Data Challenge

Estimation de valeurs manquantes dans les indicateurs ESG

ELHAGE Cyril, BREBION Matthieu

27 Novembre 2023

Table des matières

1 Introduction

1.1 Présentation du jeu de données

L'objectif de ce data challenge est de prédire les valeurs manquantes pour 15 indicateurs financiers d'entreprises (également appelées ESG pour Environnement, Social et Gouvernance). Ces 15 indicateurs sont les suivants :

Attribut	**Description**	**Valeurs manquantes (%)**
market_cap	La "taille" de l'entreprise (en $)	0%
employees	Le nombre d'employés de l'entreprise	16%
revenue	Le chiffre d'affaire annuel de l'entreprise (en $)	2%
scope_1	Les émissions scope 1 (directes) de l'entreprise (en T/CO2e)	66%
scope_2	Les émissions scope 2 (indirectes, détenues) de l'entreprise (en T/CO2e)	66%
scope_3	Les émissions scope 3 (indirectes, non détenues) de l'entreprise (en T/CO2e)	69%
waste_production	La production annuelle de déchets produits (en T)	84%
waste_recycling	La quantité annuelle de déchets recyclés (en T)	89%
water_consumption	La quantité annuelle d'eau consommée (en T)	72%
water_withdrawal	La quantité annuelle d'eau retirée (en T)	74%
energy_consumption	La consommation d'énergie électrique annuelle (en KWH)	82%
hours_of_training	Le nombre d'heures de formations dispensées pour tous les employés sur l'année (en H)	87%
gender_pay_gap	L'écart relatif entre les salaires moyen masculin et féminin (en %)	96%
independant_board_members_percentage	Le pourcentage de membres du board de l'entreprise indépendants (en %)	75%
legal_costs_paid_for_controversies	Le montant total des coûts liés à des controverses (en $)	59%
ceo_compensation	La rémunération annuelle du CEO (variable inclus, en $)	92%

TABLE 1 – Description des attributs avec leur poucentage de valeurs manquantes respectives

Ce data challenge fait partie du projet Pladifes qui est un projet de recherche visant à faciliter l'accès aux données extra-financières des entreprises. Ce projet est en collaboration avec l'entreprise Impactfull, qui fait du conseil en finance durable.

La prédiction précise des valeurs ESG manquantes revêt une importance cruciale sur le plan écologique et social. Les données ESG sont des indicateurs essentiels de l'impact environnemental, social et de gouvernance d'une entreprise. En comblant les lacunes dans ces données, nous renforçons notre capacité à évaluer l'engagement social et les pratiques environnementales d'une entreprise. Ces informations sont essentielles pour orienter les investissements vers des entreprises qui non seulement maximisent leurs performances financières, mais qui contri-

buent également positivement à l'environnement et à la société. Connaître ces données permettrait également de pouvoir faire plus de statisitiques, par exemple au niveau des déchets produits/recyclés, pour voir dans quels secteurs sont les plus gros leviers d'action potentiels.

Le jeu de données est composé de 3 fichiers .csv : X_train, y_train et X_test, et le rendu du data challenge est un fichier y_test. Les fichiers d'entrée X_train et X_test sont composées de données récoltées sur 10000 entreprises pendant 3 années (2018, 2019 et 2020), avec des données manquantes (voir pourcentage de valeurs manquantes dans le tableau 1). Le fichier de sortie y_train correspond au fichier X_train partiellement rempli. L'objectif du Data Challenge est de renvoyer un fichier de sortie y_test qui correspond au fichier X_test entièrement rempli.

1.2 Métrique utilisée

Afin de classer les différentes solutions, la métrique utilisée pour ce data challenge est l'erreur quadratique moyenne normalisée (NRMSE), qui correspond ici à l'erreur quadratique moyenne (RMSE) normalisée par la différence entre la valeur maximale et la valeur minimale (voir équations 1 et 2).

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \hat{X}_i)^2} \tag{1}$$

$$\text{NRMSE} = \frac{\text{RMSE}}{X_{\text{max}} - X_{\text{min}}} \tag{2}$$

Cette erreur est calculée pour chaque colonne avec les données complétées. Pour obtenir l'erreur finale, il faut ensuite faire la moyenne des erreurs obtenues pour chaque colonne.

2 Méthode de calculs des valeurs ESG manquantes

2.1 Méthode n°1 : Imputation Itérative avec le module IterativeImputer

La méthode la plus utilisée pour faire de l'imputation de valeurs manquantes est l'algorithme MICE (Multiple Imputation by Chained Equations). Il fonctionne en itérant à travers les variables du jeu de données, imputant séquentiellement les valeurs manquantes de chaque variable en utilisant des modèles conditionnels basés sur les autres variables observées. Ces imputations partielles sont ensuite utilisées pour mettre à jour les valeurs manquantes dans le jeu de données complet. Le processus est répété plusieurs fois pour stabiliser les imputations et générer plusieurs ensembles de données imputées, permettant de prendre en compte l'incertitude liée à l'imputation des valeurs manquantes.

La précision des valeurs imputées va principalement dépendre de deux critères : la corrélation entre les variables et le pourcentage de valeurs manquantes. Etant donné que nos valeurs sont très peu corrélées entre elles (voir figure 1), et qu'il manque en moyenne 70% des valeurs

dans chaque colonne, l'imputation itérative seule risque de ne pas donner un résultat satisfaisant.

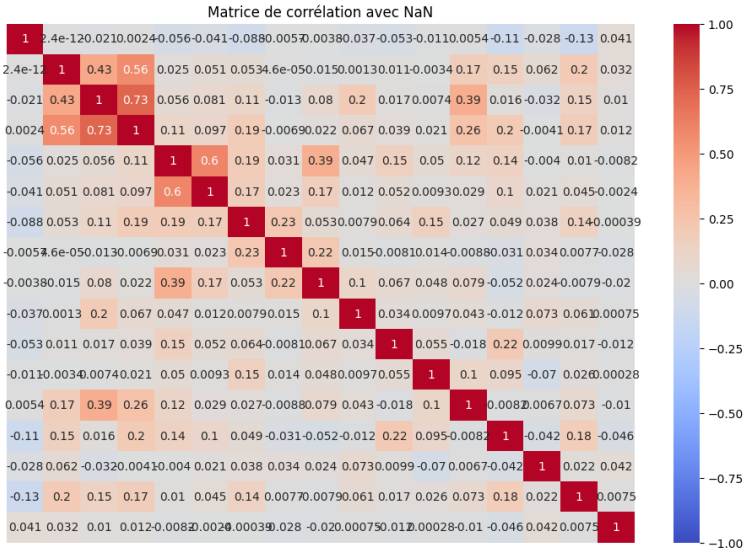

FIGURE 1 – Matrice de corrélation de X_test

Afin de mieux comprendre le fonctionnement de l'algorithme, vous pouvez voir ci-dessous un exemple issu de l'article The MICE Algorithm écrit par Sam WILSON :

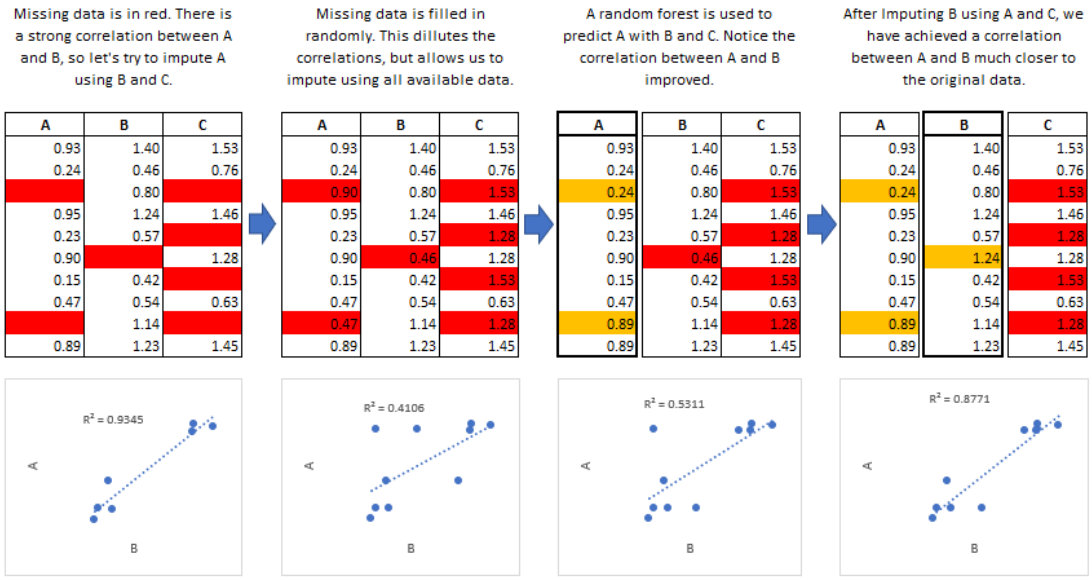

FIGURE 2 – Exemple d'application de l'algorithme MICE

Cet algorithme est accessible sur Python dans la librairie scikit-learn à l'aide de la classe IterativeImputer. En entraînant notre modèle avec les données X_train et y_train, on obtient un fichier y_test avec une erreur de **0.555**.

2.2 Méthode n°2 : IterativeImputer avec remplacement des valeurs aberrantes

2.2.1 Premier remplacement

En analysant le résultat obtenu avec IterativeImputer, on se rend compte que le module a créé des valeurs aberrantes (c'est-à-dire des valeurs négatives ou trop élevées), en accord avec ce que l'on avait prévu dans la partie 2.1 . On a fait l'hypothèse qu'une valeur est trop grande si elle se situe en dehors du 95ème centile.

Afin de résoudre ce problème, nous avons remplacé dans chaque colonne de y_test les valeurs aberrantes par la médiane de la colonne. On obtient un fichier y_test avec une erreur de **0.265**.

Cependant, cette méthode pose deux problèmes. Tout d'abord, le calcul de la médiane se fait avec les valeurs aberrantes. On obtient donc un résultat erroné. De plus, avec notre définition des valeurs trop élevées, on risque d'enlever des valeurs étant déjà présentes dans le dataset de base.

2.2.2 Calcul de la médiane sans les valeurs aberrantes

Pour résoudre le problème de la médiane, il suffit simplement de faire le remplacement en plusieurs étapes. D'abord on enlève les valeurs aberrantes, puis on calcule la nouvelle médiane et on remplace les valeurs manquantes par la médiane.

Ici, on considère que les valeurs aberrantes ne concernent que les valeurs négatives. On obtient un fichier y_test avec une erreur de **0.319**.

2.2.3 Etude des valeurs trop élevées

En analysant les valeurs en dehors du 95ème centile dans le fichier y_test créé à l'aide de la méthode 2.2.1, on se rend compte que 43% d'entre elles sont déjà présentes dans le fichier X_test. Il faut donc faire attention à ne remplacer que les valeurs créées par l'imputation itérative.

En appliquant cette méthode sur le fichier créé à la section précédente (2.2.2), en recalculant la médiane sans les valeurs en dehors du 95ème centile, on obtient un fichier y_test avec une erreur de **0.2502**.

Cependant, le choix du 95ème centile étant arbitraire, peut-être que certaines valeurs créées par l'imputation qui sont en dehors de ce centile restent légitimes. Il faudrait donc déterminer le centile optimal, en faisant des tests sur plusieurs centiles et en gardant celui avec l'erreur la plus basse. Cependant, la limitation à 2 soumissions de solution par jour nous bloque pour pouvoir tester plusieurs centiles. Nous allons donc garder notre hypothèse initiale pour le reste du projet.

2.3 Méthode n°3 : Implémentation d'une imputation

Dans cette partie, nous allons essayer d'implémenter notre propre imputation en utilisant deux modèles prédictifs différents : avec des régressions linéaires puis avec RandomForest.

Mais, avant de rentrer dans le vif du sujet, on commence par remplir les valeurs manquantes avec la médiane de la colonne correspondante. Puis on calcule la matrice de corrélation (voir figure 3).

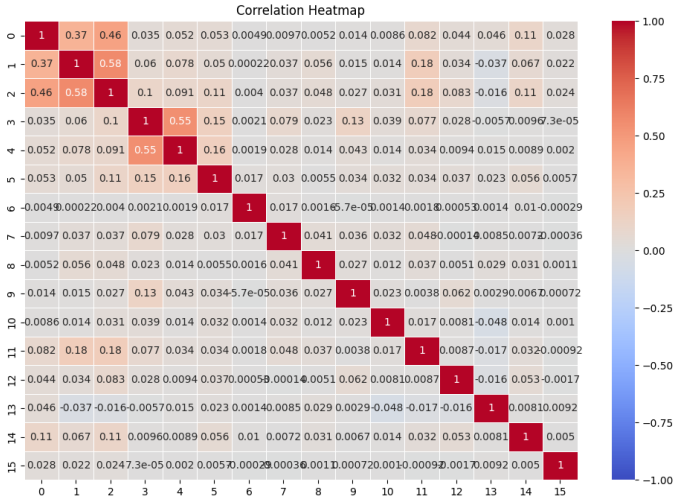

FIGURE 3 – Matrice de corrélation de X_test où les valeurs manquantes ont été remplacées par la médiane

2.3.1 Avec des régressions linéaires

Pour faire l'imputation itérative, on va parcourir chaque colonne du dataframe. Pour chaque colonne, on recupère la colonne avec l'indice de corrélation le plus élévé. C'est cette colonne qui nous servira de X pour notre régression linéaire. Ensuite, à partir de notre X_test, on crée 3 nouveaux dataframes : X_train_bis, X_test_bis et y_train_bis. Pour X_train_bis, on prend la colonne la plus corrélée à la colonne qui nous intéresse et on sélectionne les lignes sans valeurs manquantes dans la colonne à prédire. Pour y_train_bis, on sélectionne les éléments non NaN dans la colonne à prédire. Pour X_test_bis, on prend les éléments de la colonne la plus corrélée qui ne sont pas dans X_train_bis. Ensuite, on entraîne notre modèle avec X_train_bis et y_train_bis, afin de prédire y_test_bis.

Une fois toutes les colonnes parcourues, notre dataframe est entièrement rempli. Puis, pour finir, on traite les valeurs aberrantes comme expliqué précédemment dans la partie 2.2.

Finalement, on obtient un fichier y_test avec une erreur de **0.29**, ce qui est un peu moins bien qu'avec IterativeImputer, mais qui est prometteur étant donné que le modèle prédictif utilisé ici est l'un des plus basique.

2.3.2 RandomForest avec une seule feature

Afin de réduire notre erreur, nous avons remplacé la régression linéaire par un RandomForest. On parcourt toutes les colonnes et on entraîne notre modèle de manière similaire à la partie 2.3.1. Cependant, utiliser RandomForest avec une seule feature manque de sens. En effet, Dans chaque arbre individuel du Random Forest, un sous-ensemble aléatoire des caractéristiques est sélectionné pour former l'arbre. Seulement, nous limitons ici nos arbres à un seul feature. C'est certainement pour cela qu'on obtient une erreur assez conqéquente, de **0,45**.

2.3.3 RandomForest avec plusieurs features

Donc, au lieu de n'utiliser qu'une seule colonne pour la prédiction, nous utilisons RandomForest avec plus de features. On définit différemment les 3 dataframes précédent. Pour X_train_bis, on prend toutes les colonnes autres que celle que l'on veut prédire, et on séléctionne les lignes où les éléments de la colonne que l'on étudie ne sont pas NaN. Pour y_train_bis, on sélectionne tous les éléments qui ne sont pas NaN dans la colonne qui nous intéresse. Pour X_test_bis, on prend toutes les colonnes autre que celle qui nous intéresse, et on sélectionne les lignes qui ne sont pas dans X_train_bis.

Après traitement des valeurs aberrantes, on obtient un fichier y_test avec une erreur de **0,1682**.

2.3.4 RandomForest Itératif

Ensuite, afin d'améliorer cette erreur, nous avons tenté d'implémenter une imputation itérative. Le principe reste le même que pour la partie précédente (2.3.3), sauf que cette fois-ci on itère plusieurs fois le processus. A chaque iteration, on prédit à nouveau les valeurs manquantes mais en utilisant les prédictions faites à l'itération précédente. Ensuite, à la fin de l'itération, on traite les valeurs aberrantes comme expliqué plusieurs fois précédemment, afin d'éviter de faire des prédictions en utilisant des valeurs erronées.

Mais, on obtient un moins bon résultat que l'imputation non-itérative de la partie 2.3.3, avec une erreur de **0,1753**.

2.3.5 RandomForest en utilisant X_train et y_train

Cependant, nous nous sommes rendus compte que nous n'utilisions pas les données de X_train et y_train pour entraîner notre modèle. Nous avons donc concaténé X_train et y_train à X_train_bis et y_train_bis. On réutilise le code écrit dans la partie 2.3.3, qui est la méthode avec la plus faible erreur, et on obtient un fichier y_test avec une erreur de **0,1660**.

3 Conclusion

Pour conclure, la méthode la plus efficace est l'imputation avec RandomForest. Cependant, nos tentatives d'amélioration en rajoutant des itérations ou en rajoutant des données d'entraînement n'ont pas été très concluantes, avec des erreurs qui reste du même ordre de grandeur que le RandomForest classique de la partie 2.3.3 (voir tableau récapitulatif). On a donc une erreur minimale de **0,1660** pour le RandomForest utilisant les valeurs de X_train et y _train (voir partie 2.3.5).

Cependant, comme écrit dans la documentation du module, IterativeImputer est encore en phase de développement et les prédictions renvoyées peuvent changer dans le futur. Cela pourrait expliquer le fait que nous ayons obtenu des meilleurs résultats en implémentant nous-mêmes l'imputation.

4 Tableau récapitulatif des résultats

Méthode utilisée	**Erreur**
IterativeImputer 2.1	0.555
IterativeImputer 2.2.1	0.265
IterativeImputer 2.2.2	0.319
IterativeImputer 2.2.3	0.2502
Imputation RegLinéaire 2.3.1	0.29
Imputation RandomForest 2.3.2	0.45
Imputation RandomForest 2.3.3	0.1682
Imputation itérative RandomForest 2.3.4	0.1753
Imputation RandomForest 2.3.5	**0,1660**

TABLE 2 – Tableau récapitulatif des méthodes utilisées