Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

DATA PRE-PROCESSING AND ANALYSIS

Bài 5: ***Feature Engineering***

https://csc.edu.vn/data-science-machine-learning/Data-Pre-processing-and-Analysis_230

2024

Data Science and Machine Learning

Nội dung

1. **Giới thiệu**
2. **Tạo thuộc tính (feature)**
3. **Chuẩn hóa dữ liệu (Data Standardization)**
4. **Chuyển dạng dữ liệu (Data Transformation)**

Giới thiệu

Feature Engineering là gì?

- Feature Engineering là quá trình sử dụng kiến thức miền (domain knowledge) về dữ liệu để tạo ra các tính năng giúp thuật toán máy học (Machine Learning algorithms) học được hiệu quả.
- Feature Engineering là nền tảng cho các ứng dụng Machine Learning.
- Feature Engineering là công việc thiết yếu trong Machine Learning.

https://en.wikipedia.org/wiki/Feature_engineering

Giới thiệu

❑Đặc điểm

- Feature engineering tạo các tính năng đầu vào mới/ rút trích từ những tính năng hiện có của bộ dữ liệu.
- Là một trong những nhiệm vụ có giá trị nhất mà nhà khoa học dữ liệu có thể làm để cải thiện hiệu suất mô hình. Bởi vì:
  - Có thể cô lập và làm nổi bật thông tin chính, giúp thuật toán "tập trung" vào những gì quan trọng
  - Vận dụng domain knowledge để có tính năng thích hợp
  - Và đem đến cho người khác kiến thức về domain knowledge

Nội dung

1. Giới thiệu
2. **Tạo thuộc tính (feature)**
3. Chuẩn hóa dữ liệu (Data Standardization)
4. Chuyển dạng dữ liệu (Data Transformation)

Tạo thuộc tính (feature)

- Tìm hiểu các kỹ thuật liên quan đến feature engineering và cách áp dụng vào dữ liệu trong thế giới thực.

- Tìm hiểu các loại dữ liệu cơ bản và tại sao chúng có ảnh hưởng đến cách chúng ta thiết kế các thuộc tính của mình.

Tạo thuộc tính (feature)

- **Xem thông tin các thuộc tính**
  - df.info()
  - df.columns
  - df.dtypes
  - df.select_dtypes()
  - …

Tạo thuộc tính (feature)

❑ Mã hóa thuộc tính phân loại (Encoding categorical feature)

- Sử dụng các biến phân loại trong mô hình Machine Learning, trước tiên cần biểu diễn chúng theo dạng định lượng.
- Một số cách tiếp cận phổ biến là one hot encoder/dummy encoder và label encoder.

Tạo thuộc tính (feature)

Vấn đề với Categorical Data

- Một số thuật toán có thể làm việc trực tiếp với dữ liệu phân loại.
  - Ví dụ, Decision Tree có thể được huấn luyện trực tiếp từ dữ liệu phân loại (ở biến đầu ra – target /output) mà không cần chuyển đổi dữ liệu (điều này phụ thuộc vào việc triển khai cụ thể).

Tạo thuộc tính (feature)

- Nhiều thuật toán Machine Learning không thể hoạt động trực tiếp trên dữ liệu nhãn.
- Điều này có nghĩa là dữ liệu phân loại phải được chuyển đổi thành dạng số.

Tạo thuộc tính (feature)

Integer encoder/ label encoder

- Mỗi giá trị phân loại duy nhất được gán một giá trị nguyên (integer value)
  - Ví dụ: “red” là 1, “green” là 2, “blue” là 3.
- Áp dụng cho:
  - Các giá trị số nguyên có mối quan hệ thứ tự và các thuật toán Machine Learning có thể hiểu và khai thác mối quan hệ này.

Tạo thuộc tính (feature)

- Ví dụ: Label encoder

```
label_encoder_s = preprocessing.LabelEncoder()
house_area['Type_encode'] = label_encoder_s.fit_transform(house_area['Type'])
house_area
```

	Area	Type	Type_encode
0	21	Small	2
1	32	Small	2
2	64	Medium	1
3	56	Medium	1
4	128	Large	0
5	160	Large	0

Tạo thuộc tính (feature)

- Ví dụ: Label encoder (Target)

```
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
iris['species']= label_encoder.fit_transform(iris['species'])
iris['species'].unique()
```

```
array([0, 1, 2], dtype=int64)
```

```
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Tạo thuộc tính (feature)

One hot encoder/ Dummy encoder

- Đối với các biến phân loại không tồn tại mối quan hệ thứ tự, label/ integer encoder là không đủ.
- Có thể gặp phải tình huống, sau khi Label Encoder, mô hình có thể nhầm lẫn vì nghĩ rằng cột có dữ liệu với thứ tự hoặc thứ bậc nào đó.
- Trong trường hợp này, one hot encoder/dummy encoder có thể được áp dụng.

Tạo thuộc tính (feature)

- One hot encoder
  - Biến nhị phân mới được thêm vào cho mỗi giá trị số nguyên duy nhất
  - Với one hot encoder, các thuộc tính có thể giải thích rõ ràng
    - Ví dụ: “color” variable, có 3 loại vậy sẽ có 3 biến nhị phân (binary variable): giá trị “1” được thay thế trong binary variable cho color và giá trị “0” cho các color còn lại

red,	green,	blue
1,	0,	0
0,	1,	0
0,	0,	1

Tạo thuộc tính (feature)

- One hot encoder
  - Kết quả trả về là numpy array => có thể chuyển thành dataframe
  - Xử lý phức tạp hơn dummy encoder
  - Có thể lưu trữ onehot encoder model để tái sử dụng khi cần

Tạo thuộc tính (feature)

```
from sklearn.preprocessing import OneHotEncoder
```

```
one_hot_encoder = OneHotEncoder()
one_hot_encoder.fit(students[['Sex']])
students_sex_onehot = one_hot_encoder.transform(students[['Sex']]).toarray()
```

```
one_hot_encoder.categories_
```

```
[array(['Female', 'Male', 'Other'], dtype=object)]
```

```
dfOneHot = pd.DataFrame(students_sex_onehot,
                        columns = ['S_'+ i  for i in one_hot_encoder.categories_[0]])
dfOneHot
```

	S_Female	S_Male	S_Other
0	0.0	1.0	0.0
1	1.0	0.0	0.0
2	0.0	0.0	1.0
3	0.0	1.0	0.0

```
students_one_hot = pd.concat([students, dfOneHot], axis=1)
students_one_hot
```

	Name	Sex	Age	S_Female	S_Male	S_Other
0	John	Male	16	0.0	1.0	0.0
1	Lucy	Female	17	1.0	0.0	0.0
2	Lyly	Other	16	0.0	0.0	1.0
3	Mark	Male	17	0.0	1.0	0.0

Tạo thuộc tính (feature)

```
one_hot_encoder_n = OneHotEncoder(drop='first').fit(students[['Sex']])
students_sex_onehot = one_hot_encoder_n.transform(students[['Sex']]).toarray()
```

```
one_hot_encoder_n.categories_[0][1:]
```

```
array(['Male', 'Other'], dtype=object)
```

```
dfOneHot_n = pd.DataFrame(students_sex_onehot,
                          columns = ['S_'+ i  for i in one_hot_encoder_n.categories_[0][1:]])
dfOneHot_n
```

	S_Male	S_Other
0	1.0	0.0
1	0.0	0.0
2	0.0	1.0
3	1.0	0.0

```
students_one_hot_n = pd.concat([students, dfOneHot_n], axis=1)
students_one_hot_n
```

	Name	Sex	Age	S_Male	S_Other
0	John	Male	16	1.0	0.0
1	Lucy	Female	17	0.0	0.0
2	Lyly	Other	16	0.0	1.0
3	Mark	Male	17	1.0	0.0

Tạo thuộc tính (feature)

- Dummy encoder
  - Tương tự như one hot encoder
  - Kết quả trả về là các cột mới trong dataframe
  - Xử lý đơn giản hơn onehot encoder
  - Không lưu trữ dummy model encoder

Tạo thuộc tính (feature)

- Ví dụ

```
students
```

	Name	Sex	Age
0	John	Male	16
1	Lucy	Female	17
2	Lyly	Other	16
3	Mark	Male	17

```
student_one_hot = pd.get_dummies(students, columns=['Sex'] ,prefix='S')
student_one_hot
```

	Name	Age	S_Female	S_Male	S_Other
0	John	16	0	1	0
1	Lucy	17	1	0	0
2	Lyly	16	0	0	1
3	Mark	17	0	1	0

Tạo thuộc tính (feature)

- Ví dụ: Có tham số drop_first = True

```
students_dummy = pd.get_dummies(students, columns=['Sex'], prefix='S', drop_first=True)
students_dummy
```

	Name	Age	S_Male	S_Other
0	John	16	1	0
1	Lucy	17	0	0
2	Lyly	16	0	1
3	Mark	17	1	0

Tạo thuộc tính (feature)

❑ Xử lý các danh mục không phổ biến (uncommon category)

- Tính năng có thể có nhiều loại khác nhau nhưng phân phối rất không đồng đều về sự xuất hiện của chúng.
  - Lấy ví dụ các ngôn ngữ yêu thích của Data Science để mã hóa, một số lựa chọn phổ biến là Python, R, nhưng có thể có các cá nhân với các lựa chọn riêng biệt, như FORTRAN, C, v.v. Trong những trường hợp này, chúng ta có thể không muốn tạo một tính năng cho mỗi giá trị mà có thể gộp lại thành một tính năng cho các giá trị ít xuất hiện.

Tạo thuộc tính (feature)

- Ví dụ:

```
ser_country = df.Country.value_counts()
ser_country
```

```
South Africa    166
USA             164
Spain           134
Sweeden         119
France          115
Russia           97
UK               95
India            95
Ukraine           9
Ireland           5
```

```
countries = ser_country[ser_country<=10]
countries
```

```
Ukraine    9
Ireland    5
Name: Country, dtype: int64
```

```
# gom các country có đếm <=10 --> tên Other
df.loc[df['Country'].isin(countries.index), 'Country'] = 'Other'
```

```
df.Country.value_counts()
```

```
South Africa    166
USA             164
Spain           134
Sweeden         119
France          115
Russia           97
UK               95
India            95
Other            14
```

Tạo thuộc tính (feature)

Tạo cột nhị phân (Binarizing column)

- Mặc dù hầu hết các giá trị số thường được sử dụng mà không cần bất kỳ feature engineering nào, sẽ có trường hợp một số hình thức thao tác có thể hữu ích.
  - Ví dụ: Trong một số trường hợp, chúng ta có thể không quan tâm đến độ lớn của giá trị mà chỉ quan tâm đến hướng của nó hoặc nó có tồn tại hay không. Trong những tình huống này, chúng ta có thể tạo một cột mới.

Tạo thuộc tính (feature)

- Ví dụ: Trong dữ liệu so_survey_df, có một số lượng lớn người trả lời khảo sát đang làm việc tình nguyện (không nhận tiền công) -> chúng ta tạo cột mới có tiêu đề Paid_Job với mỗi người được trả công (mức lương của họ lớn hơn 0) sẽ = 1.

```
# Create the Paid_Job column filled with zeros
so_survey_df['Paid_Job'] = 0

# Replace all the Paid_Job values where ConvertedSalary is > 0
so_survey_df.loc[so_survey_df['ConvertedSalary'] > 0, 'Paid_Job'] = 1

# Print the first five rows of the columns
print(so_survey_df[['Paid_Job', 'ConvertedSalary']].head())
```

```
   Paid_Job  ConvertedSalary
0         0             NaN
1         1         70841.0
2         0             NaN
3         1         21426.0
4         1         41671.0
```

Tạo thuộc tính (feature)

Binning value

- Đối với biến liên tục, có thể ít quan tâm hơn về giá trị chính xác, thay vào đó quan tâm đến nhóm mà giá trị rơi vào.
- Điều này hữu ích khi trực quan hóa các giá trị hoặc đơn giản hóa các mô hình ML.
- Chủ yếu sử dụng trên các biến liên tục trong đó độ chính xác không phải là mối quan tâm lớn nhất, ví dụ: tuổi, lương...

Tạo thuộc tính (feature)

- Đặc điểm
  - Động lực chính của việc tạo bin là làm cho mô hình mạnh mẽ hơn và ngăn ngừa overfitting, tuy nhiên, nó có chi phí hiệu suất cao hơn. Mỗi khi chúng ta sử dụng bin một cột nào đó, chúng ta chấp nhận bớt thông tin.
  - Sự đánh đổi giữa hiệu suất và overfitting là điểm mấu chốt của quy trình tạo bin. Nhìn chung, đối với các cột số, ngoại trừ một số trường hợp quá rõ ràng, việc tạo bin có thể là dư thừa đối với một số loại thuật toán.

Tạo thuộc tính (feature)

- Các bin được tạo ra bằng cách sử dụng:

```
pd.cut(df['column_name'], bins)
```

  - Trong đó, bins chỉ định số lượng bin cách đều nhau hoặc danh sách các ranh giới của bin.

Tạo thuộc tính (feature)

- Ví dụ

```
# Bin the continuous variable ConvertedSalary into 5 bins
so_survey_df['equal_binned'] = pd.cut(so_survey_df['ConvertedSalary'], 5)
so_survey_df['equal_binned'].unique()
```

```
[NaN, (-2000.0, 400000.0], (800000.0, 1200000.0], (400000.0, 800000.0], (1600000.0, 2000000.0]]
Categories (4, interval[float64]): [(-2000.0, 400000.0] < (400000.0, 800000.0] < (800000.0, 1200000.0] < (1600000.0, 2000000.
0]]
```

```
# Print the first 5 rows of the equal_binned column
print(so_survey_df[['equal_binned', 'ConvertedSalary']].head())
```

```
          equal_binned  ConvertedSalary
0                  NaN              NaN
1  (-2000.0, 400000.0]          70841.0
2                  NaN              NaN
3  (-2000.0, 400000.0]          21426.0
4  (-2000.0, 400000.0]          41671.0
```

Tạo thuộc tính (feature)

- Ví dụ: Tạo danh sách nhãn cho các bin

```
# Import numpy
import numpy as np

# Specify the boundaries of the bins
bins = [-np.inf, 10000, 50000, 100000, 150000, np.inf]

# Bin labels
labels = ['Very low', 'Low', 'Medium', 'High', 'Very high']

# Bin the continuous variable ConvertedSalary using these boundaries
so_survey_df['boundary_binned'] = pd.cut(so_survey_df['ConvertedSalary'],
                                         bins, labels = labels)

# Print the first 5 rows of the boundary_binned column
print(so_survey_df[['boundary_binned', 'ConvertedSalary']].head())
```

```
  boundary_binned  ConvertedSalary
0             NaN             NaN
1          Medium         70841.0
2             NaN             NaN
3             Low         21426.0
4             Low         41671.0
```

Nội dung

1. Giới thiệu
2. Tạo thuộc tính (feature)
3. **Chuẩn hóa dữ liệu (Data Standardization)**
4. Chuyển dạng dữ liệu (Data Transformation)

Chuẩn hóa dữ liệu (Data Standardization)

❑ Giới thiệu

- Chuẩn hóa dữ liệu là quy trình thực hiện việc chuyển đổi cấu trúc của các bộ dữ liệu khác nhau thành định dạng dữ liệu chung (Common Data Format).
- Chuẩn hóa dữ liệu chuyển đổi các bộ dữ liệu sau khi dữ liệu được lấy từ các hệ thống nguồn và trước khi nó được tải vào các hệ thống đích.
- Chuẩn hóa dữ liệu cho phép người dùng phân tích và sử dụng dữ liệu một cách nhất quán.

Chuẩn hóa dữ liệu (Data Standardization)

❑ Giới thiệu

- Các bộ dữ liệu trong thế giới thực chứa các tính năng rất khác nhau về đơn vị và phạm vi.
- Chuẩn hóa dữ liệu thực hiện khi thang đo của một tính năng không liên quan hoặc gây hiểu lầm.
- Có nhiều phương pháp chuẩn hóa dữ liệu như logarith, scaling based on distribution, …

Chuẩn hóa dữ liệu (Data Standardization)

Khi nào cần chuẩn hóa dữ liệu?

- Khi xây dựng mô hình:
  - Mô hình trong không gian tuyến tính (linear space)
  - Các tính năng dữ liệu có phương sai cao
  - Các tính năng dữ liệu liên tục và trên các đơn vị đo (scale) khác nhau
  - Giả định tuyến tính

Log normalization

- Log của một biến số là phương thức biến đổi phổ biến được sử dụng để thay đổi hình dạng phân phối của biến trên biểu đồ phân phối.

- Được sử dụng để giảm độ lệch phải (right skewness) của biến.

Log normalization

- **Sử dụng khi nào?**
  - Dữ liệu có độ lệch phải dương.
  - Dữ liệu có phương sai cao.
  - Dữ liệu có các outlier.

Log normalization

- Áp dụng chuyển đổi log
- Log tự nhiên sử dụng hằng số e (2.718)
- Nắm bắt những thay đổi tương đối, cường độ thay đổi và giữ mọi thứ trong không gian dương (positive space)
- Chia dữ liệu vào phạm vi nhỏ hơn theo log
- Dùng cho Skewed và Wide Distribution

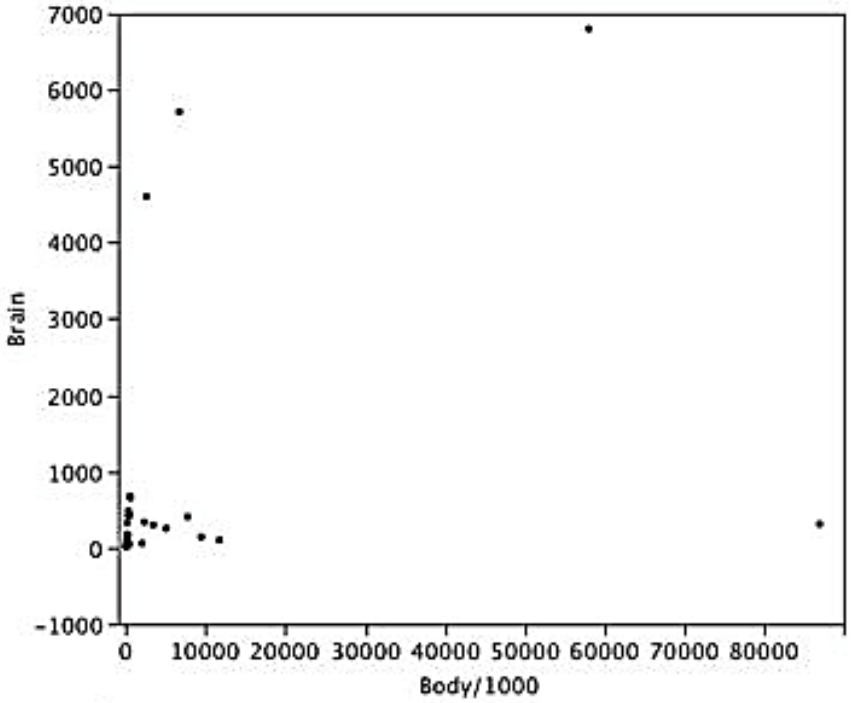
Log normalization

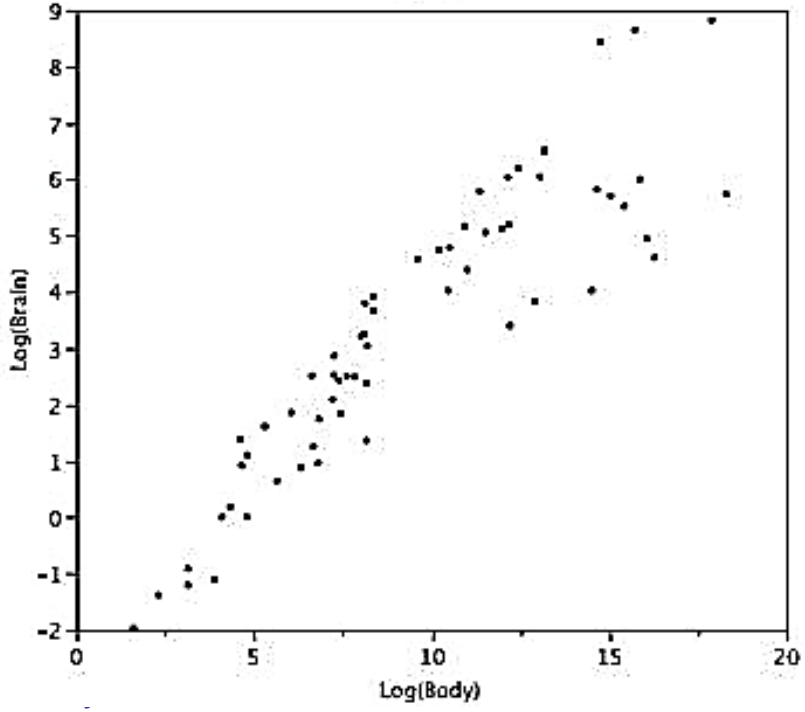
Ưu điểm

- Việc sử dụng log normalization có thể được sử dụng để làm cho các bản phân phối có độ lệch cao ít bị sai lệch.
- Giúp cho các mẫu trong dữ liệu dễ hiểu hơn và giúp đáp ứng các giả định của thống kê suy luận.

Log normalization

- Ví dụ 1:
  - Hình dưới cho thấy một ví dụ về cách chuyển đổi log có thể làm cho các mẫu hiển thị rõ hơn. Cả hai biểu đồ biểu thị trọng lượng não và trọng lượng cơ thể của động vật. Các giá trị thô được hiển thị trong hình trái; các giá trị chuyển đổi log được hiển thi trong hình phải.





Log normalization

- Ví dụ 2: Sử dụng np.log() áp dụng log normalization

