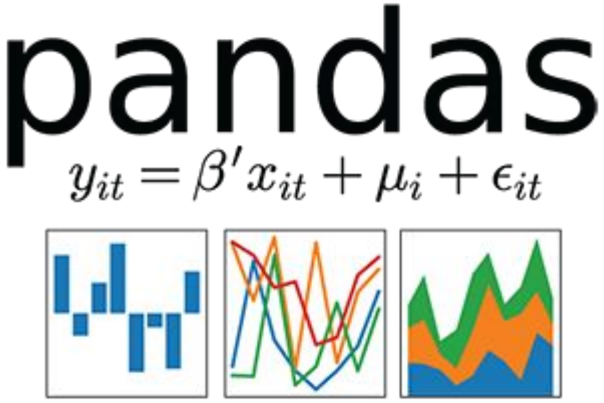
Pandas Library

Nicholas Russell Saerang

Python Data Analysis (pandas)

pandas is a software library written for the Python programming language for data manipulation and analysis.



Why Pandas?

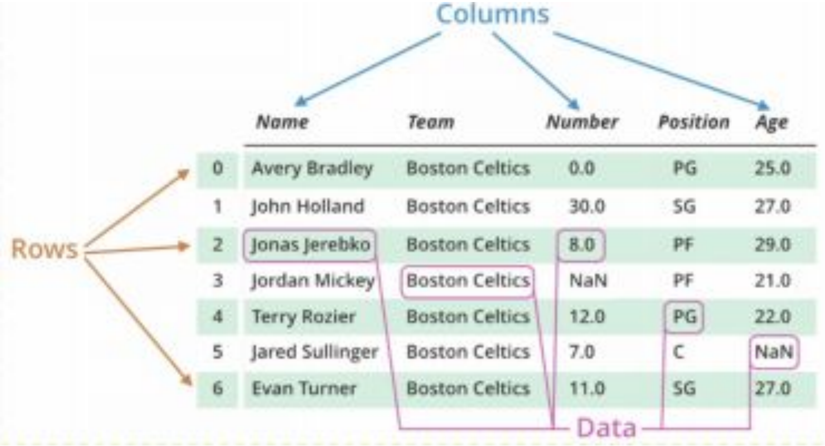
- It presents data in a way that is suitable for data analysis : Series and DataFrame
- Contains multiple easy-to-use methods
- Able to read data from different formats (JSON, CSV, XLS, XML, and many more)

```
import pandas as pd
```

DataFrame

The main data structure used in Pandas.

A two-dimensional tabular data structure with labeled axes (rows and columns)

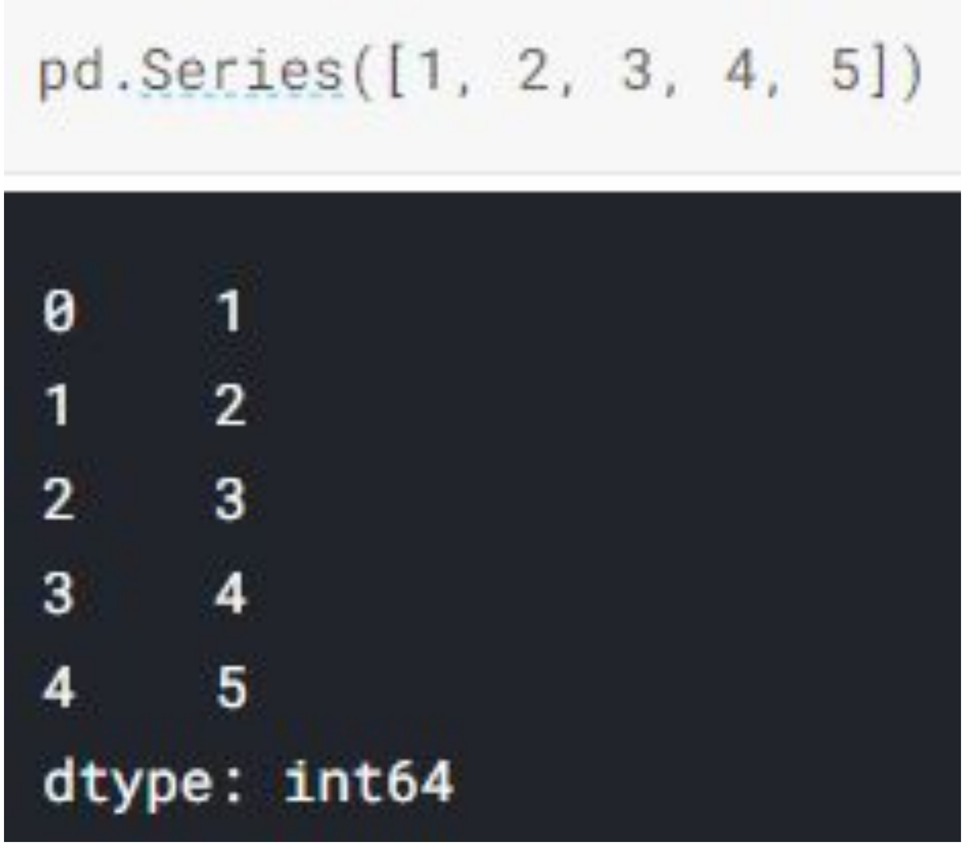


	Name	*Team*	*Number*	*Position*	*Age*
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Series

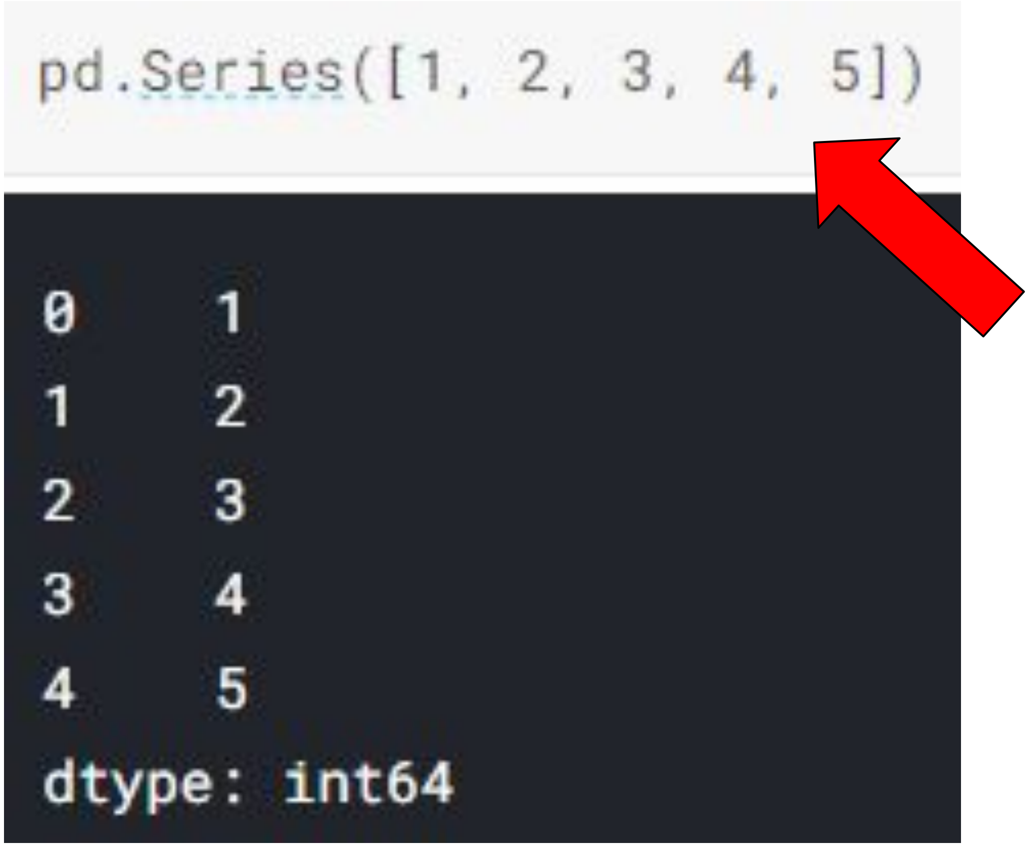
The most basic data structure used in Pandas.

A Series, by contrast, is a sequence of data values. If a DataFrame is a table, a Series is a list. And in fact you can create one with nothing more than a list.



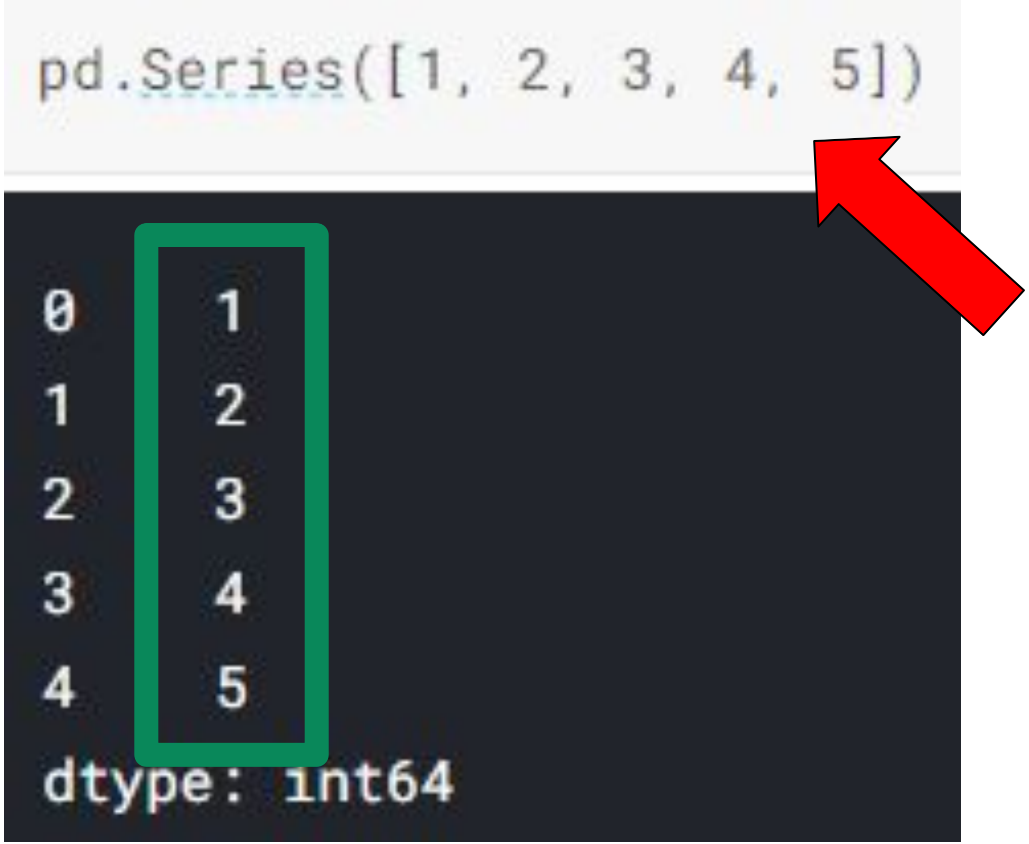
```
pd.Series([1, 2, 3, 4, 5])
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```



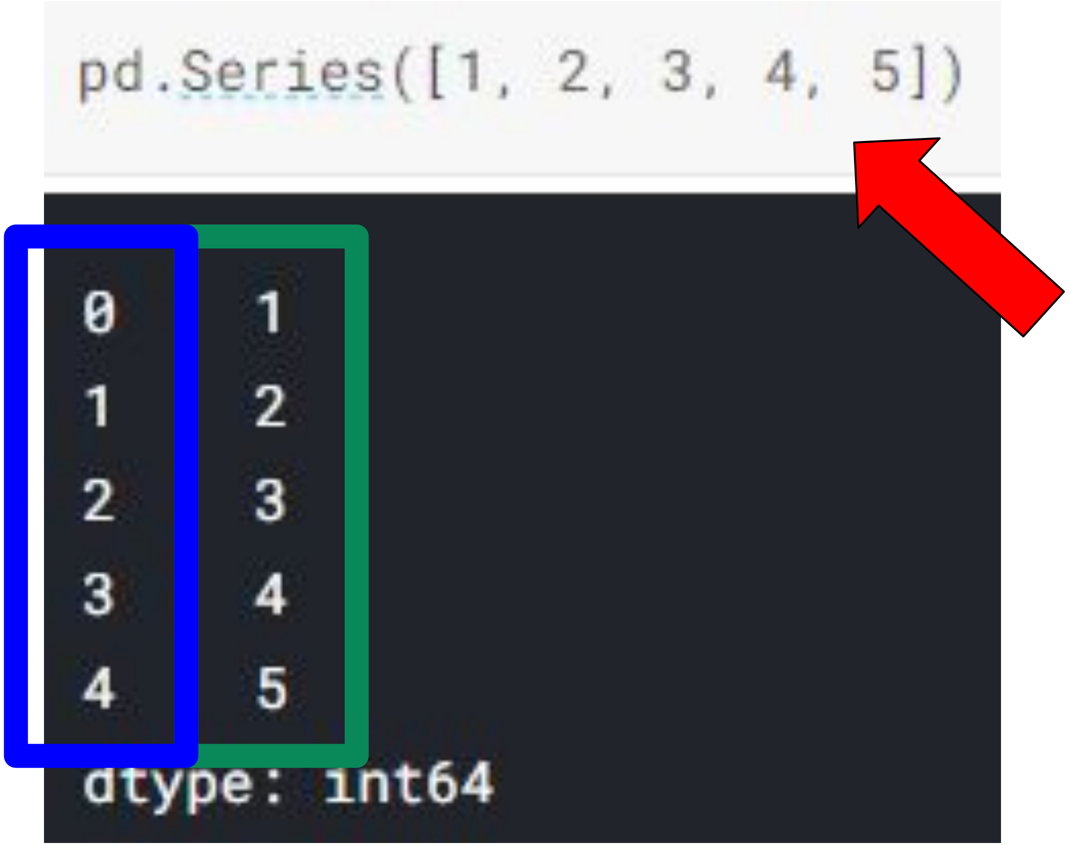
```
pd.Series([1, 2, 3, 4, 5])
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```



```
pd.Series([1, 2, 3, 4, 5])
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

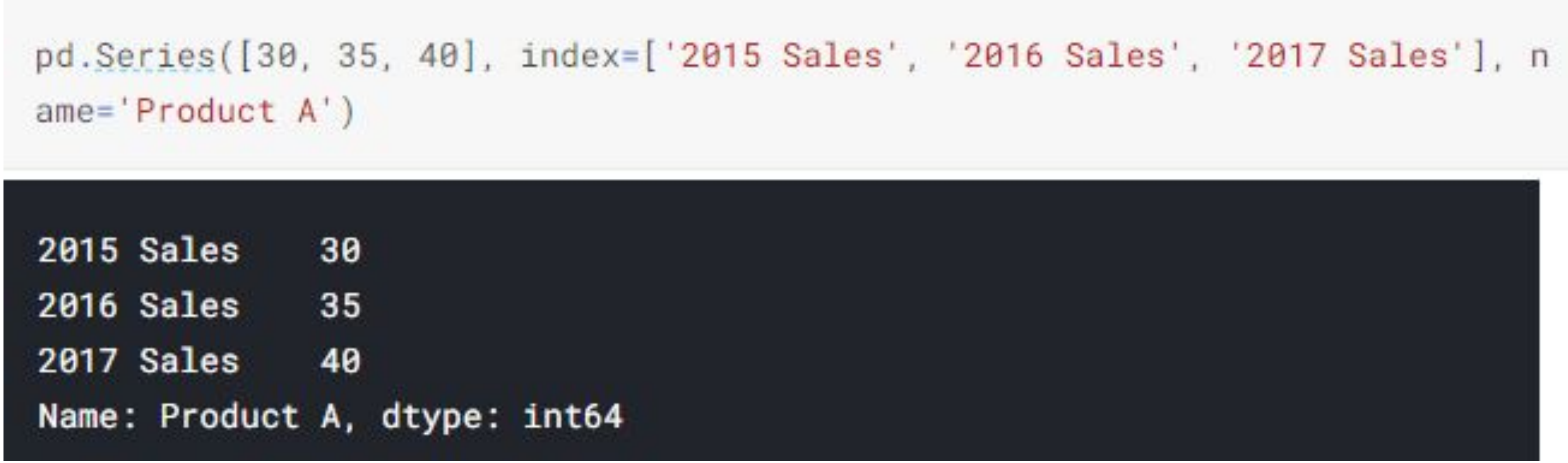


```
pd.Series([1, 2, 3, 4, 5])
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

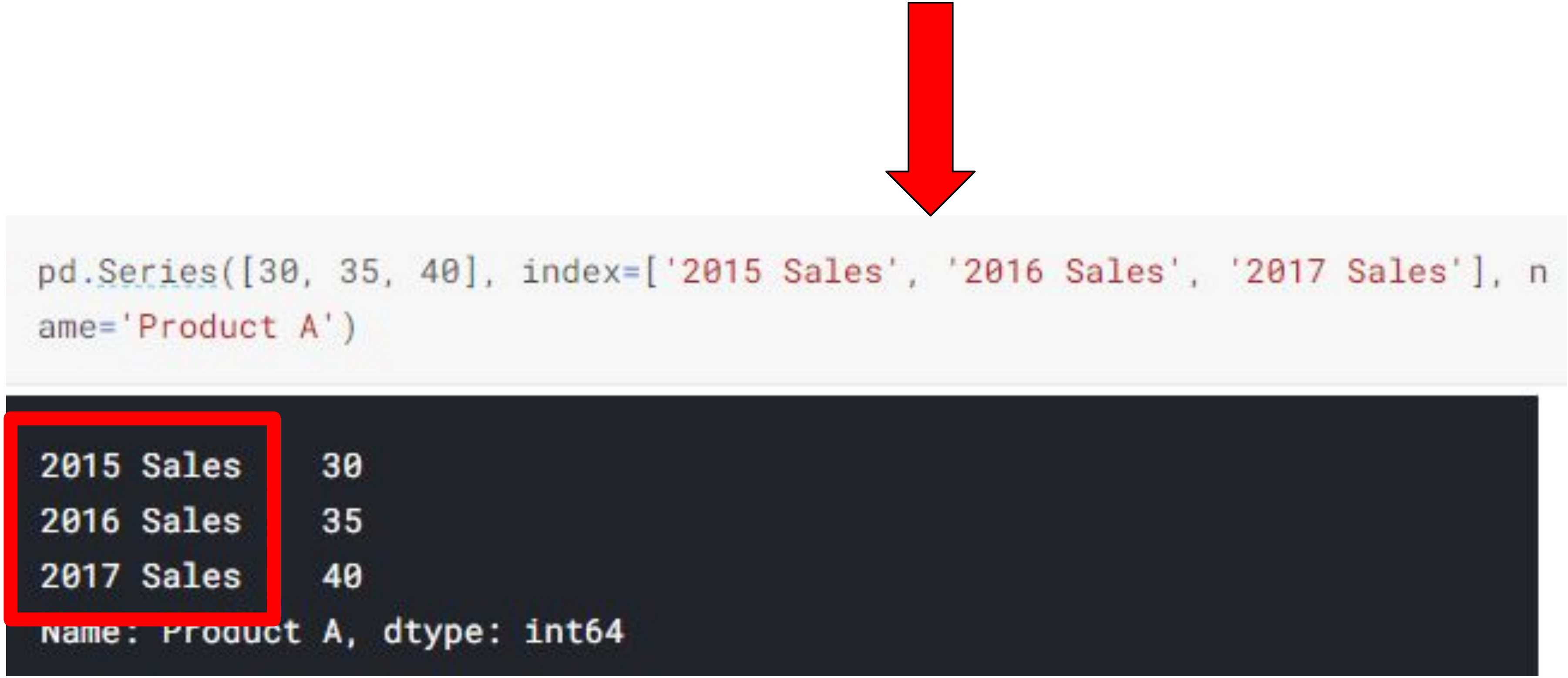
Series (ctd.)

A Series is, in essence, a single column of a DataFrame. So you can assign column values to the Series the same way as before, using an index parameter. However, a Series does not have a column name, it only has one overall **`name`**.



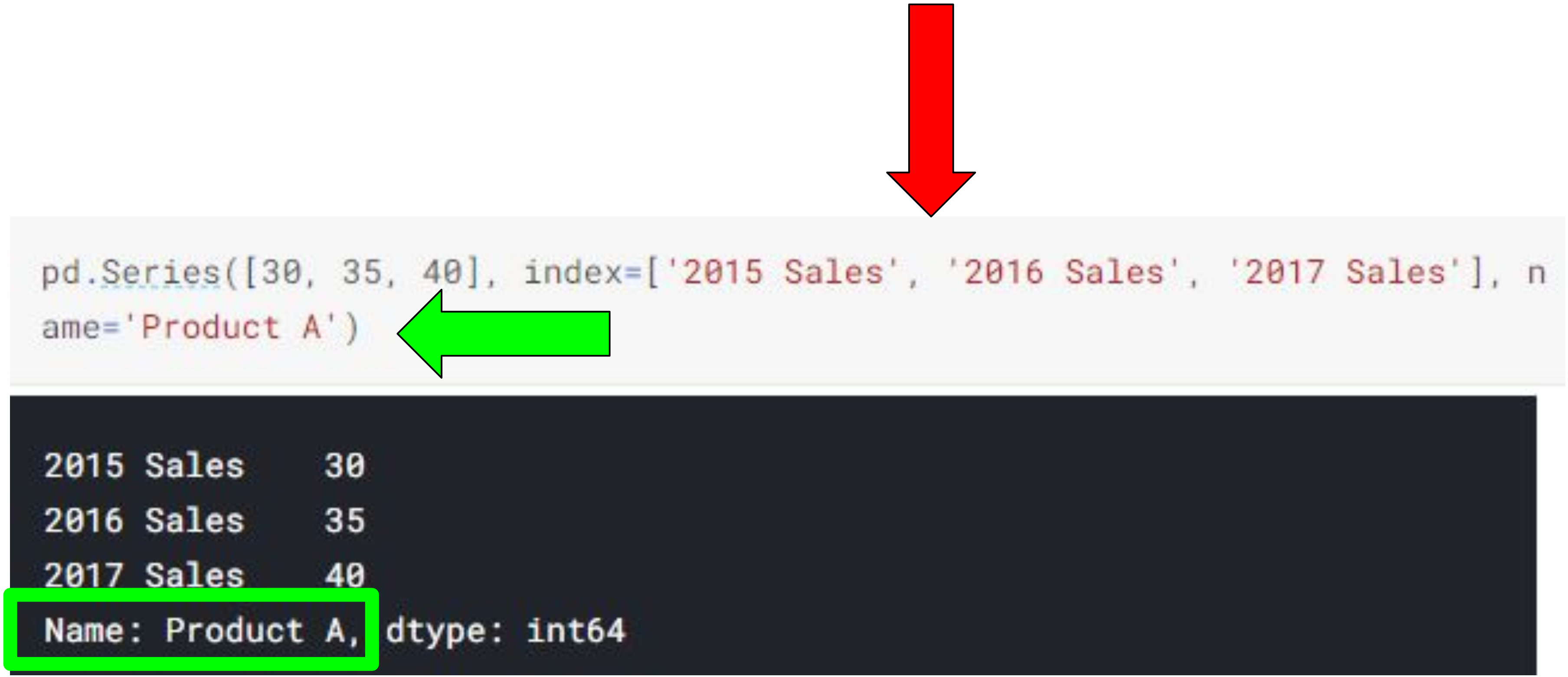
```python
pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'], name='Product A')
```

```
2015 Sales    30
2016 Sales    35
2017 Sales    40
Name: Product A, dtype: int64
```



```
pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'], n
ame='Product A')
```

```
2015 Sales    30
2016 Sales    35
2017 Sales    40
Name: Product A, dtype: int64
```

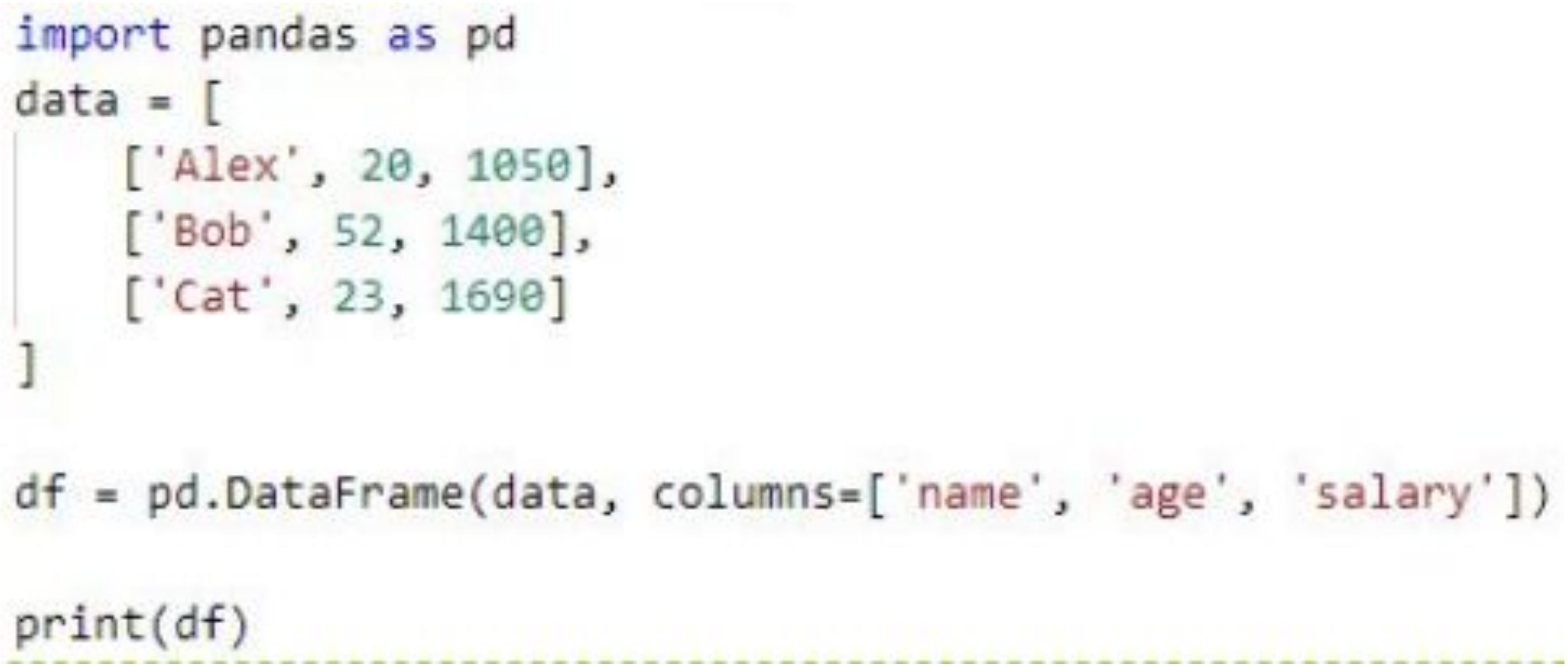


```
pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'], n
ame='Product A')
```

```
2015 Sales    30
2016 Sales    35
2017 Sales    40
Name: Product A, dtype: int64
```

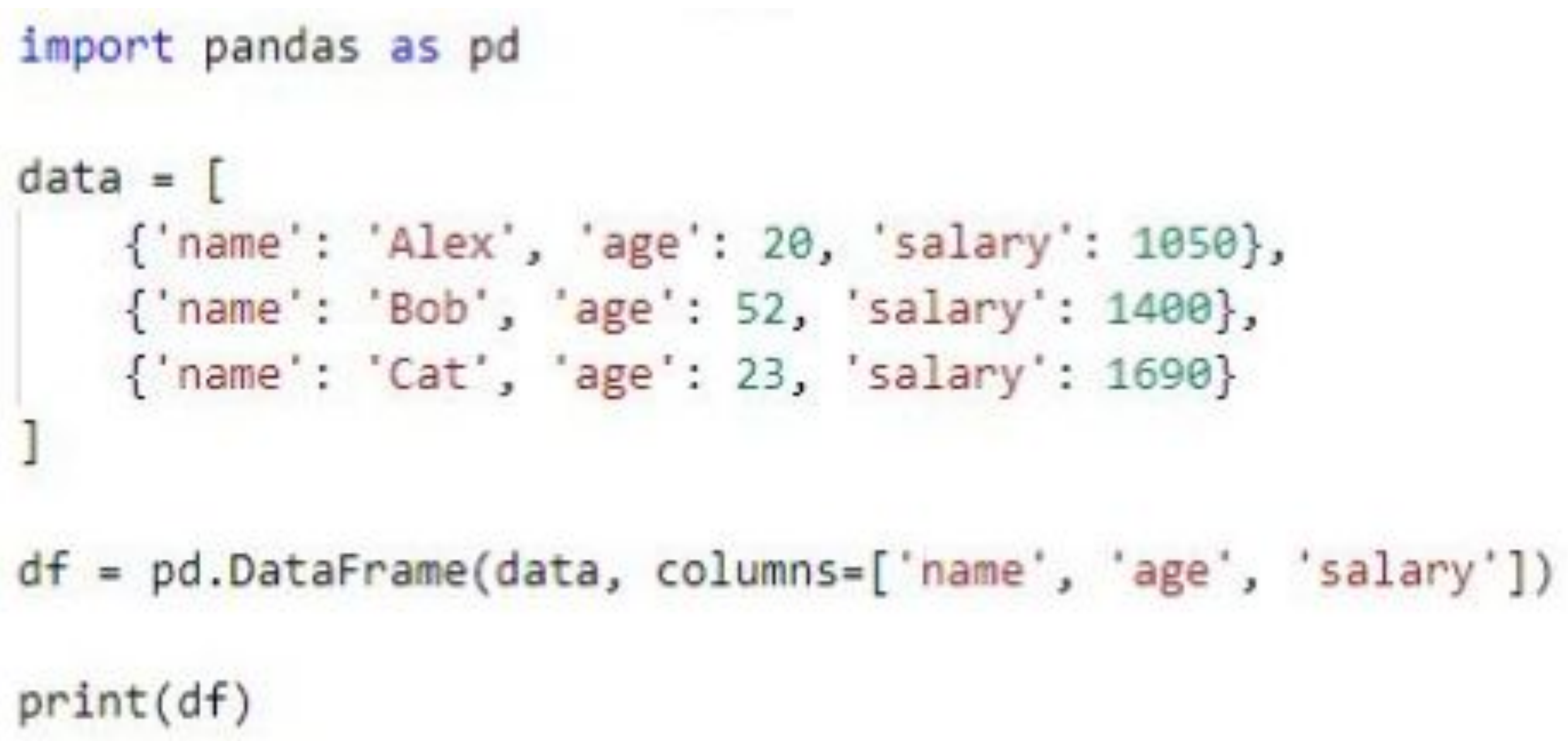
How to create a DataFrame?

- Manually, we can create a DataFrame using
  - List of lists
  - List of dictionaries
  - Dictionaries of list values
- Other way is to update an existing dataset



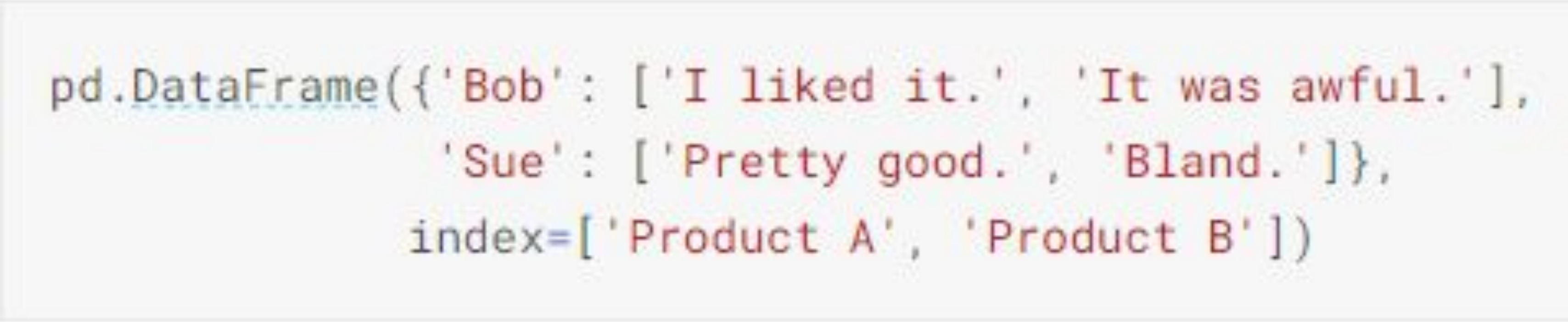
```
import pandas as pd
data = [
    ['Alex', 20, 1050],
    ['Bob', 52, 1400],
    ['Cat', 23, 1690]
]

df = pd.DataFrame(data, columns=['name', 'age', 'salary'])

print(df)
```

List of lists

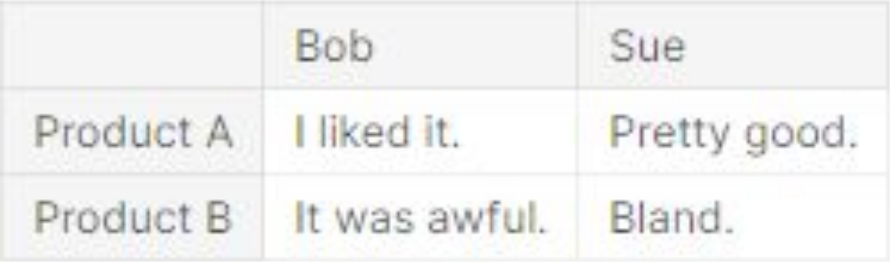


```
import pandas as pd

data = [
    {'name': 'Alex', 'age': 20, 'salary': 1050},
    {'name': 'Bob', 'age': 52, 'salary': 1400},
    {'name': 'Cat', 'age': 23, 'salary': 1690}
]

df = pd.DataFrame(data, columns=['name', 'age', 'salary'])

print(df)
```

List of dictionaries



```python
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'],
              'Sue': ['Pretty good.', 'Bland.']},
             index=['Product A', 'Product B'])
```

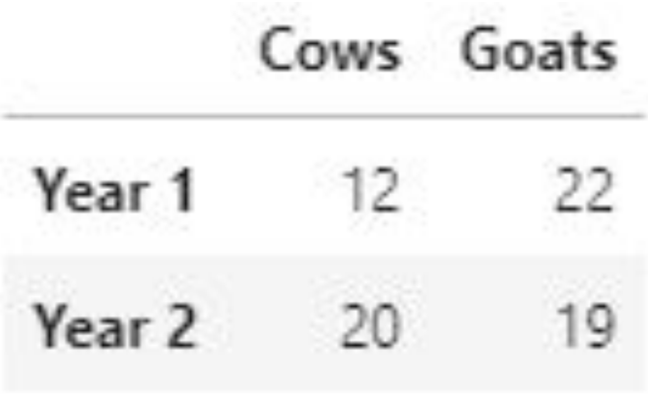


	Bob	Sue
Product A	I liked it.	Pretty good.
Product B	It was awful.	Bland.

Dictionaries of lists

Practice Part 1

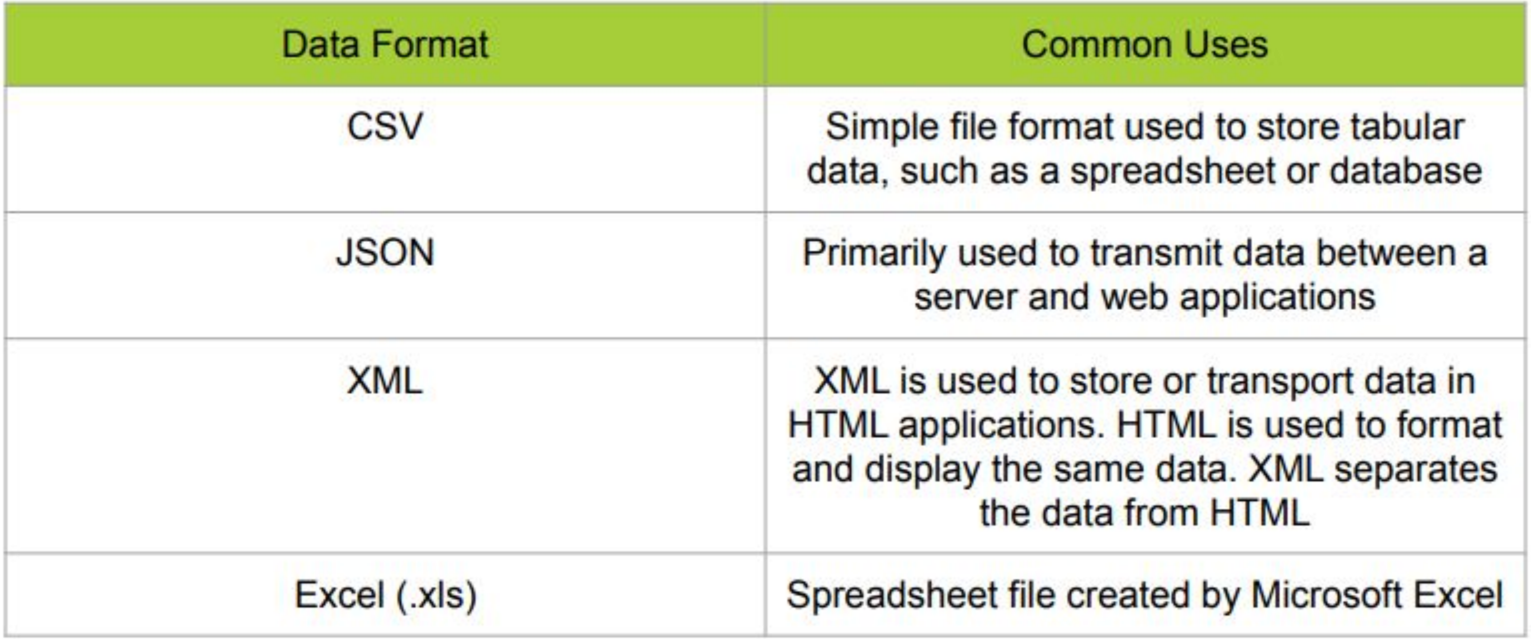
Create a Pandas DataFrame such that the resulting DataFrame is as shown.



	Cows	Goats
Year 1	12	22
Year 2	20	19

You can’t work with it if you can’t read it.

Data Formats



Data Format	Common Uses
CSV	Simple file format used to store tabular data, such as a spreadsheet or database
JSON	Primarily used to transmit data between a server and web applications
XML	XML is used to store or transport data in HTML applications. HTML is used to format and display the same data. XML separates the data from HTML
Excel (.xls)	Spreadsheet file created by Microsoft Excel

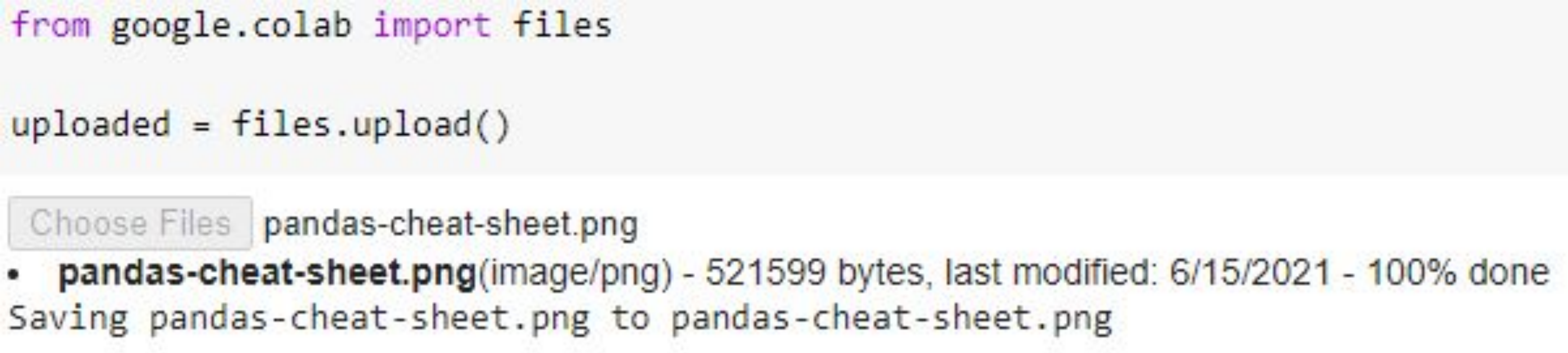
CSV File Format

- The most commonly used data format.
- Each line has a number of fields, separated by commas or some other delimiter.
- Read more at https://docs.python.org/3/library/csv.html

E.g.

```
name,age,salary

Bob,37,12000

Andy,36,13000

Carl,30,12500
```

Uploading Files to Google Colab

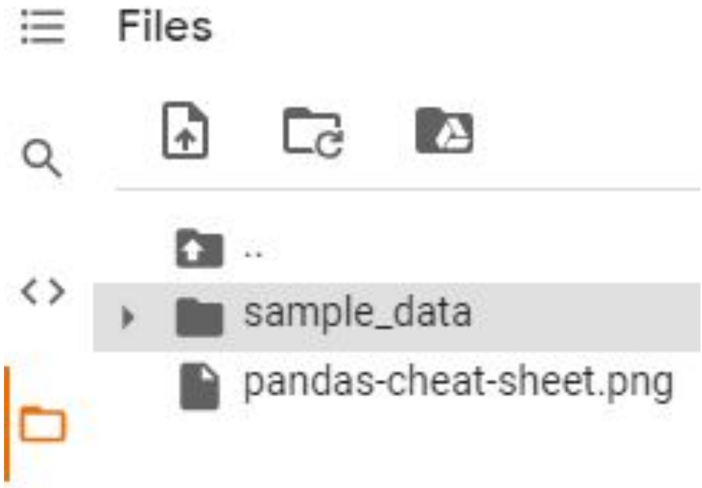


```python
from google.colab import files

uploaded = files.upload()
```

```
pandas-cheat-sheet.png
•  pandas-cheat-sheet.png(image/png) - 521599 bytes, last modified: 6/15/2021 - 100% done
Saving pandas-cheat-sheet.png to pandas-cheat-sheet.png
```

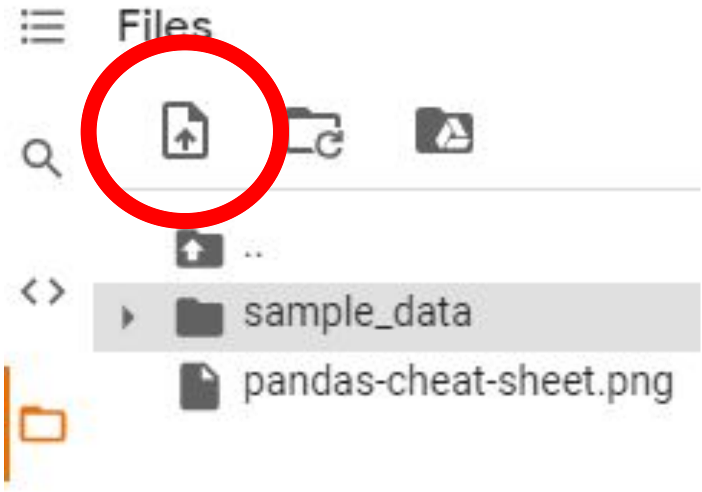
Uploading Files to Google Colab

There is also another way to upload files.



Uploading Files to Google Colab

There is also another way to upload files.

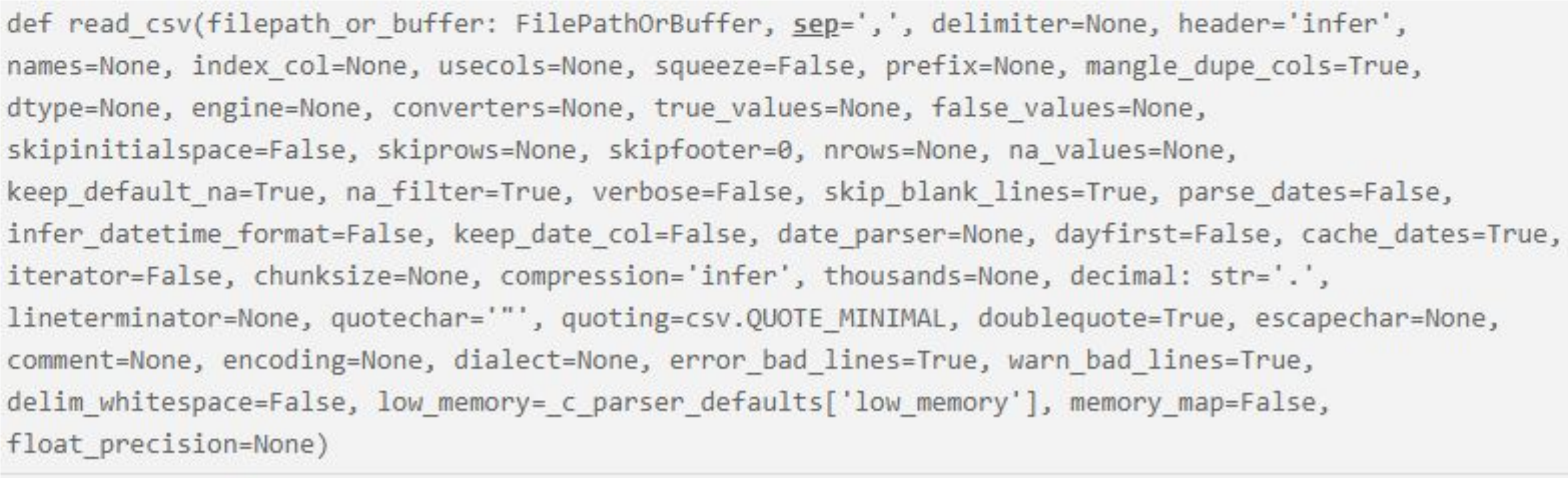


Using Pandas to Read Data Formats

```
pd.read_csv(‘data.csv’)


pd.read_excel(‘data.xlsx’, typ = ‘series’)

pd.read_excel(pd.ExcelFile(‘data.xls’), ‘Sheet1’)

pd.read_json(‘data.json’, typ = ‘series’)
```

Practice Part 2

Read the CSV file given to you before the workshop and save it to a variable cases_data.



```
def read_csv(filepath_or_buffer: FilePathOrBuffer, sep=',', delimiter=None, header='infer',
names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True,
dtype=None, engine=None, converters=None, true_values=None, false_values=None,
skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True,
iterator=False, chunksize=None, compression='infer', thousands=None, decimal: str='.',
lineterminator=None, quotechar='"', quoting=csv.QUOTE_MINIMAL, doublequote=True, escapechar=None,
comment=None, encoding=None, dialect=None, error_bad_lines=True, warn_bad_lines=True,
delim_whitespace=False, low_memory=_c_parser_defaults['low_memory'], memory_map=False,
float_precision=None)
```

Summary of Part 1 and 2

Data collection! We have learned how to read data into Pandas, with our basic understanding of Pandas data structure which we will work on later.

```
name,age,salary
Bob,37,12000
Andy,36,13000
Carl,30,12500
```

Case 1: Normal

```
Bob,37,12000
Andy,36,13000
Carl,30,12500
```

**Case 2:
Missing Headers**

```
id,name,age,salary
1,Bob,37,12000
2,Andy,36,13000
3,Carl,30,12500
```

**Case 3:
Extra Columns**

```
name,age,salary
Bob,37,12000
Andy,36,13000
Carl,30,12500
```

Case 1: Normal

names

```
Bob,37,12000
Andy,36,13000
Carl,30,12500
```

Case 2:
Missing Headers

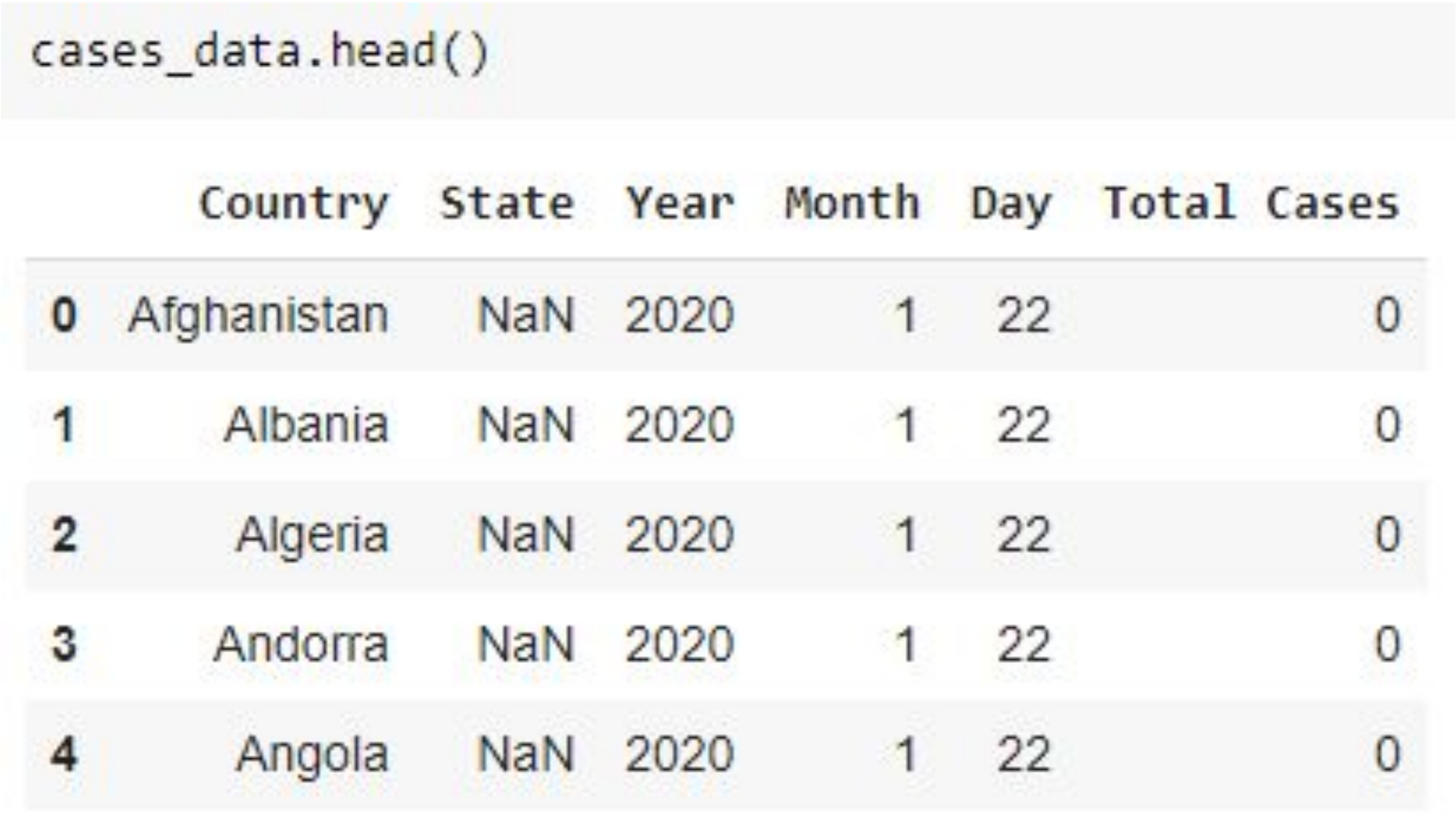
index_col

```
id,name,age,salary
1,Bob,37,12000
2,Andy,36,13000
3,Carl,30,12500
```

Case 3:
Extra Columns

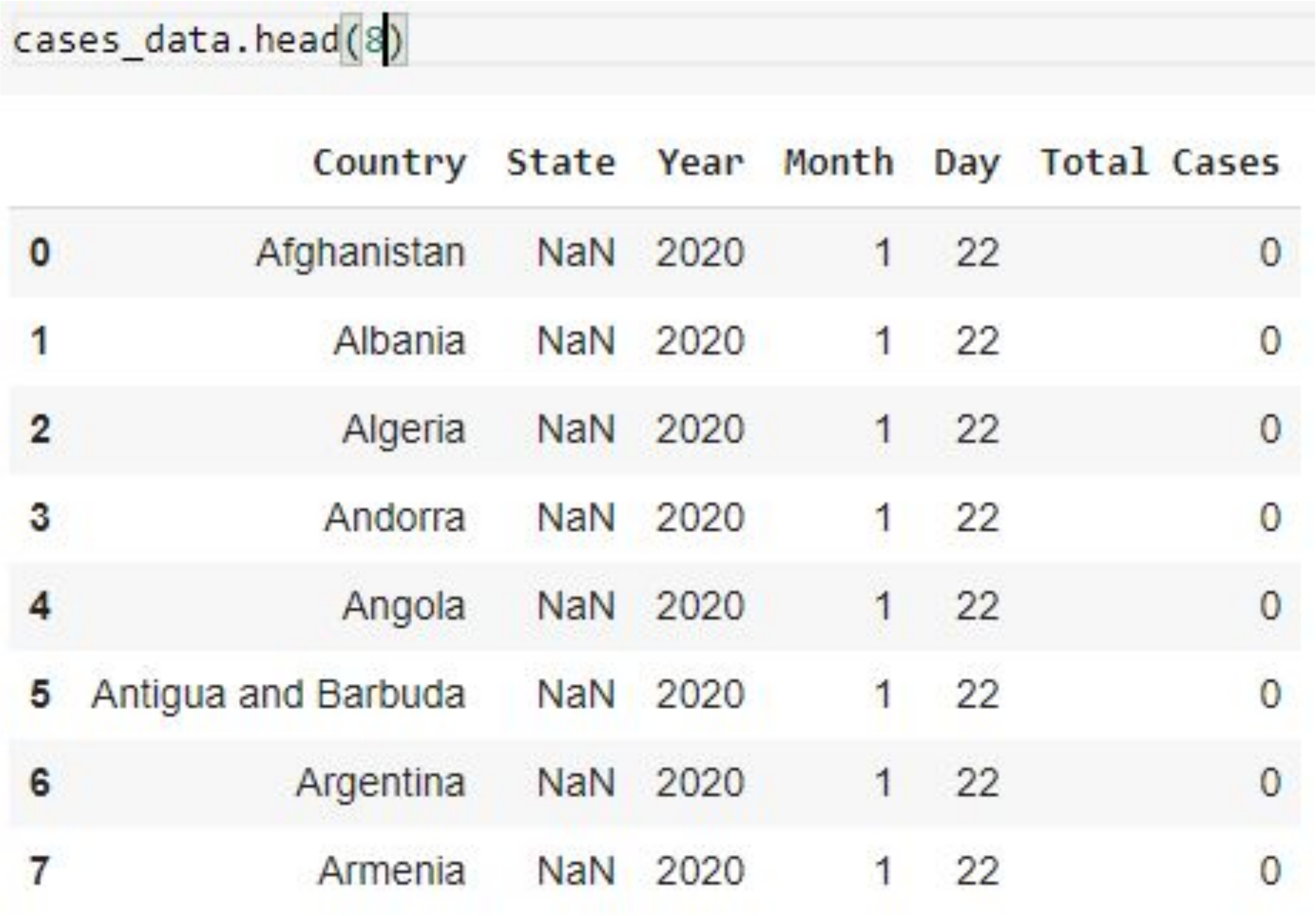
Extracting Column or Row

- head(n), extract the first n rows of the data (default = 5)
- tail(n), extract the last n rows of the data (default = 5)
- iloc[r(, c)], index-based extraction, meaning c is a list of integers or an integer
- loc[r(, c)], label-based extraction, meaning c is a list of column names as strings or a string
- iat[r(, c)], basically the same as iloc.
- at[r(, c)], basically the same as loc.
- Direct slicing, example: data[2:]
- Direct accessing, example: data[‘country’][1] or data.country



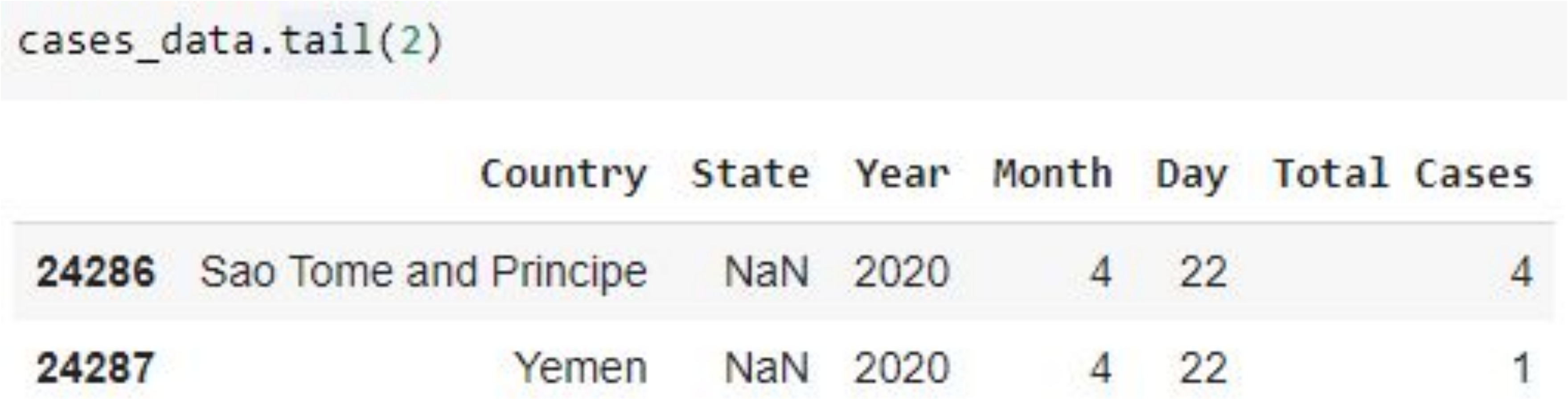
```
cases_data.head()
```

	Country	State	Year	Month	Day	Total Cases
0	Afghanistan	NaN	2020	1	22	0
1	Albania	NaN	2020	1	22	0
2	Algeria	NaN	2020	1	22	0
3	Andorra	NaN	2020	1	22	0
4	Angola	NaN	2020	1	22	0



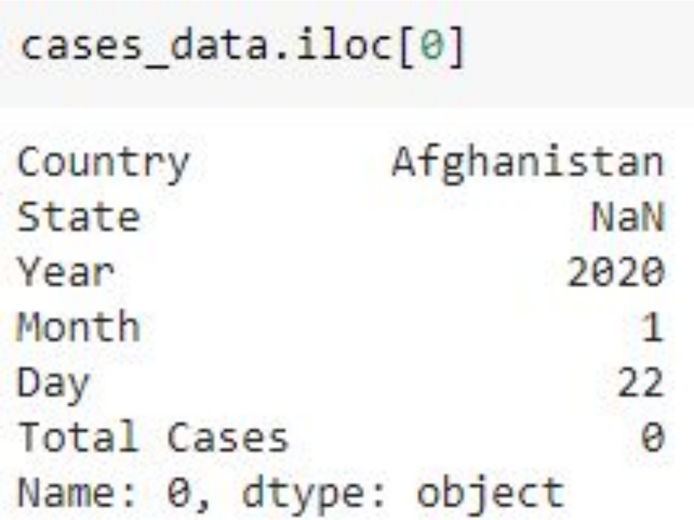
```
cases_data.head(8)
```

	Country	State	Year	Month	Day	Total Cases
0	Afghanistan	NaN	2020	1	22	0
1	Albania	NaN	2020	1	22	0
2	Algeria	NaN	2020	1	22	0
3	Andorra	NaN	2020	1	22	0
4	Angola	NaN	2020	1	22	0
5	Antigua and Barbuda	NaN	2020	1	22	0
6	Argentina	NaN	2020	1	22	0
7	Armenia	NaN	2020	1	22	0



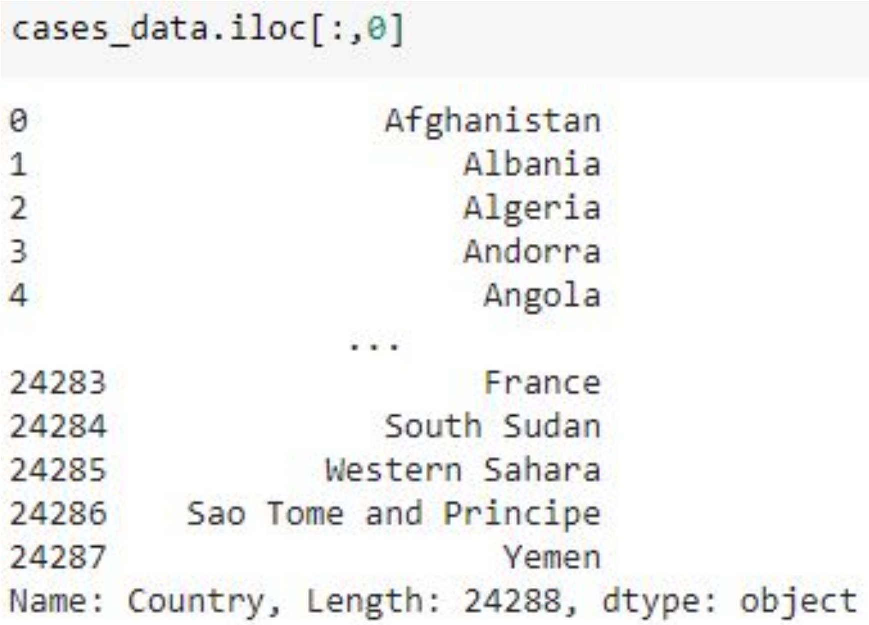
```python
cases_data.tail(2)
```

```
                        Country  State  Year  Month  Day  Total Cases
24286  Sao Tome and Principe    NaN  2020      4   22            4
24287                  Yemen    NaN  2020      4   22            1
```



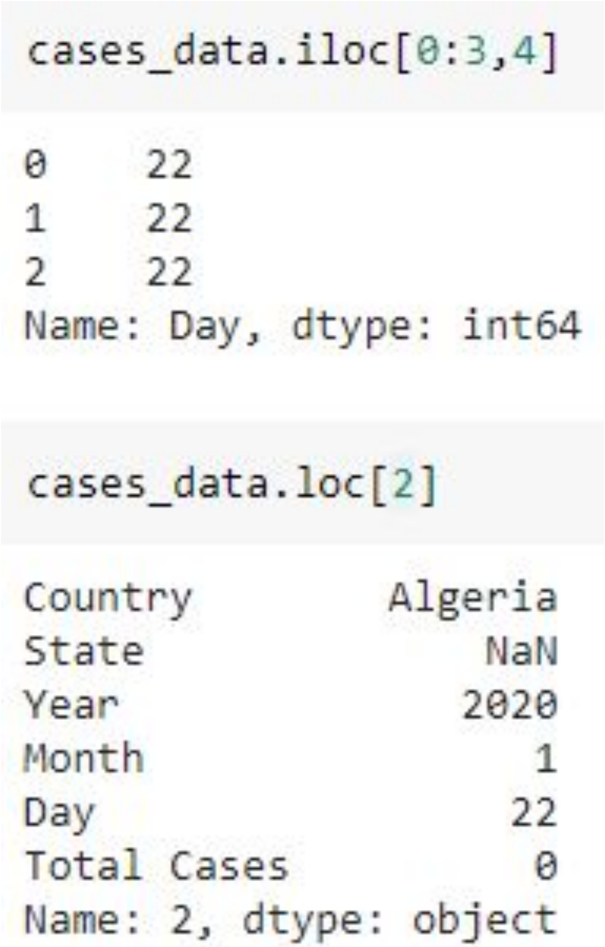
```
cases_data.iloc[0]
```

```
Country       Afghanistan
State                 NaN
Year                 2020
Month                   1
Day                    22
Total Cases             0
Name: 0, dtype: object
```



```
cases_data.iloc[:,0]
```

```
0                       Afghanistan
1                           Albania
2                           Algeria
3                           Andorra
4                            Angola
                    ...
24283                        France
24284                   South Sudan
24285                Western Sahara
24286         Sao Tome and Principe
24287                         Yemen
Name: Country, Length: 24288, dtype: object
```

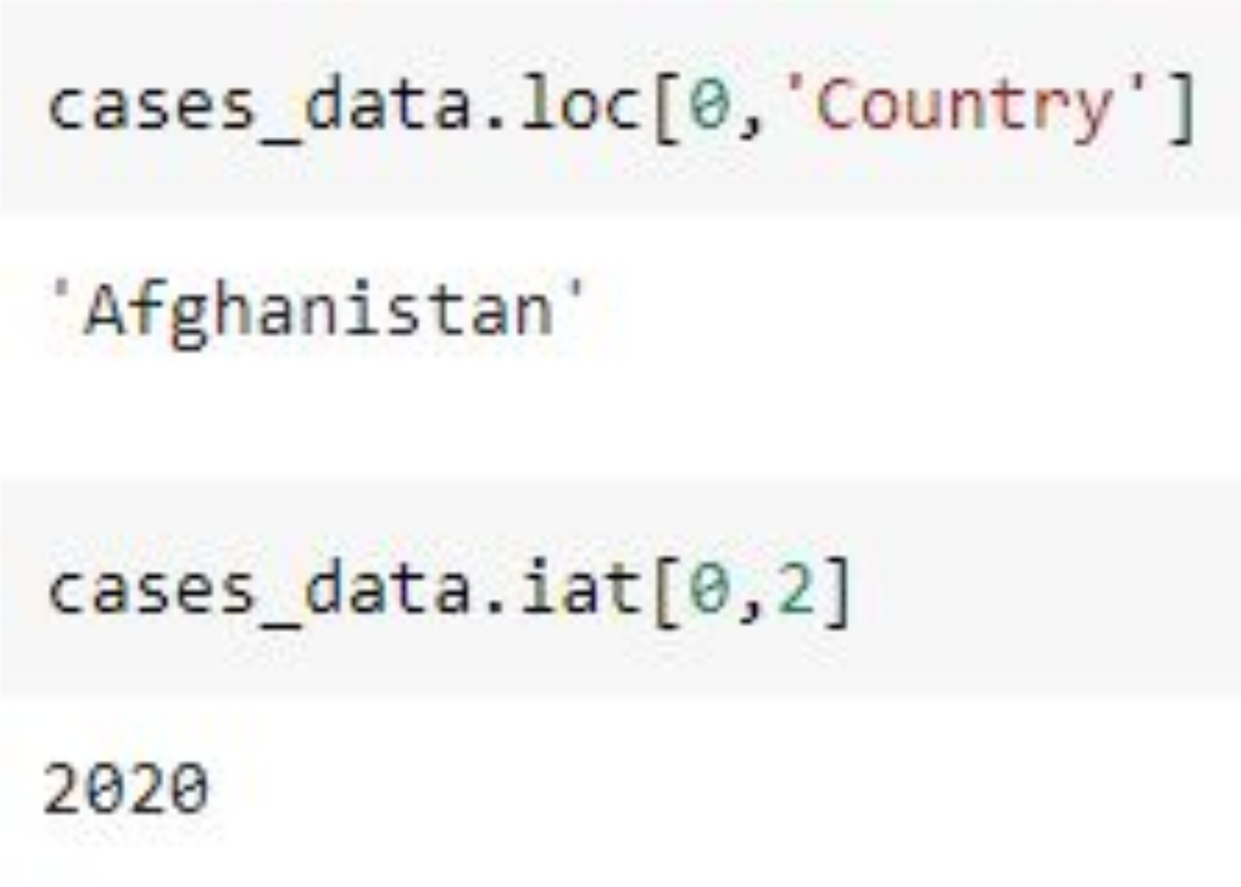


```
cases_data.iloc[0:3,4]
```

```
0    22
1    22
2    22
Name: Day, dtype: int64
```

```
cases_data.loc[2]
```

```
Country        Algeria
State              NaN
Year              2020
Month                1
Day                 22
Total Cases          0
Name: 2, dtype: object
```

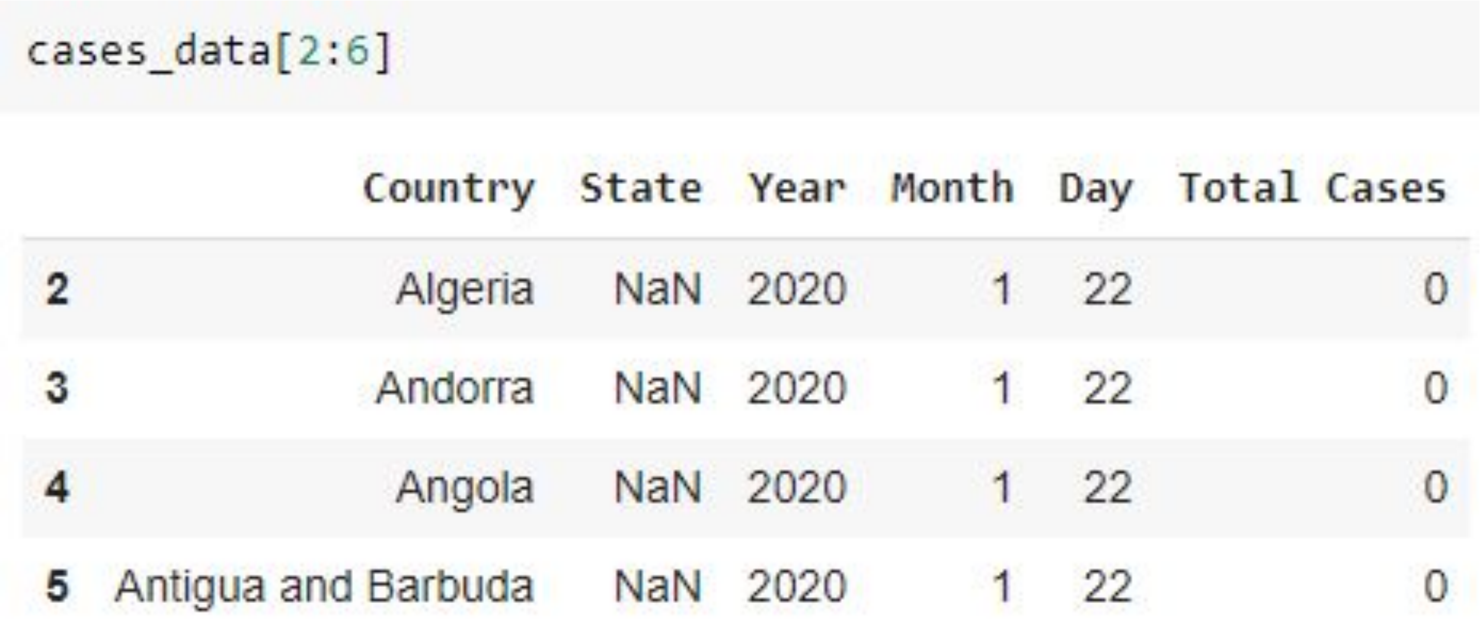


```
cases_data.loc[0,'Country']
```

```
'Afghanistan'
```

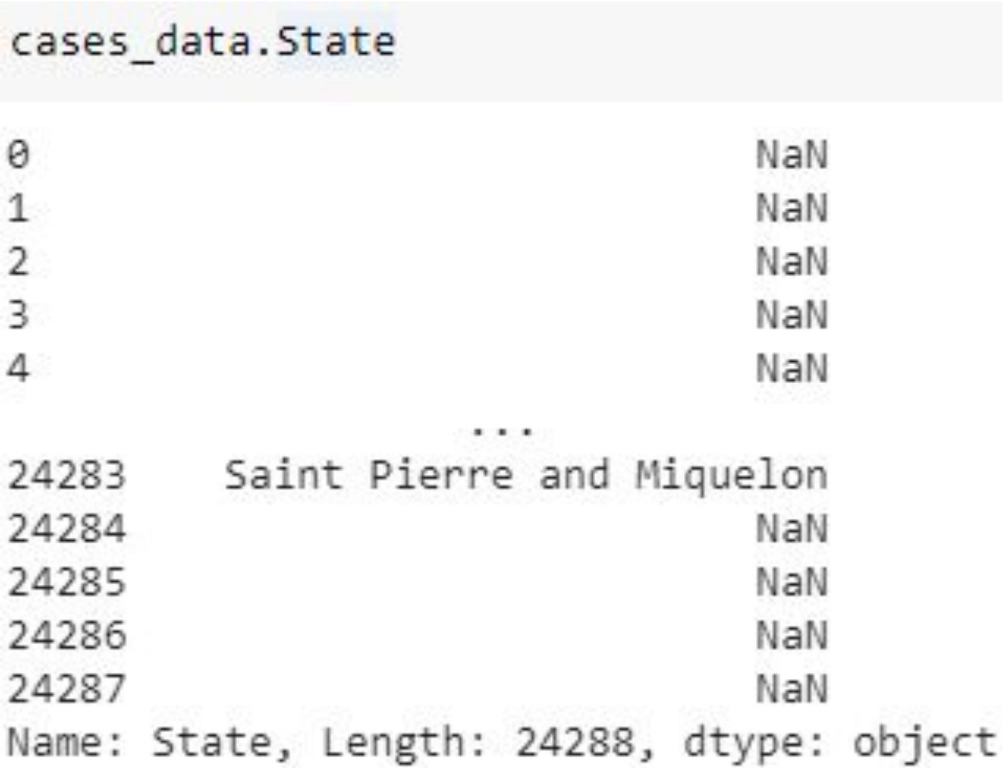
```
cases_data.iat[0,2]
```

```
2020
```



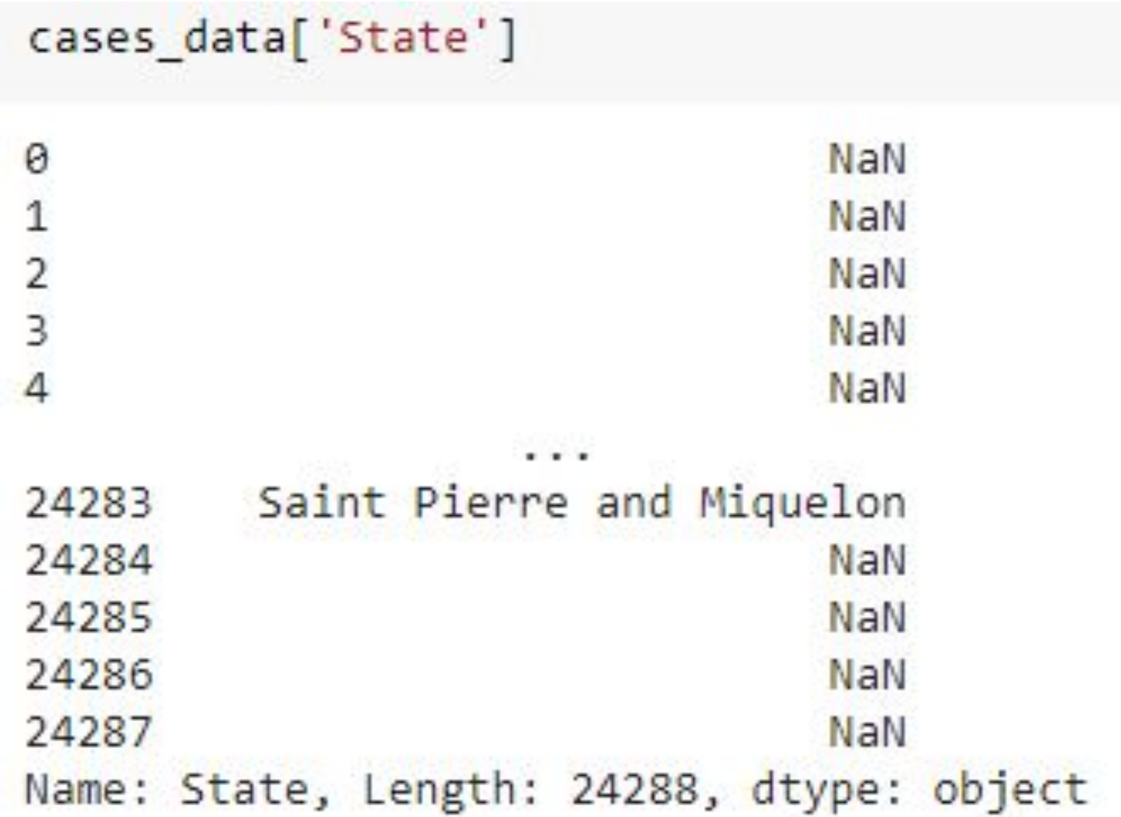
```
cases_data[2:6]
```

	Country	State	Year	Month	Day	Total Cases
2	Algeria	NaN	2020	1	22	0
3	Andorra	NaN	2020	1	22	0
4	Angola	NaN	2020	1	22	0
5	Antigua and Barbuda	NaN	2020	1	22	0



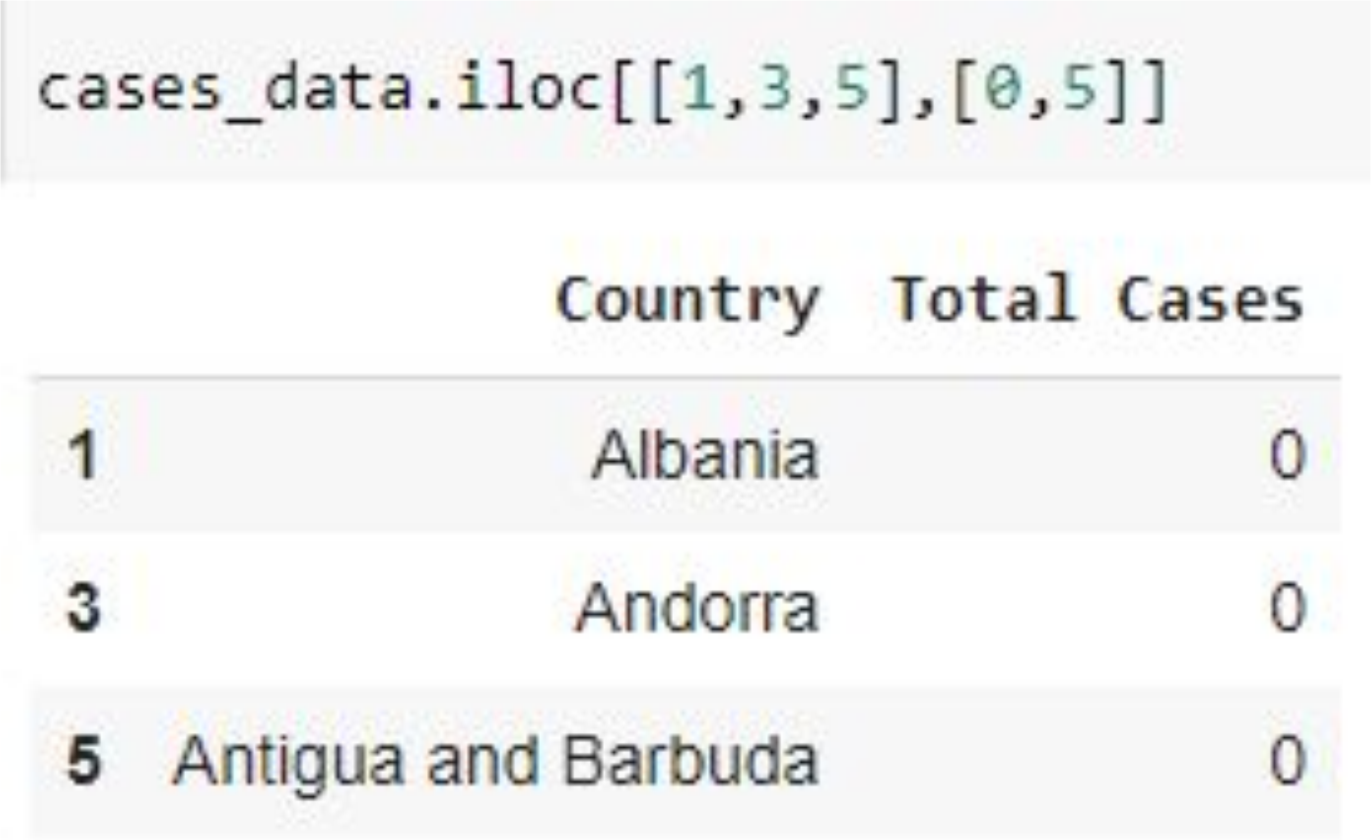
```
cases_data.State
```

```
0                              NaN
1                              NaN
2                              NaN
3                              NaN
4                              NaN
                   ...
24283    Saint Pierre and Miquelon
24284                          NaN
24285                          NaN
24286                          NaN
24287                          NaN
Name: State, Length: 24288, dtype: object
```



```
cases_data['State']
```

```
0                              NaN
1                              NaN
2                              NaN
3                              NaN
4                              NaN
                   ...
24283    Saint Pierre and Miquelon
24284                          NaN
24285                          NaN
24286                          NaN
24287                          NaN
Name: State, Length: 24288, dtype: object
```



```
cases_data.iloc[[1,3,5],[0,5]]
```

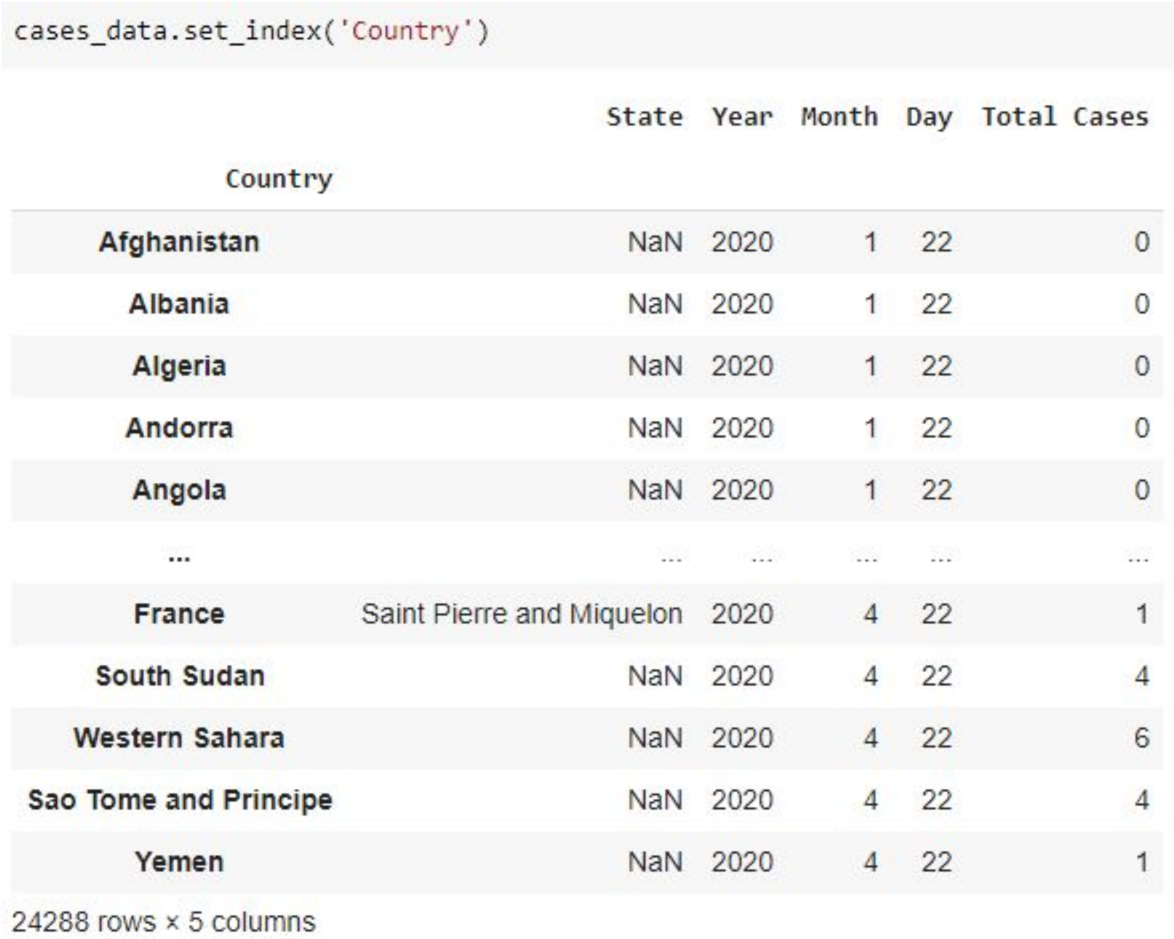
	Country	Total Cases
1	Albania	0
3	Andorra	0
5	Antigua and Barbuda	0

Changing Index

We can change the index of the dataframe with another column.

Try running this:

```
cases_data.set_index(‘Country’)
```



```
cases_data.set_index('Country')
```

	State	Year	Month	Day	Total Cases
Country					
Afghanistan	NaN	2020	1	22	0
Albania	NaN	2020	1	22	0
Algeria	NaN	2020	1	22	0
Andorra	NaN	2020	1	22	0
Angola	NaN	2020	1	22	0
...	...	...	...	...	...
France	Saint Pierre and Miquelon	2020	4	22	1
South Sudan	NaN	2020	4	22	4
Western Sahara	NaN	2020	4	22	6
Sao Tome and Principe	NaN	2020	4	22	4
Yemen	NaN	2020	4	22	1

24288 rows × 5 columns

Adding a Column/Row

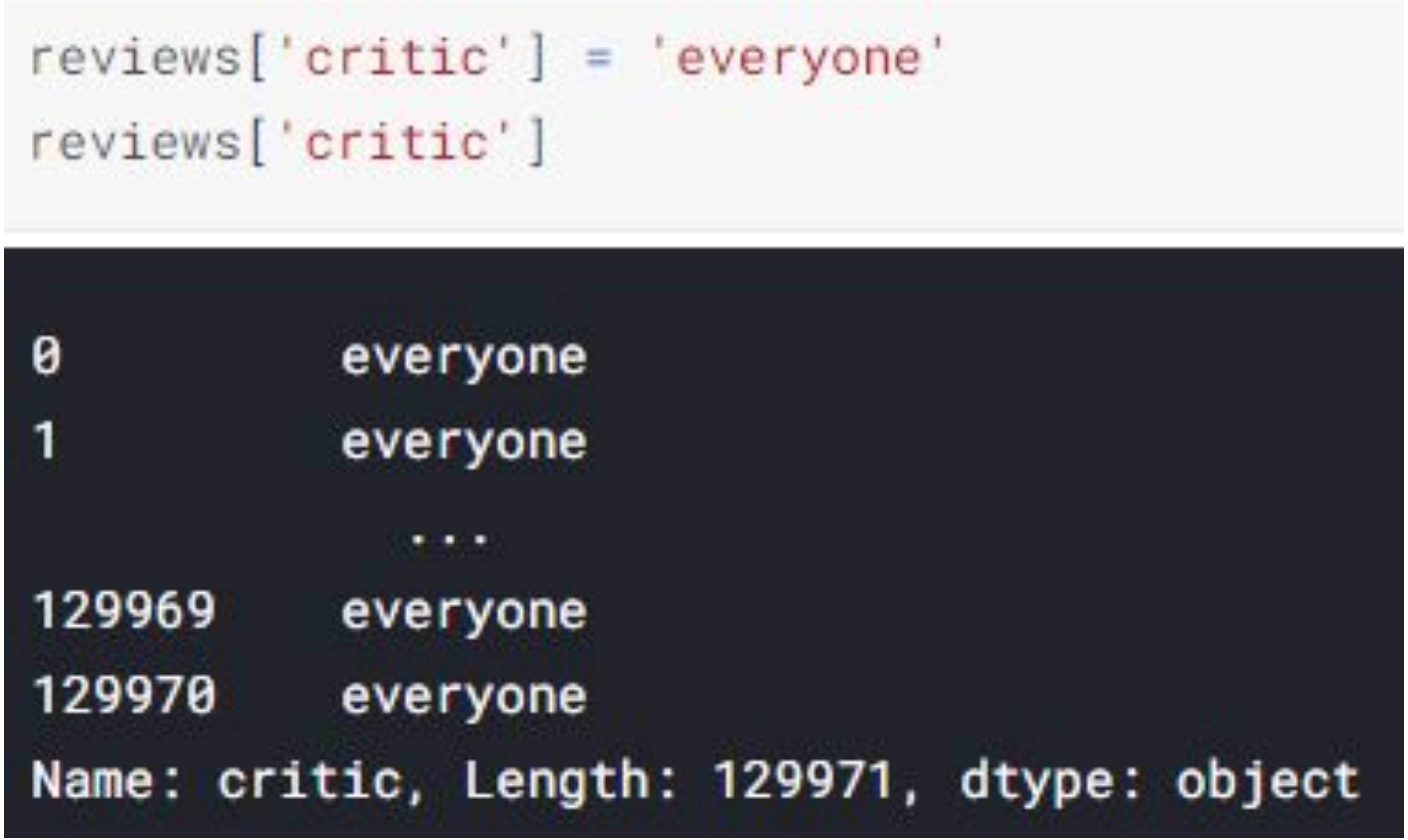
There are some ways to add a new data to an existing DataFrame.

- Adding a column:
  - `normal_data = pd.read_csv(“normal.csv”)`
  - `normal_data[‘is_married’] = [True, True, False]`

Adding a Column/Row

There are some ways to add a new data to an existing DataFrame.

- Assigning a new column:
  - `normal_data = pd.read_csv(“normal.csv”)`
  - `normal_data[‘is_married’] = [True, True, False]`
- Adding a row:
  - Use the append function! Basically adding another dataframe below the existing.
  - `normal_data.append(pd.DataFrame({'name':['Donny','Emz'], 'age':[10, 20], 'salary': [0,1000], 'is_married': [False, False]}), ignore_index = True)`



```python
reviews['critic'] = 'everyone'
reviews['critic']
```

```
0         everyone
1         everyone
            ...
129969    everyone
129970    everyone
Name: critic, Length: 129971, dtype: object
```

Adding a Column/Row

Besides append, we can also use concat, which can handle multiple DataFrames.

```
another_data = pd.read_csv(“another_data.csv”)

joined_data = pd.concat([normal_data,
another_data], ignore_index = True)
```

```
# What happens when you don’t use ignore_index?
```

Adding a Column/Row

Pandas also have functions such as merge() and join(), which can be further read at

https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

Practice Part 3

None! Take a break for 5 minutes!

Renaming Columns and Rows

Pandas provides you a rename function that can work on both columns and rows!

- Renaming columns

```
data.rename(columns = {‘old_name’: ‘new_name’,
‘old_name2’: ‘new_name2’})
```

- Renaming rows

```
data_rename(index = {0: ‘First’, 1: ‘Second’})
```

Changing a Column/Row

There are some ways to change a new data to an existing DataFrame.

- Reassigning a new row:
    - `joined_data.iloc[5] = [‘Zara’,27,3000,False]`

Practice Part 4

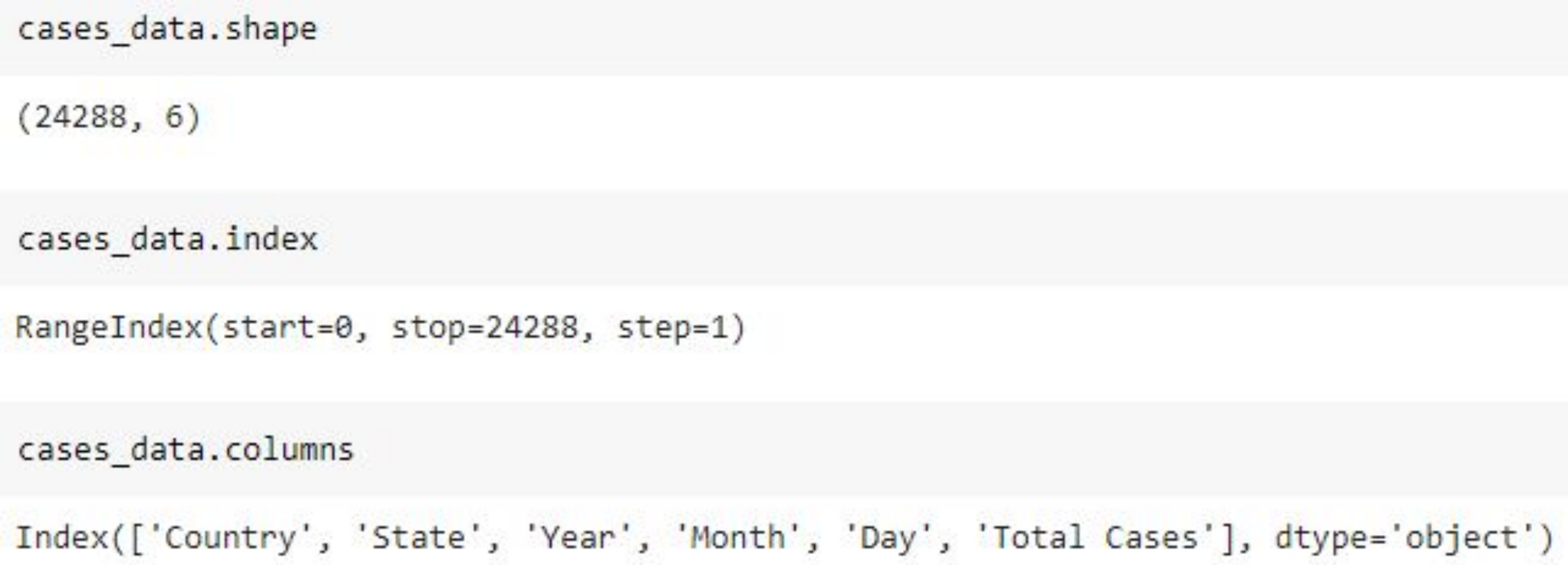
After doing all the appends and concatenations, we have `joined_data`. Rename the salary column into “wage” and the first row into “Boss”.

Now add one more person entitled ‘Secretary’, named Karen, 37 years old, has $13500 salary, and is not married yet.

Summary Functions

Often, when we are given a certain dataset, we want the big picture of it. This is where the summary functions come into action.

Some Syntaxes



```
cases_data.shape
```

```
(24288, 6)
```

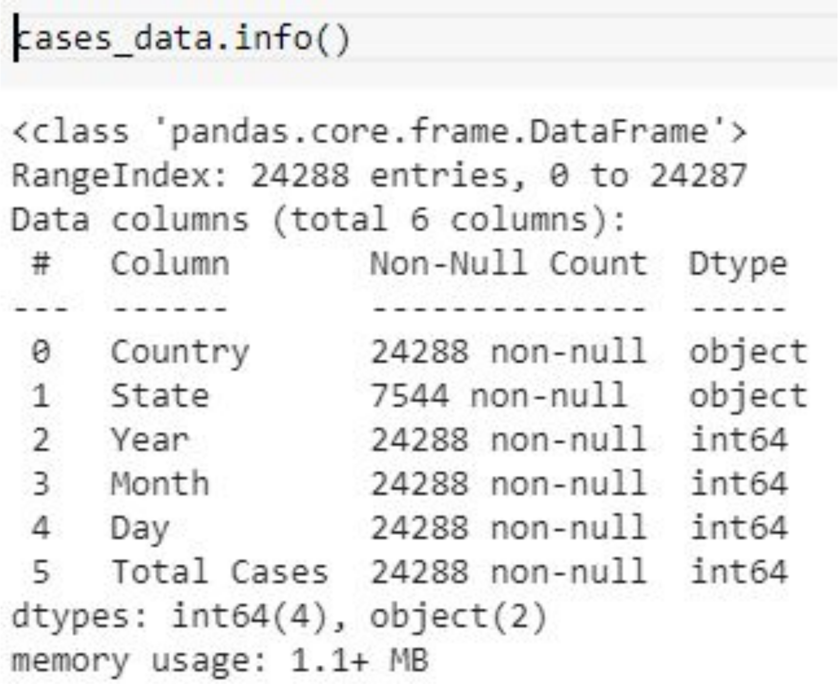
```
cases_data.index
```

```
RangeIndex(start=0, stop=24288, step=1)
```

```
cases_data.columns
```

```
Index(['Country', 'State', 'Year', 'Month', 'Day', 'Total Cases'], dtype='object')
```

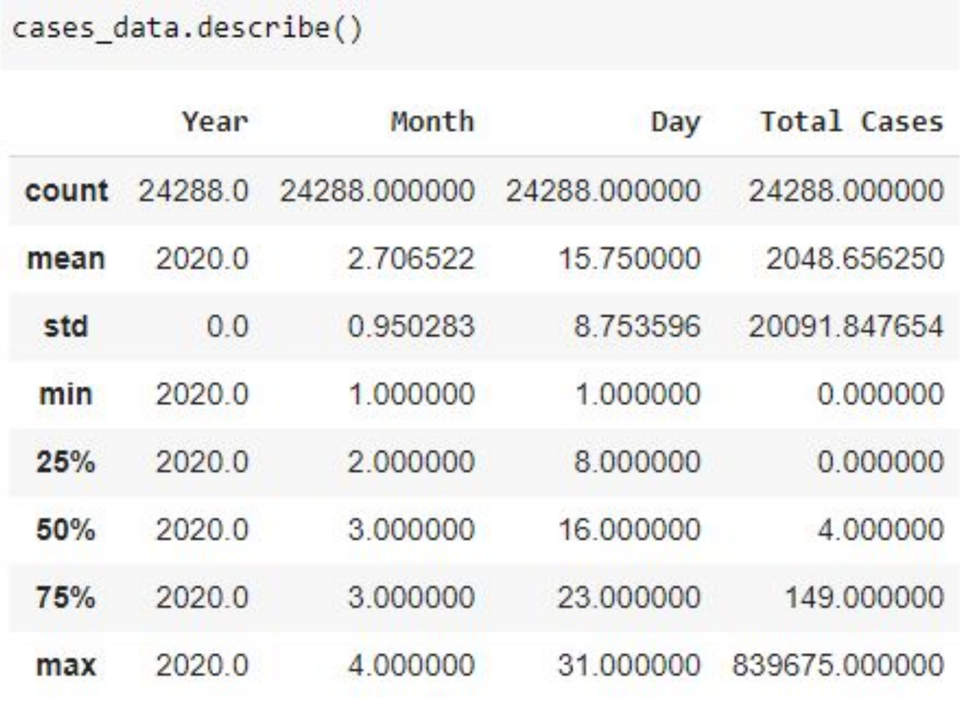
Some Syntaxes



```
cases_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24288 entries, 0 to 24287
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Country      24288 non-null  object
 1   State        7544 non-null   object
 2   Year         24288 non-null  int64 
 3   Month        24288 non-null  int64 
 4   Day          24288 non-null  int64 
 5   Total Cases  24288 non-null  int64 
dtypes: int64(4), object(2)
memory usage: 1.1+ MB
```

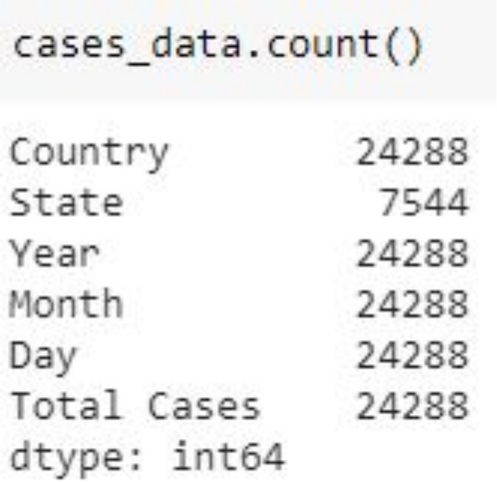
Some Syntaxes



```
cases_data.describe()
```

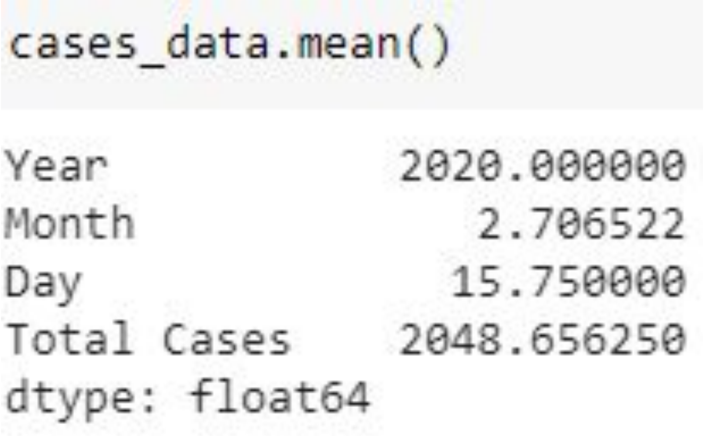
	Year	Month	Day	Total Cases
count	24288.0	24288.000000	24288.000000	24288.000000
mean	2020.0	2.706522	15.750000	2048.656250
std	0.0	0.950283	8.753596	20091.847654
min	2020.0	1.000000	1.000000	0.000000
25%	2020.0	2.000000	8.000000	0.000000
50%	2020.0	3.000000	16.000000	4.000000
75%	2020.0	3.000000	23.000000	149.000000
max	2020.0	4.000000	31.000000	839675.000000

Some Syntaxes



```
cases_data.count()
```

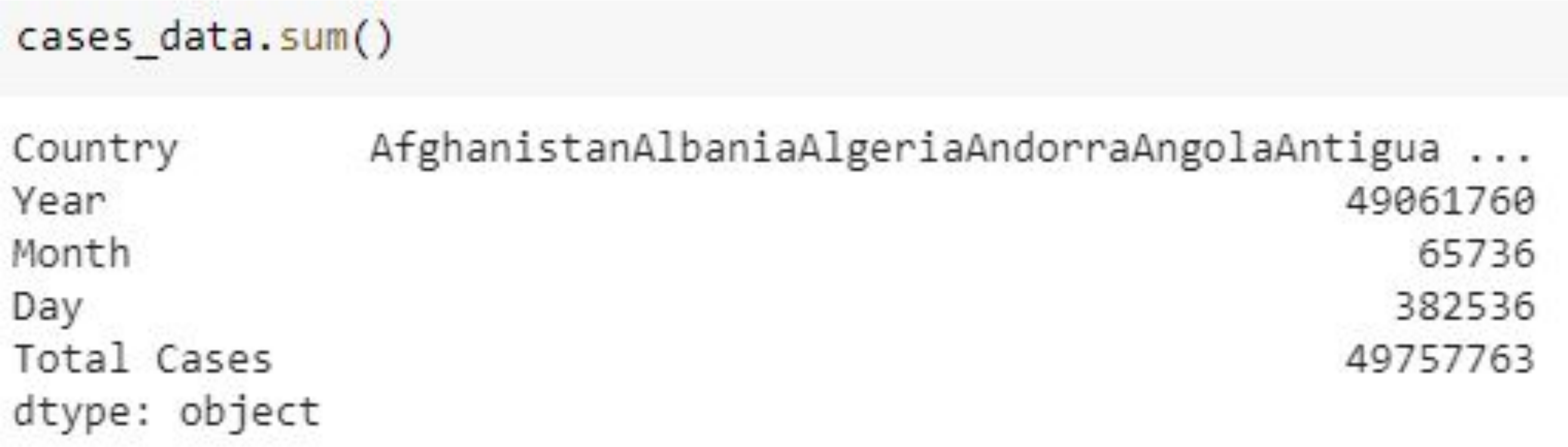
```
Country        24288
State           7544
Year           24288
Month          24288
Day            24288
Total Cases    24288
dtype: int64
```



```
cases_data.mean()
```

```
Year           2020.000000
Month             2.706522
Day              15.750000
Total Cases    2048.656250
dtype: float64
```

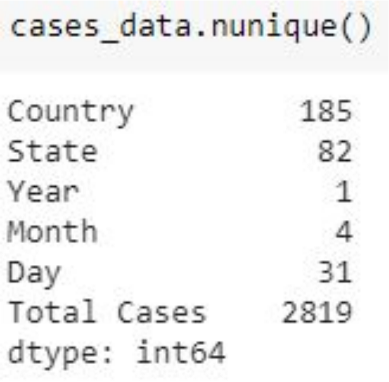
Some Syntaxes



```
cases_data.sum()
```

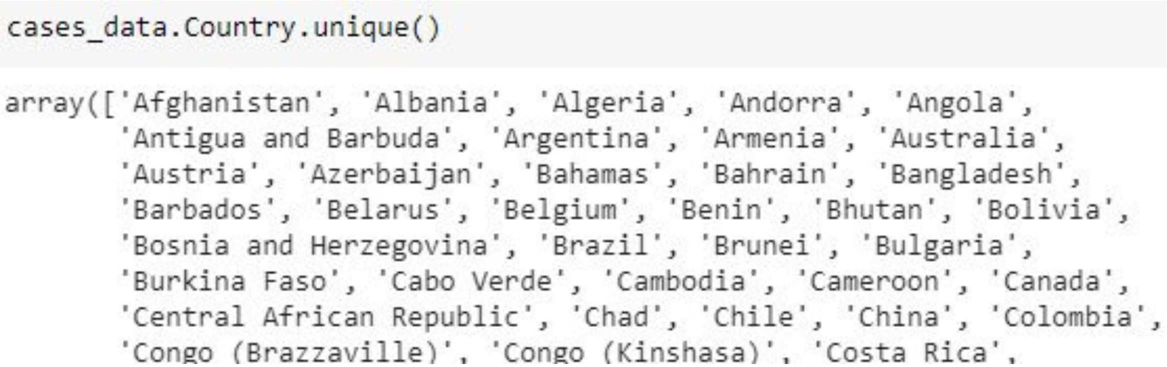
```
Country        AfghanistanAlbaniaAlgeriaAndorraAngolaAntigua ...
Year                                                    49061760
Month                                                      65736
Day                                                       382536
Total Cases                                             49757763
dtype: object
```

Some Syntaxes



```
cases_data.nunique()
```

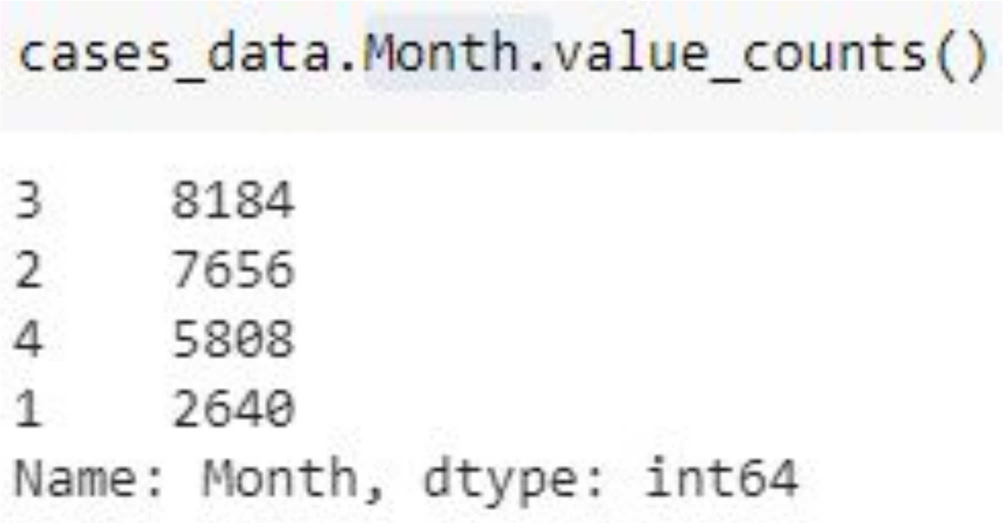
```
Country         185
State            82
Year              1
Month             4
Day              31
Total Cases    2819
dtype: int64
```



```
cases_data.Country.unique()
```

```
array(['Afghanistan', 'Albania', 'Algeria', 'Andorra', 'Angola',
       'Antigua and Barbuda', 'Argentina', 'Armenia', 'Australia',
       'Austria', 'Azerbaijan', 'Bahamas', 'Bahrain', 'Bangladesh',
       'Barbados', 'Belarus', 'Belgium', 'Benin', 'Bhutan', 'Bolivia',
       'Bosnia and Herzegovina', 'Brazil', 'Brunei', 'Bulgaria',
       'Burkina Faso', 'Cabo Verde', 'Cambodia', 'Cameroon', 'Canada',
       'Central African Republic', 'Chad', 'Chile', 'China', 'Colombia',
       'Congo (Brazzaville)', 'Congo (Kinshasa)', 'Costa Rica',
```

Some Syntaxes



```
cases_data.Month.value_counts()
```

```
3    8184
2    7656
4    5808
1    2640
Name: Month, dtype: int64
```

Maps

If you want to apply a function to a specific column/rows, Pandas has provided two methods for you.

- map, works on a single column, i.e. Series.
  - Might have learnt about this during lists, they go the same way.
- apply, works on every single row, making a new column.

Maps

Let’s say all the year in cases_data is 1 year earlier. Let’s shift it 1 years later.

```
cases_data.Year.map(lambda x: x+1)

or

cases_data[‘Year’].apply(lambda x: x+1)
```

Note that this does not modify our original DataFrame!

Maps + Practice Part 5

Similarly, we can also do simple operations on a DataFrame.

For example,

```
normal_data.salary / 5
```

```
cases_data.Day.apply(str) + ‘-’ +
cases_data.Month.apply(str) + ‘-’ +
cases_data.Year.apply(str)
```

Filtering

One of the most commonly used feature in Pandas because it can easily slice and dice any DataFrame.

There are many scenarios of filtering rows and/or columns in Pandas.

Selecting based on a Value

One way is to use a boolean expression.

- data[data[‘Year’] == 2020] will select all rows having 2020 as the value in the Year column.
- Similarly, data[data[‘Year’] != 2020] will select all rows not having 2020 as the value in the Year column.

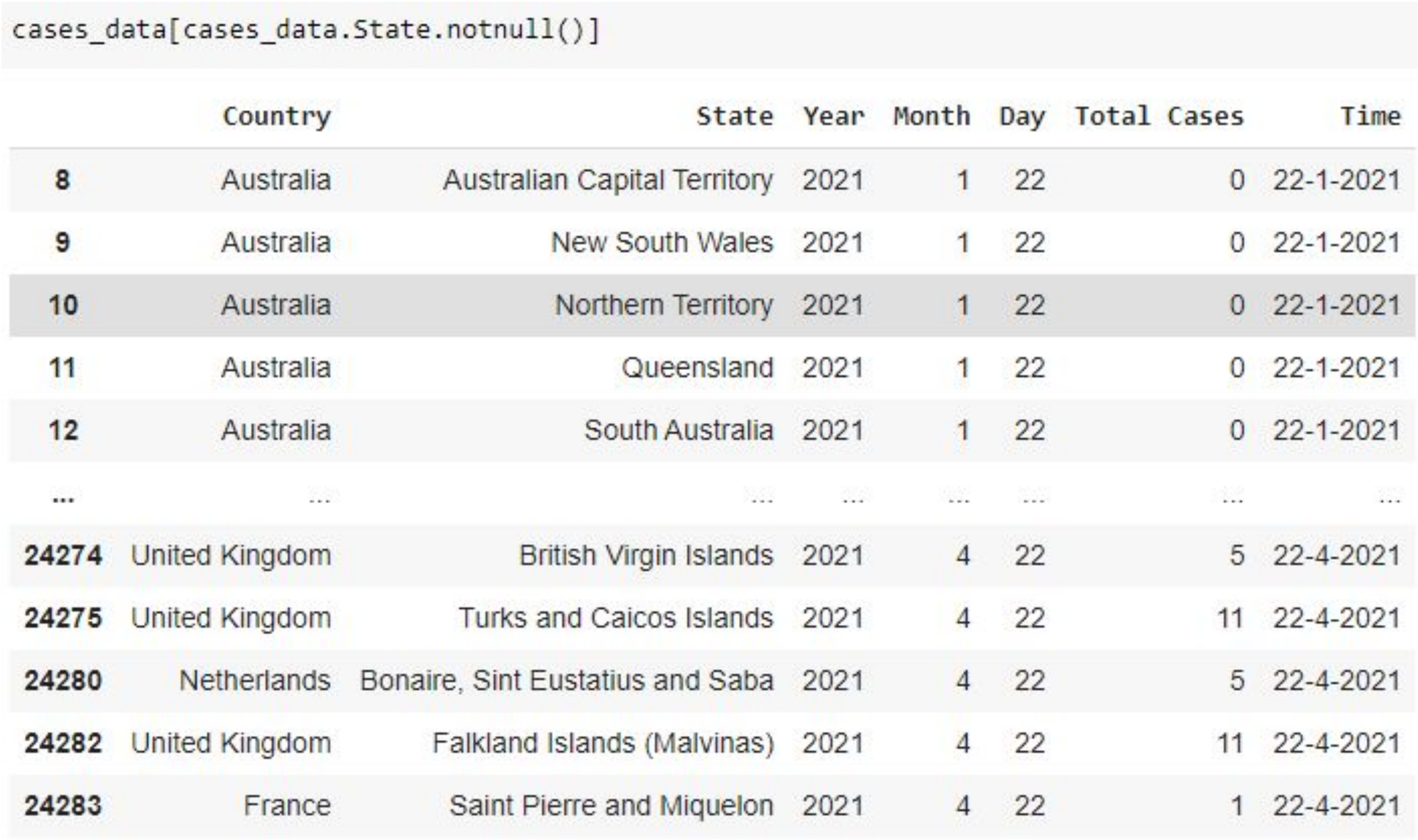
Selecting NA/NAN Values

Often, we find DataFrames with entries NA or NaN. We might want to exclude this from our cleaned dataset.

We can use the notnull() method for this.

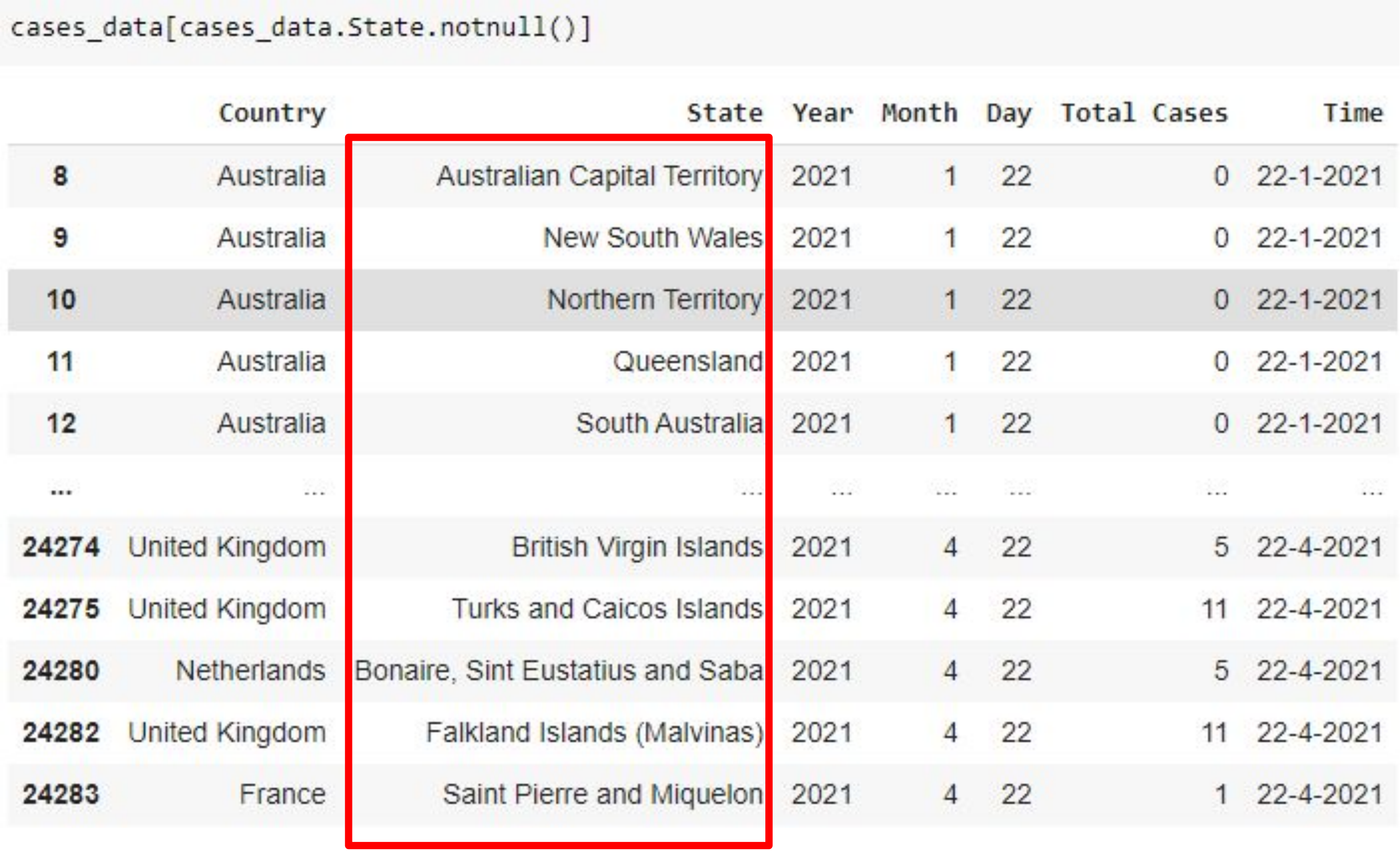
- data[data[‘Year’].notnull()], or
- data[data.Year.notnull()]

In contrast, we use isnull() should we want to include only them instead.



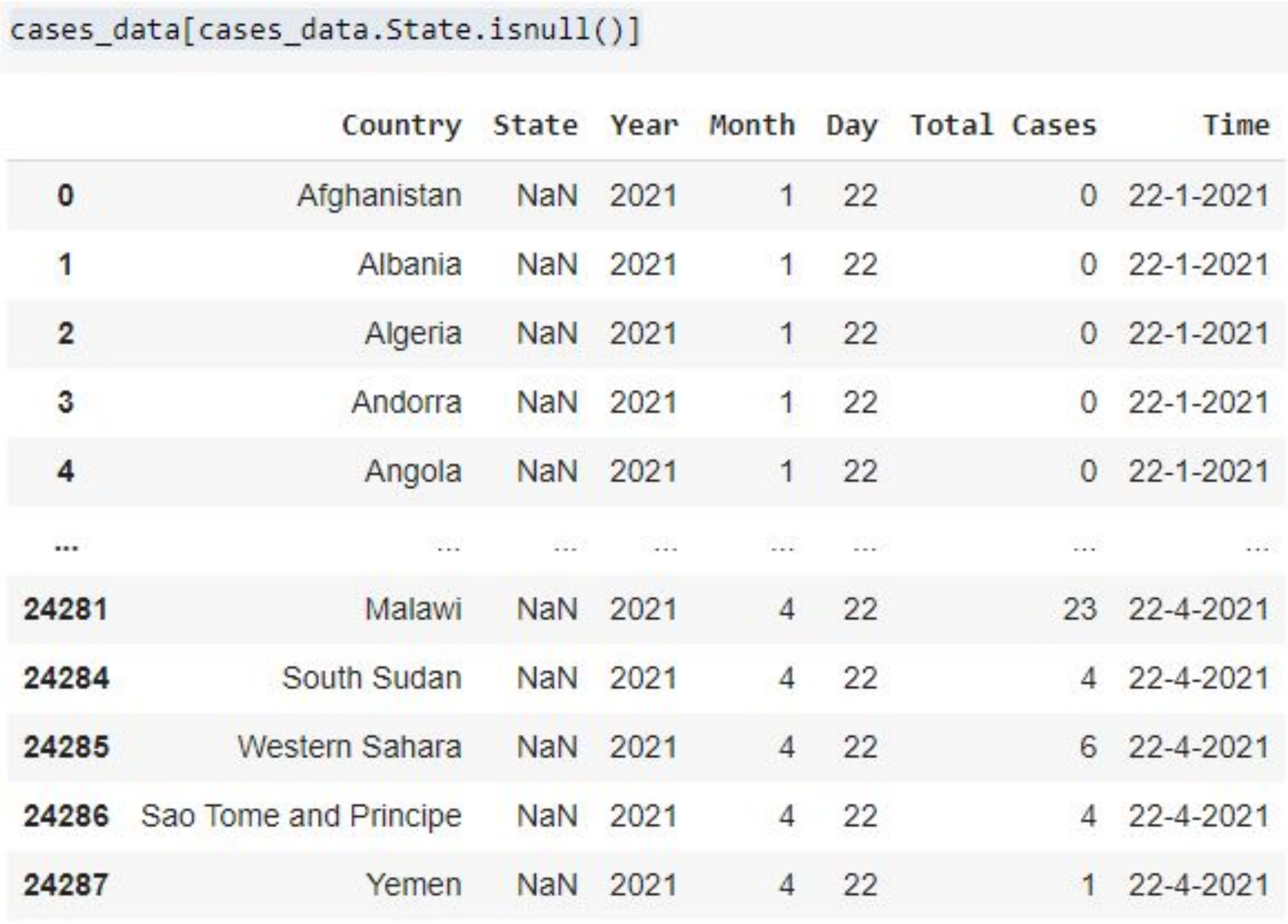
```
cases_data[cases_data.State.notnull()]
```

	Country	State	Year	Month	Day	Total Cases	Time
8	Australia	Australian Capital Territory	2021	1	22	0	22-1-2021
9	Australia	New South Wales	2021	1	22	0	22-1-2021
10	Australia	Northern Territory	2021	1	22	0	22-1-2021
11	Australia	Queensland	2021	1	22	0	22-1-2021
12	Australia	South Australia	2021	1	22	0	22-1-2021
...	...	...	...	...	...	...	...
24274	United Kingdom	British Virgin Islands	2021	4	22	5	22-4-2021
24275	United Kingdom	Turks and Caicos Islands	2021	4	22	11	22-4-2021
24280	Netherlands	Bonaire, Sint Eustatius and Saba	2021	4	22	5	22-4-2021
24282	United Kingdom	Falkland Islands (Malvinas)	2021	4	22	11	22-4-2021
24283	France	Saint Pierre and Miquelon	2021	4	22	1	22-4-2021



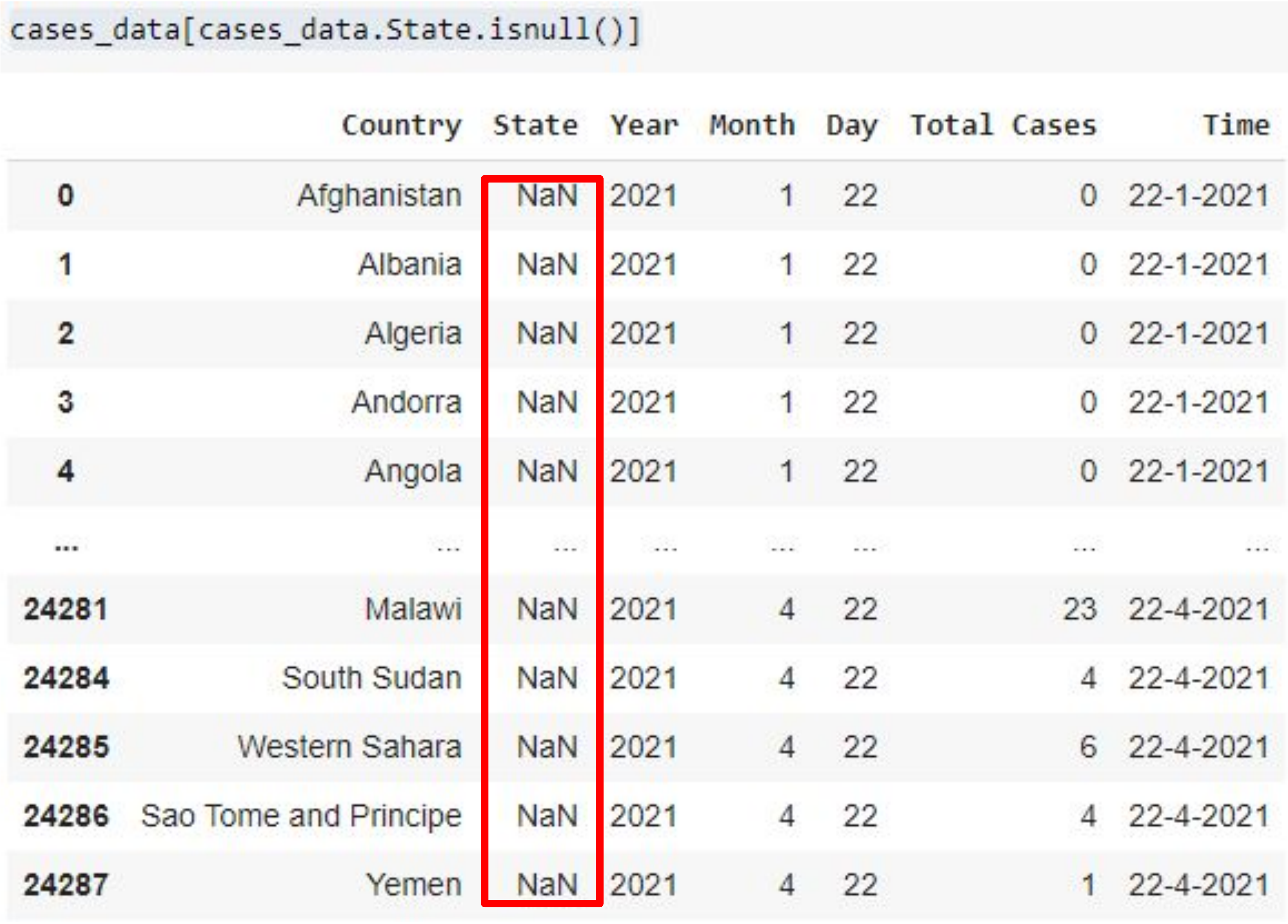
```
cases_data[cases_data.State.notnull()]
```

	Country	State	Year	Month	Day	Total Cases	Time
8	Australia	Australian Capital Territory	2021	1	22	0	22-1-2021
9	Australia	New South Wales	2021	1	22	0	22-1-2021
10	Australia	Northern Territory	2021	1	22	0	22-1-2021
11	Australia	Queensland	2021	1	22	0	22-1-2021
12	Australia	South Australia	2021	1	22	0	22-1-2021
...	...	...	...	...	...	...	...
24274	United Kingdom	British Virgin Islands	2021	4	22	5	22-4-2021
24275	United Kingdom	Turks and Caicos Islands	2021	4	22	11	22-4-2021
24280	Netherlands	Bonaire, Sint Eustatius and Saba	2021	4	22	5	22-4-2021
24282	United Kingdom	Falkland Islands (Malvinas)	2021	4	22	11	22-4-2021
24283	France	Saint Pierre and Miquelon	2021	4	22	1	22-4-2021



```
cases_data[cases_data.State.isnull()]
```

	Country	State	Year	Month	Day	Total Cases	Time
0	Afghanistan	NaN	2021	1	22	0	22-1-2021
1	Albania	NaN	2021	1	22	0	22-1-2021
2	Algeria	NaN	2021	1	22	0	22-1-2021
3	Andorra	NaN	2021	1	22	0	22-1-2021
4	Angola	NaN	2021	1	22	0	22-1-2021
...	...	...	...	...	...	...	...
24281	Malawi	NaN	2021	4	22	23	22-4-2021
24284	South Sudan	NaN	2021	4	22	4	22-4-2021
24285	Western Sahara	NaN	2021	4	22	6	22-4-2021
24286	Sao Tome and Principe	NaN	2021	4	22	4	22-4-2021
24287	Yemen	NaN	2021	4	22	1	22-4-2021



```
cases_data[cases_data.State.isnull()]
```

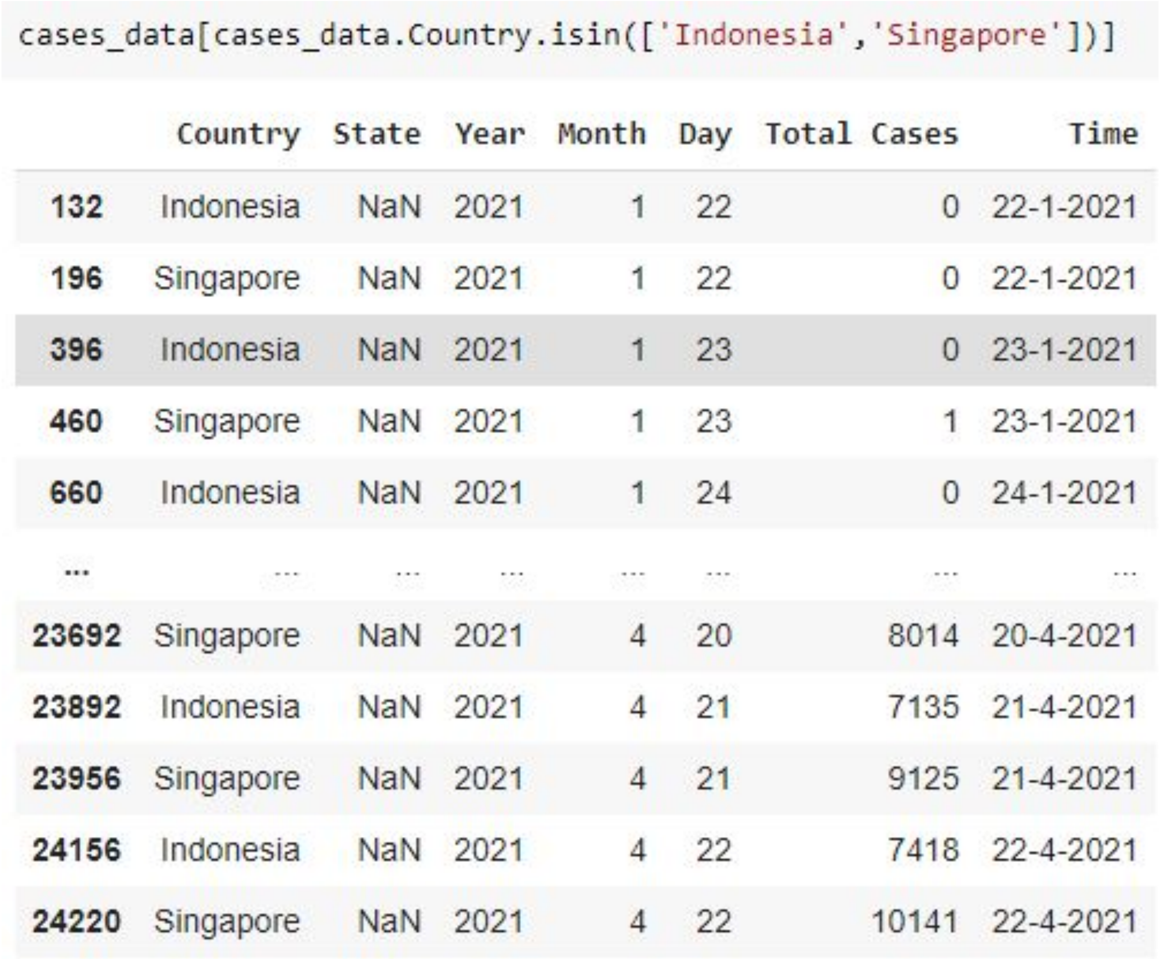
	Country	State	Year	Month	Day	Total Cases	Time
0	Afghanistan	NaN	2021	1	22	0	22-1-2021
1	Albania	NaN	2021	1	22	0	22-1-2021
2	Algeria	NaN	2021	1	22	0	22-1-2021
3	Andorra	NaN	2021	1	22	0	22-1-2021
4	Angola	NaN	2021	1	22	0	22-1-2021
...	...	...	...	...	...	...	...
24281	Malawi	NaN	2021	4	22	23	22-4-2021
24284	South Sudan	NaN	2021	4	22	4	22-4-2021
24285	Western Sahara	NaN	2021	4	22	6	22-4-2021
24286	Sao Tome and Principe	NaN	2021	4	22	4	22-4-2021
24287	Yemen	NaN	2021	4	22	1	22-4-2021

Selecting based on a List

Sometimes, we want to select rows based on multiple values, hence we use the isin() method to complete the work.

For example, to select cases data on only Indonesia and Singapore, we use

```
cases_data[cases_data.Country.isin([‘Indonesia’,’Singapore’])]
```



```
cases_data[cases_data.Country.isin(['Indonesia','Singapore'])]
```

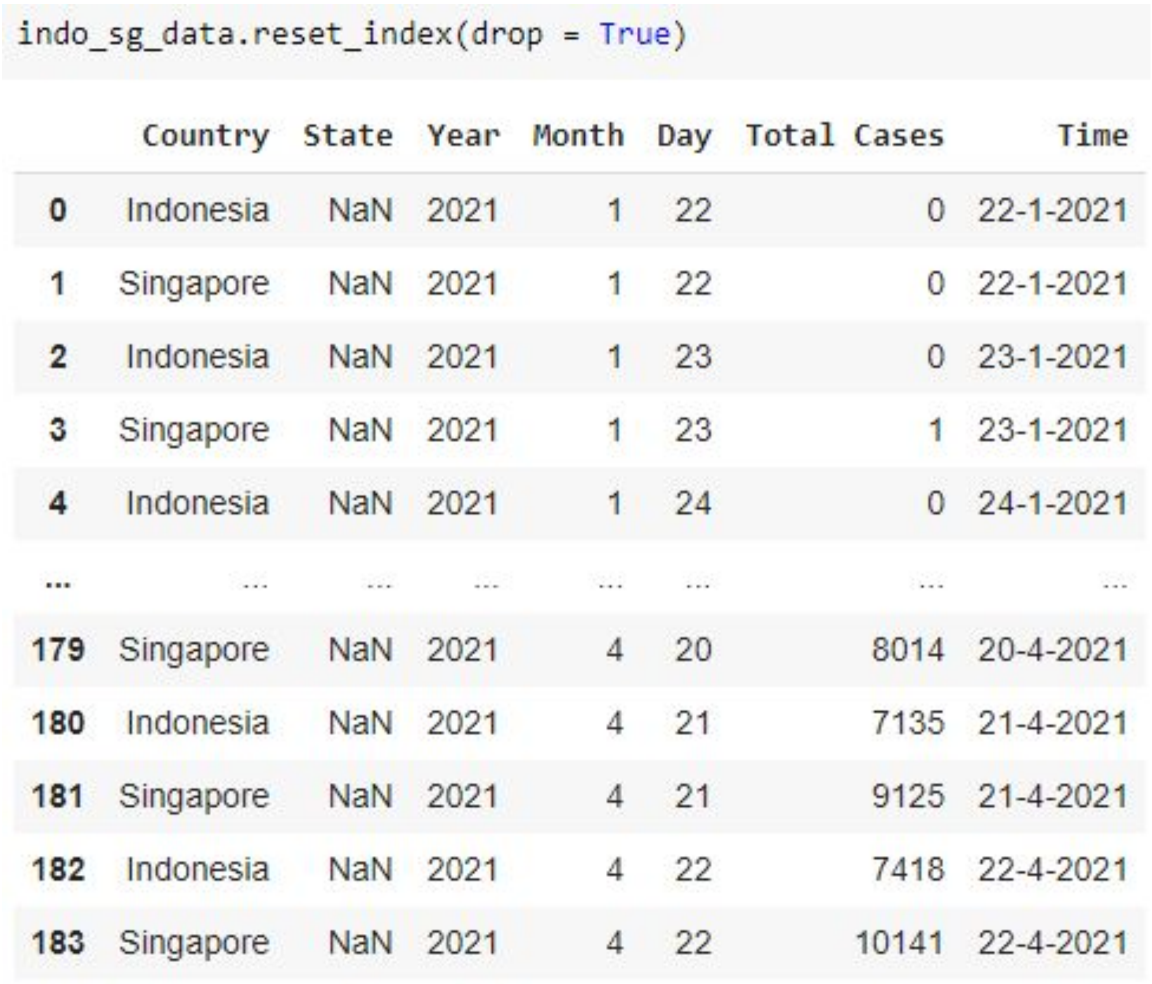
	Country	State	Year	Month	Day	Total Cases	Time
132	Indonesia	NaN	2021	1	22	0	22-1-2021
196	Singapore	NaN	2021	1	22	0	22-1-2021
396	Indonesia	NaN	2021	1	23	0	23-1-2021
460	Singapore	NaN	2021	1	23	1	23-1-2021
660	Indonesia	NaN	2021	1	24	0	24-1-2021
...	...	...	...	...	...	...	...
23692	Singapore	NaN	2021	4	20	8014	20-4-2021
23892	Indonesia	NaN	2021	4	21	7135	21-4-2021
23956	Singapore	NaN	2021	4	21	9125	21-4-2021
24156	Indonesia	NaN	2021	4	22	7418	22-4-2021
24220	Singapore	NaN	2021	4	22	10141	22-4-2021

Resetting the Index

As you see in the previous DataFrame, the indexes are messed up. To resolve this, we use the reset_index() method. Assume that we assign our previous DataFrame to indo_sg_data.

```
indo_sg_data.reset_index(drop = True)
```

What happens if we remove drop = True?



```
indo_sg_data.reset_index(drop = True)
```

	Country	State	Year	Month	Day	Total Cases	Time
0	Indonesia	NaN	2021	1	22	0	22-1-2021
1	Singapore	NaN	2021	1	22	0	22-1-2021
2	Indonesia	NaN	2021	1	23	0	23-1-2021
3	Singapore	NaN	2021	1	23	1	23-1-2021
4	Indonesia	NaN	2021	1	24	0	24-1-2021
...	...	...	...	...	...	...	...
179	Singapore	NaN	2021	4	20	8014	20-4-2021
180	Indonesia	NaN	2021	4	21	7135	21-4-2021
181	Singapore	NaN	2021	4	21	9125	21-4-2021
182	Indonesia	NaN	2021	4	22	7418	22-4-2021
183	Singapore	NaN	2021	4	22	10141	22-4-2021

Excluding based on a List

This is the opposite of selecting rows based on a list. So, we take rows whose column values are not in the list.

We simply negate our previous boolean statement with the negation symbol (~). For example, the following code will exclude Indonesia and Singapore

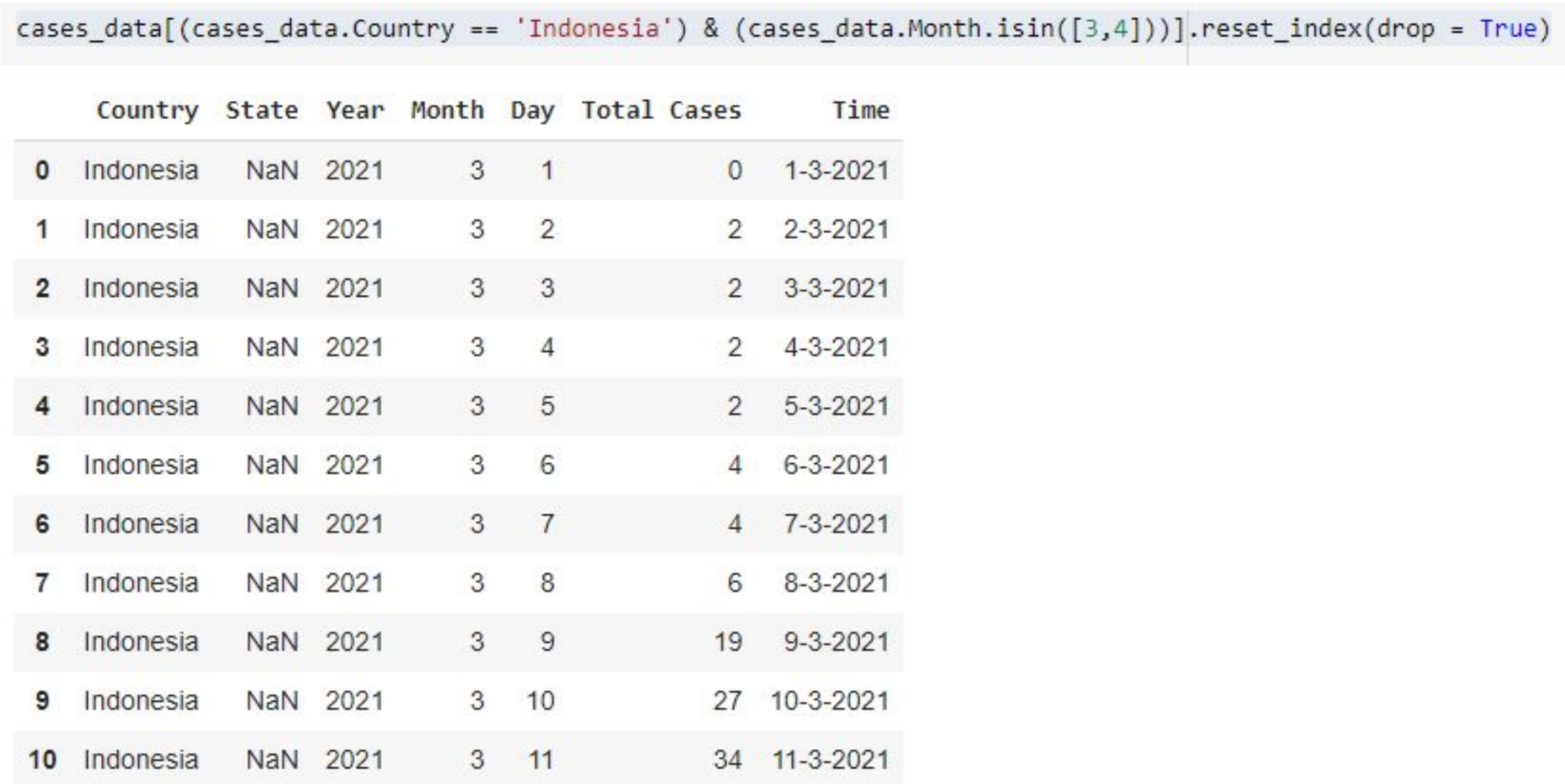
```
cases_data[~cases_data.Country.isin([‘Indonesia’,’Sin
gapore’])]
```

Selecting with Multiple Conditions

Finally, to put them altogether, we use the & symbol to combine the conditionals.

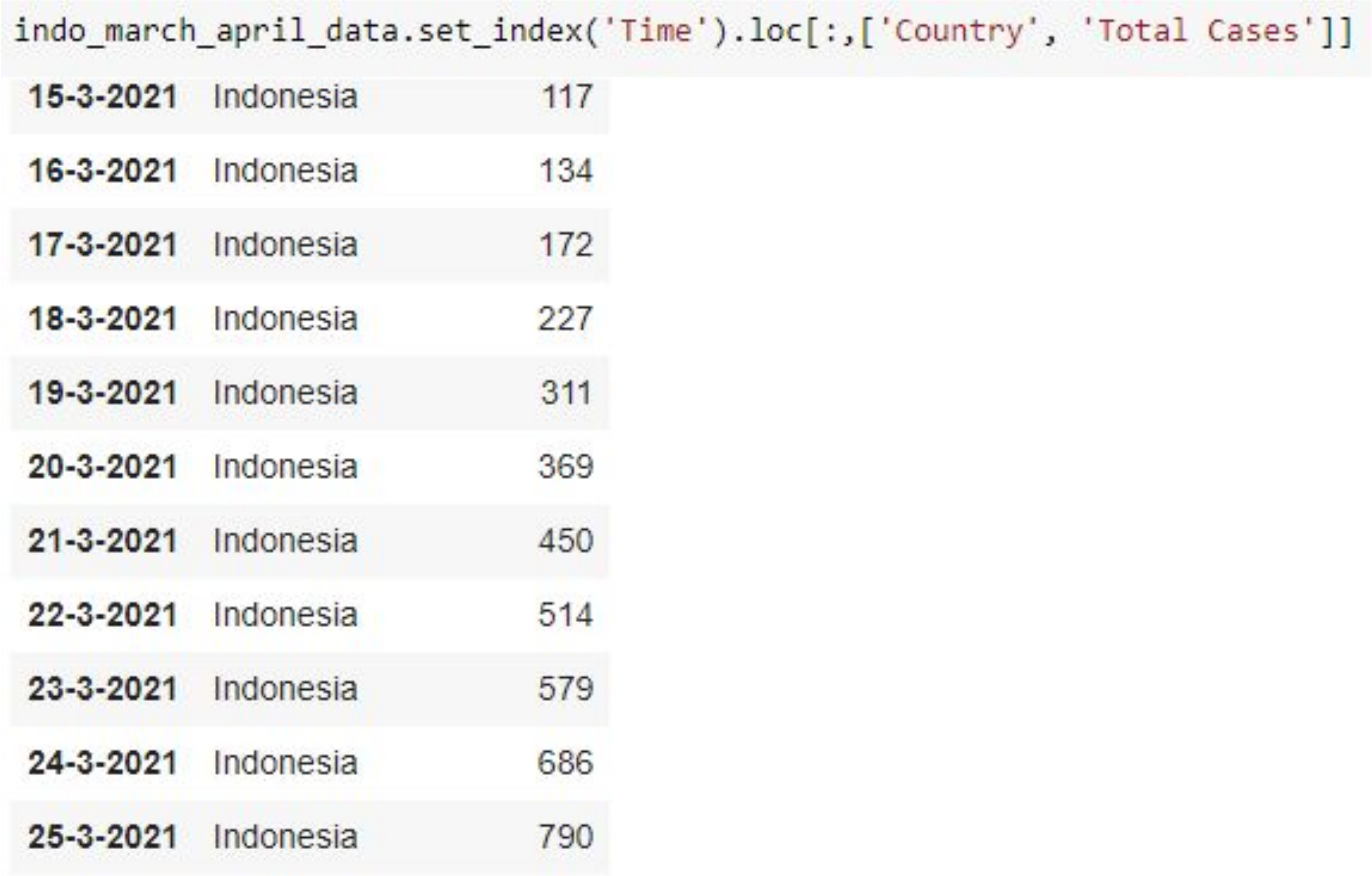
For example, I only want cases data of Indonesia on March and April. We use

```
cases_data[(cases_data.Country == 'Indonesia') &
(cases_data.Month.isin([3,4]))].reset_index(drop =
True)
```



```
cases_data[(cases_data.Country == 'Indonesia') & (cases_data.Month.isin([3,4]))].reset_index(drop = True)
```

	Country	State	Year	Month	Day	Total Cases	Time
0	Indonesia	NaN	2021	3	1	0	1-3-2021
1	Indonesia	NaN	2021	3	2	2	2-3-2021
2	Indonesia	NaN	2021	3	3	2	3-3-2021
3	Indonesia	NaN	2021	3	4	2	4-3-2021
4	Indonesia	NaN	2021	3	5	2	5-3-2021
5	Indonesia	NaN	2021	3	6	4	6-3-2021
6	Indonesia	NaN	2021	3	7	4	7-3-2021
7	Indonesia	NaN	2021	3	8	6	8-3-2021
8	Indonesia	NaN	2021	3	9	19	9-3-2021
9	Indonesia	NaN	2021	3	10	27	10-3-2021
10	Indonesia	NaN	2021	3	11	34	11-3-2021



```python
indo_march_april_data.set_index('Time').loc[:,['Country', 'Total Cases']]
```

15-3-2021	Indonesia	117
16-3-2021	Indonesia	134
17-3-2021	Indonesia	172
18-3-2021	Indonesia	227
19-3-2021	Indonesia	311
20-3-2021	Indonesia	369
21-3-2021	Indonesia	450
22-3-2021	Indonesia	514
23-3-2021	Indonesia	579
24-3-2021	Indonesia	686
25-3-2021	Indonesia	790

Part 3-6 Summary

We have learnt

- How to extract columns and/or rows
- Adding one or more columns and/or rows
- Getting important summaries from the dataset
- Applying calculations to a certain row to produce a new data
- Filtering data based on many different scenarios

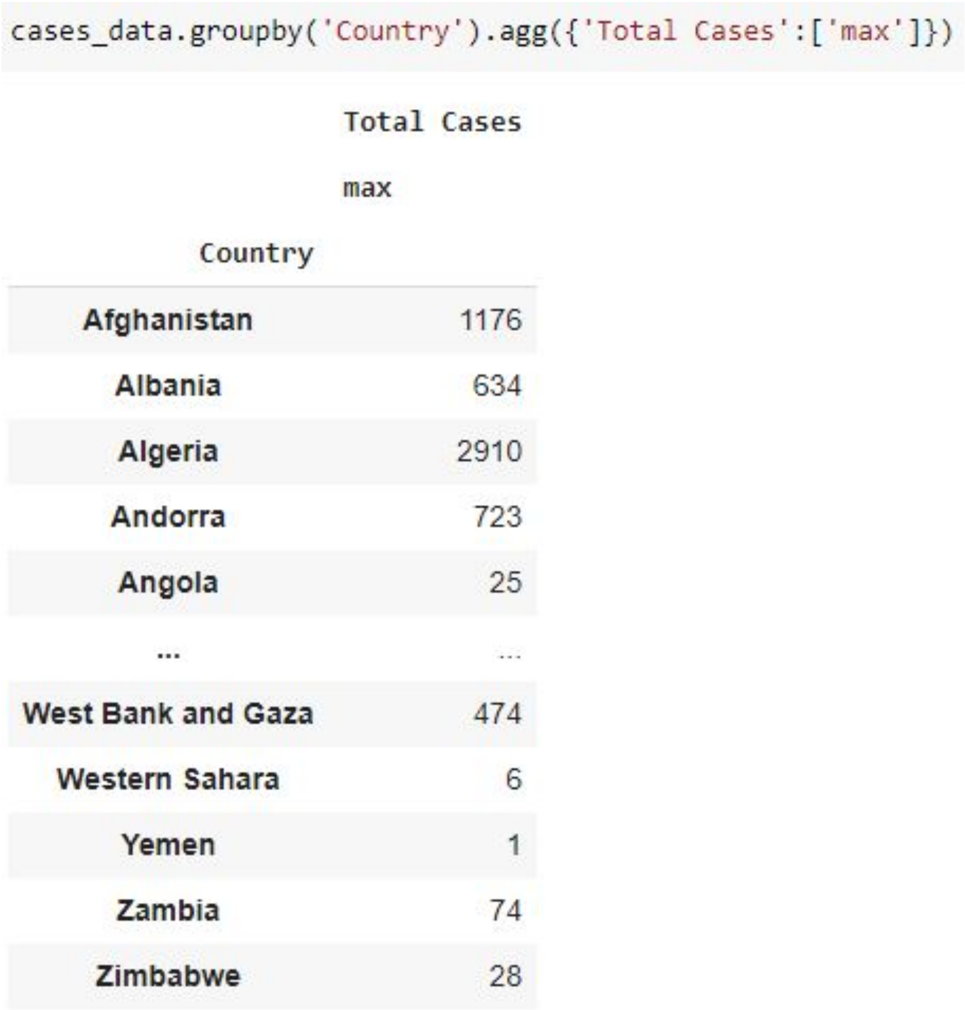
Practice Part 6

Among the countries Indonesia, Singapore, and US, select the cases on April with > 5000 total cases. Display only the country, time, and the total cases.

Grouping

Pandas provides you a function called **groupby()** to group datas with a common value on a specific column.

After grouping we can pass aggregation functions to the grouped object as a dictionary within the **agg** function.

