This post will be a quickie detailing a rather annoying…finding about the pandas package in Python.

For those not in the know, I’ve been taking some Python courses, trying to port my R finance skills into Python, because R seems to have fallen out of favor in the world of finance. So, I’m trying to get my Python skills going, hopefully sooner rather than later.

However, for those that think Python is all that and a bag of chips, I hope to be able to disabuse people of that.

First and foremost, as far as actual accessible coursework goes on using Python, just a quick review of courses I’ve seen so far (at least as far as DataCamp goes):

The R/Finance courses are of…reasonable quality, actually. I know for a fact that I’ve used Ross Bennett’s PortfolioAnalytics course teachings in a professional consulting manner before, quantstrat is used in industry, and I was explicitly told that my course is now used as a University of Washington Master’s in Computational Finance prerequisite.

In contrast, DataCamp’s Python for Finance courses have not particularly impressed me. While a course in basic time series manipulation is alright, I suppose, there is one course that just uses finance as an intro to numpy. There’s another course that tries to apply machine learning methodology to finance by measuring the performance of prediction algorithms with R-squareds, and saying it’s good when the R-squared values go from negative to zero, without saying anything of more reasonable financial measures, such as Sharpe Ratio, drawdown, and so on and so forth. There are also a couple of courses on the usual risk management/covariance/VaR/drawdown/etc. concepts that so many reading this blog are familiar with. The most interesting python for finance course I found there.

However, rather than take multiple Python courses not particularly focused on quant finance, I’d rather redirect any reader to just \*one\*, that covers all the concepts found in, well, just about all of the DataCamp finance courses–and more–in its first two (of four) chapters that I’m self-pacing right now.

However, if people think that Python is as good as R as far as finance goes, well…so far, the going hasn’t been easy. Namely, I’ve found that working on finance in R is much easier than in Python thanks to R’s fantastic libraries written by Brian Peterson, Josh Ulrich, Jeff Ryan, and the rest of the R/Finance crew (I wonder if I’m part of it considering I taught a course like they did).

In any case, I’ve been trying to replicate the endpoints function from R in Python, because I always use it to do subsetting for asset allocation, and because I think that being able to jump between yearly, quarterly, monthly, and even daily indices to account for timing luck–(EG if you rebalance a portfolio quarterly on Mar/Jun/Sep/Dec, does it have a similar performance to a portfolio rebalanced Jan/Apr/Jul/Oct, or how does a portfolio perform depending on the day of month it’s rebalanced, and so on)–is something fairly important that should be included in the functionality of any comprehensively-built asset allocation package.

Moving on, here’s an edge case in Python’s Pandas package, regarding how Python sees weeks. That is, I dub it–an edgy panda. Basically, imagine a panda in a leather vest with a mohawk. The issue is that in some cases, the very end of one year is seen as the start of a next one, and thus the week count is seen as 1 rather than 52 or 53, which makes finding the last given day of a week not exactly work in some cases.

So, here’s some Python code to get our usual Adaptive Asset Allocation universe.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pandas\_datareader import data

import datetime as dt

from datetime import datetime

tickers = ["SPY", "VGK", "EWJ", "EEM", "VNQ", "RWX", "IEF", "TLT", "DBC", "GLD"]

# We would like all available data from 01/01/2000 until 12/31/2016.

start\_date = '1990-01-01'

end\_date = dt.datetime.today().strftime('%Y-%m-%d')

# Uses pandas\_reader.data.DataReader to load the desired data. As simple as that.

adj\_prices = []

for ticker in tickers:

tickerData = data.DataReader(ticker, 'yahoo', start\_date)

adj\_etf = tickerData.loc[:,'Adj Close']

adj\_prices.append(adj\_etf)

adj\_prices = pd.concat(adj\_prices, axis = 1)

adj\_prices.columns = tickers

adj\_prices = adj\_prices.dropna()

rets = adj\_prices.pct\_change().dropna()

df = rets

Anyhow, here’s something I found interesting, when trying to port over R’s endpoints function. Namely, in that while looking for a way to get the monthly endpoints, I found the following line on StackOverflow:

tmp = df.reset\_index().groupby([df.index.year,df.index.month],as\_index=False).last().set\_index('Date')

Which gives the following ouptut:

tmp.head()

Out[59]:

SPY VGK EWJ ... TLT DBC GLD

Date ...

2006-12-29 -0.004149 -0.003509 0.001409 ... -0.000791 0.004085 0.004928

2007-01-31 0.006723 0.005958 -0.004175 ... 0.008408 0.010531 0.009499

2007-02-28 0.010251 0.010942 -0.001353 ... -0.004528 0.015304 0.016358

2007-03-30 0.000211 0.001836 -0.006817 ... -0.001923 -0.014752 0.001371

2007-04-30 -0.008293 -0.003852 -0.007644 ... 0.010475 -0.008915 -0.006957

So far, so good. Right? Well, here’s an edgy panda that pops up when I try to narrow the case down to weeks. Why? Because endpoints in R has that functionality, so for the sake of meticulousness, I simply decided to change up the line from monthly to weekly. Here’s \*that\* input and output.

tmp = df.reset\_index().groupby([df.index.year, df.index.week],as\_index=False).last().set\_index('Date')

tmp.head()

Out[61]:

SPY VGK EWJ ... TLT DBC GLD

Date ...

2006-12-22 -0.006143 -0.002531 0.003551 ... -0.007660 0.007736 0.004399

2006-12-29 -0.004149 -0.003509 0.001409 ... -0.000791 0.004085 0.004928

2007-12-31 -0.007400 -0.010449 0.002262 ... 0.006055 0.001269 -0.006506

2007-01-12 0.007598 0.005913 0.012978 ... -0.004635 0.023400 0.025400

2007-01-19 0.001964 0.010903 0.007097 ... -0.002720 0.015038 0.011886

[5 rows x 10 columns]

Notice something funny? Instead of 2007-01-07, we get 2007-12-31. I even asked some people that use Python as their bread and butter (of which, hopefully, I will be one of soon) what was going on, and after some back and forth, it was found that the [ISO standard](https://en.wikipedia.org/wiki/ISO_week_date) has some weird edge cases relating to the final week of some years, and that the output is, apparently, correct, in that 2007-12-31 is apparently the first week of 2008 according to some ISO standard. Generally, when dealing with such edge cases in pandas (hence, edgy panda!), I look for another work-around. Thanks to help from Dr. Vaidyanathan, I got that workaround with the following input and output.

tmp = pd.Series(df.index,index=df.index).resample('W').max()

tmp.head(6)

Out[62]:

Date

2006-12-24 2006-12-22

2006-12-31 2006-12-29

2007-01-07 2007-01-05

2007-01-14 2007-01-12

2007-01-21 2007-01-19

2007-01-28 2007-01-26

Freq: W-SUN, Name: Date, dtype: datetime64[ns]

Now, \*that\* looks far more reasonable. With this, we can write a proper endpoints function.

def endpoints(df, on = "M", offset = 0):

"""

Returns index of endpoints of a time series analogous to R's endpoints

function.

Takes in:

df -- a dataframe/series with a date index

on -- a string specifying frequency of endpoints

(E.G. "M" for months, "Q" for quarters, and so on)

offset -- to offset by a specified index on the original data

(E.G. if the data is daily resolution, offset of 1 offsets by a day)

This is to allow for timing luck analysis. Thank Corey Hoffstein.

"""

# to allow for familiarity with R

# "months" becomes "M" for resampling

if len(on) > 3:

on = on[0].capitalize()

# get index dates of formal endpoints

ep\_dates = pd.Series(df.index, index = df.index).resample(on).max()

# get the integer indices of dates that are the endpoints

date\_idx = np.where(df.index.isin(ep\_dates))

# append zero and last day to match R's endpoints function

# remember, Python is indexed at 0, not 1

date\_idx = np.insert(date\_idx, 0, 0)

date\_idx = np.append(date\_idx, df.shape[0]-1)

if offset != 0:

date\_idx = date\_idx + offset

date\_idx[date\_idx < 0] = 0

date\_idx[date\_idx > df.shape[0]-1] = df.shape[0]-1

out = np.unique(date\_idx)

return out

Essentially, the function takes in 3 arguments: first, your basic data frame (or series–which is essentially just a time-indexed data frame in Python to my understanding).

Next, it takes the “on” argument, which can take either a string such as “months”, or just a one-letter term for immediate use with Python’s resample function (I forget all the abbreviations, but I do know that there’s W, M, Q, and Y for weekly, monthly, quarterly, and yearly), which the function will convert a longer string into. That way, for those coming from R, this function will be backwards compatible.

Lastly, because Corey Hoffstein makes a big deal about it and I respect his accomplishments, the offset argument, which offsets the endpoints by the amount specified, at the frequency of the original data. That is, if you take quarterly endpoints using daily frequency data, the function won’t read your mind and offset the quarterly endpoints by a month, which \*is\* functionality that probably should be \*somewhere\*, but currently exists neither in R nor in Python, at least not in the public sphere, so I suppose I’ll have to write it…eventually.

Anyway, here’s how the function works (now in Python!) using the data in this post:

endpoints(rets, on = "weeks")[0:20]

Out[98]:

array([ 0, 2, 6, 9, 14, 18, 23, 28, 33, 38, 42, 47, 52, 57, 62, 67, 71,

76, 81, 86], dtype=int64)

endpoints(rets, on = "weeks", offset = 2)[0:20]

Out[99]:

array([ 2, 4, 8, 11, 16, 20, 25, 30, 35, 40, 44, 49, 54, 59, 64, 69, 73,

78, 83, 88], dtype=int64)

endpoints(rets, on = "months")

Out[100]:

array([ 0, 6, 26, 45, 67, 87, 109, 130, 151, 174, 193,

216, 237, 257, 278, 298, 318, 340, 361, 382, 404, 425,

446, 469, 488, 510, 530, 549, 571, 592, 612, 634, 656,

677, 698, 720, 740, 762, 781, 800, 823, 844, 864, 886,

907, 929, 950, 971, 992, 1014, 1034, 1053, 1076, 1096, 1117,

1139, 1159, 1182, 1203, 1224, 1245, 1266, 1286, 1306, 1328, 1348,

1370, 1391, 1412, 1435, 1454, 1475, 1496, 1516, 1537, 1556, 1576,

1598, 1620, 1640, 1662, 1684, 1704, 1727, 1747, 1768, 1789, 1808,

1829, 1850, 1871, 1892, 1914, 1935, 1956, 1979, 1998, 2020, 2040,

2059, 2081, 2102, 2122, 2144, 2166, 2187, 2208, 2230, 2250, 2272,

2291, 2311, 2333, 2354, 2375, 2397, 2417, 2440, 2461, 2482, 2503,

2524, 2544, 2563, 2586, 2605, 2627, 2649, 2669, 2692, 2712, 2734,

2755, 2775, 2796, 2815, 2836, 2857, 2879, 2900, 2921, 2944, 2963,

2986, 3007, 3026, 3047, 3066, 3087, 3108, 3130, 3150, 3172, 3194,

3214, 3237, 3257, 3263], dtype=int64)

endpoints(rets, on = "months", offset = 10)

Out[101]:

array([ 10, 16, 36, 55, 77, 97, 119, 140, 161, 184, 203,

226, 247, 267, 288, 308, 328, 350, 371, 392, 414, 435,

456, 479, 498, 520, 540, 559, 581, 602, 622, 644, 666,

687, 708, 730, 750, 772, 791, 810, 833, 854, 874, 896,

917, 939, 960, 981, 1002, 1024, 1044, 1063, 1086, 1106, 1127,

1149, 1169, 1192, 1213, 1234, 1255, 1276, 1296, 1316, 1338, 1358,

1380, 1401, 1422, 1445, 1464, 1485, 1506, 1526, 1547, 1566, 1586,

1608, 1630, 1650, 1672, 1694, 1714, 1737, 1757, 1778, 1799, 1818,

1839, 1860, 1881, 1902, 1924, 1945, 1966, 1989, 2008, 2030, 2050,

2069, 2091, 2112, 2132, 2154, 2176, 2197, 2218, 2240, 2260, 2282,

2301, 2321, 2343, 2364, 2385, 2407, 2427, 2450, 2471, 2492, 2513,

2534, 2554, 2573, 2596, 2615, 2637, 2659, 2679, 2702, 2722, 2744,

2765, 2785, 2806, 2825, 2846, 2867, 2889, 2910, 2931, 2954, 2973,

2996, 3017, 3036, 3057, 3076, 3097, 3118, 3140, 3160, 3182, 3204,

3224, 3247, 3263], dtype=int64)

endpoints(rets, on = "quarters")

Out[102]:

array([ 0, 6, 67, 130, 193, 257, 318, 382, 446, 510, 571,

634, 698, 762, 823, 886, 950, 1014, 1076, 1139, 1203, 1266,

1328, 1391, 1454, 1516, 1576, 1640, 1704, 1768, 1829, 1892, 1956,

2020, 2081, 2144, 2208, 2272, 2333, 2397, 2461, 2524, 2586, 2649,

2712, 2775, 2836, 2900, 2963, 3026, 3087, 3150, 3214, 3263],

dtype=int64)

endpoints(rets, on = "quarters", offset = 10)

Out[103]:

array([ 10, 16, 77, 140, 203, 267, 328, 392, 456, 520, 581,

644, 708, 772, 833, 896, 960, 1024, 1086, 1149, 1213, 1276,

1338, 1401, 1464, 1526, 1586, 1650, 1714, 1778, 1839, 1902, 1966,

2030, 2091, 2154, 2218, 2282, 2343, 2407, 2471, 2534, 2596, 2659,

2722, 2785, 2846, 2910, 2973, 3036, 3097, 3160, 3224, 3263],

dtype=int64)

So, that’s that. Endpoints, in Python. Eventually, I’ll try and port over Return.portfolio and charts.PerformanceSummary as well in the future.

Thanks for reading.