



Python Quick Camp

TDWI Accelerate - October 2017

Paul Boal - @paulboal - @AmitechSolution

Purpose

- Understand the latest trends in Python as a data science tool
- Use Python to load, enrich, and analyze real-world data
 - Excel, csv, web services, HTML
- Manage data using high-performance data structures
 - pandas, numpy
- Visualize data and relationships using plots
 - matplotlib, pandas, seaborn, plotly
- Organize complex code libraries using Python modules

About Me

➤ Paul Boal
@paulboal

➤ VP Delivery
<http://amitechsolutions.com>
@AmitechSolution

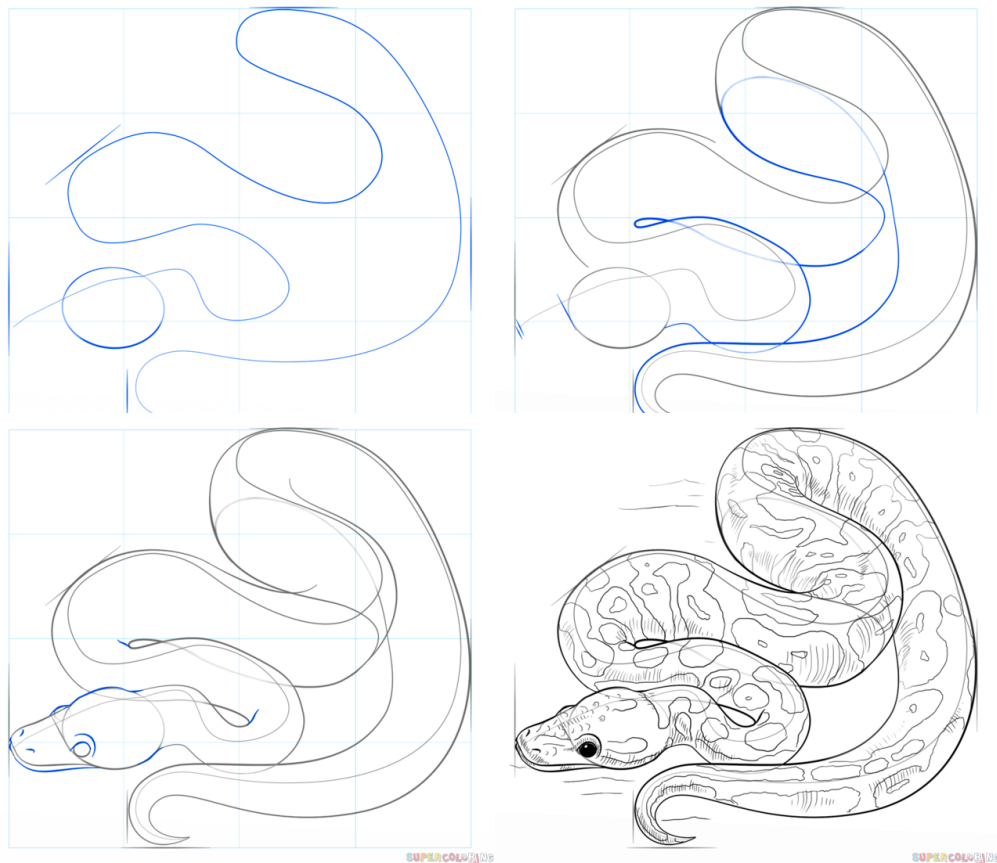
- Healthcare data and analytics solutions
- Big data, IoT, and advanced analytics
- Data strategy and data governance
- Drive change through data insights

• Husband and father of three



AMITECH

Agenda

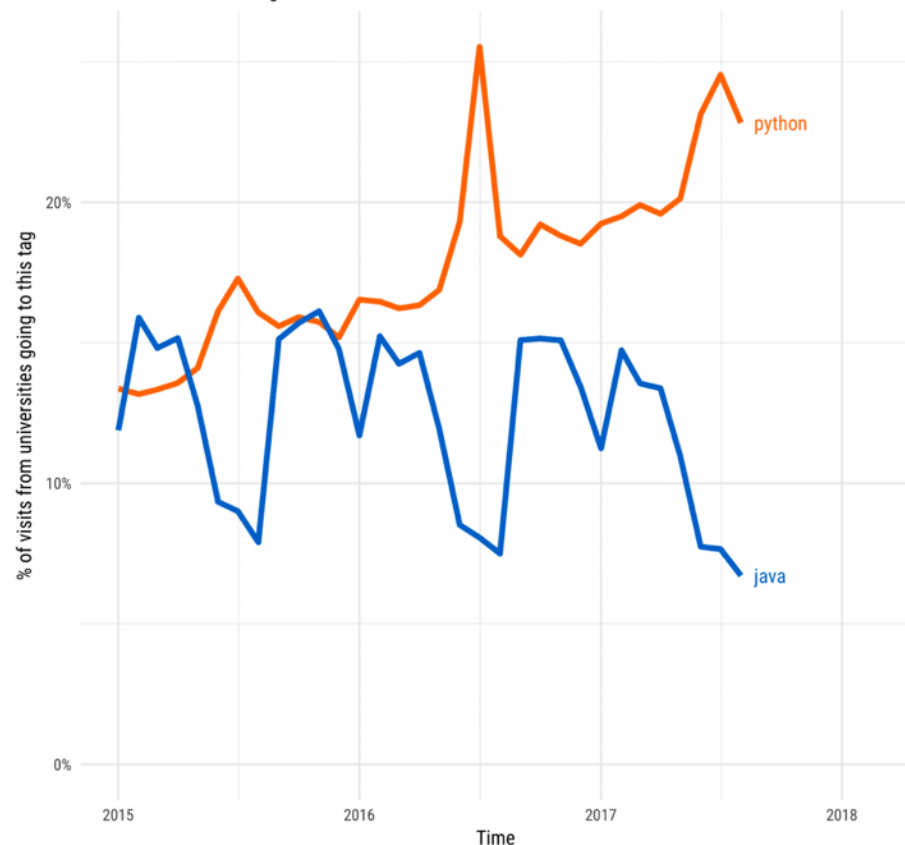


- What's going on with Python?
- Python Jupyter 101
- Real-world scenario
 - Getting data
 - Formatting data
 - Enriching data
 - Data profiling and analysis
- Exercises
- Q&A

Universities are Teaching Python

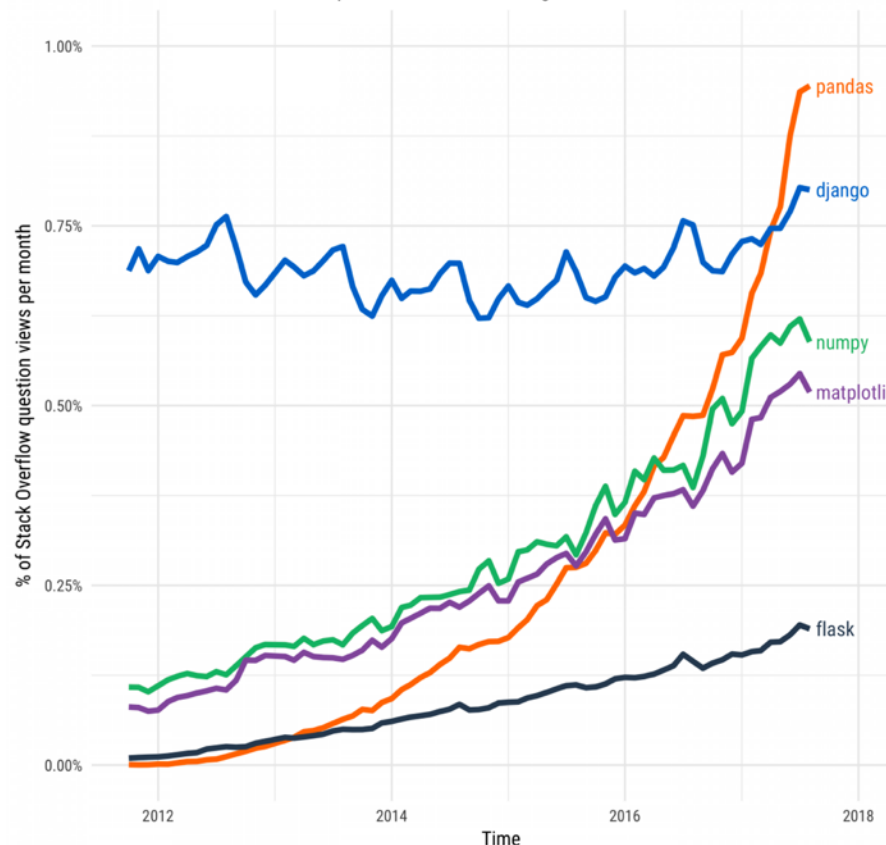
Seasonal Traffic from Colleges and Universities

Based on traffic from colleges and universities in the US and UK.



Stack Overflow Traffic to Questions About Selected Python Packages

Based on visits to Stack Overflow questions from World Bank high-income countries

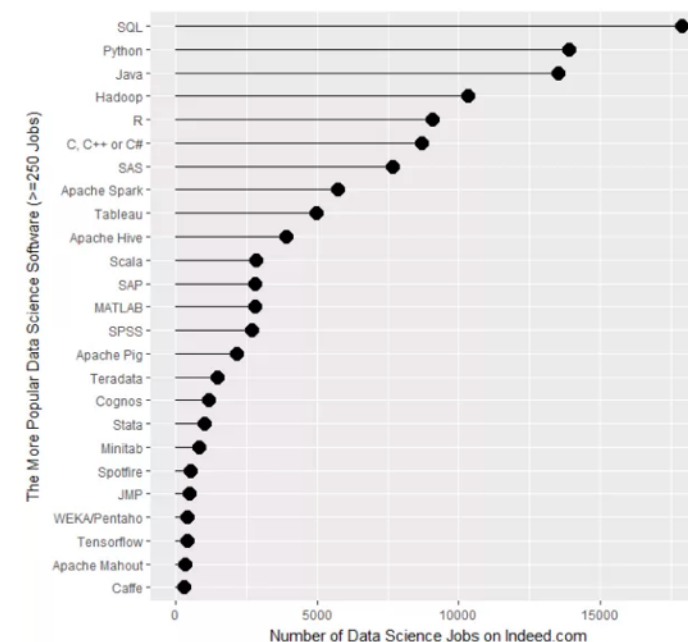


- Fastest growing Q&A topic, especially in colleges and universities.
- Growth driven almost entirely from pandas, numpy, matplotlib

<https://stackoverflow.blog/2017/09/14/python-growing-quickly/>

Python is on the rise

- Growing faster than R as a requirement for data science related jobs.
- Second only to SQL in technical skills required by a data scientist.



<https://www.r-bloggers.com/data-science-job-report-2017-r-passes-sas-but-python-leaves-them-both-behind/>

Why choose Python for Data Science?

- A multi-purpose language
- Easy to learn syntax
- Interactive and reproducible plots
- Large and growing data and analytics libraries
- Integration with big data platforms (e.g. Spark)
- Build complete applications in a single language
- Multiple ways to interact with Python: command line, IDE, notebook

Starting the Jupyter Notebook

On Windows:

1. Launch `Anaconda Prompt` from the Start menu
2. `cd <where you downloaded the repository>`
for example: `cd C:\Users\myname\Documents\tdwi-accelerate-2017-python`
3. `jupyter notebook`

If you don't have Python up and running locally:

<http://TDWI.AmitechSolutions.com>

First Notebook

- Create a new Python Notebook in Jupyter

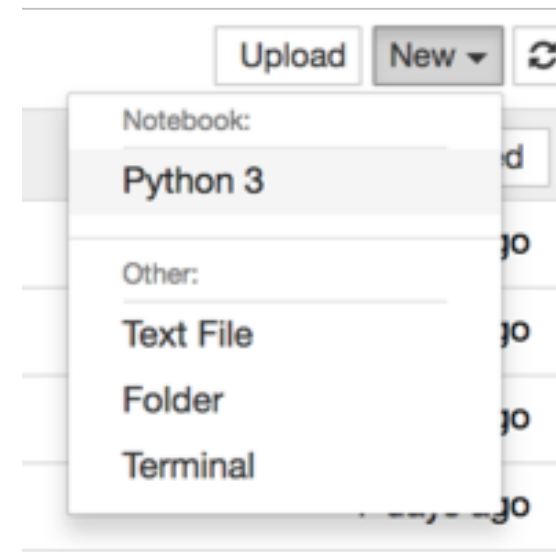
- Write your hello world program

```
In [1]: print("Hello World!")
```

- Run the current cell with shift-Enter

```
In [1]: print("Hello World!")
```

```
Hello World!
```

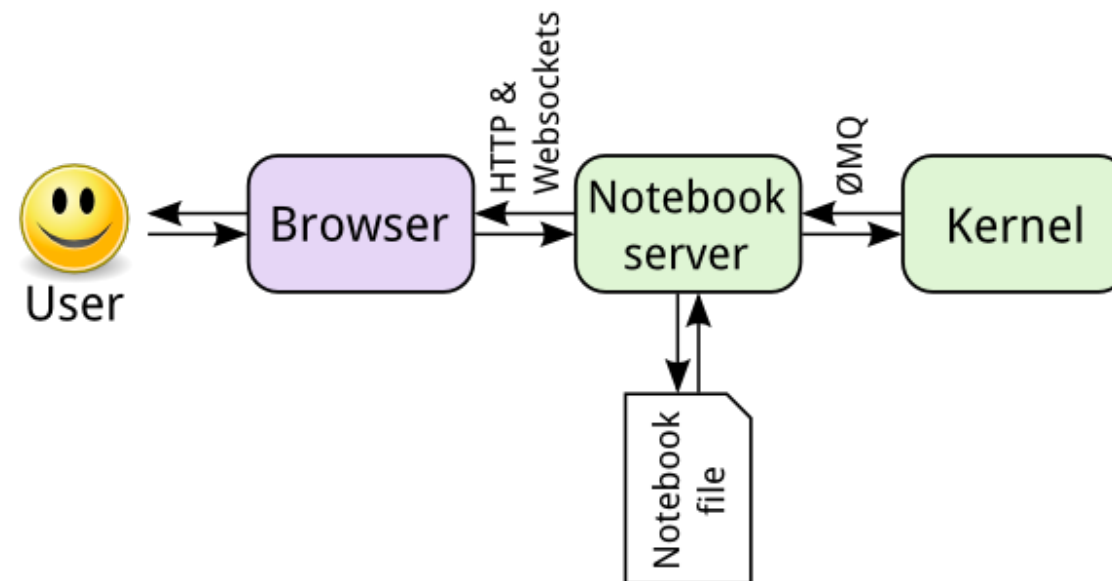


Hello Jupyter!

- **Comments** are preceded by #.
- **Variables** get assigned values with =.
- **Code blocks** require consistent indenting.
- **Function calls** use () to pass parameter values.
- Jupyter organizes commands into cells that can hold code or documentation, and have results.

```
In [1]: # This is my first Python program
planet = "Jupyter"
if planet == "Earth":
    print("Hello World!")
else:
    print("Hello Jupyter!")

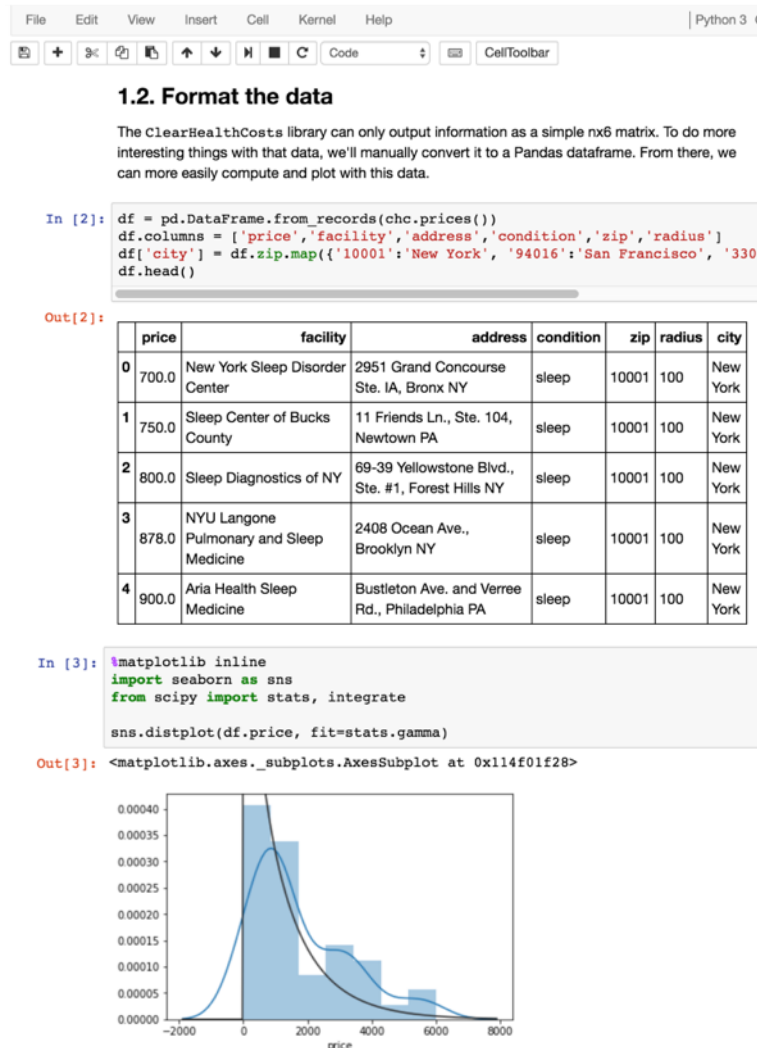
Hello Jupyter!
```



- Jupyter is an interactive web-based programming notebook that shows commands and results in-line.
- Jupyter launches separate Python interpreters for each running notebook.
- Notebooks are independent and do not share variables or data back-and-forth.

Introducing the Jupyter Notebook

- Jupyter tool bars
- Cell types
 - Documentation
 - Code
- Output
 - Print statements
 - Variable contents
 - Plots



NOTE: Jupyter saves files with a .ipynb extension. These are structured JSON and contain all of the notebook contents including documentation, code, and visible results. The notebook file does not contain a snapshot of the variable contents – that only exists in the current Python instance.

Real-World Scenario

Introduction

Business Scenario:

- Healthcare costs vary dramatically from one facility to the next, even in the same community.
- Are there any geographic features that might be tied to those cost patterns?

Steps:

1. Getting procedure costs
2. Get facility information
3. Get additional external data
4. Review correlations in data

Retrieve Git repository with code and data from

<https://github.com/paulboal/tdwi-accelerate-2017-python>

1

Get Procedure Cost

Procedure Costs

SEARCH RESULTS: SLEEP

13 price reports

Check out our prices, then [share what you paid](#). [How did we do this?](#)

REFINE RESULTS

Want to find results near to your location? Enter your zip code and click "Refine" button.

94016

100 mi

☒ Don't show \$0 results

REFINE

RELATED POSTS

What Medicare pays here?

PRICE REPORTS

95810 REGULAR SLEEP STUDY OR POLYSOMNOGRAPHY

PriceCheck journalist

Price charged

\$1,000

Insurance paid

\$0

You paid

\$0

Includes facility and doctor fees.

Redwood Sleep Centers Inc.

415-898-0801

1615 Hill Road, Ste. 16, Novato CA

Open map

PriceCheck journalist

Price charged

\$1,050

Insurance paid

\$0

You paid

\$0

Includes facility, interpretation and doctor fees.

Peninsula Sleep Center

650-636-9396

1828 El Camino Real, #707, Burlingame CA

Open map

- Clear Health Costs is a web site that lets users search for procedure costs around particular ZIP codes in several large cities.
- Our first step is to run searches against this website and collect the data.
- *No manual copy/paste of data!*

Using Modules

```
from GetPrices import ClearHealthCosts
import pandas as pd

chc = ClearHealthCosts()
chc.get_sleep_prices('10001',100) # New York
chc.get_sleep_prices('94016',100) # San Francisco
chc.get_sleep_prices('33018',100) # Miami
chc.get_sleep_prices('75001',100) # Dallas
```

TDWI_Python_QuickCamp_2017.ipynb

- In the notebook, all you see are some imports and calls to `get_sleep_prices()`.
- All of the code for how to retrieve data from Clear Health Costs is encapsulated in `GetPrices.py`.
- The key to using this personal module is `from GetPrices import ClearHealthCosts`.
- This Jupyter cell also imports pandas, but it isn't used until later.

GetPrices Module: Parse HTML with BeautifulSoup

```
from bs4 import BeautifulSoup
import requests

class ClearHealthCosts:

    ...
    def get_prices(self, condition, zip, radius):
        url = self._get_base_url(condition, zip, radius)
        result = requests.get(url)

        if result.status_code == 200:
            prices = self._parse_pricelist(result.content)
            for i in range(0, len(prices)):
                prices[i] += [condition, zip, radius]

            self._prices += prices

        return self

    ...
```

GetPrices.py

- More dependencies being imported within the GetPrices module:
 - BeautifulSoup – parsing HTML
 - requests – calling web services or pages
- Using a class structure to encapsulate variables rather than relying on global variables.
- Within a class definition, internal variables and functions are prefixed by `self`.

Getting Prices: Thinking in Objects

- Procedural Programming:

- Functions are managed and run globally
- Variables and data are either local to a function or global

Step 1

Step 1.1()

Step 1.2()

Step 2

Step 2.1()

Step 2.2()

Step 2.3()

Variable A

Variable B

Variable C

- Object-Oriented Programming

- Code execution is still linear
- Classes hide complex data structures and related functions in one variable

Step 1

Step 2

Object A

Step 1.1()

Step 1.2()

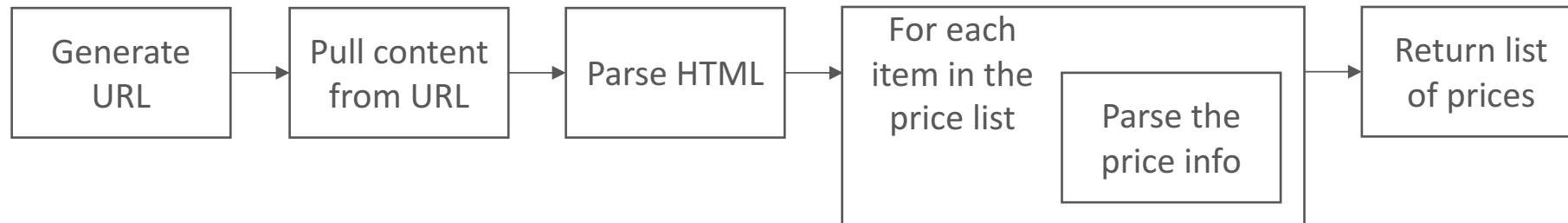
Object B

Step 2.1()

Step 2.2()

Step 2.3()

Getting Prices: Steps



Getting Prices: get_sleep_prices()

```
def get_sleep_prices(self, zip, radius=100):  
    return self.get_prices('sleep', zip, radius)
```

GetPrices.py

- **def** defines a new function or method
- All class methods include **self** as the first parameter
- Parameters can have default values, making them optional
- This function simply calls another class method
- Key goal of object-oriented programming: testability, reusability, flexibility

Getting Prices: get_prices()

```
def get_prices(self, condition, zip, radius):  
    url = self._get_base_url(condition, zip, radius)  
    result = requests.get(url)  
  
    if result.status_code == 200:  
        prices = self._parse_pricelist(result.content)  
        for i in range(0, len(prices)):  
            prices[i] += [condition, zip, radius]  
  
        self._prices += prices  
  
    return self
```

```
def _get_base_url(self, query, zip_code, radius):  
    return self._base_url+'query='+str(query)+  
        '&zip_code='+str(zip_code)+  
        '&radius='+str(radius)+'&submit='
```

GetPrices.py

- Call another function to create the URL by concatenating all the pieces together
- Use the **requests** module to call that URL and get the HTML back.
- If you get an OK result, then parse the price list and return a list that has the prices and the input parameters.
- Strings can be concatenated just by “adding” them together.

Loops and Lists

```
alpha = ['a', 'b', 'c', 'd', 'e']
```

```
alpha[0]
```

```
'a'
```

```
alpha[2:5]
```

```
['c', 'd', 'e']
```

```
for i in range(len(alpha)):
    print("{} : {}".format(i, alpha[i]))
```

```
0 : a
```

```
1 : b
```

```
2 : c
```

```
3 : d
```

```
4 : e
```

- A list is just a series of values. They don't have to be the same data type.
- To create a list, enclose the list of elements with brackets.
- Get a slice of a list using indexes inside of brackets.
- Loop through a list by using the **range** sequence generator.
- Also note how the **print** and **format** functions work together with **{}** as placeholders.

Getting Prices: parse_pricelist()

```
def _parse_pricelist(self, page):  
    prices = []  
    soup = BeautifulSoup(page, 'html.parser')  
    items = soup.findAll('div', 'ui grid segment')  
  
    for item in items:  
        price = self._parse_price(item)  
        prices.append(price)  
  
    return prices
```

GetPrices.py

- Create an empty list for storing results.
- BeautifulSoup's `findAll` function returns a list of all HTML elements of a given type (`div`) with certain classes (`ui grid segment`).
- We can also just loop over the elements in a list.
- The `append` just adds the next set of information to the end of the `prices` list.

Working with Lists

1. Create a list with your first, middle, and last name
2. Write a loop that prints each part of your name
3. Write a function that prints your name in Last, First Middle

Working with Lists

1. Create a list with your first, middle, and last name

```
name = ['Paul', 'Emre', 'Boal']
```

2. Write a loop that prints each part of your name

```
for part in name:  
    print(part)
```

3. Write a function that prints your name in Last, First Middle

```
def format_name(n):  
    print("{} {}, {}".format(n[2], n[0], n[1]))  
  
format_name(name)
```

Processing Data: Introducing the DataFrame

```
→ df = pd.DataFrame.from_records(chc.prices())
→ df.columns = ['price', 'facility', 'address', 'condition', 'zip', 'radius']
→ df.head()
```

TDWI_Python_QuickCamp_2017.ipynb

- Convert the list of values coming back from Clear Health Costs into a Pandas DataFrame
- Add labels for the columns
- Take a peak at the top 5 or 10 rows of data

700.0	New York Sleep Disorder Center	2951 Grand Concourse Ste. IA, Bronx NY	sleep	10001	100	New York
750.0	Sleep Center of Bucks County	11 Friends Ln., Ste. 104, Newtown PA	sleep	10001	100	New York
800.0	Sleep Diagnostics of NY	69-39 Yellowstone Blvd., Ste. #1, Forest Hills NY	sleep	10001	100	New York
878.0	NYU Langone Pulmonary and Sleep Medicine	2408 Ocean Ave., Brooklyn NY	sleep	10001	100	New York
900.0	Aria Health Sleep Medicine	Bustleton Ave. and Verree Rd., Philadelphia PA	sleep	10001	100	New York

`chc.prices()` returns a list of lists with no metadata

What is a DataFrame?

Values at the index of 1

The 'address' series

	price	facility	address	condition	zip	radius
0	700.0	New York Sleep Disorder Center	2951 Grand Concourse Ste. 1A, Bronx NY	sleep	10001	100
1	750.0	Sleep Center of Bucks County	11 Friends Ln., Ste. 104, Newtown PA	sleep	10001	100
2	800.0	Sleep Diagnostics of NY	69-39 Yellowstone Blvd., Ste. #1, Forest Hills NY	sleep	10001	100
3	878.0	NYU Langone Pulmonary and Sleep Medicine	2408 Ocean Ave., Brooklyn NY	sleep	10001	100
4	900.0	Aria Health Sleep Medicine	Bustleton Ave. and Verree Rd., Philadelphia PA	sleep	10001	100

- Pandas **DataFrames** can be thought of as a spreadsheet or pivot table in which every column and every row have a label.
 - Column labels are referred to as the **Series** and are usually meaningful strings
 - Row labels are referred to as the **Index** and can be numeric or strings
 - Column and row labels can be lists, creating a hierarchical structure

Creating and accessing a DataFrame

```
import pandas as pd
```

```
df = pd.DataFrame()  
df['first'] = ['Paul', 'Adelaide']  
df['middle'] = ['Emre', 'Lynn']  
df['last'] = ['Boal', 'Lester']  
df
```

	first	middle	last
0	Paul	Emre	Boal
1	Adelaide	Lynn	Lester

```
df.loc[0]
```

```
First      Paul  
middle     Emre  
last       Boal  
Name: 0, dtype: object
```

- Pandas implements **Series** and **DataFrames** to add lots of efficiency and functionality on top of basic lists and dictionaries.
- Think of a **DataFrame** as a spreadsheet and a **Series** as one column.
- Indexing a **DataFrame** using a column name returns all the values in that column allowing you to do convenient set operations.
- **DataFrames** are compatible with NumPy arrays, so calculations can be run in parallel.
- Indexing a **DataFrame** using the **loc()** function returns one row, but row-by-row access is usually less efficient.

Set operations on a DataFrame

```
df['last'] + ', ' + df['first']
```

```
0      Boal, Paul
1    Lester, Adelaide
```

```
df['full'] = df['last'] + ', ' + df['first']
df
```

	first	middle	last	full
0	Paul	Emre	Boal	Boal, Paul
1	Adelaide	Lynn	Lester	Lester, Adelaide

- Doing set operations with DataFrames should feel natural to anyone who has worked with SQL before.
- It might be awkward for developers coming from an application background.
- If you come from an object-oriented application background, just look at this like operating on a collection of objects all at once.

Working with DataFrames

1. Start with the 2-dimensional list below
2. Convert this to a DataFrame
3. Label the columns “street”, “city”, “state”
4. Add a new series called “label” that is formatted as you would expect

```
addresses = [  
    ['1 CityPace Drive', 'Creve Coeur', 'MO'],  
    ['350 Fifth Avenue', 'New York', 'NY'],  
    ['221 B Baker St', 'London', 'England']  
]
```

Working with DataFrames

```
addresses = [
    ['1 CityPace Drive', 'Creve Coeur', 'MO'],
    ['350 Fifth Avenue', 'New York', 'NY'],
    ['221 B Baker St', 'London', 'England']
]

df = pd.DataFrame(addresses)
df.columns = ['street', 'city', 'state']
df['label'] = df['street'] + ', ' + df['city'] + ', ' + df['state']
df
```

	street	city	state	label
0	1 CityPace Drive	Creve Coeur	MO	1 CityPace Drive, Creve Coeur, MO
1	350 Fifth Avenue	New York	NY	350 Fifth Avenue, New York, NY
2	221 B Baker St	London	England	221 B Baker St, London, England

Making Data Readable: Code to Descriptions Maps

```
df['city'] = df['zip'].map({'10001': 'New York',  
                           '94016': 'San Francisco',  
                           '33018': 'Miami',  
                           '75001': 'Dallas'})  
  
df.head()
```

TDWI_Python_QuickCamp_2017.ipynb

- Use a dictionary to represent the relationship between ZIP code and metropolitan area.
- Map the existing ZIP code field into a city name and store that.

	price	facility	address	condition	zip	radius	city
0	700.0	New York Sleep Disorder Center	2951 Grand Concourse Ste. 1A, Bronx NY	sleep	10001	100	New York
1	750.0	Sleep Center of Bucks County	11 Friends Ln., Ste. 104, Newtown PA	sleep	10001	100	New York
2	800.0	Sleep Diagnostics of NY	69-39 Yellowstone Blvd., Ste. #1, Forest Hills NY	sleep	10001	100	New York
3	878.0	NYU Langone Pulmonary and Sleep Medicine	2408 Ocean Ave., Brooklyn NY	sleep	10001	100	New York
4	900.0	Aria Health Sleep Medicine	Bustleton Ave. and Verree Rd., Philadelphia PA	sleep	10001	100	New York

Dictionaries

```
dictionary = {  
    name1 : value1,  
    name2 : value2...  
}
```

```
me = {  
    'last' : 'Boal',  
    'middle' : 'Emre',  
    'first' : 'Paul'  
}
```

```
me['last']
```

```
'Boal'
```

- A dictionary is a map between names and values, where the values can be any complex data type including another dictionary, a list, or a scalar value.
- Dictionaries can be accessed by using the bracket similar similar to list indexing.
- Dictionaries can easily be used for looking up information against a code or value list that maps to something definition.

Working with DataFrames

1. Start with the 2-dimensional list below
2. Convert this to a DataFrame
3. Label the columns “street”, “city”, “state”
4. Add a new series called “label” that is formatted as you would expect

```
addresses = [  
    ['1 CityPace Drive', 'Creve Coeur', 'MO'],  
    ['350 Fifth Avenue', 'New York', 'NY'],  
    ['221 B Baker St', 'London', 'England']  
]
```

Working with DataFrames

```
addresses = [
    ['1 CityPace Drive', 'Creve Coeur', 'MO'],
    ['350 Fifth Avenue', 'New York', 'NY'],
    ['221 B Baker St', 'London', 'England']
]

df = pd.DataFrame(addresses)
df.columns = ['street', 'city', 'state']
df['label'] = df['street'] + ', ' + df['city'] + ', ' + df['state']
df
```

	street	city	state	label
0	1 CityPace Drive	Creve Coeur	MO	1 CityPace Drive, Creve Coeur, MO
1	350 Fifth Avenue	New York	NY	350 Fifth Avenue, New York, NY
2	221 B Baker St	London	England	221 B Baker St, London, England

Price Distribution

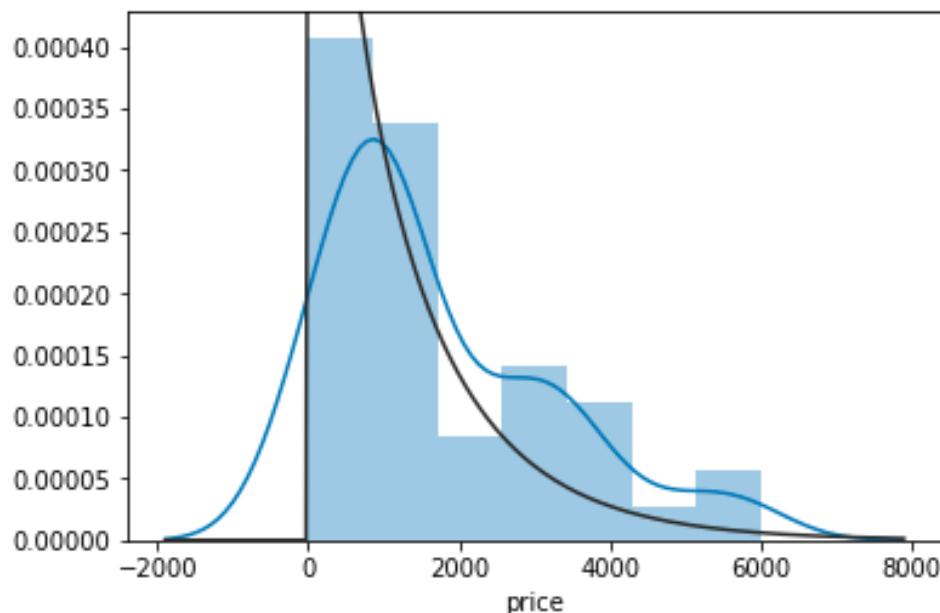
	price	facility	address	condition	zip	radius	city
0	700.0	New York Sleep Disorder Center	2951 Grand Concourse Ste. 1A, Bronx NY	sleep	10001	100	New York
1	750.0	Sleep Center of Bucks County	11 Friends Ln., Ste. 104, Newtown PA	sleep	10001	100	New York
2	800.0	Sleep Diagnostics of NY	69-39 Yellowstone Blvd., Ste. #1, Forest Hills NY	sleep	10001	100	New York
3	878.0	NYU Langone Pulmonary and Sleep Medicine	2408 Ocean Ave., Brooklyn NY	sleep	10001	100	New York
4	900.0	Aria Health Sleep Medicine	Bustleton Ave. and Verree Rd., Philadelphia PA	sleep	10001	100	New York

- What does the distribution of prices look like?
 - Descriptive statistics, histogram, distribution plot
- How do prices vary with respect to the city they are from?
 - Distribution by city, correlation analysis

Histograms

```
%matplotlib inline  
import seaborn as sns  
from scipy import stats, integrate  
  
sns.distplot(df.price, fit=stats.gamma)
```

TDWI_Python_QuickCamp_2017.ipynb

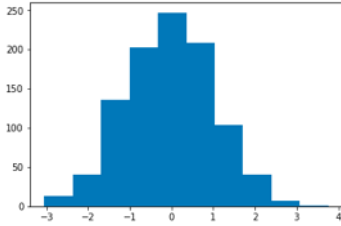
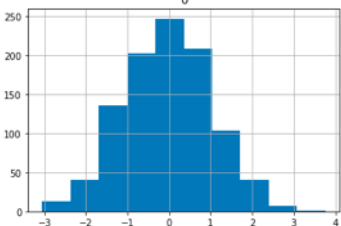
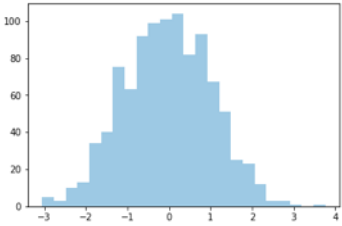
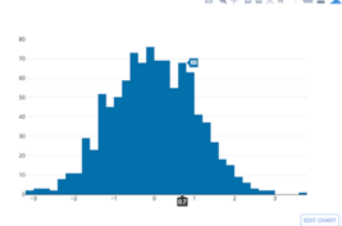
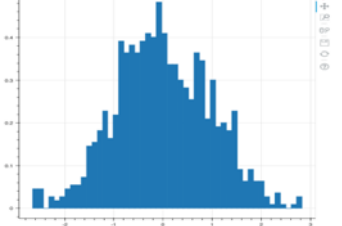


- Tell Jupyter that we want charts to appear inline in the notebook
- Import the seaborn plotting library
- Create a distribution plot including alternative distribution fits (e.g. [gamma](#))
- One observation we can make is that the distribution is weighted toward the low-end of prices, but there are bubbles at higher levels.
- What causes those?

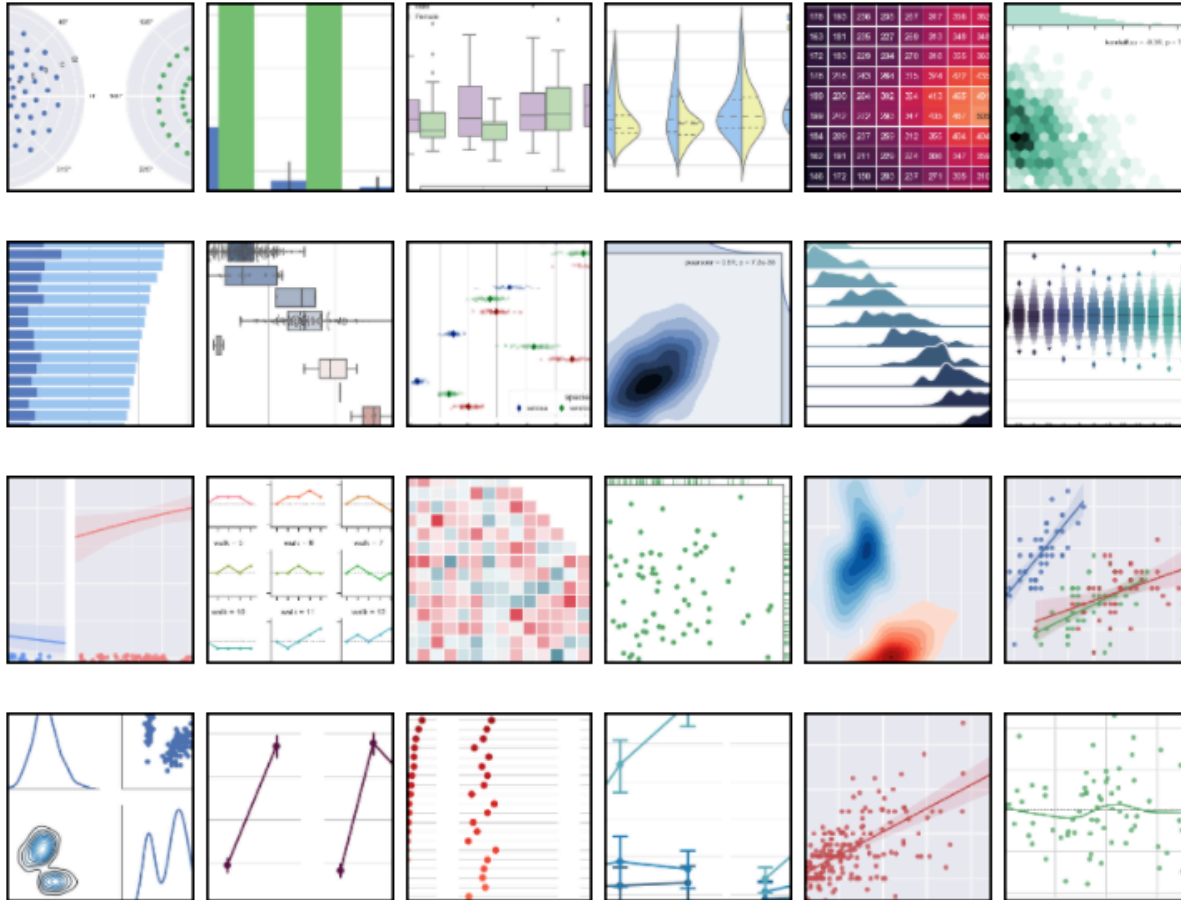
Plotting Libraries

Package	Uses
matplotlib	Basic plotting capabilities following the paradigms from Matlab designed for scientific publications. Not built with web based notebooks in mind first, but interactive charts are available.
pandas	DataFrames have built-in plotting functions that use matplotlib under the hood.
seaborn	This library makes the default matplotlib plots prettier and adds many functions to make common plotting activities easier.
plotly	Plot.ly is much more than just a plotting library. It's a comprehensive web service that gives analysts a way to manage and share data, code, and visualizations. Basic and public usage is free, but enterprise services are paid.
bokeh	An Open Source plotting library that makes interactive web graphics easier to create within environments like Jupyter.

Feature Comparison

	matplotlib	pandas	seaborn	plotly	bokeh
Histogram Code	<code>plt.hist(data)</code>	<code>df.hist()</code>	<code>sns.distplot(df, kde=False)</code>	<code>py.iplot([go.Histogram(x=data)], filename='inline')</code>	<code>p = figure() hist, edges = np.histogram(data, density=True, bins=50) p.quad(top=hist, bottom=0) show(p)</code>
Histogram Output					
Easy simple plots	✓	✓	✓	✗	✗
Easy statistical plots	✗	✓	✓	✓	✗
Interactivity	✗	✗	✗	✓	✓
Open source	✓	✓	✓	✗	✓

Seaborn Gallery



- Distribution of Dataset
- Categorical Data
- Linear Relationships
- Pairwise Relationships

<http://seaborn.pydata.org/tutorial.html>

Histogram Options

Parameter	Description
<code>a</code>	Data to plot: Series, 1d-array, or list
<code>hist</code>	True /False: Include the histogram in the output
<code>kde</code>	True /False: Include the gaussian kernel density estimate
<code>rug</code>	True/ False : Include a rugplot on the support axis
<code>color</code>	Change the color, by name
<code>axlabel</code>	Label to put on the support axis

Seaborn Histograms

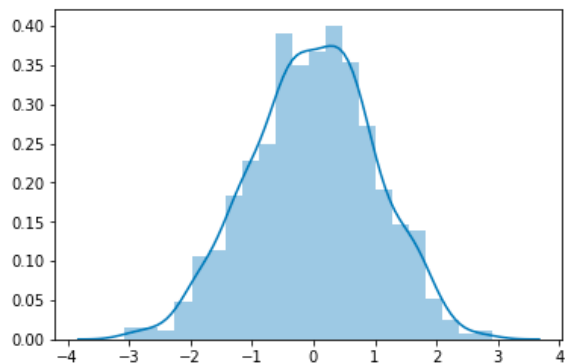
1. Start with the dataset below
2. Use seaborn to plot a histogram
3. Change the color of the plot
4. Add and remove the histogram, kde, and rug

```
data = np.random.normal(size=1000)
```

Seaborn Histograms

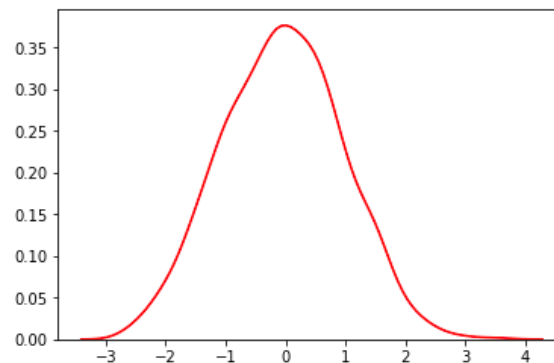
```
%matplotlib inline
import seaborn as sns
import numpy as np

data = np.random.normal(size=1000)
sns.distplot(data)
```



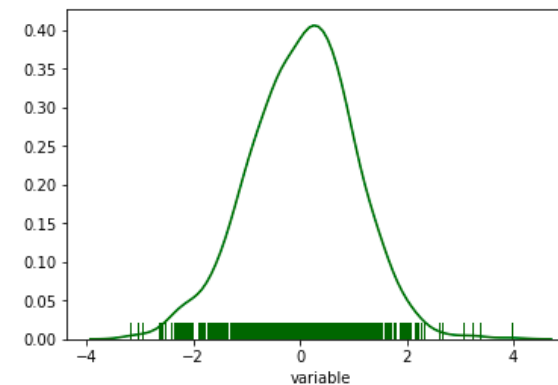
```
%matplotlib inline
import seaborn as sns
import numpy as np

data = np.random.normal(size=1000)
sns.distplot(data,
              color="Red",
              hist=False)
```

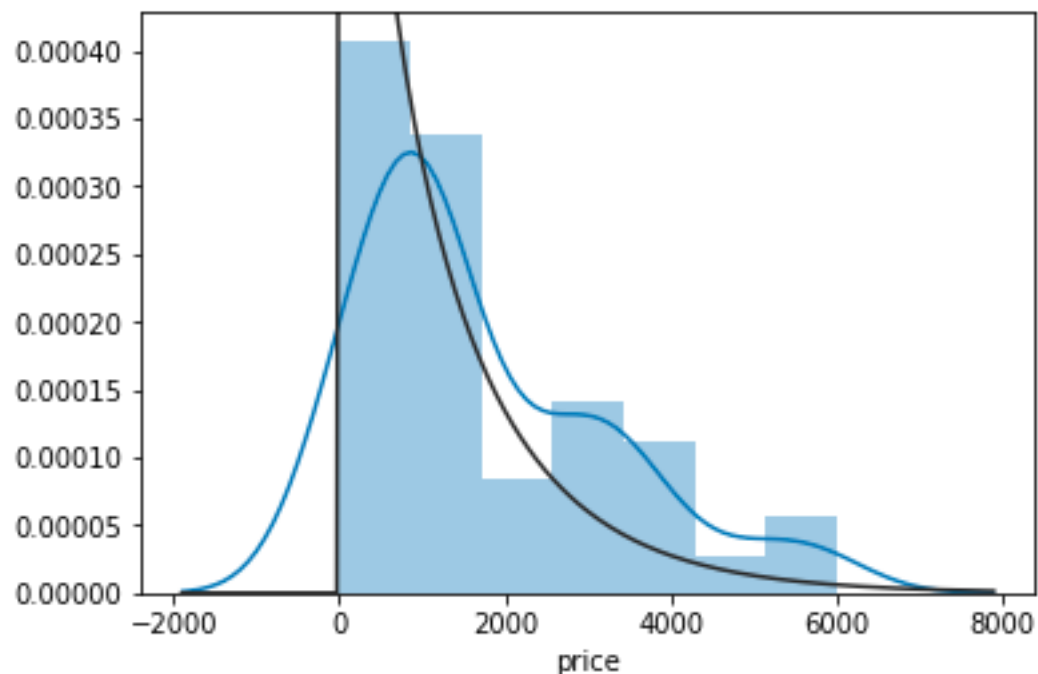


```
%matplotlib inline
import seaborn as sns
import numpy as np

data = np.random.normal(size=1000)
sns.distplot(data,
              color="DarkGreen",
              hist=False,
              rug=True,
              axlabel="variable")
```



Cleaning Bad Data: Filtering a DataFrame



- Not only are there a predominance of low values in this distribution, but there might be a lot of \$0 results.
- These don't help and they skew the results.
- Calculating how many there are and then filtering them out will be helpful.

Cleaning Bad Data: Filtering a DataFrame

```
total = len(df)
zeros = sum(df.price == 0)
print("{} out of {} have a price of $0".format(zeros, total))
```

```
12 out of 83 have a price of $0
```

- Get the total number of entries
- Count how many have a price of 0
- Print out the results

```
df = df[df.price != 0]
len(df)
```

```
71
```

- Filter the results by selecting only those that have a price not equal to 0.

Testing Values and Filtering a DataFrame

```
import pandas as pd
df = pd.DataFrame()
df['val'] = range(10)
df['val'] % 2 == 0
```

0	True
1	False
2	True
3	False
4	True
5	False
6	True
7	False
8	True
9	False

```
import pandas as pd
df = pd.DataFrame()
df['val'] = range(10)
selector = df['val'] % 2 == 0
df[selector]
```

	val
0	0
2	2
4	4
6	6
8	8

```
df['val'] % 2 == 0
```

- Tests all the contents of the val series to see if their modulus 2 equals 0.
- The result is a series of True / False values.
- Note that True/False can be summed to get a count of how many are True.

```
df[selector]
```

- The [selector] syntax selects the subset of values from the series where the selector value is True.

Filtering a DataFrame

1. Starting with the two lists below
2. Create a data frame with those series
3. Count how many people in the data frame are female
4. Display only the male people

```
people = ['Donna Wong', 'Edward Tufte', 'David Candless']  
genders = ['female', 'male', 'male']
```

Filtering a DataFrame

```
import pandas as pd
people = ['Donna Wong', 'Edward Tufte', 'David Candless']
genders = ['female', 'male', 'male']
df = pd.DataFrame({'person': people, 'gender': genders})
```

```
sum(df['gender'] == 'female')
```

1

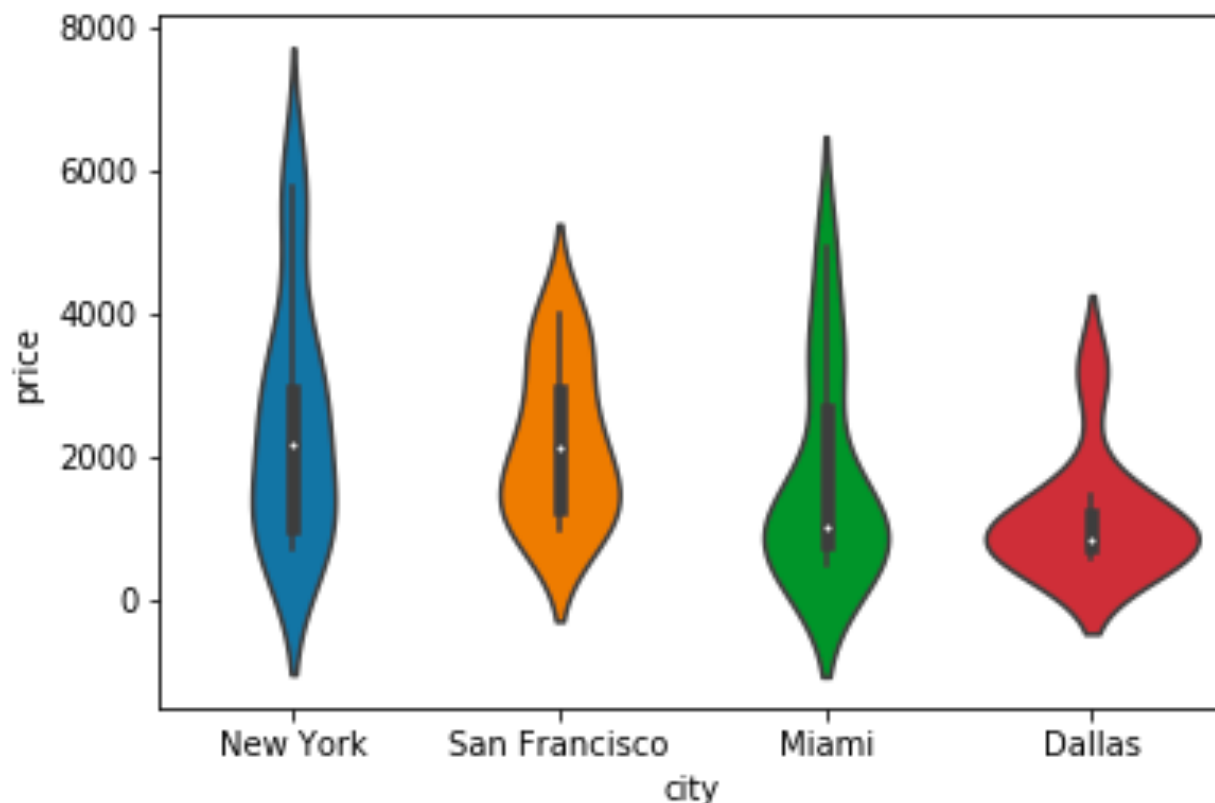
```
df[df['gender'] == 'male']
```

	gender	person
1	male	Edward Tufte
2	male	David Candless

Drill Down: Histograms by Series

```
sns.violinplot(data=df, y='price', x='city')
```

TDWI_Python_QuickCamp_2017.ipynb



- A violin plot makes a nice way to drill down into distribution by some categorical series.
- Does the distribution of values vary based on the category value?
- Seems clear that the price distribution in Dallas is different than New York.

2

Get Facility Information

Getting more information about these facilities

- Fairfield County Sleep Center
 - Sleep Disorders Institute
 - Mercy Fitzgerald Hospital Sleep Disorders Center
 - Sleep Medicine Associates of NYC
 - The Sleep Center at Greenwich Hospital
 - Lankenau Medical Center
 - Bryn Mawr Hospital
 - Paoli Hospital for Sleep Medicine
- Sometimes you can infer information about something from other information you already have. For example, these facility names may hint at if they are a free-standing clinic, hospital-based, university-affiliated, or physician groups.
 - We may be able to use natural language processing (NLP) to cluster the facility names so that we don't have to manually categorize each name.

Natural Language Processing in Python

- NLTK is the standard Python NLP toolkit
 - Tokenize – split documents into paragraphs, sentences, and words
 - Stemming – based on base words
 - Type of speech – based on vocabulary and usage context
 - Standard corpus – knowledge base for understanding language in context
- Often used in conjunction with machine learning for:
 - Supervised document classification
 - Unsupervised document clustering
 - Sentiment analysis

Clustering Facility Names

```
from ClassifyFacilities import ClassifyFacilities
places = ['New', 'York', 'Miami', 'San', 'Francisco', 'Dallas',
          'NY', 'FL', 'Florida', 'CA', 'California', 'TX', 'Texas',
          'sleep', 'east', 'north', 'west', 'south']
num_clusters = 3

classifier = ClassifyFacilities()
classifier.classify(df.facility, places, num_clusters)
```

TDWI_Python_QuickCamp_2017.ipynb

Shortcut for iterating over lists

- This long unreadable line is really two nested for loops

Brackets mean that
we're going to be
returning a list

ClassifyFacilities.py

```
tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
```

3

Just return the word,
but we could do other
operations here, like
calculate the length or
change the case for
example.

1

Tokenizing the input
text as sentences
returns a list of
sentences that will be
looped over...

2

For each of those
sentences, use the
word tokenizer to get
all the individual
words...

4

Inline for loop

1. Starting with the data below
2. Use the `str.title()` function and an inline for loop to create a new list that contains all of the names in proper / title case

```
names = [ 'paul', 'donna', 'edward', 'david' ]
```

Inline for loop

```
names = [ 'paul', 'donna', 'edward', 'david' ]  
proper = [ n.title() for n in names ]  
proper
```

```
[ 'Paul', 'Donna', 'Edward', 'David' ]
```


Labeling Facility Types: Loops and Conditions

- By inspection, we can see a sort of general pattern to the name clusters that NLTK was able to identify.
 - Two tend to be “**hospital**” or “university” related
 - Only one mentions “**physician**” group

0:medic,medic center,medicin,health,clinic,**physician**,center,intern,pulmonari,sleepm

NYU Langone Pulmonary and Sleep Medicine,Aria Health Sleep Medicine,Sleep Medicine Associates of NYC,Lankenau Medical Center,Center for Sleep Medicine, Weill Cornell Medical College,Virtua Voorhees,Sutter Pacific Medical Foundation Sleep Center,Bay Sleep Clinic,Santa Clara Valley Medical,SleepMed,Woodland Healthcare,Sleep Medicine Services,Sleep Disorders Center at UCSF Medical Center at Mount Zion,Sleep Disorders Clinic Doctors Medical Center,Virtual Imaging Miami, Hialeah Sleep Lab,Pulmonary **Physicians** of South Florida, LLC,Choice **Physicians** of South Florida,International Institute of Sleep,Broward Health Medical Center,Delray Medical Center,United Health Systems,University of Miami Health System at Kendall,...

1:center,disord,hospit,disord center,univers,**hospit** disord,hospit disord center,merci,holi cross,cross

New York Sleep Disorder Center,Sleep Center of Bucks County,New York Cardiovascular Associates Sleep Wake Center,LCD Sleep Management Center,Lower Bucks Hospital,Center for Sleep Medicine at Mercy **Hospital**,Mercy Suburban **Hospital** Sleep Disorders Center,Fairfield County Sleep Center,Sleep Disorders Institute,The Bronx Pulmonary Center for Pulmonary, Asthma and Sleep Disorders,Mercy Fitzgerald **Hospital** Sleep Disorders Center,The Sleep Center at Greenwich Hospital,Bryn Mawr **Hospital**,Paoli Hospital for Sleep Medicine,Sleep Center at Riddle **Hospital**,John T. Mather Hospital Sleep Disorders Center,Danbury **Hospital** Sleep Disorders Center,Sleep Disorders Center,New York Sleep, Sinus & Thyroid Surgery Center,Redwood Sleep Centers Inc.,Peninsula Sleep Center,California Center for Sleep Disorders,North Coast Sleep Center - Santa Rosa,The Sunnyvale Sleep Center,Sleep-Wake Disorders Center of Miami,...

2:diagnost,medic diagnost,unit diagnost,unit,medic,eo,eo diagnost,diagnost hollywood,unit diagnost hollywood,hollywood

Sleep Diagnostics of NY,EOS Sleep Diagnostics,Florida Sleep & Neuro Diagnostic Services Inc.,United Sleep Diagnostics Pembroke Pines,United Sleep Diagnostics Hollywood,Sunrise Medical Group Sleep Diagnostics,Texas Medical Diagnostics Inc.

Labeling Facility Types: Loops and Conditions

```
cluster_names = classifier.get_cluster_names()
cluster_labels = {}
for num in cluster_names:
    if 'physician' in cluster_names[num]:
        cluster_labels[num] = 'clinic'
    else:
        cluster_labels[num] = 'hospital'
print(cluster_labels)

df['cluster'] = classifier.get_clusters()
df['facility_type'] = df.cluster.map(cluster_labels)

{0: 'clinic', 1: 'hospital', 2: 'hospital'}
```

TDWI_Python_QuickCamp_2017.ipynb

- Our simplistic business rule will be that any cluster in which “physician” was a key word will be labeled as a “clinic” facility.
- All others will be labeled as “hospital” facilities.

Looping and testing for values

1. Starting with the data below
2. Classify each name to see if it is male, female, or unknown
3. Return a dictionary of

```
names = ['paul', 'donna', 'edward', 'jean']  
male = ['paul', 'donna', 'edward']  
female = ['donna', 'julia', 'alice']
```

Looping and testing for values

```
names = ['paul', 'donna', 'edward', 'jean']  
male = ['paul', 'edward']  
female = ['donna', 'julia', 'alice']
```

```
genders = {}  
for n in names:  
    if n in male:  
        genders[n] = 'male'  
    elif n in female:  
        genders[n] = 'female'  
    else:  
        genders[n] = 'unknown'
```

```
genders
```

```
{'donna': 'female', 'edward': 'male', 'jean': 'unknown', 'paul': 'male'}
```

Clean Addresses: Google Maps Geocoding

```
def get_zip(self):  
    try:  
        if self._results:  
            for component in self._results['results'][0]['address_components']:  
                if 'postal_code' in component['types']:  
                    return component['long_name']  
            else:  
                return ''  
    except:  
        return ''
```

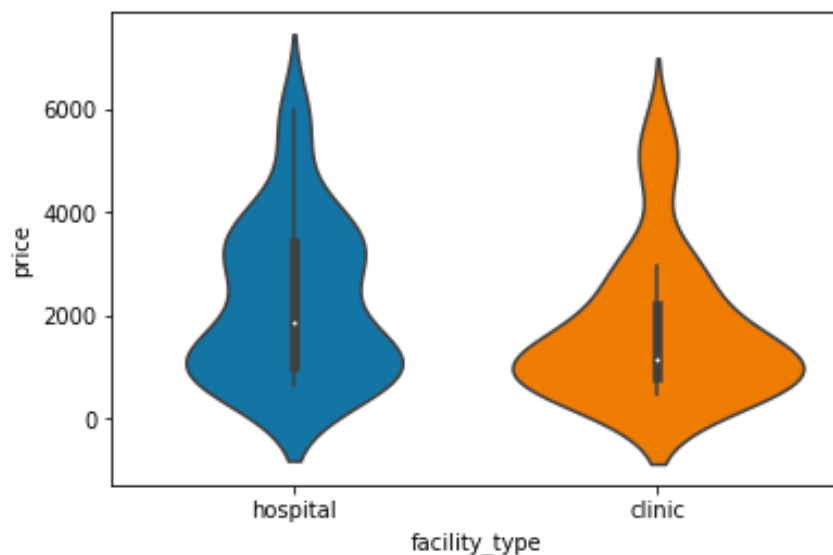
GeoCode.py

- If there are any results...
- Loop through the components of the first result's address_components
- If the type is postal_code, then return the value (long_name)

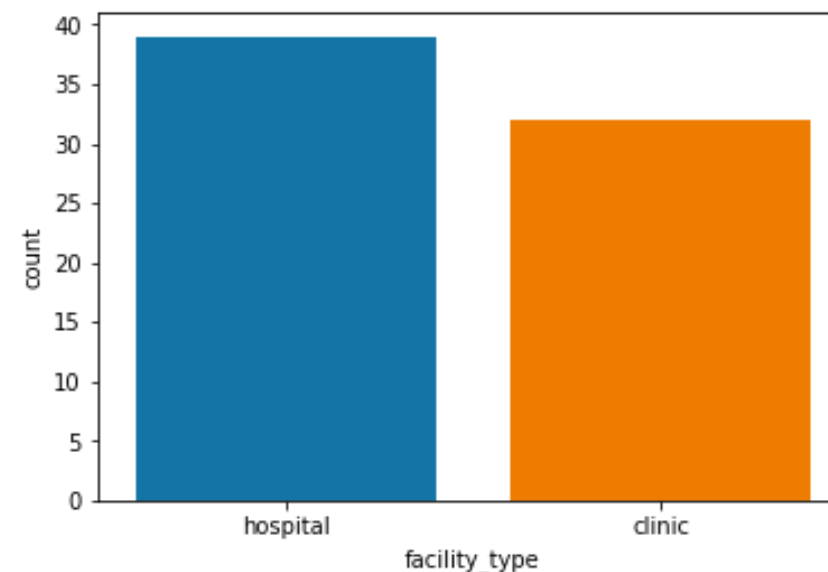
Looking at Facility Types: Histograms and Bars

```
sns.violinplot(data=df, y='price', x='facility_type')
```

TDWI_Python_QuickCamp_2017.ipynb



```
sns.countplot(data=df, x='facility_type')
```



- Look at distribution by facility type
- Look at the split between how many were classified as hospital or clinic
- Note that NLP clustering is non-deterministic (results can vary from one run to the next)

3

Get Additional External Data

External Regional Data

- 24 Hospital Ave., Danbury CT
 - 100 Bowman Drive, Voorhees, NJ 08060, United States
 - 210 Williamson St., Elizabeth NJ
 - 310 E. 14th St., #6, New York NY
 - 1615 Hill Road, Ste. 16, Novato CA
 - 1828 El Camino Real, #707, Burlingame CA
 - 985 Atlantic Avenue, Ste. 250, Alameda CA
 - 95 Montgomery Drive, Ste. 104, Santa Rosa CA
 - 1375 Sutter Street, #216, San Francisco CA
 - 3867 Montgomery Drive, Santa Rosa CA
 - 751 S. Bascom Ave., 7th Fl., #7C054, San Jose CA
- Perhaps the costs depend on the kind of neighborhood where the facility is located.
 - Census data is one convenient (if not very sophisticated) way to get those demographics.
 - However, the addresses are not consistently formatted and rarely have the facility ZIP code.

JSON Documents

- JavaScript Object Notation (JSON) is a way to describe arbitrarily complex collections of lists, dictionaries, and values.

List: [v1, v2, v3]

Dictionary: [n1: v1, n2: v2]

```
{
  "results" : [
    {
      "address_components" : [
        { "long_name" : "310",
          "types" : [ "street_number" ] },
        { "long_name" : "East 14th Street",
          "types" : [ "route" ] },
        { "long_name" : "Manhattan",
          "types" : [ "political", "sublocality", "sublocality_level_1" ] },
        { "long_name" : "New York",
          "types" : [ "locality", "political" ] },
        { "long_name" : "New York County",
          "types" : [ "administrative_area_level_2", "political" ] },
        { "long_name" : "New York",
          "types" : [ "administrative_area_level_1", "political" ] },
        { "long_name" : "United States",
          "types" : [ "country", "political" ] },
        { "long_name" : "10003",
          "types" : [ "postal_code" ] }
      ],
      "formatted_address" : "310 E 14th St, New York, NY 10003, USA",
      "geometry" : {
        "location" : {
          "lat" : 40.7318919, "lng" : -73.9845859
        },
        "location_type" : "ROOFTOP",
      },
      "place_id" : "ChIJotxfQJ5ZwokRkHhWBp-_83I",
      "types" : [ "street_address" ]
    }
  ],
  "status" : "OK"
}
```

Checkpoint Your Data: Saving and Loading Excel

- Pandas has built-in integration with Excel read/write libraries.
- This makes it easy to checkpoint your data to disk in a readable format.
- Note that when reading the DataFrame back in from Excel, Pandas (like Excel) will want to interpret the numeric values in the ZIP column as numbers. This will lose the formatting of leading zeros, so we tell Pandas to keep the ZIP code column as a string.

```
df.to_excel('./facilities_checkpoint.xlsx')
```

```
import pandas as pd
df = pd.read_excel(
    './facilities_checkpoint.xlsx',
    converters={'facility_zip':str})
```

TDWI_Python_QuickCamp_2017.ipynb



Get Local Population: DataFrames as Lookups

```
from CensusLookup import CensusLookup
lkp = CensusLookup()

population = df['facility_zip'].apply(lambda x:
    lkp.get_value(x,
        'Number; SEX AND AGE - Total population'))

df['population'] = population.apply(lambda x:
    int(str(x)) if str(x).isdigit() else 0)
```

	price	facility	address	condition	zip	radius	city	cluster	facility_type	facility_zip	population
0	700	New York Sleep Disorder Center	2951 Grand Concourse Ste. 1A, Bronx NY	sleep	10001	100	New York	2	hospital	10468	76103
1	750	Sleep Center of Bucks County	11 Friends Ln., Ste. 104, Newtown PA	sleep	10001	100	New York	1	hospital	18940	28825
2	800	Sleep Diagnostics of NY	69-39 Yellowstone Blvd., Ste. #1, Forest Hills NY	sleep	10001	100	New York	0	clinic	11375	68733
3	878	NYU Langone Pulmonary and Sleep Medicine	2408 Ocean Ave., Brooklyn NY	sleep	10001	100	New York	0	clinic	11229	80018
4	900	Aria Health Sleep Medicine	Bustleton Ave. and Verree Rd., Philadelphia PA	sleep	10001	100	New York	0	clinic	19116	33112

- CensusLookup encapsulates the work of reading in the census file and converting it to a data frame.
- We use the DataFrame `apply()` method and an inline `lambda` function to lookup the population using our CensusLookup library
- The we use `apply()` / `lambda` again to convert that population to an integer value.

Lambda syntax

```
df = pd.DataFrame({ 'birth_dt': [ '1974-05-23', '1977-09-25', '2003-01-11' ] })
```

```
def get_month(d):  
    return d[5:7]
```

```
df[ 'birth_dt' ].apply(get_month)
```

- Create a named function and apply it to the DataFrame series
- Clearer code but takes up more space

```
df[ 'birth_dt' ].apply(lambda d: d[5:7])
```

- Use an anonymous, inline lambda function to avoid creating a named function.
- Brief, but less reusable

DataFrame index values

- Each row in a DataFrame is accessible via a label or row index.
- If you know the row index, looking up the contents of that row can be very fast.
- This kind of structure is often used for lookups, just like a dictionary.

```
df = pd.DataFrame({  
    'name': ['Paul', 'Adelaide', 'Sarahlynn'],  
    'birth_dt': ['1974-05-23', '1977-09-25', '2003-01-11']  
})
```

```
df = df.set_index(df['name'])  
df
```

	birth_dt	name
name		
Paul	1974-05-23	Paul
Adelaide	1977-09-25	Adelaide
Sarahlynn	2003-01-11	Sarahlynn

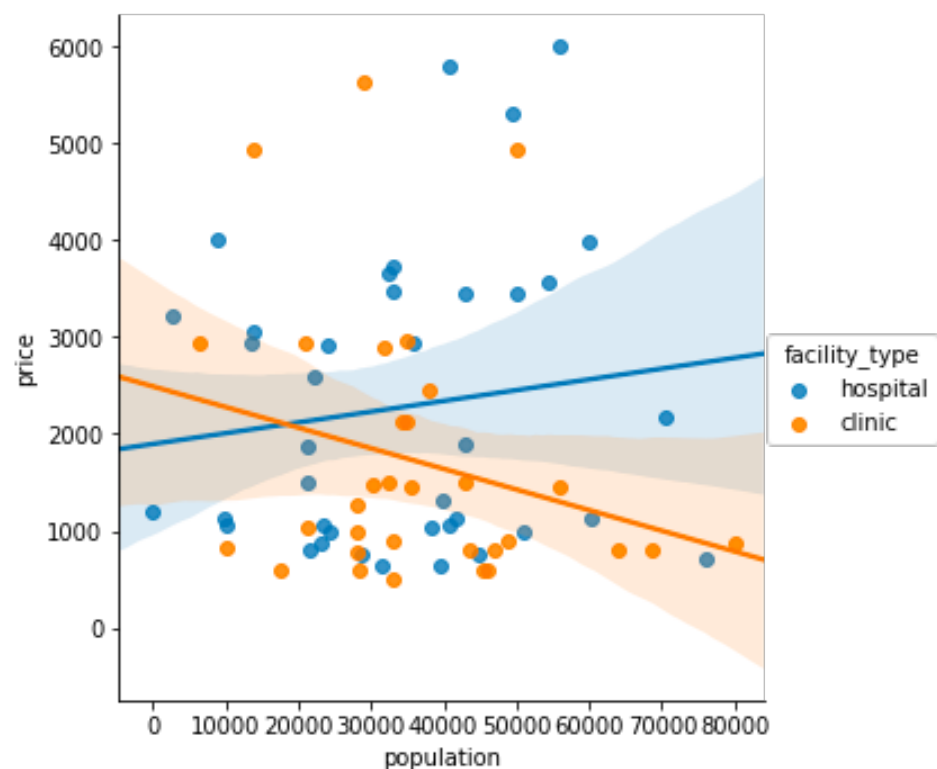
```
df.loc['Paul']['birth_dt']
```

```
'1974-05-23'
```

Plotting Relationships: Scatterplot / LM Plot

```
sns.lmplot(data=df, x='population', y='price', hue='facility_type')
```

TDWI_Python_QuickCamp_2017.ipynb



- A good way to get a hint at the relationship between two variables is to use a scatter plot.
- Seaborn has a function for quickly plotting linear regression models along with the scatter plot.
- Many Seaborn plots also permit you use the hue parameter to plot separate categorical series using separate colors.
- In this case population and price don't appear to be closely correlated.

Seaborn scatterplots

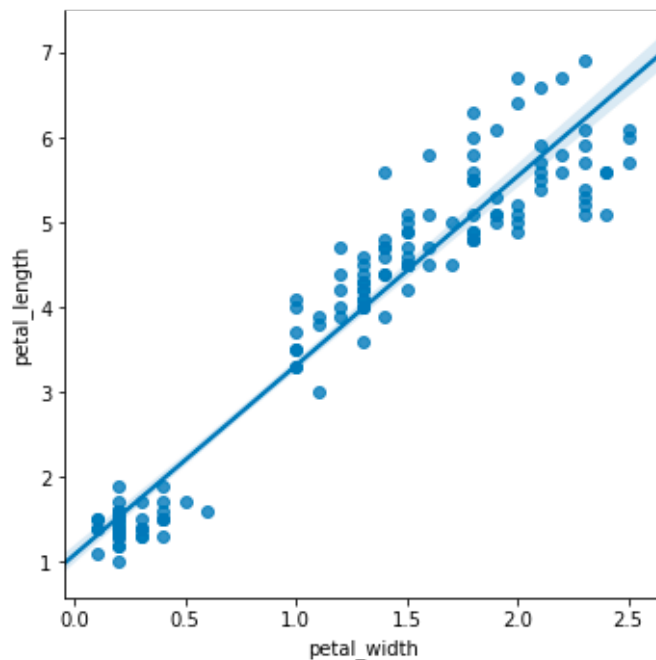
- Import the ubiquitous iris data set from seaborn
- Create a scatter plot of petal_length versus petal_width
- Color the plot in separate series based on species

```
import pandas as pd  
df = sns.load_dataset("iris")
```

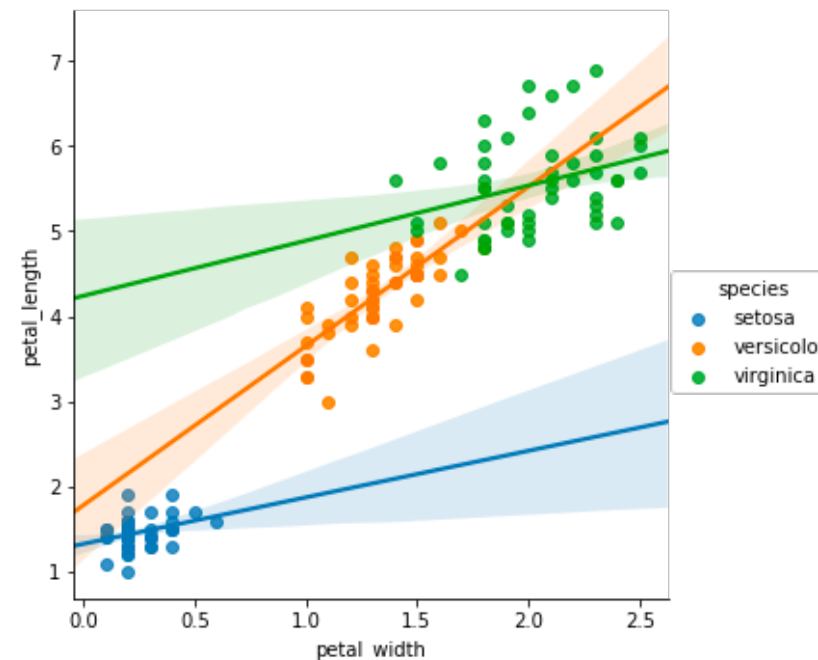
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Seaborn scatterplots

```
sns.lmplot(data=df,  
           x='petal_width',  
           y='petal_length')
```

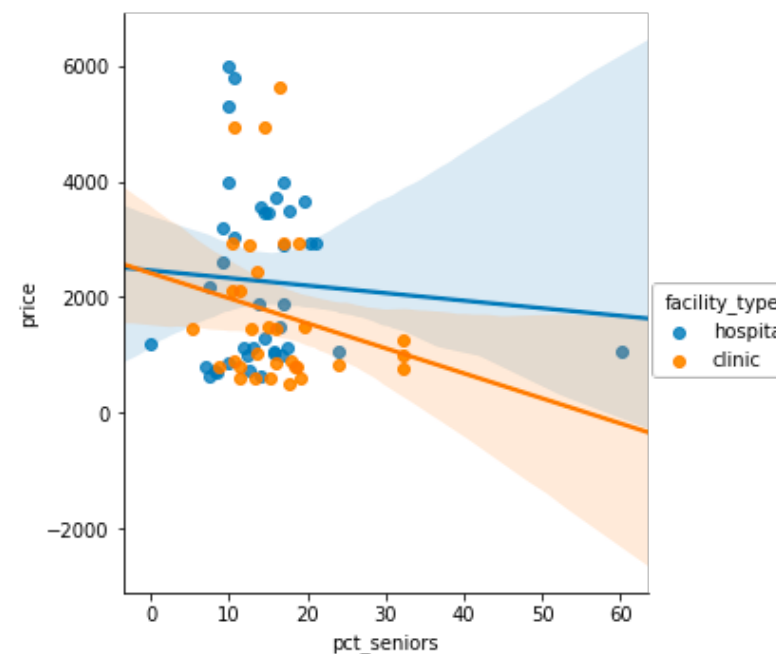
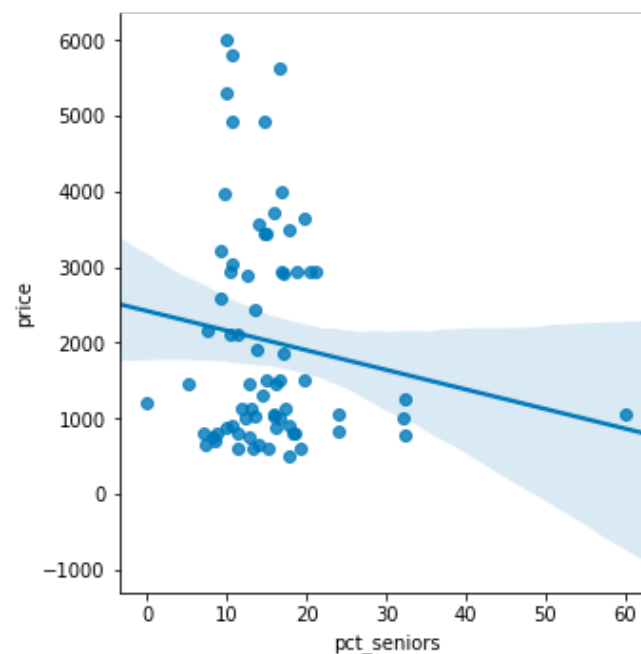
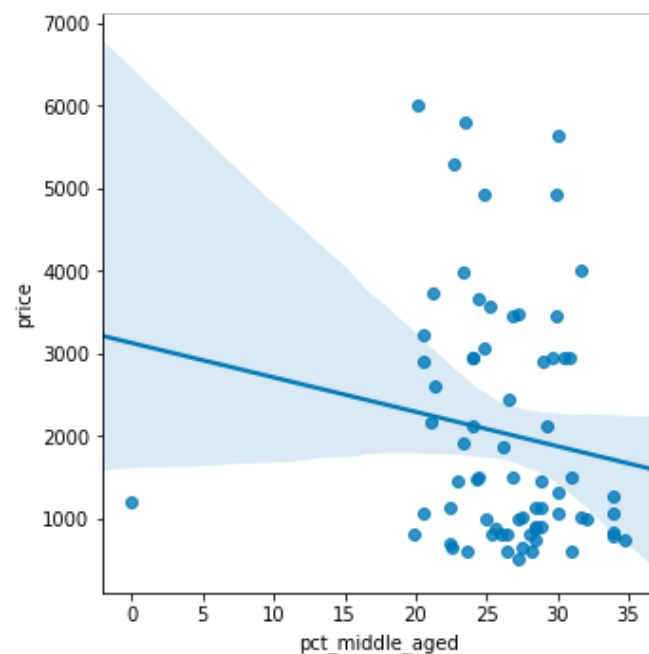


```
sns.lmplot(data=df, hue='species',  
           x='petal_width',  
           y='petal_length')
```



Plotting Relationships: Scatterplot / LM Plot

- We followed the same pattern to plot other attributes versus price:
 - The percent of people who are middle-aged (45 – 64)
 - The percent of people are over 65
- Nothing here looks particularly interesting except maybe the cut-offs.



4

Review Correlations

Possible Correlations in Data

- We want to look at several more potential relationships between price and the local ZIP code for each facility:
 - Percent; SEX AND AGE - Male population
 - Percent; RACE - Total population - One Race – White
 - Percent; RACE - Total population - One Race - Black or African American
 - Percent; RELATIONSHIP - Total population - In households – Nonrelatives
 - Percent; HOUSING OCCUPANCY - Total housing units - Vacant housing units
- This work of doing lookups based on ZIP code, formatting the data properly, and examining possible correlations is repetitive.

Be a Lazy Programmer: Lists, Loops, and Lambda

```
factors = {  
    'pct_male': 'Percent; SEX AND AGE - Male population',  
    'pct_white': 'Percent; RACE - Total population - One Race - White',  
    'pct_black': 'Percent; RACE - Total population - One Race - Black or African American',  
    'pct_non_rel': 'Percent; RELATIONSHIP - Total population - In households - Nonrelatives',  
    'pct_vacant': 'Percent; HOUSING OCCUPANCY - Total housing units - Vacant housing units',  
}
```

```
def get_factor(d, lkp, factor):  
    values = d['facility_zip'].apply(  
        lambda x: [lkp.get_value(x, factor)]  
    )  
    return values.apply(  
        lambda x: sum(float(n) if re.match('^\\d+?\\.\\d+?$', n)  
                       else 0 for n in x)  
    )  
  
for i, f in factors.items():  
    df[i] = get_factor(df, lkp, f)
```

- Dictionary that maps series name to field name
- Function to encapsulate the lookup process
- Loop through the dictionary and get the data for each series

Be a Lazy Programmer: Lists, Loops, and Lambda

- Given existing input DataFrame (`d`), a CensusLookup (`lkp`), and name of a column:
- Get the data from the lookup for each `facility_zip` in the input data.
- Return that value as a float or 0 if the value is non-numeric.

```
def get_factor(d, lkp, factor):
    values = d['facility_zip'].apply(
        lambda x: [lkp.get_value(x, factor)]
    )
    return values.apply(
        lambda x: sum(float(n) if re.match('^\\d+?\\.\\d+?$ ', n)
                       else 0 for n in x)
    )
```

- This first lambda function calls `get_value()` from CensusLookup to get the value of the specified `factor`.
- The second lambda function uses a regular expression (`re`) to test if the value looks like a number and then converts it to a float.

Be a Lazy Programmer: Lists, Loops, and Lambda

- Then we loop through all of the series that we specified in the factor dictionary
- Create a new column in our DataFrame for each result

```
for i, f in factors.items():
    df[i] = get_factor(df, lkp, f)

df.head()
```

	price	facility	address	condition	zip	radius	city	cluster	facility_type	facility_zip	population	size	pct_middle_aged	pct_seniors
0	700	New York Sleep Disorder Center	2951 Grand Concourse Ste. 1A, Bronx NY	sleep	10001	100	New York	2	hospital	10468	76103	4-large	22.4	8.5
1	750	Sleep Center of Bucks County	11 Friends Ln., Ste. 104, Newtown PA	sleep	10001	100	New York	2	hospital	18940	28825	2-small	34.7	12.7
2	800	Sleep Diagnostics of NY	69-39 Yellowstone Blvd., Ste. #1, Forest Hills NY	sleep	10001	100	New York	0	clinic	11375	68733	4-large	28.0	18.6

Scatterplot Matrix

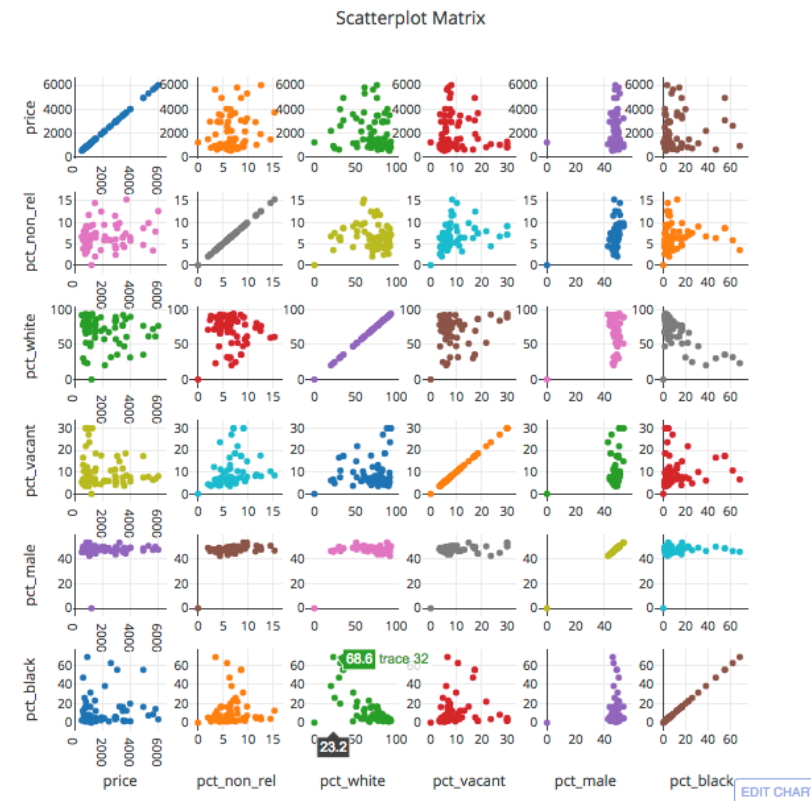
```
import pandas as pd
import numpy as np
import scipy as sp
import plotly.plotly as py
import plotly.figure_factory as ff

corr_data = df[['price']+list(factors.keys())]

fig = ff.create_scatterplotmatrix(corr_data, height=800, width=800)
py.iplot(fig, filename='Basic Scatterplot Matrix')
```

TDWI_Python_QuickCamp_2017.ipynb

- Retrieve only those columns we want to include in the correlation plot matrix.
- Use plotly's `create_scatterplotmatrix()`

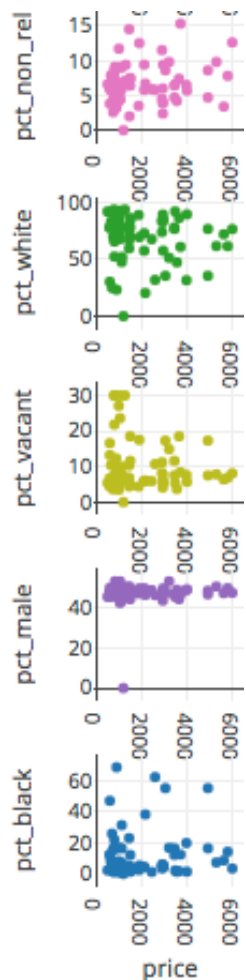


Getting vertical slices of a Data Frame

```
factors = {  
    'pct_male':    'Percent; SEX AND AGE - Male population',  
    'pct_white':   'Percent; RACE - Total population - One Race - White',  
    'pct_black':   'Percent; RACE - Total population - One Race - Black or African American',  
    'pct_non_rel': 'Percent; RELATIONSHIP - Total population - In households - Nonrelatives',  
    'pct_vacant':  'Percent; HOUSING OCCUPANCY - Total housing units - Vacant housing units',  
}  
  
df[['price']+list(factors.keys())]  
  
# df[['price', 'pct_male', 'pct_white', 'pct_black', 'pct_non_rel', 'pct_vacant']]
```

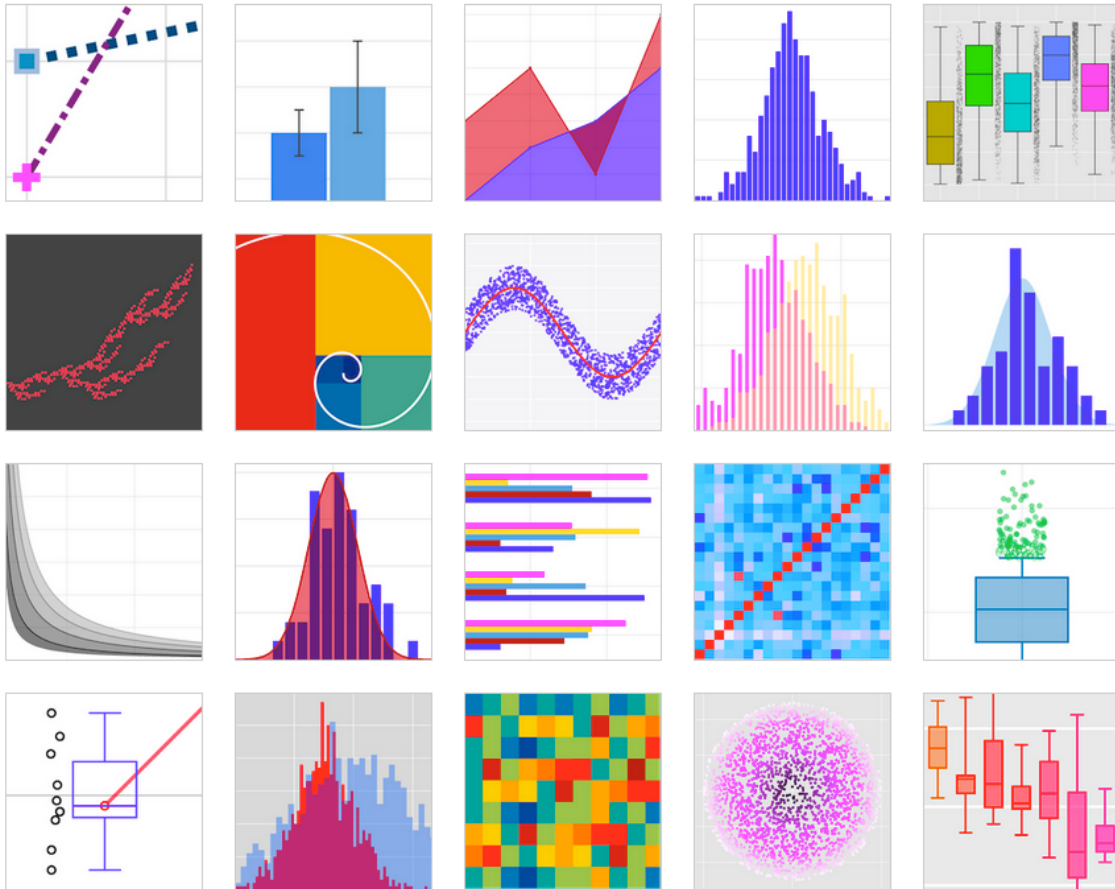
- The `list(factors.keys())` returns a list with the key names from factors.
- Providing that list plus `'price'` gives the complete list of desired columns.
- Selecting that list of columns from the DataFrame returns a new DataFrame with only those selected columns.

Scatterplot Matrix



- Appears to be no relationship between price and the percent of homes where **nonrelatives** are living.
- Doesn't appear to be a relationship between prices and percent of population of is **white**.
- It may be that those facilities where the percent of **vacant homes** is highest, only low-price options are available.
- The percent of **male** residents is narrowly around 50%
- The highest end prices appear to only be in areas where the percent of **black** residents is under 20%

Plotly Gallery



- Heatmaps
- Confidence intervals
- 3-D plots
- Network graphs
- Sankey diagrams
- Add lines and shaded areas

Review of Topics

- Variables
- Loops
- Flow control
- Functions
- Modules
- Web requests
- Inline functions (lambda)
- Inline for loops
- Inline if statement
- Lists
- Dictionaries
- Pandas Series
- Pandas DataFrames
- Plotting with seaborn and plotly
- Parsing HTML
- Reading / Writing Excel

What Next?

- Learn to manage your projects with Git:
<https://guides.github.com/activities/hello-world/>
- Turn your notebook into a slide show using nbconvert:
<http://www.damian.oquanta.info/posts/make-your-slides-with-ipython.html>
- Expand into big data processing with Spark:
<https://blog.sicara.com/get-started-pyspark-jupyter-guide-tutorial-ae2fe84f594f>
- Find a new and interesting problem!
<https://www.dataquest.io/blog/free-datasets-for-projects/>
<https://www.kaggle.com/>

Q & A

<http://amitechsolutions.com>
@AmitechSolution
@PaulBoal
paul.boal@amitechsolutions.com



AMITECH