Exercise1

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().

In [ ]:

```
# Import pickle package

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

# Print d
print(____)

# Print datatype of d
print(____)


#____________________#
#Solutions

# 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))
```

Exercise2

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 spreadsheet filename (provided above) to the variable file.
- Pass the correct argument to pd.ExcelFile() to load the file using pandas, assigning the result to the variable xls.
- Print the sheetnames of the Excel spreadsheet by passing the necessary argument to the print() function.

In [ ]:

```
# Import pandas
import pandas as pd

# Assign spreadsheet filename: file
file = ____

# Load spreadsheet: xls
xls = pd.ExcelFile(____)

# Print sheet names
print(____)


#____________________#
#Solutions

# Import pandas
import pandas as pd

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

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

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

Exercise3

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 'battledeath.xlsx' is already loaded as xls.

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.

In [ ]:

```
# Load a sheet into a DataFrame by name: df1
df1 = xls.parse(____)

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

# Load a sheet into a DataFrame by index: df2


# Print the head of the DataFrame df2
print(____)


#_____________________#
#Solutions

# Load a sheet into a DataFrame by name: df1
df1 = xls.parse('2004')

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

# Load a sheet into a DataFrame by index: df2
df2 = xls.parse(0)

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

Exercise4

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 [ ]:

```
# Parse the first sheet and rename the columns: df1
df1 = xls.parse(____, skiprows=____, names=____)

# 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(____, usecols=____, skiprows=____, names=____)

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


#_____________________#
#Solutions

# 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=[0], names=['Country'])

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

Exercise5

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!

In [ ]:

```
# Import sas7bdat package
from ____ import ____

# Save file to a DataFrame: df_sas
with SAS7BDAT('sales.sas7bdat') as file:
    ____

# Print head of DataFrame


# Plot histogram of DataFrame features (pandas and pyplot already imported)
pd.DataFrame.hist(df_sas[['P']])
plt.ylabel('count')
plt.show()


#____________________#
#Solutions

# 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()
```

Exercise6

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 [ ]:

```
# Import pandas
import pandas as pd

# Load Stata file into a pandas DataFrame: df

# Print the head of the DataFrame df

# 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()



#_____________________#
#Solutions

# Import pandas
import pandas as pd

# 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()
```

Exercise7

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'.

In [ ]:

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

# Assign filename: file


# Load file: data
data = h5py.File(____, ____)

# Print the datatype of the loaded file


# Print the keys of the file
for key in ____:
    print(____)



#_____________________#
#Solutions

# 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)
```

Exercise8

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 [ ]:

```
# Get the HDF5 group: group


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

# Set variable equal to time series data: strain


# Set number of time points to sample: num_samples


# 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()

#_____________________#
#Solutions

# 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()
```

Exercise9

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.

In [ ]:

```
# Import package

# Load MATLAB file: mat

# Print the datatype type of mat
print(____)


#_____________________#
#Solutions

# Import package
import scipy.io

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

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

Exercise10

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 UCDavis. 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!

In [ ]:

```
# Print the keys of the MATLAB dictionary
print(____)

# Print the type of the value corresponding to the key 'CYratioCyt'


# Print the shape of the value corresponding to the key '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()


#____________________#
#Solutions

# 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()
```