This assignment only needs to be completed by students intending to complete Track B.

PIC 16, Fall 2016 – Assignment 4W

Assigned 10/26/2016. Code (a single `.py` file) due 1 p.m. 11/04/2016 on CCLE. Hand in a printout of this document with the self-assessment portion completed by the end of class on 11/04/2016.

In this assignment, you will write code to analyze the attendance data of MIT's Thirsty Ear Pub from the year 2013 to determine the average number of customers in the pub at a given hour. The raw data, collected using a cell phone app by the Thirsty's door person, is distributed over 100 separate CSV files, each named according to the date and time the count was initiated, typically at the start of a shift. For instance, the data in the file `2013-1-30-19-46-56.csv` began being collected on January 30, 2013 at 7:46 p.m. (shortly after a 7:30 p.m. opening) Eastern Standard Time[1] and is formatted like.

```
1359593216315          0         0
1359593221493          1         1
1359593222349          2         2
      …                …         …
1359598113178         80        80
1359598644931         79        80
1359598645273         78        80
1359598663536         79        81
1359598722919         80        82
```

The first column is a Unix timestamp, expressed as the number of *milliseconds* since time began (12:00 a.m. UTC[2]on January 1, 1970, of course!). You can confirm this by rounding to the nearest second and using the command:

```
>>>import time; time.ctime(1359593216)
'Wed Jan 30 16:46:56 2013'
```

where the output on my machine has automatically been formatted to Pacific time, 3 hours different from Eastern time (as expected). The second column is the instantaneous house count, or the number of patrons in the pub at the time. The third column is the cumulative house count, or the number of unique patrons that visited the pub since the start of the shift. In this analysis, we are only concerned with the instantaneous house count, because we want to know the average number of patrons by time of night. Measurements are recorded whenever a person enters or leaves the pub, not at regular intervals.

Task

Use `pandas` to load and process the data to determine the average number of patrons by day and time of day (in 20 minute intervals). Save an output file like `output.txt`, with one column for day of week and time of night and one for the mean, over the course of the year, instantaneous house count at that time.

Self-Assessment (see "Hints" below for more information)

Does your function `round_time` work? (10 points)
Up to which step does your `process_file` function complete? (10 points steps 2-8)
Does your code concatenate the `DataFrames` from all files? (10 points)
Does your code produce output similar to the example (output.txt)? (10 points)
Indicate your total score:

[1] In 2013, Daylight Savings Time took effect at 2:00 a.m. on 3/10 and ended at 2:00 a.m. on 11/3.

[2] UTC is *not* adjusted for daylight savings time, whereas local time *is*, which complicates things somewhat.

This assignment only needs to be completed by students intending to complete Track B.

Hints:

This assignment is a bear, hence the extra time. I'm going to guide you through it, but you'll need to learn a lot on the fly. I'll suggest functions/methods, but you'll need to look them up and learn to use them. (Think how much harder it was for me the first time, when I didn't even know what functions existed and what they were called!)

Before you start, you need to think through how you will process the data at a high level. The ultimate goal is to determine the average number of patrons in the pub at a given time on a given day of the week. For example, the average number of patrons in the pub at 11:00 on a Thursday night is ~59, at 11:20 on a Thursday, 56, etc.... This information could be used to help set the hours of the pub, determine staffing needs, and many other purposes. In order to calculate these average, we need to know the number of people in the pub at 11:00, 11:20, etc… on several Thursday nights throughout the year. However, none of the timestamps in the data files correspond with 11:00, 11:20, etc… exactly. Therefore, we first need to infer how many people were present at 11:00, 11:20, etc…

For instance, if we load `2013-1-17-21-58-39.csv` into a `DataFrame`, remove any `NaN`s (Not a Numbers) with the help of `notnull()`, and convert the Unix timestamps to dates using `to_datetime`, we will have something like:

```
                       Time  Inst  Cumu

0   2013-01-18 02:58:39.022     0     0

1   2013-01-18 02:58:40.509     1     1

2   2013-01-18 02:58:49.555     2     2
```

Whereas we *want* something like:

```
                           Inst  Cumu             TOD

Time

2013-01-17 21:40:00-05:00     0     0  Thursday-21-40

2013-01-17 22:00:00-05:00    31    37  Thursday-22-00

2013-01-17 22:20:00-05:00    45    51  Thursday-22-20
```

There are several steps in converting the original to what we want.

- We've made the time, rather than an integer, serve as the index.
- We've converted the times to Eastern Time, which is indicated by the (UTC minus) `-05:00`
- We've made the timestamps at intervals of 20 minutes
- We've made timestamps at "round" times (timestamps are on the hour, 20 minutes after, and 40 minutes after), rather than times with
- We've added a column `TOD` (time of day) that we can use to group measurements

There is a particular order these steps need to be accomplished in; some of the order is determined by peculiarities (IMO) in `pandas` rather than common sense. Here are some facts needed to order the steps.

- The `pandas` function `asfreq` can interpolate data at a specified temporal frequency, that is, it can be used to infer the instantaneous and cumulative house count every 20min
- However, `asfreq` can only be applied to `DataFrame`s that use time as their index
- Therefore, we need to convert the timestamps to 20 minute intervals *after* making time the index

This assignment only needs to be completed by students intending to complete Track B.

- Unfortunately, `asfreq` does not allow the user to specify the starting time of interpolation, only the frequency. Rather, it uses the first timestamp of the index as the first time and all future times are multiples of the specified period afterwards. For instance, if the first time in the index is `02:58:39.022` as above, `asfreq` will interpolate the instantaneous and house counts at `3:18:39.022`, `3:18:39.022`, etc…
- Therefore, one solution is to round the first time to a multiple of 20 minutes after an hour. For instance, since the house counts at `02:58:39.022` were `0` and `0`, we can assume that the house count at `02:40:00.000` was also `0`, `0`.
- However, `pandas` does not allow us to alter elements of the index column. Ack!!
- We can easily modify elements in other columns of the `DataFrame`, so we can round the first time *before* the time column is set as the index.
- Note that the first time listed in every `.csv` file is different, so we have to round the first times (and probably interpolate `asfreq`) *before* combining the data from all the `.csv` files
- There is no easily-found function for rounding times, so we have to round the first time of each `DataFrame` when it is still a Unix timestamp (number of ms since 1/1/1970), before it is converted to a readable date.

Based on all this information, I suggest the following:

Write a function `round_time` that rounds a Unix timestamp *down* to a multiple of 20 minutes. (How many milliseconds are in 20 minutes?) For example, 1358477919022 should round to 1358476800000.

Write a function `process_file` that:

1. Reads a single CSV file (*without* setting time as the index) into a `DataFrame` and assigns names to the columns like `Time`, `Inst`, and `Cumu`.
2. Remove any NaNs from the data using `notnull()` to find all the rows with valid numbers and Boolean indexing to effectively eliminate the rest. Remember that you can only select entire rows with Boolean indexing.
3. Use your `round_time` function to round the *first* time in the time column.
4. Set the time column as the index using the `DataFrame`'s `set_index` method.
5. Convert the timestamps to `DateTime` objects, which have a human-readable representation, using `pandas`' `to_datetime` function. You will need to specify some parameters of this function, including `utc = True`, which makes the resulting `DateTime` object "timezone aware". (In other words, it tells the `DateTime` object that it is expressed in UTC. It is necessary for the `DateTime` object to know which time zone it is currently expressed in if we want to convert to another time zone.) Note that you can pass the whole `index` column into `to_datetime` to convert all of it in one fell swoop, and you can set the entire `index` column to the result.
6. Convert the `DateTime`s to the "US/Eastern" time using the `DataFrame`'s `tz_convert` method. This correctly adjusts for daylight savings time!
7. Use the `DataFrame`'s `asfreq` method to interpolate the data at 20 minute intervals. You'll need to read about `DateOffset` objects and consider whether forward filling or back filling the data is more appropriate here (There is a right answer. Considering that measurements are recorded when a person enters or leaves the pub, if there were 32 people in the pub at 10:59 and the next measurement shows 33 people at 11:01, how many were in the pub at 11:00?)
8. Use the `index` Series' `strftime` method to create a string from only the day of the week, hour, and minute of the times, and put this information in a new "`TOD`" column of the `DataFrame`.
9. Finally, return the processed `DataFrame`.

This assignment only needs to be completed by students intending to complete Track B.

Using 2013-1-17-21-58-39.csv as an example, I've included the state of the DataFrame after each step of the process above in ProcessExample.txt.

Use a function from the os module to get the names of all the .csv files in the working directory, process each with process_file, and concatenate all the resulting DataFrames into one big DataFrame.

Use the groupby method of the DataFrame to group all rows with the same entry in the TOD column, returning a DataFrameGroupBy object. You can think of the DataFrameGroupBy object as a dictionary of lists. The keys of the dictionary are the TOD entries. The value corresponding with a key, that is, the list corresponding with a TOD entry, say Friday 19:20, contains the data (instantaneous and cumulative house count) from every row in which the TOD was Friday 19:20. We want to calculate the mean of the Inst column for each of these lists (groups).

The aggregate method of the DataFrameGroupBy object accepts a different sort of dictionary in which each keys is the name of a column of the data and the values is the functions to apply to the data in the column. Use aggregate to calculate the mean of the Inst column for all the groups. You'll need to find the name of a function that calculates the mean of a Series (the instantaneous house counts) yourself.

Save the resulting DataFrame to an Excel file.

What a nightmare!

You will need to find a lot of functions/methods on your own, but I hope this helps you figure out what to look for.

I've created a Jupyter notebook to help you learn most of the functions/methods mentioned above. Perhaps doing it before attempting the assignment itself will help. You don't need to turn it in.

Some hints written before much of the information above (and thus somewhat redundant):

The os module contains a function for listing all the files in a given directory

Sometimes when the excel file loads you will see null (NaN, NaT, etc…) values that didn't appear in the original spreadsheet. You should filter these out before performing analysis. Series objects (columns) in pandas have a notnull() member function to identify non-null values, and you can use this with boolean indexing to effectively eliminate the null values.

pandas has a *function* to_datetime() for converting a Series from a Unix timestamp to a readable date (a DateTime).

Initially, DateTimes are unaware of what time zone they are expressed in. You need to make the DateTime *aware* of what time zone it is correct for ("UTC"). Series' have a method tz_localize() for doing this, or you can do it using an optional argument of the to_datetime when you create the DateTimes originally.

Once your Series of timestamps is time zone aware, use tz_convert() to convert the representation to "US/Eastern" time.

The original timestamps were taken at irregular intervals, whereas we need to know how many people were in the pub in 20 minute intervals. Series objects can resample themselves at a particular frequency using asfreq() method. The first argument is the frequency at which they should be resampled.

This assignment only needs to be completed by students intending to complete Track B.

Unfortunately, the `asfreq` method doesn't allow you to interpolate at *particular* 20 minute intervals, e.g. 7:40, 8:00, 8:20… It interpolates at 20 minute intervals starting at the time of the first measurement. You need to manually round your first measurement to the previous 1/3-hour before using `asfreq`.

You may want to add additional columns to your `DataFrame` containing the day of the week, hour, and minute separately.

You read about the pandas `concat` function to append `DataFrame`s to one another.

`DataFrame` have a `groupby` method that returns an object for which aggregated information can be calculated using the `agg` method. For instance, if I create a Series to represent the time and day for each measurement like "Friday-07:40", I can use `groupby` to collect all measurements taken the same time and day together, then I can use the `agg` method to calculate the mean of the instantaneous house count within each of those groups.