Effective Programming Practices for Economists

Data management with pandas

Setting and renaming columns and indices

Janoś Gabler and Hans-Martin von Gaudecker

Why the Index is important

The dataframe from before

	country	continent	year	life_exp
0	Cuba	Americas	2002	77.16
1	Cuba	Americas	2007	78.27
2	Spain	Europe	2002	79.78
3	Spain	Europe	2007	80.94

Same dataset, different Index

		continent	life_exp
country	year		
Cuba	2002	Americas	77.16
	2007	Americas	78.27
Spain	2002	Europe	79.78

- We have seen that pandas aligns new columns in a DataFrame by index
- Many other operations are aligned by index
- Using a meaningful index makes this even safer
- Index should be unique and not contain floats

Setting and resetting the index

assume `df` is our gapminder example

```
>>> df.index
RangeIndex(start=0, stop=4, step=1)

>>> df = df.set_index(["country", "year"])
>>> df.index
```

```
MultiIndex([( 'Cuba', 2002),
            ( 'Cuba', 2007),
            ('Spain', 2002),
            ('Spain', 2007)],
           names=['country', 'year'])
```

```
>>> df = df.reset_index()
>>> df.index
```

- `set_index` and `reset_index` are inverse functions
- `set_index` can take any column or list of columns
- Optional argument `drop=True` or `drop=False` determines what happens with the old index in `set_index`

Renaming columns

assume `df` is our gapminder example

```
>>> df.columns
```

```
Index(['country', 'continent', 'year',
 'life_exp'], dtype='string')
```

```
>>> new_names = {
...     "life_exp": "life expectancy",
...     "country": "country name",
...     "continent": "continent name",
... }

>>> df = df.rename(columns=new_names)
>>> df.columns
```

```
Index(['country name', 'continent name',
```

- Dict can contain only the subset of variables that is actually renamed
- Renaming the index works the same way but is rarely needed
- Instead of a dict, you can provide a function that converts old names to new names!