Reference Material for Homework Assignment 2

Heidi Hyeseung Choi

HYSIS
heidichoi@hanyang.ac.kr

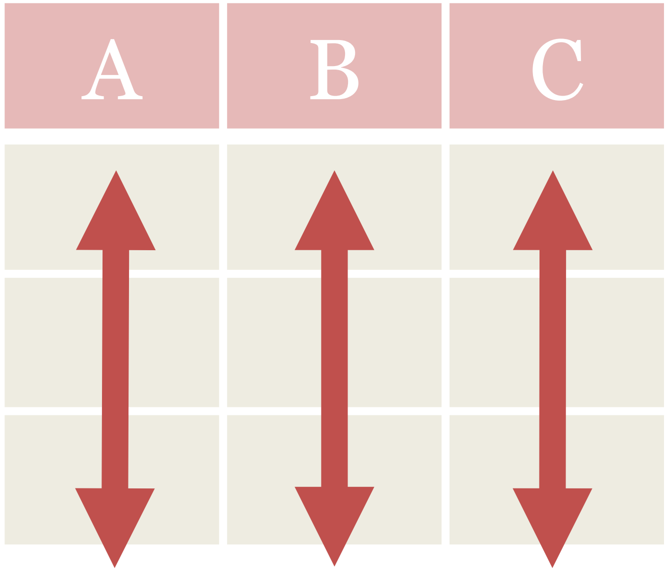
So why pandas?

- Pandas DataFrames: similar to lists and dictionaries, but it is more specialized in manipulating data.
- Lists and dictionaries do not have many methods to play around with data.

- There are so many things we can do with pandas.
- We can basically do everything you want with it.

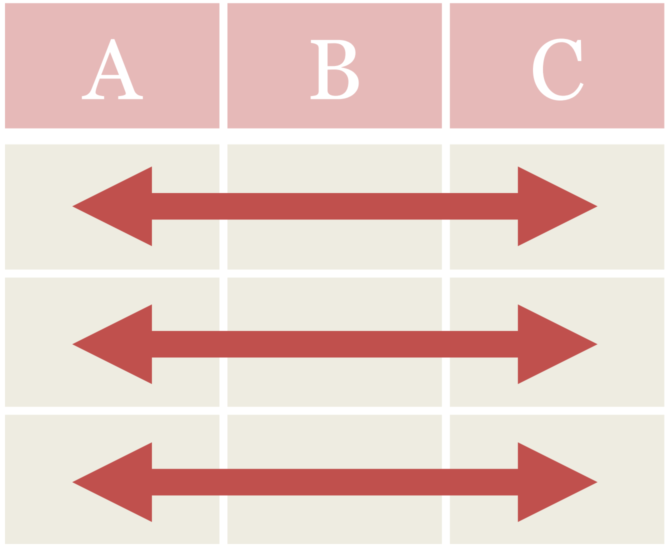
Tidy data

- In a **tidy data set**:
  - Each variable must have its own column.
  - Each observation must have its own row.
  - Each type of observational unit forms a table.



Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Messy data

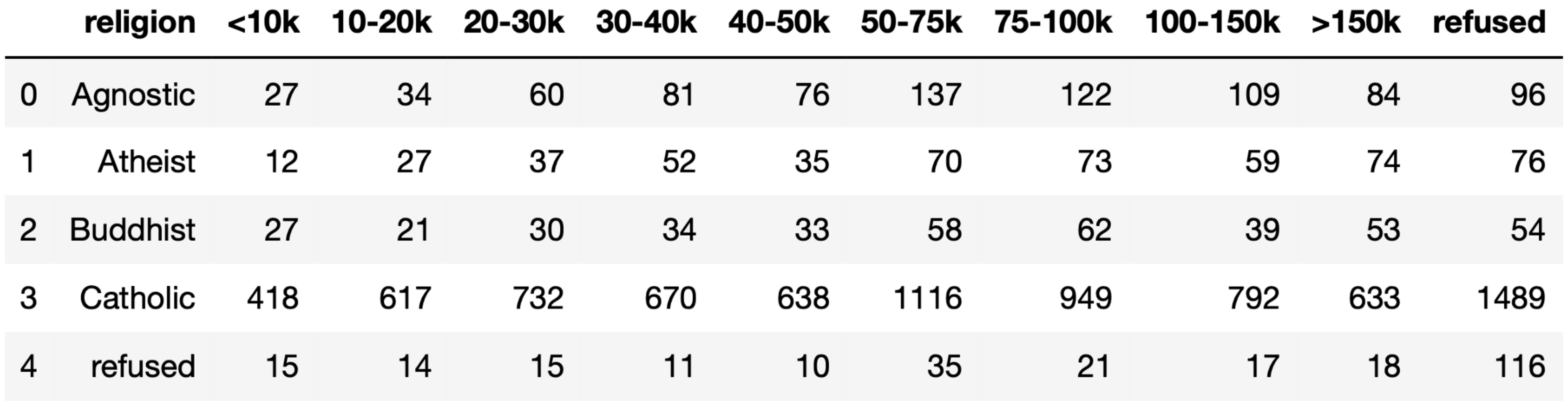
Three examples

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.

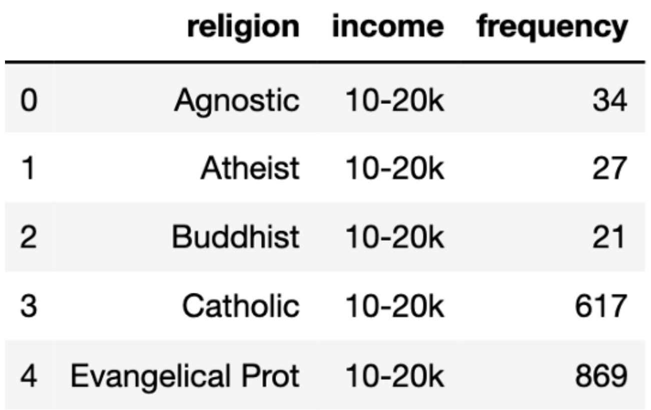
Source: https://towardsdatascience.com/whats-tidy-data-how-to-organize-messy-datasets-in-python-with-melt-and-pivotable-functions-5d52daa996c9

Messy data

- Column headers are values, not variable names.



	religion	<10k	10-20k	20-30k	30-40k	40-50k	50-75k	75-100k	100-150k	>150k	refused
0	Agnostic	27	34	60	81	76	137	122	109	84	96
1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	refused	15	14	15	11	10	35	21	17	18	116



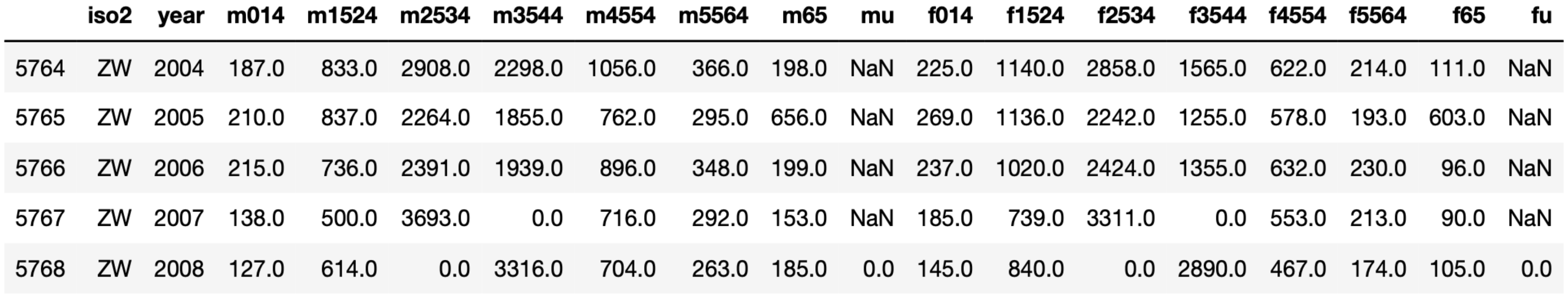
	religion	income	frequency
0	Agnostic	10-20k	34
1	Atheist	10-20k	27
2	Buddhist	10-20k	21
3	Catholic	10-20k	617
4	Evangelical Prot	10-20k	869

Source: https://towardsdatascience.com/whats-tidy-data-how-to-organize-messy-datasets-in-python-with-melt-and-pivotable-functions-5d52daa996c9

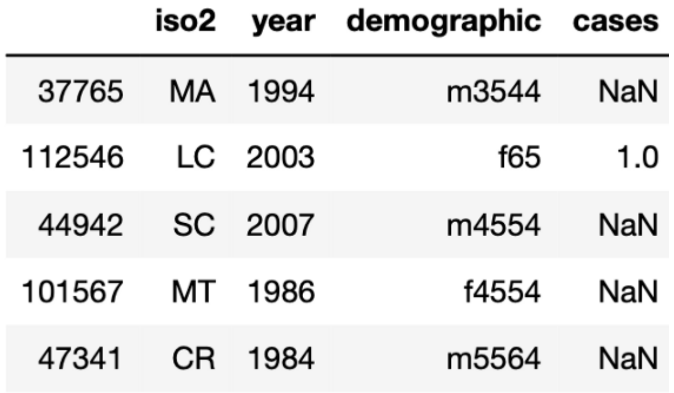
Messy data

- Multiple variables are stored in one column.

