GENERAL ASSEMBLY

USING PYTHON IN DATA SCIENCE

I. QUICK OVERVIEW OF PYTHON, ANACONDA, AND SPYDER

II. SCRIPTING & THE IPYTHON INTERPRETER

III. PANDAS

- A. LOADING & VIEWING DATA
- B. INDEXING AND SELECTING DATA
- C. ASSIGNING, REASSIGNING, & SPLITTING DATA
- D. DESCRIBING AND SUMMARIZING DATA
- E. PLOTTING DATA

PYTHON, ANACONDA, AND SPYDER OVERVIEW

- Python is a dynamically-typed scripting language popular with web applications, scientific computing, and backend / ETL work.

- The focus of Python is on simplicity and code readability – with a mantra that “there should only be one obvious way to do it.”

- Python is implemented via CPython, and many Python packages call C explicitly to speed up execution.

- Although 5-10x faster than R, Python is typically 5-8x slower than Java due to its dynamic typing and inference.

WHY USE PYTHON?

- Fast development due to concise syntax and REPL
- Clear, plain language and error messages
- Used in real-world production environments
- Useful help() function
- Lots of scientific package written in it
- In most areas, matches the functionality of R

ANACONDA, IPYTHON, & SPYDER

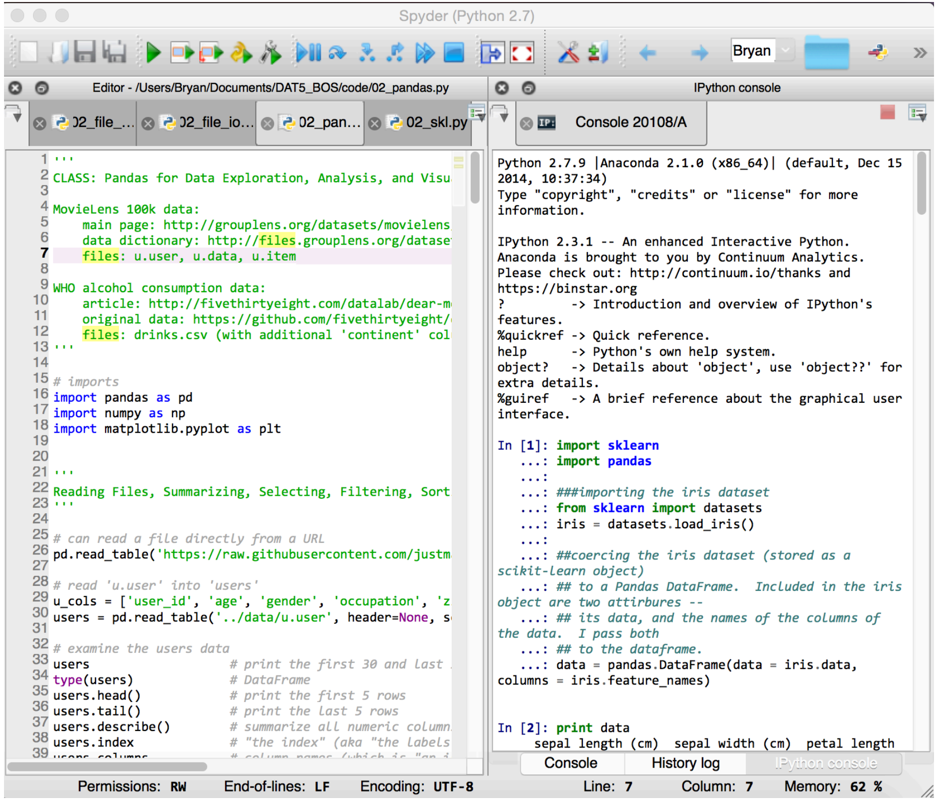
- Anaconda is a free Python distribution that includes all the scientific packages you need for this class.

- Anaconda includes iPython, an enhanced interactive shell for scientific computing, and Spyder, a great scientific IDE for Python

- Why not use iPython Notebooks? Because we want to think like software developers building for production!

THE SPYDER IDE

- Open Anaconda Launcher and select ‘spyder-app’
- Your python script is to the left, and the iPython REPL (or console) is to the right.
- Press Command-Enter to execute code from the editor in the console.
- Automatic code completion is done using the tab key



```
'''
CLASS: Pandas for Data Exploration, Analysis, and Visu

MovieLens 100k data:
    main page: http://grouplens.org/datasets/movielens
    data dictionary: http://files.grouplens.org/datase
    files: u.user, u.data, u.item

WHO alcohol consumption data:
    article: http://fivethirtyeight.com/datalab/dear-m
    original data: https://github.com/fivethirtyeight/
    files: drinks.csv (with additional 'continent' col
'''

# imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


'''
Reading Files, Summarizing, Selecting, Filtering, Sort
'''

# can read a file directly from a URL
pd.read_table('https://raw.githubusercontent.com/justm

# read 'u.user' into 'users'
u_cols = ['user_id', 'age', 'gender', 'occupation', 'z
users = pd.read_table('../data/u.user', header=None, s

# examine the users data
users                   # print the first 30 and last
type(users)             # DataFrame
users.head()            # print the first 5 rows
users.tail()            # print the last 5 rows
users.describe()        # summarize all numeric column
users.index             # "the index" (aka "the labels
users.columns           # column names (which is "an i
```

