บทที่ 8
การจัดการและวิเคราะห์ข้อมูลแบบตาราง

Contents

ข้อมูลแบบตาราง (tabular data) เป็นรูปแบบข้อมูลที่ได้รับความนิยมอย่างมากในงานประมวลผลและวิเคราะห์ข้อมูล เนื่องจากมีโครงสร้างที่เป็นระเบียบและการจัดเก็บข้อมูลที่สามารถเข้าถึงได้ง่าย แต่ละแถวแทนหนึ่งรายการหรือตัวอย่าง และแต่ละคอลัมน์มีค่าของตัวแปรหนึ่งตัว ซึ่งช่วยให้สามารถจัดการและวิเคราะห์ข้อมูลได้อย่างมีประสิทธิภาพ เราจึงจัดว่าข้อมูลแบบตารางเป็นข้อมูลที่มีโครงสร้าง (structured data) และเป็นรูปแบบการจัดเก็บข้อมูลในอุดมคติ พร้อมสำหรับการวิเคราะห์

ในการจัดการกับข้อมูลชนิดนี้ เรามักใช้ pandas ซึ่งเป็นไลบรารีของภาษาไพทอน ที่ประกอบไปด้วยเครื่องมือระดับสูงสำหรับการจัดการข้อมูลแบบตาราง ไลบรารี pandas ทำงานร่วมกับข้อมูลขนาดใหญ่ได้ดี รวมถึงมีฟังก์ชันต่าง ๆ สำหรับการเลือก กรอง และปรับแต่งข้อมูลได้อย่างง่ายดาย นอกจากนี้ยังสามารถรวมข้อมูลจากแหล่งข้อมูลหลายแห่งและรองรับการอ่านและเขียนไฟล์ข้อมูลในรูปแบบต่าง ๆ ได้อย่างหลากหลาย ทำให้เป็นเครื่องมือที่ทรงพลังและได้รับความนิยมสูงในการวิเคราะห์ข้อมูลแบบตาราง การใช้ pandas นับเป็นทักษะที่สำคัญที่สุดอย่างหนึ่งของนักวิเคราะห์ข้อมูล โดยเฉพาะอย่างยิ่งข้อมูลที่เป็นข้อความ และข้อมูลที่มีขนาดใหญ่เกินกว่าที่จะเปิดด้วยโปรแกรมวิเคราะห์ข้อมูลทั่วไป เช่น SPSS Microsoft Excel หรือ Google Spreadsheet

หากเราติดตั้งไพทอนผ่าน Anaconda เราจะได้ไลบรารี pandas มาพร้อมกับการติดตั้ง แต่หากเราใช้งานไพทอนผ่านการติดตั้งแบบอื่น เราสามารถติดตั้งไลบรารี pandas ได้โดยการใช้คำสั่ง `pip install pandas` หรือ `conda install pandas` (หากติดตั้ง miniconda) เมื่อติดตั้งเสร็จเรียบร้อยแล้ว เราสามารถนำเข้าไลบรารี pandas ได้โดยการใช้คำสั่ง `import pandas as pd` ซึ่งจะทำให้เราสามารถใช้ฟังก์ชันและเมท็อดที่อยู่ในไลบรารี pandas ได้

เราอาจจะสงสัยว่าเพราะเหตุใดเราต้องเปลี่ยนชื่อไลบรารีเป็น `pd` โดยการรัน `import pandas as pd` ซึ่งเราไม่จำเป็นต้องเปลี่ยนชื่อก็สามารถใช้ได้เหมือนกัน แต่เราเลือกเปลี่ยนชื่อเป็น `pd` เพื่อให้เราสามารถเขียนโค้ดได้สั้นลง นอกจากนั้นเราไม่ใช้ชื่ออื่นเช่น `p` หรือ `pan` หรือชื่ออื่น ๆ ที่เราเลือกเอง เหตุก็เพราะว่า `pd` จะเป็นชื่อที่คนในวงการวิทยาการข้อมูลใช้กันมากที่สุด อย่างที่เราเห็นได้ว่าหากเราไปค้นหาวิธีการใช้งานของ pandas ในเว็บไซต์ที่รวมชุมชนนักวิเคราะห์ข้อมูล เช่น stackoverflow.com หรือ medium.com จะพบว่าคำตอบจะใช้ `pd` ในการอ้างถึงไลบรารี pandas ทั้งสิ้นด้วย ดังนั้นการใช้ชื่อ `pd` เป็นการตั้งชื่อตามสมัยนิยม ตามธรรมเนียมปฏิบัติ และเป็นการให้ความสะดวกในการเรียนรู้จากแหล่งข้อมูลอื่น ๆ ที่มีอยู่ในอินเตอร์เน็ต

การเตรียมและวิเคราะห์ข้อมูลด้วย Jupyter Notebook

Jupyter Notebook เป็น IDE ที่ถูกออกแบบมาเพื่อการวิเคราะห์ข้อมูลโดยเฉพาะ เพราะว่าการวิเคราะห์ข้อมูลจำเป็นต้องมีการบันทึกว่าเราได้แก้ไขชุดข้อมูลอย่างไรบ้าง รันโค้ดอะไรบ้างที่นำไปสู่ผลการวิเคราะห์ เราจึงเรียกว่า notebook เพราะเปรียบเสมือนการจดบันทึกการวิเคราะห์ข้อมูล เพื่อให้ฟังดูคล้ายคลึงนักวิทยาศาสตร์จดบันทึกผลการทดลอง ด้วยเหตุนี้เอง Jupyter Notebook จึงมีการออกลูกเล่นฟีเจอร์ใหม่อย่างต่อเนื่อง เพื่ออำนวยความสะดวกในการวิเคราะห์ข้อมูล เช่น

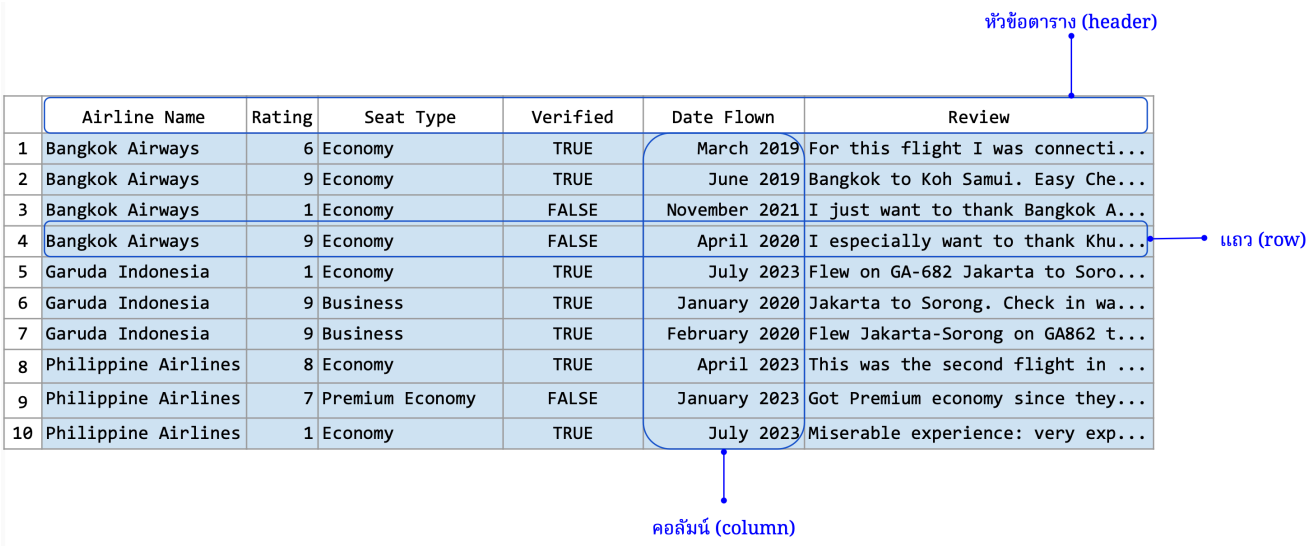
- แสดงผลข้อมูลตารางออกมาในรูปแบบที่อ่านได้ง่าย ไม่ล้นออกมาทั้งแนวตั้งและแนวนอน ทำให้ตรวจสอบข้อมูลได้ง่าย ข้อผิดพลาดน้อยลง
- แสดงแผนภูมิออกมาตามคำสั่งของไลบรารี โดยไม่ต้องเก็บภาพใส่ไฟล์แยกต่างหาก ทำให้สามารถแก้ไขและปรับแต่งแผนภูมิได้ง่าย ทั้งแผนภูมิ และโค้ดที่ใช้สร้างแผนภูมิอยู่ข้างกัน
- สามารถใช้ extension บน Visual Studio Code หรือ Google Colab ที่ทำให้วิเคราะห์ข้อมูลเชิงโต้ตอบ (interactive) ได้คล้ายคลึงกับการใช้โปรแกรมวิเคราะห์ข้อมูลอื่น ๆ ที่มี user interface สวยงามและสะดวกในการใช้งาน

ด้วยข้อดีดังกล่าวข้างต้นวงการวิทยาการข้อมูลจึงนิยมใช้ pandas บน Jupyter Notebook ในการเตรียมข้อมูลและวิเคราะห์ข้อมูล ซึ่งจัดเป็นแนวปฏิบัติที่ดีที่สุดในขณะนี้

ข้อมูลแบบตาราง

ข้อมูลแบบตารางมีส่วนประกอบหลักที่สำคัญอยู่หลายส่วน ซึ่งแต่ละส่วนมีบทบาทสำคัญในการเก็บรักษาและการจัดการข้อมูล

1. แถว (row) แสดงถึงรายการหรือตัวอย่างแต่ละชิ้น ซึ่งสามารถจัดเก็บ อธิบายลักษณะหรือสถานะของเหตุการณ์ วัตถุ หรือบุคคลที่กำลังศึกษาอยู่ ตัวอย่างเช่น ในตารางข้อมูลของผู้ป่วย แต่ละแถวอาจแทนผู้ป่วยหนึ่งคนพร้อมกับรายละเอียดต่าง ๆ เช่น อายุ เพศ และประวัติการรักษา ดังนั้นจำนวนแถวที่อยู่ในตารางจึงเท่ากับจำนวนตัวอย่างที่อยู่ในชุดข้อมูล
2. คอลัมน์ (column) แสดงถึงตัวแปรหรือคุณลักษณะที่ระบุค่าข้อมูลต่าง ๆ ที่จำเป็นต่อการศึกษาหรือวิเคราะห์ ตัวอย่างเช่น ในตารางข้อมูลการขาย คอลัมน์อาจประกอบด้วย วันที่ จำนวนเงิน และสินค้าที่ขาย
3. หัวตาราง (header) เป็นบรรทัดแรกของตารางที่ระบุชื่อของแต่ละคอลัมน์ เป็นส่วนที่สำคัญเพราะช่วยให้ผู้ใช้สามารถเข้าใจข้อมูลในแต่ละคอลัมน์ได้ง่ายและชัดเจน
4. ตัวข้อมูล (data) เป็นสาระสำคัญที่แสดงถึงค่าเฉพาะของแต่ละรายการตามที่ถูกกำหนดไว้ในคอลัมน์ ข้อมูลนี้สามารถเป็นตัวเลข ข้อความ วันที่ หรือชนิดข้อมูลอื่น ๆ ที่เหมาะสมกับลักษณะของการศึกษา วิเคราะห์ เพราะฉะนั้นข้อมูลที่อยู่ในคอลัมน์เดียวกันจะต้องเป็นแบบชนิดข้อมูลเดียวกัน เนื่องจากเป็นตัวแปรชนิดเดียวกัน



	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	Bangkok Airways	6	Economy	TRUE	March 2019	For this flight I was connecti...
2	Bangkok Airways	9	Economy	TRUE	June 2019	Bangkok to Koh Samui. Easy Che...
3	Bangkok Airways	1	Economy	FALSE	November 2021	I just want to thank Bangkok A...
4	Bangkok Airways	9	Economy	FALSE	April 2020	I especially want to thank Khu...
5	Garuda Indonesia	1	Economy	TRUE	July 2023	Flew on GA-682 Jakarta to Soro...
6	Garuda Indonesia	9	Business	TRUE	January 2020	Jakarta to Sorong. Check in wa...
7	Garuda Indonesia	9	Business	TRUE	February 2020	Flew Jakarta-Sorong on GA862 t...
8	Philippine Airlines	8	Economy	TRUE	April 2023	This was the second flight in ...
9	Philippine Airlines	7	Premium Economy	FALSE	January 2023	Got Premium economy since they...
10	Philippine Airlines	1	Economy	TRUE	July 2023	Miserable experience: very exp...

ภาพที่ 24 ส่วนประกอบที่สำคัญของข้อมูลแบบตาราง (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

ส่วนประกอบอีกอย่างหนึ่งที่สำคัญของข้อมูลแบบตารางคือ พจนานุกรมข้อมูล (data dictionary) ซึ่งเป็นเอกสารที่ระบุความหมายของแต่ละคอลัมน์ และค่าข้อมูลที่เป็นไปได้ของแต่ละคอลัมน์ ซึ่งช่วยให้ผู้ใช้เข้าใจข้อมูลได้ง่ายขึ้น และช่วยให้ผู้ใช้สามารถใช้ข้อมูลได้อย่างถูกต้องและเหมาะสม พจนานุกรมข้อมูลนี้มักจะถูกเก็บแยกออกมาอีกไฟล์หนึ่ง หรือเป็นส่วนหนึ่งของเอกสารคู่มือการใช้งานข้อมูล แต่ในหลาย ๆ กรณีผู้ดูแลรักษาข้อมูลอาจจะไม่ได้เตรียมพจนานุกรมข้อมูลเอาไว้ให้พร้อมใช้ ผู้ใช้ข้อมูลจะต้องไปสอบถามผู้ดูแลรักษาข้อมูลหรือค้นหาข้อมูลเพิ่มเติมเองโดยตรง ตัวอย่างของพจนานุกรมข้อมูลของข้อมูลการรีวิวสายการบิน อาจมีดังนี้

ชื่อคอลัมน์	คำอธิบาย	ค่าที่เป็นไปได้
Airline Name	ชื่อสายการบิน	ชื่อสายการบิน
Rating	คะแนนการรีวิว	1-10
Seat Type	ประเภทที่นั่ง	Economy, Premium Economy, Business, First
Verified	การยืนยันตัวตน	TRUE, FALSE
Date Flown	วันที่เดินทาง	วันที่เดินทางเป็นเดือนและปี

กระบวนการเตรียมข้อมูลสำหรับการวิเคราะห์

ไลบรารี pandas มีฟังก์ชันต่าง ๆ ที่ช่วยในการจัดการข้อมูลแบบตาราง ซึ่งเป็นกระบวนการที่สำคัญในการวิเคราะห์ข้อมูล โดยปกติแล้วกระบวนการเตรียมข้อมูลสำหรับการวิเคราะห์จะประกอบด้วยขั้นตอนดังนี้

1. การทำความสะอาดข้อมูล (data cleaning) คือ การคัดแยกข้อมูลที่สกปรกออกไป เช่น ข้อมูลที่ไม่สมบูรณ์ สั้นเกินไป ยาวเกินไป ผิดเพี้ยน หรือข้อความมีข้อมูลเราไม่ต้องการเช่น url หรือเครื่องหมายวรรคตอน หรือ สิ่งอื่น ๆ ที่อาจจะทำให้การวิเคราะห์ข้อมูลคลาดเคลื่อนไปได้
2. การวิเคราะห์ข้อมูลเชิงสำรวจ (exploratory data analysis: EDA) คือ การวิเคราะห์เพื่อให้เราเข้าใจข้อมูลคร่าว ๆ เนื่องจากข้อมูลมักจะมีขนาดใหญ่มาก เราไม่สามารถนำข้อมูลออกมาดูได้ทั้งหมด เรามักจะทำ EDA โดยดูว่าข้อมูลมีทั้งหมดที่แถว แต่ละคอลัมน์เก็บข้อมูลอะไรอยู่บ้าง แต่ละคอลัมน์มีค่าสูงสุด ต่ำสุดเท่าไร ถ้าเป็นข้อความ ข้อความที่ยาวสุดยาวกี่ตัวอักษร สั้นสุดกี่ตัวอักษร เป็นต้น
3. การขยำข้อมูล (data munging ซึ่งเป็นศัพท์แสลงที่ใช้ในวงการวิทยาการข้อมูล) หรือ การทะเลาะกับข้อมูล (data wrangling ซึ่งเป็นศัพท์ที่ทางการขึ้น) คือ การแปลงข้อมูลจากรูปเดิม นำมาทำความสะอาด และคัดให้เหลืออยู่เฉพาะส่วนที่สำคัญเพื่อนำไปวิเคราะห์ต่ออาทิ ลบคอลัมน์ที่ไม่เกี่ยวข้องกับการวิเคราะห์ เปลี่ยนชื่อคอลัมน์ให้สื่อความหมาย ลบคอลัมน์ที่ซ้ำซ้อนกันออกไป ลบแถวที่มีข้อมูลที่สกปรก คลาดเคลื่อนมากจนใช้การไม่ได้ ลบแถวที่ซ้ำซ้อนกันออกไป แปลงหน่วยวัด แปลงแบบชนิดข้อมูลในแต่ละคอลัมน์ หรือควบรวมเอาข้อมูลจากหลายแหล่งมารวมกัน

