

PWC Python Course: Pandas

Ramses van Zon

SciNet HPC Consortium

December 11, 2014

Handling data

Dealing with data

Before we start coding we need to understand how our data is stored, and how to handle it.

- Big Data data is generally organized more like an Excel file than as a numeric array.
- This means there are usually 'headers' (column titles), and sometimes indexes (row numbers).
- Generally speaking, the data contains a mixture of datatypes (strings, floats, integers, booleans), not just a single type.
- The data is commonly stored several different formats. We'll be focussing on CSV ("Comma Separated Values").

For this we will use the Python 'pandas' package.

Working with pandas

Working with pandas

```
>>> import pandas as pd
>>> name = ['Anna', 'William', 'Emma', 'John',
...         'James', 'Mary']
>>> gender = ['F', 'M', 'F', 'M', 'M', 'F']
>>> number = [2604, 9532, 2003, 9655, 5927, 7065]
>>> data = zip(name, gender, number)
>>> data
[('Anna', 'F', 2604),
 ('William', 'M', 9532),
 ('Emma', 'F', 2003),
 ('John', 'M', 9655),
 ('James', 'M', 5927),
 ('Mary', 'F', 7065)]
```

The 'zip' function returns a list of tuples, constructing the list from the lists in the argument.

These data were the 3 most popular American girl and boy names in 1880, and how often they were given.

Working with pandas, continued

```
>>> import pandas as pd
>>> df = pd.DataFrame(data, columns=['Name', 'Gender', 'Number'])
>>> df
```

	Name	Gender	Number
0	Anna	F	2604
1	William	M	9532
2	Emma	F	2003
3	John	M	9655
4	James	M	5927
5	Mary	F	7065

```
>>> df.to_csv('births1880.csv', index=False, header=True)
>>>
```

The data have been cast into a 'DataFrame' type, which is the type used by pandas. Note how there are now index numbers and column headings.

Pandas and Microsoft Excel Sheets

note: requires additional python modules: xlrd xlwt openpyxl

Writing

```
>>> df
      Name Gender  Number
0    Anna      F    2604
1 William      M    9532
2   Emma      F    2003
3   John      M    9655
4  James      M    5927
5   Mary      F    7065
>>> df.to_excel('births1880.xlsx', index=False, header=True)
```

Reading

```
>>> xl = pd.ExcelFile('births1880.xlsx')
>>> xl.sheet_names
['Sheet1']
>>> newdf = xl.parse('Sheet1')
```

DataFrame functions

Some DataFrame functions

```
>>> df['Name'].describe()
count      6
unique      6
top        Anna
freq        1
Name: Name, dtype: object
>>> df.Number.describe()
count      6.000000
mean      6131.000000
std       3297.861792
min       2003.000000
25%      3434.750000
50%      6496.000000
75%      8915.250000
max       9655.000000
Name: Number, dtype: float64
```

Use “help(df)” to look at all the available functions.

```
>>> sorted = df.sort(['Number'])
>>> type(sorted)
pandas.cored.frame.DataFrame
>>> sorted.head(2)
      Name  Gender  Number
2  Emma      F    2003
0  Anna      F    2604
>>> df['Name'].count()
6
>>> df.shape
(6, 3)
>>> df.Number.sum()
36786
>>> df.dtypes
Name      object
Gender    object
Number    int64
dtype: object
```

Adding columns

Adding columns

```
>>> d = [0, 1, nan, 2]
>>> df = pd.DataFrame(d)
>>> df
   0
0   0
1   1
2  NaN
3   2
>>> df.columns = ['Rev']
>>> df
   Rev
0     0
1     1
2  NaN
3     2
>>> df[1:2]
   Rev
1     1
>>> df['NewCol'] = 5
```

Adding columns

```
>>> d = [0, 1, nan, 2]
>>> df = pd.DataFrame(d)
>>> df
   0
0   0
1   1
2  NaN
3   2
>>> df.columns = ['Rev']
>>> df
   Rev
0    0
1    1
2  NaN
3    2
>>> df[1:2]
   Rev
1    1
>>> df['NewCol'] = 5
```

```
>>> df
   Rev  NewCol
0    0       5
1    1       5
2  NaN       5
3    2       5
>>> df['test'] = df['Rev'] + 1
>>> df
   Rev  NewCol  test
0    0       5     1
1    1       5     2
2  NaN       5  NaN
3    2       5     3
>>> df['test']
0    1
1    2
2  NaN
3    3
Name: test, dtype: float64
```

Adding rows

Adding rows

```
>>> df.xs(3)
Rev      2
NewCol    5
test      3
Name: 3, dtype: float64
>>> df2 = df.append(df.xs(3),
...                  ignore_index=True)
>>> df2
```

	Rev	NewCol	test
0	0	5	1
1	1	5	2
2	NaN	5	NaN
3	2	5	3
4	2	5	3

You may only append structures which are of type DataFrame or Series.

```
>>> df2.append([(-1, -2, -3)])
      0  1  2 NewCol Rev test
0  NaN NaN NaN      5  0  1
1  NaN NaN NaN      5  1  2
2  NaN NaN NaN      5 NaN NaN
3  NaN NaN NaN      5  2  3
4  NaN NaN NaN      5  2  3
0  -1  -2  -3      NaN NaN NaN

>>> df2.append([(-1, -2, -3)],
...columns=['Rev', 'NewCol', 'test'])
      Rev  NewCol  test
0      0      5      1
1      1      5      2
2     NaN      5     NaN
3      2      5      3
4      2      5      3
0     -1     -2     -3
```

Real data

Let's play with real data!

It's more fun to play with real data.

- Log in to Sharepoint.
- Go to the “2014-15 Placement w/ Dr.~Spence” directory.
- Go to the “data” directory.
- Download the file 311-service-requests.csv.zip.
- Double click on the file. Windows will uncompress it for you.
- On the upper left, click on “Extract all files”.
- Put the file somewhere easy to access.

Let's play with real data!

```
>>> filename = "/path/to/my/311-service-requests.csv"
>>> data = pd.read_csv(filename)
>>> data.shape
(111069, 52)
>>> data.columns
Index([u'Unique Key', u'Created Date', u'Closed Date', u'Agency',
      u'Agency Name', u'Complaint Type', u'Descriptor', u'Location
      Type', u'Incident Zip', u'Incident Address', u'Street Name',
      u'Cross Street 1', u'Cross Street 2', u'Intersection Street 1',
      u'Intersection Street 2', u'Address Type', u'City', u'Landmark',
      u'Facility Type', u'Status', u'Due Date', u'Resolution Action
      Updated Date', u'Community Board', u'Borough', u'X Coordinate
      (State Plane)', u'Y Coordinate (State Plane)', u'Park Facility
      Name', u'Park Borough', u'School Name', u'School Number',
      u'School Region', u'School Code', u'School Phone Number',
      u'School Address', u'School City', u'School State', u'School
      Zip', u'School Not Found', u'School or Citywide Complaint',
      u'Vehicle Type', u'Taxi Company Borough', u'Taxi Pick Up
      Location', u'Bridge Highway Name', u'Bridge Highway Direction',
      u'Road Ramp', u'Bridge Highway Segment', u'Garage Lot Name',
```

Real data!, continued

```
>>> data.values[0]
array([26589651, '10/31/2013 02:08:41 AM', nan, 'NYPD', 'New York
City Police Department', 'Noise - Street/Sidewalk', 'Loud
Talking', 'Street/Sidewalk', 11432.0, '90-03 169 STREET', '169
STREET', '90 AVENUE', '91 AVENUE', nan, nan, 'ADDRESS',
'JAMAICA', nan, 'Precinct', 'Assigned', '10/31/2013 10:08:41
AM', '10/31/2013 02:35:17 AM', '12 QUEENS', 'QUEENS', 1042027.0,
197389.0, 'Unspecified', 'QUEENS', 'Unspecified', 'Unspecified',
'Unspecified', 'Unspecified', 'Unspecified', 'Unspecified',
'Unspecified', 'Unspecified', 'Unspecified', 'N', nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, 40.70827532593202,
-73.79160395779721, '(40.70827532593202, -73.79160395779721)'],
dtype=object)
>>>
```

Specifying the index gives us all the values in that row.

Real data!, continued more

```
>>> data[0:3]
```

	Unique Key	Created Date	Closed Date	Agency
0	26589651	10/31/2013 02:08:41 AM	NaN	NYPD
1	26593698	10/31/2013 02:01:04 AM	NaN	NYPD
2	26594139	10/31/2013 02:00:24 AM	10/31/2013 02:40:32 AM	NYPD

	Agency Name	Complaint Type
0	New York City Police Department	Noise - Street/Sidewalk
1	New York City Police Department	Illegal Parking
2	New York City Police Department	Noise - Commercial

```
.  
.
```

```
>>> from numpy import unique
```

```
>>> unique(data["Complaint Type"].values)
```

```
.  
.
```

This is service request data from New York City (NYC Open Data) 

Real data!, continued some more

Suppose we only want some of the columns?

```
>>>
>>> data.loc[[12:15, ["Created Date", "Complaint Type", "Longitude"]]
```

	Created Date	Complaint Type	Longitude
12	10/31/2013 01:20:57 AM	Illegal Parking	-73.952259
13	10/31/2013 01:20:13 AM	Noise - Vehicle	-73.836457
14	10/31/2013 01:19:54 AM	Rodent	-73.999218
15	10/31/2013 01:14:02 AM	Noise - House of Worship	-73.970370
16	10/31/2013 12:54:03 AM	Noise - Street/Sidewalk	-74.116150
17	10/31/2013 12:52:46 AM	Illegal Parking	-73.888173
18	10/31/2013 12:51:00 AM	Street Light Condition	NaN

```
>>>
```

Specifying the index gives us all the values in that row.

Noise-complaint data

Noise-complaint data

Suppose we're only interested in the noise complaint data.

```
>>>
>>> noise = data[data["Complaint Type"]=="Noise - Street/Sidewalk"]
>>> noise[0:3]
```

	Unique Key	Created Date	Closed Date	Agency
0	26589651	10/31/2013 02:08:41 AM	NaN	NYPD
16	26594086	10/31/2013 12:54:03 AM	10/31/2013 02:16:39 AM	NYPD
25	26591573	10/31/2013 12:35:18 AM	10/31/2013 02:41:35 AM	NYPD

	Agency Name	Complaint Type
0	New York City Police Department	Noise - Street/Sidewalk
16	New York City Police Department	Noise - Street/Sidewalk
25	New York City Police Department	Noise - Street/Sidewalk

```
>>> noise.shape
(1928, 52)
```

This picks out all “Noise - Street/Sidewalk” complaints.

How did that work?

The last command picked out the correct entries. How did that work?

```
>>> data["Complaint Type"] == "Noise - Street/Sidewalk"
0      True
1     False
2     False
3     False
4     False
...
111064    False
111065    False
111066     True
111067    False
111068    False
Name: Complaint Type, Length: 111069, dtype: bool
>>>
```

When we index our data with this array of booleans,
we pick out the entries we're interested in.

Further data pruning

More pruning

We can combine more than one condition to restrict our search further.

```
>>> is_noise = data["Complaint Type"] == "Noise - Street/Sidewalk"
>>> in_brooklyn = data["Borough"] == "BROOKLYN"
>>> b_noise = data[is_noise & in_brooklyn]
>>> b_noise[["Complaint Type", "Borough", "Descriptor"]][:3]
```

	Complaint Type	Borough	Descriptor
31	Noise - Street/Sidewalk	BROOKLYN	Loud Music/Party
49	Noise - Street/Sidewalk	BROOKLYN	Loud Talking
109	Noise - Street/Sidewalk	BROOKLYN	Loud Music/Party

```
>>>
>>> True & True
True
>>> True & False
False
>>> False & False
False
```

The & symbol is the “AND” operator.

Plotting the data

Who complains the most?

So which borough is responsible for the most complaints?

```
>>>noise = data[is_noise]
>>>noise["Borough"].value_counts()
MANHATTAN          917
BROOKLYN           456
BRONX              292
QUEENS             226
STATEN ISLAND      36
Unspecified         1
dtype: int64
>>>
```

Manhattan!

Perhaps we should normalize to see what fraction of complaints came from each borough.

```
>>>n_counts =
...noise["Borough"].value_counts()
>>>b_counts =
...data["Borough"].value_counts()
>>>n_counts / b_counts
BRONX              0.014833
BROOKLYN           0.013864
MANHATTAN          0.037755
QUEENS             0.010143
STATEN ISLAND      0.007474
Unspecified         0.000141
dtype: float64
>>>(n_counts / b_counts).plot(
...kind="bar")
```

Who complains the most?

