

Importing Data in Python

January 14, 2020

1 Importing Data in Python (Part1)

As a Data Scientist, on a daily basis you will need to clean data, wrangle and munge it, visualize it, build predictive models and interpret these models. Before doing any of these, however, you will need to know how to get data into Python. In the prequel to this course, you have already learnt many ways to import data into Python: (i) from flat files such as .txts and .csvs; (ii) from files native to other software such as Excel spreadsheets, Stata, SAS and MATLAB files; (iii) from relational databases such as SQLite & PostgreSQL. In this course, you'll extend this knowledge base by learning to import data (i) from the web and (ii) a special and essential case of this: pulling data from Application Programming Interfaces, also known as APIs, such as the Twitter streaming API, which allows us to stream real-time tweets.

1.1 1. Introduction and flat files

In this chapter, you'll learn how to import data into Python from all types of flat files, a simple and prevalent form of data storage. You've previously learned how to use NumPy and pandas - you will learn how to use these packages to import flat files, as well as how to customize your imports.

1.1.1 1.1 Importing entire text files

In this exercise, you'll be working with the file `moby_dick.txt`. It is a text file that contains the opening sentences of Moby Dick, one of the great American novels! Here you'll get experience opening a text file, printing its contents to the shell and, finally, closing it.

Instructions

- Open the file `moby_dick.txt` as read-only and store it in the variable `file`. Make sure to pass the filename enclosed in quotation marks `' '`.
- Print the contents of the file to the shell using the `print()` function. As Hugo showed in the video, you'll need to apply the method `read()` to the object `file`.
- Check whether the file is closed by executing `print(file.closed)`.
- Close the file using the `close()` method.
- Check again that the file is closed as you did above.

```
[2]: # Open a file: file
file = open('./Data/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)
```

CHAPTER 1. Loomings.

Call me Ishmael. Some years ago--never mind how long precisely--having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off--then, I account it high time to get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.

False

True

1.1.2 1.2 Importing text files line by line

For large files, we may not want to print all of their content to the shell: you may wish to print only the first few lines. Enter the `readline()` method, which allows you to do this. When a file called `file` is open, you can print out the first line by executing `file.readline()`. If you execute the same command again, the second line will print, and so on.

In the introductory video, Hugo also introduced the concept of a context manager. He showed that you can bind a variable `file` by using a context manager construct: `~~~` with `open('huck_finn.txt')` as `file`: `~~~`

While still within this construct, the variable `file` will be bound to `open('huck_finn.txt')`; thus, to print the file to the shell, all the code you need to execute is: `~~~` with `open('huck_finn.txt')` as `file`: `print(file.readline())` `~~~` You'll now use these tools to print the first few lines of `moby_dick.txt`!

Instructions

- Open `moby_dick.txt` using the `with` context manager and the variable `file`.

- Print the first three lines of the file to the shell by using `readline()` three times within the context manager.

```
[3]: # Read & print the first 3 lines
with open('./Data/moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())
```

CHAPTER 1. Loomings.

Call me Ishmael. Some years ago--never mind how long precisely--having

1.1.3 1.3 Using NumPy to import flat files

In this exercise, you're now going to load the MNIST digit recognition dataset using the numpy function `loadtxt()` and see just how easy it can be:

The first argument will be the filename. The second will be the delimiter which, in this case, is a comma. You can find more information about the MNIST dataset here on the webpage of Yann LeCun, who is currently Director of AI Research at Facebook and Founding Director of the NYU Center for Data Science, among many other things.

Instructions

- Fill in the arguments of `np.loadtxt()` by passing `file` and a comma `,` for the delimiter.
- Fill in the argument of `print()` to print the type of the object `digits`. Use the function `type()`.
- Execute the rest of the code to visualize one of the rows of the data.

```
[7]: # Import package
import numpy as np
import matplotlib.pyplot as plt

# Assign filename to variable: file
file = './Data/mnist_kaggle_some_rows.csv'

# Load file as array: digits
digits = np.loadtxt(file, delimiter=',')

# Print datatype of digits
print(type(digits))

# Select and reshape a row
im = digits[21, 1:]
im_sq = np.reshape(im, (28, 28))
```

```
# Plot reshaped data (matplotlib.pyplot already loaded as plt)
plt.imshow(im_sq, cmap='Greys', interpolation='nearest')
plt.show()
```

```
<class 'numpy.ndarray'>
```

```
<Figure size 640x480 with 1 Axes>
```

```
[8]: print(digits)
```

```
[[1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [2. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [5. 0. 0. ... 0. 0. 0.]]
```

1.1.4 1.4 Customizing your NumPy import

What if there are rows, such as a header, that you don't want to import? What if your file has a delimiter other than a comma? What if you only wish to import particular columns?

There are a number of arguments that `np.loadtxt()` takes that you'll find useful: `delimiter` changes the delimiter that `loadtxt()` is expecting, for example, you can use `,` and `\t` for comma-delimited and tab-delimited respectively; `skiprows` allows you to specify how many rows (not indices) you wish to skip; `usecols` takes a list of the indices of the columns you wish to keep.

The file that you'll be importing, `digits_header.txt`,

- has a header
- is tab-delimited.

Instructions

- Complete the arguments of `np.loadtxt()`: the file you're importing is tab-delimited, you want to skip the first row and you only want to import the first and third columns.
- Complete the argument of the `print()` call in order to print the entire array that you just imported.

```
[11]: # Import numpy
import numpy as np

# Assign the filename: file
file = './Data/digits_header.txt'

# Load the data: data
data = np.loadtxt(file, delimiter='\t', skiprows=1, usecols=[0,2])

# Print data
```

```
print(data)
```

```
↳ -----  
  
OSError                                Traceback (most recent call↳  
↳last)  
  
    <ipython-input-11-2619080263d8> in <module>  
        6  
        7 # Load the data: data  
----> 8 data = np.loadtxt(file, delimiter='\t', skiprows=1, usecols=[0,2])  
        9  
       10 # Print data  
  
    C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\npio.py in↳  
↳loadtxt(fname, dtype, comments, delimiter, converters, skiprows, usecols,↳  
↳unpack, ndmin, encoding, max_rows)  
       960         fname = os.fspath(fname)  
       961         if _is_string_like(fname):  
--> 962             fh = np.lib._datasource.open(fname, 'rt',↳  
↳encoding=encoding)  
       963             fencoding = getattr(fh, 'encoding', 'latin1')  
       964             fh = iter(fh)  
  
    C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\_datasource.py in↳  
↳open(path, mode, destpath, encoding, newline)  
       264  
       265         ds = DataSource(destpath)  
--> 266         return ds.open(path, mode, encoding=encoding, newline=newline)  
       267  
       268  
  
    C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\_datasource.py in↳  
↳open(self, path, mode, encoding, newline)  
        622                                     encoding=encoding,↳  
↳newline=newline)  
        623     else:  
--> 624         raise IOError("%s not found." % path)  
        625  
        626
```

```
OSError: ./Data/digits_header.txt not found.
```

1.1.5 1.4 Importing different datatypes

The file `seaslug.txt`

- has a text header, consisting of strings
- is tab-delimited.

These data consists of percentage of sea slug larvae that had metamorphosed in a given time period. Read more [here](#).

Due to the header, if you tried to import it as-is using `np.loadtxt()`, Python would throw you a `ValueError` and tell you that it could not convert string to float. There are two ways to deal with this: firstly, you can set the data type argument `dtype` equal to `str` (for string).

Alternatively, you can skip the first row as we have seen before, using the `skiprows` argument.

Instructions

- Complete the first call to `np.loadtxt()` by passing `file` as the first argument.
- Execute `print(data[0])` to print the first element of `data`.
- Complete the second call to `np.loadtxt()`. The file you're importing is tab-delimited, the datatype is `float`, and you want to skip the first row.
- Print the 10th element of `data_float` by completing the `print()` command. Be guided by the previous `print()` call.
- Execute the rest of the code to visualize the data.

```
[12]: # Assign filename: file
file = './Data/seaslug.txt'

# Import file: data
data = np.loadtxt(file, delimiter='\t', dtype=str)

# Print the first element of data
print(data[0])

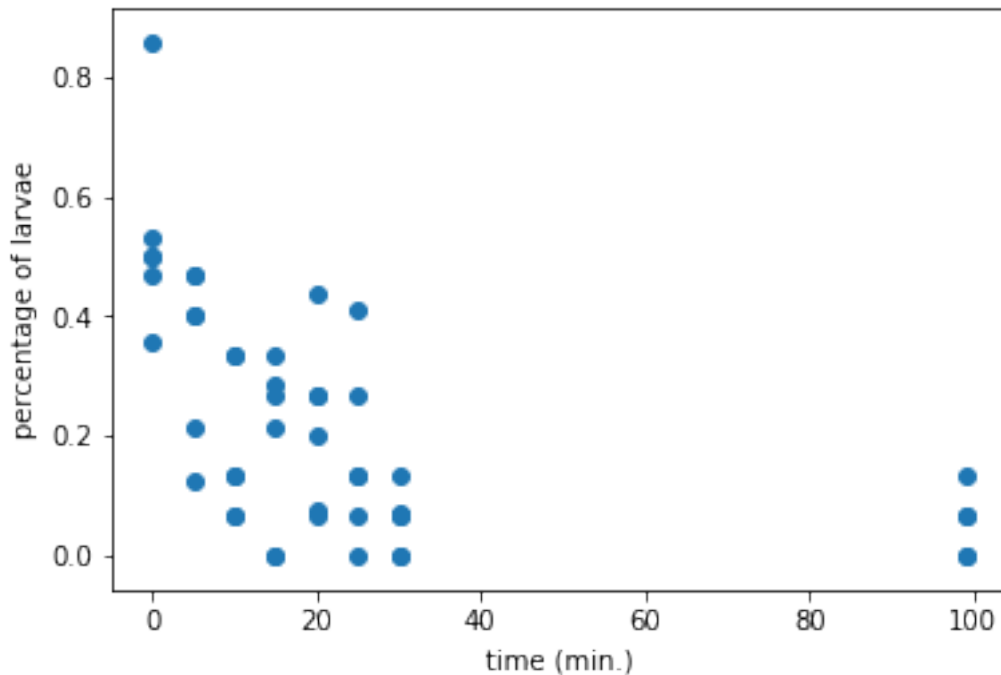
# Import data as floats and skip the first row: data_float
data_float = np.loadtxt(file, delimiter='\t', dtype=float, skiprows=1)

# Print the 10th element of data_float
print(data_float[9])

# Plot a scatterplot of the data
plt.scatter(data_float[:, 0], data_float[:, 1])
plt.xlabel('time (min.)')
plt.ylabel('percentage of larvae')
plt.show()
```

```
['Time' 'Percent']
```

[0. 0.357]



1.1.6 1.5 Working with mixed datatypes (1)

Much of the time you will need to import datasets which have different datatypes in different columns; one column may contain strings and another floats, for example. The function `np.loadtxt()` will freak at this. There is another function, `np.genfromtxt()`, which can handle such structures. If we pass `dtype=None` to it, it will figure out what types each column should be.

Import 'titanic.csv' using the function `np.genfromtxt()` as follows: `~~~ data = np.genfromtxt('titanic.csv', delimiter=',', names=True, dtype=None) ~~~` Here, the first argument is the filename, the second specifies the delimiter, and the third argument names tells us there is a header. Because the data are of different types, data is an object called a structured array. Because numpy arrays have to contain elements that are all the same type, the structured array solves this by being a 1D array, where each element of the array is a row of the flat file imported. You can test this by checking out the array's shape in the shell by executing `np.shape(data)`.

Accessing rows and columns of structured arrays is super-intuitive: to get the *i*th row, merely execute `data[i]` and to get the column with name 'Fare', execute `data['Fare']`.

Print the entire column with name Survived to the shell. What are the last 4 values of this column?

Instructions

Possible Answers

1,0,0,1.

1,2,0,0.
1,0,1,0.
0,1,1,1.

```
[15]: data = np.genfromtxt('./Data/titanic.csv', delimiter=',', names=True,
    ↳ dtype=None)
```

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1:
VisibleDeprecationWarning: Reading unicode strings without specifying the
encoding argument is deprecated. Set the encoding, use None for the system
default.

"""Entry point for launching an IPython kernel.

```
[21]: print(data)
```

```
[( 1, 0, 3, b'male', 22. , 1, 0, b'A/5 21171', 7.25 , b'', b'S')
 ( 2, 1, 1, b'female', 38. , 1, 0, b'PC 17599', 71.2833, b'C85', b'C')
 ( 3, 1, 3, b'female', 26. , 0, 0, b'STON/02. 3101282', 7.925 , b'', b'S')
 ( 4, 1, 1, b'female', 35. , 1, 0, b'113803', 53.1 , b'C123', b'S')
 ( 5, 0, 3, b'male', 35. , 0, 0, b'373450', 8.05 , b'', b'S')
 ( 6, 0, 3, b'male', nan, 0, 0, b'330877', 8.4583, b'', b'Q')
 ( 7, 0, 1, b'male', 54. , 0, 0, b'17463', 51.8625, b'E46', b'S')
 ( 8, 0, 3, b'male', 2. , 3, 1, b'349909', 21.075 , b'', b'S')
 ( 9, 1, 3, b'female', 27. , 0, 2, b'347742', 11.1333, b'', b'S')
 (10, 1, 2, b'female', 14. , 1, 0, b'237736', 30.0708, b'', b'C')
 (11, 1, 3, b'female', 4. , 1, 1, b'PP 9549', 16.7 , b'G6', b'S')
 (12, 1, 1, b'female', 58. , 0, 0, b'113783', 26.55 , b'C103', b'S')
 (13, 0, 3, b'male', 20. , 0, 0, b'A/5. 2151', 8.05 , b'', b'S')
 (14, 0, 3, b'male', 39. , 1, 5, b'347082', 31.275 , b'', b'S')
 (15, 0, 3, b'female', 14. , 0, 0, b'350406', 7.8542, b'', b'S')
 (16, 1, 2, b'female', 55. , 0, 0, b'248706', 16. , b'', b'S')
 (17, 0, 3, b'male', 2. , 4, 1, b'382652', 29.125 , b'', b'Q')
 (18, 1, 2, b'male', nan, 0, 0, b'244373', 13. , b'', b'S')
 (19, 0, 3, b'female', 31. , 1, 0, b'345763', 18. , b'', b'S')
 (20, 1, 3, b'female', nan, 0, 0, b'2649', 7.225 , b'', b'C')
 (21, 0, 2, b'male', 35. , 0, 0, b'239865', 26. , b'', b'S')
 (22, 1, 2, b'male', 34. , 0, 0, b'248698', 13. , b'D56', b'S')
 (23, 1, 3, b'female', 15. , 0, 0, b'330923', 8.0292, b'', b'Q')
 (24, 1, 1, b'male', 28. , 0, 0, b'113788', 35.5 , b'A6', b'S')
 (25, 0, 3, b'female', 8. , 3, 1, b'349909', 21.075 , b'', b'S')
 (26, 1, 3, b'female', 38. , 1, 5, b'347077', 31.3875, b'', b'S')
 (27, 0, 3, b'male', nan, 0, 0, b'2631', 7.225 , b'', b'C')
 (28, 0, 1, b'male', 19. , 3, 2, b'19950', 263. , b'C23 C25 C27', b'S')
 (29, 1, 3, b'female', nan, 0, 0, b'330959', 7.8792, b'', b'Q')
 (30, 0, 3, b'male', nan, 0, 0, b'349216', 7.8958, b'', b'S')
 (31, 0, 1, b'male', 40. , 0, 0, b'PC 17601', 27.7208, b'', b'C')
 (32, 1, 1, b'female', nan, 1, 0, b'PC 17569', 146.5208, b'B78', b'C')
 (33, 1, 3, b'female', nan, 0, 0, b'335677', 7.75 , b'', b'Q')]
```


(34, 0, 2, b'male', 66. , 0, 0, b'C.A. 24579', 10.5 , b'', b'S')
 (35, 0, 1, b'male', 28. , 1, 0, b'PC 17604', 82.1708, b'', b'C')
 (36, 0, 1, b'male', 42. , 1, 0, b'113789', 52. , b'', b'S')
 (37, 1, 3, b'male', nan, 0, 0, b'2677', 7.2292, b'', b'C')
 (38, 0, 3, b'male', 21. , 0, 0, b'A./5. 2152', 8.05 , b'', b'S')
 (39, 0, 3, b'female', 18. , 2, 0, b'345764', 18. , b'', b'S')
 (40, 1, 3, b'female', 14. , 1, 0, b'2651', 11.2417, b'', b'C')
 (41, 0, 3, b'female', 40. , 1, 0, b'7546', 9.475 , b'', b'S')
 (42, 0, 2, b'female', 27. , 1, 0, b'11668', 21. , b'', b'S')
 (43, 0, 3, b'male', nan, 0, 0, b'349253', 7.8958, b'', b'C')
 (44, 1, 2, b'female', 3. , 1, 2, b'SC/Paris 2123', 41.5792, b'', b'C')
 (45, 1, 3, b'female', 19. , 0, 0, b'330958', 7.8792, b'', b'Q')
 (46, 0, 3, b'male', nan, 0, 0, b'S.C./A.4. 23567', 8.05 , b'', b'S')
 (47, 0, 3, b'male', nan, 1, 0, b'370371', 15.5 , b'', b'Q')
 (48, 1, 3, b'female', nan, 0, 0, b'14311', 7.75 , b'', b'Q')
 (49, 0, 3, b'male', nan, 2, 0, b'2662', 21.6792, b'', b'C')
 (50, 0, 3, b'female', 18. , 1, 0, b'349237', 17.8 , b'', b'S')
 (51, 0, 3, b'male', 7. , 4, 1, b'3101295', 39.6875, b'', b'S')
 (52, 0, 3, b'male', 21. , 0, 0, b'A/4. 39886', 7.8 , b'', b'S')
 (53, 1, 1, b'female', 49. , 1, 0, b'PC 17572', 76.7292, b'D33', b'C')
 (54, 1, 2, b'female', 29. , 1, 0, b'2926', 26. , b'', b'S')
 (55, 0, 1, b'male', 65. , 0, 1, b'113509', 61.9792, b'B30', b'C')
 (56, 1, 1, b'male', nan, 0, 0, b'19947', 35.5 , b'C52', b'S')
 (57, 1, 2, b'female', 21. , 0, 0, b'C.A. 31026', 10.5 , b'', b'S')
 (58, 0, 3, b'male', 28.5 , 0, 0, b'2697', 7.2292, b'', b'C')
 (59, 1, 2, b'female', 5. , 1, 2, b'C.A. 34651', 27.75 , b'', b'S')
 (60, 0, 3, b'male', 11. , 5, 2, b'CA 2144', 46.9 , b'', b'S')
 (61, 0, 3, b'male', 22. , 0, 0, b'2669', 7.2292, b'', b'C')
 (62, 1, 1, b'female', 38. , 0, 0, b'113572', 80. , b'B28', b'')
 (63, 0, 1, b'male', 45. , 1, 0, b'36973', 83.475 , b'C83', b'S')
 (64, 0, 3, b'male', 4. , 3, 2, b'347088', 27.9 , b'', b'S')
 (65, 0, 1, b'male', nan, 0, 0, b'PC 17605', 27.7208, b'', b'C')
 (66, 1, 3, b'male', nan, 1, 1, b'2661', 15.2458, b'', b'C')
 (67, 1, 2, b'female', 29. , 0, 0, b'C.A. 29395', 10.5 , b'F33', b'S')
 (68, 0, 3, b'male', 19. , 0, 0, b'S.P. 3464', 8.1583, b'', b'S')
 (69, 1, 3, b'female', 17. , 4, 2, b'3101281', 7.925 , b'', b'S')
 (70, 0, 3, b'male', 26. , 2, 0, b'315151', 8.6625, b'', b'S')
 (71, 0, 2, b'male', 32. , 0, 0, b'C.A. 33111', 10.5 , b'', b'S')
 (72, 0, 3, b'female', 16. , 5, 2, b'CA 2144', 46.9 , b'', b'S')
 (73, 0, 2, b'male', 21. , 0, 0, b'S.O.C. 14879', 73.5 , b'', b'S')
 (74, 0, 3, b'male', 26. , 1, 0, b'2680', 14.4542, b'', b'C')
 (75, 1, 3, b'male', 32. , 0, 0, b'1601', 56.4958, b'', b'S')
 (76, 0, 3, b'male', 25. , 0, 0, b'348123', 7.65 , b'F G73', b'S')
 (77, 0, 3, b'male', nan, 0, 0, b'349208', 7.8958, b'', b'S')
 (78, 0, 3, b'male', nan, 0, 0, b'374746', 8.05 , b'', b'S')
 (79, 1, 2, b'male', 0.83, 0, 2, b'248738', 29. , b'', b'S')
 (80, 1, 3, b'female', 30. , 0, 0, b'364516', 12.475 , b'', b'S')
 (81, 0, 3, b'male', 22. , 0, 0, b'345767', 9. , b'', b'S')

(82, 1, 3, b'male', 29. , 0, 0, b'345779', 9.5 , b'', b'S')
 (83, 1, 3, b'female', nan, 0, 0, b'330932', 7.7875, b'', b'Q')
 (84, 0, 1, b'male', 28. , 0, 0, b'113059', 47.1 , b'', b'S')
 (85, 1, 2, b'female', 17. , 0, 0, b'S0/C 14885', 10.5 , b'', b'S')
 (86, 1, 3, b'female', 33. , 3, 0, b'3101278', 15.85 , b'', b'S')
 (87, 0, 3, b'male', 16. , 1, 3, b'W./C. 6608', 34.375 , b'', b'S')
 (88, 0, 3, b'male', nan, 0, 0, b'SOTON/OQ 392086', 8.05 , b'', b'S')
 (89, 1, 1, b'female', 23. , 3, 2, b'19950', 263. , b'C23 C25 C27', b'S')
 (90, 0, 3, b'male', 24. , 0, 0, b'343275', 8.05 , b'', b'S')
 (91, 0, 3, b'male', 29. , 0, 0, b'343276', 8.05 , b'', b'S')
 (92, 0, 3, b'male', 20. , 0, 0, b'347466', 7.8542, b'', b'S')
 (93, 0, 1, b'male', 46. , 1, 0, b'W.E.P. 5734', 61.175 , b'E31', b'S')
 (94, 0, 3, b'male', 26. , 1, 2, b'C.A. 2315', 20.575 , b'', b'S')
 (95, 0, 3, b'male', 59. , 0, 0, b'364500', 7.25 , b'', b'S')
 (96, 0, 3, b'male', nan, 0, 0, b'374910', 8.05 , b'', b'S')
 (97, 0, 1, b'male', 71. , 0, 0, b'PC 17754', 34.6542, b'A5', b'C')
 (98, 1, 1, b'male', 23. , 0, 1, b'PC 17759', 63.3583, b'D10 D12', b'C')
 (99, 1, 2, b'female', 34. , 0, 1, b'231919', 23. , b'', b'S')
 (100, 0, 2, b'male', 34. , 1, 0, b'244367', 26. , b'', b'S')
 (101, 0, 3, b'female', 28. , 0, 0, b'349245', 7.8958, b'', b'S')
 (102, 0, 3, b'male', nan, 0, 0, b'349215', 7.8958, b'', b'S')
 (103, 0, 1, b'male', 21. , 0, 1, b'35281', 77.2875, b'D26', b'S')
 (104, 0, 3, b'male', 33. , 0, 0, b'7540', 8.6542, b'', b'S')
 (105, 0, 3, b'male', 37. , 2, 0, b'3101276', 7.925 , b'', b'S')
 (106, 0, 3, b'male', 28. , 0, 0, b'349207', 7.8958, b'', b'S')
 (107, 1, 3, b'female', 21. , 0, 0, b'343120', 7.65 , b'', b'S')
 (108, 1, 3, b'male', nan, 0, 0, b'312991', 7.775 , b'', b'S')
 (109, 0, 3, b'male', 38. , 0, 0, b'349249', 7.8958, b'', b'S')
 (110, 1, 3, b'female', nan, 1, 0, b'371110', 24.15 , b'', b'Q')
 (111, 0, 1, b'male', 47. , 0, 0, b'110465', 52. , b'C110', b'S')
 (112, 0, 3, b'female', 14.5 , 1, 0, b'2665', 14.4542, b'', b'C')
 (113, 0, 3, b'male', 22. , 0, 0, b'324669', 8.05 , b'', b'S')
 (114, 0, 3, b'female', 20. , 1, 0, b'4136', 9.825 , b'', b'S')
 (115, 0, 3, b'female', 17. , 0, 0, b'2627', 14.4583, b'', b'C')
 (116, 0, 3, b'male', 21. , 0, 0, b'STON/O 2. 3101294', 7.925 , b'', b'S')
 (117, 0, 3, b'male', 70.5 , 0, 0, b'370369', 7.75 , b'', b'Q')
 (118, 0, 2, b'male', 29. , 1, 0, b'11668', 21. , b'', b'S')
 (119, 0, 1, b'male', 24. , 0, 1, b'PC 17558', 247.5208, b'B58 B60', b'C')
 (120, 0, 3, b'female', 2. , 4, 2, b'347082', 31.275 , b'', b'S')
 (121, 0, 2, b'male', 21. , 2, 0, b'S.O.C. 14879', 73.5 , b'', b'S')
 (122, 0, 3, b'male', nan, 0, 0, b'A4. 54510', 8.05 , b'', b'S')
 (123, 0, 2, b'male', 32.5 , 1, 0, b'237736', 30.0708, b'', b'C')
 (124, 1, 2, b'female', 32.5 , 0, 0, b'27267', 13. , b'E101', b'S')
 (125, 0, 1, b'male', 54. , 0, 1, b'35281', 77.2875, b'D26', b'S')
 (126, 1, 3, b'male', 12. , 1, 0, b'2651', 11.2417, b'', b'C')
 (127, 0, 3, b'male', nan, 0, 0, b'370372', 7.75 , b'', b'Q')
 (128, 1, 3, b'male', 24. , 0, 0, b'C 17369', 7.1417, b'', b'S')
 (129, 1, 3, b'female', nan, 1, 1, b'2668', 22.3583, b'F E69', b'C')

(130, 0, 3, b'male', 45. , 0, 0, b'347061', 6.975 , b'', b'S')
 (131, 0, 3, b'male', 33. , 0, 0, b'349241', 7.8958, b'', b'C')
 (132, 0, 3, b'male', 20. , 0, 0, b'SOTON/O.Q. 3101307', 7.05 , b'', b'S')
 (133, 0, 3, b'female', 47. , 1, 0, b'A/5. 3337', 14.5 , b'', b'S')
 (134, 1, 2, b'female', 29. , 1, 0, b'228414', 26. , b'', b'S')
 (135, 0, 2, b'male', 25. , 0, 0, b'C.A. 29178', 13. , b'', b'S')
 (136, 0, 2, b'male', 23. , 0, 0, b'SC/PARIS 2133', 15.0458, b'', b'C')
 (137, 1, 1, b'female', 19. , 0, 2, b'11752', 26.2833, b'D47', b'S')
 (138, 0, 1, b'male', 37. , 1, 0, b'113803', 53.1 , b'C123', b'S')
 (139, 0, 3, b'male', 16. , 0, 0, b'7534', 9.2167, b'', b'S')
 (140, 0, 1, b'male', 24. , 0, 0, b'PC 17593', 79.2 , b'B86', b'C')
 (141, 0, 3, b'female', nan, 0, 2, b'2678', 15.2458, b'', b'C')
 (142, 1, 3, b'female', 22. , 0, 0, b'347081', 7.75 , b'', b'S')
 (143, 1, 3, b'female', 24. , 1, 0, b'STON/O2. 3101279', 15.85 , b'', b'S')
 (144, 0, 3, b'male', 19. , 0, 0, b'365222', 6.75 , b'', b'Q')
 (145, 0, 2, b'male', 18. , 0, 0, b'231945', 11.5 , b'', b'S')
 (146, 0, 2, b'male', 19. , 1, 1, b'C.A. 33112', 36.75 , b'', b'S')
 (147, 1, 3, b'male', 27. , 0, 0, b'350043', 7.7958, b'', b'S')
 (148, 0, 3, b'female', 9. , 2, 2, b'W./C. 6608', 34.375 , b'', b'S')
 (149, 0, 2, b'male', 36.5 , 0, 2, b'230080', 26. , b'F2', b'S')
 (150, 0, 2, b'male', 42. , 0, 0, b'244310', 13. , b'', b'S')
 (151, 0, 2, b'male', 51. , 0, 0, b'S.O.P. 1166', 12.525 , b'', b'S')
 (152, 1, 1, b'female', 22. , 1, 0, b'113776', 66.6 , b'C2', b'S')
 (153, 0, 3, b'male', 55.5 , 0, 0, b'A.5. 11206', 8.05 , b'', b'S')
 (154, 0, 3, b'male', 40.5 , 0, 2, b'A/5. 851', 14.5 , b'', b'S')
 (155, 0, 3, b'male', nan, 0, 0, b'Fa 265302', 7.3125, b'', b'S')
 (156, 0, 1, b'male', 51. , 0, 1, b'PC 17597', 61.3792, b'', b'C')
 (157, 1, 3, b'female', 16. , 0, 0, b'35851', 7.7333, b'', b'Q')
 (158, 0, 3, b'male', 30. , 0, 0, b'SOTON/OQ 392090', 8.05 , b'', b'S')
 (159, 0, 3, b'male', nan, 0, 0, b'315037', 8.6625, b'', b'S')
 (160, 0, 3, b'male', nan, 8, 2, b'CA. 2343', 69.55 , b'', b'S')
 (161, 0, 3, b'male', 44. , 0, 1, b'371362', 16.1 , b'', b'S')
 (162, 1, 2, b'female', 40. , 0, 0, b'C.A. 33595', 15.75 , b'', b'S')
 (163, 0, 3, b'male', 26. , 0, 0, b'347068', 7.775 , b'', b'S')
 (164, 0, 3, b'male', 17. , 0, 0, b'315093', 8.6625, b'', b'S')
 (165, 0, 3, b'male', 1. , 4, 1, b'3101295', 39.6875, b'', b'S')
 (166, 1, 3, b'male', 9. , 0, 2, b'363291', 20.525 , b'', b'S')
 (167, 1, 1, b'female', nan, 0, 1, b'113505', 55. , b'E33', b'S')
 (168, 0, 3, b'female', 45. , 1, 4, b'347088', 27.9 , b'', b'S')
 (169, 0, 1, b'male', nan, 0, 0, b'PC 17318', 25.925 , b'', b'S')
 (170, 0, 3, b'male', 28. , 0, 0, b'1601', 56.4958, b'', b'S')
 (171, 0, 1, b'male', 61. , 0, 0, b'111240', 33.5 , b'B19', b'S')
 (172, 0, 3, b'male', 4. , 4, 1, b'382652', 29.125 , b'', b'Q')
 (173, 1, 3, b'female', 1. , 1, 1, b'347742', 11.1333, b'', b'S')
 (174, 0, 3, b'male', 21. , 0, 0, b'STON/O 2. 3101280', 7.925 , b'', b'S')
 (175, 0, 1, b'male', 56. , 0, 0, b'17764', 30.6958, b'A7', b'C')
 (176, 0, 3, b'male', 18. , 1, 1, b'350404', 7.8542, b'', b'S')
 (177, 0, 3, b'male', nan, 3, 1, b'4133', 25.4667, b'', b'S')

(178, 0, 1, b'female', 50. , 0, 0, b'PC 17595', 28.7125, b'C49', b'C')
 (179, 0, 2, b'male', 30. , 0, 0, b'250653', 13. , b'', b'S')
 (180, 0, 3, b'male', 36. , 0, 0, b'LINE', 0. , b'', b'S')
 (181, 0, 3, b'female', nan, 8, 2, b'CA. 2343', 69.55 , b'', b'S')
 (182, 0, 2, b'male', nan, 0, 0, b'SC/PARIS 2131', 15.05 , b'', b'C')
 (183, 0, 3, b'male', 9. , 4, 2, b'347077', 31.3875, b'', b'S')
 (184, 1, 2, b'male', 1. , 2, 1, b'230136', 39. , b'F4', b'S')
 (185, 1, 3, b'female', 4. , 0, 2, b'315153', 22.025 , b'', b'S')
 (186, 0, 1, b'male', nan, 0, 0, b'113767', 50. , b'A32', b'S')
 (187, 1, 3, b'female', nan, 1, 0, b'370365', 15.5 , b'', b'Q')
 (188, 1, 1, b'male', 45. , 0, 0, b'111428', 26.55 , b'', b'S')
 (189, 0, 3, b'male', 40. , 1, 1, b'364849', 15.5 , b'', b'Q')
 (190, 0, 3, b'male', 36. , 0, 0, b'349247', 7.8958, b'', b'S')
 (191, 1, 2, b'female', 32. , 0, 0, b'234604', 13. , b'', b'S')
 (192, 0, 2, b'male', 19. , 0, 0, b'28424', 13. , b'', b'S')
 (193, 1, 3, b'female', 19. , 1, 0, b'350046', 7.8542, b'', b'S')
 (194, 1, 2, b'male', 3. , 1, 1, b'230080', 26. , b'F2', b'S')
 (195, 1, 1, b'female', 44. , 0, 0, b'PC 17610', 27.7208, b'B4', b'C')
 (196, 1, 1, b'female', 58. , 0, 0, b'PC 17569', 146.5208, b'B80', b'C')
 (197, 0, 3, b'male', nan, 0, 0, b'368703', 7.75 , b'', b'Q')
 (198, 0, 3, b'male', 42. , 0, 1, b'4579', 8.4042, b'', b'S')
 (199, 1, 3, b'female', nan, 0, 0, b'370370', 7.75 , b'', b'Q')
 (200, 0, 2, b'female', 24. , 0, 0, b'248747', 13. , b'', b'S')
 (201, 0, 3, b'male', 28. , 0, 0, b'345770', 9.5 , b'', b'S')
 (202, 0, 3, b'male', nan, 8, 2, b'CA. 2343', 69.55 , b'', b'S')
 (203, 0, 3, b'male', 34. , 0, 0, b'3101264', 6.4958, b'', b'S')
 (204, 0, 3, b'male', 45.5 , 0, 0, b'2628', 7.225 , b'', b'C')
 (205, 1, 3, b'male', 18. , 0, 0, b'A/5 3540', 8.05 , b'', b'S')
 (206, 0, 3, b'female', 2. , 0, 1, b'347054', 10.4625, b'G6', b'S')
 (207, 0, 3, b'male', 32. , 1, 0, b'3101278', 15.85 , b'', b'S')
 (208, 1, 3, b'male', 26. , 0, 0, b'2699', 18.7875, b'', b'C')
 (209, 1, 3, b'female', 16. , 0, 0, b'367231', 7.75 , b'', b'Q')
 (210, 1, 1, b'male', 40. , 0, 0, b'112277', 31. , b'A31', b'C')
 (211, 0, 3, b'male', 24. , 0, 0, b'SOTON/O.Q. 3101311', 7.05 , b'', b'S')
 (212, 1, 2, b'female', 35. , 0, 0, b'F.C.C. 13528', 21. , b'', b'S')
 (213, 0, 3, b'male', 22. , 0, 0, b'A/5 21174', 7.25 , b'', b'S')
 (214, 0, 2, b'male', 30. , 0, 0, b'250646', 13. , b'', b'S')
 (215, 0, 3, b'male', nan, 1, 0, b'367229', 7.75 , b'', b'Q')
 (216, 1, 1, b'female', 31. , 1, 0, b'35273', 113.275 , b'D36', b'C')
 (217, 1, 3, b'female', 27. , 0, 0, b'STON/02. 3101283', 7.925 , b'', b'S')
 (218, 0, 2, b'male', 42. , 1, 0, b'243847', 27. , b'', b'S')
 (219, 1, 1, b'female', 32. , 0, 0, b'11813', 76.2917, b'D15', b'C')
 (220, 0, 2, b'male', 30. , 0, 0, b'W/C 14208', 10.5 , b'', b'S')
 (221, 1, 3, b'male', 16. , 0, 0, b'SOTON/OQ 392089', 8.05 , b'', b'S')
 (222, 0, 2, b'male', 27. , 0, 0, b'220367', 13. , b'', b'S')
 (223, 0, 3, b'male', 51. , 0, 0, b'21440', 8.05 , b'', b'S')
 (224, 0, 3, b'male', nan, 0, 0, b'349234', 7.8958, b'', b'S')
 (225, 1, 1, b'male', 38. , 1, 0, b'19943', 90. , b'C93', b'S')

(226, 0, 3, b'male', 22. , 0, 0, b'PP 4348', 9.35 , b'', b'S')
 (227, 1, 2, b'male', 19. , 0, 0, b'SW/PP 751', 10.5 , b'', b'S')
 (228, 0, 3, b'male', 20.5 , 0, 0, b'A/5 21173', 7.25 , b'', b'S')
 (229, 0, 2, b'male', 18. , 0, 0, b'236171', 13. , b'', b'S')
 (230, 0, 3, b'female', nan, 3, 1, b'4133', 25.4667, b'', b'S')
 (231, 1, 1, b'female', 35. , 1, 0, b'36973', 83.475 , b'C83', b'S')
 (232, 0, 3, b'male', 29. , 0, 0, b'347067', 7.775 , b'', b'S')
 (233, 0, 2, b'male', 59. , 0, 0, b'237442', 13.5 , b'', b'S')
 (234, 1, 3, b'female', 5. , 4, 2, b'347077', 31.3875, b'', b'S')
 (235, 0, 2, b'male', 24. , 0, 0, b'C.A. 29566', 10.5 , b'', b'S')
 (236, 0, 3, b'female', nan, 0, 0, b'W./C. 6609', 7.55 , b'', b'S')
 (237, 0, 2, b'male', 44. , 1, 0, b'26707', 26. , b'', b'S')
 (238, 1, 2, b'female', 8. , 0, 2, b'C.A. 31921', 26.25 , b'', b'S')
 (239, 0, 2, b'male', 19. , 0, 0, b'28665', 10.5 , b'', b'S')
 (240, 0, 2, b'male', 33. , 0, 0, b'SCO/W 1585', 12.275 , b'', b'S')
 (241, 0, 3, b'female', nan, 1, 0, b'2665', 14.4542, b'', b'C')
 (242, 1, 3, b'female', nan, 1, 0, b'367230', 15.5 , b'', b'Q')
 (243, 0, 2, b'male', 29. , 0, 0, b'W./C. 14263', 10.5 , b'', b'S')
 (244, 0, 3, b'male', 22. , 0, 0, b'STON/O 2. 3101275', 7.125 , b'', b'S')
 (245, 0, 3, b'male', 30. , 0, 0, b'2694', 7.225 , b'', b'C')
 (246, 0, 1, b'male', 44. , 2, 0, b'19928', 90. , b'C78', b'Q')
 (247, 0, 3, b'female', 25. , 0, 0, b'347071', 7.775 , b'', b'S')
 (248, 1, 2, b'female', 24. , 0, 2, b'250649', 14.5 , b'', b'S')
 (249, 1, 1, b'male', 37. , 1, 1, b'11751', 52.5542, b'D35', b'S')
 (250, 0, 2, b'male', 54. , 1, 0, b'244252', 26. , b'', b'S')
 (251, 0, 3, b'male', nan, 0, 0, b'362316', 7.25 , b'', b'S')
 (252, 0, 3, b'female', 29. , 1, 1, b'347054', 10.4625, b'G6', b'S')
 (253, 0, 1, b'male', 62. , 0, 0, b'113514', 26.55 , b'C87', b'S')
 (254, 0, 3, b'male', 30. , 1, 0, b'A/5. 3336', 16.1 , b'', b'S')
 (255, 0, 3, b'female', 41. , 0, 2, b'370129', 20.2125, b'', b'S')
 (256, 1, 3, b'female', 29. , 0, 2, b'2650', 15.2458, b'', b'C')
 (257, 1, 1, b'female', nan, 0, 0, b'PC 17585', 79.2 , b'', b'C')
 (258, 1, 1, b'female', 30. , 0, 0, b'110152', 86.5 , b'B77', b'S')
 (259, 1, 1, b'female', 35. , 0, 0, b'PC 17755', 512.3292, b'', b'C')
 (260, 1, 2, b'female', 50. , 0, 1, b'230433', 26. , b'', b'S')
 (261, 0, 3, b'male', nan, 0, 0, b'384461', 7.75 , b'', b'Q')
 (262, 1, 3, b'male', 3. , 4, 2, b'347077', 31.3875, b'', b'S')
 (263, 0, 1, b'male', 52. , 1, 1, b'110413', 79.65 , b'E67', b'S')
 (264, 0, 1, b'male', 40. , 0, 0, b'112059', 0. , b'B94', b'S')
 (265, 0, 3, b'female', nan, 0, 0, b'382649', 7.75 , b'', b'Q')
 (266, 0, 2, b'male', 36. , 0, 0, b'C.A. 17248', 10.5 , b'', b'S')
 (267, 0, 3, b'male', 16. , 4, 1, b'3101295', 39.6875, b'', b'S')
 (268, 1, 3, b'male', 25. , 1, 0, b'347083', 7.775 , b'', b'S')
 (269, 1, 1, b'female', 58. , 0, 1, b'PC 17582', 153.4625, b'C125', b'S')
 (270, 1, 1, b'female', 35. , 0, 0, b'PC 17760', 135.6333, b'C99', b'S')
 (271, 0, 1, b'male', nan, 0, 0, b'113798', 31. , b'', b'S')
 (272, 1, 3, b'male', 25. , 0, 0, b'LINE', 0. , b'', b'S')
 (273, 1, 2, b'female', 41. , 0, 1, b'250644', 19.5 , b'', b'S')

(274, 0, 1, b'male', 37. , 0, 1, b'PC 17596', 29.7 , b'C118', b'C')
 (275, 1, 3, b'female', nan, 0, 0, b'370375', 7.75 , b'', b'Q')
 (276, 1, 1, b'female', 63. , 1, 0, b'13502', 77.9583, b'D7', b'S')
 (277, 0, 3, b'female', 45. , 0, 0, b'347073', 7.75 , b'', b'S')
 (278, 0, 2, b'male', nan, 0, 0, b'239853', 0. , b'', b'S')
 (279, 0, 3, b'male', 7. , 4, 1, b'382652', 29.125 , b'', b'Q')
 (280, 1, 3, b'female', 35. , 1, 1, b'C.A. 2673', 20.25 , b'', b'S')
 (281, 0, 3, b'male', 65. , 0, 0, b'336439', 7.75 , b'', b'Q')
 (282, 0, 3, b'male', 28. , 0, 0, b'347464', 7.8542, b'', b'S')
 (283, 0, 3, b'male', 16. , 0, 0, b'345778', 9.5 , b'', b'S')
 (284, 1, 3, b'male', 19. , 0, 0, b'A/5. 10482', 8.05 , b'', b'S')
 (285, 0, 1, b'male', nan, 0, 0, b'113056', 26. , b'A19', b'S')
 (286, 0, 3, b'male', 33. , 0, 0, b'349239', 8.6625, b'', b'C')
 (287, 1, 3, b'male', 30. , 0, 0, b'345774', 9.5 , b'', b'S')
 (288, 0, 3, b'male', 22. , 0, 0, b'349206', 7.8958, b'', b'S')
 (289, 1, 2, b'male', 42. , 0, 0, b'237798', 13. , b'', b'S')
 (290, 1, 3, b'female', 22. , 0, 0, b'370373', 7.75 , b'', b'Q')
 (291, 1, 1, b'female', 26. , 0, 0, b'19877', 78.85 , b'', b'S')
 (292, 1, 1, b'female', 19. , 1, 0, b'11967', 91.0792, b'B49', b'C')
 (293, 0, 2, b'male', 36. , 0, 0, b'SC/Paris 2163', 12.875 , b'D', b'C')
 (294, 0, 3, b'female', 24. , 0, 0, b'349236', 8.85 , b'', b'S')
 (295, 0, 3, b'male', 24. , 0, 0, b'349233', 7.8958, b'', b'S')
 (296, 0, 1, b'male', nan, 0, 0, b'PC 17612', 27.7208, b'', b'C')
 (297, 0, 3, b'male', 23.5 , 0, 0, b'2693', 7.2292, b'', b'C')
 (298, 0, 1, b'female', 2. , 1, 2, b'113781', 151.55 , b'C22 C26', b'S')
 (299, 1, 1, b'male', nan, 0, 0, b'19988', 30.5 , b'C106', b'S')
 (300, 1, 1, b'female', 50. , 0, 1, b'PC 17558', 247.5208, b'B58 B60', b'C')
 (301, 1, 3, b'female', nan, 0, 0, b'9234', 7.75 , b'', b'Q')
 (302, 1, 3, b'male', nan, 2, 0, b'367226', 23.25 , b'', b'Q')
 (303, 0, 3, b'male', 19. , 0, 0, b'LINE', 0. , b'', b'S')
 (304, 1, 2, b'female', nan, 0, 0, b'226593', 12.35 , b'E101', b'Q')
 (305, 0, 3, b'male', nan, 0, 0, b'A/5 2466', 8.05 , b'', b'S')
 (306, 1, 1, b'male', 0.92, 1, 2, b'113781', 151.55 , b'C22 C26', b'S')
 (307, 1, 1, b'female', nan, 0, 0, b'17421', 110.8833, b'', b'C')
 (308, 1, 1, b'female', 17. , 1, 0, b'PC 17758', 108.9 , b'C65', b'C')
 (309, 0, 2, b'male', 30. , 1, 0, b'P/PP 3381', 24. , b'', b'C')
 (310, 1, 1, b'female', 30. , 0, 0, b'PC 17485', 56.9292, b'E36', b'C')
 (311, 1, 1, b'female', 24. , 0, 0, b'11767', 83.1583, b'C54', b'C')
 (312, 1, 1, b'female', 18. , 2, 2, b'PC 17608', 262.375 , b'B57 B59 B63 B66',
 b'C')
 (313, 0, 2, b'female', 26. , 1, 1, b'250651', 26. , b'', b'S')
 (314, 0, 3, b'male', 28. , 0, 0, b'349243', 7.8958, b'', b'S')
 (315, 0, 2, b'male', 43. , 1, 1, b'F.C.C. 13529', 26.25 , b'', b'S')
 (316, 1, 3, b'female', 26. , 0, 0, b'347470', 7.8542, b'', b'S')
 (317, 1, 2, b'female', 24. , 1, 0, b'244367', 26. , b'', b'S')
 (318, 0, 2, b'male', 54. , 0, 0, b'29011', 14. , b'', b'S')
 (319, 1, 1, b'female', 31. , 0, 2, b'36928', 164.8667, b'C7', b'S')
 (320, 1, 1, b'female', 40. , 1, 1, b'16966', 134.5 , b'E34', b'C')

(321, 0, 3, b'male', 22. , 0, 0, b'A/5 21172', 7.25 , b'', b'S')
 (322, 0, 3, b'male', 27. , 0, 0, b'349219', 7.8958, b'', b'S')
 (323, 1, 2, b'female', 30. , 0, 0, b'234818', 12.35 , b'', b'Q')
 (324, 1, 2, b'female', 22. , 1, 1, b'248738', 29. , b'', b'S')
 (325, 0, 3, b'male', nan, 8, 2, b'CA. 2343', 69.55 , b'', b'S')
 (326, 1, 1, b'female', 36. , 0, 0, b'PC 17760', 135.6333, b'C32', b'C')
 (327, 0, 3, b'male', 61. , 0, 0, b'345364', 6.2375, b'', b'S')
 (328, 1, 2, b'female', 36. , 0, 0, b'28551', 13. , b'D', b'S')
 (329, 1, 3, b'female', 31. , 1, 1, b'363291', 20.525 , b'', b'S')
 (330, 1, 1, b'female', 16. , 0, 1, b'111361', 57.9792, b'B18', b'C')
 (331, 1, 3, b'female', nan, 2, 0, b'367226', 23.25 , b'', b'Q')
 (332, 0, 1, b'male', 45.5 , 0, 0, b'113043', 28.5 , b'C124', b'S')
 (333, 0, 1, b'male', 38. , 0, 1, b'PC 17582', 153.4625, b'C91', b'S')
 (334, 0, 3, b'male', 16. , 2, 0, b'345764', 18. , b'', b'S')
 (335, 1, 1, b'female', nan, 1, 0, b'PC 17611', 133.65 , b'', b'S')
 (336, 0, 3, b'male', nan, 0, 0, b'349225', 7.8958, b'', b'S')
 (337, 0, 1, b'male', 29. , 1, 0, b'113776', 66.6 , b'C2', b'S')
 (338, 1, 1, b'female', 41. , 0, 0, b'16966', 134.5 , b'E40', b'C')
 (339, 1, 3, b'male', 45. , 0, 0, b'7598', 8.05 , b'', b'S')
 (340, 0, 1, b'male', 45. , 0, 0, b'113784', 35.5 , b'T', b'S')
 (341, 1, 2, b'male', 2. , 1, 1, b'230080', 26. , b'F2', b'S')
 (342, 1, 1, b'female', 24. , 3, 2, b'19950', 263. , b'C23 C25 C27', b'S')
 (343, 0, 2, b'male', 28. , 0, 0, b'248740', 13. , b'', b'S')
 (344, 0, 2, b'male', 25. , 0, 0, b'244361', 13. , b'', b'S')
 (345, 0, 2, b'male', 36. , 0, 0, b'229236', 13. , b'', b'S')
 (346, 1, 2, b'female', 24. , 0, 0, b'248733', 13. , b'F33', b'S')
 (347, 1, 2, b'female', 40. , 0, 0, b'31418', 13. , b'', b'S')
 (348, 1, 3, b'female', nan, 1, 0, b'386525', 16.1 , b'', b'S')
 (349, 1, 3, b'male', 3. , 1, 1, b'C.A. 37671', 15.9 , b'', b'S')
 (350, 0, 3, b'male', 42. , 0, 0, b'315088', 8.6625, b'', b'S')
 (351, 0, 3, b'male', 23. , 0, 0, b'7267', 9.225 , b'', b'S')
 (352, 0, 1, b'male', nan, 0, 0, b'113510', 35. , b'C128', b'S')
 (353, 0, 3, b'male', 15. , 1, 1, b'2695', 7.2292, b'', b'C')
 (354, 0, 3, b'male', 25. , 1, 0, b'349237', 17.8 , b'', b'S')
 (355, 0, 3, b'male', nan, 0, 0, b'2647', 7.225 , b'', b'C')
 (356, 0, 3, b'male', 28. , 0, 0, b'345783', 9.5 , b'', b'S')
 (357, 1, 1, b'female', 22. , 0, 1, b'113505', 55. , b'E33', b'S')
 (358, 0, 2, b'female', 38. , 0, 0, b'237671', 13. , b'', b'S')
 (359, 1, 3, b'female', nan, 0, 0, b'330931', 7.8792, b'', b'Q')
 (360, 1, 3, b'female', nan, 0, 0, b'330980', 7.8792, b'', b'Q')
 (361, 0, 3, b'male', 40. , 1, 4, b'347088', 27.9 , b'', b'S')
 (362, 0, 2, b'male', 29. , 1, 0, b'SC/PARIS 2167', 27.7208, b'', b'C')
 (363, 0, 3, b'female', 45. , 0, 1, b'2691', 14.4542, b'', b'C')
 (364, 0, 3, b'male', 35. , 0, 0, b'SOTON/O.Q. 3101310', 7.05 , b'', b'S')
 (365, 0, 3, b'male', nan, 1, 0, b'370365', 15.5 , b'', b'Q')
 (366, 0, 3, b'male', 30. , 0, 0, b'C 7076', 7.25 , b'', b'S')
 (367, 1, 1, b'female', 60. , 1, 0, b'110813', 75.25 , b'D37', b'C')
 (368, 1, 3, b'female', nan, 0, 0, b'2626', 7.2292, b'', b'C')

(369, 1, 3, b'female', nan, 0, 0, b'14313', 7.75 , b'', b'Q')
 (370, 1, 1, b'female', 24. , 0, 0, b'PC 17477', 69.3 , b'B35', b'C')
 (371, 1, 1, b'male', 25. , 1, 0, b'11765', 55.4417, b'E50', b'C')
 (372, 0, 3, b'male', 18. , 1, 0, b'3101267', 6.4958, b'', b'S')
 (373, 0, 3, b'male', 19. , 0, 0, b'323951', 8.05 , b'', b'S')
 (374, 0, 1, b'male', 22. , 0, 0, b'PC 17760', 135.6333, b'', b'C')
 (375, 0, 3, b'female', 3. , 3, 1, b'349909', 21.075 , b'', b'S')
 (376, 1, 1, b'female', nan, 1, 0, b'PC 17604', 82.1708, b'', b'C')
 (377, 1, 3, b'female', 22. , 0, 0, b'C 7077', 7.25 , b'', b'S')
 (378, 0, 1, b'male', 27. , 0, 2, b'113503', 211.5 , b'C82', b'C')
 (379, 0, 3, b'male', 20. , 0, 0, b'2648', 4.0125, b'', b'C')
 (380, 0, 3, b'male', 19. , 0, 0, b'347069', 7.775 , b'', b'S')
 (381, 1, 1, b'female', 42. , 0, 0, b'PC 17757', 227.525 , b'', b'C')
 (382, 1, 3, b'female', 1. , 0, 2, b'2653', 15.7417, b'', b'C')
 (383, 0, 3, b'male', 32. , 0, 0, b'STON/O 2. 3101293', 7.925 , b'', b'S')
 (384, 1, 1, b'female', 35. , 1, 0, b'113789', 52. , b'', b'S')
 (385, 0, 3, b'male', nan, 0, 0, b'349227', 7.8958, b'', b'S')
 (386, 0, 2, b'male', 18. , 0, 0, b'S.O.C. 14879', 73.5 , b'', b'S')
 (387, 0, 3, b'male', 1. , 5, 2, b'CA 2144', 46.9 , b'', b'S')
 (388, 1, 2, b'female', 36. , 0, 0, b'27849', 13. , b'', b'S')
 (389, 0, 3, b'male', nan, 0, 0, b'367655', 7.7292, b'', b'Q')
 (390, 1, 2, b'female', 17. , 0, 0, b'SC 1748', 12. , b'', b'C')
 (391, 1, 1, b'male', 36. , 1, 2, b'113760', 120. , b'B96 B98', b'S')
 (392, 1, 3, b'male', 21. , 0, 0, b'350034', 7.7958, b'', b'S')
 (393, 0, 3, b'male', 28. , 2, 0, b'3101277', 7.925 , b'', b'S')
 (394, 1, 1, b'female', 23. , 1, 0, b'35273', 113.275 , b'D36', b'C')
 (395, 1, 3, b'female', 24. , 0, 2, b'PP 9549', 16.7 , b'G6', b'S')
 (396, 0, 3, b'male', 22. , 0, 0, b'350052', 7.7958, b'', b'S')
 (397, 0, 3, b'female', 31. , 0, 0, b'350407', 7.8542, b'', b'S')
 (398, 0, 2, b'male', 46. , 0, 0, b'28403', 26. , b'', b'S')
 (399, 0, 2, b'male', 23. , 0, 0, b'244278', 10.5 , b'', b'S')
 (400, 1, 2, b'female', 28. , 0, 0, b'240929', 12.65 , b'', b'S')
 (401, 1, 3, b'male', 39. , 0, 0, b'STON/O 2. 3101289', 7.925 , b'', b'S')
 (402, 0, 3, b'male', 26. , 0, 0, b'341826', 8.05 , b'', b'S')
 (403, 0, 3, b'female', 21. , 1, 0, b'4137', 9.825 , b'', b'S')
 (404, 0, 3, b'male', 28. , 1, 0, b'STON/O2. 3101279', 15.85 , b'', b'S')
 (405, 0, 3, b'female', 20. , 0, 0, b'315096', 8.6625, b'', b'S')
 (406, 0, 2, b'male', 34. , 1, 0, b'28664', 21. , b'', b'S')
 (407, 0, 3, b'male', 51. , 0, 0, b'347064', 7.75 , b'', b'S')
 (408, 1, 2, b'male', 3. , 1, 1, b'29106', 18.75 , b'', b'S')
 (409, 0, 3, b'male', 21. , 0, 0, b'312992', 7.775 , b'', b'S')
 (410, 0, 3, b'female', nan, 3, 1, b'4133', 25.4667, b'', b'S')
 (411, 0, 3, b'male', nan, 0, 0, b'349222', 7.8958, b'', b'S')
 (412, 0, 3, b'male', nan, 0, 0, b'394140', 6.8583, b'', b'Q')
 (413, 1, 1, b'female', 33. , 1, 0, b'19928', 90. , b'C78', b'Q')
 (414, 0, 2, b'male', nan, 0, 0, b'239853', 0. , b'', b'S')
 (415, 1, 3, b'male', 44. , 0, 0, b'STON/O 2. 3101269', 7.925 , b'', b'S')
 (416, 0, 3, b'female', nan, 0, 0, b'343095', 8.05 , b'', b'S')

