Anthony VIDRAGO – Juillet 2024

AI Engineer - Projet 2

Participez à un concours sur la Smart City

Contexte

Participation au challenge de l'ONG « Data is for Good » proposé par la capitale française afin de réaliser une analyse exploratoire avec un jeu de données portant sur les arbres de la ville de Paris, dans le cadre du programme « Végétalisons la ville ».

Les résultats de ce challenge permettrons d'optimiser les tournées pour l'entretien des arbres.

Les livrables :

- Un **Jupyter Notebook** comportant votre exploration du jeu de données.
- Un **support de présentation.**
    - Avec trois parties : la présentation générale du jeu de données, la démarche méthodologique d'analyse de données et la synthèse de votre analyse de données.

Sommaire

LANGAGE PLATFORME APPLICATION / IDE

Python

Anaconda

Jupyter Notebook

VS Code

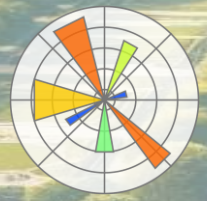
LIBRAIRIES

Pandas

Les DataFrames de Pandas offrent une représentation flexible et efficace des données tabulaires, similaires aux feuilles de calcul

Numpy

Fournit des tableaux (arrays) multidimensionnels et une vaste gamme de fonctions mathématiques pour effectuer des opérations numériques complexes de manière optimisée



Matplotlib

Offre une grande flexibilité pour créer des graphiques de haute qualité, allant des simples plots aux visualisations complexes

Seaborn

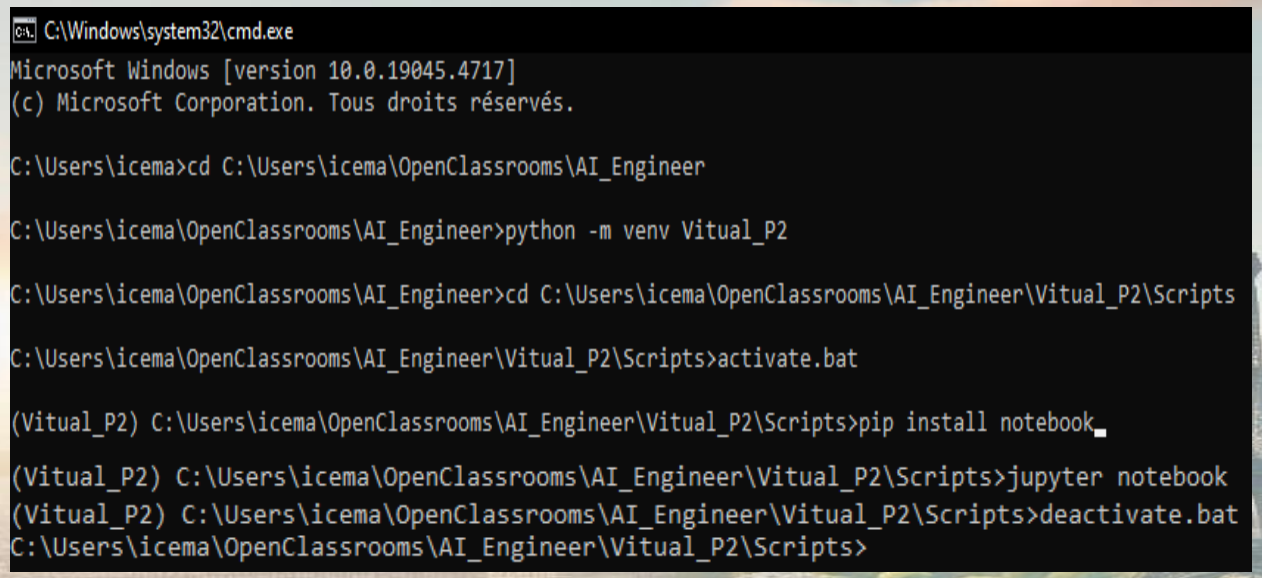
Construit sur Matplotlib, Seaborn simplifie la création de visualisations esthétiques et informatives pour explorer les données

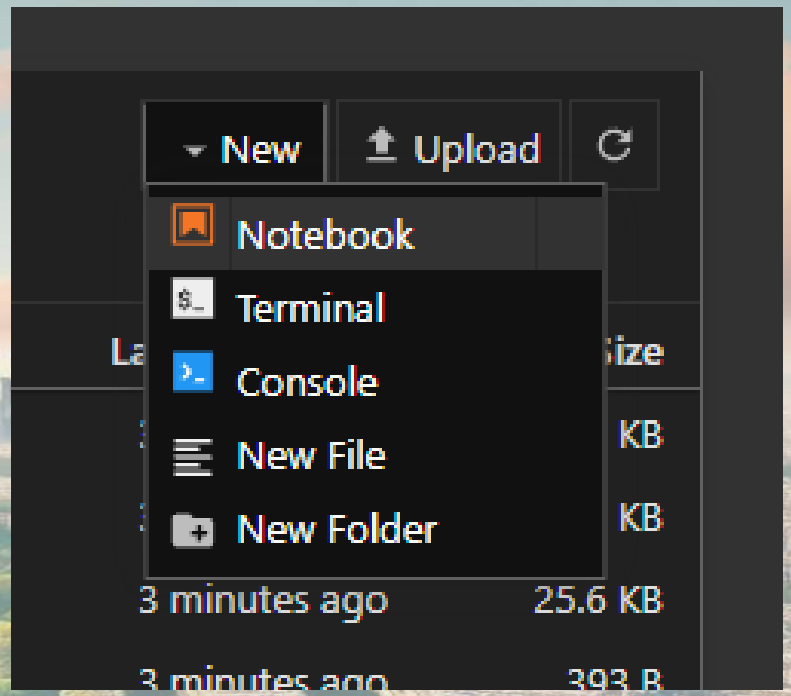
Plotly

Permet de créer des graphiques interactifs et dynamiques, idéaux pour les applications web et les présentations

Folium

Spécialisée dans la visualisation de données géospatiales, permettant de créer des cartes interactives en superposant des données sur des cartes de base





Environnement Jupyter Notebook

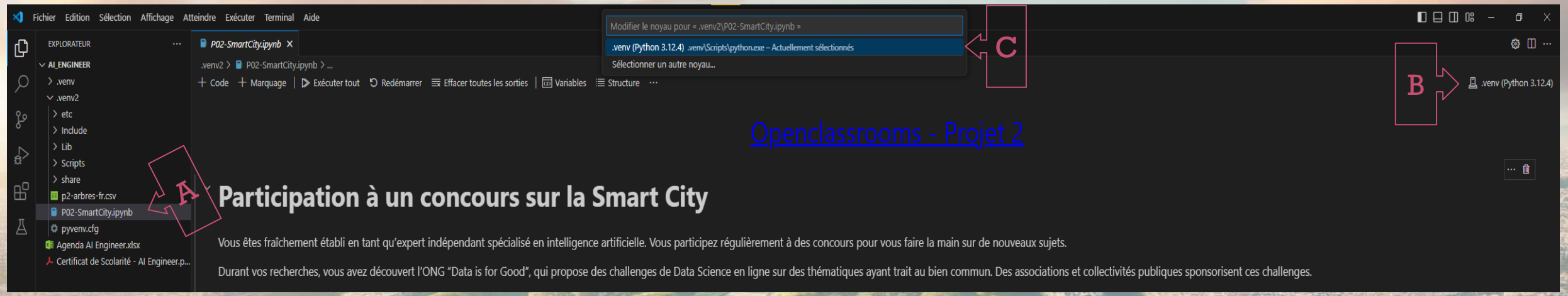
1 / La commande « cd » sélectionne l'emplacement

2 / pour installer l'environnement virtuel taper : python –m venv « nomdudossier »

3 / Utilisez la commande activate.bat pour démarrer l'environnement virtuel depuis le dossier « Scripts »

4/ installez Jupyter notebook avec « pip install notebook »

5/ Lancez l'application en tapant « jupyter notebook »



Environnement VS Code

1 / Après avoir suivi les étapes pour l'environnement Jupyter et crée votre notebook

2 / depuis le menu démarrer, taper VS Code puis rechercher votre dossier en tapant ctrl+o ou « fichier » puis « ouvrir le dossier »

3 / Sélectionner votre notebook (image A), VS Code vous demandera l'autorisation d'installer l'extension python

4 / Sélectionner le noyau (image B) puis (image C) environnement Python > créer un environnement Python > Venv Creates…. > Enter interpréter path > Find… et sélectionner l'application python.exe qui se trouve dans le dossier Scripts (dans l'exemple : .venv2\Scripts)

```
#pip install jupyterlab-language-pack-fr-FR
```

```
#!pip install contextily
#!pip install numpy==1.26.4
#!pip install --upgrade matplotlib
#!pip uninstall -y (nom librairie)
```

```
#!pip install pandas numpy matplotlib seaborn plotly folium
```

```
#pip install notebook
```

```
import pandas as pd # v2.2.2
import numpy as np # v1.26.4
import matplotlib.pyplot as plt
import seaborn as sns # v0.13.2
import plotly.express as px
import folium

from folium.plugins import HeatMap
```

```
print(f"pandas: {pd.__version__}") # v2.2.2
print(f"NumPy: {np.__version__}") # v1.26.4
print(f"Seaborn: {sns.__version__}") # v0.13.2
```

```
pandas: 2.2.2
NumPy: 1.26.4
Seaborn: 0.13.2
```

```
# v3.9.1
!pip show matplotlib
```

Librairies Python

Elles sont installées avec la commande
« **!pip install** »

Puis activé avec l'argument
« **import** »

Par convention certaines librairies ont des
« alias (as) »
pour faciliter leurs utilisations comme :
Pandas as pd
Ou
Numpy as np

J'indique également les versions actuelles de mes librairies au cas une des versions plus récentes empêcherai le code de fonctionner.
Soit avec **__version__**
Ou
!pip show

2A - Importation des données

```
# Chargement des données
data = pd.read_csv("C:\\Users\\icema\\OpenClassrooms\\AI_Engineer\\.venv2\\p2-arbres-fr.csv", sep=";")
```

```
# Affichage des 5 premières lignes
data.head()
```

	id	type_emplacement	domanialite	arrondissement	lieu	id_emplacement	libelle_francais	genre	espece	circonference_cm	hauteur_m	stad
0	99874	Arbre	Jardin	PARIS 7E ARRDT	MAIRIE DU 7E 116 RUE DE GRENELLE PARIS 7E	19	Marronnier	Aesculus	hippocastanum	20	5	
1	99875	Arbre	Jardin	PARIS 7E ARRDT	MAIRIE DU 7E 116 RUE DE GRENELLE PARIS 7E	20	If	Taxus	baccata	65	8	
2	99876	Arbre	Jardin	PARIS 7E ARRDT	MAIRIE DU 7E 116 RUE DE	21	If	Taxus	baccata	90	10	

```
# Affichage des 5 dernières lignes
data.tail()
```

	id	type_emplacement	domanialite	arrondissement	lieu	id_emplacement	libelle_francais	genre	espece	circonference_cm	hauteur_m	stade_
198874	2023464	Arbre	Alignement	PARIS 20E ARRDT	RUE DU GENERAL NIESSEL	202002	Chêne	Quercus	cerris	20	5	
198875	2023465	Arbre	Alignement	PARIS 20E ARRDT	RUE DU GENERAL NIESSEL	202003	Chêne	Quercus	cerris	20	5	
198876	2023466	Arbre	Alignement	PARIS 20E ARRDT	RUE DU GENERAL NIESSEL	202004	Chêne	Quercus	cerris	20	5	
198877	2023467	Arbre	Alignement	PARIS 20E ARRDT	RUE DU GENERAL NIESSEL	202005	Chêne	Quercus	cerris	20	5	

Le jeu de données

Data = pd.read_csv()
C'est avec cette argument que je vais importer le fichier csv qui nous a été fourni

Data.head()
Cette fonction permet d'afficher les 5 premières lignes du jeu de données

Data.tail()
Afficher les 5 dernières lignes du jeu de données

Ces deux derniers codes nous donnent un aperçu rapide du DataFrame, cela permet de vérifier si du texte est présent par exemple.

2C - Information sur le DataFrame

```
# Affichage des informations du DataFrame
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_addresse  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
```

2D - Résumé statistique

```
data.describe()
```

	id	numero	circonference_cm	hauteur_m	remarquable	geo_point_2d_a	geo_point_2d_b
count	2.001370e+05	0.0	200137.000000	200137.000000	137039.000000	200137.000000	200137.000000
mean	3.872027e+05	NaN	83.380479	13.110509	0.001343	48.854491	2.348208
std	5.456032e+05	NaN	673.190213	1971.217387	0.036618	0.030234	0.051220
min	9.987400e+04	NaN	0.000000	0.000000	0.000000	48.742290	2.210241
25%	1.559270e+05	NaN	30.000000	5.000000	0.000000	48.835021	2.307530
50%	2.210780e+05	NaN	70.000000	8.000000	0.000000	48.854162	2.351095
75%	2.741020e+05	NaN	115.000000	12.000000	0.000000	48.876447	2.386838
max	2.024745e+06	NaN	250255.000000	881818.000000	1.000000	48.911485	2.469759

2E - Structure du DataFrame

```
nb_lignes, nb_colonnes = data.shape
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
```

```
Nombre de lignes : 200137
Nombre de colonnes : 18
```

Le jeu de données

Data.info()

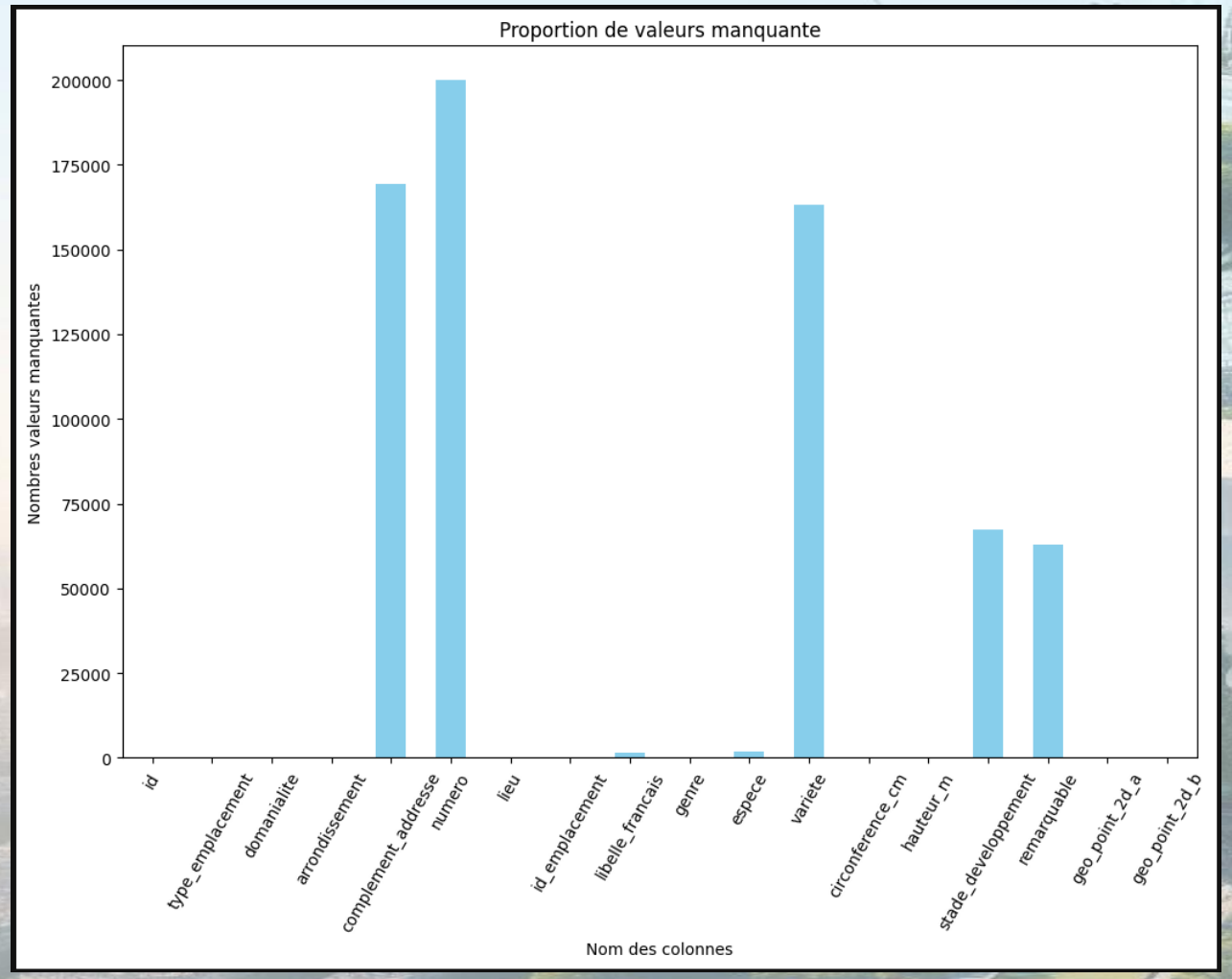
Fourni un résumé sur le contenu du DataFrame. Nombre de lignes, nombre de colonnes et leurs noms, le nombre de valeurs manquantes (NaN), le type de données (int, float) et l'estimation de la mémoire utilisée.

Data.describe()

Ce code calcul automatiquement plusieurs statistiques sur les colonnes numérique du dataset. Cela permet de repérer d'éventuel anomalies.

Data.shape

Permet d'obtenir les dimensions du jeu de données. Pour une meilleure lecture je l'ai inclus dans un « print() »




```
Il manque 200137 valeurs, soit 100.0 % dans la colonne numero
Il manque 169235 valeurs, soit 84.56 % dans la colonne complement_addresse
Il manque 163360 valeurs, soit 81.62 % dans la colonne variete
Il manque 67205 valeurs, soit 33.58 % dans la colonne stade_developpement
Il manque 63098 valeurs, soit 31.53 % dans la colonne remarquable
```

Les valeurs nulles

Il y a de nombreuses valeurs manquantes dans certaines colonnes.

Par la suite je supprimerai les colonnes « numéro », « complement_addresse » et « variete» car il manque plus de 80% des données.

J'ai conservé les colonnes « stade_développement » et « remarquable » car elles disposent de suffisamment d'informations pour notre analyse.

4A - Les doublons

```
# Vérification des doublons
doublon_id = data["id"].duplicated().sum()
if doublon_id == 0:
    print(f"Il y a {doublon_id} doublon dans la colonne id")
else:
    print(f"Il y a {doublon_id} doublons dans la colonne id")
```

```
Il y a 0 doublon dans la colonne id
```

```
doublon_gps = data[["geo_point_2d_a", "geo_point_2d_b"]].duplicated().sum()
if doublon_gps == 0:
    print(f"Il n'y a {doublon_gps} doublon dans les colonnes geo_point_2d_a et geo_point_2d_b")
else:
    print(f"Il y a {doublon_gps} doublons dans les colonnes geo_point_2d_a et geo_point_2d_b")
```

```
Il y a 11 doublons dans les colonnes geo_point_2d_a et geo_point_2d_b
```

```
gps_doublons = data[data.duplicated(["geo_point_2d_a", "geo_point_2d_b"], keep = False)].sort_values(by = "geo_point_2d_a")
gps_doublons

# Exportation en fichier excel
#gps_doublons.to_excel("Doublons_GPS.xlsx")
```

	id	type_emplacement	domanialite	arrondissement	complement_addresse	numero	lieu	id_emplacement	libelle_francais	genre	espe
185480	2006188	Arbre	Alignement	BOIS DE VINCENNES	NaN	NaN	ROUTE DAUPHINE	402030	Tilleul	Tilia	platyphyll
185479	2006187	Arbre	Alignement	BOIS DE VINCENNES	NaN	NaN	ROUTE DAUPHINE	402029	Tilleul	Tilia	platyphyll
189134	2011523	Arbre	Jardin	BOIS DE VINCENNES	NaN	NaN	PARC FLORAL DE PARIS / ROUTE DE LA PYRAMIDE	190042	Peuplier	Populus	nig
189133	2011522	Arbre	Jardin	BOIS DE	NaN	NaN	PARC FLORAL DE PARIS /	190042	Peuplier	Populus	nig

