BI

Pandas 2 – Data Preprocessing

Data analysis process

Data Understanding	• Descriptive statistics • Types of data (numerical/categorical)
Data Preprocessing	• **Basic operations** • Subset selection and **data consolidation** • Missing data handling
Calculation (Modeling)	• Basic calculation • Data aggregation
Data Visualization	• Univariate chart • Bivariate chart • Multivariate chart

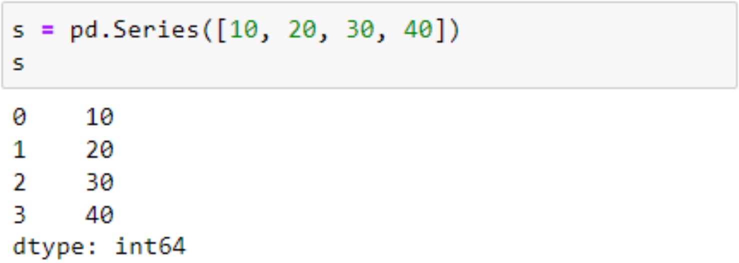
Outline

- Basic operations
  - Series: Add (or delete) a value
  - DataFrame: Add (or delete) a column (or row)
  - Copy a DataFrame
  - Sort a Dataframe

- Data consolidation
  - Concatenate
  - Merge

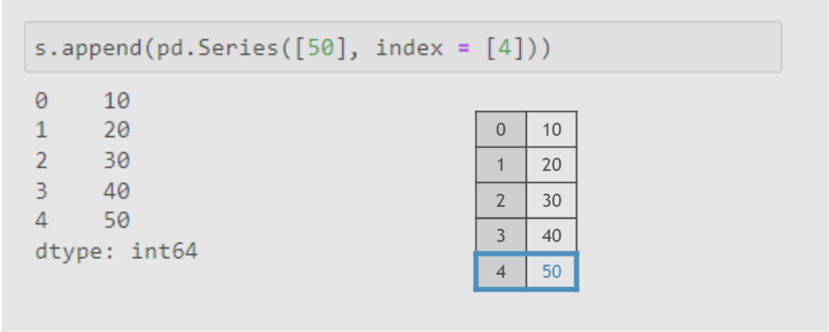
Series: Add or delete a value

- Add a value: Assign a new value to the given index
- Delete a value: Use `drop()` to delete the value at the given index.



```
s = pd.Series([10, 20, 30, 40])
s
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

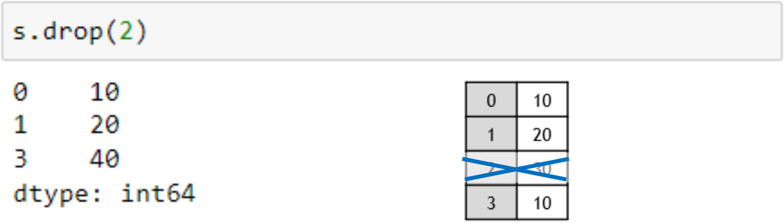


```
s.append(pd.Series([50], index = [4]))
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

0	10
1	20
2	30
3	40
4	50

The "append()" method has been deprecated since version 1.4.0. Use concat() instead.

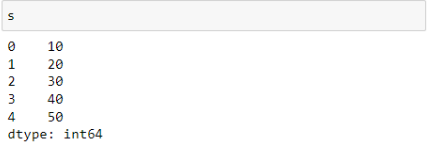


```
s.drop(2)
```

```
0    10
1    20
3    40
dtype: int64
```

0	10
1	20
~~2~~	~~30~~
3	10

By default, the parameter "inplace" of the drop method is set to "False". In this case, the series remains the same.

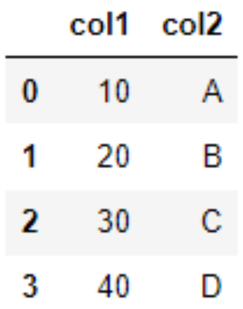


```
s
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

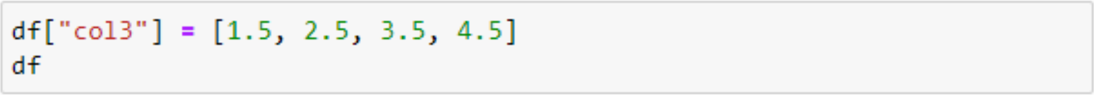
- If `inplace = True`, the change will be applied to the object directly.

DataFrame: Add a column/row

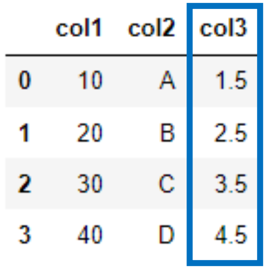


	col1	col2
0	10	A
1	20	B
2	30	C
3	40	D

- Add a new columns: Give a column name and a list of new values.

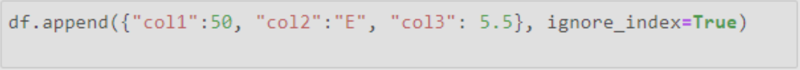


```
df["col3"] = [1.5, 2.5, 3.5, 4.5]
df
```

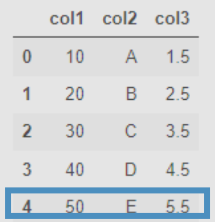


	col1	col2	col3
0	10	A	1.5
1	20	B	2.5
2	30	C	3.5
3	40	D	4.5

- Add a new row: Use append()



```
df.append({"col1":50, "col2":"E", "col3": 5.5}, ignore_index=True)
```

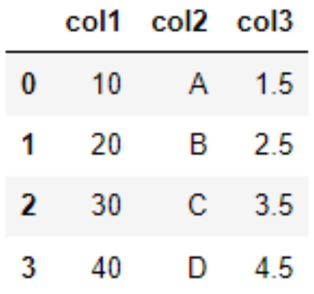


	col1	col2	col3
0	10	A	1.5
1	20	B	2.5
2	30	C	3.5
3	40	D	4.5
4	50	E	5.5

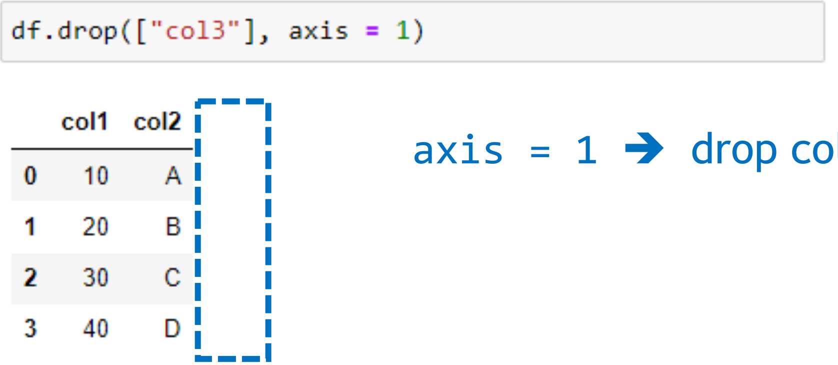
The "append()" method has been deprecated since version 1.4.0. Use concat() instead.

DataFrame: Delete a column or row

- Use `drop()` to remove column(s) or row(s).



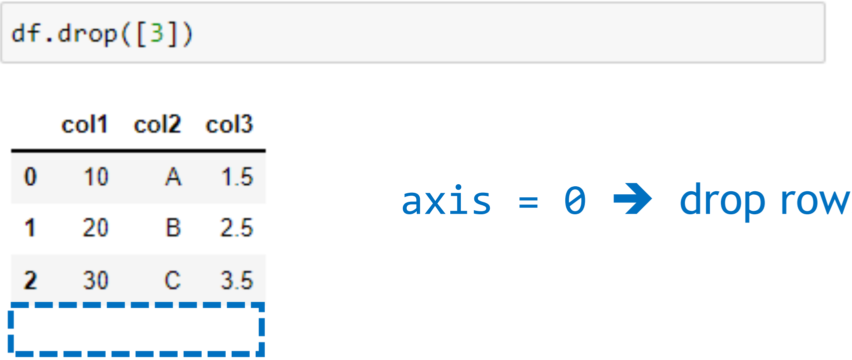
	col1	col2	col3
0	10	A	1.5
1	20	B	2.5
2	30	C	3.5
3	40	D	4.5



```
df.drop(["col3"], axis = 1)
```

	col1	col2
0	10	A
1	20	B
2	30	C
3	40	D

`axis = 1` → drop columns

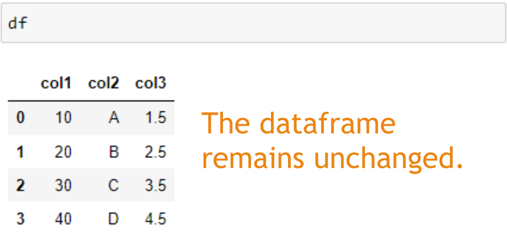


```
df.drop([3])
```

	col1	col2	col3
0	10	A	1.5
1	20	B	2.5
2	30	C	3.5

`axis = 0` → drop rows (default)

By default, inplace = False



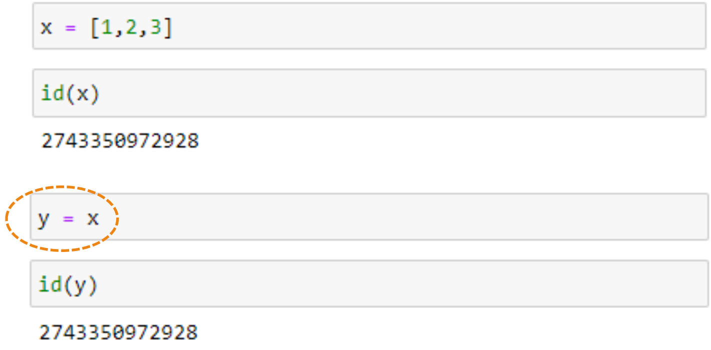
```
df
```

	col1	col2	col3
0	10	A	1.5
1	20	B	2.5
2	30	C	3.5
3	40	D	4.5

The dataframe remains unchanged.

Recall: Python variables and memory allocation

- If one variable is assigned to another variable, both variables point to the same memory address.



```
x = [1,2,3]
```

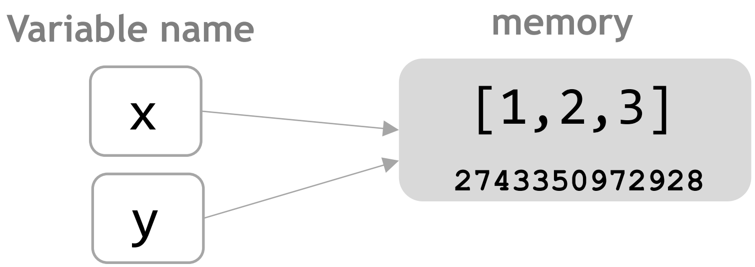
```
id(x)
```

```
2743350972928
```

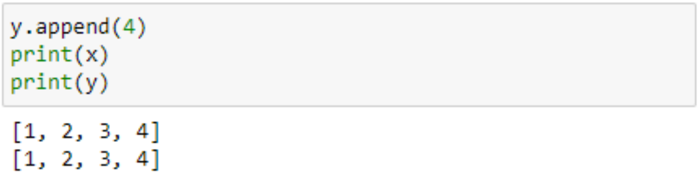
```
y = x
```

```
id(y)
```

```
2743350972928
```

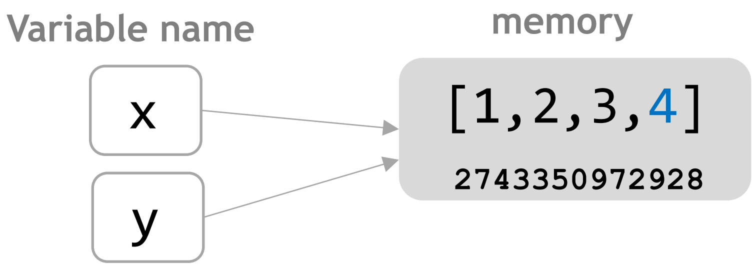


- If two variables point to the same address, changing the value of one variable will change the value of the other variable.



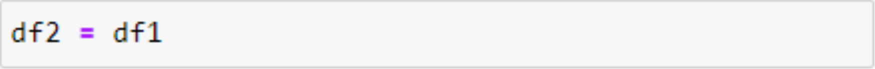
```
y.append(4)
print(x)
print(y)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```

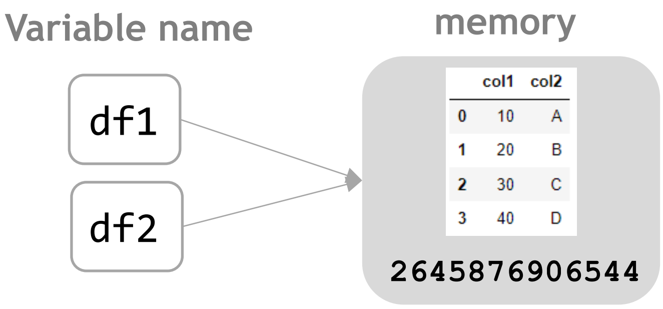


Create a copy of a DataFrame

- To avoid modifying the source DataFrame when manipulating the data, you can use copy() to create a copied DataFrame in advance.

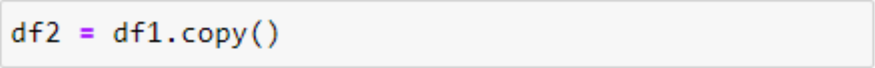


```
df2 = df1
```

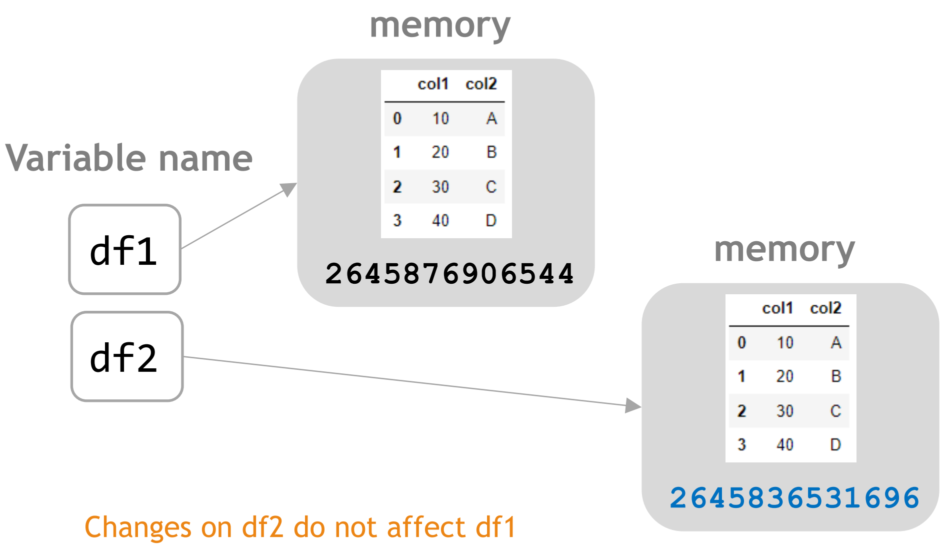


	col1	col2
0	10	A
1	20	B
2	30	C
3	40	D

Changes on df2 also affect df1.



```
df2 = df1.copy()
```

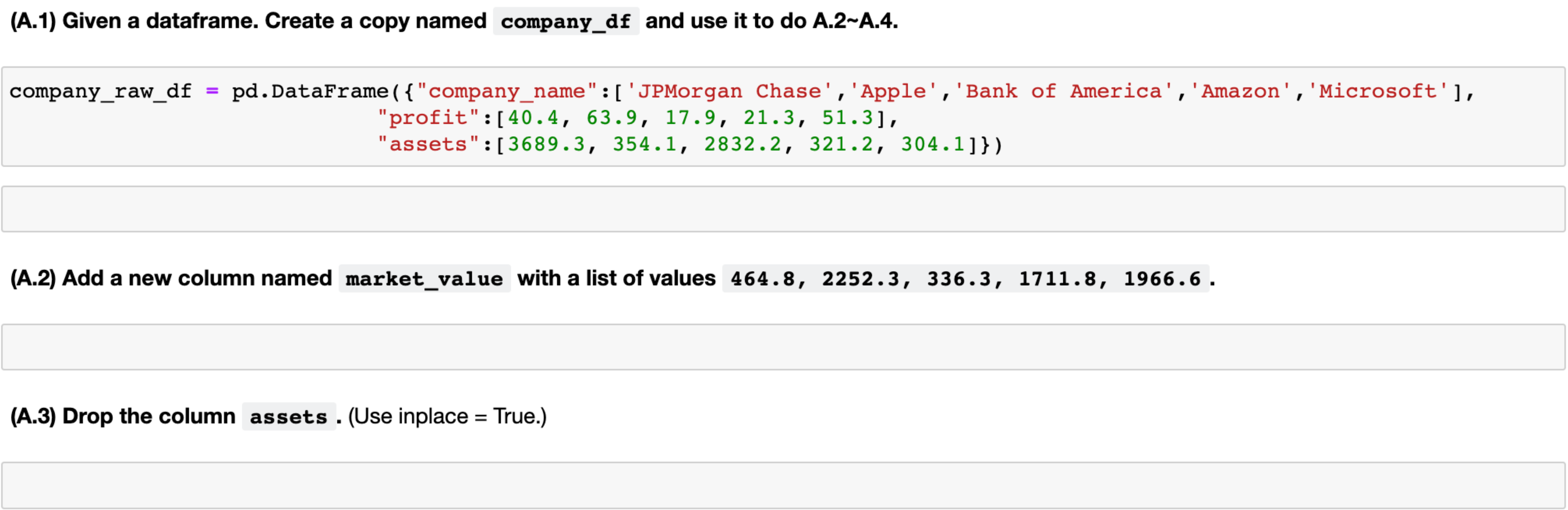


	col1	col2
0	10	A
1	20	B
2	30	C
3	40	D

	col1	col2
0	10	A
1	20	B
2	30	C
3	40	D

Changes on df2 do not affect df1

Exercise



(A.1) Given a dataframe. Create a copy named `company_df` and use it to do A.2~A.4.

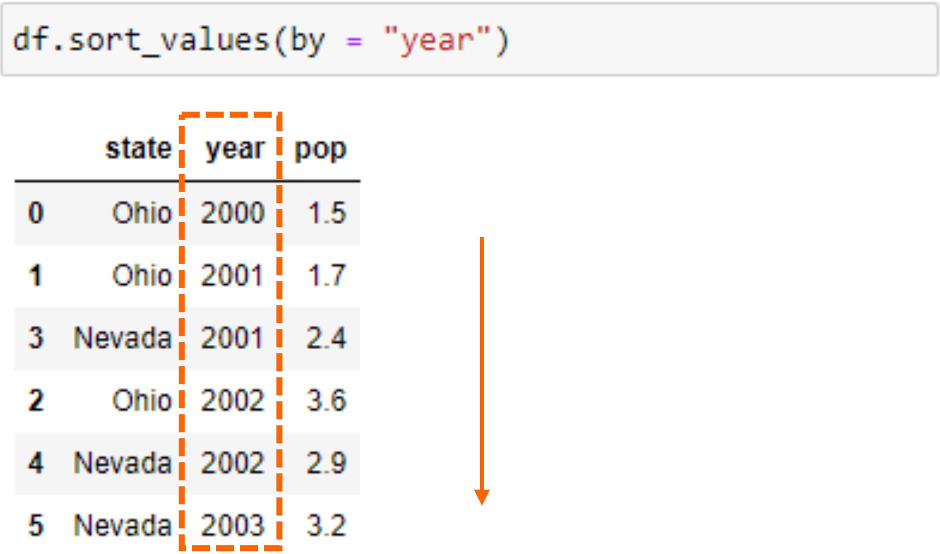
```
company_raw_df = pd.DataFrame({"company_name":['JPMorgan Chase','Apple','Bank of America','Amazon','Microsoft'],
                               "profit":[40.4, 63.9, 17.9, 21.3, 51.3],
                               "assets":[3689.3, 354.1, 2832.2, 321.2, 304.1]})
```

(A.2) Add a new column named `market_value` with a list of values `464.8, 2252.3, 336.3, 1711.8, 1966.6`.

(A.3) Drop the column `assets`. (Use inplace = True.)

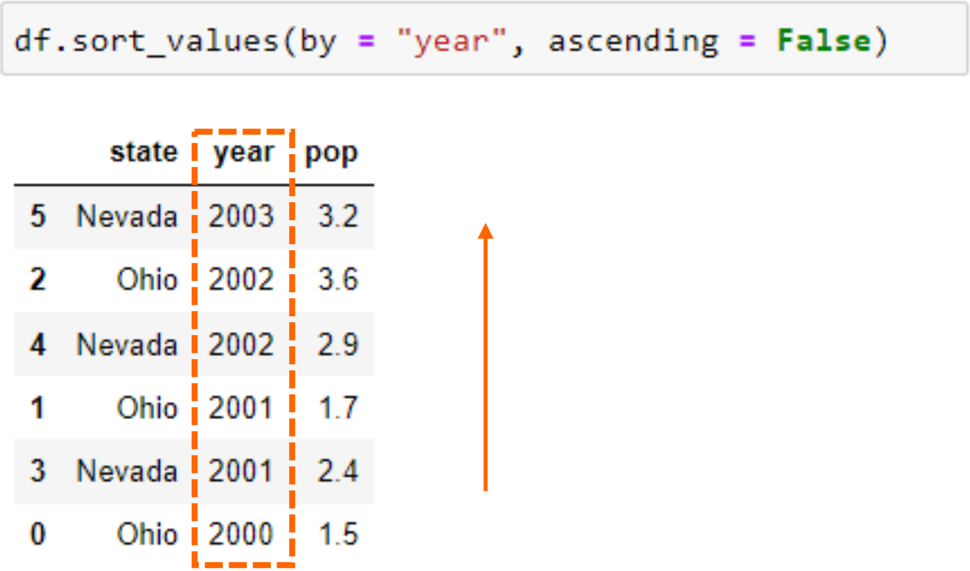
Sort a DataFrame

- Use the method `sort_values()` to sort a DataFrame.
- The data is sorted in ascending order by default but can be sorted in descending order by specifying `ascending = False`.



```
df.sort_values(by = "year")
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
3	Nevada	2001	2.4
2	Ohio	2002	3.6
4	Nevada	2002	2.9
5	Nevada	2003	3.2

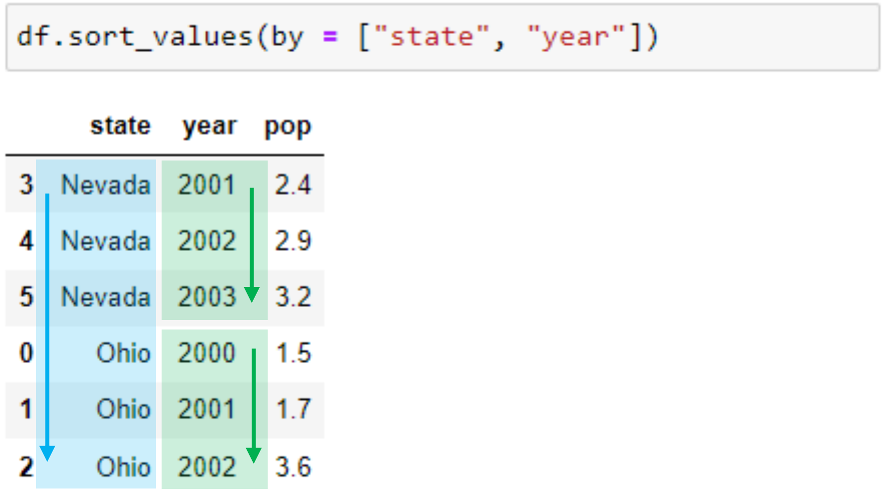


```
df.sort_values(by = "year", ascending = False)
```

	state	year	pop
5	Nevada	2003	3.2
2	Ohio	2002	3.6
4	Nevada	2002	2.9
1	Ohio	2001	1.7
3	Nevada	2001	2.4
0	Ohio	2000	1.5

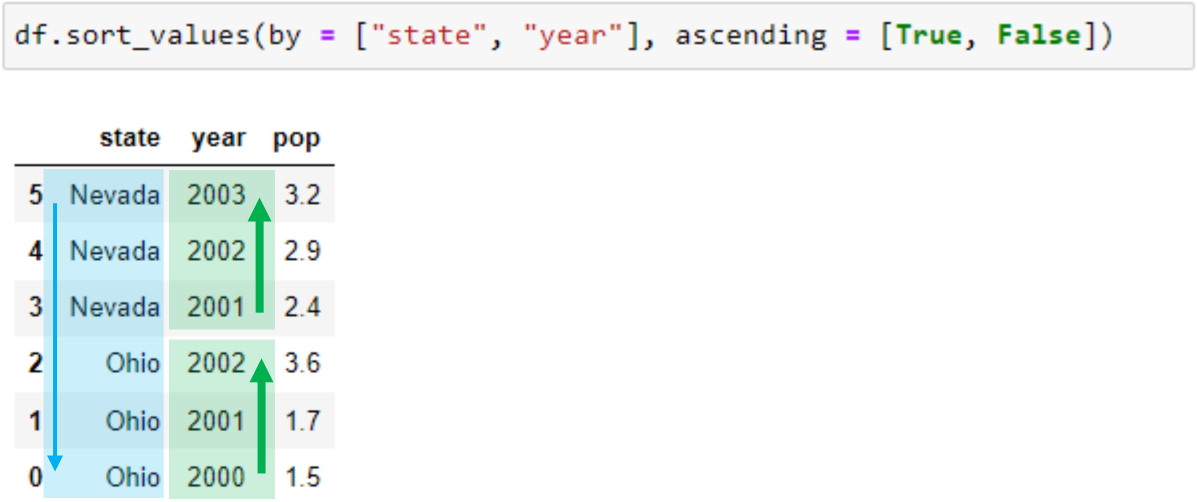
Sort a DataFrame – by multiple columns

- Pass a list of column names.



```python
df.sort_values(by = ["state", "year"])
```

	state	year	pop
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6

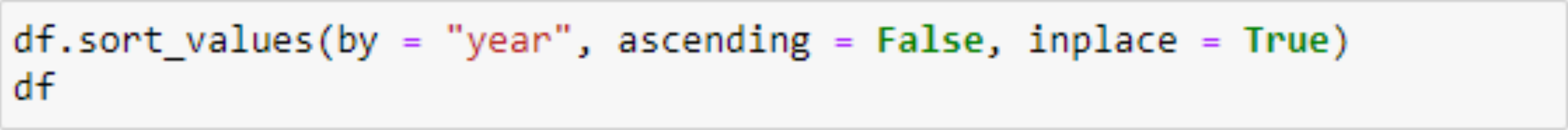


```python
df.sort_values(by = ["state", "year"], ascending = [True, False])
```

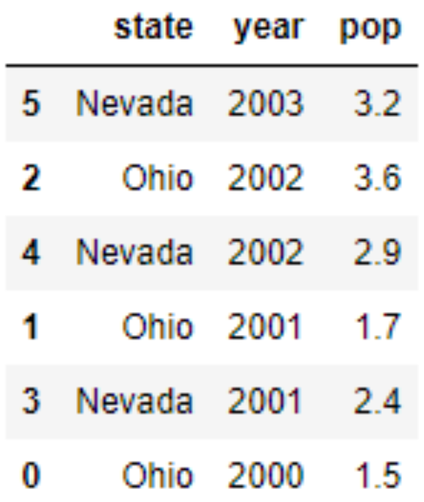
	state	year	pop
5	Nevada	2003	3.2
4	Nevada	2002	2.9
3	Nevada	2001	2.4
2	Ohio	2002	3.6
1	Ohio	2001	1.7
0	Ohio	2000	1.5

Sort a DataFrame - inplace

- If `inplace = True`, the change will be applied to the object directly.



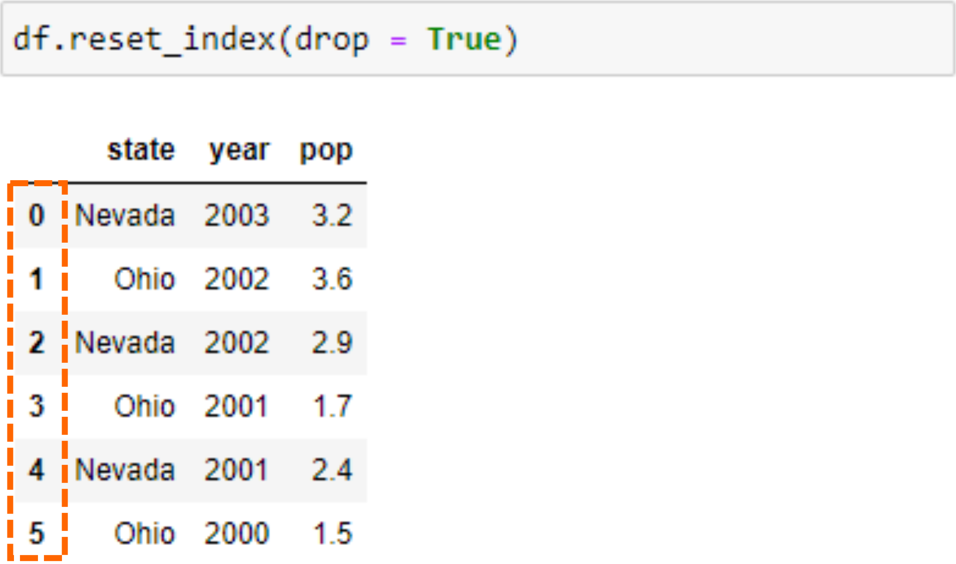
```
df.sort_values(by = "year", ascending = False, inplace = True)
df
```



	state	year	pop
5	Nevada	2003	3.2
2	Ohio	2002	3.6
4	Nevada	2002	2.9
1	Ohio	2001	1.7
3	Nevada	2001	2.4
0	Ohio	2000	1.5

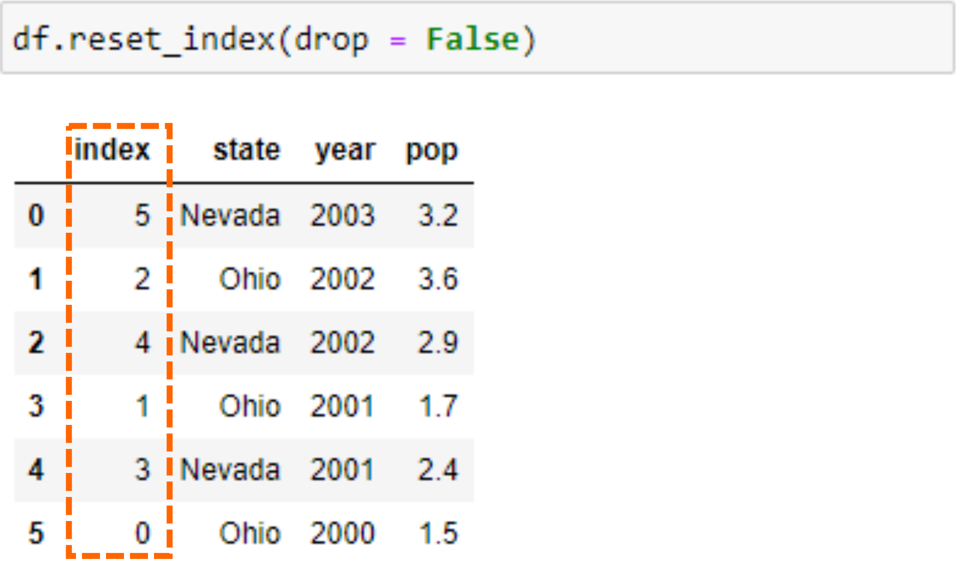
Reset index

- Reset the index of the DataFrame after sorting.
  - `drop = True`: Drop the old index.
  - `drop = False (default)`: Add the old index as an additional column to your DataFrame.



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

	state	year	pop
0	Nevada	2003	3.2
1	Ohio	2002	3.6
2	Nevada	2002	2.9
3	Ohio	2001	1.7
4	Nevada	2001	2.4
5	Ohio	2000	1.5

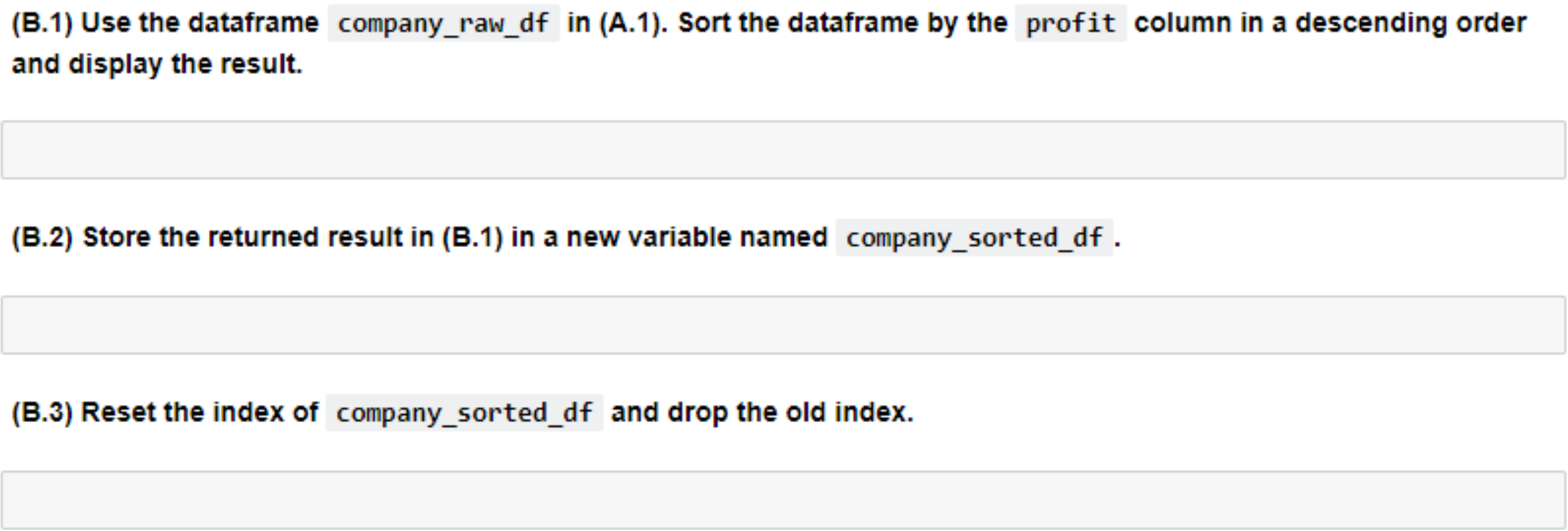


```
df.reset_index(drop = False)
```

	index	state	year	pop
0	5	Nevada	2003	3.2
1	2	Ohio	2002	3.6
2	4	Nevada	2002	2.9
3	1	Ohio	2001	1.7
4	3	Nevada	2001	2.4
5	0	Ohio	2000	1.5

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

Exercise



(B.1) Use the dataframe `company_raw_df` in (A.1). Sort the dataframe by the `profit` column in a descending order and display the result.

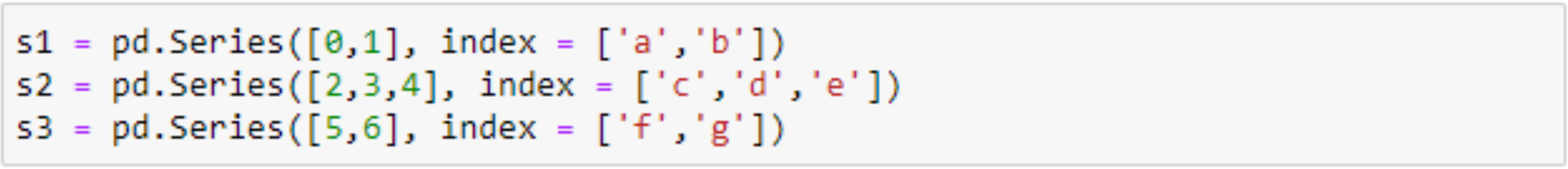
(B.2) Store the returned result in (B.1) in a new variable named `company_sorted_df`.

(B.3) Reset the index of `company_sorted_df` and drop the old index.

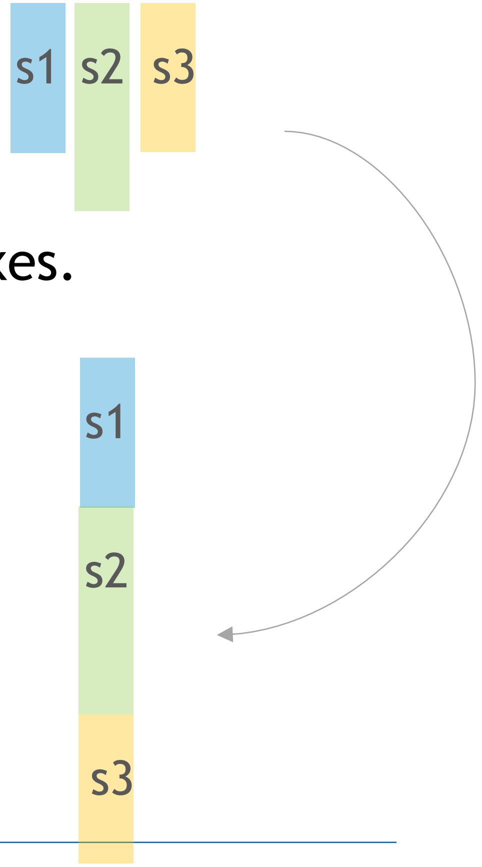
Data consolidation - Concatenate & Merge

Concatenate - Series

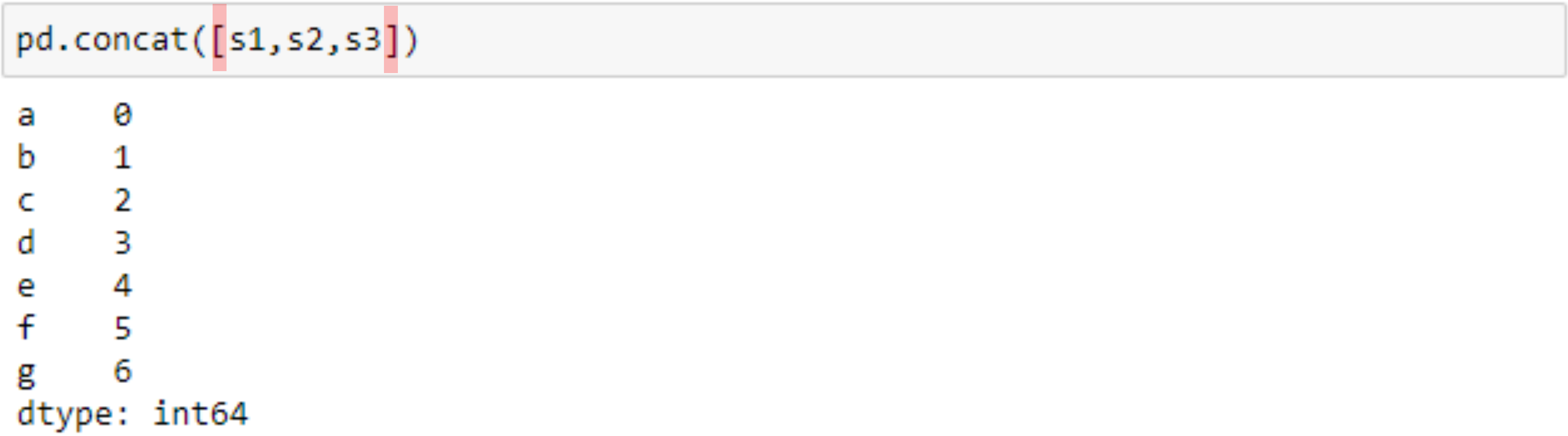
- Suppose you have three Series with no index overlap.



```
s1 = pd.Series([0,1], index = ['a','b'])
s2 = pd.Series([2,3,4], index = ['c','d','e'])
s3 = pd.Series([5,6], index = ['f','g'])
```



- Use concat() with these series in a list glues together the values and indexes.



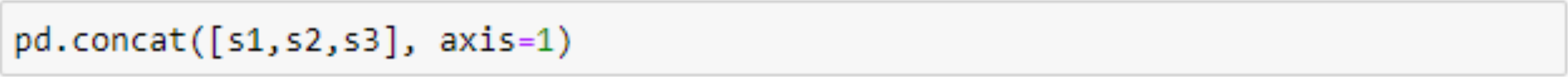
```
pd.concat([s1,s2,s3])
```

```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

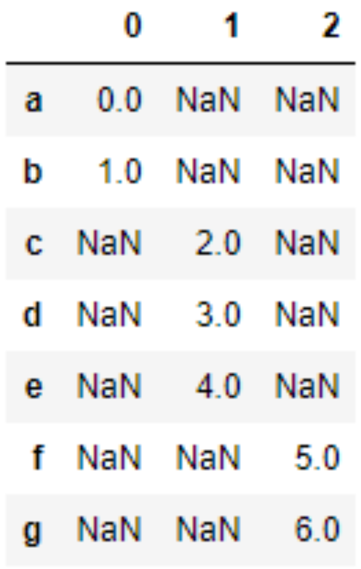
https://pandas.pydata.org/docs/reference/api/pandas.concat.html?highlight=concat

Concatenate - Series

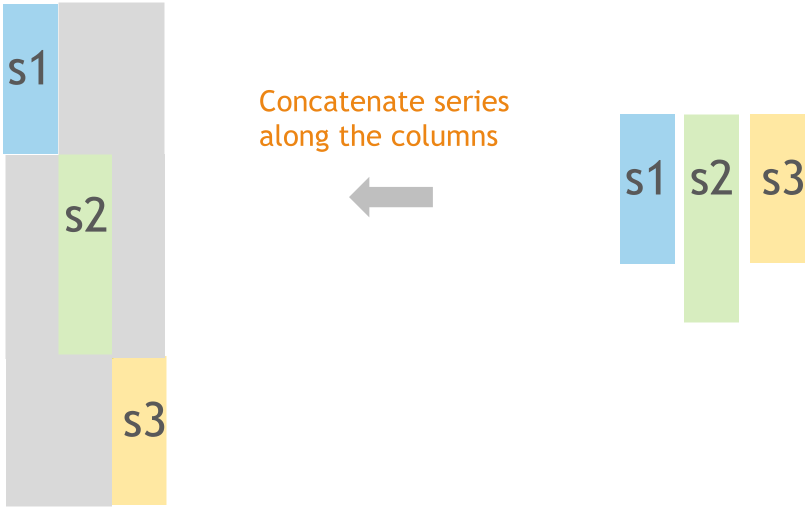
- By default, `concat()` works along `axis = 0`, producing another Series.
- If you pass `axis = 1`, the result will instead be a DataFrame.



```
pd.concat([s1,s2,s3], axis=1)
```



	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

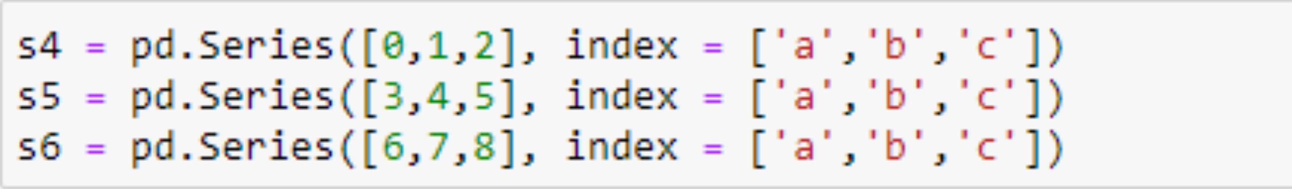


Concatenate series along the columns

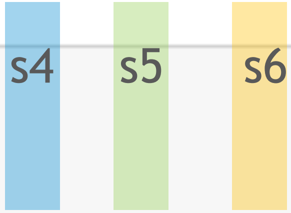
- Pandas uses **NaN** (Not a Number) to indicate missing data.

