

BÀI TẬP LỚN MÔN XÁC SUẤT THỐNG KÊ

Chủ đề: Hồi quy tuyến tính

Dựa vào bộ dữ liệu kc_house_data.csv bao gồm các thuộc tính mô tả chất lượng ngôi nhà, dự đoán giá nhà
(Nguồn Kaggle.com, đã lược bỏ một số cột)

Nội dung thực hiện

1. Khảo sát dữ liệu, làm sạch và tiền xử lý
2. Trực quan hóa dữ liệu, phân tích các mối tương quan
3. Tìm ra xem giá nhà phụ thuộc vào các yếu tố nào
4. Dự đoán giá bán theo mô hình hồi quy tuyến tính
5. Kiểm định, đánh giá mô hình hồi quy tuyến tính
6. Khảo sát các mô hình hồi quy khác => cải thiện độ chính xác dự đoán

Nhóm: #2

- 1.Võ Minh Hiếu (Trưởng nhóm)
- 2.Lê Công Phương
- 3.Nguyễn Thị Kim Oanh
- 4.Lê Công Huy
- 5.Huỳnh Ngọc Kha

```
In [1]: # Import Needed Libraries

# Thư viện xử lý dữ liệu
import pandas as pd
import numpy as np

# Thư viện trực quan hóa dữ liệu
import matplotlib.pyplot as plt
import seaborn as sns

# Các thư viện mô hình hồi quy
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import RobustScaler, StandardScaler
from sklearn.impute import SimpleImputer

from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor as KNN

import pickle
import warnings
warnings.filterwarnings('ignore')
```

Bộ dữ liệu gồm các cột dữ liệu sau:

1. id: Một số duy nhất được gán cho mỗi bản ghi (nhà) trong tập dữ liệu.
2. bedrooms: Số lượng phòng ngủ trong căn nhà, dạng dữ liệu là int64.
3. bathrooms: Số lượng phòng tắm trong căn nhà, dạng dữ liệu là float64.
4. sqft_lot: Diện tích đất của căn nhà, dạng dữ liệu là int64 (đơn vị: feet vuông).
5. floors: Số tầng của ngôi nhà được phân loại từ 1-3.5.
6. grade: Đánh giá cấp bậc của căn nhà, dạng dữ liệu là int64.
7. condition: Điều kiện kiến trúc của ngôi nhà từ 1 – 5, 1: rất tệ và 5: rất tốt.
8. view: Đánh giá cảnh quan xung quanh nhà theo mức độ từ thấp đến cao: 0-4.
9. sqft_above: Diện tích ngôi nhà.
10. sqft_living: Diện tích khuôn viên nhà.
11. sqft_basement: Diện tích tầng hầm.

12. price: Giá nhà được bán ra.

Yêu cầu

Sử dụng phương pháp hồi quy để dự đoán giá nhà dựa vào các dữ liệu thuộc tính cột (2-11) Đánh giá mô hình hồi quy tìm được

Theo mô hình hồi quy

- Biến phụ thuộc: price
- Các biến độc lập: các cột 2,3,4,5,6,7,8,9,10,11

Đọc và hiển thị dữ liệu (Import Data)

```
In [2]: # Đọc và hiển thị dữ liệu  
df = pd.read_csv("kc_house_data.csv")  
df.sample(5)
```

```
Out[2]:   id  bedrooms  bathrooms  sqft_living  sqft_lot  floors  view  condition  grade  sqft_above  sqft_basement  price  
0    8536      3879901295        3       2.50      2660     1973      3.0       3       3       9      1870          790  1240000.0  
1    14774      9412400220        4       2.75      5470     18200      2.0       4       3      11      3730          1740  1610000.0  
2    9646      2028701165        2       1.00      1050      2570      1.0       0       5       7      850           200  430000.0  
3    18407      3025049052        2       1.00      1450      7098      1.0       4       3       7      1450            0  822500.0  
4    14438      1545800730        2       1.75      1320      7540      1.0       0       3       7      1320            0  269950.0
```

Thông tin của bộ dữ liệu

```
In [3]: # Xem số hàng và số cột  
print(f"Number of Row : {df.shape[0]}\nNumber of Columns : {df.shape[1]}")
```

Number of Row : 21613
Number of Columns : 12

```
In [4]: # Xem thông tin dữ liệu  
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 21613 entries, 0 to 21612  
Data columns (total 12 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          --          --  
 0   id          21613 non-null   int64    
 1   bedrooms    21613 non-null   int64    
 2   bathrooms   21613 non-null   float64  
 3   sqft_living 21613 non-null   int64    
 4   sqft_lot    21613 non-null   int64    
 5   floors      21613 non-null   float64  
 6   view        21613 non-null   int64    
 7   condition   21613 non-null   int64    
 8   grade       21613 non-null   int64    
 9   sqft_above   21613 non-null   int64    
 10  sqft_basement 21613 non-null   int64    
 11  price       21613 non-null   float64  
dtypes: float64(3), int64(9)  
memory usage: 2.0 MB
```

Tóm tắt thống kê của dữ liệu

```
In [5]: df.describe()
```

```
Out[5]:   id  bedrooms  bathrooms  sqft_living  sqft_lot  floors  view  condition  grade  sq  
count  2.161300e+04  21613.000000  21613.000000  21613.000000  2.161300e+04  21613.000000  21613.000000  21613.000000  21613.000000  2161  
mean  4.580302e+09  3.370842    2.114757    2079.899736  1.510697e+04  1.494309    0.234303    3.409430    7.656873    178  
std   2.876566e+09  0.930062    0.770163    918.440897  4.142051e+04  0.539989    0.766318    0.650743    1.175459    82  
min   1.000102e+06  0.000000    0.000000    290.000000  5.200000e+02  1.000000    0.000000    1.000000    1.000000    29  
25%   2.123049e+09  3.000000    1.750000    1427.000000  5.040000e+03  1.000000    0.000000    3.000000    7.000000    119  
50%   3.904930e+09  3.000000    2.250000    1910.000000  7.618000e+03  1.500000    0.000000    3.000000    7.000000    156  
75%   7.308900e+09  4.000000    2.500000    2550.000000  1.068800e+04  2.000000    0.000000    4.000000    8.000000    221  
max   9.900000e+09  33.000000   8.000000   13540.000000  1.651359e+06  3.500000    4.000000    5.000000   13.000000   941
```

- count: Số lượng các mục không phải là giá trị thiếu trong mỗi cột.

- mean: Trung bình của các giá trị trong mỗi cột.
- std: Độ lệch chuẩn của các giá trị trong mỗi cột, đo lường sự phân tán của dữ liệu.
- min: Giá trị nhỏ nhất trong mỗi cột.
- 25%, 50%, 75%: Phần centile tương ứng, thường được sử dụng để xác định các phân vị trong dữ liệu.
- max: Giá trị lớn nhất trong mỗi cột.

Nhận xét sơ bộ về bộ dữ liệu

- Bộ dữ liệu có 21613 dòng và 12 cột dữ liệu
- Các cột dữ liệu hầu như đều là kiểu số
- Price (biến phụ thuộc) có phạm vi giao động khá lớn từ 75000 đến 7700000

Khảo sát dữ liệu, làm sạch => tiền xử lý

Kiểm tra dữ liệu thiếu

```
In [6]: df.isna().sum()
```

```
Out[6]: id          0
bedrooms      0
bathrooms     0
sqft_living   0
sqft_lot       0
floors        0
view          0
condition     0
grade          0
sqft_above     0
sqft_basement 0
price          0
dtype: int64
```

=> Dữ liệu không có các giá trị thiếu, và các cột đều chứa đủ dữ liệu.

Kiểm tra dữ liệu trùng lặp

```
In [7]: duplicate_rows = df[df.duplicated()]
print("Số lượng hàng trùng lặp:", len(duplicate_rows))
```

Số lượng hàng trùng lặp: 3

```
In [8]: # Xem các hàng trùng lặp
duplicate_rows
```

	id	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	condition	grade	sqft_above	sqft_basement	price
3951	1825069031	4	1.75	2410	8447	2.0	3	4	8	2060	350	550000.0
14983	6308000010	3	2.50	2290	5089	2.0	0	3	9	2290	0	585000.0
20054	8648900110	3	2.50	1940	3211	2.0	0	3	8	1940	0	555000.0

Do số lượng hàng trùng lặp ít nên ta có thể bỏ chúng đi

```
In [9]: df.drop_duplicates(inplace=True)
```

Xem lại thông tin của dữ liệu

```
In [10]: df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Index: 21610 entries, 0 to 21612
Data columns (total 12 columns):
 #   Column      Non-Null Count Dtype  
--- 
 0   id          21610 non-null   int64  
 1   bedrooms    21610 non-null   int64  
 2   bathrooms   21610 non-null   float64 
 3   sqft_living 21610 non-null   int64  
 4   sqft_lot    21610 non-null   int64  
 5   floors      21610 non-null   float64 
 6   view        21610 non-null   int64  
 7   condition   21610 non-null   int64  
 8   grade       21610 non-null   int64  
 9   sqft_above  21610 non-null   int64  
 10  sqft_basement 21610 non-null   int64  
 11  price       21610 non-null   float64 
dtypes: float64(3), int64(9)
memory usage: 2.1 MB

```

In [11]: df.shape

Out[11]: (21610, 12)

Khảo sát biến phụ thuộc (Price)

```

In [12]: # Thiết lập màu sắc
boxplot_color = '#3498db' # Màu xanh dương
histogram_color = '#e74c3c' # Màu đỏ

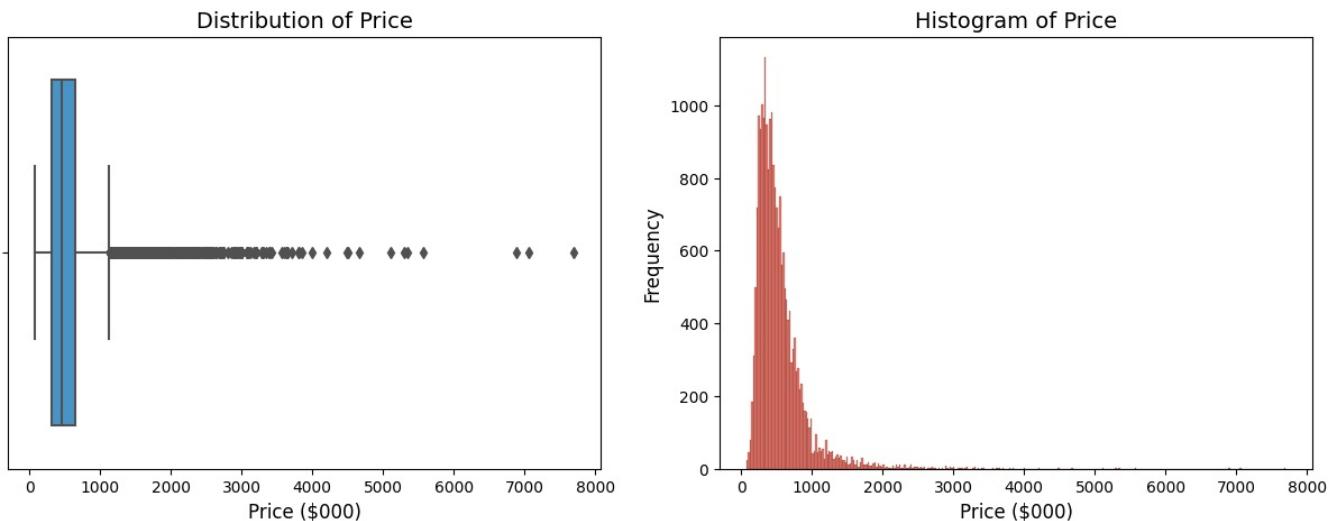
# Tạo figure và axes cho biểu đồ
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Biểu đồ Boxplot
sns.boxplot(data=df, x=df['price'] / 1000, ax=axes[0], color=boxplot_color)
axes[0].set_xlabel('Price ($000)', fontsize=12)
axes[0].set_title('Distribution of Price', fontsize=14)

# Biểu đồ Histogram
sns.histplot(data=df, x=df['price'] / 1000, ax=axes[1], color=histogram_color)
axes[1].set_xlabel('Price ($000)', fontsize=12)
axes[1].set_ylabel('Frequency', fontsize=12)
axes[1].set_title('Histogram of Price', fontsize=14)

plt.show()

```



Đánh giá

- Qua 2 biểu đồ ta thấy giá nhà không có phân bố chuẩn, phần bên phải có chứa dữ liệu ngoại lai (Outlier)
- Hãy xem xét kỹ hơn về các giá trị ngoại lai tồn tại ở phía bên phải xa.

Dữ liệu ngoại lai (Outliers)

Các thống kê cơ bản về giá nhà từ cột "price" trong DataFrame.

- print(f'median: {df.price.median()}'): In ra giá nhà ở vị trí trung vị (median) trong dữ liệu.
- print(f'IQR: {np.percentile(df.price.sort_values(), 75) - np.percentile(df.price.sort_values(), 25)}'): In ra phạm vi IQR (khoảng từ phân

vị) của dữ liệu. IQR được tính bằng sự khác biệt giữa phân vị 75% và phân vị 25% của dữ liệu.

```
In [13]: print(f'Min: {df.price.min()}')
print(f'mean: {df.price.mean()}')
print(f'median: {df.price.median()}')
print(f'IQR: {np.percentile(df.price.sort_values(), 75) - np.percentile(df.price.sort_values(), 25)}')
print(f'Max: {df.price.max()}')

Min: 75000.0
mean: 540178.9448403517
median: 450000.0
IQR: 323387.5
Max: 7700000.0
```

Thống kê cơ bản về giá nhà từ cột "price" trong DataFrame

- Giá nhà thấp nhất trong dữ liệu là 75,000 đô la.
- Giá nhà trung bình trong dữ liệu là 540,182.16 đô la.
- Giá nhà ở vị trí trung vị là 450,000 đô la.
- Phạm vi IQR (khoảng từ phân vị) của dữ liệu là 323,050 đô la.
- Giá nhà cao nhất trong dữ liệu là 7,700,000 đô la.

Xác định các ngoại lệ (outliers) trong dữ liệu giá nhà dựa trên phương pháp IQR (phạm vi giữa các phân vị)

Kết quả sẽ cho ra các dòng trong DataFrame df mà giá nhà không nằm trong khoảng IQR được xác định, sắp xếp theo cột 'id' theo thứ tự giảm dần. Điều này giúp xác định các dòng có giá trị giá nhà ngoại lệ trong tập dữ liệu.

```
In [14]: Q1 = np.percentile(df.price.sort_values(), 25)
Q3 = np.percentile(df.price.sort_values(), 75)
IQR = np.percentile(df.price.sort_values(), 75) - np.percentile(df.price.sort_values(), 25)
mask = (df['price'] < Q1 - IQR * 1.5) | (df['price'] > Q3 + IQR * 1.5)

df.loc[mask].sort_values(by = 'id', ascending = False)
```

```
Out[14]:
```

	id	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	condition	grade	sqft_above	sqft_basement	price
11924	9831200520	4	3.00	3720	5000	2.5	0	5	9	2720	1000	1440000.0
4811	9831200500	5	3.75	6810	7500	2.5	0	3	13	6110	700	2480000.0
19667	9831200172	4	3.50	2860	2199	3.0	0	3	10	2860	0	1450000.0
19758	9831200159	3	3.25	3890	3452	2.0	0	3	12	2890	1000	2250000.0
18655	9829201020	3	1.25	2400	6653	3.0	2	3	11	2400	0	1390000.0
...
15912	98000130	4	5.00	4630	24054	2.0	3	3	11	4630	0	1430000.0
17980	46100504	4	3.75	4100	22798	1.5	3	5	11	2540	1560	2030000.0
11673	46100350	5	3.50	5000	26540	2.0	3	3	10	3410	1590	1730000.0
216	46100204	5	3.00	3300	33474	1.0	3	3	9	1870	1430	1510000.0
4743	31000165	5	3.50	3620	7821	2.0	2	3	10	2790	830	1490000.0

1139 rows × 12 columns

Khoảng 5% dữ liệu được coi là ngoại lai (1146/21613). Có nhiều lựa chọn để xử lý ngoại lai, bao gồm loại bỏ chúng, giữ nguyên, vv. Đối với tập dữ liệu này, ta sẽ giữ chúng nguyên vì các căn nhà có giá cao nhất thiết là ngoại lai; chúng có thể là các bất động sản sang trọng hoặc nhà có đặc điểm nổi bật.

