



# Data Preparation

Dr. Sathien Hunta

School of Information and  
Communication Technology

University of Phayao



# Data Preparation

- Data correcting
  - Replace value
- Missing value
  - Replace missing value
- Outlier
  - Outlier detection
- Normalization (Standardization)
  - Numeric -> equal scale



# Import data



```
## Import required libraries
```

```
import numpy as np
```

```
import pandas as pd
```

```
## Upload dataset
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

```
## Read a .csv file to pandas dataframe  
df = pd.read_csv(uploaded['data.csv'])
```

```
## Read a .json file to pandas dataframe  
df = pd.read_json(uploaded['data.json'])
```

```
## Read an excel file to pandas dataframe  
df = pd.read_excel(uploaded['data.xlsx'])
```



# Handling Missing Values

1s

```
import pandas as pd
ufo = pd.read_csv(data_path+'ufo.csv')
ufo.head
```

<bound method NDFrame.head of Reported State \

	City	Colors	Reported
0	Ithaca	NaN	TRIANGLE NY
1	Willingboro	NaN	OTHER NJ
2	Holyoke	NaN	OVAl CO
3	Abilene	NaN	DISK KS
4	New York Worlds Fair	NaN	LIGHT NY
...	...	...	...
18236	Grant Park	NaN	TRIANGLE IL
18237	Spirit Lake	NaN	DISK IA
18238	Eagle River	NaN	NaN WI
18239	Eagle River	RED	LIGHT WI
18240	Ybor	NaN	OVAl FL

3s

```
ufo.tail()
```

City Colors Reported Shape Reported State Time

18236	Grant Park	NaN	TRIANGLE	IL	12/31/2000 23:00
18237	Spirit Lake	NaN	DISK	IA	12/31/2000 23:00
18238	Eagle River	NaN	NaN	WI	12/31/2000 23:45
18239	Eagle River	RED	LIGHT	WI	12/31/2000 23:45
18240	Ybor	NaN	OVAl	FL	12/31/2000 23:59

0s `ufo.isnull()`

	City	Colors	Reported	Shape	Reported	State	Time
0	False		True	False	False	False	False
1	False		True	False	False	False	False
2	False		True	False	False	False	False
3	False		True	False	False	False	False
4	False		True	False	False	False	False
...	...		...	...	...	...	...
18236	False		True	False	False	False	False
18237	False		True	False	False	False	False
18238	False		True	True	False	False	False
18239	False		False	False	False	False	False
18240	False		True	False	False	False	False

18241 rows x 5 columns

True = Missing value

0s `ufo.isnull().sum()`

City	25
Colors Reported	15359
Shape Reported	2644
State	0
Time	0
dtype: int64	

filter

0s `ufo[ufo.City.isnull()]`

	City	Colors	Reported	Shape	Reported	State	Time
21	NaN		NaN		NaN	LA	8/15/1943 0:00
22	NaN		NaN		LIGHT	LA	8/15/1943 0:00
204	NaN		NaN		DISK	CA	7/15/1952 12:30
241	NaN	BLUE		DISK		MT	7/4/1953 14:00
613	NaN		NaN	DISK		NV	7/1/1960 12:00
1877	NaN	YELLOW		CIRCLE		AZ	8/15/1969 1:00
2013	NaN		NaN		NaN	NH	8/1/1970 9:30
2546	NaN		NaN	FIREBALL		OH	10/25/1973 23:30

## Drop Missing Values

0s [8] `ufo.shape`

(18241, 5)

0s [9] `ufo.dropna(how='any').shape`

(2486, 5)

0s [10] `ufo.dropna(how='all').shape`

(18241, 5)

0s [11] `ufo.dropna(subset=['City', 'Shape Reported'], how='any').shape`

(15576, 5)

0s `ufo.dropna(subset=["City", "Shape Reported"], how='all').shape`

(18237, 5)

# Filling Missing Values

```
ufo["Shape Reported"].value_counts()
```

LIGHT	2803
DISK	2122
TRIANGLE	1889
OTHER	1402
CIRCLE	1365
SPHERE	1054
FIREBALL	1039
OVAL	845
CIGAR	617
FORMATION	434
VARIOUS	333
RECTANGLE	303
CYLINDER	294
CHEVRON	248
DIAMOND	234
EGG	197
FLASH	188
TEARDROP	119
CONE	60
CROSS	36
DELTA	7
ROUND	2
CRESCENT	2
DOME	1
PYRAMID	1
FLARE	1
HEXAGON	1

Name: Shape Reported, dtype: int64

```
ufo['Shape Reported'].value_counts(dropna=False)
```

LIGHT	2803
NaN	2644
DISK	2122
TRIANGLE	1889
OTHER	1402
CIRCLE	1365
SPHERE	1054
FIREBALL	1039
OVAL	845
CIGAR	617
FORMATION	434
VARIOUS	333
RECTANGLE	303
CYLINDER	294
CHEVRON	248
DIAMOND	234
EGG	197
FLASH	188
TEARDROP	119
CONE	60
CROSS	36
DELTA	7
ROUND	2
CRESCENT	2
DOME	1
PYRAMID	1
FLARE	1
HEXAGON	1

Name: Shape Reported, dtype: int64

```
[15] ufo['Shape Reported'].fillna(value="VARIOUS", inplace=True)
```

```
ufo["Shape Reported"].value_counts()
```

VARIOUS	2977
LIGHT	2803
DISK	2122
TRIANGLE	1889
OTHER	1402
CIRCLE	1365
SPHERE	1054
FIREBALL	1039
OVAL	845
CIGAR	617
FORMATION	434
RECTANGLE	303
CYLINDER	294
CHEVRON	248
DIAMOND	234
EGG	197
FLASH	188
TEARDROP	119
CONE	60
CROSS	36
DELTA	7
ROUND	2
CRESCENT	2
DOME	1
PYRAMID	1
FLARE	1
HEXAGON	1

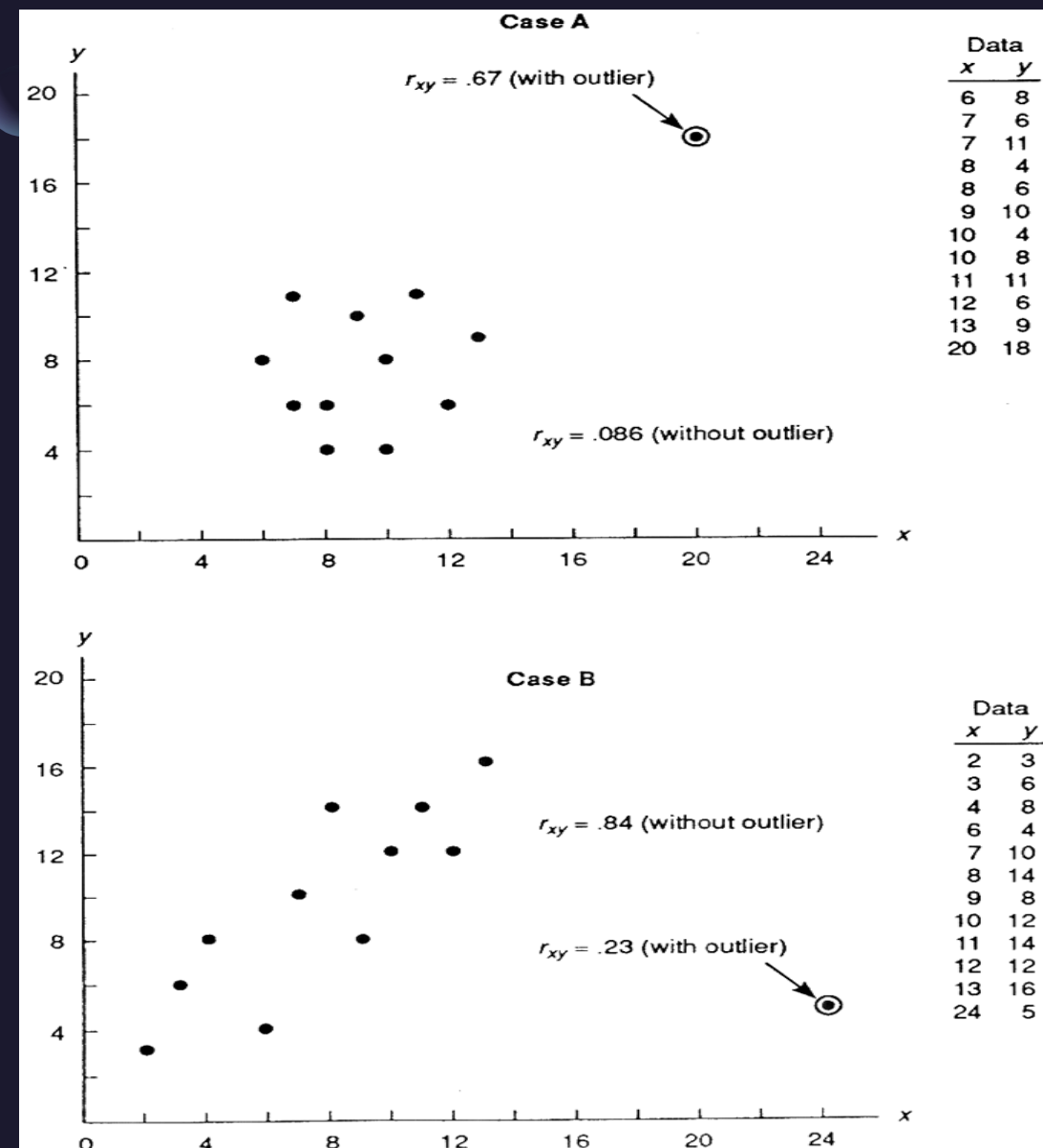
Name: Shape Reported, dtype: int64

# Outlier

Outlier คือสิ่งที่ผิดปกติภายใน Data ซึ่ง  
อาจเกิดจากความผิดพลาดที่อาจเกิด  
จากกลุ่มตัวอย่างมีความผิดปกติหรือ  
แตกต่างไปจากกลุ่ม หรือมาจากการ  
ผิดพลาดจากการบันทึกหรือเก็บ

## รวบรวมข้อมูล


หากนำข้อมูลไปใช้ Algorithm จะนำความผิดปกติเหล่านั้นไปคำนวณ  
เป็นส่วนหนึ่งของโมเดล ซึ่งหมายความว่า โมเดลที่ได้ จะมี  
ประสิทธิภาพลดลง




# Outlier Detection