(417, 1, 2, b'female', 34. , 1, 1, b'28220', 32.5 , b'', b'S')
 (418, 1, 2, b'female', 18. , 0, 2, b'250652', 13. , b'', b'S')
 (419, 0, 2, b'male', 30. , 0, 0, b'28228', 13. , b'', b'S')
 (420, 0, 3, b'female', 10. , 0, 2, b'345773', 24.15 , b'', b'S')
 (421, 0, 3, b'male', nan, 0, 0, b'349254', 7.8958, b'', b'C')
 (422, 0, 3, b'male', 21. , 0, 0, b'A/5. 13032', 7.7333, b'', b'Q')
 (423, 0, 3, b'male', 29. , 0, 0, b'315082', 7.875 , b'', b'S')
 (424, 0, 3, b'female', 28. , 1, 1, b'347080', 14.4 , b'', b'S')
 (425, 0, 3, b'male', 18. , 1, 1, b'370129', 20.2125, b'', b'S')
 (426, 0, 3, b'male', nan, 0, 0, b'A/4. 34244', 7.25 , b'', b'S')
 (427, 1, 2, b'female', 28. , 1, 0, b'2003', 26. , b'', b'S')
 (428, 1, 2, b'female', 19. , 0, 0, b'250655', 26. , b'', b'S')
 (429, 0, 3, b'male', nan, 0, 0, b'364851', 7.75 , b'', b'Q')
 (430, 1, 3, b'male', 32. , 0, 0, b'SOTON/O.Q. 392078', 8.05 , b'E10', b'S')
 (431, 1, 1, b'male', 28. , 0, 0, b'110564', 26.55 , b'C52', b'S')
 (432, 1, 3, b'female', nan, 1, 0, b'376564', 16.1 , b'', b'S')
 (433, 1, 2, b'female', 42. , 1, 0, b'SC/AH 3085', 26. , b'', b'S')
 (434, 0, 3, b'male', 17. , 0, 0, b'STON/O 2. 3101274', 7.125 , b'', b'S')
 (435, 0, 1, b'male', 50. , 1, 0, b'13507', 55.9 , b'E44', b'S')
 (436, 1, 1, b'female', 14. , 1, 2, b'113760', 120. , b'B96 B98', b'S')
 (437, 0, 3, b'female', 21. , 2, 2, b'W./C. 6608', 34.375 , b'', b'S')
 (438, 1, 2, b'female', 24. , 2, 3, b'29106', 18.75 , b'', b'S')
 (439, 0, 1, b'male', 64. , 1, 4, b'19950', 263. , b'C23 C25 C27', b'S')
 (440, 0, 2, b'male', 31. , 0, 0, b'C.A. 18723', 10.5 , b'', b'S')
 (441, 1, 2, b'female', 45. , 1, 1, b'F.C.C. 13529', 26.25 , b'', b'S')
 (442, 0, 3, b'male', 20. , 0, 0, b'345769', 9.5 , b'', b'S')
 (443, 0, 3, b'male', 25. , 1, 0, b'347076', 7.775 , b'', b'S')
 (444, 1, 2, b'female', 28. , 0, 0, b'230434', 13. , b'', b'S')
 (445, 1, 3, b'male', nan, 0, 0, b'65306', 8.1125, b'', b'S')
 (446, 1, 1, b'male', 4. , 0, 2, b'33638', 81.8583, b'A34', b'S')
 (447, 1, 2, b'female', 13. , 0, 1, b'250644', 19.5 , b'', b'S')
 (448, 1, 1, b'male', 34. , 0, 0, b'113794', 26.55 , b'', b'S')
 (449, 1, 3, b'female', 5. , 2, 1, b'2666', 19.2583, b'', b'C')
 (450, 1, 1, b'male', 52. , 0, 0, b'113786', 30.5 , b'C104', b'S')
 (451, 0, 2, b'male', 36. , 1, 2, b'C.A. 34651', 27.75 , b'', b'S')
 (452, 0, 3, b'male', nan, 1, 0, b'65303', 19.9667, b'', b'S')
 (453, 0, 1, b'male', 30. , 0, 0, b'113051', 27.75 , b'C111', b'C')
 (454, 1, 1, b'male', 49. , 1, 0, b'17453', 89.1042, b'C92', b'C')
 (455, 0, 3, b'male', nan, 0, 0, b'A/5 2817', 8.05 , b'', b'S')
 (456, 1, 3, b'male', 29. , 0, 0, b'349240', 7.8958, b'', b'C')
 (457, 0, 1, b'male', 65. , 0, 0, b'13509', 26.55 , b'E38', b'S')
 (458, 1, 1, b'female', nan, 1, 0, b'17464', 51.8625, b'D21', b'S')
 (459, 1, 2, b'female', 50. , 0, 0, b'F.C.C. 13531', 10.5 , b'', b'S')
 (460, 0, 3, b'male', nan, 0, 0, b'371060', 7.75 , b'', b'Q')
 (461, 1, 1, b'male', 48. , 0, 0, b'19952', 26.55 , b'E12', b'S')
 (462, 0, 3, b'male', 34. , 0, 0, b'364506', 8.05 , b'', b'S')
 (463, 0, 1, b'male', 47. , 0, 0, b'111320', 38.5 , b'E63', b'S')
 (464, 0, 2, b'male', 48. , 0, 0, b'234360', 13. , b'', b'S')

(465, 0, 3, b'male', nan, 0, 0, b'A/S 2816', 8.05 , b'', b'S')
 (466, 0, 3, b'male', 38. , 0, 0, b'SOTON/O.Q. 3101306', 7.05 , b'', b'S')
 (467, 0, 2, b'male', nan, 0, 0, b'239853', 0. , b'', b'S')
 (468, 0, 1, b'male', 56. , 0, 0, b'113792', 26.55 , b'', b'S')
 (469, 0, 3, b'male', nan, 0, 0, b'36209', 7.725 , b'', b'Q')
 (470, 1, 3, b'female', 0.75, 2, 1, b'2666', 19.2583, b'', b'C')
 (471, 0, 3, b'male', nan, 0, 0, b'323592', 7.25 , b'', b'S')
 (472, 0, 3, b'male', 38. , 0, 0, b'315089', 8.6625, b'', b'S')
 (473, 1, 2, b'female', 33. , 1, 2, b'C.A. 34651', 27.75 , b'', b'S')
 (474, 1, 2, b'female', 23. , 0, 0, b'SC/AH Basle 541', 13.7917, b'D', b'C')
 (475, 0, 3, b'female', 22. , 0, 0, b'7553', 9.8375, b'', b'S')
 (476, 0, 1, b'male', nan, 0, 0, b'110465', 52. , b'A14', b'S')
 (477, 0, 2, b'male', 34. , 1, 0, b'31027', 21. , b'', b'S')
 (478, 0, 3, b'male', 29. , 1, 0, b'3460', 7.0458, b'', b'S')
 (479, 0, 3, b'male', 22. , 0, 0, b'350060', 7.5208, b'', b'S')
 (480, 1, 3, b'female', 2. , 0, 1, b'3101298', 12.2875, b'', b'S')
 (481, 0, 3, b'male', 9. , 5, 2, b'CA 2144', 46.9 , b'', b'S')
 (482, 0, 2, b'male', nan, 0, 0, b'239854', 0. , b'', b'S')
 (483, 0, 3, b'male', 50. , 0, 0, b'A/5 3594', 8.05 , b'', b'S')
 (484, 1, 3, b'female', 63. , 0, 0, b'4134', 9.5875, b'', b'S')
 (485, 1, 1, b'male', 25. , 1, 0, b'11967', 91.0792, b'B49', b'C')
 (486, 0, 3, b'female', nan, 3, 1, b'4133', 25.4667, b'', b'S')
 (487, 1, 1, b'female', 35. , 1, 0, b'19943', 90. , b'C93', b'S')
 (488, 0, 1, b'male', 58. , 0, 0, b'11771', 29.7 , b'B37', b'C')
 (489, 0, 3, b'male', 30. , 0, 0, b'A.5. 18509', 8.05 , b'', b'S')
 (490, 1, 3, b'male', 9. , 1, 1, b'C.A. 37671', 15.9 , b'', b'S')
 (491, 0, 3, b'male', nan, 1, 0, b'65304', 19.9667, b'', b'S')
 (492, 0, 3, b'male', 21. , 0, 0, b'SOTON/OQ 3101317', 7.25 , b'', b'S')
 (493, 0, 1, b'male', 55. , 0, 0, b'113787', 30.5 , b'C30', b'S')
 (494, 0, 1, b'male', 71. , 0, 0, b'PC 17609', 49.5042, b'', b'C')
 (495, 0, 3, b'male', 21. , 0, 0, b'A/4 45380', 8.05 , b'', b'S')
 (496, 0, 3, b'male', nan, 0, 0, b'2627', 14.4583, b'', b'C')
 (497, 1, 1, b'female', 54. , 1, 0, b'36947', 78.2667, b'D20', b'C')
 (498, 0, 3, b'male', nan, 0, 0, b'C.A. 6212', 15.1 , b'', b'S')
 (499, 0, 1, b'female', 25. , 1, 2, b'113781', 151.55 , b'C22 C26', b'S')
 (500, 0, 3, b'male', 24. , 0, 0, b'350035', 7.7958, b'', b'S')
 (501, 0, 3, b'male', 17. , 0, 0, b'315086', 8.6625, b'', b'S')
 (502, 0, 3, b'female', 21. , 0, 0, b'364846', 7.75 , b'', b'Q')
 (503, 0, 3, b'female', nan, 0, 0, b'330909', 7.6292, b'', b'Q')
 (504, 0, 3, b'female', 37. , 0, 0, b'4135', 9.5875, b'', b'S')
 (505, 1, 1, b'female', 16. , 0, 0, b'110152', 86.5 , b'B79', b'S')
 (506, 0, 1, b'male', 18. , 1, 0, b'PC 17758', 108.9 , b'C65', b'C')
 (507, 1, 2, b'female', 33. , 0, 2, b'26360', 26. , b'', b'S')
 (508, 1, 1, b'male', nan, 0, 0, b'111427', 26.55 , b'', b'S')
 (509, 0, 3, b'male', 28. , 0, 0, b'C 4001', 22.525 , b'', b'S')
 (510, 1, 3, b'male', 26. , 0, 0, b'1601', 56.4958, b'', b'S')
 (511, 1, 3, b'male', 29. , 0, 0, b'382651', 7.75 , b'', b'Q')
 (512, 0, 3, b'male', nan, 0, 0, b'SOTON/OQ 3101316', 8.05 , b'', b'S')

(513, 1, 1, b'male', 36. , 0, 0, b'PC 17473', 26.2875, b'E25', b'S')
 (514, 1, 1, b'female', 54. , 1, 0, b'PC 17603', 59.4 , b'', b'C')
 (515, 0, 3, b'male', 24. , 0, 0, b'349209', 7.4958, b'', b'S')
 (516, 0, 1, b'male', 47. , 0, 0, b'36967', 34.0208, b'D46', b'S')
 (517, 1, 2, b'female', 34. , 0, 0, b'C.A. 34260', 10.5 , b'F33', b'S')
 (518, 0, 3, b'male', nan, 0, 0, b'371110', 24.15 , b'', b'Q')
 (519, 1, 2, b'female', 36. , 1, 0, b'226875', 26. , b'', b'S')
 (520, 0, 3, b'male', 32. , 0, 0, b'349242', 7.8958, b'', b'S')
 (521, 1, 1, b'female', 30. , 0, 0, b'12749', 93.5 , b'B73', b'S')
 (522, 0, 3, b'male', 22. , 0, 0, b'349252', 7.8958, b'', b'S')
 (523, 0, 3, b'male', nan, 0, 0, b'2624', 7.225 , b'', b'C')
 (524, 1, 1, b'female', 44. , 0, 1, b'111361', 57.9792, b'B18', b'C')
 (525, 0, 3, b'male', nan, 0, 0, b'2700', 7.2292, b'', b'C')
 (526, 0, 3, b'male', 40.5 , 0, 0, b'367232', 7.75 , b'', b'Q')
 (527, 1, 2, b'female', 50. , 0, 0, b'W./C. 14258', 10.5 , b'', b'S')
 (528, 0, 1, b'male', nan, 0, 0, b'PC 17483', 221.7792, b'C95', b'S')
 (529, 0, 3, b'male', 39. , 0, 0, b'3101296', 7.925 , b'', b'S')
 (530, 0, 2, b'male', 23. , 2, 1, b'29104', 11.5 , b'', b'S')
 (531, 1, 2, b'female', 2. , 1, 1, b'26360', 26. , b'', b'S')
 (532, 0, 3, b'male', nan, 0, 0, b'2641', 7.2292, b'', b'C')
 (533, 0, 3, b'male', 17. , 1, 1, b'2690', 7.2292, b'', b'C')
 (534, 1, 3, b'female', nan, 0, 2, b'2668', 22.3583, b'', b'C')
 (535, 0, 3, b'female', 30. , 0, 0, b'315084', 8.6625, b'', b'S')
 (536, 1, 2, b'female', 7. , 0, 2, b'F.C.C. 13529', 26.25 , b'', b'S')
 (537, 0, 1, b'male', 45. , 0, 0, b'113050', 26.55 , b'B38', b'S')
 (538, 1, 1, b'female', 30. , 0, 0, b'PC 17761', 106.425 , b'', b'C')
 (539, 0, 3, b'male', nan, 0, 0, b'364498', 14.5 , b'', b'S')
 (540, 1, 1, b'female', 22. , 0, 2, b'13568', 49.5 , b'B39', b'C')
 (541, 1, 1, b'female', 36. , 0, 2, b'WE/P 5735', 71. , b'B22', b'S')
 (542, 0, 3, b'female', 9. , 4, 2, b'347082', 31.275 , b'', b'S')
 (543, 0, 3, b'female', 11. , 4, 2, b'347082', 31.275 , b'', b'S')
 (544, 1, 2, b'male', 32. , 1, 0, b'2908', 26. , b'', b'S')
 (545, 0, 1, b'male', 50. , 1, 0, b'PC 17761', 106.425 , b'C86', b'C')
 (546, 0, 1, b'male', 64. , 0, 0, b'693', 26. , b'', b'S')
 (547, 1, 2, b'female', 19. , 1, 0, b'2908', 26. , b'', b'S')
 (548, 1, 2, b'male', nan, 0, 0, b'SC/PARIS 2146', 13.8625, b'', b'C')
 (549, 0, 3, b'male', 33. , 1, 1, b'363291', 20.525 , b'', b'S')
 (550, 1, 2, b'male', 8. , 1, 1, b'C.A. 33112', 36.75 , b'', b'S')
 (551, 1, 1, b'male', 17. , 0, 2, b'17421', 110.8833, b'C70', b'C')
 (552, 0, 2, b'male', 27. , 0, 0, b'244358', 26. , b'', b'S')
 (553, 0, 3, b'male', nan, 0, 0, b'330979', 7.8292, b'', b'Q')
 (554, 1, 3, b'male', 22. , 0, 0, b'2620', 7.225 , b'', b'C')
 (555, 1, 3, b'female', 22. , 0, 0, b'347085', 7.775 , b'', b'S')
 (556, 0, 1, b'male', 62. , 0, 0, b'113807', 26.55 , b'', b'S')
 (557, 1, 1, b'female', 48. , 1, 0, b'11755', 39.6 , b'A16', b'C')
 (558, 0, 1, b'male', nan, 0, 0, b'PC 17757', 227.525 , b'', b'C')
 (559, 1, 1, b'female', 39. , 1, 1, b'110413', 79.65 , b'E67', b'S')
 (560, 1, 3, b'female', 36. , 1, 0, b'345572', 17.4 , b'', b'S')

(561, 0, 3, b'male', nan, 0, 0, b'372622', 7.75 , b'', b'Q')
 (562, 0, 3, b'male', 40. , 0, 0, b'349251', 7.8958, b'', b'S')
 (563, 0, 2, b'male', 28. , 0, 0, b'218629', 13.5 , b'', b'S')
 (564, 0, 3, b'male', nan, 0, 0, b'SOTON/OQ 392082', 8.05 , b'', b'S')
 (565, 0, 3, b'female', nan, 0, 0, b'SOTON/O.Q. 392087', 8.05 , b'', b'S')
 (566, 0, 3, b'male', 24. , 2, 0, b'A/4 48871', 24.15 , b'', b'S')
 (567, 0, 3, b'male', 19. , 0, 0, b'349205', 7.8958, b'', b'S')
 (568, 0, 3, b'female', 29. , 0, 4, b'349909', 21.075 , b'', b'S')
 (569, 0, 3, b'male', nan, 0, 0, b'2686', 7.2292, b'', b'C')
 (570, 1, 3, b'male', 32. , 0, 0, b'350417', 7.8542, b'', b'S')
 (571, 1, 2, b'male', 62. , 0, 0, b'S.W./PP 752', 10.5 , b'', b'S')
 (572, 1, 1, b'female', 53. , 2, 0, b'11769', 51.4792, b'C101', b'S')
 (573, 1, 1, b'male', 36. , 0, 0, b'PC 17474', 26.3875, b'E25', b'S')
 (574, 1, 3, b'female', nan, 0, 0, b'14312', 7.75 , b'', b'Q')
 (575, 0, 3, b'male', 16. , 0, 0, b'A/4. 20589', 8.05 , b'', b'S')
 (576, 0, 3, b'male', 19. , 0, 0, b'358585', 14.5 , b'', b'S')
 (577, 1, 2, b'female', 34. , 0, 0, b'243880', 13. , b'', b'S')
 (578, 1, 1, b'female', 39. , 1, 0, b'13507', 55.9 , b'E44', b'S')
 (579, 0, 3, b'female', nan, 1, 0, b'2689', 14.4583, b'', b'C')
 (580, 1, 3, b'male', 32. , 0, 0, b'STON/O 2. 3101286', 7.925 , b'', b'S')
 (581, 1, 2, b'female', 25. , 1, 1, b'237789', 30. , b'', b'S')
 (582, 1, 1, b'female', 39. , 1, 1, b'17421', 110.8833, b'C68', b'C')
 (583, 0, 2, b'male', 54. , 0, 0, b'28403', 26. , b'', b'S')
 (584, 0, 1, b'male', 36. , 0, 0, b'13049', 40.125 , b'A10', b'C')
 (585, 0, 3, b'male', nan, 0, 0, b'3411', 8.7125, b'', b'C')
 (586, 1, 1, b'female', 18. , 0, 2, b'110413', 79.65 , b'E68', b'S')
 (587, 0, 2, b'male', 47. , 0, 0, b'237565', 15. , b'', b'S')
 (588, 1, 1, b'male', 60. , 1, 1, b'13567', 79.2 , b'B41', b'C')
 (589, 0, 3, b'male', 22. , 0, 0, b'14973', 8.05 , b'', b'S')
 (590, 0, 3, b'male', nan, 0, 0, b'A./5. 3235', 8.05 , b'', b'S')
 (591, 0, 3, b'male', 35. , 0, 0, b'STON/O 2. 3101273', 7.125 , b'', b'S')
 (592, 1, 1, b'female', 52. , 1, 0, b'36947', 78.2667, b'D20', b'C')
 (593, 0, 3, b'male', 47. , 0, 0, b'A/5 3902', 7.25 , b'', b'S')
 (594, 0, 3, b'female', nan, 0, 2, b'364848', 7.75 , b'', b'Q')
 (595, 0, 2, b'male', 37. , 1, 0, b'SC/AH 29037', 26. , b'', b'S')
 (596, 0, 3, b'male', 36. , 1, 1, b'345773', 24.15 , b'', b'S')
 (597, 1, 2, b'female', nan, 0, 0, b'248727', 33. , b'', b'S')
 (598, 0, 3, b'male', 49. , 0, 0, b'LINE', 0. , b'', b'S')
 (599, 0, 3, b'male', nan, 0, 0, b'2664', 7.225 , b'', b'C')
 (600, 1, 1, b'male', 49. , 1, 0, b'PC 17485', 56.9292, b'A20', b'C')
 (601, 1, 2, b'female', 24. , 2, 1, b'243847', 27. , b'', b'S')
 (602, 0, 3, b'male', nan, 0, 0, b'349214', 7.8958, b'', b'S')
 (603, 0, 1, b'male', nan, 0, 0, b'113796', 42.4 , b'', b'S')
 (604, 0, 3, b'male', 44. , 0, 0, b'364511', 8.05 , b'', b'S')
 (605, 1, 1, b'male', 35. , 0, 0, b'111426', 26.55 , b'', b'C')
 (606, 0, 3, b'male', 36. , 1, 0, b'349910', 15.55 , b'', b'S')
 (607, 0, 3, b'male', 30. , 0, 0, b'349246', 7.8958, b'', b'S')
 (608, 1, 1, b'male', 27. , 0, 0, b'113804', 30.5 , b'', b'S')

(609, 1, 2, b'female', 22. , 1, 2, b'SC/Paris 2123', 41.5792, b'', b'C')
 (610, 1, 1, b'female', 40. , 0, 0, b'PC 17582', 153.4625, b'C125', b'S')
 (611, 0, 3, b'female', 39. , 1, 5, b'347082', 31.275 , b'', b'S')
 (612, 0, 3, b'male', nan, 0, 0, b'SOTON/O.Q. 3101305', 7.05 , b'', b'S')
 (613, 1, 3, b'female', nan, 1, 0, b'367230', 15.5 , b'', b'Q')
 (614, 0, 3, b'male', nan, 0, 0, b'370377', 7.75 , b'', b'Q')
 (615, 0, 3, b'male', 35. , 0, 0, b'364512', 8.05 , b'', b'S')
 (616, 1, 2, b'female', 24. , 1, 2, b'220845', 65. , b'', b'S')
 (617, 0, 3, b'male', 34. , 1, 1, b'347080', 14.4 , b'', b'S')
 (618, 0, 3, b'female', 26. , 1, 0, b'A/5. 3336', 16.1 , b'', b'S')
 (619, 1, 2, b'female', 4. , 2, 1, b'230136', 39. , b'F4', b'S')
 (620, 0, 2, b'male', 26. , 0, 0, b'31028', 10.5 , b'', b'S')
 (621, 0, 3, b'male', 27. , 1, 0, b'2659', 14.4542, b'', b'C')
 (622, 1, 1, b'male', 42. , 1, 0, b'11753', 52.5542, b'D19', b'S')
 (623, 1, 3, b'male', 20. , 1, 1, b'2653', 15.7417, b'', b'C')
 (624, 0, 3, b'male', 21. , 0, 0, b'350029', 7.8542, b'', b'S')
 (625, 0, 3, b'male', 21. , 0, 0, b'54636', 16.1 , b'', b'S')
 (626, 0, 1, b'male', 61. , 0, 0, b'36963', 32.3208, b'D50', b'S')
 (627, 0, 2, b'male', 57. , 0, 0, b'219533', 12.35 , b'', b'Q')
 (628, 1, 1, b'female', 21. , 0, 0, b'13502', 77.9583, b'D9', b'S')
 (629, 0, 3, b'male', 26. , 0, 0, b'349224', 7.8958, b'', b'S')
 (630, 0, 3, b'male', nan, 0, 0, b'334912', 7.7333, b'', b'Q')
 (631, 1, 1, b'male', 80. , 0, 0, b'27042', 30. , b'A23', b'S')
 (632, 0, 3, b'male', 51. , 0, 0, b'347743', 7.0542, b'', b'S')
 (633, 1, 1, b'male', 32. , 0, 0, b'13214', 30.5 , b'B50', b'C')
 (634, 0, 1, b'male', nan, 0, 0, b'112052', 0. , b'', b'S')
 (635, 0, 3, b'female', 9. , 3, 2, b'347088', 27.9 , b'', b'S')
 (636, 1, 2, b'female', 28. , 0, 0, b'237668', 13. , b'', b'S')
 (637, 0, 3, b'male', 32. , 0, 0, b'STON/O 2. 3101292', 7.925 , b'', b'S')
 (638, 0, 2, b'male', 31. , 1, 1, b'C.A. 31921', 26.25 , b'', b'S')
 (639, 0, 3, b'female', 41. , 0, 5, b'3101295', 39.6875, b'', b'S')
 (640, 0, 3, b'male', nan, 1, 0, b'376564', 16.1 , b'', b'S')
 (641, 0, 3, b'male', 20. , 0, 0, b'350050', 7.8542, b'', b'S')
 (642, 1, 1, b'female', 24. , 0, 0, b'PC 17477', 69.3 , b'B35', b'C')
 (643, 0, 3, b'female', 2. , 3, 2, b'347088', 27.9 , b'', b'S')
 (644, 1, 3, b'male', nan, 0, 0, b'1601', 56.4958, b'', b'S')
 (645, 1, 3, b'female', 0.75, 2, 1, b'2666', 19.2583, b'', b'C')
 (646, 1, 1, b'male', 48. , 1, 0, b'PC 17572', 76.7292, b'D33', b'C')
 (647, 0, 3, b'male', 19. , 0, 0, b'349231', 7.8958, b'', b'S')
 (648, 1, 1, b'male', 56. , 0, 0, b'13213', 35.5 , b'A26', b'C')
 (649, 0, 3, b'male', nan, 0, 0, b'S.O./P.P. 751', 7.55 , b'', b'S')
 (650, 1, 3, b'female', 23. , 0, 0, b'CA. 2314', 7.55 , b'', b'S')
 (651, 0, 3, b'male', nan, 0, 0, b'349221', 7.8958, b'', b'S')
 (652, 1, 2, b'female', 18. , 0, 1, b'231919', 23. , b'', b'S')
 (653, 0, 3, b'male', 21. , 0, 0, b'8475', 8.4333, b'', b'S')
 (654, 1, 3, b'female', nan, 0, 0, b'330919', 7.8292, b'', b'Q')
 (655, 0, 3, b'female', 18. , 0, 0, b'365226', 6.75 , b'', b'Q')
 (656, 0, 2, b'male', 24. , 2, 0, b'S.O.C. 14879', 73.5 , b'', b'S')

(657, 0, 3, b'male', nan, 0, 0, b'349223', 7.8958, b'', b'S')
 (658, 0, 3, b'female', 32. , 1, 1, b'364849', 15.5 , b'', b'Q')
 (659, 0, 2, b'male', 23. , 0, 0, b'29751', 13. , b'', b'S')
 (660, 0, 1, b'male', 58. , 0, 2, b'35273', 113.275 , b'D48', b'C')
 (661, 1, 1, b'male', 50. , 2, 0, b'PC 17611', 133.65 , b'', b'S')
 (662, 0, 3, b'male', 40. , 0, 0, b'2623', 7.225 , b'', b'C')
 (663, 0, 1, b'male', 47. , 0, 0, b'5727', 25.5875, b'E58', b'S')
 (664, 0, 3, b'male', 36. , 0, 0, b'349210', 7.4958, b'', b'S')
 (665, 1, 3, b'male', 20. , 1, 0, b'STON/0 2. 3101285', 7.925 , b'', b'S')
 (666, 0, 2, b'male', 32. , 2, 0, b'S.O.C. 14879', 73.5 , b'', b'S')
 (667, 0, 2, b'male', 25. , 0, 0, b'234686', 13. , b'', b'S')
 (668, 0, 3, b'male', nan, 0, 0, b'312993', 7.775 , b'', b'S')
 (669, 0, 3, b'male', 43. , 0, 0, b'A/5 3536', 8.05 , b'', b'S')
 (670, 1, 1, b'female', nan, 1, 0, b'19996', 52. , b'C126', b'S')
 (671, 1, 2, b'female', 40. , 1, 1, b'29750', 39. , b'', b'S')
 (672, 0, 1, b'male', 31. , 1, 0, b'F.C. 12750', 52. , b'B71', b'S')
 (673, 0, 2, b'male', 70. , 0, 0, b'C.A. 24580', 10.5 , b'', b'S')
 (674, 1, 2, b'male', 31. , 0, 0, b'244270', 13. , b'', b'S')
 (675, 0, 2, b'male', nan, 0, 0, b'239856', 0. , b'', b'S')
 (676, 0, 3, b'male', 18. , 0, 0, b'349912', 7.775 , b'', b'S')
 (677, 0, 3, b'male', 24.5 , 0, 0, b'342826', 8.05 , b'', b'S')
 (678, 1, 3, b'female', 18. , 0, 0, b'4138', 9.8417, b'', b'S')
 (679, 0, 3, b'female', 43. , 1, 6, b'CA 2144', 46.9 , b'', b'S')
 (680, 1, 1, b'male', 36. , 0, 1, b'PC 17755', 512.3292, b'B51 B53 B55', b'C')
 (681, 0, 3, b'female', nan, 0, 0, b'330935', 8.1375, b'', b'Q')
 (682, 1, 1, b'male', 27. , 0, 0, b'PC 17572', 76.7292, b'D49', b'C')
 (683, 0, 3, b'male', 20. , 0, 0, b'6563', 9.225 , b'', b'S')
 (684, 0, 3, b'male', 14. , 5, 2, b'CA 2144', 46.9 , b'', b'S')
 (685, 0, 2, b'male', 60. , 1, 1, b'29750', 39. , b'', b'S')
 (686, 0, 2, b'male', 25. , 1, 2, b'SC/Paris 2123', 41.5792, b'', b'C')
 (687, 0, 3, b'male', 14. , 4, 1, b'3101295', 39.6875, b'', b'S')
 (688, 0, 3, b'male', 19. , 0, 0, b'349228', 10.1708, b'', b'S')
 (689, 0, 3, b'male', 18. , 0, 0, b'350036', 7.7958, b'', b'S')
 (690, 1, 1, b'female', 15. , 0, 1, b'24160', 211.3375, b'B5', b'S')
 (691, 1, 1, b'male', 31. , 1, 0, b'17474', 57. , b'B20', b'S')
 (692, 1, 3, b'female', 4. , 0, 1, b'349256', 13.4167, b'', b'C')
 (693, 1, 3, b'male', nan, 0, 0, b'1601', 56.4958, b'', b'S')
 (694, 0, 3, b'male', 25. , 0, 0, b'2672', 7.225 , b'', b'C')
 (695, 0, 1, b'male', 60. , 0, 0, b'113800', 26.55 , b'', b'S')
 (696, 0, 2, b'male', 52. , 0, 0, b'248731', 13.5 , b'', b'S')
 (697, 0, 3, b'male', 44. , 0, 0, b'363592', 8.05 , b'', b'S')
 (698, 1, 3, b'female', nan, 0, 0, b'35852', 7.7333, b'', b'Q')
 (699, 0, 1, b'male', 49. , 1, 1, b'17421', 110.8833, b'C68', b'C')
 (700, 0, 3, b'male', 42. , 0, 0, b'348121', 7.65 , b'F G63', b'S')
 (701, 1, 1, b'female', 18. , 1, 0, b'PC 17757', 227.525 , b'C62 C64', b'C')
 (702, 1, 1, b'male', 35. , 0, 0, b'PC 17475', 26.2875, b'E24', b'S')
 (703, 0, 3, b'female', 18. , 0, 1, b'2691', 14.4542, b'', b'C')
 (704, 0, 3, b'male', 25. , 0, 0, b'36864', 7.7417, b'', b'Q')

(705, 0, 3, b'male', 26. , 1, 0, b'350025', 7.8542, b'', b'S')
 (706, 0, 2, b'male', 39. , 0, 0, b'250655', 26. , b'', b'S')
 (707, 1, 2, b'female', 45. , 0, 0, b'223596', 13.5 , b'', b'S')
 (708, 1, 1, b'male', 42. , 0, 0, b'PC 17476', 26.2875, b'E24', b'S')
 (709, 1, 1, b'female', 22. , 0, 0, b'113781', 151.55 , b'', b'S')
 (710, 1, 3, b'male', nan, 1, 1, b'2661', 15.2458, b'', b'C')
 (711, 1, 1, b'female', 24. , 0, 0, b'PC 17482', 49.5042, b'C90', b'C')
 (712, 0, 1, b'male', nan, 0, 0, b'113028', 26.55 , b'C124', b'S')
 (713, 1, 1, b'male', 48. , 1, 0, b'19996', 52. , b'C126', b'S')
 (714, 0, 3, b'male', 29. , 0, 0, b'7545', 9.4833, b'', b'S')
 (715, 0, 2, b'male', 52. , 0, 0, b'250647', 13. , b'', b'S')
 (716, 0, 3, b'male', 19. , 0, 0, b'348124', 7.65 , b'F G73', b'S')
 (717, 1, 1, b'female', 38. , 0, 0, b'PC 17757', 227.525 , b'C45', b'C')
 (718, 1, 2, b'female', 27. , 0, 0, b'34218', 10.5 , b'E101', b'S')
 (719, 0, 3, b'male', nan, 0, 0, b'36568', 15.5 , b'', b'Q')
 (720, 0, 3, b'male', 33. , 0, 0, b'347062', 7.775 , b'', b'S')
 (721, 1, 2, b'female', 6. , 0, 1, b'248727', 33. , b'', b'S')
 (722, 0, 3, b'male', 17. , 1, 0, b'350048', 7.0542, b'', b'S')
 (723, 0, 2, b'male', 34. , 0, 0, b'12233', 13. , b'', b'S')
 (724, 0, 2, b'male', 50. , 0, 0, b'250643', 13. , b'', b'S')
 (725, 1, 1, b'male', 27. , 1, 0, b'113806', 53.1 , b'E8', b'S')
 (726, 0, 3, b'male', 20. , 0, 0, b'315094', 8.6625, b'', b'S')
 (727, 1, 2, b'female', 30. , 3, 0, b'31027', 21. , b'', b'S')
 (728, 1, 3, b'female', nan, 0, 0, b'36866', 7.7375, b'', b'Q')
 (729, 0, 2, b'male', 25. , 1, 0, b'236853', 26. , b'', b'S')
 (730, 0, 3, b'female', 25. , 1, 0, b'STON/02. 3101271', 7.925 , b'', b'S')
 (731, 1, 1, b'female', 29. , 0, 0, b'24160', 211.3375, b'B5', b'S')
 (732, 0, 3, b'male', 11. , 0, 0, b'2699', 18.7875, b'', b'C')
 (733, 0, 2, b'male', nan, 0, 0, b'239855', 0. , b'', b'S')
 (734, 0, 2, b'male', 23. , 0, 0, b'28425', 13. , b'', b'S')
 (735, 0, 2, b'male', 23. , 0, 0, b'233639', 13. , b'', b'S')
 (736, 0, 3, b'male', 28.5 , 0, 0, b'54636', 16.1 , b'', b'S')
 (737, 0, 3, b'female', 48. , 1, 3, b'W./C. 6608', 34.375 , b'', b'S')
 (738, 1, 1, b'male', 35. , 0, 0, b'PC 17755', 512.3292, b'B101', b'C')
 (739, 0, 3, b'male', nan, 0, 0, b'349201', 7.8958, b'', b'S')
 (740, 0, 3, b'male', nan, 0, 0, b'349218', 7.8958, b'', b'S')
 (741, 1, 1, b'male', nan, 0, 0, b'16988', 30. , b'D45', b'S')
 (742, 0, 1, b'male', 36. , 1, 0, b'19877', 78.85 , b'C46', b'S')
 (743, 1, 1, b'female', 21. , 2, 2, b'PC 17608', 262.375 , b'B57 B59 B63 B66',
 b'C')
 (744, 0, 3, b'male', 24. , 1, 0, b'376566', 16.1 , b'', b'S')
 (745, 1, 3, b'male', 31. , 0, 0, b'STON/0 2. 3101288', 7.925 , b'', b'S')
 (746, 0, 1, b'male', 70. , 1, 1, b'WE/P 5735', 71. , b'B22', b'S')
 (747, 0, 3, b'male', 16. , 1, 1, b'C.A. 2673', 20.25 , b'', b'S')
 (748, 1, 2, b'female', 30. , 0, 0, b'250648', 13. , b'', b'S')
 (749, 0, 1, b'male', 19. , 1, 0, b'113773', 53.1 , b'D30', b'S')
 (750, 0, 3, b'male', 31. , 0, 0, b'335097', 7.75 , b'', b'Q')
 (751, 1, 2, b'female', 4. , 1, 1, b'29103', 23. , b'', b'S')

(752, 1, 3, b'male', 6. , 0, 1, b'392096', 12.475 , b'E121', b'S')
 (753, 0, 3, b'male', 33. , 0, 0, b'345780', 9.5 , b'', b'S')
 (754, 0, 3, b'male', 23. , 0, 0, b'349204', 7.8958, b'', b'S')
 (755, 1, 2, b'female', 48. , 1, 2, b'220845', 65. , b'', b'S')
 (756, 1, 2, b'male', 0.67, 1, 1, b'250649', 14.5 , b'', b'S')
 (757, 0, 3, b'male', 28. , 0, 0, b'350042', 7.7958, b'', b'S')
 (758, 0, 2, b'male', 18. , 0, 0, b'29108', 11.5 , b'', b'S')
 (759, 0, 3, b'male', 34. , 0, 0, b'363294', 8.05 , b'', b'S')
 (760, 1, 1, b'female', 33. , 0, 0, b'110152', 86.5 , b'B77', b'S')
 (761, 0, 3, b'male', nan, 0, 0, b'358585', 14.5 , b'', b'S')
 (762, 0, 3, b'male', 41. , 0, 0, b'SOTON/02 3101272', 7.125 , b'', b'S')
 (763, 1, 3, b'male', 20. , 0, 0, b'2663', 7.2292, b'', b'C')
 (764, 1, 1, b'female', 36. , 1, 2, b'113760', 120. , b'B96 B98', b'S')
 (765, 0, 3, b'male', 16. , 0, 0, b'347074', 7.775 , b'', b'S')
 (766, 1, 1, b'female', 51. , 1, 0, b'13502', 77.9583, b'D11', b'S')
 (767, 0, 1, b'male', nan, 0, 0, b'112379', 39.6 , b'', b'C')
 (768, 0, 3, b'female', 30.5 , 0, 0, b'364850', 7.75 , b'', b'Q')
 (769, 0, 3, b'male', nan, 1, 0, b'371110', 24.15 , b'', b'Q')
 (770, 0, 3, b'male', 32. , 0, 0, b'8471', 8.3625, b'', b'S')
 (771, 0, 3, b'male', 24. , 0, 0, b'345781', 9.5 , b'', b'S')
 (772, 0, 3, b'male', 48. , 0, 0, b'350047', 7.8542, b'', b'S')
 (773, 0, 2, b'female', 57. , 0, 0, b'S.O./P.P. 3', 10.5 , b'E77', b'S')
 (774, 0, 3, b'male', nan, 0, 0, b'2674', 7.225 , b'', b'C')
 (775, 1, 2, b'female', 54. , 1, 3, b'29105', 23. , b'', b'S')
 (776, 0, 3, b'male', 18. , 0, 0, b'347078', 7.75 , b'', b'S')
 (777, 0, 3, b'male', nan, 0, 0, b'383121', 7.75 , b'F38', b'Q')
 (778, 1, 3, b'female', 5. , 0, 0, b'364516', 12.475 , b'', b'S')
 (779, 0, 3, b'male', nan, 0, 0, b'36865', 7.7375, b'', b'Q')
 (780, 1, 1, b'female', 43. , 0, 1, b'24160', 211.3375, b'B3', b'S')
 (781, 1, 3, b'female', 13. , 0, 0, b'2687', 7.2292, b'', b'C')
 (782, 1, 1, b'female', 17. , 1, 0, b'17474', 57. , b'B20', b'S')
 (783, 0, 1, b'male', 29. , 0, 0, b'113501', 30. , b'D6', b'S')
 (784, 0, 3, b'male', nan, 1, 2, b'W./C. 6607', 23.45 , b'', b'S')
 (785, 0, 3, b'male', 25. , 0, 0, b'SOTON/O.Q. 3101312', 7.05 , b'', b'S')
 (786, 0, 3, b'male', 25. , 0, 0, b'374887', 7.25 , b'', b'S')
 (787, 1, 3, b'female', 18. , 0, 0, b'3101265', 7.4958, b'', b'S')
 (788, 0, 3, b'male', 8. , 4, 1, b'382652', 29.125 , b'', b'Q')
 (789, 1, 3, b'male', 1. , 1, 2, b'C.A. 2315', 20.575 , b'', b'S')
 (790, 0, 1, b'male', 46. , 0, 0, b'PC 17593', 79.2 , b'B82 B84', b'C')
 (791, 0, 3, b'male', nan, 0, 0, b'12460', 7.75 , b'', b'Q')
 (792, 0, 2, b'male', 16. , 0, 0, b'239865', 26. , b'', b'S')
 (793, 0, 3, b'female', nan, 8, 2, b'CA. 2343', 69.55 , b'', b'S')
 (794, 0, 1, b'male', nan, 0, 0, b'PC 17600', 30.6958, b'', b'C')
 (795, 0, 3, b'male', 25. , 0, 0, b'349203', 7.8958, b'', b'S')
 (796, 0, 2, b'male', 39. , 0, 0, b'28213', 13. , b'', b'S')
 (797, 1, 1, b'female', 49. , 0, 0, b'17465', 25.9292, b'D17', b'S')
 (798, 1, 3, b'female', 31. , 0, 0, b'349244', 8.6833, b'', b'S')
 (799, 0, 3, b'male', 30. , 0, 0, b'2685', 7.2292, b'', b'C')

(800, 0, 3, b'female', 30. , 1, 1, b'345773', 24.15 , b'', b'S')
 (801, 0, 2, b'male', 34. , 0, 0, b'250647', 13. , b'', b'S')
 (802, 1, 2, b'female', 31. , 1, 1, b'C.A. 31921', 26.25 , b'', b'S')
 (803, 1, 1, b'male', 11. , 1, 2, b'113760', 120. , b'B96 B98', b'S')
 (804, 1, 3, b'male', 0.42, 0, 1, b'2625', 8.5167, b'', b'C')
 (805, 1, 3, b'male', 27. , 0, 0, b'347089', 6.975 , b'', b'S')
 (806, 0, 3, b'male', 31. , 0, 0, b'347063', 7.775 , b'', b'S')
 (807, 0, 1, b'male', 39. , 0, 0, b'112050', 0. , b'A36', b'S')
 (808, 0, 3, b'female', 18. , 0, 0, b'347087', 7.775 , b'', b'S')
 (809, 0, 2, b'male', 39. , 0, 0, b'248723', 13. , b'', b'S')
 (810, 1, 1, b'female', 33. , 1, 0, b'113806', 53.1 , b'E8', b'S')
 (811, 0, 3, b'male', 26. , 0, 0, b'3474', 7.8875, b'', b'S')
 (812, 0, 3, b'male', 39. , 0, 0, b'A/4 48871', 24.15 , b'', b'S')
 (813, 0, 2, b'male', 35. , 0, 0, b'28206', 10.5 , b'', b'S')
 (814, 0, 3, b'female', 6. , 4, 2, b'347082', 31.275 , b'', b'S')
 (815, 0, 3, b'male', 30.5 , 0, 0, b'364499', 8.05 , b'', b'S')
 (816, 0, 1, b'male', nan, 0, 0, b'112058', 0. , b'B102', b'S')
 (817, 0, 3, b'female', 23. , 0, 0, b'STON/02. 3101290', 7.925 , b'', b'S')
 (818, 0, 2, b'male', 31. , 1, 1, b'S.C./PARIS 2079', 37.0042, b'', b'C')
 (819, 0, 3, b'male', 43. , 0, 0, b'C 7075', 6.45 , b'', b'S')
 (820, 0, 3, b'male', 10. , 3, 2, b'347088', 27.9 , b'', b'S')
 (821, 1, 1, b'female', 52. , 1, 1, b'12749', 93.5 , b'B69', b'S')
 (822, 1, 3, b'male', 27. , 0, 0, b'315098', 8.6625, b'', b'S')
 (823, 0, 1, b'male', 38. , 0, 0, b'19972', 0. , b'', b'S')
 (824, 1, 3, b'female', 27. , 0, 1, b'392096', 12.475 , b'E121', b'S')
 (825, 0, 3, b'male', 2. , 4, 1, b'3101295', 39.6875, b'', b'S')
 (826, 0, 3, b'male', nan, 0, 0, b'368323', 6.95 , b'', b'Q')
 (827, 0, 3, b'male', nan, 0, 0, b'1601', 56.4958, b'', b'S')
 (828, 1, 2, b'male', 1. , 0, 2, b'S.C./PARIS 2079', 37.0042, b'', b'C')
 (829, 1, 3, b'male', nan, 0, 0, b'367228', 7.75 , b'', b'Q')
 (830, 1, 1, b'female', 62. , 0, 0, b'113572', 80. , b'B28', b'')
 (831, 1, 3, b'female', 15. , 1, 0, b'2659', 14.4542, b'', b'C')
 (832, 1, 2, b'male', 0.83, 1, 1, b'29106', 18.75 , b'', b'S')
 (833, 0, 3, b'male', nan, 0, 0, b'2671', 7.2292, b'', b'C')
 (834, 0, 3, b'male', 23. , 0, 0, b'347468', 7.8542, b'', b'S')
 (835, 0, 3, b'male', 18. , 0, 0, b'2223', 8.3 , b'', b'S')
 (836, 1, 1, b'female', 39. , 1, 1, b'PC 17756', 83.1583, b'E49', b'C')
 (837, 0, 3, b'male', 21. , 0, 0, b'315097', 8.6625, b'', b'S')
 (838, 0, 3, b'male', nan, 0, 0, b'392092', 8.05 , b'', b'S')
 (839, 1, 3, b'male', 32. , 0, 0, b'1601', 56.4958, b'', b'S')
 (840, 1, 1, b'male', nan, 0, 0, b'11774', 29.7 , b'C47', b'C')
 (841, 0, 3, b'male', 20. , 0, 0, b'SOTON/02 3101287', 7.925 , b'', b'S')
 (842, 0, 2, b'male', 16. , 0, 0, b'S.O./P.P. 3', 10.5 , b'', b'S')
 (843, 1, 1, b'female', 30. , 0, 0, b'113798', 31. , b'', b'C')
 (844, 0, 3, b'male', 34.5 , 0, 0, b'2683', 6.4375, b'', b'C')
 (845, 0, 3, b'male', 17. , 0, 0, b'315090', 8.6625, b'', b'S')
 (846, 0, 3, b'male', 42. , 0, 0, b'C.A. 5547', 7.55 , b'', b'S')
 (847, 0, 3, b'male', nan, 8, 2, b'CA. 2343', 69.55 , b'', b'S')

(848, 0, 3, b'male', 35. , 0, 0, b'349213', 7.8958, b'', b'C')
 (849, 0, 2, b'male', 28. , 0, 1, b'248727', 33. , b'', b'S')
 (850, 1, 1, b'female', nan, 1, 0, b'17453', 89.1042, b'C92', b'C')
 (851, 0, 3, b'male', 4. , 4, 2, b'347082', 31.275 , b'', b'S')
 (852, 0, 3, b'male', 74. , 0, 0, b'347060', 7.775 , b'', b'S')
 (853, 0, 3, b'female', 9. , 1, 1, b'2678', 15.2458, b'', b'C')
 (854, 1, 1, b'female', 16. , 0, 1, b'PC 17592', 39.4 , b'D28', b'S')
 (855, 0, 2, b'female', 44. , 1, 0, b'244252', 26. , b'', b'S')
 (856, 1, 3, b'female', 18. , 0, 1, b'392091', 9.35 , b'', b'S')
 (857, 1, 1, b'female', 45. , 1, 1, b'36928', 164.8667, b'', b'S')
 (858, 1, 1, b'male', 51. , 0, 0, b'113055', 26.55 , b'E17', b'S')
 (859, 1, 3, b'female', 24. , 0, 3, b'2666', 19.2583, b'', b'C')
 (860, 0, 3, b'male', nan, 0, 0, b'2629', 7.2292, b'', b'C')
 (861, 0, 3, b'male', 41. , 2, 0, b'350026', 14.1083, b'', b'S')
 (862, 0, 2, b'male', 21. , 1, 0, b'28134', 11.5 , b'', b'S')
 (863, 1, 1, b'female', 48. , 0, 0, b'17466', 25.9292, b'D17', b'S')
 (864, 0, 3, b'female', nan, 8, 2, b'CA. 2343', 69.55 , b'', b'S')
 (865, 0, 2, b'male', 24. , 0, 0, b'233866', 13. , b'', b'S')
 (866, 1, 2, b'female', 42. , 0, 0, b'236852', 13. , b'', b'S')
 (867, 1, 2, b'female', 27. , 1, 0, b'SC/PARIS 2149', 13.8583, b'', b'C')
 (868, 0, 1, b'male', 31. , 0, 0, b'PC 17590', 50.4958, b'A24', b'S')
 (869, 0, 3, b'male', nan, 0, 0, b'345777', 9.5 , b'', b'S')
 (870, 1, 3, b'male', 4. , 1, 1, b'347742', 11.1333, b'', b'S')
 (871, 0, 3, b'male', 26. , 0, 0, b'349248', 7.8958, b'', b'S')
 (872, 1, 1, b'female', 47. , 1, 1, b'11751', 52.5542, b'D35', b'S')
 (873, 0, 1, b'male', 33. , 0, 0, b'695', 5. , b'B51 B53 B55', b'S')
 (874, 0, 3, b'male', 47. , 0, 0, b'345765', 9. , b'', b'S')
 (875, 1, 2, b'female', 28. , 1, 0, b'P/PP 3381', 24. , b'', b'C')
 (876, 1, 3, b'female', 15. , 0, 0, b'2667', 7.225 , b'', b'C')
 (877, 0, 3, b'male', 20. , 0, 0, b'7534', 9.8458, b'', b'S')
 (878, 0, 3, b'male', 19. , 0, 0, b'349212', 7.8958, b'', b'S')
 (879, 0, 3, b'male', nan, 0, 0, b'349217', 7.8958, b'', b'S')
 (880, 1, 1, b'female', 56. , 0, 1, b'11767', 83.1583, b'C50', b'C')
 (881, 1, 2, b'female', 25. , 0, 1, b'230433', 26. , b'', b'S')
 (882, 0, 3, b'male', 33. , 0, 0, b'349257', 7.8958, b'', b'S')
 (883, 0, 3, b'female', 22. , 0, 0, b'7552', 10.5167, b'', b'S')
 (884, 0, 2, b'male', 28. , 0, 0, b'C.A./SOTON 34068', 10.5 , b'', b'S')
 (885, 0, 3, b'male', 25. , 0, 0, b'SOTON/OQ 392076', 7.05 , b'', b'S')
 (886, 0, 3, b'female', 39. , 0, 5, b'382652', 29.125 , b'', b'Q')
 (887, 0, 2, b'male', 27. , 0, 0, b'211536', 13. , b'', b'S')
 (888, 1, 1, b'female', 19. , 0, 0, b'112053', 30. , b'B42', b'S')
 (889, 0, 3, b'female', nan, 1, 2, b'W./C. 6607', 23.45 , b'', b'S')
 (890, 1, 1, b'male', 26. , 0, 0, b'111369', 30. , b'C148', b'C')
 (891, 0, 3, b'male', 32. , 0, 0, b'370376', 7.75 , b'', b'Q')]

1.1.7 1.6 Working with mixed datatypes (2)

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.

Instructions

- Import `titanic.csv` using the function `np.recfromcsv()` and assign it to the variable, `d`. You'll only need to pass `file` to it because it has the defaults `delimiter=','` and `names=True` in addition to `dtype=None`!
- Run the remaining code to print the first three entries of the resulting array `d`.

[]:

```
[24]: # Assign the filename: file
file = './Data/titanic.csv'

# Import file using np.recfromcsv: d
d = np.recfromcsv(file, encoding = None)

# Print out first three entries of d
print(d[:3])
```

```
[(1, 0, 3, 'male', 22., 1, 0, 'A/5 21171', 7.25, '', 'S')
 (2, 1, 1, 'female', 38., 1, 0, 'PC 17599', 71.2833, 'C85', 'C')
 (3, 1, 3, 'female', 26., 0, 0, 'STON/O2. 3101282', 7.925, '', 'S')]
```

1.1.8 1.7 Using pandas to import flat files as DataFrames (1)

In the last exercise, you were able to import flat files containing columns with different datatypes as numpy arrays. However, the `DataFrame` object in pandas is a more appropriate structure in which to store such data and, thankfully, we can easily import files of mixed data types as `DataFrames` using the pandas functions `read_csv()` and `read_table()`.

Instructions

- Import the `pandas` package using the alias `pd`.
- Read `titanic.csv` into a `DataFrame` called `df`. The file name is already stored in the `file` object.
- In a `print()` call, view the head of the `DataFrame`.

```
[33]: # Import pandas as pd
import pandas as pd

# Assign the filename: file
file = './Data/titanic.csv'

# Read the file into a DataFrame: df
df = pd.read_csv(file)
```

```
# View the head of the DataFrame
print(df.head())
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	\
0	1	0	3	male	22.0	1	0	
1	2	1	1	female	38.0	1	0	
2	3	1	3	female	26.0	0	0	
3	4	1	1	female	35.0	1	0	
4	5	0	3	male	35.0	0	0	

	Ticket	Fare	Cabin	Embarked
0	A/5 21171	7.2500	NaN	S
1	PC 17599	71.2833	C85	C
2	STON/O2. 3101282	7.9250	NaN	S
3	113803	53.1000	C123	S
4	373450	8.0500	NaN	S

1.1.9 1.8 Using pandas to import flat files as DataFrames (2)

In the last exercise, you were able to import flat files into a `pandas` `DataFrame`. As a bonus, it is then straightforward to retrieve the corresponding `numpy` array using the attribute `values`. You'll now have a chance to do this using the MNIST dataset, which is available as `digits.csv`.

Instructions

- Import the first 5 rows of the file into a `DataFrame` using the function `pd.read_csv()` and assign the result to `data`. You'll need to use the arguments `nrows` and `header` (there is no header in this file).
- Build a `numpy` array from the resulting `DataFrame` in `data` and assign to `data_array`.
- Execute `print(type(data_array))` to print the datatype of `data_array`.

```
[44]: # Assign the filename: file
file = './Data/digits.csv'

# Read the first 5 rows of the file into a DataFrame: data
data = pd.read_csv(file, nrows = 5, header=None)

# 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))
```

```
<class 'numpy.ndarray'>
```

1.1.10 1.9 Customizing your pandas import

The `pandas` package is also great at dealing with many of the issues you will encounter when importing data as a data scientist, such as comments occurring in flat files, empty lines and missing

values. Note that missing values are also commonly referred to as NA or NaN. To wrap up this chapter, you're now going to import a slightly corrupted copy of the Titanic dataset `titanic_corrupt.txt`, which

- contains comments after the character '#'
- is tab-delimited.

Instructions

- Complete the `sep` (the pandas version of `delim`), `comment` and `na_values` arguments of `pd.read_csv()`. `comment` takes characters that comments occur after in the file, which in this case is '#'. `na_values` takes a list of strings to recognize as NA/NaN, in this case the string 'Nothing'.
- Execute the rest of the code to print the head of the resulting DataFrame and plot the histogram of the 'Age' of passengers aboard the Titanic.

```
[45]: # Import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

# Assign filename: file
file = './Data/titanic_corrupt.txt'

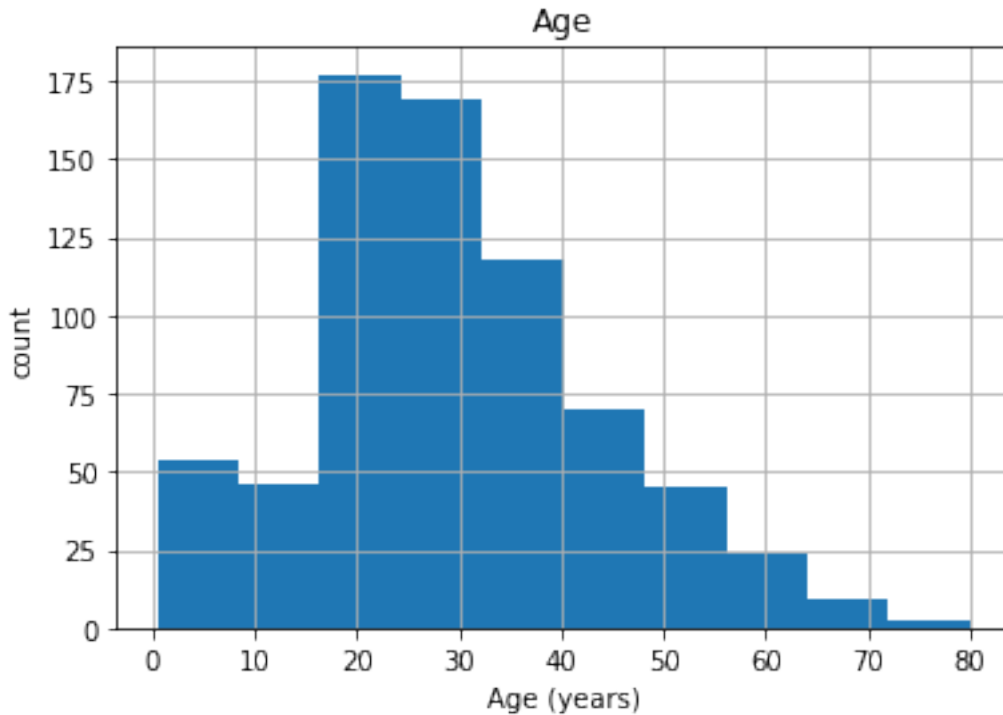
# Import file: data
data = pd.read_csv(file, sep='\t', comment='#', na_values='Nothing')

# Print the head of the DataFrame
print(data.head())

# Plot 'Age' variable in a histogram
pd.DataFrame.hist(data[['Age']])
plt.xlabel('Age (years)')
plt.ylabel('count')
plt.show()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	\
0	1	0	3	male	22.0	1	0	
1	2	1	1	female	38.0	1	0	
2	3	1	3	female	26.0	0	0	
3	4	1	1	female	35.0	1	0	
4	5	0	3	male	35.0	0	0	

	Ticket	Fare	Cabin	Embarked
0	A/5 21171	7.2500	NaN	S #dfafdad
1	PC 17599#to	71.2833	C85	C
2	STON/O2. 3101282	7.9250	NaN	S
3	113803	53.1000	C123	S
4	373450	8.0500	NaN	S



1.2 2. Importing data from other file types

You’ve learned how to import flat files, but there are many other file types you will potentially have to work with as a data scientist. In this chapter, you’ll learn how to import data into Python from a wide array of important file types. You will be importing file types such as pickled files, Excel spreadsheets, SAS and Stata files, HDF5 files, a file type for storing large quantities of numerical data, and MATLAB files.

1.2.1 2.1 Not so flat any more

In Chapter 1, you learned how to use the IPython magic command `! ls` to explore your current working directory. You can also do this natively in Python using the library `os`, which consists of miscellaneous operating system interfaces.

The first line of the following code imports the library `os`, the second line stores the name of the current directory in a string called `wd` and the third outputs the contents of the directory in a list to the shell. `~~~ import os wd = os.getcwd() os.listdir(wd) ~~~` Run this code in the IPython shell and answer the following questions. Ignore the files that begin with `..`

Check out the contents of your current directory and answer the following questions: (1) which file is in your directory and NOT an example of a flat file; (2) why is it not a flat file?

Instructions

Possible Answers

- `database.db` is not a flat file because relational databases contain structured relationships and flat files do not.
- `battledeath.xlsx` is not a flat because it is a spreadsheet consisting of many sheets, not a single table.
- `titanic.txt` is not a flat file because it is a `.txt`, not a `.csv`.

```
[1]: import os
wd = os.getcwd()
os.listdir(wd)
```

```
[1]: ['.ipynb_checkpoints', 'Data', 'Importing Data in Python.ipynb', 'Slide']
```

1.2.2 2.2 Loading a pickled file

There are a number of datatypes that cannot be saved easily to flat files, such as lists and dictionaries. 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** them. All this means is converting the object into a sequence of bytes, or a bytestream.

In this exercise, you'll import the pickle package, open a previously **pickled** data structure from a file and load it.

Instructions

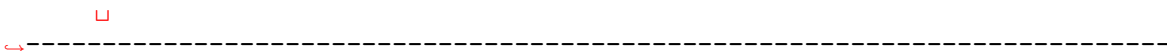
- Import the `pickle` package.
- Complete the second argument of `open()` so that it is read only for a binary file. This argument will be a string of two letters, one signifying 'read only', the other 'binary'.
- Pass the correct argument to `pickle.load()`; it should use the variable that is bound to `open`.
- Print the data, `d`.
- Print the datatype of `d`; take your mind back to your previous use of the function `type()`.

```
[3]: # 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))
```



```
FileNotFoundError                                Traceback (most recent call_
↳last)
```

```
<ipython-input-3-7b211b6854e7> in <module>
    3
    4 # Open pickle file and load data: d
----> 5 with open('data.pkl', 'rb') as file:
    6     d = pickle.load(file)
    7
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'data.pkl'
```

1.2.3 2.3 Listing sheets in Excel files

Whether you like it or not, any working data scientist will need to deal with Excel spreadsheets at some point in time. You won't always want to do so in Excel, however!

Here, you'll learn how to use **pandas** to import Excel spreadsheets and how to list the names of the sheets in any loaded **.xlsx** file.

Recall from the video that, given an Excel file imported into a variable **spreadsheet**, you can retrieve a list of the sheet names using the attribute **spreadsheet.sheet_names**.

Specifically, you'll be loading and checking out the spreadsheet '**battledeath.xlsx**', modified from the Peace Research Institute Oslo's (PRIO) **dataset**. This data contains age-adjusted mortality rates due to war in various countries over several years.

Instructions

- Assign the filename to the variable **file**.
- Pass the correct argument to **pd.ExcelFile()** to load the file using **pandas**.
- Print the sheetnames of the Excel spreadsheet by passing the necessary argument to the **print()** function.

```
[22]: # Import pandas
import pandas as pd

# Assign spreadsheet filename: file
file = './Data/battledeath.xlsx'

# Load spreadsheet: xl
xl = pd.ExcelFile(file)

# Print sheet names
print(xl.sheet_names)
```

```
['2002', '2004']
```


1.2.4 2.4 Importing sheets from Excel files

In the previous exercises, you saw that the Excel file contains two sheets, '2002' and '2004'. The next step is to import these.

In this exercise, you'll learn how to import any given sheet of your loaded .xlsx file as a DataFrame. You'll be able to do so by specifying either the sheet's name or its index.

The spreadsheet 'battleddeath.xlsx' is already loaded as `xl`.

Instructions

- Load the sheet '2004' into the DataFrame `df1` using its name as a string.
- Print the head of `df1` to the shell.
- Load the sheet 2002 into the DataFrame `df2` using its index (0).
- Print the head of `df2` to the shell.

```
[23]: # Load a sheet into a DataFrame by name: df1
df1 = xl.parse('2004')

# Print the head of the DataFrame df1
print(df1.head())

# Load a sheet into a DataFrame by index: df2
df2 = xl.parse('2002')

# Print the head of the DataFrame df2
print(df2.head())
```

	War(country)	2004
0	Afghanistan	9.451028
1	Albania	0.130354
2	Algeria	3.407277
3	Andorra	0.000000
4	Angola	2.597931

	War, age-adjusted mortality due to	2002
0	Afghanistan	36.083990
1	Albania	0.128908
2	Algeria	18.314120
3	Andorra	0.000000
4	Angola	18.964560

1.2.5 2.5 Customizing your spreadsheet import

Here, you'll parse your spreadsheets and use additional arguments to skip rows, rename columns and select only particular columns.

The spreadsheet 'battleddeath.xlsx' is already loaded as `xl`.

As before, you'll use the method `parse()`. This time, however, you'll add the additional arguments `skiprows`, `names` and `parse_cols`. These skip rows, name the columns and designate which

columns to parse, respectively. All these arguments can be assigned to lists containing the specific row numbers, strings and column numbers, as appropriate.

Instructions

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

```
[25]: # Parse the first sheet and rename the columns: df1
df1 = xl.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
#df2 = xl.parse(0, parse_cols=[0], skiprows=[0], names=['Country'])

# Print the head of the DataFrame df2
#print(df2.head())
```

	Country	AAM due to War (2002)
0	Albania	0.128908
1	Algeria	18.314120
2	Andorra	0.000000
3	Angola	18.964560
4	Antigua and Barbuda	0.000000

1.2.6 2.6 Importing SAS files

In this exercise, you'll figure out how to import a SAS file as a `DataFrame` using `SAS7BDAT` and `pandas`. The file 'sales.sas7bdat' is already in your working directory and both `pandas` and `matplotlib.pyplot` have already been imported as follows: `~~~ import pandas as pd import matplotlib.pyplot as plt ~~~` The data are adapted from the website of the undergraduate text book **Principles of Econometrics** by Hill, Griffiths and Lim.

Instructions

- Import the module `SAS7BDAT` from the library `sas7bdat`.
- 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 the head of the `DataFrame` `df_sas`.
- Execute your entire script to produce a histogram plot!

```
[29]: import pandas as pd
import matplotlib.pyplot as plt
```

```

# Import sas7bdat package
from sas7bdat import SAS7BDAT

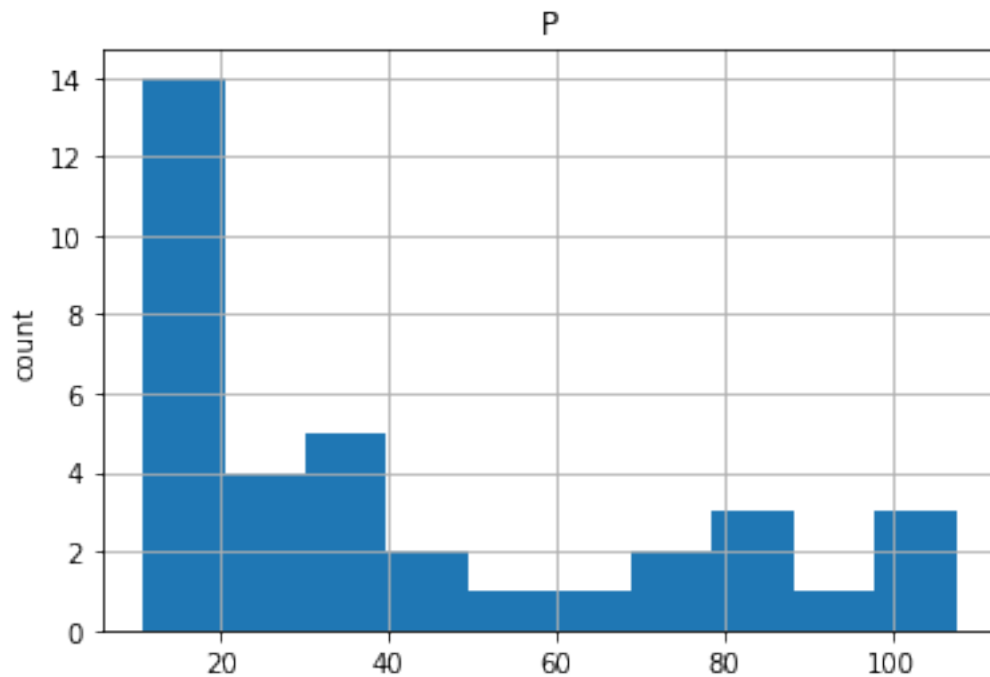
# Save file to a DataFrame: df_sas
with SAS7BDAT('./Data/sales.sas7bdat') as file:
    df_sas = file.to_data_frame()

# 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()

```

	YEAR	P	S
0	1950.0	12.9	181.899994
1	1951.0	11.9	245.000000
2	1952.0	10.7	250.199997
3	1953.0	11.3	265.899994
4	1954.0	11.2	248.500000



1.2.7 2.7 Importing Stata files

Here, you'll gain expertise in importing Stata files as DataFrames using the `pd.read_stata()` function from `pandas`. The last exercise's file, `'disarea.dta'`, is still in your working directory.

Instructions

- Use `pd.read_stata()` to load the file `'disarea.dta'` into the DataFrame `df`.
- Print the head of the DataFrame `df`.
- Visualize your results by plotting a histogram of the column `disa10`. We've already provided this code for you, so just run it!

```
[30]: # Import pandas
import pandas as pd

# Load Stata file into a pandas DataFrame: df
df = pd.read_stata('./Data/disarea.dta')

# Print the head of the DataFrame df
print(df.head())

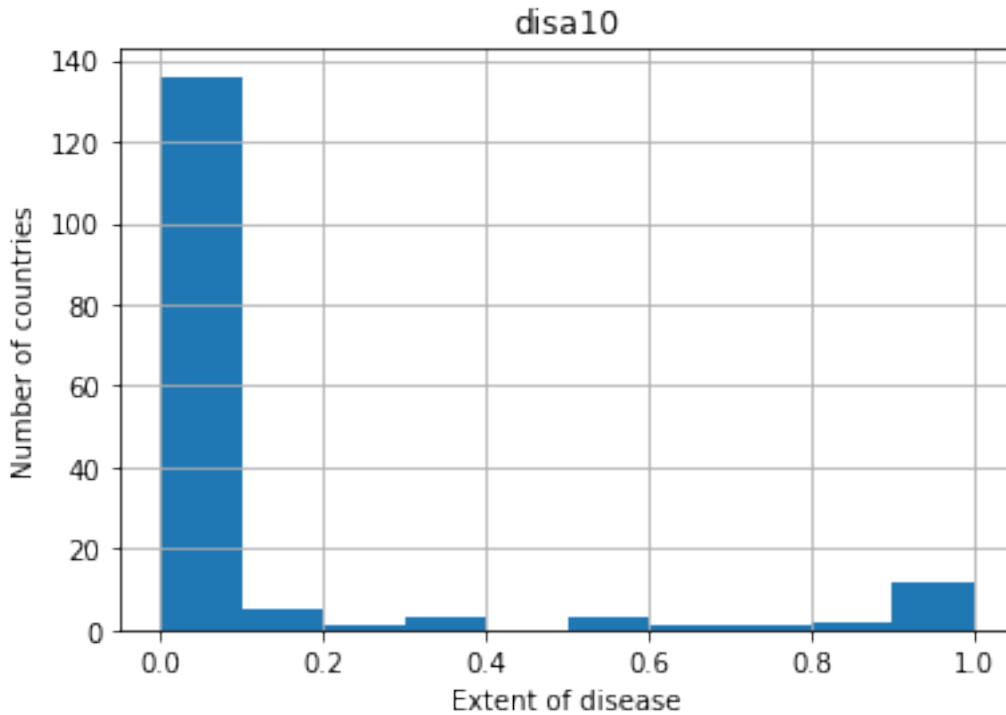
# Plot histogram of one column of the DataFrame
pd.DataFrame.hist(df[['disa10']])
plt.xlabel('Extent of disease')
plt.ylabel('Number of countries')
plt.show()
```

	wbcode		country	disa1	disa2	disa3	disa4	disa5	disa6	\
0	AFG		Afghanistan	0.00	0.00	0.76	0.73	0.0	0.00	
1	AGO		Angola	0.32	0.02	0.56	0.00	0.0	0.00	
2	ALB		Albania	0.00	0.00	0.02	0.00	0.0	0.00	
3	ARE	United Arab Emirates		0.00	0.00	0.00	0.00	0.0	0.00	
4	ARG		Argentina	0.00	0.24	0.24	0.00	0.0	0.23	

	disa7	disa8	...	disa16	disa17	disa18	disa19	disa20	disa21	disa22	\
0	0.00	0.0	...	0.0	0.0	0.0	0.00	0.00	0.0	0.00	
1	0.56	0.0	...	0.0	0.4	0.0	0.61	0.00	0.0	0.99	
2	0.00	0.0	...	0.0	0.0	0.0	0.00	0.00	0.0	0.00	
3	0.00	0.0	...	0.0	0.0	0.0	0.00	0.00	0.0	0.00	
4	0.00	0.0	...	0.0	0.0	0.0	0.00	0.05	0.0	0.00	

	disa23	disa24	disa25
0	0.02	0.00	0.00
1	0.98	0.61	0.00
2	0.00	0.00	0.16
3	0.00	0.00	0.00
4	0.01	0.00	0.11

[5 rows x 27 columns]



1.2.8 2.8 Using h5py to import HDF5 files

The file 'LIGO_data.hdf5' is already in your working directory. In this exercise, you'll import it using the `h5py` library. You'll also print out its datatype to confirm you have imported it correctly. You'll then study the structure of the file in order to see precisely what HDF groups it contains.

You can find the LIGO data plus loads of documentation and tutorials here. There is also a great tutorial on Signal Processing with the data here.

Instructions

- Import the package `h5py`.
- Assign the name of the file to the variable `file`.
- Load the file as read only into the variable `data`.
- Print the datatype of `data`.
- Print the names of the groups in the HDF5 file 'LIGO_data.hdf5'.

```
[34]: # Import packages
import numpy as np
import h5py

# Assign filename: file
file = './data/LIGO_data.hdf5'

# Load file: data
```

```
data = h5py.File(file, 'r')

# Print the datatype of the loaded file
print(type(data))

# Print the keys of the file
for key in data.keys():
    print(key)
```

```
<class 'h5py._hl.files.File'>
meta
quality
strain
```

1.2.9 2.9 Extracting data from your HDF5 file

In this exercise, you'll extract some of the LIGO experiment's actual data from the HDF5 file and you'll visualize it.

To do so, you'll need to first explore the HDF5 group 'strain'.

Instructions

- Assign the HDF5 group `data['strain']` to `group`.
- In the `for` loop, print out the keys of the HDF5 group in `group`.
- Assign to the variable `strain` the values of the time series data `data['strain']['Strain']` using the attribute `.value`.
- Set `num_samples` equal to 10000, the number of time points we wish to sample.
- Execute the rest of the code to produce a plot of the time series data in `LIGO_data.hdf5`.

```
[36]: import matplotlib.pyplot as plt

# Get the HDF5 group: group
group = data['strain']

# Check out keys of group
for key in group.keys():
    print(key)

# Set variable equal to time series data: strain
strain = data['strain']['Strain'].value

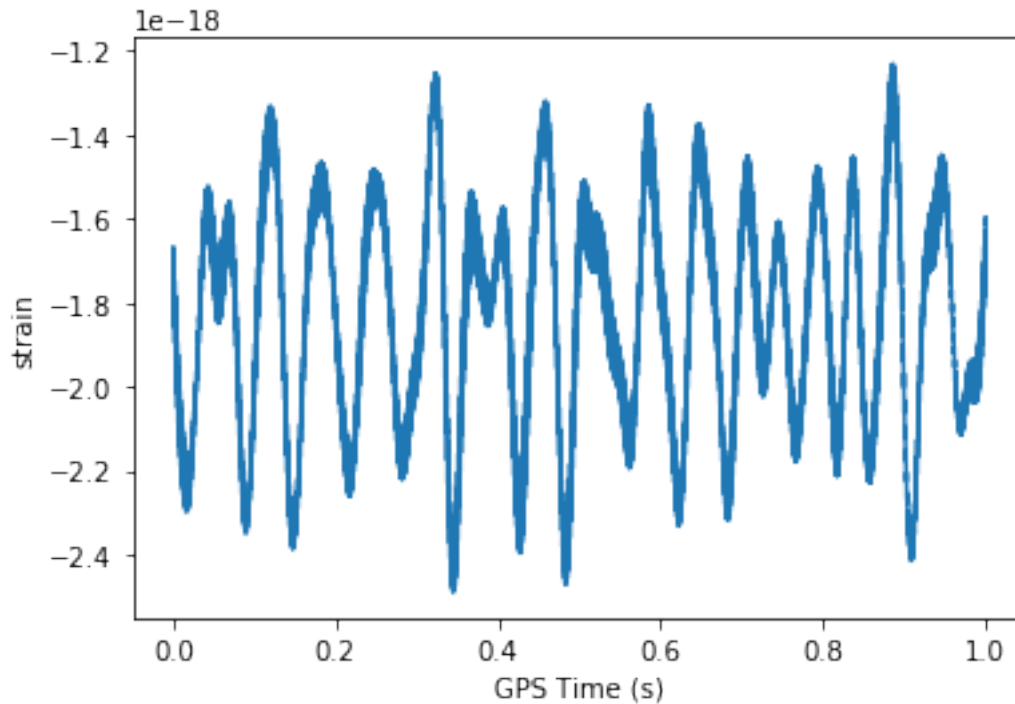
# Set number of time points to sample: num_samples
num_samples = 10000

# Set time vector
time = np.arange(0, 1, 1/num_samples)

# Plot data
```

```
plt.plot(time, strain[:num_samples])
plt.xlabel('GPS Time (s)')
plt.ylabel('strain')
plt.show()
```

Strain



1.2.10 2.10 Loading .mat files

In this exercise, you'll figure out how to load a MATLAB file using `scipy.io.loadmat()` and you'll discover what Python datatype it yields.

The file `'albeck_gene_expression.mat'` is in your working directory. This file contains gene expression data from the Albeck Lab at UC Davis. You can find the data and some great documentation [here](#).

Instructions

- Import the package `scipy.io`.
- Load the file `'albeck_gene_expression.mat'` into the variable `mat`; do so using the function `scipy.io.loadmat()`.
- Use the function `type()` to print the datatype of `mat` to the IPython shell.

```
[40]: # Import package
import scipy.io
```

```
# Load MATLAB file: mat
mat = scipy.io.loadmat('./Data/albeck_gene_expression.mat')

# Print the datatype type of mat
print(type(mat))
```

```
<class 'dict'>
```

1.2.11 2.11 The structure of .mat in Python

Here, you'll discover what is in the MATLAB dictionary that you loaded in the previous exercise.

The file 'albeck_gene_expression.mat' is already loaded into the variable `mat`. The following libraries have already been imported as follows: `~~~ import scipy.io import matplotlib.pyplot as plt import numpy as np ~~~` Once again, this file contains gene expression data from the Albeck Lab at UC Davis. You can find the data and some great documentation here.

Instructions

- Use the method `.keys()` on the dictionary `mat` to print the keys. Most of these keys (in fact the ones that do NOT begin and end with '___') are variables from the corresponding MATLAB environment.
- Print the type of the value corresponding to the key 'CYratioCyt' in `mat`. Recall that `mat['CYratioCyt']` accesses the value.
- Print the shape of the value corresponding to the key 'CYratioCyt' using the numpy function `shape()`.
- Execute the entire script to see some oscillatory gene expression data!

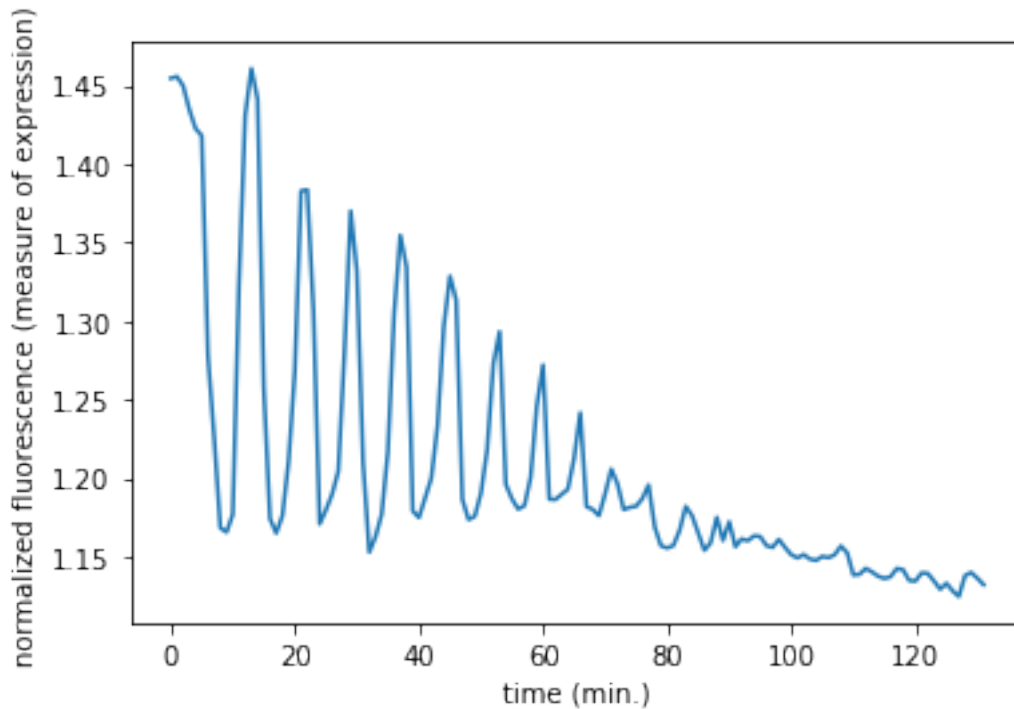
```
[41]: # Print the keys of the MATLAB dictionary
print(mat.keys())

# Print the type of the value corresponding to the key 'CYratioCyt'
print(type(mat['CYratioCyt']))

# Print the shape of the value corresponding to the key 'CYratioCyt'
print(np.shape(mat['CYratioCyt']))

# Subset the array and plot it
data = mat['CYratioCyt'][25, 5:]
fig = plt.figure()
plt.plot(data)
plt.xlabel('time (min.)')
plt.ylabel('normalized fluorescence (measure of expression)')
plt.show()
```

```
dict_keys(['__header__', '__version__', '__globals__', 'rfpCyt', 'rfpNuc',
'cfpNuc', 'cfpCyt', 'yfpNuc', 'yfpCyt', 'CYratioCyt'])
<class 'numpy.ndarray'>
(200, 137)
```

1.3 3. Working with relational databases in Python

In this chapter, you'll learn how to extract meaningful data from relational databases, an essential element of any data scientist's toolkit. You will be learning about the relational model, creating SQL queries, filtering and ordering your SQL records, and advanced querying by JOINing database tables.

1.3.1 3.1 Creating a database engine

Here, you're going to fire up your very first SQL engine. You'll create an engine to connect to the SQLite database '`Chinook.sqlite`', which is in your working directory. Remember that to create an engine to connect to '`Northwind.sqlite`', Hugo executed the command `engine = create_engine('sqlite:///Northwind.sqlite')` Here, '`sqlite:///Northwind.sqlite`' is called the connection string to the SQLite database `Northwind.sqlite`. A little bit of background on the Chinook database: the Chinook database contains information about a semi-fictional digital media store in which media data is real and customer, employee and sales data has been manually created.

Why the name Chinook, you ask? According to their website, `~~~~` The name of this sample database was based on the Northwind database. Chinooks are winds in the interior West of North America, where the Canadian Prairies and Great Plains meet various mountain ranges. Chinooks are most prevalent over southern Alberta in Canada. Chinook is a good name choice for a database that intends to be an alternative to Northwind. `~~~~` **Instructions**

- Import the function `create_engine` from the module `sqlalchemy`.

- Create an engine to connect to the SQLite database 'Chinook.sqlite' and assign it to engine.

```
[47]: # Import necessary module
from sqlalchemy import create_engine

# Create engine: engine
engine = create_engine('sqlite:///./Data/Chinook.sqlite')
```

1.3.2 3.2 What are the tables in the database?

In this exercise, you'll once again create an engine to connect to 'Chinook.sqlite'. Before you can get any data out of the database, however, you'll need to know what tables it contains!

To this end, you'll save the table names to a list using the method `table_names()` on the engine and then you will print the list.

Instructions

- Import the function `create_engine` from the module `sqlalchemy`.
- Create an engine to connect to the SQLite database 'Chinook.sqlite' and assign it to engine.
- Using the method `table_names()` on the engine `engine`, assign the table names of 'Chinook.sqlite' to the variable `table_names`.
- Print the object `table_names` to the shell.

```
[48]: # Import necessary module
from sqlalchemy import create_engine

# Create engine: engine
engine = create_engine('sqlite:///./Data/Chinook.sqlite')

# Save the table names to a list: table_names
table_names = engine.table_names()

# Print the table names to the shell
print(table_names)
```

```
['Album', 'Artist', 'Customer', 'Employee', 'Genre', 'Invoice', 'InvoiceLine',
'MediaType', 'Playlist', 'PlaylistTrack', 'Track']
```

1.3.3 3.3 The Hello World of SQL Queries!

Now, it's time for liftoff! In this exercise, you'll perform the Hello World of SQL queries, `SELECT`, in order to retrieve all columns of the table `Album` in the Chinook database. Recall that the query `SELECT *` selects all columns.

Instructions

- Open the engine connection as `con` using the method `connect()` on the engine.
- Execute the query that selects ALL columns from the `Album` table. Store the results in `rs`.

- Store all of your query results in the DataFrame `df` by applying the `fetchall()` method to the results `rs`.
- Close the connection!

```
[50]: # Import packages
from sqlalchemy import create_engine
import pandas as pd

# Create engine: engine
engine = create_engine('sqlite:///Data/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.fetchall())

# Close connection
con.close()

# Print head of DataFrame df
print(df.head())
```

	0	1	2
0	1	For Those About To Rock We Salute You	1
1	2	Balls to the Wall	2
2	3	Restless and Wild	2
3	4	Let There Be Rock	1
4	5	Big Ones	3

1.3.4 3.4 Customizing the Hello World of SQL Queries

Congratulations on executing your first SQL query! Now you're going to figure out how to customize your query in order to:

- Select specified columns from a table;
- Select a specified number of rows;
- Import column names from the database table.

Recall that Hugo performed a very similar query customization in the video: `~~~ engine = create_engine('sqlite:///Northwind.sqlite')`

with `engine.connect()` as `con`: `rs = con.execute("SELECT OrderID, OrderDate, ShipName FROM Orders")` `df = pd.DataFrame(rs.fetchmany(size=5))` `df.columns = rs.keys()` `~~~` Packages have already been imported as follows: `~~~ from sqlalchemy import create_engine import pandas as pd` `~~~` The engine has also already been created: `~~~ engine = create_engine('sqlite:///Chinook.sqlite')` `~~~` The engine connection is already open with the state-

ment ~~~ with engine.connect() as con: ~~~ All the code you need to complete is within this context.

Instructions

- Execute the SQL query that selects the columns `LastName` and `Title` from the `Employee` table. Store the results in the variable `rs`.
- Apply the method `fetchmany()` to `rs` in order to retrieve 3 of the records. Store them in the `DataFrame` `df`.
- Using the `rs` object, set the `DataFrame`'s column names to the corresponding names of the table columns.

```
[53]: # Open engine in context manager
# Perform query and save results to DataFrame: df
with engine.connect() as con:
    rs = con.execute('SELECT LastName, Title FROM Employee')
    df = pd.DataFrame(rs.fetchmany(size = 3))
    df.columns = rs.keys()

# Print the length of the DataFrame df
print(len(df))

# Print the head of the DataFrame df
print(df.head())
```

```
3
  LastName          Title
0    Adams  General Manager
1  Edwards    Sales Manager
2  Peacock  Sales Support Agent
```

1.3.5 3.4 Filtering your database records using SQL's WHERE

You can now execute a basic SQL query to select records from any table in your database and you can also perform simple query customizations to select particular columns and numbers of rows.

There are a couple more standard SQL query chops that will aid you in your journey to becoming an SQL ninja.

Let's say, for example that you wanted to get all records from the `Customer` table of the Chinook database for which the `Country` is 'Canada'. You can do this very easily in SQL using a `SELECT` statement followed by a `WHERE` clause as follows: ~~~ `SELECT * FROM Customer WHERE Country = 'Canada'` ~~~ In fact, you can filter any `SELECT` statement by any condition using a `WHERE` clause. This is called *filtering* your records.

In this interactive exercise, you'll select all records of the `Employee` table for which '`EmployeeId`' is greater than or equal to 6.

Packages are already imported as follows: ~~~ `import pandas as pd from sqlalchemy import create_engine` ~~~ Query away!

Instructions

- Complete the argument of `create_engine()` so that the engine for the SQLite database 'Chinook.sqlite' is created.
- Execute the query that selects all records from the `Employee` table where 'EmployeeId' is greater than or equal to 6. Use the `>=` operator and assign the results to `rs`.
- Apply the method `fetchall()` to `rs` in order to fetch all records in `rs`. Store them in the DataFrame `df`.
- Using the `rs` object, set the DataFrame's column names to the corresponding names of the table columns.

```
[54]: # Create engine: engine
engine = create_engine('sqlite:///./Data/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')
    df = pd.DataFrame(rs.fetchall())
    df.columns = rs.keys()

# Print the head of the DataFrame df
print(df.head())
```

	EmployeeId	LastName	FirstName	Title	ReportsTo	BirthDate	\
0	6	Mitchell	Michael	IT Manager	1	1973-07-01 00:00:00	
1	7	King	Robert	IT Staff	6	1970-05-29 00:00:00	
2	8	Callahan	Laura	IT Staff	6	1968-01-09 00:00:00	

	HireDate	Address	City	State	Country	\
0	2003-10-17 00:00:00	5827 Bowness Road NW	Calgary	AB	Canada	
1	2004-01-02 00:00:00	590 Columbia Boulevard West	Lethbridge	AB	Canada	
2	2004-03-04 00:00:00	923 7 ST NW	Lethbridge	AB	Canada	

	PostalCode	Phone	Fax	Email
0	T3B 0C5	+1 (403) 246-9887	+1 (403) 246-9899	michael@chinookcorp.com
1	T1K 5N8	+1 (403) 456-9986	+1 (403) 456-8485	robert@chinookcorp.com
2	T1H 1Y8	+1 (403) 467-3351	+1 (403) 467-8772	laura@chinookcorp.com

1.3.6 3.5 Ordering your SQL records with ORDER BY

You can also order your SQL query results. For example, if you wanted to get all records from the `Customer` table of the Chinook database and order them in increasing order by the column `SupportRepId`, you could do so with the following query: `~~ "SELECT * FROM Customer ORDER BY SupportRepId" ~~` In fact, you can order any `SELECT` statement by any column.

In this interactive exercise, you'll select all records of the `Employee` table and order them in increasing order by the column `BirthDate`.

Packages are already imported as follows: `~~~ import pandas as pd from sqlalchemy import cre-`

ate_engine ~~~ Get querying!

Instructions

- Using the function `create_engine()`, create an engine for the SQLite database `Chinook.sqlite` and assign it to the variable `engine`.
- In the context manager, execute the query that selects all records from the `Employee` table and orders them in increasing order by the column `BirthDate`. Assign the result to `rs`.
- In a call to `pd.DataFrame()`, apply the method `fetchall()` to `rs` in order to fetch all records in `rs`. Store them in the DataFrame `df`.
- Set the DataFrame's column names to the corresponding names of the table columns.

```
[3]: # Import packages
from sqlalchemy import create_engine
import pandas as pd

# Create engine: engine
engine = create_engine('sqlite:///./Data/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())
```

	EmployeeId	LastName	FirstName	Title	ReportsTo	\
0	4	Park	Margaret	Sales Support Agent	2.0	
1	2	Edwards	Nancy	Sales Manager	1.0	
2	1	Adams	Andrew	General Manager	NaN	
3	5	Johnson	Steve	Sales Support Agent	2.0	
4	8	Callahan	Laura	IT Staff	6.0	

	BirthDate	HireDate	Address	City	\
0	1947-09-19 00:00:00	2003-05-03 00:00:00	683 10 Street SW	Calgary	
1	1958-12-08 00:00:00	2002-05-01 00:00:00	825 8 Ave SW	Calgary	
2	1962-02-18 00:00:00	2002-08-14 00:00:00	11120 Jasper Ave NW	Edmonton	
3	1965-03-03 00:00:00	2003-10-17 00:00:00	7727B 41 Ave	Calgary	
4	1968-01-09 00:00:00	2004-03-04 00:00:00	923 7 ST NW	Lethbridge	

	State	Country	PostalCode	Phone	Fax	\
0	AB	Canada	T2P 5G3	+1 (403) 263-4423	+1 (403) 263-4289	
1	AB	Canada	T2P 2T3	+1 (403) 262-3443	+1 (403) 262-3322	
2	AB	Canada	T5K 2N1	+1 (780) 428-9482	+1 (780) 428-3457	
3	AB	Canada	T3B 1Y7	1 (780) 836-9987	1 (780) 836-9543	

```
4      AB      Canada      T1H 1Y8  +1 (403) 467-3351  +1 (403) 467-8772
```

```
                                Email
0  margaret@chinookcorp.com
1    nancy@chinookcorp.com
2  andrew@chinookcorp.com
3    steve@chinookcorp.com
4    laura@chinookcorp.com
```

```
[ ]:
```

1.3.7 3.6 Pandas and The Hello World of SQL Queries!

Here, you'll take advantage of the power of `pandas` to write the results of your SQL query to a `DataFrame` in one swift line of Python code!

You'll first import `pandas` and create the SQLite '`Chinook.sqlite`' engine. Then you'll query the database to select all records from the `Album` table.

Recall that to select all records from the `Orders` table in the Northwind database, Hugo executed the following command: `~~~ df = pd.read_sql_query("SELECT * FROM Orders", engine) ~~~`

Instructions

- Import the `pandas` package using the alias `pd`.
- Using the function `create_engine()`, create an engine for the SQLite database `Chinook.sqlite` and assign it to the variable `engine`.
- Use the `pandas` function `read_sql_query()` to assign to the variable `df` the `DataFrame` of results from the following query: select all records from the table `Album`.
- The remainder of the code is included to confirm that the `DataFrame` created by this method is equal to that created by the previous method that you learned.

```
[4]: # Import packages
from sqlalchemy import create_engine
import pandas as pd

# Create engine: engine
engine = create_engine('sqlite:///./Data/Chinook.sqlite')

# Execute query and store records in DataFrame: df
df = pd.read_sql_query('SELECT * FROM Album', engine)

# Print head of DataFrame
print(df.head())

# Open engine in context manager and store query result in df1
with engine.connect() as con:
    rs = con.execute("SELECT * FROM Album")
    df1 = pd.DataFrame(rs.fetchall())
    df1.columns = rs.keys()
```

```
# Confirm that both methods yield the same result
print(df.equals(df1))
```

	AlbumId	Title	ArtistId
0	1	For Those About To Rock We Salute You	1
1	2	Balls to the Wall	2
2	3	Restless and Wild	2
3	4	Let There Be Rock	1
4	5	Big Ones	3

True

1.3.8 3.7 Pandas for more complex querying

Here, you'll become more familiar with the pandas function `read_sql_query()` by using it to execute a more complex query: a `SELECT` statement followed by both a `WHERE` clause AND an `ORDER BY` clause.

You'll build a `DataFrame` that contains the rows of the `Employee` table for which the `EmployeeId` is greater than or equal to 6 and you'll order these entries by `BirthDate`.

Instructions

- Using the function `create_engine()`, create an engine for the SQLite database `Chinook.sqlite` and assign it to the variable `engine`.
- Use the pandas function `read_sql_query()` to assign to the variable `df` the `DataFrame` of results from the following query: select all records from the `Employee` table where the `EmployeeId` is greater than or equal to 6 and ordered by `BirthDate` (make sure to use `WHERE` and `ORDER BY` in this precise order).

```
[2]: # Import packages
from sqlalchemy import create_engine
import pandas as pd

# Create engine: engine
engine = create_engine('sqlite:///./Data/Chinook.sqlite')

# Execute query and store records in DataFrame: df
df = pd.read_sql_query('SELECT * FROM Employee WHERE EmeId >= 6 ORDER BY BirthDate', engine)

# Print head of DataFrame
print(df.head())
```

↳ -----


```

OperationalError                                Traceback (most recent call
↳last)

~\.conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\base.py in
↳_execute_context(self, dialect, constructor, statement, parameters, *args)
1243             self.dialect.do_execute(
-> 1244                 cursor, statement, parameters, context
1245             )

~\.conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\default.py in
↳do_execute(self, cursor, statement, parameters, context)
551     def do_execute(self, cursor, statement, parameters,
↳context=None):
--> 552         cursor.execute(statement, parameters)
553

```

OperationalError: no such column: EmeeId

The above exception was the direct cause of the following exception:

```

OperationalError                                Traceback (most recent call
↳last)

<ipython-input-2-d865aac23d18> in <module>
7
8 # Execute query and store records in DataFrame: df
----> 9 df = pd.read_sql_query('SELECT * FROM Employee WHERE EmeeId >= 6
↳ORDER BY BirthDate', engine)
10
11 # Print head of DataFrame

~\.conda\envs\datacamp\lib\site-packages\pandas\io\sql.py in
↳read_sql_query(sql, con, index_col, coerce_float, params, parse_dates,
↳chunksize)
312     return pandas_sql.read_query(
313         sql, index_col=index_col, params=params,
↳coerce_float=coerce_float,
-> 314         parse_dates=parse_dates, chunksize=chunksize)
315
316

```

```

~\.conda\envs\datacamp\lib\site-packages\pandas\io\sql.py in
↳ read_query(self, sql, index_col, coerce_float, parse_dates, params, chunksize)
    1097         args = _convert_params(sql, params)
    1098
-> 1099         result = self.execute(*args)
    1100         columns = result.keys()
    1101

```

```

~\.conda\envs\datacamp\lib\site-packages\pandas\io\sql.py in
↳ execute(self, *args, **kwargs)
    988     def execute(self, *args, **kwargs):
    989         """Simple passthrough to SQLAlchemy connectable"""
--> 990         return self.connectable.execute(*args, **kwargs)
    991
    992     def read_table(self, table_name, index_col=None,
↳ coerce_float=True,

```

```

~\.conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\base.py in
↳ execute(self, statement, *multiparams, **params)
    2164
    2165         connection = self._contextual_connect(close_with_result=True)
-> 2166         return connection.execute(statement, *multiparams, **params)
    2167
    2168     def scalar(self, statement, *multiparams, **params):

```

```

~\.conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\base.py in
↳ execute(self, object_, *multiparams, **params)
    980         """
    981         if isinstance(object_, util.string_types[0]):
--> 982             return self._execute_text(object_, multiparams, params)
    983         try:
    984             meth = object_._execute_on_connection

```

```

~\.conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\base.py in
↳ _execute_text(self, statement, multiparams, params)
    1153         parameters,
    1154         statement,
-> 1155         parameters,
    1156     )
    1157     if self._has_events or self.engine._has_events:

```

```

~\conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\base.py in
↳ _execute_context(self, dialect, constructor, statement, parameters, *args)
    1246         except BaseException as e:
    1247             self._handle_dbapi_exception(
-> 1248                 e, statement, parameters, cursor, context
    1249             )
    1250

```

```

~\conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\base.py in
↳ _handle_dbapi_exception(self, e, statement, parameters, cursor, context)
    1464         util.raise_from_cause(newraise, exc_info)
    1465         elif should_wrap:
-> 1466             util.raise_from_cause(sqlalchemy_exception, exc_info)
    1467         else:
    1468             util.reraise(*exc_info)

```

```

~\conda\envs\datacamp\lib\site-packages\sqlalchemy\util\compat.py in
↳ raise_from_cause(exception, exc_info)
    381     exc_type, exc_value, exc_tb = exc_info
    382     cause = exc_value if exc_value is not exception else None
--> 383     reraise(type(exception), exception, tb=exc_tb, cause=cause)
    384
    385

```

```

~\conda\envs\datacamp\lib\site-packages\sqlalchemy\util\compat.py in
↳ reraise(tp, value, tb, cause)
    126         value.__cause__ = cause
    127         if value.__traceback__ is not tb:
--> 128             raise value.with_traceback(tb)
    129         raise value
    130

```

```

~\conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\base.py in
↳ _execute_context(self, dialect, constructor, statement, parameters, *args)
    1242         if not evt_handled:
    1243             self.dialect.do_execute(
-> 1244                 cursor, statement, parameters, context
    1245             )
    1246         except BaseException as e:

```

```

~\conda\envs\datacamp\lib\site-packages\sqlalchemy\engine\default.py in
↳ do_execute(self, cursor, statement, parameters, context)

```

```

550
551     def do_execute(self, cursor, statement, parameters,
↪context=None):
--> 552         cursor.execute(statement, parameters)
553
554     def do_execute_no_params(self, cursor, statement, context=None):

```

```

OperationalError: (sqlite3.OperationalError) no such column: EmeeId
[SQL: SELECT * FROM Employee WHERE EmeeId >= 6 ORDER BY BirthDate]
(Background on this error at: http://sqlalche.me/e/e3q8)

```

1.3.9 3.8 The power of SQL lies in relationships between tables: INNER JOIN

Here, you'll perform your first `INNER JOIN`! You'll be working with your favourite SQLite database, `Chinook.sqlite`. For each record in the `Album` table, you'll extract the `Title` along with the `Name` of the `Artist`. The latter will come from the `Artist` table and so you will need to `INNER JOIN` these two tables on the `ArtistID` column of both.

Recall that to `INNER JOIN` the `Orders` and `Customers` tables from the `Northwind` database, Hugo executed the following SQL query:

```
~~ "SELECT OrderID, CompanyName FROM Orders INNER JOIN Customers on Orders.CustomerID = Customers.CustomerID" ~~
```

The following code has already been executed to import the necessary packages and to create the engine: `~~~ import pandas as pd from sqlalchemy import create_engine engine = create_engine('sqlite:///Chinook.sqlite') ~~~`

Instructions

- Assign to `rs` the results from the following query: select all the records, extracting the `Title` of the record and `Name` of the artist of each record from the `Album` table and the `Artist` table, respectively. To do so, `INNER JOIN` these two tables on the `ArtistID` column of both.
- In a call to `pd.DataFrame()`, apply the method `fetchall()` to `rs` in order to fetch all records in `rs`. Store them in the `DataFrame` `df`.
- Set the `DataFrame`'s column names to the corresponding names of the table columns.

```

[3]: engine = create_engine('sqlite:///Data/Chinook.sqlite')
# Open engine in context manager
# Perform query and save results to DataFrame: df
with engine.connect() as con:
    rs = con.execute('SELECT Title, Name FROM Album INNER JOIN Artist on Album.
↪ArtistID = Artist.ArtistID')
    df = pd.DataFrame(rs.fetchall())
    df.columns = rs.keys()

# Print head of DataFrame df
print(df.head())

```

	Title	Name
0	For Those About To Rock We Salute You	AC/DC

1	Balls to the Wall	Accept
2	Restless and Wild	Accept
3	Let There Be Rock	AC/DC
4	Big Ones	Aerosmith

1.3.10 3.9 Filtering your INNER JOIN

Congrats on performing your first INNER JOIN! You're now going to finish this chapter with one final exercise in which you perform an INNER JOIN and filter the result using a WHERE clause.

Recall that to INNER JOIN the Orders and Customers tables from the Northwind database, Hugo executed the following SQL query:

```
~~ "SELECT OrderID, CompanyName FROM Orders INNER JOIN Customers on Orders.CustomerID = Customers.CustomerID" ~~
```

The following code has already been executed to import the necessary packages and to create the engine: `~~~ import pandas as pd from sqlalchemy import create_engine engine = create_engine('sqlite:///Chinook.sqlite')` ***Instructions**

- Use the pandas function `read_sql_query()` to assign to the variable `df` the DataFrame of results from the following query: select all records from `PlaylistTrack` INNER JOIN `Track` on `PlaylistTrack.TrackId = Track.TrackId` that satisfy the condition `Milliseconds < 250000`.

```
[4]: # 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())
```

	PlaylistId	TrackId	TrackId	Name	AlbumId	MediaTypeId	\
0	1	3390	3390	One and the Same	271	2	
1	1	3392	3392	Until We Fall	271	2	
2	1	3393	3393	Original Fire	271	2	
3	1	3394	3394	Broken City	271	2	
4	1	3395	3395	Somedays	271	2	

	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	23	None	217732	3559040	0.99
1	23	None	230758	3766605	0.99
2	23	None	218916	3577821	0.99
3	23	None	228366	3728955	0.99
4	23	None	213831	3497176	0.99

2 Importing Data in Python (Part 2)

As a Data Scientist, on a daily basis you will need to clean data, wrangle and munge it, visualize it, build predictive models and interpret these models. Before doing any of these, however, you will need to know how to get data into Python. In the prequel to this course, you have already

learnt many ways to import data into Python: (i) from flat files such as .txts and .csvs; (ii) from files native to other software such as Excel spreadsheets, Stata, SAS and MATLAB files; (iii) from relational databases such as SQLite & PostgreSQL. In this course, you'll extend this knowledge base by learning to import data (i) from the web and (ii) a special and essential case of this: pulling data from Application Programming Interfaces, also known as APIs, such as the Twitter streaming API, which allows us to stream real-time tweets.

2.1 4 Importing data from the Internet

The web is a rich source of data from which you can extract various types of insights and findings. In this chapter, you will learn how to get data from the web, whether it be stored in files or in HTML. You'll also learn the basics of scraping and parsing web data.

2.1.1 4.1 Importing flat files from the web: your turn!

You are about to import your first file from the web! The flat file you will import will be 'winequality-red.csv' from the University of California, Irvine's Machine Learning repository. The flat file contains tabular data of physiochemical properties of red wine, such as pH, alcohol content and citric acid content, along with wine quality rating.

The URL of the file is `https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv`

After you import it, you'll check your working directory to confirm that it is there and then you'll load it into a `pandas` `DataFrame`.

Instructions

- Import the function `urlretrieve` from the subpackage `urllib.request`.
- Assign the URL of the file to the variable `url`.
- Use the function `urlretrieve()` to save the file locally as 'winequality-red.csv'.
- Execute the remaining code to load 'winequality-red.csv' in a `pandas` `DataFrame` and to print its head to the shell.

```
[6]: # Import package
from urllib.request import urlretrieve

# Import pandas
import pandas as pd

# Assign url of file: url
url = 'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/
      ↪datasets/winequality-red.csv'

# Save file locally
urlretrieve(url, 'winequality-red.csv')

# Read file into a DataFrame and print its head
df = pd.read_csv('winequality-red.csv', sep=';')
print(df.head())
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	\
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	\
0	11.0	34.0	0.9978	3.51	0.56	
1	25.0	67.0	0.9968	3.20	0.68	
2	15.0	54.0	0.9970	3.26	0.65	
3	17.0	60.0	0.9980	3.16	0.58	
4	11.0	34.0	0.9978	3.51	0.56	

	alcohol	quality
0	9.4	5
1	9.8	5
2	9.8	5
3	9.8	6
4	9.4	5

2.1.2 4.2 Opening and reading flat files from the web

You have just imported a file from the web, saved it locally and loaded it into a DataFrame. If you just wanted to load a file from the web into a DataFrame without first saving it locally, you can do that easily using **pandas**. In particular, you can use the function `pd.read_csv()` with the URL as the first argument and the separator `sep` as the second argument.

The URL of the file, once again, is `https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv`.

Instructions

- Assign the URL of the file to the variable `url`.
- Read file into a DataFrame `df` using `pd.read_csv()`, recalling that the separator in the file is `;`.
- Print the head of the DataFrame `df`.
- Execute the rest of the code to plot histogram of the first feature in the DataFrame `df`.

```
[2]: # Import packages
import matplotlib.pyplot as plt
import pandas as pd

# Assign url of file: url
url = 'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/
↳datasets/winequality-red.csv'

# Read file into a DataFrame: df
df = pd.read_csv(url, sep = ';')
```

```
# Print the head of the DataFrame
print(df.head())

# Plot first column of df
pd.DataFrame.hist(df.ix[:, 0:1])
plt.xlabel('fixed acidity (g(tartaric acid)/dm3)')
plt.ylabel('count')
plt.show()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	\
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	\
0	11.0	34.0	0.9978	3.51	0.56	
1	25.0	67.0	0.9968	3.20	0.68	
2	15.0	54.0	0.9970	3.26	0.65	
3	17.0	60.0	0.9980	3.16	0.58	
4	11.0	34.0	0.9978	3.51	0.56	

	alcohol	quality
0	9.4	5
1	9.8	5
2	9.8	5
3	9.8	6
4	9.4	5

C:\Users\124501\.conda\envs\datacamp\lib\site-packages\ipykernel_launcher.py:16:

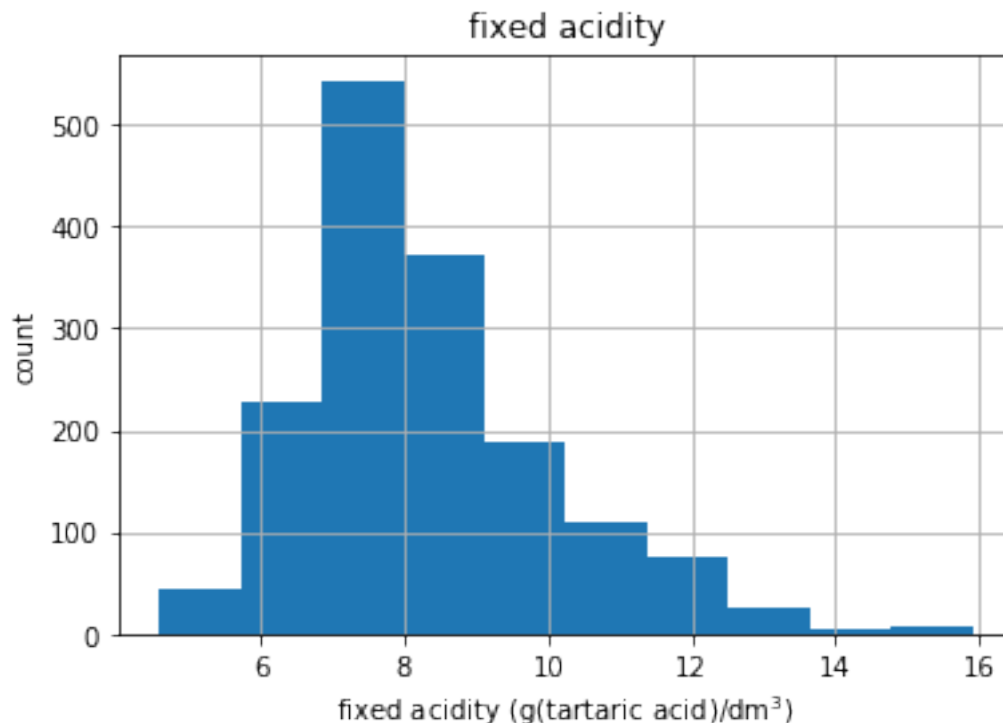
DeprecationWarning:

.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:

<http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated>

app.launch_new_instance()



2.1.3 4.3 Importing non-flat files from the web

Congrats! You've just loaded a flat file from the web into a DataFrame without first saving it locally using the `pandas` function `pd.read_csv()`. This function is super cool because it has close relatives that allow you to load all types of files, not only flat ones. In this interactive exercise, you'll use `pd.read_excel()` to import an Excel spreadsheet.

The URL of the spreadsheet is `http://s3.amazonaws.com/assets.datacamp.com/course/importing_data_into_r/latitude.xls`. Your job is to use `pd.read_excel()` to read in all of its sheets, print the sheet names and then print the head of the first sheet using its name, not its index.

Note that the output of `pd.read_excel()` is a Python dictionary with sheet names as keys and corresponding DataFrames as corresponding values.

Instructions

- Assign the URL of the file to the variable `url`.
- Read the file in `url` into a dictionary `x1` using `pd.read_excel()` recalling that, in order to import all sheets you need to pass `None` to the argument `sheetname`.
- Print the names of the sheets in the Excel spreadsheet; these will be the keys of the dictionary `x1`.
- Print the head of the first sheet using the sheet name, not the index of the sheet! The sheet name is '1700'

```
[4]: # 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, sheet_name=None)

# Print the sheetnames to the shell
print(xl.keys())

# Print the head of the first sheet (using its name, NOT its index)
print(xl['1700'].head())
```

```
odict_keys(['1700', '1900'])
              country      1700
0      Afghanistan  34.565000
1  Akrotiri and Dhekelia  34.616667
2              Albania  41.312000
3              Algeria  36.720000
4      American Samoa -14.307000
```

2.1.4 4.4 Performing HTTP requests in Python using urllib

Now that you know the basics behind HTTP GET requests, it's time to perform some of your own. In this interactive exercise, you will ping our very own DataCamp servers to perform a GET request to extract information from our teach page, "<http://www.datacamp.com/teach/documentation>".

In the next exercise, you'll extract the HTML itself. Right now, however, you are going to package and send the request and then catch the response.

Instructions

- Import the functions `urlopen` and `Request` from the subpackage `urllib.request`.
- Package the request to the url "<http://www.datacamp.com/teach/documentation>" using the function `Request()` and assign it to `request`.
- Send the request and catch the response in the variable `response` with the function `urlopen()`.
- Run the rest of the code to see the datatype of `response` and to close the connection!

```
[6]: # Import packages
from urllib.request import urlopen, Request

# Specify the url
url = "http://www.datacamp.com/teach/documentation"
```

```

# This packages the request: request
request = Request(url)

# Sends the request and catches the response: response
response = urlopen(request)

# Print the datatype of response
print(type(response))

# Be polite and close the response!
response.close()

```

```
<class 'http.client.HTTPResponse'>
```

2.1.5 4.5 Printing HTTP request results in Python using urllib

You have just packaged and sent a GET request to "http://www.datacamp.com/teach/documentation" and then caught the response. You saw that such a response is a `http.client.HTTPResponse` object. The question remains: what can you do with this response?

Well, as it came from an HTML page, you could read it to extract the HTML and, in fact, such a `http.client.HTTPResponse` object has an associated `read()` method. In this exercise, you'll build on your previous great work to extract the response and print the HTML.

Instructions

- Send the request and catch the `response` in the variable `response` with the function `urlopen()`, as in the previous exercise.
- Extract the response using the `read()` method and store the result in the variable `html`.
- Print the string `html`.
- Hit submit to perform all of the above and to close the response: be tidy!

```

[7]: # 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()
```

```
b'<!doctype html>\n<html lang="en" data-direction="ltr">\n  <head>\n    <link
href="https://fonts.intercomcdn.com" rel="preconnect" crossorigin>\n
<script src="https://www.googletagmanager.com/gtag/js?id=UA-39297847-9"
async="async" nonce="CspiV1BOZ1bwrJ1/+YKB2KhncG09YL1YZgZdRzoMqcw="></script>\n
<script nonce="CspiV1BOZ1bwrJ1/+YKB2KhncG09YL1YZgZdRzoMqcw=">\n
window.dataLayer = window.dataLayer || [];\n      function
gtag(){dataLayer.push(arguments);} \n      gtag(\'js\', new Date());\n
gtag(\'config\', \'UA-39297847-9\');\n</script>\n    <meta charset="utf-8">\n
<meta http-equiv="X-UA-Compatible" content="IE=edge">\n    <title>DataCamp Help
Center</title>\n    <meta name="description" content="">\n    <meta
name="viewport" content="width=device-width, initial-scale=1">\n    <meta
name="intercom:trackingEvent" content="{&quot;name&quot;:&quot;Viewed Help Cente
r&quot;,&quot;metadata&quot;:&quot;action&quot;:&quot;viewed&quot;,&quot;object
&quot;:&quot;educate_home&quot;,&quot;place&quot;:&quot;help_center&quot;,&quot;
owner&quot;:&quot;educate&quot;}" />\n\n    <link rel="stylesheet" media="all"
href="https://intercom.help/_assets/application-c71cce033306dfc55f25ff3f87dd3278
ec16161813e551727a60f18a92811bf6.css" />\n    <link rel="canonical"
href="http://instructor-support.datacamp.com/" />\n\n    <link
href="https://static.intercomassets.com/assets/educate/educate-favicon-64x64-at-
2x-52016a3500a250d0b118c0a04ddd13b1a7364a27759483536dd1940bccdefc20.png"
rel="shortcut icon" type="image/png" />\n    <style>\n      .header,
.avatar__image-extra { background-color: #263e63; }\n      .article a,
.c__primary { color: #263e63; }\n      .avatar__fallback { background-color:
#263e63; }\n      article a.intercom-h2b-button { background-color: #263e63;
border: 0; }\n    </style>\n\n    <meta property="og:title"
content="DataCamp Help Center" />\n    <meta name="twitter:title"
content="DataCamp Help Center" />\n\n    <meta property="og:type"
content="website" />\n    <meta property="og:image" content="" />\n\n    <meta
name="twitter:image" content="" />\n\n  </head>\n  <body class="">\n    <header
class="header">\n    <div class="container header__container o__ltr" dir="ltr">\n
<div class="content">\n      <div class="mo o__centered o__reversed
header__meta_wrapper">\n        <div class="mo__body">\n          <div
class="header__logo">\n            <a href="/">\n              \n
</a>\n          </div>\n          </div>\n          <div class="mo__aside">\n
<div class="header__home__url">\n            \n            <a target="_blank"
rel=\'noopener\' href="http://www.datacamp.com/teach"><svg width="14"
height="14" viewBox="0 0 14 14" xmlns="http://www.w3.org/2000/svg"><title>Group
65</title><g stroke="#FFF" fill="none" fill-rule="evenodd" stroke-
linecap="round" stroke-linejoin="round"><path d="M11.5
6.73v6.77H.5v-11h7.615M4.5 9.517-7M13.5 5.5v-5h-5"/></g></svg><span>Go to
```

```

DataCamp</span></a>\n          </div>\n          </div>\n          </div>\n
<h1 class="header__headline">Advice and answers from the DataCamp Team</h1>\n
<form action="/" autocomplete="off" class="header__form search">\n          <input
type="text" autocomplete="off" class="search__input js__search-input o__ltr"
placeholder="Search for articles..." tabindex="1" name="q" value="">\n
<div class="search_icons">\n          <button type="submit"
class="search__submit o__ltr"></button>\n          <a class="search__clear-
text__icon">\n          <svg class="interface-icon"
xmlns="http://www.w3.org/2000/svg" width="16" height="16" viewBox="0 0 16 16">\n
<path d="M8.018 6.643L5.375 4 4 5.375L2.643 2.643L4 10.643 5.375
12L2.643-2.625L10.625 12 12 10.643 9.357 8.018 12 5.375 10.643 4z" />\n
</svg>\n          </a>\n          </form>\n          </div>\n          </div>\n
</div>\n</header>\n\n      <div class="container">\n          <div class="content
educate_content"><section class="section">\n          <div class="g__space">\n          <a
href="/getting-started" class="paper ">\n          <div class="collection
o__ltr">\n          <div class="collection__photo">\n          <svg
role='img' viewBox='0 0 48 48'><g id="chat-star" stroke-width="2"
fill="none" fill-rule="evenodd" stroke-linejoin="round"><path d="M20
34.942c-2.083-.12-4.292-.42-6-.942L3 3914-9c-3.858-3.086-6-7.246-6-12C1 8.61
10.328 1 21.835 1 33.343 1 43 8.61 43 18c0 1.044-.117 2.065-.342
3.057"></path><path d="M36.016 25L40 33h7l-6 5 3 9-8-5.494L28
47l3-9-6-5h7l4.016-8z"></path></g></svg>\n          </div>\n          <div
class="collection_meta" dir="ltr">\n          <h2 class="t__h3
c__primary">Getting Started</h2>\n          <p
class="paper__preview">Everything you need to know to begin your DataCamp
journey!</p>\n          <div class="avatar">\n          <div class="avatar__photo
avatars__images o__ltr">\n          \n          \n          \n          <span
class="avatar__image avatar__fallback">+2</span>\n          </div>\n          <div
class="avatar__info">\n          <div>\n          <span class="c__darker">\n          11
articles in this collection\n          </span>\n          <br>\n          Written by <span
class='c__darker'> Becca Robins,</span> <span class='c__darker'> Jen
Bricker,</span> <span class='c__darker'> Yashas Roy</span> and 2 others\n
</div>\n          </div>\n</div>\n\n          </div>\n          </div>\n          </a>\n
</div>\n          <div class="g__space">\n          <a href="/courses" class="paper ">\n
<div class="collection o__ltr">\n          <div class="collection__photo">\n
<svg role='img' viewBox='0 0 48 48'><g id="devices-laptop" stroke-width="2"
fill="none" fill-rule="evenodd" stroke-linecap="round"><path d="M41
31H7V11h34v20z"></path><path d="M3 35V10a3 3 0 0 1 3-3h36a3 3 0 0 1 3 3v25m-16
0v2H19v-2H14a2 2 0 0 0 2 2h4a2 2 0 0 0 2-2v-4H29z" stroke-
linejoin="round"></path></g></svg>\n          </div>\n          <div
class="collection_meta" dir="ltr">\n          <h2 class="t__h3
c__primary">Courses</h2>\n          <p class="paper__preview">Everything you

```

```

need to know about creating DataCamp courses.</p>\n
class="avatar">\n <div class="avatar__photo avatars__images o__ltr">\n
\n\n
\n\n \n\n <span class="avatar__image
avatar__fallback">+10</span>\n </div>\n <div class="avatar__info">\n
<div>\n <span class="c__darker">\n 84 articles in this collection\n
</span>\n <br>\n Written by <span class="'c__darker'"> Yashas
Roy,</span> <span class="'c__darker'"> Nick Carchedi,</span> <span
class="'c__darker'"> Richie Cotton</span> and 10 others\n </div>\n
</div>\n</div>\n\n </div>\n </div>\n </a>\n </div>\n
<div class="g__space">\n <a href="/daily-practice" class="paper ">\n
<div class="collection o__ltr">\n <div class="collection_photo">\n
<svg role='img' viewBox='0 0 48 48'><g id="tools-dashboard" stroke-width="2"
fill="none" fill-rule="evenodd" stroke-linecap="round" stroke-
linejoin="round"><path d="M27 31a3 3 0 0 1-6 0 3 3 0 0 1 6
0zm-.88-2.12l9.9-9.9M5 32h4m34 .002L39 32m2.553-8.27l-3.696 1.53M31.27
13.447l-1.53 3.695M24 12v4m-7.27-2.553l1.53 3.695m-7.694.422l2.826 2.83M6.447
23.73l3.695 1.53"></path><path d="M24 8C11.297 8 1 18.3 1 31v9h46v-9C47 18.3
36.703 8 24 8z"></path></g></svg>\n </div>\n <div
class="collection_meta" dir="ltr">\n <h2 class="t__h3
c__primary">Daily Practice</h2>\n <p
class="paper__preview">Everything you need to know about creating DataCamp Daily
Practice.</p>\n <div class="avatar">\n <div class="avatar__photo
avatars__images o__ltr">\n \n\n </div>\n <div
class="avatar__info">\n <div>\n <span class="c__darker">\n 15
articles in this collection\n </span>\n <br>\n Written by <span
class="'c__darker'"> Anneleen Beckers</span>\n </div>\n </div>\n</div>\n\n
</div>\n </div>\n </a>\n </div>\n <div class="g__space">\n
<a href="/projects" class="paper ">\n <div class="collection o__ltr">\n
<div class="collection_photo">\n <svg role='img' viewBox='0 0 48
48'><g id="book-opened2"><path d="M24 11c0-3.866 10.297-7 23-7v33c-12.703 0-23
3.134-23 7 0-3.866-10.3-7-23-7V4c12.7 0 23 3.134 23 7zm0
0v32m-5-27.52c-3.22-1.232-7.773-2.128-13-2.48m13
8.48c-3.22-1.232-7.773-2.128-13-2.48m13 8.48c-3.22-1.232-7.773-2.128-13-2.48m13
8.48c-3.22-1.23-7.773-2.127-13-2.48m23-15.52c3.223-1.232 7.773-2.128 13-2.48m-13
8.48c3.223-1.232 7.773-2.128 13-2.48m-13 8.48c3.223-1.232 7.773-2.128
13-2.48m-13 8.48c3.223-1.23 7.773-2.127 13-2.48" stroke-width="2" fill="none"
stroke-linecap="round" stroke-linejoin="round"></path></g></svg>\n
</div>\n <div class="collection_meta" dir="ltr">\n <h2
class="t__h3 c__primary">Projects</h2>\n <p

```

```

class="paper__preview">Everything you need to know about creating DataCamp
projects.</p>\n      <div class="avatar">\n    <div class="avatar__photo
avatars__images o__ltr">\n      \n\n    </div>\n    <div
class="avatar__info">\n      <div>\n        <span class="c__darker">\n          19
articles in this collection\n        </span>\n      <br>\n      Written by <span
class='\c__darker\'> David Venturi</span>\n    </div>\n  </div>\n</div>\n\n
</div>\n    </div>\n    </a>\n    </div>\n    <div class="g__space">\n
<a href="/course-editor-basics" class="paper ">\n      <div class="collection
o__ltr">\n        <div class="collection__photo">\n          <svg
role='\img\' viewBox='\0 0 48 48\'><g id="book-bookmark" stroke-width="2"
fill="none" fill-rule="evenodd" stroke-linecap="round"><path d="M35 31l-6-6-6
6V7h12v24z"></path><path d="M35 9h6v38H11a4 4 0 0 1-4-4V5" stroke-
linejoin="round"></path><path d="M39 9V1H11a4 4 0 0 0 0 8h12" stroke-
linejoin="round"></path></g></svg>\n          </div>\n        <div
class="collection_meta" dir="ltr">\n          <h2 class="t__h3
c__primary">Course Editor Basics</h2>\n        <p
class="paper__preview">Everything you need to know to get going with our online
course editor.</p>\n      <div class="avatar">\n    <div
class="avatar__photo avatars__images o__ltr">\n      \n\n      \n\n      \n\n    </div>\n    <div
class="avatar__info">\n      <div>\n        <span class="c__darker">\n          5
articles in this collection\n        </span>\n      <br>\n      Written by <span
class='\c__darker\'> Sara Billen,</span> <span class='\c__darker\'> Nick
Carchedi,</span> and <span class='\c__darker\'> Becca Robins</span>\n
</div>\n    </div>\n</div>\n\n    </div>\n    </a>\n
</div>\n    <div class="g__space">\n      <a href="/tips-and-tricks"
class="paper ">\n        <div class="collection o__ltr">\n          <div
class="collection__photo">\n            <svg role='\img\' viewBox='\0 0 48
48\'><g id="comms-mail" stroke-width="2" fill="none" fill-rule="evenodd" stroke-
linejoin="round"><path d="M47 3L1 22l18 7L47 3z"></path><path d="M47 3l-8
37-20-11L47 3zM19 29v16l7-12"></path></g></svg>\n            </div>\n
          <div class="collection_meta" dir="ltr">\n            <h2 class="t__h3
c__primary">Tips & Tricks</h2>\n          <p class="paper__preview">Become
a DataCamp wizard!</p>\n        <div class="avatar">\n      <div
class="avatar__photo avatars__images o__ltr">\n        \n\n      </div>\n      <div class="avatar__info">\n
        <div>\n          <span class="c__darker">\n            6 articles in this collection\n          </span>\n
        <br>\n        Written by <span class='\c__darker\'> Becca Robins</span>\n

```

```

</div>\n </div>\n</div>\n\n </div>\n </div>\n </a>\n
</div>\n <div class="g__space">\n <a href="/frequently-asked-questions-
faq" class="paper ">\n <div class="collection o__ltr">\n <div
class="collection__photo">\n <svg role='img' viewbox='0 0 48
48'><g id="chat-question" fill="none" fill-rule="evenodd"><path d="M47 21.268c0
10.363-10.297 18.765-23 18.765-2.835 0-5.55-.418-8.058-1.184L2.725 45 7.9
34.668c-4.258-3.406-6.9-8.15-6.9-13.4C1 10.904 11.297 2.502 24 2.502s23 8.402 23
18.766z" stroke-width="2" stroke-linejoin="round"></path><path d="M25 28.502a2 2
0 1 0 0 4 2 2 0 0 0 0-4" fill="#231F1F"></path><path d="M19 17.75c0-3.312
2.686-6.124 6-6.124 3.313 0 6 2.626 6 5.938 0 3.315-2.687 5.938-6 5.938V26"
stroke-width="2" stroke-linecap="round" stroke-
linejoin="round"></path></g></svg>\n </div>\n <div
class="collection_meta" dir="ltr">\n <h2 class="t_h3
c__primary">Frequently Asked Questions (FAQ)</h2>\n <p
class="paper__preview">Common questions that arise during content
creation.</p>\n <div class="avatar">\n <div class="avatar__photo
avatars__images o__ltr">\n \n\n \n\n \n\n <span class="avatar__image
avatar__fallback">+3</span>\n </div>\n <div class="avatar__info">\n <div>\n
<span class="c__darker">\n 49 articles in this collection\n
</span>\n <br>\n Written by <span class='c__darker'> Becca
Robins,</span> <span class='c__darker'> Richie Cotton,</span> <span
class='c__darker'> Yashas Roy</span> and 3 others\n </div>\n
</div>\n</div>\n\n </div>\n </div>\n </a>\n </div>\n
<div class="g__space">\n <a href="/miscellaneous" class="paper ">\n
<div class="collection o__ltr">\n <div class="collection__photo">\n
<svg role='img' viewbox='0 0 48 48'><g id="tools-edit"><path d="M14.932
43.968L2 47l3.033-12.93 31.2-31.203a4 4 0 0 1 5.658 0l4.247 4.243a4 4 0 0 1 0
5.656L14.932 43.968zm29.84-29.735L34.82 4.28m7.125 12.782L31.992 7.11M15.436
43.465l-9.9-9.9" stroke-width="2" fill="none" stroke-linecap="round" stroke-
linejoin="round"></path></g></svg>\n </div>\n <div
class="collection_meta" dir="ltr">\n <h2 class="t_h3
c__primary">Miscellaneous</h2>\n <p class="paper__preview">Have a
question for DataCamp, but not about creating content? You&#39;ll probably find
the answer here.</p>\n <div class="avatar">\n <div
class="avatar__photo avatars__images o__ltr">\n \n\n \n\n <img src="https://static.in
tercomassets.com/avatars/2859053/square_128/gabriel_about_pic-1546620603.jpg?154

```



```

6620603" alt="Gabriel de Selding avatar" class="avatar__image">\n\n </div>\n
<div class="avatar__info">\n    <div>\n        <span class="c__darker">\n            9
articles in this collection\n        </span>\n        <br>\n            Written by <span
class=\'c__darker\'> Becca Robins,</span> <span class=\'c__darker\'> Lisa
Monteleone,</span> and <span class=\'c__darker\'> Gabriel de Selding</span>\n
</div>\n    </div>\n</div>\n\n        </div>\n            </div>\n                </a>\n
</div>\n</section>\n</div>\n        </div>\n            <div class="footer">\n    <div
class="container">\n        <div class="content">\n            <div class="u__cf"
dir="ltr">\n                <div class="footer__logo">\n                    <a href="/">\n
\n
</a>\n                </div>\n                <div class="footer__advert logo">\n
\n                    <a
href="https://www.intercom.com/intercom-
link?company=DataCamp&utm_campaign=intercom-
link&utm_content=We+run+on+Intercom&utm_medium=help-
center&utm_referrer=http%3A%2F%2F%2Finstructor-
support.datacamp.com%2F&utm_source=desktop-web">We run on Intercom</a>\n
</div>\n                </div>\n            </div>\n        </div>\n    </div>\n    <script
nonce="CspiV1B0Z1bwrJ1/+YKB2KhncG09YL1YZgZdRzoMqcw=">\n
window.intercomSettings = {"app_id":"ugOps1rq"};\n</script>\n    <script
nonce="CspiV1B0Z1bwrJ1/+YKB2KhncG09YL1YZgZdRzoMqcw=">\n        (function(){var
w=window;var ic=w.Intercom;if(typeof ic==="function"){ic(\'reattach_activator\')
;ic(\'update\',intercomSettings);}else{var d=document;var i=function(){i.c(argum
ents)};i.q=[];i.c=function(args){i.q.push(args)};w.Intercom=i;function l(){var s
=d.createElement(\'script\');s.type=\'text/javascript\';s.async=true;s.src="http
s://widget.intercom.io/widget/ugOps1rq";var x=d.getElementsByTagName(\'script\')
[0];x.parentNode.insertBefore(s,x);}if(w.attachEvent){w.attachEvent(\'onload\',l
);}else{w.addEventListener(\'load\',l,false);}}})();\n</script>\n\n        \n\n
<script src="https://intercom.help/_assets/application-047e02b2f0406f381c4779484
1e08df22bf3730124f3e2cc05f8b409a1fc5cf7.js"
nonce="CspiV1B0Z1bwrJ1/+YKB2KhncG09YL1YZgZdRzoMqcw="></script>\n
</body>\n</html>\n'

```

2.1.6 4.6 Performing HTTP requests in Python using requests

Now that you've got your head and hands around making HTTP requests using the `urllib` package, you're going to figure out how to do the same using the higher-level `requests` library. You'll once again be pinging DataCamp servers for their `"http://www.datacamp.com/teach/documentation"` page.

Note that unlike in the previous exercises using `urllib`, you don't have to close the connection when using `requests`!

Instructions

- Import the package `requests`.
- Assign the URL of interest to the variable `url`.

- Package the request to the URL, send the request and catch the response with a single function `requests.get()`, assigning the response to the variable `r`.
- Use the text attribute of the object `r` to return the HTML of the webpage as a string; store the result in a variable `text`.
- Hit submit to print the HTML of the webpage.

```
[8]: # 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)
```

```
<!doctype html>
<html lang="en" data-direction="ltr">
  <head>
    <link href="https://fonts.intercomcdn.com" rel="preconnect" crossorigin>
    <script src="https://www.googletagmanager.com/gtag/js?id=UA-39297847-9"
    async="async" nonce="wDbhkbBsQYjiB+z/aLpZzMGg1ZPULUJojJUxe9vbs2s="></script>
    <script nonce="wDbhkbBsQYjiB+z/aLpZzMGg1ZPULUJojJUxe9vbs2s=">
      window.dataLayer = window.dataLayer || [];
      function gtag(){dataLayer.push(arguments);}
      gtag('js', new Date());
      gtag('config', 'UA-39297847-9');
    </script>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>DataCamp Help Center</title>
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="intercom:trackingEvent"
    content="{&quot;name&quot;:&quot;Viewed Help Center&quot;,&quot;metadata&quot;:{
    &quot;action&quot;:&quot;viewed&quot;,&quot;object&quot;:&quot;educate_home&quot;
    ;,&quot;place&quot;:&quot;help_center&quot;,&quot;owner&quot;:&quot;educate&quot;
    ;}}"/>
    <link rel="stylesheet" media="all" href="https://intercom.help/_assets/appli
    cation-c71cce033306dfc55f25ff3f87dd3278ec16161813e551727a60f18a92811bf6.css" />
    <link rel="canonical" href="http://instructor-support.datacamp.com/">
```

```

    <link href="https://static.intercomassets.com/assets/educate/educate-fav
icon-64x64-at-2x-52016a3500a250d0b118c0a04ddd13b1a7364a27759483536dd1940bccdefc2
0.png" rel="shortcut icon" type="image/png" />
    <style>
        .header, .avatar__image-extra { background-color: #263e63; }
        .article a, .c__primary { color: #263e63; }
        .avatar__fallback { background-color: #263e63; }
        article a.intercom-h2b-button { background-color: #263e63; border: 0; }
    </style>

    <meta property="og:title" content="DataCamp Help Center" />
    <meta name="twitter:title" content="DataCamp Help Center" />

<meta property="og:type" content="website" />
<meta property="og:image" content="" />

<meta name="twitter:image" content="" />

</head>
<body class="">
    <header class="header">
    <div class="container header__container o__ltr" dir="ltr">
        <div class="content">
            <div class="mo o__centered o__reversed header__meta_wrapper">
                <div class="mo__body">
                    <div class="header__logo">
                        <a href="/">
                            
                        </a>
                    </div>
                </div>
            </div>
            <div class="mo__aside">
                <div class="header__home__url">

                    <a target="_blank" rel='noopener'
href="http://www.datacamp.com/teach"><svg width="14" height="14" viewBox="0 0 14
14" xmlns="http://www.w3.org/2000/svg"><title>Group 65</title><g stroke="#FFF"
fill="none" fill-rule="evenodd" stroke-linecap="round" stroke-
linejoin="round"><path d="M11.5 6.73v6.77H.5v-11h7.615M4.5 9.517-7M13.5
5.5v-5h-5"/></g></svg><span>Go to DataCamp</span></a>
                </div>
            </div>
        </div>
        <h1 class="header__headline">Advice and answers from the DataCamp
Team</h1>

```

```

    <form action="/" autocomplete="off" class="header__form search">
      <input type="text" autocomplete="off" class="search__input js__search-
input o__ltr" placeholder="Search for articles..." tabindex="1" name="q"
value="">
      <div class="search_icons">
        <button type="submit" class="search__submit o__ltr"></button>
        <a class="search__clear-text__icon">
          <svg class="interface-icon" xmlns="http://www.w3.org/2000/svg"
width="16" height="16" viewBox="0 0 16 16">
            <path d="M8.018 6.643L5.375 4 4 5.375L2.643 2.643L4 10.643 5.375
12L2.643-2.625L10.625 12 12 10.643 9.357 8.018 12 5.375 10.643 4z" />
          </svg>
        </a>
      </form>
    </div>
  </div>
</div>
</header>

```

```

<div class="container">
  <div class="content educate_content"><section class="section">
    <div class="g__space">
      <a href="/getting-started" class="paper ">
        <div class="collection o__ltr">
          <div class="collection__photo">
            <svg role='img' viewBox='0 0 48 48'><g id="chat-star" stroke-
width="2" fill="none" fill-rule="evenodd" stroke-linejoin="round"><path d="M20
34.942c-2.083-.12-4.292-.42-6-.942L3 39L4-9c-3.858-3.086-6-7.246-6-12C1 8.61
10.328 1 21.835 1 33.343 1 43 8.61 43 18c0 1.044-.117 2.065-.342
3.057"></path><path d="M36.016 25L40 33h7L6 5 3 9-8-5.494L28
47L3-9-6-5h7L4.016-8z"></path></g></svg>
          </div>
          <div class="collection_meta" dir="ltr">
            <h2 class="t__h3 c__primary">Getting Started</h2>
            <p class="paper__preview">Everything you need to know to begin your
DataCamp journey!</p>
            <div class="avatar">
              <div class="avatar__photo avatars__images o__ltr">
                

                
              </div>
            </div>
          </div>
        </div>
      </a>
    </div>
  </div>

```

```

        <span class="avatar__image avatar__fallback">+2</span>
    </div>
    <div class="avatar__info">
        <div>
            <span class="c__darker">
                11 articles in this collection
            </span>
            <br>
            Written by <span class='c__darker'> Becca Robins,</span> <span
class='c__darker'> Jen Bricker,</span> <span class='c__darker'> Yashas
Roy</span> and 2 others
        </div>
    </div>
</div>

    </div>
</div>
</a>
</div>
<div class="g__space">
    <a href="/courses" class="paper ">
        <div class="collection o__ltr">
            <div class="collection__photo">
                <svg role='img' viewBox='0 0 48 48'><g id="devices-laptop" stroke-
width="2" fill="none" fill-rule="evenodd" stroke-linecap="round"><path d="M41
31H7V11h34v20z"></path><path d="M3 35V10a3 3 0 0 1 3-3h36a3 3 0 0 1 3 3v25m-16
0v2H19v-2H1v4a2 2 0 0 0 2 2h42a2 2 0 0 0 2-2v-4H29z" stroke-
linejoin="round"></path></g></svg>
            </div>
            <div class="collection_meta" dir="ltr">
                <h2 class="t__h3 c__primary">Courses</h2>
                <p class="paper__preview">Everything you need to know about creating
DataCamp courses.</p>
                <div class="avatar">
                    <div class="avatar__photo avatars__images o__ltr">
                        

```

```

    <span class="avatar__image avatar__fallback">+10</span>
  </div>
  <div class="avatar__info">
    <div>
      <span class="c__darker">
        84 articles in this collection
      </span>
      <br>
      Written by <span class='c__darker'> Yashas Roy,</span> <span
class='c__darker'> Nick Carchedi,</span> <span class='c__darker'> Richie
Cotton</span> and 10 others
    </div>
  </div>
</div>

    </div>
  </div>
</a>
</div>
<div class="g__space">
  <a href="/daily-practice" class="paper ">
    <div class="collection o__ltr">
      <div class="collection__photo">
        <svg role='img' viewBox='0 0 48 48'><g id="tools-dashboard" stroke-
width="2" fill="none" fill-rule="evenodd" stroke-linecap="round" stroke-
linejoin="round"><path d="M27 31a3 3 0 0 1-6 0 3 3 0 0 1 6
0zm-.88-2.12l9.9-9.9M5 32h4m34 .002L39 32m2.553-8.27l-3.696 1.53M31.27
13.447l-1.53 3.695M24 12v4m-7.27-2.553l1.53 3.695m-7.694.422l12.826 2.83M6.447
23.73l13.695 1.53"></path><path d="M24 8C11.297 8 1 18.3 1 31v9h46v-9C47 18.3
36.703 8 24 8z"></path></g></svg>
      </div>
      <div class="collection_meta" dir="ltr">
        <h2 class="t__h3 c__primary">Daily Practice</h2>
        <p class="paper__preview">Everything you need to know about creating
DataCamp Daily Practice.</p>
        <div class="avatar">
          <div class="avatar__photo avatars__images o__ltr">
            
          </div>
        </div>
        <div class="avatar__info">
          <div>
            <span class="c__darker">
              15 articles in this collection
            </span>

```

```

        <br>
        Written by <span class='c__darker'> Anneleen Beckers</span>
    </div>
</div>
</div>

    </div>
</div>
</a>
</div>
<div class="g__space">
    <a href="/projects" class="paper ">
        <div class="collection o__ltr">
            <div class="collection__photo">
                <svg role='img' viewBox='0 0 48 48'><g id="book-opened2"><path
d="M24 11c0-3.866 10.297-7 23-7v33c-12.703 0-23 3.134-23 7
0-3.866-10.3-7-23-7V4c12.7 0 23 3.134 23 7zm0
0v32m-5-27.52c-3.22-1.232-7.773-2.128-13-2.48m13
8.48c-3.22-1.232-7.773-2.128-13-2.48m13 8.48c-3.22-1.232-7.773-2.128-13-2.48m13
8.48c-3.22-1.23-7.773-2.127-13-2.48m23-15.52c3.223-1.232 7.773-2.128 13-2.48m-13
8.48c3.223-1.232 7.773-2.128 13-2.48m-13 8.48c3.223-1.232 7.773-2.128
13-2.48m-13 8.48c3.223-1.23 7.773-2.127 13-2.48" stroke-width="2" fill="none"
stroke-linecap="round" stroke-linejoin="round"></path></g></svg>
            </div>
            <div class="collection_meta" dir="ltr">
                <h2 class="t__h3 c__primary">Projects</h2>
                <p class="paper__preview">Everything you need to know about creating
DataCamp projects.</p>
                <div class="avatar">
                    <div class="avatar__photo avatars__images o__ltr">
                        
                    </div>
                    <div class="avatar__info">
                        <div>
                            <span class="c__darker">
                                19 articles in this collection
                            </span>
                            <br>
                            Written by <span class='c__darker'> David Venturi</span>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

    </div>
</div>

```

```

    </a>
  </div>
  <div class="g__space">
    <a href="/course-editor-basics" class="paper ">
      <div class="collection o__ltr">
        <div class="collection__photo">
          <svg role='img' viewBox='0 0 48 48'><g id="book-bookmark" stroke-
width="2" fill="none" fill-rule="evenodd" stroke-linecap="round"><path d="M35
31l-6-6-6 6V7h12v24z"></path><path d="M35 9h6v38H11a4 4 0 0 1-4-4V5" stroke-
linejoin="round"></path><path d="M39 9V1H11a4 4 0 0 0 0 8h12" stroke-
linejoin="round"></path></g></svg>
        </div>
        <div class="collection_meta" dir="ltr">
          <h2 class="t__h3 c__primary">Course Editor Basics</h2>
          <p class="paper__preview">Everything you need to know to get going
with our online course editor.</p>
          <div class="avatar">
            <div class="avatar__photo avatars__images o__ltr">
              

            </div>
            <div class="avatar__info">
              <div>
                <span class="c__darker">
                  5 articles in this collection
                </span>
                <br>
                Written by <span class='c__darker'> Sara Billen,</span> <span
class='c__darker'> Nick Carchedi,</span> and <span class='c__darker'> Becca
Robins</span>
              </div>
            </div>
          </div>
        </div>
      </a>
    </div>
  </div>

```



```

<div class="g__space">
  <a href="/tips-and-tricks" class="paper ">
    <div class="collection o__ltr">
      <div class="collection__photo">
        <svg role='img' viewBox='0 0 48 48'><g id="comms-mail" stroke-
width="2" fill="none" fill-rule="evenodd" stroke-linejoin="round"><path d="M47
3L1 22L18 7L47 3z"></path><path d="M47 31-8 37-20-11L47 3zM19
29v16L17-12"></path></g></svg>
      </div>
      <div class="collection_meta" dir="ltr">
        <h2 class="t__h3 c__primary">Tips & Tricks</h2>
        <p class="paper__preview">Become a DataCamp wizard!</p>
        <div class="avatar">
          <div class="avatar__photo avatars__images o__ltr">
            
          </div>
          <div class="avatar__info">
            <div>
              <span class="c__darker">
                6 articles in this collection
              </span>
              <br>
              Written by <span class='c__darker'> Becca Robins</span>
            </div>
          </div>
        </div>
      </div>
    </a>
  </div>
  <div class="g__space">
    <a href="/frequently-asked-questions-faq" class="paper ">
      <div class="collection o__ltr">
        <div class="collection__photo">
          <svg role='img' viewBox='0 0 48 48'><g id="chat-question"
fill="none" fill-rule="evenodd"><path d="M47 21.268c0 10.363-10.297 18.765-23
18.765-2.835 0-5.55-.418-8.058-1.184L2.725 45 7.9
34.668c-4.258-3.406-6.9-8.15-6.9-13.4C1 10.904 11.297 2.502 24 2.502s23 8.402 23
18.766z" stroke-width="2" stroke-linejoin="round"></path><path d="M25 28.502a2 2
0 1 0 0 4 2 2 0 0 0 0-4" fill="#231F1F"></path><path d="M19 17.75c0-3.312
2.686-6.124 6-6.124 3.313 0 6 2.626 6 5.938 0 3.315-2.687 5.938-6 5.938V26"
stroke-width="2" stroke-linecap="round" stroke-
linejoin="round"></path></g></svg>
        </div>
      </div>
    </a>
  </div>

```

```

        <div class="collection_meta" dir="ltr">
            <h2 class="t_h3 c_primary">Frequently Asked Questions (FAQ)</h2>
            <p class="paper__preview">Common questions that arise during content
creation.</p>
            <div class="avatar">
                <div class="avatar__photo avatars__images o__ltr">
                    

                    <span class="avatar__image avatar__fallback">+3</span>
                </div>
                <div class="avatar__info">
                    <div>
                        <span class="c__darker">
                            49 articles in this collection
                        </span>
                        <br>
                        Written by <span class='c__darker'> Becca Robins,</span> <span
class='c__darker'> Richie Cotton,</span> <span class='c__darker'> Yashas
Roy</span> and 3 others
                    </div>
                </div>
            </div>

        </div>
    </div>
</a>
</div>
<div class="g__space">
    <a href="/miscellaneous" class="paper ">
        <div class="collection o__ltr">
            <div class="collection__photo">
                <svg role='img' viewBox='0 0 48 48'><g id="tools-edit"><path
d="M14.932 43.968L2 47L3.033-12.93 31.2-31.203a4 4 0 0 1 5.658 014.247 4.243a4 4
0 0 1 0 5.656L14.932 43.968zm29.84-29.735L34.82 4.28m7.125 12.782L31.992
7.11M15.436 43.465l-9.9-9.9" stroke-width="2" fill="none" stroke-linecap="round"
stroke-linejoin="round"></path></g></svg>
            </div>

```

```

        <div class="collection_meta" dir="ltr">
            <h2 class="t_h3 c_primary">Miscellaneous</h2>
            <p class="paper__preview">Have a question for DataCamp, but not
about creating content? You&#39;ll probably find the answer here.</p>
            <div class="avatar">
                <div class="avatar__photo avatars__images o_ltr">
                    

                </div>
                <div class="avatar__info">
                    <div>
                        <span class="c__darker">
                            9 articles in this collection
                        </span>
                        <br>
                        Written by <span class='c__darker'> Becca Robins,</span> <span
class='c__darker'> Lisa Monteleone,</span> and <span class='c__darker'> Gabriel
de Selding</span>
                    </div>
                </div>
            </div>
        </div>
    </div>
</a>
</div>
</section>
</div>
</div>
<footer class="footer">
<div class="container">
    <div class="content">
        <div class="u_cf" dir="ltr">
            <div class="footer__logo">
                <a href="/">
                    
                </a>
            </div>
        </div>
    </div>
</div>

```

```

</div>
  <div class="footer__advert logo">
    
    <a href="https://www.intercom.com/intercom-
link?company=DataCamp&utm_content=customer-support&utm_campaign=intercom-
link&utm_medium=help-
center&utm_referrer=http%3A%2F%2Finstructor-
support.datacamp.com%2F&utm_source=desktop-web">We run on Intercom</a>
  </div>
</div>
</div>
</div>
</footer>

```

2.1.7 4.7 Parsing HTML with BeautifulSoup

Instructions

- Import the function BeautifulSoup from the package bs4.
- Assign the URL of interest to the variable url.
- Package the request to the URL, send the request and catch the response with a single function requests.get(), assigning the response to the variable r.
- Use the text attribute of the object r to return the HTML of the webpage as a string; store the result in a variable html_doc.
- Create a BeautifulSoup object soup from the resulting HTML using the function BeautifulSoup().
- Use the method prettify() on soup and assign the result to pretty_soup.
- Hit submit to print to prettified HTML to your shell!

```
[11]: # 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)
```

```
<html>
<head>
  <title>
    Guido's Personal Home Page
  </title>
</head>
<body bgcolor="#FFFFFF" text="#000000">
  <h1>
    <a href="pics.html">
      
    </a>
    Guido van Rossum - Personal Home Page
    <a href="pics.html">
      
```

```

    </a>
</h1>
<p>
    <a href="http://www.washingtonpost.com/wp-
    srv/business/longterm/microsoft/stories/1998/raymond120398.htm">
        <i>
            "Gawky and proud of it."
        </i>
    </a>
<h3>
    <a href="http://metalab.unc.edu/Dave/Dr-Fun/df200004/df20000406.jpg">
        Who
I Am
    </a>
</h3>
<p>
    Read
my
    <a href="http://neopythonic.blogspot.com/2016/04/kings-day-speech.html">
        "King's
Day Speech"
    </a>
    for some inspiration.
<p>
    I am the author of the
    <a href="http://www.python.org">
        Python
    </a>
    programming language. See also my
    <a href="Resume.html">
        resume
    </a>
    and my
    <a href="Publications.html">
        publications list
    </a>
    , a
    <a href="bio.html">
        brief bio
    </a>
    , assorted
    <a href="http://legacy.python.org/doc/essays/">
        writings
    </a>
    ,
    <a href="http://legacy.python.org/doc/essays/ppt/">
        presentations
    </a>

```

```

and
<a href="interviews.html">
    interviews
</a>
(all about Python), some
<a href="pics.html">
    pictures of me
</a>
,
<a href="http://neopythonic.blogspot.com">
    my new blog
</a>
, and
my
<a href="http://www.artima.com/weblogs/index.jsp?blogger=12088">
    old
blog
</a>
on Artima.com. I am
<a href="https://twitter.com/gvanrossum">
    @gvanrossum
</a>
on Twitter.
<p>
    In January 2013 I joined
    <a href="http://www.dropbox.com">
        Dropbox
    </a>
    . I work on various Dropbox
products and have 50% for my Python work, no strings attached.
Previously, I have worked for Google, Elemental Security, Zope
Corporation, BeOpen.com, CNRI, CWI, and SARA. (See
my
    <a href="Resume.html">
        resume
    </a>
    .) I created Python while at CWI.
<h3>
    How to Reach Me
</h3>
<p>
    You can send email for me to guido (at) python.org.
I read everything sent there, but if you ask
me a question about using Python, it's likely that I won't have time
to answer it, and will instead refer you to
help (at) python.org,
    <a href="http://groups.google.com/groups?q=comp.lang.python">
        comp.lang.python

```


 or
 [
 . If you need to
 talk to me on the phone or send me something by snail mail, send me an
 email and I'll gladly email you instructions on how to reach me.
 <h3>
 My Name
 </h3>
 <p>
 My name often poses difficulties for Americans.
 <p>

 Pronunciation:

 in Dutch, the "G" in Guido is a hard G,
 pronounced roughly like the "ch" in Scottish "loch". \(Listen to the

 sound clip

 .\) However, if you're
 American, you may also pronounce it as the Italian "Guido". I'm not
 too worried about the associations with mob assassins that some people
 have. :-\)
 <p>

 Spelling:

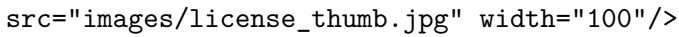
 my last name is two words, and I'd like to keep it
 that way, the spelling on some of my credit cards notwithstanding.
 Dutch spelling rules dictate that when used in combination with my
 first name, "van" is not capitalized: "Guido van Rossum". But when my
 last name is used alone to refer to me, it is capitalized, for
 example: "As usual, Van Rossum was right."
 <p>

 Alphabetization:

 in America, I show up in the alphabet under
 "V". But in Europe, I show up under "R". And some of my friends put
 me under "G" in their address book...
 <h3>
 More Hyperlinks
 </h3>

](http://stackoverflow.com)

Here's a collection of
[essays](http://legacy.python.org/doc/essays/)
relating to Python
that I've written, including the foreword I wrote for Mark Lutz' book
"Programming Python".

- I own the official
[!\[\]\(a3ea015cc5581cad732d1eb81613fe7b_img.jpg\)](images/license.jpg)

Python license.

The Audio File Formats FAQ

I was the original creator and maintainer of the Audio File Formats
FAQ. It is now maintained by Chris Bagwell
at
<http://www.cnpbagwell.com/audio-faq>
. And here is a link to
[SOX](http://sox.sourceforge.net/)
, to which I contributed
some early code.

```
</html>
<hr/>
<a href="images/internetdog.gif">
  "On the Internet, nobody knows you're
a dog."
</a>
<hr/>
```

2.1.8 4.8 Turning a webpage into data using BeautifulSoup: getting the text

As promised, in the following exercises, you'll learn the basics of extracting information from HTML soup. In this exercise, you'll figure out how to extract the text from the BDFL's webpage, along with printing the webpage's title.

Instructions

- In the sample code, the HTML response object `html_doc` has already been created: your first task is to Soupify it using the function `BeautifulSoup()` and to assign the resulting soup to the variable `soup`.
- Extract the title from the HTML soup `soup` using the attribute `title` and assign the result to `guido_title`.
- Print the title of Guido's webpage to the shell using the `print()` function.
- Extract the text from the HTML soup `soup` using the method `get_text()` and assign to `guido_text`.
- Hit submit to print the text from Guido's webpage to the shell.

```
[12]: # 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)
```

<title>Guido's Personal Home Page</title>

Guido's Personal Home Page

Guido van Rossum - Personal Home Page

"Gawky and proud of it."

Who

I Am

Read

my "King's

Day Speech" for some inspiration.

I am the author of the Python

programming language. See also my resume

and my publications list, a brief bio, assorted writings, presentations and interviews (all about Python), some

pictures of me,

my new blog, and

my old

blog on Artima.com. I am

@gvanrossum on Twitter.

In January 2013 I joined

Dropbox. I work on various Dropbox

products and have 50% for my Python work, no strings attached.

Previously, I have worked for Google, Elemental Security, Zope Corporation, BeOpen.com, CNRI, CWI, and SARA. (See my resume.) I created Python while at CWI.

How to Reach Me

You can send email for me to guido (at) python.org.

I read everything sent there, but if you ask

me a question about using Python, it's likely that I won't have time to answer it, and will instead refer you to

help (at) python.org,

comp.lang.python or
StackOverflow. If you need to
talk to me on the phone or send me something by snail mail, send me an
email and I'll gladly email you instructions on how to reach me.

My Name

My name often poses difficulties for Americans.

Pronunciation: in Dutch, the "G" in Guido is a hard G,
pronounced roughly like the "ch" in Scottish "loch". (Listen to the
sound clip.) However, if you're
American, you may also pronounce it as the Italian "Guido". I'm not
too worried about the associations with mob assassins that some people
have. :-)

Spelling: my last name is two words, and I'd like to keep it
that way, the spelling on some of my credit cards notwithstanding.
Dutch spelling rules dictate that when used in combination with my
first name, "van" is not capitalized: "Guido van Rossum". But when my
last name is used alone to refer to me, it is capitalized, for
example: "As usual, Van Rossum was right."

Alphabetization: in America, I show up in the alphabet under
"V". But in Europe, I show up under "R". And some of my friends put
me under "G" in their address book...

More Hyperlinks

Here's a collection of essays relating to Python
that I've written, including the foreword I wrote for Mark Lutz' book
"Programming Python".
I own the official
Python license.

The Audio File Formats FAQ

I was the original creator and maintainer of the Audio File Formats
FAQ. It is now maintained by Chris Bagwell
at <http://www.cnpbagwell.com/audio-faq>. And here is a link to
SOX, to which I contributed
some early code.

"On the Internet, nobody knows you're
a dog."

2.1.9 4.9 Turning a webpage into data using BeautifulSoup: getting the hyperlinks

In this exercise, you'll figure out how to extract the URLs of the hyperlinks from the BDFL's webpage. In the process, you'll become close friends with the soup method `find_all()`.

Instructions

- Use the method `find_all()` to find all hyperlinks in `soup`, remembering that hyperlinks are defined by the HTML tag `<a>` but passed to `find_all()` without angle brackets; store the result in the variable `a_tags`.
- The variable `a_tags` is a results set: your job now is to enumerate over it, using a `for` loop and to print the actual URLs of the hyperlinks; to do this, for every element `link` in `a_tags`, you want to `print()` `link.get('href')`.

```
[13]: # 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.findAll('a')

# Print the URLs to the shell
for link in a_tags:
    print(link.get('href'))
```

```
<title>Guido's Personal Home Page</title>
pics.html
pics.html
http://www.washingtonpost.com/wp-
srv/business/longterm/microsoft/stories/1998/raymond120398.htm
http://metalab.unc.edu/Dave/Dr-Fun/df200004/df20000406.jpg
http://neopythonic.blogspot.com/2016/04/kings-day-speech.html
```

```
http://www.python.org
Resume.html
Publications.html
bio.html
http://legacy.python.org/doc/essays/
http://legacy.python.org/doc/essays/ppt/
interviews.html
pics.html
http://neopythonic.blogspot.com
http://www.artima.com/weblogs/index.jsp?blogger=12088
https://twitter.com/gvanrossum
http://www.dropbox.com
Resume.html
http://groups.google.com/groups?q=comp.lang.python
http://stackoverflow.com
guido.au
http://legacy.python.org/doc/essays/
images/license.jpg
http://www.cnpbagwell.com/audio-faq
http://sox.sourceforge.net/
images/internetdog.gif
```

2.1.10 5. Interacting with APIs to import data from the web

In this chapter, you will push further on your knowledge of importing data from the web. You will learn the basics of extracting data from APIs, gain insight on the importance of APIs and practice getting data from them with dives into the OMDb and Library of Congress APIs.

2.1.11 5.1 Loading and exploring a JSON

Now that you know what a JSON is, you'll load one into your Python environment and explore it yourself. Here, you'll load the JSON 'a_movie.json' into the variable `json_data`, which will be a dictionary. You'll then explore the JSON contents by printing the key-value pairs of `json_data` to the shell.

Instructions

- Load the JSON 'a_movie.json' into the variable `json_data` within the context provided by the `with` statement. To do so, use the function `json.load()` within the context manager.
- Use a `for` loop to print all key-value pairs in the dictionary `json_data`. Recall that you can access a value in a dictionary using the syntax: `dictionary[key]`.

```
[9]: import json
# Load JSON: json_data
#with open("./Data/a_movie.json") as json_file:
with open("./Data/example_2.json") as json_file:
    json_data = json.load(json_file)

# Print each key-value pair in json_data
```

```
for k in json_data.keys():
    print(k + ': ', json_data[k])
```

```
quiz: {'sport': {'q1': {'question': 'Which one is correct team name in NBA?',
'options': ['New York Bulls', 'Los Angeles Kings', 'Golden State Warriros',
'Huston Rocket'], 'answer': 'Huston Rocket'}}}, 'maths': {'q1': {'question': '5 +
7 = ?', 'options': ['10', '11', '12', '13'], 'answer': '12'}, 'q2': {'question':
'12 - 8 = ?', 'options': ['1', '2', '3', '4'], 'answer': '4'}}
```

2.1.12 5.2 API requests

Now it's your turn to pull some movie data down from the Open Movie Database (OMDB) using their API. The movie you'll query the API about is The Social Network. Recall that, in the video, to query the API about the movie Hackers, Hugo's query string was 'http://www.omdbapi.com/?t=hackers' and had a single argument `t=hackers`.

Note: recently, OMDB has changed their API: you now also have to specify an API key. This means you'll have to add another argument to the URL: `apikey=72bc447a`.

Instructions

- Import the `requests` package.
- Assign to the variable `url` the URL of interest in order to query 'http://www.omdbapi.com' for the data corresponding to the movie The Social Network. The query string should have two arguments: `apikey=72bc447a` and `t=the+social+network`. You can combine them as follows: `apikey=72bc447a&t=the+social+network`.
- Print the text of the reponse object `r` by using its `text` attribute and passing the result to the `print()` function.

```
[10]: # 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)
```

```
{"Title": "The Social Network", "Year": "2010", "Rated": "PG-13", "Released": "01 Oct
2010", "Runtime": "120 min", "Genre": "Biography, Drama", "Director": "David
Fincher", "Writer": "Aaron Sorkin (screenplay), Ben Mezrich
(book)", "Actors": "Jesse Eisenberg, Rooney Mara, Bryan Barter, Dustin
Fitzsimons", "Plot": "Harvard student Mark Zuckerberg creates the social
networking site. That would become known as Facebook but is later sued by two
brothers who claimed he stole their idea, and the co-founder who was later
squeezed out of the business.", "Language": "English,
French", "Country": "USA", "Awards": "Won 3 Oscars. Another 165 wins & 168
```

```

nominations.", "Poster": "https://m.media-amazon.com/images/M/MV5B0GUyZDUxZjEtMmIz
MCO0MzlmLTg4MGltZWJmMzBhZjEOMjc1XkEyXkFqcGdeQXVyMTMxODk2OTU@._V1_SX300.jpg", "Rat
ings": [{"Source": "Internet Movie Database", "Value": "7.7/10"}, {"Source": "Rotten T
omatoes", "Value": "95%"}, {"Source": "Metacritic", "Value": "95/100"}], "Metascore": "9
5", "imdbRating": "7.7", "imdbVotes": "567,207", "imdbID": "tt1285016", "Type": "movie",
"DVD": "11 Jan 2011", "BoxOffice": "$96,400,000", "Production": "Columbia
Pictures", "Website": "http://www.thesocialnetwork-movie.com/", "Response": "True"}

```

2.1.13 5.3 JSON—from the web to Python

Wow, congrats! You've just queried your first API programmatically in Python and printed the text of the response to the shell. However, as you know, your response is actually a JSON, so you can do one step better and decode the JSON. You can then print the key-value pairs of the resulting dictionary. That's what you're going to do now!

Instructions

- Pass the variable `url` to the `requests.get()` function in order to send the relevant request and catch the response, assigning the resultant response message to the variable `r`.
- Apply the `json()` method to the response object `r` and store the resulting dictionary in the variable `json_data`.
- Hit Submit Answer to print the key-value pairs of the dictionary `json_data` to the shell.

```

[11]: # 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])

```

```

Title: The Social Network
Year: 2010
Rated: PG-13
Released: 01 Oct 2010
Runtime: 120 min
Genre: Biography, Drama
Director: David Fincher
Writer: Aaron Sorkin (screenplay), Ben Mezrich (book)
Actors: Jesse Eisenberg, Rooney Mara, Bryan Barter, Dustin Fitzsimons
Plot: Harvard student Mark Zuckerberg creates the social networking site. That

```


would become known as Facebook but is later sued by two brothers who claimed he stole their idea, and the co-founder who was later squeezed out of the business.

Language: English, French

Country: USA

Awards: Won 3 Oscars. Another 165 wins & 168 nominations.

Poster: https://m.media-amazon.com/images/M/MV5B0GUyZDUxZjEtMmIzMCO0MzlmLTg4MGItZWJmMzBhZjEOMjc1XkEyXkFqcGdeQXVyMTMxODk2OTU@._V1_SX300.jpg

Ratings: [{'Source': 'Internet Movie Database', 'Value': '7.7/10'}, {'Source': 'Rotten Tomatoes', 'Value': '95%'}, {'Source': 'Metacritic', 'Value': '95/100'}]

Metascore: 95

imdbRating: 7.7

imdbVotes: 567,207

imdbID: tt1285016

Type: movie

DVD: 11 Jan 2011

BoxOffice: \$96,400,000

Production: Columbia Pictures

Website: <http://www.thesocialnetwork-movie.com/>

Response: True

2.1.14 5.4 Checking out the Wikipedia API

You're doing so well and having so much fun that we're going to throw one more API at you: the Wikipedia API (documented [here](#)). You'll figure out how to find and extract information from the Wikipedia page for Pizza. What gets a bit wild here is that your query will return nested JSONs, that is, JSONs with JSONs, but Python can handle that because it will translate them into dictionaries within dictionaries.

The URL that requests the relevant query from the Wikipedia API is

~~ <https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=pizza> ~~ **In-**
structions

- Assign the relevant URL to the variable `url`.
- Apply the `json()` method to the response object `r` and store the resulting dictionary in the variable `json_data`.
- The variable `pizza_extract` holds the HTML of an extract from Wikipedia's Pizza page as a string; use the function `print()` to print this string to the shell.

```
[12]: # Import package
import requests

# Assign URL to variable: url
url = 'https://en.wikipedia.org/w/api.php?
      ↪action=query&prop=extracts&format=json&exintro=&titles=pizza'

# 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 the Wikipedia page extract
pizza_extract = json_data['query']['pages']['24768']['extract']
print(pizza_extract)

```

```

<p class="mw-empty-elt">
</p>

```

```

<p><b>Pizza</b> (<small>Italian: </small><span title="Representation in the
International Phonetic Alphabet (IPA)">[pittsa]</span>,
<small>Neapolitan: </small><span title="Representation in the International
Phonetic Alphabet (IPA)">[pittsə]</span>) is a savory dish of Italian origin,
consisting of a usually round, flattened base of leavened wheat-based dough
topped with tomatoes, cheese, and various other ingredients (anchovies, olives,
meat, etc.) baked at a high temperature, traditionally in a wood-fired oven. In
formal settings, like a restaurant, pizza is eaten with knife and fork, but in
casual settings it is cut into wedges to be eaten while held in the hand. Small
pizzas are sometimes called pizzettas.
</p><p>The term <i>pizza</i> was first recorded in the 10th century in a Latin
manuscript from the Southern Italian town of Gaeta in Lazio, on the border with
Campania. Modern pizza was invented in Naples, and the dish and its variants
have since become popular in many countries. It has become one of the most
popular foods in the world and a common fast food item in Europe and North
America, available at pizzerias (restaurants specializing in pizza),
restaurants offering Mediterranean cuisine, and via pizza delivery. Many
companies sell ready-baked frozen pizzas to be reheated in an ordinary home
oven.
</p><p>The <i>Associazione Verace Pizza Napoletana</i> (lit. True Neapolitan
Pizza Association) is a non-profit organization founded in 1984 with
headquarters in Naples that aims to promote traditional Neapolitan pizza. In
2009, upon Italy's request, Neapolitan pizza was registered with the European
Union as a Traditional Speciality Guaranteed dish, and in 2017 the art of its
making was included on UNESCO's list of intangible cultural heritage.</p>

```

2.2 6. Diving deep into the Twitter API

In this chapter, you will consolidate your knowledge of interacting with APIs in a deep dive into the Twitter streaming API. You'll learn how to stream real-time Twitter data and to analyze and visualize it!

2.2.1 6.1 API Authentication

The package `tweepy` is great at handling all the Twitter API OAuth Authentication details for you. All you need to do is pass it your authentication credentials. In this interactive exercise, we have created some mock authentication credentials (if you wanted to replicate this at home, you would need to create a Twitter App as Hugo detailed in the video). Your task is to pass these credentials to `tweepy`'s OAuth handler.

Instructions

- Import the package `tweepy`.
- Pass the parameters `consumer_key` and `consumer_secret` to the function `tweepy.OAuthHandler()`.
- 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`.

```
[30]: # Import package
import tweepy

# Store OAuth authentication credentials in relevant variables
access_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fF08k8EPy"
access_token_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"
consumer_key = "nZ6EA0FxZ293SxGNg8g8aPOHM"
consumer_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"

# Pass OAuth details to tweepy's OAuth handler
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
```

2.2.2 6.2 Streaming tweets

Now that you have set up your authentication credentials, it is time to stream some tweets! We have already defined the tweet stream listener class, `MyStreamListener`, just as Hugo did in the introductory video. You can find the code for the tweet stream listener class here.

Your task is to create the `Streamobject` and to filter tweets according to particular keywords.

Instructions

- Create your `Stream` object with authentication by passing `tweepy.Stream()` the authentication handler `auth` and the `Stream` listener `l`;
- To filter Twitter streams, pass to the `track` argument in `stream.filter()` a list containing the desired keywords 'clinton', 'trump', 'sanderson', and 'cruz'.

```
[36]: # Import package
import tweepy
import json

class MyStreamListener(tweepy.StreamListener):
    def __init__(self, api=None):
        super(MyStreamListener, self).__init__()
        self.num_tweets = 0
        self.file = open("tweets.txt", "w")
    def on_status(self, status):
        tweet = status._json
        self.file.write(json.dumps(tweet) + '\n')
        tweet_list.append(status)
```

```

        self.num_tweets += 1
        if self.num_tweets < 100:
            return True
        else:
            return False
        self.file.close()

# Store OAuth authentication credentials in relevant variables (these are ↵
↵DataCamp tokens)
access_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fF08k8EPy"
access_token_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"
consumer_key = "nZ6EA0FxZ293SxGNg8g8aPOHM"
consumer_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"

# Pass OAuth details to tweepy's OAuth handler
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

# Initialize Stream listener
l = MyStreamListener()

# Create you Stream object with authentication
stream = tweepy.Stream(auth, l)

# Filter Twitter Streams to capture data by the keywords:
stream.filter(track=['clinton', 'trump', 'sanders', 'cruz'])

```

2.2.3 6.3 Load and explore your Twitter data

Now that you've got your Twitter data sitting locally in a text file, it's time to explore it! This is what you'll do in the next few interactive exercises. In this exercise, you'll read the Twitter data into a list: `tweets_data`.

Instructions

- Assign the filename 'tweets.txt' to the variable `tweets_data_path`.
- Initialize `tweets_data` as an empty list to store the tweets in.
- Within the for loop initiated by `for line in tweets_file:`, load each tweet into a variable, `tweet`, using `json.loads()`, then append `tweet` to `tweets_data` using the `append()` method.
- Hit submit and check out the keys of the first tweet dictionary printed to the shell.

```

[2]: # Import package
import json

# String of path to file: tweets_data_path
tweets_data_path = './Data/tweets3.txt'

```

```

# Initialize empty list to store tweets: tweets_data
tweets_data = []

# Open connection to file
tweets_file = open(tweets_data_path, "r")

# Read in tweets and store in list: tweets_data
for line in tweets_file:
    tweet = json.loads(line)
    tweets_data.append(tweet)

# Close connection to file
tweets_file.close()

# Print the keys of the first tweet dict
print(tweets_data[0].keys())

```

```

dict_keys(['in_reply_to_user_id', 'created_at', 'filter_level', 'truncated',
'possibly_sensitive', 'timestamp_ms', 'user', 'text', 'extended_entities',
'in_reply_to_status_id', 'entities', 'favorited', 'retweeted',
'is_quote_status', 'id', 'favorite_count', 'retweeted_status',
'in_reply_to_status_id_str', 'in_reply_to_user_id_str', 'id_str',
'in_reply_to_screen_name', 'coordinates', 'lang', 'place', 'contributors',
'geo', 'retweet_count', 'source'])

```

2.2.4 6.4 Twitter data to DataFrame

Now you have the Twitter data in a list of dictionaries, `tweets_data`, where each dictionary corresponds to a single tweet. Next, you're going to extract the text and language of each tweet. The text in a tweet, `t1`, is stored as the value `t1['text']`; similarly, the language is stored in `t1['lang']`. Your task is to build a DataFrame in which each row is a tweet and the columns are 'text' and 'lang'.

Instructions

- 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`.
- Print the head of the DataFrame.

```

[3]: # 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())

```

	text	lang
0	RT @bpolitics: .@krollbondrating's Christopher...	en
1	RT @HeidiAlpine: @dmartosko Cruz video found...	en
2	Njihuni me Zonjën Trump !!! Ekskluzive https...	et
3	Your an idiot she shouldn't have tried to grab...	en
4	RT @AlanLohner: The anti-American D.C. elites ...	en

2.2.5 6.5 A little bit of Twitter text analysis

Now that you have your DataFrame of tweets set up, you're going to do a bit of text analysis to count how many tweets contain the words 'clinton', 'trump', 'sanderson' and 'cruz'. In the pre-exercise code, we have defined the following function `word_in_text()`, which will tell you whether the first argument (a word) occurs within the 2nd argument (a tweet). `~~~ import re`

```
def word_in_text(word, text): word = word.lower() text = tweet.lower() match = re.search(word,
text)
```

```
if match:
    return True
return False
```

`~~~` You're going to iterate over the rows of the DataFrame and calculate how many tweets contain each of our keywords! The list of objects for each candidate has been initialized to 0.

Instructions

- Within the for loop `for index, row in df.iterrows():`, the code currently increases the value of `clinton` by 1 each time a tweet (text row) mentioning 'Clinton' is encountered; complete the code so that the same happens for `trump`, `sanders` and `cruz`.

```
[6]: import re

def word_in_text(word, text):
    word = word.lower()
    text = text.lower()
    match = re.search(word, text)
    if match:
        return True
    return False

import inspect
lines = inspect.getsource(word_in_text)
print(lines)

# Initialize list to store tweet counts
[clinton, trump, sanderson, cruz] = [0, 0, 0, 0]

# Iterate through df, counting the number of tweets in which
# each candidate is mentioned
for index, row in df.iterrows():
```

```

clinton += word_in_text('clinton', row['text'])
trump += word_in_text('trump', row['text'])
sanders += word_in_text('sanders', row['text'])
cruz += word_in_text('cruz', row['text'])

```

```

def word_in_text(word, text):
    word = word.lower()
    text = text.lower()
    match = re.search(word, text)
    if match:
        return True
    return False

```

2.2.6 6.6 Plotting your Twitter data

Now that you have the number of tweets that each candidate was mentioned in, you can plot a bar chart of this data. You'll use the statistical data visualization library seaborn, which you may not have seen before, but we'll guide you through. You'll first import seaborn as `sns`. You'll then construct a barplot of the data using `sns.barplot`, passing it two arguments: ~~~ 1. a list of labels and 2. a list containing the variables you wish to plot (clinton, trump and so on.) ~~~ Hopefully, you'll see that Trump was unreasonably represented! We have already run the previous exercise solutions in your environment.

Instructions - Import both `matplotlib.pyplot` and `seaborn` using the aliases `plt` and `sns`, respectively. - Complete the arguments of `sns.barplot`: the first argument should be the labels to appear on the x-axis; the second argument should be the list of the variables you wish to plot, as produced in the previous exercise.

```

[7]: # Import packages
import seaborn as sns
import matplotlib.pyplot as plt

# 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()

```

