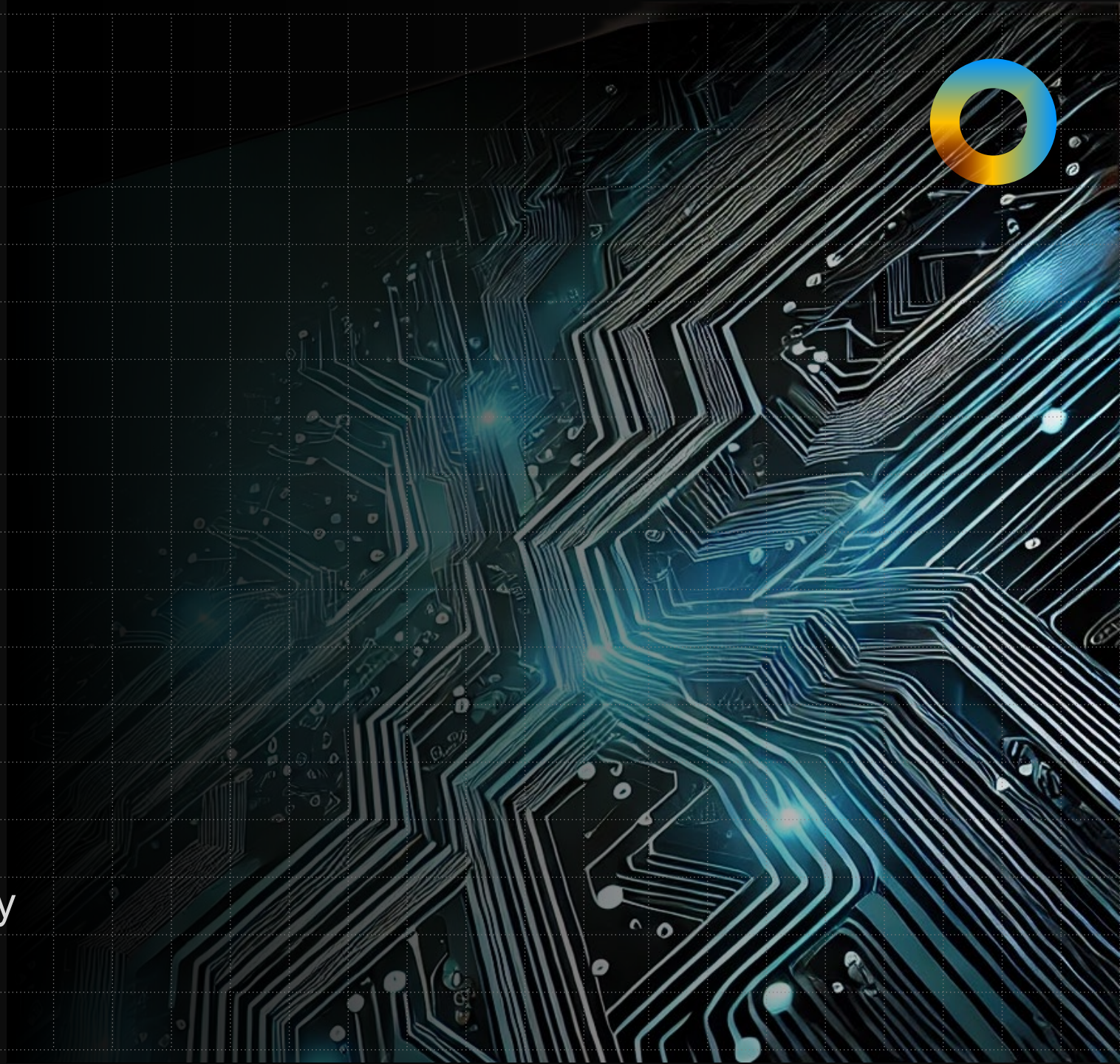
Python Programming

Pandas

Dr. Chun-Hsiang Chan
Department of Geography
National Taiwan Normal University



Outlines

- Make a DataFrame
- Indexing
- Merge & Concatenation
- Groupby & Drop Duplicated
- Data I/O
- IsNA, Fillna & Dropna
- Pandas.Series.str
- Datetime Formation
- Datetime Analysis

Pandas – Make a DataFrame

- Pandas is another powerful Python library for data cleaning or data preprocessing, even for data analysis.
- Basically, you can imagine pandas as excel or table-like data (panel data).

```
# import package
import pandas as pd
# declare dict
ds = {'name':['mike','amy','may'], 'score':[70, 80, 23]}
df = pd.DataFrame.from_dict(ds) # use dict
df.head()
```

	name	score
0	mike	70
1	amy	80
2	may	23

Pandas – Make a DataFrame

- In addition to dictionary, we may also use list to construct a Pandas dataframe.

```
# declare list
ls = [['mike','amy','may','joy','mai','zoy','hoy'], [70, 80, 23 , 50, 72 ,
        82, 73]]
df2 = pd.DataFrame.from_records(np.column_stack(ls),
        columns=['name','score']) # use list
# how many rows show in the following code?
df2.head()
# try this
df2.head(7)
```

	name	score
0	mike	70
1	amy	80
2	may	23
3	joy	50
4	mai	72

	name	score
0	mike	70
1	amy	80
2	may	23
3	joy	50
4	mai	72
5	zoy	82
6	hoy	73

Pandas – Indexing

- Get to know the data type and indexing of pandas dataframe.

```
# data type
print(type(df2))
<class 'pandas.core.frame.DataFrame'>
print(type(df2['name']))
<class 'pandas.core.series.Series'>
# indexing
print(df2['name'][2])
 may
```

```
# use iloc
df2.iloc[:2, :4]
```

	name	score
0	mike	70
1	amy	80

```
# use loc
df2.loc[1:4,'score']
```

```
1    80
2    23
3    50
Name: score, dtype: object
```

Pandas – Indexing

- Get to know the data type and indexing of pandas dataframe.

```
# ask data row if it has may and mike
df2['name'].isin(['may','mike'])
```

```
0     True
1    False
2     True
3    False
4    False
5    False
6    False
Name: name, dtype: bool
```

```
# get data based on a condition
df2[df2['name'].isin(['may','mike'])]
```

	name	score
0	mike	70
2	may	23

```
# get the data type of elements
print(type(df2['score'][0]))
```

```
<class 'numpy.str_'>
```

Pandas – Indexing

- Get to know the data type and indexing of pandas dataframe.

```
# change data type
df2['score'] = df2['score'].astype(int)
df2['score']
```

```
0    70
1    80
2    23
3    50
4    72
5    82
6    73
Name: score, dtype: int64
```

```
# get column values
df2['score'].values
```

```
array(['70', '80', '23', '50', '72', '82',
'73'], dtype=object)
```

```
# use two or more conditions
df2.loc[(df2['score']>60) &
(df2['score']<80)]
```

	name	score
0	mike	70
4	mai	72
6	hoy	73

Pandas – Reset Index

```
# use two or more conditions and reset index
df2.loc[(df2['score']>60) & (df2['score']<80)].reset_index()
```

	index	name	score
0	0	mike	70
1	4	mai	72
2	6	hoy	73

```
# use two or more conditions and reset index
df2.loc[(df2['score']>60) & (df2['score']<80)].reset_index(drop=True)
```

	name	score
0	mike	70
1	mai	72
2	hoy	73

```
# what is the difference between the abovementioned codes?
```

Pandas – Reset Index

use two or more conditions and reset index

```
df2.loc[(df2['score']>60) | (df2['score']<80)].reset_index(drop=True)
```

	name	score
0	mike	2
1	amy	3
2	may	0
3	joy	4
4	mai	6
5	zoy	1
6	hoy	5

	name	score
0	mike	70
1	amy	80
2	may	23
3	joy	50
4	mai	72
5	zoy	82
6	hoy	73

why do we need **drop=False**?

Pandas – Sorting

- Sorting in Pandas dataframe

```
# sorting by column value
# get sorted value
df2['score'].sort_values()

# get sorted index
df2['score'] = df2['score'].sort_values().index
# get sorted index for re-order dataframe
df2.iloc[df2['score'].sort_values().index]
# how to re-index the sorted dataframe?
```

```
2    23
3    50
0    70
4    72
6    73
1    80
5    82
Name: score, dtype: int64
```

	name	score
2	may	0
5	zoy	1
0	mike	2
1	amy	3
3	joy	4
6	hoy	5
4	mai	6

Pandas – Merge & Concatenation

- What is the difference between merge and concat?

```
df1 = pd.DataFrame({
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    },index=[0, 1, 2, 3],)

df2 = pd.DataFrame({
        "B": ["B2", "B3", "B6", "B7"],
        "D": ["D2", "D3", "D6", "D7"],
        "F": ["F2", "F3", "F6", "F7"],
    },index=[0, 1, 2, 3],)
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	B	D	F
0	B2	D2	F2
1	B3	D3	F3
2	B6	D6	F6
3	B7	D7	F7

Pandas – Merge & Concatenation

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	B	D	F
0	B2	D2	F2
1	B3	D3	F3
2	B6	D6	F6
3	B7	D7	F7

```
df1.merge(left_on='B', right=df2, right_on='B', how='left')
```

left

	A	B	C	D_x	D_y	F
0	A0	B0	C0	D0	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN
2	A2	B2	C2	D2	D2	F2
3	A3	B3	C3	D3	D3	F3

inner

	A	B	C	D_x	D_y	F
0	A2	B2	C2	D2	D2	F2
1	A3	B3	C3	D3	D3	F3

right

	A	B	C	D_x	D_y	F
0	A2	B2	C2	D2	D2	F2
1	A3	B3	C3	D3	D3	F3
2	NaN	B6	NaN	NaN	D6	F6
3	NaN	B7	NaN	NaN	D7	F7

outer

	A	B	C	D_x	D_y	F
0	A0	B0	C0	D0	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN
2	A2	B2	C2	D2	D2	F2
3	A3	B3	C3	D3	D3	F3
4	NaN	B6	NaN	NaN	D6	F6
5	NaN	B7	NaN	NaN	D7	F7

Pandas – Merge & Concatenation

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	B	D	F
0	B2	D2	F2
1	B3	D3	F3
2	B6	D6	F6
3	B7	D7	F7

`pd.concat([df1, df2], axis=0)`

	A	B	C	D	F
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2	NaN
3	A3	B3	C3	D3	NaN
0	NaN	B2	NaN	D2	F2
1	NaN	B3	NaN	D3	F3
2	NaN	B6	NaN	D6	F6
3	NaN	B7	NaN	D7	F7

`pd.concat([df1, df2], axis=1)`

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	B2	D2	F2
1	A1	B1	C1	D1	B3	D3	F3
2	A2	B2	C2	D2	B6	D6	F6
3	A3	B3	C3	D3	B7	D7	F7

Pandas – Groupby & Drop Duplicated

- Group by and drop duplicated values

```
ls = [['mike','amy','may','joy','mai','zoy','hoy','mai','zoy','hoy'], [70, 80, 23
, 50, 72 , 82, 73 , 80, 100, 45]]
df2 = pd.DataFrame.from_records(np.column_stack(ls),
          columns=['name','score'])
# drop duplicates
df2.drop_duplicates(['name'])

