# Lab 5: Data cleaning and analysis with Pandas

**Data Science for Biologists** • University of Washington • BIOL 419/519 • Winter 2019

Course design and lecture material by [Bingni Brunton (https://github.com/bwbrunton)](https://github.com/bwbrunton) and [Kameron Harris (https://github.com/kharris/)](https://github.com/kharris/). Lab design and materials by [Eleanor Lutz (https://github.com/eleanorlutz/)](https://github.com/eleanorlutz/), with helpful comments and suggestions from Bing and Kam.

## Table of Contents

1. Inspecting and cleaning data in Pandas
2. Working with figure objects in Matplotlib
3. Joining two Pandas dataframes
4. Bonus exercises

## Helpful resources

- [Python Data Science Handbook (http://shop.oreilly.com/product/0636920034919.do)](http://shop.oreilly.com/product/0636920034919.do) by Jake VanderPlas
- [10 minute Pandas tutorial (http://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html)](http://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html)
- [Pandas Cheat Sheet (https://datacamp-community-prod.s3.amazonaws.com/9f0f2ae1-8bd8-4302-a67b-e17f3059d9e8)](https://datacamp-community-prod.s3.amazonaws.com/9f0f2ae1-8bd8-4302-a67b-e17f3059d9e8) by Python for Data Science
- [Importing Data Cheat Sheet (https://datacamp-community-prod.s3.amazonaws.com/50d31142-3de0-4159-89b9-18b718a728ef)](https://datacamp-community-prod.s3.amazonaws.com/50d31142-3de0-4159-89b9-18b718a728ef) by Python for Data Science
- [Matplotlib introduction to Pyplot (https://matplotlib.org/tutorials/introductory/pyplot.html)](https://matplotlib.org/tutorials/introductory/pyplot.html)
- [Matplotlib usage guide: includes helpful anatomy of a figure (https://matplotlib.org/tutorials/introductory/usage.html)](https://matplotlib.org/tutorials/introductory/usage.html)
- [Detailed explanation of joining dataframes in Pandas (http://chris.friedline.net/2015-12-15-rutgers/lessons/python2/04-merging-data.html)](http://chris.friedline.net/2015-12-15-rutgers/lessons/python2/04-merging-data.html)

## Data

- The data in this lab is originally from [Gapminder (https://www.gapminder.org/data/)](https://www.gapminder.org/data/) and was edited for teaching purposes.

```
In [ ]:  import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
```

# Lab 5 Part 1: Inspecting and cleaning data in Pandas

In Lab 4 we imported and worked with a very clean dataset. However, real data often has missing values or is saved in several different files. In this lab we'll review common issues when working with real data, such as removing missing values, reshaping and merging data matrices, and creating more complex figures.

Today we will work with two spreadsheets in the `Lab_05` folder: `fertility_data.csv` and `child_mortality_data.csv` . These spreadsheets contain data from the years 1800 to 2000 on a small subset of countries. One dataset contains fertility over time (children per woman), and the other contains child mortality over time (deaths in children aged 0-5 years per 1000 births).

**Exercise 1:** Import the data using the code below. Edit the code to look at the first five rows of each dataset.

```
In [ ]:  fertility = pd.read_csv("./data/Lab_05/fertility_data.csv")
```

```
In [ ]:  mortality = pd.read_csv("./data/Lab_05/child_mortality_data.csv")
```

## Removing discrepancies between datasets

In this lab we plan to directly compare global fertility trends to child mortality trends. Therefore, we are not interested in countries that only have data for one of the two variables. To remove these countries we first need to find out what they are. The `numpy` function `setdiff1d` returns all values in the first array that are not in the second array:

```
In [ ]:  missing_in_mortality = np.setdiff1d(fertility['country'], mortality['cou
         print(missing_in_mortality)
```

**Exercise 2:** Similarly, use `setdiff1d` to find the coutries in the mortality dataset that are missing from the fertility dataset. Place these missing countries in a variable called `missing_in_fertility` .

In [ ]: 

Next, remove the problem countries from each dataset. In English, the code below means "create a new dataframe called `clean_F` that does not ( `~` ) contain any rows in which the `country` column value is in the `missing_in_mortality` array.

In [ ]:
```python
clean_F = fertility[~fertility['country'].isin(missing_in_mortality)]
```

**Exercise 3:** Create a new dataframe called `clean_M` that similarly removes problematic countries from the original mortality dataset.

In [ ]: 

At this point both cleaned datasets should have the same shape:

In [ ]:
```python
print("shape of fertility data:", clean_F.shape)
print("shape of mortality data:", clean_M.shape)
```

# Lab 5 Part 2: Working with figure objects in Matplotlib

Now that we are working with data that has many different variables, it's useful to know some more complex plotting commands to look at this data.

In particular, we can declare `Matplotlib` figures as variables. Giving these objects variable names allows us to edit the objects after they are made - for example, by adding multiple subplots or graphs to the same figure, or adding a colorbar key for all colors used to create a heatmap.

## Making a heatmap in Matplotlib

Since both of our datasets are three dimensional (years, countries, and fertility / mortality data), a heatmap might help visualize this data in a way that's easier to understand.

To create a heatmap in `Matplotlib` , we need three variables:

* The values for the x variables (in our case, years)
* The values for the y variables (in our case, country)
* A matrix or Pandas dataframe containing only the numbers to visualize (in our case, fertility or mortality)

```
In [ ]:  # Get the values for all of the y variables by
         # creating a variable "countries" holding all values from
         # the "country" column in the clean_F dataframe

         countries = clean_F['country'].values
         print(countries)
```

To get all of the year values in a list, we can use the `Pandas` `df.columns.values` function to get a list of column headers. However, since the `clean_F` dataframe also contains "continent" and "country" columns, we need to remove these using the `drop()` function.

```
In [ ]:  # axis=1 specifies that we are dropping columns, not rows
         temp = clean_F.drop(['continent', 'country'], axis=1)
         years = temp.columns.values

         print(years)
         temp.head()
```

Using these three variables we can now plot the heatmap in `Matplotlib`. Notice that the missing values in the dataset show up as white blocks in the heatmap - `Matplotlib` does this automatically to missing data in heatmaps.

```python
In [ ]:  # Create a figure called fig
         # with the sizes 10 inches wide by 4 inches high.
         fig = plt.figure(figsize=(10, 4))

         # Add a subplot within the figure
         ax1 = fig.add_subplot(1,1,1)

         # Create the heatmap as the variable img
         img = ax1.imshow(temp.values)

         # Plot the labels for the y axis (countries)
         ax1.set_yticks(np.arange(len(countries)))
         ax1.set_yticklabels(countries)

         # Plot the labels for the x axis (years)
         ax1.set_xticks(np.arange(len(years)))
         ax1.set_xticklabels(years, rotation=90)

         # Add a colorbar for the variable img
         cbar = fig.colorbar(img, ax=ax1)

         cbar.set_label('Mean children per woman')
         ax1.set_title("Fertility over time")
         plt.show()
```

**Exercise 4:** Similarly, make a heatmap showing the mortality data over time.

```python
In [ ]:
```

# Lab 5 Part 3: Joining two Pandas dataframes

After visualizing both the fertility and mortality dataset, it looks like both datasets seem to be following similar trends. We would like to see if this relationship can be described using a linear regression model.

Right now the data is in two different datasets, so before we can place these data points in a matrix we need a way to pair each mortality value to the corresponding fertility value (by year and country).

## Melting Pandas dataframes

The first step is to reduce the dataframe to a format that facilitates merging. Right now the fertility values are arranged throughout the fertility dataframe under many different columns. It would be more helpful if the year values were contained in their own column, so that all of the fertility data could be contained a single column instead of many:

```
In [ ]:  # Using a temporary dataframe like before,
         # create a list that contains all of the year values
         # that should be stored in the new "year" column
         temp = clean_F.drop(['continent', 'country'], axis=1)
         years = temp.columns.values

         # The pandas melt function pivots dataframes into different arrangements
         # Here is what each argument means:
         # id_vars: columns that should remain the same after melting
         # value_vars: the column heades that should be unwrapped to the row axis
         # var_name: the column header for the new column created by unwrapping v
         # value_name: the column header for the new column created by unwrapping
         melt_F = clean_F.melt(id_vars=['continent','country'], value_name='ferti
                         value_vars=years, var_name="year")
         melt_F.head()
```

**Exercise 5:** Similarly, melt the `clean_M` cleaned mortality dataframe. *Hint:* Remember to change the `value_name` argument to something that best describes the data in this particular dataframe!

```
In [ ]:
```

## Merging Pandas dataframes

Now that both dataframes are in a simplified, identical format, they can be merged together. The `Pandas merge` function can join two or more dataframes together in many different ways. We will use only the default way in this lab (an inner join) but you can read about other options [here (http://chris.friedline.net/2015-12-15-rutgers/lessons/python2/04-merging-data.html)](http://chris.friedline.net/2015-12-15-rutgers/lessons/python2/04-merging-data.html) if you're interested. The `on` argument in `merge()` specifies that rows from the two dataframes will only be included if they share the same values for `country` *and* `continent` *and* `year` in both columns. Rows with these "join keys" present in only one dataframe will not be included.

```
In [ ]:  merge = pd.merge(melt_F, melt_M, on=['country', 'continent', 'year'])
         merge.head()
```

Now that we've combined the two dataframes, we can manipulate pairs of data points individually. For example, now we can remove all data pairs where one (or more) of the data points are missing. First we'll print out the total number of missing (or NaN) values in this merged dataframe.

```
In [ ]:  merge.isnull().sum().sum()
```

The Pandas `dropna()` function can remove missing data from our dataframe. The [userguide (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html) for `dropna()` informs us that the parameter `axis=0` will "Drop rows which contain any missing values" (as opposed to columns):

```
In [ ]:  merge = merge.dropna(axis=0)
```

Now when we count the number of missing values in this dataset again, we can see that there are (hopefully) no NaN values left. This is important because many statistical tests, such as the matrix linear regression we learned ( `np.linalg.lstsq()` ), will not work if there are missing data points.

```
In [ ]:  merge.isnull().sum().sum()
```

**Exercise 6:** Using the dataframe with no missing values, compute $p1$ and $p2$ for the regression equation $mortality = p1 * fertility + p2$.

```
In [ ]:
```

**Exercise 7:** Make a scatterplot of data from all countries and all years. Add the linear regression line you just computed to this plot.

In [ ]:

# Lab 5 Bonus exercises

**Bonus Exercise 1:** On the code for the previous graph, color the scatterpoints from each continent a different color. Add a legend to the plot.

In [ ]:

So far this lab has only included `Matplotlib` figures with one subplot. However, the same syntax used today can be used to create many subplots and plot several different datasets on the same figure. The block below provides the bare bones code to plot the fertility vs mortality data for each individual country.

**Bonus Exercise 2:** Add the country name as the title of each subplot.

**Bonus Exercise 3:** Set the axis limits of all subplots to be the same, for easier comparison between countries. Make sure that the limits are appropriate and do not omit any data points.

**Bonus Exercise 4:** Add a y axis label for just the leftmost five subplots. Add a x axis label for just the bottom five subplots.

In [ ]:
```python
fig = plt.figure(figsize=(10,10))
countries = merge["country"].unique()

for n in range(len(countries)):
    country = countries[n]
    ax = fig.add_subplot(5, 5, n+1)
    temp = merge[merge["country"] == country]
    ax.scatter(temp['fertility'], temp['mortality'])

plt.tight_layout()
plt.show()
```

**Bonus exercise 4:** By default, `Matplotlib` plots figures in its "default" style. However, there are many others that use different different color schemes, label locations, etc. The code below sets the `Matplotlib` style to the popular "ggplot" style. Like `%matplotlib inline`, this command will apply to any code cell run subsequently.

After running the cell, try re-running the figure code from Bonus Exercises 2~4. Look through the [stylesheet gallery (https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html)](https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html) and try a couple other styles for the figure you created in Bonus Exercises 2~4.

A particularly helpful plot style is `seaborn-colorblind`, which changes the default colors to be distinguishable to colorblind people.

In [ ]:
```
plt.style.use("ggplot")
```

In [ ]: