|  |
| --- |
|  |
| Cheatsheet data science |

1. Python : https://www.python.org/
   1. Structures :

Commandes :

|  |
| --- |
| Comprehension\_list = [item.upper() for item in items] |

* 1. Profilage de performance :

Mesurer le temps :

|  |
| --- |
| import time |
| start = time.time() |
| end = time.time() – start |
| print(f "{round(end, 2)}") |

Mesurer l’utilisation mémoire :

|  |
| --- |
| import memory\_profiler |
| @profile |
| def function() :  [...] |

Autres outils :

|  |
| --- |
| https://toucantoco.com/en/tech-blog/tech/python-performance-optimization |

* 1. Regex  :

Commandes :

|  |
| --- |
| re\_target = re.compile(r"Targets,\"(.\*)\"") |
| re\_ip\_mask = re.compile(r"(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})(/[25]{3}\.[25]{1,3}\.\d{1,3}\.\d{1,3})?") |
| re\_ip = re.compile(r"^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$") |

1. Pandas : https://pandas.pydata.org/
   1. Lecture de données :

Commandes :

|  |
| --- |
| pd.read\_csv(io, sheet\_name=0, header=0, names=None, index\_col=None, usecols=None, squeeze=False, dtype=None, engine=None, converters=None, true\_values=None, false\_values=None, skiprows=None, nrows=None, na\_values=None, keep\_default\_na=True, verbose=False, parse\_dates=False, date\_parser=None, thousands=None, comment=None, skip\_footer=0, skipfooter=0, convert\_float=True, mangle\_dupe\_cols=True, \*\*kwds) |
| pd.read\_excel(io, sheet\_name=0, header=0, names=None, index\_col=None, usecols=None, squeeze=False, dtype=None, engine=None, converters=None, true\_values=None, false\_values=None, skiprows=None, nrows=None, na\_values=None, keep\_default\_na=True, verbose=False, parse\_dates=False, date\_parser=None, thousands=None, comment=None, skip\_footer=0, skipfooter=0, convert\_float=True, mangle\_dupe\_cols=True, \*\*kwds) |
| pd.read\_html(io, match='.+', flavor=None, header=None, index\_col=None, skiprows=None, attrs=None, parse\_dates=False, tupleize\_cols=None, thousands=', ', encoding=None, decimal='.', converters=None, na\_values=None, keep\_default\_na=True, displayed\_only=True) |

Arguments :

|  |
| --- |
| file\_path=path object or str 🡺 path vers le fichier à lire |
| sheet\_name=str 🡺 le nom de la feuille à lire dans le fichier |
| names=array like 🡺 le(s) nom(s) des colonnes à extraire |
| header=int 🡺 le numéro de la ligne contenant le nom des colonnes |
| index\_col=str 🡺 le nom de la colonne à utiliser comme index |
| skiprows=int 🡺 le nombre de ligne (en partant du haut) dans le fichier à ignorer |
| nrows=int 🡺 le nombre de row à extraire |
| dtype=type name or dict 🡺 le type des données ex. : {‘nom\_col’ : type} |
| convert\_float=bool 🡺 transforme les float en integer |
| date\_parser=function 🡺 fonction qui permet de formater des dates sous la forme de str |
| na\_values=str, list or dict 🡺 motifs et valeurs qui seront interprétés comme des NaN |
| keep\_default\_na=bool 🡺 si False, écrase les NaN |
| match=regex (pd.read\_html only) 🡺 extrait les tableaux qui match le regex |

Pour extraire le contenu d’un fichier html dans un dataframe pandas :

|  |
| --- |
| import pandas as pd |
| from lxml import etree, html |
| try: |
| with open(html\_path) as file\_data: |
| print("Firewall rules and groups found, treating...") |
| lines = file\_data.readlines() |
| content = "".join(lines) |
| except FileNotFoundError: |
| print("Firewall rules and groups not found") |
| else: |
| # Tagging groups |
| doc = html.fromstring(content) |
| for element in doc.xpath('\*//tr[@class="odd"]/td/a'): |
| element.text = f"TAG\_{element.text}" |
| dataframes = pd.read\_html(etree.tostring(doc), match=RE\_TABLES, header=0,) |

* 1. Type des données et NaN dans un dataframe :

Commande :

|  |
| --- |
| df.astype(self, dtype, copy=True, errors='raise', \*\*kwargs) |
| df.fillna(self, value=None, method=None, axis=None, inplace=False, limit=None, downcast=None, \*\*kwargs) |

* 1. Manipulation de données :

Commande :

|  |
| --- |
| pd.merge(self, right, how='inner', on=None, left\_on=None, right\_on=None, left\_index=False, right\_index=False, sort=False, suffixes=('\_x', '\_y'), copy=True, indicator=False, validate=None) |

Arguments :

|  |
| --- |
| self=pd.DataFrame 🡺 premier dataframe, celui de gauche |
| right=pd.DataFrame 🡺 second dataframe, celui de droite |
| how={‘left’, ‘right’, ‘outer’, ‘inner’} 🡺 type de merge |
| on=label or list 🡺 nom de la ou des colonnes utilisées pour le merge |
| left\_on=label or list 🡺 la ou les colonnes dans le tableau de gauche utilisées pour le merge |
| right\_on=label or list 🡺 la ou les colonnes dans le tableau de droite utilisées pour le merge |
| indicator=bool or str 🡺 si True, ajoute une colonne ‘\_merge’ ou ‘<str>’ contenant les information ‘left\_only’, ‘right\_only’ ou ‘both’ si la valeur est contenue dans le dataframe de gauche, de droite ou les deux respectivement |
| copy=bool 🡺 ignore la copy si c’est possible (car les dataframe sont des objets mutables) |

Exemple :

|  |
| --- |
| >>> df1 |
| lkey value |
| 0 foo 1 |
| 1 bar 2 |
| 2 baz 3 |
| 3 foo 5 |
| >>> df2 |
| rkey value |
| 0 foo 5 |
| 1 bar 6 |
| 2 baz 7 |
| 3 foo 8 |
| >>> df1.merge(df2, left\_on='lkey', right\_on='rkey', |
| ... suffixes=('\_left', '\_right')) |
| lkey value\_left rkey value\_right |
| 0 foo 1 foo 5 |
| 1 foo 1 foo 8 |
| 2 foo 5 foo 5 |
| 3 foo 5 foo 8 |
| 4 bar 2 bar 6 |
| 5 baz 3 baz 7 |

Commande :

|  |
| --- |
| pd.concat(objs, axis=0, join='outer', join\_axes=None, ignore\_index=False, keys=None, levels=None, names=None, verify\_integrity=False, sort=None, copy=True) |

Arguments :

|  |
| --- |
| objs=list of series or list of dict 🡺 concatene ces objets |
| join={‘inner’, ‘outer’} default ‘outer’ 🡺 pour gérer les indexes |
| ignore\_index=bool 🡺 si Vrai, reset les index pour créer les nouveaux : 0, 1, …, n |
| sort=bool 🡺 classe les entrées en fonction des index |

Exemple :

|  |
| --- |
| >>> s1 = pd.Series(['a', 'b']) |
| >>> s2 = pd.Series(['c', 'd']) |
| >>> pd.concat([s1, s2]) |
| 0 a |
| 1 b |
| 0 c |
| 1 d |

Commande :

|  |
| --- |
| df.str.replace(self, pat, repl, n=1, case=None, flags=0, regex=True) |

Arguments :

|  |
| --- |
| to\_replace=str, regex, list, dict, int, float or pd.Series 🡺 quelle(s) valeur(s) sont à replacer ? le pattern doit être exact |
| value=scalar, dict, list, str or regex 🡺 remplace par les valeurs indiquées |
| inplace=bool 🡺 si Vrai, effectue le remplacement directement dans le dataframe |
| regex=bool ou même type que pour to\_replace 🡺 pareil que to\_replace mais en regex, permet donc de replacer une partie des str plutôt que tout |

Exemples :

|  |
| --- |
| >>> s = pd.Series([0, 1, 2, 3, 4]) |
| >>> s.replace(0, 5) |
| 0 5 |
| 1 1 |
| 2 2 |
| 3 3 |
| 4 4 |
| dtype: int64 |
| >>> df = pd.DataFrame({'A': ['bat', 'foo', 'bait'], |
| ... 'B': ['abc', 'bar', 'xyz']}) |
| >>> df.replace(to\_replace=r'^ba.$', value='new', regex=True) |
| A B |
| 0 new abc |
| 1 foo new |
| 2 bait xyz |

Commande :

|  |
| --- |
| df.grouby(self, by=None, axis=0, level=None, as\_index=True, sort=True, group\_keys=True, squeeze=False, observed=False, \*\*kwargs) |

Arguments :

|  |
| --- |
| by=mapping, function, label or list 🡺 la ou les colonnes utilisées pour faire le groupby() |
| axis={0, 1} 🡺 sur les index (lignes) : 0, ou sur les colonnes : 1 |
| level=int, level name or sequence 🡺 à utiliser si le dataframe contient des MultiIndex |
| sort=bool 🡺 tri par les clés données en ‘by’ |

Exemples :

|  |
| --- |
| >>> df |
| Animal Max Speed |
| 0 Falcon 380.0 |
| 1 Falcon 370.0 |
| 2 Parrot 24.0 |
| 3 Parrot 26.0 |
| >>> df.groupby(['Animal']).mean() |
| Max Speed |
| Animal |
| Falcon 375.0 |
| Parrot 25.0 |

Autres commandes :

|  |
| --- |
| df.loc[(df[‘nom\_col1’] == condition1) & ~(df[‘nom\_col2’] == condition2)] |
| df.loc[df[‘nom\_col1’].str.startswith(condition1)] |
| if not df.empty :  [...] |
| df.drop(self, labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise') |
| df.drop\_duplicates(self, subset=None, keep='first', inplace=False) |
| for index, row in df.iterrows(self) :  [...] |
| df.itertuples(self, index=True, name='Pandas') |
| df.pivot\_table(data, values=None, index=None, columns=None, aggfunc='mean', fill\_value=None, margins=False, dropna=True, margins\_name='All', observed=False) |
| df.apply(self, func, axis=0, broadcast=None, raw=False, reduce=None, result\_type=None, args=(), \*\*kwds) |
| df.unique(values) |
| df.to\_list(self) |

* 1. Optimisation de code  :

Liens :

Reduce Pandas memory usage by loading less data. On a déjà vu comment ne charger que les colonnes nécessaires, mais cet article va plus loin : https://pythonspeed.com/articles/pandas-load-less-data/

When your data doesn’t fit in memory: the basic techniques. J'ai pas encore lu, mais ça peut être intéressant : https://pythonspeed.com/articles/data-doesnt-fit-in-memory/

Pandas GroupBy: Your Guide to Grouping Data in Python. Récemment, on a aussi vu comment groupby pouvait accélérer les traitements, mais si l'utilisation est encore assez flou, cet article devrait lever tous les doutes (et bonus, ils utilisent usecols et dtypes comme expliqué dans le premier article du jour) : https://realpython.com/pandas-groupby/

1. Git : https://github.com/

Commandes :

|  |
| --- |
| git clone <link> 🡺 récupère un répertoire de git et en créer un clone dans le répertoire courant |
| git status 🡺 affiche le statut du repo courant |
| git log 🡺 affiche les logs des commits |
| git fetch 🡺 récupère les branches sur le repo courant |
| git branch -a 🡺 liste les branches du repo courant |
| git checkout <nom\_branche> 🡺 se déplace sur la branche <nom\_branche> |
| git add 🡺 ajoute un file au repo, il sera ainsi versionné |
| git commit -m <message> 🡺 créer un nouveau commit avec le commentaire <message> |
| git push 🡺 push les commits en attente |
| git pull 🡺 met le repo courant à jour avec le contenu présent sur git |
| git reset –hard 🡺 annule le dernier commit |

1. Autres cheat sheets  :

Liens :

TLDR, un man simplifié pour les utilitaires les plus courants : https://tldr.ostera.io/

Dev Hints, pour des cheat sheets bien pratiques sur tout un tas d'applications, d'utilitaires, etc : https://devhints.io/

Cours :

|  |
| --- |
| Datacamp : https://www.datacamp.com/ |