# groupby with count, mean, median, std, sum, …
df2.groupby(['name']).count().reset_index()
# try and observe
```

	name	score
0	mike	70
1	amy	80
2	may	23
3	joy	50
4	mai	72
5	zoy	82
6	hoy	73

	name	score
0	amy	1
1	hoy	2
2	joy	1
3	mai	2
4	may	1
5	mike	1
6	zoy	2

Pandas – Data I/O

- In most scenarios, we have to load the external dataset from various file formats. Hopefully, Pandas proffers several common file formats, such as **csv**, **xlsx**, and **xml**.

csv	**json**	**xml**	**html**
excel	**sas**	**spss**	**stata**
sql	**hdl**	**parquet**	**feather**
orc	**pickle**	**fwf**	**gbq**

Pandas – Data I/O

- Here, we use the most commonly used file format – **CSV**.
- **pandas.read_csv**(***filepath_or_buffer***, *, ***sep=<no_default>***, ***delimiter=None***, ***header='infer'***, *names=<no_default>*, *index_col=None*, *usecols=None*, *dtype=None*, *engine=None*, *converters=None*, *true_values=None*, *false_values=None*, *skipinitialspace=False*, ***skiprows=None***, *skipfooter=0*, *nrows=None*, *na_values=None*, *keep_default_na=True*, *na_filter=True*, *verbose=<no_default>*, *skip_blank_lines=True*, *parse_dates=None*, *infer_datetime_format=<no_default>*, *keep_date_col=<no_default>*, *date_parser=<no_default>*, ***date_format=None***, *dayfirst=False*, *cache_dates=True*, *iterator=False*, *chunksize=None*, *compression='infer'*, *thousands=None*, *decimal='.'*, *lineterminator=None*, *quotechar='"'*, *quoting=0*, *doublequote=True*, *escapechar=None*, *comment=None*, *encoding=None*, *encoding_errors='strict'*, *dialect=None*, *on_bad_lines='error'*, *delim_whitespace=<no_default>*, *low_memory=True*, *memory_map=False*, *float_precision=None*, *storage_options=None*, *dtype_backend=<no_default>*)

Pandas – Data I/O

Data https://catalog.data.gov/dataset/nypd-arrest-data-year-to-date

/ / City of New York / data.cityofnewyork.us Feedback

City

City of New York

There is no description for this organization

Topics

Local Government

Publisher

data.cityofnewyork.us

Contact

NYC OpenData

Share on Social Sites

Twitter

Facebook

This is a Non-Federal dataset covered by different Terms of Use than Data.gov. See Terms

NYPD Arrest Data (Year to Date)

Metadata Updated: October 25, 2024

This is a breakdown of every arrest effected in NYC by the NYPD during the current year. This data is manually extracted every quarter and reviewed by the Office of Management Analysis and Planning. Each record represents an arrest effected in NYC by the NYPD and includes information about the type of crime, the location and time of enforcement. In addition, information related to suspect demographics is also included. This data can be used by the public to explore the nature of police enforcement activity. Please refer to the attached data footnotes for additional information about this dataset.

Access & Use Information

Public: This dataset is intended for public access and use.
Non-Federal: This dataset is covered by different Terms of Use than Data.gov. See Terms
License: No license information was provided.

Downloads & Resources

Comma Separated Values File 184 views — Download

RDF File 19 views — Download

Data Spec https://data.cityofnewyork.us/Public-Safety/NYPD-Arrest-Data-Year-to-Date-/uip8-fykc/about_data

Columns (19)

Column Name	Description	API Field Name	Data Type
ARREST_KEY	Randomly generated persistent ID for each arrest	arrest_key	Text
ARREST_DATE	Exact date of arrest for the reported event	arrest_date	Floating Timestamp
PD_CD	Three digit internal classification code (more granular than Key Code)	pd_cd	Number
PD_DESC	Description of internal classification corresponding with PD code (more granular than Offense Description)	pd_desc	Text
KY_CD	Three digit internal classification code (more general category than PD code)	ky_cd	Number
OFNS_DESC	Description of internal classification corresponding with KY code (more general category than PD description)	ofns_desc	Text
LAW_CODE	Law code charges corresponding to the NYS Penal Law, VTL and other various local laws	law_code	Text
LAW_CAT_CD	Level of offense: felony, misdemeanor, violation	law_cat_cd	Text
ARREST_BORO	Borough of arrest. B(Bronx), S(Staten Island), K(Brooklyn), M(Manhattan), Q(Queens)	arrest_boro	Text
ARREST_PRECINCT	Precinct where the arrest occurred	arrest_precinct	Number
JURISDICTION_CODE	Jurisdiction responsible for arrest. Jurisdiction codes 0(Patrol), 1(Transit) and 2(Housing) represent NYPD whilst codes 3 and more represent non NYPD jurisdictions	jurisdiction_code	Number

Pandas – Data I/O

Source: https://catalog.data.gov/dataset/nypd-arrest-data-year-to-date

- Read the file first and use **.head()** or **.tail()** to preview the content.

```
df = pd.read_csv('NYPD_Arrest_Data__Year_to_Date_.csv')
# num is the number of lines you want to preview; default is 5
df.head() # you may try df.tail()
```

	ARREST_KEY	ARREST_DATE	PD_CD	PD_DESC	KY_CD	OFNS_DESC	LAW_CODE	LAW_CAT_CD	ARREST_BORO	ARREST_PRECINCT	JURISDICT
0	281369711	01/30/2024	177.0	SEXUAL ABUSE	116.0	SEX CRIMES	PL 1306501	F	M	25	
1	284561406	03/30/2024	105.0	STRANGULATION 1ST	106.0	FELONY ASSAULT	PL 1211200	F	B	44	
2	284896016	04/06/2024	105.0	STRANGULATION 1ST	106.0	FELONY ASSAULT	PL 1211200	F	M	19	
3	285569016	04/18/2024	105.0	STRANGULATION 1ST	106.0	FELONY ASSAULT	PL 1211200	F	K	69	
4	287308954	05/22/2024	464.0	JOSTLING	230.0	JOSTLING	PL 1652501	M	M	18	

Pandas – Data I/O

After a series of preprocessing or analysis, you need to export the DataFrame to a specific file format.

