Lab 5 Data Structures

Resources

- https://pandas.pydata.org/pandas-docs/version/0.17.0/generated/pandas.io.json.json_normalize.html
- https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html
- https://docs.python.org/2/library/xml.etree.elementtree.html

In [1]:
```
# import modules
import pandas as pd
import numpy as np

from pandas.io.json import json_normalize #special package in pandas
import json
from lxml import etree
import sqlite3
```

Part A: Reading json

In [194…
```
# Run this cell to import the following JSON strings
players = '[{"name":"Gazinsky","team":"Russia"},{"name":"Dzyuba","team":"Russia"},{"na
stadium_data = '{"stadiums":[{"name": "Ekaterinburg Arena","city": "Ekaterinburg"},{"r
```

In [195…
```
# load players string as a json object (using json.loads)
p = json.loads(players)
# print the dictionary containing information about the third player only
# HINT: Once the data is in python data structures (i.e. lists/dictionaries)
# navigate through the nested lists and dictionaries to extract the relevant informati
# HINT: Remember, the index starts at 0, so the first player is index 0, the second pl
print(p)
```

```
[{'name': 'Gazinsky', 'team': 'Russia'}, {'name': 'Dzyuba', 'team': 'Russia'}, {'nam
e': 'Lukaku', 'team': 'Belgium'}]
```

In [196…
```
# load stadium_data as a json object
s = json.loads(stadium_data)
# print the city of Luzhniki Stadium by navigating through the nested dictionaries and
s['stadiums'][1]['city']
```

Out[196]:
```
'Moscow'
```

In [197…
```
# read world cup match data from the worldcup.json file into a json object
worldcup = open("worldcup.json", encoding = 'utf-8')
wc = json.load(worldcup)
# print the date of the first match (key="date")
# HINT: Once the data is in python data structures (i.e. lists/dictionaries)
# navigate through the nested lists and dictionaries to extract the relevant informati
wc['rounds'][0]['matches'][0]['date']
```

Out[197]:
```
'2018-06-14'
```

In [198…
```
# read world cup team data from the worldcup.teams.json file into json object
worldcupteams = open("worldcup.teams.json", encoding = 'utf-8')
wct = json.load(worldcupteams)
# print the name of the first team in the team data set (key="name")
# HINT: Once the data is in python data structures (i.e. lists/dictionaries)
# navigate through the nested lists and dictionaries to extract the relevant informati
wct['teams'][0]['name']
```

Out[198]:
```
'Egypt'
```

In [199…
```
# manually create a list of dictionaries with the following soccer associations and cc
# The keys should be  "continent" and "association".

# Europe - Union of European Football Associations
# Asia - Asian Football Confederation
# Africa - Confederation Africaine de Football

soccerasc = '[{"continent":"Europe","association":"Union of European Football Associat
```

Part B: Flattening json

In [200…
```
# flatten the players json object created in Part A (second cell) into a data frame wi
p_df = json_normalize(p)
# print first few rows of data set
p_df.head()
```

```
C:\Users\Jivinnii\AppData\Local\Temp\ipykernel_20044\2549829104.py:2: FutureWarning:
pandas.io.json.json_normalize is deprecated, use pandas.json_normalize instead.
  p_df = json_normalize(p)
```

Out[200]:

	name	team
0	Gazinsky	Russia
1	Dzyuba	Russia
2	Lukaku	Belgium

In [201…
```
# flatten the stadium data json object created in Part A (third cell) into a data fram
s_df = json_normalize(s, record_path = "stadiums")
# print first few rows of data set
s_df.head()
```

```
C:\Users\Jivinnii\AppData\Local\Temp\ipykernel_20044\1292605132.py:2: FutureWarning:
pandas.io.json.json_normalize is deprecated, use pandas.json_normalize instead.
  s_df = json_normalize(s, record_path = "stadiums")
```

Out[201]:

	name	city
0	Ekaterinburg Arena	Ekaterinburg
1	Luzhniki Stadium	Moscow
2	Nizhny Novgorod Stadium	Nizhny Novgorod

In [202…
```
# flatten the world cup match json object created in Part A (fourth cell)  into a data
# HINT: The data you want is nested in "rounds", and further in the "matches" key in t
# HINT: You should include both rounds & matches as nested keys in record path
# (see https://stackoverflow.com/questions/47242845/pandas-io-json-json-normalize-with
# e.g. record_path=["level1key",["level2key"]]
# where level1key = "rounds" and level2key = "matches"
wc_df = json_normalize(wc, record_path = ["rounds", ["matches"]])
# print number of rows
wc_df.shape[0]
# print first few rows of data set
wc_df.head()
```

```
C:\Users\Jivinnii\AppData\Local\Temp\ipykernel_20044\2558462357.py:7: FutureWarning:
pandas.io.json.json_normalize is deprecated, use pandas.json_normalize instead.
  wc_df = json_normalize(wc, record_path = ["rounds", ["matches"]])
```

