

# **Manipulation de données avec Python pour des travaux d'investigation**

Partie 1 - Lecture et preparation des données

NOE BOUDON - CECILE LIN

Année 2022 - 2023

# Contents

<b>1 Lire et observer des données avec Python</b>	<b>3</b>
1.0.1 Création du DataFrame applications . . . . .	3
1.0.2 Création du DataFrame personnes . . . . .	4
1.0.3 Création du DataFrame projets . . . . .	4
1.0.4 Création du DataFrame logs . . . . .	5
1.0.5 Création du DataFrame taches . . . . .	6
<b>2 Vérifications et préparation des données</b>	<b>6</b>
2.1 Base des projets . . . . .	7
2.2 Base des applications . . . . .	18
2.3 Base des personnes . . . . .	20
2.4 Base des logs . . . . .	24
2.5 Base des tâches . . . . .	25
<b>3 Sauvegarde des jeux de données</b>	<b>26</b>
3.1 Conclusion . . . . .	26

## Partie 1 - Lecture et preparation des donnees

Les données obtenues dans le cadre professionnel peuvent avoir **différents formats**, dont certains sont en fait du texte (.csv, .txt, etc.) et ne sont pas toujours utilisables comme telles – **il faut parfois enrichir ces données** en ajoutant des colonnes qui découlent des colonnes initiales, notamment (i) pour nettoyer les données (e.g. identification des doublons, suppression de dates erronées), (ii) pour faire ressortir des indicateurs statistiques plus pertinents par combinaison d'indicateurs unitaires (e.g. calculs de ratios, de maximum), (iii) pour homogénéiser des données qui sont hétérogènes fautes de standards dans l'entreprise.

Le travail d'enrichissement peut également permettre de **fiabiliser** une base de données par calcul d'agrégats, analyse de séquences logiques (si je dépasse ce seuil, alors cette colonne doit être égale à, etc.).

Tout ceci requiert une bonne maîtrise des fonctions de Python que nous allons explorer dans cette partie.

```
[617]: # imports
import pandas as pd
import datetime
import re
import matplotlib.pyplot as plt
import numpy as np
import os

# Permet d'afficher jusqu'à 50 colonnes lorsqu'on observe un DataFrame
pd.options.display.max_columns = 50

# Permet de définir une taille par défaut de 15x5 pour les graphiques matplotlib
plt.rcParams["figure.figsize"] = (15,5)
```

### 1 Lire et observer des données avec Python

Les données reçues sont réparties dans 3 fichiers : - *DIT - Application, Personne, Projet.xlsx* - Ce fichier contient 3 jeux de données différents *Application, Personne* et *Projet - log.txt - tache.csv*

Pour commencer, utilisez pandas pour lire chacun des 3 fichiers. Attention ces fichiers ont des extensions différentes (.xlsx, .csv et .txt). Il peut également y avoir des difficultés de lecture liées à l'encodage ou aux séparateurs. N'hésitez pas à observer les fichiers !

#### 1.0.1 Création du DataFrame applications

Créez un DataFrame applications composé des données du fichier *DIT - Application, Personne, Projet.xlsx* issues de la feuille de calcul *Application*. Et enregistrez celui-ci dans une variable nommée applications.

Affichez ensuite les 2 premières lignes du DataFrame

```
[2]: # Votre code ci-dessous 'Examens blancs/'

path1 = '_Data/DIT - Application, Personne, Projet.xlsx'
applications = pd.read_excel(path1, 'Application')
applications.head(2)
```

```
[2]:
```

	AppCode	Name	RiskLevel	LastResilienceTest	NombreUtilisateur2017	\
0	A6205	Yearin	HIGH	42795	9704	
1	A7527	Goodsilron	LOW	41477	6969	

	NombreUtilisateur2016
0	9740
1	6938

### 1.0.2 Création du DataFrame personnes

Créez un DataFrame personnes composé des données du fichier *DIT - Application, Personne, Projet.xlsx* issues de la feuille de calcul *Personne*. Et enregistrez celui-ci dans une variable nommée *personnes*.

Affichez ensuite les 2 premières lignes du DataFrame

```
[568]: # Votre code ci-dessous

path2 = '_Data/Personnes.csv'
personnes = pd.read_csv(path2)
personnes.head(2)
```

```
[568]:
```

	Prenom	Nom	Mail	ID	\
0	Charles	Smith	charles.smith@tgp.com	A692817085	
1	Madeleine	Chapman	madeleine.chapman@tgp.com	A185252110	

	Role	Experience	Valeur
0	Developpeur	Junior	1
1	Developpeur	Junior	1

### 1.0.3 Création du DataFrame projets

Créez un DataFrame projets composé des données du fichier *DIT - Application, Personne, Projet.xlsx* issues de la feuille de calcul *Projet*. Et enregistrez celui-ci dans une variable nommée *projets*.

Puis retirez les 7 dernières colonnes de ce DataFrame et affichez les 3 premières lignes de celui-ci.

```
[280]: # Votre code ci-dessous

path3 = '_Data/DIT - Application, Personne, Projet.xlsx'
projets = pd.read_excel(path3, 'Projet')
```

```
projets = projets.drop(['ExperienceChefDeProjet', 'Appli2_Nom', 'Appli1_Nom', 'Appli2', 'Appli1', 'DevisBudgetEUR_Clean', 'RankBudget'], axis=1)
projets.head(3)
```

```
[280]:
```

	Annee	NumeroVersion	FicheDeDemande	ReferenceID	NomDuProjet
0	2014	1	x	218486	Projet Girl Scout
1	2015	1	+	230240	Projet Cobra
2	2015	4	x	240101	Projet Bee - LOT 4

	DateCreation	Statut	InterlocuteurMetier	ChefDeProjet
0	41741	RAS	Joseph DICKENS	Jane SHARP
1	42128	Termine	Madeleine BROWN	Gavin KING
2	42090	En attente de decision	Leonard JONES	Ian HEMMINGS

	DateDemande	DateReponse	TypeDemande	TypeDevis	CauseReestimation
0	41640	41883	Catalogue	Reestime	Autre
1	41640	42896	Projet	Initial	NaN
2	41779	42968	Evolution	Reestime	Autre

	CodeApplicAtion	DateDebutTravaux	DateFinTravaux	DateDebutHomologation
0	A4701; A7843	41929	42176	42244
1	A3758; A3774	42917	43083	43128
2	A4452; A5146	43021	43167	43195

	DevisBudgetJH	DevisBudgetEUR	DevisBudgetJHEUR	DevisBudgetInfraEUR
0	1900	2,6 MEUR	1235000	1365000.0
1	2000	2,7 MEUR	1300000	1400000.0
2	1400	2 MEUR	910000	1090000.0

#### 1.0.4 Création du DataFrame logs

Nous allons créer un DataFrame logs à partir des données issues du fichier *log.txt*. Mais nous ne savons pas quel séparateur est utilisé...

Dans un premier temps, utilisez la fonction `open()` pour lire les données qui y sont contenues et affichez les 100 premiers caractères. Il est possible qu'il y ait également des problèmes liés à l'encoding...

```
[618]: # Votre code ci-dessous

#logs = pd.read_csv('./_Data/log.txt', sep=" ", header=None)
#logs.columns = ['ProjectKey', 'TaskAssignee', 'TaskKey', 'DateFinalLog', 'MDLogged']

with open('./_Data/log.txt', 'r', encoding = 'utf-16') as f:
    logs = f.read()

print(logs[0:100])
```

ProjectKey	TaskAssignee	TaskKey	DateFinalLog	MDLogged
218486	A810043709	M1	29/01/2015	1
218486	A8100			

Maintenant que vous avez identifié le séparateur, utilisez la méthode appropriée de pandas pour créer un DataFrame avec ces données. Puis, affichez les 2 premières lignes de ce DataFrame.

```
[622]: # Votre code ci-dessous

logs = pd.read_csv('./_Data/log.txt', encoding = 'utf-16', sep= "\t" , decimal = "\u2192"."")
logs.head(2)
```

```
[622]:
```

	ProjectKey	TaskAssignee	TaskKey	DateFinalLog	MDLogged
0	218486	A810043709	M1	29/01/2015	1
1	218486	A810043709	M1	11/01/2015	1

### 1.0.5 Création du DataFrame taches

Créez un DataFrame taches composé des données du fichier *tache.csv* et enregistrez celui-ci dans une variable nommée taches.

Puis affichez les 3 premières lignes de celui-ci.

```
[620]: # Votre code ci-dessous

path4 = './_Data/tache.csv'
taches = pd.read_csv(path4, encoding = 'latin-1', sep=';', decimal='.')
taches.head(3)
```

```
[620]:
```

	ProjectKey	TaskAssignee	Priority	TaskKey	MDPlanned	MDUpdatedPlanned	\
0	218486	A810043709	VERY LOW	M1	5	4	
1	218486	A722019848	LOW	M2	8	12	
2	218486	A69888201	MEDIUM	M3	9	7	

	TaskType	Location
0	CODING	Bangalore
1	STRUCTURE CHANGE	Bangalore
2	BUG	Paris

## 2 Vérifications et préparation des données

Quelques informations sur les différents jeux de données disponibles : - applications contient des données relatives aux différentes applications - i.e. les identifiants, noms, niveau de risque (criticité), date du dernier test de résilience et nombre d'utilisateurs sur deux années successives (2017 et 2018). - personnes contient des données relatives aux collaborateurs de la société TGP (une ESN - Entreprise de services du numérique). On y retrouve les noms, prénoms, adresses mail, identifiants de chacun des collaborateurs ainsi que leur poste et leur expérience (en années).

- projets contient des données concernant des informations concernant les différents projets réalisées par TGP. Entre autres, (i) le nom du chef de projet, (ii) les dates des différentes étapes, (iii) les applications concernées par chaque projet... - logs contient les informations relatives au *time tracking* de ces projets - i.e. le nombre de jours (*Jour Homme* JH, également appelé *Man Day* MD) passé par chaque collaborateur sur chaque tâche du projet. - taches contient des informations sur les différentes tâches des projets. En particulier le nombre de jours (MD) planifiés sur chacune des tâches, leur niveau de priorité, l'endroit (ville) où elles ont été réalisées.

Maintenant que les données ont été chargées dans des DataFrames, nous allons en vérifier la qualité. Les objectifs de cette partie sont (i) découvrir les données disponibles, (ii) d'identifier les différents problèmes dans les données pouvant bloquer nos analyses et (iii) résoudre ces problèmes.

## 2.1 Base des projets

```
[105]: projets.head()
```

```
[105]:
```

	Annee	NumeroVersion	FicheDeDemande	ReferenceID	NomDuProjet	\
0	2014	1	x	218486	Projet Girl Scout	
1	2015	1	+	230240	Projet Cobra	
2	2015	4	x	240101	Projet Bee - LOT 4	
3	2016	3	x	294007	Projet Templer	
4	2017	1	+	304283	Projet Alpha	

	DateCreation	Statut	InterlocuteurMetier	ChefDeProjet	\
0	41741	RAS	Joseph DICKENS	Jane SHARP	
1	42128	Termine	Madeleine BROWN	Gavin KING	
2	42090	En attente de decision	Leonard JONES	Ian HEMMINGS	
3	42518	GO	Madeleine BROWN	David JACKSON	
4	42746	GO	Madeleine BROWN	David JACKSON	

	DateDemande	DateReponse	TypeDemande	TypeDevis	CauseReestimation	\
0	41640	41883	Catalogue	Reestime	Autre	
1	41640	42896	Projet	Initial	NaN	
2	41779	42968	Evolution	Reestime	Autre	
3	41814	42607	Projet	Reestime	Autre	
4	41848	42790	Evolution	Reestime	Ajout contributeur	

	CodeApplicAtion	DateDebutTravaux	DateFinTravaux	DateDebutHomologation	\
0	A4701; A7843	41929	42176	42244	
1	A3758; A3774	42917	43083	43128	
2	A4452; A5146	43021	43167	43195	
3	A4317; A8067	42675	42957	43009	
4	A3774; A5876	42827	42976	43058	

	DevisBudgetJH	DevisBudgetEUR	DevisBudgetJHEUR	DevisBudgetInfraEUR
0	1900	2,6 MEUR	1235000	1365000.0
1	2000	2,7 MEUR	1300000	1400000.0

2	1400	2 MEUR	910000	1090000.0
3	2200	3 MEUR	1430000	1570000.0
4	1500	2,2 MEUR	975000	1225000.0

**Modification des colonnes devant contenir des dates** En observant rapidement le DataFrame, on se rend compte que les colonnes *DateCreation*, *DateDemande*, *DateReponse*, *DateFinTravaux* et *DateDebutHomologation* contiennent des nombres...

Or, vu le nom de ces colonnes, elles devraient contenir des dates ! Cela est dû au fait qu'Excel enregistre les dates comme étant le nombre de jours depuis le 30 décembre 1899.

Pour corriger ce problème, écrivez un script permettant de transformer les valeurs de ces colonnes en dates (i.e. 1899-12-30 + n jours, avec n - valeurs présentes actuellement dans ces colonnes).

```
[312]: #nb_annee_en_plus_C = projets['DateCreation']//365
#annee_C = nb_annee_en_plus_C + 1 + 1899 # en prenant en compte qu'on démarre du
↳30/12 et non pas du 31/12

#nb_annee_en_plus_D = projets['DateDemande']//365
#annee_D = nb_annee_en_plus_D + 1 + 1899

#nb_annee_en_plus_R = projets['DateReponse']//365
#annee_R = nb_annee_en_plus_R + 1 + 1899

#nb_annee_en_plus_T = projets['DateFinTravaux']//365
#annee_T = nb_annee_en_plus_T + 1 + 1899

#nb_annee_en_plus_H = projets['DateDebutHomologation']//365
#annee_H = nb_annee_en_plus_H + 1 + 1899
from datetime import *

import warnings
warnings.filterwarnings('ignore')
warnings.warn('DelftStack')
warnings.warn('Do not show this message')

j = 0
for i in projets['DateCreation']:
    projets['DateCreation'][j] = datetime.fromordinal(datetime(1900, 1, 1).
↳toordinal() + i - 2)
    #print(projets['DateCreation'][j])
    j = j + 1

j = 0
for i in projets['DateDemande']:
    projets['DateDemande'][j] = datetime.fromordinal(datetime(1900, 1, 1).
↳toordinal() + i - 2)
    j = j + 1
```

```

j = 0
for i in projets['DateReponse']:
    projets['DateReponse'][j] = datetime.fromordinal(datetime(1900, 1, 1).
→toordinal() + i - 2)
    j = j + 1

j = 0
for i in projets['DateFinTravaux']:
    projets['DateFinTravaux'][j] = datetime.fromordinal(datetime(1900, 1, 1).
→toordinal() + i - 2)
    j = j + 1

j = 0
for i in projets['DateDebutHomologation']:
    projets['DateDebutHomologation'][j] = datetime.fromordinal(datetime(1900, 1, 1)
→1).toordinal() + i - 2)
    j = j + 1

projets.head()

```

```

[312]:
Annee  NumeroVersion  FicheDeDemande  ReferenceID  NomDuProjet  \
0  2014              1             x      218486  Projet Girl Scout
1  2015              1             +      230240           Projet Cobra
2  2015              4             x      240101  Projet Bee - LOT 4
3  2016              3             x      294007           Projet Templer
4  2017              1             +      304283           Projet Alpha

      DateCreation      Statut  InterlocuteurMetier  \
0  2014-04-12 00:00:00      RAS      Joseph DICKENS
1  2015-05-04 00:00:00      Termine      Madeleine BROWN
2  2015-03-27 00:00:00  En attente de decision      Leonard JONES
3  2016-05-28 00:00:00      GO      Madeleine BROWN
4  2017-01-11 00:00:00      GO      Madeleine BROWN

      ChefDeProjet      DateDemande      DateReponse  TypeDemande  \
0      Jane SHARP  2014-01-01 00:00:00  2014-09-01 00:00:00  Catalogue
1      Gavin KING  2014-01-01 00:00:00  2017-06-10 00:00:00      Projet
2      Ian HEMMINGS  2014-05-20 00:00:00  2017-08-21 00:00:00  Evolution
3      David JACKSON  2014-06-24 00:00:00  2016-08-25 00:00:00      Projet
4      David JACKSON  2014-07-28 00:00:00  2017-02-24 00:00:00  Evolution

      TypeDevis  CauseReestimation  CodeApplicAtion  DateDebutTravaux  \
0  Reestime      Autre      A4701; A7843      41929
1  Initial      NaN      A3758; A3774      42917
2  Reestime      Autre      A4452; A5146      43021
3  Reestime      Autre      A4317; A8067      42675

```

4 Reestime Ajout contributeur A3774; A5876 42827

	DateFinTravaux	DateDebutHomologation	DevisBudgetJH	DevisBudgetEUR	\
0	2015-06-21 00:00:00	2015-08-28 00:00:00	1900	2,6	MEUR
1	2017-12-14 00:00:00	2018-01-28 00:00:00	2000	2,7	MEUR
2	2018-03-08 00:00:00	2018-04-05 00:00:00	1400	2	MEUR
3	2017-08-10 00:00:00	2017-10-01 00:00:00	2200	3	MEUR
4	2017-08-29 00:00:00	2017-11-19 00:00:00	1500	2,2	MEUR

	DevisBudgetJHEUR	DevisBudgetInfraEUR
0	1235000	1365000.0
1	1300000	1400000.0
2	910000	1090000.0
3	1430000	1570000.0
4	975000	1225000.0

```
[314]: # Vous pourriez également utiliser le package xlrd !
# Exemple pour DateCreation de la première ligne

import xlrd
xlrd.xldate_as_datetime(41741,0)
```

```
[314]: datetime.datetime(2014, 4, 12, 0, 0)
```

**Observation et correction des valeurs nulles** Créez un *barchart* permettant d'observer le nombre de valeurs nulles dans chacune des colonnes.

```
[315]: # Votre code ci-dessous

#import missingno as msno
#msno.bar(projets)

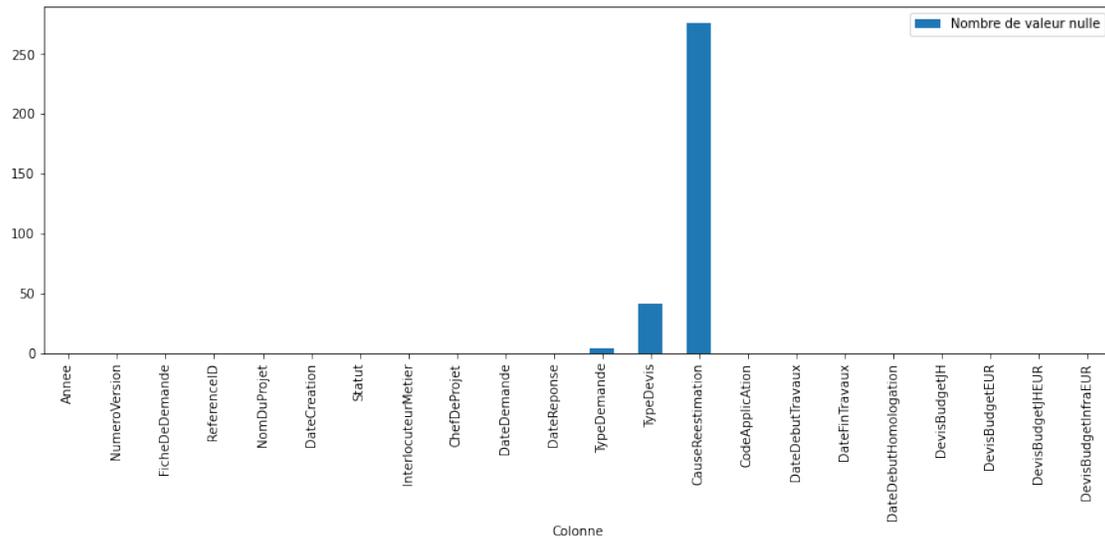
a = []

for i in projets:
    a = a + [projets[i].isna().sum()]

bc = pd.DataFrame(
    {"Nombre de valeur nulle":a},
    index = projets.columns
)

bc.plot(kind = 'bar', y = "Nombre de valeur nulle");
plt.xlabel("Colonne")
plt.legend()
plt.show()
```

```
#print(a)
```



```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 41, 276, 0, 0, 0, 0, 0, 0, 0, 0]
```

