

Foundations of Data Science

2024 / 2025

TABLE OPERATIONS

Tables

Special Column, called “Index”, or
“ID”, or “Key”
Usually, no duplicates Allowed

Variables
(also called Attributes, or
Columns, or Labels)

Observations,
Rows, or
Tuples

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

The diagram illustrates the components of a table. A purple arrow points from the text 'Special Column, called “Index”, or “ID”, or “Key” Usually, no duplicates Allowed' to the 'ID' column header. Three blue arrows point from the text 'Variables (also called Attributes, or Columns, or Labels)' to the 'age', 'wgt_kg', and 'hgt_cm' column headers. Four grey arrows point from the text 'Observations, Rows, or Tuples' to the four data rows of the table.

Tables

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

ID	Address
1	College Park, MD, 20742
2	Washington, DC, 20001
3	Silver Spring, MD 20901

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085

1. Select/slicing

- Select only some of the rows, or some of the columns, or a combination

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Only columns
ID and Age

ID	age
1	12.2
2	11.0
3	15.6
4	35.1

Only rows with
wgt > 41

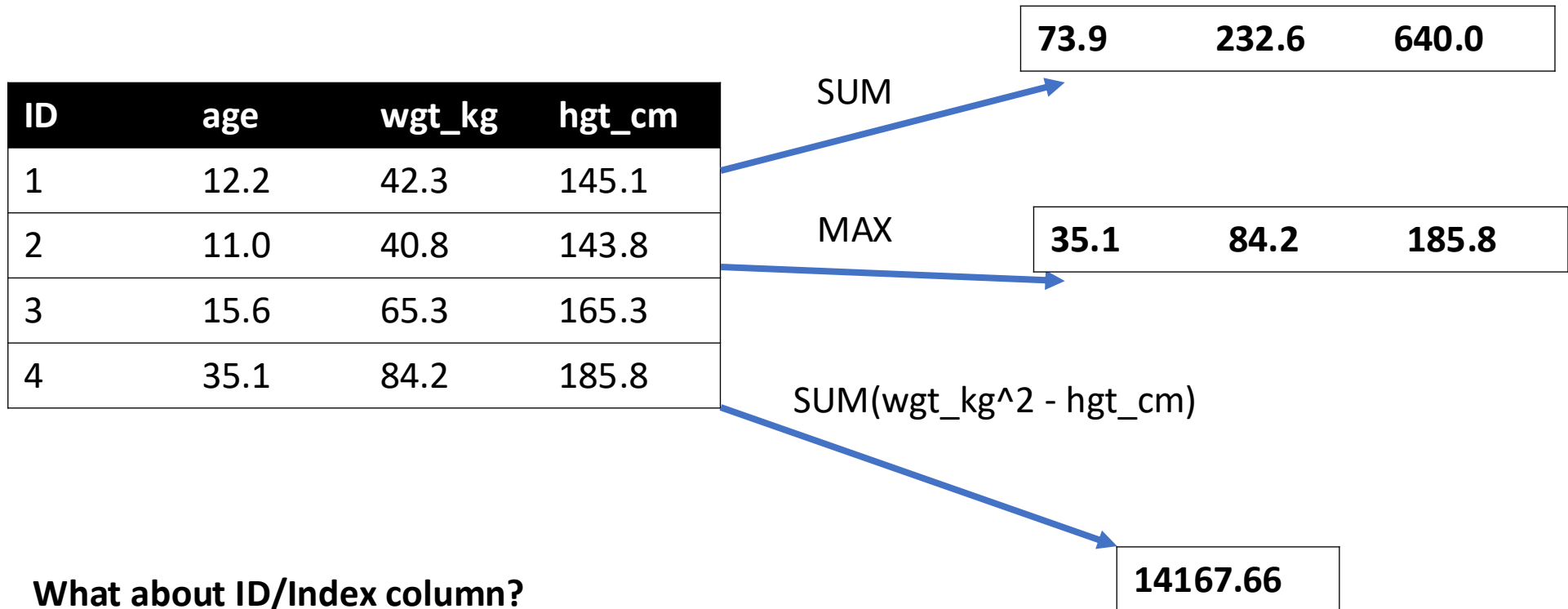
ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Both

ID	age
1	12.2
3	15.6
4	35.1

2. Aggregate/Reduce

- Combine values across a column into a single value



What about ID/Index column?

Usually not meaningful to aggregate across it

May need to explicitly add an ID column

3. Map

- Apply a function to every row, possibly creating more or fewer columns

ID	Address
1	College Park, MD, 20742
2	Washington, DC, 20001
3	Silver Spring, MD 20901



ID	City	State	Zipcode
1	College Park	MD	20742
2	Washington	DC	20001
3	Silver Spring	MD	20901

4. Group By

- Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'A'



A = foo

ID	B	C
1	3	6.6
3	4	3.1
4	3	8.0
7	4	2.3
8	3	8.0

A = bar

ID	B	C
2	2	4.7
5	1	1.2
6	2	2.5

4. Group By

- Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'B'

B = 1

ID	A	C
5	bar	1.2

B = 2

ID	A	C
2	bar	4.7
6	bar	2.5

B = 3

ID	A	C
1	foo	6.6
4	foo	8.0
8	foo	8.0

B = 4

ID	A	C
3	foo	3.1
7	foo	2.3

4. Group By

- Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'A', 'B'



A = bar, B = 1

ID	C
5	1.2

A = bar, B = 2

ID	C
2	4.7
6	2.5

A = foo, B = 3

ID	C
1	6.6
4	8.0
8	8.0

A = foo, B = 4

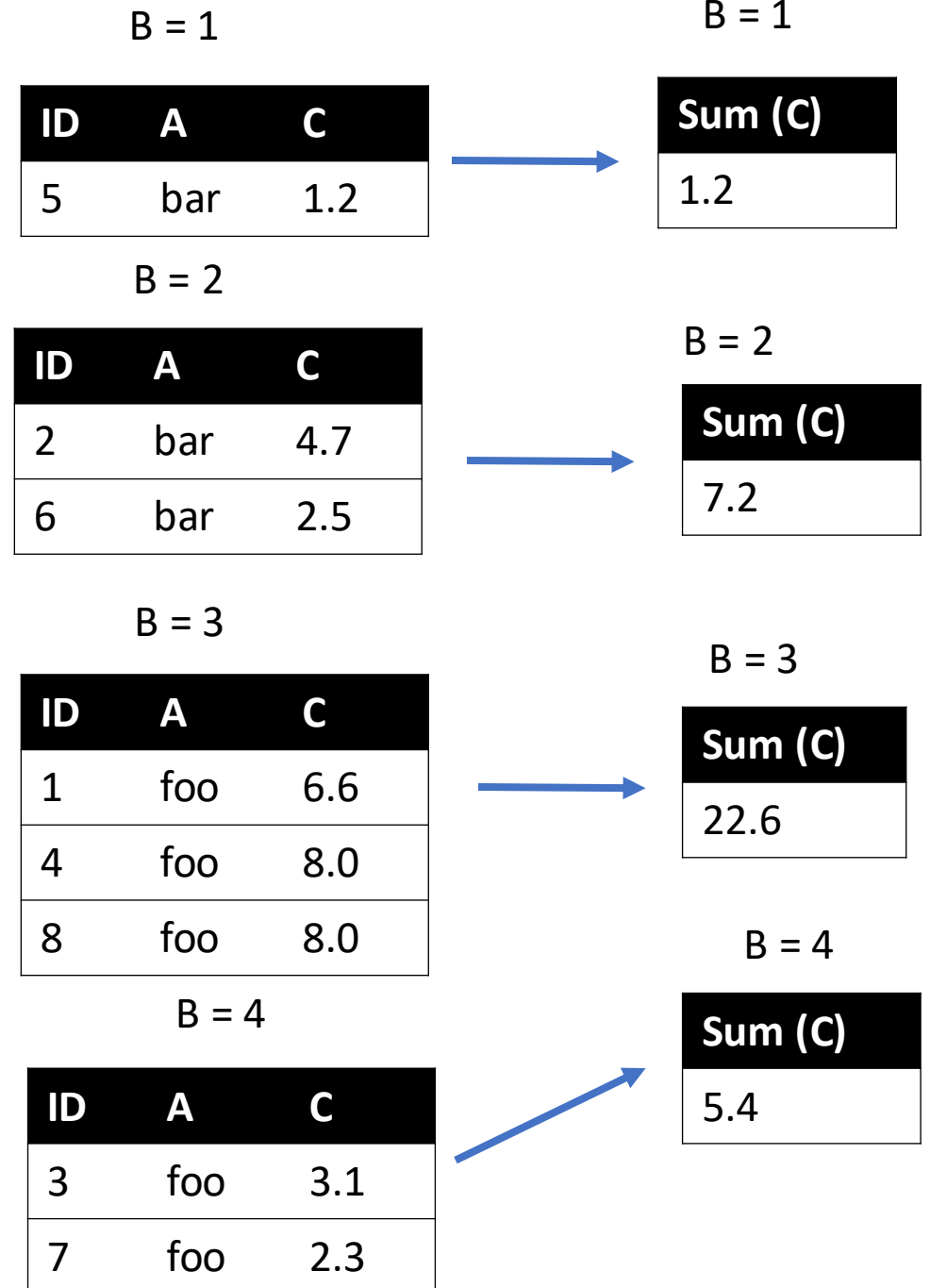
ID	C
3	3.1
7	2.3

5. Group By Aggregate

- Compute one aggregate per group

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group by 'B'
Sum on C



5. Group By Aggregate

- Final result usually seen as a table

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group by 'B'
Sum on C

B = 1

Sum (C)
1.2

B = 2

Sum (C)
7.2

B = 3

Sum (C)
22.6

B = 4

Sum (C)
5.4



B	SUM(C)
1	1.2
2	7.2
3	22.6
4	5.4

6. Union/Intersection/Difference

- Set operations – only if the two tables have identical attributes/columns

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0

U

ID	A	B	C
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0



ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

7. Merge or Join

- Combine rows/tuples across two tables if they have the same key



What about IDs not present in both tables?

Often need to keep them around

Can “pad” with NaN

7. Merge or Join

- Combine rows/tuples across two tables if they have the same key
- Outer joins can be used to "pad" IDs that don't appear in both tables
- Three variants: LEFT, RIGHT, FULL
- SQL Terminology -- Pandas has these operations as well

ID	A	B
1	foo	3
2	bar	2
3	foo	4
4	foo	3



ID	C
1	1.2
2	2.5
3	2.3
5	8.0



ID	A	B	C
1	foo	3	1.2
2	bar	2	2.5
3	foo	4	2.3
4	foo	3	NaN
5	NaN	NaN	8.0

NUMPY

Numpy, Scipy

- Numpy
 - Multidimensional arrays
 - Linear algebra
- Scipy
 - Linear algebra
 - Statistics, inc. distributions
 - Optimization, Interpolation, ...

Numpy data types

- ndarray: n-dimensional array
 - Fixed size, same data type
- int8, int16, int32, int64
- uint8, uint16, uint32, uint64
- float16, float32, float64
- complex64, complex128

```
x = np.array([3, 6, 2])  
np.arange(3, dtype=np.uint8)
```

1-dimensional arrays

```
import numpy as np

a1 = np.array([1.87, 1.87, 1.82, 1.91, 1.90, 1.85])
a2 = np.array([81.65, 97.52, 95.25, 92.98, 86.18, 88.45])

np.mean(a1)

# 1.87

np.mean(a2)

# 90.33833333333335
```

N-dimensional arrays

```
import numpy as np

x = np.arange(27).reshape((3,3,3))

# array([[[ 0,  1,  2],
#         [ 3,  4,  5],
#         [ 6,  7,  8]],
#        [[ 9, 10, 11],
#         [12, 13, 14],
#         [15, 16, 17]],
#        [[18, 19, 20],
#         [21, 22, 23],
#         [24, 25, 26]]])

x.sum(axis=0)

# array([[27, 30, 33],
#        [36, 39, 42],
#        [45, 48, 51]])
```

Useful methods

```
np.zeros((3, 4))
```

```
np.linspace(-3, 3, 20)
```

```
np.random.random((3, 4))
```

```
np.diag(range(4))
```

Indexing and slicing

```
x = np.arange(30)  
x[-2:]
```

```
array([28, 29])
```

```
x = x.reshape(3, 10)  
x
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],  
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])
```

```
x[1]
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
x[2, :5]
```

```
array([20, 21, 22, 23, 24])
```

```
x[:, 1::3]
```

```
array([[ 1,  4,  7],  
       [11, 14, 17],  
       [21, 24, 27]])
```

Array operations

```
x1 = np.arange(9).reshape(3,3)
x2 = x1 + 2
x2
```

```
array([[ 2,  3,  4],
       [ 5,  6,  7],
       [ 8,  9, 10]])
```

```
x1 + x2                                # element-wise
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

