Pandas

2019/10/15

lecture 3

outline

1. pandas data types
2. basic pandas functions
3. manipulating pandas data

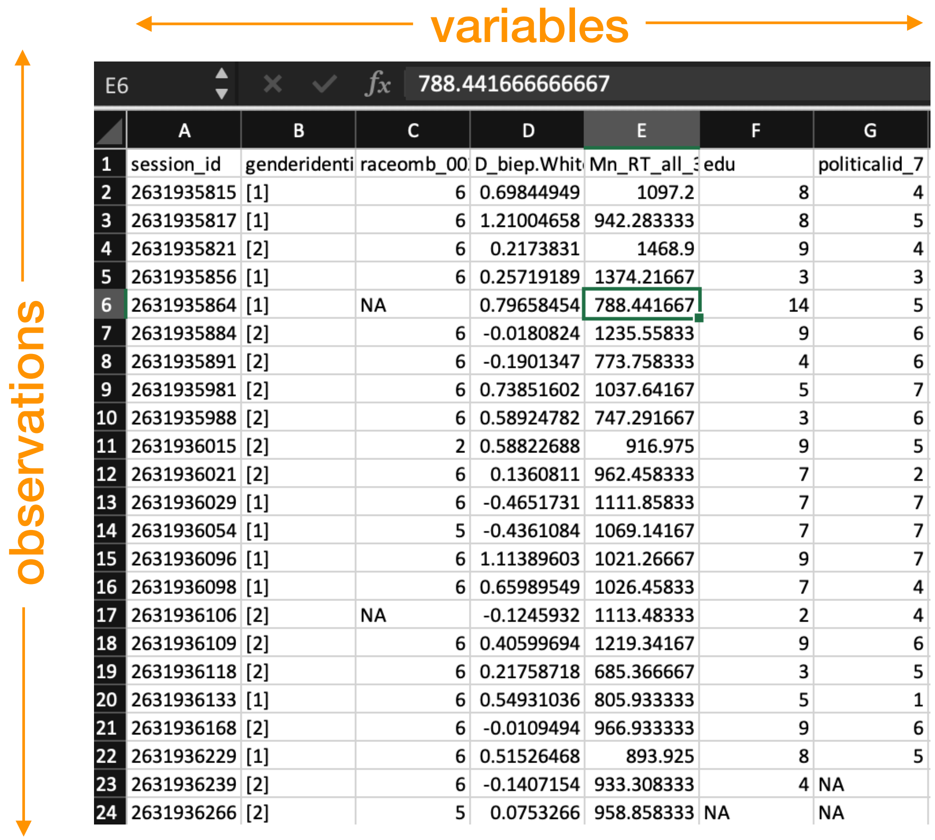
pandas data types

- overview
- series
- dataframes

data tables

native python and numpy data types let you organize data into lists/arrays

often more intuitive & efficient to use tables



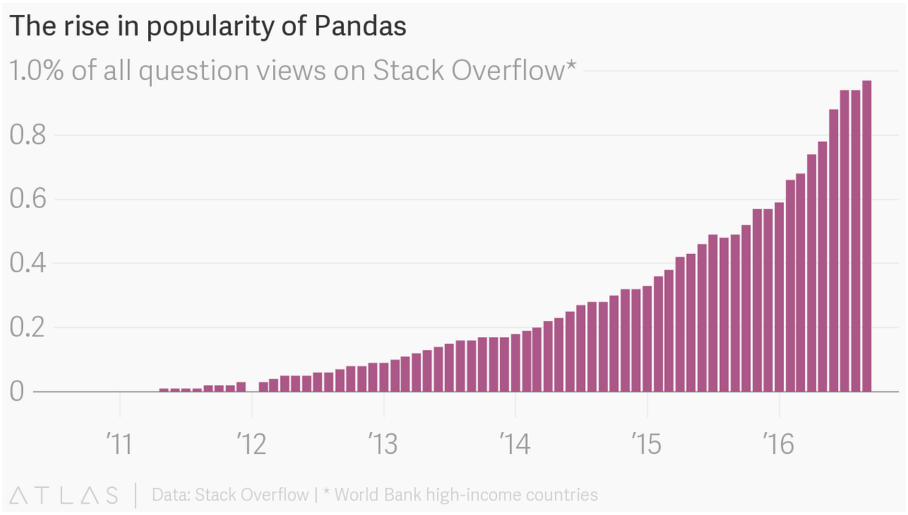
E6 788.441666666667

	A	B	C	D	E	F	G
1	session_id	genderidenti	raceomb_00	D_biep.Whit	Mn_RT_all_3	edu	politicalid_7
2	2631935815	[1]	6	0.69844949	1097.2	8	4
3	2631935817	[1]	6	1.21004658	942.283333	8	5
4	2631935821	[2]	6	0.2173831	1468.9	9	4
5	2631935856	[1]	6	0.25719189	1374.21667	3	3
6	2631935864	[1]	NA	0.79658454	788.441667	14	5
7	2631935884	[2]	6	-0.0180824	1235.55833	9	6
8	2631935891	[2]	6	-0.1901347	773.758333	4	6
9	2631935981	[2]	6	0.73851602	1037.64167	5	7
10	2631935988	[2]	6	0.58924782	747.291667	3	6
11	2631936015	[2]	2	0.58822688	916.975	9	5
12	2631936021	[2]	6	0.1360811	962.458333	7	2
13	2631936029	[1]	6	-0.4651731	1111.85833	7	7
14	2631936054	[1]	5	-0.4361084	1069.14167	7	7
15	2631936096	[1]	6	1.11389603	1021.26667	9	7
16	2631936098	[1]	6	0.65989549	1026.45833	7	4
17	2631936106	[2]	NA	-0.1245932	1113.48333	2	4
18	2631936109	[2]	6	0.40599694	1219.34167	9	6
19	2631936118	[2]	6	0.21758718	685.366667	3	5
20	2631936133	[1]	6	0.54931036	805.933333	5	1
21	2631936168	[2]	6	-0.0109494	966.933333	9	6
22	2631936229	[1]	6	0.51526468	893.925	8	5
23	2631936239	[2]	6	-0.1407154	933.308333	4	NA
24	2631936266	[2]	5	0.0753266	958.858333	NA	NA

overview

pandas is a fantastic package that brings data tables to python

it basically adds R data management functionality



*****familiar with R?*** *use this cheatsheet:* *https://pandas.pydata.org/pandas-docs/stable/comparison_with_r.html*

https://pandas.pydata.org/pandas-docs/stable/index.html also has great documentation and tutorials

series

most of pandas functionality comes from the methods attached to two object types: series and dataframes

series are the simplest type of pandas object

a series is a 1D array with a labeled index

is created by calling

```
    >> pd.Series(data)
```

where data can be a list, np.array, or dictionary

optional arguments include:

- -index: array of same length of data which will become the index; helps you keep track of your data
- -dtype: type of data (e.g. str, int)
- -name: name of series; useful when combine series into a dataframe

dataframe

pandas' most common object type

a 2D table-like object with labeled columns and row indices

like a tidy excel sheet, an R data.frame, or matlab table

is created by calling

```
    >> pd.DataFrame(data)
```

where data can be a np.array, dictionary, or pd.Series

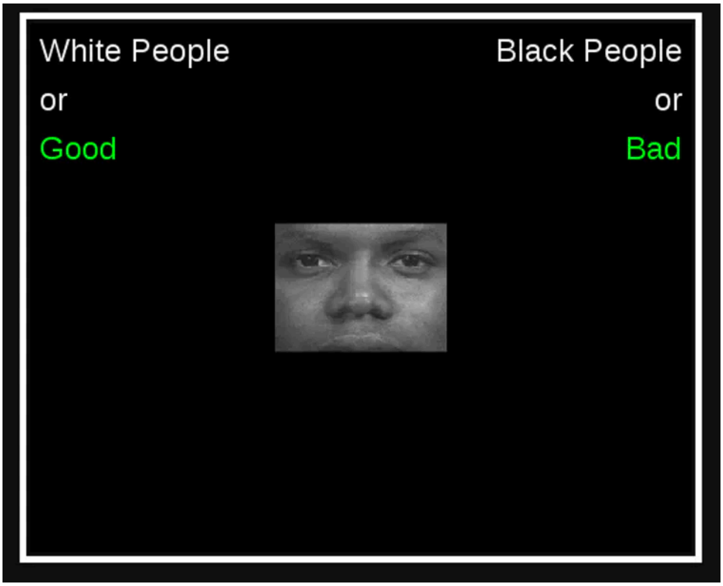
optional arguments include:

- -columns: array of column names
- -index: array of same length of data which will become the index
- -dtype: type of data (e.g. str, int)

basic pandas functions

- reading & writing
- indexing
- editing
- missing values

IAT data



Will be using data collected in 2018 stored on https://osf.io/52qxl/

389669 subjects worth of IAT results + demographic data

included only a small subset of possible variables — check out the codebook for details on all variables included on osf

reading data

pandas has functions for reading files:

```
>>pd.read_csv('filename.csv')
```

be sure to include the full path to the file

there are a ton of options to help deal with messy files: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

```
>>pd.read_excel('filename.xlsx')
```

you can also include specific sheet names

requires xlrd, which you can install with this command in your terminal after activating your conda environment:

```
$ pip install xlrd
```

writing data

pandas has functions for writing files:

```
>>pd.to_csv('filename.csv')
```

be sure to include the full path to the file

there are also a lot of ways to tweak file writing

e.g. add: “mode=‘a’, header = False” to append to an existing file

you can also write directly to excel files with pd.to_excel()

indexing

dataframes have two dimensions so you need to access data by specifying the (row, column)

pandas has specialized methods for indexing

if you want to use the names of columns use:

```
    >>df.loc[index, column]
```

** note that the first input is the index label not the row number

if you want to use row and column numbers use:

```
    >>df.iloc[row_num, column_num]
```

both loc and iloc accept slices (e.g. [0:3], [‘var1’:’var3’]) and logical statements

editing dataframes

change the order of columns:

```
>>df = df[[list of variables in new order]]
```

change the name of a column:

```
>>df = df.rename(columns={‘oldname’ : ‘newname’})
```

sort according to columns:

```
>>df = df.sort_values(by=[‘var1’,’var2'])
```

reset the index:

```
>>df = df.reset_index(drop=True)
```

add a column:

```
>>df[‘newvar’] = values
```

replace a value:

```
>>df[df==0] = -1
```

*you can use any type of indexing to replace values

missing values

pandas has some special methods to manage missing data

To find missing data use:

```
>>df.isna()
>>df.isnull()
```

To remove rows or columns with missing data use:

```
>>df.dropna(how, axis)
```

where, how can be ‘any’ or ‘all’ and axis=0 drops rows and axis=1 drops columns

To fill missing data use:

```
>>df.fillna(value)
```

to replace all missing data with a specific value; you can also use this method to replace with the previous or next value

manipulating pandas data

- basic stats
- pivot_table
- crosstab
- merging

basic stats

pandas is built on NumPy so you can use any NumPy stats method on a pandas object

you can also specify an axis to determine whether the stat is performed row- or column-wise.

example stats are:

```
>>df.mean()
>>df.median()
>>df.sum()
>>df.std()
>>df.count()
>>df.min()
>>df.max()
>>df.abs()
```

pivot table

pivot_table works like excel pivot tables!

you basically build a new dataframe however you like, using the variables in a dataframe and a summary statistic

>> pd.pivot_table(dataframe, values, index, columns, aggfunc, margins)

- values — list of variables that will be summarized (usually dependent variables)
- index — list of variables that will organize rows (often subject number)
- columns — list of variables that will organize columns (often independent variables)
- aggfunc — NumPy statistic (default mean)
- margins — includes column and row means when True

crosstab

crosstab comes in handy for building frequency tables

```
>> pd.crosstab(rowSeries,colSeries)
```

the first series organizes rows and the second organizes columns

you can use lists of series for more complex tables

include normalize=True to get the proportion of all data in each cell

include normalize=‘columns’ to get the proportion of each column a cell

include normalize=‘index’ to get the proportion of each row in a cell

merging

you can stack dataframes using:

```
>> pd.concatenate(list, axis)
```

where list contains series or dataframes

axis = 1 will stack vertically

axis = 0 will stack horizontally

you merge data frames according to key variables using

```
>> pd.merge(df1, df2, on)
```

were df1 and df2 are dataframes that contain at least one linking variable

on takes a list of the linking variable names

* if the linking variables have different column names, you can use left_on and right_on