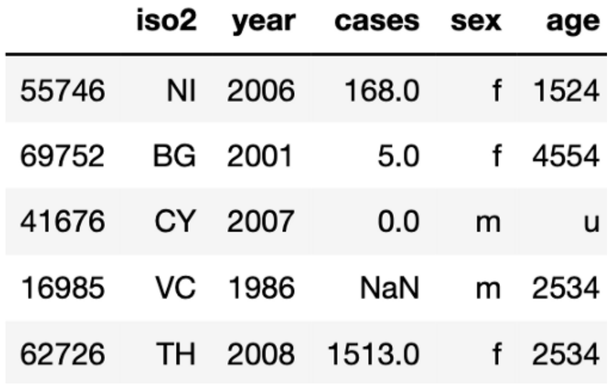
<the tuberculosis dataset from the World Health Organisation>



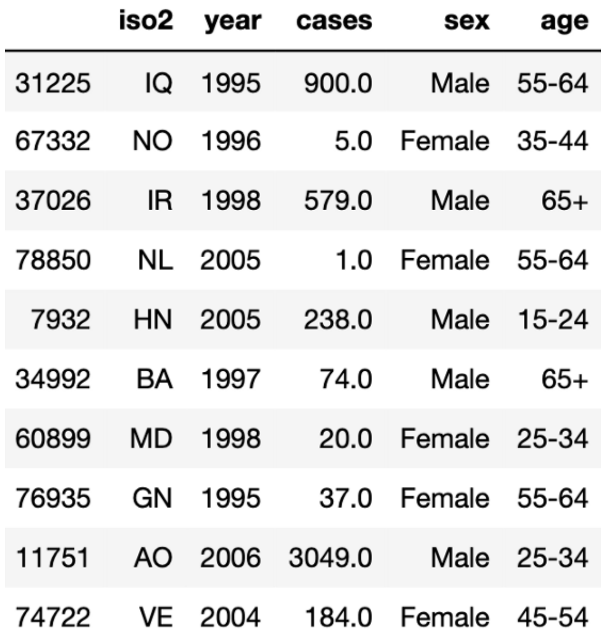
	iso2	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014	f1524	f2534	f3544	f4554	f5564	f65	fu
5764	ZW	2004	187.0	833.0	2908.0	2298.0	1056.0	366.0	198.0	NaN	225.0	1140.0	2858.0	1565.0	622.0	214.0	111.0	NaN
5765	ZW	2005	210.0	837.0	2264.0	1855.0	762.0	295.0	656.0	NaN	269.0	1136.0	2242.0	1255.0	578.0	193.0	603.0	NaN
5766	ZW	2006	215.0	736.0	2391.0	1939.0	896.0	348.0	199.0	NaN	237.0	1020.0	2424.0	1355.0	632.0	230.0	96.0	NaN
5767	ZW	2007	138.0	500.0	3693.0	0.0	716.0	292.0	153.0	NaN	185.0	739.0	3311.0	0.0	553.0	213.0	90.0	NaN
5768	ZW	2008	127.0	614.0	0.0	3316.0	704.0	263.0	185.0	0.0	145.0	840.0	0.0	2890.0	467.0	174.0	105.0	0.0



	iso2	year	demographic	cases
37765	MA	1994	m3544	NaN
112546	LC	2003	f65	1.0
44942	SC	2007	m4554	NaN
101567	MT	1986	f4554	NaN
47341	CR	1984	m5564	NaN



	iso2	year	cases	sex	age
55746	NI	2006	168.0	f	1524
69752	BG	2001	5.0	f	4554
41676	CY	2007	0.0	m	u
16985	VC	1986	NaN	m	2534
62726	TH	2008	1513.0	f	2534



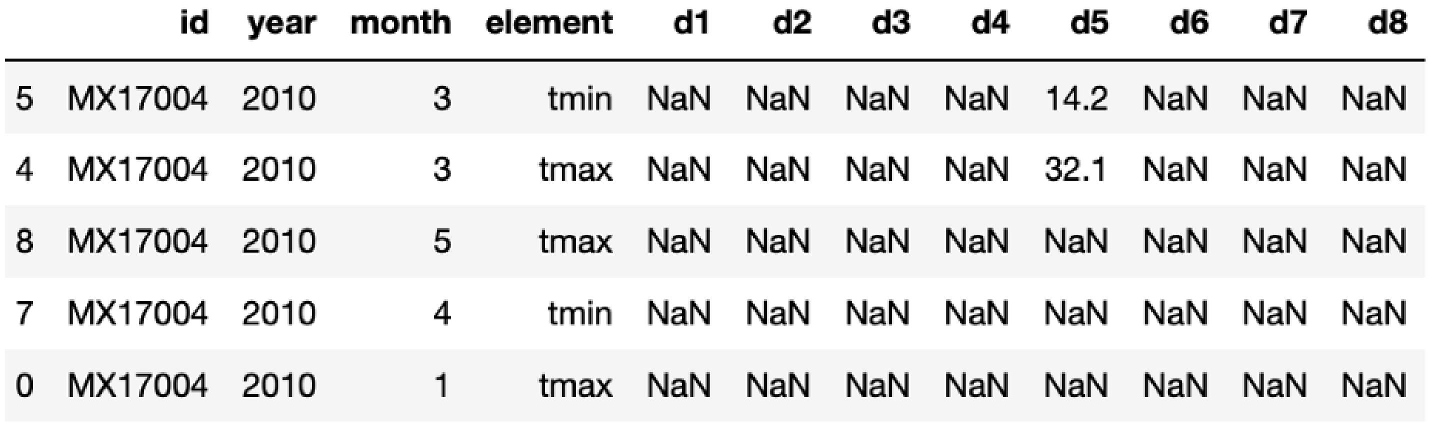
	iso2	year	cases	sex	age
31225	IQ	1995	900.0	Male	55-64
67332	NO	1996	5.0	Female	35-44
37026	IR	1998	579.0	Male	65+
78850	NL	2005	1.0	Female	55-64
7932	HN	2005	238.0	Male	15-24
34992	BA	1997	74.0	Male	65+
60899	MD	1998	20.0	Female	25-34
76935	GN	1995	37.0	Female	55-64
11751	AO	2006	3049.0	Male	25-34
74722	VE	2004	184.0	Female	45-54

Source: https://towardsdatascience.com/whats-tidy-data-how-to-organize-messy-datasets-in-python-with-melt-and-pivotable-functions-5d52daa996c9

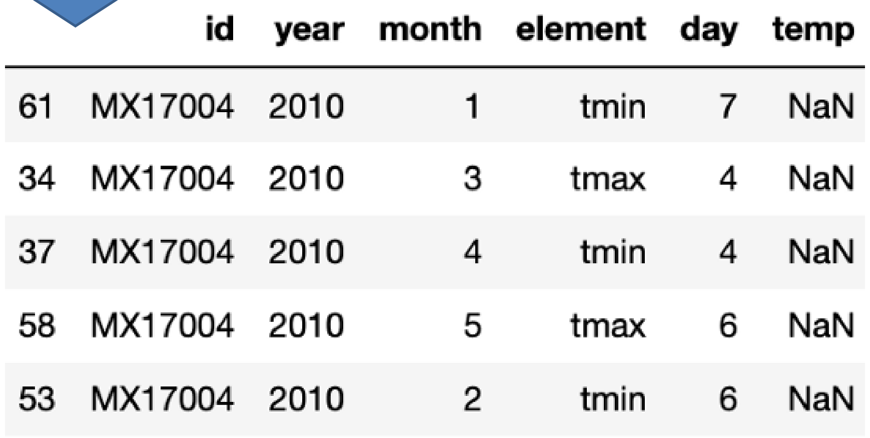
Messy data

- Variables are stored in both rows and columns.

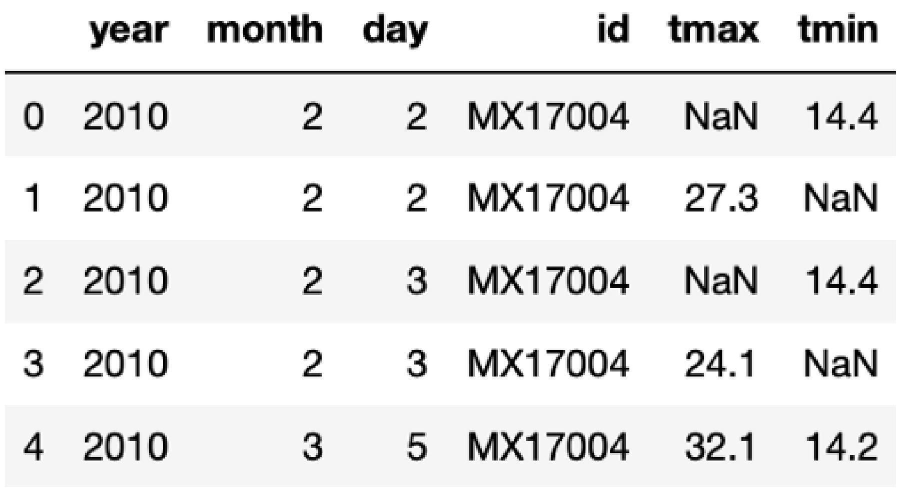
<Global Historical Climatology Network: the daily weather station (MX17004) in Mexico for five months in 2010>



	id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
5	MX17004	2010	3	tmin	NaN	NaN	NaN	NaN	14.2	NaN	NaN	NaN
4	MX17004	2010	3	tmax	NaN	NaN	NaN	NaN	32.1	NaN	NaN	NaN
8	MX17004	2010	5	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	MX17004	2010	4	tmin	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
0	MX17004	2010	1	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN



	id	year	month	element	day	temp
61	MX17004	2010	1	tmin	7	NaN
34	MX17004	2010	3	tmax	4	NaN
37	MX17004	2010	4	tmin	4	NaN
58	MX17004	2010	5	tmax	6	NaN
53	MX17004	2010	2	tmin	6	NaN



	year	month	day	id	tmax	tmin
0	2010	2	2	MX17004	NaN	14.4
1	2010	2	2	MX17004	27.3	NaN
2	2010	2	3	MX17004	NaN	14.4
3	2010	2	3	MX17004	24.1	NaN
4	2010	3	5	MX17004	32.1	14.2

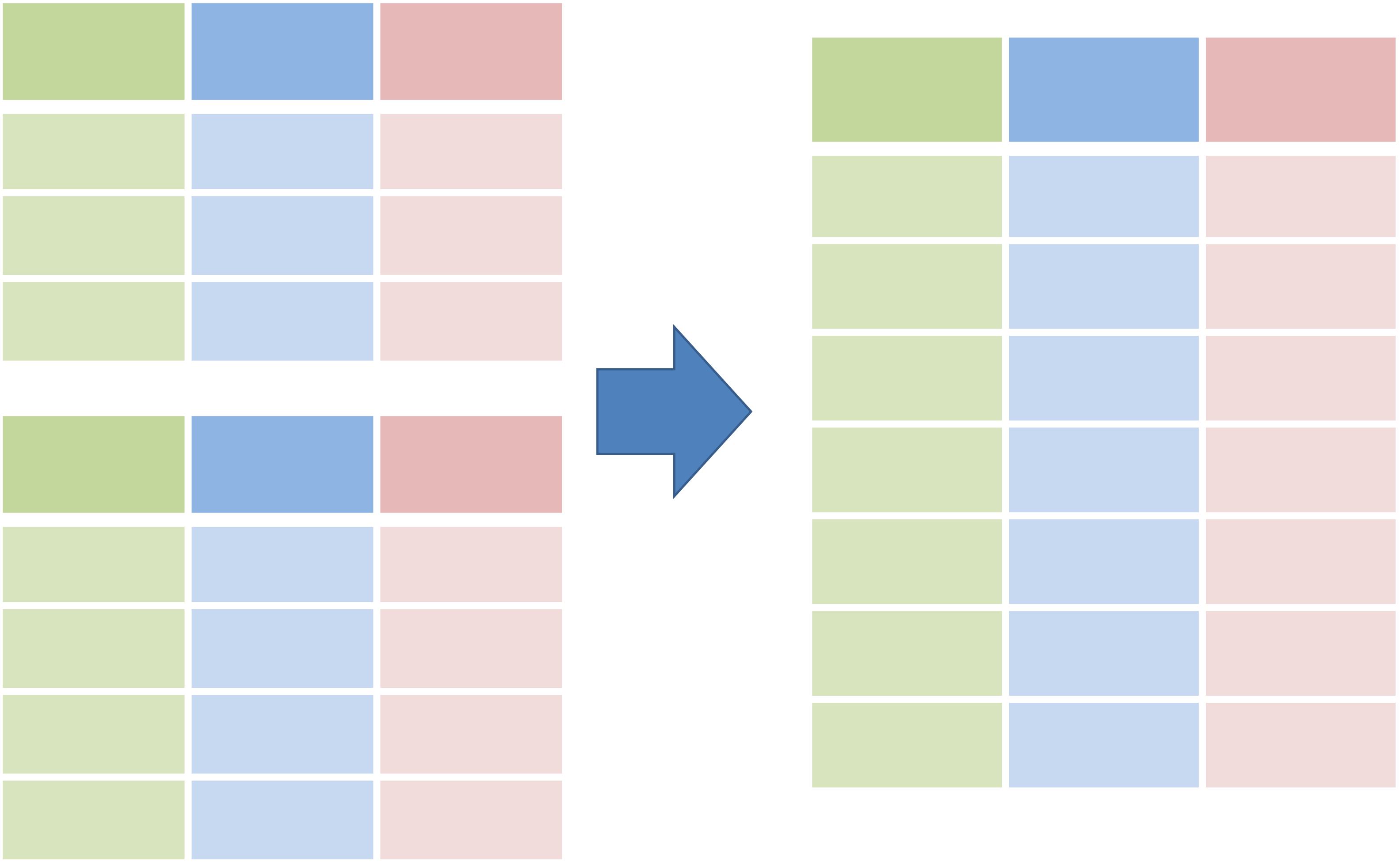
Source: https://towardsdatascience.com/whats-tidy-data-how-to-organize-messy-datasets-in-python-with-melt-and-pivotable-functions-5d52daa996c9

Reshaping data

changing the layout of a data set

Concatenation

append rows of DataFrames

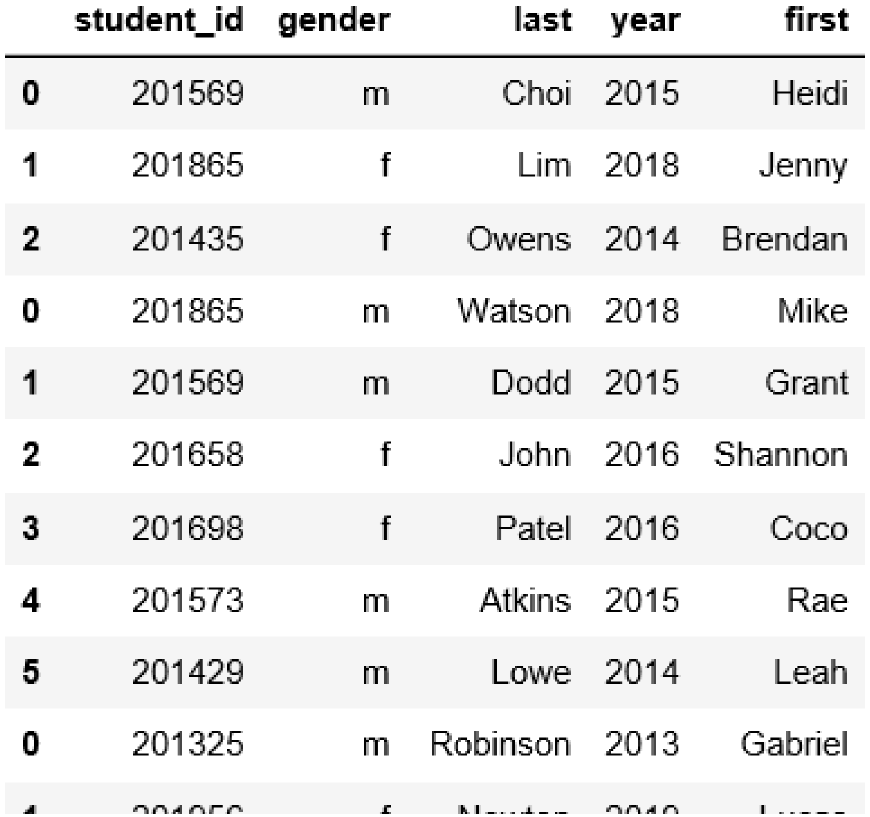


pd.concat([df1,df2])

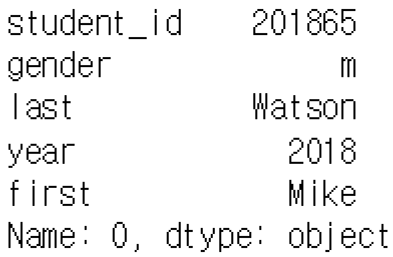
Concatenation

- If you want to combine (append) data into one, you can use **concat()** method in pandas.

```
>>> import pandas as pd
>>> df1 = pd.read_csv("student1.csv")
>>> df2 = pd.read_csv("student2.csv")
>>> df3 = pd.read_csv("student3.csv")
>>> students = pd.concat([df1, df2, df3])
>>> print(students.iloc[3,])
```



	student_id	gender	last	year	first
0	201569	m	Choi	2015	Heidi
1	201865	f	Lim	2018	Jenny
2	201435	f	Owens	2014	Brendan
0	201865	m	Watson	2018	Mike
1	201569	m	Dodd	2015	Grant
2	201658	f	John	2016	Shannon
3	201698	f	Patel	2016	Coco
4	201573	m	Atkins	2015	Rae
5	201429	m	Lowe	2014	Leah
0	201325	m	Robinson	2013	Gabriel
1	201956	f	Newton	2019	Lucas

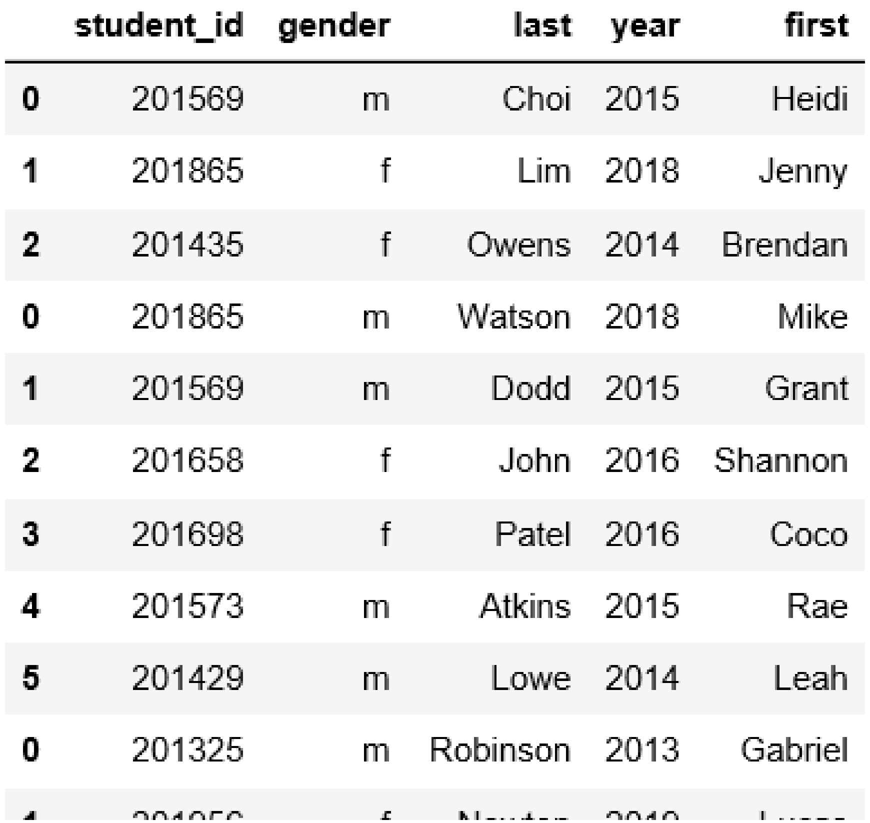


```
student_id    201865
gender             m
last          Watson
year            2018
first           Mike
Name: 0, dtype: object
```

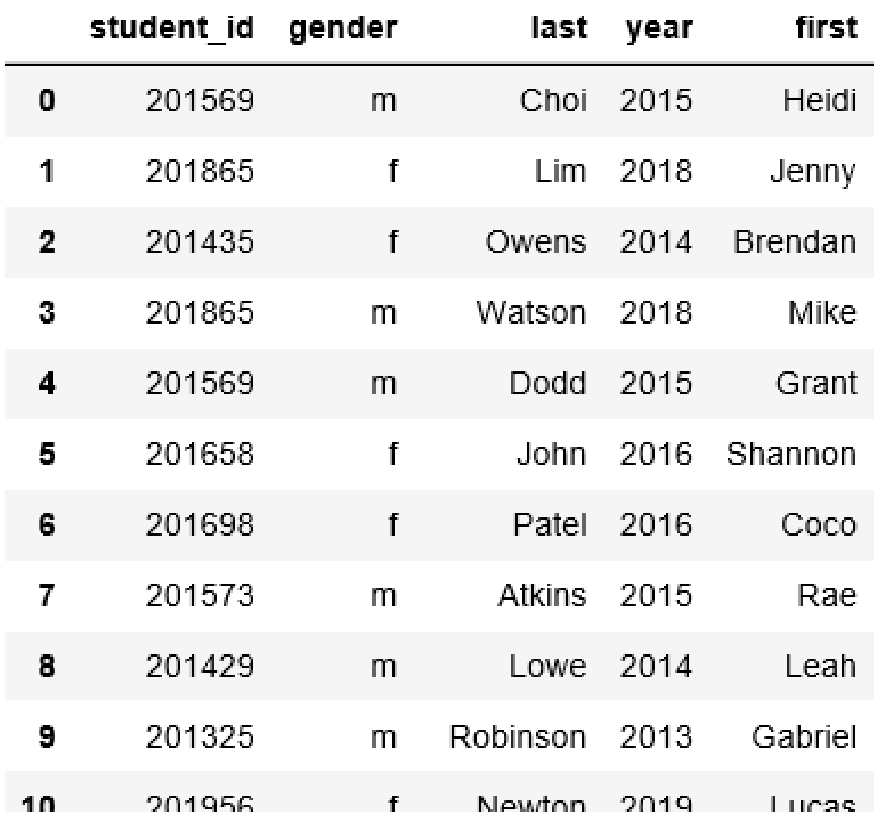
Concatenation

- To reset the index number starting from 0, add **ignore_index= True**

```
>>> students = students.append(new_row_df, ignore_index= True)
```



	student_id	gender	last	year	first
0	201569	m	Choi	2015	Heidi
1	201865	f	Lim	2018	Jenny
2	201435	f	Owens	2014	Brendan
0	201865	m	Watson	2018	Mike
1	201569	m	Dodd	2015	Grant
2	201658	f	John	2016	Shannon
3	201698	f	Patel	2016	Coco
4	201573	m	Atkins	2015	Rae
5	201429	m	Lowe	2014	Leah
0	201325	m	Robinson	2013	Gabriel
1	201956	f	Newton	2019	Lucas



	student_id	gender	last	year	first
0	201569	m	Choi	2015	Heidi
1	201865	f	Lim	2018	Jenny
2	201435	f	Owens	2014	Brendan
3	201865	m	Watson	2018	Mike
4	201569	m	Dodd	2015	Grant
5	201658	f	John	2016	Shannon
6	201698	f	Patel	2016	Coco
7	201573	m	Atkins	2015	Rae
8	201429	m	Lowe	2014	Leah
9	201325	m	Robinson	2013	Gabriel
10	201956	f	Newton	2019	Lucas

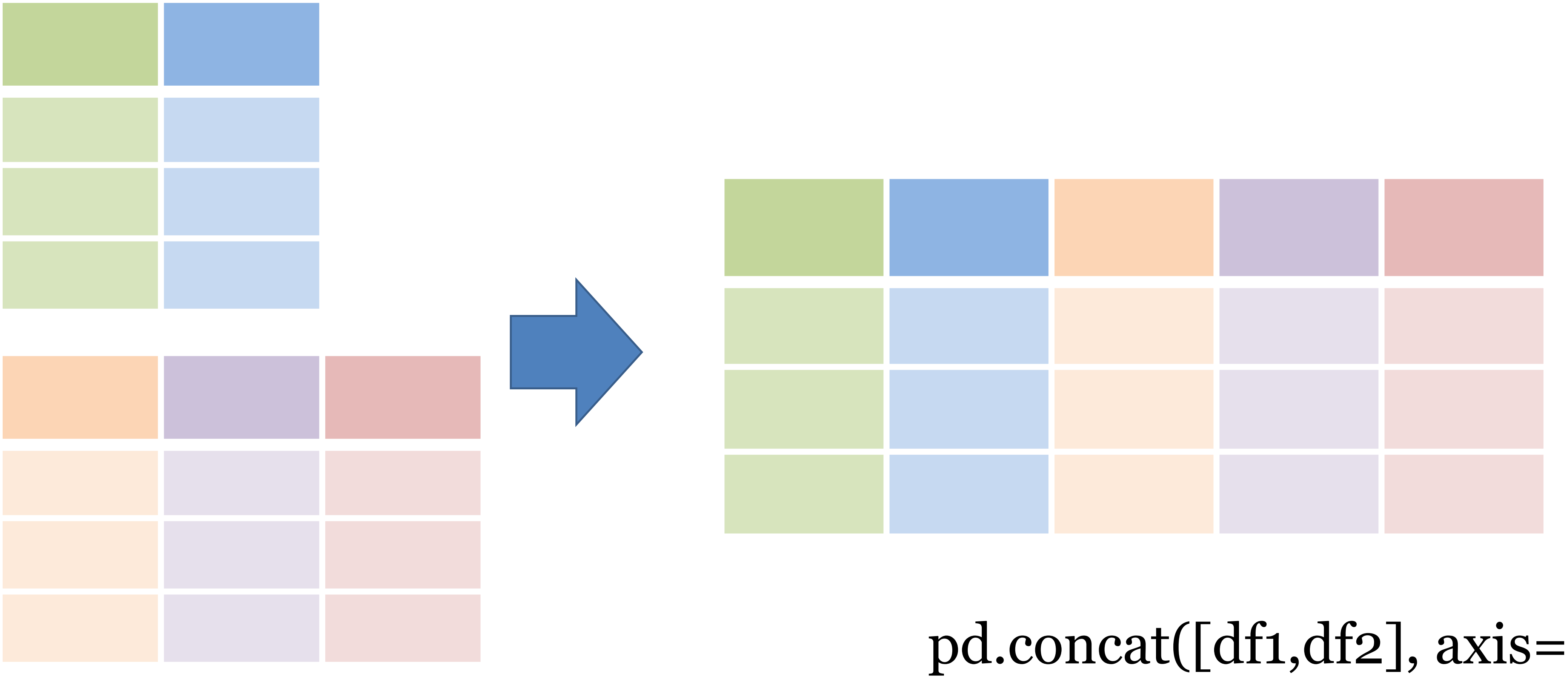
Concatenation

- If you want to add an entry, then you have to specify the column names!
- If you add one dataframe, then you can use append().
- if you add more than one, then use concat().

```
>>> new_row_df = pd.DataFrame([[‘Alex’, ‘Watchman’, ‘m’, 201729,
2017]], columns = [‘first’, ‘last’, ‘gender’ , ‘student_id’, ‘year’])
>>> students = students.append(new_row_df)
```

Concatenation

append columns of DataFrames

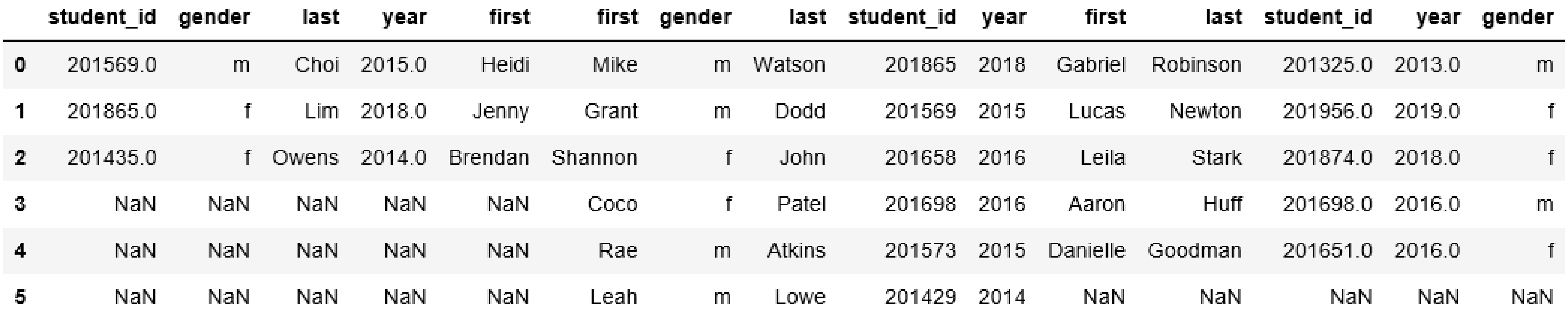


```
pd.concat([df1,df2], axis=1)
```

Concatenation

- append columns of the DataFrames

```
>>> col_concat = pd.concat([df1,df2,df3], axis = 1)
```

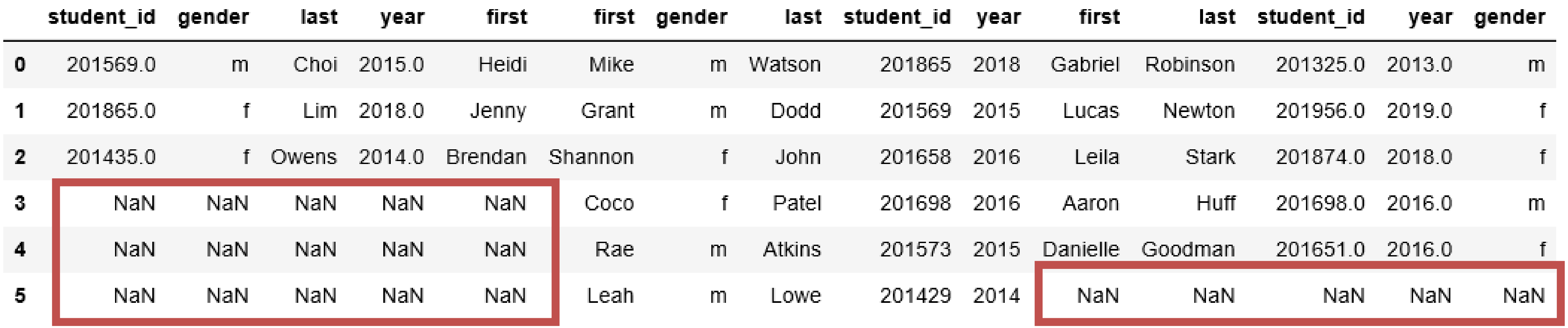


	student_id	gender	last	year	first	first	gender	last	student_id	year	first	last	student_id	year	gender
0	201569.0	m	Choi	2015.0	Heidi	Mike	m	Watson	201865	2018	Gabriel	Robinson	201325.0	2013.0	m
1	201865.0	f	Lim	2018.0	Jenny	Grant	m	Dodd	201569	2015	Lucas	Newton	201956.0	2019.0	f
2	201435.0	f	Owens	2014.0	Brendan	Shannon	f	John	201658	2016	Leila	Stark	201874.0	2018.0	f
3	NaN	NaN	NaN	NaN	NaN	Coco	f	Patel	201698	2016	Aaron	Huff	201698.0	2016.0	m
4	NaN	NaN	NaN	NaN	NaN	Rae	m	Atkins	201573	2015	Danielle	Goodman	201651.0	2016.0	f
5	NaN	NaN	NaN	NaN	NaN	Leah	m	Lowe	201429	2014	NaN	NaN	NaN	NaN	NaN

Concatenation

- append columns of the DataFrames

```
>>> col_concat = pd.concat([df1,df2,df3], axis = 1)
```



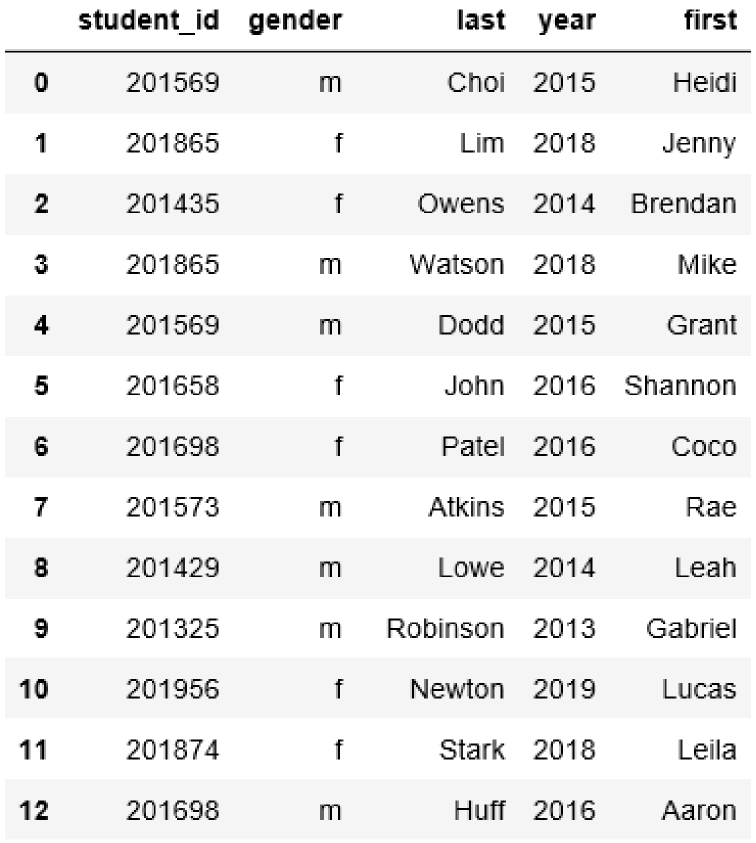
	student_id	gender	last	year	first	first	gender	last	student_id	year	first	last	student_id	year	gender
0	201569.0	m	Choi	2015.0	Heidi	Mike	m	Watson	201865	2018	Gabriel	Robinson	201325.0	2013.0	m
1	201865.0	f	Lim	2018.0	Jenny	Grant	m	Dodd	201569	2015	Lucas	Newton	201956.0	2019.0	f
2	201435.0	f	Owens	2014.0	Brendan	Shannon	f	John	201658	2016	Leila	Stark	201874.0	2018.0	f
3	NaN	NaN	NaN	NaN	NaN	Coco	f	Patel	201698	2016	Aaron	Huff	201698.0	2016.0	m
4	NaN	NaN	NaN	NaN	NaN	Rae	m	Atkins	201573	2015	Danielle	Goodman	201651.0	2016.0	f
5	NaN	NaN	NaN	NaN	NaN	Leah	m	Lowe	201429	2014	NaN	NaN	NaN	NaN	NaN

NaN? (Not a Number)
Python’s way of representing a ‘missing value’

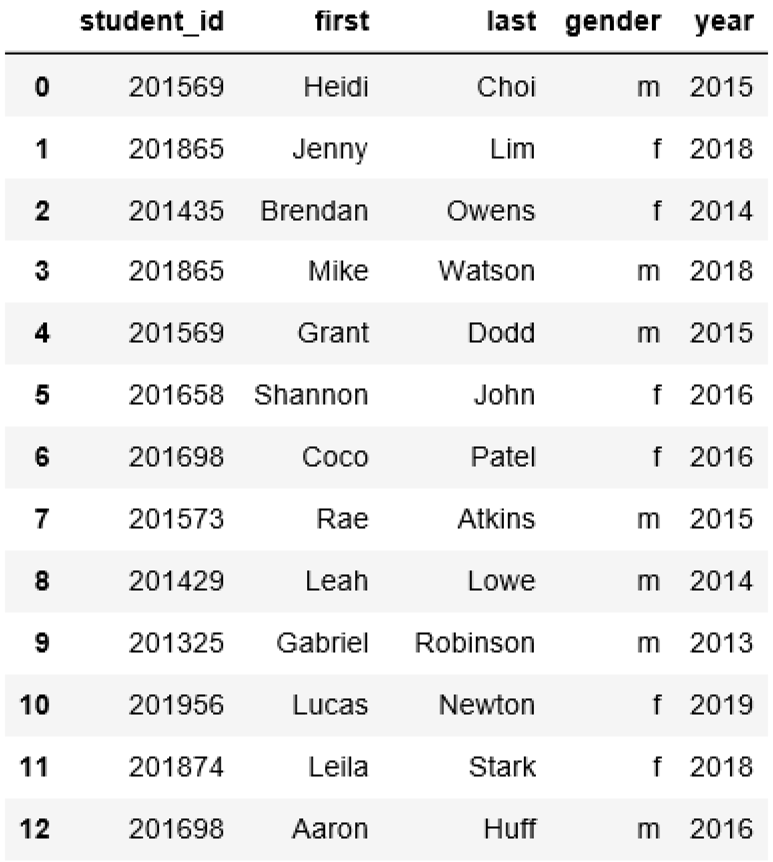
change the order of columns

- if you want to change the order of the columns, then we can change the order as below

```
>>> students = students[['student_id', 'first', 'last' ,'gender', 'year']]
```



	student_id	gender	last	year	first
0	201569	m	Choi	2015	Heidi
1	201865	f	Lim	2018	Jenny
2	201435	f	Owens	2014	Brendan
3	201865	m	Watson	2018	Mike
4	201569	m	Dodd	2015	Grant
5	201658	f	John	2016	Shannon
6	201698	f	Patel	2016	Coco
7	201573	m	Atkins	2015	Rae
8	201429	m	Lowe	2014	Leah
9	201325	m	Robinson	2013	Gabriel
10	201956	f	Newton	2019	Lucas
11	201874	f	Stark	2018	Leila
12	201698	m	Huff	2016	Aaron



	student_id	first	last	gender	year
0	201569	Heidi	Choi	m	2015
1	201865	Jenny	Lim	f	2018
2	201435	Brendan	Owens	f	2014
3	201865	Mike	Watson	m	2018
4	201569	Grant	Dodd	m	2015
5	201658	Shannon	John	f	2016
6	201698	Coco	Patel	f	2016
7	201573	Rae	Atkins	m	2015
8	201429	Leah	Lowe	m	2014
9	201325	Gabriel	Robinson	m	2013
10	201956	Lucas	Newton	f	2019
11	201874	Leila	Stark	f	2018
12	201698	Aaron	Huff	m	2016

Other useful methods

```
>>> df.sort_values(‘column_name’)
```

order rows by values of a column (low to high)

```
>>> df.sort_values(‘column_name’, ascending=False)
```

order rows by values of a column (high to low).

```
>>> df.rename(columns = {‘current_column_name’ : ‘new_column_name’})
```

Rename the column name of a dataframe.

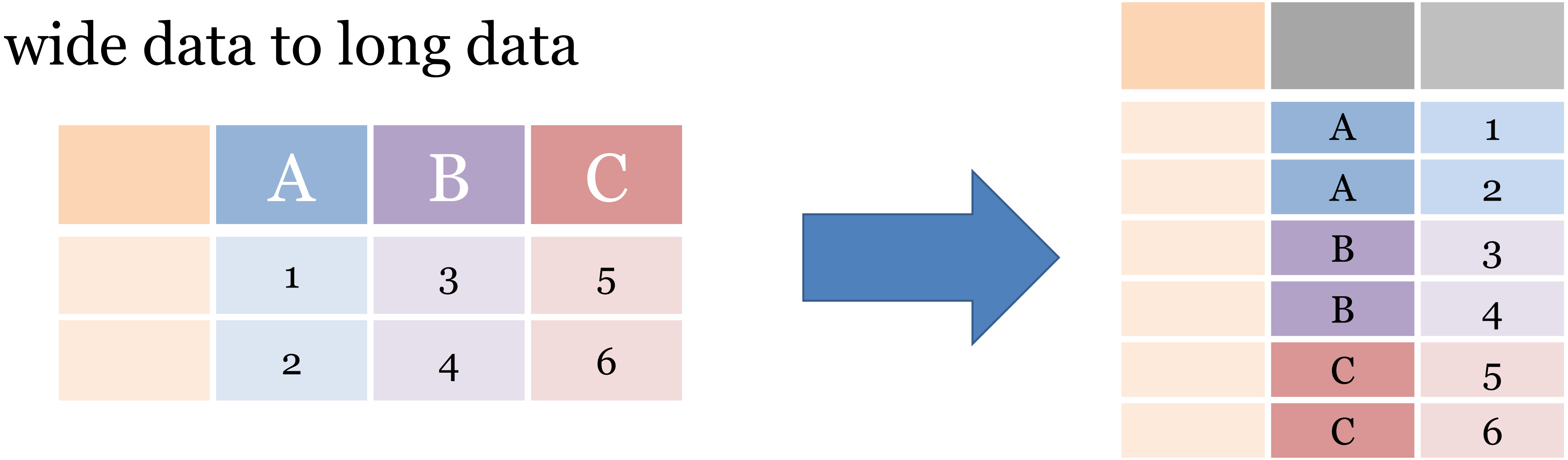
```
>>> df.sort_index()
```

sort the index of a dataframe

```
>>> df.drop(columns = [‘column_name’])
```

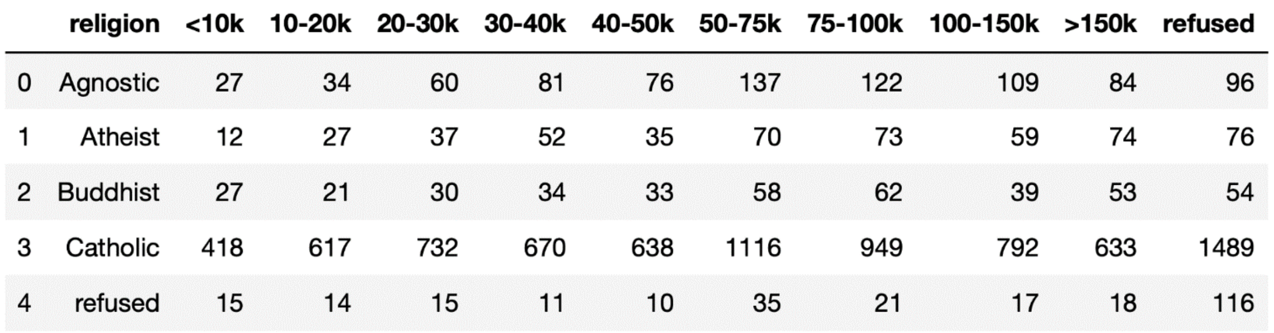
Melt

- wide data to long data

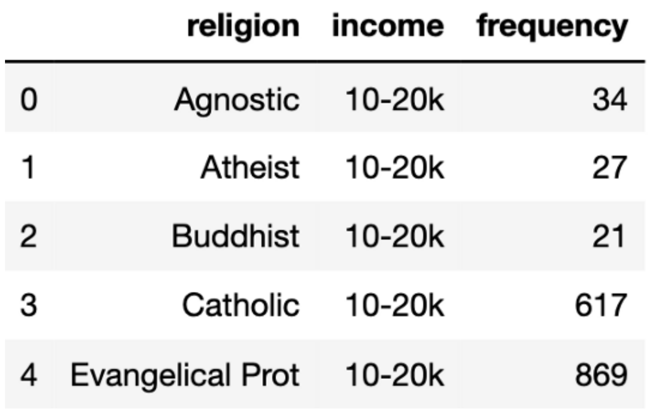


	A	B	C
	1	3	5
	2	4	6

	A	1
	A	2
	B	3
	B	4
	C	5
	C	6



	religion	<10k	10-20k	20-30k	30-40k	40-50k	50-75k	75-100k	100-150k	>150k	refused
0	Agnostic	27	34	60	81	76	137	122	109	84	96
1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	refused	15	14	15	11	10	35	21	17	18	116

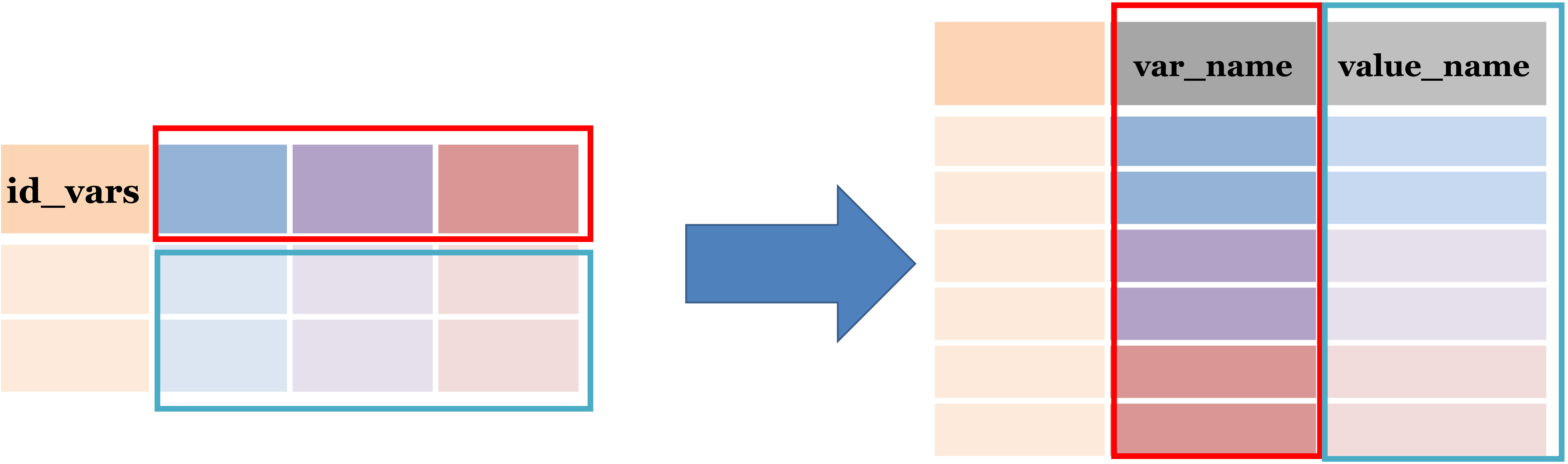


	religion	income	frequency
0	Agnostic	10-20k	34
1	Atheist	10-20k	27
2	Buddhist	10-20k	21
3	Catholic	10-20k	617
4	Evangelical Prot	10-20k	869

```
pd.melt(df, id_vars = ‘religion’, var_name = ‘income’, value_name = ‘frequency`’)
```

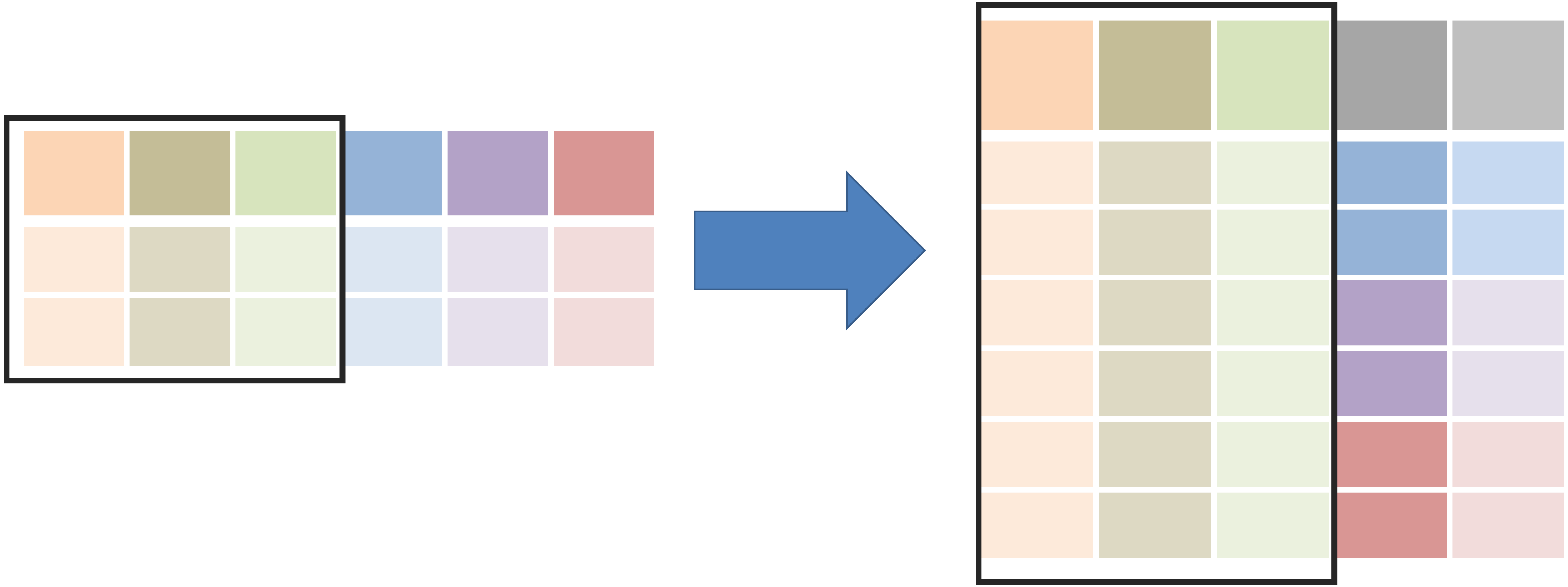
Melt

- **id_vars** = the column that you want to fix
- **var_name** = a new name of the column that is to go where current values are.
- **value_name** = a new name of the column for values (what values represent)



Melt

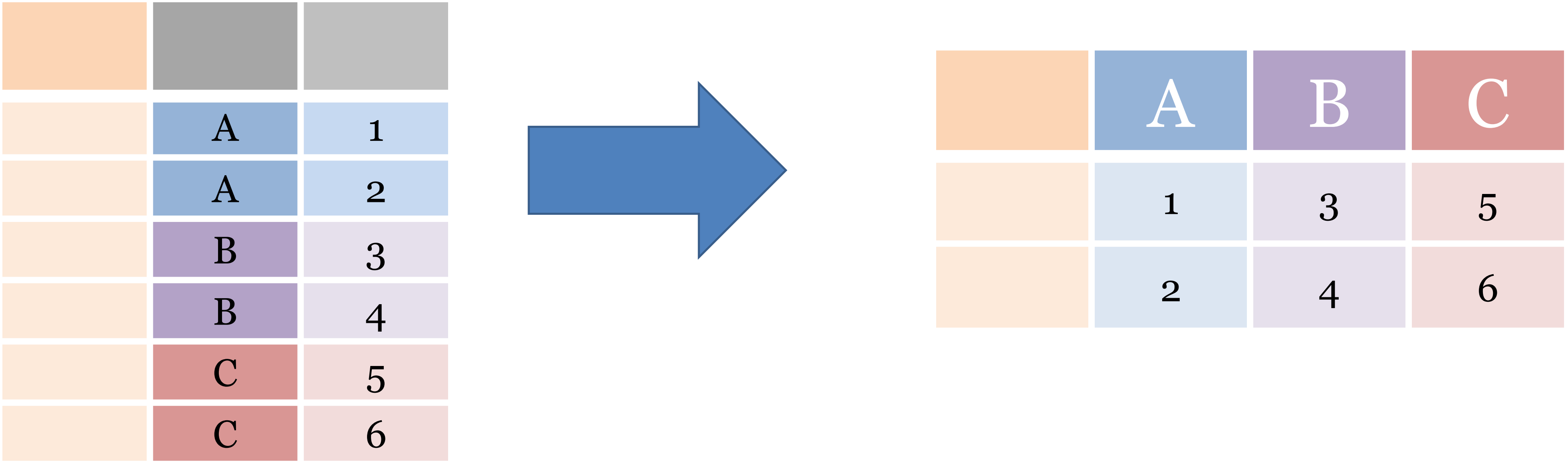
- we can also fix multiple columns and change the rest into long!



pd.melt(df, id_vars = [‘a *list of column names that you want to fix*’],
var_name = ‘income’, value_name = ‘count’)

Pivot

- long to wide.



	A	1
	A	2
	B	3
	B	4
	C	5
	C	6

	A	B	C
	1	3	5
	2	4	6

df.pivot(index = ‘*what you want to fix*’ , columns = ‘*name of the new column*’, values = ‘*name of the column that you wish to make it as value*’)

4 ways to combine data sets: merge!

df1

x1	x2
A	1
B	2
C	3

+

df2

x1	x3
A	T
B	T
D	F

=

x1	x2	x3
A	1	T
B	2	T
C	3	NaN

```
pd.merge(df1, df2,
            how = ‘left’, on = ‘x1’)
```

✓ Join matching rows from df2 to df1

x1	x2	x3
A	1	T
B	2	T
D	NaN	F

```
pd.merge(df1, df2,
            how = ‘right’, on = ‘x1’)
```

✓ Join matching rows from df1 to df2

x1	x2	x3
A	1	T
B	2	T

```
pd.merge(df1, df2,
            how = ‘inner’, on = ‘x1’)
```

✓ Only retain rows existing in both sets

x1	x2	x3
A	1	T
B	2	T
C	3	NaN
D	NaN	F

```
pd.merge(df1, df2,
            how = ‘outer’, on = ‘x1’)
```

✓ Retain all values, all rows

Throughout the class.

I am sure you must be wondering whether you have to memorize all the parameters to code.

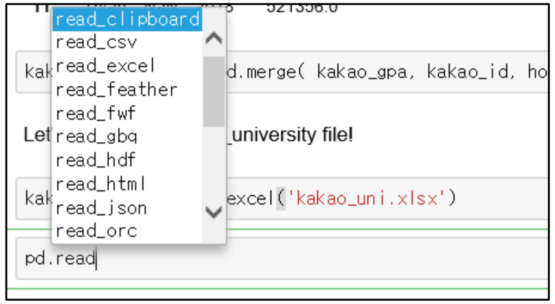
- No, you can just look them up every time you try to do something.
- But, depending on your work, you will be using some methods very often.
- In that case, if you repeat a lot, you will be able to memorize them.

There are so many parameters that you can set.

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

A huge tip on methods and arguments: Jupyter Notebook

tab



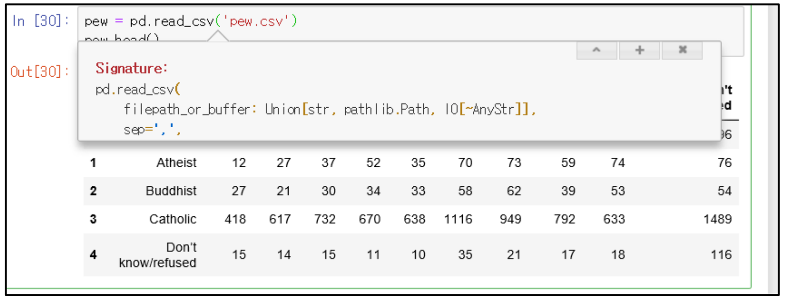
```
kak d.merge( kakao_gpa, kakao_id, ho
```

Let _university file!

```
kak excel('kakao_uni.xlsx')
```

```
pd.read
```

shift + tab



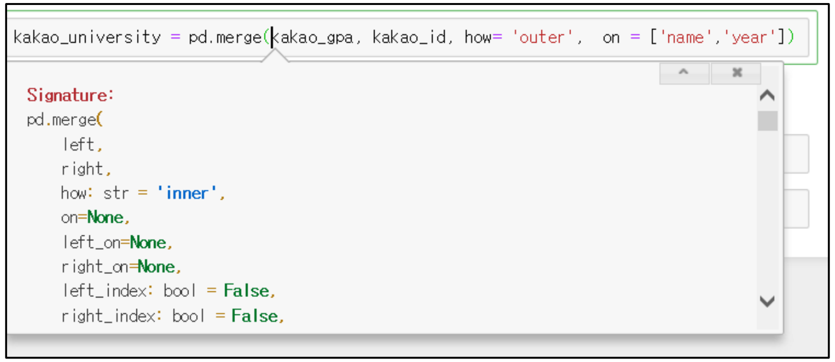
```
In [30]: pew = pd.read_csv('pew.csv')
```

Out[30]:

```
Signature:
pd.read_csv(
    filepath_or_buffer: Union[str, pathlib.Path, IO[~AnyStr]],
    sep=',',
```

1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	Don't know/refused	15	14	15	11	10	35	21	17	18	116

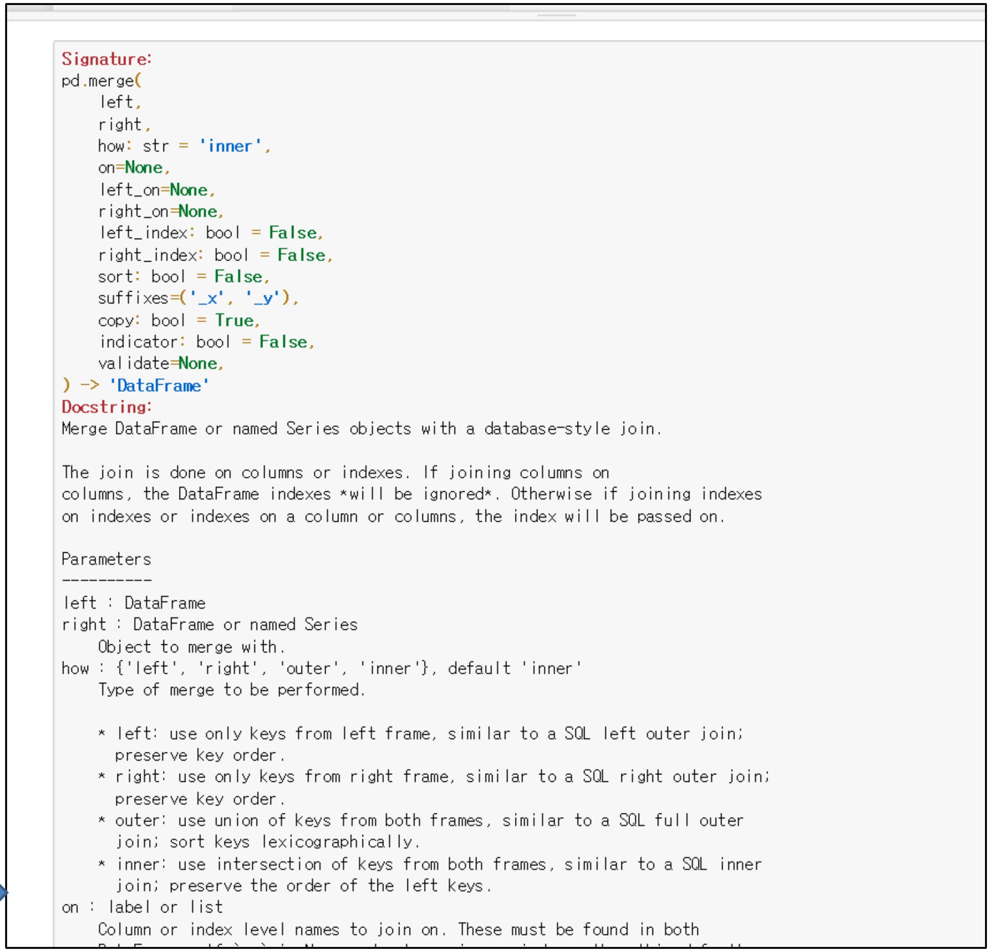
shift + tab + tab



```
kakao_university = pd.merge(kakao_gpa, kakao_id, how= 'outer',  on = ['name','year'])
```

```
Signature:
pd.merge(
    left,
    right,
    how: str = 'inner',
    on=None,
    left_on=None,
    right_on=None,
    left_index: bool = False,
    right_index: bool = False,
```

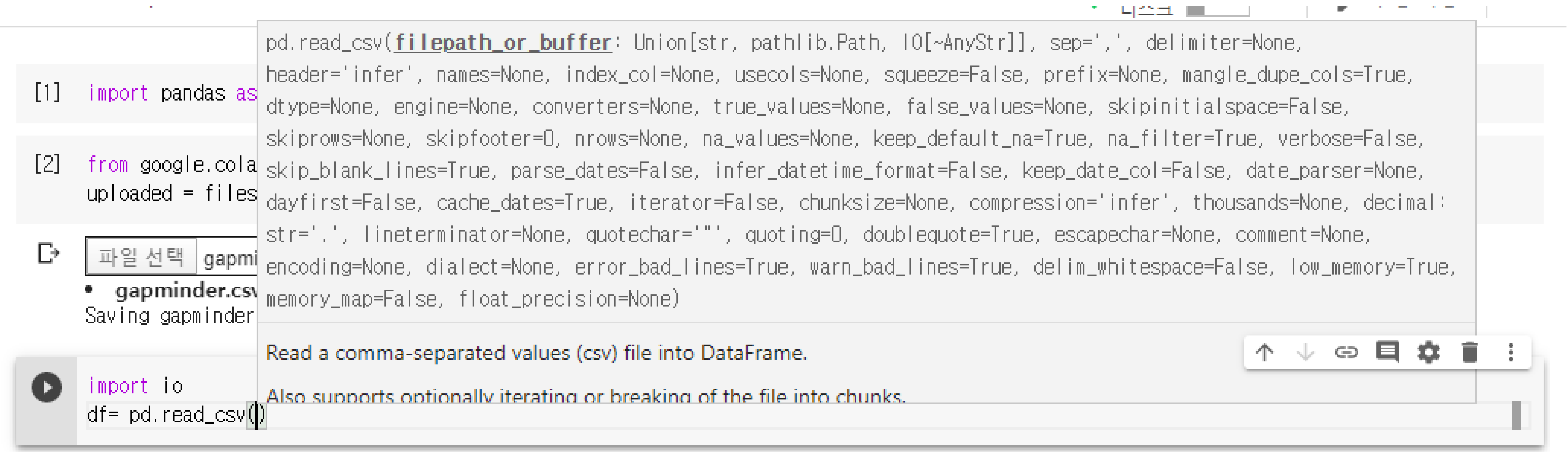
shift+ tab + tab + tab



```
Signature:
pd.merge(
    left,
    right,
    how: str = 'inner',
    on=None,
    left_on=None,
    right_on=None,
    left_index: bool = False,
    right_index: bool = False,
    sort: bool = False,
    suffixes=('_x', '_y'),
    copy: bool = True,
    indicator: bool = False,
    validate=None,
) -> 'DataFrame'
Docstring:
Merge DataFrame or named Series objects with a database-style join.

The join is done on columns or indexes. If joining columns on
columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes
on indexes or indexes on a column or columns, the index will be passed on.

Parameters
----------
left : DataFrame
right : DataFrame or named Series
    Object to merge with.
how : {'left', 'right', 'outer', 'inner'}, default 'inner'
    Type of merge to be performed.

    * left: use only keys from left frame, similar to a SQL left outer join;
      preserve key order.
    * right: use only keys from right frame, similar to a SQL right outer join;
      preserve key order.
    * outer: use union of keys from both frames, similar to a SQL full outer
      join; sort keys lexicographically.
    * inner: use intersection of keys from both frames, similar to a SQL inner
      join; preserve the order of the left keys.
on : label or list
    Column or index level names to join on. These must be found in both
```

A huge tip on methods and arguments: Google Colab

Just wait until it pops up.



```
[1]  import pandas as

[2]  from google.cola
     uploaded = files
```

```
• gapminder.csv
Saving gapminder

import io
df= pd.read_csv()
```

```
pd.read_csv(filepath_or_buffer: Union[str, pathlib.Path, IO[~AnyStr]], 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=0, doublequote=True, escapechar=None, comment=None,
encoding=None, dialect=None, error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False, low_memory=True,
memory_map=False, float_precision=None)
```

Read a comma-separated values (csv) file into DataFrame.

Also supports optionally iterating or breaking of the file into chunks.

Saving files in csv or excel files

In Jupyter Notebook

```
>>> df.to_csv(‘filename.csv’)
>>> df.to_excel(‘filename.xls’)
>>> df.to_excel(‘filename.xlsx’)
```

In google colab

```
>>> df.to_csv('filename.csv')
>>> files.download('filename.csv')
```

or..

1. open the left pane
2. select 'Files' tab
3. click 'Refresh'
4. right click the file, then download