Université de Montréal

IFT-6758-A - A22 - Science des données

Devoir 1

Notation du devoir:

Section	Required Files	Score
Premiers pas avec numpy	`np_summary.py`	15
Premiers pas avec pandas	`pd_summary.py`	15
Analyse avec pandas	`monthly_totals.py`	65
Comparaison des délais	`timing.ipynb`	5

En général, votre devoir sera noté automatiquement, c'est-à-dire que vous ne devez **pas** modifier la signature du fonctions définies (mêmes entrées et sorties). Pour les questions nécessitant Numpy et Pandas, vous **devez** gérer toute itération via des appels aux fonctions pertinentes dans les bibliothèques. Cela signifie que vous ne pouvez pas utiliser de Python natif les boucles for (loops), les boucles while (loops), la compréhension de liste, etc.

Soumission

Pour soumettre les fichiers, veuillez soumettre **uniquement les fichiers requis** (énumérés dans le tableau ci-dessus) que vous avez remplis sur **gradescope** ; n'incluez pas de données ou d'autres fichiers divers. Vous n'avez pas besoin de soumettre tous les fichiers en même temps pour exécuter l'évaluateur automatique. Par exemple, si vous n'avez rempli que `np_summary.py` , vous n'avez pas besoin de soumettre le reste des fichiers pour obtenir des commentaires sur `np_summary.py` .

Initiation à Python

Vous ferez toute votre programmation dans ce cours en Python 3.x (pas 2 !). Nous vous recommandons de vous en tenir à Python 3.9 ou version ultérieure, mais vous aurez peut-être de la chance avec des versions antérieures de Python 3.

Si Python n'est pas déjà installé sur votre système, vous pouvez consulter le guide du débutant pour plus d'informations.

Comme démarrage rapide, pour les systèmes basés sur Ubuntu/Debian, vous pouvez exécuter:

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

Environnements

La première chose que vous devez configurer est votre environnement Python isolé. Vous pouvez gérer vos environnements via Conda ou pip. Les deux méthodes sont valables, assurez-vous simplement de comprendre la méthode que vous choisissez pour votre système. Si jamais vous collaborez avec quelqu'un (c'est-à-dire pour le projet, les devoirs doivent être faits indépendamment), il est préférable que tout le monde utilise la même méthode ou vous devrez maintenir les deux fichiers d'environnement! Des instructions sont fournies pour les deux méthodes.

Remarque : Si vous rencontrez des difficultés pour rendre les figures d'intrigue interactives et que vous utilisez la méthode pip + virtualenv, essayez plutôt d'utiliser Conda.

Conda

Conda utilise le fichier `environment.yml` fourni. Vous pouvez ignorer `requirements.txt` si vous choisissez cette méthode. Assurez-vous que Miniconda ou Anaconda est installé sur votre système. Une fois installé, ouvrez votre terminal (ou l'invite Anaconda si vous êtes sous Windows). Installez l'environnement à partir du fichier d'environnement spécifié:

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

Après l'installation, enregistrez l'environnement afin que jupyter puisse le voir:

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

Vous devriez maintenant pouvoir lancer jupyter et voir votre environnement conda:

```
jupyter-lab
```

Si vous effectuez des mises à jour de votre conda `environment.yml`, vous pouvez utiliser la commande update pour mettre à jour votre environnement existant plutôt que d'en créer un nouveau:

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

Vous pouvez créer un nouveau fichier d'environnement à l'aide de la commande `create`:

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

Pip + Virtualenv

Une alternative à Conda consiste à utiliser pip et virtualenv pour gérer vos environnements. Cela peut jouer moins bien avec Windows, mais fonctionne bien sur les appareils Unix. Cette méthode utilise le fichier `requirements.txt`; vous pouvez ignorer le fichier `environment.yml` si vous choisissez cette méthode.

Assurez-vous d'avoir installé l'outil virtualenv sur votre système. Une fois installé, créez un nouvel environnement virtuel:

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

Installez les packages à partir d'un fichier requirements.txt:

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

Comme précédemment, enregistrez l'environnement afin que jupyter puisse le voir:

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

Vous devriez maintenant pouvoir lancer jupyter et voir votre environnement conda:

```
jupyter-lab
```

Si vous souhaitez créer un nouveau fichier `requirements.txt`, vous pouvez utiliser `pip freeze`:

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

Questions

1. Premiers pas avec NumPy

Nous allons d'abord jouer avec l'archive de données NumPy `monthdata.npz`. Celui-ci comporte deux tableaux contenant des informations sur les précipitations dans des villes canadiennes (chaque ligne représente une ville) par mois (chaque colonne correspond à un mois de janvier à décembre d'une année particulière). Les tableaux sont les précipitations totales observées durant diverses journées et le nombre d'observations enregistrées. Vous pouvez extraire les tableaux NumPy du fichier de données comme ceci:

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

Utilisez ces données et complétez le programme Python `np_summary.py`. Nous allons le tester sur un ensemble différent d'entrées. Votre code ne doit pas supposer qu'il existe un nombre spécifique de stations météorologiques. Vous pouvez supposer qu'il y a exactement un an (12 mois) de données.