```
✓ [23] import pandas as pd  
0s      import numpy as np
```

```
✓ [25] from sklearn.datasets import load_boston  
0s
```

```
✓  boston = load_boston()  
0s      boston.data.shape
```

```
(506, 13)
```

```
✓  print(boston.DESCR)
```

```
.. _boston_dataset:
```

Boston house prices dataset

-----

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

:Attribute Information (in order):

- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX nitric oxides concentration (parts per 10 million)
- RM average number of rooms per dwelling
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per \$10,000
- PTRATIO pupil-teacher ratio by town
- B  $1000(B_k - 0.63)^2$  where  $B_k$  is the proportion of black people by town
- LSTAT % lower status of the population
- MEDV Median value of owner-occupied homes in \$1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.

<https://archive.ics.uci.edu/ml/machine-learning-databases/housing/>



# Box Plot

For univariate

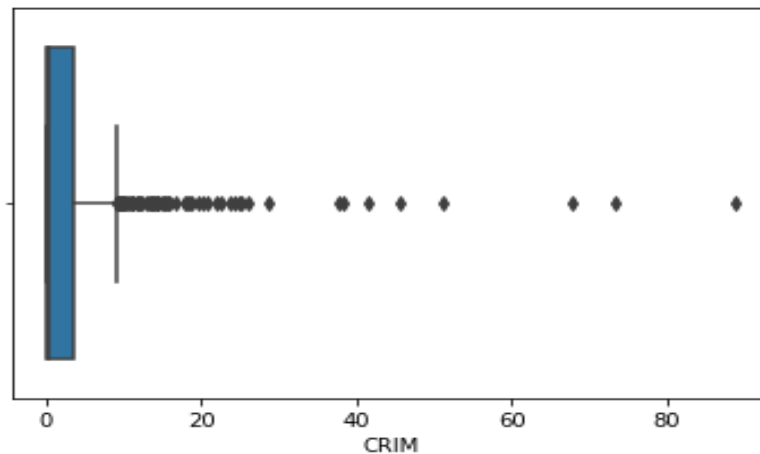
✓  
0s



