Traitement des données de séries temporelles avec Pandas

Pandas : Un manuel complet pour les Data Lovers · https://moncoachdata.com/tutos/traitement-des-donnees-de-series-temporelles-avec-pandas/

Les **séries temporelles avec Pandas** (*time series*) sont des points de données collectés ou enregistrés à des intervalles de temps précis : ventes quotidiennes, cours d'une action, mesures d'un capteur... Pandas propose un support robuste pour travailler avec les dates et les heures.

Travailler avec des dates et heures (datetime)

Les données de type date et heure sont essentielles pour analyser des séries temporelles. La fonction `pd.to_datetime()` convertit une colonne de dates (souvent lue comme du texte) en véritable type `datetime`. Tu peux ensuite extraire facilement l'année, le mois, le jour ou l'heure grâce à l'accesseur `.dt`.

```python
import pandas as pd

# DataFrame avec une colonne de dates stockées en texte
data = {'DateTime': ['2023-01-01 08:30:00', '2023-02-01 14:45:00', '2023-03-01 20:15:00']}
df = pd.DataFrame(data)

# Convertir la colonne 'DateTime' en véritable type datetime
df['DateTime'] = pd.to_datetime(df['DateTime'])

# Extraire l'année, le mois, le jour et l'heure
df['Year'] = df['DateTime'].dt.year
df['Month'] = df['DateTime'].dt.month
df['Day'] = df['DateTime'].dt.day
df['Hour'] = df['DateTime'].dt.hour

print(df)
```

Python

```
            DateTime  Year  Month  Day  Hour
0 2023-01-01 08:30:00  2023      1    1     8
1 2023-02-01 14:45:00  2023      2    1    14
2 2023-03-01 20:15:00  2023      3    1    20
```

Rééchantillonnage et décalage

Le rééchantillonnage (resampling)

Le **rééchantillonnage** consiste à changer la fréquence de tes données temporelles. Il sert principalement à :

- **Agréger :** tu as des données à haute fréquence (quotidiennes, par exemple) et tu veux les agréger à une fréquence plus basse (mensuelle, par exemple) pour obtenir une vue d'ensemble ;
- **Interpoler :** tu as des données à intervalles irréguliers et tu veux les ramener à une fréquence régulière, pour l'analyse ou la visualisation (en comblant les trous avec `.interpolate()` par exemple).

Dans cet exemple, on rééchantillonne des ventes quotidiennes à une fréquence mensuelle, en calculant la somme des ventes de chaque mois :

```python
# Ventes quotidiennes sur 40 jours
data = {'Date': pd.date_range(start='2023-01-01', periods=40, freq='D'),
        'Sales': [i for i in range(40)]}
df = pd.DataFrame(data)

# Rééchantillonner à une fréquence mensuelle ('ME' = fin de mois)
monthly_sales = df.resample('ME', on='Date').sum()
print(monthly_sales)
```

Python

```
            Sales
Date
2023-01-31    465
2023-02-28    315
```

À savoir : l'ancienne fréquence `'M'` est remplacée par `'ME'` (*month end*) dans les versions récentes de Pandas.

Le décalage (shifting)

Le **décalage** (*shifting*) déplace les points de données vers l'avant ou vers l'arrière dans le temps. Il sert souvent à calculer des différences ou à créer des variables temporelles. Deux cas d'usage courants :

- **Calculer des différences :** la différence entre la valeur actuelle et une valeur passée ou future, utile pour repérer des tendances ou des changements ;
- **Créer des décalages temporels** (*time lag*) : en décalant une variable, tu analyses comment ses valeurs passées influencent les résultats futurs.

Dans cet exemple, on calcule la variation du prix sur un jour en soustrayant le prix de la veille du prix du jour :

```python
# Cours quotidien d'une action
data = {'Date': pd.date_range(start='2023-01-01', periods=5, freq='D'),
        'Price': [100, 105, 110, 108, 112]}
df = pd.DataFrame(data)

# Variation sur un jour (décalage d'une ligne)
df['Price_Change'] = df['Price'] - df['Price'].shift(1)
print(df)
```

Python

```
        Date  Price  Price_Change
0 2023-01-01    100           NaN
1 2023-01-02    105           5.0
2 2023-01-03    110           5.0
3 2023-01-04    108          -2.0
4 2023-01-05    112           4.0
```

Statistiques glissantes (rolling)

Les **statistiques glissantes** (ou *rolling*, calculs roulants) appliquent une fonction statistique à une fenêtre de taille fixe qui se déplace d'un pas à la fois dans le jeu de données. Elles atténuent

le bruit des séries temporelles et rendent les tendances plus visibles, comme avec les moyennes mobiles.

- **Taille de la fenêtre :** elle détermine le nombre de points pris en compte dans chaque calcul. Une fenêtre de 3 considère le point actuel et les deux précédents ;
- **Fonction :** tu choisis ce qui est calculé dans la fenêtre : la moyenne, la somme, l'écart type, etc.

Ici, on calcule la moyenne mobile du prix sur une fenêtre de 3 jours. Les deux premières lignes valent `NaN`, car la fenêtre n'est pas encore remplie :

```python
data = {'Date': pd.date_range(start='2023-01-01', periods=10, freq='D'),
        'Price': [100, 105, 110, 108, 112, 115, 118, 120, 122, 125]}
df = pd.DataFrame(data)

# Moyenne mobile sur 3 jours
df['Rolling_Mean'] = df['Price'].rolling(window=3).mean()
print(df)
```

Python

```
        Date  Price  Rolling_Mean
0 2023-01-01    100           NaN
1 2023-01-02    105           NaN
2 2023-01-03    110    105.000000
3 2023-01-04    108    107.666667
4 2023-01-05    112    110.000000
5 2023-01-06    115    111.666667
6 2023-01-07    118    115.000000
7 2023-01-08    120    117.666667
8 2023-01-09    122    120.000000
9 2023-01-10    125    122.333333
```

Tu sais maintenant manipuler des dates, rééchantillonner, décaler et lisser une série temporelle. La suite de la série porte sur les données catégorielles avec Pandas. Documentation : séries temporelles dans Pandas.

Pour aller plus loin, consulte aussi notre guide des 10 astuces Pandas en Python.

Les séries temporelles se maîtrisent sur des données réelles. Dans le cours Analyse et visualisation de données, tu travailles par exemple sur le trafic d'une autoroute ou des taux de

change.

