LAB #1: EXPLORATORY DATA ANALYSIS

CS 109A, STAT 121A, AC 209A: Data Science

Fall 2016

Harvard University

- Solutions to HW0 Part B is on Canvas (under Assignments)
- Non-AC 209 students: each Challenge Problem with be +0.5% towards your final grade.
- Our philosophy for HW:
  - All homework problems (except for Challenge problems) can be completed using only the concepts/tools/programming patterns discussed in class/lab/lab materials. **You are free to do more**.
  - If we want to show you a new trick in the solutions, **we will give an explanation in the solutions**.
  - Homework solutions will try to stick to simple and readable code. **You are free to do fancier programming**.

ANNOUNCEMENTS

Question: What libraries/packages am I allowed to use?

- If the question explicitly tells you to use a library/package, then do that!

  Part (a): Reading CSV Data with Numpy

  Open the file `dataset_HW0.txt`, containing birth biometrics as well as maternal data for a number of U.S. births, and inspect the csv formatting of the data. Load the data, without the column headers, into an **numpy array**.

  Do some preliminary explorations of the data by printing out the dimensions as well as the first three rows of the **array**. Finally, for each **column**, print out the range of the values.

- If the question doesn't explicitly specify the tools you must use, then do what you like (**within reason**)!

  Part (e): Sentence and Word Count

  Count the number of words in The Metamorphosis. Compute the average word length and plot a histogram of word lengths.

  Count the number of sentences in The Metamorphosis. Compute the average sentence length and plot a histogram of sentence lengths.

Rules for being fancy: Use only packages that are included with the standard Anaconda install. Code to communicate, not to impress or intimidate!

Question: What cool analysis/concepts/models am I allowed to use?

- If the question explicitly tells you to use a concept/model/analysis, then do that!
- If the questions doesn't explicitly tells you to use a concept/model/analysis, then do what you like (**within reason**)
- Again, **all homework problems (except for Challenge problems) can be completed using only the concepts/tools/programming patterns discussed in class/lab/lab materials**.
- Labs, solutions will not use jargon that is not defined in lecture/lab/readings. **You are free do use more (within reason)**.

Question: What should your HW submission look like?

- File name should be formatted:

`last_first_CourseNumber_HW1.ipynb`

- Exposition, narrative, explanations should be in markdown cells.
- Code should be in code cells.
- Cells need to be executed or rendered. I.e. I should be able to view your notebook like a static final report, with properly formatted paragraphs, code and outputs of code, **without having to execute a single cell**!

Working with `numpy` has it's draw-backs!

- It's not easy to read tabular data from csv files where the values are mixed in type (some strings, some floats)
- It's not easy to read in and store the column headers (strings) in the numpy array representing the data
- We can only reference columns by position rather than column header. E.g. I want the "height" column, but I need to remember that it's the 3rd column in the array.

Wish List: we want a data structure

- that can easily store variables of different types
- that stores column names
- where we can reference column by position as well as by column name
- that comes with built in functions for manipulating this structure

Answer: Python has a package/library for that, `pandas`.

Question: Why learn/use `numpy`?

Answer:

- `pandas` is a set of objects and tools built on top of `numpy`.
- computing with `pandas` data structures vs `numpy` arrays can mean performance difference!

INTRODUCTION TO pandas

pandas objects can be thought of as “enhanced” versions of numpy arrays in which the rows and columns are identified with labels rather than integers. Basics pandas objects:

- **Series:** 1D array of data with an index object (labels).

```
In [4]: import pandas as pd

        column = pd.Series([0.25, 0.5, 0.75, 1.0],
                           index=['one', 'two', 'three', 'four'])
        column

Out[4]: one      0.25
        two      0.50
        three    0.75
        four     1.00
        dtype: float64

In [6]: column['four']

Out[6]: 1.0
```

  Each series has a “values” component and an “index” component. Series come with the same built-in functions, like mean, min, std, as numpy arrays.

pandas objects can be thought of as “enhanced” versions of numpy arrays in which the rows and columns are identified with labels rather than integers. Basics pandas objects:

- **Series:** 1D array of data with an index object (labels).

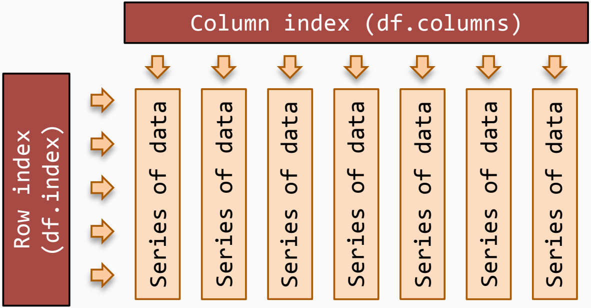
```
In [7]: column.index
Out[7]: Index([u'one', u'two', u'three', u'four'], dtype='object')

In [11]: column.values
Out[11]: array([ 0.25,  0.5 ,  0.75,  1.  ])
```

  Each series has a “values” component and an “index” component. Series come with the same built-in functions, like mean, min, std, as numpy arrays.

`pandas` objects can be thought of as “enhanced” versions of `numpy` arrays in which the rows and columns are identified with labels rather than integers. Basics `pandas` objects:

- **DataFrame:** 2D table of data with column and row index objects (labels).



Each column in the data frame is a *series*.

We can create a data frame from columns (series objects):

```
In [15]: column_1 = pd.Series(range(4),
                              index=['one', 'two', 'three', 'four'])

         column_2 = pd.Series(range(4, 8),
                              index=['one', 'two', 'three', 'four'])

         table = pd.DataFrame({'col_1': column_1,
                               'col_2': column_2})

         table
```

Out[15]:

	col_1	col_2
one	0	4
two	1	5
three	2	6
four	3	7

We can import tabular data in a csv file into a data frame:

In [16]:
```
df = pd.read_csv('dataset_HW0.txt')
df
```

Out[16]:

	birth_weight	femur_length	mother_age
0	2.969489	1.979156	16
1	4.038963	3.555681	16
2	5.302643	3.385633	15
3	6.086107	4.495427	17

We should start by getting a rough sense of what's in the data

- **The indices of your data frame:**

```
In [32]: df.columns
Out[32]: Index([u'birth_weight', u'femur_length', u'mother_age'], dtype='object')

In [5]: df.columns.values
Out[5]: array(['birth_weight', 'femur_length', 'mother_age'], dtype=object)

In [6]: df.index
Out[6]: Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
            ...
            390, 391, 392, 393, 394, 395, 396, 397, 398, 399],
           dtype='int64', length=400)

In [7]: df.index.values
Out[7]: array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
              13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
              26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
              39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
```

The `.index` and `.columns` attributes give access to the index objects of rows and columns (resp).

We should start by getting a rough sense of what's in the data

- **The shape of your data frame:**

```
In [23]: df.shape
Out[23]: (400, 3)

In [10]: len(df.index)
Out[10]: 400
```

We should start by getting a rough sense of what's in the data

- The first entries in your data frame:

```
In [25]: df.head(n=5)
```

Out[25]:

	birth_weight	femur_length	mother_age
0	2.969489	1.979156	16
1	4.038963	3.555681	16
2	5.302643	3.385633	15
3	6.086107	4.495427	17
4	5.749260	4.017437	16

The `.head()` function returns a (row-wise) truncated version of your data frame!

We should start by getting a rough sense of what's in the data

- A summary of your data frame:

```
In [29]: df.describe()
```

Out[29]:

	birth_weight	femur_length	mother_age
count	400.000000	400.000000	400.00000
mean	6.104070	3.827591	27.06000
std	1.097011	0.853577	10.34984
min	2.967426	0.479154	15.00000
25%	5.429120	3.281786	17.75000
50%	6.110025	3.817888	25.00000
75%	6.839935	4.351204	34.25000
max	9.021942	6.648730	49.00000

The `.describe()` function returns all the descriptive stats for each column as a data frame object!

- Accessing a column by label:

```
In [34]: type(df['birth_weight'])

Out[34]: pandas.core.series.Series

In [35]: df['birth_weight']

Out[35]: 0       2.969489
         1       4.038963
         2       5.302643
         3       6.086107
         4       5.749260
         5       6.049903
```

You can access a column by it's column name or position (you can also access a *list* of columns)!

- Accessing the values of column:

```
In [36]: df['birth_weight'].values

Out[36]: array([ 2.9694893 ,  4.03896294,  5.30264328,  6.08610661,  5.74926036,
                 6.04990317,  5.42681579,  6.23910323,  5.34504952,  4.16297458,
                 5.27487188,  5.57627684,  5.49364519,  6.66031745,  4.79466787,
                 5.98546786,  4.62521954,  5.60683336,  4.52477222,  6.3162985 ,
                 5.5922901 ,  6.23730155,  5.19645533,  4.61051962,  4.38347209,
                 5.00708476,  4.10801732,  5.18226899,  3.91916625,  5.8955964 ,
```

You can access a column by it's column name or position (you can also access a *list* of columns)!

- Accessing columns by position:

```
In [63]: df[[0, 1]]
```

Out[63]:

	birth_weight	femur_length
0	2.969489	1.979156
1	4.038963	3.555681
2	5.302643	3.385633
3	6.086107	4.495427
4	5.749260	4.017437
5	6.049903	4.378892
6	5.426816	2.851801

You can access a column by it's column name or position (you can also access a *list* of columns)!

- Accessing a row by position:

```
In [46]: type(df.iloc[0])

Out[46]: pandas.core.series.Series

In [47]: df.iloc[0]

Out[47]: birth_weight      2.969489
         femur_length      1.979156
         mother_age       16.000000
         Name: 0, dtype: float64
```

You can access a column by it's row name or position!

- Accessing a row by label:

```
In [52]: table
```

Out[52]:

	col_1	col_2
one	0	4
two	1	5
three	2	6
four	3	7

```
In [54]: type(table.loc['one'])
Out[54]: pandas.core.series.Series

In [53]: table.loc['one']
Out[53]: col_1    0
         col_2    4
         Name: one, dtype: int64
```

You can access a column by it's row name or position!

Filtering works very much like with `numpy` arrays!

```
In [57]: df[(df['mother_age'] > 18) & (df['mother_age'] < 35)]
```

Out[57]:

	birth_weight	femur_length	mother_age
100	6.904530	4.164637	34
101	8.096642	4.536759	22
102	8.165373	5.507030	20
104	6.255286	3.769024	19
105	6.515220	5.568954	23
106	6.464462	3.310628	25
107	6.579616	3.670224	20
108	7.171024	5.159946	24

Slides at:

https://github.com/cs109Alabs/lab_files/tree/master/Lab_1

- You can slice (just like numpy arrays) using `.iloc`:
  `df.iloc[:, :2]`
- You can add a column to an existing data frame:
  `df['new_col'] = (some ''list" of values)`
- You can add a row to an existing data frame:
  `df.loc['new_row'] = (some ''list" of values)`
- You can compute the mean (or max, min or std etc) for **every column in a dataframe**:
  `df.mean()`
- You can generate plots directly from the values in a column:
  `df['some_column'].hist()`
- More fun with data frame indexing
  http://chrisalbon.com/python/pandas_indexing_selecting.html
- pandas cheatsheet
  http://www.webpages.uidaho.edu/~stevel/504/Pandas%20DataFrame%20Notes.pdf
- Another intro to pandas
  https://www.oreilly.com/learning/introducing-pandas-objects

TASKS

- Open the notebook for HW 1 (you should have already downloaded the starter files)
- Do parts (a) and (b)
- We will regroup and do parts (c) and (d)