

Reshaping and combining data

RESHAPING DATA WITH PANDAS



Maria Eugenia Inzaugarat
Data Scientist

Reshaping and statistical functions

sales

		office supply		Technology	
		online	onsite	online	onsite
country	year				
Italy	2017	310	123	510	340
	2018	110	100	610	120
Spain	2017	229	200	300	240
	2018	120	220	190	210

Statistical functions

- Sum: `.sum()`
- Mean: `.mean()`
- Median: `.median()`
- Difference: `.diff()`

Stacking and stats

- Total amount of online and on-site sales by year in the two countries

```
sales.stack().sum(axis=1)
```

country	year	shop	
Italy	2017	online	820
		onsite	463
	2018	online	720
		onsite	220
Spain	2017	online	529
		onsite	440
	2018	online	310
		onsite	430

Stacking and stats

- Total amount of online and on- site sales by year in the two countries

```
sales.stack().sum(axis=1).unstack()
```

	shop	online	onsite
country	year		
Italy	2017	820	463
	2018	720	220
Spain	2017	529	440
	2018	310	430

Unstacking and stats

- Mean amount of product sales by year in both countries

```
sales.unstack(level=0).mean(axis=1)
```

```
year
2017    281.5
2018    210.0
```

Unstacking and stats

- Difference in the amount of sales between years

```
sales["office supply"].unstack(level='country')
```

Unstacking and stats

- Difference in the amount of sales between years

```
sales["office supply"].unstack(level='country').diff(axis=1, periods=2)
```

	office		supply	
shop	online		onsite	
country	Italy	Spain	Italy	Spain
year				
2017	NaN	NaN	-187.0	-29.0
2018	NaN	NaN	-10.0	100.0

Reshaping and grouping

- Total amount of different products by online or on-site regardless of the country

```
sales.stack().head(4)
```

			office supply	Technology
country	year	shop		
Italy	2017	online	310	510
		onsite	123	340
	2018	online	110	610
		onsite	100	120

Reshaping and grouping

- Total amount of different products by online or on-site regardless of the country

```
sales.stack().groupby(level='shop').sum()
```

	office supply	Technology
shop		
online	769	1610
onsite	643	910

Reshaping after grouping

- Median amount of products by year

```
sales.groupby(level='year').median()
```

	office	supply		Technology	
shop	online	onsite		online	onsite
year					
2017	269.5	161.5		405.0	290.0
2018	115.0	160.0		400.0	165.0

Reshaping after grouping

- Median amount of products by year

```
sales.groupby(level=1).median().stack(level=[0, 1]).unstack(level='year')
```

	year	2017	2018
Technology	shop		
	online	405.0	400.0
office supply	onsite	290.0	165.0
	online	269.5	115.0
	onsite	161.5	160.0

Let's practice!
RESHAPING DATA WITH PANDAS

Transforming a list-like column

RESHAPING DATA WITH PANDAS



Maria Eugenia Inzaugarat
Data Scientist

List-like columns

	city	country	zip code
0	Los Angeles	USA	90001, 90004, 90008
1	Madrid	Spain	28001, 28004, 28005
2	Rabat	Morocco	10010, 10170

Transforming list-like columns

	city	country	zip code
0	Los Angeles	USA	90001, 90004, 90008
1	Madrid	Spain	28001, 28004, 28005
2	Rabat	Morocco	10010, 10170



	city	country	zip code
0	Los Angeles	USA	90001
1	Los Angeles	USA	90004
2	Los Angeles	USA	90008
3	Madrid	Spain	28001
4	Madrid	Spain	28004
5	Madrid	Spain	28005
6	Rabat	Morocco	10010
7	Rabat	Morocco	10170

The .explode() method

	city	country	zip code
0	Los Angeles	USA	90001, 90004, 90008
1	Madrid	Spain	28001, 28004, 28005
2	Rabat	Morocco	10010, 10170



	city	country	zip code
0	Los Angeles	USA	90001
1	Los Angeles	USA	90004
2	Los Angeles	USA	90008
3	Madrid	Spain	28001
4	Madrid	Spain	28004
5	Madrid	Spain	28005
6	Rabat	Morocco	10010
7	Rabat	Morocco	10170

`df.explode()`

Exploding a column

`cities`

```
   city  country  zip_code
0  Los Angeles    USA  [90001, 90004, 90008]
1    Madrid    Spain  [28001, 28004, 28005]
2    Rabat  Morocco  [10010, 10170]
```

Exploding a column

```
cities_explode = cities['zip_code'].explode()  
cities_explode
```

```
0    90001  
0    90004  
0    90008  
1    28001  
1    28004  
1    28005  
2    10010  
2    10170
```

Exploding a column

```
cities[['city', 'country']]
```

Exploding a column

```
cities[['city', 'country']].merge(cities_explode,
```

```
)
```

Exploding a column

```
cities[['city', 'country']].merge(cities_explode, left_index=True, right_index=True)
```

	city	country	zip_code
0	Los Angeles	USA	90001
0	Los Angeles	USA	90004
0	Los Angeles	USA	90008
1	Madrid	Spain	28001
1	Madrid	Spain	28004
1	Madrid	Spain	28005
2	Rabat	Morocco	10010
2	Rabat	Morocco	10170

