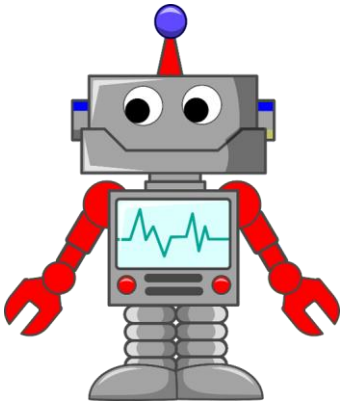
ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

CS116 – LẬP TRÌNH PYTHON CHO MÁY HỌC

Introduction Feature Engineering & Selection



TS. Nguyễn Vinh Tiệp

NỘI DUNG

- Feature Engineering & Data Transformation
- Feature Engineering Technique
- Feature Engineering Faster
- Feature Selection Techniques & Tools

Tiền xử lý Dữ liệu

Xử lý các giá trị bị thiếu và ngoại lệ

Các loại giá trị bị thiếu

Dữ liệu bị thiếu về mặt cấu trúc

Những giá trị này bị thiếu vì lẽ ra chúng không tồn tại.

Dữ liệu thiếu hoàn toàn ngẫu nhiên
MCAR (Missing Completely At Random)

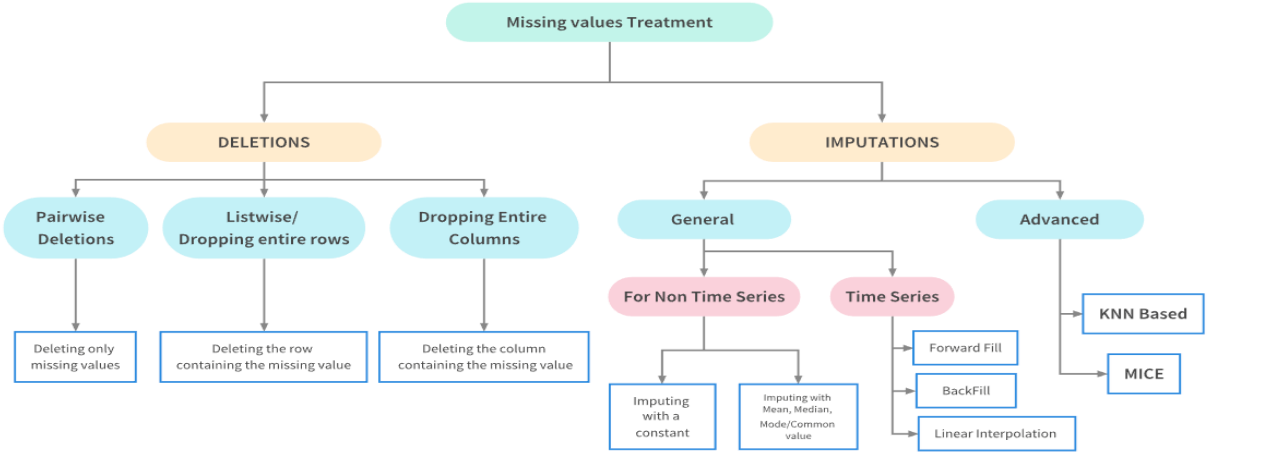
Các giá trị bị thiếu xảy ra hoàn toàn ngẫu nhiên.

Dữ liệu thiếu ngẫu nhiên
MAR (Missing At Random)
Giả định ở đây là các giá trị còn thiếu có liên quan phần nào đến các quan sát khác trong dữ liệu.

Dữ liệu thiếu không ngẫu nhiên
NMAR (Not Missing At Random)
Các giá trị còn thiếu ở đây có nguồn gốc không phải ngẫu nhiên mà có chủ ý

Xử lý các giá trị bị thiếu

- Có ba cách tiếp cận chính để xử lý vấn đề thiếu giá trị:
  - Bằng cách loại bỏ
  - Bằng cách thay thế: sklearn-imputation of missing values
  - Các mô hình (Công cụ ước tính) có thể xử lý giá trị NaN
  - Cách tiếp cận khác: tạo cột mới chứa thông tin có giá trị bị thiếu



https://www.kaggle.com/code/parulpandey/a-guide-to-handling-missing-values-in-python/notebook

Loại bỏ theo danh sách

- Loại bỏ theo danh sách
  - Loại bỏ toàn bộ hàng hoặc cột chứa giá trị còn thiếu
  - Phương pháp này đơn giản nhưng có thể dẫn đến mất dữ liệu nếu giá trị bị thiếu không phải là MCAR.

Loại bỏ theo cặp

- Loại bỏ theo cặp
  - Được sử dụng khi giá trị là MCAR hoặc MAR. Trong quá trình loại bỏ theo cặp, chỉ những giá trị còn thiếu mới bị loại.
  - Phương pháp này tối đa hóa dữ liệu được sử dụng trong phân tích nhưng có thể dẫn đến kết quả không nhất quán nếu việc thiếu dữ liệu không phải là ngẫu nhiên.

Thay thế đặc trưng đơn biến

- Các giá trị bị thiếu có thể được thay bằng một giá trị hằng số được cung cấp hoặc sử dụng số liệu thống kê (trung bình, trung vị hoặc giá trị thường xuyên nhất) của mỗi cột chứa các giá trị bị thiếu.

```
# imputing with a constant

from sklearn.impute import SimpleImputer
train_constant = train.copy()
#setting strategy to 'constant'
mean_imputer = SimpleImputer(strategy='constant') # imputing using constant value
train_constant.iloc[:,:] = mean_imputer.fit_transform(train_constant)
train_constant.isnull().sum()
```

```
from sklearn.impute import SimpleImputer
train_most_frequent = train.copy()
#setting strategy to 'mean' to impute by the mean
mean_imputer = SimpleImputer(strategy='most_frequent')# strategy can also be mean or median
train_most_frequent.iloc[:,:] = mean_imputer.fit_transform(train_most_frequent)
```

Thay thế đặc trưng đa biến

- Một chiến lược để gán các giá trị bị thiếu bằng cách mô hình hóa từng đặc trưng có giá trị bị thiếu dưới dạng hàm của các đặc trưng khác theo kiểu vòng tròn.
- Nó thực hiện nhiều phép hồi quy trên mẫu dữ liệu ngẫu nhiên, sau đó lấy trung bình của các giá trị hồi quy và sử dụng giá trị đó để thay thế giá trị còn thiếu.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
train_mice = train.copy(deep=True)

mice_imputer = IterativeImputer()
train_mice['Age'] = mice_imputer.fit_transform(train_mice[['Age']])
```

Thay thế bằng phương pháp K-Nearest Neighbor

- Thay thế các giá trị còn thiếu bằng cách sử dụng phương pháp K-Nearest Neighbors.
- Mỗi đặc trưng bị thiếu được gán bằng cách sử dụng các giá trị từ n_neighbors hàng xóm gần nhất có giá trị cho đặc trưng đó.
- Đặc trưng của các hàng xóm được tính trung bình đồng đều hoặc có trọng số theo khoảng cách đến từng hàng xóm.

```
train_knn = train.copy(deep=True)
```

```
from sklearn.impute import KNNImputer
train_knn = train.copy(deep=True)

knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
train_knn['Age'] = knn_imputer.fit_transform(train_knn[['Age']])
```

Thay thế cho dữ liệu chuỗi thời gian

- Các kỹ thuật tính toán cơ bản:
  - Điền theo phía trước: Thay thế NaN s bằng giá trị được quan sát lần cuối
  - Điền theo phía sau: Thay thế NaN bằng giá trị được quan sát tiếp theo
  - Phương pháp nội suy tuyến tính

```
df.fillna(method = 'ffill', inplace = True)
df.fillna(method = 'bfill', inplace = True)
df.interpolate(limit_direction = "both", inplace = True)
```

2015-02-24	6.05
2015-02-25	0.81
2015-02-26	NaN
2015-02-27	NaN
2015-02-28	NaN
2015-03-01	1.32
2015-03-02	0.22

2015-02-24	6.05
2015-02-25	0.81
2015-02-26	0.81
2015-02-27	0.81
2015-02-28	0.81
2015-03-01	1.32
2015-03-02	0.22

Forward Fill

2015-01-25	NaN
2015-01-26	NaN
2015-01-27	NaN
2015-01-28	NaN
2015-01-29	209.0
2015-01-30	328.0

2015-01-25	209.0
2015-01-26	209.0
2015-01-27	209.0
2015-01-28	209.0
2015-01-29	209.0
2015-01-30	328.0

Back Fill

2015-02-24	6.05
2015-02-25	0.81
2015-02-26	NaN
2015-02-27	NaN
2015-02-28	NaN
2015-03-01	1.32
2015-03-02	0.22

2015-02-24	6.0500
2015-02-25	0.8100
2015-02-26	0.9375
2015-02-27	1.0650
2015-02-28	1.1925
2015-03-01	1.3200
2015-03-02	0.2200

Linear Interpolation

Phương pháp thay thế: Ưu điểm và nhược điểm

Phương pháp	Ưu điểm và nhược điểm
Thay thế đặc trưng đơn biến (mean/median/mode)	• Thay thế giá trị còn thiếu bằng giá trị **trung bình**, **trung vị** hoặc xuất hiện **thường xuyên nhất** của dữ liệu có sẵn cho biến tương ứng.
Thay thế giá trị hằng số	• Thay thế giá trị còn thiếu bằng một giá trị **không đổi**, ví dụ: "không xác định" đối với các biến phân loại.
Thay thế bằng phương pháp K-Nearest Neighbors	• Thay thế giá trị bị thiếu bằng giá trị **trung bình** hoặc thường xuyên nhất của **K lân cận gần nhất** trong không gian đặc trưng. • Phương pháp này có thể cung cấp các phép thay thế **chính xác** hơn nhưng có thể **tốn kém** về mặt tính toán đối với các tập dữ liệu lớn.

Phương pháp thay thế: Ưu điểm và nhược điểm

Phương pháp	Ưu điểm và nhược điểm
Phép nội suy tuyến tính	• Thay thế giá trị bị thiếu bằng giá trị được **nội suy tuyến tính** dựa trên các điểm dữ liệu **không bị thiếu lân cận**. • Giả sử mối quan hệ tuyến tính giữa các điểm dữ liệu và có thể không phù hợp với tất cả các loại dữ liệu
Thay thế bằng phương pháp hồi qui	• Ước tính giá trị còn thiếu bằng cách **khớp mô hình hồi quy** sử dụng các biến khác làm yếu tố dự đoán. • Cung cấp các phép **thay thế chính xác hơn** nhưng có thể gây ra hiện tượng **đa cộng tuyến** và quá khớp nếu các giá trị được gán có mối **tương quan cao** với các yếu tố dự đoán khác.
Thay thế dựa trên mô hình	• Sử dụng **mô hình ML** để **ước tính** các giá trị còn thiếu dựa trên dữ liệu được quan sát. • Phương pháp này có thể cung cấp các phép thay thế có **độ chính xác cao** nhưng có thể **phức tạp hơn** và **tốn kém hơn** về mặt tính toán.

Xử lý ngoại lệ

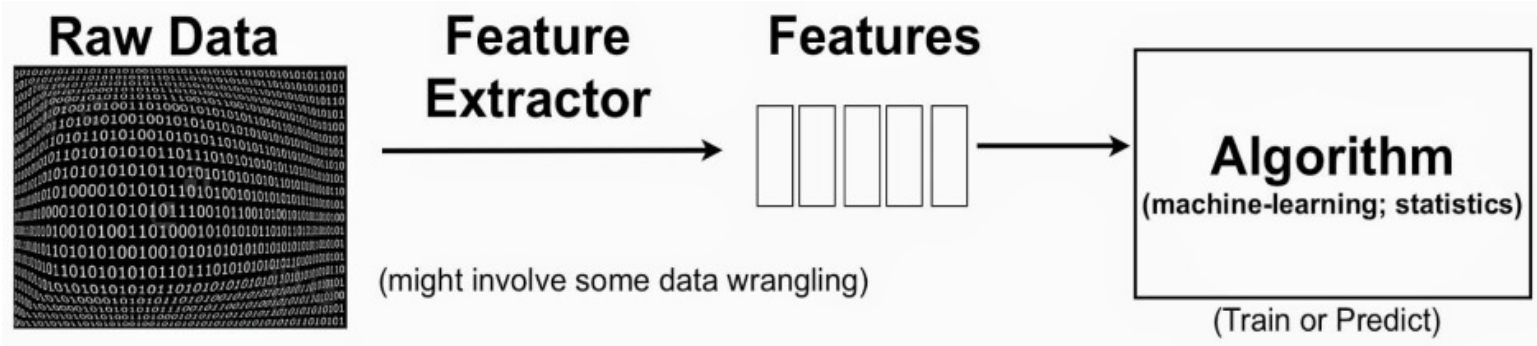
- Phương pháp thống kê (Chi tiết ở tuần 1)
- Tự động phát hiện ngoại lệ

Thiết kế đặc trưng

Giới thiệu

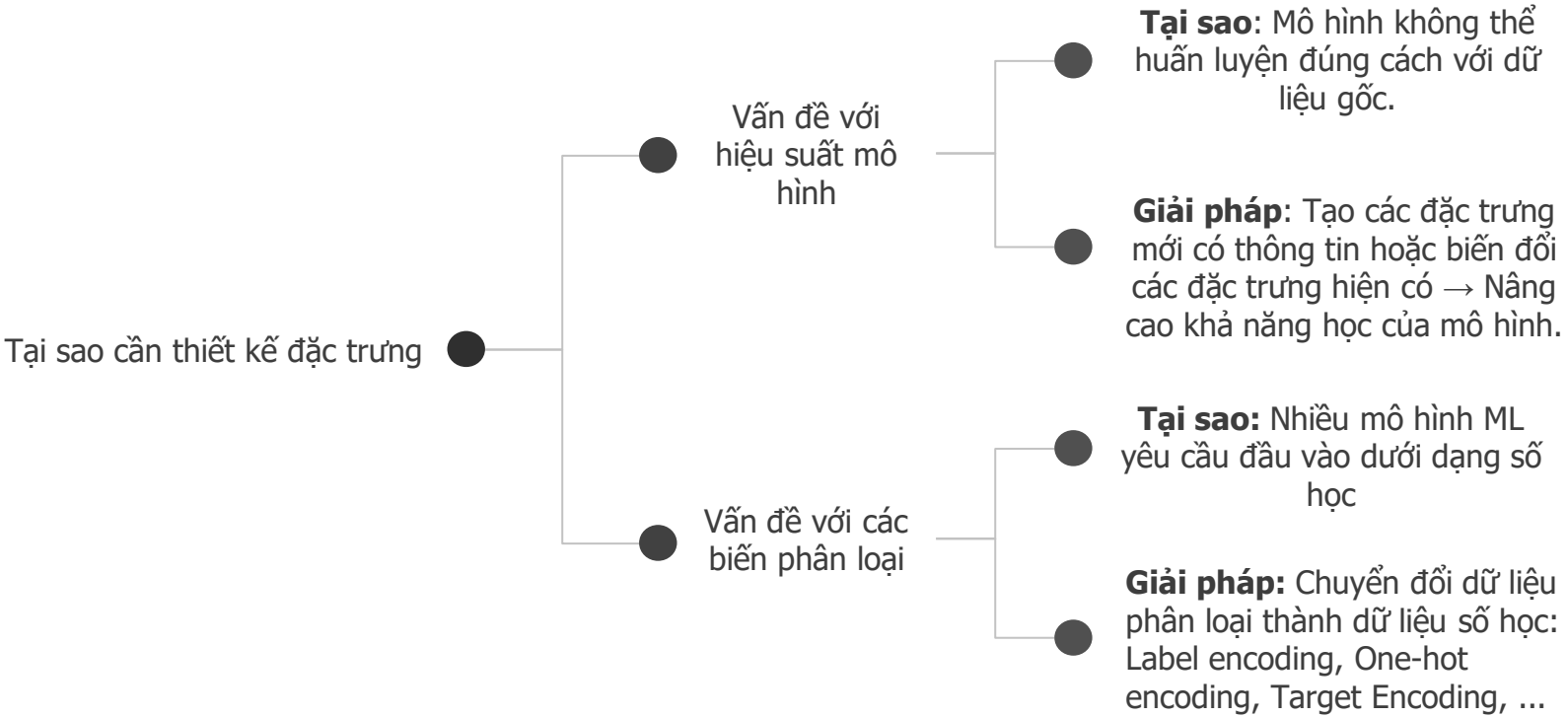
Thiết kế đặc trưng là gì

- **Thiết kế đặc trưng (Feature Engineering – Feature Extraction)** là quá trình sử dụng kiến thức về lĩnh vực để trích xuất các đặc trưng (đặc điểm, thuộc tính) từ dữ liệu gốc → cải thiện hiệu suất các mô hình máy học (ML models)

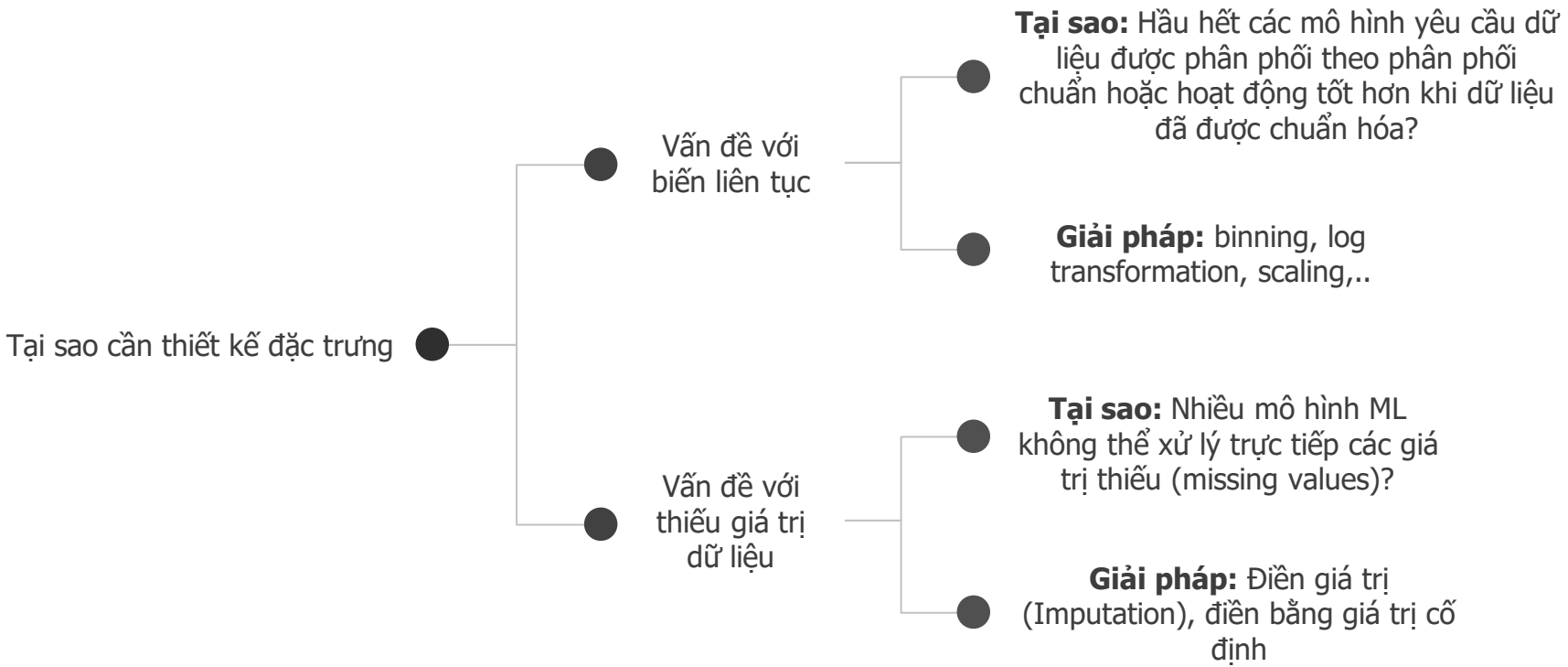


https://adataanalyst.com/machine-learning/comprehensive-guide-feature-engineering/

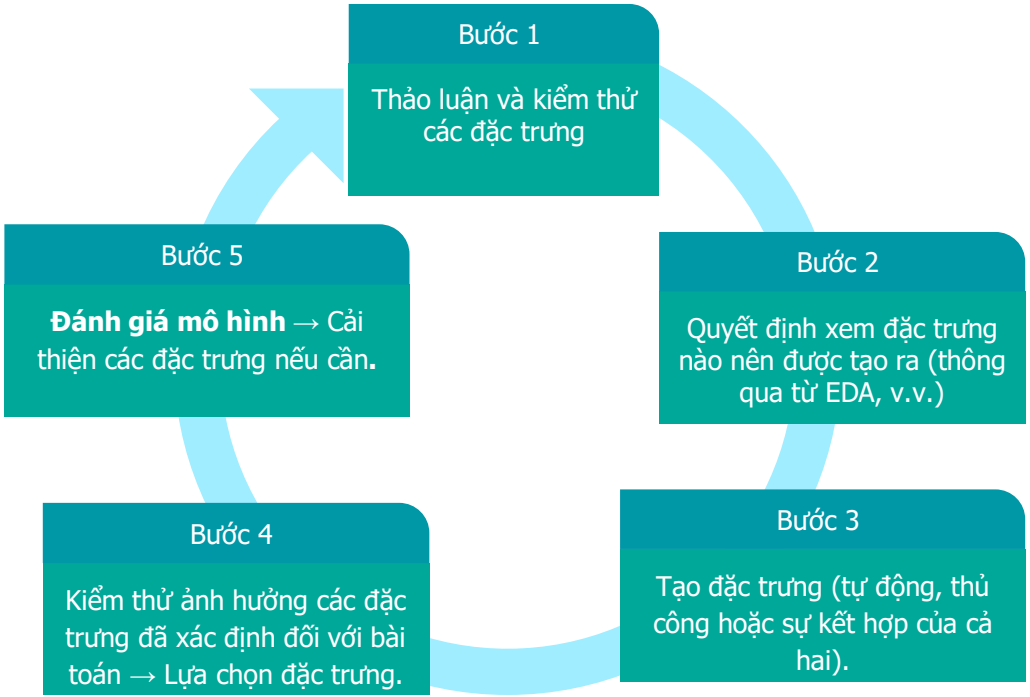
Tại sao cần thiết kế đặc trưng



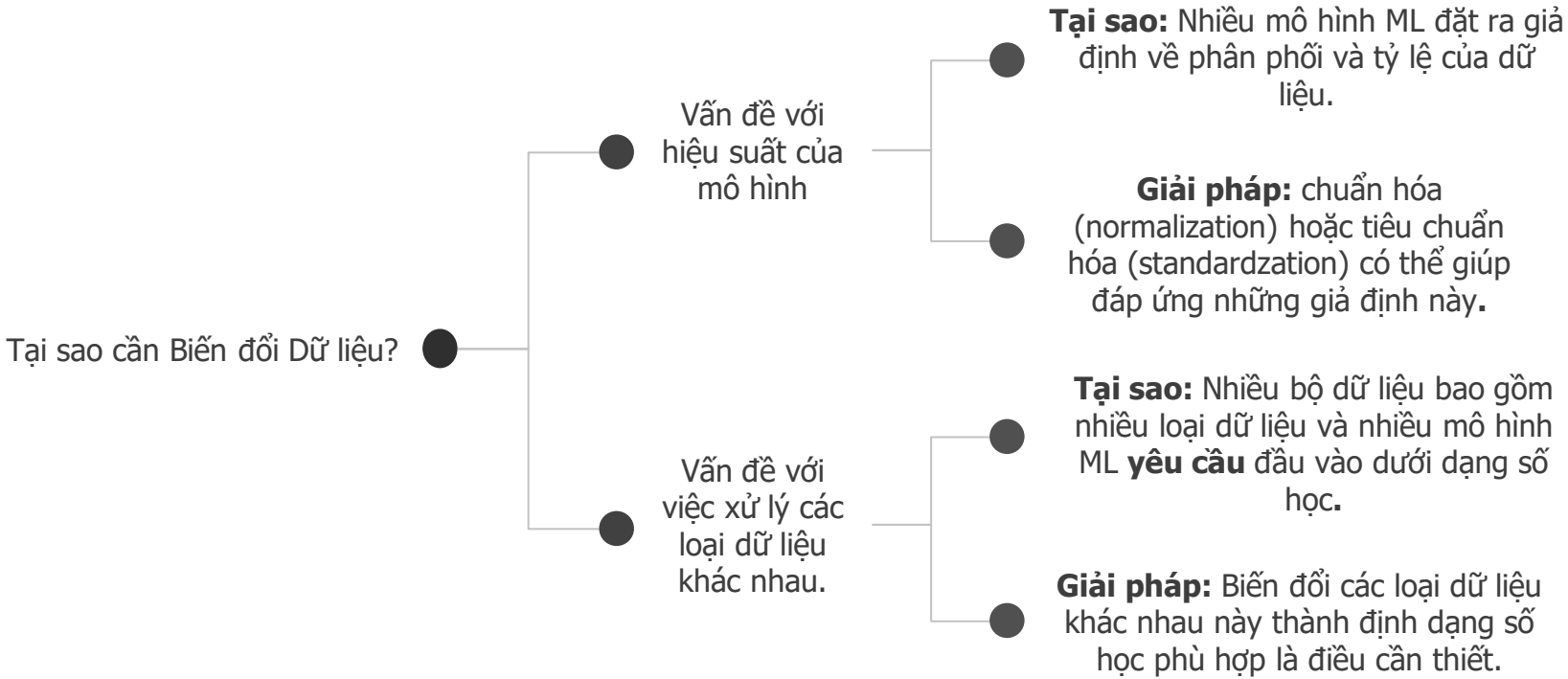
Tại sao cần thiết kế đặc trưng



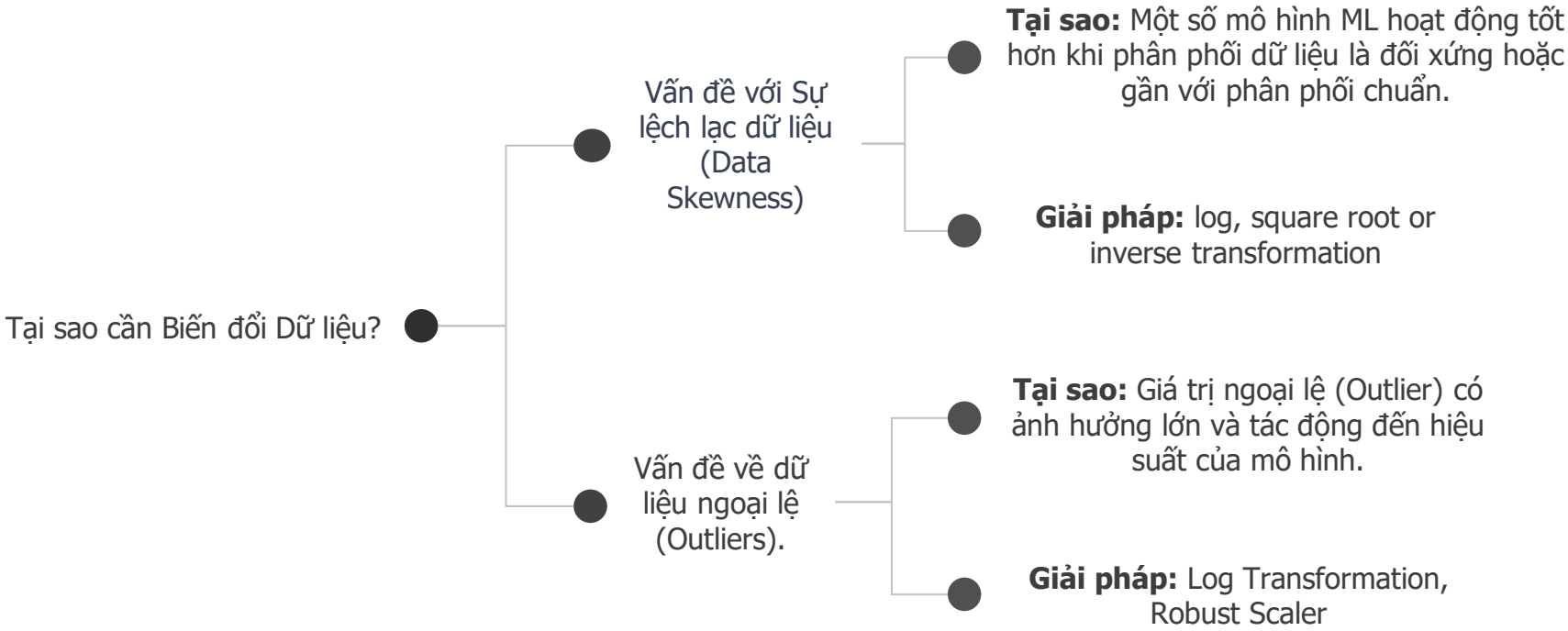
Quá trình lặp



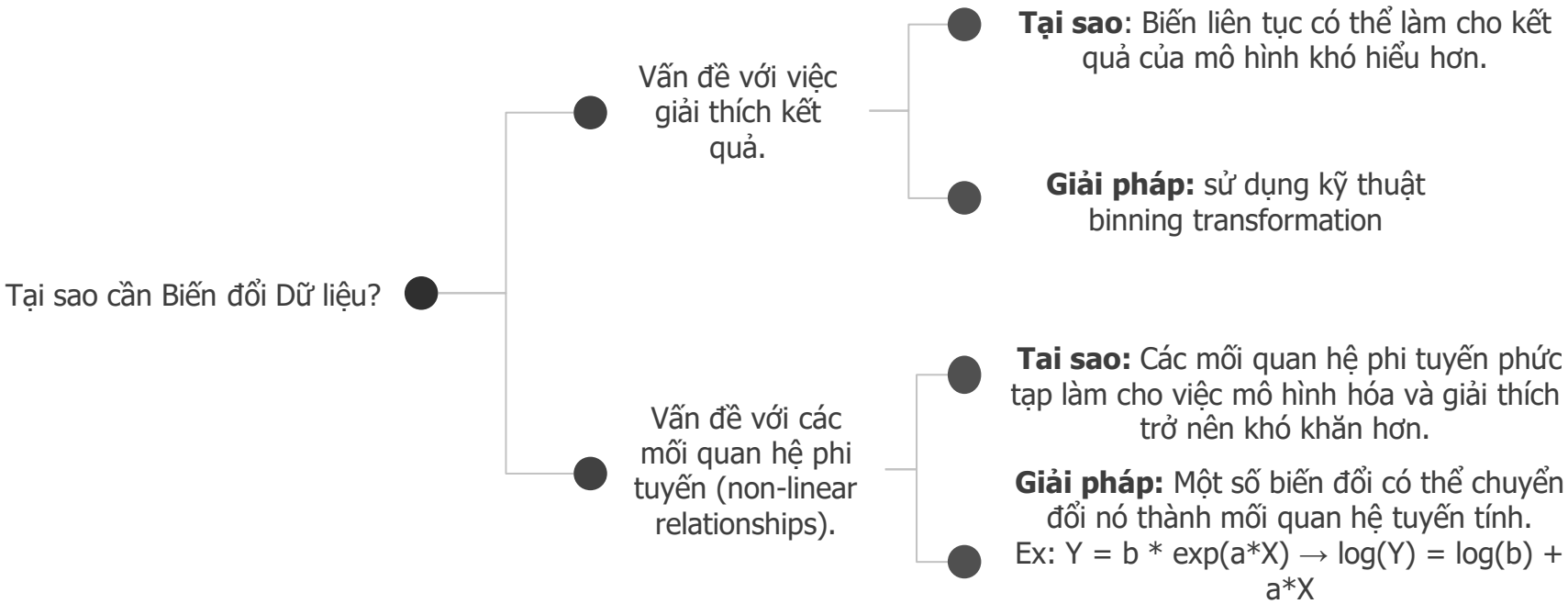
Tại sao cần Biến đổi Dữ liệu?



Tại sao cần Biến đổi Dữ liệu?

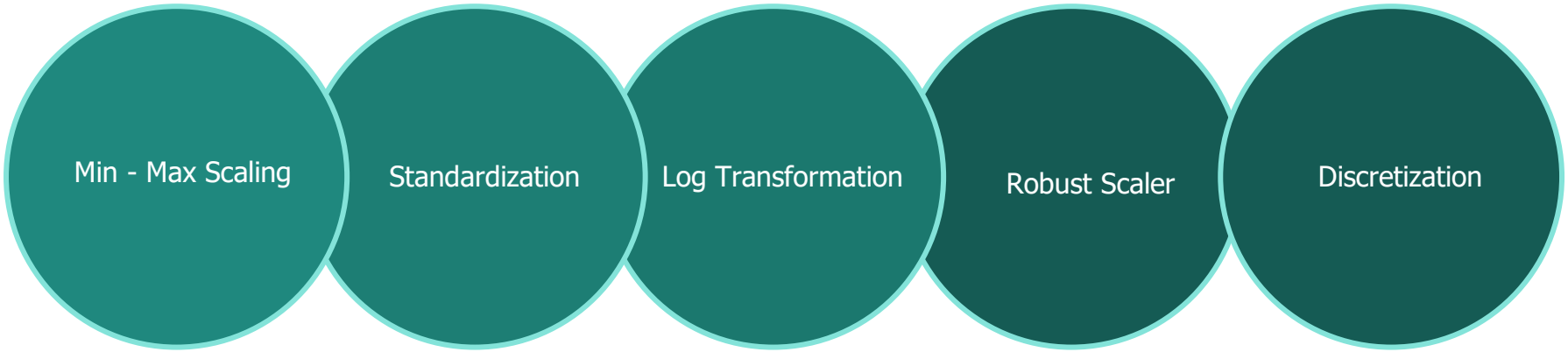


Tại sao cần Biến đổi Dữ liệu?



Biến đổi dữ liệu – Data Transformation

Dữ liệu dạng số



Kỹ thuật Min-Max Scaling

- Normalization (Min-Max Scaling): Chuẩn hóa (Normalization) tỷ lệ lại dữ liệu thành một phạm vi cố định, thường là phạm vi từ [0, 1].

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
0	0.0	0.0	0.0	0.0	0.0
1	109.0	9.0	25.0	549.0	44.0
2	43.0	3576.0	0.0	6715.0	49.0
3	0.0	1283.0	371.0	3329.0	193.0
4	303.0	70.0	151.0	565.0	2.0

Trước khi áp dụng Min-Max Scaling

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.007608	0.000302	0.001064	0.024500	0.001823
2	0.003001	0.119948	0.000000	0.299670	0.002030
3	0.000000	0.043035	0.015793	0.148563	0.007997
4	0.021149	0.002348	0.006428	0.025214	0.000083

Sau khi áp dụng Min-Max Scaling

Ví dụ

```
from sklearn.preprocessing import MinMaxScaler
col_names = ['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']


features = df[col_names]

scaler = MinMaxScaler().fit(features.values)
features = scaler.transform(features.values)
scaled_features = pd.DataFrame(features, columns = col_names)
scaled_features.head()
```

Yes

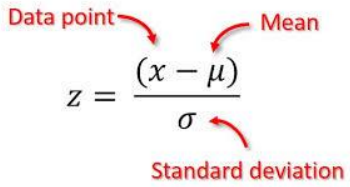
Sử dụng thư viện Scikit-lean

No

```
col_names = ['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']
scaled_features = df.copy()

for col in col_names:
    scaled_features[col] =  (scaled_features[col] - scaled_features[col].min())/(scaled_features[col].max() + scaled_features[col].min())


scaled_features[col_names].head()
```

Kỹ thuật Standardization (Z-score scaling)

- Điều này đặc biệt hữu ích cho các thuật toán mà giả định rằng các đặc trưng tuân theo phân phối Gaussian hoặc khi các đặc trưng có các đơn vị và tỷ lệ khác nhau.



	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
0	0.0	0.0	0.0	0.0	0.0
1	109.0	9.0	25.0	549.0	44.0
2	43.0	3576.0	0.0	6715.0	49.0
3	0.0	1283.0	371.0	3329.0	193.0
4	303.0	70.0	151.0	565.0	2.0

Trước khi áp dụng Standard Scaling

	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
0	-0.337025	-0.284274	-0.287317	-0.273736	-0.266098
1	-0.173528	-0.278689	-0.245971	0.209267	-0.227692
2	-0.272527	1.934922	-0.287317	5.634034	-0.223327
3	-0.337025	0.511931	0.326250	2.655075	-0.097634
4	0.117466	-0.240833	-0.037590	0.223344	-0.264352

Sau khi áp dụng Standard Scaling

Kỹ thuật Standardization (Z-score scaling)

```python
from sklearn.preprocessing import StandardScaler
col_names = ['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']


features = df[col_names]

scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
scaled_features = pd.DataFrame(features, columns = col_names)
scaled_features.head()
```

Yes

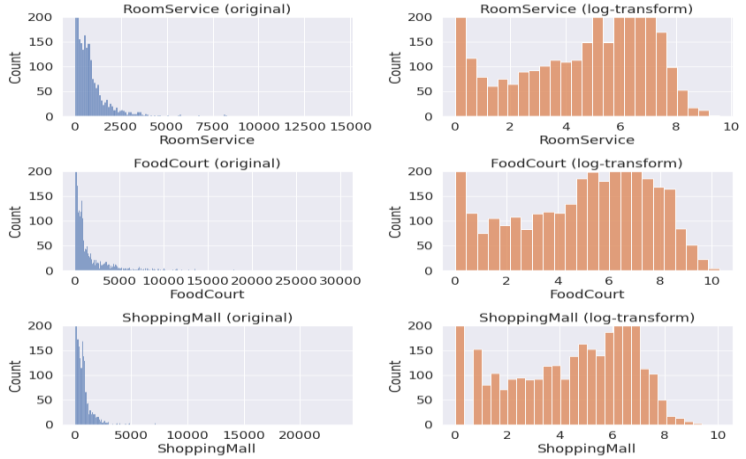
Sử dụng thư viện scikit-learn

No

```python
col_names = ['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']
scaled_features = df.copy()

for col in col_names:
    scaled_features[col] = (scaled_features[col] - scaled_features[col].mean())/(scaled_features[col].std())


scaled_features[col_names].head()
```

Kỹ thuật Log Transformation

- Biến đổi logarithm (Log transformation) được sử dụng để giảm tác động của giá trị ngoại lệ (outliers) và xử lý phân phối dữ liệu bị lệch lạc (handle skewed).
- Bằng cách áp dụng hàm logarithm cho dữ liệu, nó nén phạm vi của dữ liệu và làm cho phân phối trở nên đối xứng hơn.



Biểu đồ histogram trước (bên trái) và sau (bên phải) biến đổi logarithm.

Discretization (Binning)

- **Rời rạc hóa** (quantization hoặc binning) là một kỹ thuật chuyển đổi biến liên tục thành các danh mục rời rạc bằng cách chia phạm vi của biến thành các khoảng (bins) hoặc khoảng cách cố định.
- Việc này có thể giúp xử lý dữ liệu nhiễu, đơn giản hóa mô hình hoặc tiết lộ các mẫu trong dữ liệu mà không rõ ràng với biến liên tục.

	PassengerId	Age
0	0001_01	39.0
1	0002_01	24.0
2	0003_01	58.0
3	0003_02	33.0
4	0004_01	16.0

Liên tục

	PassengerId	Age_group
0	0001_01	Age_31-50
1	0002_01	Age_18-25
2	0003_01	Age_51+
3	0003_02	Age_31-50
4	0004_01	Age_13-17

Rời rạc

Ví dụ

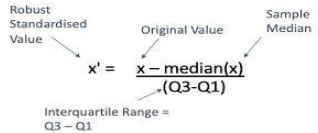
```
train['Age_group']=np.nan
train.loc[train['Age']<=12,'Age_group']='Age_0-12'
train.loc[(train['Age']>12) & (train['Age']<18),'Age_group']='Age_13-17'
train.loc[(train['Age']>=18) & (train['Age']<=25),'Age_group']='Age_18-25'
train.loc[(train['Age']>25) & (train['Age']<=30),'Age_group']='Age_26-30'
train.loc[(train['Age']>30) & (train['Age']<=50),'Age_group']='Age_31-50'
train.loc[train['Age']>50,'Age_group']='Age_51+'
```

```
# The first way
binned = pd.cut(train['Age'], bins = 6,labels = ['Age_0-12', 'Age_13-17', 'Age_18-25',
                                                 'Age_26-30', 'Age_31-50', 'Age_51+'])
# Second way
bins = pd.IntervalIndex.from_tuples([(0, 12), (12, 18), (18, 25), (25, 30), (30, 50), (50, 100)])
binned = pd.cut(train['Age'], bins = bins, labels = ['Age_0-12', 'Age_13-17', 'Age_18-25',
                                                 'Age_26-30', 'Age_31-50', 'Age_51+'])

pdcut_output = np.squeeze(np.array([list(binned)]).reshape(-1, 1))

train['Age_group'] =  pdcut_output
```

```
from sklearn.preprocessing import KBinsDiscretizer

data_age = np.array(train['Age'].fillna(0)).reshape(-1, 1)
age_group = KBinsDiscretizer(n_bins = 6, encode = "ordinal").fit_transform(data_age).squeeze()

train['Age_group'] =  age_group
```

Chuẩn hóa đặc trưng bằng Robust Scaler

- Robust scaling sử dụng giá trị trung vị và phạm vi giữa các phân vị (interquartile range) để tỷ lệ dữ liệu, làm cho nó ít nhạy cảm hơn đối với giá trị ngoại lệ so với Min-Max hoặc Z-score scaling.
- ***Điều này hữu ích khi bộ dữ liệu chứa giá trị ngoại lệ hoặc khi phân phối dữ liệu không đối xứng..***



	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
0	0.0	0.0	0.0	0.0	0.0
1	109.0	9.0	25.0	549.0	44.0
2	43.0	3576.0	0.0	6715.0	49.0
3	0.0	1283.0	371.0	3329.0	193.0
4	303.0	70.0	151.0	565.0	2.0

Trước Robust Scaler

	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	2.319149	0.118421	0.925926	9.305085	0.956522
2	0.914894	47.052632	0.000000	113.813559	1.065217
3	0.000000	16.881579	13.740741	56.423729	4.195652
4	6.446809	0.921053	5.592593	9.576271	0.043478

Sau Robust Scaler

Example

```sklearn
from sklearn.preprocessing import RobustScaler
col_names = ['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']


features = df[col_names]

scaler = RobustScaler().fit(features.values)
features = scaler.transform(features.values)
scaled_features = pd.DataFrame(features, columns = col_names)
scaled_features.head()
```

Sử dụng scikit-learn

```pandas
col_names = ['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']
scaled_features = df.copy()

for col in col_names:
    q1 = scaled_features[col].quantile(0.25)
    median = scaled_features[col].quantile(0.5)
    q3 = scaled_features[col].quantile(0.75)
    iqr = q3 - q1
    scaled_features[col] = (scaled_features[col] - median)/iqr

scaled_features[col_names].head()
```

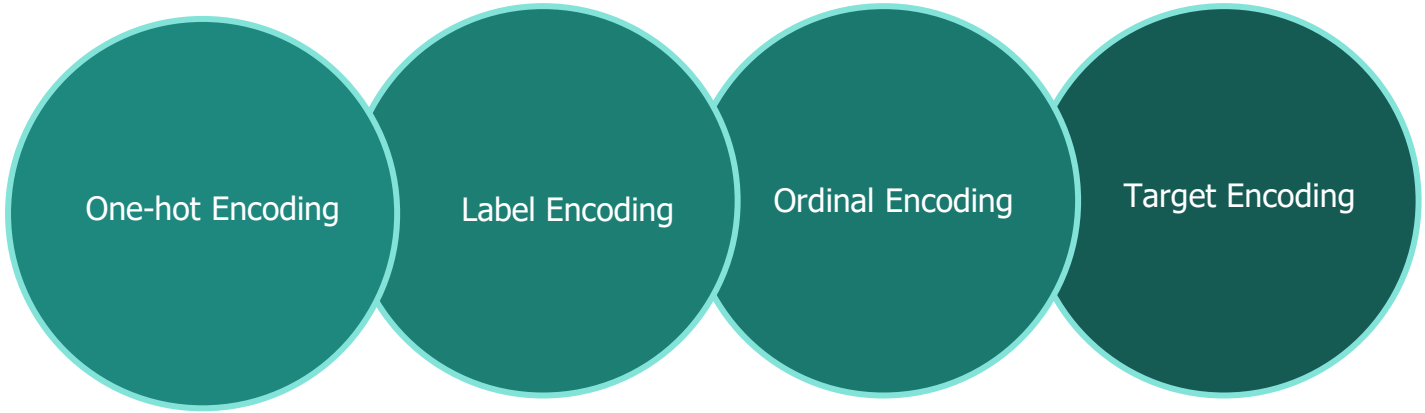
Sử dụng pandas

Kết luận

01	Min - Max Scaling	• x_norm = (x - x_min)/(x_max - x_min) • Khi đặc trưng phân bố gần đều trên một phạm vi cố định.
02	Standard Scaling	• x_norm = (x - mean(x))/std(x) • Phân phối của đặc trưng không chứa các giá trị ngoại lệ cực đoan, giả định rằng đặc trưng tuân theo phân phối Gaussian hoặc có các đơn vị và tỷ lệ khác nhau.
03	Log Transformation	• x_norm = log(x) or x_norm = log(1+x) • Giảm tác động của các giá trị ngoại lệ và xử lý các phân phối dữ liệu bị lệch.
04	Binning	• Chuyển đổi biến liên tục thành các danh mục rời rạc bằng cách chia phạm vi của biến thành các khoảng (bins). • Xử lý dữ liệu nhiễu, đơn giản hóa mô hình.
05	Robust Scaling	• x_norm = (x - median(x))/IQR • Làm cho nó ít nhạy cảm hơn đối với các giá trị ngoại lệ so với việc tỷ lệ Min-Max hoặc Z-score. • Hữu ích khi bộ dữ liệu chứa các giá trị ngoại lệ hoặc phân phối dữ liệu không đối xứng.

Chuyển hóa dữ liệu

Kiểu dữ liệu danh mục (Categorical data)



Kỹ thuật One-hot Encoding

- Đối với mỗi giá trị, sẽ được tạo một cột nhị phân mới, với giá trị 1 và 0 thể hiện sự có mặt hoặc vắng mặt của danh mục đó trong dữ liệu gốc.

	PassengerId	HomePlanet
0	0001_01	Europa
1	0002_01	Earth
2	0003_01	Europa
3	0003_02	Europa
4	0004_01	Earth

```
pd.get_dummies(df['HomePlanet'], prefix = 'is')
```

	is_Earth	is_Europa	is_Mars
0	0	1	0
1	1	0	0
2	0	1	0
3	0	1	0
4	1	0	0

Kỹ thuật Ordinal Encoding

- Mã hóa Ordinal (Ordinal encoding) gán các giá trị số nguyên cho biến phân loại dựa trên thứ tự hoặc xếp hạng của chúng..
- Nó phù hợp cho dữ liệu thứ tự (ordinal data), trong đó có sự thứ tự tự nhiên giữa các danh mục.

```
age_group_mapping = {'Age_0-12': 1, 'Age_13-17': 2, 'Age_18-25': 3,
                     'Age_26-30': 4, 'Age_31-50': 5, 'Age_51+': 6}

train['Age_group_encode'] = train['Age_group'].map(age_group_mapping)
train[['Age', 'Age_group', 'Age_group_encode']].head()
```

	Age	Age_group	Age_group_encode
0	39.0	Age_31-50	5.0
1	24.0	Age_18-25	3.0
2	58.0	Age_51+	6.0
3	33.0	Age_31-50	5.0
4	16.0	Age_13-17	2.0

Kỹ thuật Label Encoding

- Mã hóa nhãn (Label encoding) là một phương pháp khác để chuyển đổi biến phân loại thành giá trị số học bằng cách gán một số nguyên duy nhất cho mỗi loại.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(train['HomePlanet'].astype(str).fillna('Unknow').tolist() +
                  test['HomePlanet'].astype(str).fillna('Unknow').tolist())

train['HomePlanet_encode'] = le.transform(train['HomePlanet'])
test['HomePlanet_encode'] = le.transform(test['HomePlanet'])

train[['PassengerId', 'HomePlanet', 'HomePlanet_encode']].head()
```

	PassengerId	HomePlanet	HomePlanet_encode
0	0001_01	Europa	1
1	0002_01	Earth	0
2	0003_01	Europa	1
3	0003_02	Europa	1
4	0004_01	Earth	0

Use scikit-learn

```
train['HomePlanet_enc'],_ = train['HomePlanet'].factorize()
train[['PassengerId', 'HomePlanet', 'HomePlanet_enc']].head()
```

	PassengerId	HomePlanet	HomePlanet_enc
0	0001_01	Europa	0
1	0002_01	Earth	1
2	0003_01	Europa	0
3	0003_02	Europa	0
4	0004_01	Earth	1

Use pandas

Kỹ thuật Target Encoding

- Mã hóa mục tiêu (Target encoding) là loại mã hóa thay thế các nhãn của một đặc trưng bằng một số được tính từ mục tiêu (target).
- Phương pháp
  - Một phiên bản đơn giản và hiệu quả là áp dụng một phép tổng hợp theo nhóm như trung bình (mean). Xem example
  - Mã hóa mục tiêu với kỹ thuật làm mịn (smoothing). Xem example
  - Mã hóa mục tiêu Bayesian phân cấp (Hierarchical Bayesian Target Encoding). Xem example

Kỹ thuật Simple Target Encoding (Mean encoding)

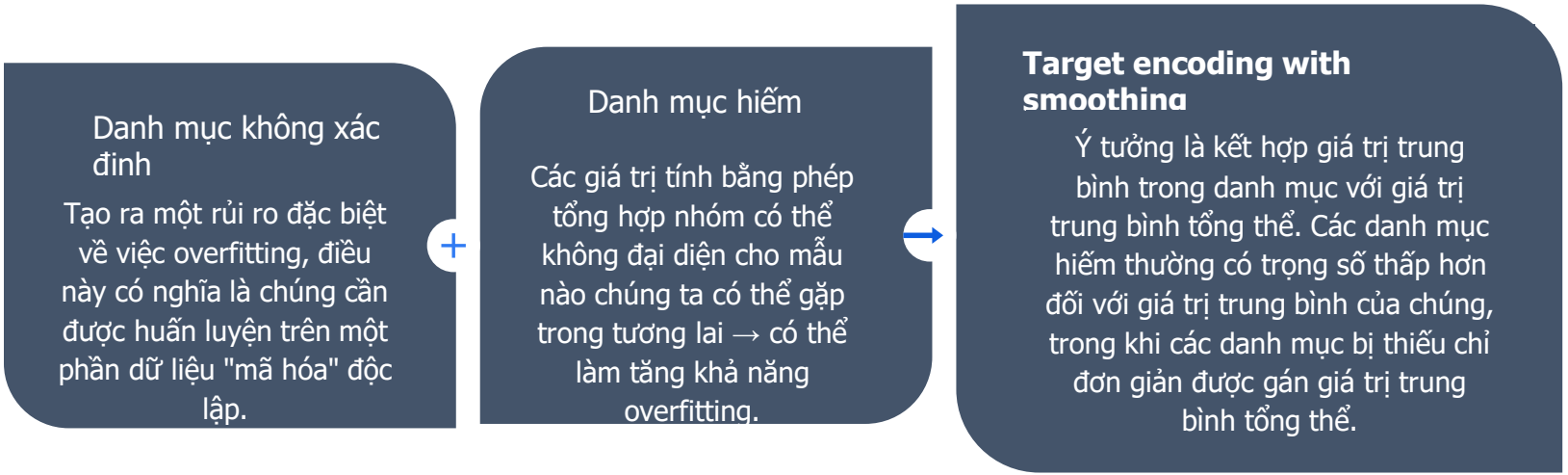
- Khi áp dụng cho mục tiêu nhị phân, phương pháp này còn được gọi là "bin counting”.
- Các tên gọi khác mà bạn có thể gặp bao gồm: mã hóa xác suất (likelihood encoding), mã hóa tác động (impact encoding) và mã hóa "leave-one-out".

```
train["HomePlanet_target_en"] = train.groupby("HomePlanet")["Transported"].transform("mean")

train[["HomePlanet", "Transported", "HomePlanet_target_en"]].head(10)
```

	HomePlanet	Transported	HomePlanet_target_en
0	Europa	False	0.658846
1	Earth	True	0.423946
2	Europa	False	0.658846
3	Europa	False	0.658846
4	Earth	True	0.423946
5	Earth	True	0.423946

Target encoding with smoothing

Tại sao sử dụng mã hóa mục tiêu với kỹ thuật làm mịn?



Mã giả - Pseudo code

In pseudocode:

```
encoding = weight * in_category + (1 - weight) * overall
```

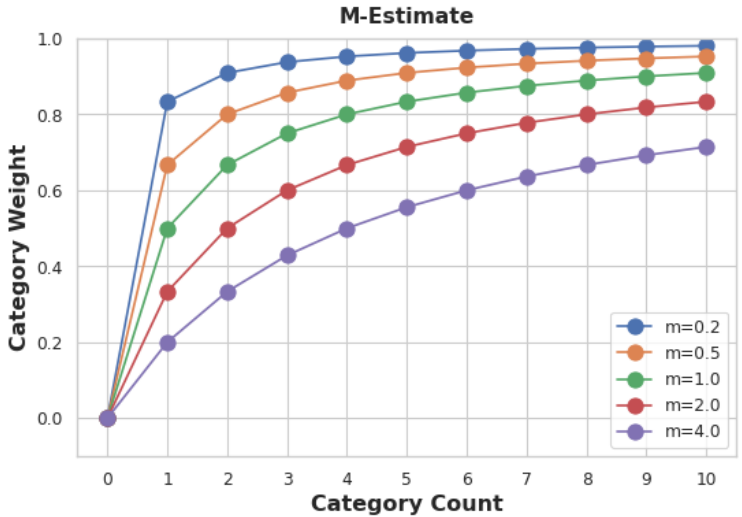
where `weight` is a value between 0 and 1 calculated from the category frequency.

An easy way to determine the value for `weight` is to compute an **m-estimate**:

```
weight = n / (n + m)
```

where `n` is the total number of times that category occurs in the data. The parameter `m` determines the "smoothing factor". Larger values of `m` put more weight on the overall estimate.

https://www.kaggle.com/code/ryanholbrook/target-encoding



https://www.kaggle.com/code/ryanholbrook/target-encoding

Kỹ thuật tạo đặc trưng

Phần 1

Kỹ thuật tạo đặc trưng

- Các phép biến đổi toán học
- Đếm số lượng & Mã hóa tần số
- Tổng hợp và phân rã đặc trưng
- Nhóm tổng hợp
- Phương pháp khác
  - Đặc trưng cụm
  - Đặc trưng PCA
- Đặc trưng văn bản
- Chuỗi thời gian

Tips khám phá các đặc trưng mới

Hiểu đặc trưng

Tham khảo tài liệu dữ liệu của bạn, nếu có

Hiểu đặc trưng

Nghiên cứu để thu được các kiến thức trong lĩnh vực

Nghiên cứu công trình trước đó

Các bài viết giải pháp từ các cuộc thi Kaggle trước đây là nguồn tài nguyên tuyệt vời

Sử dụng trực quan hóa dữ liệu

Trực quan hóa có thể tiết lộ các vấn đề trong phân bố của một đặc trưng hoặc các mối quan hệ phức tạp có thể được đơn giản hóa

Các phép biến đổi toán học

- Mối quan hệ giữa các đặc trưng số học thường được thể hiện thông qua các công thức toán học.
- Cộng, trừ, nhân, v.v.

```
autos["stroke_ratio"] = autos.stroke / autos.bore
```

```
autos["displacement"] = (
    np.pi * ((0.5 * autos.bore) ** 2) * autos.stroke * autos.num_
of_cylinders
)
```

```
# If the feature has 0.0 values, use np.log1p (log(1+x)) instead of
np.log
accidents["LogWindSpeed"] = accidents.WindSpeed.apply(np.log1p)
```

Đếm số lượng/Mã hóa tần số

- Mã hóa tần số là một kỹ thuật mạnh mẽ cho phép mô hình (ví dụ: LightGBM) xem liệu các giá trị cột là hiếm hay phổ biến.

```
temp = df['card1'].value_counts().to_dict()
df['card1_counts'] = df['card1'].map(temp)
```

```
def encode_FE(df1, df2, cols):
    for col in cols:
        df = pd.concat([df1[col],df2[col]])
        vc = df.value_counts(dropna=True, normalize=True).to_dict
()
        vc[-1] = -1
        nm = col+'_FE'
        df1[nm] = df1[col].map(vc)
        df1[nm] = df1[nm].astype('float32')
        df2[nm] = df2[col].map(vc)
        df2[nm] = df2[nm].astype('float32')
        print(nm,', ',end='')
```

Đếm số lượng với các đặc trưng Boolean

- Các đặc điểm mô tả sự hiện diện hay vắng mặt của một thứ gì đó thường xuất hiện theo bộ, tập hợp các yếu tố nguy cơ của một căn bệnh. Bạn có thể tổng hợp các tính năng như vậy bằng cách đếm số lượng.

```
roadway_features = ["Amenity", "Bump", "Crossing", "GiveWay",
    "Junction", "NoExit", "Railway", "Roundabout", "Station", "St
op",
    "TrafficCalming", "TrafficSignal"]
accidents["RoadwayFeatures"] = accidents[roadway_features].sum(ax
is=1)

accidents[roadway_features + ["RoadwayFeatures"]].head(10)
```

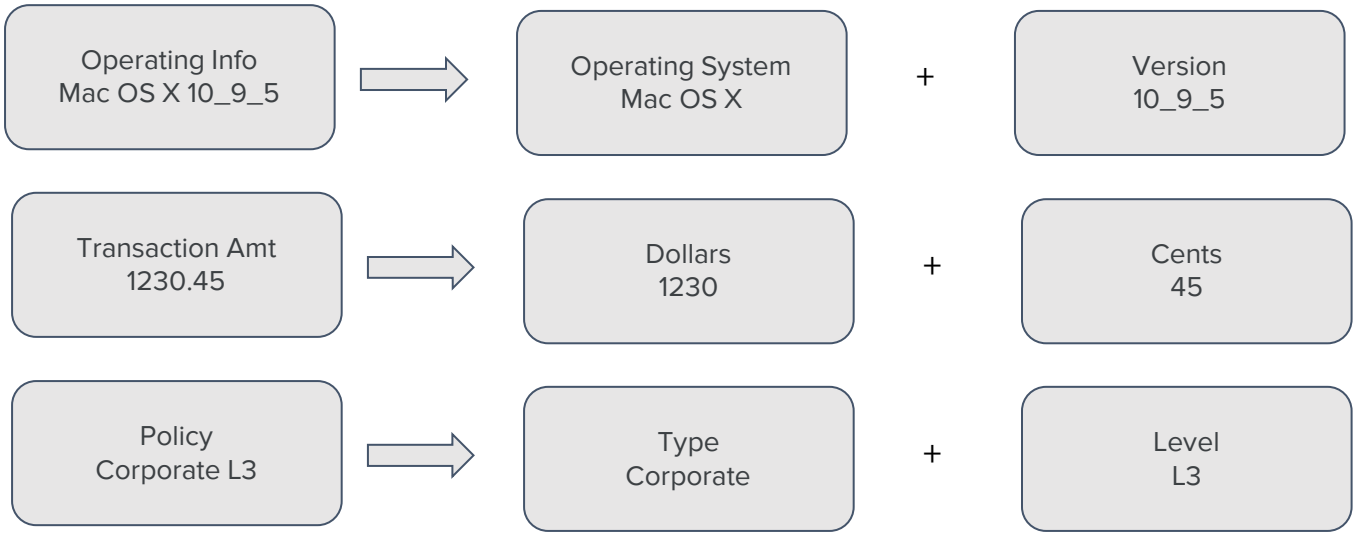
Railway	Roundabout	Station	Stop	TrafficCalming	TrafficSignal	RoadwayFeatures
False	False	False	False	False	False	0
False	False	False	False	False	False	0
False	False	False	False	False	False	0
False	False	False	False	False	False	0
False	False	False	False	False	False	0
False	False	False	False	False	False	1

```
components = [ "Cement", "BlastFurnaceSlag", "FlyAsh", "Water",
               "Superplasticizer", "CoarseAggregate", "FineAggreg
ate"]
concrete["Components"] = concrete[components].gt(0).sum(axis=1)

concrete[components + ["Components"]].head(10)
```

ceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Components
	0.0	162.0	2.5	1040.0	676.0	5
	0.0	162.0	2.5	1055.0	676.0	5
	0.0	228.0	0.0	932.0	594.0	5
	0.0	228.0	0.0	932.0	594.0	5
	0.0	192.0	0.0	978.4	825.5	5
	0.0	228.0	0.0	932.0	670.0	5

Phân rã đặc trưng

- Thường thì bạn sẽ có những chuỗi phức tạp có thể được chia thành các phần đơn giản hơn một cách hữu ích.
- PP dựa trên cây (LGBM/XGBoost) không thể tự thấy các phần này, vì vậy cần tách chúng ra.



Tổng hợp đặc trưng

- Bạn cũng có thể kết hợp các đặc trưng đơn giản thành một đặc trưng tổng hợp nếu bạn có lý do để tin rằng có sự tương tác nào đó trong sự kết hợp đó.

```
autos["make_and_style"] = autos["make"] + "_" + autos["body_styl
e"]
autos[["make", "body_style", "make_and_style"]].head()
```

	make	body_style	make_and_style
0	alfa-romero	convertible	alfa-romero_convertible
1	alfa-romero	convertible	alfa-romero_convertible
2	alfa-romero	hatchback	alfa-romero_hatchback
3	audi	sedan	audi_sedan
4	audi	sedan	audi_sedan

Tổng hợp đặc trưng (Code)

```
# LABEL ENCODE
def encode_LE(col,train=X_train,test=X_test,verbose=True):
    df_comb = pd.concat([train[col],test[col]],axis=0)
    df_comb,_ = df_comb.factorize(sort=True)
    nm = col
    if df_comb.max()>32000:
        train[nm] = df_comb[:len(train)].astype('int32')
        test[nm] = df_comb[len(train):].astype('int32')
    else:
        train[nm] = df_comb[:len(train)].astype('int16')
        test[nm] = df_comb[len(train):].astype('int16')
```

```
# COMBINE FEATURES
def encode_CB(col1,col2,df1=X_train,df2=X_test):
    nm = col1+'_'+col2
    df1[nm] = df1[col1].astype(str)+'_'+df1[col2].astype(str)
    df2[nm] = df2[col1].astype(str)+'_'+df2[col2].astype(str)
    encode_LE(nm,verbose=False)
```

Nhóm tổng hợp

- Thông tin **tổng hợp** trên nhiều hàng được nhóm theo một số danh mục
- Việc cung cấp LGBM với số liệu **thống kê nhóm** cho phép mô hình (ví dụ: dựa trên cây) xác định xem một giá trị là **phổ biến hay hiếm** đối với một nhóm cụ thể.
- Có thể tính toán số liệu thống kê nhóm bằng cách cung cấp cho pandas: **nhóm, biến** quan tâm và **loại thống kê**.

```
temp = df.groupby('card1')['TransactionAmt'].agg(['mean'])
    .rename({'mean':'TransactionAmt_card1_mean'},axis=1)
df = pd.merge(df,temp,on='card1',how='left')
```

Nhóm tổng hợp (Ví dụ)

```
customer["AverageIncome"] = (
    customer.groupby("State")  # for each state
    ["Income"]                 # select the income
    .transform("mean")         # and compute its mean
)

customer[["State", "Income", "AverageIncome"]].head(10)
```

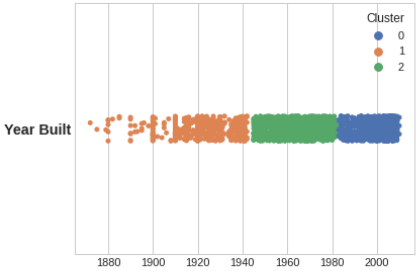
	State	Income	AverageIncome
0	Washington	56274	38122.733083
1	Arizona	0	37405.402231
2	Nevada	48767	38369.605442
3	California	0	37558.946667
4	Washington	43836	38122.733083
5	Oregon	62902	37557.283353

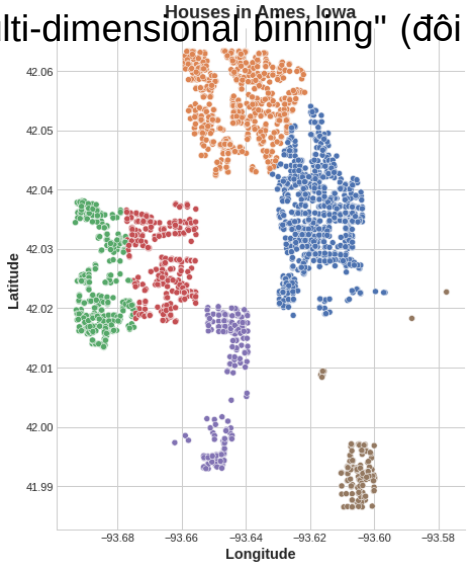
Nhóm tổng hợp: Mã giả

```
def encode_AG(main_columns, uids, aggregations=['mean'], train_df=X_train, test_df=X_test,
              fillna=True, usena=False):
    # AGGREGATION OF MAIN WITH UID FOR GIVEN STATISTICS
    for main_column in main_columns:
        for col in uids:
            for agg_type in aggregations:
                new_col_name = main_column+'_'+col+'_'+agg_type
                temp_df = pd.concat([train_df[[col, main_column]], test_df[[col,main_column]]])
                if usena: temp_df.loc[temp_df[main_column]==-1,main_column] = np.nan
                temp_df = temp_df.groupby([col])[main_column].agg([agg_type]).reset_index().rename(
                                                        columns={agg_type: new_col_name})

                temp_df.index = list(temp_df[col])
                temp_df = temp_df[new_col_name].to_dict()

                train_df[new_col_name] = train_df[col].map(temp_df).astype('float32')
                test_df[new_col_name]  = test_df[col].map(temp_df).astype('float32')

                if fillna:
                    train_df[new_col_name].fillna(-1,inplace=True)
                    test_df[new_col_name].fillna(-1,inplace=True)
```

Đặc trưng theo cụm

- Khi áp dụng cho đặc trưng có một giá trị thực duy nhất, việc phân cụm hoạt động giống như một phép biến đổi “binning" truyền thống.
- Trên nhiều đặc trưng, nó giống như "multi-dimensional binning" (đôi khi được gọi là lượng tử hóa vectơ).





https://www.kaggle.com/code/ryanholbrook/clustering-with-k-means

Đặc trưng theo cụm (2)

- Đặc trưng theo cụm là loại đặc trưng phân loại
- Chiến thuật “chia để trị”:
  - Chia nhỏ các mối quan hệ phức tạp giữa các đặc trưng thành các phần đơn giản hơn.
  - Sau đó, mô hình chỉ cần học từng phần đơn giản hơn

```
# Create cluster feature
kmeans = KMeans(n_clusters=6)
X["Cluster"] = kmeans.fit_predict(X)
X["Cluster"] = X["Cluster"].astype("category")

X.head()
```

	MedInc	Latitude	Longitude	Cluster
0	8.3252	37.88	-122.23	5
1	8.3014	37.86	-122.22	5
2	7.2574	37.85	-122.24	5
3	5.6431	37.85	-122.25	5
4	3.8462	37.85	-122.25	1

Đặc trưng PCA

- Các thành phần thể hiện trực tiếp cấu trúc biến đổi của dữ liệu, chúng thường có thể mang lại nhiều thông tin hơn các đặc trưng ban đầu.

PCA chỉ hoạt động với các đặc trưng số, như số lượng hoặc số đếm liên tục

PCA rất nhạy cảm với quy mô (ví dụ: chuẩn hóa dữ liệu trước khi áp dụng PCA)

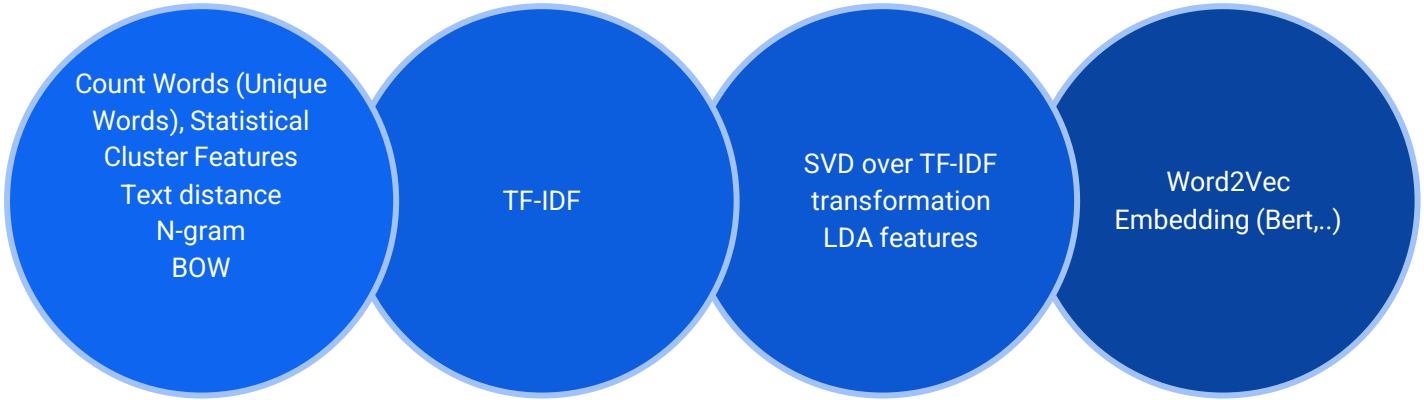
Hãy cân nhắc việc loại bỏ hoặc hạn chế các giá trị ngoại lệ vì chúng có thể có ảnh hưởng không đáng có đến kết quả.

Đặc trưng PCA: Ví dụ

- Sử dụng PCA với thư viện scikit-learn

```
features_c = list([x for x in train_features_df.columns if x.star
tswith("c-")])

pca = PCA(n_components=n_comp_CELLS, random_state=42)
pca_feat = [f'pca_C-{i}' for i in range(n_comp_CELLS)]

train_features_df[pca_feat] = pca.fit_transform(train_features_df
[features_c])
```

Đặc trưng văn bản



Đặc trưng văn bản (ví dụ)

- Đếm từ (Từ duy nhất), tỷ lệ tiêu đề/mô tả,..

```
# Meta Text Features
textfeats = ["description", "title"]
df['desc_punc'] = df['description'].apply(lambda x: len([c for c in str(x) if c in string.punctuation]))

df['title'] = df['title'].apply(lambda x: cleanName(x))
df["description"]   = df["description"].apply(lambda x: cleanName(x))

for cols in textfeats:
    df[cols] = df[cols].astype(str)
    df[cols] = df[cols].astype(str).fillna('missing') # FILL NA
    df[cols] = df[cols].str.lower() # Lowercase all text, so that capitalized words dont get treated differently
    df[cols + '_num_words'] = df[cols].apply(lambda comment: len(comment.split())) # Count number of Words
    df[cols + '_num_unique_words'] = df[cols].apply(lambda comment: len(set(w for w in comment.split())))
    df[cols + '_words_vs_unique'] = df[cols+'_num_unique_words'] / df[cols+'_num_words'] * 100 # Count Unique Words
    df[cols + '_num_letters'] = df[cols].apply(lambda comment: len(comment)) # Count number of Letters

# Extra Feature Engineering
df['title_desc_len_ratio'] = df['title_num_letters']/df['description_num_letters']
```

https://www.kaggle.com/code/shubhinderrana/pyt-user-id-features-vgg16-cat2vec-sentiment

Đặc trưng văn bản: Tf-idf

```
russian_stop = set(stopwords.words('russian'))

tfidf_para = {
    "stop_words": russian_stop,
    "analyzer": 'word',
    "token_pattern": r'\w{1,}',
    "sublinear_tf": True,
    "dtype": np.float32,
    "norm": 'l2',
    #"min_df":5,
    #"max_df":.9,
    "smooth_idf":False
}
```

tf-idf params

```
def get_col(col_name): return lambda x: x[col_name]
##I added to the max_features of the description. It did not change my
vectorizer = FeatureUnion([
        ('description',TfidfVectorizer(
            ngram_range=(1, 2),
            max_features=17000,
            **tfidf_para,
            preprocessor=get_col('description'))),
        ('title',CountVectorizer(
            ngram_range=(1, 2),
            stop_words = russian_stop,
            max_features=130000,
            preprocessor=get_col('title')))
    ])

start_vect=time.time()

#Fit my vectorizer on the entire dataset instead of the training rows
#Score improved by .0001
vectorizer.fit(df.to_dict('records'))

ready_df = vectorizer.transform(df.to_dict('records'))
tfvocab = vectorizer.get_feature_names()
```

