Introduction to other file types

1. 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 [1]:

```
# Created data.pkl file
import pickle
data = {'June': '69.4', 'Aug': '85', 'Airline': '8', 'Mar': '84.4'}
filename = 'data.pkl'
outfile = open(filename,'wb')
pickle.dump(data,outfile)
outfile.close()
```

In [2]:

```
# 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))
```

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

2.0 Listing sheets in Excel files

In [3]:

```
# Import pandas
import pandas as pd

# Assign spreadsheet filename:
file = 'battledeath.xlsx'
# Load spreadsheet: xl
xls = pd.ExcelFile(file)

# Print
print(xls.sheet_names)
```

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

2.1 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 'battledeath.xlsx' is already loaded as xls.

As before, you'll use the method parse(). This time, however, you'll add the additional arguments skiprows, names and usecols. 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 usecols parameter, skip the first row and rename the column 'Country'. The argument passed to usecols also needs to be of type list.

In [4]:

```
# Parse the first sheet and rename the columns: df1
df1 = xls.parse(0, skiprows=[0], names=['Country', 'AAM due to War (2002)'])

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

# Parse the first column of the second sheet and rename the column: df2
df2 = xls.parse(1, usecols=[0], skiprows=[1], 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
```

3. How to import SAS7BDAT

How do you correctly import the function SAS7BDAT() from the package sas7bdat?

```
from sas7bdat import SAS7BDAT
```

3.1 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

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!

In [5]:

```
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('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

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

4 Using read_stata to import Stata files

The pandas package has been imported in the environment as pd and the file `disarea.dta` is in your working directory. The data consist of disease extents for several diseases in various countries (more information can be found here).

What is the correct way of using the `read_stata()` function to import `disarea.dta` into the object `df` ?

```
df = pd.read_stata('disarea.dta')
```

5.1 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.

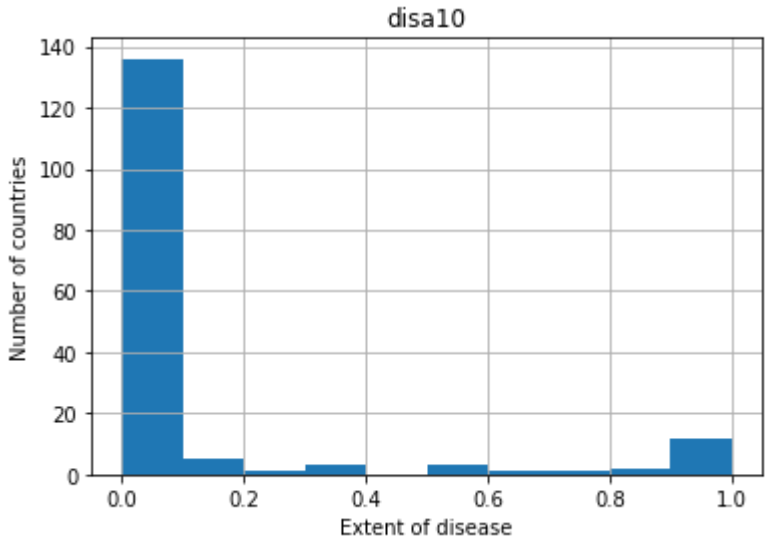
Instructions

- 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!

In [6]:

```
# Import pandas
import pandas as pd
import matplotlib.pyplot as plt

# Load Stata file into a pandas DataFrame: df

df = pd.read_stata('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]
```



6. Using File to import HDF5 files

The `h5py` package has been imported in the environment and the file `LIGO_data.hdf5` is loaded in the object `h5py_file`.

What is the correct way of using the h5py function, `File()`, to import the file in `h5py_file` into an object, `h5py_data`, for reading only?

```
h5py_data = h5py.File(h5py_file, 'r')
```

6.1 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 (https://losc.ligo.org/events/GW150914/). There is also a great tutorial on Signal Processing with the data here (https://www.gw-openscience.org/GW150914data/LOSC_Event_tutorial_GW150914.html)

In [7]:

```
# Import packages
import numpy as np
import h5py

# Assign filename: file

file= '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
```

6.2 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'`.

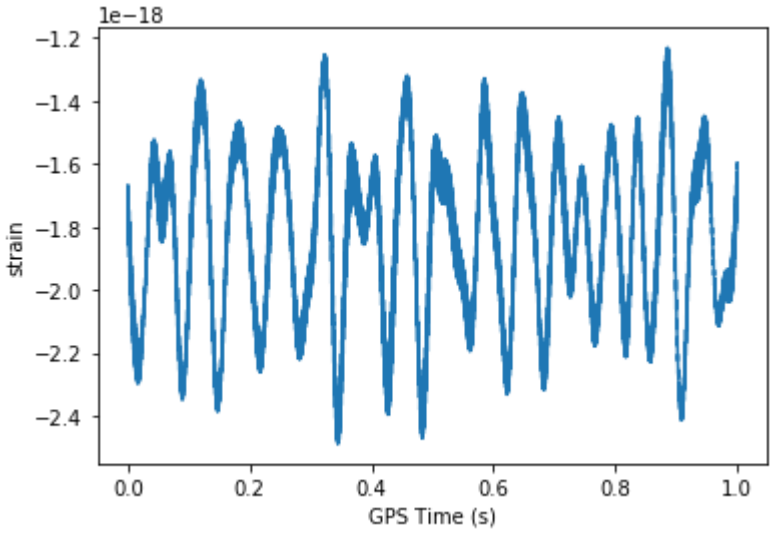
Instructions

- 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`.

In [8]:

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

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

# Set variable equal to time series data:

strain = data['strain']['Strain'].value
# Set number of time points to sample:
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

C:\Users\Jesus\Anaconda3\lib\site-packages\h5py\_hl\dataset.py:313: H5pyDepr
ecationWarning: dataset.value has been deprecated. Use dataset[()] instead.
  "Use dataset[()] instead.", H5pyDeprecationWarning)
```



7. Importing MATLAB files

7.1 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 (https://www.mcb.ucdavis.edu/faculty-labs/albeck/workshop.htm) from the Albeck Lab at UC Davis. You can find the data and some great documentation here (https://www.mcb.ucdavis.edu/faculty-labs/albeck/workshop.htm).

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.

In [9]:

```
# Import package
import scipy.io

# Load MATLAB file:

file = 'albeck_gene_expression.mat'
mat = scipy.io.loadmat(file)
# Print the datatype type of mat
print(type(mat))
```

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

7.2 The structure of .mat in Python

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

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!

In [10]:

```
# Print the keys of the MATLAB dictionary
print(mat.keys())

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

# 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'])
[[0.         1.53071547 1.54297013 ... 1.34990123 1.35329984 1.34922173]
 [0.         1.28605578 1.29385656 ... 1.31307311 1.30039694 1.30563938]
 [0.         1.32731222 1.32884617 ... 1.24887565 1.24506205 1.25825831]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         1.44552606 1.42862357 ... 0.         0.         0.        ]
 [0.         1.45794466 0.         ... 1.1229479  1.12224652 1.1486481 ]]
<class 'numpy.ndarray'>
(200, 137)
```