```
import seaborn as sns  
sns.boxplot(x=boston_df['CRIM'])
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5a55fd490>
```

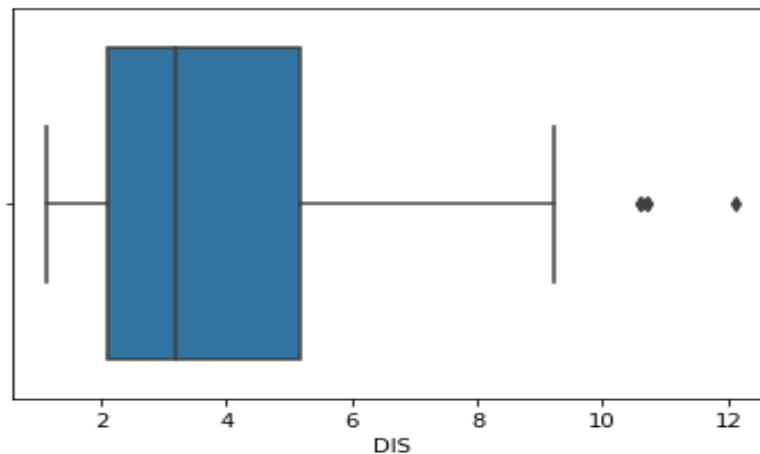


✓  
0s



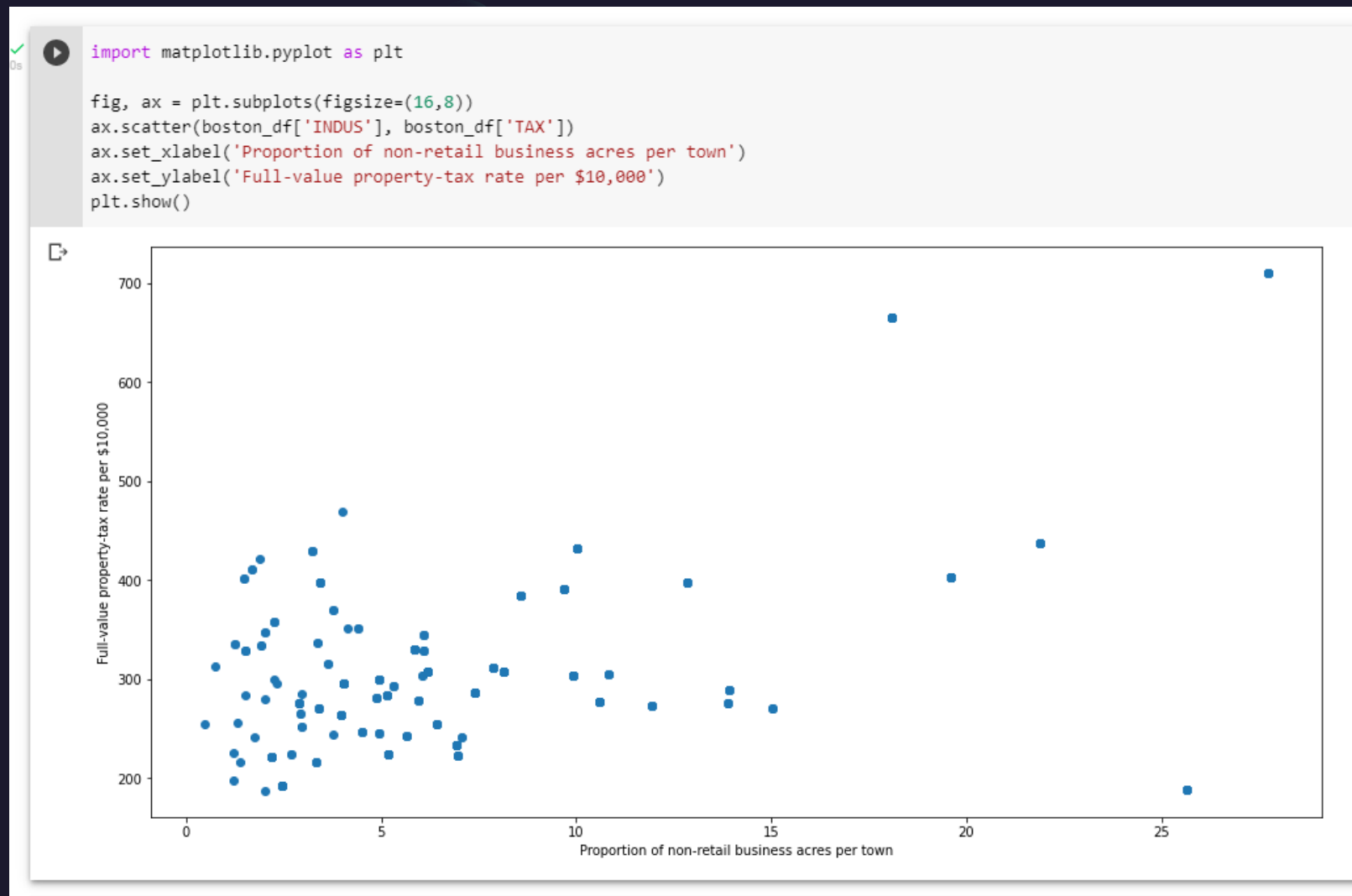
```
sns.boxplot(x=boston_df['DIS'])
```

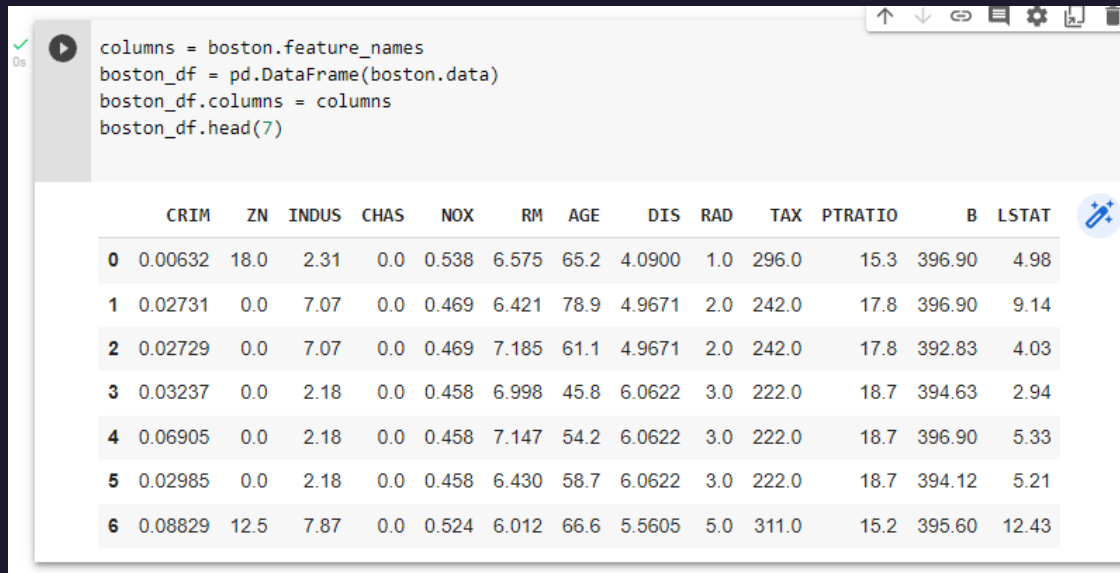
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5a5562290>
```



# Scatter plot

For multivariate

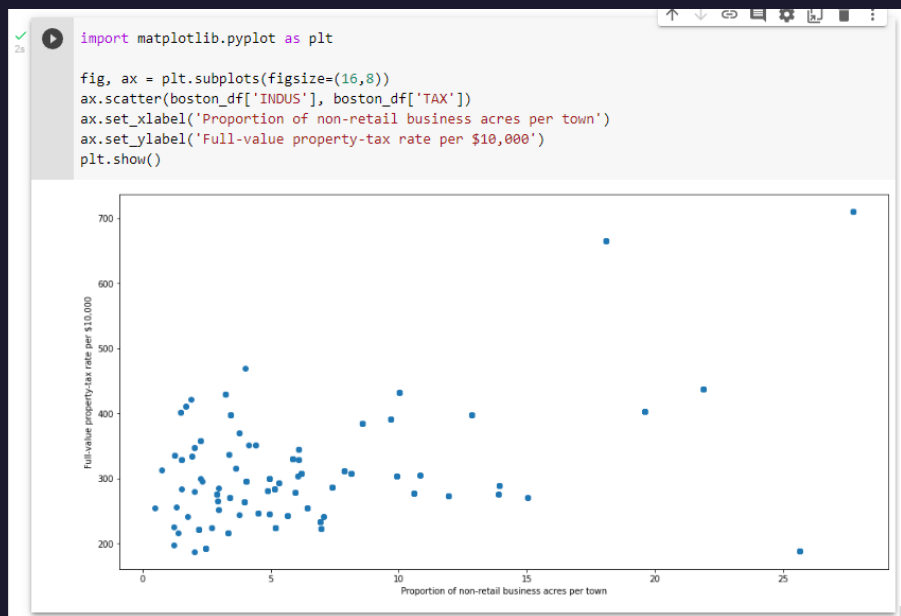




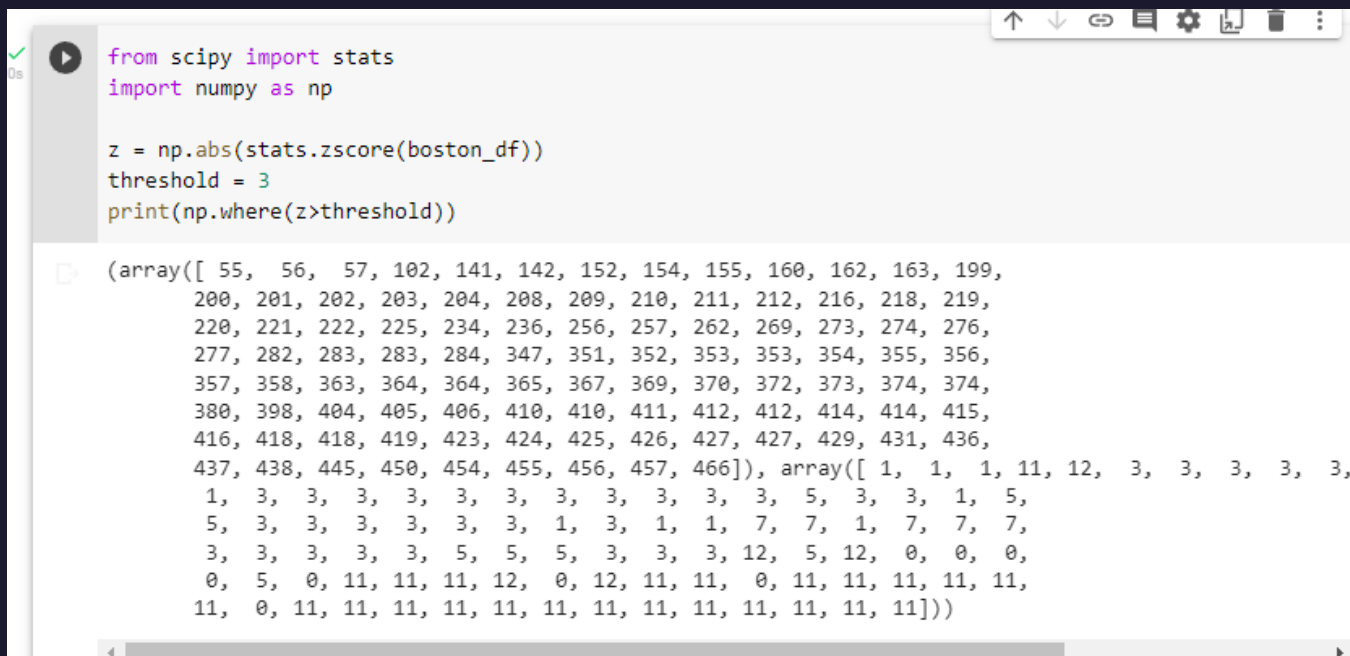
**Z-Score** คือ คะแนนมาตรฐาน ที่เป็น อัตราส่วนระหว่างการเบี่ยงเบนของคะแนนจากค่าเฉลี่ย กับส่วนเบี่ยงเบนมาตรฐาน หรือเป็นการเปรียบเทียบให้เห็นว่าคะแนนอยู่ห่างจากค่าเฉลี่ย เป็นกี่หน่วยส่วนเบี่ยงเบนมาตรฐาน (ถ้าค่ามากจะมีโอกาสเป็น outlier)

สูตรสำหรับคำนวณ **z-score** ของชุดข้อมูลใด ๆ คือ  $z = (x - \mu) / \sigma$  โดยที่  $\mu$  คือค่าเฉลี่ยของจำนวนประชากรและ  $\sigma$  คือส่วนเบี่ยงเบนมาตรฐานของประชากร ค่าสัมบูรณ์ของ **Z** หมายถึง **z-score** ของประชากรระยะห่างระหว่างคะแนนดิบและค่าเฉลี่ยของประชากรในหน่วยของส่วนเบี่ยงเบนมาตรฐาน

## Scatter plot



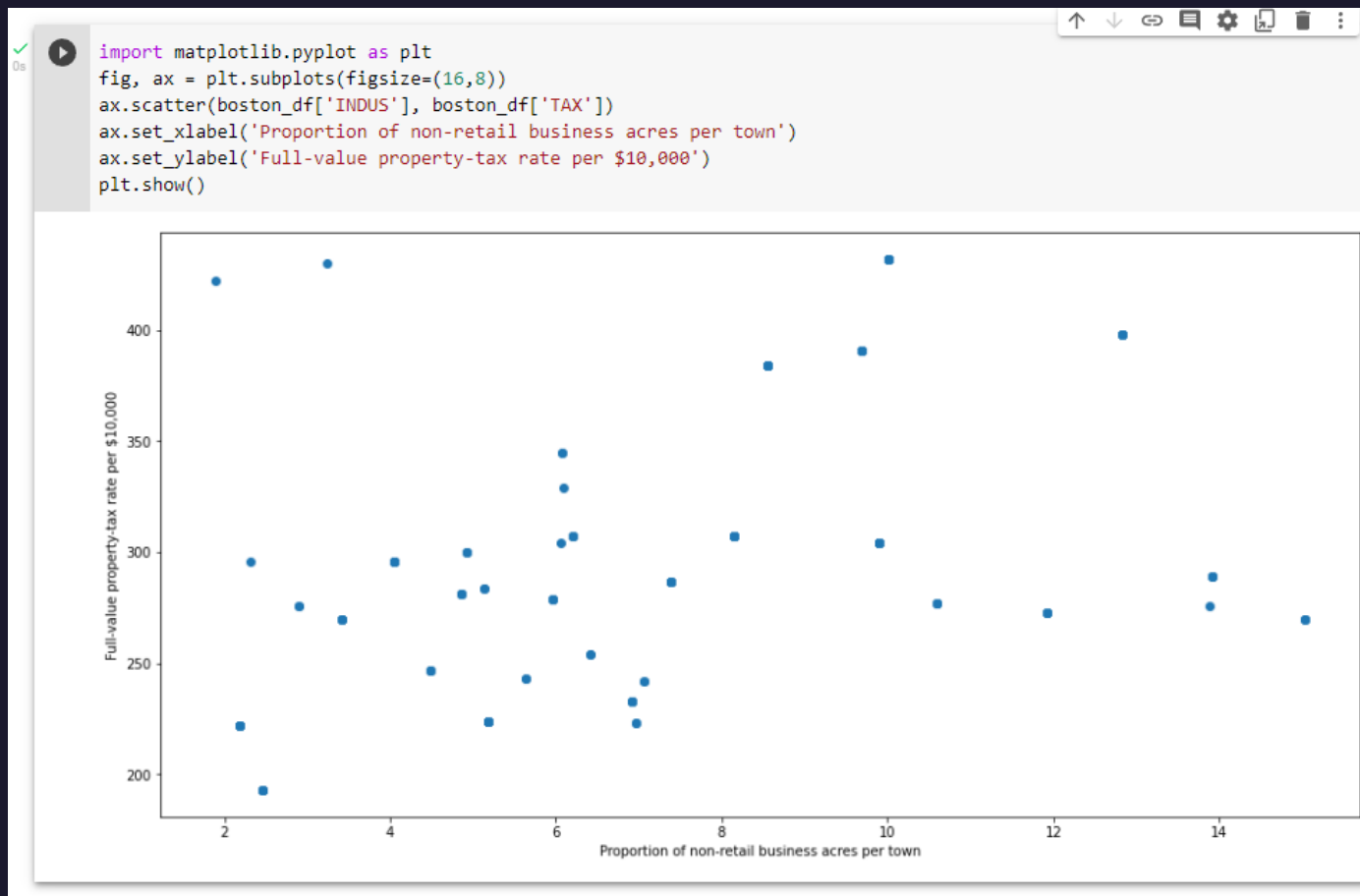
## Z-Score



# Remove Outlier

```
✓ boston_df = boston_df[(z<2).all(axis=1)]  
  
✓ [43] boston_df.shape  
1s  
(415, 13)
```

ในตัวอย่าง Dataframe ที่ใช้จะเลือกเอาเฉพาะ  $z < 2$  ส่วนที่เหลือจะเป็น outlier



# Data Standardization

```
[4] import pandas as pd
import numpy as np
```

```
data = pd.read_csv(data_path+'wine.csv')
data.head()
```

	Wine	Alcohol	Malic.acid	Ash	Ac1	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
[17] data.describe()
```

Label

Features

	Wine	Alcohol	Malic.acid	Ash	Ac1	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000

# Data Standardization

ข้อมูลดิบที่เราได้รับมานั้นมีความหลากหลาย ทั้งชนิดข้อมูล รูปแบบข้อมูล และ Scale ช่วงของข้อมูล

เมื่อทำการสร้าง Model เมื่อเลือกใช้ Feature ในการ train จะมีช่วงของข้อมูลไม่เท่ากัน ทำให้ Feature น้ำหนัก ที่จำนวนใหญ่กว่า ช่วงใหญ่กว้างกว่า จะบดบัง Feature อื่น ๆ ไปหมด การปรับ scale ของ Feature ให้อยู่ในช่วงที่ต้องการ เช่น  $[0, 1]$  หรือ  $[-1, +1]$  เหมือนกันหมด ก็จะแก้ปัญหานี้ได้

Standardization (Z-Score Normalization)

$$\text{New value} = (x - \mu) / \sigma$$

where:

$x$ : Original value

$\mu$ : Mean of data

$\sigma$ : Standard deviation of data

อัลกอริทึม Machine Learning หลาย ๆ ตัว จำเป็นต้องปรับข้อมูลให้เป็นแบบนี้ก่อน ที่จะป้อนให้โมเดลใช้ในการ train เพื่อให้ประสิทธิภาพดีขึ้น

# Modeling without normalizing

```
✓ 0s ▶ X = data[['Proline', 'Phenols', 'Hue', 'Nonflavanoid.phenols']]
      y = data['Wine']
      from sklearn.model_selection import train_test_split
      from sklearn.neighbors import KNeighborsClassifier

      knn = KNeighborsClassifier()

      # Split the dataset and labels into training and test sets
      X_train, X_test, y_train, y_test = train_test_split(X, y)

      # Fit the k-nearest neighbors model to the training data
      knn.fit(X_train, y_train)

      # Score the model on the test data
      print(knn.score(X_test, y_test))
```

0.6666666666666666

# Modeling with normalizing

```
✓ 1s ▶ from sklearn.preprocessing import StandardScaler
X = data[['Proline', 'Phenols', 'Hue', 'Nonflavanoid.phenols']]
y = data['Wine']
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

# Create the scaling method
ss = StandardScaler()

# Apply the scaling method to the dataset used for modeling
X_scaled = ss.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y)

# Fit the k-nearest neighbors model to the training data.
knn.fit(X_train, y_train)

# Score the model on the test data
print(knn.score(X_test, y_test))
```

0.8444444444444444



# Standardized data and modeling

```
0s ✓ ▶ print(X_scaled)
0s [ [ 1.51861254 -0.5622498  0.23205254 ... 0.36217728 1.84791957
    1.01300893]
    [ 0.24628963 -0.49941338 -0.82799632 ... 0.40605066 1.1134493
    0.96524152]
    [ 0.19687903  0.02123125  1.10933436 ... 0.31830389 0.78858745
    1.39514818]
    ...
    [ 0.33275817  1.74474449 -0.38935541 ... -1.61212515 -1.48544548
    0.28057537]
    [ 0.20923168  0.22769377  0.01273209 ... -1.56825176 -1.40069891
    0.29649784]
    [ 1.39508604  1.58316512  1.36520822 ... -1.52437837 -1.42894777
    -0.59516041]]
```



```
0s ✓ ▶ X = data.drop('Wine', axis=1)
    y = data['Wine']

    knn = KNeighborsClassifier()
    # Split the dataset and labels into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y)

    # Fit the k-nearest neighbors model to the training data
    knn.fit(X_train, y_train)

    # Score the model on the test data
    print(knn.score(X_test, y_test))

0.7111111111111111 ←
```

```
0s ✓ ▶ from sklearn.preprocessing import StandardScaler
    knn = KNeighborsClassifier()

    # Create the scaling method
    ss = StandardScaler()

    # Apply the scaling method to the dataset used for modeling
    X_scaled = ss.fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y)

    # Fit the k-nearest neighbors model to the training data.
    knn.fit(X_train, y_train)

    # Score the model on the test data
    print(knn.score(X_test, y_test))

0.9555555555555556 ←
```



# Summary

**Data Preparation** หมายถึงกระบวนการใด ๆ ที่จำเป็นต้องทำกับข้อมูลดิบที่ได้มา เพื่อปรับเปลี่ยนข้อมูลให้อยู่ในรูปแบบที่เหมาะสมกับการนำไปใช้งาน

# Thank You

Sathien Hunta

[sathien.hu@up.ac.th](mailto:sathien.hu@up.ac.th)

<http://ict.up.ac.th>

