TP 02

Objectifs :
— librairie pandas : manipulation et agrégation
— visualisation des données
— nettoyage de données

Exercice 1 : Premier contact avec la librairie `pandas`

En préambule, nous rappelons que l'aide sur une fonction/méthode donnée peut s'obtenir rapidement au sein d'une cellule de code. Par exemple, il suffit de saisir `pandas.read_csv?` pour obtenir directement l'aide sur la fonction `read_csv(...)`.

Q1. Intéressons-nous d'abord à la question de l'indexation et de l'accès aux données par défaut.

Q1.1. Importez la colonne '`age`' du fichier `data_mm02_clients_series.csv` grâce à l'option `usecols` de la méthode `read_csv(...)`.

Q1.2. Affichez les 5 premières lignes, via la méthode `head()`.

Q1.3. Affichez les 5 dernières lignes, via la méthode `tail()`.

Q1.4. Affichez la taille des données, via l'attribut `shape`.

Q1.5. Affichez les données d'indice 2 et d'étiquette 2 via les méthodes `iloc` et `loc`. Quelle différence cela fait-il ? Pourquoi ?

Q2. Indexons différemment les données.

Q2.1. Importez les colonnes `age` et `nom` du fichier `data_mm02_clients_series.csv` en utilisant cette fois le nom comme index, grâce à l'option `index_col` de la fonction `read_csv(...)`.

Q2.2. Affichez Les 5 premières lignes et constatez la différence avec l'indexation par défaut.

Q2.3. Affichez également la taille des données. Qu'est-ce qui change ?

Q2.4. Affichez l'âge d'indice 2 et l'âge de '*Beaufort Lesage*' via les méthodes `iloc` et `loc`.

Q3. Pratiquons maintenant quelques restrictions afin de filtrer notre *dataframe* !

Q3.1. Affichez les informations des personnes ayant exactement 41 ans (en combinant l'indexation booléenne et la méthode `.loc(...)`)

Q3.2. Affichez uniquement le nom (autrement dit l'index) de la dernière personne sur cette liste.

Q3.3. Affichez uniquement les informations des personnes ayant entre 40 et 50 ans.

Q3.4. Exportez la liste de toutes les quadragénaires dans un fichier CSV nommé '`result_mm02_quadra.csv`' dans lequel le séparateur est '`|`' (option `sep`) en utilisant la méthode `to_csv(...)`.

Q4. Voyons maintenant comment associer plusieurs fichiers.

Q4.1. Importez dans un *dataframe* `clients1` les données du fichier CSV `data_mm02_clients_part1.csv`.

Q4.2. Malheureusement, les données initiales ont été morcelées en deux fichiers, sous des formats différents en plus ! Importez dans un autre *dataframe* `clients2` les données du fichier JSON `data_mm02_clients_part2.json`, via la fonction `read_json(...)`.

Q4.3. Fusionnez les deux *dataframes* afin de n'en obtenir qu'un seul, contenant toutes les données. Pour les mettre bout à bout, la fonction `concat(...)` est nécessaire.

Q4.4. Récupérez et importez le fichier complémentaire contenant les identifiants et les ages de certains clients : le fichier `data_mm02_clients_age.csv`.

Q4.5. Par jointure entre vos deux *dataframes*, reconstituez un ensemble de données complets (via la fonction `merge(...)` et les options `on` et `how`), sans perdre de clients au passage : on souhaite conserver les clients pour lesquels nous n'avons pas l'âge mais oublier les âges pour lesquels aucun client ne correspond. Vérifiez le nombre de lignes obtenues.

Q5. Encore quelques restrictions.

Q5.1. Affichez les informations liées aux clients dont l'age reste inconnu, via la méthode `isnull()`.

Q5.2. Affichez tous les clients ayant un numéro supérieur à 222.

Q5.3. Dès qu'on souhaite analyser ou effectuer un traitement sur une colonne de type '*chaîne de caractère*', il faut appliquer la méthode `.str.<la-méthode>()`. De cette manière, affichez tous les clients ayant un nom commençant par la lettre 'B' via la méthode `startswith(...)`.

Q5.4. Ajoutez une colonne `longueur_nom` à votre *dataframe* contenant le nombre de caractères du champ '*nom*', via la méthode `len()`.

Q5.5. Le fait de trouver à la fois le nom et le prénom à l'intérieur du même champ ne nous arrange pas. Faites les modifications nécessaires afin d'obtenir une colonne `nom` et une colonne `prénom` dédiées. La méthode `split(...)` avec son option booléenne `expand` mérite votre attention.

Q6. Un peu de visualisation.