Out[202]:

	num	date	time	score1	score2	score1i	score2i	goals1	goals2	group	…	team1
0	1	2018-06-14	18:00	5	0	2	0	[{'name': 'Gazinsky', 'minute': 12, 'score1': …	[]	Group A	…	
1	2	2018-06-15	17:00	0	1	0	0	[]	[{'name': 'Giménez', 'minute': 89, 'score1': 0…	Group A	…	
2	3	2018-06-15	21:00	3	3	2	1	[{'name': 'Ronaldo', 'minute': 4, 'score1': 1,…	[{'name': 'Costa', 'minute': 24, 'score1': 1, …	Group B	…	
3	4	2018-06-15	18:00	0	1	0	0	[]	[{'name': 'Bouhaddouz', 'minute': 90, 'offset'…	Group B	…	
4	5	2018-06-16	13:00	2	1	0	0	[{'name': 'Griezmann', 'minute': 58, 'score1':…	[{'name': 'Jedinak', 'minute': 62, 'score1': 1…	Group C	…	

5 rows × 23 columns

In [203…
```
# flatten the team data json object created in Part A (fifth cell) into a data frame w
wct_df = json_normalize(wct, record_path = "teams")
# print number of rows
wct_df.shape[0]
# print first few rows of data set
wct_df.head()
```

```
C:\Users\Jivinnii\AppData\Local\Temp\ipykernel_20044\2839655693.py:2: FutureWarning:
pandas.io.json.json_normalize is deprecated, use pandas.json_normalize instead.
  wct_df = json_normalize(wct, record_path = "teams")
```

Out[203]:

	name	code	continent	assoc.key	assoc.name	assoc.continental.name	assoc.continental.code
0	Egypt	EGY	Africa	egy	Egyptian Football Association	Confédération Africaine de Football (CAF)	CAF
1	Morocco	MAR	Africa	mar	Fédération Royale Marocaine de Football	Confédération Africaine de Football (CAF)	CAF
2	Tunisia	TUN	Africa	tun	Fédération Tunisienne de Football	Confédération Africaine de Football (CAF)	CAF
3	Senegal	SEN	Africa	sen	Fédération Sénégalaise de Football	Confédération Africaine de Football (CAF)	CAF
4	Nigeria	NGA	Africa	nga	Nigeria Football Federation	Confédération Africaine de Football (CAF)	CAF

In [204…

```
# flatten the soccer association list of dictionaries created in Part A (sixth cell) i
sa = json.loads(soccerasc)
sa_df = json_normalize(sa)
# print number of rows
sa_df.shape[0]
# print first few rows of data set
sa_df.head()
```

```
C:\Users\Jivinnii\AppData\Local\Temp\ipykernel_20044\3113017416.py:3: FutureWarning:
pandas.io.json.json_normalize is deprecated, use pandas.json_normalize instead.
  sa_df = json_normalize(sa)
```

Out[204]:

	continent	association
0	Europe	Union of European Football Associations
1	Asia	Asian Football Confederation
2	Africa	Confederation Africaine de Football

Part C: SQL

In [205…

```
# create a database and sql connection
con = sqlite3.connect('soc_database.sqlite')
```

In [206…

```
# save the players data set created in part B as a sql table in your database
p_df.to_sql('playerstbl', con, if_exists = 'replace')
```

Out[206]: 3

In [207…

```
# save the stadiums data set created in part B as a sql table in your database
s_df.to_sql('stadiumstbl', con, if_exists = 'replace')
```

Out[207]:
```
3
```

In [208…
```
# save the world cup match data set created in part B as a sql table in your database
# (you'll need to FIRST slice the data frame by selecting only the columns we want. Dr
wc_dfs = wc_df[['num', 'date', 'time', 'score1', 'score2', 'score1i', 'score2i', 'grou
wc_dfs.to_sql('matchestbl', con, if_exists = 'replace')
```

Out[208]:
```
64
```

In [209…
```
# save the teams data set created in part B as a sql table in your database. Keep the
# variables: "code","continent", & "name"
wct_dfs = wct_df[['code', 'continent', 'name']]
wct_dfs.to_sql('teamstbl', con, if_exists = 'replace')
```

Out[209]:
```
32
```

In [210…
```
# From the world cup match data table, use SQL syntax to select only those rows where
# HINT: You'll need to use WHERE to select the right rows
# VERY IMPORTANT HINT: if a column name includes a period (.), you must enclose the
# column name  AND the value for the WHERE statement in double quotes, and enclose the
# eg. 'SELECT * FROM df WHERE "col1.name"="Atlanta" OR "col2.name"="Atlanta"'
query1 = 'SELECT * FROM matchestbl WHERE "team1.name" = "Mexico" OR "team2.name" = "Me
matches1 = pd.read_sql_query(query1, con)
matches1.to_sql('matches1tbl', con, if_exists = 'replace')
# count the number of rows in the data frame - This represents the number of games Mex
queryc1 = "SELECT COUNT(*) AS 'Total Games' FROM matches1tbl"

# print the first few rows of the data frame
totalgames = pd.read_sql_query(queryc1, con)
print(totalgames)
```

```
   Total Games
0            4
```

In [211…
```
# From the world cup match data table, select only those rows that were tied games fro
# HINT: By tie, this means that score1 is the same as score2
query2 = 'SELECT * FROM matchestbl WHERE "score1" == "score2"'
tie_matches = pd.read_sql_query(query2, con)
tie_matches.to_sql('matchtietbl', con, if_exists = 'replace')
# count the number of rows in the data frame
queryc2 = "SELECT COUNT(*) AS 'Total Ties' FROM matchtietbl"

# print the first few rows of the data frame
tie_games = pd.read_sql_query(queryc2, con)
print(tie_games)
```

```
   Total Ties
0          14
```

In [ ]:
```
```