Séance du 26 janvier 2023

On continue notre exploration des `DataFrame`

Les exemples sont à tester dans le **shell** IDLE ou dans la **console** spyder3. Je ne mets plus les prompts afin de faciliter la copie dans votre environnement

Modifier certaines valeurs d'une colonne

On repart avec un version propre de nos deux tables

```
dates = pd.date_range("2020-01", end="2022-06", freq="W-MON")
categories = "trottinette vélo voiture marche tram bus train".split()
true_false = (dates.month > 3) * (dates.month < 7)
delay = (pd.Timestamp('now') - dates).round('D')
big_df = pd.DataFrame(
    {'mobility': pd.Categorical(random.choices(categories, k=dates.size)),
    'start': 0,
    'stop': 0.1 * np.array(random.choices(list(range(100)), k=dates.size)),
    'delay':delay, 'date':dates, 'target':true_false})

ventes = pd.read_csv("sales.csv")
```

On va commencer par mettre une colonne "couleur" dans `big_df` puis on va remplacer tous les `NaN` par la valeur 'noire'

```
big_df['couleur'] = pd.Series({0:'green', 25:'yellow', 100:'red'})
big_df.couleur.fillna('noire', inplace=True)
```

On réalise que l'on aurait voulu la valeur 'black'. Pour faire l'opération proprement, on va:
1. utiliser la commande `.loc`
2. spécifier les lignes que l'on souhaite sélectionner
3. spécifier la colonne où l'on souhaite faire la modification
4. préciser la valeur que l'on souhaite mettre

```
big_df.loc[big_df.couleur=='noire', 'couleur'] = 'black'
```

Mais on dispose aussi, d'une commande `replace` à qui on peut fournir un dictionnaire contenant, pour chaque valeur à changer, la valeur de substitution. On aurait donc pu écrire

```
big_df.couleur.replace({'noire':"black"}) # attention nouvelle série
```

Supposons que l'on souhaite construire la série des sommes cumulées de la colonne "stop" uniquement lorsque la couleur est "black"

```
big_df[big_df.couleur=='black'].stop.cumsum()
```

Corrélation et covariance et autres

On dispose d'une table ventes, et on va supposer que toutes les informations portent sur une seule source, qui aurait évalué (disons chaque semaine) son investissement dans la pub et le bénéfice engrangé. Imaginons qu'elle souhaite suivre l'évolution en termes de "pourcentage d'évolution"

```
analyse = ventes.pct_changes()
analyse.columns
```

Une fois cette nouvelle table construite, on peut souhaiter connaître s'il y a corrélation entre 'TV' et 'Sales'

```
analyse.TV.corr(analyse.Sales)
```

Pour la covariance, cela marche de la même manière en utilisant `.cov`

On peut souhaiter avoir les corrélations d'une valeur particulière, par rapport à toutes les autres

```
analyse.corrwith(analyse.Sales)
```

Enfin on peut aussi vouloir toutes les valeurs 2 à 2

```
analyse.corr()
analyse.cov()
```

tri et classement

- On peut classer par index `.sort_index` et dans ce cas, on peut préciser le paramètre `axis` 0 désigne les lignes, 1 désigne les colonnes (par défaut `axis=0`) et si on veut un ordre spécifique `ascending` False ou True (par défaut `ascending=True`)
- On peut classer par valeur `.sort_values`, on doit définir la critère de tri (la colonne, à l'aide de `by=`) `analyze.sort_index()` `analyse.sort_values(by"TV")`
- On peut classer les valeurs en utilisant `.rank`, on peut décider si l'ordonnancement se fait de manière croissante ou décroissante (via le paramètre `ascending`), mais on peut aussi préciser par quelle méthode on souhaite que le classement résolve les égalités, par défaut c'est la méthode `average` (la moyenne des rangs) qui est utilisée

mais on peut (via le paramètre `method`) spécifier `average`, `min`, `max`, `first` (la dernière méthode stipule que c'est par ordre d'apparition dans les données)

Exercice

Quel est dans `ventes` le numéro de l'enregistrement classé au 7ème rang si on trie par ordre décroissant en fonction de la colonne "TV" ?
Classez `big_df` en fonction de la colonne "mobility", quelle est la valeur de la variable "stop" pour le premier enregistrement ?
Utilisez `sort_index` sur les colonnes de la table `big_df`, quel est l'ordre des colonnes ?

détection et filtre des valeurs aberrantes

```
import numpy as np
np.random.seed(42) # graine pour tirage aléatoire controlé
valeurs = np.random.randn(100, 4) # matrice 100x4 de valeurs aléatoire
df = pd.DataFrame(valeurs, columns=[x*3 for x in 'ABCD'])
ages = random.choices(list(range(16,125)), k=100)
df['age'] = ages
df.columns
```

Supposons que vous ne souhaitiez avoir de valeurs dans les colonnes (autre que âge) que dans $[-2.5, 2.5]$, le but est donc de caper les valeurs

```
small = df.drop('age', axis=1)
small[np.abs(small)>2.5]= np.sign(small)*2.5
```

Exercice

Dans le fichier `ventes`, ajoutez une colonne "pub" qui a pour valeur la moyenne des 3 médias. Construisez alors une colonne "cout" qui vaudra "gros" si pub est dans le dernier quartile, "micro" si pub est dans le premier quartile et "moyen" sinon

discrétisation et regroupement

`.cut` et `.qcut`

`.cut`

On peut découper les valeurs en catégories la notation $(40, 65]$ signifie 40 exclu, 65 inclus de même $[40, 65)$ signifie 40 inclus, 65 exclu. Le découpage par défaut étant $(x, y]$, pour

indiquer que vous voulez l'autre, il faut mettre `right=False`, on a aussi la possibilité de nommer les catégories avec `labels=...`

```
bins = [16, 18, 25, 35, 60, 100]
tags = "ados jeunes juniors adultes séniors".split()
cat1 = pd.cut(ages, bins)
cat2 = pd.cut(ages, bins, right=False, labels=tags)
```

On peut, au lieu de spécifier les bornes, donner le nombre de découpages, dans ce cas, chaque intervalle aura une répartition voisine

```
cat3 = pd.cut(ages, 5, precision=1) # nombre de chiffres décimaux
cat3.value_counts()
```

`.qcut`

On peut faire un découpage en fonction des quantiles et on peut aussi préciser les quantiles que l'on souhaite

```
cat4 = pd.qcut(ages, 4)
cat4.value_counts()
cat5 = pd.qcut(ages, [0, .1, .4, .6, .8, 1.], labels=tags)
```

one hot encoding aka dummy encoding

```
pd.get_dummies(cat1)
pd.get_dummies(cat5, prefix="cat")
```

Opérations d'ajout et de jointures `concat`, `merge`

> **Note**
>
> L'opération `concat` est à privilégier quand vous faites un ajout (que ce soit en ligne axis=0) ou en colonne (axis=1).
> L'opération `merge` est plutôt réservée à la fusion (regroupement d'information similaire), c'est l'équivalent de la commande `join` des bases de données

```
s1 = pd.Series([0,1], index=['a', 'b'])
s2 = pd.Series([2,3,4], index=list('cde'))
s3 = pd.Series([5,6], index=list('xy'))
pd.concat([s1,s2,s3]) # concat lignes
pd.concat([s1,s2,s3], axis=1) # concat colonnes
s4 = pd.concat([s1*5, s3])
pd.concat([s1, s4], axis=1)
```

```
pd.concat([s1, s4], axis=1, join='inner') # intersection lignes
pd.concat([s1, s4], axis=1, join='outer') # comportement par défaut
```

Cela fonctionne aussi avec les tables

```
t1 = pd.DataFrame({'a': [1,2], 'b':[3,4]})
t2 = pd.DataFrame({'a': [11,12,13], 'c':[23,24,25]})
pd.concat([t1,t2]) # concaténation ligne, union colonne
pd.concat([t1,t2], axis=1) # concatenation colonne, union ligne
pd.concat([t1,t2], join='inner') # concat lignes, intersection colonnes
pd.concat([t1,t2], axis=1, join='inner') # concat colonnes, intersection
lignes
```