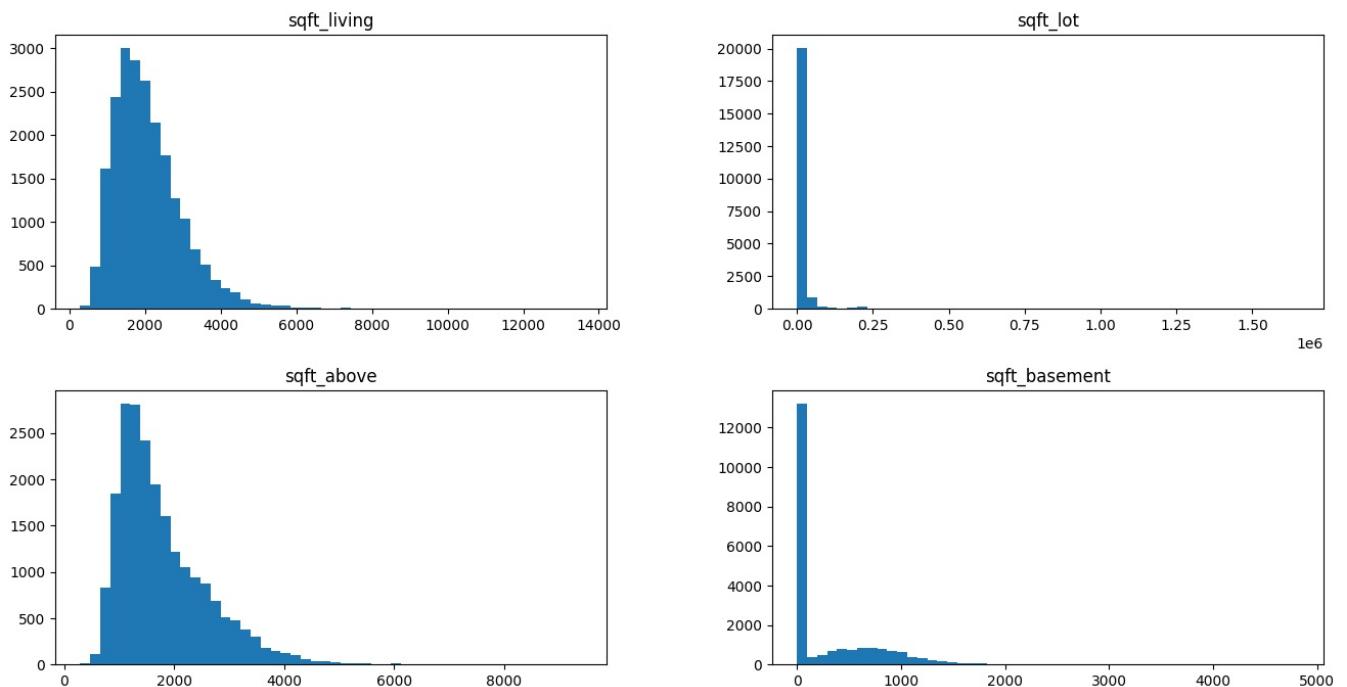
```
In [15]: # Bỏ cột id vì không cần thiết nữa
df.drop("id", axis=1, inplace=True)
```

Khảo sát các biến độc lập

Đối với các biến liên tục

```
In [16]: features_cont = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement']

df[features_cont].hist(bins = 50, figsize = (16, 8), grid = False);
```



```
In [17]: # Danh sách các đặc trưng liên tục
features_cont = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement']

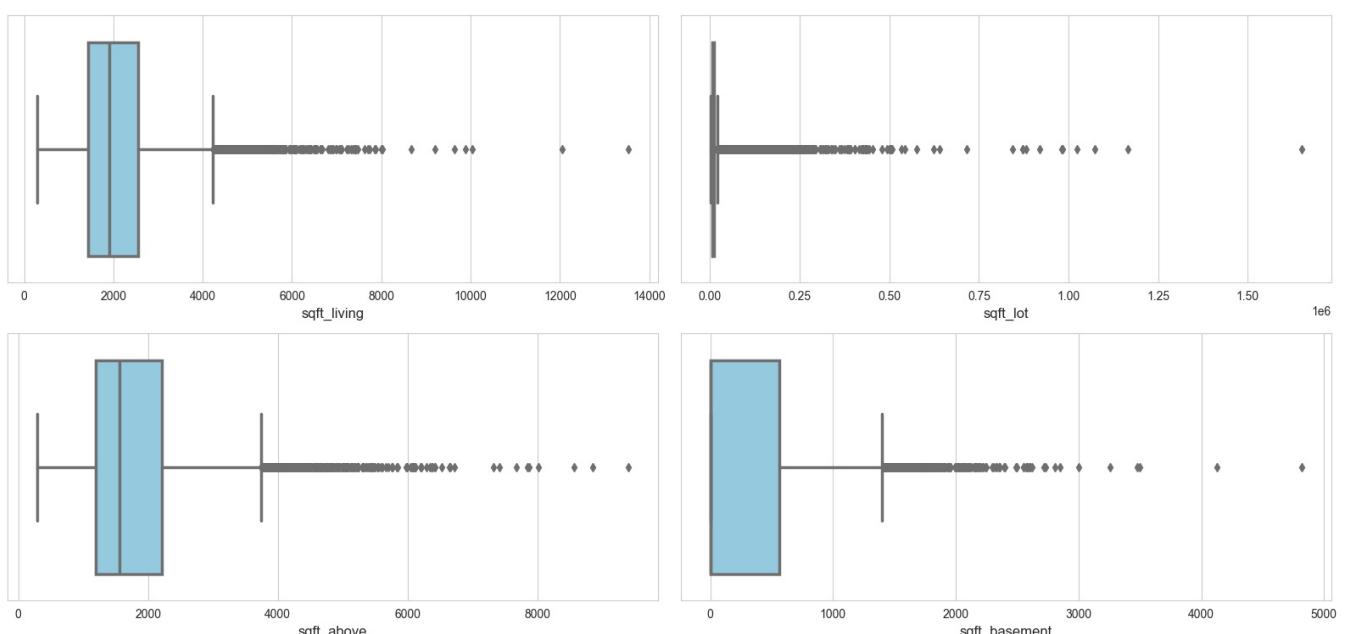
# Thiết lập kích thước và bố cục của biểu đồ
plt.figure(figsize=(16, 8))
sns.set_style("whitegrid")

# Vẽ từng biểu đồ boxplot cho từng đặc trưng
for i, feature in enumerate(features_cont, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=df[feature], color='skyblue', linewidth=2.5)
    plt.xlabel(feature, fontsize=12)

# Định dạng tiêu đề và ghi chú
plt.suptitle('Boxplot of Continuous Features', fontsize=16, y=1.03)
plt.tight_layout()

# Hiển thị biểu đồ
plt.show()
```

Boxplot of Continuous Features



Đánh giá:

- Tất cả các đặc điểm "sqft" đều lệch về phía bên trái (tương tự như phân phối của "price").
- Đều có dữ liệu ngoại lai bên phải rất rõ
- Khi phân phối của một đặc điểm có đuôi dài, việc tỷ lệ đặc điểm (cả tỷ lệ min-max và chuẩn hóa) sẽ làm giá trị của hầu hết các mẫu bị nén vào một phạm vi nhỏ. Thường thì các mô hình máy học không thích điều này. Do đó, trước khi tỷ lệ đặc điểm, chúng ta nên trước tiên biến đổi nó để giảm đuôi dài và làm cho phân phối gần như đối xứng, giúp mô hình học được hiệu quả hơn.

Giải thích: Khi dữ liệu có phân phối lệch và có đuôi dài, việc tỷ lệ hóa có thể làm mất thông tin quan trọng trong các mẫu có giá trị cao hoặc thấp. Điều này có thể làm giảm hiệu suất của mô hình máy học. Thay vào đó, việc biến đổi dữ liệu trước khi tỷ lệ có thể làm cho phân phối gần như đối xứng và giúp mô hình học được hiệu quả hơn.

- Một cách phổ biến để làm điều này cho các đặc điểm dương với đuôi dài về phía bên phải là thay thế đặc điểm bằng căn bậc hai của nó (hoặc tăng đặc điểm lên một lũy thừa giữa 0 và 1). Nếu đặc điểm có đuôi dài và nặng, thì việc thay thế đặc điểm bằng logarithm của nó có thể hữu ích để giảm độ biến thiên của dữ liệu và làm cho phân phối gần như đối xứng.

=> Việc thay thế đặc điểm bằng căn bậc hai hoặc logarit có thể giúp làm giảm độ biến thiên của dữ liệu và làm cho phân phối gần như đối xứng. Điều này có thể làm cho dữ liệu phù hợp hơn với giả định của các mô hình máy học và giúp cải thiện hiệu suất của chúng.

Xem ở dạng 3D

```
In [18]: import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Tạo figure cho biểu đồ 1
fig1 = plt.figure(figsize=(10, 8))
ax1 = fig1.add_subplot(111, projection='3d')

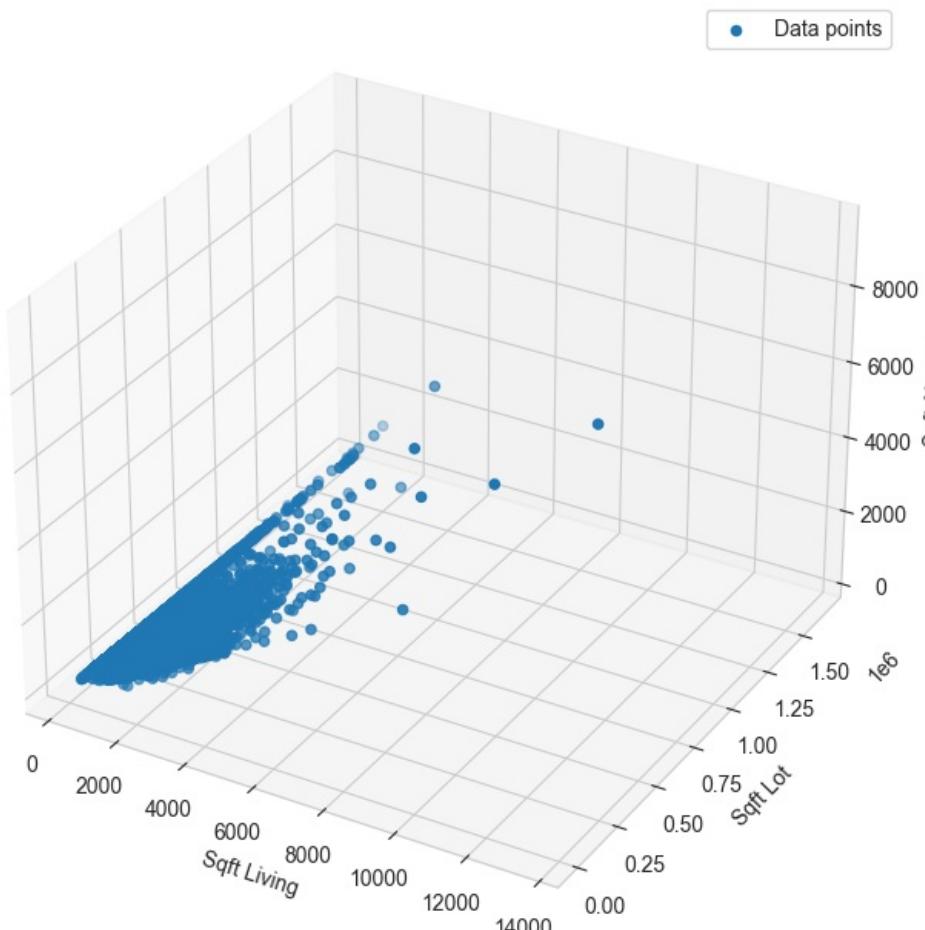
# Vẽ biểu đồ 1
ax1.scatter(df['sqft_living'], df['sqft_lot'], df['sqft_above'], label='Data points')
ax1.set_xlabel('Sqft Living')
ax1.set_ylabel('Sqft Lot')
ax1.set_zlabel('Sqft Above')
ax1.set_title('Scatter Plot 1')
ax1.legend()

# Tạo figure cho biểu đồ 2
fig2 = plt.figure(figsize=(10, 8))
ax2 = fig2.add_subplot(111, projection='3d')

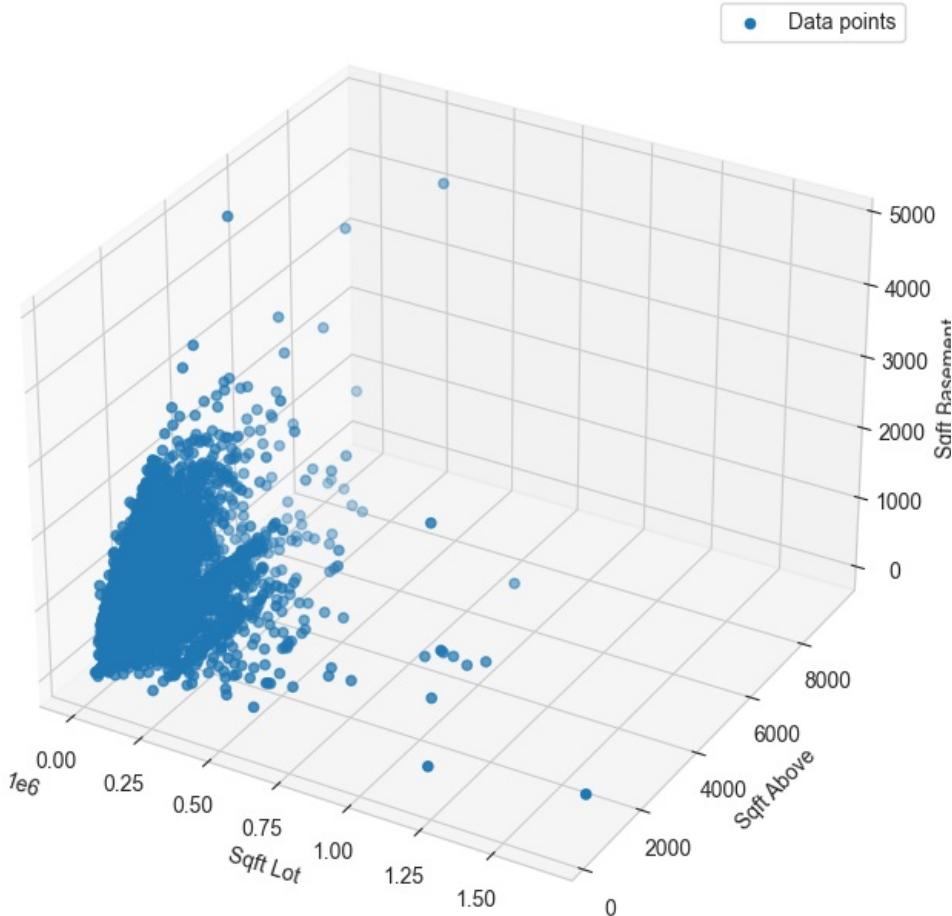
# Vẽ biểu đồ 2
ax2.scatter(df['sqft_lot'], df['sqft_above'], df['sqft_basement'], label='Data points')
ax2.set_xlabel('Sqft Lot')
ax2.set_ylabel('Sqft Above')
ax2.set_zlabel('Sqft Basement')
ax2.set_title('Scatter Plot 2')
ax2.legend()

# Hiển thị biểu đồ
plt.show()
```

Scatter Plot 1



Scatter Plot 2

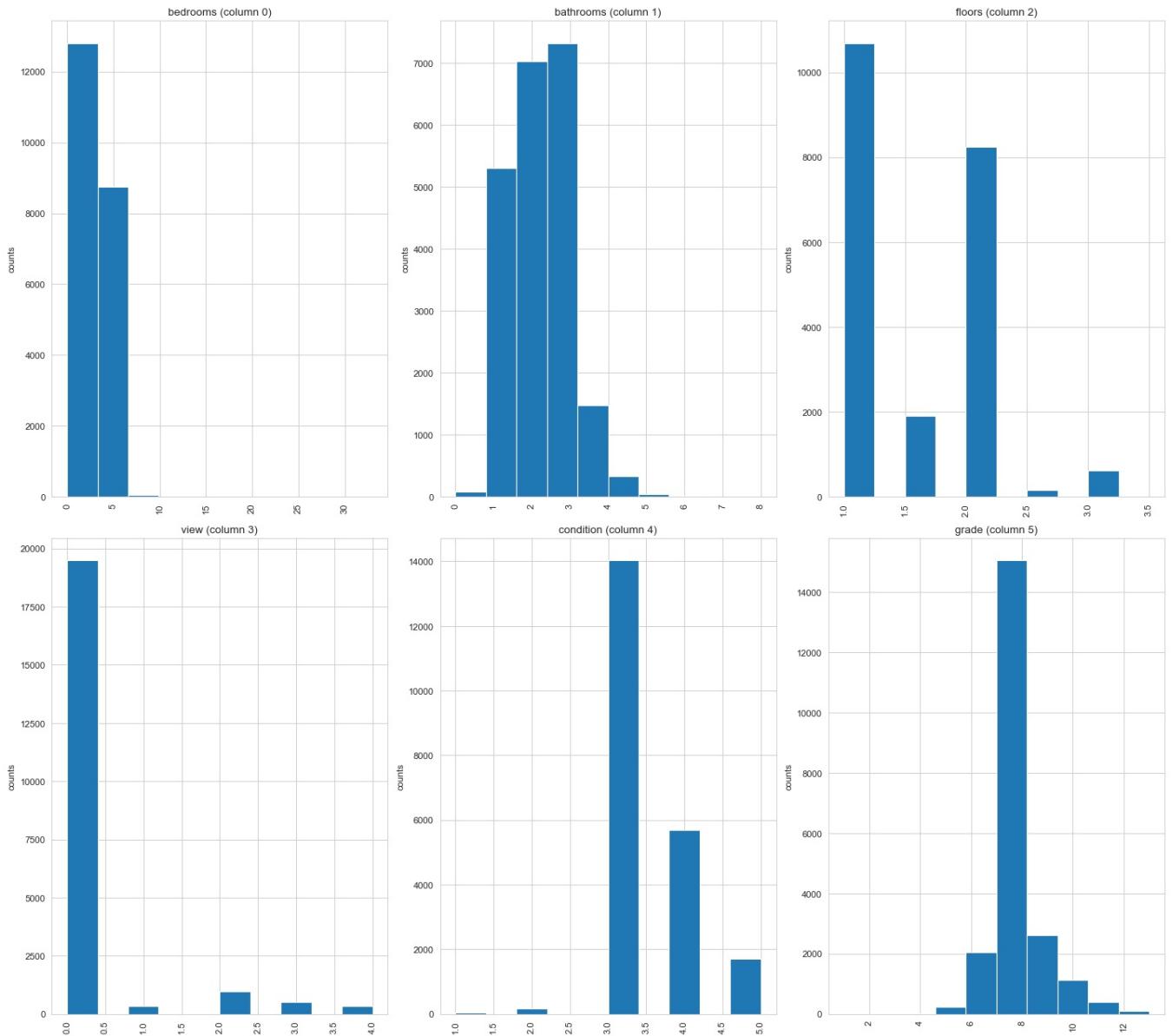


Đối với các biến rời rạc

```
In [19]: # Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]]
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) // nGraphPerRow # Sửa thành phép chia lấy phần nguyên
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor =
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
            plt.ylabel('counts')
            plt.xticks(rotation = 90)
            plt.title(f'{columnNames[i]} ({column {i}})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()
```

