NPRs Python Notes

Nicholas P. Ross

October 8, 2017

Abstract

The is my (NPR's) set of Pandas notes, spun-out from my set of Python notes. Things started as wanting to just be an "IDL to Python CheatSheet", and have naturally and organically snowballed from there. Suffice to say, when this document reached 47 pages long (and I started to take notes on how to do pytests ;-), this was no long a Cheat Sheet, and became something else; a general Python resource.

You will be able to find the latest version of these notes and indeed the .tex file at:
`https://github.com/d80b2t/Research_Notes`.

Contents

1 General Intro

Pandas - Python Data Analysis Library
`pandas.pydata.org/pandas-docs`

Pandas puts things into a spreadsheet-like format. But it's friggin fast, with a whole bunch of pre-made commands and functions.

2 DataFrames

So, pandas tends to work in these things called `DataFrames`. I (currently) don't understand DataFrames :-/
http://pandas.pydata.org/pandas-docs/stable/dsintro.html

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

If axis labels are not passed, they will be constructed from the input data based on common sense rules.

```
In [11]: data = pd.read_table(path, delimiter=’\s+’)

In [12]: type(data)
Out[12]: pandas.core.frame.DataFrame

In [13]: df = pd.DataFrame(data, columns=cols)

In [14]: type(df)
Out[14]: pandas.core.frame.DataFrame

df = pd.DataFrame(data, columns=cols)
```

```
  In [1]: from pandas import Series, DataFrame

  In [2]: import pandas as pd
```

“Thus whenever you see `pd.` in the code, it’s referring to pandas. Series and DataFrame are used so much that I find it easier to import them into the local namespace.”

Some DataFrame basics:

```
import pandas as pd

data_dir = '/cos_pc19a_npr/data/String/UNSW-NB15/'
data_file = 'UNSW-NB15_1.csv'
data_path = data_dir+data_file

cols = ["srcip", "sport", "dstip", "dsport", "proto", "state",
    "dur", "sbytes", "dbytes", "sttl", "dttl", " sloss",
       "dloss", "service", "Sload", "Dload", "Spkts", "Dpkts",
           "swin", "dwin", "stcpb", "dtcpb", "smeansz ",
       "dmeansz", "trans_depth", "res_bdy_len", "Sjit", "Djit",
           "Stime", "Ltime", "Sintpkt", "Dintpkt", "tcprtt",
       "synack", "ackdat", "is_sm_ips_ports", "ct_state_ttl",
           "ct_flw_http_mthd", "is_ftp_login", "ct_ftp_cmd",
       "ct_srv_src", "ct_srv_dst", "ct_dst_ltm", "ct_src_ltm",
           "ct_src_dport_ltm", "ct_dst_sport_ltm",
       "ct_dst_src_ltm", "attack_cat", "Label"]
df = pd.read_csv(data_path, names=cols)

df               ## prints out a summary of the whole table/dataframe

df[0]          ## is an error
df[0][0]     ## is an error

df["srcip"]      ## gives the srcip column
df["srcip"][0]  ## gives the first entry in the
                    ## single quotes are absolutely fine too.

df.iloc[0]       ## gives the first row, and is a
    pandas.core.series.Series
df.iloc[i]        ## returns the ith row of df

print(df['srcip'][0:20], type(df['srcip'])) ## will print the first
    20 rows of the scrip colum
```

The recommended methods of indexing are:

- `.loc` if you want to label index
- `.iloc` if you want to positionally index.

```
print(type(df))
print(type(df["dur"]))

x_tup =(df["dur"], df['sbytes'], df['dbytes'], df['Sload'],
    df['Dload'],
       df['ct_flw_http_mthd'], df['ct_srv_dst'], df['ct_src_ltm'])
x_lis =[df["dur"], df['sbytes'], df['dbytes'], df['Sload'],
    df['Dload'],
    df['ct_flw_http_mthd'], df['ct_srv_dst'], df['ct_src_ltm']]

print(type(x_tup))
print(type(x_lis))

<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
<class 'tuple'>
<class 'list'>
```

3 YouTube videos of some Panda Basics

Okay, some real basics. The following codes should be found at:
https://github.com/d80b2t/python/tree/master/pandas/YouTube_resources.

3.1 intro_code_p1 and basics_p2

```
'''
www.youtube.com/watch?v=0UA49Ds1XXo&index=2&list=PLQVvvaa0QuDc-3szzjeP6N6b0aDrrKyL-
pythonprogramming.net/basics-data-analysis-python-pandas-tutorial/
'''

import pandas as pd

## Start with a dictionary:
web_stats = {'Day':[1,2,3,4,5,6],
    'Visitors':[43,34,65,56,29,76],
    'Bounce Rate':[65,67,78,65,45,52]}

## Can turn this dictionary into a dataframe with:
df = pd.DataFrame(web_stats)

## Print the first five lines:
print(df.head())

## Print the last five lines::
print(df.tail())
## Print the last two lines::
print(df.tail(2))

## change the index, and keep it changed (with inplace)!!
df.set_index('Day', inplace=True)

## Print/reference a column...
#print(df['Visitors'])

## or could have done::
# print(df['Bounce Rate'])
## if there was a gap above for Bounce Rate
## and very often there will be!!!

## or...
#print(df.Visitors)
## but NOT::
# print(df.Bounce Rate) #when no underscore.

## Referencing several columns...
#print(df[['','']])
```

```
print(df[[’Bounce_Rate’,’Visitors’]])

## convert this to a List...
print(df.Visitors.tolist)
##Name: Visitors, dtype: int64>
print(df.Visitors.tolist())
#[43, 34, 65, 56, 29, 76]

##  In Pandas, there’s lists, and there are lists of Lists.
##  But, there’s no such thing as an array. So,
print(df[[’Bounce_Rate’,’Visitors’]].tolist())
##  Gives an error!!!
##      AttributeError: ’DataFrame’ object has no attribute ’tolist’
##  it’s being treated like it’s an array

## So, try instead:
print(np.array(df[[’Bounce_Rate’,’Visitors’]]))
#[[65 43]
# [67 34]
# [78 65]
# [65 56]
# [45 29]
# [52 76]]

## And finally:
df2 = pd.DataFrame(np.array(df[[’Bounce_Rate’,’Visitors’]]))
print(df2)
#     0   1
# 0  65  43
# 1  67  34
# 2  78  65
# 3  65  56
# 4  45  29
# 5  52  76
```

3.2 io_basics_p3.py

```
'''
http://pandas.pydata.org/pandas-docs/stable/io.html
'''
import pandas as pd
df = pd.read_csv('ZILL-Z77006_MLP.csv')
print(df.head())
df.set_index('Date', inplace=True)

## Write to a new csv file
df.to_csv('new_csv2.csv')
df=pd.read_csv('new_csv2.csv')
print(df.head())

## Specifying Date as the Index.
df=pd.read_csv('new_csv2.csv', index_col=0)

## Note Index is NOT a column!!
df.columns = ['Austin_HPI']
#print(df.head)
#print(df.head())

df.to_csv('new_csv3.csv')

## If you don't want headers::
df.to_csv('new_csv4.csv', header=False)

df = pd.read_csv('new_csv4.csv', names=['Date', 'Austin_HPI'],
    index_col=0)
print(df.head())

## Make and write to a .json file and format
df.to_json('new_json.json')
## Make and write to html (html table); Pretty cool.
df.to_html('example.html')

df = pd.read_csv('new_csv4.csv', names=['Date', 'Austin_HPI'])
print(df.head())

df.rename(columns={'Austin_HPI':'77006_HPI'}, inplace=True)
print(df.head())
```

3.3 Building dataset, p4

```python
'''
www.youtube.com/watch?v=3GpvWlVinf0&index=4&list=PLQVvvaa0QuDc-3szzjeP6N6b0aDrrKyL-
pythonprogramming.net/dataset-data-analysis-python-pandas-tutorial/?completed=/input-output-

Also::
https://www.quandl.com/tools/python
http://help.quandl.com/article/205-how-do-i-download-a-dataset-using-python

'''
import quandl
import pandas as pd

# Not necessary, do this so you do not show my API key
api_key = open('quandlapikey.txt','r').read()

#df = quandl.get("FMAC/HPI_TX", authtoken=api_key)
df = quandl.get("FMAC/HPI_TX")
print(df.head())


fiddy_states =
    pd.read_html('https://simple.wikipedia.org/wiki/List_of_U.S._states')
## this is gonna be a whole **list** of DataFrames
print(fiddy_states)

## This is a DataFrames
print(fiddy_states[0])

## And now we want column zero... is a DataFrames
print(fiddy_states[0][0])

## Of column zero, we want everything from row one onward...
for abbv in fiddy_states[0][0][1:]:
    print("FMAC/HPI_"+str(abbv))
```

3.4 Concate_Append_dfs_p5

```
’’’
https://www.youtube.com/watch?v=ylIRlTFt_Fk&index=5&list=PLQVvvaa0QuDc-3szzjeP6N6b0aDrrKyL-
https://pythonprogramming.net/concatenate-append-data-analysis-python-pandas-tutorial/?comp
’’’
import pandas as pd

## Note, df1 and df3 have the same index, but not the same columns
## df2 and df3 have different indexes (and different columns).
df1 = pd.DataFrame({’HPI’:[80,85,88,85],
                   ’Int_rate’:[2, 3, 2, 2],
                   ’US_GDP_Thousands’:[50, 55, 65, 55]},
                  index = [2001, 2002, 2003, 2004])

df2 = pd.DataFrame({’HPI’:[80,85,88,85],
                   ’Int_rate’:[2, 3, 2, 2],
                   ’US_GDP_Thousands’:[50, 55, 65, 55]},
                  index = [2005, 2006, 2007, 2008])

df3 = pd.DataFrame({’HPI’:[80,85,88,85],
                   ’Int_rate’:[2, 3, 2, 2],
                   ’Low_tier_HPI’:[50, 52, 50, 53]},
                  index = [2001, 2002, 2003, 2004])

## Fine, since they have the same columns; becomes a single
    DataFrame
concat = pd.concat([df1,df2])

## Try with df3, and get some NaNs
concat = pd.concat([df1,df2, df3])
print(concat)

df4 = df1.append(df2)
print(df4)
## Note, DataFrames are not really meant to be updated.
## Meant to be manipulated, analysed, just not really changed...
#df5 = df1.append(df3)

## Doesn’t really help...
s = pd.Series([80,2,50])
df6 = df1.append(s, ignore_index=True)

s = pd.Series([80,2,50], index=[’HPI’, ’Int_rate’,
    ’US_GDP_Thousands’])
df6 = df1.append(s, ignore_index=True)
print(df6)
## 4   80        2              50, is the series s
```

3.5 Joining_Merging_dfs_p6.py

```
'''
https://www.youtube.com/watch?v=XMjSGGej9y8&list=PLQVvvaa0QuDc-3szzjeP6N6b0aDrrKyL-&index=6

https://pythonprogramming.net/join-merge-data-analysis-python-pandas-tutorial/?completed=/c
'''
import pandas as pd

df1 = pd.DataFrame({'HPI':[80,85,88,85],
                  'Int_rate':[2, 3, 2, 2],
                  'US_GDP_Thousands':[50, 55, 65, 55]},
                  index = [2001, 2002, 2003, 2004])

df2 = pd.DataFrame({'HPI':[80,85,88,85],
                  'Int_rate':[2, 3, 2, 2],
                  'US_GDP_Thousands':[50, 55, 65, 55]},
                 index = [2005, 2006, 2007, 2008])

df3 = pd.DataFrame({'HPI':[80,85,88,85],
                  'Unemployment':[7, 8, 9, 6],
                  'Low_tier_HPI':[50, 52, 50, 53]},
                 index = [2001, 2002, 2003, 2004])

## Merging
#print(pd.merge(df1,df2, on='HPI'))
## Gives duplicated data...
##print(pd.merge(df1,df2, on=['HPI']))

#print(pd.merge(df1,df2, on=['HPI', 'Int_rate']))

## Joining
df1.set_index('HPI', inplace=True)
df3.set_index('HPI', inplace=True)

joined = df1.join(df3)
#print(joined)
## Still got some data duplication


## Merging...
df1 = pd.DataFrame({
                  'Int_rate':[2, 3, 2, 2],
                  'US_GDP_Thousands':[50, 55, 65, 55],
                  'Year':[2001, 2002, 2003, 2004]
                  })

df3 = pd.DataFrame({
                  'Unemployment':[7, 8, 9, 6],
                  'Low_tier_HPI':[50, 52, 50, 53],
                  'Year':[2001, 2003, 2004, 2005]})

merged = pd.merge(df1,df3, on='Year')
## Only picks out Years in common:: 2001, 2003, 2004
merged.set_index('Year', inplace=True)
#print(merged)

## Four choices for the databases/merging::
## Left, Right, Outer, Inner.
##
## merged = pd.merge(Left, Right)
#merged = pd.merge(df1,df3, on='Year', how='left')
#merged = pd.merge(df1,df3, on='Year', how='right')
#merged.set_index('Year', inplace=True)
#print(merged)

## Outer is a Union of the keys.
merged = pd.merge(df1,df3, on='Year', how='outer')
merged.set_index('Year', inplace=True)
print(merged)

## Inner is the default
merged = pd.merge(df1,df3, on='Year', how='inner')
merged.set_index('Year', inplace=True)
print(merged)

## In summary::
## With Join, index is honoured.
## With Merge (and Cat and Append) index really doesn't come into
    it.
## The key to Merge is to merge on a common column (name)
```

3.6 Pickling_p7.py

```
'''
www.youtube.com/watch?v=WkIW0YLoQEE&list=PLQVvvaa0QuDc-3szzjeP6N6b0aDrrKyL-&index=7
pythonprogramming.net/pickle-data-analysis-python-pandas-tutorial/?completed=/join-merge-da

Key Notes::
pythonprogramming.net/community/143/Problem%20(and%20solution)%20to%20a%20pandas%20join%20i
stackoverflow.com/questions/26645515/pandas-join-issue-columns-overlap-but-no-suffix-specif
stackoverflow.com/questions/26027877/join-two-dataframes-on-one-key-column-error-columns-ov
'''
import quandl
import pandas as pd
import pickle

api_key='xTrUFEADji37fXJVMPxM' ## my API key for quandl.com

def state_list():
    fiddy_states =
        pd.read_html('https://simple.wikipedia.org/wiki/List_of_U.S._states')
    return fiddy_states[0][0][1:]

def grab_initial_state_data():
    states = state_list()
    main_df = pd.DataFrame()

    for abbv in states:
        query = "FMAC/HPI_"+str(abbv)
        print(query)

        df = quandl.get(query, authtoken=api_key)
        df.columns = [abbv] ### This is the fix ###

        if main_df.empty:
            main_df = df
        else:
            main_df = main_df.join(df)
            #main_df = main_df.join(df, lsuffix=abbv) ### Could also
                do this ###

    print(main_df.head())

    pickle_out = open('fiddy_states.pickle','wb') ##write bytes
    pickle.dump(main_df, pickle_out)
    pickle_out.close()

#grab_initial_state_data()
pickle_in = open('fiddy_states.pickle', 'rb')
HPI_data = pickle.load(pickle_in)
```

```
#print(HPI_data)

## Pandas has it’s own Pickling methodolgy
## Apparently it’s faster on Really Big data sets
## It’s also shorter code::

HPI_data.to_pickle('pickle.pickle')
HPI_data2 = pd.read_pickle('pickle.pickle')
print(HPI_data2)
```

3.7 PercentChange_Correlation_p8.py

```
’’’
www.youtube.com/watch?v=P90mCSsGE1c&index=8&list=PLQVvvaa0QuDc-3szzjeP6N6b0aDrrKyL-
pythonprogramming.net/percent-change-correlation-data-analysis-python-pandas-tutorial/?compl
’’’
import quandl
import pandas as pd
import pickle
import matplotlib.pyplot as plt
from matplotlib import style
style.use(’fivethirtyeight’)

api_key=’xTrUFEADji37fXJVMPxM’

def state_list():
    fiddy_states =
        pd.read_html(’https://simple.wikipedia.org/wiki/List_of_U.S._states’)
    return fiddy_states[0][0][1:]

def grab_initial_state_data():
    states = state_list()
    main_df = pd.DataFrame()

    for abbv in states:
        query = "FMAC/HPI_"+str(abbv)
        print(query)

        df = quandl.get(query, authtoken=api_key)
        print(df.head())
        df.columns = [abbv] ### This is the fix ###

        ## doing some manipulation on the DataFrame
        #df = df.pct_change()
        df[abbv] = (df[abbv] - df[abbv][0]) / df[abbv][0] *100

        if main_df.empty:
            main_df = df
        else:
            main_df = main_df.join(df)
            #eammain_df = main_df.join(df, lsuffix=abbv) ### Could
                also do this ###

    print(main_df.head())

    pickle_out = open(’fiddy_states3.pickle’,’wb’) ##write bytes
    pickle.dump(main_df, pickle_out)
    pickle_out.close()
    ## Everything above here is in the function
```

```
def HPI_Benchmark():
    ## US-wide HPI
    df_us = quandl.get("FMAC/HPI_USA", authtoken=api_key)
    ## Can do a df.head() to see what the columns are named
    print(df_us.head())
    #df_us.columns = ["United States"] ## This doesn't help
    #print(df_us[Value], df_us[Value][0])

    #df_us["United States"] = (df_us["United States"] -
        df_us["United States"][0]) / df_us["United States"][0] *100
        ## Is given in the video, but doesn't work
    df_us["Value"] = (df_us["Value"] - df_us["Value"][0]) /
        df_us["Value"][0] *100

    return df_us ## critical line, duh!!!


## Running this generates the new fiddy_*.pickle file
#grab_initial_state_data()


fig = plt.figure()
ax1 = plt.subplot2grid((1,1), (0,0))

#HPI_data = pd.read_pickle('pickle.pickle')
HPI_data = pd.read_pickle('fiddy_states3.pickle')

##  Modifying columns (and modifying columns within Pandas)
#HPI_data['TX2'] = HPI_data['TX']*2
#print(HPI_data[['TX', 'TX2']])

## Run the function..
benchmark = HPI_Benchmark()


#HPI_data.plot(ax=ax1, linewidth=1)
HPI_data.plot(ax=ax1, linewidth=2)
## for colors, b is blue, k is black
benchmark.plot(color='k',ax=ax1, linewidth=6, linestyle='--')

#HPI_data.plot()
plt.legend().remove()
plt.show()

## Let's create a correlation table. WOOT WOOT!!!
HPI_State_Correlation = HPI_data.corr()
```

```
## Noting HPI_State_Correlation is a dataframe...
##In [9]: type(HPI_State_Correlation)
##Out[9]: pandas.core.frame.DataFrame

print(HPI_State_Correlation)

print(HPI_State_Correlation.min)
## prints the 50x50... (??)
print(HPI_State_Correlation.min())
## prints the min of each State...
print(HPI_State_Correlation.min()[0])
## gives the min for AL
print(HPI_State_Correlation.min()[1])
## gives the min for AK etc.
type(HPI_State_Correlation.min()[1])
## is a numpy.float64

## Describe the detail, some top level-nos and stats...
print(HPI_State_Correlation.describe())
```

3.8 p9.py

3.9 p10.py

3.10 p11.py

4 For WISE W4 Analysis’

```
import numpy
import matplotlib.pyplot as plt
import re
import pandas as pd

# The data directory PATH
data_dir = "/cos_pc19a_npr/data/WISE/W4/"

# The data directory FILENAME
data_file="WISE_W4_W4SNRge3_W4MPROlt4.0_nohdr.tbl"

data_in = data_dir+data_file

cols = [’designation’, ’ra’, ’dec’, ’sigra’, ’sigdec’, ’w1mpro’,
    ’w1sigmpro’, ’w1snr’, ’w2mpro’, ’w2sigmpro’, ’w2snr’, ’w3mpro’,
    ’w3sigmpro’, ’w3snr’, ’w4mpro’, ’w4sigmpro’, ’w4snr’, ’w4rchi2’]

## If using a file *WITH* a header
# df = pd.read_table(data_in, header=54)

df = pd.read_table(data_in)
```

5 Other Useful Stuff

This is useful:
http://pandas.pydata.org/pandas-docs/version/0.15.2/10min.html
.

If youre using IPython, tab completion for column names (as well as public attributes) is automatically enabled. Heres a subset of the attributes that will be completed: In [10]: df.$< TAB >$...

```
In [24]: type(df)
Out[24]: pandas.core.frame.DataFrame

In [25]: type(df.values)
Out[25]: numpy.ndarray

In [26]: type(df.values[0])
Out[26]: numpy.ndarray

In [27]: type(df.values[0][0])
Out[27]: str
```

http://pandas.pydata.org/pandas-docs/version/0.18.1/basics.html

Okay, taking time-out to do this:
http://synesthesiam.com/posts/an-introduction-to-pandas.html

5.1 “Counting Time Zones with Pandas”

From the book and website:
https://github.com/wesm/pydata-book and “Python for Data Analysis” by Wes McKinney, published by O’Reilly Media.

```
%matplotlib inline
from __future__ import division
from numpy.random import randn
from pandas import DataFrame, Series

import os
import json
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4)

path = 'ch02/usagov_bitly_data2012-03-16-1331923249.txt'
```

```
lines = open(path).readlines()
records = [json.loads(line) for line in lines]

frame = DataFrame(records)

frame

frame.info()

frame.info
```

5.2 “Counting Time Zones with Pandas”

```
In [30]: df = pd.DataFrame(data, columns=['ra', 'dec'])
In [31]: type(data)
In [32]: type(ra)
Out[32]: pandas.core.series.Series
```

5.3 Ecosystem Visualization

http://pandas.pydata.org/pandas-docs/stable/ecosystem.html#ecosystem-visualization

5.4 Pandas and Histograms

```
df = pd.read_csv(data_path, names=cols)

type(df)
#pandas.core.frame.DataFrame

df.plot('dur','sbytes')
#<matplotlib.axes._subplots.AxesSubplot at 0x122df9588>

df.plot.hexbin(x='dur', y='sbytes')
#<matplotlib.axes._subplots.AxesSubplot at 0x122df9b70>

df.plot.hexbin(x='dur', y='sbytes',xscale='log',yscale='log',
    gridsize=15 )
# <matplotlib.axes._subplots.AxesSubplot at 0x122ddd240>

df.hist('dur', log=True, bins=200, bottom=0.1)
# array([[<matplotlib.axes._subplots.AxesSubplot object at
    0x11f56f5c0>]], dtype=object)

df.plot.hist('dur', log=True, bins=200, bottom=0.1)
```

```
#<matplotlib.axes._subplots.AxesSubplot at 0x1295a59e8>
```