



NHÓM 1

BÁO CÁO BÀI TẬP NHÓM MÔN KHOA HỌC DỮ LIỆU

Đề tài: Huấn luyện, lựa chọn và kiểm thử mô hình dự đoán giá xe hơi đã qua sử dụng tại Việt Nam

THÀNH VIÊN:

- | | | |
|----------------------|---|----------|
| Nguyễn Nhật Hoàng | - | 22T_DT2 |
| Nguyễn Văn Thương | - | 22T_KHDL |
| Nguyễn Hữu Hùng Dũng | - | 22T_KHDL |

MỤC LỤC



1 Tổng quan

2 Feature Engineering

3 Chia dữ liệu

4 Mô hình hóa và đánh giá

TỔNG QUAN

Mục tiêu:

- Xây dựng mô hình dự đoán giá xe ô tô cũ tại Việt Nam
- Input: Các thông tin liên quan đến xe ô tô cũ
- Output: Giá dự đoán của xe ô tô cũ

Giải pháp:

- Dữ liệu:
 - Cào dữ liệu từ website bán xe
 - Thực hiện các công đoạn làm sạch, tiền xử lý và trích xuất đặc trưng của dữ liệu
 - Trực quan hóa dữ liệu
- Mô hình:
 - Linear Regression (Lasso)
 - Extreme Gradient Boosting Regressor
 - Decision Tree Regressor
- Metrics: R Squared, MAE, MAPE



FEATURE ENGINEERING

Biến mục tiêu

Giá xe

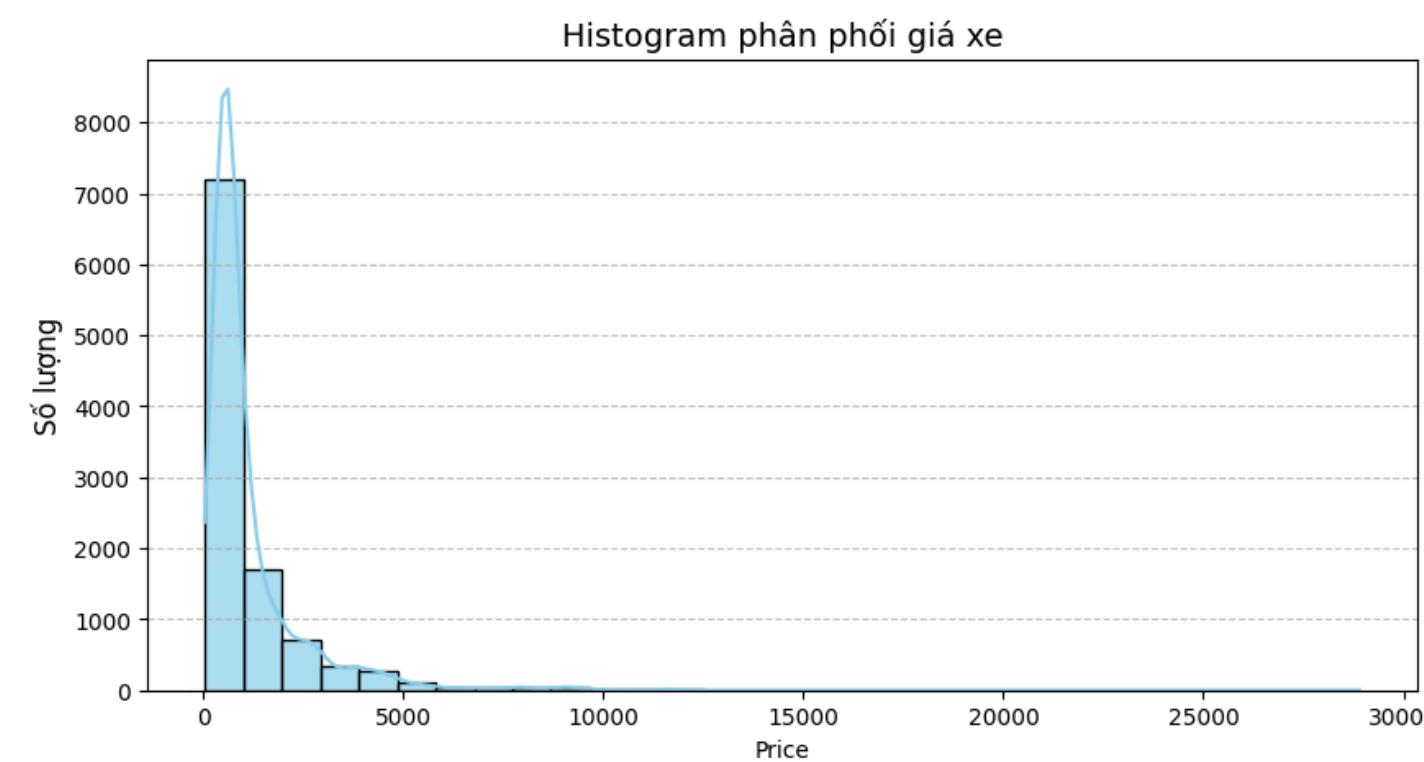
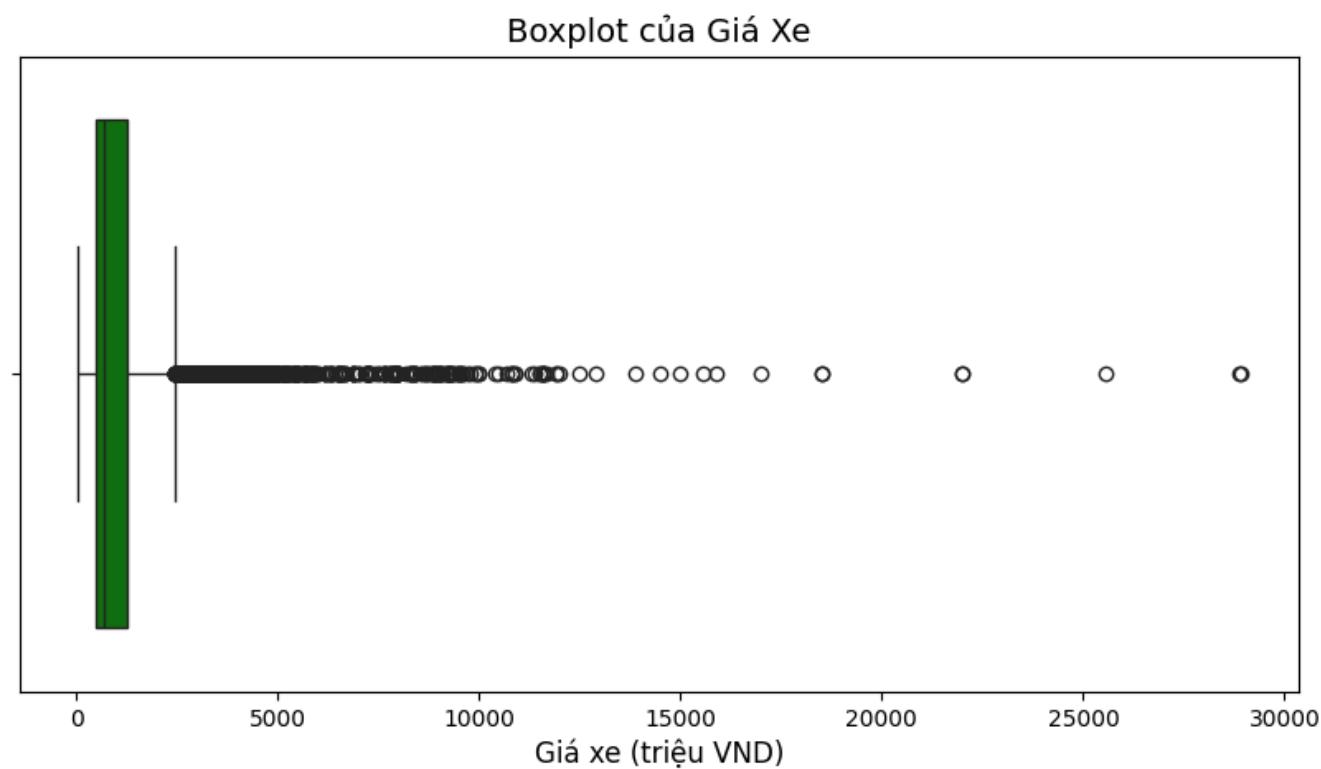
Các đặc trưng

Hãng, Tuổi xe, Dung tích, Dẫn động, Kiểu dáng,
Số Km đã đi, Tên xe

FEATURE ENGINEERING

Biến mục tiêu

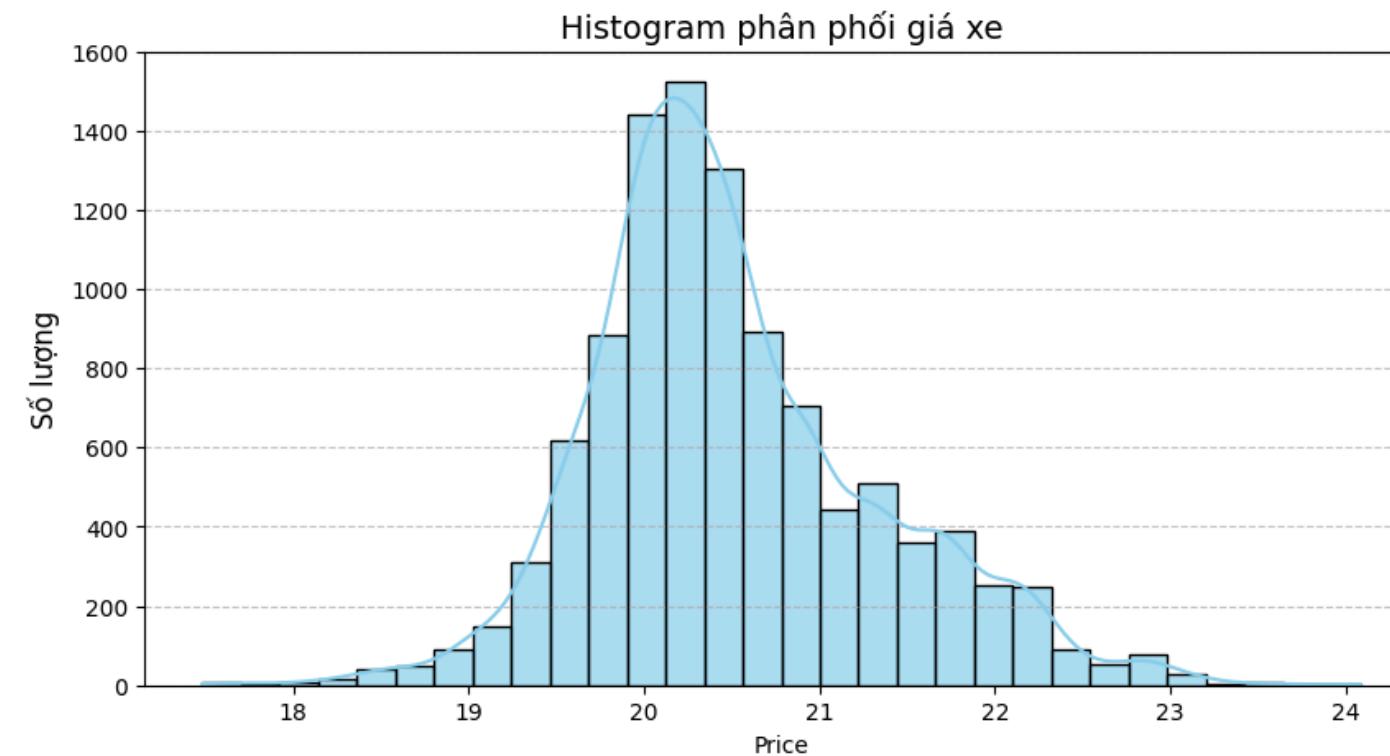
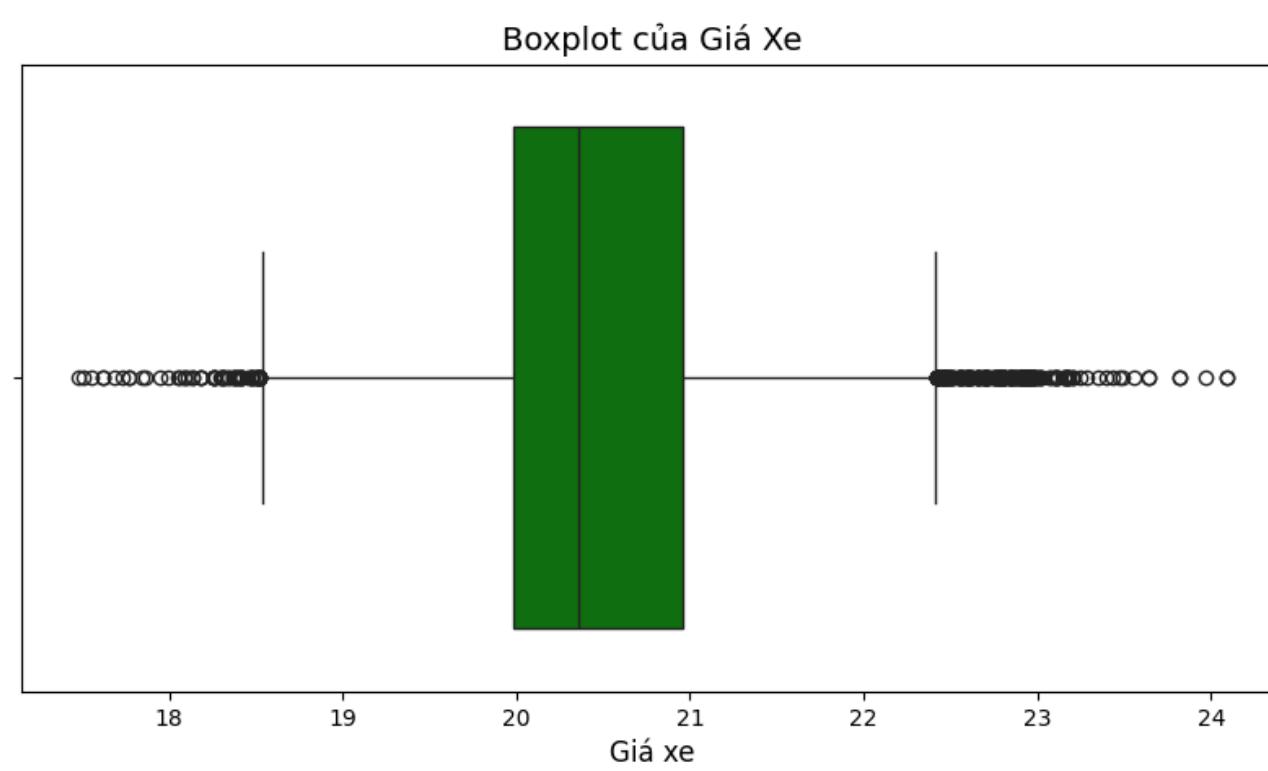
Giá xe trước khi xử lý



FEATURE ENGINEERING

Biến mục tiêu

Giá xe sau khi xử lý



CHIA DỮ LIỆU

```
Y = df['Giá'].copy()

X = df[['Hãng', 'Tuổi xe', 'Dung tích', 'Dẫn động', 'Kiểu dáng',
'Số Km đã đi', 'Tên xe', 'Năm sản xuất']].copy()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=43)
```

CHIA DỮ LIỆU

Xử lý ngoại lệ trên tập train

```
Q1 = X_train['Số Km đã đi'].quantile(0.25)
Q3 = X_train['Số Km đã đi'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

train_mask = (X_train['Số Km đã đi'] >= lower_bound) & (X_train['Số Km đã đi'] <= upper_bound)
X_train = X_train[train_mask]
y_train = y_train[train_mask]
```

CHIA DỮ LIỆU

Xử lý ngoại lệ trên tập train

```
temp_df = X_train[["Năm sản xuất", "Số Km đã đi"]].copy()
temp_df.loc[temp_df["Số Km đã đi"] == 0, "Số Km đã đi"] = np.nan
temp_df_sorted = temp_df.sort_values(by="Năm sản xuất")

known_x = temp_df_sorted["Năm sản xuất"][temp_df_sorted["Số Km đã đi"].notna()]
known_y = temp_df_sorted["Số Km đã đi"][temp_df_sorted["Số Km đã đi"].notna()]

from scipy.interpolate import UnivariateSpline
spline = UnivariateSpline(known_x, known_y, k=2, ext=3)
missing_mask = temp_df_sorted["Số Km đã đi"].isna()
predicted_values = spline(temp_df_sorted.loc[missing_mask, "Năm sản xuất"])
temp_df_sorted.loc[missing_mask, "Số Km đã đi"] = predicted_values

if temp_df_sorted["Số Km đã đi"].isna().sum() > 0:
    temp_df_sorted["Số Km đã đi"] = temp_df_sorted["Số Km đã đi"].interpolate(method='linear')
    if temp_df_sorted["Số Km đã đi"].isna().sum() > 0:
        median_mileage = temp_df_sorted["Số Km đã đi"].median()
        temp_df_sorted["Số Km đã đi"] = temp_df_sorted["Số Km đã đi"].fillna(median_mileage)

temp_df_sorted["Số Km đã đi"] = temp_df_sorted["Số Km đã đi"].apply(lambda x: max(x, 500))
X_train.loc[temp_df_sorted.index, "Số Km đã đi"] = temp_df_sorted["Số Km đã đi"].round().astype(int)
```

CHIA DỮ LIỆU

Chuẩn hóa dữ liệu trên tập train, áp dụng cho tập test, tránh rò rỉ dữ liệu

```
X_train_copy = X_train.copy()
X_test_copy = X_test.copy()
y_train_copy = y_train.copy()
y_test_copy = y_test.copy()

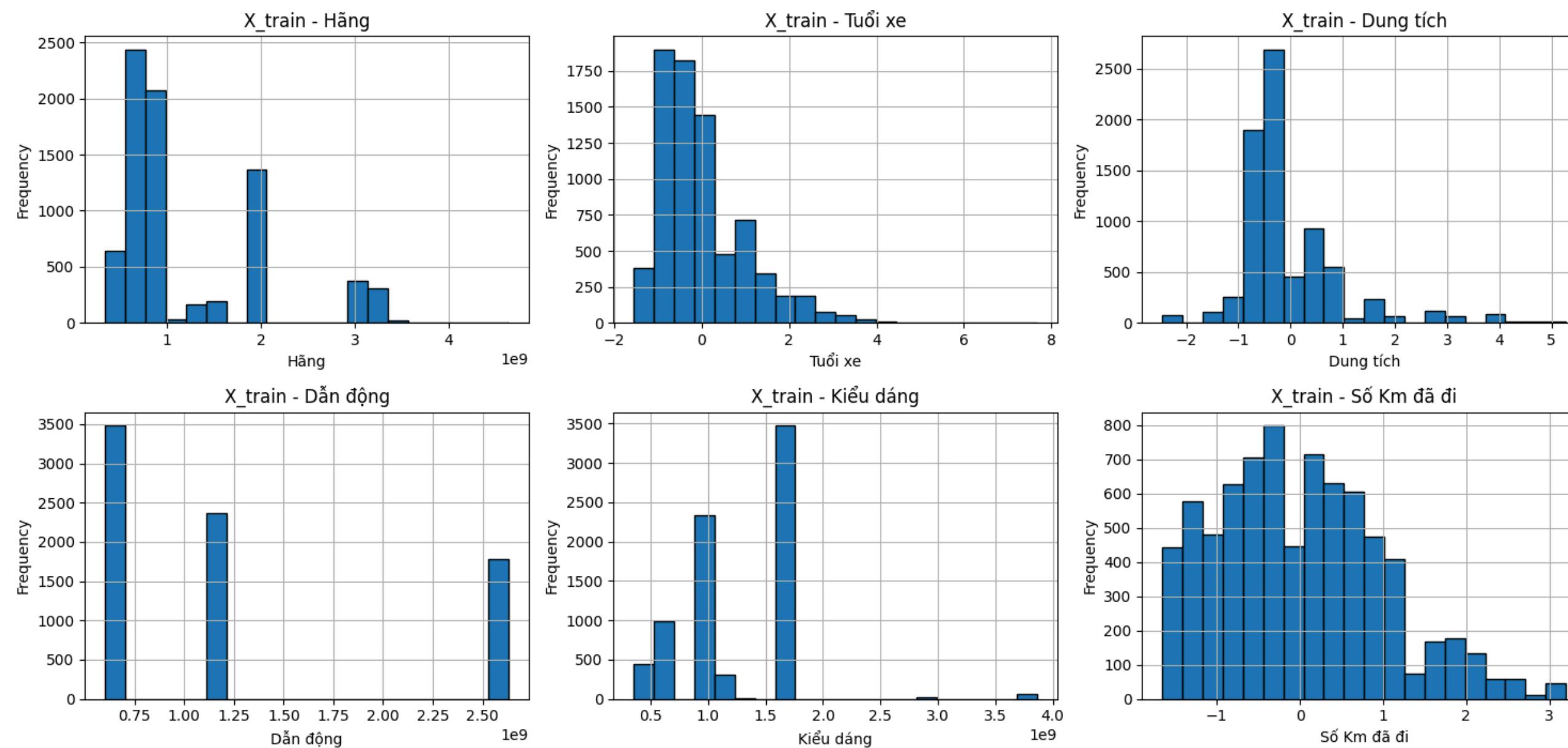
numerical_cols = X_train_copy.select_dtypes(include=["number"]).columns
categorical_cols = X_train_copy.select_dtypes(exclude=["number"]).columns

ss = StandardScaler()
ss.fit(X_train_copy[numerical_cols])
X_train_copy[numerical_cols] = ss.transform(X_train_copy[numerical_cols])
X_test_copy[numerical_cols] = ss.transform(X_test_copy[numerical_cols])

encoder = ce.TargetEncoder()
encoder.fit(X_train_copy[categorical_cols], y_train_copy)
X_train_copy[categorical_cols] = encoder.transform(X_train_copy[categorical_cols])
X_test_copy[categorical_cols] = encoder.transform(X_test_copy[categorical_cols])
```

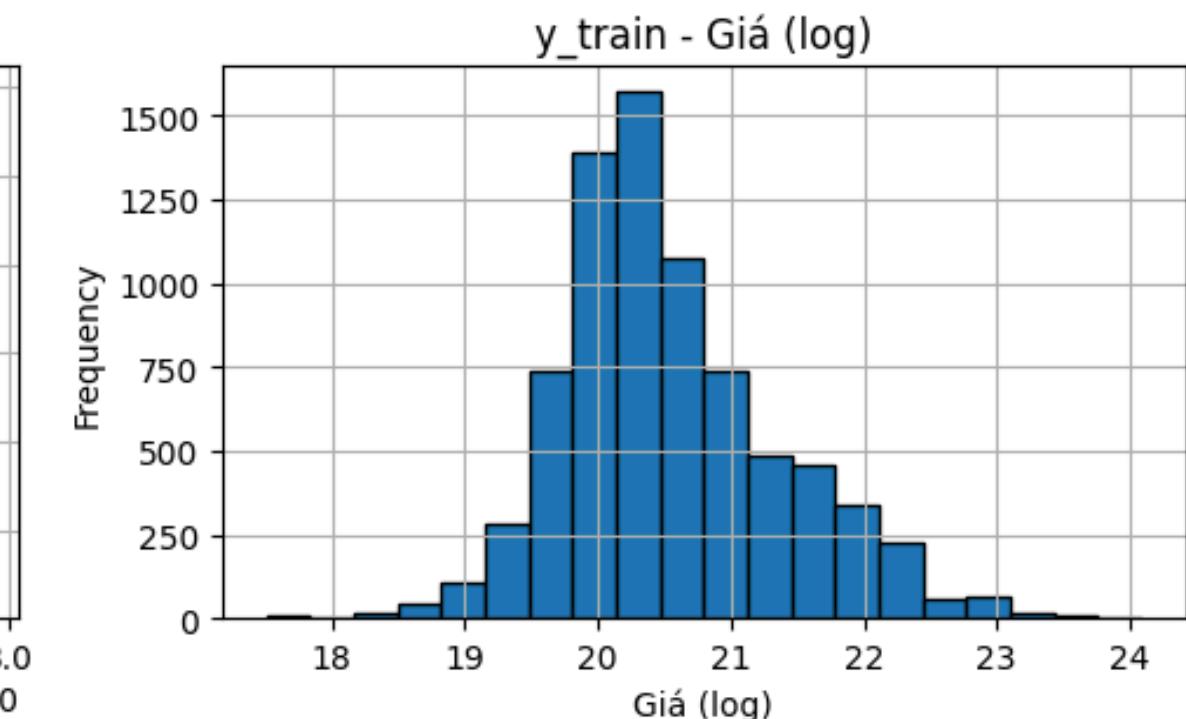
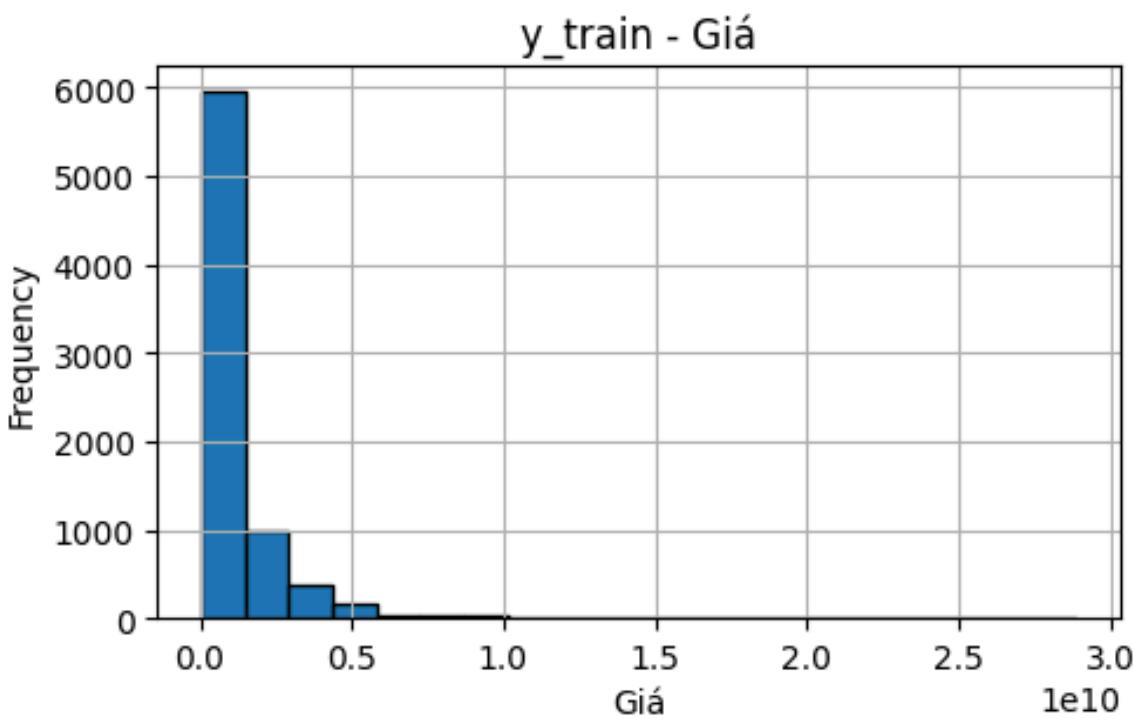
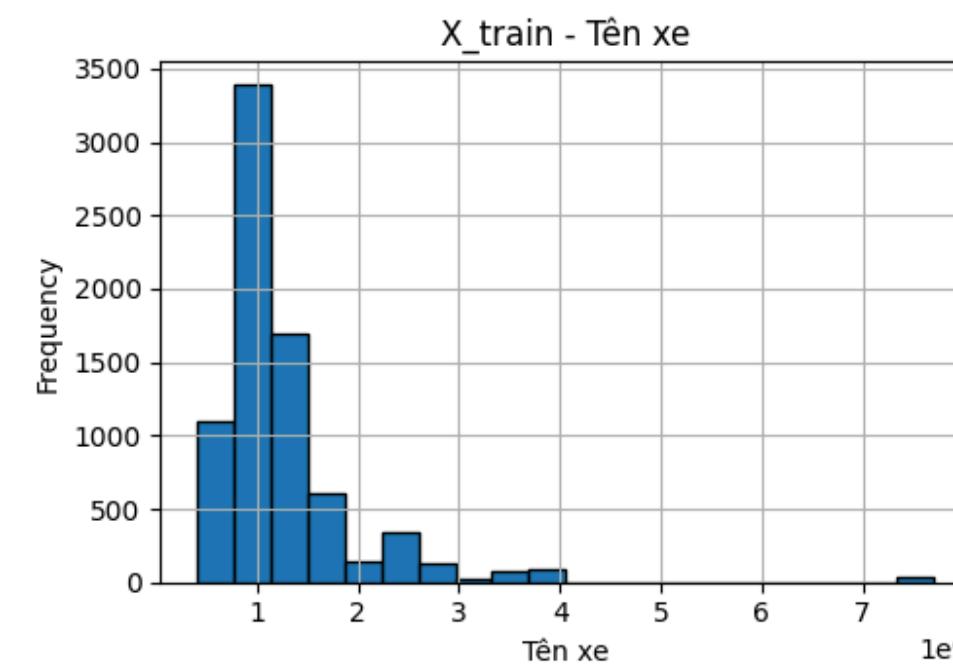
CHIA DỮ LIỆU

Phân bố dữ liệu trên tập train



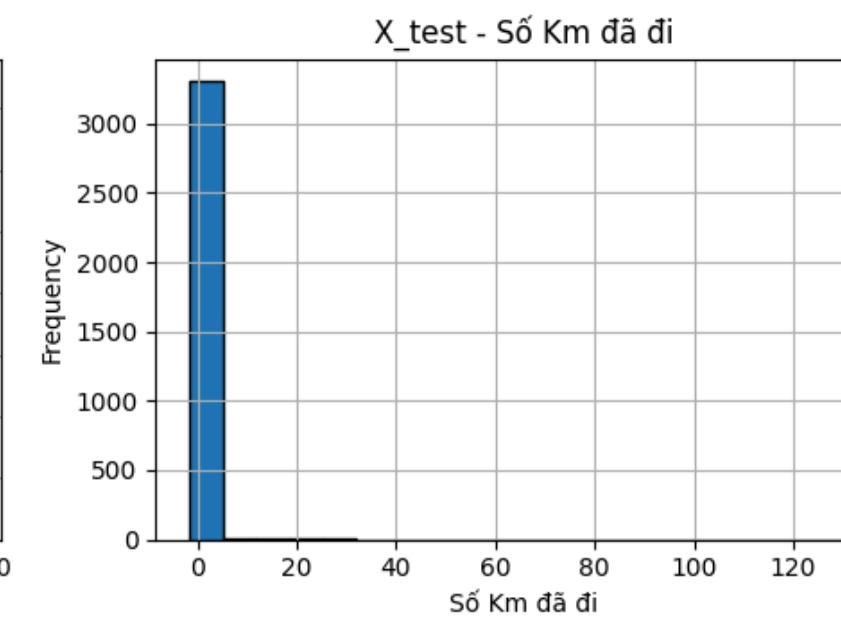
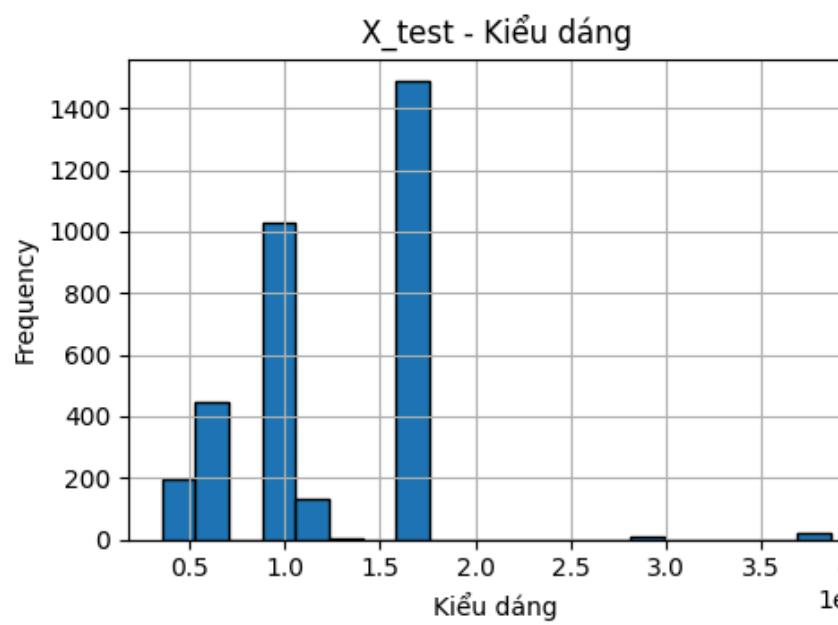
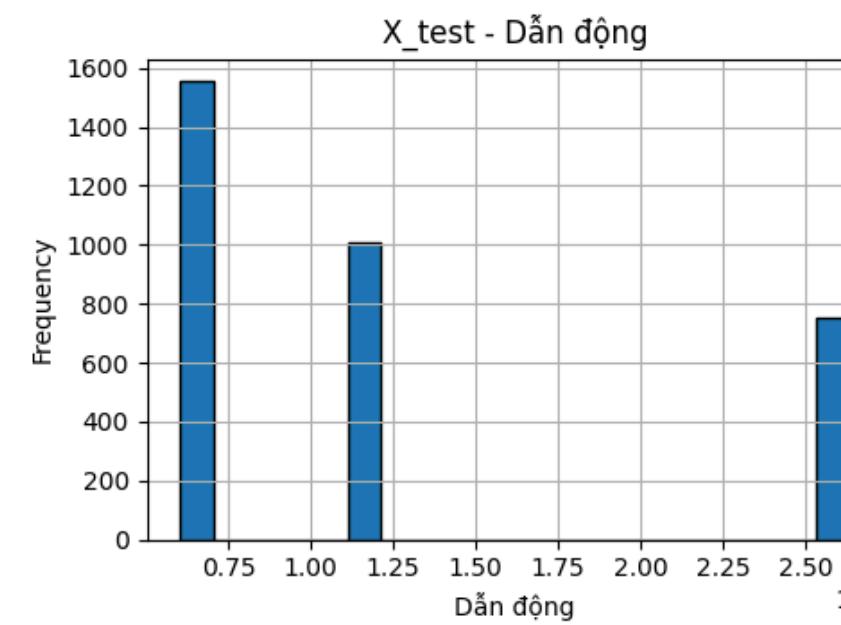
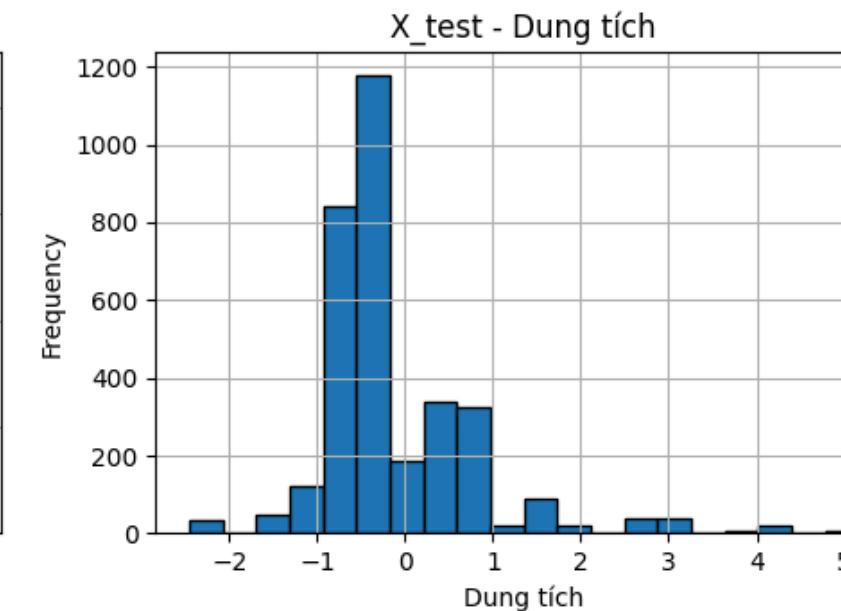
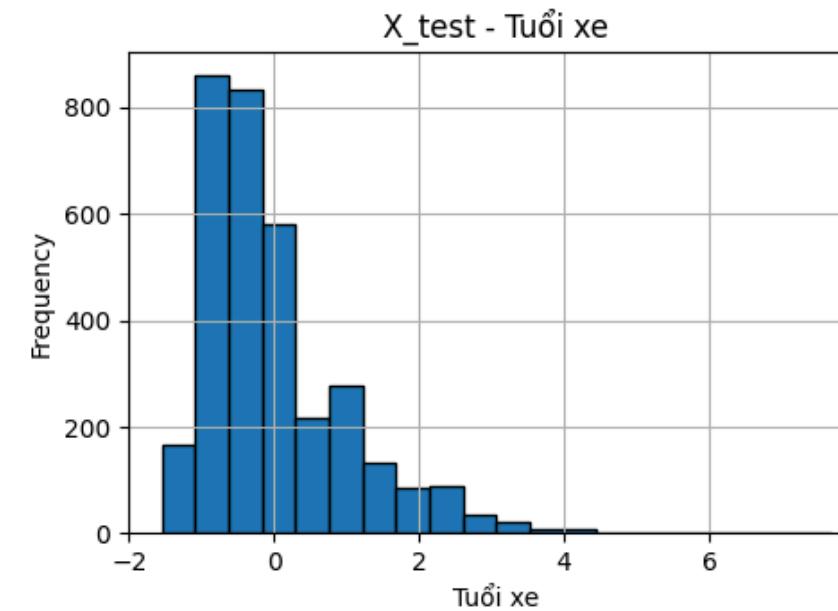
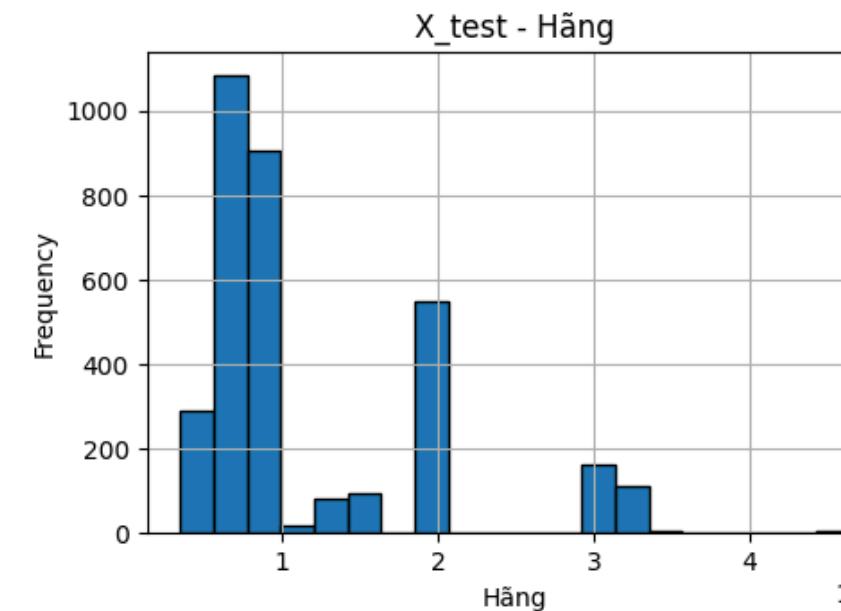
CHIA DỮ LIỆU

Phân bố dữ liệu trên tập train



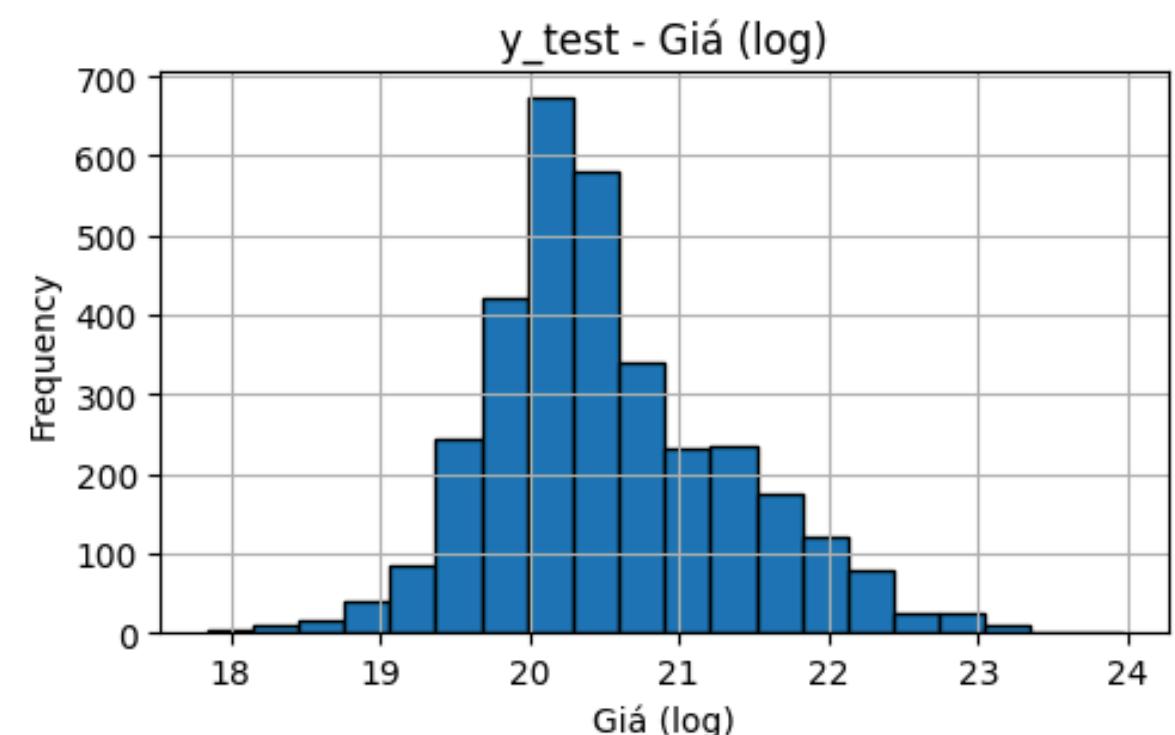
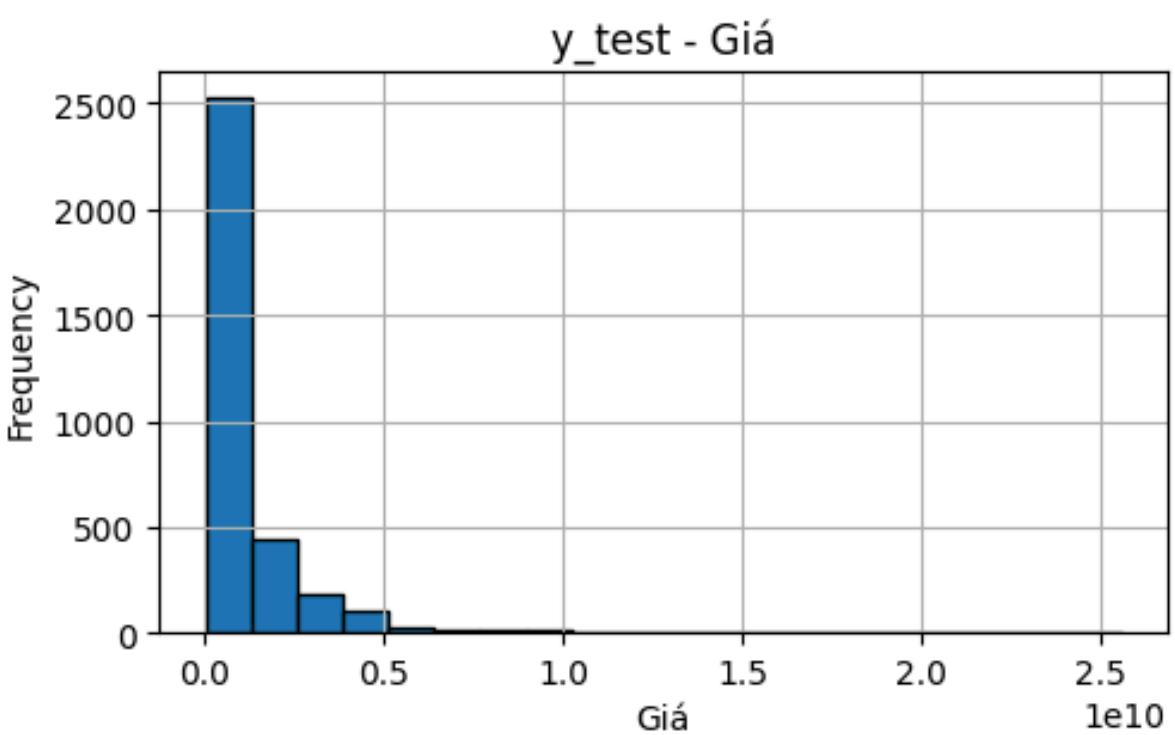
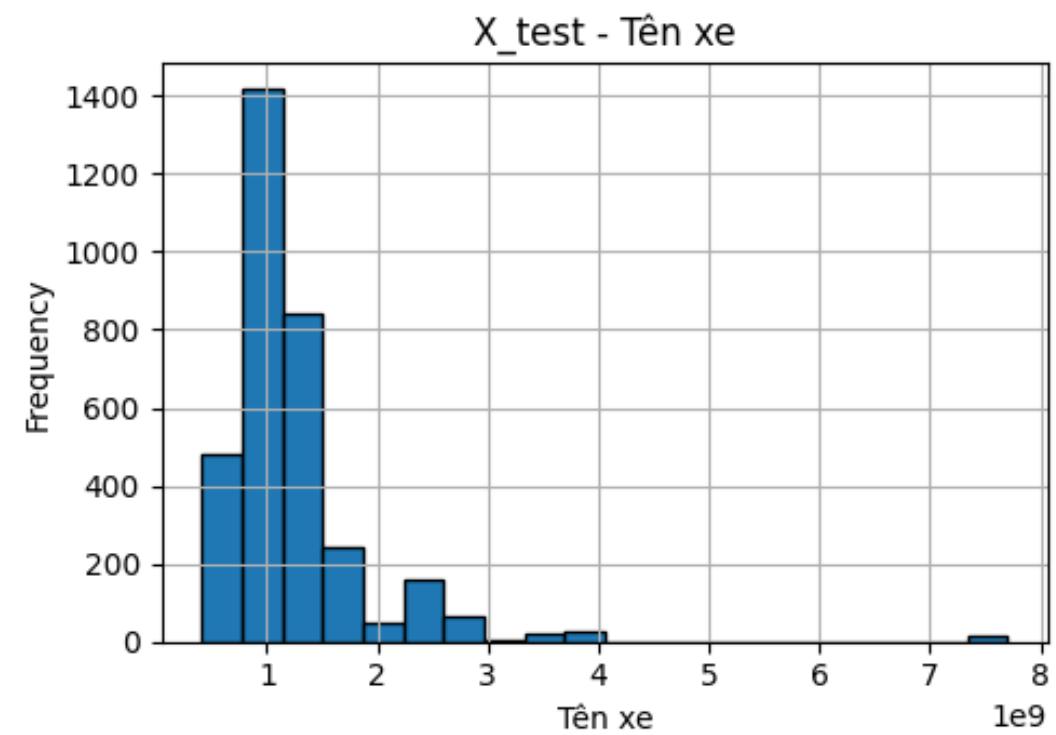
CHIA DỮ LIỆU

Phân bố dữ liệu trên tập test



CHIA DỮ LIỆU

Phân bố dữ liệu trên tập test



MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

Mô hình:

Linear Regression (Lasso)

Extreme Gradient Boosting Regressor

Decision Tree Regressor

Metrics:

R Squared, MAE, MAPE

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

RandomizedSearchCV: Tìm bộ siêu tham số tốt nhất cho mô hình ML

1. Xác định không gian siêu tham số (param_distributions).
2. Lấy ngẫu nhiên một số tổ hợp (n_iter)
3. Huấn luyện và đánh giá mô hình cho từng tổ hợp. Với mỗi tổ hợp được chọn:
 - a. Huấn luyện mô hình với các siêu tham số đó.
 - b. Dùng cross-validation (cv) để đánh giá hiệu suất (ví dụ: chia 5-fold).
 - c. Tính điểm (như accuracy, neg_mean_squared_error, ...) trung bình qua các fold.
4. Chọn tổ hợp có điểm tốt nhất: Sau khi thử n_iter tổ hợp, nó chọn ra tổ hợp nào cho hiệu suất tốt nhất.

```
xg = XGBRegressor(verbosity= 0)

n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
learning_rate=[0.05,0.1,0.15,0.20]

parameter_grid = {
    'n_estimators': n_estimators,
    'max_depth':max_depth,
    'learning_rate':learning_rate,
}

xg_rs = RandomizedSearchCV(estimator=xg, param_distributions=parameter_grid)
```

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

XGBOOST:

Siêu tham số	Ý nghĩa	Giá trị thường dùng
n_estimators	Số lượng cây boosting	100 – 1500
max_depth	Độ sâu tối đa của cây	2 – 15
learning_rate	Hệ số học, càng nhỏ thì học càng chậm nhưng chính xác hơn	0.01 – 0.3
subsample	Tỷ lệ mẫu dùng để huấn luyện mỗi cây	0.5 – 1

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

🧱 Bước 0: Khởi tạo mô hình

Tùy vào bài toán:

- **Hồi quy** (regression):

$$\hat{y}_i^{(0)} = \text{mean}(y)$$

- **Phân loại nhị phân** (binary classification):

$$\hat{y}_i^{(0)} = \text{logit}(p) = \log\left(\frac{p}{1-p}\right), \quad p = \text{mean}(y)$$

(ví dụ, nếu 60% mẫu là nhãn 1 thì $p = 0.6 \rightarrow \text{logit}(0.6) \approx 0.405$)

- Với phân loại, XGBoost hoạt động trên log-odds, chứ không phải xác suất.

⟳ Bước 1: Duyệt từng vòng boosting (vòng lặp huấn luyện)

Mục tiêu là **tối thiểu hóa hàm mất mát tổng**:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t)$$

Dùng khai triển Taylor bậc hai:

- $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ là đạo hàm bậc 1 (gradient),
- $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$ là đạo hàm bậc 2 (Hessian).

📌 Ví dụ cụ thể: **Phân loại nhị phân** (Log loss)

- Loss gốc:

$$\mathcal{L}(y_i, \hat{y}) = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i) \quad \text{với } p_i = \sigma(\hat{y}) = \frac{1}{1 + e^{-\hat{y}}}$$

- Từ đó ta có:

$$g_i = \sigma(\hat{y}_i^{(t-1)}) - y_i \quad h_i = \sigma(\hat{y}_i^{(t-1)}) \cdot (1 - \sigma(\hat{y}_i^{(t-1)}))$$

📌 Với hồi quy bình phương:

- $\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$
 $\rightarrow g_i = \hat{y}_i^{(t-1)} - y_i, h_i = 1$

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ XGBOOST

 Bước 3: Xây dựng một cây quyết định mới (f_t)

 Công thức Gain:

$$\text{Gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

- G_L, H_L : tổng gradient và Hessian ở nhánh trái.
- G_R, H_R : tổng gradient và Hessian ở nhánh phải.
- λ, γ : tham số regularization giúp tránh overfitting.

 XGBoost chọn điểm chia nào làm giảm loss nhiều nhất.

 Bước 5: Cập nhật dự đoán

Sau khi có cây mới f_t , cập nhật dự đoán:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta f_t(x_i)$$

- η : learning rate (shrinkage) để điều chỉnh mức độ sửa sai (giúp ổn định và tránh overfitting).

 Bước 4: Gán giá trị cho mỗi lá

Sau khi cây được tạo ra, mỗi lá cần một giá trị dự đoán w_j . Ta chọn w_j sao cho làm giảm loss nhiều nhất:

$$w_j = -\frac{G_j}{H_j + \lambda}$$

→ Càng nhiều gradient (lỗi cần sửa) trong một lá, giá trị của lá càng lớn.

Dự đoán cho 1 điểm dữ liệu mới"

Tức là:

$$\hat{y}_i = \sum_{t=1}^T \eta \cdot f_t(x_i)$$

Mỗi cây đưa ra một phần "đóng góp" (delta prediction), và mô hình cộng dồn tất cả các phần đó lại.

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

XGBOOST: Training and results

```
from xgboost import XGBRegressor
from sklearn.model_selection import RandomizedSearchCV

xg = XGBRegressor(verbosity= 0)

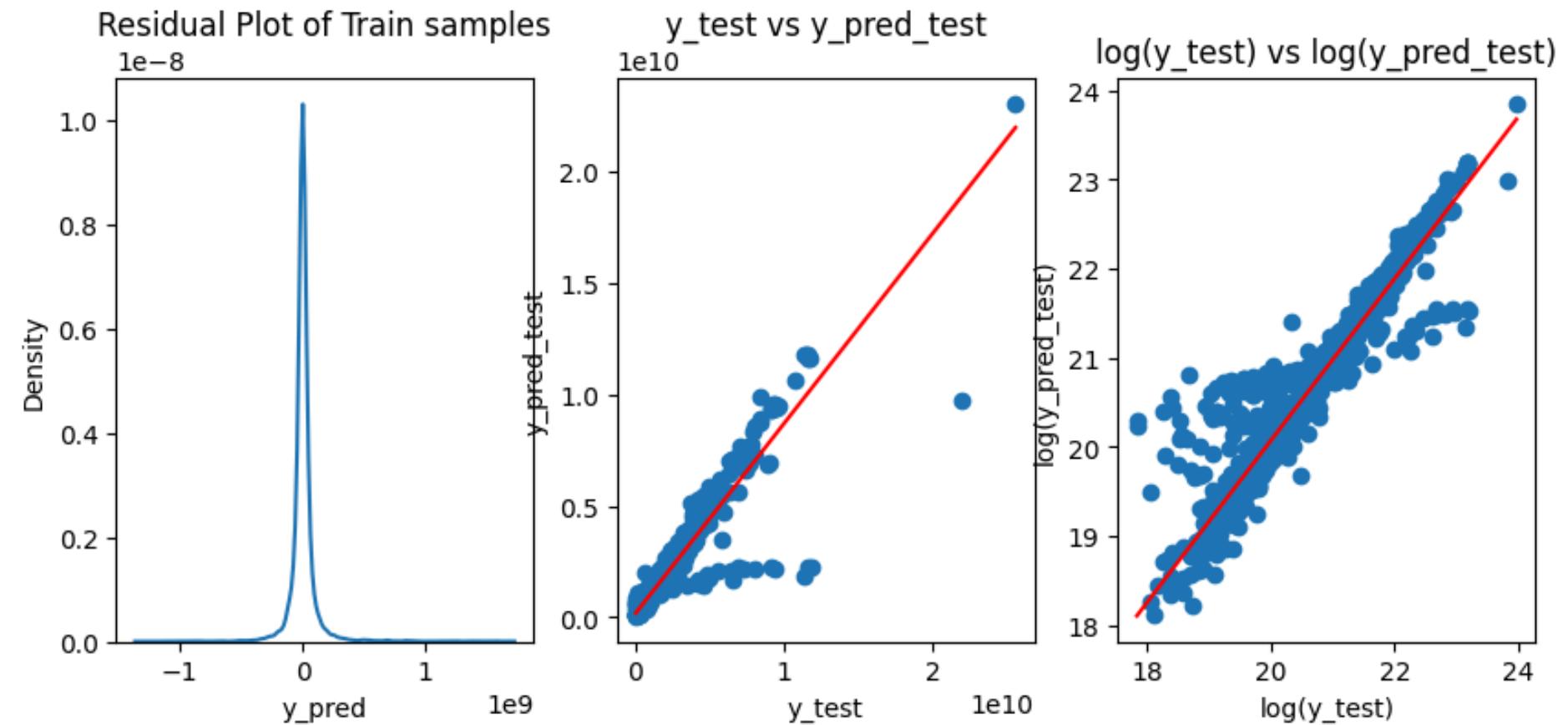
n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
learning_rate=[0.05,0.1,0.15,0.20]

parameter_grid = {
    'n_estimators': n_estimators,
    'max_depth':max_depth,
    'learning_rate':learning_rate,
}

xg_rs = RandomizedSearchCV(estimator=xg, param_distributions=parameter_grid)

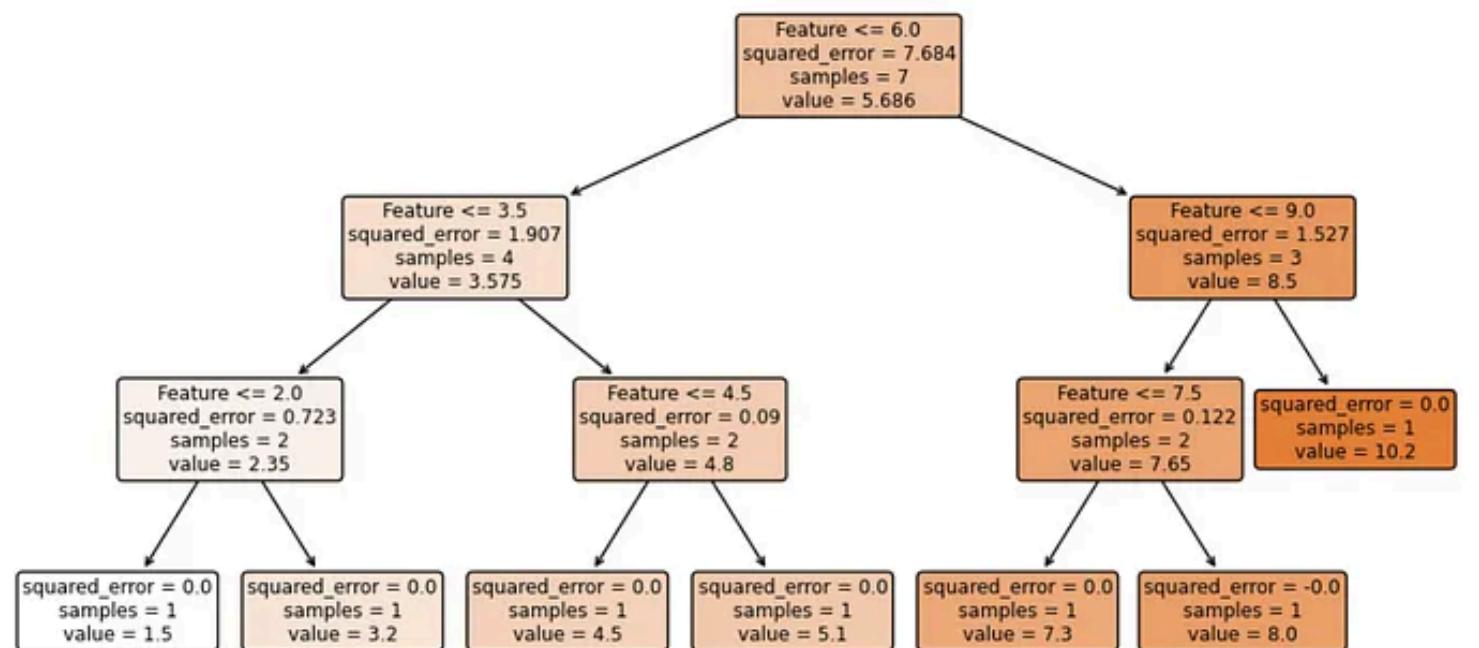
car_price_prediction_model(xg_rs)
```

```
Training set R2 scores: 0.9954044222831726
Test set R2 scores: 0.8849207758903503
Training set R2 scores (log): 0.9947573197056622
Test set R2 scores (log): 0.9212234172640466
MAE scores: 112608472.0
MAPE scores: 12.400132753143717
Training cross validation score: [0.9148683  0.97647923  0.62840062  0.82202554  0.88193834]
Training cross validation mean score: 0.8447424054145813
-----
Best model's hyperparameters: {'n_estimators': 900, 'max_depth': 5, 'learning_rate': 0.05}
```



MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

DecisionTreeRegressor:



Siêu tham số	Ý nghĩa	Giá trị thường dùng
max_depth	Độ sâu tối đa của cây	3 – 20
min_samples_split	Số mẫu tối thiểu để chia một node	2 – 20
min_samples_leaf	Số mẫu tối thiểu ở một lá	1 – 10
criterion	Hàm mất mát để chia	"squared_error" (mặc định), "absolute_error"

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

DecisionTreeRegressor: Loss function

Mean Squared Error:

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

Mean Absolute Error:

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

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

DecisionTreeRegressor: Training and results

```
from sklearn.tree import DecisionTreeRegressor
from scipy.stats import randint
from sklearn.model_selection import RandomizedSearchCV

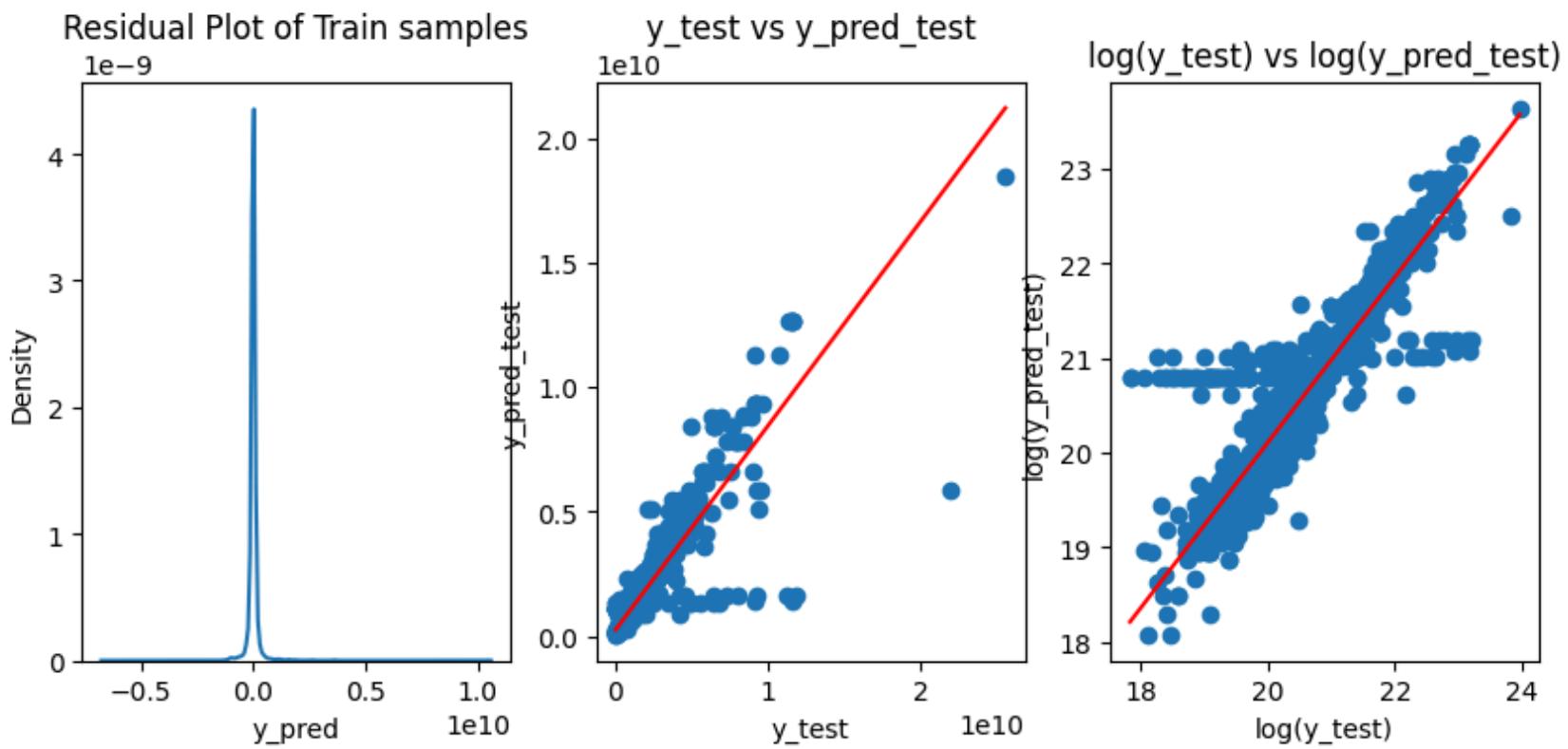
dt = DecisionTreeRegressor()

param_dist = {
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'],
    'max_depth': randint(1, 20),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 20)
}

dt_rs = RandomizedSearchCV(estimator=dt, param_distributions=param_dist)

car_price_prediction_model(dt_rs)
```

```
Training set R2 scores: 0.9528868684023174
Test set R2 scores: 0.8218942182218828
Training set R2 scores (log): 0.9796479481941301
Test set R2 scores (log): 0.8552592780887206
MAE scores: 160117039.64581242
MAPE scores: 20.10206380747247
Training cross validation score: [0.8629696  0.84294779  0.53018292  0.74820553  0.85525408]
Training cross validation mean score: 0.7679119844733728
-----
Best model's hyperparameters: {'criterion': 'friedman_mse', 'max_depth': 17, 'min_samples_leaf': 8, 'min_samples_split': 17}
```



MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

Lasso Regression:

Siêu tham số	Ý nghĩa	Giá trị thường dùng
alpha	Mức độ regularization (L1)	0.0001 – 1
max_iter	Số vòng lặp tối đa khi tối ưu	1000 – 10000
tol	Sai số dừng thuật toán	1e-4 đến 1e-2

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

Lasso Regression: Loss function

$$\mathcal{L} = \frac{1}{2n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^p |\beta_j|$$

Trong đó:

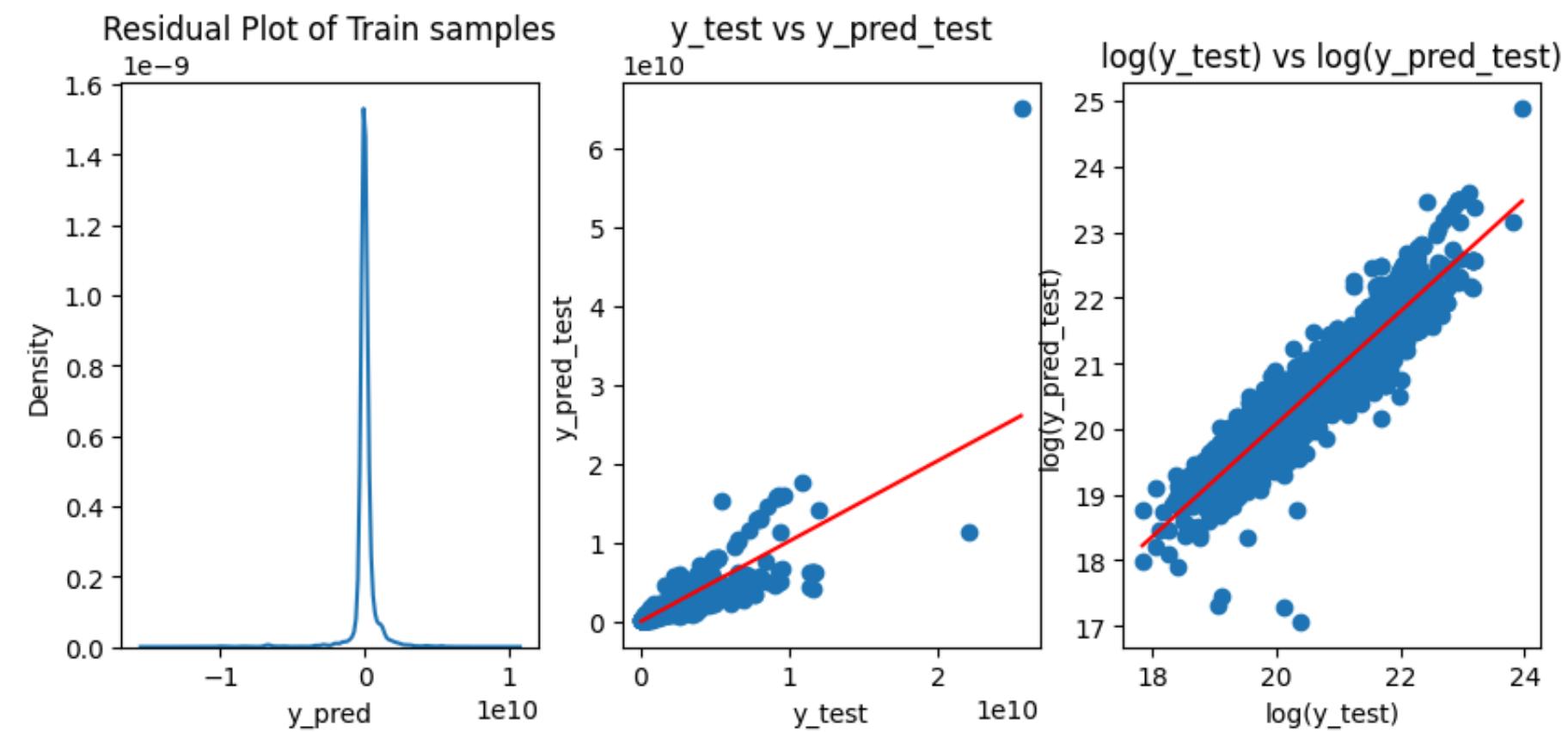
- $\hat{y}_i = X_i \cdot \beta$: dự đoán tuyến tính
- α : siêu tham số điều chỉnh mức phạt (L1 penalty)
- β_j : hệ số của đặc trưng thứ j
- L1 penalty làm cho một số β_j bằng 0 → **lựa chọn đặc trưng tự động**

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

Lasso Regression: Training and results

```
ls = Lasso()  
|  
param_distributions = {  
    'alpha': loguniform(1e-4, 1e4),  
    'tol': [1e-5, 1e-4, 1e-3],  
    'max_iter': [1000, 2000, 5000]  
}  
  
ls_rs = RandomizedSearchCV(  
    estimator=ls,  
    param_distributions=param_distributions,  
)  
  
car_price_prediction_model(ls_rs)
```

```
Training set R2 scores: 0.6746240481927784  
Test set R2 scores: 0.505867699681872  
Training set R2 scores (log): 0.8665639641773221  
Test set R2 scores (log): 0.8561918167022482  
MAE scores: 327279577.57426363  
MAPE scores: 23.386748568396996  
Training cross validation score: [0.6456035 0.63639502 0.63269736 0.62952986 0.71516542]  
Training cross validation mean score: 0.6518782335519469  
-----  
Best model's hyperparameters: {'alpha': np.float64(0.00013198493689305476), 'max_iter': 1000, 'tol': 0.0001}  
-----
```



MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

SO SÁNH 3 MÔ HÌNH

STT	Mô hình	R ² (Train)	R ² (Test)	R ² (Train, log)	R ² (Test, log)	MAE	MAPE
1	Linear Regression (Lasso)	0.674624	0.505868	0.866564	0.856192	3.27×10^8	23.39%
2	Extreme Gradient Boosting Regressor	0.995404	0.884921	0.994757	0.921223	1.13×10^8	12.40%
3	Decision Tree Regressor	0.952887	0.821894	0.979648	0.855259	1.60×10^8	20.10%

XGBoost tốt nhất, tất cả chỉ số đều tốt hơn 2 mô hình còn lại.

Lasso tệ trên giá gốc (R^2 Test = 0.506) nhưng tốt trên log giá (R^2 Test log = 0.856)

MÔ HÌNH HÓA VÀ ĐÁNH GIÁ

Giải thích

- Dữ liệu trong tập test nhiều giá trị ngoại lệ, Lasso nhạy cảm với giá trị ngoại lệ, còn XGBoost và DecisionTree mạnh mẽ với những giá trị ngoại lệ hơn
- XGBoost tốt hơn DecisionTree vì nó xây dựng các cây quyết định tuần tự, mỗi cây sửa lỗi của cây trước bằng cách tối ưu hóa hàm mất mát, trong khi DecisionTree chỉ dùng 1 cây
- Log giá giảm độ lệch, làm phân phối gần chuẩn, giúp tất cả mô hình đạt R^2 Test log cao
- Lasso áp dụng phạt L1 ($|w|$) để làm một số trọng số đặc trưng về 0, giúp giảm overfitting nhưng cũng làm mô hình đơn giản hơn. Điều này khiến Lasso nhạy cảm với các mẫu ngoại lệ hoặc nhiễu
- Lasso nhạy cảm với phân phối lêch của giá gốc. XGBoost và Decision Tree, là các mô hình phi tuyến, không phụ thuộc vào phân phối chuẩn, nên ít bị ảnh hưởng bởi độ lệch