```
# to csv
df.to_csv('data.csv', index=False)
# to excel
df.to_excel('data.xlsx', index=False)
# to json
df.to_json('data.json', index=False)
```

	A	B	C
1	ARREST_KEY	ARREST_DAT	PD_CD
2	281369711	01/30/2024	177
3	284561406	03/30/2024	105
4	284896016	04/06/2024	105
5	285569016	04/18/2024	105
6	287308954	05/22/2024	464
7	286793332	05/13/2024	155
8	279892607	01/03/2024	153
9	280263905	01/10/2024	157
10	288072319	06/06/2024	808
11	288408753	06/12/2024	105
12	280676776	01/17/2024	263
13	280733020	01/18/2024	153
14	280751787	01/18/2024	155
15	288638726	06/17/2024	105
16	288681123	06/18/2024	153
17	281035905	01/24/2024	777
18	281223962	01/27/2024	105

	A	B	C
1		ARREST_KEY	ARREST_DAT
2	0	281369711	01/30/2024
3	1	284561406	03/30/2024
4	2	284896016	04/06/2024
5	3	285569016	04/18/2024
6	4	287308954	05/22/2024
7	5	286793332	05/13/2024
8	6	279892607	01/03/2024
9	7	280263905	01/10/2024
10	8	288072319	06/06/2024
11	9	288408753	06/12/2024
12	10	280676776	01/17/2024
13	11	280733020	01/18/2024
14	12	280751787	01/18/2024
15	13	288638726	06/17/2024
16	14	288681123	06/18/2024
17	15	281035905	01/24/2024
18	16	281223962	01/27/2024

IsNA, FillNa & Dropna

- Most datasets have many NA values with different styles (i.e., -9999, -9998, -9997, ' ', and NaN).
- We detect NA values for each column and think about the preprocessing method for NA values.
  - Fill NA values with specified values
  - Direct NA values

 # Why do we need to preprocess NA values?

IsNA, FillNa & Dropna

- NA Detection

```
pd.isna(df)
```

	ARREST_KEY	ARREST_DATE	PD_CD	PD_DESC	KY_CD	OFNS_DESC	LAW_CODE	LAW_CAT_CD	ARREST_BORO	ARREST_PRECINCT	JURISDICTIC
0	False	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	
260498	False	False	False	False	False	False	False	False	False	False	
260499	False	False	False	False	False	False	False	False	False	False	
260500	False	False	False	False	False	False	False	False	False	False	
260501	False	False	False	False	False	False	False	False	False	False	
260502	False	False	False	False	False	False	False	False	False	False	

260503 rows × 19 columns

IsNA, FillNa & Dropna

- Count the number of NA values

```
pd.isna(df)
```

- Investigate the column of LAW_CAT_CD

```
df['LAW_CAT_CD'].drop_duplicates()
```

```
0              F
4              M
37             9
102            V
232          NaN
769            I
11314    (null)
Name: LAW_CAT_CD, dtype: object
```

Level of offense: felony, misdemeanor, violation

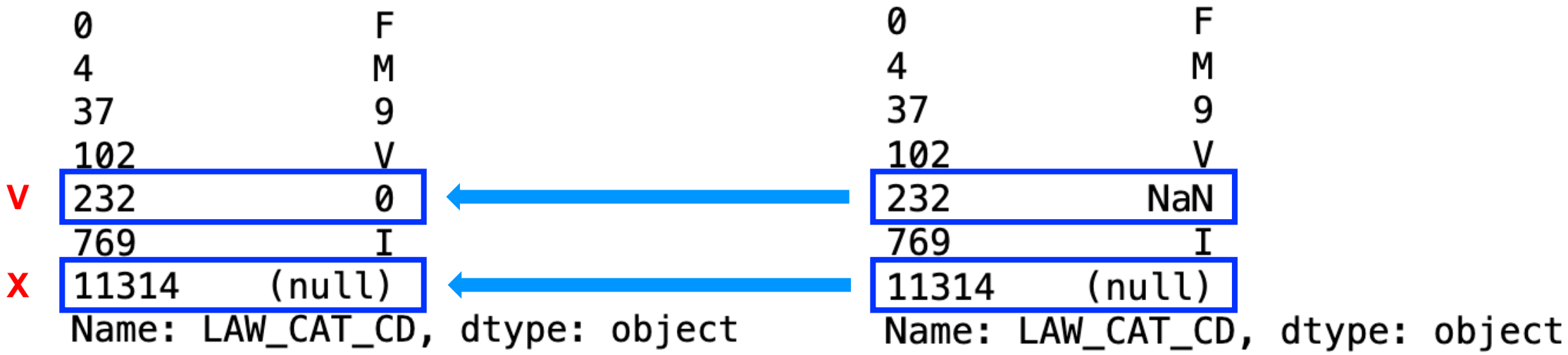
```
ARREST_KEY                    0
ARREST_DATE                   0
PD_CD                         8
PD_DESC                       0
KY_CD                        32
OFNS_DESC                     0
LAW_CODE                      0
LAW_CAT_CD                 1390
ARREST_BORO                   0
ARREST_PRECINCT               0
JURISDICTION_CODE             0
AGE_GROUP                     0
PERP_SEX                      0
PERP_RACE                     0
X_COORD_CD                    0
Y_COORD_CD                    0
Latitude                      4
Longitude                     4
New Georeferenced Column      4
dtype: int64
```

Source: https://data.cityofnewyork.us/Public-Safety/NYPD-Arrest-Data-Year-to-Date-/uip8-fykc/about_data

IsNA, FillNa & Dropna

- We fill all NaN/ NA with 0.

```
df['LAW_CAT_CD'] = df['LAW_CAT_CD'].fillna('0')
df['LAW_CAT_CD'].drop_duplicates()
```



IsNA, FillNa & Dropna

- Drop all NA in the columns of PD_CD, KY_CD, Latitude, Longitude, and New Georeferenced Column.

```
# drop all NAs
df.dropna()
# if you only want to drop all NAs of the PD_CD column
# how will you do?
```

	ARREST_KEY	ARREST_DATE	PD_CD	PD_DESC	KY_CD	OFNS_DESC	LAW_CODE	LAW_CAT_CD	ARREST_BORO	ARREST_PRECINCT
0	281369711	2024-01-30	177.0	SEXUAL ABUSE	116.0	SEX CRIMES	PL 1306501	F	M	25
1	284561406	2024-03-30	105.0	STRANGULATION 1ST	106.0	FELONY ASSAULT	PL 1211200	F	B	44
...	...	...	...	...	...	...	...	...	...	..
260502	298548871	2024-12-27	681.0	CHILD, ENDANGERING WELFARE	233.0	SEX CRIMES	PL 2601001	M	B	47

260467 rows × 19 columns

Pandas.Series.str

- Until now, we cannot use Fillna or Dropna to deal with the strange NA format, i.e., '**(null)**'.
- Here, we introduce a series of Pandas built-in functions for string data processing.

Source: https://pandas.pydata.org/docs/reference/api/pandas.Series.str.capitalize.html

Pandas.Series.str.replace

- Now, we adopt **replace** to replace all '**(null)**' with '**0**'.

	ARREST_KEY	ARREST_DATE	PD_CD	PD_DESC	KY_CD	OFNS_DESC	LAW_CODE	LAW_CAT_CD	ARREST_BORO	ARREST_PRECINCT	JURISDICTIO
11314	283941614	03/18/2024	NaN	(null)	NaN	(null)	(null)	(null)	K	67	
26225	288234486	06/09/2024	NaN	(null)	NaN	(null)	(null)	(null)	Q	110	
64006	280961365	01/23/2024	NaN	(null)	NaN	(null)	(null)	(null)	M	5	
89850	291765618	08/15/2024	NaN	(null)	NaN	(null)	(null)	(null)	M	14	
152599	288286007	06/10/2024	NaN	(null)	NaN	(null)	(null)	(null)	M	18	
173706	296636818	11/17/2024	NaN	(null)	NaN	(null)	(null)	(null)	B	40	
182936	290067051	07/14/2024	NaN	(null)	NaN	(null)	(null)	(null)	Q	115	
249025	297743412	12/09/2024	NaN	(null)	NaN	(null)	(null)	(null)	M	5	

Pandas.Series.str.replace

- Now, we adopt **replace** to replace all '**(null)**' with '**0**'.

```
# replace '(null)' with '0'
df['LAW_CAT_CD'] = df['LAW_CAT_CD'].str.replace('(null)', '0')
# drop duplicated values
df['LAW_CAT_CD'].drop_duplicates()
```

```
0       F
4       M
37      9
102     V
232     0
769     I
Name: LAW_CAT_CD, dtype: object
```

```
0           F
4           M
37          9
102         V
232         0
769         I
11314    (null)
Name: LAW_CAT_CD, dtype: object
```

Pandas.Series.str – Find Elements

- If you want to find the specific elements in a column, and then you may conduct **find**, **contains**, **startswith**, and **endswith**.

```
df[df['PERP_RACE'].str.contains('BLACK')]['PERP_RACE'].drop_duplicates()
```

```
0              BLACK
5    BLACK HISPANIC
Name: PERP_RACE, dtype: object
```

```
df[df['PERP_RACE'].str.startswith('BLACK')]['PERP_RACE'].drop_duplicates()
```

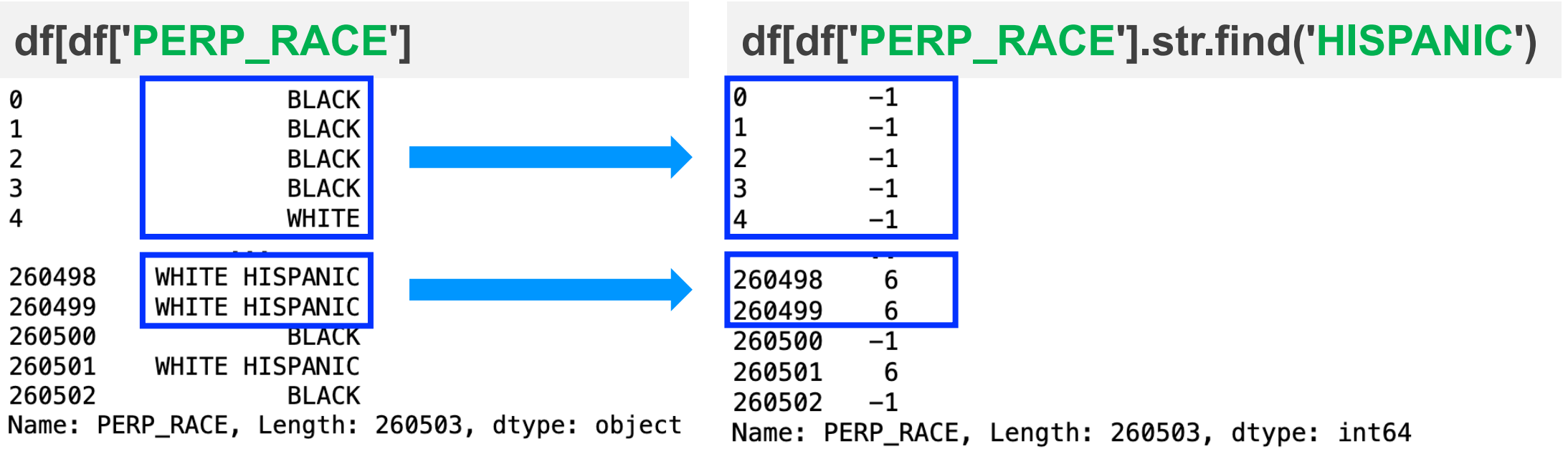
```
0              BLACK
5    BLACK HISPANIC
Name: PERP_RACE, dtype: object
```

```
df[df['PERP_RACE'].str.endswith('HISPANIC')]['PERP_RACE'].drop_duplicates()
```

```
5     BLACK HISPANIC
17    WHITE HISPANIC
Name: PERP_RACE, dtype: object
```

Pandas.Series.str – Find Elements

- Sometimes, the keyword is neither at the beginning nor the end of the text; instead, it may appear anywhere within it. In such cases, using **startswith** or **endswith** alone cannot retrieve all rows containing the specified elements.



Pandas.Series.str – Splitting Texts

- In feature engineering, we usually separate one text into several segments to enrich the number of features. Here, we demonstrate an example of **Pandas.Series.str.split**.

df['ARREST_DATE'].str.split('/')

```
0         [01, 30, 2024]
1         [03, 30, 2024]
2         [04, 06, 2024]
3         [04, 18, 2024]
4         [05, 22, 2024]
              ...
260498    [12, 20, 2024]
260499    [12, 23, 2024]
260500    [12, 30, 2024]
260501    [12, 21, 2024]
260502    [12, 27, 2024]
Name: ARREST_DATE, Length: 260503, dtype: object
```

df['ARREST_DATE'].str.split('/', expand=True)

	0	1	2
0	01	30	2024
1	03	30	2024
2	04	06	2024
...	...	...	...
260502	12	27	2024

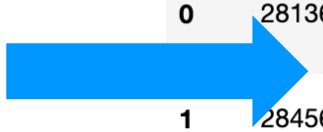
260503 rows × 3 columns

DateTime Formation

- **Datetime** is one of the most important features of data science. We may adopt pd.to_datetime with strftime to convert all datetime format into datetime.datetime format.

```
# convert to datetime
df['ARREST_DATE'] = pd.to_datetime(df['ARREST_DATE'], format='%m/%d/%Y')
# preview dataset
df.head(3)
```

	ARREST_KEY	ARREST_DATE
0	281369711	01/30/2024
1	284561406	03/30/2024
2	284896016	04/06/2024



	ARREST_KEY	ARREST_DATE	PD_CD
0	281369711	2024-01-30	177.0
1	284561406	2024-03-30	105.0
2	284896016	2024-04-06	105.0

```
0         2024-01-30
1         2024-03-30
2         2024-04-06
3         2024-04-18
4         2024-05-22
             ...
260498    2024-12-20
260499    2024-12-23
260500    2024-12-30
260501    2024-12-21
260502    2024-12-27
Name: ARREST_DATE, Length: 260503, dtype: datetime64[ns]
```

DateTime Formation

Directive	Meaning	Example
%a	Abbreviated weekday name.	Sun, Mon, ...
%A	Full weekday name.	Sunday, Monday, ...
%w	Weekday as a decimal number.	0, 1, ..., 6
%d	Day of the month as a zero-padded decimal.	01, 02, ..., 31
%-d	Day of the month as a decimal number.	1, 2, ..., 30
%b	Abbreviated month name.	Jan, Feb, ..., Dec
%B	Full month name.	January, February, ...
%m	Month as a zero-padded decimal number.	01, 02, ..., 12
%-m	Month as a decimal number.	1, 2, ..., 12
%y	Year without century as a zero-padded decimal number.	00, 01, ..., 99
%-y	Year without century as a decimal number.	0, 1, ..., 99
%Y	Year with century as a decimal number.	2013, 2019 etc.
%H	Hour (24-hour clock) as a zero-padded decimal number.	00, 01, ..., 23
%-H	Hour (24-hour clock) as a decimal number.	0, 1, ..., 23
%I	Hour (12-hour clock) as a zero-padded decimal number.	01, 02, ..., 12
%-I	Hour (12-hour clock) as a decimal number.	1, 2, ... 12

DateTime Formation

Directive	Meaning	Example
%p	Locale's AM or PM.	AM, PM
%M	Minute as a zero-padded decimal number.	00, 01, ..., 59
%-M	Minute as a decimal number.	0, 1, ..., 59
%S	Second as a zero-padded decimal number.	00, 01, ..., 59
%-S	Second as a decimal number.	0, 1, ..., 59
%f	Microsecond as a decimal number, zero-padded on the left.	000000 - 999999
%z	UTC offset in the form +HHMM or -HHMM.	
%Z	Time zone name.	
%j	Day of the year as a zero-padded decimal number.	001, 002, ..., 366
%-j	Day of the year as a decimal number.	1, 2, ..., 366
%U	Week number of the year (Sunday as the first day of the week). All days in a new year preceding the first Sunday are considered to be in week 0.	00, 01, ..., 53
%W	Week number of the year (Monday as the first day of the week). All days in a new year preceding the first Monday are considered to be in week 0.	00, 01, ..., 53

DateTime Formation

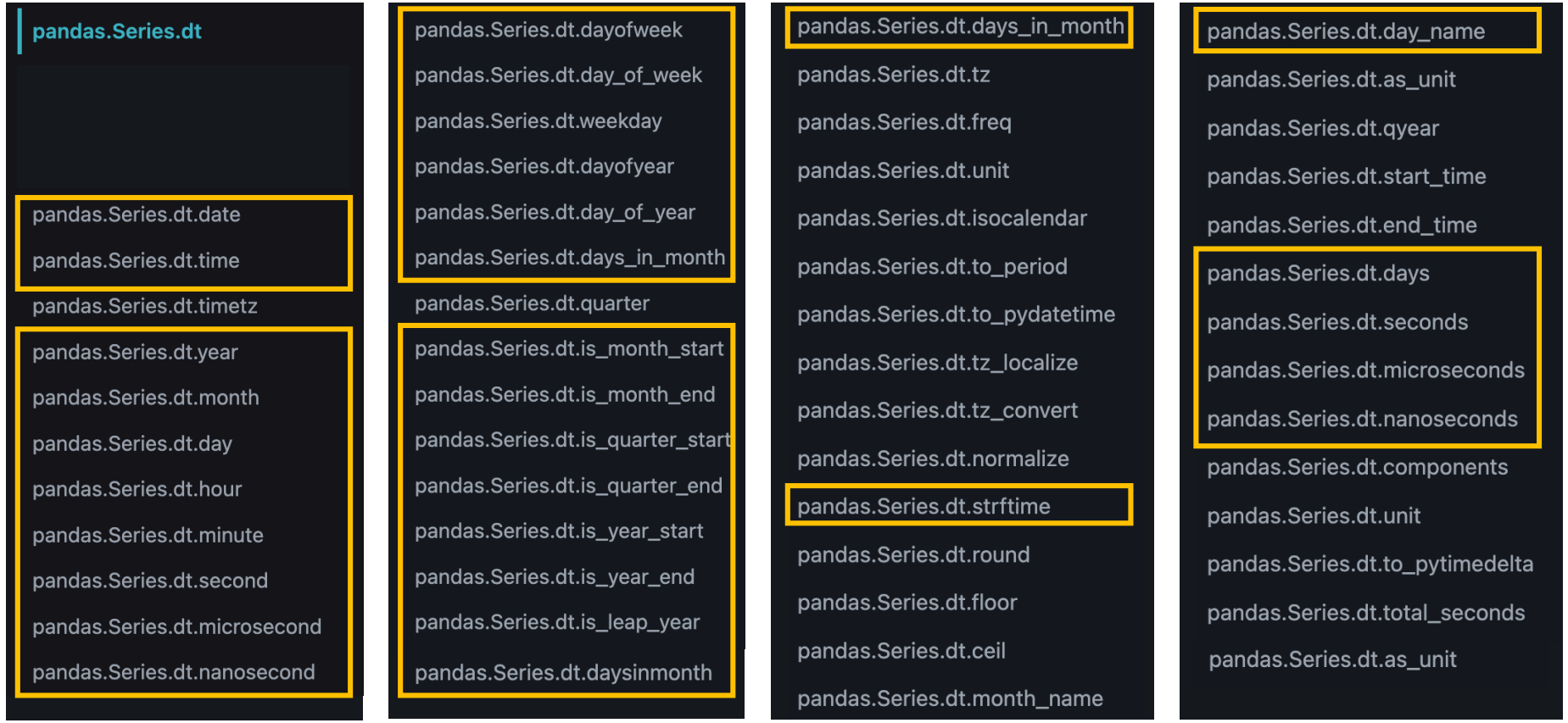
Directive	Meaning	Example
%c	Locale’s appropriate date and time representation.	Mon Sep 30 07:06:05 2013
%x	Locale’s appropriate date representation.	09/30/13
%X	Locale’s appropriate time representation.	07:06:05
%%	A literal '%' character.	%

DateTime Analysis

- In addition to handling **DateTime** formats, temporal analysis plays a crucial role in data engineering, enabling deeper insights into time-dependent patterns, trends, and anomalies. Temporal data is essential in fields such as time-series forecasting, event detection, and scheduling optimizations.
- Properly managing temporal data involves handling different time zones, dealing with missing or inconsistent timestamps, aggregating data over time windows (e.g., daily, weekly, monthly), and ensuring efficient indexing for performance optimization.
- Without robust temporal analysis, time-based insights can be inaccurate, leading to flawed decision-making in applications such as financial modeling, real-time monitoring, and predictive analytics.

DateTime Analysis

- **Pandas.Series.dt** functions



DateTime Analysis

- Here, we introduce some commonly used **Pandas.Series.dt** functions as follows:
  - Get DateTime information
  - DateTime indexing
  - Month indexing
  - Weekday information
  - Day of month and year
  - Month name
  - DateTime Range Indexing
  - DateTime Computing

DateTime Analysis – Get DateTime Info

```
# get DateTime information
print(df['ARREST_DATE'].dt.year[0], end='/')
print(df['ARREST_DATE'].dt.month[0], end='/')
print(df['ARREST_DATE'].dt.day[0], end=' ')
print(df['ARREST_DATE'].dt.hour[0], end=':')
print(df['ARREST_DATE'].dt.minute[0], end=':')
print(df['ARREST_DATE'].dt.second[0], end=':')
print(df['ARREST_DATE'].dt.microsecond[0], end=':')
print(df['ARREST_DATE'].dt.nanosecond[0])
```

```
2024/1/30 0:0:0:0:0
```

DateTime Analysis – Info

```
date:               2024-04-06
day of week:        5
day of week:        5
day of month:       5
days of month:      30
days of month:      30
day name:           Saturday
month name:         April
```

```
# DateTime related information
print('date:\t\t', df['ARREST_DATE'].dt.date[2])
print('day of week:\t', df['ARREST_DATE'].dt.dayofweek[2])
print('day of week:\t', df['ARREST_DATE'].dt.day_of_week[2])
print('day of month:\t', df['ARREST_DATE'].dt.weekday[2])
print('days of month:\t', df['ARREST_DATE'].dt.daysinmonth[2])
print('days of month:\t', df['ARREST_DATE'].dt.days_in_month[2])
print('day name:\t', df['ARREST_DATE'][2].day_name())
print('month name:\t', df['ARREST_DATE'][2].month_name())
```

DateTime Analysis – Info

```
is leap year:    True
is month start:  False
is month end:    False
is quarter start: False
is quarter end:  False
is year start:   False
is year end:     False
```

```
# Quick Check Datetime Related Information
print('is leap year:\t', df['ARREST_DATE'].dt.is_leap_year[2])
print('is month start:\t', df['ARREST_DATE'].dt.is_month_start[2])
print('is month end:\t', df['ARREST_DATE'].dt.is_month_end[2])
print('is quarter start:', df['ARREST_DATE'].dt.is_quarter_start[2])
print('is quarter end:\t', df['ARREST_DATE'].dt.is_quarter_end[2])
print('is year start:\t', df['ARREST_DATE'].dt.is_year_start[2])
print('is year end:\t', df['ARREST_DATE'].dt.is_year_end[2])
```

DateTime Analysis – Format Conversion

```
# change DateTime format
df['ARREST_DATE'].dt.strftime(date_format='%A, %AB %d, %Y')
```

```
0              Tuesday, January 30, 2024
1               Saturday, March 30, 2024
2               Saturday, April 06, 2024
3               Thursday, April 18, 2024
4                Wednesday, May 22, 2024
                       ...
260498          Friday, December 20, 2024
260499          Monday, December 23, 2024
260500          Monday, December 30, 2024
260501        Saturday, December 21, 2024
260502          Friday, December 27, 2024
Name: ARREST_DATE, Length: 260503, dtype: object
```

DateTime Analysis – DateTime Indexing

- Here, we demonstrate how to **filter a Pandas DataFrame based on a date range**, a common task in **time-series analysis** and **data preprocessing**.
- It ensures that only records within a specified time window are selected.

```
# Define a time window
s_date = pd.to_datetime("2024-03-05")
e_date = pd.to_datetime("2024-03-25")
# change DateTime format
df[(df['ARREST_DATE']>=s_date) & (df['ARREST_DATE']<e_date)]
```

DateTime Analysis

- **Execution results**

	ARREST_KEY	ARREST_DATE	PD_CD	PD_DESC	KY_CD	OFNS_DESC	LAW_CODE	LAW_CAT_CD	ARREST_BORO
25	283278758	2024-03-06	153.0	RAPE 3	104.0	RAPE	PL 1302502	F	K
79	283562758	2024-03-12	922.0	TRAFFIC,UNCLASSIFIED MISDEMEAN	348.0	VEHICLE AND TRAFFIC LAWS	VTL0511001	M	M
86	283254270	2024-03-05	439.0	LARCENY,GRAND FROM OPEN AREAS, UNATTENDED	109.0	GRAND LARCENY	PL 1553001	F	M
88	283553057	2024-03-11	779.0	PUBLIC ADMINISTRATION,UNCLASSI	126.0	MISCELLANEOUS PENAL LAW	PL 215510B	F	K
89	283242467	2024-03-05	101.0	ASSAULT 3	344.0	ASSAULT 3 & RELATED OFFENSES	PL 1200001	M	K
...	...	...	...	...	...	...	...	...	...
116204	284249898	2024-03-24	339.0	LARCENY,PETIT FROM OPEN AREAS,	341.0	PETIT LARCENY	PL 1552500	M	M
116305	284197298	2024-03-22	681.0	CHILD, ENDANGERING WELFARE	233.0	SEX CRIMES	PL 2601001	M	Q
116449	284263746	2024-03-24	114.0	OBSTR BREATH/CIRCUL	344.0	ASSAULT 3 & RELATED OFFENSES	PL 1211100	M	Q
116502	284245110	2024-03-24	115.0	RECKLESS ENDANGERMENT 2	355.0	OFFENSES AGAINST THE PERSON	PL 1202000	M	B
116547	284265263	2024-03-24	101.0	ASSAULT 3	344.0	ASSAULT 3 & RELATED OFFENSES	PL 1200001	M	Q

14723 rows × 19 columns

DateTime Analysis – Computing

```
# Create a Pandas Series with DateTime values
dates = pd.Series([
    "2024-01-01 11:23:34.34523955802",
    "2024-02-15 21:13:14.56707076436",
    "2024-03-10 07:52:23.89876345677"])

# Convert the Series to DateTime format
dates = pd.to_datetime(dates)
```

DateTime Analysis – Computing

```
# Define a reference date
reference_date = pd.to_datetime(
        "2024-01-01 23:59:32.57989844790")


# Compute the difference in days
days_difference = (dates - reference_date).dt.days
secs_difference = (dates - reference_date).dt.seconds
microsec_difference = (dates - reference_date).dt.microseconds
nanosec_difference = (dates - reference_date).dt.nanoseconds
```

DateTime Analysis – Computing

dates

```
0   2024-01-01 11:23:34.345239558
1   2024-02-15 21:13:14.567070764
2   2024-03-10 07:52:23.898763456
```

Reference_date

```
'2024-01-01 23:59:32.579898447'
```

```
# Display the results
print(days_difference)
```

```
0    -1
1    44
2    68
dtype: int64
```

```
print(secs_difference)
```

```
0    41041
1    76421
2    28371
dtype: int32
```

```
# Display the results
print(microsec_difference)
```

```
0    765341
1    987172
2    318865
dtype: int32
```

```
print(nanosec_difference)
```

```
0    111
1    317
2      9
dtype: int32
```

Lab Practice #1

- Read File (csv, xlsx, xls, and json…)

```
# read file
df = pd.read_csv('Airline Dataset Updated - v2.csv')
df.head()
```

	Passenger ID	First Name	Last Name	Gender	Age	Nationality	Airport Name	Airport Country Code	Country Name	Airport Continent	Continents	Departure Date	Arrival Airport	Pilot Name	Flight Status
0	ABVWIg	Edithe	Leggis	Female	62	Japan	Coldfoot Airport	US	United States	NAM	North America	6/28/2022	CXF	Fransisco Hazeldine	On Time
1	jkXXAX	Elwood	Catt	Male	62	Nicaragua	Kugluktuk Airport	CA	Canada	NAM	North America	12/26/2022	YCO	Marla Parsonage	On Time
2	CdUz2g	Darby	Felgate	Male	67	Russia	Grenoble-Isère Airport	FR	France	EU	Europe	1/18/2022	GNB	Rhonda Amber	On Time
3	BRS38V	Dominica	Pyle	Female	71	China	Ottawa / Gatineau Airport	CA	Canada	NAM	North America	9/16/2022	YND	Kacie Commucci	Delayed
4	9kvTLo	Bay	Pencost	Male	21	China	Gillespie Field	US	United States	NAM	North America	2/25/2022	SEE	Ebonee Tree	On Time

Lab Practice #1

- Number of pax took an On Time flight
- How many airports are arriving in the dataset?
- How many men and women are in the dataset?
- How many pax are in the dataset?
- How many countries are in the dataset?
- Which country has the highest number of pax in this dataset?
- Which airport has the highest number of pax in this dataset?
- How many male Russian pax arrive in France?
- How many female Japan pax arrive in Canada on time?
- Following the previous question, what is their average age?
- How many female pax flows travel on the weekends?

The End

Thank you for your attention!

Email: chchan@ntnu.edu.tw

Website: https://toodou.github.io/