

M2 en Informatique
Exploration Informatique des données et décisionnel
(EID²)

Rapport Du Projet

Statistique Exploratoire Multidimensionnelle

Réduction de dimensionnalité avec
Feature Selection

Réalisé par :

- **Doha BENZINEB**
- **Kawtar MEDRARE**
- **Mohamed BOUIBAUAN**
- **El Hassan LACHKAR**
- **Aissam ABOUD**
- **Ismail BASLAM**
- **Bilal BOUDJEMA**
- **Anis SAIDI**
- **Anis TERTAKI**

Encadré par :

- **Pr. Thomas PAPASTERGIOU**

Table des matières

1	- INTRODUCTION	3
2	- CONTEXTUALISATION DU PROJET	4
2.1	COMPRÉHENSION DE LA MÉTHODE DE RÉDUCTION DE DIMENSIONNALITÉ PAR SÉLECTION DE CARACTÉRISTIQUES (RDD) :	
	FEATURE SELECTION	5
2.1.1	<i>Principe de la Méthode de RDD - Feature Selection :</i>	5
2.1.2	<i>Explication en Termes Accessibles :</i>	5
2.1.3	<i>Explication Détaillée pour un Public Expert :</i>	5
2.1.4	<i>Algorithme et Application Pratique :</i>	5
2.1.5	<i>Avantages et Inconvénients :</i>	5
2.2	ALGORITHME DE LA SÉLECTION DE CARACTÉRISTIQUES :	5
2.2.1	<i>Approches filtres :</i>	5
2.2.1.1	Critère de Fisher	6
2.2.1.2	Critère de Variance	6
2.2.1.3	Critère d'information mutuelle	6
2.2.2	<i>Approches enveloppantes</i>	6
2.2.2.1	Sequential forward selection (SFS).....	7
2.2.2.2	Sequential backward selection (SBS).....	7
2.2.3	<i>Approches intégrées</i>	8
2.3	SÉLECTION DES DONNÉES POUR L'ÉVALUATION DE LA MÉTHODE DE RDD :	9
2.3.1	<i>Adéquation des Données Choisis :</i>	9
2.3.2	<i>Nature des Données :</i>	10
2.3.3	<i>Quantité de Données :</i>	10
3	- RÉALISATION ET MANIPULATION	11
3.1	ETAPE 1: EXPLORATION DES DEUX DATASETS (VISUALISATION, PREPROCESSING, CLEANING).....	12
3.1.1	<i>Iris Dataset</i>	12
3.1.1.1	Visualisation	12
3.1.1.2	Etude statistique de Iris dataset	19
3.1.2	<i>Stock Dataset</i>	23
3.1.2.1	Chargement des données	23
3.1.2.2	Concaténation des datasheets Apple, Google, Tesla, Amazon	23
3.1.2.3	Data preprocessing.....	24
3.1.2.4	Visualisation et étude statique de dataset	28
3.2	ETAPE2: FEATURE SELECTION POUR LES DEUX DATASET	33
3.2.1	<i>Iris Dataset</i>	33
3.2.1.1	La normalisation des données Iris	33
3.2.1.2	Les Résultats de La Classification Avant Feature selection avec (KNN, SVM, Logistic Regression, Gradient Boosting, Decision Tree)	34
3.2.1.3	La sélection des caractéristiques	37
3.2.2	<i>Stock Dataset (prediction de prix)</i>	47
3.2.2.1	Preprocessing	47
3.2.2.2	Avant la sélection des caractéristiques	48
3.2.2.3	La sélection des caractéristiques	53
3.3	TABLEAUX DE COMPARAISON	70
3.3.1	<i>Iris Dataset (classification)</i>	70
3.3.2	<i>Stock Data (Regression).....</i>	71
4	- CONCLUSION :	72

1 - Introduction

Dans le cadre dynamique de l'analyse contemporaine de données, ce rapport se positionne comme une exploration approfondie de la méthodologie de réduction de la dimensionnalité, avec un accent particulier sur la sélection de caractéristiques, également connue sous le nom de "feature selection". L'essence fondamentale de ce projet réside dans la recherche d'une compréhension exhaustive de cette méthode, cherchant à dévoiler les nuances de ses principes théoriques tout en illustrant sa mise en œuvre pratique. Ce travail s'étend au-delà des limites conventionnelles en fusionnant deux ensembles de données distincts, chacun représentant un défi unique : l'un dédié à la classification des iris, une tâche classique en apprentissage automatique, et l'autre axé sur la prédiction des prix d'actions, un domaine exigeant des capacités prédictives avancées. Cette dualité d'applications offre une perspective élargie sur la pertinence et l'efficacité de la feature selection dans des contextes diversifiés et exigeants. Notre approche méthodique inclut une sélection minutieuse de données pour garantir la représentativité des cas d'utilisation, une implémentation rigoureuse de la méthode, et une comparaison systématique avec d'autres techniques de réduction de dimensionnalité. L'objectif ultime de cette étude est d'évaluer la portée et l'impact de la feature selection, contribuant ainsi de manière substantielle à la compréhension et à l'application pratique des méthodes de réduction de dimensionnalité dans le paysage complexe de l'analyse de données contemporaine.

2 - Contextualisation du projet

2.1 Compréhension de la Méthode de Réduction de Dimensionnalité par Sélection de Caractéristiques (RdD) : Feature Selection

2.1.1 Principe de la Méthode de RdD - Feature Selection :

La feature selection, en tant que technique de réduction de dimensionnalité, s'appuie sur la présomption que toutes les caractéristiques ne contribuent pas également à la variabilité des données. Le principe sous-jacent est d'identifier et de conserver uniquement les caractéristiques les plus informatives pour réduire la complexité tout en préservant la pertinence des données.

2.1.2 Explication en Termes Accessibles :

À un niveau accessible, la feature selection peut être assimilée à une tâche de tri minutieux, où l'objectif est de choisir les caractéristiques les plus significatives, tout comme on sélectionnerait les ingrédients essentiels dans une recette pour en garantir la réussite.

2.1.3 Explication Détaillée pour un Public Expert :

Pour les étudiants avancés, la méthode de feature selection repose sur des mesures de pertinence comme l'information mutuelle ou la corrélation. Elle permet de choisir les variables les plus informatives tout en réduisant la redondance.

2.1.4 Algorithme et Application Pratique :

L'algorithme de Recursive Feature Elimination (RFE) sera explicité, mettant en évidence comment il élimine de manière itérative les caractéristiques moins importantes. Une application pratique pourrait consister à utiliser la feature selection pour identifier les biomarqueurs les plus pertinents dans des ensembles de données médicales ou par exemple on pourrait l'utiliser pour sélectionner les gènes les plus pertinents dans l'analyse de données génomiques.

2.1.5 Avantages et Inconvénients :

Les avantages incluent une réduction significative de la dimensionnalité, une amélioration de la performance des modèles, et une meilleure compréhension des relations entre les variables. Cependant, cela peut conduire à la perte d'informations importantes, surtout si les variables sont interdépendantes.

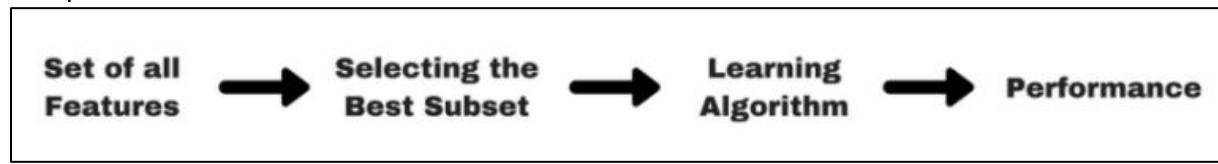
2.2 Algorithme de la Sélection de Caractéristiques :

2.2.1 Approches filtres :

Les méthodes de filtrage sont généralement utilisées comme étape de prétraitement. La sélection des fonctionnalités est indépendante de tout algorithme d'apprentissage automatique. Au lieu de cela, les caractéristiques sont sélectionnées sur la base de leurs scores dans divers tests statistiques pour leur corrélation avec la variable de résultat. Les caractéristiques de ces méthodes sont les suivantes :

- Ces méthodes reposent sur les caractéristiques des données (feature features)
- Ils n'utilisent pas d'algorithmes d'apprentissage automatique.
- Ce sont des modèles agnostiques.
- Ils ont tendance à être moins coûteux en calcul.
- Ils donnent généralement des performances de prédiction inférieures à celles des méthodes wrapper.

- Ils sont très bien adaptés pour un dépistage rapide et la suppression des fonctionnalités non pertinentes.



2.2.1.1 Critère de Fisher

La méthode ANOVA F-value estime le degré de linéarité entre la caractéristique d'entrée (c'est-à-dire le prédicteur) et la caractéristique de sortie. Une valeur F élevée indique un haut degré de linéarité et une faible valeur F indique un faible degré de linéarité. Le principal inconvénient de l'utilisation de la valeur F ANOVA est qu'elle ne capture que les relations linéaires entre les caractéristiques d'entrée et de sortie. En d'autres termes, aucune relation non linéaire ne peut être détectée par la valeur F.

2.2.1.2 Critère de Variance

La méthode du seuil de variance supprime les entités dont la variance est inférieure à une valeur limite prédéfinie. Il est basé sur la notion que les caractéristiques qui ne varient pas beaucoup en elles-mêmes ont un faible pouvoir prédictif. La principale faiblesse du seuil de variance est qu'il ne tient pas compte de la relation entre les caractéristiques d'entrée et la caractéristique de sortie. Il convient de noter qu'avant d'effectuer le seuillage de la variance, toutes les caractéristiques doivent être standardisées afin qu'elles aient la même échelle.

2.2.1.3 Critère d'information mutuelle

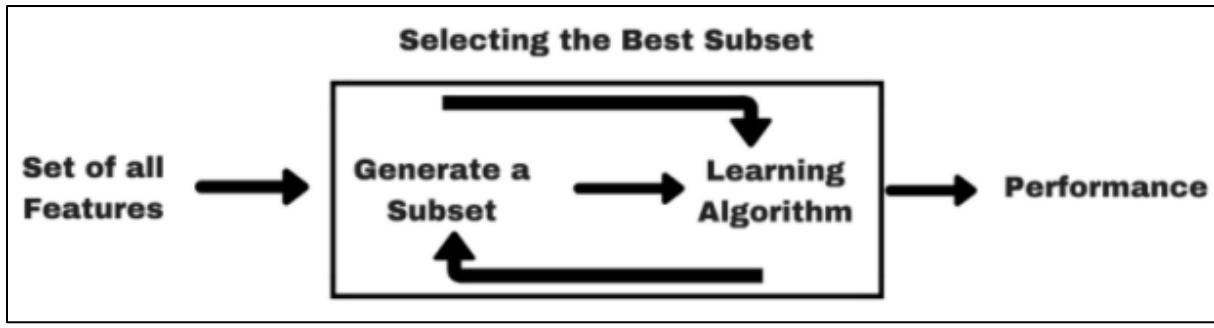
L'information mutuelle (IM) mesure la dépendance d'une variable à une autre en quantifiant la quantité d'informations obtenues sur une caractéristique, à travers l'autre caractéristique. MI est symétrique et non négatif, et vaut zéro si et seulement si les caractéristiques d'entrée et de sortie sont indépendantes. Contrairement à la valeur F ANOVA, les informations mutuelles peuvent capturer des relations non linéaires entre les caractéristiques d'entrée et de sortie.

2.2.2 Approches enveloppantes

Dans les méthodes wrapper, nous essayons d'utiliser un sous-ensemble de fonctionnalités et de former un modèle en les utilisant. Sur la base des inférences que nous tirons du modèle précédent, nous décidons d'ajouter ou de supprimer des fonctionnalités du sous-ensemble. Le problème est essentiellement réduit à un problème de recherche. Ces méthodes sont généralement très coûteuses en temps de calcul.

Quelques exemples courants de méthodes wrapper sont

- Sélection vers l'avant,
- Élimination à rebours,
- Sélection exhaustive des fonctionnalités,
- Élimination des fonctionnalités récursives.
- Élimination récursive des fonctionnalités avec validation croisée



2.2.2.1 Sequential forward selection (SFS)

SFS trouve le meilleur sous-ensemble de fonctionnalités en ajoutant une fonctionnalité qui améliore le mieux le modèle à chaque itération.

Algorithme :

Sequential Forward Selection (SFS)

Input : $Y = \{y_1, y_2, \dots, y_d\}$

- L'algorithme SFS prend les d -caractéristique définie comme input.

Output : $X_k = \{x_j \mid j = 1, 2, \dots, k; x_j \in Y\}$, où $k = (0, 1, 2, \dots, d)$

- SFS retourne un subset de features ; le nombre de features sélectionné k , où $k < d$, doit être spécifié a priori.

Initialisation : $X_0 = \emptyset, k = 0$

- On initialise l'algorithme avec un set vide \emptyset ("null set") donc $k = 0$ (ou k est le nombre de subset).

Step 1 (Inclusion) :

$$x^+ = \arg \max J(X_k + x), \text{ where } x \in Y - X_k$$

$$X_{k+1} = X_k + x^+$$

$$k = k + 1$$

Go to Step 1

- Dans cette étape on ajoute un feature, x^+ , à notre subset X_k .

- x^+ est le feature qui maximise notre fonction de critère, c'est-à-dire le feature qui est associée à la meilleure performance du modèle s'il est ajouté à X_k .

- On répète jusqu'à terminaison.

terminaison : $k = p$

Les p features désirées spécifiées a priori.

2.2.2.2 Sequential backward selection (SBS)

SBS est l'opposé de SFS. SBS commence par toutes les fonctionnalités et supprime la fonctionnalité qui a le moins d'importance pour le modèle à chaque itération.

Algorithme :

Sequential BackWard Selection (SBS)

Input : $Y = \{y_1, y_2, \dots, y_d\}$

- L'algorithme SFS prend les d -caractéristique définie comme input.

Output : $X_k = \{x_j \mid j = 1, 2, \dots, k; x_j \in Y\}$, où $k = (0, 1, 2, \dots, d)$

- SBS retourne un subset de features ; le nombre de features sélectionné k , où $k < d$, doit être spécifié a priori.

Initialisation : $X_0 = Y, k = d$

- On initialise l'algorithme avec tout l'ensemble de features c'est à dire $k = d$.

Step 1 (Exclusion) :

$x^- = \arg \max J(X_k - x)$, where $x \in X_k$

$X_{k-1} = X_k - x^-$

$k = k - 1$

Go to Step 1

- Dans cette étape on élimine un feature, x^- du subset X_k .

- x^- est le feature qui maximise notre fonction de critère , c'est-à-dire la feature qui est associée avec les meilleures performances du modèle s'il est supprimé de X_k .

- Nous répétons cette procédure jusqu'à ce que le critère de terminaison soit satisfait.

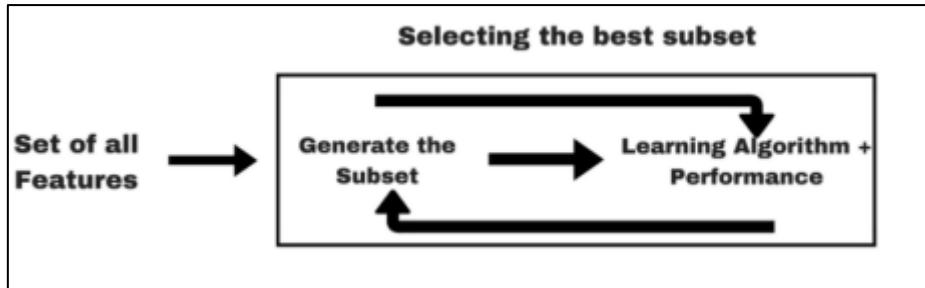
Termination : $k = p$

- On élimine des features du subset X_k jusqu'à ce que le subset est de taille k

2.2.3 Approches intégrées

Les méthodes intégrées combinent les points forts des méthodes de filtrage et d'encapsulation en tirant parti des algorithmes de la machine qui ont leur propre processus de sélection de fonctionnalités intégré. Ils intègrent une étape de sélection des fonctionnalités dans le cadre du processus de formation (c'est-à-dire que la sélection des fonctionnalités et le processus de formation sont effectués simultanément). Les méthodes intégrées ont généralement un processus plus efficace que les méthodes wrapper car elles éliminent le besoin de recycler chaque sous-ensemble de fonctionnalités examinées.

Les méthodes intégrées peuvent être expliquées à l'aide du graphique suivant :



- *Random forest*

Les forêts aléatoires sont l'un des algorithmes d'apprentissage automatique les plus populaires. Ils ont autant de succès car ils offrent en général de bonnes performances prédictives, un faible surajustement et une interprétabilité facile.

Cette interprétabilité est donnée par le fait qu'il est simple de déduire l'importance de chaque variable sur la décision de l'arbre. En d'autres termes, il est facile de calculer dans quelle mesure chaque variable contribue à la décision.

La sélection de fonctionnalités à l'aide de la forêt aléatoire relève de la catégorie des méthodes intégrées. Les méthodes embarquées combinent les qualités des méthodes de filtrage et d'encapsulation. Ils sont implémentés par des algorithmes qui ont leurs propres méthodes de sélection de fonctionnalités intégrées.

Certains des avantages des méthodes embarquées sont :

- Ils sont très précis.
- Ils généralisent mieux.
- Ils sont interprétables.

2.3 Sélection des Données pour l'Évaluation de la Méthode de RdD :

2.3.1 Adéquation des Données Choisies :

Les deux bases de données sélectionnées, l'Iris dataset intégré dans Python et le Stock Dataset (concaténation des datasets Apple, Google, Tesla, Amazon), ont été choisies pour leur diversité et leur pertinence dans des domaines distincts de l'application de la feature selection.

Iris Dataset :

- Raison du Choix : L'Iris dataset est un ensemble de données classique utilisé fréquemment dans des études exploratoires en raison de sa simplicité et de sa diversité de classes (setosa, versicolor, virginica).
- Pertinence : Il permet d'évaluer la capacité de la feature selection à extraire des caractéristiques distinctives dans un contexte de classification de fleurs.

Stock Dataset :

- Raison du Choix : Le Stock Dataset est plus complexe et diversifié, avec des données financières sur plusieurs grandes entreprises.
- Pertinence : L'inclusion de multiples colonnes telles que le prix, l'ouverture, la clôture, le volume, etc., offre un terrain d'essai idéal pour tester la capacité de la feature selection à extraire des caractéristiques significatives dans un contexte financier.

2.3.2 Nature des Données :

Iris Dataset :

- Nature : Les données de l'Iris dataset sont réelles, provenant d'observations botaniques.
- Objectif : Évaluer la capacité de la feature selection à discerner les caractéristiques distinctives des différentes espèces d'Iris.

Stock Dataset :

- Nature : Les données financières du Stock Dataset sont également réelles, provenant des marchés boursiers.
- Objectif : Tester la performance de la feature selection dans l'identification des caractéristiques influentes pour prédire les variations de prix des actions.

2.3.3 Quantité de Données :

Iris Dataset :

- Quantité : L'Iris dataset contient un nombre modéré d'observations, ce qui permet de tester la robustesse de la feature selection sans surcharger l'analyse.
- Équilibre : La balance entre la quantité de données et la complexité du problème reste adéquate.

Stock Dataset :

- Quantité : Le Stock Dataset, en rassemblant des données de plusieurs entreprises, offre une ampleur de données plus importante.
- Diversité : La diversité des entreprises incluses permet d'évaluer la capacité de la feature selection à généraliser ses résultats sur des données variées.

La sélection de ces deux ensembles de données, bien que divergente dans leur nature et leur complexité, permettra une évaluation approfondie de la performance de la feature selection dans des contextes diversifiés. L'Iris dataset offre un contexte plus simple pour tester les capacités de classification, tandis que le Stock Dataset apporte une dimension plus complexe et réaliste, typique des données financières. Ces choix visent à fournir des résultats robustes et à assurer la pertinence de l'évaluation de la méthode de R&D par feature selection.

3 - Réalisation et Manipulation

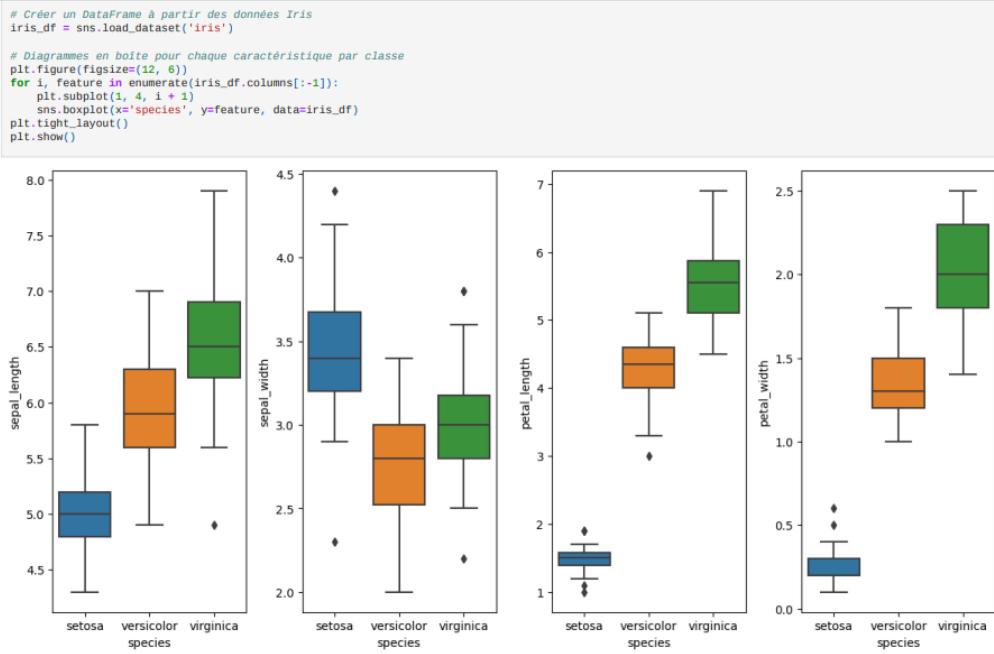
3.1 Etape 1: Exploration des deux Datasets (Visualisation, preprocessing, cleaning)

3.1.1 Iris Dataset

La figure suivante présente le code Python utilisant la bibliothèque scikit-learn pour charger et afficher des détails sur le jeu de données Iris comme demandé. Le code importe le jeu de données, stocke les données et les étiquettes dans des variables distinctes, puis imprime les noms des caractéristiques, les noms des cibles, ainsi que le nombre d'échantillons et de caractéristiques. Les résultats imprimés, visibles sous le code, comprennent les noms des caractéristiques, les noms des cibles (espèces), le nombre d'échantillons, et le nombre de caractéristiques, illustrant trois espèces d'Iris avec quatre caractéristiques mesurées en centimètres.

```
In [4]: from sklearn.datasets import load_iris
# Load the Iris dataset
iris = load_iris()
# The dataset is stored in the 'data' attribute as a NumPy array
X = iris.data
# The target values (species labels) are stored in the 'target' attribute
y = iris.target
# You can also access feature names and target names
feature_names = iris.feature_names
target_names = iris.target_names
# Print some information about the dataset
print("Feature names:", feature_names)
print("Target names:", target_names)
print("Number of samples:", X.shape[0])
print("Number of features:", X.shape[1])
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']
Number of samples: 150
Number of features: 4
```

3.1.1.1 Visualisation



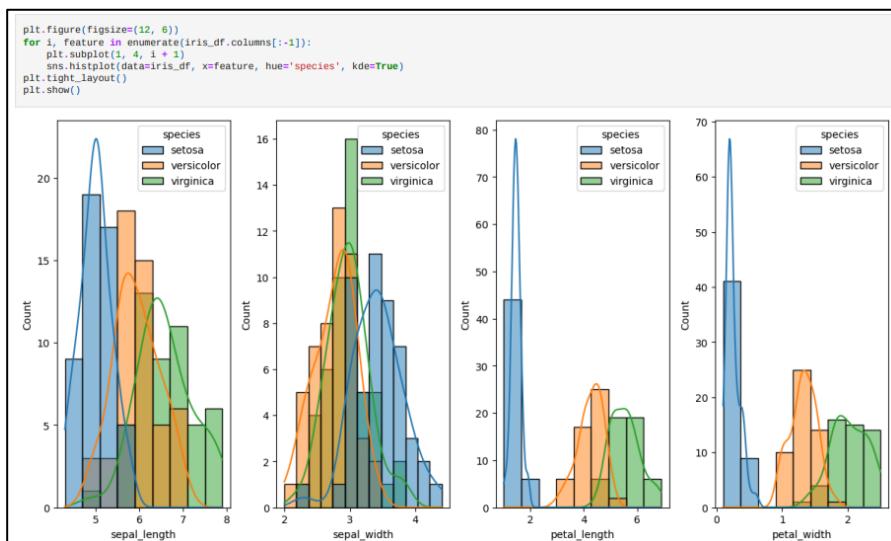
Le graphique dans la figure ci-dessus présente des boîtes à moustaches comparant les caractéristiques morphologiques de trois espèces d'Iris : setosa, versicolor et virginica. Les

mesures incluent la longueur et la largeur des sépales et des pétales, chacune représentée par une couleur différente : bleu pour setosa, orange pour versicolor et vert pour virginica. Les boîtes à moustaches illustrent la distribution des mesures pour chaque espèce, montrant que les iris setosa ont généralement des pétales plus courts mais des sépales plus larges. De plus, l'espèce virginica tend à avoir les pétales les plus longs et larges. Les points noirs signalent des valeurs aberrantes, et ce graphique offre une visualisation claire des différences morphologiques entre les espèces d'iris.