Các biểu đồ phân phối (histogram/bar graph) của dữ liệu trong từng cột

```
In [20]: # Các biểu đồ phân phối (histogram/bar graph) của dữ liệu trong từng cột
plotPerColumnDistribution(df, 10, 3)
```



```
# Hàm xử lý nhóm dữ liệu có dạng category (rời rạc)
# Sai số chuẩn = S/sqrt(n)
def groupCategory(df, catCol, numberCol):
    cols = {'count': 'Số lượng', 'mean': 'Trung bình', 'std': 'Độ lệch chuẩn', 'sem': 'Sai số chuẩn'}
    return df.groupby(catCol, as_index=False)[numberCol] \
        .agg(['count', 'mean', 'std', 'sem']) \
        .sort_values('count') \
        .rename(columns=cols) \
        .reset_index(drop=True)

def chartCategory(dfCat, figsize=(12, 6)):
    fig, ax1 = plt.subplots(figsize=figsize)
    dfCat.plot.bar(x=0, y=['Trung bình', 'Độ lệch chuẩn'], ax=ax1, legend=False, ylabel='price', color=['blue', 'red'])
    ax2 = ax1.twinx()
    dfCat.plot(x=0, y=['Số lượng'], ax=ax2, legend=False, ylabel='Số lượng', color='green')
    for label in ax2.get_yticklabels():
        label.set_color('darkgreen')

    ax3 = ax1.twinx()
    dfCat.plot(x=0, y=['Sai số chuẩn'], ax=ax3, legend=False, ylabel='Sai số chuẩn', color='red')
    ax3.spines.right.set_position(("axes", 1.2))
    for label in ax3.get_yticklabels():
        label.set_color('red')

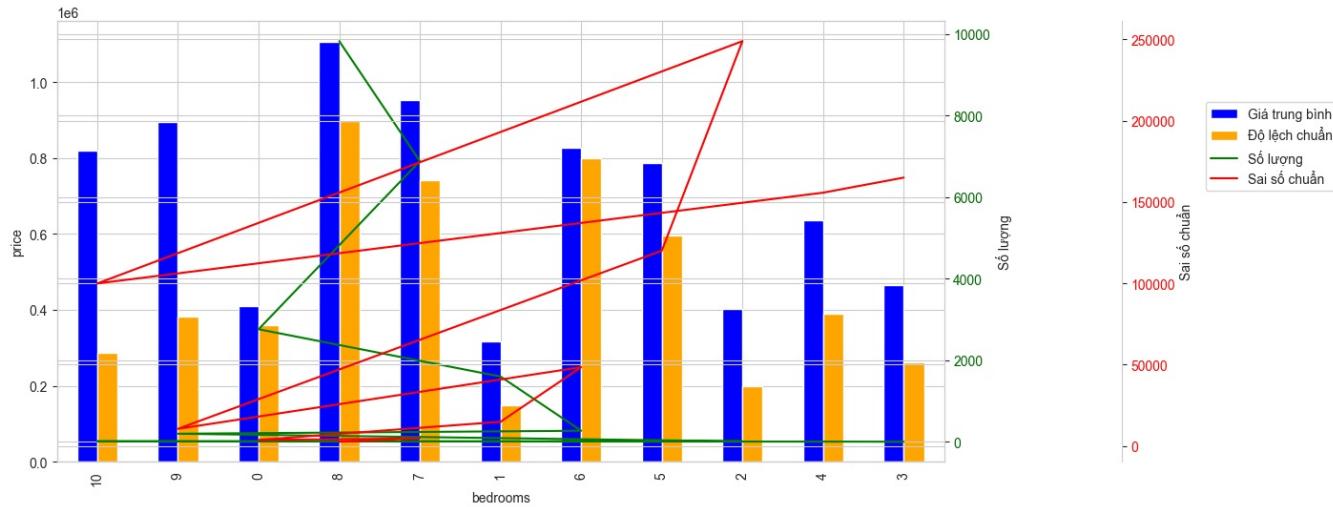
    legends = ['Giá trung bình', 'Độ lệch chuẩn', 'Số lượng', 'Sai số chuẩn']
    fig.legend(legends, bbox_to_anchor=(1.25, 0.75))
```

```
# Loại bỏ các dòng có bedrooms là 11 hoặc 33 vì nó chỉ có 1 giá trị sẽ ảnh hưởng tới Độ lệch chuẩn và Sai số chuẩn
df_filtered = df[~df['bedrooms'].isin([11, 33])]
# Sử dụng hàm groupCategory để nhóm dữ liệu và hiển thị biểu đồ
dfClass = groupCategory(df_filtered, 'bedrooms', 'price')
dfClass
```

Out [22]:

	bedrooms	Số lượng	Trung bình	Độ lệch chuẩn	Sai số chuẩn
0	10	3	8.200000e+05	285832.118559	165025.250593
1	9	6	8.939998e+05	381533.900984	155760.562831
2	0	13	4.102231e+05	360374.846292	99949.998976
3	8	13	1.105077e+06	897495.725295	248920.527466
4	7	38	9.514478e+05	740350.184560	120100.659074
5	1	199	3.176580e+05	148959.667008	10559.470825
6	6	272	8.258535e+05	799610.343019	48483.498582
7	5	1601	7.868741e+05	596536.599057	14908.756718
8	2	2760	4.013877e+05	198128.920756	3771.319553
9	4	6881	6.355771e+05	388928.076013	4688.603596
10	3	9822	4.662555e+05	262643.166265	2650.123666

In [23]: chartCategory(dfClass)

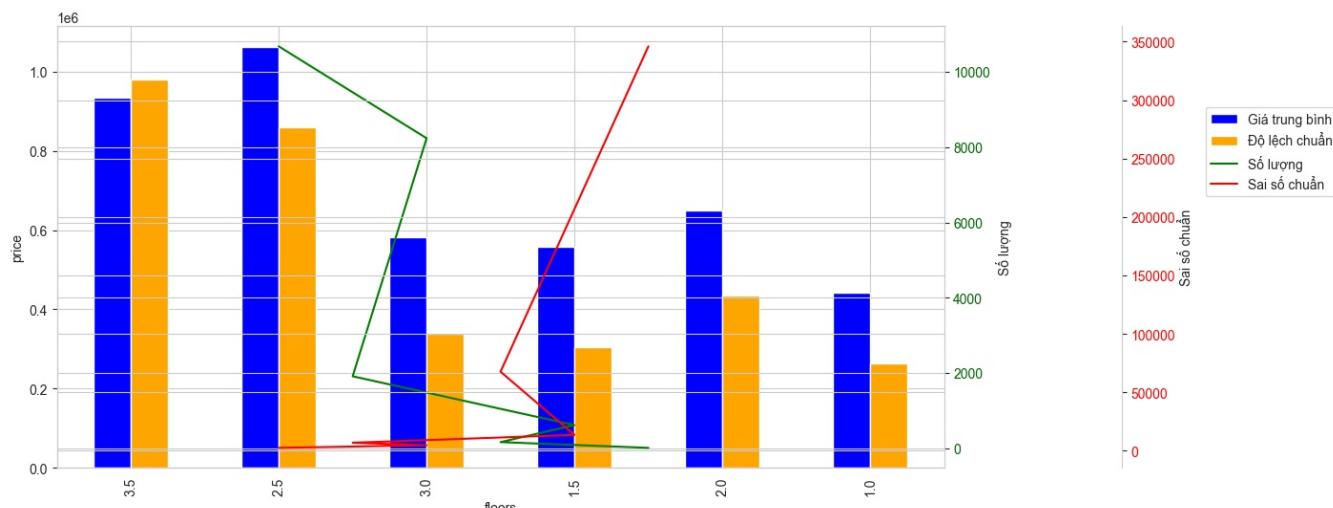


In [24]: dfFloors = groupCategory(df, 'floors', 'price')
dfFloors

Out [24]:

	floors	Số lượng	Trung bình	Độ lệch chuẩn	Sai số chuẩn
0	3.5	8	9.339375e+05	978736.081364	346035.460062
1	2.5	161	1.061021e+06	858836.032602	67685.761455
2	3.0	613	5.826201e+05	338499.140322	13671.851508
3	1.5	1910	5.590449e+05	303722.101177	6949.598457
4	2.0	8238	6.490828e+05	434301.519356	4784.983614
5	1.0	10680	4.422196e+05	264156.610688	2556.088329

In [25]: chartCategory(dfFloors)

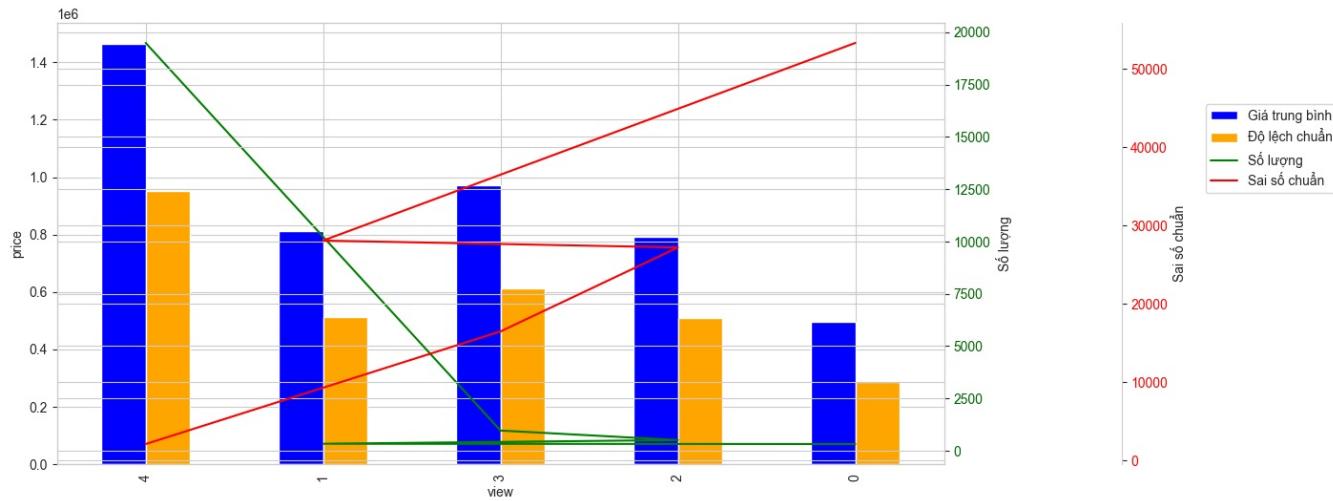


In [26]: dfView = groupCategory(df, 'view', 'price')
dfView

Out [26]:

	view	Số lượng	Trung bình	Độ lệch chuẩn	Sai số chuẩn
0	4	319	1.464363e+06	952502.774655	53329.917019
1	1	332	8.125186e+05	511346.054675	28063.760645
2	3	509	9.732984e+05	613406.273734	27188.755067
3	2	963	7.927462e+05	510517.106752	16451.183846
4	0	19487	4.966160e+05	287330.208507	2058.300575

In [27]: chartCategory(dfView)

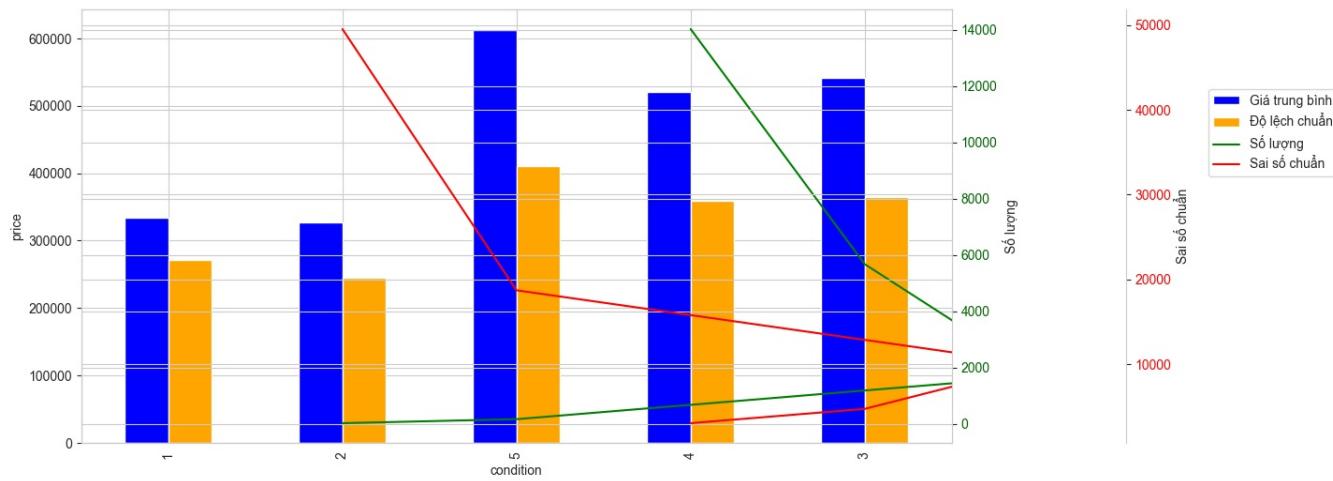


In [28]: dfCondition = groupCategory(df, 'condition', 'price')

Out [28]:

	condition	Số lượng	Trung bình	Độ lệch chuẩn	Sai số chuẩn
0	1	30	334431.666667	271172.804837	49509.154064
1	2	172	327316.215116	245683.980963	18733.227925
2	5	1701	612577.742504	411317.859815	9972.990621
3	4	5678	521295.650757	358800.205886	4761.622805
4	3	14029	542093.108133	364687.591003	3078.985392

In [29]: chartCategory(dfCondition)



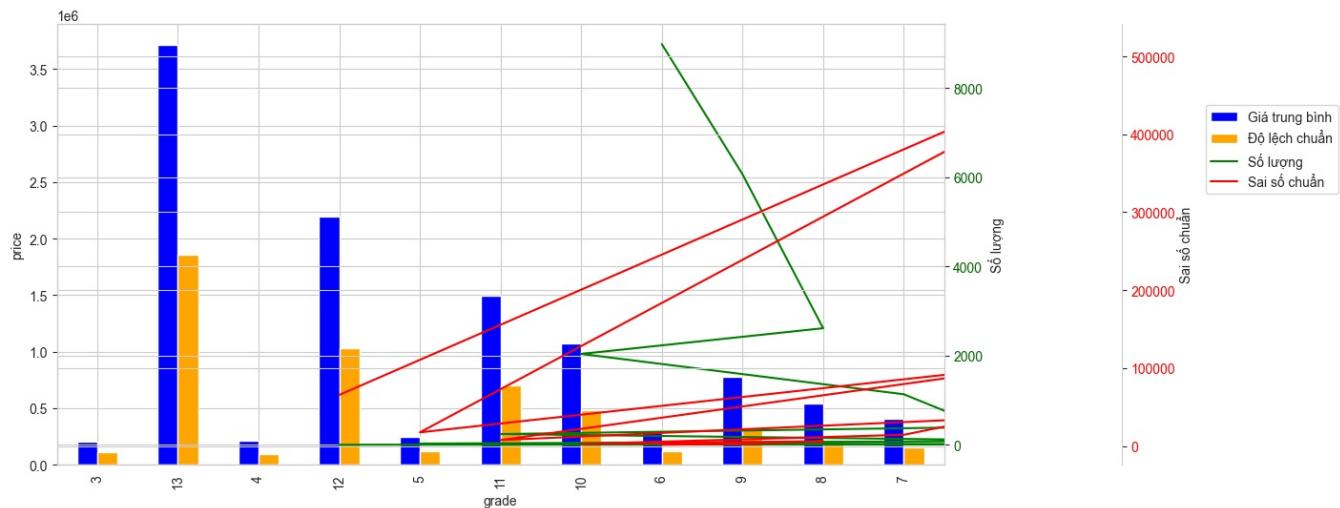
In [30]: # Loại bỏ các dòng có grade là 1 vì nó chỉ có 1 giá trị sẽ ảnh hưởng tới Độ lệch chuẩn và Sai số chuẩn
df_filtered = df[~df['grade'].isin([1])]
dfGrade = groupCategory(df_filtered, 'grade', 'price')
dfGrade

Out [30]:

	grade	Số lượng	Trung bình	Độ lệch chuẩn	Sai số chuẩn
0	3	3	2.056667e+05	1.135180e+05	65539.640253
1	13	13	3.710769e+06	1.859545e+06	515744.994555
2	4	29	2.143810e+05	9.430617e+04	17512.215725
3	12	90	2.192500e+06	1.027337e+06	108290.808421
4	5	242	2.485240e+05	1.181003e+05	7591.773531
5	11	399	1.497792e+06	7.051274e+05	35300.521460
6	10	1134	1.072347e+06	4.840508e+05	14374.224521
7	6	2038	3.019166e+05	1.229522e+05	2723.543931
8	9	2614	7.738104e+05	3.165611e+05	6191.627461
9	8	6066	5.428923e+05	2.176490e+05	2794.508567
10	7	8981	4.025933e+05	1.558953e+05	1645.018312

In [31]:

chartCategory(dfGrade)



In [32]:

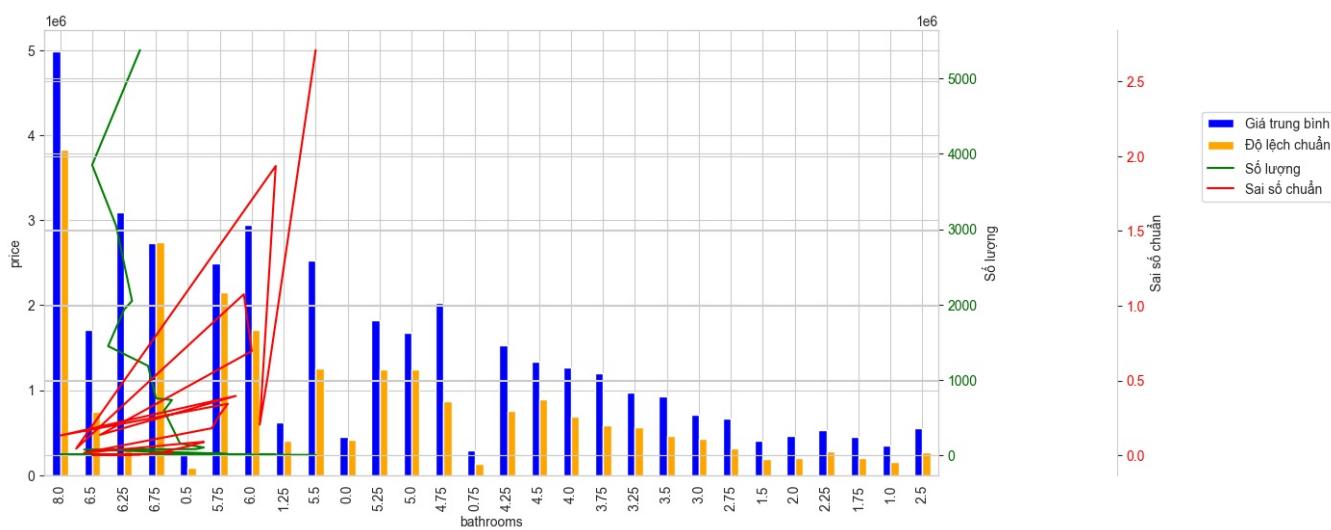
```
# Loại bỏ các dòng có bathrooms là 7.50 và 7.75 vì nó chỉ có 1 giá trị sẽ ảnh hưởng tới Độ lệch chuẩn và Sai số chuẩn
df_filtered = df[~df['bathrooms'].isin([7.50,7.75])]
dfBathrooms = groupCategory(df_filtered, 'bathrooms', 'price')
dfBathrooms
```

Out [32] :

	bathrooms	Số lượng	Trung bình	Độ lệch chuẩn	Sai số chuẩn
0	8.00	2	4.990000e+06	3.832519e+06	2.710000e+06
1	6.50	2	1.710000e+06	7.495332e+05	5.300000e+05
2	6.25	2	3.095000e+06	2.899138e+05	2.050000e+05
3	6.75	2	2.735000e+06	2.736503e+06	1.935000e+06
4	0.50	4	2.373750e+05	8.888886e+04	4.444443e+04
5	5.75	4	2.492500e+06	2.153778e+06	1.076889e+06
6	6.00	6	2.948333e+06	1.706475e+06	6.966655e+05
7	1.25	9	6.217722e+05	4.088817e+05	1.362939e+05
8	5.50	10	2.522500e+06	1.254634e+06	3.967502e+05
9	0.00	10	4.490950e+05	4.172420e+05	1.319435e+05
10	5.25	13	1.817962e+06	1.241665e+06	3.443760e+05
11	5.00	21	1.674167e+06	1.248295e+06	2.724004e+05
12	4.75	23	2.022300e+06	8.672724e+05	1.808388e+05
13	0.75	72	2.945209e+05	1.380845e+05	1.627342e+04
14	4.25	79	1.526653e+06	7.616379e+05	8.569096e+04
15	4.50	100	1.334211e+06	8.910568e+05	8.910568e+04
16	4.00	136	1.268405e+06	6.933076e+05	5.945064e+04
17	3.75	155	1.198179e+06	5.902669e+05	4.741138e+04
18	3.25	589	9.707532e+05	5.633904e+05	2.321410e+04
19	3.50	731	9.324017e+05	4.652734e+05	1.720876e+04
20	3.00	753	7.086619e+05	4.308749e+05	1.570195e+04
21	2.75	1185	6.603505e+05	3.098475e+05	9.000961e+03
22	1.50	1446	4.093457e+05	1.938899e+05	5.098836e+03
23	2.00	1930	4.579050e+05	2.060723e+05	4.690736e+03
24	2.25	2047	5.337688e+05	2.811597e+05	6.214329e+03
25	1.75	3047	4.548846e+05	2.020750e+05	3.660803e+03
26	1.00	3852	3.470412e+05	1.545593e+05	2.490302e+03
27	2.50	5378	5.536557e+05	2.671153e+05	3.642406e+03

In [33] :

chartCategory(dfBathrooms)



Xem tương quan ở dạng biểu đồ hộp

In [34] :

```
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize = (15, 35))
gs = gridspec.GridSpec(ncols = 2, nrows = 7, hspace = 0.5)

ax1 = fig.add_subplot(gs[0, 0:1])
sns.boxplot(x=df['bedrooms'],y=df['price'], ax=ax1)

ax2 = fig.add_subplot(gs[0, 1:2])
sns.boxplot(x=df['floors'],y=df['price'], ax=ax2)

ax4 = fig.add_subplot(gs[1, 1:2])
```

```

sns.boxplot(x=df['view'],y=df['price'], ax=ax4)

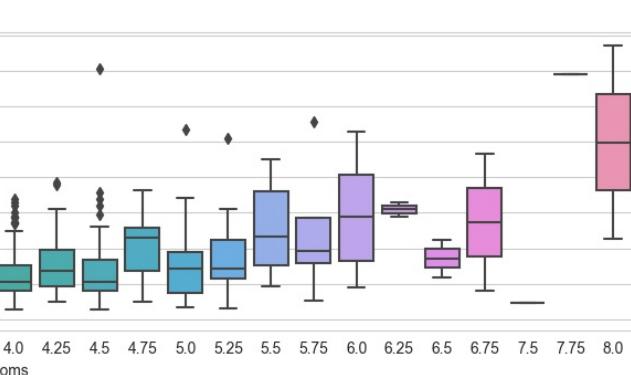
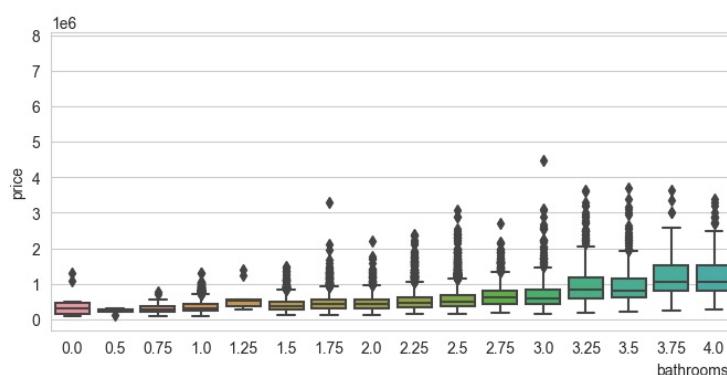
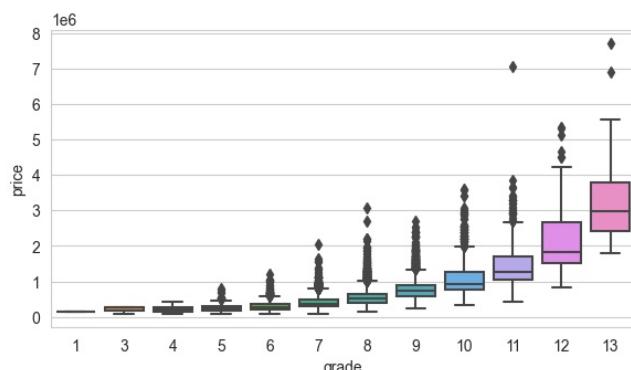
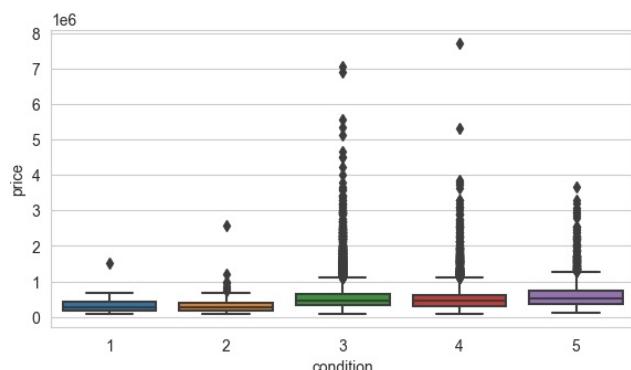
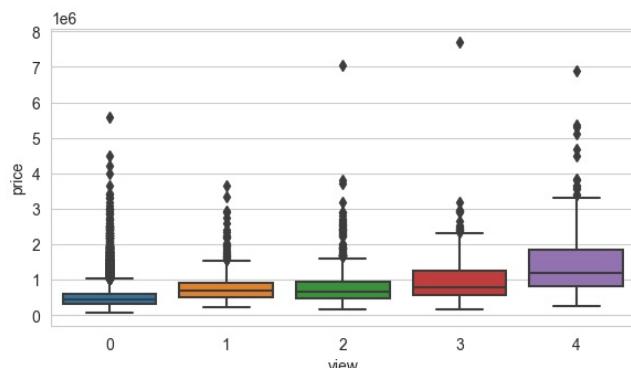
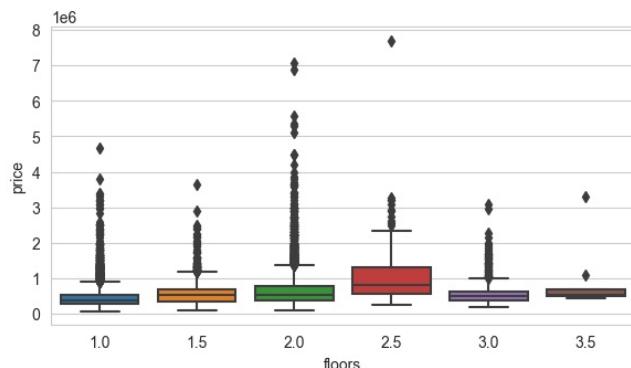
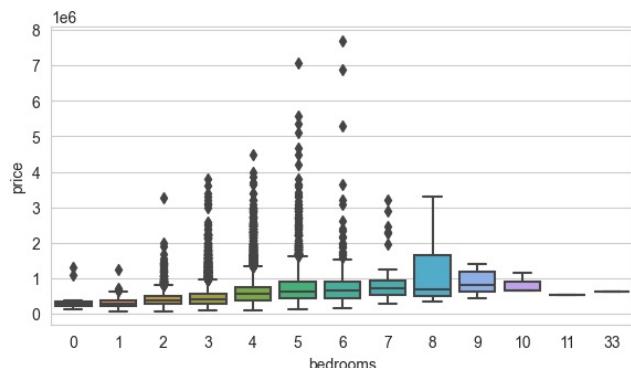
ax5 = fig.add_subplot(gs[2, 0:1])
sns.boxplot(x=df['condition'],y=df['price'], ax=ax5)

ax6 = fig.add_subplot(gs[2, 1:2])
sns.boxplot(x=df['grade'],y=df['price'], ax=ax6)

ax7 = fig.add_subplot(gs[3, 0:2])
sns.boxplot(x=df['bathrooms'],y=df['price'], ax=ax7)

```

Out[34]: <Axes: xlabel='bathrooms', ylabel='price'>



Đánh giá

"bedrooms", "view", "grade", và "bathrooms" có một mẫu tương tự: càng tăng giá trị, biến thiên càng nhiều.

Ma trận tương quan (Correlation)

Xem tương quan giữa các biến với nhau

In [35]:

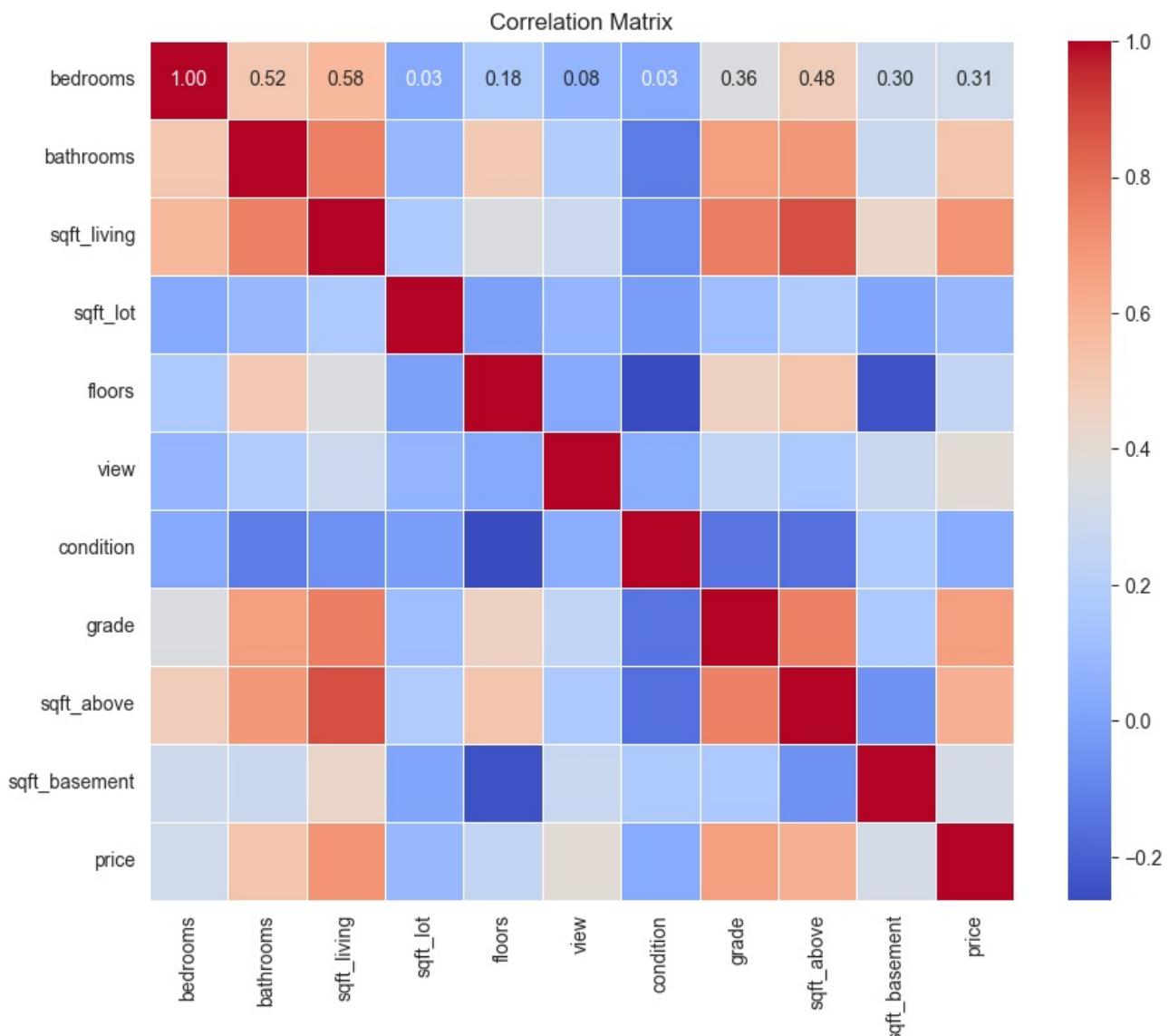
```
# Tạo ma trận tương quan
corr_matrix = df.corr()

# Tạo một figure mới
plt.figure(figsize=(10, 8))

# Vẽ biểu đồ ma trận tương quan bằng seaborn
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

# Đặt tiêu đề cho biểu đồ
plt.title('Correlation Matrix')

# Hiển thị biểu đồ
plt.show()
```



Biểu đồ heatmap của ma trận tương quan hiển thị mối quan hệ tương quan giữa các cặp biến số trong dữ liệu, với các giá trị tương quan được biểu diễn bằng màu sắc và các số trên biểu đồ.

Mối quan hệ giữa các biến số và phân phối của chúng

Biểu đồ scatter matrix là một công cụ mạnh mẽ để trực quan hóa mối quan hệ giữa các biến số trong dữ liệu, đặc biệt là để phát hiện mối quan hệ tương quan giữa chúng.

In [36]:

```
# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include=[np.number]) # chỉ giữ lại các cột số
    # Loại bỏ các dòng có NaN
    df = df.dropna(axis='rows')
    df = df[[col for col in df if df[col].nunique() > 1]]
    columnNames = list(df)

    if len(columnNames) > 10:
```

```

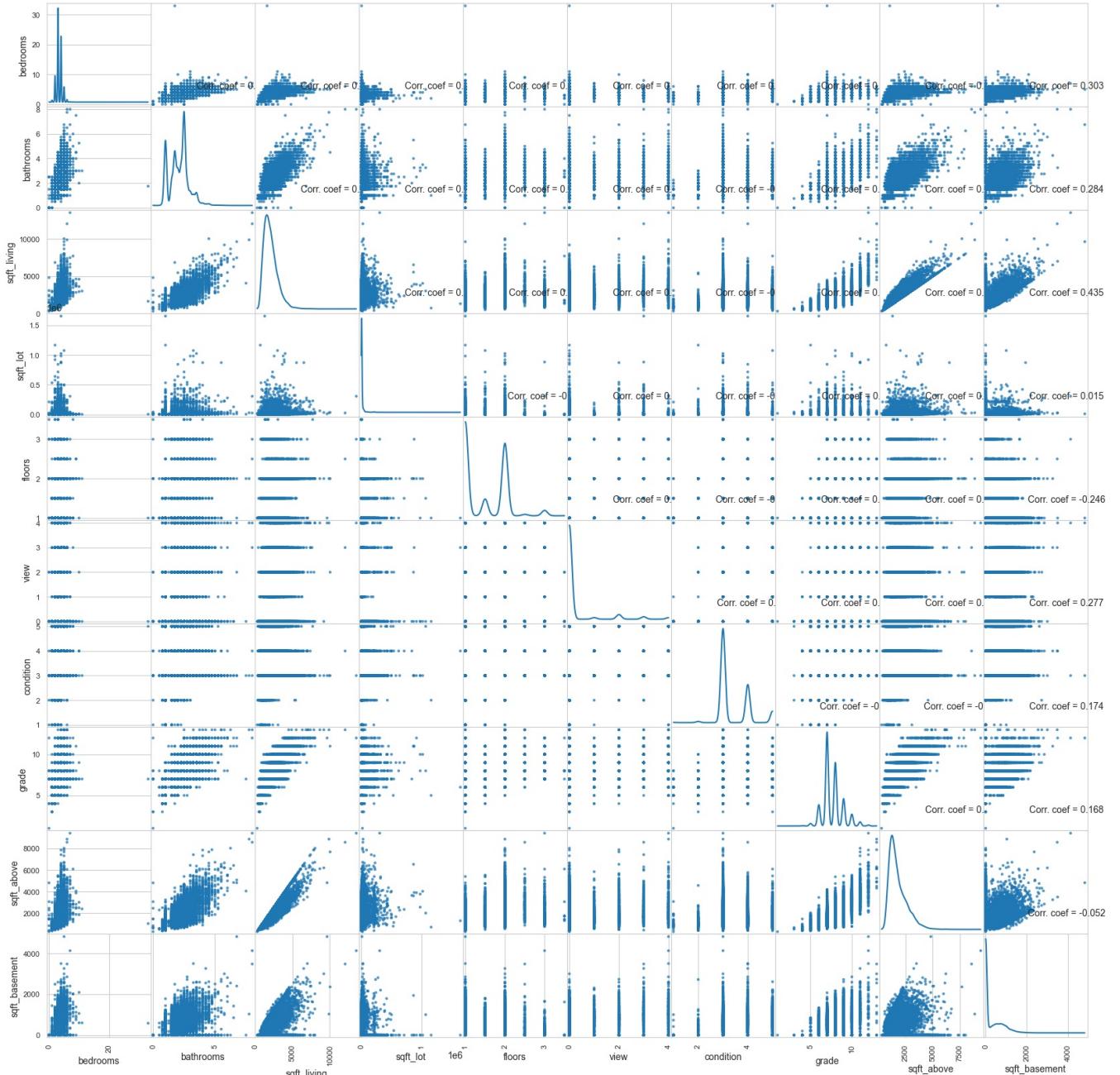
columnNames = columnNames[:10]

df = df[columnNames]
ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
corrs = df.corr().values
for i, j in zip(*plt.np.triu_indices_from(ax, k=1)):
    ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center',
    plt.suptitle('Scatter and Density Plot')
    plt.show()

plotScatterMatrix(df, 20, 10)

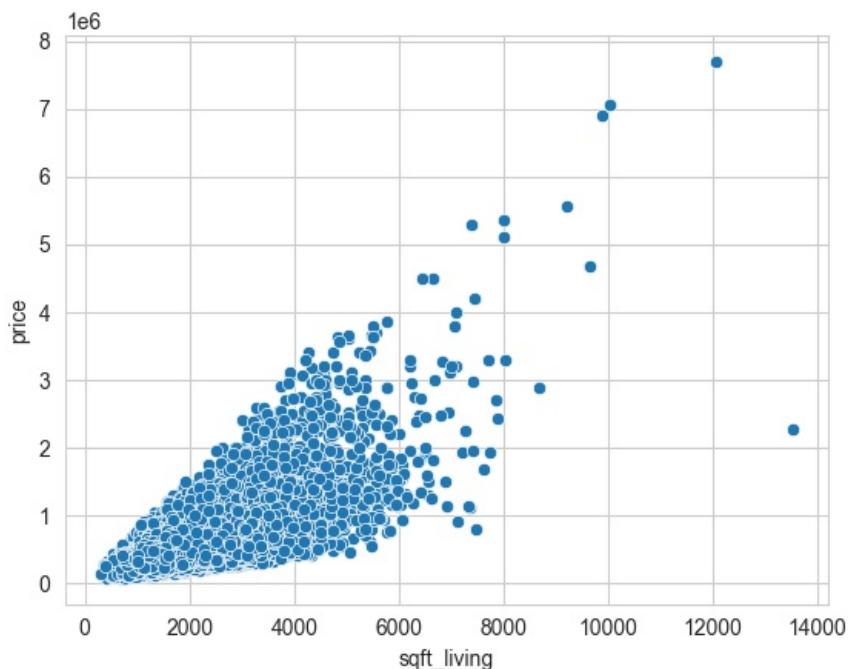
```