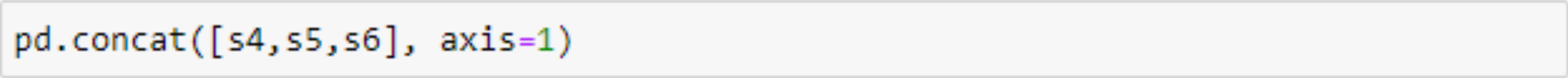
Concatenate - Series

- Suppose you have three Series with the same index.

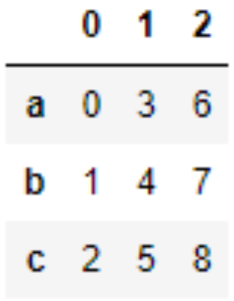


```
s4 = pd.Series([0,1,2], index = ['a','b','c'])
s5 = pd.Series([3,4,5], index = ['a','b','c'])
s6 = pd.Series([6,7,8], index = ['a','b','c'])
```

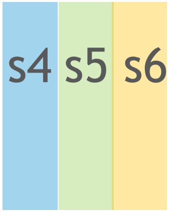




```
pd.concat([s4,s5,s6], axis=1)
```

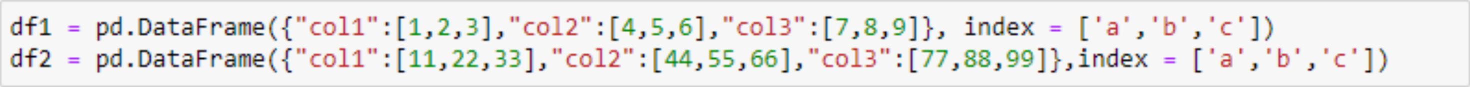


	0	1	2
a	0	3	6
b	1	4	7
c	2	5	8



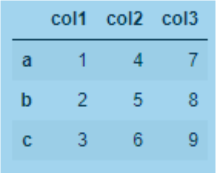
Concatenate - DataFrame

- Suppose you have two DatafFrames with the same index.



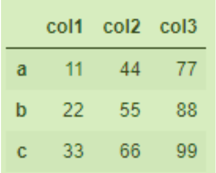
```python
df1 = pd.DataFrame({"col1":[1,2,3],"col2":[4,5,6],"col3":[7,8,9]}, index = ['a','b','c'])
df2 = pd.DataFrame({"col1":[11,22,33],"col2":[44,55,66],"col3":[77,88,99]},index = ['a','b','c'])
```

df1



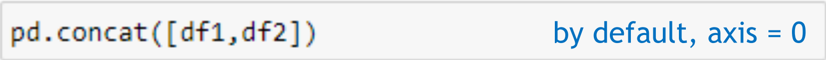
	col1	col2	col3
a	1	4	7
b	2	5	8
c	3	6	9

df2



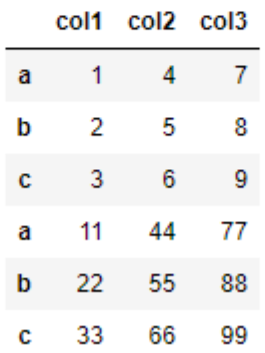
	col1	col2	col3
a	11	44	77
b	22	55	88
c	33	66	99

- Concatenate DataFrames

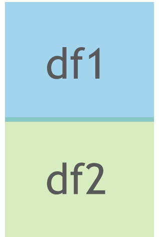


```python
pd.concat([df1,df2])
```

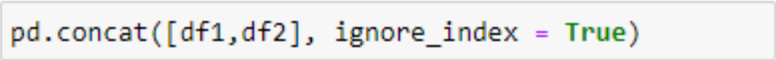
by default, axis = 0



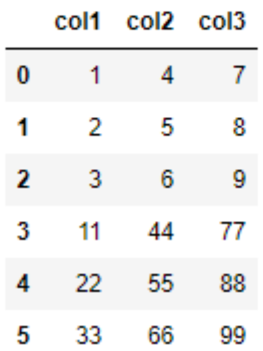
	col1	col2	col3
a	1	4	7
b	2	5	8
c	3	6	9
a	11	44	77
b	22	55	88
c	33	66	99



- Ignore index



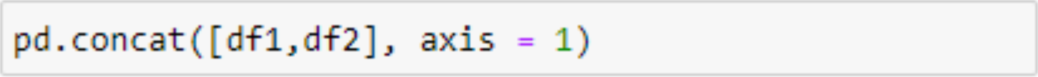
```python
pd.concat([df1,df2], ignore_index = True)
```



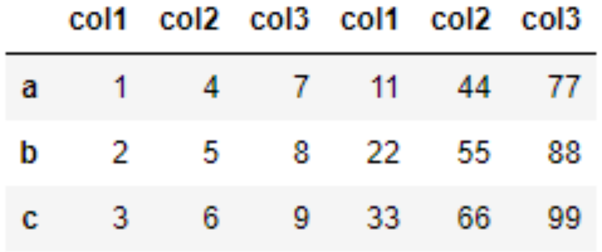
	col1	col2	col3
0	1	4	7
1	2	5	8
2	3	6	9
3	11	44	77
4	22	55	88
5	33	66	99

Concatenate - DataFrame

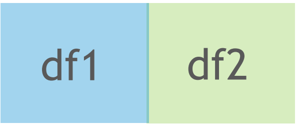
- If you pass `axis = 1`, df1 and df2 will be concatenated along the columns



