

Computing in Context: Fall 2024

Lecture 3 | Tidy Data, Tidy Mind

Tidy Data

Does it spark joy?

“Tidy” Data?

“Tidy datasets are all alike but every messy dataset is messy in its own way”

– Hadley Wickham

“Tidy” Data?

Same data, different organization.
Which one is best for data analysis?

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

“Tidy” Datasets

Three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	127201272
China	2000	216766	128042583

variables

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	127201272
China	2000	216766	128042583

observations

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	127201272
China	2000	216766	128042583

values

“Tidy” Datasets

Identify variables, observations, and values. What would a tidy version look like?

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

“Tidy” Datasets

Identify variables, observations, and values. What would a tidy version look like?

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

Table 11: Original weather dataset. There is a column for each possible day in the month. Columns d9 to d31 have been omitted to conserve space.

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) Tidy data

“Messy” data – prepare yourselves

“Messy” data

Variables are
stored in both
rows and columns



city	type	date	temperature
Bilbao	tmax	2024-07-03	34
Bilbao	tmin	2024-07-03	25
Bordeaux	tmax	2024-03-21	29
Bordeaux	tmin	2024-03-21	23
Berlin	tmax	2021-08-16	21
Berlin	tmin	2021-08-16	14
Heraklion	tmax	2021-09-01	30
Heraklion	tmin	2021-09-01	23

“Messy” data

Variables are
stored in both
rows and columns



city	type	date	temperature
Bilbao	tmax	2024-07-03	34
Bilbao	tmin	2024-07-03	25
Bordeaux	tmax	2024-03-21	29
Bordeaux	tmin	2024-03-21	23
Berlin	tmax	2021-08-16	21
Berlin	tmin	2021-08-16	14
Heraklion	tmax	2021-09-01	30
Heraklion	tmin	2021-09-01	23

Column headers
are values, not
variable names



subject	date	A	B
PB	2024-07-03	0.12	0.19
VM	2024-03-21	0.37	0.41
TZ	2021-08-16	0.68	0.73
LS	2021-09-01	0.07	0.08
ZS	2023-11-11	0.08	0.16

“Messy” data

Variables are
stored in both
rows and columns



city	type	date	temperature
Bilbao	tmax	2024-07-03	34
Bilbao	tmin	2024-07-03	25
Bordeaux	tmax	2024-03-21	29
Bordeaux	tmin	2024-03-21	23
Berlin	tmax	2021-08-16	21
Berlin	tmin	2021-08-16	14
Heraklion	tmax	2021-09-01	30
Heraklion	tmin	2021-09-01	23

Column headers
are values, not
variable names



subject	date	A	B
PB	2024-07-03	0.12	0.19
VM	2024-03-21	0.37	0.41
TZ	2021-08-16	0.68	0.73
LS	2021-09-01	0.07	0.08
ZS	2023-11-11	0.08	0.16

Some variables
are stored in the
file names



```
2024-01_prices_DE.csv
2024-01_prices_FR.csv
2024-02_prices_DE.csv
2024-02_prices_FR.csv
```

“Messy” data

Variables are
stored in both
rows and columns



city	type	date	temperature
Bilbao	tmax	2024-07-03	34
Bilbao	tmin	2024-07-03	25
Bordeaux	tmax	2024-03-21	29
Bordeaux	tmin	2024-03-21	23
Berlin	tmax	2021-08-16	21
Berlin	tmin	2021-08-16	14
Heraklion	tmax	2021-09-01	30
Heraklion	tmin	2021-09-01	23

Column headers
are values, not
variable names



subject	date	A	B
PB	2024-07-03	0.12	0.19
VM	2024-03-21	0.37	0.41
TZ	2021-08-16	0.68	0.73
LS	2021-09-01	0.07	0.08
ZS	2023-11-11	0.08	0.16

Some variables
are stored in the
file names



2024-01_prices_DE.csv
2024-01_prices_FR.csv
2024-02_prices_DE.csv
2024-02_prices_FR.csv

Multiple variables
are stored in one
column



country	year	variable	cases
Angola	2000	sp_m_014	186.0
Angola	2001	sp_m_014	230.0
Angola	2002	sp_m_014	435.0
Angola	2003	sp_m_014	409.0
Angola	2004	sp_m_014	554.0
Angola	2005	sp_m_014	520.0
Angola	2006	sp_m_014	540.0

Messy Datasets - Too Long?

Messy!

	country	year	type	count
0	Afghanistan	1999	cases	745
1	Afghanistan	1999	population	19987071
2	Afghanistan	2000	cases	2666
3	Afghanistan	2000	population	20595360
4	Brazil	1999	cases	37737
5	Brazil	1999	population	172006362
6	Brazil	2000	cases	80488
7	Brazil	2000	population	174504898
8	China	1999	cases	212258
9	China	1999	population	1272915272
10	China	2000	cases	213766
11	China	2000	population	1280428583

What's wrong here?



Messy Datasets - Too Long?

Messy!

	country	year	type	count
0	Afghanistan	1999	cases	745
1	Afghanistan	1999	population	19987071
2	Afghanistan	2000	cases	2666
3	Afghanistan	2000	population	20595360
4	Brazil	1999	cases	37737
5	Brazil	1999	population	172006362
6	Brazil	2000	cases	80488
7	Brazil	2000	population	174504898
8	China	1999	cases	212258
9	China	1999	population	1272915272
10	China	2000	cases	213766
11	China	2000	population	1280428583

What's wrong here?

Each variable does not
have its own column!



Tidy!

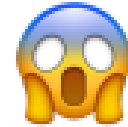
	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



The life-changing magic of tidying your data

city	type	date	temperature
Bilbao	tmax	2024-07-03	34
Bilbao	tmin	2024-07-03	25
Bordeaux	tmax	2024-03-21	29
Bordeaux	tmin	2024-03-21	23
Berlin	tmax	2021-08-16	21
Berlin	tmin	2021-08-16	14
Heraklion	tmax	2021-09-01	30
Heraklion	tmin	2021-09-01	23

The “type” column is
storing variable names



pivot



	city	date	tmax	tmin
	Berlin	2021-08-16	21	14
	Bilbao	2024-07-03	34	25
	Bordeaux	2024-03-21	29	23
	Heraklion	2021-09-01	30	23

```
df.pivot_table(  
    index=['city', 'date'], columns='type',  
    values='temperature', aggfunc='max',  
)
```


Messy Datasets - Too Wide?

Messy!

	country	1999	2000
0	Afghanistan	745	2666
1	Brazil	37737	80488
2	China	212258	213766

What's wrong here?



Messy Datasets - Too Wide?

Messy!

	country	1999	2000
0	Afghanistan	745	2666
1	Brazil	37737	80488
2	China	212258	213766

What's wrong here?

Each observation (country-year)
does not have its own row!



Tidy!

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



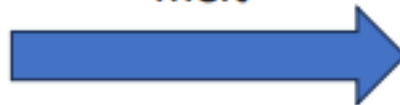
The life-changing magic of tidying your data

The treatment values are stored as a column name



subject	date	A	B
PB	2024-07-03	0.12	0.19
VM	2024-03-21	0.37	0.41
TZ	2021-08-16	0.68	0.73
LS	2021-09-01	0.07	0.08
ZS	2023-11-11	0.08	0.16

melt



subject	date	variable	response_time
PB	2024-07-03	A	0.12
VM	2024-03-21	A	0.37
TZ	2021-08-16	A	0.68
LS	2021-09-01	A	0.07
ZS	2023-11-11	A	0.08
PB	2024-07-03	B	0.19
VM	2024-03-21	B	0.41
TZ	2021-08-16	B	0.73
LS	2021-09-01	B	0.08
ZS	2023-11-11	B	0.16

```
pd.melt(data, id_vars=['subject', 'date'], value_name='response_time')
```

Split the columns in (A) “id_vars” and (B) non-”id_vars”. The column names in (B) are used as new values in a new column “variable”. The values in columns (B) go into a new column, “response_time”.

The life-changing magic of tidying your data

`pd.concat` – add together tables with the same variables (columns)

Some variables
are stored in the
file names



```
2024-01_prices_DE.csv
2024-01_prices_FR.csv
2024-02_prices_DE.csv
2024-02_prices_FR.csv
```

```
tables = []
for filename in filenames:
    # Parse filename
    year_month, _, country = filename[:-4].split('_')
    # Read table and add columns for the variables
    df = pd.read_csv(filename)
    # Add the variables that were in the filename
    df['year_month'] = year_month
    df['country'] = country
    # Store table
    tables.append(df)

# Create complete table
tidy_df = pd.concat(tables)
```



The life-changing magic of tidying your data

Multiple variables
are stored in one
column



country	year	variable	cases
Angola	2000	sp_m_014	186.0
Angola	2001	sp_m_014	230.0
Angola	2002	sp_m_014	435.0
Angola	2003	sp_m_014	409.0
Angola	2004	sp_m_014	554.0
Angola	2005	sp_m_014	520.0
Angola	2006	sp_m_014	540.0

Sometimes you'll want to "separate" the strings in a column into two separate columns.

Other times you'll want to "unite" (i.e., concatenate) the strings in two columns into a single column.

Separating

```
new_columns = (table3. # create new df `new_columns`  
    rate.str.split("/", expand = True). # split 'rate' on '/' into two columns  
    rename(columns = {0: "cases", 1: "population"}) # rename the columns  
)  
  
# add these columns onto original table (and drop rate column)  
pd.concat([table3.drop(columns = 'rate'), new_columns], axis = 1)
```

Sometimes you'll want to “separate” the strings in a column into two separate columns.

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

<https://byuidatascience.github.io/python4ds/tidy-data.html>

Uniting

```
# overwrite year column (previously 2-digit) with 4-digit year, then drop century column
table5['year'] = table5['century'] + table5['year']
table5.drop('century', axis = 1)
```



country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

country	century	year	rate
Afghanistan	19	99	745 / 19987071
Afghanistan	20	0	2666 / 20595360
Brazil	19	99	37737 / 172006362
Brazil	20	0	80488 / 174504898
China	19	99	212258 / 1272915272
China	20	0	213766 / 1280428583

Other times you'll want to “unite” (i.e., concatenate) the strings in two columns into a single column.

<https://byuidatascience.github.io/python4ds/tidy-data.html>

Note “axis” parameter

```
# columns: axis = 1
table5 = table5.assign(year = table5['century'] + table5['year']).drop('century', axis = 1)
table5
```

	country	year	rate
0	Afghanistan	191999	745/19987071
1	Afghanistan	202000	2666/20595360
2	Brazil	191999	37737/172006362
3	Brazil	202000	80488/174504898
4	China	191999	212258/1272915272
5	China	202000	213766/1280428583

```
table5.drop(0, axis = 0) # rows: axis = 0
```

	country	year	rate
1	Afghanistan	202000	2666/20595360
2	Brazil	191999	37737/172006362
3	Brazil	202000	80488/174504898
4	China	191999	212258/1272915272
5	China	202000	213766/1280428583

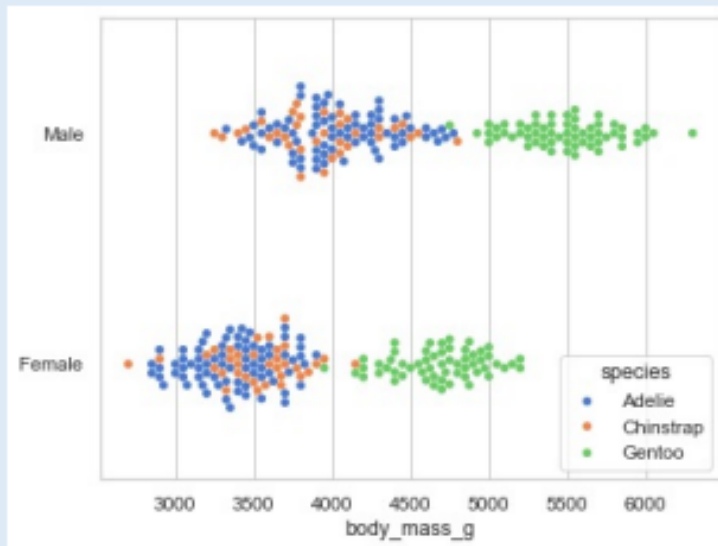
Rows: axis = 0

Columns: axis = 1

Why is tidy data good?

- Many analyses require a simple sequence of steps:
 - Filter by individual variables to discard data that is not needed
 - Group and summarize
 - Re-arrange (e.g. sort)
 - Visualize
- Joining tidy tables is easy!
- One can write generic code that takes tidy data as input.
For example, **seaborn** relies on tidy data to make complex plots

```
sns.swarmplot(  
    data=df,  
    x="body_mass_g",  
    y="sex",  
    hue="species",  
)
```



Data De-Duplication

Another potential headache

Data de-duplication

What if instead of missing data, you have ***too many*** values?

If you have too many versions of the same information, it could lead to misleading analyses.

You need to find some way to **de-duplicate**/unify the records at the *level of analysis*. Sounds easy, right?

Duplicates – Level I: Annoying

Pure **duplicates** in your data.

Example

Whoever was responsible for transferring data from the hospital's EHR system to your database accidentally did it twice.

Solution

Simple: drop the duplicate.

ID	Name	Sex	Age
80415	Sterling Archer	M	34
85148	Lana Kane	F	33
61598	Pam Poovey	F	35
61598	Pam Poovey	F	35
21846	Cyrill Figgis	M	40
51487	Cheryl Tunt	F	31
32545	Ray Gillette	M	37



Duplicates – Level II: Frustrating

ID	Name	Sex	Age
80415	Sterling Archer	M	34
85148	Lana Kane	F	33
61598	Pam Poovey	F	35
63579	Pam Poovey	F	35
21846	Cyrill Figgis	M	40
51487	Cheryl Tunt	F	31
32545	Ray Gillette	M	37

Another form of duplication: data pertaining to a single entity is entered as if it is multiple entities.

Example: A patient has an existing EHR file at a hospital. Upon returning to the hospital, a hospital employee mistakenly creates a new EHR file for this patient, rather than simply adding the most recent visit to the patient's existing record.

Duplicates – Level II: Frustrating

ID	Name	Sex	Age
80415	Sterling Archer	M	34
85148	Lana Kane	F	33
61598	Pam Poovey	F	35
63579	Pam Poovey	F	35
21846	Cyrill Figgis	M	40
51487	Cheryl Tunt	F	31
32545	Ray Gillette	M	37



Solution: Assuming enough identifying information was entered correctly in both files (DOB, sex, name, etc) ...

Relatively straightforward to identify the duplicate in the data and combine the two records.

Duplicates – Level III: Exasperating

ID	Name	Sex	Age
80415	Sterling Archer	M	34
85148	Lana Kane	F	33
61598	Pam Poovey	F	35
63579	Pamm Poovey	F	35
21846	Cyrill Figgis	M	40
51487	Cheryl Tunt	F	31
32545	Ray Gillette	M	37

Another form of duplication: when data pertaining to a single entity is entered as if it is multiple entities, and identifying information is mis-recorded:

BUT you know which ones are wrong.

Example: Imagine that when the patient's record is mistakenly duplicated, the name is misspelled. BUT you know the correct version.

Duplicates – Level III: Exasperating

ID	Name	Sex	Age
80415	Sterling Archer	M	34
85148	Lana Kane	F	33
61598	Pam Poovey	F	35
63579	Pamm Poovey	F	35
21846	Cyrill Figgis	M	40
51487	Cheryl Tunt	F	31
32545	Ray Gillette	M	37

Solution: It's more difficult to identify the duplicates, since the identifying information is no longer identical.

Maybe use some kind of “fuzzy” string matching technique (using a string metric, like the “Hamming Distance” implemented in an early lab!) to identify if two names are likely to refer to the same person.

Duplicates – Level IV: Existential Despair

ID	Name	Sex	Age
80415	Sterling Archer	M	34
85148	Lana Kane	F	33
61598	Pam Puvey	F	35
63579	Pamm Poovey	F	35
89532	Pam Poovey	F	35
51487	Cheryl Tunt	F	31
32545	Ray Gillette	M	37

Now imagine you have many records which contain errors, all pertaining to the same entity, but you **don't** know which one (if any) is correct.

Example: EHR data from a large city's EMS service. However, their identifying information is collected haphazardly, resulting in several distinct records pertaining to a single patient. Worse yet, you don't know which record is the "correct" one to link them to another database.

Duplicates – Level IV: Existential Despair

ID	Name	Sex	Age
80415	Sterling Archer	M	34
85148	Lana Kane	F	33
61598	Pam Puvey	F	35
63579	Pamm Poovey	F	36
89532	Pam Poovey	F	34
51487	Cheryl Tunt	F	31
32545	Ray Gillette	M	37

?
?
?

Solution: Cry.

When you're done, spend countless hours figuring out how best to probabilistically link records.

Maybe using some kind of machine learning algorithm?
(may lead to more crying)

Data Copies or Alterations

A pox on thee...

Create new DFs, or alter existing DFs?


```
table5['year'] = table5['century'] + table5['year']  
table5.drop('century', axis = 1)
```

OR (*important: what's the difference?*)

```
table5 = table5.assign(year = table5['century'] + table5['year']).drop('century', axis = 1)
```

Create new DFs, or alter existing DFs?

Creates a new (overwrites existing) column within the original object `table5`



```
table5['year'] = table5['century'] + table5['year']  
table5.drop('century', axis = 1)
```

OR (important: what's the difference?)

```
table5 = table5.assign(year = table5['century'] + table5['year']).drop('century', axis = 1)
```



pd.assign() method returns a NEW DF object, called `table5`



Why might use of 'assign()' be preferable?

```
table5.assign(year = table5['century'] + table5['year']).drop('century', axis = 1)
```

Common reasons:

- You may want to avoid altering the original DF
- Can “chain” methods (*assign()* and *drop()*) on a single line
- But – may be more memory intensive

When in doubt, check documentation!

pandas.DataFrame.assign

`DataFrame.assign(**kwargs)`

[\[source\]](#)

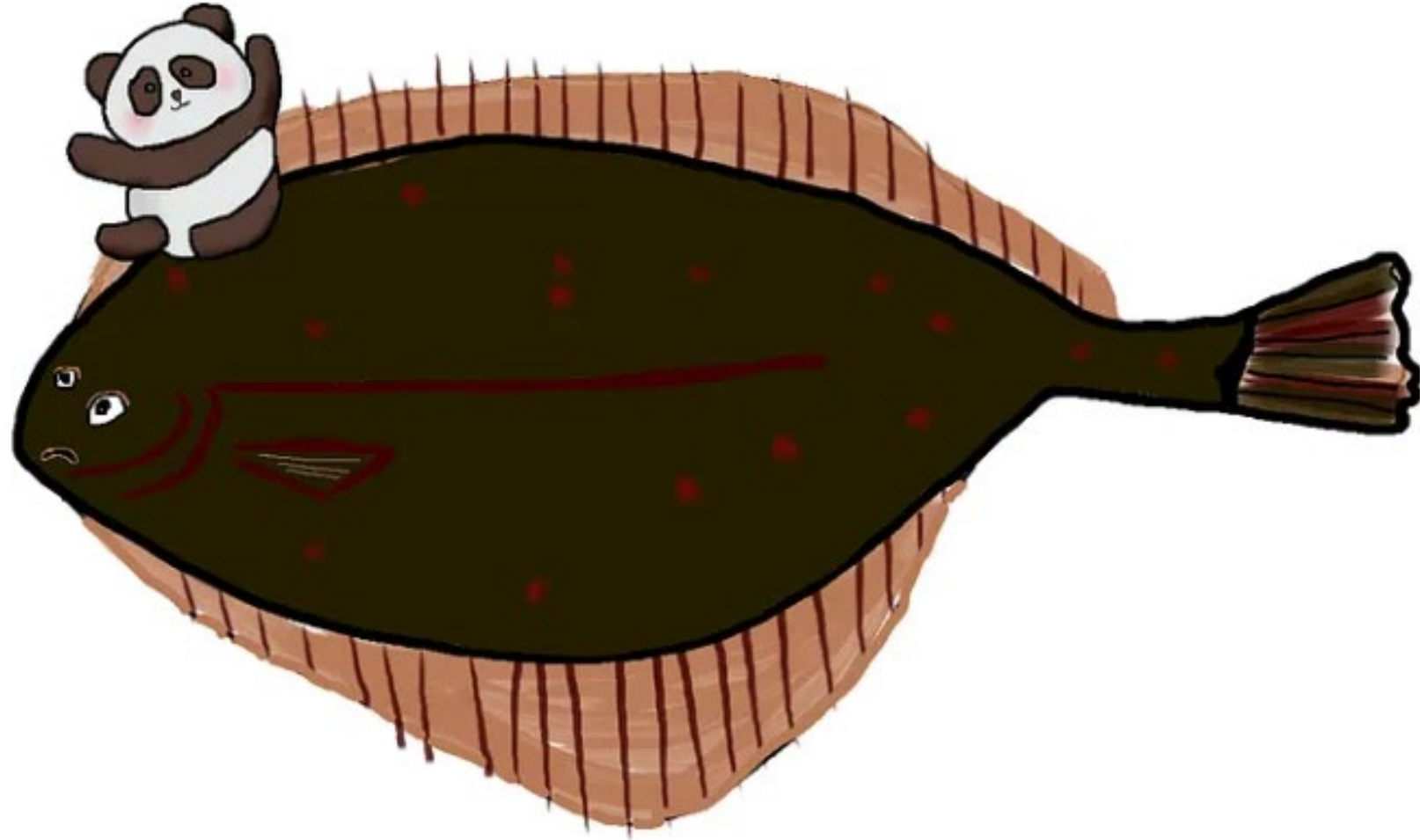
Assign new columns to a DataFrame.

Returns a new object with all original columns in addition to new ones. Existing columns that are re-assigned will be overwritten.

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.assign.html>

Many DF methods like this, so be sure to use them properly!

Beware pandas`, inplace = ` option!



Beware pandas `inplace = True` option!

```
table5.drop(0, axis = 0) # rows: axis = 0
```

	country	year	rate
1	Afghanistan	202000	2666/20595360
2	Brazil	191999	37737/172006362
3	Brazil	202000	80488/174504898
4	China	191999	212258/1272915272
5	China	202000	213766/1280428583

table5

	country	year	rate
0	Afghanistan	191999	745/19987071
1	Afghanistan	202000	2666/20595360
2	Brazil	191999	37737/172006362
3	Brazil	202000	80488/174504898
4	China	191999	212258/1272915272
5	China	202000	213766/1280428583

Why You Should Probably Never Use pandas `inplace=True`

Truly it is a curse on the library and a pox on thee if you use it

<https://towardsdatascience.com/why-you-should-probably-never-use-pandas-inplace-true-9f9f211849e4>

```
table5.drop(0, axis = 0, inplace=True) # rows: axis = 0
```

table5

	country	year	rate
1	Afghanistan	202000	2666/20595360
2	Brazil	191999	37737/172006362
3	Brazil	202000	80488/174504898
4	China	191999	212258/1272915272
5	China	202000	213766/1280428583

vs.

Questions?