



CLEANING DATA IN PYTHON

Concatenating data

Combining data

- Data may not always come in 1 huge file
 - 5 million row dataset may be broken into 5 separate datasets
 - Easier to store and share
 - May have new data for each day
- Important to be able to combine then clean, or vice versa



Concatenation

| | date | element | value |
|---|------------|---------|-------|
| 0 | 2010-01-30 | tmax | 27.8 |
| 1 | 2010-01-30 | tmin | 14.5 |

| | date | element | value |
|---|------------|---------|-------|
| 0 | 2010-02-02 | tmax | 27.3 |
| 1 | 2010-02-02 | tmin | 14.4 |

| | date | element | value |
|---|------------|---------|-------|
| 0 | 2010-01-30 | tmax | 27.8 |
| 1 | 2010-01-30 | tmin | 14.5 |
| 0 | 2010-02-02 | tmax | 27.3 |
| 1 | 2010-02-02 | tmin | 14.4 |



pandas concat

```
In [1]: concatenated = pd.concat([weather_p1, weather_p2])
```

```
In [2]: print(concatenated)
```

| | date | element | value |
|---|------------|---------|-------|
| 0 | 2010-01-30 | tmax | 27.8 |
| 1 | 2010-01-30 | tmin | 14.5 |
| 0 | 2010-02-02 | tmax | 27.3 |
| 1 | 2010-02-02 | tmin | 14.4 |



pandas concat

```
In [3]: concatenated = concatenated.loc[0, :]
```

```
Out[3]:
```

| | date | element | value |
|---|------------|---------|-------|
| 0 | 2010-01-30 | tmax | 27.8 |
| 0 | 2010-02-02 | tmax | 27.3 |



pandas concat

```
In [4]: pd.concat([weather_p1, weather_p2], ignore_index=True)
```

```
Out[4]:
```

| | date | element | value |
|---|------------|---------|-------|
| 0 | 2010-01-30 | tmax | 27.8 |
| 1 | 2010-01-30 | tmin | 14.5 |
| 2 | 2010-02-02 | tmax | 27.3 |
| 3 | 2010-02-02 | tmin | 14.4 |



Concatenating DataFrames

| | country | year | variable | value |
|---|---------|------|----------|-------|
| 0 | AD | 2000 | m014 | 0 |
| 1 | AE | 2000 | m014 | 2 |
| 2 | AF | 2000 | m014 | 52 |
| 3 | AD | 2000 | m1524 | 0 |
| 4 | AE | 2000 | m1524 | 4 |
| 5 | AF | 2000 | m1524 | 228 |

| | age_group | sex |
|---|-----------|-----|
| 0 | 014 | m |
| 1 | 014 | m |
| 2 | 014 | m |
| 3 | 1524 | m |
| 4 | 1524 | m |
| 5 | 1524 | m |



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Finding and concatenating data

Concatenating many files

- Leverage Python's features with data cleaning in pandas
- In order to concatenate DataFrames:
 - They must be in a list
 - Can individually load if there are a few datasets
 - But what if there are thousands?
- Solution: glob function to find files based on a pattern



Globbing

- Pattern matching for file names
- Wildcards: * ?
 - Any csv file: *.csv
 - Any single character: file_?.csv
- Returns a list of file names
- Can use this list to load into separate DataFrames



The plan

- Load files from globbing into pandas
- Add the DataFrames into a list
- Concatenate multiple datasets at once



Find and concatenate

```
In [1]: import glob
```

```
In [2]: csv_files = glob.glob('*.*csv')
```

```
In [3]: print(csv_files)
```

```
['file5.csv', 'file2.csv', 'file3.csv', 'file1.csv', 'file4.csv']
```



Using loops

```
In [4]: list_data = []
```

```
In [5]: for filename in csv_files:
...:     data = pd.read_csv(filename)
...:     list_data.append(filename)
```

```
In [6]: pd.concat(list_data)
```



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Merge data



Combining data

- Concatenation is not the only way data can be combined

| | state | population_2016 |
|---|------------|-----------------|
| 0 | California | 39250017 |
| 1 | Texas | 27862596 |
| 2 | Florida | 20612439 |
| 3 | New York | 19745289 |

| | name | ANSI |
|---|------------|------|
| 0 | California | CA |
| 1 | Florida | FL |
| 2 | New York | NY |
| 3 | Texas | TX |



Merging data

- Similar to joining tables in SQL
- Combine disparate datasets based on common columns

| | state | population_2016 |
|---|------------|-----------------|
| 0 | California | 39250017 |
| 1 | Texas | 27862596 |
| 2 | Florida | 20612439 |
| 3 | New York | 19745289 |

| | name | ANSI |
|---|------------|------|
| 0 | California | CA |
| 1 | Florida | FL |
| 2 | New York | NY |
| 3 | Texas | TX |



Merging data

```
In [1]: pd.merge(left=state_populations, right=state_codes,  
....:            on=None, left_on='state', right_on='name')
```

```
Out[1]:
```

| | state | population_2016 | name | ANSI |
|---|------------|-----------------|------------|------|
| 0 | California | 39250017 | California | CA |
| 1 | Texas | 27862596 | Texas | TX |
| 2 | Florida | 20612439 | Florida | FL |
| 3 | New York | 19745289 | New York | NY |



Types of merges

- One-to-one
- Many-to-one / one-to-many
- Many-to-many



One-to-one

| | state | population_2016 |
|---|------------|-----------------|
| 0 | California | 39250017 |
| 1 | Texas | 27862596 |
| 2 | Florida | 20612439 |
| 3 | New York | 19745289 |

| | name | ANSI |
|---|------------|------|
| 0 | California | CA |
| 1 | Florida | FL |
| 2 | New York | NY |
| 3 | Texas | TX |



One-to-one

| | state | population_2016 | name | ANSI |
|----------|------------|-----------------|------------|------|
| 0 | California | 39250017 | California | CA |
| 1 | Texas | 27862596 | Texas | TX |
| 2 | Florida | 20612439 | Florida | FL |
| 3 | New York | 19745289 | New York | NY |



Many-to-one / one-to-many

| | state | City |
|----------|------------|---------------|
| 0 | California | San Diego |
| 1 | California | Sacramento |
| 2 | New York | New York City |
| 3 | New York | Albany |

| | name | ANSI |
|----------|------------|------|
| 0 | California | CA |
| 1 | Florida | FL |
| 2 | New York | NY |
| 3 | Texas | TX |



Many-to-one / one-to-many

| | name | ANSI | state | City |
|---|------------|------|------------|---------------|
| 0 | California | CA | California | San Diego |
| 1 | California | CA | California | Sacramento |
| 2 | New York | NY | New York | New York City |
| 3 | New York | NY | New York | Albany |



Different types of merges

- One-to-one
- Many-to-one
- Many-to-many
- All use the same function
- Only difference is the DataFrames you are merging



CLEANING DATA IN PYTHON

Let's practice!