Université de Montréal

IFT-6758-B - A22 - Data Science

Homework 1

Assignment scoring:

Section	Required Files	Score
Getting started with numpy	`np_summary.py`	15
Getting started with pandas	`pd_summary.py`	15
Analysis with pandas	`monthly_totals.py`	65
Timing Comparison	`timing.ipynb`	5

In general your homework will be autograded, i.e. you must **not modify the signature of the defined functions** (same inputs and outputs). For questions involving Numpy and Pandas, you **must** handle any iteration through calls to relevant functions in the libraries. This means you may not use any native Python for loops, while loops, list comprehension, etc.

Submitting

To submit the files, please submit **only the required files** (listed in the above table) that you completed to **gradescope**; do not include data or other miscellaneous files. You do not need to submit all of the files at once in order to run the autograder. For example, if you only completed `np_summary.py`, you don't need to submit the rest of the files in order to get feedback for `np_summary.py`.

Getting Started with Python

You will be doing all of your programming in this course in Python 3.x (not 2!). We would recommend sticking to Python 3.9 or later, but you may have luck with earlier versions of Python 3.

In case you don't already have Python installed on your system, you can checkout the beginners guide for more info.

As a quickstart, for Ubuntu/Debian based systems, you could run:

```
sudo apt-get install python3 python3-dev python3-pip
sudo apt-get build-dep python3-scipy python3-matplotlib
```

Environments

The first thing you should setup is your isolated Python environment. You can manage your environments through either Conda or pip. Both ways are valid, just make sure you understand the method you choose for your system. If you're ever collaborating with someone (i.e. for the project, homeworks are to be done independently), it's best if everyone uses on the same method or you will have to maintain both environment files! Instructions are provided for both methods.

Note: If you are having trouble rendering interactive plotly figures and you're using the pip + virtualenv method, try using Conda instead.

Conda

Conda uses the provided `environment.yml` file. You can ignore `requirements.txt` if you choose this method. Make sure you have Miniconda or Anaconda installed on your system. Once installed, open up your terminal (or Anaconda prompt if you're on Windows). Install the environment from the specified environment file:

```
conda env create --file environment.yml
conda activate ift6758-conda-env
```

After you install, register the environment so jupyter can see it:

```
python -m ipykernel install --user --name=ift6758-conda-env
```

You should now be able to launch jupyter and see your conda environment:

```
jupyter-lab
```

If you make updates to your conda `environment.yml`, you can use the update command to update your existing environment rather than creating a new one:

```
conda env update --file environment.yml
```

You can create a new environment file using the `create` command:

```
conda env export > environment.yml
```

Pip + Virtualenv

An alternative to Conda is to use pip and virtualenv to manage your environments. This may play less nicely with Windows, but works fine on Unix devices. This method makes use of the `requirements.txt` file; you can disregard the `environment.yml` file if you choose this method.

Ensure you have installed the virtualenv tool on your system. Once installed, create a new virtual environment:

```
vitualenv ~/ift6758-venv
source ~/ift6758-venv/bin/activate
```

Install the packages from a requirements.txt file:

```
pip install -r requirements.txt
```

As before, register the environment so jupyter can see it:

```
python -m ipykernel install --user --name=ift6758-venv
```

You should now be able to launch jupyter and see your conda environment:

```
jupyter-lab
```

If you want to create a new `requirements.txt` file, you can use `pip freeze`:

```
pip freeze > requirements.txt
```

Questions

1. Getting Started with NumPy

We will first play with 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 and complete the Python program `np_summary.py`. 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.

2. 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'])
counts = pd.read_csv('counts.csv').set_index(keys=['name'])
```

This is the same data, but has the cities and months labelled, which is nicer to look at. The difference will be that you can produce more informative output, since the actual months and cities are known.

Use this data to complete the Python program `pd_summary.py`. You won't implement the quarterly results because that's a bit of a pain.

3. Analysis with Pandas

3.1. Data cleaning

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. To get the data down to a reasonable size, we 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 as 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).

- 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.]
- Use the Pandas groupby method to aggregate over the name and month columns. Sum each of the aggregated values to get the total.
    - Hint: `grouped_data.sum().reset_index()`
- Use the Pandas pivot method to create a row for each station (name) and column for each month.
- Repeat with the 'count' aggregation to get the count of observations.

3.2. Pairwise distances and correlation

We will now do a quick analysis on computing pairwise distances between stations, as well as seeing if there is any correlation of daily precipitation between stations.

- Complete the `compute_pairwise()` function using `pdist` and `squareform` from the `scipy.spatial` library. We provided a simple test case to help you implement this function (should only be 1-2 lines).
- Use this method along with the already completed `geodesic()` method to implement `compute_pairwise_distance()`, which will return a matrix of pairwise distances between the stations. The point here is to make sure you are correctly pivoting the data.
- Complete the `correlation()` method which computes the correlation between two sets of samples. Note that the equation for correlation is $$\mathrm{corr}(X,Y) = \frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}\,, \quad \sigma_X,\sigma_Y > 0 $$ where μ is the mean and σ is the standard deviation.
- Use `compute_pairwise()` and `correlation()` to compute the correlation matrix of daily precipitation between the stations. Intuitively, two stations would be correlated if it often rains at both stations on the same day (perhaps indicating that they may be located close to each other... or not!). Note that you'll likely get zeros along the diagonal - this is okay (although technically incorrect, as a station should be perfectly correlated with itself).
- Of course pandas can do this for you in pretty much one line; complete `compute_pairwise_correlation_pandas()` with a slightly different pivot table than what you used before, and use the `df.corr()` method to return the correlation matrix. This should match what was returned in your manual implementation (except the diagonal will correctly be ones).

4. Timing Comparison

Use the provided `timing.ipynb` notebook to test your function against the pivot_months_loops function that is provided. (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 (which uses Python timeit) 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. Submit this file; we are simply checking that the timing has been run.

References

This assignment is based off of Greg Baker's data science course at SFU, with several modifications, additions, and reformatting.