INFOPRESS

Python เรียนรู้ Concept และฝึกฝน Coding

Data Science

- เรียนรู้หลักการและวิธีการ Coding เกี่ยวกับ Data Science เบื้องต้นอย่างครบถ้วน
- แนะนำการ Coding กับ NumPy, Pandas, MongoDB, Matplotlib และ PandasGUI
- หากมีพื้นฐาน Python เบื้องต้นมาแล้ว สามารถศึกษาด้วยตนเอง Step-by-Step ได้อย่างรวดเร็ว
- เหมาะสำหรับนักศึกษาและผู้สนใจจะเพิ่ม Skill การ Coding เกี่ยวกับ Data Science

ไฟล์ตัวอย่างภายในเล่ม
https://serazu.com/
9786164872394

ศุภชัย สมพานิช

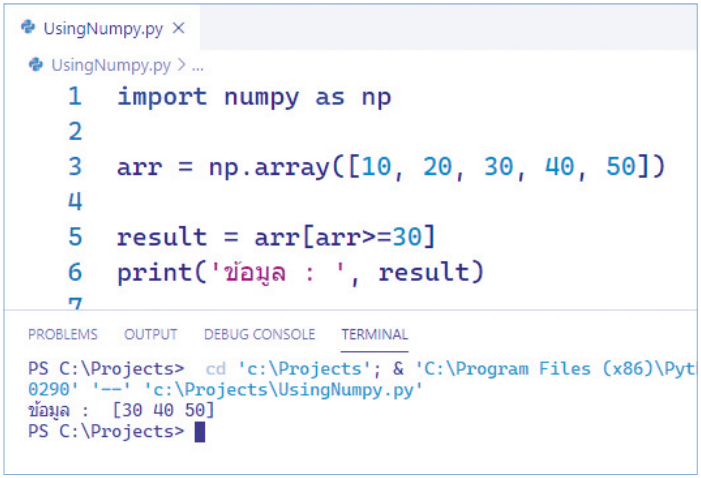
สารบัญ

การเลือกข้อมูลจากอาร์เรย์แบบมีเงื่อนไขด้วยฟังก์ชัน where()

โครงสร้างข้อมูลแบบอาร์เรย์ของ NumPy มีความสามารถเลือกดูข้อมูลแบบมีเงื่อนไขติดตัวอยู่แล้ว เช่น มีข้อมูลเป็นตัวเลขจำนวนเต็ม 5 ตัว เก็บอยู่ในตัวแปร arr หากกำหนดเงื่อนไขว่า ต้องการข้อมูลที่มีค่ามากกว่าหรือเท่ากับ 30 ผลที่ได้คือ 30, 40 และ 50 มีโค้ดดังต่อไปนี้

Python

```
import numpy as np

arr = np.array([10, 20, 30, 40, 50])

result = arr[arr>=30]
print('ข้อมูล : ', result)
```



รูปที่ 2-33 กรณีมีข้อมูลตรงกับเงื่อนไข

ต่อมาเปลี่ยนเงื่อนไขใหม่ ต้องการข้อมูลที่มีค่ามากกว่าหรือเท่ากับ 100 ถือเป็นเงื่อนไขที่ไม่ตรงกับข้อมูลที่มีอยู่ ผลที่ได้คือ เป็นค่าว่าง ดังรูปที่ 2-34

Python

```
result = arr[arr>=100]
```

การรวม DataFrame ด้วยฟังก์ชัน concat()

ฟังก์ชัน **concat()** ทำหน้าที่เชื่อมต่อโครงสร้างข้อมูล DataFrame เข้าด้วยกัน การรวมข้อมูลด้วยวิธีนี้มีข้อดีตรงที่ผู้อ่านสามารถกำหนดค่า Index ได้ด้วยตัวเอง

ตัวอย่างที่ 4-8 การรวม DataFrame ด้วยฟังก์ชัน concat()

มีข้อมูลเก็บอยู่ในโครงสร้าง DataFrame ที่แตกต่างกันอยู่ 2 ชุด เก็บอยู่ในตัวแปร data1 กับ data2 ตามลำดับ หากต้องการรวมข้อมูลเข้าด้วยกัน มีโค้ดดังต่อไปนี้

โค้ดไพธอนที่ 4-8 การรวม DataFrame ด้วยฟังก์ชัน concat() (DataFrameConcat.ipynb)

```
import pandas as pd

data1 = [
            ['สมชาย', 185, 16000], ['วีระชัย', 175, 17500],
            ['พรชัย', 182, 20000], ['วีระ', 183, 19900]
        ]
data2 = [
            ['ดวงใจ', 165, 17800, 'กรุงเทพ'], ['วิไล', 163, 19300, 'เชียงใหม่']
        ]

colsname1 = ['ชื่อ', 'ส่วนสูง', 'เงินเดือน']
colsname2 = ['ชื่อ', 'ส่วนสูง', 'เงินเดือน', 'ที่อยู่']

df1 = pd.DataFrame(data1, columns=colsname1, index={3, 4, 5, 6})
df2 = pd.DataFrame(data2, columns=colsname2, index={1, 2})

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

	ชื่อ	ส่วนสูง	เงินเดือน	ที่อยู่
3	สมชาย	185	16000	NaN
4	วีระชัย	175	17500	NaN
5	พรชัย	182	20000	NaN
6	วีระ	183	19900	NaN
1	ดวงใจ	165	17800	กรุงเทพ
2	วิไล	163	19300	เชียงใหม่

รูปที่ 4-21 แสดงโครงสร้างข้อมูลจาก DataFrame 2 ชุด

จากรูปที่ 4-21 พบว่าโครงสร้างข้อมูลที่แตกต่างกัน ถูกเติมด้วยข้อมูลสูญหาย NaN

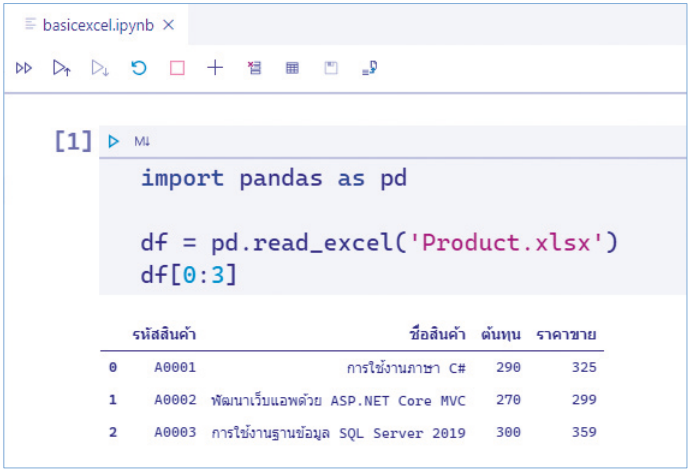
การเลือกข้อมูลเป็นช่วงจากไฟล์ Excel

การเรียกดูข้อมูลจากไฟล์ Excel ผู้เขียนเลือกเขียนโค้ดแบบ Jupyter Notebook เพื่อให้ส่วนแสดงผลชัดเจนยิ่งขึ้นดังนี้

1. **การระบุช่วงแถวที่ต้องการ** เช่น ต้องการข้อมูลแถวที่ 1 ถึง 3 ผู้อ่านต้องระบุช่วง [0:3]

Python – Jupyter Notebook

```
import pandas as pd

df = pd.read_excel('Product.xlsx')
df[0:3]
```



	รหัสสินค้า	ชื่อสินค้า	ต้นทุน	ราคาขาย
0	A0001	การใช้งานภาษา C#	290	325
1	A0002	พัฒนาเว็บแอพด้วย ASP.NET Core MVC	270	299
2	A0003	การใช้งานฐานข้อมูล SQL Server 2019	300	359

รูปที่ 5-8 แสดงข้อมูล 3 แถว

2. **การระบุคอลัมน์และระบุแถว** ทำได้โดยการระบุชื่อคอลัมน์และระบุช่วงแถวที่ต้องการ เช่น ต้องการข้อมูลคอลัมน์รหัสสินค้าและชื่อสินค้า จำนวน 3 แถวแรก มีโค้ดดังต่อไปนี้

Python – Jupyter Notebook

```
import pandas as pd

df = pd.read_excel('Product.xlsx')
df[['รหัสสินค้า', 'ชื่อสินค้า']][0:3]
```

บทที่ 6

ทำงานกับข้อมูลในไฟล์ CSV

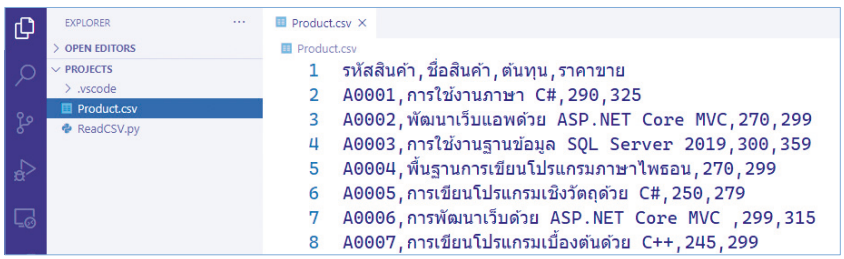
ไฟล์ CSV มีนามสกุลไฟล์ .csv ถือเป็นแหล่งข้อมูลอีกประเภทหนึ่งที่มีความนิยมสูงมาก ข้อมูลที่อยู่ในไฟล์ CSV มีรูปแบบที่เป็นมาตรฐานชัดเจน ส่งผลให้ได้ข้อมูลที่มีคุณภาพสูงอีกรูปแบบหนึ่ง

พื้นฐานการอ่านข้อมูลจากไฟล์ CSV ด้วยฟังก์ชัน read_csv() ของ Pandas

ไฟล์ CSV โดยปกติแล้วใช้เครื่องหมาย , คั่นข้อมูลแต่ละส่วน ส่งผลให้เกิดการแบ่งส่วนของข้อมูลอย่างชัดเจน จึงทำให้ข้อมูลมีรูปแบบแน่นอนเช่นเดียวกับไฟล์ Excel

การทำงานกับไฟล์ประเภท CSV ด้วยภาษาไพธอน ผู้อ่านสามารถใช้ฟังก์ชันที่ชื่อว่า **read_csv()** ของ Pandas เข้ามาทำหน้าที่อ่านข้อมูลที่ได้จากไฟล์ CSV

โปรแกรม Visual Studio Code สามารถเปิดไฟล์ CSV ได้โดยตรง เพราะว่าเป็นไฟล์ข้อความธรรมดา ในกรณีนี้มีไฟล์เก็บข้อมูลสินค้าที่ชื่อว่า Product.csv เก็บอยู่ในโฟลเดอร์เดียวกับไฟล์โค้ดภาษาไพธอน เป็นไฟล์ที่มีข้อมูลเป็นภาษาไทยร่วมอยู่ด้วย ดังรูปที่ 6-1



รูปที่ 6-1 แสดงข้อมูลสินค้าที่อยู่ในไฟล์ Product.csv

การจัดการข้อมูล (CRUD) ในฐานข้อมูล SQL Server ด้วยไพธอน

การทำงานพื้นฐานกับข้อมูลประกอบด้วย 4 อย่าง คือ การเพิ่ม (Create), การอ่านข้อมูล (Read), การแก้ไขข้อมูล (Update) และการลบข้อมูล (Delete) เรียกว่า **CRUD**

ในทางปฏิบัติแล้ว ข้อมูลที่เราต้องทำงานด้วยไม่ได้มาจากไฟล์ของโปรแกรมต่างๆ เพียงอย่างเดียว แต่ยังรวมถึงข้อมูลที่ถูกจัดเก็บอยู่ในระบบฐานข้อมูลด้วย เราต้องรู้วิธีจัดการข้อมูลเหล่านี้ด้วยภาษาไพธอนก่อน

การอ่านข้อมูลจากฐานข้อมูล SQL Server

การทำงานในระดับพื้นฐานที่สุดคือ เราต้องรู้จักวิธีการ Import ข้อมูลจากตาราง Product เข้ามาทำงานในโปรเจกต์ของเราให้ได้ก่อน กล่าวได้อีกนัยหนึ่งว่า เราต้องการอ่านข้อมูล (Read) จากตารางในฐานข้อมูลนั่นเอง

ตัวอย่างที่ 7-1 การอ่านข้อมูลจากฐานข้อมูล SQL Server

ผู้เขียนต้องการอ่านรายการสินค้าจากตาราง Product แบบไม่มีเงื่อนไข มีโค้ดดังต่อไปนี้

โค้ดไพธอนที่ 7-1 การอ่านข้อมูลจากฐานข้อมูล SQL Server (CRUD_Read.py)

```
import pyodbc

strConn = 'driver=ODBC Driver 17 for SQL Server;server=.\SQLExpress;database=thaivbBiz;
trusted_connection=yes;charset=UTF8'
sql = 'SELECT ProductID, ProductName, ProductCost, ProductPrice FROM Product'

print('รหัสสินค้า' + '\tชื่อสินค้า' + '\t\tต้นทุน' + '\t\tราคาขาย')
print('-------------------------------------------------------------')
with pyodbc.connect(strConn) as Conn:
    data = Conn.execute(sql)
    for item in data:
        print(str(item[0]).ljust(10) + '\t' + str(item[1]).ljust(20)
              + str(item[2]) + '\t' + str(item[3]))

print(type(data))
```

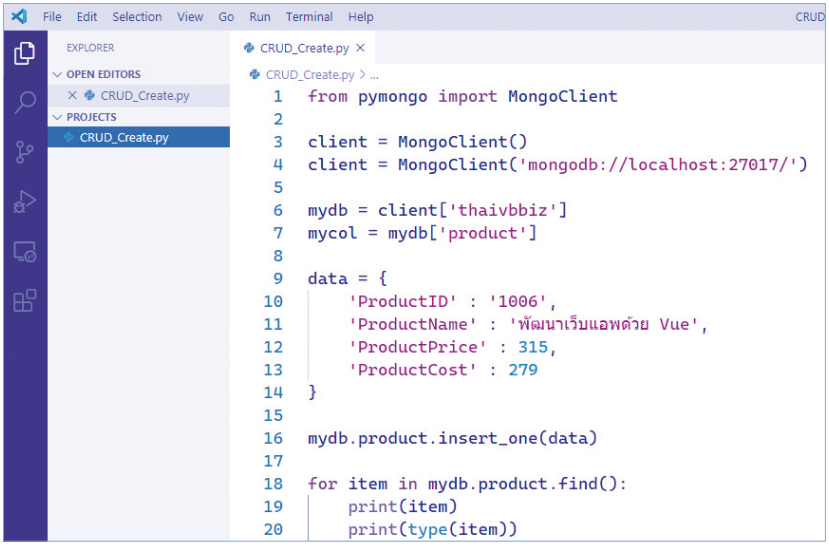
การเพิ่มข้อมูลใหม่เข้าไปในฐานข้อมูล MongoDB

ตัวอย่างที่ 8-1 การเพิ่มข้อมูลในฐานข้อมูล MongoDB

ผู้เขียนต้องการเพิ่มรายการสินค้าใหม่เข้าไปในคอลเล็คชันที่ชื่อว่า product ด้วยภาษาไพธอน รหัส 1006 จำนวน 1 เล่ม ดังโค้ดต่อไปนี้

โค้ดไพธอนที่ 8-1 การเพิ่มข้อมูลในฐานข้อมูล MongoDB (CRUD_Create.py)

```
from pymongo import MongoClient

client = MongoClient()
client = MongoClient('mongodb://localhost:27017/')

mydb = client['thaivbbiz']
mycol = mydb['product']

data = {
    'ProductID' : '1006',
    'ProductName' : 'พัฒนาเว็บแอพด้วย Vue',
    'ProductPrice' : 315,
    'ProductCost' : 279
}

mydb.product.insert_one(data)

for item in mydb.product.find():
    print(item)
    print(type(item))
```



บทที่ 9

การทำงานกับข้อมูลสูญหายและค่าผิดปกติ

ความต้องการสูงสุดของข้อมูล นั่นคือ ข้อมูลที่อยู่ในสถานะสมบูรณ์แบบ, ไม่มีข้อมูลผิดพลาด, ข้อมูลครบถ้วน และข้อมูลถูกต้อง เช่น อายุของพนักงานต้องมีทุกคน มีค่าติดลบไม่ได้, อายุเกิน 150 ปีไม่ได้ เป็นต้น เห็นได้ว่า ปัญหาข้อมูลสูญหายถือเป็นปัญหาพื้นฐานข้อแรกที่ผู้อ่านหลีกเลี่ยงไม่ได้ เราจะมาดูกันว่า เราสามารถจัดการปัญหานี้อย่างไรได้บ้าง

ปัญหาของข้อมูลสูญหาย

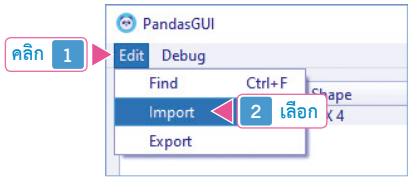
มีข้อมูลพนักงาน 4 คน ประกอบด้วย Id, Name, Salary และ Age เก็บอยู่ในตัวแปรที่ชื่อว่า data พบว่า ข้อมูลชุดนี้มีปัญหาข้อมูลสูญหายที่เกิดมาจากค่า pn.nan, pd.NaT และ None ดังโค้ดต่อไปนี้

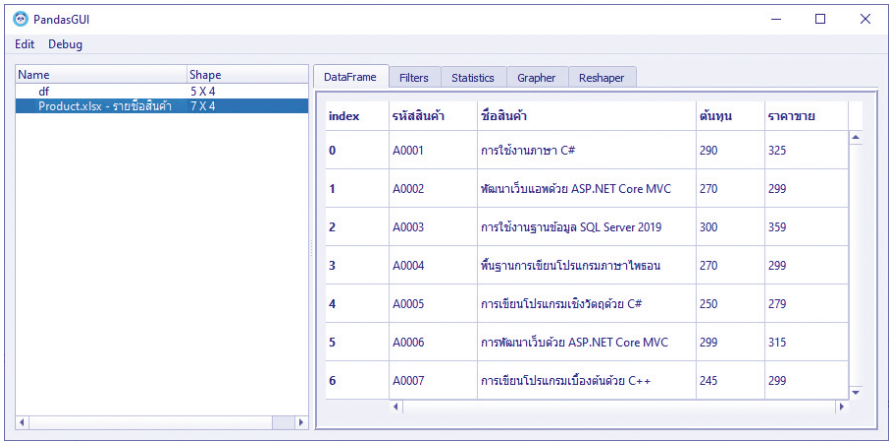
Python – Jupyter Notebook

```
import pandas as pd
import numpy as np

data = {
        'Id' : [100, 200, 300, 400],
        'Name' : ['ศุภชัย', 'สมใจ', 'ศักดิ์ชัย', 'พรชัย'],
        'Salary' : [20000, 22000, np.nan, pd.NaT],
        'Age' : [35, 36, np.nan, None]
    }

df = pd.DataFrame(data)
df
```

การ Import ข้อมูลจากไฟล์ CSV และ Excel

ในกรณีที่ผู้อ่านต้องการ Import ข้อมูลจากไฟล์ CSV หรือ Excel ก็สามารถทำได้เช่นกัน โดยการคลิกเมนู Edit > Import ดังรูปที่ 10-6



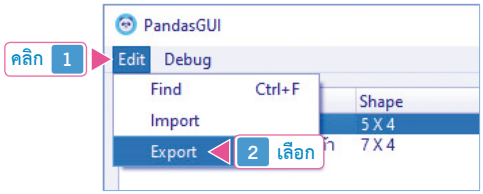


รูปที่ 10-6 แสดงข้อมูลที่ได้จากไฟล์ Excel

จากรูปที่ 10-6 พบว่าเราทำงานกับข้อมูล 2 ชุด คือ ข้อมูลที่ได้จากตัวแปร df และข้อมูลที่ได้จากไฟล์ Excel

การ Export ข้อมูลออกเป็นไฟล์ CSV

ผู้อ่านสามารถ Export ข้อมูลของผู้อ่านออกเป็นไฟล์ CSV ได้อีกด้วย ในกรณีนี้ต้องการ Export ข้อมูลที่อยู่ในตัวแปร df เป็นไฟล์ CSV ผลที่ได้คือ PandasGUI จะใช้เครื่องหมาย , คั่นข้อมูลระหว่างคอลัมน์ ดังรูปที่ 10-7



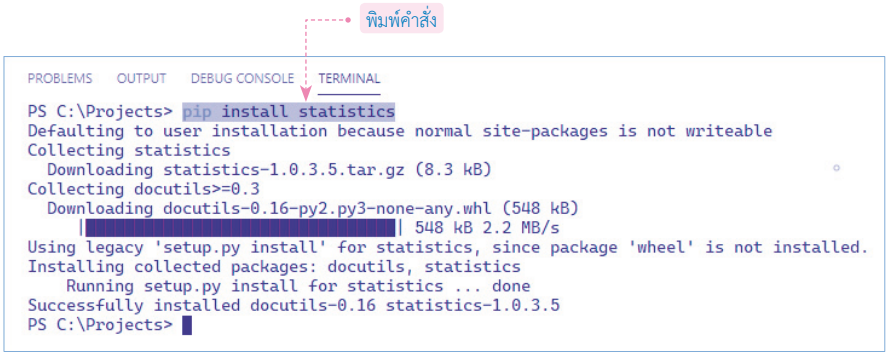
การหาค่าเฉลี่ยเลขคณิตกับตัวเลขทศนิยมด้วยฟังก์ชัน fmean() ของ statistics

ในกรณีที่ผู้อ่านมีข้อมูลที่เป็นตัวเลขทศนิยม ผู้อ่านสามารถหาค่าทางสถิติเบื้องต้น โดยอาศัยกลุ่มฟังก์ชันของ Package ที่ชื่อว่า **statistics** เข้ามาทำงานด้านนี้โดยเฉพาะเลยก็ได้

ให้ผู้อ่านพิมพ์คำสั่งต่อไปนี้ใน TERMINAL ของโปรแกรม Visual Studio Code เพื่อดาวน์โหลดและติดตั้ง statistics ก่อน

TERMINAL

```
pip install statistics
```



รูปที่ 11-7 แสดงการดาวน์โหลดและติดตั้ง statistics

ใน statistics มีฟังก์ชันที่ชื่อว่า **fmean()** ทำหน้าที่คำนวณค่าเฉลี่ยเลขคณิตกับข้อมูลที่เป็นตัวเลขทศนิยมโดยเฉพาะ ดังโค้ดต่อไปนี้

Python

```
import statistics as st

data = [25.10, 40.68, 35.99, 23.4]
result = st.fmean(data)

print('{0:.3f}'.format(result))
```

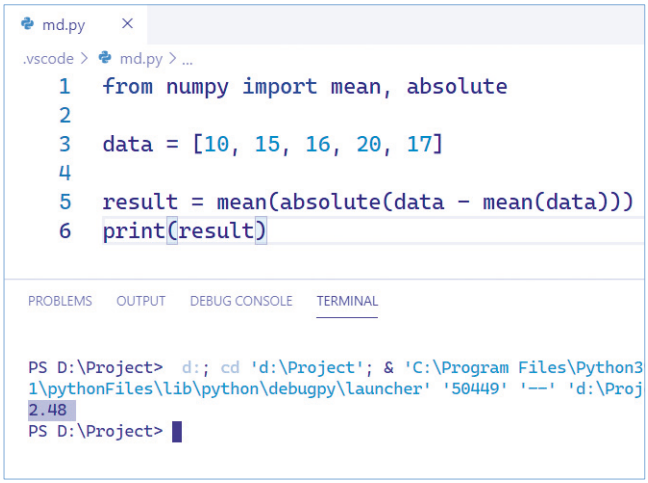
ตัวอย่างที่ 13-3 การคำนวณส่วนเบี่ยงเบนเฉลี่ยใน NumPy

ในกรณีที่ผู้อ่านต้องการใช้งาน NumPy เข้ามาคำนวณส่วนเบี่ยงเบนเฉลี่ย ต้องใช้ 2 ฟังก์ชัน คือ

- **ฟังก์ชัน mean()** สำหรับคำนวณค่าเฉลี่ยเลขคณิต
- **ฟังก์ชัน absolute()** สำหรับตัดเครื่องหมายลบออกจากผลต่าง ดังโค้ดต่อไปนี้

ตัวอย่างที่ 13-3 การคำนวณส่วนเบี่ยงเบนเฉลี่ยใน NumPy (md.py)

```
from numpy import mean, absolute

data = [10, 15, 16, 20, 17]

result = mean(absolute(data - mean(data)))
print(result)
```



รูปที่ 13-3 แสดงส่วนเบี่ยงเบนเฉลี่ยของ NumPy

อธิบายการทำงานของโค้ด

1. เริ่มต้นให้หาผลต่างของข้อมูลแต่ละตัวกับค่าเฉลี่ยเลขคณิต (mean(data)) ก่อน แล้วใช้ฟังก์ชัน absolute() เข้ามาทำหน้าที่ตัดเครื่องหมายลบออก

md.py

```
absolute(data - mean(data))
```

ตัวอย่างที่ 13-9 การคำนวณค่าสัมประสิทธิ์ของส่วนเบี่ยงเบนเฉลี่ยใน NumPy มีโค้ดดังต่อไปนี้

โค้ดไพธอนที่ 13-9 การคำนวณค่าสัมประสิทธิ์ของส่วนเบี่ยงเบนเฉลี่ยใน NumPy (ca.py)

```
from numpy import mean, absolute

a = [20, 11, 12, 10, 9]
b = [15, 16, 17, 18, 17]

mean_a = mean(a)
md_a = mean(absolute(a - mean(a)))
cv_a = md_a/mean_a

mean_b = mean(b)
md_b = mean(absolute(b - mean(b)))
cv_b = md_b/mean_b

print('ค่าเฉลี่ย A : ', mean_a)
print('ส่วนเบี่ยงเบนเฉลี่ย A : ', md_a)
print('สัมประสิทธิ์ส่วนเบี่ยงเบนเฉลี่ย A : ', cv_a)
print('----------------------------------------')
print('ค่าเฉลี่ย B : ', mean_b)
print('ส่วนเบี่ยงเบนเฉลี่ย B : ', md_b)
print('สัมประสิทธิ์ส่วนเบี่ยงเบนเฉลี่ย B : ', cv_b)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS D:\Project>  d:; cd 'd:\Project'; & 'C:\Program
61\pythonFiles\lib\python\debugpy\launcher' '49980
ค่าเฉลี่ย A :  12.4
ส่วนเบี่ยงเบนเฉลี่ย A :  3.04
สัมประสิทธิ์ส่วนเบี่ยงเบนเฉลี่ย A :  0.24516129032258063
----------------------------------------
ค่าเฉลี่ย B :  16.6
ส่วนเบี่ยงเบนเฉลี่ย B :  0.8799999999999997
สัมประสิทธิ์ส่วนเบี่ยงเบนเฉลี่ย B :  0.05301204819277106
PS D:\Project>
```

รูปที่ 13-10 ผลการคำนวณสัมประสิทธิ์ของส่วนเบี่ยงเบนเฉลี่ย

อธิบายการทำงานของโค้ด

1. ที่ข้อมูลชุด A คำนวณค่าเฉลี่ยเลขคณิตและส่วนเบี่ยงเบนเฉลี่ยก่อน กล่าวคือ
   - **ค่าเฉลี่ยเลขคณิต** ใช้ฟังก์ชัน mean() เก็บไว้ที่ตัวแปร mean_a
   - **ส่วนเบี่ยงเบนเฉลี่ย** ใช้ฟังก์ชัน mean() ร่วมกับฟังก์ชัน absolute() เก็บไว้ที่ตัวแปร md_a
   - **สัมประสิทธิ์ส่วนเบี่ยงเบนเฉลี่ย** โดยการนำตัวแปร md_a หารด้วยตัวแปร mean_a

การกำหนดความเข้มจางของจุดข้อมูล

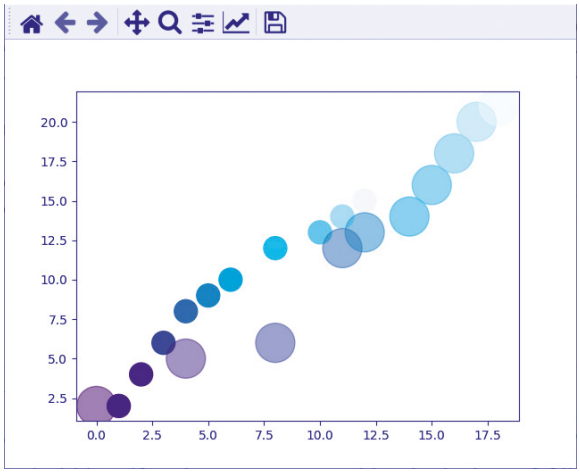
ในกรณีที่ผู้อ่านมีข้อมูลจำนวนมาก มีโอกาสสูงมากที่ข้อมูลจะอยู่ในตำแหน่งทับซ้อนกัน ทำให้การสื่อสารข้อมูลอาจจะไม่ชัดเจน สื่อสารไม่ครบถ้วน วิธีการแก้ไขปัญหานี้ก็คือ ผู้อ่านสามารถกำหนดให้ข้อมูลแสดงผลด้วยความเข้มจางแตกต่างกัน

ตัวอย่างที่ 14-4 การกำหนดความเข้มจางของจุดข้อมูล

ผู้เขียนกำหนดให้ข้อมูลชุดที่ 1 แสดงผลตามปกติ ส่วนข้อมูลชุดที่ 2 กำหนดให้แสดงผล 50% (จางกว่าปกติหรือกึ่งโปร่งใส) โดยอาศัยพารามิเตอร์ที่ชื่อว่า **alpha** กำหนดค่า 0.5 ดังโค้ดต่อไปนี้

โค้ดไพธอนที่ 14-4 การกำหนดความเข้มจางของจุดข้อมูล (MultipleData.py)

```
import matplotlib.pyplot as plt
import numpy as np

x1 = [1, 2, 3, 4, 5, 6, 8, 10, 11, 12]
y1 = [2, 4, 6, 8, 9, 10, 12, 13, 14, 15]

x2 = [0, 4, 8, 11, 12, 14, 15, 16, 17, 18]
y2 = [2, 5, 6, 12, 13, 14, 16, 18, 20, 21]

colors = np.array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90])

plt.scatter(x1, y1, c=colors, s=350)
plt.scatter(x2, y2, c=colors, s=1000, alpha=0.5)
plt.show()
```



รูปที่ 14-7 แสดงข้อมูลที่ทับซ้อนกันอยู่

จากรูปที่ 14-7 เมื่อข้อมูลทั้ง 2 ชุดมีความเข้มจางแตกต่างกัน ทำให้การดูข้อมูลชัดเจนยิ่งขึ้น

การสร้างกราฟแท่ง (Bar Graph)

กราฟแท่ง (Bar Graph) ใช้ในการแสดงความแตกต่างของข้อมูลได้อย่างชัดเจน เป็นกราฟอีกรูปแบบหนึ่งที่เราคุ้นเคยกันเป็นอย่างดี เป็นหน้าที่ของฟังก์ชันที่ชื่อว่า **bar()**

ตัวอย่างที่ 14-5 พื้นฐานการสร้างกราฟแท่ง

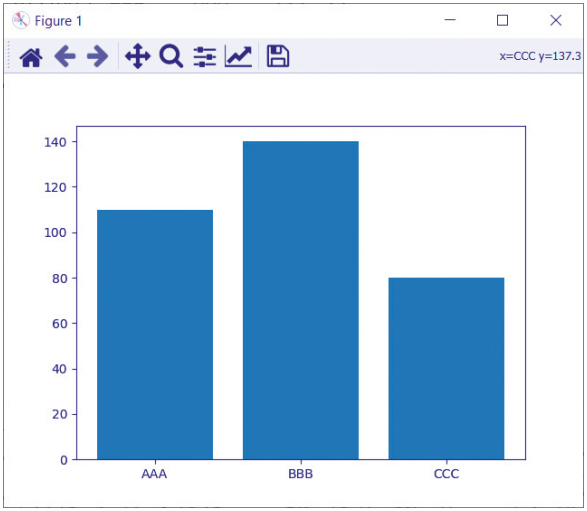
มีข้อมูล 3 ชุดของนาย AAA, BBB และ CCC ต้องการแสดงข้อมูลในรูปแบบกราฟแท่ง ดังโค้ดต่อไปนี้

โค้ดไพธอนที่ 14-5 พื้นฐานการสร้างกราฟแท่ง (UsingBar.py)

```
import matplotlib.pyplot as plt
import numpy as np

name = np.array(["AAA", "BBB", "CCC"])
data = np.array([110, 140, 80])

plt.bar(name, data)
plt.show()
```

โดยปกติแล้ว เราจะได้กราฟแท่งตามแนวตั้ง โดยที่

- **แกน x** แสดงชื่อ ได้จากตัวแปร name
- **แกน y** แสดงค่าของข้อมูลแต่ละตัว ได้จากตัวแปร data ดังรูป 14-8



รูปที่ 14-8 แสดงกราฟแท่งแนวตั้ง

การแบ่งสัดส่วนข้อมูลออกจากส่วนของกราฟวงกลม

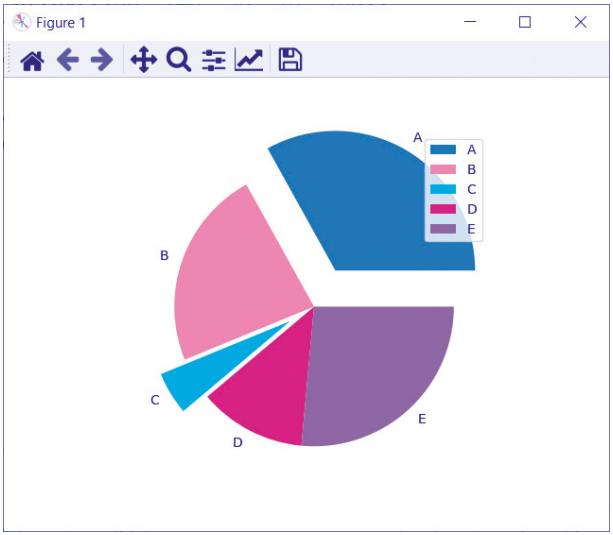
ในกรณีที่ผู้อ่านต้องการเน้นสัดส่วนของข้อมูลบางตัว ให้โดดเด่นมากกว่าส่วนอื่นๆ ก็สามารถกำหนดให้แบ่งส่วนดังกล่าว แสดงผลออกห่างจากกราฟวงกลมโดยการกำหนดค่า **explode** เช่น

ต้องการกำหนดให้ข้อมูลตัวที่ 1 (A) กับตัวที่ 3 (C) แบ่งส่วนออกไป โดยการสร้างตัวแปรที่ชื่อว่า data_explode ค่าที่กำหนดขึ้นมายิ่งมีค่ามากทำให้มีระยะห่างมากขึ้น จากนั้นนำไปกำหนดค่าให้กับพารามิเตอร์ explode ดังโค้ดต่อไปนี้

ตัวอย่างที่ 14-7 การแบ่งสัดส่วนข้อมูลออกจากส่วนของกราฟวงกลม

โค้ดไพธอนที่ 14-7 การแบ่งสัดส่วนข้อมูลออกจากส่วนของกราฟวงกลม (UsingPie.py)

```
import matplotlib.pyplot as plt
import numpy as np

data = np.array([200, 140, 30, 75, 160])
data_label = ['A', 'B', 'C', 'D', 'E']
data_explode = [0.3, 0, 0.2, 0, 0]

plt.pie(data, labels = data_label, explode=data_explode)
plt.legend()
plt.show()
```



รูปที่ 14-13 กราฟวงกลมที่มีการแบ่งสัดส่วนออกไป

การสร้างกราฟ BoxPlot

กราฟ BoxPlot ทำหน้าที่บอกผู้อ่านว่า ข้อมูลที่เราสนใจอยู่มีการกระจายตัวมากน้อยเพียงใด ก็จะช่วยทำให้มองข้อมูลได้ครบถ้วนยิ่งขึ้น

ตัวอย่างที่ 14-15 การสร้างกราฟ BoxPlot

เพื่อให้ผู้อ่านเข้าใจวิธีการดูกราฟแบบ BoxPlot ผู้เขียนเลือกใช้ข้อมูลแบบปกติ มีลักษณะเกาะกลุ่มกัน กระจายตัวน้อยเป็นข้อมูลที่ไม่มีค่ามากเกินไปหรือน้อยเกินไป (Outliers) คือ 42, 40, 45, 48, 50 และ 45 จากนั้นนำข้อมูลชุดนี้มาสร้าง BoxPlot ดังโค้ดต่อไปนี้

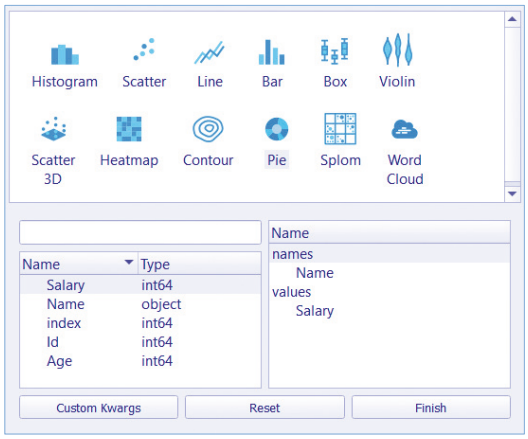
โค้ดไพธอนที่ 14-15 การสร้างกราฟ BoxPlot (UsingBoxPlot.py)

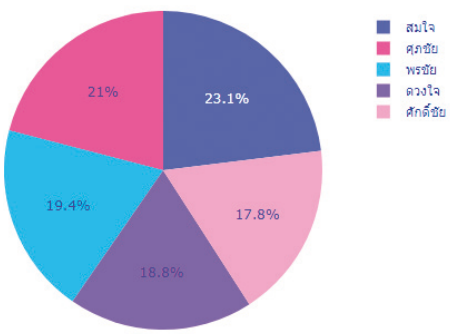
```
import matplotlib.pylab as plt
import numpy as np

data = [42, 40, 45, 48, 50, 45]
fig = plt.figure(1, figsize=(10, 6))
ax = fig.add_subplot(111)

ax.boxplot(data, vert=False, manage_ticks=True)
ax.set_xlabel('Values')
ax.set_yticklabels(['Products'])

q = np.quantile(data, [0.00, 0.25, 0.50, 0.75, 1.00])
ax.vlines(q, ymin=0, ymax=100, color='red', ls=':')

ax.set_xticks(q)
ax.set_ylim(0.5, 1.5)
plt.show()
```

การสร้างกราฟวงกลม (Pie Graph)

การสร้างกราฟวงกลม (Pie Graph) เป็นกราฟที่ทำหน้าที่แสดงสัดส่วนข้อมูลเพื่อเปรียบเทียบว่า ข้อมูลใดมีมากน้อยกว่ากัน ต้องการข้อมูล 2 อย่าง คือ

- **names** หมายถึง ชื่อของข้อมูล ในกรณีนี้คือ ชื่อพนักงาน Name
- **values** หมายถึง ข้อมูลที่ต้องการแสดงค่า ในกรณีนี้คือ เงินเดือน Salary





รูปที่ 15-9 แสดงกราฟวงกลม

สรุปท้ายเล่ม

ในการทำงานกับข้อมูลด้วยภาษาไพธอน ที่ผู้เขียนเลือกมานำเสนอในหนังสือเล่มนี้ มีเจตนาต้องการให้ผู้อ่านทราบว่า ข้อมูลที่ผู้อ่านกำลังสนใจอยู่ ล้วนแล้วแต่มีสิ่งที่ถูกปกปิดซ่อนอยู่ภายใน ใครสามารถใช้ประโยชน์จากสิ่งที่ซ่อนอยู่ ย่อมได้เปรียบกว่าการไม่ทราบอะไรเลย อาจจะเป็นเพียงเนื้อหาในขั้นต้น ไม่สามารถนำเสนอได้ครบถ้วนทุกแง่มุม แต่อย่างน้อยก็อยากให้เป็นรากฐานไปสู่ในระดับต่อไป... สวัสดีครับ