Exercise 1

Due Friday May 21 2021.

This exercise is designed to help you get used to the NumPy and Pandas APIs. To force you to do this, **you may not write any loops** in your code: no `for`, no `while`, no list comprehensions, no recursion. All iteration can be handled by calls to relevant functions in the libraries that will iterate for you.

Some files are provided that you need below: E1.zip.

Python Libraries

Before you get started, you will need some Python libraries installed. See the InstallingPython page for some ways to do that and instructions.

For this week, you'll need at least the `numpy`, `pandas`, `statsmodels`, and `jupyter` libraries. You can install the others mentioned there if you like: they will be used later in the course.

Getting Started with Jupyter

Let's start with a Jupyter Notebook (which you may have previsouly heard of called an iPython Notebook: it has been renamed). If you have the Jupyter package installed, you can run the command “`jupyter notebook`” or if you installed Anaconda, run its “Jupyter Notebook”. See my Jupyter instructions for more information on getting started.

Experiment a little to see how it works. In particular, pressing enter starts a new line of code and control-enter runs the cell where your cursor is, and the ⏩ button in the toolbar will reset the interpreter and run all cells.

The first task will be to do a simple data processing kind of task: create arrays holding data for a sine wave “signal”; simulate a noisy sensor reading that signal; plot both; use a signal processing filter to try to reconstruct the true signal from the noisy data.

Have a look at a screenshot of my notebook, which does this, but has a few parts redacted so you don't get bored. Plot formatting varies a little between versions (or browsers or operating systems or something): if they look slightly different, that's okay.

Create a Jupyter notebook that reproduces my calculations, named `signal-plot.ipynb`.

Getting Started with NumPy

Download the NumPy data archive `monthdata.npz`. This has two arrays containing information about precipitation in Canadian cities (each row represents a city) by month (each column is a month Jan–Dec of a particular year). The arrays are the total precipitation observed on different days, and the number of observations recorded. You can get the NumPy arrays out of the data file like this:

```
data = np.load('monthdata.npz')
totals = data['totals']
counts = data['counts']
```

Use this data to find these things:

- Which city had the lowest total precipitation over the year? Hints: sum across the rows (axis 1); use `argmin` to determine which row has the lowest value. Print the row number.

- Determine the average precipitation in these locations for each month. That will be the total precipitation for each month (axis 0), divided by the total observations for that months. Print the resulting array.
- Do the same for the cities: give the average precipitation (daily precipitation averaged over the month) for each city by printing the array.
- Calculate the total precipitation for each quarter in each city (i.e. the totals for each station across three-month groups). You can assume the number of columns will be divisible by 3. Hint: use the reshape function *(https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html)* to reshape to a 4*n* by 3 array, sum, and reshape back to *n* by 4.

Write a Python program `np_summary.py` that produces the values specified here. Its output (with `print()`) should exactly match the provided `np_summary.txt`. We will test it on a different set of inputs: your code **should not assume** there is a specific number of weather stations. You can assume that there is exactly one year (12 months) of data.

Getting Started with Pandas

To get started with Pandas, we will repeat the analysis we did with Numpy. Pandas is more data-focussed and is more friendly with its input formats. We can use nicely-formatted CSV files, and read it into a Pandas dataframe like this:

```
totals = pd.read_csv('totals.csv').set_index(keys=['name'])
```

This is the same data, but has the cities and months labelled, which is nicer to look at.

Reproduce the values you calculated with NumPy, **except** the quarterly totals, which are a bit of a pain. The difference will be that you can produce more informative output, since the actual months and cities are known. When you print a Pandas DataFrame or series, the format will be nicer.

Write a Python program `pd_summary.py` that produces the values specified here. Its output should exactly match the provided `pd_summary.txt`.

Analysis with Pandas

The data in the provided files had to come from *somewhere*. What you got started with 180MB of data for 2016 from the Global Historical Climatology Network *(https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/global-historical-climatology-network-ghcn)* . To get the data down to a reasonable size, filtered out all but a few weather stations and precipitation values, joined in the names of those stations, and got the file provided as `precipitation.csv`.

The data in `precipitation.csv` is a fairly typical result of joining tables in a database, but not easy to analyse as you did above.

Create a program `monthly_totals.py` that recreates the `totals.csv`, `counts.csv`, and `monthdata.npz` files as you originally got them. The provided `monthly_totals_hint.py` provides an outline of what needs to happen. You need to fill in the `pivot_months_pandas` function (and leave the other parts intact for the next part).

1. Add a column 'month' that contains the results of applying the `date_to_month` function to the existing 'date' column. [You may have to modify `date_to_month` slightly, depending how your data types work out.]
2. Use the Pandas groupby method *(http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html)* to aggregate over the name and month columns. Sum each of the aggregated values to get the total. Hint: `grouped_data.aggregate('sum').reset_index()`
3. Use the Pandas pivot method *(http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.pivot.html)* to create a row for each station (name) and column for each month.
4. Repeat with the 'count' aggregation to get the count of observations.

When you submit, make sure your code is using the `pivot_months_pandas` function you wrote.

Timing Comparison

Use the provided `timing.ipynb` notebook to test your function against the `pivot_months_loops` function that I wrote. (It should import into the notebook as long as you left the `main` function and `__name__ == '__main__'` part intact.)

The notebook runs the two functions and ensures that they give the same results. It also uses the %timeit magic *(https://ipython.org/ipython-doc/3/interactive/magics.html#magic-timeit)* (which uses Python timeit *(https://docs.python.org/3/library/timeit.html)*) to do a simple benchmark of the functions.

Run the notebook. Make sure all is well, and compare the running times of the two implementations.

Questions

Answer these questions in a file `answers.txt`. [Generally, these questions should be answered in a few sentences each.]

1. Where you did the same calculations with NumPy and Pandas, which did you find easier to work with? Which code do you think is easier to read?
2. What were the running times of the two `pivot_months_*` functions? How can you explain the difference?

Submitting

Submit your files through CourSys for Exercise 1.

Updated Fri April 30 2021, 09:17 by ggbaker.