Puis remplacez les valeurs nulles des colonnes TypeDemande et TypeDevis par les chaînes de caractères "Inconnu" et "Sans devis" respectivement.

```
[311]: path3 = '_Data/DIT - Application, Personne, Projet.xlsx'
projets = pd.read_excel(path3, 'Projet')
projets = projets.drop(['ExperienceChefDeProjet', 'Appli2_Nom', 'Appli1_Nom',
↳ 'Appli2', 'Appli1', 'DevisBudgetEUR_Clean', 'RankBudget'], axis=1)
projets.head(3)
```

```
[311]:
```

	Annee	NumeroVersion	FicheDeDemande	ReferenceID	NomDuProjet
0	2014	1	x	218486	Projet Girl Scout
1	2015	1	+	230240	Projet Cobra
2	2015	4	x	240101	Projet Bee - LOT 4

	DateCreation	Statut	InterlocuteurMetier	ChefDeProjet
0	41741	RAS	Joseph DICKENS	Jane SHARP
1	42128	Termine	Madeleine BROWN	Gavin KING
2	42090	En attente de decision	Leonard JONES	Ian HEMMINGS

	DateDemande	DateReponse	TypeDemande	TypeDevis	CauseReestimation
0	41640	41883	Catalogue	Reestime	Autre
1	41640	42896	Projet	Initial	NaN
2	41779	42968	Evolution	Reestime	Autre

	CodeApplicAtion	DateDebutTravaux	DateFinTravaux	DateDebutHomologation
--	-----------------	------------------	----------------	-----------------------

0	A4701; A7843	41929	42176	42244
1	A3758; A3774	42917	43083	43128
2	A4452; A5146	43021	43167	43195

	DevisBudgetJH	DevisBudgetEUR	DevisBudgetJHEUR	DevisBudgetInfraEUR
0	1900	2,6 MEUR	1235000	1365000.0
1	2000	2,7 MEUR	1300000	1400000.0
2	1400	2 MEUR	910000	1090000.0

```
[318]: # Votre code ci-dessous

#projets['TypeDemande'] = projets['TypeDemande'].fillna('Inconnu')
#projets['TypeDevis'] = projets['TypeDevis'].fillna('Sans devis')

valeur_null = projets['TypeDevis'].isna()

j = 0
for i in range(len(valeur_null)):
    if valeur_null[i]:
        projets['TypeDevis'][i] = 'Sans devis'
        j +=1

#print(projets['TypeDevis'])
#print(j)

valeur_null = projets['TypeDemande'].isna()

j = 0
for i in range(len(valeur_null)):
    if valeur_null[i]:
        projets['TypeDemande'][i] = 'Inconnu'
        j +=1 #permet de verifier qu'on a bien tout remplacer

#print(projets['TypeDemande'])
#print(j)
```

0

Y a-t-il des lignes pour lesquelles le TypeDevis est "Reestime" mais la colonne CauseReestimation est nulle (NaN) ?

Ecrivez un script pour vérifier cela.

```
[330]: # Votre code ci-dessous

valeur_null_TypeDevis = (projets['TypeDevis'] == 'Reestime')
valeur_null_Cause = projets['CauseReestimation'].isna()

for i in range(len(valeur_null_TypeDevis)):
```

```

    if valeur_null_TypeDevis[i] and valeur_null_Cause[i] and i > 1:
        # la condition i > 1 pour s'assurer que plusieurs lignes ont les deux
        → conditions demandées
        # néanmoins la condition n'aurait pas été nécessaire si la question
        → avait été 'Y a-t-il au moins une ligne..'
        print(True)
        break

```

True

Ecrivez maintenant un script permettant remplacer les valeurs nulles de la colonne CauseReestimation mais cela uniquement lorsque TypeDevis est "Reestime"

```

[346]: # Votre code ci-dessous

l_null_reestime = []
for i in range(len(valeur_null_TypeDevis)):
    if valeur_null_TypeDevis[i] and valeur_null_Cause[i]:
        l_null_reestime = l_null_reestime + [i]

for i in l_null_reestime:
    #print(i)
    #print(projets['CauseReestimation'][i])
    x = input('par quoi on remplace la valeur nulle à la ligne? ' + str(i) + ' :')
    →)
    projets['CauseReestimation'][i] = x

```

**Modification de la colonne CodeApplication** La colonne code application peut contenir jusqu'à 2 codes d'applications (séparés par un ";" )...

Créez 2 nouvelles colonnes: - Appli1 - qui contient le premier code application (celui avant le ";" )  
;- Appli2 - qui contient le second code application, si il existe (et NaN sinon).

Attention : dans la colonne "CodeApplication", il pourrait y avoir des espaces avant ou après les codes... Cela pourrait causer des problèmes. Il faudra utiliser la méthode .strip() pour supprimer ces espaces inutiles.

```

[362]: # Votre code ci-dessous

projets['CodeApplication'].str.strip()

projets['Appli1'] = projets['CodeApplication'].copy()
projets['Appli2'] = projets['CodeApplication'].copy()

j = 0
for i in projets['CodeApplication']:
    #print(i)
    position = i.find(';')
    #print(position)

```

```

projets['Appli1'][j] = i[0:position]
projets['Appli2'][j] = i[position+1:len(i)]
j += 1

```

```
projets.head()
```

```

[362]:
Annee  NumeroVersion  FicheDeDemande  ReferenceID  NomDuProjet \
0  2014            1            x      218486  Projet Girl Scout
1  2015            1            +      230240      Projet Cobra
2  2015            4            x      240101  Projet Bee - LOT 4
3  2016            3            x      294007      Projet Templer
4  2017            1            +      304283      Projet Alpha

      DateCreation          Statut InterlocuteurMetier \
0  2014-04-12 00:00:00          RAS      Joseph DICKENS
1  2015-05-04 00:00:00      Termine      Madeleine BROWN
2  2015-03-27 00:00:00  En attente de decision      Leonard JONES
3  2016-05-28 00:00:00          GO      Madeleine BROWN
4  2017-01-11 00:00:00          GO      Madeleine BROWN

      ChefDeProjet      DateDemande      DateReponse TypeDemande \
0      Jane SHARP  2014-01-01 00:00:00  2014-09-01 00:00:00  Catalogue
1      Gavin KING  2014-01-01 00:00:00  2017-06-10 00:00:00      Projet
2      Ian HEMMINGS  2014-05-20 00:00:00  2017-08-21 00:00:00  Evolution
3  David JACKSON  2014-06-24 00:00:00  2016-08-25 00:00:00      Projet
4  David JACKSON  2014-07-28 00:00:00  2017-02-24 00:00:00  Evolution

      TypeDevis      CauseReestimation  CodeApplicAtion  DateDebutTravaux \
0  Reestime          Autre      A4701; A7843      41929
1  Initial          NaN      A3758; A3774      42917
2  Reestime          Autre      A4452; A5146      43021
3  Reestime          Autre      A4317; A8067      42675
4  Reestime  Ajout contributeur      A3774; A5876      42827

      DateFinTravaux  DateDebutHomologation  DevisBudgetJH  DevisBudgetEUR \
0  2015-06-21 00:00:00  2015-08-28 00:00:00      1900      2,6 MEUR
1  2017-12-14 00:00:00  2018-01-28 00:00:00      2000      2,7 MEUR
2  2018-03-08 00:00:00  2018-04-05 00:00:00      1400      2 MEUR
3  2017-08-10 00:00:00  2017-10-01 00:00:00      2200      3 MEUR
4  2017-08-29 00:00:00  2017-11-19 00:00:00      1500      2,2 MEUR

      DevisBudgetJHEUR  DevisBudgetInfraEUR  Appli1  Appli2
0      1235000      1365000.0  A4701  A7843
1      1300000      1400000.0  A3758  A3774
2      910000      1090000.0  A4452  A5146
3      1430000      1570000.0  A4317  A8067
4      975000      1225000.0  A3774  A5876

```

**Modification de la colonne DevisBudgetEUR** Cette colonne contient des chaînes de caractères. En effet, le nombre est ici suivi de “MEUR” ou “KEUR” selon les cas.

Proposez un script permettant de créer une colonne “DevisBudgetEUR\_clean” contenant les valeurs présentes dans “DevisBudgetEUR” transformées en nombres. Il faudra ici multiplier le nombre par 1000000 dans le cas “MEUR” et par 1000 dans le cas “KEUR”.

Vous pourrez par exemple créer une fonction pour convertir un budget en nombre puis l’utiliser dans un `.apply()`.

```
[411]: # Votre code ci-dessous

projets['DevisBudgetEUR_clean'] = projets['DevisBudgetEUR']

def effacer_virgule(x):
    position_r1 = x.find(',')
    return(x[0:position_r1] + x[position_r1 + 1:len(x)])

#effacer_virgule('2,5')

j = 0
for i in projets['DevisBudgetEUR']:
    #print(i)
    position = i.find(' ')
    #print(position)
    projets['DevisBudgetEUR_clean'][j] = i[0:position]
    projets['DevisBudgetEUR_clean'][j] =_
    →effacer_virgule(projets['DevisBudgetEUR_clean'][j])
    if i[position + 1] == 'M':
        projets['DevisBudgetEUR_clean'][j] = projets['DevisBudgetEUR_clean'][j]_
    →+ '00000'
    if i[position + 1] == 'K':
        projets['DevisBudgetEUR_clean'][j] = projets['DevisBudgetEUR_clean'][j]_
    →+ '00'
    if i[position - 2] != ',':
        projets['DevisBudgetEUR_clean'][j] = projets['DevisBudgetEUR_clean'][j]_
    →+ '0'
    j += 1

projets.head()
```

```
[411]:
```

	Annee	NumeroVersion	FicheDeDemande	ReferenceID	NomDuProjet	\
0	2014	1	x	218486	Projet Girl Scout	
1	2015	1	+	230240	Projet Cobra	
2	2015	4	x	240101	Projet Bee - LOT 4	
3	2016	3	x	294007	Projet Templer	
4	2017	1	+	304283	Projet Alpha	

	DateCreation	Statut	InterlocuteurMetier	\
0	2014-04-12 00:00:00	RAS	Joseph DICKENS	
1	2015-05-04 00:00:00	Termine	Madeleine BROWN	
2	2015-03-27 00:00:00	En attente de decision	Leonard JONES	
3	2016-05-28 00:00:00	GO	Madeleine BROWN	
4	2017-01-11 00:00:00	GO	Madeleine BROWN	

	ChefDeProjet	DateDemande	DateReponse	TypeDemande	\
0	Jane SHARP	2014-01-01 00:00:00	2014-09-01 00:00:00	Catalogue	
1	Gavin KING	2014-01-01 00:00:00	2017-06-10 00:00:00	Projet	
2	Ian HEMMINGS	2014-05-20 00:00:00	2017-08-21 00:00:00	Evolution	
3	David JACKSON	2014-06-24 00:00:00	2016-08-25 00:00:00	Projet	
4	David JACKSON	2014-07-28 00:00:00	2017-02-24 00:00:00	Evolution	

	TypeDevis	CauseReestimation	CodeApplicAtion	DateDebutTravaux	\
0	Reestime	Autre	A4701; A7843	41929	
1	Initial	NaN	A3758; A3774	42917	
2	Reestime	Autre	A4452; A5146	43021	
3	Reestime	Autre	A4317; A8067	42675	
4	Reestime	Ajout contributeur	A3774; A5876	42827	

	DateFinTravaux	DateDebutHomologation	DevisBudgetJH	DevisBudgetEUR	\
0	2015-06-21 00:00:00	2015-08-28 00:00:00	1900	2,6 MEUR	
1	2017-12-14 00:00:00	2018-01-28 00:00:00	2000	2,7 MEUR	
2	2018-03-08 00:00:00	2018-04-05 00:00:00	1400	2 MEUR	
3	2017-08-10 00:00:00	2017-10-01 00:00:00	2200	3 MEUR	
4	2017-08-29 00:00:00	2017-11-19 00:00:00	1500	2,2 MEUR	

	DevisBudgetJHEUR	DevisBudgetInfraEUR	Appli1	Appli2	DevisBudgetEUR_clean
0	1235000	1365000.0	A4701	A7843	2600000
1	1300000	1400000.0	A3758	A3774	2700000
2	910000	1090000.0	A4452	A5146	2000000
3	1430000	1570000.0	A4317	A8067	3000000
4	975000	1225000.0	A3774	A5876	2200000

**Création d'une colonne DevisBudgetJH\_Rank** Créez une nouvelle colonne DevisBudgetJH\_Rank en utilisant la méthode .rank() de pandas appliquée à la colonne "DevisBudgetJHEUR".

```
[423]: # Votre code ci-dessous

projets['DevisBudgetJH_Rank'] = projets['DevisBudgetJHEUR'].rank()

projets.head()
```

[423]:	Annee	NumeroVersion	FicheDeDemande	ReferenceID	NomDuProjet	\
0	2014	1	x	218486	Projet Girl Scout	

1	2015	1	+	230240	Projet Cobra
2	2015	4	x	240101	Projet Bee - LOT 4
3	2016	3	x	294007	Projet Templer
4	2017	1	+	304283	Projet Alpha

	DateCreation		Statut	Interlocuteur	Metier \
0	2014-04-12 00:00:00		RAS	Joseph DICKENS	
1	2015-05-04 00:00:00		Termine	Madeleine BROWN	
2	2015-03-27 00:00:00	En attente de decision		Leonard JONES	
3	2016-05-28 00:00:00		GO	Madeleine BROWN	
4	2017-01-11 00:00:00		GO	Madeleine BROWN	

	ChefDeProjet	DateDemande	DateReponse	TypeDemande \
0	Jane SHARP	2014-01-01 00:00:00	2014-09-01 00:00:00	Catalogue
1	Gavin KING	2014-01-01 00:00:00	2017-06-10 00:00:00	Projet
2	Ian HEMMINGS	2014-05-20 00:00:00	2017-08-21 00:00:00	Evolution
3	David JACKSON	2014-06-24 00:00:00	2016-08-25 00:00:00	Projet
4	David JACKSON	2014-07-28 00:00:00	2017-02-24 00:00:00	Evolution

	TypeDevis	CauseReestimation	CodeApplicAtion	DateDebutTravaux \
0	Reestime	Autre	A4701; A7843	41929
1	Initial	NaN	A3758; A3774	42917
2	Reestime	Autre	A4452; A5146	43021
3	Reestime	Autre	A4317; A8067	42675
4	Reestime	Ajout contributeur	A3774; A5876	42827

	DateFinTravaux	DateDebutHomologation	DevisBudgetJH	DevisBudgetEUR \
0	2015-06-21 00:00:00	2015-08-28 00:00:00	1900	2,6 MEUR
1	2017-12-14 00:00:00	2018-01-28 00:00:00	2000	2,7 MEUR
2	2018-03-08 00:00:00	2018-04-05 00:00:00	1400	2 MEUR
3	2017-08-10 00:00:00	2017-10-01 00:00:00	2200	3 MEUR
4	2017-08-29 00:00:00	2017-11-19 00:00:00	1500	2,2 MEUR

	DevisBudgetJHEUR	DevisBudgetInfraEUR	Appli1	Appli2	DevisBudgetEUR_clean \
0	1235000	1365000.0	A4701	A7843	2600000
1	1300000	1400000.0	A3758	A3774	2700000
2	910000	1090000.0	A4452	A5146	2000000
3	1430000	1570000.0	A4317	A8067	3000000
4	975000	1225000.0	A3774	A5876	2200000

	DevisBudgetJH_Rank
0	128.5
1	137.0
2	90.0
3	154.0
4	98.5

**Vérification des budgets** Assurez-vous que DevisBudgetJHEUR + DevisBudgetInfraEUR = DevisBudgetEUR\_Clean.

```
[506]: ### Votre code ci-dessous

somme = projets['DevisBudgetJHEUR'] + projets['DevisBudgetInfraEUR']

projets['DevisBudgetEUR_clean'] = projets['DevisBudgetEUR_clean'].astype(float)

#somme == projets['DevisBudgetEUR_clean']

j = 0

for i in range(len(somme)):
    if projets['DevisBudgetEUR_clean'][i] != somme[i] and np.
    →abs(projets['DevisBudgetEUR_clean'][i] - somme[i]) > 10**(-6):
        print("égalité non respecté à la ligne d'indice", i)
    else:
        j += 1

if j == len(somme):
    print("Ok on a bien projets['DevisBudgetJHEUR'] +
    →projets['DevisBudgetInfraEUR'] = projets['DevisBudgetEUR_clean']")

# on aurait pu verifier juste avec somme == projets['DevisBudgetEUR_clean'] mais
    →on aurait dû verfier
# qu'on ait bien True à chaque ligne
# Avec ce tableau qui n'est pas "trop" grand, utilisé une boucle permet de voir
    →de suite en sortie s'il y a ou non un probleme sur une des lignes
```

Ok on a bien projets['DevisBudgetJHEUR'] + projets['DevisBudgetInfraEUR'] =  
projets['DevisBudgetEUR\_clean']

## 2.2 Base des applications

```
[19]: applications.head()
```

```
[19]:
```

	AppCode	Name	RiskLevel	LastResilienceTest	NombreUtilisateur2017	\
0	A6205	Yearin	HIGH	42795	9704	
1	A7527	Goodsilron	LOW	41477	6969	
2	A7194	Condax	MEDIUM	42572	1754	
3	A4701	Opentech	MEDIUM	41049	12692	
4	A1956	Golddex	HIGH	41647	7559	

	NombreUtilisateur2016
0	9740
1	6938

```
2           1716
3          12653
4           7594
```

### Comptez le nombre de valeurs nulles dans le DataFrame applications

```
[510]: # Votre code ci-dessous

print('Nombre de valeurs nulles :', applications.isnull().sum().sum())
```

Nombre de valeurs nulles : 0

### Comptez le nombre d'AppCode dupliqués et le nombre d'AppCode uniques dans applications

```
[516]: # Votre code ci-dessous

#dup = applications.pivot_table(index = ['AppCode'], aggfunc = 'size')

nb_duplique = len(applications['AppCode']) - applications['AppCode'].nunique()
nb_unique = applications['AppCode'].nunique()

print("Nombre d'AppCode dupliqués :", nb_duplique)
print("Nombre d'AppCode uniques :", nb_unique)
```

Nombre d'AppCode dupliqués : 0  
Nombre d'AppCode uniques : 64

### Affichez la fréquence d'apparition de chaque RiskLevel

```
[540]: # Votre code ci-dessous

risk_level = applications.pivot_table(index = ['RiskLevel'], aggfunc = 'size')
risk_level
```

```
[540]: RiskLevel
HIGH           13
LOW            10
MEDIUM        22
VERY HIGH      4
VERY LOW       15
dtype: int64
```

### Ecrivez un script permettant de transformer LastResilienceTest en date

```
[543]: # Votre code ci-dessous

applications['LastResilienceTest'] = pd.
    →to_datetime(applications['LastResilienceTest'], unit='D', origin='1899-12-30')
#applications['LastResilienceTest']
```

```
applications.head()
```

```
[543]:  AppCode      Name RiskLevel LastResilienceTest  NombreUtilisateur2017 \
0    A6205    Yearin      HIGH      2017-03-01             9704
1    A7527  Goodsilron      LOW      2013-07-22             6969
2    A7194    Condax      MEDIUM  2016-07-21             1754
3    A4701    Opentech      MEDIUM  2012-05-20            12692
4    A1956    Golddex      HIGH      2014-01-08             7559
```

```
      NombreUtilisateur2016
0                9740
1                6938
2                1716
3            12653
4                7594
```

Utilisez la fonction `describe` pour afficher des statistiques descriptives des 2 dernières colonnes i.e. `NombreUtilisateur2017` & `NombreUtilisateur2016`

```
[544]: # Votre code ci-dessous
      applications.describe()
```

```
[544]:      NombreUtilisateur2017  NombreUtilisateur2016
count                64.000000                64.000000
mean                 6735.562500                6734.546875
std                  3721.294018                3726.222215
min                   166.000000                117.000000
25%                  3295.750000                3262.750000
50%                  6776.000000                6751.000000
75%                  9917.000000                9894.750000
max                 12692.000000                12653.000000
```

## 2.3 Base des personnes

```
[569]: personnes.head()
```

```
[569]:      Prenom      Nom      Mail      ID \
0    Charles  Smith      charles.smith@tgp.com  A692817085
1  Madeleine  Chapman  madeleine.chapman@tgp.com  A185252110
2     Colin   King      colin.king@tgp.com  A769284797
3     Piers  Skinner  piers.skinner@tgp.com  A370950207
4     Gavin  Turner      gavin.turner@tgp.com  A234744045
```

```
      Role  ExperienceValeur
0  Developpeur Junior      1
1  Developpeur Junior      1
2  Developpeur Junior      4
```

3	Developpeur Junior	1
4	Developpeur Junior	0

**Vérification des adresses mails** Vérifiez que chaque adresse mail a bien le format suivant "xxx.xxx@tgp.com".

Il y a plusieurs manière de faire cela, par exemple : - Utiliser une expression régulière avec (i) [a-z] représentant n'importe quel caractère de l'alphabet, (ii) le signe + qui, placé derrière un caractère, signifie qu'il apparaît une ou plusieurs fois d'affilé, (iii) \. qui représente un point et enfin ^ et \$ représentant respectivement le début et la fin d'une chaîne de caractères. - Vérifier avec une boucle que tous les caractères avant "@" sont des lettres et qu'il y a un unique point avec au moins une lettre avant et après celui-ci. Puis vérifier que "tgp.com" est présent après le signe "@".

```
[26]: # Exemple d'utilisation des regex vérifier un pattern
# Ici - 1 ou plusieurs lettres, suivies de 1 ou plusieurs chiffres, suivies d'un
      ↳point

bool(re.search('[a-z]+[0-9]+\.$', 'abc12.'))
```

[26]: True

Quelles sont les lignes pour lesquelles l'adresse mail (colonne Mail) n'a pas le bon format ?

```
[574]: # Votre code ci-dessous

personnes2 = personnes.copy() # nous utilisons une copie du tableau pour ne pas
      ↳avoir a reimporter le tableau si la modification est fausse
# néanmoins on aurait pu appliquer le code direct sur le tableau de depart
# on modifiera le tableau de depart a chaque fois qu'on obtiendra la reponse
      ↳souhaitee par des boucles puisque le tableau reste de taille raisonnable

pattern = re.compile(r"^[a-zA-Z0-9_.-]+\.[a-zA-Z0-9_.-]+@tgp.com$")

personnes2['isemail'] = personnes2['Mail'].apply(lambda x: True if pattern.
      ↳match(x) else False)
wrong_mail_df = personnes.loc[personnes2['isemail'] == False]

wrong_mail_df
```

[574]:	Prenom	Nom	Mail	ID	\
44	Joshua	Skinner	joshua.skinner@tgp	A431854713	
107	Isaac	Chapman	isaac.chapmantgp.com	A627677732	
158	Jane	Randall	jane.randall	A761729136	
249	Ryan	Watson	ryan@tgp.com	A494768410	
304	Sophie	Wilson	sophie.wilsons@tgp.com	A197223116	
313	Phil	Mackenzie	phil.mackenzie@com	A582718828	

	Role	Experience	Valeur
44	Developpeur	Junior	3
107	Developpeur	Junior	1
158	Developpeur	Junior	3
249	Developpeur	Senior	3
304	Chef de projet		16
313	COO		39

**Modification des adresses mails erronées** Proposez un script permettant de modifier les adresses mails ayant un format erroné. Pour cela, on remplacera les valeurs erronées par `prenom.nom@tgp.com` et cela en minuscule.

Attention : dans les colonnes "Nom" et "Prenom", il pourrait y avoir des espaces avant ou après le nom... Cela pourrait causer des problèmes. Il faudra utiliser la méthode `.strip()` pour supprimer ces espaces inutiles.

```
[594]: # Votre code ci-dessous

wrong_mail_df['Prenom'] = wrong_mail_df['Prenom'].str.strip()
wrong_mail_df['Nom'] = wrong_mail_df['Nom'].str.strip()

wrong_mail_df['Full_Name'] = np.where(wrong_mail_df[['Prenom', 'Nom']].eq('').
    →any(axis=1),
                                     wrong_mail_df['ID'],
                                     wrong_mail_df[['Prenom', 'Nom']].apply(' '.join,
    →axis=1))

wrong_mail_df['Mail'] = wrong_mail_df.Full_Name.str.replace('\s+', '.') + '@tgp.
    →com'

wrong_mail_df['Mail'] = wrong_mail_df['Mail'].str.lower()

wrong_mail_df

# on pourrait appliquer cela sur le tableau entier pour changer tous les mails,
    →y compris ceux ayant le bon format
# puisque dans la question precedente on n'a verifié que le format xxx.xxx@tgp.
    →com
# sans préciser que xxx ne doit contenir que des lettres minuscules

for i in wrong_mail_df.index:
    personnes['Mail'][i] = wrong_mail_df['Mail'][i]
```

On peut d'ailleurs vérifier maintenant que l'ensemble des adresses mail ont le bon format !

```
[596]: # Votre code ci-dessous

pattern = re.compile(r"^[a-zA-Z0-9_.-]+\.[a-zA-Z0-9_.-]+@tgp.com$")
```

```

personnes2['isemail'] = personnes2['Mail'].apply(lambda x: True if pattern.
→match(x) else False)
wrong_mail_df2 = personnes.loc[personnes2['isemail'] == False]

wrong_mail_df2

# le tableau est vide donc ils ont tous le format xxx.xxx@tgp.com

```

```

[596]: Empty DataFrame
Columns: [Prenom, Nom, Mail, ID, Role, ExperienceValeur]
Index: []

```

**Création d'un clé de jointure** Nous aurons besoin d'une clé de correspondance pour réaliser une jointure entre les DataFrame personneset projets.

En effet, dans la colonne "ChefDeProjet" de la base projet le format est Prenom NOM.

Créez une nouvelle colonne "Cle" concaténant, pour chaque ligne, la valeur de la colonne "Prenom", suivie d'un espace et enfin de la valeur de la colonne "Nom" en majuscule.

Attention : dans les colonnes "Nom" et "Prenom", il pourrait y avoir des espaces avant ou après le nom... Cela pourrait causer des problèmes. Il faudra utiliser la méthode .strip() pour supprimer ces espaces inutiles.

```

[614]: # Votre code ci-dessous

personnes['Cle'] = personnes['Prenom'] + ' ' + personnes['Nom'].str.upper()

personnes.head()

```

```

[614]:

```

	Prenom	Nom	Mail	ID	\
0	Charles	Smith	charles.smith@tgp.com	A692817085	
1	Madeleine	Chapman	madeleine.chapman@tgp.com	A185252110	
2	Colin	King	colin.king@tgp.com	A769284797	
3	Piers	Skinner	piers.skinner@tgp.com	A370950207	
4	Gavin	Turner	gavin.turner@tgp.com	A234744045	

	Role	ExperienceValeur	Cle
0	Developpeur Junior	1	Charles SMITH
1	Developpeur Junior	1	Madeleine CHAPMAN
2	Developpeur Junior	4	Colin KING
3	Developpeur Junior	1	Piers SKINNER
4	Developpeur Junior	0	Gavin TURNER

## 2.4 Base des logs

```
[623]: logs.head()
```

```
[623]:
```

	ProjectKey	TaskAssignee	TaskKey	DateFInalLog	MDLogged
0	218486	A810043709	M1	29/01/2015	1
1	218486	A810043709	M1	11/01/2015	1
2	218486	A810043709	M1	29/01/2015	1
3	218486	A810043709	M1	04/03/2015	1
4	218486	A722019848	M2	03/05/2015	1

### Comptez le nombre de valeurs nulles dans la base

```
[624]: # Votre code ci-dessous  
print('Nombre de valeurs nulles :', logs.isnull().sum().sum())
```

Nombre de valeurs nulles : 0

### Transformez la colonne DateFInalLog en vraie date (datetime)

```
[632]: # Votre code ci-dessous  
  
logs['DateFInalLog'] = pd.to_datetime(logs['DateFInalLog'])  
#logs['DateFInalLog']  
  
logs.head()
```

```
[632]:
```

	ProjectKey	TaskAssignee	TaskKey	DateFInalLog	MDLogged
0	218486	A810043709	M1	2015-01-29	1
1	218486	A810043709	M1	2015-11-01	1
2	218486	A810043709	M1	2015-01-29	1
3	218486	A810043709	M1	2015-04-03	1
4	218486	A722019848	M2	2015-03-05	1

### Comptez le nombre de valeurs unique dans la colonne MDLogged

```
[643]: # Votre code ci-dessous  
  
#pd.unique(logs['MDLogged'])[0]  
  
logs.pivot_table(index = ['MDLogged'], aggfunc = 'size')
```

```
[643]: MDLogged  
1      347895  
dtype: int64
```

Cette colonne est-elle réellement utile ? Si oui, gardez-la, sinon, supprimez-la.

```
[654]: # Votre code ci-dessous  
# je ne pense pas qu'elle soit utile puisqu'identique a chaque ligne
```

```
logs = logs.drop(['MDLogged'],axis = 1)
logs
```

```
[654]:
```

	ProjectKey	TaskAssignee	TaskKey	DateFinalLog
0	218486	A810043709	M1	2015-01-29
1	218486	A810043709	M1	2015-11-01
2	218486	A810043709	M1	2015-01-29
3	218486	A810043709	M1	2015-04-03
4	218486	A722019848	M2	2015-03-05
...	...	...	...	...
347890	390064	A382829264	M154	2017-06-19
347891	390064	A382829264	M154	2017-07-25
347892	390064	A382829264	M154	2017-05-18
347893	390064	A382829264	M154	2017-05-29
347894	390064	A382829264	M154	2017-07-16

[347895 rows x 4 columns]

## 2.5 Base des tâches

```
[649]: taches.head()
```

```
[649]:
```

	ProjectKey	TaskAssignee	Priority	TaskKey	MDPlanned	MDUpdatedPlanned	\
0	218486	A810043709	VERY LOW	M1	5		4
1	218486	A722019848	LOW	M2	8		12
2	218486	A69888201	MEDIUM	M3	9		7
3	218486	A434385931	MEDIUM	M4	2		4
4	218486	A331360422	MEDIUM	M5	4		2

	TaskType	Location
0	CODING	Bangalore
1	STRUCTURE CHANGE	Bangalore
2	BUG	Paris
3	STRUCTURE CHANGE	Bangalore
4	CODING	Bangalore

### Comptez le nombre de valeurs nulles dans le DataFrame taches

```
[650]: # Votre code ci-dessous
print('Nombre de valeurs nulles :',taches.isnull().sum().sum())
```

Nombre de valeurs nulles : 0

Nous nous arrêterons là pour la préparation de ce dernier DataFrame.

### 3 Sauvegarde des jeux de données

Sauvegardez chacun des DataFrames modifiés dans un dossier "03 - Data prep" que vous aurez préalablement créé.

Vous pouvez les enregistrer au format Excel ou CSV. Une bonne méthode, pour éviter toute confusion avec les fichiers de départ et de nommer un peu différemment ces nouveaux fichiers. Par exemple, vous pourriez les appeler "\_prep.csv" (en remplaçant bien sûr par le nom approprié).

```
[651]: # Votre code ci-dessous

out_path = './03 - Data prep/projets_prep.xlsx'
writer = pd.ExcelWriter(out_path , engine='xlsxwriter')
projets.to_excel(writer)
writer.save()

out_path1 = './03 - Data prep/applications_prep.xlsx'
writer1 = pd.ExcelWriter(out_path1 , engine='xlsxwriter')
applications.to_excel(writer1)
writer1.save()

out_path2 = './03 - Data prep/personnes_prep.xlsx'
writer2 = pd.ExcelWriter(out_path2 , engine='xlsxwriter')
personnes.to_excel(writer2)
writer2.save()

out_path3 = './03 - Data prep/logs_prep.xlsx'
writer3 = pd.ExcelWriter(out_path3 , engine='xlsxwriter')
logs.to_excel(writer3)
writer3.save()

out_path4 = './03 - Data prep/taches_prep.xlsx'
writer4 = pd.ExcelWriter(out_path4 , engine='xlsxwriter')
taches.to_excel(writer4)
writer4.save()
```

#### 3.1 Conclusion

Nous avons abordé ici différentes méthodes de préparation des données. Cette étape est capitale car des données de mauvaise qualité empêchent de réaliser certaines analyses et/ou conduisent à des analyses fausses.

Il est donc nécessaire (i) d'adresser les problèmes de qualité de la donnée dans toutes les bases de données que vous pourrez recevoir à l'avenir et (ii) tenter au maximum de les corriger (mais il faut à minima en avoir conscience).

Les méthodes vues ici ne sont bien sûr pas exhaustives. Soyez créatifs, posez-vous des questions et utilisez votre esprit critique !

Pour vous aider, voici des indicateurs usuels de la qualité de données : - l'exactitude : les données

reflètent-elles l'activité / le métier ? Elle se mesure en comptant le taux de valeurs incorrectes dans les données ; - la complétude : toutes les données nécessaires à l'observation de l'activité métier sont-elles présentes ? Elle se mesure en comptant le nombre de valeurs manquantes dans la base ; - la conformité : toutes les valeurs de l'activité respectent-elles les règles métier spécifiées ? ; - l'intégrité : les données sont-elles cohérentes les unes par rapport aux autres ? ; - la consistance : les données renferment-elles des doublons ? ; - la disponibilité : les données sont-elles disponibles dans des délais raisonnables ?

[ ]: