PythonRevewClass2-Completed

February 7, 2022

[17]:
```
# import the necessary packages
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

1 Basics

1.1 Indexing

In python, when an object is a collection (like a list, a tuple, or a string...etc), you can specify which items you want from that collection by putting the *index* (or multiple indices) of the item in square brackets.

[18]:
```
my_list = [4,8,15,16,23,42]

# grab 15 from the list above

my_list[2] # remember indexing begins at 0
```

[18]:
```
15
```

[19]:
```
my_string = "Beautiful is better than ugly. Explicit is better than implicit.␣
↪Simple is better than complex. Complex is better than complicated."

# grab "better" from my_string

my_string[13:19]
```

[19]:
```
'better'
```

[20]:
```
my_tuple = (50,100,25)

# grab 50 from my_tuple
```

```
my_tuple[0]
```

[20]: 50

Dictionaries, because they're not ordered, rely on key-value pairs rather than numeric indices. To grab a value from a dictionary, specify the key related to the value you'd like.

[21]:
```
my_dict = {"Happy": True, "Age": 45, "Name": "Janice", "Children": ["Ted",␣
↪"Alan", "Jake"]}

# grab name of 3rd child, Jake, from my_dict

my_dict["Children"][2] # from value associated with the key Children, grab the␣
↪item @ the 2nd index
```

[21]: 'Jake'

1.2 Your Turn

From the object `CPSC392`, grab the average quiz grade for Carla.

[22]:
```
CPSC392 = {"Jimmy": {"QuizGrades": [9,5,9,8,7], "Major": "Computer Science"},
           "Brenda": {"QuizGrades": [9,10,9,10,7], "Major": "Business"},
           "Jacqueline": {"QuizGrades": [3,6,2,8,9], "Major": "Computer␣
↪Science"},
           "Bethany": {"QuizGrades": [2,2,0,4,5], "Major": "Business"},
           "Kristen": {"QuizGrades": [9,7,9,9,9], "Major": "Computer Science"},
           "Elissa": {"QuizGrades": [4,4,5,8,2], "Major": "Foreign Languages"},
           "Carly": {"QuizGrades": [7,6,8,7,9], "Major": "Biology"}}

###

np.mean(CPSC392["Carly"]["QuizGrades"])
```

[22]: 7.4

1.3 If, Else, Elif

If, Else, and Elif are a part of *control flow* which allows you to run blocks of code only when a condition is met (or not).

1.3.1 If

An if statement allows you to run a block of code only when a conition is `True`.

For example:

[23]:
```
if 10%2 == 0:
    print("10 is even")
```

```
10 is even
```

In the code above the condition, `10%2 == 0`, evaluates to `True`, so the if suite (`print("10 is even")`) ran.

The if suite will not run if the condition is `False`.

[24]:
```
# nothing prints out

if 11%2 == 0:
    print("11 is even")
```

1.3.2 Else

Else statements allow you to run *alternative* code when the if statement is **not** `True`.

For example, in the code below, the if suite checks whether the number `i` is even (using the `%` operator), and prints out `"even"` if it is. If it is not even (and the condition is `False`), it prints out `"odd"`.

[25]:
```
i = 6

if i % 2 == 0:
    print("even")
else:
    print("odd")
```

```
even
```

1.3.3 Elif

As you might gather from the name, elif is a combination of else + if. It allows you to check an *additional* condition if the original one (from the if statement) is not met.

For example, as a teacher, I often teach in different rooms for my MW classes, compared to my TTh classes. And on Friday I might work in my office. The If/Elif/Else below can help me keep track of where I need to go.

[26]:
```
day = input("What day is it? ")

if day in ["Monday", "Wednesday"]:
    print("Go to Keck")
elif day in ["Tuesday", "Thursday"]:
    print("Go to Leatherby Library")
elif day == "Friday":
    print("Go to Swenson Hall")
else:
    print("Why are you going to campus!?!")
```

```
What day is it? Monday
Go to Keck
```

Notice that elif only runs when the original if condition is `False`. If all the if and elif conditions are `False`, then (and only then) will the else suite run.

1.4 Your Turn

Using if, elif, and else as needed, check whether a variable, x, is negative, zero, or positive.

[27]:
```
x = 10

if x < 0:
    print("negative number")
elif x == 0:
    print("zero")
else:
    print("positive")
```

```
positive
```

1.5 Your Turn

Use if/elif/else statements to ask the user to input their grade as a percentage (0-100%), and print out whether they have an A, B, C, D or F. (Use 90,80,70,60, and < 60 as your grade cutoffs.)

[28]:
```
grade = 90

if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
elif grade >= 70:
    print("C")
elif grade >= 60:
    print("D")
else:
    print("F")
```

```
A
```

1.6 While Loops

While loops are a control flow method that allow you to run a block of code over and over until the condition you're looping on is `False`.

For example, the code below will run until `i` is 10 or greater.

[29]:
```
i = 0
while i < 10:
```

```
    print("i is now", i)
    i += 1
```

```
i is now 0
i is now 1
i is now 2
i is now 3
i is now 4
i is now 5
i is now 6
i is now 7
i is now 8
i is now 9
```

notice that our condition `i < 10`, is evaluated multiple times. The first time, `i = 0`, so `i < 10` is true, and the while suite runs. Inside the while suite, `i` is incremented by 1, leaving `i` equal to 1. Thus, the next time the condition is checked, it is also `True`, because `1` is less than `10`, and so the while suite runs again.

This will continue until `i` is finally equal to `10`, in which case `i < 10` would evaluate to `False` and the while loop will stop.

It's important that your condition could EVENTUALLY be `False`, otherwise you run into an *infinite loop*.

[30]:
```
# don't run! this is an infinite loop!
# while 1 == 1:
#     print("ahhh!!!")
```

1.7 Your Turn

Use a while loop to ask a user to enter a password (using `input()`), and continue to ask them for a password until the one they give you is at least 10 characters long and has at least one of the following special characters: `'!@#$%^&*()?'`.

[31]:
```
bad_pw = True

while bad_pw:
    pw = input("What password do you want to use? ")

    special_char = False
    for s in "!@#$%^&*()?":
        if s in pw:
            special_char = True

    if len(pw) > 10 and special_char:
        bad_pw = False
else:
    print("Great password")
```

```
What password do you want to use? Chelsea
What password do you want to use? ChelseaIsCool
What password do you want to use? ChelseaIsCool!
Great password
```

1.8 Range

You can use the `range()` function in order to get a sequence of integers. Range takes 3 arguments:

- `start`: which number to start at
- `stop`: which number to stop at (this number itself is not included)
- `step`: what incrememnt to step by (default is 1, `step = 2` would give you every other integer)

[32]:
```
r = range(0,100,2)

r_list = list(r) # just to print it out

print(r_list)
```

```
[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, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80,
82, 84, 86, 88, 90, 92, 94, 96, 98]
```

[33]:
```
s = range(10, 24)

s_list = list(s)

print(s_list)
```

```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
```

1.9 For Loops

For loops are a convenient way to do the same set of actions once for every item in a collection. For example, say I have a list of names `["Chelsea", "Tony", "Julie", "Sam", "Chris"]`, and for each name, I want to print out the greeting `"Hello, [name]"`. I can do that using a for loop.

The syntax is:

```
for item in collection:
    # stuff you want to do
```

[34]:
```
# print out a greeting for each person
names = ["Chelsea", "Tony", "Julie", "Sam", "Chris", "Rocco"]

for n in names:
    print("Hello, " + n)
```

```
Hello, Chelsea
Hello, Tony
```

```
Hello, Julie
Hello, Sam
Hello, Chris
Hello, Rocco
```

For loops can loop through any collection, a list, a tuple, a dictionary, or a string! For example, we can use a for loop to count the number of punctuation marks in the string `my_string` below.

[35]:
```
my_string = '''
Man:    Man!
Arthur: Man, sorry....  What night lives in that castle over there?
Man:    I'm thirty-seven!
Arthur: (suprised) What?
Man:    I'm thirty-seven!  I'm not old--
Arthur: Well I can't just call you "man"...
Man:    Well you could say "Dennis"--
Arthur: I didn't know you were called Dennis!
Man:    Well, you didn't bother to find out, did you?!
Arthur: I did say sorry about the "old woman", but from behind, you looked--
Man:    Well I object to your...you automatically treat me like an inferior!
Arthur: Well I *am* king...
Man:    Oh, king, eh, very nice.  And 'ow'd you get that, eh?
        (he reaches his destination and stops, dropping the cart)
        By exploiting the workers!  By 'angin' on to outdated imperialist dogma
        which perpetuates the economic and social differences in our society.
        If there's ever going to be any progress,--
Woman:  Dennis!  There's some lovely filth down 'ere!
        (noticing Arthur) Oh!  'Ow'd'ja do?
Arthur: How do you do, good lady.  I am Arthur, king of the Britons.  Whose
        castle is that?
Woman:  King of the 'oo?
Arthur: King of the Britons.
Woman:  'Oo are the Britons?
Arthur: Well we all are!  We are all Britons!  And I am your king.
Woman:  I didn't know we 'ad a king!  I thought we were autonomous collective.
Man:    (mad)  You're fooling yourself!  We're living in a dictatorship!  A
        self-perpetuating autocracy in which the working classes--
Woman:  There you go, bringing class into it again...
Man:    That's what it's all about!  If only people would--
Arthur: Please, *please*, good people, I am in haste!  WHO lives in that
        castle?
Woman:  No one lives there.
Arthur: Then who is your lord?
Woman:  We don't have a lord!
Arthur: (spurised) What??
Man:    I *told* you!  We're an anarcho-syndicalist commune!  We're taking
        turns to act as a sort of executive-officer-for-the-week--
Arthur: (uninterested) Yes...
```

```
Man:    But all the decisions *of* that officer 'ave to be ratified at a
        special bi-weekly meeting--
Arthur: (perturbed) Yes I see!
Man:    By a simple majority, in the case of purely internal affairs--
Arthur: (mad) Be quiet!
Man:    But by a two-thirds majority, in the case of more major--
Arthur: (very angry) BE QUIET!  I *order* you to be quiet!
Woman:  "Order", eh, 'oo does 'e think 'e is?
Arthur: I am your king!
Woman:  Well I didn't vote for you!
Arthur: You don't vote for kings!
Woman:  Well 'ow'd you become king then?
(holy music up)
Arthur: The Lady of the Lake-- her arm clad in the purest shimmering samite,
        held aloft Excalibur from the bosom of the water, signifying by
        divine providence that I, Arthur, was to carry Excalibur.  THAT is why
        I am your king!
Man:    (laughingly) Listen: Strange women lying in ponds distributing swords
        is no basis for a system of government!  Supreme executive power
        derives from a mandate from the masses, not from some... farcical
        aquatic ceremony!
Arthur: (yelling) BE QUIET!
Man:    You can't expect to wield supreme executive power just 'cause some
        watery tart threw a sword at you!!
Arthur: (coming forward and grabbing the man) Shut *UP*!
Man:    I mean, if I went 'round, saying I was an emperor, just because some
        moistened bink had lobbed a scimitar at me, they'd put me away!
Arthur: (throwing the man around) Shut up, will you, SHUT UP!
Man:    Aha!  Now we see the violence inherent in the system!
Arthur: SHUT UP!
Man:    (yelling to all the other workers) Come and see the violence inherent
        in the system!  HELP, HELP, I'M BEING REPRESSED!
Arthur: (letting go and walking away)  Bloody PEASANT!
Man:    Oh, what a giveaway!  Did'j'hear that, did'j'hear that, eh?  That's
        what I'm all about!  Did you see 'im repressing me?  You saw it,
        didn't you?!
'''

count = 0
for char in my_string:
    if char in "!@#$%^&*():?.,":
        count += 1

print("There are", count, "punctuation marks!")
```

```
There are 229 punctuation marks!
```

We can use for loops to apply pretty much any code we'd like, including functions.

[36]:
```
def prime(n):
    for i in range(2,n):
        if n%i == 0:
            return(False)
    return(True)

