1. **Kvejveorv**
2. **Introduction To importing Data**

Q1 :

battledeath.xlsx is not a flat because it is a spreadsheet consisting of many sheets, not a single table.

Q2 :

* 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().

# 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))

|  |  |
| --- | --- |
|  | Q3:- |
|  | 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. |
|  | # 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) |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | Q4:- |
|  | 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. |
|  | Print the head of df2 to the shell. |
|  |  |
|  | Solution:-  # 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()) |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | Q5:- |
|  | 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. |
|  |  |
|  | Solution:-  # 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()) |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | Q6:- |
|  | * 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. |
|  |  |
|  | Solution:-  # 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() |
|  |  |
|  |  |
|  | Q7:- |
|  | What is the correct way of using the read\_stata() function to import disarea.dta into the object df? |
|  |
|  | Solution:-  df = pd.read\_stata('disarea.dta') |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | Q8:- |
|  | * 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! |
|  |  |
|  | Solution:- |
|  |  |
|  | # 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() |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Q9:- |
|  | 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? |
|  |  |
|  | Solution:- |
|  | h5py\_data = h5py.File(h5py\_file, 'r') |
|  |  |
|  | Q10:- |
|  | 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'. |
|  |  |
|  | Solution:- |
|  | # 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) |
|  |  |
|  | Q11:- |
|  | 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. |
|  |  |
|  | Solution:  # 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 = np.array(data['strain']['Strain'])  # 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() |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| Q12: |  |

* 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.

# 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)))

1. **Working with relational database in python**

Q1:

**Problème:**

**Solution:**

# Import necessary module

from sqlalchemy import create\_engine

# Create engine: engine

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

Q2 :

**Problème:**

**Solution:**

# Import necessary module

from sqlalchemy import create\_engine

# Create engine: engine

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

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

table\_names= engine.table\_names()

# Print the table names to the shell

print(table\_names)

Q3:

**Problème:**

**Solution:**

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

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

# Open engine connection: con

con= engine.connect()

# Perform query: rs

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

# Save results of the query to DataFrame: df

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

# Close connection

con.close()

# Print head of DataFrame df

print(df.head())

Q4 :

**Problème:**

**Solution:**

# 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)) # return 3 rows

df.columns = rs.keys()

# Print the length of the DataFrame df

print(len(df))

# Print the head of the DataFrame df

print(df.head()) # affiche le 5 premieres lignes par defaut

Q5 :

**Problème:**

* 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.

**Solution:**

# Create engine: engine

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

# Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

    rs = con.execute("SELECT \* FROM Employee WHERE EmployeeId>=6 ")

    df = pd.DataFrame(rs.fetchall())

    df.columns = rs.keys()

# Print the head of the DataFrame df

print(df.head())

Q6 :

Problème :

* 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.

Solution :

# Create engine: engine

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

# Open engine in context manager

with engine.connect() as con:

    rs = engine.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())

Q7 :

* 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.

Solution :

# Import packages

from sqlalchemy import create\_engine

import pandas  as pd

# Create engine: engine

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

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("SELECT \* FROM Album", engine)

# 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))

Q8 :

Probleme :

* 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).

Solution :

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

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

# Execute query and store records in DataFrame: df

df= pd.read\_sql\_query("SELECT \* FROM Employee WHERE EmployeeId>=6  ORDER BY BirthDate  ", engine )

# Print head of DataFrame

print(df.head())

Q9 :

* 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.

Solution :

# 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())

Q10 :

* 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.

Solution :

# Execute query and store records in DataFrame: df

df = pd.DataFrame(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())

Partie 2 :

1. Chap 1 : Importing float files from the web

Q1 :

* 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.

Solution :

# Import package

from urllib.request import urlretrieve

# Import pandas

import pandas as pd

# Assign url of file: url

url='https://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())

Q2 :

* Q2 : 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.

Solution :

# Import packages

import matplotlib.pyplot as plt

import pandas as pd

# Assign url of file: url

url='https://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

df.iloc[:, 0].hist()

plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')

plt.ylabel('count')

plt.show()

Q3 :

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

Solution :

# Import package

import pandas as pd

# Assign url of file: url

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

# Read in all sheets of Excel file: xls

xls= pd.read\_excel(url,sheet\_name=None)

# Print the sheetnames to the shell

print(xls.keys())

# Print the head of the first sheet (using its name, NOT its index)

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

Q4 :

* Import the functions urlopen and Request from the subpackage urllib.request.
* Package the request to the url "https://campus.datacamp.com/courses/1606/4135?ex=2" 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!

Solution :

# Import packages

from urllib.request import urlopen , Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# 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()

Q 5:

* 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!

Solution:

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# 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()

Q 6:

* 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.

Solution:

# 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)

Q7 :

* 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!

Solution :

# 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)

Q8 :

* 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.

Solution :

# 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)

Q9 :

* 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').

Solution :

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url

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

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

r = requests.get(url)

# Extracts the response as html: html\_doc

html\_doc = r.text

# create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Print the title of Guido's webpage

print(soup.title)

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

a\_tags= soup.find\_all('a')

# Print the URLs to the shell

for link in a\_tags:

    print(link.get('href'))

# chap2:Introduction to APIs and JSONs

Q1 :

The function json.load() will load the JSON into Python as a list.

Q2 :

* 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].

Solution :

# Load JSON: json\_data

with open("a\_movie.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])

Q3 :

The title is 'The Social Network' and the year is 2010.

Q4 :

* 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 response object r by using its text attribute and passing the result to the print() function.

Solution :

# 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)

Q 5 :

* 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 to print the key-value pairs of the dictionary json\_data to the shell.

Solution :

# 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])

Q6 :

* 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.

Solution:

# 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)

chap 3 :

Q 1 :

* Create your Stream object with the credentials given.
* Filter your Stream variable for the keywords "clinton", "trump", "sanders", and "cruz".

# Store credentials in relevant variables

consumer\_key = "nZ6EA0FxZ293SxGNg8g8aP0HM"

consumer\_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"

access\_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fFO8k8EPy"

access\_token\_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"

# Create your Stream object with credentials

stream = tweepy.Stream(consumer\_key, consumer\_secret, access\_token, access\_token\_secret)

# Filter your Stream variable

stream.filter(track=['clinton', 'trump' ,'sanders','cruz'])

Q2 :

* 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.

Solution :

# Import package

import json

# String of path to file: tweets\_data\_path

tweets\_data\_path="tweets.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())

Q3 :

* 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.

Solution :

# 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())

Q4 :

* 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.

Solution :

# Initialize list to store tweet counts

[clinton, trump, sanders, 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'])

Q5 :

* 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 list of labels to appear on the x-axis (created in the previous step).
  + The second argument should be a list of the variables you wish to plot, as produced in the previous exercise (i.e. a list containing clinton, trump, etc).

Solution:

# Import packages

import matplotlib.pyplot as plt

import seaborn as sns

# Set seaborn style

sns.set(color\_codes=True)

# Create a list of labels:cd

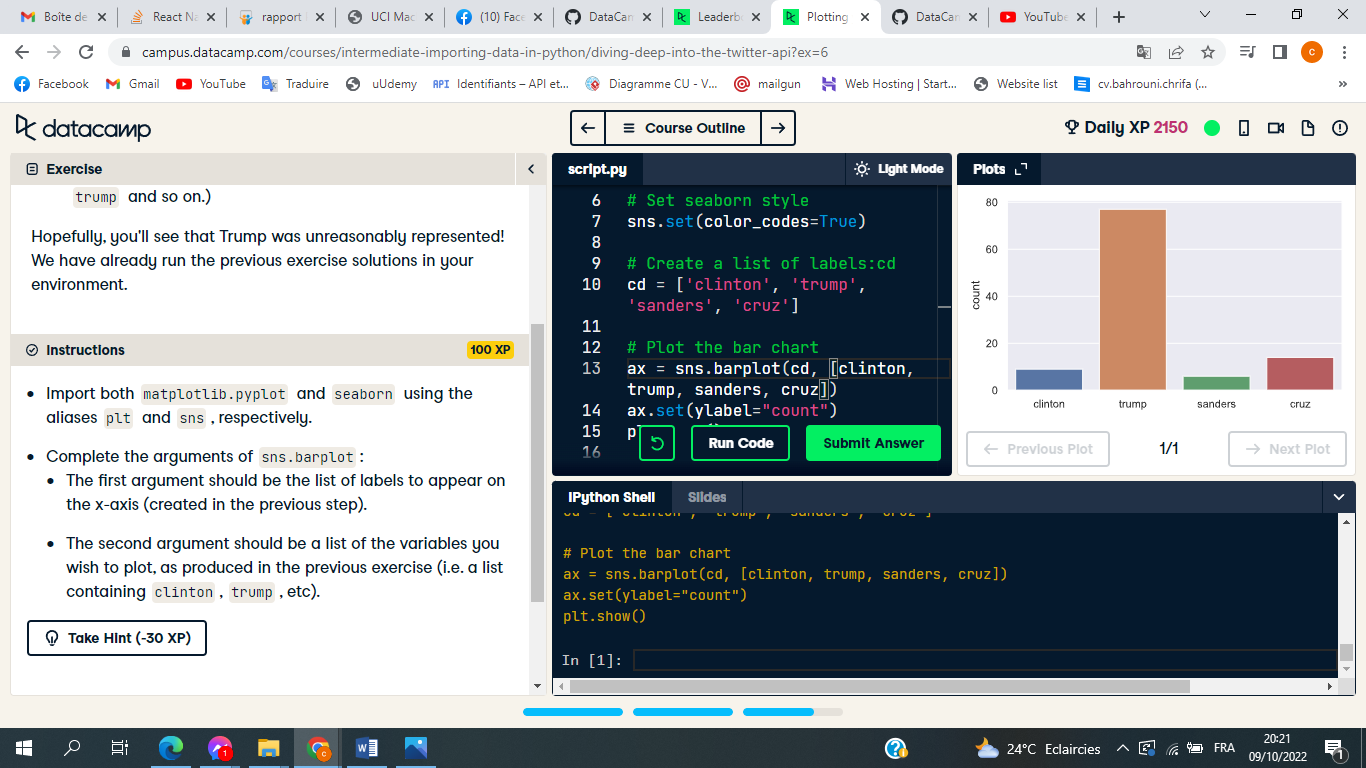
cd = ['clinton', 'trump', 'sanders', 'cruz']

# Plot the bar chart

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

ax.set(ylabel="count")

plt.show()



Partie 3 :

