

# main

June 26, 2021

See [Readme](#) for installation instructions

---

## 1 Concours Data is for Good : aidons Paris à devenir une smart-city !

### 1.1 Contexte

Dans le cadre du programme “Végétalisons la ville” organisé par la ville de Paris, nous proposons ici une analyse exploratoire des données OpenData concernant les arbres gérés par la ville de Paris.

L’objectif est d’aider Paris à devenir une “Smart-City” en gérant ses arbres de la manière la plus responsable possible. C’est-à-dire en optimisant les trajets nécessaires pour entretenir ces arbres.

### 1.2 Outils utilisés

Nous allons utiliser le langage Python, et présenter ici le code, les résultats et l’analyse sous forme de [Notebook Jupyter](#).

Nous allons aussi utiliser les bibliothèques usuelles d’exploration et analyse de données, afin d’améliorer la simplicité et la performance de notre code : \* [NumPy](#) et [Pandas](#) : effectuer des calculs scientifiques (statistiques, algèbre, ...) et manipuler des séries et tableaux de données volumineuses et complexes \* [Matplotlib](#), [Pyplot](#), [Seaborn](#) et [Plotly](#) : générer des graphiques lisibles, interactifs et pertinents

```
[1]: # Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

## If you use Notebook (and not JupyterLab), uncomment following lines
# import plotly.io as pio
# pio.renderers.default='notebook'
```

### 1.3 Chargement des données et premier aperçu

Les données mises à disposition sont issues de [opendata.paris.fr](https://opendata.paris.fr) et représentent “l’ensemble des arbres, ainsi que les arbres d’alignement, présents sur le territoire parisien et des cimetières extra-muros (hors de Paris).”

Nous allons dans un premier temps simplement charger les données en mémoire et observer quelques valeurs.

```
[2]: # load raw data into a Pandas DataFrame, separator is ";"
raw_data = pd.read_csv("https://s3-eu-west-1.amazonaws.com/static.oc-static.com/
↳prod/courses/files/AI+Engineer/
↳Project+2+Participez+%C3%A0+un+concours+sur+la+Smart+City/p2-arbres-fr.csv",
↳sep=';')

# display first 5 rows
raw_data.head()
```

```
[2]:      id type_emplacement domanialite  arrondissement complement_adresse \
0  99874      Arbre      Jardin  PARIS 7E ARRDT      NaN
1  99875      Arbre      Jardin  PARIS 7E ARRDT      NaN
2  99876      Arbre      Jardin  PARIS 7E ARRDT      NaN
3  99877      Arbre      Jardin  PARIS 7E ARRDT      NaN
4  99878      Arbre      Jardin  PARIS 17E ARRDT      NaN
```

```
      numero      lieu id_emplacement \
0      NaN  MAIRIE DU 7E 116 RUE DE GRENELLE PARIS 7E      19
1      NaN  MAIRIE DU 7E 116 RUE DE GRENELLE PARIS 7E      20
2      NaN  MAIRIE DU 7E 116 RUE DE GRENELLE PARIS 7E      21
3      NaN  MAIRIE DU 7E 116 RUE DE GRENELLE PARIS 7E      22
4      NaN  PARC CLICHY-BATIGNOLLES-MARTIN LUTHER KING  000G0037
```

```
      libelle_francais  genre      espece variete  circonference_cm \
0      Marronnier  Aesculus  hippocastanum  NaN      20
1      If  Taxus  baccata  NaN      65
2      If  Taxus  baccata  NaN      90
3      Erable  Acer  negundo  NaN      60
4      Arbre à miel  Tetradium  daniellii  NaN      38
```

```
      hauteur_m stade_developpement  remarquable  geo_point_2d_a  geo_point_2d_b
0      5      NaN      0.0  48.857620  2.320962
1      8      A      NaN  48.857656  2.321031
2      10     A      NaN  48.857705  2.321061
3      8      A      NaN  48.857722  2.321006
4      0      NaN      NaN  48.890435  2.315289
```

Nous voyons que, pour chaque arbre listé, nous disposons des informations suivantes (la description des colonnes est disponible sur le site [OpenData](https://opendata.paris.fr)) : - id : simple identifiant de l’arbre (entier, ex. : 99874) - type\_emplacement : type de l’emplacement (texte, ex. : "Arbre") - domanialite : type

de lieu auquel appartient l'arbre (texte, ex. : "Jardin") - `arrondissement` : arrondissement de Paris où est situé l'arbre (texte, ex. : "PARIS 7E ARRD") - `complement_adresse` : complement d'adress (texte, pas d'exemple visible) - `numero` : numéro de l'adress (texte, pas d'exemple visible) - `lieu` : adresse de l'arbre (texte, ex. : "MAIRIE DU 7E 116 RUE DE GRENELLE PARIS 7E") - `id_emplacement` : identifiant de l'emplacement (texte, ex. : "19") - `libelle_francais` : nom commun (vernaculaire) de l'espèce de l'arbre (texte, ex. : "Marronnier") - `genre` : genre de l'arbre (texte, ex. : "Aesculus") - `espece` : espèce de l'arbre (texte, ex. : "hippocastanum") - `variete` : variété de l'arbre (texte, pas d'exemple visible) - `circonference_cm` : circonférence en centimètres de l'arbre (entier, ex. : 20) - `hauteur_m` : taille en mètres de l'arbre (entier, ex. : 5) - `stade_developpement` : stade de développement de l'arbre (texte, ex. : "A" pour "Adulte") - `remarquable` : si l'arbre est "remarquable" ou non (booléen, ex. : 0 pour un arbre "non remarquable") - `geo_point_2d_a` : latitude de la position de l'arbre (nombre à virgule, ex. : 48.857620) - `geo_point_2d_b` : longitude de la position de l'arbre (nombre à virgule, ex. : 2.320962)

Nous voyons déjà que parmi les quelques premières données : - un certain certain nombre de valeurs ne sont pas fournies (NaN = "Not a Number" = donnée non disponible) - nous pouvons classer les variables selon leur type : - quantitatives - discrètes : `id`, `circonference_cm`, `hauteur_m` - continues : `geo_point_2d_a`, `geo_point_2d_b` - qualitatives - nominales : `type_emplacement`, `domanialite`, `arrondissement`, `complement_adresse`, `numero`, `lieu`, `id_emplacement`, `libelle_francais`, `genre`, `espece`, `variete` - ordinales : `stade_developpement`, `remarquable` - on peut aussi les classer en trois grandes catégories, d'après leur sens : - métadonnées internes au système : `id`, `id_emplacement`, `type_emplacement` - données de localisation : `arrondissement`, `complement_adresse`, `numero`, `lieu`, `geo_point_2d_a`, `geo_point_2d_b` - données de description : - taille : `circonference_cm`, `hauteur_m` et `stade_developpement` - type : `libelle_francais`, `genre`, `espece` et `variete` - autre : `remarquable`

Nous allons observer plus précisément les types de valeurs et les valeurs vides :

```
[3]: # Display data types and empty values
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200137 entries, 0 to 200136
Data columns (total 18 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id                    200137 non-null  int64
1   type_emplacement      200137 non-null  object
2   domanialite           200136 non-null  object
3   arrondissement        200137 non-null  object
4   complement_adresse    30902 non-null   object
5   numero                0 non-null       float64
6   lieu                  200137 non-null  object
7   id_emplacement        200137 non-null  object
8   libelle_francais      198640 non-null  object
9   genre                 200121 non-null  object
10  espece                198385 non-null  object
11  variete               36777 non-null   object
```

```

12  circonference_cm      200137 non-null  int64
13  hauteur_m            200137 non-null  int64
14  stade_developpement  132932 non-null  object
15  remarquable          137039 non-null  float64
16  geo_point_2d_a       200137 non-null  float64
17  geo_point_2d_b       200137 non-null  float64
dtypes: float64(4), int64(3), object(11)
memory usage: 27.5+ MB

```

Nous voyons alors que : - la colonne `numero` n'est jamais renseignée (`Non-Null count = 0`) - ce critère n'apporte donc pas d'information - les colonnes `complement_adresse` (`Non-Null count = 30902`) et `variete` (`Non-Null count = 36777`) sont très peu renseignées ( $> 80\%$  de valeurs non définies) - les informations apportées par ces colonnes seront donc très difficilement exploitables en l'état - les colonnes `stade_developpement` (`Non-Null count = 132932`) et `remarquable` (`Non-Null count = 137039`) sont partiellement renseignées ( $> 30\%$  de valeurs non définies) - les informations apportées par ces colonnes seront donc pas facilement exploitables en l'état - les colonnes `libelle_francais` (`Non-Null count = 198640`) et `espece` (`Non-Null count = 198385`) sont pas toujours renseignées ( $> 0,5\%$  de valeurs non définies) - les informations apportées par ces colonnes sont assez fiables, mais il faudra faire attention aux cas non renseignés

## 1.4 Première analyse statistique

Nous allons maintenant chercher à comprendre comment sont réparties les valeurs pour chaque caractéristique de nos arbres.

Une simple description statistique de chaque colonne nous donne les informations suivantes : - pour chaque donnée numérique (`id`, `circonference_cm`, `hauteur_m`, `remarquable`, `geo_point_2d_a` et `geo_point_2d_b`), nous obtenons : - le nombre de valeurs non vides (`count`) - la moyenne (`mean`) - l'écart-type (`std`) - les valeurs minimale (`min`) et maximale (`max`) - les 25, 50 (médiane) et 75 centiles (25%, 50% et 75%)

- our chaque donnée textuelle (`type_emplacement`, `domanialite`, `arrondissement`, `complement_adresse`, `lieu`, `id_emplacement`, `libelle_francais`, `genre`, `espece`, `variete` et `stade_developpement`), nous obtenons :
  - le nombre de valeurs non vides (`count`)
  - le nombre de valeurs différentes (`unique`)
  - la valeur la plus représentée (`top`)
  - la fréquence de la valeur la plus représentée (`freq`)

```

[4]: # Display statistical summary of each column
raw_data.describe(include="all")

```

```

[4]:
      count  2.001370e+05  200137  200136  200137
unique      NaN          1          9          25
top          NaN        Arbre  Alignement  PARIS 15E ARRD
freq          NaN  200137    104949    17151
mean  3.872027e+05      NaN      NaN      NaN
std    5.456032e+05      NaN      NaN      NaN

```

min	9.987400e+04	NaN	NaN	NaN
25%	1.559270e+05	NaN	NaN	NaN
50%	2.210780e+05	NaN	NaN	NaN
75%	2.741020e+05	NaN	NaN	NaN
max	2.024745e+06	NaN	NaN	NaN

	complement_adresse	numero \
count	30902	0.0
unique	3795	NaN
top	SN°	NaN
freq	557	NaN
mean	NaN	NaN
std	NaN	NaN
min	NaN	NaN
25%	NaN	NaN
50%	NaN	NaN
75%	NaN	NaN
max	NaN	NaN

	lieu id_emplacement \
count	200137 200137
unique	6921 69040
top	PARC FLORAL DE PARIS / ROUTE DE LA PYRAMIDE 101001
freq	2995 1324
mean	NaN NaN
std	NaN NaN
min	NaN NaN
25%	NaN NaN
50%	NaN NaN
75%	NaN NaN
max	NaN NaN

	libelle_francais	genre	espece	variete	circonference_cm \
count	198640	200121	198385	36777	200137.000000
unique	192	175	539	436	NaN
top	Platane	Platanus	x hispanica	Baumannii'	NaN
freq	42508	42591	36409	4538	NaN
mean	NaN	NaN	NaN	NaN	83.380479
std	NaN	NaN	NaN	NaN	673.190213
min	NaN	NaN	NaN	NaN	0.000000
25%	NaN	NaN	NaN	NaN	30.000000
50%	NaN	NaN	NaN	NaN	70.000000
75%	NaN	NaN	NaN	NaN	115.000000
max	NaN	NaN	NaN	NaN	250255.000000

	hauteur_m	stade_developpement	remarquable	geo_point_2d_a \
count	200137.000000	132932	137039.000000	200137.000000

unique	NaN	4	NaN	NaN
top	NaN	A	NaN	NaN
freq	NaN	64438	NaN	NaN
mean	13.110509	NaN	0.001343	48.854491
std	1971.217387	NaN	0.036618	0.030234
min	0.000000	NaN	0.000000	48.742290
25%	5.000000	NaN	0.000000	48.835021
50%	8.000000	NaN	0.000000	48.854162
75%	12.000000	NaN	0.000000	48.876447
max	881818.000000	NaN	1.000000	48.911485

	geo_point_2d_b
count	200137.000000
unique	NaN
top	NaN
freq	NaN
mean	2.348208
std	0.051220
min	2.210241
25%	2.307530
50%	2.351095
75%	2.386838
max	2.469759

Observons maintenant la distribution empirique de chaque variable, de manière non visuelle dans un premier temps, afin de voir quels types de graphes seront ensuite le plus adaptés :

```
[5]: # display value frequencies per column
for col in raw_data.columns:
    print(f'\n \
===== \n \
> { col } \n \
-----')

    counts = raw_data[col].value_counts()
    freq = raw_data[col].value_counts(normalize=True)
    display(pd.DataFrame({'count': counts, 'freq': freq}))
```

```
=====
> id
-----

count    freq
262144    1  0.000005
209715    1  0.000005
150300    1  0.000005
148253    1  0.000005
```

```

152351      1  0.000005
...
249171      1  0.000005
259412      1  0.000005
261461      1  0.000005
255318      1  0.000005
264191      1  0.000005

```

[200137 rows x 2 columns]

```

=====
> type_emplacement
-----

```

```

      count  freq
Arbre 200137  1.0

```

```

=====
> domanialite
-----

```

```

      count      freq
Alignement 104949 0.524388
Jardin      46262 0.231153
CIMETIERE   31926 0.159522
DASCO       6422  0.032088
PERIPHERIQUE 5327  0.026617
DJS         3900  0.019487
DFPE        1325  0.006620
DAC          21  0.000105
DASES        4  0.000020

```

```

=====
> arrondissement
-----

```

```

      count      freq
PARIS 15E ARRD 17151 0.085696
PARIS 13E ARRD 16712 0.083503
PARIS 16E ARRD 16403 0.081959
PARIS 20E ARRD 15340 0.076647
PARIS 19E ARRD 13709 0.068498
PARIS 12E ARRD 12600 0.062957
SEINE-SAINT-DENIS 11570 0.057810
BOIS DE VINCENNES 11510 0.057511
PARIS 14E ARRD 11399 0.056956
PARIS 17E ARRD 10762 0.053773
PARIS 18E ARRD 10011 0.050021

```

PARIS 7E ARRD	8617	0.043056
VAL-DE-MARNE	7580	0.037874
PARIS 8E ARRD	7245	0.036200
PARIS 11E ARRD	5658	0.028271
HAUTS-DE-SEINE	5298	0.026472
BOIS DE BOULOGNE	3978	0.019876
PARIS 10E ARRD	3385	0.016913
PARIS 4E ARRD	2740	0.013691
PARIS 5E ARRD	2368	0.011832
PARIS 6E ARRD	1764	0.008814
PARIS 1ER ARRD	1413	0.007060
PARIS 3E ARRD	1209	0.006041
PARIS 9E ARRD	1167	0.005831
PARIS 2E ARRD	548	0.002738

```
=====
> complement_adresse
-----
```

	count	freq
SN°	557	0.018025
1	552	0.017863
2	547	0.017701
3	498	0.016115
4	464	0.015015
...	...	...
N° 21	1	0.000032
288 .	1	0.000032
face au n 32	1	0.000032
f32 ou f25	1	0.000032
face 172 quai blieriot	1	0.000032

[3795 rows x 2 columns]

```
=====
> numero
-----
```

Empty DataFrame  
Columns: [count, freq]  
Index: []

```
=====
> lieu
-----
```

	count	freq
PARC FLORAL DE PARIS / ROUTE DE LA PYRAMIDE	2995	0.014965



PARC DES BUTTES CHAUMONT	2331	0.011647
PARC ANDRE CITROEN	2095	0.010468
PARC OMNISPORT SUZANNE LENGLEN / 7 BOULEVARD DE...	1478	0.007385
INSEP / AVENUE DU TREMBLAY	1293	0.006461
...	...	...
PISCINE D'AUTEUIL / 1 ROUTE DES LACS A PASSY	1	0.000005
RUE DU DOCTEUR LECENE	1	0.000005
CIMETIERE DE MONTPARNASSE GRAND / DIV 19	1	0.000005
ST GERMAIN L'AUXERROIS (6) MAT	1	0.000005
CIMETIERE DE SAINT-OUEN ANCIEN / DIV 7	1	0.000005

[6921 rows x 2 columns]

```
=====
> id_emplacement
-----
```

	count	freq
101001	1324	0.006615
101002	1241	0.006201
101003	1128	0.005636
202001	1032	0.005156
101004	1020	0.005097
...	...	...
A07700008005	1	0.000005
93506010	1	0.000005
85701008	1	0.000005
00ND0143	1	0.000005
A05800144010	1	0.000005

[69040 rows x 2 columns]

```
=====
> libelle_francais
-----
```

	count	freq
Platane	42508	0.213995
Marronnier	25207	0.126898
Tilleul	21305	0.107254
Erable	18389	0.092575
Sophora	11797	0.059389
...	...	...
Distylium	1	0.000005
Fremontia	1	0.000005
Cordyline	1	0.000005
Andromède	1	0.000005
Genêt	1	0.000005

[192 rows x 2 columns]

```
=====
>     genre
-----
```

	count	freq
Platanus	42591	0.212826
Aesculus	25341	0.126628
Tilia	21550	0.107685
Acer	18471	0.092299
Sophora	11830	0.059114
...	...	...
Enkianthus	1	0.000005
Asimina	1	0.000005
Distylium	1	0.000005
Pistacia	1	0.000005
Heptacodium	1	0.000005

[175 rows x 2 columns]

```
=====
>     espee
-----
```

	count	freq
x hispanica	36409	0.183527
hippocastanum	20039	0.101011
japonica	11822	0.059591
n. sp.	9063	0.045684
tomentosa	8962	0.045175
...	...	...
americanum	1	0.000005
myrtilloides	1	0.000005
x kewensis	1	0.000005
recurvata	1	0.000005
trojana	1	0.000005

[539 rows x 2 columns]

```
=====
>     variete
-----
```

	count	freq
Baumannii'	4538	0.123392
Briotii'	2827	0.076869

Euchlora'	2756	0.074938
Chanticleer'	2595	0.070560
Fastigiata'	2483	0.067515
...	...	...
Cockleshell'	1	0.000027
Nicoline'	1	0.000027
Otto Luyken'	1	0.000027
Minsum'	1	0.000027
Villa Taranto'	1	0.000027

[436 rows x 2 columns]

```
=====
>   circonference_cm
-----
```

	count	freq
0	25867	0.129246
20	9711	0.048522
70	6780	0.033877
60	6369	0.031823
80	6206	0.031009
...	...	...
357	1	0.000005
485	1	0.000005
1125	1	0.000005
1205	1	0.000005
511	1	0.000005

[531 rows x 2 columns]

```
=====
>   hauteur_m
-----
```

	count	freq
0	39219	0.195961
10	28632	0.143062
5	26345	0.131635
15	17228	0.086081
8	13628	0.068093
...	...	...
5155	1	0.000005
218	1	0.000005
91	1	0.000005
219	1	0.000005
255	1	0.000005

[143 rows x 2 columns]

```
=====
> stade_developpement
-----

      count      freq
A   64438  0.484744
JA  35444  0.266633
J   26937  0.202637
M    6113  0.045986
```

```
=====
> remarquable
-----

      count      freq
0.0 136855  0.998657
1.0   184  0.001343
```

```
=====
> geo_point_2d_a
-----

      count      freq
48.833321     2  0.000010
48.848812     2  0.000010
48.837168     2  0.000010
48.829912     2  0.000010
48.838318     2  0.000010
...           ...
48.849614     1  0.000005
48.835809     1  0.000005
48.872331     1  0.000005
48.841893     1  0.000005
48.870508     1  0.000005
```

[200107 rows x 2 columns]

```
=====
> geo_point_2d_b
-----

      count      freq
2.439665     2  0.000010
2.387348     2  0.000010
2.337371     2  0.000010
2.446277     2  0.000010
```

```

2.386442      2  0.000010
...          ...    ...
2.337337      1  0.000005
2.349625      1  0.000005
2.281117      1  0.000005
2.360074      1  0.000005
2.342481      1  0.000005

```

```
[200114 rows x 2 columns]
```

Nous voyons alors que : - chaque arbre possède un `id` unique - cette variable n'apporte donc aucun information - il n'y a qu'une seule valeur possible pour la variable `type_emplacement` : "Arbre" - ce critère n'apporte donc pas d'information - les valeurs respectives de `complement_adresse` et `id_emplacement` sont très disparates dans leur format (pas de valeurs très représentatives) et ne sont pas humainement parlantes - la colonne `lieu` peut être découpée avec le séparateur " / " afin de regrouper par exemple tous les lieux commençant par "CIMETIERE DE PANTIN" - les données de `circonference_cm` et `hauteur_m` ont des valeurs aberrantes dont il faudra tenir compte : - `minimum = 0` , ce qui semble impossible - `circonference_cm` : `maximum = 250255` et `hauteur_m` : `maximum = 881818` , ce qui semble impossible

## 1.5 Un peu de nettoyage

Nous allons : - supprimer les colonnes inutiles : `type_emplacement` et `numero` - renommer les valeurs de `stade_developpement` pour des valeurs plus explicites - découper la valeur de `lieu` avec le séparateur " / " - créer de nouvelles colonnes `top_XXX` où les valeurs les moins fréquentes seront remplacées par la valeur "Other" pour les colonnes `lieu`, `lieu_1`, `libelle_francais`, `genre`, `espece` et `variete`

```

[6]: # drop useless columns
clean_data = raw_data.drop(columns=['type_emplacement', 'numero'])

# replace `stade_developpement` values
clean_data.stade_developpement.replace({
    'J' : 'Jeune',
    'JA' : 'Jeune Adulte',
    'A' : 'Adulte',
    'M' : 'Mature',
}, inplace=True)

# extract the first part of column `lieu`
clean_data['lieu_1'] = clean_data["lieu"].str.split("/", expand=True)[0].str.
    ↳strip()

```

```

[7]: # Display top 10 values of lieu and lieu_1
fig, (ax1, ax2) = plt.subplots(2, 1,
    figsize=(16,12),
)

```

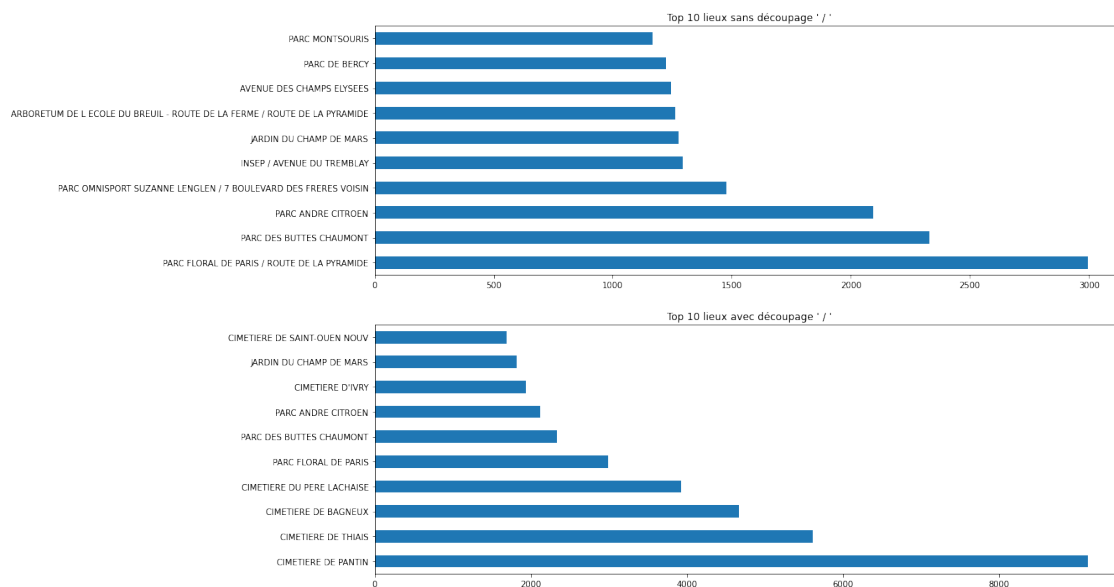
```

clean_data['lieu'].value_counts().head(10).plot(
    kind='barh',
    ax=ax1,
    title="Top 10 lieux sans découpage ' / '",
)

clean_data['lieu_1'].value_counts().head(10).plot(
    kind='barh',
    ax=ax2,
    title="Top 10 lieux avec découpage ' / '",
)

plt.show()

```



```

[8]: # Let's keep only the top values and merge the rest into "Other"
for col in ['lieu', 'lieu_1', 'libelle_francais', 'genre', 'espece', 'variete']:
    freq = clean_data[col].value_counts()
    clean_data['top_'+col] = clean_data[col].where(
        clean_data[col].isna() | clean_data[col].isin(freq.index[:20]),
        other='Other',
    )

```

```

[9]: # Let's see if categories are well organised
fig = px.parallel_categories(clean_data,
    dimensions=['arrondissement', 'top_lieu_1', 'top_lieu'],
    title="Classification des lieux",
    width=1000,

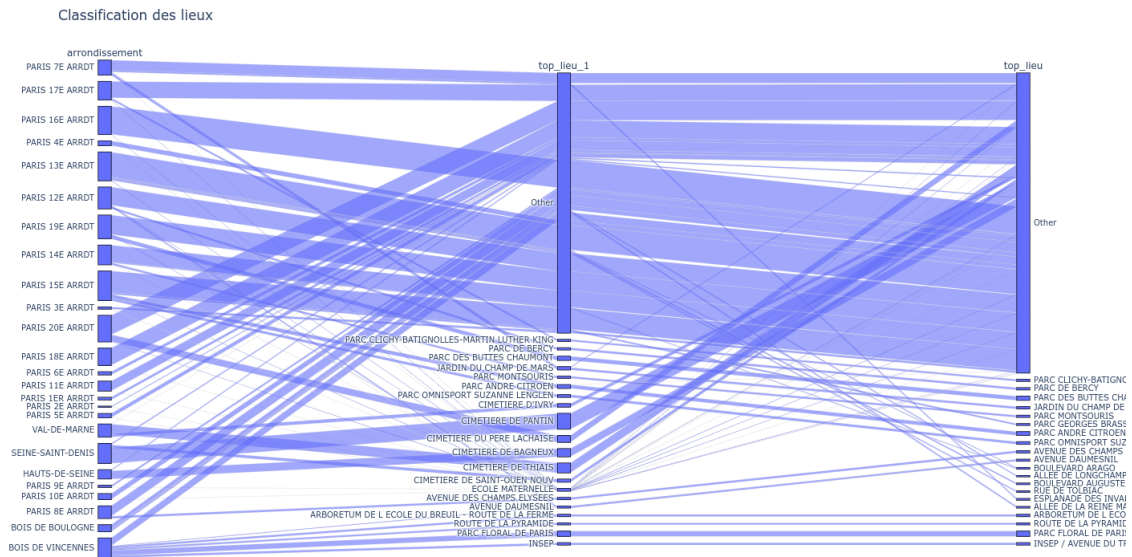
```

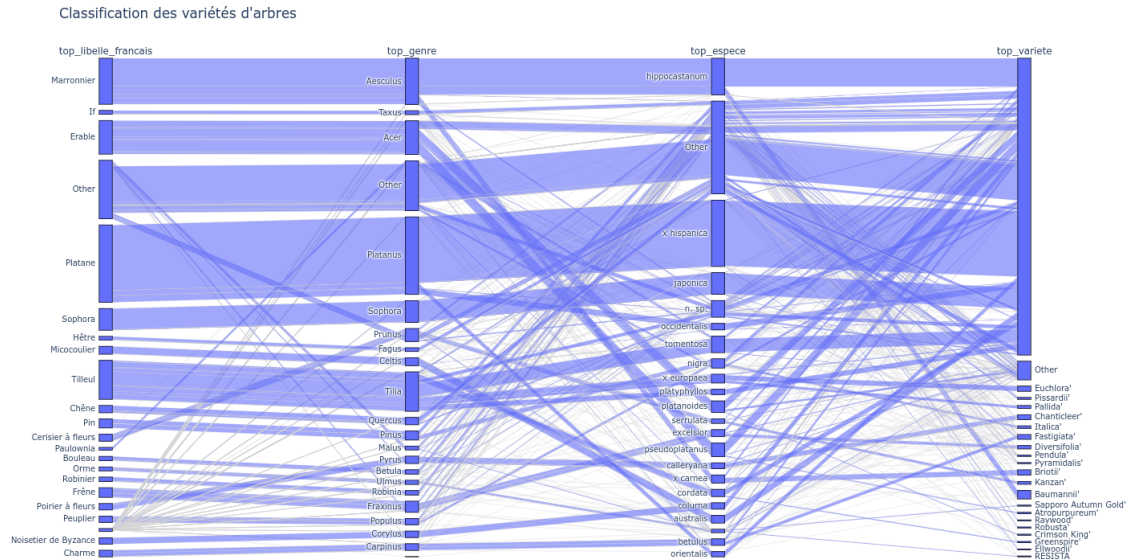
```

        height=800,
    )
    fig.show()

fig = px.parallel_categories(clean_data,
    dimensions=['top_libelle_francais', 'top_genre', 'top_espece',
↪ 'top_variete'],
    title="Classification des variétés d'arbres",
    width=1000,
    height=800,
)
fig.show()

```





## 1.6 Améliorons la gestion des arbres

Nous allons ici nous appuyer sur des analyses statistiques et des graphiques afin de voir comment il serait possible d'améliorer le service de gestion des arbres de Paris.

### 1.6.1 Quels arbres faut-il mesurer à nouveau ?

Pour la suite de l'analyse, nous allons éliminer les données aberrantes ("outliers"). Pour cela, nous allons utiliser le critère [IQR](#). Nous allons considérer toutes les données de taille trop éloignées de la norme, ainsi que les valeurs égales à 0 comme des données aberrantes.

Nous allons dans un premier temps afficher une cartographie de ces arbres, car ceux-ci devront être mesurés à nouveau afin d'améliorer la fiabilité de la gestion de nos arbres. Nous allons ensuite considérer ces données comme nulles (NaN).

```
[10]: # Let's work on a copy of our clean data.
data = clean_data.copy()

# First, let's consider zeros as NaN
data['circonference_cm'].where(data['circonference_cm'] > 0, inplace=True)
data['hauteur_m'].where(data['hauteur_m'] > 0, inplace=True)

# Let's compute the InterQuartile range in order to identify outliers
quartiles = data[['circonference_cm', 'hauteur_m']].quantile([0.25, 0.75])
iqr = quartiles.loc[0.75]-quartiles.loc[0.25]
limits = pd.DataFrame({
    'circonference_cm': [
```



```

        max(0, quartiles.loc[0.25, 'circonference_cm'] - 1.5 *
↳ iqr['circonference_cm']), # min
        quartiles.loc[0.75, 'circonference_cm'] + 1.5 * iqr['circonference_cm'],
↳ # max
    ],
    'hauteur_m': [
        max(0, quartiles.loc[0.25, 'hauteur_m'] - 1.5 * iqr['hauteur_m']), # min
        quartiles.loc[0.75, 'hauteur_m'] + 1.5 * iqr['hauteur_m'], # max
    ]
}, index=['min', 'max'])

display(quartiles, limits)

```

	circonference_cm	hauteur_m
0.25	45.0	6.0
0.75	123.0	14.0

	circonference_cm	hauteur_m
min	0.0	0.0
max	240.0	26.0

Nous voyons qu'un arbre "normal" aura : - une circonférence comprise entre 0 et 240 cm - une hauteur comprise entre 0 et 26 m

Nous allons maintenant visualiser où sont situés ces arbres "anormaux" (outliers), afin de planifier les tournées de mesure de ces arbres.

```

[11]: # outliers are the trees outside the IQR range
outliers = clean_data[
    ( clean_data['circonference_cm'] <= limits.loc['min', 'circonference_cm'] )
    | ( clean_data['circonference_cm'] >= limits.loc['max', 'circonference_cm'] )
    | ( clean_data['hauteur_m'] <= limits.loc['min', 'hauteur_m'] )
    | ( clean_data['hauteur_m'] >= limits.loc['max', 'hauteur_m'] )
]

```

```

[12]: # Count trees per burrough
count_per_arrondissement = outliers['arrondissement'].value_counts().head(20)

# resize figure
plt.figure(figsize=(16,9))

# plot horizontal bar chart
plt.barh(
    y=count_per_arrondissement.index,
    width=count_per_arrondissement.values,
)

# add labels for the value of each bar
for index, value in enumerate(count_per_arrondissement):

```

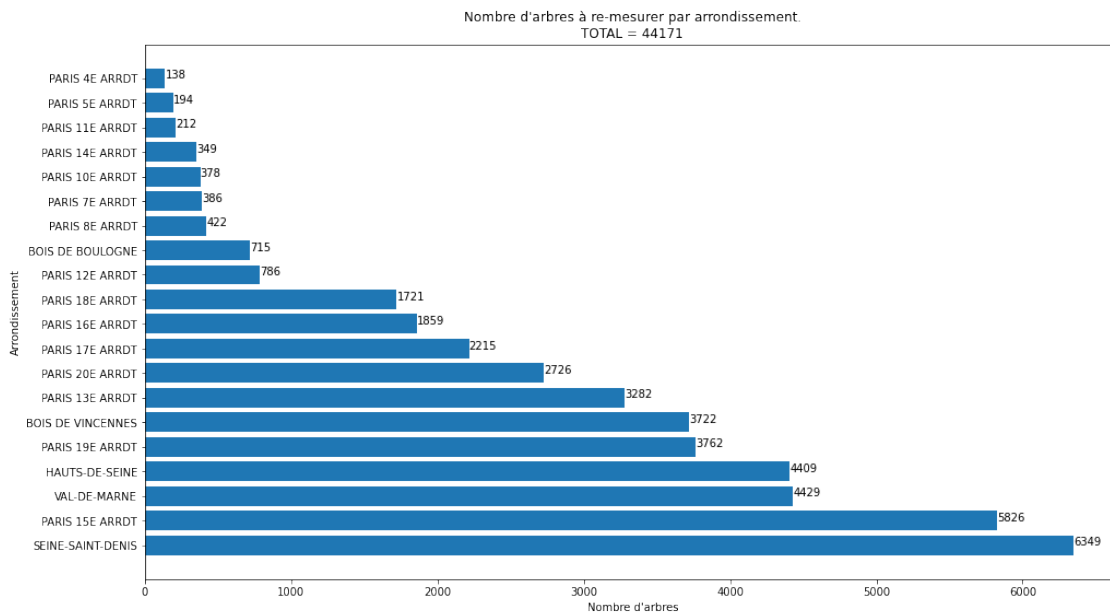
```

plt.text(y=index , x=value+1 , s=f"{value}")

# add title and labels
plt.xlabel("Nombre d'arbres")
plt.ylabel("Arrondissement")
plt.title(f"Nombre d'arbres à re-mesurer par arrondissement.\nTOTAL = {
    len(outliers) }")

# display the figure
plt.show()

```

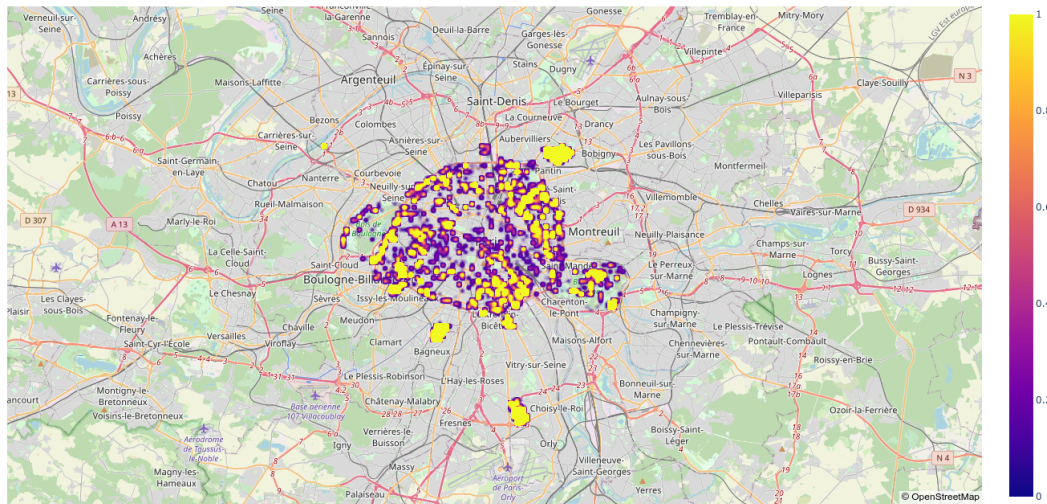


```

[13]: # Display the outliers on a map
fig = px.density_mapbox(outliers,
    lat='geo_point_2d_a', lon='geo_point_2d_b',
    hover_data=['circonference_cm', 'hauteur_m', 'arrondissement', 'lieu',
    'domanialite'],
    radius=2,
    zoom=10,
    mapbox_style="open-street-map",
    title="Localisation des arbres à re-mesurer",
    width=1000,
    height=800,
)
fig.show()

```

Localisation des arbres à re-mesurer



Nous voyons ici la carte des 44171 arbres qu'il faudrait mesurer à nouveau.

En attendant que ces arbres soient mesurés à nouveau, nous allons maintenant les ignorer dans nos prochaines analyses : passer à NaN les valeurs aberrantes.

```
[14]: # set to NaN data that are outside the range
for col in ['circonference_cm', 'hauteur_m']:
    clean_data[col] = clean_data[col].where((
        ( limits.loc['min', col] < clean_data[col] )
        & ( clean_data[col] < limits.loc['max', col] )
    ))
```

### 1.6.2 Quels arbres ont un développement anormal ?

Afin de gérer efficacement le patrimoine arboricole, il faut être capable de détecter les potentiels arbres malades ou qui ont des problèmes de développement.

Nous allons ici chercher quels arbres semblent avoir un développement anormal et donc qu'il faudrait contrôler en priorité.

```
[15]: # Let's remove empty values
clean_data_dropna = clean_data.dropna(subset=['circonference_cm', 'hauteur_m',
    ↳ 'stade_developpement', 'top_libelle_francais'])

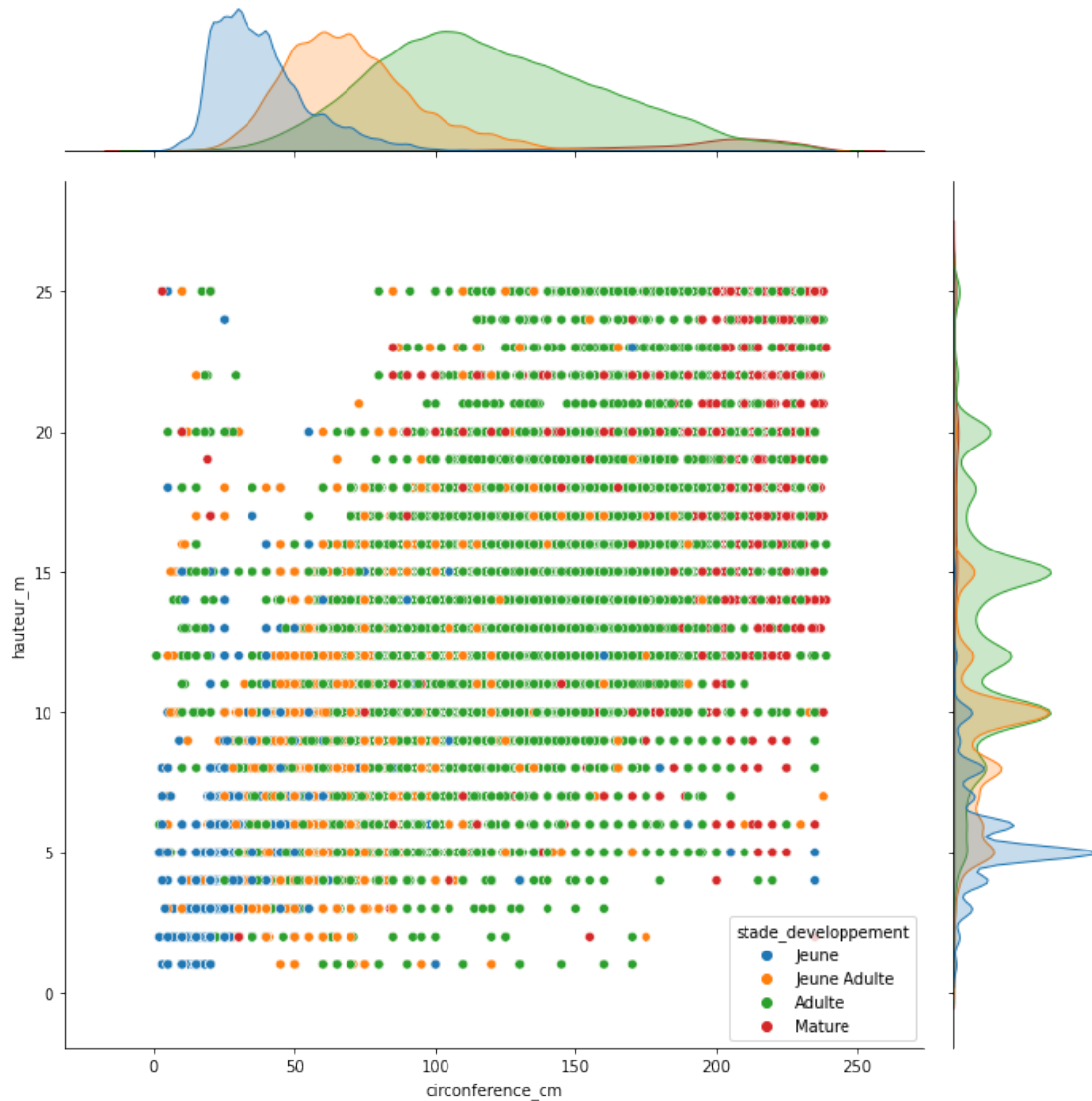
sns.jointplot(data=clean_data_dropna,
    x="circonference_cm",
    y="hauteur_m",
    hue="stade_developpement",
```

```

hue_order=['Jeune', 'Jeune Adulte', 'Adulte', 'Mature'],
height=10,
)

```

[15]: <seaborn.axisgrid.JointGrid at 0x7f47f47e4790>



```

[16]: # Display box plots for trees height and circumference per development stage
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,8))
ax1.set_title("Hauteur par stade de développement")
ax2.set_title("Circonférence par stade de développement")

sns.boxplot(data=clean_data_dropna,

```

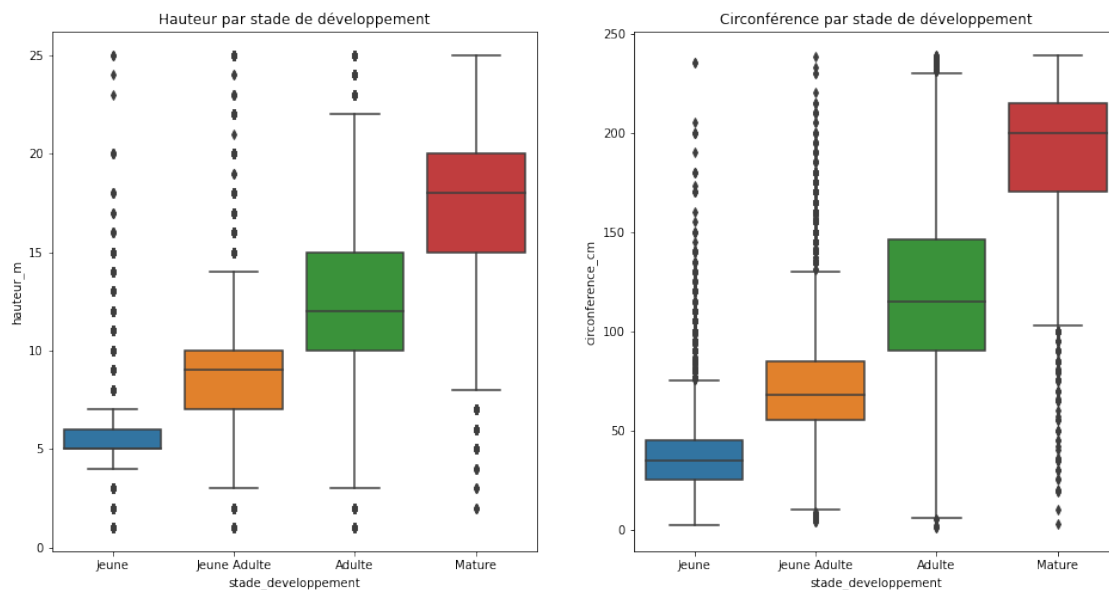
```

x="stade_developpement",
y="hauteur_m",
order=['Jeune', 'Jeune Adulte', 'Adulte', 'Mature'],
ax=ax1,
)

sns.boxplot(data=clean_data_dropna,
x="stade_developpement",
y="circonference_cm",
order=['Jeune', 'Jeune Adulte', 'Adulte', 'Mature'],
ax=ax2,
)

```

[16]: <AxesSubplot:title={'center':'Circonférence par stade de développement'},  
xlabel='stade\_developpement', ylabel='circonference\_cm'>



Nous voyons qu'il y a des arbres qui ont une taille anormale par rapport à leur stade de développement. Il faudrait contrôler leur santé et leur apporter les soins nécessaires (engrais, arrosage, traitements, ...).

### 1.6.3 Où sont situés les arbres qui vont nécessiter le plus d'entretien ?

Plus un arbre est grand, plus il nécessitera de techniciens, de temps, de matériel, d'arrosage et de produits pour son entretien. Maintenant que nous avons éliminé les valeurs aberrantes, nous allons cartographier les arbres en les pondérant avec leur hauteur.

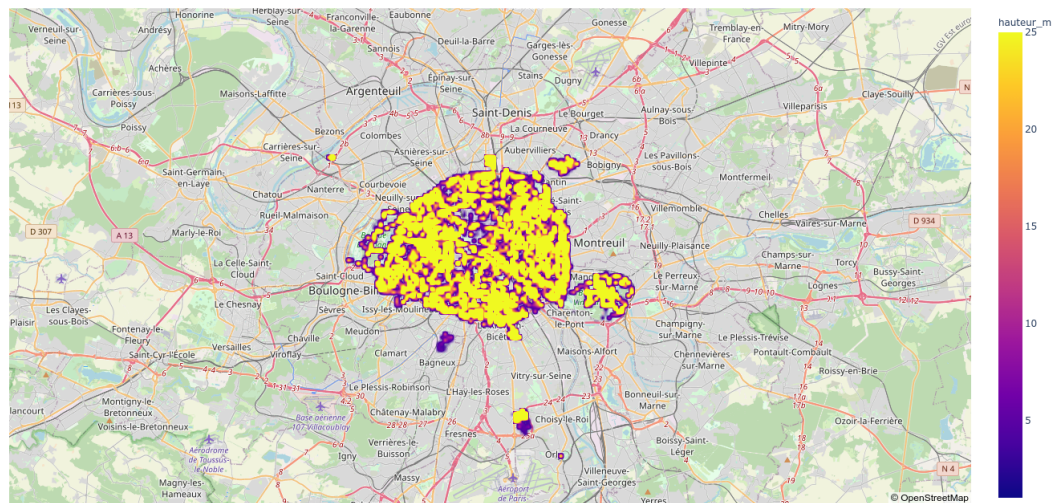
[17]: *# Display the trees on a map, weighted by size*  
fig = px.density\_mapbox(clean\_data,

```

lat='geo_point_2d_a', lon='geo_point_2d_b',
z='hauteur_m',
hover_data=['circonference_cm', 'hauteur_m', 'arrondissement', 'lieu', 'domanialite'],
radius=2,
zoom=10,
mapbox_style="open-street-map",
title="Localisation des arbres nécessitant le plus de moyens",
width=1000,
height=800,
)
fig.show()

```

Localisation des arbres nécessitant le plus de moyens



### 1.6.4 Quels sont les arbres les plus plantés actuellement ?

Nous allons travailler sur les données de catégories d'arbres, en se limitant aux valeurs les plus représentatives. Nous allons chercher à observer quel sont les types d'arbres les plus représentés selon leur type, et leur localisation.

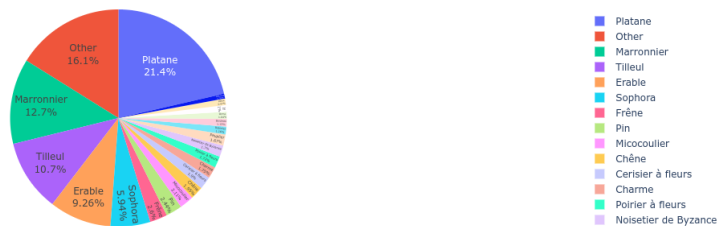
```

[18]: # Visualize repartition of type of trees
freq = clean_data['top_libelle_francais'].value_counts()
fig = px.pie(freq,
names=freq.index,
values=freq.values,
title="Types d'arbres",
)

```

```
fig.update_traces(
    textposition='inside',
    textinfo='percent+label'
)
fig.show()
```

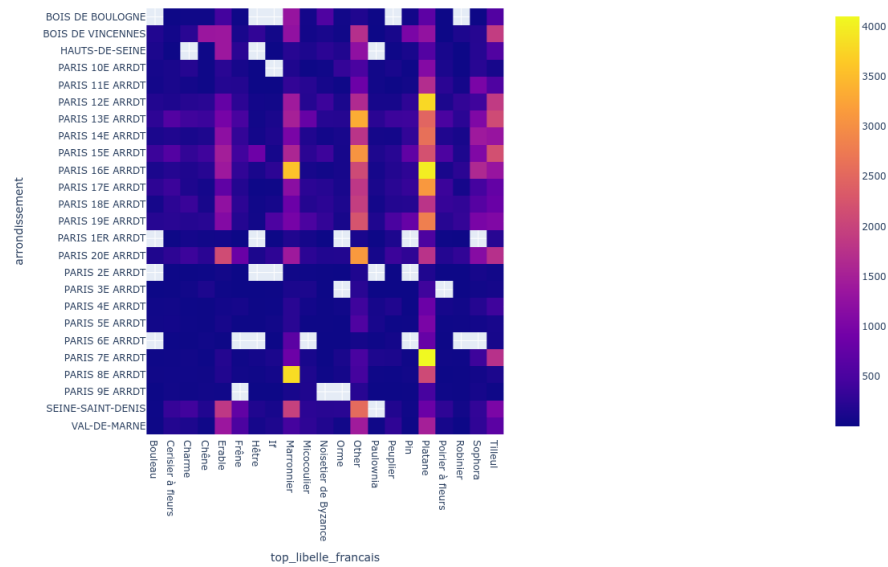
Types d'arbres



Nous voyons ici que seules 4 essences d'arbres représentent plus de 50% des arbres plantés. Cette information est importante pour adapter le matériel et les produits nécessaires à l'entretien des arbres. Cette information montre aussi qu'il pourrait y avoir un problème diversité des essences et donc de résilience du parc arboricole de Paris.

```
[19]: # Most common trees per burrough
table = clean_data.pivot_table(
    values='id',
    index='arrondissement',
    columns='top_libelle_francais',
    aggfunc='count',
    observed=True,
)
fig = px.imshow(table,
    title="Type d'arbre par arrondissement",
    width=1000,
    height=800,
)
fig.show()
```

Type d'arbre par arrondissement

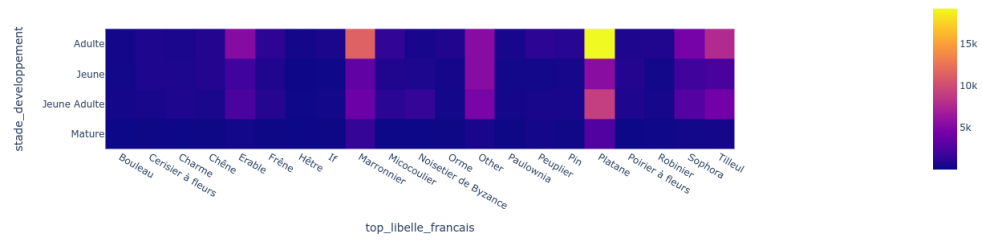


Nous voyons ici que nous avons beaucoup de peupliers, notamment dans le 7ème, le 12ème et le 16ème, ainsi que des marronniers dans le 8ème et le 16ème arrondissement. Cette information permet de dimensionner et répartir géographiquement les équipes et le matériel en fonction des types d'arbres plantés dans chaque arrondissement.

```
[20]: # Most common trees per age
table = clean_data.pivot_table(
    values='id',
    index='stade_developpement',
    columns='top_libelle_francais',
    aggfunc='count',
    observed=True,
)
fig = px.imshow(table,
    title="Type d'arbre par stade de développement",
    width=1000,
    height=400,
)
fig.show()
```



Type d'arbre par stade de développement



Nous voyons ici que la plupart des arbres sont des platanes adultes. Cette information permet d’optimiser les achats et le stockage du matériel et des produits adaptés spécifiquement à l’entretien de ces arbres.

---

*Licence GPL-v3*