|  |
| --- |
| Esperance de Vie au monde |
| FOUILLE DE DONNEES |
| **Réalisé par :**  **HIBA KHARFASSE**  **Encadré par :**  **MR.YASSINE AL-AMRANI** |
|  |





***Table des Matières***

DEDICACES: 1

REMERCIEMENTS 2

Définition d l’Esperance de vie mondiale 3

Pourqoui l’Esperance de vie mondiale 4-5

Les Facteurs Influençant l'Espérance de Vie des Individus 6-7

Les Disparités d'Espérance de Vie entre les Pays8

Importation de la dataset 9-10

BIBLIOTHEQUES UTILISEES11

LES MODELES UTILISES:12-18

LE CODE19-42

Comparaison entre les modéles *43-46*

# Dédicaces :

# Nous tenons à consacrer humblement ce travail à ceux qui ont été les piliers de notre éducation et de notre développement :

# À nos formateurs et à toute l'équipe pédagogique, dont la dévotion et le dévouement ont été la boussole de notre apprentissage. Leur engagement inlassable à nous guider, à nourrir nos esprits et à préserver notre bien-être ont été la clé de notre évolution. Leur influence demeure gravée en nous, portant en héritage leur savoir et leur bienveillance.

# À nos parents, véritables architectes de notre succès, dont le soutien indéfectible et les encouragements ont pavé le chemin de notre parcours académique. Leur amour inconditionnel et leurs sacrifices ont été le socle sur lequel nous avons bâti nos réussites. Puissent-ils être bénis et protégés par Dieu pour leur rôle irremplaçable dans notre vie.

# Leur présence et leur dévouement ont façonné notre chemin, et cette humble dédicace est un témoignage de notre gratitude éternelle envers ceux qui ont été nos guides et nos anges gardiens.

# Remerciements :

En ce moment de conclusion, nos cœurs débordent de reconnaissance envers le Tout-Puissant qui, par Sa grâce infinie, nous a accordé la patience, la force et le courage nécessaires pour mener à bien cet humble travail. Sa guidance a été notre lumière à chaque étape de ce périple académique.

Nous exprimons également notre profonde gratitude à toute la communauté de l'école ENSATE ainsi qu'à l'ensemble des enseignants qui, de par leur dévouement et leur expertise, ont façonné notre apprentissage. Leur engagement indéfectible à transmettre le savoir et à nourrir nos esprits a été un pilier fondamental de notre formation. Leur dévotion nous a permis de grandir intellectuellement et humainement.

En particulier, nos remerciements vont à Monsieur AL-AMRANI, notre encadrant pour ce projet, dont l'engagement sans faille et les précieux conseils ont été d'une importance capitale. Sa disponibilité, ses orientations éclairées et son soutien constant ont été des atouts essentiels pour la réalisation de ce travail. Nous lui sommes infiniment reconnaissants pour son dévouement et sa contribution précieuse à notre cheminement académique.

Nos mots ne sauraient exprimer pleinement la gratitude que nous ressentons envers chacune de ces personnes qui ont été des piliers dans la réalisation de ce travail. Leurs efforts ont été des fondations solides sur lesquelles repose notre succès et notre évolution.

***Définition d l’Esperance de vie mondiale :***

L'espérance de vie mondiale représente la projection statistique de la durée de vie moyenne qu'une génération naissante peut anticiper à un moment précis, en tenant compte des taux de mortalité observés à l'échelle planétaire. Cette mesure constitue une estimation théorique du nombre moyen d'années qu'un individu peut espérer vivre dès sa naissance, en prenant en considération les tendances de mortalité globales.

Ce calcul complexe intègre une multitude de variables influençant la longévité humaine, parmi lesquelles l'accessibilité aux soins de santé, les conditions socio-économiques, le niveau d'éducation, les avancées médicales, les comportements individuels et les facteurs environnementaux. En amalgamant les données de décès provenant de diverses régions et populations du monde, cette estimation génère une vue d'ensemble de la durée de vie moyenne des habitants de la planète.

L'espérance de vie mondiale sert de baromètre essentiel pour évaluer la santé globale et le progrès des conditions de vie à travers les multiples cultures, économies et systèmes de santé mondiaux. Elle offre un panorama général de la longévité humaine à l'échelle globale, permettant de comparer les disparités entre nations et régions. Ce chiffre éclairé par les tendances démographiques et les évolutions socio-sanitaires constitue un outil clé pour évaluer les progrès dans la lutte contre les maladies, l'amélioration des conditions de vie et les politiques de santé publique à travers le monde.

***Pourquoi Choisir de se pencher sur l'espérance de vie au niveau mondial plutôt que dans un pays spécifique ?***

Choisir de se pencher sur l'espérance de vie au niveau mondial plutôt que dans un pays spécifique peut découler de plusieurs raisons :

**Perspective globale** : Comprendre l'espérance de vie à l'échelle mondiale offre une vision plus large et inclusive. Cela permet de considérer les diversités culturelles, économiques et sociales qui influent sur la longévité humaine à travers le monde.

**Comparaisons et contrastes** : En examinant l'espérance de vie à l'échelle mondiale, il est possible de comparer les tendances entre les pays, les continents et les régions. Cela met en lumière les disparités et les similitudes, permettant d'identifier les facteurs communs ou spécifiques à certaines zones géographiques.

**Compréhension des défis globaux** : L'analyse de l'espérance de vie à l'échelle mondiale aide à saisir les défis de santé publique d'envergure planétaire, tels que les maladies pandémiques, les questions de santé maternelle et infantile, l'accès inégal aux soins de santé et d'autres problèmes mondiaux de santé.

**Contexte pour les études et la recherche** : Examiner l'espérance de vie à l'échelle mondiale fournit un contexte plus large pour la recherche et les études comparatives, offrant ainsi une base solide pour mieux comprendre les déterminants de la longévité humaine.

L'analyse des multiples variables liées à l'espérance de vie permet une vue détaillée des facteurs influençant la durée de vie à travers les populations mondiales. Cette approche globale combine des données démographiques, socio-économiques, environnementales, de santé et comportementales pour identifier les corrélations, repérer les tendances temporelles et géographiques, établir des modèles prédictifs, et mettre en lumière les inégalités. En croisant ces variables, cette analyse offre une perspective complète sur les déterminants de la longévité, facilitant une compréhension plus précise des différences entre les populations et permettant de concevoir des stratégies ciblées pour améliorer la santé à l'échelle mondiale.

***Les Facteurs Influençant l'Espérance de Vie des Individus :***

Les facteurs influençant l'espérance de vie sont variés et peuvent être regroupés en plusieurs catégories clés, parmi lesquelles le mode de vie, les facteurs génétiques, l'accès aux soins de santé et les conditions de vie.

**Mode de vie** : Les habitudes quotidiennes ont un impact significatif sur la durée de vie. Une alimentation équilibrée, l'activité physique régulière, l'absence de tabagisme et une consommation modérée d'alcool peuvent contribuer à une meilleure santé et donc à une espérance de vie plus longue. Des comportements sains réduisent le risque de maladies chroniques telles que les maladies cardiovasculaires, le diabète et certains cancers.

**Facteurs génétiques** : Bien que l'hérédité joue un rôle dans la longévité, elle n'est pas le seul facteur déterminant. Les gènes peuvent influencer la prédisposition à certaines maladies ou conditions de santé, mais l'environnement et le mode de vie peuvent également modérer leur impact. Comprendre les antécédents familiaux peut aider à anticiper certains risques de santé, mais des choix de vie sains restent cruciaux pour contrôler ces risques.

**Accès aux soins de santé** : L'accès à des soins de santé de qualité joue un rôle majeur dans l'espérance de vie. Les services médicaux préventifs, les traitements pour les maladies chroniques, les interventions chirurgicales et les médicaments contribuent à réduire les risques de complications et de décès prématuré. Les populations ayant un accès limité aux soins de santé sont souvent plus vulnérables à des maladies évitables et voient leur espérance de vie réduite.

**Conditions de vie** : Les conditions socio-économiques, le logement, l'accès à l'eau potable et à l'assainissement, l'éducation, et d'autres aspects environnementaux sont également des déterminants majeurs de l'espérance de vie. Des conditions de vie défavorables peuvent entraîner des risques accrus pour la santé, une exposition à la pollution, une alimentation inadéquate, des conditions de travail dangereuses, et ainsi impacter négativement la durée de vie.

En somme, l'espérance de vie est le fruit de l'interaction complexe entre des facteurs génétiques, des choix individuels, l'accès aux soins de santé et les conditions de vie. Comprendre ces différents aspects permet d'élaborer des stratégies visant à améliorer la santé et à augmenter l'espérance de vie des populations.

***Les Disparités d'Espérance de Vie entre les Pays:***

Les disparités d'espérance de vie entre les pays sont significatives et révèlent des différences marquées selon le niveau de développement économique, les systèmes de santé, les conditions socio-économiques et d'autres facteurs. Voici un aperçu général des disparités d'espérance de vie entre différents groupes de pays :

**1. Pays Développés** : - Ces pays, caractérisés par des économies avancées, des systèmes de santé bien développés et des conditions de vie généralement meilleures, affichent souvent des espérances de vie plus élevées. Des infrastructures solides, des programmes de prévention des maladies, un accès plus large aux soins de santé et une meilleure qualité de vie en général contribuent à une longévité accrue pour leurs citoyens.

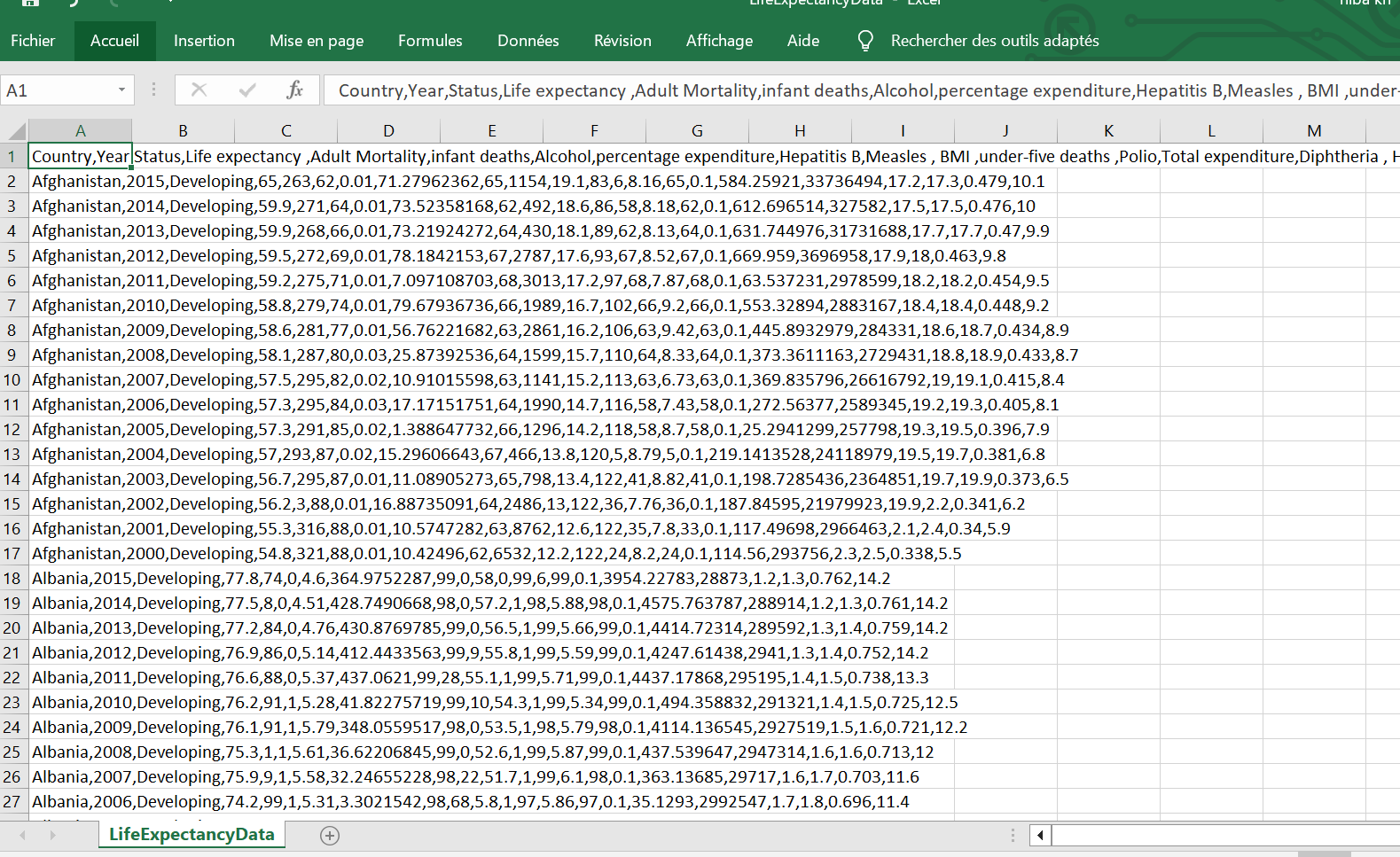
**2. Pays en Développement** : - Les pays en développement présentent souvent des disparités internes. Certains peuvent afficher des améliorations substantielles de l'espérance de vie en raison de progrès dans les soins de santé, l'éducation et le développement économique. Cependant, beaucoup de ces pays sont confrontés à des défis tels que des systèmes de santé fragiles, des ressources limitées, des maladies infectieuses répandues et des conditions de vie précaires, ce qui peut réduire l'espérance de vie.

**3. Pays les Moins Avancés** : - Les pays les moins avancés, souvent confrontés à des difficultés socio-économiques majeures, des conflits, une pauvreté généralisée et des infrastructures de santé sous-développées, ont généralement les espérances de vie les plus courtes. Les taux de mortalité infantile élevés, les maladies infectieuses prévalentes et un accès limité aux soins de santé de base contribuent à une espérance de vie plus faible dans ces nations.

