Le Mans Université, Master 1 économie

Python – TP3 : Un peu de données – 4h

Pandas (https ://pandas.pydata.org/) est la librairie incontournable pour manipuler les données. Elle permet de manipuler aussi bien les données sous forme de tables qu'elle peut récupérer ou exporter en différents formats. Elle permet également de créer facilement des graphes.

I Manipulations avec Pandas

La librairie pandas implémente la classe DataFrame. C'est une structure de table, chaque colonne porte un nom et contient un seul type de données. C'est très similaire au langage SQL. A noter qu'il existe aussi une classe Series, plus ressemblante à une colonne Excel.

Création d'un dataframe

Il existe une grande variété de méthodes pour créer un DataFrame. Voici les deux principaux. Le premier : une liste de dictionnaires. Chaque clé est le nom de la colonne.

```
import pandas as pd

rows = [{'col1': 0.5, 'col2': 'schtroumph'},
        {'col1': 0.6, 'col2': 'schtroumphette'}]
pd.DataFrame(rows)
```

On peut aussi importer les données en lisant un fichier. Récupérer le fichier "data1.csv" sur UMTICE. Pour vérifier qu'on est dans le bon répertoire on pourra utiliser :

```
import os
os.getcwd()
```

Lire le fichier avec la commande `read.csv`.

La maîtrise des index

Les index fonctionnent à peu près comme numpy (https ://numpy.org/) mais offre plus d'options puisque les colonnes mais aussi les lignes ont un nom.

Accès par colonne

```
df['col1']
df[['col1', 'col2']]
```

Accès par ligne

```
df[:1]
```

Accès par positions ou positions entières

```
df.loc[0, 'col1']
df.iloc[0, 0]
```

La création d'un dataframe donne l'impression que les index des lignes sont des entiers mais cela peut être changer. Il faut se souvenir de cette particularité lors de la fusion de tables.

```
dfi = df.set_index('col2')
dfi
```

Les colonnes sont nommées. On peut les renommer.

```
df.columns
df.columns = ["valeur", "nom"]
```

Lien vers numpy

`pandas` utilise `numpy` pour stocker les données. Il est possible de récupérer des matrices depuis des DataFrame avec `values`.

```
df.values
df[['valeur']].values
```

La maîtrise du nan

`nan` est une convention pour désigner une valeur manquante.

```
rows = [{'col1': 0.5, 'col2': 'schtroumph'},{'col2': '
    schtroumphette'}]
pd.DataFrame(rows)
```

Création de colonnes, modifications de valeurs

On peut facilement créer de nouvelles colonnes.

```
df['sup055'] = df['valeur'] >= 0.55
print(df)
df['sup055'] = (df['valeur'] >= 0.55).astype(numpy.int64)
print(df)
df['sup055+'] = df['valeur'] + df['sup055']
print(df)
```

On peut modifier les valeurs une à une en utilisant les index. Les notations sont souvent intuitives. Elles ne seront pas toutes détaillées. Ci-dessous un moyen de modifer certaines valeurs selon une condition.

```
df.loc[df['nom'] == 'alpha', 'sup055+'] += 1000
print(df)
```

Une erreur ou warning fréquent.

```
rows = [{'col1': 0.5, 'col2': 'schtroumph'},
        {'col1': 1.5, 'col2': 'schtroumphette'}]
df = DataFrame(rows)
df1 = df[df['col1'] > 1.]
print(df1)
df1["col3"] = df1["col1"] + 1.
print(df1)
```

`A value is trying to be set on a copy of a slice from a DataFrame.` : Par défaut, l'instruction `df[df['col1'] > 1.]` ne crée pas un nouveau DataFrame, elle crée ce qu'on appelle une vue pour éviter de copier les données. Le résultat ne contient que l'index des lignes qui ont été sélectionnées et un lien vers le dataframe original. L'avertissement stipule que *pandas* ne peut pas modifier le dataframe original mais qu'il doit effectuer une copie.

La solution pour faire disparaître ce warning est de copier le dataframe.

```
df2 = df1.copy()
df2["col3"] = df2["col1"] + 1.
```

La maîtrise des fonctions

Les fonctions de pandas créent par défaut un nouveau dataframe plutôt que de modifier un dataframe existant. Cela explique pourquoi parfois la mémoire se retrouve congestionnée.

- **création** : `read_csv`, `read_excel`
- **index** : `set_index`, `reset_index`
- **utilitaires** : `astype`, `isna`, `fillna`, `to_datetime`, `dtypes`, `shape`, `values`, `head`, `tail`, `isin`, `drop`
- **concaténation** : `concat`
- **SQL** : `filter`, `groupby`, `join`, `merge`
- **calcul** : `sum`, `cumsum`, `quantile`, `var`

II Cas pratiques

Exercice 1 : Données COVID

On récupère les données du COVID par région et par âge à l'adresse "Données hospitalières relatives à l'épidémie de COVID-19" (https//www.data.gouv.fr/). Le fichier voulu est sur UMTICE : `covid-hospit.csv`.

1. En utilisant la fonction `tail` (ou `head`), visualiser la structure du tableau. Quelles sont les différentes données et leur type (utiliser `dtypes`).
2. Les dates sont considérées comme des chaînes de caractères. Il est plus simple pour réaliser des opérations de convertir la colonne sous forme de dates (utiliser `to_datetime`).
3. On supprime les colonnes relatives aux départements et au sexe puis on aggrège par jour.

```
agg_par_jour = covid.drop(['dep', 'sexe'], axis=1).groupby(
    'jour').sum()
print(agg_par_jour.tail())
```

4. Enfin on trace les données (utiliser l'option `logy` pour une échelle logarithmique).

```
agg_par_jour.plot(title="Evolution des hospitalisations par
    jour")
```

5. Refaire le même graphique pour votre sexe.
6. Faire de même avec les séries différenciées puis avec des séries lissées sur 7 jours (fonctions `diff, rolling, mean`).

Exercice 2 : Températures

1. Récupérez sur UMTICE le fichier `temperature.csv`.
2. Utiliser la fonction `describe` sur le dataframe. Que fait cette fonction ?
3. Créer un nouveau dataframe ne contenant que les mois de mars, juin, septembre et décembre et en supprimant les villes de la région 'East'.
4. Récupérer les données sous numpy et calculer la moyenne des températures pour chaque mois. Déterminer aussi la matrice de corrélation entre les 4 mois de l'année.

Exercice 3 : Publicité

1. Importez le jeu de données `Advertising.csv` sous Python puis décrivez sommairement vos variables.
2. Vérifier qu'il n'y a pas de valeur nulle, et étudier les corrélations entre les variables. On pourra importer `seaborn` et utliser `seaborn.heatmap`.
3. Représenter sur un graphique Sales en fonction de TV. Effectuer une régression linéaire des deux variables (utiliser `scipy.stats.linregress`). Commenter les sorties et préciser les coefficients.
4. Étudier la pertinence ainsi que la qualité de l'ajustement.