Nom prénom : TAILLEBOIS Nicolas

Nom prénom : RIOU Margot

Nom prénom : AZEAU Julia

Classe : CIR

Module : Big Data

Responsable du module : Nesma SETTOUTI & Nadine ABDALLAH-SAAB

Titre du document : Rapport projet Big Data

Date : Vendredi 21 juin 2024

Nombre de mots : 3142 mots

- ☑ En adressant ce document à l'enseignant, je certifie que ce travail est le mien et que j'ai pris connaissance des règles relatives au référencement et au plagiat.

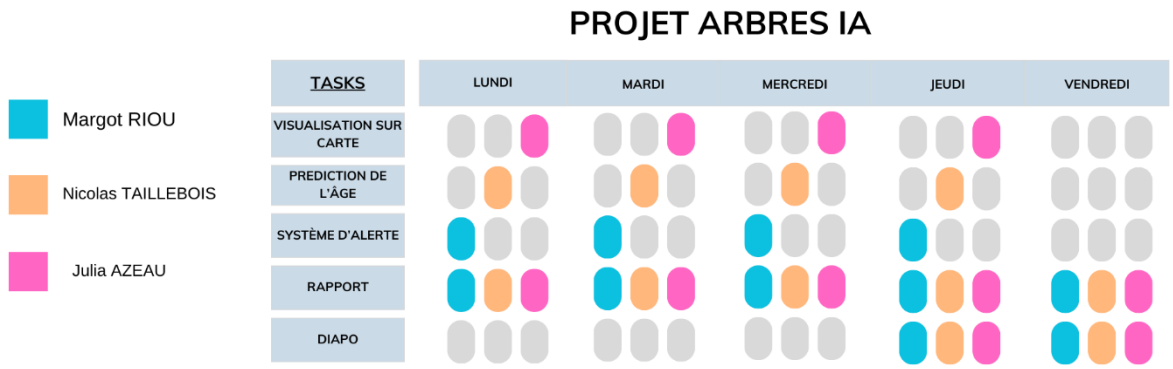
Sommaire

Introduction

Ce rapport va traiter de notre projet de d'IA à partir d'une base de données contenant les informations sur des arbres dans la ville de Saint-Quentin. Il s'agit de la même base de données que nous avions nettoyée lors du projet de Big Data.

Ce projet se divisait en trois parties différentes, la première consistait à la visualisation des données sur une carte en y affichant également les anomalies. La deuxième, dans la continuité à ce que nous avons produit la semaine dernière, nous invitait à tester différentes méthodes afin de prédire le plus précisément possible l'âge des arbres. La troisième avait pour but la prédiction du risque de voir un arbre déraciné à partir des informations de la base.

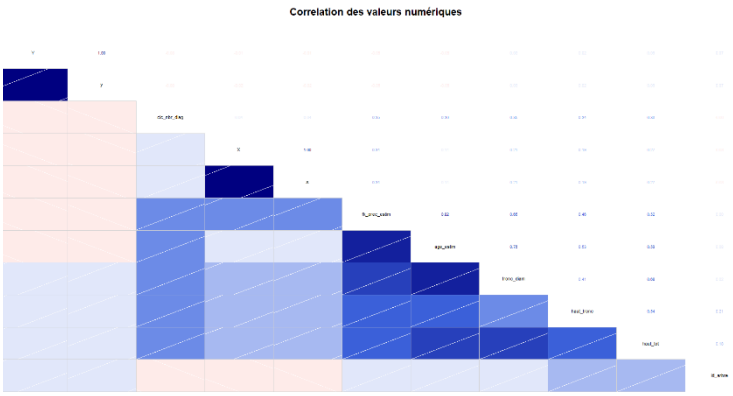
Diagramme de Gantt



Fonctionnalité 1 : Visualisation sur carte

1. Sélection des données

Nous avons deux catégories parmi les données choisies, celle pour afficher sur la carte les hauteurs d'arbres. On va donc garder la position x et y ainsi que la hauteur totale des arbres.



Et celle pour la réalisation des anomalies. Nous avons donc choisi : l'âge estimé de l'arbre, le diamètre du tronc, et la hauteur du tronc.
Nous avons pu choisir ces variables avec l'ancien graphique de corrélation, qui montre qu'elles ont un lien important avec la hauteur totale de l'arbre.

2. Fonctionnement des clusterings

Le script final prend en entrée le nombre de clusters, on peut donc facilement privilégier les méthodes de clustering prenant un nombre de clusters en entrée :

- **Kmeans :**

Place aléatoirement k clusters, à chaque tour il associe les données avec le cluster le plus proche, puis place les clusters à la position moyenne de leurs données associées. Si l'itération précédente est similaire à la nouvelle alors Kmeans s'arrête.

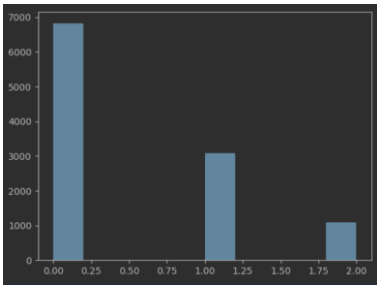
- **Spectral :**

Créé une matrice de similarité (matrice qui calcul la similarité entre chaque paire dans le jeu de données), puis spectral calcul les vecteurs propres associés à la matrice, et applique ensuite Kmeans ou un autre clustering sur les vecteurs propres pour les regrouper en cluster selon leurs similarités.

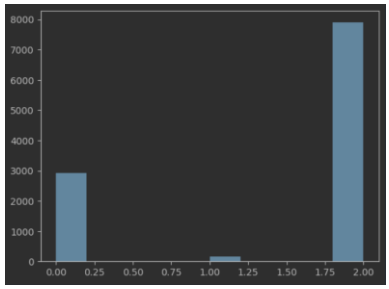
- **Agglomérative :**

Fait partie des méthodes de clustering hiérarchique.
Considère chaque donnée comme un cluster, et avec une des méthodes de calcul possible regroupe les données dans un même cluster.
Pour cet algorithme on a pu utiliser plusieurs méthodes de calcul mais la meilleure en termes de métriques reste celle par défaut : « euclidien ».

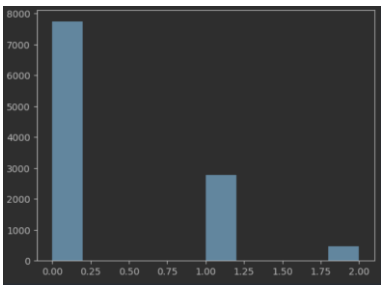
On trace le nombre d'arbres stocké dans chaque cluster :



KMeans



Spectral



Agglomérative

Nous remarquons que spectral découpe de manière assez inégale, si on prend par exemple le cluster numéro 1, il contient que très peu d'arbre.

3. Recherche du nombre optimal de clusters (k)

Pour chaque modèle on peut trouver quel est le nombre optimal de clusters pour nos données.

- **Pour le KMeans il existe la méthode Eblow (ou méthode coude)** :

L'objectif est de calculer l'inertie du modèle pour chaque nombre de clusters différents (exemple 2 à 10), qui correspond à la somme des distances des données par rapport au centroïde, le tout au carré. On peut alors tracer un graphique coude où au point de changement vertical horizontal la valeur corresponds au meilleur nombre de clusters.

- **Pour Spectral** :

Après une tentative de calcul de valeurs propres s'appuyant sur une matrice Laplacienne normalisée qui a échoué nous avons décidé de sans tenir au résultat de la métrique silhouette.

- **Pour Agglomérative on peut se servir d'un dendrogramme :**

Nous avons réalisé un dendrogramme avec la librairie « sch » issue de « plotly.figure_factory._dendrogram ». Il existe plusieurs méthodes pour établir les liens (average, centroid, ward...), Nous avons utilisé ward c'est une des méthodes les plus répandue, elle minimise la variance au sein des clusters, ce qui nous permet une meilleure similarité dans

les éléments regroupés. On peut alors compter le nombre de ligne vertical en partant de la plus longue sans couper une ligne horizontale, pour obtenir le nombre optimal de cluster.

Les (k) sont pour chaque modèle :
KMeans : 3 / SpectralClustering : 2 / AgglomerativeClustering : 2
Nous remarquons que les 3 clustering ont des (k) qui correspondent à ce que nous souhaitons c'est-à-dire soit 2 soit 3 clusters.