2. Premiers pas avec Pandas

Pour commencer avec Pandas, nous allons répéter l'analyse que nous avons faite avec Numpy. Pandas est plus axé sur les données et est plus convivial avec ses formats d'entrée. Nous pouvons utiliser des fichiers CSV bien formatés et les lire dans une trame de données Pandas comme celle-ci:

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

Ce sont les mêmes données, mais les villes et les mois sont étiquetés, ce qui est plus agréable à regarder. La différence sera que vous pourrez produire une sortie plus informative, puisque les mois et les villes réels sont connus.

Utilisez ces données pour compléter le programme Python `pd_summary.py`. Vous n'appliquerez pas les résultats trimestriels parce que c'est un peu pénible.

3. Analyse avec Pandas

3.1. Nettoyage des données

Les données contenues dans les fichiers fournis devaient provenir de quelque part. Les données que vous avez viennent de 180 Mo de données pour 2016 provenant du Global Historical Climatology Network. Pour réduire les données à une taille raisonnable, nous avons filtré toutes les stations météorologiques et valeurs de précipitations sauf quelques-unes, joint les noms de ces stations et obtenu le fichier fourni sous le nom de "precipitation.csv".

Les données dans `precipitation.csv` sont un résultat assez typique de la jointure de tables dans une base de données, mais ne sont pas aussi faciles à analyser que les données de la question ci-dessus.

Créez un programme `monthly_totals.py` qui recrée les fichiers `totals.csv`, `counts.csv` et `monthdata.npz` tels que vous les avez obtenus à l'origine. Le fichier `monthly_totals_hint.py` fournit un aperçu de ce qui doit se passer. Vous devez remplir la fonction `pivot_months_pandas()` (et laisser les autres parties intactes pour la partie suivante).

- Ajoutez une colonne 'month' (mois) qui contient les résultats de l'application de la fonction date_to_month à la colonne 'date' existante. [Vous devrez peut-être modifier légèrement `date_to_month()`, selon la façon dont vos types de données fonctionnent.]

- Utilisez la méthode Pandas groupby pour agréger les colonnes de nom et de mois. Additionnez chacune des valeurs agrégées pour obtenir le total.
    - Astuce: `grouped_data.sum().reset_index()`
- Utilisez la méthode pivot Pandas pour créer une ligne pour chaque station (name) et une colonne pour chaque mois.
- Recommencez avec l'agrégation 'count' pour obtenir le nombre d'observations.

3.2. Distances et corrélation par paires

Nous allons maintenant faire une analyse rapide sur le calcul des distances par paires entre les stations, ainsi que voir s'il existe une corrélation des précipitations quotidiennes entre les stations.

- Complétez la fonction `compute_pairwise()` en utilisant `pdist` et [`squareform`](https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.squareform.html) de la bibliothèque `scipy.spatial`. Nous avons fourni un cas de test simple pour vous aider à implémenter cette fonction (ne devrait contenir que 1 à 2 lignes).
- Utilisez cette méthode avec la méthode `geodesic()` déjà complétée pour implémenter `compute_pairwise_distance()`, qui renverra une matrice des distances par paires entre les stations. Le point ici est de s'assurer que vous pivotez correctement les données.
- Complétez la méthode `correlation()` qui calcule la corrélation entre deux ensembles d'échantillons. Notez que l'équation de corrélation est $$\mathrm{corr}(X,Y) = \frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}\ ,, \quad \sigma_X,\sigma_Y > 0 $$ où μ est la moyenne et σ est l'écart-type.
- Utilisez `compute_pairwise()` et `correlation()` pour calculer la matrice de corrélation des précipitations quotidiennes entre les stations. Intuitivement, deux stations seraient corrélées s'il pleut souvent aux deux stations le même jour (indiquant peut-être qu'elles peuvent être proches l'une de l'autre... ou pas ! ou (https://fr.wikipedia.org/wiki/Cum_hoc_ergo_propter_hoc). Notez que vous obtiendrez probablement des zéros le long de la diagonale - ce n'est pas grave (bien que techniquement incorrect, car une station doit être parfaitement corrélée avec elle-même).
- Bien sûr, pandas peut le faire pour vous en une seule ligne ; complétez `compute_pairwise_correlation_pandas()` avec un tableau croisé dynamique légèrement différent de celui que vous utilisiez auparavant, et utilisez le `df.corr()` pour renvoyer la matrice de corrélation. Cela devrait correspondre à ce qui a été renvoyé dans votre implémentation manuelle (sauf que la diagonale sera correctement des uns).

4. Comparaison des délais

Utilisez le cahier `timing.ipynb` fourni pour tester votre fonction par rapport à la fonction pivot_months_loops fournie. (Il devrait importer dans le cahier tant que vous avez laissé la fonction principale et la partie `__name__ == '__main__'` intactes.)

Le notebook exécute les deux fonctions et s'assure qu'elles donnent les mêmes résultats. Il utilise également la magie `%timeit` (qui utilise Python timeit) pour faire un simple benchmark des fonctions.

Exécutez le cahier. Assurez-vous que tout va bien et comparez les temps d'exécution des deux implémentations. Soumettez ce fichier; nous vérifions simplement que le chronométrage a été exécuté.

Références

Ce devoir est basé sur le cours de science des données de Greg Baker à SFU, avec plusieurs modifications, ajouts et reformatage.