**DSC 450: Database Processing for Large-Scale Analytics**

**Final Project**

Due March 24th at 11PM (No extensions!!)

**Objective:** The objective of the final is to perform data cleaning on restaurant data obtained from web, load into a database, and analyze it using SQL queries and stored procedures. A brief overview of tasks is as follows:

* You will download a restaurant data file and load it into a Pandas dataframe.
* Then you will upload the data into a SQLite database.
* You will read some additional geolocation data downloaded from the web and append to the restaurant data.
* Finally, you will find matching pair from another restaurant data file downloaded from the web.

The above tasks are divided into 3 parts. Read each part. Do them sequentially one after the other. Submissions instructions follow after the part.

**Part A.**

1. Download zagat.txt from D2L. This file has a list of names and address provided by restaurant review company Zagat.
2. Load the file into a Pandas dataframe. If using read\_csv() then make sure its header is set to ‘None’ and you are reading all 331 rows. Assign column names to be ﻿‘line’,‘rname’,‘address’,‘city’,‘phone’,‘cuisine’ , where:
   * ‘line’ refers to the nameaddress string in zagat.txt,
   * ‘rname’ refers to the restaurant name,
   * ‘address’ refers to restaurant address (except for the city),
   * ‘city’ refers to the restaurant city,
   * ‘phone’ refers to the phone number, and
   * ‘cuisine’ refers to the designated cuisine.

Call the dataframe zagat.

1. Extract the 'name' field from ‘line’ attribute. Pandas provides the function [extract()](http://pandas.pydata.org/pandas-docs/stable/text.html#extracting-substrings) that takes a regular expression to create new fields from text data. Since the original name and address strings from Zagat are in the column named ‘line’ in zagat. Then the call:

zagat['line'].str.extract(r'^([^\d]\*)(\d.\*)$', expand=**True**)

**returns a dataframe** with two columns—each corresponding to a group in the regular expression—the first group, ([^\d]\*), containing the longest prefix of the line string that contains no digits, and the second group, (\d.\*), containing the rest of the string. Note, the first group gives the restaurant name. *The columns of the returned dataframe can be appended to the zagat dataframe using the pandas function concat().*

1. Use variations of the above pattern to further extract 'address','city','phone','cuisine'. As you extract, populate the data into zagat.

The following is useful in the data munging process:

* Build the regular expression(s) you need by iterative refinement—look at some of the data, try out an initial regular expression, observe where it works and where it fails, and improve upon it.
* If the regular expression doesn’t match a given string, extract returns null values. So if you want to look at the strings where your regular expression failed, use, e.g., zagat.ix[df.iloc[:,0].isnull()], in which df is the name of the dataframe returned by extract().
* Even so, you may not find a single regular expression that extracts all columns. It may be easier to extract the columns in stages, using separate calls to extract(). Further, there may be special cases that are unlike all other data and are best handled by either changing a few data rows to suit your pattern or hard-coding their values. In the approximately 331 rows in the combined data, I hard-coded the extraction for about 5 restaurants. You should not hard-code more than 10 cases.
* Addresses often start with a street number. But not always, and sometimes the restaurant name itself has a number as well.
* Addresses also end with a period except a few such as “Broadway”. The special period can be used to separate addresses from city names.
* City names often begin after the name of a road, i.e., after a substring ending with “Blvd.”, “Ave.”, “PCH”, or “Broadway”. It is simple to build a complete list of such suffixes by iterative refinement.
* Phone numbers have a regular 3-3-4 digit pattern.
* For large dataframes, python console tends to limit the number of lines that it will pretty print. You can force the console to print all lines by using ﻿print(df.to\_string()), in which df is the name of the dataframe.

**Submission:** You must create a parta.py file which reads zagat.txt and finally prints the zagat dataframe. Submit *parta.py*.

**Part B.**

We will now insert the zagat dataframe into a sqlite table.

1. Connect to Sqlite, create a dsc450.db, and create a ‘Restaurants’ table with the columns ‘rname’,‘address’,‘city’,‘phone’,‘cuisine’  and meaningful datatypes. In this table add two new columns ‘latitude’ and ‘longitude’.
2. Read the geocode.json file. This file has geocoded json for each of the restaurant addresses. Extract latitude and longitude from the json document.
3. Prepare an insert statement for Restaurants in which you will insert data from zagat and ‘latitude’ and ‘longitude’ from geocode.csv. The following will help you in insertion:
   * The geocoded json are in the order of restaurants listed in zagat.txt.
   * Your insertion must take care of SQLInjection issues.
   * Before inserting you must match the address.
   * Report the number of incorrect entries i.e., entries in which address does not match, in a file named ‘errors.txt’.
4. Once inserted answer the following queries:
   * Find the count of restaurants based on city and cuisine. The result should be of the form:

city, cuisine, count

Chicago,      Indian,  55

Chicago, Chinese, 43

San Francisco, Indian, 43

* + Find the average location (‘latitude’ and ‘longitude’) of ‘American’ cuisine restaurants in ‘San Francisco’ city.

**Submission:** Extend your *parta.py* to now connect to database, create tables, read geocode.csv, and create insert statements, which are executed on the sqlite database. Call this new extension *partb.py*. Submit *partb.py*, *dsc450.db*, *errors.txt*, and *answers.sql* (this contains query statements of #7).

You may use Python DB API to connect to database. Using SQLAlchemy is worth a bonus of 10 points.

**Part C.**

We will now check how good a code we wrote in PartA using a bigger dataset and iterating further.

1. Download fodors.txt from D2L. This file has a list of names and address provided by restaurant review company Fodors.
2. Modify your program in Part A to read fodors.txt similar to zagat.txt into a fodors dataframe Note:
   * Phone numbers in fodors.txt have more than two kinds of format.
   * There are sometimes multiple phone numbers.
3. Change the phone number format in fodors dataframe to match the format of zagat dataframe. Then use the phone number string to find matching pairs. Report matching pairs in file matching.txt. Note, Matching.txt has entries of this format:

Arnie Morton's of Chicago 435 S. La Cienega Blvd. Los Angeles 310-246-1501 Steakhouses

Arnie Morton's of Chicago 435 S. La Cienega Blvd. Los Angeles 310-246-1501

American

########################

Art's Deli 12224 Ventura Blvd. Studio City 818-762-1221 Delis

Art's Delicatessen 12224 Ventura Blvd. Studio City 818-762-1221 American

########################

**Submission:** Extend *partb.py* for Part C and submit as *partc.py* You must submit *partc.py* and matching.txt. Submit the no-collaboration statement in a text file.

**All the best!**