***Python Basic***

List1 = [‘mov1’,’mov2’]

So indexing

**list1[1] 🡪 mov2**

Dictionary

Dict1 = { key1 : value1 , key2 : value2}

So indexing

**dict[‘key1’] 🡪 output value1**

7 Athematic (+ - \* /) f%n = returns remainders f\*\*n = f to the power of n f//n = remove remainders

Strings print(“%s %s %s” % (‘I like’, quote, bun, eat)

Strings print(“I don’t like “, end=””) # removed newline like backspace

Strings print(‘\n’ \* 5) # print newline 5x times

Print “X”, # the comma means next print must remain same line as previous

type(thing) = str | type(30) = int | type(20.22) = float # returns the type whether it’s a string/integer/float

abs(-30.22) = 30.22 # returns the absolute value or removes negative (-) example abs(-1) = 1

round(30.88) = 30 # round off, it follows the round off rule 0.5 = 1

noel = int(1234) # converts any type to integer

noel = str(1234) # converts any type to string

factorial(4) # returns 4\*3\*2\*1=24, but has a problem if factorial(0) returns 1 which is wrong

inputlist = sorted(inputlist) # Sort LIST, works on numerical and strings 123…abc

inputlist.sort() # same effect

**t**otal = sum(numbers) # built-in function sum(xxx + xxx)

total = float(numbers) # built-in function turns numbers into a float

total = len(numbers) # built-in function counts the count in the ()

The \ character is a continuation character. The following line is considered a continuation of the current line.

**Pass**  # pass command doesn’t do anything, but useful to tell Phython that code is fine…

**\n \t # newline and new tab**

**result = str(1) + str(2) # becomes str1 and str2 combines 1 2, but below it int(result 12) become numerical 12.**

**print(int(result) + 3) # Output 15**

'dog'.upper() # Outputs DOG | same as animal.upper() if animal = ‘dog’ uppercase, lower() works too

name.capitalize(), name.title(), name.islower(), name.center(40), and name.center(40, '\*'). They are all built I functions.

Explained in <https://docs.python.org/3/library/stdtypes.html#string-methods>

while input('> ') != 'yes': # waiting for user to enter yes

\*args # arbitrary, when not knowing how many arguments, for example def adder(\*args): then instances can be adder(7, 33) or adder(1, 2, 3)

\*\*kwargs # like \*args ways of a dictionary of key-value pairs. Example def kwarg\_printer(\*\*kwargs): then instances

kwarg\_printer(var=10, other\_var='a string')

output > {'var':10, 'other\_var':'a string'}

XXXX.isdigit() 🡪 checks whether it is a digit or not, return true or false

Odd\_list = my\_list[1::2] # generate a list of odd

Advance for loop 🡪 age\_plus\_one = [age+1 for age in ages] 🡪 output [43 , 65, 21 , …. ]

Advance if else into for loop

new\_list = [1 if y==0 else 0 if y==1 else None for y in n ]

ord() function to convert characters into their ASCII values.

ord('a') 🡪 output 97

[dict['imdb'] for dict in movies] 🡪 movie is a list of dictionary, diction contained imdb, for loop pull only imdb from movie

alist.append(new\_data\_into\_alist) 🡪 append syntax

Assert area\_square(5) == 25 # for checking function purpose

try, else, except, finally, raise , except (NameError, ValueError, TypeError ) as target: statement # need to read more, for error handling

**lambda**

map(function\_to\_apply, list\_of\_inputs)

**a = [1,2,3,4]**

**b = [1,2,3,4]**

**map(lambda x,y:x+y, a,b)**

**[2,4,6,8]**

**LIST []**

Grocery\_list = [‘Juice’, ‘Tomatoes’, ‘Potatoes’, ‘bananas’]

Print(‘First item’, grocery\_list[0] #prints Juice

Grocery\_list[0] = “green juice” #Change the value from juice to green juice

Print(Grocery\_list[1:3]) # prints Tomatoes Potatoes

Other\_events = [‘wash car’, ‘pick up kids’, ‘cash check’] #new list

To\_do\_list = [Other\_events, grocery\_list] #LIST3 = [LIST1 , LIST2]

Print(To\_do\_list) # wash car’, ‘pick up kids’, ‘cash check’, Juice’, ‘Tomatoes’, ‘Potatoes’, ‘bananas’

Print(to\_do\_list[1][1]) # the 1st [1] means get from grocery\_list, 2nd [1] means Tomatoes. Hence prints Tomatoes

Grocery\_list.append(‘Onions’) # Append a new item into Grocery. Prints [‘Juice’, ‘Tomatoes’, ‘Potatoes’, ‘bananas’, ‘Onions’]

Grocery\_list.insert(1,”Pickle”) # insert into index 1, a pickle

Grocery\_list.remove(“Pickle”) # remove specific item

Grocery\_list.sort() # Sort A-Z

Grocery\_list.reverse() # reverse Sort Z-A

Del grocery\_list[4] # del item 4

To\_do\_list2 = other\_events + grocery\_list # Combine 2 LIST into 1 big LIST

Print(len(to\_do\_list2)) # Count the length of the list, returns 7

Print(max(to\_do\_list)) # Max strings means printA-Z the Z item, returns Wash Car

Print(min(to\_do\_list)) # Min strings means printA-Z the A item, returns Cash check

print l[2:9:2] # Slice LIST[start:end:steps] inside[] are index, Reverse order l[2:9:-2]

print l[:9:2] l[2::2] l[::2] # Slice LIST[::] if leaves blank, it defaults ALL numbers

letters = ['a', 'b', 'c', 'd'] # How to remove commas in a LIST?

print " ".join(letters) # replace with a space # a b c d

print "---".join(letters) # replace with --- # a---b---c---d

n = [1, 3, 5] # REMOVE vs POP vs DEL

n.pop(1) # removes index 1, Returns 3, but if print n return [1,5]

del(n[1]) or del list[index] # delete index 1, no Returns, but print n return [1, 5]

n.remove(1) # removes value 1, Returns [3, 5]

list\_a = [3, 9, 17, 15, 19] #zip will create pairs of elements when passed two lists, and will stop at the end of the shorter list.

list\_b = [2, 4, 8, 10, 30, 40, 50, 60, 70, 80, 90] # zip can do 3x or more LIST

for a, b in zip(list\_a, list\_b): # this loop stops at 4, bcos list\_a is shorter

**Range()Passing a range into a function**

range(6) # range(stop) return [0, 1, 2, 3, 4, 5]

range(1, 6) # range(start, stop) return [1, 2, 3, 4, 5]

range(1, 6, 3) # range(start, stop, step) return [1, 4]

**to do stats range using ptp**

#### Range

sample = [3, 75, 98, 2, 10, 3, 14, 99, 44, 25, 31, 100, 356, 4, 23, 55, 327, 64, 6, 20]

# With numpy:

sample.sort()

print sample

last\_minus\_first = sample [-1] - sample [0]

print last\_minus\_first

# Now implement this in pure Python

np.ptp(sample)

**TUPLE () is like LIST but cannot change after creation, append() and pop() do not exist.**

Pi\_tuple = (3,1,4,1,5,9) # create a Tuple

New\_tuple = list(pi\_tuple) # convert tuple to a LIST

New\_list = tuple(pi\_tuple) # convert LIST to a TUPLE

Len(pi\_tuple) max(pi\_tuple) min(pi\_tuple) # same as LIST, you can len, max, min a tuple

**DICTIONARY {}** cannot join dic like LIST using + | DICT is unordered

Dict = {‘Keys’ : ‘Value’} # Dict = {Keys : Value} are unordered. Dict&Keys can be “string”

Print(Dict[‘Keys’]) # This returns Value of the key

Del Dict[‘Keys’] # delete the KEY and its Value

Dict[‘KEY’] = ‘new Value’ # insert a new KEY with new Value into old DICT

Print(len(DICT)) # Counts number of KEYS

Print(DICT.get(“KEY”)) # gets the Value

For I in DICT: # I = KEY , loop I increments the KEY

print DICT.items() # .items means print KEY and Value together

Print DICT.keys() # print the whole list of keys in this DICT

Print DICT.values() # Print the Values

choices = ['pizza', 'pasta', 'salad', 'nachos'] # Your choices are:

print 'Your choices are:' #0 pizza

for **index, item** in enumerate(choices): #1 pasta

print index, item #2 salad

**Set {}** Curly bracket but not dict, bcos only contain one element, and unordered, removes duplicate

Whateverset = {‘xxx’, ‘ccc’}

Sets contain no duplicate elements. {1, 2, 2} is equivalent to {1, 2}.

**Conditional**

if else elif [ ==, !=, >, >=, <=, < ] , logical operator [ and, or, not ]

**for / else loop** # for prints a newline everytime. ‘else’ statement is executed after the ‘for’ completes properly without break. If break, else won’t executed

for x in range(0,10): print(x) # print the value 0 to 10 in range. Range goes into x.

for x in [2,4,6,8,10]: print (x) # print 2 4 6 8 10

num\_list = [[1,2,3], [10,20,30],[100,200,300]] # loop a LIST

for x in range(0,3): # loop a LIST

for y in range(0,3): # loop a LIST

print(num\_list[x],[y]) # print 1 2 3 10 20 30 100 200 300

evens\_to\_50 = [i for i in range(51) if i % 2 == 0] # Advance for loop to create LIST with condition

print evens\_to\_50 # Outputs [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50]

doubles = [x \* 2 for x in range(1, 6)] # => [2, 4, 6, 8, 10] one to five. If you want those numbers doubled:

doubles\_by\_3 = [x \* 2 for x in range(1, 6) if (x \* 2) % 3 == 0] # => [6] only wanted the doubled numbers that are evenly divisible by three:

even\_squares = [x \*\* 2 for x in range(1, 12) if x % 2 == 0] # => [4, 16, 36, 64, 100]

c = ['C' for x in range(5) if x < 3] # Advance for loop printing strings

cubes\_by\_four = [x \*\* 3 for x in range(1, 11) if ((x \*\* 3) % 4) == 0] # => [8, 64, 216, 512, 1000] cubes of the numbers 1 through 10 only if the cube is evenly divisible by four.

**import random**

**While / else loops**

from random import randint # there are 2 ways to call a random module

coin = randint(1, 6) # method 1

Random\_num = random.randrange(0,100) # method 2, generate a random range numbers from 0 – 100

While(random\_num !=15): # cycle thru random no, while not equal to 15

Print(random\_num) # Print the number

Random\_num = random.randrange(0,100) # generate a new random number

# 2 8 87 26 37 … until 15, it stops

I = 0; # Create a iterator

While(I <= 20): # while

If(I%2 == 0): # checks for even number

Print(I) # then print that even number

Elif(I == 9): # condition

Break # END the while loop

Else:

I = I + 1 # condition

Continue # continue the while loop

I = I + 1 # Increment

**NumPy & SciPy**

NumPy 🡺 vector and matrix creation and array math operations. Only can do simple array generation

Scipy 🡺 can do more complex work like multiply arrays or sum .sum

import numpy as np # dun have .mode

import scipy

from scipy import stats

**Using NumPy for Vectors (one dimension).**Vectors are a sequence of numbers

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

a + b

Output[]:

array([5, 7, 9])

my\_array = np.array([[2,1],[7,4],[5,2],[6,3]])

my\_array.shape # Return the number of rows and columns. .shape to get the dimensions of a NumPy array 🡺 (4L, 2L)

sample = [3, 75, 98, 2, 10, 3, 14, 99, 44, 25, 31, 100, 356, 4, 23, 55, 327, 64, 6, 20]

mean = np.mean(sample)

median = np.median(sample)

mode = stats.mode(sample)

print "mean / median:", mean, median 🡺 mean / median: 67.95 28.0

**Discrete random variables :** is countable, example coin flips or the number of aces from a deck of cards. Each outcome is called the probability mass function, or pmf.

**Continuous random variables:** not countable. example, the temperature at noon, is infinite

np.random.rand(7) # generate 7 random numbers. 🡺 [ 0.21508189, 0.55978152, 0.64720631, 0.57905971, 0.46305698, 0.70073415, 0.73717498]

**Using NumPy for Matrices (two dimensions).**

np.random.rand(3,2) # generate ramdom matrix

Out[]:

array([ [ 0.14022471, 0.96360618], #random

[ 0.37601032, 0.25528411], #random

[ 0.49313049, 0.94909878]]) #random

**Using NumPy for Multi-dimensional arrays (three or more dimensions).**

mean = 0

std = 10

series\_length = 20

np.random.normal(mean, std, series\_length) # this prints out Normal distribution

normal\_series = np.random.normal(mean, std, series\_length)

print (normal\_series)

Output []

[-12.1819094 -1.10817893 -13.27377051 20.28193636 10.8787821

-16.70313467 -11.69218729 2.77141727 -2.58325263 2.68374526

-15.37812735 11.06065337 -7.19609026 -7.56049486 -4.77832489

-4.22084877 -9.8486844 11.3143184 -15.36866494 2.83687985]

**Pandas** package. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

import pandas

for creating and manipulating tabular data that are represented as DataFrames(Excel), with each column stored as a Series object. Objects that hold data in row-and-column tables. A DataFrame holds one table, 2D (like an excel sheet). A Series is a single-column DataFrame (1x dimensional data).

Pandas as dictionary: column name as the key and the column content as the value. dict[[key1, key2, key3]] SAME dataframe[[col1, col2, col3]]

Pandas accessing rows like LIST 🡺 df[0:3] # selects rows 0, 1 and 2

Pandas accessing specific rows via index number 🡺 df.iloc[index] 🡺 Output the whole row datas from index 1

Example of series: has the data and index

My\_series = pd.series ([10,20,30], index = [2012, 2013, 2014])

* 2012 10
* 2013 20
* 2014 30

Example of dataframe: 2D data

my\_dataframe = pd.DataFrame({‘col\_1’:[2012,2013,2014,2015], col\_2’:[1,2,3,4,5]})

df = pd.DataFrame(np.array([[.25,10,600], [-.9,40,200], [.4,70,800], [.8,50,300]), columns=["A","B","C"])

print df

Out[]:

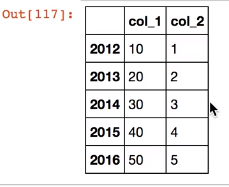
A B C

0 0.25 10.0 600.0

1 -0.90 40.0 200.0

2 0.40 70.0 800.0

3 0.80 50.0 300.0



pd.read\_csv('data\_file.csv'): Retrieves data from a local CSV file or from the web.

pd.read\_table('data\_file.xml'): Retrieves data from other delimited data text files.

pd.read\_json('data\_file.json'): Reads in a JSON object and converts it to a DataFrame.

Attributes are locations.

Methods are functions to run for deep analyze or manipulate the DataFrame.

**Example attribute for a DataFrame named df:**

df.shape: Returns an array of the number of the rows and columns in the df DataFrame.

df.columns: Returns a list of the column names.

df.index: Returns a list of the row names.

**Example methods for a DataFrame named df:**

df.head(n): Returns the top rows of the df DataFrame.

df.tail(n): where n is which row

df.dtypes # returns the type of its columns

type(data) 🡺 check the object's type

pd.sort\_values(): Returns a copy of the DataFrame sorted by one or more columns.

pd.describe(): Returns the summary statistics of all numeric columns in the DataFrame.

Example of dataframe : using Boolean into Subset

print df

Out[]:

A B C

0 0.25 10.0 600.0

1 -0.90 40.0 200.0

2 0.40 70.0 800.0

3 0.80 50.0 300.0

**df[df['A'] > 0] # here’s the magic** **For example, the following output includes the contents of all rows in which the condition we're testing is True**

**# note if you code only ‘df['A'] > 0’ will only output True and False, but if df[dfxxxx], then shows below**

Out[]:

A B C

0 0.25 10.0 600.0

2 0.40 70.0 800.0

3 0.80 50.0 300.0

**Pandas Using Method Parameters to Extract Subsets** using inplace statement to replace a DataFrame's contents with a subset of altered values.

df.sort\_values(by="A", inplace=True)

**Pandas data cleaning involves removing unknown, null, NA, or NaN values. Using .dropna**

df2.dropna(inplace=True)

**Pandas Selecting Individual Cells or Cell Ranges**

**df.iloc[row, column] # where df is the name of dataframe**

df.iloc[0,] # Locates the cells in row 0.

df.iloc[0] # Locates the cells in row 0 (an all-column search is implied).

df.iloc[:5] # Locates rows 0–4 in df.

df.iloc[:5,] # Locates rows 0–4 in df.

df.iloc[:5,2:6] # Locates cells that only appear in both rows 0–4 and columns 2–5.

new\_dataframe1 = original\_df.iloc[:, :6] 🡺 Splitting DataFrames then assigned to new dataframe

Pandas Adding rows using **.append() ,** but **.join()** works too. Need to read up

Pandas Adding columns using **.concat() 🡺 pd.concat([df1, df2], axis = 1) 🡺 column axis 1 (col 1)**

df["Unique\_visitors"] = [45000, 8000, 15000, 7000, 3000] == Added new columns called Unique visitors and its data in index order 1,2,3,….

**Pandas .join() methods 🡺 to perform index JOINs or single-column JOINs. Noel: It joins by matching Index names, in this example “Country”**

|  |  |
| --- | --- |
| df1  Out[]:  Continent Country Pageviews  0 Europe UK 100000  1 Europe DE 20000  2 Africa Kenya 40000  3 Africa Morocco 20000  4 Africa Chad 10000 | capitals  Out[]:  Country Capital  0 UK London  1 DE Germany  2 Kenya Nairobi  3 Chad N'Djamena  4 Italy Rome |

Join on Country and the indices:

**df1.join(capitals.set\_index('Country'), on='Country') 🡺 here’s the magic**

Out[]:

Continent Country Pageviews Capital

0 Europe UK 100000 London

1 Europe DE 20000 Germany

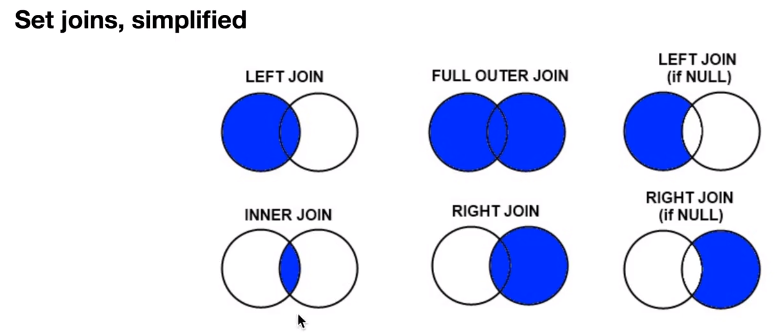
2 Africa Kenya 40000 Nairobi

3 Africa Morocco 20000 NaN

4 Africa Chad 10000 N'Djamena

**.join() explains in SQL**

Df.join(df\_geocode,how = ‘inner’)**.head() 🡺 .head is optional if u want to show a bit only**



|  |  |
| --- | --- |
| #load file1.csv as a DataFrame and print the result to screen  data = pandas.read\_csv("file1.csv")  print("data:")  print(data)  print() | data:  col1 col2 col3  0 2.5 hello True  1 3.2 goodbye False  2 7.0 where True  3 1.0 are True  4 3.3 you False  5 7.5 going True |
| #select only those rows where col1 is greater than 3 and print to screen  print("col1 > 3:")  print(data[data["col1"] > 3])  print() | col1 > 3:  col1 col2 col3  1 3.2 goodbye False  2 7.0 where True  4 3.3 you False  5 7.5 going True |
| #select only those rows where col3 is False and print to screen  print("col3 is False:")  print(data[data["col3"] == False])  print() | col3 is False:  col1 col2 col3  1 3.2 goodbye False  4 3.3 you False |
| #load file2.csv as a DataFrame object and print it to screen  data2 = pandas.read\_csv("file2.csv")  print("data2:")  print(data2)  print() | data2:  col2 col4  0 hello 38  1 where 45  2 are 20  3 you 17  4 going 12  5 today 70 |
| #left join data and data2 on col2, keeping all of data's values and print the resulting DataFrame to the screen  left\_join\_data = data.merge(data2,on="col2",how="left")  print("data left joined with data2:")  print(left\_join\_data)  print() | data left joined with data2:  col1 col2 col3 col4  0 2.5 hello True 38.0  1 3.2 goodbye False NaN  2 7.0 where True 45.0  3 1.0 are True 20.0  4 3.3 you False 17.0  5 7.5 going True 12.0 |
| #right join data and data2 on col2, keeping all of data2's values  right\_join\_data = data.merge(data2,on="col2",how="right")  print("data right joined with data2:")  print(right\_join\_data)  print() | data right joined with data2:  col1 col2 col3 col4  0 2.5 hello True 38  1 7.0 where True 45  2 1.0 are True 20  3 3.3 you False 17  4 7.5 going True 12  5 NaN today NaN 70 |
| #inner join data and data2  inner\_join\_data = data.merge(data2,on="col2",how="inner")  print("data inner joined with data2:")  print(inner\_join\_data) | data inner joined with data2:  col1 col2 col3 col4  0 2.5 hello True 38  1 7.0 where True 45  2 1.0 are True 20  3 3.3 you False 17  4 7.5 going True 12 |
| #outer join data and data2  outer\_join\_data = data.merge(data2,on="col2",how="outer")  print("data outer joined with data2:")  print(outer\_join\_data) | data outer joined with data2:  col1 col2 col3 col4  0 2.5 hello True 38.0  1 3.2 goodbye False NaN  2 7.0 where True 45.0  3 1.0 are True 20.0  4 3.3 you False 17.0  5 7.5 going True 12.0  6 NaN today NaN 70.0 |

.to\_numeric() is a Pandas function that automatically changes the data type of a DataFrame column to a numeric format.

df["C"] = pd.to\_numeric(df.C) | then do df.dtypes to check

.astype() to change column A to string

df["C"] = df.C.astype('float64')

df["A"] = df.A.astype('str')

df.groupby('Continent').Pageviews.mean() 🡺 Group by the Continent data is the unique value, then get its mean from pageviews

df.groupby('Continent').Pageviews.median() 🡺 sometimes this syntax works too 🡺 df.groupby('Continent')[‘Pageviews’].median()

df.groupby('Continent').Pageviews.sum()

df.groupby('Continent').Pageviews.min()

df.groupby('Continent').Pageviews.max()

df.groupby('Column').apply(custom\_function) 🡺 using custom function, do whatever you want

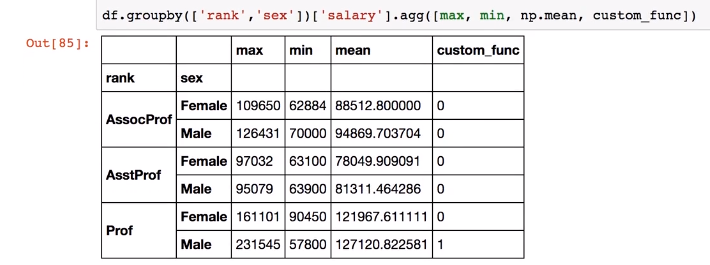
pd.pivot\_table(df, columns="Continent", aggfunc=[np.mean]) 🡺 doing pivot table on Pandas, Creating a PivotTable using mean as the aggregation

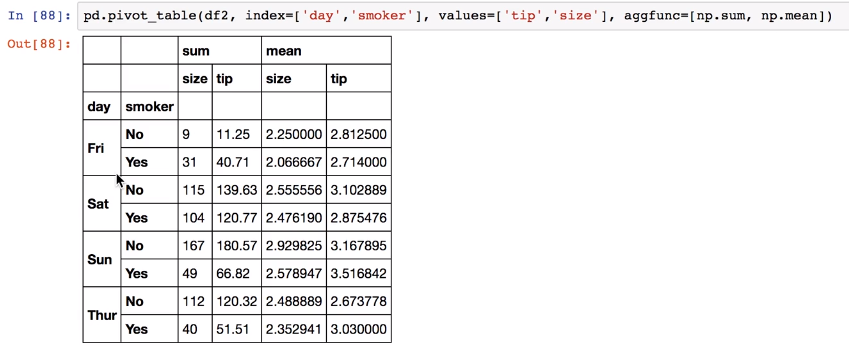
pd.pivot\_table(df, columns="Continent", aggfunc=[len]) 🡺 obtain the counts of the non-NaN rows grouped by Continent

pd.pivot\_table(df, columns="Continent", aggfunc=[sum]) 🡺 Or, obtain the sum of values for every column and Continent.

pd.pivot\_table(df, columns="Continent", aggfunc=[np.std]) 🡺 obtain the standard deviation of every column grouped by Continent.

