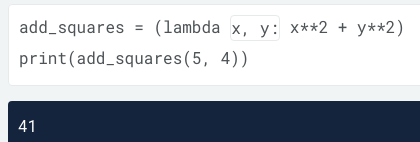
Continue with <https://campus.datacamp.com/courses/statistical-thinking-in-python-part-2/parameter-estimation-by-optimization?ex=11>

<https://repl.it/@HTran/PythonIdiom>

**<https://campus.datacamp.com/courses/statistical-thinking-in-python-part-2/parameter-estimation-by-optimization?ex=4>**

**EDA exploratory data analysis.**

****

**29-6-19 4/5 30mins**

**Idioms:**

[**https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/graphical-exploratory-data-analysis?ex=11**](https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/graphical-exploratory-data-analysis?ex=11)

**# 2. Test for “truthy” and “falsy”**

**# values**

**# if x:**

**# if not x:**

**name = 'Safe'**

**pets = ['Dog', 'Cat', 'Hamster']**

**owners = {'Safe': 'Cat', 'George': 'Dog'}**

**name = input()**

**#Good**

**if name and pets and owners:**

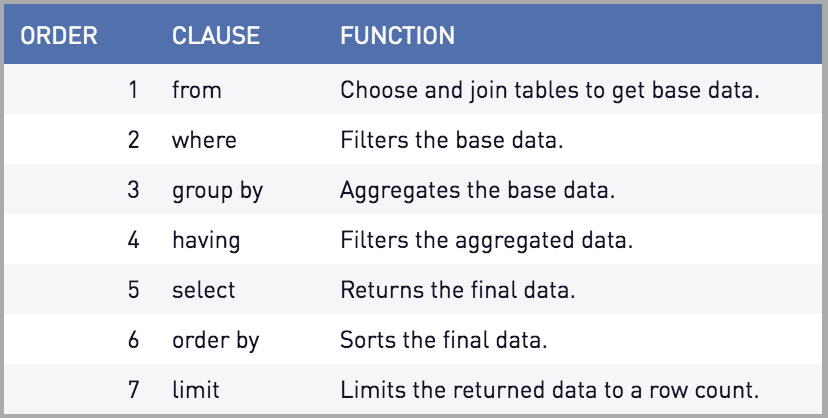
**print('We have pets!')**

**# NOT SO GOOD**

**# if name != '' and len(pets) > 0 and owners != {}:**

**# print('We have pets!')**

**29-6-19 4/5 30mins**

****

**Consider the SQL SELECT statement syntax:**

**SELECT DISTINCT *<TOP\_specification> <select\_list>***

**FROM *<left\_table>***

***<join\_type>* JOIN *<right\_table>***

**ON *<join\_condition>***

**WHERE *<where\_condition>***

**GROUP BY *<group\_by\_list>***

**HAVING *<having\_condition>***

**ORDER BY *<order\_by\_list>***

**In SQL, the first clause that is processed is the FROM clause, while the SELECT clause, which appears first in an SQL query, is processed much later. The phases involved in the logical processing of an SQL query are as follows:**

1. **FROM clause**
2. **ON clause**
3. **OUTER clause**
4. **WHERE clause**
5. **GROUP BY clause**
6. **HAVING clause**
7. **SELECT clause**
8. **DISTINCT clause**
9. **ORDER BY clause**
10. **LIMIT or OFFSET clause**

**in SQL, aggregate functions can't be used in WHEREclauses. For example, the following query is invalid:**

**SELECT release\_year**

**FROM films**

**GROUP BY release\_year**

**WHERE COUNT(title) > 10;**

**This means that if you want to filter based on the result of an aggregate function, you need another way! That's where the HAVING clause comes in. For example,**

**SELECT release\_year**

**FROM films**

**GROUP BY release\_year**

**HAVING COUNT(title) > 10;**

**shows only those years in which more than 10 films were released.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_**

select country, sum(budget)

from films

Where budget IS NOT NULL

group by 1

order by 2 DESC;

**29-6-19 ⅗ 30mins.**

**Select \***

**from films**

**Where release\_year NOT IN (2000, 2012)**

**order by duration;**

**Select title**

**from films**

**Where release\_year = 2000 or release\_year = 2012**

**order by release\_year;**

**Or**

**Where release\_year IN (2000, 2012)**

<https://campus.datacamp.com/courses/python-data-science-toolbox-part-2/bringing-it-all-together-3?ex=16>

**29-6-19 2/5 30mins (session 1 35 minutes and also yesterday did a bit)**

Note that since a file object is already a generator, you don't have to explicitly create a generator object with your read\_large\_file()function. However, it is still good to practice how to create generators - well done!

# Define read\_large\_file()

def read\_large\_file(file\_object):

"""A generator function to read a large file lazily."""

# Loop indefinitely until the end of the file

while True:

# Read a line from the file: data

data = file\_object.readline()

# Break if this is the end of the file

if not data:

break

# Yield the line of data

yield data

# Open a connection to the file

with open('world\_dev\_ind.csv') as file:

# Create a generator object for the file: gen\_file

gen\_file = read\_large\_file(file)

# Print the first three lines of the file

print(next(gen\_file))

print(next(gen\_file))

print(next(gen\_file))

# **Writing a generator to load data in chunks (2)**

**In the previous exercise, you processed a file line by line for a given number of lines. What if, however, you want to do this for the entire file?**

**In this case, it would be useful to use generators. Generators allow users to** [***lazily evaluate* data**](http://www.blog.pythonlibrary.org/2014/01/27/python-201-an-intro-to-generators/)**. This concept of *lazy evaluation* is useful when you have to deal with very large datasets because it lets you generate values in an efficient manner by *yielding* only chunks of data at a time instead of the whole thing at once.**

**In this exercise, you will define a generator function read\_large\_file() that produces a generator object which yields a single line from a file each time next()is called on it. The csv file 'world\_dev\_ind.csv' is in your current directory for your use.**

**Note that when you open a connection to a file, the resulting file object is already a generator! So out in the wild, you won't have to explicitly create generator objects in cases such as this. However, for pedagogical reasons, we are having you practice how to do this here with the read\_large\_file() function.**

**Complete Pre-Work**

The following [DataCamp Courses](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2F%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) are recommended and required for entry into the program:

* [Intro to Python for Data Science](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintro-to-python-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Intermediate Python for Data Science](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintermediate-python-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Statistical Thinking in Python (Part 1)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fstatistical-thinking-in-python-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Statistical Thinking in Python (Part 2)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fstatistical-thinking-in-python-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Importing Data in Python (Part 1)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fimporting-data-in-python-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Importing Data in Python (Part 2)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fimporting-data-in-python-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Python Data Science Toolbox (Part 1) DONE](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fpython-data-science-toolbox-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Python Data Science Toolbox (Part 2) Done](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fpython-data-science-toolbox-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Intro to SQL](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintro-to-sql-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)

**Need to understand how it works…**

<https://campus.datacamp.com/courses/python-data-science-toolbox-part-1/lambda-functions-and-error-handling?ex=13>

**27-6-19 2/5 35mins**

**# Extract the created\_at column from df: tweet\_time**

**tweet\_time = df['created\_at']**

**# Extract the clock time: tweet\_clock\_time**

**tweet\_clock\_time = [entry[11:19] for entry in tweet\_time]**

**# Print the extracted times**

**print(tweet\_clock\_time)**

**# Create generator object: result**

**result = (num for num in range(31))**

**# Print the first 5 values**

**print(next(result))**

**print(next(result))**

**print(next(result))**

**print(next(result))**

**print(next(result))**

**# Print the rest of the values**

**for value in result:**

**print(value)**

**\_\_\_\_\_\_\_**

**# Create a list of strings**

**lannister = ['cersei', 'jaime', 'tywin', 'tyrion', 'joffrey']**

**# Define generator function get\_lengths**

**def get\_lengths(input\_list):**

**"""Generator function that yields the**

**length of the strings in input\_list."""**

**# Yield the length of a string**

**for person in input\_list:**

**yield len(person)**

**# Print the values generated by get\_lengths()**

**for value in get\_lengths(lannister):**

**print(value)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Create a list of strings: fellowship

fellowship = ['frodo', 'samwise', 'merry', 'aragorn', 'legolas', 'boromir', 'gimli']

# Create dict comprehension: new\_fellowship

new\_fellowship = {member:len(member) for member in fellowship}

# Print the new dictionary

print(new\_fellowship)

**27-6-19 1/5 35mins**

# Create a list of strings: fellowship

fellowship = ['frodo', 'samwise', 'merry', 'aragorn', 'legolas', 'boromir', 'gimli']

# Create list comprehension: new\_fellowship

new\_fellowship = [member if len(member) >=7 else '' for member in fellowship]

# Print the new list

print(new\_fellowship)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print([nu for nu in range(7)])

var\_y = [num \*\* 2 if num % 3 == 0 else 0 for num in range(7)]

print(var\_y)

pos\_neg = {num: -num for num in range(6)}

print(pos\_neg)

print(type(pos\_neg))

**26-6-19 4&5/5 45mins**

**# Create a 5 x 5 matrix using a list of lists: matrix**

**matrix = [[[y for y in range(3)] for x in range(5)]for z in range(2)]**

**print(matrix)**

**[**

**[[0, 1, 2],**

**[0, 1, 2],**

**[0, 1, 2],**

**[0, 1, 2],**

**[0, 1, 2]],**

**[[0, 1, 2],**

**[0, 1, 2],**

**[0, 1, 2],**

**[0, 1, 2],**

**[0, 1, 2]]**

**]**

# Create a 5 x 5 matrix using a list of lists: matrix

matrix = [[col for col in range(5)] for row in range(5)]

# Print the matrix

for row in matrix:

print(row)

# **list comprehension**

* range (start, stop[, step])

pairs\_1 = []

In [2]: for num1 in range(0, 2):

...: for num2 in range(6, 8):

...: pairs\_1.append(num1, num2)

**doctor = ['house', 'cuddy', 'chase', 'thirteen', 'wilson']**

**In [4]: [doc + '!!!' for doc in doctor]**

**Out[4]: ['house!!!', 'cuddy!!!', 'chase!!!', 'thirteen!!!', 'wilson!!!']**

**In [5]: num = [1,22,2,4,48,66]**

**In [6]: [number + 3 for number in num]**

**Out[6]: [4, 25, 5, 7, 51, 69]**

[**https://campus.datacamp.com/courses/python-data-science-toolbox-part-2/using-iterators-in-pythonland?ex=12**](https://campus.datacamp.com/courses/python-data-science-toolbox-part-2/using-iterators-in-pythonland?ex=12)

**Answer:**

**# Define count\_entries()**

**def count\_entries(csv\_file,c\_size,colname):**

**"""Return a dictionary with counts of**

**occurrences as value for each key."""**

**# Initialize an empty dictionary: counts\_dict**

**counts\_dict = {}**

**# Iterate over the file chunk by chunk**

**for chunk in pd.read\_csv(csv\_file, chunksize=c\_size):**

**# Iterate over the column in DataFrame**

**for entry in chunk[colname]:**

**if entry in counts\_dict.keys():**

**counts\_dict[entry] += 1**

**else:**

**counts\_dict[entry] = 1**

**# Return counts\_dict**

**return counts\_dict**

**# Call count\_entries(): result\_counts**

**result\_counts = count\_entries('tweets.csv',10,'lang')**

**# Print result\_counts**

**print(result\_counts)**

**\_\_\_\_\_\_\_\_\_\_**

# Initialize an empty dictionary: counts\_dict

counts\_dict = {}

# Iterate over the file chunk by chunk

for chunk in pd.read\_csv('tweets.csv',chunksize =10):

# Iterate over the column in DataFrame

for entry in chunk['lang']:

if entry in counts\_dict.keys():

counts\_dict[entry] += 1

else:

counts\_dict[entry] = 1

# Print the populated dictionary

print(counts\_dict)

# **Extracting information for large amounts of Twitter data**

**Great job chunking out that file in the previous exercise. You now know how to deal with situations where you need to process a very large file and that's a very useful skill to have!**

**It's good to know how to process a file in smaller, more manageable chunks, but it can become very tedious having to write and rewrite the same code for the same task each time. In this exercise, you will be making your code more *reusable* by putting your work in the last exercise in a *function definition*.**

**The pandas package has been imported as pd and the file 'tweets.csv' is in your current directory for your use.**

**26-6-19 3/5 20mins**

**Let's play around with zip() a little more. There is no *unzip* function for doing the reverse of what zip() does. We can, however, reverse what has been zipped together by using zip() with a little help from \*! \* unpacks an *iterable* such as a list or a tuple into *positional arguments* in a function call.**

**In this exercise, you will use \* in a call to zip() to unpack the tuples produced by zip().**

**Two tuples of strings, mutants and powers have been pre-loaded.**

**# Print the tuples in z1 by unpacking with \***

**print(\*z1)**

**# Re-create a zip object from mutants and powers: z1**

**z1 = zip(mutants,powers)**

**# 'Unzip' the tuples in z1 by unpacking with \* and zip(): result1, result2**

**result1, result2 = zip(\*z1)**

**# Check if unpacked tuples are equivalent to original tuples**

**print(result1 == mutants)**

**print(result2 == powers)**

output:

('charles xavier', 'telepathy') ('bobby drake', 'thermokinesis') ('kurt wagner', 'teleportation') ('max eisenhardt', 'magnetokinesis') ('kitty pryde', 'intangibility')

True

True

**--------------**

# Create a list of strings: mutants

mutants = ['charles xavier',

'bobby drake',

'kurt wagner',

'max eisenhardt',

'kitty pryde']

# Create a list of tuples: mutant\_list

mutant\_list = list(mutants)

# Print the list of tuples

print(mutant\_list)

# Unpack and print the tuple pairs

for index1,value1 in enumerate(mutants):

print(index1, value1)

# Change the start index

for index2,value2 in enumerate(mutants, start =1):

print(index2, value2)

**26-6-19 2/5 20mins**

* **Create a range object that would produce the values from 10 to 20 using range(). Assign the result to values.**
* **Use the list() function to create a list of values from the range object values. Assign the result to values\_list.**
* **Use the sum() function to get the sum of the values from 10 to 20 from the range object values. Assign the result to values\_sum.**

**# Create a range object: values**

**values = range(10,21)**

**# Print the range object**

**print(values)**

**# Create a list of integers: values\_list**

**values\_list = list(values)**

**# Print values\_list**

**print(values\_list)**

**# Get the sum of values: values\_sum**

**values\_sum = sum(range(10,21))**

**# Print values\_sum**

**print(values\_sum)**

**range(10, 21)**

**[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]**

**165**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Create a list of strings: flash

flash = ['jay garrick', 'barry allen', 'wally west', 'bart allen']

# Print each list item in flash using a for loop

for person in flash:

print(person)

# Create an iterator for flash: superspeed

superspeed = iter(flash)

# Print each item from the iterator

print(next(superspeed))

print(next(superspeed))

print(next(superspeed))

print(next(superspeed))

\_\_\_\_\_

jay garrick

barry allen

wally west

bart allen

jay garrick

barry allen

wally west

bart allen

**26-6-19 1/5 20mins**

**Complete Pre-Work**

The following [DataCamp Courses](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2F%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) are recommended and required for entry into the program:

* [Intro to Python for Data Science](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintro-to-python-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Intermediate Python for Data Science](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintermediate-python-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Statistical Thinking in Python (Part 1)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fstatistical-thinking-in-python-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Statistical Thinking in Python (Part 2)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fstatistical-thinking-in-python-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Importing Data in Python (Part 1)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fimporting-data-in-python-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Importing Data in Python (Part 2)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fimporting-data-in-python-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) DONE
* [Python Data Science Toolbox (Part 1) DONE](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fpython-data-science-toolbox-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Python Data Science Toolbox (Part 2)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fpython-data-science-toolbox-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Intro to SQL](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintro-to-sql-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)

**Need to understand how it works…**

<https://campus.datacamp.com/courses/python-data-science-toolbox-part-1/lambda-functions-and-error-handling?ex=13>

# **Bringing it all together (1)**

This is awesome! You have now learned how to write anonymous functions using lambda, how to pass lambda functions as arguments to other functions such as map(), filter(), and reduce(), as well as how to write errors and output custom error messages within your functions. You will now put together these learnings to good use by working with a Twitter dataset. Before practicing your new error handling skills,in this exercise, you will write a lambda function and use filter() to select retweets, that is, tweets that begin with the string 'RT'.

To help you accomplish this, the Twitter data has been imported into the DataFrame, tweets\_df. Go for it!

##### **Instructions**

**0 XP**

* In the filter() call, pass a lambda function and the sequence of tweets as strings, tweets\_df['text']. The lambda function should check if the first 2 characters in a tweet xare 'RT'. Assign the resulting filter object to result. To get the first 2 characters in a tweet x, use x[0:2]. To check equality, use a Boolean filter with ==.
* Convert result to a list and print out the list.

# Select retweets from the Twitter DataFrame: result

result = filter(lambda x: x[0:2] == 'RT', tweets\_df['text'])

# Create list from filter object result: res\_list

res\_list = list(result)

# Print all retweets in res\_list

for tweet in res\_list:

print(tweet)

**25-6-19 4/5 25mins**

**Try and Except**

# Select retweets from the Twitter DataFrame: result

result = filter(lambda x: x[0:2] == 'RT', tweets\_df['text'])

# Create list from filter object result: res\_list

res\_list = list(result)

# Print all retweets in res\_list

for tweet in res\_list:

print(tweet)

**25-6-19 3/5 25mins**

**Lamda and also practise closure in python.**

# Import reduce from functools

from functools import reduce

# Create a list of strings: stark

stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result

result = reduce(lambda item1,item2:item1 + item2, stark)

# Print the result

print(result)

\_\_\_\_\_\_\_\_\_\_\_\_\_

# Create a list of strings: spells

spells = ["protego", "accio", "expecto patronum", "legilimens"]

# Use map() to apply a lambda function over spells: shout\_spells

shout\_spells = map(lambda item: item + '!!!', spells)

# Convert shout\_spells to a list: shout\_spells\_list

shout\_spells\_list = list(shout\_spells)

# Print the result

print(shout\_spells\_list)

**25-6-19 2/5 25mins**

<https://campus.datacamp.com/courses/python-data-science-toolbox-part-1/lambda-functions-and-error-handling?ex=3>

**25-6-19 1/5 25mins**

# Import numpy

import numpy as np

# Compute number of data points: n\_data

n\_data = len(versicolor\_petal\_length)

# Number of bins is the square root of number of data points: n\_bins

n\_bins = np.sqrt(n\_data)

# Convert number of bins to integer: n\_bins

n\_bins = int(n\_bins)

# Plot the histogram

\_ = plt.hist(versicolor\_petal\_length, bins = n\_bins)

# Label axes

\_ = plt.xlabel('petal length (cm)')

\_ = plt.ylabel('count')

**The "square root rule" is a commonly-used rule of thumb for choosing number of bins: choose the number of bins to be the square root of the number of samples. Plot the histogram of *Iris versicolor* petal lengths again, this time using the square root rule for the number of bins. You specify the number of bins using the binskeyword argument of plt.hist().**

# Import plotting modules

import matplotlib.pyplot as plt

import seaborn as sns

# Set default Seaborn style

sns.set()

# name of the bin edges as edges

edges = (3,3.5,4,4.5,5,5.5)

# Plot histogram of versicolor petal lengths

\_ = plt.hist(versicolor\_petal\_length, bins = edges)

# Show histogram

plt.show()

**24-6-19 3/5 35mins**

**Discrete and Continuous Random Variables:**

**A variable is a quantity whose value changes.**

**A discrete variable is a variable whose value is obtained by counting.**

***Examples*: number of students present**

**number of red marbles in a jar**

**number of heads when flipping three coins**

**students’ grade level**

**A continuous variable is a variable whose value is obtained by measuring.**

***Examples*: height of students in class**

**weight of students in class**

**time it takes to get to school**

**distance traveled between classes**

**A random variable is a variable whose value is a numerical outcome of a random phenomenon.**

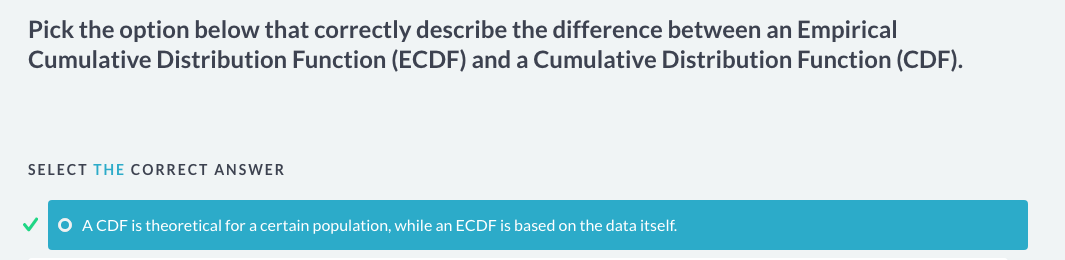
**▪ A random variable is denoted with a capital letter**

**▪ The probability distribution of a random variable *X* tells what the possible values of *X* are and how probabilities are assigned to those values**

**▪ A random variable can be discrete or continuous**

**A discrete random variable *X* has a countable number of possible values.**

***Example*: Let *X* represent the sum of two dice.**



**24-6-19 2/5 25mins**

**print("\nEND REPORT") # \ n for new line**

# Define echo\_shout()

def echo\_shout(word):

"""Change the value of a nonlocal variable"""

# Concatenate word with itself: echo\_word

echo\_word = word + word

# Print echo\_word

print(echo\_word)

# Define inner function shout()

def shout():

"""Alter a variable in the enclosing scope"""

# Use echo\_word in nonlocal scope

nonlocal echo\_word

# Change echo\_word to echo\_word concatenated with '!!!'

echo\_word = echo\_word + '!!!'

# Call function shout()

shout()

# Print echo\_word

print(echo\_word)

# Call function echo\_shout() with argument 'hello'

echo\_shout('hello')

**24-6-19 1/5 20mins**

**exploratory data analysis**

**EDA**

**the histograms by using the pandas library and indexing the DataFrame to extract the desired column. Here, however, you only need to use the provided NumPy array. Also, Justin assigned his plotting statements (except for plt.show()) to the dummy variable \_. This is to prevent unnecessary output from being displayed. It is not required for your solutions to these exercises, however it is good practice to use it. Alternatively, if you are working in an interactive environment such as a Jupyter notebook, you could use a ; after your plotting statements to achieve the same effect. Justin prefers using \_.**

<http://downloads.niceware.com/TECH-pdf/PythonStyle-Writing_idiomatic_python_3.pdf>

**22-6-19 5/5 8:50pm 60mins**

[**https://campus.datacamp.com/courses/python-data-science-toolbox-part-1/default-arguments-variable-length-arguments-and-scope?ex=7**](https://campus.datacamp.com/courses/python-data-science-toolbox-part-1/default-arguments-variable-length-arguments-and-scope?ex=7)

# **Nested Functions II**

Great job, you've just nested a function within another function. One other pretty cool reason for nesting functions is the idea of a **closure**. This means that the nested or inner function remembers the state of its enclosing scope when called. Thus, anything defined locally in the enclosing scope is available to the inner function even when the outer function has finished execution.

Let's move forward then! In this exercise, you will complete the definition of the inner function inner\_echo() and then call echo() a couple of times, each with a different argument. Complete the exercise and see what the output will be!

# Define echo

def echo(n):

"""Return the inner\_echo function."""

# Define inner\_echo

def \_\_\_\_(\_\_\_\_):

"""Concatenate n copies of word1."""

echo\_word = word1 \* n

return echo\_word

# Return inner\_echo

# Call echo: twice

twice = echo(2)

# Call echo: thrice

# Call twice() and thrice() then print

print(twice('hello'), thrice('hello'))

* **Complete the function header of the inner function with the function name inner\_echo() and a single parameter word1.**
* **Complete the function echo() so that it returns inner\_echo.**
* **We have called echo(), passing 2 as an argument, and assigned the resulting function to twice. Your job is to call echo(), passing 3 as an argument. Assign the resulting function to thrice.**
* **Hit Submit to call twice() and thrice() and print the results.**

###### **Hint**

* **A nested function is defined the same way you would define a regular function: def *function name*(*parameters*):.**
* **Make sure to return the function inner\_echo without the parentheses () from *inside* the echo() function.**
* **The second call to echo() follows the same format as the previous echo() call, but with 3 as an argument. Don't forget to assign the result to the object thrice.**

**Answer:**

**# Define echo**

**def echo(n):**

**"""Return the inner\_echo function."""**

**# Define inner\_echo**

**def inner\_echo(word1):**

**"""Concatenate n copies of word1."""**

**echo\_word = word1 \* n**

**return echo\_word**

**# Return inner\_echo**

**return inner\_echo**

**# Call echo: twice**

**twice = echo(2)**

**# Call echo: thrice**

**thrice = echo(3)**

**# Call twice() and thrice() then print**

**print(twice('hello'), thrice('hello'))**

**---------------------**

**# Define three\_shouts**

**def three\_shouts(word1, word2, word3):**

**"""Returns a tuple of strings**

**concatenated with '!!!'."""**

**# Define inner**

**def \_\_\_\_(\_\_\_\_):**

**"""Returns a string concatenated with '!!!'."""**

**return word + '!!!'**

**# Return a tuple of strings**

**return (\_\_\_\_, \_\_\_\_, \_\_\_\_)**

**# Call three\_shouts() and print**

**print(three\_shouts('a', 'b', 'c'))**

dir(builtins) # after importing the library

After executing import builtins in the IPython Shell, execute dir(builtins)

# Create a string: team

team = "teen titans"

# Define change\_team()

def change\_team():

"""Change the value of the global variable team."""

# Use team in global scope

global team

# Change the value of team in global: team

team = 'justice league'

# Print team

print(team)

# Call change\_team()

change\_team()

# Print team

print(team)

#output is ‘justice league'

**22-6-19 4/5 35 mins**

# Import pandas

import pandas as pd

# Import Twitter data as DataFrame: df

df = pd.read\_csv('tweets.csv')

# Initialize an empty dictionary: langs\_count

langs\_count = {}

# Extract column from DataFrame: col

col = df['lang']

# Iterate over lang column in DataFrame

for entry in col:

# If the language is in langs\_count, add 1

if entry in langs\_count.keys():

langs\_count[entry] += 1

# Else add the language to langs\_count, set the value to 1

else:

langs\_count[entry] = 1

# Print the populated dictionary

print(langs\_count)

**22-6-19 3/5 40 mins**

<https://campus.datacamp.com/courses/python-data-science-toolbox-part-1/writing-your-own-functions?ex=1>

1.20 mins to go.

def sum(var):

yr = var \* 2

return yr

num = sum(4)

print(num)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_--

# Import packages

import matplotlib.pyplot as plt

import seaborn as sns

# Set seaborn style

sns.set(color\_codes=True)

# Create a list of labels:cd

cd = ['clinton', 'trump', 'sanders', 'cruz']

# Plot histogram

ax = sns.barplot(cd, [clinton, trump, sanders, cruz])

ax.set(ylabel="count")

plt.show()

**22-6-19 2/5 65mins**

# Import package

import pandas as pd

# Build DataFrame of tweet texts and languages

df = pd.DataFrame(tweets\_data, columns=['text', 'lang'])

# Print head of DataFrame

print(df.head())

* Use pd.DataFrame() to construct a DataFrame of tweet texts and languages; to do so, the first argument should be tweets\_data, a list of dictionaries. The second argument to pd.DataFrame() is a *list* of the keys you wish to have as columns. Assign the result of the pd.DataFrame() call to df.

###### **Hint**

* Try pd.DataFrame(tweets\_data).
* You should construct a *list* of keys and assign it to the argument columns for the second argument to pd.DataFrame(). The list should contain the strings text and lang.
* Use the method head() on the DataFrame df and pass to print().

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Initialize Stream listener

l = MyStreamListener()

# Create your Stream object with authentication

stream = tweepy.Stream(auth, l)

# Filter Twitter Streams to capture data by the keywords \_To filter Twitter streams, pass to the track argument in stream.filter() a list containing the desired keywords 'clinton', 'trump', 'sanders', and 'cruz'.

:

track = ['clinton','trump','sanders','cruz']

stream.filter(track)

###### **Hint**

* Check to make sure that you passed auth as the first argument to tweepy.Stream() and l as the second argument.
* Assign the list with the keywords to the argument track in stream.filter().

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Initialize Stream listener

l = MyStreamListener()

# Create your Stream object with authentication

stream = tweepy.Stream(auth, l)

# Filter Twitter Streams to capture data by the keywords:

track = ['clinton','trump','sanders','cruz']

stream.filter(track)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Import package

import tweepy

# Store OAuth authentication credentials in relevant variables

access\_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fFO8k8EPy"

access\_token\_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"

consumer\_key = "nZ6EA0FxZ293SxGNg8g8aP0HM"

consumer\_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"

# Pass OAuth details to tweepy's OAuth handler Complete the passing of OAuth credentials to the OAuth handler auth by applying to it the method set\_access\_token(), along with arguments access\_token and access\_token\_secret.

auth = tweepy.OAuthHandler(consumer\_key, consumer\_secret)

auth.set\_access\_token(access\_token, access\_token\_secret)

\_\_\_\_\_\_\_\_\_\_

* Apply the json() method to the response object r and store the resulting dictionary in the variable json\_data.

# Decode the JSON data into a dictionary: json\_data

json\_data = r.json()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Import package

import requests

# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=72bc447a&t=social+network'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Decode the JSON data into a dictionary: json\_data

json\_data = r.json()

# Print each key-value pair in json\_data

for k in json\_data.keys():

print(k + ': ', json\_data[k])

* Apply the json() method to the response object r and store the resulting dictionary in the variable json\_data.

# Decode the JSON data into a dictionary: json\_data

json\_data = json(r)

\_\_\_\_\_\_\_\_\_\_\_

# Import requests package

import requests

# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=72bc447a&t=the+social+network'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Print the text of the response

print(r.text)

**22-6-19 1/5 80 mins**

Function is text.read()

Argument is something eg. link.get(‘href’)

# Print the URLs to the shell

for link in a\_tags:

print(link.get('href'))

* Pass the *HTML tag* to find (without the angle brackets <>) as a string argument to find\_all().
* Recall that the for loop recipe is: for *loop variable* in *results set*:. Don't forget to pass link.get('href') as an argument to print() inside the for loop body.

Incorrect below:

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extracts the response as html: html\_doc

html\_doc = r.text

# create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Print the title of Guido's webpage

print(soup.title)

# Find all 'a' tags (which define hyperlinks): a\_tags

a\_tags = soup.find\_all('a')

# Print the URLs to the shell

for i in a\_tags:

link.get(print(i))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url: url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extract the response as html: html\_doc

html\_doc = r.text

# Create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Get the title of Guido's webpage: guido\_title

guido\_title = soup.title

# Print the title of Guido's webpage to the shell

print(guido\_title)

# Get Guido's text: guido\_text

guido\_text = soup.get\_text()

# Print Guido's text to the shell

print(guido\_text)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url: url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extracts the response as html: html\_doc

html\_doc = r.text

# Create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Prettify the BeautifulSoup object: pretty\_soup

pretty\_soup = soup.prettify()

# Print the response

print(pretty\_soup)

\_\_\_\_\_\_\_\_\_\_

# Import package

import requests

# Specify the url: url

url = "http://www.datacamp.com/teach/documentation"

# Packages the request, send the request and catch the response: r

r = requests.get(url)

# Extract the response: text

text = r.text

# Print the html

print(text)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "http://www.datacamp.com/teach/documentation"

# This packages the request

request = Request(url)

# Sends the request and catches the response: response

response = urlopen(request)

# Extract the response: html

html = response.read()

# Print the html

print(html)

# Be polite and close the response!

response.close()

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Import package

import pandas as pd

# Assign url of file: url

url = 'http://s3.amazonaws.com/assets.datacamp.com/course/importing\_data\_into\_r/latitude.xls'

# Read in all sheets of Excel file: xl

xl = pd.read\_excel(url, sep = ';', sheetname = None)

# Print the sheetnames to the shell - Print the names of the sheets in the Excel spreadsheet; these will be the keys of the dictionary xl.

print(xl.keys())

# Print the head of the first sheet (using its name, NOT its index) Print the head of the first sheet *using the sheet name, not the index of the sheet*! The sheet name is '1700'

print(xl['1700'].head())

\_\_\_\_\_\_\_\_\_\_\_\_\_

Uniform/Universal Resource Locator References to web resources

Focus: web addresses

Ingredients:

Protocol identifier - http:

Resource name - datacamp.com These specify web addresses uniquely

HyperText Transfer Protocol

Foundation of data communication for the web

HTTPS - more secure form of HTTP

Going to a website = sending HTTP request

GET request

urlretrieve() performs a GET request

HTML - HyperText Markup Language

In [1]: from urllib.request import urlretrieve

In [2]: url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv'

In [3]: urlretrieve(url, 'winequality-white.csv')

Out[3]: ('winequality-white.csv', <http.client.HTTPMessage at 0x103cf1128>)

**21-6-19 2/3 50 mins**

<https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/thinking-probabilistically-continuous-variables?ex=9>

# Compute mean and standard deviation: mu, sigma

mu = np.mean(belmont\_no\_outliers)

sigma = np.std(belmont\_no\_outliers)

# Sample out of a normal distribution with this mu and sigma: samples

samples = np.random.normal(mu, sigma, 10000)

# Get the CDF of the samples and of the data

x\_theor, y\_theor = ecdf(samples)

x, y = ecdf(belmont\_no\_outliers)

# Plot the CDFs and show the plot

\_ = plt.plot(x\_theor, y\_theor)

\_ = plt.plot(x, y, marker='.', linestyle='none')

\_ = plt.xlabel('Belmont winning time (sec.)')

\_ = plt.ylabel('CDF')

plt.show()

**21-6-19 1/3 20 mins**

<https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/thinking-probabilistically-continuous-variables?ex=5>

Need to watch the video before the above exercise

# Draw 100000 samples from Normal distribution with stds of interest: samples\_std1, samples\_std3, samples\_std10

samples\_std1 = np.random.normal(20, 1, 100000)

samples\_std3 = np.random.normal(20, 3, 100000)

samples\_std10 = np.random.normal(20, 10, 100000)

# Make histograms

\_ = plt.hist(samples\_std1, histtype='step', normed=True, bins = 100)

\_ = plt.hist(samples\_std3, histtype='step', normed=True, bins = 100)

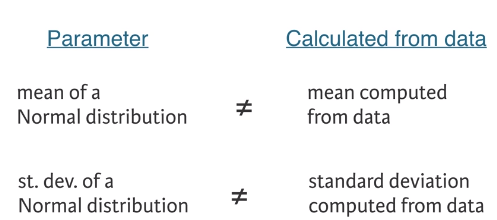
\_ = plt.hist(samples\_std10, histtype='step', normed=True, bins = 100)

# Make a legend, set limits and show plot

\_ = plt.legend(('std = 1', 'std = 3', 'std = 10'))

plt.ylim(-0.01, 0.42)

plt.show()



<https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/thinking-probabilistically-discrete-variables?ex=15>

# Draw 10,000 samples out of Poisson distribution: n\_nohitters

n\_nohitters = np.random.poisson(251/115, 10000)

# Compute number of samples that are seven or greater: n\_large

n\_large = np.sum(n\_nohitters >=7)

# Compute probability of getting seven or more: p\_large

p\_large = n\_large/10000

# Print the result

print('Probability of seven or more no-hitters:', p\_large)

20-6-19 3/3 10 mins

<https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/thinking-probabilistically-discrete-variables?ex=13>

# Draw 10,000 samples out of Poisson distribution: samples\_poisson

samples\_poisson = np.random.poisson(10, size=10000)

# Print the mean and standard deviation

print('Poisson: ', np.mean(samples\_poisson),

np.std(samples\_poisson))

# Specify values of n and p to consider for Binomial: n, p

n = [20, 100, 1000]

p = [0.5, 0.1, 0.01]

# Draw 10,000 samples for each n,p pair: samples\_binomial

for i in range(3):

samples\_binomial = np.random.binomial(n[i], p[i], size=10000)

# Print results

print('n =', n[i], 'Binom:', np.mean(samples\_binomial),

np.std(samples\_binomial))

20-6-19 2/3 30 mins

<https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/thinking-probabilistically-discrete-variables?ex=11>

# Compute bin edges: bins

bins = np.arange(0, max(n\_defaults) + 1.5) - 0.5

# Generate histogram

\_ = plt.hist(n\_defaults, normed=True, bins=bins)

# Label axes

\_ = plt.xlabel('number of defaults out of 100 loans')

\_ = plt.ylabel('PMF')

# Show the plot

plt.show()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_--

# Seed random number generator

np.random.seed(42)

# Initialize the number of defaults: n\_defaults

n\_defaults = np.empty(1000)

# Compute the number of defaults

for i in range(1000):

n\_defaults[i] = perform\_bernoulli\_trials(100, p = 0.05)

# Plot the histogram with default number of bins; label your axes

\_ = plt.hist(n\_defaults, normed=True)

\_ = plt.xlabel('number of defaults out of 100 loans')

\_ = plt.ylabel('probability')

# Show the plot

plt.show()

20-6-19 1/3 17 mins

Classes

ID Enrolled Passed

A 1 20 17

B 2 18 16

C 3 22 19

D 4 30 20

E 5 25 18

**What is the output of this code?**

print(classes[0:3])

**SELECT THE OUTPUT**

ID Enrolled Passed

A 1 20 17

B 2 18 16

C 3 22 19

19-6-19 3/3 15 mins

<https://campus.datacamp.com/courses/statistical-thinking-in-python-part-1/thinking-probabilistically-discrete-variables?ex=7>

def perform\_bernoulli\_trials(n, p):

"""Perform n Bernoulli trials with success probability p

and return number of successes."""

# Initialize number of successes: n\_success

n\_success = 0

# Perform trials

for i in range(n):

# Choose random number between zero and one: random\_number

random\_number = np.random.random()

# If less than p, it's a success so add one to n\_success

if random\_number < p:

n\_success += 1

return n\_success

19-6-19 2/3 25mins

* Look at the spread in the x-direction in the plots: The plot with the largest spread is the one that has the highest variance.
* High covariance means that when x is high, y is also high, and when x is low, y is also low.
* Negative covariance means that when x is high, y is low, and when x is low, y is high.

19-6-19 ⅓ 15mins

# Definition of dictionary

europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin',

'norway':'oslo', 'italy':'rome', 'poland':'warsaw', 'austria':26 }

# Iterate over europe

for k,v in europe.items():

print(f'{v} is the capital of {k}')

\_\_\_\_\_\_\_\_\_\_\_\_

# numpy and matplotlib imported, seed set

np.random.seed(123)

# Simulate random walk 500 times

all\_walks = []

for i in range(500) :

random\_walk = [0]

for x in range(100) :

step = random\_walk[-1]

dice = np.random.randint(1,7)

if dice <= 2:

step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

if np.random.rand() <= 0.001 :

step = 0

random\_walk.append(step)

all\_walks.append(random\_walk)

# Create and plot np\_aw\_t

np\_aw\_t = np.transpose(np.array(all\_walks))

# Select last row from np\_aw\_t: ends

ends = np\_aw\_t[-1,:]

# Plot histogram of ends, display plot

plt.hist(ends, bins= 4)

plt.show()

### Calculate the odds

The histogram of the previous exercise was created from a Numpy array ends, that contains 500 integers. Each integer represents the end point of a random walk. To calculate the chance that this end point is greater than or equal to 60, you can count the number of integers in ends that are greater than or equal to 60 and divide that number by 500, the total number of simulations.

Well then, what's the estimated chance that you'll reach 60 steps high if you play this Empire State Building game? The endsarray is everything you need; it's available in your Python session so you can make calculations in the IPython Shell.

In [4]: np.mean(ends >= 60)

Out[4]: 0.784

-------------------------------------

C:\Users\rato.li>psql

'psql' is not recognized as an internal or external command,

operable program or batch file.

C:\Users\rato.li>cd /d "C:\Program Files\PostgreSQL"

C:\Program Files\PostgreSQL>dir

Volume in drive C is Windows

Volume Serial Number is E68F-2DE1

Directory of C:\Program Files\PostgreSQL

14-05-2019 04:33 PM <DIR> .

14-05-2019 04:33 PM <DIR> ..

07-06-2019 04:56 PM <DIR> 11

14-05-2019 04:33 PM <DIR> psqlODBC

0 File(s) 0 bytes

4 Dir(s) 21,349,990,400 bytes free

C:\Program Files\PostgreSQL>cd 11

C:\Program Files\PostgreSQL\11>dir

Volume in drive C is Windows

Volume Serial Number is E68F-2DE1

Directory of C:\Program Files\PostgreSQL\11

07-06-2019 04:56 PM <DIR> .

07-06-2019 04:56 PM <DIR> ..

14-05-2019 04:11 PM <DIR> bin

07-05-2019 06:36 PM 35,986 commandlinetools\_3rd\_party\_licenses.txt

07-06-2019 04:56 PM 8,518,953 uninstall-postgresql.exe

9 File(s) 8,793,864 bytes

12 Dir(s) 21,349,949,440 bytes free

C:\Program Files\PostgreSQL\11>dir bin

Volume in drive C is Windows

Volume Serial Number is E68F-2DE1

Directory of C:\Program Files\PostgreSQL\11\bin

14-05-2019 04:11 PM <DIR> .

14-05-2019 04:11 PM <DIR> ..

07-05-2019 06:36 PM 97,280 clusterdb.exe

07-05-2019 06:36 PM 96,768 createdb.exe

07-05-2019 06:36 PM 98,816 createuser.exe

07-05-2019 06:36 PM 93,696 dropdb.exe

07-05-2019 06:36 PM 620,032 wxmsw28u\_xrc\_vc\_custom.dll

07-05-2019 06:36 PM 90,624 zic.exe

07-05-2019 06:36 PM 86,528 zlib1.dll

69 File(s) 53,605,904 bytes

2 Dir(s) 21,349,949,440 bytes free

C:\Program Files\PostgreSQL\11>bin\psql.exe -U postgres

Password for user postgres:

psql: FATAL: password authentication failed for user "postgres"

C:\Program Files\PostgreSQL\11>dir

Volume in drive C is Windows

Volume Serial Number is E68F-2DE1

Directory of C:\Program Files\PostgreSQL\11

07-06-2019 04:56 PM <DIR> .

07-06-2019 04:56 PM <DIR> ..

14-05-2019 04:11 PM <DIR> bin

07-05-2019 06:36 PM 35,986 commandlinetools\_3rd\_party\_licenses.txt

07-06-2019 04:56 PM <DIR> data

14-05-2019 04:09 PM <DIR> debug\_symbols

14-05-2019 04:09 PM <DIR> doc

14-05-2019 04:09 PM <DIR> include

07-06-2019 04:56 PM 1,328 installation\_summary.log

14-05-2019 04:09 PM <DIR> installer

14-05-2019 04:11 PM <DIR> lib

07-06-2019 04:55 PM <DIR> pgAdmin 4

07-05-2019 06:37 PM 64,069 pgAdmin\_3rd\_party\_licenses.txt

07-05-2019 06:37 PM 1,026 pgAdmin\_license.txt

07-06-2019 04:56 PM 341 pg\_env.bat

14-05-2019 04:09 PM <DIR> scripts

07-05-2019 06:37 PM 1,388 server\_license.txt

14-05-2019 04:11 PM <DIR> share

07-05-2019 06:37 PM 4,178 StackBuilder\_3rd\_party\_licenses.txt

07-06-2019 04:56 PM 166,595 uninstall-postgresql.dat

07-06-2019 04:56 PM 8,518,953 uninstall-postgresql.exe

9 File(s) 8,793,864 bytes

12 Dir(s) 21,349,810,176 bytes free

C:\Program Files\PostgreSQL\11>bin\psql.exe -U postgres

Password for user postgres:

psql (11.3)

WARNING: Console code page (437) differs from Windows code page (1252)

8-bit characters might not work correctly. See psql reference

page "Notes for Windows users" for details.

Type "help" for help.

postgres=# \c

WARNING: Console code page (437) differs from Windows code page (1252)

8-bit characters might not work correctly. See psql reference

page "Notes for Windows users" for details.

You are now connected to database "postgres" as user "postgres".

postgres=# \dt

Did not find any relations.

postgres=# \q

C:\Program Files\PostgreSQL\11>bin\psql.exe -U postgres < C:\DB\_Backup\otp-qa.sql

Password for user postgres:

SET

SET

set\_config

------------

(1 row)

SET

SET

ALTER TABLE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

select

body->>'name' as "name",

body->>'type' as "type",

body->>'status' as "status",

body->'finance'->>'currency' as "currency",

--body->'finance'->>'legalEntity' as "legalEntity",

body->'finance'->'legalEntity'->'Edge' as "Edge",

body->'finance'->'legalEntity'->'Australia' as "Australia",

body->'finance'->'legalEntity'->'New Zealand' as "New Zealand",

body->'finance'->'termsOfPayment' as "termsOfPayment",

body->'finance'->'taxWithholdCalculate' as "taxWithholdCalculate",

body->>'gstRate' as "GST",

body->>'commission' as "commission",

body->>'creditStatus' as "creditStatus",

body->>'ownerPayment' as "ownerPayment",

body->>'abn' as "abn",

body->>'gstApplicable' as "gstApplicable",

body->'contactableFlags'->'homePhone' as "homePhone",

body->'contactableFlags'->'workEmail' as "workEmail",

body->'contactableFlags'->'mobilePhone' as "mobilePhone",

body->'contactableFlags'->'businessPhone' as "businessPhone",

body->'contactableFlags'->'personalEmail' as "personalEmail",

body->>'abn' as "abn",

body->>'approvedForAxIntegration' as "approvedForAxIntegration",

body->>'mediaInvoicesBillToSelection' as "mediaInvoicesBillToSelection",

body->>'productionInvoicesBillToSelection' as "productionInvoicesBillToSelection",

body->>'installationInvoicesBillToSelection' as "installationInvoicesBillToSelection",

body->>'phoneNumber' as "phoneNumber"

from crm.accounts;

18-6-19 # 5/5 25 mins

Need to revise

# np.random.seed(123)

# for x in range(100):

# times = [0]

# coin = np.random.randint(0,2)

# times.append(times(x) + coin)

# print(times)

# Numpy is imported; seed is set

np.random.seed(123)

# Initialize all\_walks (don't change this line)

all\_walks = []

# Simulate random walk 10 times

for i in range(10):

# Code from before

random\_walk = [0]

for x in range(100) :

step = random\_walk[-1]

dice = np.random.randint(1,7)

if dice <= 2:

step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

random\_walk.append(step)

# Append random\_walk to all\_walks

all\_walks.append(random\_walk)

# Print all\_walks

print(all\_walks)

18-6-19 # 4/5 28 mins

variable x never goes below 10 when you decrease it, you can use:

x = max(10, x - 1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Numpy is imported, seed is set

np.random.seed(123)

# Initialize random\_walk

random\_walk = [0]

# Complete the \_\_\_

for x in range(100) :

# Set step: last element in random\_walk

step = random\_walk[-1]

# Roll the dice

dice = np.random.randint(1,7)

# Determine next step

if dice <= 2:

step = step - 1

# #Replace below: use max to make sure step can't go below 0

# step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

# append next\_step to random\_walk

random\_walk.append(step)

# Print random\_walk

print(random\_walk)

\_\_\_\_\_\_\_\_\_\_\_

Last Part of this exercise:

# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

# Plot random\_walk

plt.plot(random\_walk)

# Show the plot

plt.show()



18-6-19 # 3/5 25 mins

# Import numpy and set seed

import numpy as np

np.random.seed(123) # seed

# Use randint() to simulate a dice

print(np.random.randint(1,7)) # dice 1 to 6 but need to put 7 as it stop one before.

# Use randint() again

print(np.random.randint(1,7))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Import numpy as np

import numpy as np

# Set the seed

np.random.seed(123)

#seed(): sets the random seed, so that your results are the reproducible between simulations. As an argument, it takes an integer of your choosing. If you call the function, no output will be generated.

# Generate and print random float

print(np.random.rand())

print(np.random.rand())

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

18-6-19 # 2/5 40mins

# Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Use .apply(str.upper)

#for lab, row in cars.iterrows() :

# cars.loc[lab, "COUNTRY"] = row["country"].upper()

#for lab, row in brics.iterrows() :

# brics.loc[lab, "name\_length"] = len(row["country"])

#brics["name\_length"] = brics["country"].apply(len)

cars["COUNTRY"] = cars["country"].apply(str.upper) #.apply(str.upper) same as upper()

print(cars)

# Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Code for loop that adds COUNTRY column

for lab, row in cars.iterrows():

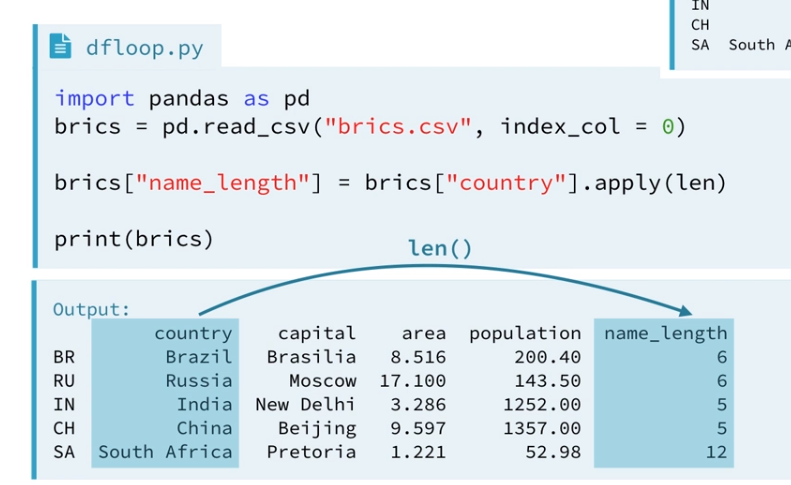
cars.loc[lab, "COUNTRY"] = row["country"].upper()

#for lab, row in brics.iterrows() :

# brics.loc[lab, "name\_length"] = len(row["country"])

# Print cars

print(cars)



# Definition of dictionary

europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin',

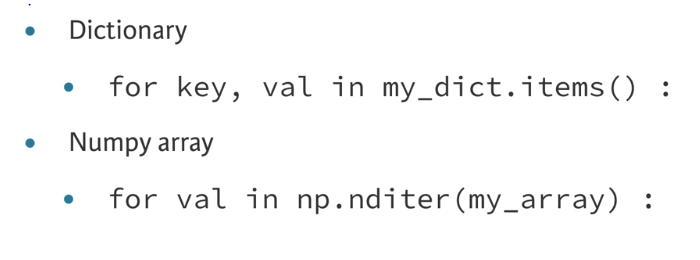
'norway':'oslo', 'italy':'rome', 'poland':'warsaw', 'austria':26 }

# Iterate over europe

for key, value in europe.items():

print(f'the capital of {key} is {str(value)}')

18-6-19 # 1/5 15mins



for key, val in my\_dictionary.items() :

for val in np.nditer(my\_array) :

17-6-19 # 3/4 10 mins simple stuff, came home and ate and watch youtube,

It was late so I need to go to bed,

Do NOT do it again tomorrow try 5 times 25, becare about youtube. And videos.

Manage time properly.

17-6-19 # 2/4 35min

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Convert code to a one-liner

sel = cars[cars['drives\_right']]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Import numpy, you'll need this

import numpy as np

# Create medium: observations with cars\_per\_cap between 100 and 500

cpc = cars['cars\_per\_cap']

between = np.logical\_and(cpc > 100,cpc < 500)

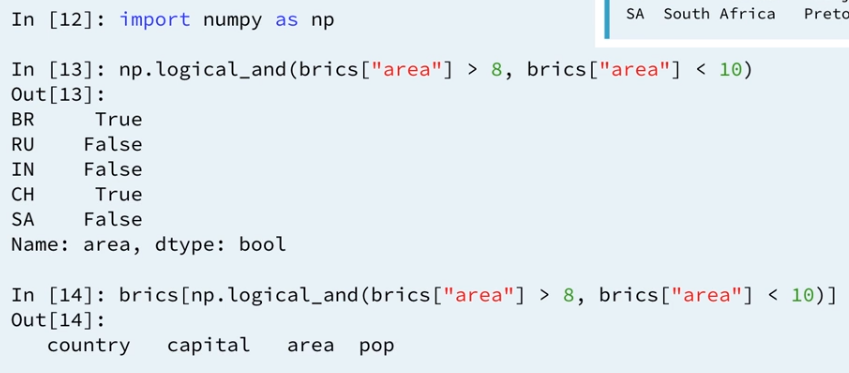
medium = cars[between]

# Print medium

print(medium)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

17-6-19 # 1/4 15min



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

product profit

0 apples 20

1 apples 10

df.index = ['may', 'jun']

print(df)

product profit

may apples 20

jun apples 10

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fruits = {

'apples':{'cost':3, 'units':100},

'bananas':{'cost':1, 'units':80},

'grapes':{'cost':5, 'units':500}}

print(fruits['bananas']['cost'])

1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Import pandas as pd

df = pd.read\_csv(dataset,index\_col=0)

print(df)

Product Units Cost

X A 10 20

Y A 12 22

Z A 10 25

------------------------

import numpy as np

y = np.array([[4, 5, 6],

[11, 12, 13]])

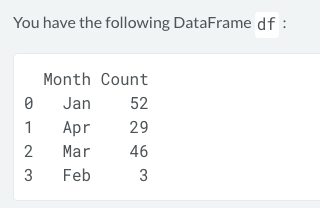
np.transpose(y)

array([[ 4, 11],

[ 5, 12],

[ 6, 13]])

16-6-19 # 8/10 10min

****

print(df['Count'].quantile([0.10, 0.90]))

0.1 10.8

0.9 50.2

Name: Count, dtype: float64

print(df['Count'].quantile([0.10, 0.90]))

y = [15, 15, 14, 9]

y.append(65)

print(y)

# Find the mean of the second column of costs

import numpy as np

costs = np.column\_stack(([2, 1, 2, 1],

[4, 6, 5, 5]))

mean\_costs = np.mean(costs[:,1])

print(mean\_costs)

|  |
| --- |
| # Python Program illustrating  # numpy.median() method    import numpy as np    # 1D array  arr = [20, 2, 7, 1, 34]    print("arr : ", arr)  print("median of arr : ", np.median(arr)) |

**Output :**

arr : [20, 2, 7, 1, 34]

median of arr : 7.0

16-6-19 # 7/10 35min

import pandas as pd

from sqlalchemy import create\_engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("select \* from PlaylistTrack INNER JOIN Track on PlaylistTrack.TrackID = Track.TrackID where milliseconds < 250000", engine)

# Print head of DataFrame

print(df.head())

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("select \* from Album", engine) # do it in one line.

# Print head of DataFrame

print(df.head())

\_\_\_\_\_\_\_\_\_\_\_\_\_

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine in context manager

with engine.connect() as con:

rs = con.execute("SELECT \* FROM Employee ORDER BY BirthDate")

df = pd.DataFrame(rs.fetchall())

# Set the DataFrame's column names

df.columns = rs.keys()

# Print head of DataFrame

print(df.head())

16-6-19 # 7/10 27min

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

rs = con.execute("SELECT \* FROM Employee WHERE EmployeeId >= 6") #” instead of ‘ one comma

df = pd.DataFrame(rs.fetchall()) # versus (rs.fetchmany(size = 3))

df.columns = rs.keys()

# Print the head of the DataFrame df

print(df.head())

------------

16-6-19 # 6/10 10min

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<https://www.sqlite.org/eqp.html>

<https://stackoverflow.com/questions/7359702/how-do-i-obtain-a-query-execution-plan>

<https://www.khanacademy.org/computing/computer-programming/sql/relational-queries-in-sql/a/more-efficient-sql-with-query-planning-and-optimization> For example, remember that query we discussed at the top? If we knew ahead of time that we would want to do hundreds of queries that restricted WHEREon the author column, then we could explicitly create the index, using CREATE INDEX. Then the SQL engine would be able to use that index to efficiently find the matching rows. You can read this guide about [SQLite query planning](https://www.sqlite.org/queryplanner.html)to help you understand when indexes would help.

Creating indexes can often make repeated queries more efficient. But there are many other approaches as well. For SQLite, you can get more insight in their [query planner overview](https://www.sqlite.org/optoverview.html) and take careful note of the "manual" sections.

We can't cover all the complexities of query optimization and query tuning here, so I recommend that you dive deeper into it when you need it.

different SQL query planners that I found interesting: [SQL Server Query Optimizer](http://sqlblog.com/blogs/paul_white/archive/2012/04/28/query-optimizer-deep-dive-part-1.aspx), [Oracle SQL Tuning](http://docs.oracle.com/cd/B19306_01/server.102/b14211/sql_1016.htm#g42927), [MSSQL Execution Plan Basics](https://www.simple-talk.com/sql/performance/execution-plan-basics/)

log base 2 of 10 million is 23

15-6-19 # 5/10 15min

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine connection: con

con = engine.connect()

# Perform query: rs

rs = con.execute('SELECT \* FROM Album')

# Save results of the query to DataFrame: df

df = pd.DataFrame(rs)

# Close connection

con.close()

# Print head of DataFrame df

print(df.head())

15-6-19 # 4/10 25min

# Import necessary module

from sqlalchemy import create\_engine

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Save the table names to a list: table\_names

table\_names = engine.table\_names() # learnt that the method table\_names for engine.

# Print the table names to the shell

print(table\_names)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_--

\_\_\_\_\_\_\_\_\_\_# Import necessary module

from sqlalchemy import create\_engine

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

15-6-19 # 3/10 25min

for key in h5py\_data.keys(): print(key)

Stata files:

In [1]: df = pd.read\_stata('disarea.dta')

In [2]: df.head()

\_\_\_

Some but not all UCD computer labs have **Stata**.**Stata** stores data in a special format that cannot be read by other programs. **Stata** can read data in several other formats. A standard format is a comma-separated values **file** with extension .csv (which can be created by Excel for example).

\_\_\_

<https://campus.datacamp.com/courses/importing-data-in-python-part-1/importing-data-from-other-file-types-2?ex=9>

# Import sas7bdat package

from sas7bdat import SAS7BDAT

# Save file to a DataFrame: df\_sas

with SAS7BDAT('sales.sas7bdat') as file:

df\_sas = file.to\_data\_frame()

#In the context of the file 'sales.sas7bdat', load its contents to a DataFrame df\_sas, using the method to\_data\_frame() on the object file.

# Print head of DataFrame

print(df\_sas.head())

# Plot histogram of DataFrame features (pandas and pyplot already imported)

pd.DataFrame.hist(df\_sas[['P']])

plt.ylabel('count')

plt.show()

# **Loading a pickled file**

There are a number of datatypes that cannot be saved easily to flat files, such as lists and dictionaries.

# Import pickle package

import pickle

# Open pickle file and load data: d

with open('data.pkl', 'rb') as file:

d = pickle.load(file)

# Print d

print(d)

# Print datatype of d

print(type(d))

If you want your files to be human readable, you may want to save them as text files in a clever manner. JSONs, which you will see in a later chapter, are appropriate for Python dictionaries.

However, if you merely want to be able to import them into Python, you can [**serialize**](https://en.wikipedia.org/wiki/Serialization) them. All this means is converting the object into a sequence of bytes, or a bytestream.

‘rb’ readable bytestream

15-6-19 # 2/10 35min

# Parse the first sheet and rename the columns: df1 - The values passed to skiprows and names all need to be of type list. Parse the first sheet by index. In doing so, skip the first row of data and name the columns 'Country' and 'AAM due to War (2002)' using the argument names. The values passed to skiprows and names all need to be of type list.

df1 = xls.parse(0, skiprows=[0], names=['Country', 'AAM due to War (2002)'])

# Print the head of the DataFrame df1

print(df1.head())

# Parse the first column of the second sheet and rename the column: df2 Parse the second sheet by index. In doing so, parse only the first column with the usecols parameter, skip the first row and rename the column 'Country'. The argument passed to usecols also needs to be of type list.

df2 = xls.parse(1, usecols=0, skiprows=[0], names=['Country'])

# Print the head of the DataFrame df2

print(df2.head())

\_\_\_\_\_\_\_\_\_-

Sat 15-6-19 # 1 Goal for at least 10 of these on Saturday or Free Day approx. 25min each.

# Assign the filename: file

file = 'digits.csv'

# Read the first 5 rows of the file into a DataFrame: data

data = pd.read\_csv('digits.csv' ,nrows = 5, header = None)

print(data)

# Build a numpy array from the DataFrame: data\_array

data\_array = data.values

# Print the datatype of data\_array to the shell

print(type(data\_array))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

data = pd.read\_csv(file, header = None ,nrows = 5)

Read the first 5 rows of the file into a DataFrame: data

You have just used np.genfromtxt() to import data containing mixed datatypes. There is also another function np.recfromcsv() that behaves similarly to np.genfromtxt(), except that its default dtype is None. In this exercise, you'll practice using this to achieve the same result.

In the video, Justin plotted the histograms by using the pandas library and indexing the DataFrame to extract the desired column. Here, however, you only need to use the provided NumPy array. Also, Justin assigned his plotting statements (except for plt.show()) to the dummy variable \_. This is to prevent unnecessary output from being displayed. It is not required for your solutions to these exercises, however it is good practice to use it. Alternatively, if you are working in an interactive environment such as a Jupyter notebook, you could use a ;after your plotting statements to achieve the same effect. Justin prefers using \_. Therefore, you will see it used in the solution code.

# Compute ECDF for versicolor data: x\_vers, y\_vers

x\_vers, y\_vers = ecdf(versicolor\_petal\_length)

# Generate plot

plt.plot(x\_vers,y\_vers, marker='.', linestyle = 'none')

# Label the axes

\_ = plt.xlabel('versicolor\_petal\_length')

\_ = plt.ylabel('ECDF')

# Display the plot

plt.show()

<https://pythonbuddy.com/>

**data = {‘weekday’: ['Sun', 'Sun., 'Mon', ‘Son'], 'city': ['Austin', 'Dallas.,'Austin' 'DatSas’] ‘Visitors’: [139, 237, 326, 456], ‘signups’: [7, 12, 3, 5]}**

**Great for check syntax errors**

## Logarithm:

An exponent refers to the number of times a number is multiplied by itself.

A logarithm is the power to which a number must be raised in order to get some other number

For example, the base ten logarithm of 100 is 2, because ten raised to the power of two is 100:

**log 100 = 2**

because

**102 = 100**

This is an example of a base-ten logarithm. We call it a base ten logarithm because ten is the number that is raised to a power. The base unit is the number being raised to a power. There are logarithms using different base units. If you wanted, you could use two as a base unit. For instance, the base two logarithm of eight is three, because two raised to the power of three equals eight:

**log2 8 = 3**

because

**23 = 8**

Congratulations on securing your place in the Data Science Full Time Program starting in September 2019!

Before starting the in-class room training, students require a basic knowledge and skills in python.

**Complete Pre-Work**

The following [DataCamp Courses](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2F%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e) are recommended and required for entry into the program:

* [Intro to Python for Data Science](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintro-to-python-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Intermediate Python for Data Science](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintermediate-python-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Statistical Thinking in Python (Part 1)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fstatistical-thinking-in-python-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Statistical Thinking in Python (Part 2)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fstatistical-thinking-in-python-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Importing Data in Python (Part 1)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fimporting-data-in-python-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Importing Data in Python (Part 2)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fimporting-data-in-python-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Python Data Science Toolbox (Part 1)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fpython-data-science-toolbox-part-1%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Python Data Science Toolbox (Part 2)](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fpython-data-science-toolbox-part-2%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)
* [Intro to SQL](https://t.sidekickopen05.com/s1t/c/5/f18dQhb0S7lM8dDMPbW2n0x6l2B9nMJN7t5X-FfhMynW2z8Tgv8rl4jgN56dJ-2fD2R8102?t=https%3A%2F%2Fwww.datacamp.com%2Fcourses%2Fintro-to-sql-for-data-science%3Ftap_a%3D5644-dce66f%26tap_s%3D155299-d0cfee&si=7000000002026730&pi=84dfa37f-f5ff-488e-8746-a092dc756f3e)

Once these courses are complete, you are ready to start your training for the Data Science Full Time Program. Part of the modules are free and there is a small charge (USD$30) to access all of the content. Please save a screenshot when you complete each module to provide to the education provider.

**Full Time Program Payment Plan - Total course fees: $16 500 (Interest Free,** includes $1000 discount**)**

The remaining pre-payment of $2301 will be due before your course commences in September 2019. You will be contacted about this closer to the start date. No repayments are due for the remaining program fees ($14k) for 6 months after starting the program. At 6 months, a 24-month payment plan will commence for the remaining program fees. When you achieve employment, the payment plan term will switch to 12 months. This payment plan will be interest free.

Let me know if you have any questions and congratulations on starting your Data Science Career with Black Cat Data!

Kindest regards,

statistics

Jasleen Kaur

Community Manager

**P** +61 4 3576 7786

# string to experiment with: place

place = "poolhouse"

# Use upper() on place: place\_up

place\_up = place.upper()

# Print out place and place\_up

print(place)

print(place\_up)

# Print out the number of o's in place

print(place.count('o'))

*“Trust in the Lord with all your heart, and do not lean on your own understanding. In all your ways acknowledge him, and he will make straight your paths.”*

**Proverbs 3:5-6**

*"For I know the plans I have for you," declares the LORD, "plans to prosper you and not to harm you, plans to give you hope and a future.”*

**Jeremiah 29:11**

Just as you are confident in God when making your decisions, you should trust Him with the path your life will take. That being said, you should still have an idea of where your life is going! Plans are often necessary to enable yourself to support your partner.

When you ask her her day was, listen! It’s not enough to pretend you’re interested in what she has to say, you need to mean it. We are to love our wives or partners as ourselves, and that means we need to be active and supportive listeners. (Tip: don't try to fix everything immediately. It's often much more helpful to just be there and support her)

# Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Print out drives\_right column as Series

print(cars.loc[:,'drives\_right'])

# Print out drives\_right column as DataFrame

print(cars.loc[:,['drives\_right']])

# Print out cars\_per\_cap and drives\_right as DataFrame

print(cars.loc[:,['cars\_per\_cap', 'drives\_right']])

To filter out missing values:

IS NOT NULL operator.

SELECT name

FROM people

WHERE birthdate IS NOT NULL;

select name

From people

where deathdate IS NULL;

The % wildcard will match zero, one, or many characters in text. For example, the following query matches companies like 'Data', 'DataC' 'DataCamp', 'DataMind', and so on:

SELECT name

FROM companies

WHERE name LIKE 'Data%';

The \_ wildcard will match a *single* character. For example, the following query matches companies like 'DataCamp', 'DataComp', and so on:

SELECT name

FROM companies

WHERE name LIKE 'DataC\_mp';

You can also use the NOT LIKE operator to find records that *don't* match the pattern you specify.

Select name

From people

Where name like '\_r%';

the following gives a result of 1:

SELECT (4 / 3);

What's going on here?

SQL assumes that if you divide an integer by an integer, you want to get an integer back. So be careful when dividing!

If you want more precision when dividing, you can add decimal places to your numbers. For example,

SELECT (4.0 / 3.0) AS result;

gives you the result you would expect: 1.333

|  |
| --- |
| **name** |
| Ara Celi |
| Aramis Knight |
| Arben Bajraktaraj |

select MAX(gross)

from films

Where release\_year between 2000 and 2012;

AVG(column),SUM

MIN(column)

Make A Schedule & Use It Religiously

If you’re like me, there’s something soothing about having your schedule mapped out when life gets hectic. If you’re not like me yet, you’re about to be. Noone can be truly organised unless they take the time to set up a schedule that works realistically for them. Putting it in writing helps keep you honest and avoid those tele binges and the extreme regret you feel when it’s done and you still have 10 million words to write.

That Includes Any Chill Time

The trick is not to get caught up in too much downtime, and the best way to avoid that is to add pleasure to your daily schedule. Maybe that’s an hour of mindless movies, maybe it’s the gym, maybe it’s time with mates.

Use Your Commute

To read and get flash cards

pd.read\_csv(‘cars.csv’, index\_col=0)

Out[8]: a b

0 hello pandas

ndarray[row\_index,column\_index]

Numpy and Pandas

ndarray[row\_index,column\_index]

# or if you want to select all

# columns for a given set of rows

Ndarray[row\_index]

# ***Pandas in Jupyter***

Loading a CSV file as a data frame is pretty easy:

|  |  |
| --- | --- |
| 1 | data\_frame = pandas.read\_csv('file.csv', sep=';') |

Sometimes the CSV file contains padding spaces in front of the values. To ignore them use the *skipinitialspaces* parameter:

|  |  |
| --- | --- |
| 1 | pandas.read\_csv('file.csv', sep=';', skipinitialspace=True) |

If the padding white spaces occur on both sides of the cell values we need to use a regular expression separator. In this case, we need to use the ‘python’ processing engine, instead of the underlying native one, in order to avoid warnings. This will degrade the performance a bit:

|  |  |
| --- | --- |
| 1 | pandas.read\_csv('file.csv', sep='\s\*;\s\*', skipinitialspace=True, engine='python') |

**What are the optional arguments to the function**

function\_1()

**?**

def greet(name, msg = "Good morning!", msg2="Come on in"):

# Open a file: file

file = open('moby\_dick.txt', mode = 'r')

# Print it

print(file.read())

# Check whether file is closed

print(file.closed)

# Close file

file.close()

# Check whether file is closed

print(file.closed)

# Read & print the first 3 lines

with open('moby\_dick.txt') as file:

print(file.readline())

print(file.readline())

print(file.readline())

**Pythonic**

An idea or piece of code which closely follows the most common idioms of the Python language, rather than implementing code using concepts common to other languages. For example, a common idiom in Python is to loop over all elements of an iterable using a [for](https://docs.python.org/3.3/reference/compound_stmts.html#for) statement. Many other languages don’t have this type of construct, so people unfamiliar with Python sometimes use a numerical counter instead:

**for** i **in** range(len(food)):

print(food[i])

As opposed to the cleaner, Pythonic method:

**for** piece **in** food:

print(piece)

# Import numpy

import numpy as np

# Assign the filename: file

file = 'digits\_header.txt'

# Load the data: data

data = np.loadtxt(file, delimiter='\t', skiprows=1, usecols=[0,2]) # usecols is for first column and 3 columns because the header has been remove( skiprows)

# Print data

print(data)

data = np.loadtxt(file, delimiter='\t', skiprows=1, dtype=float) # usecols is for first column and 3 columns because the header has been remove( skiprows)

# Assign the filename: file

file = 'digits.csv'

# Read the first 5 rows of the file into a DataFrame: data

# Build a numpy array from the DataFrame: data\_array

# Print the datatype of data\_array to the shell

print(type(data\_array))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_’

ANswer

# Assign the filename: file

file = 'digits.csv'

# Read the first 5 rows of the file into a DataFrame: data

data = pd.read\_csv(file,header = None, nrows=5)

# Build a numpy array from the DataFrame: data\_array

data\_array = np.array(data)

# Print the datatype of data\_array to the shell

print(type(data\_array))