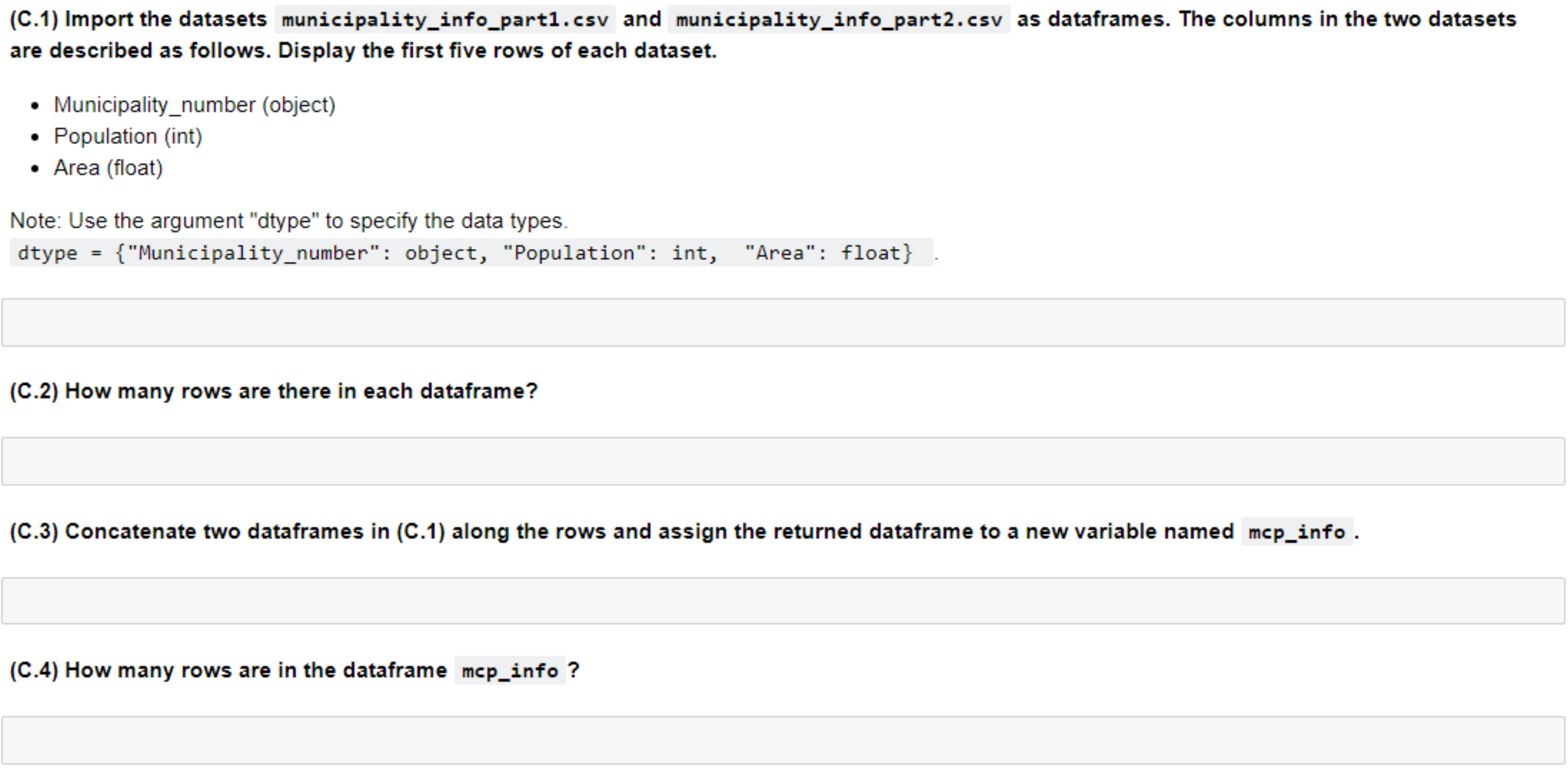
```
pd.concat([df1,df2], axis = 1)
```



	col1	col2	col3	col1	col2	col3
a	1	4	7	11	44	77
b	2	5	8	22	55	88
c	3	6	9	33	66	99



Exercise



(C.1) Import the datasets `municipality_info_part1.csv` and `municipality_info_part2.csv` as dataframes. The columns in the two datasets are described as follows. Display the first five rows of each dataset.

- Municipality_number (object)
- Population (int)
- Area (float)

Note: Use the argument "dtype" to specify the data types. `dtype = {"Municipality_number": object, "Population": int,  "Area": float}` .

(C.2) How many rows are there in each dataframe?

(C.3) Concatenate two dataframes in (C.1) along the rows and assign the returned dataframe to a new variable named `mcp_info` .

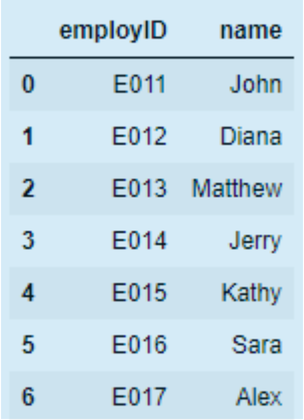
(C.4) How many rows are in the dataframe `mcp_info` ?

- Pandas read_csv https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

Merge – left join

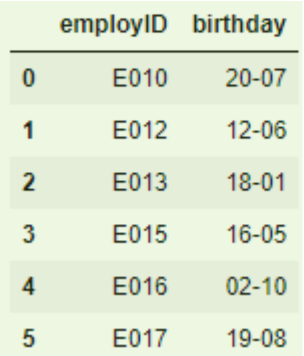
- merge(): Merge dataframes based on the common column (key).
- Left join: Use keys from left frame.

df1



	employID	name
0	E011	John
1	E012	Diana
2	E013	Matthew
3	E014	Jerry
4	E015	Kathy
5	E016	Sara
6	E017	Alex

df2

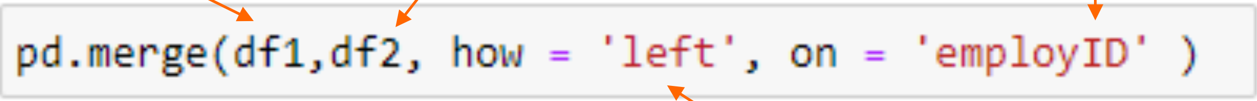


	employID	birthday
0	E010	20-07
1	E012	12-06
2	E013	18-01
3	E015	16-05
4	E016	02-10
5	E017	19-08

Left DataFrame

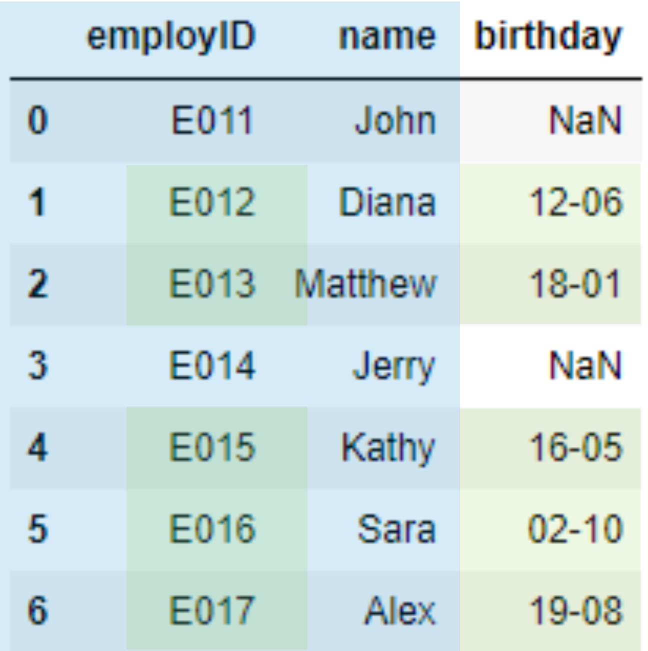
Right DataFrame

key



```
pd.merge(df1,df2, how = 'left', on = 'employID' )
```

To be merged on the left side



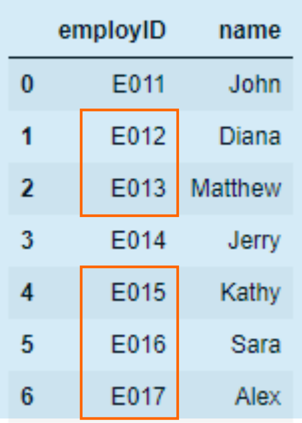
	employID	name	birthday
0	E011	John	NaN
1	E012	Diana	12-06
2	E013	Matthew	18-01
3	E014	Jerry	NaN
4	E015	Kathy	16-05
5	E016	Sara	02-10
6	E017	Alex	19-08

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html

Merge – inner join

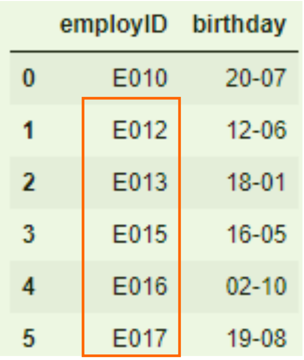
- Inner join: Use intersection of keys from both frames.

df1

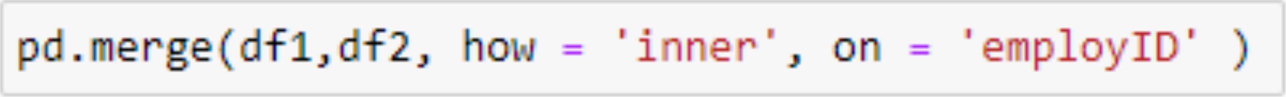


	employID	name
0	E011	John
1	E012	Diana
2	E013	Matthew
3	E014	Jerry
4	E015	Kathy
5	E016	Sara
6	E017	Alex

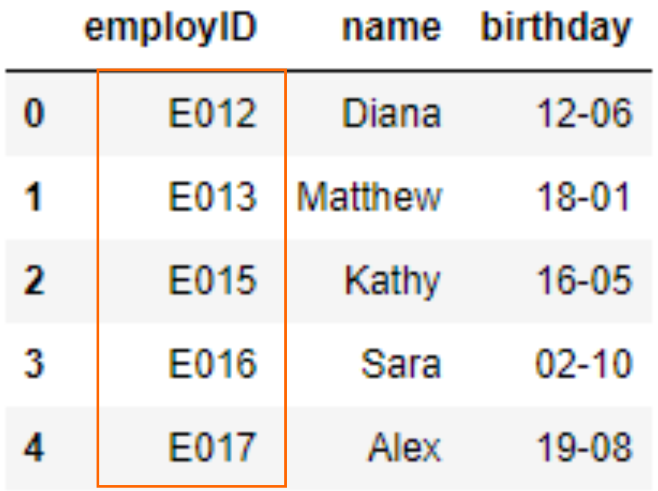
df2



	employID	birthday
0	E010	20-07
1	E012	12-06
2	E013	18-01
3	E015	16-05
4	E016	02-10
5	E017	19-08