Using .agg function





|  |  |
| --- | --- |
| #load file1.csv as a DataFrame and print the result to screen  data = pandas.read\_csv("file1.csv")  print("data:")  print(data) | data:  col1 col2 col3  0 2.5 something True  1 3.2 nothing False  2 7.0 something True  3 1.0 everything True  4 3.3 nothing False  5 7.5 something True |
| #get overall sense of data by getting numeric summary info of numeric columns using .describe()  print(data.describe())  print() | col1  count 6.000000  mean 4.083333  std 2.591846  min 1.000000  25% 2.675000  50% 3.250000  75% 6.075000  max 7.500000  () |
| #get distribution of non-numeric columns using .value\_counts() on col2 and col3  print(data.col2.value\_counts())  print()  print(data.col3.value\_counts())  print() | something 3  nothing 2  everything 1  Name: col2, dtype: int64  ()  True 4  False 2  Name: col3, dtype: int64  () |
| #get the mean of col1 grouped by col2  print("col1 mean grouped by col2:")  print(data.groupby("col2")["col1"].mean())  print() | col1 mean grouped by col2:  col2  everything 1.000000  nothing 3.250000  something 5.666667  Name: col1, dtype: float64  () |
| #get the mean of col1 grouped by col3  print("col1 mean grouped by col3:")  print(data.groupby("col3")["col1"].mean())  print() | col1 mean grouped by col3:  col3  False 3.25  True 4.50  Name: col1, dtype: float64 |
| #pivot data into a new table where col2 is the index, col3's values are the individual columns, and the average value in col1 is the value in each cell  pivoted\_data = pandas.pivot\_table(data,values="col1",index=["col2"],columns=["col3"],aggfunc="mean",fill\_value=0)  print(pivoted\_data) | col3 False True  col2  everything 0.00 1.000000  nothing 3.25 0.000000  something 0.00 5.666667 |

Pandas filtering (see they used ‘&’ instead of ‘and’, for 2x filters)

df = df[(df[‘Country’]==’BROOKLYN’) & (df[‘Employees’]>10)] 🡺 This will list out Brooklyn and Employee>10 filtered in data set.

Pandas .merge() methods 🡺

loc works on labels in the index.

ix will usually attempt to behave like loc but reverts to behaving like iloc if it can't find the specified label in the index. This is a more generalized method, but it increases the challenge of writing the Python statement correctly.

**Function**

Def addNumber(fNum, lNum):

sumNum = fNum + lNum # Noel note: sumNum do not exist outside function

return sumNum

print addnumber(1,4)) # prints 5

string = addNumber(1,4) # string will have value 5

print(sumNum) # Noel note: sumNum do not exist outside function. This will error

**lambda function / often used with** **filter(), map() and reduce().** # if you want a quick function without creating def name

lambda x: x % 3 == 0 [same as] def by\_three(x): return x % 3 == 0

my\_list = range(16) print filter(lambda x: x % 3 == 0, my\_list) # => [0, 3, 6, 9, 12, 15]

print filter(\_\_\_\_\_\_\_, \_\_\_\_\_\_\_) takes two arguments: the first is the function that tells it what to filter, and the second is the object to perform the filtering on.

my\_list = range(16) # => [0, 1,. . . 12,13,14,15]

print filter(lambda x: x % 3 == 0, my\_list) # => [0, 3, 6, 9, 12, 15]

squares = [x \*\* 2 for x in range(1, 11)] # => [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

print filter(lambda x: x >= 30 and x <= 70, squares) # => [36, 49, 64]

**BITWISE OPERATORS** # NAND AND NOT OR gate function

print 5 >> 4 # Right Shift => 0 also same as 0b111 >> 4

print 5 << 1 # Left Shift => 10

print 8 & 5 # Bitwise AND => 0

print 9 | 4 # Bitwise OR => 13 usually to change only some bit to a 1, or turn on , also same print bin(0b1110 | 0b101)

print 12 ^ 42 # Bitwise XOR => 38 usually to flip a bit

print ~88 # Bitwise NOT => -89 strange not doing NOT, it is adding one to the number and then making it negative.

print 0b110, # => 6 bcos binary 110, I can also to 0b110 + 0b100 => 11

print bin(8) # => 0b1000 convert to binary(num)

print oct(18) # => 022 convert Octal to decimal

print hex(18) # => 0x12 convert Hex to Dec

**import sys**

print(“What is your name”) # What is your name

name = sys.stdin.readline() # Derek

print(“Hello”,name) # hello Derek

**String**

Long\_string = “I’ll catch you If fall – The Floor”

Print(long\_string[0:4]) # I’ll

Print(long\_string[-5: ]) # Floor

Print(long\_string[ :-5]) # I’ll catch you If fall

Print(long\_string[ :4] + “ be there”) # I’ll be there

Print(“%c is my %s letter and my number %d number is %.5f” % ‘X’, ‘favorite’, 1, 0.14)

# X is my favorite letter and my number 1 number is 0.14000

Print(long\_string.capitalize()) # turn 1st letter to CAP, returns **I**’ll catch you If fall – The Floor

Print(long\_string.find(“Floor”)) # what is position of the Floor, returns 33

Print(long\_string.isalpha()) # return TRUE if all are alphabet? This Returns False

Print(long\_string.isalnum()) # return TRUE if all are number? This Returns False

Print(len(long\_string)) # count length of the string. Return 38

Print(long\_string.replace(“Floor”, “Ground”)) # replace a specific word

Print(long\_string.strip()) # strip away white space. This returns nothing since no white

Quote\_list = long\_string.split(“ ”)) # Separate each into a LIST

Print(Quote\_list) # [“I’ll”, “catch”, “you”, “If”, “fall”, “–“, “The”, “Floor”]

**import os**

test\_file = open(“test.txt”, “wb”) # ‘wb’ command to Create a new file

test\_file = open(“test.txt”, “ab+”) # ‘ab+’ command a new file to Read and Append to file

print(test\_file.mode) # what mode is this file? returns wb

print(test\_file.name) # what filename? Returns text.txt

test\_file.write(bytes(“Write me to file\n”, ‘UTF-8)) # Write a string into this file

test\_file.close() # Close this file

test\_file = open(“test.txt”, “r+” # Since closed, now re-open to r+ Read & Write command

test\_in\_file = test\_file.read() # Reading and put into test\_in\_file

print(text\_in\_file) # Print what was READ

os.remove(“test.txt”) # Delete file

Example i/o files from codeacdemy

my\_list = [i \*\* 2 for i in range(1, 11)]

my\_file = open("output.txt", "w")

# Add your code below!

for value in my\_list:

my\_file.write(str(value) + "\n") # writing : #=> inside output.txt has text 1 4 9 16 25 36 49 64 81 100

my\_file.close() # 🡺 Must close properly else wont work

my\_file = open("output.txt", "r") # 🡺 this is how you read from files

print my\_file.read() # 🡺 this is how you read from files

my\_file.close() # 🡺 this is how you read from files

my\_file = open("text.txt", "r")

print my\_file.readline() # 🡺 I'm the first line of the file!

print my\_file.readline() # 🡺 I'm the second line.

print my\_file.readline() # 🡺 Third line here, boss.

my\_file.close()

**Still on file I/O. if lazy to close(), python has auto mode using with / as:**

with open("text.txt", "w") as my\_file:

my\_file.write("Success!")

**Python file objects have a closed attribute which is True when the file is closed and False otherwise.**

**By checking file\_object.closed, we'll know whether our file is closed and can call close() on it if it's still open.**

f = open("bg.txt")

f.closed # False

f.close()

f.closed # True

#Example 2 about file close auto

with open("text.txt", "w") as my\_file:

my\_file.write("My Data!")

if not file.closed:

file.close()

print my\_file.closed

**Class notes:**

class CapName(INHERIT): # A Class contains NAME of the class , and the class from which the new class INHERIT in parentheses. Inside (), must be (object), dun know why.

**\_\_init\_\_()** used to initialize the objects it creates. \_\_init\_\_() always takes at least one argument, **self**, that refers to the object being created. You can think of \_\_init\_\_() as the function that "boots up" each object the class creates.

# Functions in a class is not called Function, but called Methods

class Animal(object): # """Makes cute animals."""

def \_\_init\_\_(self, name, age, is\_hungry): # For initializing our instance objects, aka, called “member variables”

#memberVariable = "initialValue"

self.name = name # Note that self is only used in the \_\_init\_\_() function definition; we don't need to pass it to

self.age = age # our instance objects.

self.is\_hungry = is\_hungry

# outside class

zebra = Animal("Jeffrey", 2, True) # zebra is also called instance of classname Animal

giraffe = Animal("Bruce", 1, False) # giraffe is also called instance of classname Animal

panda = Animal("Chad", 7, True)

print zebra.name, zebra.age, zebra.is\_hungry # => Jeffrey 2 True

print giraffe.name, giraffe.age, giraffe.is\_hungry

print panda.name, panda.age, panda.is\_hungry

Another Class example: take 2x methods

class Animal(object):

"""Makes cute animals."""

is\_alive = True

def \_\_init\_\_(self, name, age): # Method 1, with self

self.name = name

self.age = age

# Add your method here!

def description(self): # Method 2 , also can use self

print self.name

print self.age

hippo = Animal("Anderson", 36)

hippo.description() # => Anderson 36

**Anotherr class example:**

class ShoppingCart(object):

"""Creates shopping cart objects

for users of our fine website."""

items\_in\_cart = {}

def \_\_init\_\_(self, customer\_name):

self.customer\_name = customer\_name

def add\_item(self, product, price):

"""Add product to the cart."""

if not product in self.items\_in\_cart:

self.items\_in\_cart[product] = price

print product + " added."

else:

print product + " is already in the cart."

def remove\_item(self, product):

"""Remove product from the cart."""

if product in self.items\_in\_cart:

del self.items\_in\_cart[product]

print product + " removed."

else:

print product + " is not in the cart."

my\_cart = ShoppingCart("Eric")

my\_cart.add\_item("Ukelele", 10) # => Ukelele added.

**Class example about Inheritance :**

class Customer(object):

"""Produces objects that represent customers."""

def \_\_init\_\_(self, customer\_id):

self.customer\_id = customer\_id

def display\_cart(self):

print "I'm a string that stands in for the contents of your shopping cart!"

class ReturningCustomer(Customer):

"""For customers of the repeat variety."""

def display\_order\_history(self):

print "I'm a string that stands in for your order history!"

monty\_python = ReturningCustomer("ID: 12345")

monty\_python.display\_cart() # => I'm a string that stands in for the contents of your shopping cart!

monty\_python.display\_order\_history() # => I'm a string that stands in for your order history!

**Class example about Inheritance :**

class Shape(object):

"""Makes shapes!"""

def \_\_init\_\_(self, number\_of\_sides):

self.number\_of\_sides = number\_of\_sides

**# Add your Triangle class below! About inheritance**

class Triangle(Shape):

def \_\_init\_\_(self, side1, side2, side3):

self.side1 = side1

self.side2 = side2

self.side3 = side3

**# class again!!**

**Override!**

**Sometimes you'll want one class that inherits from another to not only take on the methods and attributes of its parent, but to override one or more of them.**

class Employee(object):

"""Models real-life employees!"""

def \_\_init\_\_(self, employee\_name):

self.employee\_name = employee\_name

def calculate\_wage(self, hours):

self.hours = hours

return hours \* 20.00

# Add your code below!

class PartTimeEmployee(Employee):

def calculate\_wage(self, hours):

self.hours = hours

return hours \* 12.00

**# class practice example:** **On the flip side, sometimes you'll be working with a derived class (or subclass) and realize that you've overwritten a method or attribute defined in that class' base class (also called a parent or superclass)**

class Employee(object):

"""Models real-life employees!"""

def \_\_init\_\_(self, employee\_name):

self.employee\_name = employee\_name

def calculate\_wage(self, hours):

self.hours = hours

return hours \* 20.00

# Add your code below!

class PartTimeEmployee(Employee):

def calculate\_wage(self, hours):

self.hours = hours

return hours \* 12.00

def full\_time\_wage(self, hours):

return super(PartTimeEmployee, self).calculate\_wage(hours)

milton = PartTimeEmployee('Milton')

print milton.full\_time\_wage(10) # => 200.0

**Last example about Class:**

class Triangle(object):

number\_of\_sides = 3

def \_\_init\_\_(self, angle1, angle2, angle3):

self.angle1 = angle1

self.angle2 = angle2

self.angle3 = angle3

def check\_angles(self):

if (self.angle1 + self.angle2 + self.angle3) == 180:

return True

else:

return False

class Equilateral(Triangle):

angle = 60

def \_\_init\_\_(self):

self.angle1 = self.angle

self.angle2 = self.angle

self.angle3 = self.angle

**# Advance Class about Inheritance**  See how ElectricCar inherit property from Car, see \_\_init\_\_

class Car(object):

condition = "new"

def \_\_init\_\_(self, model, color, mpg):

self.model = model

self.color = color

self.mpg = mpg

def display\_car(self):

print "This is a %s %s with %s MPG." % (self.color, self.model, str(self.mpg))

def drive\_car(self):

self.condition = "used"

class ElectricCar(Car):

def \_\_init\_\_(self, model, color, mpg, battery\_type):

self.model = model

self.color = color

self.mpg = mpg

self.battery\_type = battery\_type

def drive\_car(self):

self.condition = "like new"

my\_car = ElectricCar("DeLorean", "silver", 88, "molten salt")

print my\_car.condition

my\_car.drive\_car()

print my\_car.condition

**# New about Class \_\_repr\_\_() built-in \_\_repr\_\_() method,** **which is short for representation; by providing a return value in this method, we can tell Python how to represent an object of our class (for instance, when using a print statement).**

class Point3D(object):

def \_\_init\_\_(self, x, y, z):

self.x = x

self.y = y

self.z = z

def \_\_repr\_\_(self):

return "(%d, %d, %d)" % (self.x, self.y, self.z)

my\_point = Point3D(1, 2, 3)

print my\_point ## => Outputs (1, 2, 3)

# Class example

class Genre(object):

def \_\_init\_\_(self, band, members, albums, sold, genre):

self.members = members

self.band = band

self.albums = albums

self.sold = sold

self.genre = genre

def print\_stats(self):

return 'band: {} members: {} albums: {} sold: {} genre: {}'.format(self.band, self.members, self.albums, self.sold, self.genre)

Queen = Genre('Queen',4,15,105000000,'Rock')

print(Queen.print\_stats()) # Outputs🡺 band: Queen members: 4 albums: 15 sold: 105000000 genre: Rock

***Dictionary***

d = {"foo" : "bar"} # Dict = {Keys : Value} are unordered. Dict&Keys can be “string”

Dict[Keys] # This returns Value

lloyd = {"homework": [], "quizzes" : []} # Value empty, so do this {Keys:[]}

lloyd = {"homework": [100.0, 92.0, 98.0, 100.0]} # Value not empty, can be LIST

tyler = {

"name": "Tyler",

"homework": [0.0, 87.0, 75.0, 22.0],

"quizzes": [0.0, 75.0, 78.0],

"tests": [100.0, 100.0]

}

students = [lloyd, alice, tyler]

for x in students:

print x["name"]

print x["homework"]

print x["quizzes"]

print x["tests"]

# this prints

Tyler

[0.0, 87.0, 75.0, 22.0]

[0.0, 75.0, 78.0]

[100.0, 100.0]

===============================================================

tyler = {

"name": "Tyler",

"homework": [0.0, 87.0, 75.0, 22.0],

"quizzes": [0.0, 75.0, 78.0],

"tests": [100.0, 100.0]

}

# Add your function below!

def average(numbers):

total = sum(numbers)

total = float(total)

return total / len(numbers)

def get\_average(student):

homework = average(student["homework"])

quizzes = average(student["quizzes"])

tests = average(student["tests"])

total = homework \*.1 + quizzes \* .3 + tests \* .6

return total

# No prints but this Multiply the 3 averages by their weights and return the sum of those three. Homework is 10%, quizzes are 30% and tests are 60%

## Define a function called get\_class\_average that has one argument class\_list. You can expect class\_list to be a list containing your three students. First, make an empty list called results. For each student item in the class\_list, calculate get\_average(student) and then call results.append() with that result. Finally, return the result of calling average() with results.

def get\_class\_average(class\_list):

results = []

for student in class\_list:

student\_avg = get\_average(student)

results.append(student\_avg)

return average(results)

### Declare a list of students. Your students should be [lloyd, alice, tyler]. Finally, print out the result of calling get\_class\_average with your students list. Then, print the result of get\_letter\_grade for the class's average.

def get\_letter\_grade(score):

if score >= 90:

return "A"

elif score >=80:

return "B"

elif score >=70:

return "C"

elif score >=60:

return "D"

else:

return "F"

def get\_class\_average(class\_list):

results = []

for student in class\_list:

student\_avg = get\_average(student)

results.append(student\_avg)

return average(results)

students = [lloyd, alice, tyler]

class\_avg = get\_class\_average(students)

print class\_avg

print get\_letter\_grade(class\_avg)

**List**

a = [1, 3, 4, 7] # A List in numbers

students = [lloyd, alice, tyler] # A List in strings

for x in a: # for loop a LIST, ‘x’ gets the value of ‘a’ inside for loop

**Function**

def count\_fun(lotto)

small = count\_fun(lotto)

print small # calling a function def count\_fun()

print fizz\_count(["fizz","cat","fizz"]) #call a function in print

print get\_letter\_grade(get\_average(lloyd)) # call a function((double call another function))

**for loop**

for x in ‘string’ : #for loop in STRINGS, strings are like lists with characters

word = "Programming is fun!" #for loop in STRINGS, strings are like lists with characters

for letter in word: #for loop in STRINGS, strings are like lists with characters

for letter in "Codecademy": #for loop in STRINGS, strings are like lists with characters

print letter #for loop in STRINGS, strings are like lists with characters

#####################################################

inventory = {

'gold' : 500,

'pouch' : ['flint', 'twine', 'gemstone'],

'backpack' : ['xylophone','dagger', 'bedroll','bread loaf']

}

inventory['burlap bag'] = ['apple', 'small ruby', 'three-toed sloth']

inventory['pouch'].sort()

inventory['pocket'] = ['seashell', 'strange berry', 'lint']

inventory['backpack'].sort()

inventory['backpack'].remove('dagger')

inventory['gold'] = inventory['gold'] + 50

#####################################################

zoo\_animals = { 'Unicorn' : 'Cotton Candy House',

'Sloth' : 'Rainforest Exhibit',

'Bengal Tiger' : 'Jungle House',

'Atlantic Puffin' : 'Arctic Exhibit',

'Rockhopper Penguin' : 'Arctic Exhibit'}

del zoo\_animals['Unicorn']

animals = ["cat", "ant", "bat"]

animals.sort()

for animal in animals:

print animal

#####################################################

def distance\_from\_zero(thing): # type checks whether it’s a integer or float, but has a problem, example 7.0 is not integer due to decimal

if type(thing) == int or type(thing) == float:

return abs(thing) # returns the absolute value or removes negative (-) example abs(-1) = 1

else:

return "Nope"

#####################################################

from module import function

from math import sqrt

#####################################################

s = "Charlie"

print s[0]

# will print "C"

print s[1:4]

# will print "har"

#####################################################

**Nice functions**

#A prime number is a number that has no divisors, other than 1 or itself, 2, 3, 5, 7, 11, 13, 17, 19, 23. But 0, 1, 2 are not prime. Strange, no such module. Function below:

def is\_prime(x):

if x < 2:

return False

else:

for n in range(2, x-1):

if x % n == 0:

return False

return True

print is\_prime(53) # returns True

print is\_prime(51) # returns False bcos 51 is divisible by 3

# To reverse a text function:

def reverse(text):

word = ""

l = len(text) - 1

while l >= 0:

word = word + text[l]

l -= 1

return word

print reverse('abcd') # Outputs dcba

# To remove vowel from a sentence aeiouAEIOU , can be modified to remove any parts of a sentence, by scanning thru all

def anti\_vowel(text):

t=""

for c in text:

for i in "ieaouIEAOU":

if c==i:

c=""

else:

c=c

t=t+c

return t

print anti\_vowel("I am groot") # Outputs m grt

# Scrabble is a game where players get points by spelling words. Using dictionary to give a Value to the word by scanning thru the KEYS

score = {"a": 1, "c": 3, "b": 3, "e": 1, "d": 2, "g": 2,

"f": 4, "i": 1, "h": 4, "k": 5, "j": 8, "m": 3,

"l": 1, "o": 1, "n": 1, "q": 10, "p": 3, "s": 1,

"r": 1, "u": 1, "t": 1, "w": 4, "v": 4, "y": 4,

"x": 8, "z": 10}

def scrabble\_score(word):

word = word.lower()

v = 0

for c in word:

for i in score:

if c == i:

v += score[i]

return v

print scrabble\_score("datasciencist") # Output 18

# To replace a WORD with another WORD in a sentence

def censor(text, word):

words2list = text.split() #split text into LIST ["I", "am", "Groot"]

result = ''

count = 0

for i in words2list: # loop 5 counts in LIST

if i == word: # checking if i[1]==hack

words2list[count] = '\*\*\*\*\*' # replace with \*\*\*\*\*

count += 1

result =' '.join(words2list)

return result

print censor("this hack is wack hack", "hack") # Outputs this \*\*\*\*\* is wack \*\*\*\*\*

# To remove odd number from a LIST

def purify(lst):

result = []

for x in lst:

if x % 2 == 0:

result.append(x)

return result

print purify([1,2,3]) # should return [2]

# To remove duplicates from a LIST , but it sorted 123abc . Doesn’t work if don’t want sort

def remove\_duplicates(inputlist):

if inputlist == []:

return []

# Sort the input list from low to high

inputlist = sorted(inputlist)

# Initialize the output list, and give it the first value of the now-sorted input list

outputlist = [inputlist[0]]

# Go through the values of the sorted list and append to the output list

# ...any values that are greater than the last value of the output list

for i in inputlist:

if i > outputlist[-1]:

outputlist.append(i)

return outputlist

print remove\_duplicates(['aaa', '1', '2f', 'aaaa']) # should return [1, 2f, aaa, aaaa].

# To find median . If cannot use numpy.median(a, axis=None, out=None, overwrite\_input=False, keepdims=False)

def median(lst):

sorted\_list = sorted(lst)

if len(sorted\_list) % 2 != 0:

index = len(sorted\_list)//2

return sorted\_list[index]

elif len(sorted\_list) % 2 == 0:

index\_1 = len(sorted\_list)/2 - 1

index\_2 = len(sorted\_list)/2

mean = (sorted\_list[index\_1] + sorted\_list[index\_2])/2.0

return mean

print median([2, 4, 5, 9]) # Outputs 4.5

print median([2, 4, 5, 9, 13]) # Outputs 5

# To find variance

def grades\_average(grades\_input):

sum\_of\_grades = grades\_sum(grades\_input)

average = sum\_of\_grades / float(len(grades\_input))

return average

def grades\_variance(scores):

average = grades\_average(scores)

variance = 0

for score in scores:

variance += ((average - score) \*\* 2) / len(scores)

return variance

print grades\_average(grades)

print grades\_variance(grades)

# To find std\_deviation

def grades\_std\_deviation(variance):

return variance \*\* 0.5

variance = grades\_variance(grades)

print grades\_std\_deviation(variance)

print grades\_variance(grades)

# To find average, variance and std\_deviation(variance)

grades = [100, 100, 90, 40, 80, 100, 85, 70, 90, 65, 90, 85, 50.5]

def print\_grades(grades\_input):

for grade in grades\_input:

print grade

def grades\_sum(scores):

total = 0

for score in scores:

total += score

return total

def grades\_average(grades\_input):

sum\_of\_grades = grades\_sum(grades\_input)

average = sum\_of\_grades / float(len(grades\_input))

return average

def grades\_variance(scores):

average = grades\_average(scores)

variance = 0

for score in scores:

variance += ((average - score) \*\* 2) / len(scores)

return variance

def grades\_std\_deviation(variance):

return variance \*\* 0.5

variance = grades\_variance(grades)

print print\_grades(grades)

print grades\_sum(grades)

print grades\_average(grades)

print grades\_variance(grades)

print grades\_std\_deviation(variance)

Git

git init > Initialize this repo as a Git repository | convert a local repository into a Git repository

ls -A or dir /a > should see hidden called .git

touch file1.txt file2.txt > touch in Mac means create a file named xxx.txt or touch this file without editing/adding new stuff

type nul > filename.txt > window’s equivalent of touch, do nothing but did something

git status > to see what's changed in your working directory

git add file1.txt > tell git that we want it to keep tabs on our changes

git add . > add whole bunch

git commit -m "message" > move our file changes from the staging index to the head, -m to write a message about this changes

git push origin master > synchronize and "push" those edits up to our remote Github.com repo

git clone hyperlink > Clone or Download link to a new repos. new local folder for your repo, containing any files your repo had

* Here must do a git init , git add. , git commit, git push, lastly refresh online git to see changes

git download hyperlink > Same as clone method

git pull origin master > Sync from Github to Local

git remote add origin hyperlink > to push new local files to github, if never done before.

rmdir directory /s > to delete whole directory

git clone hyperlink > to fork is to clone other pp repo to yours

git branch newbranch > branch out from master. Then give this branch a name, example “newbranch”

git checkout newbranch > this enters into branch called newbranch

* After done changes, proceed git add, commit, then switch back to master branch

git checkout master > switch back to master branch, you will realize haven’t merge yet

git merge newbranch > after checkout master branch, here to merge the newbranch into master. Hopefully no conflict.

* If merge conflicted, open file to fix it manually, the git add, git commit

git fetch origin > if create wrong branching, here’s to reset branch. Fetch then reset

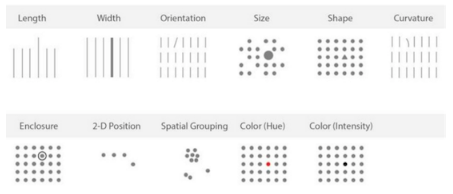
git reset --hard origin/master> reset here. Resets back to HEAD file.

**Visualization:**

**import matplotlib.pyplot as plt**

**%matplotlib inline**

Below are different ways to emphasizing your analysis



**Plotting with Pandas**

Bar chart Syntax: df.plot(kind='bar', x='col', y='col') or df.plot.bar(x='col', y='col').

Pie chart Syntax: df.plot.pie(y='column'), df.column.plot.pie(), or df.plot(kind='pie', y='column')

Scatterplots Syntax: df.plot(kind='scatter', x='col', y='col') or df.plot.scatter(x='column', y='column', c='column', s=numeric).

Histogram : <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.hist.html#pandas.DataFrame.hist>

DataFrame.hist(data, column=None, by=None, grid=True, xlabelsize=None, xrot=None, ylabelsize=None, yrot=None, ax=None, sharex=False, sharey=False, figsize=None, layout=None, bins=10, \*\*kwds)

alpha, which allows data to become transparent, enables the comparison of two distributions, while stacked places them on top of one another to show aggregate trends.

While the embedded functions of .plot.kind() allow for more customization, using .plot(kind='') can be more straightforward and simple. It's often a matter of preference.

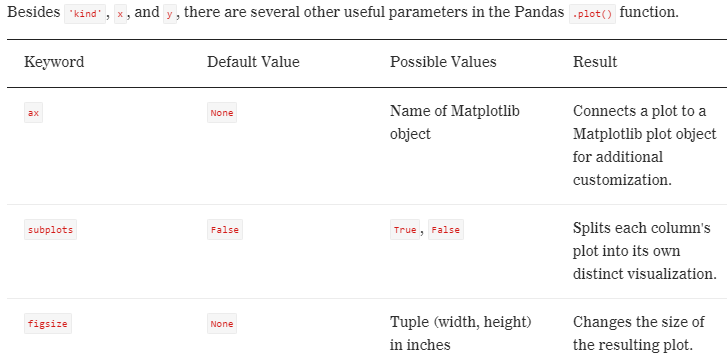
import pandas as pd

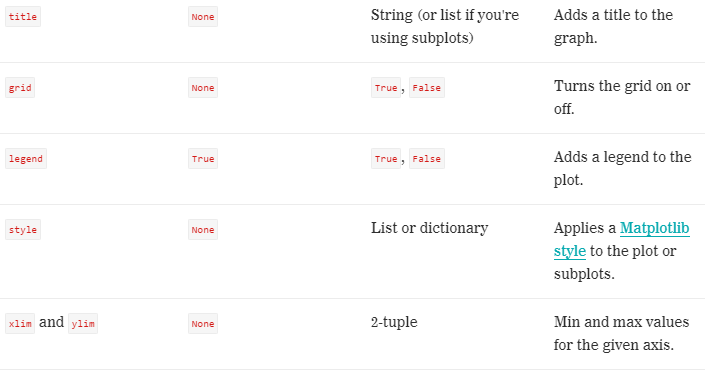
import matplotlib.pyplot as plt

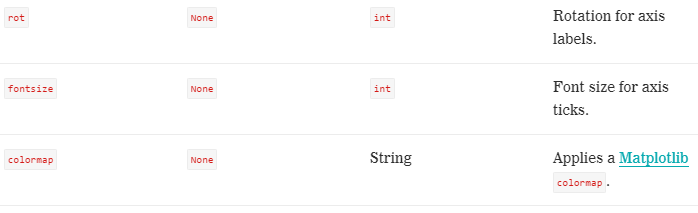
% matplotlib inline

df.plot(x='columnname', y='columnname' , kind='yourchoice')

* bar or barh for bar plots.
* hist for histograms.
* box for box plots.
* kde or density for density plots.
* area for area plots.
* scatter for scatterplots.
* hexbin for hexagonal bin plots.
* pie for pie plots.







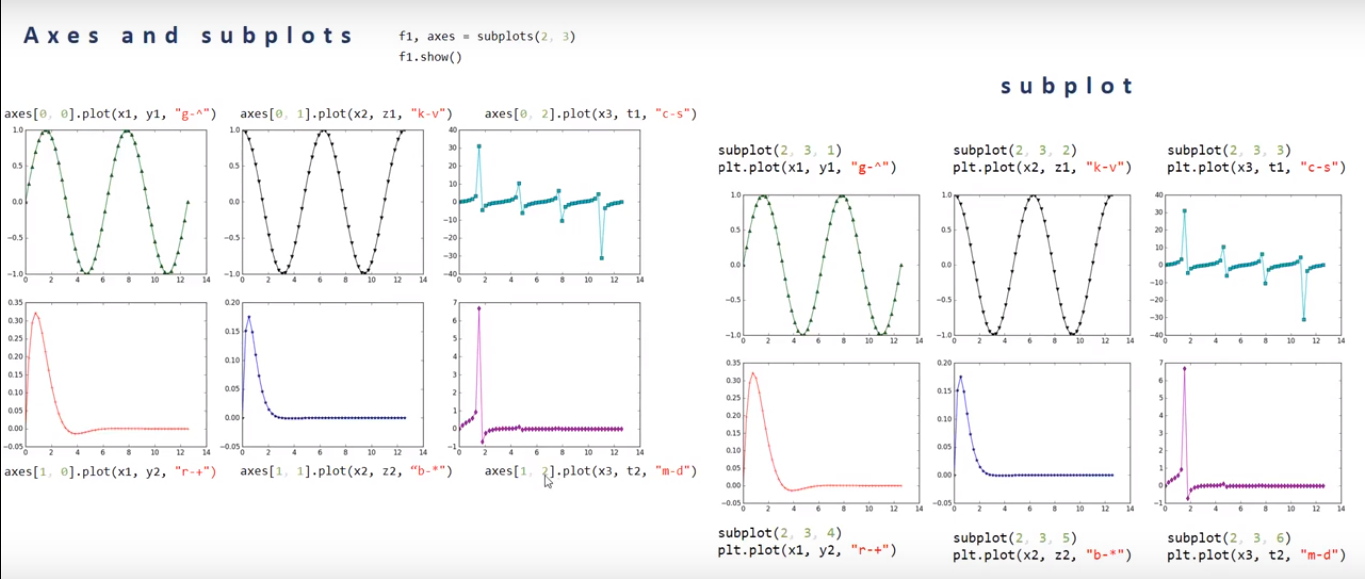
Other parameters to explore are layout, sharex/sharey, use\_index, logx/logy/log log, xticks/yticks, table, xerr/yerr, stacked, secondary\_y, and sort\_columns. Check out their documentation [here](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html#pandas.DataFrame.plot).

**Figure and Axes** Noel dun understandly explained, Split a big Figure into several smaller plots called Axes, like doing subplot many times but made simple

fig, ax = plt.subplots()

Figures: These objects store all of the figure-level attributes and allow the plot to output as an image. Every axes object has a parent figure object.

Axes: These objects have plotting methods and define the coordinate system among other attributes.



**Seaborn** 🡺 can do pairplot , regplot, boxplot, heatmap

import seaborn as sns

sns.set\_style(style)

sns.set\_style(style, {'attribute':'value'})

Seaborn has themes style: darkgrid, whitegrid, dark, white, and ticks

Example:

sns.set\_style('darkgrid', {'figure.facecolor':'skyblue'})

plt.plot(x, 'g--')

**Seaborn's pairplot** graphs allow you to see the relationship between all of your numeric variables. They compare each to another on a mini scatterplot. They also show distribution, as they include a histogram for when each variable is plotted against itself in the grid.

**Seaborn's heatmap** graphs also display relationships between variables, but in this case, they use color to visualize the value of a target variable. While this is less precise, it allows us to look at a continuous variable split by two categories at once. We'll often use this to inspect correlations.

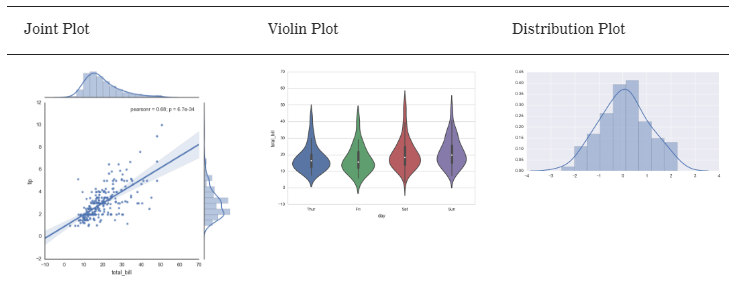
**Seaborn's regplot** function wraps linear regression right into the building of the chart. These charts show a scatterplot of two variables plotted against one another, then fit a linear regression to the plot and shade the area against the regression line that represents a given confidence interval. **NOEL: it’s a scattered plot cum a straight line.**

**Seaborn's boxplot** function is a more powerful than Matplotlib's (and, consequently, Pandas'). In particular, it's much more adept at calculating and representing outliers, as well as providing more aesthetic appeal.

jointplot: This links a comparative chart (e.g., a scatterplot or regression plot) to distribution plots (e.g., histograms) by placing the latter on the edges of the former.

violinplots: Like box plots but with varying widths to show density of data.

distplot: A souped-up version of histograms, these charts automatically calculate a good bin size rather than just using a default and plot a kdeplot or a scipy.stats distribution over the bars.



**Summaries**

Matplotlib is perfect for simpler graphs (scatterplots, line graphs, bar charts, etc.), and

Seaborn can adeptly handle more complex ones (heat maps, regression plots, distribution plots, etc.)

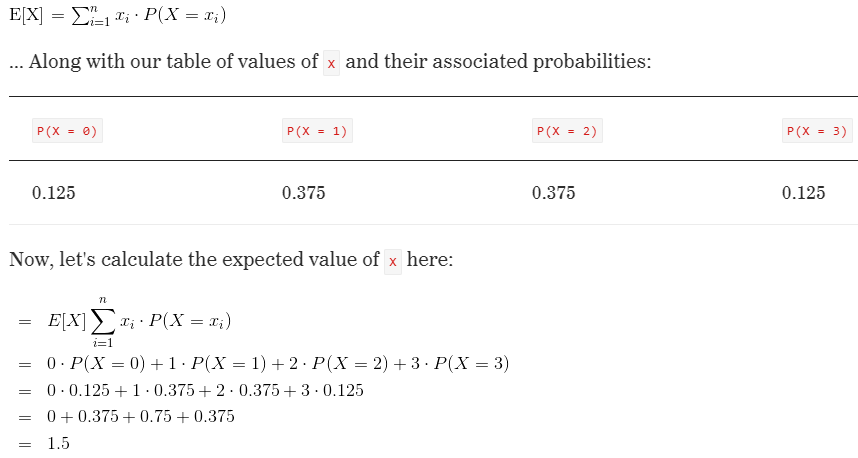
More example exercises here in link: <pandasplot.pdf>

**Math**

**CONTINUOUS" or "DISCRETE** Noel explained: If I can predict a finite whole number, it is Discrete. If cannot predict a finite whole number, it is continuous, like asking measurement bcos the number can go on and on….0.00001. Continuous variables are measured and discrete variables can be counted.

**Expected Value E[X]**: is average value..

**Example of E[X of a Discrete Distribution:**



**Example of E[X of a Continuous Distribution:** cannot use same formula bcos i is uncountable.

https://ga-instruction.s3.amazonaws.com/json/DF-Python/assets/unit-7/L7.1_expected_value_equation3.png

**Descriptive statistics:** Use this when there is not much to look at in a Data Set. *Descriptive statistics* is branch of statistics provides us with ways of understanding and summarizing data — and presenting visually appealing results.

**Descriptive vs. Inferential Statistics**

**Descriptive statistics** focus on summarizing, describing, and understanding the data we observe.

**Inferential statistics** focus on generalizing the results from a **sample** to a larger population.

**variance and standard** deviation of [2,4,6,8].

1) The mean is 5.

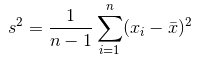
2) The differences from each observation to the mean are [-3, -1, 1, 3].

3) The squared results are [9, 1, 1, 9].

4) The summed results are 20.

5) Dividing 20 by (n-1) = (4-1) = 3 and yields 20/3. (This is the **variance**, which is approximately 6.7.)

6) Taking the square root of 20/3 yields the **standard deviation**, which is approximately 2.6.



**Percentiles**

**25th percentile of [10, 14, 16, 19, 20] is (10 + 14) / 2 = 12**

**50th percentile is equivalent to the median = 16**

**75th percentile of [10, 14, 16, 19, 20] is (19 + 20) / 2 = 19.5**

**Central Limit Theorem** is a Inferential statistics where focus on generalizing results from a sample to a larger population.

The **central limit theorem states that**, regardless of the distribution of the original data x, the sampling distribution of Ẍ will approach a normal distribution as the sample size n gets larger and larger.

Step 1: 2x thing to consider. 1st: Is it normal dis. 2nd sample size > 30. If normal, then dun need CLT, it will be normal regardless of sample size, just use normal distribution for inferential statistic like “confidence intervals”, “hypothesis tests”, etc.). But if dun know normal or is NOT normal, then we depend on CLT if sample size > 30, and will approach normal as sample size n increases.

**Univariate** statistics are statistics calculated from only one variable.

**Bivariate** statistics are statistics calculated from two variables. The two most widely known bivariate statistics are correlation and covariance. Question we ask ourselves : "As one variable changes, how does the other change?" We calculate **Covariance**.

**Covariance vs Correlation**

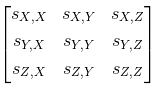
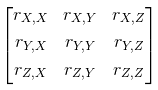
**Covariance** measures the strength and direction of the linear relationship between X and Y, but **has units** e.g litre meter, or diff units, which doesn’t makes sense.

If x and y goes up or goes down together, values will be either +ve or –ve, covariance will be > 0. Meaning Values greater than zero indicate that the variables have a positive linear relationship.

If x and y moves opposite direction, then covar < 0, meaning Values less than zero indicate that the variables have a negative linear relationship. .

If these variable are independent of each other, no relationship, then covar = 0. Values close to zero indicate little to no linear relationship between the two variables.

**Pearson’s Correlation** measures the strength and direction of the linear relationship between two variables but is bounded between -1 and 1. Correlation is **unitless**, easy to interpret. If value close to zero, it has a weak linear relationship, whereas values if farther from zero have a stronger linear relationship. If Correlation -1 = perfectly inversely correlated. Correlation +1 = perfectly correlated. Correlation 0 = Not correlated

**Covariance Matrices correlation matrix**

Correlation and Covariance in Python

x = [1,2,3,4,5]

y = [6,5,4,3,2]

import numpy as np

print np.cov(x,y) ## Prints the covariance matrix.

print np.corrcoef(x,y) ## Prints the correlation matrix.

Other Correlation Measures

Kendall Rank Correlation: The Kendall approach is used to measure the ordinal association between two measured quantities. Specifically, it measures the strength of dependence between two variables. For example, you may want to determine if there is a correlation between two track runners finishing in 1st and 2nd place.

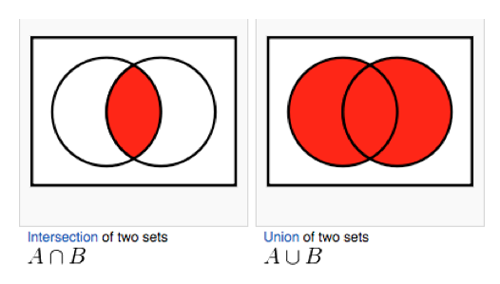
Spearman's Rank Correlation: Spearman's correlation is used to measure the degrees of association between two variables. A question that might be asked with Spearman correlation would be, "Is there a statistically significant relationship between track runner age and how they place in a given race?"

**Axioms of Probability and Bayes' Theorem**

An event ‘E’ is any collection of outcomes from an experiment. Noel: Actual Outcome.

Sample space is the set of all possible outcomes of an experiment, denoted by S. Noel: Possible outcome

**Unions and intersections.**

alt text

Axioms of probability**. = rules of probability**

1. **probability of an event cannot be negative**
2. **alt text probability of an outcome occurring in the sample space is 1. A scale from 0% to 100%.**
3. **for a series of events that are mutually exclusive, the probability of a union of those events is the sum of the probabilities of the individual events.** alt text

**Marginal probabilities P(A) :** probabilities of a single event E.

**Joint probability :** alt textprobability of two events occurring. joint probability of A and B, alt text, is given by alt text.

alt text

Example: Probability that an individual will develop cancer, single event. Probability that an individual will both smoke and develop cancer, Joint.

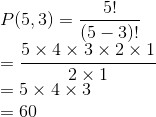
**Conditional probability:** are used in cases where we know that an event has already occurred. Example : Probability of cancer given that someone smokes. In other words, we know that smoking has already occurred. alt text. “|” means given that. alt text, probabilty it is a spam, given that it has estranged and bank…

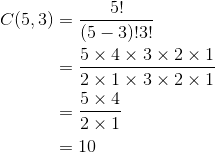
alt text**Bayes' theorem:** allows us to connect two related conditional probabilities. alt textWe use the Bayes' theorem because it describes the probability of an event based on prior knowledge of conditions that might be related to the event. In other words, you're reasoning backwards to find the sequence of events that lead to a given outcome.

**Counting Principle:** Now let's say we have three meals to cook: an appetizer, an entrée, and a dessert. There are five ways to prepare the appetizer, two ways to prepare the entrée, and four ways to prepare the dessert. How many ways are there to prepare all three meals?

Answer The counting principle indicates that there are 5 x 2 x 4 = 40 ways of preparing all three meals.

**Factorials** are important to learn because n! is the number of ways to list a set of n objects.

**Permutation:** An arrangement of objects without repetition in which order matters. alt text where n is the number of options and k is the number of options we want to choose. Example: there are five qualified candidates to fill three roles: president, vice president, and secretary. How many ways can these roles be filled, assuming one person per position?

**Combination:** An arrangement of objects without repetition in which order doesn't matter**.** alt textwhere n is the number of options and k is the number of options we want to choose. Example: there are five qualified candidates to fill three identical roles. How many ways can these roles be filled, assuming one person per position?? In this case, because the roles are identical, our order does not matter.

**Advance combination example: Combination and Counting principle together**

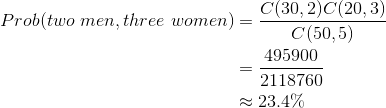
Out of a total of 20 women and 30 men, how many faculty committees of five contain two men and three women?

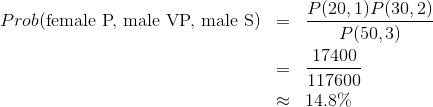
Task 1: How many ways can we choose two men from 30? C(30,2)

Task 2: How many ways can we choose three women from 20? C(20,3)

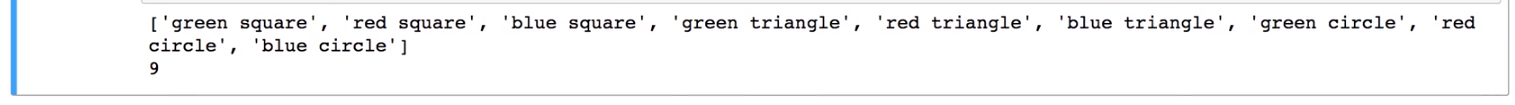
Given the counting principle, we can multiply these two together to find how many committees contain two men and three women.

alt text

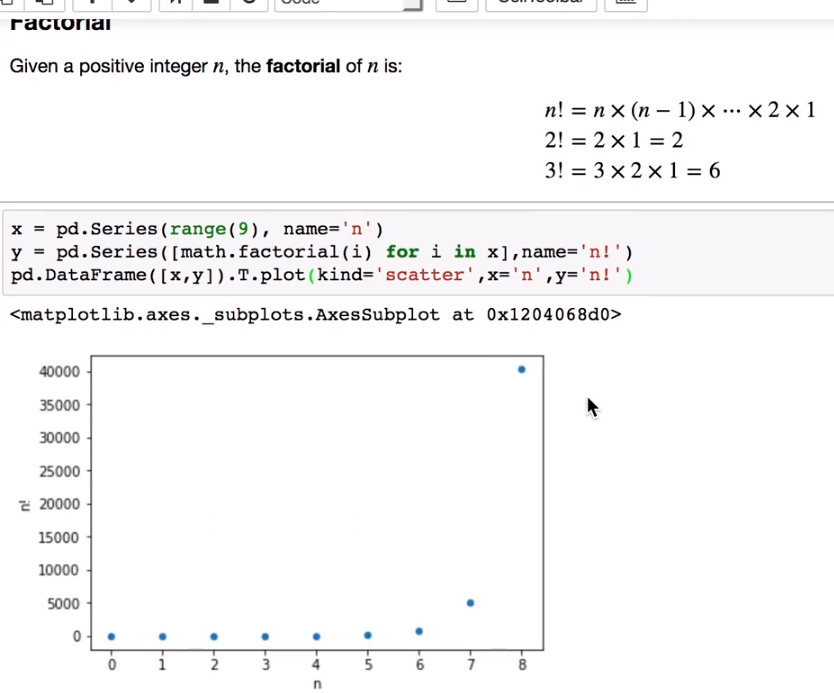
**From Combinations to Probability:** Example how likely it is that (given the totals above) a faculty committee will be composed of two men and three women. Noel note: Answer is less than 1. So must x100 for percent.

**From Combinations to Probability:** What is the probability that — out of a total of 30 men and 20 women — a woman is selected as president and two men are selected to be vice president and secretary?

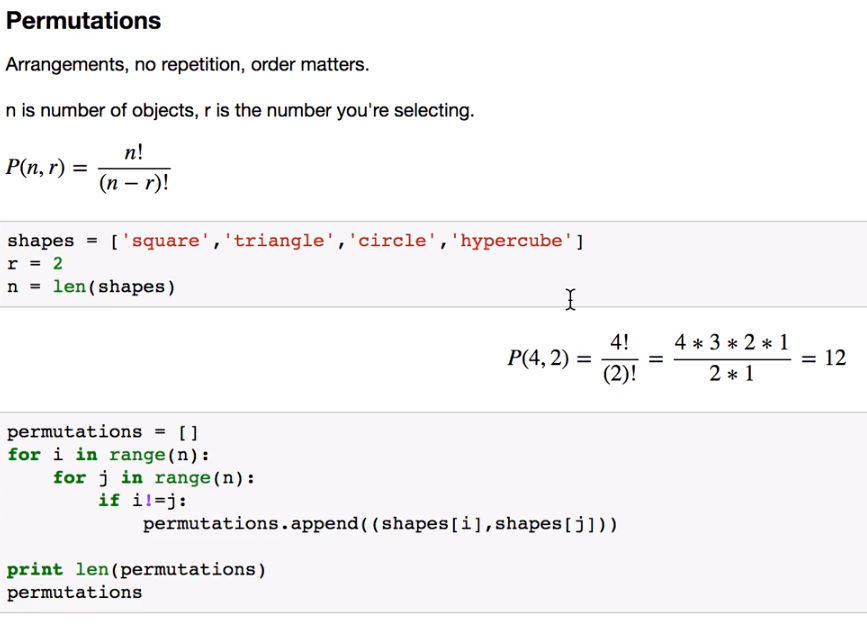
**Counting Principle in Python**

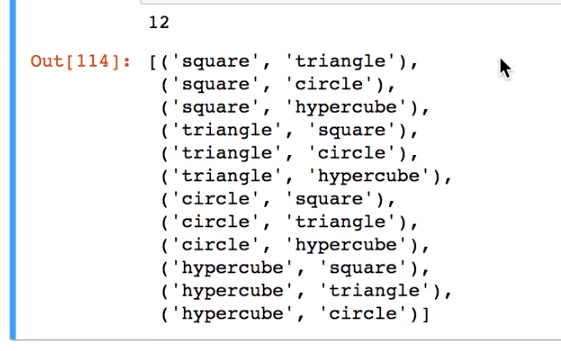
 

Factorial in Python

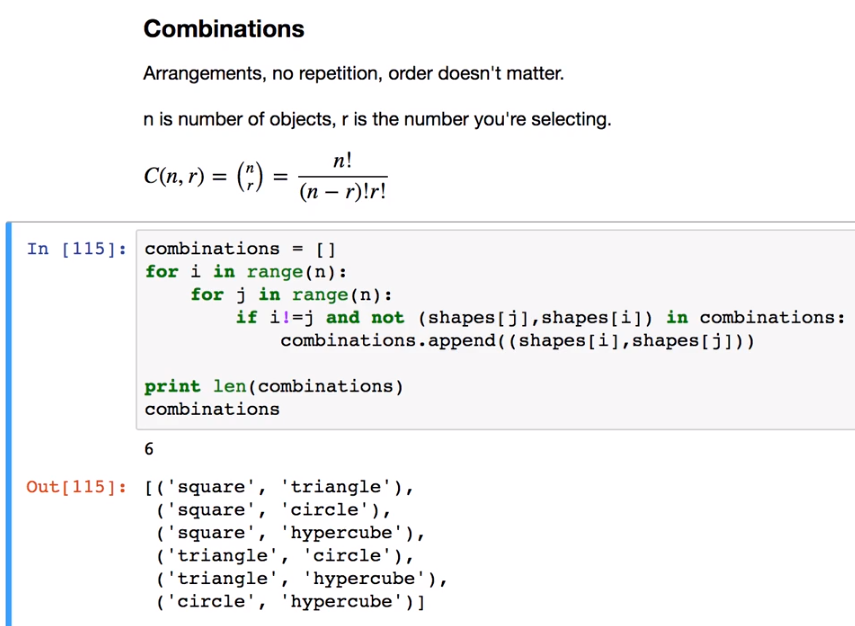


**Permutation in Python**

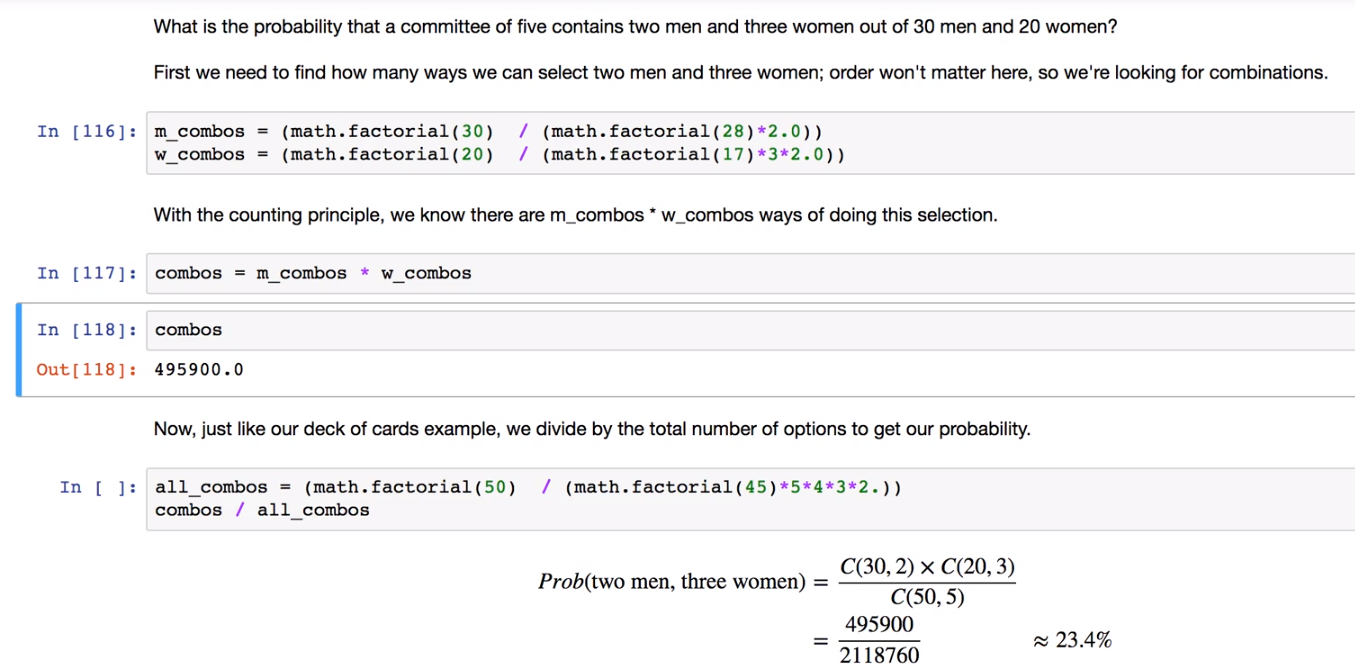




**Combination in python**



**Combination to Probability in Python**



**Scalars and Vectors**

A **scalar** is a single number. Scalars are often used for measurements such as length, distance, and speed. Example 30.4KM/h, 37 words etc.. typically written as a non-bold, lowercase character.

A **vector** is an ordered list of scalars. This is called the "component form. Typically expressed in bold, lowercase letter (e.g., a) or a lowercase letter with an arrow on top.

Scalar is a magnitude, while a vector represents a magnitude and direction.

Example there are 3 vectors below

a = (-4, 81, 28, 0)

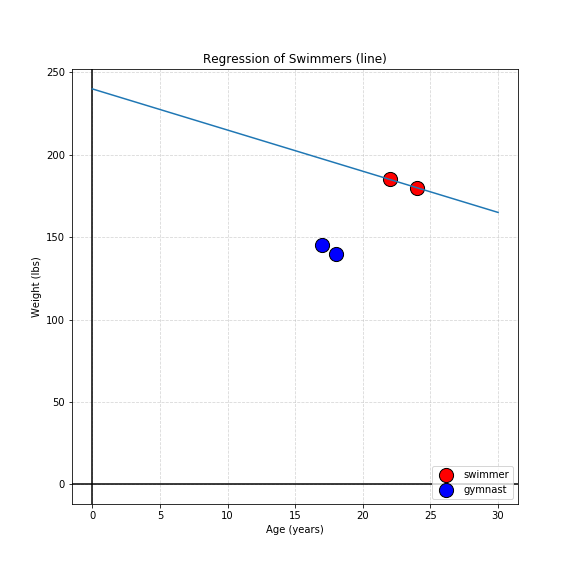
b = (3.5, -8.2)

c = (1,0,0,1,1,0,0)

**Vectorizing** : CountVectorizer() The process of converting non-numeric data points to vectors is called vectorizing the data. hy is this important? In regression models (e.g., linear/logistic regression), the bias is the value where all variables are 0. So, if we use a one hot encoding, then the intercept is difficult to interpret.

**Noel, just read this about scalars and vectors, I dun understand anyway**: A value of 0 is meaningful — it corresponds to fencer. Typically, fewer components are desirable because fewer features mean fewer coefficients to fit. Finally, information is not lost by using a single scalar because of the 0/1/2 encoding scheme. The key here is that models rely on distance measurements to determine how far off we are, for example, for errors. This is guessing that a swimmer gives an error of 2 instead of fencer, while gymnasts and fencers give an error of 1. These values are arbitrary, as categories fundamentally have no distance metric. By using dummy variables, we work around this (for linear models) by allowing a separate coefficient per category rather than multiplying the "category value" by a single coefficient.

R**egression**. Instead of using the line to discriminate between two groups and predict categories, we'll use it to help predict exact numbers.



**Generalizing a Line**

In 2D dimensions, we solve for a line that separates the points (y = Ax + B).

In 3D dimensions, we solve for the equation of a **plane** (z = Ax + By + C).

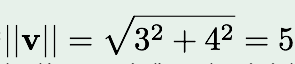
In more than 3D dimensions, we solve for a **hyperplane** (w = Ax + By + Cz + ...).

Where: capital letters denote coefficients — precomputed known values. Lowercase letters denote variables. Specifically, we're predicting the dependent variable ( z in z = Ax + By + C) by knowing the values of the independent variables ( x and y in z = Ax + By + C).

<span class="katex-display"><span class="katex"><span class="katex-mathml"><math><semantics><mrow><mrow><mi mathvariant="bold">x</mi></mrow><mo>=</mo><mo>(</mo><mo>−</mo><mn>1</mn><mo separator="true">,</mo><mn>1</mn><mo separator="true">,</mo><mn>3</mn><mo separator="true">,</mo><mn>5</mn><mo>)</mo></mrow><annotation encoding="application/x-tex">\mathbf{x} = (-1, 1, 3, 5)</annotation></semantics></math></span><span class="katex-html" aria-hidden="true"><span class="strut" style="height:0.75em;"></span><span class="strut bottom" style="height:1em;vertical-align:-0.25em;"></span><span class="base"><span class="mord"><span class="mord mathbf">x</span></span><span class="mrel">=</span><span class="mopen">(</span><span class="mord">−</span><span class="mord mathrm">1</span><span class="mpunct">,</span><span class="mord mathrm">1</span><span class="mpunct">,</span><span class="mord mathrm">3</span><span class="mpunct">,</span><span class="mord mathrm">5</span><span class="mclose">)</span></span></span></span></span>Calculating **magnitude** of vector x:

<span class="katex-display"><span class="katex"><span class="katex-mathml"><math><semantics><mrow><msqrt><mrow><mo>(</mo><mo>−</mo><mn>1</mn><msup><mo>)</mo><mn>2</mn></msup><mo>+</mo><msup><mn>1</mn><mn>2</mn></msup><mo>+</mo><msup><mn>3</mn><mn>2</mn></msup><mo>+</mo><msup><mn>5</mn><mn>2</mn></msup></mrow></msqrt><mo>=</mo><msqrt><mrow><mn>3</mn><mn>6</mn></mrow></msqrt><mo>=</mo><mn>6</mn></mrow><annotation encoding="application/x-tex">\sqrt{(-1)^2 + 1^2 + 3^2 + 5^2} = \sqrt{36} = 6</annotation></semantics></math></span><span class="katex-html" aria-hidden="true"><span class="strut" style="height:0.983875em;"></span><span class="strut bottom" style="height:1.24em;vertical-align:-0.25612499999999994em;"></span><span class="base"><span class="mord sqrt"><span class="vlist-t vlist-t2"><span class="vlist-r"><span class="vlist svg-align" style="height:0.983875em;"><span style="top:-3.2em;"><span class="pstrut" style="height:3.2em;"></span><span class="mord" style="padding-left:1em;"><span class="mopen">(</span><span class="mord">−</span><span class="mord mathrm">1</span><span class="mclose"><span class="mclose">)</span><span class="msupsub"><span class="vlist-t"><span class="vlist-r"><span class="vlist" style="height:0.740108em;"><span style="top:-2.9890000000000003em;margin-right:0.05em;"><span class="pstrut" style="height:2.7em;"></span><span class="sizing reset-size6 size3 mtight"><span class="mord mathrm mtight">2</span></span></span></span></span></span></span></span><span class="mbin">+</span><span class="mord"><span class="mord mathrm">1</span><span class="msupsub"><span class="vlist-t"><span class="vlist-r"><span class="vlist" style="height:0.740108em;"><span style="top:-2.9890000000000003em;margin-right:0.05em;"><span class="pstrut" style="height:2.7em;"></span><span class="sizing reset-size6 size3 mtight"><span class="mord mathrm mtight">2</span></span></span></span></span></span></span></span><span class="mbin">+</span><span class="mord"><span class="mord mathrm">3</span><span class="msupsub"><span class="vlist-t"><span class="vlist-r"><span class="vlist" style="height:0.740108em;"><span style="top:-2.9890000000000003em;margin-right:0.05em;"><span class="pstrut" style="height:2.7em;"></span><span class="sizing reset-size6 size3 mtight"><span class="mord mathrm mtight">2</span></span></span></span></span></span></span></span><span class="mbin">+</span><span class="mord"><span class="mord mathrm">5</span><span class="msupsub"><span class="vlist-t"><span class="vlist-r"><span class="vlist" style="height:0.740108em;"><span style="top:-2.9890000000000003em;margin-right:0.05em;"><span class="pstrut" style="height:2.7em;"></span><span class="sizing reset-size6 size3 mtight"><span class="mord mathrm mtight">2</span></span></span></span></span></span></span></span></span></span><span style="top:-2.9438750000000002em;"><span class="pstrut" style="height:3.2em;"></span><span class="hide-tail" style="min-width:1.02em;height:1.2em;"><svg width='400em' height='1.2em' viewBox='0 0 400000 1200' preserveAspectRatio='xMinYMin slice'><path d='M263 601c.667 0 18 39.667 52 119s68.167
 158.667 102.5 238 51.833 119.333 52.5 120C810 373.333 980.667 17.667 982 11
c4.667-7.333 11-11 19-11h398999v40H1012.333L741 607c-38.667 80.667-84 175-136
 283s-89.167 185.333-111.5 232-33.833 70.333-34.5 71c-4.667 4.667-12.333 7-23
 7l-12-1-109-253c-72.667-168-109.333-252-110-252-10.667 8-22 16.667-34 26-22
 17.333-33.333 26-34 26l-26-26 76-59 76-60zM1001 0h398999v40H1012z'/></svg></span></span></span><span class="vlist-s">​</span></span><span class="vlist-r"><span class="vlist" style="height:0.25612499999999994em;"></span></span></span></span><span class="mrel">=</span><span class="mord sqrt"><span class="vlist-t vlist-t2"><span class="vlist-r"><span class="vlist svg-align" style="height:0.956095em;"><span style="top:-3em;"><span class="pstrut" style="height:3em;"></span><span class="mord" style="padding-left:0.833em;"><span class="mord mathrm">3</span><span class="mord mathrm">6</span></span></span><span style="top:-2.916095em;"><span class="pstrut" style="height:3em;"></span><span class="hide-tail" style="min-width:0.853em;height:1em;"><svg width='400em' height='1em' viewBox='0 0 400000 1000' preserveAspectRatio='xMinYMin slice'><path d='M95 622c-2.667 0-7.167-2.667-13.5
-8S72 604 72 600c0-2 .333-3.333 1-4 1.333-2.667 23.833-20.667 67.5-54s
65.833-50.333 66.5-51c1.333-1.333 3-2 5-2 4.667 0 8.667 3.333 12 10l173
378c.667 0 35.333-71 104-213s137.5-285 206.5-429S812 17.333 812 14c5.333
-9.333 12-14 20-14h399166v40H845.272L620 507 385 993c-2.667 4.667-9 7-19
7-6 0-10-1-12-3L160 575l-65 47zM834 0h399166v40H845z'/></svg></span></span></span><span class="vlist-s">​</span></span><span class="vlist-r"><span class="vlist" style="height:0.08390500000000001em;"></span></span></span></span><span class="mrel">=</span><span class="mord mathrm">6</span></span></span></span></span>

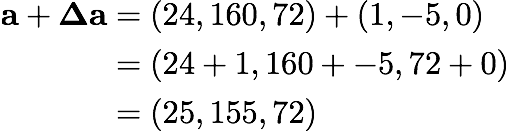
When we say a **vector is normalized:** means By definition, normalizing a vector produces a vector in the same direction with magnitude 1. Hence, the normalized vector will lie on the unit hypersphere, the collection of all points with magnitude 1. (Recall that in two dimensions, a circle is x^2 + y^2 = 1. A circle is the collection of all points at a distance of 1 from the origin.) Because we know the direction of a normalized vector but not its magnitude, all points in the same direction are normalized to the same vector, making the original irrecoverable.

Question: **Normalize the vector v = (3, 4).** To find the unit vector, first we must compute its magnitude:

Next, we divide the vector by its magnitude. So, (3, 4)/5 = (3/5, 4/5). Remember that this vector point

lies on the unit circle. You can verify this by finding the magnitude of the point, which should be 1.

Adding Two Vectors : Suppose we have a 24-year-old Olympic athlete who weighs 160 pounds and is 72 inches tall. We'll vectorize this as (24, 160, 72). Suppose that one year later, the athlete gains one year and loses five pounds — we'll vectorize the displacement as (1, -5, 0).



Multiplying a Vector by a Vector (Dot Product) : For example, suppose x = (1, 2, 3) and y = (4, 5, 6): <span class="katex-display"><span class="katex"><span class="katex-mathml"><math><semantics><mrow><mrow><mi mathvariant="bold">x</mi></mrow><mo>⋅</mo><mrow><mi mathvariant="bold">y</mi></mrow><mo>=</mo><mn>1</mn><mo>⋅</mo><mn>4</mn><mo>+</mo><mn>2</mn><mo>⋅</mo><mn>5</mn><mo>+</mo><mn>3</mn><mo>⋅</mo><mn>6</mn><mo>=</mo><mn>4</mn><mo>+</mo><mn>1</mn><mn>0</mn><mo>+</mo><mn>1</mn><mn>8</mn><mo>=</mo><mn>3</mn><mn>2</mn></mrow><annotation encoding="application/x-tex">\mathbf{x} \cdot \mathbf{y} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 4 + 10 + 18 = 32</annotation></semantics></math></span><span class="katex-html" aria-hidden="true"><span class="strut" style="height:0.64444em;"></span><span class="strut bottom" style="height:0.8388800000000001em;vertical-align:-0.19444em;"></span><span class="base"><span class="mord"><span class="mord mathbf">x</span></span><span class="mbin">⋅</span><span class="mord"><span class="mord mathbf" style="margin-right:0.01597em;">y</span></span><span class="mrel">=</span><span class="mord mathrm">1</span><span class="mbin">⋅</span><span class="mord mathrm">4</span><span class="mbin">+</span><span class="mord mathrm">2</span><span class="mbin">⋅</span><span class="mord mathrm">5</span><span class="mbin">+</span><span class="mord mathrm">3</span><span class="mbin">⋅</span><span class="mord mathrm">6</span><span class="mrel">=</span><span class="mord mathrm">4</span><span class="mbin">+</span><span class="mord mathrm">1</span><span class="mord mathrm">0</span><span class="mbin">+</span><span class="mord mathrm">1</span><span class="mord mathrm">8</span><span class="mrel">=</span><span class="mord mathrm">3</span><span class="mord mathrm">2</span></span></span></span></span>.

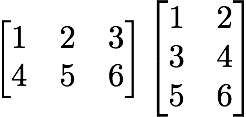
Explained : If the vectors were more than 90 degrees apart, the dot product would be negative. If they were 90 degrees apart, it would be 0. So, neither of these are possible. At an angle of 0 degrees, cos(0) = 1, so, 1,000 \* 0.1 \* cos(0) = 100. At an angle approaching 90 degrees, the cosine would approach 0 sufficiently enough to scale down large magnitudes to 100 as needed.

If The dot product is the sum of the squares of each component. So, 3x4 + -2x11 + 5x2 = 12 - 22 + 10 = 0. Because the dot product is 0, the vectors are orthogonal.

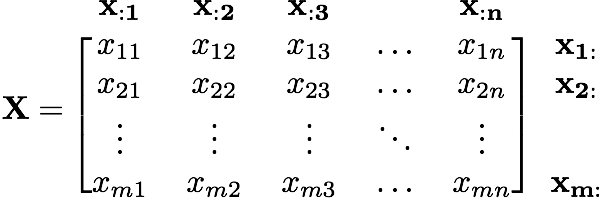
**Matrix**

**Noel explain How to read dimension x dimension = what dimension**

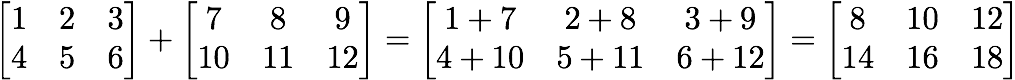
**Below is [2 x 3] . [3 x 2] = should get a [2 x 2]**



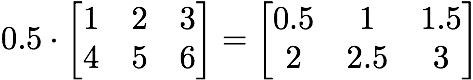
**Another Noel example [2 x 3] . [3 x 1] = should get a [2 x 1]**



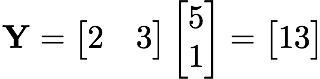
**Matrix Addition (same goes for subtraction):**

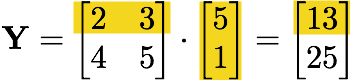


**Matrix Multiplication by a Scalar: (Matrix division by a scalar is defined similarly)**



**Another example**





**What use is Matrix Multiplication??**

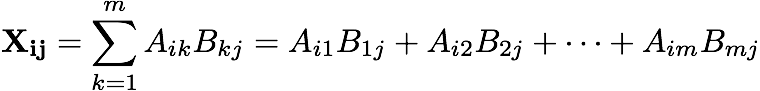
**Instead of just storing data points, we're now using matrices to store systems of equations.**

**m: Number of equations.**

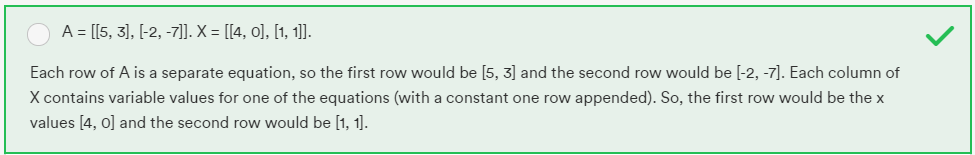
**n: Number of features/coefficients (excluding the bias).**

**n+1: Number of features/coefficients (including the bias).**

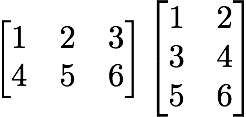
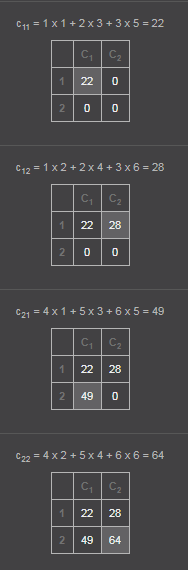
matrix multiplication as a formula. For **X** = **AB**, where **A** has *m* columns, the ith row and jth column of **X** are given as:



**Example: How might the equations y1 = 5x + 3 and y2 = -2x - 7, evaluated at the points x = 4 and x = 0, be represented as AX (where X includes values for the constant coefficients)?**



**Example Matrix Multiplication**

 **=**   = [22 28] [49 64]

**Why might we want to perform multiple calculations at once? Answer:** Matrix Multiplication: Distance From Predictions. In data science, we're using working with lots of data points. A common reason for using matrix multiplication is that we may want to know the squared sum of the distances of all data points from the predicted data points. This provides us with a measure of how far our predictions are from the actual data points. This is often used as a constraint to minimize in linear regression (and even sometimes in neural networks).

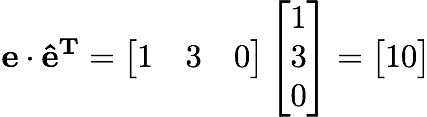
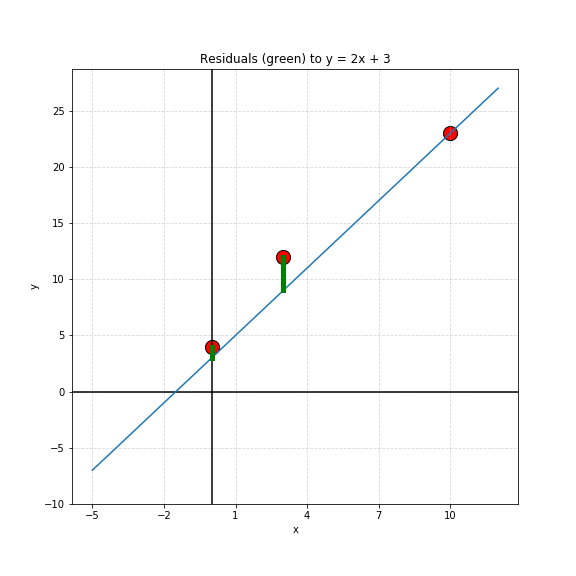
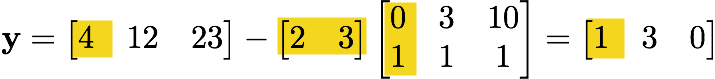
If we have hundreds of data points, we'll have to perform hundreds of calculations to make predictions. If we organize this problem as a matrix multiplication, we can take advantage of major algorithmic speed-ups, or even the parallelization of GPUs.

That's right, matrix multiplication is our calculations' turbo-boost! To compare how far off our model predictions are from known points, we must calculate the residuals — the distance between the actual and predicted values. These are extremely important in data science. The mean residual gives us some indication of how well our model fits the data.

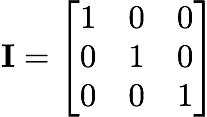
The vector dot product is commutative because each component of the result is the scalar product of the two respective components, and scalar multiplication is commutative. For example, (1, 2) . (3, 4) = (1x3, 2x4) = (3x1, 4x2) = (3, 4) . (1, 2). Matrix multiplication, however, is non-commutative — order matters! For example, in many cases the result will be undefined. Multiplying a two-by-three matrix and a three-by-four matrix yields a two-by-four matrix. However, swapping the order, the product of a three-by-four matrix and a two-by-three matrix is undefined, as the number of columns of in the first matrix does not match the number of rows in the second.

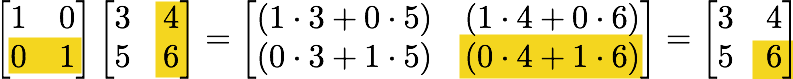
**Matrix Multiplication: Squared Error**

Suppose we have the actual points (0, 4), (3, 12), and (10, 23). Our predictive model is y = 2x + 3. Taking each of the x values from our actual points, our predicted y values would be 3, 9, and 23, respectively. These are mostly different from our actual values. In fact, the residuals are 4 - 3 = 1, 12 - 9 = 3, and 23 - 23 = 0, respectively, for a total error of 4. In practice, we might square each difference and then take the sum of the squared distances for a **total sum of squared errors** of 1^2 + 3^2 + 0^2 = **10.**

https://ga-instruction.s3.amazonaws.com/json/DF-Python/assets/linear-algebra-and-regression/459.png

**The Identity Matrix:** is a matrix of all zeros except for the diagonal, which is all ones. The diagonal of a matrix is the set of all elements where the row and column indices are equal. Noel explained: A x A = I **so I x A = A. Identity Matrix**. For example, the three-by-three identity matrix is:

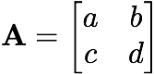
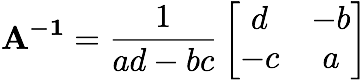




**Matrix Inverse: Noel explain, sometimes need to move matrix across equal like this. But A must be a SQUARE matrix.**

**AX = B**

**X = (A-1) x B**

 🡺  🡺 but if ad-bc = 0 then A-1 will be undefined. This number is an important value in linear algebra, known as the determinant of the matrix.

**Matrix Inverse non-sqaure:** Noel explain, to inverse non-square matrix, must do this in sequence.

**Transpose original m 🡺 becomes n 🡺 Multiply m x n 🡺 will result a square matrix 🡺 do inverse formula 🡺 Done.**

**Matrix in Python**