Q6.1. Tracez un diagramme circulaire (`pie(...)`) représentant le nombre de clients par tranche d'âge. Pour ce faire :

1. créez les catégories dans une variable `bins`, via la fonction `arange(...)`
2. associez chaque age à sa catégorie grâce à la fonction `cut(...)`
3. comptez le nombre de données dans chaque catégorie via la méthode `value_counts()`
4. filtrez pour éliminer les catégories non nécessaires
5. tracez le diagramme circulaire via la fonction `pie(...)`

Q6.2. Cette représentation n'apporte pas suffisamment d'information. Réalisez un histogramme avec un découpage des tranches d'âge deux fois plus fin (fonction `hist(...)` et l'option `bins`).

Exercice 2 : Un premier nettoyage de données

Q1. Récupérez le fichier '`data_mm02_person.csv` et importez-le dans un *dataframe.*'

Q2. Regardons de plus près les valeurs nulles.

Q2.1. Déterminez le nombre de valeurs nulles par catégorie, en combinant les méthodes `isnll()` et `sum()`. Veillez à n'afficher que les catégories pour lesquelles il existe des valeurs nulles.

Q2.2. Affichez toutes les lignes contenant au moins une valeur nulle, grâce à la méthode `any(...)` et son option `axis`.

Q3. Les erreurs lexicales : commençons par regarder les pays.

Q3.1. Établissez une liste sans doublon des pays représentés, via la méthode `unique()`.

Q3.2. Une valeur n'est clairement pas un nom de pays. Affectez la valeur `np.NaN` à toutes les lignes dont le champ '`pays`' n'est pas correct. La méthode `isin(...)` pourra vous servir.

Q4. Intéressons-nous aux tailles et aux erreurs d'irrégularité.

Q4.1. Affichez les données qui ne sont pas exprimées en mètre (par indexation booléenne, grâce à méthode `contains(...)` en cherchant les unités).

Q4.2. Corrigez manuellement cette erreur (puisqu'elle est peut répandue) pour obtenir une colonne homogène et exploitable.

Q4.3. Créez une nouvelle colonne '`taille_cm`' entière contenant les tailles en centimètres (et uniquement les valeurs numériques, pas d'unité, via la méthode `replace(...)`). Notez que la méthode `astype(...)` est utile pour typer correctement une nouvelle colonne.

Q4.4. Supprimez la colonne `taille` dont le format ne permet pas son exploitation, grâce à la méthode `drop(...)`.

Q4.5. Pour plus d'aisance, renommez la colonne '`taille_cm`' en `taille` via la méthode `rename(...)`.

Q4.6. On peut traiter les colonnes de manière plus automatique. Voici les poids correspondants aux personnes. Ajoutez une colonne '`poids`' contenant ces données à votre *dataframe*.

```
poids = ["65000g", "88400g", "68000g", "75200g", "00Kg", "70200g", "70200g"]
```

Q4.7. Supprimez le dernier caractère dans cette colonne (pour normalement obtenir uniquement des chiffres). On fermera volontairement les yeux sur les éventuelles erreurs dans cette liste. Convertissez ensuite la colonne `poids` en valeurs numériques grâce à la méthode `pd.to_numeric(...)` tout en forçant les erreurs de conversion à des valeurs manquantes (`np.NaN`), via l'option `errors`.

Q4.8. Remplacez les éventuelles valeurs manquantes de la colonne '`poids`' par la moyenne des valeurs existantes.

Q5. Intéressons-nous maintenant aux courriels et aux erreurs de formattage.

Q5.1. Certaines lignes contiennent plusieurs valeurs au lieu d'une. Générez une série déterminant le nombre de caractère '`@`' dans chaque valeur de la colonne '`email`', via la méthode `count()`. Affichez les données où plusieurs courriels sont fournis.

Q5.2. Conservez uniquement la dernière valeur. Veillez à ne pas faire un code *ad-hoc* et ne pas corriger manuellement la ligne.

Q5.3. On peut également constater que toutes les dates ne respectent pas le même format. Il convient de régler la question. Il faut procéder en deux fois :

— créez une nouvelle colonne `naiss` en faisant une conversion selon un premier format (`%d/%m/%Y` par exemple) grâce à la méthode `pd.to_datetime(...)`, en spécifiant ce format et en écartant les échecs (comme pour les poids) par l'option `errors` ;

— refaites une conversion selon un autre format (`%d %b. %Y` en l'occurrence) pour les valeurs nulles de cette nouvelle colonne.

Q5.4. Terminez proprement en remplaçant l'ancienne colonne par la nouvelle, contenant les mêmes dates mais qui respectent un même format et sont donc exploitables.

Q6. Intéressons-nous enfin au cas des doublons.

Q6.1. Affichez les lignes en doublons, d'après leur courriel, grâce à la méthode `duplicated(...)`.

Q6.2. Vous pouvez constater que chacune des lignes apportent son lot d'information. Il faut consolider tout ça au cas par cas en complétant la première occurence avec les informations supplémentaires fournies par les autres occurences. En cas de valeurs incohérentes, on affectera la moyenne des valeurs existantes.

Q6.3. Supprimez les doublons en conservant la première occurence, qui contient désormais les données aggrégées, via la méthode `drop_duplicates(...)`.