Data Science with Python

Why Python

1. Easy to learn
2. Not just statistics language but data acquisition, cleaning, databases, high performance computing and modeling
3. Strong data science libraries eg Scipy ecosystem

Python pandas toolkit

Advanced querying and manipulation. Basic statistical analysis with numpy and scipy

Good data science brings skepticism, experimentation, simulation and replication.

Python is high level language, interpreted directly compiled into machine code. Good for where exploration is better, dynamically typed language. Set and modify the variables. Python interpreter is stateful.

Static typing in python – strings

Tuples – immutable, mixed types, unpacking in python

Lists – mutable, mixed types, + sign concatenates lists, \* operation repeats elements in the lists, all strings are lists of characters, Slicing in python

Dictionaries – iterate with items function

%precision 2

With open(‘file.csv’) as csvfile:

List(csv.DictReader(csvfile))

Data science with python continued

Jan 1st 1970 epic time

Import datetime as dt

Import time as tm

Tm.time()

Dtnow = dt.datetime.fromtimestamp(tm.time())

dtnow.year, dtnow.month , how , minute, second

dt.timedelta(100)

Object oriented python

class Person:

dept = “ssdsdf”

def set\_name(self,new\_name):

## do something

Implications of oops in python

Dont have private and protected members

No need for explicit constructers in python

Map function is basis for functional programming in python

map(function, iterable), it helps for lazy evaluation and so does not allow till we try to look at the value

Advanced python lambda and list comprehensions

Lambda are anonymous functions and they don’t have name

my\_function = lambda a,b,c : a+b

Very useful for data cleaning tasks

for person in people:

    print(split\_title\_and\_name(person) == (lambda x: x.split()[0] + ' ' + x.split()[-1])(person))

#option 2

list(map(split\_title\_and\_name, people)) == list(map(lambda person: person.split()[0] + ' ' + person.split()[-1], people))

List comprehensions

numpy for matrices

Bumpy arange, resize, np.linespace,eye,diag,ones,repeat,vstack,hstack

+,-,\*\*,\*,dot function,dtype,astype,sum,max,min,mean,std,argmax(),argmin(),

Indexing and slicing

R2 = r[:3,:3]

If we make any changes to the values in r2 then automatically it will be reflected in the corresponding section of r as well

len(), range(), enumerate functions to iterate over the array

Zip to iterate over both arrays in together

Week2

Pd.Series? shares the documentation

List to pandas series

Pandas stores series in a typed array using numpy library

Numpy nan is not none

Python dictionary to pandas series

Query by numeric index use iloc or directly give it index position

Query by index value use loc

Use vectorization instead of looping which numpy library provides like sum function on an iterable item

Very good example of differentiating

%%timeit –n 100 will run 100 loops of code in the cell and tells the best time

Broadcasting like changing every number by 2

Pandas series s+=2 will directly increment value of all the elements by 2

Append method in pandas series is awesome

Dataframe is conceptually a 2d series object

Column projections

Chaining will create copy on dataframe

Df.loc[:][‘col1’]

Instead use like below

Df.loc[:,[‘col1’,’col2’]]

Del to delete column

Week 3

Apply merge and join on DataFrames

print(pd.merge(products, invoices, left\_index=True, right\_on='Product ID'))

Pandas idioms

(df.where(df['SUMLEV']==50)

.dropna()

.set\_index(['STNAME','CTYNAME'])

.rename(columns={'ESTIMATESBASE2010': 'Estimates Base 2010'}))

print(df.drop(df[df['Quantity'] == 0].index).rename(columns={'Weight': 'Weight (oz.)'}))

applymap

apply heavily used instead of applymap

Employ slicing and indexing on DataFrames

Analyze data with groupby and understand categorical variables

%%timeit -n 10

for group, frame in df.groupby('STNAME'):

avg = np.average(frame['CENSUS2010POP'])

print(df.groupby('Category').apply(lambda df,a,b: sum(df[a] \* df[b]), 'Weight (oz.)', 'Quantity'))

data scaling

s = pd.Series(['Low', 'Low', 'High', 'Medium', 'Low', 'High', 'Low'])

s.astype('category', categories=['Low', 'Medium', 'High'], ordered=True)

get\_dummies function in pandas to get 0/1 categorical columns

pandas function cut for binning especially in classficiation based ml algorithms

s = pd.Series([168, 180, 174, 190, 170, 185, 179, 181, 175, 169, 182, 177, 180, 171])

pd.cut(s, 3)

# You can also add labels for the sizes [Small < Medium < Large].

pd.cut(s, 3, labels=['Small', 'Medium', 'Large'])

Pivot Tables

print(pd.pivot\_table(Bikes, index=['Manufacturer','Bike Type']))

handling datetime

pd.to\_datetime('4.7.12', dayfirst=True)

pd.Timestamp('9/3/2016')-pd.Timestamp('9/1/2016')

dates = pd.date\_range('10-01-2016', periods=9, freq='2W-SUN')

if datetime is the index

df.index.weekday\_name will return weekday names

df['2016-12']

df['2016-12':]

import matplotlib.pyplot as plt

%matplotlib inline

df.plot()

time series resampling in pandas

<http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.resample.html>

Week 4

Distributions: -

Coin flipping a binomial distribution, evenly weighted, 2 possible outcomes, discrete

x = np.random.binomial(20, .5, 10000)

print((x>=15).mean())

Uniform distribution

Value of the observation and probability of the observation – if it is flat then uniform

Normal/Gausian/ bell curve

Mean – measure of central tendency

Std – measure of variability – covers approx. 68% around the mean

Expected value – mean of lets say 3 flip of coins 1,2,6 = 1+2+6/3 = 3

Import scipy.stats as stats

Stats.kurtosis

Stats.skew()

Chi square distribution

Degrees of freedom – related to no of samples to take

Bimodal distributions – multiple peaks

Gaussian mixture models

P –hacking

Bonferroni correction

Hold out sets

Investigation pre-registration

For instance, in social sciences research, a value of 0.05 or 0.01 is often used, which indicates a tolerance for a probability of between 5% and 1% of chance. In a physics experiment where the conditions are much more controlled and thus, the burden of proof is much higher, you might expect to see alpha levels of 10 to the negative 5 or 100,000th of a percentage.

When a data scientist runs many tests in this way, it's called p-hacking or dredging and it's a serious methodological issue. P-hacking results in spurious correlations instead of generalizable results. There are a couple of different ways you can deal with p-hacking. The first is called the Bonferroni correction. In this case, you simply tighten your alpha value, the threshold of significance, based on the number of tests you're running. So if you choose 0.05 with 1 test, and you want to run 3 tests, you reduce alpha by multiplying 0.05 by one-third to get a new value of 0.017. I personally find this approach to be very conservative. Another option is to hold out some of your data for testing to see how generalizable your result is. In this case, we might take half of our data for each of the two DataFrames, run our t-test with that, form specific hypothesis based on the result of these tests, then run very limited tests on the rest of the data.

This method is actually heavily used in machine learning when building predictive models, where it's called cross fold validation and you'll learn more about this in third course in this specialization.