



# Intro to Python and Pandas

*DataFrames for Data Science*

Sébastien Biass 

*sebastien.biass@unige.ch*

*Earth Sciences*

October 15, 2025

# Background



We assume that you all followed Guy Simpson's Python crash course

**pandas**: A **package** for data manipulation and analysis handling **structured data**

- **Reading/writing data** from common formats (CSV, Excel, JSON, etc.)
- Handling **missing data**
- **Filtering, sorting, reshaping** and **grouping** data
- **Aggregating** data (sum, mean, count, etc.)
- **Time series support** (date ranges, frequency conversions)
- **Statistical operations**



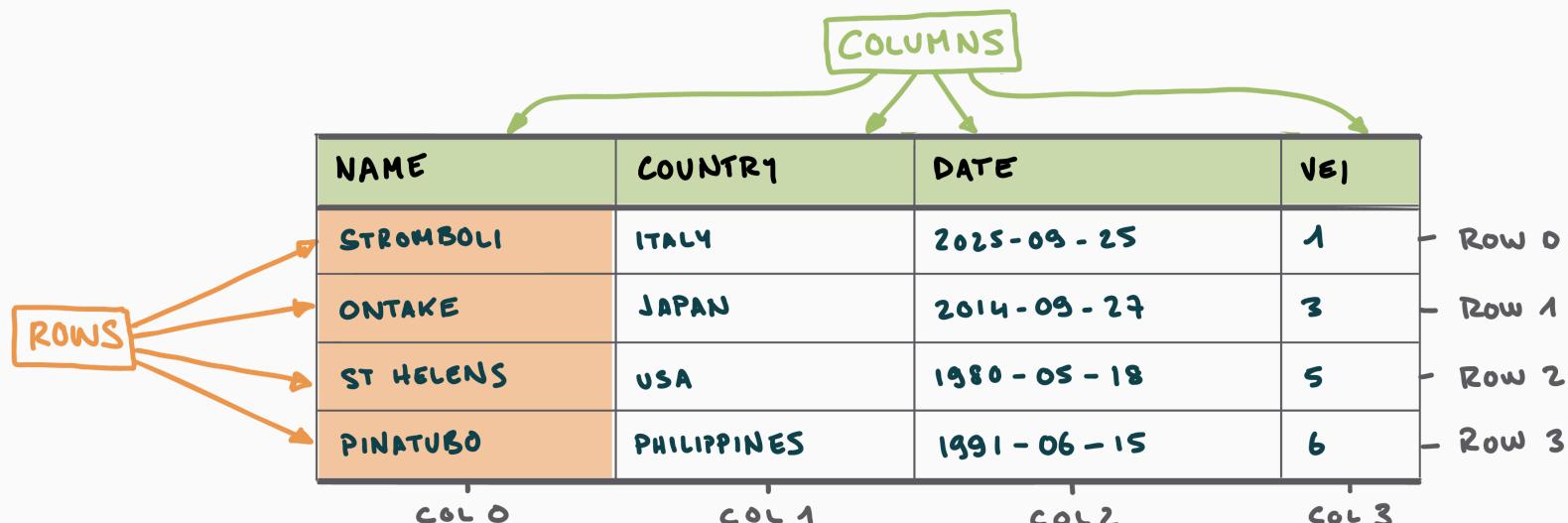
# Today's objectives

Understand what is a **pandas** DataFrame and its basic anatomy

- How to load data in a DataFrame
- How to access data → *query by label/position*
- How to filter data → *comparison and logical operators*
- How to rearrange data → *sorting values*
- How to operate on data → *arithmetic and string operations*

# Introduction to pandas

# Anatomy of a DataFrame



- Similar to *Excel* → contains *tabular data* composed of **rows** and **columns**
- In *Excel*:
  - **Rows** are accessed using *numbers*
  - **Columns** are accessed using *letters*

# Anatomy of a DataFrame

NAME	COUNTRY	DATE	VEI	COLUMNS
INDEX				= LABELS ALONG COLUMNS → df.columns
STROMBOLI	ITALY	2025-09-25	1	Row 0
ONTAKE	JAPAN	2014-09-27	3	Row 1
ST HELENS	USA	1980-05-18	5	Row 2
PINATUBO	PHILIPPINES	1991-06-15	6	Row 3

COL 0                    COL 1                    COL 2

- Unlike Excel, **rows** and **columns** can be labelled
  - Index refers to the label of the **rows**. In the *index*, **values are usually unique** - meaning that each entry has a different label.
  - Column refers to the label of - logically - the **columns**

# Data structure

# The dataset

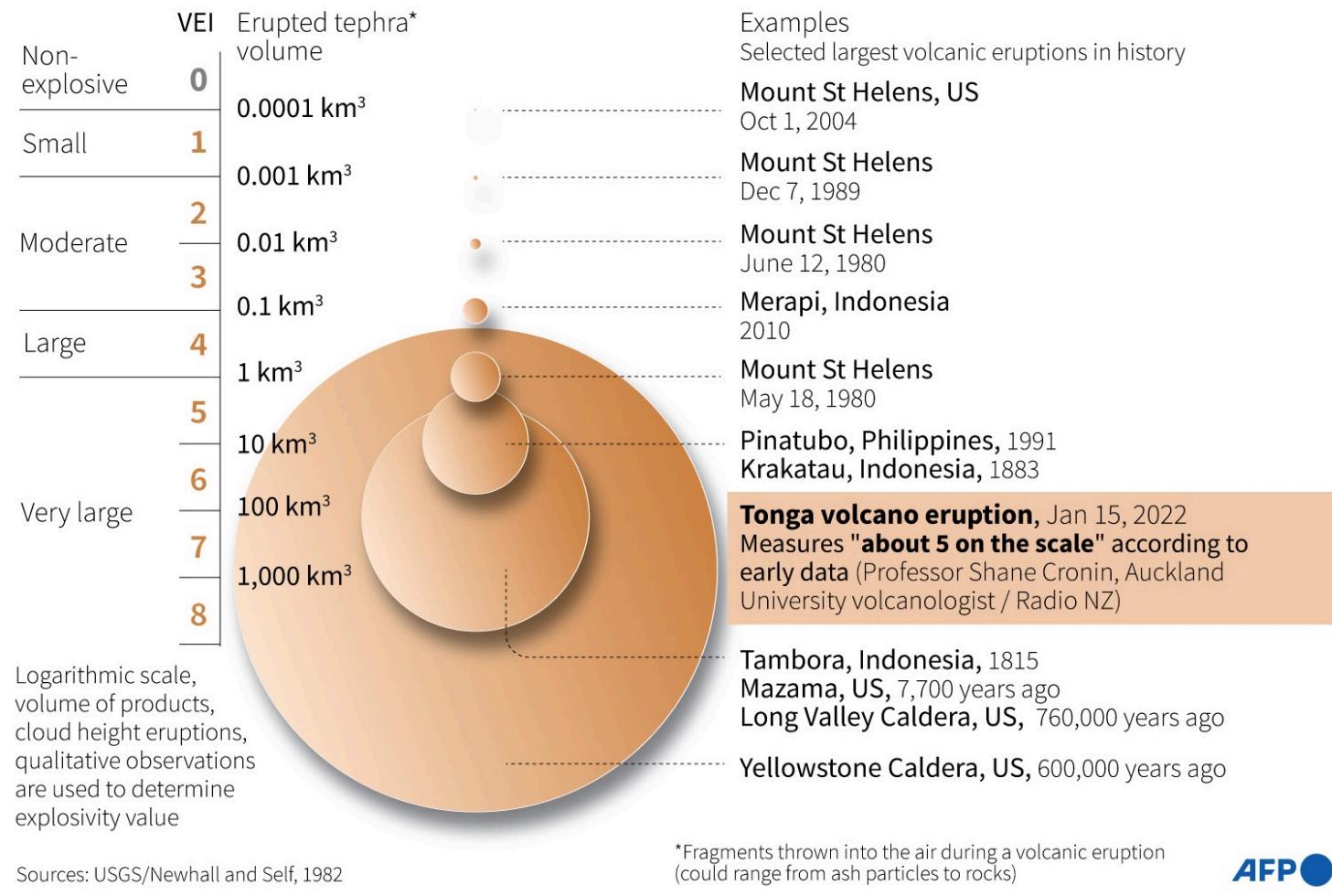
Synthetic dataset of **selected volcanic eruptions** → first 5 rows:

Name	Country	Date	VEI	Latitude	Longitude
St. Helens	USA	1980-05-18	5	46.1914	-122.196
Pinatubo	Philippines	1991-04-02	6	15.1501	120.347
El Chichón	Mexico	1982-03-28	5	17.3559	-93.2233
Galunggung	Indonesia	1982-04-05	4	-7.2567	108.077
Nevado del Ruiz	Colombia	1985-11-13	3	4.895	-75.322