La syntaxe de la commande `concat`

```
concat(objs: 'Iterable[NDFrame] | Mapping[Hashable, NDFrame]', axis: 'Axis'
= 0, join: 'str' = 'outer', ignore_index: 'bool' = False, keys=None,
levels=None, names=None, verify_integrity: 'bool' = False, sort: 'bool' =
False, copy: 'bool' = True) -> 'DataFrame | Series'
    Concatenate pandas objects along a particular axis with optional set
logic
    along the other axes.
```

Note

L'option `verify_integrity=True` est coûteuse, elle vérifie qu'il n'y a pas de doublons le long de l'axe
Pour créer un nouvel index pour la table (série) résultante utilisez `ignore_index=True`

La commande `merge` effectue une gestion des collisions

```
pd.merge(t1, t2, left_index=True, right_index=True)
pd.merge(t1, t2, how='outer', left_index=True, right_index=True)
```

Après fusion, il est en général nécessaire, de s'assurer que les colonnes sont compatibles afin de pouvoir supprimer les informations redondantes.et de régler les éventuels conflits d'informations. On suppose que des colonnes ayant le même nom dans chaque table, représente le même paramètre.

La syntaxe de la commande `merge`

```
merge(left: 'DataFrame | Series', right: 'DataFrame | Series', how: 'str' =
'inner', on: 'IndexLabel | None' = None, left_on: 'IndexLabel | None' =
None, right_on: 'IndexLabel | None' = None, left_index: 'bool' = False,
right_index: 'bool' = False, sort: 'bool' = False, suffixes: 'Suffixes' =
```

```
('_x', '_y'), copy: 'bool' = True, indicator: 'bool' = False, validate: 'str
| None' = None) -> 'DataFrame'
    Merge DataFrame or named Series objects with a database-style join.

    A named Series object is treated as a DataFrame with a single named
column.

    The join is done on columns or indexes. If joining columns on
    columns, the DataFrame indexes *will be ignored*. Otherwise if joining
indexes
    on indexes or indexes on a column or columns, the index will be passed
on.
    When performing a cross merge, no column specifications to merge on are
    allowed.
```

option	action	défaut
how	type de fusion 'inner', 'outer', 'left', 'right', 'cross'	'inner'
on	précise la ou les colonnes sur lesquels on fait la jointure, **doivent** être présents dans les deux DF	None
indicator	booléen, à vrai, ajoute une colonne indiquant comment a été fait le merge	False
validate	peut prendre les valeurs '1:1', '1:m', 'm:1', 'm:m' vérifie la nature de la jointure	None

Application

```
coeff = pd.Series([1,1,2,2,.5], index=['ecoge', 'scico', 'math', 'info',
'option'])
maths = pd.DataFrame({'userID':[8,4,2,7,1,5], 'eval':[12,10,5,8,13,17]})
info = pd.DataFrame({'userID':[1,2,4,3,5,7], 'eval':[10,5,8,13,15,11]})
ecoge = pd.DataFrame({'userID':[1,4,3,8], 'eval': [14, 17, 12, 10]})
scico = pd.DataFrame({'userID': [6,5,2,7], 'eval': [17, 13, 9, 11]} )
option = pd.DataFrame({'userID': range(1,9), 'eval': [14]*8 })
report = pd.DataFrame({'userID': [6,8,6], 'note': [10, 13, 12], 'matiere':
['maths', 'info', 'info']})
```

Exercice

Vous devez sortir le classement d'une promotion, déterminer quels sont ceux qui valident ou pas leur année. Pour valider son année, il faut avoir la moyenne sur l'ensemble des matières. On suppose qu'une personne inscrite en "ecoge" n'est pas inscrite en "scico" et réciproquement, par ailleurs toutes les autres matières sont obligatoires, une note manquante est un **0**. Les reports, sont les notes attribuées à une session antérieure.