Rapport TDM3 : Introduction aux sciences des données et à l'intelligence artificielle

Fanani Selma et Amsaf Rim (Groupe 4)

September 27, 2024

1 Etude d'un régresseur linéaire simple

Pour ce troisième TDM, nous allons évaluer l'efficacité du modèle de régression linéaire.

Nous utilisons la fonction **LinearRegression()**[2] pour ajuster une régression linéaire sur un jeu de données réel. Les données représentent le prix de vente (x) et le nombre de produits vendus (y).

Avec le modèle obtenu, nous obtenons une erreur quadratique moyenne de **121,77**, cela indique un écart relativement faible entre les valeurs prédites et réelles, ce qui suggère que le modèle a une bonne capacité à capturer la relation entre le prix de vente et le nombre de produits vendus.

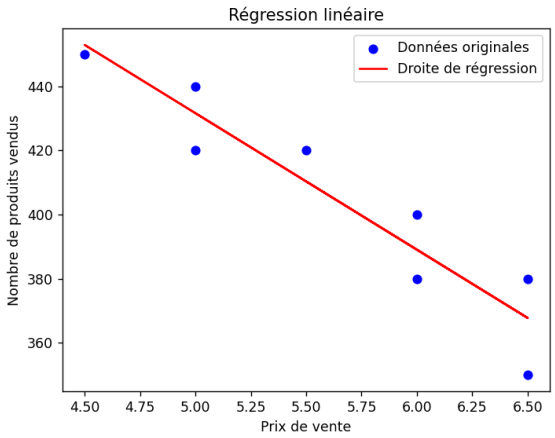
Les attributs du modèle révèlent des informations importantes :

- **coef_** (Coefficient) [−42.58] : Ce coefficient indique que pour chaque augmentation d'une unité du prix, le nombre de produits vendus diminue en moyenne de 42.58 unités. Cela correspond à l'intuition selon laquelle une augmentation des prix réduit les ventes.
- **intercept_** (Ordonnée à l'origine) 644.52 : Cela représente le nombre de produits vendus lorsque le prix est nul. Bien que cette situation n'ait pas de sens pratique (le prix ne peut pas être nul), il s'agit d'un paramètre calculé par le modèle pour optimiser au mieux la relation linéaire.

Le coefficient de détermination R^2 obtenu avec ce modèle est de 0,88. Ce score indique que 88 % de la variance des ventes peut être expliquée par le modèle de régression linéaire, ce qui est considéré comme un excellent ajustement.

En effet, un R^2 proche de 1 signifie que le modèle prédit bien les résultats, tandis qu'un R^2 proche de 0 indiquerait un modèle inefficace. Dans notre cas, ce score suggère que le régresseur linéaire s'ajuste bien aux données.

Pour visualiser les résultats, la droite de régression a été tracée en rouge sur le nuage de points des données. Ce graphique permet de voir comment le modèle s'ajuste aux valeurs observées. En réajustant

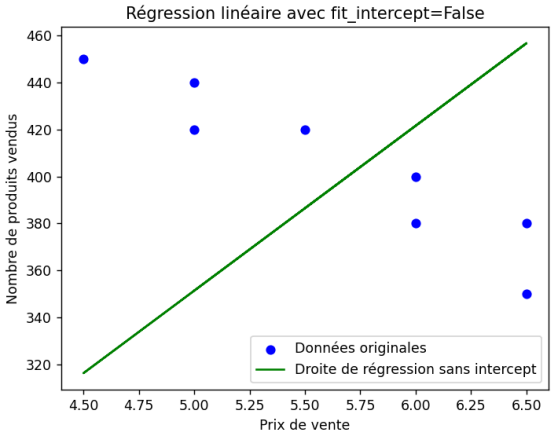


le modèle avec l'option fit_intercept=False, nous avons observé des résultats significativement différents par rapport à l'ajustement précédent. En choisissant cette option, le modèle n'inclut plus l'ordonnée à

l'origine (intercept), ce qui contraint la droite de régression à passer par le point (0,0). Cela a conduit à une valeur de 0 pour l'intercept, et le coefficient appris est passé à 70.27, ce qui signifie que pour chaque augmentation d'une unité du prix, le nombre de produits vendus augmente de 70.27. Ce résultat est contre-intuitif et incohérent avec les tendances réelles des données.

L'erreur quadratique moyenne (MSE) a fortement augmenté, atteignant 6385.12, ce qui montre que le modèle ne correspond plus du tout aux données observées.En outre, le coefficient de détermination R^2 est devenu négatif, avec une valeur de -5.39. Un R^2 négatif signifie que le modèle est non seulement inefficace, mais qu'il explique encore moins bien la variance des ventes qu'un modèle qui se contenterait de prédire la moyenne des ventes pour toutes les observations. En d'autres termes, forcer la droite de régression à passer par l'origine a dégradé les performances du modèle, à tel point qu'il fait pire qu'un modèle basique.

Enfin, pour visualiser les résultats avec **fit_intercept=False**, nous avons également tracé la droite de régression en rouge superposée au nuage de points des données. Cette représentation graphique met en évidence comment le modèle, en l'absence d'une ordonnée à l'origine, ne parvient pas à s'ajuster correctement aux valeurs observées.

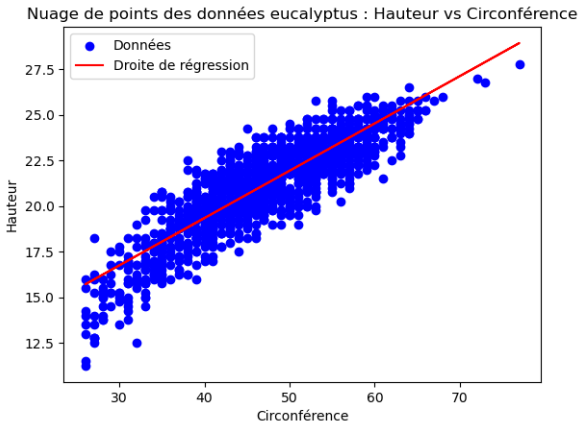


2 Jeu de données Eucaliptus :

Dans cette deuxième partie, nous allons essayer de prédire la hauteur des arbres d'eucalyptus grâce à leur circonférence. Nous utiliserons les données du fichier *Eucalyptus* téléchargé sur Ametice.

Dans un premier temps, nous allons extraire les données et les stocker dans un tableau. Ensuite, nous allons stocker la première colonne du tableau dans une variable h (hauteur) et la seconde dans une variable c (circonférence).

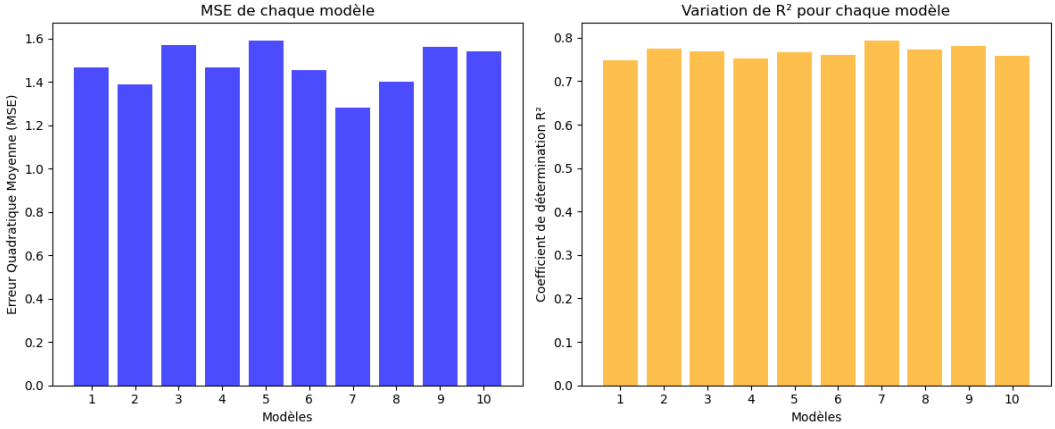
En représentant le nuage de points à l'aide des méthodes de la bibliothèque `Matplotlib`, ainsi que la droite de régression obtenue à partir d'un modèle de prédiction créé avec la méthode `LinearRegression` sur nos données, nous obtenons le graphique ci-dessous :



Nous pouvons clairement observer une corrélation entre nos variables : plus la circonférence est élevée, plus la hauteur a tendance à augmenter.

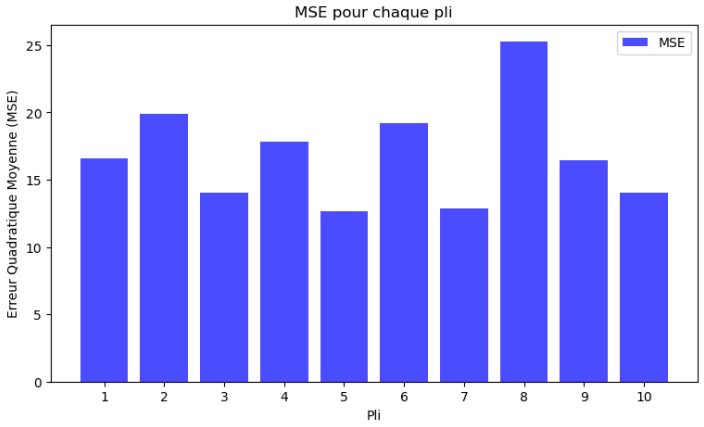
En utilisant la méthode `train_test_split`, nous avons obtenu un modèle de régression linéaire entraîné sur 80% des données et qui prédit la hauteur des arbres sur les 20% restants. L'erreur quadratique moyenne (MSE) obtenue est de 1.5367, ce qui constitue une valeur acceptable, indiquant que notre modèle reflète fidèlement les données observées.

Dans le but d'obtenir un modèle plus performant, nous allons créer 10 modèles différents en utilisant la méthode *hold-out*. Pour chaque modèle, nous calculerons les valeurs de l'erreur quadratique moyenne (MSE) et du coefficient de détermination (R^2), que nous représenterons dans le graphique ci-dessous:



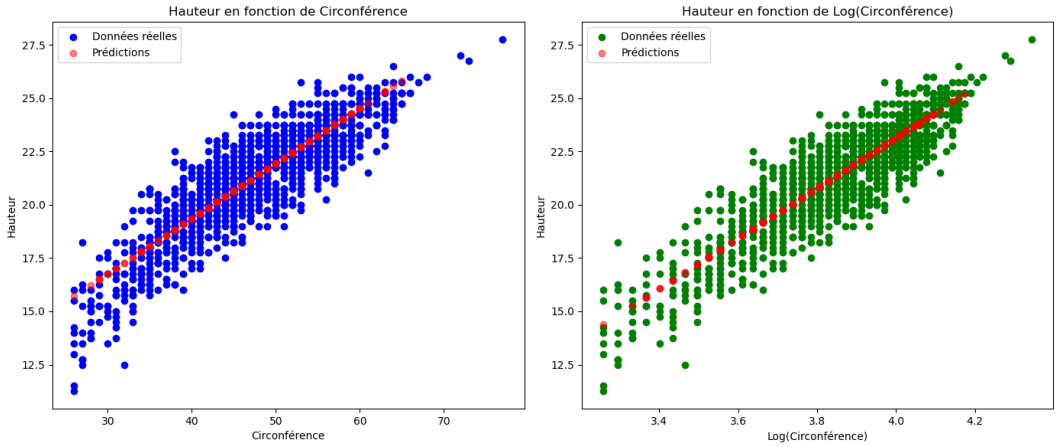
En analysant les valeurs de MSE et de R^2 obtenues, nous constatons que le modèle 7 semble être celui qui représente le mieux nos données. Il présente une erreur quadratique moyenne relativement faible ainsi qu'un coefficient de détermination proche de 1, ce qui indique une bonne capacité prédictive.

Étant donné que les valeurs de la variable *random state*, qui détermine les sous-ensembles d'apprentissage et de tests, est aléatoire, nous n'avons aucune garantie d'avoir obtenu le meilleur modèle possible en utilisant la méthode *hold-out*. Nous allons donc utiliser la méthode de validation croisée, qui est une approche plus robuste et efficace pour évaluer les performances d'un modèle sur plusieurs partitions des données. Nous observons ainsi une plus grande variation de la MSE calculée sur les différents modèles, comme l'illustre le graphique ci-dessous :



Ce graphique nous offre une idée plus claire de l'ensemble d'apprentissage capturant au mieux les caractéristiques de nos données. De plus, nous avons créé un graphique représentant les valeurs de R^2 calculées sur chaque pli, disponible sur notre dépôt GitHub.

Avant de passer à la régression linéaire multivariée, nous allons chercher à identifier s'il existe une meilleure relation de corrélation entre nos variables. Pour cela, nous représenterons la variation de la hauteur des arbres en fonction de la circonférence c, puis la variation de h en fonction de $\log(c)$. Nous obtenons les deux graphiques suivants :



Il est rapidement évident qu'il existe une meilleure relation de corrélation entre la hauteur et le logarithme de c, ce qui est également confirmé par la valeur de la MSE calculée, qui est plus faible que celle obtenue dans notre premier modèle de régression basé sur les valeurs de c [4]. Il serait donc intéressant d'ajouter une troisième variable à prendre en compte pour la prédiction de la hauteur.

Pour effectuer la régression linéaire multivariée sur nos données, nous allons utiliser la méthode `columnstack` de la bibliothèque NumPy [1], qui nous permettra de créer un tableau à deux colonnes en concaténant la colonne contenant les valeurs de c et celle contenant les valeurs correspondantes de $\log(c)$. En utilisant la méthode de régression linéaire nous obtenons le graphique que vous retrouverez sur github [3].

Après avoir calculé la MSE moyenne ainsi que le coefficient de détermination R^2 moyen de nos modèles de régression multivariée obtenus grâce à la méthode de validation croisée, nous remarquons qu'il existe un modèle de régression plus performant que ceux que nous avons identifiés jusqu'à présent[4].

References

[1] Numpy Documentation. *Numpy User Guide*. https://numpy.org/devdocs/user/

[2] Scikit-learn Documentation. *Scikit-learn User Guide*. https://scikit-learn.org/stable/

[3] matplotlib Documentation. *matplotlib User Guide*. https://matplotlib.org/stable/index.html

[4] Vous retrouverez le code directement en suivant ce lien. https://github.com/SelmaFanani/TDM3