Manipulation de données avec Python pour des travaux d'investigation

Partie 2 - Observation des données

NOE BOUDON - CECILE LIN

Année 2022 - 2023

Contents

Partie 2 - Observation des données

Pour rappel - 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, 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 réalisés par TGP. Entre autres, (i) le nom du chef de projet, (ii) les dates des différentes étapes, (iii) les applications concernées par chaque projets... - `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) passés 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.

On remarque que ces différents jeux de données communiquent entre eux ! Chaque **projet** impacte des **applications**, des **personnes** ont travaillé sur ces différents projets (dont un chef de projet qui l'a piloté et est responsable de son bon déroulement), différentes **tâches** ont été réalisées sur ces projets et enfin, les collaborateurs ont renseigné chaque jour le temps passé sur chacune des tâches sous forme de **logs**.

A ce stade, il est possible d'analyser les bases de données séparément. C'est ce que nous ferons dans la première partie de ce notebook. En revanche, Python donne aussi la possibilité de croiser ces données à l'aide de jointures. Cela permet d'analyser ensemble des informations issues de sources disjointes, donc d'avoir une vue transversale de l'univers à analyser et de produire des analyses à forte valeur ajoutée. C'est ce que nous ferons dans la seconde partie.

Travail à réaliser dans ce notebook

Ce notebook est dédié à la visualisation de données dans un but d'investigation. J'ai généré tous les graphiques et les ai accompagnés d'un commentaire succinct (il sera en italique après les visualisations).

Le but de cette partie est donc de reproduire l'ensemble des graphiques.

J'ai utilisé 3 packages pour réaliser ces visualisations : *matplotlib*, *seaborn* et *plotly*. Je vous dirai, pour chaque graphique, le package utilisé. Cependant, vous êtes libres d'utiliser d'autres packages et/ou de paramétrer les graphiques de manière différente. L'objectif est simplement de faire ressortir la même information que sur mes graphiques.

[3]:
```
# Pour ignorer les warnings
import warnings
warnings.filterwarnings('ignore')

# Pour la manipulation de données
import pandas as pd

# Package de visualisations vu en cours
import matplotlib.pyplot as plt
```

```
# Autre package de visualisation basé sur matplotlib
import seaborn as sns

# Autre package de visualisation
# Il permet en particuler de :
# - Créer des visualisations dynamiques (possibilité d'intéragir avec le graphique)
# - Créer des Dashboards avec dash (nous n'en feront pas ici)
!pip install plotly
import plotly.express as px


# Les visualisations générées avec matplotlib auront maintenant une taille de 15x5
plt.rcParams["figure.figsize"] = (15,5)
```

```
Requirement already satisfied: plotly in c:\users\lince\anaconda3\lib\site-
packages (5.11.0)
Requirement already satisfied: tenacity>=6.2.0 in
c:\users\lince\anaconda3\lib\site-packages (from plotly) (8.1.0)
```

1 Analyse des disjointes des jeux de données

1.1 Les applications

Commencez par charger les données des applications que vous avez enregistré à la fin de la partie 1

[13]:
```
# votre code ci-dessous
applications = pd.read_excel('./03 - Data prep/applications_prep.xlsx')
applications
#on peut eventuellement supprimer la colonne 'Unnamed: 0'
```

[13]:

	Unnamed: 0	AppCode	Name	RiskLevel	LastResilienceTest
0	0	A6205	Yearin	HIGH	2017-03-01
1	1	A7527	Goodsilron	LOW	2013-07-22
2	2	A7194	Condax	MEDIUM	2016-07-21
3	3	A4701	Opentech	MEDIUM	2012-05-20
4	4	A1956	Golddex	HIGH	2014-01-08
..	...	...	...	...	...
59	59	A9801	Mathtouch	LOW	2018-09-17
60	60	A3156	Rantouch	LOW	2012-02-08
61	61	A8819	Silis	LOW	2015-02-27
62	62	A2896	Plussunin	LOW	2013-01-01
63	63	A5876	Plexzap	MEDIUM	2015-03-25

```
    NombreUtilisateur2017  NombreUtilisateur2016
0                    9704                   9740
1                    6969                   6938
2                    1754                   1716
3                   12692                  12653
4                    7559                   7594
..                    ...                    ...
59                  11578                  11625
60                   9349                   9321
61                  12166                  12156
62                  12228                  12237
63                  10945                  10977

[64 rows x 7 columns]
```

Q1 - Nombre d'utilisateurs et tests de résilience

Tracez un *barchart* présentant le nombre d'utilisateurs en 2017 par projets. Les barres relatives aux projets pour lesquels le dernier test de résilience à été réalisé en 2014 ou avant doivent être d'une couleur différente des projets pour lesquels le test a été réalisé plus récemment.

Pour cela, vous devrez (i) créer une nouvelle colonne "test_avant_2014" ayant pour valeur True si la date du dernier test de résilience est antérieure au 2015-01-01 puis (ii) tracer un bar chart.

Vous pouvez utiliser les fonctions `histogram` ou `bar` de plotly express pour réaliser ce graphique. Il faudra renseigner les paramètres `x`, `y`, `color` et `color_discrete_sequence`.

[14]:
```
# votre code ci-dessous

applications['test_avant_2014'] = applications['LastResilienceTest'] < pd.
 ↪to_datetime('2015-01-01')
applications.sort_values(by=['NombreUtilisateur2017'], inplace=True,␣
 ↪ascending=False)
fig = px.bar(applications, x='AppCode', y='NombreUtilisateur2017', color =␣
 ↪'test_avant_2014',color_discrete_sequence=['red', 'gray']).
 ↪update_xaxes(categoryorder = 'total descending')
fig.show()

#applications
```

Commentaires : On remarque que certaines applications, pourtant très utilisées, n'ont pas subi de test de résilience depuis longtemps...

Q2 - Nombre d'utilisateurs par criticité des applications

Tracez un *treemap* montrant le nombre d'utilisateurs par année du dernier test de résilience et niveau de criticité.

Pour cela, vous pouvez (i) réaliser un `groupby` sur le RiskLevel et l'année du dernier test de résilience (cette colonne est à créer au préalable) afin de sommer le nombre d'utilisateur en 2017 puis (ii) tracer un treemap.

Vous pouvez utiliser la fonction `treemap` de plotly express pour réaliser ce graphique. Il faudra renseigner les paramètres `data_frame`, `path` et `values`.

[15]:
```
# votre code ci-dessous

applications['year'] = pd.DatetimeIndex(applications['LastResilienceTest']).year
#applications1 = applications.groupby(['year', 'RiskLevel',␣
 ↪'NombreUtilisateur2017'])

fig1 = px.treemap(applications, path = ['year', 'RiskLevel'], values =␣
 ↪'NombreUtilisateur2017')
fig1.show()
```

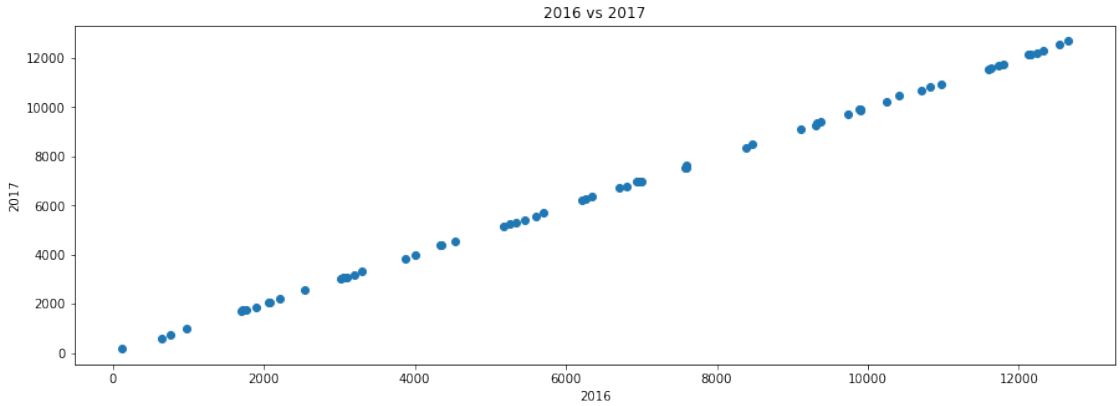
Commentaires : Les applications à haut niveau de risque (High) n'ayant pas subi de test de résilience depuis 2015 comptent 12k utilisateurs. Elles présentent un risque élevé de panne.

Q3 : Nombre d'utilisateur 2016 vs 2017

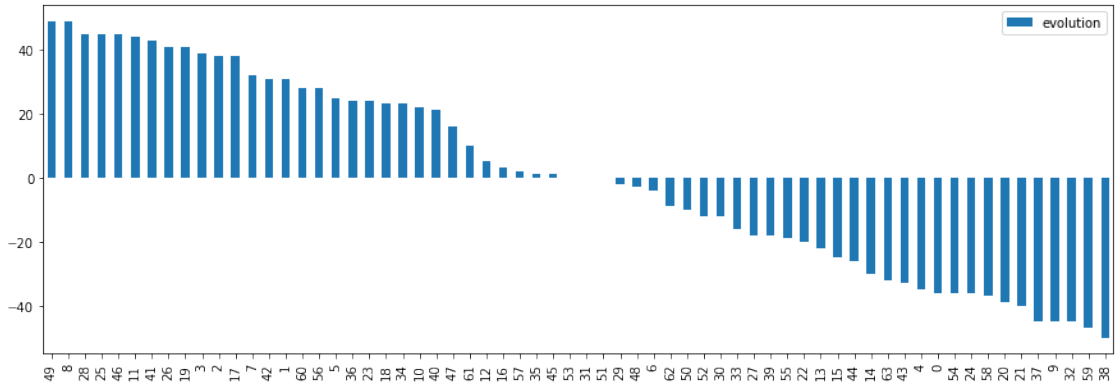
Tracez (i) un *scatterplot* du nombre d'utilisateurs en 2017 en fonction du nombre d'utilisateurs en 2016 et (ii) un *barchart* du gain de clients, entre 2016 et 2017, par applications.

Pour la deuxième partie, vous devrez créer une nouvelle colonne représentant ce gain de clients.

[16]:
```
# votre code ci-dessous

x1 = applications['NombreUtilisateur2016']
y1 = applications['NombreUtilisateur2017']
plt.scatter(x = x1, y = y1)
plt.title('2016 vs 2017')
plt.xlabel('2016')
plt.ylabel('2017')

plt.show()
```



[17]:
```
# votre code ci-dessous

applications['evolution']=applications['NombreUtilisateur2017'].
 ↪sub(applications['NombreUtilisateur2016'], axis=0)
applications.sort_values(by=['evolution'], inplace=True, ascending=False)
applications.plot(y='evolution', kind='bar', xlabel = '')
plt.show()
```



Commentaire : Les nombres d'utilisateurs en 2016 et 2017 sont très corrélés. Les gains et pertes d'utilisateurs d'une année à l'autre sont très faibles comparés au nombre d'utilisateurs des applications.

1.2 Personnes

Commencez par charger les données des personnes que vous avez enregistré à la fin de la partie 1

[18]:
```
# votre code ci-dessous

personnes = pd.read_excel('./03 - Data prep/personnes_prep.xlsx')
personnes.head()
```

[18]:

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

```
1  Madeleine CHAPMAN
2         Colin KING
3      Piers SKINNER
4       Gavin TURNER
```

Q1 - Répartition des rôles au sein de l'entité

Tracez un *barchart* présentant le nombre de personnes pour chaque rôle.

Pour cela, vous pourrez au préalable utiliser la méthode `.value_counts()` ou réaliser un `.groupby()`.

Vous pouvez utiliser la fonction`bar` de plotly express pour réaliser ce graphique.

[19]:
```
# votre code ci-dessous

a = personnes['Role'].value_counts()
#print(a)

fig2 = px.bar(x = a.index , y = a.values, labels = {'x' : 'index', 'y': 'value'})
#fig2.update_layout(xaxis={'title': 'index'}, yaxis={'title':'value'})
fig2.show()
```

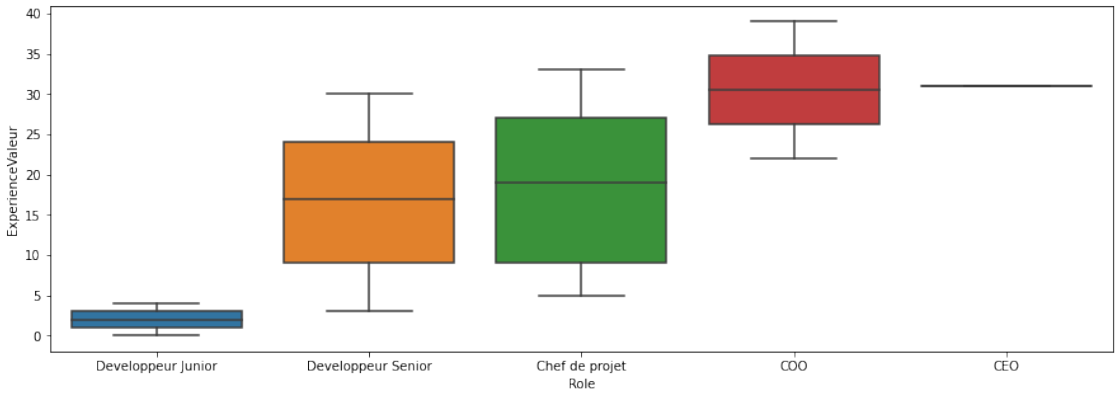
Commentaires : La répartition des rôles semble cohérente - 1 CEO, 2 COO et moins de chefs de projets que que développeurs seniors, eux-même moins nombreux que les développeurs juniors.

Q2 - Répartition des niveau d'expérience par rôles

Tracez un *boxplot* pour chaque rôle permettant d'observer le nombre d'années d'expérience des collaborateurs.

Vous pouvez utiliser la fonction`boxplot` de seaborn pour réaliser ce graphique.

[20]:
```
# votre code ci-dessous

sns.boxplot(data = personnes, x='Role', y='ExperienceValeur')
plt.show()
```



Commentaires : La répartition des postes semble également cohérente. Les développeurs juniors ont en particulier moins d'expérience que les développeurs séniors.

1.3 Projets

Commencez par charger les données des projets que vous avez enregistré à la fin de la partie 1

[21]:
```
# votre code ci-dessous
projets = pd.read_excel('./03 - Data prep/projets_prep.xlsx')
projets.head()
```

[21]:

	Unnamed: 0	Annee	NumeroVersion	FicheDeDemande	ReferenceID	NomDuProjet	DateCreation	Statut	InterlocuteurMetier	ChefDeProjet	...	DateFinTravaux	DateDebutHomologation	DevisBudgetJH	DevisBudgetEUR	DevisBudgetJHEUR	DevisBudgetInfraEUR	Appli1	Appli2	DevisBudgetEUR_clean	DevisBudgetJH_Rank
0	0	2014	1	x	218486	Projet Girl Scout	2014-04-12	RAS	Joseph DICKENS	Jane SHARP	...	2015-06-21	2015-08-28	1900	2,6 MEUR	1235000	1365000	A4701	A7843	2600000	128.5
1	1	2015	1	+	230240	Projet Cobra	2015-05-04	Termine	Madeleine BROWN	Gavin KING	...	2017-12-14	2018-01-28	2000	2,7 MEUR	1300000	1400000	A3758	A3774	2700000	137.0
2	2	2015	4	x	240101	Projet Bee - LOT 4	2015-03-27	En attente de decision	Leonard JONES	Ian HEMMINGS	...	2018-03-08	2018-04-05	1400	2 MEUR	910000	1090000	A4452	A5146	2000000	90.0
3	3	2016	3	x	294007	Projet Templer	2016-05-28	GO	Madeleine BROWN	David JACKSON	...	2017-08-10	2017-10-01	2200	3 MEUR	1430000	1570000	A4317	A8067	3000000	154.0
4	4	2017	1	+	304283	Projet Alpha	2017-01-11	GO	Madeleine BROWN	David JACKSON	...	2017-08-29	2017-11-19	1500	2,2 MEUR	975000	1225000	A3774	A5876	2200000	98.5

[5 rows x 27 columns]

Q1 - Répartition des statuts des projets

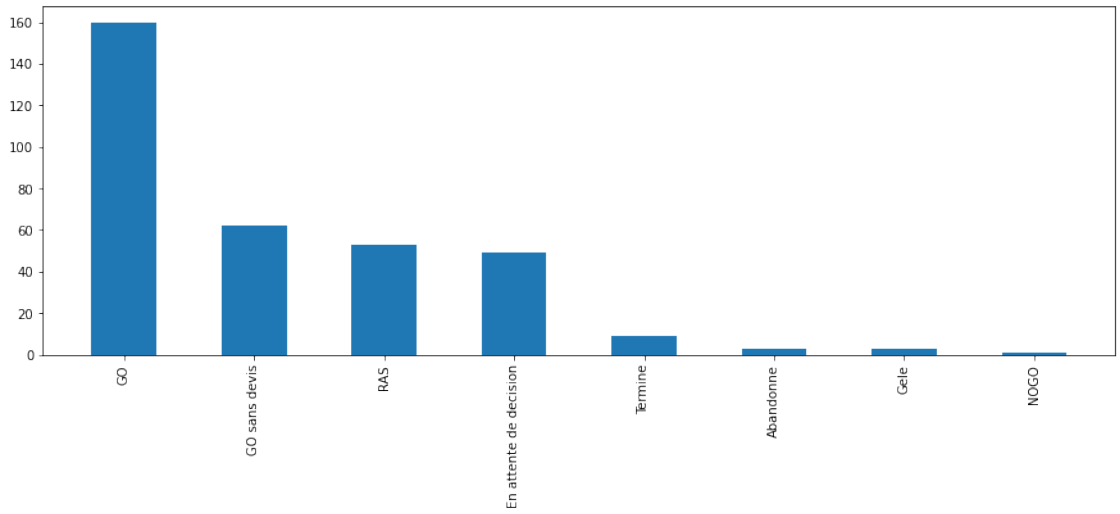
Tracez un *barchart* présentant le nombre de projets par statut.

Pour cela, vous pourrez au préalable utiliser la méthode `.value_counts()` ou réaliser un

`.groupby()`.

Vous pouvez utiliser la fonction`bar` de matplotlib ou appliquer directement `.plot.bar()`à votre DataFrame "groupé" ou votre série pour réaliser ce graphique.

[22]:
```
# votre code ci-dessous

b = projets['Statut'].value_counts()
#print(b)

plt.bar(x = b.index, height = b.values, width=0.5)
plt.xticks(rotation='vertical')
plt.show()
```



Commentaire : On observe de nombreux projets "GO", "GO sans devis" et "RAS" ces projets sont en cours de réalisation. Cependant, il n'y a que peu de projets "Terminés". Est-ce normal ? A-t-on bien renseigné les changement de statuts des projets qui ont été achevés ?

Q2 - Répartition des types de devis

Tracez un *piechart* présentant la quantité de projets par type de devis.

Pour cela, vous pourrez au préalable utiliser la méthode `.value_counts()` ou réaliser un `.groupby()`.

Vous pouvez utiliser la fonction `pie` de plotly express ou pour réaliser ce graphique.

[23]:
```
# votre code ci-dessous

c = projets.value_counts('TypeDevis')

fig4 = px.pie(values = c.values, names = c.index)
fig4.show()
```

Commentaires : Pas de réel commentaires ici. En revanche, il pourrait être intéressant de creuser les raisons des des réestimations des devis (1 projet sur 4) et les raisons de l'existence de ces projets sans devis (12%

quand même...).

1.4 Tâches

Commencez par charger les données des taches que vous avez enregistré à la fin de la partie 1

[24]:
```
# votre code ci-dessous

taches = pd.read_excel('./03 - Data prep/taches_prep.xlsx')
taches.head()
```

[24]:

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

	MDUpdatedPlanned	TaskType	Location
0	4	CODING	Bangalore
1	12	STRUCTURE CHANGE	Bangalore
2	7	BUG	Paris
3	4	STRUCTURE CHANGE	Bangalore
4	2	CODING	Bangalore

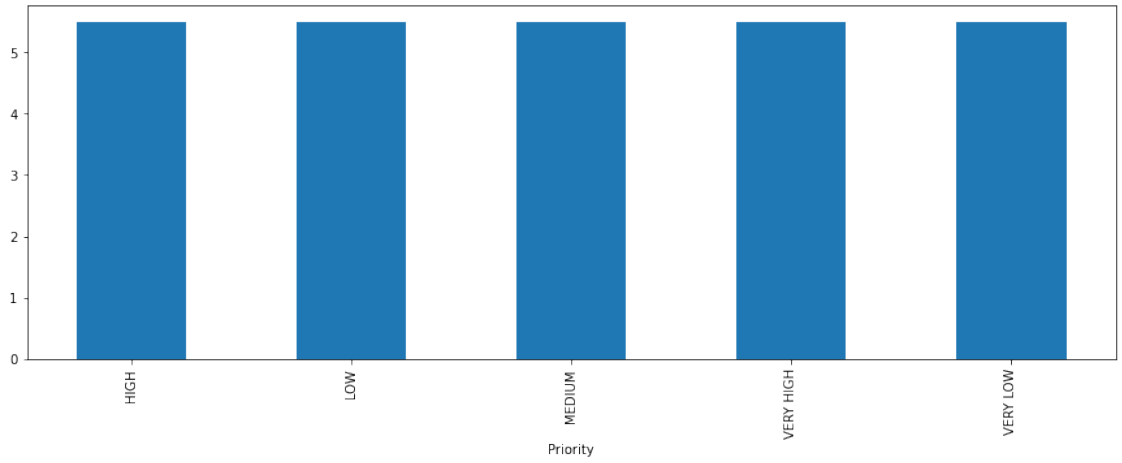
Q1 - Nombre moyen de jours-hommes (JH) par niveau de priorité

Tracez un *barchart* présentant le nombre moyen de jours-hommes (JH) par niveau de priorité.

Pour cela, vous pourrez réaliser un `.groupby()` et appliquer la méthode d'agrégation appropriée.

Vous pouvez utiliser la fonction`bar` de matplotlib ou appliquer directement `.plot.bar()`à votre DataFrame "groupé" pour réaliser ce graphique.

[25]:
```
# votre code ci-dessous

taches_new = taches.groupby(['Priority','MDPlanned'], as_index=False).mean().
 ↪groupby('Priority')['MDPlanned'].mean()

plt.bar(taches_new.index, taches_new.values, width=0.5)
plt.xticks(rotation='vertical')
plt.xlabel('Priority')
plt.show()
```



Commentaires : Le nombre de JH moyen par tâche ne varie que très peu en fonction de la priorité.

Q2 - Répartition des JH planifiés par type de tâche et priorité

Tracez un *sunburst* présentant la quantité de JH par type de tâche et priorité.

Pour cela, vous pourrez au préalable réaliser un `.groupby()`.

Vous pouvez utiliser la fonction `sunburst` de plotly express ou pour réaliser ce graphique.

[26]:
```
# votre code ci-dessous

fig5 = px.sunburst(taches, path = ['TaskType', 'Priority'], values = 'MDPlanned')

fig5.show()
```

Commentaires : La répartition des priorité est manifestement la même dans l'ensemble des types de tâches (cela est dû au fait que j'ai généré les données avec un code Python). Dans le monde réel, il est souvent possible de tirer des informations pertinentes de ce type de graphique. Typiquement, on aurait pu se rendre compte que la plupart des tâches hautement prioritaires sont localisées dans un ou quelques types de tâches. Cela aurait pu nous amener à une réflexion sur la pertinence de la répartition.

Q3 - Nombre de JH ajoutés par projet

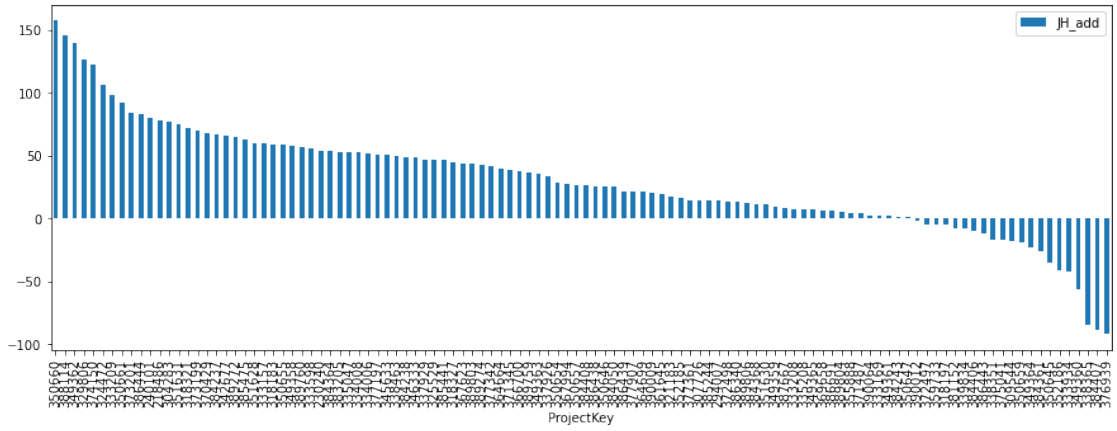
Tracez un *barchart* présentant le nombre de JH ajoutés (MDUpdatedPlanned - MDPlanned) par projet (ProjectKey).

Pour cela, vous devrez calculer l'ajout de JH pour chaque tâche puis vous pourrez réaliser un `.groupby()` et appliquer la méthode d'agrégation appropriée.

Vous pouvez utiliser la fonction`bar` de matplotlib ou appliquer directement `.plot.bar()`à votre DataFrame "groupé" pour réaliser ce graphique.

[27]:
```
# votre code ci-dessous

taches1 = taches.groupby('ProjectKey')
taches['JH_add']= taches['MDUpdatedPlanned'].sub(taches['MDPlanned'], axis=0)
taches.sort_values(by=['JH_add'], inplace=True, ascending=False)
```

```
#d = taches.groupby('ProjectKey')
#for ID, group in d:
#    print("\n ID :", ID)
#    print("")
#    print(group)
#    break

taches['JH_add']= taches['MDUpdatedPlanned'].sub(taches['MDPlanned'], axis=0)
#taches.pivot_table(index = ['JH_add'], aggfunc = 'size')
JH = taches.groupby(by=['ProjectKey']).sum()
JH.sort_values(by=['JH_add'], inplace=True, ascending=False)
JH
JH.plot(y='JH_add', kind='bar')
plt.show()

#JH
```



Commentaires : Les projets demandent plus régulièrement des ajouts que des retraits de JH.

1.5 Les logs

Chargez simplement les données des logs que vous avez enregistré à la fin de la partie 1. Nous ne ferons pas de visualisations supplémentaire dans cette partie.

[5]:
```
# votre code ci-dessous
logs = pd.read_excel('./03 - Data prep/logs_prep.xlsx')
logs.head()
```

[5]:

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

2 Story telling

Le but de cette partie est de construire une réflexion basée sur nos visualisations et de pouvoir la raconter.

Dans cette partie, suivez les instructions en commentaire pour construire les visualisations nécessaires à notre histoire.

Dans cette partie, nous allons croiser les données pour analyser des données "enrichies" !! Nous allons donc réaliser des jointures.

2.1 Le suivi des projets est désastreux...

Q0 - Réalisez la jointure des bases projets et logs. Nous souhaitons observer le nombre de logs par projets. Pour faire cela, il faudra (i) transformer la base des logs pour avoir le nombre de logs par ProjectKey puis (ii) réaliser une jointure entre la base des projets et la base des logs "groupée". **Important : On souhaite garder l'ensemble des projets.**

Il faudra a minima retrouver les colonnes suivantes : - Statut (projets) - DateCreation (projets) - ReferenceID/ProjectKey (projets et logs) - DevisBudgetJH (projets) - Nombre de logs (aggrégation des logs)

[43]:
```
# votre code ci-dessous

logs2 = logs.groupby(by = ['ProjectKey']).size().reset_index(name='counts')
logs2
projets

logs2 = logs2.rename(columns={'ProjectKey': 'ReferenceID'})
logs2
#projets_new = projets.rename(columns = {'ReferenceID':'ProjectKey'}, inplace =
↪True)
jointure = pd.merge(projets, logs2, on = ['ReferenceID'], how = 'left').fillna(0)

jointure.head()
```

[43]:

	Unnamed: 0	Annee	NumeroVersion	FicheDeDemande	ReferenceID	\
0	0	2014	1	x	218486	
1	1	2015	1	+	230240	
2	2	2015	4	x	240101	
3	3	2016	3	x	294007	
4	4	2017	1	+	304283	

