ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

---

# Báo cáo bài tập thực hành 1

**Đề tài: Dự đoán lương dựa theo dữ liệu việc làm**

---

**Môn học: Nhập môn học máy**

*Sinh viên thực hiện:*

Tô Hữu Danh (23127336)

Nguyễn Đăng Hưng (23127050)

Lê Phú Cường (23127164)

Nguyễn Bá Đăng Khoa (23127392)

*Giáo viên hướng dẫn:*

Thầy Võ Nhật Tân

Ngày 11 tháng 11 năm 2025

# Mục lục

# Danh sách bảng

# Danh sách hình vẽ

# 1 Mô tả đồ án và động lực chọn đề tài

# 2 Mô tả và kỹ thuật dữ liệu

## 2.1 Mô tả dữ liệu

## 2.2 Tiền xử lí dữ liệu

### 2.2.1 Loại bỏ thuộc tính ID không mang giá trị dự đoán

Trước tiên, ta cần loại bỏ thuộc tính ID (chỉ số định danh của mẫu) khỏi bộ dữ liệu. Thuộc tính ID thường là một mã định danh duy nhất cho mỗi mẫu và không chứa thông tin hữu ích để dự đoán biến mục tiêu. Việc giữ lại ID có thể gây nhiễu cho mô hình học máy do bản chất duy nhất của nó – mỗi giá trị ID chỉ xuất hiện một lần và không liên quan đến nhãn dự đoán. Loại bỏ cột ID giúp giảm số chiều dữ liệu và tránh việc mô hình học máy học những mẫu giả tạo từ ID.

### 2.2.2 Khảo sát kích thước và kiểu dữ liệu

Tiếp theo, ta tiến hành khảo sát tổng quan về bộ dữ liệu nhằm nắm rõ kích thước và kiểu dữ liệu của các trường. Cụ thể:

- Ta dùng phương thức `.shape` để xác định số lượng mẫu và số thuộc tính trong tập dữ liệu. Kết quả dữ liệu có 205 mẫu và 25 thuộc tính.
- Ta dùng phương thức `.info()` của pandas để liệt kê các cột, kiểu dữ liệu từng cột, số lượng giá trị không rỗng cũng như lượng bộ nhớ sử dụng. Kết quả kiểu dữ liệu của các cột phù hợp với dữ liệu, dữ liệu không bị thiếu và không có sai lệch trong định dạng dữ liệu.

### 2.2.3 Kiểm tra dữ liệu trùng lặp

Sau khi xem xét cấu trúc dữ liệu, ta kiểm tra sự trùng lắp trong tập dữ liệu. Bằng việc sử dụng phương thức `.duplicated()` của pandas, ta xác định được các bản ghi không có dữ liệu trùng lắp. Bước kiểm tra này là cần thiết vì nếu dữ liệu trùng lắp quá nhiều có thể làm cho mô hình tin rằng mẫu đó quan trọng hơn, từ đó mô hình có thể bị thiên lệch và giảm khả năng tổng quát hóa

### 2.2.4 Chia dữ liệu thành các tập huấn luyện, xác thực và kiểm thử

Sau các bước làm sạch dữ liệu, ta tiến hình chia bộ dữ liệu thành ba tập: tập huấn luyện (60%), tập xác thực (20%) và tập kiểm thử (20%). Tập huấn luyện được sử dụng để huấn luyện mô hình,

tập validation dùng để điều chỉnh siêu tham số và đánh giá mô hình một cách công bằng trong quá trình phát triển (do mô hình không nhìn thấy tập này khi huấn luyện), và tập kiểm thử được giữ riêng để đánh giá cuối cùng về khả năng tổng quát hóa của mô hình. Việc phân chia theo tỉ lệ 60/20/20 được lựa chọn nhằm đảm bảo đủ dữ liệu cho huấn luyện (giảm phương sai của mô hình) đồng thời vẫn dành một phần hợp lý cho việc xác thực và kiểm thử. Việc giữ tập kiểm thử tách biệt đến cuối cùng mô phỏng tình huống áp dụng mô hình trên dữ liệu thực tế chưa từng được thấy, do đó độ chính xác thu được trên tập này phản ánh sát hơn hiệu năng kỳ vọng trong thực tiễn.

### 2.2.5 Mã hóa one-hot cho biến không có kiểu dữ liệu là số

Đối với các thuộc tính có kiểu dữ liệu là Object mà ta đã khảo sát ở trên, ở bước này ta áp dụng kỹ thuật mã hóa one-hot để biến chúng thành các đặc trưng nhị phân. Phương pháp one-hot encoding sẽ tạo ra các biến giả (dummy variables) cho mỗi hạng mục có thể có của thuộc tính phân loại: mỗi cột mới đại diện cho một hạng mục, nhận giá trị 1 nếu mẫu thuộc hạng mục đó và 0 nếu không. Cụ thể, với pandas, chúng tôi sử dụng hàm `pd.get_dummies` trên tập dữ liệu huấn luyện để chuyển đổi mỗi cột phân loại thành nhiều cột dummy tương ứng. Kỹ thuật này cho phép mô hình học máy của nhóm xử lý được dữ liệu phân loại một cách hiệu quả, bởi vì 4 mô hình của nhóm chỉ làm việc với dữ liệu số.

### 2.2.6 Căn chỉnh ma trận đặc trưng giữa các tập dữ liệu bằng reindex

Sau khi thực hiện one-hot encoding, một vấn đề quan trọng là đảm bảo tập validation và tập kiểm thử có cùng cấu trúc đặc trưng như tập huấn luyện. Do quá trình `get_dummies` tạo cột dựa trên các hạng mục xuất hiện trong từng tập, nên các tập khác nhau có thể lệch nhau về số lượng hoặc tên cột dummy (ví dụ: một hạng mục nhất định có thể xuất hiện ở tập kiểm thử mà không có ở tập huấn luyện, hoặc ngược lại).

Để giải quyết việc này, chúng ta sử dụng phương pháp reindex của pandas để căn chỉnh cột của tập validation và kiểm thử theo đúng tập cột của tập huấn luyện. Cụ thể, sau khi mã hóa one-hot trên tập huấn luyện và tập con cần căn chỉnh, chúng ta thực hiện: `X_val = X_val.reindex(columns=X_train.columns, fill_value=0)` (tương tự cho `X_test`). Cách làm này bổ sung những cột còn thiếu trong tập validation/test (nếu tập huấn luyện có cột dummy mà tập khác không có, thì cột đó sẽ được thêm vào tập kia và điền giá trị 0 cho tất cả các hàng), đồng thời loại bỏ những cột thừa

không có trong tập huấn luyện (tương ứng với hạng mục không xuất hiện khi huấn luyện). Kết quả là mọi tập dữ liệu đều có cùng số chiều và tên cột đặc trưng, sẵn sàng để đưa vào mô hình.

Bước căn chỉnh này đặc biệt quan trọng bởi nếu one-hot encoding tạo ra số cột khác nhau giữa tập huấn luyện và tập kiểm thử, mô hình sẽ gặp lỗi hoặc hoạt động sai; thậm chí lỗi này không hiển thị rõ ràng mà có thể chỉ làm giảm độ chính xác dự báo do bất nhất trong ma trận đặc trưng

### 2.2.7 Chuẩn hóa đặc trưng bằng StandardScaler

Cuối cùng, trước khi huấn luyện mô hình, chúng ta tiến hành chuẩn hóa các đặc trưng đầu vào bằng phương pháp StandardScaler. Bộ chuẩn hóa StandardScaler sẽ loại bỏ giá trị trung bình của mỗi đặc trưng và chia cho độ lệch chuẩn, đưa các đặc trưng về cùng một thang đo với trung bình 0 và phương sai đơn vị.

Có hai lợi ích chính từ việc chuẩn hóa dữ liệu:

- các thuật toán học máy hồi quy tuyến tính thường hội tụ nhanh hơn và cho kết quả tốt hơn khi các đặc trưng được đưa về cùng một quy mô
- tránh hiện tượng một số thuộc tính có giá trị tuyệt đối lớn lấn át ảnh hưởng của các thuộc tính khác chỉ vì đơn vị đo hoặc phạm vi khác nhau.

Quá trình chuẩn hóa được thực hiện cẩn trọng để không gây rò rỉ dữ liệu. Cụ thể, chúng ta fit đối tượng StandardScaler trên tập huấn luyện (tính toán trung bình và độ lệch chuẩn của từng thuộc tính dựa trên dữ liệu huấn luyện), sau đó dùng scaler này để transform tập validation và tập kiểm thử theo cùng thông số. Lưu ý, chúng ta tuyệt đối không fit StandardScaler trên toàn bộ dữ liệu hay trên tập kiểm thử, bởi làm như vậy đồng nghĩa với việc sử dụng thông tin của tập kiểm thử trong quá trình chuẩn hóa tập huấn luyện – một dạng "data leakage" có thể dẫn đến đánh giá quá mức hiệu năng mô hình. Thay vào đó, việc chỉ dùng thống kê của tập huấn luyện để chuẩn hóa mọi tập khác đảm bảo rằng quy trình tiền xử lý của chúng ta tuân thủ chặt chẽ nguyên tắc huấn luyện trên dữ liệu quá khứ và kiểm thử trên dữ liệu chưa thấy. Nhờ bước chuẩn hóa này, dữ liệu đầu vào cho mô hình có phân phối ổn định, giúp mô hình học được trọng số tối ưu mà không bị ảnh hưởng bởi khác biệt về thang đo của các thuộc tính.

## 2.3 Chọn lọc thuộc tính

### 2.3.1 Mục tiêu

Với tập dữ liệu được chọn, nếu sử dụng tất cả các thuộc tính có sẵn sẽ vô tình chứa các thuộc tính gây nhiễu, làm cho kết quả sau cùng của mô hình chua phải là tốt nhất. Vì thế, cần xây dựng được một mô hình hôi quy tốt nhất bằng cách xác định một tập con các thuộc tính (hay biến dự đoán) phù hợp $X$ để dự đoán biến mục tiêu $y$.

### 2.3.2 Thách thức

Tuy vậy, quá trình lựa chọn thuộc tính không phải chỉ là áp dụng công thức có sẵn mà phải đảm bảo mô hình vẫn cân bằng được các yếu tố sau:

- ***Độ vừa vặn (Goodness-of-Fit):*** Mô hình phải giải thích được sự biến thiên trong dữ liệu, điều đó sẽ được thể hiện bằng độ lớn của $R^2$.
- ***Tính đơn giản (Parismony):*** Mô hình nên càng đơn giản càng tốt, càng phức tạp (hay số lượng biến lớn) thì càng khó diễn giải và dễ dẫn đến *overfitting*.

Ngoài ra, thách thức về mặt tính toán cũng quan trọng không kém khi với $p$ thuộc tính, sẽ có $2^p$ mô hình tập con có thể. Việc kiểm tra toàn bộ là bất khả thi về mặt tính toán nếu $p$ lớn (từ 40 trở lên).

### 2.3.3 Xây dựng mô hình chọn tập con và đánh giá

Ta cần xây dựng một mô hình chọn lọc từng thuộc tính, do đó mỗi khi mô hình chọn được một thuộc tính thì nó sẽ có thêm một biến dẫn đến cần kỹ thuật nâng cao hơn trong việc so sánh nó với phiên bản cũ của chính nó. Để so sánh các mô hình có số lượng biến khác nhau, không thể chỉ sử dụng $R^2$ (vì chỉ số này sẽ luôn tăng khi thêm biến) mà cần thêm các tiêu chí có "điều khoản phạt" (*penalty*) cho sự phức tạp. Đề xuất tiêu biểu nhất là **Tiêu chí thông tin (*Information Criterion*)** [1], phương pháp này yêu cầu các tiêu chí phải cân bằng trực tiếp giữa độ vừa vặn (đo bằng Log-Likelihood, liên quan đến $RSS$) và số lượng tham số $p$ để tối thiểu hoá giá trị tiêu chí chỉ bằng một con số duy nhất dùng cho việc đánh giá mô hình. Có hai loại chỉ số:

- **AIC (Akaike Information Criterion):** [1] Công thức tổng quát dựa trên Log-Likelihood ($L$) của mô hình. Đối với mô hình hồi quy tuyến tính sử dụng ước tính tổng bình phương phần dư ($RSS$), công thức có thể được viết là

$$AIC = n \ln \frac{RSS}{n} + 2k.$$

Trong đó $\left(n \ln \frac{RSS}{n}\right)$ là thành phần dùng để đo mức độ vừa vặn, $(2k)$ là một khoảng trừng phạt (*penalty*), với $k$ là tổng số tham số mà mô hình phải ước tính (bao gồm tất cả các thuộc tính và hệ số chặn). Với mỗi thuộc tính được thêm vào, $k$ sẽ tăng thêm 1 đơn vị dẫn đến $AIC$ bị tăng 2 đơn vị. Điều đó dẫn đến khi thêm một thuộc tính vào mô hình, $RSS$ sẽ giảm nhưng $2k$ sẽ tăng để phạt mô hình. $AIC$ chỉ giảm (tức là mô hình sẽ tốt hơn) khi mức độ cải thiện $RSS$ đủ lớn để bù lại khoản phạt.

- **BIC (Bayesian Information Criterion):** [1] Tương tự như AIC, nhưng xuất phát từ lý thuyết xác suất Bayes. Được thiết kế để tìm ra mô hình có xác suất cao nhất là "mô hình thực sự" (*true model*) tạo ra dữ liệu. Công thức của BIC rất giống AIC nhưng có một khoản phạt khác biệt

$$BIC = n \ln \frac{RSS}{n} + k \ln n.$$

Trong đó $\left(n \ln \frac{RSS}{n}\right)$ giống hệt $AIC$, khoản phạt $(k \ln n)$ được coi là nặng hơn khá nhiều vì $n$ (số lượng mẫu) thường rất lớn. Điều đó chứng tỏ rằng $BIC$ yêu cầu mô hình đưa ra thuộc tính mang lại sự cải thiện rất lơn cho $RSS$, đây là một điểm có thể được coi là lợi vì khi đó $BIC$ có thể đưa ra mô hình đơn giản hơn hay ít thuộc tính hơn so với $AIC$. Vì vậy, $BIC$ sẽ phù hợp vơi một mô hình cô đọng, dễ diễn giải và tránh các thuộc tính có ảnh hưởng yếu.

### 2.3.4 Thuật toán tìm kiếm tự động

Như đã trình bày ở trên, vì số lượng mô hình cần kiểm tra là quá lớn ($2^p$ mô hình) nên cần áp dụng các thuật toán tìm kiếm tham lam để khám phá không gian mô hình theo cách tối ưu nhất, tránh lãng phí tài nguyên tính toán hay tối thiểu nhất là có thể "khả thi hoá" quy trình tính toán.
Một thuật toán được đề xuất trong sách *"Applied Linear Regression"* là **Forward Selection** [2], với ý tưởng chính là tiếp cận từ dưới lên bằng cách xây dựng mô hình từng bước một từ mô hình

ban đầu chỉ có một biến hằng số. Với mỗi lần lặp, thuật toán sẽ chọn mô hình với bộ thuộc tính mới (vừa mới được thêm một thuộc tính) có chỉ số AIC/BIC tốt hơn so với mô hình trước đó (chỉ số AIC/BIC càng thấp là mô hình càng tốt), điều kiện dừng của thuật toán là khi tất cả các thuộc tính đều đã được kiểm tra.

Để trực quan hơn, dưới đây là mã giả của thuật toán được viết lại từ mã nguồn của tác giả **talhahascelik** được đăng trong một repository Github [3]

**Algorithm 1** Mã giả của thuật toán Forward Selection

$remaining\_features \leftarrow list(X.columns)$
$remaining\_features.remove('intercept')$
$selected\_features \leftarrow ['intercept']$
$current\_best\_bic \leftarrow OLS(y, X[selected\_features]).fit().bic$
**while** $remaining\_features$ is not empty **do**
  $best\_feature\_to\_add \leftarrow None$ ▷ Biến dùng để chọn ra thuộc tính tốt nhất
  $best\_new\_bic \leftarrow current\_best\_bic$ ▷ Biến để tính toán chỉ số BIC tốt nhất
  **for** $feature \in remaining\_features$ **do** ▷ Xét từng thuộc tính trong tập thuộc tính còn lại
    $model\_features \leftarrow selected\_features + [feature]$ ▷ Tạo danh sách các thuộc tính tạm thời gồm thuộc tính đã chọn và thuộc tính đang xét
    $model \leftarrow OLS(y, X[model\_features]).fit()$ ▷ Huấn luyện mô hình OLS mới với bộ thuộc tính tạm thời và lấy BIC mới
    $new\_bic \leftarrow model.bic$
    **if** $new\_bic < best\_new\_bic$ **then** ▷ So sánh với BIC tốt nhất để hoán đổi nếu nó tốt hơn
      $best\_new\_bic \leftarrow new\_bic$
      $best\_feature\_to\_add \leftarrow feature$
    **end if**
  **end for**
  **if** $best\_feature\_to\_add \neq None$ **then** ▷ Nếu có thuộc tính mới được chọn
    $selected\_features.append(best\_new\_to\_add)$ ▷ Thêm vào danh sách được chọn và loại khỏi danh sách thuộc tính còn lại
    $remaining\_features.remove(best\_feature\_to\_add)$
    $current\_best\_bic \leftarrow best\_new\_bic$ ▷ Cập nhật giá trị BIC mới
  **end if**
  **if** $!best\_feature\_to\_add \neq None$ **then return** $selected\_features$ ▷ Dừng khi không có thuộc tính nào làm cho BIC tốt hơn
  **end if**
**end while**

Thuật toán Foward Selection là một giải pháp hoàn toàn khả thi và hợp lí dành cho tập dữ liệu lớn ($p > n$) mà tại đó số thuộc tính nhiều hơn số mẫu. Lựa chọn tiến chỉ yêu cầu đánh giá tối đa $\frac{p(p+1)}{2}$ [2] mô hình so với số lượng mô hình thực sự có là $2^p$. Sự giảm thiểu đáng kể về mặt chi phí tính toán sẽ làm thuật toán trở nên khả thi mặc cho $p$ rất lớn. Hơn nữa, Forward Selection có

một lợi thế cấu trúc so với phương pháp **Backward Elimination** [2] vì nó bắt đầu từ mô hình rỗng (*null model*) và xây dựng dần lên thay vì yêu cầu $n > p$ như Backward Elimination để ước tính mô hình đầy đủ.

Tuy nhiên, hạn chế cơ bản của Forward Selection nằm ở bản chất tham lam (*greedy*) của thuật toán vốn chỉ tìm kiếm giải pháp tối ưu cục bộ (*local optimum*) ở mỗi bước mà không đảm bảo tìm được giải pháp tối ưu toàn cục (*global optimum*). Forward Selection có sự thiếu sót về bước lùi hay nói cách khác là nó không có khả năng quay lui (*backtrack*), một thuộc tính khi đã được thêm vào mô hình sẽ không bao giờ bị loại bỏ ở các bước sau, ngay cả khi nó trở nên thừa thãi. Điều đó dẫn đến tiềm ẩn vấn đề che khuất (*masking problem*), nơi một biến ban đầu được chọn vì nó là đại diện tạm thời tốt nhất cho một cấu trúc dữ liệu, nhưng sau đó trở nên không còn ý nghĩa thống kê khi một biến tương quan mạnh hơn khác được thêm vào. Do đó, không thể đảm bảo rằng mô hình $k$-biến với bộ thuộc tính sau cùng là mô hình $k$-biến tốt nhất.

# 3 Lý thuyết về các mô hình và lựa chọn

## 3.1 Mô hình Linear Regression

## 3.2 Mô hình Linear Regression

Linear regression là một mô hình học có giám sát với khả năng tìm mối liên hệ tuyến tính giữa input và output.

**Vấn đề:** Model này nhận vào một vector $x \in \mathbb{R}^{D+1}$ gồm các đặt trưng của một mẫu và trả về giá trị dự đoán $y \in \mathbb{R}$. Ví dụ: Nhận vào các giá trị thuộc tính của một ngôi nhà như Diện tích sàn ($m^2$), số lượng phòng ngủ, khoảng cách tới trung tâm thành phố (m), tuổi nhà (năm) và trả về dự đoán giá của căn nhà đó.

**Đánh giá hiệu năng:** Mô hình Linear Regression thường được đánh giá bằng Mean Square Error (MSE) trên tập train:

$$MSE_{\text{train}} = \frac{1}{N}\|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

với $\hat{\mathbf{y}}$ là tập giá trị dự đoán, $\mathbf{y}$ là tập giá trị thực tế của tập train.

**Huấn luyện mô hình**: Các phương pháp huấn luyện mô hình Linear Regression được dựa trên cách tính toán vector trọng số $\omega$ để giảm MSE xuống thấp nhất. Một cách làm phổ biến là tính toán đạo hàm của hàm MSE theo $\omega$ và cho nó bằng 0 để tìm cực tiểu.

$$\nabla_{\mathbf{w}}(MSE_{\text{train}}) = \nabla_{\mathbf{w}}(\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{y}^T\mathbf{y}) = 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y}$$

$$\begin{aligned}
\nabla_{\mathbf{w}}(MSE_{\text{train}}) &= 0 \\
\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} &= 0 \\
\mathbf{X}^T\mathbf{X}\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\
\mathbf{w} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}
\end{aligned}$$

## 3.3 Mô hình Ridge

### 3.3.1 Cơ sở lý thuyết

Hồi quy Ridge (***Ridge Regression***) là một kỹ thuật hồi quy tuyến tính đuợc điều chỉnh (*regularized*) nhằm giải quyết hiện tượng quá khớp (*overfitting*) bằng cách dùng các kỹ thuật Chính quy hoá (*Regularization*). Phương pháp này thêm một số hạng (thường là *penalty*) vào hàm mất mát (*loss function*) của mô hình. Thành phần phạt này dùng để đánh giá và kiểm soát độ phức tạp của mô hình

Cụ thể, mô hình Ridge sử dụng kỹ thuật *L2 Regularization*, thay vì chỉ tối thiểu hoá tổng bình phương sai số (*Mean Square Error - MSE*) như mô hình hồi quy tuyến tính cơ bản thì Ridge sẽ tối thiểu hoá một hàm mục tiêu mới:

$$J(w) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda.\|\mathbf{w}\|_2^2 \text{ [4]}$$

Trong đó, $\mathbf{w}$ là vector trọng số của mô hình và $\lambda$ là siêu tham số (*hyperparameter*) điều chỉnh độ mạnh của thành phần phạt $\|\mathbf{w}\|_2^2$.

[5] Việc thêm thành phần phạt sẽ ép các trọng số của mô hình phải giữ ở mức nhỏ, co về gần 0. Điều này làm giảm sự phụ thuộc của mô hình vào bất kỳ một biến đầu vào cụ thể nào, khiến mô hình đơn giản hơn, ít nhạy cảm với nhiễu dữ liệu và tăng khả năng tổng quát hoá. Vì thế, bài toán tối ưu hàm mất mát của hồi quy Ridge thực chất là tối ưu song song hai thành phần bao gồm tổng bình phương sai số và thành phần phạt hay thành phần điều chuẩn (*regularization term*).

- Trường hợp $\lambda = 0$, thành phần điều chuẩn bị tiêu giảm và chúng ta quay về hồi quy tuyến tính.
- Trường hợp $\lambda \approx 0$, thành phần điều chuẩn trở nên ít quan trọng, mức độ kiểm soát quá khớp trở nên kém.
- Trường hợp $\lambda$ lớn, mức độ kiểm soát lên độ lớn của các hệ số ước lượng tăng lên qua đó giảm bớt quá khớp.

Khi $\lambda$ tăng dần, hồi quy Ridge có xu hướng thu hẹp hệ số ước lượng $\mathbf{w}$ từ mô hình.

### 3.3.2 Triển khai thực nghiệm

Mô hình Ridge được triển khai và cấu hình như sau:

- Pipeline: Mô hình được gói trong một `sklearn.pipeline.Pipeline` để chuẩn hoá quy trình xử lý.
- Chuẩn hoá (Scaling): Bước đầu tiên trong pipeline là chuẩn hoá dữ liệu, rất quan trọng đối với các mô hình được chính quy hoá như Ridge, vì thành phần phạt nhạy cảm với sự chênh lệch về thang đo (*scale*) của các biến đầu vào.
- Cấu hình: Mô hình Ridge được khởi tạo với siêu tham số `alpha=33.6` và `random_state=42` (được dùng để đảm bảo kết quả có thể được tái lập).
- Dữ liệu: Không giống như mô hình `baseline_linear` sử dụng bộ dữ liệu được chọn từ phương pháp chọn lọc thuộc tính ở phần trước, mô hình Ridge được huấn luyện trên bộ dữ liệu đầy đủ (đã được lưu lại vào `X_train` và `y_train`).

Siêu tham số `alpha` lúc này sẽ được lựa chọn thông qua các bước sau:

- Định nghĩa không gian tìm kiếm: Một tập hợp các giá trị `alpha` tiềm năng được định nghĩa bằng `numpy.logspace(-4, 3, 20)`, tức là một mảng chứa 20 giá trị thực phân bố theo thang logarit từ $10^{-4}$ đến $10^{3}$. Việc sử dụng thang logarit cho phép khám phá hiệu qua các giá trị ở nhiều bậc độ lớn khác nhau.
- Đóng gói Pipeline: Dùng chính mô hình Ridge để chuẩn hoá dữ liệu trước khi huấn luyện mô hình, mục tiêu là tìm ra và đánh giá $R^2$.
- Kiểm định chéo (*Cross-Validation*): `GridSearchCV` được cấu hình để sử dụng phương pháp kiểm định chéo. Toàn bộ tập dữ liệu huấn luyện sẽ được chia thành 5 phần (*folds*). Quá trình tìm kiếm sẽ lặp lại 5 lần, mỗi lần sẽ có một phàn được giữ lại làm tập validation tạm thời và 4 phần còn lại sẽ dùng để huấn luyện.
- Tiêu chí đánh giá: Chỉ số được sử dụng để đánh giá là $R^2$
- Lựa chọn tối ưu: quy trình `GridSearchCV` tự động huấn luyện và đánh giá mô hình Ridge với từng giá trị `alpha` trong không gian tìm kiếm. Giá trị nào mang $R^2$ trung bình cao nhất (qua 5 lượt) sẽ được chọn làm siêu tham số.

Từ quy trình trên, mô hình Ridge chọn được giá trị `alpha` $= 33.6$.

### 3.3.3 Kết luận

Hồi quy Ridge, thông qua cơ chế chính quy hoá L2, được chọn như một giải pháp trực tiếp và hiệu quả cho vấn đề đa cộng tuyến. Bằng cách chấp nhận một lượng nhỏ độ lệch (*bias*) thông qua việc co rút hệ số để từ đó giảm thiểu đáng kể được phương sai. Kết quả là một mô hình dự đoán ổn định, mạnh mẽ hơn và có khả năng tổng quát hoá tốt hơn trên dữ liệu mới so với mô hình tuyến tính cơ sở.

## 3.4 Mô hình Lasso

### 3.4.1 Cơ sở lý thuyết

Lasso (Least Absolute Shrinkage and Selection Operator) là một phương pháp hồi quy tuyến tính sử dụng chuẩn hóa L1 (L1 regularization). Kỹ thuật này đồng thời thực hiện lựa chọn biến (variable selection) và điều chuẩn hóa (regularization) để nâng cao độ chính xác dự báo và khả năng diễn giải của mô hình. Nói cách khác, Lasso bổ sung một khoản phạt L1 vào hàm mất mát của hồi quy thường, khiến một số hệ số hồi quy bị kéo về 0, từ đó đơn giản hóa mô hình và giúp tránh hiện tượng quá khớp. [6]

Hàm mục tiêu của hồi quy Lasso (với dữ liệu có $N$ mẫu, $p$ đặc trưng) có dạng:

$$\min_{\beta_0,\boldsymbol{\beta}} \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

trong đó vế thứ nhất là lỗi bình phương trung bình (MSE) và vế thứ hai là khoản phạt $L1$ nhân với hệ số điều chuẩn $\lambda$. Tham số $\lambda$ (ký hiệu điều chuẩn, trong scikit-learn tham số này được gọi là `alpha`) kiểm soát độ mạnh của hình phạt $L1$, qua đó quyết định mức độ phức tạp của mô hình. $\lambda$ càng lớn thì mô hình bị phạt càng nhiều: các hệ số $\beta_j$ bị kéo về 0 đáng kể hơn, nhiều hệ số nhỏ gần như bị triệt tiêu; kết quả là mô hình giữ lại rất ít biến quan trọng (tránh overfitting). Ngược lại, $\lambda$ nhỏ chỉ phạt nhẹ, mô hình sẽ giữ lại nhiều đặc trưng hơn (gần với hồi quy thường). Tài liệu từ IBM mô tả: *"Larger values of lambda increase the penalty, shrinking more of the coefficients towards zero; this subsequently reduces the importance of (or altogether eliminates) some of the features from the model, resulting in automatic feature selection. Conversely, smaller values of lambda reduce the effect of the penalty, retaining more features within the model."*[7] (Dịch: Giá trị $\lambda$ lớn làm tăng mức phạt, kéo nhiều hệ số hơn về gần 0; điều này làm giảm tầm quan trọng (hoặc thậm chí loại

bỏ hoàn toàn) một số đặc trưng khỏi mô hình, dẫn đến việc tự động chọn lọc biến. Ngược lại, giá trị $\lambda$ nhỏ làm giảm ảnh hưởng của mức phạt, giữ lại nhiều đặc trưng hơn trong mô hình). Nhìn trên phương diện bias–variance, $\lambda$ đóng vai trò cân bằng giữa độ chệch và phương sai của mô hình. Cũng theo IBM: *"As $\lambda$ increases, the bias increases, and the variance decreases, leading to a simpler model with fewer parameters. Conversely, as $\lambda$ decreases, the variance increases, leading to a more complex model with more parameters. If $\lambda$ is zero, then one is left with an OLS function – that is, a standard linear regression model without any regularization."*[7] (Dịch: Khi $\lambda$ tăng, bias tăng và variance giảm, mô hình đơn giản hơn với ít tham số hơn. Ngược lại, khi $\lambda$ giảm, variance tăng lên, mô hình phức tạp hơn với nhiều tham số hơn. Nếu $\lambda = 0$ thì hàm mục tiêu trở thành OLS – tức là mô hình hồi quy tuyến tính thông thường không có regularization). Trường hợp $\lambda = 0$ nghĩa là không áp dụng phạt, Lasso lúc này tương đương mô hình hồi quy thường và sẽ không có tác dụng chống overfitting; ngược lại, $\lambda$ quá lớn sẽ phạt mạnh đến mức hầu hết các hệ số bị triệt tiêu về 0, mô hình khi đó có thể bị underfitting (thiếu độ linh hoạt). Do đó, việc lựa chọn $\lambda$ tối ưu là rất quan trọng để mô hình đạt hiệu năng cao nhất.

Một ưu điểm nổi bật của Lasso là khả năng chọn lọc đặc trưng tự động nhờ vào chuẩn hóa $L1$. Khoản phạt $L1$ thúc đẩy nghiệm *thưa* (sparse solution), nhiều hệ số hồi quy có thể bị đẩy về đúng bằng 0. Điều này có nghĩa là mô hình Lasso sẽ loại bỏ hẳn những biến không quan trọng, chỉ giữ lại những biến thật sự có đóng góp lớn. Nói cách khác, Lasso vừa giảm overfitting vừa đơn giản hóa mô hình bằng cách bỏ qua các đặc trưng dư thừa. Như IBM mô tả: *"Some variables will shrink exactly to zero, leaving the model with a subset of the most important variables to make predictions."*[7] (Dịch: Một số biến sẽ được kéo về đúng 0, khiến mô hình chỉ còn một tập hợp các biến quan trọng nhất để dự đoán). Nhờ đó, Lasso đặc biệt hữu ích khi xử lý dữ liệu có số lượng đặc trưng rất lớn hoặc có nhiều biến ít liên quan – mô hình sẽ tự động bỏ qua những biến ít liên quan, giảm nguy cơ overfitting và cải thiện tính diễn giải (model interpretability) do mô hình trở nên gọn nhẹ hơn.

### 3.4.2 Triển khai thực nghiệm

Trong thực nghiệm, chúng tôi xây dựng một pipeline gồm hai bước: (1) Chuẩn hóa dữ liệu bằng `StandardScaler` và (2) Hồi quy Lasso (scikit-learn). Việc chuẩn hóa thang đo các đặc trưng là cần thiết trước khi áp dụng Lasso, nhằm đảm bảo các hệ số bị phạt công bằng giữa các đặc trưng. Một blog khoa học dữ liệu nhấn mạnh: *"It is crucial to scale (e.g. StandardScaler) input features because regression models are sensitive to them."*[8] (Dịch: Việc chuẩn hóa các đặc trưng đầu vào (ví dụ

dùng StandardScaler) là cực kỳ quan trọng vì các mô hình hồi quy rất nhạy cảm với đặc trưng có đơn vị hay độ lớn khác nhau). Nếu không chuẩn hóa, đặc trưng có độ lớn lớn sẽ bị phạt nặng hơn đặc trưng nhỏ, dẫn đến mức phạt $L1$ không đồng đều và ảnh hưởng xấu đến kết quả hồi quy Lasso. Do đó, toàn bộ features được chuẩn hóa về trung bình 0 và phương sai 1 trước khi huấn luyện mô hình.

Tiếp theo, để tìm giá trị điều chuẩn tối ưu cho mô hình Lasso, chúng tôi sử dụng phương pháp tìm kiếm lưới kết hợp cross-validation. Cụ thể, chúng tôi thực hiện `GridSearchCV` (5-fold cross-validation, scoring theo $R^2$) trên tham số $\alpha$ của Lasso (tương ứng với $\lambda$) trong khoảng logarithmic từ $10^{-4}$ đến $10^3$. Việc tìm kiếm trên không gian log-space giúp thử nhiều cấp độ regularization, từ rất nhẹ đến rất mạnh. Kết quả cho thấy $\alpha \approx 78.48$ là giá trị tối ưu cho mô hình (đạt $R^2$ cao nhất trên tập validation). Với $\alpha$ này, mô hình Lasso giữ lại được độ đơn giản cần thiết đồng thời vẫn giải thích tốt phương sai của dữ liệu. Mô hình cuối cùng được huấn luyện với `alpha=78.48`, `random_state=42` (để kết quả tái lập) và `max_iter=10000`. Việc tăng `max_iter` lên 10000 vòng lặp nhằm đảm bảo thuật toán Lasso (coordinate descent) hội tụ, nhất là khi $\alpha$ khá lớn.

### 3.4.3 Kết luận

Lasso Regression với hình phạt $L1$ đã chứng tỏ vai trò hiệu quả trong việc kiểm soát hiện tượng overfitting và thực hiện chọn lọc đặc trưng tự động. Nhờ $\lambda$ được lựa chọn phù hợp, mô hình Lasso có thể giảm độ phức tạp (tránh overfit) bằng cách triệt tiêu các hệ số không cần thiết, đồng thời vẫn giữ được những đặc trưng quan trọng giúp mô hình dự báo chính xác. So với Ridge Regression, điểm khác biệt chính là Lasso tạo ra nghiệm thưa với một số hệ số đúng bằng 0 (tương ứng loại bỏ hoàn toàn những biến không quan trọng), trong khi Ridge (chuẩn hóa $L2$) không bao giờ đưa bất kỳ hệ số nào về 0 – do đó Ridge không có khả năng chọn lọc đặc trưng. Mặt khác, trong một số trường hợp, Ridge có thể cho kết quả tốt hơn Lasso nếu mô hình tối ưu không thực sự thưa hoặc khi các đặc trưng độc lập có tương quan cao. Khi các biến dự báo đều có đóng góp nhất định và tương quan với nhau, Lasso có xu hướng chỉ giữ lại ngẫu nhiên một biến rồi triệt tiêu các biến còn lại, có thể làm mất thông tin. Ngược lại, Ridge vẫn giữ lại tất cả các biến (mỗi biến đều có hệ số nhỏ hơn), nhờ đó tránh loại bỏ nhầm những biến hữu ích. Tóm lại, Lasso và Ridge đều là những kỹ thuật regularization hữu hiệu để giảm overfitting, nhưng Lasso tỏ ra ưu việt ở khả năng rút gọn mô hình và chọn lọc biến tự động, giúp mô hình vừa đơn giản vừa dễ diễn giải hơn.

## 3.5 Mô hình Elastic Net

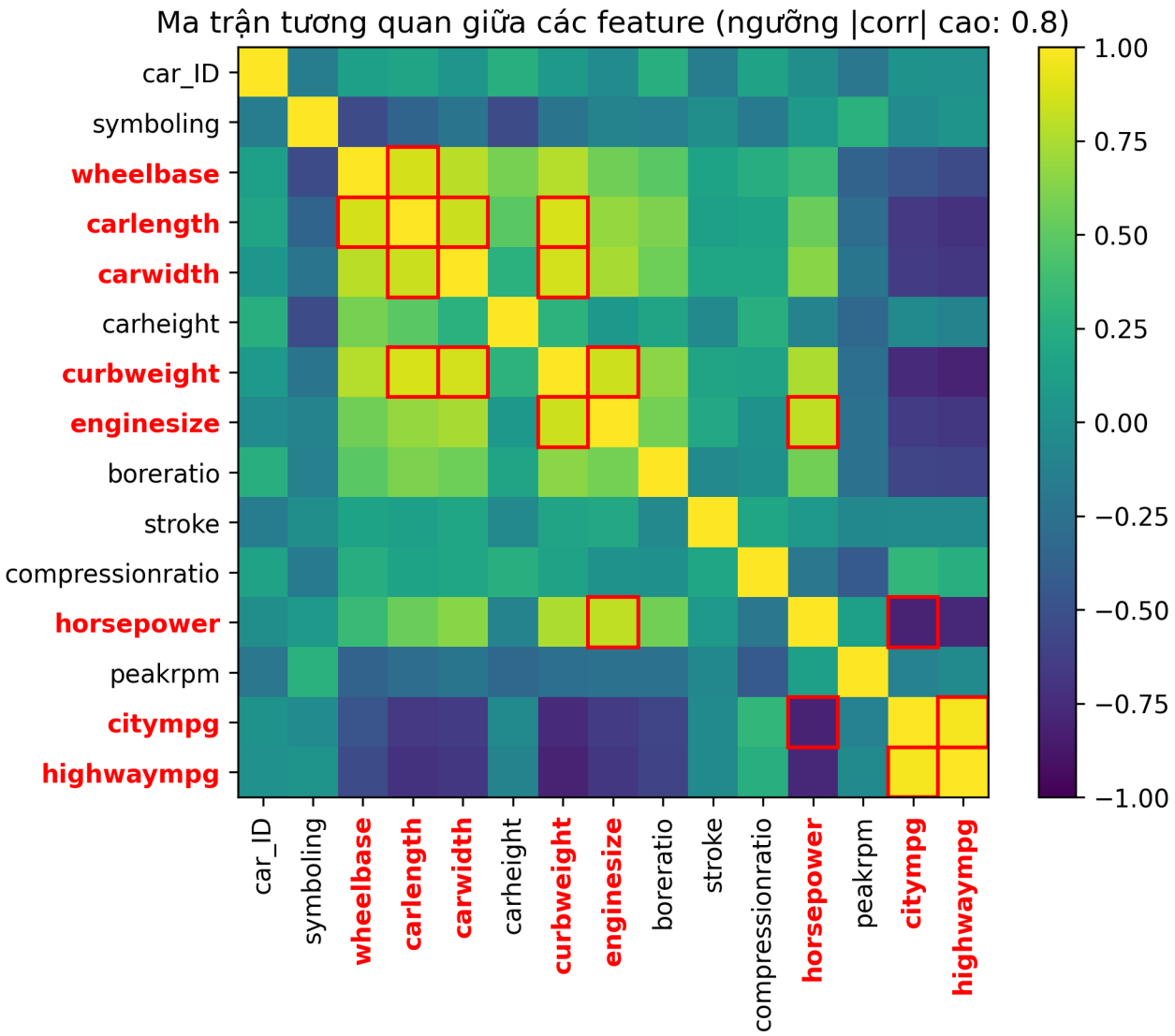
Theo [9], khi kết hợp cả hai dạng regularization $l_1$ và $l_2$, ta thu được mô hình Elastic Net Regression. Lúc đó, **hàm loss của Elastic Net** sẽ có dạng:

$$J(w) = \frac{1}{2}\|y - Xw\|_2^2 + \lambda_1\|w\|_1 + \lambda_2\|w\|_2^2$$

trong đó, $\lambda_1$ và $\lambda_2$ lần lượt là các hệ số điều chuẩn tương ứng với $l_1$ và $l_2$ regularization, giúp cân bằng giữa khả năng chọn lọc đặc trưng và việc giảm độ phức tạp của mô hình. Nhờ đó, Elastic Net đặc biệt hữu ích khi dữ liệu vừa chứa nhiều đặc trưng không quan trọng, vừa tồn tại hiện tượng đa cộng tuyến (multicollinearity) giữa các biến.

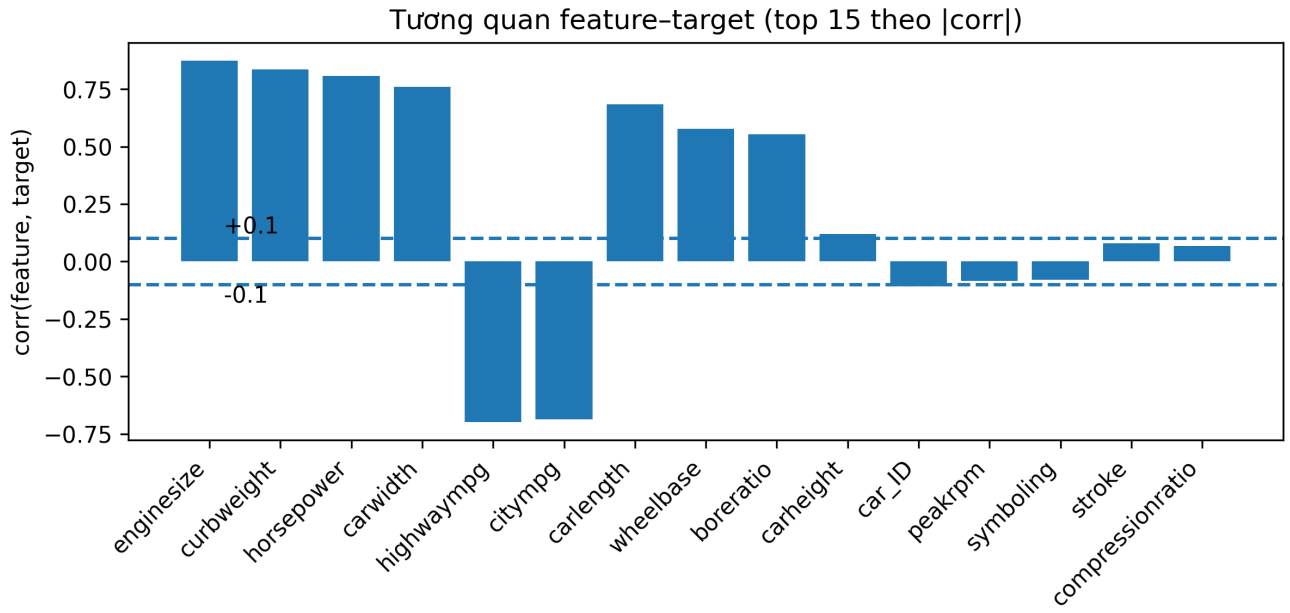
## 3.6 Phân tích dữ liệu và lựa chọn mô hình

Ta tiến hành trích xuất một số đặc trưng của các cột dữ liệu (trừ carname) của dataset:




Hình 1: Heatmap correlation giữa các feature

Có 8 cặp feature có correlation cao được highlight trên hình như là carlength với wheelbase, carwidth, curbweight; carwidth và curbweight, enginesize và curbweight, horsepower và enginesize, citympg và horsepower, và highwaympg với citympg.




Hình 2: Bar chart correlation giữa feature và target (price)

Ngược lại, khi kiểm tra correlation giữa các feature và target, ta thấy có những biến có correlation rất là thấp: carhieght, car_ID, peakrpm, symboling, stroke, và compressionratio.

=> Đây là một dataset vừa có các feature có correlation với nhau rất cao, vừa có các feature khác có correlation với target gần như bằng 0.

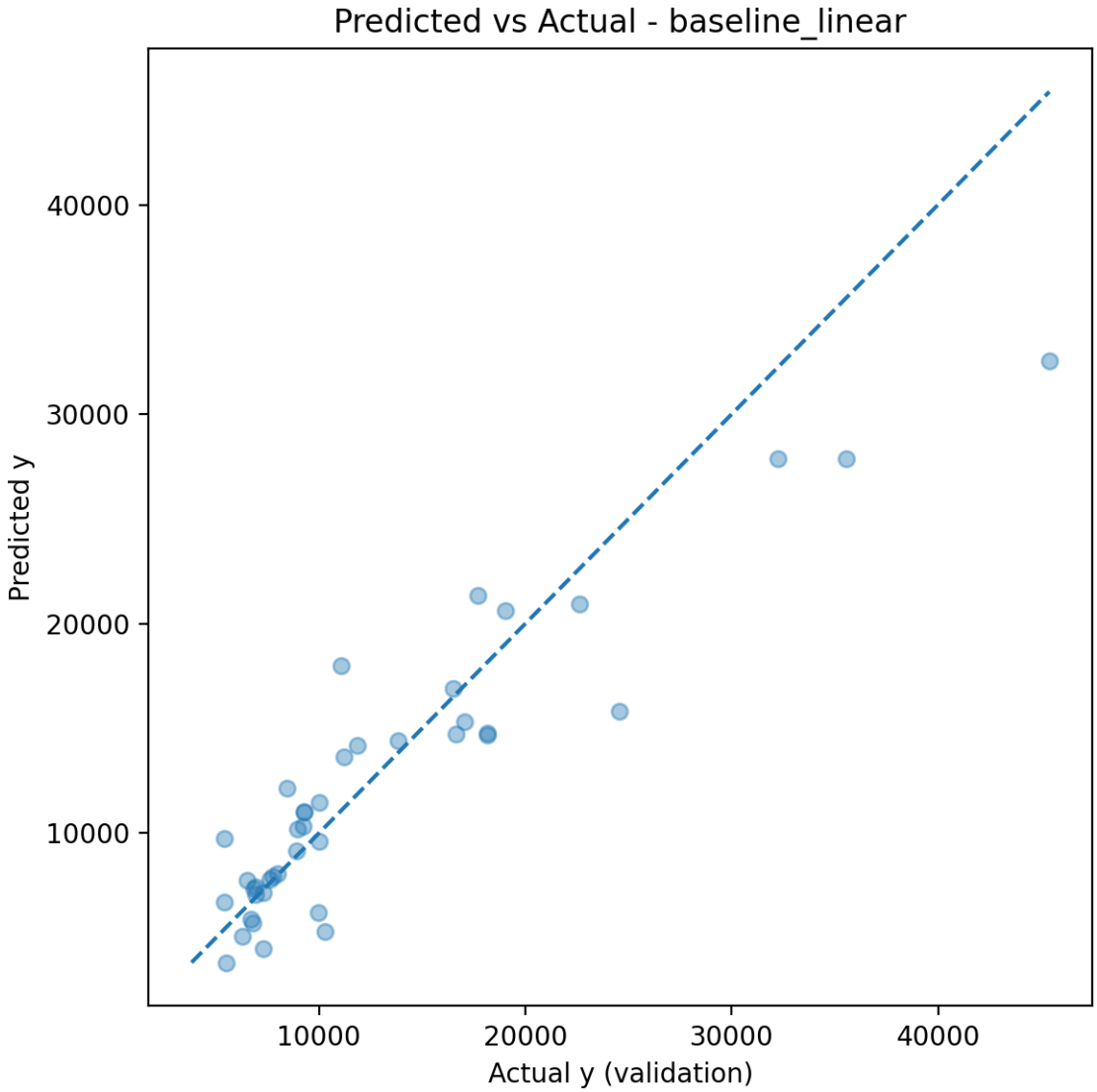
Với vấn đề đầu tiên, ta có thể giải quyết bằng Ridge. Với vấn đề thứ 2, ta có thể giải quyết bằng Lasso. Ngoài ra, thay vì thêm một chuẩn bậc nhất hoặc bậc 2 vào hàm loss nhằm giải quyết các vấn đề liên quan đến dataset, ta cũng có thể thử tối ưu hóa dataset ngay từ bước đầu tiên thông qua việc chọn các feature cho phù hợp.

Vì vậy, nhóm quyết định sẽ thử chạy dataset này thông qua các model: Ridge, Lasso, Elastic Net (Ridge + Lasso), và Linear Regression cơ bản nhưng các feature được chọn trước thông qua thuật toán Forward Selection BIC.

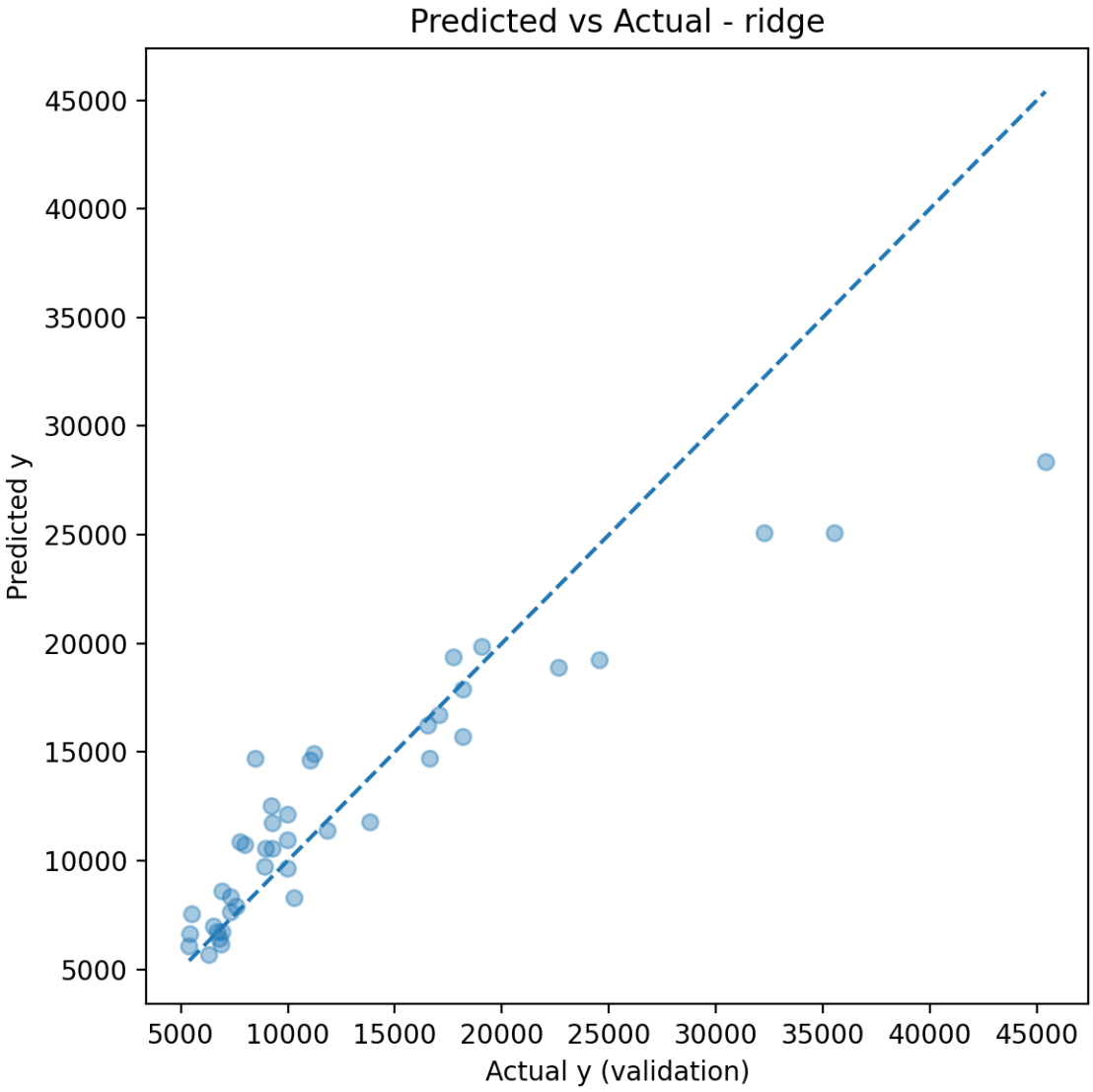
Với các model: Ridge, Lasso, Elastic, hệ số $\lambda_1$ và $\lambda_2$ được chọn đơn giản bằng grid search với target là $R^2$ cao nhất.

# 4 Đánh giá và so sánh kết quả

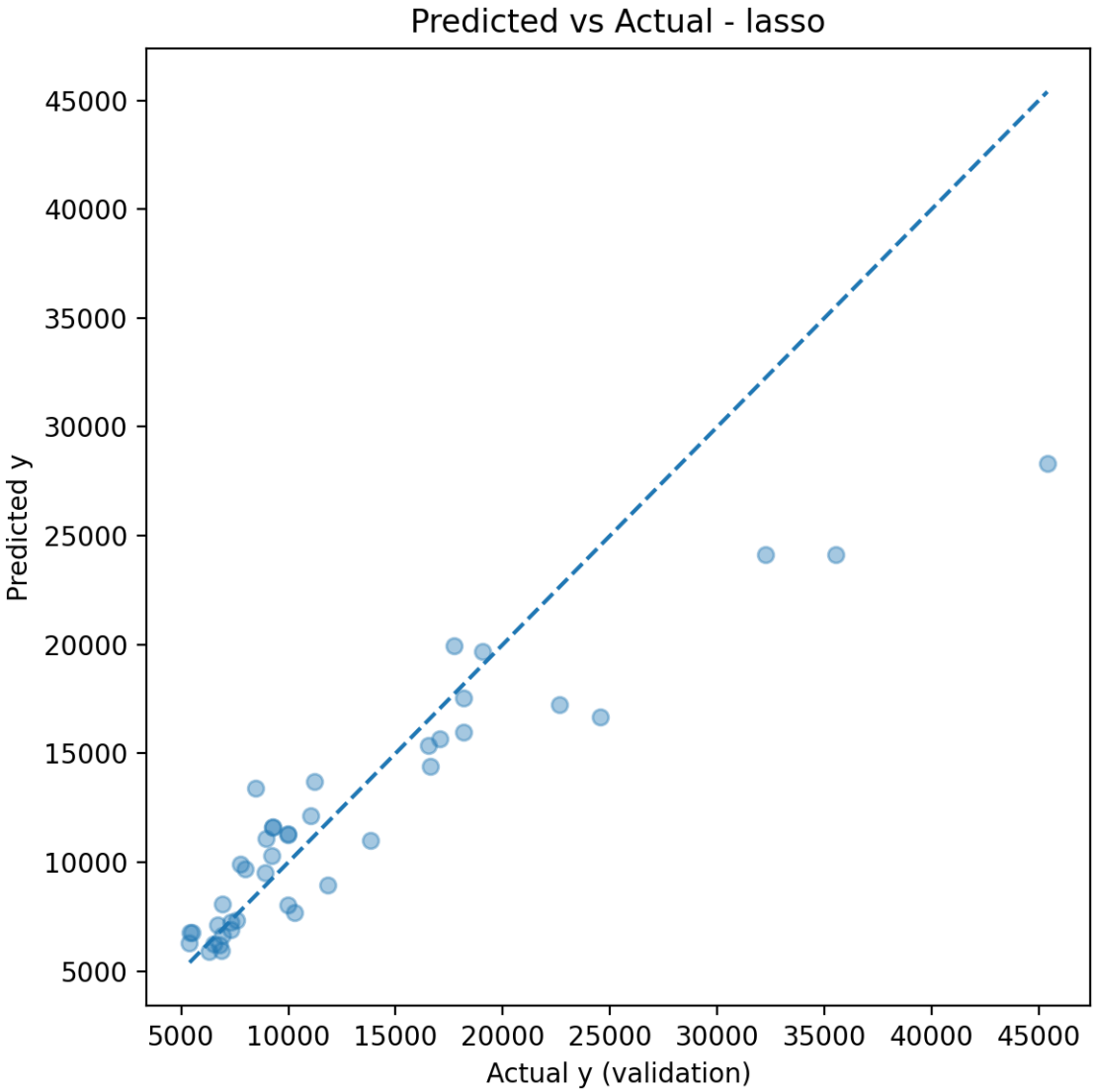
## 4.1 Kết quả dự đoán của 4 mô hình trên tập validation



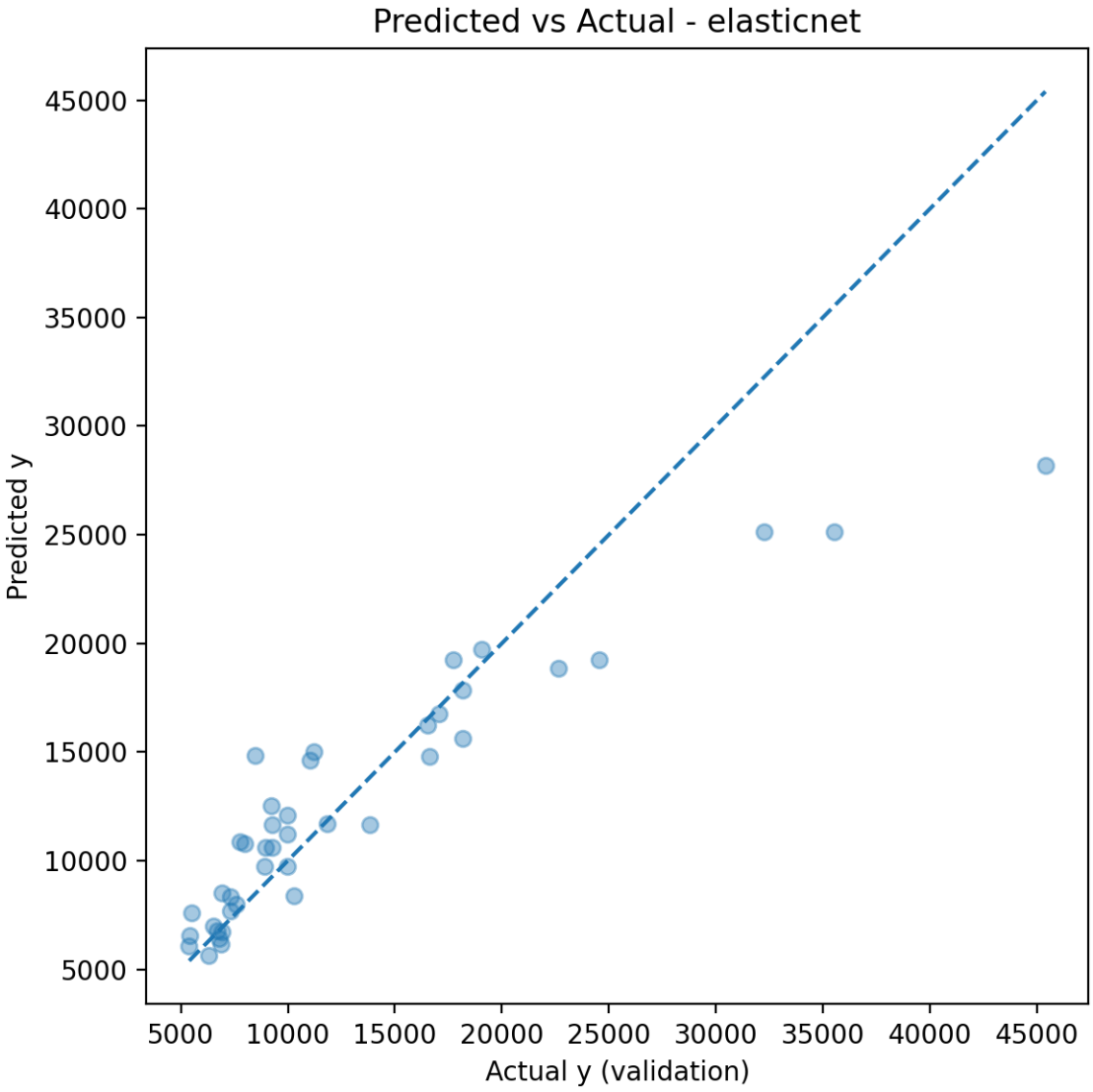

Hình 3: So sánh giá trị dự đoán và giá trị thực tế – Mô hình Linear Regression




Hình 4: So sánh giá trị dự đoán và giá trị thực tế – Mô hình Ridge



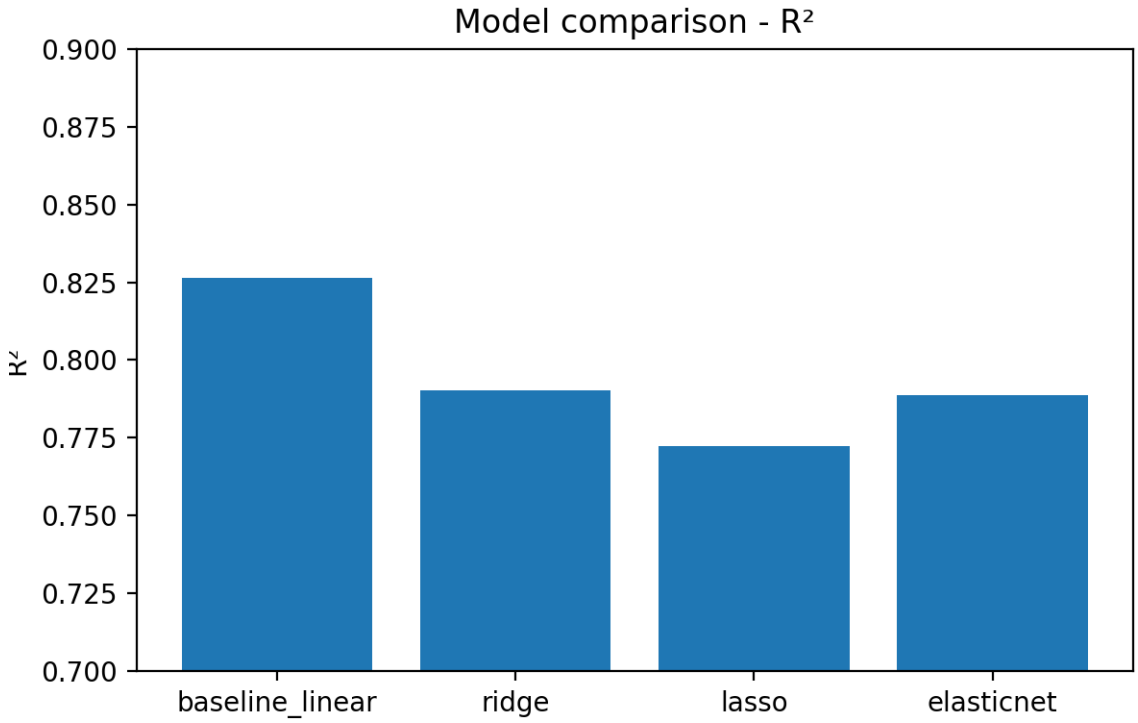

Hình 5: So sánh giá trị dự đoán và giá trị thực tế – Mô hình Lasso




Hình 6: So sánh giá trị dự đoán và giá trị thực tế – Mô hình Elastic Net

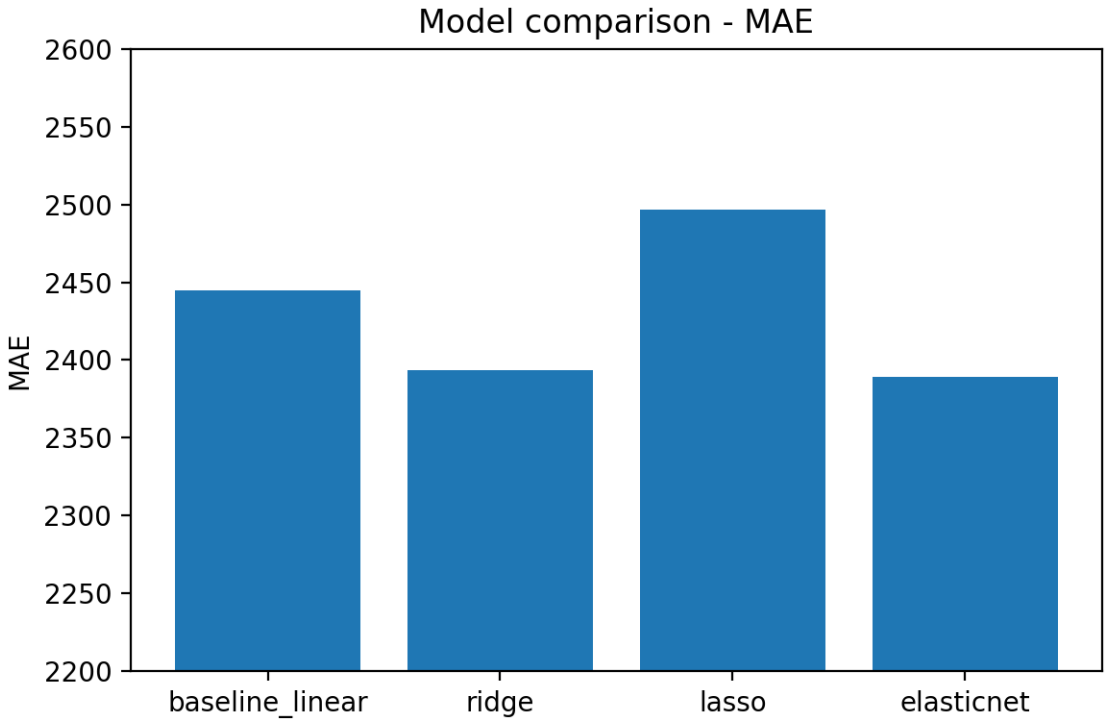
Đầu tiên, ta có thể thấy 4 mô hình đều đưa ra dự đoán giá khá là tốt ở khoảng giá thấp và có xu hướng dự đoán giá thấp hơn giá thực tế với các xe ở giá cao.

## 4.2 So sánh giữa các mô hình với nhau




Hình 7: So sánh giá trị $R^2$ giữa các model

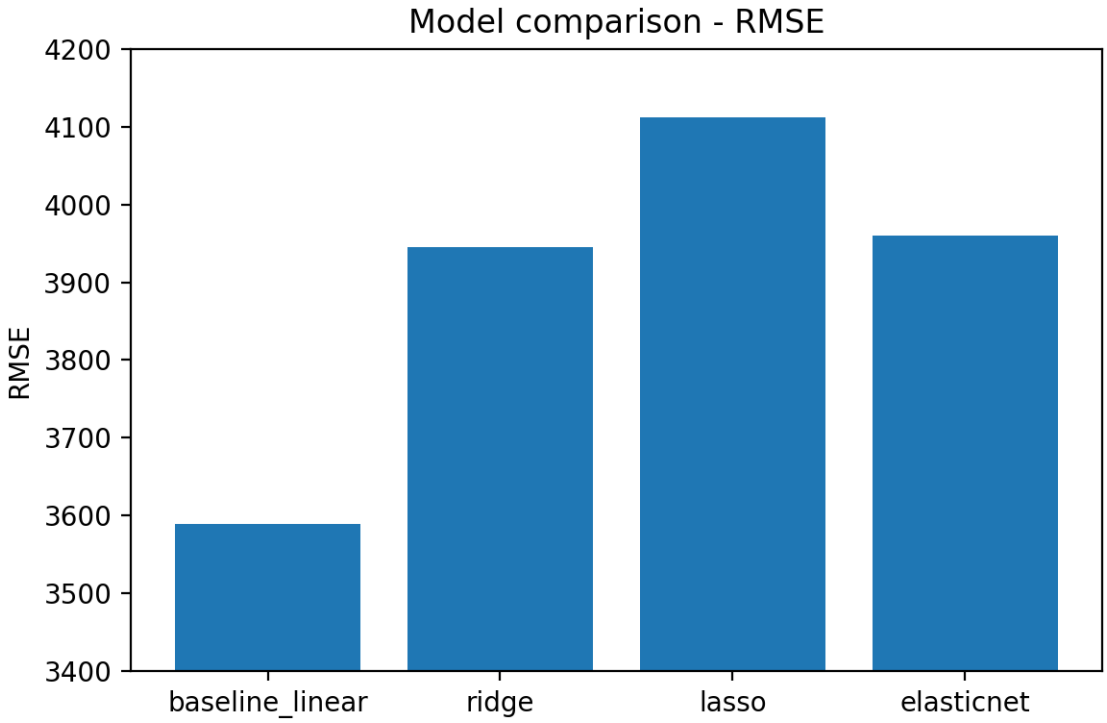
Mô hình Linear Regression sử dụng BIC đạt $R^2 = 0.83$ cao nhất trong 4 model => Giải thích được khoảng 83% độ biến thiên của dữ liệu. Ba mô hình còn lại có giá trị $R^2$ không chênh lệch nhau quá nhiều lần lượt là 0.79 với Ridge và Elastic Net và 0.77 với Lasso.




Hình 8: So sánh giá trị MAE giữa các model

ElasticNet và Ridge có MAE thấp nhất và ngang nhau khoảng 2400 trong khi đó Lasso có MAE cao nhất (2500).

=> Dù $R^2$ thấp hơn Linear Regression sử dụng BIC, ElasticNet và Ridge vẫn có MAE nhỏ hơn, nghĩa là trung bình dự đoán gần thực tế hơn.

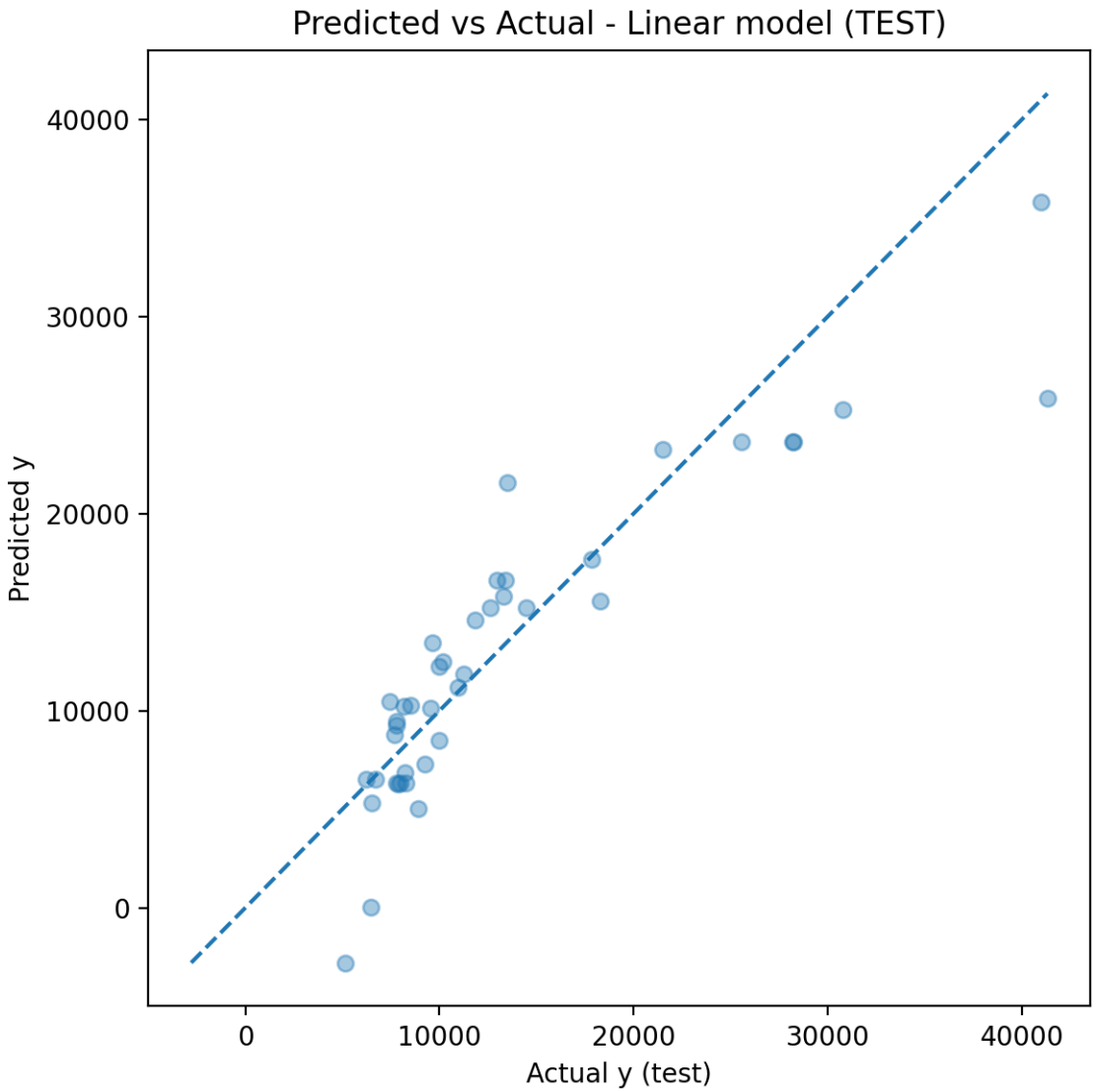



Hình 9: So sánh giá trị RMSE giữa các model

RMSE cao hơn ở các mô hình có Ridge, Lasso, và Elastic Net, cho thấy các ngoại lệ bị dự đoán sai nhiều hơn. Trong khi đó, Linear Regression với BIC có RMSE nhỏ nhất (3600) => mô hình này fit sát cả vùng có giá trị cao. Kết quả cho thấy mô hình Linear Regression cơ bản với BIC đạt $R^2$ cao nhất 0.83, cho thấy mối quan hệ tuyến tính mạnh giữa biến độc lập và biến mục tiêu. Mặc dù các mô hình có thêm ràng buộc về ma trận w vào hàm loss như Ridge và ElasticNet có MAE nhỏ hơn, nghĩa là dự đoán trung bình chính xác hơn và ít bị lệch nghiêm trọng hơn, sự khác nhau là không đáng kể giữa ElasticNet, Ridge, và Linear Regression với BIC. Do đó, nhóm kết luận rằng với dataset này, Linear Regression sử dụng BIC để chọn feature là mô hình đạt kết quả cao nhất với tập validation.

## 4.3 Chạy thực tế trên tập test

Với mô hình Linear Regression sử dụng BIC được chọn ở trên, ta tiến hành chạy lại trên tập test để đánh giá chất lượng:




Hình 10: So sánh giá trị dự đoán và giá trị thực tế của mô hình Linear Regression BIC trên tập test

```
=== Hiệu suất mô hình Linear trên TEST set ===
Test MAE:  2,874.2168
Test RMSE: 3,988.0138
Test R²:   0.7985
```

Hình 11: Các metrics của mô hình Linear Regression BIC trên tập test

Khi chạy lại trên tập test chất lượng của mô hình sụt giảm một chút, tuy nhiên sự suy giảm về chất lượng này là chấp nhận được và không thấy dấu hiệu của việc overfitting khi train.

# 5 Kết luận và nhận định rút ra từ mô hình

# 6 Mô tả ứng dụng

# 7 Tài liệu tham khảo

[1] Sanford Weisberg. "Applied Linear Regression". In: 4th. 2014. Chap. Chapter 10: Variable Selection, pp. 238–239. ISBN: 978-1-118-38608-8.

[2] Sanford Weisberg. "Applied Linear Regression". In: 4th. 2014. Chap. Chapter 10: Variable Selection, p. 240. ISBN: 978-1-118-38608-8.

[3] Talhahascelik. *Automated Stepwise Backward and Forward Selection*. en. URL: https://github.com/talhahascelik/python_stepwiseSelection/blob/master/stepwiseSelection.py.

[4] Tiep Vu. *Bài 15: Overfitting*. Mar. 2017. URL: https://machinelearningcoban.com/2017/03/04/overfitting.

[5] *2.2.2. Hồi qui Ridge — Deep AI KhanhBlog*. en. URL: https://phamdinhkhanh.github.io/deepai-book/ch%5C_ml/RidgedRegression.html.

[6] Robert Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288. DOI: 10.1111/j.2517-6161.1996.tb02080.x.

[7] *What is lasso regression?* IBM. URL: https://www.ibm.com/think/topics/lasso-regression (visited on 11/11/2025).

[8] Arjun Mota. *Lasso Regression*. URL: https://arjun-mota.github.io/posts/lasso-regression/ (visited on 11/11/2025).

[9] Machine Learning Cơ Bản. *Bài 15: Overfitting*. vi. https://machinelearningcoban.com/2017/03/04/overfitting/. Truy cập ngày 11 tháng 7 năm 2025. Mar. 2017.