```
wine_sub.head()
```

	Alcohol	Proline	Proanthocyanins	Ash
0	14.23	1065	2.29	2.43
1	13.20	1050	1.28	2.14
2	13.16	1185	2.81	2.67
3	14.37	1480	2.18	2.50
4	13.24	735	1.82	2.87

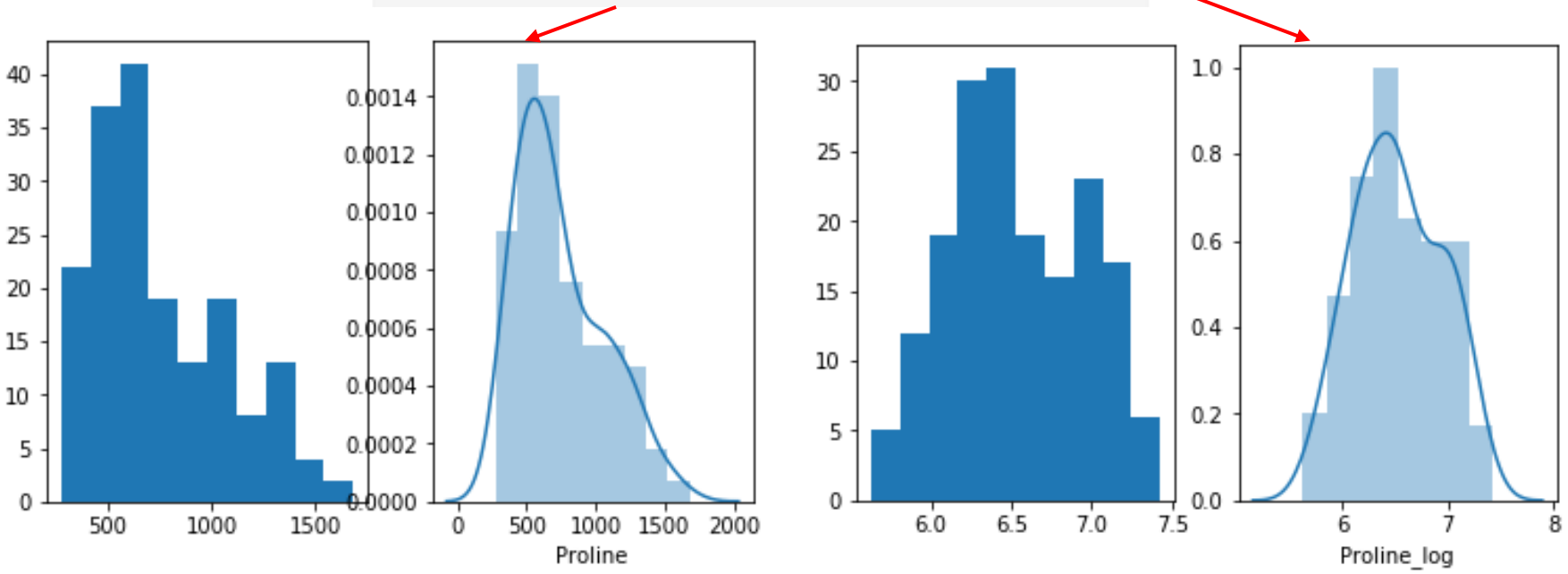
```
# dữ liệu cột cần được chuẩn hóa? Proline
print("The variance of the Alcohol column:", wine_sub["Alcohol"].var())

print("The variance of the Proline column:", wine_sub["Proline"].var())

wine_sub["Proline_log"] = np.log(wine_sub["Proline"])

# Check the variance of the Proline column again
print("The variance of the Proline column after log:", wine_sub["Proline_log"].var())
```

```
The variance of the Alcohol column: 0.6590623278105763
The variance of the Proline column: 99166.71735542428
The variance of the Proline column after log: 0.17231366191842018
```

Log normalization

```
wine_sub.head()
```

	Alcohol	Proline	Proanthocyanins	Ash	Proline_log
0	14.23	1065	2.29	2.43	6.970730
1	13.20	1050	1.28	2.14	6.956545
2	13.16	1185	2.81	2.67	7.077498
3	14.37	1480	2.18	2.50	7.299797
4	13.24	735	1.82	2.87	6.599870



Feature Scaling

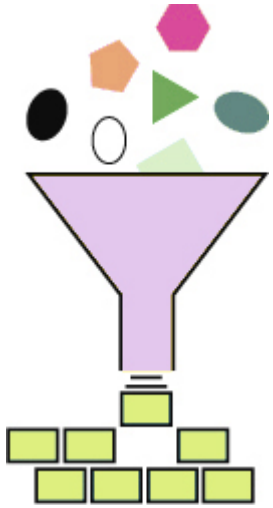
Giới thiệu

- Chuẩn hóa dữ liệu trong một phạm vi cụ thể.
- Áp dụng cho các tính năng của dữ liệu.
- Giúp tăng tốc tính toán trong một thuật toán.

Feature Scaling

Khi nào thì áp dụng Feature Scaling?

- Các tính năng có thể ở các thang đo khác nhau.
- Các mô hình có đặc điểm tuyến tính.



Feature Scaling

- Các thuật toán sử dụng thước đo Khoảng cách Euclide rất nhạy cảm với **độ lớn**.

- Nếu một tính năng trong bộ dữ liệu có quy mô lớn so với các tính năng khác thì trong các thuật toán được đo bằng khoảng cách Euclide, tính năng có tỷ lệ lớn sẽ trở nên thống trị và cần được chuẩn hóa → dùng Feature scaling giúp cân đối các tính năng.

Feature Scaling

- Các thuật toán cần chuẩn hoá dữ liệu:
  - PCA: Cố gắng để có được tính năng với phương sai tối đa → phải scaling.
  - Gradient Descent: Tốc độ tính toán tăng khi tính toán Theta trở nên nhanh hơn sau khi scaling.
  - K-nearest neighbors (KNN): sử dụng đo khoảng cách bằng Euclidean, nếu chúng ta muốn tất cả các thuộc tính có đóng góp như nhau.
  - K-means: các cụm cũng được xem xét gần nhau theo khoảng cách như KNN.

Feature Scaling

- Logistic regression, SVMs, perceptrons, neural networks.... nếu chúng ta sử dụng tối ưu hoá là gradient descent/ ascent-based optimization thì cần scale dữ liệu, nếu không, một số trọng số sẽ cập nhật nhanh hơn các trọng số khác.
- Linear discriminant analysis (LDA), principal component analysis (PCA), kernel principal component analysis: khi chúng ta muốn tìm hướng tối đa hoá phương sai (variance) theo các ràng buộc về directions/ eigenvectors/ principal components orthogonal thì các thuộc tính cần có cùng một tỉ lệ.

Feature Scaling

- Chú ý
  - Naive Bayes, Tree-Based models… không bị ảnh hưởng bởi feature scaling. Do đó, các thuật toán không dựa trên khoảng cách thì không nhất thiết phải áp dụng feature scaling.

Feature Scaling

❑ Sử dụng thư viện: sklearn.preprocessing

- Là thư viện cung cấp một số hàm tiện ích phổ biến và các lớp biến đổi để thay đổi các vector tính năng thô thành biểu diễn phù hợp hơn.
- Một số Scaler thông dụng
  - StandardScaler
  - MinMaxScaler
  - RobustScaler
  - Binarizer
  - …

https://scikit-learn.org/stable/modules/preprocessing.html#standardization-or-mean-removal-and-variance-scaling

Feature Scaling

StandardScaler

- StandardScaler là một kỹ thuật hữu ích để biến đổi các thuộc tính có phân phối Gaussian và các giá trị trung bình (mean) & độ lệch chuẩn (std) khác nhau thành phân phối Gaussian tiêu chuẩn với giá trị trung bình là 0 và độ lệch chuẩn là 1. Và tuân theo công thức sau cho mỗi tính năng/ thuộc tính:

$$\frac{x_i - mean(x)}{stdev(x)}$$

Feature Scaling

- StandardScaler phù hợp nhất cho các kỹ thuật giả định phân phối Gaussian trong các biến đầu vào và hoạt động tốt hơn với dữ liệu cần được chuẩn hóa lại, như hồi quy tuyến tính, hồi quy logistic...
- Nếu dữ liệu không được phân phối chuẩn thì đây không phải là cách chia tỷ lệ tốt nhất để sử dụng.

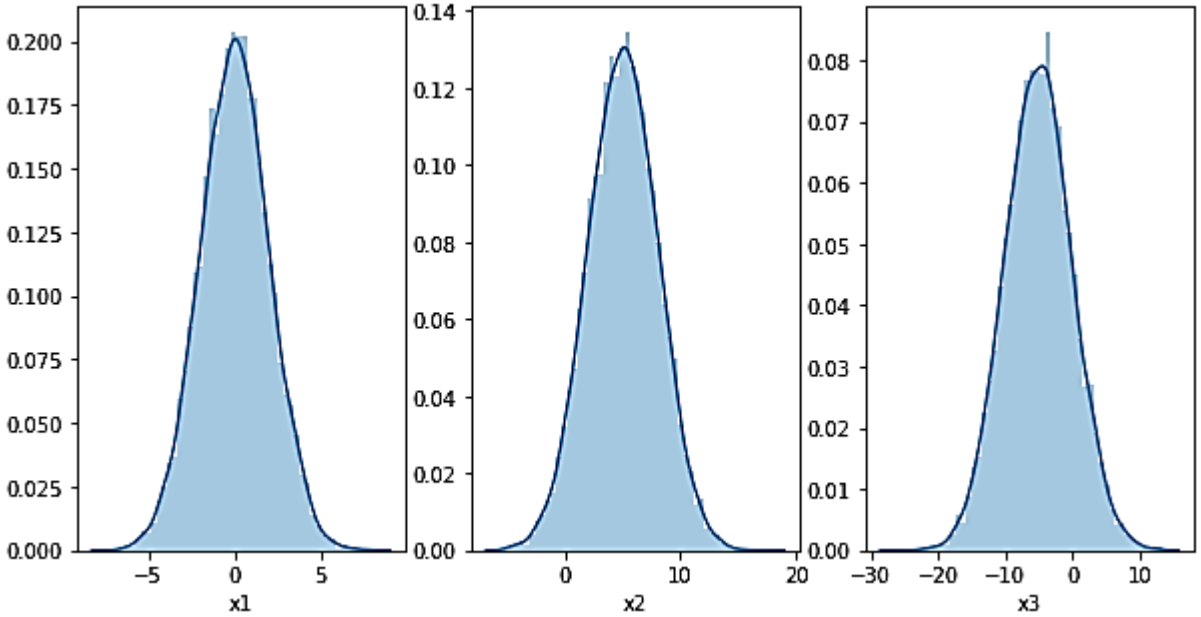
Feature Scaling

- Ví dụ

```
np.random.seed(1)
df = pd.DataFrame({
    'x1': np.random.normal(0, 2, 10000),
    'x2': np.random.normal(5, 3, 10000),
    'x3': np.random.normal(-5, 5, 10000)
})
```

```
df.head()
```

	x1	x2	x3
0	3.248691	4.632578	-14.657819
1	-1.223513	5.684509	-5.802131
2	-1.056344	3.943085	-9.161098
3	-2.145937	2.508340	-6.030558
4	1.730815	4.216731	0.131275



Phân phối Gaussian
=> StandardScaler

Feature Scaling

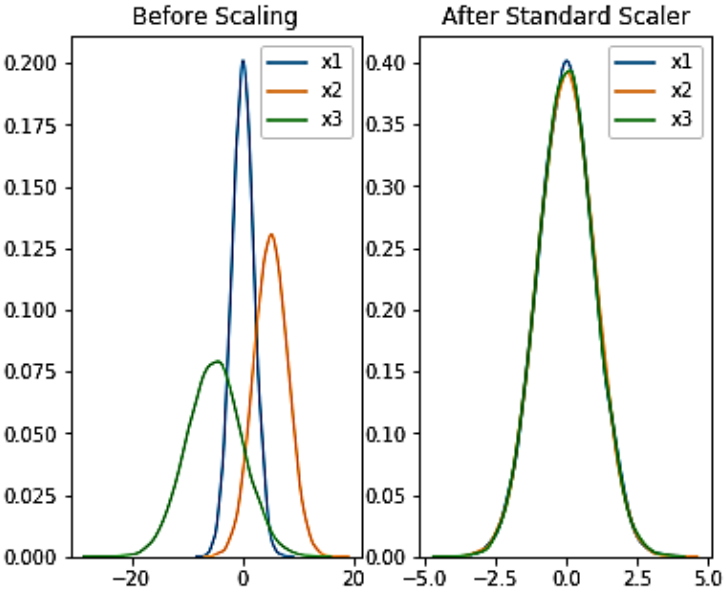
```
df2.head()
```

	x1	x2	x3
0	3.248691	4.632578	-14.657819
1	-1.223513	5.684509	-5.802131
2	-1.056344	3.943085	-9.161098
3	-2.145937	2.508340	-6.030558
4	1.730815	4.216731	0.131275

```
scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df2)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])
```

```
scaled_df.head()
```

	x1	x2	x3
0	1.616535	-0.131753	-1.924757
1	-0.622285	0.218475	-0.146416
2	-0.538598	-0.361311	-0.820942
3	-1.084057	-0.838991	-0.192287
4	0.856675	-0.270204	1.045092



Feature Scaling

MinMaxScaler

- MinMaxScaler có thể được xem là thuật toán chia tỷ lệ nổi tiếng nhất và tuân theo công thức sau cho mỗi tính năng/ thuộc tính:

$$\frac{x_i - min(x)}{max(x) - min(x)}$$

- Về cơ bản, nó thu hẹp phạm vi sao cho phạm vi mới nằm trong khoảng từ 0 đến 1.

Feature Scaling

- Bộ chia tỷ lệ này hoạt động tốt hơn trong các trường hợp mà bộ chia tỷ lệ tiêu chuẩn (standard scaler) có thể không hoạt động tốt.
- Nếu phân phối không phải là Gaussian hoặc độ lệch chuẩn là rất nhỏ, min-max scaler hoạt động tốt hơn.
- Động lực để sử dụng tỷ lệ này bao gồm độ lệch chuẩn của các tính năng rất nhỏ và duy trì các entry có giá trị 0 trong dữ liệu thưa thớt.

Feature Scaling

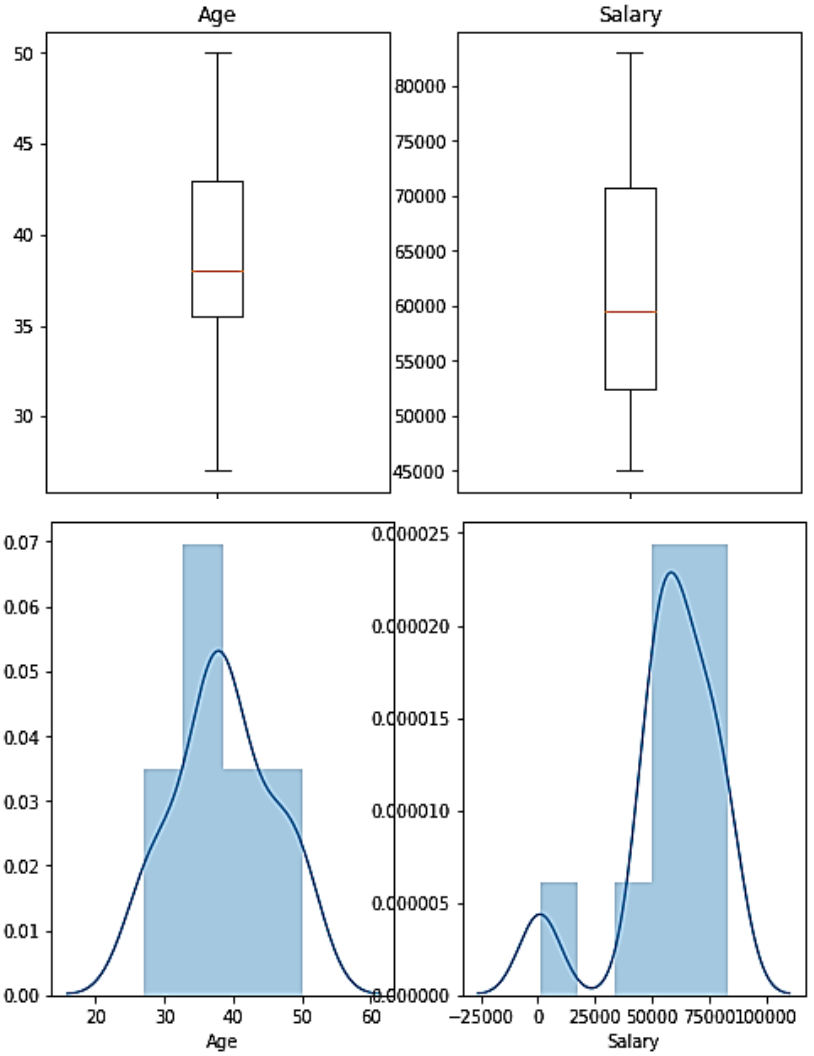
- Tuy nhiên, nó rất nhạy cảm với các ngoại lệ (outlier), vì vậy nếu có các ngoại lệ trong dữ liệu, chúng ta có thể áp dụng Robust Scaler

Feature Scaling

- Ví dụ

X

	Age	Salary
0	44.0	72000.0
1	27.0	48000.0
2	30.0	54000.0
3	38.0	61000.0
4	40.0	45000.0
5	35.0	58000.0
6	38.0	52000.0
7	48.0	79000.0
8	50.0	83000.0
9	37.0	67000.0



- **Phân phôi không phải là Gausian**
- **Không có oulier**

=> MinMaxScaler

Feature Scaling

- Ví dụ

X

	Age	Salary
0	44.0	72000.0
1	27.0	48000.0
2	30.0	54000.0
3	38.0	61000.0
4	40.0	45000.0
5	35.0	58000.0
6	38.0	52000.0
7	48.0	79000.0
8	50.0	83000.0
9	37.0	67000.0

```
from sklearn import preprocessing
```

```
min_max_scaler = preprocessing.MinMaxScaler()  # feature_range =(0, 1)
```

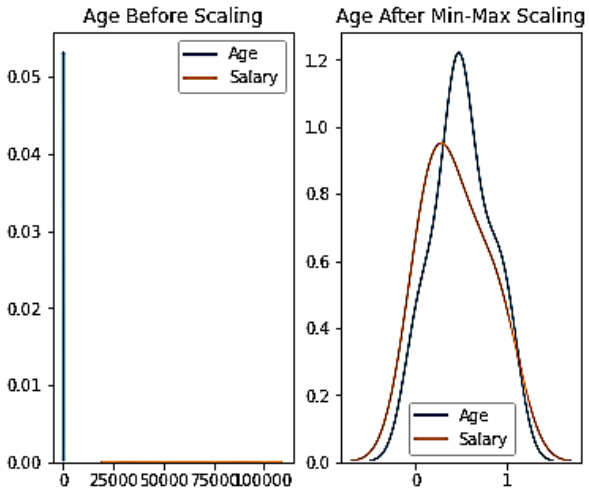
```
X_after_min_max_scaler = min_max_scaler.fit_transform(X)
```

```
X_after_min_max_scaler = pd.DataFrame(X_after_min_max_scaler, columns=['Age', 'Salary'])
```

```
print("After min max Scaling :")
X_after_min_max_scaler
```

```
After min max Scaling :
```

	Age	Salary
0	0.739130	0.710526
1	0.000000	0.078947
2	0.130435	0.236842
3	0.478261	0.421053
4	0.565217	0.000000
5	0.347826	0.342105
6	0.478261	0.184211
7	0.913043	0.894737
8	1.000000	1.000000
9	0.434783	0.578947



Feature Scaling

Robust Scaler

- RobustScaler sử dụng một phương pháp tương tự như Min-Max Scaler nhưng nó sử dụng Interquartile range thay cho min-max.
- Tính toán theo công thức sau cho mỗi tính năng/ đặc tính:

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

Feature Scaling

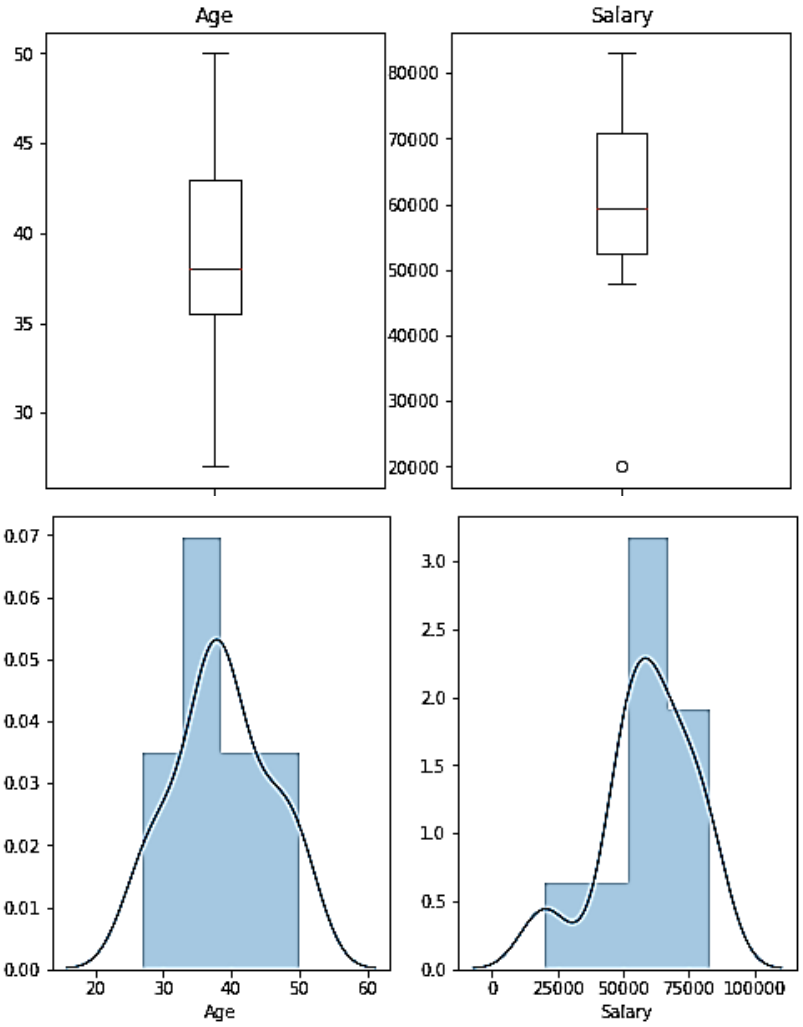
- Dĩ nhiên, điều này có nghĩa là RobustScaler đang sử dụng ít dữ liệu hơn khi chia tỷ lệ để nó phù hợp hơn khi có các ngoại lệ trong dữ liệu.
- Chú ý: Sau khi áp dụng Robust scaling, các phân phối được dựa vào cùng một tỷ lệ và trùng lắp, nhưng các ngoại lệ vẫn nằm ngoài phần lớn của các bản phân phối mới.

Feature Scaling

- Ví dụ

X3

	Age	Salary
0	44.0	72000.0
1	27.0	48000.0
2	30.0	54000.0
3	38.0	61000.0
4	40.0	20000.0
5	35.0	58000.0
6	38.0	52000.0
7	48.0	79000.0
8	50.0	83000.0
9	37.0	67000.0



- **Phân phối không phải là Gausian**
- **Có oulier**

=> RobustScaler

Feature Scaling

- Ví dụ:

X3

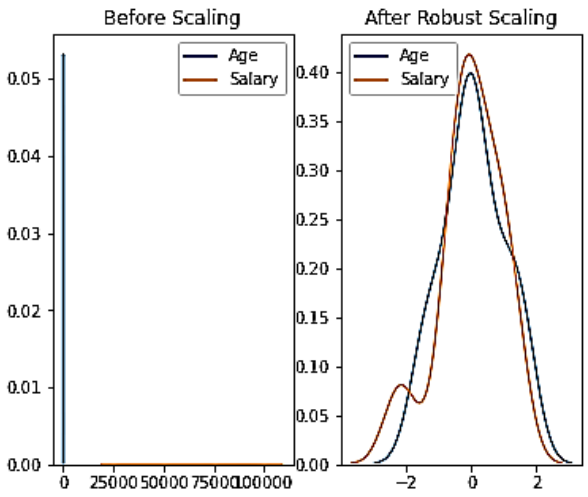
	Age	Salary
0	44.0	72000.0
1	27.0	48000.0
2	30.0	54000.0
3	38.0	61000.0
4	40.0	20000.0
5	35.0	58000.0
6	38.0	52000.0
7	48.0	79000.0
8	50.0	83000.0
9	37.0	67000.0

```
scaler = preprocessing.RobustScaler()
robust_scaled_df = scaler.fit_transform(X3)
robust_scaled_df = pd.DataFrame(robust_scaled_df, columns=['Age', 'Salary'])
```

```
print("After Robust Scaling:")
robust_scaled_df
```

After Robust Scaling:

	Age	Salary
0	0.800000	0.684932
1	-1.466667	-0.630137
2	-1.066667	-0.301370
3	0.000000	0.082192
4	0.266667	-2.164384
5	-0.400000	-0.082192
6	0.000000	-0.410959
7	1.333333	1.068493
8	1.600000	1.287671
9	-0.133333	0.410959



Feature Scaling

Binarizer

- Có thể chuyển đổi dữ liệu bằng cách dùng binary threshold. Tất cả các giá trị trên threshold được thay bằng 1 và các giá trị <= threshold được thay bằng 0.
- Nó có thể hữu ích khi chúng ta có các xác suất mà ta muốn tạo ra các giá trị rõ nét.
- Nó cũng hữu ích khi feature engineering và chúng ta muốn thêm các tính năng mới có ý nghĩa.

Feature Scaling

- Ví dụ
  - Ví dụ 1: Dữ liệu liên tục của các giá trị pixel của ảnh với grayscale 8-bit có các giá trị nằm trong khoảng từ 0 (đen) đến 255 (trắng) và người ta cần có màu đen và trắng. Vì vậy, bằng cách sử dụng Binarizer (), người ta có thể đặt ngưỡng chuyển đổi các giá trị pixel từ 0 - 127 thành 0 và 128 - 255 thành 1.
  - Ví dụ 2: Một bản ghi của một máy có thuộc tính “Success Percentage”. Các giá trị này liên tục nằm trong khoảng từ 10% đến 99% nhưng một nhà nghiên cứu chỉ muốn sử dụng dữ liệu này để dự đoán tình trạng vượt qua hoặc thất bại cho máy dựa trên các tham số đã cho khác.

Feature Scaling

- Ví dụ:

```
Y = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.],
     [ 3. , -2., 1. ]]
```

```
scaler = preprocessing.Binarizer(threshold=0)
scaled_Y = scaler.fit_transform(Y)
scaled_Y
```

```
array([[1., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 1.]])
```

Nội dung

1. Giới thiệu
2. Tạo thuộc tính (feature)
3. Chuẩn hóa dữ liệu (Data Standardization)
4. **Chuyển dạng dữ liệu (Data Transformation)**

Chuyển dạng dữ liệu (Data Transformation)

- Một số loại dữ liệu "chưa gọn gàng" (untidying data) cần được giải quyết:
  - Tiêu đề cột là giá trị, không phải tên biến.
  - Nhiều biến được lưu trữ trong một cột.
  - Các biến được lưu trữ trong cả hàng và cột.
  - Nhiều đơn vị mẫu khác nhau được lưu trữ trong cùng một bảng
  - Một mẫu quan sát duy nhất được lưu trữ trong nhiều bảng/ tập tin.

Chuyển dạng dữ liệu (Data Transformation)

- Dữ liệu gọn gàng (tidy data) là dữ liệu thu được từ kết quả của một quá trình gọi là thu dọn dữ liệu.

- Bộ dữ liệu gọn gàng có cấu trúc và làm việc với chúng rất dễ dàng: dễ dàng thao tác, mô hình hóa và trực quan hóa.

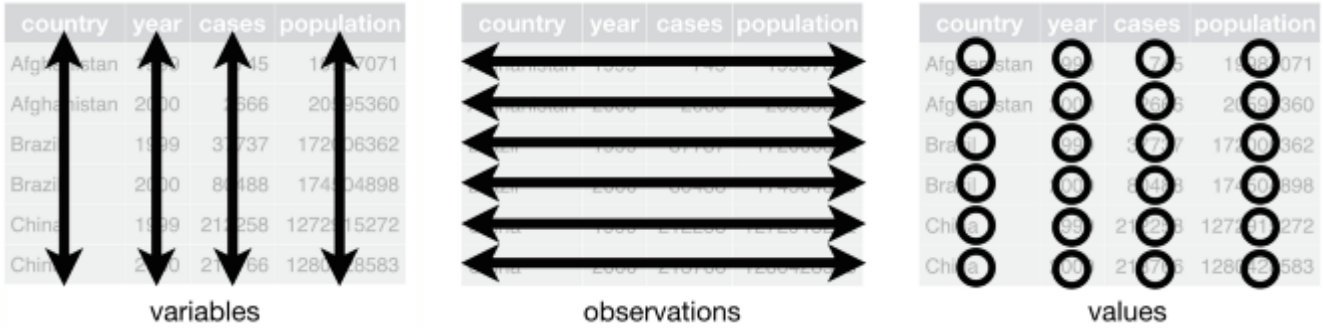
- Các tập dữ liệu gọn gàng được sắp xếp sao cho mỗi biến là một cột và mỗi quan sát (trường hợp) là một hàng.

Theo: *https://en.wikipedia.org/wiki/Tidy_data*

Chuyển dạng dữ liệu (Data Transformation)

❑ Đặc điểm của tidy data

- Mỗi biến đo lường (variable/ feature) phải ở trong một cột.
- Mỗi mẫu (sample/ observation) phải ở trên một hàng.
- Mỗi loại mẫu là một bảng
- Nếu có nhiều bảng, chúng nên có một cột trong bảng cho phép chúng liên kết.

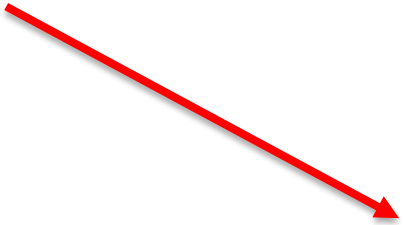


Theo "Jeff Leek - *The Elements of Data Analytic Style*" **:** https://en.wikipedia.org/wiki/Tidy_data

Chuyển dạng dữ liệu (Data Transformation)

- Ví dụ
  - Dữ liệu này không phải là “tidy data” vì tên cột chứa thông tin về phép đo được thực hiện: số lượng xi lanh (cyl), công suất ngựa (hp) và số thùng bộ chế hòa khí (carb).

maker	cyl	hp	carb
Delorian	4	160	4
Fantom	2	80	2



Chuyển thành “tidy data”

maker	metric	value
Delorian	cyl	4
Fantom	cyl	2
Delorian	hp	160
Fantom	hp	80
Delorian	carb	4
Fantom	carb	2

Chuyển dạng dữ liệu (Data Transformation)

Chuyển dữ liệu thành Tidy data

- Vấn đề về dữ liệu cần khắc phục
  - Cột chứa giá trị thay vì chứa biến
- Giải pháp
  - Dùng pd.melt()

Chuyển dạng dữ liệu (Data Transformation)

```
df
```

	marker	cyl	hp	carb
0	Delorian	4	160	4
1	Fantom	2	80	2

```
df_new = pd.melt(frame=df, id_vars="marker", value_vars=["cyl", "hp", "carb"], var_name="metric")
df_new
```

	marker	metric	value
0	Delorian	cyl	4
1	Fantom	cyl	2
2	Delorian	hp	160
3	Fantom	hp	80
4	Delorian	carb	4
5	Fantom	carb	2

Chuyển dạng dữ liệu (Data Transformation)

Pivoting data (un-melting data)

- Ngược lại với melting data
  - Trong melting data, chúng ta chuyển các cột thành các dòng
  - Trong pivoting data: chuyển các giá trị duy nhất thành các cột riêng biệt
  - Dùng để tạo các báo cáo
  - Vi phạm nguyên tắc của tidy data: các dòng chứa các mẫu
    - Nhiều biến được lưu trữ trong cùng một cột

Chuyển dạng dữ liệu (Data Transformation)

```
df_new
```

	marker	metric	value
0	Delorian	cyl	4
1	Fantom	cyl	2
2	Delorian	hp	160
3	Fantom	hp	80
4	Delorian	carb	4
5	Fantom	carb	2

```
df_pivot = df_new.pivot(index = 'marker', columns='metric', values='value')
df_pivot
```

metric	carb	cyl	hp
marker			
Delorian	4	4	160
Fantom	2	2	80

Chuyển dạng dữ liệu (Data Transformation)

```
df_new = df_new.append({'marker' : 'Fantom' , 'metric' : 'carb', 'value' : 4} , ignore_index=True)
df_new
```

	marker	metric	value
0	Delorian	cyl	4
1	Fantom	cyl	2
2	Delorian	hp	160
3	Fantom	hp	80
4	Delorian	carb	4
5	Fantom	carb	2
6	Fantom	carb	4

```
df_pivot_2 = df_new.pivot(index = 'marker', columns='metric', values='value')
df_pivot_2
```

```
ValueError: Index contains duplicate entries, cannot reshape
```

Chuyển dạng dữ liệu (Data Transformation)

Phương thức pivot_table

- Có tham số aggfunc chỉ định cách xử lý trùng lặp giá trị
  - Ví dụ: Có thể tổng hợp các giá trị trùng lặp bằng cách lấy trung bình cộng (np.mean)

Chuyển dạng dữ liệu (Data Transformation)

```
df_new
```

	marker	metric	value
0	Delorian	cyl	4
1	Fantom	cyl	2
2	Delorian	hp	160
3	Fantom	hp	80
4	Delorian	carb	4
5	Fantom	carb	2
6	Fantom	carb	4

```
# Pivot table
df_pivot_3 = df_new.pivot_table(index = 'marker', columns='metric',
                                values='value', aggfunc= np.mean)
df_pivot_3
```

metric	carb	cyl	hp
marker			
Delorian	4	4	160
Fantom	3	2	80