

AI VIET NAM – COURSE 2024

Data Analysis - Exercise

Ngày 17 tháng 8 năm 2024

Phần I: Lý thuyết

Pandas là một thư viện trong Python với ưu điểm là nhanh, mạnh, linh động, dễ sử dụng, mã nguồn mở, công cụ dùng để phân tích và thao tác dữ liệu. Pandas được xây dựng trên thư viện NumPy và có nhiều functions hỗ trợ cleaning, analyzing, và manipulating data, có thể giúp ta extract valuable insights của các tập dữ liệu. Pandas rất hiệu quả khi sử dụng trên dữ liệu bảng, như SQL table hoặc Excel spreadsheets.



Hình 1: Logo thư viện Pandas

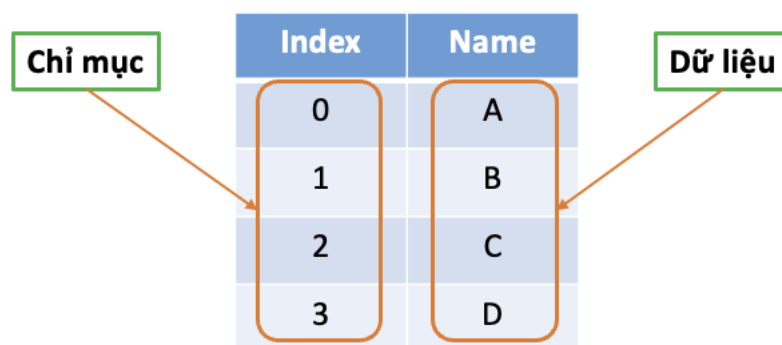
Một số đặc điểm của Pandas:

- Thao tác với các nguồn dữ liệu từ file csv, excel file, SQL, JSON file.
- Cung cấp các loại cấu trúc dữ liệu khác nhau như Series, DataFrame và Panel.
- Có thể đáp ứng nhiều dạng dataset khác nhau như time series, heterogeneous data, tabular và matrix data.
- Có thể làm việc với missing data bằng cách xóa chúng hoặc gán cho chúng giá trị zeros hoặc giá trị phù hợp với trạng thái test.
- Có thể dùng cho việc parsing và conversion data.
- Cung cấp các kỹ thuật lọc dữ liệu.
- Cung cấp time series functionality – date range generation, frequency conversion, moving window statistics, data shifting và lagging.

- Tích hợp tốt với các thư viện khác của Python như Scikit-learn, statmodels và SciPy.
- Có hiệu năng cao.

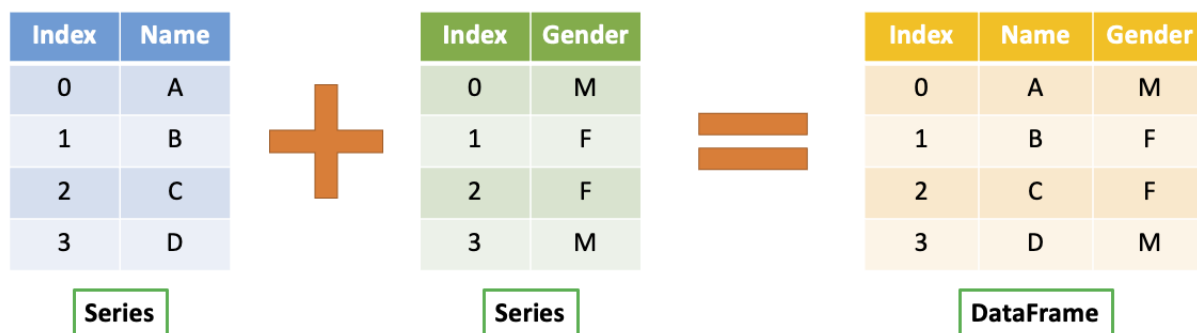
Cấu trúc dữ liệu trong Pandas: Pandas được xây dựng trên NumPy array, bao gồm Series, DataFrame và Panel:

- **Series:** Có cấu trúc là mảng 1D với dữ liệu đồng nhất, loại dữ liệu có thể là integer, string, float,.. trục đánh nhãn được gọi là chỉ mục (index). Kích thước của series là không thể thay đổi (immutable) và giá trị dữ liệu có thể thay đổi (mutable). Để khởi tạo Series có thể dùng `pandas.Series(data, index, dtype, copy)`, trong đó:
 - **data:** Nhận các giá trị có dạng ndarray, list, dictionary, constant,...
 - **index:** Giá trị index phải là duy nhất (unique), có thể hash và có kích thước bằng **data**, mặc định **index** có giá trị 0, 1, 2.
 - **dtype:** Loại dữ liệu của giá trị bên trong Series.



Hình 2: Ví dụ về một Series trong Pandas

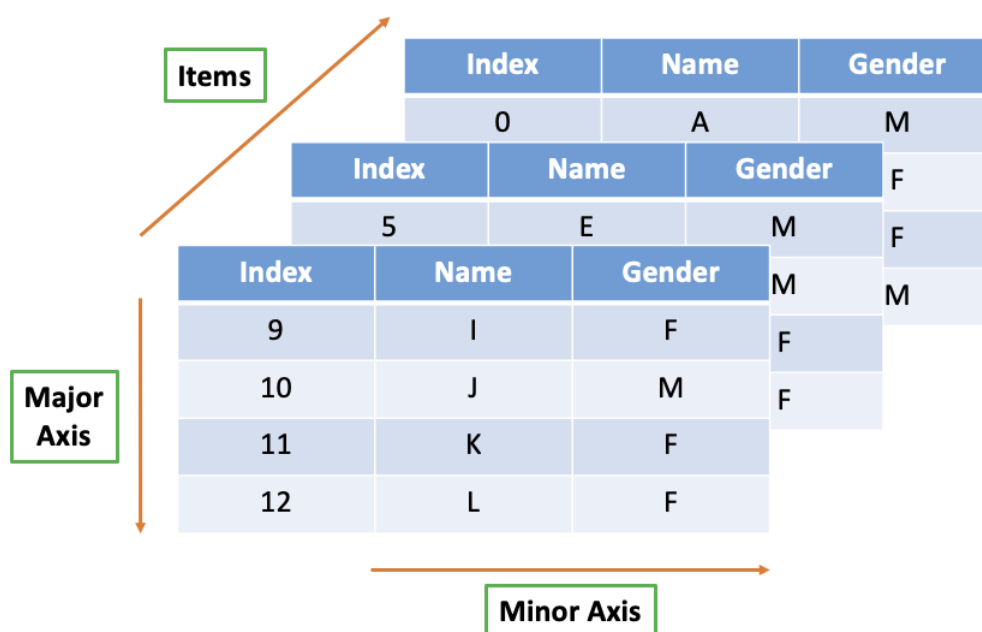
- **DataFrame:** Là cấu trúc dữ liệu 2D, có dạng bảng bao gồm các cột và hàng, các cột có thể định nghĩa loại dữ liệu khác nhau. Các cột có các kiểu dữ liệu khác nhau như float64, int, bool,.. **Một cột của DataFrame là một cấu trúc Series.** Các chiều DataFrame được đánh nhãn theo các hàng và cột. Từ đó, ta có thể thao tác trên cả hàng và cột. Để khởi tạo DataFrame, có thể thực hiện bởi `pandas.DataFrame(data, index, columns, dtype, copy)`.
 - **data:** Nhận các giá trị như ndarray, series, map, lists, dict, constants và DataFrame khác.
 - Các tham số khác tương tự như Series, pandas DataFrame có thể được tạo dùng các input như Lists, Dict, Series, Numpy ndarrays, DataFrame khác.



Hình 3: Ví dụ về DataFrame trong Pandas. Có thể coi DataFrame là một danh sách chứa các Series.

- **Panel:** Là một 3D container, trong đó:

- **items:** axis 0, mỗi item tương ứng DataFrame chứa bên trong.
- **major_axis:** axis 1, nó là các hàng (rows) của mỗi DataFrame.
- **minor_axis:** axis 2, nó là các cột (columns) của mỗi DataFrame



Hình 4: Ví dụ về Panel trong Pandas. Có thể coi Panel là một danh sách chứa các DataFrame.

Một số function trên Pandas thường dùng để xử lý dữ liệu:

- **Handle missing values:** `isna()`, `notna()` – tìm kiếm các giá trị NA, `isnull()`.
- **Indexing and slicing in Pandas:** `.loc` (label based), `.iloc` (integer based), `.ix` (label and integer based).
- **Các query như trong excel hay SQL:** `where()`, `query()`.

- **Sort:** `sort_index()`, `sort_values()`.
- **Series basic functionality:** `axes`, `dtype`, `empty`, `ndim`, `size`, `values`, `head()`, `tail()`.
- **Dataframe basic functionality:** `T`, `axes`, `dtypes`, `empty`, `ndim`, `shape`, `size`, `values`, `head()`, `tail()`.
- **Các function liên quan thống kê:** `count()`, `sum()`, `mean()`, `median()`, `mode()`, `std()`, `min()`, `max()`, `abs()`, `prod()`, `cumsum()`, `cumprod()`, `describe()`, `ptc_change()`, `cov()`, `corr()`, `rank()`, `var()`, `skew()`, `apply()`.
- **Các function filter data:** `groupby()`, `get_group()`, `merge()`, `concat()`, `append()`, `melt()`, `pivot()`, `pivot_table()`.
- **Một số function khác:** `get_option()`, `set_option()`, `reset_option()`, `describe_option()`, `option_context()`.

Phần II: Bài tập

Trong phần này, chúng ta sẽ sử dụng pandas để thực hiện một số kỹ thuật phân tích trên hai bộ dữ liệu về text và time-series. Các câu bài tập được chia thành các bước thực hiện trong bài toán.

A. Data Analysis with IMDB Movie data

IMDB Movie dataset là một bộ dữ liệu đánh giá phim, dùng để phân tích mức độ quan tâm của phim theo một số tiêu chí như: đạo diễn, diễn viên, tên phim... nhằm đưa ra những góc nhìn hỗ trợ dự đoán trong tương lai. Các bạn tải bộ dữ liệu **IMDB-Movie-Data.csv** tại [đây](#).

Các bước cần thực hiện trong bài toán:

1. Read data
2. View the data
3. Understand some basic information about the data
4. Data Selection – Indexing and Slicing data
5. Data Selection – Based on Conditional filtering
6. Groupby operations
7. Sorting operation
8. View missing values
9. Deal with missing values - Deleting
10. Deal with missing values - Filling
11. Apply() functions

Ta bắt đầu thực hiện và nhận định ở mỗi bước, code được thực hiện trên Google Colab:

1. **Import libraries và load dataset:** Để đọc một file .csv trong pandas, ta có thể dùng hàm `read_csv()` như sau:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 dataset_path = 'IMDB-Movie-Data.csv'
6
7 # Read data from .csv file
8 data = pd.read_csv(dataset_path)
```

Ngoài ra, ta có thể đọc đồng thời chỉ định cột làm chỉ mục cho bảng dữ liệu (mặc định pandas sẽ tự tạo một cột chỉ mục riêng). Ở đây, ta có thể chọn cột **Title** làm cột chỉ mục như sau (cột chỉ mục không được chứa giá trị trùng lặp):

```
1 # Read data with specified explicit index.
2 # We will use this later in our analysis
3 data_indexed = pd.read_csv(dataset_path, index_col="Title")
```

2. View the data: Coi qua 5 hàng đầu tiên của bảng dữ liệu bằng cách sử dụng `head()`:

```
1 # Preview top 5 rows using head()
2 data.head()
```

	Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)
0	1	Guardians of the Galaxy	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1	757074	333.13
1	2	Prometheus	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012	124	7.0	485820	126.46
2	3	Split	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016	117	7.3	157606	138.12
3	4	Sing	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey, Reese Witherspoon, Seth Ma...	2016	108	7.2	60545	270.32
4	5	Suicide Squad	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016	123	6.2	393727	325.02

data

Hình 5: Một số mẫu dữ liệu đầu tiên của bộ dữ liệu

3. Understand some basic information about the data:

```
1 #Let's first understand the basic information about this data
2 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Rank                  1000 non-null  int64
1   Title                 1000 non-null  object
2   Genre                 1000 non-null  object
3   Description            1000 non-null  object
4   Director              1000 non-null  object
5   Actors                1000 non-null  object
6   Year                  1000 non-null  int64
7   Runtime (Minutes)     1000 non-null  int64
8   Rating                1000 non-null  float64
9   Votes                 1000 non-null  int64
10  Revenue (Millions)    872 non-null   float64
11  Metascore             936 non-null   float64
dtypes: float64(3), int64(4), object(5)
memory usage: 93.9+ KB
```

Hình 6: Thông tin cơ bản về bảng dữ liệu

```
1 data.describe()
```

	Rank	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
count	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	872.000000	936.000000
mean	500.500000	2012.783000	113.172000	6.723200	1.698083e+05	82.956376	58.985043
std	288.819436	3.205962	18.810908	0.945429	1.887626e+05	103.253540	17.194757
min	1.000000	2006.000000	66.000000	1.900000	6.100000e+01	0.000000	11.000000
25%	250.750000	2010.000000	100.000000	6.200000	3.630900e+04	13.270000	47.000000
50%	500.500000	2014.000000	111.000000	6.800000	1.107990e+05	47.985000	59.500000
75%	750.250000	2016.000000	123.000000	7.400000	2.399098e+05	113.715000	72.000000
max	1000.000000	2016.000000	191.000000	9.000000	1.791916e+06	936.630000	100.000000

Hình 7: Tổng quan thống kê dữ liệu từ dataset

Ở đây ta có thể thấy:

- Giá trị min và max của Year, tức dataset chứa các bộ phim từ **2006** tới **2016**.
- Rating trung bình cho các bộ phim là **6.7**, thấp nhất là **1.9**, cao nhất là **9.0**.
- Doanh thu cao nhất đạt được là **936.6** triệu dollar.

4. **Data Selection – Indexing and Slicing data:** Từ bảng dữ liệu, ta có thể tách bất kì cột nào trong bảng dữ liệu để trở thành một Series hoặc một DataFrame, tùy vào phương thức tách ta sử dụng. Ở đây, ta sẽ tách một số cột trong **data** sử dụng kỹ thuật **Indexing**. Để tách cột thành Series, ta thực hiện:

```
1 # Extract data as series
2 genre = data['Genre']
3 genre
```

```
0    Action,Adventure,Sci-Fi
1    Adventure,Mystery,Sci-Fi
2           Horror,Thriller
3    Animation,Comedy,Family
4    Action,Adventure,Fantasy
...
995    Crime,Drama,Mystery
996           Horror
997    Drama,Music,Romance
998    Adventure,Comedy
999    Comedy,Family,Fantasy
Name: Genre, Length: 1000, dtype: object
```

Hình 8: Tách cột **Genre** thành một Series

Để tách cột thành DataFrame, ta thực hiện:

```
1 # Extract data as dataframe
2 data[['Genre']]
```

	Genre
0	Action,Adventure,Sci-Fi
1	Adventure,Mystery,Sci-Fi
2	Horror,Thriller
3	Animation,Comedy,Family
4	Action,Adventure,Fantasy
...	...
995	Crime,Drama,Mystery
996	Horror
997	Drama,Music,Romance
998	Adventure,Comedy
999	Comedy,Family,Fantasy

1000 rows x 1 columns

Hình 9: Tách cột **Genre** thành một DataFrame

Ta có thể chọn và tách cùng một lúc nhiều cột với nhau, tạo thành một DataFrame mới:

```
1 some_cols = data[['Title', 'Genre', 'Actors', 'Director', 'Rating']]
```

Đối với việc tách hàng, ta có thể tách ra một số lượng hàng nhất định, từ chỉ mục X đến chỉ mục Y trong bảng dữ liệu, gọi là **Slicing**. Ví dụ, để tách các hàng thứ 10 đến thứ 15, ta làm như sau:

```
1 data.iloc[10:15][['Title', 'Rating', 'Revenue (Millions)']]
```

Kết hợp với việc chọn cột, ta có một bảng dữ liệu gồm 5 mẫu dữ liệu với các trường thông tin **Title, Rating, Revenue (Millions)**.

	Title	Rating	Revenue (Millions)
10	Fantastic Beasts and Where to Find Them	7.5	234.02
11	Hidden Figures	7.8	169.27
12	Rogue One	7.9	532.17
13	Moana	7.7	248.75
14	Colossal	6.4	2.87

Hình 10: Tách một số cột tạo thành một DataFrame mới

5. **Data Selection – Based on Conditional filtering:** Ta có lấy các hàng trong bảng dữ liệu dựa trên một số điều kiện cần tuân theo. Ví dụ, ta mong muốn lấy các bộ phim từ 2010 tới 2015, với rating nhỏ hơn 6.0 nhưng lại có doanh thu thuộc top 5% trên toàn bộ dataset. Theo đó, ta có thể triển khai code như sau:

```
1 data[((data['Year'] >= 2010) & (data['Year'] <= 2015))
2      & (data['Rating'] < 6.0)
3      & (data['Revenue (Millions)'] > data['Revenue (Millions)'].quantile(0.95))]
```


	Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes
941	942	The Twilight Saga: Eclipse	Adventure,Drama,Fantasy	As a string of mysterious killings grips Seatt...	David Slade	Kristen Stewart, Robert Pattinson, Taylor Laut...	2010	124	4.9	192740

Hình 11: Phim với doanh thu cao trong giai đoạn năm 2010-2015

6. **Groupby Operations:** Groupby là một phép gom nhóm dữ liệu dựa trên một hoặc nhiều biến (ở đây là cột dữ liệu trong bảng). Ví dụ, ta có thể tìm số rating trung bình mà các đạo diễn đạt được bằng cách gom nhóm các chỉ số Rating của các bộ phim theo Director.