4. Choix du model

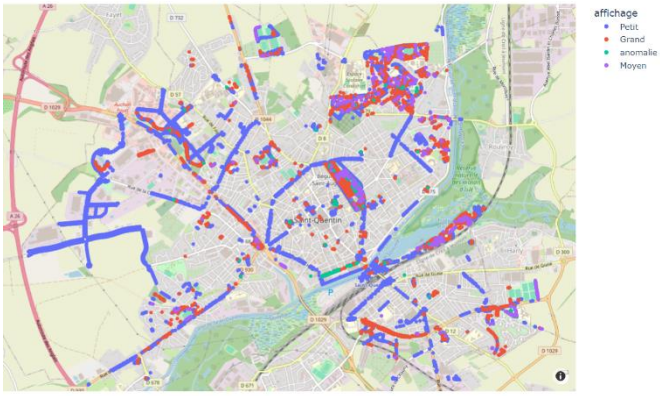
Pour le choix du modèle on peut donc comparer les différents modèles :

MODEL	SCORE POUR K = 2	SCORE POUR K = 3	VISUALISATION : GRAPHIQUE/CARTE	TEMPS (EN S)
KMEANS	SILHOUETTE : 0.6559 DAVIES BOULDIN : 0.4947 CALINSKI : 26264	SILHOUETTE : 0.6309 DAVIES BOULDIN : 0.5056 CALINSKI : 29654	✓	POUR K=3 : 0.0069 POUR K=2 : 0.0059
SPECTRAL	SILHOUETTE : 0.6527 DAVIES BOULDIN : 0.4936 CALINSKI : 25394	SILHOUETTE : 0.5169 DAVIES BOULDIN : 0.4291 CALINSKI : 14261	✗	POUR K=3 : 31.2660 POUR K=2 : 34.4380
AGGLOMÉRATIVE	SILHOUETTE : 0.6559 DAVIES BOULDIN : 0.4947 CALINSKI : 26264	SILHOUETTE : 0.6189 DAVIES BOULDIN : 0.4563 CALINSKI : 21189	✓	POUR K=3 : 1.8889 POUR K=2 : 1.4831

"On voit que les scores pour K=2 sont identiques pour Agglomérative et Kmeans. Cependant, pour K=3, Kmeans a de meilleures métriques, ainsi qu'un temps d'exécution beaucoup plus rapide. En combinant ces conclusions avec le tableau récapitulatif, on peut conclure que le meilleur algorithme de clustering dans notre cas est Kmeans."

5. Détection des anomalies

En premier lieu l'idée qui nous est venue est de réaliser une limite inférieure et une supérieure à l'aide d'un écart interquartile, pour repérer les anomalies qui dépasseraient les limites.
Bien sur cette méthode n'est pas du tout optimiser, mais nous avons quand même pu afficher une carte avec les anomalies trouvées.

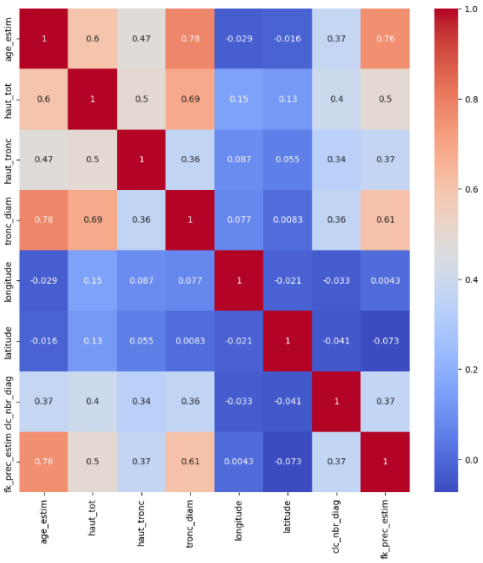


Carte avec IsolationForest et Kmeans

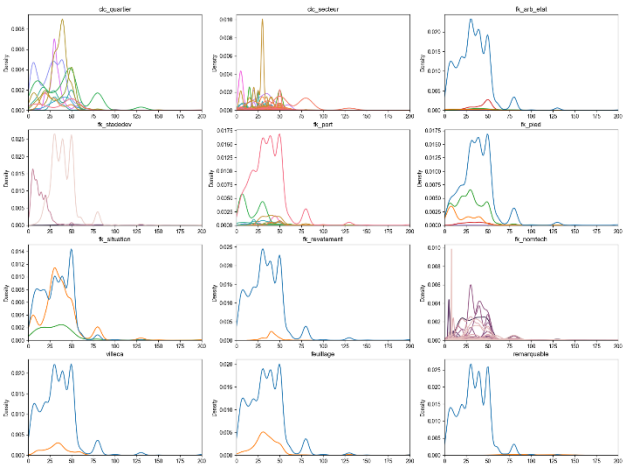
Puis nous avons réalisé un modèle d'Isolation Forest avec les 3 variables corrélé citez plus haut. L'objectif du modèle est de ranger les données dans un arbre en fonction de la distance, puis d'attribuer une valeur à chaque donnée en fonction de sa distance avec la racine. Avec une moyenne de ces valeurs attribuer l'Isolation Forest peux déterminer les anomalies dans les données.

Fonctionnalité 2 : Modèle de prédiction de l'âge

1. Préparation des données



Avant de pouvoir utiliser les différents modèles de régression linéaire pour essayer de prédire l'âge des arbres, il faut sélectionner les données à utiliser et les préparer.



Nous avons donc sélectionné 'haut_tot', 'haut_tronc', 'tronc_diam', 'fk_stadedev', 'fk_nomtech' et 'clc_nbr_diag' car nous avons trouvé de la corrélation grâce aux diagrammes ci-dessus.

Ensuite nous avons encodé 'fk_nomtech' et 'fk_stadedev' pour les transformer en nombre et donc pouvoir les utiliser dans les régressions. Puis nous avons standardisé les valeurs pour augmenter la précision des estimations.

2. Modèles de prédiction

Pour prédire l'âge des arbres, nous avons utilisé les 4 modèles suivants :

RandomForestRegressor : Ce modèle utilise un certain nombre de modèles CART sur des échantillons aléatoires du dataset et retourne la moyenne des résultats. Nous avons aussi implémenté une version from scratch de ce modèle.

Résultats avec sklearn :

```
Temps d'execution : 9.093828678131104
RMSE : 0.4327165268692333
R^2 Score: 0.8113270254569768

Cross val score :0.7659478759465259
```

Résultats from scratch :

```
Temps d'execution : 90.669306278228765
RMSE :0.46344388027269134
OOB Score : 0.7609023403509251
R2 Score : 0.7835801784751618
```

On peut observer que la version sklearn est plus précise et surtout bien plus rapide.

CART ou DecisionTreeRegressor : Ce modèle est un arbre de décision. Il split les valeurs en deux avec la meilleure condition qu'il a pu trouver et recommence pour chaque branche jusqu'à

ce toutes les valeurs soient ‘triées’. Pour prédire l’âge d’un arbre nouveau, il regarde dans quelle feuille il se place (toujours avec les conditions) et calcule la moyenne des valeurs dans cette feuille qui deviendra l’estimation pour cette valeur.

Résultats avec sklearn :

```
Temps d'execution : 0.18063759803771973
RMSE : 0.4889216226846457
R^2 Score: 0.7591308278162374
Cross val score :0.6866245088238874
```

Résultats from scratch :

```
Temps d'execution : 10.0456063747406
RMSE :0.5187553697630765
R2 Score : 0.7288385537470274
Cross val score :0.68653693512272
```

On peut, là aussi, voir que la version sklearn est plus précis et bien plus rapide.

PassiveAgressiveRegressor : Ce modèle fonctionne de la manière suivante :

Initialisation : Le modèle commence avec des poids initialisés (souvent à zéro).

Prédiction : Pour chaque nouvel échantillon de données, le modèle fait une prédiction basée sur les poids actuels.

Calcul de l'erreur : L'erreur de prédiction est calculée comme la différence entre la valeur prédite et la valeur réelle.

Modification des poids :

Passive : Si la prédiction actuelle est correcte ou proche de la vérité (c'est-à-dire si l'erreur est en dessous d'un certain seuil), il ne fait rien (ou presque rien) pour mettre à jour le modèle.

Aggressive : Si la prédiction actuelle est incorrecte ou loin de la vérité (c'est-à-dire si l'erreur dépasse un certain seuil), il met à jour le modèle de manière agressive pour corriger l'erreur.