Scatter and Density Plot



Có vẻ như cột ảnh hưởng nhiều đến kết quả dự đoán nhất là "sqft_living". Hãy phóng to biểu đồ phân tán của nó.

```
In [37]: sns.scatterplot(df,
                      x = 'sqft_living',
                      y = 'price');
```



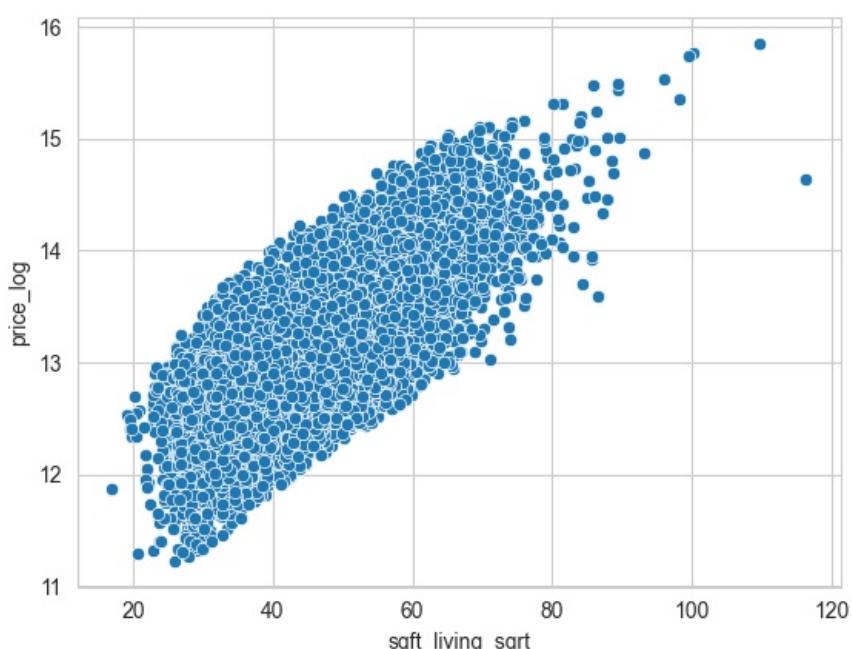
Chú ý

Nó rõ ràng cho thấy sự không đồng nhất của phương sai, nơi biến thể của các sai số không đồng đều trên tất cả các cấp độ của biến độc lập (khi "sqft_living" tăng, biến thể trong "price" trở nên lớn hơn). Khi quan sát thấy sự không đồng nhất của phương sai, thường tốt hơn là áp dụng các biến đổi cho các biến liên quan, như lấy logarith hoặc căn bậc hai để ổn định phương sai.

Biểu đồ dưới đây là một cái nhìn trước về mối quan hệ giữa căn bậc hai của "sqft-living" và logarithm của "price". Nó thể hiện một mối quan hệ tuyến tính tốt, và mô hình có khả năng sẽ hoạt động tốt hơn với điều này.

```
In [38]: sns.scatterplot(df,
                      x = np.sqrt(df['sqft_living']),
                      y = np.log(df['price']))

plt.xlabel('sqft_living_sqrt')
plt.ylabel('price_log');
```



LinearRegression (Hồi quy tuyến tính)

Hồi quy một biến: $y = f(x) = b_0 + b_1x$

- b_0 còn được gọi là intercept, giá trị $f(x)$ tại $x=0$

- b_1 là độ dốc của đường hồi quy tuyến tính

Hồi quy tuyến tính đa biến

$$y = f(x_1, \dots, x_r) = b_0 + b_1x_1 + \dots + b_rx_r$$

$$\bullet \quad f(x_1, x_2) = b_0 + b_1x_1 + b_2x_2$$

Các chỉ số đánh giá mô hình hồi quy tuyến tính

MEA (Mean Absolute Error): trung bình các giá trị tuyệt đối của các sai số:

$$1/n \sum_{i=1}^n |y_i - \hat{y}_i|$$

MSE (Mean Squared Error): trung bình các bình phương của các sai số:

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

RMSE (Root Mean Squared Error): căn bậc 2 của trung bình các bình phương của các sai số:

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

R² (R Squared): hệ số xác định của hàm hồi quy:

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$$

RSS (Regression Sum of Squares): tổng các độ lệch bình phương từ hồi quy:

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

ESS (Residual Sum of Squares): tổng các độ lệch bình phương phần dư:

$$\text{ESS} = n \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$$

TSS (Total Sum of Squares): tổng các độ lệch bình phương toàn bộ:

$$\text{TSS} = n \sum_{i=1}^n (y_i - \bar{y})^2$$

Giá trị R² dao động từ 0 đến 1. R² càng gần 1 thì mô hình đã xây dựng càng phù hợp với bộ dữ liệu dùng chạy hồi quy. R² càng gần 0 thì mô hình đã xây dựng càng kém phù hợp với bộ dữ liệu dùng chạy hồi quy. Trường hợp đặt biệt, phương trình hồi quy đơn biến (chỉ có 1 biến độc lập) thì R² chính là bình phương của hệ số tương quan r giữa hai biến đó.

R² càng lớn thì giá trị tổng bình phương sai số càng nhỏ.

Ví dụ giả sử R² là 0.60, thì mô hình hồi quy tuyến tính này phù hợp với tập dữ liệu ở mức 60%. Nói cách khác, 60% biến thiên của biến phụ thuộc được giải thích bởi các biến độc lập. (40% còn lại do sai số do lường, do cách thu thập dữ liệu, do có thể có biến độc lập khác giải thích cho biến phụ thuộc mà chưa được đưa vào mô hình nghiên cứu...)

R² hiệu chỉnh (Adjusted R²):

$$\text{Adj R}^2 = R^2 - \left(k - 1 - \frac{n}{k} \right) (1 - R^2)$$

n là số lượng mẫu quan sát và k là số biến độc lập.

Chỉ số R² tăng khi số biến độc lập tăng, do đó R hiệu chỉnh được tính từ R² thường được sử dụng hơn vì giá trị này phản ánh sát hơn mức độ phù hợp của mô hình hồi quy tuyến tính đa biến.

Ví dụ giá trị R² hiệu chỉnh là 0.725. Như vậy, 72.5% sự biến thiên của biến phụ thuộc được giải thích bởi các biến độc lập. Phần còn lại

27.5% được giải thích bởi các biến ngoài mô hình và sai số ngẫu nhiên.

```
In [39]: # Hàm tính toán R bình phương hiệu chỉnh
def R2Adj(r2, n, k):
    return r2 - (k - 1) / (n - k) * (1 - r2)

# Hàm tính toán hồi quy n trả về các tham số đánh giá và bằng hệ số
def regression(data, model = LinearRegression()):
    model.fit(data['X_train'], data['y_train']) # chạy mô hình hồi quy
    y_predict = model.predict(data['X_test']).reshape(-1, 1) # chạy dự đoán với tập
    mae = metrics.mean_absolute_error(data['y_test'], y_predict)
    mse = metrics.mean_squared_error(data['y_test'], y_predict)
    rmse = np.sqrt(mse)
    r2_train = model.score(data['X_train'], data['y_train'])
    r2_test = model.score(data['X_test'], data['y_test'])
    n, k = data['X_train'].shape[0], data['X_train'].shape[1]
    r2_adj_train = R2Adj(r2_train, n, k)
    r2_adj_test = R2Adj(r2_test, n, k)

    # Trả về các kết quả
    return {
        'model': model,
        'params': {
            'MAE': mae,
            'MSE': mse,
            'RMSE': rmse,
            'R2_train': r2_train,
            'R2_test': r2_test,
            'R2_adj_train': r2_adj_train,
            'R2_adj_test': r2_adj_test
        }
    }
```

Xây dựng mô hình hồi quy

Phân chia dữ liệu (Split The Data) : Tập dữ liệu cần được chia thành 2 phần : dữ liệu để huấn luyện mô hình và kiểm tra

- Tập huấn luyện (training set) : dùng tập này để huấn luyện mô hình tìm ra hàm hồi quy
- Tập kiểm tra (testing set) : dùng để đánh giá mức độ chính xác của mô hình

Sử dụng hàm train_test_split từ thư viện sklearn.model_selection để phân chia dữ liệu thành tập huấn luyện và tập kiểm tra:

```
In [40]: def split_data(X,y, train_size=0.6, random_state=123):
    X_train , X_test , y_train , y_test = train_test_split(X ,y ,train_size= train_size , random_state= random_
    return {'X_train':X_train, 'X_test':X_test, 'y_train':y_train, 'y_test': y_test}
```

```
In [41]: data = df.copy()
data.sample(5)
```

```
Out[41]:   bedrooms bathrooms sqft_living sqft_lot floors view condition grade sqft_above sqft_basement price
14187         3      2.75     2980    27144    1.5      2       5      8      2180          800  947500.0
11081         4      2.50     1930    7862     2.0      0       3      7      1930          0  339500.0
  92          3      1.00     1200   10500     1.0      0       3      7      1200          0  153000.0
  488         4      2.75     2810    7302     2.0      0       3      9      2810          0  699900.0
  2450         2      2.50     1590   1845     2.0      0       3      9      1590          0  409900.0
```

Tách dữ liệu

```
In [42]: X = data.iloc[:, :-1]
y = data.iloc[:, -1:]
```

```
In [43]: print(X.shape, y.shape)
(21610, 10) (21610, 1)
```

```
In [44]: # Xem tập X
X
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	condition	grade	sqft_above	sqft_basement
0	3	1.00	1180	5650	1.0	0	3	7	1180	0
1	3	2.25	2570	7242	2.0	0	3	7	2170	400
2	2	1.00	770	10000	1.0	0	3	6	770	0
3	4	3.00	1960	5000	1.0	0	5	7	1050	910
4	3	2.00	1680	8080	1.0	0	3	8	1680	0
...
21608	3	2.50	1530	1131	3.0	0	3	8	1530	0
21609	4	2.50	2310	5813	2.0	0	3	8	2310	0
21610	2	0.75	1020	1350	2.0	0	3	7	1020	0
21611	3	2.50	1600	2388	2.0	0	3	8	1600	0
21612	2	0.75	1020	1076	2.0	0	3	7	1020	0

21610 rows × 10 columns

```
In [45]: # Xem tập y
y
```

	price
0	221900.0
1	538000.0
2	180000.0
3	604000.0
4	510000.0
...	...
21608	360000.0
21609	400000.0
21610	402101.0
21611	400000.0
21612	325000.0

21610 rows × 1 columns

Chạy mô hình hồi quy

```
In [46]: model_data = split_data(X, y, 0.7)
result = regression(model_data)
```

```
In [47]: result
```

```
Out[47]: {'model': LinearRegression(),
'params': {'MAE': 155261.1410552667,
'MSE': 54534750020.70202,
'RMSE': 233526.76510563414,
'R2_train': 0.6007132036675172,
'R2_test': 0.5592874598995881,
'R2_adj_train': 0.6004754700629266,
'R2_adj_test': 0.5590250615891287}}
```

Truy cập vào các hệ số của mô hình hồi quy tìm được sau khi huấn luyện

```
In [48]: result['model'].coef_
```

```
Out[48]: array([[-3.99371791e+04, -1.86623210e+04, 1.43012739e+02,
 -3.80716112e-01, -5.31892898e+03, 9.29686225e+04,
 5.36140670e+04, 1.01860732e+05, 5.92482149e+01,
 8.37645244e+01]])
```

Kết quả trả về là một mảng numpy chứa các hệ số của mô hình hồi quy. Mỗi hệ số tương ứng với một đặc trưng trong dữ liệu đầu vào. Ví dụ, trong trường hợp này, có mười hệ số được trả về.

Ví dụ:

- Hệ số đầu tiên là -39937.1791, tương ứng với đặc trưng đầu tiên.
- Hệ số thứ hai là -18662.3210, tương ứng với đặc trưng thứ hai.
- Và cứ tiếp tục như vậy cho đến hệ số cuối cùng.

Các hệ số này cho biết mức độ ảnh hưởng của mỗi đặc trưng đầu vào đến dự đoán của mô hình. Đối với các mô hình hồi quy tuyến

tính, hệ số càng lớn (hoặc càng xa khỏi 0), thì đặc trưng đó càng có ảnh hưởng lớn đến dự đoán.

```
In [49]: # Kết quả hệ số
coefficients = np.abs(result['model'].coef_)

# Xác định đặc trưng ảnh hưởng nhất
most_influential_feature_index = np.argmax(coefficients)
most_influential_feature = X.columns[most_influential_feature_index]

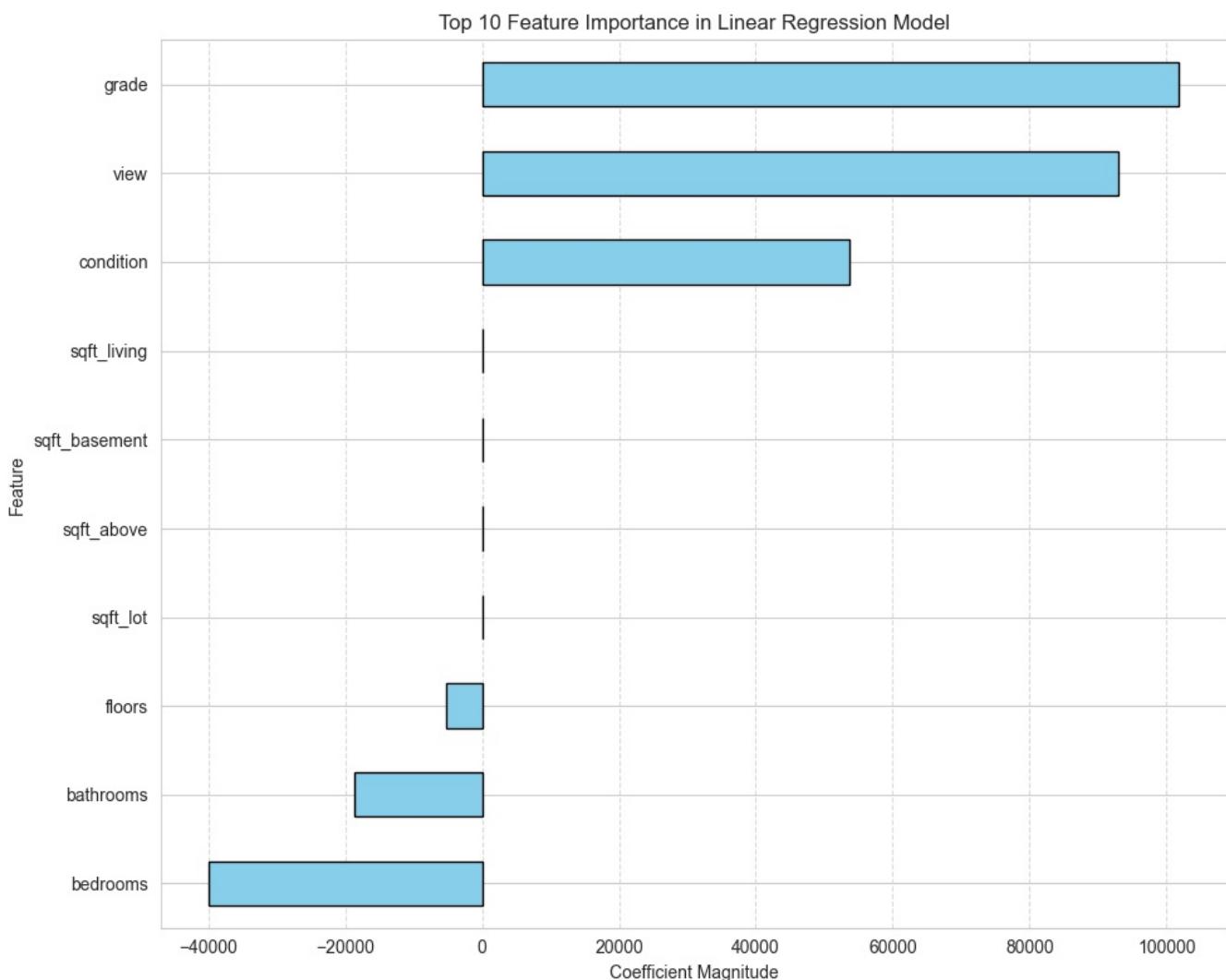
print("Đặc trưng ảnh hưởng nhất là:", most_influential_feature)

Đặc trưng ảnh hưởng nhất là: grade
```

Trực quan hóa độ quan trọng của từng đặc trưng trong mô hình hồi quy tuyến tính

```
In [50]: # this is the importance of each feature
importances = result['model'].coef_[0]
features = X.columns
feat_imp = pd.Series(importances, index=features).sort_values()

# Plotting
plt.figure(figsize=(10, 8))
feat_imp.tail(10).plot(kind='barh', color='skyblue', edgecolor='black')
plt.xlabel('Coefficient Magnitude')
plt.ylabel('Feature')
plt.title('Top 10 Feature Importance in Linear Regression Model')
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Xem các hệ số hồi quy

```
In [51]: print('bo= ', result['model'].intercept_)
coeff_df = pd.DataFrame(result['model'].coef_[0], X.columns, columns=['Coeff'])

bo= [-683989.66146102]
```

Out[51]:

	Coeff
bedrooms	-39937.179085
bathrooms	-18662.321012
sqft_living	143.012739
sqft_lot	-0.380716
floors	-5318.928977
view	92968.622531
condition	53614.066956
grade	101860.731548
sqft_above	59.248215
sqft_basement	83.764524

Giải thích ý nghĩa các hệ số:

b_0 là hệ số chặn trong một mô hình hồi quy tuyến tính là giá trị dự đoán của biến phụ thuộc khi tất cả các biến độc lập đều bằng 0. Trong trường hợp này, hệ số chặn là -683989.66146102.

Các giá trị trên là các hệ số ba, b...b, dự đoán trong hàm $y = f(x_1, \dots, x_r) = b_0 + b_1x_1 + \dots + b_rx_r$

Coeff: Đây là các hệ số hồi quy tương ứng với mỗi biến độc lập.

- bedrooms: Mỗi phòng ngủ tương ứng với giảm trung bình 10816.76 đô la trong giá nhà.
- bathrooms: Mỗi phòng tắm tương ứng với giảm trung bình 20024.93 đô la trong giá nhà.
- sqft_living: Mỗi foot vuông diện tích sống tương ứng với tăng trung bình 75.15 đô la trong giá nhà.
- sqft_lot: Mỗi foot vuông diện tích lô đất tương ứng với tăng trung bình 0.0022 đô la trong giá nhà.
- floors: Mỗi tầng tương ứng với tăng trung bình 25772.47 đô la trong giá nhà.
- view: Mỗi điểm view tương ứng với tăng trung bình 39536.14 đô la trong giá nhà.
- condition: Mỗi điểm condition tương ứng với tăng trung bình 40094.29 đô la trong giá nhà.
- grade: Mỗi điểm grade tương ứng với tăng trung bình 83603.60 đô la trong giá nhà.
- sqft_above: Mỗi foot vuông diện tích ở trên tầng tương ứng với tăng trung bình 17.60 đô la trong giá nhà.
- sqft_basement: Mỗi foot vuông diện tích tầng hầm tương ứng với tăng trung bình 57.55 đô la trong giá nhà.

Đánh giá mô hình qua các chỉ số

In [52]: # Xem các tham số đánh giá mô hình hồi quy
result['params']

Out[52]: {'MAE': 155261.1410552667,
'MSE': 54534750020.70202,
'RMSE': 233526.76510563414,
'R2_train': 0.6007132036675172,
'R2_test': 0.5592874598995881,
'R2_adj_train': 0.6004754700629266,
'R2_adj_test': 0.5590250615891287}

Đây là một tập hợp các tham số đánh giá một mô hình hồi quy. Cụ thể:

- MAE (Mean Absolute Error): Là trung bình của giá trị tuyệt đối của sự sai lệch giữa giá trị thực và giá trị dự đoán. Trong trường hợp này, MAE là khoảng 155,261 đơn vị.
- MSE (Mean Squared Error): Là trung bình của bình phương của sự sai lệch giữa giá trị thực và giá trị dự đoán. MSE là khoảng 54,534,750,020.70202.
- RMSE (Root Mean Squared Error): Là căn bậc hai của MSE. Nó đo lường sai số trung bình giữa giá trị thực và giá trị dự đoán, và có cùng đơn vị với biến số đo. Trong trường hợp này, RMSE là khoảng 233,526.77.
- R2 (R-squared): Là hệ số xác định mức độ giải thích của mô hình đối với biến phụ thuộc. Nó cung cấp một ước lượng về mức độ biến thiên của biến phụ thuộc được giải thích bởi biến độc lập trong mô hình. Giá trị của R2 nằm trong khoảng từ 0 đến 1, với 1 cho biết một mô hình hoàn hảo. Trong trường hợp này, R2 cho tập huấn luyện là khoảng 0.6007 và cho tập kiểm tra là khoảng 0.5593.
- R2 Adjusted (Adjusted R-squared): Đây là phiên bản được điều chỉnh của R2, được sử dụng khi có nhiều biến độc lập trong mô hình. Nó điều chỉnh cho số lượng biến độc lập trong mô hình, và thường thấp hơn hoặc bằng R2. Trong trường hợp này, R2 điều chỉnh cho tập huấn luyện là khoảng 0.6005 và cho tập kiểm tra là khoảng 0.5590.

Các giá trị này cung cấp thông tin quan trọng về hiệu suất của mô hình hồi quy, giúp đánh giá độ chính xác và phù hợp của mô hình.

R hiệu chỉnh của tập test (R2_adj_train) ~ 60% sự biến thiên của giá sản phẩm (biến phụ thuộc) được giải thích bởi các biến độc lập.

Đánh giá mô hình qua hình ảnh

Áp dụng mô hình cho tập dữ liệu kiểm tra:

Lấy giá trị dự đoán trên tập kiểm tra so với giá trị y thực của tập kiểm tra, xem thử mức độ chính xác như thế nào

```
In [53]: # Kiểm tra dự đoán trên dữ liệu tập Train và tập test
# Residual = Observed value - Predicted value
y_train = model_data['y_train'].to_numpy()
y_test = model_data['y_test'].to_numpy()

predict_train = result['model'].predict(model_data['X_train']).reshape(-1, 1)
predict_test = result['model'].predict(model_data['X_test']).reshape(-1, 1)

residual_train = (model_data['y_train'] - predict_train).reset_index(drop=True)
residual_test = (model_data['y_test'] - predict_test).reset_index(drop=True)
```

```
In [54]: model_data['y_train']
```

```
Out[54]:      price
```

```
18252  539000.0
12823  145000.0
9384   403000.0
20337  550000.0
12432  485000.0
...
15379  344500.0
21605  610685.0
17732  565000.0
15727  799950.0
19968  631625.0
```

15126 rows × 1 columns

- Tách dữ liệu đầu ra (target): Dữ liệu đầu ra được chia thành hai phần: y_train chứa dữ liệu đầu ra cho tập huấn luyện và y_test chứa dữ liệu đầu ra cho tập kiểm tra.
- Dự đoán giá trị đầu ra: Sử dụng mô hình hồi quy đã được huấn luyện, dự đoán giá trị đầu ra tương ứng với dữ liệu đầu vào của cả tập huấn luyện (X_train) và tập kiểm tra (X_test). Kết quả dự đoán được lưu trong predict_train và predict_test.
- Tính toán sai số (residuals): Số lỗi hoặc sai số được tính bằng cách trừ giá trị thực tế từ giá trị dự đoán tương ứng. Điều này tạo ra hai chuỗi số: residual_train chứa sai số cho tập huấn luyện và residual_test chứa sai số cho tập kiểm tra.

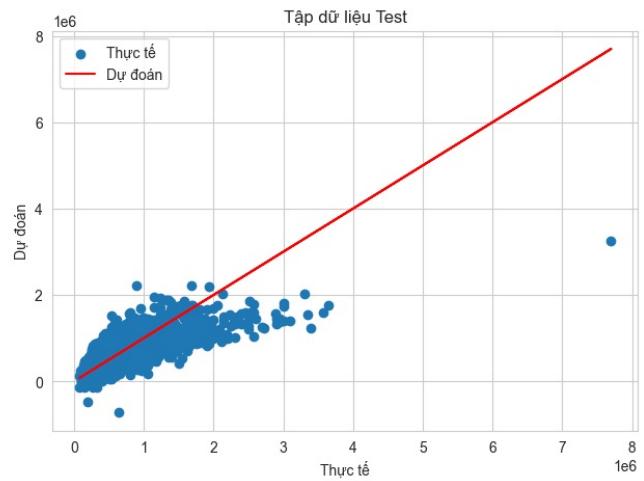
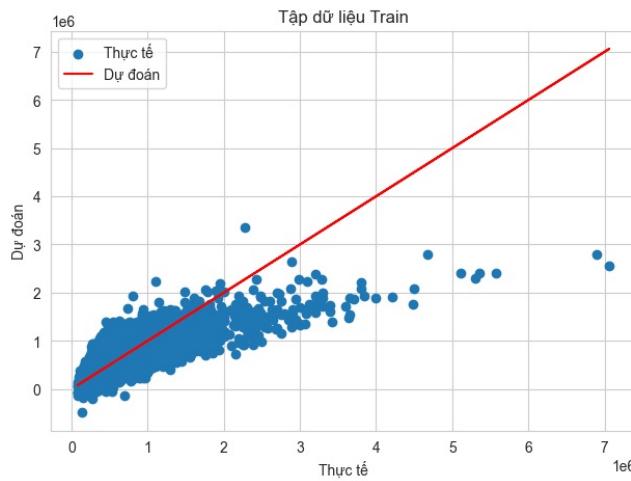
Điều này giúp đánh giá hiệu suất của mô hình trên cả tập huấn luyện và tập kiểm tra bằng cách xem xét sai số giữa giá trị thực tế và giá trị dự đoán.

```
In [55]: def checkPredict(real, predict, ax, title):
    ax.scatter(real, predict) # So sánh tương quan giữa kết quả thực và kết quả dự đoán
    ax.plot(real, real, 'r') # vẽ đường tham chiếu kết quả thực
    ax.set_title(title)
    ax.set_xlabel('Thực tế')
    ax.set_ylabel('Dự đoán')
    ax.legend(['Thực tế', 'Dự đoán'])

def checkResidual(predict, residual, ax, title):
    ax.scatter(predict, residual) # So sánh tương quan giữa kết quả dự đoán và sai số
    ax.hlines(y=0, xmin=predict.min(), xmax=predict.max(), linewidth=2, color='r')
    ax.set_title(title)
    ax.set_xlabel('Thực tế')
    ax.set_ylabel('Dự đoán')
    ax.legend(['Thực tế', 'Dự đoán'])
```

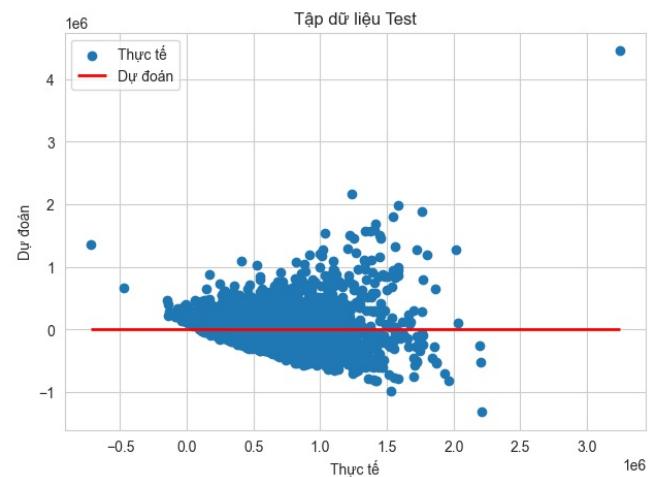
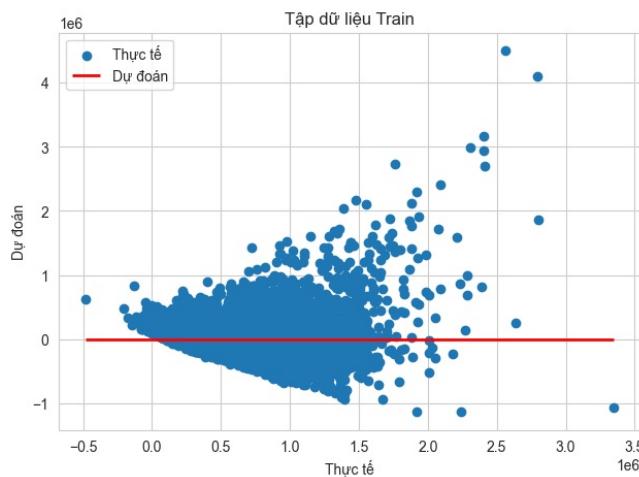
So sánh giá trị thực và giá trị dự đoán)

```
In [56]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,5))
checkPredict(y_train, predict_train, ax=ax1, title='Tập dữ liệu Train')
checkPredict(y_test, predict_test, ax=ax2, title='Tập dữ liệu Test')
```

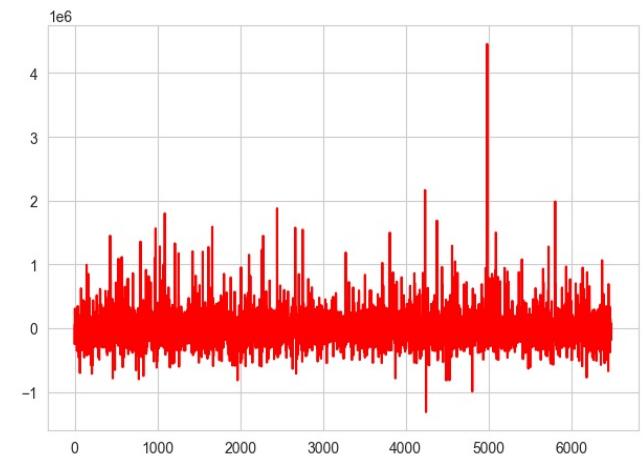
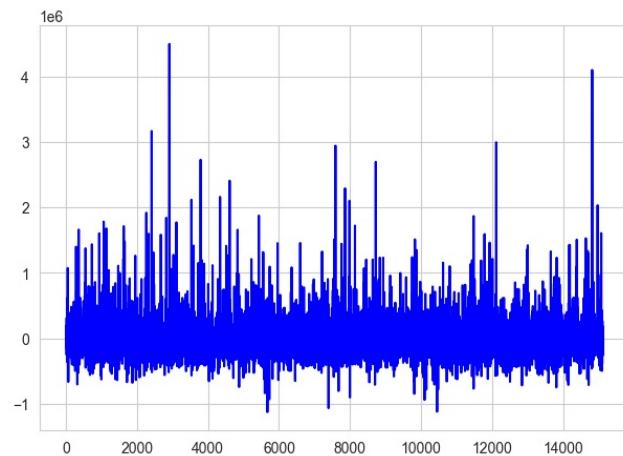


Xem tương quan giá trị dự đoán và sai số(giá trị thực _ giá trị dự đoán)

```
In [57]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,5))
checkResidual (predict_train, residual_train, ax=ax1, title='Tập dữ liệu Train')
checkResidual (predict_test, residual_test, ax=ax2, title='Tập dữ liệu Test')
```



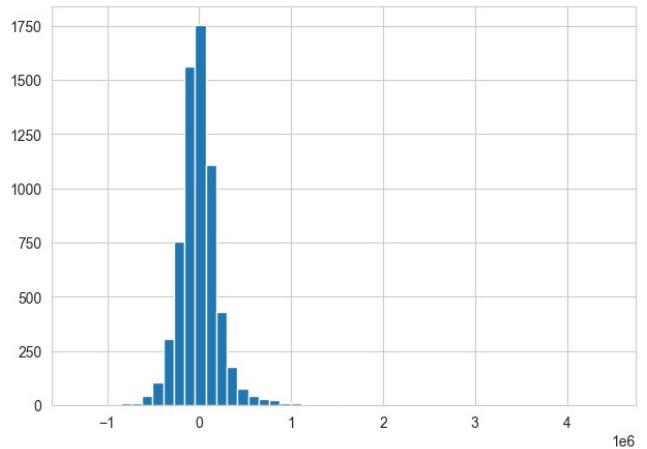
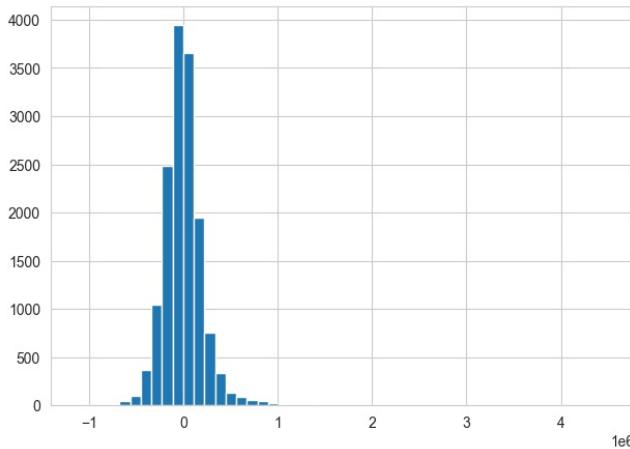
```
In [58]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,5))
ax1.plot(residual_train, 'b');
ax2.plot(residual_test, 'r');
```



Biểu đồ Sai số trên tập huấn luyện (màu xanh) và Biểu đồ Sai số trên tập kiểm tra (màu đỏ)

Các giá trị sai số được giữ lại ổn định gần giá trị 0 trên hầu hết các điểm dữ liệu. Điều này có thể cho thấy mô hình hoạt động tương đối tốt trên 2 tập dữ liệu. Tuy nhiên có một số giá trị sai số lớn hơn so với tập huấn luyện, đặc biệt là các giá trị âm và dương lớn. Nhìn chung, việc phân tích sai số trên cả hai tập dữ liệu giúp ta hiểu rõ hơn về hiệu suất của mô hình trên dữ liệu huấn luyện và dữ liệu mới.

```
In [59]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,5))
ax1.hist(residual_train, bins=50)
ax2.hist(residual_test, bins=50);
```



Chạy mô hình với kích thước tập train khác nhau

- Câu hỏi đặt ra là lựa chọn kích thước mẫu (tập train) như thế nào cho hợp lý?
- Chạy thử lần lượt với các kích thước khác nhau, ghi nhận kết quả

```
In [60]: train_size_name = ['TS60', 'TS70', 'TS80', 'TS90']
train_size_list = [0.6, 0.7, 0.8, 0.9] # tương ứng kích thước tập train 60%, 70%, 80%, 90%
```

```
In [61]: results = [regression(split_data(X, y, train_size)) for train_size in train_size_list]
```

So sánh các tham số đánh giá sai số

```
In [62]: params_list = [item['params'] for item in results]
params_df = pd.DataFrame(data=params_list, index=train_size_name)
params_df
```

	MAE	MSE	RMSE	R2_train	R2_test	R2_adj_train	R2_adj_test
TS60	155527.443399	5.439029e+10	233217.254199	0.599199	0.573006	0.598920	0.572709
TS70	155261.141055	5.453475e+10	233526.765106	0.600713	0.559287	0.600475	0.559025
TS80	155056.863483	5.207584e+10	228201.319507	0.597059	0.552764	0.596849	0.552531
TS90	157706.834421	5.487270e+10	234249.226992	0.590930	0.578407	0.590741	0.578212

Nhận xét

Khi thay đổi kích thước tập train / test, các tham số mô hình có thay đổi, nhưng rất nhỏ

Kiểm định mô hình hồi quy tuyến tính

Các giả thuyết của mô hình hồi quy:

$$Y_i = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \epsilon_i, i = 1, \dots, n$$

- Tính tuyến tính của dữ liệu: mỗi quan hệ giữa biến phụ thuộc
- Sai số có phân phối chuẩn
- Phương sai của các sai số là hằng số: $\epsilon_i \sim N(0, \sigma^2)$
- Các sai số $\epsilon_1, \dots, \epsilon_n$ thì độc lập với nhau

Biểu đồ #1 (Residuals vs Fitted):

Vẽ các giá trị dự đoán (\hat{y}_i) với các giá trị sai số dự đoán (Residuals: phần dư) tương ứng, dùng để kiểm tra tính tuyến tính của dữ liệu (giả thuyết 1) và tính đồng nhất của các phương sai sai số (giả thuyết 3). Nếu như giả thuyết về tính tuyến tính của dữ liệu KHÔNG thỏa, ta sẽ quan sát thấy rằng các điểm sai số (residuals) trên đồ thị sẽ phân bố theo một hình mẫu đặc trưng nào đó (ví dụ parabol). Nếu đường màu đỏ trên đồ thị phân tán là đường thẳng nằm ngang mà không phải là đường cong, thì giả thuyết tính tuyến tính của dữ liệu được thỏa mãn. Để kiểm tra giả thuyết thứ 3 (phương sai đồng nhất) thì các điểm sai số phải phân tán đều nhau xung quanh đường thẳng $y=0$.

Kiểm tra giả thuyết về phân phối chuẩn của các sai số. Nếu các điểm sai số nằm trên cùng 1 đường chéo thẳng thì điều kiện về phân phối chuẩn được thỏa.

Biểu đồ #3 (Scale - Location)

Vẽ căn bậc hai của các giá trị sai số (được chuẩn hóa) với các giá trị dự đoán, được dùng để kiểm tra giả thuyết thứ 3 (phương sai của các sai số là hằng số). Nếu như đường màu đỏ trên đồ thị là đường thẳng nằm ngang và các điểm sai số phân tán đều xung quanh đường thẳng này thì giả thuyết 3 được thỏa. Nếu như đường màu đỏ có độ dốc (hoặc cong) hoặc các điểm sai số phân tán không đều xung quanh đường thẳng này, thì giả định thứ 3 bị vi phạm.

Biểu đồ #4 (Residuals vs Leverage)

Cho phép xác định những điểm có ảnh hưởng cao, nếu chúng có hiện diện trong bộ dữ liệu. Những điểm có ảnh hưởng cao này có thể là các điểm outliers, là những điểm có thể gây nhiều ảnh hưởng nhất khi phân tích dữ liệu. Nếu như ta quan sát thấy một đường thẳng màu đỏ đứt nét (khoảng cách Cook's), và có một số điểm vượt qua đường thẳng khoảng cách này, nghĩa là các điểm đó là các điểm có ảnh hưởng cao. Nếu như ta chỉ quan sát thấy đường thẳng khoảng cách Cook ở góc của đồ thị và không có điểm nào vượt qua nó, nghĩa là không có điểm nào thực sự có ảnh hưởng cao.

Ta thực hiện phân tích sai số (phần dư) để kiểm tra các giả định của mô hình bằng 4 biểu đồ

```
In [63]: import statsmodels.api as sm
from scipy import stats

def graph(formula, x_range, label=None, ax=None):
    x = x_range
    y = formula(x)
    ax.plot(x, y, label=label, lw=1, ls='--', color='red')

def diagnostic_plots(x, y, model_fit=None):
    if not model_fit:
        model_fit = sm.OLS(y, sm.add_constant(x)).fit()

    dataframe = pd.concat([x, y], axis=1)
    model_fitted_y = model_fit.fittedvalues
    model_residuals = model_fit.resid
    model_norm_residuals = model_fit.get_influence().resid_studentized_internal
    model_norm_residuals_abs_sqrt = np.sqrt(np.abs(model_norm_residuals))
    model_abs_resid = np.abs(model_residuals)
    model_leverage = model_fit.get_influence().hat_matrix_diag
    model_cooks = model_fit.get_influence().cooks_distance[0]

    fig, axs = plt.subplots(4, 1, figsize=(20, 30))

    # Residuals vs Fitted plot
    sns.residplot(x=model_fitted_y, y=dataframe.columns[-1], data=dataframe, lowess=True,
                   scatter_kws={'alpha': 0.5}, line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8}, ax=axs[0])
    axs[0].set_title('Sai số - Dự đoán (Residuals vs Fitted)')
    axs[0].set_xlabel('Giá trị Dự đoán')
    axs[0].set_ylabel('Sai số (phần dư)')

    # Normal Q-Q plot
    stats.probplot(model_norm_residuals, dist="norm", plot=axs[1])
    axs[1].set_title('Normal Q-Q')
    axs[1].set_xlabel('Theoretical Quantiles')
    axs[1].set_ylabel('Standardized Residuals')

    # Scale-Location plot
    axs[2].scatter(model_fitted_y, model_norm_residuals_abs_sqrt, alpha=0.5)
    sns.regplot(x=model_fitted_y, y=model_norm_residuals_abs_sqrt, scatter=False,
                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8}, ax=axs[2])
    axs[2].set_title('Scale-Location')
    axs[2].set_xlabel('Giá trị Dự đoán')
    axs[2].set_ylabel('$\sqrt{|Sai số chuẩn hóa|}$')

    # Residuals vs Leverage plot
    axs[3].scatter(model_leverage, model_norm_residuals, alpha=0.5)
    sns.regplot(x=model_leverage, y=model_norm_residuals, scatter=False, ci=False, lowess=True,
                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8}, ax=axs[3])
    axs[3].set_xlim(0, max(model_leverage)+0.01)
    axs[3].set_ylim(-3, 5)
    axs[3].set_title('Residuals vs Leverage')
    axs[3].set_xlabel('Leverage')
    axs[3].set_ylabel('Sai số chuẩn hóa')

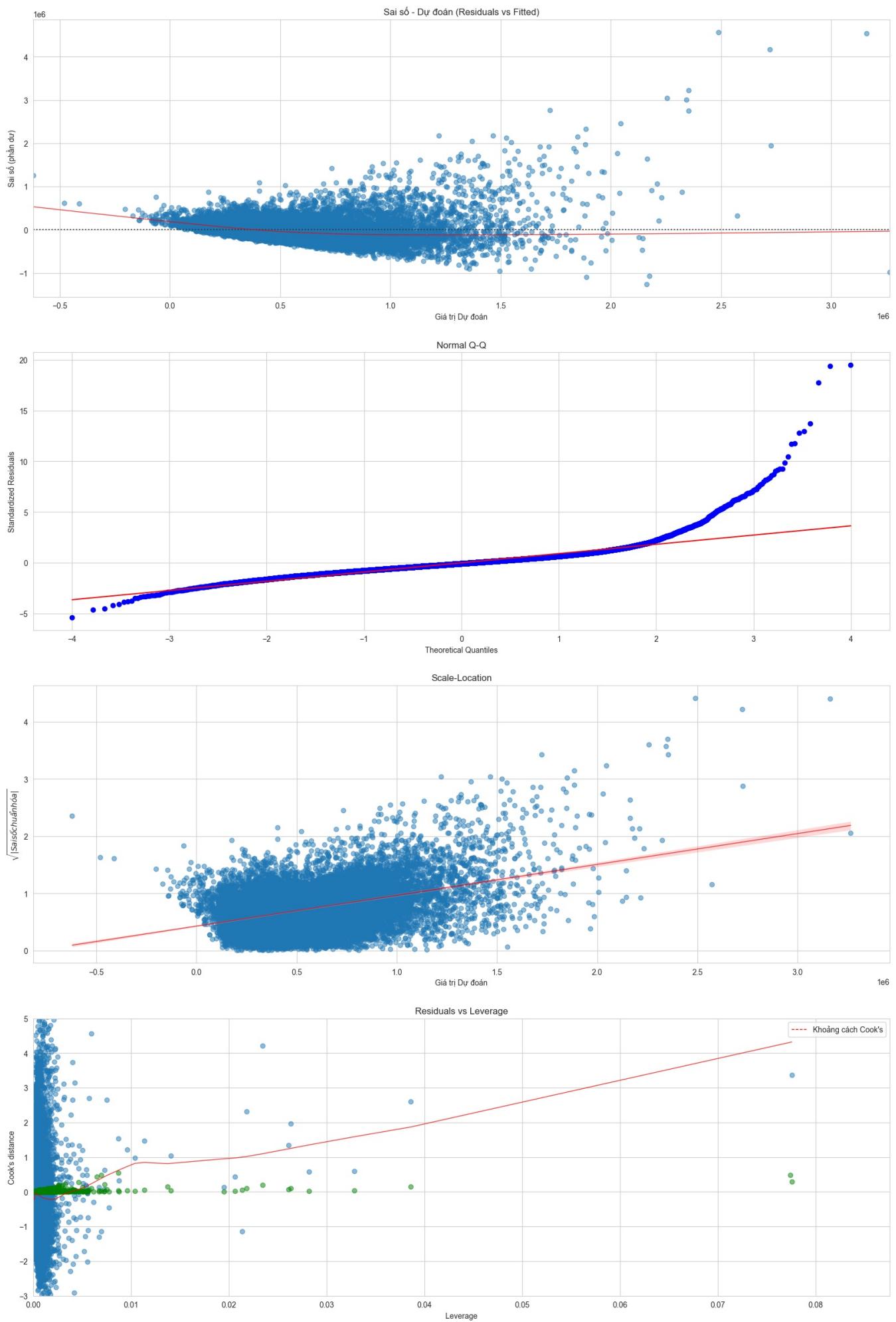
    # Tính Cook's distance
    model_cooks = model_fit.get_influence().cooks_distance[0]
```

```
# Vẽ Cook's distance trên cùng biểu đồ
axs[3].scatter(model_leverage, np.sqrt(model_cooks), alpha=0.5, color='green')
axs[3].set_ylabel('Cook's distance')

p = len(model_fit.params) # number of model parameters
graph(lambda x: np.sqrt((0.5 * p * (1 - x)) / x),
      np.linspace(0.001,max(model_leverage), 50), 'Khoảng cách Cook\''s', ax=axs[3]) # 0.5 L
graph(lambda x: np.sqrt((1 * p * (1 - x)) / x),
      np.linspace(0.001, max(model_leverage), 50), ax=axs[3]) # 1 Line

axs[3].legend(loc='upper right')
plt.show()

# Example Usage:
# diagnostic_plots(X, y, model_fit)
# Xem các biểu đồ kiểm định
diagnostic_plots(X, y)
```



Nhận xét về kiểm định mô hình hồi quy tuyến tính

- Biểu đồ 1 cho thấy đường màu đỏ trên đồ thị phân tán không phải là đường thẳng nằm ngang mà hơi cong \Rightarrow giả thuyết 1 không đúng \Rightarrow dữ liệu không tuyến tính

- Biểu đồ 2 cho thấy các điểm sai số (phần dư) không nằm nằm trên 1 đường chéo thẳng => sai số không có dạng phân phối chuẩn => giả thuyết #2 chưa đúng
- Biểu đồ 3 đường màu đỏ là đường thẳng nằm ngang và các điểm sai số phân tán đều xung quanh đường thẳng này => giả thuyết Phương sai của các sai số là hằng số đã đúng
- Biểu đồ 4 cho thấy đường thẳng khoảng cách Cook ở góc của đồ thị và không có điểm nào vượt qua nó, nghĩa là không có điểm nào thực sự có ảnh hưởng cao.

Tóm lại: mức độ phù hợp mô hình hồi quy tuyến tính này không cao đối với bộ dữ liệu khảo sát.

Khảo sát mô hình hồi quy khác

Mô hình hồi quy Ridge và Lasso

Ridge và Lasso là hai mô hình hồi qui áp dụng kỹ thuật hiệu chuẩn (regularization) để tránh hiện tượng quá khớp (overfitting).

- Quá khớp là hiện tượng mà mô hình chỉ khớp tốt trên tập dữ liệu huấn luyện nhưng không dự báo tốt trên dữ liệu kiểm tra. Đây là trường hợp thường gặp khi huấn luyện các mô hình machine learning. Hiện tượng này gây ảnh hưởng xấu và dẫn tới mô hình không thể áp dụng được vì các dự báo bị sai khi áp dụng vào thực tiễn. Có nhiều nguyên nhân dẫn tới quá khớp. Một trong những nguyên nhân phổ biến đó là tập dữ liệu huấn luyện và dữ liệu dự báo có phân phối khác xa nhau dẫn tới các qui luật học được ở dữ liệu huấn luyện không còn đúng trên dữ liệu dự báo. Hoặc cũng có thể xuất phát từ phía mô hình quá nhiều tham số nên khả năng biểu diễn dữ liệu của nó không mang tính đại diện.
- Regularization là kỹ thuật tránh overfitting bằng cách cộng thêm vào loss function thành phần hiệu chuẩn.
- Đối với trường hợp bậc 1 gọi là Lasso regression
- Thông thường thành phần này ở dạng norm chuẩn bậc 1 hoặc 2 của các hệ số.

$$L(w) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \alpha \|w\|_2^2$$

- Trong trường hợp bậc 2 ta gọi là Ridge regression.

$$L(w) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \alpha \|w\|_2^2$$

- Đối với những hồi qui này thì chúng ta cần tinh chỉnh hệ số α để tìm ra một hệ số là tốt nhất với từng bộ dữ liệu.
- Trong trường hợp dữ liệu bị quá khớp nặng thì cần giảm quá khớp bằng cách tăng ảnh hưởng của thành phần điều khiển (regularization term) thông qua tăng hệ số α .
- Nếu mô hình không bị quá khớp thì có thể lựa chọn α gần 0.
- Trường hợp $\alpha = 0$ thì phương trình hồi qui tương đương với hồi qui tuyến tính đa biến.
- Tuning hệ số alpha: Để lựa chọn ra một hệ số alpha phù hợp với mô hình Ridge/Lasso chúng ta sẽ cần phải tạo ra một list các giá trị có thể của tham số này và dùng vòng lặp for để đánh giá mô hình với trên từng giá trị của tham số. Giá trị được lựa chọn là giá trị mà có MSE trên tập kiểm tra là nhỏ nhất.

```
In [64]: from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neighbors import KNeighborsRegressor

# Chạy với các model khác nhau
linear_models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression A1': Ridge(alpha=1),
    'Ridge Regression A100': Ridge(alpha=100),
    'Ridge Regression A1000': Ridge(alpha=1000),
    'Lasso Regression A1': Lasso(alpha=1),
    'Lasso Regression A100': Lasso(alpha=100),
    'Lasso Regression A1000': Lasso(alpha=1000)
}
```

```
In [65]: results = [regression(split_data(X, y), model) for model in linear_models.values()]
params_list = [item['params'] for item in results]
params_df = pd.DataFrame(data=params_list, index=linear_models.keys())
params_df
```

Out[65]:

	MAE	MSE	RMSE	R2_train	R2_test	R2_adj_train	R2_adj_test
Linear Regression	155527.443399	5.439029e+10	233217.254199	0.599199	0.573006	0.598920	0.572709
Ridge Regression A1	155526.680064	5.439012e+10	233216.898716	0.599199	0.573007	0.598920	0.572710
Ridge Regression A100	155456.148456	5.437640e+10	233187.475325	0.599181	0.573115	0.598903	0.572818
Ridge Regression A1000	155236.574310	5.443767e+10	233318.819798	0.597827	0.572634	0.597548	0.572337
Lasso Regression A1	155527.192162	5.439022e+10	233217.101806	0.599199	0.573006	0.598920	0.572710
Lasso Regression A100	155502.765649	5.438332e+10	233202.321587	0.599198	0.573060	0.598919	0.572764
Lasso Regression A1000	155308.339559	5.433366e+10	233095.822629	0.599097	0.573450	0.598818	0.573154

Mô hình hồi quy đa thức Polynomial

Trong trường hợp các điểm dữ liệu không phù hợp với hồi quy tuyến tính Linear Regression (Các điểm không phân bố dưới dạng đường thẳng). Các điểm phân bố, phân tán dưới dạng đường cong, do đó thuật toán hồi quy tuyến tính Linear Regression không phù hợp vì vậy bạn phải cần sử dụng Polynomial Regression để tối ưu hơn. Polynomial Regression gần giống như thuật toán hồi quy tuyến tính, sử dụng mối quan hệ giữa các biến độc lập x và biến phụ thuộc y được biểu diễn dưới dạng đa thức bậc n, để tìm cách tốt nhất về một đường qua các điểm dữ liệu sao cho tối ưu và phù hợp nhất.

$$y = f(x_1, \dots, x_n) = b_0 + b_1 x_1 + \dots + b_n x_n$$

Ưu điểm của việc sử dụng hồi quy đa thức

- Phạm vi rộng của hàm có thể được phù hợp với nó.
- Đa thức về cơ bản phù hợp với nhiều độ cong.
- Đa thức cung cấp giá trị gần đúng nhất của mối quan hệ giữa biến phụ thuộc và biến độc lập.

Nhược điểm của việc sử dụng hồi quy đa thức

- Rất nhạy cảm với các yếu tố ngoại lai.
- Sự hiện diện của một hoặc hai điểm ngoại lai trong dữ liệu có thể ảnh hưởng nghiêm trọng đến kết quả của một phân tích phi tuyến.
- ít công cụ xác thực mô hình để phát hiện các giá trị ngoại lệ trong hồi quy phi tuyến so với hồi quy tuyến tính.

In [66]:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import pandas as pd

def polynomial(data, model=LinearRegression(), degree=2):
    poly = PolynomialFeatures(degree=degree)
    data['X_train'] = poly.fit_transform(data['X_train'])
    data['X_test'] = poly.fit_transform(data['X_test'])
    return regression(data, model)
```

In [67]:

```
results = [polynomial(split_data(X, y, train_size=0.8), model) for model in linear_models.values()]
params_list = [item['params'] for item in results]
params_df = pd.DataFrame(data=params_list, index=linear_models.keys())
params_df
```

Out[67]:

	MAE	MSE	RMSE	R2_train	R2_test	R2_adj_train	R2_adj_test
Linear Regression	146632.458454	7.008685e+10	264739.203603	0.689804	0.398083	0.688633	0.395811
Ridge Regression A1	146595.595714	6.980457e+10	264205.553494	0.689800	0.400507	0.688629	0.398244
Ridge Regression A100	146206.055518	6.392423e+10	252832.419615	0.688368	0.451008	0.687192	0.448936
Ridge Regression A1000	146116.614540	6.054146e+10	246051.735064	0.686829	0.480060	0.685647	0.478098
Lasso Regression A1	146784.314715	7.267019e+10	269574.093866	0.689398	0.375896	0.688225	0.373541
Lasso Regression A100	146511.779044	7.004806e+10	264665.946044	0.689332	0.398416	0.688160	0.396145
Lasso Regression A1000	146085.229076	6.156495e+10	248122.846422	0.686187	0.471270	0.685003	0.469274

Đánh giá

Mô hình hồi quy đa thức cải thiện được độ chính xác của dự đoán, cho ra hệ số xác định R khá tốt

Mô hình hồi quy K-Neighbors

Mô hình KNN (K-Nearest Neighbors), một phương pháp học máy được sử dụng cho các bài toán phân loại và hồi quy

- KNN là một thuật toán học máy không cần huấn luyện.
- Ý tưởng chính của KNN là dự đoán nhãn của một điểm dữ liệu mới dựa trên nhãn của các điểm dữ liệu gần nhất trong tập dữ liệu huấn luyện.
- KNN hoạt động dựa trên nguyên lý "gần càng tốt". Nó tính toán khoảng cách giữa điểm dữ liệu mới và tất cả các điểm dữ liệu trong tập huấn luyện, sau đó chọn ra k điểm gần nhất (k là một siêu tham số được chọn trước), và dự đoán nhãn của điểm dữ liệu mới dựa trên phần đông của nhãn trong k điểm gần nhất.
- KNN thường được sử dụng cho các bài toán phân loại.

```
In [68]: knn_models = {
    'K-Neighbors Regressor K15': KNeighborsRegressor(n_neighbors=15),
    'K-Neighbors Regressor K25': KNeighborsRegressor(n_neighbors=25),
    'K-Neighbors Regressor K27': KNeighborsRegressor(n_neighbors=27)
}

results = [regression(split_data(X, y), model) for model in knn_models.values()]
params_list = [item['params'] for item in results]
params_df = pd.DataFrame(data=params_list, index=knn_models.keys())
params_df
```

	MAE	MSE	RMSE	R2_train	R2_test	R2_adj_train	R2_adj_test
K-Neighbors Regressor K15	159361.215386	6.214354e+10	249286.058426	0.583245	0.512138	0.582955	0.511799
K-Neighbors Regressor K25	158422.017760	6.261161e+10	250223.111936	0.547056	0.508464	0.546741	0.508122
K-Neighbors Regressor K27	158180.167961	6.274235e+10	250484.223968	0.540792	0.507437	0.540473	0.507095

Đánh giá

Với bộ dữ liệu khảo sát mô hình hồi quy K-Neighbors không cải thiện gì mấy so với hồi quy tuyến tính

Hồi quy cây quyết định (Decision Tree Regressor)

Hồi quy cây quyết định quan sát các đặc điểm của một đối tượng và huấn luyện một mô hình theo cấu trúc của cây để dự đoán dữ liệu

```
In [69]: from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score

X = df.iloc[:, :-1] # Lấy n-1 cột đầu tiên
y = df.iloc[:, -1:] # Lấy cột cuối cùng
print(X.shape, y.shape)

model_data = split_data(X, y, train_size=0.7)
dtModel = DecisionTreeRegressor(random_state=0)
dtModel.fit(model_data['X_train'], model_data['y_train'])

predict_train = dtModel.predict(model_data['X_train']).astype(int).reshape(-1, 1)
predict_test = dtModel.predict(model_data['X_test']).astype(int).reshape(-1, 1)

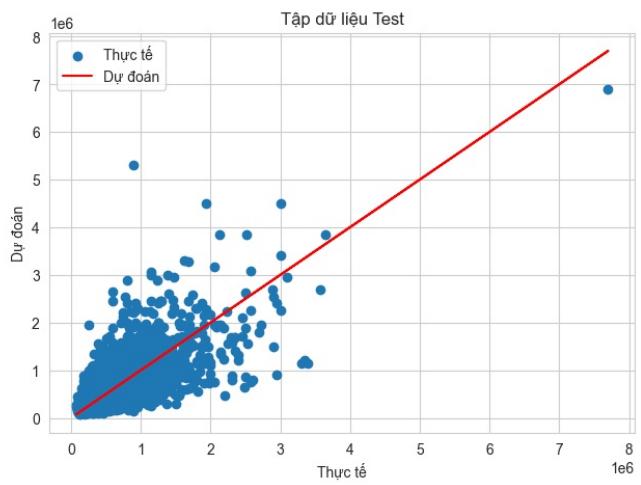
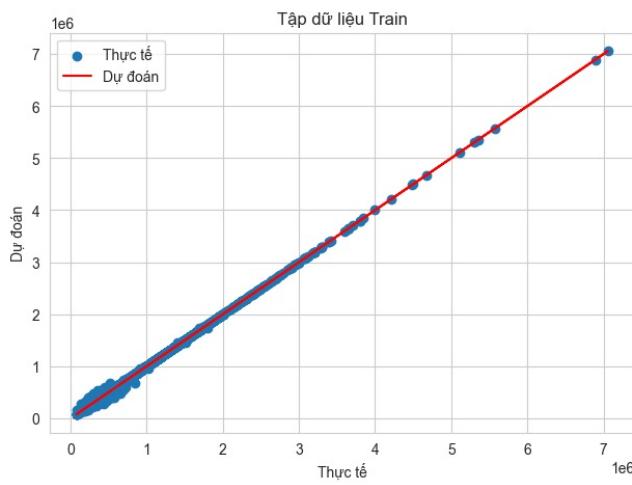
residual_train = (model_data['y_train'] - predict_train).reset_index(drop=True)
residual_test = (model_data['y_test'] - predict_test).reset_index(drop=True)

print(f'R2 tập train: {r2_score(model_data['y_train'], predict_train) * 100}')
print(f'R2 tập test: {r2_score(model_data['y_test'], predict_test) * 100:.2f}')

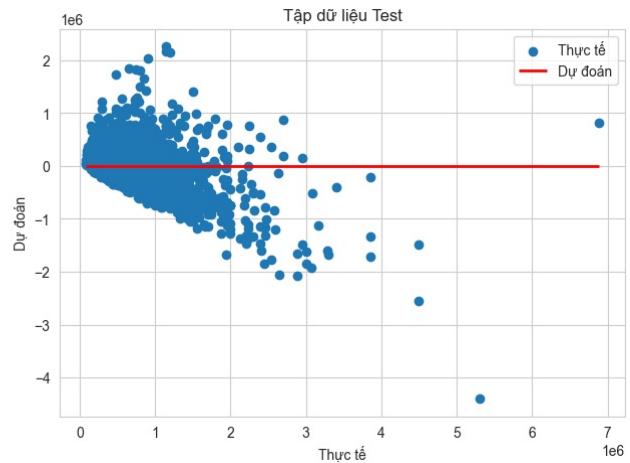
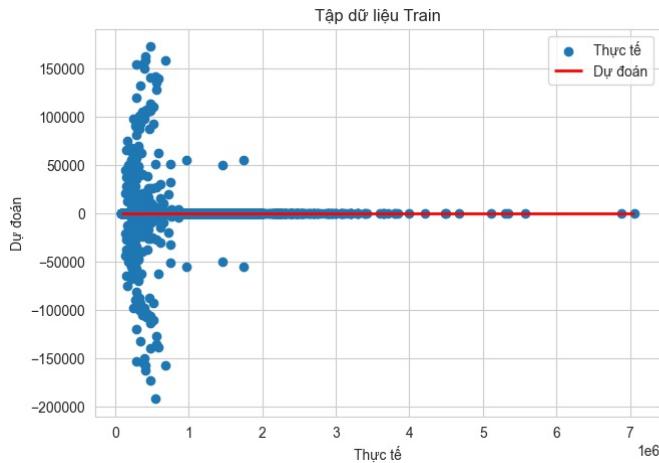
(21610, 10) (21610, 1)
R2 tập train: 99.94315968590664
R2 tập test: 33.45
```

```
In [70]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 5))

checkPredict(model_data['y_train'].to_numpy(), predict_train, ax=ax1, title='Tập dữ liệu Train')
checkPredict(model_data['y_test'].to_numpy(), predict_test, ax=ax2, title='Tập dữ liệu Test')
```



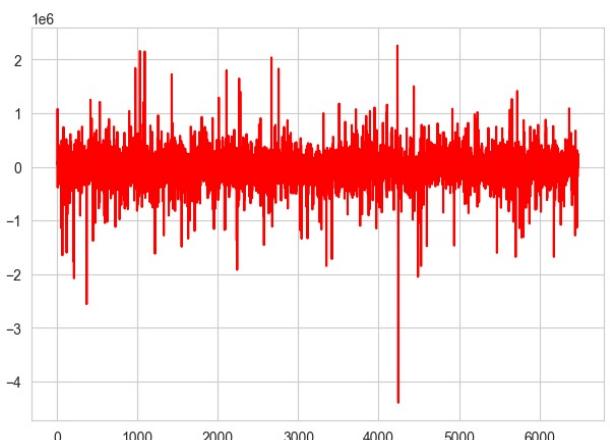
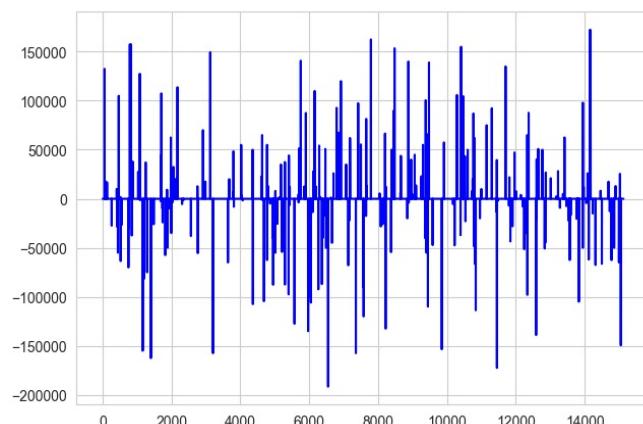
```
In [71]: # Subplot 1 - Residuals for Train Data and Test Data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 5))
checkResidual(predict_train, residual_train, ax=ax1, title='Tập dữ liệu Train')
checkResidual(predict_test, residual_test, ax=ax2, title='Tập dữ liệu Test')
```



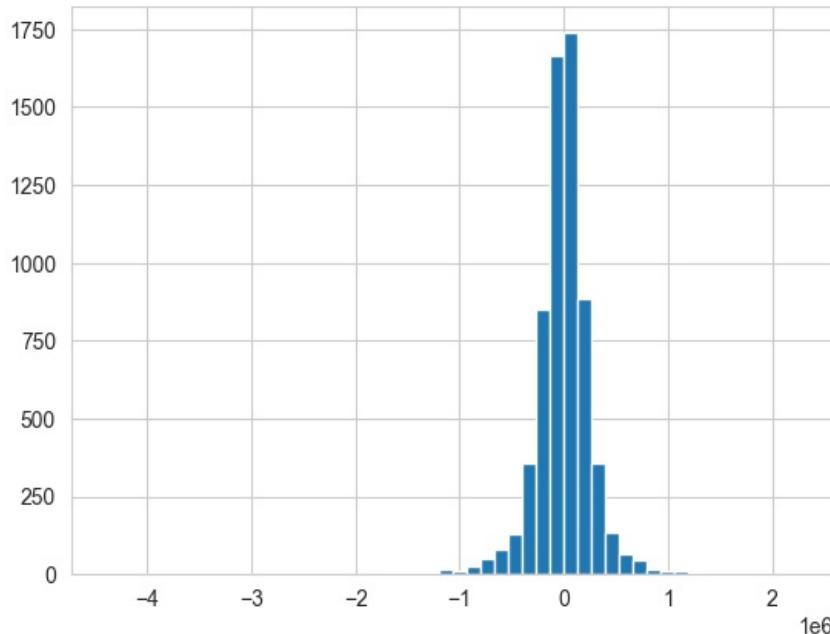
```
In [72]: # Subplot 2 - Residuals Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 5))
ax1.plot(residual_train, 'b')
ax2.plot(residual_test, 'r')
```

```
Out[72]: [

```



```
In [73]: plt.hist(residual_test, bins=50);
```



Đánh giá

- Hệ số xác định R (R-squared): Mô hình hồi quy cây quyết định đã cho ra hệ số xác định R khá cao trên tập huấn luyện, đạt khoảng 99.94%. Điều này cho thấy mô hình có khả năng giải thích được phần lớn sự biến thiên của biến phụ thuộc bằng các biến độc lập trong tập dữ liệu huấn luyện.
- Overfitting đối với tập dữ liệu huấn luyện: Tuy mô hình đạt hiệu suất cao trên tập huấn luyện, nhưng hiệu suất trên tập kiểm tra lại thấp hơn đáng kể, chỉ đạt khoảng 33.45%. Sự chênh lệch đáng kể giữa hai kết quả này là dấu hiệu rõ ràng của hiện tượng overfitting. Mô hình đã học quá mức từ dữ liệu huấn luyện, không tổng quát hóa được cho dữ liệu mới.
- Sai số giảm đáng kể: Mặc dù mô hình có hiện tượng overfitting, sai số của nó vẫn giảm đáng kể so với một mô hình cơ sở đơn giản như dự đoán trung bình. Điều này cho thấy mô hình có khả năng học được mối quan hệ giữa các biến đầu vào và biến mục tiêu, tuy nhiên, việc ứng dụng mô hình này trên dữ liệu mới ngoài tập huấn luyện cần được thận trọng.

Tóm lại, mặc dù mô hình hồi quy cây quyết định có hệ số xác định R tốt trên tập huấn luyện và giảm sai số đáng kể, tuy nhiên, hiện tượng overfitting là một vấn đề đáng chú ý cần được giải quyết để cải thiện khả năng tổng quát hóa của mô hình.

XGBOOST

Mô hình XGBoost (eXtreme Gradient Boosting) là một thuật toán học máy rất mạnh mẽ và phổ biến trong các cuộc thi khoa học dữ liệu và ứng dụng thực tế. Đây là một thuật toán boosting ensemble, nghĩa là nó xây dựng một chuỗi các cây quyết định (decision trees) nhằm cải thiện hiệu suất dự đoán.

- XGBoost là một thuật toán học máy thuộc loại Gradient Boosting.
- Ý tưởng chính của XGBoost là xây dựng một loạt các cây quyết định theo cách tuần tự, mỗi cây đều cố gắng cải thiện việc dự đoán của cây trước đó.
- XGBoost sử dụng hàm mất mát (loss function) để đánh giá hiệu suất của mô hình và cố gắng tối ưu hóa hàm mất mát này bằng cách điều chỉnh các cây quyết định.
- XGBoost được sử dụng cho cả bài toán phân loại và hồi quy, và thường được coi là một trong những thuật toán hiệu quả và mạnh mẽ nhất trong học máy.
- Nó cung cấp một số tính năng như tối ưu hóa song song và khả năng xử lý dữ liệu lớn.

```
In [74]: from xgboost import XGBRegressor
xgb = XGBRegressor(n_estimators= 2000 , max_depth= 7 , learning_rate = 0.01)
```

```
In [75]: model_data = split_data(X,y,0.75)
```

```
In [76]: X_trains = model_data['X_train']
y_trains = model_data['y_train']
X_tests = model_data['X_test']
y_tests = model_data['y_test']
```

```
In [77]: xgb.fit(X_trains , y_trains);
```

```
In [78]: y_xgb_pred=xgb.predict(X_tests)
```

```
In [79]: print ("train accuracy",xgb.score(X_trains , y_trains))
print ("test accuracy",xgb.score(X_tests , y_tests))

train accuracy 0.8901349614009839
test accuracy 0.610426357952683
```

```
In [80]: # r2_score
r2=r2_score(y_tests , y_xgb_pred)
print("Test Accuracy:", round(r2, 4))

Test Accuracy: 0.6104
```

```
In [81]: # mean_absolute_error
mae=mean_absolute_error(y_tests , y_xgb_pred)
print("Test Accuracy:", round(mae, 4))

Test Accuracy: 133951.4848
```

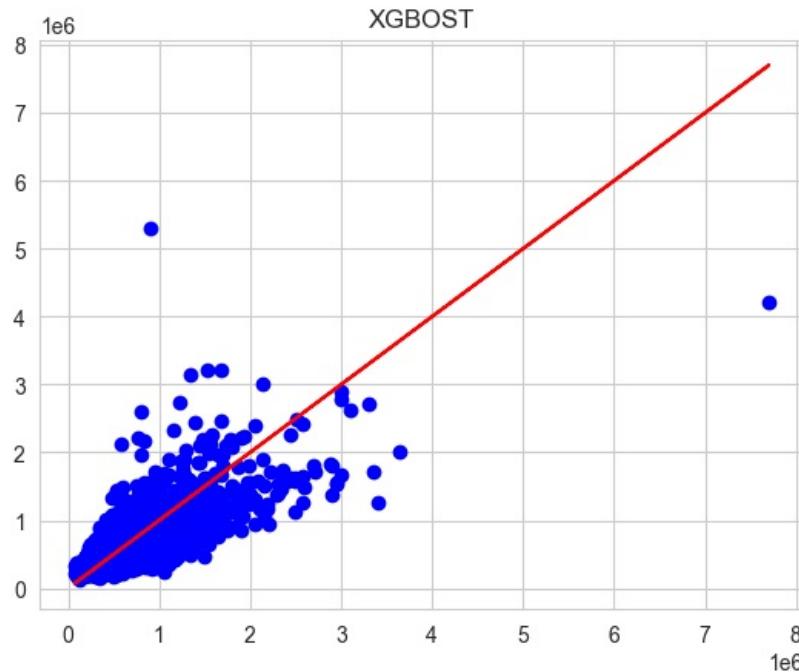
```
In [82]: # cross_val_score
cv=cross_val_score(xgb, X, y, cv=10)
cv
```

```
Out[82]: array([0.66690533, 0.59588728, 0.67627962, 0.66751707, 0.64587566,
       0.66201316, 0.64786616, 0.69765389, 0.71560663, 0.63365746])
```

```
In [83]: print("the mean of cross_val_score is ",cv.mean())

the mean of cross_val_score is  0.6609262245448949
```

```
In [84]: # XGB graph
plt.scatter(y_tests,y_xgb_pred, color="b")
plt.plot(y_tests,y_tests, color="r")
plt.title("XGBOST");
```



Mô hình XGBoost cho thấy hiệu suất khá tốt trên cả tập huấn luyện và tập kiểm tra, với điểm số R-squared cao và sai số thấp. Điều này cho thấy XGBoost là một lựa chọn mạnh mẽ và phù hợp cho bài toán.

Processing math: 90%