Intro

L'objectif du devoir est comme indiqué dans les consignes du projet "de créer un plan de raccordement qui priorise les bâtiments les plus simples à raccorder (en minimisant les coûts) tout en maximisant le nombre de prises raccordées", le dataset est composé de 5 colonnes, avec une colonne id bâtiment qui comme son nom l'indique donne un identifiant unique à un bâtiment,un bâtiment à une propriété qui est son nombre de maisons, en effet un id_batiment peut avoir plusieurs maisons (la colonne nb_maisons). Les bâtiments sont eux-mêmes reliés à des infrastructures, et les infrastructures sont identifiés par leur identifiant (la colonne id_infra) , les infrastructures peuvent reliés plusieurs bâtiments, et les infrastructures ont plusieurs propriétés, tout d'abord l'état de l'infra qui a un état 'infra_intacte' ou 'a_remplacer'. Si l'état de l'infra est de type intacte alors pas besoin de modifier celle-ci. Cependant si elle est de type a remplacer alors elle a besoin de modification.

I.Data exploration

Nous pouvons nous poser des questions sur le dataset comme combien y a-t-il de lignes,y a-t-il des valeurs nulles, ou bien la proportion des types d'infrastructure. Tout d'abord le dataset se compose de 6107 lignes,avec la ligne de code 'df.isnull().sum()' cela nous permet de voir le nombre de valeurs nulles dans le dataset.Dans ce dataset nous n'avons pas de valeurs nulles.Cependant en ayant vérifier avec la commande 'len(df[df.duplicated])' on remarque que 521 lignes sont des lignes dupliqué et n'ont donc aucun intérêt. On va alors les supprimer avec la ligne 'self.new_df.drop_duplicates(inplace=True)' on va supprimer toutes les lignes dupliquées nous arrivons donc à 5586 lignes propres. Maintenant que le dataset est propre nous pouvons vérifier la proportion de lignes par type d'infra.

infra_type	id_batiment
a_remplacer	9.613319
infra_intacte	90.386681

Environ 90 % des infrastructures sont intactes (5049 lignes) donc pas nécessairement utiles à l'analyse, et environ 10% sont à remplacer (537 lignes).
Maintenant que nous savons toutes ces informations sur le dataset nous pouvons nous intéresser au développement de la métrique de priorisation.

II.Métrique de priorisation

Le but de notre métrique de priorisation sera de rendre optimal nos réparations , optimal dans le sens ou l'algorithme doit prioriser les infra ou la réparation a un coût minimum dans lequel le coût de l'infrastructure est égale à la longueur/nb_maisons. Dans mon code il y a 3 files , la premiere main.py qui fait la liaison entre les deux autres files, la première la file infrastructures

dans laquelle cette classe va nous servir à avoir le niveau de difficulté d'une infrastructure.La première étape est de grouper par id d'infrastructure et sommer le nombre de maisons on aura par exemple {'infra_id':'x','nb_maisons':4},ensuite on filtre le dataset par "a_remplacer" et créer une colonne qui modélise la fonction coût: longueurs/nb_maisons.
L'autre classe Batiment va nous permettre de faire l'algorithme de priorisation. Tout d'abord il faut avoir la difficulté bâtiment qui est égale à la somme des difficulté infrastructures. Alors tout d'abord on va créer 2 listes , la première liste est la liste de bâtiment et la deuxième liste est la liste de valeurs des difficulté bâtiment.

```
for _ in range(0,len(self.building_dificulty)):
        minimum_value = min(self.building_dificulty)
        index_min_value = self.building_dificulty.index(minimum_value)



        self.list_e.append(self.id_batiment_to_fix_list[index_min_value])
        self.id_batiment_to_fix_list.remove(self.id_batiment_to_fix_list[index_min_value])
        self.building_dificulty.remove(minimum_value)

        minimum_value = None
        #print(self.id_batiment_to_fix_list,self.building_dificulty)
        self.see_index()
```

Le but est que a chaque itération on prend la valeur minimum de la liste des difficulté bâtiment, ensuite on prend son indexe pour avoir son id batiment associé. Cette id bâtiment associé est ajouté dans une liste , on supprime des listes bâtiments l'id bâtiment à réparer et sa valeur associé car elle n'aura plus besoin d'être réparée. Ensuite une fois avoir fait ça on appel une instance de classe see_index.

```
 for i in range(self.counter,len(self.list_e)):

        self.counter +=1
            self.df = self.dataset_with_difficulty.loc[(self.dataset_with_difficulty['id_batiment'] ==
self.list_e[i]),:]
        self.df = self.df.loc[self.df['infra_type']=='a_remplacer']
        if len(self.df)>0 :
           self.list_infra_to_fix = self.df['infra_id'].to_list()
    self.change_state_infra()
    self.data = {'id_batiment':self.list_e,'difficulty_infra':self.val}


    self.list_infra_to_fix = []
```

Pourquoi range(self.counter,len(self.list_e))?
Au départ la len de list_e est de est de 1 donc, range(self.counter,len(self.list_e)) = (0,1) mais si len(list_e)=20 si on fait de (0,len(self.list_e)) on va faire des itération qui ne servent a rien, car quand la len sera de 20 on aura (self.counter,len(list_e)) = (19,20) (toujours 1 iterations), mais si (0,len(self.list_e)) on aura (0,20) donc une repetition de 19 itération inutiles. Comme la max(len(list_e)) = 85 on aura 3655 itération si (0,len(list_e)), contre 85 avec (counter,len(list_e)).

self.df=self.dataset_with_difficulty.loc[(self.dataset_with_difficulty['id_batiment']==self.list_e[i]),:]
Cette ligne nous permet de localiser l'id bâtiment à réparer cette ligne va filtrer et permettre de voir que les id batiment ou il y a cette id (comme itération de 1 il n'y a qu'un id batiment contre si (0,20) par exemple il y aura 20 id bâtiment à chercher un suite suite),
Output:

	Id_batiment	nb_maisons	infra_id	infra_type	longueur
0	E000001	4	P007111	infra_intacte	12.314461
1	E000001	4	P007983	infra_intacte	40.320929
2	E000001	4	P000308	infra_intacte	39.140799
3	E000001	4	P007819	infra_intacte	17.390464

self.list_infra_to_fix = self.df['infra_id'].to_list()
Cette ligne permet de convertir en list les infra id associé a l'id batiment.

```
def change_state_infra(self):

        for infra_id in self.list_infra_to_fix:

                self.dataset_with_difficulty.loc[self.dataset_with_difficulty['infra_id']
== infra_id, 'infra_type'] = 'infra_intacte'
                self.a.append(i)



#print(self.dataset_with_difficulty.loc[self.dataset_with_difficulty['infra_type']=='a
_remplacer',:])
        self.dataset_with_difficulty.to_csv('dataset.csv')
```

Il faut que quand on change son état elle se change dans tout les lignes ou elle se trouve par exemple le loc va avoir un output:

	id_batiment	nb_maisons	infra_id	infra_type	longueur
5986	E000377	1	P008035	a_remplacer	11.719759
6015	E000378	1	P008035	a_remplacer	11.719759

Dans ce cas, on change l'état dans les lignes 5986 et 6015 car cette infra est reliée à différents bâtiments

Le but de see_order est de voir l'efficacité de l'algorithme et a-t-on atteint l'objectif de d'avoir maximiser les prises raccordées avec une minimisation des coûts.
Pour vérifier cela nous pouvons diviser par 5 le dataset, et voir si dans un premier temps le nombre de maisons raccordées est maximisé et le coût minimiser et voici le résultat.

Chaque ligne représente un set du dataset.

Diff longueur nb_maisons
5.78 | 86.33872569486178 | 269.0
8.65 | 83.60912239417699 | 258.0
11.52 | 122.85679014462332 | 373.0
15.56 | 113.87261311087643 | 379.0
22.68 | 78.56963603101137 | 337.0

Comment expliquer cela ?
Cela s'explique par le fait que la difficulté infra (longueur/nombre maisons) favorise les infrastructures ou il y a des longueurs faibles, en général plus la longueur est faible moins il y aura de maison .Donc par défaut la difficulté infra sera plus faible, pour cela que dans les premier set la difficulté est la plus faible mais la longueur d'infra réparée est moins élevé et le nombres de maisons moins élevés. Le nombre de maisons réparées n'est pas le plus élevé car en général la difficulté bâtiment est quand même plus élevée pour les infra longues, et de fait si la longueur est plus élevée la valeur coût sera aussi plus élevée.

Conclusion

L'algorithme est optimisé en termes de coût, c'est-à-dire que l'algorithme a réparé de la plus petite difficulté infra a la plus ce qui a permis d'avoir le moins de coût possible. Cependant en constatant le tableau au dessus on remarque que le nombre de maisons réparées n'est pas au maximum et la longueur non plus. Ce qui faudrait faire pour encore plus optimisé il faudrait mettre une constante bonus/malus dans la fonction, ou coût =(longueur/nb maisons)-a , ou a est une constante qui diminue la difficulté infra si le nombre de maison est élevé pour favoriser la mutualisation des réparation, comme ca la valeur coût sera moins élevé et donc sera réparée plus tot.