot \pi(0)$$

- Hàm `multiply_matrix_vector` thực hiện phép nhân ma trận P với vector phân phối.
- Lặp lại phép nhân này k lần để có phân phối ngày k.

In [179…

```python
def multiply_matrix_vector(P, v):
    n = len(P)
    result = [0]*n
    for i in range(n):
        s = 0
        for j in range(n):
            s += P[i][j]*v[j]
        result[i] = s
    return result

# Hàm tính phân phối ngày k: pi(k) = P^k * pi(0)
def prop_distribution(P, k, initial_distribution=None):
    if initial_distribution is None:
        initial_distribution = [1/3]*len(P)  # vector cột đều ban đầu

    dist = initial_distribution[:]
    for _ in range(k):
        dist = multiply_matrix_vector(P, dist)
    return dist

# Kiểm tra phân phối sau 1, 2, 3, 10 ngày
for day in [1,2,3,10, 100]:
    dist = prop_distribution(P, day)
    print(f"\nPhân phối trạng thái ngày thứ {day}:")
```

```
    for st, prob in zip(states, dist):
        print(f"{st}: {round(prob,4)}")
```

```
Phân phối trạng thái ngày thứ 1:
K: 0.3616
B: 0.398
T: 0.2404

Phân phối trạng thái ngày thứ 2:
K: 0.3749
B: 0.3884
T: 0.2367

Phân phối trạng thái ngày thứ 3:
K: 0.3754
B: 0.388
T: 0.2366

Phân phối trạng thái ngày thứ 10:
K: 0.3754
B: 0.388
T: 0.2366

Phân phối trạng thái ngày thứ 100:
K: 0.3754
B: 0.388
T: 0.2366
```

Câu e

Tìm phân phối dừng của xích Markov

Phân phối dừng π của ma trận chuyển trạng thái P là vector xác suất thỏa mãn:

$$P\pi = \pi$$

Điều này tương đương với hệ phương trình:

$$(P - I)\pi = 0$$

trong đó I là ma trận đơn vị.

Vì π là vector xác suất, các phần tử π_i phải thỏa mãn:

$$\sum_i \pi_i = 1$$

Do đó, để giải hệ, ta thay thế một trong các phương trình của $(P - I)\pi = 0$ bằng phương trình tổng xác suất bằng 1 để hệ có nghiệm duy nhất.

Thuật toán giải hệ phương trình

Ở đây, ta sử dụng phương pháp khử Gauss để giải hệ phương trình tuyến tính:

- Tạo ma trận mở rộng $[A|b]$ với $A = P - I$ và vector b có tất cả giá trị 0 ngoại trừ phần tử cuối cùng là 1 (tương ứng với ràng buộc tổng xác suất).
- Thực hiện khử Gauss để đưa ma trận về dạng tam giác trên.
- Tính nghiệm π từ ma trận đã biến đổi.

Kết quả

Sau khi giải, ta thu được phân phối dừng π, đại diện cho xác suất lâu dài mà hệ thống giao thông sẽ ở mỗi trạng thái.

In [180…

```python
def gauss_solve(A, b):
    """
    Giải hệ Ax = b bằng phương pháp khử Gauss.
    A là ma trận vuông, b là vector cột.
    """
    n = len(A)
    # Tạo ma trận mở rộng [A|b]
    M = [A[i][:] + [b[i]] for i in range(n)]

    for i in range(n):
        # Tìm pivot
        max_row = max(range(i, n), key=lambda r: abs(M[r][i]))
        if abs(M[max_row][i]) < 1e-15:
            raise ValueError("Ma trận suy biến, không thể giải")
```

```
        # Đổi dòng
        M[i], M[max_row] = M[max_row], M[i]

        # Chuẩn hóa pivot
        pivot = M[i][i]
        for j in range(i, n+1):
            M[i][j] /= pivot

        # Khử các dòng khác
        for r in range(n):
            if r != i:
                factor = M[r][i]
                for c in range(i, n+1):
                    M[r][c] -= factor * M[i][c]

    # Trả về nghiệm
    x = [M[i][n] for i in range(n)]
    return x

def solve_steady_state(P):
    n = len(P)
    # Tạo ma trận A = P - I
    A = [[P[i][j] for j in range(n)] for i in range(n)]
    for i in range(n):
        A[i][i] -= 1

    # Thay dòng cuối bằng ràng buộc sum pi_i = 1
    A[-1] = [1]*n
    b = [0]*(n-1) + [1]

    pi = gauss_solve(A, b)
    return pi


steady_dist_gauss = solve_steady_state(P)
print("Phân phối dừng (giải bằng Gauss: (P-I)π = 0):")
for st, prob in zip(states, steady_dist_gauss):
    print(f"{st}: {round(prob,6)}")
```

Phân phối dừng (giải bằng Gauss: (P-I)π = 0):
K: 0.375394
B: 0.388013
T: 0.236593