Functions

* **Append**: Var.append(3) - add data to a list
* **Concatenate lists**: + [1,2] + [3,4]
* **Dates**: epoch time: January 1, 1970
  + dtnow = dt.datetime.fromtimestamp(tm.time()

dtnow

< datetime.datetime(2016,10,17,10,28,40,500572)

dtnow.year, dtnow.month, dtnow.day, dtnow.hour

* + delta = dt.timedelta(days = 100)

today = dt.date.today()

today - delta

* **Def**: define - used to define a function
  + def add\_numbers(x,y):

Return x + y

* + - def add\_numbers (x,y,z=none) :

if (z==none):

return x + y

else:

return x + y +z

\* used to define a function when don’t know if two or three vars to be added

* **float:** converts to a decimal value; for example change a value in a dict to a float
* **for**: used to create a loop
  + x = [1,’a’,y,’b’]

for item in x:

print(item)

* **in:** gives a Boolean if an element in a list
  + 1 in [1,2,3]
* **iterating:** 
  + for a list: for letter in x:

print(x[letter])

* + for key pair: for name in x.values():

print(name)

* + unpacking: for name, email in x.items():

print(name)

print(email)

* lambda: used to define a quick temporary function
  + my\_function = lambda a, b, c, : a + b

my\_function(1,2,3)

* **len()**: identifies the length of a variable or data set
* **map():** used to look at multiple lists to apply a function
  + store1 = [5, 2, 7, 6]

store2 = [3,5,7,10]

cheapest = map(min, store1, store2) – must use a with to open map and see contents

* **read\_csv:** 
  + df=pd.read.csv(‘olympics.csv’, index\_col=0, skiprows=1) : ignore first row of data; but start with first collumn
* **repeat lists**: [1]\*3
* **set:** used to create a set of keys
  + cylinders= set (d[‘cyl’] for d in mpg)
* **slicing**: cutting a list
  + x = ‘this is a string’

print(x[0:2]) – prints only first and second item

* x[-1] – gives back end of string
* x[:3] – implicitly implies start from first
* **splitting:** splitting a string by some key such as a space
  + firstname= “William Thomas Draco Clarke”.split(‘ ‘)[0]

print(firstname)

* **time**:
  + import datetime as dt
  + import time as tm
    - tm.time() : gets current time since epoch

**type():** to identify the type of variable

* **unpacking: x = (‘john’, ‘williams’, ‘data analyst’)**

**fname, lname, function = x**

**lname**

**> ‘williams’**

* **while:** used in loops for something of undetermined length
  + i=0

while (i != len(x)):

print(x[i])

i = i + 1

**types**

* **tuple:** x=(1,’a’,2,’b’)
* **list:** x = [1,’a’,y,’b’]
* **string:** ‘this is the day of days’
* **dictionary:** x ={‘a’: ‘John’, ‘b’: ‘Mark’} to call x[‘a’]
  + **identify keys:** mpg[0].keys() : ideintifies columns in a csv converted to dict
* **list comprehension:**
  + my\_list = [for number in range(1, 1000) if number %2==0]

my\_list

* **python series**: cross between a list and a dictionary; uses an index and values; the values has a label

**Libraries**

* **Numpy:** import Numpy as np ; for array and data frames**.**
  + Mylist = [1,2,3]

x = np.array(mylist)

* + y = np.array([1,2,3])
  + z = np.array([[1,2,3,], [7,8,9]])
  + check by z.shape
  + n = np.arange(0,30,2) : crate an array from 0 to 30 by 2
  + n = np.reshape(3,5) : 3 rows and 5 columns, creates new array
  + o = np.linspace(0,4,9): automatically splits up space into equal parts
  + o.resize(3,3): change dimensions in place, does not actually change o
  + np.ones((3,2)): an array of one’s or use zeros
  + np.ones(((3,2), dtype=np.int)
  + np.array([1,2,3]\*3) : to repeat list
  + np.repreat([1,2,3],3) : repeat each element
  + array operations
    - x + y, x \* y, x\*\*2, x.T (transpose)
    - x.astype(‘f’) : change to decimals instead of integers
    - x.sum()
    - x.max()
    - x.mean()
    - x.std()
    - x.argmax(): to find index of maximum
  + indexing and slicing
    - np.arange(13)\*\*2 : create an array of squared values form 0 to 12
    - x[-4:] : slice of last four elements
    - x[:-4] : removes last 4
    - x[-4]: gives 4th element from last
    - x[-5::-2] : starting fifth from end and counting backwards by 2
    - x[2,2] : to get 3rd row and 3rd column
    - x[3, 3:6] 4th row columns 4,5,6
    - x[:2,:-1] : first two rows and all columns except last
    - x[-1,::2] ; every second element from last row
    - x[x>30] for elemtns 31 and higher
    - x[x>30]=30 : assigns all elements greater than 30 to 30
    - x2 = x[:3, :3] : changes x too
    - x2[:] = 0 : sets everything to zero, changes x too
    - x\_copy=x.copy() : copies an array thus does not change original
  + iterating over arrays
    - x = np.random.randint(0, 10, (4,3)) : create random integers in 4 x 3
    - for row in x: print(row)
    - for i in range(len(x)): print(x[i])
    - for i, row in enumerate(x): print(‘row’, i, ‘is’, row)
    - y = x\*\*2
      * for i,j in zip (x, x2): print(i, ‘+’, j, ‘=’, i+j) : iterate over two arrays
* Pandas Series
  + Import pandas as pd
    - pd.series? : gives documentation
    - animals = [‘dog’, ‘cat’, ‘pig’]

pd.Series(animals)

* + - sports = {‘Archery’: ’Bhutan’, ‘Golf’: ’Scotland’}

s = pd.Series(sports)

s.index

* + - s = pd.Series{[‘Archery’, ‘Golf’], index = [‘Bhutan’, ‘Scotland’]) : to create index explicitly
    - s.iloc[1] : uses default numeric index to find output
    - s.loc[‘Archery’]: uses assigned index for output
    - note: use square brackets to query attributes and parenthesis for methods
    - s = pd.Series([100.00, 120.00, 101.00, 3.00])

import numpy as np

total = np.sum(s)

print(total)

* + - %tab –gives list of magic functions
    - a = pd.Series(np.random.randomint(0,1000,10000))

a.head() = gives first five entries

len(a) – gvies length genreated

* + - %%timeit –n 100 : checks average time for a run

summary = 0

for item in s:

summary += item

* + - a += 2 : broadcast change of entire data set
    - cricket\_countries =pd.Series([‘Australia’, ‘Barbados’],

index = [‘Cricket’, ‘Cricket’])

all\_countries = sport.append(cricket\_countries) : creates new series with combination of both

* Data Frames
  + purchase\_1 = pd.Series({‘Name’:‘Chris’, ‘Item Purchased’:’Dog Food’, ‘Cost’: 22.50})

purchase\_2 = pd.Series({‘Name’:’Kevyn’, ‘Item Purchased’:’Kitty Litter’,

‘Cost’:2.50})

purchase\_3 = pd.Series({‘Name’:’Vinod’, ‘Item Purchased’:’Bird Seed’,

‘Cost’: 5.00})

df=pd.DataFrame([purchase\_1, purchase\_2, purchase\_3], index =

[‘Store 1’, ‘Store 1’, ‘Store 2’])

df.head()

* + queries:
    - df.loc[‘Store 2’] : store 2 info only
    - df.loc[‘Store 1’, ‘Cost’] : store 1 costs only
    - df.T.loc[‘Cost’] : to get cost column or just df.[‘cost’]
    - df.loc[:,[’Name’, ‘Cost’]] : get all rows and only name and cost
    - df.drop[‘Store 1’] : removes store 1 (copy of data frame with rows removed)
    - copy\_df = df.copy()

copy\_df = copy\_df.drop[‘Store 1’] : to get a permanaet copy

df.drop[‘Store 1’, TRUE] : drops in place, place 1 in third argument to drop columns

* + - del copy\_df[‘Name’] : drops column but not as a copy of df
    - df[‘Location’] = none : adds a column for location with none in it
    - df[‘Cost’] \*= .8 : boradcasts a 20% reduction in all costs
    - costs = df.[‘Cost’]

costs += 2 : **changes original data frame**

* + df.columns : to list column names in dataframe
  + for col in df.columns:

if col[:2] == ‘01’:

df.rename(columns={col:’Gold’+col[4:]}, inplace = TRUE) : finds columns with a 01 in first two spots, renames it Gold then adds remaining name back on; inplace changes original dataframe

* + further queries:
    - df[‘Gold’]>0 : identify rows and Boolean values that have a column value 1 or more
    - only\_gold=df.where([‘Gold’]>0) : create a dataframe of above
    - only\_gold[‘Gold’].count() : counts gold
    - only\_gold=only\_gold.dropna() : drops nan’s
    - only\_gold=df[df[‘Gold’]>0] : better way than where statement; removes nan’s too.
    - Total\_gold =df[(df[‘Gold’] > 0 | df[‘Gold.1’]>0])
    - Only\_winter=df[(df[‘Gold,1’] > 0) & ( df[‘Gold’]==0)]
    - df[‘Name’][df[‘Cost’]>3] : get names for all costs greater than 3
  + indexing
    - df[‘country’] = df.index : assigns current index to df column

df=df.set\_index(‘Gold’) : assings a new index from a column

df=df.reset\_index() : to reset index with default numbering index

* census data
  + - df[‘SUMLEV’].unique() : gives unique levels in column sumlev
    - df=df[df[‘SUMLEV’]==50] : creates new df with only counties
    - columns\_to\_keep = [‘STNAME’, CTYNAME’, BIRTHS210’]

df = df[columns\_to\_keep] : create df with only columns we want

* + - df=df.set\_index([‘STNAME’, ‘CTYNAME’]) : index by state and county

df.index.name=[‘state’,’county’] : rename index

df=df.loc[‘Michigan’, ‘Washtenaw County’] : to query

df=df.loc[ [(‘Michigan’, ‘Washtenaw County’), (‘Michigan’, ‘Wayne County’)] ] : to query two

* video data
  + df.fillna? : gives parameters for fill
  + df = df.set\_index(‘time’) : to set time as index

df=df.sort\_index() : to sort on index , in this case index not unique, need a multi-level index

df=df.reset\_index()

df=.set\_index([‘time’,’user’]) : create multi-level index

next can use forward fill and backward fill as necessary in piece meal fashion

* store data as before
* df[‘Date’] = [“December 1’, ‘January 1’] : to add new column, must be equal to number of rows, include none if missing
* df[‘Delivered’] = TRUE : adds to all rows
* adf=df.reset\_index()

adf[‘Date’] = pd.Series({0:’December 1’, 2:’mid-day’}) : another approach that does not require and entry for each row

* merging data frames
  + staff\_df=pd.DataFrame([{‘Name’:’Kelly’, ‘Role’:”Director’}])

staff\_df= staff\_df.set\_index(‘Name’)

student\_df=pd.DataFrame([{‘Name’:’James’, ‘School’:’Business’}])

student\_df=student\_df.set\_index(‘Name’)

pd.merge(staff\_df, student\_df, how=’outer’, left\_index=TRUE, right\_index=TRUE) : must have same index, using left data index and right data index

pd.merge(staff\_df, student\_df, how=’inner’, left\_index=TRUE, right\_index=TRUE): to get only common data

pd.merge(staff\_df, student\_df, how=’left’, left\_index=TRUE, right\_index=TRUE):all staff and their student records

add left\_on = ‘Name’, right\_on= ‘Name’ : to join on column instead of index

* pandas idioms
  + method chaining
    - (df.where(df[‘SUMLEV’]==50)

.dropna()

.set\_index([‘STNAME’, ‘CTYNAME’])

.rename(columns={‘ESTMATEBASE2010’:’Estimate Base 2010’}))

* + - df = df[df[‘SUMLEV’]==50]

df.set\_index([‘STNAME’, ‘CTYNAME’], inplace =TRUE)

df.rename(columns={‘ESTMATEBASE2010’:’Estimate Base 2010’}): equivalent to abve but not as pandorable

* + - print(df.drop(df[df[‘Quantity’]==0].index).rename(columns={‘Quantity’:’Quantity (oz.)’})) ; drop entries equal to zero and rename
  + apply
    - def min\_max(row):

data = row[[‘POPEST2010’, ‘POPEST2011’]]

return pd.Series({‘min’:np.min(data), ‘max’:np.max(data)})

df.apply(min\_max, acis = 1) : 1 is all rows

* + - def min\_max(row):

data = row[[‘POPEST2010’, ‘POPEST2011’]]

row[‘max’]=np.max(data)

row[‘min’]=np.min(data)

return row

df.apply(min\_max, acis = 1) : creates a data frame not a series

* + - rows=[‘POPEST2010’, ‘POPEST2011’]

df.apply(lambda x: np.max(x[rows]), axis=1) : using lambda to create above

* group by function
  + %%time it –n 10

for state in df[‘STNAME’].unique():

avg = np.average(df.where(df[‘STNAME’] ==state).dropna()[‘CENSUS2010POP’]

print(‘Counties in state’ + state + ‘ have an average population of ‘ + str(avg)) : an inefficient method

* + for group, frame in df.groupby(‘STNAME’):

avg=np.average(frame[‘CENSUS2010POP’])

print(‘Counties in state ‘ + group + ‘ have an average population of ‘ + str(avg))

* + df.groupby(‘STNAME’).agg((‘CENSUS2010POP’: na.average)): should this have braces?
  + (df.set\_index(‘STNAME’).groupby(level=0)[‘CENSUS2010POP’].agg({‘avg’: np.average, ‘sum’:np.sum})) : sense only one column applies both agg function to it
  + def totalweight (df, w, q):

return( sum(df[w]\*df[q])

print(df.groupby(‘Category’).apply(totalweight, ‘Weight(oz.)’, ‘Quantity’)

* scales
  + df= pd.DataFrame([‘A+’, ‘A’, ‘A-‘, ‘B+’], index = [‘excellent’, ‘excellent’, ‘excellent’, ‘good’])

df.rename(columns={0: ‘Grades’}, inplace = TRUE)

df[‘Grades’].astype(‘category’).head() : assign a categorical data type

* + df[‘Grades’].astype(‘category’, categories = [‘B+’, ‘A-‘, ‘A’, ‘A+’], ordered=TRUE) : to make category ordinal
  + df=pd.read\_csv(‘census.csv’)

df = df[df[‘SUMLEV’]==50]

df=df.set\_index(‘STNAME’).groupby(level=0)[‘CENSUS2010POP’].agg({‘avg’: np.average})

pd.cut(df[‘avg’], 10) : create ten bins for categorization

* + s = pd.Series([168, 180, 174, 190])

pd.cut(s,3,labels=[‘small’, ‘medium’, ‘large’]): to cut series into 3 and add labels

* pivot tables
  + df=pd.read\_csv(‘cars.csv’)

df.head()

df.pivot\_table(values= ‘ (kW)’, index = ‘YEAR’, columns = ‘Make’, aggfunc=np.mean, margins = True)

* + pd.pivot\_table(Bikes, index = [‘Manufacturer’, ‘Bike Type’])
* date functionality
  + import pandas as pd

import numpy as np

pd.Timestamp(‘9/1/2016 10:05AM’)

pd.Period(‘1/2016’)

* t1 = pd.Series(list(‘abc’), [pd.Timestamp(‘2016-09-01’), pd.Timestamp(‘2016-09-02’), pd.Timestampt(2016-09-3’)])

type(t1.index) : shows it’s a datetime index

* converting to datetime
* d1 = [‘2 June 2013’, ‘Aug 29, 2014’, ‘7/12/16’]

t3 = pd.DataFrame(np.random.randint(10,100, (3,2)), index=d1, columns = list(‘ab’))

t3.index = pd.to\_datetime(t3.index)

* Timedeltas:
* pd.Timestamp(‘9/3/2016’)- pd.Timestamp(‘9/1/2016’)
* pd.Timestamp(‘9/3/2016’)+ pd.Timedelta(‘12D 3H’)
* dates = pd.date\_range(’10-01-2016’, periods=9, freq=’2W-SUN’): bi weekly every Sunday

df=pd.DataFrame({‘Count 1’: 100 + np.random.randomint(-5,10,9).cumsum(), ‘Count 2’: 120 +np.random.randint(-5,10,9)}, index=dates)

df.index.weekday\_name : check the day of week

df.diff() : to see the difference betwee each dates value

df.resample(‘M’).mean() : mean coutn is for each month ‘M’

df[‘2017’] : to find values from particular year

df[‘2016-12’:] : want values form December 2016 onward

df.asfreq(‘W’, method =’ffill’) : add weekly data using forward fill

* to plot:
  + import matplotlib.pyplot as plt

%matplotlib inline

df.plot()

* statisitcs
  + distributions
    - import pandas as pd

import numpy as np

np.random.binomial(1,.5): args(number of times to run, the chance of a 0)

np.random.binomial(1000, .5)/1000 : get avg close to .5

* x = np.random.binomial(20,.5,1000) : 1000 simulations

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

* chance\_of\_tornado = .01

tornado\_events=np.random.binomial(1,chance\_of\_tornado, 1000000)

tow\_days\_in\_a\_row=0

for j in range(1, len(tornado\_events)-1):

if tornado\_events[j] ==1 and tornado\_events[j-1]==1:

two\_days\_in\_a\_row +=1

print(‘{} tornadoes back to back in{} years’.format(tow\_days\_in\_a\_row, 1000000/365))

* stdv:

dist=np.random.normal(.75, size =1000): 1000 samples with E(x) = .75 , default sd = 1

np.sqrt(np.sum((np.mean(dist)-dist)\*\*2/len(dist))

* built in function:

np.std(dist)

* another measure of distribution; shape of the tails (kurtosis)

import scipy.stats as stats

stats.kurtosis(dist) : negative means curve is slightly more flat than normal distritution

* check skew:

stats.skew(dist)

* chi squared dist (skewed); only one argument, degrees of freedom

chi\_2=np.radnom.chisquare(2, size =1000)

stats.skew(chi\_2)

chi\_5=np.radnom.chisquare(5, size =1000)

stats.skew(chi\_5)

import matplotlib

import matplotlib.pyplot as plt

out= plt.hist([chi\_2, chi\_5], bins = 50, histtype=’step’, label = [‘2 df’, ‘5 df’])

plt.legend(loc = ‘upper right’)

* df=pd.read\_csv(‘grades’)

df.head()

len(df)

early=df[df[‘ass\_1’]<= ’2015-12-31’]

late =df[df[‘ass\_1’]> ‘2015-12-31’]

early.mean() : to see means of all assignments

last.mean()

from scipy import stats

stats.ttest\_ind(early[‘ass\_1\_grade’], late{‘ass\_1\_grade’])

* + running multiple t tests until you find a positive is called P-hacking or Dredging; since an alpha of 5% means you will get a positive 1 out of 20 times; best preventative is to use a holdout, aka cross validation to verify results
* matplotlib architecture
  + %matplotlib notebook

import matplotlib as mpl

mpl.get\_backend()

* import matplotlib.pyplot as plt

plt.plot(3,2, ‘.’)

* from matplotlib.backend.backend\_agg import FigureCanvasAgg from matplotlib.figure import Figure : backend method

fig = Figure()

canvas = FigureCanvasAgg(fig)

ax = fig.add\_subplot(111) : add one plot

ax.plot(3,2,’.’)

canvas.print\_png(‘test.png’) : cannot plot to screen

%%html : to see the rendered image

<img src=’test.png’ />

* plt.figure()

plt.plot(3,2, ‘o’)

ax = plt.gca()

ax.axis([0,6,0,10])

add more data:

plt.figure()

plt.plot(1.5,1.5,’o’)

plt.plot(2,2,’o’)

* scatterplots
  + import numpy as np

x = np.array([1,2,3,4,5,6,7,8])

y=x

plt.figure()

plt.scattter(x,y)

* + zip functions:
    - zip\_generator = zip([1,2,3],[5,6,7])

list(zip\_generator)

x,y = zip(\*zip\_generator)

print(x)

print(y)

plt.figure()

plt.scatter(x[:2], y[:2], s=100, c = ‘red’, label = ‘tall students’) : s = size of dot

plt.scatter(x[2:], y[2:], c=’blue’, label=’short students’)

plt.xlabel(‘the number of kicks’)

plt.ylabel(‘the grade’)

plt.title(‘relationships’)

plt.legend()

plt.legend(loc=4, frameon=false, title=’Legend’) : mpves legend to lower right, removes frame, adds name

* line plots
  + import numpy as np

import pandas as pd ### for correct date time feature

linear\_data = np.array([1,2,3,4,5])

quadratic\_data = linear\_data\*\*2

plt.figure()

obs\_dates= np.arange(‘2017-01-01’, ‘2017-01-09’, dtype = ‘datetime64[D]’)

obs\_dates = list(map(pd.to\_datetime, obs\_dates)) ##3 map returns an iterator so need to convert to a list

plt.plot(obs\_dates, linear\_data, ‘-o’, quadratic\_data, ‘-o’) ####-o = lineplot

plt.plot([22,44,55], ‘—r’) #### - - r creates dashed line

plt.xlabel(‘some data’)

plt.ylabel(‘other data’)

plt.title(‘the title’)

plt.legend([‘baseline’, ‘competition’, ‘us’])

x = plt.gca().xaxis ### to rotate labels

for item in x.get\_ticklables():

item.set\_rotation(45)

plt.subplots\_adjust(bottom=.25) ### to expand bottom for labels

plt.gca().fill\_between(range(len(linear\_data)),

linear\_data, quadratic\_data,

facecolor=’blue’,

alpha=.25)

* bar charts
  + plt.figure()

xvals = range(len(linear\_data)

plt.bar(xvals, linear\_data, width = .3)

new\_vals=[] ### add second bar

for item in xvals:

new\_vals.append(item+.3)

plt.bar(new\_vals, quadratic\_data, width =.3, color =’red’)

* + plt.figure()

xvals=range(len(linear\_data))

plt.bar(xvals, linear\_data, width =.3, color=’b’)

plt.bar(xvals, quadratic\_data, width=.3, bottom=linear\_data, color=’r’) ## adds bar on top of other bar

* + import matplotlib.pyplot as plt

import numpy as np

plt.figure()

languages = [‘python’, ‘sql’, ‘java’, ‘c++’, ‘javascript’]

pos = np.arange(len(languages)

popularity = [56, 39, 34, 34, 29]

bars = plt.bar(pos, popularity, align = ‘center’, linewidth=0, color=’lightslategrey’) ### change color

bars[0].set\_color(#1F77B4) ### change python bar to contrasting color

plt.xticks(pos, languages, alpha =.8) ### the alpha softens the color

#plt.ylabel(‘% popularity’, alpha =.8) ## can deselect since bars will be directly labled

plt.title(‘top five languages \nby % popularity on stack overflow’, alpha = .8)

plt.tick\_params(top=’off’, bottom=’off’, left=’off’, right=’off’, labelleft=’off’, labelbottom=’on’)

plt.show()

for spline in plt.gca().spine.values():

spine.set\_visible(False)

for bar in bars:

plt.gca().text(bar.get\_x()+bar.get\_width()/2, bar.get\_height()-5,

(int(bar.get\_height())) + ‘%’,

ha=’center’, color=’w’, fontsize=11)

plt.show()

sample code:

**Strings**

* sales\_record = {‘price’: 3.24, ‘num\_items’: 4, ‘person’: ‘Chris’}

sales\_statement = ‘{} bought {} item(s) at a price of {} each for a total of {}’

print(sales\_statement.format(sales\_record[‘person’], slaes\_record[‘num\_items’], sales\_rrcord[‘price’], sales\_record[‘num\_items’]\*sa;es\_record[‘price’]))

**Reading Files In:**

* csv as dictionary:
  + with open(‘car.csv’) as csvfile:

mpg = list(csv.DictReader(csvfile))

mpg[:3] - read first 3 files

* load a csv:
  + import pandas as pd
  + mpg = pd.read\_csv('cars.csv', sep = ';')
* !cat Olympics.csv (may be Jupiter notebook only; try leaveing off !)

**Random Code**:

* get average of a dict value:
  + sum(float(d['CITY (Le/100 km)']) for d in mpg)/len(mpg)
* get average by cyclinder type
  + ctympgbycyl = []

for c in cylinders:

summpg = 0

cyltypecount = 0

for d in mpg:

if d[‘cyl’] == c:

summpg += float(d[‘cty’])

cyltypecount += 1

ctympgbycyl.append((c, summpg/ cyltypecount))

ctympgbycyl

* my\_list = []

for number in range(1,1000):

if number % 2 == 0:

my\_list.append(number)

my\_list