```
1 data.groupby('Director')[['Rating']].mean().head()
```

	Rating
Director	
Aamir Khan	8.5
Abdellatif Kechiche	7.8
Adam Leon	6.5
Adam McKay	7.0
Adam Shankman	6.3

Hình 12: Sử dụng groupby để tìm số rating trung bình đạt được của các đạo diễn trong bộ dữ liệu.

7. **Sorting Operations:** Sorting cho phép ta sắp xếp các hàng trong bảng dữ liệu theo thứ tự tăng/giảm dần dựa theo giá trị của cột nào đó trong bảng dữ liệu. Ví dụ, dựa trên kết quả groupby phần trước, ta có thể tìm top 5 đạo diễn đạt số rating trung bình cao nhất như sau:

```
1 data.groupby('Director')[['Rating']].mean().sort_values(['Rating'], ascending=False).head()
```

	Rating
Director	
Nitesh Tiwari	8.80
Christopher Nolan	8.68
Olivier Nakache	8.60
Makoto Shinkai	8.60
Aamir Khan	8.50

Hình 13: 5 đạo diễn có được số Rating trung bình cao nhất.

8. **View missing values:** Các bộ dữ liệu thường sẽ xuất hiện tình trạng bị giá trị rỗng (missing value) trong một vài trường thông tin của một số mẫu dữ liệu. Khi xử lý dữ liệu, ta cần khắc phục vấn đề này. Vì vậy, việc đầu tiên ta cần kiểm tra xem vị trí bị mất mát dữ liệu theo cách sau:

```
1 # To check null values row-wise
2 data.isnull().sum()
```

Rank	0
Title	0
Genre	0
Description	0
Director	0
Actors	0
Year	0
Runtime (Minutes)	0
Rating	0
Votes	0
Revenue (Millions)	128
Metascore	64
dtype:	int64

Hình 14: Bảng tổng sắp số lượng các giá trị null có trong từng cột của bảng dữ liệu

Ở đây ta thấy Revenue (Millions) và Metascore là 2 cột có chứa dữ liệu null. Để xử lý vấn đề mất mát dữ liệu, có hai phương án chính: hoặc thế các vùng trống bằng một giá trị nào đó hoặc loại bỏ chúng.

9. **Deal with missing values - Deleting** Đối với phương án loại bỏ, ta có thể loại bỏ toàn bộ cột chứa nhiều giá trị null (nếu có thể) hoặc chỉ loại bỏ các hàng chứa giá trị null. Đối với xóa cột, ta thực hiện:

```
1 # Use drop function to drop columns
2 data.drop('Metascore', axis=1).head()
```

Lệnh trên vẫn chưa drop data thực trên server cho tới khi ta thêm **inplace=True**.

Đối với xóa hàng, ta dùng:

```
1 data.dropna()
```

10. **Dealing with missing values - Filling:** Đối với phương án thế giá trị mới vào các ô trống, ta có thể sử dụng các giá trị mean, median... của cột dữ liệu tương ứng để thay thế (việc chọn giá trị để thay thế còn tùy thuộc vào tính chất của bộ dữ liệu, bài toán đang giải quyết...). Ví dụ, có một vài hàng có Revenue mang giá trị null, ta có thể gán cho nó giá trị trung bình như sau:

```
1 revenue_mean = data_indexed['Revenue (Millions)'].mean()
2 print("The mean revenue is: ", revenue_mean)
3
4 # We can fill the null values with this mean revenue
5 data_indexed['Revenue (Millions)'].fillna(revenue_mean, inplace=True)
```

11. **apply() functions:** Apply functions được dùng khi ta muốn thực thi một hàm nào đó lên các hàng trong bảng dữ liệu. Sau khi thực thi, kết quả trả về từ hàm chính là giá trị mới của hàng tương ứng. Ví dụ, ta muốn phân loại phim theo ba mức độ ['Good', 'Average', 'Bad'] dựa trên Rating, ta có thể định nghĩa một hàm để làm điều này và apply nó lên DataFrame:

```
1 # Classify movies based on ratings
2 def rating_group(rating):
3     if rating >= 7.5:
4         return 'Good'
5     elif rating >= 6.0:
6         return 'Average'
7     else:
```

```

8         return 'Bad'
9
10 # Lets apply this function on our movies data
11 # creating a new variable in the dataset to hold the rating category
12 data['Rating_category'] = data['Rating'].apply(rating_group)
13
14 data[['Title', 'Director', 'Rating', 'Rating_category']].head(5)

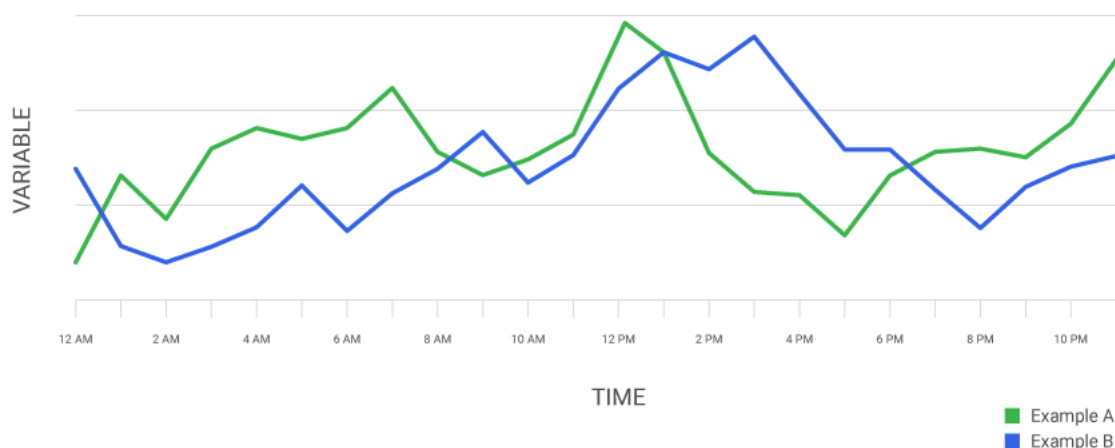
```

	Title	Director	Rating	Rating_category
0	Guardians of the Galaxy	James Gunn	8.1	Good
1	Prometheus	Ridley Scott	7.0	Average
2	Split	M. Night Shyamalan	7.3	Average
3	Sing	Christophe Lourdelet	7.2	Average
4	Suicide Squad	David Ayer	6.2	Average

Hình 15: DataFrame sau khi được apply hàm `rating_group()`. Kết quả trả về sau khi thực thi hàng này sẽ được đưa vào một cột mới mang tên **Rating_category**

B. Data Analysis with Time Series data

Time series data là một dạng dữ liệu với giá trị được đo lường tại những điểm khác nhau theo thời gian. Một số dữ liệu time series được phân bố theo tần suất nhất định, ví dụ như thời tiết trong 1 giờ, lượng truy cập website trong ngày, tổng doanh thu trong tháng... Dữ liệu time series cũng có thể phân bố với khoảng cách không đều, ví dụ như số lượng cuộc gọi khẩn cấp trong ngày hoặc nhật ký hệ thống.



Hình 16: Hình dạng đồ thị của dữ liệu time-series

Trong phần này, chúng ta sẽ khai thác khía cạnh sắp xếp và trực quan hóa dữ liệu cho time series. Cụ thể với dữ liệu time series cho năng lượng, ta sẽ làm quen với áp dụng của các kỹ thuật time-based indexing, resampling, và rolling. Việc này sẽ giúp ta phân tích được các khía cạnh thông tin ẩn quan trọng trong dữ liệu. Ví dụ, Rolling windows có thể giúp ta khám phá các biến thể về nhu cầu điện và cung cấp năng lượng tái tạo theo thời gian. Chúng ta dùng bộ dữ liệu daily time series của Open Power System Data (OPSD) ở Đức, gồm tổng lượng tiêu thụ điện, sản xuất điện gió và sản xuất điện mặt trời trên toàn quốc trong giai đoạn 2006-2017. Các bạn tải bộ dữ liệu `opsd_germany_daily.csv` tại [đây](#).

Chúng ta sẽ thực hiện các vấn đề sau:

- Import libraries and read dataset
- Time-based indexing
- Visualizing time series data
- Seasonality
- Frequencies
- Resampling
- Rolling windows

- Trends

Chúng ta sẽ khám phá mức tiêu thụ và sản xuất điện ở Đức thay đổi theo thời gian như thế nào, và trả lời các câu hỏi:

- Khi nào mức tiêu thụ điện thường cao nhất và thấp nhất?
- Năng lượng gió và mặt trời được sản xuất đã thay đổi theo mùa như thế nào?
- Xu hướng dài hạn trong tiêu thụ điện, năng lượng mặt trời và năng lượng gió là gì?
- So sánh tỷ lệ sản lượng năng lượng gió và mặt trời với mức tiêu thụ năng lượng gió và mặt trời, và tỷ lệ này đã thay đổi như thế nào theo thời gian?

1. **Import libraries and read dataset:** Đầu tiên, ta vẫn dùng hàm `read_csv()` để đọc bảng dữ liệu:

```
1 import pandas as pd
2
3 dataset_path = './opsd_germany_daily.csv'
4
5 # Read data from .csv file
6 opsd_daily = pd.read_csv(dataset_path)
7
8 print(opsd_daily.shape)
9 print(opsd_daily.dtypes)
10 opsd_daily.head(3)
11
```

Ta được kết quả như hình bên dưới, có thể quan sát thấy nhiều giá trị bị bỏ trống ở các cột Wind, Solar, Wind+Solar:

```
(4383, 5)
Date          object
Consumption    float64
Wind           float64
Solar          float64
Wind+Solar     float64
dtype: object
```

	Date	Consumption	Wind	Solar	Wind+Solar
0	2006-01-01	1069.184	NaN	NaN	NaN
1	2006-01-02	1380.521	NaN	NaN	NaN
2	2006-01-03	1442.533	NaN	NaN	NaN

Hình 17: Một số dữ liệu đầu tiên của DataFrame

Đối với dạng dữ liệu Time Series, ta có thể chọn cột **Date** làm index (vì giá trị cột này trong bộ dữ liệu luôn là duy nhất (unique)):

```
1 opsd_daily = opsd_daily.set_index('Date')
2 opsd_daily.head(3)
3
```

	Consumption	Wind	Solar	Wind+Solar
Date				
2006-01-01	1069.184	NaN	NaN	NaN
2006-01-02	1380.521	NaN	NaN	NaN
2006-01-03	1442.533	NaN	NaN	NaN

Hình 18: Bảng dữ liệu sau khi chọn cột **Date** làm index

Ta có thể thực hiện lại bước load file và lúc này, chỉ định cột sẽ làm chỉ mục ngay từ lúc thực hiện lời gọi hàm, đồng thời tạo thêm các số cột **Year**, **Month**, **Weekday** trích từ cột **Date** để thuận tiện cho việc xử lý một số bước về sau:

```

1 opsd_daily = pd.read_csv('opsd_germany_daily.csv', index_col=0, parse_dates=True)
2
3 # Add columns with year, month, and weekday name
4 opsd_daily['Year'] = opsd_daily.index.year
5 opsd_daily['Month'] = opsd_daily.index.month
6 opsd_daily['Weekday Name'] = opsd_daily.index.day_name()
7 # Display a random sampling of 5 rows
8 opsd_daily.sample(5, random_state=0)

```

	Consumption	Wind	Solar	Wind+Solar	Year	Month	Weekday Name
Date							
2008-08-23	1152.011	NaN	NaN	NaN	2008	8	Saturday
2013-08-08	1291.984	79.666	93.371	173.037	2013	8	Thursday
2009-08-27	1281.057	NaN	NaN	NaN	2009	8	Thursday
2015-10-02	1391.050	81.229	160.641	241.870	2015	10	Friday
2009-06-02	1201.522	NaN	NaN	NaN	2009	6	Tuesday

Hình 19: DataFrame với các cột mới: Year, Month, Weekday

2. **Time-based indexing:** Một trong những tính năng nổi trội của pandas khi xử lý dữ liệu time series là tính năng time-based indexing, liên quan đến việc dùng dates và times để tổ chức và truy cập dữ liệu (khá giống với Indexing ở phần trước nhưng giá trị lúc này sẽ là ngày tháng năm). Việc này cho phép ta dùng loc accessor để thực thi. Ví dụ, ta có thể truy cập dữ liệu theo một khoảng thời gian từ ngày **2014-01-20** đến ngày **2014-01-22**:

```

1 opsd_daily.loc['2014-01-20':'2014-01-22']

```

	Consumption	Wind	Solar	Wind+Solar	Year	Month	Weekday Name
Date							
2014-01-20	1590.687	78.647	6.371	85.018	2014	1	Monday
2014-01-21	1624.806	15.643	5.835	21.478	2014	1	Tuesday
2014-01-22	1625.155	60.259	11.992	72.251	2014	1	Wednesday

Hình 20: Lấy các mẫu dữ liệu từ ngày 20/1/2014 đến 22/1/2014

Một tính năng khác của pandas là partial-string indexing, cho phép ta Slicing theo mô tả thời gian một cách chung chung, không cần cụ thể ngày tháng năm như ở phần trên. Ví dụ:

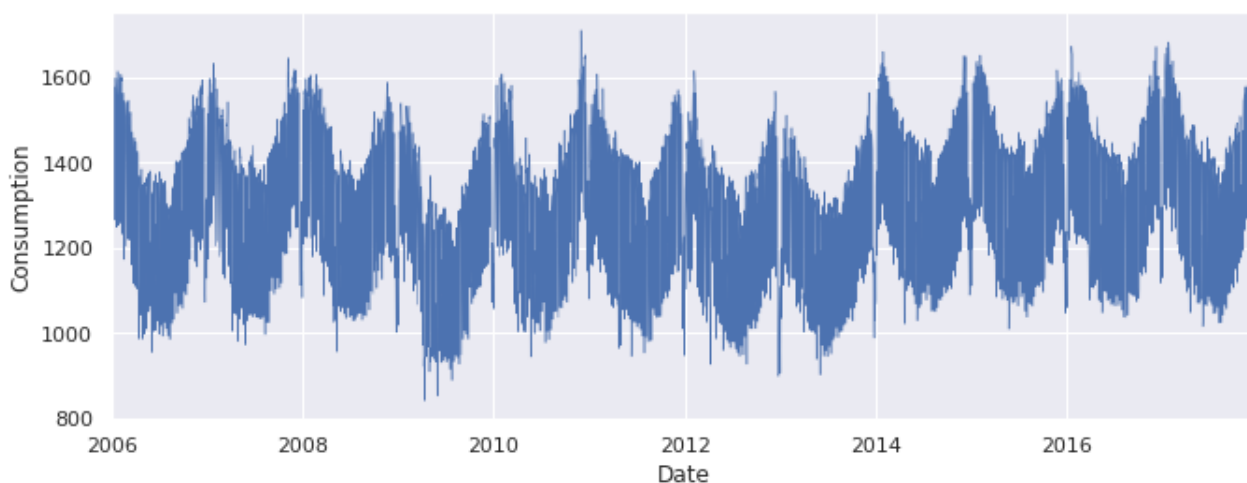
```
1 opsd_daily.loc['2012-02']
```

	Consumption	Wind	Solar	Wind+Solar	Year	Month	Weekday	Name
Date								
2012-02-01	1511.866	199.607	43.502	243.109	2012	2	Wednesday	
2012-02-02	1563.407	73.469	44.675	118.144	2012	2	Thursday	
2012-02-03	1563.631	36.352	46.510	82.862	2012	2	Friday	
2012-02-04	1372.614	20.551	45.225	65.776	2012	2	Saturday	
2012-02-05	1279.432	55.522	54.572	110.094	2012	2	Sunday	

Hình 21: Partial-string indexing. Với việc chỉ đặt '2012-02', ta có thể lấy được toàn bộ các mẫu dữ liệu thuộc '2012-02'.

3. **Visualizing time series data:** Với việc pandas có hỗ trợ trực quan hóa dữ liệu lên đồ thị, phối hợp với thư viện seaborn ta có thể dễ dàng trực quan hóa được dữ liệu time-series lên đồ thị. Ví dụ, ta trực quan (plot) dữ liệu cột **Consumption** như sau:

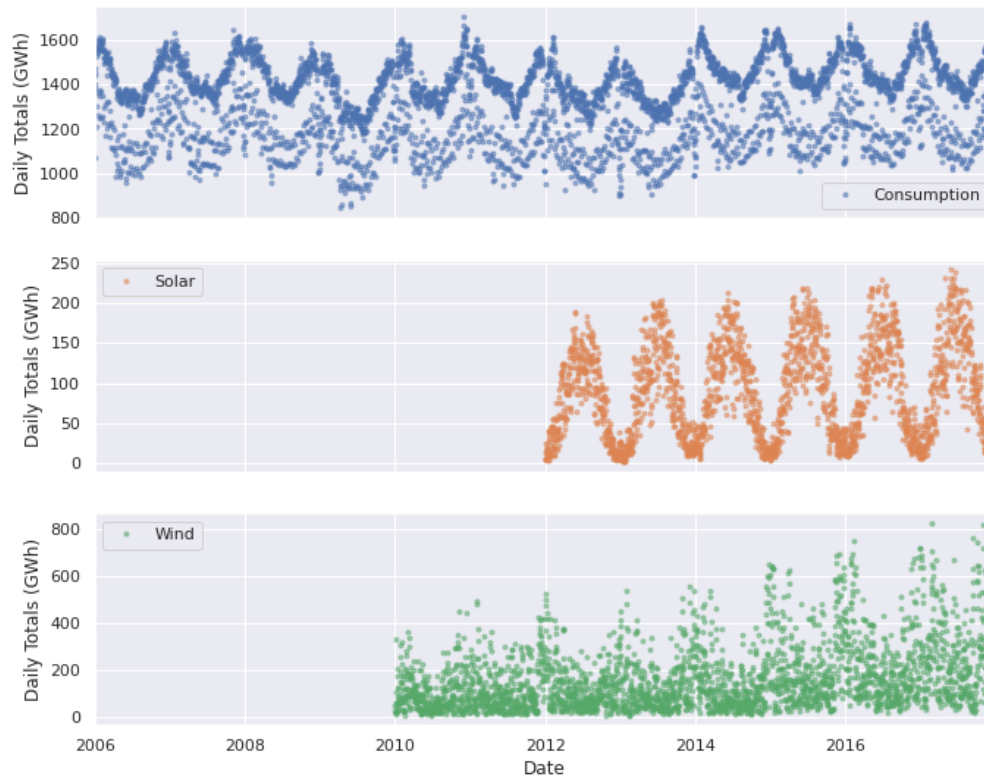
```
1 import matplotlib.pyplot as plt
2 # Display figures inline in Jupyter notebook
3
4 import seaborn as sns
5 # Use seaborn style defaults and set the default figure size
6 sns.set(rc={'figure.figsize':(11, 4)})
7 opsd_daily['Consumption'].plot(linewidth=0.5);
```



Hình 22: Đồ thị dữ liệu về mức tiêu thụ điện năng hàng ngày tại Đức

Ta có thể plot cùng lúc một số cột dữ liệu khác thành từng đồ thị riêng lẻ:

```
1 cols_plot = ['Consumption', 'Solar', 'Wind']
2 axes = opsd_daily[cols_plot].plot(marker='.', alpha=0.5, linestyle='None',
3     figsize=(11, 9), subplots=True)
4 for ax in axes:
5     ax.set_ylabel('Daily Totals (GWh)')
6 plt.show()
```



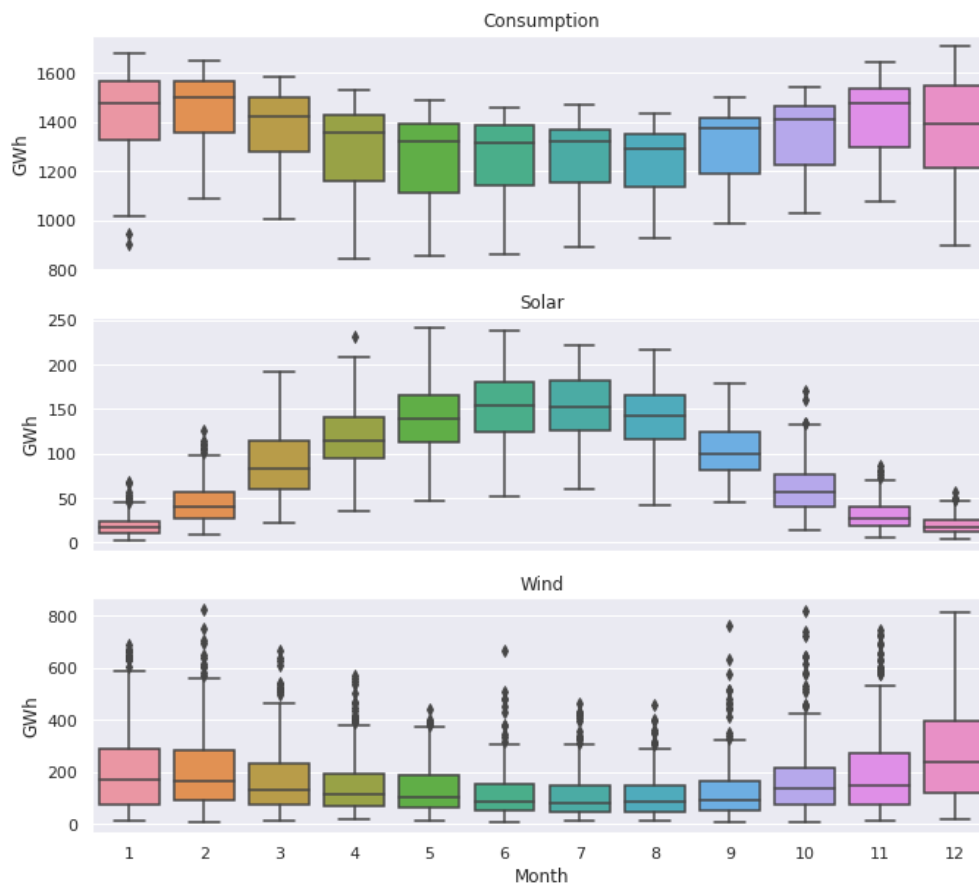
Hình 23: Đồ thị về mức tiêu thụ điện, sản lượng điện năng từ mặt trời và sản lượng điện năng từ gió

4. **Seasonality:** Tạm dịch: tính thời vụ. Chỉ số về các đặc trưng lặp đi lặp lại trong một khoảng thời gian cố định xuyên suốt các năm. Các dạng đặc trưng này thường được ảnh hưởng bởi rất nhiều yếu tố khác nhau. Ở trong dữ liệu của bài, ta có thể khai phá tính thời vụ của dữ liệu, dùng seaborn để vẽ, và group dữ liệu thành từng nhóm như sau:

```

1 fig, axes = plt.subplots(3, 1, figsize=(11, 10), sharex=True)
2 for name, ax in zip(['Consumption', 'Solar', 'Wind'], axes):
3     sns.boxplot(data=opsd_daily, x='Month', y=name, ax=ax)
4     ax.set_ylabel('GWh')
5     ax.set_title(name)
6     # Remove the automatic x-axis label from all but the bottom subplot
7     if ax != axes[-1]:
8         ax.set_xlabel('')

```

Hình 24: Biểu diễn phân bố của các cột Consumption, Wind, Solar theo Month

5. **Frequencies:** Trong `DatetimeIndex` của pandas, ta có thể sử dụng các giá trị thời gian sẵn có để tạo thành một chuỗi giá trị theo tần suất. Ví dụ, với hai giá trị '1998-03-10' và '1998-03-14', ta có thể tạo một danh sách thời gian với tần suất theo ngày. Tức danh sách mới của chúng ta trở thành: '1998-03-10', '1998-03-11', '1998-03-12', '1998-03-13', '1998-03-14'. Việc này được thực hiện bằng cách cài đặt thuộc tính 'freq'.

```
1 pd.date_range('1998-03-10', '1998-03-15', freq='D')
```

```
DatetimeIndex(['1998-03-10', '1998-03-11', '1998-03-12', '1998-03-13',
               '1998-03-14', '1998-03-15'],
              dtype='datetime64[ns]', freq='D')
```

Hình 25: Ví dụ về lấy tần suất theo ngày từ 10/3/1998 đến 15/3/1998

Với tính năng này của pandas, ta có thể thực hiện việc thể dữ liệu bị mất bằng kỹ thuật forward fill (ffill). Kỹ thuật này liên quan đến việc sử dụng giá trị ghi nhận được tại thời điểm trước đó làm giá trị thay thế cho toàn bộ giá trị bị mất mất sau đó trước khi gặp được mẫu dữ liệu có giá trị. Ví dụ, giả sử ta biết được giá trị Consumption của một vài ngày như sau:

```
1 # To select an arbitrary sequence of date/time values from a pandas time series,
2 # we need to use a DatetimeIndex, rather than simply a list of date/time strings
3 times_sample = pd.to_datetime(['2013-02-03', '2013-02-06', '2013-02-08'])
4 # Select the specified dates and just the Consumption column
```

```

5 consum_sample = opsd_daily.loc[times_sample, ['Consumption']].copy()
6 consum_sample

```

	Consumption
2013-02-03	1109.639
2013-02-06	1451.449
2013-02-08	1433.098

Hình 26: Lấy dữ liệu của 3 ngày trong bộ dữ liệu gốc làm ví dụ mẫu

```

1 # Convert the data to daily frequency, without filling any missings
2 consum_freq = consum_sample.asfreq('D')
3 # Create a column with missings forward filled
4 consum_freq['Consumption - Forward Fill'] = consum_sample.asfreq('D', method='
    ffill')
5 consum_freq

```

	Consumption	Consumption - Forward Fill
2013-02-03	1109.639	1109.639
2013-02-04	NaN	1109.639
2013-02-05	NaN	1109.639
2013-02-06	1451.449	1451.449
2013-02-07	NaN	1451.449
2013-02-08	1433.098	1433.098

Hình 27: Thực hiện ffill vào các ngày khác trong phạm vi từ ngày 3/2/2013 đến 8/2/2013

Với giá trị tiêu thụ điện năng của 3 ngày, ta có thể thế giá trị cho các ngày còn lại trong phạm vi của 3 ngày trên sử dụng ffill.

- Resampling:** Là kỹ thuật dùng để thay đổi tần số biểu diễn của bộ dữ liệu time series, có thể gia tăng hoặc giảm đi tần số lấy mẫu. Ví dụ, ta có thể giảm tần số của bộ dữ liệu hiện tại từ ngày sang tháng. Điều này đồng nghĩa với việc bộ dữ liệu mới của chúng ta sẽ có ít mẫu dữ liệu hơn bản gốc.

Resampling thường hữu dụng với time series cho lower hoặc higher frequency. Resampling cho lower frequency (downsampling) thường liên quan tới hoạt động tổng hợp, ví dụ mức doanh thu trong tháng từ dữ liệu ngày. Resampling cho higher frequency (upsampling) ít phổ biến hơn, thường dùng trong việc nội suy. Ở đây, ta thử áp dụng downsampling cho bộ dữ liệu hiện tại như sau:

```

1 # Specify the data columns we want to include (i.e. exclude Year, Month, Weekday
    Name)
2 data_columns = ['Consumption', 'Wind', 'Solar', 'Wind+Solar']
3 # Resample to weekly frequency, aggregating with mean
4 opsd_weekly_mean = opsd_daily[data_columns].resample('W').mean()
5 opsd_weekly_mean.head(3)

```

Ở đoạn code trên, ta downsampling từ tần số theo ngày sang tuần. Giá trị của các cột lúc này sẽ là trung bình của 7 ngày trong tuần.

	Consumption	Wind	Solar	Wind+Solar
Date				
2006-01-01	1069.184000	NaN	NaN	NaN
2006-01-08	1381.300143	NaN	NaN	NaN
2006-01-15	1486.730286	NaN	NaN	NaN

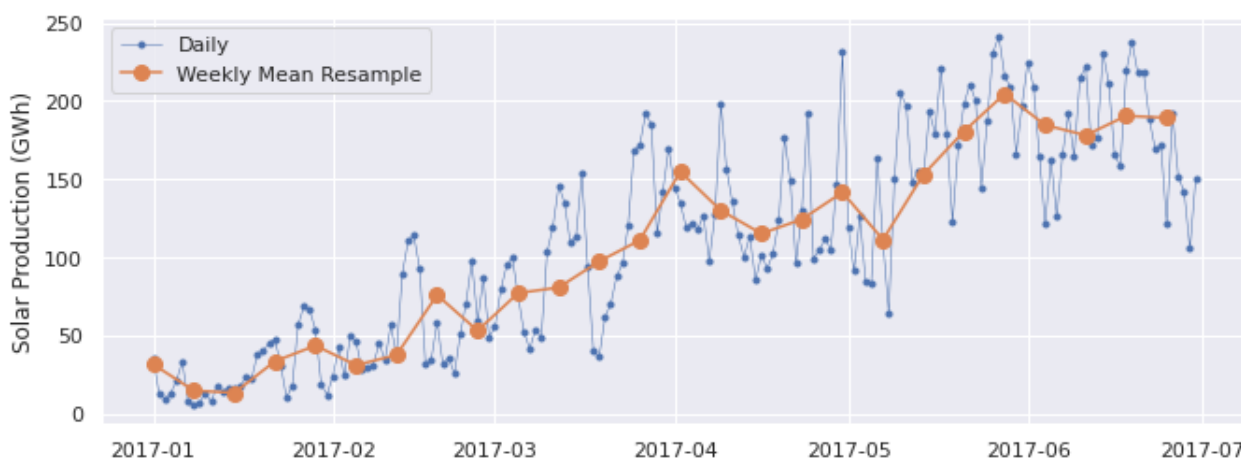
Hình 28: Sử dụng kỹ thuật Resampling để đổi tần số lấy mẫu của bộ dữ liệu từ ngày sang tuần

Dĩ nhiên, khi ta downsampling bộ dữ liệu, số lượng mẫu dữ liệu của bảng dữ liệu mới sẽ ít hơn so với bảng gốc và ít hơn 1/7 lần. Có thể kiểm tra bằng cách dùng thuộc tính shape của DataFrame:

```
1 print(opsd_daily.shape[0])
2 print(opsd_weekly_mean.shape[0])
```

Ta visualize daily và weekly time series của Solar trong 6 tháng như sau:

```
1 # Start and end of the date range to extract
2 start, end = '2017-01', '2017-06'
3 # Plot daily and weekly resampled time series together
4 fig, ax = plt.subplots()
5 ax.plot(opsd_daily.loc[start:end, 'Solar'],
6 marker='.', linestyle='-', linewidth=0.5, label='Daily')
7 ax.plot(opsd_weekly_mean.loc[start:end, 'Solar'],
8 marker='o', markersize=8, linestyle='-', label='Weekly Mean Resample')
9 ax.set_ylabel('Solar Production (GWh)')
10 ax.legend()
11 plt.show()
```



Hình 29: Đồ thị Time series của **Solar** theo ngày và theo tuần

Lưu ý rằng bảng dữ liệu gốc của chúng ta có tồn một số giá trị null. Vì vậy để đảm bảo toàn bộ các mẫu có giá trị, ta cài đặt tham số **min_count** vào để xử lý vấn đề này. Ví dụ, ta resampling bộ dữ liệu thành theo năm, để đảm bảo các ngày trong năm đều tồn tại giá trị non-null, ta có thể cài đặt **min_count=360** (các bạn có thể chọn **min_count** bằng một giá trị khác tùy vào quan sát cá nhân):

```
1 # Compute the annual sums, setting the value to NaN for any year which has
2 # fewer than 360 days of data
3 opsd_annual = opsd_daily[data_columns].resample('YE').sum(min_count=360)
```

```

4 # The default index of the resampled DataFrame is the last day of each year,
5 # ('2006-12-31', '2007-12-31', etc.) so to make life easier, set the index
6 # to the year component
7 opsd_annual = opsd_annual.set_index(opsd_annual.index.year)
8 opsd_annual.index.name = 'Year'
9 # Compute the ratio of Wind+Solar to Consumption
10 opsd_annual['Wind+Solar/Consumption'] = opsd_annual['Wind+Solar'] / opsd_annual['Consumption']
11 opsd_annual.tail(3)

```

	Consumption	Wind	Solar	Wind+Solar	Wind+Solar/Consumption
Year					
2015	505264.56300	77468.994	34907.138	112376.132	0.222410
2016	505927.35400	77008.126	34562.824	111570.950	0.220528
2017	504736.36939	102667.365	35882.643	138550.008	0.274500

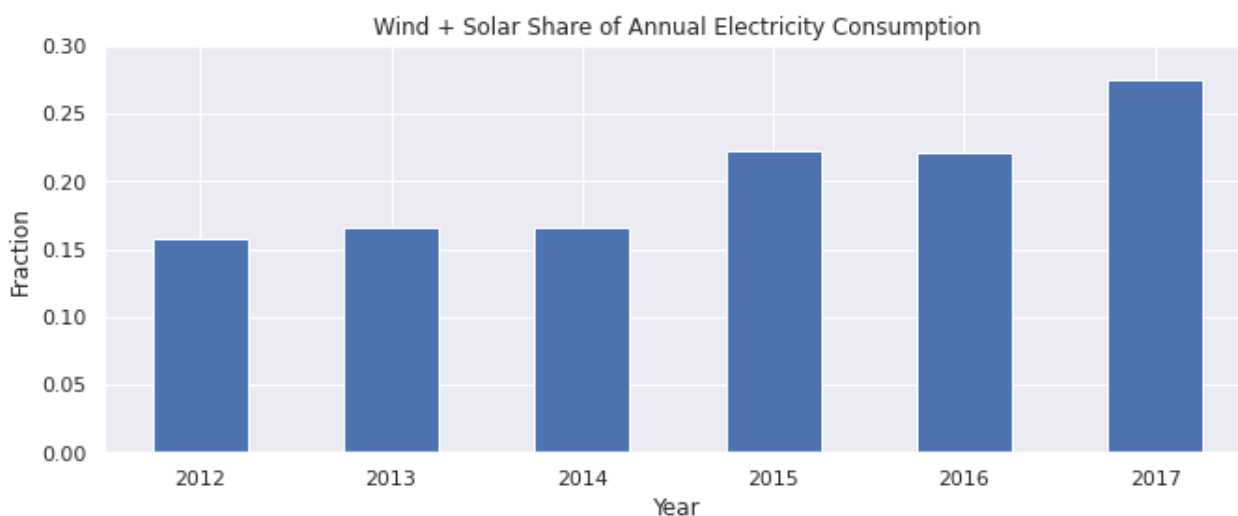
Hình 30: Annual resampling với bộ dữ liệu hiện tại

Ta có thể vẽ biểu đồ hiển thị sản lượng sản xuất năng lượng gió và mặt trời đóng góp vào mức độ tiêu thụ điện năng kể từ năm 2012 như sau:

```

1 # Plot from 2012 onwards, because there is no solar production data in earlier
  years
2 ax = opsd_annual.loc[2012:, 'Wind+Solar/Consumption'].plot.bar(color='C0')
3 ax.set_ylabel('Fraction')
4 ax.set_ylim(0, 0.3)
5 ax.set_title('Wind + Solar Share of Annual Electricity Consumption')
6 plt.xticks(rotation=0)

```



Hình 31: Biểu đồ cột biểu thị **Solar + Wind** đóng góp vào mức tiêu thụ điện năng

- Rolling windows:** Rolling window cũng là một hoạt động chuyển thông tin quan trọng trong dữ liệu time series. Giống downsampling, rolling windows chia dữ liệu thành các time windows (các khoảng thời gian như tuần, tháng... được trượt trên các mẫu dữ liệu theo ngày) và dữ liệu trong mỗi window đó được tổng hợp với hàm mean(), median(), sum(),... Tuy nhiên, không giống như

downsampling, khi mà dữ liệu không overlap nhau và output luôn có tần số thấp hơn input, rolling windows overlap và gom thành những dữ liệu có cùng tần số, vì thế time series được chuyển có cùng tần số với time series gốc. Ta ví dụ với rolling trong 7 ngày:

```
1 # Compute the centered 7-day rolling mean
2 opsd_7d = opsd_daily[data_columns].rolling(7, center=True).mean()
3 opsd_7d.head(10)
```

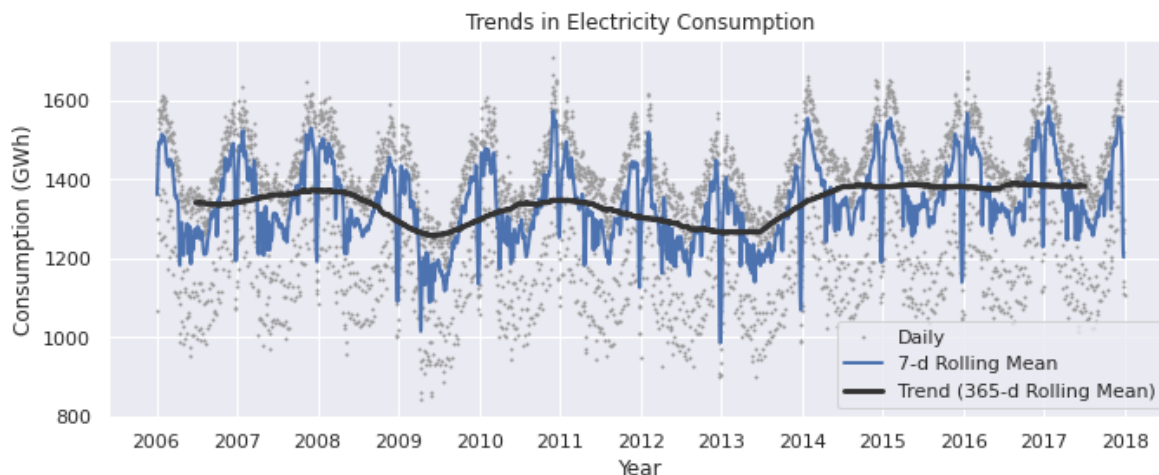
	Consumption	Wind	Solar	Wind+Solar
Date				
2006-01-01	NaN	NaN	NaN	NaN
2006-01-02	NaN	NaN	NaN	NaN
2006-01-03	NaN	NaN	NaN	NaN
2006-01-04	1361.471429	NaN	NaN	NaN
2006-01-05	1381.300143	NaN	NaN	NaN
2006-01-06	1402.557571	NaN	NaN	NaN
2006-01-07	1421.754429	NaN	NaN	NaN
2006-01-08	1438.891429	NaN	NaN	NaN

Hình 32: Rolling windows với chu kỳ 7 ngày

Ở đây, 2006-01-01 đến 2006-01-07 được đánh nhãn là 2006-01-04, 2006-01-02 đến 2006-01-08 được đánh nhãn là 2006-01-05, tương tự cho các dòng khác.

8. **Trends:** Là một đặc trưng chỉ xu hướng của dữ liệu, có thể tăng hoặc giảm đi trong một khoảng thời gian dài. Với kỹ thuật rolling windows, ta có thể dễ dàng trực quan hóa trends của bộ dữ liệu, tại các time scales khác nhau. Ví dụ, ta tính 365-day rolling mean:

```
1 import matplotlib.dates as mdates
2
3 # The min_periods=360 argument accounts for a few isolated missing days in the
4 # wind and solar production time series
5 opsd_365d = opsd_daily[data_columns].rolling(window=365, center=True, min_periods
6         =360).mean()
7
8 # Plot daily, 7-day rolling mean, and 365-day rolling mean time series
9 fig, ax = plt.subplots()
10 ax.plot(opsd_daily['Consumption'], marker='.', markersize=2, color='0.6',
11         linestyle='None', label='Daily')
12 ax.plot(opsd_7d['Consumption'], linewidth=2, label='7-d Rolling Mean')
13 ax.plot(opsd_365d['Consumption'], color='0.2', linewidth=3,
14         label='Trend (365-d Rolling Mean)')
15 # Set x-ticks to yearly interval and add legend and labels
16 ax.xaxis.set_major_locator(mdates.YearLocator())
17 ax.legend()
18 ax.set_xlabel('Year')
19 ax.set_ylabel('Consumption (GWh)')
20 ax.set_title('Trends in Electricity Consumption')
21 plt.show()
```

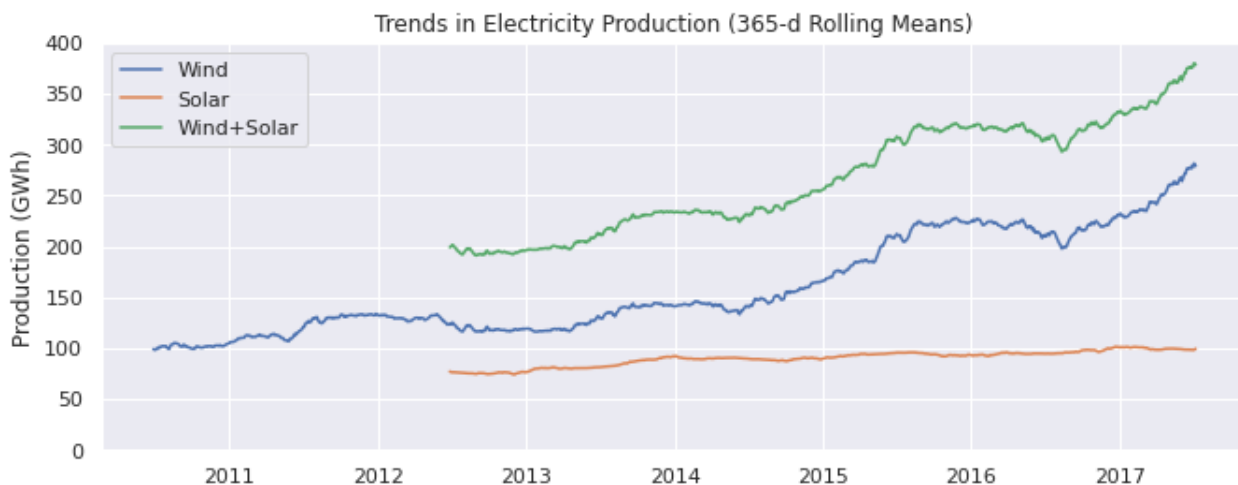


Hình 33: Xu hướng tiêu thụ điện, tuần và năm, tăng mạnh vào cuối năm

```

1 # Plot 365-day rolling mean time series of wind and solar power
2 fig, ax = plt.subplots()
3 for nm in ['Wind', 'Solar', 'Wind+Solar']:
4     ax.plot(opsd_365d[nm], label=nm)
5     # Set x-ticks to yearly interval, adjust y-axis limits, add legend and labels
6     ax.xaxis.set_major_locator(mdates.YearLocator())
7     ax.set_ylim(0, 400)
8     ax.legend()
9     ax.set_ylabel('Production (GWh)')
10    ax.set_title('Trends in Electricity Production (365-d Rolling Means)')
11 plt.show()

```



Hình 34: Xu hướng sản xuất năng lượng điện gió và mặt trời có xu hướng tăng qua hằng năm, đặc biệt là năng lượng gió

Như vậy với một số bước thực hiện trên, ta đã xem qua cách sắp xếp, phân tích và trực quan hóa dữ liệu time series data trong pandas, dùng các kỹ thuật như time-based indexing, resampling, rolling windows. Áp dụng kỹ thuật này vào bộ dataset OPSD, thu được các thông tin chi tiết về thời điểm, các kỳ, và xu hướng trong sản xuất và tiêu thụ điện.

Phần III: Câu hỏi

A. Phần trắc nghiệm

1. Data Analysis là gì?

- (a) Quá trình thu thập dữ liệu. (c) Quá trình xử lý dữ liệu.
(b) Quá trình tìm kiếm và khai thác dữ liệu. (d) Các phương án trên đều đúng.

2. Cấu trúc dữ liệu nào sau đây không thuộc pandas:

- (a) Series (c) Panel
(b) DataFrame (d) Tensor

3. Ý nghĩa của phương thức head() đối với bảng dữ liệu trong pandas là:

- (a) Hiển thị các hàng cuối cùng (c) Hiển thị ngẫu nhiên một số hàng
(b) Hiển thị các hàng đầu tiên (d) Hiển thị tất cả các hàng

4. Ý nghĩa của phương thức describe() đối với bảng dữ liệu trong pandas là:

- (a) Bảng thống kê của các cột dữ liệu string (c) Bảng thống kê của các cột dữ liệu list
(b) Bảng thống kê của các cột dữ liệu số (d) Bảng thống kê của các cột dữ liệu dict

5. Phương thức nào sau đây được dùng để đọc một file .csv từ bộ nhớ trong pandas?

- (a) pd.load_csv() (c) pd.read_file()
(b) pd.read_csv() (d) pd.load_file()

6. Ý nghĩa của phương thức groupby() đối với bảng dữ liệu trong pandas là:

- (a) Lọc các hàng theo điều kiện (c) Nối các bảng dữ liệu
(b) Tổng hợp thống kê các cột dữ liệu (d) Gom nhóm dữ liệu theo giá trị của một hoặc nhiều cột

7. Phương thức nào sau đây dùng để kiểm tra các giá trị NaN có trong bảng dữ liệu?

- (a) df.isna() (c) df.notnull()
(b) df.notna() (d) df.tail()

8. Phương thức nào sau đây có thể được dùng để bỏ đi một hàng có giá trị null trong bảng dữ liệu?

- (a) df.drop_null() (c) df.dropna()
(b) df.drop_missing() (d) df.remove_null()

9. Phương thức nào sau đây trong pandas được dùng để fill các giá trị bị mất trong bảng dữ liệu sử dụng kỹ thuật forward filling?

- (a) `fillna(method='bfill')` (c) `fillna(method='ffill')`
 (b) `fillna(method='pad')` (d) `fillna(method='forward')`
10. Phương thức nào sau đây trong pandas được dùng để resample dữ liệu?
 (a) `resample()` (c) `reduce()`
 (b) `downsample()` (d) `shrink()`
11. Phương thức nào sau đây trong pandas được dùng để tính rolling windows?
 (a) `rolling()` (c) `average()`
 (b) `mean()` (d) `smooth()`
12. Phương thức nào sau đây trong pandas được dùng thực thi một hàm bất kì vào tất cả phần tử trong một Series?
 (a) `pd.Series.transform()` (c) `pd.Series.map()`
 (b) `pd.Series.applymap()` (d) `pd.Series.apply()`
13. Phương thức nào sau đây có thể được dùng để lấy toàn bộ một cột sử dụng tên của nó từ bảng dữ liệu?
 (a) `df[col]` (c) `df.ix[col]`
 (b) `df.loc[col]` (d) `df.iloc[col]`

Xem qua bảng dữ liệu sau đây (df) và trả lời các câu hỏi dưới đây:

Date	Open	High	Low	Close	Volume	Adj Close
6/29/2010	19.000000	25.000000	17.540001	23.889999	18766300	23.889999
6/30/2010	25.790001	30.420000	23.299999	23.830000	17187100	23.830000
7/1/2010	25.000000	25.920000	20.270000	21.959999	8218800	21.959999
7/2/2010	23.000000	23.100000	18.709999	19.200001	5139800	19.200001
7/6/2010	20.000000	20.000000	15.830000	16.110001	6866900	16.110001

14. Dòng lệnh nào sau đây dùng để chọn các hàng có giá trị 'Close' lớn hơn 25?
 (a) `df[df['Close'] > 25]` (c) `df.iloc[df['Close'] > 25]`
 (b) `df['Close'] > 25` (d) `df[df.Close > 25]`
15. Dòng lệnh nào sau đây dùng để chọn các hàng có giá trị 'Volume' nhỏ hơn hoặc bằng 1000000?
 (a) `df.iloc[df['Volume'] <= 1000000]` (c) `df[df.Volume <= 1000000]`
 (b) `df[df['Volume'] <= 1000000]` (d) `df['Volume'] <= 1000000`
16. Dòng lệnh nào sau đây dùng để chọn các hàng có giá trị 'High' nhỏ hơn hoặc bằng 'Low'?

(a) `df.iloc[df['High'] <= df['Low']]`

(c) `df[df.High <= df.Low]`

(b) `df[df['High'] <= df['Low']]`

(d) `df['High'] <= df['Low']`

17. Dòng lệnh nào sau đây dùng để tìm giá trị trung bình của cột 'Close'?

(a) `df.mean()`

(c) `df['Close'].sum()`

(b) `df['Close'].mean`

(d) `df['Close'].mean()`

- *Hết* -