Importing Data in Python (Part 1)

July 27, 2017

1 Introduction and flat files

1.1 Welcome to the course! (video)

1.2 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.

```
In [45]: # Open a file: file
         file = open('data/moby_dick.txt', '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
```

1.3 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.read()) You'll now use these tools to print the first few lines of moby_dick.txt!

```
In [46]: # 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
```

1.4 The importance of flat files in data science (video)

1.5 Using NumPy to import flat files

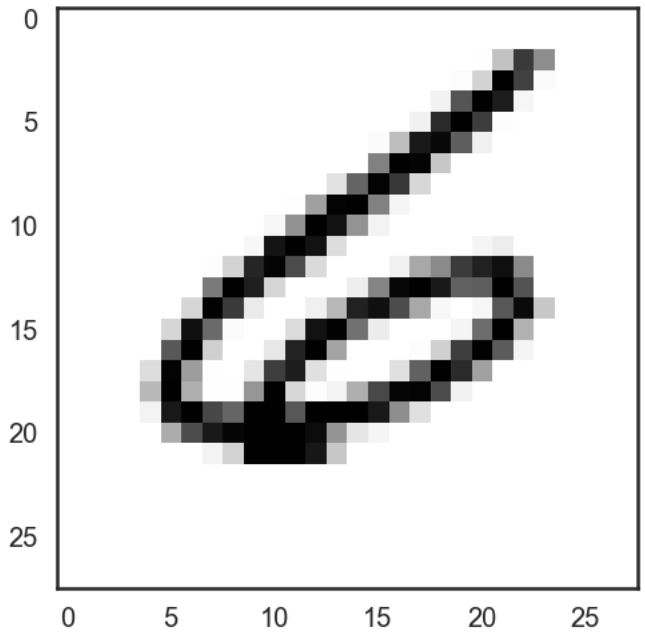
In this exercise, you're now going to load the MNIST digit recognition dataset using the numpy function `loadtxt()` and see just how easy it can be:

- 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.

In [47]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# Import package
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

plt.style.use('seaborn-white')

# 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'>
```



In [48]:
```
# 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)
```

```
[[ 1.  0.]
 [ 0.  0.]
 [ 1.  0.]
 [ 4.  0.]
 [ 0.  0.]
 [ 0.  0.]
 [ 7.  0.]
 [ 3.  0.]
 [ 5.  0.]
 [ 3.  0.]
 [ 8.  0.]
 [ 9.  0.]
 [ 1.  0.]
 [ 3.  0.]
 [ 3.  0.]
 [ 1.  0.]
 [ 2.  0.]
 [ 0.  0.]
 [ 7.  0.]
 [ 5.  0.]
 [ 8.  0.]
 [ 6.  0.]
 [ 2.  0.]
 [ 0.  0.]
 [ 2.  0.]
 [ 3.  0.]
 [ 6.  0.]
 [ 9.  0.]
 [ 9.  0.]
 [ 7.  0.]
 [ 8.  0.]
 [ 9.  0.]
 [ 4.  0.]
```

```
[ 9.  0.]
[ 2.  0.]
[ 1.  0.]
[ 3.  0.]
[ 1.  0.]
[ 1.  0.]
[ 4.  0.]
[ 9.  0.]
[ 1.  0.]
[ 4.  0.]
[ 4.  0.]
[ 2.  0.]
[ 6.  0.]
[ 3.  0.]
[ 7.  0.]
[ 7.  0.]
[ 4.  0.]
[ 7.  0.]
[ 5.  0.]
[ 1.  0.]
[ 9.  0.]
[ 0.  0.]
[ 2.  0.]
[ 2.  0.]
[ 3.  0.]
[ 9.  0.]
[ 1.  0.]
[ 1.  0.]
[ 1.  0.]
[ 5.  0.]
[ 0.  0.]
[ 6.  0.]
[ 3.  0.]
[ 4.  0.]
[ 8.  0.]
[ 1.  0.]
[ 0.  0.]
[ 3.  0.]
[ 9.  0.]
[ 6.  0.]
[ 2.  0.]
[ 6.  0.]
[ 4.  0.]
[ 7.  0.]
[ 1.  0.]
[ 4.  0.]
[ 1.  0.]
[ 5.  0.]
```

```
 [ 4.  0.]
 [ 8.  0.]
 [ 9.  0.]
 [ 2.  0.]
 [ 9.  0.]
 [ 9.  0.]
 [ 8.  0.]
 [ 9.  0.]
 [ 6.  0.]
 [ 3.  0.]
 [ 6.  0.]
 [ 4.  0.]
 [ 6.  0.]
 [ 2.  0.]
 [ 9.  0.]
 [ 1.  0.]
 [ 2.  0.]
 [ 0.  0.]
 [ 5.  0.]]
```

1.6 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.

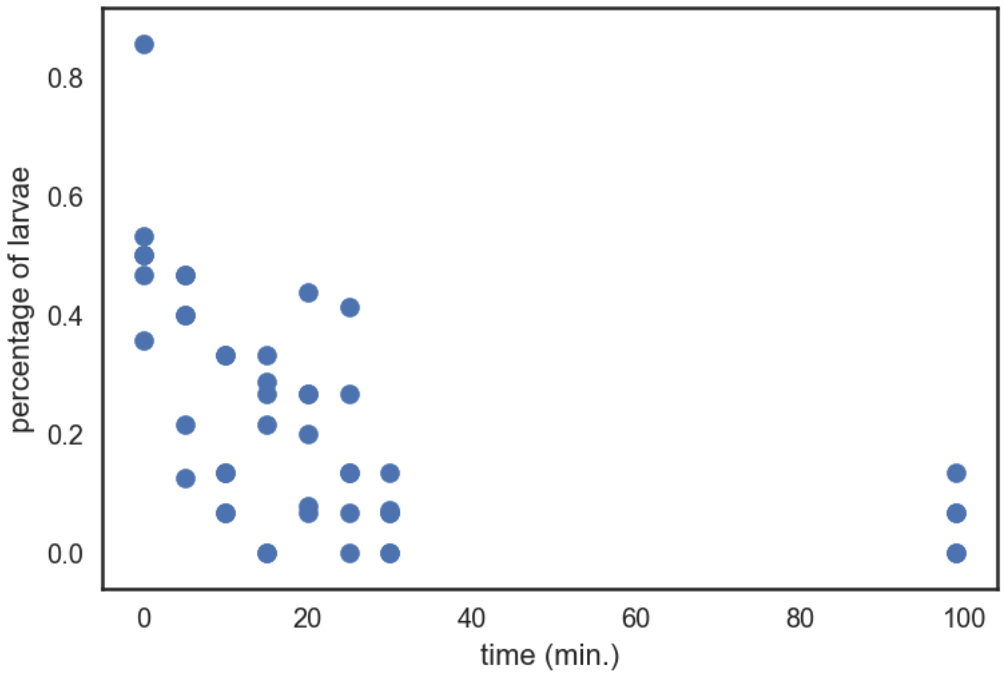
Due to the header, if you tried to import it as-is using `np.loadtxt()`, Python would throw you a ValueError and tell you that it could not convert `string` to `float`. There are two ways to deal with this: firstly, you can set the data type argument `dtype` equal to `str` (for string).

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

```
In [49]: # 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]
```



1.7 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:

```
In [50]: data = np.genfromtxt('data/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)`.

Acccessing rows and columns of structured arrays is super-intuitive: to get the ith 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

```
In [51]: print(data['Survived'])
```

```
[0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 0 1 0 0 1 1 0 0 0 1
 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0
 1 0 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0
 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1
 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0
 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1
 1 0 1 0 0 0 0 0 1 1 1 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0
 0 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 0
 0 1 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1
 1 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0
 1 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 1
 1 1 1 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0
 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0
 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1 1
 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1
 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0
 0 1 0 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0
 0 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0
 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0
 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0
 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0
 0 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1
 0 1 0]
```

1.8 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`, `delimiter=','` and `names=True`. In this exercise, you'll practice using this to achieve the same result.

```
In [52]: # Assign the filename: file
         file = 'data/titanic.csv'

         # Import file using np.recfromcsv: d
         d = np.recfromcsv(file)

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

```
[(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/O2. 3101282',   7.925 , b'', b'S')]
```

1.9 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().

```
In [53]: # 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
```

1.10 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.

```
In [54]: # 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'>
```

1.11 Customizing your pandas import

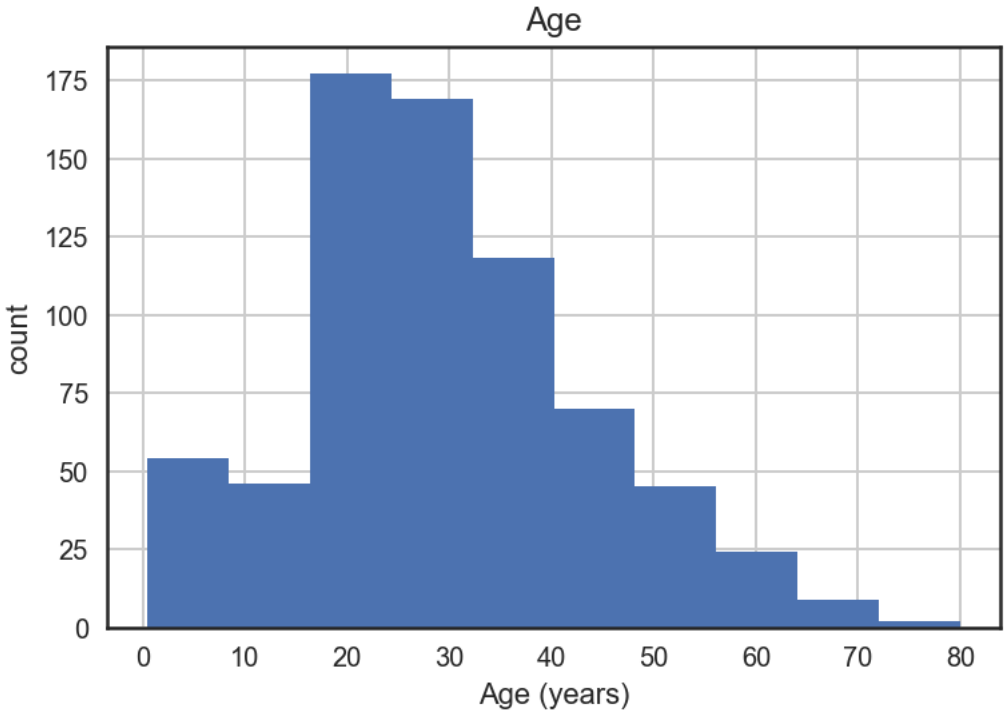
The pandas package is also great at dealing with many of the issues you will encounter when importing data as a data scientist, such as comments occurring in flat files, empty lines and missing values. Note that missing values are also commonly referred to as NA or NaN. To wrap up this chapter, you're now going to import a slightly corrupted copy of the Titanic dataset titanic_corrupt.txt, which

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

```
In [55]: # 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            Ticket  \
0            1         0       3    3  22.0      1      0         A/5 21171
1            2         1       1    1  38.0      1      0          PC 17599
2            3         1       3    3  26.0      0      0  STON/O2. 3101282
3            4         1       1    1  35.0      1      0            113803
4            5         0       3    3  35.0      0      0            373450

     Fare Cabin Embarked
0   7.250   NaN        S
1     NaN   NaN      NaN
2   7.925   NaN        S
3  53.100  C123        S
```

```
4    8.050    NaN        S
```



1.12 Final thoughts on data import (video)

2 Importing data from other file types

2.1 Introduction to other file types (video)

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.

```
In [56]: # Import pickle package
         import pickle

         # Open pickle file and load data: d
         with open('data/data.pkl', 'rb') as file:
             d = pickle.load(file)
```

```
        # Print d
        print(d)

        # Print datatype of d
        print(type(d))
```

```
{'June': '69.4', 'Airline': '8', 'Aug': '85', 'Mar': '84.4'}
<class 'dict'>
```

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.

```
In [57]: file = 'data/battledeath.xlsx'

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

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

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

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 .xslx file as a DataFrame. You'll be able to do so by specifying either the sheet's name or its index.

```
In [58]: # 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(0)
```

```
        # 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
```

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.

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.

```
In [59]: # 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())
         print()

         # Parse the first column of the second sheet and rename the column: df2
         df2 = xl.parse(1, 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

               Country
0              Albania
```

```
1              Algeria
2              Andorra
3               Angola
4  Antigua and Barbuda
```

2.6 Importing SAS/Stata files using pandas (video)

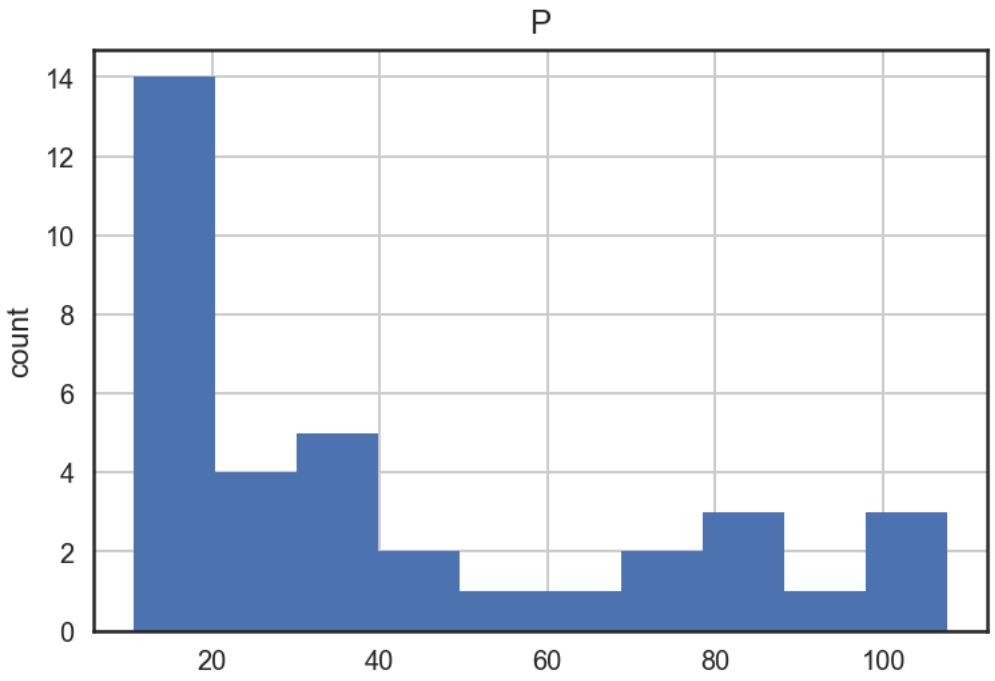
2.7 Importing SAS files

In this exercise, you'll figure out how to import a SAS file as a DataFrame using `SAS7BDAT` and `pandas`.

The data are adapted from the website of the undergraduate text book Principles of Economics by Hill, Griffiths and Lim.

```
In [60]: 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
```



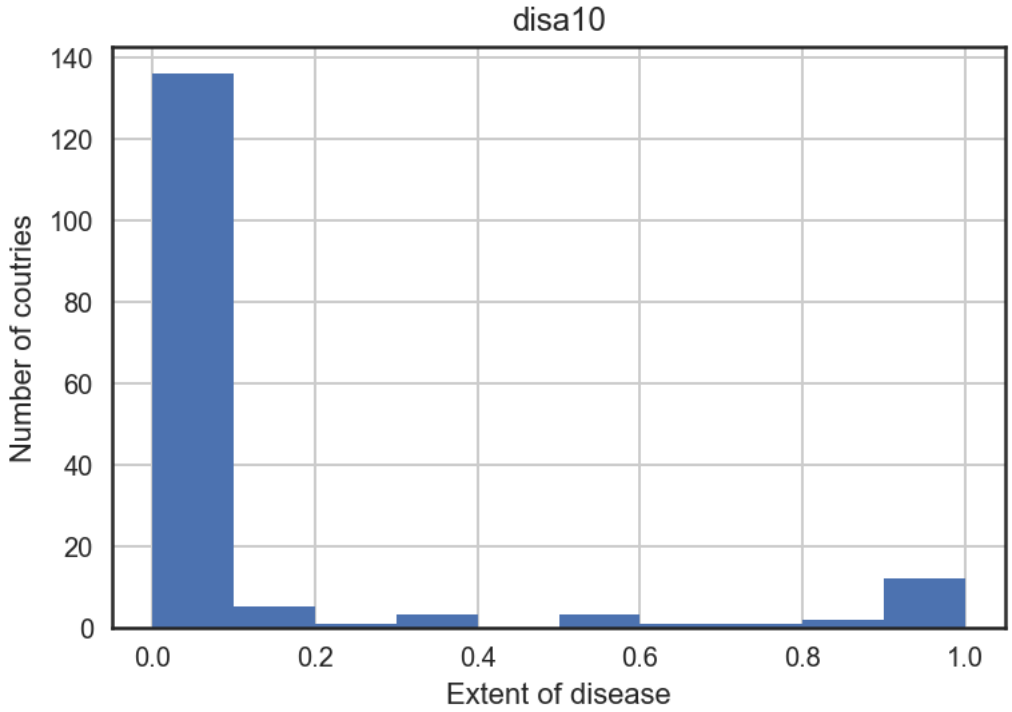
2.8 Importing Stata files

Here, you'll gain expertise in importing Stata files as DataFrames using the `pd.read_stata()` function from pandas.

```
In [61]: # 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 coutries')
         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  \
0   0.00    0.0   ...       0.0     0.0     0.0    0.00    0.00     0.0
1   0.56    0.0   ...       0.0     0.4     0.0    0.61    0.00     0.0
2   0.00    0.0   ...       0.0     0.0     0.0    0.00    0.00     0.0
3   0.00    0.0   ...       0.0     0.0     0.0    0.00    0.00     0.0
4   0.00    0.0   ...       0.0     0.0     0.0    0.00    0.05     0.0

   disa22  disa23  disa24  disa25
0    0.00    0.02    0.00    0.00
1    0.99    0.98    0.61    0.00
2    0.00    0.00    0.00    0.16
3    0.00    0.00    0.00    0.00
4    0.00    0.01    0.00    0.11

[5 rows x 27 columns]
```



2.9 Importing HDF5 files (video)

2.10 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.

```
In [62]: # 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
```

2.11 Extracting data from your HDF5 file

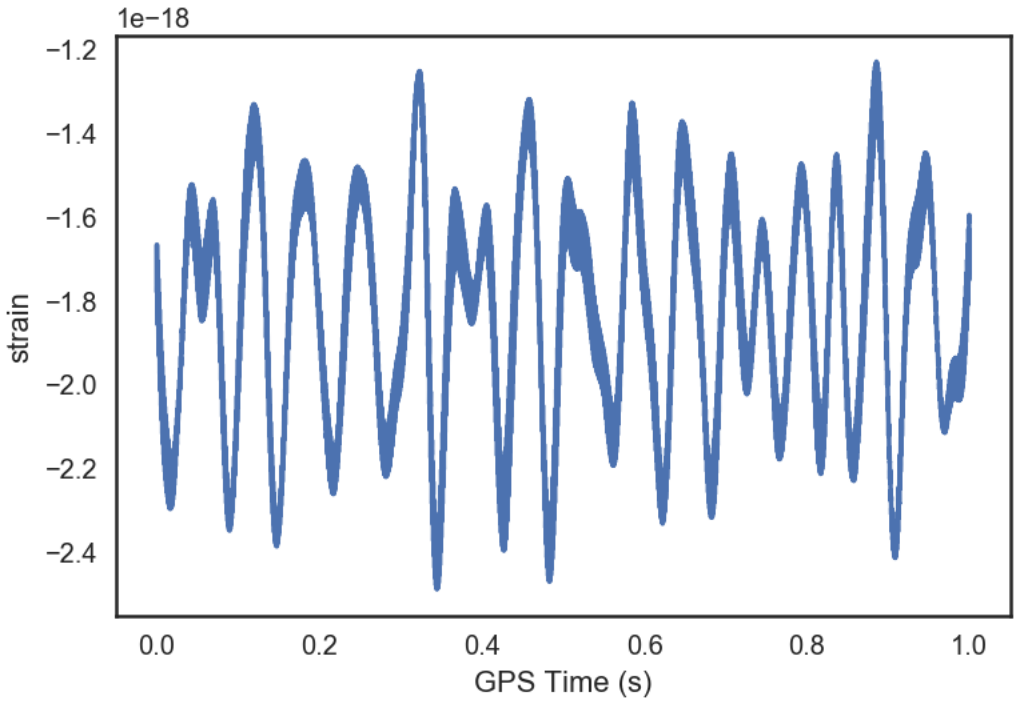
In this exercise, you'll extract some of the LIGO experiment's actual data from the HDF5 file and you'll visualize it.

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

```
In [63]: # 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
```



2.12 Importing MATLAB files (video)

2.13 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'` contains gene expression data from the Albeck Lab at UC Davis. You can find the data and some great documentation here.

```
In [64]: # 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))
```

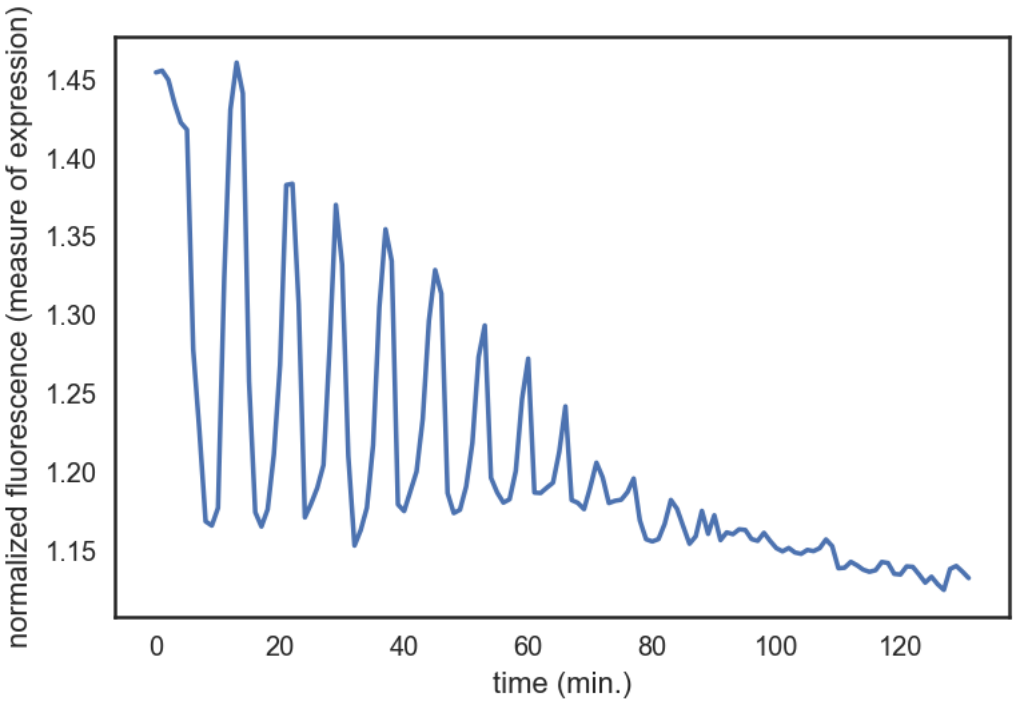
```
<class 'dict'>
```

2.14 The structure of .mat in Python

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

```
In [65]: # 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', '
<class 'numpy.ndarray'>
(200, 137)
```



3 Working with relational databases

3.1 Introduction to relational databases (video)

3.2 Creating a database engine in Python (video)

3.3 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'`. 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.*

```
In [66]: # Import necessary module
         from sqlalchemy import create_engine

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

3.4 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.

```
In [67]: # 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', 'Pla
```

3.5 Querying relational databases in Python (video)

3.6 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.

```
In [68]: # 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
```

3.7 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()
```

```
In [69]: # 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(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
```

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

```
In [70]: # 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
```

3.9 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`.

```
In [71]: # 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
```

3.10 Querying relational databases directly with pandas (video)

3.11 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)
```

```
In [72]: # 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
         # Perform query and save results to DataFrame: 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: does df = df1 ?
         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
```

3.12 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`.

```
In [73]: # Execute query and store records in DataFrame: df
         df = pd.read_sql_query("SELECT * FROM Employee WHERE EmployeeId >= 6 ORDER BY Birthdate

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

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

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

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

3.13 Advanced Querying: exploiting table relationships (video)

3.14 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.Cu
```

```
In [74]: # 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
             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
```

3.15 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: select all records from `PlaylistTrack INNER JOIN Track on PlaylistTrack.TrackId = Track.TrackId` that satisfy the condition `Milliseconds < 250000`.

```
In [75]: # Execute query and store records in DataFrame: df
         df = pd.read_sql_query("SELECT * FROM PlaylistTrack INNER JOIN Track ON PlaylistTrack.T

         # 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
```

```
In [ ]:
```