P2_DIOUNOU_Bailly

September 8, 2021

1 Phases préliminaires

importation des librairies utiles

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import folium
import seaborn as sns
sns.set()
```

Chargement du fichier de données brutes

[2]:
```
arbres = pd.read_csv("p2-arbres-fr_test.csv",sep=";")
```

```
␣
↪---------------------------------------------------------------------------

FileNotFoundError                         Traceback (most recent call␣
↪last)

<ipython-input-2-a143f85fbb97> in <module>
----> 1 arbres = pd.read_csv("p2-arbres-fr_test.csv",sep=";")


~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪parser_f(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
↪usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters,␣
↪true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,␣
↪na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates,␣
↪infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates,␣
↪iterator, chunksize, compression, thousands, decimal, lineterminator,␣
↪quotechar, quoting, doublequote, escapechar, comment, encoding, dialect,␣
↪error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map,␣
↪float_precision)
    674         )
    675
```

```
   --> 676         return _read(filepath_or_buffer, kwds)
       677
       678     parser_f.__name__ = name


       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪_read(filepath_or_buffer, kwds)
       446
       447     # Create the parser.
   --> 448     parser = TextFileReader(fp_or_buf, **kwds)
       449
       450     if chunksize or iterator:


       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪__init__(self, f, engine, **kwds)
       878             self.options["has_index_names"] = kwds["has_index_names"]
       879
   --> 880         self._make_engine(self.engine)
       881
       882     def close(self):


       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪_make_engine(self, engine)
      1112     def _make_engine(self, engine="c"):
      1113         if engine == "c":
   -> 1114             self._engine = CParserWrapper(self.f, **self.options)
      1115         else:
      1116             if engine == "python":


       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪__init__(self, src, **kwds)
      1889         kwds["usecols"] = self.usecols
      1890
   -> 1891         self._reader = parsers.TextReader(src, **kwds)
      1892         self.unnamed_cols = self._reader.unnamed_cols
      1893


       pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()


       pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.
↪_setup_parser_source()
```

```
        FileNotFoundError: [Errno 2] File p2-arbres-fr_test.csv does not exist:␣
↪'p2-arbres-fr_test.csv'
```

2 Présentation générale du jeu de données

2.1 Présentation résumée du fichier brut de données

[ ]:
```
arbres.head()
```

2.2 Description statistique

[ ]:
```
arbres.describe().to_excel(r'description.xlsx', index = False)
arbres.describe()
```

[ ]:
```
arbres.columns;
```

[ ]:
```
arbres.nunique(); # sert à faire ce qui est fait dans la case d'après, mais en␣
↪une ligne. [de PAG]
```

[ ]:
```
plt.barh(arbres.nunique().index, width = arbres.nunique().values)

plt.semilogx()
plt.title('Nombre de modalités par caractéristique');
plt.xlabel('Log(Effectif)'); plt.ylabel('Caractéristiques');

plt.savefig("./figNbModal.png", bbox_inches='tight') # le secnd param permet␣
↪d'éviter un rognement de l'image en sortie
```

3 Démarche méthodologique d'analyse de données

3.1 Pré-traitement du fichier brute

3.1.1 Détermination du taux de remplissage

[ ]:
```
np.shape(arbres)[0]
na_level = 1-arbres.isna().sum()/np.shape(arbres)[0]
print(na_level[na_level.values!=1.000000])
```

[ ]:
```
plt.barh(na_level.index, width = na_level.values)
plt.title('Taux de remplissage des variables');
```

```
plt.xlabel('Taux en %'); plt.ylabel('Caractéristiques');

plt.savefig("./FillRateRaw.png", bbox_inches='tight') # le secnd param permet␣
 ↪d'éviter un rognement de l'image en sortie
```

Elimination des variables dont le taux de remplissage est en deça de la valeur 70%

[ ]:
```
arbres_clean = arbres
invalid_na_rate = na_level[na_level.values<=0.7]
arbres_clean = arbres_clean.drop(columns=list(invalid_na_rate.index))
```

3.1.2 Elimination des variables non-pertinentes

[ ]:
```
arbres_clean.columns
```

Eliminations déduites de l'observation des modalités

Sur la base de l'observation des modalités, faite plus haut dans la section "Présentation résumée…", il nous apparait pertinent de supprimer les colonnes:

'id': Sert essentiellement de clé d'identification unique pour chaque individu de l'échantillon; mais cette fonction peut-être accomplie par les indices implicites des lignes.

'type_emplacement': 1 modalité

'id_emplacement': Consisant en codes inintelligibles, cette variable **semble** faire office d'espace de clé d'identification pour la variable "lieu" par une relation surjective. On pourrait exploiter cette donnée pour voir la distribution.

'genre': Avec les variables 'libelle_francais' et (dans une moindre mesure) 'espece', ce sont des variables qui servent à désigner les arbres et de les distinguer par groupes. Le genre a un aspect nominal, de même que le libelle_francais; cependant le dernier cité est plus pertinent pour un public non-scientifique (1). Par ailleurs, en plus de l'aspect nominal et distinctif, l'espece a un aspect fonctionnel, puisque les modalités sont définies par le critère booléen de reproduction possible intra-espèce.

En particulier, en conservant la var 'espece', il pourrait être pertinent de déterminer *au préalable* la **corrélation entre les variables 'espece' et 'lieu'** pour évaluer la concentration relative de la même espèce. Si il y a une concentration géographîque suffisamment importante intragroupe, cela pourrait permettre d'établir des tournées conjointes intra-espèce et inter-cluster de la même espèce:

1. Qui et comment ? Association d'équipe d'entrtien avec équipe de polénisation artificielle (optimisation resources)
2. Quand ? Saison naturelle de pollénisation + conditions météorologiques favorables (vent)
3. Quoi ? Pollénisation artificielle + Transport de ruches d'abeilles et autres agents pollénisants

Ou au contraire, l'on pourrait contribuer à inhiber la production de fruit sur certaines artères pour lesquelles l'esthétique est privilégiée sur la fécondité, ou quand la fécondité pose des problème de salubrité (fruits pourris).

Par ailleurs, nous avons constaté a posteriori (lors des opérations d'imputation statistique, que la variable 'genre' peut servir à l'imputation stastique des variables 'espèce' et 'libelle_francais'. En effet, les corrélations statistiques et taxonomiques entre les deux dernières variables avec le genre permettent de déterminer leurs valeurs manquantes par le trûchement du dictionnaire les associant respectivement au genre correspondant. **L'élimination de la variable genre sera donc effectuée après la phase d'imputation statistique**.

[ ]:
```
arbres_clean = arbres_clean.
↪drop(columns=['id','type_emplacement','id_emplacement'])
```

[ ]:
```
arbres_clean.head()
```

[ ]:
```
np.shape(arbres_clean)
```

3.1.3 Elimination des valeurs abbérhantes

[ ]:
```
q1_circ = np.percentile(arbres_clean['circonference_cm'],25)
q3_circ = np.percentile(arbres_clean['circonference_cm'],75)
circonf_max_stat = q3_circ + 1.5*q1_circ

print(q1_circ)
print(q3_circ)
print('La circonférence limite calculée statistiquement est de',␣
↪circonf_max_stat, 'cm')
```

[ ]:
```
q1_haut = np.percentile(arbres_clean['hauteur_m'],25)
q3_haut = np.percentile(arbres_clean['hauteur_m'],75)
hauteur_max_stat = q3_haut + 1.5*q1_haut

print(q1_haut)
print(q3_haut)
print('La hauteur limite calculée statistiquement est de', hauteur_max_stat,␣
↪'m')
```

Détection et élimination des outliers pour les variables physiques circonference et hauteur

Nous avons déterminé les multiples suivants, en considérant les données disponibles:

1. circonference maximum ~= 2 x circonf_max_stat = 300 cm
2. hauteur maximum ~= 1.5 x hauteur_max_stat = 30 m

[ ]:
```
circonf_max_aparis = 300
hauteur_max_aparis = 30

arbres_clean =␣
↪arbres_clean[arbres_clean["circonference_cm"]<=circonf_max_aparis]
```

```
arbres_clean = arbres_clean[arbres_clean["hauteur_m"]<= hauteur_max_aparis]
```

[ ]:
```
np.shape(arbres_clean)
```

3.1.4 Bilan de la phase de nettoyage

[ ]:
```
print("le nombre de (lignes, colonnes) supprimées lors de la phase de nettoyage␣
↪est: (", np.shape(arbres)[0]-np.shape(arbres_clean)[0],",",np.
↪shape(arbres)[1]-np.shape(arbres_clean)[1],")")
```

3.1.5 Imputation statistique

Détermination des variables nécessitant imputation statistique par affichage des taux de remplissage

[ ]:
```
na_level_clean = 1-arbres_clean.isna().sum()/np.shape(arbres_clean)[0]
#print(na_level_clean);
print(na_level_clean[na_level_clean.values!=1.000000].index)
```

[ ]:
```
#Affichons les différents genre d'arbres, dont les valeurs de libellé français␣
↪sont NA

arbres_clean[pd.isna(arbres_clean['libelle_francais'])].genre.nunique()
arbres_clean[pd.isna(arbres_clean['libelle_francais'])].genre.unique();
```

[ ]:
```
#Affichons la distribution des libellés français pour tous les genres d'arbres␣
↪pour lesquels

test = 'Broussonetia'

arbres_clean[arbres_clean.genre==test].libelle_francais.unique();
arbres_clean[arbres_clean.genre==test].libelle_francais.mode()
arbres_clean[arbres_clean.genre==test].libelle_francais.nunique();
test_df = arbres_clean[arbres_clean.genre==test].libelle_francais.value_counts()
```

Imputation des valeurs manquantes de libellés français

[ ]:
```
#Si un libellé français ayant la valeur NA a un genre, remplaçons cette valeur␣
↪NA par le mode de la distribution des "autres" libellés français liés à son␣
↪genre

# Copions tous les arbres dont le libelle français est non spécifié, dans un␣
↪dataframe tampon
test_df = arbres_clean[pd.isna(arbres_clean['libelle_francais'])]
```

```
# Remplaçons la valeur non spécifiée par le mode de la distribution des␣
 ↪libellés français associés au genre de chacun de ces arbres
for ind, content in test_df.iterrows():

    test_val = arbres_clean[arbres_clean.genre==content.genre].libelle_francais.
 ↪mode() #Détermination du mode de la distribution des libellés français␣
 ↪associés au genre de chacun des arbres en question
    if (test_val.empty):
        arbres_clean.at[ind,'libelle_francais'] = np.nan
    else:
        if test_val.iloc[0] in ['Non spécifié']:
            arbres_clean.at[ind,'libelle_francais'] = np.nan
        else:
                arbres_clean.at[ind,'libelle_francais'] = test_val.iloc[0]
```

Imputation des valeurs manquantes d'espèces

[ ]:
```
#Si une espèce ayant la valeur NA a un genre, remplaçons cette valeur NA par le␣
 ↪mode de la distribution des "autres" espèces liées à son genre

# Copions tous les arbres dont l'espèce est non spécifiée, dans un dataframe␣
 ↪tampon
test_df = arbres_clean[pd.isna(arbres_clean['espece'])]

# Remplaçons la valeur non spécifiée par le mode de la distribution des espèces␣
 ↪associées au genre de chacun de ces arbres
for ind, content in test_df.iterrows():

    test_val = arbres_clean[arbres_clean.genre==content.genre].espece.mode()␣
 ↪#Détermination du mode de la distribution des espèces associées au genre de␣
 ↪chacun des arbres en question

    if (test_val.empty):
        arbres_clean.at[ind,'espece'] = np.nan
    else:
        if test_val.iloc[0] in ['n. sp.']:
            arbres_clean.at[ind,'espece'] = np.nan
        else:
                arbres_clean.at[ind,'espece'] = test_val.iloc[0]
```

Verification des taux de remplissages

[ ]:
```
np.shape(arbres_clean)[0]
na_level = 1-arbres_clean.isna().sum()/np.shape(arbres_clean)[0]
print(na_level[na_level.values!=1.000000])
```

[ ]:
```
plt.barh(na_level.index, width = na_level.values)
plt.title('Taux de remplissage des variables');
plt.xlabel('Taux en %'); plt.ylabel('Caractéristiques');

plt.savefig("./fillRatePostImput.png", bbox_inches='tight') # le secnd param permet d'éviter un rognement de l'image en sortie
```

Suppression post-imputation de la variable 'genre'

[ ]:
```
arbres_clean.drop(columns='genre')
```

3.1.6 Suppression des résidus de valeurs manquantes

[ ]:
```
arbres_clean = arbres_clean.dropna()
```

[ ]:
```
arbres_clean
```

4 Analyse Univariée

4.1 Variables quantitatives

4.1.1 Circonference

[ ]:
```
med_circ = np.median(arbres_clean['circonference_cm'])
```

[ ]:
```
plt.subplot()
plt.hist(arbres_clean['circonference_cm']);
plt.title('Distribution des valeurs de circonférences');
plt.xlabel('Circonférences agrégées'); plt.ylabel('Effectif');

plt.savefig("./histoCirconf.png", bbox_inches='tight') # le secnd param permet d'éviter un rognement de l'image en sortie
```

[ ]:
```
plt.boxplot(arbres_clean['circonference_cm']);
plt.title('Distribution Boxplot de la circonférence');
plt.xlabel('Boxplot'); plt.ylabel('Circonférences');

plt.savefig("./bxplotCirconf.png", bbox_inches='tight') # le secnd param permet d'éviter un rognement de l'image en sortie
```

4.1.2 Hauteur

[ ]:
```
med_haut = np.median(arbres_clean['hauteur_m'])
```

[ ]:
```
arbres_clean['hauteur_m'].hist(figsize=(10,4));
plt.title('Distribution des valeurs de hauteurs');
plt.xlabel('Hauteurs agrégées'); plt.ylabel('Effectifs');
```

[ ]:
```
plt.boxplot(arbres_clean['hauteur_m']);
plt.title('Distribution Boxplot de la hauteur');
plt.xlabel('Boxplot'); plt.ylabel('Hauteurs');
```

4.2 Variables qualitatives

4.2.1 Domanialité

[ ]:
```
uneliste =arbres_clean['domanialite'].value_counts().
 ↪sort_values(ascending=False).head(10).index.tolist()
arbres_clean[arbres_clean['domanialite'].isin(uneliste)]
ax = sns.countplot(x='domanialite',
                   order= uneliste,
                   data=arbres_clean[arbres_clean['domanialite'].
 ↪isin(uneliste)])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
plt.title('Effectifs d\'arbres dans les domanialités');

plt.savefig("./brplotDoman.png", bbox_inches='tight') # le secnd param permet␣
 ↪d'éviter un rognement de l'image en sortie
```

4.2.2 Arrondissement

[ ]:
```
uneliste =arbres_clean['arrondissement'].value_counts().
 ↪sort_values(ascending=False).head(20).index.tolist()
arbres_clean[arbres_clean['arrondissement'].isin(uneliste)]
ax = sns.countplot(x='arrondissement',
                   order= uneliste,
                   data=arbres_clean[arbres_clean['arrondissement'].
 ↪isin(uneliste)])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
plt.title('Effectifs d\'arbres dans les arrondissements');
```

4.2.3 Libellé français

[ ]:
```
uneliste =arbres_clean['libelle_francais'].value_counts().
 ↪sort_values(ascending=False).head(20).index.tolist()
ax = sns.countplot(x='libelle_francais',
                   order= uneliste,
                   data=arbres_clean[arbres_clean['libelle_francais'].
 ↪isin(uneliste)])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
plt.title('Effectifs d\'arbres par nom (20 plus importants)');
```

4.2.4 Lieu

[ ]:
```
uneliste =arbres_clean['lieu'].value_counts().sort_values(ascending=False).
 ↪head(20).index.tolist()
arbres_clean[arbres_clean['lieu'].isin(uneliste)]
ax = sns.countplot(x='lieu',
                   order= uneliste,
                   data=arbres_clean[arbres_clean['lieu'].isin(uneliste)])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90) ;
plt.title('Effectifs d\'arbres par lieu (20 plus importants)');
```

4.2.5 Espèce

[ ]:
```
uneliste =arbres_clean['espece'].value_counts().sort_values(ascending=False).
 ↪head(30).index.tolist();
arbres_clean[arbres_clean['espece'].isin(uneliste)];
ax = sns.countplot(x='espece',
                   order= uneliste,
                   data=arbres_clean[arbres_clean['espece'].isin(uneliste)])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
plt.title('Effectifs d\'arbres par espèce (30 plus importants)');
```

5 Analyse Bivariée

5.1 Domanialité

5.1.1 Domanialité - Arrondissement

[ ]:
```
uneliste1 =arbres_clean['arrondissement'].value_counts().
 ↪sort_values(ascending=False).head(20).index.tolist()
uneliste2 =arbres_clean['domanialite'].value_counts().
 ↪sort_values(ascending=False).head(2).index.tolist()
ax = sns.countplot(x='arrondissement',
                   hue='domanialite',
                   order= uneliste1,
                   data=arbres_clean[(arbres_clean['arrondissement'].
 ↪isin(uneliste1)) &
                                     (arbres_clean['domanialite'].
 ↪isin(uneliste2))])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);# SAns le semi-colon, dES␣
 ↪CHAINES DE CARACTÈRE apparaissent avant le graphe

plt.title('Effectifs d\'arbres par arrondissement, différenciés par domanialité␣
 ↪(Jardin vs Alignement)');
plt.legend(loc='upper right')

plt.savefig("./figDamArr.png", bbox_inches='tight') # le secnd param permet␣
 ↪d'éviter un rognement de l'image en sortie
```

5.1.2 Domanialité - Libellé français

[ ]:
```
uneliste1 =arbres_clean['libelle_francais'].value_counts().
 ↪sort_values(ascending=False).head(20).index.tolist()
uneliste2 =arbres_clean['domanialite'].value_counts().
 ↪sort_values(ascending=False).head(2).index.tolist()
ax = sns.countplot(x='libelle_francais',
                   hue='domanialite',
                   order= uneliste1,
                   data=arbres_clean[(arbres_clean['libelle_francais'].
 ↪isin(uneliste1)) &
                                     (arbres_clean['domanialite'].
 ↪isin(uneliste2))])
plt.title('Effectifs d\'arbres par libellé français, différenciés par␣
 ↪domanialité (Jardin vs Alignement)');
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
```

[ ]:

5.1.3 Domanialité - espèce

[ ]:
```
uneliste1 =arbres_clean['espece'].value_counts().sort_values(ascending=False).
 ↪head(20).index.tolist()
uneliste2 =arbres_clean['domanialite'].value_counts().
 ↪sort_values(ascending=False).head(2).index.tolist()
ax = sns.countplot(x='espece',
                   hue='domanialite',
                   order= uneliste1,
                   data=arbres_clean[(arbres_clean['espece'].isin(uneliste1)) &
                                     (arbres_clean['domanialite'].
 ↪isin(uneliste2))] )
plt.title('Effectifs d\'arbres par espèce, différenciés par domanialité (Jardin␣
 ↪vs Alignement)');
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
```

5.1.4 Domanialité - Circonference

[ ]:
```
uneliste1 =arbres_clean['circonference_cm'].value_counts().
 ↪sort_values(ascending=False).head(40).index.tolist()
uneliste2 =arbres_clean['domanialite'].value_counts().
 ↪sort_values(ascending=False).head(2).index.tolist()
ax = sns.countplot(x='circonference_cm',
                   hue='domanialite',
                   data=arbres_clean[(arbres_clean['circonference_cm'].
 ↪isin(uneliste1)) &
                                     (arbres_clean['domanialite'].
 ↪isin(uneliste2))] )

plt.title('Effectifs d\'arbres par circonférence, différenciés par domanialité␣
 ↪(Jardin vs Alignement)');
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
```

5.2 Arrondissement

5.2.1 Arrondissement - Espece

[ ]:
```
uneliste1 =arbres_clean['espece'].value_counts().sort_values(ascending=False).
 ↪head(15).index.tolist()
```

```
uneliste2 =arbres_clean['arrondissement'].value_counts().
 ↪sort_values(ascending=False).head(3).index.tolist()
ax = sns.countplot(x='espece',
                   hue='arrondissement',
                   order= uneliste1,
                   data=arbres_clean[arbres_clean['espece'].isin(uneliste1) &
                                     arbres_clean['arrondissement'].
 ↪isin(uneliste2)])

plt.title('Effectifs d\'arbres par espèce, différenciés par arrondissement (15E␣
 ↪13E 16E)');
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);

plt.savefig("./ArrEspece.png", bbox_inches='tight') # le secnd param permet␣
 ↪d'éviter un rognement de l'image en sortie
```

5.2.2 Circonference - Hauteur

[ ]:
```
uneliste1 =arbres_clean['circonference_cm'].value_counts().
 ↪sort_values(ascending=False).head(40).index.tolist()
uneliste2 =arbres_clean['hauteur_m'].value_counts().
 ↪sort_values(ascending=False).head(40).index.tolist()
arbres_clean[arbres_clean['lieu'].isin(uneliste)]
ax = sns.jointplot(x='circonference_cm', y='hauteur_m',␣
 ↪data=arbres_clean[arbres_clean['circonference_cm'].isin(uneliste1)],␣
 ↪kind='scatter')

plt.title('Distribution jointe \ncirconférence-hauteur',loc='left');
```

6 Synthèse de l'analyse de données

Le jeu de données analysé a porté sur les arbres de la ville de Paris.

Il comportait 200 137 individus et 19 caractéristiques dont 3 quantitatives portant sur les dimensions de l'arbre. On peut voir sur le chemin ci-dessous la carte de densité par arrondissement.

6.0.1 Démarche méthodologique d'analyse

J'ai procédé à un (1) un nettoyage des données, (2) une imputation des valeurs manquantes, puis finalement (3) une analyse univariée et bivariée.

(1) Nettoyage

Le nettoyage a permis de: ° Retirer toutes les colonnes ayant un taux de remplissage inférieur à 70%, ° Eliminer les variables non-pertinentes, sur la base du critère du nombre et de la qualité des modalités ° Filtrer les dimensions des arbres avec les multiples des quantiles de leurs distributions

(2) Imputation

Nous avons utiliser un **critère dendrologique** pour faire une imputation. Ce critère portait sur la relation genre-espèce et genre-libellé français. Il a été préféré à l'imputation simple ou imputation itérative, car en plus de la baleur statistique, il a aussi une valeur dendrologique. Nous avons pu gagner ainsi **92%** de valeurs manquantes sur le **libellé français** et **35%** sur les espèces

–> Bilan: Suppression: 13% de lignes, ~50% des colonnes

(3) Analyse univariée et bivariée

Une analyse univariée a été effectuée sur la grande majorité des caractéristiques restantes après le nettoyage. Elle a consisté à tracer les distributions, ainsi que calculer les indicateurs statistique de base (moyenne, médiane, écart-type).

Une analyse bivariée a été effectuée en croisant, avec principalement la variable domanialité (2 plus fortes modalités: Jardin et Alignement), les 4 variables Arrondissement, Libellé français, Espèce et Circonférence. Les distributions différenciées ont été tracées.

6.0.2 Carte de densité

[ ]:
```
#Calcul de la densité par arrondissement
arr_density = arbres_clean['arrondissement'].value_counts().
 ↪sort_values(ascending=False)/np.shape(arbres_clean)[0]
```

[ ]:
```
m1 = folium.Map(location=[arbres_clean.describe().loc['mean'].geo_point_2d_a,␣
 ↪arbres_clean.describe().loc['mean'].geo_point_2d_b],
                zoom_start=11.5)

for arr,dens in arr_density.items():

    folium.CircleMarker(
    location=[arbres_clean[arbres_clean['arrondissement']==arr].iloc[1].
 ↪geo_point_2d_a,
              arbres_clean[arbres_clean['arrondissement']==arr].iloc[1].
 ↪geo_point_2d_b],
    radius=120*dens,
    popup= arr[6:9] if arr.startswith('PARIS') else arr,
    fill=True,
    fill_color='Green'
    ).add_to(m1)

#Affichage de la carte instanciée
m1
```

[ ]: