Intro to Python for Social Scientists

November 20, 2017

```
In [1]: %matplotlib inline
```

1 Intro to Python for Social Scientists

This tutorial provides an introduction to programming in Python, along with a few introductory examples on how Python is generally used in social science research. We will cover:

- Data types: integers, floats, strings, booleans
- Data structures: lists, sets, dictionaries and tuples
- Loops
- Conditional statements
- Writing functions
- Reading and writing data
- Importing third party modules
- Working with data in different formats
- Basic visualization
- Additional resources

1.1 Variables, data types and operators

You create a new variable by simply declaring it.

```
In [347]: a="Hello World!" #a string variable. Strings need to be placed in single or double q
          b=2 #an integer variable
          c=2/3 #a float variable
          d=(b==24) #a boolean variable
```

To print to console:

```
In [326]: print(a)

Hello World!


In [327]: b

Out[327]: 2
```

```
In [328]: print(c+b)

2.6666666666666665


In [329]: d

Out[329]: False
```

You should always know what type your variables are, since some operations can only be done on certain types of variables. To check variable types:

```
In [330]: print("a is", type(a), ",b is", type(b),  ",c is", type(b),  ",d is", type(d) )

a is <class 'str'> ,b is <class 'int'> ,c is <class 'int'> ,d is <class 'bool'>
```

1.1.1 Operators

Mathematical, comparison and boolean operations and their order or evaluation:
1. exponent: ** 2. multiplication, division, modulo *, /, % 3. addition, subtraction +, -
4. comparison operators <, <=, >=, >, ==, != 5. comparison operators: is, is not, in, not in 6. boolean NOT, AND, OR: not, and, or

Use () to change the default order. This is just maths.

```
In [331]: 2**b+20/b<=15

Out[331]: True

In [332]: 2**(b+20)/b<=15

Out[332]: False

In [333]: d==False

Out[333]: True

In [334]: d is not True

Out[334]: True

In [335]: d is not True and b==3

Out[335]: False

In [336]: d is not True or b==3

Out[336]: True

In [337]: result=(b+c)-(d*2)
          result
```

```
Out[337]: 2.6666666666666665
```

You can use some of these operators on strings as well.

```
In [338]: "Hello" in a

Out[338]: True
```

But try:

```
In [339]: x="2"
          y="3"
          result=x+y
          print(result)

23
```

For strings, '+' performs concatenation.

```
In [341]: type(result)

Out[341]: str
```

Now try this:

```
In [342]: x="2"
          y=3
          result=x+y
          print(result)


          ---------------------------------------------------------------------------

          TypeError                                 Traceback (most recent call last)

          <ipython-input-342-30f6eda1ce84> in <module>()
            1 x="2"
            2 y=3
       ----> 3 result=x+y
            4 print(result)


          TypeError: must be str, not int
```

What does the error say?

1.1.2 Converting variables from one type to another

Sometimes, your data has variables that you would like to use as numbers coded as string, just as we have x and y above. Or some of the variables are coded as strings, while others are numbers, although you believe they should all be numbers. If you try to add them however, you get an error saying that you can't add strings and numbers. Assuming all the values of the dataset variables look like numbers, you can convert them into integers or floats. Or the other way around. Now try this:

```
In [343]: x="2"
          y="3"
          result=int(x)+float(y)
          print(result)
```

```
5.0
```

```
In [344]: type(result)
```

```
Out[344]: float
```

And back to string:

```
In [345]: type(str(result))
```

```
Out[345]: str
```

1.1.3 Exercise:

1. Create a new variable called 'birth_year' that contains your year of birth.
2. Using your birth year, calculate your age and assign it to a new variable called 'age'.

3. Print a sentence of the form "I am *age* years old." to the console.
4. Create a new string variable called 'sentence' that contains this statement.

Write your code in the box below. Let's see who finishes first!

1.2 Data Structures

Note that the variable we've been working with so far contain a single value. However, what we normally refer to as "variables" in data analysis are variables from datasets, which contain more than one value. In python, these types of data structures can be lists, sets, dictionaries and tuples.

1.2.1 Lists

Lists are stored between square brackets, and the elements are separated by commas. Here is a list of ages:

```
In [349]: ages=[21, 20, 19, 21, 20, 33, 22, 23, 26, 21, 22, 30, 19, 28]
          ages
```

```
Out[349]: [21, 20, 19, 21, 20, 33, 22, 23, 26, 21, 22, 30, 19, 28]
```

```
In [348]: len(ages) # this is the number of elements in the list
```

```
Out[348]: 9
```

Lists can be indexed and sliced:

```
In [350]: # Indexing - getting an element by position. Note that we start from 0 and we stop a
          first_element=ages[0] # this is the element at index 0
          last_element=ages[13] # this is the element at index 13
          print(first_element, "to", last_element)
```

```
21 to 28
```

```
In [351]: # Slicing - getting a subset of the elements in the list.
          first_3=ages[0:3] # the same thing as ages[:3]
          last_3=ages[-3:] # the same thing as ages[10:14]
          print(first_3, "and", last_3)
```

```
[21, 20, 19] and [30, 19, 28]
```

```
In [352]: ages[10:14]
```

```
Out[352]: [22, 30, 19, 28]
```

1.2.2 Other common list operations

```
In [354]: # Check if values in list:
          40 not in ages # true if value is not in the list
```

```
Out[354]: True
```

```
In [355]: # sorting the list by values:
          ages.sort()
          ages
```

```
Out[355]: [19, 19, 20, 20, 21, 21, 21, 22, 22, 23, 26, 28, 30, 33]
```

```
In [361]: # adding to the list
          ages.append(2)
          ages
```

```
Out[361]: [19, 19, 20, 20, 21, 21, 21, 22, 22, 23, 26, 28, 30, 33, 2]
```

```
In [359]: # concatenating two lists:
          l1=["a", "b", "c"]
          l2=[1, 2, 3]
          l3=l1+l2
          l3
```

```
Out[359]: ['a', 'b', 'c', 1, 2, 3]

In [362]: # removing an element from the list by value
          ages.remove(2)
          ages

Out[362]: [19, 19, 20, 20, 21, 21, 21, 22, 22, 23, 26, 28, 30, 33]

In [363]: # finding the index (position) of the first place where the value occurs in the list
          ages.index(21)

Out[363]: 4

In [364]: # remove an element from the list by index
          del ages[0:5]
          ages

Out[364]: [21, 21, 22, 22, 23, 26, 28, 30, 33]
```

1.2.3 Sets

A set contains an unordered collection of unique and immutable objects. If you want to get all unique values in a list, a quick way it to transform the list into a set:

```
In [365]: set_ages=set(ages)
          set_ages

Out[365]: {21, 22, 23, 26, 28, 30, 33}

In [366]: unique_ages=list(set_ages)
          unique_ages

Out[366]: [33, 21, 22, 23, 26, 28, 30]
```

1.2.4 Dictionaries

In a dictionary, an entry consists of a word and the word's definition. The word is the key to finding out what a word means, and what the word means is considered the value for that key. In Python, dictionaries have keys and values. Keys are used to find values. Here is a dictionary of people and their ages:

```
In [367]: mydict = {"John": 21,
                    "Jake": 20,
                    "Jack": 23,
                   }
          mydict

Out[367]: {'Jack': 23, 'Jake': 20, 'John': 21}

In [368]: mydict.keys()
```

```
Out[368]: dict_keys(['John', 'Jake', 'Jack'])

In [369]: mydict.values()

Out[369]: dict_values([21, 20, 23])

In [370]: mydict["John"]

Out[370]: 21
```

Dictionaries will be very useful when we start working with web data, such as social media data.

1.3 Indentation

Python **requires** blocks to be structured through indentation. Not just as a matter of style, but as a rule. Statements with the same distance to the left belong to the same block of code. To nest blocks, you need to indent them further to the right. The number of white spaces doesn't matter, what matters is that you are consistently using the same number for blocks that are at the same level. Usually, we start at the very left edge, and each level in goes a further 1 tab (or 4 white spaces) to the right. If the code does not follow this rule about the relative indentation of blocks, then you will get an **IndentationError**.

However, the indentation level is ignored when you use explicit (or implicit) continuation lines. You can split a list or dictionary across multiple lines, and the indentation doesn't matter.

You will see a few examples in the sections below.

1.4 Loops

Most of our work involves some type of iteration over observations in a dataset. Iteration is very easy and intuitive in Python, and there are many ways to loop through data in order to access and manipulate it.

```
In [371]: # for loops
          for i in range(5):
           print("I can count to "+str(i))

I can count to 0
I can count to 1
I can count to 2
I can count to 3
I can count to 4


In [373]: # while loops
          counter = 0
          while counter < 5:
              print("I can count to", counter)
              counter += 1
```

```
I can count to 0
I can count to 1
I can count to 2
I can count to 3
I can count to 4
```

```
In [375]: #List comprehension
          k=[key for key in mydict.keys()]
          k

Out[375]: ['John', 'Jake', 'Jack']
```

1.5 Conditional statements

Data management, processing and analysis involve taking a series of decisions. We use conditional statements (most often in the form of if statements) to take these decitions.

```
In [376]: # if statement
          for i in range(5):
              if i  % 2 == 0:
                  print("I can count even numbers to "+str(i))
```

```
I can count even numbers to 0
I can count even numbers to 2
I can count even numbers to 4
```

```
In [377]: # if-else statement
          for i in range(5):
              if i  % 2 == 0:
                  print("I can count even numbers to "+str(i))
              else:
                  print("I can count odd numbers to "+str(i))
```

```
I can count even numbers to 0
I can count odd numbers to 1
I can count even numbers to 2
I can count odd numbers to 3
I can count even numbers to 4
```

```
In [378]: # if-elif-else statements
          for i in range(5):
              if i<=1:
                  print("I can count to "+str(i))
              elif 2<=i<=3:
                  print("I can also count to "+str(i))
              else:
                  print("But I can't count to "+str(i))
```

```
I can count to 0
I can count to 1
I can also count to 2
I can also count to 3
But I can't count to 4
```

1.6 Writing functions

You often have to perform the same type of task many times, on different data. To avoid writing the same code over and over, you can write functions that can be called every time you want to perform the specific task.

```
In [379]: def power_of(a, b):
              return a**b
          print(power_of(2,3))
```

```
8
```

```
In [381]: print(power_of(3,5))
```

```
243
```

1.7 Reading and writing files

You can use the read, write, readlines and writelines functions from base R to read and write files. We have the examples.csv file that you saved from ELE. Let's say you are interested in what regions there are in this data. Let's start by creating a set called regions, which we will populate with the values available in the data.

```
In [306]: regions=set() # create an empty set
          with open("mydataset.csv", "r") as myfile:  # open the file for reading
              data = myfile.readlines()
              for line in data:
                  region=line.split(",")[1]
                  regions.add(region)
          print(regions)
```

```
{'East', 'North-East', 'West', 'North', 'South', 'region'}
```

Now we can open a new file, and write the regions to it:

```
In [382]: with open("regions.txt", "w") as myfile:  # open the file for writing
              for region in regions:
                  myfile.write(region + '\n' )
```

You can also append to a file in mode "a" and open it both for reading and writing in mode "r+".

```
In [383]: with open("regions.txt", "a") as myfile:  # open the file for writing
              for region in regions:
                  myfile.write(region + '\n' )
```

1.8 Importing third party modules

Everything that we've done so far was based on functions from base Python. However, we will often need to import other packages which can handle more complex or specific tasks. For example, we may want to use a module that is able to better read and write csv data, such as the 'csv' module. To do that, we have to first import the module. For packages that are already installed, you can simply do that by typing 'import' and the name of the package. Many of the useful packages are already installed in Anaconda.

But how do you know which packages are installed? If you open Anaconda Prompt, and you type "conda list", it will list all installed package. You can do this in any terminal/command prompt.

1.9 Reading data in different formats

The 'csv' module is already installed in Anaconda, so we can go ahead and import it. Let's read the file in csv format, recode missing values as NA, and write it out as a new clean.csv. The 'csv' module is very useful for manipulating large files that contain long text fields.

```
In [3]: import csv
        with open("clean.csv", "w") as outfile:
            writer=csv.writer(outfile)
            with open("mydataset.csv", "r") as infile:  # open the file for writing
                reader=csv.reader(infile)
                writer.writerow(next(reader))
                for row in reader:
                    writer.writerow(row[0:6]+[(row[6].replace("missing", "NaN"))])
```

You can also read csv files, as well as other file formats using Pandas. Pandas is one of the main libraries for data analysis in Python. For those of you familiar with R, the data frames structure and Pandas will make it very easy to use. Let's see what we can do, by importing the clean.csv file that you just saved.

```
In [4]: import pandas as pd # we import it as pd because it's easier to type
        df=pd.read_csv("clean.csv")
        df
```

```
Out[4]:       id region   party chamber    spent   raised  reelected
        0      1   East  Centre       H   285937   411847        0.0
        1      2   East  Centre       H   308530  1301546        1.0
        2      3   East  Centre       H   435962   629768        4.0
        3      4   East  Centre       H   685526   737446        3.0
```

4	5	East	Centre	H	242312	370557	1.0
5	6	East	Centre	H	149546	432485	3.0
6	7	East	Centre	H	618818	850163	2.0
7	8	East	Centre	H	354655	364555	2.0
8	9	East	Centre	H	147248	165364	0.0
9	10	East	Centre	H	306052	360675	3.0
10	11	East	Centre	H	746673	1025318	0.0
11	12	East	Centre	H	1265171	4205366	5.0
12	13	East	Centre	H	54084	100084	0.0
13	14	East	Centre	H	260806	457341	1.0
14	15	East	Centre	H	71157	237351	4.0
15	16	East	Centre	H	261123	373064	3.0
16	17	East	Centre	H	136185	571276	1.0
17	18	East	Centre	H	218830	320998	2.0
18	19	East	Centre	H	251084	700724	2.0
19	20	East	Centre	H	200186	588016	2.0
20	21	East	Centre	H	455185	557252	0.0
21	22	East	Centre	H	322763	712911	3.0
22	23	East	Centre	H	360835	539662	NaN
23	24	East	Centre	H	245872	280121	5.0
24	25	East	Centre	H	316460	370905	0.0
25	26	East	Centre	H	253641	937422	1.0
26	27	East	Centre	H	286431	834714	4.0
27	28	East	Centre	H	237073	296125	3.0
28	29	East	Centre	H	210111	1159071	1.0
29	30	East	Centre	S	66606	48704	3.0
..	...	...	...	...	...	...	...
509	510	West	Right	H	212548	218840	3.0
510	511	West	Right	H	96166	251470	2.0
511	512	West	Right	H	187035	324529	2.0
512	513	West	Right	H	272139	454256	0.0
513	514	West	Right	H	433568	843939	3.0
514	515	West	Right	H	421667	691779	0.0
515	516	West	Right	H	297943	250585	5.0
516	517	West	Right	H	208831	380948	0.0
517	518	West	Right	H	226642	420613	1.0
518	519	West	Right	H	299537	664525	4.0
519	520	West	Right	H	361325	250110	3.0
520	521	West	Right	H	170404	244851	1.0
521	522	West	Right	H	230738	455582	3.0
522	523	West	Right	H	242817	263879	2.0
523	524	West	Right	H	199311	255132	2.0
524	525	West	Right	H	372081	449664	0.0
525	526	West	Right	H	526982	990676	3.0
526	527	West	Right	H	266526	720413	0.0
527	528	West	Right	H	72911	148110	5.0
528	529	West	Right	H	291832	613410	0.0
529	530	West	Right	H	1417682	2122890	1.0

```
530  531   West   Right       H   154252   561509        4.0
531  532   West   Right       H   495108  1261714        3.0
532  533   West   Right       H   396456   967929        1.0
533  534   West   Right       H   139957   257002        3.0
534  535   West   Right       H   187541   469759        2.0
535  536   West   Right       S   112443    78720        2.0
536  537   West   Right       S   157211  1073163        0.0
537  538   West   Right       S  1692394  4631824        3.0
538  539   West   Right       S   424965   474590        0.0

[539 rows x 7 columns]
```

```
In [388]: df.head(5) # first 5 entries
```

Out[388]:

	id	region	party	chamber	spent	raised	reelected
0	1	East	Centre	H	285937	411847	0.0
1	2	East	Centre	H	308530	1301546	1.0
2	3	East	Centre	H	435962	629768	4.0
3	4	East	Centre	H	685526	737446	3.0
4	5	East	Centre	H	242312	370557	1.0

```
In [389]: df.columns # the column names
```

```
Out[389]: Index(['id', 'region', 'party', 'chamber', 'spent', 'raised', 'reelected'], dtype='ob
```

```
In [390]: df["reelected"][0:5] # select a column, and a slice within it
```

```
Out[390]: 0    0.0
          1    1.0
          2    4.0
          3    3.0
          4    1.0
          Name: reelected, dtype: float64
```

```
In [391]: # Subsetting data: create another data frame that only includes obswervations from th
          value_list=["South", "East"]
          df_SE=df[df.region.isin(value_list)] # Replace this with df[~df.region...] to keep os
          df_SE.count()
```

```
Out[391]: id           216
          region       216
          party        216
          chamber      216
          spent        216
          raised       216
          reelected    214
          dtype: int64
```

```
In [392]: # Select only dataframes that meet multiple conditions:
          df_restricted=df[(df['region']=="South") & (df["chamber"]=="S") & (df["reelected"]==0
          df_restricted.head(5)
```

```
Out[392]:       id region   party chamber    spent   raised  reelected
          356  357  South  Centre       S   199176   436192        0.0
          358  359  South  Centre       S   221402   424304        0.0
          392  393  South    Left       S  1768956  4699994        0.0
          394  395  South    Left       S  1972873    48947        0.0
          428  429  South   Right       S   719563  3231786        0.0
```

```
In [5]: # Group and aggregate
        grouped=df.groupby(["region", "chamber"])
        aggregated=grouped.agg({"spent":['sum','mean', 'min'],
                                'raised':['sum', 'mean', 'max']})
        aggregated
```

```
Out[5]:                          spent                            raised                \
                                   sum           mean    min       sum           mean
        region     chamber
        East       H         29446708  320072.913043      0  55481753  6.030625e+05
                   S          7259432  453714.500000      0  12413611  7.758507e+05
        North      H         21778335  259265.892857      0  39890400  4.748857e+05
                   S         12482798  520116.583333      0  27547225  1.147801e+06
        North-East H         27511352  348244.962025      0  53487404  6.770557e+05
                   S         13722168  490077.428571      0  43154780  1.541242e+06
        South      H         29046467  330073.488636      0  56751507  6.449035e+05
                   S         11140983  557049.150000  40206  22286751  1.114338e+06
        West       H         30186789  331722.956044  43175  59857901  6.577791e+05
                   S          6797473  399851.352941      0  15904129  9.355370e+05


                                  max
        region     chamber
        East       H         4205366
                   S         3959212
        North      H         1773323
                   S         6263060
        North-East H         4091159
                   S         9790929
        South      H         3020933
                   S         4699994
        West       H         5169778
                   S         4631824
```

1.10 Basic visualization

To display graphs inline in Jupyter notebooks make sure you add "%matplotlib inline" in the first cell.

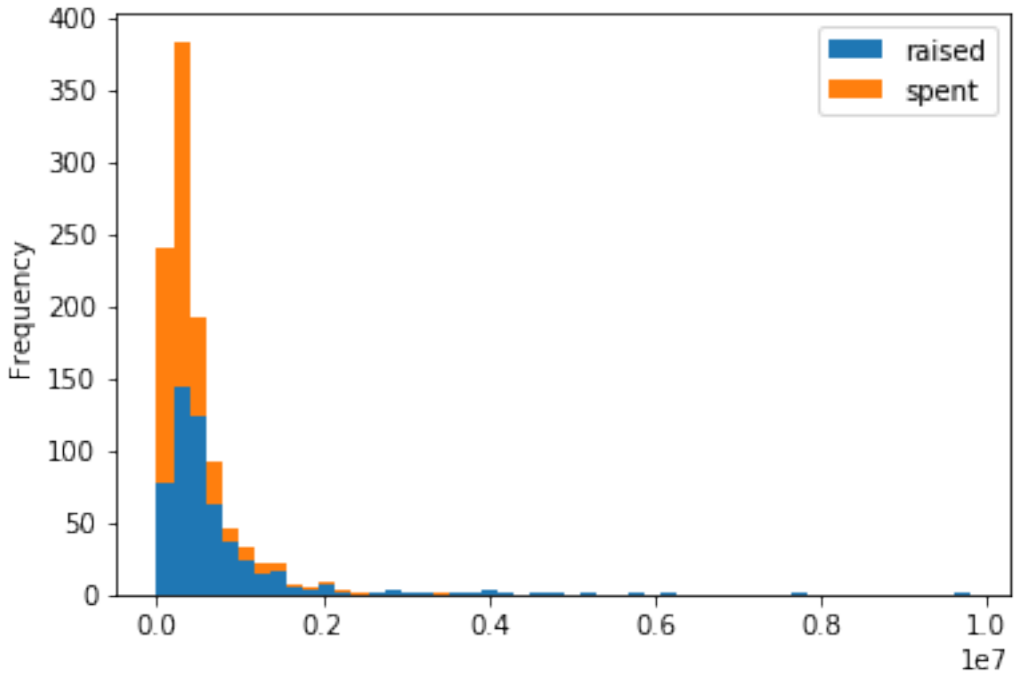
```
In [394]: %matplotlib inline
          %matplotlib notebook
```

1.10.1 Histograms, comparing two distributions.

```
In [6]: import matplotlib.pyplot as plt
        df_money=df[["raised","spent"]]
        plt.figure()
        df_money.plot.hist(stacked=True, bins=50)
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x201f099ac88>

<matplotlib.figure.Figure at 0x201f0956eb8>
```

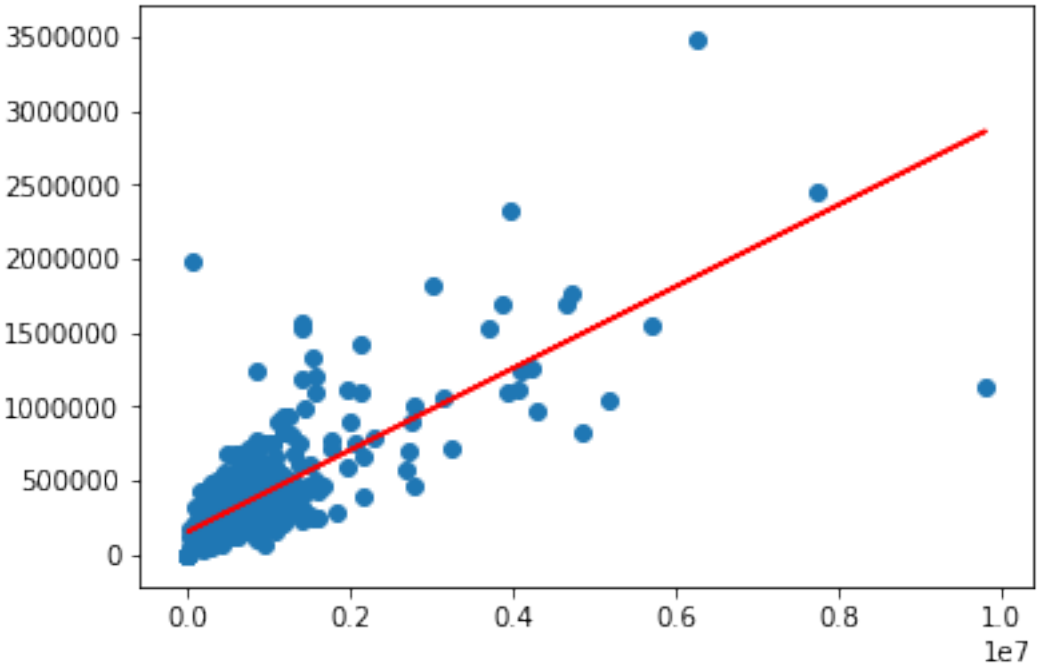


1.10.2 Scatterplot with linear fit line

```
In [7]: import numpy as np

        x=df_money.raised.values
        y=df_money.spent.values
        fig, ax = plt.subplots()
        fit = np.polyfit(x, y, deg=1)
        ax.plot(x, fit[0] * x + fit[1], color='red')
        ax.scatter(x, y)
```

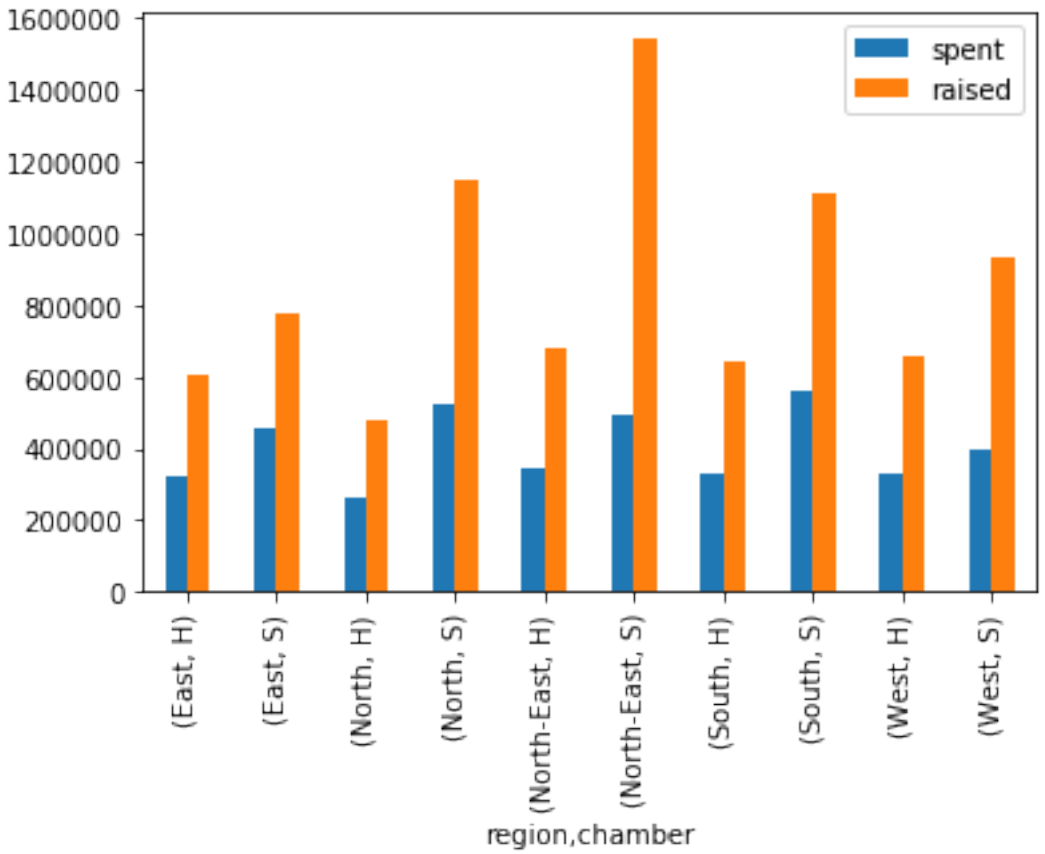
```
Out[7]: <matplotlib.collections.PathCollection at 0x201f1058cc0>
```



1.10.3 Barplots

```
In [8]: agg2=grouped.agg({"spent":"mean",
                          "raised":"mean"})
        agg2.plot.bar()
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x201f1072cc0>
```



1.11 Additional resources

1.11.1 Q-Step workshops

Term 1

7 December: **Social media data collection and analysis**

Working with the Twitter and Facebook APIs, data management, text processing and intro to text analysis, basic network analysis.

Term 2:

TBA: **Data analysis in Python**

Covering: Overview of most common packages, descriptive statistics, statistical analysis (regression, etc.), visualization.

TBA: **Text analysis in Python**

An introduction to text analysis.

1.11.2 Other beginner resources

All very hands-on, excellent for beginners, both in Python and in programming in general.

The Python Tutorial

Learn Python the Hard Way

Dive Into Python 3