# prime(269)

for i in range(5,25):
    if prime(i):
        print(i, "is a prime number.")
```

```
5 is a prime number.
7 is a prime number.
11 is a prime number.
13 is a prime number.
17 is a prime number.
19 is a prime number.
23 is a prime number.
```

1.10 Your Turn

Use a for loop to loop through the words in the list below, and print out only words that have *all* the vowels in them.

[37]:
```
words = ["adieu", "adoulie" , "canard", "douleia",
         "gemstone", "holistic", "eucosia", "books",
         "parking", "justify", "boring","eulogia",
         "eunomia", "gorgeous", "blissful", "nocturnal",
         "blanket", "border","eutopia", "morose", "sided",
         "miaoued", "bountiful", "unclean", "boisterous",
         "volatile", "germane", "listless", "fantasy",
         "inkling","moineau", "fondness", "saddle", "quotient",
         "sequoia", "somber", "soliloqy", "petrified",
         "jubilant", "docile","suoidea"]

 ### YOUR CODE HERE ###

for word in words:
    if "a" in word and "e" in word and "i" in word and "o" in word and "u" in␣
 ↪word:
        print(word)
```

```
adoulie
douleia
eucosia
eulogia
```

```
eunomia
eutopia
miaoued
moineau
sequoia
suoidea
```

2 Functions

If you need to review: Video 1, Video 2

Functions are a way to write multi-use code that can be called over and over in different circumstances. Functions are groups/suites of code that perform a specific calculation given variable inputs. Functions allow us to change the inputs we are operating on through *arguments* which are given to the function when you define and call the function. For example, in the code below which defines the function `square()`, the argument `n` tells us which number to square:

[38]:
```
def square(n):
    return(n*n)
```

2.1 Arguments

The argument `n` is a variable which will hold whatever number we give the function when calling it. For example, if I want to square the number 7 I would use this code:

[39]:
```
square(7)
```

[39]:
```
49
```

in this case, `n` will be equal to `7`, and every time we reference `n` in the function code, we will use `7`. We could also use other numbers. For example:

[40]:
```
square(n = 12)
```

[40]:
```
144
```

Notice that you can explicitly tell python which argument you're setting when calling the function by saying the name of the argument (here: `n`) `=` the value for that argument (here: `12`).

2.2 Default Arguments

When *defining* a function, you can set *default arguments*, which are values that the function can use if the user does not provide values for that specific argument. For example, in our `square()` function, we can specify that if the user does not provide a value for `n`, then the function should use `n = 1`. We set default arguments by giving the argument a value when *defining* the function.

[41]:
```
def square(n = 1):
    return(n*n)
```

When defining functions, arguments with NO default must come before arguments with a default. For example in this function that multiplies two numbers, `a` and `b`, the argument `a` must appear first in the parentheses when defining `mult()` because it does NOT have a default value, while `b` does.

[42]:
```
def mult(a, b = 2):
    return(a * b)
```

Default arguments allow users to call the function without spefifying a value for that argument. For example if we called `square()` as defined in this section *without* an argument, it would return `1`, because it uses the default value of 1 when no value for `n` is given.

2.3 Calling vs. Defining Functions

When you define a function, you're giving python instructions about what to do *if* you ever call that function. This is why when you write your `def` statement, and run the code, nothing actually outputs. This is because when you run a function definition, you are just asking python to *store* the directions for later. To actually execute the function, you need to *call* it.

For example when you run the below cell. Nothing will output.

[43]:
```
# FUNCTION DEFINITION

def censorDang(sentence):
    # this function takes in a sentence as a string and returns the same
    # sentence censored for any occurance of the word "dang".

    sentence_list = sentence.split()

    for i in range(0,len(sentence_list)):
        if sentence_list[i].lower() == "dang":
            sentence_list[i] = "****"

    return(" ".join(sentence_list))
```

But when you call the function in the following cell, python will actually execute the code and there will be an output.

[44]:
```
## FUNCTION CALL

censorDang("Dang this is good tea.")
```

[44]:
```
'**** this is good tea.'
```

Remember that when you're writing a function, the arguments are just placeholders or variables. They don't refer to specific objects/values, they're meant to be malleable. So for example, when I wrote `censorDang()`, I didn't write it with a *specific* sentence in mind. The function should work for any sentence!

2.4 Your Turn

Write a function, `max_list()` that takes in a list of numbers as an argument, and returns the maximum value of that list.

[45]:
```
###

def max_list(LoN):
    return(max(LoN))

max_list([1,4,7,8,2])
```

[45]: 8

Write a function, `q_finder()` that takes in a list of words/strings as an argument, and returns a list of only the words that contain q (upper OR lower case).

[46]:
```
###

def q_finder(listOfWords):
    return([word for word in listOfWords if "q" in word.lower()])

q_finder(["Quality", "corn", "quibble", "cash"])
```

[46]: ['Quality', 'quibble']

Write a function `cluster_subsetter()` that takes in a data frame (see example below), `df`, and a string `cluster`, as arguments, and returns a data frame with only the rows who are in the cluster specified by `cluster`.

[47]:
```
d = pd.DataFrame({"x": np.random.uniform(0,1,size = 100),
                  "y" : np.random.uniform(0,1,size = 100),
                  "cluster" : np.repeat(["A", "B", "C"], [50,20,30]) })
d
# cluster_subsetter(d,"A") should return only the rows in d that belong in␣
 ↪cluster "A"


###

def cluster_subsetter(df, cluster):
    return(df.loc[df["cluster"] == cluster])

cluster_subsetter(d, "A")
```

[47]:

	x	y	cluster
0	0.119375	0.337845	A
1	0.480236	0.359271	A
2	0.451347	0.971585	A

3	0.452016	0.639311	A
4	0.704587	0.192349	A
5	0.258840	0.499453	A
6	0.946911	0.162296	A
7	0.803587	0.884005	A
8	0.992570	0.641869	A
9	0.412359	0.518520	A
10	0.648073	0.818125	A
11	0.961328	0.362286	A
12	0.539202	0.418590	A
13	0.724192	0.126564	A
14	0.402266	0.683642	A
15	0.080241	0.754872	A
16	0.509532	0.298628	A
17	0.620953	0.108404	A
18	0.129963	0.298615	A
19	0.978382	0.797339	A
20	0.741741	0.516270	A
21	0.807354	0.055844	A
22	0.475685	0.016685	A
23	0.516708	0.340385	A
24	0.814093	0.730600	A
25	0.688325	0.826695	A
26	0.300403	0.306491	A
27	0.260713	0.319587	A
28	0.634381	0.328056	A
29	0.022640	0.271607	A
30	0.178207	0.289820	A
31	0.325504	0.712965	A
32	0.649613	0.075475	A
33	0.871375	0.924520	A
34	0.429519	0.801397	A
35	0.322820	0.519778	A
36	0.350376	0.897700	A
37	0.850320	0.128451	A
38	0.912997	0.005307	A
39	0.871974	0.085300	A
40	0.486586	0.637278	A
41	0.416044	0.608216	A
42	0.837774	0.518610	A
43	0.912786	0.294243	A
44	0.200500	0.345339	A
45	0.574262	0.136371	A
46	0.272503	0.791564	A
47	0.836534	0.457779	A
48	0.099508	0.295595	A
49	0.647251	0.031044	A

3 List Comprehension

3.1 List Comp Intro

List comprehension is a way to create lists using iteration, essentially as an alternative to using a for loop.

If a for loop looks like this:

[48]:
```
my_list = []

for i in range(0,10):
    my_list.append(i**2)
```

then the list comprehension would look like this:

[49]:
```
my_list2 = [i**2 for i in range(0,10)]
```

3.2 Other examples:

3.2.1 making a list of all lower case letters in a string

[50]:
```
# for loop
censored_list = []

s = "The rain in Spain falls mainly in the plains"

for letter in s:
    censored_list.append(letter.lower())
```

[51]:
```
# list comp

censored_list2 = [letter.lower() for letter in s]

censored_list == censored_list2
```

[51]:
```
True
```

3.2.2 calculating factorials for a bunch of numbers

[52]:
```
# for loop

factorials = []

def factorial(n):
    mult = range(1,n+1)
    p = 1
    for i in mult:
        p = p * i
```

```
    return(p)

n = [2,5,6,10,144]
fac = []
for num in n:
    fac.append(factorial(num))
```

[53]:
```
# list comp

fac2 = [factorial(num) for num in n]

fac == fac2
```

[53]: True

3.2.3 multiplying all possible combos of items from two lists together

You can even do combine 2 for loops into 1 list comprehension!

[54]:
```
# for loop

a = [1,2,3,4,5]
b = [6,7,3,4,-1]

mults = []

for i in a:
    for j in b:
       mults.append(i*j)
```

[55]:
```
# list comp

mult2 = [i*j for i in a for j in b]

mult == mult2
```

[55]: False

3.2.4 flattening a list of lists into a single list

[56]:
```
# for loop

a = [[1,2,3,4,42,4,3,2],[4,2,3,9,5,83,8,2,9,0,3], [4,8,15,16,23,42]]

newList = []
for sub in a:
    for i in sub:
```

```
        newList.append(i)
```

[57]:
```
# list comp
newList2 = [i for sub in a for i in sub]

newList == newList2
```

[57]: True

3.2.5 making a list of all possible playing cards

[58]:
```
suits = ["Hearts", "Spades", "Diamond", "Clubs"]
cards = ["A","K","Q","J", "10", "9", "8", "7", "6", "5", "4", "3", "2"]

deck = []
for suit in suits:
    for card in cards:
        deck.append(suit+card)


#list comp

deck2 = [card + " of " + suit for suit in suits for card in cards]

print(deck2)

deck == deck2
```

```
['A of Hearts', 'K of Hearts', 'Q of Hearts', 'J of Hearts', '10 of Hearts', '9
of Hearts', '8 of Hearts', '7 of Hearts', '6 of Hearts', '5 of Hearts', '4 of
Hearts', '3 of Hearts', '2 of Hearts', 'A of Spades', 'K of Spades', 'Q of
Spades', 'J of Spades', '10 of Spades', '9 of Spades', '8 of Spades', '7 of
Spades', '6 of Spades', '5 of Spades', '4 of Spades', '3 of Spades', '2 of
Spades', 'A of Diamond', 'K of Diamond', 'Q of Diamond', 'J of Diamond', '10 of
Diamond', '9 of Diamond', '8 of Diamond', '7 of Diamond', '6 of Diamond', '5 of
Diamond', '4 of Diamond', '3 of Diamond', '2 of Diamond', 'A of Clubs', 'K of
Clubs', 'Q of Clubs', 'J of Clubs', '10 of Clubs', '9 of Clubs', '8 of Clubs',
'7 of Clubs', '6 of Clubs', '5 of Clubs', '4 of Clubs', '3 of Clubs', '2 of
Clubs']
```

[58]: False

3.2.6 Words with e's

You can also include boolean statements like if/else in your list comprehension.

[59]:
```
words = ["Hello", "Mother", "hello", "father", "fleas", "ticks", "mosquitos",␣
↪"really", "bother"]

es = []

for word in words:
    if "e" in word.lower():
        es.append(word)
```

[60]:
```
es2 = [word for word in words if "e" in word.lower()]

es == es2
```

[60]:
```
True
```

3.2.7 Using list comp with sklearn

You can use list comprehension with all sorts of functions. For example, you can use it to create a bunch of KMeans models and calculate their silhouette scores.

[61]:
```
data = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/
↪CPSC392ParlettPelleriti/master/Data/programmers2.csv")

kmod = KMeans()

kms = [KMeans(n_clusters = k).fit(data) for k in range(2,10)]

silhouettes = [silhouette_score(data,model.predict(data)) for model in kms]

# silhouettes = [silhouette_score(data,KMeans(n_clusters = k).
↪fit_predict(data)) for k in range(2,10)]

print(silhouettes)

print("\nThe maximum silhouette score is: ", max(silhouettes))
```

```
[0.4342031414710615, 0.5368423138393504, 0.6513061716958818, 0.616849785320789,
0.5481112041018308, 0.5074774704085537, 0.4127864224109295, 0.36427034924130064]

The maximum silhouette score is:  0.6513061716958818
```

3.3 Your Turn

use the `prime()` function below, as well as list comprehension to create a list called `primes` that contains all the prime numbers between 3 and 1000.

[62]:
```
def prime(n = 10):
    if n < 2:
```

```
        return(False)
    if n == 2:
        return(True)
    for div in range(2,n):
        if n%div == 0:
            return(False)
    return(True)

primes = [i for i in range(3,1001) if prime(i)]###
primes
```

[62]:
```
[3,
 5,
 7,
 11,
 13,
 17,
 19,
 23,
 29,
 31,
 37,
 41,
 43,
 47,
 53,
 59,
 61,
 67,
 71,
 73,
 79,
 83,
 89,
 97,
 101,
 103,
 107,
 109,
 113,
 127,
 131,
 137,
 139,
 149,
 151,
 157,
```

```
163,
167,
173,
179,
181,
191,
193,
197,
199,
211,
223,
227,
229,
233,
239,
241,
251,
257,
263,
269,
271,
277,
281,
283,
293,
307,
311,
313,
317,
331,
337,
347,
349,
353,
359,
367,
373,
379,
383,
389,
397,
401,
409,
419,
421,
431,
433,
```

```
439,
443,
449,
457,
461,
463,
467,
479,
487,
491,
499,
503,
509,
521,
523,
541,
547,
557,
563,
569,
571,
577,
587,
593,
599,
601,
607,
613,
617,
619,
631,
641,
643,
647,
653,
659,
661,
673,
677,
683,
691,
701,
709,
719,
727,
733,
739,
```

```
 743,
 751,
 757,
 761,
 769,
 773,
 787,
 797,
 809,
 811,
 821,
 823,
 827,
 829,
 839,
 853,
 857,
 859,
 863,
 877,
 881,
 883,
 887,
 907,
 911,
 919,
 929,
 937,
 941,
 947,
 953,
 967,
 971,
 977,
 983,
 991,
 997]
```

Use list comprehension to turn this list of numbers, into a list of strings (for example if the list is [1,2,3] you want to return ["1", "2", "3"]).

[63]:
```
nums = [1,1,3,5,8,13]
string_nums = [str(i) for i in nums]

string_nums###
```

[63]:
```
['1', '1', '3', '5', '8', '13']
```

Use list comprehension to create a list of ONLY words from `sentence` that have an even number of letters.

[64]:
```
sentence = "Lorem ipsum dolor sit amet consectetur adipiscing elit sed do␣
 ↪eiusmod tempor incididunt ut labore et dolore magna aliqua Ut enim ad minim␣
 ↪veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea␣
 ↪commodo consequat Duis aute irure dolor in reprehenderit in voluptate velit␣
 ↪esse cillum dolore eu fugiat nulla pariatur Excepteur sint occaecat␣
 ↪cupidatat non proident sunt in culpa qui officia deserunt mollit anim id est␣
 ↪laborum"

even_words = [word for word in sentence.split() if len(word)%2 == 0]###
even_words
```

[64]:
```
['amet',
 'adipiscing',
 'elit',
 'do',
 'tempor',
 'incididunt',
 'ut',
 'labore',
 'et',
 'dolore',
 'aliqua',
 'Ut',
 'enim',
 'ad',
 'veniam',
 'quis',
 'exercitation',
 'nisi',
 'ut',
 'ex',
 'ea',
 'Duis',
 'aute',
 'in',
 'in',
 'esse',
 'cillum',
 'dolore',
 'eu',
 'fugiat',
 'pariatur',
 'sint',
 'occaecat',
 'proident',
```

```
 'sunt',
 'in',
 'deserunt',
 'mollit',
 'anim',
 'id']
```

4 Pandas Data Frames

Data frames are a way to store data in python. It's similar to a single spreadsheet which contains rows (observations) and columns (features).

You can grab the size of the data frame with `.shape` which gives you the number of rows, and columns in the data frame.

[65]:
```
data = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/
↪CPSC392ParlettPelleriti/master/Data/KMEM4.csv")

data.shape
```

[65]:
```
(600, 2)
```

To see a small # of rows of your data frame, use `.head()`

[66]:
```
data.head()
```

[66]:
```
          x         y
0 -0.006848  0.395527
1  0.314820 -0.289261
2  0.171705  1.078077
3 -1.203661  1.325926
4 -0.179379 -0.036615
```

To grab columns from a data frame you can use multiple ways:

[67]:
```
data.x
```

[67]:
```
0     -0.006848
1      0.314820
2      0.171705
3     -1.203661
4     -0.179379
         ...
595    4.189792
596   -4.249038
597    3.411690
598    3.629434
599   -4.904774
```

```
Name: x, Length: 600, dtype: float64
```

[68]:
```
data["x"]
```

[68]:
```
0     -0.006848
1      0.314820
2      0.171705
3     -1.203661
4     -0.179379
         …
595    4.189792
596   -4.249038
597    3.411690
598    3.629434
599   -4.904774
Name: x, Length: 600, dtype: float64
```

[69]:
```
data.iloc[:,0]
```

[69]:
```
0     -0.006848
1      0.314820
2      0.171705
3     -1.203661
4     -0.179379
         …
595    4.189792
596   -4.249038
597    3.411690
598    3.629434
599   -4.904774
Name: x, Length: 600, dtype: float64
```

[70]:
```
data.loc[:, "x"]
```

[70]:
```
0     -0.006848
1      0.314820
2      0.171705
3     -1.203661
4     -0.179379
         …
595    4.189792
596   -4.249038
597    3.411690
598    3.629434
599   -4.904774
Name: x, Length: 600, dtype: float64
```

To acces rows from a data frame, we'll often use `.loc[]` or `.iloc[]`. You can remember the difference by telling yourself that the `i` in `.iloc[]` stands for integer/index, because `.iloc[]` takes indices/integers, whereas `.loc[]` can take booleans, and labels/strings.

Let's use `.loc[]` to grab rows 19-25 (assuming first row is 0) from `data`.

[71]:
```
data.iloc[19:26,] # remember it STARTS from the first number and goes up to BUT␣
↪NOT INCLUDING the second
```

[71]:
```
           x         y
19 -0.746738 -0.440733
20  1.524822 -1.256318
21 -0.728119 -0.015010
22  0.914411 -1.748031
23 -0.491472  1.115153
24  0.597467 -1.437617
25 -0.875947  0.582450
```

Now let's grab only the rows where x $>$ 3, and y $>$ 3.

[72]:
```
gt3 = (data.x > 3) & (data.y > 3)

data.loc[gt3]
```

[72]:
```
            x         y
128  3.457686  3.200131
140  3.408910  3.062241
201  3.067726  3.221080
272  3.711039  3.285028
313  3.141992  3.761917
361  3.706717  3.030354
422  3.486926  3.055562
436  3.002480  3.544937
471  3.012580  3.290013
554  3.275193  3.012374
583  3.313295  3.404577
592  3.071065  3.151695
```

There are TONS of useful data frame functions, so I'll demonstrate just a few:

[73]:
```
# grab the mean of columns

data.mean()
```

[73]:
```
x   -0.125429
y    0.695387
dtype: float64
```

[74]:
```
# grab the max of the columns

data.max()
```

[74]:
```
x    4.976464
y    4.979621
dtype: float64
```

[75]:
```
# what columns are in my df?

data.columns
```

[75]:
```
Index(['x', 'y'], dtype='object')
```

[76]:
```
# drop missing values

data = data.dropna()
```

[77]:
```
# groupb data frame by cluster assignment and then get the mean for each cluster

prog = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/
 ↪CPSC392ParlettPelleriti/master/Data/programmers.csv")
prog["Assignment"] = np.repeat(["A", "B", "C", "D", "E"],50)

prog.groupby("Assignment").mean() # rows are each cluster, columns represent␣
 ↪the different features
```

[77]:

Assignment	py	r	c	sql	js
A	89.750298	75.704910	18.941242	24.833986	5.055579
B	69.799266	69.569211	10.891900	64.164052	9.186321
C	96.017893	21.015196	87.057530	22.728009	31.075785
D	18.554134	22.925694	29.934704	33.196274	18.982596
E	63.830289	20.361404	14.394041	87.165838	92.782235

4.1 Your Turn

Using the pandas skills you've learned in class and reviewed here, what is the mean dancibility for each artist in the `popDivas` dataset? Who has the highest average danceability?

[78]:
```
popDivas = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/
 ↪CPSC392ParlettPelleriti/master/Data/PopDivas_data.csv")
popDivas.head()
```

[78]:

	Unnamed: 0	artist_name	danceability	energy	key	loudness	mode	\
0	1	Beyoncé	0.386	0.28800	1	-18.513	1	
1	2	Beyoncé	0.484	0.36300	5	-8.094	0	

```
2           3     Beyoncé         0.537  0.24700    2   -17.750     1
3           4     Beyoncé         0.672  0.69600    4    -6.693     0
4           5     Beyoncé         0.000  0.00515    9   -22.612     0

   speechiness  acousticness  instrumentalness  liveness  valence  \
0       0.0602         0.533           0.01670    0.1410    0.399
1       0.0368         0.645           0.00000    0.1250    0.201
2       0.0793         0.199           0.00001    0.4230    0.170
3       0.1770         0.200           0.02750    0.0736    0.642
4       0.0000         0.524           0.95000    0.1140    0.000

   duration_ms                    track_name
0        43850   balance (mufasa interlude)
1       226479                        BIGGER
2        46566  the stars (mufasa interlude)
3       162353           FIND YOUR WAY BACK
4        13853   uncle scar (scar interlude)
```

[79]:
```
###
artists = pd.unique(popDivas.artist_name)

[popDivas.loc[popDivas.artist_name == artist, "danceability"].mean() for artist␣
↪in artists]
```

[79]:
```
[0.5675717299578059,
 0.719620253164557,
 0.5856818181818182,
 0.5941027397260275,
 0.6265100286532952,
 0.6129830508474575]
```

Grab only the songs that are by Beyonce or Britney Spears, and have an energy score above 0.5.

[80]:
```
popDivas.loc[((popDivas.artist_name == "Britney Spears") | (popDivas.
↪artist_name == "Beyoncé")) & (popDivas.energy > 0.5)]###
```

[80]:
```
     Unnamed: 0     artist_name  danceability  energy  key  loudness  mode  \
3             4         Beyoncé         0.672   0.696    4    -6.693     0
5             6         Beyoncé         0.932   0.775    7    -5.345     0
7             8         Beyoncé         0.790   0.693    0    -5.767     1
11           12         Beyoncé         0.608   0.709    7    -6.175     1
12           13         Beyoncé         0.000   0.558    0   -17.238     1
..          …               …             …       …    …         …   …
546         547  Britney Spears         0.847   0.801    9    -3.838     1
547         548  Britney Spears         0.765   0.791    8    -5.707     1
548         549  Britney Spears         0.732   0.888    7    -2.415     1
550         551  Britney Spears         0.810   0.924    0    -4.020     0
```

```
551         552  Britney Spears         0.693   0.625    7    -7.147     1

     speechiness  acousticness  instrumentalness  liveness  valence  \
3         0.1770        0.2000          0.027500    0.0736    0.642
5         0.1150        0.0184          0.015700    0.3180    0.584
7         0.0605        0.0420          0.026700    0.0790    0.855
11        0.3740        0.1190          0.000000    0.3370    0.709
12        0.0000        0.7560          0.000000    0.5330    0.000
..           …             …                 …       …       …
546       0.0564        0.0876          0.000204    0.7660    0.841
547       0.0317        0.2620          0.000154    0.0669    0.966
548       0.0792        0.0702          0.000003    0.3790    0.674
550       0.0356        0.1190          0.000087    0.1490    0.868
551       0.0431        0.0553          0.000000    0.6590    0.439

     duration_ms                          track_name
3         162353                FIND YOUR WAY BACK
5         155990                  DON'T JEALOUS ME
7         190108                            JA ARA E
11        272068   MOOD 4 EVA (feat. Oumou Sangaré)
12          8619  reunited (nala & simba interlude)
..           …                                 …
546       216706         What U See (Is What U Get)
547       206226                              Lucky
548       203280                 One Kiss from You
550       197186             Can't Make You Love Me
551       269773              When Your Eyes Say It

[416 rows x 14 columns]
```

5 Math with Arrays

numpy arrays allow us to do *vectorized* operations. *Vectorized* operations are applied elementwise to each item in an array, rather than to the array as a whole. For example, if we want to get the square of every number in an array, we can say `array**2`. You can see below, that calling `**2` on the array `x` squares each item in `x`.

[81]:
```
x = np.array([1,2,3,4,5,10])

x**2
```

[81]:
```
array([  1,   4,   9,  16,  25, 100])
```

Similarly we can subtract one array from another `a - b`, which will substract the first element of `b` from the first element of `a`, etc.

[82]:
```
a = np.array([1,2,3,4,5])
b = np.array([1,4,-2,5,9])

a-b
```

[82]:
```
array([ 0, -2,  5, -1, -4])
```

5.1 Your Turn

Using the array knowledge you just reviewed, multiply the arrays `a` and `b` together and then find the sum of those producs.

[83]:
```
###

np.sum(a*b)
```

[83]:
```
68
```

6 np.random

`np.random` is a great package that allows us to generate random values from different distributions, or randomly choose items from a collection. The two most common functions we use are `np.random.choice()` and `np.random.normal()`.

`np.random.choice()` takes 3 main arguments:

- `a`: an array or collection of items to choose from.
- `size`: an integer that represents how many items you want to choose/sample from `a`
- `replace`: a boolean that tells you whether or not to allow the function to select an item more than once in the sample.

`np.random.normal()` takes 3 main arguments as well:

- `loc`: the mean of the normal distribution to sample from
- `scale`: the standard deviation of the normal distribution to sample from
- `size`: the number of samples to draw.

[84]:
```
# draw 100 samples from a standard normal distribution with mean = 0, sd = 1

samp100 = np.random.normal(0,1,100)

samp100
```

[84]:
```
array([-0.9249086 ,  0.22624359,  0.10151563, -1.47050774,  1.42077839,
        0.4164867 , -0.03360924, -1.38402279,  1.30562102, -0.60438246,
        0.36310946,  0.87600058,  0.12143398,  0.85053506, -0.39886356,
        1.24534215, -1.22670405, -0.16739495, -0.56942622,  0.19431748,
```

```
 0.43219226,  0.20064156,  1.62689381,  0.19407179,  0.48949367,
-0.91052619,  1.18437835, -0.45489467, -0.07662868, -0.69392268,
 0.23210336,  0.71185407, -0.50854367, -1.83981923,  1.54515686,
-0.7609373 , -0.26284067, -1.3814667 , -0.17037072, -0.371339  ,
 0.21222714, -0.94389513, -1.35533745,  2.06393605, -0.01483079,
 0.90637742, -0.34034663, -0.14067677, -0.31749206, -0.5441076 ,
 1.5508507 , -0.99182545,  0.2598589 ,  1.23610951, -0.12765486,
 0.01576106,  1.29960446,  0.82762587,  1.02530936, -0.78033154,
 0.47771163, -1.64573924,  0.15551424,  0.4998344 ,  0.94968117,
-1.09136235, -0.71690682, -1.78032669, -1.12003489, -2.26314402,
-1.2261685 , -0.39969833,  0.34893421,  0.33577726,  3.21551896,
 0.75545576,  0.00820206, -0.09405116,  1.13559607,  0.95607149,
 1.80172372,  0.85719888,  0.41830383,  1.16349734,  0.32567506,
 0.90549987,  0.42733278,  0.69941025, -0.31275257, -0.65734395,
 1.12831139, -0.57340049, -0.3046446 ,  0.87359815,  0.16557342,
-0.46328489, -0.09282863,  1.03112328,  0.25058586,  2.25508285])
```

[85]:
```
# draw 657 samples from `my_list` with replacement

my_list = range(0,250)

samp657 = np.random.choice(my_list, 657, replace = True)
samp657
```

[85]:
```
array([ 10,  43, 163, 144, 138,  88,  53,   2, 191, 107, 178, 102,  79,
       243, 177, 191, 243, 146, 183, 214, 164, 132, 135, 223, 245,  87,
       195, 152,  20, 148, 191,  32, 122, 161, 240,  56, 214,  19, 127,
        35,  27, 218, 157, 221, 202, 132,  45, 195, 128, 152,  80, 208,
        30,  98,  58,  98,   3, 211, 136,  86, 121, 187, 200, 172,  91,
       185, 112,  57,  76,  51,  62,  37, 232, 237,  44, 186, 220, 180,
       161,  32,  29, 149, 124, 101,  55,  20, 174, 177,  83, 148, 124,
         7,  67,  67, 105, 169,  62, 219,  15, 114,  34,  72,  85, 131,
        38,  79, 175,  68,  51,  70, 144,  73,  18, 227, 234, 230, 141,
        86, 177, 137,  69,  65,  52, 121, 181,  31,  44,  98, 121,  11,
        71, 148, 161,  62, 202,  70,  89, 227,   6, 241, 148, 157,  29,
       226,  88, 122, 225,  79, 106, 137, 199, 174, 195,  39, 206, 195,
       240, 140, 142,  20, 201,  88,  77,  38,  55,   7, 242, 202, 103,
        76, 144,  54,  96,  10, 170, 182, 248, 182,  15, 168,  63,   1,
        93, 181,  19, 153, 114, 228, 211,  36, 109, 183,  37, 247,  71,
        18,  23,  99, 163,  89, 168, 113,  44, 107, 147, 113, 183, 158,
        14, 148, 191,  77,  37, 146,  69,  10, 204,   5,  92, 218,  54,
       225,  91, 155, 236,  12, 135,  53, 169, 114, 143, 101,  78, 224,
        35, 214,  30, 227, 172,  16, 240, 207, 235, 237,  99, 172,  28,
       144,  43,  18,  22,  42, 193,  88, 226,  47, 139,  60, 106, 227,
       166, 235,  64, 171,  48, 180, 130,  93,  19, 138, 128, 156,  49,
       144, 188, 125, 100, 124,  24, 166, 229, 247, 233, 157,  58, 137,
       222, 136,  56,  54,  30, 243, 138, 217, 191,  40,  52,   1, 148,
```

```
         86, 161, 199, 157, 214, 237, 235, 110, 128,  93,  28,  43,  87,
        158,  32,  38,  91, 139, 205,  78,  47,  41,  75, 188, 208, 213,
         74, 184, 156, 103, 217,  94, 160, 244,  90, 120,  49,  36, 196,
        140, 150, 245,  28, 225,  11, 104,  83,  30,  61, 182,  24, 213,
         44, 183, 150,   9,  23, 130,  21,   0, 105, 107, 174, 164, 151,
         87,  66,   9, 238, 124,  56, 178, 155,  29,  29, 202, 212,  70,
         13, 202, 227, 175, 153, 248,  31, 109,  16, 144, 100,   5, 118,
         27, 175,  16, 229, 115, 113, 166,  99,  28, 243, 131,  95, 208,
        146,  41, 155, 113, 221,  24,  75, 116, 199, 162, 155,  69, 176,
         75,  92,  82, 237, 174, 163, 132,   6, 125,  99, 143, 153,   2,
        168, 248, 130,  14,  98, 217,  81, 128,  71,  85, 149, 192,  91,
        145, 211, 120, 202,   7,  13, 105, 198, 199, 137,  29, 135, 116,
        101, 154,  91,  54, 236, 216,   9, 218, 125,  99, 160, 246,  55,
        110, 187, 231,  30,  62, 104, 198, 123, 123, 115, 111, 113, 201,
        219, 206, 235, 143,  48, 157,  57,  76,  34, 190, 122, 182, 147,
        222, 200,  32, 114, 146, 241,  80,  47, 128, 247,  42, 239,  12,
        240, 119,  37,  97, 144, 216, 149, 238, 153, 185,  18, 140,   9,
        152, 123, 143,  63,  39,  60,  29, 151, 103,  91, 116, 216, 241,
         44, 179,  34, 243, 203, 149,  72, 234,  14, 246, 161, 247, 226,
         48,  15, 202, 137,  10, 133, 205, 124,  75, 152, 220, 214, 246,
        127,  47,  80,  77, 180, 110, 236, 113,  17, 222,  24, 123,  49,
        169, 174,  47,  64, 203, 182, 213, 217, 171,  35, 109, 115,  21,
        194, 144, 212, 243, 180,  61, 188,  84,  26,  41, 174, 109, 216,
        191,  32,  36, 228,  93,  96,  44,  48, 133, 248, 142,   3, 121,
        197, 225,  52,  79,  44, 209, 138,  58, 116,  50,  61, 164,  37,
        238,  23,  83, 105, 136,  38,  24, 158,  74, 106, 217, 213, 203,
        217,  53,  90, 201, 102,  28, 239, 189,  46,   6,  74,  75, 203,
        191,  18,  50, 152,  53, 248, 146])
```

[86]:
```
for i in range(0,10):
    np.random.seed(123)
    print(list(np.random.choice(range(0,1000), 10, replace = True)))
```

```
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
[510, 365, 382, 322, 988, 98, 742, 17, 595, 106]
```

6.1 Your Turn

Choose 100 samples from `range(100, 1000)` without replacement.

[87]:
```
###
np.random.choice(range(100,1000), size = 100)
```

[87]:
```
array([223, 669, 314, 837, 196, 213, 738, 147, 173, 644, 324, 211, 509,
       439, 946, 353, 520, 708, 308, 168, 917, 923, 551, 102, 440, 139,
       422, 696, 659, 604, 276, 235, 973, 199, 480, 960, 280, 458, 965,
       313, 730, 962, 511, 390, 688, 780, 999, 206, 937, 676, 943, 518,
       926, 494, 890, 817, 246, 584, 371, 511, 865, 258, 280, 682, 465,
       471, 254, 820, 490, 882, 355, 344, 459, 229, 655, 286, 457, 795,
       637, 534, 924, 405, 880, 630, 565, 613, 151, 985, 845, 404, 156,
       442, 615, 167, 239, 761, 573, 715, 589, 454])
```

Make a data frame with two columns, `x` and `y`. `x` should be created by randomly sampling 100 samples from a normal distribution with mean = 0, and sd = 1. `y` should be created by randomly sampling 100 samples from a normal distribution with mean = 12, sd = 20.

[88]:
```
###
x = np.random.normal(loc = 0, scale = 1, size = 100)
y = np.random.normal(loc = 12, scale = 20, size = 100)

df = pd.DataFrame({"x": x,
                   "y": y})

df.head()
```

[88]:

	x	y
0	1.799889	6.256882
1	1.238366	-8.638000
2	0.913757	44.426469
3	0.466268	-27.449135
4	0.125040	7.186410