Làm sạch dữ liệu

1. Dữ liệu xấu
   - ▼ Các loại lỗi
     - Thiếu giá trị
     - Trùng lặp giá trị
     - Giá trị xấu
     - …
   - ▼ Giá trị xấu
     - Là dữ liệu tạo ra do lỗi
     - Các phương pháp tìm dữ liệu xấu
       - Sử dụng groupby để xem thống kê trên mỗi số liệu

         ```
         df.groupby('name').describe()
         ```

       - Phân loại dữ liệu

         ```
         df['name'].value_counts()
         ```

       - Bảng tổng hợp trong đó index là thời gian

         ```
         df.pivot(df, index='time', columns='name').plot(subplots=True)
         ```

       - Câu truy vấn

         ```
         df.query('name == "cpu" & (value < 0 | value > 100)
         ```

- Sử dụng Z, độ lệch chuẩn, mean

```
men = df[df['name'] == 'mem']['value']
z_score = (mem-mem.mean())/mem.std()
bad_mem = mem[z_score.abs() > 2]

df.loc[bad_mem.index]
```

▼ Trùng lặp

- Tìm hàng trùng

```
df.duplicated()
```

- Tìm giá trị trùng lặp

```
df.duplicated(['date','name'])
```

2. Nguyên nhân gây ra lỗi
   - Lỗi của con người
   - Lỗi máy
   - Lỗi thiết kế
     - Giao diện người dùng
     - Thiết kế hệ thống
3. Phát hiện lỗi

▼ Lược đồ

- Lược đồ JSON, lược đồ cơ sở dữ liệu
- Các ngôn ngữ như QUE
- Trong python sử dụng các thư viện như pydantic hoặc marshmallow để kiểm tra tính hợp lệ của dữ liệu

▼ Thẩm định

▼ Tìm dữ liệu bị thiếu

- Xem các hàng có ít nhất một giá trị thiếu

```
df[df.isnull().any(axis=1)]
```

- Xem tên ở hàng cuối cùng

```
df.iloc[-1]['name']
```

- Tách các khoảng trắng và xem tên ở hàng cuối cùng

```
df['name'] = df['name'].str.strip()
df.iloc[-1]['name']
```

- Tìm các giá trị nan

```
mask =df['name'].str.strip() == ''
df.loc[mask, 'name'] = np.nan

df[df.isnull().any(axis=1)]
```

▼ Kiến thức miền

- Kiến thức miền giúp thấy được giá trị hợp lệ cho cột là gì

▼ Nhóm con

- Ghép hai dataframe

```
df1= pd.merge(df, df0, how = 'left')
```

4. Ngăn ngừa lỗi

▼ Định dạng tuần tự hóa

▼ Chữ kí điện tử

▼ Đường ống dữ liệu và tự động hóa

- Các hệ thống để tạo đường dẫn dữ liệu
  - Apache Airflow
  - Đường ống dữ liệu
    - Sử dụng thư viện Pyinvoke
    - Phát hiện xem có thời gian bắt đầu lớn hơn hoặc bằng thời gian kết thúc không

```
def validate(df):
  bad_time = df.query('strat >= end')
  if len(bad_time) >0:
    raise ValueError(bad_time)
```

▼ Giao dịch

▼ Tổ chức dữ liệu và dữ liệu ngăn nắp

▼ Chỉ số chất lượng dữ liệu và quy trình

5. Sửa lỗi

▼ Đổi tên các trường

- Sử dụng hàm rename

▼ Sửa chữa các loại

▼ Kết hợp và tách dữ liệu

- Sử dụng hàm concat để kết hợp dữ liệu

```
pd.concat([df,time], axis =1)
```

- Sử dụng hàm split

```
time = df['time'].str.split('-', expand = True)
time.columns = ['start','end']
```

▼ Xóa dữ liệu

- Sử dụng hàm drop
- Nếu đã lọc ra những trường hợp muốn xóa, ta có thể sử dụng

```
df = df[~mask]
```

▼ Điền các giá trị còn thiếu

- Sử dụng hàm fillna

▼ Định hình lại dữ liệu

- Sử dụng hàm melt để chuyển đổi định dạng rông sang định dạng dài hoặc một định dạng hẹp
- Ví dụ

```
df = pd.melt(
    df,
    value_vars=['cpu', 'memory'],
    id_vars=['time'],
    var_name='metric',
)
df
```

- Còn có thể sử dụng hàm pivot_table để định dạng lại dữ liệu ở nhiều định dạng khác nhau