# Volcanic explosivity index (VEI)

## Volcanic explosivity index

Measures the relative explosivity of volcanic eruptions





# Setting up the notebook

- We start by importing the `pandas` library
- We import it under the name `pd` - which is faster to type!

```
1 # Import the required packages  
2 import pandas as pd
```



# Setting up the notebook

- We then load the specified data with the `pd.read_csv()` function
- This returns a `DataFrame` object in a variable named `df`

```
1 # Import the required packages
2 import pandas as pd
3
4 # Read the data
5 df = pd.read_csv('data/dummy_volcanoes.csv', parse_dates=['Date']) # Load data
```



# Setting up the notebook

- We print some data for inspection with `df.head()`
- The functions are now directly called from the `DataFrame df` object

```
1 # Import the required packages
2 import pandas as pd
3
4 # Read the data
5 df = pd.read_csv('../class1/data/dummy_volcanoes.csv', parse_dates=['Date']) # Load data
6
7 # Show the first 3 rows
8 df.head(3)
```

	Name	Country	Date	VEI	Latitude	Longitude
0	St. Helens	USA	1980-05-18	5	46.1914	-122.1956
1	Pinatubo	Philippines	1991-04-02	6	15.1501	120.3465
2	El Chichón	Mexico	1982-03-28	5	17.3559	-93.2233

# Setting the index

NAME	COUNTRY	DATE	VEI
STROMBOLI	ITALY	2025-09-25	1
ONTAKE	JAPAN	2014-09-27	3
ST HELENS	USA	1980-05-18	5
PINATUBO	PHILIPPINES	1991-06-15	6

**INDEX** = LABELS ALONG ROWS → df.index

**COLUMNS** = LABELS ALONG COLUMNS → df.columns

COL 0      COL 1      COL 2

Row 0  
Row 1  
Row 2  
Row 3

# Setting the index

- ...for now, the index (**→ the first column**) is and *integer*
- This might be acceptable in datasets where the *label* is not important

```
1 # Show the first 3 rows  
2 df.head(3)
```

	Name	Country	Date	VEI	Latitude	Longitude
0	St. Helens	USA	1980-05-18	5	46.1914	-122.1956
1	Pinatubo	Philippines	1991-04-02	6	15.1501	120.3465
2	El Chichón	Mexico	1982-03-28	5	17.3559	-93.2233

# Setting the index

- Here we want to access the data using the **name of the volcano**
- We **set the index** using `set_index()`

```
1 # Set the index to the 'Name' column
2 df = df.set_index('Name')
3
4 # Show the first 3 rows
5 df.head(3)
```

Name	Country	Date	VEI	Latitude	Longitude
St. Helens	USA	1980-05-18	5	46.1914	-122.1956
Pinatubo	Philippines	1991-04-02	6	15.1501	120.3465
El Chichón	Mexico	1982-03-28	5	17.3559	-93.2233



# Exploring data

- Here are some **basic functions** to review the structure of the dataset:

Function	Description
<code>df.head()</code>	Prints the <i>first</i> 5 rows of the DataFrame.
<code>df.tail()</code>	Prints the <i>last</i> 5 rows of the DataFrame.
<code>df.info()</code>	Displays some info about the DataFrame, including the number of rows (entries) and columns.
<code>df.shape</code>	Returns a list containing the number of rows and columns of the DataFrame.
<code>df.index</code>	Returns a list containing the index along the <i>rows</i> of the DataFrame.
<code>df.columns</code>	Returns a list containing the index along the <i>columns</i> of the DataFrame.

# Sorting data

- Sorting numerical, datetime or strings using `.sort_values`
- Importance of **documentation** to understand arguments

```
1 df.sort_values('VEI').head() # Sort volcanoes by VEI in ascending number
```

	Country	Date	VEI	Latitude	Longitude	
	Name					
	Nyiragongo	DR Congo	2021-05-22	1	-1.5200	29.2500
	Merapi	Indonesia	2023-12-03	2	-7.5407	110.4457
	Ontake	Japan	2014-09-27	2	35.5149	137.4781
	Kīlauea	USA	2018-05-03	2	19.4194	-155.2811
	Etna	Italy	2021-03-16	2	37.7510	15.0044

# Sorting data

- Sorting numerical, datetime or strings using `.sort_values`
- Importance of **documentation** to understand arguments

```
1 df.sort_values('VEI').head() # Sort volcanoes by VEI in ascending number  
2 df.sort_values('Date', ascending=False).head() # Sort volcanoes by eruption dates from recent
```

	Country	Date	VEI	Latitude	Longitude
Name					
Merapi	Indonesia	2023-12-03	2	-7.5407	110.4457
Cleveland	USA	2023-05-23	3	52.8250	-169.9444
Sinabung	Indonesia	2023-02-13	3	3.1719	98.3925
Nyiragongo	DR Congo	2021-05-22	1	-1.5200	29.2500
La Soufrière	Saint Vincent	2021-04-09	4	13.2833	-61.3875

# Sorting data

- Sorting numerical, datetime or strings using `.sort_values`
- Importance of **documentation** to understand arguments

```
1 df.sort_values('VEI').head() # Sort volcanoes by VEI in ascending number
2 df.sort_values('Date', ascending=False).head() # Sort volcanoes by eruption dates from recent
3 df.sort_values('Country').head() # Also works on strings to sort alphabetically
```

	Country	Date	VEI	Latitude	Longitude
	Name				
Calbuco	Chile	2015-04-22	4	-41.2972	-72.6097
Nevado del Ruiz	Colombia	1985-11-13	3	4.8950	-75.3220
Nyiragongo	DR Congo	2021-05-22	1	-1.5200	29.2500
Eyjafjallajökull	Iceland	2010-04-14	4	63.6333	-19.6111
Galunggung	Indonesia	1982-04-05	4	-7.2567	108.0771

# Sorting data

- Sorting numerical, datetime or strings using `.sort_values`
- Importance of **documentation** to understand arguments

```
1 df.sort_values('VEI').head() # Sort volcanoes by VEI in ascending number
2 df.sort_values('Date', ascending=False).head() # Sort volcanoes by eruption dates from recent
3 df.sort_values('Country').head() # Also works on strings to sort alphabetically
4 df.sort_values(['Latitude', 'Longitude']).head() # Sorting using multiple columns
```

	Country	Date	VEI	Latitude	Longitude
Name					
Calbuco	Chile	2015-04-22	4	-41.2972	-72.6097
Agung	Indonesia	2017-11-21	3	-8.3422	115.5083
Merapi	Indonesia	2023-12-03	2	-7.5407	110.4457
Galunggung	Indonesia	1982-04-05	4	-7.2567	108.0771
Tavurvur	Papua New Guinea	2014-08-29	3	-4.3494	152.2847

# Querying data

# Accessing data in a DataFrame

NAME	COUNTRY	DATE	VEI
STROMBOLI	ITALY	2025-09-25	1
ONTAKE	JAPAN	2014-09-27	3
ST HELENS	USA	1980-05-18	5
PINATUBO	PHILIPPINES	1991-06-15	6

**INDEX** = LABELS ALONG ROWS → df.index

**COLUMNS** = LABELS ALONG COLUMNS → df.columns

COL 0      COL 1      COL 2

Row 0  
Row 1  
Row 2  
Row 3

# Accessing data in a DataFrame

## Option 1: label-based indexing

- Use the labels of **index** and **columns** to retrieve data
- Function to use: `df.loc`

`df.loc['ONTAKE', 'DATE']` → 2014-09-27

NAME	COUNTRY	DATE	VEI	
STROMBOLI	ITALY	2025-09-25	1	- Row 0
ONTAKE	JAPAN	2014-09-27	3	- Row 1
ST HELENS	USA	1980-05-18	5	- Row 2
PINATUBO	PHILIPPINES	1991-06-15	6	- Row 3

# Accessing data in a DataFrame

## Option 2: position-based indexing

- Use the positions of **index** and **columns** to retrieve data
- Function to use: `df.iloc`

`df.iloc [0,2] → USA`

NAME	COUNTRY	DATE	VEI	
STROMBOLI	ITALY	2025-09-25	1	- Row 0
ONTAKE	JAPAN	2014-09-27	3	- Row 1
ST HELENS	USA	1980-05-18	5	- Row 2
PINATUBO	PHILIPPINES	1991-06-15	6	- Row 3

COL 0                    COL 1                    COL 2

# Label-based indexing: Rows

- Query a **row** with `.loc` → Use **square brackets** `[ ]`
  - Query the *row* for which the *index label* is `Calbuco`
  - Returns all *columns*

```
1 df.loc['Calbuco']
```

```
Country          Chile
Date    2015-04-22 00:00:00
VEI                  4
Latitude        -41.2972
Longitude       -72.6097
Name: Calbuco, dtype: object
```

```
1 df.loc[['Calbuco']]
```

Country	Date	VEI	Latitude	Longitude
Name				
Calbuco	Chile	2015-04-22	-41.2972	-72.6097

→ Returns a `pd.Series`

→ Returns a `pd.DataFrame`

# Label-based indexing: Rows

- Query **multiple rows** with `.loc`
  - Query the rows for which the *index labels* are `Calbuco` or `Taal`
  - Returns all *columns*

```
1 df.loc[['Calbuco', 'Taal']]
```

	Country	Date	VEI	Latitude	Longitude
Name					
Calbuco	Chile	2015-04-22	4	-41.2972	-72.6097
Taal	Philippines	2020-01-12	4	14.0020	120.9934

# Label-based indexing: Columns

- **Query columns**

- Query the *columns* for which the *column labels* are `Country` or `VEI`
- Returns all rows

- **Option 1:** with `.loc`:

```
1 df.loc[:, ['Country', 'VEI']].head(3)
```

- **Option 2:** without `.loc`:

```
1 df[['Country', 'VEI']].head(3)
```

Country VEI		
Name		
St. Helens	USA	5
Pinatubo	Philippines	6
El Chichón	Mexico	5

Country VEI		
Name		
St. Helens	USA	5
Pinatubo	Philippines	6
El Chichón	Mexico	5

# Label-based indexing: Rows and Columns

- Again, choice on whether to use `.loc` to query columns
- Option 1:** Columns are specified **inside** `.loc`:
- Option 2:** Columns are specified **outside** `.loc`:

```
1 df.loc[['Calbuco', 'Taal'], ['Country', 'V
```

```
1 df.loc[['Calbuco', 'Taal']][['Country', 'V
```

Country VEI

Name

Calbuco Chile 4

Taal Philippines 4

Country VEI

Name

Calbuco Chile 4

Taal Philippines 4

# Position-based indexing: Rows

- Query a **row** with `.iloc`

→ Returns all *columns*

- One row** (first row):

```
1 df.iloc[0]
```

Country	Date	VEI	Latitude	Longit
Name				
St. USA	1980- 5 05-18	46.1914	-122.1	
Helens				

- Range of rows:** (rows 3-4):

```
1 df.iloc[2:4]
```

Country	Date	VEI	Latitude
Name			
El Chichón	1982- 5 03-28	17.3559	
Galunggung	1982- 4 04-05	-7.2567	

# Position-based indexing: Rows

- Query rows from the end:
- Example:
  - Get the **last 5 rows** of the DataFrame:

```
1 df.iloc[-5:]
```

	Country	Date	VEI	Latitude	Longitude
	Name				
La Soufrière	Saint Vincent	2021-04-09	4	13.2833	-61.3875
Calbuco	Chile	2015-04-22	4	-41.2972	-72.6097
St. Augustine	USA	2006-03-27	3	57.8819	-155.5611
Eyjafjallajökull	Iceland	2010-04-14	4	63.6333	-19.6111
Cleveland	USA	2023-05-23	3	52.8250	-169.9444

# Position-based and label-based queries

- Mix position-based and label-based indexing:
  - **Rows** → *labels*
  - **Columns** → *positions*

```
1 df.iloc[0:5][['Country', 'VEI']]
```

Country VEI		
Name		
St. Helens	USA	5
Pinatubo	Philippines	6
El Chichón	Mexico	5
Galunggung	Indonesia	4
Nevado del Ruiz	Colombia	3

# Filtering data

# Boolean indexing

- **Filtering** data with boolean indexing
  - Returns either **True** or **False** depending on whether the **condition** is satisfied
- Example:

```
1 a = 1  
2 b = 2
```

## Comparison operators:

Operator	Meaning	Example	Result
<code>==</code>	Equal to	<code>a == b</code>	<code>False</code>
<code>!=</code>	Not equal to	<code>a != b</code>	<code>True</code>
<code>&gt;</code>	Greater than	<code>a &gt; b</code>	<code>False</code>
<code>&lt;</code>	Less than	<code>a &lt; b</code>	<code>True</code>
<code>&gt;=</code>	Greater than or equal	<code>a &gt;= b</code>	<code>False</code>
<code>&lt;=</code>	Less than or equal	<code>a &lt;= b</code>	<code>True</code>

# Boolean indexing: Example

- Query all volcanoes where `VEI == 4`

```
1 df['VEI'] == 4
```

Name	
St. Helens	False
Pinatubo	False
El Chichón	False
Galunggung	True
Nevado del Ruiz	False
Merapi	False
Ontake	False
Soufrière Hills	False
Etna	False
Nyiragongo	False
Kīlauea	False
Agung	False
Tavurvur	False
Sinabung	False
Taal	True
La Soufrière	True
Calbuco	True



# Boolean indexing: Example

- Query all volcanoes where `VEI == 4`

```
1 df.loc[df['VEI'] == 4]
```

	Country	Date	VEI	Latitude	Longitude
	Name				
Galunggung	Indonesia	1982-04-05	4	-7.2567	108.0771
Taal	Philippines	2020-01-12	4	14.0020	120.9934
La Soufrière	Saint Vincent	2021-04-09	4	13.2833	-61.3875
Calbuco	Chile	2015-04-22	4	-41.2972	-72.6097
Eyjafjallajökull	Iceland	2010-04-14	4	63.6333	-19.6111

# Boolean indexing: Strings

- **Filtering** also works with strings
  - Use **string comparison** operations

- **Example:**

```
1 df.loc[df['Country'] == 'Indonesia']
```

	Country	Date	VEI	Latitude	Longitude
	Name				
Galunggung	Indonesia	1982-04-05	4	-7.2567	108.0771
Merapi	Indonesia	2023-12-03	2	-7.5407	110.4457
Agung	Indonesia	2017-11-21	3	-8.3422	115.5083
Sinabung	Indonesia	2023-02-13	3	3.1719	98.3925



# Boolean indexing: Strings

- **Filtering** also works with strings
  - Use **string comparison** operations

- **String comparison operators:**

Operation	Example	Description
contains	<code>df['Name'].str.contains('Soufrière')</code>	Checks if each string contains a substring
startswith	<code>df['Name'].str.startswith('E')</code>	Checks if each string starts with a substring
endswith	<code>df['Name'].str.endswith('o')</code>	Checks if each string ends with a substring

# Operations



# Data management operations

- Common data management functions for pandas columns:

Operation	Example	Description
Round	<code>df['VEI'].round(1)</code>	Rounds values to the specified number of decimals
Floor	<code>df['VEI'].apply(np.floor)</code>	Rounds values down to the nearest integer
Ceil	<code>df['VEI'].apply(np.ceil)</code>	Rounds values up to the nearest integer
Absolute value	<code>df['VEI'].abs()</code>	Returns the absolute value of each element
Fill missing	<code>df['VEI'].fillna(0)</code>	Replaces missing values with a specified value

- Round the '**Latitude**' to two decimals → make sure you store the output!

```
1 df['Latitude'] = df['Latitude'].round(2)
```

# Arithmetic operations

- Arithmetic operations on parts of the DataFrame ( $\rightarrow$  columns) using native Python arithmetic operators

Operation	Symbol	Example	Description
Addition	+	<code>df['VEI'] + 1</code>	Adds a value to each element
Subtraction	-	<code>df['VEI'] - 1</code>	Subtracts a value from each element
Multiplication	*	<code>df['VEI'] * 2</code>	Multiplies each element by a value
Division	/	<code>df['VEI'] / 2</code>	Divides each element by a value
Exponentiation	**	<code>df['VEI'] ** 2</code>	Raises each element to a power
Modulo	%	<code>df['VEI'] % 2</code>	Remainder after division for each element

- Divide `VEI` by 2 and store results a new column ( $\rightarrow$  `VEI_haved`)

```
1 df['VEI_halved'] = df['VEI'] / 2
```

# Expanded arithmetic operations

- The range of arithmetic operations can be expanded using `numpy`

Operation	Symbol	Example	Description
Exponentiation	<code>np.power</code>	<code>np.power(df['VEI'], 2)</code>	Element-wise exponentiation
Square root	<code>np.sqrt</code>	<code>np.sqrt(df['VEI'])</code>	Element-wise square root
Logarithm (base e)	<code>np.log</code>	<code>np.log(df['VEI'])</code>	Element-wise natural logarithm
Logarithm (base 10)	<code>np.log10</code>	<code>np.log10(df['VEI'])</code>	Element-wise base-10 logarithm
Exponential	<code>np.exp</code>	<code>np.exp(df['VEI'])</code>	Element-wise exponential ( $e^x$ )

- Divide `VEI` by 2 and store results a new column ( $\rightarrow$  `VEI_haved`)

```
1 df['VEI_square'] = np.power(df['VEI'], 2)
```