รูปแบบการจัดเก็บข้อมูลแบบตาราง

ข้อมูลแบบตารางสามารถเก็บได้ในหลายรูปแบบไฟล์ (file format) ที่มีลักษณะและคุณสมบัติที่ต่างกัน ทำให้เหมาะสมกับการใช้งานในสถานการณ์ที่ต่างกันด้วย รูปแบบไฟล์ที่พบบ่อยมีดังนี้

ไฟล์ CSV (comma-separated value) และ TSV (tab-separated value)

ไฟล์ CSV (comma-separated value) สกุลของไฟล์คือ .csv ถูกจัดเก็บในลักษณะที่ง่ายและเป็นระเบียบเพื่อให้ง่ายต่อการเข้าใจและใช้งานได้กับโปรแกรมหลากหลายชนิด หลักการหลักของการเก็บข้อมูลแบบ CSV ได้แก่

1. ใช้เครื่องหมายจุลภาค (comma) เป็นอักขระคั่น (delimiter) ซึ่งเป็นอักขระที่ทำหน้าที่ตัวแบ่งระหว่างข้อมูลในแต่ละคอลัมน์
2. ใช้สัญลักษณ์พิเศษ `\n` ในการบ่งบอกว่าข้อมูลในแถวนั้นจบลงแล้ว และข้อมูลในแถวถัดไปจะเริ่มต้นขึ้นใหม่ถ้าหากยังมีข้อมูลเพิ่มเติมอีก
3. ข้อมูลที่เป็นข้อความจะถูกครอบด้วยเครื่องหมายคำพูด เพื่อป้องกันการสับสนกับอักขระคั่น หรืออักขระที่ใช้ในการขึ้นบรรทัดใหม่

โดยปกติไฟล์ CSV จะเริ่มต้นด้วยบรรทัดหัวตารางที่บอกชื่อคอลัมน์ ชื่อคอลัมน์แต่ละชื่อถูกแบ่งด้วยจุลภาค ตามด้วยข้อมูลในแต่ละแถว โดยข้อมูลแต่ละชิ้นในแถวเดียวกันจะถูกแบ่งออกจากกันด้วยจุลภาค แต่ละแถวของข้อมูลจะถูกแยกออกจากกันด้วยการขึ้นบรรทัดใหม่ หากตัวข้อมูลเองมีเครื่องหมายจุลภาคหรือการขึ้นบรรทัดใหม่ ข้อมูลนี้ต้องถูกครอบด้วยเครื่องหมายคำพูดเพื่อไม่ให้เกิดความสับสนในโครงสร้างของไฟล์ ตัวอย่างเช่น

```
Airline Name,Rating,Seat Type,Verified,Date Flown
Bangkok Airways,6,Economy,TRUE,March 2019
Bangkok Airways,9,Economy,TRUE,June 2019
```

```
Bangkok Airways,1,Economy,FALSE,November 2021
Bangkok Airways,9,Economy,FALSE,April 2020
Garuda Indonesia,1,Economy,TRUE,July 2023
Garuda Indonesia,9,Business,TRUE,January 2020
Garuda Indonesia,9,Business,TRUE,February 2020
Philippine Airlines,8,Economy,TRUE,April 2023
Philippine Airlines,7,Premium Economy,FALSE,January 2023
Philippine Airlines,1,Economy,TRUE,July 2023
```

ชุดข้อมูลในไฟล์ CSV ข้างต้นประกอบไปด้วยคอลัมน์ 5 คอลัมน์ ชื่อว่า

- Airline Name
- Rating
- Seat Type
- Verified
- Date Flown

ซึ่งถูกเก็บอยู่ในหัวตารางในบรรทัดแรก และประกอบไปด้วยแถว 10 แถวซึ่งเก็บข้อมูลของการรีวิวให้คะแนนจากผู้โดยสารของสายการบินต่าง ๆ แต่ละคน แต่ละเที่ยวบิน

รูปแบบไฟล์ CSV คือมีโครงสร้างที่ง่าย ยืดหยุ่น และสามารถใช้กับโปรแกรมได้หลากหลายตัว จึงเป็นที่นิยมมาก ตัวอย่างเช่น เราสามารถใช้ Visual Studio Code เปิดไฟล์ CSV เพื่อตรวจสอบข้อมูลเบื้องต้นได้ เพราะเป็นรูปแบบไฟล์ที่มนุษย์อ่านได้ (human-readable format) และจะเป็นรูปแบบไฟล์ที่เราจะใช้เป็นตัวอย่างในบทนี้ อย่างไรก็ตามหากตัวข้อมูลเองมีการขึ้นบรรทัดใหม่ เช่น เรียงความ บทความ หรือทวีต จะทำให้ไฟล์ซับซ้อนขึ้นมาก เพราะว่าต้องมีการแบ่งแยกว่าการขึ้นบรรทัดใหม่แสดงถึง แถวใหม่ หรือเป็นการขึ้นบรรทัดใหม่ในสตริงที่เก็บอยู่ในตัวข้อมูลเอง ทำให้ไฟล์ CSV ที่เรานับว่าเป็นไฟล์ที่มนุษย์อ่านได้กลายเป็นไฟล์ที่อ่านได้ยาก หากเราใช้ไลบรารี pandas หรือโปรแกรม Excel ในการจัดการข้อมูล CSV ตัวไลบรารีเองมักจะแก้ไขปัญหาเหล่านี้ให้

นอกจากนั้นยังมีไฟล์ตระกูล .tsv ซึ่งย่อมาจาก tab-separated value ไฟล์ TSV ที่จริงแล้วมีรูปแบบการเก็บข้อมูลใกล้เคียงกันกับไฟล์ CSV เพียงแค่เปลี่ยนตัวแบ่งระหว่างข้อมูลในแต่ละคอลัมน์เป็นแท็บ (`\t`) แทนที่จุลภาค ข้อดีของการใช้แท็บแทนจุลภาคคือ แท็บแสดงผลออกมาเป็นการเว้นระยะทำให้เห็นความแบ่งแยกระหว่างคอลัมน์ที่ชัดเจนและเป็นระเบียบ ได้ดีกว่าเมื่อเปิดด้วยโปรแกรมแก้ไขข้อความธรรมดา ทำให้เหมาะสำหรับการตรวจสอบข้อมูลด้วยตาเปล่าหรือการปรับแต่งข้อมูลเบื้องต้น ตัวอย่างเช่น

```
Airline Name    Rating  Seat Type       Verified        Date Flown
Bangkok Airways 6       Economy TRUE    March 2019
Bangkok Airways 9       Economy TRUE    June 2019
Bangkok Airways 1       Economy FALSE   November 2021
Bangkok Airways 9       Economy FALSE   April 2020
Garuda Indonesia        1       Economy TRUE    July 2023
Garuda Indonesia        9       Business        TRUE    January 2020
Garuda Indonesia        9       Business        TRUE    February 2020
Philippine Airlines     8       Economy TRUE    April 2023
Philippine Airlines     7       Premium Economy FALSE   January 2023
Philippine Airlines     1       Economy TRUE    July 2023
```

จากตัวอย่างข้างต้นจะเห็นว่าข้อมูลในแต่ละคอลัมน์ซึ่งถูกแบ่งด้วยแท็บ ซึ่งถูกแสดงผลออกมาเป็นเหมือนการเคาะเว้นระยะทำให้เห็นความแบ่งแยกระหว่างคอลัมน์ที่ชัดเจนและเป็นระเบียบขึ้น แต่ว่าแต่ละคอลัมน์อาจจะไม่แสดงผลตรงกันหมดทุกบรรทัด ขึ้นอยู่กับความสั้นยาวของตัวข้อมูลในแต่ละคอลัมน์เอง

ทั้งไฟล์ CSV และ TSV ต่างเป็นไฟล์ที่ได้รับความนิยมในวงการวิทยาการข้อมูล เนื่องจากมีโครงสร้างที่ง่าย มนุษย์สามารถอ่านได้ และสามารถใช้กับโปรแกรมหลายตัวได้ ทำให้เหมาะสำหรับการใช้งานในสถานการณ์ที่ต่างกัน รวมถึงรองรับการบันทึกชุดข้อมูลขนาดใหญ่อีกด้วย

ไฟล์ Excel

ไฟล์ Excel เป็นรูปแบบไฟล์ที่ใช้กับโปรแกรม Microsoft Excel ซึ่งเป็นส่วนหนึ่งของชุดโปรแกรม Microsoft Office โดยไฟล์ Excel มักมีนามสกุลไฟล์เป็น .xls หรือ .xlsx สำหรับ Microsoft Office เวอร์ชันใหม่กว่า ซึ่งใช้รูปแบบการเก็บข้อมูลที่ซับซ้อนและสามารถรองรับคุณสมบัติต่าง ๆ ได้หลากหลายมากกว่า CSV ได้แก่

1. *เวิร์กชีต (worksheet)* ข้อมูลใน Excel จะถูกจัดเก็บในเวิร์กชีตซึ่งแต่ละเวิร์กชีตสามารถมีข้อมูลแบบตารางได้ โดยผู้ใช้สามารถมีหลายเวิร์กชีตภายในไฟล์เดียว
2. *เซลล์ (cell)* ข้อมูลในเวิร์กชีตจะถูกจัดเก็บในเซลล์ โดยแต่ละเซลล์สามารถรองรับข้อมูลได้ทั้งข้อความ, ตัวเลข, สูตรการคำนวณ, หรือแม้กระทั่งกราฟิก
3. *สูตร (formula)* Excel รองรับสูตรการคำนวณที่ช่วยให้สามารถประมวลผลข้อมูลจากเซลล์ต่าง ๆ และอัปเดตผลลัพธ์อัตโนมัติเมื่อข้อมูลเปลี่ยนแปลง
4. *การจัดรูปแบบ* สามารถกำหนดรูปแบบให้กับข้อมูลได้อย่างละเอียด เช่น สี, ขนาด, และประเภทของตัวอักษร, การจัดตำแหน่ง, การใส่กรอบเซลล์, และอื่น ๆ

การจัดการและวิเคราะห์ข้อมูลด้วยโปรแกรม Microsoft Excel และไฟล์ตระกูล .xlsx เป็นที่นิยมเป็นอย่างมาก เนื่องจากมีฟีเจอร์หลากหลาย รองรับงานคำนวณตั้งแต่ขั้นเบื้องต้นจนไปถึงขั้นสูง ใช้งานค่อนข้างสะดวกสบายมีการแสดงผลที่สวยงาม สามารถจัดรูปแบบให้เกิดความโดดเด่น และความชัดเจนในการวิเคราะห์ข้อมูลได้ ทักษะการใช้โปรแกรมประเภทนี้เป็นทักษะจำเป็นอย่างยิ่งสำหรับนักวิเคราะห์ข้อมูล และนักวิทยาการข้อมูลทุกคน แต่ว่าข้อจำกัดที่สำคัญที่สุดของการใช้ Excel คือการรับมือกับข้อมูลขนาดใหญ่ที่มีจำนวนคอลัมน์มากจนไม่สามารถแสดงผลออกมาบนหน้าจอได้หมด หรือมีจำนวนแถวมากจนตัวโปรแกรมและเครื่องคอมพิวเตอร์ที่เราใช้ไม่สามารถเปิดขึ้นมาได้ ดังนั้นไฟล์ CSV มักจะเป็นรูปแบบการจัดเก็บไฟล์ที่เราเลือกใช้หากข้อมูลมีขนาดใหญ่

ดาตาเฟรม (dataframe) และการโหลดข้อมูลจากไฟล์

ดาตาเฟรม (dataframe) เป็นโครงสร้างข้อมูลหลักของ pandas ที่ใช้ในการจัดเก็บข้อมูลแบบตาราง เป็นโครงสร้างข้อมูลสองมิติที่คล้ายคลึงกับตารางข้อมูลที่ใช้ในโปรแกรม Excel หรือโปรแกรมตารางคำนวณอื่น ๆ ดาตาเฟรม ใช้เพื่อจัดเก็บข้อมูลในรูปแบบที่จัดระเบียบเป็นแถวและคอลัมน์ ซึ่งแต่ละแถวและคอลัมน์สามารถมีป้ายกำกับได้ เพราะฉะนั้นขั้นตอนแรกของการเตรียมข้อมูลสำหรับการวิเคราะห์ คือ การโหลดเอาข้อมูลมาใส่ในดาตาเฟรม

การสร้างดาตาเฟรมจากลิสต์ของดิกชันนารี

การสร้างดาตาเฟรมจากลิสต์ของดิกชันนารีเป็นวิธีที่ถูกใช้ในหลายสถานการณ์ เมื่อทำงานกับไลบรารี pandas เช่น

- การแปลงข้อมูลจาก API หรือ JSON: ข้อมูลที่ได้จากการเรียกใช้ API หรือจากไฟล์ JSON มักจะอยู่ในรูปแบบของดิกชันนารีหรือลิสต์ของดิกชันนารี ซึ่งมักจะมีโครงสร้างซ้อนในที่ซับซ้อนทำให้เขียนโปรแกรมวิเคราะห์ได้ไม่ค่อยสะดวก ดังนั้นเรามักแปลงถ่ายข้อมูลเหล่านี้เข้าสู่ดาตาเฟรม ช่วยให้สามารถวิเคราะห์และประมวลผลข้อมูลได้ง่ายขึ้น
- การรวมข้อมูลจากหลายแหล่ง: ถ้ามีการรวบรวมข้อมูลจากหลาย ๆ แหล่งที่จ่ายข้อมูลมาในรูปแบบข้อมูลที่ต่างกัน เรามักจะเขียนโค้ดเพิ่มเติมเพื่อรวบรวมเข้ามาใส่ดิกชันนารีซึ่งเป็นโครงสร้างข้อมูลที่ใช้ง่าย การสร้างดาตาเฟรม จากลิสต์ของดิกชันนารีทำให้การรวมข้อมูลเหล่านี้เป็นกลุ่มก้อนเดียวกัน เพื่อนำมาแสดงผล และวิเคราะห์ได้อย่างสะดวกขึ้น

ตัวอย่าง สมมติว่าเรานำชื่อนักเรียนมาจากหน้าเว็บไซต์ และต้องการจัดเก็บข้อมูลนักเรียนเหล่านี้ในรูปแบบของดาตาเฟรม โดยมีคอลัมน์เป็นชื่อนักเรียนและอายุ ข้อมูลนักเรียนเหล่านี้จะถูกเก็บในลิสต์ของดิกชันนารี ดังนี้

```
students = [{'student name': 'pang', 'age': 20},
     {'student name': 'dream', 'age': 19},
     {'student name':'tangmay', 'age': 19}
    ]
```

เราสามารถสร้างดาตาเฟรมจากลิสต์ของดิกชันนารีดังกล่าวได้โดยใช้ฟังก์ชัน `pd.DataFrame()` ดังนี้

```
import pandas as pd
students = [{'student name': 'pang', 'age': 20},
     {'student name': 'dream', 'age': 19},
     {'student name':'tangmay', 'age': 19}
    ]
df = pd.DataFrame(students)
```

เมื่อรันโค้ดด้านบนเสร็จเรียบร้อย ค่าของตัวแปร `df` จะเป็นดาตาเฟรมที่มีคอลัมน์ `student name` และ `age` และมีแถวที่เก็บข้อมูลของนักเรียนทั้งหมด 3 คน ดังนี้

```
  student name  age
0         pang   20
1        dream   19
2      tangmay   19
```

ถ้าหากเรารันโค้ดข้างต้นบน Jupyter notebook หรือ Visual Studio Code หรือ Google Colab เครื่องจะแสดงผลตารางออกมาอย่างสวยงาม รวมถึงสามารถดูตารางแบบ interactive ใกล้เคียงกับการใช้งาน Excel ได้ หรือใช้ลูกเล่นในการสร้างกราฟอัตโนมัติต่อได้ แต่ถ้าหากรันโค้ดด้านบนบนโปรแกรมไพทอนทั่วไป จะแสดงผลออกมาเป็นข้อความเท่านั้น

การสร้างดาตาเฟรมจากไฟล์ CSV

กรณีส่วนใหญ่ของการวิเคราะห์ข้อมูล เราจะต้องโหลดข้อมูลจากไฟล์ CSV ซึ่งเป็นรูปแบบไฟล์ที่ใช้งานกันอย่างแพร่หลาย เพราะใช้งานได้ง่าย ใช้งานได้ด้วยหลากหลายโปรแกรม pandas มีฟังก์ชัน `pd.read_csv()` ซึ่งจะโหลดข้อมูลจากไฟล์ CSV และสร้างดาตาเฟรมจากข้อมูลที่ได้ ตัวอย่างเช่น ถ้าเรามีไฟล์ CSV ที่เก็บข้อมูลการรีวิวของผู้โดยสารของสายการบินต่าง ๆ ชื่อว่า `airline-reviews-small.csv` เราสามารถอ่านไฟล์ดังกล่าวและโหลดเข้ามาใส่ในดาตาเฟรม ดังนี้

```
import pandas as pd
df = pd.read_csv('airline-reviews-small.csv')
```

ถ้าหากรันได้อย่างราบรื่น ไพทอนจะไม่แสดงข้อผิดพลาด และตัวแปร `df` จะเป็นดาตาเฟรมที่เก็บข้อมูลจากไฟล์ CSV ที่เราโหลดเข้ามา หากตัวไฟล์เองไม่ได้มีรูปแบบผิดเพี้ยน (เช่น จำนวนจุลภาคไม่เท่ากันทุกแถว) แล้ว pandas จะอ่านไฟล์ CSV และสร้างดาตาเฟรมจากข้อมูลขึ้นมา นักวิเคราะห์ข้อมูลมักจะติดนิสัยการใช้ชื่อตัวแปร `df` ในการเก็บดาตาเฟรม เพราะว่าเป็นชื่อที่สั้น และเกือบจะนับได้ว่าเป็นธรรมเนียมปฏิบัติของนักวิเคราะห์ข้อมูล แต่ผู้เขียนมีความเห็นว่าเราควรตั้งชื่อตัวแปรให้แสดงถึงชุดข้อมูลที่เราเก็บอยู่ เช่น `airline_reviews` หรือ `airline_reviews_small` จะช่วยให้เราเข้าใจว่าตัวแปรเก็บข้อมูลเป็นชุดข้อมูลของรีวิวของสายการบิน และเป็นดาตาเฟรม แน่นอนว่าชื่อตัวแปรเหล่านี้ยาวกว่าการเรียกว่า `df` เฉย ๆ แต่ว่าชื่อที่เหมาะสมจะช่วยให้เราเข้าใจโค้ดของเราได้ง่ายขึ้น รวมถึงในสมัยนี้เรามักใช้ IDE ที่ช่วยในการเติมชื่อตัวแปรให้ ความยาวของตัวแปรเองจึงไม่ควรจะเป็นประเด็นในการเลือกชื่อมากเหมือนก่อน เราจึงจะเรียกดาตาเฟรมที่เก็บข้อมูลรีวิวของสายการบินว่า `airline_reviews_small` ในตัวอย่างนี้

```
import pandas as pd
airline_reviews_small = pd.read_csv('airline-reviews-small.csv')
```

ปัญหาที่พบบ่อยในการอ่านไฟล์ CSV คือ ไฟล์มีการจัดรูปแบบไม่เหมาะสม เช่น อาจจะไม่ได้ใช้จุลภาคในการแบ่งคอลัมน์จริง อาจจะใช้ `\t` เป็นอักขระคั่นแทนแต่ว่าใช้สกุลไฟล์เป็น .csv หรือว่าแต่ละแถวมีจำนวนอักขระคั่นไม่เท่ากันทุกแถว ในกรณีเหล่านี้เราต้องเปิดไฟล์และตรวจสอบว่าข้อมูลถูกแบ่งอย่างถูกต้องหรือไม่ และแก้ไขข้อมูลด้วยมือให้ถูกต้องก่อนที่จะโหลดข้อมูลเข้ามาในดาตาเฟรม

การตรวจสอบส่วนประกอบหลักของดาตาเฟรม

ส่วนประกอบหลักของดาตาเฟรมล้อไปกับส่วนประกอบหลักของข้อมูลแบบตาราง ได้แก่

1. แถว
2. คอลัมน์
3. หัวตาราง
4. ดัชนี (index)
5. ตัวข้อมูล

ในการเตรียมข้อมูลเบื้องต้น ควรมีการตรวจสอบข้อมูลดังนี้

สำรวจข้อมูล

ขั้นตอนแรกเราต้องทำความรู้จักกับข้อมูลในมิติต่าง ๆ เพื่อทำให้เราเห็นภาพคร่าว ๆ ว่าตารางข้อมูลของเรามีลักษณะเป็นอย่างไร ในกรณีส่วนใหญ่เรามักจะมีข้อมูลอยู่จำนวนหลายแถวด้วยกัน เราเลยมักจะดูข้อมูลด้วยตาเปล่าเพียงส่วนหนึ่งเท่านั้น ได้แก่ ดูข้อมูล 15 แถวแรก โดยการใช้คำสั่ง `.head` ดูข้อมูล 15 แถวสุดท้าย `.tail` สุ่มดูข้อมูล 15 แถว `.sample`

คำสั่ง `.head` จะแสดงดาตาเฟรมเฉพาะแถวบนสุดเท่านั้น และเรากำหนดเองได้ว่าอยากให้แสดงผลทั้งหมดกี่แถว คำสั่งนี้จะคืนค่าออกมาเป็นดาตาเฟรมใหม่ที่มีจำนวนแถวตามที่กำหนด ตัวอย่างเช่น

```
airline_reviews_small.head(2)
```

จะแสดงผลลัพธ์ออกมาเป็นดาตาเฟรมที่มีแถว 2 แถวแรกเท่านั้น

คำสั่ง `.tail` จะแสดงดาตาเฟรมเฉพาะแถวล่างสุดเท่านั้น และเรากำหนดเองได้ว่าอยากให้แสดงผลทั้งหมดกี่แถว คล้ายคลึงกับคำสั่ง `.head` คำสั่งนี้จะคืนค่าออกมาเป็นดาตาเฟรมใหม่ที่มีจำนวนแถวตามที่กำหนด ตัวอย่างเช่น

```
airline_reviews_small.tail(2)
```

จะแสดงผลลัพธ์ออกมาเป็นดาตาเฟรมที่มีแถว 2 แถวล่างเท่านั้น

คำสั่ง `.sample` จะสุ่มแถวของดาตาเฟรมออกมาใส่ในอีกดาตาเฟรมหนึ่ง โดยเราสามารถกำหนดจำนวนแถวที่ต้องการให้แสดงผลได้เช่นกัน เรามักจะใช้คำสั่งนี้หลาย ๆ ครั้ง เพื่อให้พอเห็นภาพรวมของข้อมูลมากขึ้น ตัวอย่างเช่น

```
airline_reviews_small.sample(2)
```

จะแสดงผลลัพธ์ออกมาเป็นดาตาเฟรมที่มีแถว 2 แถวที่ถูกสุ่มออกมาจากดาตาเฟรม `airline_reviews_small` โดยแต่ละครั้งจะได้ผลออกมาไม่เหมือนกัน ยกเว้นเสียแต่เราจะตั้ง seed ของการสุ่มไว้เท่ากัน

ตรวจสอบจำนวนแถวและคอลัมน์

เมื่อได้ข้อมูลมาทุกครั้งเราควรตรวจสอบว่าปริมาณข้อมูลตรงกับที่ควรจะเป็นหรือไม่ และมีปริมาณมากพอสำหรับการวิเคราะห์ที่ต้องการทำต่อไปหรือไม่ และจำนวนคอลัมน์ตรงกับที่พจนานุกรมข้อมูล หรือตรงกับจำนวนคอลัมน์ที่ควรจะเป็นหรือไม่ เราสามารถตรวจสอบจำนวนแถวและจำนวนคอลัมน์ของดาตาเฟรมได้โดยใช้ฟังก์ชัน `shape` ซึ่งจะแสดงจำนวนแถวและจำนวนคอลัมน์ของดาตาเฟรม ตัวอย่างเช่น

```
airline_reviews_small.shape
```

จะแสดงผลลัพธ์ออกมาเป็นทูเปิลที่มีสมาชิกสองตัว คือ จำนวนแถวและจำนวนคอลัมน์ ตัวอย่างเช่น ถ้าดาตาเฟรม `airline_reviews_small` มี 10 แถว และ 5 คอลัมน์ ผลลัพธ์ที่ได้จะเป็น `(10, 5)`

หรือหากเราต้องการดูเฉพาะจำนวนแถวเพียงอย่างเดียวสามารถใช้คำสั่ง `len` ได้เหมือนกับการหาจำนวนข้อมูลในลิสต์ เช่น

```
len(airline_reviews_small)
```

จะแสดงผลลัพธ์ออกมาเป็นจำนวนแถวของดาตาเฟรม

ตรวจสอบและแก้ไขหัวตาราง

เราควรเปลี่ยนชื่อคอลัมน์ที่อยู่ในหัวตารางให้สื่อความหมาย และเข้าใจง่าย หัวตารางของดาตาเฟรมจะถูกเก็บในรูปของลิสต์ของสตริง ซึ่งแสดงชื่อของคอลัมน์ หากเราต้องการดูหัวตารางของดาตาเฟรม เราสามารถใช้ฟังก์ชัน `columns` ได้ ตัวอย่างเช่น

```
airline_reviews_small.columns
```

จะแสดงผลลัพธ์ออกมาเป็นอ็อบเจกต์ชื่อว่า `Index` แต่สามารถนำมาใช้ในลักษณะเดียวกันกับลิสต์ได้ ดังนี้

```
Index(['Airline Name', 'Rating', 'Seat Type', 'Verified', 'Date Flown'], dtype='object')
```

สังเกตได้ว่าคำสั่งไม่มี `()` ต่อท้าย ซึ่งแสดงว่าเป็นการเรียกลักษณะประจำ (attribute) ของดาตาเฟรม และไม่ใช่การเรียกฟังก์ชัน เพราะฉะนั้นเราสามารถเปลี่ยนชื่อคอลัมน์ทั้งหมด โดยให้ค่าใหม่กับลักษณะประจำได้ วิธีที่สะดวกที่สุดคือคัดลอกลิสต์ของชื่อคอลัมน์เดิมมา และแก้ไขชื่อคอลัมน์ที่ต้องการเปลี่ยน และให้ค่าใหม่กับลักษณะประจำ ตัวอย่างเช่น

```
airline_reviews_small.columns =
    ['Airline Name', 'Rating', 'Seat Type', 'Verified', 'Date Flown', 'Review']
```

จากนั้นให้แก้สตริงที่อยู่ในลิสต์ตามที่เห็นควร เช่น

```
airline_reviews_small.columns =
    ['Airline', 'Overall rating', 'Seat type', 'Verified', 'Date flown', 'Review text']
```

การแก้ไขหัวตารางหรือชื่อคอลัมน์ด้วยวิธีนี้มีข้อดี คือ ทำให้เราบันทึกชื่อคอลัมน์ทั้งหมดของตารางนี้ไว้ในโค้ดของเราโดยปริยาย ทำให้กลับมาดูโค้ดเราในภายหลังแล้วเข้าใจได้ง่าย และทราบทันทีว่าคอลัมน์มีชื่อว่าอะไรบ้าง

แต่ถ้าหากตารางมีคอลัมน์เยอะมาก ๆ เราไม่อยากได้ลิสต์ที่ยาวมาก ๆ มาอยู่ในโค้ดของเรา เราสามารถใช้คำสั่ง `df.rename` ในการเปลี่ยนชื่อคอลัมน์ได้ เช่น

```
airline_reviews_small.rename(columns={'Airline Name': 'Airline', 'Rating': 'Overall rating', 'Seat Typ
```

สังเกตว่าคำสั่งมีการใช้พารามิเตอร์ `inplace=True` ซึ่งหมายความว่าเราจะแก้ไขคอลัมน์ในตัวดาตาเฟรมเดิม และไม่ได้สร้างดาตาเฟรมใหม่ขึ้นมา ถ้าเราไม่ใส่พารามิเตอร์นี้ คำสั่งจะสร้างดาตาเฟรมใหม่ที่มีคอลัมน์ที่แก้ไขแล้ว และเราต้องเก็บค่าใหม่ไว้ในตัวแปรใหม่ เช่น

```
airline_reviews_small = airline_reviews_small.rename(columns={'Airline Name': 'Airline', 'Rating': 'Ov
```

หากเราต้องการรันคำสั่งที่มีการเปลี่ยนแปลงค่าของตัวดาตาเฟรม เราควรรันคำสั่งโดยไม่ตั้ง `inplace=True` ก่อน และให้ Jupyter notebook แสดงผลออกมาและตรวจสอบว่าผลตรงกับที่คาดหวังหรือไม่ จากนั้นค่อยรันคำสั่งเดิม แต่ตั้ง `inplace=True` และรันอีกครั้ง

ตรวจสอบและแปลงแบบชนิดของข้อมูลให้ถูกต้อง

ตัวข้อมูลที่อยู่ในคอลัมน์เดียวกันในดาตาเฟรมของ pandas จะต้องมีชนิดของข้อมูลเดียวกัน นั่นเป็นเพราะว่าการประมวลผลข้อมูลใน pandas จะมีประสิทธิภาพมากขึ้นเมื่อข้อมูลในคอลัมน์นั้นเป็นประเภทเดียวกัน เช่น ตัวเลขทั้งหมดหรือข้อความทั้งหมด หากมีการผสมกันของชนิดข้อมูลในคอลัมน์เดียวกัน อาจทำให้เกิดข้อผิดพลาดในการคำนวณและวิเคราะห์ข้อมูล

ยกตัวอย่างเช่น หากเรามีคอลัมน์ที่ควรจะเก็บตัวเลขเพื่อนำมาหาค่าสถิติ แต่กลับมีตัวอักษรปนอยู่ เมื่อเราพยายามหาค่าเฉลี่ย (mean) ของคอลัมน์นั้น การคำนวณจะไม่สามารถทำได้ เนื่องจากการหาค่าเฉลี่ยต้องการข้อมูลที่เป็นตัวเลขทั้งหมด การมีตัวข้อมูลที่เป็นตัวอักษรในคอลัมน์จะทำให้เกิดข้อผิดพลาดในการคำนวณ ดังนั้น การรักษาความเป็นหนึ่งเดียวของชนิดข้อมูลในคอลัมน์จึงเป็นสิ่งสำคัญ เพื่อให้เราสามารถใช้ประโยชน์จากฟังก์ชันต่าง ๆ ของ pandas ได้อย่างเต็มที่

เราสามารถตรวจสอบชนิดของข้อมูลในคอลัมน์ของดาตาเฟรมได้โดยใช้ฟังก์ชัน `dtypes` ซึ่งจะแสดงชนิดของข้อมูลของแต่ละคอลัมน์ ตัวอย่างเช่น

```
airline_reviews_small.dtypes
```

จะแสดงผลลัพธ์ออกมาเป็นชนิดของข้อมูลของแต่ละคอลัมน์ ตัวอย่างเช่น

```
Airline Name    object
Rating         float64
Seat Type       object
Verified          bool
Date Flown      object
dtype: object
```

ผลลัพธ์ที่ได้จะออกมาเป็นโครงสร้างข้อมูลที่เรียกว่า `Series` ซึ่งเป็นโครงสร้างข้อมูลที่คล้ายคลึงกับลิสต์ แต่ว่าสมาชิกแต่ละตัวจะมีค่าดัชนี (index) ซึ่งเปรียบได้กับชื่อที่เราให้กับสมาชิกแต่ละตัวใน `Series` ในผลลัพธ์ข้างต้น ชนิดของข้อมูลของแต่ละคอลัมน์ถูกแสดงออกมา โดยใช้ชื่อของคอลัมน์เป็นดัชนี และชนิดของข้อมูลเป็นค่าที่เก็บอยู่ใน `Series` นั้น ๆ