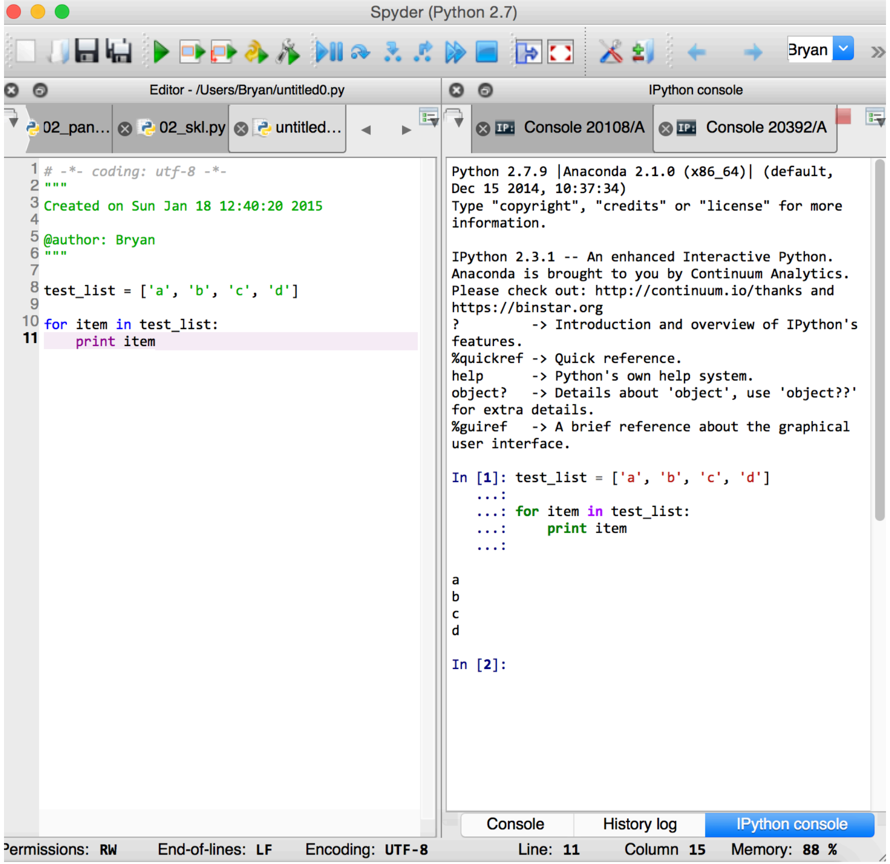
```
Python 2.7.9 |Anaconda 2.1.0 (x86_64)| (default, Dec 15
2014, 10:37:34)
Type "copyright", "credits" or "license" for more
information.

IPython 2.3.1 -- An enhanced Interactive Python.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and
https://binstar.org
?         -> Introduction and overview of IPython's
features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for
extra details.
%guiref   -> A brief reference about the graphical user
interface.

In [1]: import sklearn
   ...: import pandas
   ...:
   ...: ###importing the iris dataset
   ...: from sklearn import datasets
   ...: iris = datasets.load_iris()
   ...:
   ...: ##coercing the iris dataset (stored as a
scikit-learn object)
   ...: ## to a Pandas DataFrame.  Included in the iris
object are two attirbures --
   ...: ## its data, and the names of the columns of
the data.  I pass both
   ...: ## to the dataframe.
   ...: data = pandas.DataFrame(data = iris.data,
columns = iris.feature_names)


In [2]: print data
      sepal length (cm)  sepal width (cm)  petal length
```

EXERCISE: LISTS AND LOOPS IN SPYDER

- Create a new file named test.py in your ‘My Documents’ folder.

- In the script:
  - Instantiate a list of four strings
  - Print the first three items in the list
  - Write a for loop to print out each string individually
  - Using tab completion, append a fifth string to the list

- Run the script in the iPython console using command-enter.



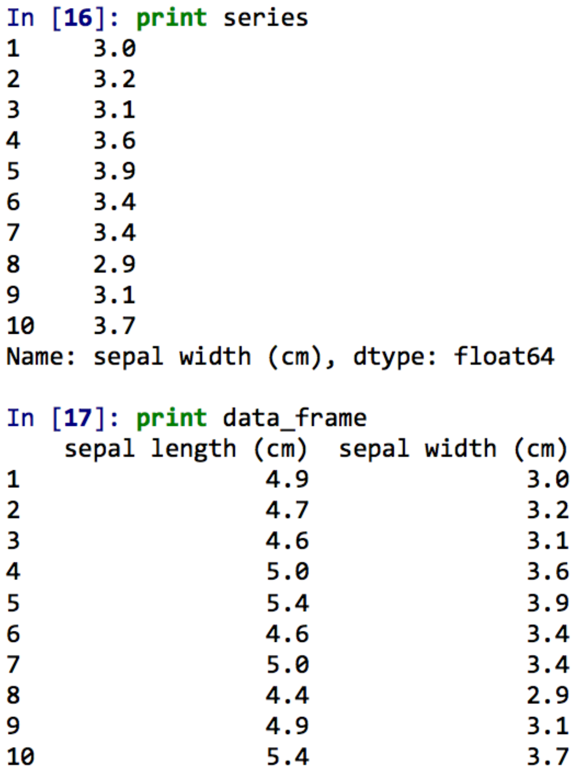
```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 18 12:40:20 2015

@author: Bryan
"""

test_list = ['a', 'b', 'c', 'd']

for item in test_list:
    print item
```

```
Python 2.7.9 |Anaconda 2.1.0 (x86_64)| (default,
Dec 15 2014, 10:37:34)
Type "copyright", "credits" or "license" for more
information.

IPython 2.3.1 -- An enhanced Interactive Python.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and
https://binstar.org
?         -> Introduction and overview of IPython's
features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??'
for extra details.
%guiref   -> A brief reference about the graphical
user interface.

In [1]: test_list = ['a', 'b', 'c', 'd']
   ...:
   ...: for item in test_list:
   ...:     print item
   ...:

a
b
c
d

In [2]:
```

Permissions: RW   End-of-lines: LF   Encoding: UTF-8   Line: 11   Column 15   Memory: 88 %

PANDAS

- Unlike R, Python does not have a built-in data type for tabular data.
- Pandas fills that gap with its Series and DataFrame objects.
- Series and DataFrames have two basic axes: an index axis and a column index.
- Series represent a single column of data, while a DataFrame represents multiple Series sharing a common index.



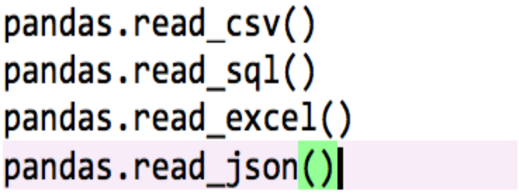
```
In [16]: print series
1     3.0
2     3.2
3     3.1
4     3.6
5     3.9
6     3.4
7     3.4
8     2.9
9     3.1
10    3.7
Name: sepal width (cm), dtype: float64

In [17]: print data_frame
    sepal length (cm)  sepal width (cm)
1                 4.9               3.0
2                 4.7               3.2
3                 4.6               3.1
4                 5.0               3.6
5                 5.4               3.9
6                 4.6               3.4
7                 5.0               3.4
8                 4.4               2.9
9                 4.9               3.1
10                5.4               3.7
```

LOADING AND VIEWING DATA IN PANDAS

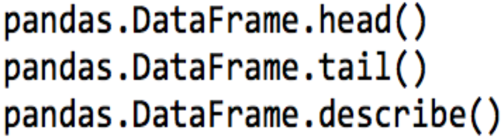
- Pandas makes loading data from external sources into DataFrames easy via its read.*() methods such as read.csv, read.sql, read.json, and read.excel.

- Viewing the data in the DataFrame is also very easy to do: use the head(), tail(), and describe() functions to get an overview of the data.

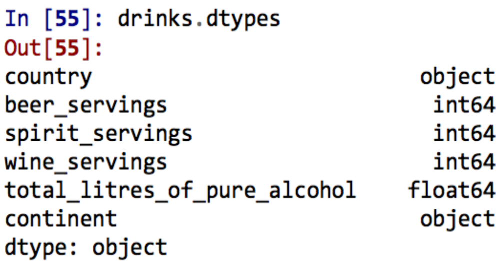
- Pandas data is in five basic types: int, float, Boolean, object, and category.



```
pandas.read_csv()
pandas.read_sql()
pandas.read_excel()
pandas.read_json()
```



```
pandas.DataFrame.head()
pandas.DataFrame.tail()
pandas.DataFrame.describe()
```



```
In [55]: drinks.dtypes
Out[55]:
country                         object
beer_servings                    int64
spirit_servings                  int64
wine_servings                    int64
total_litres_of_pure_alcohol   float64
continent                       object
dtype: object
```

EXERCISE: LOADING AND VIEWING DATA

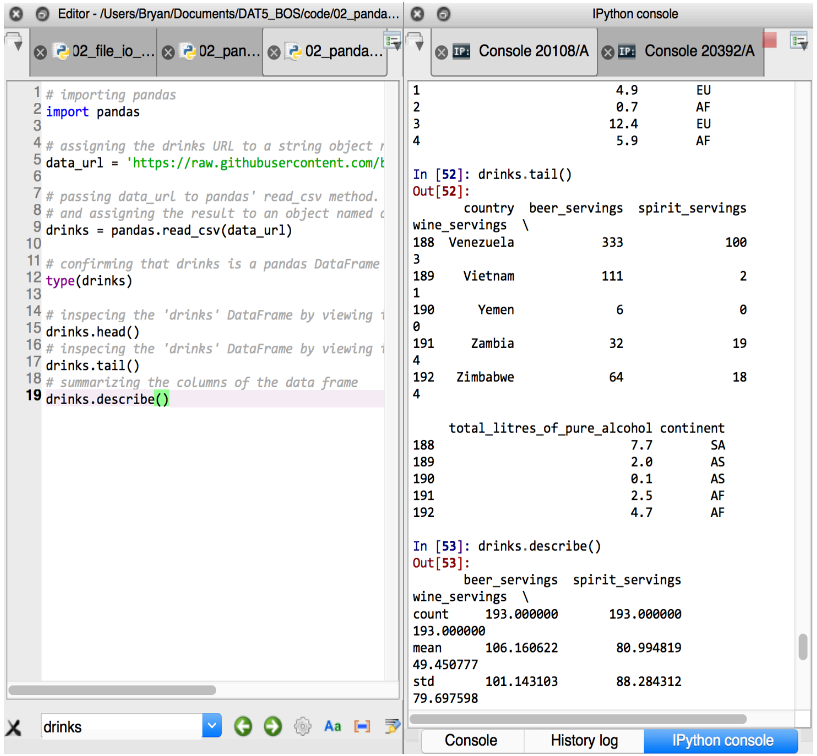
1. Import Pandas

2. Copy and paste the URL I passed to you on Slack to the script editor.

3. assign it to an object named data_url.

4. Pass data_url to pandas' read.csv() function and assign the results to an object named 'drinks'.

5. Inspect the data with drinks.head(), drinks.tail(), and drinks.describe().



```python
# importing pandas
import pandas

# assigning the drinks URL to a string object r
data_url = 'https://raw.githubusercontent.com/t

# passing data_url to pandas' read_csv method.
# and assigning the result to an object named d
drinks = pandas.read_csv(data_url)

# confirming that drinks is a pandas DataFrame
type(drinks)

# inspecing the 'drinks' DataFrame by viewing i
drinks.head()
# inspecing the 'drinks' DataFrame by viewing i
drinks.tail()
# summarizing the columns of the data frame
drinks.describe()
```

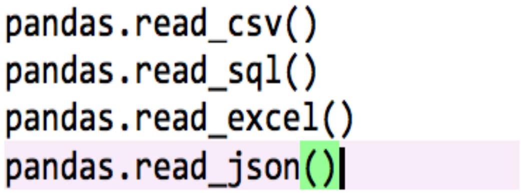
```
1                            4.9        EU
2                            0.7        AF
3                           12.4        EU
4                            5.9        AF

In [52]: drinks.tail()
Out[52]:
       country  beer_servings  spirit_servings
wine_servings  \
188  Venezuela            333              100
3
189    Vietnam            111                2
1
190      Yemen              6                0
0
191     Zambia             32               19
4
192   Zimbabwe             64               18
4

     total_litres_of_pure_alcohol continent
188                           7.7        SA
189                           2.0        AS
190                           0.1        AS
191                           2.5        AF
192                           4.7        AF

In [53]: drinks.describe()
Out[53]:
       beer_servings  spirit_servings
wine_servings  \
count     193.000000       193.000000
193.000000
mean      106.160622        80.994819
49.450777
std       101.143103        88.284312
79.697598
```

INDEXING AND SELECTING DATA

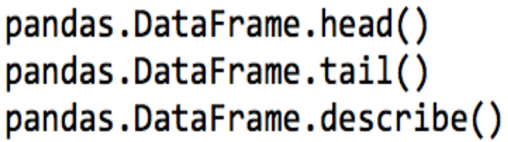
- Pandas data is arranged along an index (rows) and columns.

- There are three primary ways to access data within a DataFrame: by name, by index location, and by Boolean
    1. Name: drinks[‘country’] or drinks.country
    2. Index: drinks.ix[0:3, 0]
    3. Boolean: drinks.country[drinks.index < 3]

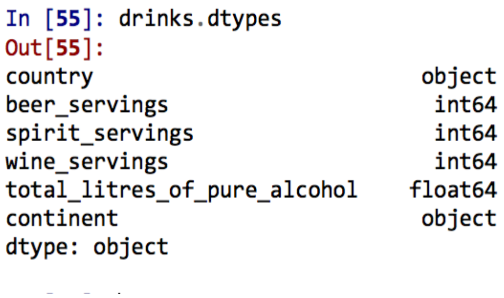
- Series use the same exact logic, except that all their operators are along a single (vertical) axis!



```
pandas.read_csv()
pandas.read_sql()
pandas.read_excel()
pandas.read_json()
```



```
pandas.DataFrame.head()
pandas.DataFrame.tail()
pandas.DataFrame.describe()
```



```
In [55]: drinks.dtypes
Out[55]:
country                          object
beer_servings                     int64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

EXERCISE: INDEXING AND SELECTING DATA

1. Get the column names of the ‘drinks’ DataFrame via drinks.columns.

2. Select only the column labeled ‘country’ using selecting by name.

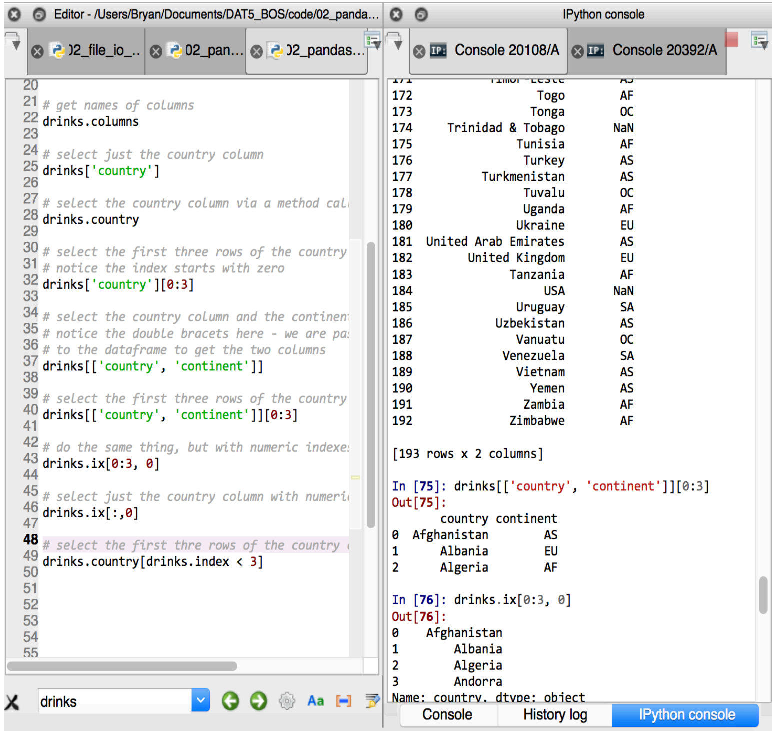
3. Select the first three rows of data in the column.

4. Select the ‘country’ and ‘continent’ columns in ‘drinks’.

5. Select the first three rows in the two columns.

6. Repeat the same exercise using numeric indices only.

7. Select the first three rows of the column ‘country’ using Boolean indexing.



```

# get names of columns
drinks.columns

# select just the country column
drinks['country']

# select the country column via a method cal
drinks.country

# select the first three rows of the country
# notice the index starts with zero
drinks['country'][0:3]

# select the country column and the continen
# notice the double bracets here - we are pa
# to the dataframe to get the two columns
drinks[['country', 'continent']]

# select the first three rows of the country
drinks[['country', 'continent']][0:3]

# do the same thing, but with numeric indexe
drinks.ix[0:3, 0]

# select just the country column with numeri
drinks.ix[:,0]

# select the first thre rows of the country
drinks.country[drinks.index < 3]
```

```
172                 Togo        AF
173                Tonga        OC
174    Trinidad & Tobago       NaN
175              Tunisia        AF
176               Turkey        AS
177         Turkmenistan        AS
178               Tuvalu        OC
179               Uganda        AF
180              Ukraine        EU
181 United Arab Emirates        AS
182       United Kingdom        EU
183             Tanzania        AF
184                  USA       NaN
185              Uruguay        SA
186           Uzbekistan        AS
187              Vanuatu        OC
188            Venezuela        SA
189              Vietnam        AS
190                Yemen        AS
191               Zambia        AF
192             Zimbabwe        AF

[193 rows x 2 columns]

In [75]: drinks[['country', 'continent']][0:3]
Out[75]:
       country continent
0  Afghanistan        AS
1      Albania        EU
2      Algeria        AF

In [76]: drinks.ix[0:3, 0]
Out[76]:
0    Afghanistan
1        Albania
2        Algeria
3        Andorra
Name: country, dtype: object
```

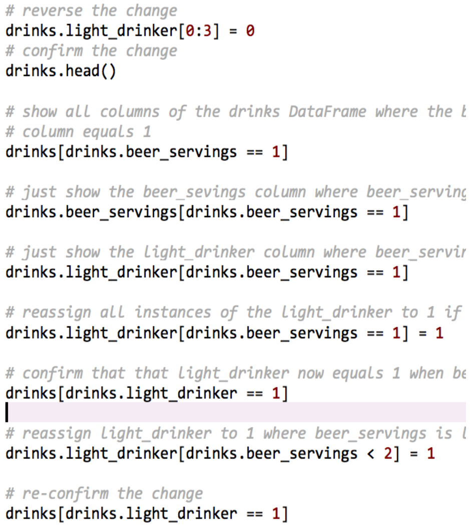
ASSIGNING, REASSIGNING, AND SPLITTING OUT DATA

- New columns in a DataFrame can be arbitrarily created using the df[‘name’] = value syntax.

- Data in a Series or DataFrame can be reassigned using the = sign.

- You can make conditional reassignments (say, assign all instances where a row’s value equals 1 to zero) using Boolean indexing.

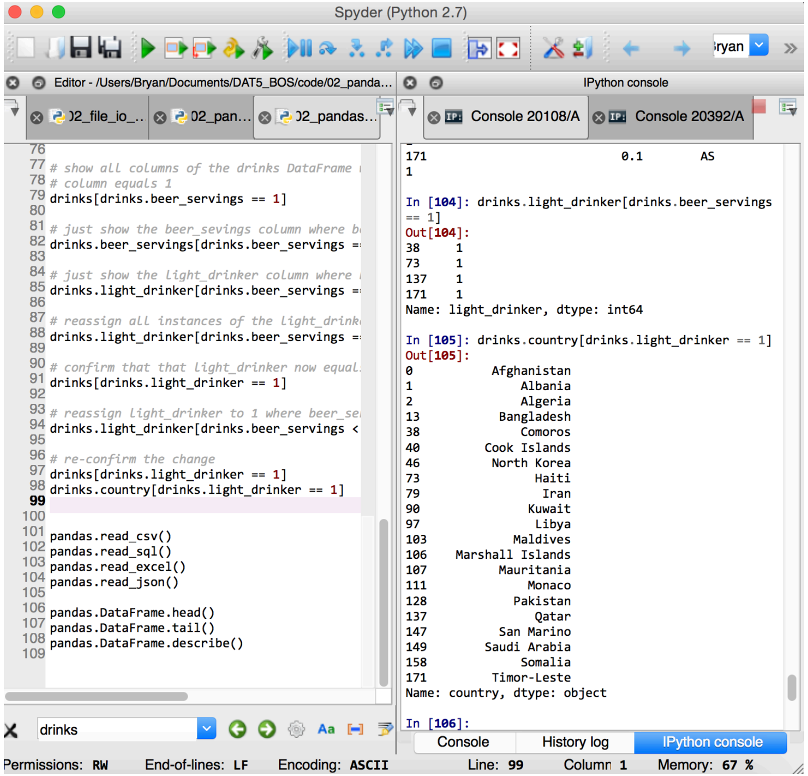
- You can also use Boolean indexing to reassign values in one column when they equal some value in ***another*** column.



```
# reverse the change
drinks.light_drinker[0:3] = 0
# confirm the change
drinks.head()

# show all columns of the drinks DataFrame where the b
# column equals 1
drinks[drinks.beer_servings == 1]

# just show the beer_sevings column where beer_serving
drinks.beer_servings[drinks.beer_servings == 1]

# just show the light_drinker column where beer_servi
drinks.light_drinker[drinks.beer_servings == 1]

# reassign all instances of the light_drinker to 1 if
drinks.light_drinker[drinks.beer_servings == 1] = 1

# confirm that that light_drinker now equals 1 when be
drinks[drinks.light_drinker == 1]

# reassign light_drinker to 1 where beer_servings is l
drinks.light_drinker[drinks.beer_servings < 2] = 1

# re-confirm the change
drinks[drinks.light_drinker == 1]
```

EXERCISE: ASSIGNING, REASSIGNING, AND SPLITTING OUT DATA

1. Arbitrarily create a column called ‘light_drinker’ and assign it a value of zero.
2. Reassign the first three rows of the light_drinker column to have 1 as their value.
3. Reverse this change.
4. Show all columns of the DataFrame where beer_servings equals 1
5. Reassign the light_drinker column to 1 **only where** beer_servings equals 1
6. Do the same where beer_servings is less than 2.
7. Confirm this change.

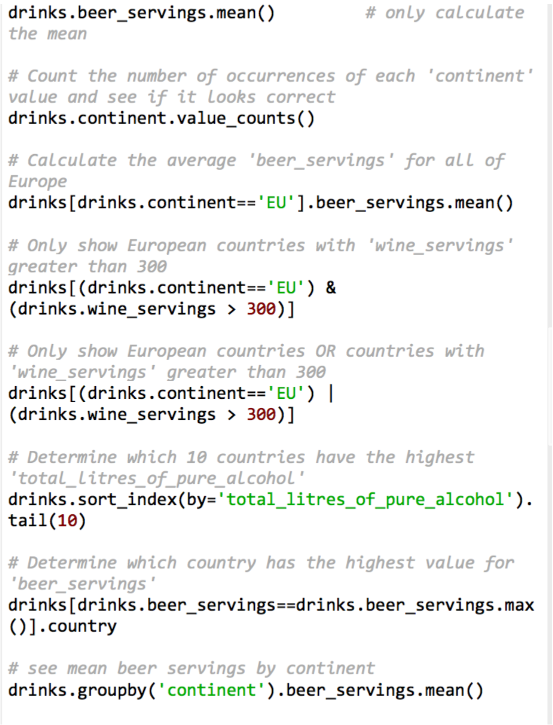


```
# show all columns of the drinks DataFrame w
# column equals 1
drinks[drinks.beer_servings == 1]

# just show the beer_sevings column where be
drinks.beer_servings[drinks.beer_servings ==

# just show the light_drinker column where b
drinks.light_drinker[drinks.beer_servings ==

# reassign all instances of the light_drinke
drinks.light_drinker[drinks.beer_servings ==

# confirm that that light_drinker now equals
drinks[drinks.light_drinker == 1]

# reassign light_drinker to 1 where beer_ser
drinks.light_drinker[drinks.beer_servings <

# re-confirm the change
drinks[drinks.light_drinker == 1]
drinks.country[drinks.light_drinker == 1]


pandas.read_csv()
pandas.read_sql()
pandas.read_excel()
pandas.read_json()

pandas.DataFrame.head()
pandas.DataFrame.tail()
pandas.DataFrame.describe()
```

```
171                           0.1        AS
1

In [104]: drinks.light_drinker[drinks.beer_servings
== 1]
Out[104]:
38     1
73     1
137    1
171    1
Name: light_drinker, dtype: int64

In [105]: drinks.country[drinks.light_drinker == 1]
Out[105]:
0           Afghanistan
1               Albania
2               Algeria
13           Bangladesh
38              Comoros
40         Cook Islands
46          North Korea
73                Haiti
79                 Iran
90               Kuwait
97                Libya
103            Maldives
106    Marshall Islands
107          Mauritania
111              Monaco
128            Pakistan
137               Qatar
147          San Marino
149        Saudi Arabia
158             Somalia
171         Timor-Leste
Name: country, dtype: object

In [106]:
```

DESCRIBING AND SUMMARIZING DATA

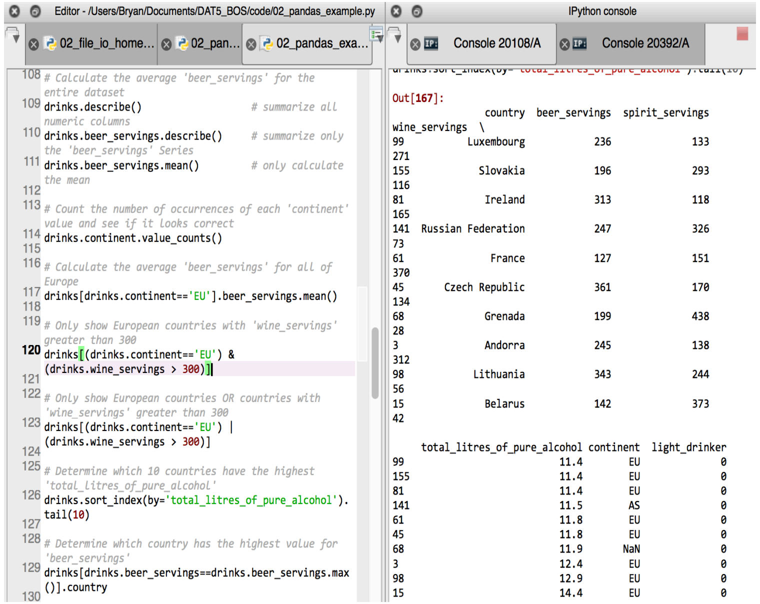
- Use .dtypes, and .info() to understand the DataFrame object itself.
- Use .describe(), .mean(), .max(), .min(), and .value_counts() to understand the data inside the DataFrame.
- Combine these methods with Boolean selections to understand relationships within certain categories or values. The & (and) and | (pipe/or) operands let you do Booleans that involve multiple columns.
- Use sort_index() to sort your DataFrame by a particular column.
- Use .groupby() to get averages, maxes, and mins by category.
- Watch out for missing values! These can be identified with .isnull(), dropped with .dropna(), and filled with .fillna().



```
drinks.beer_servings.mean()         # only calculate
the mean

# Count the number of occurrences of each 'continent'
value and see if it looks correct
drinks.continent.value_counts()

# Calculate the average 'beer_servings' for all of
Europe
drinks[drinks.continent=='EU'].beer_servings.mean()

# Only show European countries with 'wine_servings'
greater than 300
drinks[(drinks.continent=='EU') &
(drinks.wine_servings > 300)]

# Only show European countries OR countries with
'wine_servings' greater than 300
drinks[(drinks.continent=='EU') |
(drinks.wine_servings > 300)]

# Determine which 10 countries have the highest
'total_litres_of_pure_alcohol'
drinks.sort_index(by='total_litres_of_pure_alcohol').
tail(10)

# Determine which country has the highest value for
'beer_servings'
drinks[drinks.beer_servings==drinks.beer_servings.max
()].country

# see mean beer servings by continent
drinks.groupby('continent').beer_servings.mean()
```

EXERCISE: DESCRIBING AND SUMMARIZING DATA

1. Summarize all numeric columns of the ‘drinks’ dataset.
2. Get a value count of the strings in the ‘continent’ column.
3. Find the mean number of beer servings for countries within the EU.
4. Find European countries where wine_servings is greater than 300.
5. Determine which 10 countries have the highest total liters of alcohol.
6. Determine which country has the highest number of beer servings per capita.
7. See mean beer servings by continent.
8. Find missing values in the DataFrame.
9. Drop missing values.
10. Fill missing values.

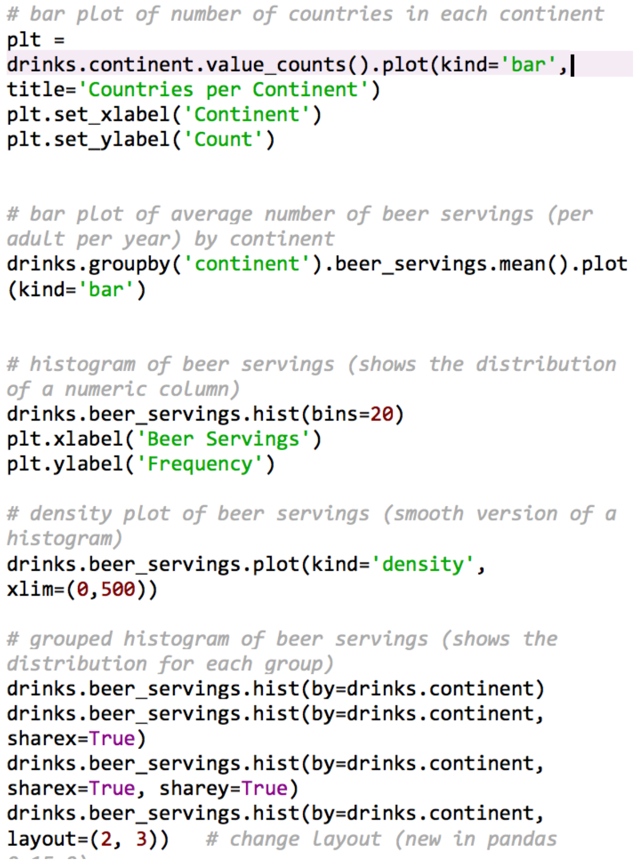


```
# Calculate the average 'beer_servings' for the
entire dataset
drinks.describe()                 # summarize all
numeric columns
drinks.beer_servings.describe()   # summarize only
the 'beer_servings' Series
drinks.beer_servings.mean()       # only calculate
the mean

# Count the number of occurrences of each 'continent'
value and see if it looks correct
drinks.continent.value_counts()

# Calculate the average 'beer_servings' for all of
Europe
drinks[drinks.continent=='EU'].beer_servings.mean()

# Only show European countries with 'wine_servings'
greater than 300
drinks[(drinks.continent=='EU') &
(drinks.wine_servings > 300)]

# Only show European countries OR countries with
'wine_servings' greater than 300
drinks[(drinks.continent=='EU') |
(drinks.wine_servings > 300)]

# Determine which 10 countries have the highest
'total_litres_of_pure_alcohol'
drinks.sort_index(by='total_litres_of_pure_alcohol').
tail(10)

# Determine which country has the highest value for
'beer_servings'
drinks[drinks.beer_servings==drinks.beer_servings.max
()].country
```

```
drinks.sort_index(by='total_litres_of_pure_alcohol').tail(10)

Out[167]:
                country  beer_servings  spirit_servings
wine_servings  \
99           Luxembourg            236              133
271
155            Slovakia            196              293
116
81              Ireland            313              118
165
141  Russian Federation            247              326
73
61               France            127              151
370
45       Czech Republic            361              170
134
68              Grenada            199              438
28
3               Andorra            245              138
312
98            Lithuania            343              244
56
15              Belarus            142              373
42

     total_litres_of_pure_alcohol continent  light_drinker
99                           11.4        EU              0
155                          11.4        EU              0
81                           11.4        EU              0
141                          11.5        AS              0
61                           11.8        EU              0
45                           11.8        EU              0
68                           11.9       NaN              0
3                            12.4        EU              0
98                           12.9        EU              0
15                           14.4        EU              0
```

PLOTTING DATA

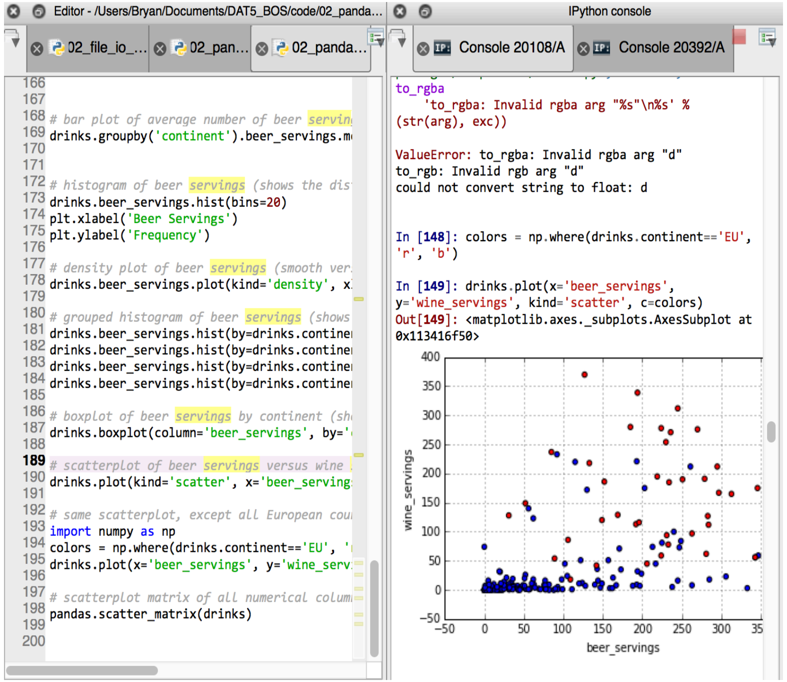
- Pandas uses a Python package called matplotlib for its plotting.
- .plot() lets you access most simple plot functions
- .plot(kind = bar) is for bar plots. Other kinds are line, barh (horizontal bar), density, area, scatter, and hexbin.
- You can add plot titles with the title= assignment, as well as .set_xlabel() and .set_ylabel()
- .hist() lets you create histograms, with by= letting you see them by a particular group
- Use .groupby() to create summary plots for a particular string or category of data (such as continent)



```
# bar plot of number of countries in each continent
plt =
drinks.continent.value_counts().plot(kind='bar',
title='Countries per Continent')
plt.set_xlabel('Continent')
plt.set_ylabel('Count')


# bar plot of average number of beer servings (per
adult per year) by continent
drinks.groupby('continent').beer_servings.mean().plot
(kind='bar')


# histogram of beer servings (shows the distribution
of a numeric column)
drinks.beer_servings.hist(bins=20)
plt.xlabel('Beer Servings')
plt.ylabel('Frequency')

# density plot of beer servings (smooth version of a
histogram)
drinks.beer_servings.plot(kind='density',
xlim=(0,500))

# grouped histogram of beer servings (shows the
distribution for each group)
drinks.beer_servings.hist(by=drinks.continent)
drinks.beer_servings.hist(by=drinks.continent,
sharex=True)
drinks.beer_servings.hist(by=drinks.continent,
sharex=True, sharey=True)
drinks.beer_servings.hist(by=drinks.continent,
layout=(2, 3))   # change layout (new in pandas
```

EXERCISE: PLOTTING DATA

1. Create a bar plot of the number of countries in each continent.
2. Create a bar plot of the average number of beer servings (per adult per year) by continent.
3. Create a histogram of beer servings by number of countries.
4. Create a density plot (a smoothed version of a histogram) of beer servings by number of countries.
5. Create grouped histograms of beer servings of countries by continent.
6. Create a box plot of of beer servings by continent.
7. Create a scatterplot of beer servings versus wine servings.
8. Create a scatterplot matrix of all numeric columns in the DataFrame.



```


# bar plot of average number of beer servin
drinks.groupby('continent').beer_servings.m


# histogram of beer servings (shows the dis
drinks.beer_servings.hist(bins=20)
plt.xlabel('Beer Servings')
plt.ylabel('Frequency')

# density plot of beer servings (smooth ver
drinks.beer_servings.plot(kind='density', x

# grouped histogram of beer servings (shows
drinks.beer_servings.hist(by=drinks.contine
drinks.beer_servings.hist(by=drinks.contine
drinks.beer_servings.hist(by=drinks.contine
drinks.beer_servings.hist(by=drinks.contine

# boxplot of beer servings by continent (sh
drinks.boxplot(column='beer_servings', by='

# scatterplot of beer servings versus wine
drinks.plot(kind='scatter', x='beer_serving

# same scatterplot, except all European cou
import numpy as np
colors = np.where(drinks.continent=='EU', '
drinks.plot(x='beer_servings', y='wine_serv

# scatterplot matrix of all numerical colum
pandas.scatter_matrix(drinks)

```

```
to_rgba
    'to_rgba: Invalid rgba arg "%s"\n%s' %
(str(arg), exc))

ValueError: to_rgba: Invalid rgba arg "d"
to_rgb: Invalid rgb arg "d"
could not convert string to float: d


In [148]: colors = np.where(drinks.continent=='EU',
'r', 'b')

In [149]: drinks.plot(x='beer_servings',
y='wine_servings', kind='scatter', c=colors)
Out[149]: <matplotlib.axes._subplots.AxesSubplot at
0x113416f50>
```

HOMEWORK

- Open http://nbviewer.ipython.org/github/cs109/content/blob/master/lec_04_wrangling.ipynb.
- Read through the entire iPython Notebook.
- As you get to each code block, **copy it into your own Python script** and run the code yourself. Try to understand exactly how each line works. You will run into Python functions that you haven't seen before!
- Explore the data on your own using Pandas. At the bottom of your script, write out (as comments) **two interesting facts** that you learned about the data, and show the code you used to find those facts.
- Create **two new plots** that show something interesting about the data, and save those plots as files. Include the plotting code at the bottom of your script.
- Save your **Python script and image files** to your local hard drive. We will upload them to the class Git on Tuesday.

QUESTIONS?