TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN
NIÊN HỌC 2022 – 2026

PROJECT PHÂN TÍCH DỮ LIỆU THÔNG MINH

PREDICTION INTERVAL COMPETITION II: HOUSE PRICE

Nhóm thực hiện dự án:
Lê Đại Hòa – 22120108
Lê Châu Hữu Thọ – 22120350
Nguyễn Tường Bách Hỷ – 22120455
Lê Hoàng Vũ – 22120461

Giáo viên hướng dẫn:
TS. Nguyễn Tiến Huy
TS. Lê Thanh Tùng
ThS. Nguyễn Trần Duy Minh

Thành phố Hồ Chí Minh, ngày 17 tháng 07 năm 2025

Mục lục

I

TÓM TẮT

1 Tóm tắt nội dung báo cáo

Kính gửi Quý Thầy/Cô,

Nhóm chúng em xin phép được trình bày báo cáo về dự án cuối kỳ môn học **Phân tích dữ liệu thông minh** với nội dung: **"Dự đoán khoảng giá nhà bằng mô hình học máy và phương pháp Conformal Prediction"**. Nội dung của báo cáo phản ánh toàn bộ quá trình phân tích, khám phá, xử lý dữ liệu, huấn luyện mô hình và tối ưu hóa dự đoán trên tập dữ liệu thực tế. Cụ thể, nhóm đã thực hiện:

(i) Thực hiện **phân tích khám phá dữ liệu (EDA)**: biểu đồ phân phối, biểu đồ tần suất, kiểm tra dữ liệu thiếu, kiểu dữ liệu, và các nhận xét sơ bộ giúp định hướng xử lý;

(ii) Tiến hành **xử lý dữ liệu đầu vào**: mã hóa biến phân loại, chuẩn hóa dữ liệu, trích xuất đặc trưng thời gian, xử lý dữ liệu thiếu và tách tập `train-validation`;

(iii) Xây dựng và đánh giá **mô hình dự báo khoảng giá**: thử nghiệm các mô hình hồi quy lượng tử (`quantile regression`), áp dụng phương pháp `Conformal Prediction` để hiệu chỉnh khoảng dự đoán và đánh giá bằng chỉ số `Winkler Score`.

Tóm tắt báo cáo

Báo cáo tập trung xây dựng một hệ thống dự báo khoảng giá bất động sản, thay vì dự đoán giá trị đơn lẻ. Hệ thống tiếp nhận dữ liệu giao dịch thực tế (gồm thông tin vị trí, diện tích, tiện nghi, tầm nhìn, năm xây dựng...), sau đó trích xuất đặc trưng, chuẩn hóa và huấn luyện mô hình học máy để đưa ra hai đầu mút: `giới hạn dưới` và `giới hạn trên` của giá nhà.

Ở phần `(iii)` – Xây dựng mô hình dự đoán, nhóm đã triển khai nhiều phương án, trong đó bao gồm:

(i) `LightGBM` với hàm mất mát lượng tử (`Quantile Regression`);

(ii) Mạng `neuron` đơn giản kết hợp với hàm mất mát `Quantile Loss`;

(iii) Kỹ thuật `Conformal Prediction` nhằm điều chỉnh khoảng dự đoán theo độ tin cậy 90%–95%.

Tập dữ liệu có quy mô lớn (200,000 dòng), nhiều biến dạng số, phân loại, nhị phân và thời gian. Nhóm đã áp dụng quy trình xử lý dữ liệu chuyên nghiệp, từ `EDA`, chuẩn hóa, trích xuất đặc trưng thời gian, đến xây dựng pipeline huấn luyện. Kết quả trên tập test cho thấy hệ thống đạt độ chính xác tốt, khoảng dự đoán hẹp và có khả năng tổng quát cao.

Tất cả mã nguồn, dữ liệu thử nghiệm, và slide trình bày đều đã được chuẩn bị đầy đủ nhằm hỗ trợ quá trình đánh giá, chấm điểm. Quý Thầy/Cô có thể truy cập tài nguyên báo cáo tại đây

Nhóm xin chân thành cảm ơn sự quan tâm và đánh giá từ Quý Thầy/Cô!

Quy trình	Nội dung	Mức độ hoàn thành
Lên kế hoạch tầm nhìn dự án		
Phân tích yêu cầu và quản lý nhóm	[Lê Đại Hòa] Khảo sát và phân tích yêu cầu bài toán từ đề bài cuộc thi; chủ trì các cuộc họp dự án trong nhóm	Tốt (100%)
Chuẩn bị dữ liệu cho việc huấn luyện mô hình		
Làm EDA	[Lê Châu Hữu Thọ] Thống kê, phân tích đơn biến, đa biến và trực quan hóa dữ liệu bất động sản	Tốt (100%)
Làm xử lý dữ liệu	[Lê Châu Hữu Thọ, Nguyễn Tường Bách Hỷ] Chuẩn hóa, xử lý thiếu, tạo đặc trưng thời gian, chia dữ liệu `train/val/test`, chuẩn hóa dữ liệu	Tốt (100%)
Huấn luyện mô hình dự đoán		
Cài đặt pipeline mô hình	[Lê Hoàng Vũ] Triển khai pipeline huấn luyện mô hình `LightGBM` và `Neural Network`	Tốt (100%)
Tích hợp Conformal Prediction	[Nguyễn Tường Bách Hỷ] Áp dụng kỹ thuật `Conformal Prediction` để điều chỉnh khoảng dự báo với mức tin cậy cao	Tốt (100%)
Đánh giá và lưu mô hình	[Lê Hoàng Vũ, Lê Châu Hữu Thọ] Lưu mô hình, đóng gói dữ liệu đầu ra, đánh giá hiệu suất qua `Winkler Score`	Tốt (100%)
So sánh mô hình	[Nguyễn Tường Bách Hỷ] So sánh độ rộng khoảng dự đoán và độ ổn định giữa các mô hình đã triển khai	Tốt (100%)
Resource		
`Slide` trình bày	[Lê Đại Hòa] Thiết kế `slide` tổng quan toàn bộ dự án	Tốt (100%)
Báo cáo dự án	[Lê Hoàng Vũ] Soạn thảo báo cáo chi tiết, đúng định dạng yêu cầu	Tốt (100%)
`Video demo`	[Lê Đại Hòa] Dựng `video demo` hệ thống dự đoán, đánh giá mô hình	Tốt (100%)

Lời kết

Nhóm đã nỗ lực trình bày báo cáo một cách ngắn gọn, `logic` và đầy đủ các phần yêu cầu. Nếu có thiếu sót, nhóm rất mong nhận được góp ý từ Thầy/Cô để có thể hoàn thiện tốt hơn ở những lần sau.

Trân trọng cảm ơn Quý Thầy/Cô đã hướng dẫn và hỗ trợ nhóm trong quá trình thực hiện đồ án.

II

NỘI DUNG CHÍNH

2 Exploratory Data Analysis (EDA)

§2.1 Tổng quan dữ liệu

§2.1.1 Thông tin và kích thước

Dữ liệu được cung cấp từ cuộc thi **Prediction Interval Competition II - House Price** trên nền tảng `Kaggle`, bao gồm hai tập:

- `dataset.csv`: Tập huấn luyện (train)
- `test.csv`: Tập kiểm tra (test)

Sau khi đọc dữ liệu, ta có kích thước tập huấn luyện và kiểm tra như sau:

```
Kích thước dataset: (200000, 47)
Kích thước test: (200000, 46)
```

Thông tin chi tiết lấy được từ các cột trong `dataset`:

```
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 47 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   id              200000 non-null  int64
 1   sale_date       200000 non-null  object
 2   sale_price      200000 non-null  int64
 3   sale_nbr        157818 non-null  float64
 4   sale_warning    200000 non-null  object
 5   join_status     200000 non-null  object
 6   join_year       200000 non-null  int64
...
44  view_otherwater 200000 non-null  int64
45  view_other      200000 non-null  int64
46  submarket       198283 non-null  object
dtypes: float64(4), int64(36), object(7)
memory usage: 71.7+ MB
```

§2.1.2 Mô tả các nhóm thuộc tính trong dữ liệu

Tập dữ liệu bao gồm thông tin chi tiết về các giao dịch bất động sản, được phân chia thành 4 nhóm chính:

1. Nhóm định danh và thông tin giao dịch

- `id`: Mã định danh duy nhất cho mỗi bản ghi hoặc tài sản.
- `sale_date`: Ngày diễn ra giao dịch bán hàng.
- `sale_price`: Giá bán của tài sản.
- `sale_nbr`: Số của giao dịch bán hàng.
- `sale_warning`: Cảnh báo liên quan đến giao dịch bán hàng (ví dụ: các vấn đề hoặc ghi chú đặc biệt).
- `join_status`: Trạng thái tham gia hoặc liên kết của bản ghi.
- `join_year`: Năm tham gia hoặc liên kết.

2. Nhóm vị trí và khu vực

- `latitude`: Vĩ độ của tài sản.
- `longitude`: Kinh độ của tài sản.
- `area`: Mã hoặc chỉ số của khu vực.
- `city`: Thành phố nơi tài sản tọa lạc.
- `zoning`: Phân loại quy hoạch đất đai (ví dụ: khu dân cư, thương mại).
- `subdivision`: Tên khu phân lô hoặc khu dân cư phụ.
- `submarket`: Tiểu thị trường mà tài sản thuộc về.

3. Nhóm đặc điểm tài sản và xây dựng

- `present_use`: Mục đích sử dụng hiện tại của tài sản.
- `land_val`: Giá trị đất đai.
- `imp_val`: Giá trị cải tiến của tài sản (ví dụ: nhà cửa, công trình xây dựng).
- `year_built`: Năm tài sản được xây dựng.
- `year_reno`: Năm tài sản được cải tạo hoặc sửa chữa lần cuối. Giá trị 0 có thể chỉ ra rằng không có cải tạo nào được thực hiện.
- `sqft_lot`: Diện tích lô đất tính bằng feet vuông.
- `sqft`: Tổng diện tích xây dựng của tài sản tính bằng feet vuông.
- `sqft_1`: Diện tích tầng 1 tính bằng feet vuông.
- `sqft_fbsmt`: Diện tích tầng hầm hoàn chỉnh (`fbsmt = finished basement`) tính bằng `feet` vuông.
- `grade`: Cấp độ hoặc chất lượng tổng thể của tài sản (ví dụ: từ 1–13, trong đó 7 là mức trung bình).
- `fbsmt_grade`: Cấp độ hoặc chất lượng của tầng hầm hoàn chỉnh.
- `condition`: Tình trạng chung của tài sản (ví dụ: từ 1–5, trong đó 3 là mức trung bình).
- `stories`: Số tầng của tài sản.
- `beds`: Số phòng ngủ.
- `bath_full`: Số phòng tắm đầy đủ.
- `bath_3qtr`: Số phòng tắm 3/4.
- `bath_half`: Số phòng tắm 1/2.
- `garb_sqft`: Diện tích nhà để xe rời (detached garage) tính bằng feet vuông.
- `gara_sqft`: Diện tích nhà để xe gắn liền (attached garage) tính bằng feet vuông.

4. Nhóm tầm nhìn và tiện ích xung quanh

Các cột này thường là biến nhị phân (0 hoặc 1), cho biết liệu tài sản có tầm nhìn hoặc bị ảnh hưởng bởi yếu tố đó hay không:

- `wfnt`: Có tầm nhìn hoặc tiếp giáp mặt nước (`waterfront`).
- `golf`: Có tầm nhìn ra sân golf.
- `greenbelt`: Có tầm nhìn ra vành đai xanh (khu vực cây xanh bảo tồn).
- `noise_traffic`: Có bị ảnh hưởng bởi tiếng ồn giao thông.
- `view_rainier`: Có tầm nhìn ra Núi Rainier.
- `view_olympics`: Có tầm nhìn ra Dãy núi Olympic.
- `view_cascades`: Có tầm nhìn ra Dãy núi Cascades.
- `view_territorial`: Có tầm nhìn ra khu vực xung quanh (territorial view).
- `view_skyline`: Có tầm nhìn ra đường chân trời thành phố (skyline).
- `view_sound`: Có tầm nhìn ra Puget Sound.
- `view_lakewash`: Có tầm nhìn ra Hồ Washington.
- `view_lakesamm`: Có tầm nhìn ra Hồ Sammamish.
- `view_otherwater`: Có tầm nhìn ra nguồn nước khác (không phải hồ hoặc Puget Sound).
- `view_other`: Có tầm nhìn khác.

§2.1.3 Tỷ lệ thiếu dữ liệu

```python
print("\nThông tin về dữ liệu thiếu trong dataset:")
missing_dataset = train.isnull().sum()
print(missing_dataset[missing_dataset > 0])

print("\nThông tin về dữ liệu thiếu trong test:")
missing_test = test.isnull().sum()
print(missing_test[missing_test > 0])
```

Listing 2.1: Thống kê missing values

Kết quả thu được như sau:

```
Thông tin về dữ liệu thiếu trong train:
sale_nbr       42182
subdivision    17550
submarket       1717
dtype: int64

Thông tin về dữ liệu thiếu trong test:
sale_nbr       42412
subdivision    17550
submarket       1718
dtype: int64
```

Nhận xét (Về tỉ lệ thiếu của dữ liệu)

Ta nhận thấy một số vấn đề sau từ phân tích tỉ lệ `missing values` ở trên:

- Cả tập `train` và `test` đều có 3 cột chứa dữ liệu bị thiếu là: `sale_nbr`, `subdivision`, và `submarket`.
- Số lượng giá trị thiếu của các cột này gần như giống nhau giữa hai tập dữ liệu.
- Cần cân nhắc xử lý dữ liệu thiếu trước khi huấn luyện mô hình.

§2.2 Phân tích đơn biến (Univariate Analysis)

Phân tích đơn biến về cơ bản là hình thức đơn giản nhất để phân tích dữ liệu. Mục tiêu chính của phân tích đơn biến `non-graphical` là mô tả, tóm tắt dữ liệu và tìm ra `patterns` trong dữ liệu, thông qua việc tìm ra các chỉ số thống kê sau:

(i) **Mức độ tập trung của dữ liệu (Measure of Central Tendency):** Ba chỉ số quan trọng nhất để đánh giá mức độ tập trung của dữ liệu là `mean` (giá trị trung bình), `median` (giá trị trung vị), và `mode`.

(ii) `Khoảng dữ liệu (Measure of Range):` Khoảng dữ liệu là khoảng dao động của các giá trị trong tập dữ liệu. Các chỉ số quan trọng nhất cần được quan tâm là `min` (giá trị nhỏ nhất), `max` (giá trị lớn nhất), `quartiles` và `interquartile`.

(iii) **Phương sai (Variance) và Độ lệch chuẩn (Standard Deviation):** Phương sai (`Variance`) là thước đo mức độ phân tán của dữ liệu xung quanh giá trị trung tâm (`mean`) và được tính bằng bình phương trung bình giữa mỗi giá trị dữ liệu và giá trị trung bình. Độ lệch chuẩn (`Standard deviation`) cũng được tính toán để đo lường mức độ phân tán của các giá trị và có giá trị bằng căn bậc hai của `variance`. Giá trị của `standard deviation` càng lớn thì mức độ phân tán của dữ liệu càng rộng.

§2.2.1 Phân tích biến phân loại

```
for col in train.columns:
    if train[col].dtype == 'object':
        num_unique = train[col].nunique()
        print(f"- Cột '{col}': {num_unique} giá trị unique")
```

Listing 2.2: Kiểm tra các biến thuộc nhóm categor

```
- Cột 'sale_date': 313 giá trị unique
- Cột 'sale_warning': 142 giá trị unique
- Cột 'join_status': 8 giá trị unique
- Cột 'city': 41 giá trị unique
- Cột 'zoning': 500 giá trị unique
- Cột 'subdivision': 10376 giá trị unique
- Cột 'submarket': 19 giá trị unique
```

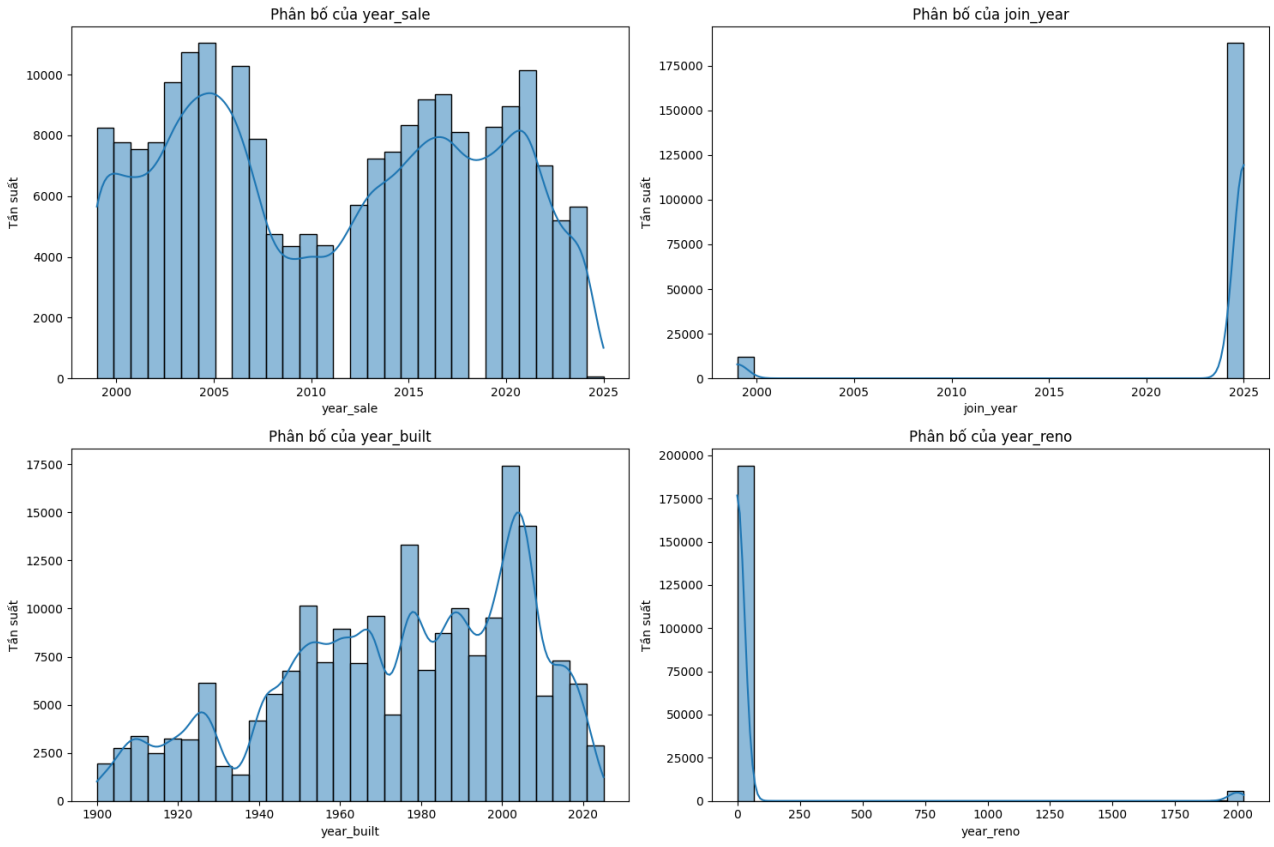
Nhận xét (Về dữ liệu thuộc nhóm category)

Ngoại trừ cột sale_date, các cột còn lại cho quá nhiều nhóm phân loại, nếu làm `One-Hot Encoding` sẽ làm dữ liệu bị phình to, nếu bỏ dữ liệu thì có khả năng mất thông tin quan trọng. Vì vậy nhóm đề xuất sử dụng `LabelEncoding` để xử lý

§2.2.2 Phân tích dữ liệu thuộc nhóm thời gian

```python
#Chuyển cột thời gian về đúng định dạng
train['sale_date'] = pd.to_datetime(train['sale_date'])
train['year_sale'] = train['sale_date'].dt.year

#Các cột năm cần quan sát
year_columns = ['year_sale', 'join_year', 'year_built', 'year_reno']

#Vẽ matplotlb
plt.figure(figsize=(15, 10))

for i, col in enumerate(year_columns):
    plt.subplot(2, 2, i + 1)
    sns.histplot(train[col].dropna(), kde=True, bins=30)
    plt.title(f'Phân bố của {col}')
    plt.xlabel(col)
    plt.ylabel('Tần suất')

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

Listing 2.3: Trích xuất và trực quan hoá các năm



Hình 2.1: Phân bố của các biến thời gian: `year_sale`, `join_year`, `year_built`, `year_reno`. Biểu đồ cho thấy các đặc điểm khác nhau về phân phối và tính tập trung của từng biến. Đặc biệt, `join_year` và `year_reno` có nhiều giá trị trùng nhau (như 0 hoặc 2024), trong khi `year_built` trải dài trong hơn 100 năm.

Nhận xét (Về dữ liệu thuộc nhóm thời gian)

Quan sát `histogram` từ 4 cột dữ liệu `year_sale, join_year, year_built, year_reno`, ta nhận thấy một số điểm đáng chú ý:

(i) **year_sale, year_built** có phân bố bình thường, không cần thay đổi

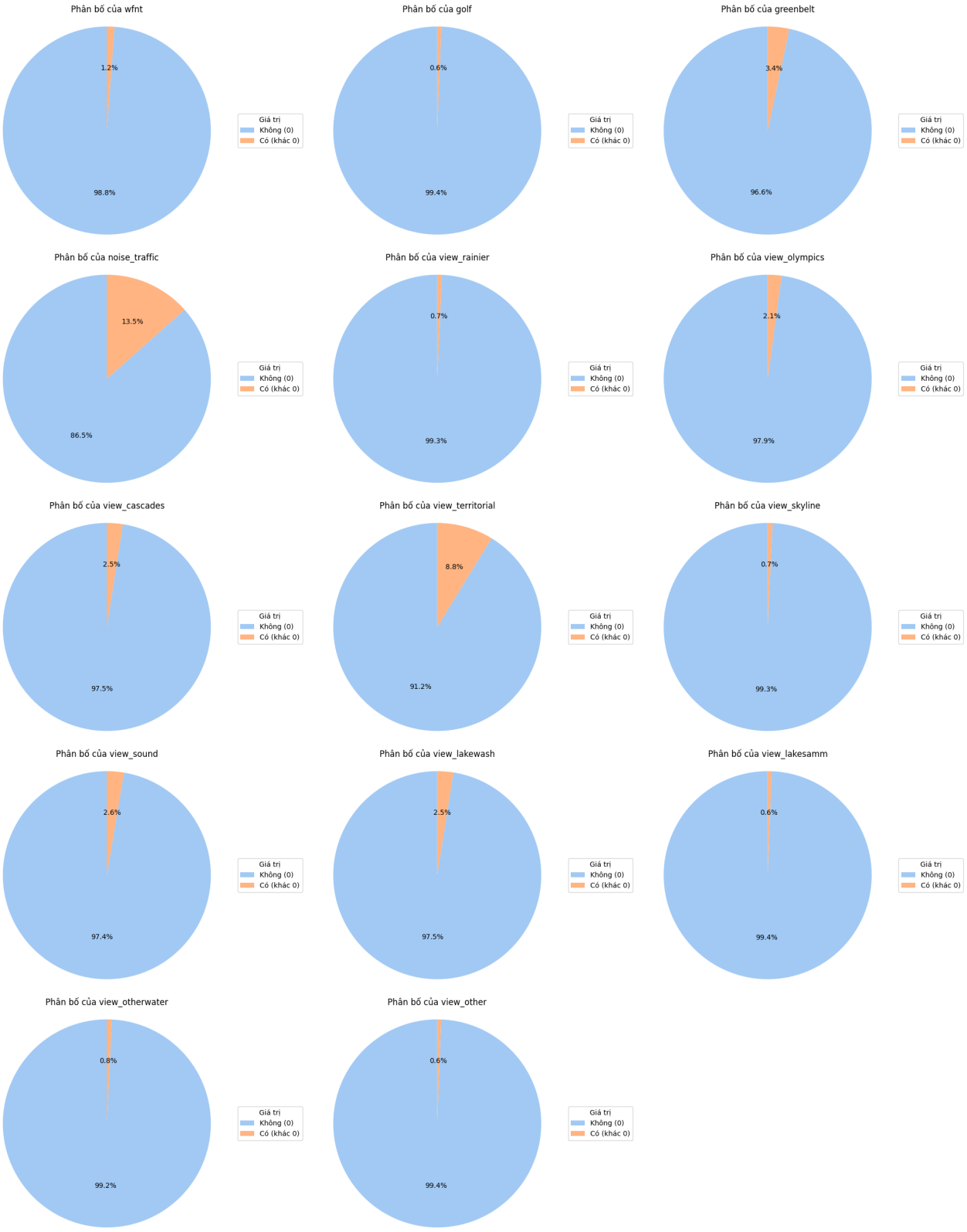
(ii) **join_year** chỉ có 2 giá trị năm là 1999 và 2025, vì dữ liệu chỉ có 2 lớp và nó cũng không mang tính định tính cho lắm, nên ta sẽ cho nó vào nhóm dữ liệu `category`.

(iii) **year_reno** có nhiều giá trị 0, do dữ liệu quy định nếu nhà chưa cải tạo thì không có năm cải tạo và đặt bằng 0.

§2.2.3 Phân tích các biến nhị phân (thuộc nhóm tầm nhìn)

```
binary_view_columns = [
    'wfnt', 'golf', 'greenbelt', 'noise_traffic',
    'view_rainier', 'view_olympics', 'view_cascades',
    'view_territorial', 'view_skyline', 'view_sound',
    'view_lakewash', 'view_lakesamm', 'view_otherwater', 'view_other'
]

plt.figure(figsize=(20, 25))

for i, col in enumerate(binary_view_columns):
    # Đếm số lượng "Không" (0) và "Có" (khác 0)
    count_0 = (train[col] == 0).sum()
    count_nonzero = (train[col] != 0).sum()

    if count_0 == 0 or count_nonzero == 0:
        print(f"Cột '{col}' chỉ có 1 loại giá trị, không vẽ.")
        continue

    # Chuẩn bị pie chart
    labels = ['Không (0)', 'Có (khác 0)']
    sizes = [count_0, count_nonzero]
    colors = sns.color_palette('pastel')

    # Tạo subplot và pie chart
    ax = plt.subplot(5, 3, i + 1)
    wedges, texts, autotexts = ax.pie(
        sizes,
        labels=None,  # Không in label trực tiếp lên pie
        autopct='%1.1f%%',
        startangle=90,
        colors=colors
    )

    ax.set_title(f'Phân bố của {col}')
    ax.axis('equal')  # Hình tròn

    # Thêm legend thay vì label trực tiếp
    ax.legend(wedges, labels, title="Giá trị", loc="center left", bbox_to_anchor=(1,
    0.5))

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

Listing 2.4: Phân bố biến nhị phân của dữ liệu thuộc nhóm tầm nhìn



Hình 2.2: Phân bố tỷ lệ giữa giá trị `0` (không có) và `khác 0` (có) trong các biến nhị phân liên quan đến tầm nhìn và đặc điểm vị trí bất động sản.

Nhận xét (Về các biến nhị phân thuộc nhóm tầm nhìn)

Biểu đồ trên minh hoạ tỉ lệ phân bố giữa giá trị "Có" và "Không" của các biến nhị phân liên quan đến đặc điểm tầm nhìn và yếu tố môi trường trong dữ liệu (thuộc nhóm 4). Từ đó, có thể rút ra một số nhận xét sau:

(i) Các biến như `wfnt`, `golf`, `greenbelt`, `view_rainier`, `view_olympics`, ... có giá trị "Có" rất hiếm (chỉ xuất hiện khoảng $0.5 - 3\%$ tổng số), cho thấy đây là những yếu tố cao cấp/hiếm có trong bất động sản. Những đặc điểm này có thể làm tăng giá trị tài sản nếu được xác nhận có liên quan đến giá bán, và có thể là yếu tố phân biệt giữa các phân khúc nhà phổ thông và nhà sang trọng.

(ii) Biến `noise_traffic` xuất hiện giá trị "Có" phổ biến hơn (khoảng 13.5%), phản ánh tác động tiêu cực đến môi trường sống. Điều này có thể làm giảm giá trị tài sản hoặc ảnh hưởng đến quyết định mua của khách hàng.

(iii) Các biến nhị phân này đều mang thông tin quan trọng đối với giá nhà, vì vậy nhóm không thực hiện biến đổi dữ liệu (như bỏ cột hoặc mã hóa) đối với nhóm biến này.

§2.3 Tổng kết

Quá trình phân tích khám phá dữ liệu (Exploratory Data Analysis – `EDA`) đóng vai trò thiết yếu trong việc hiểu rõ cấu trúc và bản chất của bộ dữ liệu trước khi bước vào các giai đoạn tiền xử lý và mô hình hóa. Dữ liệu trong bài toán bao gồm nhiều loại biến khác nhau: biến số liên tục (như diện tích, giá), biến phân loại (loại nhà, khu vực), biến nhị phân (có/không), và các đặc trưng dạng thời gian (ngày giao dịch, năm xây dựng). Việc nhận diện đúng kiểu biến là tiền đề quan trọng cho việc xử lý phù hợp ở các bước sau.

Một số biến trong tập dữ liệu có tỷ lệ thiếu khá cao, đặc biệt ở các trường thông tin mô tả chi tiết (ví dụ: số tầng, vật liệu xây dựng), đặt ra yêu cầu nghiêm túc trong việc xử lý thiếu dữ liệu – có thể bằng cách loại bỏ, nội suy, hoặc gán giá trị mặc định có ý nghĩa thống kê. Việc lựa chọn kỹ thuật phù hợp phụ thuộc vào phân phối và mối quan hệ của từng biến với mục tiêu dự đoán.

Những kết quả rút ra từ `EDA` cho phép định hình chiến lược tiền xử lý và xây dựng đặc trưng hiệu quả hơn. Việc hiểu rõ các cụm giá trị bất thường, xu hướng theo thời gian hoặc sự mất cân bằng dữ liệu giúp mô hình sau này tránh được `overfitting` và phản ánh đúng bản chất của thị trường. Từ đó, dữ liệu đã sẵn sàng cho các bước xử lý và trích chọn đặc trưng nâng cao nhằm tối ưu hóa hiệu suất mô hình dự đoán.

3 Xử lý Dữ liệu

§3.1 Xử lý dữ liệu thời gian

❒ **Ý tưởng:** Biến đổi `sale_date, year_built, year_reno` thành các đặc trưng định lượng có ý nghĩa hơn cho mô hình học máy. Quy trình xử lý:

(i) Chuyển cột `sale_date` từ chuỗi sang định dạng thời gian chuẩn (`datetime`), để dễ thao tác.

(ii) Trích xuất năm bán (**year_sale**) sau đó tính tuổi của ngôi nhà tính đến thời điểm bán, vì có thể có đặc trưng quan trọng phản ánh chất lượng hoặc độ khấu hao.

(iii) Tính tuổi của lần cải tạo gần nhất (nếu có), nếu **year_reno** > 0. Nếu chưa cải tạo thì gán là 0.

(iv) Xóa các cột không cần thiết sau khi đã trích xuất đặc trưng để tránh `multicollinearity` (đa cộng tuyến), nghĩa là một đặc trưng có thể được suy ra từ các đặc trưng khác, chẳng hạn như:

$$\textbf{age\_built} = \textbf{year\_sale} - \textbf{year\_built}$$

```python
def extract_date_features(df, date_column='sale_date'):
    """
    Trích xuất các đặc trưng từ cột ngày tháng:
    - 'age_built': tuổi của ngôi nhà tại thời điểm bán
    - 'age_reno': số năm kể từ lần cải tạo (nếu có)
    Sau khi trích xuất, loại bỏ các cột gốc: 'sale_date', 'year_sale', 'year_built', 'year_reno'
    """
    df_copy = df.copy()

    if date_column in df_copy.columns:
        # Chuyển đổi sang định dạng datetime
        df_copy[date_column] = pd.to_datetime(df_copy[date_column])

        # Trích xuất năm bán
        df_copy['year_sale'] = df_copy[date_column].dt.year

        # Tính tuổi nhà
        df_copy['age_built'] = df_copy['year_sale'] - df_copy['year_built']

        # Tính số năm kể từ lần cải tạo (nếu có), nếu không thì NaN
        df_copy['age_reno'] = np.where(df_copy['year_reno'] > 0,
                                       df_copy['year_sale'] - df_copy['year_reno'],
                                       0)

        # Loại bỏ các cột gốc không còn cần thiết
        df_copy = df_copy.drop(columns=[date_column, 'year_sale', 'year_built', 'year_reno'])

        print(f"Đã trích xuất các đặc trưng từ {date_column}")

    return df_copy

# Chạy trên 2 tập dữ liệu
train_with_date_features = extract_date_features(train, 'sale_date')
test_with_date_features = extract_date_features(test, 'sale_date')
```

Listing 3.1: Xử lý và trích xuất đặc trưng thời gian từ sale_date

§3.2 Xử lý dữ liệu phân loại

❒ **Ý tưởng:** Biến đổi `sale_date, year_built, year_reno` thành các đặc trưng định lượng có ý nghĩa hơn cho mô hình học máy. Quy trình xử lý:

(i) Dùng **LabelEncoder** để ánh xạ từng giá trị phân loại (`string`) thành số nguyên.

(ii) Thay `NaN` bằng `'unknown'`. Với tập kiểm tra, nếu có giá trị chưa từng xuất hiện trong tập huấn luyện, sẽ gán chúng thành `'unknown'` trước khi mã hóa.

(iii) Lưu **LabelEncoder** của từng cột trong một `dict (encoders)` để dùng lại khi xử lý dữ liệu `test/new data`.

(iv) Tránh lỗi khi dữ liệu `test` có giá trị mới (chưa từng gặp) bằng cách chủ động gán `'unknown'`

```
categorical_columns = ['sale_warning', 'join_status', 'city', 'zoning', 'subdivision', 'submarket', 'join_year']

def preprocess_categorical(df, categorical_cols, label_encoders=None, is_training=True):
    """
    Tiền xử lý các cột phân loại bằng Label Encoding.
    Nếu là dữ liệu huấn luyện, sẽ khởi tạo và huấn luyện encoder.
    Nếu là dữ liệu kiểm tra, sử dụng encoder đã huấn luyện để mã hóa.

    Tham số:
    - df: DataFrame gốc
    - categorical_cols: danh sách các cột phân loại cần xử lý
    - label_encoders: dict chứa các LabelEncoder (chỉ dùng khi is_training=False)
    - is_training: True nếu đang xử lý tập huấn luyện, False nếu là tập kiểm tra/test

    Trả về:
    - df_processed: DataFrame đã được mã hóa
    - encoders: dict các LabelEncoder (chỉ trả về nếu là is_training=True)
    """
    df_processed = df.copy()

    if is_training:
        encoders = {}
        for col in categorical_cols:
            if col in df_processed.columns:
                le = LabelEncoder()

                # Thay thế giá trị thiếu bằng 'unknown'
                df_processed[col] = df_processed[col].fillna('unknown')
                df_processed[col] = df_processed[col].astype(str)

                # Đảm bảo 'unknown' nằm trong danh sách các giá trị duy nhất
                unique_values = df_processed[col].unique().tolist()
                if 'unknown' not in unique_values:
                    unique_values.append('unknown')

                # Huấn luyện encoder và biến đổi dữ liệu
                le.fit(unique_values)
                df_processed[col] = le.transform(df_processed[col])

                # Lưu lại encoder cho cột đó
                encoders[col] = le

        return df_processed, encoders

    else:
        for col in categorical_cols:
            if col in df_processed.columns and col in label_encoders:
```

```
                df_processed[col] = df_processed[col].fillna('unknown')
                df_processed[col] = df_processed[col].astype(str)

                le = label_encoders[col]

                # Gán 'unknown' cho các giá trị mới chưa từng gặp
                unknown_mask = ~df_processed[col].isin(le.classes_)
                df_processed.loc[unknown_mask, col] = 'unknown'

                # Mã hóa dữ liệu theo encoder đã huấn luyện
                df_processed[col] = le.transform(df_processed[col])

        return df_processed

# Gọi hàm cho tập huấn luyện
train_processed, label_encoders = preprocess_categorical(train_with_date_features,
    categorical_columns, is_training=True)
```

Listing 3.2: Label Encoding cho các biến phân loại

Ghi chú: Đối với các biến phân loại được gán nhãn theo cách này, cần lưu lại bộ mã hóa `LabelEncoder` để sử dụng nhất quán trong giai đoạn triển khai mô hình (`inference`).

§3.3 Xử lý dữ liệu số bị thiếu

❒ **Ý tưởng:** Duyệt qua toàn bộ các cột số trong tập dữ liệu để kiểm tra và xử lý các giá trị bị thiếu. Vì `sale_price` là biến mục tiêu nên không thực hiện xử lý thiếu ở bước này. Với mỗi cột có thiếu, thay thế các giá trị thiếu bằng trung vị (`median`) của cột đó, giúp tránh ảnh hưởng của ngoại lệ so với dùng giá trị trung bình.

```
# Lấy danh sách các cột kiểu số (numeric) trong DataFrame
numeric_columns = train_processed.select_dtypes(include=[np.number]).columns.tolist()

# Loại bỏ cột đích (target) 'sale_price' ra khỏi danh sách vì không nên xử lý thiếu ở
    cột này tại bước này
if 'sale_price' in numeric_columns:
    numeric_columns.remove('sale_price')

# Duyệt qua từng cột số còn lại
for col in numeric_columns:
    # Kiểm tra nếu cột có giá trị thiếu
    if train_processed[col].isnull().sum() > 0:
        # Tính giá trị trung vị (median) của cột đó
        median_val = train_processed[col].median()
        # Điền các giá trị thiếu bằng median
        train_processed[col] = train_processed[col].fillna(median_val)
        # In ra thông báo để theo dõi
        print(f"Điền {col} với giá trị median: {median_val}")
```

Listing 3.3: Điền giá trị thiếu của các cột số bằng trung vị

§3.4 Chuẩn bị dữ liệu cho mô hình

❒ **Quy trình:** Sau khi xử lý các đặc trưng, cần chuẩn bị dữ liệu đầu vào cho mô hình học máy. Các bước chuẩn bị bao gồm:

(i) **Tách dữ liệu**: Phân tách các biến đầu vào (`features`) và biến mục tiêu (`target`).

(ii) **Chia tập train/validation**: Dữ liệu huấn luyện được chia làm 2 phần: `train` và `validation` để đánh giá mô hình.

(iii) **Chuẩn hóa dữ liệu (Scaling)**: Áp dụng `Standard Scaler` để đưa các đặc trưng về cùng thang đo.

(iv) **Xử lý tập test**: Áp dụng cùng phép biến đổi đã dùng ở tập `train` để đảm bảo thống nhất dữ liệu.

(v) **Lưu dữ liệu đã xử lý**: Lưu lại dữ liệu sau khi xử lý để tái sử dụng trong huấn luyện và thử nghiệm mô hình.

Cụ thể hơn, ta sẽ đi qua từng bước như sau:

§3.4.1 Tách features và target

❒ **Ý tưởng:** Sau khi hoàn tất các bước xử lý đặc trưng, bước tiếp theo là tách biến mục tiêu (`target`) ra khỏi tập dữ liệu, giữ lại các biến độc lập (`features`) để làm đầu vào cho mô hình học máy. Trong bài toán này, biến mục tiêu là `sale_price`, còn tất cả các cột còn lại được dùng làm đặc trưng.

```
feature_columns = [col for col in dataset_processed.columns if col not in ['
    sale_price']]
X = dataset_processed[feature_columns]
y = dataset_processed['sale_price']

print(f"\nSố lượng features: {X.shape[1]}")
print(f"Các đặc trưng từ ngày: sale_year, sale_month, sale_quarter, sale_day_of_week,
     sale_day_of_month")
print(f"Tên các features: {X.columns.tolist()}...")
```

Listing 3.4: Tách features và target từ dataset đã xử lý

Ghi chú: Sau bước này, dữ liệu huấn luyện gồm:

- `X`: ma trận đặc trưng đầu vào có 50 cột,
- `y`: biến mục tiêu là giá bán nhà `sale_price`,
- Nhiều đặc trưng quan trọng được tạo từ dữ liệu ngày tháng như `sale_year`, `sale_month`, `sale_quarter`, `sale_day_of_week`, `sale_day_of_month`,...

§3.4.2 Chia tập `train` và `validation`

❒ **Ý tưởng:** Để đánh giá hiệu suất của mô hình một cách khách quan trước khi áp dụng lên tập kiểm tra thực sự, ta chia dữ liệu thành hai phần:

- **Tập huấn luyện (train set)**: để mô hình học các đặc trưng từ dữ liệu.
- **Tập kiểm định (validation set)**: để đánh giá hiệu quả mô hình trên dữ liệu chưa từng thấy.

Đối với bộ dữ liệu hiện tại, ta sử dụng hàm `train_test_split` từ thư viện `scikit-learn` để chia dữ liệu theo tỷ lệ 80% huấn luyện và 20% kiểm định, đảm bảo tính tái lập kết quả trong quá trình phân tích với `random_state = 42`. Đây là một bộ `parameter` thông dụng khi chia tập dữ liệu sử dụng `train_test_split`.

```
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=None
)

print(f"\nKích thước tập train: {X_train.shape}")
print(f"Kích thước tập validation: {X_val.shape}")
```

Listing 3.5: Chia dữ liệu thành tập train và validation

❒ **Kết quả:** Ta chia được tập huấn luyện và tập kiểm định với kích thước và số đặc trưng như bên dưới:

- Tập huấn luyện: 160000 dòng × 50 đặc trưng.
- Tập kiểm định: 40000 dòng × 50 đặc trưng.

§3.4.3 Chuẩn hóa dữ liệu

❒ **Ý tưởng:** Dữ liệu đầu vào của mô hình học máy nên được đưa về cùng thang đo để mô hình học ổn định và nhanh hội tụ hơn. Do đó, ta áp dụng `StandardScaler` để chuẩn hóa các đặc trưng đầu vào, sao cho mỗi đặc trưng có trung bình bằng 0 và độ lệch chuẩn bằng 1. Một điểm cần lưu ý là chỉ sử dụng thống kê từ tập huấn luyện để chuẩn hóa các tập còn lại, nhằm tránh **data leakage**.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

print("\nĐã hoàn thành chuẩn hóa dữ liệu")
```

Listing 3.6: Chuẩn hóa dữ liệu huấn luyện và validation

§3.4.4 Xử lý tập test

❒ **Ý tưởng:** Tập test cần được xử lý giống như tập huấn luyện, có nghĩa là `encode` các biến phân loại, xử lý thiếu, và chuẩn hóa. Đặc biệt, cần dùng **cùng encoder và scaler** đã học từ tập huấn luyện để đảm bảo thống nhất.

```
test_processed = preprocess_categorical(
    test_with_date_features,
    categorical_columns,
    label_encoders,
    is_training=False
)

for col in feature_columns:
    if col in test_processed.columns and \
       test_processed[col].dtype in ['int64', 'float64'] and \
       test_processed[col].isnull().sum() > 0:
        median_val = dataset_processed[col].median()
        test_processed[col] = test_processed[col].fillna(median_val)

X_test = test_processed[feature_columns]
X_test_scaled = scaler.transform(X_test)

print(f"Kích thước tập test sau xử lý: {X_test_scaled.shape}")
print(f"Đã xử lý tập test với {len(feature_columns)} features")
```

Listing 3.7: Xử lý và chuẩn hóa tập test

§3.4.5 Lưu dữ liệu đã xử lý

❒ **Ý tưởng:** Sau khi xử lý và chuẩn hóa xong dữ liệu, ta nên lưu lại tất cả các đối tượng và tập dữ liệu dưới dạng `file` để phục vụ cho huấn luyện mô hình hoặc `inference` sau này, tránh phải xử lý lại từ đầu.
Các đối tượng cần lưu bao gồm:

- `scaler.pkl`: đối tượng chuẩn hóa dữ liệu.

- `label_encoders.pkl`: các encoder áp dụng cho biến phân loại.
- `feature_columns.pkl`: danh sách các cột được sử dụng làm đặc trưng.
- Các tập dữ liệu đã xử lý: `X_train_scaled.npy`, `X_val_scaled.npy`, `X_test_scaled.npy`, `y_train.npy`, `y_val.npy`.

```
os.makedirs('processed', exist_ok=True)

joblib.dump(scaler, 'processed/scaler.pkl')
joblib.dump(label_encoders, 'processed/label_encoders.pkl')
joblib.dump(feature_columns, 'processed/feature_columns.pkl')

np.save('processed/X_train_scaled.npy', X_train_scaled)
np.save('processed/X_val_scaled.npy', X_val_scaled)
np.save('processed/X_test_scaled.npy', X_test_scaled)
np.save('processed/y_train.npy', y_train.values)
np.save('processed/y_val.npy', y_val.values)

print("\nĐã lưu tất cả dữ liệu đã xử lý:")
print("- processed/scaler.pkl")
print("- processed/label_encoders.pkl")
print("- processed/feature_columns.pkl")
print("- processed/X_train_scaled.npy")
print("- processed/X_val_scaled.npy")
print("- processed/X_test_scaled.npy")
print("- processed/y_train.npy")
print("- processed/y_val.npy")
```

Listing 3.8: Lưu lại scaler

§3.5 Tổng kết quá trình xử lý dữ liệu

Sau khi hoàn tất tất cả các bước xử lý, ta tiến hành thống kê một số đặc trưng cơ bản của biến mục tiêu `sale_price` để nắm tổng quan về phân phối giá nhà. Các chỉ số thống kê được thể hiện như sau:

```
Thống kê về sale_price:
Mean: 584,149
Median: 459,950
Std: 417,060
Min: 50,293
Max: 2,999,950
```

Nhận xét (Về kết quả của quá trình xử lý dữ liệu)

Có thể nhận thấy phân phối của `sale_price` có độ lệch đáng kể: giá trị trung bình (`mean`) cao hơn nhiều so với trung vị (`median`), cho thấy sự tồn tại của các `outlier` phía đuôi phải (`right-skewed distribution`). Độ lệch chuẩn lớn ($\approx 417,000$) và khoảng giá trị trải dài từ hơn 50 nghìn đến gần 3 triệu phản ánh mức độ phân tán rất cao của dữ liệu.

Điều này đặt ra một số thách thức cho mô hình dự báo, vì mô hình cần phải học được các mối quan hệ trong điều kiện dữ liệu không đồng đều và có thể bị chi phối bởi các quan sát bất thường. Do đó, việc lựa chọn hàm mất mát phù hợp (ví dụ: `Quantile Loss`) và đánh giá mô hình bằng các thước đo như `MAE` hoặc `Winkler Score` sẽ trở nên quan trọng trong giai đoạn tiếp theo.

4 Huấn luyện Mô hình Dự đoán

§4.1 Ý tưởng tổng quan

Trong thực tế, giá bất động sản có thể dao động trong một khoảng lớn do nhiều yếu tố không quan sát được (ví dụ: chất lượng nội thất, điều kiện pháp lý, nhu cầu thị trường). Do đó, thay vì dự đoán một giá trị duy nhất, bài toán được mở rộng sang dạng **dự đoán khoảng giá** với độ tin cậy cao (ví dụ: 90%).

Phương pháp được lựa chọn gồm hai bước chính:

(i) Dự đoán hai phân vị `quantile` là 0.05 và 0.95 bằng mô hình `LightGBM Quantile Regressor`

(ii) Áp dụng `Conformal Prediction` để hiệu chỉnh khoảng sao cho đảm bảo độ phủ thực tế đạt được đúng kỳ vọng

Toàn bộ mô hình được huấn luyện và đánh giá dựa trên các chỉ số như `MAE`, `RMSE`, `Coverage`, và `Winkler Score` — một thước đo định lượng chất lượng khoảng dự đoán.

Trước khi đi sâu hơn, ta sẽ đi qua đôi chút thông tin về `LightGBM Quantile` và `Conformal Prediction` – 2 nhân tố chính giúp thiết kế mô hình dự đoán của nhóm.

§4.1.1 LightGBM Quantile Regression

Trong các bài toán hồi quy truyền thống, mô hình thường dự đoán một giá trị trung bình kỳ vọng $\mathbb{E}[y|x]$. Tuy nhiên, điều này không phản ánh được sự không chắc chắn nội tại trong dữ liệu. Thay vào đó, hồi quy phân vị (`quantile regression`) cho phép mô hình ước lượng các phân vị cụ thể của phân phối đầu ra có điều kiện theo đầu vào, chẳng hạn như:

$$\hat{y}^{(l)} = Q_{0.05}(y|x), \quad \hat{y}^{(u)} = Q_{0.95}(y|x)$$

Trong đó, $Q_\alpha(y|x)$ là phân vị bậc α của biến ngẫu nhiên y biết x.

`LightGBM` là một thư viện Gradient Boosting rất hiệu quả, hỗ trợ huấn luyện mô hình với hàm mất mát `Quantile Loss`:

$$\mathcal{L}_\alpha(y, \hat{y}) = \begin{cases} \alpha(y - \hat{y}), & y > \hat{y} \\ (1-\alpha)(\hat{y} - y), & y \leq \hat{y} \end{cases}$$

Khi huấn luyện hai mô hình với $\alpha = 0.05$ và $\alpha = 0.95$, ta thu được khoảng dự đoán với độ tin cậy 90%, tức:

$$\mathbb{P}(y \in [\hat{y}^{(l)}, \hat{y}^{(u)}]) \approx 90\%$$

Nhận xét (Lý do chọn LightGBM Quantile)

`LightGBM` kết hợp tốt giữa tốc độ huấn luyện, khả năng xử lý tập dữ liệu lớn và khả năng hỗ trợ trực tiếp bài toán hồi quy phân vị. Việc huấn luyện hai mô hình độc lập cho hai phân vị thấp và cao cũng linh hoạt hơn so với cách dự đoán khoảng dựa vào độ lệch chuẩn từ hồi quy truyền thống.

§4.1.2 Conformal Prediction

Trong thực tế, khoảng dự đoán từ mô hình học máy thường không đảm bảo **coverage** đúng với xác suất mong muốn, do hiện tượng `overfitting` hoặc phân phối dữ liệu `validation` khác với test. **Conformal Prediction** là một kỹ thuật hiệu chỉnh `post-hoc` (nghĩa là chỉnh sửa kết quả sau khi mô hình đã huấn luyện xong, không can thiệp vào bên trong mô hình), đảm bảo độ phủ thống kê đúng, không phụ thuộc vào giả định phân phối.

Cách tiếp cận đơn giản nhất là sử dụng phương sai phần dư để điều chỉnh khoảng:

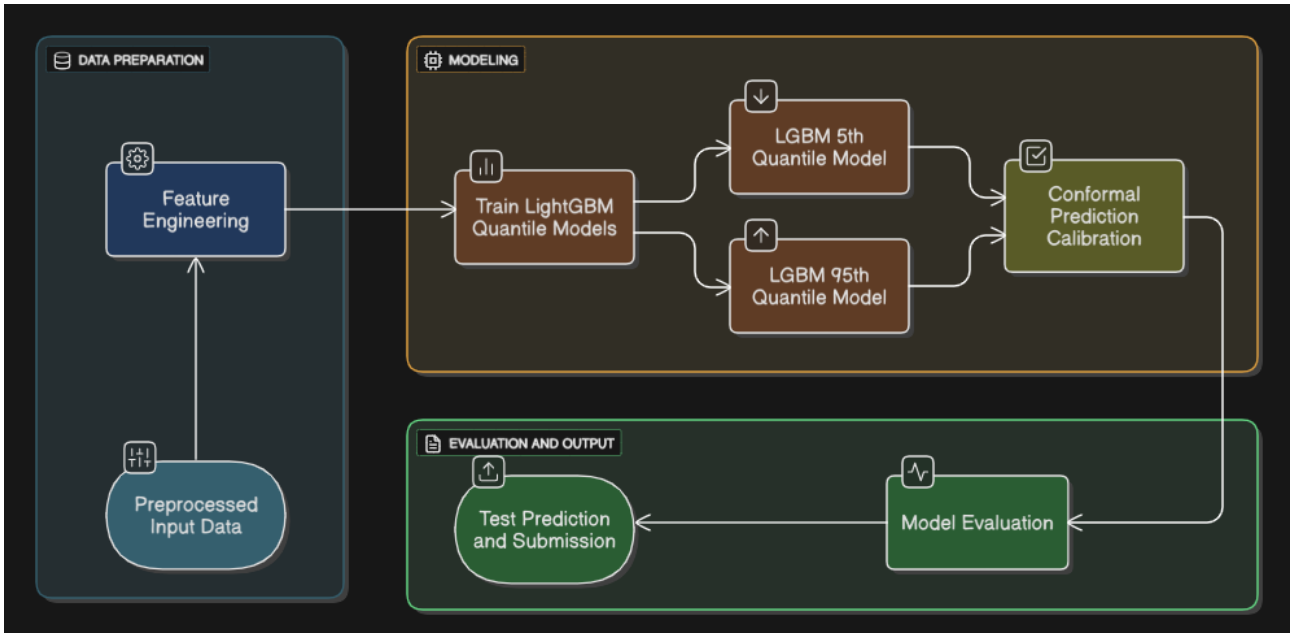
$$
\begin{aligned}
r_i &= \max(0, \hat{y}_i^{(l)} - y_i) + \max(0, y_i - \hat{y}_i^{(u)}) \\
q_{\text{hat}} &= \text{Quantile}_{1-\alpha}(r_i) \\
\hat{y}_{\text{adj}}^{(l)} &= \hat{y}^{(l)} - q_{\text{hat}}, \quad \hat{y}_{\text{adj}}^{(u)} = \hat{y}^{(u)} + q_{\text{hat}}
\end{aligned}
$$

Nhận xét (Ý nghĩa của hiệu chỉnh Conformal)

Conformal Prediction đóng vai trò như một lớp "bảo hiểm thống kê" cho mô hình học máy. Dù mô hình chính có bị lệch hoặc không thể học hết phân phối dữ liệu, việc cộng/trừ thêm q_{hat} sẽ giúp khoảng dự đoán thực tế đạt được độ tin cậy đúng như thiết kế — ví dụ 90%. Ưu điểm nổi bật là: đơn giản, không giả định về phân phối lỗi, và áp dụng được cho bất kỳ mô hình nào.

§4.2 Quy trình huấn luyện

Quy trình huấn luyện mô hình được thiết kế như sau: Dữ liệu đầu vào được chuẩn hóa và trải qua bước `Feature Engineering` nhằm tạo thêm các đặc trưng tương tác có ý nghĩa. Sau đó, hai mô hình LightGBM được huấn luyện riêng biệt cho hai phân vị 5% và 95%, tạo nên khoảng dự đoán ban đầu. Kỹ thuật `Conformal Prediction` tiếp tục được áp dụng để hiệu chỉnh khoảng này, đảm bảo tỷ lệ bao phủ đạt khoảng 90% như mong muốn. Cuối cùng, mô hình được đánh giá bằng các chỉ số định lượng (`MAE, RMSE, Winkler Score`) trước khi áp dụng lên tập kiểm thử để tạo tập `submission` sẵn sàng cho quá trình chấm điểm. Minh họa cho quy trình tổng thể được minh họa như hình dưới.



Hình 4.1: Pipeline huấn luyện mô hình dự đoán khoảng giá.

§4.2.1 Tạo đặc trưng nâng cao (Feature Engineering)

❒ **Ý tưởng:** Các đặc trưng ban đầu trong dữ liệu thường chỉ là thông tin cơ bản (diện tích, số phòng, năm xây, ...). Tuy nhiên, để mô hình học tốt hơn, ta cần tạo thêm các đặc trưng mở rộng như tỷ lệ, tương tác, biến logarit hoặc biến độ tuổi. Những đặc trưng này có thể tiết lộ các mối quan hệ phi tuyến hoặc tương tác ngầm giữa các yếu tố ảnh hưởng đến giá nhà, từ đó giúp mô hình dự đoán tốt hơn.

```
def create_advanced_features(X_scaled, feature_names):
    X_df = pd.DataFrame(X_scaled, columns=feature_names)
    current_year = date.today().year

    if 'sqft' in X_df and 'sqft_lot' in X_df:
        X_df['sqft_ratio'] = X_df['sqft'] / (X_df['sqft_lot'] + 1)
        X_df['sqft_lot_log'] = np.log1p(np.abs(X_df['sqft_lot']))
        X_df['sqft_log'] = np.log1p(np.abs(X_df['sqft']))

    if 'land_val' in X_df and 'imp_val' in X_df:
        X_df['total_val'] = X_df['land_val'] + X_df['imp_val']
        X_df['imp_land_ratio'] = X_df['imp_val'] / (X_df['land_val'] + 1)
        X_df['total_val_log'] = np.log1p(np.abs(X_df['total_val']))
        X_df['land_val_per_sqft'] = X_df['land_val'] / (X_df['sqft_lot'] + 1)

    if 'beds' in X_df and 'sqft' in X_df:
        X_df['sqft_per_bed'] = X_df['sqft'] / (X_df['beds'] + 1)
        X_df['beds_sqft_interaction'] = X_df['beds'] * X_df['sqft']

    if 'bath_full' in X_df and 'bath_3qtr' in X_df and 'bath_half' in X_df:
        X_df['total_bath'] = X_df['bath_full'] + 0.75 * X_df['bath_3qtr'] + 0.5 *
    X_df['bath_half']
        X_df['bath_per_bed'] = X_df['total_bath'] / (X_df['beds'] + 1)

    if 'year_built' in X_df:
        X_df['house_age'] = current_year - X_df['year_built']
        X_df['house_age_squared'] = X_df['house_age'] ** 2
        if 'year_reno' in X_df:
            X_df['years_since_reno'] = np.where(X_df['year_reno'] > 0, current_year -
     X_df['year_reno'], X_df['house_age'])

    view_cols = [col for col in X_df if col.startswith('view_')]
    if view_cols:
        X_df['total_view_score'] = X_df[view_cols].sum(axis=1)
        X_df['has_view'] = (X_df['total_view_score'] > 0).astype(int)

    if 'gara_sqft' in X_df and 'sqft' in X_df:
        X_df['garage_ratio'] = X_df['gara_sqft'] / (X_df['sqft'] + 1)
        X_df['has_garage'] = (X_df['gara_sqft'] > 0).astype(int)

    if 'latitude' in X_df and 'longitude' in X_df:
        X_df['lat_long_interaction'] = X_df['latitude'] * X_df['longitude']
        X_df['distance_from_center'] = np.sqrt((X_df['latitude'] - X_df['latitude'].
    mean())**2 + (X_df['longitude'] - X_df['longitude'].mean())**2)

    if 'grade' in X_df and 'condition' in X_df:
        X_df['grade_condition_interaction'] = X_df['grade'] * X_df['condition']
        X_df['quality_score'] = X_df['grade'] + X_df['condition']

    return X_df.values

X_train = create_advanced_features(X_train_scaled, feature_columns)
X_val = create_advanced_features(X_val_scaled, feature_columns)
X_test = create_advanced_features(X_test_scaled, feature_columns)
```

Listing 4.1: Tạo đặc trưng mở rộng từ dữ liệu đã chuẩn hóa

Nhận xét (Về việc mở rộng đặc trưng)

Việc mở rộng đặc trưng giúp mô hình học được các quan hệ tiềm ẩn như `sqft/beds`, `giá trị đất/diện tích`, hay tương tác `grade × condition`. Đặc biệt, các biến như `total_bath`, `years_since_reno` và `lat × long` cung cấp thông tin bổ sung từ dữ liệu địa lý và cải tạo — những yếu tố ảnh hưởng lớn đến giá nhà mà không thể hiện trực tiếp trong các cột gốc.

§4.2.2 Huấn luyện mô hình LightGBM Quantile Regression

❒ **Ý tưởng:** Thay vì dự đoán một giá trị trung bình như hồi quy cổ điển, ta huấn luyện hai mô hình dự đoán hai phân vị: phân vị thấp (`alpha = 0.05`) và phân vị cao (`alpha = 0.95`). Từ đó, mô hình trả về một khoảng giá — giúp định lượng mức độ không chắc chắn và phù hợp với bài toán định giá thực tế, nơi giá nhà có thể biến động lớn.

```
lgb_params = {
    'objective': 'quantile',
    'n_estimators': 1200,
    'learning_rate': 0.02,
    'num_leaves': 64,
    'max_depth': 7,
    'min_child_samples': 20,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'reg_alpha': 1.0,
    'reg_lambda': 1.0,
    'random_state': 42
}

lgb_5 = lgb.LGBMRegressor(**lgb_params, alpha=0.05)
lgb_95 = lgb.LGBMRegressor(**lgb_params, alpha=0.95)

lgb_5.fit(X_train, y_train)
lgb_95.fit(X_train, y_train)
```

Listing 4.2: Huấn luyện mô hình phân vị 5% và 95%

Nhận xét (Về cấu hình mô hình)

Cấu hình mô hình hướng đến tính tổng quát và độ ổn định: sử dụng bộ lá lớn (`num_leaves = 64`), độ sâu vừa phải (`max_depth = 7`). Thêm vào đó, việc kết hợp đồng thời cả `L1` (`reg_alpha = 1.0`) và `L2` (`reg_lambda = 1.0`) `regularization` giúp giảm `overfitting` trong các cây quyết định. Hai mô hình được huấn luyện riêng biệt giúp ước lượng rõ ràng hai điểm phân vị thay vì dựa vào độ lệch chuẩn như các mô hình hồi quy truyền thống.

§4.2.3 Hiệu chỉnh bằng Conformal Prediction

❒ **Ý tưởng:** Sử dụng các sai lệch quan sát được giữa nhãn thực tế và khoảng dự đoán ban đầu trên tập `validation` để điều chỉnh lại độ rộng của khoảng dự đoán cho dữ liệu `test`. Cách làm này không đòi hỏi giả định phân phối xác suất cho sai số dự đoán, cũng không phụ thuộc vào kiến trúc mô hình đã huấn luyện, nên có thể áp dụng linh hoạt cho nhiều mô hình phức tạp như `LightGBM`, `XGBoost`, hoặc mạng `neuron`. Việc hiệu chỉnh được thực hiện theo kiểu `post-hoc`, tức là sau khi mô hình đã huấn luyện xong và được cố định, giúp tránh `overfitting` trong quá trình tinh chỉnh mô hình. Thông qua việc sử dụng phân vị thích hợp của phần dư, phương pháp này giúp đảm bảo `coverage` tiệm cận với mức mong muốn (90%) mà vẫn giữ được độ chính xác cao. Đây là một trong những kỹ thuật đơn giản nhưng hiệu quả nhất để mở rộng khoảng dự đoán từ mô hình học máy mà không cần cấu trúc lại `pipeline` huấn luyện.

```
residuals = np.maximum(0, val_l - y_val) + np.maximum(0, y_val - val_u)
q_hat = np.quantile(residuals, 0.9)

val_l_adj = val_l - q_hat
val_u_adj = val_u + q_hat
```

Listing 4.3: Hiệu chỉnh khoảng dự đoán bằng phần dư

Nhận xét (Hiệu chỉnh bằng phần dư)

`Conformal Prediction` cung cấp cách hiệu chỉnh không phụ thuộc vào giả định phân phối, và có thể áp dụng sau bất kỳ mô hình nào. Phân vị thứ 90 của phần dư chính là một giới hạn bảo thủ nhưng hiệu quả để đảm bảo $\sim 90\%$ mẫu thực nằm trong khoảng được dự đoán. Phương pháp này đơn giản nhưng có cơ sở lý thuyết chặt chẽ và đặc biệt phù hợp trong bài toán ước lượng khoảng giá.

§4.3 Đánh giá hiệu năng trên tập validation

Để đánh giá toàn diện chất lượng mô hình dự đoán khoảng, chúng em sử dụng kết hợp nhiều thước đo: **Coverage** để kiểm tra độ bao phủ thực tế của khoảng dự đoán; **Winkler Score** để đánh giá chất lượng khoảng xét cả độ rộng và sai số; **MAE**, **RMSE** để đo sai lệch trung bình của giá trị dự đoán trung tâm; và **Width** để theo dõi mức độ tự tin của mô hình thông qua độ rộng trung bình của khoảng.

```
def winkler_score(y_true, l, u, alpha=0.1):
    width = u - l
    penalty = np.where(y_true < l, (2/alpha)*(l - y_true),
              np.where(y_true > u, (2/alpha)*(y_true - u), 0))
    return width + penalty

coverage = np.mean((y_val >= val_l_adj) & (y_val <= val_u_adj))
val_winkler = np.mean(winkler_score(y_val, val_l_adj, val_u_adj, alpha=0.1))
mean_width = np.mean(val_u_adj - val_l_adj)
val_mae = mean_absolute_error(y_val, (val_l_adj + val_u_adj) / 2)
val_rmse = np.sqrt(mean_squared_error(y_val, (val_l_adj + val_u_adj) / 2))
```

Listing 4.4: Tính các chỉ số đánh giá hiệu suất mô hình

Mỗi thước đo trong quá trình đánh giá đều mang một vai trò riêng biệt và bổ trợ cho nhau trong việc phản ánh hiệu năng tổng thể của mô hình:

- **Coverage**: phản ánh độ tin cậy thực tế của khoảng dự đoán, được tính bằng tỷ lệ phần trăm các điểm thực nằm trong khoảng $[y_L, y_U]$. Kỳ vọng giá trị này tiệm cận mức 90%, tương ứng với mức ý nghĩa $\alpha = 0.1$ được sử dụng trong hiệu chỉnh bằng `Conformal Prediction`. Coverage thấp hơn nhiều cho thấy mô hình bị thiếu bao phủ (*undercoverage*), trong khi cao hơn nhiều là dấu hiệu mô hình quá thận trọng (*overcoverage*).
- **Winkler Score**: là một thước đo toàn diện hơn, kết hợp giữa độ rộng của khoảng và hình phạt nếu giá trị thực nằm ngoài khoảng. Winkler Score càng thấp càng tốt, vì nó phản ánh rằng khoảng vừa ngắn (tức là mô hình tự tin), vừa đủ rộng để bao phủ chính xác. Đây là chỉ số then chốt giúp cân bằng giữa tính chính xác và độ tin cậy.
- **MAE** và **RMSE**: được tính trên trung điểm của khoảng dự đoán, tức là $(y_L + y_U)/2$, thể hiện sai lệch trung bình tuyệt đối và sai lệch bình phương giữa dự đoán điểm và giá trị thực. MAE cho biết mức độ sai lệch trung bình một cách trực quan, còn RMSE nhạy hơn với các sai số lớn — cả hai đều được kỳ vọng đạt giá trị thấp để đảm bảo mô hình chính xác khi cần một dự đoán duy nhất.

- **Width**: thể hiện độ rộng trung bình của khoảng dự đoán. Đây là đại diện cho mức độ "tự tin" của mô hình: khoảng càng hẹp thì mô hình càng tự tin, nhưng cần được đánh đổi với coverage. Kỳ vọng độ rộng đủ nhỏ để không làm mất tính hữu dụng của khoảng, nhưng vẫn đảm bảo coverage mong muốn.

Kết quả đo được hiệu năng trên tập `Validation` như bên dưới:

```
Coverage: 0.900
Winkler Score: 341,850.64
MAE: 63,012.56
RMSE: 116,814.94
Mean Interval Width: 247,135.10
q_hat: 5863.87
```

Nhận xét (Hiệu năng tổng thể)

Kết quả về hiệu năng của mô hình trên tập `Validation` cho ta một số góc nhìn có ý nghĩa:

(i) **Coverage**: Mô hình đạt độ phủ đúng 90%, trùng khớp với mức ý nghĩa $\alpha = 0.1$ trong bước hiệu chỉnh bằng `Conformal Prediction`. Điều này xác nhận rằng khoảng dự đoán phản ánh đúng mức độ bất định trong dữ liệu thực tế, thay vì quá lạc quan hay quá bảo thủ.

(i) **Winkler Score**: Giá trị 341,850.64 thể hiện chất lượng tổng thể của khoảng dự đoán, cân bằng giữa hai mục tiêu: độ rộng không quá lớn và mức độ bao phủ cao. Winkler Score thấp cho thấy mô hình hiếm khi dự đoán sai nghiêm trọng bên ngoài khoảng, điều vốn bị phạt nặng trong công thức của thước đo này.

(i) `MAE` và `RMSE`: Với `MAE` = 63,012.56 và `RMSE` = 116,814.94, dự đoán trung tâm của mô hình tương đối gần với giá trị thực — điều đáng ghi nhận trong bối cảnh dữ liệu phân tán cao và giá bất động sản có thể lên đến hàng triệu đô.

(i) **Chiều rộng trung bình**: Khoảng 247,135.10 phản ánh mô hình đã học được mức độ biến động thực sự của thị trường. Độ rộng này không quá hẹp gây undercoverage, cũng không quá rộng gây mất ý nghĩa dự đoán — minh chứng rằng mô hình duy trì được sự cân bằng giữa tính `informativeness` và `reliability`.

§4.4 Dự đoán trên tập kiểm thử và tạo submission

Sau khi mô hình đã được hiệu chỉnh và đánh giá trên tập `Validation`, ta sử dụng các mô hình đã huấn luyện để đưa ra khoảng dự đoán cho tập `Test`. Kết quả được lưu thành file `CSV` để phục vụ việc nộp bài và so sánh với các phương pháp khác.

```
test_l = lgb_5.predict(X_test) - q_hat
test_u = lgb_95.predict(X_test) + q_hat

submission = pd.DataFrame({
    'id': test_df['id'],
    'pi_lower': test_l,
    'pi_upper': test_u
})
submission.to_csv("submission.csv", index=False)
```

Listing 4.5: Dự đoán trên tập test và lưu submission

Nhận xét (Về khả năng triển khai mô hình)
Vì mô hình đã được chuẩn hóa và lưu sẵn các biến chuyển đổi, quá trình `inference` trên tập `test` rất nhanh và nhất quán. Mô hình có thể triển khai trực tiếp trong môi trường thực tế với đầu vào chuẩn hóa, và dễ dàng sinh ra khoảng giá giúp người dùng cá nhân hoặc tổ chức định giá nhà theo mức độ rủi ro họ chấp nhận.

§4.5 Kết Quả - Ưu điểm và Nhược điểm

Hình 4.2: Kết quả chấm điểm trên cuộc thi Kaggle

Có thể thấy kết quả này (do tên nhóm **FIT-HCMUS-Delay** nộp nhầm `file submit` nên đến 0h ngày 19/07 mới hiện ra lần `submit` tốt nhất, trong phần `source code` nộp và báo cáo em đưa kết quả tốt nhất lên trước ạ) vẫn còn lệch tương đối so với kết quả hiệu năng trên tập `Validation`. Dù còn tồn tại một số hạn chế cần lưu ý nhưng phương pháp kết hợp giữa `LightGBM Quantile Regression` và `Conformal Prediction` cũng đã thể hiện nhiều ưu điểm đáng kể trong bài toán dự đoán khoảng:

(i) **Ưu điểm:**
- Không yêu cầu giả định phân phối xác suất của biến mục tiêu.
- Huấn luyện nhanh và dễ triển khai nhờ vào đặc trưng của `LightGBM`.
- Hiệu chỉnh *post-hoc* đơn giản, đảm bảo coverage chính xác.
- Dễ dàng mở rộng cho các mức phân vị khác nhau hoặc nhiều mô hình song song.
- Tương thích tốt với bài toán có quy mô lớn hoặc dữ liệu không chuẩn hóa.

(ii) **Nhược điểm:**
- Chiều rộng khoảng dự đoán còn khá lớn trong dữ liệu có phương sai cao.
- Thiếu khả năng mô hình hóa sự phụ thuộc phức tạp giữa các phân vị (ví dụ: không đồng thời tối ưu cả $q_{0.05}$ và $q_{0.95}$).
- Phụ thuộc vào chất lượng của tập `validation` khi tính toán sai số phần dư trong hiệu chỉnh.
- Không tận dụng được tiềm năng biểu diễn của các mô hình học sâu trong các ngữ cảnh phi tuyến mạnh.

§4.6 Tổng kết và nhận xét

Mô hình `LightGBM Quantile` kết hợp với `Conformal Prediction` đã đáp ứng được yêu cầu của bài toán: dự đoán khoảng giá với độ phủ chính xác, khoảng không quá rộng, và có khả năng triển khai thực tế. Phương pháp này vừa mạnh, vừa dễ huấn luyện, không đòi hỏi giả định phân phối, và linh hoạt mở rộng cho các phân vị khác.

Điểm hạn chế nằm ở chiều rộng khoảng còn khá lớn do dữ liệu gốc có phương sai cao. Hướng cải thiện tiếp theo có thể là:

- Thử các mô hình học sâu kết hợp `Quantile Loss`
- Sử dụng mô hình `ensemble` hoặc `stacking` giữa `GBDT` và `Neural Network`
- Điều chỉnh các phân vị động theo `cluster` hoặc `segment` thị trường