- คอลัมน์ `Airline Name` และ `Seat Type` มีแบบชนิดของข้อมูลเป็น `object` ซึ่งหมายความว่าเป็นข้อความหรือตัวแปรจำแนกประเภท (categorical variable)
- คอลัมน์ `Rating` มีแบบชนิดของข้อมูลเป็น `float64` ซึ่งหมายความว่าเป็นตัวเลขที่มีทศนิยม
- คอลัมน์ `Verified` มีแบบชนิดของข้อมูลเป็น `bool` ซึ่งหมายความว่าเป็นข้อมูลที่มีค่าเป็นจริงหรือเท็จ
- คอลัมน์ `Date Flown` มีแบบชนิดของข้อมูลเป็น `object` ซึ่งหมายความว่าเป็นข้อความ

ชนิดของข้อมูลที่ `Series` รองรับมีหลายแบบ แบบชนิดที่ใช้บ่อยที่สุดได้แก่

1. `int64` สำหรับข้อมูลตัวเลขจำนวนเต็ม
2. `float64` สำหรับข้อมูลตัวเลขที่มีทศนิยม
3. `bool` สำหรับข้อมูลที่มีค่าเป็นจริงหรือเท็จ
4. `datetime64` สำหรับข้อมูลวันที่และเวลา
5. `category` สำหรับข้อมูลที่เป็นตัวแปรจำแนกประเภท
6. `object` สำหรับข้อมูลที่เป็นข้อความหรือข้อมูลที่ไม่เข้ากับแบบชนิดของข้อมูลอื่น ๆ

แต่ในพจนานุกรมข้อมูลบอกว่าคอลัมน์ `Rating` ควรจะเป็นชนิดของข้อมูล `int64` เนื่องจากผู้ที่ให้ความเห็นให้คะแนนมาเป็นจำนวนเต็ม คอลัมน์ `Seat Type` ควรจะเป็นชนิดของข้อมูล `category` เนื่องจากมีค่าที่เป็นตัวแปรจำแนกประเภทที่มีค่าที่เป็นไปได้ 4 แบบ ไม่ใช่ข้อความที่เขียนได้อย่างอิสระ และคอลัมน์ `Date Flown` ควรจะเป็นชนิดของข้อมูล `datetime64` เนื่องจากเป็นวันที่ ไม่ใช่ข้อความอิสระ ดังนั้นเราจำเป็นต้องแปลงชนิดของข้อมูลให้ถูกต้องก่อนที่จะดำเนินการวิเคราะห์ข้อมูลต่อไป การแปลงชนิดของข้อมูลใน pandas สามารถทำได้โดยใช้ฟังก์ชัน `astype()` โดยใส่ชื่อแบบชนิดของข้อมูลเข้ามาเป็นพารามิเตอร์ ตัวอย่างเช่น

```
airline_reviews_small['Rating'] = airline_reviews_small['Rating'].astype('int64')
airline_reviews_small['Seat Type'] = airline_reviews_small['Seat Type'].astype('category')
airline_reviews_small['Date Flown'] = pd.to_datetime(airline_reviews_small['Date Flown'])
```

ในตัวอย่างข้างต้น เราแปลงชนิดของข้อมูลในคอลัมน์ `Rating` จาก `float64` เป็น `int64` คอลัมน์ `Seat Type` จาก `object` เป็น `category` และแปลงชนิดของข้อมูลในคอลัมน์ `Date Flown` จาก `object` เป็น `datetime64` โดยใช้ฟังก์ชัน `astype()` และ `pd.to_datetime()` ตามลำดับ

จากนั้นให้เราตรวจสอบแบบชนิดของทุกคอลัมน์อีกครั้ง ด้วยคำสั่ง `airline_reviews_small.dtypes()`

```
Airline Name            object
Rating                   int64
Seat Type             category
Verified                  bool
Date Flown      datetime64[ns]
Review                  object
dtype: object
```

การทำความสะอาดข้อมูล

การทำความสะอาดข้อมูล (data cleaning) เป็นขั้นตอนที่สำคัญในการเตรียมข้อมูลเพื่อนำมาวิเคราะห์ ประกอบไปด้วยการตรวจสอบความครบถ้วนว่ามีแถวหรือคอลัมน์ใดหรือไม่ที่ไม่มีตัวข้อมูลอยู่ และการตรวจสอบความถูกต้องมีแถวหรือคอลัมน์ใดหรือไม่ที่มีข้อมูลที่ไม่ถูกต้องอยู่

การตรวจหาข้อมูลที่หายไป

ข้อมูลที่หายไป (missing data) คือข้อมูลที่ไม่มีค่าในคอลัมน์ หรือข้อมูลที่มีค่าเป็น `NaN` ใน pandas ข้อมูลที่หายไปจะมีผลกระทบต่อการวิเคราะห์ข้อมูล และการสร้างโมเดลที่ถูกต้อง การตรวจหาข้อมูลที่หายไปสามารถทำได้โดยใช้ฟังก์ชัน `isnull()` หรือ `isna()` ซึ่งจะแสดงข้อมูลที่หายไปในรูปของ `Series` ที่มีค่าเป็น `True` หรือ `False` ตามที่ข้อมูลหายไปหรือไม่ จากนั้นเราจึงพิจารณาการบริบทของเก็บข้อมูลของชุดข้อมูลนี้ และตัดสินใจว่าควรจะเติมค่าอะไรเข้าไป หรือว่าควรจะลบแถวหรือคอลัมน์ที่มีข้อมูลที่หายไปออกไป

ในบริบทของการวิเคราะห์ข้อมูลด้วย pandas คำว่า NaN หมายถึง “Not a Number” ซึ่งเป็นค่าที่ใช้แทนข้อมูลที่หายไปหรือข้อมูลที่ไม่สามารถแสดงเป็นตัวเลขได้ ในการทำงานกับข้อมูลที่เป็นเชิงปริมาณ NaN จะถูกใช้บ่อยในการจัดการกับข้อมูลที่ขาดหายไป ตัวอย่างเช่น หากเรามีชุดข้อมูลของคะแนนสอบนักเรียน และบางคนไม่ได้เข้าสอบ คะแนนเหล่านี้อาจถูกแทนที่ด้วย NaN สืบเนื่องจากไฟล์ CSV หรือ Excel อาจจะบันทึกข้อมูลของนักเรียนที่ขาดสอบเป็นเซลล์ว่าง ๆ แทนที่จะใส่ค่าเป็น 0 ไปซึ่งอาจจะแปลว่า เข้าสอบแต่ว่าตอบข้อสอบผิดทุกข้อ

ฟังก์ชัน `isnull()` และ `isna()` ใช้สำหรับตรวจสอบข้อมูลที่หายไปในดาตาเฟรมโดยจะคืนค่าเป็นดาตาเฟรมที่มีค่าเป็น `True` หากข้อมูลในช่องนั้นเป็น NaN และเป็น `False` หากข้อมูลในช่องนั้นมีค่าข้อมูลอยู่ ตัวอย่างเช่น หากเรามีดาตาเฟรมที่มีข้อมูลคะแนนสอบและมีบางช่องที่ตัวข้อมูลหายไป เมื่อใช้ `isnull()` เราจะเห็นว่าในช่องที่ไม่มีค่าจะแสดงเป็น `True` ฟังก์ชัน `isna()` มีการทำงานที่เหมือนกับ `isnull()` ทุกประการ การใช้งานทั้งสองฟังก์ชันนี้ไม่ได้มีความแตกต่างกันในเชิงปฏิบัติ ทั้งคู่สามารถใช้แทนกันได้ ขึ้นอยู่กับความชอบส่วนบุคคลในการเขียนโค้ดหรือการอ่านโค้ด

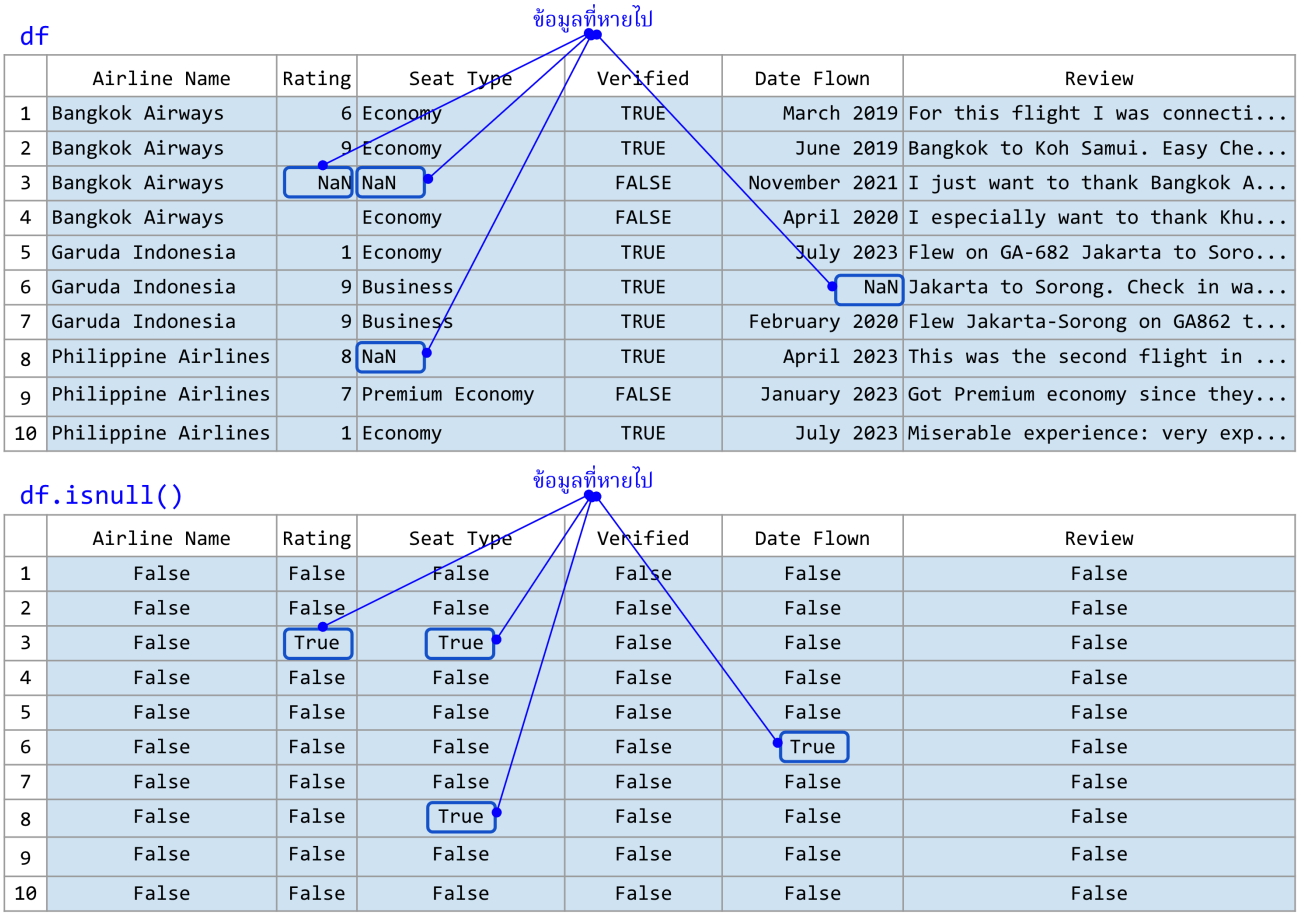
สมมติว่าเราได้ข้อมูลที่มีค่าที่หายไปอยู่ในไฟล์ CSV (ข้อมูลแต่ละบรรทัดลดเหลือเพียงบรรทัดละ 75 ตัวอักษรเพื่อที่จะแสดงผลบนหน้าจอและกระดาษได้พอดี) ดังนี้

```
Airline Name,Rating,Seat Type,Verified,Date Flown,Review
Bangkok Airways,6,Economy,TRUE,March 2019,"For this flight I was connec...
Bangkok Airways,9,Economy,TRUE,June 2019,"Bangkok to Koh Samui. Easy Ch...
Bangkok Airways,,,FALSE,November 2021,"I just want to thank Bangkok Air...
Bangkok Airways,9,Economy,FALSE,April 2020,"I especially want to thank ...
Garuda Indonesia,1,Economy,TRUE,July 2023,"Flew on GA-682 Jakarta to So...
Garuda Indonesia,9,Business,TRUE,,"Jakarta to Sorong. Check in was stra...
Garuda Indonesia,9,Business,TRUE,February 2020,"Flew Jakarta-Sorong on ...
Philippine Airlines,8,,TRUE,April 2023,"This was the second flight in 7...
Philippine Airlines,7,Premium Economy,FALSE,January 2023,Got Premium ec...
Philippine Airlines,11,Economy,TRUE,July 2023,"Miserable experience: ver...
```

จากข้อมูลข้างต้น เราสังเกตหาข้อมูลที่หายไปค่อนข้างยาก และในการใช้งานจริงข้อมูลมักจะมีขนาดใหญ่มากจนเราไม่ได้มองหาข้อมูลที่หายไปด้วยตาเปล่าได้ ในกรณีนี้เราสามารถใช้ฟังก์ชัน `isnull()` หรือ `isna()` ในการตรวจสอบข้อมูลที่หายไปได้

ตัวอย่างเช่น

```
bad_df = pd.read_csv('airlines-reviews-small-missing.csv')
bad_df.isnull()
```



	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	Bangkok Airways	6	Economy	TRUE	March 2019	For this flight I was connecti...
2	Bangkok Airways	9	Economy	TRUE	June 2019	Bangkok to Koh Samui. Easy Che...
3	Bangkok Airways	NaN	NaN	FALSE	November 2021	I just want to thank Bangkok A...
4	Bangkok Airways		Economy	FALSE	April 2020	I especially want to thank Khu...
5	Garuda Indonesia	1	Economy	TRUE	July 2023	Flew on GA-682 Jakarta to Soro...
6	Garuda Indonesia	9	Business	TRUE	NaN	Jakarta to Sorong. Check in wa...
7	Garuda Indonesia	9	Business	TRUE	February 2020	Flew Jakarta-Sorong on GA862 t...
8	Philippine Airlines	8	NaN	TRUE	April 2023	This was the second flight in ...
9	Philippine Airlines	7	Premium Economy	FALSE	January 2023	Got Premium economy since they...
10	Philippine Airlines	1	Economy	TRUE	July 2023	Miserable experience: very exp...

	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	True	True	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	True	False
7	False	False	False	False	False	False
8	False	False	True	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False

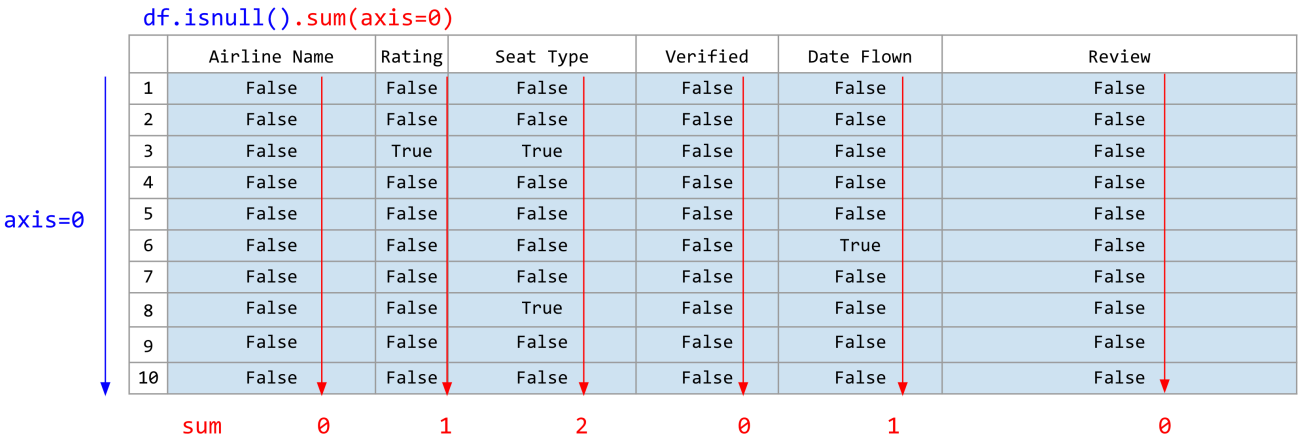
ภาพที่ 25 (บน) ดาตาเฟรมที่มีข้อมูลที่หายไป แทนค่าด้วย NaN

(ล่าง) ดาตาเฟรมที่ถูกคืนค่ามาจากคำสั่ง `.isnull()`

(ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

ผลที่ได้จะเป็นดาตาเฟรมที่เต็มไปด้วยค่า `True` และ `False` ซึ่งยังไม่ได้ช่วยให้เรามองหาช่องที่มีข้อมูลที่หายไปได้ แต่เราสามารถใช้ฟังก์ชัน `sum()` ในการนับจำนวนข้อมูลที่หายไปในแต่ละคอลัมน์ได้ เราอาจจะสงสัยว่าเราจะหาผลรวมของข้อมูลที่ไม่ใช่ตัวเลขอย่างบูลีน `True` หรือ `False` ได้อย่างไร ที่จริงแล้ว `True` ถูกแทนค่าด้วย 1 และ `False` ถูกแทนค่าด้วย 0 เมื่อนำมาบวกหรือลบกัน เพราะฉะนั้นการหาผลรวมของคอลัมน์ที่บูลีนอยู่เท่ากับการนับจำนวน `True` ที่อยู่ในคอลัมน์นั้น ตัวอย่างเช่น

```
airline_reviews_small.isnull().sum(axis=0)
```



	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	True	True	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	True	False
7	False	False	False	False	False	False
8	False	False	True	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False

ภาพที่ 26 หาผลรวมของบูลีนที่อยู่ในแต่ละคอลัมน์ (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

คำสั่งข้างต้นทำความเข้าใจได้ยากเล็กน้อย เพราะมีการเรียกสองเมท็อดต่อเนื่องกันในบรรทัดเดียว เราเรียกวิธีนี้ว่าการโยงเมท็อด (method chaining) ซึ่งการเรียกเมท็อดต่อเนื่องกันโดยไม่ต้องเก็บผลลัพธ์ของเมท็อดก่อนหน้าไว้ในตัวแปรก่อน และเรียกเมท็อดต่อไปได้ทันที ในตัวอย่างข้างต้นเราให้ดาตาเฟรมเรียกเมท็อด `isnull()` ก่อน และให้ผลลัพธ์เป็นดาตาเฟรม จากนั้นใช้ดาตาเฟรมนั้นเรียกเมท็อด `sum()` ต่อ โดยให้พารามิเตอร์ `axis=0` ซึ่งหมายถึงให้ฟังก์ชัน `sum()` นับจำนวนข้อมูลที่หายไปในแต่ละคอลัมน์ ดังนั้นจึงมีค่าเท่ากับการเรียกเมท็อดแยกกัน โดยใช้คำสั่งดังนี้

```
empty_boolean_df = airline_reviews_small.isnull()
empty_boolean_df.sum(axis=0)
```

การเรียกโค้ดโดยวิธีข้างต้นได้ผลลัพธ์ออกมาเหมือนกัน แต่ว่าโค้ดจะยาวกว่าเล็กน้อย และมีความชัดเจนมากขึ้น ซึ่งก็ขึ้นอยู่กับความชอบส่วนตัวของแต่ละคนว่าต้องการใช้การโยงเมท็อดหรือไม่ แต่ว่าการใช้วิธีการโยงเมท็อดนี้เป็นที่นิยมในการเขียนโค้ดของ pandas เพราะว่าโค้ดสั้นกว่าและเข้าใจง่ายสำหรับคนที่เขียนโค้ดค่อนข้างคล่องแล้ว

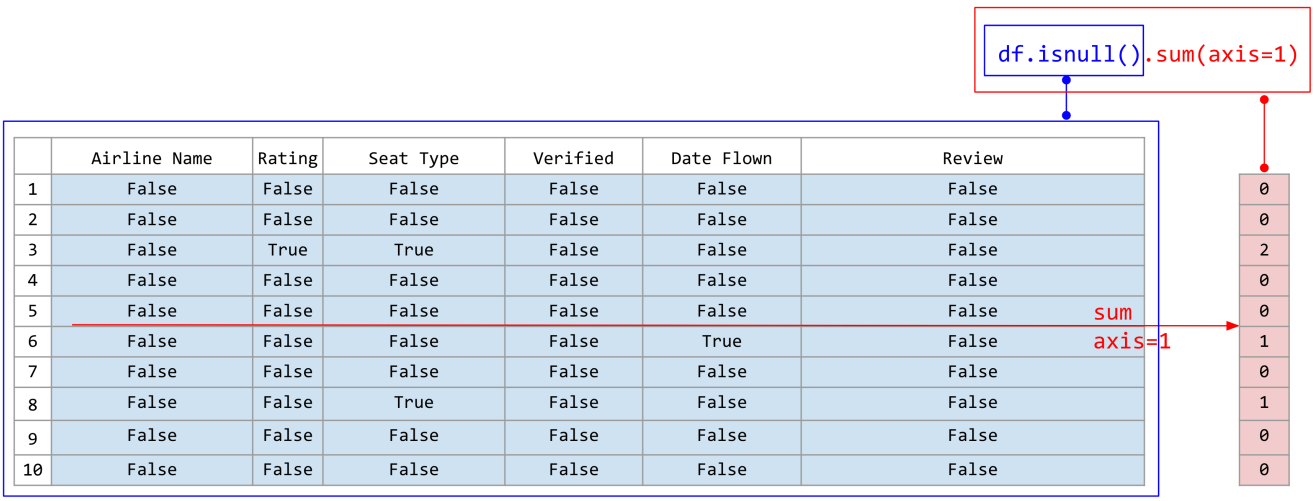
ผลลัพธ์ที่ได้จากทั้งสองวิธีจะเป็นจำนวนข้อมูลที่หายไปในแต่ละคอลัมน์ คืนค่ามาเป็น `Series` ที่ index เป็นชื่อคอลัมน์ของดาตาเฟรมเดิม ดังนี้

```
Airline Name    0
Rating          1
Seat Type       2
Verified        0
Date Flown      1
Review          0
dtype: int64
```

จากผลลัพธ์ที่ได้ เราสามารถเห็นได้ว่าคอลัมน์ `Rating` มีข้อมูลที่หายไป 1 ค่า คอลัมน์ `Seat Type` มีข้อมูลที่หายไป 2 ค่า และคอลัมน์ `Date Flown` มีข้อมูลที่หายไป 1 ค่า แต่คอลัมน์ `Airline Name` และ `Verified` ไม่มีข้อมูลที่หายไปเลย

ในลักษณะเดียวกัน เราสามารถคำนวณจำนวนแถวที่มีข้อมูลที่หายไปได้โดยใช้ฟังก์ชัน `sum()` โดยกำหนด `axis=1` ซึ่งหมายถึงให้ฟังก์ชัน `sum()` นับจำนวนข้อมูลที่หายไปในแต่ละแถว ตัวอย่างเช่น

```
airline_reviews_small.isnull().sum(axis=1)
```



	Airline Name	Rating	Seat Type	Verified	Date Flown	Review	
1	False	False	False	False	False	False	0
2	False	False	False	False	False	False	0
3	False	True	True	False	False	False	2
4	False	False	False	False	False	False	0
5	False	False	False	False	False	False	0
6	False	False	False	False	True	False	1
7	False	False	False	False	False	False	0
8	False	False	True	False	False	False	1
9	False	False	False	False	False	False	0
10	False	False	False	False	False	False	0

ภาพที่ 27 หาผลรวมของบูลีนที่อยู่ในแต่ละแถว (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

ผลลัพธ์ที่ได้จะเป็นจำนวนข้อมูลที่หายไปในแต่ละแถว คืนค่ามาเป็น `Series` ที่ดัชนี เป็นชื่อแถวของดาตาเฟรมเดิมดังนี้

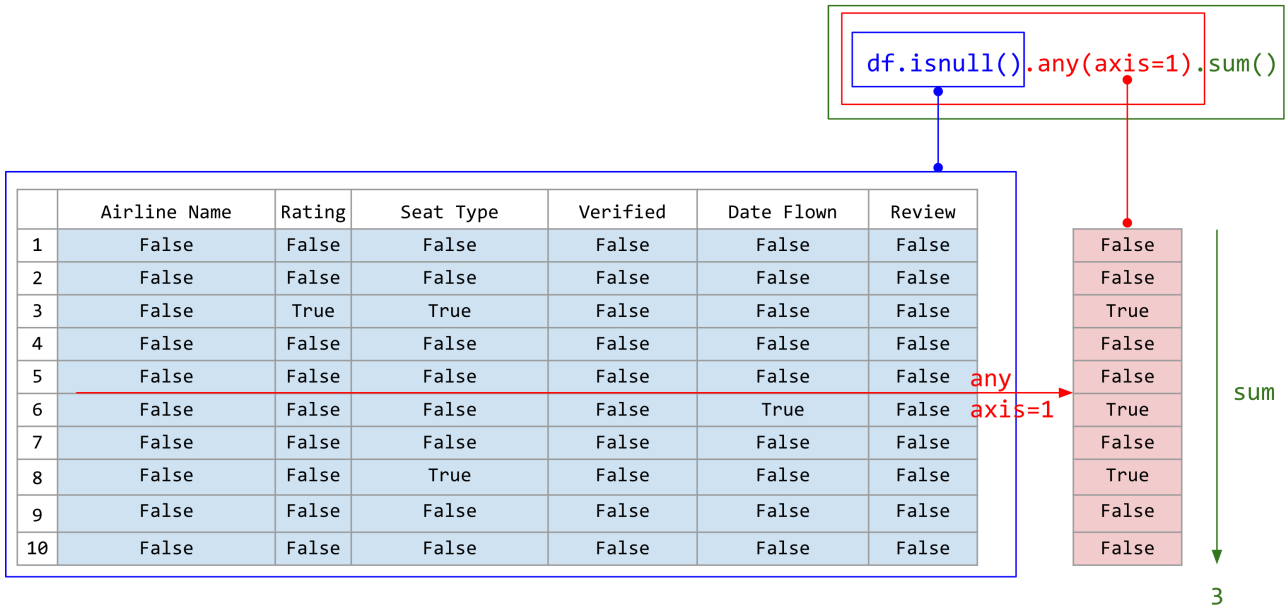
```
0    0
1    0
2    2
3    0
4    0
```

```
5    1
6    0
7    1
8    0
9    0
dtype: int64
```

จากผลลัพธ์ที่ได้ เราสามารถเห็นได้ว่าแถวที่ 2 มีข้อมูลที่หายไป 2 ค่า และแถวที่ 5 มีข้อมูลที่หายไป 1 ค่า แต่แถวอื่น ๆ ไม่มีข้อมูลที่หายไปเลย

แต่ถ้าหากดาตาเฟรมมีจำนวนแถวเยอะมาก เราเพียงต้องการทราบว่ามีกี่แถวที่มีข้อมูลที่หายไปอย่างน้อยหนึ่งช่อง ในกรณีนี้เราสามารถใช้ฟังก์ชัน `any()` หรือ `all()` เพื่อดูถามว่าแต่แถวนั้นมี `True` อย่างน้อยหนึ่งตัวหรือไม่ โดยกำหนด `axis=1` ในการตรวจสอบข้อมูลที่หายไปในแต่ละแถว ตัวอย่างเช่น

```
airline_reviews_small.isnull().any(axis=1).sum()
```



ภาพที่ 28 หาจำนวนแถวที่มีช่องที่ข้อมูลหายไปอย่างน้อยหนึ่งช่องโดยการโยงเมท็อดสามเมท็อด (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

`airline_reviews_small.isnull().any(axis=1)` จะคืนค่าเป็น `Series` ที่มีค่าเป็น `True` หากแถวนั้นมีข้อมูลที่หายไปอย่างน้อยหนึ่งช่อง และ `False` หากแถวนั้นไม่มีข้อมูลที่หายไปเลย ตัวอย่างเช่น

```
0    False
1    False
2     True
3    False
4    False
5     True
6    False
7     True
8    False
9    False
dtype: bool
```

เมื่อได้ผลลัพธ์ข้างต้นแล้วเราสามารถหาผลรวมของจำนวนแถวที่มีข้อมูลที่หายไปอย่างน้อยหนึ่งช่องได้ โดยใช้ฟังก์ชัน `sum()` ต่อท้าย ซึ่งจะคืนค่าเป็นจำนวนแถวที่มีข้อมูลที่หายไปอย่างน้อยหนึ่งช่อง

การกำจัดแถวที่มีข้อมูลที่หายไป หรือเติมค่าให้กับข้อมูลที่หายไป

เมื่อเราพบแล้วว่าแถวใดบ้างมีข้อมูลที่หายไป วิธีที่ง่ายที่สุดคือการลบแถวที่มีข้อมูลที่หายไปออกไป แต่ว่าเราต้องคำนวณก่อนว่าถ้าหากลบแถวเหล่านั้นออกไปแล้ว เราจะเหลือข้อมูลอยู่ร้อยละเท่าไร ถ้าหากลบแถวที่มีข้อมูลที่หายไปออกไปแล้วเราจะเหลือข้อมูลอยู่น้อยกว่า 75% ให้ตรวจสอบว่าข้อมูลที่เหลืออยู่เพียงพอที่จะทำการวิเคราะห์ข้อมูลต่อไปหรือไม่ และอาจจะต้องปรึกษากับผู้ที่รวบรวมและจัดเก็บข้อมูลว่าเพราะเหตุใดจึงข้อมูลที่หายไปในสัดส่วนที่มากดังที่ปรากฏ

ฟังก์ชัน `dropna()` ใช้สำหรับลบแถวที่มีข้อมูลที่หายไปออกไป ตัวอย่างเช่น

```
airline_reviews_small.dropna(inplace=True)
```

ในตัวอย่างข้างต้น เราใช้ฟังก์ชัน `dropna()` โดยกำหนดพารามิเตอร์ `inplace=True` ซึ่งหมายถึงการลบแถวที่มีข้อมูลที่หายไปออกไปจากดาตาเฟรม `airline_reviews_small` โดยที่ไม่ต้องเก็บผลลัพธ์ไว้ในตัวแปรใหม่ หลังจากการลบแถวที่มีข้อมูลที่หายไปออกไปแล้ว เราสามารถตรวจสอบดูว่าข้อมูลที่เหลืออยู่เพียงพอที่จะทำการวิเคราะห์ข้อมูลต่อไปหรือไม่ โดยใช้ฟังก์ชัน `shape` ซึ่งจะแสดงจำนวนแถวและคอลัมน์ที่มีในดาตาเฟรม ตัวอย่างเช่น

```
airline_reviews_small.shape
```

ผลลัพธ์ที่ได้จะเป็นจำนวนแถวและคอลัมน์ที่มีในดาตาเฟรม ตัวอย่างเช่น

```
(7, 6)
```

จากผลลัพธ์ที่ได้ เราสามารถเห็นได้ว่าหลังจากลบแถวที่มีข้อมูลที่หายไปออกไปแล้ว เราเหลือแถวอยู่ 7 แถว และมีคอลัมน์อยู่ 6 คอลัมน์ ซึ่งเราสามารถทำการวิเคราะห์ข้อมูลต่อไปได้

ถ้าสมมติว่าการวิเคราะห์ของเราไม่ได้สนใจว่าคนเขียนรีวิวบินสายการบินนั้นวันที่เท่าไรซึ่งเก็บอยู่ในคอลัมน์ *Date flown* เพราะฉะนั้นเราต้องการจะกำจัดแถวที่ไม่มีข้อมูลเกี่ยวกับคะแนน (คอลัมน์ *Rating*) และประเภทของที่นั่ง (คอลัมน์ *Seat type*) เท่านั้น ส่วนคอลัมน์ *Date flown* จะมีหรือไม่มีก็ได้ เราสามารถกำหนดคอลัมน์ที่สนใจไว้ในพารามิเตอร์ `subset` ของฟังก์ชัน `dropna()` ตัวอย่างเช่น

```
airline_reviews_small.dropna(subset=['Rating', 'Seat Type'], inplace=True)
```

ในตัวอย่างข้างต้น เราใช้ฟังก์ชัน `dropna()` โดยกำหนดพารามิเตอร์ `subset=['Rating', 'Seat Type']` ซึ่งหมายถึงการลบแถวที่มีข้อมูลที่หายไปออกไปจากดาตาเฟรม `airline_reviews_small` โดยที่ไม่ต้องเก็บผลลัพธ์ไว้ในตัวแปรใหม่ และเรากำหนดให้ลบแถวที่มีข้อมูลที่หายไปออกไปเฉพาะในคอลัมน์ `Rating` และ `Seat Type` เท่านั้น หลังจากการลบแถวที่มีข้อมูลที่หายไปออกไปแล้ว เราสามารถตรวจสอบดูว่าข้อมูลที่เหลืออยู่เพียงพอที่จะทำการวิเคราะห์ข้อมูลต่อไปหรือไม่ โดยใช้ฟังก์ชัน `shape` ซึ่งจะแสดงจำนวนแถวและคอลัมน์ที่มีในดาตาเฟรม ตัวอย่างเช่น

```
airline_reviews_small.shape
```

ผลลัพธ์ที่ได้จะเป็นจำนวนแถวและคอลัมน์ที่มีในดาตาเฟรม ตัวอย่างเช่น

```
(8, 6)
```

ในตัวอย่างข้างบนเราตั้งค่า `inplace=True` เพื่อให้การเปลี่ยนแปลงเกิดขึ้นกับดาตาเฟรม `airline_reviews_small` ทันที และไม่ต้องเก็บผลลัพธ์ไว้ในตัวแปรใหม่ ถ้าหากเราเขียนคำสั่งผิดและสั่ง `inplace=True` ทำให้เราย้อนกลับไปแก้ไม่ได้ ต้องกลับไปโหลดข้อมูลใหม่ตั้งแต่ต้น ทำให้การวิเคราะห์ติดขัด เพราะฉะนั้นผู้เขียนแนะนำว่าให้รันคำสั่งโดยไม่ตั้ง `inplace=True` ก่อน คำสั่งจะคืนค่าเป็นดาตาเฟรมใหม่ออกมาให้ จากนั้นเราตรวจสอบว่าผลลัพท์ที่ได้มานั้นถูกต้องหรือไม่ ถ้าหากถูกต้องแล้วให้ย้อนไปเปลี่ยนคำสั่งเป็น `inplace=True` แล้วรันคำสั่งอีกครั้งหนึ่งเพื่อเปลี่ยนแปลงดาตาเฟรมตามที่เราต้องการ

การเติมค่าให้กับข้อมูลที่หายไป

การแก้ไขข้อมูลที่หายไป สามารถทำได้โดยใช้ฟังก์ชัน `fillna()` ซึ่งจะแทนที่ข้อมูลที่หายไปด้วยค่าที่เรากำหนด สมมติว่าเราไปสืบทราบมาจากผู้ดูแลชุดข้อมูลนี้มาว่า ถ้าหากคอลัมน์ *Seat Type* ถูกเว้นว่างไว้ แปลว่าลูกค้าที่รีวิวนั่งตั๋วชั้นประหยัด เราสามารถเติมแถวที่มีค่าของ *Seat Type* เว้นว่างไว้ด้วยค่า `'Economy'` ดังนี้

```
airline_reviews_small['Seat Type'].fillna('Economy', inplace=True)
```

ในตัวอย่างข้างต้น เราสังเกตเห็นว่ามีคำสั่งในการเลือกคอลัมน์ `'Seat Type'` โดยการใช้ `[ชื่อคอลัมน์]` เพื่อเลือกคอลัมน์ออกมาก่อนที่จะใช้เมท็อด `.fillna()` ไปบนคอลัมน์นั้น ในคำสั่ง `fillna()` เราต้องกำหนดค่าที่เราต้องการเติมให้กับข้อมูลที่หายไปในคอลัมน์ `'Seat Type'` ซึ่งเป็นค่า `'Economy'` โดยที่ไม่ต้องเก็บผลลัพธ์ไว้ในตัวแปรใหม่ หลังจากการเติมค่าให้กับข้อมูลที่หายไปแล้ว

เราสามารถตรวจสอบดูว่าข้อมูลที่เหลืออยู่เพียงพอที่จะทำการวิเคราะห์ข้อมูลต่อไปหรือไม่ โดยใช้ฟังก์ชัน `isnull()` และ `sum()` ซึ่งจะแสดงจำนวนข้อมูลที่หายไปในแต่ละคอลัมน์ ตัวอย่างเช่น

```
airline_reviews_small.isnull().sum(axis=0)
```

ผลลัพธ์ที่ได้จะเป็นจำนวนข้อมูลที่หายไปในแต่ละคอลัมน์ ตัวอย่างเช่น

```
Airline Name    0
Rating          1
Seat Type       0
Verified        0
Date Flown      1
Review          0
dtype: int64
```

จากผลลัพธ์ที่ได้ เราสามารถเห็นได้ว่าคอลัมน์ *Seat Type* ไม่มีข้อมูลที่หายไปอีกแล้ว

การกำจัดคอลัมน์ที่ไม่ต้องการ

ในบางกรณีข้อมูลของเราอาจจะมีจำนวนคอลัมน์เยอะมาก ๆ และเราอาจจะไม่ได้ต้องการนำมาวิเคราะห์ทุกคอลัมน์ เราควรจะกำจัดคอลัมน์ที่ไม่ต้องการออกไป เพื่อให้ดาตาเฟรมมีขนาดเล็กลง ใช้พื้นที่ในหน่วยความจำของเครื่องน้อยลง และสะดวกต่อการวิเคราะห์ เพราะคอลัมน์ไม่ล้นทางขวาของจอ ทำให้เรามองเห็นข้อมูลที่สำคัญได้ชัดเจนขึ้น

เราสามารถใช้ฟังก์ชัน `drop()` ในการลบคอลัมน์ที่ไม่ต้องการออกไป ตัวอย่างเช่น

```
airline_reviews_small.drop(columns=['Date Flown', 'Verified'], inplace=True)
```

ในตัวอย่างข้างต้น เราใช้ฟังก์ชัน `drop()` โดยกำหนดพารามิเตอร์ `columns=['Date Flown', 'Verified']` ซึ่งหมายถึงการลบคอลัมน์ `Date Flown` และ `Verified` ออกจากดาตาเฟรม `airline_reviews_small` โดยที่ไม่ต้องเก็บผลลัพธ์ไว้ในตัวแปรใหม่ และเช่นเคยเราควรจะรันคำสั่งโดยไม่ตั้งค่า `inplace=True` และตรวจสอบด้วยสายตาก่อนว่าได้ผลตามที่คาดหวัง ก่อนที่จะรันคำสั่งอีกครั้งโดยการตั้ง `inplace=True` เพื่อเปลี่ยนแปลงดาตาเฟรมตามที่เราต้องการ

ในกรณีที่จำนวนคอลัมน์ที่ต้องการมีมากกว่าจำนวนคอลัมน์ที่ไม่ต้องการมาก ๆ เช่น เราอาจจะอยากได้เพียงคอลัมน์ *Review* และ *Rating* เท่านั้น ในกรณีนี้เราสามารถใช้ฟังก์ชัน `filter()` ในการเลือกคอลัมน์ที่ต้องการออกมา ตัวอย่างเช่น

```
airline_reviews_small.filter(['Review', 'Rating'], inplace=True)
```

ในตัวอย่างข้างต้น เราใช้ฟังก์ชัน `filter()` โดยกำหนดพารามิเตอร์ `['Review', 'Rating']` ซึ่งหมายถึงการเลือกคอลัมน์ `Review` และ `Rating` จากดาตาเฟรม `airline_reviews_small` คอลัมน์อื่น ๆ ที่เหลือจะถูกตัดออกหมด

นอกจากนั้นแล้ว ฟังก์ชัน `filter()` สามารถใช้เลือกคอลัมน์ที่ต้องการออกมาได้อีกหลายวิธี เช่น เลือกคอลัมน์ด้วยเรกเอกซ์ หรือเลือกตามเงื่อนไขอื่น ๆ ที่สามารถกำหนดได้ตามใจ ซึ่งผู้อ่านสามารถอ้างอิงเอกสารประกอบการใช้บนเว็บไซต์ของ pandas

การทำความสะอาดและแก้ไขข้อมูลที่เป็นข้อมูลตัวเลข

นอกเหนือจากการหาแถวที่มีข้อมูลที่หายไปแล้ว เรายังจะต้องตรวจสอบด้วยว่าข้อมูลที่มีอยู่แล้วนั้นถูกต้องหรือไม่ วิธีการตรวจสอบง่าย ๆ คือการเปิดไฟล์ขึ้นมาและกวาดสายตาดูคร่าว ๆ ว่ามีสิ่งแปลกปลอมหรือไม่ แต่ว่าเราสามารถใช้ฟังก์ชันต่าง ๆ ของ pandas ในการคำนวณสถิติเชิงพรรณนา (descriptive statistics) เพื่อตรวจสอบข้อมูลได้ง่ายขึ้น โดยเราจะตรวจสอบดังนี้

- ค่าเฉลี่ย อยู่ในช่วงที่ควรจะเป็นหรือไม่
- ค่าสูงสุด และค่าต่ำสุด อยู่ในช่วงที่ควรจะเป็นหรือไม่ เช่น คะแนนรีวิวในชุดข้อมูลของเราควรจะอยู่ในช่วง 1 ถึง 10 เพราะฉะนั้นค่าต่ำสุดที่พบไม่ควรต่ำกว่า 1 และค่าสูงสุดที่พบไม่ควรมากกว่า 10

เราสามารถใช้ฟังก์ชัน `describe()` ในการคำนวณสถิติเชิงพรรณนาของข้อมูลตัวเลขได้ ตัวอย่างเช่น

```
airline_reviews_small.describe()
```

ผลลัพธ์ที่ได้จะเป็นสถิติเชิงพรรณนาของข้อมูลตัวเลขที่มีในดาตาเฟรม ตัวอย่างเช่น

```
            Rating
count   9.000000
mean    7.666667
std         2.872281
min         1.000000
25%         7.000000
50%         9.000000
75%         9.000000
max         11.000000
```

จะเห็นว่ามีบางแถวที่มีค่า Rating มีค่าที่ผิดพลาด คือมีค่าที่สูงกว่า 10 ซึ่งอาจจะเกิดจากการบันทึกผลที่ผิดพลาด ในกรณีนี้เราควรจะปรึกษาผู้ที่เก็บข้อมูลว่า หากมีข้อผิดพลาดดังกล่าวเป็นเพราะสาเหตุใด ถ้าหากจากการบันทึกผิด บันทึกผิดจากอะไร เช่น จาก 1 กลายเป็น 11 เพราะเผลอกดปุ่ม 1 ไปสองครั้งหรือ จาก 10 กลายเป็น 11 สมมติเราต้องการให้ค่าที่เกิน 10 กลายเป็น 10 ให้ใช้คำสั่งดังนี้

```
airline_reviews_small.loc[airline_reviews_small['Rating'] > 10, 'Rating'] = 10
```

คำสั่งนี้ประกอบไปด้วยหลายส่วนด้วยกันที่อาจจะดูซับซ้อน ส่วนแรกคือคำสั่ง `.loc[]` ซึ่งเป็นคำสั่งในการอ้างอิงถึงกลุ่มของคอลัมน์หรือแถว โดยมีหลักการใช้ดังนี้

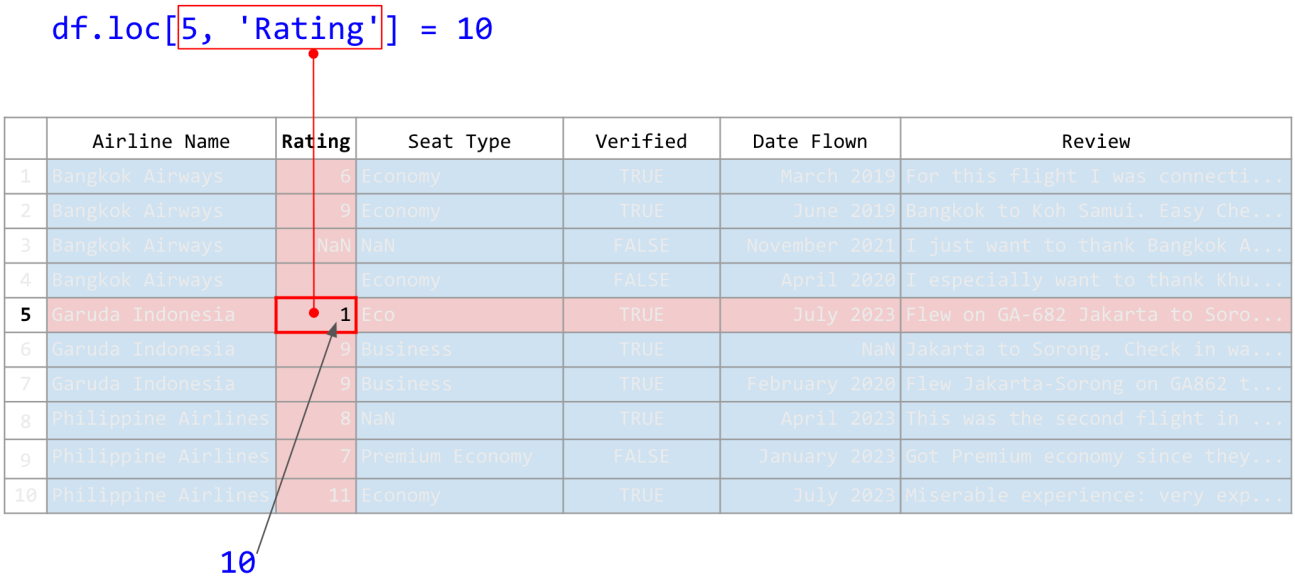
```
dataframe.loc[ตัวเลือกแถว, ตัวเลือกคอลัมน์]
dataframe.loc[ตัวเลือกแถว]
dataframe.loc[:, ตัวเลือกคอลัมน์]
```

การกำหนดตัวเลือกแถว หรือตัวเลือกคอลัมน์สามารถกำหนดได้ 4 วิธี ดังนี้

1. เลือกด้วยชื่อแถว (มักจะเป็นสตริง) หรือ ชื่อคอลัมน์ (มักจะเป็นตัวเลข)

เราสามารถใช้สตริงในการระบุชื่อคอลัมน์ หรือชื่อแถว (ดัชนี) โดยส่วนใหญ่แล้วชื่อแถวมักจะเป็นตัวเลขที่แสดงหมายเลขแถว เราจึงมักจะใช้ตัวเลขในการระบุแถว ตัวอย่างเช่น

```
airline_reviews_small.loc[5, 'Rating'] = 10
```



	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	Bangkok Airways	6	Economy	TRUE	March 2019	For this flight I was connecti...
2	Bangkok Airways	9	Economy	TRUE	June 2019	Bangkok to Koh Samui. Easy Che...
3	Bangkok Airways	NaN	NaN	FALSE	November 2021	I just want to thank Bangkok A...
4	Bangkok Airways		Economy	FALSE	April 2020	I especially want to thank Khu...
5	Garuda Indonesia	1	Eco	TRUE	July 2023	Flew on GA-682 Jakarta to Soro...
6	Garuda Indonesia	9	Business	TRUE	NaN	Jakarta to Sorong. Check in wa...
7	Garuda Indonesia	9	Business	TRUE	February 2020	Flew Jakarta-Sorong on GA862 t...
8	Philippine Airlines	8	NaN	TRUE	April 2023	This was the second flight in ...
9	Philippine Airlines	7	Premium Economy	FALSE	January 2023	Got Premium economy since they...
10	Philippine Airlines	11	Economy	TRUE	July 2023	Miserable experience: very exp...

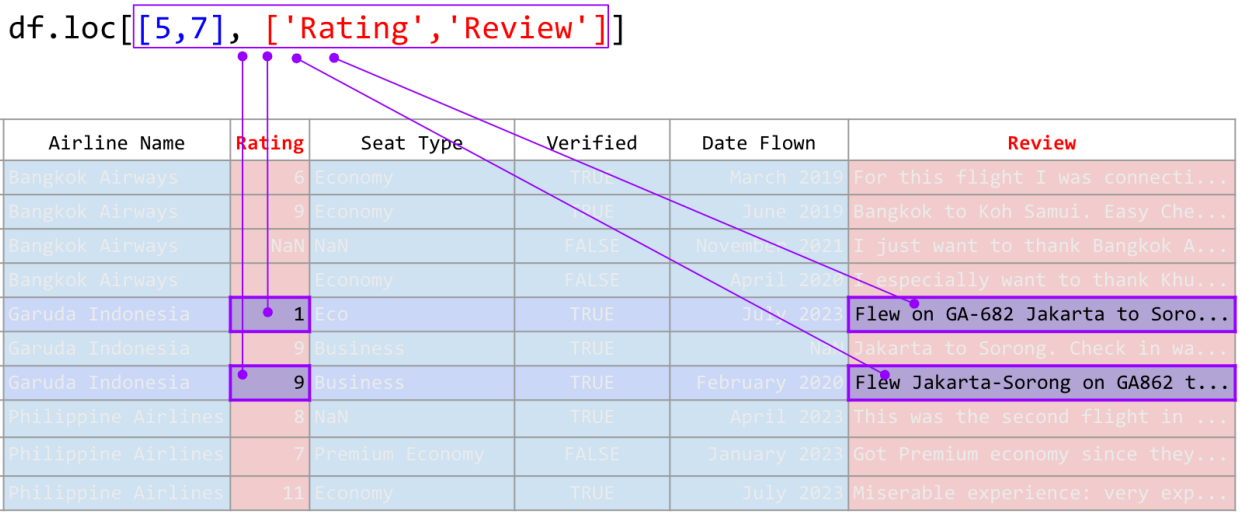
ภาพที่ 29 การใช้คำสั่ง `.loc` เพื่อระบุตำแหน่งในตารางด้วยค่าเดี่ยว (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

คำสั่งข้างต้นแก้ไขค่าข้อมูลในแถวที่ 5 (ค่าดัชนีเท่ากับ 5) และคอลัมน์ *Rating* ให้เป็น 10

2. เลือกด้วยลิสต์ของชื่อคอลัมน์ หรือ ลิสต์ของชื่อแถว

ในกรณีที่เราต้องการเลือกหลายคอลัมน์และหลายแถวในเวลาเดียวกัน เราสามารถใช้ในลิสต์ที่มีสตริงหรือตัวเลขในการเลือกได้ ตัวอย่างเช่น

```
airline_reviews_small.loc[[5, 7], ['Rating', 'Review']]
```



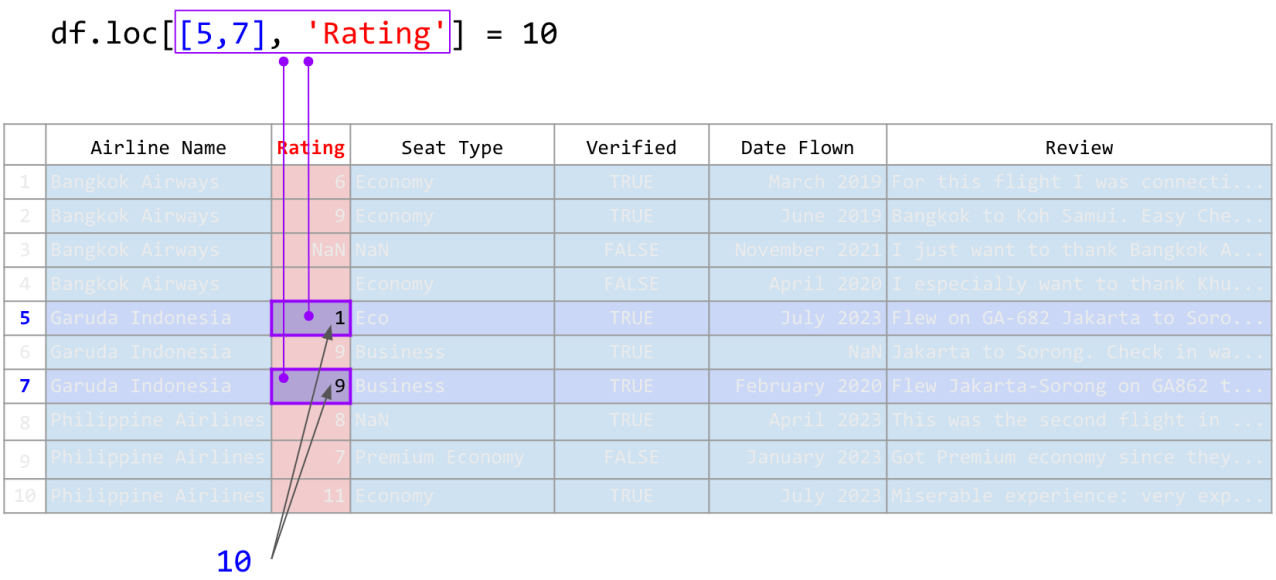
	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	Bangkok Airways	6	Economy	TRUE	March 2019	For this flight I was connecti...
2	Bangkok Airways	9	Economy	TRUE	June 2019	Bangkok to Koh Samui. Easy Che...
3	Bangkok Airways	NaN	NaN	FALSE	November 2021	I just want to thank Bangkok A...
4	Bangkok Airways		Economy	FALSE	April 2020	I especially want to thank Khu...
5	Garuda Indonesia	1	Eco	TRUE	July 2023	Flew on GA-682 Jakarta to Soro...
6	Garuda Indonesia	9	Business	TRUE	NaN	Jakarta to Sorong. Check in wa...
7	Garuda Indonesia	9	Business	TRUE	February 2020	Flew Jakarta-Sorong on GA862 t...
8	Philippine Airlines	8	NaN	TRUE	April 2023	This was the second flight in ...
9	Philippine Airlines	7	Premium Economy	FALSE	January 2023	Got Premium economy since they...
10	Philippine Airlines	11	Economy	TRUE	July 2023	Miserable experience: very exp...

ภาพที่ 30 การใช้คำสั่ง `.loc` เพื่อระบุตำแหน่งในตารางด้วยลิสต์ (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

คำสั่งข้างต้นเลือกแถวที่ 5 และ 7 และคอลัมน์ Rating และ Review

อีกตัวอย่าง เช่น

```
airline_reviews_small.loc[[5, 7], 'Rating'] = 10
```



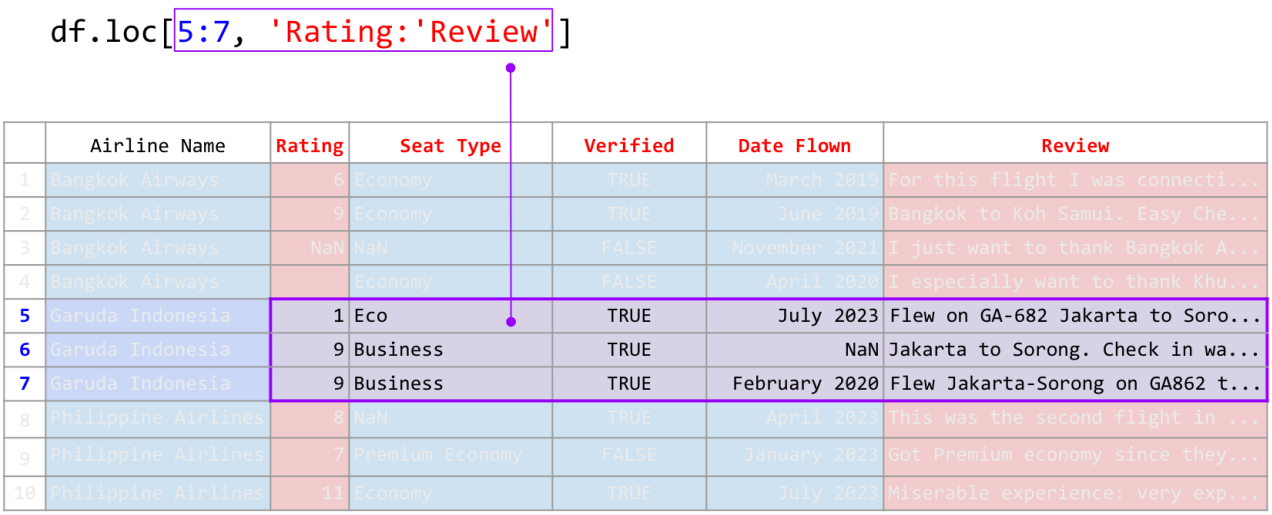
	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	Bangkok Airways	6	Economy	TRUE	March 2019	For this flight I was connecti...
2	Bangkok Airways	9	Economy	TRUE	June 2019	Bangkok to Koh Samui. Easy Che...
3	Bangkok Airways	NaN	NaN	FALSE	November 2021	I just want to thank Bangkok A...
4	Bangkok Airways		Economy	FALSE	April 2020	I especially want to thank Khu...
5	Garuda Indonesia	1	Eco	TRUE	July 2023	Flew on GA-682 Jakarta to Soro...
6	Garuda Indonesia	9	Business	TRUE	NaN	Jakarta to Sorong. Check in wa...
7	Garuda Indonesia	9	Business	TRUE	February 2020	Flew Jakarta-Sorong on GA862 t...
8	Philippine Airlines	8	NaN	TRUE	April 2023	This was the second flight in ...
9	Philippine Airlines	7	Premium Economy	FALSE	January 2023	Got Premium economy since they...
10	Philippine Airlines	11	Economy	TRUE	July 2023	Miserable experience: very exp...

ภาพที่ 31 การใช้คำสั่ง `.loc` เพื่อระบุตำแหน่งในตารางด้วยลิสต์และค่าเดี่ยว (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

3. เลือกด้วยช่วงของชื่อคอลัมน์ หรือ ชื่อแถว

เราสามารถกำหนดช่วงของคอลัมน์หรือแถวโดยใช้เครื่องหมาย : เป็นตัวคั่นระหว่างจุดเริ่มต้นไปถึงจุดหมาย คล้ายคลึงกับการหั่นลิสต์ เช่น

คำสั่ง	คำอธิบาย
`airline_reviews_small.loc[5:7]`	เลือกแถวที่ 5 ถึง 7
`airline_reviews_small.loc[5:7, 'Rating']`	เลือกแถวที่ 5 ถึง 7 และคอลัมน์ Rating
`airline_reviews_small.loc[5:7, 'Rating':'Review']`	เลือกแถวที่ 5 ถึง 7 และคอลัมน์ Rating ถึง Review



	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	Bangkok Airways	6	Economy	TRUE	March 2019	For this flight I was connecti...
2	Bangkok Airways	9	Economy	TRUE	June 2019	Bangkok to Koh Samui. Easy Che...
3	Bangkok Airways	NaN	NaN	FALSE	November 2021	I just want to thank Bangkok A...
4	Bangkok Airways		Economy	FALSE	April 2020	I especially want to thank Khu...
5	Garuda Indonesia	1	Eco	TRUE	July 2023	Flew on GA-682 Jakarta to Soro...
6	Garuda Indonesia	9	Business	TRUE	NaN	Jakarta to Sorong. Check in wa...
7	Garuda Indonesia	9	Business	TRUE	February 2020	Flew Jakarta-Sorong on GA862 t...
8	Philippine Airlines	8	NaN	TRUE	April 2023	This was the second flight in ...
9	Philippine Airlines	7	Premium Economy	FALSE	January 2023	Got Premium economy since they...
10	Philippine Airlines	11	Economy	TRUE	July 2023	Miserable experience: very exp...

ภาพที่ 32 การใช้คำสั่ง `.loc` เพื่อระบุตำแหน่งในตารางด้วยช่วง (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

4. เลือกด้วยลิสต์ของบูลีน

เราสามารถใช้ลิสต์ของบูลีนที่มีจำนวนสมาชิกเท่ากับจำนวนแถว หรือคอลัมน์ วิธีนี้เรียกว่าบูลีนพราง (boolean masking) วิธีนี้เรามักจะใช้กับการเลือกแถวโดยการกำหนดเงื่อนไขตามคอลัมน์ เช่น ถ้าหากเราต้องการเลือกแถวที่ค่าคะแนนสูงกว่าสิบ ซึ่งไม่ควรจะเกิดขึ้นในชุดข้อมูลนี้ เรามีวิธีการกำหนดเงื่อนไขดังนี้

```
condition = airline_reviews_small['Rating'] > 10
```

ซึ่งจะสร้างลิสต์ของบูลีนที่มีค่าเป็น `True` หากแถวแถวนั้นคะแนนสูงกว่า 10 และ `False` หากคะแนนไม่สูงกว่า 10 จะได้ผลลัพท์ดังนี้

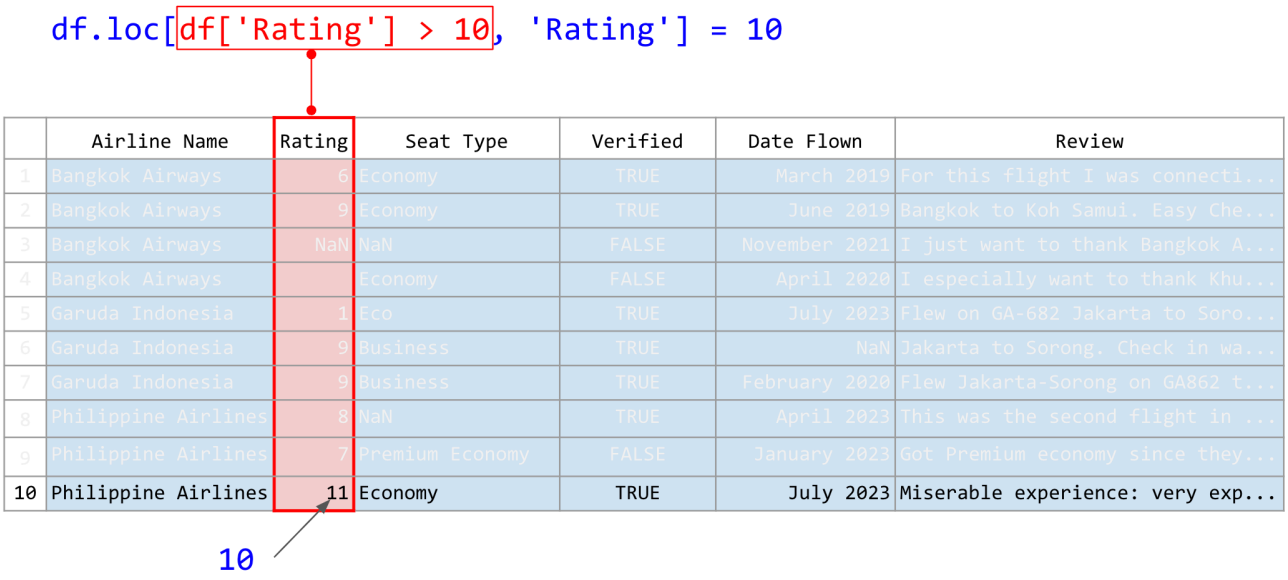
```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9     True
Name: Rating, dtype: bool
```

จากนั้นเราสามารถใช้ลิสต์ของบูลีนนี้ในการเลือกแถวที่ต้องการดังนี้

```
airline_reviews_small.loc[condition]
```

เมื่อนำไปใช้จริงเรามักจะไม่เก็บบูลีนพรางไว้ในตัวแปร แต่จะใช้ไปเลยในการเลือกแถว ดังนี้

```
airline_reviews_small.loc[airline_reviews_small['Rating'] > 10, 'Rating'] = 10
```



	Airline Name	Rating	Seat Type	Verified	Date Flown	Review
1	Bangkok Airways	6	Economy	TRUE	March 2019	For this flight I was connecti...
2	Bangkok Airways	9	Economy	TRUE	June 2019	Bangkok to Koh Samui. Easy Che...
3	Bangkok Airways	NaN	NaN	FALSE	November 2021	I just want to thank Bangkok A...
4	Bangkok Airways		Economy	FALSE	April 2020	I especially want to thank Khu...
5	Garuda Indonesia	1	Eco	TRUE	July 2023	Flew on GA-682 Jakarta to Soro...
6	Garuda Indonesia	9	Business	TRUE	NaN	Jakarta to Sorong. Check in wa...
7	Garuda Indonesia	9	Business	TRUE	February 2020	Flew Jakarta-Sorong on GA862 t...
8	Philippine Airlines	8	NaN	TRUE	April 2023	This was the second flight in ...
9	Philippine Airlines	7	Premium Economy	FALSE	January 2023	Got Premium economy since they...
10	Philippine Airlines	11	Economy	TRUE	July 2023	Miserable experience: very exp...

ภาพที่ 33 การใช้คำสั่ง `.loc` ร่วมกับการใช้บูลีนพรางในเลือกแแถว และสตริงในการเลือกคอลัมน์ ก่อนที่จะแทนค่าด้วยค่าที่กำหนด (ข้อมูลดัดแปลงจาก Airline Reviews Dataset จาก airlinequality.com)

คำสั่งข้างต้นเลือกแถวที่มีคะแนนสูงกว่า 10 โดยการใช้บูลีนพราง และเลือกคอลัมน์ด้วยการใช้สตริงเดี่ยว ๆ จากนั้นกำหนดให้คะแนนเป็น 10 เพราะฉะนั้นแถวที่มีค่าคอลัมน์ *Rating* มากกว่า 10 จะถูกเปลี่ยนเป็น 10 ทั้งหมด

การทำความสะอาดข้อมูลที่เป็นข้อมูลที่เป็นตัวแปรจำแนกประเภท

คอลัมน์ที่เป็นตัวแปรจำแนกประเภท (categorical variable) มีลักษณะคล้ายคลึงกับสตริงที่เป็นข้อความ แต่มีความแตกต่างตรงที่คอลัมน์ที่เป็นตัวแปรจำแนกประเภทมีค่าที่เป็นที่ต้องมาจากเซตที่กำหนดไว้แล้ว ไม่ใช่ค่าอิสระ ตัวอย่างเช่นในชุดข้อมูลตัวอย่าง เรามีคอลัมน์ที่เป็นตัวแปรจำแนกประเภทอยู่ 2 คอลัมน์ คือ *Airline Name* และ *Seat Type* ซึ่งค่าที่อยู่ในคอลัมน์นี้มีค่าที่เป็นชื่อของสายการบิน และชนิดของที่นั่งตามลำดับ ทั้งสองคอลัมน์ต่างเป็นค่าที่มาจากเซตที่กำหนดไว้แล้ว ได้แก่ เซตของชื่อสายการบินทั้งหมด และเซตของประเภทที่นั่งทั้งหมด เพราะฉะนั้นเราต้องตรวจสอบว่าคอลัมน์เหล่านี้มีค่าที่ไม่ได้มาจากเซตที่เหมาะสมหรือไม่

วิธีเบื้องต้นในการตรวจสอบค่าของตัวแปรจำแนกประเภท คือ การใช้ฟังก์ชัน `value_counts()` คำสั่งนี้ใช้ในการแจกแจงว่าคอลัมน์นั้นมีค่าประเภทเป็นอะไรบ้าง และแต่ละค่าปรากฏในชุดข้อมูลกี่ครั้ง การแจกแจงค่าในลักษณะนี้เรียกว่า การกระจายตัว (distribution) ของค่า ตัวอย่างเช่น

```
airline_reviews_small['Seat Type'].value_counts()
```

คำสั่งข้างต้นใช้ `[]` และระบุสตริงที่เป็นชื่อคอลัมน์ ในการเลือกคอลัมน์ที่ต้องการนับ เพราะฉะนั้นคำสั่ง `airline_reviews_small['Seat Type']` จะเข้าถึงและเลือกคอลัมน์ *Seat Type* ออกมาก่อนเป็น `Series` และหลังจากนั้นนำมาเชื่อมด้วยเมท็อด `.value_counts` เพื่อหาการกระจายตัวของค่าที่อยู่ใน `Series` นั้น ผลลัพธ์ที่ได้จะเป็นจำนวนของค่าที่มีอยู่ในคอลัมน์ที่เป็นตัวแปรจำแนกประเภท ดังนี้

```
Economy            4
Business           2
Eco                1
Premium Economy    1
Name: Seat Type, dtype: int64
```

จากผลลัพธ์ที่ได้ เราสามารถเห็นได้ว่าคอลัมน์ *Seat Type* มีค่าที่ไม่ได้มาจากเซตที่กำหนดไว้ คือค่า *Eco* ซึ่งอาจจะเป็นค่าที่ผิดพลาด หรือเป็นค่าที่ไม่ได้กำหนดไว้ในเซต ในกรณีนี้เราควรจะปรึกษาผู้ที่เก็บข้อมูลว่า ค่า *Eco* นั้นมีความหมายว่า *Economy* หรือไม่ ถ้ามีความหมายเหมือนกันให้ทำความสะอาดโดยเปลี่ยนคำว่า *Eco* เป็น *Economy* ดังนี้

```
airline_reviews_small.loc[airline_reviews_small['Seat Type'] == 'Eco', 'Seat Type'] = 'Economy'
```

ในตัวอย่างข้างต้น เราใช้ฟังก์ชัน `loc[]` โดยกำหนดเงื่อนไข `airline_reviews_small['Seat Type'] == 'Eco'` ซึ่งหมายถึงการเลือกแถวที่มีค่าของคอลัมน์ `Seat Type` เป็น `Eco` และเปลี่ยนค่านั้นให้เป็น `Economy` หลังจากนั้นเราสามารถตรวจสอบดูว่าข้อมูลที่เหลืออยู่เพียงพอที่จะทำการวิเคราะห์ข้อมูลต่อไปหรือไม่ โดยใช้ฟังก์ชัน `value_counts()` อีกครั้ง ตัวอย่างเช่น

```
airline_reviews_small['Seat Type'].value_counts()
```

ผลลัพธ์ที่ได้จะเป็นจำนวนของค่าที่มีอยู่ในคอลัมน์ที่เป็นตัวแปรจำแนกประเภท ตัวอย่างเช่น

```
Economy            5
Business           2
Premium Economy    1
Name: Seat Type, dtype: int64
```

การทำความสะอาดข้อมูลที่เป็นข้อมูลข้อความ

ในบทที่แล้วเราได้เรียนรู้วิธีการทำความสะอาดข้อมูลที่เป็นข้อความไปแล้ว เราสามารถนำฟังก์ชันเดียวกันนี้นำมาใช้กับดาตาเฟรมได้ ตัวอย่างเช่น

```
import re
def normalize(tweet):
    # ถ้าเจอ [ก-์] สามตัวขึ้นไป ทำให้เหลือตัวเดียว
```

```
    tweet = re.sub(r'([ก-ฺ])\1{2,}', r'\1', tweet)
    return tweet
```

ฟังก์ชันข้างต้นเป็นฟังก์ชันที่ใช้ในการลบตัวอักษรที่ซ้ำกัน 3 ตัวขึ้นไปในภาษาไทย ในกรณีที่เราต้องการทำความสะอาดข้อมูลที่เป็นข้อความในคอลัมน์ *Review* ในดาตาเฟรม `airline_reviews_small` เราสามารถใช้ฟังก์ชัน `apply()` ในการนำฟังก์ชัน `normalize()` ไปใช้กับทุกแถวในคอลัมน์ *Review* ดังนี้

```
airline_reviews_small['Cleaned review'] = airline_reviews_small['Review'].apply(normalize)
```

คำสั่งทางขวามือของเครื่องหมาย = มีส่วนประกอบหลายส่วนด้วยกัน

- ส่วนแรกคือการเลือกคอลัมน์ *Review* โดยใช้ `[]` และระบุชื่อคอลัมน์ที่ต้องการ
- ส่วนที่สองคือการใช้เมท็อด `.apply()` โดยระบุฟังก์ชันที่ต้องการใช้ ซึ่งในที่นี้คือ `normalize` ส่วนที่สำคัญคืออาร์กิวเมนต์ของ `.apply` จะต้องเป็นชื่อฟังก์ชัน ไม่ใช่การเรียกฟังก์ชัน ดังนั้นไม่ต้องมีวงเล็บ และไม่ต้องมีการเรียกฟังก์ชันด้วยวงเล็บ ภาษาไพทอนจัดเก็บฟังก์ชันเป็นอ็อบเจกต์ประเภทหนึ่ง ซึ่งสามารถนำมาใช้เป็นอาร์กิวเมนต์ได้เหมือนกัน ตัวเลข สตริง หรือลิสต์

นิพจน์ทางฝั่งซ้ายมือของเครื่องหมาย = คล้ายคลึงกับการเข้าถึงคอลัมน์อย่างที่เราเคยเห็นมาแล้ว แต่เป็นการเข้าถึงคอลัมน์ที่เราสร้างขึ้นมาใหม่ ซึ่งเราสามารถใช้ชื่อคอลัมน์ใหม่นี้ในการเข้าถึงคอลัมน์นี้ในภายหลัง ในกรณีนี้เราสร้างคอลัมน์ใหม่ที่ชื่อว่า *Cleaned review* ซึ่งเป็นคอลัมน์ที่เก็บข้อมูลที่มาจากผลลัพธ์ของคำสั่งทางขวามือของเครื่องหมาย = ซึ่งก็คือผลจากการรันฟังก์ชัน `normalize` ลงบนแต่แถวในคอลัมน์ *Review* ในดาตาเฟรม `airline_reviews_small`

การส่งออกข้อมูล

ดาตาเฟรมเป็นโครงสร้างที่สามารถนำไปใช้กับไลบรารีทางวิทยาการข้อมูลได้หลายไลบรารี เช่น scikit-learn TensorFlow PyTorch huggingface และอื่น ๆ ซึ่งเราสามารถนำดาตาเฟรมที่เราทำความสะอาดแล้วไปใช้กับไลบรารีเหล่านี้ได้โดยไม่ต้องทำการเปลี่ยนแปลงข้อมูลใด ๆ อีกต่อไป แต่ในบางกรณีเราอาจจะต้องการส่งออกข้อมูลเป็นไฟล์เพื่อนำไปใช้กับโปรแกรมอื่น ๆ หรือเพื่อนำไปแสดงผล ในกรณีนี้เราสามารถใช้เมท็อด `to_csv()` ในการส่งออกข้อมูลเป็นไฟล์ CSV ได้ ตัวอย่างเช่น

```
airline_reviews_small.to_csv('airline_reviews_small_cleaned.csv', index=False)
```

คำสั่งข้างต้นจะส่งออกดาตาเฟรม `airline_reviews_small` ในรูปแบบของไฟล์ CSV โดยบันทึกไว้ในไฟล์ชื่อ `airline_reviews_small_cleaned.csv` โดยไม่บันทึก index ของแถว ซึ่งเราสามารถเปิดไฟล์นี้ด้วยโปรแกรมที่สามารถเปิดไฟล์ CSV ได้ เช่น Microsoft Excel หรือ Google Sheets หรือนำไปใช้กับโปรแกรมอื่น ๆ ที่รองรับการอ่านไฟล์ CSV ได้

ไลบรารี NLP บางไลบรารีไม่ได้รองรับการใช้ดาตาเฟรมโดยตรง pandas มีคำสั่งที่แปลงดาตาเฟรมเป็นลิสต์ของลิสต์ หรือลิสต์ของสตริง ซึ่งเป็นรูปแบบที่สามารถใช้กับไลบรารีเหล่านี้ได้ ตัวอย่างเช่น

```
airline_reviews_small['Review'].to_list()
```

ผลลัพธ์ที่ได้จะเป็นลิสต์ของสตริงที่อยู่ในคอลัมน์ *Review* ของดาตาเฟรม `airline_reviews_small` ซึ่งเราสามารถนำไปใช้กับไลบรารี NLP ได้โดยตรง เช่น เราอาจจะใช้ไลบรารี pythainlp ในการตัดคำ ตัดประโยค หรือใช้ไลบรารี spacy ในการตรวจจับชื่อเฉพาะ หรือไลบรารีอื่น ๆ ในภาษาไพทอนได้ทั้งหมด

สรุป

ในบทนี้เราได้เรียนรู้วิธีการทำความสะอาดข้อมูลด้วย pandas โดยการตรวจสอบข้อมูลที่หายไป และทำการแทนที่ด้วยค่าที่เหมาะสม การตรวจสอบข้อมูลที่ผิดพลาด และทำการแก้ไขข้อมูลที่ผิดพลาด การทำความสะอาดข้อมูลที่เป็นข้อมูลตัวเลข ข้อมูลที่เป็นข้อมูลตัวแปรจำแนกประเภท และข้อมูลที่เป็นข้อมูลข้อความ นอกจากนี้เรายังได้เรียนรู้วิธีการส่งออกข้อมูลเพื่อส่งต่อให้กับโปรแกรมอื่น หรือไลบรารีอื่น ๆ ในไพทอนเอง ทักษะทั้งหมดนี้จัดเป็นทักษะที่เป็นพื้นฐานสำหรับผู้ที่ต้องการเรียนรู้เกี่ยวกับวิทยาการข้อมูล เนื่องจากว่าไลบรารี pandas สามารถรองรับข้อมูลได้หลากหลายรูปแบบ และรองรับข้อมูลที่มีขนาดใหญ่เกินที่จะใช้โปรแกรมอื่น ๆ ในการวิเคราะห์

คำสั่งที่เราเรียนรู้ในบทนี้ที่จริงมีวิธีการปรับแต่งการใช้งานได้อีกมาก นอกจากนั้นไลบรารี pandas ที่จริงแล้วมีคำสั่งอื่น ๆ อีกมาก จนไม่สามารถครอบคลุมได้หมดในบทเดียว หรือในหนังสือเล่มเดียวได้ เพราะฉะนั้นเนื้อหาวิธีการใช้ในบทนี้เป็นเพียงจุดเริ่มต้นเท่านั้น และเราสามารถเรียนรู้เพิ่มเติมได้จากเอกสารอ้างอิงของ pandas (https://pandas.pydata.org) ซึ่งอธิบายทุกเมท็อดที่มีอยู่ในไลบรารีนี้ และข้อมูลเป็นปัจจุบันเสมอ คำสั่งทั้งหมดที่เราเรียนรู้ในบทนี้สามารถสรุปได้ดังในตารางต่อไปนี้

หมวดหมู่คำสั่ง	คลาส	คำสั่ง	คำอธิบาย
อ่านข้อมูลจากไฟล์	-	`.read_csv`	อ่านข้อมูลจากไฟล์ .csv
อ่านข้อมูลจากไฟล์	-	`.read_excel`	อ่านข้อมูลจากไฟล์ excel (สกุล .xls, .xlsx)
สำรวจข้อมูล	df	`.describe`	แสดงข้อมูลสถิติเชิงพรรณนา เช่น ค่าเฉลี่ย (mean) ส่วนเบี่ยงเบนมาตรฐาน ค่าต่ำสุด
สำรวจข้อมูล	df	`.shape`	แสดงข้อมูลจำนวนแถว และจำนวนคอลัมน์ เช่น (1000, 5) หมายความว่ามี 1,000 แถว 5 คอลัมน์
สำรวจข้อมูล	df	`.columns`	แสดงชื่อคอลัมน์ทั้งหมดที่มีใน DataFrame
เลือกคอลัมน์	df	`[สตริงชื่อคอลัมน์]`	เลือก 1 คอลัมน์ ข้อมูลที่ได้จะกลายเป็น Series
เลือกคอลัมน์	df	`[ลิสต์ชื่อคอลัมน์]`	เลือกหลาย ๆ คอลัมน์ ทำให้เป็น dataframe ที่ผอมลง
เลือกแถว	df	`.sample`	เลือกแถวแบบสุ่ม
เลือกแถว	df	`.head`	เลือกแถวจำนวน ... แถวแรก
เลือกแถว	df	`.tail`	เลือกแถวจำนวน ... แถวสุดท้าย
เลือกคอลัมน์และหรือแถว	df	`.iloc`	เลือกของกลุ่มเซลล์ที่ต้องการ โดยการระบุด้วยพิกัดของเซลล์
เลือกคอลัมน์และหรือแถว	df	`.loc`	เลือกของกลุ่มเซลล์ที่ต้องการ โดยการระบุด้วยชื่อแถว ชื่อคอลัมน์
ข้อมูลที่หายไป	df	`.isnull().sum()`	หาจำนวนแถวที่ข้อมูลหายไป
ข้อมูลที่หายไป	df	`.dropna`	กำจัดแถวที่มีข้อมูลขาดหายไป
ข้อมูลที่หายไป	Series	`.fillna()`	เติมข้อมูลที่หายไปด้วยค่าที่ระบุ
คอลัมน์ที่มีตัวแปรจำแนกประเภท	Series	`.astype('category')`	แปลงให้เป็นตัวแปรจำแนกประเภท
คอลัมน์ที่มีตัวแปรจำแนกประเภท	Series	`.value_counts`	หาการกระจายตัวของตัวแปร
คอลัมน์ที่มีตัวเลข	Series	`.astype(int)`	แปลงคอลัมน์นี้ให้เป็นจำนวนเต็ม
Numerical columns	Series	`.astype(float)`	แปลงคอลัมน์นี้ให้เป็นจำนวนจริง