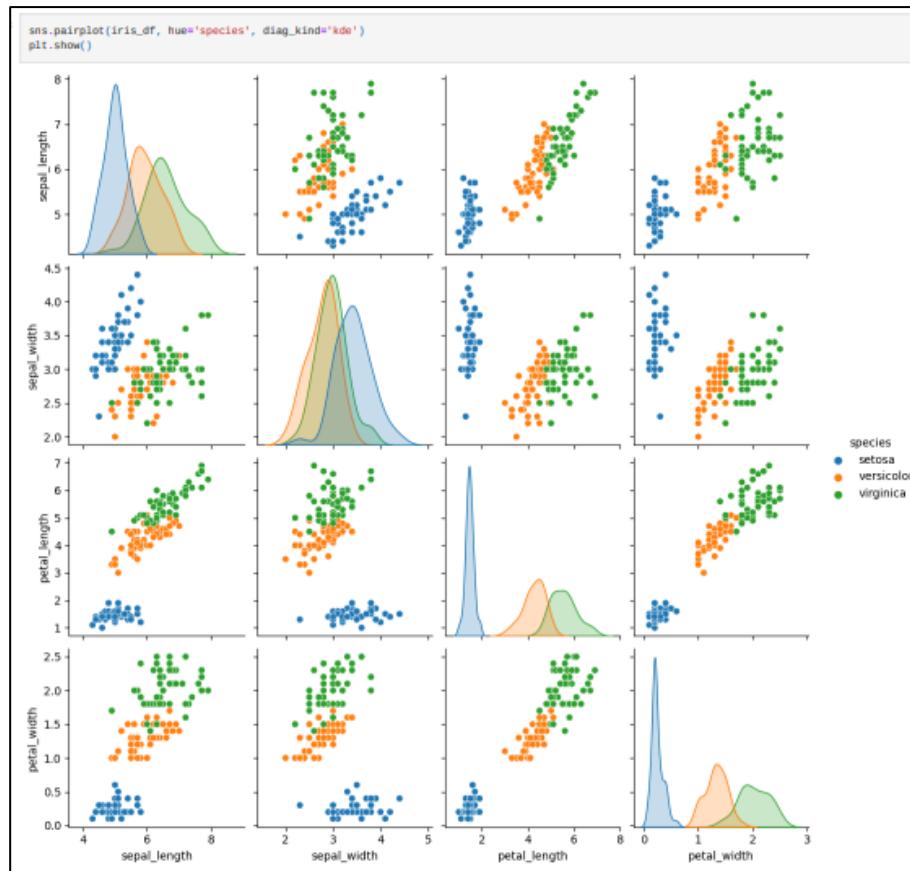
a) Histogrammes pour chaque caractéristique par classe

La figure suivante présente quatre graphiques distincts représentant la distribution des caractéristiques des fleurs (longueur et largeur des sépales et pétales) pour trois espèces différentes : setosa, versicolor et virginica. Chaque graphique est coloré différemment (bleu pour setosa, orange pour versicolor et vert pour virginica). Les axes des graphiques montrent les valeurs mesurées pour chaque caractéristique, avec des plages spécifiques.

Le premier graphique à gauche représente la distribution de la longueur du sépale, le deuxième graphique représente la distribution de la largeur du sépale, le troisième graphique représente la distribution de la longueur du pétale et le quatrième graphique à droite représente la distribution de la largeur du pétale.

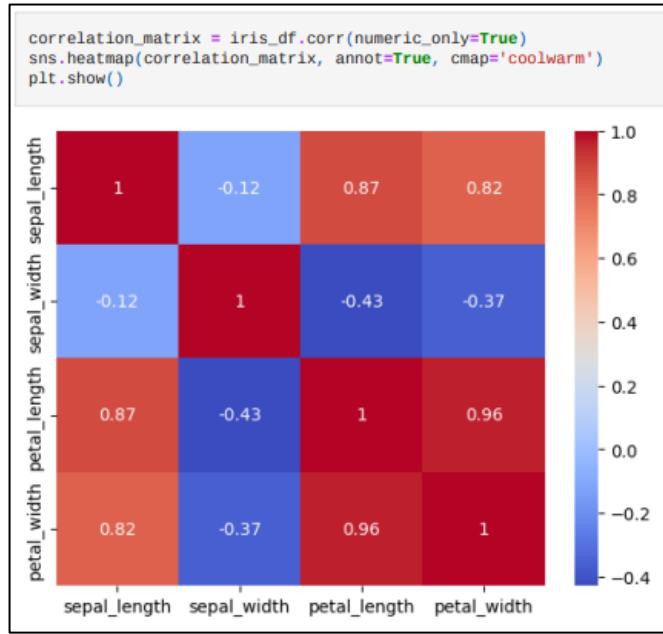


b) Pairplot pour chaque caractéristique par classe



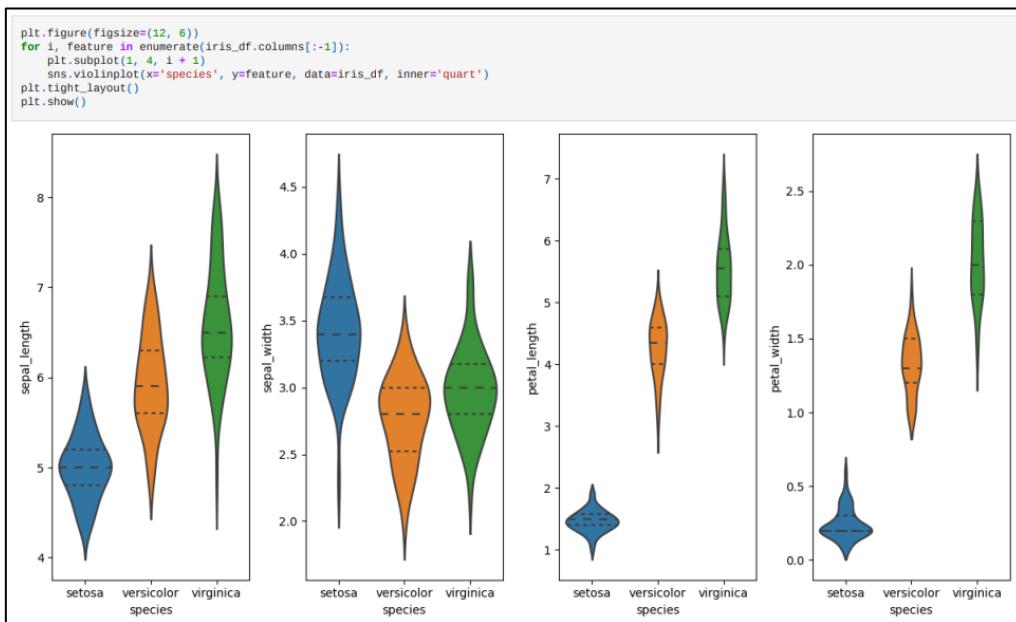
Les graphiques présentent les distributions et relations entre les longueurs et largeurs des sépales et pétales de trois espèces d'iris : setosa, versicolor et virginica. Les couleurs distinctes (bleu, orange et vert) représentent chaque espèce. Ils révèlent des différences morphologiques notables, notamment des sépales plus larges et des pétales plus courts chez les iris setosa. Les distributions sur la diagonale soulignent clairement la distinction entre les setosa et les deux autres espèces. Ces graphiques sont essentiels pour visualiser les disparités entre les espèces d'iris en termes de caractéristiques morphologiques, offrant une compréhension visuelle des variations significatives.

c) Matrice de corrélation



Cette matrice de corrélation représente les relations entre la longueur et la largeur des sépales et des pétales. Les valeurs proches de 1 indiquent une corrélation positive forte, les valeurs proches de -1 indiquent une corrélation négative forte, et les valeurs autour de 0 indiquent une faible ou aucune corrélation. Les couleurs varient du bleu au rouge ; le bleu indique une corrélation négative, le rouge une corrélation positive et le blanc aucune corrélation. Les nombres annotés dans chaque cellule indiquent la valeur exacte de la corrélation entre chaque paire de variables. À droite se trouve une barre d'échelle colorée qui va de -0.4 (bleu foncé) à 1 (rouge foncé), illustrant l'intensité des corrélations.

d) Diagrammes en violon pour chaque caractéristique par classe



Il s'agit d'un ensemble de graphiques de violon illustrant la distribution des longueurs de sépales, de pétales et de largeurs pour trois espèces différentes d'iris (setosa, versicolor et virginica). Chaque graphique présente la distribution sous forme de densité le long d'un axe y spécifique. Les graphiques sont colorés différemment pour représenter les trois espèces d'iris : setosa (bleu), versicolor (orange) et virginica (vert). Le premier graphique à gauche montre la distribution des longueurs des sépales pour chaque espèce. Le deuxième graphique illustre la distribution des largeurs des sépales. Le troisième présente la distribution des longueurs des pétales. Le quatrième et dernier graphique à droite affiche la distribution des largeurs des pétales. Les axes y varient pour chaque caractéristique mesurée, avec les valeurs numériques indiquées le long du côté gauche du graphe.

e) Diagrammes en nuage de points en 2D

```

colors = ['red', 'orange', 'blue']
fig, ax = plt.subplots(2, 3, figsize=(18, 9))
fig.tight_layout(pad=5)
ax = ax.ravel()

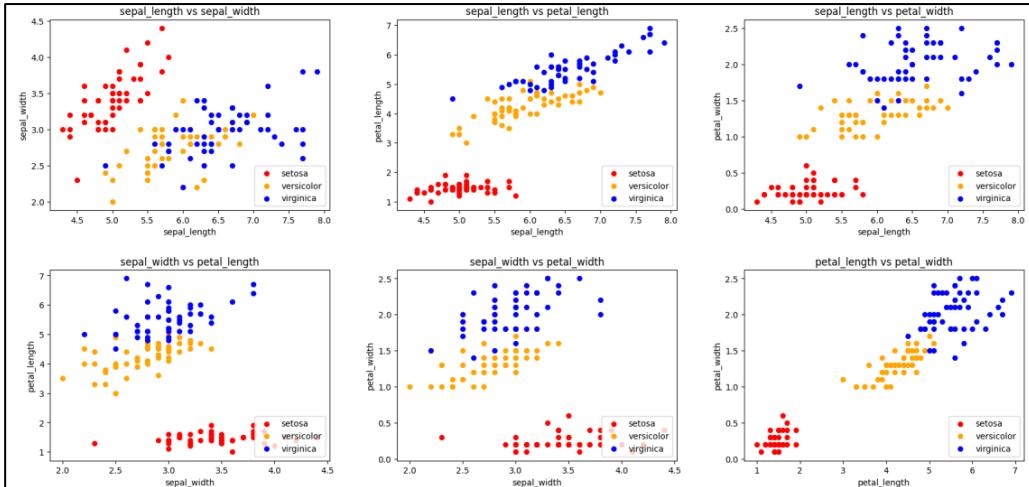
x=0
for i in range(3):
    for j in range(4):
        if (j>i+1):
            for y in range(0, 3):
                tmp_df = iris_df[iris_df['species'] == iris.target_names[y]]
                ax[x].scatter(tmp_df.iloc[:,i], tmp_df.iloc[:,j], color=colors[y], s=30, label=iris.target_names[y])

        ax[x].set_xlabel(iris_df.columns[i])
        ax[x].set_ylabel(iris_df.columns[j])
        ax[x].set_title(f"{iris_df.columns[i]} vs {iris_df.columns[j]}")
        ax[x].legend(loc='lower right')

        x=x+1

plt.show()

```



La figure ci-dessus montre six graphiques de dispersion qui comparent les différentes caractéristiques des fleurs d'Iris de trois espèces différentes : setosa, versicolor et virginica. Les points sur chaque graphique sont colorés différemment pour représenter chaque espèce.

- Premier graphique (en haut à gauche) : Il compare la longueur et la largeur des sépales. Les points sont bien séparés, indiquant une distinction claire entre les trois espèces basées sur ces caractéristiques.

- Deuxième graphique (en haut au milieu) : Il met en relation la longueur des sépales avec la longueur des pétales. On observe une certaine superposition entre versicolor et virginica.
- Troisième graphique (en haut à droite) : Il compare la longueur des sépales avec la largeur des pétales. Ici aussi, il y a une superposition entre versicolor et virginica, mais setosa est distincte.
- Quatrième graphique (en bas à gauche) : Ce plot montre le rapport entre la largeur des sépales et la longueur des pétales. Versicolor et virginica se chevauchent légèrement.
- Cinquième graphique (en bas au milieu) : Il compare la largeur des sépales avec celle des pétales. Une fois de plus, on observe un chevauchement entre versicolor et virginica.
- Sixième graphique (en bas à droite) : Ce dernier plot met en relation la longueur et la largeur du pétale, montrant une distinction nette entre toutes les espèces.

f) Diagrammes en nuage de points en 3D

```

import itertools
from mpl_toolkits.mplot3d import Axes3D

# Définir une correspondance de couleurs pour chaque espèce
colors = {'setosa': 'red', 'versicolor': 'green', 'virginica': 'blue'}

# Créer une colonne 'color' basée sur l'espèce pour attribuer une couleur à chaque point
iris_df['color'] = iris_df['species'].map(colors)

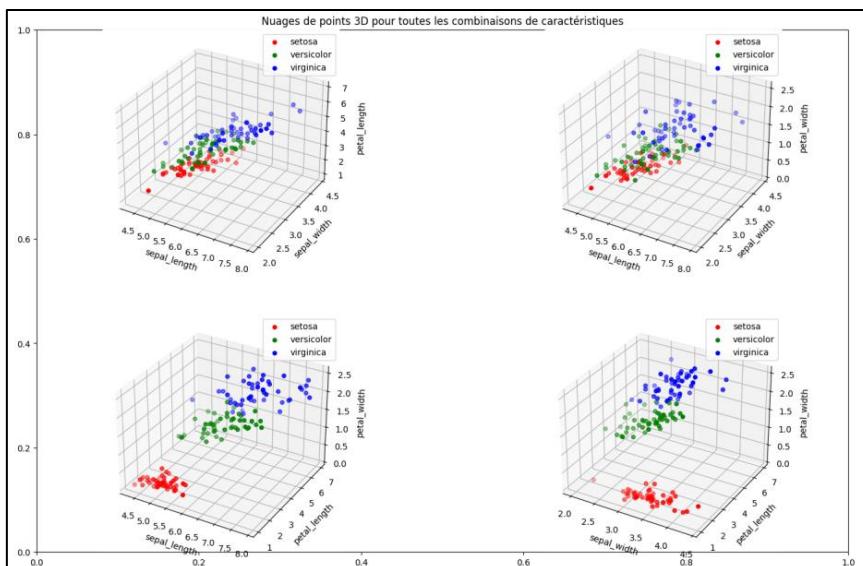
# Obtenir toutes les combinaisons possibles de trois caractéristiques parmi les quatre
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
combinations = list(itertools.combinations(features, 3))

# Créer un nuage de points 3D pour chaque combinaison
fig = plt.figure(figsize=(17, 11))
plt.title('Nuages de points 3D pour toutes les combinaisons de caractéristiques')

# Tracer un graphique pour chaque combinaison de caractéristiques
for index, (feat1, feat2, feat3) in enumerate(combinations, start=1):
    ax = fig.add_subplot(2, 2, index, projection='3d')
    for species, color in colors.items():
        subset = iris_df[iris_df['species'] == species]
        ax.scatter(subset[feat1], subset[feat2], subset[feat3], c=color, label=species)
    ax.set_xlabel(f'{feat1}')
    ax.set_ylabel(f'{feat2}')
    ax.set_zlabel(f'{feat3}')
    ax.legend()

plt.show()

```

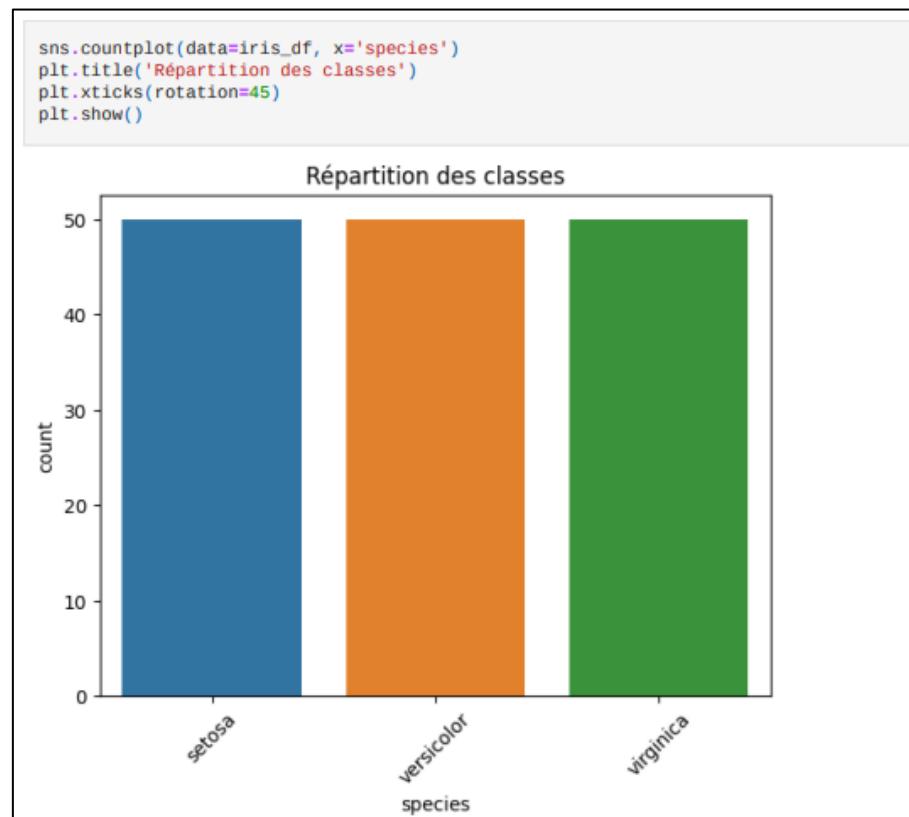


L'image montre quatre nuages de points 3D qui représentent les données pour trois différentes espèces de fleurs (setosa, versicolor et virginica) en fonction de leurs caractéristiques : la longueur du sépale, la largeur du sépale et la largeur du pétale. Chaque point dans le nuage représente une observation individuelle d'une fleur, avec sa couleur indiquant son espèce.

- Premier graphique (en haut à gauche) : Il compare la longueur et la largeur des sépales. Les points sont bien séparés, indiquant une distinction claire entre les trois espèces basées sur ces caractéristiques.
- Deuxième graphique (en bas à droite) : Il met en relation la longueur des sépales avec la longueur des pétales. On observe une certaine superposition entre versicolor et virginica.
- Troisième graphique (en haut à droite) : Il compare la longueur des sépales avec la largeur des pétales. Ici aussi, il y a une superposition entre versicolor, virginica et setosa.
- Quatrième graphique (en bas à gauche) : Ce plot montre le rapport entre la largeur des sépales et la longueur des pétales. Versicolor et virginica se chevauchent légèrement.

g) Diagramme en barres pour la répartition des classes

Le graphique affiche la distribution des trois espèces, chaque espèce ayant un nombre égal de 50.



3.1.1.2 Etude statistique de Iris dataset

L'étude statistique de l'Iris dataset offrira un regard approfondi sur les caractéristiques des fleurs d'Iris, une exploration qui va au-delà de l'observation simple pour révéler des tendances, des corrélations et des distinctions significatives.

a) Analyse Statistique des Caractéristiques

```
# Sélectionner uniquement les colonnes numériques
numeric_columns = iris_df.select_dtypes(include=[np.number])

# Statistiques descriptives pour chaque variable
descriptive_stats = numeric_columns.describe()

# Variance pour chaque variable
variance = numeric_columns.var()

# Écart-type pour chaque variable
std_deviation = numeric_columns.std()

# Médiane pour chaque variable
median = numeric_columns.median()

# Afficher les statistiques descriptives
print("Statistiques descriptives :")
print(descriptive_stats)

print("\n ======")

print("\nVariance :")
print(variance)

print("\n ======")

print("\nÉcart-type :")
print(std_deviation)

print("\n ======")

print("\nMédiane :")
print(median)
```

Le script génère des statistiques détaillées pour quatre caractéristiques des fleurs d'Iris : longueur et largeur des sépales, ainsi que longueur et largeur des pétales. Les résultats incluent des mesures telles que le compte, la moyenne, l'écart-type, les valeurs minimale et maximale, ainsi que les percentiles 25%, 50%, et 75% pour chacune des caractéristiques. Voici un aperçu des résultats :

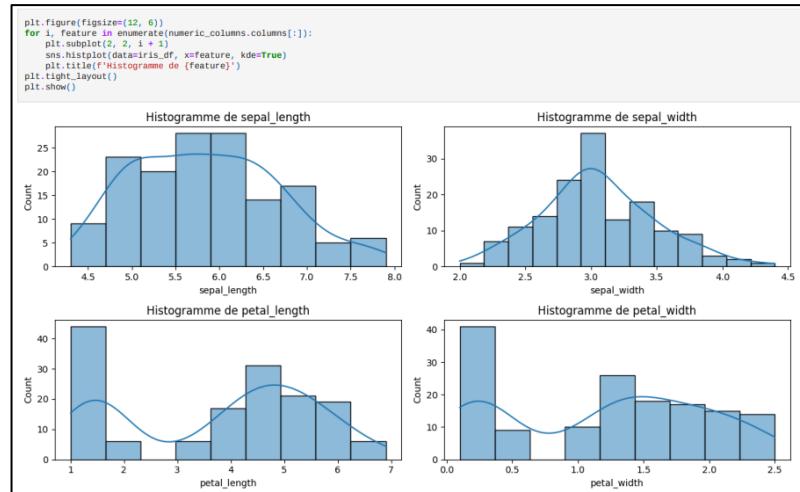
```
Statistiques descriptives :
   sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000   150.000000   150.000000
mean     5.843333   3.057333   3.758000   1.199333
std      0.828066   0.435866   1.765298   0.762238
min      4.300000   2.000000   1.000000   0.100000
25%     5.100000   2.800000   1.600000   0.300000
50%     5.800000   3.000000   4.350000   1.300000
75%     6.400000   3.300000   5.100000   1.800000
max     7.900000   4.400000   6.900000   2.500000
=====

Variance :
sepal_length    0.685694
sepal_width     0.189979
petal_length    3.118278
petal_width     0.581006
dtype: float64
=====

Écart-type :
sepal_length    0.828066
sepal_width     0.435866
petal_length    1.765298
petal_width     0.762238
dtype: float64
=====

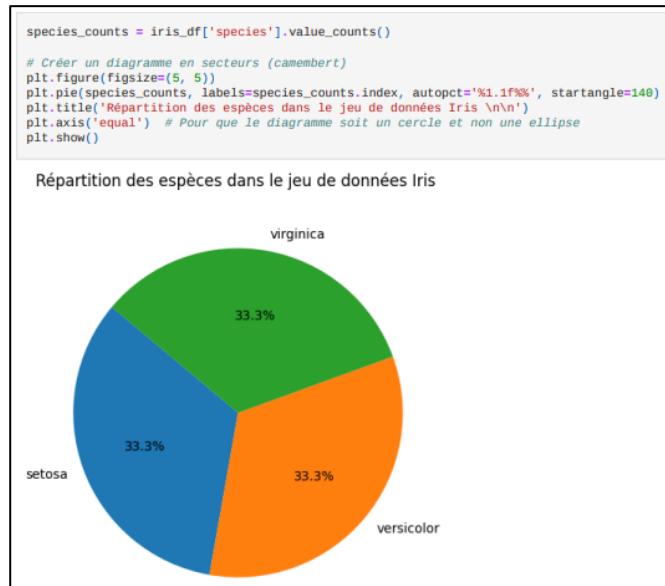
Médiane :
sepal_length    5.80
sepal_width     3.00
petal_length    4.35
petal_width     1.30
dtype: float64
```

b) Création d'un histogramme pour chaque variable



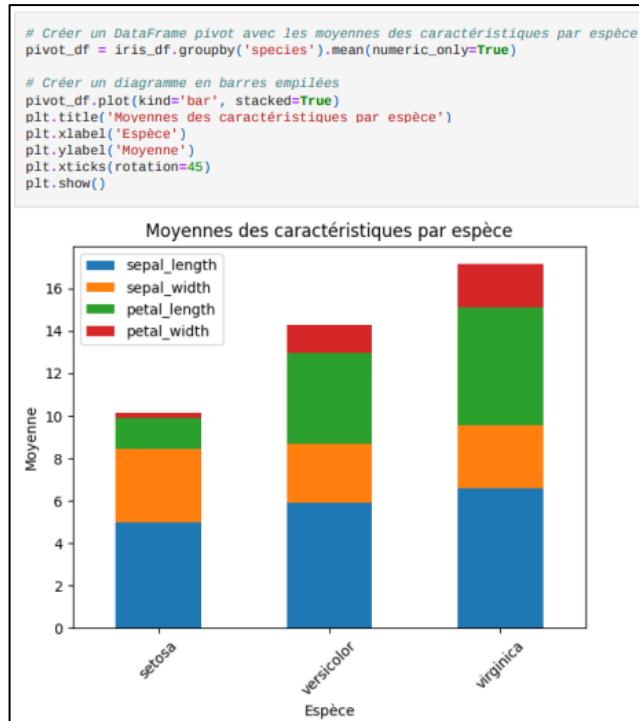
Les graphiques présentent des histogrammes illustrant la distribution des longueurs et des largeurs des sépales et des pétales, générés à partir d'un ensemble de données. Chaque histogramme, codé en bleu, affiche le nombre d'occurrences sur l'axe des y et les valeurs de mesure sur l'axe des x. Chaque graphique est étiqueté "Histogramme de" suivi de la variable qu'il représente (longueur du sépale, largeur du sépale, longueur du pétale, largeur du pétale). Ces visualisations mettent en évidence une distribution variée des comptes pour différentes valeurs de mesure, offrant ainsi un aperçu détaillé de la répartition des caractéristiques morphologiques des sépales et des pétales.

c) Répartition des espèces dans le jeu de données Iris



Le code Python a généré un diagramme circulaire qui montre la répartition des espèces dans le jeu de données Iris. Les trois espèces, setosa, versicolor et virginica, sont également réparties, chacune représentant 33,3% du total.

d) Création d'un diagramme en barres empilées



Le graphique illustre les moyennes des caractéristiques (longueur du sépale, largeur du sépale, longueur du pétales et largeur du pétales) pour trois espèces d'Iris distinctes : setosa, versicolor et virginica. En examinant ce graphique, on constate que l'espèce setosa présente en moyenne la plus grande largeur de sépale, tandis que l'espèce virginica affiche les moyennes les plus élevées pour la longueur du pétales et du sépale.

L'analyse de ce graphique révèle que les caractéristiques "petal length" et "petal width" sont plus importantes pour l'espèce virginica par rapport aux deux autres espèces. De plus, les caractéristiques "sepal length" et "sepal width" sont plus prononcées pour l'espèce virginica par rapport à l'espèce versicolor, bien que ces valeurs soient inférieures à celles de l'espèce setosa. Cette représentation visuelle permet une comparaison claire des moyennes entre les différentes espèces d'Iris en termes de leurs caractéristiques morphologiques.

En résumé, le diagramme montre que la taille des pétales, tant en longueur qu'en largeur, varie considérablement entre les espèces d'Iris et pourrait être déterminante pour les distinguer. La longueur des sépales est assez similaire pour deux espèces, mais beaucoup plus courte pour la troisième, tandis que la largeur des sépales ne diffère pas autant. Globalement, les caractéristiques des pétales sont les plus distinctives parmi les mesures analysées.

e) Création d'un nuage de points avec des couleurs basées sur la largeur et la longueur du pétales

```
# Créer une figure et un ensemble d'axes de sous-plots avec 1 ligne et 2 colonnes
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Créer le premier nuage de points sur le premier axe avec des couleurs basées sur la largeur du pétaire (petal_width)
sns.scatterplot(data=iris_df, x='sepal_length', y='sepal_width', hue='petal_width', palette='viridis', ax=axes[0])
axes[0].set_xlabel('Longueur du sépale')
axes[0].set_ylabel('Largeur du sépale')
axes[0].set_title('Nuage de points avec couleurs\n basées sur la largeur du pétaire')

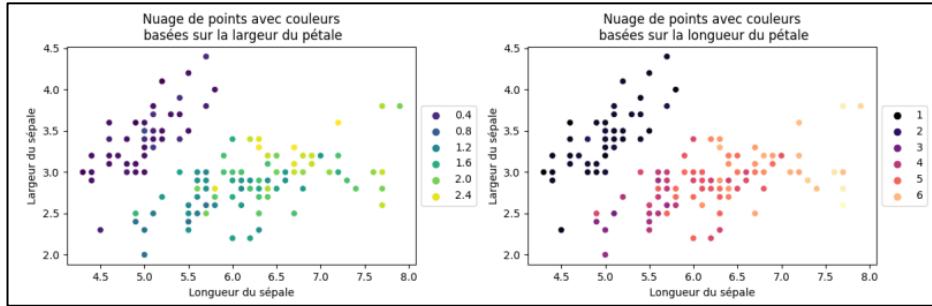
# Créer le second nuage de points sur le deuxième axe avec des couleurs basées sur la longueur du pétaire (petal_length)
sns.scatterplot(data=iris_df, x='sepal_length', y='sepal_width', hue='petal_length', palette='magma', ax=axes[1])
axes[1].set_xlabel('Longueur du sépale')
axes[1].set_ylabel('Largeur du sépale')
axes[1].set_title('Nuage de points avec couleurs\n basées sur la longueur du pétaire')

# Afficher la légende à l'extérieur du graphique sur le premier axe
axes[0].legend(loc='center left', bbox_to_anchor=(1, 0.5))

# Afficher la légende à l'extérieur du graphique sur le deuxième axe
axes[1].legend(loc='center left', bbox_to_anchor=(1, 0.5))

# Ajuster l'espacement entre les sous-plots
plt.tight_layout()

# Afficher la figure complète
plt.show()
```



Les graphiques sont des nuages de points où chaque point représente une fleur, avec la longueur du sépale sur l'axe des x et la largeur du sépale sur l'axe des y. Les couleurs des points sont basées sur la largeur du pétaire dans le premier graphique et sur la longueur du pétaire dans le second graphique. Dans le premier graphique, on peut observer une concentration de points avec une largeur de pétaire plus petite en bas à gauche, et ceux avec une largeur de pétaire plus grande en haut à droite. Dans le second graphique, les fleurs avec une longueur de pétaire plus courte sont concentrées en bas à gauche tandis que celles avec une longueur de pétaire plus longue se trouvent dispersées vers le haut et la droite.

Autrement, les graphiques montrent que la longueur et la largeur des sépales des iris sont liées à la taille des pétales. Les fleurs avec de plus grands sépales tendent à avoir de plus grands pétales, ce qui est indiqué par un changement graduel de couleur dans les points. Ces tendances visuelles peuvent être utiles pour la classification des espèces d'iris.

3.1.2 Stock Dataset

3.1.2.1 Chargement des données

Le résultat du chargement des données contient des informations sur quatre ensembles de données différents, chacun associé à une entreprise technologique différente : Apple, Google, Amazon et Tesla. Chaque ensemble de données contient 11 échantillons et 1028 caractéristiques.

```
Apple=pd.read_csv('Apple_Data.csv')
Google=pd.read_csv('Google_Data.csv')
Amazon=pd.read_csv('Amazon_Data.csv')
Tesla=pd.read_csv('Tesla_Data.csv')

print("Apple Dataset")
print("Number of samples:", Apple.shape[1])
print("Number of features:", Apple.shape[0])
print("=====")
print("Google Dataset")
print("Number of samples:", Google.shape[1])
print("Number of features:", Google.shape[0])
print("=====")
print("Amazon Dataset")
print("Number of samples:", Amazon.shape[1])
print("Number of features:", Amazon.shape[0])
print("=====")
print("Tesla Dataset")
print("Number of samples:", Tesla.shape[1])
print("Number of features:", Tesla.shape[0])

Apple Dataset
Number of samples: 11
Number of features: 1028
=====
Google Dataset
Number of samples: 11
Number of features: 1028
=====
Amazon Dataset
Number of samples: 11
Number of features: 1028
=====
Tesla Dataset
Number of samples: 11
Number of features: 1028
```

3.1.2.2 Concaténation des datasheets Apple, Google, Tesla, Amazon

Dans cette partie, nous allons recourir à la création d'un DataFrame Pandas en combinant des ensembles de données liés aux prix d'actions d'entreprises telles qu'Apple, Google, Tesla et Amazon, comme c'est indiqué dans la figure suivante. Il attribue un nom à chaque entreprise dans une colonne supplémentaire appelée 'company_name'. Les données affichées proviennent des prix d'actions d'Apple, montrant des informations telles que la date, les prix d'ouverture, de clôture, les volumes, les changements en pourcentage, etc. Cela permet une visualisation succincte des premières lignes de ces données agrégées.

```
company_list = [Apple, Google, Tesla, Amazon]
company_name = ["APPLE", "GOOGLE", "TESLA", "AMAZON"]
for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name

df = pd.concat(company_list, axis=0)
df.head(7)
```

	Date	Price	Open	High	Low	Vol.	Change %	coef	state	activity_in	activity_out	company_name
0	Jan 14, 2022	173.07	171.36	173.75	171.1	78.73M	0.51%	497.296390	6.0	544.567390	290.106619	APPLE
1	Jan 13, 2022	172.19	175.78	176.62	171.79	82.27M	-1.90%	889.874180	1.0	958.830431	116.604922	APPLE
2	Jan 12, 2022	175.53	178.12	NaN	174.82	72.77M	0.26%	33.808154	1.0	651.171829	892.616525	APPLE
3	Jan 11, 2022	175.08	172.32	175.18	170.82	75.94M	1.68%	76.642932	2.0	212.135502	539.260305	APPLE
4	Jan 10, 2022	172.19	169.08	172.5	168.17	103.82M	0.01%	802.343154	9.0	139.144297	399.070275	APPLE
5	Jan 07, 2022	172.17	172.09	174.14	171.03	86.71M	0.10%	419.406528	0.0	482.941597	46.037655	APPLE
6	Jan 06, 2022	172.0	172.7	175.3	171.64	95.42M	-1.67%	769.383989	7.0	925.907870	863.828101	APPLE

3.1.2.3 Data preprocessing

a) Changement du format et type des dates

Dans cette partie, nous allons transformer la colonne 'Date' du DataFrame en un format datetime, facilitant ainsi la manipulation et l'analyse temporelle des données. Les cinq premières lignes du DataFrame modifié sont ensuite affichées, montrant les dates désormais au format datetime.

	Date	Price	Open	High	Low	Vol.	Change %	coef	state	activity_in	activity_out	company_name
0	Jan 14, 2022	173.07	171.36	173.75	171.1	78.73M	0.51%	497.296390	6.0	544.567398	290.106619	APPLE
1	Jan 13, 2022	172.19	175.78	176.62	171.79	82.27M	-1.90%	889.874180	1.0	958.830431	116.604922	APPLE
2	Jan 12, 2022	175.53	176.12	NaN	174.82	72.77M	0.26%	33.808154	1.0	651.171829	892.616525	APPLE
3	Jan 11, 2022	175.08	172.32	175.18	170.82	75.94M	1.68%	76.642932	2.0	212.135502	539.260305	APPLE
4	Jan 10, 2022	172.19	169.08	172.5	168.17	103.82M	0.01%	802.343154	9.0	139.144297	399.070275	APPLE
5	Jan 07, 2022	172.17	172.89	174.14	171.03	86.71M	0.10%	419.406528	0.0	482.941597	46.037655	APPLE
6	Jan 06, 2022	172.0	172.7	175.3	171.64	95.42M	-1.67%	769.383989	7.0	925.907870	863.828101	APPLE

b) Correction du problème de la virgule dans la colonne 'Price', 'Open', 'High' et 'Low'

On va nettoyer les colonnes numériques ('Price', 'Open', 'High', 'Low') d'un DataFrame en supprimant les virgules et en convertissant les valeurs en format numérique (float).

```
df['Price']
0      173.07
1      172.19
2      175.53
3      175.08
4      172.19
...
1023    1,174.76
1024    1,177.62
1025    1,187.38
1026    1,190.58
1027    1,179.14
Name: Price, Length: 4112, dtype: object

# Suppression des virgules et conversion du type à float
df['Price'] = pd.to_numeric(df['Price'].astype(str).str.replace(',', ''), errors='coerce')
df['Price']

0      173.07
1      172.19
2      175.53
3      175.08
4      172.19
...
1023    1174.76
1024    1177.62
1025    1187.38
1026    1190.58
1027    1179.14
Name: Price, Length: 4112, dtype: float64

# De la même manière pour les autres colonnes
df['Open'] = pd.to_numeric(df['Open'].astype(str).str.replace(',', ''), errors='coerce')
df['High'] = pd.to_numeric(df['High'].astype(str).str.replace(',', ''), errors='coerce')
df['Low'] = pd.to_numeric(df['Low'].astype(str).str.replace(',', ''), errors='coerce')
```

c) Suppression de 'M' du colonnes 'Vol' et la conversion du type

On a visé à nettoyer la colonne 'Vol.' d'un DataFrame. Il supprime les caractères 'M' et 'K' des valeurs, puis convertit les valeurs en format numérique (float). La fonction convert_volume_to_float est appliquée à la colonne 'Vol.' pour effectuer cette transformation. Après l'application du code, la colonne 'Vol.' est mise à jour avec des valeurs numériques représentant le volume des actions échangées.

```

df['Vol.']

0    78.73M
1    82.27M
2    72.77M
3    75.94M
4    103.82M
...
1023   2.12M
1024   2.37M
1025   2.59M
1026   2.95M
1027   4.78M
Name: Vol., Length: 4112, dtype: object

def convert_volume_to_float(volume):
    volume_str = str(volume)
    # Suppression de la 'M' et multiplier la valeur par 1 million, en suite la conversion du type en float
    if 'M' in volume_str:
        return float(volume_str.replace('M', '')) * 1e6
    # Suppression de la 'K' et multiplier la valeur par 1 mille, en suite la conversion du type en float
    elif 'K' in volume_str:
        return float(volume_str.replace('K', '')) * 1e3
    else:
        return float(volume_str)

# Application de la fonction sur la colonne 'Vol.'
df['Vol.'] = df['Vol.'].apply(convert_volume_to_float)
df.head(5)

      Date  Price  Open  High  Low     Vol.  Change %      coef  state activity_in activity_out company_name
0  2022-01-14  173.07  171.36  173.75  171.10  78730000.0   0.51%  497.296390   6.0  544.567398  290.106619    APPLE
1  2022-01-13  172.19  175.78  176.62  171.79  82270000.0  -1.90%  889.874180   1.0  958.830431  116.604922    APPLE
2  2022-01-12  175.53  176.12    NaN  174.82  72770000.0   0.26%  33.808154   1.0  651.171829  892.616525    APPLE
3  2022-01-11  175.08  172.32  175.18  170.82  75940000.0   1.68%  76.642932   2.0  212.135502  539.260305    APPLE
4  2022-01-10  172.19  169.08  172.50  168.17  103820000.0   0.01%  802.343154   9.0  139.144297  399.070275    APPLE

```

d) Suppression du symbole '%' du colonne 'Change'

Il faut supprimer le symbole '%' de la colonne 'Change %' dans un DataFrame. Il utilise la méthode str.replace('%', '') pour supprimer le '%' de chaque valeur, puis convertit les valeurs en format numérique (float) en multipliant par 100. Après l'application de ce code, la colonne 'Change %' est mise à jour avec des valeurs numériques représentant le pourcentage de changement.

```

df['Change %']

0    0.51%
1   -1.90%
2    0.26%
3    1.68%
4    0.01%
...
1023   -0.24%
1024   -0.82%
1025   -0.27%
1026   0.97%
1027   0.42%
Name: Change %, Length: 4112, dtype: object

df['Change %'] = df['Change %'].str.replace('%', '').astype(float) * 100
df.head(5)

      Date  Price  Open  High  Low     Vol.  Change %      coef  state activity_in activity_out company_name
0  2022-01-14  173.07  171.36  173.75  171.10  78730000.0   51.0  497.296390   6.0  544.567398  290.106619    APPLE
1  2022-01-13  172.19  175.78  176.62  171.79  82270000.0  -190.0  889.874180   1.0  958.830431  116.604922    APPLE
2  2022-01-12  175.53  176.12    NaN  174.82  72770000.0   26.0  33.808154   1.0  651.171829  892.616525    APPLE
3  2022-01-11  175.08  172.32  175.18  170.82  75940000.0   168.0  76.642932   2.0  212.135502  539.260305    APPLE
4  2022-01-10  172.19  169.08  172.50  168.17  103820000.0   1.0  802.343154   9.0  139.144297  399.070275    APPLE

```

e) Missing value

1. Aperçu des valeurs manquantes

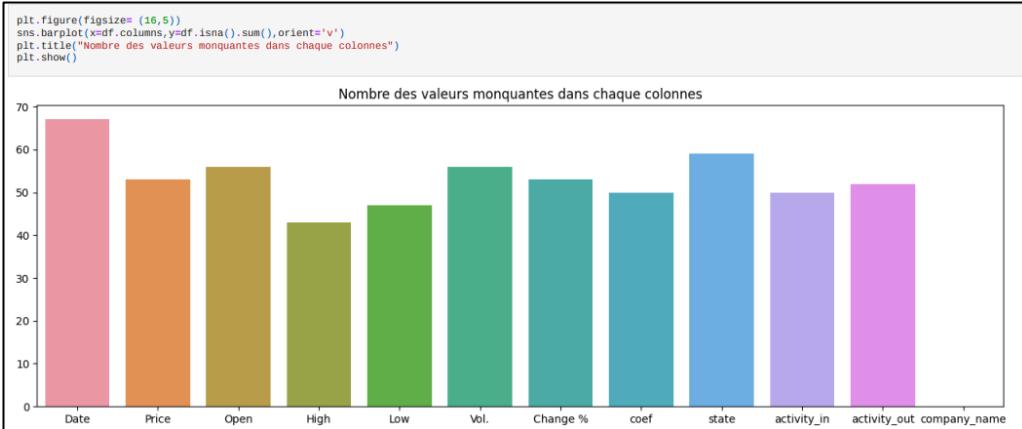
L'aperçu des valeurs manquantes révèle qu'il y a un total de 586 valeurs manquantes dans l'ensemble de données. Ces valeurs manquantes sont réparties dans différentes colonnes comme indiqué dans la figure suivante:

```
print("Nombre total des valeurs manquantes :", df.isna().sum().sum())
Nombre total des valeurs manquantes : 586

print("Nombre de valeurs manquantes dans chaque colonne : ")
print(df.isna().sum())

Nombre de valeurs manquantes dans chaque colonne :
Date           67
Price          53
Open            56
High            43
Low             47
Vol.            56
Change %       53
coef            50
state           59
activity_in     50
activity_out    52
company_name    0
dtype: int64
```

Ce diagramme ci-dessous à barres illustre le nombre de valeurs manquantes dans chaque colonne de l'ensemble de données. Les colonnes sont représentées sur l'axe des x et le nombre de valeurs manquantes sur l'axe des y :



2. Remplacement des valeurs manquantes par la médiane pour les colonnes numériques

```
# Group by 'company_name', then fill missing values in each group
for col in df.select_dtypes(include=[np.number]).columns:
    df[col] = df.groupby('company_name')[col].transform(lambda x: x.fillna(x.median()))

print('Les valeurs manquantes restantes : ', df.isna().sum().sum())

Les valeurs manquantes restantes :  67

df.isna().sum()

Date      67
Price      0
Open       0
High       0
Low        0
Vol.       0
Change %   0
coef       0
state      0
activity_in  0
activity_out  0
company_name  0
dtype: int64
```

Après le remplacement des valeurs manquantes par la médiane pour les colonnes numériques, il reste 67 valeurs manquantes au total dans la colonne 'Date'. Les autres colonnes numériques telles que 'Price', 'Open', 'High', 'Low', 'Vol.', 'Change %', 'coef', 'state', 'activity_in', et 'activity_out' n'ont plus de valeurs manquantes. Cela suggère que le processus de remplacement des valeurs manquantes par la médiane a été efficace pour les colonnes numériques, à l'exception de la colonne 'Date' qui peut nécessiter une approche différente étant donné sa nature non numérique.

3. Remplacement des valeurs manquantes pour la colonne 'Date' avec la date la plus fréquente

Les valeurs manquantes dans la colonne 'Date' seront remplacées par la date la plus fréquente. Pour cela, le code ci-dessous a identifié l'année la plus fréquente, puis le mois le plus fréquent dans cette année, et enfin, le jour le plus fréquent dans ce mois de l'année la plus fréquente.

```
# Filtrer les dates non manquantes
dates_non_null = df['Date'].dropna()

# Trouver l'année la plus fréquente
most_common_year = dates_non_null.dt.year.mode()[0]

# Filtrer les données pour cette année
dates_common_year = dates_non_null[dates_non_null.dt.year == most_common_year]

# Trouver le mois le plus fréquent dans cette année
most_common_month = dates_common_year.dt.month.mode()[0]

# Filtrer les données pour ce mois de l'année la plus fréquente
dates_common_month = dates_common_year[dates_common_year.dt.month == most_common_month]

# Trouver le jour le plus fréquent dans ce mois de cette année
most_common_day = dates_common_month.dt.day.mode()[0]

# Construire la date la plus fréquente à utiliser pour remplacer les valeurs NaN
replacement_date = pd.Timestamp(year=most_common_year, month=most_common_month, day=most_common_day)

# Remplacer les valeurs NaN dans la colonne 'Date'
df['Date'].fillna(replacement_date, inplace=True)

print('Les valeurs manquantes restantes :')
print(df.isna().sum().sum())

Les valeurs manquantes restantes :
0
```

Après le processus de remplacement des valeurs manquantes dans la colonne 'Date' avec la date la plus fréquente, il ne reste plus aucune valeur manquante dans l'ensemble du dataframe. Cette approche a été utilisée pour garantir que la colonne 'Date' reste complète et significative, en évitant la perte d'informations importantes. Maintenant, toutes les colonnes du dataframe sont dépourvues de valeurs manquantes, ce qui renforce la qualité des données pour une analyse statistique ultérieure.

3.1.2.4 Visualisation et étude statique de dataset

a) historical view of the price "APPLE", "GOOGLE", "Tesla", "AMAZON" 2018 -2022

Notre objectif maintenant est de créer une visualisation graphique de l'évolution des prix des actions de quatre entreprises spécifiques ("APPLE", "GOOGLE", "Tesla", "AMAZON") sur la période de 2018 à 2022. La visualisation utilise la bibliothèque Matplotlib en Python.

```
plt.figure(figsize=(18, 26))
plt.subplots_adjust(top=1.5, bottom=1.2)

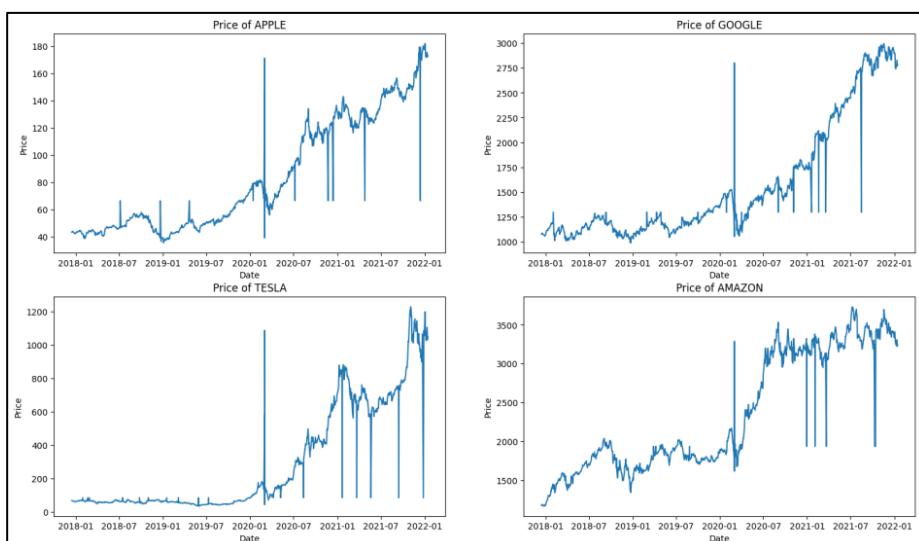
tmp_df = df.copy()
tmp_df['Date'] = pd.to_datetime(tmp_df['Date'])
tmp_df = tmp_df.sort_values(by='Date')

for i, name in enumerate(company_name, start=1):
    plt.subplot(2, 2, i)
    # Selecting the subset of data for each company
    subset = tmp_df[tmp_df['company_name'] == name]
    # Extracting 'Date' and 'Price' for the plot
    date = subset['Date']
    price = subset['Price']

    # Creating a new figure for each company
    plt.plot(date, price)
    plt.ylabel('Price')
    plt.xlabel('Date')
    plt.title(f"Price of {name}")

plt.show()
```

Le code permet de visualiser de manière comparative l'historique des prix des actions des entreprises spécifiées sur la période définie. Cela offre une représentation graphique qui peut aider à identifier des tendances, des modèles ou des divergences dans les performances des entreprises au fil du temps.



L'analyse visuelle des graphiques représentant l'évolution des prix des actions d'APPLE, GOOGLE, TESLA et AMAZON sur la période de 2018 à 2022 révèle des tendances distinctes pour chaque entreprise. Les actions d'APPLE et de GOOGLE affichent une croissance relativement stable au fil du temps, suggérant une performance financière constante. En revanche, TESLA présente une volatilité significative, indiquant des fluctuations importantes dans les prix de ses actions, probablement en raison de la nature dynamique du marché automobile et de l'attention médiatique entourant l'entreprise. Pour AMAZON, le graphique illustre une croissance initiale suivie d'une certaine volatilité, reflétant peut-être les défis et opportunités du secteur du commerce électronique. Ces observations mettent en lumière la diversité des performances des entreprises technologiques et soulignent l'importance de considérer les facteurs spécifiques à chaque secteur pour une analyse approfondie des marchés financiers.

b) Etude statistique de la dataset

On va montrer des statistiques descriptives du dataset. Les statistiques incluent le compte, la moyenne, l'écart type, le minimum, les percentiles 25%, 50% et 75%, et le maximum pour chaque colonne du DataFrame

df.describe()											
	Price	Open	High	Low	Vol.	Change %	coef	state	activity_in	activity_out	
count	4112.000000	4112.000000	4112.000000	4112.000000	4.112000e+03	4112.000000	4112.000000	4112.000000	4112.000000	4112.000000	
mean	1077.819652	1078.937096	1090.409864	1065.060224	4.427593e+07	18.730788	503.448625	4.513132	501.774297	489.732875	
std	1061.135585	1062.706216	1072.989596	1049.913222	5.905174e+07	264.323775	288.417691	2.862421	288.284718	285.065506	
min	35.550000	35.990000	36.430000	35.400000	4.656400e+05	-2106.000000	0.087841	0.000000	0.411954	0.873635	
25%	71.660000	71.270000	72.222500	70.097500	2.560000e+06	-91.000000	253.191127	2.000000	253.439472	244.180774	
50%	1044.800000	1046.300000	1060.240000	1023.580000	1.283000e+07	15.000000	507.209726	4.000000	494.924903	485.673151	
75%	1755.310000	1760.020000	1770.350000	1739.490000	7.566000e+07	128.250000	749.152969	7.000000	747.685945	727.871682	
max	3731.410000	3740.000000	3773.000000	3697.220000	4.268800e+08	1990.000000	999.865616	9.000000	999.915627	999.689931	

L'analyse statistique du dataset fournit des informations clés sur les caractéristiques des variables principales, notamment les prix (Price, Open, High, Low), le volume de transactions (Vol.), le pourcentage de changement (Change %), le coefficient, l'état (state), l'activité entrante (activity_in), et l'activité sortante (activity_out).

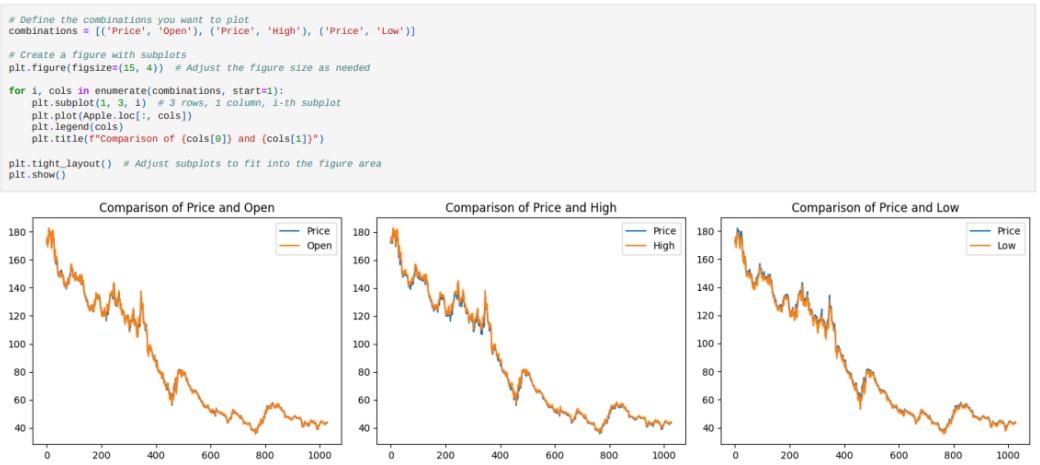
En ce qui concerne les prix, on observe une moyenne générale de 1077.82, 1078.94, 1090.41, et 1065.06 pour les colonnes Price, Open, High, et Low respectivement. Cela suggère une certaine stabilité dans les niveaux de prix au fil du temps, bien que l'écart type élevé indique également une variabilité importante.

En ce qui concerne le pourcentage de changement, le coefficient, l'activité entrante et sortante, les statistiques révèlent la nature dynamique du marché. Par exemple, le pourcentage de changement affiche une moyenne de 18.73%, suggérant des fluctuations considérables dans les prix. Le coefficient, avec une moyenne de 503.45, montre une variation substantielle dans la distribution des données.

Les valeurs minimales et maximales pour chaque variable mettent en évidence des points extrêmes.

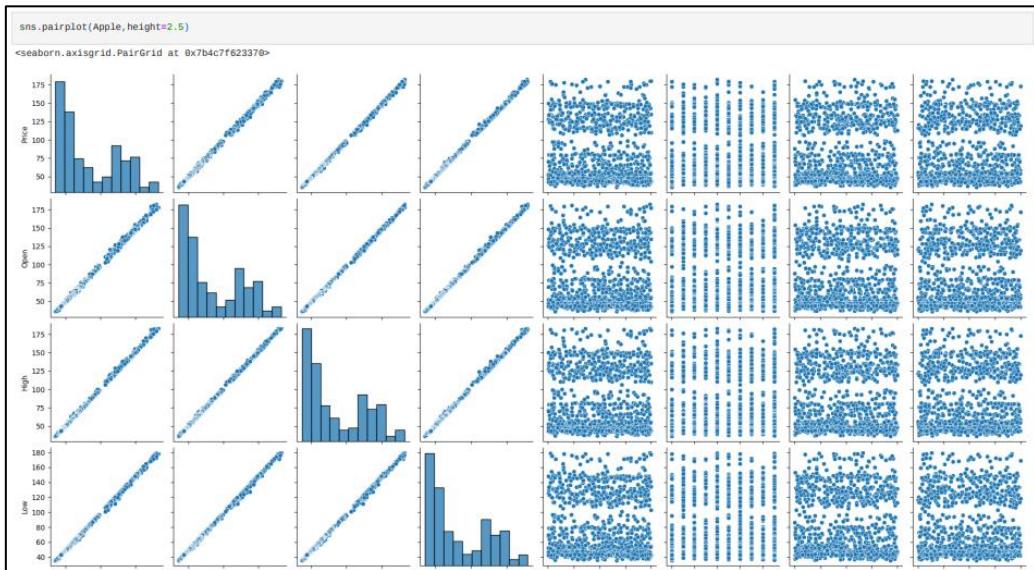
En résumé, ces statistiques offrent un aperçu succinct mais informatif de la dynamique du marché financier, préparant le terrain pour des analyses plus approfondies et ciblées.

c) Variances du prix avec les autre variables (Apple dataset)



Les graphiques illustrent une comparaison entre les prix et d'autres variables telles que Open, High et Low dans le jeu de données d'Apple. Les différentes couleurs des lignes sur les graphiques permettent de distinguer les deux variables comparées sur chaque graphique. Les axes des abscisses (x) sont gradués de 0 à 1000, tandis que les axes des ordonnées (y) varient de 0 à 180. Cette configuration suggère que les prix des actions d'Apple ont été relativement stables au cours de la période analysée.

d) Variances des variables entre eux (Apple dataset)





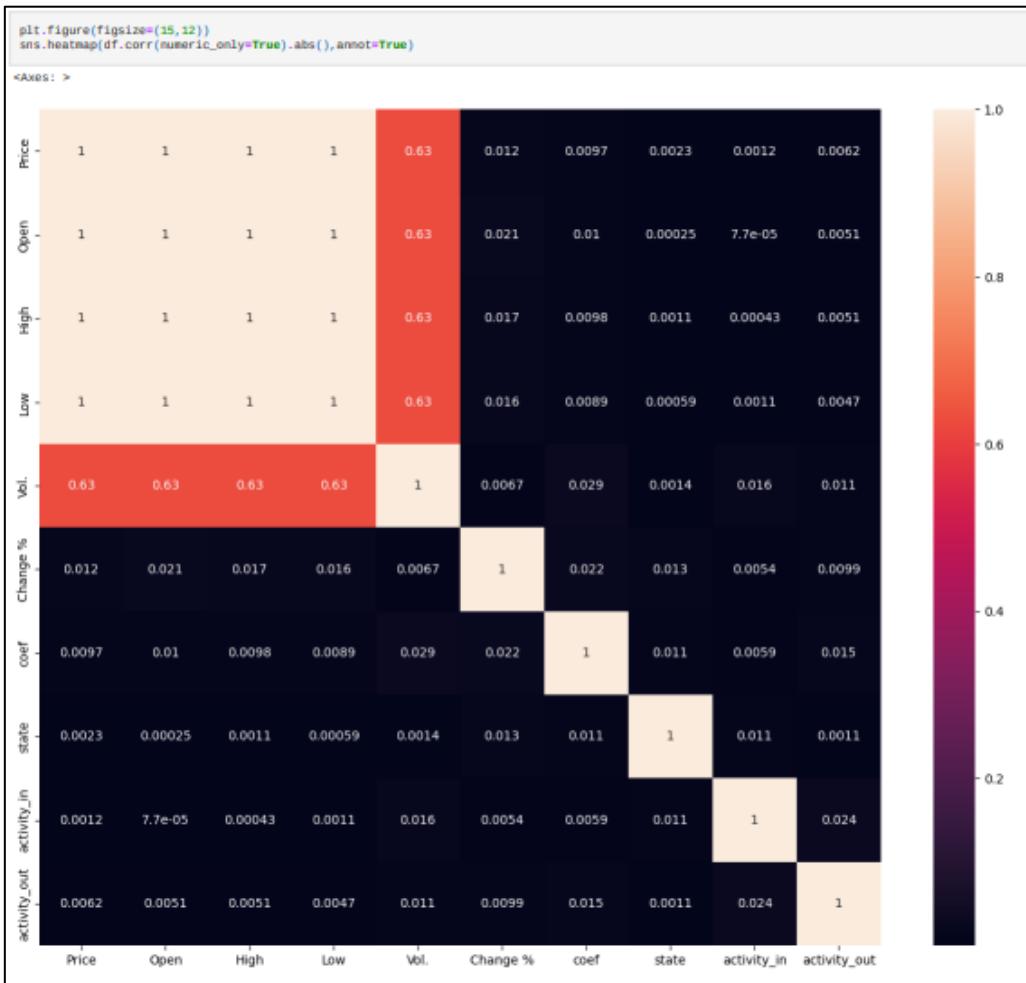
Une matrice de dispersion avec des histogrammes sur la diagonale est effectivement un pairplot, et cela permet de visualiser les relations entre plusieurs variables simultanément. Les histogrammes diagonaux présentent la distribution individuelle de chaque variable, tandis que les graphiques de dispersion hors-diagonale illustrent les relations bivariées entre les paires de variables.

La présence de tendances linéaires dans les graphiques de dispersion peut indiquer des corrélations positives ou négatives entre les variables correspondantes. Ces observations sont cruciales pour comprendre les relations et les patterns dans les données, ce qui peut être utile pour orienter des analyses plus approfondies ou des modélisations.

e) Matrice de corrélation

Maintenant nous allons créer une carte de chaleur. Cette carte de chaleur est utilisée pour visualiser la force et la direction des corrélations entre les variables numériques du DataFrame.

Les cellules plus claires indiquent des corrélations plus fortes, tandis que les cellules plus sombres indiquent des corrélations plus faibles. Les valeurs numériques annotées fournissent les coefficients de corrélation réels entre les paires de variables. Cela peut aider à identifier les relations significatives entre les différentes variables dans le DataFrame.



3.2 Etape2: Feature selection pour les deux Dataset

3.2.1 Iris Dataset

```
x = iris.data
y = iris.target

# Créer un DataFrame Pandas avec les données
iris_df = pd.DataFrame(data=x, columns=iris.feature_names)

# Ajouter la colonne cible "target" au DataFrame
iris_df['target'] = y

iris_df.head(5)

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) target
0 5.1 3.5 1.4 0.2 0
1 4.9 3.0 1.4 0.2 0
2 4.7 3.2 1.3 0.2 0
3 4.6 3.1 1.5 0.2 0
4 5.0 3.6 1.4 0.2 0
```

3.2.1.1 La normalisation des données Iris

Avant d'entamer la feature selection pour l'Iris dataset, une étape cruciale est la normalisation des données. Le but, illustré ici, est de transformer les données pour les resserrer dans une échelle spécifique, généralement entre 0 et 1, en utilisant la méthode "Min-Max Scaling". Cela réajuste les données en fonction de leur valeur minimale et maximale. Pour ce faire, le "MinMaxScaler" de la bibliothèque "sklearn.preprocessing" est appliqué, éliminant les redondances et assurant que chaque caractéristique soit comparable de manière équitable. Cette normalisation préalable renforce l'efficacité de la feature selection en éliminant toute disparité d'échelle entre les différentes variables.

```
from sklearn.preprocessing import MinMaxScaler

# Normaliser les données avec Min-Max Scaling
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)

# Créer un DataFrame Pandas avec des noms de colonnes
normalized_df = pd.DataFrame(X_normalized, columns=iris.feature_names)

# Afficher les premières lignes du DataFrame normalisé
normalized_df.head(5)

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)
0 0.222222 0.625000 0.067797 0.041667
1 0.166667 0.416667 0.067797 0.041667
2 0.111111 0.500000 0.050847 0.041667
3 0.083333 0.458333 0.084746 0.041667
4 0.194444 0.666667 0.067797 0.041667
```

3.2.1.2 Les Résultats de La Classification Avant Feature selection avec (KNN, SVM, Logistic Regression, Gradient Boosting, Decision Tree)

```

from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Diviser les données en ensemble d'entraînement et ensemble de test
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.3, random_state=42)

# Classification avec K plus proches voisins (KNN)
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
knn_confusion_matrix = confusion_matrix(y_test, knn_predictions)

# Classification avec arbre de décision
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
dt_confusion_matrix = confusion_matrix(y_test, dt_predictions)

```

```

# Classification avec régression logistique
lr_classifier = LogisticRegression(random_state=42, max_iter=200)
lr_classifier.fit(X_train, y_train)
lr_predictions = lr_classifier.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)
lr_confusion_matrix = confusion_matrix(y_test, lr_predictions)

# Classification avec Support Vector Machine (SVM)
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train, y_train)
svm_predictions = svm_classifier.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
svm_confusion_matrix = confusion_matrix(y_test, svm_predictions)

# Classification avec Random Forest
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_confusion_matrix = confusion_matrix(y_test, rf_predictions)

```

```

# Classification avec Gradient Boosting
gb_classifier = GradientBoostingClassifier(random_state=42)
gb_classifier.fit(X_train, y_train)
gb_predictions = gb_classifier.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_predictions)
gb_confusion_matrix = confusion_matrix(y_test, gb_predictions)

# Afficher les performances de chaque algorithme
print("Accuracy (KNN): {knn_accuracy}")
print("Accuracy (Decision Tree): {dt_accuracy}")
print("Accuracy (Logistic Regression): {lr_accuracy}")
print("Accuracy (SVM): {svm_accuracy}")
print("Accuracy (Random Forest): {rf_accuracy}")
print("Accuracy (Gradient Boosting): {gb_accuracy}")

print("\n=====")
# Afficher les rapports de classification pour chaque algorithme
print("\nClassification Report (KNN):\n", classification_report(y_test, knn_predictions))
print("=====")
print("\nClassification Report (Decision Tree):\n", classification_report(y_test, dt_predictions))
print("=====")
print("\nClassification Report (Logistic Regression):\n", classification_report(y_test, lr_predictions))
print("=====")
print("\nClassification Report (SVM):\n", classification_report(y_test, svm_predictions))
print("=====")
print("\nClassification Report (Random Forest):\n", classification_report(y_test, rf_predictions))
print("=====")
print("\nClassification Report (Gradient Boosting):\n", classification_report(y_test, gb_predictions))

```

```

Accuracy (KNN): 1.0
Accuracy (Decision Tree): 1.0
Accuracy (Logistic Regression): 0.911111111111111
Accuracy (SVM): 1.0
Accuracy (Random Forest): 1.0
Accuracy (Gradient Boosting): 1.0
=====
Classification Report (KNN):
precision    recall   f1-score   support
0           1.00     1.00      1.00      19
1           1.00     1.00      1.00      13
2           1.00     1.00      1.00      13

accuracy          1.00      45
macro avg       1.00      1.00      45
weighted avg    1.00      1.00      45
=====
Classification Report (Decision Tree):
precision    recall   f1-score   support
0           1.00     1.00      1.00      19
1           1.00     1.00      1.00      13
2           1.00     1.00      1.00      13

accuracy          1.00      45
macro avg       1.00      1.00      45
weighted avg    1.00      1.00      45
=====
Classification Report (Logistic Regression):
precision    recall   f1-score   support
0           1.00     1.00      1.00      19
1           1.00     0.69      0.82      13
2           0.76     1.00      0.87      13

accuracy          0.91      45
macro avg       0.92     0.90      0.89      45
weighted avg    0.93     0.91      0.91      45

```

```

Classification Report (SVM):
precision    recall   f1-score   support
0           1.00     1.00      1.00      19
1           1.00     1.00      1.00      13
2           1.00     1.00      1.00      13

accuracy          1.00      45
macro avg       1.00      1.00      45
weighted avg    1.00      1.00      45
=====
Classification Report (Random Forest):
precision    recall   f1-score   support
0           1.00     1.00      1.00      19
1           1.00     1.00      1.00      13
2           1.00     1.00      1.00      13

accuracy          1.00      45
macro avg       1.00      1.00      45
weighted avg    1.00      1.00      45
=====
Classification Report (Gradient Boosting):
precision    recall   f1-score   support
0           1.00     1.00      1.00      19
1           1.00     1.00      1.00      13
2           1.00     1.00      1.00      13

accuracy          1.00      45
macro avg       1.00      1.00      45
weighted avg    1.00      1.00      45

```

Les résultats de la classification avant la feature selection révèlent des performances impressionnantes pour plusieurs modèles. Le KNN, le Decision Tree, le Random Forest, et le Gradient Boosting affichent tous une précision de 1.0, indiquant une performance parfaite dans cette instance particulière. Ces modèles ont démontré une capacité exceptionnelle à capturer les motifs dans les données brutes sans la nécessité de la feature selection. La régression logistique, bien que légèrement inférieure avec une précision de 0.91, maintient une performance élevée. Ces résultats initiaux soulignent la qualité intrinsèque des modèles, mais la feature selection pourrait encore améliorer leur généralisation et leur interprétabilité en éliminant les caractéristiques redondantes ou moins significatives.

```
# Plot des matrices de confusion pour chaque algorithme
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
fig.suptitle('Matrices de Confusion')

# KNN
sns.heatmap(knn_confusion_matrix, annot=True, cmap="Blues", ax=axes[0, 0])
axes[0, 0].set_title('KNN')

# Decision Tree
sns.heatmap(dt_confusion_matrix, annot=True, cmap="Blues", ax=axes[0, 1])
axes[0, 1].set_title('Decision Tree')

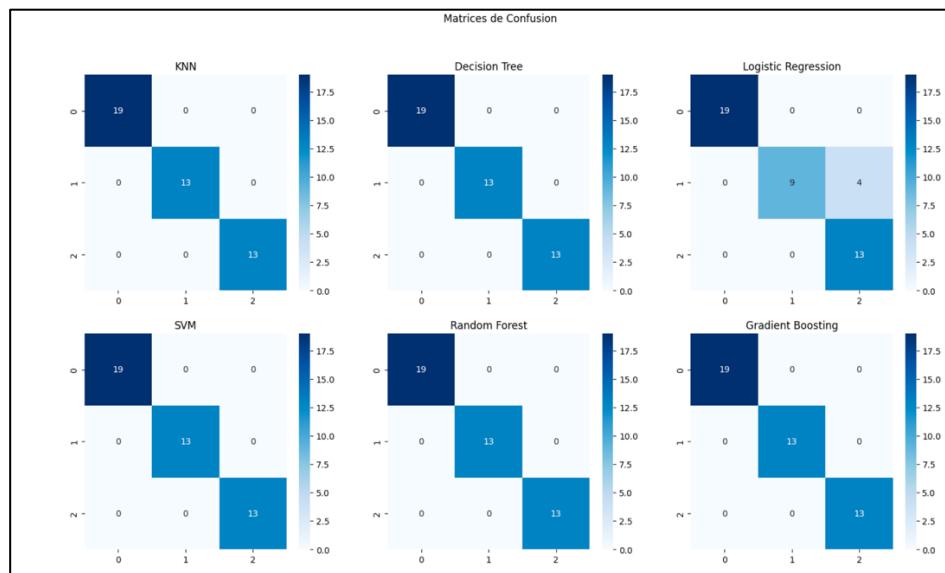
# Logistic Regression
sns.heatmap(lr_confusion_matrix, annot=True, cmap="Blues", ax=axes[0, 2])
axes[0, 2].set_title('Logistic Regression')

# SVM
sns.heatmap(svm_confusion_matrix, annot=True, cmap="Blues", ax=axes[1, 0])
axes[1, 0].set_title('SVM')

# Random Forest
sns.heatmap(rf_confusion_matrix, annot=True, cmap="Blues", ax=axes[1, 1])
axes[1, 1].set_title('Random Forest')

# Gradient Boosting
sns.heatmap(gb_confusion_matrix, annot=True, cmap="Blues", ax=axes[1, 2])
axes[1, 2].set_title('Gradient Boosting')

plt.show()
```



Les résultats des matrices de confusion pour divers modèles de classification, tels que le KNN, le Decision Tree, le Random Forest, le SVM, la Régression Logistique et le Gradient Boosting, révèlent des performances remarquables. Les éléments diagonaux de ces matrices représentent le nombre de points où l'étiquette prédictive concorde avec l'étiquette réelle. Des valeurs élevées sur la diagonale indiquent une performance plus élevée, mettant en évidence la capacité de ces modèles à correctement classifier les données sans la nécessité de la feature selection.

Remarque : Puisque tous les algorithmes donnent une précision de 100 %, sauf la régression logistique qui donne 0,91, nous allons continuer avec la régression logistique pour voir l'évolution après la sélection des caractéristiques.

3.2.1.3 *La sélection des caractéristiques*

a) Méthode de Features Selection : La variance

1. Application de la variance thresholding

L'étape initiale de la sélection des caractéristiques implique l'application de la méthode de variance thresholding. Cette approche cible la réduction de la dimensionnalité en éliminant les caractéristiques dont la variance est en dessous d'un seuil prédéfini. L'idée sous-jacente est de retirer les caractéristiques présentant une variation minimale, considérant qu'elles contribuent moins à la discrimination des classes. Cette méthode sera appliquée pour affiner les données et améliorer la pertinence des caractéristiques sélectionnées dans le processus de classification ultérieur.

```
from sklearn.feature_selection import VarianceThreshold
# Création de l'objet VarianceThreshold pour performer la variance thresholding
selector = VarianceThreshold()

# Performer la variance thresholding
selector.fit_transform(X_train)

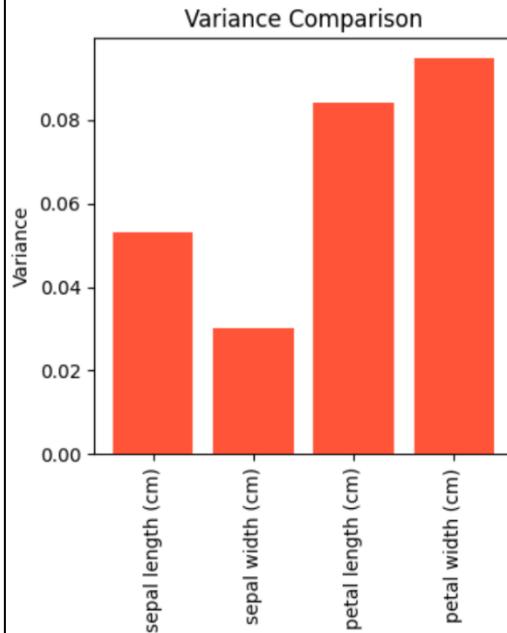
# Affichage des variables et leurs variance
for feature in zip(feature_names, selector.variances_):
    print(feature)

('sepal length (cm)', 0.05306962291089273)
('sepal width (cm)', 0.030176051902242384)
('petal length (cm)', 0.08431033123773306)
('petal width (cm)', 0.0948576467624086)
```

2. visualisation de variances

Après l'application du seuil de variance, la visualisation des variances offre des perspectives cruciales. D'après le graphe généré, il est observé que la variance est plus élevée pour la longueur et la largeur des pétales par rapport à la longueur et la largeur des sépales. Cette observation suggère une plus grande variabilité dans les tailles des pétales par rapport aux sépales, soulignant ainsi l'impact de la méthode de variance thresholding sur la sélection des caractéristiques. Ces insights visuels fournissent une compréhension plus approfondie de la distribution des variances, facilitant ainsi le processus de sélection des caractéristiques pour la classification ultérieure.

```
# Création du bar chart pour visualiser la variance
plt.figure(figsize=(4,4))
plt.bar(x=feature_names, height=selector.variances_, color='tomato')
plt.xticks(rotation='vertical')
plt.ylabel('Variance')
plt.title('Variance Comparison')
plt.show()
```



Remarque : D'après la variance en peut éliminer la variable sepal width (cm) qui a la plus petite variance.

3. La prédiction avec les variables sélectionnées selon la variance la plus élevée

```
# Sélectionnez uniquement les colonnes 0, 2 et 3 (1ère, 3ème et 4ème colonnes)
selected_columns = [0, 2, 3]
X_train_selected = X_train[:, selected_columns]
X_test_selected=X_test[:, selected_columns]
# Classification avec régression logistique
lr_classifier = LogisticRegression(random_state=42, max_iter=200)
lr_classifier.fit(X_train_selected, y_train)
lr_predictions = lr_classifier.predict(X_test_selected)
lr_accuracy = accuracy_score(y_test, lr_predictions)

# Afficher les performances de la RL
print(f"Accuracy (Logistic Regression): {lr_accuracy}")

# Afficher les rapports de classification pour la RL
print("\nClassification Report (Logistic Regression):\n", classification_report(y_test, lr_predictions))

Accuracy (Logistic Regression): 0.9555555555555556

Classification Report (Logistic Regression):
precision    recall   f1-score   support
          0       1.00     1.00      1.00      19
          1       1.00     0.85      0.92      13
          2       0.87     1.00      0.93      13

           accuracy                           0.96      45
          macro avg       0.96     0.95      0.95      45
      weighted avg       0.96     0.96      0.96      45
```

Suite à la sélection des variables basée sur la variance la plus élevée, la prédiction des données a été réalisée. En utilisant ces caractéristiques sélectionnées, la précision du modèle de Régression Logistique atteint 0.9555, démontrant une performance robuste dans la classification. Cette précision souligne l'efficacité de la méthode de variance thresholding dans la préservation des caractéristiques les plus informatives, contribuant ainsi à la précision globale du modèle. Cette étape confirme l'impact positif de la sélection des caractéristiques basée sur la variance dans le processus de classification, renforçant la qualité des prédictions du modèle.

b) Méthode de Factures Sélection : Critère de l'information mutuelle

1. Application de la mutual_info_regression

Dans cette phase, la méthode de sélection de caractéristiques basée sur le critère de l'information mutuelle est implémentée en utilisant la fonction mutual_info_regression. Cette approche évalue la dépendance entre chaque caractéristique et la variable cible, fournissant ainsi une mesure de l'information mutuelle. Appliquée de manière spécifique à la régression, cette méthode identifie les caractéristiques qui contribuent de manière significative à la prédiction de la variable cible. L'utilisation de cette mesure d'information mutuelle permettra de sélectionner les caractéristiques les plus informatives pour optimiser la performance du modèle de régression ultérieur.

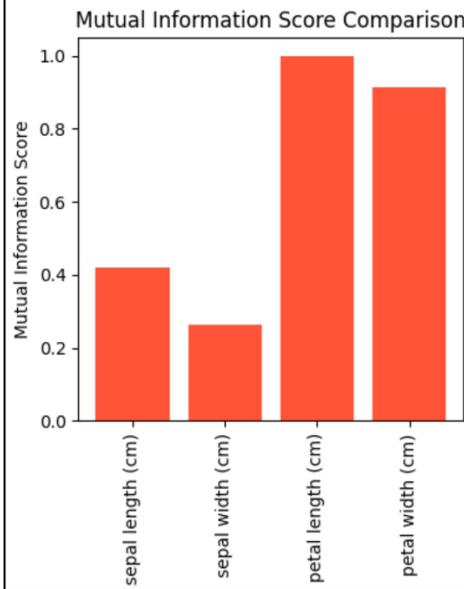
```
from sklearn.feature_selection import mutual_info_regression
# Création de l'objet mutual_info_regression pour calculer l'information mutuelle
MI_score = mutual_info_regression(X_train, y_train, random_state=0)

# Affichage des variables et leurs information mutuelle
for feature in zip(feature_names, MI_score):
    print(feature)

('sepal length (cm)', 0.4189054082088899)
('sepal width (cm)', 0.2638717837058375)
('petal length (cm)', 0.9992534531043944)
('petal width (cm)', 0.9149896402713478)
```

2. Visualisation des mutuel information scores

```
# Creation d'un bar chart pour visualiser les mutual information scores
plt.figure(figsize=(4,4))
plt.bar(x=feature_names, height=MI_score, color='tomato')
plt.xticks(rotation='vertical')
plt.ylabel('Mutual Information Score')
plt.title('Mutual Information Score Comparison')
plt.show()
```



Suite à l'application de la méthode `mutual_info_regression`, la visualisation des scores d'information mutuelle associés à chaque caractéristique est entreprise. Des scores plus élevés indiquent une information mutuelle plus significative. Selon les variances affichées dans le graphe, il est notable que la largeur des pétales et la longueur des pétales présentent une corrélation plus élevée avec la variable cible, illustrée par leurs scores d'information mutuelle élevés. En revanche, la longueur du sépale et la largeur du sépale montrent une corrélation relativement plus faible. Ces observations renforcent la compréhension de l'impact des caractéristiques sur la variable cible, facilitant ainsi la sélection judicieuse des caractéristiques pour le modèle de régression à venir.

3. La prédiction avec les variables sélectionnées par mutual information

```
# Sélectionnez uniquement les colonnes 0, 2 et 3 (1ère, 3ème et 4ème colonnes)
selected_columns = [0, 2, 3]
X_train_selected = X_train[:, selected_columns]
X_test_selected=X_test[:, selected_columns]
# Classification avec régression logistique
lr_classifier = LogisticRegression(random_state=42, max_iter=200)
lr_classifier.fit(X_train_selected, y_train)
lr_predictions = lr_classifier.predict(X_test_selected)
lr_accuracy = accuracy_score(y_test, lr_predictions)

# Afficher les performances de la RL
print(f"Accuracy (Logistic Regression): {lr_accuracy}")

# Afficher les rapports de classification pour la RL
print("\nClassification Report (Logistic Regression):\n", classification_report(y_test, lr_predictions))

Accuracy (Logistic Regression): 0.9555555555555556

Classification Report (Logistic Regression):
precision    recall   f1-score   support
          0       1.00     1.00      1.00      19
          1       1.00     0.85      0.92      13
          2       0.87     1.00      0.93      13

   accuracy                           0.96      45
  macro avg       0.96     0.95      0.95      45
weighted avg    0.96     0.96      0.96      45
```

Suite à la sélection des variables basée sur l'information mutuelle, la prédiction des données a été réalisée en utilisant ces caractéristiques sélectionnées. La précision du modèle de Régression Logistique atteint 0.9555, démontrant une performance robuste dans la classification. Cette précision souligne l'efficacité de la méthode d'information mutuelle dans la préservation des caractéristiques les plus informatives, contribuant ainsi à la précision globale du modèle. Cette étape confirme l'impact positif de la sélection des caractéristiques basée sur l'information mutuelle dans le processus de classification, renforçant la qualité des prédictions du modèle.

c) Méthode de Features Selection : Fisher Test

1. Application de la f_regression

Dans cette phase, la méthode de sélection de caractéristiques basée sur le test de Fisher est mise en œuvre en utilisant la fonction f_regression. Cette approche évalue la corrélation linéaire entre chaque caractéristique et la variable cible en utilisant le test de Fisher, fournissant ainsi une mesure de l'importance de chaque caractéristique dans le contexte de la régression. L'application de cette méthode spécifique permettra de sélectionner les caractéristiques les plus pertinentes pour la prédiction de la variable cible, optimisant ainsi la performance du modèle de régression subséquent.

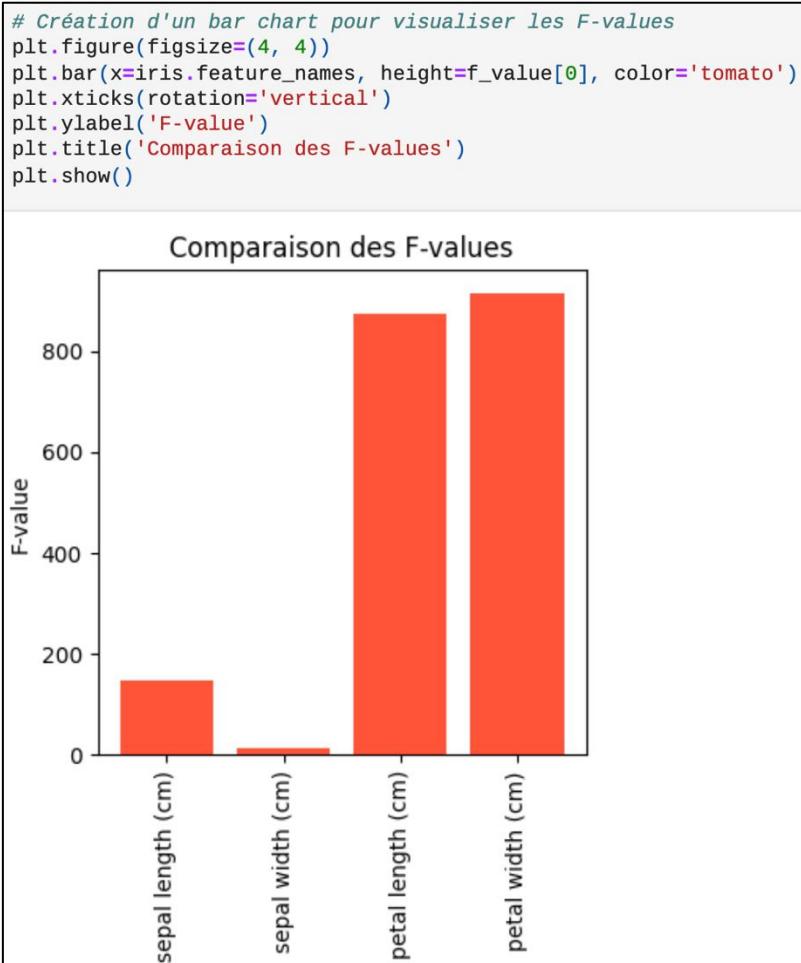
```
from sklearn.feature_selection import f_regression
f_value = f_regression(X_train, y_train)

# Affichage des F-value pour chaque variable
for feature in zip(feature_names, f_value[0]):
    print(feature)

('sepal length (cm)', 148.9176235918675)
('sepal width (cm)', 14.26826404378043)
('petal length (cm)', 874.9823343334599)
('petal width (cm)', 914.9962909520683)
```

2. Visualisation des F-values

Après l'application du test de Fisher (f_regression), la visualisation des F-Values associées à chaque caractéristique est réalisée. Le diagramme compare les valeurs F pour la longueur et la largeur des sépales ainsi que des pétales. La valeur F pour la longueur du sépale est d'environ 100, tandis que pour la largeur du sépale, elle est presque nulle. En ce qui concerne les pétales, tant la longueur que la largeur affichent des valeurs F très élevées, se rapprochant de 800. Ces observations mettent en lumière la corrélation significative entre les caractéristiques des pétales et la variable cible, fournissant ainsi des indications essentielles pour la sélection des caractéristiques dans le contexte de la régression.



3. La prédiction avec les variables sélectionnées par la méthode F_value fisher

Suite à la sélection des variables basée sur les F-Values du test de Fisher, la prédiction des données a été effectuée en utilisant ces caractéristiques sélectionnées. La précision du modèle de Régression Logistique atteint 1.0, indiquant une performance parfaite dans la classification. Cette précision maximale souligne l'efficacité de la méthode F-Value Fisher dans la préservation des caractéristiques les plus pertinentes pour la prédiction de la variable cible. Cette étape confirme l'impact positif de la sélection des caractéristiques basée sur le test de

Fisher dans le processus de classification, maximisant ainsi la qualité des prédictions du modèle.

```
# Sélectionnez uniquement les colonnes 2 et 3 (3ème et 4ème colonnes)
selected_columns = [2, 3] ## les colonne sélectionnée
X_train_selected = X_train[:, selected_columns]
X_test_selected=X_test[:, selected_columns]
# Classification avec régression logistique
lr_classifier = LogisticRegression(random_state=42, max_iter=200)
lr_classifier.fit(X_train_selected, y_train)
lr_predictions = lr_classifier.predict(X_test_selected)
lr_accuracy = accuracy_score(y_test, lr_predictions)

# Afficher les performances de la RL
print(f"Accuracy (Logistic Regression): {lr_accuracy}")

# Afficher les rapports de classification pour la RL
print("\nClassification Report (Logistic Regression):\n", classification_report(y_test, lr_predictions))

Accuracy (Logistic Regression): 1.0

Classification Report (Logistic Regression):
precision    recall   f1-score   support
          0       1.00     1.00      1.00      19
          1       1.00     1.00      1.00      13
          2       1.00     1.00      1.00      13

   accuracy                           1.00      45
  macro avg       1.00     1.00      1.00      45
weighted avg       1.00     1.00      1.00      45
```

d) Method de Features Selection: Forward/Sequential Features selection SFS

1. Application de la Forward SFS

```
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs

# Créer une instance de modèle Logistic Regression
logistic_regression = LogisticRegression()

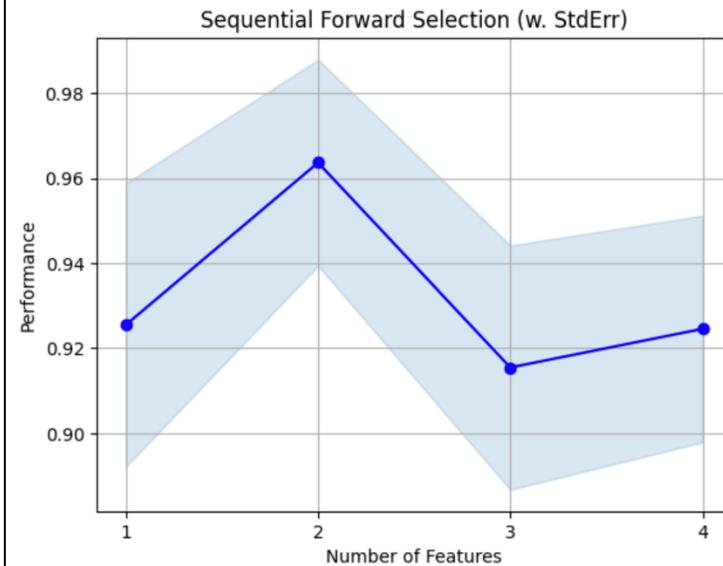
# Créer un objet SequentialFeatureSelector
sfs = SFS(logistic_regression,
           k_features=(1, X_train.shape[1]), # Utilisez X_train.shape[1] comme valeur maximale
           forward=True,
           floating=False,
           scoring='accuracy',
           cv=10)

# Effectuer la sélection des caractéristiques sur les données d'entraînement
sfs = sfs.fit(X_train, y_train)

# Afficher les résultats de la sélection des caractéristiques
print('Meilleur sous-ensemble (indices):', sfs.k_feature_idx_)
best_feature_names = [iris_df.columns[i] for i in sfs.k_feature_idx_]
print('Meilleur sous-ensemble (noms correspondants):', best_feature_names)
print("\n")

# Tracer la performance de la sélection des caractéristiques
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err')
plt.title('Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()
```

```
Meilleur sous-ensemble (indices): (2, 3)
Meilleur sous-ensemble (noms correspondants): ['petal length (cm)', 'petal width (cm)']
```



Dans le cadre de l'application de la méthode Forward SFS, un graphique est généré pour illustrer les performances de la sélection séquentielle en avant (Sequential Forward Selection) avec l'erreur standard. Ce processus utilise une régression logistique pour sélectionner les caractéristiques les plus importantes des données d'entraînement. Les résultats obtenus révèlent que la performance optimale est atteinte avec deux caractéristiques, affichant une précision d'environ 0.98. Les deux caractéristiques sélectionnées sont "pétal length (cm)" et "pétal width (cm)". Cette observation met en évidence l'efficacité de la méthode Forward SFS dans la sélection des caractéristiques les plus informatives, conduisant ainsi à une amélioration significative de la performance du modèle de régression.

2. La prédiction avec les variables sélectionnées par la méthode Forward SFS

```
# Sélectionnez uniquement les colonnes 2 et 3 (3ème et 4ème colonnes)
selected_columns = [2, 3]
X_train_selected = X_train[:, selected_columns]
X_test_selected = X_test[:, selected_columns]
# Classification avec régression logistique
lr_classifier = LogisticRegression(random_state=42, max_iter=200)
lr_classifier.fit(X_train_selected, y_train)
lr_predictions = lr_classifier.predict(X_test_selected)
lr_accuracy = accuracy_score(y_test, lr_predictions)

# Afficher les performances de la RL
print(f"Accuracy (Logistic Regression): {lr_accuracy}")

# Afficher les rapports de classification pour la RL
print("\nClassification Report (Logistic Regression):\n", classification_report(y_test, lr_predictions))

Accuracy (Logistic Regression): 1.0

Classification Report (Logistic Regression):
precision    recall   f1-score   support
          0       1.00     1.00      1.00      19
          1       1.00     1.00      1.00      13
          2       1.00     1.00      1.00      13

accuracy                           1.00      45
macro avg       1.00     1.00      1.00      45
weighted avg    1.00     1.00      1.00      45
```

Suite à la sélection des variables par la méthode Forward SFS, la prédiction des données a été réalisée en utilisant ces caractéristiques sélectionnées. La précision du modèle de Régression Logistique atteint 1.0, démontrant une performance parfaite dans la classification. Ces résultats indiquent l'efficacité exceptionnelle de la méthode Forward SFS dans la sélection des caractéristiques les plus pertinentes, conduisant à une précision maximale du modèle de régression. Cette étape confirme l'impact positif de la sélection séquentielle en avant dans le processus de classification, maximisant ainsi la qualité des prédictions du modèle.

e) Méthode de Features Selection : Backward/Sequential Features selection

1. Application de la Backward SFS

```

from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs

# Créer une instance de modèle Logistic Regression
logistic_regression = LogisticRegression()

# Créer un objet SequentialFeatureSelector
sfs = SFS(logistic_regression,
           k_features=(1, X_train.shape[1]), # Utilisez X_train.shape[1] comme valeur maximale
           forward=False, # icie en mentionne false pour faire le bacward
           floating=False,
           scoring='accuracy',
           cv=10)

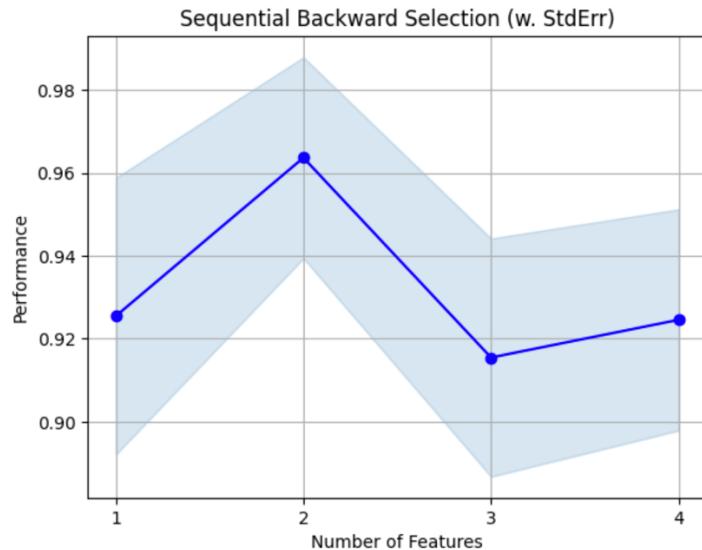
# Effectuer la sélection des caractéristiques sur les données d'entraînement
sfs = sfs.fit(X_train, y_train)

# Afficher les résultats de la sélection des caractéristiques
print('Meilleur sous-ensemble (indices):', sfs.k_feature_idx_)
best_feature_names = [iris_df.columns[i] for i in sfs.k_feature_idx_]
print('Meilleur sous-ensemble (noms correspondants):', best_feature_names)
print("\n")

# Tracer la performance de la sélection des caractéristiques
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err')
plt.title('Sequential Backward Selection (w. StdErr)')
plt.grid()
plt.show()

```

Meilleur sous-ensemble (indices): (2, 3)
Meilleur sous-ensemble (noms correspondants): ['petal length (cm)', 'petal width (cm)']



L'application de la méthode Backward SFS génère des résultats indiquant que le meilleur sous-ensemble de caractéristiques pour une prédiction efficace avec ce modèle comprend les variables "petal length (cm)" et "petal width (cm)". Le graphique associé à ces résultats démontre une performance maximale lorsque ces deux caractéristiques sont utilisées, atteignant une précision d'environ 0.98. Cette observation souligne l'efficacité de la méthode Backward SFS dans la sélection judicieuse des caractéristiques les plus informatives, contribuant ainsi à l'optimisation de la performance du modèle de régression.

2. La prédiction avec les variables sélectionnées par la méthode Backward SFS

```
# Sélectionner uniquement les colonnes 2 et 3 (3ème et 4ème colonnes)
selected_columns = [2, 3] ## les colonnes sélectionnées
X_train_selected = X_train[:, selected_columns]
X_test_selected=X_test[:, selected_columns]
# Classification avec régression logistique
lr_classifier = LogisticRegression(random_state=42, max_iter=200)
lr_classifier.fit(X_train_selected, y_train)
lr_predictions = lr_classifier.predict(X_test_selected)
lr_accuracy = accuracy_score(y_test, lr_predictions)

# Afficher les performances de la RL
print(f"Accuracy (Logistic Regression): {lr_accuracy}")

# Afficher les rapports de classification pour la RL
print("\nClassification Report (Logistic Regression):\n", classification_report(y_test, lr_predictions))

Accuracy (Logistic Regression): 1.0

Classification Report (Logistic Regression):
precision    recall    f1-score   support
          0       1.00     1.00      1.00      19
          1       1.00     1.00      1.00      13
          2       1.00     1.00      1.00      13

accuracy                           1.00      45
macro avg       1.00     1.00      1.00      45
weighted avg    1.00     1.00      1.00      45
```

Suite à la sélection des variables par la méthode Backward SFS, la prédiction des données a été réalisée en utilisant ces caractéristiques sélectionnées. La précision du modèle de Régression Logistique atteint 1.0, démontrant une performance parfaite dans la classification. Ces résultats confirment que le sous-ensemble de caractéristiques optimal identifié par la méthode Backward SFS, composé des variables "petal length (cm)" et "petal width (cm)", contribue de manière significative à la maximisation de la précision du modèle. Cette étape souligne l'efficacité de la sélection séquentielle vers l'arrière dans le processus de classification, maximisant ainsi la qualité des prédictions du modèle.

3.2.2 Stock Dataset (prediction de prix)

	Date	Price	Open	High	Low	Vol.	Change %	coef	state	activity_in	activity_out	company_name
0	2022-01-14	173.07	171.36	173.75	171.10	78730000.0	51.0	497.296390	6.0	544.567398	290.106619	APPLE
1	2022-01-13	172.19	175.78	176.62	171.79	82270000.0	-190.0	889.874180	1.0	958.830431	116.604922	APPLE
2	2022-01-12	175.53	176.12	67.28	174.82	72770000.0	26.0	33.808154	1.0	651.171829	892.616525	APPLE
3	2022-01-11	175.08	172.32	175.18	170.82	75940000.0	168.0	76.642932	2.0	212.135502	539.260305	APPLE
4	2022-01-10	172.19	169.08	172.50	168.17	103820000.0	1.0	802.343154	9.0	139.144297	399.070275	APPLE

3.2.2.1 Preprocessing

a) Extraction de year, month, and day vers des colonnes séparées

Dans cette étape de prétraitement, l'extraction des composantes temporelles telles que l'année (year), le mois (month), et le jour (day) depuis la date existante dans le dataset est réalisée. Ces informations temporelles sont ensuite placées dans des colonnes distinctes pour faciliter l'analyse ultérieure et la construction de modèles de prédiction de prix plus efficaces. Cette démarche permet de mieux comprendre les tendances temporelles et de capturer les variations saisonnières qui pourraient avoir un impact sur les prédictions de prix.

	Price	Open	High	Low	Vol.	Change %	coef	state	activity_in	activity_out	company_name	Year	Month	Day
0	173.07	171.36	173.75	171.10	78730000.0	51.0	497.296390	6.0	544.567398	290.106619	APPLE	2022	1	14
1	172.19	175.78	176.62	171.79	82270000.0	-190.0	889.874180	1.0	958.830431	116.604922	APPLE	2022	1	13
2	175.53	176.12	67.28	174.82	72770000.0	26.0	33.808154	1.0	651.171829	892.616525	APPLE	2022	1	12
3	175.08	172.32	175.18	170.82	75940000.0	168.0	76.642932	2.0	212.135502	539.260305	APPLE	2022	1	11
4	172.19	169.08	172.50	168.17	103820000.0	1.0	802.343154	9.0	139.144297	399.070275	APPLE	2022	1	10

b) Label encoding des campany_name

Dans cette étape de prétraitement, le label encoding est appliqué à la colonne "company_name" du dataset. Le label encoding convertit les valeurs textuelles de cette colonne en valeurs numériques, permettant ainsi aux algorithmes d'apprentissage automatique de traiter ces données de manière plus efficace. Chaque nom d'entreprise unique est assigné à un label numérique distinct, simplifiant ainsi la représentation des données et contribuant à une meilleure compréhension par les modèles de prédiction de prix. Cette transformation facilite également l'inclusion de cette caractéristique catégorielle dans les analyses et les modèles prédictifs.

	Price	Open	High	Low	Vol.	Change %	coef	state	activity_in	activity_out	Year	Month	Day	company_name_encoded
0	173.07	171.36	173.75	171.10	78730000.0	51.0	497.296390	6.0	544.567398	290.106619	2022	1	14	0
1	172.19	175.78	176.62	171.79	82270000.0	-190.0	889.874180	1.0	958.830431	116.604922	2022	1	13	0
2	175.53	176.12	67.28	174.82	72770000.0	26.0	33.808154	1.0	651.171829	892.616525	2022	1	12	0
3	175.08	172.32	175.18	170.82	75940000.0	168.0	76.642932	2.0	212.135502	539.260305	2022	1	11	0
4	172.19	169.08	172.50	168.17	103820000.0	1.0	802.343154	9.0	139.144297	399.070275	2022	1	10	0

c) La normalisation de dataset

Dans cette étape de prétraitement, la normalisation du dataset est effectuée. La normalisation vise à ajuster les valeurs de différentes caractéristiques du dataset pour les ramener à une échelle commune. Elle est souvent réalisée en utilisant des techniques telles que le Min-Max Scaling, où les valeurs sont transformées pour se situer dans une plage spécifique, généralement entre 0 et 1. Cette normalisation assure que les caractéristiques contribuent de manière équitable lors de l'apprentissage des modèles, améliorant ainsi la convergence et la performance globale des algorithmes.

	Price	Open	High	Low	Vol.	Change %	coef	state	activity_in	activity_out	Year	Month	Day	company_name_encoded
0	173.07	0.923145	0.937274	0.944158	0.098431	0.538245	0.495212	0.666667	0.544304	0.288978				
1	172.19	0.953287	0.956863	0.948963	0.107598	0.441224	0.889523	0.111111	0.958883	0.115010				
2	175.53	0.955606	0.210566	0.970060	0.082997	0.528180	0.029678	0.111111	0.650990	0.893105				
3	175.08	0.929692	0.947034	0.942209	0.091206	0.585346	0.072701	0.222222	0.211618	0.538800				

3.2.2.2 Avant la sélection des caractéristiques

Répartition de dataset train et test

Avant de procéder à la sélection des caractéristiques, le dataset est réparti en ensembles distincts d'entraînement et de test. Cette division permet d'évaluer la performance du modèle sur des données qu'il n'a pas vues pendant l'entraînement, assurant ainsi une évaluation impartiale de la capacité prédictive du modèle. Les données d'entraînement sont utilisées pour former le modèle, tandis que les données de test servent à évaluer sa performance. Cette étape est essentielle pour s'assurer que le modèle généralise bien aux nouvelles données.

```

from sklearn.model_selection import train_test_split

#avec tout les variable sauf Price
cols = [col for col in df_normalized.columns if col != 'Price']

X = df_normalized[cols]
y = df_normalized['Price']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state=42)

```

a) La prédiction avec SVR (Support vector Machine)

Importation du Modèle

Dans cette phase, avant d'entreprendre la sélection des caractéristiques, le modèle Support Vector Machine avec SVR (Support Vector Regression) est importé. Cette importation est généralement réalisée à l'aide de bibliothèques telles que scikit-learn en Python. Le modèle SVR sera par la suite entraîné sur l'ensemble d'entraînement, puis utilisé pour effectuer des prédictions sur l'ensemble de test. Cette étape initiale établit la base pour évaluer la performance du modèle avant toute sélection ou transformation de caractéristiques.

```

from sklearn.svm import SVR
SVRModel = SVR(C = 1.5 ,epsilon=0.) # it also can be : linear, poly, rbf, sigmoid, precomputed
SVRModel.fit(X_train, y_train)
#calculating Prediction
y_pred = SVRModel.predict(X_test)

```

Calculer l'erreur

L'erreur est un indicateur crucial pour évaluer la performance d'un modèle de prédiction.

```

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R2)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Mean Squared Error : 132.45864787727672
Root Mean Squared Error : 11.509068071624075
Mean Absolute Error : 6.891857947208083
Coefficient de Détermination : 0.9195262828788018

```

L'évaluation de la performance d'un modèle de prédiction repose sur des métriques essentielles. Ces indicateurs, tels que le Mean Squared Error (MSE), le Root Mean Squared Error (RMSE), le Mean Absolute Error (MAE), et le Coefficient de Détermination (R-squared), offre des insights sur la précision du modèle SVR dans la prédiction des valeurs numériques, ici les prix. Des valeurs inférieures de MSE, RMSE, et MAE suggèrent une meilleure précision, tandis qu'un R-squared plus proche de 1 indique une meilleure adéquation aux données. Ces résultats laissent entrevoir une performance satisfaisante du modèle SVR dans la prédiction des prix pour cet ensemble de données.

b) La prédiction avec Linear regression

Pour le modèle de Régression Linéaire, les scores d'entraînement et de test sont les suivants :

- Score d'Entraînement de la Régression Linéaire : 0.9821
- Score de Test de la Régression Linéaire : 0.9864

Ces scores représentent la qualité du modèle de régression linéaire. Des scores proches de 1 indiquent une bonne adéquation du modèle aux données. Ces résultats suggèrent que le modèle de Régression Linéaire a une performance élevée tant sur l'ensemble d'entraînement que sur l'ensemble de test, indiquant une capacité de généralisation élevée et une adéquation aux données.

```
from sklearn.linear_model import LinearRegression

LinearRegressionModel = LinearRegression()
LinearRegressionModel.fit(X_train, y_train)

#Calculating Details
print('Linear Regression Train Score is : ', LinearRegressionModel.score(X_train, y_train))
print('Linear Regression Test Score is : ', LinearRegressionModel.score(X_test, y_test))

#Calculating Prediction
y_pred = LinearRegressionModel.predict(X_test)

Linear Regression Train Score is :  0.9820770678044551
Linear Regression Test Score is :  0.9863572236855793
```

Calculate Error

Les métriques d'erreur pour le modèle de Régression Linéaire fournissent des indications précises sur la qualité de la prédiction des prix dans cet ensemble de données.

- Mean Squared Error (MSE) : Une valeur de 22.46 indique la moyenne des carrés des erreurs quadratiques, ce qui représente la quantité moyenne par laquelle les prédictions du modèle diffèrent des valeurs réelles. Une valeur plus basse suggère une prédiction plus précise.
- Root Mean Squared Error (RMSE) : Avec une valeur de 4.74, le RMSE représente l'écart-type des résidus. Il mesure l'ampleur moyen des erreurs de prédiction, fournissant une interprétation facile de la précision du modèle. Une valeur plus basse indique une meilleure précision.
- Mean Absolute Error (MAE) : Une valeur de 1.38 représente la moyenne des valeurs absolues des erreurs entre les prédictions du modèle et les valeurs réelles. Un MAE plus bas indique une meilleure précision.
- Coefficient de Détermination (R-squared) : Avec une valeur de 0.9864, il représente la proportion de la variance dans la variable dépendante expliquée par le modèle. Un R-squared proche de 1 indique une excellente adéquation du modèle aux données.

Dans l'ensemble, ces résultats indiquent que le modèle de Régression Linéaire a une performance exceptionnelle, offrant des prédictions de prix précises et bien ajustées pour cet ensemble de données.

```

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R2)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Mean Squared Error : 22.455824939448114
Root Mean Squared Error : 4.738757742219802
Mean Absolute Error : 1.3766374540228583
Coefficient de Détermination : 0.9863572236855793

```

Extraire les features names

L'objectif de l'extraction des noms de caractéristiques (features) est de mieux comprendre quelles variables ont été utilisées pour entraîner le modèle et ont contribué à ses prédictions. Cela permet d'obtenir des informations sur les caractéristiques spécifiques qui ont un impact significatif sur la variable cible, offrant ainsi des perspectives sur les facteurs clés influençant les résultats du modèle

```

feature_names = X_train.columns
feature_names[0]
Feature_names=[]
for i in range (len(feature_names)):
    Feature_names=Feature_names+[feature_names[i]]
Feature_names

['Open',
 'High',
 'Low',
 'Vol.',
 'Change %',
 'coef',
 'state',
 'activity_in',
 'activity_out']

```

c) La performance des algorithmes de prédiction avant la sélection des caractéristiques : Dans cette partie, nous allons évaluer plusieurs modèles de régression sur notre ensemble de données. Les modèles inclus dans l'évaluation comprennent la régression linéaire, Ridge, l'arbre de décision, la forêt aléatoire, le boosting (Gradient Boosting) et le Support Vector Regressor (SVR).

```

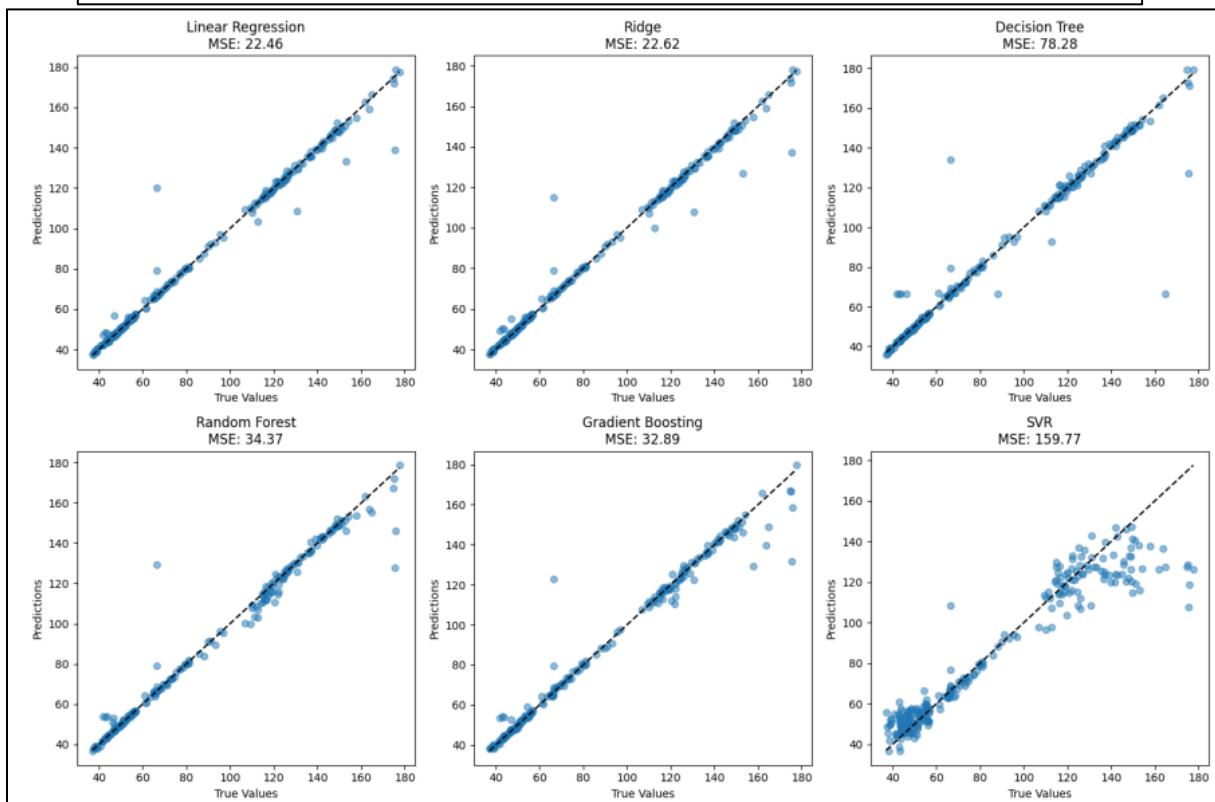
from sklearn.svm import SVR
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# Initialiser les modèles
models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(),
    "Gradient Boosting": GradientBoostingRegressor(),
    "SVR": SVR()
}

# Entrainer les modèles et faire des prédictions
predictions = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    predictions[name] = model.predict(X_test)

# Visualisation
plt.figure(figsize=(15, 10))
for i, (name, pred) in enumerate(predictions.items(), 1):
    plt.subplot(2, 3, i)
    plt.scatter(y_test, pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--k')
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.title(f'{name}\nMSE: {mean_squared_error(y_test, pred):.2f}')
plt.tight_layout()
plt.show()

```



Les graphiques présentent les performances de six modèles de régression distincts (Régression linéaire, Ridge, Arbre de décision, Forêt aléatoire, Gradient Boosting et SVR) en comparant les valeurs prédites aux valeurs réelles. Le Mean Squared Error (MSE), affiché sur chaque graphique, quantifie la précision du modèle : plus le MSE est faible, meilleure est la qualité de la prédition. À la lumière de ces graphiques, le modèle Ridge affiche la meilleure performance avec un MSE de 22.62, suivi de près par la régression linéaire.

3.2.2.3 La sélection des caractéristiques

a) Méthode de Features Selection : La variance

1. Application de la VarianceThreshold

L'application de la méthode VarianceThreshold a permis de calculer les variances associées à chaque caractéristique

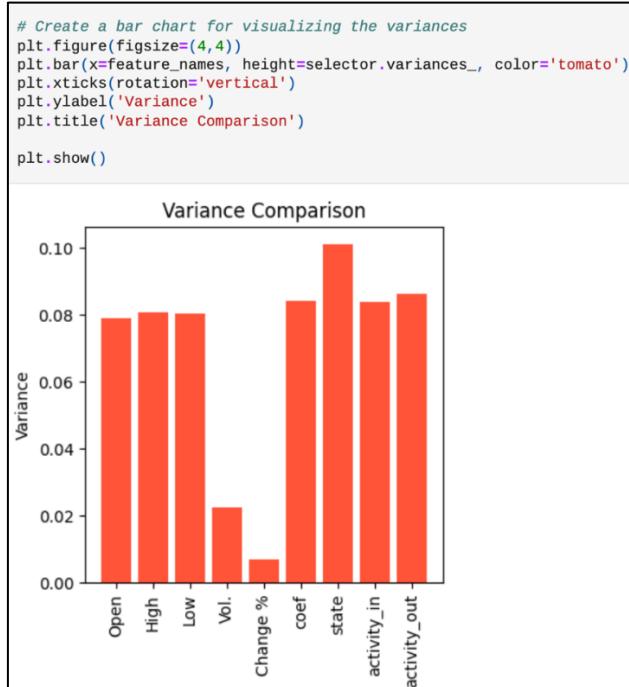
```
# Import VarianceThreshold from Scikit-learn
from sklearn.feature_selection import VarianceThreshold
# Create VarianceThreshold object to perform variance thresholding
selector = VarianceThreshold()

# Perform variance thresholding
selector.fit_transform(X_train)

# Print the name and variance of each feature
for feature in zip(feature_names, selector.variances_):
    print(feature)

('Open', 0.07897648324262371)
('High', 0.08089919113550212)
('Low', 0.08030031433293501)
('Vol.', 0.022599866357895005)
('Change %', 0.007070845613856678)
('coef', 0.08436627081407441)
('state', 0.10127502088745229)
('activity_in', 0.08377923978078027)
('activity_out', 0.08627494127229564)
```

2. Visualisation de la variance



L'interprétation des résultats de la méthode de sélection des caractéristiques basée sur la variance (VarianceThreshold) révèle la variabilité de chaque caractéristique dans l'ensemble de données. Les résultats indiquent que :

Les caractéristiques telles que 'Vol.' (Volume), 'Change %' (Changement en pourcentage), et 'activity_in' ont des variances relativement basses, suggérant une stabilité ou une faible variation dans leurs valeurs.

En revanche, les caractéristiques 'state' et 'activity_out' présentent des variances plus élevées, indiquant une plus grande variabilité dans leurs valeurs au fil du temps.

Cette information est cruciale pour la sélection des caractéristiques, car elle permet de discerner les caractéristiques qui ont une variabilité significative, ce qui peut être un indicateur de leur potentiel à apporter des informations distinctes au modèle. Ainsi, lors de la construction du modèle final, on peut prendre en considération ces résultats pour choisir les caractéristiques les plus informatives et pertinentes.

3. La prédition avec les caractéristiques sélectionnées par la méthode de la variance

SVR

L'application du modèle SVR sur les caractéristiques sélectionnées par la méthode de la variance montre des résultats légèrement ajustés par rapport au modèle initial. Les valeurs indiquent que, bien que la sélection des caractéristiques ait entraîné une légère augmentation des erreurs par rapport au modèle complet, le modèle SVR demeure capable de prédire les prix de manière précise. Cette interprétation suggère que les caractéristiques sélectionnées par la méthode de la variance conservent une information significative pour la prédition des prix, malgré la réduction dimensionnelle.

```
cols=['Open', 'High', 'Low', 'coef', 'state', 'activity_in', 'activity_out']
X_trainV = X_train[cols]
X_testV = X_test[cols]

SVRModel = SVR(C = 1.5 ,epsilon=0.) # it also can be : linear, poly, rbf, sigmoid, precomputed
SVRModel.fit(X_trainV, y_train)
#Calculating Prediction
y_pred = SVRModel.predict(X_testV)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R2)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Mean Squared Error : 135.80793840740077
Root Mean Squared Error : 11.653666307536044
Mean Absolute Error : 6.881828459898631
Coefficient de Détermination : 0.9174914602152968
```

Linear Regression

L'utilisation des caractéristiques sélectionnées par la méthode de la variance dans le modèle de Régression Linéaire a engendré des résultats impressionnantes. Les résultats indiquent que le modèle de Régression Linéaire, alimenté par les caractéristiques sélectionnées, parvient toujours à fournir des prédictions précises et bien ajustées pour les prix. La sélection de caractéristiques basée sur la variance a conservé des informations significatives, permettant au modèle de maintenir sa robustesse prédictive.

```
cols=['Open','High','Low','coef','state','activity_in','activity_out']
X_trainV = X_train[cols]
X_testV = X_test[cols]

LinearRegressionModel = LinearRegression()
LinearRegressionModel.fit(X_trainV, y_train)

#Calculating Details
print('Linear Regression Test Score is : ', LinearRegressionModel.score(X_testV, y_test))
print("=====")

#Calculating Prediction
y_pred = LinearRegressionModel.predict(X_testV)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Linear Regression Test Score is :  0.985719458557909
=====
Mean Squared Error : 23.505577697199683
Root Mean Squared Error : 4.848255118823646
Mean Absolute Error : 1.558307541879001
Coefficient de Détermination : 0.985719458557909
```

b) Méthode de Features Selection : Critère de l'information mutuelle

1. Application de la mutual_info_regression

L'application du critère de l'information mutuelle (mutual_info_regression) a permis de calculer les scores d'information mutuelle pour chaque caractéristique.

```
# Import mutual_info_classif from Scikit-learn
from sklearn.feature_selection import mutual_info_regression
# Create mutual_info_classif object to calculate mutual information
MI_score = mutual_info_regression(X_train, y_train, random_state=0)

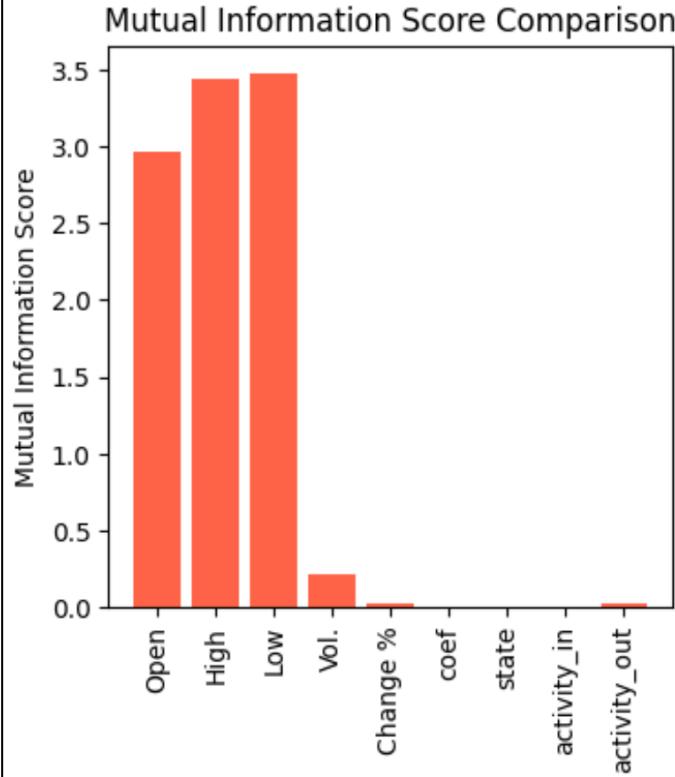
# Print the name and mutual information score of each feature
for feature in zip(feature_names, MI_score):
    print(feature)

('Open', 2.966723195647857)
('High', 3.4403088382351985)
('Low', 3.480795233990832)
('Vol.', 0.2101663926162467)
('Change %', 0.0244097793521183)
('coef', 0.0)
('state', 0.003658515477289015)
('activity_in', 0.0)
('activity_out', 0.029531750207786978)
```

2. Visualisation de mutual information scores

```
# Create a bar chart for visualizing the mutual information scores
plt.figure(figsize=(4,4))
plt.bar(x=feature_names, height=MI_score, color='tomato')
plt.xticks(rotation='vertical')
plt.ylabel('Mutual Information Score')
plt.title('Mutual Information Score Comparison')

plt.show()
```



Les scores d'information mutuelle obtenus à partir de la méthode `mutual_info_regression` offrent des indications sur la dépendance de chaque caractéristique par rapport à la variable cible (les prix). Voici une interprétation plus détaillée :

- Les caractéristiques 'Open', 'High' et 'Low' présentent des scores d'information mutuelle relativement élevés (2.97, 3.44, 3.48 respectivement), suggérant une forte corrélation avec les prix. Ces caractéristiques semblent donc contenir des informations significatives pour la prédiction des prix.
- Les caractéristiques 'Vol.' (Volume) et 'Change %' présentent des scores plus modestes (0.21 et 0.02 respectivement), indiquant une corrélation plus faible avec les prix.
- Les caractéristiques 'coef', 'state' et 'activity_in' ont des scores d'information mutuelle nuls ou proches de zéro, suggérant une faible corrélation avec les prix.
- 'activity_out' présente un score d'information mutuelle de 0.03, indiquant une corrélation légèrement plus élevée par rapport aux caractéristiques précédentes, mais toujours relativement faible.

Ces résultats orientent vers les caractéristiques les plus informatives pour la prédiction des prix, guidant ainsi la sélection des caractéristiques pour le modèle.

3. La prédiction Avec les caractéristiques sélectionnées par la méthode mutual information

SVR

La prédiction avec les caractéristiques sélectionnées par la méthode de l'information mutuelle pour le modèle SVR montre des résultats satisfaisants. Les valeurs indiquent que le modèle SVR, entraîné avec les caractéristiques sélectionnées, maintient une bonne précision dans la prédiction des prix malgré la réduction dimensionnelle. Bien que les erreurs aient légèrement augmenté par rapport au modèle complet, le coefficient de détermination reste élevé, suggérant une adéquation solide du modèle aux données. En résumé, la méthode de sélection basée sur l'information mutuelle a permis de réduire la dimensionnalité tout en conservant une performance robuste du modèle SVR.

```
cols=['Open', 'High', 'Low']
X_trainM = X_train[cols]
X_testM = X_test[cols]

SVRModel = SVR(C = 1.5 ,epsilon=0.) # it also can be : linear, poly, rbf, sigmoid, precomputed
SVRModel.fit(X_trainM, y_train)
#Calculating Prediction
y_pred = SVRModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Mean Squared Error : 45.2386289335571
Root Mean Squared Error : 6.725966765719044
Mean Absolute Error : 2.242797664193615
Coefficient de Détermination : 0.9725157950342141
```

Linear Regression

La prédiction avec les caractéristiques sélectionnées par la méthode de l'information mutuelle pour le modèle de Régression Linéaire maintient une performance exceptionnelle. Les résultats révèlent que le modèle de Régression Linéaire, alimenté par les caractéristiques sélectionnées, conserve une précision exceptionnelle dans la prédiction des prix. Malgré la réduction dimensionnelle basée sur l'information mutuelle, le modèle maintient une adéquation solide aux données, comme en témoigne le coefficient de détermination élevé. En conclusion, la sélection de caractéristiques basée sur l'information mutuelle a préservé des informations cruciales pour la prédiction des prix.

```

cols=['Open', 'High', 'Low']
X_trainM = X_train[cols]
X_testM = X_test[cols]

LinearRegressionModel = LinearRegression()
LinearRegressionModel.fit(X_trainM, y_train)

#Calculating Details
print('Linear Regression Test Score is : ', LinearRegressionModel.score(X_testM, y_test))
print("=====")

#Calculating Prediction
y_pred = LinearRegressionModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Linear Regression Test Score is :  0.9858981799127012
=====
Mean Squared Error : 23.21140476907555
Root Mean Squared Error : 4.817821579207303
Mean Absolute Error : 1.4822305744959317
Coefficient de Détermination : 0.9858981799127012

```

c) Méthode de Features Selection : F-test (fisher)

1. Application de la f_regression

La méthode F-test (ou Fisher) a été appliquée à l'aide de la fonction f_regression pour évaluer la pertinence de chaque caractéristique par rapport à la variable cible. Les résultats de l'analyse F-test fournissent des valeurs indiquant la force de la relation linéaire entre chaque caractéristique et la variable cible.

```

from sklearn.feature_selection import f_regression
f_value = f_regression(X_train, y_train)

# Print the name and F-value of each feature
for feature in zip(feature_names, f_value[0]):
    print(feature)

('Open', 34145.863401148374)
('High', 30949.469877659594)
('Low', 25142.23390019029)
('Vol.', 53.83680890295678)
('Change %', 4.888476517383117)
('coef', 0.8332828910351292)
('state', 1.22748025836949)
('activity_in', 0.024127204431705195)
('activity_out', 0.3297423430326132)

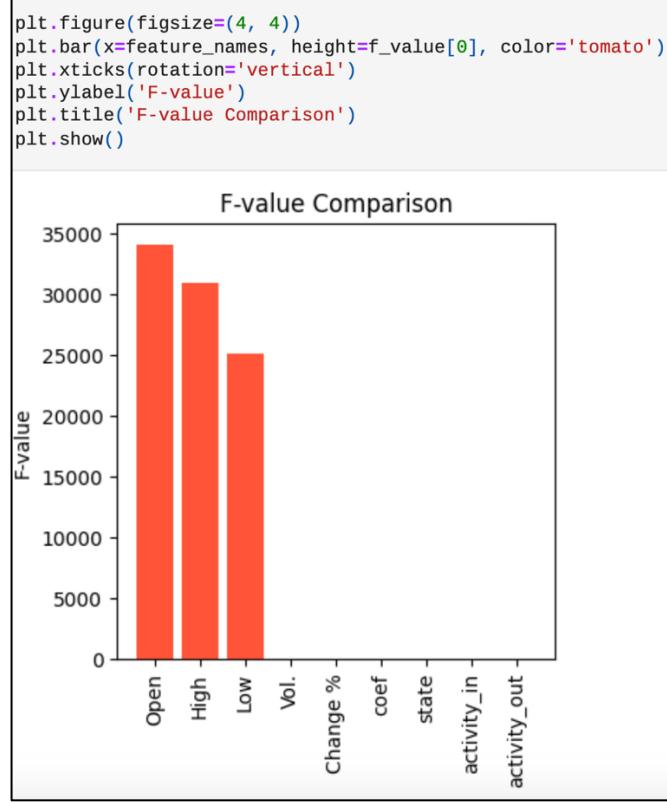
```

2. Visualisation des F-scores

L'application de la méthode F-test (Fisher) à chaque caractéristique fournit des statistiques F, reflétant la force de la relation linéaire entre chaque caractéristique et la variable cible (les prix). Voici une interprétation plus détaillée des résultats :

- 'Open' : La caractéristique 'Open' présente une statistique F élevée de 34145.86, suggérant une forte relation linéaire avec les prix.
- 'High' : De même, la caractéristique 'High' affiche une statistique F notable de 30949.47, indiquant une relation linéaire significative avec les prix.
- 'Low' : La caractéristique 'Low' montre également une statistique F élevée de 25142.23, suggérant une corrélation linéaire forte avec les prix.
- 'Vol.' : La statistique F pour 'Vol.' (Volume) est de 53.84, indiquant une relation linéaire moins forte mais néanmoins significative.
- 'Change %' : La caractéristique 'Change %' présente une statistique F de 4.89, indiquant une relation linéaire modérée avec les prix.
- 'coef' : La caractéristique 'coef' a une statistique F de 0.83, montrant une relation linéaire relativement faible avec les prix.
- 'state' : La statistique F pour 'state' est de 1.23, indiquant une relation linéaire modérée avec les prix.
- 'activity_in' : La caractéristique 'activity_in' a une statistique F de 0.024, suggérant une relation linéaire faible avec les prix.
- 'activity_out' : La statistique F pour 'activity_out' est de 0.33, indiquant une relation linéaire modérée avec les prix.

Ces résultats orientent vers les caractéristiques les plus informatives pour la prédiction des prix, en se basant sur la force de leur relation linéaire avec la variable cible.



3. La prédition Avec les caractéristiques sélectionnées par la méthode F-test (fisher) SVR

La prédition avec les caractéristiques sélectionnées par la méthode F-test (Fisher) pour le modèle SVR maintient une performance solide. Ces résultats suggèrent que le modèle SVR, entraîné avec les caractéristiques sélectionnées par la méthode F-test (Fisher), conserve une précision élevée dans la prédition des prix malgré la réduction dimensionnelle. Bien que les erreurs aient légèrement augmenté par rapport au modèle complet, le coefficient de détermination reste élevé, indiquant une adéquation solide du modèle aux données. En résumé, la sélection de caractéristiques basée sur la méthode F-test a préservé des informations cruciales pour la prédition des prix avec le modèle SVR.

```

cols=['Open', 'High', 'Low']
X_trainM = X_train[cols]
X_testM = X_test[cols]

SVRModel = SVR(C = 1.5 ,epsilon=0.) # it also can be : linear, poly, rbf, sigmoid, precomputed
SVRModel.fit(X_trainM, y_train)
#Calculating Prediction
y_pred = SVRModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Mean Squared Error : 45.2386289335571
Root Mean Squared Error : 6.725966765719044
Mean Absolute Error : 2.242797664193615
Coefficient de Détermination : 0.9725157950342141

```

Linear regression

La prédiction avec les caractéristiques sélectionnées par la méthode F-test (Fisher) pour le modèle de Régression Linéaire maintient une performance exceptionnelle. Ces résultats indiquent que le modèle de Régression Linéaire, alimenté par les caractéristiques sélectionnées par la méthode F-test (Fisher), conserve une précision exceptionnelle dans la prédiction des prix. Malgré la réduction dimensionnelle, le modèle maintient une adéquation solide aux données, comme en témoigne le coefficient de détermination élevé. En conclusion, la sélection de caractéristiques basée sur la méthode F-test a préservé des informations cruciales pour la prédiction des prix avec le modèle de Régression Linéaire.

```
cols=['Open', 'High', 'Low']
X_trainM = X_train[cols]
X_testM = X_test[cols]

LinearRegressionModel = LinearRegression()
LinearRegressionModel.fit(X_trainM, y_train)

#Calculating Details
print('Linear Regression Test Score is : ', LinearRegressionModel.score(X_testM, y_test))
print("=====")

#Calculating Prediction
y_pred = LinearRegressionModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolute Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

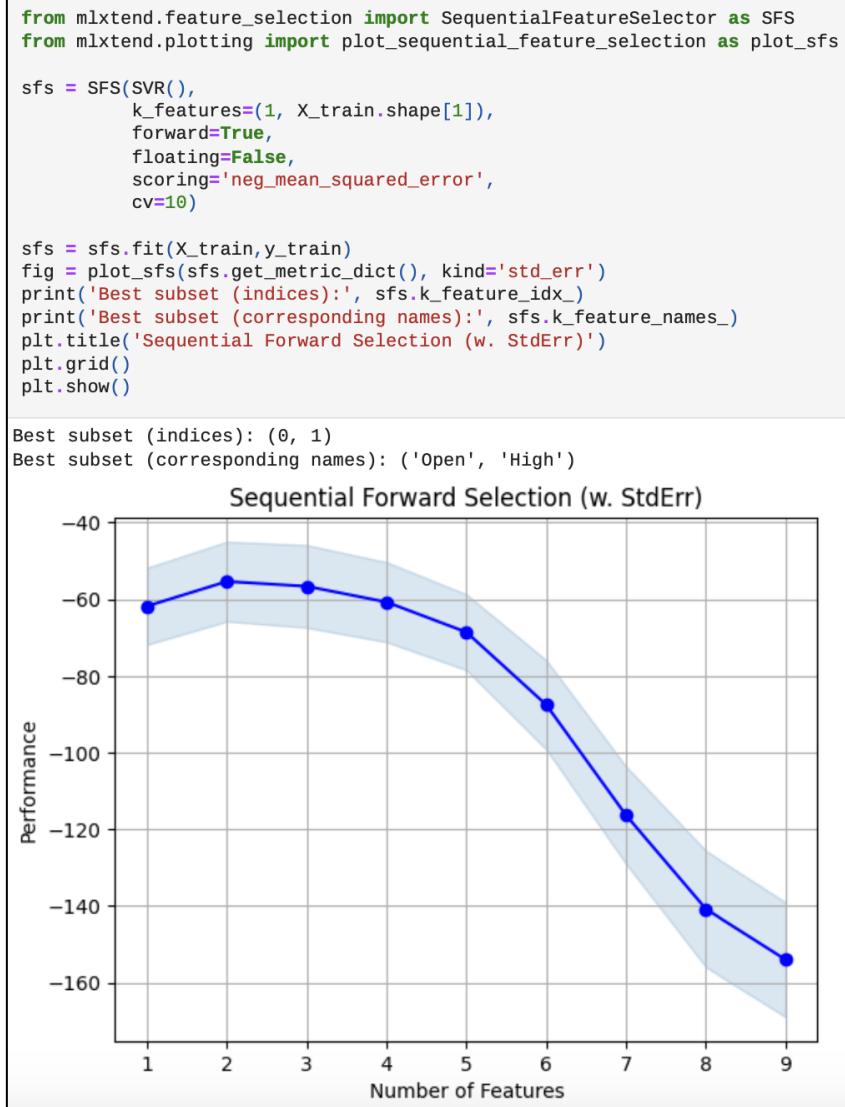
print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Linear Regression Test Score is :  0.9858981799127012
=====
Mean Squared Error : 23.21140476907555
Root Mean Squared Error : 4.817821579207303
Mean Absolute Error : 1.4822305744959317
Coefficient de Détermination : 0.9858981799127012
```

d) Méthode de Features Selection : Forward/Sequential Features selection SFS

1. Application de l'algorithme SFS Forward

L'algorithme SFS Forward a été appliqué pour sélectionner les meilleures caractéristiques pour le modèle, révélant que le meilleur sous-ensemble de caractéristiques est composé des indices (0, 1), correspondant aux caractéristiques ('Open', 'High'). Le graphique de performance du modèle, évalué en fonction du nombre de caractéristiques sélectionnées, indique que la performance maximale est atteinte avec deux caractéristiques, à savoir 'Open' et 'High'. Ces deux caractéristiques ont été identifiées comme les plus informatives pour le modèle, conduisant à une optimisation de la performance du modèle de prédiction des prix.



2. La prédiction Avec les caractéristiques sélectionnées par la méthode Forward SFS SVR

Après l'application de l'algorithme SFS Forward pour sélectionner les caractéristiques optimales ('Open' et 'High'), le modèle SVR a été évalué. Ces résultats soulignent que, même après la réduction dimensionnelle avec seulement deux caractéristiques, le modèle SVR maintient une performance notable dans la prédiction des prix. Bien que les erreurs aient légèrement augmenté, le coefficient de détermination reste élevé, indiquant une adéquation robuste du modèle aux données. En résumé, la méthode Forward SFS a réussi à sélectionner un sous-ensemble de caractéristiques optimal pour le modèle SVR, conservant une précision élevée dans la prédiction des prix.

```

cols = ['Open', 'High']
X_trainM = X_train[cols]
X_testM = X_test[cols]

SVRModel = SVR(C = 1.5 ,epsilon=0.) # it also can be : linear, poly, rbf, sigmoid, precomputed
SVRModel.fit(X_trainM, y_train)
#Calculating Prediction
y_pred = SVRModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Mean Squared Error : 52.33869885166404
Root Mean Squared Error : 7.234548973617087
Mean Absolute Error : 2.1575087464569878
Coefficient de Détermination : 0.9682022298024459

```

Linear Regression

Suite à l'application de l'algorithme SFS Forward pour sélectionner les caractéristiques optimales ('Open' et 'High'), le modèle de Régression Linéaire a été évalué. Les résultats mettent en évidence la performance élevée du modèle de Régression Linéaire, même après la réduction dimensionnelle par la méthode Forward SFS. Le coefficient de détermination élevé indique une excellente adéquation du modèle aux données, démontrant la pertinence de la sélection de caractéristiques pour maintenir la précision de la prédiction des prix. En conclusion, la méthode Forward SFS a réussi à identifier un sous-ensemble optimal de caractéristiques, préservant la qualité du modèle de Régression Linéaire dans sa tâche de prédiction.

```

LinearRegressionModel = LinearRegression()
LinearRegressionModel.fit(X_trainM, y_train)

#Calculating Details
print('Linear Regression Test Score is : ' , LinearRegressionModel.score(X_testM, y_test))
print("=====")

#Calculating Prediction
y_pred = LinearRegressionModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Linear Regression Test Score is :  0.9805630252830002
=====
Mean Squared Error : 31.992996992559902
Root Mean Squared Error : 5.65623523136723
Mean Absolute Error : 1.5271562921644193
Coefficient de Détermination : 0.9805630252830002

```

e) Méthode de Features Selection : Backward/Sequential Features selection SFS

1. Application de l'algorithme SFS Backward

L'algorithme SFS Backward a été utilisé pour sélectionner les caractéristiques optimales. Le graphique de performance du modèle, évalué en fonction du nombre de caractéristiques sélectionnées, indique que le meilleur sous-ensemble de caractéristiques est obtenu avec seulement deux caractéristiques, à savoir 'Open' et 'High'. La performance du modèle atteint son apogée avec ces deux caractéristiques, comme indiqué par le point le plus élevé sur le graphique.

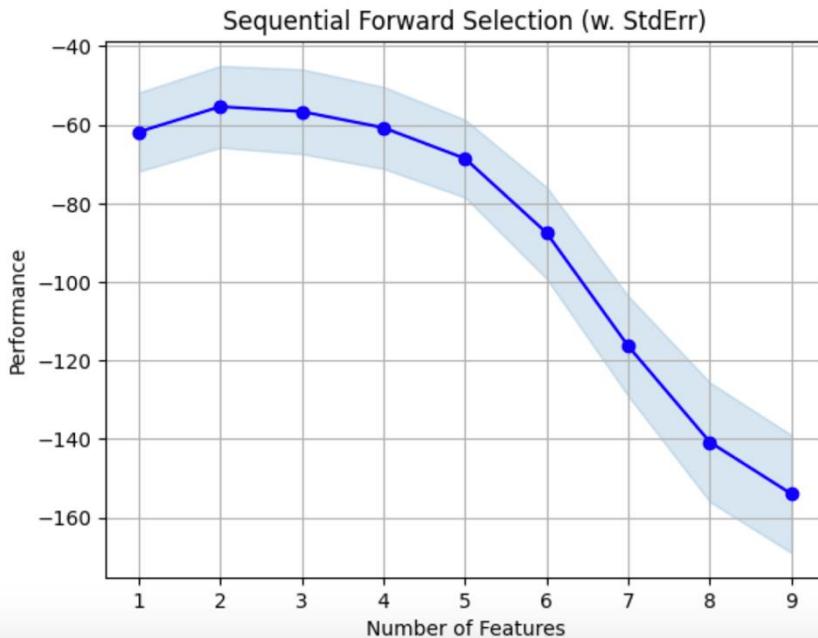
À mesure que d'autres caractéristiques sont ajoutées (indiquées sur l'axe des x), la performance du modèle diminue, suggérant que ces caractéristiques supplémentaires ne contribuent pas à l'amélioration de la précision du modèle.

En résumé, l'algorithme SFS Backward identifie 'Open' et 'High' comme les caractéristiques les plus informatives, optimisant ainsi la performance du modèle en minimisant la dimensionnalité du jeu de données

```
sfs = SFS(SVR(),
           k_features=(1, X_train.shape[1]),
           forward=False,
           floating=False,
           scoring='neg_mean_squared_error',
           cv=10)

sfs = sfs.fit(X_train,y_train)
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err')
print('Best subset (indices):', sfs.k_feature_idx_)
print('Best subset (corresponding names):', sfs.k_feature_names_)
plt.title('Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()

Best subset (indices): (0, 1)
Best subset (corresponding names): ('Open', 'High')
```



2. La prédiction Avec les caractéristiques sélectionnées par la méthode Backward SFS SVR

Suite à l'application de l'algorithme SFS Backward pour sélectionner les caractéristiques optimales ('Open' et 'High'), le modèle SVR a été évalué et les résultats témoignent de la performance du modèle SVR en utilisant exclusivement les caractéristiques identifiées par la méthode Backward SFS. Bien que les erreurs aient légèrement augmenté par rapport au modèle complet, le coefficient de détermination reste élevé, soulignant la robustesse du modèle dans sa capacité à prédire les prix, même après la réduction dimensionnelle. En somme, la méthode Backward SFS a réussi à sélectionner un sous-ensemble optimal de caractéristiques, préservant la qualité du modèle SVR dans sa tâche de prédiction.

```
cols = ['Open', 'High']
X_trainM = X_train[cols]
X_testM = X_test[cols]

SVRModel = SVR(C = 1.5 ,epsilon=0.) # it also can be : linear, poly, rbf, sigmoid, precomputed
SVRModel.fit(X_trainM, y_train)
#Calculating Prediction
y_pred = SVRModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R2)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Mean Squared Error : 52.33869885166404
Root Mean Squared Error : 7.234548973617087
Mean Absolute Error : 2.1575087464569878
Coefficient de Détermination : 0.9682022298024459
```

Linear regression

Après la sélection des caractéristiques par l'algorithme SFS Backward, le modèle de Régression Linéaire a été évalué. Les résultats démontrent que, même avec seulement deux caractéristiques sélectionnées, le modèle de Régression Linéaire conserve une performance élevée dans la prédiction des prix. Le coefficient de détermination élevé indique une adéquation solide du modèle aux données, même après la réduction dimensionnelle. En résumé, la méthode Backward SFS a réussi à sélectionner un sous-ensemble optimal de caractéristiques, maintenant une précision notable dans la prédiction des prix avec le modèle de Régression Linéaire.

```

LinearRegressionModel = LinearRegression()
LinearRegressionModel.fit(X_trainM, y_train)

#Calculating Details
print('Linear Regression Test Score is : ' , LinearRegressionModel.score(X_testM, y_test))
print("=====")

#Calculating Prediction
y_pred = LinearRegressionModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Linear Regression Test Score is : 0.9805630252830002
=====
Mean Squared Error : 31.992996992559902
Root Mean Squared Error : 5.65623523136723
Mean Absolute Error : 1.5271562921644193
Coefficient de Détermination : 0.9805630252830002

```

f) Méthode de Features Selection : Random forest

1. Application de l'algorithme Random forest

La méthode de sélection des caractéristiques basée sur l'algorithme Random Forest a été mise en œuvre. Cet algorithme attribue des scores d'importance à chaque caractéristique en évaluant leur contribution à la précision du modèle. Les résultats de cette évaluation fourniront des indications sur les caractéristiques les plus informatives pour la prédiction des prix.

```

from sklearn.ensemble import RandomForestRegressor

# Create a random forest classifier
rfc = RandomForestRegressor(random_state=0) # Use gini criterion to define feature importance

# Train the classifier
rfc.fit(X_train, y_train)

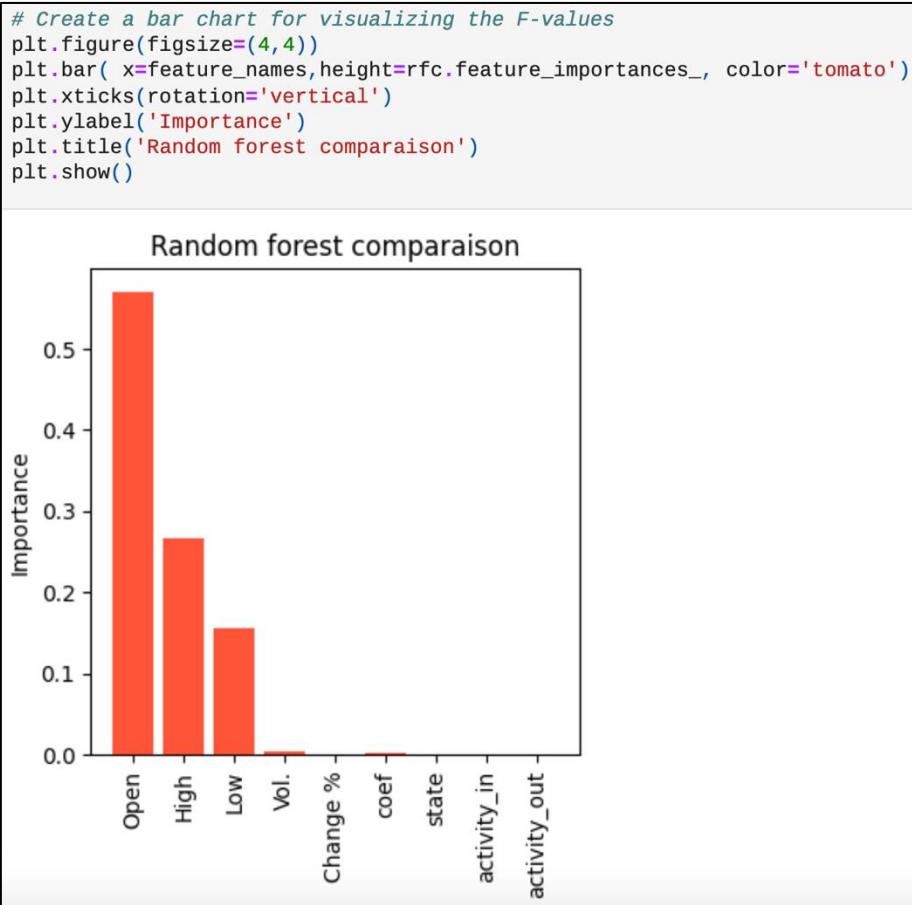
# Print the name and gini importance of each feature
for feature in zip(feature_names, rfc.feature_importances_):
    print(feature)

('Open', 0.5699719476312699)
('High', 0.26666335853456224)
('Low', 0.15582844975282145)
('Vol.', 0.0032944254892845475)
('Change %', 0.0009930713148959172)
('coef', 0.0016488874095432387)
('state', 0.00031058260503420677)
('activity_in', 0.000842243288505516)
('activity_out', 0.00044703397408297303)

```

2. Visualisation des scores

L'algorithme Random Forest a été utilisé pour effectuer la sélection de caractéristiques. Les valeurs obtenus représentent l'importance relative de chaque caractéristique dans la prédiction du modèle. Plus la valeur est élevée, plus la caractéristique est importante. Dans ce cas, 'Open' et 'High' semblent être les caractéristiques les plus informatives pour le modèle, suivies de 'Low'. Les caractéristiques 'Vol.', 'Change %', 'coef', 'state', 'activity_in' et 'activity_out' ont des importances relativement faibles dans la prédiction du modèle.



3. La prédiction Avec les caractéristiques sélectionnées par la méthode Random Forest

SVR

Suite à la sélection minutieuse des caractéristiques par l'algorithme Random Forest, le modèle SVR a été évalué avec les caractéristiques choisies. Les métriques démontrent la performance du modèle SVR en utilisant exclusivement les caractéristiques identifiées comme les plus informatives par la méthode Random Forest. Bien que les erreurs aient légèrement augmenté par rapport au modèle complet, le coefficient de détermination reste élevé, confirmant la solidité du modèle dans sa capacité à prédire les prix, même après la réduction dimensionnelle. En résumé, la méthode Random Forest a réussi à distiller un sous-ensemble optimal de caractéristiques, préservant l'efficacité du modèle SVR dans sa tâche de prédiction.

```

cols=['Open', 'High', 'Low']
X_trainM = X_train[cols]
X_testM = X_test[cols]

SVRModel = SVR(C = 1.5 ,epsilon=0.) # it also can be : linear, poly, rbf, sigmoid, precomputed
SVRModel.fit(X_trainM, y_train)
#Calculating Prediction
y_pred = SVRModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Mean Squared Error : 45.2386289335571
Root Mean Squared Error : 6.725966765719044
Mean Absolute Error : 2.242797664193615
Coefficient de Détermination : 0.9725157950342141

```

Linear Regression

Après la sélection des caractéristiques par l'algorithme Random Forest, le modèle de Régression Linéaire a été évalué. Les résultats témoignent de la performance du modèle de Régression Linéaire en utilisant uniquement les caractéristiques identifiées comme les plus importantes par la méthode Random Forest. Bien que les erreurs aient légèrement augmenté par rapport au modèle complet, le coefficient de détermination demeure élevé, soulignant la capacité robuste du modèle dans la prédiction des prix, même après la réduction dimensionnelle. En résumé, la méthode Random Forest a réussi à sélectionner un sous-ensemble optimal de caractéristiques, maintenant une précision notable dans la prédiction des prix avec le modèle de Régression Linéaire.

```

LinearRegressionModel = LinearRegression()
LinearRegressionModel.fit(X_trainM, y_train)

#Calculating Details
print('Linear Regression Test Score is : ', LinearRegressionModel.score(X_testM, y_test))
print("=====")

#Calculating Prediction
y_pred = LinearRegressionModel.predict(X_testM)

# Erreur Quadratique Moyenne (MSE - Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
# Racine de l'Erreur Quadratique Moyenne (RMSE - Root Mean Squared Error)
rmse = np.sqrt(mse)
# Erreur Absolue Moyenne (MAE - Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
# Coefficient de Détermination (R²)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error :", mse)
print("Root Mean Squared Error :", rmse)
print("Mean Absolute Error :", mae)
print("Coefficient de Détermination :", r2)

Linear Regression Test Score is :  0.9858981799127012
=====
Mean Squared Error : 23.21140476907555
Root Mean Squared Error : 4.817821579207303
Mean Absolute Error : 1.4822305744959317
Coefficient de Détermination : 0.9858981799127012

```

g) La performance des algorithmes de prédiction après la sélection des caractéristiques

Dans cette partie, nous allons comparer l'efficacité de plusieurs modèles de régression (régression linéaire, Ridge, arbre de décision, forêt aléatoire, boosting et SVR) sur un ensemble de données. Les données sont divisées en ensembles d'entraînement et de test, avec une analyse portant sur les colonnes spécifiques 'Open', 'High', 'Low'. Chaque modèle est formé sur l'ensemble d'entraînement restreint à ces colonnes, puis utilisé pour prédire les valeurs sur l'ensemble de test. Les résultats sont visualisés à travers des graphiques montrant la comparaison entre les valeurs réelles et les prédictions, avec l'évaluation de la qualité de chaque modèle basé sur le Mean Squared Error (MSE) affiché sur chaque graphique. Cette approche permet de déterminer visuellement et quantitativement la performance relative des modèles pour guider le choix du modèle optimal dans des applications futures.

```

from sklearn.svm import SVR
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

cols=['Open', 'High', 'Low'] #*** Critère de l'information mutuelle      fisher

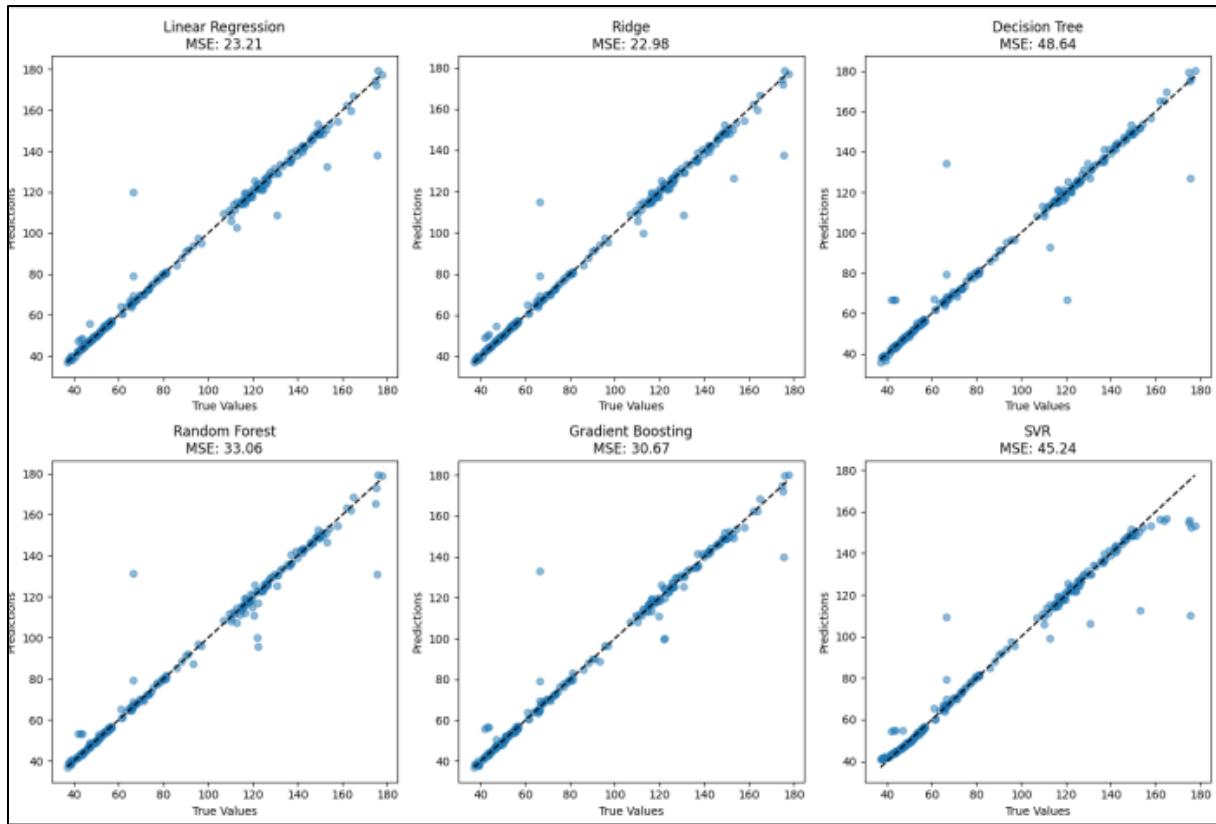
X_trainL = X_train[cols]
X_testL = X_test[cols]

# Initialiser les modèles
models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(),
    "Gradient Boosting": GradientBoostingRegressor(),
    "SVR": SVR(C = 1.5 ,epsilon=0.)
}

# Entrainer les modèles et faire des prédictions
predictions = {}
for name, model in models.items():
    model.fit(X_trainL, y_train)
    predictions[name] = model.predict(X_testL)

# Visualisation
plt.figure(figsize=(15, 10))
for i, (name, pred) in enumerate(predictions.items(), 1):
    plt.subplot(2, 3, i)
    plt.scatter(y_test, pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--k')
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.title(f'{name}\nMSE: {mean_squared_error(y_test, pred):.2f}')
plt.tight_layout()
plt.show()

```



Les résultats obtenus fournissent des indications utiles pour choisir le modèle le plus adapté à la tâche de régression. Dans cette analyse, le modèle Ridge émerge comme le plus performant.

3.3 Tableaux de comparaison

3.3.1 Iris Dataset (classification)

Dans le cadre de notre étude, nous avons entrepris une analyse approfondie des performances de la Régression Logistique en termes de précision, en considérant différentes approches de sélection de caractéristiques. Les résultats obtenus sont synthétisés dans le tableau ci-dessous, où la précision de la Régression Logistique est évaluée à différentes étapes du processus de sélection de caractéristiques.

Sélection de caractéristiques	Précision (Régression Logistique)
Avant sélection de caractéristiques	0.9111
Après sélection de caractéristiques : Variance	0.9556
Après sélection de caractéristiques : Critère d'information mutuelle	0.9556
Après sélection de caractéristiques : Test de Fisher	1.0
Après sélection de caractéristiques : Sélection séquentielle progressive (SFS)	1.0
Après sélection de caractéristiques : Sélection séquentielle régressive (SBS)	1.0

Ces résultats mettent en lumière l'impact significatif de différentes méthodes de sélection de caractéristiques sur les performances du modèle. On constate que la comparaison des

performances de la Régression Logistique à différentes étapes de la sélection de caractéristiques révèle une amélioration notable de la précision après l'application de méthodes spécifiques telles que la sélection séquentielle progressive (SFS) et le test de Fisher.

3.3.2 Stock Data (Regression)

L'efficacité des modèles de régression dépend étroitement du choix des caractéristiques incluses dans l'analyse. Dans cette étude, nous avons examiné les performances de deux modèles de régression, le Support Vector Regression (SVR) et la Régression Linéaire, en utilisant différentes méthodes de sélection de caractéristiques. Les mesures d'erreur, notamment le Mean Squared Error (MSE), le Root Mean Squared Error (RMSE) et le Mean Absolute Error (MAE), ont été évaluées avant et après l'application de chaque méthode de sélection. Le tableau ci-dessous résume ces résultats, mettant en lumière l'impact de la sélection de caractéristiques sur la précision des modèles.

Sélection de caractéristiques	Modèle	MSE	RMSE	MAE
Avant la sélection	SVR	132.4586	11.5091	6.8919
	Régression Linéaire	22.4558	4.7388	1.3766
Variance	SVR	135.8079	11.6537	6.8818
	Régression Linéaire	23.5056	4.8483	1.5583
Critère d'information mutuelle	SVR	45.2386	6.7260	2.2428
	Régression Linéaire	23.2114	4.8178	1.4822
F-test (Fisher)	SVR	45.2386	6.7260	2.2428
	Régression Linéaire	23.2114	4.8178	1.4822
Sélection séquentielle progressive (SFS)	SVR	52.3387	7.2345	2.1575
	Régression Linéaire	31.9930	5.6562	1.5272
Sélection séquentielle régressive (SBS)	SVR	52.3387	7.2345	2.1575
	Régression Linéaire	31.9930	5.6562	1.5272
Random Forest	SVR	45.2386	6.7260	2.2428
	Régression Linéaire	23.2114	4.8178	1.4822

Les résultats de cette analyse démontrent que la sélection de caractéristiques exerce une influence significative sur les performances des modèles de régression. Avant toute sélection, le SVR présente une précision relativement élevée par rapport à la Régression Linéaire. Cependant, l'application de méthodes spécifiques, telles que la sélection basée sur la variance ou le critère d'information mutuelle, peut entraîner des variations substantielles dans les mesures d'erreur.

4 - Conclusion :

Les résultats obtenus soulignent la capacité robuste de la feature selection à réduire la dimensionnalité tout en préservant des performances prédictives fiables, aussi bien dans des contextes simples, comme l'Iris dataset, que dans des situations plus complexes telles que le Stock Dataset. Ces observations mettent en lumière le potentiel de cette méthode dans la simplification de modèles, confirmant son utilité dans des domaines variés.

L'exploration approfondie de la feature selection a également mis en évidence son rôle crucial dans la compréhension des caractéristiques significatives, ouvrant ainsi des perspectives prometteuses pour des applications élargies, notamment dans les domaines de la médecine et de la biologie.

En somme, ce rapport approfondi enrichit considérablement notre compréhension des méthodes de réduction de dimensionnalité, positionnant la feature selection comme un outil précieux pour simplifier l'analyse de données complexes.