Data Processing :

Exploratory Data Analysis

Crafted by :
Raudhoh Fitra H

Objective

- To understand the process in Exploratory Data Analysis as a part of Data Science Lifecycle
- To have first hand experience in doing exploratory data analysis

Outline

01 Introduction to Exploratory Data Analysis

02 Implement data wrangling, and EDA

Scope of Exploratory Data Analysis

Data Wrangling

Exploration

Data Cleansing

Data Wrangling and Data Cleansing

- Collect the data
- Join data from various sources
- Inspect the initial condition of data
- Inspect null value of each variables
- Inspect dataset shape
- Inspect data type
- Impute null value
- Data type transformation
- Data reshaping
- Some data scientist will refer to cleansing and wrangling as same thing

Data Cleansing (Recall)

- Missing Values Checking and Handling
- Duplicates Checking
- Anomaly and Outlier Detection
- Data Type Checking
- Data type correction
- Feature extraction

Duplicate Data

Is a condition where some rows has partially or completely same.

DataFrame.**drop_duplicates**(*self, subset: Union[Hashable, Sequence[Hashable], NoneType] = None, keep: Union[str, bool] = 'first', inplace: bool = False, ignore_index: bool = False*).

	ColumnA	ColumnB
0	111	444
1	222	555
2	333	666
3	111	444
4	222	777

```
data.drop_duplicates()
```

	ColumnA	ColumnB
0	111	444
1	222	555
2	333	666
4	222	777

```
data.drop_duplicates(subset = 'ColumnA')
```

	ColumnA	ColumnB
0	111	444
1	222	555
2	333	666

Missing Value

Why missing value exist?

- Values are missed during data collection / acquisition process
- Values are deleted accidentally
- Corrupt data
- Mismatch between row and column position

Data Imputation

Methods for Data Imputation:

- **Median** (Used for skewness distribution)
- **Mode** (Used for categorical type)
- **Mean** (Used for normally distributed data)
- **Custom Values**

DataFrame.**fillna**(self, value=None, method=None, axis=None, inplace=False, limit=None, downcast=None)

Data Imputation

	ColumnA	ColumnB
0	111	444.0
1	222	555.0
2	333	666.0
3	111	444.0
4	222	NaN

```
fill_value = data['ColumnB'].median()
data['ColumnB'] = data['ColumnB'].fillna(fill_value)
data
```

	ColumnA	ColumnB
0	111	444.0
1	222	555.0
2	333	666.0
3	111	444.0
4	222	499.5

Median

	ColumnA	ColumnB
0	111	A
1	222	A
2	333	A
3	111	B
4	222	NaN

```
fill_value = df['ColumnB'].mode()[0]
df['ColumnB'] = df['ColumnB'].fillna(fill_value)
df
```

	ColumnA	ColumnB
0	111	A
1	222	A
2	333	A
3	111	B
4	222	A

Mode

Data Imputation

	ColumnA	ColumnB
0	111	444.0
1	222	555.0
2	333	666.0
3	111	444.0
4	222	NaN

```
fill_value = data['ColumnB'].mean()
data['ColumnB'] = data['ColumnB'].fillna(fill_value)
data
```

	ColumnA	ColumnB
0	111	444.00
1	222	555.00
2	333	666.00
3	111	444.00
4	222	527.25

→ Mean

```
df['columnA'].fillna("none", inplace=True)
```

```
df['columnB'].fillna(0, inplace=True)
```

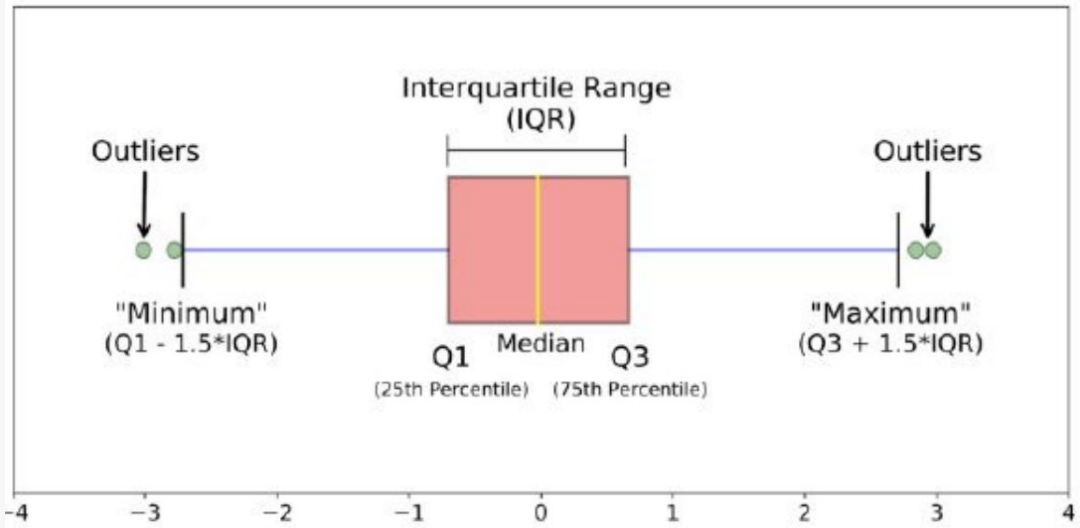
→ Other values

Outliers

An outlier is a data point that lies an abnormal distance from other values in data

Basic Outlier Formula :

1. Lower Bound = Q1 1.5 x IQR
2. Upper Bound = Q3 + 1.5 x IQR
3. IQR = Q3 - Q1



The **box plot** is a useful graphical display for describing the behavior of the data in the middle as well as at the ends of the distributions

Data Type

Data Frame

	ColumnA	ColumnB
0	111	444
1	222	555
2	333	666
3	111	444
4	222	777

Type of Column A

```
data['ColumnA'].dtype
```

```
dtype('int64')
```

To string

```
data['ColumnA'] = data['ColumnA'].astype('str')
data['ColumnA'].dtype
```

```
dtype('O')
```

To float

```
data['ColumnA'] = data['ColumnA'].astype('float64')
data['ColumnA'].dtype
data
```

	ColumnA	ColumnB
0	111.0	444
1	222.0	555
2	333.0	666
3	111.0	444
4	222.0	777

Data Type

To Datetime

```
from datetime import datetime as dtime
data['columnC'] = pd.to_datetime(data['columnC'], format="%Y-%m-%d")
```

```
data['columnC']

0         2014-08-03
1         2014-08-03
2         2014-08-03
3         2014-08-03
4         2014-08-03
             ...
641909    2016-06-03
641910    2016-06-03
641911    2016-06-03
641912    2016-06-03
641913    2016-06-03
Name: columnC, Length: 641914, dtype: object
```

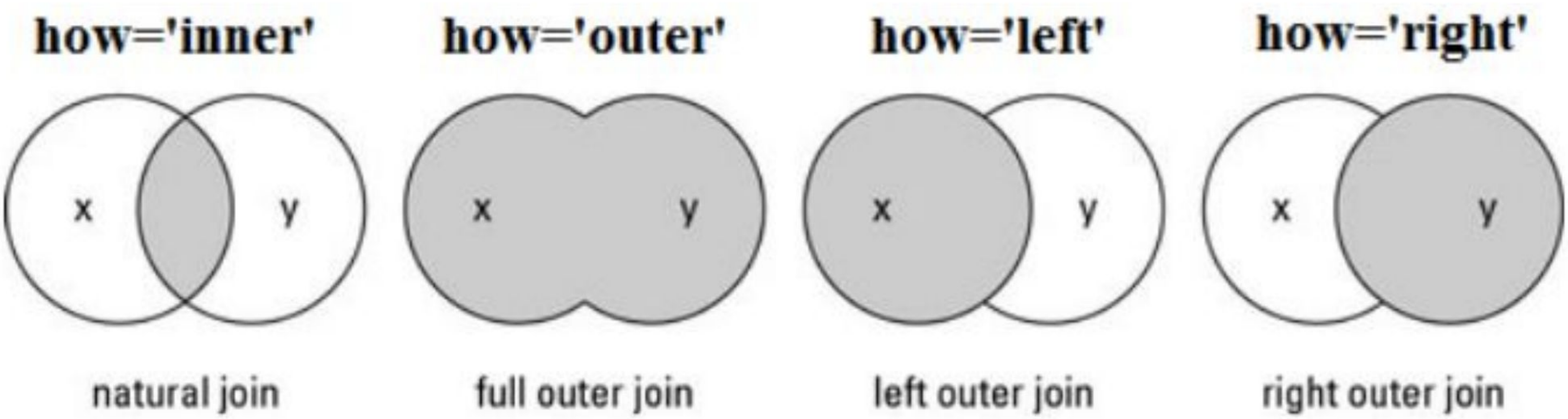
→

```
data['columnC']

0         2014-08-03
1         2014-08-03
2         2014-08-03
3         2014-08-03
4         2014-08-03
             ...
641909    2016-06-03
641910    2016-06-03
641911    2016-06-03
641912    2016-06-03
641913    2016-06-03
Name: columnC, Length: 641914, dtype: datetime64[ns]
```

Combining Data

Join/ Merge types :



Combining Data : concatenating

The concat() function (in the main pandas namespace) does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

pd.concat(objs, axis=0, join='outer', ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, copy=True)

```
pd.concat([df1,df2], join='outer')
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1

	A	B	C	E
0	A4	B4	C7	E7
1	A5	B5	C7	E7

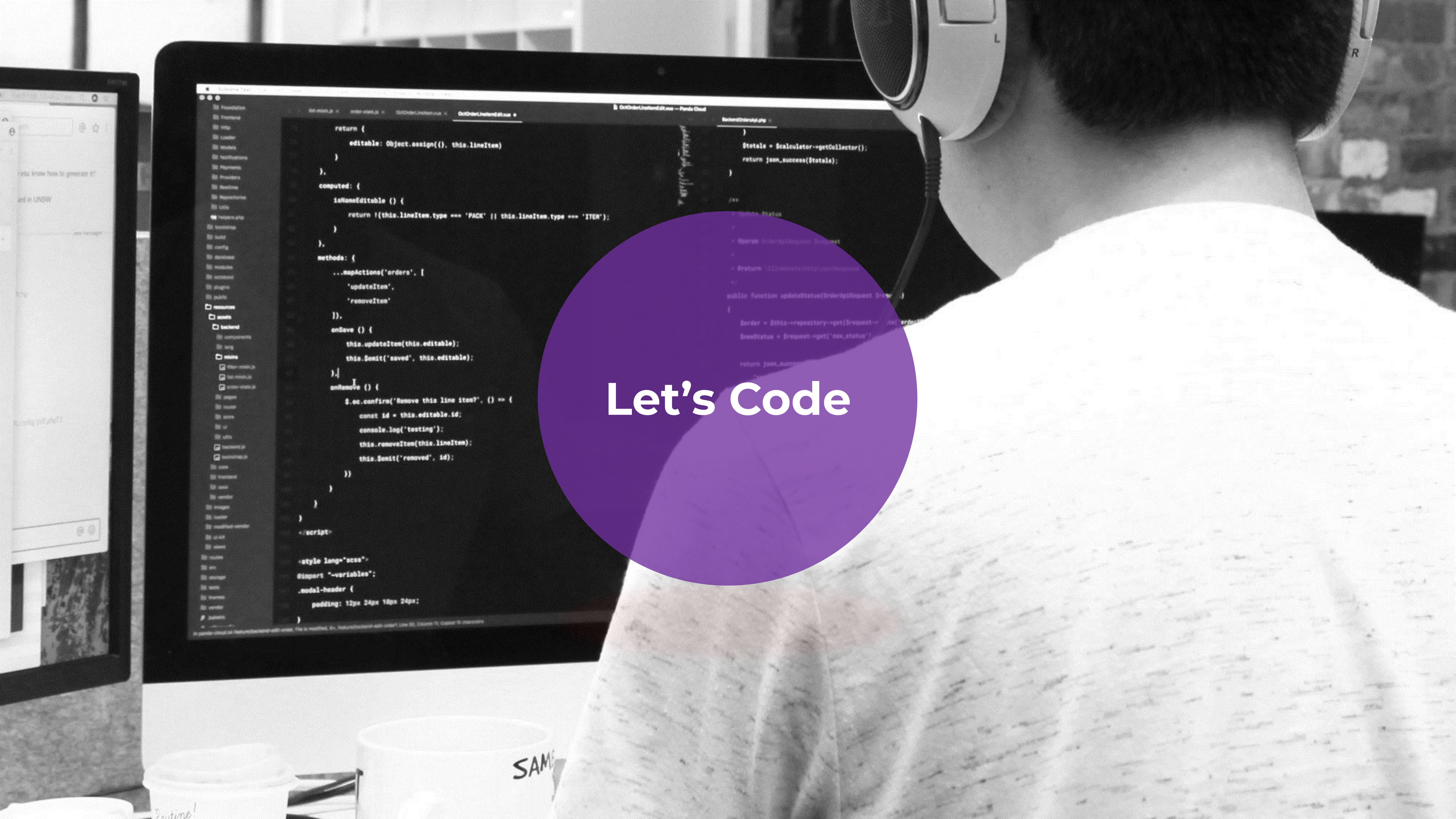
	A	B	C	D	E
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
0	A4	B4	C7	NaN	E7
1	A5	B5	C7	NaN	E7

```
pd.concat([df1,df2], join='inner')
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1

	A	B	C	E
0	A4	B4	C7	E7
1	A5	B5	C7	E7

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
0	A4	B4	C7
1	A5	B5	C7



Let’s Code

Thank you!

Data MBA Online Program

2021

iykra | Data MBA Management Business Analytics