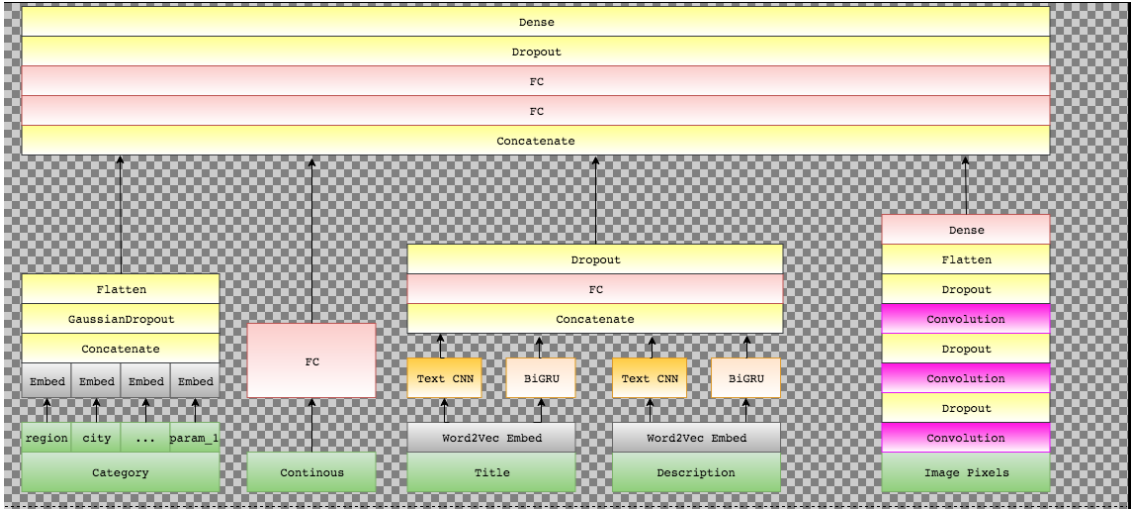
tf-idf fit and transform

Đặc trưng văn bản: Tf-idf + SVD

```
# Always start with these features. They work (almost) everytime!
tfv = TfidfVectorizer(min_df=3,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 3), use_idf=1,smooth_idf=1,sublinear_tf=1,
            stop_words = 'english')

# Fitting TF-IDF to both training and test sets (semi-supervised learning)
tfv.fit(list(xtrain) + list(xvalid))
xtrain_tfv =  tfv.transform(xtrain)
xvalid_tfv = tfv.transform(xvalid)
```

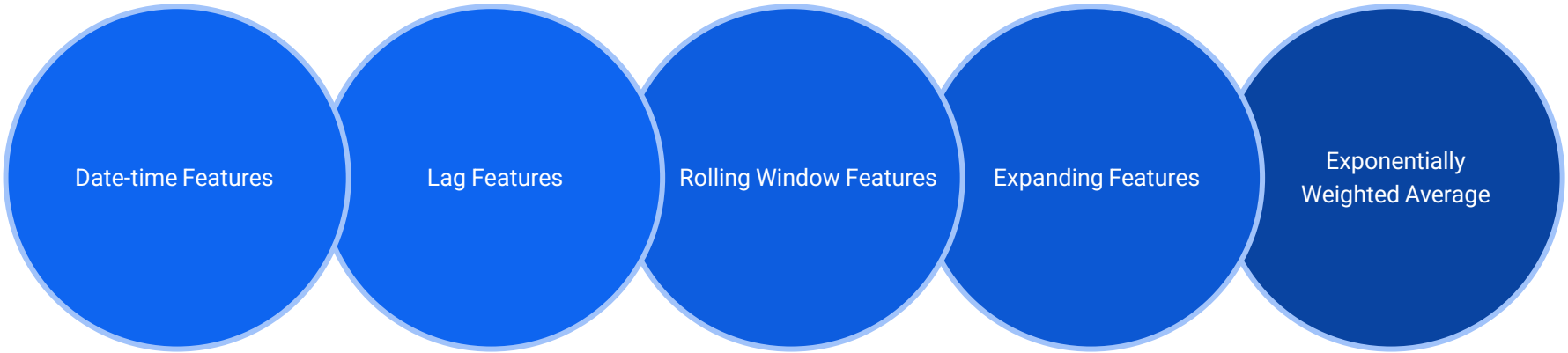
```
# Apply SVD, I chose 120 components. 120-200 components are good enough for SVM model
svd = decomposition.TruncatedSVD(n_components=120)
svd.fit(xtrain_tfv)
xtrain_svd = svd.transform(xtrain_tfv)
xvalid_svd = svd.transform(xvalid_tfv)
```

Đặc trưng văn bản: Word2Vec



https://www.kaggle.com/competitions/avito-demand-prediction/discussion/59917

Kỹ thuật tạo đặc trưng cho chuỗi thời gian



Đặc trưng Date-time

- Các tính năng này dựa trên ngày tháng và hữu ích cho việc tìm kiếm xu hướng cũng như tính thời vụ trong chuỗi thời gian.
  - năm tháng ngày
  - ngày trong tuần, tuần trong năm, mùa (quý trong năm)
  - là cuối tuần, ngày lễ

```
def get_time_feature(df, col, keep=False):
    df_copy = df.copy()
    prefix = col + "_"
    df_copy[col] = pd.to_datetime(df_copy[col])
    df_copy[prefix + 'year'] = df_copy[col].dt.year
    df_copy[prefix + 'month'] = df_copy[col].dt.month
    df_copy[prefix + 'day'] = df_copy[col].dt.day
    df_copy[prefix + 'hour'] = df_copy[col].dt.hour
    df_copy[prefix + 'weekofyear'] = df_copy[col].dt.weekofyear
    df_copy[prefix + 'dayofweek'] = df_copy[col].dt.dayofweek
    df_copy[prefix + 'is_wknd'] = df_copy[col].dt.dayofweek // 4
    df_copy[prefix + 'quarter'] = df_copy[col].dt.quarter
    df_copy[prefix + 'is_month_start'] = df_copy[col].dt.is_month_start.astype(int)
    df_copy[prefix + 'is_month_end'] = df_copy[col].dt.is_month_end.astype(int)
    if keep: return df_copy
    else: return df_copy.drop([col], axis=1)

df = get_time_feature(df, "time_col")
```

Đặc trưng độ trễ (Lag)

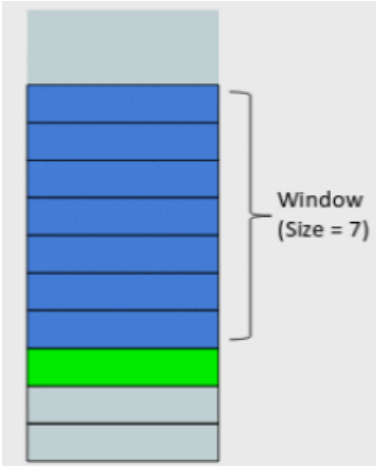
- Điều này dựa trên ý tưởng rằng các giá trị trong quá khứ có thể có ảnh hưởng đến các giá trị trong tương lai
- Các tính năng này dựa trên việc dịch chuyển các giá trị của chuỗi thời gian theo một số đơn vị thời gian nhất định.

```
keys = ...
val = ...
lag = 1
df.groupby(keys)[val].transform(lambda x: x.shift(lag))
```

Đặc trưng Rolling Window

- Các số liệu thống kê như giá trị trung bình, trung vị, độ lệch chuẩn, phương sai, tối thiểu, tối đa, v.v., được tính toán trên một cửa sổ cuộn của dữ liệu.
- Hữu ích cho việc tìm kiếm xu hướng và tính thời vụ trong một chuỗi thời gian
- Ví dụ: bạn có thể tạo một đặc trưng mang lại giá trị trung bình trong tuần qua

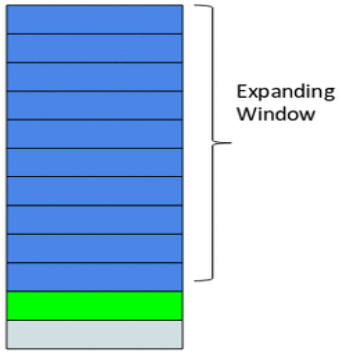
```
keys = ...
val = ...
window = 7
df.groupby(keys)[val].transform(lambda x: x.rolling(window=window, min_periods=3, win_type="triang").mean())
df.groupby(keys)[val].transform(lambda x: x.rolling(window=window, min_periods=3).std())
```



Đặc trưng Expanding Window

- Phiên bản nâng cao của kỹ thuật Rolling Window, nhưng cửa sổ bao gồm tất cả dữ liệu trước đó
- Chỉ xem xét các giá trị gần đây nhất và bỏ qua các giá trị trong quá khứ

```
keys = ...
val = ...
df.groupby(keys)[val].transform(lambda x: x.expanding(2).mean())
df.groupby(keys)[val].transform(lambda x: x.expanding(2).std())
```



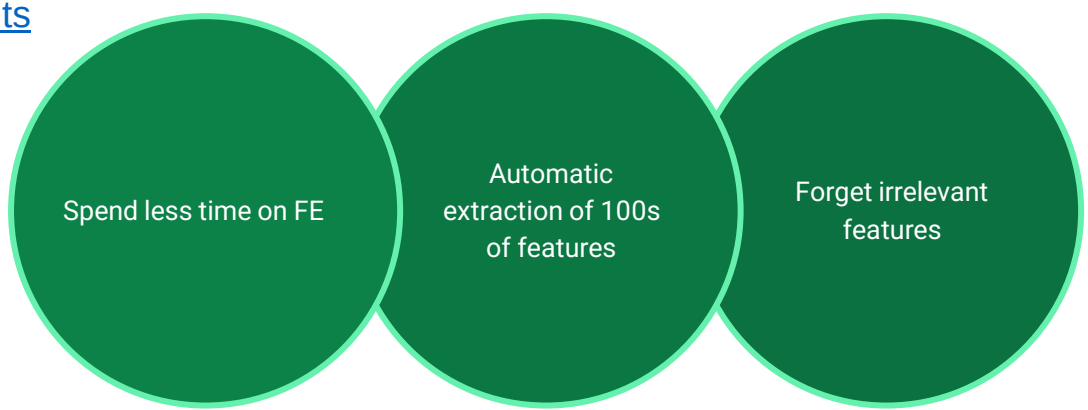
Trung bình theo cấp số nhân

- Tương tự như Đặc trưng Rolling Window, nhưng trọng số giảm khi các điểm dữ liệu cũ hơn

```
keys = ...
val = ...
lag = 1
alpha=0.95
df_temp.groupby(keys)[val].transform(lambda x: x.shift(lag).ewm(alpha=alpha).mean())
```

tsfresh: Tạo đặc trưng tự động cho chuỗi thời gian

- Tự động tính toán một số lượng lớn các đặc trưng chuỗi thời gian
- Github
- Documents



Tips tạo đặc trưng

Linear Model

Mô hình tuyến tính học tổng quát, phân biệt một cách tự nhiên nhưng không thể học được gì phức tạp hơn

(Linear + NN) Model

Nói chung hoạt động tốt hơn với các đặc trưng được chuẩn hóa (dựa trên cây ít bị ảnh hưởng hơn). NN đặc biệt cần các đặc trưng được chia tỷ lệ không quá xa 0.

Tree-based Model

Các mô hình dựa trên cây có thể học cách ước chừng hầu hết mọi sự kết hợp các tính năng, nhưng khi sự kết hợp đó đặc biệt quan trọng thì chúng vẫn có thể hưởng lợi từ việc tạo nó một cách rõ ràng, đặc biệt là khi dữ liệu bị hạn chế.

Counts features

Số lượng đặc biệt hữu ích cho các mô hình cây vì các mô hình này không có cách tổng hợp thông tin tự nhiên trên nhiều đặc trưng cùng một lúc.

HỎI ĐÁP