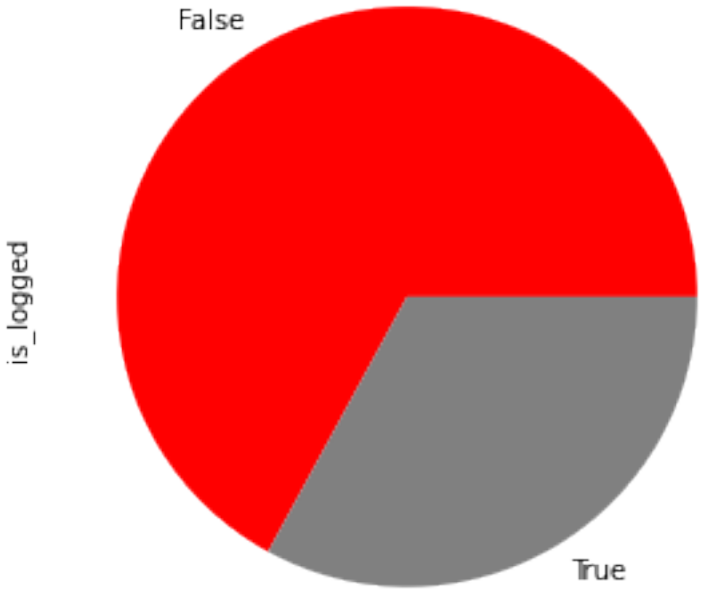
	NomDuProjet	DateCreation	Statut	\
0	Projet Girl Scout	2014-04-12	RAS	
1	Projet Cobra	2015-05-04	Termine	
2	Projet Bee - LOT 4	2015-03-27	En attente de decision	
3	Projet Templer	2016-05-28	GO	
4	Projet Alpha	2017-01-11	GO	

```
  InterlocuteurMetier   ChefDeProjet  ... DateDebutHomologation DevisBudgetJH  \
0      Joseph DICKENS     Jane SHARP  ...            2015-08-28          1900
1     Madeleine BROWN     Gavin KING  ...            2018-01-28          2000
2       Leonard JONES   Ian HEMMINGS  ...            2018-04-05          1400
3     Madeleine BROWN  David JACKSON  ...            2017-10-01          2200
4     Madeleine BROWN  David JACKSON  ...            2017-11-19          1500

  DevisBudgetEUR DevisBudgetJHEUR DevisBudgetInfraEUR Appli1  Appli2  \
0       2,6 MEUR          1235000             1365000  A4701   A7843
1       2,7 MEUR          1300000             1400000  A3758   A3774
2         2 MEUR           910000             1090000  A4452   A5146
3         3 MEUR          1430000             1570000  A4317   A8067
4       2,2 MEUR           975000             1225000  A3774   A5876

  DevisBudgetEUR_clean DevisBudgetJH_Rank  counts
0              2600000              128.5  2531.0
1              2700000              137.0  3093.0
2              2000000               90.0  2596.0
3              3000000              154.0  3213.0
4              2200000               98.5  3706.0

[5 rows x 28 columns]
```

Q1 - Tracez un *piechart* présentant la quantité de projets ayant effectivement des logs. Pour cela, il faudra créer une nouvelle colonne sur la base des projets et des logs. Elle contiendra True si un nombre de logs est renseigné et False sinon. Puis vous pourrez utiliser la méthode `.value_counts()` ou réaliser un `.groupby()`.

Vous pouvez utiliser la fonction `pie` de matplotlib ou appliquer `.plot.pie()` pour réaliser ce graphique.

[44]:
```
# votre code ci-dessous

jointure['islog']=jointure['counts'].astype({'counts': bool})
#jointure['islog'].value_counts()[0]
#jointure['islog'].value_counts()
#jointure['islog'].unique()[0]

data = [jointure['islog'].value_counts()[0], jointure['islog'].value_counts()[1]]
state = ['False', 'True']

plt.pie(data, labels = state, colors = ['red', 'grey'])
plt.axes().set_ylabel('is_logged')
plt.show()
```



Beaucoup de projets (presque 2/3) n'ont pas de JH logués. Cela pourrait être dû, par exemple, à leur statuts (typiquement ils n'est pas anormal de n'avoir aucun JH loggué sur un projet abandonné). Cependant ces 2/3 de projets n'ayant aucun de logs paraît vraiement élevé...

Q2 - Tracer un *barchart* du taux de projets (en pourcentage) ayant des JH loggués par statut des projets. Il faudra donc pouvoir créer un *stacked bar chart* mais il faudra aussi que les données soient sous le bon format ! Il faudra donc créer un DataFrame groupé reflétant ce "taux de projets ayant des JH logués par statut des projets". Pour faire cela, vous pouvez suivre les instructions de ce post : https://stackoverflow.com/questions/65233123/adding-percentage-of-count-to-a-stacked-bar-chart-in-plotly

[45]:
```
# votre code ci-dessous

jointure_1 = jointure.groupby(['Statut', 'islog']).size().reset_index()


jointure_1['percentage'] = jointure.groupby(['Statut', 'islog']).size().
↪groupby(level=0).apply(lambda x: 100 * x / float(x.sum())).values


jointure_1.columns = ['Statut', 'islog', 'Counts', 'percentage']

figur = px.bar(jointure_1, x='Statut', y=['percentage'], color='islog',␣
↪text=jointure_1['percentage'].apply(lambda x:'{0:1.2f}%'.format(x)))
figur.show()
```

Les projets avec aucun JH loggué sont présents quelque soit le statut. Pour les projets "NOGO", c'est logique, les projets n'ont pas débuté. Pour les projets "Abandonnés", "Gele" ou "En attente de décisions" il est possible que certains projets n'aient pas réellement démarré non plus. Cependant, pour les projets "GO", "GO sans devis" et surtout "Terminés" cela est tout à fait anormal. Nous sommes donc face à un problème de qualité de données. Il sera nécessaire que l'entreprise fasse l'effort de logguer correctement les tâches de ses collaborateurs pour assurer un suivi plus efficace des projets.

Q3 - Tracer un *barchart* du taux de projets ayant des JH loggués par année de création des projets. Même logique que sur le graphique précédent.

[46]:
```
# votre code ci-dessous
jointure_2 = jointure.groupby(['Annee', 'islog']).size().reset_index()

jointure_2['percentage'] = jointure.groupby(['Annee', 'islog']).size().
 ↪groupby(level=0).apply(lambda x: 100 * x / float(x.sum())).values

jointure_2.columns = ['Annee', 'islog', 'Counts', 'percentage']

figur1 = px.bar(jointure_2, x='Annee', y=['percentage'], color='islog',␣
 ↪text=jointure_2['percentage'].apply(lambda x:'{0:1.2f}%'.format(x)))
figur1.show()
```

Ce phénomène semble s'aggraver avec le temps. Aucun log n'a été recensé en 2018 !

2.2 Et le budget temps réel est rarement en ligne avec le devis

Q0 - Réalisez la jointure des bases logs et tâches Nous voulons comparer le nombre de JH logués et le nombre de JH planifiés. Pour cela il faudra (i) transformer la base des logs pour avoir le nombre de logs par ProjectKey (comme pour 2.1), (ii) transformer la base des tâches pour pouvoir observer le nombre de JH planifiés (MDPlannned) par projets (ProjectKey) et enfin (iii) réaliser une jointure des deux bases. **Important : On souhaite garder l'ensemble des éléments de la base des tâches et de la base des logs.**

Il faudra a minima retrouver les colonnes suivantes : - MDPlanned aggrégé (agrégation des tâches) - Nombre de logs (agrégation des logs) - ProjectKey (tâches et logs)

[68]:
```
# votre code ci-dessous

logsGB = logs.groupby(['ProjectKey']).sum()
#logsGB.columns = ['Counts']
logsGB = logsGB.rename(columns={"MDLogged": "counts"})

tachesGB = taches.groupby(['ProjectKey']).sum().reset_index()

#tachesGB.head()

jointure1 = pd.merge(logsGB, tachesGB, on = ['ProjectKey'], how = 'left').
 ↪fillna(0)
```

```
jointure1 = jointure1.drop(columns=["Unnamed: 0_x","Unnamed: 0_y"])

jointure1.head()
```

[68]:

	ProjectKey	counts	MDPlanned	MDUpdatedPlanned	JH_add
0	218486	2531	1901	1979	78
1	230240	3093	2403	2457	54
2	240101	2596	2000	2080	80
3	294007	3213	2500	2514	14
4	304283	3706	2804	2881	77

Q1 - Calcul du dérapage budgétaire Le dérapage budgétaire est défini comme l'erreur entre le nombre de JH logués et le nombre de JH planifiés (en pourcentage).

Donc $ Dérapage = 100*MD$_{logged} - MD_{Planned}$ $\overline{MD_{Planned}\$}$

Créez une nouvelle colonne contenant le dérapage budgétaire pour chaque projet.

[69]:
```
# votre code ci-dessous

jointure1['DerBudg'] = 100*((jointure1['counts']-jointure1['MDPlanned'])/
 ↪jointure1['MDPlanned'])

jointure1.head()
```

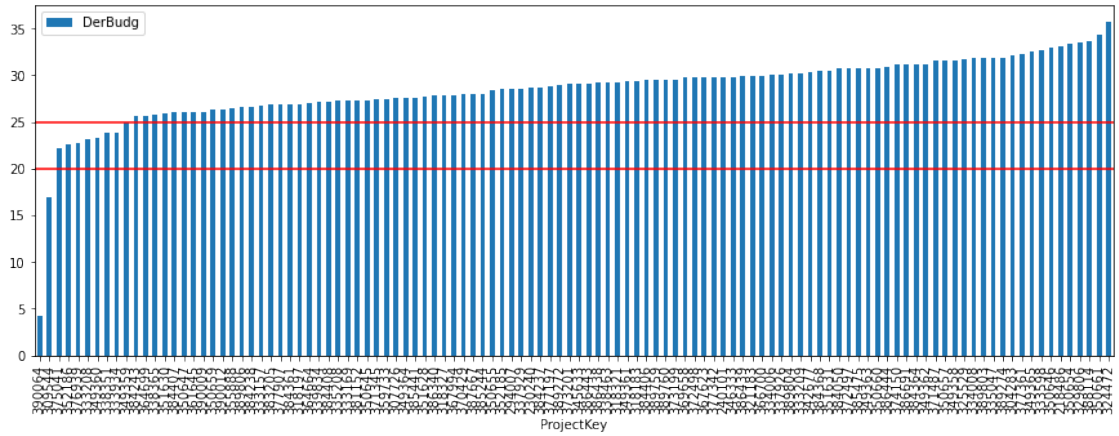
[69]:

	ProjectKey	counts	MDPlanned	MDUpdatedPlanned	JH_add	DerBudg
0	218486	2531	1901	1979	78	33.140452
1	230240	3093	2403	2457	54	28.714107
2	240101	2596	2000	2080	80	29.800000
3	294007	3213	2500	2514	14	28.520000
4	304283	3706	2804	2881	77	32.168331

Q2 - Tracez un *barchart* permettant d'observer le dérapage budgétaire par projet Vous pourrez ajouter une ligne à 20% et 25% de dérapage budgétaire pour faire ressortir ces niveaux sur le graphique.

[70]:
```
# votre code ci-dessous
jointure1.sort_values(by=['DerBudg'], inplace=True, ascending=True)

fig123 = jointure1.plot.bar(x='ProjectKey', y='DerBudg')
plt.axhline(y=20, color='red')
plt.axhline(y=25, color='red')
```

[70]:
```
<matplotlib.lines.Line2D at 0x1f3c93ef970>
```



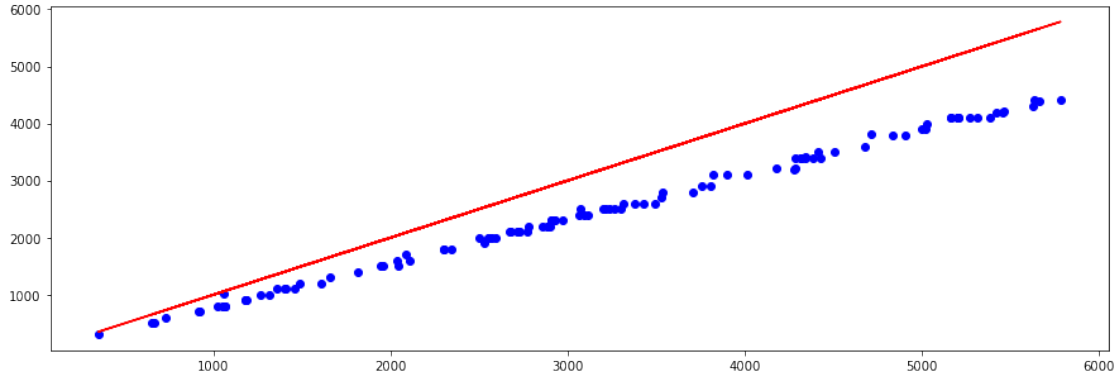
Il est normal d'observer quelques cas de dérapage budgétaire mais ici, il s'agit de la totalité des projets ! Pire, moins de 2% des projets (2 sur les 112 analysés) dépassent le budget de plus de 20% (ligne noire sur le graphique) !!

Il est nécessaire de revoir la manière dont les budgets sont estimés. Un méthode simple pourrait être d'ajouter systématiquement 25% (ligne rouge sur le graphique) de budget temps sur l'ensemble des projets.

Q3 - Tracez un *scatterplot* représentnant le nombre de JH planifiés en fonction du nombre de JH Loggués Vous pourrez également ajouter une ligne d'équation $x = y$

[71]:
```
# votre code ci-dessous
x_1 = jointure1['counts']
y_1 = jointure1['MDPlanned']
plt.scatter(x = x_1, y = y_1, c='blue')
plt.plot(x_1, x_1, color='red')
plt.show
```

[71]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Les JH planifiés et logués sont très corrélés... Cependant, à un JH planifié est souvent associé plus d'un JH logué.

Il est possible de réaliser une régression pour obtenir un coefficient qui permettrait de mieux appréhender cette différence entre les JH logués et les JH planifiés.

Rien à faire ici, je vous donne le code (vous reverrez cela au cours de *Machine Learning* que vous aurez prochainement).

Nous allons faire simple, nous ignorerons l'*intercept* (il est en plus faible, ici) et n'allons récupérer que le coefficient de la pente.

Nous pouvons faire cela très simplement avec le package `scipy`.

[74]:
```
from scipy import stats
slope, _, _, _, _ = stats.linregress(jointure1['MDPlanned'], jointure1['counts'])

print(slope)
```

```
1.2832076304762186
```

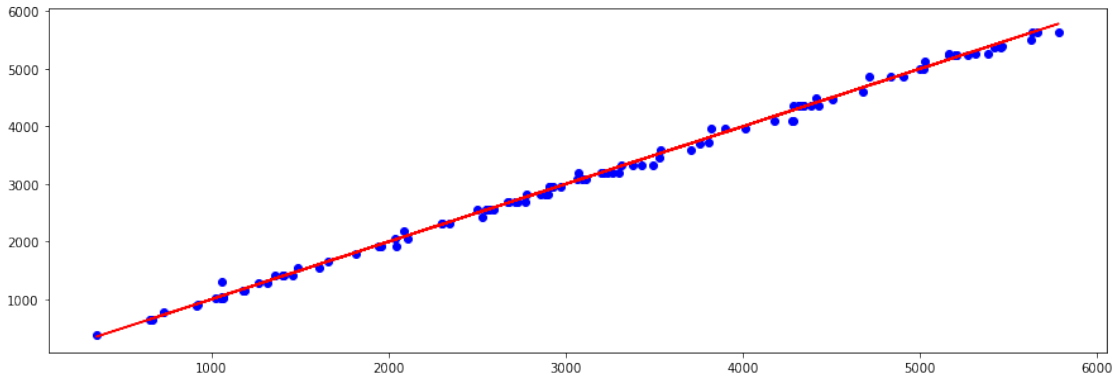
Nous obtenons une pente de 1.28. Cela signifie qu'il est possible de corriger nos JH planifiés de cette valeur. Nous devrions obtenir $MD_{logged} \approx 1.28 * MD_{planned}$.

NB : L'intercept est ignoré ici. Vous pouvez jeter un oeil sur le projet "régression" de l'an dernier pour plus de détails.

Q4 - Retracez le graphique de la question précédente mais cette fois-ci en corrigeant les JH planifiés

[75]:
```
# votre code ci-dessous
jointure1['MDPlannedCorr'] = 1.28*jointure1['MDPlanned']
x_11 = jointure1['counts']
y_11 = jointure1['MDPlannedCorr']
plt.scatter(x = x_11, y = y_11, c='blue')
plt.plot(x_1, x_1, color='red')
plt.show
```

[75]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Une régression linéaire permet de montrer que $MDLog \approx 1.28 * MDPlanned$ (en ignorant l'intercept). Cela est cohérent avec la proposition d'ajouter systématiquement 25% de budget temps sur l'ensemble des projets. Ici, la régression montre que la planification des tâches est sous-estimée et qu'il faudrait ajouter 28% de budget temps pour être en ligne avec les observations des logs.