

Intro to Programming for Public Policy Week 8

More Web APIs

Eric Potash

May 9, 2018

Missing Values

None

- ▶ None is the most basic missing value in python
- ▶ It's the default return value from functions that don't return:

```
>>> def test(): print('Hello, World')  
>>> r = test()  
>>> print(r)  
None
```

is

Python has another comparison operator `is`:

```
>>> a, b, c = 1, 2, 1
>>> a is b
False
>>> a is c
True
```

- ▶ Same as `==` on “primitive” types (numbers, booleans, strings)
- ▶ Commonly used in the expression `is None` or `is not None`
 - ▶ Equivalent to `== None` or `!= None` but more readable

is

For objects like lists and dictionaries `is` is not equivalent to `==`:

```
>>> a, b, c = [1], [2], [1]
>>> a is b
False
>>> a is c
False
```

For complex objects `is` is only True if the variables reference to the same data:

```
>>> a = [1]
>>> c = a
>>> a is c
True
```

NaN and NaT

- ▶ `nan/NaN`: provided by NumPy/pandas for missing floats (use `np.nan`)
 - ▶ Pandas automatically converts `None` to `nan` (which it calls `NaN`):

```
>>> pd.Series([1, None, 3])  
0    1.0  
1    NaN  
2    3.0  
dtype: float64
```

- ▶ `NaT`: provided by pandas for missing Timestamps (use `pd.NaT`)

Comparing missing values

NaN and NaT can be used in comparisons but are always False

- ▶ `np.nan < x` is False regardless of what `x` is
 - ▶ Same with `>`, `<=`, `>=`, and `==`
 - ▶ So `np.nan == np.nan` is False; but so is `np.nan != np.nan`
 - ▶ (R is better about this: `NA > 0` is itself `NA`)
- ▶ Same is true of `pd.NaT`

Consequences

```
>>> s = pd.Series([1, None, 3])  
>>> (s > 2).mean()  
0.33333333333333331  
>>> (s <= 2).mean()  
0.33333333333333331
```


Arithmetic with missing values

Any arithmetic with `nan` results in `nan`:

```
>>> 1 + np.nan
nan
>>> np.nan * 2
nan
>>> 1/np.nan
nan
```

Pandas does better, ignoring nans in aggregation:

```
>>> s = pd.Series([0, None, 2])
>>> s.mean()
1.0
>>> s.sum()
2.0
```

Error handling

Guantanamo code

```
def get_years(det_page):  
    """  
    Given detainee page get years detained  
    """  
    div = det_page.find('div',  
                        class_='nytint-detainee-fullcol')  
    matches = time_pattern.findall(div.text)  
    return int(matches[0].rstrip(' year'))
```

What happens if there is no div of class
nytint-detainee-fullcol on the detainee page?

Guantanamo code: raise an error

```
def get_years(det_page):  
    """  
    Given detainee page get years detained  
    """  
    div = det_page.find('div',  
                        class_='nytint-detainee-fullcol')  
  
    if div is None:  
        raise ValueError("No detainee div found")  
    else:  
        matches = time_pattern.findall(div.text)  
        return int(matches[0].rstrip(' year'))
```

Guantanamo code: ignore error

```
def get_years(det_page):  
    """  
    Given detainee page get years detained  
    """  
    div = det_page.find('div',  
                        class_='nytint-detainee-fullcol')  
  
    if div is None:  
        return None  
    else:  
        matches = time_pattern.findall(div.text)  
        return int(matches[0].rstrip(' year'))
```

Guantanamo code: ignore error revised


```
def get_years(det_page):  
    """  
    Given detainee page get years detained  
    """  
    div = det_page.find('div',  
                        class_='nytint-detainee-fullcol')  
  
    if div is not None:  
        matches = time_pattern.findall(div.text)  
        return int(matches[0].rstrip(' year'))
```

Guantanamo code: ignore error revised

```
def get_years(det_page):  
    """  
    Given detainee page get years detained  
    """  
    try:  
        div = det_page.find('div',  
                             class_='nytint-detainee-fullcol  
  
        matches = time_pattern.findall(div.text)  
        return int(matches[0].rstrip(' year'))
```

Pandas read_html()

HTML Table



WIKIPEDIA
The Free Encyclopedia

- Main page
- Contents
- Featured content
- Current events
- Random article
- Donate to Wikipedia
- Wikipedia store

Interaction

- Help
- About Wikipedia
- Community portal
- Recent changes
- Contact page

Tools

- What links here
- Related changes
- Upload file
- Special pages
- Permanent link
- Page information


Not logged in

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)

Table of biofuel crop yields

From Wikipedia, the free encyclopedia



This article **needs additional citations for verification**. Please help [improve this article](#) by adding citations to reliable [sources](#). Unsourced material may be challenged and removed. *(April 2008)* [\(Learn how and when to remove this message\)](#)

The following table shows the [vegetable oil](#) yields of common [energy crops](#) associated with [biodiesel](#) production. This is unrelated to [ethanol](#) and [cellulose](#) content instead of oil yields.

Crop	kg oil/ha/yr	litres oil/ha	lbs oil/acre	US gal/acre
maize (corn)	147	172	129	18
cashew nut	148	176	132	19
oats	183	217	163	23
lupin (lupine)	195	232	175	25
kenaf	230	273	205	29
calendula	256	305	229	33
cotton	273	325	244	35
hemp	305	363	272	39
soybean	375	446	335	48
coffee	386	459	345	49

Figure 1: https://en.wikipedia.org/wiki/Table_of_biofuel_crop_yields

read_html()

```
In [21]: import pandas as pd  
pd.read_html('https://en.wikipedia.org/wiki/Table_of_biofuel_crop_yields')[0]
```

```
Out[21]:
```

	0	1
0	NaN	This article needs additional citations for ve...

```
In [22]: pd.read_html('https://en.wikipedia.org/wiki/Table_of_biofuel_crop_yields')[1]
```

```
Out[22]:
```

	0	1	2	3	4
0	Crop	kg oil/ha/yr	litres oil/ha	lbs oil/acre	US gal/acre
1	maize (corn)	147	172	129	18
2	cashew nut	148	176	132	19
3	oats	183	217	163	23
4	lupin (lupine)	195	232	175	25
5	kenaf	230	273	205	29
6	calendula	256	305	229	33
7	cotton	273	325	244	35

read_html() arguments

```
In [23]: import pandas as pd
pd.read_html('https://en.wikipedia.org/wiki/Table_of_biofuel_crop_yields',
             match='Crop',
             header=0)[0]
```

Out[23]:

	Crop	kg oil/ha/yr	litres oil/ha	lbs oil/acre	US gal/acre
0	maize (corn)	147.0	172.0	129.0	18.0
1	cashew nut	148.0	176.0	132.0	19.0
2	oats	183.0	217.0	163.0	23.0
3	lupin (lupine)	195.0	232.0	175.0	25.0
4	kenaf	230.0	273.0	205.0	29.0
5	calendula	256.0	305.0	229.0	33.0
6	cotton	273.0	325.0	244.0	35.0
7	hemp	305.0	363.0	272.0	39.0
8	soybean	375.0	446.0	335.0	48.0
9	coffee	386.0	459.0	345.0	49.0
10	flax (linseed)	402.0	478.0	359.0	51.0

APIs

API Overview

- ▶ Application Programming Interfaces (APIs) are web-based resources that serve data directly
- ▶ Typically in json, csv, or xml format
- ▶ You can access them with requests in python or curl
- ▶ The parameters are typically provided in the URL with a query string (?param1=value1¶m2=value2)

REST

REST is a standard style for organizing API resources. The philosophy is that:

- ▶ Client and server are “stateless”, i.e. each request is independent and the server does not save state between requests
- ▶ Service is scalable and cacheable
- ▶ Typically use HTTP methods (GET, DELETE, PUT, etc.) meaningfully

Census API

- ▶ U.S. Census data available through **API**
- ▶ For example the **5-year ACS** estimates
 - ▶ List of variables

https://api.census.gov/data/2014/acs5/profile?get=DP02_0037PE,NAME&for=state:*

- ▶ Can simply `curl`

curl

```
$ curl "https://api.census.gov/data/2014/acs5/profile?get=I"
-o fertility.json
$ head fertility.json
[["DP02_0037PE","NAME","state"],
["41.7","Alabama","01"],
["34.2","Alaska","02"],
["39.7","Arizona","04"],
["37.7","Arkansas","05"],
["33.2","California","06"],
["28.3","Colorado","08"],
["33.7","Connecticut","09"],
["39.3","Delaware","10"],
["47.7","District of Columbia","11"],
...]
```


What is JSON?

JSON stands for JavaScript Object Notation and is a data format that can represent both lists and dictionaries with numeric and string elements, keys and values.

The format is basically the same as Python's own representations of lists and dictionaries.

python

```
▶ >>> path = "https://api.census.gov/data/2014/acs5/profiles"
>>> j = requests.get(path).json()
>>> j
[[u'DP02_0037PE', u'NAME', u'state'],
 [u'41.7', u'Alabama', u'01'],
 [u'34.2', u'Alaska', u'02'],
 [u'39.7', u'Arizona', u'04'],
 [u'37.7', u'Arkansas', u'05'],
 [u'33.2', u'California', u'06'],
 ...]
```

How can we put this data in a pandas DataFrame?

python

```
▶ >>> path = "https://api.census.gov/data/2014/acs5/profi
>>> j = requests.get(path).json()
>>> j
[[u'DP02_0037PE', u'NAME', u'state'],
 [u'41.7', u'Alabama', u'01'],
 [u'34.2', u'Alaska', u'02'],
 [u'39.7', u'Arizona', u'04'],
 [u'37.7', u'Arkansas', u'05'],
 [u'33.2', u'California', u'06'],
 ...]
```

How can we put this data in a pandas DataFrame?

```
▶ >>> pd.DataFrame(j[1:], columns=j[0])
```

Example: Census geocoder

The [Census geocoder website](#) geocodes addresses. I.e. given a CSV of addresses:

```
$ cat short.csv  
a, "4701 Ocean View Ave", "Virginia Beach", "VA", "23455"  
b, "3630 S WELLS ST", "CHICAGO", "IL", "60609"  
c, "2957 N HOYNE AVE", "CHICAGO", "IL", "60618"  
d, "5110 N DAMEN AVE", "CHICAGO", "IL", "60625"  
e, "231 N PINE AVE", "CHICAGO", "IL", "60644"  
f, "231 N PINE AVE", "CHICAGO", "IL", "60644"
```

it can add columns for coordinates, county, census tract, and other geographic components.

Geocoder curl

The geocoder can be used programmatically with curl:

```
$ curl -F addressFile=@short.csv \  
      -F vintage=Current_Current \  
      -F benchmark=Public_AR_Current \  
      -F layers=9 \  
      "https://geocoding.geo.census.gov/geocoder/geographies
```

(Script [here](#))

See the documentation [here](#).

Example: Google Maps API

- ▶ Well-documented APIs for [geocoding](#), [directions](#), [distances](#), etc.
- ▶ For more than a few calls requires a free API key.

```
api = 'https://maps.googleapis.com/maps/api/'  
  
geoc = 'geocode/json?address=Harris School'  
key  = '&key=YOUR_KEY'  
  
j = requests.get(api + geoc).json()
```

params dictionary

Instead of building the parameters into the URL manually, use the `params` argument to `get`:

```
api = 'https://maps.googleapis.com/maps/api/'  
geoc = 'geocode/json'  
params = {'address': 'Harris School'}  
  
j = requests.get(api + geoc, params=params).json()
```

Distance matrix

```
dist = 'distancematrix/json'  
params = {'origins': 'Chicago,IL|Tucson,AZ',  
          'destinations': 'Philly'}  
j = requests.get(api + dist, params=params).json()
```


Hidden APIs

Some websites have APIs that are not publicly documented.

Example: HealthData.org

The visualization at [HealthData.org](https://vizhub.healthdata.org/gbd-compare/) shows disease data (Deaths, YLDs, DALYs) for different diseases for different subpopulations (men/women, age groups).

By loading the page with the web developer tools on, we can find that the data is transmitted through a hidden API:

`https://vizhub.healthdata.org/gbd-compare/api/metadata?lang=41`

We can load this JSON directly into python to analyze the data ourselves.

```
url = ('https://vizhub.healthdata.org/' +  
       'gbd-compare/api/metadata?lang=41')  
j = requests.get(url).json()
```