# A Quick Introduction to the “Pandas” Python Library

Adi Bronshtein Follow
Apr 17, 2017 · 7 min read ★


Pandas are cute, but it’s a different kind of panda :)

## Some Background

Hello everyone! Today I want to write about the Pandas library (link to the website). Pandas stands for “Python Data Analysis Library”. According to the Wikipedia page on Pandas, “the name is derived from the term “panel data”, an econometrics term for multidimensional structured data sets.” But I think it’s just a cute name to a super-useful Python library!

Pandas is quite a game changer when it comes to analyzing data with Python and it is one of the most preferred and widely used tools in data munging/wrangling if not THE most used one. Pandas is an open source, free to use (under a BSD license) and it was originally written by Wes McKinney (here’s a link to his GitHub page).

What’s cool about Pandas is that it takes data (like a CSV or TSV file, or a SQL database) and creates a Python object with rows and columns called data frame that looks very similar to table in a statistical software (think Excel or SPSS for example. People who are familiar with R would see similarities to R too). This is so much easier to work with in comparison to

working with lists and/or dictionaries through for loops or list comprehension (please feel free to check out one of my previous blog posts about very basic data analysis using Python. It would have been so much easier to do what I did there using Pandas!).

. . .

## Installation and Getting Started

In order to “get” Pandas you would need to install it. You would also need to have Python 3.5.3 and above. as a pre-requirement for installation (will work with Python 3.6, 3.7, or 3.8) It is also dependent on other libraries (like NumPy) and has optional dependancies (like Matplotlib for plotting). Therefore, I think that the easiest way to get Pandas set up is to install it through a package like the Anaconda distribution , “a cross platform distribution for data analysis and scientific computing.” There you can download the Windows, OS X and Linux versions. If you want to install in a different way, these are the full installation instructions.

In order to use Pandas in your Python IDE (Integrated Development Environment) like Jupyter Notebook or Spyder (both of them come with Anaconda by default), you need to *import* the Pandas library first. Importing a library means loading it into the memory and then it’s there for you to work with. In order to import Pandas all you have to do is run the following code:

```
import pandas as pd
import numpy as np
```

Usually you would add the second part (‘as pd’) so you can access Pandas with ‘pd.command’ instead of needing to write ‘pandas.command’ every time you need to use it. Also, you would import numpy as well, because it is very useful library for scientific computing with Python. Now Pandas is ready for use! Remember, you would need to do it every time you start a new Jupyter Notebook, Spyder file etc.


. . .

# Working with Pandas

## Loading and Saving Data with Pandas

When you want to use Pandas for data analysis, you’ll usually use it in one of three different ways:

- Convert a Python’s list, dictionary or Numpy array to a Pandas data frame
- Open a local file using Pandas, usually a CSV file, but could also be a delimited text file (like TSV), Excel, etc
- Open a remote file or database like a CSV or a JSONon a website through a URL or read from a SQL table/database

There are different commands to each of these options, but when you open a file, they would look like this:

```
pd.read_filetype()
```

As I mentioned before, there are different filetypes Pandas can work with, so you would replace “filetype” with the actual, well, filetype (like CSV). You would give the path, filename etc inside the parenthesis. Inside the parenthesis you can also pass different arguments that relate to how to open the file. There are numerous arguments and in order to know all you them, you would have to read the documentation (for example, the documentation for pd.read_csv() would contain all the arguments you can pass in this Pandas command).

In order to convert a certain Python object (dictionary, lists etc) the basic command is:

```
pd.DataFrame()
```

Inside the parenthesis you would specify the object(s) you’re creating the data frame from. This command also has different arguments (clickable link).

You can also save a data frame you’re working with/on to different kinds of files (like CSV, Excel, JSON and SQL tables). The general code for that is:

```
df.to_filetype(filename)
```

## Viewing and Inspecting Data

Now that you’ve loaded your data, it’s time to take a look. How does the data frame look? Running the name of the data frame would give you the entire table, but you can also get the first n rows with `df.head(n)` or the last n rows with `df.tail(n)`. `df.shape` would give you the number of rows and columns. `df.info()` would give you the index, datatype and memory information. The command `s.value_counts(dropna=False)` would allow you to view unique values and counts for a series (like a column or a few columns). A very useful command is `df.describe()` which inputs summary statistics for numerical columns. It is also possible to get **statistics** on the entire data frame or a series (a column etc):

- `df.mean()` Returns the mean of all columns
- `df.corr()` Returns the correlation between columns in a data frame
- `df.count()` Returns the number of non-null values in each data frame column
- `df.max()` Returns the highest value in each column
- `df.min()` Returns the lowest value in each column
- `df.median()` Returns the median of each column
- `df.std()` Returns the standard deviation of each column

## Selection of Data

One of the things that is so much easier in Pandas is selecting the data you want in comparison to selecting a value from a list or a dictionary. You can select a column (`df[col]`) and return column with label col as Series or a few columns (`df[[col1, col2]]`) and returns columns as a new DataFrame. You can select by position (`s.iloc[0]`), or by index (`s.loc['index_one']`). In order to select the first row you can use `df.iloc[0,:]` and in order to select the first element of the first column you would run `df.iloc[0,0]`. These can also be used in different combinations, so I hope it gives you an idea of the different selection and indexing you can perform in Pandas.

### Filter, Sort and Groupby

You can use different conditions to filter columns. For example, `df[df[year] > 1984]` would give you only the column year is greater than 1984. You can use & (and) or | (or) to add different conditions to your filtering. This is also called *boolean filtering*.

It is possible to sort values in a certain column in an ascending order using `df.sort_values(col1)`; and also in a descending order using `df.sort_values(col2,ascending=False)`. Furthermore, it’s possible to sort values by `col1` in ascending order then `col2` in descending order by using `df.sort_values([col1,col2],ascending=[True,False])`.

The last command in this section is groupby. It involves splitting the data into groups based on some criteria, applying a function to each group independently and combining the results into a data structure. `df.groupby(col)` returns a groupby object for values from one column while `df.groupby([col1,col2])` returns a groupby object for values from multiple columns.

### Data Cleaning

Data cleaning is a very important step in data analysis. For example, we always check for missing values in the data by running `pd.isnull()` which checks for null Values, and returns a boolean array (an array of *true* for missing values and *false* for non-missing values). In order to get a sum of null/missing values, run `pd.isnull().sum()`. `pd.notnull()` is the opposite of `pd.isnull()`. After you get a list of missing values you can get rid of them, or drop them by using `df.dropna()` to drop the rows or `df.dropna(axis=1)` to drop the columns. A different approach would be to

fill the missing values with other values by using `df.fillna(x)` which fills the missing values with x (you can put there whatever you want) or `s.fillna(s.mean())` to replace all null values with the mean (mean can be replaced with almost any function from the statistics section).

It is sometimes necessary to replace values with different values. For example, `s.replace(1,'one')` would replace all values equal to `1` with `'one'`. It’s possible to do it for multiple values: `s.replace([1,3], ['one','three'])` would replace all 1 with `'one'` and `3` with `'three'`. You can also rename specific columns by running: `df.rename(columns= {'old_name': 'new_ name'})` or use `df.set_index('column_one')` to change the index of the data frame.

### Join/Combine

The last set of basic Pandas commands are for joining or combining data frames or rows/columns. The three commands are:

- `df1.append(df2)` — add the rows in `df1` to the end of `df2` (columns should be identical)
- `df.concat([df1, df2],axis=1)` — add the columns in `df1` to the end of `df2` (rows should be identical)
- `df1.join(df2,on=col1,how='inner')` — SQL-style join the columns in `df1` with the columns on `df2` where the rows for `col` have identical values. how can be equal to one of: `'left'`, `'right'`, `'outer'`, `'inner'`

. . .

These are the very basic Pandas commands but I hope you can see how powerful Pandas can be for data analysis. This post is just the tip of the iceberg — after all, entire books can be (and have been) written about data analysis with Pandas. I also hope this post made you feel like taking a dataset and playing around with it using Pandas! :)

As always, if you have any comments, notes, suggestions or questions, please don’t hesitate to write me! Thanks for reading :) I will end, naturally with a picture of cute pandas and with a question — which do you prefer, giant pandas or red pandas???

See you next time!

P.S.

If you liked this tutorial, please check out my quick introduction to NumPy!



Data Science Python Pandas Data Analysis