

NB. : see [Readme](#) for installation instructions

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

## 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.

## 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

In [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'
```

## Chargement des données et premier aperçu

Les données mises à disposition sont issues de [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.

```
In [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.co  
  
# display first 5 rows  
raw_data.head()
```

Out[2]:

	<b>id</b>	<b>type_emplacement</b>	<b>domanialite</b>	<b>arrondissement</b>	<b>complement_adresse</b>	<b>numero</b>	
<b>0</b>	99874	Arbre	Jardin	PARIS 7E ARRDT		NaN	NaN MAIR 116 GF F
<b>1</b>	99875	Arbre	Jardin	PARIS 7E ARRDT		NaN	NaN MAIR 116 GF F
<b>2</b>	99876	Arbre	Jardin	PARIS 7E ARRDT		NaN	NaN MAIR 116 GF F
<b>3</b>	99877	Arbre	Jardin	PARIS 7E ARRDT		NaN	NaN MAIR 116 GF F
<b>4</b>	99878	Arbre	Jardin	PARIS 17E ARRDT		NaN	NaN PARC BATIGI LUTH

Nous voyons que, pour chaque arbre listé, nous disposons des informations suivantes (la description des colonnes est disponible sur le site [OpenData](#)) :

- **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 ARRDT" )
- **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" )
- **variете** : 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 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
    - ordinaires : 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 :

```
In [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

# 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%` )
  - pour 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` )

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

```
Out[4]:
```

	<b>id</b>	<b>type_emplacement</b>	<b>domanialite</b>	<b>arrondissement</b>	<b>complement_adresse</b>	<b>nui</b>
<b>count</b>	2.001370e+05		200137	200136	200137	30902
<b>unique</b>		NaN		1	9	25
<b>top</b>		NaN	Arbre	Alignement	PARIS 15E ARRDT	SN°
<b>freq</b>		NaN	200137	104949	17151	557
<b>mean</b>	3.872027e+05		NaN	NaN	NaN	NaN
<b>std</b>	5.456032e+05		NaN	NaN	NaN	NaN
<b>min</b>	9.987400e+04		NaN	NaN	NaN	NaN
<b>25%</b>	1.559270e+05		NaN	NaN	NaN	NaN
<b>50%</b>	2.210780e+05		NaN	NaN	NaN	NaN
<b>75%</b>	2.741020e+05		NaN	NaN	NaN	NaN
<b>max</b>	2.024745e+06		NaN	NaN	NaN	NaN

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 :

```
In [5]: for col in raw_data.columns:
    """ display value frequencies per column
    """
    print(f'\n {col}\n-----\n')
    counts = raw_data[col].value_counts()
    freq = raw_data[col].value_counts(normalize=True)
    display(pd.DataFrame({'count': counts, 'freq': freq}))
```

	<b>count</b>	<b>freq</b>
<b>262144</b>	1	0.000005
<b>209715</b>	1	0.000005
<b>150300</b>	1	0.000005
<b>148253</b>	1	0.000005
<b>152351</b>	1	0.000005
...	...	...
<b>249171</b>	1	0.000005
<b>259412</b>	1	0.000005
<b>261461</b>	1	0.000005
<b>255318</b>	1	0.000005

	count	freq
<b>264191</b>	1	0.000005

200137 rows × 2 columns

=====

> type\_emplacement

-----

	count	freq
<b>Arbre</b>	200137	1.0

=====

> domanalite

-----

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

=====

> arrondissement

-----

	count	freq
<b>PARIS 15E ARRDT</b>	17151	0.085696
<b>PARIS 13E ARRDT</b>	16712	0.083503
<b>PARIS 16E ARRDT</b>	16403	0.081959
<b>PARIS 20E ARRDT</b>	15340	0.076647
<b>PARIS 19E ARRDT</b>	13709	0.068498
<b>PARIS 12E ARRDT</b>	12600	0.062957
<b>SEINE-SAINT-DENIS</b>	11570	0.057810
<b>BOIS DE VINCENNES</b>	11510	0.057511
<b>PARIS 14E ARRDT</b>	11399	0.056956
<b>PARIS 17E ARRDT</b>	10762	0.053773
<b>PARIS 18E ARRDT</b>	10011	0.050021
<b>PARIS 7E ARRDT</b>	8617	0.043056
<b>VAL-DE-MARNE</b>	7580	0.037874
<b>PARIS 8E ARRDT</b>	7245	0.036200
<b>PARIS 11E ARRDT</b>	5658	0.028271
<b>HAUTS-DE-SEINE</b>	5298	0.026472
<b>BOIS DE BOULOGNE</b>	3978	0.019876

	count	freq
<b>PARIS 10E ARRDТ</b>	3385	0.016913
<b>PARIS 4E ARRDТ</b>	2740	0.013691
<b>PARIS 5E ARRDТ</b>	2368	0.011832
<b>PARIS 6E ARRDТ</b>	1764	0.008814
<b>PARIS 1ER ARRDТ</b>	1413	0.007060
<b>PARIS 3E ARRDТ</b>	1209	0.006041
<b>PARIS 9E ARRDТ</b>	1167	0.005831
<b>PARIS 2E ARRDТ</b>	548	0.002738

=====

> complement\_adresse

-----

	count	freq
<b>SN°</b>	557	0.018025
<b>1</b>	552	0.017863
<b>2</b>	547	0.017701
<b>3</b>	498	0.016115
<b>4</b>	464	0.015015
...	...	...
<b>f 49</b>	1	0.000032
<b>07-13</b>	1	0.000032
<b>acces pavillon</b>	1	0.000032
<b>16-38</b>	1	0.000032
<b>N103</b>	1	0.000032

3795 rows × 2 columns

=====

> numero

-----

count freq

=====

> lieu

-----

	count	freq
<b>PARC FLORAL DE PARIS / ROUTE DE LA PYRAMIDE</b>	2995	0.014965
<b>PARC DES BUTTES CHAUMONT</b>	2331	0.011647
<b>PARC ANDRE CITROEN</b>	2095	0.010468
<b>PARC OMNISPORT SUZANNE LENGLEN / 7 BOULEVARD DES FRERES VOISIN</b>	1478	0.007385
<b>INSEP / AVENUE DU TREMBLAY</b>	1293	0.006461
...	...	...
<b>CIMETIERE DE BAGNEUX / DIV 06</b>	1	0.000005
<b>CIMETIERE DE MONTMARTRE / DIV 5</b>	1	0.000005
<b>CIMETIERE DU PERE LACHAISE / AVENUE DES THUYAS / DIV 46</b>	1	0.000005

	count	freq
--	-------	------

JARDINET ANGLE DU BOULEVARD BEAUMARCHAIS SCARRON / 1 RUE SCARRON	1	0.000005
---	---	----------

CIMETIERE D'IVRY / AVENUE DU SUD / DIV 0	1	0.000005
--	---	----------

6921 rows × 2 columns

=====

> id\_emplacement

-----

	count	freq
<b>101001</b>	1324	0.006615
<b>101002</b>	1241	0.006201
<b>101003</b>	1128	0.005636
<b>202001</b>	1032	0.005156
<b>101004</b>	1020	0.005097
...	...	...
<b>A01500060007</b>	1	0.000005
<b>A05300122011</b>	1	0.000005
<b>D00000053017</b>	1	0.000005
<b>A14800094016</b>	1	0.000005
<b>K69K0028</b>	1	0.000005

69040 rows × 2 columns

=====

> libelle\_francais

-----

	count	freq
<b>Platane</b>	42508	0.213995
<b>Marronnier</b>	25207	0.126898
<b>Tilleul</b>	21305	0.107254
<b>Erable</b>	18389	0.092575
<b>Sophora</b>	11797	0.059389
...	...	...
<b>Asiminier</b>	1	0.000005
<b>Amla</b>	1	0.000005
<b>Euscaphis</b>	1	0.000005
<b>Fremontia</b>	1	0.000005
<b>Goyavier</b>	1	0.000005

192 rows × 2 columns

=====

> genre

-----

	count	freq
--	-------	------

	count	freq
<b>Platanus</b>	42591	0.212826
<b>Aesculus</b>	25341	0.126628
<b>Tilia</b>	21550	0.107685
<b>Acer</b>	18471	0.092299
<b>Sophora</b>	11830	0.059114
...	...	...
<b>Washingtonia</b>	1	0.000005
<b>Caragana</b>	1	0.000005
<b>Euscaphis</b>	1	0.000005
<b>Philadelphus</b>	1	0.000005
<b>Garrya</b>	1	0.000005

175 rows × 2 columns

	count	freq
<b>x hispanica</b>	36409	0.183527
<b>hippocastanum</b>	20039	0.101011
<b>japonica</b>	11822	0.059591
<b>n. sp.</b>	9063	0.045684
<b>tomentosa</b>	8962	0.045175
...	...	...
<b>lophantha</b>	1	0.000005
<b>recurvata</b>	1	0.000005
<b>baccata var. mandshurica</b>	1	0.000005
<b>platyphylla</b>	1	0.000005
<b>robusta</b>	1	0.000005

539 rows × 2 columns

	count	freq
<b>Baumannii'</b>	4538	0.123392
<b>Briotii'</b>	2827	0.076869
<b>Euchlora'</b>	2756	0.074938
<b>Chanticleer'</b>	2595	0.070560
<b>Fastigiata'</b>	2483	0.067515
...	...	...
<b>Nannetensis'</b>	1	0.000027

	count	freq
Nicoline'	1	0.000027
Hamabo'	1	0.000027
Lucombeana'	1	0.000027
Lemon Pippin'	1	0.000027

436 rows × 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 × 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 × 2 columns

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

	<b>count</b>	<b>freq</b>
<b>A</b>	64438	0.484744
<b>JA</b>	35444	0.266633
<b>J</b>	26937	0.202637
<b>M</b>	6113	0.045986

---



---



---

>     remarquable

---



---

	<b>count</b>	<b>freq</b>
<b>0.0</b>	136855	0.998657
<b>1.0</b>	184	0.001343

---



---

>     geo\_point\_2d\_a

---



---

	<b>count</b>	<b>freq</b>
<b>48.833321</b>	2	0.000010
<b>48.848812</b>	2	0.000010
<b>48.837168</b>	2	0.000010
<b>48.829912</b>	2	0.000010
<b>48.838318</b>	2	0.000010
...	...	...
<b>48.849614</b>	1	0.000005
<b>48.835809</b>	1	0.000005
<b>48.872331</b>	1	0.000005
<b>48.841893</b>	1	0.000005
<b>48.870508</b>	1	0.000005

---

200107 rows × 2 columns

---



---

>     geo\_point\_2d\_b

---



---

	<b>count</b>	<b>freq</b>
<b>2.439665</b>	2	0.000010
<b>2.387348</b>	2	0.000010
<b>2.337371</b>	2	0.000010
<b>2.446277</b>	2	0.000010
<b>2.386442</b>	2	0.000010
...	...	...
<b>2.337337</b>	1	0.000005
<b>2.349625</b>	1	0.000005
<b>2.281117</b>	1	0.000005
<b>2.360074</b>	1	0.000005
<b>2.342481</b>	1	0.000005

---

200114 rows × 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

## 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`

In [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.
```

In [7]:

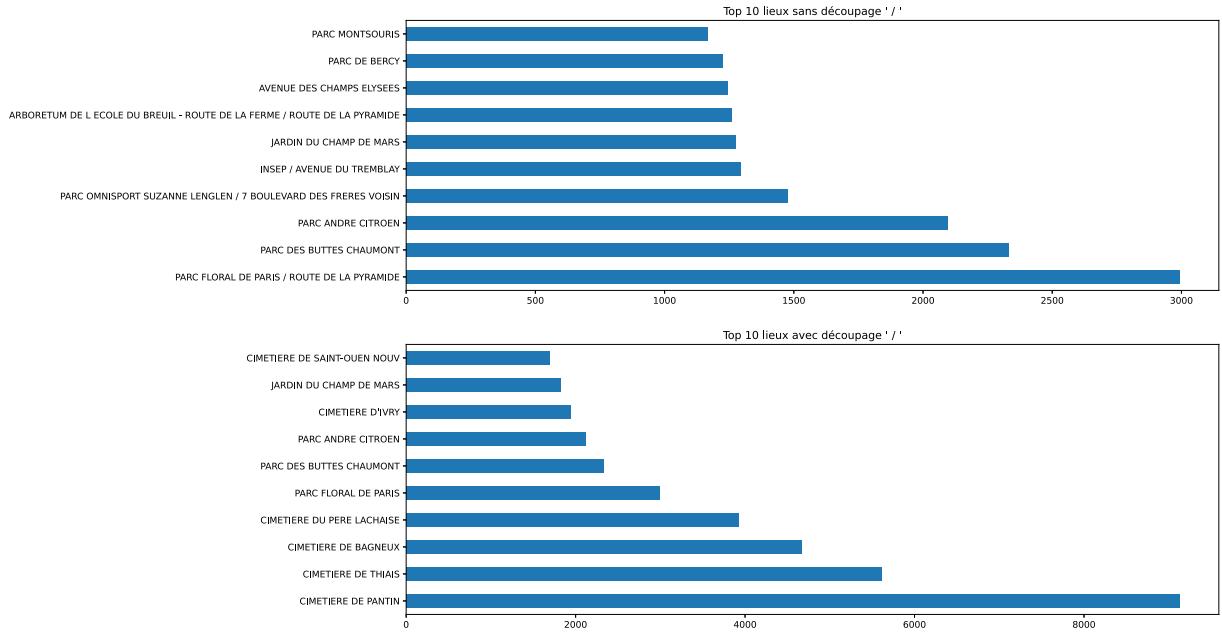
```
# Display values of lieu and lieu_1
fig, (ax1, ax2) = plt.subplots(2, 1,
    figsize=(16,12),
)

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() # Affiche le graphique

```



In [8]:

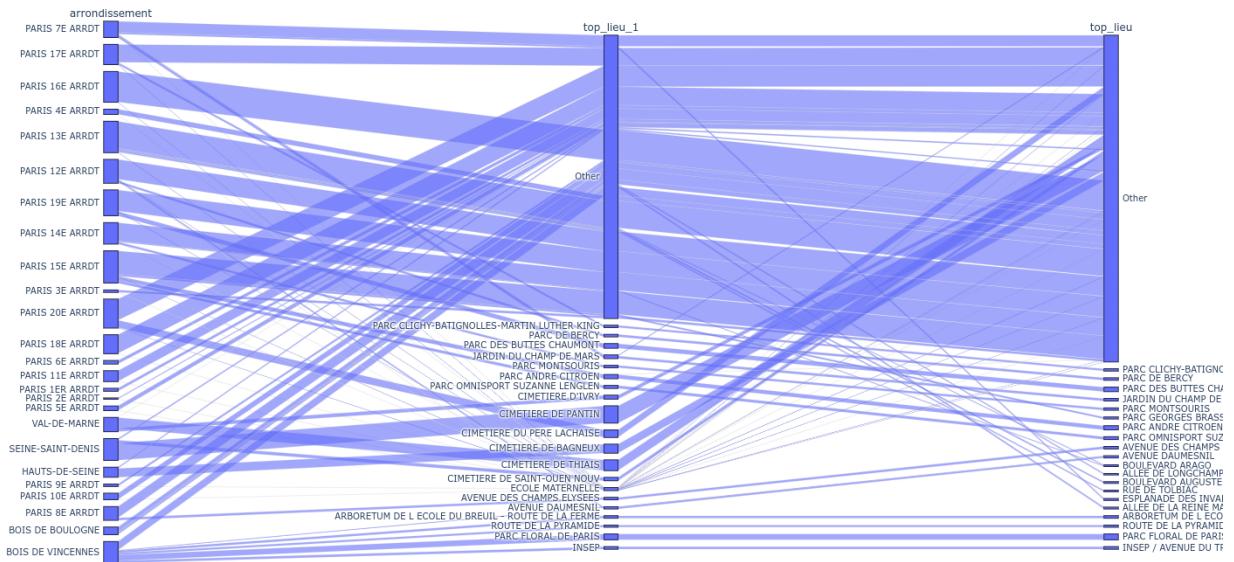
```
# Let's keep only the top values and merge the rest into "Other"
for col in ['lieu', 'lieu_1', 'libelle_francais', 'genre', 'espece', 'variете']:
    freq = clean_data[col].value_counts()
    clean_data['top_'+col] = clean_data[col].where(
        clean_data[col].isna() | clean_data[col].isin(freq.index[:20]),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

In [9]:

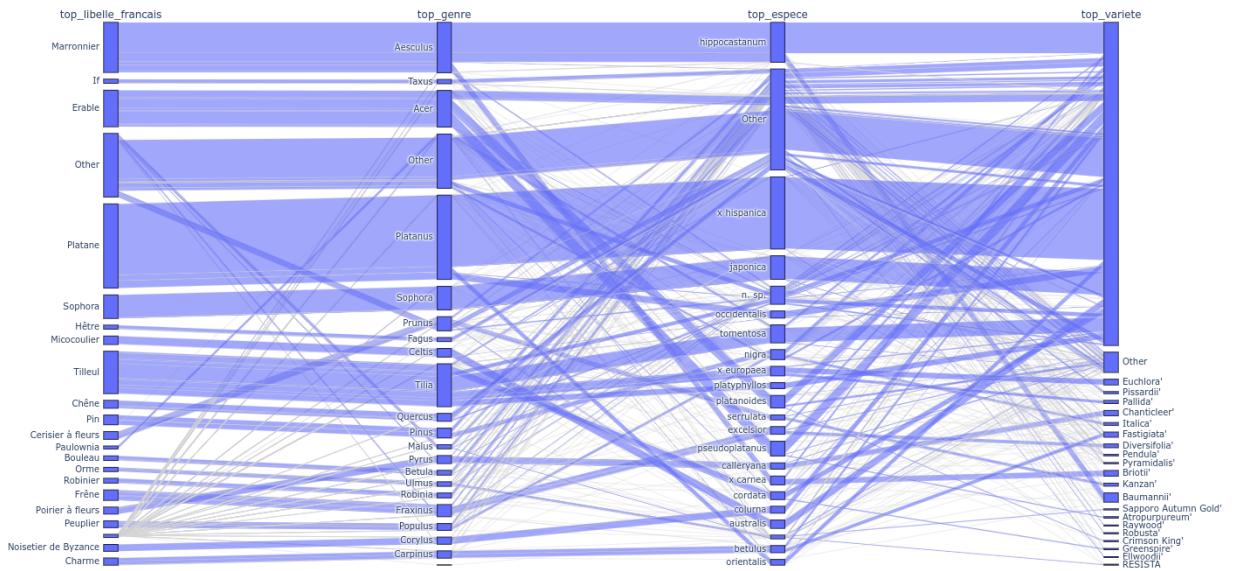
```
# Let's see if categories are well organised
fig = px.parallel_categories(clean_data,
    dimensions=['arrondissement', 'top_lieu_1', 'top_lieu'],
    title="Classification des lieux",
    width=1000,
    height=800,
)
fig.show()

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

### Classification des lieux



### Classification des variétés d'arbres



## 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.

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

Pour la suite de l'analyse, nous allons éliminer les données aberrantes ("outliers"). Pour celà, 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 ).

```
In [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']),
        quartiles.loc[0.75,'circonference_cm'] + 1.5 * iqr['circonference_cm']
    ],
    '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.

```
In [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'] )
]
```

```
In [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,
```

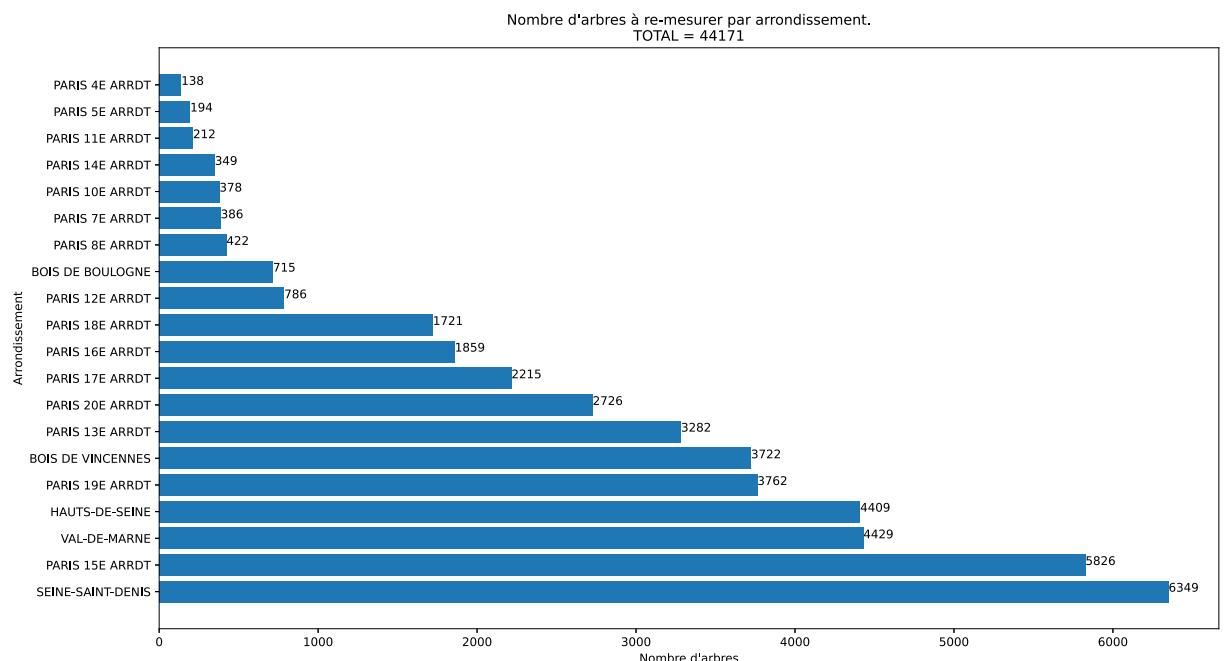
```

)
# 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(count_per_arrondissement)}")

# display the figure
plt.show()

