

CLEANING DATA IN PYTHON

Diagnose data for cleaning



Cleaning data

- Prepare data for analysis
- Data almost never comes in clean
- Diagnose your data for problems

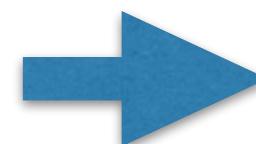


Common data problems

- Inconsistent column names
- Missing data
- Outliers
- Duplicate rows
- Untidy
- Need to process columns
- Column types can signal unexpected data values



Unclean data



	Continent	Country	female literacy	fertility	population
0	ASI	Chine	90.5	1.769	1.324655e+09
1	ASI	Inde	50.8	2.682	1.139965e+09
2	NAM	USA	99.0	2.077	3.040600e+08
3	ASI	Indonésie	88.8	2.132	2.273451e+08
4	LAT	Brésil	90.2	1.827	NaN

- Column name inconsistencies
- Missing data
- Country names are in French



Load your data

```
In [1]: import pandas as pd
```

```
In [2]: df = pd.read_csv('literary_birth_rate.csv')
```



Visually inspect

```
In [3]: df.head()
```

```
Out[3]:
```

```
   Continent    Country  female literacy  fertility  population
0      ASI        Chine          90.5       1.769  1.324655e+09
1      ASI         Inde          50.8       2.682  1.139965e+09
2     NAM         USA          99.0       2.077  3.040600e+08
3      ASI  Indonésie          88.8       2.132  2.273451e+08
4     LAT        Brésil          90.2       1.827           NaN
```

```
In [4]: df.tail()
```

```
Out[4]:
```

```
   Continent    Country  female literacy  fertility  population
0       AF  Sao Tomé-et-Principe          90.5       1.769  1.324655e+09
1     LAT          Aruba          50.8       2.682  1.139965e+09
2     ASI          Tonga          99.0       2.077  3.040600e+08
3     OCE        Australia          88.8       2.132  2.273451e+08
4     OCE          Sweden          90.2       1.827           NaN
```



Visually inspect

```
In [5]: df.columns  
Out[5]: Index(['Continent', 'Country', 'female literacy',  
'fertility', 'population'], dtype='object')
```

```
In [6]: df.shape  
Out[6]: (164, 5)
```

```
In [7]: df.info()  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 164 entries, 0 to 163  
Data columns (total 5 columns):  
 Continent           164 non-null object  
 Country             164 non-null object  
 female literacy    164 non-null float64  
 fertility           164 non-null object  
 population          122 non-null float64  
 dtypes float64(2), object(3)  
 memory usage: 6.5+ KB
```



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Exploratory data analysis



Frequency counts

- Count the number of unique values in our data



Data type of each column

```
In [1]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 5 columns):
continent           164 non-null object
country              164 non-null object
female literacy      164 non-null float64
fertility             164 non-null object
population            122 non-null float64
dtypes float64(2), object(3)
memory usage: 6.5+ KB
```



Frequency counts: continent

```
In [2]: df.continent.value_counts(dropna=False)
Out[2]:
AF      49
ASI     47
EUR     36
LAT     24
OCE      6
NAM      2
Name: continent, dtype: int64
```



Frequency counts: continent

```
In [3]: df['continent'].value_counts(dropna=False)
Out[3]:
AF      49
ASI     47
EUR     36
LAT     24
OCE      6
NAM      2
Name: continent, dtype: int64
```



Frequency counts: country

```
In [4]: df.country.value_counts(dropna=False).head()  
Out[4]:  
Sweden    2  
Algerie   1  
Germany   1  
Angola    1  
Indonésie 1  
Name: country, dtype: int64
```



Frequency counts: fertility

```
In [5]: df.fertility.value_counts(dropna=False).head()  
Out[5]:  
missing    5  
1.854     2  
1.93       2  
1.841     2  
1.393     2  
Name: fertility, dtype: int64
```



Frequency counts: population

```
In [6]: df.population.value_counts(dropna=False).head()
```

```
Out[6]:
```

```
NaN          42  
5.667325e+06    1  
3.773100e+06    1  
1.333388e+06    1  
1.661115e+08    1
```

```
Name: population, dtype: int64
```



Summary statistics

- Numeric columns
- Outliers
 - Considerably higher or lower
 - Require further investigation



Summary statistics: Numeric data

```
In [7]: df.describe()
```

```
Out[7]:
```

	female_literacy	population
count	164.000000	1.220000e+02
mean	80.301220	6.345768e+07
std	22.977265	2.605977e+08
min	12.600000	1.035660e+05
25%	66.675000	3.778175e+06
50%	90.200000	9.995450e+06
75%	98.500000	2.642217e+07
max	100.000000	2.313000e+09



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Visual exploratory data analysis



Data visualization

- Great way to spot outliers and obvious errors
- More than just looking for patterns
- Plan data cleaning steps



Summary statistics

```
In [1]: df.describe()
```

```
Out[1]:
```

	female_literacy	fertility	population
count	164.000000	163.000000	1.220000e+02
mean	80.301220	2.872853	6.345768e+07
std	22.977265	1.425122	2.605977e+08
min	12.600000	0.966000	1.035660e+05
25%	66.675000	1.824500	3.778175e+06
50%	90.200000	2.362000	9.995450e+06
75%	98.500000	3.877500	2.642217e+07
max	100.000000	7.069000	2.313000e+09



Bar plots and histograms

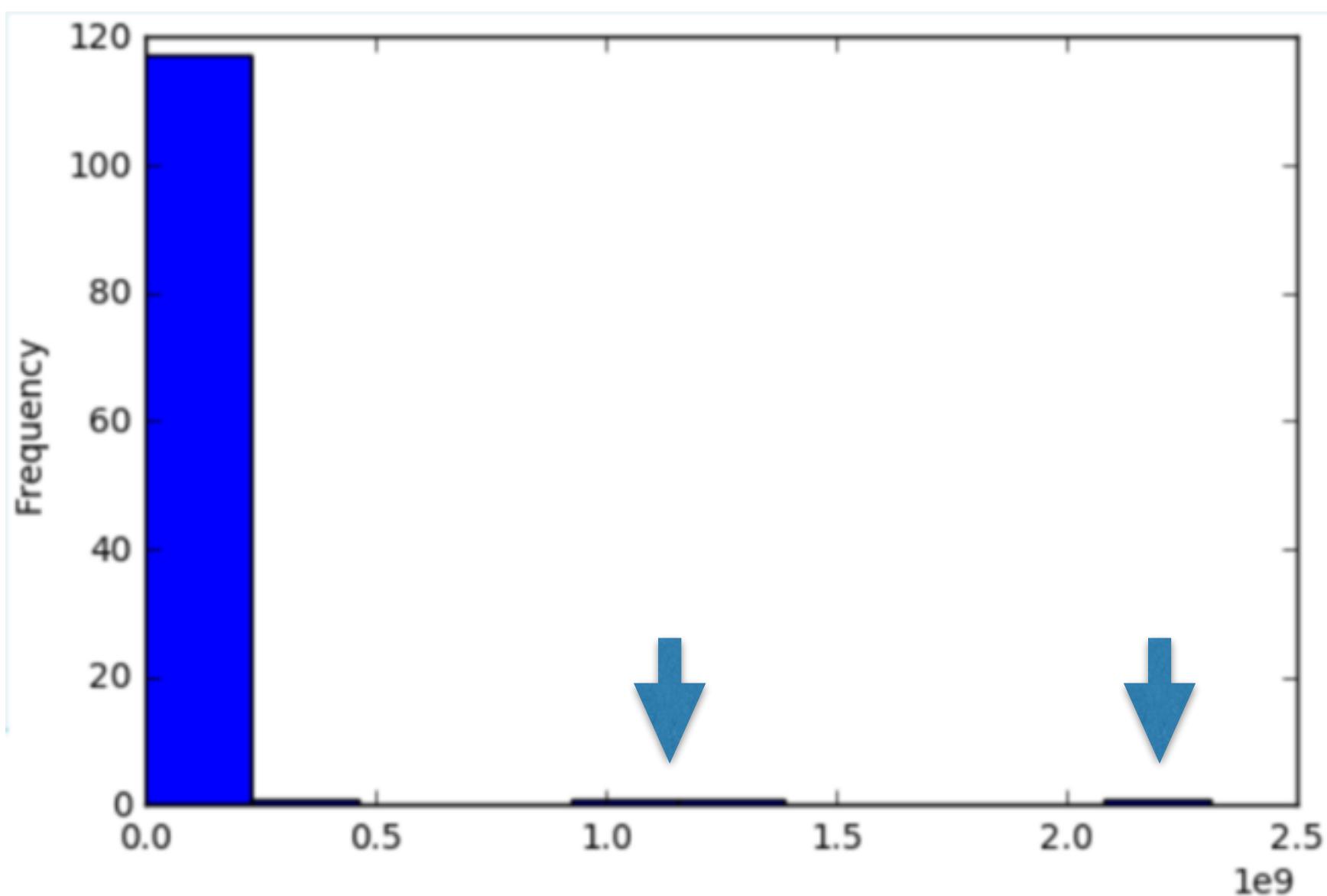
- Bar plots for discrete data counts
- Histograms for continuous data counts
- Look at frequencies

Histogram

```
In [2]: df.population.plot('hist')  
Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x7f78e4abaf0>
```

```
In [3]: import matplotlib.pyplot as plt
```

```
In [4]: plt.show()
```





Identifying the error

```
In [5]: df[df.population > 1000000000]
```

```
Out[5]:
```

```
continent    country   female literacy  fertility  population
0          ASI        Chine       90.5      1.769  1.324655e+09
1          ASI        Inde        50.8      2.682  1.139965e+09
162         OCE    Australia      96.0      1.930  2.313000e+09
```

- Not all outliers are bad data points
- Some can be an error, but others are valid values



Box plots

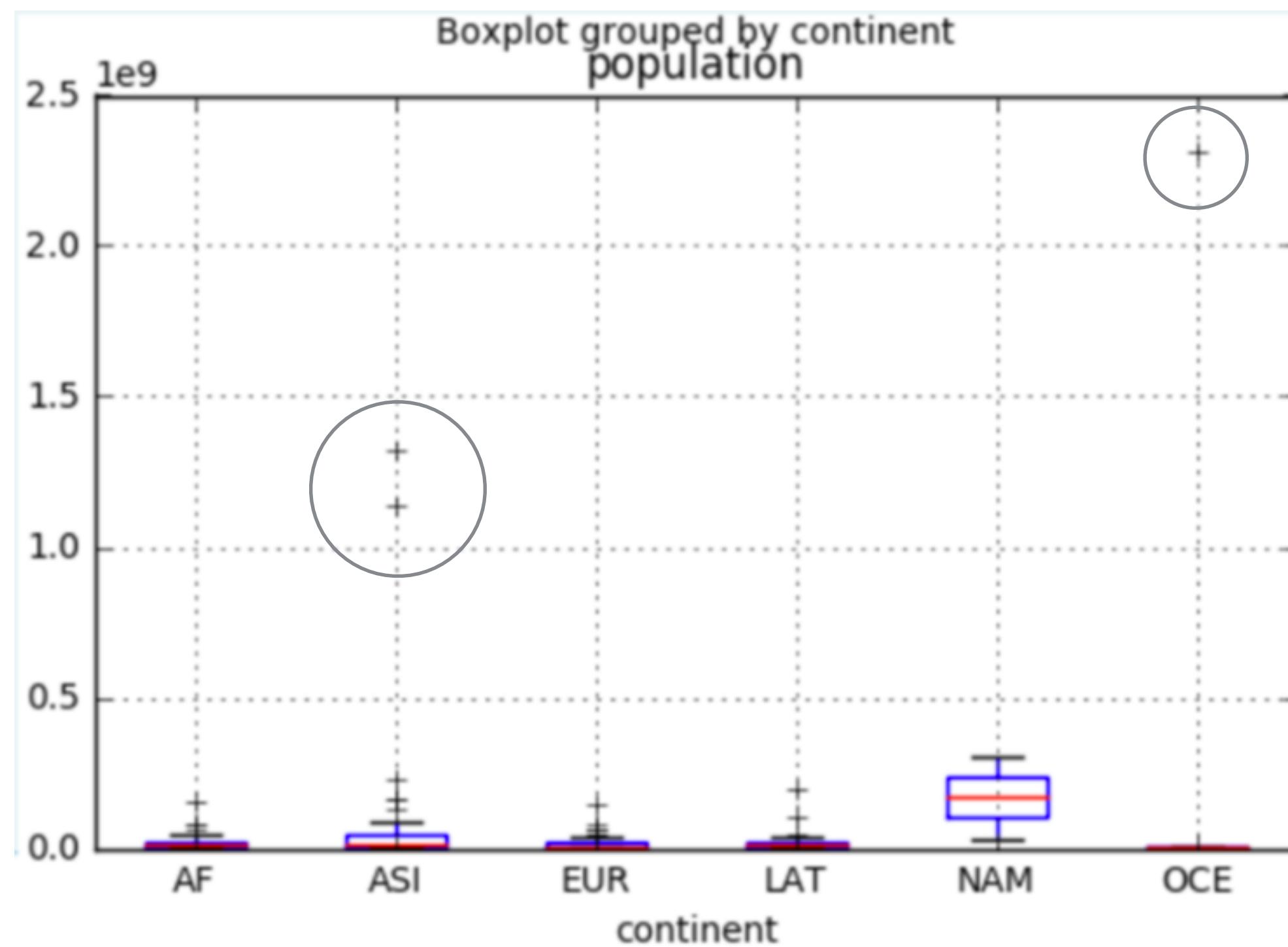
- Visualize basic summary statistics
 - Outliers
 - Min/max
 - 25th, 50th, 75th percentiles



Box plot

```
In [6]: df.boxplot(column='population', by='continent')
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff5581bb630>
```

```
In [7]: plt.show()
```





Scatter plots

- Relationship between 2 numeric variables
- Flag potentially bad data
 - Errors not found by looking at 1 variable



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Tidy data



Tidy data

- “Tidy Data” paper by Hadley Wickham, PhD
- Formalize the way we describe the shape of data
- Gives us a goal when formatting our data
- “Standard way to organize data values within a dataset”



Motivation for tidy data

	name	treatment a	treatment b
0	Daniel	-	42
1	John	12	31
2	Jane	24	27

	0	1	2
name	Daniel	John	Jane
treatment a	-	12	24
treatment b	42	31	27



Principles of tidy data

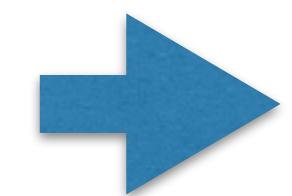
- Columns represent separate variables
- Rows represent individual observations
- Observational units form tables

	name	treatment a	treatment b
0	Daniel	-	42
1	John	12	31
2	Jane	24	27



Converting to tidy data

	name	treatment a	treatment b
0	Daniel	-	42
1	John	12	31
2	Jane	24	27



	name	treatment	value
0	Daniel	treatment a	-
1	John	treatment a	12
2	Jane	treatment a	24
3	Daniel	treatment b	42
4	John	treatment b	31
5	Jane	treatment b	27

- Better for reporting vs. better for analysis
- Tidy data makes it easier to fix common data problems



Converting to tidy data

- The data problem we are trying to fix:
 - Columns containing values, instead of variables
- Solution: pd.melt()



Melting

```
In [1]: pd.melt(frame=df, id_vars='name',
...:                 value_vars=['treatment a', 'treatment b'])
```

Out[1]:

	name	variable	value
0	Daniel	treatment a	_
1	John	treatment a	12
2	Jane	treatment a	24
3	Daniel	treatment b	42
4	John	treatment b	31
5	Jane	treatment b	27



Melting

```
In [2]: pd.melt(frame=df, id_vars='name',
...:             value_vars=['treatment a', 'treatment b'],
...:             var_name='treatment', value_name='result')
```

Out[2]:

	name	treatment	result
0	Daniel	treatment a	_
1	John	treatment a	12
2	Jane	treatment a	24
3	Daniel	treatment b	42
4	John	treatment b	31
5	Jane	treatment b	27



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Pivoting data



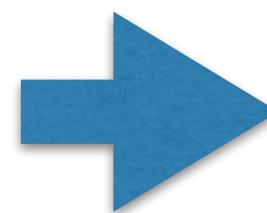
Pivot: un-melting data

- Opposite of melting
- In melting, we turned columns into rows
- Pivoting: turn unique values into separate columns
- Analysis friendly shape to reporting friendly shape
- Violates tidy data principle: rows contain observations
 - Multiple variables stored in the same column



Pivot: un-melting data

	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



element	tmax	tmin
date		
2010-01-30	27.8	14.5
2010-02-02	27.3	14.4



Pivot

```
In [2]: print(weather_tidy)
element      tmax  tmin
date
2010-01-30    27.8  14.5
2010-02-02    27.3  14.4
```



Pivot

	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

	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
4	2010-02-02	tmin	16.4



Using pivot when you have duplicate entries

```
In [3]: import numpy as np
```

```
In [4]: weather2_tidy = weather.pivot(values='value',
...:                                     index='date',
...:                                     columns='element')
```

```
Out[4]:
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-9-2962bb23f5a3> in <module>()  
      1 weather2_tidy = weather2.pivot(values='value',  
      2                                         index='date',  
----> 3                                         columns='element')  
ValueError: Index contains duplicate entries, cannot reshape
```



Pivot table

- Has a parameter that specifies how to deal with duplicate values
- Example: Can aggregate the duplicate values by taking their average



Pivot table



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Beyond melt and pivot



Beyond melt and pivot

- Melting and pivoting are basic tools
- Another common problem:
 - Columns contain multiple bits of information



Beyond melt and pivot

	country	year	m014	m1524
0	AD	2000	0	0
1	AE	2000	2	4
2	AF	2000	52	228



Melting and parsing

```
In [1]: pd.melt(frame=tb, id_vars=['country', 'year'])
```

```
Out[1]:
```

	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

- Nothing inherently wrong about original data shape
- Not conducive for analysis



Melting and parsing

```
In [2]: tb_melt['sex'] = tb_melt.variable.str[0]
```

```
In [3]: tb_melt
```

```
Out[3]:
```

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



CLEANING DATA IN PYTHON

Let's practice!



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(data)

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!



CLEANING DATA IN PYTHON

Data types



Prepare and clean data

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27



Data types

```
In [1]: print(df.dtypes)
name          object
sex           object
treatment a   object
treatment b   int64
dtype: object
```

- There may be times we want to convert from one type to another
 - Numeric columns can be strings, or vice versa



Converting data types

```
In [2]: df['treatment b'] = df['treatment b'].astype(str)
```

```
In [3]: df['sex'] = df['sex'].astype('category')
```

```
In [4]: df.dtypes
```

```
Out[4]:
```

name	object
sex	category
treatment a	object
treatment b	object
dtype:	object



Categorical data

- Converting categorical data to ‘category’ dtype:
 - Can make the DataFrame smaller in memory
 - Can make them be utilized by other Python libraries for analysis



Cleaning data

- Numeric data loaded as a string

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27



Cleaning bad data

```
In [5]: df['treatment a'] = pd.to_numeric(df['treatment a'],
...:                                         errors='coerce')
```

```
In [6]: df.dtypes
```

```
Out[6]:
```

name	object
sex	category
treatment a	float64
treatment b	object
dtype:	object



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Using regular expressions to clean strings



String manipulation

- Much of data cleaning involves string manipulation
 - Most of the world's data is unstructured text
- Also have to do string manipulation to make datasets consistent with one another



Validate values

- 17
- \$17
- \$17.89
- \$17.895



String manipulation

- Many built-in and external libraries
- ‘re’ library for regular expressions
 - A formal way of specifying a pattern
 - Sequence of characters
- Pattern matching
 - Similar to globbing



Example match

- 17 12345678901 $\backslash d^*$
- \$17 \$12345678901 $\backslash \$ \backslash d^*$
- \$17.00 \$12345678901.42 $\backslash \$ \backslash d^* \backslash . \backslash d^*$
- \$17.89 \$12345678901.24 $\backslash \$ \backslash d^* \backslash . \backslash d\{2\}$
- ~~\$17.895~~ ~~\$12345678901.999~~ $^\wedge \backslash \$ \backslash d^* \backslash . \backslash d\{2\} \$$

“I have 17.89 USD”



Using regular expressions

- Compile the pattern
- Use the compiled pattern to match values
- This lets us use the pattern over and over again
- Useful since we want to match values down a column of values



Using regular expressions

```
In [1]: import re  
  
In [2]: pattern = re.compile('\$\d*\.\d{2}')  
  
In [3]: result = pattern.match('$17.89')  
  
In [4]: bool(result)  
True
```



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Using functions to clean data



Complex cleaning

- Cleaning step requires multiple steps
 - Extract number from string
 - Perform transformation on extracted number
- Python function



Apply

```
In [1]: print(df)
      treatment a    treatment b
Daniel            18                  42
John              12                  31
Jane              24                  27
```

```
In [2]: df.apply(np.mean, axis=0)
Out[2]:
treatment a    18.000000
treatment b    33.333333
dtype: float64
```



Apply

```
In [3]: print(df)
      treatment a    treatment b
Daniel          18              42
John           12              31
Jane           24              27
```

```
In [4]: df.apply(np.mean, axis=1)
Out[4]:
Daniel    30.0
John     21.5
Jane     25.5
dtype: float64
```



Applying functions

	Job #	Doc #	Borough	Initial Cost	Total Est. Fee
0	121577873	2	MANHATTAN	\$75000.00	\$986.00
1	520129502	1	STATEN ISLAND	\$0.00	\$1144.00
2	121601560	1	MANHATTAN	\$30000.00	\$522.50
3	121601203	1	MANHATTAN	\$1500.00	\$225.00
4	121601338	1	MANHATTAN	\$19500.00	\$389.50



Write the regular expression

```
In [5]: import re  
  
In [6]: from numpy import NaN  
  
In [7]: pattern = re.compile('^\$\d*\.\d{2}$')
```



Writing a function

example.py

```
def my_function(input1, input2):  
    # Function Body  
    return value
```



Write the function

diff_money.py

```
def diff_money(row, pattern):  
  
    icost = row['Initial Cost']  
    tef = row['Total Est. Fee']  
  
    if bool(pattern.match(icost)) and bool(pattern.match(tef)):  
  
        icost = icost.replace("$", "")  
        tef = tef.replace("$", "")  
  
        icost = float(icost)  
        tef = float(tef)  
  
    return icost - tef  
else:  
  
    return(NaN)
```



Write the function

```
In [8]: df_subset['diff'] = df_subset.apply(diff_money,  
... :  
... :  
... : pattern=pattern)
```

```
In [9]: print(df_subset.head())
```

	Job #	Doc #	Borough	Initial Cost	Total Est.	Fee	diff
0	121577873	2	MANHATTAN	\$75000.00		\$986.00	74014.0
1	520129502	1	STATEN ISLAND	\$0.00		\$1144.00	-1144.0
2	121601560	1	MANHATTAN	\$30000.00		\$522.50	29477.5
3	121601203	1	MANHATTAN	\$1500.00		\$225.00	1275.0
4	121601338	1	MANHATTAN	\$19500.00		\$389.50	19110.5



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Duplicate and missing data



Duplicate data

- Can skew results
- ‘.drop_duplicates()’ method

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27
3	Daniel	male	-	42



Drop duplicates

```
In [1]: df = df.drop_duplicates()
```

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

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27



Missing data

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2.0
1	NaN	1.66	Male	No	Sun	Dinner	3.0
2	21.01	3.50	Male	No	Sun	Dinner	3.0
3	23.68	NaN	Male	No	Sun	Dinner	2.0
4	24.59	3.61	NaN	NaN	Sun	NaN	4.0

- Leave as-is
- Drop them
- Fill missing value



Count missing values

```
In [3]: tips_nan.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill      202 non-null float64
tip            220 non-null float64
sex             234 non-null object
smoker          229 non-null object
day             243 non-null object
time            227 non-null object
size            231 non-null float64
dtypes: float64(3), object(4)
memory usage: 13.4+ KB
None
```



Drop missing values

```
In [4]: tips_dropped = tips_nan.dropna()
```

```
In [5]: tips_dropped.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 147 entries, 0 to 243
Data columns (total 7 columns):
total_bill      147 non-null float64
tip            147 non-null float64
sex             147 non-null object
smoker          147 non-null object
day             147 non-null object
time            147 non-null object
size            147 non-null float64
dtypes: float64(3), object(4)
memory usage: 9.2+ KB
```



Fill missing values with .fillna()

- Fill with provided value
- Use a summary statistic



Fill missing values

```
In [6]: tips_nan['sex'] = tips_nan['sex'].fillna('missing')
```

```
In [7]: tips_nan[['total_bill', 'size']] = tips_nan[['total_bill',
...: 'size']].fillna(0)
```

```
In [8]: tips_nan.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip          220 non-null float64
sex           244 non-null object
smoker        229 non-null object
day            243 non-null object
time           227 non-null object
size           244 non-null float64
dtypes: float64(3), object(4)
memory usage: 13.4+ KB
```



Fill missing values with a test statistic

- Careful when using test statistics to fill
- Have to make sure the value you are filling in makes sense
- Median is a better statistic in the presence of outliers



Fill missing values with a test statistic

```
In [9]: mean_value = tips_nan['tip'].mean()
```

```
In [10]: print(mean_value)  
2.964681818181819
```

```
In [11]: tips_nan['tip'] = tips_nan['tip'].fillna(mean_value)
```

```
In [12]: tips_nan.info()  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 244 entries, 0 to 243  
Data columns (total 7 columns):  
total_bill    244 non-null float64  
tip          244 non-null float64  
sex           244 non-null object  
smoker        229 non-null object  
day           243 non-null object  
time          227 non-null object  
size          244 non-null float64  
dtypes: float64(3), object(4)  
memory usage: 13.4+ KB
```



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Testing your data



Assert statements

- Programmatically vs visually checking
- If we drop or fill NaNs, we expect 0 missing values
- We can write an assert statement to verify this
- We can detect early warnings and errors
- This gives us confidence that our code is running correctly



Asserts

```
In [1]: assert 1 == 1
```

```
In [2]: assert 1 == 2
```

```
AssertionError                                     Traceback (most recent call last)
<ipython-input-65-a810b3a4aded> in <module>()
----> 1 assert 1 == 2
```

```
AssertionError:
```



Google stock data

	Date	Open	High	Low	Close	Volume	Adj Close
0	2017-02-09	831.729980	NaN	826.500000	830.059998	1192000.0	NaN
1	2017-02-08	830.530029	834.250000	825.109985	829.880005	1300600.0	829.880005
2	2017-02-07	NaN	NaN	823.289978	NaN	1664800.0	NaN
3	2017-02-06	820.919983	822.390015	NaN	821.619995	NaN	821.619995
4	2017-02-03	NaN	826.130005	819.349976	820.130005	1524400.0	820.130005



Test column

```
In [1]: assert google.Close.notnull().all()
```

```
-----  
AssertionError                                 Traceback (most recent call last)  
<ipython-input-49-eec77130a77f> in <module>()  
----> 1 assert google.Close.notnull().all()
```

```
AssertionError:
```



Test column

```
In [1]: google_0 = google.fillna(value=0)
```

```
In [2]: assert google.Close.notnull().all()
```



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

**Putting it all
together**



Putting it all together

- Use the techniques you've learned on Gapminder data
- Clean and tidy data saved to a file
 - Ready to be loaded for analysis!
- Dataset consists of life expectancy by country and year
- Data will come in multiple parts
 - Load
 - Preliminary quality diagnosis
 - Combine into single dataset



Useful methods

```
In [1]: import pandas as pd
```

```
In [2]: df = pd.read_csv('my_data.csv')
```

```
In [3]: df.head()
```

```
In [4]: df.info()
```

```
In [5]: df.columns
```

```
In [6]: df.describe()
```

```
In [7]: df.column.value_counts()
```

```
In [8]: df.column.plot('hist')
```



Data quality

```
In [9]: def cleaning_function(row_data):  
    ...:     # data cleaning steps  
    ...:     return ...
```

```
In [10]: df.apply(cleaning_function, axis=1)
```

```
In [11]: assert (df.column_data > 0).all()
```



Combining data

- `pd.merge(df1, df2, ...)`
- `pd.concat([df1, df2, df3, ...])`



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Initial impressions of the data



Principles of tidy data

- Rows form observations
- Columns form variables
- Tidying data will make data cleaning easier
- Melting turns columns into rows
- Pivot will take unique values from a column and create new columns



Checking data types

```
In [1]: df.dtypes
```

```
In [2]: df['column'] = df['column'].to_numeric()
```

```
In [3]: df['column'] = df['column'].astype(str)
```



Additional calculations and saving your data

```
In [4]: df['new_column'] = df['column_1'] + df['column_2']
```

```
In [5]: df['new_column'] = df.apply(my_function, axis=1)
```

```
In [6]: df.to_csv('my_data.csv')
```



CLEANING DATA IN PYTHON

Let's practice!



CLEANING DATA IN PYTHON

Final thoughts



You've learned how to...

- Load and view data in pandas
- Visually inspect data for errors and potential problems
- Tidy data for analysis and reshape it
- Combine datasets
- Clean data by using regular expressions and functions
- Test your data and be proactive in finding potential errors



CLEANING DATA IN PYTHON

Congratulations!