Temperatures

June 2, 2020

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

0.0.1 Setting & removing indexes

pandas allows you to designate columns as an index. This enables cleaner code when taking subsets (as well as providing more efficient lookup under some circumstances).

In this chapter, you'll be exploring temperatures, a DataFrame of average temperatures in cities around the world. pandas is loaded as pd.

[3]:
```
temperatures = pd.read_csv('temperatures.csv')
temperatures
```

[3]:

	date	city	country	avg_temp_c
0	2000-01-01	Abidjan	Côte D'Ivoire	27.293
1	2000-02-01	Abidjan	Côte D'Ivoire	27.685
2	2000-03-01	Abidjan	Côte D'Ivoire	29.061
3	2000-04-01	Abidjan	Côte D'Ivoire	28.162
4	2000-05-01	Abidjan	Côte D'Ivoire	27.547
..	…	…	…	…
85	2007-07-01	Bangalore	India	24.967
86	2007-08-01	Bangalore	India	24.768
87	2007-09-01	Bangalore	India	24.933
88	2007-10-01	Bangalore	India	24.940
89	2007-11-01	Bangalore	India	23.917

[90 rows x 4 columns]

[4]:
```
# Set the index of temperatures to "city", assigning to temperatures_ind
temperatues_ind = temperatures.set_index("city")
temperatues_ind
```

[4]:

city	date	country	avg_temp_c
Abidjan	2000-01-01	Côte D'Ivoire	27.293

```
Abidjan    2000-02-01  Côte D'Ivoire      27.685
Abidjan    2000-03-01  Côte D'Ivoire      29.061
Abidjan    2000-04-01  Côte D'Ivoire      28.162
Abidjan    2000-05-01  Côte D'Ivoire      27.547
…                …             …          …
Bangalore  2007-07-01          India      24.967
Bangalore  2007-08-01          India      24.768
Bangalore  2007-09-01          India      24.933
Bangalore  2007-10-01          India      24.940
Bangalore  2007-11-01          India      23.917

[90 rows x 3 columns]
```

[5]:
```
# Reset the index of temperatures_ind, keeping its contents
temperatues_ind.reset_index()
```

[5]:
```
          city        date        country  avg_temp_c
0     Abidjan  2000-01-01  Côte D'Ivoire      27.293
1     Abidjan  2000-02-01  Côte D'Ivoire      27.685
2     Abidjan  2000-03-01  Côte D'Ivoire      29.061
3     Abidjan  2000-04-01  Côte D'Ivoire      28.162
4     Abidjan  2000-05-01  Côte D'Ivoire      27.547
..        …         …             …          …
85  Bangalore  2007-07-01          India      24.967
86  Bangalore  2007-08-01          India      24.768
87  Bangalore  2007-09-01          India      24.933
88  Bangalore  2007-10-01          India      24.940
89  Bangalore  2007-11-01          India      23.917

[90 rows x 4 columns]
```

[6]:
```
# Reset the index of temperatures_ind, dropping its contents.
temperatues_ind.reset_index(drop = True)
```

[6]:
```
           date        country  avg_temp_c
0   2000-01-01  Côte D'Ivoire      27.293
1   2000-02-01  Côte D'Ivoire      27.685
2   2000-03-01  Côte D'Ivoire      29.061
3   2000-04-01  Côte D'Ivoire      28.162
4   2000-05-01  Côte D'Ivoire      27.547
..         …             …          …
85  2007-07-01          India      24.967
86  2007-08-01          India      24.768
87  2007-09-01          India      24.933
88  2007-10-01          India      24.940
89  2007-11-01          India      23.917
```

```
[90 rows x 3 columns]
```

0.0.2 Subsetting with .loc[]

The killer feature for indexes is .loc[]: a subsetting method that accepts index values. When you pass it a single argument, it will take a subset of rows.

The code for subsetting using .loc[] can be easier to read than standard square bracket subsetting, which can make your code less burdensome to maintain.

pandas is loaded as pd. temperatures and temperatures_ind are available; the latter is indexed by city.

[7]:
```
temperatues_ind
```

[7]:

city	date	country	avg_temp_c
Abidjan	2000-01-01	Côte D'Ivoire	27.293
Abidjan	2000-02-01	Côte D'Ivoire	27.685
Abidjan	2000-03-01	Côte D'Ivoire	29.061
Abidjan	2000-04-01	Côte D'Ivoire	28.162
Abidjan	2000-05-01	Côte D'Ivoire	27.547
…	…	…	…
Bangalore	2007-07-01	India	24.967
Bangalore	2007-08-01	India	24.768
Bangalore	2007-09-01	India	24.933
Bangalore	2007-10-01	India	24.940
Bangalore	2007-11-01	India	23.917

[90 rows x 3 columns]

[8]:
```
# Create a list of cities to subset on: Baghdad and Bangalore. Assign to cities
cities = ['Baghdad', 'Bangalore']
# Use [] subsetting to filter temperatures for rows where the city column takes␣
 ↪a value in cities
temperatures[temperatures['city'].isin(cities)]
```

[8]:

	date	city	country	avg_temp_c
20	2000-11-01	Baghdad	Iraq	16.577
21	2000-12-01	Baghdad	Iraq	12.108
22	2001-01-01	Baghdad	Iraq	10.627
23	2001-02-01	Baghdad	Iraq	12.727
24	2001-03-01	Baghdad	Iraq	19.097
..	…	…	…	…
85	2007-07-01	Bangalore	India	24.967
86	2007-08-01	Bangalore	India	24.768
87	2007-09-01	Bangalore	India	24.933

```
88  2007-10-01  Bangalore   India      24.940
89  2007-11-01  Bangalore   India      23.917

[70 rows x 4 columns]
```

[9]:
```
# Use .loc[] subsetting to filter temperatures_ind for rows where the city is␣
↪in cities
temperatues_ind.loc[cities]
```

[9]:
```
                date country  avg_temp_c
city
Baghdad    2000-11-01    Iraq      16.577
Baghdad    2000-12-01    Iraq      12.108
Baghdad    2001-01-01    Iraq      10.627
Baghdad    2001-02-01    Iraq      12.727
Baghdad    2001-03-01    Iraq      19.097
…                 …       …          …
Bangalore  2007-07-01   India      24.967
Bangalore  2007-08-01   India      24.768
Bangalore  2007-09-01   India      24.933
Bangalore  2007-10-01   India      24.940
Bangalore  2007-11-01   India      23.917

[70 rows x 3 columns]
```

0.0.3 Setting multi-level indexes

Indexes can also be made out of multiple columns, forming a multi-level index (sometimes called a hierarchical index). There is a trade-off to using these.

The benefit is that multi-level indexes make it more natural to reason about nested categorical variables. For example, in a clinical trial you might have control and treatment groups. Then each test subject belongs to one or other group, and we can say that test subject is nested inside treatment group. Similarly, in the temperature dataset, the city is located in the country, so we can say city is nested inside country.

The main downside is that the code for manipulating indexes is different to the code for the manipulating columns, so you have to learn two syntaxes, and keep track of how your data is represented.

pandas is loaded as pd. temperatures is available.

[11]:
```
# Set the index of temperatures to the "country" and "city" columns, assigning␣
↪to temperatures_ind
temperatues_ind = temperatures.set_index(["country", "city"])
temperatues_ind
```

[11]:
```
                              date  avg_temp_c
country       city
Côte D'Ivoire Abidjan    2000-01-01      27.293
              Abidjan    2000-02-01      27.685
              Abidjan    2000-03-01      29.061
              Abidjan    2000-04-01      28.162
              Abidjan    2000-05-01      27.547
…                              …           …
India         Bangalore  2007-07-01      24.967
              Bangalore  2007-08-01      24.768
              Bangalore  2007-09-01      24.933
              Bangalore  2007-10-01      24.940
              Bangalore  2007-11-01      23.917

[90 rows x 2 columns]
```

[14]:
```
# Specify two country/city pairs to keep: India/Bangalore and Iraq/Baghdad,␣
↪assigning to rows_to_keep.
rows_to_keep = [("India", "Bangalore"),("Iraq", "Baghdad")] #outer level, inner␣
↪level
```

[13]:
```
# Subset for rows_to_keep using .loc[]
temperatues_ind.loc[rows_to_keep]
```

[13]:
```
                         date  avg_temp_c
country city
India   Bangalore  2003-10-01      25.116
        Bangalore  2003-11-01      24.158
        Bangalore  2003-12-01      23.068
        Bangalore  2004-01-01      23.564
        Bangalore  2004-02-01      25.210
…                         …           …
Iraq    Baghdad    2002-02-01      13.269
        Baghdad    2002-03-01      18.289
        Baghdad    2002-04-01      21.785
        Baghdad    2002-05-01      28.945
        Baghdad    2002-06-01      34.071

[70 rows x 2 columns]
```

0.0.4 Sorting by index values

Previously, you changed the order of the rows in a DataFrame by calling .sort_values(). It's also useful to be able to sort by elements in the index. For this, you need to use .sort_index().

pandas is loaded as pd. temperatures_ind has a multi-level index of country and city, and is available.

[15]:
```
# Sort temperatures_ind by the index values
temperatues_ind.sort_index() #sort by outer level
```

[15]:

country	city	date	avg_temp_c
Côte D'Ivoire	Abidjan	2000-01-01	27.293
	Abidjan	2000-02-01	27.685
	Abidjan	2000-03-01	29.061
	Abidjan	2000-04-01	28.162
	Abidjan	2000-05-01	27.547
…	…	…	…
Iraq	Baghdad	2002-02-01	13.269
	Baghdad	2002-03-01	18.289
	Baghdad	2002-04-01	21.785
	Baghdad	2002-05-01	28.945
	Baghdad	2002-06-01	34.071

[90 rows x 2 columns]

[18]:
```
# Sort temperatures_ind by the index values at the "city" level.
temperatues_ind.sort_index(level = "city")
```

[18]:

country	city	date	avg_temp_c
Côte D'Ivoire	Abidjan	2000-01-01	27.293
	Abidjan	2000-02-01	27.685
	Abidjan	2000-03-01	29.061
	Abidjan	2000-04-01	28.162
	Abidjan	2000-05-01	27.547
…	…	…	…
India	Bangalore	2007-07-01	24.967
	Bangalore	2007-08-01	24.768
	Bangalore	2007-09-01	24.933
	Bangalore	2007-10-01	24.940
	Bangalore	2007-11-01	23.917

[90 rows x 2 columns]

[26]:
```
# Sort temperatures_ind by ascending country then descending city
temperatues_ind.sort_index(level = ["country","city"], ascending=[True,False])␣
↪#First sort country and then city
```

[26]:

country	city	date	avg_temp_c
Côte D'Ivoire	Abidjan	2000-01-01	27.293
	Abidjan	2000-02-01	27.685
	Abidjan	2000-03-01	29.061

```
                Abidjan  2000-04-01      28.162
                Abidjan  2000-05-01      27.547
…                              …          …
Iraq            Baghdad  2002-02-01      13.269
                Baghdad  2002-03-01      18.289
                Baghdad  2002-04-01      21.785
                Baghdad  2002-05-01      28.945
                Baghdad  2002-06-01      34.071

[90 rows x 2 columns]
```

[ ]: