**Pandas**

[Pandas library](http://pandas.pydata.org/) (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](https://en.wikipedia.org/wiki/Panel_data)”, an [econometrics](https://en.wikipedia.org/wiki/Econometrics) term for multidimensional structured data sets.”

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](https://en.wikipedia.org/wiki/Data_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](https://en.wikipedia.org/wiki/Wes_McKinney).

What’s cool about Pandas is that it takes data (like a CSV or TXT file, or a SQL database) and creates a Python object with rows and columns called DataFrame that looks very similar to table in Excel. The columns are Series. A DataFrame is a collection of multiple Series. A Panel is a 3D container of multiple DataFrames similar to a three axis numpy ndarray but the Panel class has been removed from the latest version of Pandas.

**Installation and Getting Started**

python -m pip install pandas

python -m pip install pandas --user

Once you have installed it, you can import it in a similar manner to importing numpy. Typically, we will give it an alias such as pd.

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. Typically, you will import both pandas and numpy because we will use many numpy commands in conjuction with pandas objects and functions!

**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 Series or DataFrame.
* Open a local (or remote!) file using Pandas, usually a CSV or JSON file, and convert it to a Series or DataFrame.
* Create a Pandas Series or DataFrame from a database such as Access or MySQL.

pd.Series(np.arange(26,99,3))

pd.DataFrame([[1,2],[3,4]], columns = [**'a'**,**'b'**])

pd.read\_json(**'json\_sample\_pandas.json'**)

pd.read\_csv(**'some.csv’, delimiter=’|’)**

All Pandas data structures are value mutable (can be changed) and except Series are all size mutable. Series are size immutable!

**Viewing and Inspecting Data**

Once are data are loaded, here are a few common commands that will be useful when working with rows and columns!

* df.head(n) Returns n number of top rows
* df.tail(n)Returns n number of bottom rows
* df.shape Returns the shape of the DataFrame
* df.info() Returns the index, datatype and memory information
* s.value\_counts(dropna=False) would allow you to view unique values and counts for a series (like a column or a few columns).
* df.describe() Returns summary statistics for numerical columns (Series)
* 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
* df.sample(frac=value)Returns a randomly selected fraction of the rows in the DataFrame
* df.sample(n=NumberofRows)Returns a randomly selected number of rows from a DataFrame.
* df[score>=3]Returns all rows where the score column is greater than or equal to 3. This is referred to as Boolean indexing and can be quite useful for a variety of analyses!

**Selection of Data**

Selecting data in Pandas is similar to selecting data in a list or a dictionary. You can select a single column (df[‘ColName’]) as Series or a few columns (df[[‘ColName1’, ‘ColName2’]]) as a DataFrame.

You can also select items 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] > 2020] would give you only the column year is greater than 2020. 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]).

We can use a groupby command by 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 and can be easily done with Pandas Series and DataFrames.

For example, we can check for missing values by running pd.isnull() which checks for null values (nan), 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, execute 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 something like the following: df.rename(columns={'old\_name': 'new\_ name'}) or use df.set\_index('column\_one') to change the index of the data frame.

**Join/Combine**

We can also join or combine DataFrames or rows/columns. Three popular commands for these operations are the following:

* 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'