Lab/Homework 5

Deadline: 23:59 pm, Sunday, Dec 17.

What to submit:

A report with answers to each exercise and corresponding python program (.py file), packaged into a zip file. Named zip as "class_name_HW5 ",for example, "AI1-jason-HW5". Please submit TA (jiashuo Zheng)

Requirements on Coding:

1. Adding header to each .py file.

"""

xxxx.py
author:
date:
description:
"""

2. Please add a space around the operator and after the comma.
3. Add a blank line between code of different functions
4. Indent your code blocks with 4 spaces. Never use tabs or mix tabs and spaces.

Exercise 5.0 Filemarkets(20pts)

Download the file markets.tsv, which contains geographic information for over 7000 farmers markets in the US. The file is in a tabular form, where each line/row lists data for a farmers market and fields are separated by a single tab (tab-separated-value format). The fields are (in this order): state, market name, street address, city, zip code, longitude, latitude.

Your task is to write a tool that allows users to search for farmers markets in their town or zip code.
(a) Write a function that, given a filename, opens the file in the format described and reads in the data. Each farmers market should be represented as a tuple of strings. The function should return two objects: A dictionary mapping zip codes to lists of such tuples and a dictionary mapping towns to sets of zip codes. Note that you can use return a, b to return two values and result a, result b = function(args) to capture the return values when you call the function.

(b) Write a program that first reads in the data file once (using the function from part (a)), and then asks the user repeatedly to enter a zip code or a town name (in a while loop until the user types "quit"). For each request, the program prints all farmers markets for this town or zip code (using the function from part (b)). If town names are ambiguous (because the town name exists in multiple US states), all entries should be printed.

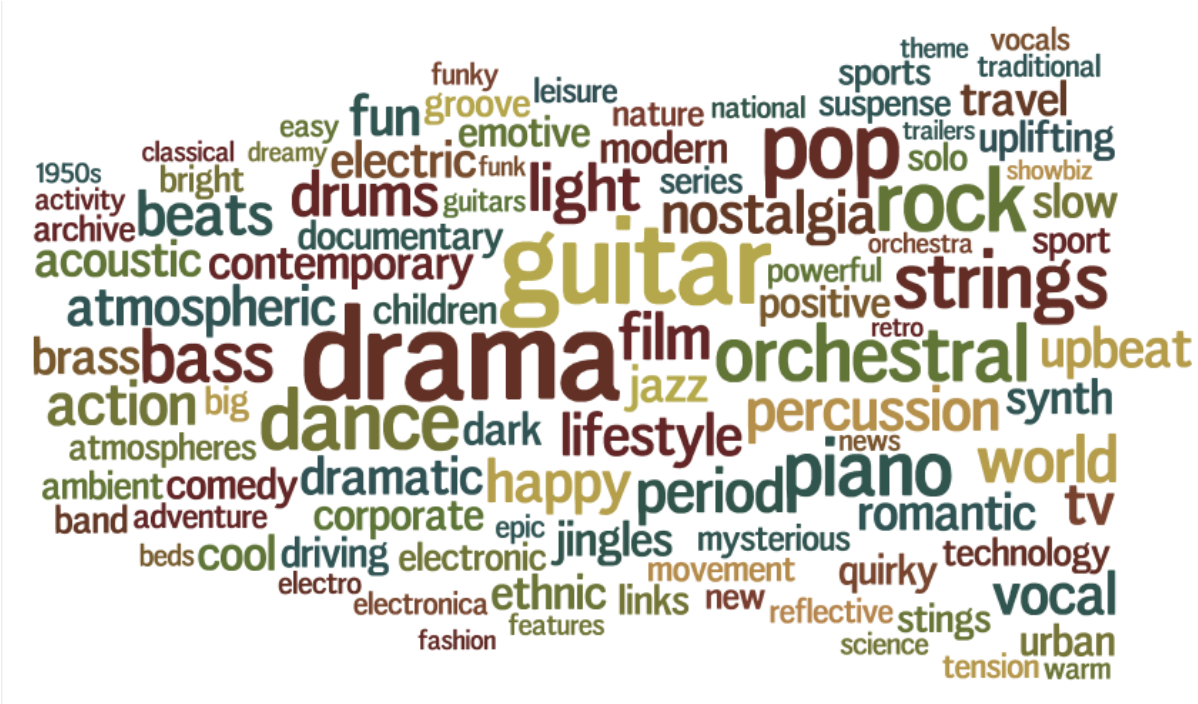
Exercise 5.1 TagClouds(30pts)

For this exercise, you may use Python strings, files, functions, lists, tuples, and dictionaries.

Background:

A tag cloud is a way to represent text data visually. It can help us see what ideas are most important to a writer or speaker by showing us information about words that appear in the data using size, color, and other visual features.

Here is tag showing the hundred words used most often to describe production music tracks, from [a 2012 BBC blog article]():



In a tag cloud, size usually indicates the frequency of words, with more frequent words presented in a larger font. Color and placement can also carry information, such as categories, though they are often used to create a more visually appealing tag cloud.

Of course, many words in a data set may carry little information about important ideas. For example, some words appear frequently in almost every set of spoken or written English. If we tabulate the number of times *a*, *the*, and *of* occur in a document, those numbers will dwarf the occurrences of the actual content of the document. They will also obscure differences among speakers, writers, and data sets. So, most implementations of tag clouds filter out common words, as well as numbers and punctuation.

Data for the exercise:

For this exercise, you will write a Python program to analyze the transcript of a presidential debate and write a text file containing a tag cloud for a candidate based on the words he or she used.

Transcripts:

We will use transcripts of the three 2012 Presidential debates between Pres. Obama and Gov. Romney. These data files are listed as follows:

- 2012-debate-01.txt—the October 3, 2012, debate
- 2012-debate-02.txt—the October 16, 2012, debate
- 2012-debate-03.txt—the October 22, 2012, debate
- 2012-debate-01-partial.txt— a partial transcript of the first debate, which contains only the first five minutes or so of the full transcript

You may want to use the partial transcript for testing. This file is still large (~14,000 characters) but gives you an opportunity to work with a data set smaller than a full debate (~100,000 characters). This can provide you faster feedback than processing a full transcript would.

Notice that the transcript files share a common format. There are three speakers: Pres. Obama, Gov. Romney, and the moderator (Lehrer, Crowley, and Schieffer, respectively). Each time one of the three participants begins to speak, a line is marked with the speaker's name in all caps, followed by a colon -- *OBAMA:*, *ROMNEY:*, and, say, *LEHRER:*. Once such a label appears, all words are attributed to that speaker until another label occurs. Notice that this often does not occur for many lines.

Stop Words:

We need to omit common words that carry no value in your tag cloud. This data file is named as “stop-words.txt”.Each line in the file contains a single word. Any word in the stop-word list should be omitted from the tag cloud. You can omit it entirely from your processing.

All five of these data files are included in the zip file for the homework.

Code for the exercise:

Create a program file named tag_cloud.py, define a function named *main(filepath, spoken_name)* that realize the following tasks:

1. Reads through a debate transcript file.
2. Create a dictionary of words spoken by a specific speaker.
3. Removes all stop words.
4. identifies the 40 most frequently used words by the speaker
5. Write the 40 most frequently used words and its corresponding frequency into one text file.

main(filepath, spoken_name) must take two parameters:

1. a string, the name of a debate transcript file, and
2. a string, the name of the speaker to analyze

For example, *main('2012-debate-01.txt', 'Romney')*.

Your code should produce a file named ‘Romney.txt’ containing the word and its frequency in the first 2012 debate.

Exercise 5.2 TransposeFileContent(20pts)

Write the function TransposeContent(path), which takes the path of a file input by user, transpose its content, and output the newly generated content to another file named as “transpose_content.txt”. You may assume that each row has the same number of columns, and each field is separated by one space ‘ ’ character.

Write the main function to ask the user to input the file path until the file is read.

Example: If Attached “author_paper_stat.csv” has the following content:

"author index AAAI CIKM CVPR ECIR"

"A-Chuan Hsueh 6680 1 0 0 0"

"A. Alzghoul 30643 0 0 0 0"

"A. Bagchi 7741 1 0 0 0"

Then output file has the following content:

author,A-Chuan Hsueh, A. Alzghoul, A. Bagchi

index, 6680 , 30643, 7741

AAAI, 1, 0, 1

CIKM, 0, 0, 0

CVPR, 0, 0, 0

ECIR, 0, 0,0

Exercise 5.3 DataMiningStockPrice(30pts)

In this small project, you will do basic data mining for financial analysis related to stock prices. Please check the attached file TSLA.csv, which shows the daily stock price of tesla in the past five years. The file contains data look like this:

Date,Open,High,Low,Close,Adj Close,Volume
2017-04-24,61.844002,62.110001,61.203999,61.605999,61.605999,25417500
2017-04-25,61.599998,62.796001,61.172001,62.757999,62.757999,33688500
2017-04-26,62.473999,62.900002,61.799999,62.034000,62.034000,23475000
2017-04-27,62.338001,62.618000,61.500000,61.726002,61.726002,17343000
2017-04-28,61.966000,62.959999,61.599998,62.813999,62.813999,22527500

Designing Your Program.

The main program should contains the following functions:

- ***load_daily_data_list(filename)***
  The data is in a CSV (comma-separated value) format. Split each line into a list of values. Remember to convert strings that are really numbers (all but the date) into the appropriate type of number. The resulting list is the record of a single day's stock activity. This function should return a list of daily activity records.

- ***compute_monthly_averages(list_of_daily_records)***
  Use the Date field and the Adj Close field to compute the average price for each month. For each month, create a tuple containing the month's average stock price and the month-year part of the date. We use a tuple so that we don't accidentally change this data later. This function should return a list of monthly average tuples. You may want to break this task down into smaller tasks and write functions to perform the smaller tasks for you.

- ***print_extreme_months(list_of_monthly_tuples)***
  Find the six highest monthly averages and the six lowest monthly averages for the stock. Print each list in order: the highest averages from highest to lowest, and the lowest from lowest to highest. Format the output in columns with informative headers, with the averages shown to two decimal places. This function should not return a value. You may want to break this task down into smaller tasks and write functions to perform the smaller tasks for you.

Name your program as mine_stock_prices.py.