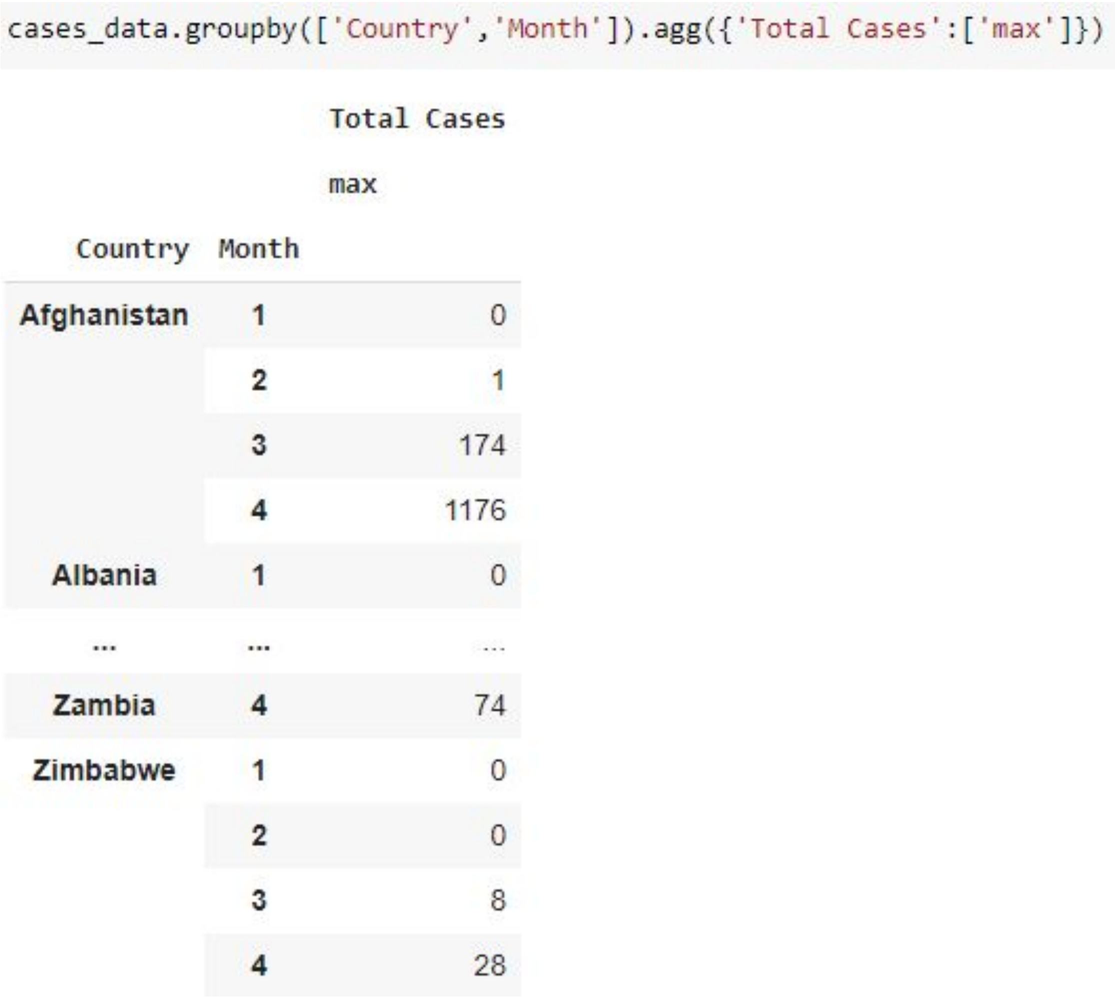


```python
cases_data.groupby('Country').agg({'Total Cases':['max']})
```

```
                    Total Cases
                    max
         Country
     Afghanistan           1176
         Albania            634
         Algeria           2910
         Andorra            723
          Angola             25
             ...            ...
West Bank and Gaza          474
  Western Sahara              6
           Yemen              1
          Zambia             74
        Zimbabwe             28
```

Grouping

We can also group by multiple columns, just change the parameter into a list of column names!



```
cases_data.groupby(['Country','Month']).agg({'Total Cases':['max']})
```

```
                     Total Cases
                             max
    Country  Month
Afghanistan      1             0
                 2             1
                 3           174
                 4          1176
    Albania      1             0
        ...    ...           ...
     Zambia      4            74
   Zimbabwe      1             0
                 2             0
                 3             8
                 4            28
```

Sorting

To get data in the order want it in we can sort it ourselves. The **sort_values()** method is handy for this.

```
data.sort_values(by = ‘Year’, ascending = False)
```

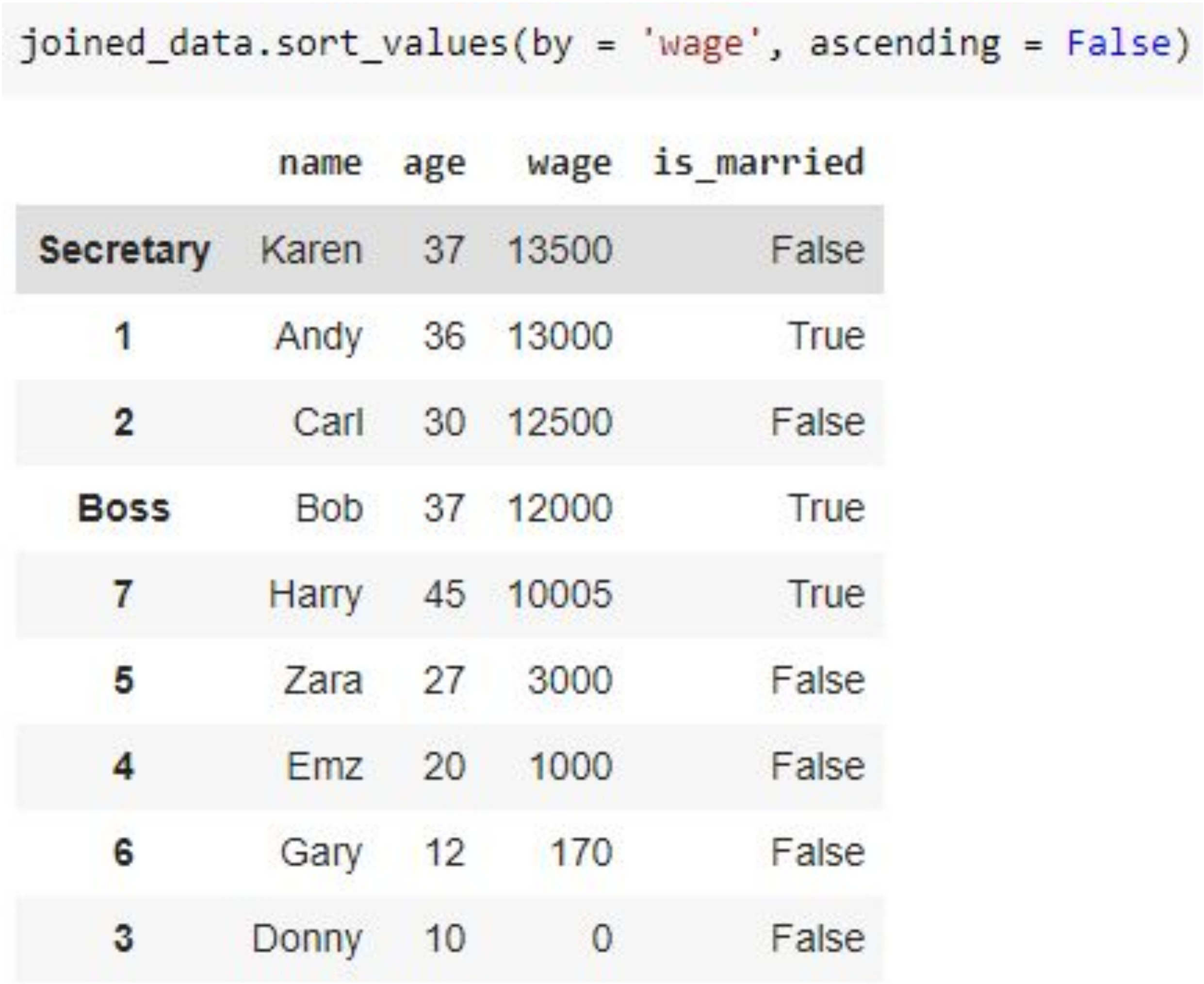
```
data.sort_values(by = [‘C1’, ‘C2’], ascending =
[True, False])
```

```
data.sort_values(by = [‘C1’, ‘C2’], ascending =
False)
```

Sorting

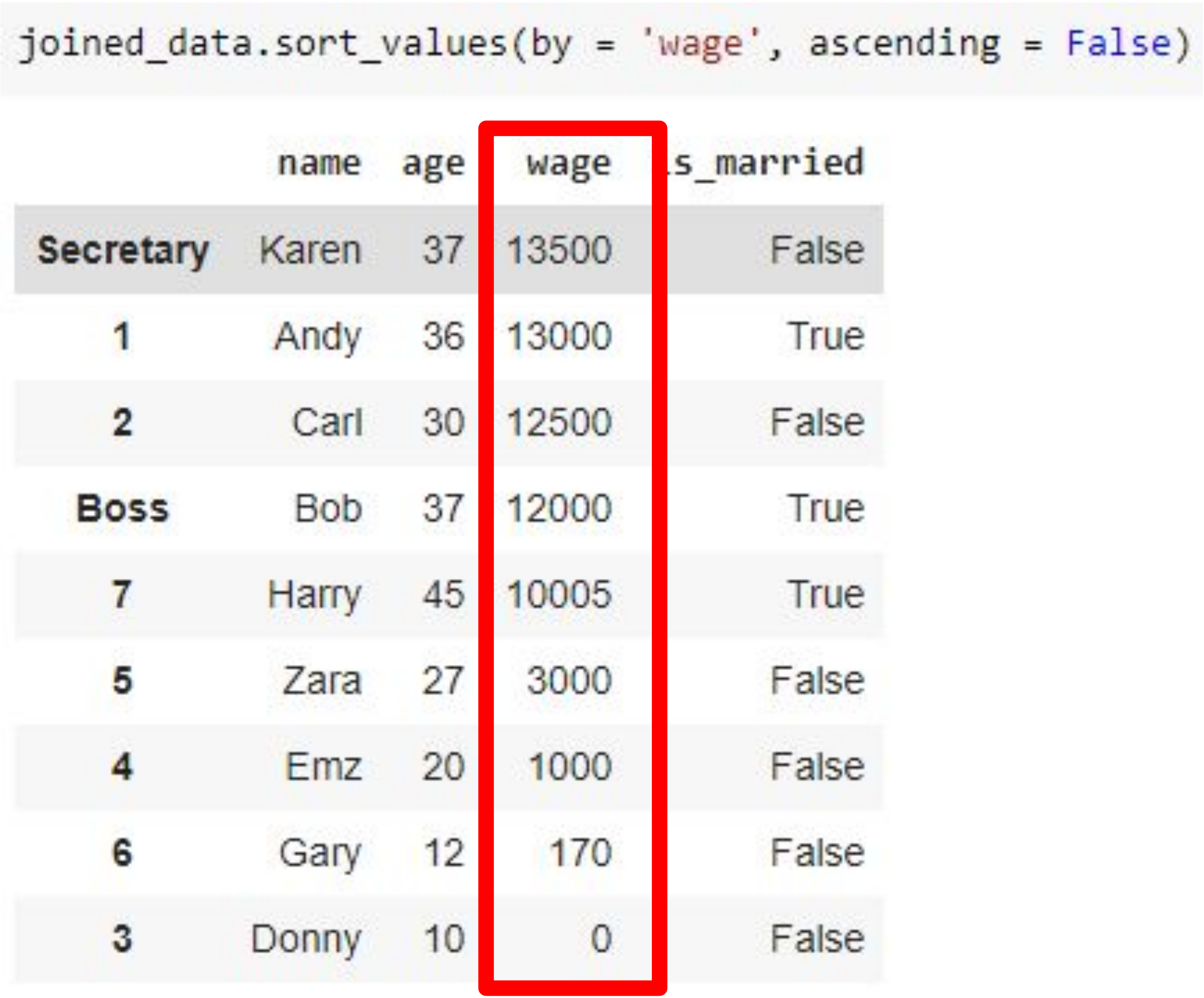
We can also sort the indexes instead using the **sort_index()** method.

```
data.sort_index(axis = 0) # sorts the row names

data.sort_index(axis = 1) # sorts the column names
```



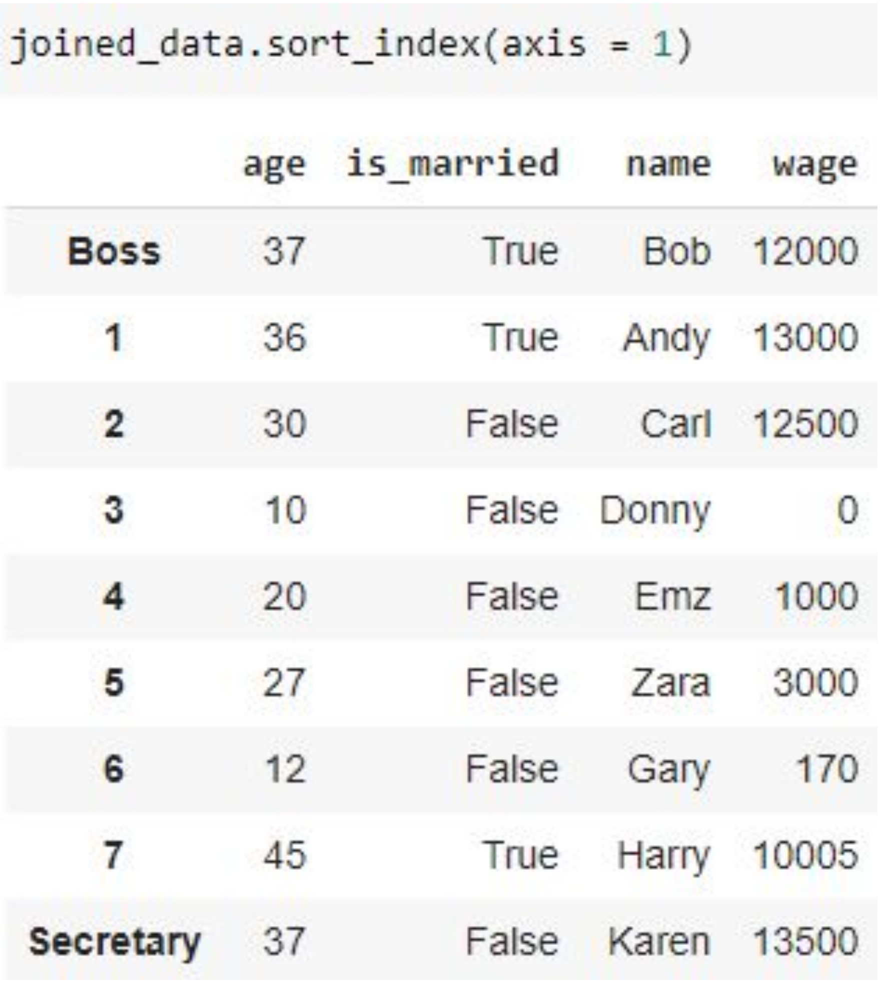
```python
joined_data.sort_values(by = 'wage', ascending = False)
```

	name	age	wage	is_married
Secretary	Karen	37	13500	False
1	Andy	36	13000	True
2	Carl	30	12500	False
Boss	Bob	37	12000	True
7	Harry	45	10005	True
5	Zara	27	3000	False
4	Emz	20	1000	False
6	Gary	12	170	False
3	Donny	10	0	False



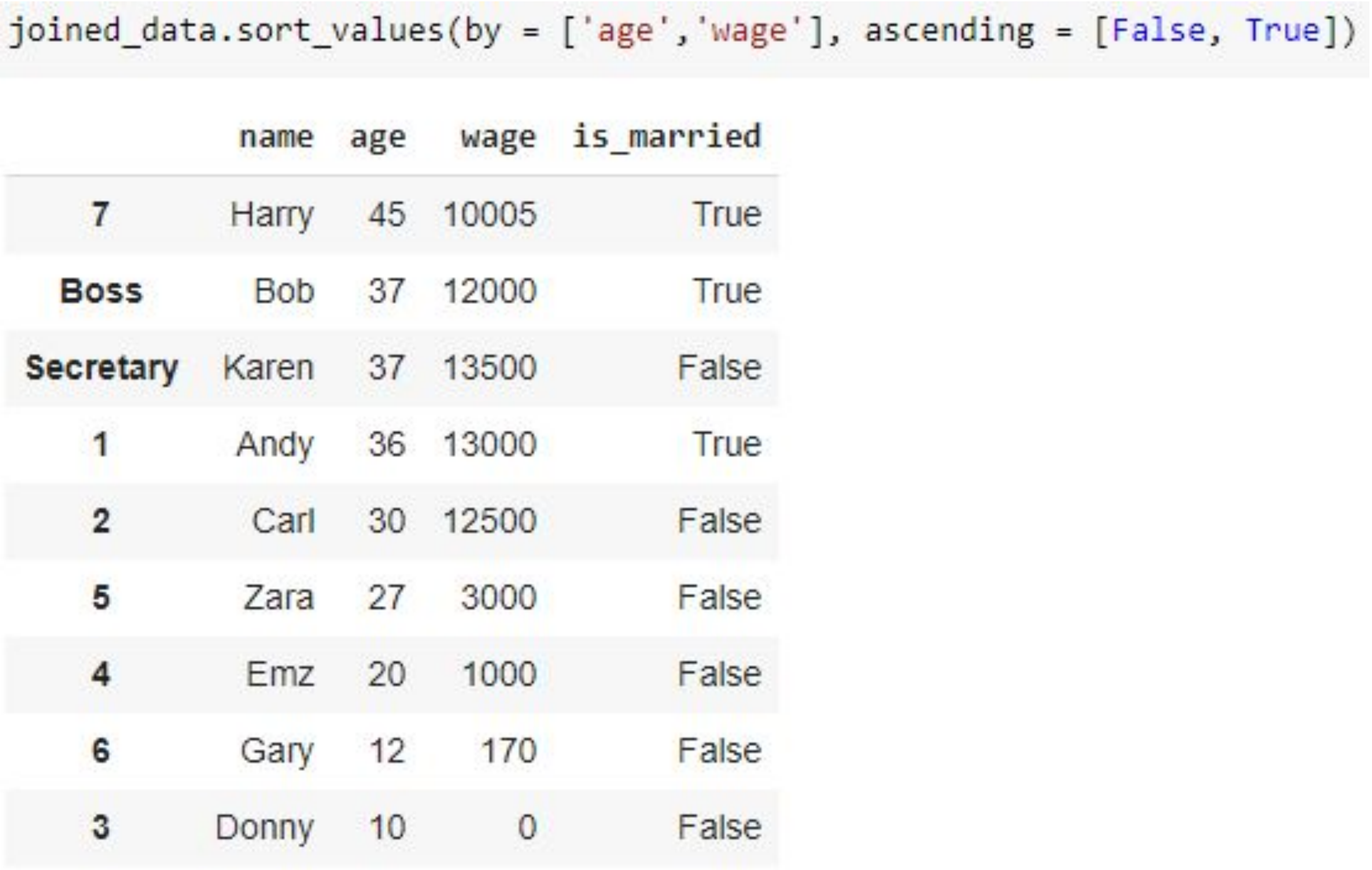
```
joined_data.sort_values(by = 'wage', ascending = False)
```

	name	age	wage	s_married
Secretary	Karen	37	13500	False
1	Andy	36	13000	True
2	Carl	30	12500	False
Boss	Bob	37	12000	True
7	Harry	45	10005	True
5	Zara	27	3000	False
4	Emz	20	1000	False
6	Gary	12	170	False
3	Donny	10	0	False



```python
joined_data.sort_index(axis = 1)
```

	age	is_married	name	wage
Boss	37	True	Bob	12000
1	36	True	Andy	13000
2	30	False	Carl	12500
3	10	False	Donny	0
4	20	False	Emz	1000
5	27	False	Zara	3000
6	12	False	Gary	170
7	45	True	Harry	10005
Secretary	37	False	Karen	13500



```
joined_data.sort_values(by = ['age','wage'], ascending = [False, True])
```

	name	age	wage	is_married
7	Harry	45	10005	True
Boss	Bob	37	12000	True
Secretary	Karen	37	13500	False
1	Andy	36	13000	True
2	Carl	30	12500	False
5	Zara	27	3000	False
4	Emz	20	1000	False
6	Gary	12	170	False
3	Donny	10	0	False

Pivoting

We won’t be going through this, but I encourage you to read

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html

And try some yourself! :)

Practice Part 7

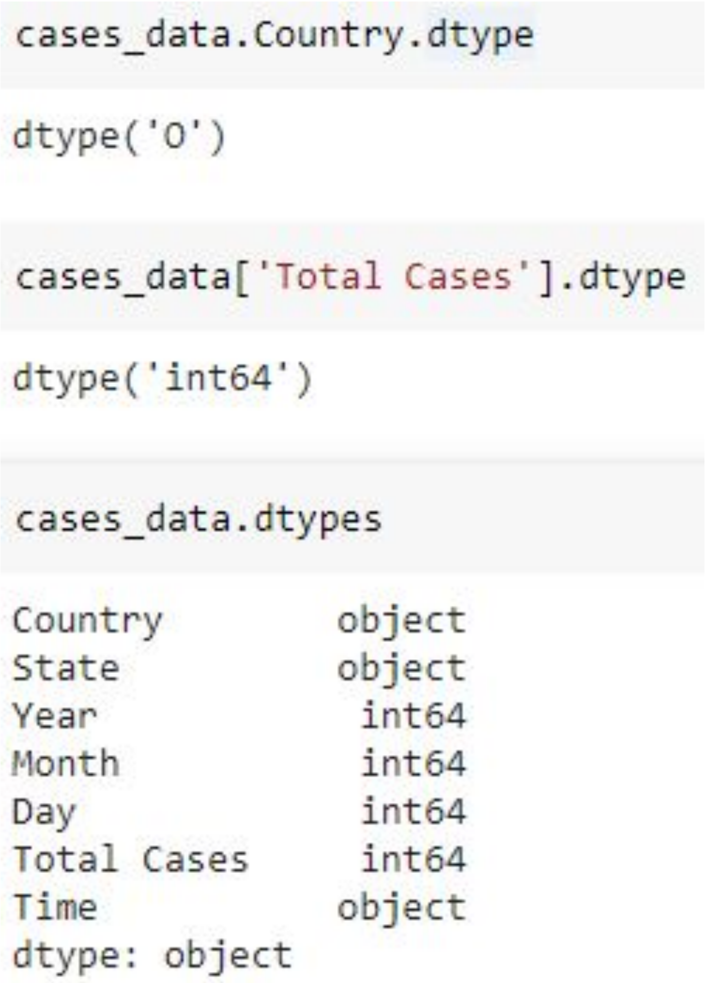
Among Indonesia, US, and Singapore, how many cases are registered after the first day of each month? Sort your result by country alphabetically in reverse order.

Dtypes

The data type for a column in a DataFrame or a Series is known as the dtype.

You can use the **dtype** property to grab the type of a specific column.

Alternatively, the **dtypes** property returns the dtype of every column in the DataFrame.



```
cases_data.Country.dtype
```

```
dtype('O')
```

```
cases_data['Total Cases'].dtype
```

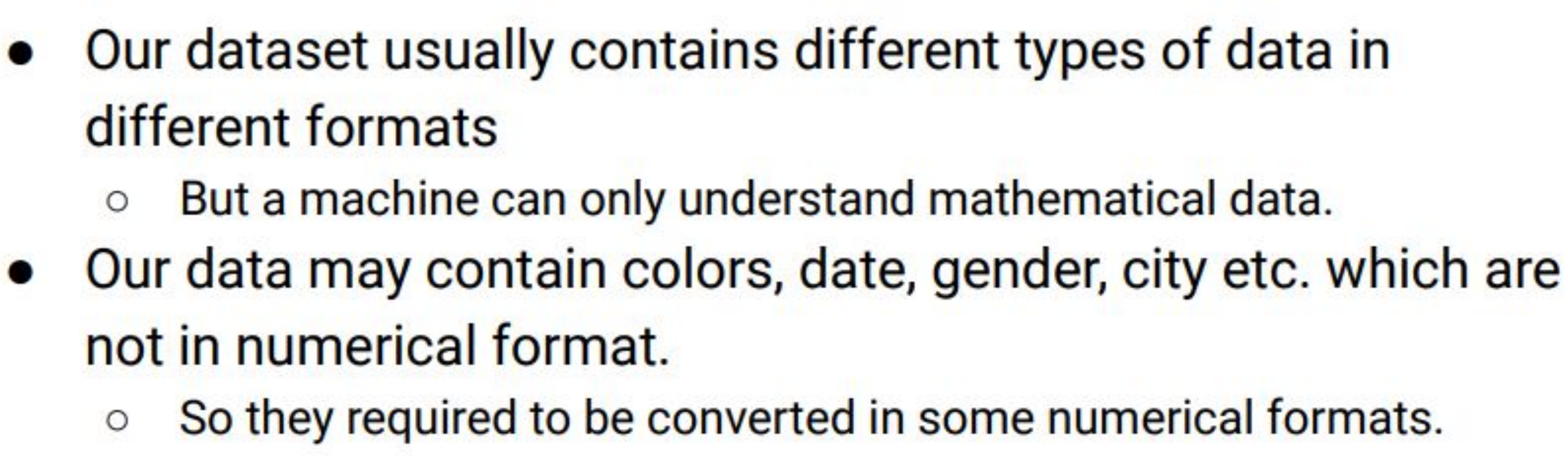
```
dtype('int64')
```

```
cases_data.dtypes
```

```
Country        object
State          object
Year            int64
Month           int64
Day             int64
Total Cases     int64
Time           object
dtype: object
```

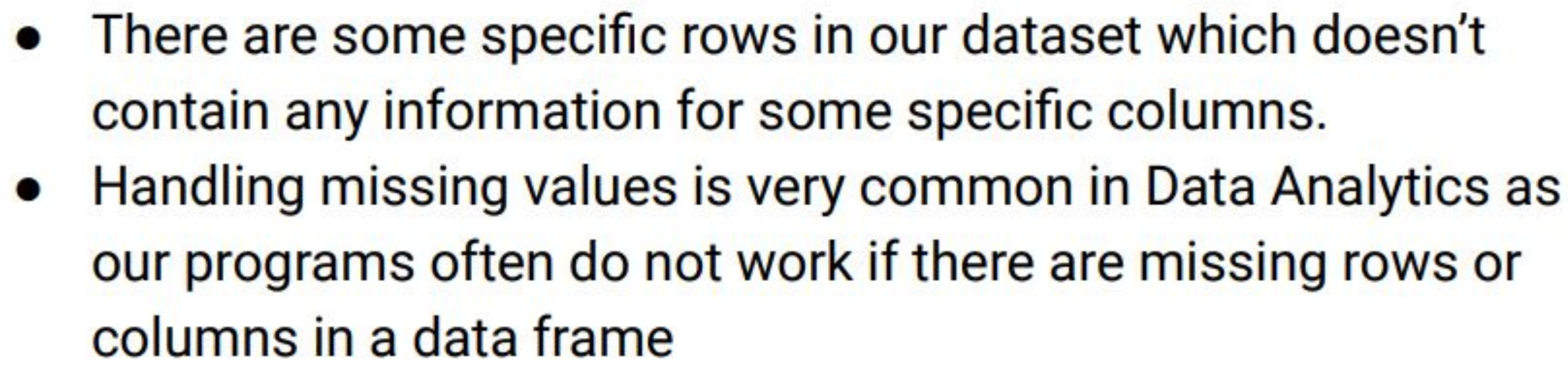
Two main motivations in data cleaning: Unstructured data and Missing data

Unstructured Data



- Our dataset usually contains different types of data in different formats
  - But a machine can only understand mathematical data.
- Our data may contain colors, date, gender, city etc. which are not in numerical format.
  - So they required to be converted in some numerical formats.

Missing Data



- There are some specific rows in our dataset which doesn’t contain any information for some specific columns.
- Handling missing values is very common in Data Analytics as our programs often do not work if there are missing rows or columns in a data frame

Data Preprocessing

Therefore, due to the existence of unstructured data, data preprocessing is necessary!

Data preprocessing is a process to convert unstructured types of data into a valid numerical format or categorical format that allows to work on these easier in further analysis.

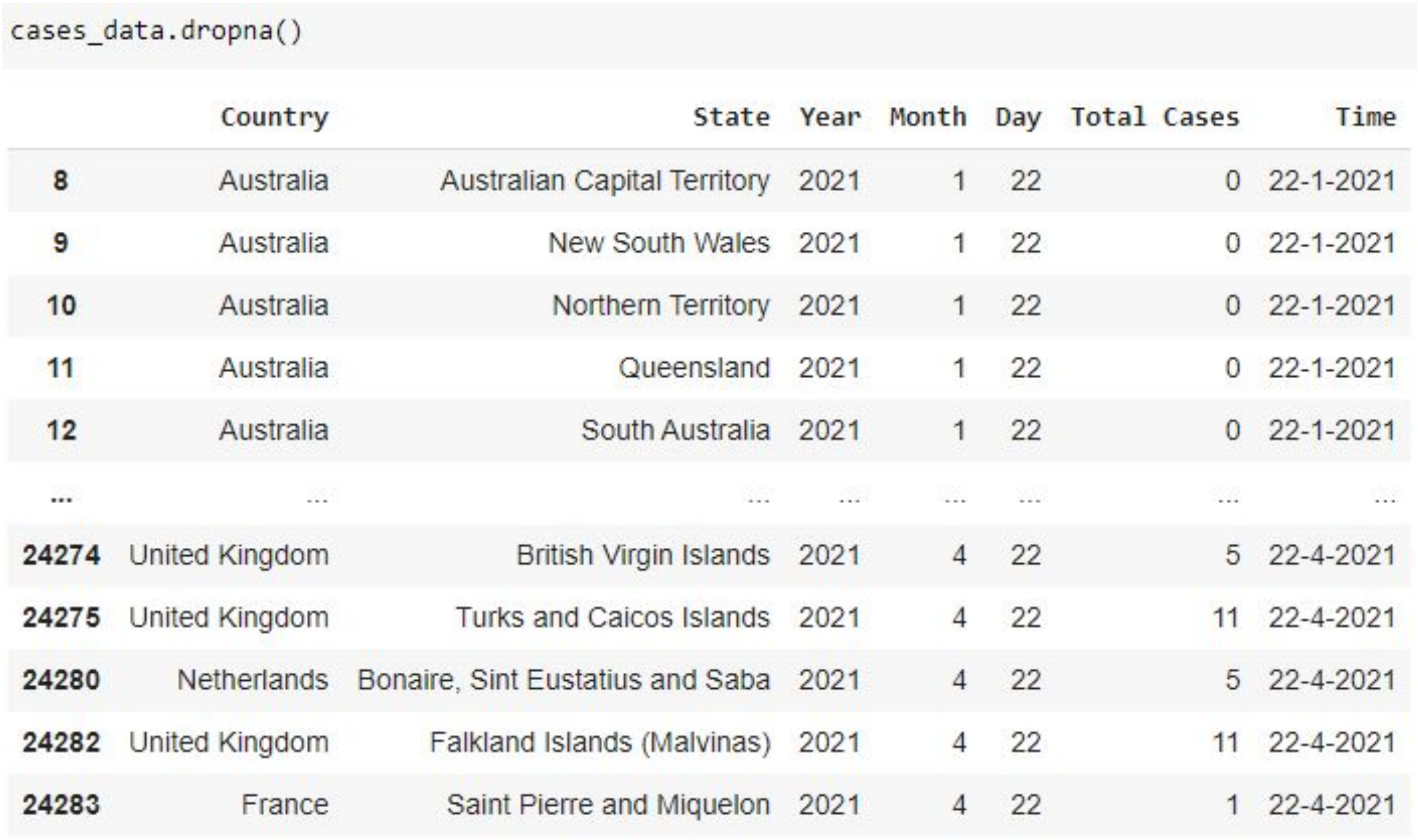
Handling Missing Data

There are two ways to handle missing data (usually denoted as NaN in a DataFrame)

- Remove it!
- Replace with a placeholder/dummy value. This is called **data imputation.**

Removing Data

- We can use **dropna()** method to remove rows that have missing data.
- For example, suppose we want to select cases data that has a state name. We have done this using notnull() on the state row before, but we can do it using dropna().

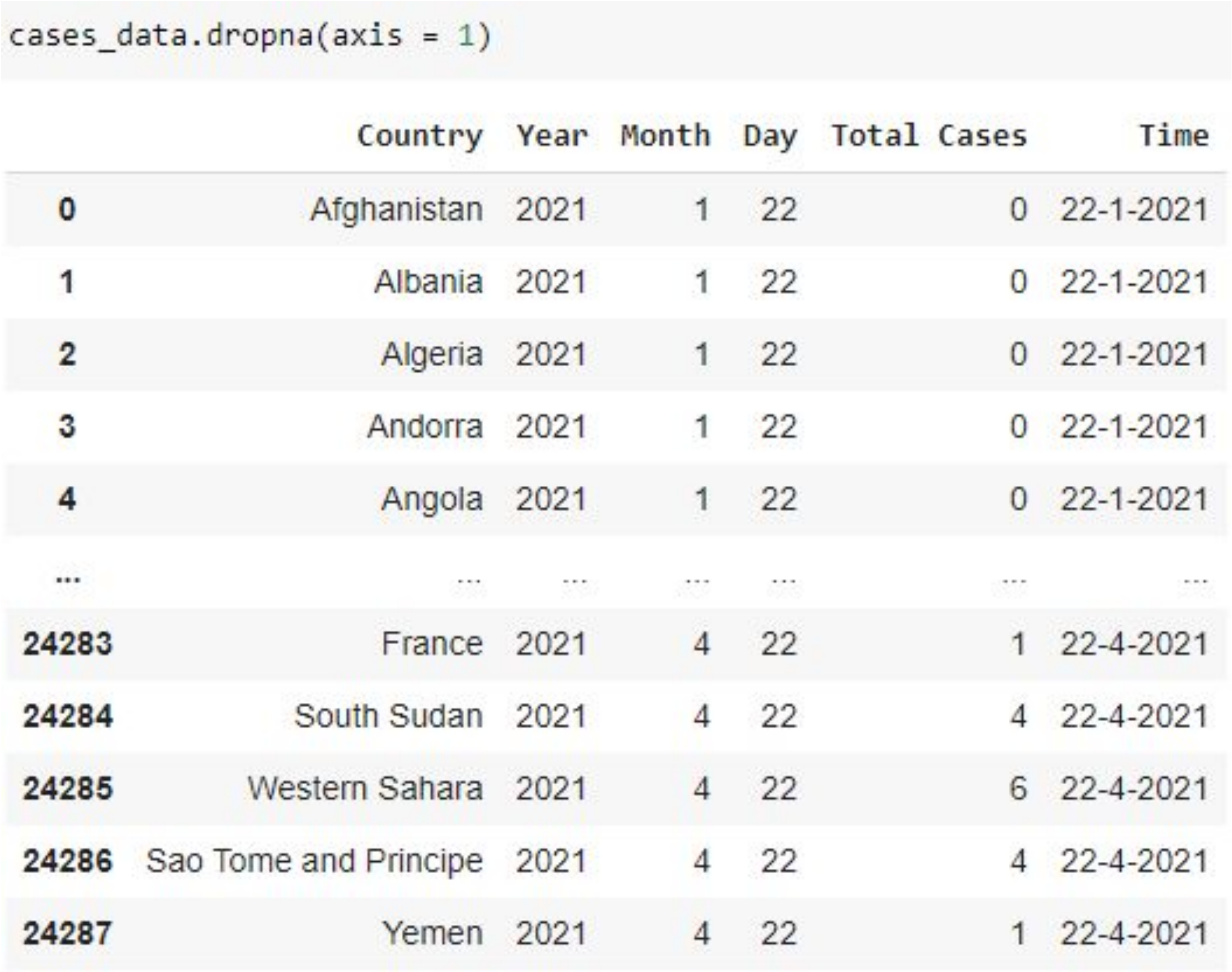


```
cases_data.dropna()
```

```
              Country                               State  Year  Month  Day  Total Cases       Time
    8       Australia         Australian Capital Territory  2021      1   22            0  22-1-2021
    9       Australia                     New South Wales  2021      1   22            0  22-1-2021
   10       Australia                  Northern Territory  2021      1   22            0  22-1-2021
   11       Australia                          Queensland  2021      1   22            0  22-1-2021
   12       Australia                     South Australia  2021      1   22            0  22-1-2021
  ...             ...                                 ...   ...    ...  ...          ...        ...
24274  United Kingdom              British Virgin Islands  2021      4   22            5  22-4-2021
24275  United Kingdom            Turks and Caicos Islands  2021      4   22           11  22-4-2021
24280     Netherlands  Bonaire, Sint Eustatius and Saba  2021      4   22            5  22-4-2021
24282  United Kingdom          Falkland Islands (Malvinas)  2021      4   22           11  22-4-2021
24283          France          Saint Pierre and Miquelon  2021      4   22            1  22-4-2021
```

Removing Data

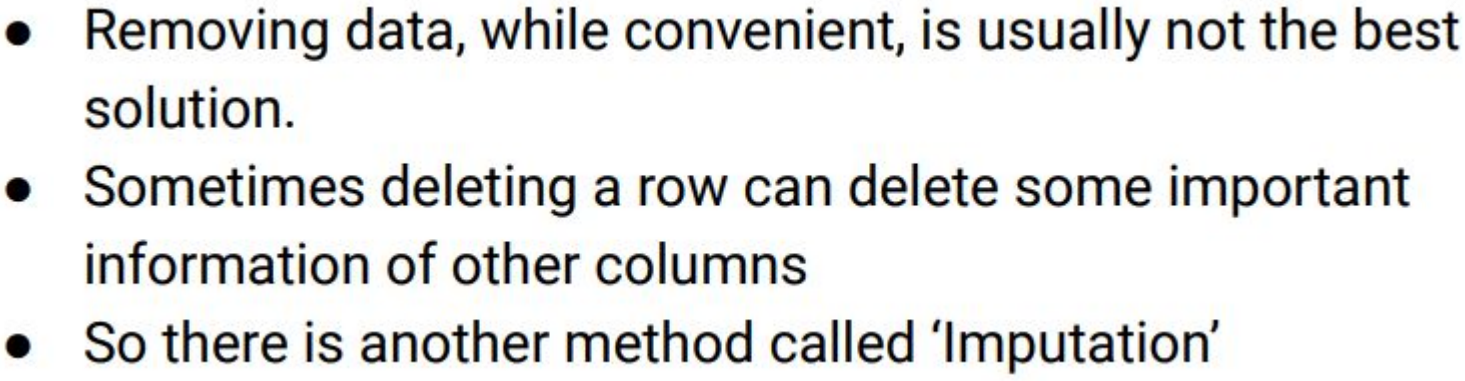
- We can also use **dropna()** method to remove columns that have missing data.
- For example, in this case, we use the axis parameter to drop the State column.



```
cases_data.dropna(axis = 1)
```

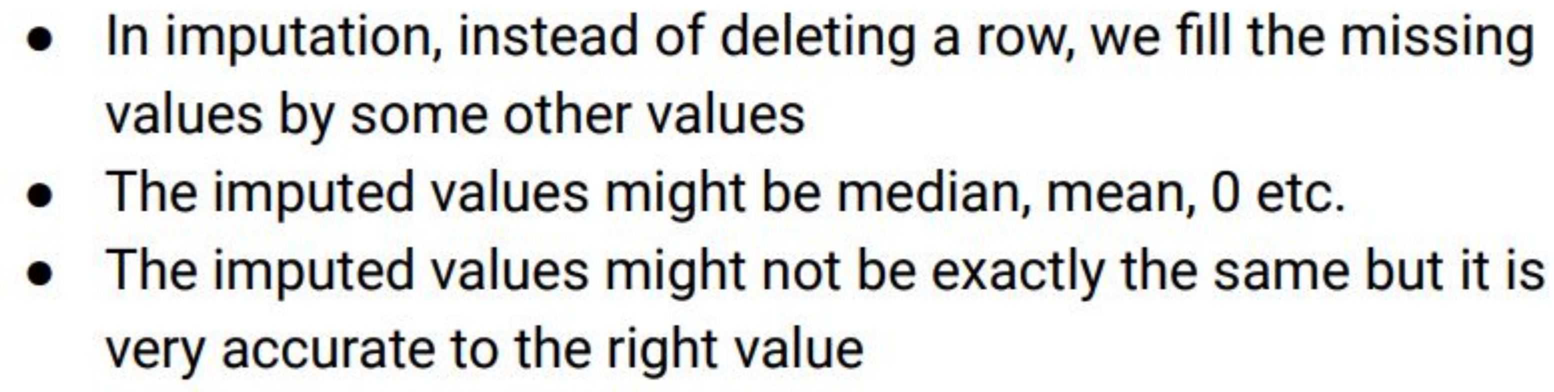
```
                      Country  Year  Month  Day  Total Cases       Time
    0             Afghanistan  2021      1   22            0  22-1-2021
    1                 Albania  2021      1   22            0  22-1-2021
    2                 Algeria  2021      1   22            0  22-1-2021
    3                 Andorra  2021      1   22            0  22-1-2021
    4                  Angola  2021      1   22            0  22-1-2021
  ...                     ...   ...    ...  ...          ...        ...
24283                  France  2021      4   22            1  22-4-2021
24284             South Sudan  2021      4   22            4  22-4-2021
24285          Western Sahara  2021      4   22            6  22-4-2021
24286   Sao Tome and Principe  2021      4   22            4  22-4-2021
24287                   Yemen  2021      4   22            1  22-4-2021
```

Removing Data



- Removing data, while convenient, is usually not the best solution.
- Sometimes deleting a row can delete some important information of other columns
- So there is another method called ‘Imputation’

Data Imputation



- In imputation, instead of deleting a row, we fill the missing values by some other values
- The imputed values might be median, mean, 0 etc.
- The imputed values might not be exactly the same but it is very accurate to the right value

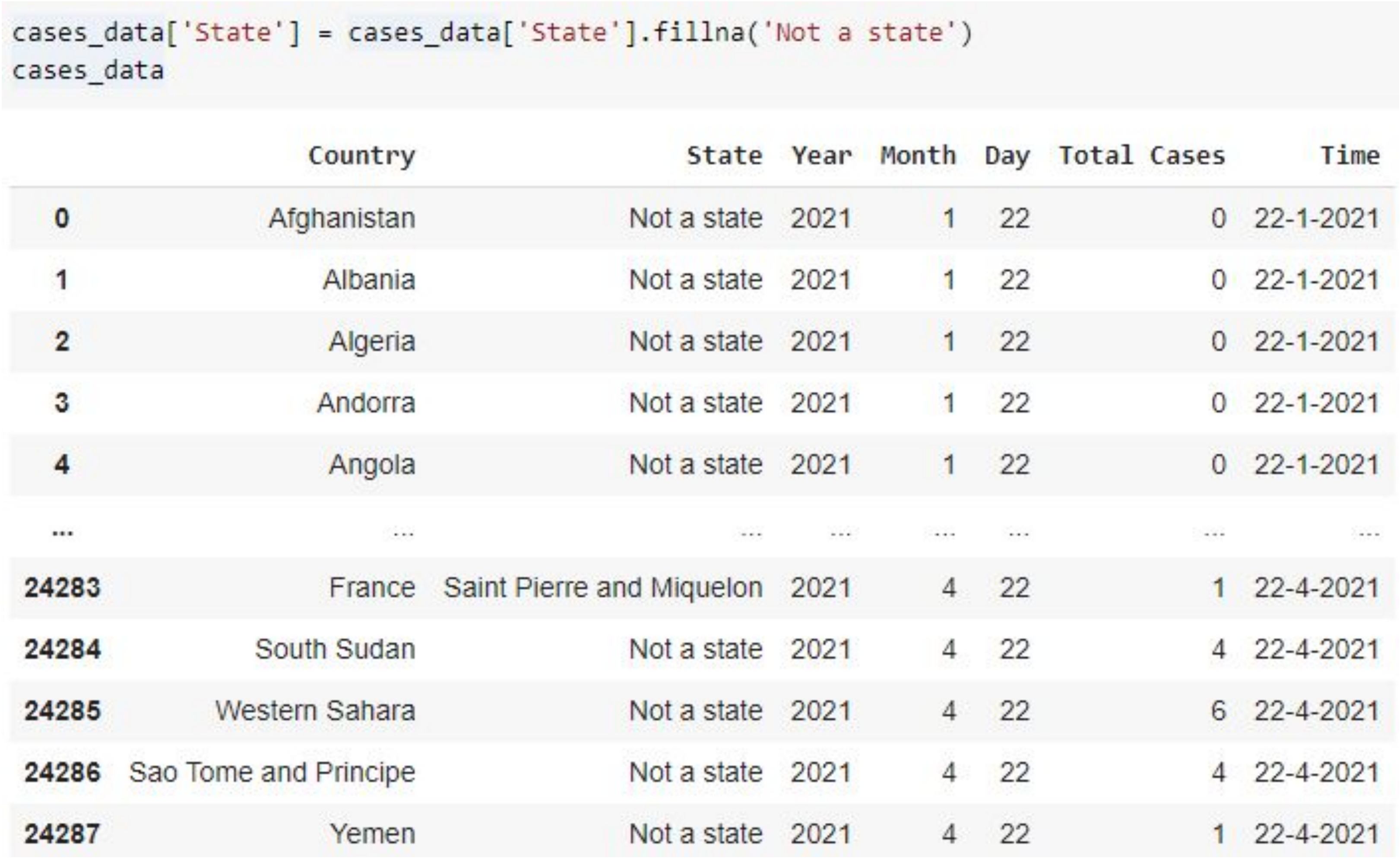
Data Imputation

To do so, Pandas provides us a function fillna() that can perform data imputation.

For example, instead of NaN, we can fill the data with missing states with ‘None’. This method will return a new Series/DataFrame, depends on our input.

```
cases_data[‘State’].fillna(‘Not a state’)
```

```
data.fillna(0)
```



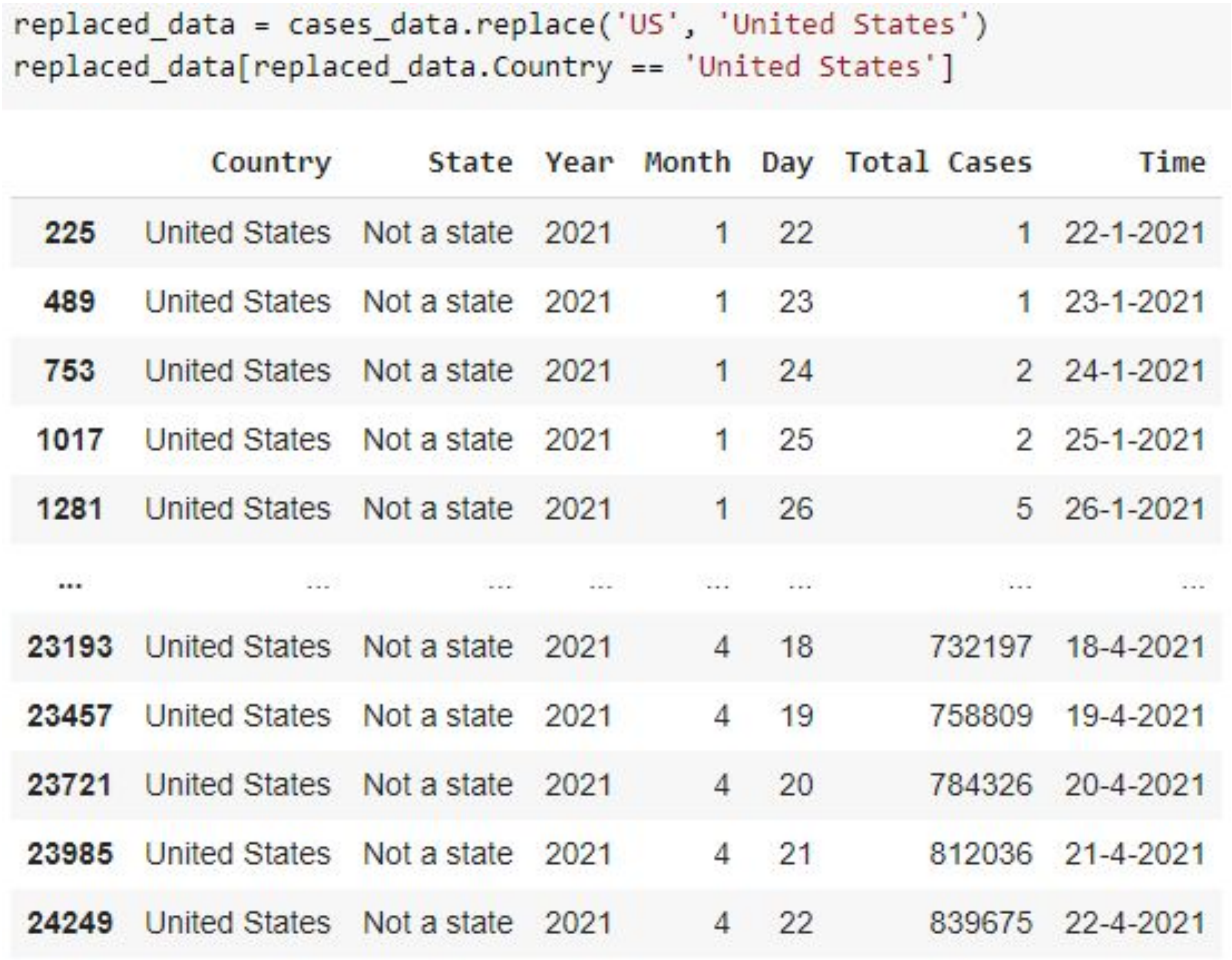
```
cases_data['State'] = cases_data['State'].fillna('Not a state')
cases_data
```

	Country	State	Year	Month	Day	Total Cases	Time
0	Afghanistan	Not a state	2021	1	22	0	22-1-2021
1	Albania	Not a state	2021	1	22	0	22-1-2021
2	Algeria	Not a state	2021	1	22	0	22-1-2021
3	Andorra	Not a state	2021	1	22	0	22-1-2021
4	Angola	Not a state	2021	1	22	0	22-1-2021
...	...	...	...	...	...	...	...
24283	France	Saint Pierre and Miquelon	2021	4	22	1	22-4-2021
24284	South Sudan	Not a state	2021	4	22	4	22-4-2021
24285	Western Sahara	Not a state	2021	4	22	6	22-4-2021
24286	Sao Tome and Principe	Not a state	2021	4	22	4	22-4-2021
24287	Yemen	Not a state	2021	4	22	1	22-4-2021

Data Imputation

We can also use imputation to replace an invalid data other than NaN with another value. This time we are using the **replace()** method.

```
cases_data.replace(‘US’, ‘United States’)
```



```
replaced_data = cases_data.replace('US', 'United States')
replaced_data[replaced_data.Country == 'United States']
```

	Country	State	Year	Month	Day	Total Cases	Time
225	United States	Not a state	2021	1	22	1	22-1-2021
489	United States	Not a state	2021	1	23	1	23-1-2021
753	United States	Not a state	2021	1	24	2	24-1-2021
1017	United States	Not a state	2021	1	25	2	25-1-2021
1281	United States	Not a state	2021	1	26	5	26-1-2021
...	...	...	...	...	...	...	...
23193	United States	Not a state	2021	4	18	732197	18-4-2021
23457	United States	Not a state	2021	4	19	758809	19-4-2021
23721	United States	Not a state	2021	4	20	784326	20-4-2021
23985	United States	Not a state	2021	4	21	812036	21-4-2021
24249	United States	Not a state	2021	4	22	839675	22-4-2021

Practice Part 8

None! Take a break for 5 minutes!

Outputting Data

Of course, once our data is ready, we need to save it into a new file. Pandas provides a method that does this.

```
data.to_csv('new_data.csv', index = False)

data.to_excel('new_data.xlsx', sheet_name = 'Sheet 1')
```

Practice Part 9

Save your joined_data into a new file called ‘employee_data.csv’. Keep the indexes.

Final Practice

Take a look at

https://colab.research.google.com/drive/1Ytzy3sOdimPtg8qqHoCXUjB4no6r4eTx?usp=sharing (Workshop codes)

https://pandas.pydata.org/pandas-docs/dev/user_guide/10min.html (Further Reading)

QnA