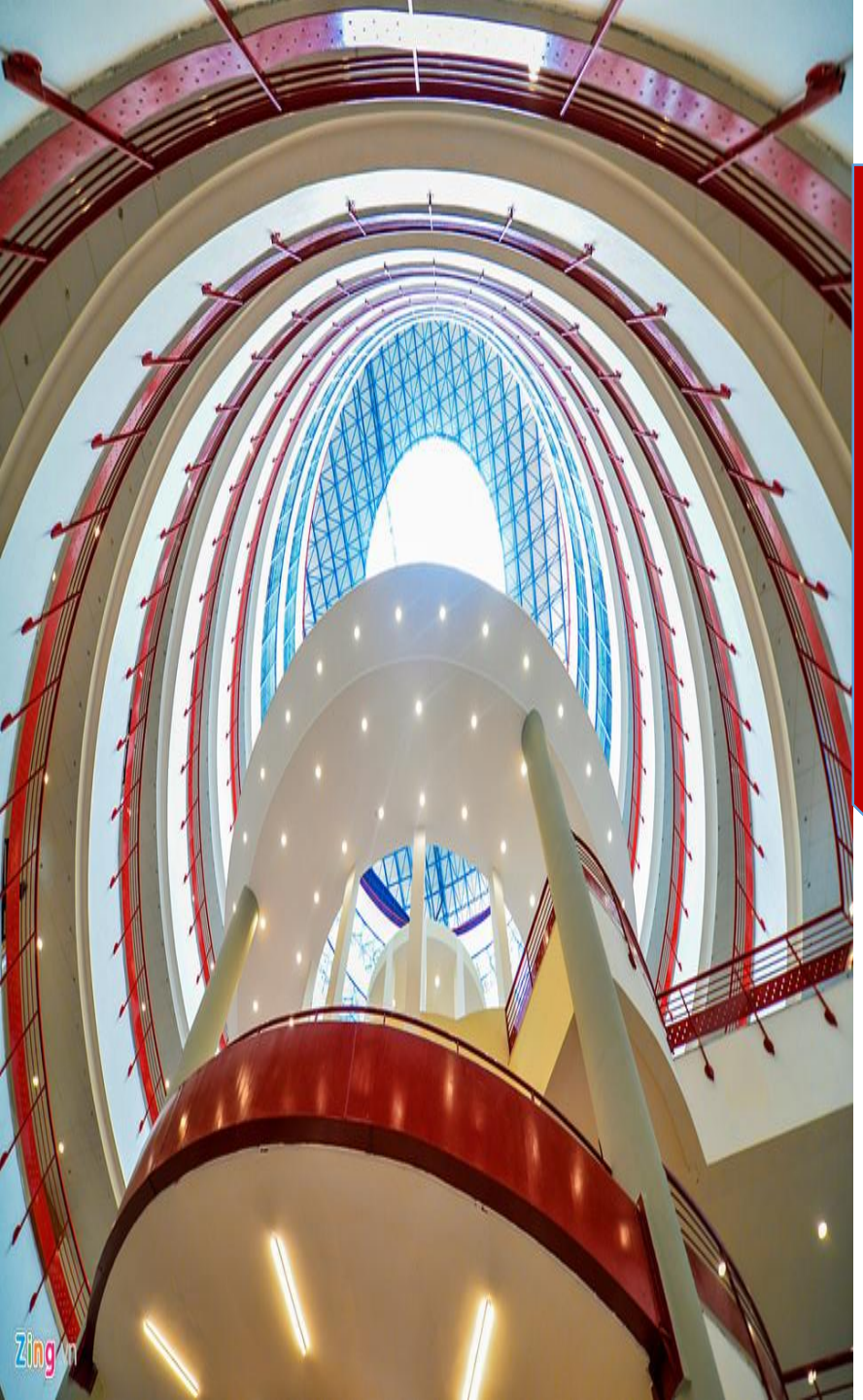

DATA PREPARATION AND VISUALIZATION

Mathematical Economics Faculty

National Economics University
https://www.neu.edu.vn/

Chapter 6: Working with Date and Time in Python

Working with Dates and Times in Python

Python has three standard modules for working with dates and times:

- The calendar module
- The time module
- The datetime module

The most useful module for working with data is the datetime module:

- datetime.datetime: for working with date and time data
- datetime.time: for working with the time data only
- datetime.timedelta: for representing time periods

Working with Dates and Times in Python

Import the whole module by name

```
# import the datetime module
import datetime

# use the datetime class
my_datetime_object = datetime.datetime()
```

The datetime module — The datetime class

- *Pro:* it's clear whenever you use `datetime` whether you're referring to the module or the class
- *Con:* it has the potential to create long lines of code, which can be more difficult to read

