

Python for Data Analysis

NMSU Pandas/Python/Jupyter Notebooks Workshop 2018

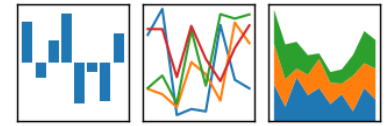
Dr. Bleu Knight (bleu@nmsu.edu)

Python Libraries for Data Science

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
- many other python libraries are built on NumPy (i.e. Pandas)

Link: <http://www.numpy.org/>



Python Libraries for Data Science

Pandas:

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

Link: <http://pandas.pydata.org/>

Reading data using pandas

```
In [ ]: #Read csv file
        pols = pd.read_csv("pollutants.csv")
```

Note: The above command has many optional arguments to fine-tune the data import process.

There are a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx', sheet_name='Sheet1', index_col=None, na_values=['NA'])
```

```
pd.read_stata('myfile.dta')
```

```
pd.read_sas('myfile.sas7bdat')
```

```
pd.read_hdf('myfile.h5', 'df')
```

Data Frame data types

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Data Frame data types

```
In [4]: #Check a particular column type  
pols['units'].dtype
```

```
Out[4]: dtype('object')
```

```
In [5]: #Check types for all the columns  
pols.dtypes
```

Data Frames attributes

Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Data Frames methods

Unlike attributes, python methods have *parenthesis*.

All attributes and methods can be listed with a *dir()* function: `dir(pols)`

pols.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Selecting a column in a Data Frame

Method 1: Subset the data frame using column name:

```
pols['parameter']
```

Method 2: Use the column name as an attribute:

```
df.parameter
```

Data Frames *groupby* method

Using "group by" method we can:

- Split the data into groups based on some criteria
- Calculate statistics (or apply a function) to each group
- Similar to `dplyr()` function in R

```
In [ ]: #Group data using rank  
Par = pols.groupby(['parameter'])
```

```
In [ ]: #Calculate mean value for each numeric column per each group  
Par.mean()
```

Data Frames *groupby* method

Once groupby object is create we can calculate various statistics for each group:

```
In [ ]: #Find the pollutants that have the largest result values:  
pols.groupby( 'parameter' )[[ 'result' ]].mean( )
```

Note: If single brackets are used to specify the column (e.g. result), then the output is Pandas Series object. When double brackets are used the output is a Data Frame

Data Frames *groupby* method

groupby performance notes:

- no grouping/splitting occurs until it's needed. Creating the *groupby* object only verifies that you have passed a valid mapping
- by default the group keys are sorted during the *groupby* operation. You may want to pass `sort=False` for potential speedup:

Data Frame: filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the result value is greater than 0:

```
In [ ]: #Calculate mean salary for each professor rank:  
pols2= pols[ pols['result'] > 0 ]
```

Any Boolean operator can be used to subset the data:

>	greater;	>=	greater or equal;
<	less;	<=	less or equal;
==	equal;	!=	not equal;

Data Frames: Slicing

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

Data Frames: Slicing

When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

```
In [ ]: #Select general location:  
pols['general_location']
```

When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

```
In [ ]: #Select column salary:  
pols[['general_location', 'specific_location']]
```

Data Frames: Selecting rows

If we need to select a range of rows, we can specify the range using ":"

```
In [ ]: #Select rows by their position:  
pols[10:20]
```

Notice that the first row has a position 0, and the last value in the range is omitted:
So for 0:10 range the first 10 rows are returned with the positions starting with 0
and ending with 9

Data Frames: method loc

If we need to select a range of rows, using their labels we can use method loc:

```
In [ ]: #Select rows by their labels:  
df_sub.loc[10:20,['general_location','specific_location']]
```

Data Frames: method iloc

If we need to select a range of rows and/or columns, using their positions we can use method iloc:

```
In [ ]: #Select rows by their labels:  
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

Data Frames: method iloc (summary)

```
df.iloc[0]    # First row of a data frame  
df.iloc[i]     #(i+1)th row  
df.iloc[-1]   # Last row
```

```
df.iloc[:, 0] # First column  
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]      #First 7 rows  
df.iloc[:, 0:2]    #First 2 columns  
df.iloc[1:3, 0:2]  #Second through third rows and first 2 columns  
df.iloc[[0,5], [1,3]] #1st and 6th rows and 2nd and 4th columns
```

Missing Values

There are a number of methods to deal with missing values in the data frame:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Aggregation Functions in Pandas

Aggregation - computing a summary statistic about each group, i.e.

- compute group sums or means
- compute group sizes/counts

Common aggregation functions:

min, max

count, sum, prod

mean, median, mode, mad

std, var

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis