DS 3 ALGORITHME des k plus proches voisins

Problème : Algorithme des k plus proches voisins

On se propose de mettre au point un algorithme qui pourrait être utilisé dans une agence immobilière pour aider un propriétaire à estimer le prix de son bien immobilier. On utilisera une base de données qui contient, pour les différentes ventes immobilières réalisées dans l'année précédente, les coordonnées Gps des biens déjà vendus et le prix au m² qui a été utilisé pour réaliser la transaction.

Pour simplifier l'étude, cette base de données sera ici, une liste nommée « *liste_transactions* » .

Cette liste retournée par l'appel de la fonction ci-contre, contient elle-même 8 listes qui contiennent chacune :
[x , y , P , *None*] avec :

- x = abscisse du bien déjà vendu
- y = ordonnée
- P = prix au m² et en €
- champ vide pour l'instant (**None**)

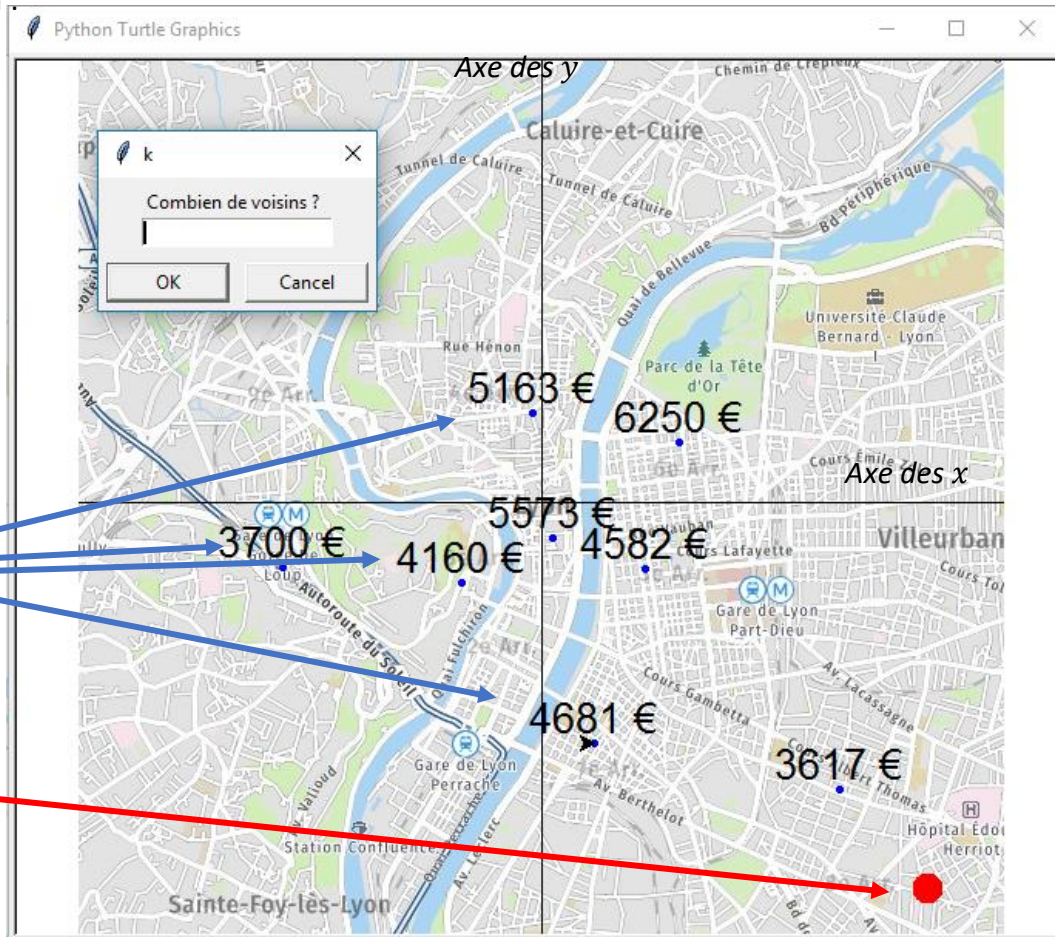
```
def bdd_biens_deja_vendus() :
    """
        Retourne la liste contenant les coordonnées, la valeur
        et un champ encore vide(none)
    """
    liste  =  [ [-52 , -52 , 4160 , None],\
                [-168 ,-42 , 3700 , None],\
                [-6 ,58 , 5163 , None],\
                [7 ,-23 , 5573 , None],\
                [89 ,39 , 6250 , None],\
                [67 ,-43 , 4582 , None],\
                [193 ,-186 , 3617 , None],\
                [34 ,-156 , 4681 , None] ]

    return liste
```

`liste_transactions = bdd_biens_deja_vendus()` : appel ce cette fonction dans le *Main.*

Les coordonnées données sont celles, en pixels, dans une fenêtre graphique du module *Turtle* de python. L'emplacement du bien immobilier **que le propriétaire veut estimer** est contenu dans une liste nommée « *emplacement_bien* » contenant les coordonnées [x , y] de ce bien. On prendra dans ce DS :

```
emplacement_bien = [250,-250]
```

Une fonction `def visualiser(liste,point,k,moy)` déjà écrite, permet de visualiser tous ces éléments sur la carte de la ville de Lyon :



biens référencés dans liste_transactions

(biens déjà vendus)

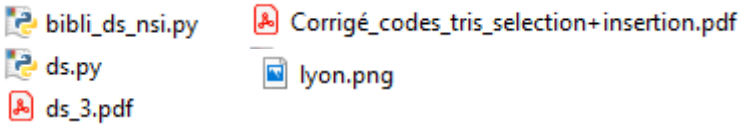
emplacement_bien

(bien dont on veut estimer le prix au m²)

DS 3 ALGORITHME des k plus proches voisins

PARTIE A : Prise en main et démarrage

⇨ **Copier** le dossier *NSI-16oct2020* : ***Examens(Z :) /exam01/sujets/NSI-16oct2020*** dans **votre** zone de travail. Vous y trouvez 5 fichiers :



⇨ Ouvrez les 2 fichiers pythons dans pyzo.

Le fichier bibli_ds_nsi.py contient les fonctions `def bdd_biens_deja_vendus() :` et `def visualiser(liste,point,k,moy)` évoquées précédemment. **Ce fichier ne doit pas être modifié**. Afin de ne pas surcharger le code, ces 2 fonctions ont été séparées de celles que vous allez écrire vous-même dans la suite.

Le fichier ds.py est celui que vous allez compléter. Son contenu est actuellement le suivant :

```
1  from turtle import *
2  from math import sqrt
3  from bibli_ds_nsi import visualiser,bdd_biens_deja_vendus
4
5  # ------- Fonctions ------------
6  def distance(xa,ya,xb,yb) :
7      """
8          Renvoie la distance entre les points de coordonnées (xa,ya)
9          et (xb,yb)
10     """
11     return
12
13 def complete_distance(liste,point) :
14     """
15         Renvoie la liste mise en argument. Le champs liste[i][3] prend la valeur
16         de la distance d entre point et l'emplacement du bien de liste[i]
17     """
18     return
19
20 def tri(liste):
21     """
22         Renvoie liste triée par rapport à la valeur de la distance contenues
23         dans liste[i][3]. On utilise un algorithme de tri par sélection
24     """
25     return
26
27 def k_voisins(k,liste) :
28     """
29         Renvoie la moyenne des k premieres valeurs V contenues
30         dans liste
31     """
32     return
33
34 # -------------- MAIN ----------------------
35 liste_transactions = bdd_biens_deja_vendus()
36 emplacement_bien = [250,-250]
37 k = visualiser(liste_transactions,emplacement_bien,None,None)
38
39 print(f"Valeur saisie pour k : {k} ")
40 prix_moyen = 0
41
42 visualiser(liste_transactions,emplacement_bien,k,prix_moyen)
43
44 # ------------ Pour pouvoir fermer turtle (ne pas enlever) -----------
45 exitonclick()
```

4 fonctions que vous allez devoir compléter dans ce DS

L'importation des fonctions du fichier *bibli_ds_nsi.py*

La partie programme principal que vous allez compléter dans ce DS

DS 3 ALGORITHME des k plus proches voisins

⇨ Dans pyzo, aller dans le fichier ds.py et l'exécuter dans le shell (F5). Dans la fenêtre graphique *Turtle* qui apparait, saisir un entier compris entre 1 et 8 dans la fenêtre de dialogue. Le programme s'arrête pour l'instant et affiche dans le shell (ici pour une saisi du nombre 2) :

```
>>> (executing file "ds.py")
Valeur saisie pour k : 2
```

Pour l'instant le programme principal comprend :

- ligne **35** : exécution de la fonction déjà évoquée qui retourne la liste *liste_transactions* qui contiendra :

```
[ [-52 , -52 , 4160 , None],\
  [-168 ,-42 , 3700 , None],\
  [-6 ,58 , 5163 , None],\
  [7 ,-23 , 5573 , None],\
  [89 ,39 , 6250 , None],\
  [67 ,-43 , 4582 , None],\
  [193 ,-186 , 3617 , None],\
  [34 ,-156 , 4681 , None] ]
```

- ligne **36** : coordonnées du bien à estimer : [250 , -250]
- ligne **37** : exécution de la fonction déjà évoquée qui permet l'affichage de la fenêtre graphique et qui retourne la valeur de la variable k saisie dans la fenêtre de dialogue.
- ligne **42** : exécution de la même fonction pour souligner en rouge sur la carte les « *k plus proches voisins* » et y afficher le prix au m² du bien à estimer, qui sera calculé par ce code.

```
34 # -------------- MAIN ----------------------
35 liste_transactions = bdd_biens_deja_vendus()
36 emplacement_bien = [250,-250]
37 k = visualiser(liste_transactions,emplacement_bien,None,None)
38
39 print(f"Valeur saisie pour k : {k} ")
40 prix_moyen = 0
41
42 visualiser(liste_transactions,emplacement_bien,k,prix_moyen)
```

Seule la zone encadrée ci-dessus sera à modifier pour la partie programme principal.

OBJECTIFS DE CE TRAVAIL :

L'objectif du travail que vous allez réaliser sera de compléter ce code en suivant les étapes suivantes :

- 1ère étape : Pour chacun des biens de la liste *liste_transactions*, utiliser les coordonnées x et y pour calculer la distance géométrique qui sépare ce bien de la position de celui que l'on souhaite estimer (*emplacement_bien*).
- 2nde étape : Rechercher dans *liste_transactions* , les k biens les plus proches de *emplacement_bien*.
- 3ième étape : Pour ces k plus proches voisins, calculer la moyenne de leur prix au m² . Cette moyenne sera celle du bien que l'on veut estimer. L'exécution se termine alors.

On appelle cet algorithme celui des k plus proches voisins. En modifiant la valeur de k, le prix du bien à estimer change, puisqu'il est calculé par rapport à davantage de biens situés dans le voisinage. Cet algorithme est simple. Il est néanmoins très efficace si la base de données à disposition est importante, ce qui n'est pas le cas ici dans ce DS.

DS 3 ALGORITHME des k plus proches voisins

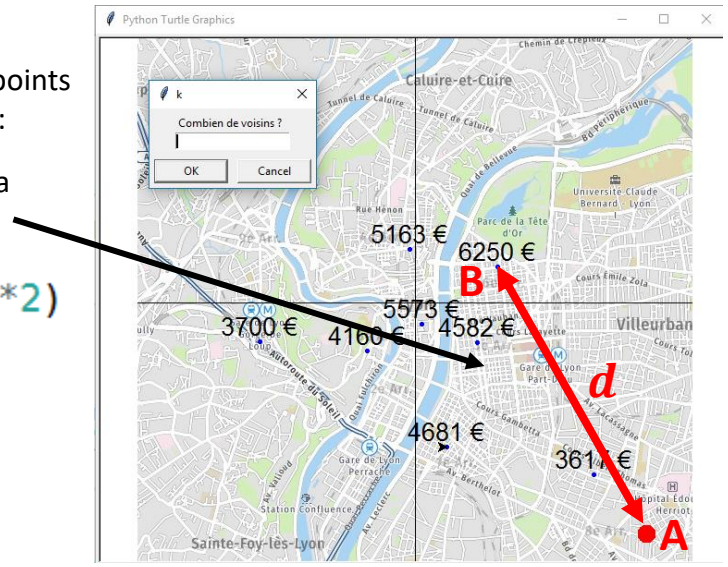
TRAVAIL A FAIRE :

PARTIE B : 1ère étape : « Pour chacun des biens de la liste *liste_transactions*, utiliser les coordonnées x et y pour calculer la distance géométrique qui le sépare de la position de *emplacement_bien* »

La distance géométrique $d = AB$ entre 2 points $A(x_A\ , y_A)$ et $B(x_B\ , y_B)$ est donnée par la relation :

$d = \sqrt{(x_B - x_A\ )^2 + (y_B - y_A\ )^2}$. En python, cela donne :

```
d = sqrt((xb-xa)**2 + (yb-ya)**2)
```



1- Compléter la fonction *distance()* qui prend en arguments les coordonnées de $A(x_A\ , y_A)$ et $B(x_B\ , y_B)$ et **renvoie la valeur de d .**

```
def distance(xa,ya,xb,yb) :
    """
        Renvoie la distance entre les points de coordonnées (xa,ya)
         et (xb,yb)
    """
    d = sqrt((xb-xa)**2 + (yb-ya)**2)
    return d
```

2- Compléter la fonction *complete_distance()* qui prend en argument la liste *liste_transactions* et la liste *emplacement_bien* et renvoie la même liste *liste_transactions* en ayant remplacé pour chaque élément i , le ***None*** de *liste_transactions[i] par la distance d qui sépare ce bien de celui à estimer.*

```
def complete_distance(liste,point) :
    """
        Renvoie la liste mise en argument. Le champs liste[i][3] prend la valeur
        de la distance d entre point et l'emplacement du bien de liste[i]
    """
    xa , ya = point
    for i in range(len(liste)) :
        x = liste[i][0]
        y = liste[i][1]
        d = distance(xa,ya,x,y)
        liste[i][3] = d
    return liste
```

3- Appeler la fonction *complete_distance()* dans le programme principal afin que *liste_transactions* soit modifié et complété.

```
53 # --------------- MAIN ----------------------
54 liste_transactions = bdd_biens_deja_vendus()
55 emplacement_bien = [250,-250]
56 k = visualiser(liste_transactions,emplacement_bien,None,None)
57
58 liste_transactions = complete_distance(liste_transactions,emplacement_bien)
```

PARTIE C : 2nde étape : « Rechercher dans *liste_transactions* les k plus proches voisins de *emplacement_bien* ».

4- Compléter la fonction *tri()* qui prend en argument *liste_transactions* et tri ses éléments *liste_transactions[i]* par ordre croissant de la valeur de la distance contenu dans *liste_transactions[i][3]* . Ne pas utiliser la fonction *sorted()* de python : réutiliser **en les adaptant à cette nouvelle situation**, les codes vus en TP 2 ou TP 3 (corrigé de ces codes disponibles dans le répertoire *NSI-16oct2020* copié en début de Tp).

liste_transactions_trie[0] sera parmi les 8 listes ci-contre, celle qui aura la valeur de la distance (qui remplace **None**) la plus petite.

```
[ [-52 , -52 , 4160 , None],\
  [-168 ,-42 , 3700 , None],\
  [-6 ,58 , 5163 , None],\
  [7 ,-23 , 5573 , None],\
  [89 ,39 , 6250 , None],\
  [67 ,-43 , 4582 , None],\
  [193 ,-186 , 3617 , None],\
  [34 ,-156 , 4681 , None] ]
```

Info utile : pour échanger le contenu de 2 variables a et b, on peut écrire en python : a , b = b , a
Si ces variables sont des listes de listes, on peut faire la même chose. Par exemple, si $\ell = [\,[a1, b1]\,,\, [a2, b2]\,]$
L'exécution de : $\ell[0]\,, \ell[1] = \ell[1]\,, \ell[0]$ permet d'obtenir : $\ell = [\,[a2, b2]\,,\, [a1, b1]\,]$
L'exécution de : $a = \ell[0]$, puis $\ell[0] = l[1]$ puis $\ell[1] = a$ permet d'obtenir aussi le même résultat.

```
29 def tri(liste):
30     """
31         Renvoie liste triée par rapport à la valeur de la distance contenues
32         dans liste[i][3]. On utilise un algorithme de tri par sélection
33     """
34     for i in range(len(liste)) :
35         indice = i
36         for j in range(i , len(liste)) :
37             if liste[j][3] < liste[indice][3] : indice = j
38         liste[i] , liste[indice] = liste[indice] , liste[i]
39     return liste
```

Appeler ensuite la fonction *tri()* dans le programme principal afin que la liste retournée et donc triée ait le nom : *liste_transactions_trie*

```
58 liste_transactions = complete_distance(liste_transactions,emplacement_bien)
59 liste_transactions_trie = tri(liste_transactions)
```

PARTIE D : 3ième étape : « Pour les k plus proches voisins, calculer la moyenne de leur prix au m² . Cette moyenne sera celle du bien que l'on veut estimer».

5- Compléter la fonction *k_voisins()* qui prend en argument la valeur de k saisie par l'utilisateur et la liste *liste_transactions_trie*. Cette fonction renvoie la moyenne de prix au m² des k premiers éléments de la liste *liste_transactions_trie*.

```
42  def k_voisins(k,liste) :
43      """
44          Renvoie la moyenne des k premieres valeurs V contenues
45          dans liste
46      """
47      s = 0
48      for i in range(k) :
49          s += liste[i][2]
50      return s / k
```

6- Appeler la fonction *k_voisins()* dans le programme principal, suivi de l'appel de la fonction *visualiser()* afin d'afficher sur la carte l'estimation du bien en € et de souligner les k voisins pris en compte :

```
prix_moyen = k_voisins(k,liste_transactions_trie)

visualiser(liste_transactions_trie,emplacement_bien,k,prix_moyen)
```

PARTIE E : Conclusion

Faire des exécutions pour déterminer le prix moyen au m² pour un bien à estimer dont l'emplacement et le nombre de voisins à prendre en compte est défini sur le tableau ci-dessous :

x	*250*	*250*	*250*	*125*	*125*	*- 250*	*-250*
y	*-250*	*-250*	*-250*	*200*	*200*	200	*-250*
k	$k = 1$	$k = 3$	$k = 7$	$k = 1$	$k = 3$	$k = 4$	$k = 2$
Prix estimé au m² en €	3617,0 €	4293,0 €	4861,0 €	6250,0 €	5332,0 €	4349,0 €	3930,0 €