Working with Dates and Times in Python

Import definitions via name or wildcard

```
# import the datetime class by name
from datetime import datetime

# import all definitions using wildcard
from datetime import *

# use the datetime class
my_datetime_object = datetime()
```

- *Pro:* shorter lines of code, which are easier to read
- *Con:* when we use `datetime`, it's not clear whether we are referring to the module or the class

Working with Dates and Times in Python

Import whole module by alias

```
# import the datetime module
import datetime as dt

# use the datetime class
my_datetime_object = dt.datetime()
```

- *Pro:* there is no ambiguity between `dt` (alias for the module) and `dt.datetime` (the class).
- *Con:* the `dt` alias isn't a common convention, which would confuse other people reading our code

In the end, even though using an alias for the datetime module (the third option) is less common, it's a nice compromise between avoiding module versus class confusion, and it keeps our code easy to read. Let's use that technique to import the datetime module.

Working with Dates and Times in Python

The datetime class has a number of attributes that simplify retrieving the various parts that make up the date stored within the object:

- datetime.day: the day of the month
- datetime.month: the month of the year
- datetime.year: the year
- datetime.hour: the hour of the day
- datetime.minute: the minute of the hour

If we wanted to create string representation of a datetime object representing the date like *December 24th, 1984* in the form *day/month/year*, we could use those attributes to extract the values and then insert them into a string:

Working with Dates and Times in Python

TIMEDELTA

Converting Between String and Datetime

- You can format *datetime* objects and *pandas Timestamp objects as string* using str or the strftime method
  - Ex:

```
In [22]: stamp = datetime(2011, 1, 3)

In [23]: str(stamp)
Out[23]: '2011-01-03 00:00:00'

In [24]: stamp.strftime('%Y-%m-%d')
Out[24]: '2011-01-03'
```

Converting Between String and Datetime(I)

Datetime format specification (ISO C89 compatible)

Type	Description
%Y	Four-digit year
%y	Two-digit year
%m	Two-digit month [01, 12]
%d	Two-digit day [01, 31]
%H	Hour (24-hour clock) [00, 23]
%I	Hour (12-hour clock) [01, 12]
%M	Two-digit minute [00, 59]
%S	Second [00, 61] (seconds 60, 61 account for leap seconds)
%w	Weekday as integer [0 (Sunday), 6]

Type	Description
%U	Week number of the year [00, 53]; Sunday is considered the first day of the week, and days before the first Sunday of the year are "week 0"
%W	Week number of the year [00, 53]; Monday is considered the first day of the week, and days before the first Monday of the year are "week 0"
%z	UTC time zone offset as +HHMM or -HHMM; empty if time zone naive
%F	Shortcut for %Y-%m-%d (e.g., 2012-4-18)
%D	Shortcut for %m/%d/%y (e.g., 04/18/12)

Converting Between String and Datetime(II)

- You can use these same format codes to *convert strings to dates* using *datetime.strptime*
  - Ex:

```
In [25]: value = '2011-01-03'

In [26]: datetime.strptime(value, '%Y-%m-%d')
Out[26]: datetime.datetime(2011, 1, 3, 0, 0)

In [27]: datestrs = ['7/6/2011', '8/6/2011']

In [28]: [datetime.strptime(x, '%m/%d/%Y') for x in datestrs]
Out[28]:
[datetime.datetime(2011, 7, 6, 0, 0),
 datetime.datetime(2011, 8, 6, 0, 0)]
```

Converting Between String and Datetime(III)

- You can use these same format codes to *convert strings to dates* using *datetime.strptime*
  - Ex:

```
In [25]: value = '2011-01-03'

In [26]: datetime.strptime(value, '%Y-%m-%d')
Out[26]: datetime.datetime(2011, 1, 3, 0, 0)

In [27]: datestrs = ['7/6/2011', '8/6/2011']

In [28]: [datetime.strptime(x, '%m/%d/%Y') for x in datestrs]
Out[28]:
[datetime.datetime(2011, 7, 6, 0, 0),
 datetime.datetime(2011, 8, 6, 0, 0)]
```

Converting Between String and Datetime(III)

- *Datetime.strptime* is used to parse a date with a known format
- The *to_datetime method* parses many different kinds of date representations
  - ex

```
Entrée [66]: birthdays
```

Out[66]:

	name	adress	birthday
0	An	HN	28.3.2019
1	Hang	QN	5/12/2020
2	Nga	BT	25-Sep-19
3	Anh	TB	3-Jan-19
4	Phuong	ND	03.08.2019

```
Entrée [67]: birthdays['birthday'] = pd.to_datetime(birthdays['birthday'])
```

```
Entrée [63]: birthdays
```

```
Out[63]: 0   2019-03-28
         1   2020-05-12
         2   2019-09-25
         3   2019-01-03
         4   2019-03-08
         Name: birthday, dtype: datetime64[ns]
```