TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN


# ĐỒ ÁN 3: LINEAR REGRESSION

TOÁN ỨNG DỤNG VÀ THỐNG KÊ

**Triệu Nhật Minh — 21127112 — 21CLC02**

*Giảng viên hướng dẫn*

Vũ Quốc Hoàng
Lê Thanh Tùng
Nguyễn Văn Quang Huy
Phan Thị Phương Uyên

Ngày 23 tháng 8 năm 2023

# Mục lục

# 1 Thư viện sử dụng

## 1.1 pandas

Thư viện cho phép thao tác với dữ liệu dạng bảng. pandas cung cấp các đối tượng DataFrame và Series, cho phép lưu trữ, truy xuất, lọc, nhóm, biến đổi và thống kê dữ liệu một cách hiệu quả và dễ dàng. Trong đồ án này, pandas được sử dụng để đọc dữ liệu từ file csv và lưu trữ dữ liệu dưới dạng DataFrame.

## 1.2 numpy

Thư viện cho phép thao tác với mảng nhiều chiều. Với bài toán data fitting sử dụng phương pháp bình phương tối thiểu (OLS Linear Regression), để giải nghiệm $x$ cho hệ phương trình được tính bằng công thức $x = (A^TA)^{-1}A^Tb$. Nhằm tối ưu thời gian tính toán, ta sử dụng hàm có sẵn từ thư viện này. Hầu hết các hàm có sẵn đã quen thuộc từ những đồ án trước, duy có hàm *numpy.ravel* và *numpy.triu* sẽ được giải thích rõ hơn ở phần liệt kê hàm.

## 1.3 matplotlib

Thư viện matplotlib (cụ thể là module pyplot) cho phép ta tạo ra các biểu đồ dạng 2D. matplotlib.pyplot cũng cho phép điều chỉnh các thuộc tính của đồ thị, như màu sắc, kích thước, chú thích và tiêu đề cho đồ thị.

## 1.4 seaborn

Thư viện cho phép vẽ heatmap. Heatmap là một loại biểu đồ 2D biểu diễn giá trị của một ma trận bằng cách sử dụng các ô có màu sắc khác nhau. seaborn cung cấp các hàm để vẽ heatmap từ các đối tượng DataFrame hoặc numpy array, cũng như điều chỉnh các thuộc tính như bản đồ màu, khoảng giá trị, nhãn và tiêu đề. Heatmap là thành phần không thể thiếu để tìm hiểu mối quan hệ giữa các biến trong bộ dữ liệu và là tiền đề để thực hiện tìm mô hình cho yêu cầu 1d.

## 1.5 sklearn

Thư viện cho phép chia dữ liệu thành các fold để thực hiện cross-validation. Cross-validation là một kỹ thuật kiểm tra hiệu năng của mô hình học máy bằng cách sử dụng một phần của dữ liệu làm tập kiểm tra và phần còn lại làm tập huấn luyện. *sklearn.model_selection.KFold* cho phép chia dữ liệu thành k fold có kích thước bằng nhau và lặp qua từng fold để sử dụng làm tập kiểm tra hoặc tập huấn luyện.

*sklearn.model_selection.KFold* là một lớp trong thư viện scikit-learn, cung cấp các chỉ số để chia dữ liệu thành các tập huấn luyện và kiểm tra. Nó chia tập dữ liệu thành k fold liên tiếp. Mỗi fold được sử dụng một lần làm tập kiểm tra trong khi các fold còn lại được sử dụng làm tập huấn luyện.

Hàm tự cài đặt có thể thực hiện chức năng tương tự như KFold, nhưng có thể khác biệt về hiệu suất và tính năng. Việc sử dụng KFold từ scikit-learn có thể đảm bảo tính ổn định và độ tin cậy của kết quả, do nó được sử dụng rộng rãi trong cộng đồng khoa học dữ liệu, nhất là khi bộ dữ liệu được sử dụng trong đồ án khó có thể kiểm tra thủ công. Tuy nhiên, một hàm tự cài đặt có thể được tùy chỉnh để phù hợp với nhu cầu đặc biệt của người dùng, thậm chí có thể có hiệu suất tốt hơn so với KFold trong một số ít trường hợp.

### 1.6 IPython

Thư viện này không đóng góp vào việc giải quyết bài toán, nhưng vẫn được sử dụng vì khả năng hiển thị ngôn ngữ LaTeX để trình bày công thức hồi quy tuyến tính cho yêu cầu 1a do số lượng biến lớn. Module IPython.display.Latex cho phép chèn các biểu thức LaTeX vào Jupyter Notebook.

## 2 Hàm sử dụng

### 2.1 Hàm built-in từ thư viện

#### 2.1.1 Hàm pandas.read_csv

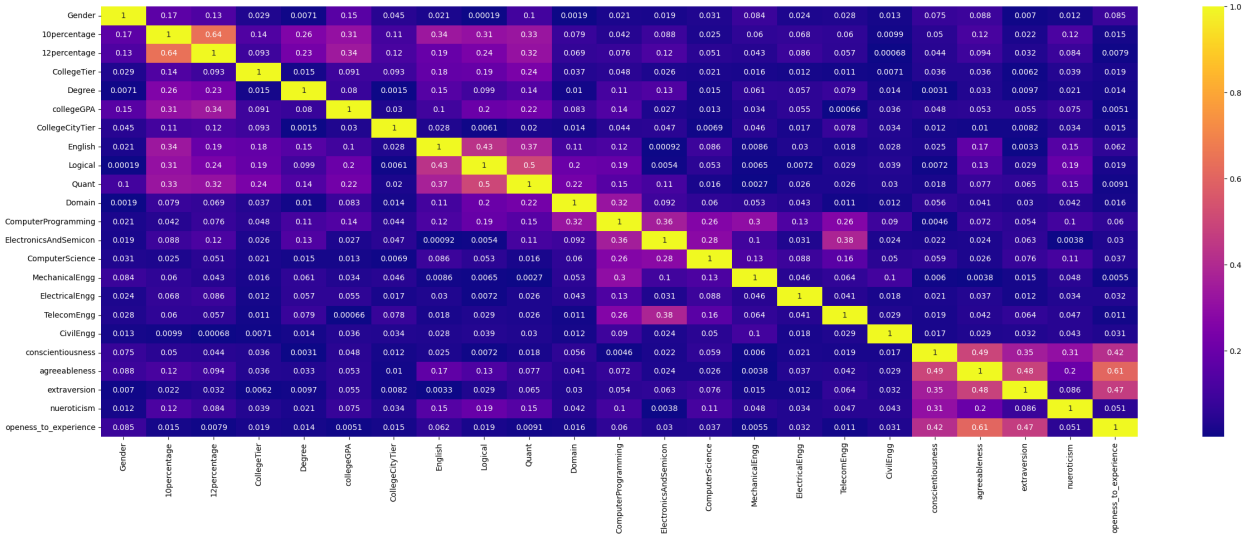
**Input:** Đường dẫn đến file csv.
**Output:** DataFrame chứa dữ liệu từ file csv.

**Mô tả:** Hàm *pandas.read_csv* [12] được sử dụng để đọc dữ liệu từ file csv và lưu trữ dữ liệu dưới dạng DataFrame. Hàm này có thể nhận thêm các tham số để tùy chỉnh cách đọc dữ liệu, nhưng trong đồ án này ta giữ nguyên các tham số khác xem như mặc định, chỉ tùy chỉnh đường dẫn đến file csv.

#### 2.1.2 Hàm pandas.drop

**Input:** Tên cột cần xóa.
**Output:** DataFrame sau khi đã xóa cột.

**Mô tả:** Trong yêu cầu 1d, để xây dựng mô hình chứa các đặc trưng chứa ít sự tương quan nhát (least correlation features), sau khi đã tìm được ma trận tương quan giữa các cột, ta sẽ xóa các cột có độ tương quan cao hơn 0.6. Mặc dù trong cộng đồng khoa học dữ liệu, các đặc trưng có độ tương quan cao hơn 0.95 mới được xem là có sự tương quan cao, nhưng trong đồ án này, dựa vào ma trận tương quan thì giá trị lớn nhất là 0.64 nên ta sẽ xóa các cột có độ tương quan cao hơn 0.6. Và hàm *pandas.drop* [11] được gọi để xóa cột dựa trên tên cột thoả yêu cầu đã đề cập.




Hình 1: Ma trận tương quan giữa các cột trong yêu cầu 1d

### 2.1.3 Hàm pandas.corr

**Input:** Các tham số để tùy chỉnh cách tính ma trận tương quan.
**Output:** DataFrame chứa ma trận tương quan giữa các cột.

**Mô tả:** Ma trận tương quan (correlation matrix) là một ma trận vuông chứa các hệ số tương quan giữa nhiều biến. Mỗi ô trong bảng cho biết mối tương quan giữa hai biến cụ thể. Ma trận tương quan thường được sử dụng để tóm tắt dữ liệu, làm đầu vào cho phân tích nâng cao hơn và làm chẩn đoán cho phân tích nâng cao

Một số điểm cần lưu ý khi đọc ma trận tương quan:

- Các hệ số tương quan trên đường chéo của bảng đều bằng 1 vì mỗi biến hoàn toàn tương quan với chính nó.
- Chỉ một nửa của ma trận tương quan cần được hiển thị vì nửa còn lại của các hệ số tương quan trong ma trận là dư thừa và không cần thiết.
- Ta có thể tô màu ma trận tương quan sẽ được như một bản đồ nhiệt (sử dụng tham số *cmap* trong hàm heatmap) để làm cho các hệ số tương quan dễ đọc hơn.

Trong đồ án này, phương pháp Pearson [13] được sử dụng để tính ma trận tương quan. Phương pháp này được sử dụng để đo lường mối quan hệ giữa hai biến ngẫu nhiên X và Y. Giá trị của hệ số tương quan Pearson nằm trong khoảng [-1, 1]. Hệ số tương quan bằng 1 nếu có mối quan hệ tuyến tính thuận hoàn hảo giữa hai biến, bằng -1 nếu có mối quan hệ tuyến tính nghịch hoàn hảo giữa hai biến. Hệ số tương quan bằng 0 nếu không có mối quan hệ tuyến tính giữa hai biến.

Công thức tính hệ số tương quan Pearson giữa hai biến X và Y:

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{1}$$

Trong đó:

- $r_{X,Y}$ là hệ số tương quan Pearson giữa hai biến X và Y.
- $x_i$ là giá trị của biến X tại điểm dữ liệu thứ i.
- $y_i$ là giá trị của biến Y tại điểm dữ liệu thứ i.
- $\bar{x}$ là giá trị trung bình của biến X.
- $\bar{y}$ là giá trị trung bình của biến Y.
- n là số lượng điểm dữ liệu.

Do ta cần quan tâm độ tương quan giữa hai đặc trưng chứ không quan tâm chúng có tương quan thuận hay nghịch, nên ta sẽ lấy giá trị tuyệt đối của hệ số tương quan Pearson để tính ma trận tương quan. Bằng cách gọi hàm *abs()* trên DataFrame chứa ma trận tương quan, ta sẽ được ma trận tương quan giữa các cột như hình

### 2.1.4 Hàm numpy.linalg.inv

**Input:** Ma trận vuông cần tính ma trận nghịch đảo.
**Output:** Ma trận nghịch đảo của ma trận đầu vào.

**Mô tả:** Hàm *numpy.linalg.inv* [4] được sử dụng để tính ma trận nghịch đảo của ma trận vuông. Trong đồ án này, hàm này được sử dụng để tính ma trận nghịch đảo của ma trận $A^TA$ để tìm nghiệm của hệ phương trình tuyến tính $x = (A^TA)^{-1}A^Tb$ (hàm *fit* trong class OLSLinearRegression tự cài đặt, song để tối ưu hoá thời gian tính toán, ta sẽ sử dụng hàm *numpy.linalg.inv* để tính ma trận nghịch đảo thay vì tự cài đặt).

### 2.1.5 Hàm numpy.ravel

**Input:** Mảng NumPy được đọc theo thứ tự được chỉ định bởi tham số order và được đóng gói thành một mảng 1 chiều và được đọc theo thứ tự C.

**Output:** Mảng NumPy một chiều.

**Mô tả:** Trong numpy, tham số order được sử dụng để chỉ định cách mà các phần tử của một mảng được lưu trữ trong bộ nhớ C trong *order='C'*, có nghĩa là mảng được lưu trữ theo thứ tự liên tục của C, hay chỉ số cuối cùng thay đổi nhanh nhất [1]. Điều này có nghĩa là khi bạn duyệt qua các phần tử của một mảng nhiều chiều theo thứ tự C, bạn sẽ duyệt qua các phần tử của chỉ số cuối cùng trước, sau đó tăng chỉ số kế cuối lên 1 và tiếp tục duyệt qua các phần tử của chỉ số cuối cùng, và cứ tiếp tục như vậy cho đến khi duyệt hết các phần tử của mảng.

Trong bài toán hồi quy tuyến tính, chúng ta cần tìm một ma trận trọng số self.w sao cho tổng bình phương sai số giữa giá trị dự đoán và giá trị thực tế là nhỏ nhất. Để làm được điều này, chúng ta cần biến đổi ma trận trọng số thành một vector có kích thước bằng với số lượng tham số trong mô hình. Hàm *numpy.ravel()* [7] giúp chúng ta thực hiện việc này một cách dễ dàng và nhanh chóng. Bằng cách sử dụng hàm này, chúng ta có thể tính toán tích vô hướng giữa vector trọng số và ma trận đầu vào X bằng cách nhân từng phần tử tương ứng và cộng lại bằng cách gọi hàm *numpy.sum*. Cuối cùng ta thu được giá trị dự đoán cho từng quan sát trong ma trận đầu vào X.

Hàm *numpy.ravel* cũng được sử dụng trong hàm 2.2.4 để làm phẳng các mảng đầu vào y và y_hat thành các mảng 1 chiều. Điều này cho phép tính toán trực tiếp sự khác biệt tuyệt đối giữa các phần tử tương ứng của hai mảng bằng cách trừ chúng và lấy giá trị tuyệt đối. Sau đó, giá trị trung bình của các sự khác biệt tuyệt đối được tính toán bằng hàm *numpy.mean* để trả về giá trị lỗi trung bình tuyệt đối (MAE) giữa hai mảng.