```
x1 * x2                                # also element-wise
```

```
array([[ 0,  3,  8],
       [15, 24, 35],
       [48, 63, 80]])
```

```
np.dot(x1, x2)                          # matrix multiplication
```

```
array([[ 21,  24,  27],
       [ 66,  78,  90],
       [111, 132, 153]])
```

PANDAS

Pandas

- **It is a powerful Python library for data manipulation and analysis.**
- Data Structures:
 - DataFrame and Series for handling tabular and time series data.
- Seamlessly integrates with other libraries like NumPy, Matplotlib, and SciPy.
- Advantages of Using Pandas:
 - **Easy to Use:** Intuitive syntax for data manipulation.
 - **Flexible Data Handling:** Handles missing data, merging, reshaping, and more.
 - **Performance:** Optimized for performance with efficient data operations.
 - **Rich Functionality:** Comprehensive tools for data cleaning, transformation, and visualization.
 - **Community Support:** Extensive documentation and a large user community for support.

Data Structures

DATA STRUCTURE	DIMENSIONS	DESCRIPTION
Series	1	1D labeled homogeneous array, size-immutable.
Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3D labeled, size-mutable array (<i>deprecated</i>)

Pandas: Series

- Subclass of `numpy.ndarray`
- Data: any type
- Index labels need not be ordered
- Duplicates possible but result in reduced functionality

index		values
A	→	5
B	→	6
C	→	12
D	→	-5
E	→	6.7

pd.Series

Create

```
import pandas as pd

# From a list
s = pd.Series([1, 2, 3, 4])

# From a dictionary
s = pd.Series({'a': 1, 'b': 2, 'c': 3})

# With custom index
s = pd.Series([1, 2, 3], index=['x', 'y', 'z'])
```

Access

```
# By position
s[0]

# By label
s['a']
```

pd.Series

Basic Operations

```
# Adding a scalar  
s + 2  
  
# Adding two Series  
s1 + s2  
  
# Applying a function  
s.apply(lambda x: x**2)
```

Descriptive Statistics

```
# Summary statistics  
s.describe()  
  
# Mean  
s.mean()  
  
# Standard deviation  
s.std()
```

pd.Series

Filtering Data

```
# Boolean indexing  
s[s > 2]  
  
# Using conditions  
s[(s > 1) & (s < 3)]
```

Handling Missing Data

```
# Detect missing values  
s.isnull()  
  
# Fill missing values  
s.fillna(0)  
  
# Drop missing values  
s.dropna()
```

Pandas: DataFrame

- Each column can have a different type
- Row and Column index
- Mutable size: insert and delete columns

columns		foo	bar	baz	qux
index					
A	→	0	x	27	True
B	→	4	y	6	True
C	→	8	z	10	False
D	→	-12	w	NA	False
E	→	16	a	18	False

pd.DataFrame

Series			Series			DataFrame		
	apples			oranges			apples	oranges
0	3	+	0	0	=	0	3	0
1	2		1	3		1	2	3
2	0		2	7		2	0	7
3	1		3	2		3	1	2

DataFrame is a two-dimensional array with heterogeneous data.

pd.DataFrame

- Constructor - `pandas.DataFrame(data, index, columns, dtype, copy)`
- A DataFrame can be created using various inputs like:
 - lists
 - dict
 - series
 - Numpy ndarrays
 - Another DataFrame

Sr.No	Parameter & Description
1	data data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	index For the row labels, the Index to be used for the resulting frame is Optional Default <code>np.arange(n)</code> if no index is passed.
3	columns For column labels, the optional default syntax is - <code>np.arange(n)</code> . This is only true if no index is passed.
4	dtype Data type of each column.
5	copy This command (or whatever it is) is used for copying of data, if the default is False.

pd.DataFrame

Creating a Pandas DataFrame

```
import pandas as pd

# From a dictionary
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data)

# From a list of dictionaries
data = [{'A': 1, 'B': 4}, {'A': 2, 'B': 5}, {'A': 3, 'B': 6}]
df = pd.DataFrame(data)

# With custom index
df = pd.DataFrame(data, index=['a', 'b', 'c'])
```

Accessing Data

```
# By column
df['A']

# By row (label-based)
df.loc['a']

# By row (integer-based)
df.iloc[0]
```

pd.DataFrame

Basic Operations

```
# Adding a new column
df['C'] = df['A'] + df['B']

# Dropping a column
df.drop('C', axis=1)

# Applying a function to each element
df.applymap(lambda x: x*2)
```

Descriptive Statistics

```
# Summary statistics
df.describe()

# Mean of each column
df.mean()

# Correlation between columns
df.corr()
```

In [34]: data.describe()

Out[34]:

	Area Code	Item Code	Element Code	latitude	longitude	Y1961	Y1962	Y1963
count	21477.000000	21477.000000	21477.000000	21477.000000	21477.000000	17938.000000	17938.000000	17938.000000
mean	125.449411	2694.211529	5211.687154	20.450613	15.794445	195.262069	200.782250	205.464611
std	72.868149	148.973406	146.820079	24.628336	66.012104	1864.124336	1884.265591	1861.174711
min	1.000000	2511.000000	5142.000000	-40.900000	-172.100000	0.000000	0.000000	0.000000
25%	63.000000	2561.000000	5142.000000	6.430000	-11.780000	0.000000	0.000000	0.000000
50%	120.000000	2640.000000	5142.000000	20.590000	19.150000	1.000000	1.000000	1.000000
75%	188.000000	2782.000000	5142.000000	41.150000	46.870000	21.000000	22.000000	23.000000
max	276.000000	2961.000000	5521.000000	64.960000	179.410000	112227.000000	109130.000000	106356.000000

8 rows x 58 columns

pd.DataFrame

Filtering Data

```
# Boolean indexing
df[df['A'] > 1]

# Using conditions
df[(df['A'] > 1) & (df['B'] < 6)]
```

	continent	GDP	population
USA	Americas	19,390,604	322,179,605
China	Asia	12,237,700	1,403,500,365
Japan	Asia	4,872,137	127,748,513
Germany	Europe	3,677,439	81,914,672
UK	Europe	2,622,434	65,788,574
India	Asia	2,597,491	1,324,171,354

Handling Missing Data

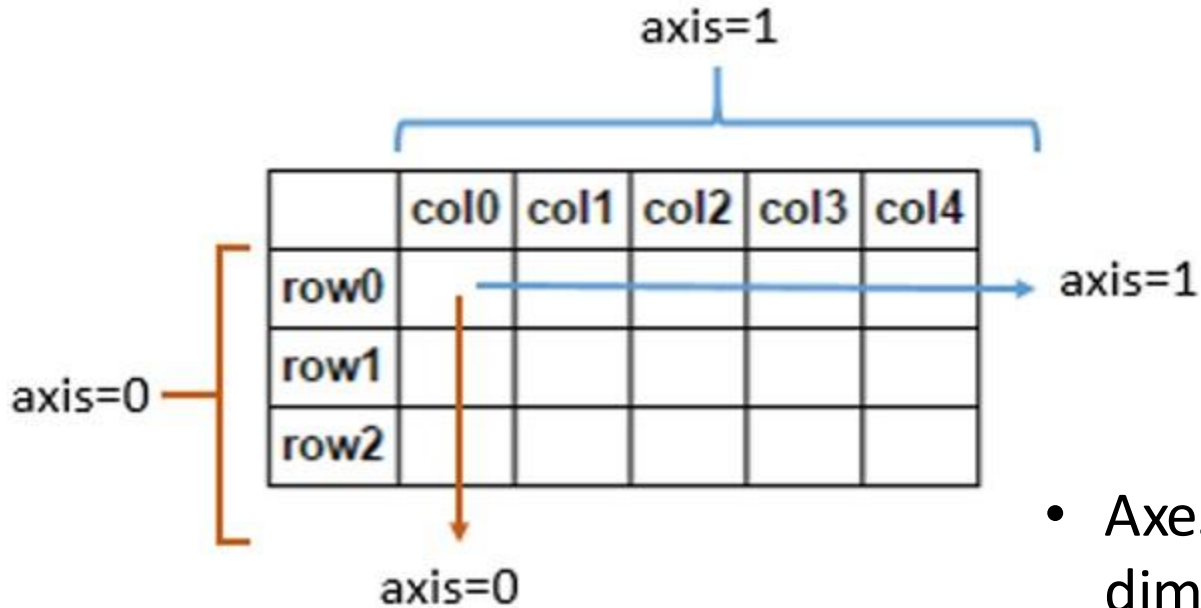
```
# Detect missing values
df.isnull()

