(../00-short-introduction-to-Python/)

Python for ecologists (../)

Starting With Data

(../02-index-slice-subset/)

Overview

Teaching: 30 min
Exercises: 30 min
Questions

- How can I import data in Python?
- What is Pandas?
- Why should I use Pandas to work with data?

Objectives

- Navigate the workshop directory and download a dataset.
- Explain what a library is and what libraries are used for.
- Describe what the Python Data Analysis Library (Pandas) is.
- Load the Python Data Analysis Library (Pandas).
- Use `read_csv` to read tabular data into Python.
- Describe what a DataFrame is in Python.
- Access and summarize data stored in a DataFrame.
- Define indexing as it relates to data structures.
- Perform basic mathematical operations and summary statistics on data in a Pandas DataFrame.
- Create simple plots.

Working With Pandas DataFrames in Python

We can automate the process above using Python. It's efficient to spend time building the code to perform these tasks because once it's built, we can use it over and over on different datasets that use a similar format. This makes our methods easily reproducible. We can also easily share our code with colleagues and they can replicate the same analysis.

Starting in the same spot

To help the lesson run smoothly, let's ensure everyone is in the same directory. This should help us avoid path and file name issues. At this time please navigate to the workshop directory. If you working in IPython Notebook be sure that you start your notebook in the workshop directory.

A quick aside that there are Python libraries like OS Library (https://docs.python.org/3/library/os.html) that can work with our directory structure, however, that is not our focus today.

Our Data

For this lesson, we will be using the Portal Teaching data, a subset of the data from Ernst et al Long-term monitoring and experimental manipulation of a Chihuahuan Desert ecosystem near Portal, Arizona, USA (http://www.esapubs.org/archive/ecol/E090/118/default.htm)

We will be using files from the Portal Project Teaching Database (https://figshare.com/articles/Portal_Project_Teaching_Database/1314459). This section will use the `surveys.csv` file that can be downloaded here: https://ndownloader.figshare.com/files/2292172 (https://ndownloader.figshare.com/files/2292172)

We are studying the species and weight of animals caught in plots in our study area. The dataset is stored as a `.csv` file: each row holds information for a single animal, and the columns represent:

Column	Description
record_id	Unique id for the observation
month	month of observation
day	day of observation
year	year of observation
plot_id	ID of a particular plot
species_id	2-letter code
sex	sex of animal ("M", "F")
hindfoot_length	length of the hindfoot in mm
weight	weight of the animal in grams

The first few rows of our first file look like this:

```
record_id,month,day,year,plot_id,species_id,sex,hindfoot_length,weight
1,7,16,1977,2,NL,M,32,
2,7,16,1977,3,NL,M,33,
3,7,16,1977,2,DM,F,37,
4,7,16,1977,7,DM,M,36,
5,7,16,1977,3,DM,M,35,
6,7,16,1977,1,PF,M,14,
7,7,16,1977,2,PE,F,,
8,7,16,1977,1,DM,M,37,
9,7,16,1977,1,DM,F,34,
```

About Libraries

A library in Python contains a set of tools (called functions) that perform tasks on our data. Importing a library is like getting a piece of lab equipment out of a storage locker and setting it up on the bench for use in a project. Once a library is set up, it can be used or called to perform many tasks.

Pandas in Python

One of the best options for working with tabular data in Python is to use the Python Data Analysis Library (http://pandas.pydata.org/) (a.k.a. Pandas). The Pandas library provides data structures, produces high quality plots with matplotlib (http://matplotlib.org/) and integrates nicely with other libraries that use NumPy (http://www.numpy.org/) (which is another Python library) arrays.

Python doesn't load all of the libraries available to it by default. We have to add an `import` statement to our code in order to use library functions. To import a library, we use the syntax `import libraryName`. If we want to give the library a nickname to shorten the command, we can add `as nickNameHere`. An example of importing the pandas library using the common nickname `pd` is below.

```
import pandas as pd
```

Each time we call a function that's in a library, we use the syntax `LibraryName.FunctionName`. Adding the library name with a `.` before the function name tells Python where to find the function. In the example above, we have imported Pandas as `pd`. This means we don't have to type out `pandas` each time we call a Pandas function.

Reading CSV Data Using Pandas

We will begin by locating and reading our survey data which are in CSV format. We can use Pandas' `read_csv` function to pull the file directly into a DataFrame (http://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe).

So What's a DataFrame?

A DataFrame is a 2-dimensional data structure that can store data of different types (including characters, integers, floating point values, factors and more) in columns. It is similar to a spreadsheet or an SQL table or the `data.frame` in R. A DataFrame always has an index (0-based). An index refers to the position of an element in the data structure.

```
# note that pd.read_csv is used because we imported pandas as pd
pd.read_csv("surveys.csv")
```

The above command yields the **output** below:

```
record_id  month  day  year  plot_id species_id sex  hindfoot_length  weight
0          1      7   16  1977        2         NL   M                32   NaN
1          2      7   16  1977        3         NL   M                33   NaN
2          3      7   16  1977        2         DM   F                37   NaN
3          4      7   16  1977        7         DM   M                36   NaN
4          5      7   16  1977        3         DM   M                35   NaN
...
35544      35545     12   31  2002       15     AH  NaN              NaN  NaN
35545      35546     12   31  2002       15     AH  NaN              NaN  NaN
35546      35547     12   31  2002       10     RM    F               15   14
35547      35548     12   31  2002        7     DO    M               36   51
35548      35549     12   31  2002        5     NaN  NaN             NaN  NaN

[35549 rows x 9 columns]
```

We can see that there were 33,549 rows parsed. Each row has 9 columns. The first column is the index of the DataFrame. The index is used to identify the position of the data, but it is not an actual column of the DataFrame. It looks like the `read_csv` function in Pandas read our file properly. However, we haven't saved any data to memory so we can work with it.We need to assign the DataFrame to a variable. Remember that a variable is a name for a value, such as `x`, or `data`. We can create a new object with a variable name by assigning a value to it using `=`.

Let's call the imported survey data `surveys_df`:

```
surveys_df = pd.read_csv("surveys.csv")
```

Notice when you assign the imported DataFrame to a variable, Python does not produce any output on the screen. We can print the value of the `surveys_df` object by typing its name into the Python command prompt.

```
surveys_df
```

which prints contents like above

Manipulating Our Species Survey Data

Now we can start manipulating our data. First, let's check the data type of the data stored in `surveys_df` using the `type` method. **The `type` method and `__class__` attribute** tell us that `surveys_df` is <class 'pandas.core.frame.DataFrame'>.

```
type(surveys_df)
# this does the same thing as the above!
surveys_df.__class__
```

We can also enter `surveys_df.dtypes` at our prompt to view the data type for each column in our DataFrame. `int64` represents numeric integer values - `int64` cells can not store decimals. `object` represents strings (letters and numbers). `float64` represents numbers with decimals.

```
surveys_df.dtypes
```

which returns:

```
record_id            int64
month                int64
day                  int64
year                 int64
plot_id              int64
species_id          object
sex                 object
hindfoot_length    float64
weight             float64
dtype: object
```

We'll talk a bit more about what the different formats mean in a different lesson.

Useful Ways to View DataFrame objects in Python

There are many ways to summarize and access the data stored in DataFrames, using attributes and methods provided by the DataFrame object.

To access an attribute, use the DataFrame object name followed by the attribute name `df_object.attribute`. Using the DataFrame `surveys_df` and attribute `columns`, an index of all the column names in the DataFrame can be accessed with `surveys_df.columns`.

Methods are called in a similar fashion using the syntax `df_object.method()`. As an example, `surveys_df.head()` gets the first few rows in the DataFrame `surveys_df` using **the `head()` method**. With a method, we can supply extra information in the parens to control behaviour.

Let's look at the data using these.

Challenge - DataFrames

Using our DataFrame `surveys_df`, try out the attributes & methods below to see what they return.

1. `surveys_df.columns`
2. `surveys_df.shape` Take note of the output of `shape` - what format does it return the shape of the DataFrame in?

   HINT: More on tuples, here (https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences).
3. `surveys_df.head()` Also, what does `surveys_df.head(15)` do?
4. `surveys_df.tail()`

Calculating Statistics From Data In A Pandas DataFrame

We've read our data into Python. Next, let's perform some quick summary statistics to learn more about the data that we're working with. We might want to know how many animals were collected in each plot, or how many of each species were caught. We can perform summary stats quickly using groups. But first we need to figure out what we want to group by.

Let's begin by exploring our data:

```
# Look at the column names
surveys_df.columns.values
```

which **returns**:

```
array(['record_id', 'month', 'day', 'year', 'plot_id', 'species_id', 'sex',
       'hindfoot_length', 'weight'], dtype=object)
```

Let's get a list of all the species. The `pd.unique` function tells us all of the unique values in the `species_id` column.

```
pd.unique(surveys_df['species_id'])
```

which **returns**:

```
array(['NL', 'DM', 'PF', 'PE', 'DS', 'PP', 'SH', 'OT', 'DO', 'OX', 'SS',
       'OL', 'RM', nan, 'SA', 'PM', 'AH', 'DX', 'AB', 'CB', 'CM', 'CQ',
       'RF', 'PC', 'PG', 'PH', 'PU', 'CV', 'UR', 'UP', 'ZL', 'UL', 'CS',
       'SC', 'BA', 'SF', 'RO', 'AS', 'SO', 'PI', 'ST', 'CU', 'SU', 'RX',
       'PB', 'PL', 'PX', 'CT', 'US'], dtype=object)
```

Challenge - Statistics

1. Create a list of unique plot ID's found in the surveys data. Call it `plot_names`. How many unique plots are there in the data? How many unique species are in the data?
2. What is the difference between `len(plot_names)` and `surveys_df['plot_id'].nunique()`?

Groups in Pandas

We often want to calculate summary statistics grouped by subsets or attributes within fields of our data. For example, we might want to calculate the average weight of all individuals per plot.

We can calculate basic statistics for all records in a single column using the syntax below:

```
surveys_df['weight'].describe()
```

gives **output**

```
count    32283.000000
mean        42.672428
std         36.631259
min          4.000000
25%         20.000000
50%         37.000000
75%         48.000000
max        280.000000
Name: weight, dtype: float64
```

We can also extract one specific metric if we wish:

```
surveys_df['weight'].min()
surveys_df['weight'].max()
surveys_df['weight'].mean()
surveys_df['weight'].std()
surveys_df['weight'].count()
```

But if we want to summarize by one or more variables, for example sex, we can use **Pandas' `.groupby` method**. Once we've created a groupby DataFrame, we can quickly calculate summary statistics by a group of our choice.

```
# Group data by sex
grouped_data = surveys_df.groupby('sex')
```

The **pandas function `describe`** will return descriptive stats including: mean, median, max, min, std and count for a particular column in the data. Pandas' `describe` function will only return summary values for columns containing numeric data.

```
# summary statistics for all numeric columns by sex
grouped_data.describe()
# provide the mean for each numeric column by sex
grouped_data.mean()
```

`grouped_data.mean()` **OUTPUT:**

```
        record_id     month        day         year    plot_id  \
sex
F    18036.412046  6.583047  16.007138  1990.644997  11.440854
M    17754.835601  6.392668  16.184286  1990.480401  11.098282

     hindfoot_length     weight
sex
F          28.836780  42.170555
M          29.709578  42.995379
```

The `groupby` command is powerful in that it allows us to quickly generate summary stats.

Challenge - Summary Data

1. How many recorded individuals are female `F` and how many male `M`
2. What happens when you group by two columns using the following syntax and then grab mean values:
   - `grouped_data2 = surveys_df.groupby(['plot_id','sex'])`
   - `grouped_data2.mean()`
3. Summarize weight values for each plot in your data. HINT: you can use the following syntax to only create summary statistics for one column in your data `by_plot['weight'].describe()`

Did you get #3 right?

Quickly Creating Summary Counts in Pandas

Let's next count the number of samples for each species. We can do this in a few ways, but we'll use `groupby` combined with **a `count()` method**.

```
# count the number of samples by species
species_counts = surveys_df.groupby('species_id')['record_id'].count()
print(species_counts)
```

Or, we can also count just the rows that have the species "DO":

```
surveys_df.groupby('species_id')['record_id'].count()['DO']
```

Challenge - Make a list

What's another way to create a list of species and associated `count` of the records in the data? Hint: you can perform `count`, `min`, etc functions on groupby DataFrames in the same way you can perform them on regular DataFrames.

Basic Math Functions

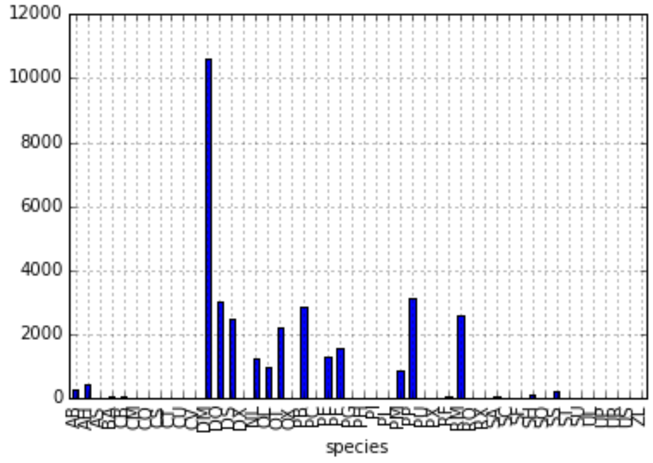
If we wanted to, we could perform math on an entire column of our data. For example let's multiply all weight values by 2. A more practical use of this might be to normalize the data according to a mean, area, or some other value calculated from our data.

```
# multiply all weight values by 2
surveys_df['weight']*2
```

Quick & Easy Plotting Data Using Pandas

We can plot our summary stats using Pandas, too.

```
# make sure figures appear inline in Ipython Notebook
%matplotlib inline
# create a quick bar chart
species_counts.plot(kind='bar');
```



Weight by species plot

We can also look at how many animals were captured in each plot:

```
total_count = surveys_df.groupby('plot_id')['record_id'].nunique()
# let's plot that too
total_count.plot(kind='bar');
```

Challenge - Plots

1. Create a plot of average weight across all species per plot.
2. Create a plot of total males versus total females for the entire dataset.

Summary Plotting Challenge

Create a stacked bar plot, with weight on the Y axis, and the stacked variable being sex. The plot should show total weight by sex for each plot. Some tips are below to help you solve this challenge:

- For more on Pandas plots, visit this link. (http://pandas.pydata.org/pandas-docs/stable/visualization.html#basic-plotting-plot)
- You can use the code that follows to create a stacked bar plot but the data to stack need to be in individual columns. Here's a simple example with some data where 'a', 'b', and 'c' are the groups, and 'one' and 'two' are the subgroups.

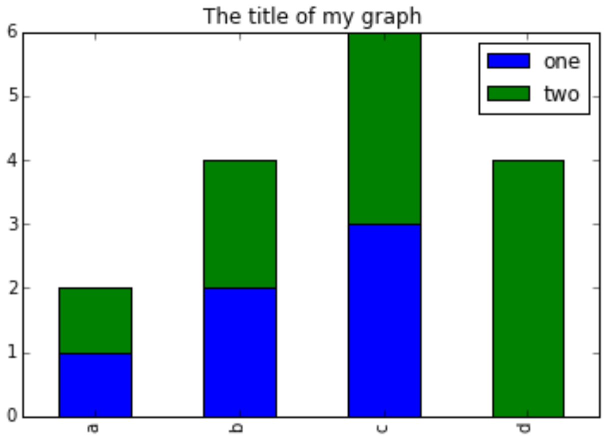
```
d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),'two' : pd.Series([1., 2.,
3., 4.], index=['a', 'b', 'c', 'd'])}
pd.DataFrame(d)
```

shows the following data

```
      one  two
  a     1    1
  b     2    2
  c     3    3
  d   NaN    4
```

We can plot the above with

```
# plot stacked data so columns 'one' and 'two' are stacked
my_df = pd.DataFrame(d)
my_df.plot(kind='bar',stacked=True,title="The title of my graph")
```



- You can use the `.unstack()` method to transform grouped data into columns for each plotting. Try running `.unstack()` on some DataFrames above and see what it yields.

Start by transforming the grouped data (by plot and sex) into an unstacked layout, then create a stacked plot.

Solution to Summary Challenge

Key Points

(../00-short-introduction-to-Python/)

› (../02-index-slice-subset/)

Source (https://github.com/datacarpentry/python-ecology-lesson/) / Contributing (https://github.com/datacarpentry/python-ecology-lesson/blob/gh-pages/CONTRIBUTING.md) / Cite (https://github.com/datacarpentry/python-ecology-lesson/blob/gh-pages/CITATION) / Contact ()