```

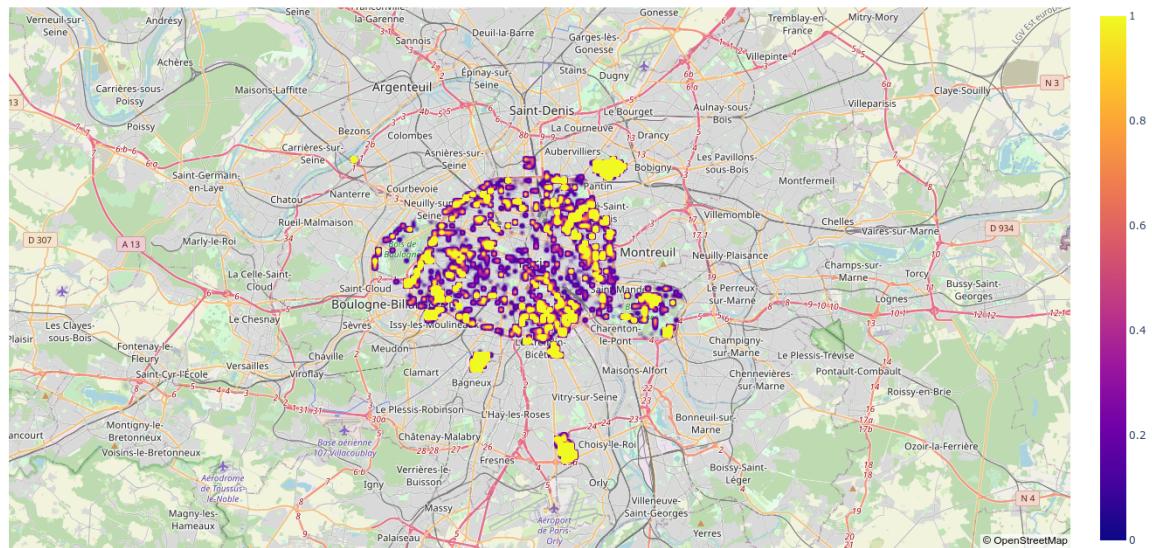


In [13]:

```

# 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', 'code_insee'],
                        radius=2,
                        zoom=10,
                        mapbox_style="open-street-map",
                        title="Localisation des arbres à re-mesurer",
                        width=1000,
                        height=800,
)
fig.show()

```



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 "hors norme".

In [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])
    ))
```

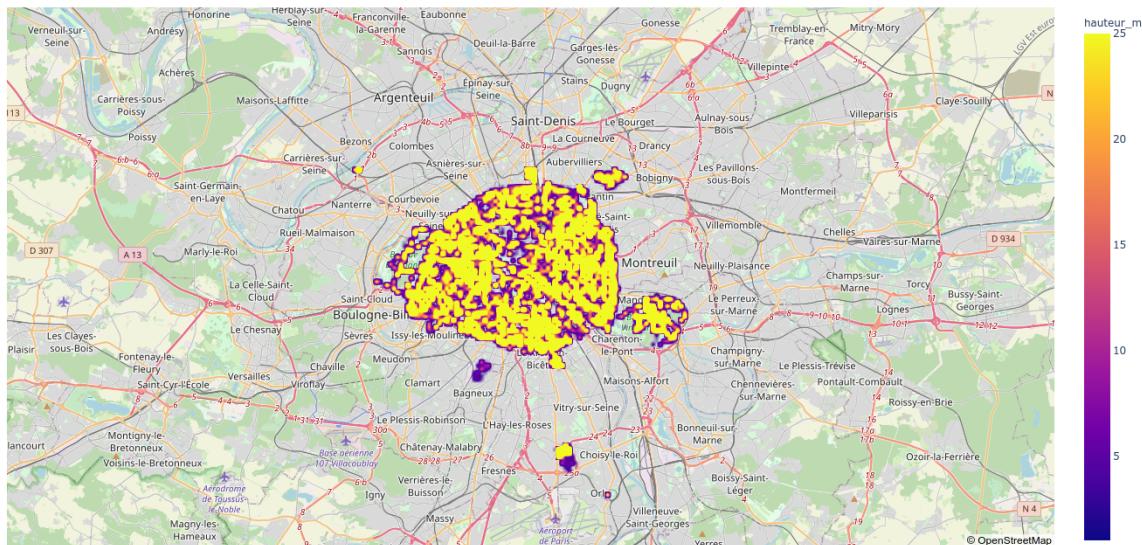
## Où sont situés les arbres qui vont nécessiter le plus d'entretien ?

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.

In [15]:

```
# 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', 'c
    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



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.

In [16]:

```
# 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()
```



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'enretien 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.

In [17]:

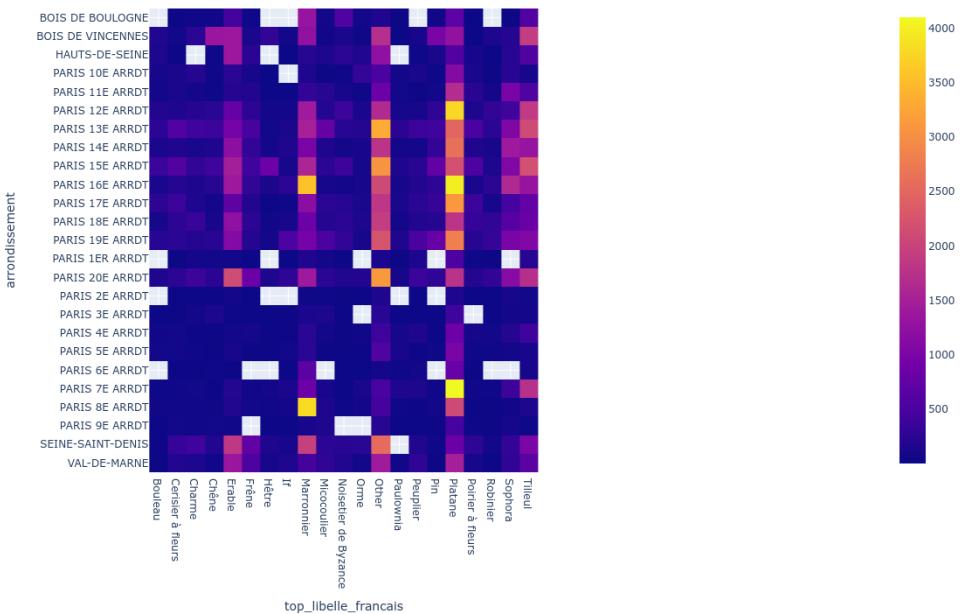
# Most common trees per burrough

```

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 les équipes et le matériel en fonction des types d'arbres plantés dans chaque arrondissement.

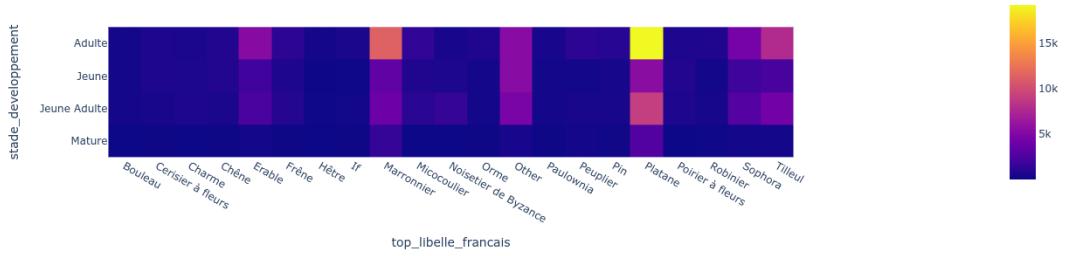
In [18]:

```

# 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'adapter le matériel et les produits adaptés à l'entretien de ces arbres.

## Quels arbres ont un développement anormal ?

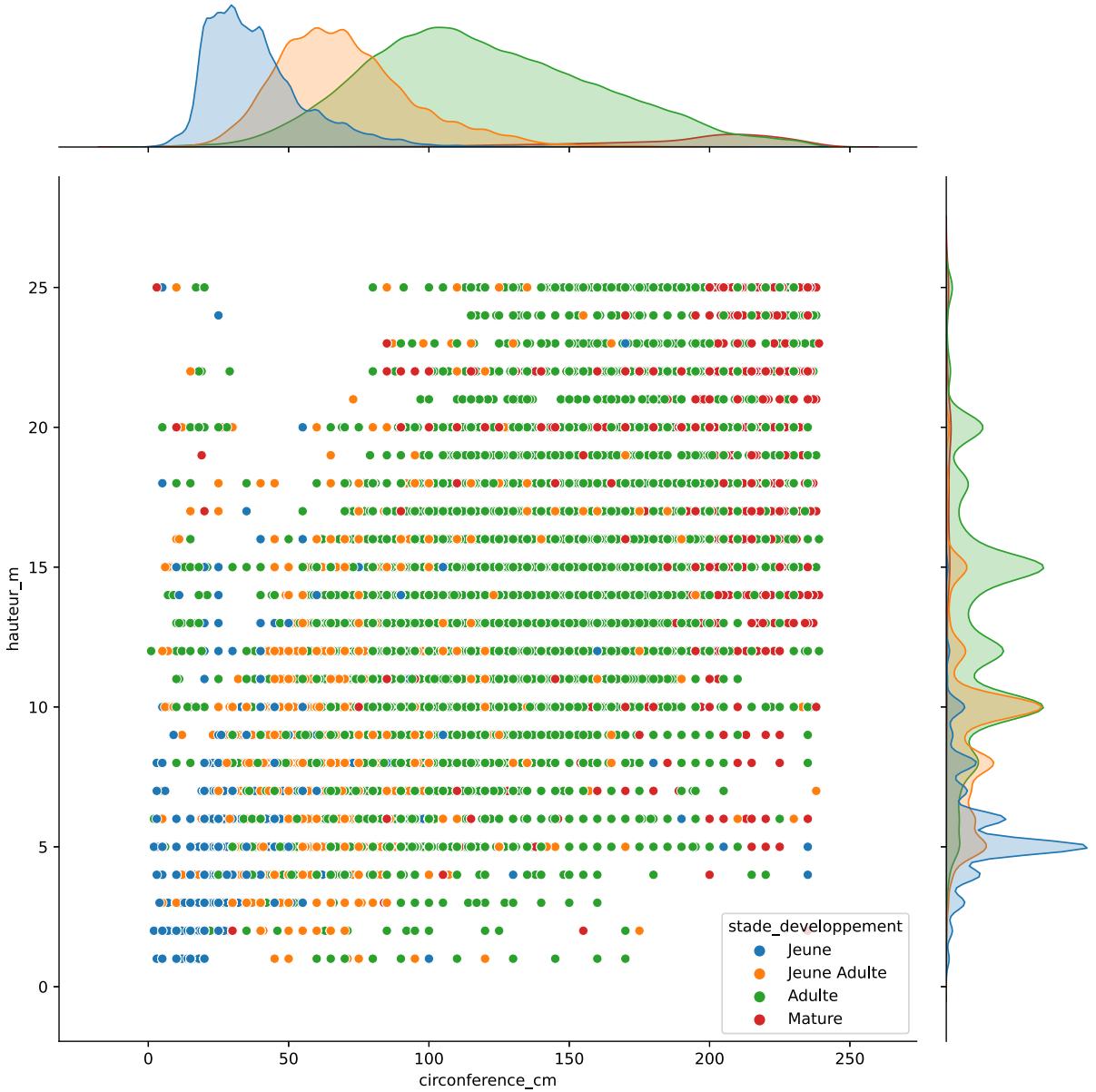
Afin de gérer efficacement le patrimoine arboricole, il faut être capable de détecter les potentiels arbres malades ou qui ont des problèmes de développement.

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

In [19]:

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

sns.jointplot(data=clean_data_dropna,
               x="circonference_cm",
               y="hauteur_m",
               hue="stade_developpement",
               hue_order=['Jeune', 'Jeune Adulte', 'Adulte', 'Mature'],
               height=10,
);
```



In [20]:

```

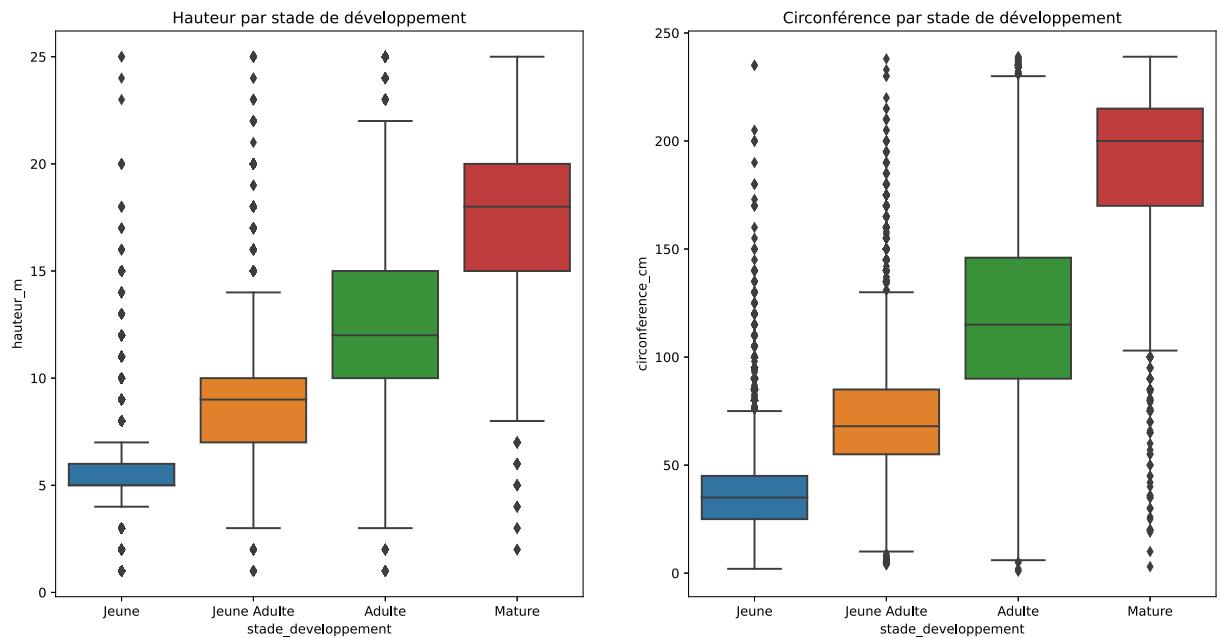
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,8))
ax1.set_title("Hauteur par stade de développement")
ax2.set_title("Circonférence par stade de développement")

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,
            )

```

Out[20]: <AxesSubplot:title={'center':'Circonférence par stade de développement'}, xlabel='stade\_developpement', ylabel='circonference\_cm'>



Nous voyons qu'il y a des arbres "hors norme" qu'il faudrait contrôler et traiter en priorité.