Université de Montréal

IFT-6758-B – A22 – Data Science

Homework 1

Grading breakdown:

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 assignment will be graded automatically, which means you **must not** change the signatures of the defined functions (same inputs and outputs).
For the questions that require NumPy and pandas, you **must** handle any iteration via calls to the relevant library functions.
This means you may not use native Python `for` loops, `while` loops, list comprehensions, etc.

Submission

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

Getting Started with Python

You will do all your programming in this course in Python 3.x (not 2!).
We recommend sticking to Python 3.9 or later, though older 3.x versions may also work.

If Python isn't already installed on your system, consult the Beginner's Guide for more information.

As a quick start, for Ubuntu/Debian-based systems you can 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 set up is your isolated Python environment.
You can manage your environments with either Conda or `pip`.
Both approaches are valid—just be sure you understand the method you choose on your system.
If you ever collaborate with someone (i.e., for the project; assignments must be done independently), it's best if everyone uses the same method; otherwise you'll need to maintain both environment files!
Instructions are provided for both approaches.

Note: If you have trouble rendering interactive plotting figures and you used the `pip` + `virtualenv` route, try using Conda instead.

Conda

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

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

After installation, 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 update your Conda `environment.yml`, you can update your existing environment instead of creating a new one:

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

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

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

Pip + Virtualenv

An alternative to Conda is to use `pip` and `virtualenv` to manage your environments.
This may work less well on Windows but works fine on Unix-like systems.
This method uses the `requirements.txt` file; you can ignore `environment.yml` if you choose this route.

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

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

Install packages from a `requirements.txt`:

```
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 environment:

```
jupyter-lab
```

If you want to create a new `requirements.txt`, 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`.
It contains two arrays with information on precipitation in Canadian cities (each row is a city) by month (each column corresponds to a month from January to December of a particular year).
The arrays are the total precipitation observed over various days and the number of observations recorded.
You can extract the NumPy arrays from the data file like this:

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

Use these data and complete the Python program `np_summary.py`.
We will test it on a different input set.
Your code must not assume a specific number of weather stations.
You may assume 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-oriented and more user-friendly with its input formats.
We can use well-formatted CSV files and read them 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'])
```

These are the same data, but the cities and months are labeled, which is nicer to look at.
The difference is that you will be able to produce more informative output since the actual months and city names are known.

Use these data to complete the Python program `pd_summary.py`.
You won't compute quarterly results because that's a bit cumbersome.

3. Analysis with pandas

3.1. Data Cleaning

The data in the provided files had to come from somewhere.
Your data come from 180 MB of 2016 data from the Global Historical Climatology Network.
To reduce the data to a reasonable size, we filtered all weather stations and precipitation values except a few, joined the names of those stations, and obtained the provided file named `precipitation.csv`.

The data in `precipitation.csv` are a fairly typical result of joining tables in a database, but they are not as easy to analyze as the data from the question above.

Create a program `monthly_totals.py` that recreates the files `totals.csv`, `counts.csv`, and `monthdata.npz` as you originally received them.
The file `monthly_totals_hint.py` provides an outline of what needs to happen.
You must complete the function `pivot_months_pandas()` (and leave the other parts intact for the next section).

- Add a `'month'` column that contains the results of applying the `date_to_month` function to the existing `'date'` column.
  [You may need to tweak `date_to_month()` slightly, depending on how your data types behave.]
- Use pandas' `groupby` to aggregate by the `name` and `month` columns. Sum each of the aggregated values to get the totals.
  – Tip: `grouped_data.sum().reset_index()`
- Use pandas' `pivot` to create one row per station (`name`) and one column per month.

- Repeat with the `'count'` aggregation to obtain the number of observations.

3.2. Pairwise Distances and Correlation

We will now do a quick analysis computing pairwise distances between stations, and also check whether there is correlation in daily precipitation between stations.

- Complete the function `compute_pairwise()` using `pdist` and `squareform` from the `scipy.spatial` library.
  We have provided a simple test case to help you implement this function (it should contain only 1–2 lines).
- Use this method together with the already-completed `geodesic()` method to implement `compute_pairwise_distance()`, which will return a matrix of pairwise distances between stations.
  The point here is to ensure you pivoted the data correctly.
- Complete the `correlation()` method that computes the correlation between two sets of samples.
  Note that the correlation equation 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 stations.
  Intuitively, two stations would be correlated if it often rains at both on the same day (which might indicate they are near each other… or not!).
  Note that you will likely get zeros along the diagonal—that's okay (though technically incorrect, since a station should be perfectly correlated with itself).
- Of course, pandas can do this for you in one line; complete `compute_pairwise_correlation_pandas()` with a pivot table slightly different from before, and use `df.corr()` 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 provided `pivot_months_loops` function. (It should import into the notebook as long as you left the main function and the `__name__ == '__main__'` part intact.)

The notebook runs both functions and ensures they produce the same results.
It also uses the `%timeit` magic (which uses Python's `timeit`) to do a simple benchmark of the functions.

Run the notebook.
Make sure everything works and compare the execution times of the two implementations.
Submit this file; we will simply check that the timing ran.

References

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