### 2.1.6 Hàm numpy.sum

**Input:** Mảng NumPy cần tính tổng, cột cần tính tổng
**Output:** Hàm này trả về kết quả là tổng của các phần tử trong mảng đầu vào theo các tham số đã xác định. Hàm có thể xử lý các mảng có kích thước và chiều khác nhau, và có thể thực hiện broadcasting nếu các kích thước tương thích.

**Mô tả:** Hàm *numpy.sum* [8] được sử dụng để tính tổng các phần tử trong mảng. Trong đồ án này, hàm này được sử dụng để tính tổng các phần tử trong mảng đầu vào X bằng cách nhân từng phần tử tương ứng của vector trọng số và ma trận đầu vào X và cộng lại. Cuối cùng ta thu được giá trị dự đoán cho từng quan sát trong ma trận đầu vào X.

### 2.1.7 Hàm numpy.mean

**Input:** Mảng NumPy cần tính trung bình, cột cần tính trung bình
**Output:** Hàm này trả về kết quả là trung bình của các phần tử trong mảng đầu vào theo các

tham số đã xác định. Hàm có thể xử lý các mảng có kích thước và chiều khác nhau, và có thể thực hiện broadcasting nếu các kích thước tương thích.

**Mô tả:** Hàm *numpy.mean* [5] được sử dụng để tính giá trị trung bình của các phần tử trong mảng. Trong đồ án này, hàm này được sử dụng để tính giá trị trung bình của các sự khác biệt tuyệt đối giữa các phần tử tương ứng của hai mảng bằng cách trừ chúng và lấy giá trị tuyệt đối. Sau đó, giá trị trung bình của các sự khác biệt tuyệt đối được tính toán bằng hàm *numpy.mean* để trả về giá trị lỗi trung bình tuyệt đối (MAE) giữa hai mảng.

#### 2.1.8 Hàm numpy.triu

**Input:** Mảng NumPy cần tìm ma trận tam giác trên, kích thước của ma trận tam giác trên
**Output:** Mảng NumPy chứa ma trận tam giác trên của mảng đầu vào.

**Mô tả:** Hàm này trả về một bản sao của mảng đầu vào với các phần tử dưới đường chéo thứ k bị đưa về 0. Đối với các mảng có số chiều lớn hơn 2, triu sẽ áp dụng cho hai trục cuối cùng [9].

#### 2.1.9 Hàm numpy.where

**Input:** Mảng NumPy cần tìm vị trí, giá trị cần tìm vị trí
**Output:** Mảng NumPy chứa các vị trí của giá trị cần tìm.

**Mô tả:** Hàm *numpy.where* [10] được sử dụng để lọc các phần tử của một mảng dựa trên một điều kiện cho trước. Cụ thể hơn, *np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)* sẽ tạo ra một mảng có cùng kích thước với corr_matrix, trong đó các phần tử nằm trên đường chéo chính (k=1) sẽ có giá trị là True, còn các phần tử còn lại có giá trị là False.

Sau đó, mảng này được sử dụng như một mặt nạ để lọc các phần tử của corr_matrix bằng cách sử dụng phương thức where. Kết quả cuối cùng là một mảng mới chỉ chứa các phần tử nằm trên đường chéo chính của corr_matrix, còn các phần tử còn lại sẽ bị loại bỏ (có giá trị là NaN).

Nói cách khác, đoạn code trên lọc ra ma trận tam giác trên của ma trận tương quan corr_matrix, loại bỏ các phần tử nằm dưới đường chéo chính và giữ lại các phần tử nằm trên đường chéo chính.

#### 2.1.10 Hàm numpy.ones

**Input:** Kích thước của mảng, kiểu dữ liệu của mảng
**Output:** Mảng NumPy chứa các phần tử có giá trị là 1.

**Mô tả:** Hàm *numpy.ones* [6] được sử dụng để tạo ra một mảng có kích thước và kiểu dữ liệu nhất định, trong đó các phần tử có giá trị là 1.

#### 2.1.11 Hàm seaborn.heatmap

**Input:** Ma trận tương quan, các tham số để tùy chỉnh cách vẽ heatmap.
**Output:** Biểu đồ heatmap.

**Mô tả:** Hàm *seaborn.heatmap* [16] được sử dụng để vẽ heatmap từ các đối tượng DataFrame hoặc numpy array, cũng như điều chỉnh các thuộc tính như bản đồ màu, khoảng giá trị, nhãn và tiêu đề. Heatmap là thành phần không thể thiếu để tìm hiểu mối quan hệ giữa các biến trong bộ dữ liệu và là tiền đề để thực hiện tìm mô hình cho yêu cầu 1d.

Tham số *annot* trong hàm *sns.heatmap* được sử dụng để kiểm soát việc hiển thị giá trị của các ô trong biểu đồ heatmap. Nếu *annot=True*, thì giá trị của mỗi ô sẽ được hiển thị bên trong ô đó.

Biểu đồ heatmap với ma trận tương quan corr_matrix, sử dụng bản đồ màu plasma, và hiển thị giá trị của mỗi ô bên trong ô đó. Nếu tham số này là False, thì giá trị của mỗi ô sẽ không được hiển thị bên trong ô đó, chỉ hiển thị các màu tương ứng với giá trị của ô đó.

#### 2.1.12 Hàm sklearn.model_selection.KFold

**Input:** Số lượng fold, các tham số để tùy chỉnh cách chia dữ liệu.
**Output:** Các chỉ số để chia dữ liệu thành các fold.

**Mô tả:** Chiến lược phổ biến nhất trong học máy là chia tập dữ liệu thành tập huấn luyện và tập kiểm tra. Tỷ lệ chia có thể là 70:30 hoặc 80:20 [3]. Một trong số những phương pháp là sử dụng k-fold Cross Validation. [18]

Trong đồ án này, hàm *sklearn.model_selection.KFold* [17] được sử dụng để chia dữ liệu thành các fold. Hàm này có thể nhận thêm các tham số để tùy chỉnh cách chia dữ liệu, nhưng trong đồ án này ta giữ nguyên các tham số khác xem như mặc định, chỉ tùy chỉnh số lượng fold, kiểu chia dữ liệu và trộn dữ liệu trước khi chia (lần lượt các tham số *n_splits*, *shuffle* và *random_state*). Tham số *n_splits* được sử dụng để chỉ định số lượng fold (k = 20 với các yêu cầu 1b, 1c, 1d), tham số *shuffle* được sử dụng để chỉ định cách chia dữ liệu, và tham số *random_state* được sử dụng để chỉ định cách trộn dữ liệu trước khi chia.

Tham số *shuffle* được sử dụng để chỉ định cách chia dữ liệu. Nếu *shuffle=True*, thì dữ liệu sẽ được trộn ngẫu nhiên trước khi chia. Nếu *shuffle=False*, thì dữ liệu sẽ không được trộn ngẫu nhiên trước khi chia. Trong đồ án này, ta sử dụng *shuffle=True* để trộn dữ liệu ngẫu nhiên trước khi chia.

Đến với tham số *random_state*, nếu *random_state=None*, thì mỗi lần chạy hàm *KFold* sẽ cho ra kết quả khác nhau. Trên thực tế, việc truyền bất kì số nguyên nào (kể cả số 0) thì cũng không

thành vấn đề. Đây là một random seed trong thuật toán random của máy tính. [14] Các thí nghiệm có cùng random seed và cùng các tham số khác sẽ cho ra kết quả giống nhau. 42 là một con số đến từ cuốn sách Hướng dẫn du lịch vũ trụ. Câu trả lời cho cuộc sống vũ trụ và mọi thứ và được coi là một trò đùa. Nó không có ý nghĩa khác ngoài việc là một con số ngẫu nhiên. [15] Trong đồ án này, ta sử dụng *random_state=42* để đảm bảo kết quả của các thí nghiệm là nhất quán.

Việc chia dữ liệu thành k phần giúp chúng ta có thể kiểm tra độ chính xác và ổn định của mô hình trên nhiều tập dữ liệu khác nhau, tránh hiện tượng quá khớp (overfitting) hoặc thiếu khớp (underfitting) mô hình. Việc lựa chọn giá trị k trong k-fold cross validation là tùy thuộc vào nhiều yếu tố. Một giá trị k lớn sẽ cho phép mô hình được huấn luyện trên nhiều dữ liệu hơn, nhưng cũng sẽ tăng thời gian tính toán. Một giá trị k nhỏ sẽ giảm thời gian tính toán, nhưng cũng có thể làm tăng sai số trong việc đánh giá hiệu suất của mô hình. [2]

#### 2.1.13 Hàm IPython.display.Latex

**Input:** Chuỗi kí tự cần hiển thị ở dạng LaTeX.
**Output:** Hiển thị chuỗi kí tự ở dạng LaTeX.

### 2.2 Hàm tự cài đặt

#### 2.2.1 Hàm OLSLinearRegression.fit

**Input:** Ma trận chứa các giá trị của đặc trưng, y là một vector chứa các giá trị của biến mục tiêu.
**Output:** Chính nó (self) để có thể gọi phương thức khác trên cùng đối tượng OLSLinearRegression.

**Mô tả:** Hàm này sử dụng phương pháp bình phương nhỏ nhất để tính toán trọng số $w$ cho mô hình tuyến tính. Đầu tiên, nó tính toán ma trận giả nghịch đảo của ma trận $A^TA$ bằng hàm *numpy.linalg.inv* (thay vì tự cài đặt để tối ưu hoá thời gian tính toán).

Mục đích hàm fit của lớp OLSLinearRegression là tìm ra ma trận trọng số $w$ ứng với mô hình tuyến tính. Để làm được điều này, ta cần giải hệ phương trình tuyến tính $A^TAw = A^Tb$ để tìm ra giá trị của $w$. Để giải hệ phương trình này, ta nhân cả hai vế của phương trình với ma trận nghịch đảo của ma trận $A^TA$, ta được $w = (A^TA)^{-1}A^Tb$. Do đó, ta sẽ tính ma trận nghịch đảo của ma trận $A^TA$ bằng hàm *numpy.linalg.inv* và nhân với ma trận $A^Tb$ để tìm ra giá trị của $w$.

#### 2.2.2 Hàm OLSLinearRegression.predict

**Input:** Ma trận chứa các giá trị của đặc trưng.
**Output:** Mảng NumPy chứa các giá trị dự đoán.

**Mô tả:** Trong hàm *predict*, ta tính toán giá trị dự đoán bằng cách nhân ma trận X với vector trọng số w (được tính toán trong hàm và cộng các giá trị lại với nhau theo chiều thứ nhất (axis=1) để thu được một vector kết quả. Cụ thể, ta sử dụng phép toán *np.sum(self.w.ravel() * X, axis=1)* để thực hiện việc này.

Sở dĩ ta phải gọi hàm *ravel* được sử dụng để chuyển đổi vector trọng số $w$ thành một mảng 1 chiều liên tục trước khi thực hiện phép nhân với ma trận đầu vào $X$. Điều này cần thiết để đảm bảo rằng kích thước của $w$ và $X$ phù hợp với nhau và phép nhân. Trái lại, nếu không sử dụng hàm *ravel*, vector trọng số $w$ có kích thước không phù hợp và dẫn đến lỗi khi thực hiện phép nhân.

### 2.2.3 Hàm OLSLinearRegression.get_params

**Input:** Không có.
**Output:** Mảng NumPy chứa các giá trị của vector trọng số $w$.

**Mô tả:** Hàm *get_params* được sử dụng để trả về các giá trị của vector trọng số $w$. Mô hình OLSLinearRegression sau khi được huấn luyện sẽ có một vector trọng số $w$ duy nhất, và hàm *get_params* được sử dụng để trả về các giá trị của vector trọng số này.

### 2.2.4 Hàm mae

**Input:** Vector chứa các giá trị thực tế từ tập dữ liệu cho trước, vector chứa các giá trị dự đoán tính được từ mô hình.
**Output:** Giá trị lỗi trung bình tuyệt đối (MAE).

**Mô tả:** Phương pháp bình phương tối thiểu khi sử dụng để tìm mô hình tuyến tính sẽ tìm ra mô hình có giá trị trung bình của tổng bình phương sai số là nhỏ nhất. Tuy nhiên, trong thực tế, chúng ta thường quan tâm đến giá trị trung bình của tổng sai số tuyệt đối, hay còn gọi là lỗi trung bình tuyệt đối (MAE). MAE được tính bằng cách lấy giá trị tuyệt đối của sự khác biệt giữa các giá trị thực tế và giá trị dự đoán, sau đó lấy giá trị trung bình của các sự khác biệt tuyệt đối này.

Trong hàm mae, ta sử dụng hàm *numpy.ravel* để làm phẳng các mảng đầu vào y và y_hat thành các mảng 1 chiều. Điều này cho phép tính toán trực tiếp sự khác biệt tuyệt đối giữa các phần tử tương ứng của hai mảng bằng cách trừ chúng và lấy giá trị tuyệt đối. Sau đó, giá trị trung bình của các sự khác biệt tuyệt đối được tính toán bằng hàm *numpy.mean* để trả về giá trị lỗi trung bình tuyệt đối (MAE) giữa hai mảng.

### 2.2.5 Hàm latex_text

**Input:** Mảng NumPy chứa các tham số của mô hình tuyến tính, từ điển chứa các tên cột tương ứng với các tham số trong mảng NumPy.
**Output:** Hiển thị chuỗi kí tự ở dạng LaTeX.

**Mô tả:** Hàm latex_text có hai đầu vào là params và dict. Đầu vào params là một mảng numpy chứa các tham số của mô hình tuyến tính, trong khi đầu vào dict là một từ điển chứa các tên cột tương ứng với các tham số trong params. Hàm này trả về một chuỗi LaTeX biểu diễn phương trình tuyến tính của mô hình dựa trên các tham số và tên cột được cung cấp.

Trong hàm này, ta bắt đầu bằng cách khởi tạo chuỗi text với giá trị ban đầu là phần đầu của phương trình LaTeX: Hàm latex_text có hai đầu vào là params và dict. Đầu vào params là một mảng numpy chứa các tham số của mô hình tuyến tính, trong khi đầu vào dict là một từ điển chứa các tên cột tương ứng với các tham số trong params. Hàm này trả về một chuỗi LaTeX biểu diễn phương trình tuyến tính của mô hình dựa trên các tham số và tên cột được cung cấp.

### 2.2.6 Hàm kfold_cross_model

**Input:** Mảng NumPy chứa các giá trị của đặc trưng, mảng NumPy chứa các giá trị của biến mục tiêu, đối tượng KFold, mảng để lưu trữ các giá trị MAE.
**Output:** Không có.

**Mô tả:** Trong hàm này, ta sử dụng vòng lặp để duyệt qua các phần chia của dữ liệu huấn luyện khi thực hiện phương pháp kfold cross validation. Mỗi phần tử của iterator là một tuple chứa hai mảng chỉ số: *train_index* và *test_index*. Mảng *train_index* chứa các chỉ số của phần huấn luyện, trong khi mảng *test_index* chứa các chỉ số của phần kiểm tra1.

Vòng lặp *for train_index, test_index in kf.split(X_train_np)* được sử dụng để duyệt qua các phần chia của dữ liệu huấn luyện. Đối với mỗi phần chia, ta sử dụng chỉ số của phần huấn luyện và phần kiểm tra để trích xuất dữ liệu tương ứng từ X_train_np và y_train_np. Sau đó, ta có thể huấn luyện mô hình với dữ liệu huấn luyện đã trích xuất và đánh giá hiệu suất của mô hình trên dữ liệu kiểm tra. Cuối cùng, ta thêm giá trị MAE vào danh sách mae_arr để lưu trữ kết quả. Nếu X_train_np là một vector 1 chiều, ta thêm một chiều mới vào cuối bằng cách sử dụng cú pháp [:, None]. Ngược lại, ta chỉ cần trích xuất dữ liệu bình thường. Sau đó, ta huấn luyện một mô hình hồi quy tuyến tính bằng cách sử dụng phương thức fit của class OLSLinearRegression với dữ liệu huấn luyện đã trích xuất. Ta sử dụng mô hình đã huấn luyện để dự đoán giá trị đầu ra cho phần kiểm tra của dữ liệu và tính toán giá trị MAE giữa giá trị thực tế và giá trị dự đoán. Cuối cùng, ta thêm giá trị MAE vào danh sách mae_arr để lưu trữ kết quả.

Hàm này không có giá trị trả về, nhưng nó thay đổi nội dung của danh sách mae_arr bằng

cách thêm các giá trị MAE tính toán được trong quá trình kiểm định chéo.

#### 2.2.7 Hàm scientific_notation_converter

**Input:** Mảng chứa các trọng số của mô hình tuyến tính.
**Output:** Mảng chứa các trọng số của mô hình tuyến tính ở dạng ký hiệu khoa học với 3 chữ số thập phân.

**Mô tả:** Trong đồ án yêu cầu ta phải trình bày mô hình dự đoán mức lương với tham số được làm tròn 3 chữ số thập phân. Ta có thể sử dụng hàm *round* trong phương thức *get_params*. Song nếu làm tròn ngay từ bước huấn luyện, ta sẽ mất đi độ chính xác của mô hình. Do đó, ta sẽ sử dụng hàm *scientific_notation_converter* để chuyển đổi các trọng số của mô hình.

Trong hàm này, ta sử dụng kỹ thuật list comprehension để duyệt qua các phần tử trong params và chuyển đổi chúng thành dạng ký hiệu khoa học (scientific notation). Đối với mỗi phần tử x trong params, ta chuyển đổi nó thành số thực bằng cách sử dụng hàm float(x), sau đó định dạng nó thành chuỗi ký hiệu khoa học với 3 chữ số thập phân bằng cách sử dụng phương thức format: *"{:.3f}".format(float(x))*. Kết quả cuối cùng là một danh sách mới chứa các chuỗi ký hiệu khoa học tương ứng với các phần tử trong params.

# 3 Đánh giá kết quả mô hình

Trước khi thực hiện tìm mô hình hồi quy tuyến tính cho tất cả yêu cầu đề bài, ta cần đọc dữ liệu từ *train.csv* và *test.csv* ứng với tập dữ liệu huấn luyện (**train**) và tập dữ liệu kiểm tra (**test**), đồng thời xử lý DataFrame vừa đọc được để nhận biết đâu là tập các đặc trưng (**X**) và đâu là tập biến mục tiêu (**y**). Sau đó, ta sẽ chia tập dữ liệu huấn luyện thành hai phần: tập huấn luyện (**X_train**, **y_train**) và tập kiểm tra (**X_test**, **y_test**). Do tập mục tiêu là cố định (Salary ứng với **y_train** và **y_test**), nên ta dùng chung hai tập này cho các câu 1a, 1b, 1c, 1d.

Sở dĩ phải chuyển về NumPy array với tất cả DataFrame vừa đọc được là vì khi thực hiện yêu cầu tìm đặc trưng tốt nhất trong số các đặc trưng (yêu cầu 1b, 1c), ma trận đặc trưng $X$ chỉ có 1 cột, nên khi thực hiện các phép toán trên ma trận, ta cần mở rộng 1 chiều mới

## 3.1 Yêu cầu 1a

### 3.1.1 Các bước thực hiện

1. Thực hiện lấy 11 đặc trưng đầu tiên đề bài cung cấp.
2. Thực hiện chia tập dữ liệu huấn luyện thành hai phần: tập huấn luyện (**X_1a_train**, **y_train**) và tập kiểm tra (**X_test**, **y_test**).

3.

#### 3.1.2 Công thức hồi quy

#### 3.1.3 Kết quả mô hình

#### 3.1.4 Nhận xét

### 3.2 Yêu cầu 1b

### 3.3 Yêu cầu 1c

### 3.4 Yêu cầu 1d

# 4 Tài liệu tham khảo


[1] *Cheapest way to get a numpy array into C-contiguous order? — stackoverflow.com*. https://stackoverflow.com/questions/29947639/cheapest-way-to-get-a-numpy-array-into-c-contiguous-order. [Accessed 11-08-2023].

[2] *Choice of K in K-fold cross-validation — stats.stackexchange.com*. https://stats.stackexchange.com/questions/27730/choice-of-k-in-k-fold-cross-validation. [Accessed 16-08-2023].

[3] *How to Implement K fold Cross-Validation in Scikit-Learn — section.io*. https://www.section.io/engineering-education/how-to-implement-k-fold-cross-validation/. [Accessed 11-08-2023].

[4] *numpy.linalg.inv; NumPy v1.25 Manual — numpy.org*. https://numpy.org/doc/stable/reference/generated/numpy.linalg.inv.html. [Accessed 11-08-2023].

[5] *numpy.mean; NumPy v1.25 Manual — numpy.org*. https://numpy.org/doc/stable/reference/generated/numpy.mean.html. [Accessed 11-08-2023].

[6] *numpy.ones; NumPy v1.25 Manual — numpy.org*. https://numpy.org/doc/stable/reference/generated/numpy.ones.html. [Accessed 19-08-2023].

[7] *numpy.ravel; NumPy v1.25 Manual — numpy.org*. https://numpy.org/doc/stable/reference/generated/numpy.ravel.html. [Accessed 11-08-2023].

[8] *numpy.sum; NumPy v1.25 Manual — numpy.org*. https://numpy.org/doc/stable/reference/generated/numpy.sum.html. [Accessed 11-08-2023].

[9] *numpy.triu; NumPy v1.25 Manual — numpy.org*. https://numpy.org/doc/stable/reference/generated/numpy.triu.html. [Accessed 19-08-2023].

[10] *numpy.where; NumPy v1.25 Manual — numpy.org*. https://numpy.org/doc/stable/reference/generated/numpy.where.html. [Accessed 19-08-2023].

[11] *pandas.DataFrame.drop; pandas 2.0.3 documentation — pandas.pydata.org*. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html. [Accessed 19-08-2023].

[12] *pandas.read_csv; pandas 2.0.3 documentation — pandas.pydata.org*. https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html. [Accessed 11-08-2023].

[13] *Pearson correlation coefficient - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Pearson_correlation_coefficient. [Accessed 19-08-2023].

[14] *Random seed - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Random_seed. [Accessed 16-08-2023].

[15] *Random state (Pseudo-random number) in Scikit learn — stackoverflow.com*. https://stackoverflow.com/questions/28064634/random-state-pseudo-random-number-in-scikit-learn. [Accessed 16-08-2023].

[16] *seaborn.heatmap; seaborn 0.12.2 documentation — seaborn.pydata.org*. https://seaborn.pydata.org/generated/seaborn.heatmap.html. [Accessed 19-08-2023].

[17] *sklearn.model_selection.KFold — scikit-learn.org*. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html. [Accessed 11-08-2023].

[18] Isheunesu Tembo. *Cross-Validation Using K-Fold With Scikit-Learn — isheunesu48.medium.com*. https://isheunesu48.medium.com/cross-validation-using-k-fold-with-scikit-learn-cfc44bf1ce6. [Accessed 11-08-2023].