STATS 607

ASSIGNMENT 3

PART I

In this exercise you should only use the core Python language and as needed the standard library (not pandas, numpy, etc.).

Here, you will process data from the US National Bridge Inventory (NBI) - https://www.fhwa.dot.gov/bridge/nbi.cfm, then do some analyses with it. The NBI is an annual inventory of all bridges in the United States.

The data download site is https://www.fhwa.dot.gov/bridge/nbi/ascii.cfm, but you can use the script 'nbi_setup.py' to download and process all of the files that you will need. Each state's file has fixed width format, following the specification in https://www.fhwa.dot.gov/bridge/nbi/format.cfm. More detailed information about the variable codes is in https://www.fhwa.dot.gov/bridge/mtguide.pdf.

First, you should write Python code to process all the files for a given year, and construct a list L = [L(1), L(2), ...] containing information about the bridges.
Each L(i) is itself a list representing one bridge with the following data: [structure number, state, year built, year reconstructed, structure length, average daily traffic].
There is a condition defining a highway bridge. You can find it in the bottom of this page: https://www.fhwa.dot.gov/bridge/britab.cfm. You should only include in L the bridges meeting this condition.

Note that when reading the text you will need to convert strings containing numbers into actual numbers. Normally you can use float() for this, but some of the strings are blank. The function below will use NaN values to represent these missing values in numeric form. Note that you will need to use math.isinan() to detect these values when doing the analyses below.

```
def myfloat(x):
    try:
        return float(x)
    except ValueError:
        return float('nan')
```

Once you have written code to construct the L, you should write additional code to answer the following questions (note that "state" can refer to any of the 52 geographic regions labeled as "state" in the dataset):

Questions:

1.1 Which state has the most bridges? Assign the outcome to a variable named 'stateWithMostBridges'. The outcome is a float.

1.2 Determine the average length of bridges in each state. The outcome should be a dictionary named 'avgLenBridges'.

1.3 Determine which states have the shortest and longest average bridge length. Assign the outcome to a tuple with two elements named 'shortLongStates'.

1.4 For bridges that were rebuilt, determine the average duration between the original construction and the reconstruction. Assign the outcome to a variable named 'avgRebuilt'.

1.5 Comparing the average daily traffic values from 2000 to 2010, what proportion of bridges saw increased traffic? Assign the outcome to a variable named 'propIncTraffic'. Note: use values from the years 2000 and 2010 (ignore the year 2005).

1.6 Comparing the average daily traffic values from 2000 to 2010, what was the average percentage change in average daily traffic over all bridges? Assign the outcome to a variable named 'avgPercentChange'. Note: use values from the years 2000 and 2010 (ignore the year 2005).

PART II

No real data will be used this time around… You can use everything you want to solve this exercise, including Numpy and Pandas.

2.1 Create a dataframe named 'dailyValues1' with a random value associated to every single day of a year. The dataframe should have the following columns:

Year: 2018
Month: January, February, …
Day: 1, 2, …
Weekday: Monday, Tuesday, …
Value: A random integer from 0 to 100

Note that the number of rows of the dataframe should be equal to the number of days in the year and the number of columns is 5. Also note that (for column 'Day') January has 31 days, February has 28 days, and so on…

2.2 Create a dataframe named 'dailyValues2' with indices corresponding to month names and columns corresponding to days of the month. The dataframe should have 12 rows (for the month names – type 'object') and 31 columns (the maximum number of days in any given month – type 'int'). The values of the dataframe should be the ones obtained in 2.1 for the particular day of the month. The cell corresponding to days that do not exist for the month should be filled with NaN. For example, February 2018 has only 28 days. Then, there should be 3 NaNs for the row corresponding to the month of February.

PART III

You can use everything you want to solve this exercise, including Numpy and Pandas.

New York City has a vast and complex transportation system, including one of the largest subway systems in the world and a large fleet of taxis. There is a large number of people in NYC that relies on public transportation and take millions of taxi trips per year. I downloaded taxi trip data corresponding to the month of January 2013, and made some additional transformations to it so to make it amenable to analysis without relying on super machines. The data you will be working with is in the file 'trips.csv'. Each row or observation in this data file correspond to one single taxi trip.

The ultimate goal of this exercise is to predict the number of taxi pickups throughout NYC as it changes from day to day, hour to hour and weather conditions. So, given a specific location, date, time, and weather conditions, can we predict the number of pickups with a reasonable high accuracy?

Bonus points will be given if you solve 3.9.

Questions:

3.1 We are only interested in a subset of all taxi trips. Create a dataframe named 'trips1' that retains all trips matching the following conditions:

3.1.1 Latitude is greater than 40 and less than 41.5.
3.1.2 Longitude is greater than -75 and less than -72.
3.1.3 The number of passengers is greater than 0 and less than 5.

3.1.4 The trip time in seconds is greater than 1800.

3.2 Create a tuple named 'tripStats' where each of four elements represents: number of trips in your data; average number of passengers per trip; the average trip time; the standard deviation trip time.

3.3 I've done some reverse geocoding and obtained zip codes for latitudinal and longitudinal coordinates (data is in the file 'zipcodes.csv'). Note that the coordinates in 'Coords' correspond to latitude and longitude in this order 'lat,long'. Add the zip code info data to your data in 'trips1' to create a dataframe named 'trips2'. Note that the coordinates in 'zipcodes.csv' have only two decimal places, so you should round to two decimal places the relevant values in 'trips1'. Now, drop any lines that have at least one missing data.

3.4 Aggregate the data so to create a new column 'Count' with the number of trips per date-hour and zip code. The outcome should be a dataframe named 'trips3'.

3.5 I've also collected weather data for the same period (data is in the file 'weather.csv'). Add weather information to 'trips3' and drop any lines that have at least one missing data to create a new dataframe named 'trips4'.

3.6 Remove from 'trips4' any redundant or unnecessary columns to create a new dataframe named 'trips5'. This new dataframe should contain only the following columns: 'Zipcode'; 'Count'; 'Day'; 'Hour'; 'Weekday'; 'Temp'; 'Icon'; 'Rain'; 'Snow'.

3.7 Remove from 'trips5' zip codes that have less than 2 trips to create a new dataframe named 'trips6'.

3.8 Plot the total number of taxi trips and average temperature per hour (from 0 to 23) in one single figure. For the purpose, create a secondary axis. What do the trends tell you?

3.9 (Optional) Train a negative binomial and a random forest model to predict the number of taxi trip pickups based on the 'Zipcode', 'Weekday', 'Hour', 'Temp' and 'Icon'. Create a tuple named 'rmseResults' with the root mean square errors of both models applied on a test set (use 20% observations on your test set). Note: we will also accept your solution if you decide on an appropriate but different methodology from the ones asked here.

Make sure you have the provided files ('nbi_setup.py' , 'trips.csv', 'zipcodes.csv', 'weather.csv') for the assignment in your current working directory before starting to solve the problems. Your main scripts (one for each part of this assignment) should be called 'test_assignment3_part[1 or 2 or 3]_[your name].py'. Remember, your scripts should properly demonstrate outcomes. Make sure you create the variable names exactly as asked.

Use exceptions and create your own functions for all parts of this assignment as you see fit (you should need it). Define the functions in a separate module (file) named 'assignment3_[your name].py'. For example, my module containing function definitions for all parts of the assignment would be called 'assignment3_Marcio.py'. Your code, including functions, should be well documented. For functions, state the input and output variables as well as their type. For example, see the following docstring:

```
"""
Input:
  a: np.array
  b: string

Output:
  out: tuple
"""
```

Note that your final solution should consist of three .py files - one main script for each of the three parts of the assignment; and one potential additional module file you created with the definition of your functions.

Your code should be efficient and readable. Please make sure it runs without throwing errors on both the development environment you are using and on the command line. You don't need to print your output, for example, L in part 1 or the Dataframes in part 2 or part 3, but please make sure you are creating variable names exactly as asked for and not printing any extra or test output. All this will be taken into account in the grading process. You have until October 22nd at 3pm to upload your solutions via canvas.

Good Luck!