Data Wrangling with MongoDB

Lesson 1: Problem Set

Using CSV Module:

parsecvs.py

```
#!/usr/bin/env python
"""
Your task is to process the supplied file and use the csv module to extract data from it.
The data comes from NREL (National Renewable Energy Laboratory) website. Each file
Contains information from one meteorological station, in particular - about amount of
Solar and wind energy for each hour of day.

Note that the first line of the data file is neither data entry, nor header. It is a line
Describing the data source. You should extract the name of the station from it.

The data should be returned as a list of lists (not dictionaries).
You can use the csv modules "reader" method to get data in such format.
Another useful method is next() - to get the next line from the iterator.
You should only change the parse file function.
"""
import csv
import os

DATADIR = ""
DATAFILE = "745090.csv"

def parse_file(datafile):
    name = ""
    data = []

    with open(datafile,'rb') as f:
        r = csv.reader(f)
        name = r.next()[1]
        header = r.next()
        data = [row for row in r]
        pass
    # Do not change the line below
    return (name, data)

def test():
    datafile = os.path.join(DATADIR, DATAFILE)
    name, data = parse_file(datafile)
    assert name == "MOUNTAIN VIEW MOFFETT FLD NAS"
    assert data[0][1] == "01:00"
    assert data[2][0] == "01/01/2005"
    assert data[2][5] == "2"
```

```
if __name__ == "__main__":
    test()
```

parsecvs.py Output: All data values are correct!

Excel To CSV:

excel_csv.py

```
# -*- coding: utf-8 -*-
# Find the time and value of max load for each of the regions
# COAST, EAST, FAR_WEST, NORTH, NORTH_C, SOUTHERN, SOUTH_C, WEST
# and write the result out in a csv file, using pipe character | as the delimiter.
# An example output can be seen in the "example.csv" file.

import xlrd
import os
import csv
from zipfile import ZipFile

datafile = "2013_ERCOT_Hourly_Load_Data.xls"
outfile = "2013_Max_Loads.csv"

def open_zip(datafile):
    with ZipFile('{0}.zip'.format(datafile), 'r') as myzip:
        myzip.extractall()

def parse_file(datafile):
    workbook = xlrd.open_workbook(datafile)
    sheet = workbook.sheet_by_index(0)
    data = []
    # YOUR CODE HERE
    # Remember that you can use xlrd.xldate_as_tuple(sometime, 0) to convert
    # Excel date to Python tuple of (year, month, day, hour, minute, second)
    #get time column values
    time_col = sheet.col_values(0, 1)

    for row in range(1, 9):
        region = sheet.cell_value(0, row)

        #get data from column
        col = sheet.col_values(row, 1)
        #get max value of column
        max_value = max(col)
        #get column index of max value so we can get the time form time column
        max_index = col.index(max(col))
        exceltime_max = time_col[max_index]
        maxtime = xlrd.xldate_as_tuple(exceltime_max, 0)
        data.append([region, maxtime[0], maxtime[1], maxtime[2],maxtime[3], max_value])
    return data

def save_file(data, filename):
    # YOUR CODE HERE
    with open(filename, 'wb') as ofile:
        writer = csv.writer(ofile, delimiter="|")
        writer.writerow(["Station", "Year", "Month", "Day", "Hour", "Max Load"])
        for each_row in data:
            writer.writerow(each_row)


def test():
    open_zip(datafile)
    data = parse_file(datafile)
    save_file(data, outfile)
    number_of_rows = 0
    stations = []
    ans = {'FAR_WEST': {'Max Load': '2281.2722140000024',
                        'Year': '2013',
                        'Month': '6',
                        'Day': '26',
                        'Hour': '17'}}
    correct_stations = ['COAST', 'EAST', 'FAR_WEST', 'NORTH',
                        'NORTH_C', 'SOUTHERN', 'SOUTH_C', 'WEST']
    fields = ['Year', 'Month', 'Day', 'Hour', 'Max Load']

    with open(outfile) as of:
        csvfile = csv.DictReader(of, delimiter="|")
        for line in csvfile:
            station = line['Station']
            if station == 'FAR_WEST':
                for field in fields:
                    # Check if 'Max Load' is within .1 of answer
                    if field == 'Max Load':
                        max_answer = round(float(ans[station][field]), 1)
                        max_line = round(float(line[field]), 1)
                        assert max_answer == max_line

                    # Otherwise check for equality
                    else:
                        assert ans[station][field] == line[field]

            number_of_rows += 1
            stations.append(station)
        # Output should be 8 lines not including header
        assert number_of_rows == 8
        # Check Station Names
        assert set(stations) == set(correct_stations)
```

```
if __name__ == "__main__":
    test()
```

excel_csv.py Output: Congratulations, your solution is correct!