# Fill missing values
df.fillna(0)

# Drop rows with missing values
df.dropna()
```

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	60
1	NaN	NaN	NaN	67
2	NaN	45.0	80.0	68
3	95.0	56.0	98.0	65

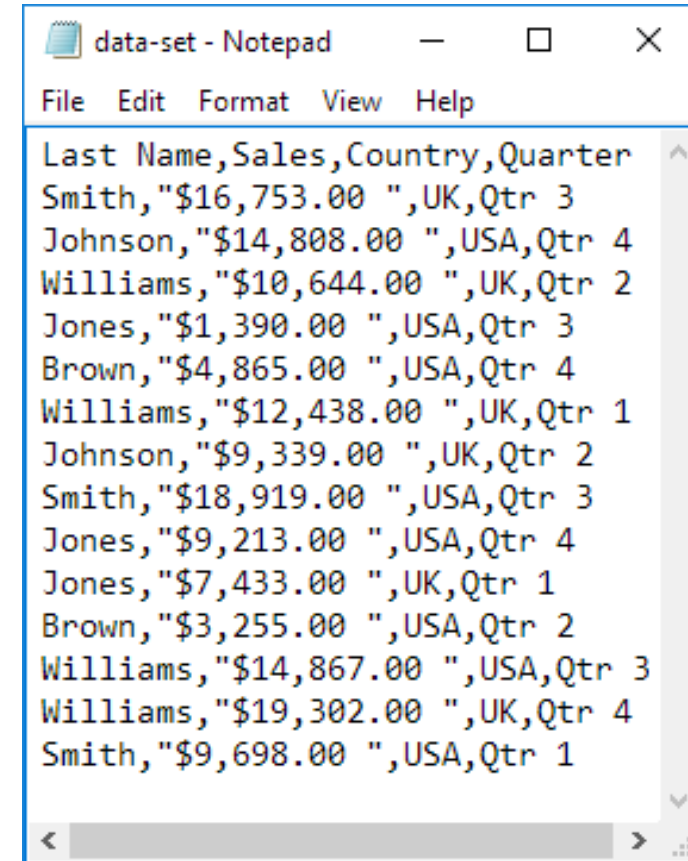
Important note regarding “axis”



- Axes are defined for arrays with more than one dimension.
- A 2-dimensional array has two corresponding axes:
 - axis 0: running vertically downwards across rows
 - axis 1: running horizontally across columns

Data reading/writing – csv files

- **Read CSV:** `pd.read_csv('file.csv')`
- **Write CSV:** `df.to_csv('file.csv', index=False)`
- **Advantages:**
 - Simple and widely supported
 - Easy to read and write
- **Use Cases:**
 - Data exchange between applications
 - Large datasets