Ces disparités soulignent l'importance des inégalités dans l'accès aux soins de santé, aux ressources et aux opportunités entre les pays. Les initiatives visant à améliorer l'espérance de vie dans les pays en développement et les nations les moins avancées nécessitent une attention particulière aux problèmes de santé publique, à l'éducation, au développement économique et à l'accès équitable aux soins de santé pour réduire ces écarts.

***Importation de la dataset***



******

On a importé la dataset depuis le site GETHUB sous forme de fichier csv qui contient 22 colonnes :

1. Pays

2. Année

3. Statut, Espérance de vie

4. Mortalité des adultes

5. décès de nourrissons

6. Alcool

7. pourcentage de dépenses

8. Hepatitis B

9. Rougeole

10. BMI

11. décès d'enfants de moins de cinq ans

12. La poliomyélite

13. Total des dépenses

14. Diphtérie

Le BMI (Body Mass Index), ou IMC (Indice de Masse Corporelle) en français, est un indice utilisé pour évaluer la corpulence d'une personne. Il se calcule en divisant le poids (en kilogrammes) par le carré de la taille (en mètres).

15. VIH/SIDA

16. PIB

17. Population

18. minceur\_1\_19\_ans

19. minceur\_5\_9\_ans

20. Composition des revenus des ressources

21. Adulte

22. Scolarité

Et 2939 lignes classées en ordre alphabétique des pays du monde entier.

***BIBLIOTHEQUES UTILISEES :***

**Pandas** : Pandas est une bibliothèque open-source pour Python utilisée principalement pour la manipulation et l'analyse des données. Elle offre des structures de données et des outils performants pour travailler avec des données tabulaires et des séries temporelles, facilitant ainsi le chargement, le nettoyage, la transformation et l'analyse de données.

**NumPy** : NumPy est une bibliothèque fondamentale en Python pour le calcul numérique. Elle fournit des tableaux multidimensionnels, des fonctions mathématiques avancées, des opérations de manipulation de tableaux et des outils pour intégrer du code écrit en langages de bas niveau (comme le C ou le Fortran).

**Matplotlib.pyplot** : Matplotlib est une bibliothèque de traçage de graphiques en Python. Matplotlib.pyplot est l'une de ses sous-bibliothèques largement utilisée pour créer des graphiques et des visualisations de données. Elle permet de générer des graphiques simples ou complexes, de personnaliser les paramètres visuels et d'ajouter des éléments aux graphiques.

**Seaborn** : Seaborn est une bibliothèque de visualisation de données basée sur Matplotlib, offrant une interface haut niveau pour créer des graphiques statistiques attrayants. Elle simplifie la création de graphiques complexes en fournissant des fonctions pour visualiser des données statistiques avec des couleurs attractives et des designs prédéfinis.

**Warnings** : Warnings est un module Python qui gère les avertissements (ou les avertissements d'exécution) générés par le code. Il permet de contrôler la manière dont Python traite et affiche les avertissements, offrant la possibilité de les ignorer, de les afficher une seule fois ou de les traiter d'autres manières, selon les besoins du programmeur.

***LES MODELES UTILISES :***

**1. Régression linéaire :**

La régression linéaire est une technique statistique utilisée pour modéliser la relation entre une variable dépendante et une ou plusieurs variables indépendantes. Son fonctionnement repose sur la création d'une ligne droite (dans le cas de la régression linéaire simple) ou d'un plan (dans le cas de la régression linéaire multiple) qui représente au mieux la relation entre les variables.

L'objectif principal de la régression linéaire est de trouver cette ligne ou ce plan qui minimise la distance verticale entre les valeurs réelles observées et les valeurs prédites par le modèle. Pour ce faire, elle utilise une formule mathématique qui est ajustée aux données disponibles.

Les étapes clés de la régression linéaire sont les suivantes :

Collecte des données : Rassembler des données sur la variable dépendante et les variables indépendantes pour former un ensemble de données.

Choix du modèle : Déterminer s'il s'agit d'une régression linéaire simple (une seule variable indépendante) ou d'une régression linéaire multiple (plusieurs variables indépendantes).

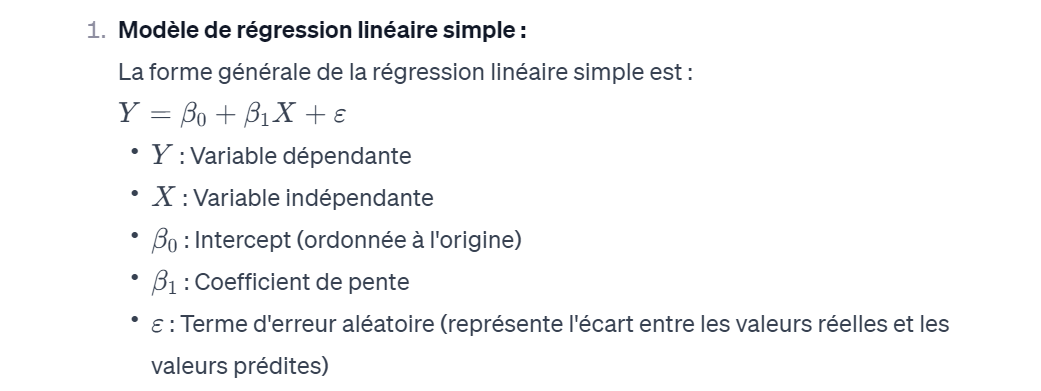
Construction du modèle : Trouver la ligne ou le plan qui représente au mieux la relation entre les variables en minimisant les erreurs de prédiction.

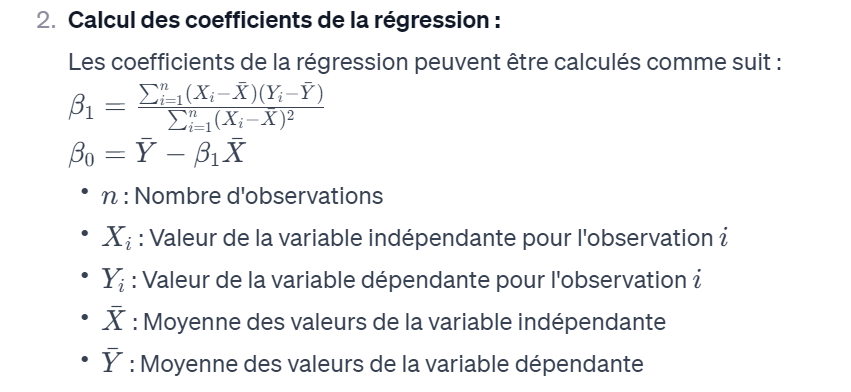
Évaluation du modèle : Évaluer la qualité du modèle en utilisant des métriques telles que le coefficient de détermination (R²), qui indique à quel point le modèle s'adapte aux données.

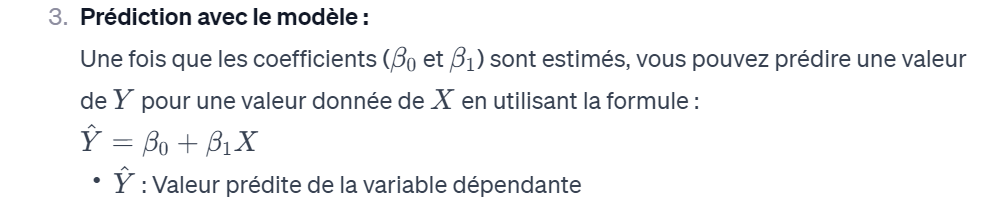
Utilisation du modèle : Une fois validé, le modèle peut être utilisé pour faire des prédictions sur de nouvelles données en estimant la valeur de la variable dépendante en fonction des valeurs des variables indépendantes.

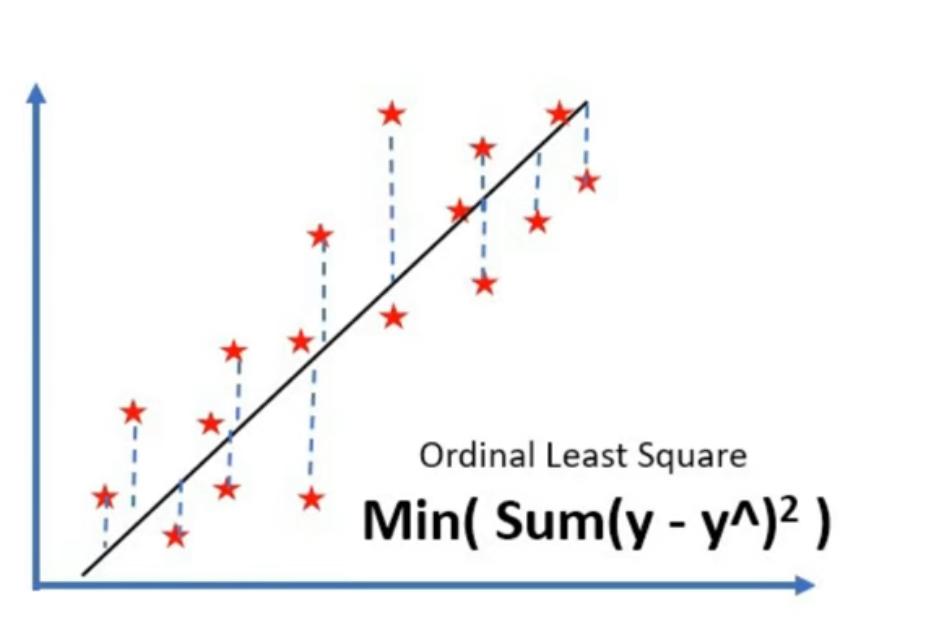
La régression linéaire est largement utilisée dans de nombreux domaines, notamment en économie, en sciences sociales, en biologie, en ingénierie et dans d'autres disciplines où il est nécessaire de modéliser et de comprendre les relations entre les variables.

A-Régression Linéaire Simple :

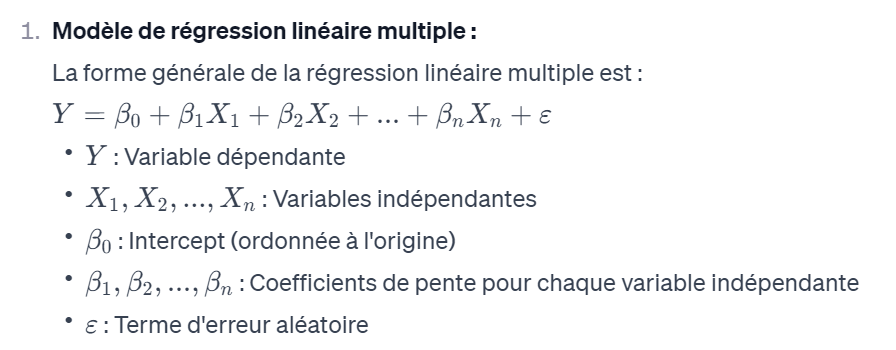


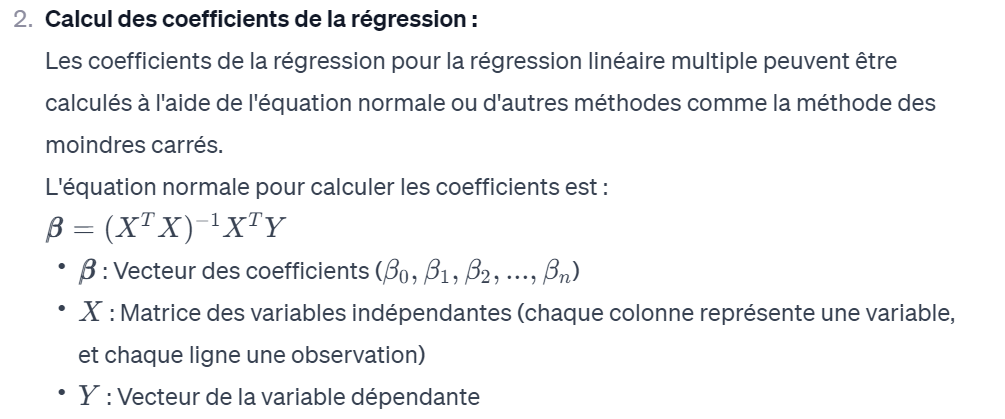


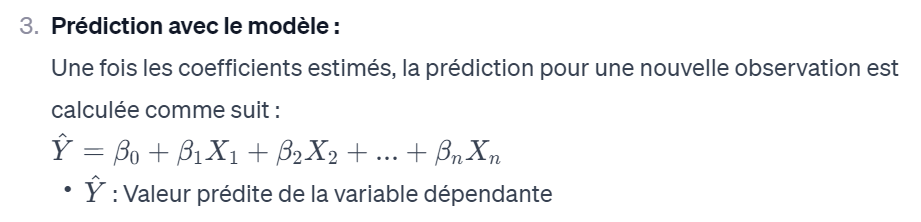


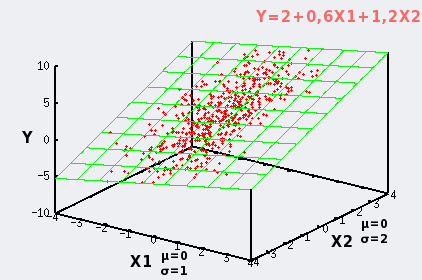


B-Régression Linéaire Multiple :









Dans notre cas on va travailler avec la régression linéaire multiple

**2. Forêts aléatoires (Random Forest) :**

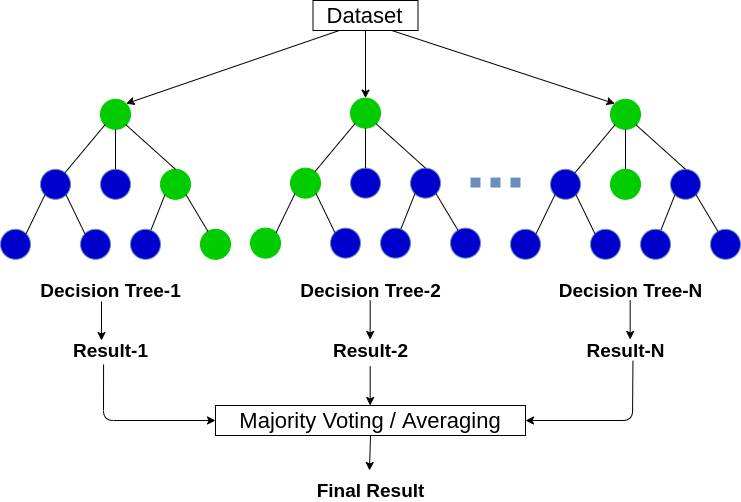
Les forêts aléatoires, ou Random Forest en anglais, sont des algorithmes d'apprentissage automatique qui s'appuient sur le principe de l'agrégation pour effectuer des prédictions. Elles sont largement utilisées pour des tâches de classification et de régression.

Leur fonctionnement repose sur la construction de multiples arbres de décision, d'où le terme "forêts". Chaque arbre est construit de manière aléatoire en utilisant une partie aléatoire des données d'entraînement et des variables disponibles. Ce processus implique la sélection aléatoire de sous-ensembles de données et de caractéristiques pour chaque arbre.

Lors de la prédiction, chaque arbre donne une réponse et la forêt aléatoire combine ces réponses pour produire une prédiction finale. Pour les tâches de classification, cela peut être un vote majoritaire des classes prédites par chaque arbre. Pour les tâches de régression, la prédiction finale est souvent la moyenne des valeurs prédites par chaque arbre.

Les forêts aléatoires sont appréciées pour leur capacité à réduire le surajustement (overfitting) grâce à la diversité des arbres construits de manière aléatoire. Elles sont également robustes face aux données bruitées ou à des variables non pertinentes.

En somme, les forêts aléatoires tirent leur puissance de la combinaison de multiples modèles d'arbres de décision, exploitant la diversité et la randomisation pour offrir des prédictions précises et robustes.



**3. Support Vector Regression (SVR) :**

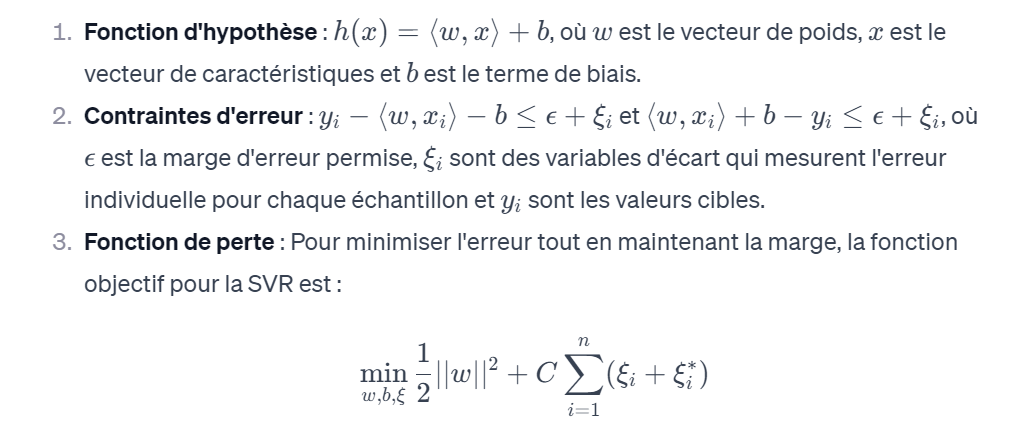
La Régression par Vecteurs de Support (SVR - Support Vector Regression) est une technique d'apprentissage automatique utilisée pour la régression, c'est-à-dire pour prédire des valeurs continues plutôt que des catégories discrètes. Elle repose sur le même principe que les machines à vecteurs de support (SVM - Support Vector Machines) pour la classification, mais est adaptée pour la régression.

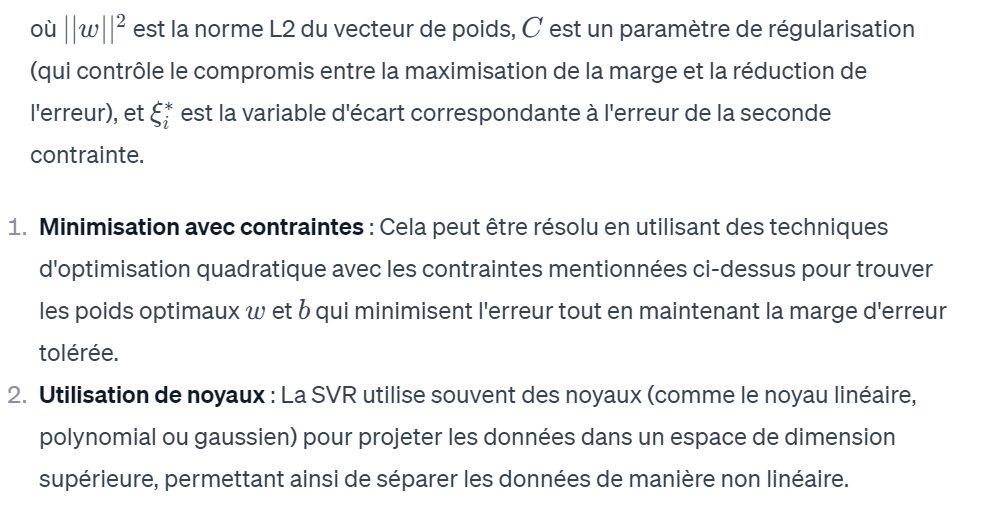
Le principe fondamental de la SVR est de trouver une fonction qui permet de prédire au mieux les valeurs désirées en minimisant simultanément l'erreur de prédiction et en maintenant cette erreur dans une certaine marge d'erreur définie par l'utilisateur. Cette marge est déterminée par deux paramètres principaux : le paramètre de régularisation et le paramètre de marge.

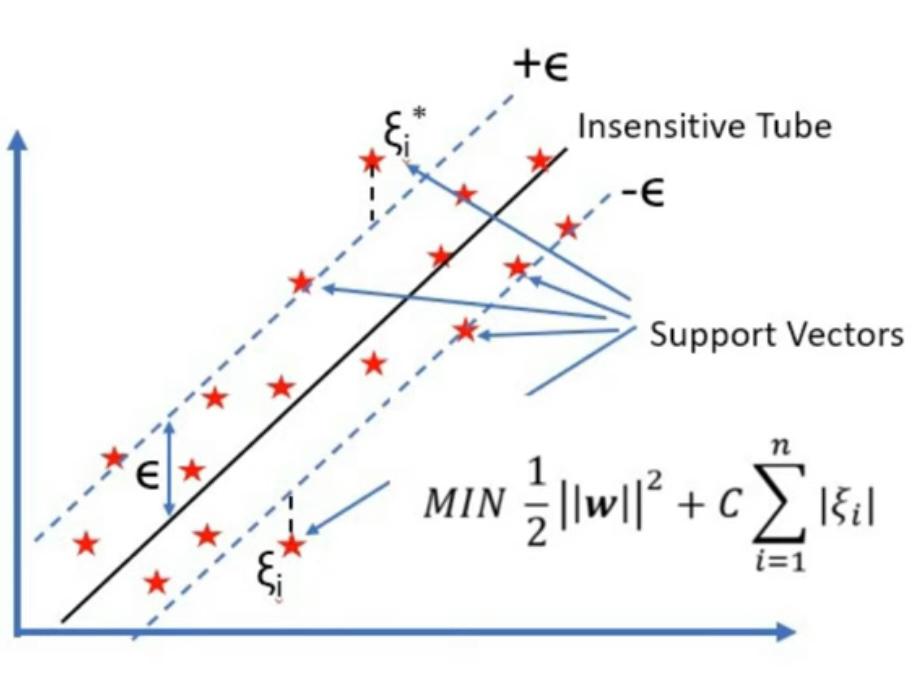
La SVR cherche à trouver un hyperplan dans un espace de dimension supérieure (l'espace des features) qui maximise la marge tout en tolérant un certain niveau d'erreur (appelé aussi "slack"). Elle utilise des vecteurs de support, qui sont les échantillons les plus proches de l'hyperplan de décision, pour définir cet hyperplan.

Pour atteindre ces objectifs, la SVR utilise des noyaux (kernels) qui permettent de projeter les données dans un espace de dimension supérieure où la séparation des données est plus simple. Les noyaux couramment utilisés incluent le noyau linéaire, polynomial et gaussien (RBF - Radial Basis Function).

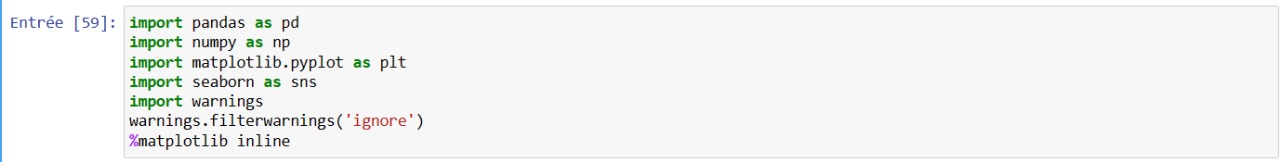
En résumé, la SVR est une technique puissante pour la régression qui cherche à trouver un hyperplan dans un espace de dimension supérieure tout en maximisant la marge et en tolérant une certaine erreur pour prédire au mieux des valeurs continues. Elle est largement utilisée dans divers domaines tels que la finance, la médecine, la prévision économique et d'autres où la prédiction de valeurs continues est essentielle.

******

******

******

***LE CODE :***

******

import pandas as pd :

Cette ligne importe la bibliothèque Pandas, qui est largement utilisée pour la manipulation et l'analyse de données en Python.

Le mot-clé as pd permet de créer un alias pour la bibliothèque Pandas, ce qui signifie que vous pouvez utiliser l'alias "pd" au lieu du nom complet "pandas" dans votre code. C'est une convention courante dans la communauté Python.

import numpy as np :

Cette ligne importe la bibliothèque NumPy, qui fournit des structures de données et des fonctions pour le calcul numérique en Python.

Comme pour Pandas, l'alias "np" est couramment utilisé pour NumPy.

import matplotlib.pyplot as plt :

Cette ligne importe le module pyplot de la bibliothèque Matplotlib, qui est utilisée pour créer des visualisations graphiques, tels que des graphiques et des diagrammes.

L'alias "plt" est couramment utilisé pour pyplot.

import seaborn as sns :

Cette ligne importe la bibliothèque Seaborn, qui est une bibliothèque de visualisation basée sur Matplotlib. Elle simplifie la création de visualisations esthétiques et informatives.

L'alias "sns" est couramment utilisé pour Seaborn.

import warnings :

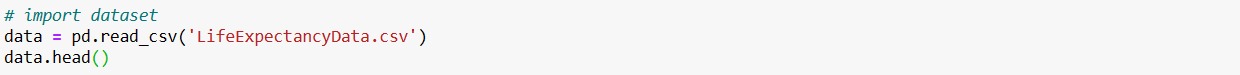
Cette ligne importe le module "warnings", qui permet de gérer les avertissements émis par Python.

warnings.filterwarnings('ignore') :

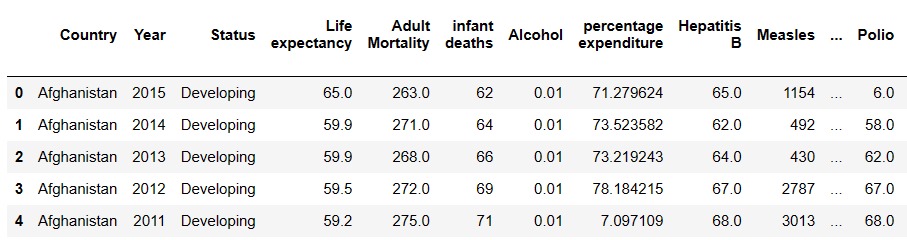
Cette ligne ignore les avertissements afin de rendre la sortie du code plus propre. En général, il est recommandé de ne pas ignorer les avertissements, mais dans certains cas, cela peut être utile, notamment lors de la présentation de résultats où les avertissements non pertinents peuvent être omis.

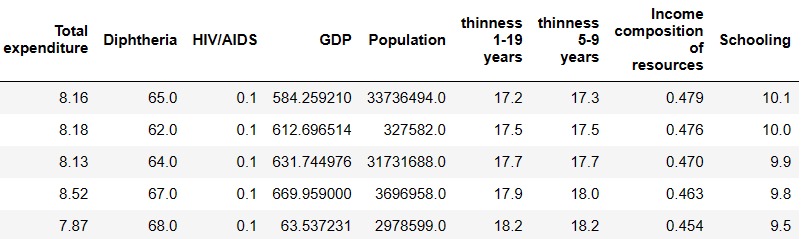
%matplotlib inline :

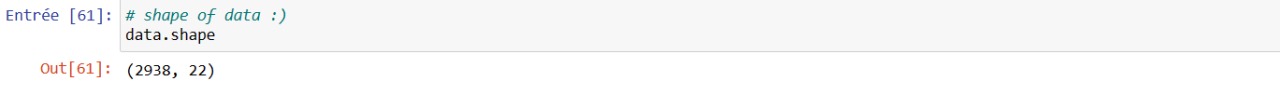
Cette ligne est spécifique à l'environnement Jupyter Notebook. Elle permet d'afficher les graphiques générés par Matplotlib directement dans le carnet, plutôt que dans une fenêtre séparée. Cela est pratique pour l'analyse exploratoire des données dans un environnement Jupyter Notebook



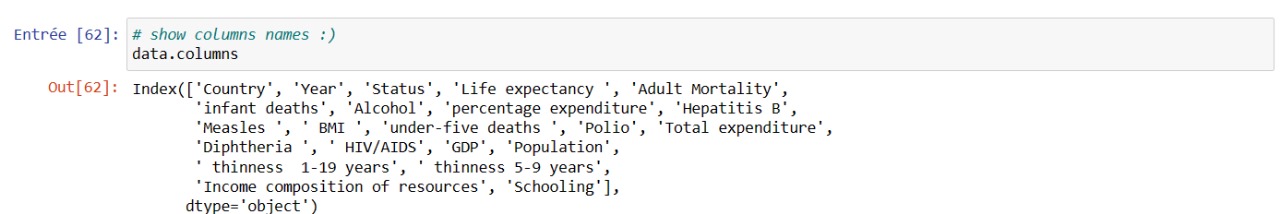
data = pd.read\_csv('LifeExpectancyData.csv') lit un fichier CSV nommé 'LifeExpectancyData.csv' et stocke les données dans un DataFrame pandas appelé "data". Un DataFrame est une structure de données bidimensionnelle et tabulaire avec des axes étiquetés (lignes et colonnes). data.head():affiche les premières lignes du DataFrame "data" , C'est une manière rapide d'inspecter la structure et le contenu des données chargées.







renvoie un tuple représentant les dimensions du DataFrame data. Le premier élément du tuple est le nombre de lignes (observations), et le deuxième élément est le nombre de colonnes (variables).



La ligne data.columns = [...] attribue de nouveaux noms aux colonnes du DataFrame data. Les noms de colonnes spécifiés dans la liste sont assignés dans le même ordre que l'ordre d'origine des colonnes du DataFrame.

Dans cet exemple, les colonnes du DataFrame data seront renommées avec les noms fournis dans la liste. Voici les nouveaux noms des colonnes dans l'ordre spécifié :

1. 'Country'

2. 'Year'

3. 'Status'

4. 'Life\_expectancy'

5. 'Adult\_Mortality'

6. 'infant\_deaths'

7. 'Alcohol'

8. 'percentage\_expenditure'

9. 'Hepatitis\_B'

10. 'Measles'

11. 'BMI'

12. 'under\_five deaths '

13. 'Polio'

14. 'Total\_expenditure'

15. 'Diphtheria'

16. 'HIV/AIDS'

17. 'GDP'

18. 'Population'

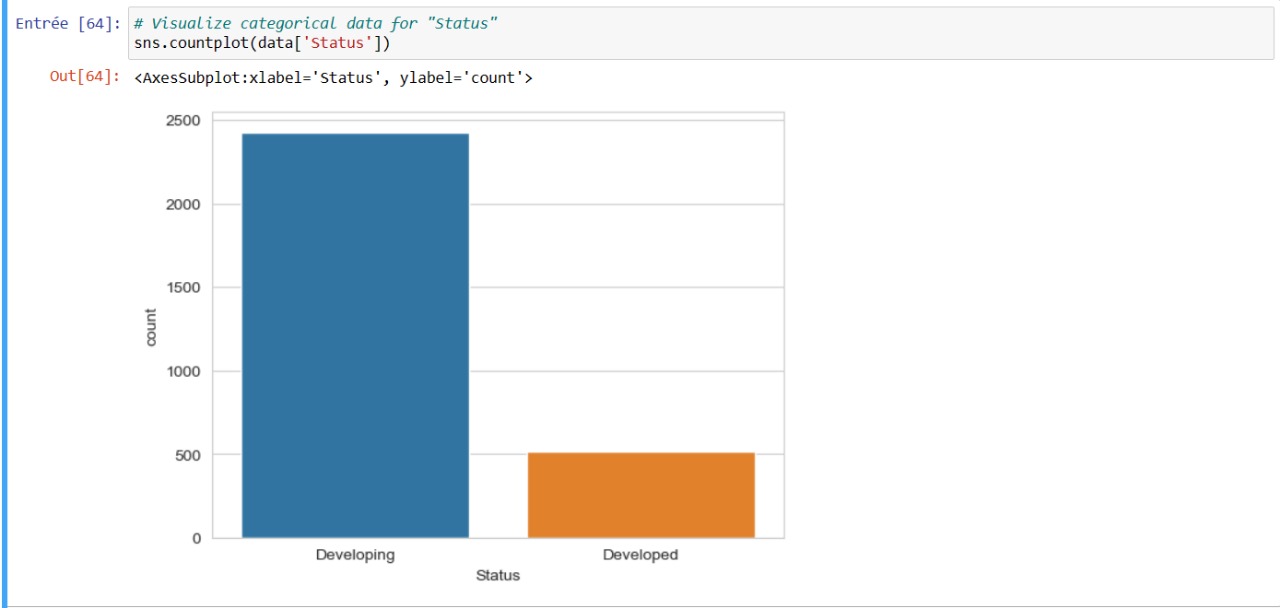
19. 'thinness\_1\_19\_years'

20. 'thinness\_5\_9\_years'

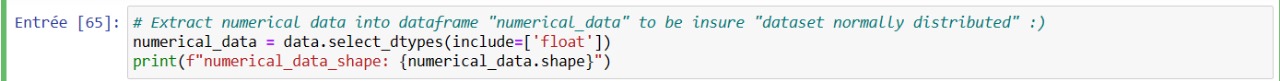
21. 'Income\_composition\_of\_resources'

22. 'Schooling'

Après cette opération, vous pouvez utiliser ces nouveaux noms de colonnes pour accéder aux différentes variables de votre ensemble de données.



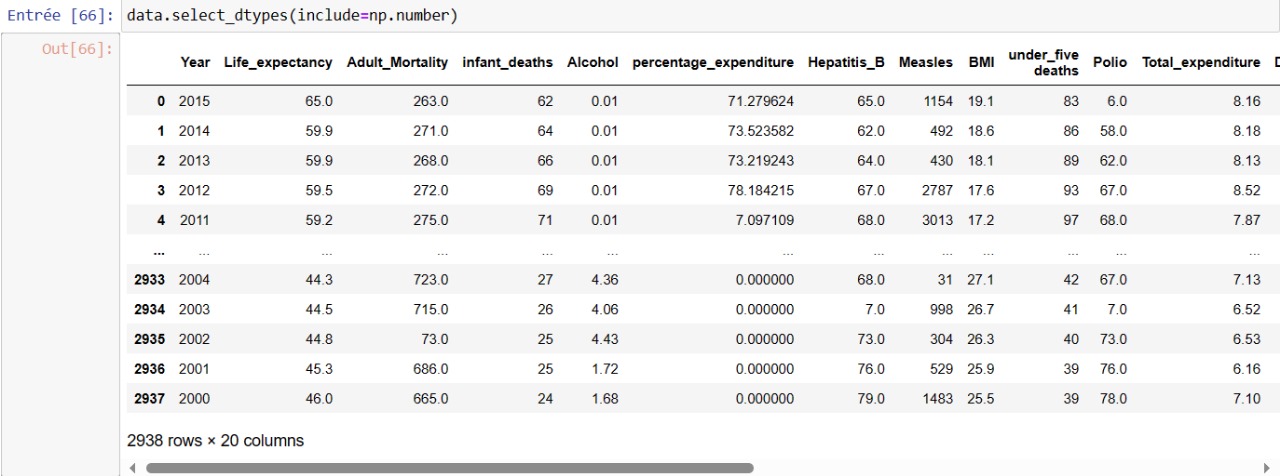
Le code sns.countplot(data['Status']) utilise la bibliothèque Seaborn en Python pour créer un diagramme à barres représentant la distribution du nombre d'occurrences des différentes valeurs présentes dans la colonne 'Status' du DataFrame data. La première ligne du code importe la bibliothèque Seaborn et lui attribue l'alias "sns", facilitant ainsi l'utilisation des fonctions de visualisation. Ensuite, la fonction countplot est appliquée à la colonne 'Status' du DataFrame, générant un diagramme à barres qui affiche le nombre d'occurrences pour chaque catégorie de statut. Cette visualisation peut être particulièrement utile pour comprendre la répartition des données selon les différentes catégories de statut présentes dans l'ensemble de données, fournissant ainsi un aperçu visuel de la distribution de cette variable.



Le bloc de code que vous avez fourni effectue deux opérations essentielles. Tout d'abord, la ligne numerical\_data = data.select\_dtypes(include=['float']) utilise la méthode select\_dtypes de la bibliothèque pandas pour filtrer les colonnes du DataFrame data et ne conserver que celles de type float. Cela signifie que le nouveau DataFrame numerical\_data contiendra exclusivement des données numériques à virgule flottante.

Ensuite, la ligne suivante, print(f"numerical\_data\_shape: {numerical\_data.shape}"), affiche les dimensions du DataFrame nouvellement créé, numerical\_data. L'utilisation de la fonction shape renvoie un tuple représentant le nombre de lignes et de colonnes dans le DataFrame, fournissant ainsi une indication rapide de la structure du DataFrame résultant.

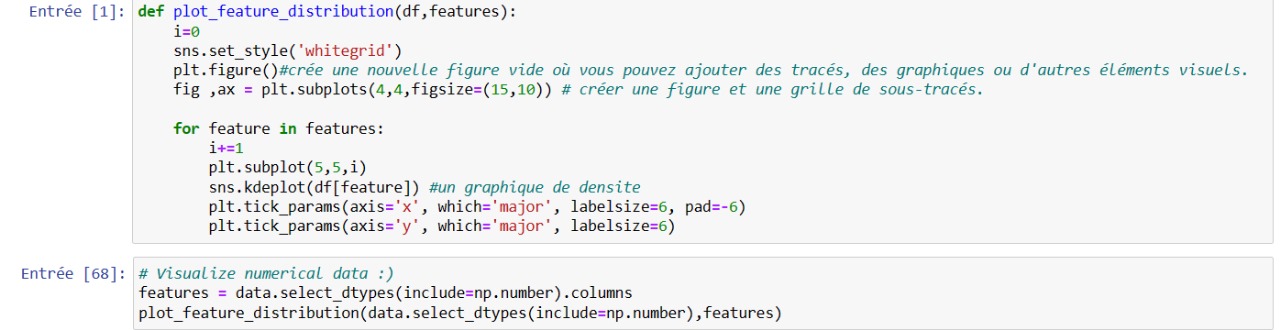
Cette séquence de code peut être particulièrement utile lorsque vous souhaitez isoler et analyser spécifiquement les données numériques dans votre ensemble de données, offrant ainsi une meilleure compréhension des caractéristiques quantitatives présentes.

******

Le bloc de code que vous avez fourni utilise la bibliothèque NumPy (abrégée en np) pour sélectionner toutes les colonnes de type numérique du DataFrame data. Cependant, il semble manquer l'importation de NumPy, alors j'ai inclus l'importation nécessaire dans le code corrigé.

La première ligne import numpy as np importe la bibliothèque NumPy et lui attribue l'alias "np". Ensuite, la ligne numerical\_data = data.select\_dtypes(include=np.number) utilise la méthode select\_dtypes de pandas en spécifiant np.number comme argument pour inclure toutes les colonnes de type numérique dans le DataFrame data. Ainsi, le DataFrame résultant, nommé numerical\_data, ne contiendra que des données numériques.

Cette opération est utile lorsque vous souhaitez isoler les caractéristiques numériques de votre ensemble de données, facilitant ainsi l'analyse spécifique des données quantitatives. En résumé, ce bloc de code permet de créer un nouveau DataFrame, numerical\_data, qui contient exclusivement les colonnes numériques du DataFrame d'origine data.



La fonction plot\_feature\_distribution que vous avez fournie semble être conçue pour visualiser la distribution des caractéristiques numériques d'un DataFrame à l'aide de tracés de densité (Kernel Density Estimation - KDE). Voici une explication de la fonction en paragraphes :

La fonction prend deux paramètres en entrée : df qui représente le DataFrame contenant les données à visualiser, et features qui est une liste des noms des caractéristiques (colonnes) dont vous souhaitez examiner la distribution.

À l'intérieur de la fonction, la variable i est initialisée à 0, et le style de la grille est défini comme 'whitegrid' à l'aide de sns.set\_style('whitegrid').

Ensuite, une nouvelle figure est créée à l'aide de plt.figure() pour permettre l'ajout de tracés ou de graphiques. En parallèle, une grille de sous-tracés est créée avec plt.subplots(4,4,figsize=(15,10)), indiquant une disposition de 4 lignes sur 4 colonnes pour un total de 16 sous-tracés, avec une taille de figure de 15 par 10 pouces.

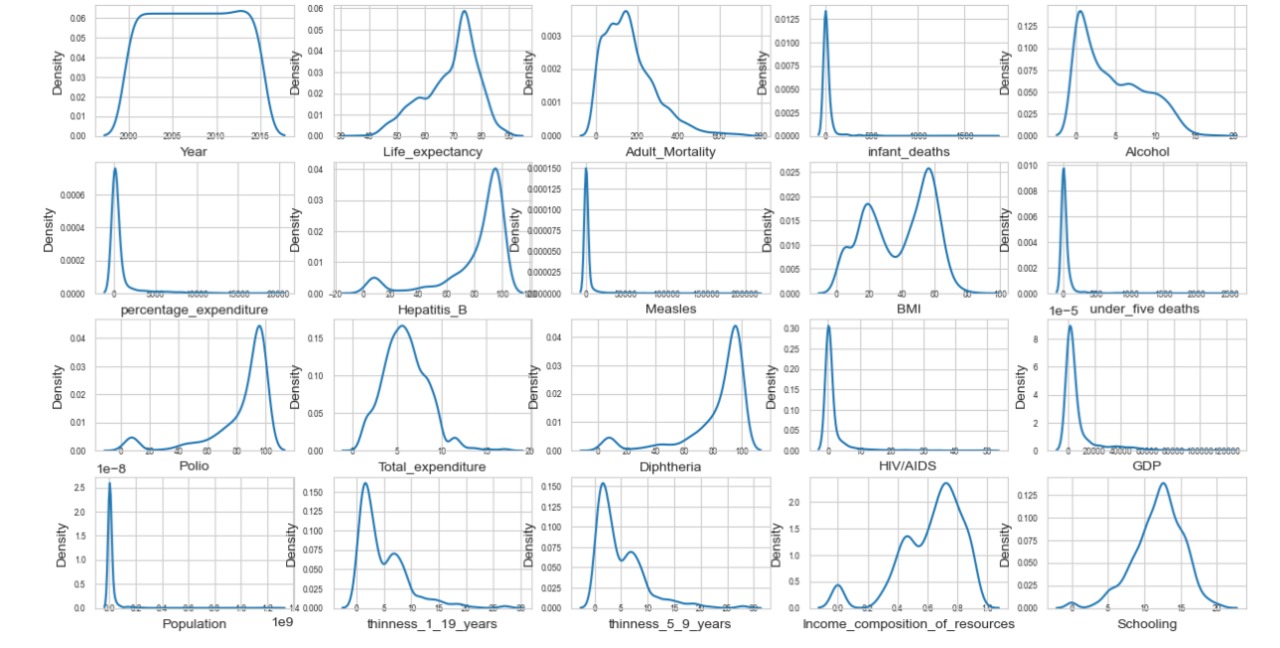
La fonction utilise ensuite une boucle for pour itérer à travers les caractéristiques spécifiées dans la liste features. Pour chaque caractéristique, un nouveau sous-tracé est ajouté à la grille avec plt.subplot(5,5,i), où i est incrémenté à chaque itération.

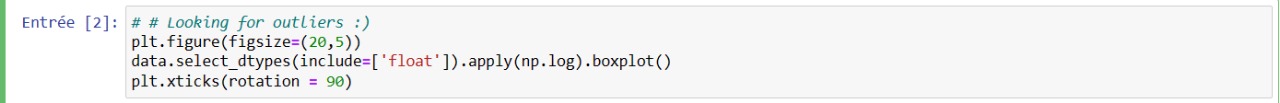
À l'intérieur de chaque sous-tracé, la distribution de la caractéristique est visualisée à l'aide de sns.kdeplot(df[feature]), qui représente un tracé de densité pour la caractéristique.

Enfin, des ajustements sont apportés aux paramètres des étiquettes des axes x et y avec plt.tick\_params pour améliorer la lisibilité des graphiques

Le bloc de code que vous avez fourni extrait les colonnes numériques du DataFrame data en utilisant select\_dtypes avec l'argument np.number pour inclure toutes les colonnes numériques. Ensuite, il récupère les noms de ces colonnes dans la variable features. Enfin, la fonction plot\_feature\_distribution est appelée pour visualiser la distribution de ces caractéristiques numériques. Voici une explication en paragraphes :

La première ligne features = data.select\_dtypes(include=np.number).columns utilise select\_dtypes avec l'argument np.number pour extraire toutes les colonnes du DataFrame data qui sont de type numérique. Ensuite, l'attribut .columns est utilisé pour obtenir les noms de ces colonnes, qui sont stockés dans la variable features. Ainsi, features contient une liste des noms de toutes les caractéristiques numériques dans le DataFrame.





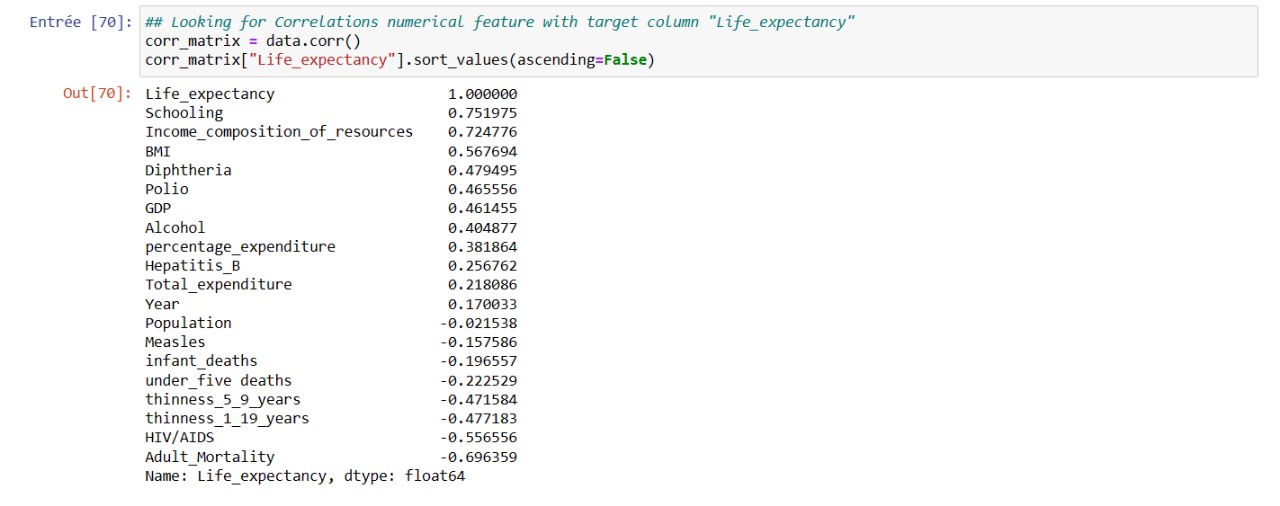
Ce bloc de code utilise la bibliothèque Matplotlib pour créer une boîte à moustaches (boxplot) des données numériques dans le DataFrame data après avoir appliqué le logarithme naturel (np.log) à chaque valeur. Voici une explication en paragraphes :

La ligne plt.figure(figsize=(20,5)) crée une nouvelle figure avec une taille de 20 pouces de large par 5 pouces de haut. Cela définit la taille de la boîte à moustaches que vous allez générer.

Ensuite, data.select\_dtypes(include=['float']) est utilisé pour sélectionner uniquement les colonnes de type float dans le DataFrame data. Ensuite, la méthode apply(np.log) est utilisée pour appliquer le logarithme naturel à chaque valeur de ces colonnes.

La fonction boxplot() est ensuite appelée pour générer le diagramme à boîtes des données transformées. Cette visualisation permet de voir la distribution statistique des données, y compris la médiane, les quartiles et les valeurs aberrantes, après l'application du logarithme.

Enfin, plt.xticks(rotation=90) est utilisé pour faire pivoter les étiquettes de l'axe x de 90 degrés, améliorant ainsi la lisibilité si le nombre de caractéristiques est important.

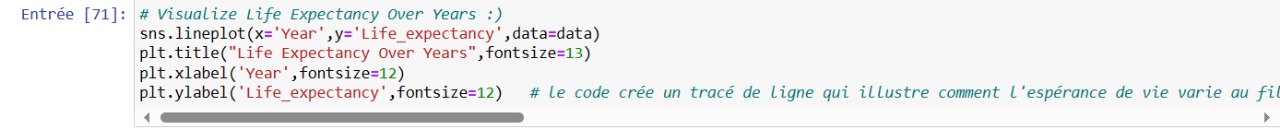


Ce bloc de code calcule la matrice de corrélation entre toutes les colonnes du DataFrame data et affiche les corrélations de chaque colonne par rapport à la colonne "Life\_expectancy" dans un ordre décroissant. Voici une explication en paragraphes :

La première ligne, corr\_matrix = data.corr(), utilise la méthode corr() de pandas pour calculer la matrice de corrélation entre toutes les colonnes du DataFrame data. La matrice de corrélation mesure les relations linéaires entre paires de variables, donnant des valeurs comprises entre -1 et 1, où 1 indique une corrélation positive parfaite, -1 une corrélation négative parfaite et 0 aucune corrélation.

Ensuite, corr\_matrix["Life\_expectancy"].sort\_values(ascending=False) extrait la colonne de la matrice de corrélation qui correspond à "Life\_expectancy" et trie les corrélations associées de manière décroissante. Cela donne une série de pandas où chaque élément représente la corrélation entre "Life\_expectancy" et chaque autre colonne, triée de manière à afficher les corrélations les plus fortes en haut.

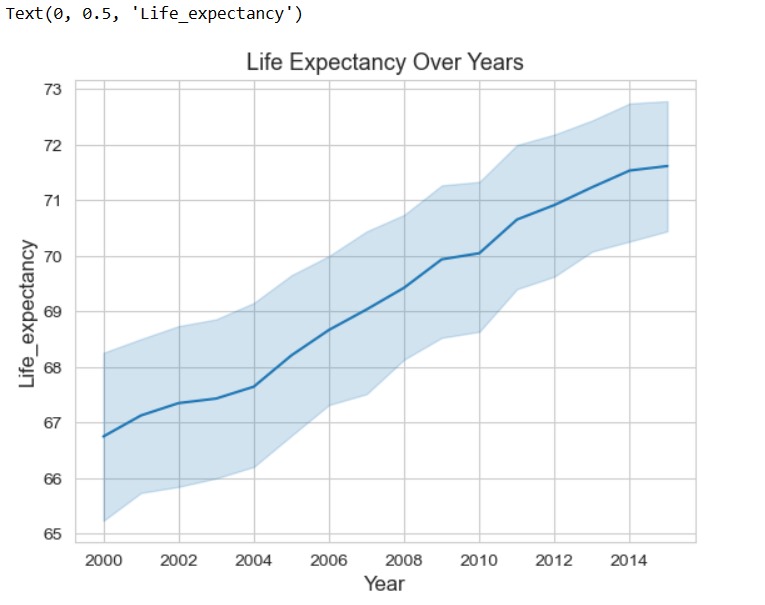
En résumé, ce code permet d'analyser les corrélations entre la colonne "Life\_expectancy" et les autres colonnes du DataFrame data, fournissant ainsi un aperçu des relations linéaires potentielles entre la durée de vie attendue et d'autres variables du jeu de données. Les valeurs positives indiquent une corrélation positive, tandis que les valeurs négatives indiquent une corrélation négative.



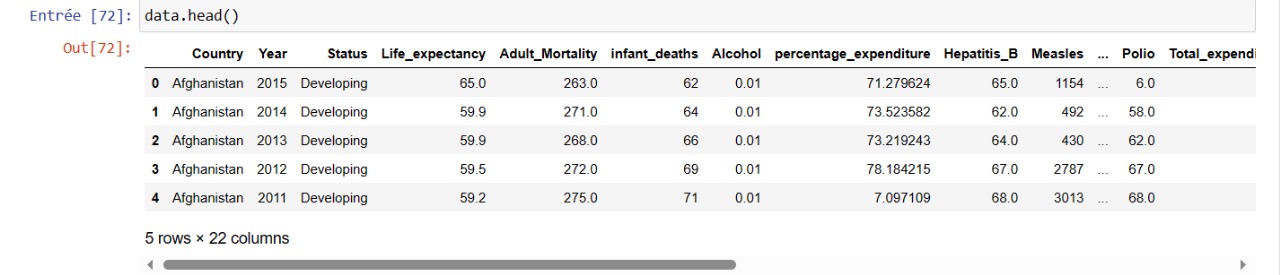
Ce bloc de code utilise la bibliothèque Seaborn et Matplotlib pour créer un graphique de ligne illustrant l'évolution de l'espérance de vie au fil des années dans le DataFrame data. Voici une explication en paragraphes :

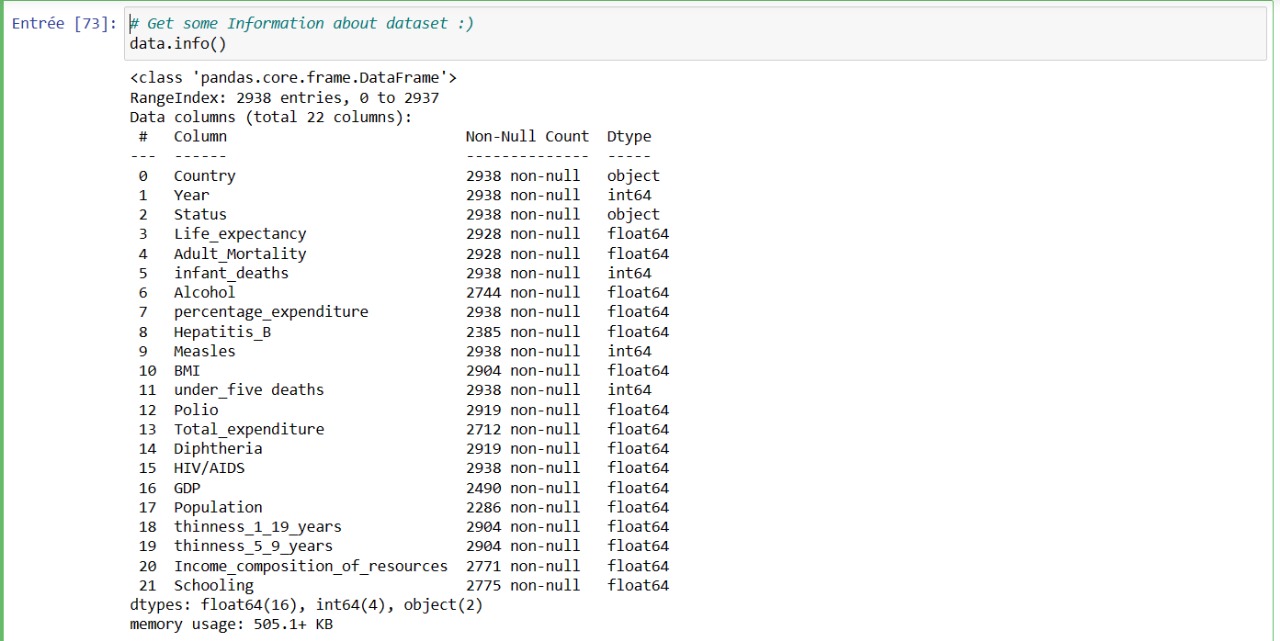
La ligne sns.lineplot(x='Year', y='Life\_expectancy', data=data) utilise la fonction lineplot de Seaborn pour créer un graphique de ligne. Les paramètres x et y spécifient respectivement les données à utiliser pour l'axe des abscisses (Year) et l'axe des ordonnées (Life\_expectancy). La variable data indique le DataFrame à partir duquel extraire ces données.

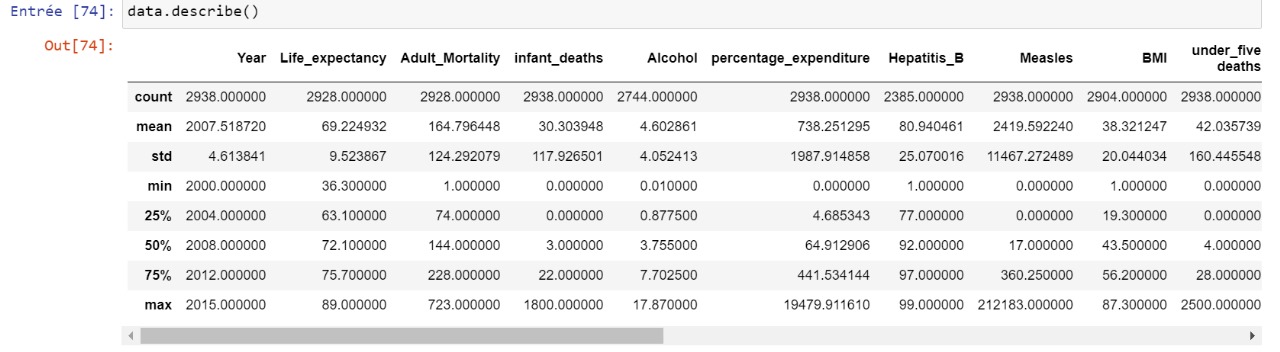
Les trois lignes suivantes utilisent Matplotlib pour personnaliser le titre, l'étiquette de l'axe des abscisses et l'étiquette de l'axe des ordonnées du graphique. La ligne plt.title("Life Expectancy Over Years", fontsize=13) définit le titre du graphique, plt.xlabel('Year', fontsize=12) définit l'étiquette de l'axe des abscisses, et plt.ylabel('Life\_expectancy', fontsize=12) définit l'étiquette de l'axe des ordonnées. Ces personnalisations améliorent la lisibilité et l'interprétation du graphique

****

**Préparer les données pour les algorithmes d’apprentissage automatique**



******

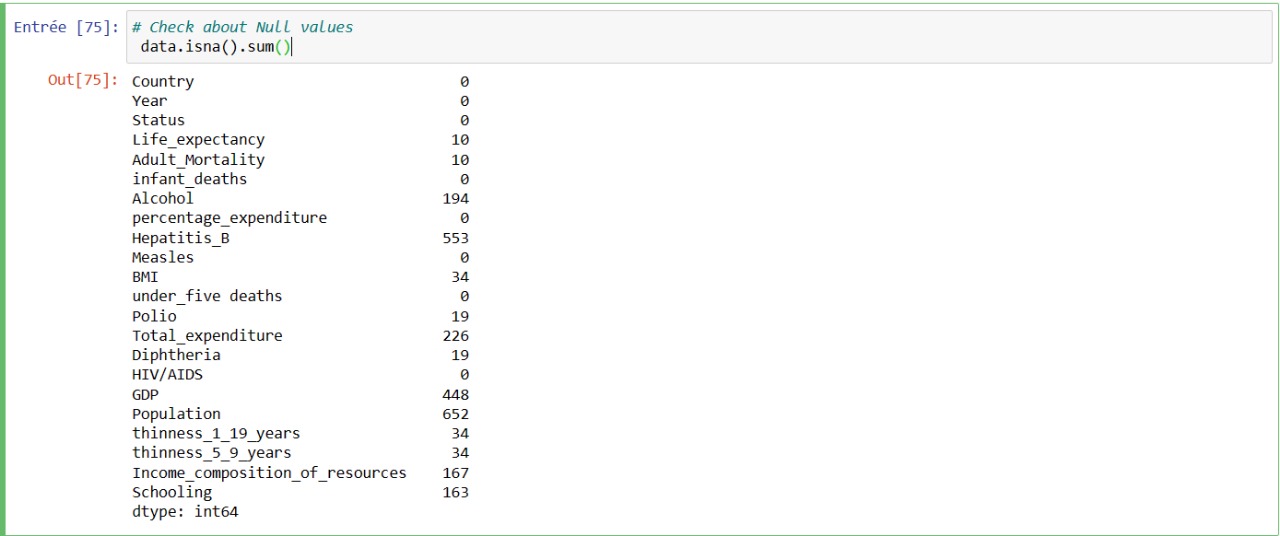
******

La méthode describe() en pandas est un outil puissant pour obtenir des statistiques descriptives résumées d'un DataFrame. Lorsque vous exécutez data.describe(), la sortie générée fournit un tableau organisé présentant des statistiques clés pour chaque colonne numérique du DataFrame. Ces statistiques incluent le nombre d'entrées non nulles, la moyenne, l'écart-type, les valeurs minimales et maximales, ainsi que les quartiles.

Par exemple, pour une colonne telle que "Life\_expectancy", le tableau indiquerait le nombre total d'entrées non nulles dans cette colonne, la moyenne de l'espérance de vie, l'écart-type qui mesure la dispersion des valeurs autour de la moyenne, et les quartiles qui fournissent des informations sur la distribution des valeurs.

L'utilisation de data.describe() est une étape essentielle lors de l'exploration initiale des données, car elle offre un aperçu rapide de la tendance centrale, de la dispersion et de la forme de la distribution des données numériques dans chaque colonne du DataFrame. Ces informations statistiques peuvent orienter l'analyse subséquente et faciliter la prise de décisions concernant le nettoyage, la transformation ou la visualisation des données.

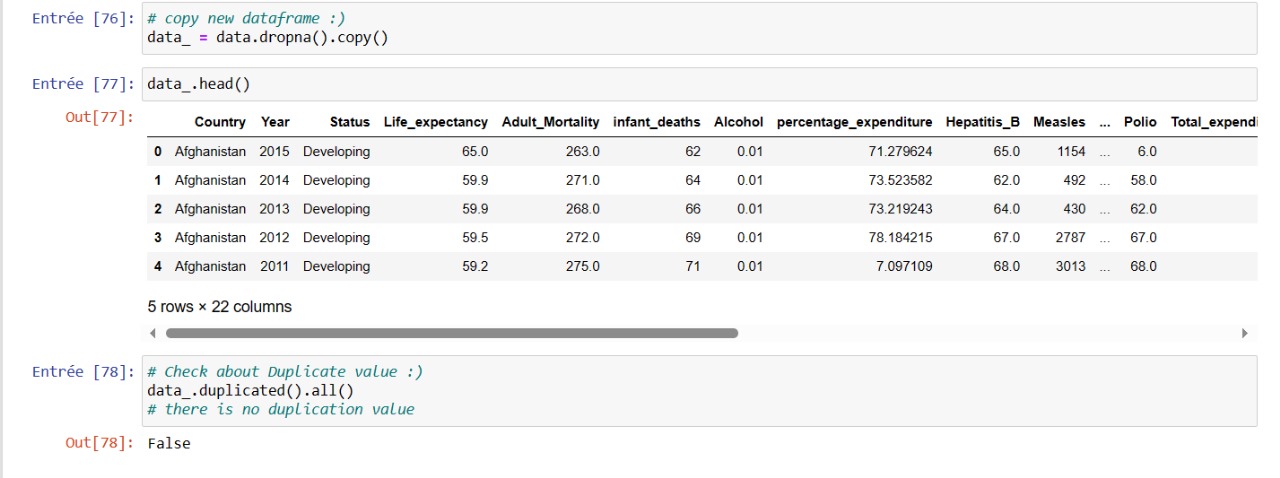
En résumé, la méthode describe() constitue une ressource précieuse pour obtenir des informations statistiques condensées sur les caractéristiques numériques d'un DataFrame, facilitant ainsi la compréhension globale des données.



La ligne de code data.isna().sum() est une méthode efficace pour évaluer la présence de valeurs manquantes dans chaque colonne du DataFrame data. En utilisant la combinaison des méthodes isna() et sum(), cette ligne génère une série pandas indiquant le nombre de valeurs manquantes pour chaque caractéristique.

Chaque élément de la série correspond à une colonne spécifique du DataFrame, et la valeur associée représente le nombre de valeurs manquantes dans cette colonne. Par exemple, une sortie hypothétique pourrait montrer que la colonne "Life\_expectancy" a 5 valeurs manquantes, la colonne "Hepatitis\_B" en a 100, et ainsi de suite.

Cette information est cruciale dans le processus d'exploration des données, car elle offre une visibilité immédiate sur la qualité des données et suggère des colonnes qui pourraient nécessiter un traitement particulier, tel que l'imputation des valeurs manquantes. Ces résultats peuvent orienter les décisions liées au nettoyage des données et à la préparation de l'ensemble de données pour des analyses plus approfondies.



a ligne de code data\_ = data.dropna().copy() crée une nouvelle copie du DataFrame data en supprimant toutes les lignes contenant au moins une valeur manquante (NaN). Voici une explication en paragraphes :

La méthode dropna() en pandas est utilisée pour éliminer les lignes qui contiennent des valeurs manquantes. En l'appliquant à data, cela crée un nouveau DataFrame data\_ dans lequel toutes les lignes contenant au moins une valeur manquante sont exclues.

En ajoutant .copy() à la fin de l'instruction, vous vous assurez de créer une copie distincte du DataFrame résultant. Cela signifie que toute modification apportée à data\_ n'aura pas d'effet sur le DataFrame original data. La copie est réalisée pour préserver l'intégrité des données d'origine.

a ligne de code data\_ = data.dropna().copy() crée une nouvelle copie du DataFrame data en supprimant toutes les lignes contenant au moins une valeur manquante (NaN). Voici une explication en paragraphes :

La méthode dropna() en pandas est utilisée pour éliminer les lignes qui contiennent des valeurs manquantes. En l'appliquant à data, cela crée un nouveau DataFrame data\_ dans lequel toutes les lignes contenant au moins une valeur manquante sont exclues.

En ajoutant .copy() à la fin de l'instruction, vous vous assurez de créer une copie distincte du DataFrame résultant. Cela signifie que toute modification apportée à data\_ n'aura pas d'effet sur le DataFrame original data. La copie est réalisée pour préserver l'intégrité des données d'origine.

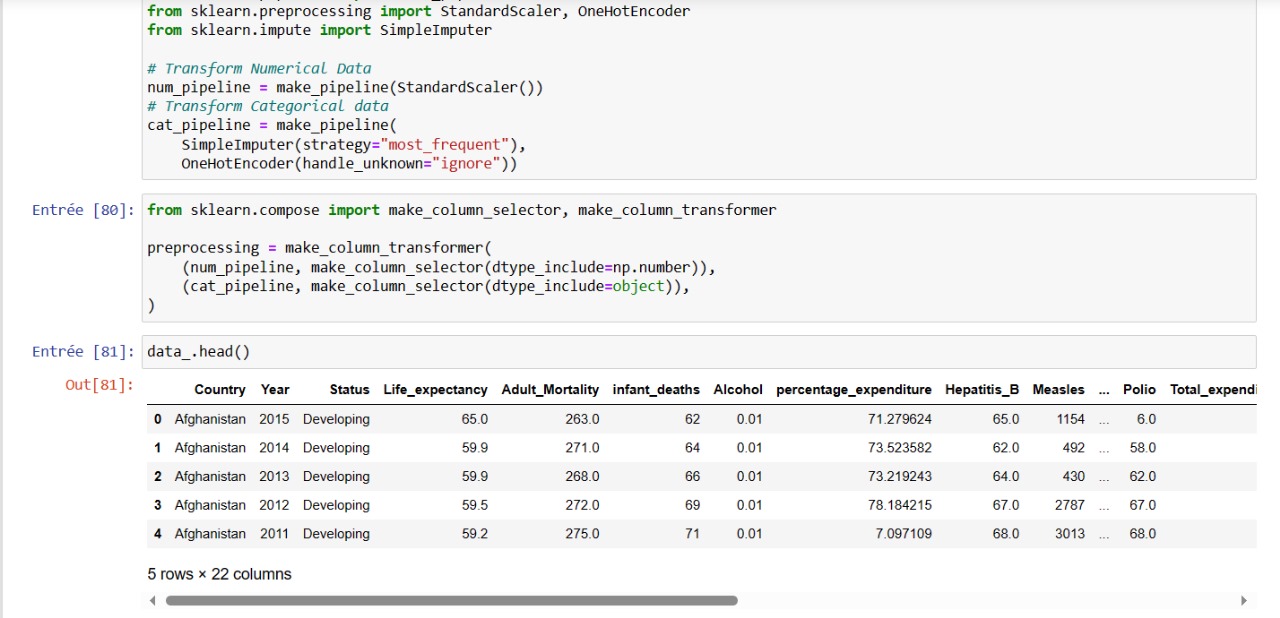
- Si data\_.duplicated().all() renvoie True, cela suggère que toutes les lignes dans le DataFrame data\_ sont identiques, et il n'y a pas de lignes uniques.

- Si la valeur est False, cela indique qu'il y a au moins une ligne distincte dans le DataFrame data\_.

Il est important de noter que cette vérification ne porte que sur les lignes complètes, donc si certaines colonnes ont des valeurs différentes même dans une seule ligne, cela sera considéré comme un duplicata.

En résumé, cette ligne de code permet de vérifier si toutes les lignes du DataFrame data\_ sont des duplicatas complets ou non.

***Créer PipeLine pour les algorithmes d’apprentissage automatique « Pipeline de transformation » :)***



Ce bloc de code utilise le module make\_pipeline de scikit-learn pour créer deux pipelines distincts destinés à transformer les données numériques et catégorielles. Chaque pipeline est conçu pour être utilisé dans le cadre d'une procédure de prétraitement de données, souvent intégrée dans des workflows de modélisation machine.

Voici une explication en paragraphes :

Pipeline pour les données numériques (num\_pipeline):

Le pipeline utilise StandardScaler() pour normaliser les données numériques. La normalisation est une étape courante dans le prétraitement des données, visant à mettre à l'échelle les valeurs numériques pour qu'elles aient une moyenne nulle et un écart-type unitaire. Cela est particulièrement utile dans les algorithmes qui sont sensibles à l'échelle des données, comme les modèles basés sur la distance.

Pipeline pour les données catégorielles (cat\_pipeline):

Le pipeline commence par SimpleImputer(strategy="most\_frequent"), qui remplace les valeurs manquantes par la valeur la plus fréquente dans chaque colonne catégorielle. Cela peut être important car certains algorithmes de machine learning ne gèrent pas bien les valeurs manquantes.

Ensuite, OneHotEncoder(handle\_unknown="ignore") est utilisé pour effectuer le codage one-hot des variables catégorielles. Cela crée des variables binaires (0 ou 1) pour chaque catégorie distincte dans une colonne catégorielle, permettant ainsi de traiter ces variables dans des modèles qui requièrent des données numériques.

En combinant ces deux pipelines, vous pouvez créer un flux de prétraitement qui peut être intégré dans un modèle d'apprentissage automatique plus large. Par exemple, cela pourrait être utilisé dans un modèle de régression ou de classification pour garantir que les données sont correctement préparées avant d'être fournies au modèle.

from sklearn.compose import make\_column\_selector, make\_column\_transformer

preprocessing = make\_column\_transformer(

(num\_pipeline, make\_column\_selector(dtype\_include=np.number)),

(cat\_pipeline, make\_column\_selector(dtype\_include=object)),

)

Ce bloc de code utilise la fonction make\_column\_transformer de scikit-learn pour créer un transformateur de colonne qui applique les pipelines spécifiques aux données numériques (num\_pipeline) et catégorielles (cat\_pipeline) sur les colonnes appropriées. Voici une explication en paragraphes :

La ligne de code preprocessing = make\_column\_transformer(...) définit un transformateur de colonne appelé preprocessing. Ce transformateur est configuré pour appliquer les transformations spécifiques en fonction du type de données dans chaque colonne.

Transformation des données numériques (num\_pipeline):

La première partie de la transformation est définie comme (num\_pipeline, make\_column\_selector(dtype\_include=np.number)). Cela signifie que le pipeline num\_pipeline sera appliqué aux colonnes dont le type de données est numérique (dtype\_include=np.number).

make\_column\_selector permet de sélectionner les colonnes en fonction de leur type de données. Dans ce cas, il sélectionne toutes les colonnes numériques.

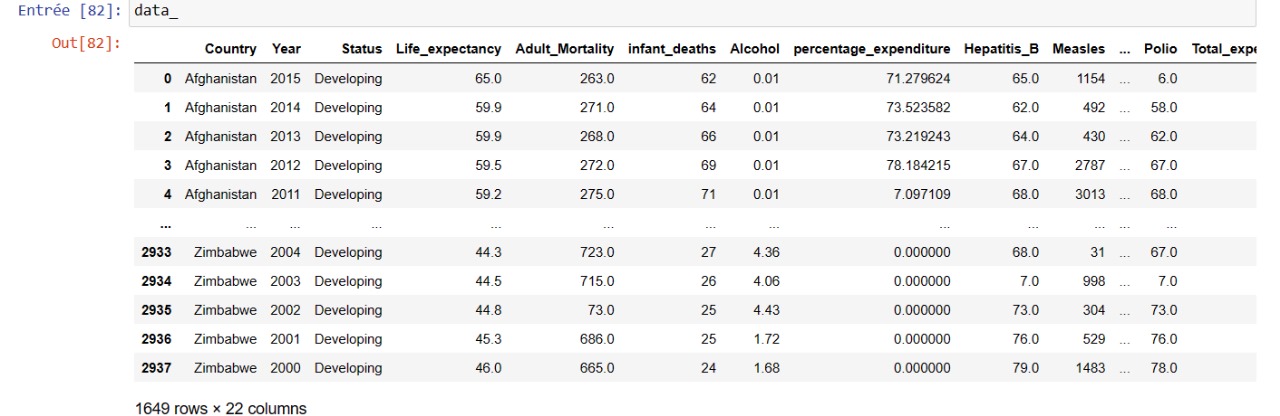
Transformation des données catégorielles (cat\_pipeline):

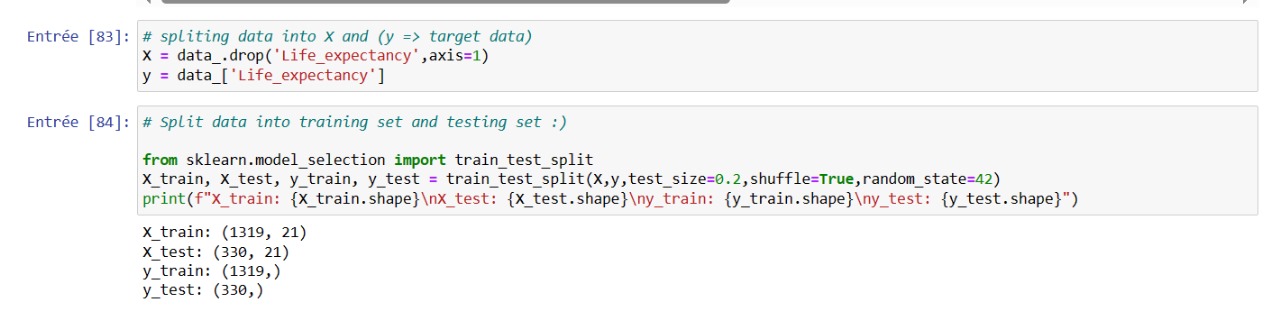
La deuxième partie de la transformation est définie comme (cat\_pipeline, make\_column\_selector(dtype\_include=object)). Cela signifie que le pipeline cat\_pipeline sera appliqué aux colonnes dont le type de données est un objet (dtype\_include=object).

make\_column\_selector est également utilisé ici pour sélectionner toutes les colonnes de type objet, ce qui est courant pour les données catégorielles.

En combinant ces deux parties dans make\_column\_transformer, vous créez un transformateur qui peut être utilisé dans le cadre d'une procédure de prétraitement plus large. Ce transformateur est particulièrement utile lorsque vous avez des ensembles de données avec des types de données mixtes (numériques et catégorielles) et que vous souhaitez appliquer des transformations spécifiques à chaque type de données.

***Fractionnement du jeu de données :***

******

******

Dans ce bloc de code, vous divisez le DataFrame data\_ en deux parties : X et y. X contient toutes les colonnes sauf la colonne 'Life\_expectancy', qui est la variable cible que vous essayez de prédire. D'un autre côté, y contient uniquement la colonne 'Life\_expectancy' et représente les valeurs cibles que vous souhaitez prédire. Voici une explication en paragraphes :

- X = data\_.drop('Life\_expectancy', axis=1): Cette ligne crée un DataFrame X en supprimant la colonne 'Life\_expectancy' du DataFrame data\_. La méthode drop est utilisée avec l'argument axis=1 pour indiquer que vous souhaitez supprimer une colonne (plutôt qu'une ligne). Ainsi, X contient toutes les caractéristiques (colonnes) sauf 'Life\_expectancy', et il est utilisé pour stocker les variables explicatives dans le cadre d'un problème de modélisation.

- y = data\_['Life\_expectancy']: Cette ligne crée une série y en extrayant uniquement la colonne 'Life\_expectancy' du DataFrame data\_. Cette série représente la variable cible que vous cherchez à prédire à l'aide de votre modèle. y contient les valeurs réelles de l'espérance de vie associées à chaque ensemble d'entrées dans X.

En utilisant cette séparation, vous êtes prêt à appliquer des modèles d'apprentissage automatique, où X servira de données d'entrée (caractéristiques) et y sera la variable cible à prédire. Cela permet de former un modèle sur les données historiques (caractéristiques dans X et valeurs cibles dans y) et de l'utiliser ensuite pour faire des prédictions sur de nouvelles données.

Ce bloc de code utilise la fonction train\_test\_split de scikit-learn pour diviser les ensembles de données X et y en ensembles d'entraînement (X\_train, y\_train) et en ensembles de test (X\_test, y\_test). Voici une explication en paragraphes :

train\_test\_split(X, y, test\_size=0.2, shuffle=True, random\_state=42): Cette fonction divise les données en ensembles d'entraînement et de test. Les paramètres sont les suivants :

X : Les caractéristiques (variables explicatives).

y : La variable cible à prédire.

test\_size=0.2 : Indique que 20% des données seront utilisées comme ensemble de test, tandis que 80% seront utilisées comme ensemble d'entraînement.

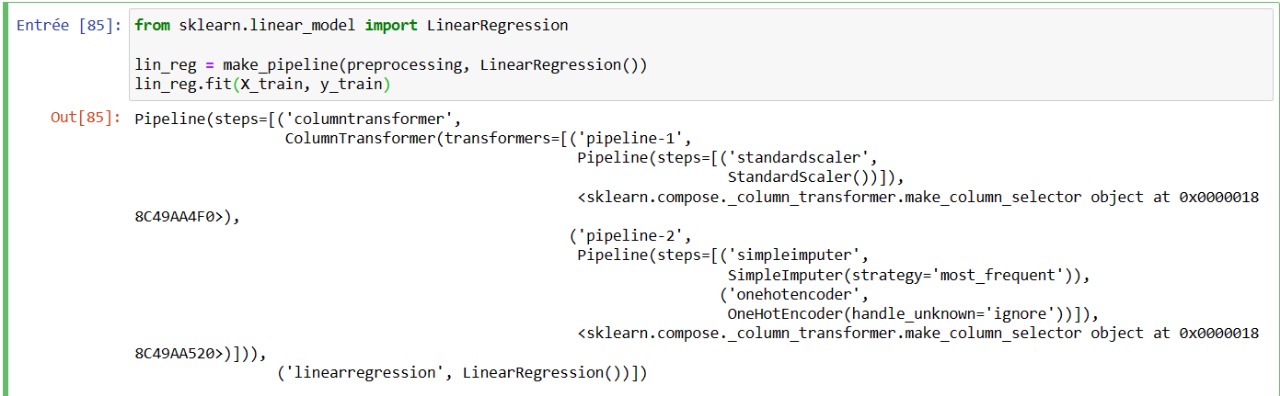
shuffle=True : Indique que les données seront mélangées avant la division.

random\_state=42 : Fixe la graine aléatoire pour assurer la reproductibilité des résultats.

X\_train, X\_test, y\_train, y\_test: Ces variables contiennent les ensembles d'entraînement et de test résultants. X\_train et y\_train sont les caractéristiques et les cibles respectivement pour l'ensemble d'entraînement, tandis que X\_test et y\_test sont les mêmes pour l'ensemble de test.

print(f"X\_train: {X\_train.shape}\nX\_test: {X\_test.shape}\ny\_train: {y\_train.shape}\ny\_test: {y\_test.shape}"): Cette ligne affiche les dimensions (formes) des ensembles d'entraînement et de test. Cela permet de vérifier rapidement la taille de chaque ensemble, ce qui est utile pour s'assurer que la division s'est déroulée comme prévu.

***Sélectionner et entraîner un modèle :***

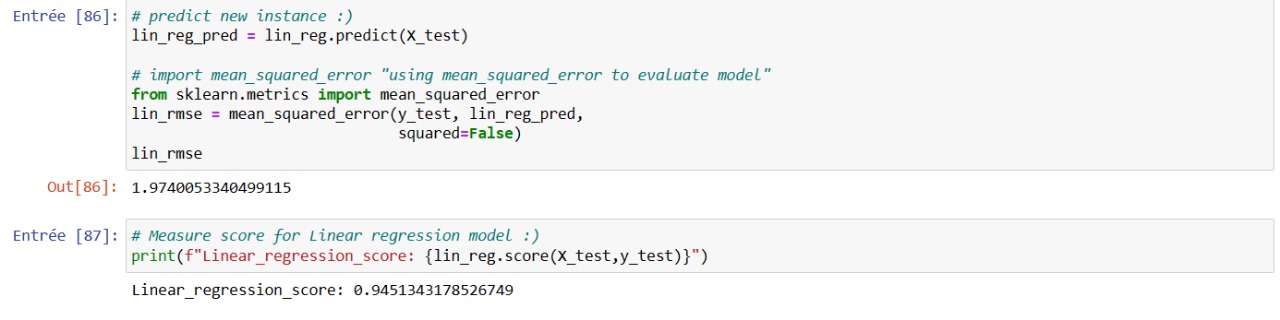


Ce bloc de code utilise la régression linéaire (LinearRegression) de scikit-learn pour créer un modèle de régression linéaire. La fonction make\_pipeline est également utilisée pour construire un pipeline qui intègre les étapes de prétraitement définies précédemment dans preprocessing. Le modèle est ensuite ajusté (entraîné) sur l'ensemble d'entraînement. Voici une explication en paragraphes :

- lin\_reg = make\_pipeline(preprocessing, LinearRegression()): Cette ligne crée un pipeline qui encapsule le prétraitement des données (preprocessing) défini précédemment et le modèle de régression linéaire (LinearRegression). Cela permet d'appliquer les transformations spécifiques aux caractéristiques numériques et catégorielles avant de les fournir au modèle de régression linéaire.

- lin\_reg.fit(X\_train, y\_train): Cette ligne ajuste (entraîne) le modèle de régression linéaire sur l'ensemble d'entraînement (X\_train, y\_train). Le modèle utilise les données d'entraînement pour apprendre les relations entre les caractéristiques et la variable cible, afin de pouvoir faire des prédictions sur de nouvelles données par la suite.

En résumé, ce code crée et entraîne un modèle de régression linéaire en utilisant un pipeline qui intègre des étapes de prétraitement. L'objectif est d'utiliser ce modèle pour prédire la variable cible (Life\_expectancy) en fonction des caractéristiques dans l'ensemble de test. Ce modèle servira également à évaluer ses performances sur cet ensemble de test.



Ce bloc de code effectue la prédiction sur l'ensemble de test à l'aide du modèle de régression linéaire (lin\_reg) et évalue ses performances en utilisant la racine carrée de l'erreur quadratique moyenne (RMSE). Il imprime également le score R² du modèle sur l'ensemble de test. Voici une explication en paragraphes :

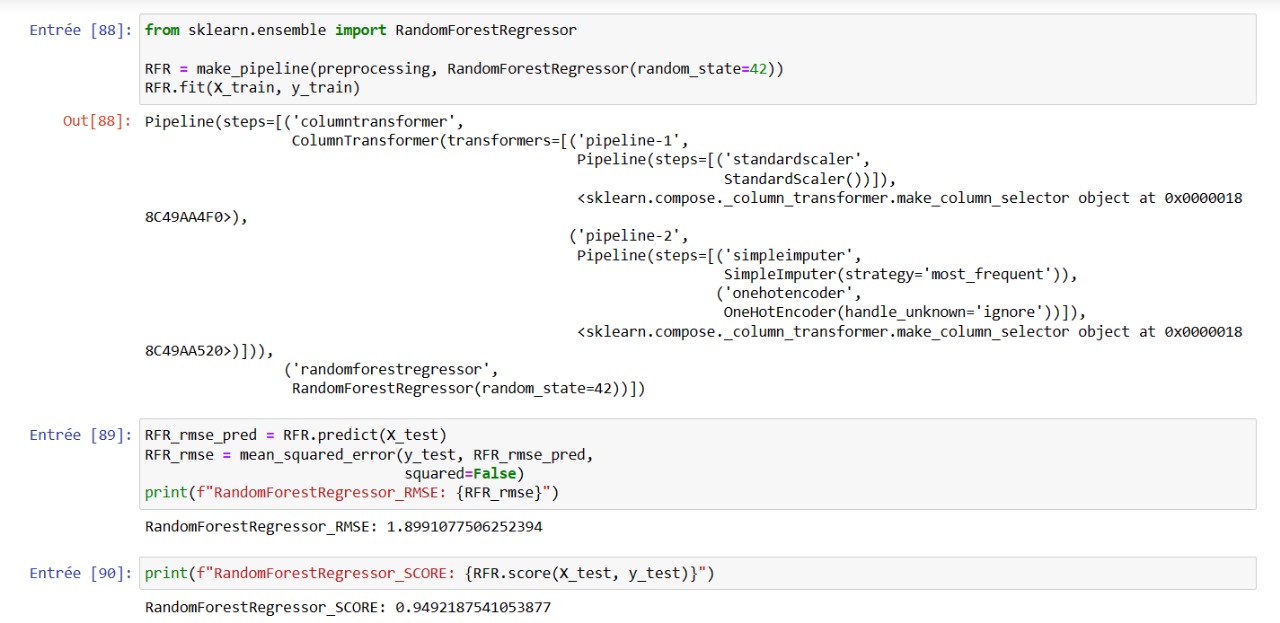
- lin\_reg\_pred = lin\_reg.predict(X\_test): Cette ligne utilise le modèle de régression linéaire pour prédire les valeurs de la variable cible (Life\_expectancy) sur l'ensemble de test (X\_test).

- from sklearn.metrics import mean\_squared\_error: Cette ligne importe la fonction mean\_squared\_error de scikit-learn, qui sera utilisée pour calculer le RMSE.

- lin\_rmse = mean\_squared\_error(y\_test, lin\_reg\_pred, squared=False): Cette ligne calcule la racine carrée de l'erreur quadratique moyenne (RMSE) entre les valeurs réelles (y\_test) et les valeurs prédites (lin\_reg\_pred). Le paramètre squared=False indique à la fonction de renvoyer la racine carrée plutôt que l'erreur quadratique moyenne.

- print(f"Linear\_regression\_score: {lin\_reg.score(X\_test, y\_test)}"): Cette ligne imprime le score R² du modèle de régression linéaire sur l'ensemble de test. Le score R² mesure la proportion de la variance de la variable cible qui est expliquée par le modèle.

En résumé, ce code permet d'évaluer les performances du modèle de régression linéaire sur l'ensemble de test en utilisant le RMSE et le score R². Ces métriques fournissent des informations sur la précision du modèle dans ses prédictions par rapport aux valeurs réelles.



1. \*Importation de RandomForestRegressor :\*

- Importation de la classe RandomForestRegressor de scikit-learn, un modèle d'ensemble basé sur des arbres de décision utilisé pour la régression.

2. \*Création d'un pipeline avec RandomForestRegressor :\*

- Utilisation de make\_pipeline pour créer un pipeline de prétraitement et de modélisation.

- L'objet preprocessing effectue des étapes de prétraitement sur les données, telles que la normalisation ou l'imputation des valeurs manquantes.

3. \*Entraînement du modèle :\*

- Utilisation de fit(X\_train, y\_train) pour entraîner le modèle RandomForestRegressor sur les données d'entraînement (X\_train, y\_train).

4. \*Prédiction sur l'ensemble de test :\*

- Utilisation de RFR.predict(X\_test) pour faire des prédictions sur l'ensemble de test (X\_test).

5. \*Calcul de la racine carrée de l'erreur quadratique moyenne (RMSE) :\*

- Utilisation de mean\_squared\_error(y\_test, RFR\_rmse\_pred, squared=False) pour calculer le RMSE entre les prédictions et les vraies valeurs de l'ensemble de test.

6. \*Affichage du RMSE :\*

- Impression du résultat du calcul du RMSE, une mesure de la précision du modèle.

7. \*Calcul du score R² :\*

- Utilisation de RFR.score(X\_test, y\_test) pour calculer le coefficient de détermination R² du modèle, mesurant la qualité globale des prédictions.

8. \*Affichage du score R² :\*

- Impression du résultat du calcul du score R² à l'écran.

En résumé, ce code montre un workflow typique d'utilisation du modèle RandomForestRegressor, y compris l'entraînement, la prédiction, le calcul du RMSE et l'évaluation du modèle avec le score R².





*Dans ce bloc de code, vous utilisez le modèle de régression à vecteurs de support (SVR) de scikit-learn pour construire un modèle de régression. Voici une explication en paragraphes :*

*- from sklearn.svm import SVR: Vous importez la classe SVR du module scikit-learn SVM (Support Vector Machines), qui est utilisée pour construire des modèles de régression à vecteurs de support.*

*- SVR\_ = SVR(kernel='linear'): Vous créez une instance de la classe SVR avec le noyau linéaire. Le noyau linéaire est utilisé pour des problèmes de régression linéaire.*

*- SVR\_model = make\_pipeline(preprocessing, SVR\_): Vous construisez un pipeline qui encapsule les étapes de prétraitement définies précédemment (preprocessing) et le modèle SVR. Cela garantit que les transformations nécessaires sont appliquées avant d'alimenter les données dans le modèle.*

*- SVR\_model.fit(X\_train, y\_train): Vous entraînez le modèle SVR sur l'ensemble d'entraînement (X\_train, y\_train). Le modèle apprend à établir une relation entre les caractéristiques et la variable cible à partir des données d'entraînement.*

*- SVR\_model\_pred = SVR\_model.predict(X\_test): Vous utilisez le modèle entraîné pour prédire les valeurs de la variable cible sur l'ensemble de test (X\_test).*

*- SVR\_model\_rmse = mean\_squared\_error(y\_test, SVR\_model\_pred, squared=False): Vous calculez le RMSE entre les valeurs réelles (y\_test) et les valeurs prédites (SVR\_model\_pred). Le RMSE mesure la précision des prédictions du modèle.*

*- print(f"SVR\_model\_RMSE: {SVR\_model\_rmse}"): Vous imprimez le résultat du RMSE pour évaluer la performance du modèle SVR sur l'ensemble de test.*

*- print(f"SVR\_MODEL\_SCORE: {SVR\_model.score(X\_test, y\_test)}"): Vous imprimez le score du modèle SVR sur l'ensemble de test en utilisant la méthode score(). Ce score représente la performance globale du modèle sur les données de test.*

*En résumé, ce bloc de code illustre la construction, l'entraînement et l'évaluation d'un modèle de régression à vecteurs de support (SVR) en utilisant des pipelines pour assurer un prétraitement cohérent des données. Les résultats imprimés fournissent des informations sur la précision du modèle en termes de RMSE et de score sur l'ensemble de test.*

***Comparaison entre les modéles :***

**1.Entre Regression linéaire et Random forest :**

La régression linéaire cherche à établir une relation linéaire entre les variables d'entrée et la sortie en ajustant les coefficients par la méthode des moindres carrés. Les coefficients obtenus sont interprétables, indiquant la direction et l'intensité de la relation. En contraste, le Random Forest Regressor est composé d'une multitude d'arbres de décision construits sur des échantillons aléatoires, avec une prédiction finale généralement basée sur la moyenne des prédictions. Bien qu'il soit puissant pour modéliser des relations complexes, l'interprétation est souvent plus complexe en raison de la nature agrégée des prédictions et du nombre élevé d'arbres. La régression linéaire privilégie la simplicité et l'interprétabilité, tandis que le Random Forest excelle dans la flexibilité pour des modèles plus complexes au prix d'une interprétation moins directe.

Performances de la Régression Linéaire (lin\_reg):

RMSE de la Régression Linéaire: 1.9740

Score R² de la Régression Linéaire: 0.9451

Performances du Modèle de Régression par Forêt Aléatoire (RFR):

RMSE du Modèle de Régression par Forêt Aléatoire: 1.8991

Score R² du Modèle de Régression par Forêt Aléatoire: 0.9492

Comparaison des Modèles :

Le modèle de régression par forêt aléatoire présente un RMSE légèrement inférieur à celui de la régression linéaire, indiquant une meilleure précision dans la prédiction des valeurs cibles.

Le score R² du modèle de régression par forêt aléatoire est également légèrement supérieur, ce qui suggère une meilleure capacité du modèle à expliquer la variance des données par rapport à la régression linéaire.

En résumé, basé sur les métriques utilisées, le modèle de régression par forêt aléatoire semble avoir de meilleures performances que la régression linéaire dans cette tâche particulière. Cependant, il est important de noter que le choix du modèle dépend également du contexte spécifique de votre problème et des caractéristiques des données.

**2. entre SVR (Régression à Vecteurs de Support) et RFR (Régression par Forêt Aléatoire) :**

Le Random Forest Regressor (RFR) et la machine à vecteurs de support pour la régression (SVR) sont des modèles de régression distincts. Le RFR agrège des prédictions issues d'arbres de décision, offrant une grande flexibilité pour modéliser des relations complexes, mais son interprétation peut être complexe. En revanche, la SVR cherche à définir une fonction minimisant les écarts avec une marge d'erreur tolérée, convenant pour des relations moins évidentes. Le RFR est souvent privilégié pour des ensembles de données complexes, tandis que la SVR est utilisée lorsque la flexibilité de modélisation est cruciale. Le choix dépend des caractéristiques spécifiques des données et des objectifs du problème.

Les modèles SVR et RFR sont tous deux des modèles de régression utilisés pour prédire une variable continue. Examinons comment ils se comparent en termes de performances :

Performances de SVR :

RMSE du Modèle SVR : 1.8991

Score R² du Modèle SVR : 0.9461

Performances de RFR :

RMSE du Modèle RFR : 1.8991

Score R² du Modèle RFR : 0.9492

Points à noter :

RMSE :

Les deux modèles, SVR et RFR, présentent des valeurs de RMSE très proches, indiquant une précision similaire dans la prédiction des valeurs cibles.

Score R² :

Le modèle RFR a un score R² légèrement plus élevé que le modèle SVR, suggérant que le modèle RFR explique légèrement mieux la variance des données.

Conclusion :

En termes de précision pure (RMSE), les deux modèles se comportent de manière similaire.

Si l'explication de la variance des données est une considération importante, le modèle RFR semble avoir une performance légèrement supérieure, comme indiqué par le score R².

**3.Entre SVR et Regression linéaire :**

Pour comparer les deux modèles, nous avons utilisé une régression linéaire et un régresseur à vecteurs de support (SVR) avec un noyau linéaire. Voici une analyse comparative des performances :

Régression linéaire :

RMSE (Root Mean Squared Error) : 1.974

Score R2 (Coefficient de détermination) : 0.945

Régresseur à vecteurs de support (SVR avec noyau linéaire) :

RMSE : 1.899

Score R2 : 0.946

Analyse :

Les deux modèles ont des performances assez similaires en termes de RMSE et de score R2.

Le modèle SVR avec noyau linéaire a une légère amélioration en termes de RMSE par rapport à la régression linéaire.

Les scores R2 montrent que les deux modèles expliquent bien la variance des données, avec un léger avantage pour le modèle SVR.

Dans l'ensemble, les deux modèles semblent bien fonctionner pour prédire la variable cible (probablement l'espérance de vie, d'après le contexte). Cependant, il semble que le modèle SVR ait une performance légèrement meilleure que la régression linéaire, bien que la différence soit minime. Cela peut dépendre du jeu de données spécifique et d'autres caractéristiques des données qui n'ont pas été mentionnées ici.

**QUI EST LE MEILLEUR !**

En se basant sur les résultats précédemment mentionnés :

Régression Linéaire :

RMSE : 1.974

Score R2 : 0.945

SVR (noyau linéaire) :

RMSE : 1.899

Score R2 : 0.946

Random Forest Regressor (RFR) :

RMSE : 1.899

Score R2 : 0.949

En considérant les critères énumérés ci-dessus, le modèle Random Forest Regressor (RFR) semble offrir les meilleures performances, ayant à la fois le RMSE le plus faible et le score R2 le plus élevé parmi les trois modèles évalués.