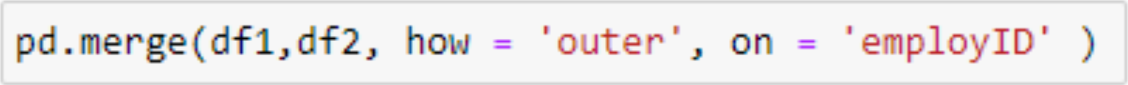
```
pd.merge(df1,df2, how = 'inner', on = 'employID' )
```



	employID	name	birthday
0	E012	Diana	12-06
1	E013	Matthew	18-01
2	E015	Kathy	16-05
3	E016	Sara	02-10
4	E017	Alex	19-08

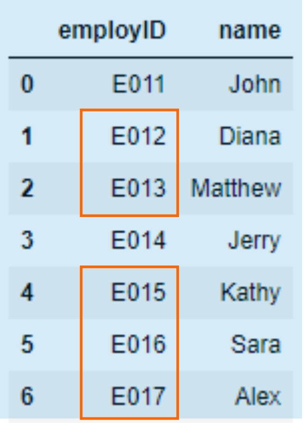
Merge – outer join

- Outer join: Use union of keys from both frames



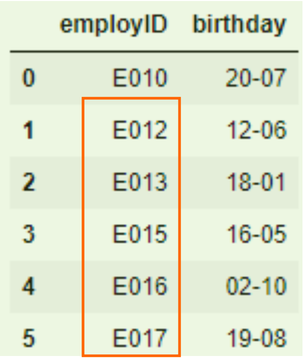
```
pd.merge(df1,df2, how = 'outer', on = 'employID' )
```

df1

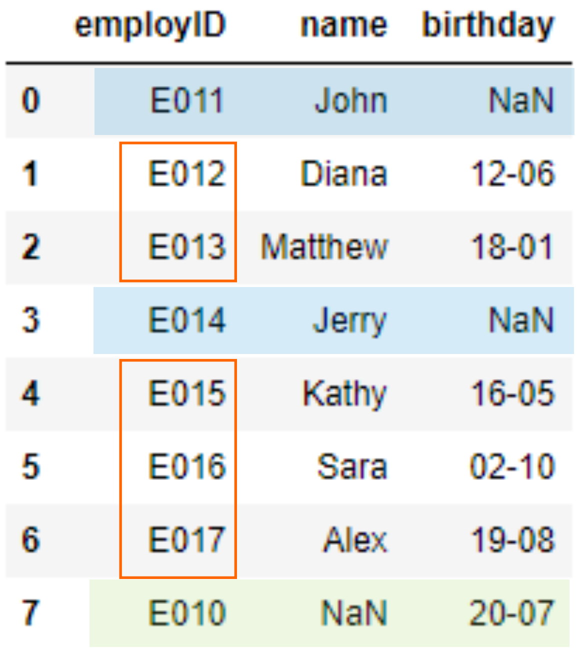


	employID	name
0	E011	John
1	E012	Diana
2	E013	Matthew
3	E014	Jerry
4	E015	Kathy
5	E016	Sara
6	E017	Alex

df2

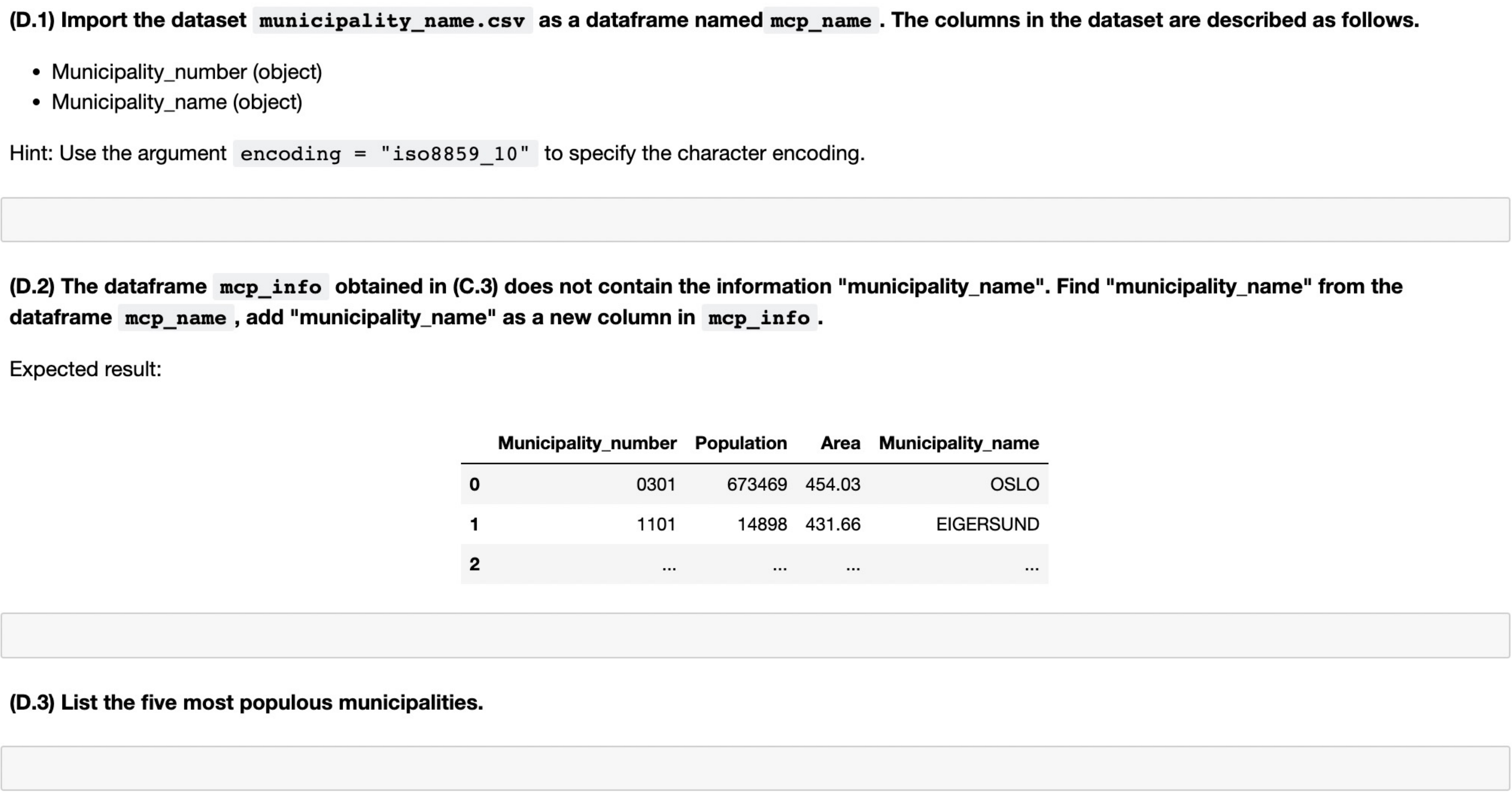


	employID	birthday
0	E010	20-07
1	E012	12-06
2	E013	18-01
3	E015	16-05
4	E016	02-10
5	E017	19-08



	employID	name	birthday
0	E011	John	NaN
1	E012	Diana	12-06
2	E013	Matthew	18-01
3	E014	Jerry	NaN
4	E015	Kathy	16-05
5	E016	Sara	02-10
6	E017	Alex	19-08
7	E010	NaN	20-07

Exercise



(D.1) Import the dataset `municipality_name.csv` as a dataframe named `mcp_name` . The columns in the dataset are described as follows.

- Municipality_number (object)
- Municipality_name (object)

Hint: Use the argument `encoding = "iso8859_10"` to specify the character encoding.

(D.2) The dataframe `mcp_info` obtained in (C.3) does not contain the information "municipality_name". Find "municipality_name" from the dataframe `mcp_name` , add "municipality_name" as a new column in `mcp_info` .

Expected result:

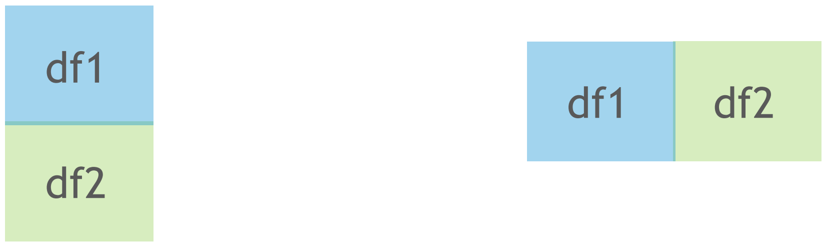
	Municipality_number	Population	Area	Municipality_name
0	0301	673469	454.03	OSLO
1	1101	14898	431.66	EIGERSUND
2	...	...	...	...

(D.3) List the five most populous municipalities.

- Standard encodings https://docs.python.org/3/library/codecs.html#standard-encodings

Differences between merge() and concat()

- concat() simply stacks multiple DataFrames together either vertically or horizontally.



- merge() first align the selected common columns of the two DataFrames, and then pick up the remaining columns from the aligned rows of each DataFrame.