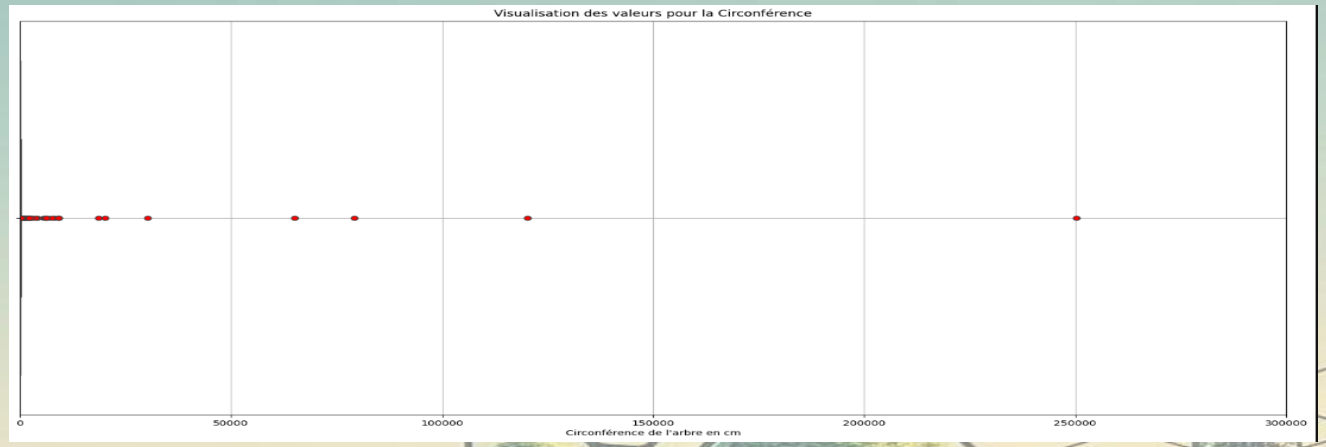
Les doublons

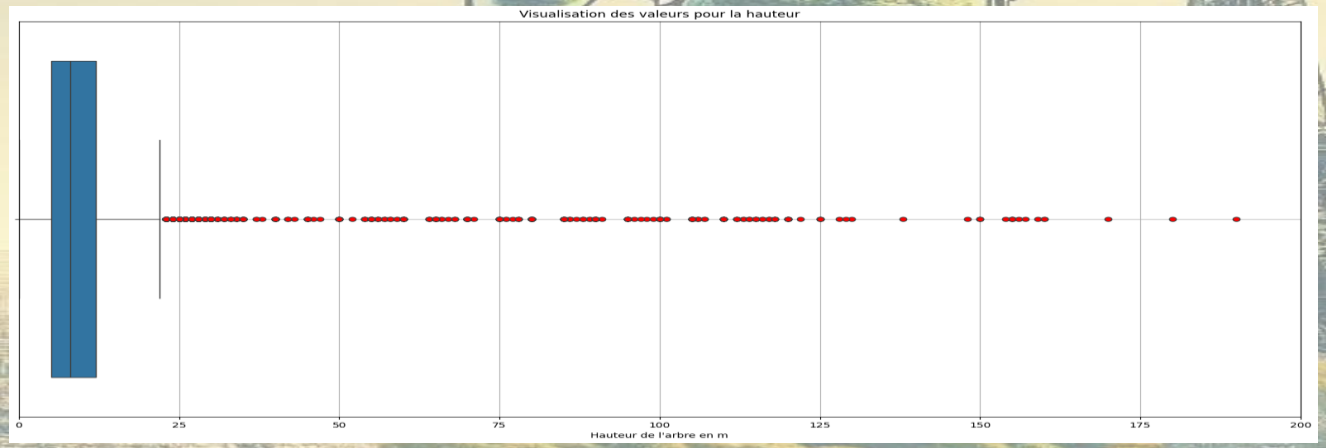
Chaque ID correspond à un arbre et comme on le constate il n'y a aucun doublon.

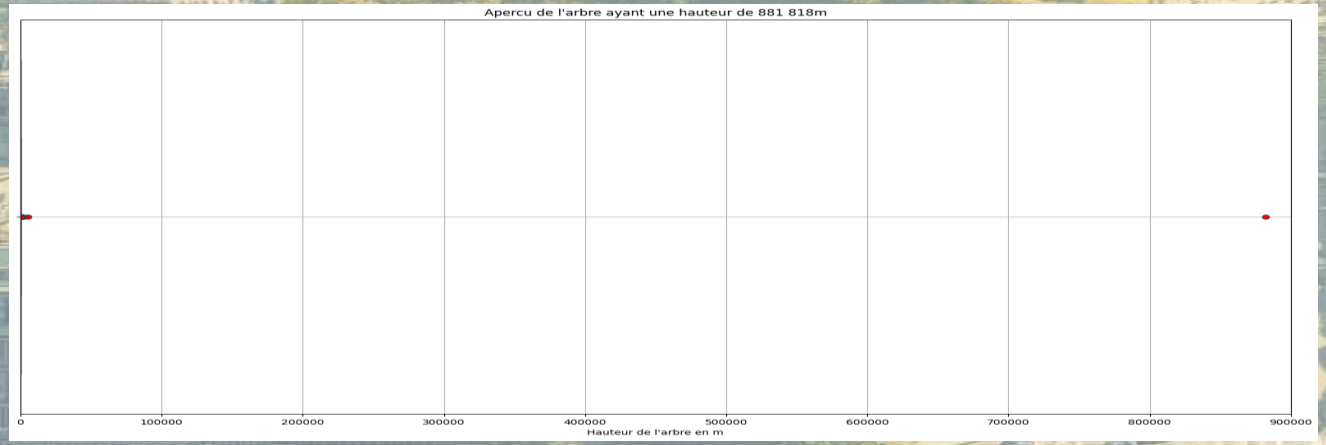
J'ai trouvé 11 doublons en vérifiant les coordonnées GPS.

Après une exportation en fichier Excel, j'ai choisi de ne pas supprimer les doubles car sans une vérification humaine il n'est pas possible d'avoir une information fiable.

- Les GPS traditionnel ont une précision de 5 à 10m deux arbres peuvent donc être à proximité l'un de l'autre.
- Cela peut être due à une erreur de saisie (remplacement d'un arbre par exemple)
- Un arbre qui aura plusieurs troncs







Gestion des outliers

(Valeurs aberrantes)

Les boxplot (diagrammes en boite) sont des outils de visualisation de données qui nous permet d'identifié ces valeurs

À l'aide du code ax.set(xlim=(0,300000)) que l'on peut ajuster avec les information fournis par le describe(), j'ai découvert un arbre avec une circonférence de 2,5 km dans le premier graphique.

Le second graphique nous montre que la majorité des arbres ont une hauteur comprise entre 25 et 125 mètres

Dans le troisième graphique on voit un marqueur rouge à près de 900km pour la hauteur d'un arbre.

Je vais utiliser la méthode des interquartiles pour identifier les limites acceptables.

La Circonference des arbres

```
Q1_circonference = data["circonference_cm"].quantile(0.25)
Q3_circonference = data["circonference_cm"].quantile(0.75)
IQR_circonference = Q3_circonference - Q1_circonference
low_bound_circonference = Q1_circonference - 1.5 * IQR_circonference
up_bound_circonference = Q3_circonference + 1.5 * IQR_circonference # Ne sera pas utilisée mais reste disponible
```

```
data.loc[data["circonference_cm"] < low_bound_circonference, "circonference_cm"] = data["circonference_cm"].median()
data.loc[data["circonference_cm"] > 700, "circonference_cm"] = data["circonference_cm"].median()
```

Hauteur des arbres

```
Q1_hauteur = data["hauteur_m"].quantile(0.25)
Q3_hauteur = data["hauteur_m"].quantile(0.75)
IQR_hauteur = Q3_hauteur - Q1_hauteur
low_bound_hauteur = Q1_hauteur - 1.5 * IQR_hauteur
up_bound_hauteur = Q3_hauteur + 1.5 * IQR_hauteur # Ne sera pas utilisée mais reste disponible
```

```
data.loc[data["hauteur_m"] < low_bound_hauteur, "hauteur_m"] = data["hauteur_m"].median()
data.loc[data["hauteur_m"] > 40 , "hauteur_m"] = data["hauteur_m"].median()
```

Les valeurs à 0

```
median_hauteur = data[data["hauteur_m"] != 0]["hauteur_m"].median()
median_circonference = data[data["circonference_cm"] != 0]["circonference_cm"].median()
print("hauteur médiane en mètre :", median_hauteur)
print("circonférence médiane en cm :", median_circonference)
```

```
hauteur médiane en mètre : 10.0
circonférence médiane en cm : 80.0
```

```
data.loc[data["hauteur_m"] == 0, "hauteur_m"] = median_hauteur
data.loc[data["circonference_cm"] == 0, "circonference_cm"] = median_circonference
```

Gestion des outliers

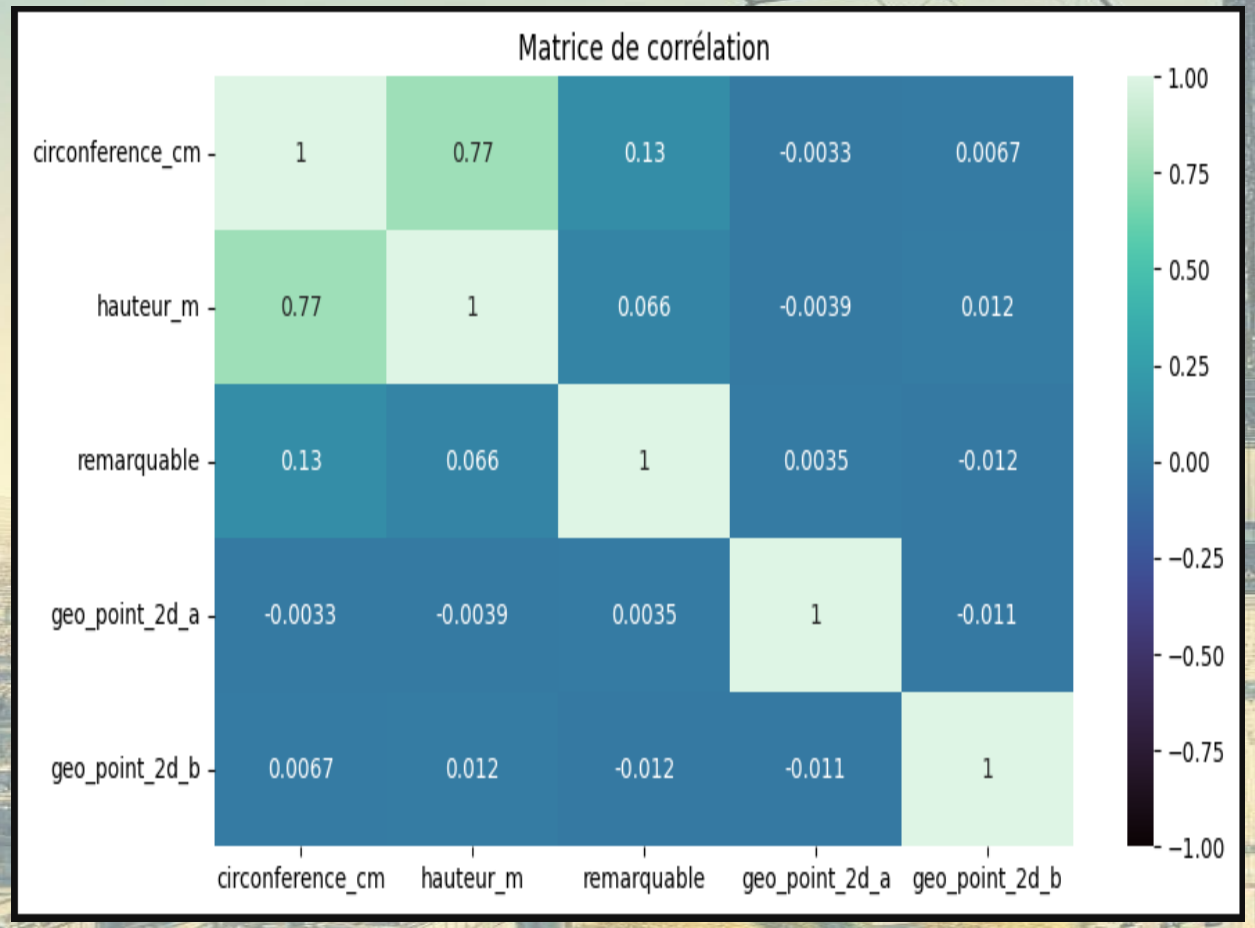
(Valeurs aberrantes)

Ici j'ai nettoyé les données, en traitant les valeurs aberrantes et les valeurs nulles*[1].

J'ai utilisé l'IQR pour détecter et corriger les outliers et remplacé les valeurs nulles*[2] par des médianes pour améliorer la qualité de mes données afin de poursuivre mon analyse.

*[1] J'ai utilisé la médiane à la place de la moyenne car celle-ci n'est pas affectée par les valeurs extrêmes

*[2] Remplacé les valeurs nulles par la médiane peut introduire un biais (valeurs erronées) dans les données.
Etant donné que cette analyse vise à améliorer l'entretien des arbres, il y a plus d'avantage à les intégrer qu'a les supprimer



La matrice de corrélation

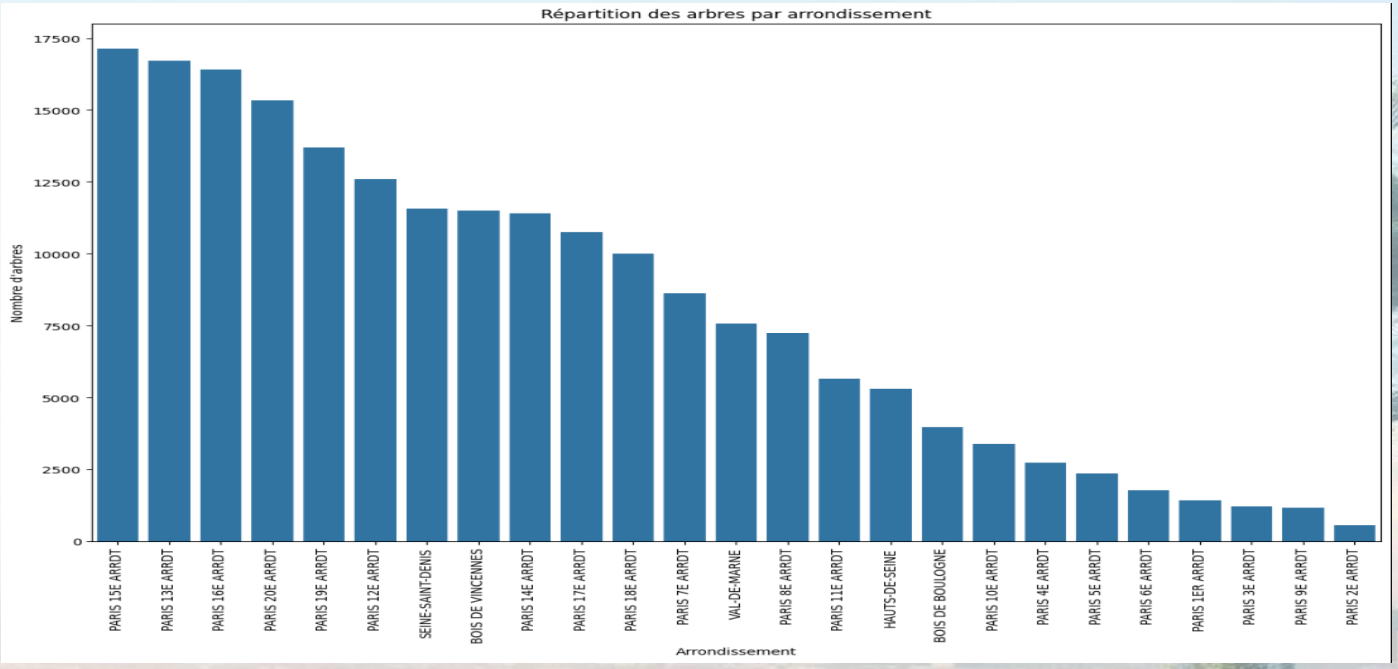
(Relation entre les variables)

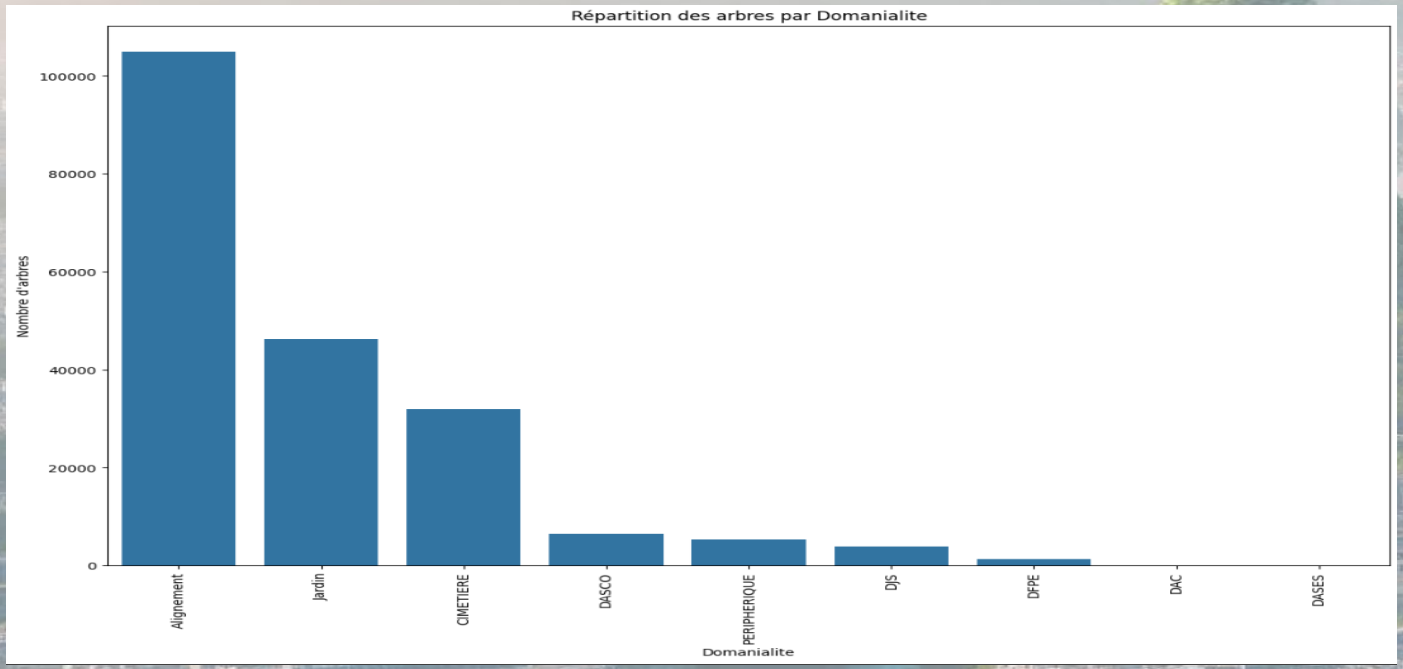
Cette matrice nous montre qu'il existe une forte corrélation positive (0,77/1) entre la circonférence et la hauteur des arbres.

Une corrélation positive indique que les deux variables évoluent dans le même sens.

Si la hauteur augmente la circonférence augmentera également.

Dans le contexte des arbres, cela nous aide à comprendre les relations entre les différentes caractéristiques.





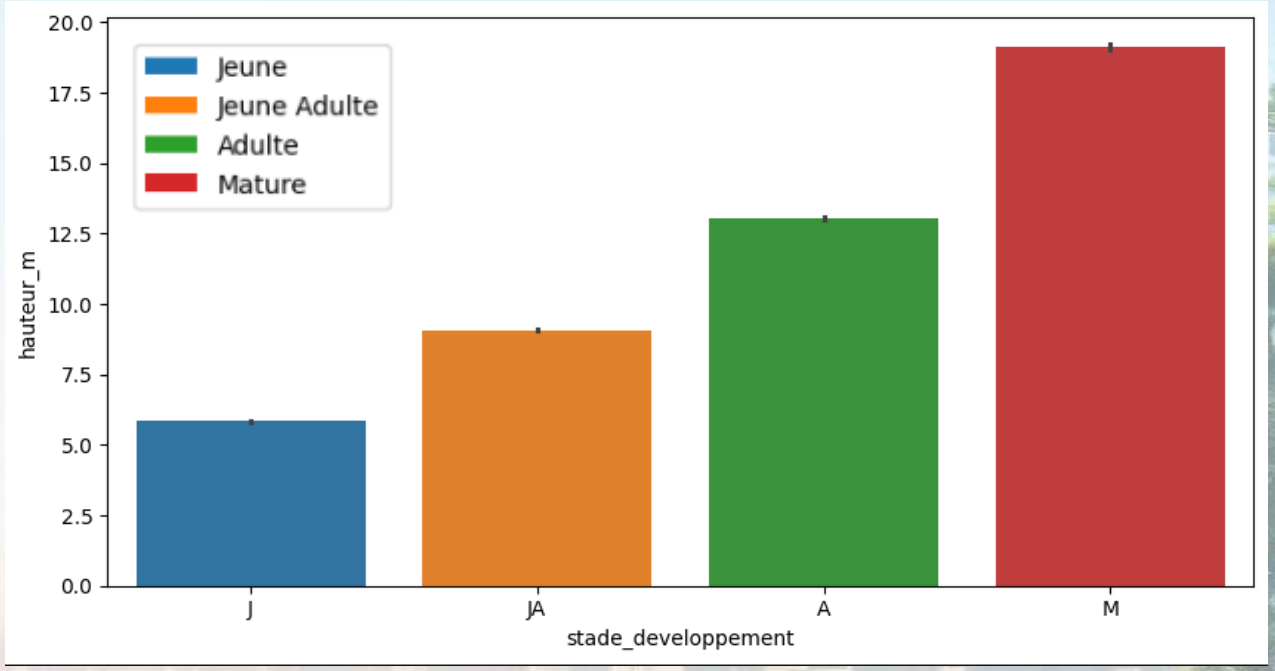
Exploration des données
(Les arbres de Paris)

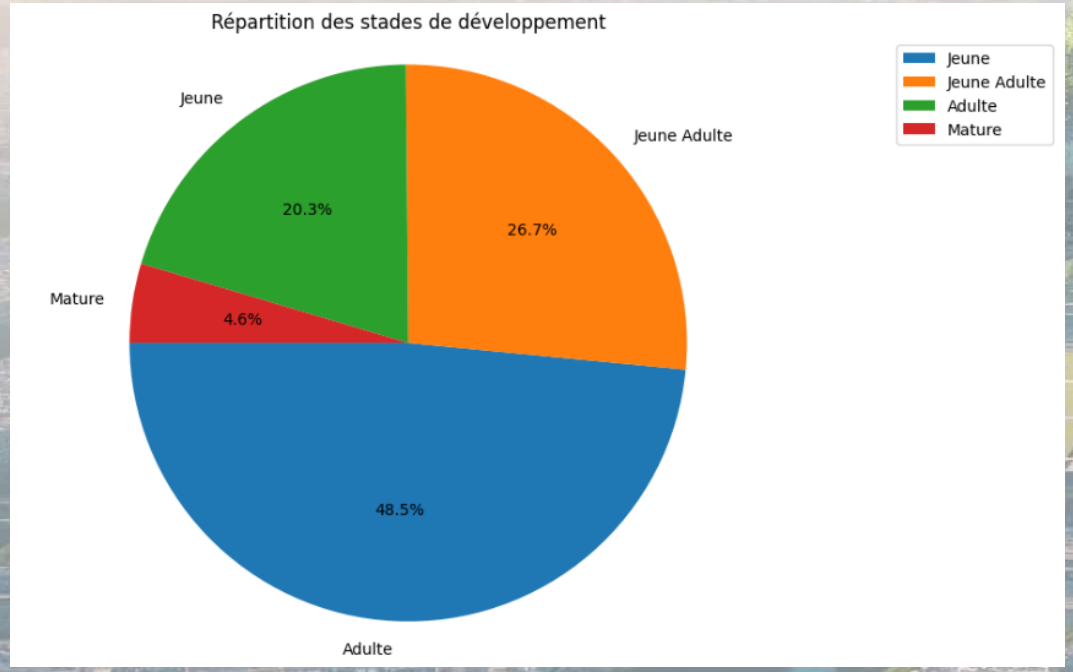
Dans le premier graphique, nous observons la répartition par arrondissement des arbres.

Le 15ème arr. est le plus verdoyant, à contrario le 2e arr. est le moins boisé.

Dans le second graphique, on voit nettement que la plus grande partie des arbres ce trouvent dans l'alignement des routes.