```
data-set - Notepad
File Edit Format View Help
Last Name,Sales,Country,Quarter
Smith,"$16,753.00 ",UK,Qtr 3
Johnson,"$14,808.00 ",USA,Qtr 4
Williams,"$10,644.00 ",UK,Qtr 2
Jones,"$1,390.00 ",USA,Qtr 3
Brown,"$4,865.00 ",USA,Qtr 4
Williams,"$12,438.00 ",UK,Qtr 1
Johnson,"$9,339.00 ",UK,Qtr 2
Smith,"$18,919.00 ",USA,Qtr 3
Jones,"$9,213.00 ",USA,Qtr 4
Jones,"$7,433.00 ",UK,Qtr 1
Brown,"$3,255.00 ",USA,Qtr 2
Williams,"$14,867.00 ",USA,Qtr 3
Williams,"$19,302.00 ",UK,Qtr 4
Smith,"$9,698.00 ",USA,Qtr 1
```

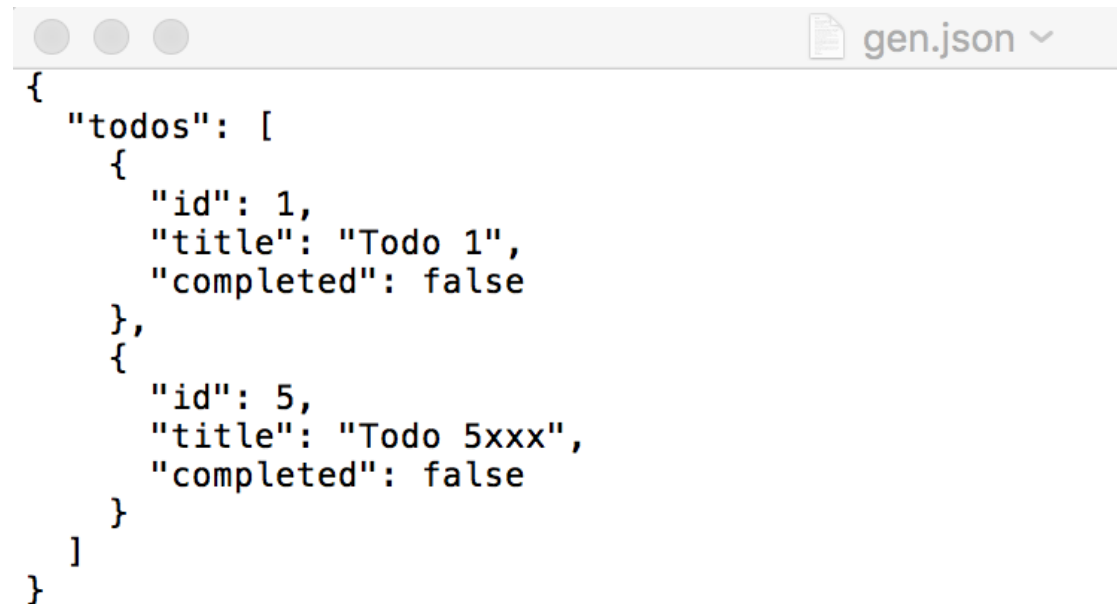
Data reading/writing – Excel files

- **Read Excel:** `pd.read_excel('file.xlsx')`
- **Write Excel:** `df.to_excel('file.xlsx', index=False)`
- **Advantages:**
 - Supports multiple sheets
 - Common in business and finance
- **Use Cases:**
 - Complex data structures
 - Data analysis and reporting

	A	B	C	D	E
1	Last Name	Sales	Country	Quarter	
2	Smith	\$16,753.00	UK	Qtr 3	
3	Johnson	\$14,808.00	USA	Qtr 4	
4	Williams	\$10,644.00	UK	Qtr 2	
5	Jones	\$1,390.00	USA	Qtr 3	
6	Brown	\$4,865.00	USA	Qtr 4	
7	Williams	\$12,438.00	UK	Qtr 1	
8	Johnson	\$9,339.00	UK	Qtr 2	
9	Smith	\$18,919.00	USA	Qtr 3	
10	Jones	\$9,213.00	USA	Qtr 4	
11	Jones	\$7,433.00	UK	Qtr 1	
12	Brown	\$3,255.00	USA	Qtr 2	
13	Williams	\$14,867.00	USA	Qtr 3	
14	Williams	\$19,302.00	UK	Qtr 4	
15	Smith	\$9,698.00	USA	Qtr 1	
16					

Data reading/writing – json

- **Read JSON:** `pd.read_json('file.json')`
- **Write JSON:** `df.to_json('file.json')`
- **Advantages:**
 - Human-readable and machine-readable
 - Nested data structures
- **Use Cases:**
 - Web data exchange
 - APIs and data serialization



```
{
  "todos": [
    {
      "id": 1,
      "title": "Todo 1",
      "completed": false
    },
    {
      "id": 5,
      "title": "Todo 5xxx",
      "completed": false
    }
  ]
}
```


Data reading/writing – html

- **Read HTML:** `df_list = pd.read_html('file.html')`
- **Write HTML:** `df.to_html('file.html')`
- **Advantages:**
 - Parses tables directly from HTML
 - Useful for web scraping
- **Use Cases:**
 - Extracting data from websites
 - Converting web tables to DataFrames

Community Courses -- Bath Autumn 1997				
Course Name	Course Tutor	Summary	Code	Fee
After the Civil War	Dr. John Wroughton	The course will examine the turbulent years in England after 1646. <i>6 weekly meetings starting Monday 13th October.</i>	H27	£32
An Introduction to Anglo-Saxon England	Mark Cottle	One day course introducing the early medieval period reconstruction the Anglo-Saxons and their society. <i>Saturday 18th October.</i>	H28	£18
The Glory that was Greece	Valerie Lorenz	Birthplace of democracy, philosophy, heartland of theater, home of argument. The Romans may have done it but the Greeks did it first. <i>Saturday day school 25th October 1997</i>	H30	£18