Exploding a column in the DataFrame

```
cities_explode = cities.explode('zip_code')  
cities_explode
```

	city	country	zip_code
0	Los Angeles	USA	90001
0	Los Angeles	USA	90004
0	Los Angeles	USA	90008
1	Madrid	Spain	28001
1	Madrid	Spain	28004
1	Madrid	Spain	28005
2	Rabat	Morocco	10010
2	Rabat	Morocco	10170

Exploding a column in the DataFrame

```
cities_explode.reset_index(drop=True, inplace=True)
```

	city	country	zip_code
0	Los Angeles	USA	90001
1	Los Angeles	USA	90004
2	Los Angeles	USA	90008
3	Madrid	Spain	28001
4	Madrid	Spain	28004
5	Madrid	Spain	28005
6	Rabat	Morocco	10010
7	Rabat	Morocco	10170

Empty lists

```
cities_new
```

	city	country	zip_code
0	Los Angeles	USA	[90001, 90004, 90008]
1	Madrid	Spain	[]
2	Rabat	Morocco	[10010, 10170]

```
cities_new.explode('zip_code')
```

	city	country	zip_code
0	Los Angeles	USA	90001
0	Los Angeles	USA	90004
0	Los Angeles	USA	90008
1	Madrid	Spain	NaN
2	Rabat	Morocco	10010
2	Rabat	Morocco	10170

Chaining operations

```
cities
```

```
      city country      zip_code
0  Los Angeles    USA  90001, 90004, 90008
1    Madrid    Spain  28001, 28004, 28005
2    Rabat  Morocco    10010, 10170
```

Chaining operations

```
cities['zip_code'].str.split(',', expand=True)
```

	0	1	2
0	90001	90004	90008
1	28001	28004	28005
2	10010	10170	None

Chaining operations

```
cites.assign(zip_code=)
```

Chaining operations

```
cites.assign(zip_code=cities['zip_code'].str.split(','))
```

Chaining operations

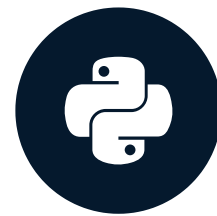
```
cites.assign(zip_code=cities['zip_code'].str.split(',')).explode('zip_code')
```

	city	country	zip_code
0	Los Angeles	USA	90001
0	Los Angeles	USA	90004
0	Los Angeles	USA	90008
1	Madrid	Spain	28001
1	Madrid	Spain	28004
1	Madrid	Spain	28005
2	Rabat	Morocco	10010
2	Rabat	Morocco	10170

Let's practice!
RESHAPING DATA WITH PANDAS

Reading nested data into a DataFrame

RESHAPING DATA WITH PANDAS



Maria Eugenia Inzaugarat
Data Scientist

Review

- Reshape DataFrames and Series
- Explode lists contained in columns
- Split and concatenate strings

JSON format

- JavaScript Object Notation
- Data-interchange format
- Easy for humans to read and write
- Easy for machines to parse and generate

JSON format

my_writer

```
{  
  "first" : "Mary",  
  "last" : "Shelley",  
  "country" : "England",  
  "books" : 12  
}
```

Nested JSON

writers

```
writers = [  
    {  
        "first": "Mary",  
        "last": "Shelley",  
        "books": {"title": "Frankenstein", "year": 1818}  
    },  
    {  
        "first": "Ernest",  
        "last": "Hemingway",  
        "books": {"title": "The Old Man and the Sea", "year": 1951}  
    }  
]
```

Data normalization

```
from pandas import json_normalize
```

```
json_normalize(writers)
```

	first	last	books.title	books.year
0	Mary	Shelley	Frankenstein	1818
1	Ernest	Hemingway	The Old Man and the Sea	1951

Data normalization

```
writers_norm = json_normalize(writers, sep='_')  
writers_norm
```

	first	last	books_title	books_year
0	Mary	Shelley	Frankenstein	1818
1	Ernest	Hemingway	The Old Man and the Sea	1951

Data normalization

```
pd.wide_to_long(writers_norm, stubnames=['books'], i=['first', 'last'], j='feature', sep='_', suffix='\w+')
```

```
books
first last  feature
Mary  Shelley title    Frankenstein
      year          1818
Ernest Hemingway title  The Old Man and the Sea
      year          1951
```

Complex JSON

writers

```
[
  {'name': 'Mary',
    'last': 'Shelley',
    'books': [{'title': 'Frankenstein', 'year': 1818},
               {'title': 'Mathilda ', 'year': 1819},
               {'title': 'The Last Man', 'year': 1826}]},
  {'name': 'Ernest',
    'last': 'Hemmingway',
    'books': [{'title': 'The Old Man and the Sea', 'year': 1951},
               {'title': 'The Sun Also Rises', 'year': 1927}]}
]
```


Complex JSON

```
json_normalize(writers)
```

	name	last	books
0	Mary	Shelley	[{'title': 'Frankenstein', 'year': 1818}, {'tit...
1	Ernest	Hemingway	[{'title': 'The Old Man and the Sea', 'year': ...

Record path

```
json_normalize(writers, record_path='books')
```

	title	year
0	Frankenstein	1818
1	Mathilda	1819
2	The Last Man	1826
3	The Old Man and the Sea	1951
4	The Sun Also Rises	1927

Metadata

```
json_normalize(writers, record_path='books', meta=['name', 'last'])
```

	title	year	name	last
0	Frankenstein	1818	Mary	Shelley
1	Mathilda	1819	Mary	Shelley
2	The Last Man	1826	Mary	Shelley
3	The Old Man and the Sea	1951	Ernest	Hemingway
4	The Sun Also Rises	1927	Ernest	Hemingway

Let's practice!
RESHAPING DATA WITH PANDAS

Dealing with nested data columns

RESHAPING DATA WITH PANDAS



Maria Eugenia Inzaugarat
Data Scientist

Review

- How to read nested JSON into DataFrame using `json_normalize()` .

Nested data in columns

```
writers = ["Mary Shelley", "Ernest Hemingway"]
books = ["{'title': 'Frankenstein', 'year': 1818}",
        "{'title': 'The Old Man and the Sea', 'year': 1951}"]
collection = pd.DataFrame()
```

Nested data in columns

```
writers = ["Mary Shelley", "Ernest Hemingway"]
books = ["{'title': 'Frankenstein', 'year': 1818}",
        "{'title': 'The Old Man and the Sea', 'year': 1951}"]
collection = pd.DataFrame(dict(
```


Nested data in columns

```
writers = ["Mary Shelley", "Ernest Hemingway"]
books = ['{"title": "Frankenstein", "year": "1818"}',
        '{"title": "The Old Man and the Sea", "year": "1951"}']
collection = pd.DataFrame(dict(writers=writers, books=books))
collection
```

	writers	books
0	Mary Shelley	{'title': 'Frankenstein', 'year': 1818}
1	Ernest Hemingway	{'title': 'The Old Man and the Sea', 'year': 1951}

Converting nested data

```
import json  
books = collection['books']
```

Converting nested data

```
import json  
books = collection['books'].apply(      )
```

Converting nested data

```
import json  
books = collection['books'].apply(json.loads)
```

Converting nested data

```
import json
books = collection['books'].apply(json.loads).apply(pd.Series)
books
```

```
      title  year
0  Frankenstein  1818
1  The Old Man and the Sea  1951
```

Concatenate back

```
collection = collection.drop(columns='books')  
pd.concat([collection, books], axis=1)
```

	writers	title	year
0	Mary Shelley	Frankenstein	1818
1	Ernest Hemingway	The Old Man and the Sea	1951

Dumping nested data

```
import json  
books = collection['books'].apply(json.loads)
```

Dumping nested data

```
import json
books = collection['books'].apply(json.loads).to_list()
books_dump = json.dumps(books)
new_books = pd.read_json(books_dump)
new_books
```

```
      title  year
0  Frankenstein  1818
1  The Old Man and the Sea  1951
```


Dumping nested data

```
pd.concat([collection['writers'], new_books], axis=1)
```

	writers	title	year
0	Mary Shelley	Frankenstein	1818
1	Ernest Hemingway	The Old Man and the Sea	1951

Let's practice!
RESHAPING DATA WITH PANDAS

The final reshape

RESHAPING DATA WITH PANDAS



Maria Eugenia Inzaugarat
Data Scientist

	Message
0	CON
1	GRA
2	TU
3	LA
4	TIONS



Message					
	CON	GRA	TU	LA	TIONS

Chapter 1

	Chapter 1	Chapter 2	Chapter 3	Chapter 4
0	Long & Wide	melt	stack	Reshape & Combining
1	pivot	wide_to_long	unstack	List-like col
2	pivot_table	string col	np.nan	Nested data

Concept of long and wide formats

Use `.pivot()` method - columns as unique variables, index as individual observations

Create pivot tables

Learn the difference between `.pivot()` and `.pivot_table()`

Chapter 2

	Chapter 1	Chapter 2	Chapter 3	Chapter 4
0	Long & Wide	melt	stack	Reshape & Combining
1	pivot	wide_to_long	unstack	List-like col
2	pivot_table	string col	np.nan	Nested data

From a wide to a long format using:

- the `.melt()` method
- the `wide_to_long()` function

Splitting or concatenating string columns

Chapter 3

	Chapter 1	Chapter 2	Chapter 3	Chapter 4
0	Long & Wide	melt	stack	Reshape & Combining
1	pivot	wide_to_long	unstack	List-like col
2	pivot_table	string col	np.nan	Nested data

Multi-level index

Use `.stack()` and `.unstack()`

Handle generated missing data

Chapter 4

	Chapter 1	Chapter 2	Chapter 3	Chapter 4
0	Long & Wide	melt	stack	Reshape & Combining
1	pivot	wide_to_long	unstack	List-like col
2	pivot_table	string col	np.nan	Nested data

Combine reshaping and grouping processes

List-like column transformation

Nested data in columns

Thank you!

RESHAPING DATA WITH PANDAS