

### Pandas Data Frame

```
df_1 = pd.DataFrame({'A': [0, 1, 2],
                    'B': [3, 4, 5]})
df_2 = pd.DataFrame([[0, 1, 2], [3, 4, 5]],
                    columns=['A', 'B', 'C'])
-----
df[:5] #First 5 lines
df.head()
df[df.key==10] #Where key = 10
df[df.key==10][:5] #First 5 lines &
Where key=10
-----
df_3.iloc[0]
df_3.loc['column-A']
df_3['row-A']
df_3.iloc[1, 3]
df_3.iloc[1:4]
df_3[['row-A', 'row-D']]
df.values.sum()
-----
df_1 = pd.DataFrame({'A': [0, 1, 2],
                    'B': [3, 4, 5]})
df_1.shift(axis=0)
      A B
0 NaN NaN
1 0.0 3.0
2 1.0 4.0
df_1.shift(axis=1)
      A B
0 NaN 0.0
```

### Pandas Data Frame (cont)

```
1 NaN 1.0
2 NaN 2.0
# To replace NaN with value (eg. 0)
df_1.shift(axis=0).fillna(0)
-----
# To see the difference between
columns/rows
df_1.diff(axis=0)
df_1.diff(axis=1)
-----
df = pd.DataFrame({
    'a': [1, 2, 3],
    'b': [10, 20, 30],
    'c': [5, 10, 15]
})
def add_one(x):
    return x + 1
df.applymap(add_one)
      a b c
0 2 11 6
1 3 21 11
2 4 31 16
# applymap is different from apply
in Pandas DF
# apply does column by column (or
row by row)
# instead of element by element
def standardize_column(column):
    return (column -
column.mean())/column.std()
df.apply(standardize_column)
-----
# only works in a single series
(column or rows)
```

### Pandas Data Frame (cont)

```
# from entire DF use apply()
df.iloc[:, 0].sort_values()
-----
# columns vs rows operations,
respectively
df - df.mean(axis=0)
df.sub(df.mean(axis=1), axis=0)
-----
# group by
df.groupby('column').groups
df.groupby(['column1',
'column2']).groups
df.groupby('column').sum()
df.groupby('column').get_group('value1')
list(df.groupby('column'))
df.groupby('c4')
[['c1', 'c2']].apply(func)
http://wesmckinney.com/blog/groupby-fu-improvements-in-grouping-and-aggregating-data-in-pandas/
-----
# merge (join)
dfLeft.merge(dfRight,
             left_on=['A', 'B', 'C'],
             right_on=['A', 'BB', 'CC'],
             how='inner')
```

### Numpy Functions

a.max()	Max value
a.argmax()	Index of max value
a.mean(axis=0)	For 2D arrays (Column)
a.mean(axis='columns')	
a.mean(axis=1)	For 2D arrays (Row)
a.mean(axis='index')	



By **drykka01**  
[cheatography.com/drykka01/](https://cheatography.com/drykka01/)

Not published yet.  
 Last updated 17th July, 2017.  
 Page 1 of 5.

Sponsored by **Readability-Score.com**  
 Measure your website readability!  
<https://readability-score.com>

### Important - Numpy Arrays

```
import numpy as np
a=np.array([1,2,3,4,5])
slice=a[:3]
slice[0]=100
print a
[] [100,2,3,4,5]
```

### RegEx

```
pattern = "([0-9]){2}"
prog = re.compile(pattern)
match = re.match(prog, str)
if not match:
    print date_str
```

<http://regexr.com/>  
<https://docs.python.org/3.6/library/re.html>

### PyMongo

```
from pymongo import MongoClient
client =
MongoClient("mongodb://localhost:27
017")
db = client.examples
#what do you want to find
query = { "abc" : "xyz", "ghi":
"mop" }
query = { "tuv" : { "$gt" : 1,
"$lte" : 100 } }
query = { "abc" : { "$ne" : "xyz"
} }
#what information you want to see,
removing id
projection = { "_id": 0, "name": 1
}
autos = db.autos.find(query,
projection)
#find
db.autos.find().count() #count all
#insert
```

### PyMongo (cont)

```
a = {} #dic
db.autos.insert(a)
#check if doc with field exists &
count
db.cities.find({"abc":
{"$exists":1}}).count()
#pretty print
db.cities.find().pretty()
#regex
db.cities.find({"abc":{"$regex":"
[0-9]"}})
#in - return docs with at least one
of the values
db.cities.find({"abc":{"$in":
[1,2,3]})}
#all - return docs with all of the
values
db.cities.find({"abc":{"$all":
[1,2,3]})}
#arrays inside arrays
"n1" : { "n2" : { "n3" : [4] } }
db.city.find( {"n1.n2.n3" : 4} )
#update V1
doc = db.data.find_one({"a":"b"})
doc["new"] = "value"
db.data.save(doc)
#update V2 (adding)
doc =
db.data.update({"a":"b"}, {"$set":
{"c":"d"}})
#update V3 (removing) & multi
lines
doc = db.data.update({"a":"b"},
{"$unset":
{"c":""}},multi=True)
#remove
db.data.remove({"a":"b"})
<command line>
```

### PyMongo (cont)

```
mongoimport -d examples -c
myautos2 --file autos.json
```

<https://docs.mongodb.com/manual/reference/operator/query/>

### Load Data

```
# Read from csv file
import csv
def parse_file(datafile):
    data = []
    with open(datafile,'rb') as
f:
        reader = csv.reader(f)
        for row in reader:
            data.append(row) # List
of Lists
with open(input_file, "r") as f:
    reader = csv.DictReader(f)
    header = reader.fieldnames
    rows = list(reader) # Lists of
Dic +Header
import unicodcsv
with open('file.csv', 'rb') as f:
    reader =
unicodcsv.DictReader(f)
    file_dic = list(reader) # Dic?
import pandas as pd
file_df = pd.read_csv('file.csv',
index_col=False, header=0); # Data
Frame
```

### Data Types

#### Dictionary

```
dictionary = {}
key = 'abc'
value = '123'
dictionary[key] = value
for key in dictionary:
    print(dictionary[key])
[] 123
for key, value in
dictionary.items():
    print(key)
    print(value)
[] abc
[] 123
```

#### Set

```
uniq_dataset = set()
uniq_dataset.add('A')
uniq_dataset.add('A')
uniq_dataset.add('B')
print(uniq_dataset)
[] {'A', 'B'}
```

### Panda Series

```
import pandas as pd
-----
s1 = pd.Series([1, 2, 3, 4],
               index=['a', 'b',
                    'c', 'd'])
s2 = pd.Series([10, 20, 30, 40],
               index=['b', 'd',
                    'a', 'c'])
print s1 + s2
a 31
```

### Panda Series (cont)

```
b 12
c 43
d 24
dtype: int64
-----
Obs: NaN if the index does not
exists for both
res = s1 + s2
res.dropna()
or
s1.add(s2, fill_value=0)
-----
s = pd.Series([1, 2, 3, 4, 5])
def add_three(x):
    return x + 3
print s.apply(add_three)
4 5 6 7 8
```

### Indexes and Slices

len(a)	6
a[0]	0
a[5]	5
a[-1]	5
a[-2]	4
a[1:]	[1,2,3,4,5]
a[:5]	[0,1,2,3,4]
a[:-2]	[0,1,2,3]
a[1:3]	[1,2]
a[1:-1]	[1,2,3,4]
b=a[:]	Shallow copy of a

Indexes and Slices of a=[0,1,2,3,4,5]

### XML

```
import xml.etree.ElementTree as ET
tree =
ET.parse('country_data.xml')
root = tree.getroot() # First tag
for child in root:
    print child.tag,
child.attrib

print root[0][1].text
for country in
root.findall('country'):
    rank =
country.find('rank').text
    name = country.get('name')
    print name, rank
<au id="A1" ca="yes">
<snm>Mei-Dan</snm>
<fnm>Omer</fnm>
<insr iid="I1"/>
<insr iid="I2"/>
<email>omer@extremegate.com<
/email>
</au>
for author in
root.findall('./fm/bibl/aug/au'):
insr = []
for i in
author.findall('./insr'):
    insr.append(i.get('iid'))
#insr.append(i.attrib["iid"])

data = {
    "fnm":
author.find('./fnm').text,
    "snm":
author.find('./snm').text,
    "email":
author.find('./email').text,
    "insr": insr
}
```

<https://docs.python.org/2/library/xml.etree.elementtree.html>

### HTML Requests

```
from bs4 import BeautifulSoup
import requests

html_page = "page_source.html"

with open(page) as fp:
    soup = BeautifulSoup(fp,
        "lxml")

r =
requests.post("http://www.transtats
.bts.gov/
Data_Elements.aspx?Data=2",
    data={'AirportList': "BOS",
        'CarrierList': "VX",
        'Submit': 'Submit',
        "__EVENTTARGET": "",
        "__EVENTARGUMENT": "",
        "__EVENTVALIDATION":
soup.find(id="__EVENTVALIDATION"),
    "__VIEWSTATE":
soup.find(id="__VIEWSTATE")
    })

print r.text
```

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

### Write to CSV

```
import csv

with open("output.csv", "wb") as f:
    writer = csv.writer(f)
    writer.writerows(your_list_of_lists)
```

### Dates

```
from datetime import datetime as dt

date_str = '2017-04-19'
date_dt = dt.strptime(date_str,
    '%Y-%m-%d')

print(date_dt)
[] 2017-04-19 00:00:00

dt.strptime(date_str, '%Y-%m-%dT%H:%M:%S%z')

datetime(2000, 1, 1)
```

<https://docs.python.org/3/library/datetime.html#strptime-and-strptime-behavior>

### If

```
if x not in dictionary and y != z:
    print()

if not x or y > 7:
    print()

if x <= y <= z:
    print()

if "a" in "abc":
    print()
```

### Loops

```
for i in range(list.size):
for e in list:

# Iterate through two lists in parallel

# zip stops when the shorter of foo or bar stops

for a,b in list(zip(foo,bar)):
```

### Lists - General

list = []	new list
list.append('shemp')	append at end
list.insert(0, 'xxx')	insert at index 0
print list.index('curly')	2
list.extend(['yyy', 'zzz'])	list of elems at end
list.remove('curly')	search and remove
list.pop(1)	remove and return

### List of Dictionaries

```
people = [
    {'name': "Tom", 'age': 10},
    {'name': "Mark", 'age': 5},
    {'name': "Pam", 'age': 7}
]

list(filter(lambda person:
person['name'] == 'Pam', people))
[] [{'age': 7, 'name': 'Pam'}]

people[1]['name']
[] Mark
```

### Dictionary of Lists

```
from collections import
defaultdict

s = [('yellow', 1), ('blue', 2),
('yellow', 3), ('blue', 4), ('red',
1)]

d = defaultdict(list)

for k, v in s:
    d[k].append(v)

d.items()
```



By **drykka01**  
[cheatography.com/drykka01/](https://cheatography.com/drykka01/)

Not published yet.  
 Last updated 17th July, 2017.  
 Page 4 of 5.

Sponsored by **Readability-Score.com**  
 Measure your website readability!  
<https://readability-score.com>

### Dictionary of Lists (cont)

```
{ ('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3]) }
```

The idea is to group the values by the keys

### Matplotlib

```
data = [1, 2, 1, 3, 3, 1, 4, 2]
import matplotlib.pyplot as plt
plt.hist(data)
```

To show it:

```
%matplotlib inline (for notebooks)
or
plt.show()
```

[https://matplotlib.org/users/pyplot\\_tutorial.html](https://matplotlib.org/users/pyplot_tutorial.html)

### Statistics

#Correlation

```
std_x = (x - x.mean()) /
x.std(ddof=0)
```

```
std_y = (y - y.mean()) /
y.std(ddof=0)
```

```
correlation = (std_x *
std_y).mean()
```

#By default, Pandas' std() function computes the standard deviation using Bessel's correction. Calling std(ddof=0) ensures that Bessel's correction will not be used.

NumPy's corrcoef() function can be used to calculate Pearson's r, also known as the correlation coefficient.

### Excel Files

```
import xlrd
datafile = "file.xls"
workbook =
xlrd.open_workbook(datafile)
sheet =
workbook.sheet_by_index(0)
sheet_data =
[[sheet.cell_value(r, col) for col
in range(sheet.ncols)] for r in
range(sheet.nrows)]
# (column, line)
sheet.cell_type(3, 2)
sheet.cell_value(3, 2)
# (column, start_line, end_line)
sheet.col_values(3, start_rowx=1,
end_rowx=4)
sheet.ncols
xlrd.xldate_as_tuple(cell_in_exceltime, 0)
```

### Pretty Print

```
import pprint
pprint.pprint(stuff)
```

<https://docs.python.org/3/library/pprint.html>

### Map, Filter and Reduce

```
#Map
map(function_to_apply,
list_of_inputs)
map(float, list) #Convert all in
list to float
```

[http://book.pythontips.com/en/latest/map\\_filter.htm](http://book.pythontips.com/en/latest/map_filter.htm)

### JSON requests (WS)

```
import json
import requests
BASE_URL =
"http://musicbrainz.org/ws/2/"
ARTIST_URL = BASE_URL + "artist/"
params["fmt"] = "json"
params["query"] = "artist:Avril"
r = requests.get(url + uid,
params)
print r.json()
```

<http://docs.python-requests.org/en/master/user/quickstart/>

### Pkg

```
!pip install <link>
```



By [drykka01](https://cheatography.com/drykka01/)  
cheatography.com/drykka01/

Not published yet.  
Last updated 17th July, 2017.  
Page 5 of 5.

Sponsored by [Readability-Score.com](https://readability-score.com)  
Measure your website readability!  
<https://readability-score.com>