Cela permet par exemple d'améliorer l'esthétique urbain, améliorer la qualité de l'air ou encore aider à la réduction des îlots de chaleur.





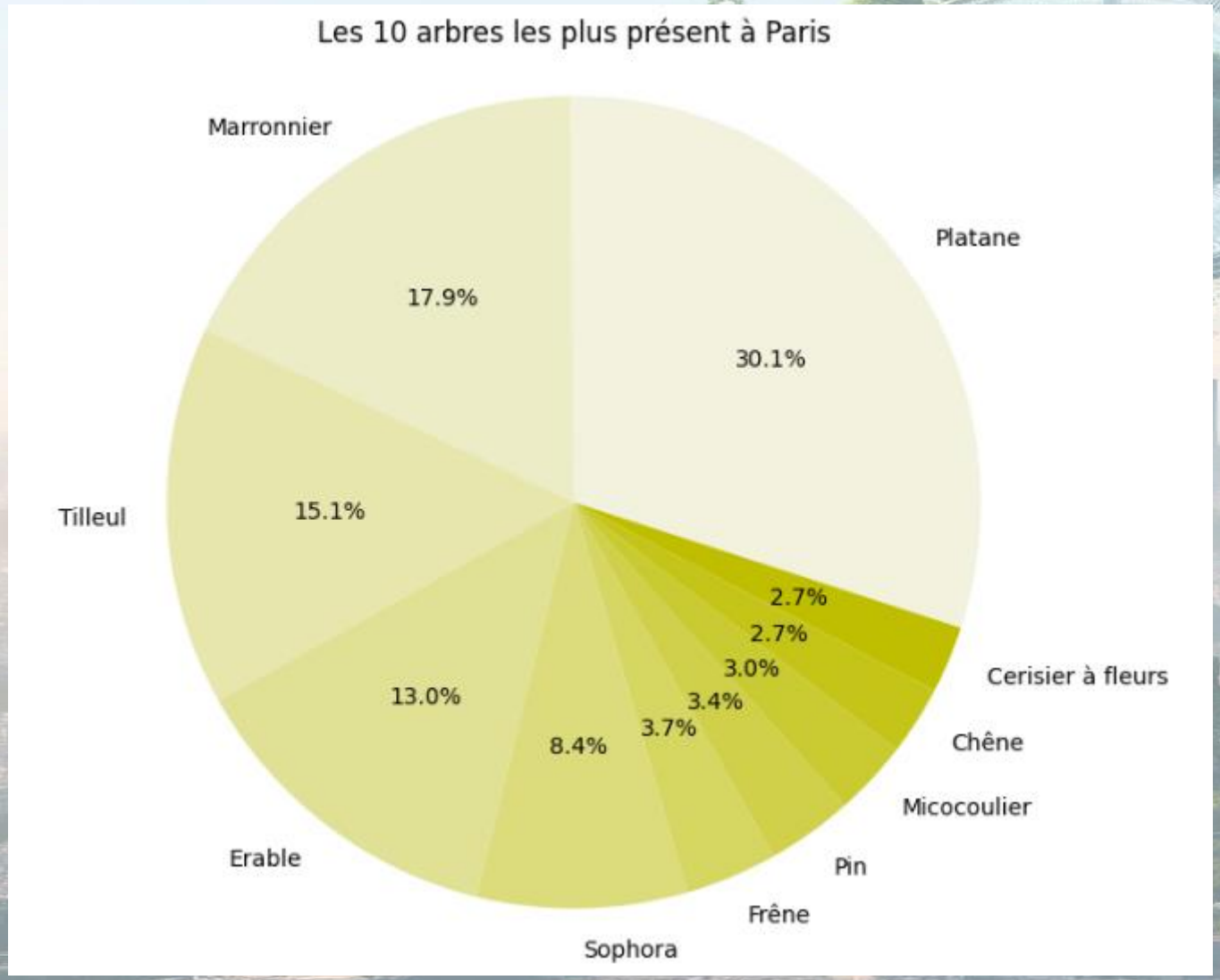
Exploration des données

(Les stades de développement)

Ces graphiques nous communiquent des informations sur les stades de développement des arbres à Paris

Le premier graphique nous montre les moyennes des hauteurs pour chaque stade de développement.

Le second nous donne la répartition des stades de développement entre les différents types de croissances des arbres.



Exploration des données
(Les essences)

Plus de 30% des arbres à Paris sont des Platane, suivi par des marronniers avec 17,9%, les tilleuls représentent 15,1% et 13% pour les érables.

Cela ne concerne que les 10 types d'arbres les plus représenté, d'autre avec des pourcentage plus faible existe.

Nous n'avons pas l'information mais cette sélection d'arbres peut-être le résultat d'un souhait de la ville pour l'esthétique, leurs acclimatations à l'environnement ou encore pour des raisons économiques.

Les arbres remarquables

La ville de Paris compte 184 arbres dit « remarquable », ces arbres peuvent soit être imposant par leurs hauteur (40m), leurs circonférences (7m) ou encore leurs âges.

Sur cette carte on peut voir ou ils se situe dans la ville Parisienne, représenté par un point orange.

Les points bleus représentent les autres arbres qui pour l'instant ne sont pas encore dans cette catégorie.

Synthèse

Cette analyse permet de connaitre de manière géographique ou se situe les arbres, leurs stades de développement, leurs caractéristiques (circonférence, hauteur....) et la répartition des arbres par domanialités.

Ceci n'est qu'une première étape, afin maximiser le travail des agents de la ville de Paris ainsi que l'optimisation de leurs trajets il nous manque les informations concernant les centres d'où partiront les agents, le matériel qu'ils ont à disposition ou encore le nombre d'employées.

En suivant les informations présente dans cette synthèse un chef d'équipe pourrait veiller à ce que les agents partent avec le matériel adéquat (comme pour l'entretien d'un arbre de 40m), organiser les tournées (ex Equipe "A" entretien des jeunes arbres, "B" des arbres jeune adulte, ect...) et favoriser la diversité des arbres, afin d'augmenter le ratio que nous avons vu plus haut (les platanes représentent +30% des arbre à Paris).

Grace a cette analyse on s'aperçoit que le 15ème arr. est le plus verdoyant a contrario le 2ème est l'arrondissement le moins boisé.

il aurait été possible d'approfondir le nettoyage en travaillant sur les valeurs nan avec des algorithmes (knn ou encore une régression par exemple) mais ce n'était pas l'objectif de ce projet.

On peut tirer énormément d'informations provenant des jeux de données, mais il faut également apprendre à rester dans les demandent des interlocuteurs afin de ne pas se perdre dans des données trop complexes pour les non-initiés


Merci