Résultats :

```
Temps d'execution : 0.41785120964050293
RMSE : 0.6350654294923301
R^2 Score: 0.5936131230582657
Cross val score :0.5974440375390817
```

MLPRegressor ou MultiLayerPerceptronRregressor : Ce modèle fonctionne avec plusieurs couches de neurones où chaque neurone de la couche récupère les résultats des neurones de la couche précédente, le dernier neurone de la dernière couche renvoie la prédiction. Il utilise aussi une fonction de perte pour calculer la différence entre les valeurs prédites et les valeurs réelles et peut modifier les poids des connexions entre les neurones si la perte est trop élevée.

Résultats :

```
Temps d'execution : 43.23438858985901
RMSE : 0.4753355819442378
R^2 Score: 0.7723312717281421

Cross val score :0.7377314724998797
```

On peut observer que le temps d'exécution est plus lent que les autres modèles pour une précision plus faible que le RandomForestRegressor par exemple. Cela peut s'expliquer par le fait que ce modèle ait besoin d'une très grande quantité de données et qu'il soit très sensibles aux hyperparamètres.

Comparaison des modèles : On a pu observer que des quatre modèles utilisés, le RandomForestRegressor n'est pas parmi les modèles les plus rapides (comparé à CART par exemple), en revanche, il a été le modèle le plus précis (0.81 de r2score contre 0.77 pour le MLPRegressor par exemple)

Choix des modèles : Nous avons utilisé RandomForestRegressor et CART car nous devions les implémenter from scratch par la suite et nous voulions pouvoir comparer les différents résultats entre ces modèles.

Pour ce qui est du MLPRegressor, nous trouvions intéressant de tester un modèle de réseau de neurones pour voir comment ça fonctionnait et pour pouvoir comparer des catégories différentes de modèles.

Pour finir, nous avons trouvé le fonctionnement du PassiveAgressiveRegressor intéressant : le fait qu'il puisse s'adapter plus ou moins fortement aux résultats qu'il obtient et le fait qu'il soit optimisé pour des données en continu (ce n'est pas le cas dans ce contexte mais nous voulions comparer les résultats)

3. Choix des métriques

Pour tester la précision de nos modèles, nous avons utilisé différentes métriques.

r2_score : le coefficient de détermination, détermine la précision du modèle (entre 0 et 1, plus il s'approche de 1, plus le modèle est précis)

cross_val_score : méthode plus robuste et précise pour calculer la précision et la performance d'un modèle. Il divise les données en folds et utilise le 1er pour l'entrainement et les autres pour la prédiction.

RMSE : Erreur Quadratique Moyenne, calcule l'erreur entre les valeurs réelles et les valeurs prédites

OOB score : Cette métrique a été utilisée pour estimer la performance du RandomForestRegressor from scratch et remplace le cross_val_score. L'OOB score fonctionne de la sorte : à chaque fois que le modèle choisit ses valeurs aléatoires dans le dataset, il met les autres dans 'un sac' à côté, il fait ses prédictions pour tous ses arbres puis renvoie son résultat. A la fin du script, l'OOB score est calculé, c'est la comparaison de la moyenne des prédictions des valeurs dans le 'sac' à chaque arbre comparé aux valeurs réelles, plus elles sont proches, plus l'OOB score sera élevé.

Utilisation de GridSearch :

Pour trouver les meilleures valeurs d'hyperparamètres nous avons utilisé la fonction GridSearch. Voici les différents hyperparamètres qui ont été testés :

n_estimators : Le nombre d'arbres dans la forêt
max_depth : La profondeur maximale de l'arbre
min_samples_split : Le nombre minimum d'échantillons requis pour diviser un nœud interne
min_samples_leaf : Le nombre minimum d'échantillons requis pour être à un nœud feuille
max_features : Le nombre de features à considérer pour trouver la meilleure séparation
max_iter : Le nombre d'itérations pour lesquelles le modèle doit être entrainé
hidden_layer_sizes : Le nombre de couches et le nombre de neuronnes par couche (tuple)
random_state : La graine pour la génération de nombres aléatoires
average : Calculer la moyenne des poids des vecteurs de support

Fonctionnalité 3 : Système d'alerte pour les tempêtes

1. Préparation des Données

En effectuant des recherches nous avons pu remarquer que le type de sol au pied de l'arbre, le diamètre de son tronc, sa hauteur totale ainsi que son type de feuillage peuvent impacter le fait qu'il soit déraciné lors d'une tempête.

Nom de la variable	Raison de la sélection
Fk_arb_etat	C'est la variable cible, on cherche la proba qu'un arbre soit essouché ou risque de l'être.
Fk_pied	L'espace au pied de l'arbre peut impacter son developpement et sa résistance face aux
Tronc_diam	Le diamètre du tronc impacte son poids et en fonction de sa prise au sol il a plus ou moins de chance d'être déraciné
Haut_tot	La hauteur de l'arbre tout comme son diamètre impacte le poids de l'arbre.
Fk_stadedev	En fonction du stade de développement de l'arbre, il est plus ou moins solide et résistant
Feuillage	En fonction du type de feuillage l'arbre absorbe plus ou moins d'eau et prend par conséquent du poids lors des tempêtes.
Coordonnées x et y	En fonction de sa localisation l'arbre est plus ou moins soumis aux conditions météorologiques et augmentant ou non le risque d'être déraciné.

D'autres critères peuvent être pris en compte comme la météo ou l'environnement autour de l'arbre afin de déterminer le risque qu'a un arbre d'être déraciné lors d'une tempête

https://www.rtbf.be/article/tous-les-arbres-ne-sont-pas-egaux-devant-la-tempete-voici-pourquoi-certains-sont-plus-susceptibles-de-tomber-10430150

2. Problèmes rencontrés

Lors de la première prédiction de l'état de l'arbre nous avons pu remarquer que tous les états de l'arbre n'étaient pas pertinents pour notre étude. Nous avons donc fait le choix de les séparer de manière binaire avec d'un côté les arbres en place ou avec encore une souche et de l'autre les arbres essouchés ou supprimés. Malheureusement cette séparation nous donne 10 013 exemples d'un côté et 770 de l'autre. Car nous ne prenons pas en compte les arbres remplacés qui sont des arbres en place qui ont été remplacé. Le ratio entre les deux est donc beaucoup trop faible pour avoir un modèle fiable.

En effet, en faisant un apprentissage sur ces valeurs le cross val est bien trop élevé, qui nous indique que notre modèle n'est pas bon. Afin de rajouter des valeurs d'arbres essouché nous avons utilisé la fonction smote qui permet de créer des exemples fictifs de la feature minoritaire. A cela nous avons ajouté un GridSearch afin de déterminer le meilleur classifieur à utiliser par la suite.

Exemple d'affichage console lors des tests :

On voit que le cross val score diminue légèrement mais, c'est la matrice de confusion qui est la plus impactée, et même si les valeurs des vrai positifs et négatifs sont élevées, elle nous semble plus cohérentes avec les autres résultats.

```
Le cv score est =  [0.9931694  0.9931694  0.99043716]
La première matrice de confusion:
 [[0.         1.        ]
 [0.00228938 0.99771062]]
Le cvs 2 est :  [0.97732738 0.97852603 0.98161828]
La deuxième matrice de confusion:
 [[0.98236372 0.01763628]
 [0.02404947 0.97595053]]
```

Les modèles précédents étant toujours déséquilibré, nous avons fini par faire le choix de garder uniquement les données des arbres étant essouché ou non essouché. A partir de ce nouveau data frame nous avons pu refaire des entrainements qui donnaient des résultats plus concluant malgré la perte de données conséquente.

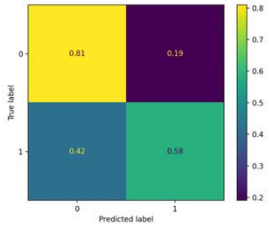
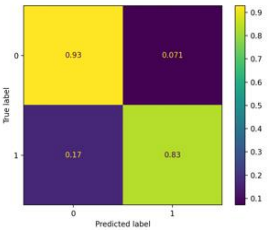
Un autre point qu'il a fallu prendre en compte est l'encodage des données, en effet une partie d'entre elle est qualitative rendant compliqué l'apprentissage et la prédiction. Les exemples étant différents pour chaque variable, il nous a semblé préférable de prévoir un encodeur par variable catégorielle en prévoyant un ordre défini dans l'organisation des variables pour éviter que les valeurs numériques attribuées aux exemples ne changent d'un encodage à l'autre. Les encodeurs comme le modèle sont envoyés sous la forme d'un dictionnaire dans un fichier pickle afin de pouvoir les réutiliser à chaque étape de prédiction ou d'entrainement.