<u>Wrangling JSON:</u>
<u>nytimes.py</u>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
This exercise shows some important concepts that you should be aware about:
- using codecs module to write unicode files
- using authentication with web APIs
- using offset when accessing web APIs

To run this code locally you have to register at the NYTimes developer site
and get your own API key. You will be able to complete this exercise in our UI without doing so,
as we have provided a sample result.

Your task is to process the saved file that represents the most popular (by view count)
articles in the last day, and return the following data:
- list of dictionaries, where the dictionary key is "section" and value is "title"
- list of URLs for all media entries with "format": "Standard Thumbnail"

All your changes should be in the article_overview function.
The rest of functions are provided for your convenience, if you want to access the API by yourself.
"""
import json
import codecs
import requests

URL_MAIN = "http://api.nytimes.com/svc/"
URL_POPULAR = URL_MAIN + "mostpopular/v2/"
API_KEY = { "popular": "",
           "article": ""}

def get_from_file(kind, period):
   filename = "popular-{0}-{1}.json".format(kind, period)
   with open(filename, "r") as f:
      return json.loads(f.read())

def article_overview(kind, period):
   data = get_from_file(kind, period)
   titles = []
   urls =[]
   # YOUR CODE HERE
   #- return a list of dictionaries, where the dictionary key is "section" and value is "title"
   for each_article in data:
      for (key, value) in each_article.items():
         if key == 'section':
            #print (key, value)
            titles.append({each_article['section']: each_article['title']})
```

```
    #- return a list of URLs for all media entries with "format": "Standard Thumbnail"
    #data is a list of dictionaries. Each dictionary is one article and its data
    for each_article in data:
        #each_article['media'] is a list of dictionaries. each dict is a list medias
        for each_media in each_article['media']:
            #each_media is a dictionary of medias. each dict is a medias data
            #each media-metadata is a list of dictionaries. Each dict is the metadata info
            for each_metadata in each_media['media-metadata']:
                if each_metadata['format'] == "Standard Thumbnail":
                    urls.append(each_metadata['url'])

    return (titles, urls)

def query_site(url, target, offset):
    # This will set up the query with the API key and offset
    # Web services often use offset paramter to return data in small chunks
    # NYTimes returns 20 articles per request, if you want the next 20
    # You have to provide the offset parameter
    if API_KEY["popular"] == "" or API_KEY["article"] == "":
        print "You need to register for NYTimes Developer account to run this program."
        print "See Intructor notes for information"
        return False
    params = {"api-key": API_KEY[target], "offset": offset}
    r = requests.get(url, params = params)

    if r.status_code == requests.codes.ok:
        return r.json()
    else:
        r.raise_for_status()

def get_popular(url, kind, days, section="all-sections", offset=0):
    # This function will construct the query according to the requirements of the site
    # and return the data, or print an error message if called incorrectly
    if days not in [1,7,30]:
        print "Time period can be 1,7, 30 days only"
        return False
    if kind not in ["viewed", "shared", "emailed"]:
        print "kind can be only one of viewed/shared/emailed"
        return False

    url = URL_POPULAR + "most{0}/{1}/{2}.json".format(kind, section, days)
    data = query_site(url, "popular", offset)

    return data

def save_file(kind, period):
    # This will process all results, by calling the API repeatedly with supplied offset value,
    # combine the data and then write all results in a file.
    data = get_popular(URL_POPULAR, "viewed", 1)
    num_results = data["num_results"]
    full_data = []
    with codecs.open("popular-{0}-{1}-full.json".format(kind, period), encoding='utf-8', mode='w') as v:
        for offset in range(0, num_results, 20):
            data = get_popular(URL_POPULAR, kind, period, offset=offset)
            full_data += data["results"]

        v.write(json.dumps(full_data, indent=2))

def test():
    titles, urls = article_overview("viewed", 1)
    assert len(titles) == 20
    assert len(urls) == 30
    assert titles[2] == {'Opinion': 'Professors, We Need You!'}
    assert urls[20] == 'http://graphics8.nytimes.com/images/2014/02/17/sports/ICEDANCE/ICEDANCE-
thumbStandard.jpg'

if __name__ == "__main__":
    test()
```

nytimes.py Output: Your program returned the expected data.