3. Fonction principale

Concernant le script de la fonction principale nous avons choisi de garder le modèle avec le smote pour avoir des résultats moins biaisés. Ce script permet de récupérer des données avant de les retransformer via l'encodeur et de les filtrer pour garder uniquement celles qui nous intéresse pour faire une prédiction et définir les probabilités qu'un arbre soit déraciné. On affiche également en console différentes métriques permettant de montrer la fiabilité du modèle.

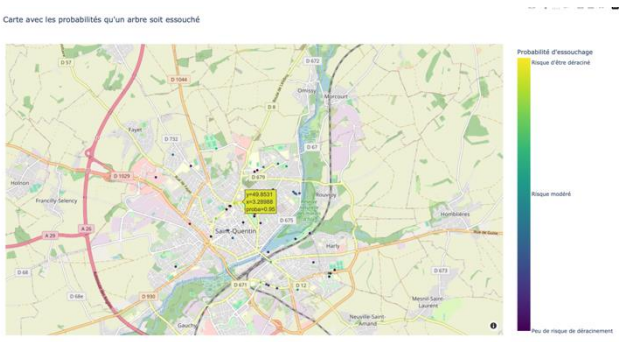
```
Le cvs est :  [0.94444444 0.55555556 0.77777778]
La matrice de confusion du cvp:
 [[0.80952381 0.19047619]
 [0.41666667 0.58333333]]
La matrice de confusion du predict:
 [[0.92857143 0.07142857]
 [0.16666667 0.83333333]]
D'autres métriques :

Accuracy score =  0.9074074074074074
Precision score =  0.9107775693141548
Recall score =  0.9074074074074074
F1 score =  0.9087014725568943
```

Métrique choisie	Interprétation des résultats
Cross val score (3)	Cette méthode nous permet d'évaluer les performances du modèle, ici on voit bien que le résultat varie beaucoup d'un essai à l'autre. Cela nous montre que le modèle peut encore être amélioré pour être plus fiable.
Matrice de confusion du cross val predict 	Cette matrice nous montre graphiquement ce que nous avons pu remarquer lors du cross val score, en général le modèle a 41% de risque de prédire un faux négatif et 58% de chance de prédire un vrai positif. Ces résultats peuvent s'expliquer par la base d'entrainement qui ne contenait pas suffisamment de valeurs pour traiter toutes les possibilités.
Matrice de confusion du y predit 	Cette matrice nous montre que la prédiction du modèle à partir des données de test est fiable car les taux de vrais positifs et négatifs sont important à contrario des taux de faux.
Accuracy score	Le taux de classification du modèle est de 90,74%, pour la prédiction.
Precision score	Quand le modèle prétend qu'un arbre a un état essouché, il est correct 91% du temps, dans le cas de la dernière prédiction.
Recall score	Le modèle détecte 90,74% des arbres essouchés comme étant bien essouché.
F1 score	La performance de la dernière prédiction est de 90,8%.

Nous avons choisi de présenter toutes ces métriques car nous avons fait face à de nombreux problèmes, ces dernières permettent de voir si un problème est présent à l'apprentissage ou lors de la prédiction.

Finalement, nous avons affiché ces résultats sur une carte en changeant la couleur en fonction du degré de risque.



Conclusion

Pour conclure, ces 3 besoins nous ont permis de prendre des décisions sur un grand nombres de données, qu'on pourrait difficilement analyser sans l'IA.

Ce type de besoin pourra servir pour la réduction des risques et l'adaptation à la météo, une meilleure répartition et compréhension de la biodiversité. Ou encore une meilleure organisation et gestion de la ville et de l'espace public…

Chaque besoin client, chaque code peut être réalisé de manière diverse, là est toute l'importance de nos choix des variables, des algos, des méthodes de prédiction…

Il faut alors connaitre le fonctionnement des modèles afin de comprendre leur porté et leur qualités et défauts. Pour ce faire, nous utilisons des métriques pour évaluer les différents tests appliquer durant le projet, telle que silhouette, cross val score, RMSE…

On pourrait imaginer par la suite ajouter de nouveaux critères pour faire évoluer nos analyses, faire appel à des experts pour collaborer et mieux comprendre certains impacts… Tout cela dans le but de partager ces informations sur site web ouvert au grand public.