Rapport Projet Data

Anticiper les évolutions hebdomadaires d'actions via des méthodes de Machine Learning

M2 ISD

N° étudiant 22214059

Année 2023 -2024

Introduction:

Ce projet a pour objectif d'anticiper les évolutions hebdomadaires des actions en utilisant des méthodes de Machine Learning.
À l'aide de données boursières journalières couvrant la période de 2000 à 2023, notre démarche consiste à prédire le changement des cours sur les cinq prochains jours, en classifiant les mouvements en baisse, stabilité ou hausse. Cette tâche complexe nécessite l'exploration de diverses approches, allant des modèles statistiques aux réseaux de neurones. Ce rapport détaillera la méthodologie adoptée, en mettant en lumière l'utilisation d'indicateurs, pour finalement évaluer et recommander des modèles pertinents dans ce contexte financier dynamique.

Données

Le jeu de données "Historiques_cours_boursiers.xlsx" contient les historiques boursiers journaliers de 13 cours boursiers majeurs, s'étalant sur diverses périodes de 2000 à 2023.

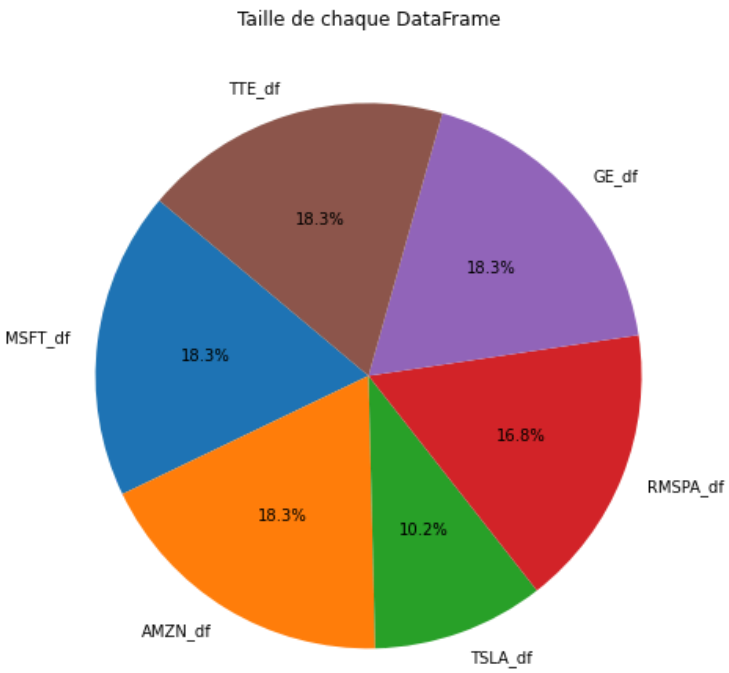
j'ai choisi 6 cours spécifiques parmi eux : MSFT, AMZN, TSLA, RMS.PA, GE, et TTE.

Ticker	Intitulé
MSFT	Microsoft
AMZN	Amazon
TSLA	Tesla
RMS.PA	Hermès International
GE	General Electric
TTE	TotalEnergies

Chacune de ces sélections comprend les colonnes suivantes :

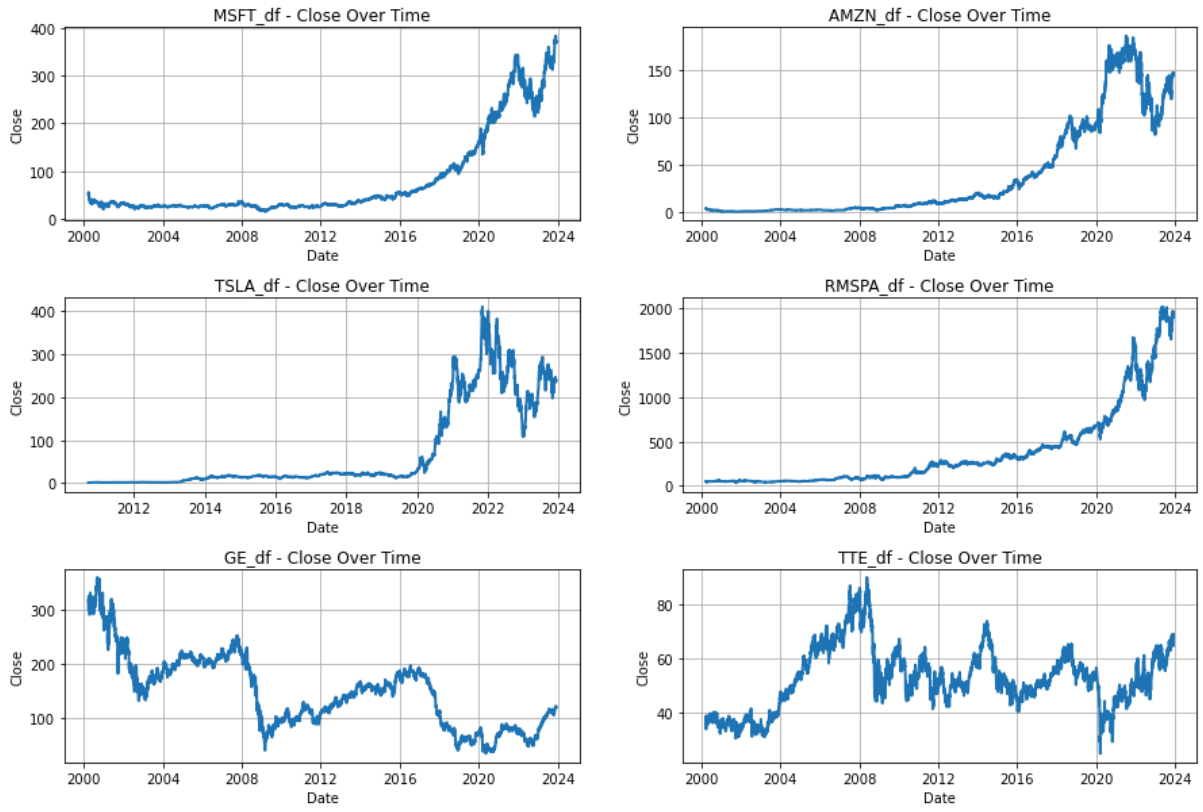
- Open : prix de l’action à l’ouverture ;
- High : prix le plus élevé du jour de cotation (H) ;
- Low : prix le plus bas sur la journée (L) ;
- Close : prix de clôture de l’action en fin de séance (C) ;
- Volume : nombre d’actions achetées et vendues d’une action au cours de la journée;
- Adjusted : mesure du profit de l’investissement par rapport au risque d’investissement sur une période donnée.

1. Vérifiez si les DataFrames ont approximativement le même nombre de lignes. (Pour savoir comment entraîner nos modèles)



Microsoft, Amazon, General Electric et Hermès présentent un nombre équivalent de lignes, soit 5967, tandis que Tesla comporte 3330 lignes.

2. évolution des valeurs de clôture (Close) au fil des années pour chaque groupe :



L'évolution des valeurs de clôture présente des disparités significatives d'un groupe à l'autre. Certains, tels que Microsoft, Amazon et Hermès, initient leur évolution au cours de la période de 2000 à 2016. En revanche, Tesla commence sa série chronologique en 2010.Chacun affichant des niveaux initiaux différents.

En détail, Microsoft et Tesla débutent avec des cours boursiers situés entre 0 et 100, tandis qu'Amazon débute dans une fourchette de 0 à 50. De manière notable, le titre Hermès affiche une amplitude plus importante, débutant avec des valeurs comprises entre 0 et 1000. En revanche, General Electric entame sa série chronologique avec un cours initial d'environ 300, mais connaît une tendance à la baisse. Enfin, Total Energies se caractérise par des valeurs de clôture débutant entre 40 et 60.

L'analyse graphique met en évidence que l'évolution des cours de clôture est propre à chaque groupe. Chaque titre exhibe des tendances et des comportements uniques au fil du temps. Cela nous dit qu'une simple agrégation homogène des données n'est pas envisageable. Bien que la consolidation des données puisse être bénéfique pour enrichir nos modèles avec un volume de données plus important, la singularité de chaque groupe nous contraint à aborder ces ensembles de données de manière distincte dans nos analyses et nos modèles.

1. Définition de la var explicatique

J'ai ensuite procédé au calcul de la variable à expliquer, que j'ai appelé 'Trend', pour chaque groupe, que nous chercherons par la suite à prédire.

Pour la définition de 'Trend', j'ai opté pour l'utilisation des labels suivants :

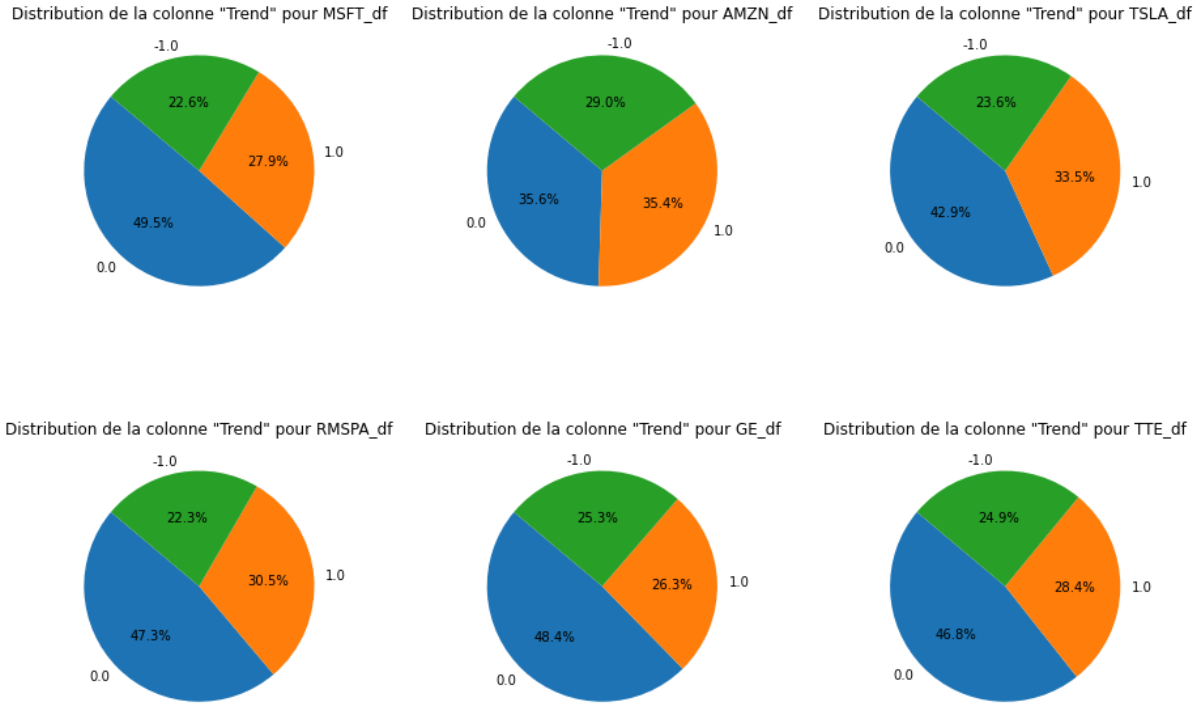
- Label -1 pour représenter une tendance à la baisse,
- Label 0 pour indiquer une stabilité,
- Label 1 pour signifier une tendance à la hausse.

Pour illustrer, voici un exemple sur l'indice Microsoft :

```
list_choice['MSFT_df'].head(15)
```

	Date	Open	High	Low	Close	Adj Close	Volume	Day	Month	Year	Trend
0	2000-01-03	58.687500	59.31250	56.00000	58.28125	36.132240	53228400	3	1	2000	-1.0
1	2000-01-04	56.781250	58.56250	56.12500	56.31250	34.911697	54119000	4	1	2000	-1.0
2	2000-01-05	55.562500	58.18750	54.68750	56.90625	35.279823	64059600	5	1	2000	-1.0
3	2000-01-06	56.093750	56.93750	54.18750	55.00000	34.098007	54976600	6	1	2000	0.0
4	2000-01-07	54.312500	56.12500	53.65625	55.71875	34.543606	62013600	7	1	2000	0.0
5	2000-01-10	56.718750	56.84375	55.68750	56.12500	34.795467	44963600	10	1	2000	1.0
6	2000-01-11	55.750000	57.12500	54.34375	54.68750	33.904270	46743600	11	1	2000	-1.0

Par la suite, j'ai visualisé la distribution des labels pour chaque groupe.



Ces graphiques illustrent la répartition des labels au sein de chaque groupe. Nous observons que le label 0, qui représente la stabilité, est prédominant, constituant la part majoritaire avec un pourcentage oscillant entre 35% et 49.5%.

Pour le label -1, la distribution varie entre 22% et 29%, tandis que le label 1 affiche une plage de pourcentage allant de 26.3% à 35%.

Cette différence notable dans la distribution des labels au sein des groupes pourrait potentiellement influencer nos modèles.

2. Définition des variables explicatives

J'ai généré un ensemble de variables explicatives à partir des données boursières en séries temporelles, visant à capturer divers aspects des tendances du marché.

1. **Indicateur MACD** : Moving Average Convergence Divergence

   Cette mesure permet d'identifier les changements dans la dynamique des prix. En complément, j'ai introduit la variable `MACD_trend`, qui catégorise la tendance haussière, baissière ou stable, selon les règles spécifiées dans l'énoncé

   (Hausse lorsque la ligne ou l'histogramme MACD est positive
   Baisse si lorsque la ligne ou l'histogramme MACD est négative)

2. **RSI (Relative Strength Index) :**
   Il a été calculé pour deux périodes différentes (14 jours et 21 jours), offrant des indications sur la force des mouvements à la hausse ou à la baisse. Les tendances RSI correspondantes, notées `RSI_trend_14` et `RSI_trend_21`, ont été introduites pour catégoriser ces forces en termes de labels -1, 0 ou 1.

3. **L'indicateur ATR (Average True Range)** :
   a été calculé pour mesurer la volatilité du marché. En parallèle, les variables CMF (Chaikin Money Flow) ont été dérivées avec des fenêtres de 21 et 28 jours, fournissant des informations sur les flux monétaires et la pression d'achat ou de vente. Les colonnes `cmf_trend_21` et `cmf_trend_28` catégorisent ces flux en tendances haussières, baissières ou stables.

4. **Daily_Range** représentant l'amplitude quotidienne des prix

5. **Gap** indiquant le gap entre l'ouverture et la clôture par rapport à la journée précédente

Méthodologie

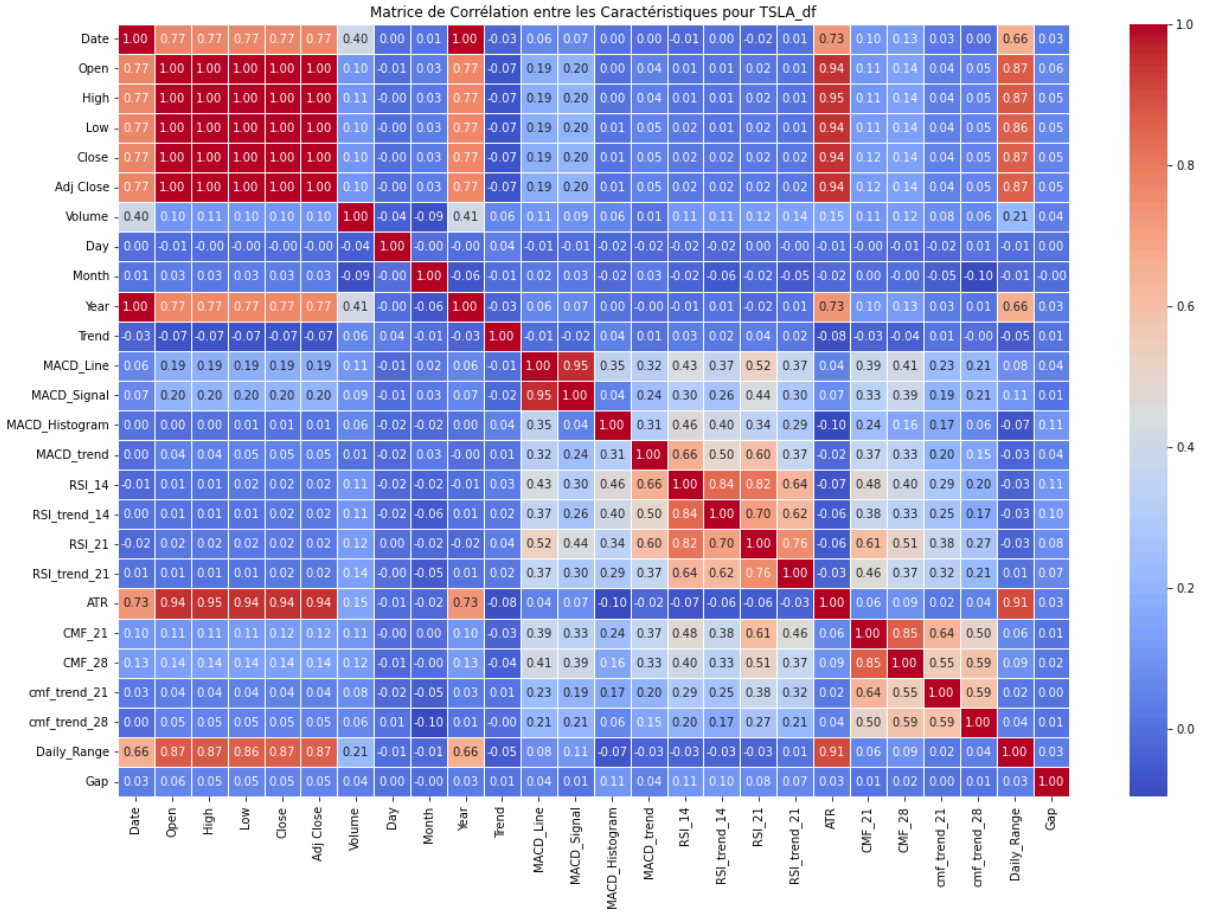
1. Choix des var explicatives (Features)

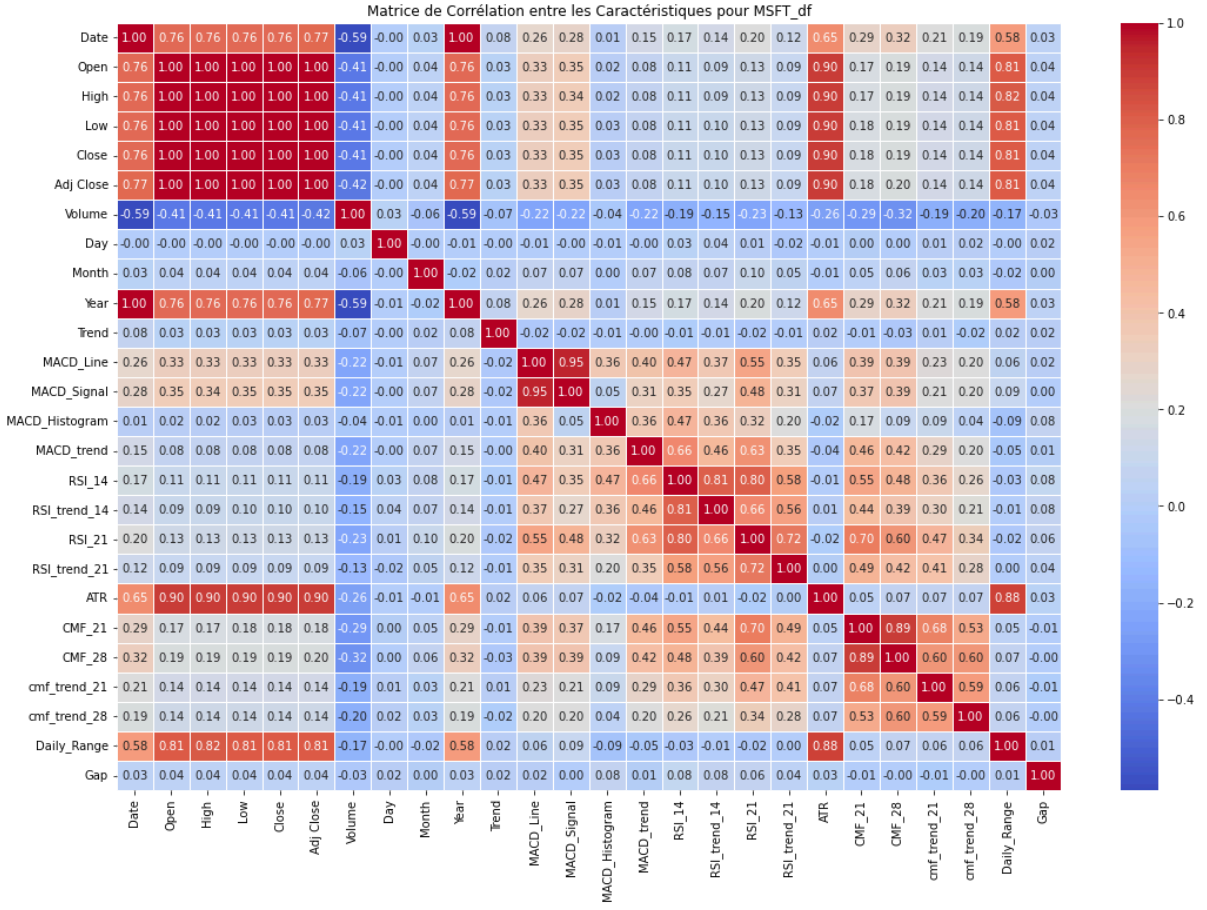
La sélection des variables dans le cadre de cette analyse s'est déroulée en plusieurs étapes, avec comme objectif de choisir les variables les plus pertinentes pour prédire la tendance des cours boursiers.

a. Observation des corrélations :

La première étape a consisté à examiner la matrice de corrélation pour chaque groupe de données.

Pour illustrer cette observation, je présente ci-dessous deux exemples de matrices de corrélation, pour MICROSOFT et TESLA


Matrice de Corrélation entre les Caractéristiques pour TSLA_df
Date Open High Low Close Adj Close Volume Day Month Year Trend MACD_Line MACD_Signal MACD_Histogram MACD_trend RSI_14 RSI_trend_14 RSI_21 RSI_trend_21 ATR CMF_21 CMF_28 cmf_trend_21 cmf_trend_28 Daily_Range Gap
1.0
0.8
0.6
0.4
0.2
0.0


Matrice de Corrélation entre les Caractéristiques pour MSFT_df
Date Open High Low Close Adj Close Volume Day Month Year Trend MACD_Line MACD_Signal MACD_Histogram MACD_trend RSI_14 RSI_trend_14 RSI_21 RSI_trend_21 ATR CMF_21 CMF_28 cmf_trend_21 cmf_trend_28 Daily_Range Gap
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

On peut voir une corrélation parfaite (100%) entre les variables High, Open, Low, Close et Adjusted Close. J'ai opté pour conserver uniquement la variable Close, étant donné qu'elle est spécifiquement utilisée dans le calcul de la tendance.

- **Analyse de l'indicateur MACD :**

  La variable MACD_Line a été retenue, et MACD_Signal a été éliminée en raison de la corrélation presque parfaite entre les deux, évitant ainsi la redondance d'informations.

- **Choix entre CMF_21 et CMF_28 :**
  Les variables CMF_21 et CMF_28 présentaient une corrélation. Conformément à la possibilité mentionnée dans l'énoncé de choisir l'une des deux, CMF_28 a été sélectionnée pour éliminer la redondance d'informations et simplifier le modèle.

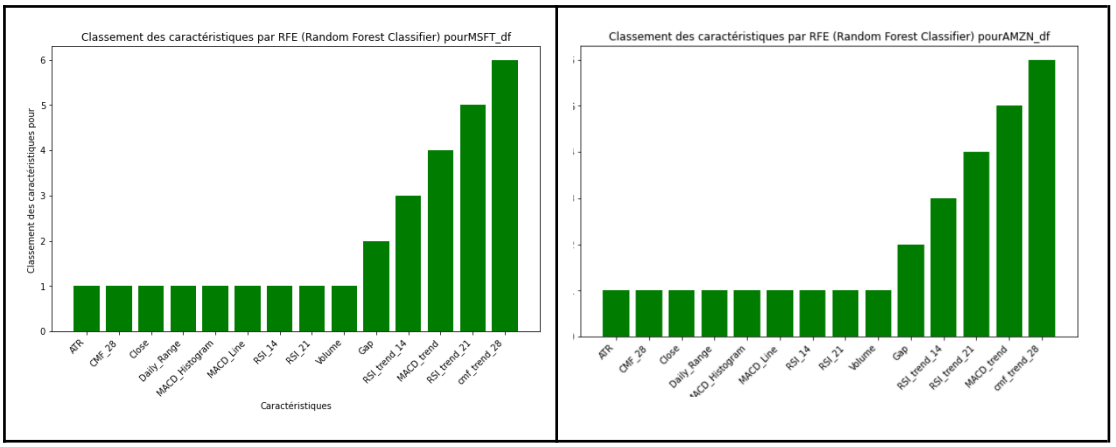
- **Évaluation séparée de RSI_14 et RSI_21 :**

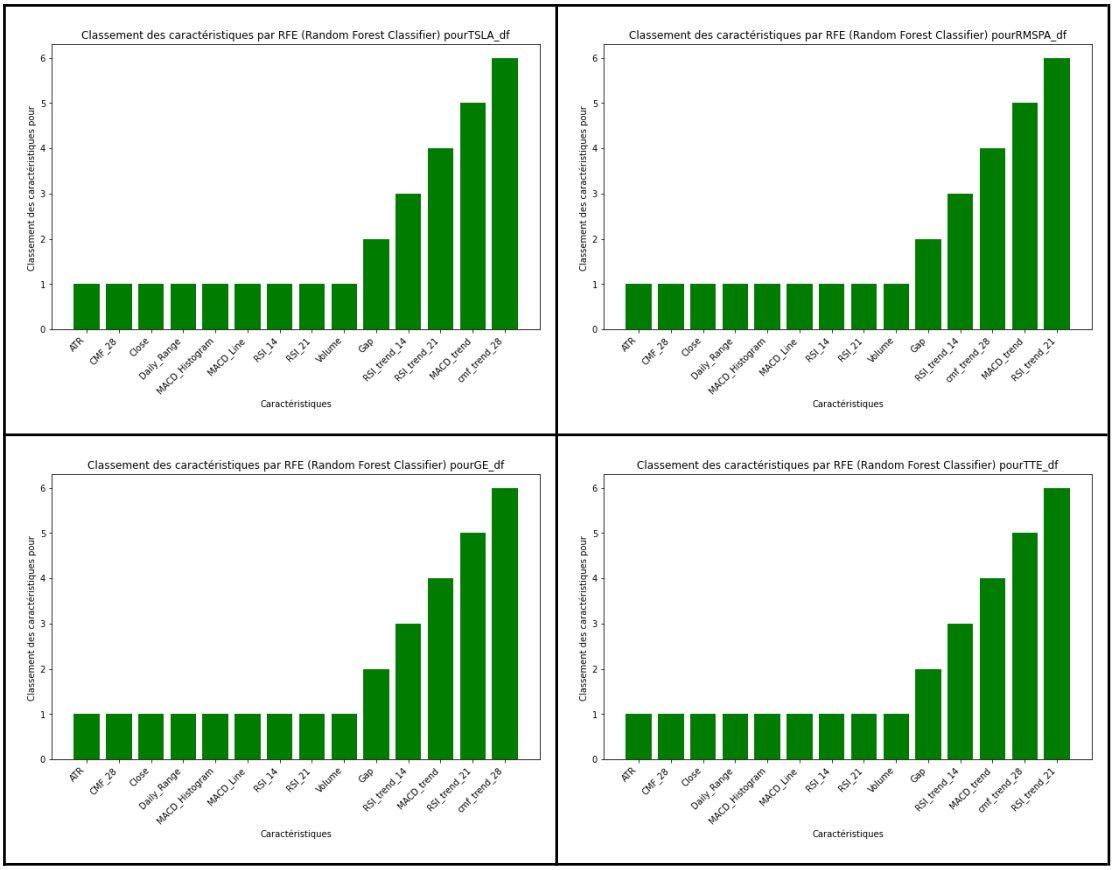
  Bien que les variables RSI_14 et RSI_21 présentaient une corrélation, une évaluation séparée a été entreprise pour déterminer laquelle des deux est plus performante en termes de prédiction.

b. Application de Recursive Feature Elimination (RFE) :

Pour affiner la sélection des variables, J'ai utilisé la méthode RFE avec un classificateur RandomForest. Cette technique attribue des scores aux variables en fonction de leur importance dans la prédiction de la tendance.

Les caractéristiques sélectionnées ont été classées et présentées graphiquement pour chaque groupe spécifique





Tous les graphiques générés à l'aide de cette méthode montrent les mêmes variables avec un scores=1, cela suggère que ces variables sont considérées comme très importantes dans la prédiction de la tendance des cours boursiers.

J’ai décidé alors de retenir les variables :

Avec N=14 pour RSI:

```
selected_features = [ 'ATR','CMF_28', 'Close', 'Daily_Range','MACD_Histogram' , 'MACD_Line' , 'RSI_14', 'Volume']
```

Avec N=21 pour RSI:

```
selected_features = [ 'ATR','CMF_28', 'Close', 'Daily_Range','MACD_Histogram' , 'MACD_Line' , 'RSI_21', 'Volume']
```

Suite à l'évaluation des performances des modèles avec RSI_14 et RSI_21, aucune différence significative n'a été observée entre les deux périodes. Cependant, N=21 améliore légèrement la performance. Par conséquent, je retiens la variable RSI_21 pour les analyses ultérieures.

2. Différentes modélisations de la variable à expliquer

Pour la modélisation, j'ai divisé les données en échantillons d'apprentissage et tests en allouant 80 % des données à l'entraînement et 20 % pour les tests.

J'ai utilisé plusieurs types de modèles, chacun adapté à la nature des données et à l'objectif de classification.

1. **Modèles Classiques :**

   - Régression Logistique: qui est est un modèle statistique permettant d'étudier les relations entre un ensemble de variables.

   - SVM (Support Vector Machine) :

     Ce modèle, dans le contexte de la prédiction des mouvements de marché, cherche à trouver une frontière de décision optimale qui maximise la séparation entre les différentes tendances.C' est un modèle à noyau, j'ai donc utilisé le noyau RBF(Radial Basis Function) pour son efficacité dans la modélisation de relations

   - K plus proches voisins (KNN)

     Les voisins les plus proches servent de référence pour estimer la classe de tendance d'un point donné, en exploitant la continuité et les motifs temporels présents dans les séries.

2. **Modèles Ensemblistes :**
   - Random Forest :
     Ce modèle nécessite la spécification des **hyperparamètres** clés :
     1. n_estimators : le nombre d'arbres dans la forêt
     2. min_samples_leaf :le nombre minimal d'échantillons requis pour constituer une feuille d'arbre
     3. min_samples_split : le nombre minimum d'échantillons requis pour diviser un nœud interne

     4. max_depth : la profondeur maximale des arbres individuels.

Pour choisir les valeurs optimales des hyperparamètres qui maximisent les performances du modèle, j'ai utilisé la technique **GridSearchCV**. Cette méthode de recherche par grille permet d'évaluer systématiquement différentes combinaisons d'hyperparamètres, en testant chaque combinaison à travers une validation croisée.

	max_depth	min_samples_leaf	min_samples_split	n_estimators
Microsoft	10	2	2	200

Amazon	10	1	5	200
Tesla	10	1	5	200
Hermès International	10	1	2	50
General Electric	10	1	2	50
TotalEnergies	10	1	2	200

- Gradient Boosting :
  Ce modèle nécessite la spécification des hyperparamètres :
  1. n_estimators : le nombre total d' arbres qui seront construits dans l'ensemble
  2. max_depth : la profondeur maximale de chaque arbre
  3. learning_rate ;contrôle la contribution de chaque arbre à l'ensemble

Afin de sélectionner les valeurs optimales pour ces hyperparamètres et maximiser les performances du modèle, j'ai également opté pour la technique de **GridSearchCV**.

	n_estimators	min_samples_leaf	max_depth
Microsoft	100	6	0.3
Amazon	100	6	0.3
Tesla	100	6	0.3
Hermès International	100	6	0.3
General Electric	100	6	0.3
TotalEnergies	10	1	2

3. **Réseau de neurone :**
   - à Mémoire à Long Terme (LSTM):

     Ils intègrent une mémoire à long terme qui peut être mise à jour, ajoutée ou supprimée sélectivement, permettant au modèle de retenir des informations importantes sur de longues séquences

- Récurrent (RNN) :
  Ils possèdent une mémoire interne qui leur permet de prendre en compte les informations antérieures lors du traitement de nouvelles données séquentielles. Cela les rend particulièrement adaptés à la modélisation de séquences temporelles, comme les séries chronologiques financières utilisées pour prédire les tendances.

- Convolutif (CNN):
  Ils utilisent des filtres convolutifs pour extraire des motifs et caractéristiques importantes des données en entrée

Résultats

1. Metrics

Afin d'évaluer les performances des modèles, conformément aux directives de l'énoncé, nous utiliserons les métriques suivantes

1. Accuracy :
   Mesure la proportion d'observations correctes parmi l'ensemble des observations.

2. Hamming
   Mesure la fraction des labels incorrects par rapport à l'ensemble des labels

3. Macro-average Recall
   Représente la moyenne des taux de rappel pour chaque classe individuelle, Le rappel mesure la capacité du modèle à identifier toutes les occurrences réelles d'une classe.

4. Macro-average F1 Score
   la moyenne des scores F1 pour chaque classe individuelle. Le score F1 est une mesure qui prend en compte à la fois la précision et le rappel

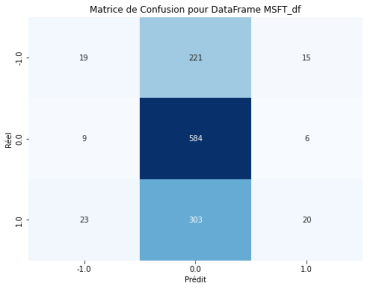
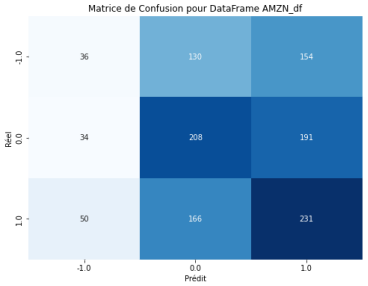
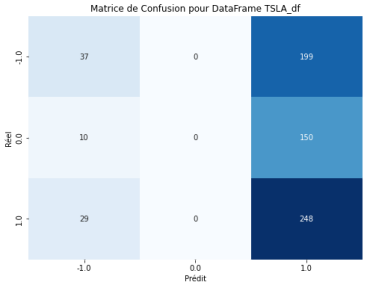
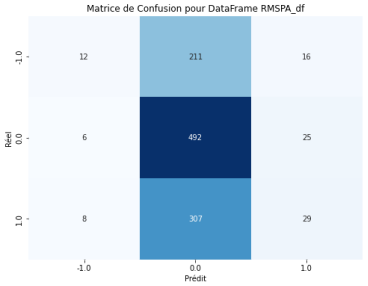
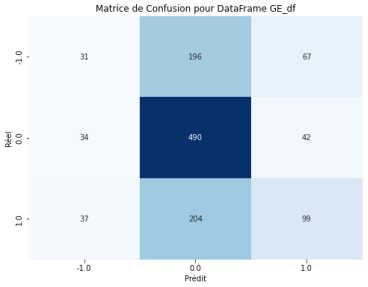
5. Micro-average Recall
   calcule le rappel global sur toutes les classes. Cela donne plus de poids aux classes avec un grand nombre d'observations.

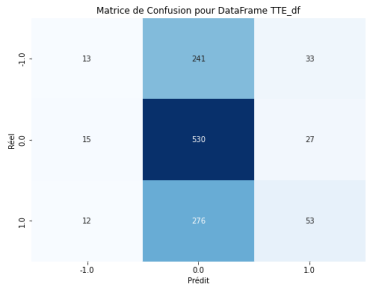
6. Micro-average F1 Score
   calcule le score F1 global sur toutes les classes, également en accordant plus de poids aux classes avec un grand nombre d'observations.

2. Modèles machine learning

Modèles Classiques

- Régression Logistique

	Metrics	Matrice de confusion
Microsoft	********** MSFT_df ********** +----------+---------+ | Metric | Value | +----------+---------+ | Accuracy | 51.92% | | Hamming | 48.08% | | Macro-average Recall | 36.91% | | Macro-average F1 Score | 30.39% | | Micro-average Recall | 51.92% | | Micro-average F1 Score | 51.92% | +----------+---------+	
Amazon	********** AMZN_df ********** +----------+---------+ | Metric | Value | +----------+---------+ | Accuracy | 39.58% | | Hamming | 60.42% | | Macro-average Recall | 36.99% | | Macro-average F1 Score | 35.31% | | Micro-average Recall | 39.58% | | Micro-average F1 Score | 39.58% | +----------+---------+	
Tesla	********** TSLA_df ********** +----------+---------+ | Metric | Value | +----------+---------+ | Accuracy | 42.35% | | Hamming | 57.65% | | Macro-average Recall | 35.07% | | Macro-average F1 Score | 26.82% | | Micro-average Recall | 42.35% | | Micro-average F1 Score | 42.35% | +----------+---------+	
Hermès International	********** RMSPA_df ********** +----------+---------+ | Metric | Value | +----------+---------+ | Accuracy | 48.19% | | Hamming | 51.81% | | Macro-average Recall | 35.84% | | Macro-average F1 Score | 29.08% | | Micro-average Recall | 48.19% | | Micro-average F1 Score | 48.19% | +----------+---------+	
General Electric	********** GE_df ********** +----------+---------+ | Metric | Value | +----------+---------+ | Accuracy | 51.67% | | Hamming | 48.33% | | Macro-average Recall | 42.08% | | Macro-average F1 Score | 39.70% | | Micro-average Recall | 51.67% | | Micro-average F1 Score | 51.67% | +----------+---------+	

TotalEnergies	********** TTE_df ********** +-----------+--------+ | Metric | Value | +-----------+--------+ | Accuracy | 49.67% | | Hamming | 50.33% | | Macro-average Recall | 37.58% | | Macro-average F1 Score | 32.26% | | Micro-average Recall | 49.67% | | Micro-average F1 Score | 49.67% | +-----------+--------+	

En général, l'analyse des performances des modèles appliqués à différents DataFrames révèle des résultats variables. Pour l'ensemble des données, l' accuracy variait de 39.58% à 51.92%. La Hamming Loss affiche des valeurs entre 48.08% et 60.42%. Les métriques macro-average, telles que le rappel et le score F1, démontrent des performances modérées, avec des valeurs allant de 35.07% à 42.08% et de 26.82% à 39.70%, respectivement. Les métriques micro-average présentent une performance globalement plus modérée, avec des valeurs entre 39.58% et 51.92%. Ces résultats soulignent la diversité des performances des modèles sur différents ensembles de données et suggèrent la nécessité d'une analyse approfondie et d'ajustements potentiels pour améliorer la robustesse du modèle.

- SVM (Support Vector Machine)

Microsoft	+-----------+--------+ | Metric | Value | +-----------+--------+ | Accuracy | 53.33% | | Hamming | 46.67% | | Macro-average Recall | 38.50% | | Macro-average F1 Score | 33.10% | | Micro-average Recall | 53.33% | | Micro-average F1 Score | 53.33% | +-----------+--------+
Amazon	+-----------+--------+ | Metric | Value | +-----------+--------+ | Accuracy | 43.67% | | Hamming | 56.33% | | Macro-average Recall | 41.09% | | Macro-average F1 Score | 39.82% | | Micro-average Recall | 43.67% | | Micro-average F1 Score | 43.67% | +-----------+--------+
Tesla	+-----------+--------+ | Metric | Value | +-----------+--------+ | Accuracy | 44.13% | | Hamming | 55.87% | | Macro-average Recall | 36.60% | | Macro-average F1 Score | 28.33% | | Micro-average Recall | 44.13% | | Micro-average F1 Score | 44.13% | +-----------+--------+

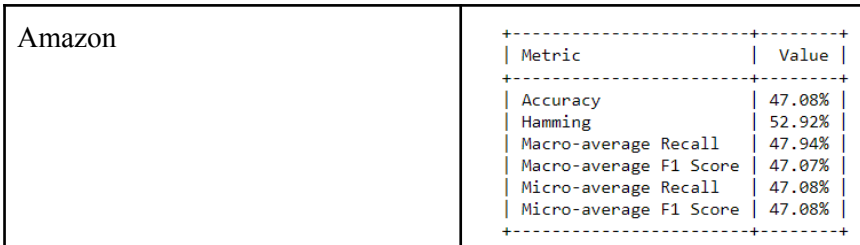
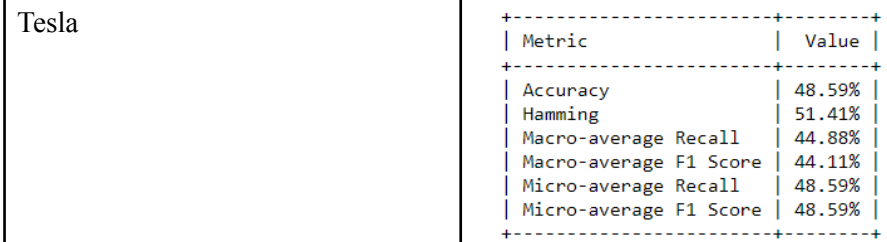
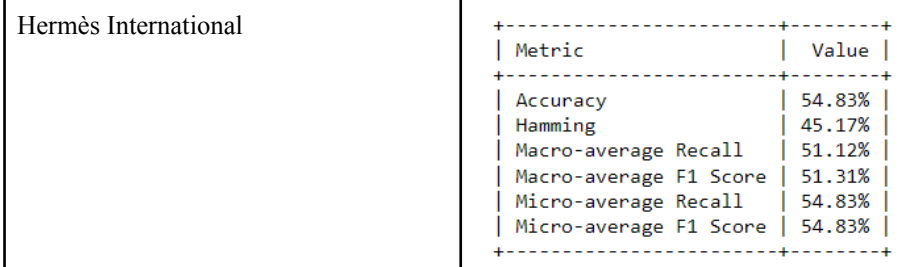
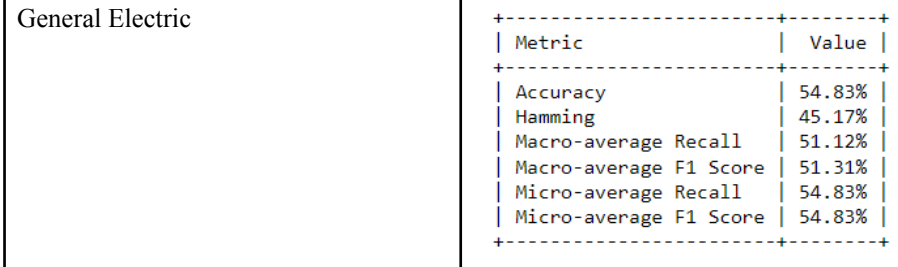
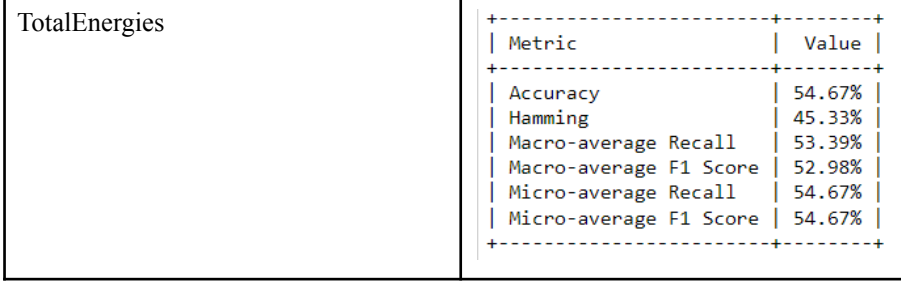
Hermès International	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 49.73% | | Hamming | 50.27% | | Macro-average Recall | 37.85% | | Macro-average F1 Score | 32.65% | | Micro-average Recall | 49.73% | | Micro-average F1 Score | 49.73% | +------------------------+--------+
General Electric	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 52.17% | | Hamming | 47.83% | | Macro-average Recall | 42.74% | | Macro-average F1 Score | 40.95% | | Micro-average Recall | 52.17% | | Micro-average F1 Score | 52.17% | +------------------------+--------+
TotalEnergies	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 51.17% | | Hamming | 48.83% | | Macro-average Recall | 39.65% | | Macro-average F1 Score | 35.86% | | Micro-average Recall | 51.17% | | Micro-average F1 Score | 51.17% | +------------------------+--------+

Les résultats montrent des performances variables d'un indice à l'autre.
L'accuracy varie entre 43.67% et 53.33%, indiquant une précision modérée.
La Hamming Loss oscille entre 46.67% et 56.33%, révélant des incohérences entre prédictions et étiquettes.
Les métriques macro-average montrent des scores de rappel et F1 allant de 36.60% à 42.74%, indiquant des difficultés à bien performer pour toutes les classes.
Les métriques micro-average démontrent une performance plus modérée, avec des scores entre 43.67% et 53.33%.

Cependant on remarque que la régression logistique a tendance à montrer des performances globalement inférieures à celles du SVM.

- K plus proches voisins (KNN)

Microsoft	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 52.08% | | Hamming | 47.92% | | Macro-average Recall | 43.80% | | Macro-average F1 Score | 43.21% | | Micro-average Recall | 52.08% | | Micro-average F1 Score | 52.08% | +------------------------+--------+

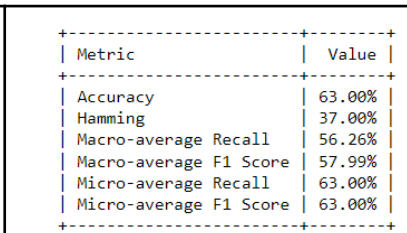
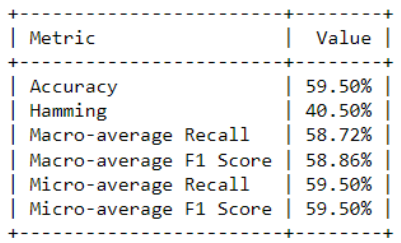
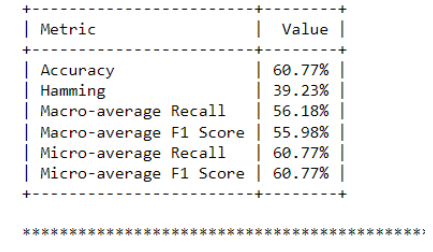
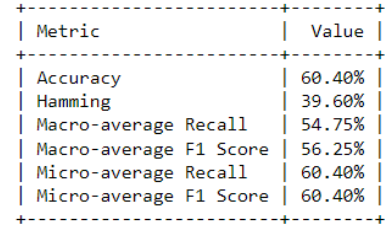
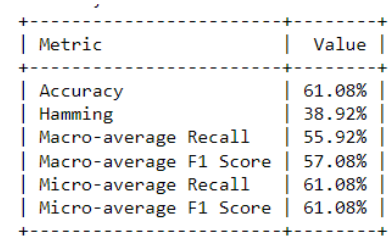
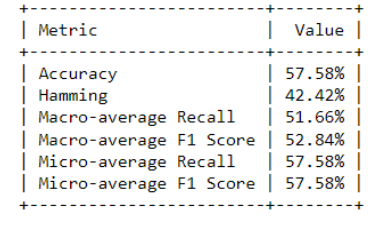
Amazon	
Tesla	
Hermès International	
General Electric	
TotalEnergies	

Le modèle KNN présente des performances globalement équilibrées sur diverses données. Avec une précision de 52.08% en moyenne, le modèle démontre une capacité modérée à effectuer des prédictions précises. La perte de Hamming est de 47.92% . Le rappel moyen par classe et le score F1 moyen par classe reflètent une capacité modérée à rappeler et à maintenir un équilibre entre précision et rappel pour chaque classe.

Les modèles classiques semblent avoir des difficultés à capturer la complexité des données financières.

Modèles Ensemblistes

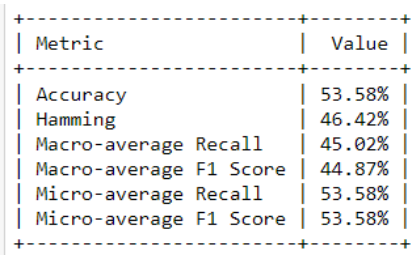
- Random Forest

Microsoft	
Amazon	
Tesla	
Hermès International	
General Electric	
TotalEnergies	

Le modèle a atteint une précision moyenne allant de 57.25% à 63.83%, ce qui suggère une capacité significative à classifier correctement les mouvements du marché. La métrique Hamming indique une adéquation globale du modèle aux données.
Ce modèle Random Forest présente de bonnes performances comparé aux modèles classiques, avec une capacité notable à généraliser sur différents indices boursiers.

- Gradient Boosting

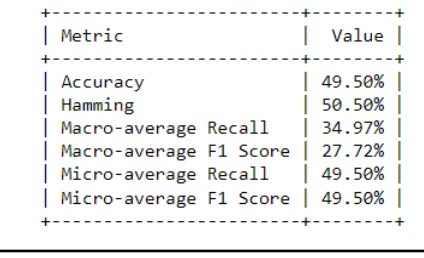
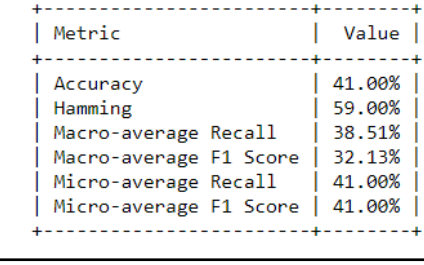
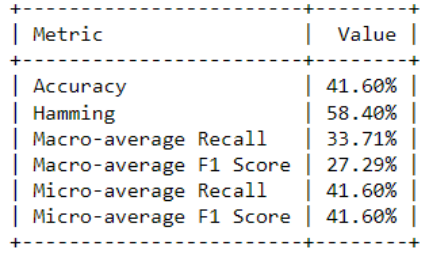
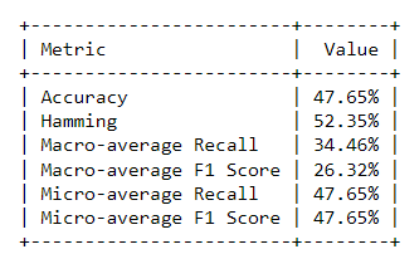
Microsoft	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 58.92% | | Hamming | 41.08% | | Macro-average Recall | 48.45% | | Macro-average F1 Score | 48.93% | | Micro-average Recall | 58.92% | | Micro-average F1 Score | 58.92% | +-------------------------+--------+
Amazon	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 53.33% | | Hamming | 46.67% | | Macro-average Recall | 51.98% | | Macro-average F1 Score | 52.07% | | Micro-average Recall | 53.33% | | Micro-average F1 Score | 53.33% | +-------------------------+--------+
Tesla	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 52.60% | | Hamming | 47.40% | | Macro-average Recall | 46.01% | | Macro-average F1 Score | 42.62% | | Micro-average Recall | 52.60% | | Micro-average F1 Score | 52.60% | +-------------------------+--------+
Hermès International	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 55.06% | | Hamming | 44.94% | | Macro-average Recall | 45.79% | | Macro-average F1 Score | 45.39% | | Micro-average Recall | 55.06% | | Micro-average F1 Score | 55.06% | +-------------------------+--------+
General Electric	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 56.00% | | Hamming | 44.00% | | Macro-average Recall | 48.51% | | Macro-average F1 Score | 48.66% | | Micro-average Recall | 56.00% | | Micro-average F1 Score | 56.00% | +-------------------------+--------+

TotalEnergies	

Le modèle a atteint une précision moyenne allant de 52.83% à 57.08%, indiquant une capacité notable à classifier correctement les mouvements du marché.

Réseau de neurone

- LSTM

Microsoft	
Amazon	
Tesla	
Hermès International	

General Electric	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 51.50% | | Hamming | 48.50% | | Macro-average Recall | 41.22% | | Macro-average F1 Score | 38.19% | | Micro-average Recall | 51.50% | | Micro-average F1 Score | 51.50% | +------------------------+--------+
TotalEnergies	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 48.00% | | Hamming | 52.00% | | Macro-average Recall | 36.62% | | Macro-average F1 Score | 30.49% | | Micro-average Recall | 48.00% | | Micro-average F1 Score | 48.00% | +------------------------+--------+

En moyenne, le modèle LSTM a affiché des performances modestes avec une précision moyenne d'environ 46.30%.

Le score Hamming moyen de 53.70% indique une certaine difficulté à traiter les erreurs de classification multi-étiquettes. Les scores de rappel macro-moyen (38.92%) et de F1 Score macro-moyen (32.70%) soulignent les défis rencontrés par le modèle pour généraliser efficacement les tendances

- RNN

Microsoft	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 49.17% | | Hamming | 50.83% | | Macro-average Recall | 35.00% | | Macro-average F1 Score | 28.39% | | Micro-average Recall | 49.17% | | Micro-average F1 Score | 49.17% | +------------------------+--------+
Amazon	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 41.67% | | Hamming | 58.33% | | Macro-average Recall | 39.13% | | Macro-average F1 Score | 32.63% | | Micro-average Recall | 41.67% | | Micro-average F1 Score | 41.67% | +------------------------+--------+

Tesla	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 42.94% | | Hamming | 57.06% | | Macro-average Recall | 34.20% | | Macro-average F1 Score | 25.99% | | Micro-average Recall | 42.94% | | Micro-average F1 Score | 42.94% | +------------------------+--------+
Hermès International	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 47.29% | | Hamming | 52.71% | | Macro-average Recall | 34.32% | | Macro-average F1 Score | 26.37% | | Micro-average Recall | 47.29% | | Micro-average F1 Score | 47.29% | +------------------------+--------+
General Electric	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 50.83% | | Hamming | 49.17% | | Macro-average Recall | 40.10% | | Macro-average F1 Score | 35.19% | | Micro-average Recall | 50.83% | | Micro-average F1 Score | 50.83% | +------------------------+--------+
TotalEnergies	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 48.00% | | Hamming | 52.00% | | Macro-average Recall | 36.46% | | Macro-average F1 Score | 30.09% | | Micro-average Recall | 48.00% | | Micro-average F1 Score | 48.00% | +------------------------+--------+

Les résultats du modèle RNN pour la prédiction des tendances montrent une performance moyenne, avec une précision moyenne de 47.78% et le score Hamming de 52.22%.

- CNN

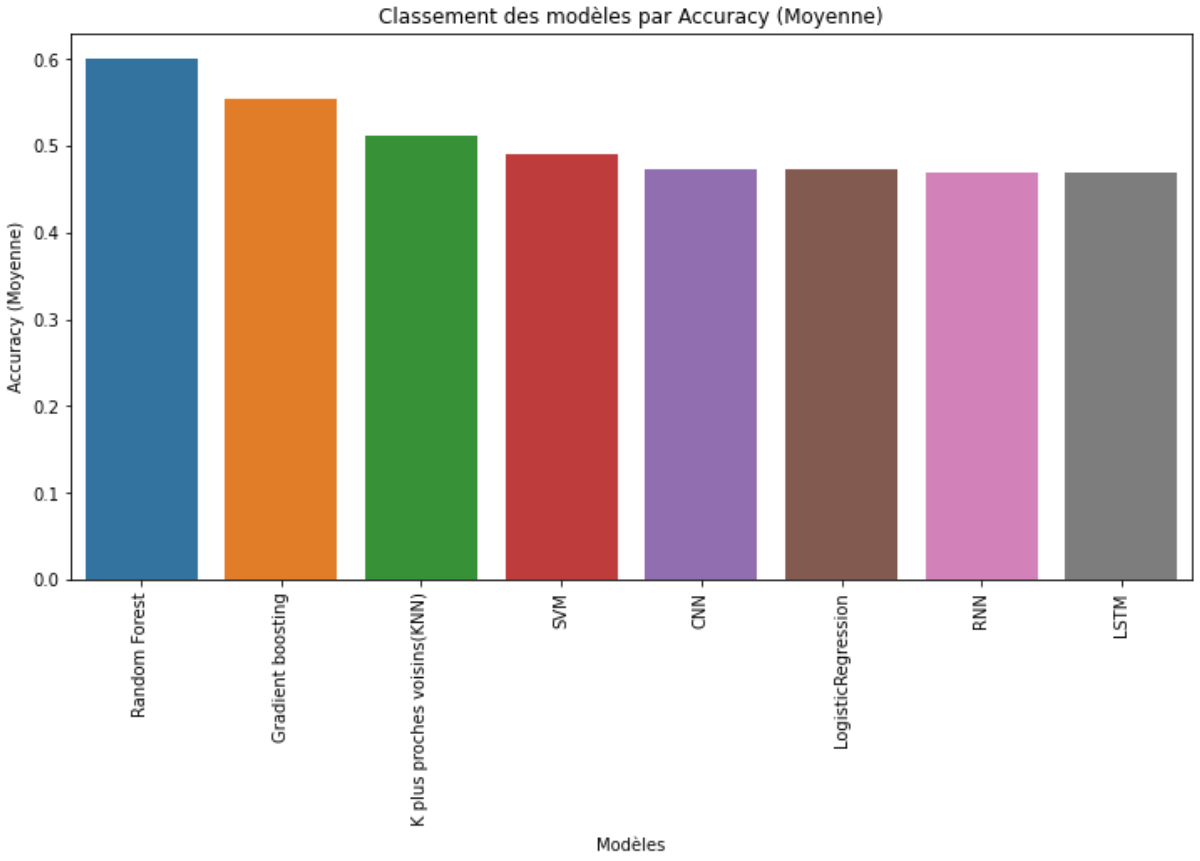
Microsoft	+------------------------+--------+ | Metric | Value | +------------------------+--------+ | Accuracy | 50.08% | | Hamming | 49.92% | | Macro-average Recall | 34.60% | | Macro-average F1 Score | 25.53% | | Micro-average Recall | 50.08% | | Micro-average F1 Score | 50.08% | +------------------------+--------+

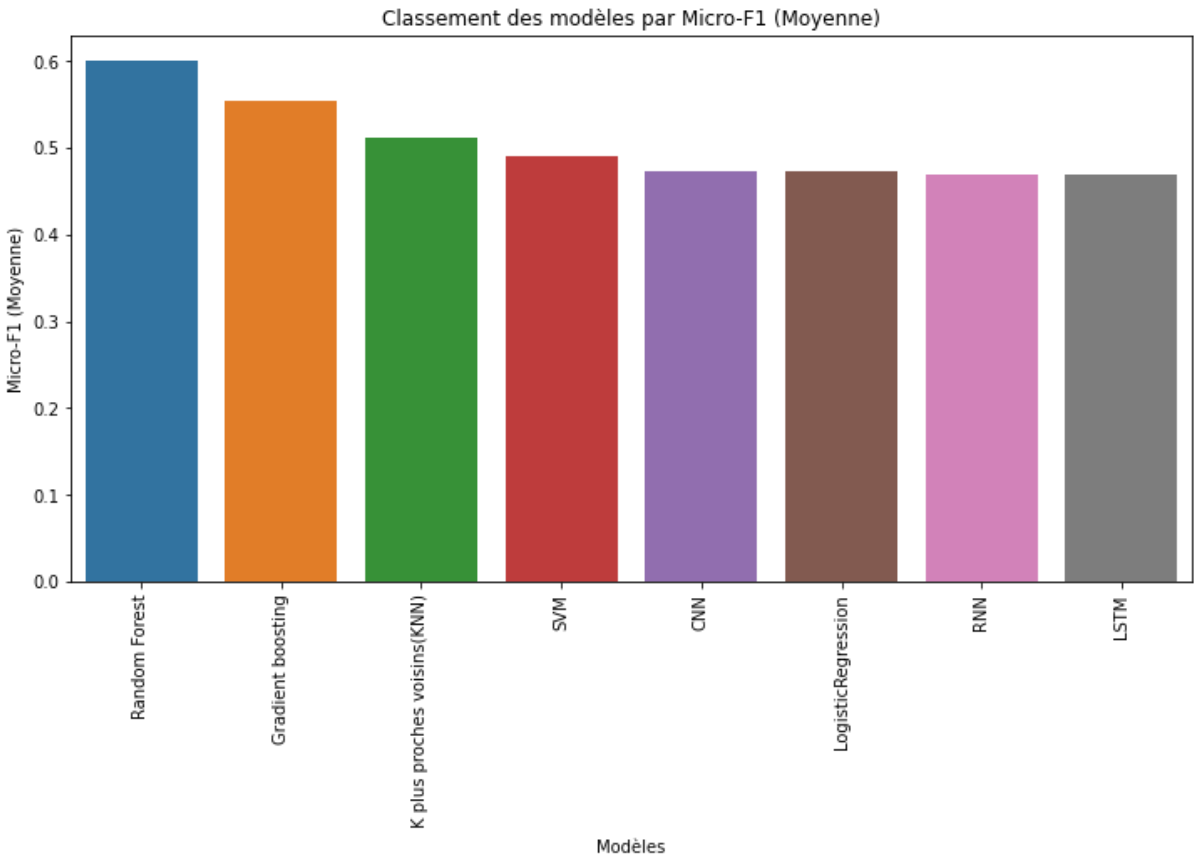
Amazon	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 41.33% | | Hamming | 58.67% | | Macro-average Recall | 38.99% | | Macro-average F1 Score | 33.72% | | Micro-average Recall | 41.33% | | Micro-average F1 Score | 41.33% | +-------------------------+--------+
Tesla	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 42.20% | | Hamming | 57.80% | | Macro-average Recall | 34.88% | | Macro-average F1 Score | 29.48% | | Micro-average Recall | 42.20% | | Micro-average F1 Score | 42.20% | +-------------------------+--------+ **********
Hermès International	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 47.65% | | Hamming | 52.35% | | Macro-average Recall | 34.31% | | Macro-average F1 Score | 25.76% | | Micro-average Recall | 47.65% | | Micro-average F1 Score | 47.65% | +-------------------------+--------+
General Electric	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 51.75% | | Hamming | 48.25% | | Macro-average Recall | 42.41% | | Macro-average F1 Score | 41.00% | | Micro-average Recall | 51.75% | | Micro-average F1 Score | 51.75% | +-------------------------+--------+
TotalEnergies	+-------------------------+--------+ | Metric | Value | +-------------------------+--------+ | Accuracy | 48.58% | | Hamming | 51.42% | | Macro-average Recall | 38.86% | | Macro-average F1 Score | 34.48% | | Micro-average Recall | 48.58% | | Micro-average F1 Score | 48.58% | +-------------------------+--------+

Les performances du modèle CNN pour la prédiction des tendances montrent une précision moyenne de 47.42%. Bien que le score Hamming de 52.58% indique des difficultés dans la gestion des erreurs de classification multi-étiquettes, les résultats montrent une tendance à surperformer le modèle RNN. Les scores de rappel macro-moyen (36.59%) et de F1 Score macro-moyen (30.01%) suggèrent que le

modèle a une certaine capacité à généraliser les tendances du marché. Cependant, les performances varient d'une entreprise à l'autre.

3. Comparaison des modèles





1. Les modèles ensemblistes, en particulier Random Forest et Gradient Boosting, ont démontré une meilleure capacité. Il est à noter que ces deux modèles, bien performants, exigent un temps d'entraînement considérable.

2. Suivi par le KNN et SVM

3. En ce qui concerne les modèles de réseau de neurones tels que LSTM, RNN et CNN, leurs performances sont plus modestes, soulignant la complexité de la prédiction des tendances sur des données financières.

Conclusions et perspectives

Dans le cadre de ce projet visant à anticiper les évolutions hebdomadaires des actions de six indices. Les performances diverses des modèles appliqués à chacun des ensembles de données ont révélé des nuances essentielles dans la prédiction des tendances boursières.

Les approches classiques, telles que la régression logistique, le SVM, et le KNN, ont manifesté des performances modérées, soulignant la nécessité d'explorer des méthodes plus avancées. Les modèles ensemblistes, notamment Random Forest et Gradient Boosting, ont émergé comme des choix plus robustes, illustrant leur capacité à généraliser sur divers indices boursiers.

Les modèles de réseau de neurones, tels que LSTM, RNN, et CNN, bien qu'offrant un potentiel prometteur, ont démontré des performances plus modestes. Ceci souligne la complexité inhérente à la prédiction des tendances financières, nécessitant une approche personnalisée et une adaptation continue pour maximiser leur robustesse.

La dynamique et la complexité du marché financier imposent des défis importants, soulignant l'impact crucial des choix méthodologiques et des caractéristiques des données sur les performances des modèles. Les limitations telles que la taille restreinte de l'échantillon de données et l'absence de causalité dans les algorithmes mettent en exergue la nécessité d'une approche itérative et d'une amélioration continue des modèles.

Pour renforcer la précision des prédictions, des avenues d'amélioration sont envisageables, notamment une optimisation plus approfondie des hyperparamètres, l'intégration de données supplémentaires, l'enrichissement des features, l'utilisation d'ensembles de modèles, et une analyse plus approfondie des erreurs de prédiction.

En conclusion, bien que la prédiction des évolutions hebdomadaires des actions demeure un défi complexe, ce projet a tracé une voie pour des améliorations continues. Il souligne l'importance capitale de l'itération et de l'adaptation constante dans le domaine de la data science appliquée à la finance, offrant ainsi des perspectives optimistes pour des avancées futures dans la précision des modèles financiers.

Bibliographie

Travers, Emmeline. "Les principaux modèles de prédiction des cours." Regards Croisés sur l'Économie, 2008, p. 141. [lien]
Vincent Bouchet.“ Machine learning en finance : vers de nouvelles stratégies ?”
lien

Prashant Sharma. “Stock Market Prediction Using Machine Learning” lien

Antoine Krajnc. “PRÉDIRE LE COURS D’UNE ACTION EN BOURSE” lien

Annexe

code_DS

January 31, 2024

[1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib. pyplot as plt
import matplotlib
from sklearn.preprocessing import MinMaxScaler
from keras.layers import LSTM, Dense, Dropout
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib. dates as mandates
from sklearn.preprocessing import MinMaxScaler
from sklearn import linear_model
from keras.models import Sequential
from keras.layers import Dense
import keras.backend as K
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
from keras.models import load_model
from keras.layers import LSTM
from keras.utils.vis_utils import plot_model


import warnings

# Ignorer tous les avertissements
warnings.filterwarnings("ignore")
```

```
C:\Users\bouch\anaconda3\lib\site-packages\numpy\_distributor_init.py:30:
UserWarning: loaded more than 1 DLL from .libs:
C:\Users\bouch\anaconda3\lib\site-
packages\numpy\.libs\libopenblas.GK7GX5KEQ4F6UYO3P26ULGBQYHGQO7J4.gfortran-
win_amd64.dll
C:\Users\bouch\anaconda3\lib\site-
packages\numpy\.libs\libopenblas.XWYDX2IKJW2NMTWSFYNGFUWKQU3LYTCZ.gfortran-
win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
<frozen importlib._bootstrap>:228: RuntimeWarning:
scipy._lib.messagestream.MessageStream size changed, may indicate binary
```

```
incompatibility. Expected 56 from C header, got 64 from PyObject
```

0.1 Chargement et Exploration des Données Boursières depuis un Fichier Excel

[2]:
```python
# Charger le fichier Excel
excel_file_path = 'Historiques_cours_boursiers.xlsx'
xls = pd.ExcelFile('Historiques_cours_boursiers.xlsx')

# Lire chaque onglet dans une boucle
dataframes = {}  # Utiliser un dictionnaire pour stocker les DataFrames de␣
 ↪chaque onglet

for sheet_name in xls.sheet_names:
    # Lire les données de l'onglet en cours
    df = pd.read_excel(excel_file_path, sheet_name)

    # Stocker le DataFrame dans le dictionnaire
    dataframes[sheet_name] = df

# Accéder à un DataFrame spécifique (par exemple, le premier onglet)
premier_onglet = dataframes[xls.sheet_names[0]]
```

Ce code charge un fichier Excel contenant des données boursières, lit chaque onglet du fichier Excel dans un DataFrame distinct, puis stocke ces DataFrames dans un dictionnaire.

6 indices boursiers parmi les 13

[3]:
```python
#6 choix

list_choice= {'MSFT_df':dataframes[xls.sheet_names[9]],
              'AMZN_df' :dataframes[xls.sheet_names[12]],
              'TSLA_df':dataframes[xls.sheet_names[4]],
            'RMSPA_df':dataframes[xls.sheet_names[3]],
              'GE_df':dataframes[xls.sheet_names[2]],
            'TTE_df':dataframes[xls.sheet_names[1]]}
```

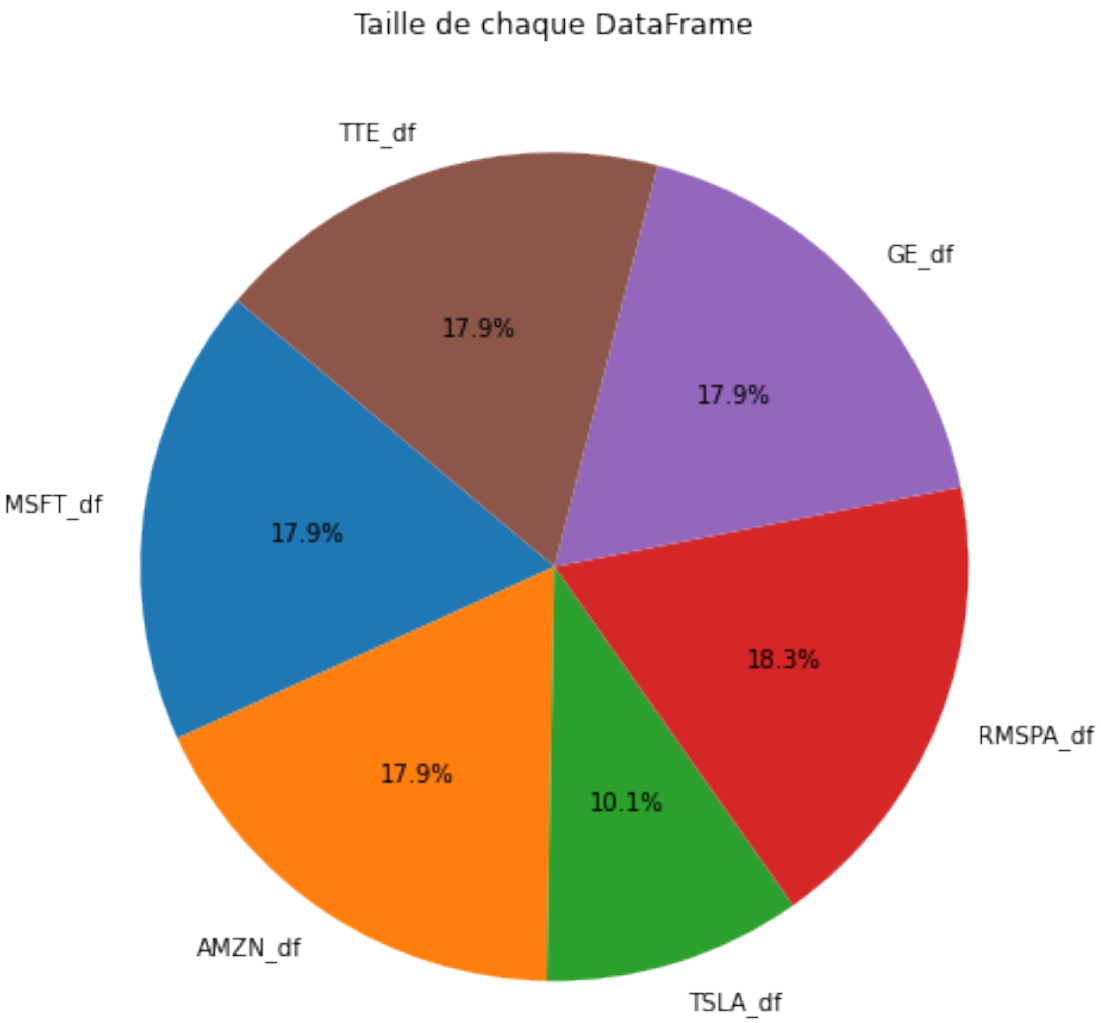
Tailles des fichiers

[4]:
```python
# Récupérer les noms des DataFrames et leurs tailles
df_sizes = {name: df.shape[0] for name, df in list_choice.items()}

# Afficher la taille de chaque DataFrame en texte
for name, size in df_sizes.items():
    print(f"La taille de {name} est {size} lignes.")
```

```
La taille de MSFT_df est 6031 lignes.
La taille de AMZN_df est 6031 lignes.
La taille de TSLA_df est 3394 lignes.
La taille de RMSPA_df est 6165 lignes.
```

```
La taille de GE_df est 6031 lignes.
La taille de TTE_df est 6031 lignes.
```

[5]:
```python
df_names = list(list_choice.keys())
df_sizes = [df.shape[0] for df in list_choice.values()]

# Créer un diagramme circulaire
plt.figure(figsize=(8, 8))
plt.pie(df_sizes, labels=df_names, autopct='%1.1f%%', startangle=140)
plt.title('Taille de chaque DataFrame')
plt.show()
```



Microsoft, Amazon, General Electric et Hermès présentent un nombre équivalent de lignes, soit 5967, tandis que Tesla comporte 3330 lignes.

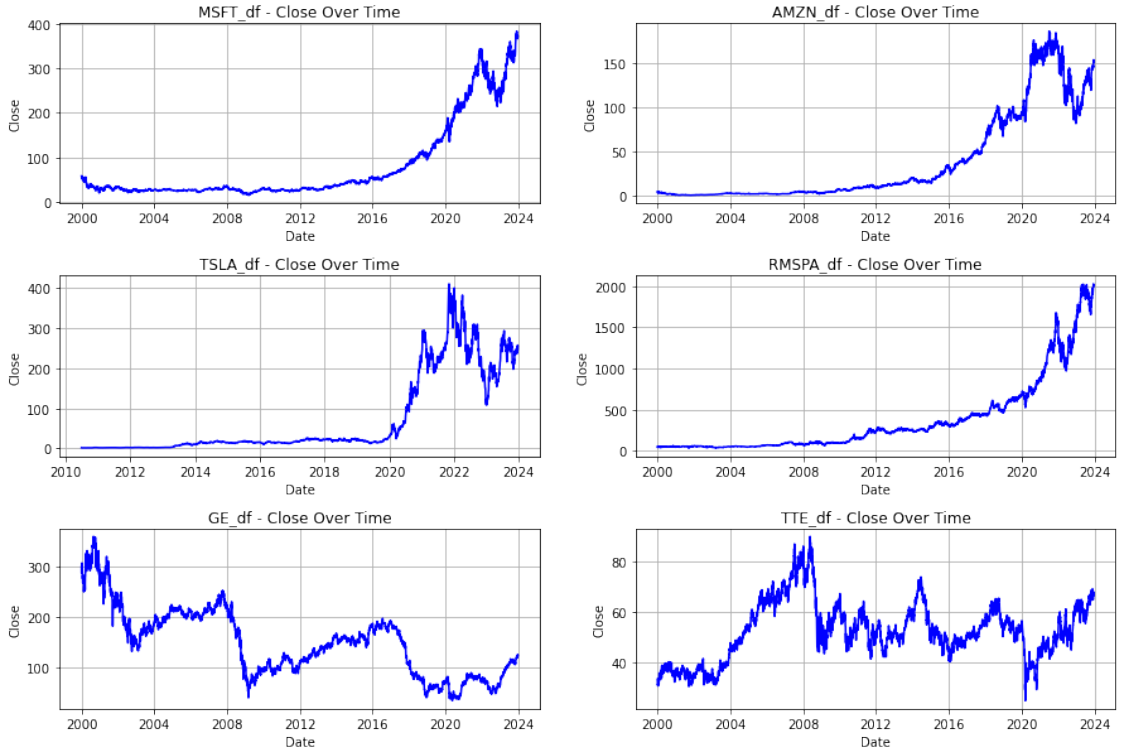
0.2 1. Data preparation

[6]:
```
for key, df in list_choice.items():

    df['Day'] = df['Date'].dt.day
    df['Month'] = df['Date'].dt.month
    df['Year'] = df['Date'].dt.year
```

0.2.1 2. Exploratory Data Analysis

[7]:
```
import matplotlib.pyplot as plt

# Nombre total de sous-plots
num_plots = len(list_choice)

# Définir le nombre de lignes et de colonnes dans la grille
num_rows = 3
num_cols = 2

# Créer la grille de sous-plots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))

# Ajuster l'espacement entre les sous-plots
plt.subplots_adjust(hspace=0.4)

# Parcourir chaque DataFrame dans la liste
for idx, (df_name, df) in enumerate(list_choice.items()):
    # Convertir la colonne 'Date' en un tableau NumPy
    dates = df['Date'].to_numpy()

    # Convertir la colonne 'Adj Close' en un tableau NumPy
    close = df['Close'].to_numpy()
    opens=df['Open'].to_numpy()
    # Calculer les indices de ligne et de colonne
    row_idx = idx // num_cols
    col_idx = idx % num_cols

    # Tracer le graphique pour le DataFrame actuel dans le sous-plot␣
 ↪correspondant
    axes[row_idx, col_idx].plot(dates, close, label='Close',color='blue')
#     axes[row_idx, col_idx].plot(dates, opens, label='Open', color='blue')
    axes[row_idx, col_idx].set_title(f'{df_name} - Close Over Time')
    axes[row_idx, col_idx].set_xlabel('Date')
    axes[row_idx, col_idx].set_ylabel('Close')
    axes[row_idx, col_idx].grid(True)

# Afficher la grille de sous-plots
```

```
plt.show()
```

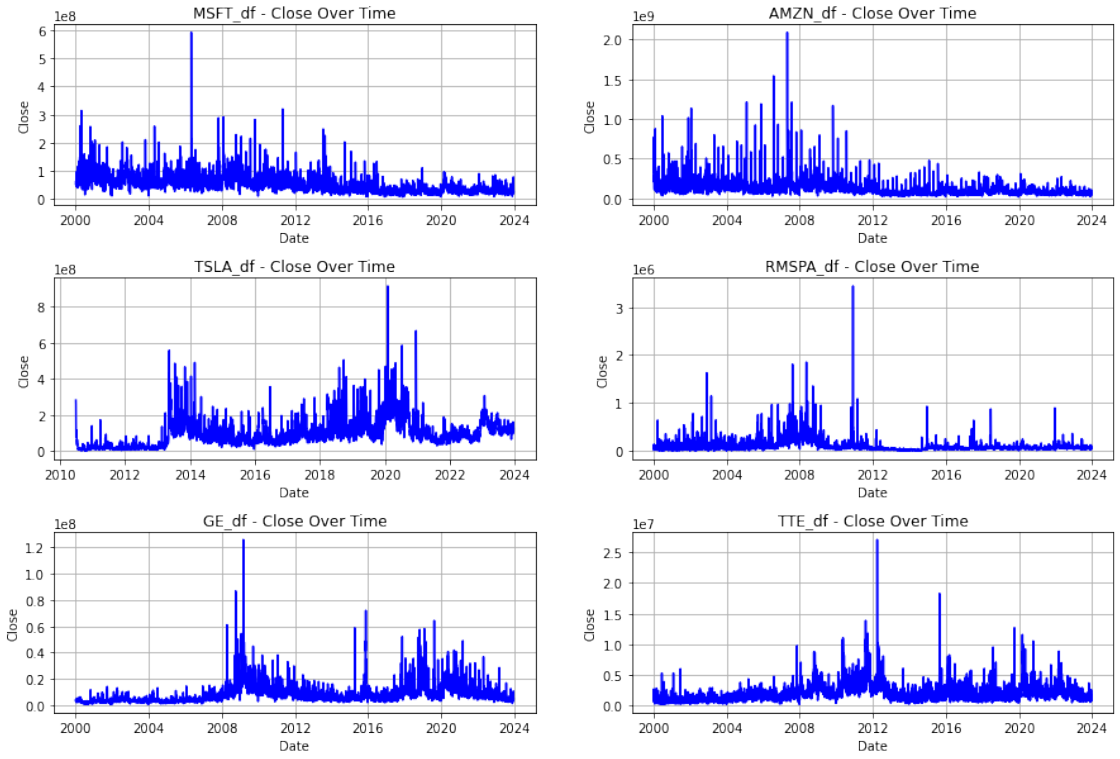


La trajectoire des valeurs de clôture varie considérablement d'un groupe à l'autre. Certains, tels que Microsoft, Amazon et Hermès, initient leur évolution entre 0 et 100 au cours de la période de 2000 à 2016. En revanche, Tesla commence sa série chronologique en 2010.

L'analyse graphique met en évidence que l'évolution des cours de clôture est propre à chaque groupe. Chaque titre exhibe des tendances et des comportements uniques au fil du temps.

[8]:
```
import matplotlib.pyplot as plt

# Nombre total de sous-plots
num_plots = len(list_choice)

# Définir le nombre de lignes et de colonnes dans la grille
num_rows = 3
num_cols = 2

# Créer la grille de sous-plots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))

# Ajuster l'espacement entre les sous-plots
plt.subplots_adjust(hspace=0.4)
```

```python
# Parcourir chaque DataFrame dans la liste
for idx, (df_name, df) in enumerate(list_choice.items()):
    # Convertir la colonne 'Date' en un tableau NumPy
    dates = df['Date'].to_numpy()

    # Convertir la colonne 'Adj Close' en un tableau NumPy
    volume = df['Volume'].to_numpy()

    # Calculer les indices de ligne et de colonne
    row_idx = idx // num_cols
    col_idx = idx % num_cols

    # Tracer le graphique pour le DataFrame actuel dans le sous-plot␣
 ↪correspondant
    axes[row_idx, col_idx].plot(dates, volume, label='Volume',color='blue')
#     axes[row_idx, col_idx].plot(dates, opens, label='Open', color='blue')
    axes[row_idx, col_idx].set_title(f'{df_name} - Close Over Time')
    axes[row_idx, col_idx].set_xlabel('Date')
    axes[row_idx, col_idx].set_ylabel('Close')
    axes[row_idx, col_idx].grid(True)

# Afficher la grille de sous-plots
plt.show()
```



0.2.2 var à expliquer

[9]:
```
def calculate_trend(close_prices):
    # Initialize the target column with neutral values
    target = pd.Series(index=close_prices.index, dtype=int).fillna(0)

# Calculate the percentage change between the current closing price and the␣
 ↪closing price five days later
    pct_change = 100 * (close_prices.shift(-5) - close_prices) / close_prices

# Determine the trend
    target[pct_change > 2] = 1  # Bullish trend
    target[pct_change < -2] = -1  # Bearish trend

# Return the target series
    return target.iloc[:-5].astype(int)
```

[10]:
```
for key, df in list_choice.items():
    # Extraire la colonne 'Close' comme une série
    close_prices = df['Close']

    # Utiliser la fonction calculate_trend
    trend = calculate_trend(close_prices)

    # Ajouter la colonne 'Trend' au DataFrame
    df['Trend'] = trend

    # Afficher le DataFrame avec la colonne 'Trend'
#     print(f"\nDataFrame {key} avec la colonne 'Trend':")
#     print(df)
```

[11]:
```
list_choice['MSFT_df'].head(15)
```

[11]:

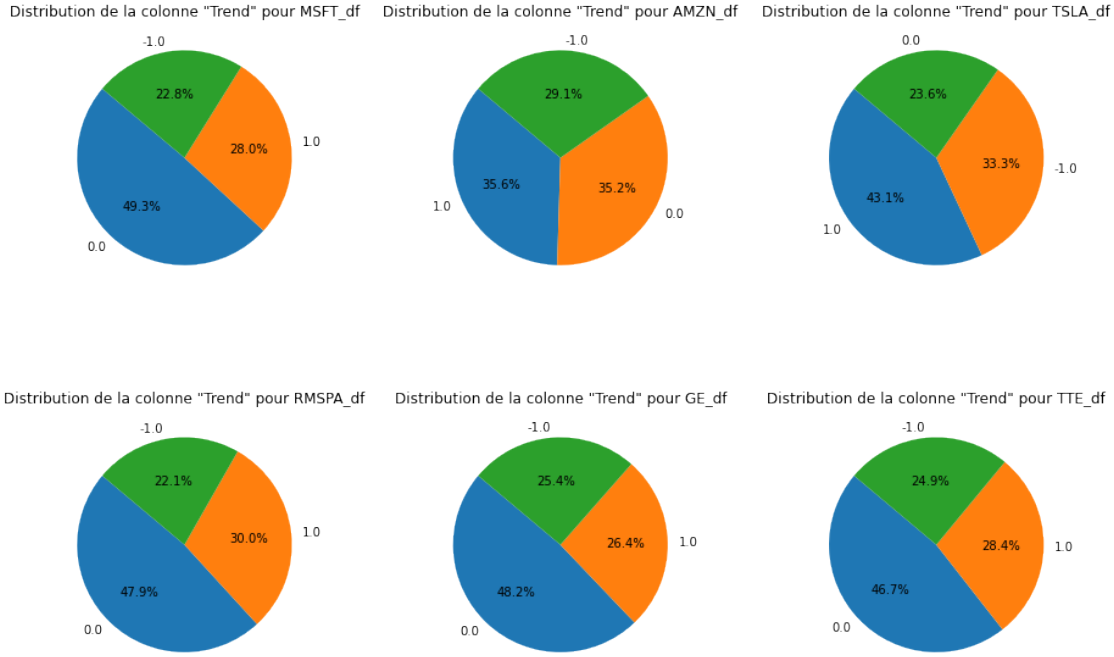
	Date	Open	High	Low	Close	Adj Close	Volume	\
0	2000-01-03	58.687500	59.31250	56.00000	58.28125	36.132240	53228400	
1	2000-01-04	56.781250	58.56250	56.12500	56.31250	34.911697	54119000	
2	2000-01-05	55.562500	58.18750	54.68750	56.90625	35.279823	64059600	
3	2000-01-06	56.093750	56.93750	54.18750	55.00000	34.098007	54976600	
4	2000-01-07	54.312500	56.12500	53.65625	55.71875	34.543606	62013600	
5	2000-01-10	56.718750	56.84375	55.68750	56.12500	34.795467	44963600	
6	2000-01-11	55.750000	57.12500	54.34375	54.68750	33.904270	46743600	
7	2000-01-12	54.250000	54.43750	52.21875	52.90625	32.799969	66532400	
8	2000-01-13	52.187500	54.31250	50.75000	53.90625	33.419930	83144000	
9	2000-01-14	53.593750	56.96875	52.87500	56.12500	34.795467	73416400	
10	2000-01-18	55.906250	58.25000	55.87500	57.65625	35.744789	81483600	
11	2000-01-19	55.250000	55.75000	53.00000	53.50000	33.168060	97568200	
12	2000-01-20	53.531250	54.84375	52.93750	53.00000	32.858070	56349800	

```
13 2000-01-21  53.500000  53.62500  51.62500  51.87500  32.160625  68416200
14 2000-01-24  51.898438  52.84375  50.40625  50.62500  31.385658  63597600
```

```
    Day  Month  Year  Trend
0     3      1  2000   -1.0
1     4      1  2000   -1.0
2     5      1  2000   -1.0
3     6      1  2000    0.0
4     7      1  2000    0.0
5    10      1  2000    1.0
6    11      1  2000   -1.0
7    12      1  2000    0.0
8    13      1  2000   -1.0
9    14      1  2000   -1.0
10   18      1  2000   -1.0
11   19      1  2000   -1.0
12   20      1  2000   -1.0
13   21      1  2000   -1.0
14   24      1  2000   -1.0
```

[12]:
```python
import matplotlib.pyplot as plt

# Définir le nombre de colonnes dans la grille
num_cols = 3

# Calculer le nombre de lignes nécessaire en fonction de la longueur de␣
 ↪list_choice
num_rows = -(-len(list_choice) // num_cols)  # Utilisation de la division␣
 ↪entière pour arrondir vers le haut

# Créer une grille de sous-graphiques
fig, axs = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))

# Ajuster l'espacement entre les sous-graphiques
plt.subplots_adjust(wspace=0.4, hspace=0.4)

# Parcourir chaque dataframe dans list_choice
for i, (key, df) in enumerate(list_choice.items()):
    # Calculer les indices de la ligne et de la colonne pour placer le pie chart
    row_idx = i // num_cols
    col_idx = i % num_cols
#     trend_counts = df['Trend'].to_numpy()
    # Sélectionner le sous-graphique correspondant dans la grille
    ax = axs[row_idx, col_idx] if num_rows > 1 else axs[col_idx]
    trend_counts = df['Trend'].value_counts()
    # Créer le pie chart pour la distribution de la colonne 'Trend'
```

```
    ax.pie(df['Trend'].value_counts(), labels=trend_counts.index, autopct='%1.
↪1f%%', startangle=140)
    ax.set_title(f'Distribution de la colonne "Trend" pour {key}')

# Afficher les sous-graphiques
plt.show()
```



0.3 Variables explicatives

[13]:
```
# Définir les périodes N
N_14 = 14
N_21 = 21
N_ATR = 14
N_CMF_21 = 21
N_CMF_28 = 28

# Seuil pour CMF
seuil_haussier = 0.25
seuil_baissier = -0.25

# Fonction pour calculer RSI
def calculate_rsi(data, period):
    delta = data.diff()
    gain = delta.where(delta > 0, 0)
```

```
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=period).mean()
    avg_loss = loss.rolling(window=period).mean()
    rs = avg_gain / avg_loss
    rsi_column = 100 - (100 / (1 + rs))
    return rsi_column

# Fonction pour calculer RSI trend
def calculate_rsi_trend(rsi_column, overbought=70, oversold=30):
    return np.where((rsi_column > overbought), 1, np.where((rsi_column <␣
 ↪oversold), -1, 0))

# Fonction pour calculer CMF
def calculate_cmf(data, period):
    mf_multiplier = ((data['Close'] - data['Low']) - (data['High'] -␣
 ↪data['Close'])) / (data['High'] - data['Low'])
    mf_volume = mf_multiplier * data['Volume']
    cmf_column = mf_volume.rolling(window=period).sum() / data['Volume'].
 ↪rolling(window=period).sum()
    return cmf_column

# Parcourir chaque dataframe dans list_choice
for key, df in list_choice.items():
    # Calculer MACD
    df['MACD_Line'] = df['Close'].ewm(span=12).mean() - df['Close'].
 ↪ewm(span=26).mean()
    df['MACD_Signal'] = df['MACD_Line'].ewm(span=9).mean()
    df['MACD_Histogram'] = df['MACD_Line'] - df['MACD_Signal']
    df['MACD_trend'] = np.where((df['MACD_Line'] > 0) | (df['MACD_Histogram'] >␣
 ↪0), 1,
                                np.where((df['MACD_Line'] < 0) |␣
 ↪(df['MACD_Histogram'] < 0), -1, 0))

    # Calculer RSI et les tendances pour chaque période
    df['RSI_14'] = calculate_rsi(df['Close'], N_14)
    df['RSI_trend_14'] = calculate_rsi_trend(df['RSI_14'])

    df['RSI_21'] = calculate_rsi(df['Close'], N_21)
    df['RSI_trend_21'] = calculate_rsi_trend(df['RSI_21'])

    # Calculer ATR
    tr = pd.DataFrame(index=df.index)
    tr['HL'] = df['High'] - df['Low']
    tr['HC'] = abs(df['High'] - df['Close'].shift())
    tr['LC'] = abs(df['Low'] - df['Close'].shift())
    tr['TrueRange'] = tr[['HL', 'HC', 'LC']].max(axis=1)
```

```
    df['ATR'] = tr['TrueRange'].rolling(window=N_ATR).mean()

    # Calculer CMF
    df['CMF_21'] = calculate_cmf(df, N_CMF_21)
    df['CMF_28'] = calculate_cmf(df, N_CMF_28)

    # Ajouter la colonne cmf_trend
    df['cmf_trend_21'] = np.where(df['CMF_21'] > seuil_haussier, 1,
                                  np.where(df['CMF_21'] < seuil_baissier, -1,␣
 ↪0))

    df['cmf_trend_28'] = np.where(df['CMF_28'] > seuil_haussier, 1,
                                  np.where(df['CMF_28'] < seuil_baissier, -1,␣
 ↪0))

    # Calculer Daily Range
    df['Daily_Range'] = df['High'] - df['Low']

    # Calculer Gap
    df['Gap'] = df['Open'] - df['Close'].shift(1)

    df.dropna(inplace=True)
    # Afficher le DataFrame avec les nouvelles colonnes
#     print(f"\nDataFrame {key} avec les nouvelles colonnes:")
#     print(df)
```

[14]:
```
list_choice['MSFT_df']
```

[14]:

	Date	Open	High	Low	Close	Adj Close	\
27	2000-02-10	51.945313	53.281250	51.250000	53.000000	32.858070	
28	2000-02-11	52.437500	52.437500	49.562500	49.968750	30.978819	
29	2000-02-14	50.617188	50.875000	49.531250	49.812500	30.881937	
30	2000-02-15	49.875000	50.000000	49.062500	49.281250	30.552582	
31	2000-02-16	49.625000	50.093750	48.562500	48.812500	30.261980	
…	…	…	…	…	…	…	
6021	2023-12-07	368.230011	371.450012	366.320007	370.950012	370.950012	
6022	2023-12-08	369.200012	374.459991	368.230011	374.230011	374.230011	
6023	2023-12-11	368.480011	371.600006	366.100006	371.299988	371.299988	
6024	2023-12-12	370.850006	374.420013	370.459991	374.380005	374.380005	
6025	2023-12-13	376.019989	377.640015	370.769989	374.369995	374.369995	

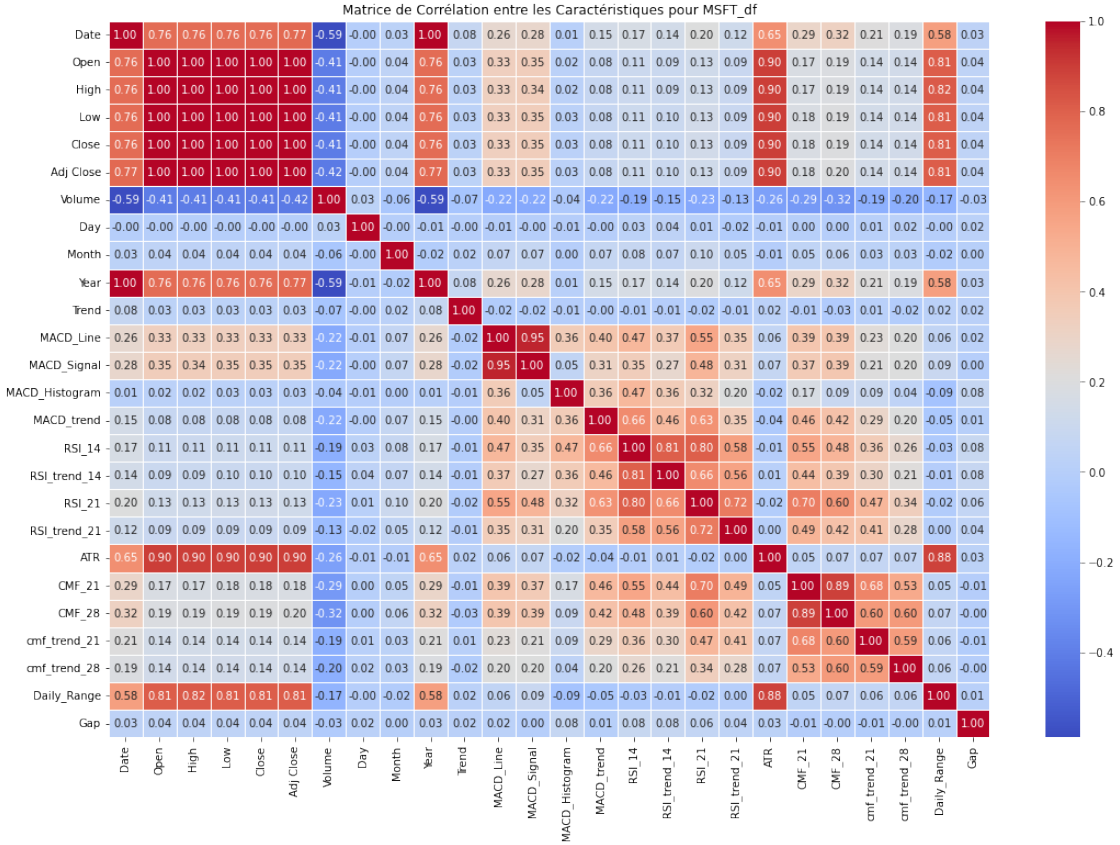
	Volume	Day	Month	Year	…	RSI_trend_14	RSI_21	RSI_trend_21	\
27	54527800	10	2	2000	…	0	47.084233	0	
28	115559000	11	2	2000	…	0	45.134576	0	
29	81028600	14	2	2000	…	0	43.024494	0	
30	71027600	15	2	2000	…	0	37.627119	0	
31	65202600	16	2	2000	…	0	33.372503	0	

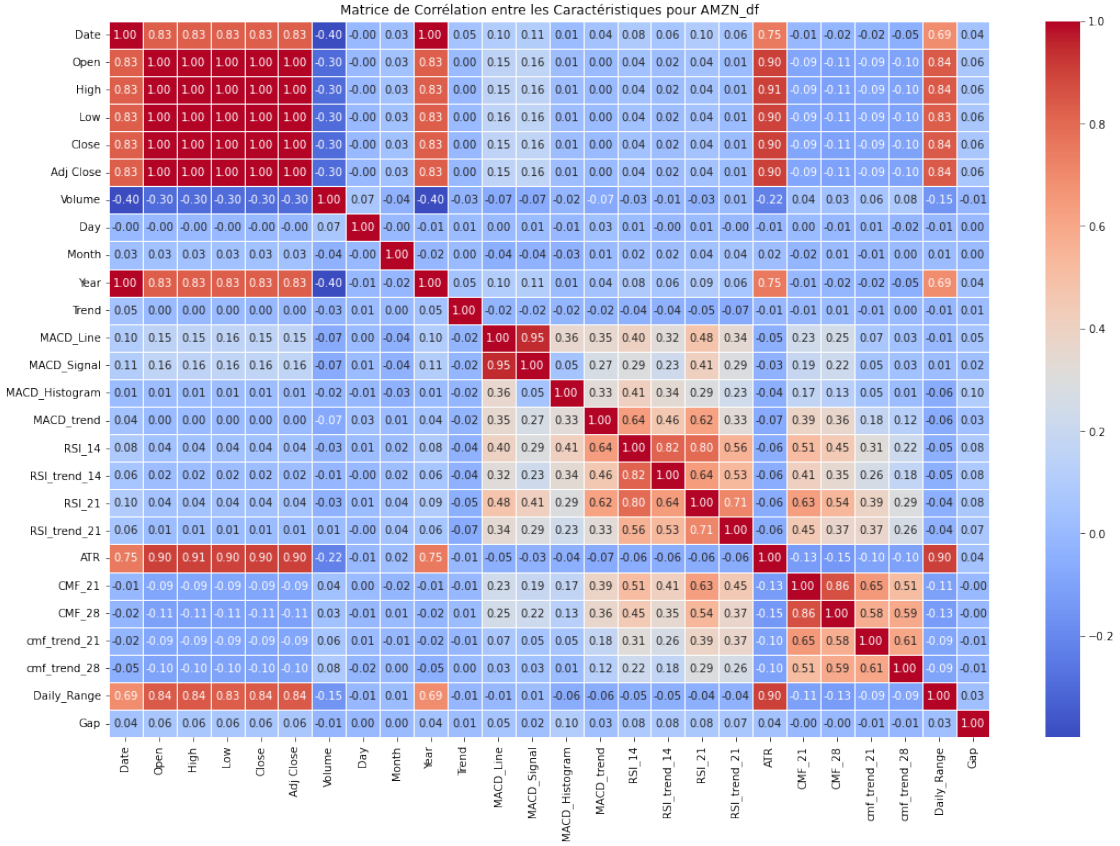
```
…          …  …    …   …  …            …         …           …
6021   23118900    7     12  2023  …             0  56.551815             0
6022   20144800    8     12  2023  …             0  56.882562             0
6023   27708800   11     12  2023  …             0  56.585959             0
6024   24838300   12     12  2023  …             0  53.154714             0
6025   30955500   13     12  2023  …             0  55.364865             0

            ATR    CMF_21    CMF_28  cmf_trend_21  cmf_trend_28  Daily_Range  \
27     2.058036  0.045718  0.014677             0             0     2.031250
28     2.129464  0.002176 -0.044101             0             0     2.875000
29     2.071429 -0.078925 -0.044173             0             0     1.343750
30     1.982143 -0.138220 -0.074553             0             0     0.937500
31     1.950893 -0.202426 -0.085955             0             0     1.531250
…           …         …         …            …             …           …
6021   6.626428  0.238533  0.304441             0             1     5.130005
6022   6.416426  0.246111  0.317292             0             1     6.229980
6023   6.352855  0.325731  0.331306             1             1     5.500000
6024   6.184285  0.325746  0.340839             1             1     3.960022
6025   6.195001  0.335786  0.315424             1             1     6.870026

            Gap
27    -0.054687
28    -0.562500
29     0.648438
30     0.062500
31     0.343750
…           …
6021  -0.569977
6022  -1.750000
6023  -5.750000
6024  -0.449982
6025   1.639984

[5999 rows x 26 columns]
```

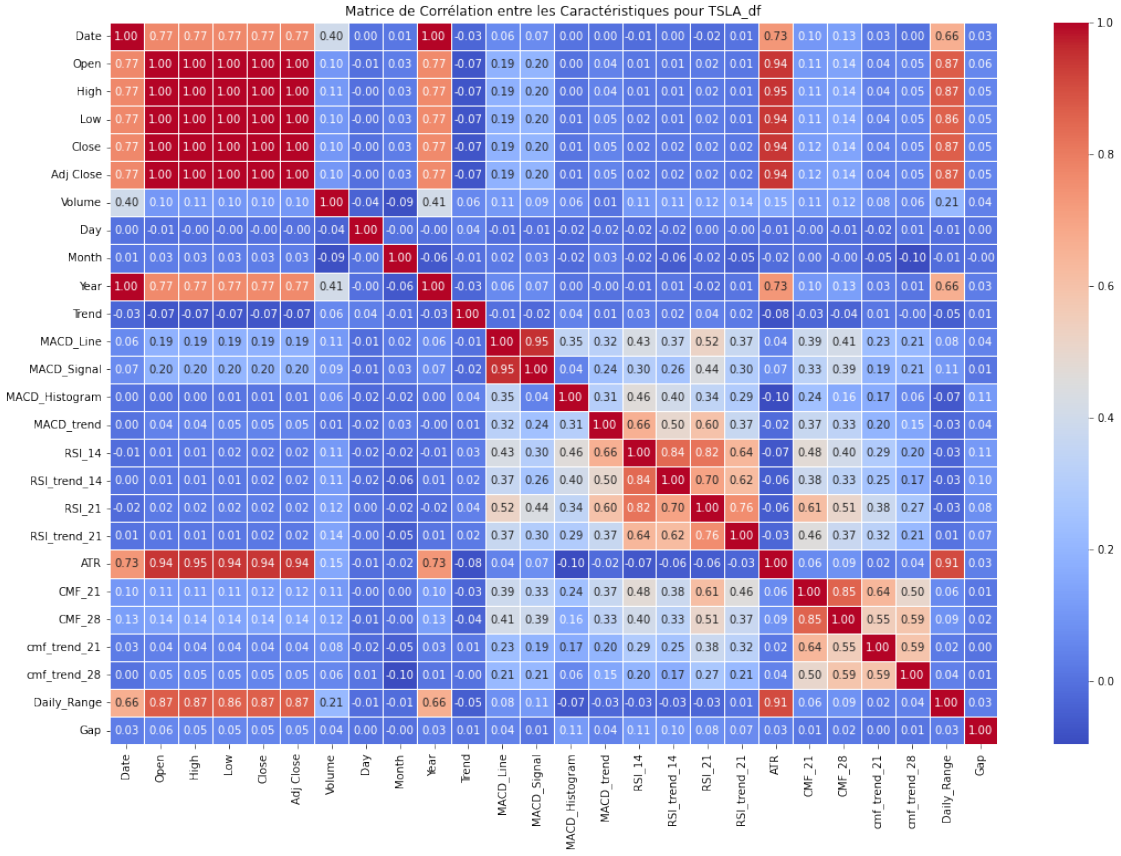
0.3.1 Observation des corrélations

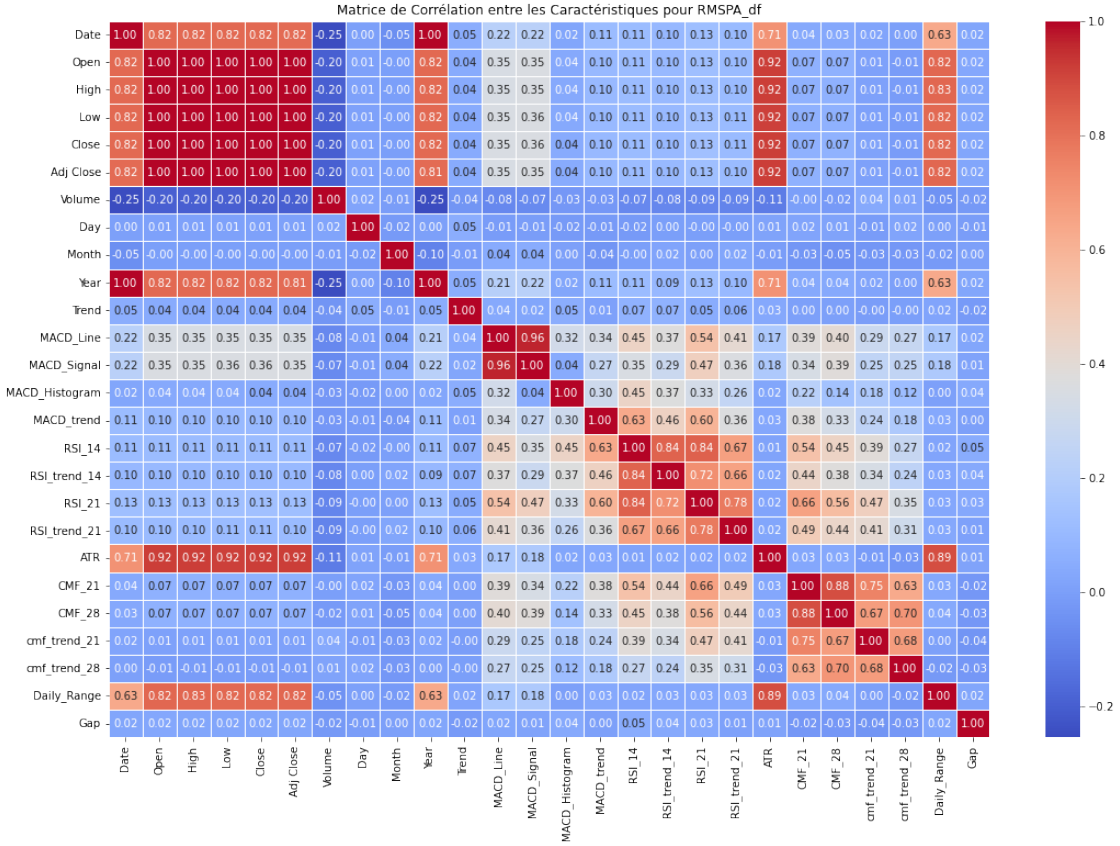
[15]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

for key, df in list_choice.items():
    # Calculer la matrice de corrélation
    correlation_matrix = df.corr()

    # Afficher le heatmap de la matrice de corrélation
    plt.figure(figsize=(18, 12))
```

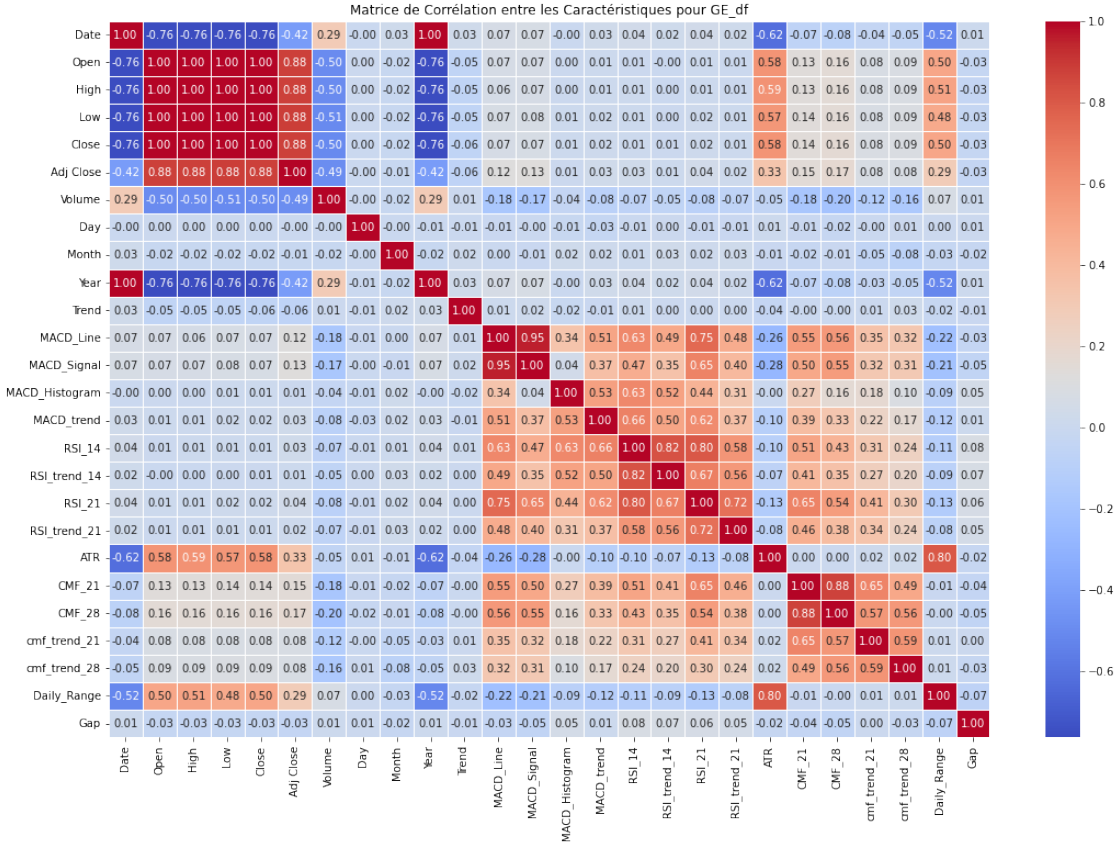
```
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
↪linewidths=.5)
    plt.title(f"Matrice de Corrélation entre les Caractéristiques pour {key}") ␣
↪# Utilisez f-string pour incorporer la variable 'key'
    plt.show()
```

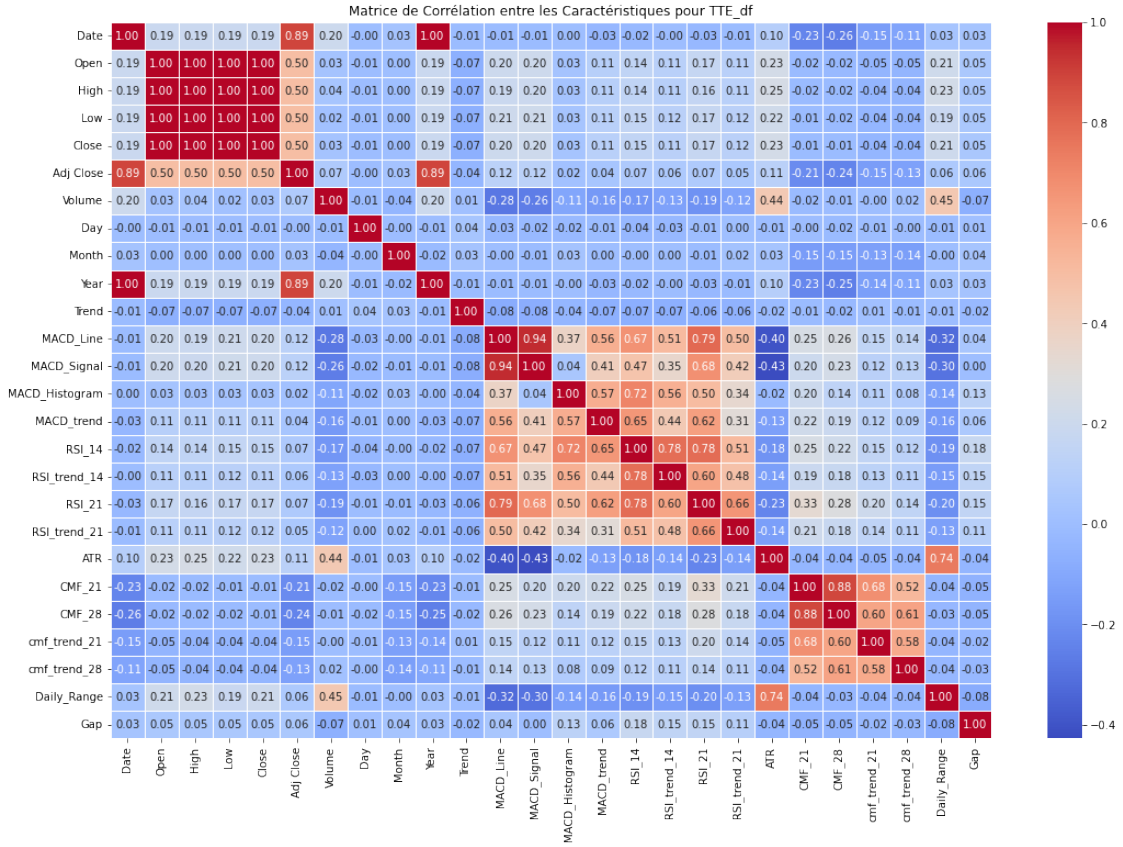



Matrice de Corrélation entre les Caractéristiques pour AMZN_df
Date 1.00 0.83 0.83 0.83 0.83 0.83 -0.40 -0.00 0.03 1.00 0.05 0.10 0.11 0.01 0.04 0.08 0.06 0.10 0.06 0.75 -0.01 -0.02 -0.02 -0.05 0.69 0.04
Open 0.83 1.00 1.00 1.00 1.00 1.00 -0.30 -0.00 0.03 0.83 0.00 0.15 0.16 0.01 0.00 0.04 0.02 0.04 0.01 0.90 -0.09 -0.11 -0.09 -0.10 0.84 0.06
High 0.83 1.00 1.00 1.00 1.00 1.00 -0.30 -0.00 0.03 0.83 0.00 0.15 0.16 0.01 0.00 0.04 0.02 0.04 0.01 0.91 -0.09 -0.11 -0.09 -0.10 0.84 0.06
Low 0.83 1.00 1.00 1.00 1.00 1.00 -0.30 -0.00 0.03 0.83 0.00 0.16 0.16 0.01 0.00 0.04 0.02 0.04 0.01 0.90 -0.09 -0.11 -0.09 -0.10 0.83 0.06
Close 0.83 1.00 1.00 1.00 1.00 1.00 -0.30 -0.00 0.03 0.83 0.00 0.15 0.16 0.01 0.00 0.04 0.02 0.04 0.01 0.90 -0.09 -0.11 -0.09 -0.10 0.84 0.06
Adj Close 0.83 1.00 1.00 1.00 1.00 1.00 -0.30 -0.00 0.03 0.83 0.00 0.15 0.16 0.01 0.00 0.04 0.02 0.04 0.01 0.90 -0.09 -0.11 -0.09 -0.10 0.84 0.06
Volume -0.40 -0.30 -0.30 -0.30 -0.30 -0.30 1.00 0.07 -0.04 -0.40 -0.03 -0.07 -0.07 -0.02 -0.07 -0.03 -0.01 -0.03 0.01 -0.22 0.04 0.03 0.06 0.08 -0.15 -0.01
Day -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 0.07 1.00 -0.00 -0.01 0.01 0.00 0.01 -0.01 0.03 0.01 -0.00 0.01 -0.00 -0.01 0.00 -0.01 0.01 -0.02 -0.01 0.00
Month 0.03 0.03 0.03 0.03 0.03 0.03 -0.04 -0.00 1.00 -0.02 0.00 -0.04 -0.04 -0.03 0.01 0.02 0.02 0.04 0.04 0.02 -0.02 0.01 -0.01 0.00 0.01 0.00
Year 1.00 0.83 0.83 0.83 0.83 0.83 -0.40 -0.01 -0.02 1.00 0.05 0.10 0.11 0.01 0.04 0.08 0.06 0.09 0.06 0.75 -0.01 -0.02 -0.02 -0.05 0.69 0.04
Trend 0.05 0.00 0.00 0.00 0.00 0.00 -0.03 0.01 0.00 0.05 1.00 -0.02 -0.02 -0.02 -0.02 -0.04 -0.04 -0.05 -0.07 -0.01 -0.01 0.01 -0.01 0.00 -0.01 0.01
MACD_Line 0.10 0.15 0.15 0.16 0.15 0.15 -0.07 0.00 -0.04 0.10 -0.02 1.00 0.95 0.36 0.35 0.40 0.32 0.48 0.34 -0.05 0.23 0.25 0.07 0.03 -0.01 0.05
MACD_Signal 0.11 0.16 0.16 0.16 0.16 0.16 -0.07 0.01 -0.04 0.11 -0.02 0.95 1.00 0.05 0.27 0.29 0.23 0.41 0.29 -0.03 0.19 0.22 0.05 0.03 0.01 0.02
MACD_Histogram 0.01 0.01 0.01 0.01 0.01 0.01 -0.02 -0.01 -0.03 0.01 -0.02 0.36 0.05 1.00 0.33 0.41 0.34 0.29 0.23 -0.04 0.17 0.13 0.05 0.01 -0.06 0.10
MACD_trend 0.04 0.00 0.00 0.00 0.00 0.00 -0.07 0.03 0.01 0.04 -0.02 0.35 0.27 0.33 1.00 0.64 0.46 0.62 0.33 -0.07 0.39 0.36 0.18 0.12 -0.06 0.03
RSI_14 0.08 0.04 0.04 0.04 0.04 0.04 -0.03 0.01 0.02 0.08 -0.04 0.40 0.29 0.41 0.64 1.00 0.82 0.80 0.56 -0.06 0.51 0.45 0.31 0.22 -0.05 0.08
RSI_trend_14 0.06 0.02 0.02 0.02 0.02 0.02 -0.01 -0.00 0.02 0.06 -0.04 0.32 0.23 0.34 0.46 0.82 1.00 0.64 0.53 -0.06 0.41 0.35 0.26 0.18 -0.05 0.08
RSI_21 0.10 0.04 0.04 0.04 0.04 0.04 -0.03 0.01 0.04 0.09 -0.05 0.48 0.41 0.29 0.62 0.80 0.64 1.00 0.71 -0.06 0.63 0.54 0.39 0.29 -0.04 0.08
RSI_trend_21 0.06 0.01 0.01 0.01 0.01 0.01 0.01 -0.00 0.04 0.06 -0.07 0.34 0.29 0.23 0.33 0.56 0.53 0.71 1.00 -0.06 0.45 0.37 0.37 0.26 -0.04 0.07
ATR 0.75 0.90 0.91 0.90 0.90 0.90 -0.22 -0.01 0.02 0.75 -0.01 -0.05 -0.03 -0.04 -0.07 -0.06 -0.06 -0.06 -0.06 1.00 -0.13 -0.15 -0.10 -0.10 0.90 0.04
CMF_21 -0.01 -0.09 -0.09 -0.09 -0.09 -0.09 0.04 0.00 -0.02 -0.01 -0.01 0.23 0.19 0.17 0.39 0.51 0.41 0.63 0.45 -0.13 1.00 0.86 0.65 0.51 -0.11 -0.00
CMF_28 -0.02 -0.11 -0.11 -0.11 -0.11 -0.11 0.03 -0.01 0.01 -0.02 0.01 0.25 0.22 0.13 0.36 0.45 0.35 0.54 0.37 -0.15 0.86 1.00 0.58 0.59 -0.13 -0.00
cmf_trend_21 -0.02 -0.09 -0.09 -0.09 -0.09 -0.09 0.06 0.01 -0.01 -0.02 -0.01 0.07 0.05 0.05 0.18 0.31 0.26 0.39 0.37 -0.10 0.65 0.58 1.00 0.61 -0.09 -0.01
cmf_trend_28 -0.05 -0.10 -0.10 -0.10 -0.10 -0.10 0.08 -0.02 0.00 -0.05 0.00 0.03 0.03 0.01 0.12 0.22 0.18 0.29 0.26 -0.10 0.51 0.59 0.61 1.00 -0.09 -0.01
Daily_Range 0.69 0.84 0.84 0.83 0.84 0.84 -0.15 -0.01 0.01 0.69 -0.01 -0.01 0.01 -0.06 -0.06 -0.05 -0.05 -0.04 -0.04 0.90 -0.11 -0.13 -0.09 -0.09 1.00 0.03
Gap 0.04 0.06 0.06 0.06 0.06 0.06 -0.01 0.00 0.00 0.04 0.01 0.05 0.02 0.10 0.03 0.08 0.08 0.08 0.07 0.04 -0.00 -0.00 -0.01 -0.01 0.03 1.00
Date Open High Low Close Adj Close Volume Day Month Year Trend MACD_Line MACD_Signal MACD_Histogram MACD_trend RSI_14 RSI_trend_14 RSI_21 RSI_trend_21 ATR CMF_21 CMF_28 cmf_trend_21 cmf_trend_28 Daily_Range Gap
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


Matrice de Corrélation entre les Caractéristiques pour TSLA_df
Date 1.00 0.77 0.77 0.77 0.77 0.77 0.40 0.00 0.01 1.00 -0.03 0.06 0.07 0.00 0.00 -0.01 0.00 -0.02 0.01 0.73 0.10 0.13 0.03 0.00 0.66 0.03
Open 0.77 1.00 1.00 1.00 1.00 1.00 0.10 -0.01 0.03 0.77 -0.07 0.19 0.20 0.00 0.04 0.01 0.01 0.02 0.01 0.94 0.11 0.14 0.04 0.05 0.87 0.06
High 0.77 1.00 1.00 1.00 1.00 1.00 0.11 -0.00 0.03 0.77 -0.07 0.19 0.20 0.00 0.04 0.01 0.01 0.02 0.01 0.95 0.11 0.14 0.04 0.05 0.87 0.05
Low 0.77 1.00 1.00 1.00 1.00 1.00 0.10 -0.00 0.03 0.77 -0.07 0.19 0.20 0.01 0.05 0.02 0.01 0.02 0.01 0.94 0.11 0.14 0.04 0.05 0.86 0.05
Close 0.77 1.00 1.00 1.00 1.00 1.00 0.10 -0.00 0.03 0.77 -0.07 0.19 0.20 0.01 0.05 0.02 0.02 0.02 0.02 0.94 0.12 0.14 0.04 0.05 0.87 0.05
Adj Close 0.77 1.00 1.00 1.00 1.00 1.00 0.10 -0.00 0.03 0.77 -0.07 0.19 0.20 0.01 0.05 0.02 0.02 0.02 0.02 0.94 0.12 0.14 0.04 0.05 0.87 0.05
Volume 0.40 0.10 0.11 0.10 0.10 0.10 1.00 -0.04 -0.09 0.41 0.06 0.11 0.09 0.06 0.01 0.11 0.11 0.12 0.14 0.15 0.11 0.12 0.08 0.06 0.21 0.04
Day 0.00 -0.01 -0.00 -0.00 -0.00 -0.00 -0.04 1.00 -0.00 -0.00 0.04 -0.01 -0.01 -0.02 -0.02 -0.02 -0.02 0.00 -0.00 -0.01 -0.00 -0.01 -0.02 0.01 -0.01 0.00
Month 0.01 0.03 0.03 0.03 0.03 0.03 -0.09 -0.00 1.00 -0.06 -0.01 0.02 0.03 -0.02 0.03 -0.02 -0.06 -0.02 -0.05 -0.02 0.00 -0.00 -0.05 -0.10 -0.01 -0.00
Year 1.00 0.77 0.77 0.77 0.77 0.77 0.41 -0.00 -0.06 1.00 -0.03 0.06 0.07 0.00 -0.00 -0.01 0.01 -0.02 0.01 0.73 0.10 0.13 0.03 0.01 0.66 0.03
Trend -0.03 -0.07 -0.07 -0.07 -0.07 -0.07 0.06 0.04 -0.01 -0.03 1.00 -0.01 -0.02 0.04 0.01 0.03 0.02 0.04 0.02 -0.08 -0.03 -0.04 0.01 -0.00 -0.05 0.01
MACD_Line 0.06 0.19 0.19 0.19 0.19 0.19 0.11 -0.01 0.02 0.06 -0.01 1.00 0.95 0.35 0.32 0.43 0.37 0.52 0.37 0.04 0.39 0.41 0.23 0.21 0.08 0.04
MACD_Signal 0.07 0.20 0.20 0.20 0.20 0.20 0.09 -0.01 0.03 0.07 -0.02 0.95 1.00 0.04 0.24 0.30 0.26 0.44 0.30 0.07 0.33 0.39 0.19 0.21 0.11 0.01
MACD_Histogram 0.00 0.00 0.00 0.01 0.01 0.01 0.06 -0.02 -0.02 0.00 0.04 0.35 0.04 1.00 0.31 0.46 0.40 0.34 0.29 -0.10 0.24 0.16 0.17 0.06 -0.07 0.11
MACD_trend 0.00 0.04 0.04 0.05 0.05 0.05 0.01 -0.02 0.03 -0.00 0.01 0.32 0.24 0.31 1.00 0.66 0.50 0.60 0.37 -0.02 0.37 0.33 0.20 0.15 -0.03 0.04
RSI_14 -0.01 0.01 0.01 0.02 0.02 0.02 0.11 -0.02 -0.02 -0.01 0.03 0.43 0.30 0.46 0.66 1.00 0.84 0.82 0.64 -0.07 0.48 0.40 0.29 0.20 -0.03 0.11
RSI_trend_14 0.00 0.01 0.01 0.01 0.02 0.02 0.11 -0.02 -0.06 0.01 0.02 0.37 0.26 0.40 0.50 0.84 1.00 0.70 0.62 -0.06 0.38 0.33 0.25 0.17 -0.03 0.10
RSI_21 -0.02 0.02 0.02 0.02 0.02 0.02 0.12 0.00 -0.02 -0.02 0.04 0.52 0.44 0.34 0.60 0.82 0.70 1.00 0.76 -0.06 0.61 0.51 0.38 0.27 -0.03 0.08
RSI_trend_21 0.01 0.01 0.01 0.01 0.02 0.02 0.14 -0.00 -0.05 0.01 0.02 0.37 0.30 0.29 0.37 0.64 0.62 0.76 1.00 -0.03 0.46 0.37 0.32 0.21 0.01 0.07
ATR 0.73 0.94 0.95 0.94 0.94 0.94 0.15 -0.01 -0.02 0.73 -0.08 0.04 0.07 -0.10 -0.02 -0.07 -0.06 -0.06 -0.03 1.00 0.06 0.09 0.02 0.04 0.91 0.03
CMF_21 0.10 0.11 0.11 0.11 0.12 0.12 0.11 -0.00 0.00 0.10 -0.03 0.39 0.33 0.24 0.37 0.48 0.38 0.61 0.46 0.06 1.00 0.85 0.64 0.50 0.06 0.01
CMF_28 0.13 0.14 0.14 0.14 0.14 0.14 0.12 -0.01 -0.00 0.13 -0.04 0.41 0.39 0.16 0.33 0.40 0.33 0.51 0.37 0.09 0.85 1.00 0.55 0.59 0.09 0.02
cmf_trend_21 0.03 0.04 0.04 0.04 0.04 0.04 0.08 -0.02 -0.05 0.03 0.01 0.23 0.19 0.17 0.20 0.29 0.25 0.38 0.32 0.02 0.64 0.55 1.00 0.59 0.02 0.00
cmf_trend_28 0.00 0.05 0.05 0.05 0.05 0.05 0.06 0.01 -0.10 0.01 -0.00 0.21 0.21 0.06 0.15 0.20 0.17 0.27 0.21 0.04 0.50 0.59 0.59 1.00 0.04 0.01
Daily_Range 0.66 0.87 0.87 0.86 0.87 0.87 0.21 -0.01 -0.01 0.66 -0.05 0.08 0.11 -0.07 -0.03 -0.03 -0.03 -0.03 0.01 0.91 0.06 0.09 0.02 0.04 1.00 0.03
Gap 0.03 0.06 0.05 0.05 0.05 0.05 0.04 0.00 -0.00 0.03 0.01 0.04 0.01 0.11 0.04 0.11 0.10 0.08 0.07 0.03 0.01 0.02 0.00 0.01 0.03 1.00
Date Open High Low Close Adj Close Volume Day Month Year Trend MACD_Line MACD_Signal MACD_Histogram MACD_trend RSI_14 RSI_trend_14 RSI_21 RSI_trend_21 ATR CMF_21 CMF_28 cmf_trend_21 cmf_trend_28 Daily_Range Gap
1.0
0.8
0.6
0.4
0.2
0.0


Matrice de Corrélation entre les Caractéristiques pour RMSPA_df
Date 1.00 0.82 0.82 0.82 0.82 0.82 -0.25 0.00 -0.05 1.00 0.05 0.22 0.22 0.02 0.11 0.11 0.10 0.13 0.10 0.71 0.04 0.03 0.02 0.00 0.63 0.02
Open 0.82 1.00 1.00 1.00 1.00 1.00 -0.20 0.01 -0.00 0.82 0.04 0.35 0.35 0.04 0.10 0.11 0.10 0.13 0.10 0.92 0.07 0.07 0.01 -0.01 0.82 0.02
High 0.82 1.00 1.00 1.00 1.00 1.00 -0.20 0.01 -0.00 0.82 0.04 0.35 0.35 0.04 0.10 0.11 0.10 0.13 0.10 0.92 0.07 0.07 0.01 -0.01 0.83 0.02
Low 0.82 1.00 1.00 1.00 1.00 1.00 -0.20 0.01 -0.00 0.82 0.04 0.35 0.36 0.04 0.10 0.11 0.10 0.13 0.11 0.92 0.07 0.07 0.01 -0.01 0.82 0.02
Close 0.82 1.00 1.00 1.00 1.00 1.00 -0.20 0.01 -0.00 0.82 0.04 0.35 0.36 0.04 0.10 0.11 0.10 0.13 0.11 0.92 0.07 0.07 0.01 -0.01 0.82 0.02
Adj Close 0.82 1.00 1.00 1.00 1.00 1.00 -0.20 0.01 -0.00 0.81 0.04 0.35 0.35 0.04 0.10 0.11 0.10 0.13 0.10 0.92 0.07 0.07 0.01 -0.01 0.82 0.02
Volume -0.25 -0.20 -0.20 -0.20 -0.20 -0.20 1.00 0.02 -0.01 -0.25 -0.04 -0.08 -0.07 -0.03 -0.03 -0.07 -0.08 -0.09 -0.09 -0.11 -0.00 -0.02 0.04 0.01 -0.05 -0.02
Day 0.00 0.01 0.01 0.01 0.01 0.01 0.02 1.00 -0.02 0.00 0.05 -0.01 -0.01 -0.02 -0.01 -0.02 0.00 -0.00 -0.00 0.01 0.02 0.01 -0.01 0.02 0.00 -0.01
Month -0.05 -0.00 -0.00 -0.00 -0.00 -0.00 -0.01 -0.02 1.00 -0.10 -0.01 0.04 0.04 0.00 -0.04 -0.00 0.02 0.00 0.02 -0.01 -0.03 -0.05 -0.03 -0.03 -0.02 0.00
Year 1.00 0.82 0.82 0.82 0.82 0.81 -0.25 0.00 -0.10 1.00 0.05 0.21 0.22 0.02 0.11 0.11 0.09 0.13 0.10 0.71 0.04 0.04 0.02 0.00 0.63 0.02
Trend 0.05 0.04 0.04 0.04 0.04 0.04 -0.04 0.05 -0.01 0.05 1.00 0.04 0.02 0.05 0.01 0.07 0.07 0.05 0.06 0.03 0.00 0.00 -0.00 -0.00 0.02 -0.02
MACD_Line 0.22 0.35 0.35 0.35 0.35 0.35 -0.08 -0.01 0.04 0.21 0.04 1.00 0.96 0.32 0.34 0.45 0.37 0.54 0.41 0.17 0.39 0.40 0.29 0.27 0.17 0.02
MACD_Signal 0.22 0.35 0.35 0.36 0.36 0.35 -0.07 -0.01 0.04 0.22 0.02 0.96 1.00 0.04 0.27 0.35 0.29 0.47 0.36 0.18 0.34 0.39 0.25 0.25 0.18 0.01
MACD_Histogram 0.02 0.04 0.04 0.04 0.04 0.04 -0.03 -0.02 0.00 0.02 0.05 0.32 0.04 1.00 0.30 0.45 0.37 0.33 0.26 0.02 0.22 0.14 0.18 0.12 0.00 0.04
MACD_trend 0.11 0.10 0.10 0.10 0.10 0.10 -0.03 -0.01 -0.04 0.11 0.01 0.34 0.27 0.30 1.00 0.63 0.46 0.60 0.36 0.03 0.38 0.33 0.24 0.18 0.03 0.00
RSI_14 0.11 0.11 0.11 0.11 0.11 0.11 -0.07 -0.02 -0.00 0.11 0.07 0.45 0.35 0.45 0.63 1.00 0.84 0.84 0.67 0.01 0.54 0.45 0.39 0.27 0.02 0.05
RSI_trend_14 0.10 0.10 0.10 0.10 0.10 0.10 -0.08 0.00 0.02 0.09 0.07 0.37 0.29 0.37 0.46 0.84 1.00 0.72 0.66 0.02 0.44 0.38 0.34 0.24 0.03 0.04
RSI_21 0.13 0.13 0.13 0.13 0.13 0.13 -0.09 -0.00 0.00 0.13 0.05 0.54 0.47 0.33 0.60 0.84 0.72 1.00 0.78 0.02 0.66 0.56 0.47 0.35 0.03 0.03
RSI_trend_21 0.10 0.10 0.10 0.11 0.11 0.10 -0.09 -0.00 0.02 0.10 0.06 0.41 0.36 0.26 0.36 0.67 0.66 0.78 1.00 0.02 0.49 0.44 0.41 0.31 0.03 0.01
ATR 0.71 0.92 0.92 0.92 0.92 0.92 -0.11 0.01 -0.01 0.71 0.03 0.17 0.18 0.02 0.03 0.01 0.02 0.02 0.02 1.00 0.03 0.03 -0.01 -0.03 0.89 0.01
CMF_21 0.04 0.07 0.07 0.07 0.07 0.07 -0.00 0.02 -0.03 0.04 0.00 0.39 0.34 0.22 0.38 0.54 0.44 0.66 0.49 0.03 1.00 0.88 0.75 0.63 0.03 -0.02
CMF_28 0.03 0.07 0.07 0.07 0.07 0.07 -0.02 0.01 -0.05 0.04 0.00 0.40 0.39 0.14 0.33 0.45 0.38 0.56 0.44 0.03 0.88 1.00 0.67 0.70 0.04 -0.03
cmf_trend_21 0.02 0.01 0.01 0.01 0.01 0.01 0.04 -0.01 -0.03 0.02 -0.00 0.29 0.25 0.18 0.24 0.39 0.34 0.47 0.41 -0.01 0.75 0.67 1.00 0.68 0.00 -0.04
cmf_trend_28 0.00 -0.01 -0.01 -0.01 -0.01 -0.01 0.01 0.02 -0.03 0.00 -0.00 0.27 0.25 0.12 0.18 0.27 0.24 0.35 0.31 -0.03 0.63 0.70 0.68 1.00 -0.02 -0.03
Daily_Range 0.63 0.82 0.83 0.82 0.82 0.82 -0.05 0.00 -0.02 0.63 0.02 0.17 0.18 0.00 0.03 0.02 0.03 0.03 0.03 0.89 0.03 0.04 0.00 -0.02 1.00 0.02
Gap 0.02 0.02 0.02 0.02 0.02 0.02 -0.02 -0.01 0.00 0.02 -0.02 0.02 0.01 0.04 0.00 0.05 0.04 0.03 0.01 0.01 -0.02 -0.03 -0.04 -0.03 0.02 1.00
Date Open High Low Close Adj Close Volume Day Month Year Trend MACD_Line MACD_Signal MACD_Histogram MACD_trend RSI_14 RSI_trend_14 RSI_21 RSI_trend_21 ATR CMF_21 CMF_28 cmf_trend_21 cmf_trend_28 Daily_Range Gap
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


Matrice de Corrélation entre les Caractéristiques pour GE_df
Date
Open
High
Low
Close
Adj Close
Volume
Day
Month
Year
Trend
MACD_Line
MACD_Signal
MACD_Histogram
MACD_trend
RSI_14
RSI_trend_14
RSI_21
RSI_trend_21
ATR
CMF_21
CMF_28
cmf_trend_21
cmf_trend_28
Daily_Range
Gap
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6



On peut voir une corrélation parfaite (100%) entre les variables High, Open, Low, Close et Adjusted Close. J'ai opté pour conserver uniquement la variable Close, étant donné qu'elle est spécifiquement utilisée dans le calcul de la tendance.

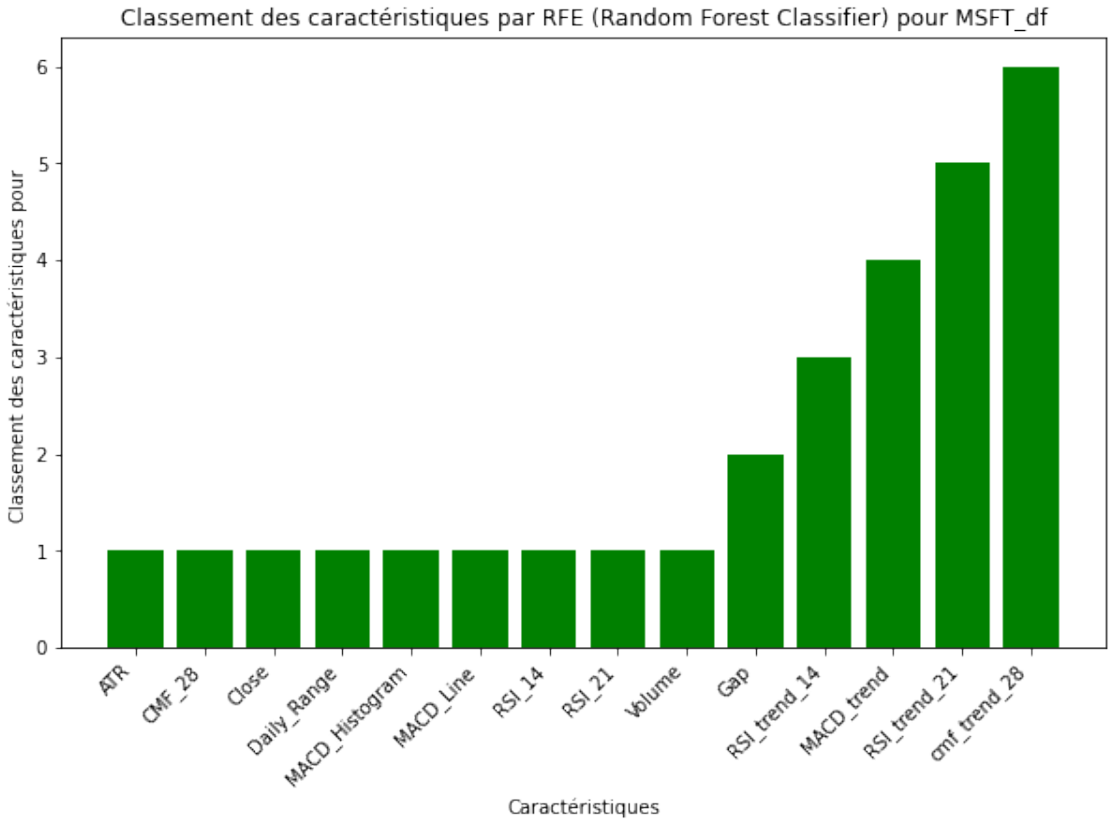
Analyse de l'indicateur MACD : La variable MACD_Line a été retenue, et MACD_Signal a été éliminée en raison de la corrélation presque parfaite entre les deux, évitant ainsi la redondance d'informations.

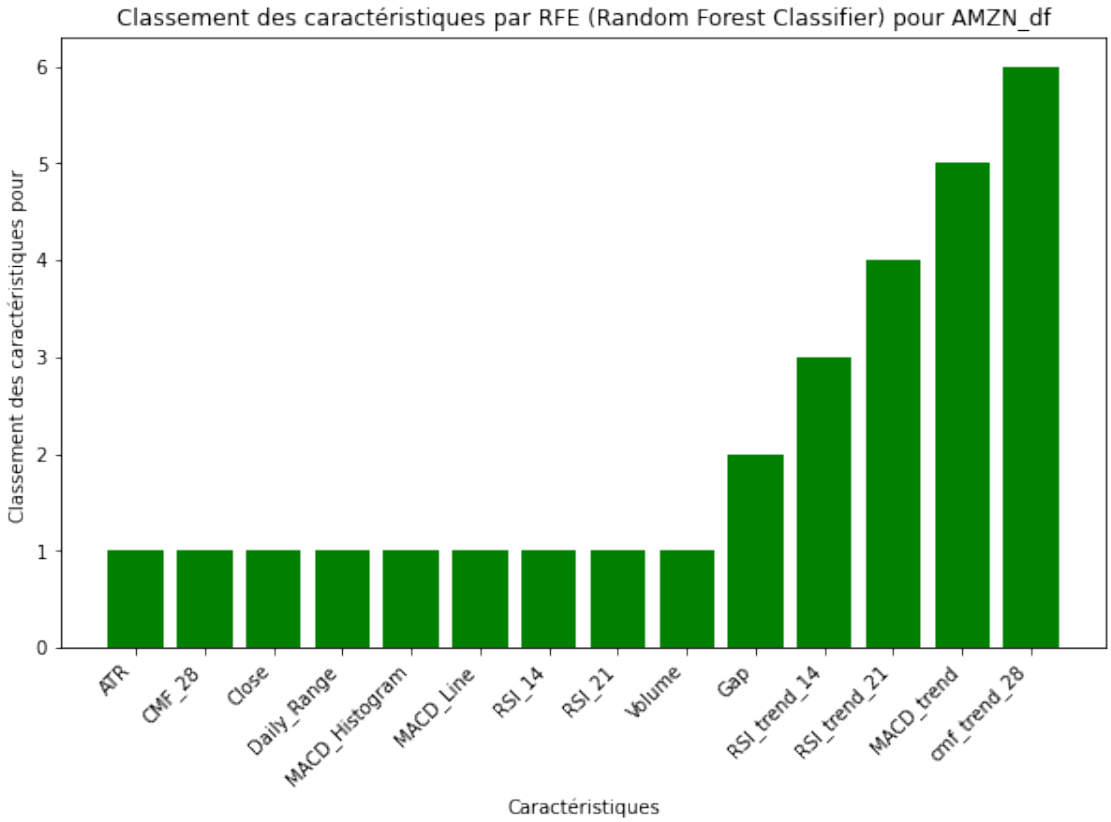
Choix entre CMF_21 et CMF_28 : Les variables CMF_21 et CMF_28 présentaient une corrélation. Conformément à la possibilité mentionnée dans l'énoncé de choisir l'une des deux, CMF_28 a été sélectionnée pour éliminer la redondance d'informations et simplifier le modèle.

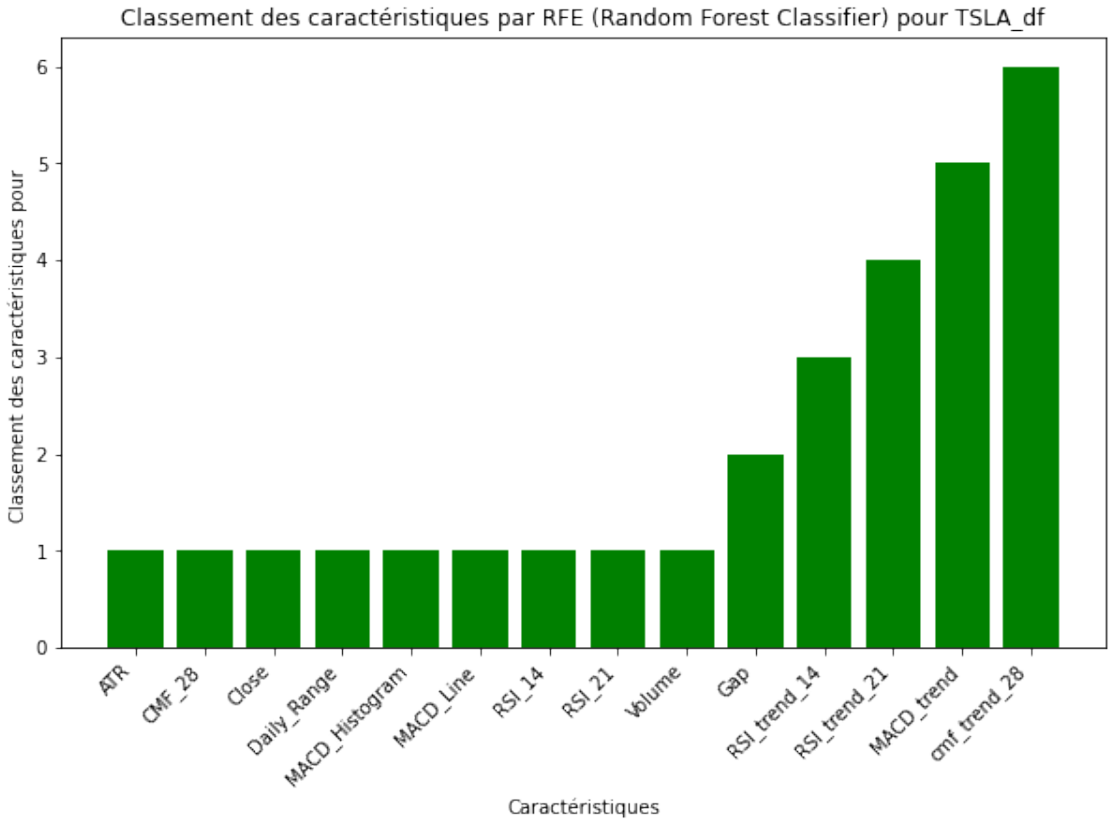
0.3.2 Application de Recursive Feature Elimination (RFE)

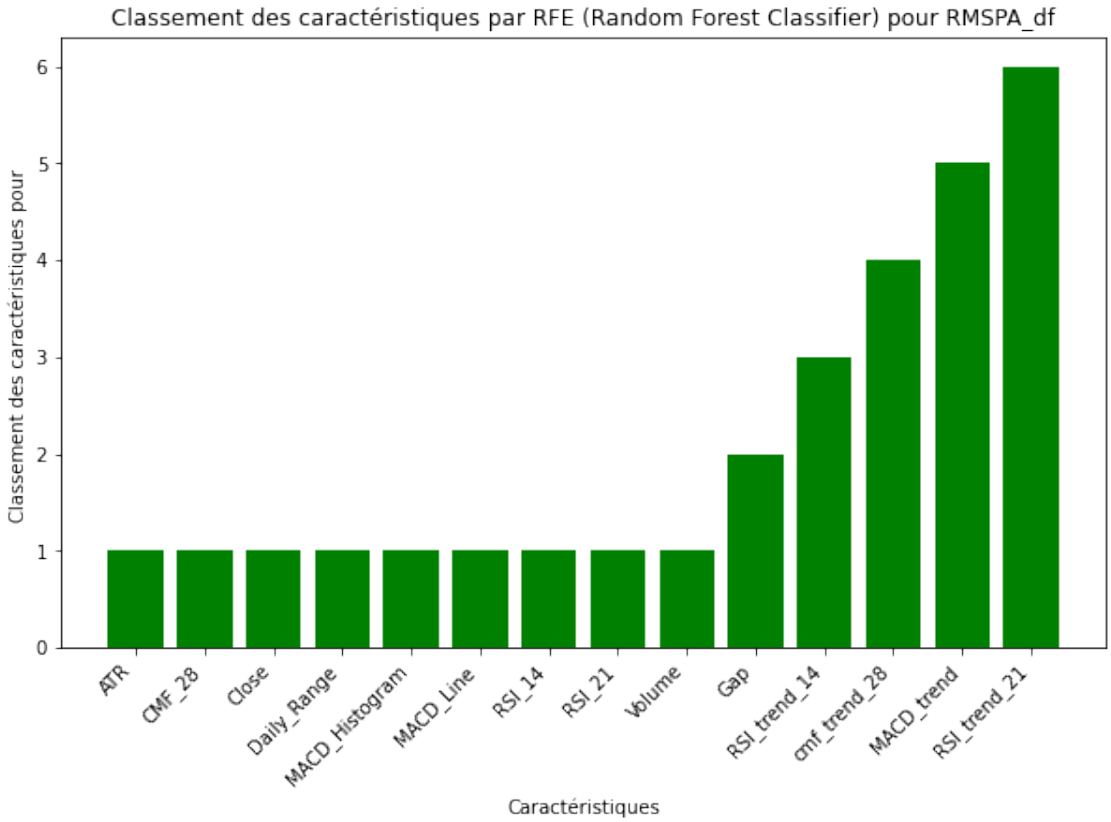
[16]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import pandas as pd

# Supposons que vos données soient dans un DataFrame 'df'
```

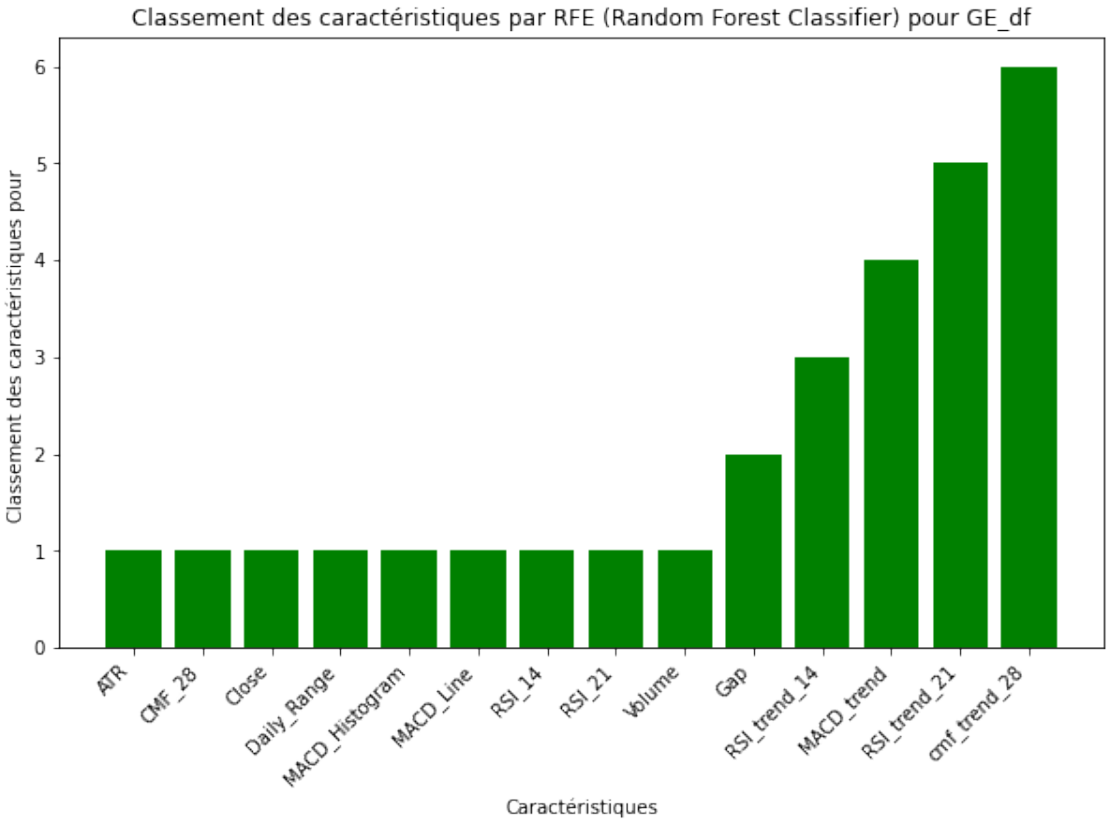
```
# Assurez-vous que 'df' contient toutes les colonnes nécessaires, sauf 'Trend' et 'Date'
for key, df in list_choice.items():
    # Définir x comme le DataFrame sans 'Trend' et 'Date'
    x = df.drop(['Trend', 'Date', 'CMF_21', 'Open', 'High', 'cmf_trend_21', 'Low', 'Adj Close', 'Day', 'Month', 'Year', 'MACD_Signal'], axis=1)

    # Définir y comme la colonne 'Trend'
    y = df['Trend']

    # Créer un objet StandardScaler
    scaler = StandardScaler()

    # Normaliser les données x
    x_scaled = scaler.fit_transform(x)

    # Exemple d'utilisation de RFE
    n_features_to_select = 9
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
    rfe = RFE(estimator=rf_model, n_features_to_select=n_features_to_select)
    rfe.fit(x_scaled, y)

    # Obtenez les caractéristiques sélectionnées et leur classement
    selected_features = x.columns[rfe.support_]
    ranking = rfe.ranking_

    # Trier les caractéristiques en fonction de leur classement
    sorted_features = [col for _, col in sorted(zip(ranking, x.columns))]

    # Trier les valeurs de classement
    sorted_ranking = sorted(ranking)

    # Créer un graphique à barres
    plt.figure(figsize=(10, 6))

    # Utilisez la fonction `bar` pour créer un graphique à barres verticales
    plt.bar(sorted_features, sorted_ranking, color='green')

    # Ajoutez une annotation pour chaque barre avec la valeur de classement
    plt.xlabel('Caractéristiques')
    plt.ylabel('Classement des caractéristiques pour ')
    plt.title('Classement des caractéristiques par RFE (Random Forest Classifier) pour ' + key)
    plt.xticks(rotation=45, ha='right')  # Rotation des étiquettes sur l'axe x pour une meilleure lisibilité
    plt.show()
```

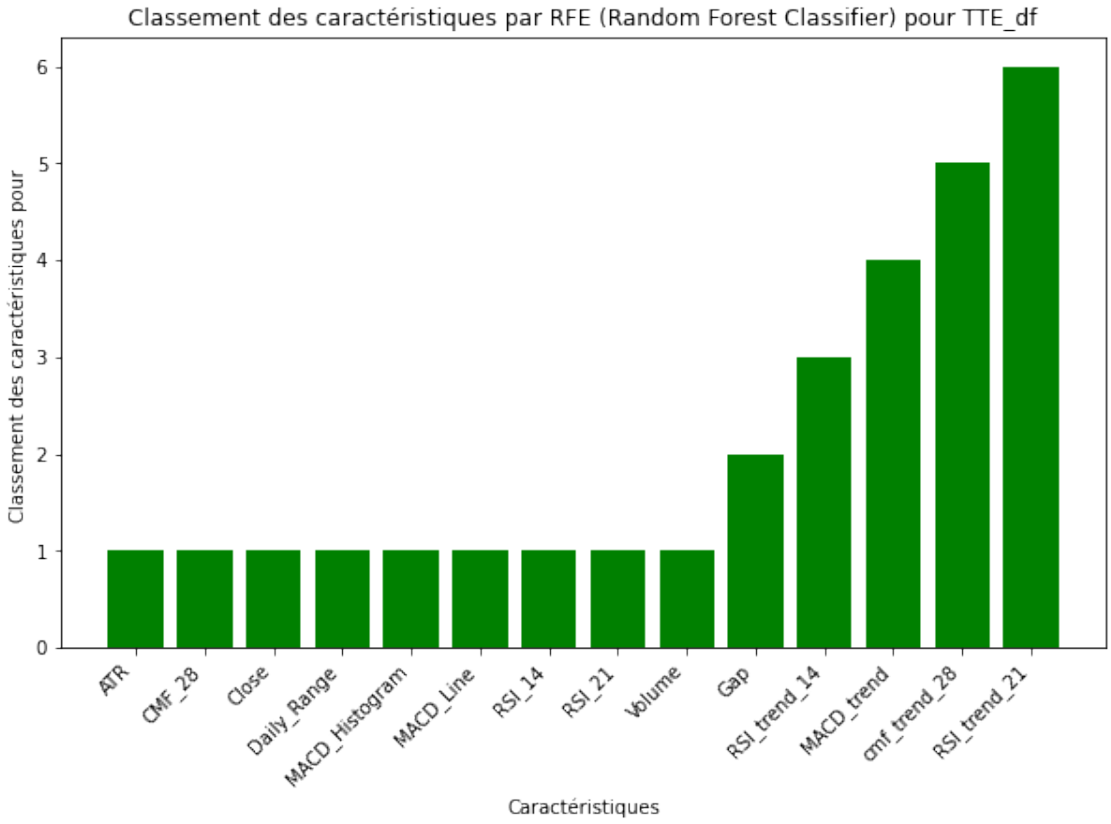

Classement des caractéristiques par RFE (Random Forest Classifier) pour MSFT_df
Classement des caractéristiques pour
0
1
2
3
4
5
6
ATR
CMF_28
Close
Daily_Range
MACD_Histogram
MACD_Line
RSI_14
RSI_21
Volume
Gap
RSI_trend_14
MACD_trend
RSI_trend_21
cmf_trend_28
Caractéristiques


Classement des caractéristiques par RFE (Random Forest Classifier) pour AMZN_df
Classement des caractéristiques pour
0
1
2
3
4
5
6
ATR
CMF_28
Close
Daily_Range
MACD_Histogram
MACD_Line
RSI_14
RSI_21
Volume
Gap
RSI_trend_14
RSI_trend_21
MACD_trend
cmf_trend_28
Caractéristiques


Classement des caractéristiques par RFE (Random Forest Classifier) pour TSLA_df
Classement des caractéristiques pour
0
1
2
3
4
5
6
ATR
CMF_28
Close
Daily_Range
MACD_Histogram
MACD_Line
RSI_14
RSI_21
Volume
Gap
RSI_trend_14
RSI_trend_21
MACD_trend
cmf_trend_28
Caractéristiques


Classement des caractéristiques par RFE (Random Forest Classifier) pour RMSPA_df
Classement des caractéristiques pour
0
1
2
3
4
5
6
ATR
CMF_28
Close
Daily_Range
MACD_Histogram
MACD_Line
RSI_14
RSI_21
Volume
Gap
RSI_trend_14
cmf_trend_28
MACD_trend
RSI_trend_21
Caractéristiques


Classement des caractéristiques par RFE (Random Forest Classifier) pour GE_df
Classement des caractéristiques pour
0
1
2
3
4
5
6
ATR
CMF_28
Close
Daily_Range
MACD_Histogram
MACD_Line
RSI_14
RSI_21
Volume
Gap
RSI_trend_14
MACD_trend
RSI_trend_21
cmf_trend_28
Caractéristiques



D'aprés RFE , nous allons choisir [ATR , CMF_21 , CMF_28 , CLOSE , MACD_Histogram , MACD_Line , MACD_Signal , RSI_21, Volume]

[ ]:

[17]:
```
selected_features = [ 'ATR','CMF_28', 'Close', 'Daily_Range','MACD_Histogram' ,␣
↪'MACD_Line' , 'RSI_14', 'RSI_21', 'Volume']

    # Sélectionnez la colonne 'Trend' comme votre variable cible
target_column = 'Trend'
```

[ ]:

0.4 8 Différentes modélisations de la variable à expliquer (target)

[ ]:

0.4.1 1. Régression logistique :

[18]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,hamming_loss, confusion_matrix,␣
 ↪classification_report
import seaborn as sns
import matplotlib.pyplot as plt
from tabulate import tabulate
from sklearn.metrics import recall_score, f1_score
```

[19]:
```
!pip install prettytable
```

```
Requirement already satisfied: prettytable in c:\users\bouch\anaconda3\lib\site-
packages (3.9.0)
Requirement already satisfied: wcwidth in c:\users\bouch\anaconda3\lib\site-
packages (from prettytable) (0.2.5)
```

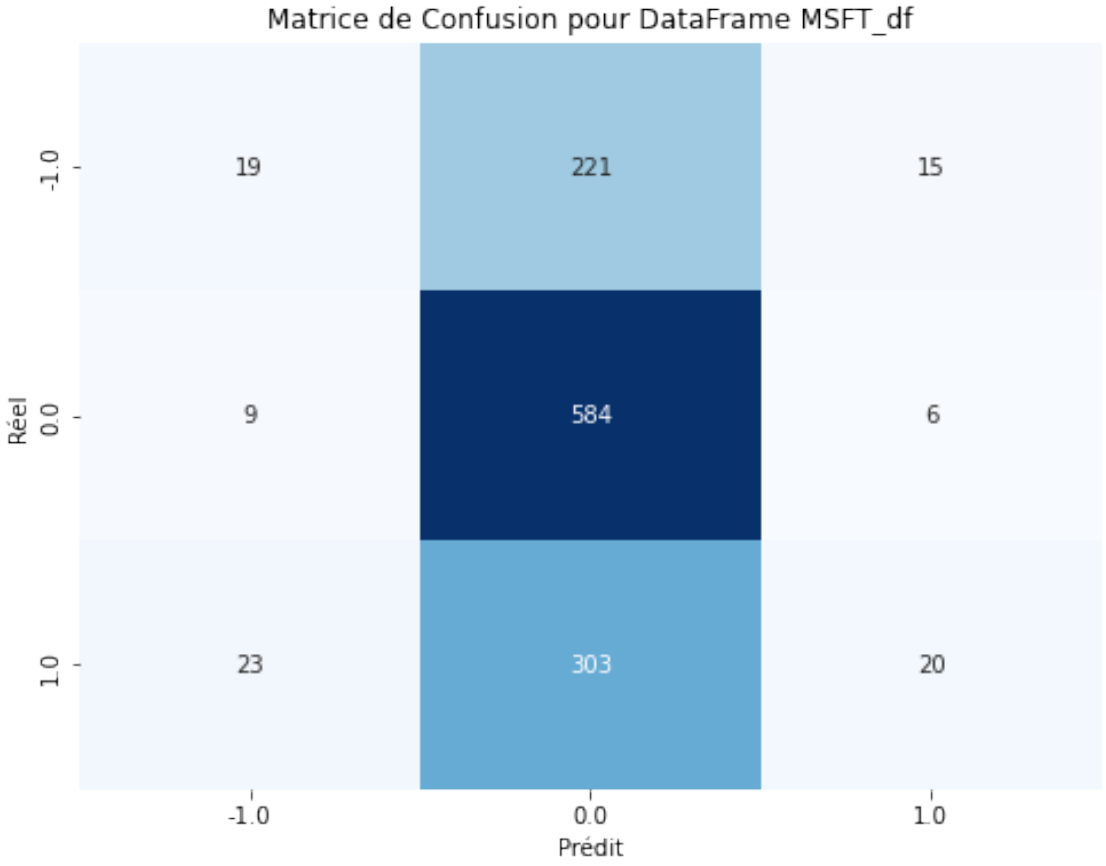
[20]:
```
# Liste pour stocker les résultats de chaque modèle
from prettytable import PrettyTable
results_regression = []

# Parcourir chaque dataframe dans list_choice
for key, df in list_choice.items():


       # Créez un DataFrame avec les colonnes sélectionnées
    data = df[selected_features + [target_column]]
    # Supprimer les lignes avec des valeurs manquantes
    data.dropna(inplace=True)


    # Divisez les données en ensembles d'entraînement et de test
    X_train, X_test, y_train, y_test = train_test_split(data.
 ↪drop(target_column, axis=1), data[target_column], test_size=0.2,␣
 ↪random_state=42)
```

```
    # Standardisez les données
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Entraînez le modèle de régression logistique
    model = LogisticRegression()
    model.fit(X_train_scaled, y_train)

    # Prédiction sur l'ensemble de test
    y_pred = model.predict(X_test_scaled)

    # Évaluation de la performance
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)
    hamming = hamming_loss(y_test, y_pred)
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    micro_recall = recall_score(y_test, y_pred, average='micro')
    micro_f1 = f1_score(y_test, y_pred, average='micro')
    macro_recall = recall_score(y_test, y_pred, average='macro')
    macro_f1 = f1_score(y_test, y_pred, average='macro')

    # Stocker les résultats dans la liste
    results_regression.append({
        'Model':'LogisticRegression',
        'DataFrame': key,
        'Accuracy': accuracy,
        'Hamming':hamming ,
        'Confusion Matrix': conf_matrix,
        'Classification Report': class_report,
        'Recall': recall,
        'F1 Score': f1,
        'Macro-Recall': macro_recall,
        'Macro-F1': macro_f1,
        'Micro-Recall': micro_recall,
        'Micro-F1': micro_f1,
    })

# Affichage des résultats pour chaque dataframe
for result in results_regression:
    print('\n' + '*' * 70)
    print(result['DataFrame'])
    print('*' * 70)

    field_names = ['Metric', 'Value']
```

```
        # Create a PrettyTable object
    table = PrettyTable(field_names)

    # Add each metric value to the table

    table.add_row(['Accuracy', f"{result['Accuracy']:.2%}"])
    table.add_row(['Hamming', f"{result['Hamming']:.2%}"])
    table.add_row(['Macro-average Recall', f"{result['Macro-Recall']:.2%}"])
    table.add_row(['Macro-average F1 Score', f"{result['Macro-F1']:.2%}"])
    table.add_row(['Micro-average Recall', f"{result['Micro-Recall']:.2%}"])
    table.add_row(['Micro-average F1 Score', f"{result['Micro-F1']:.2%}"])

    # Style the table
    table.align['Metric'] = 'l'
    table.align['Value'] = 'r'

    # Print the table
    print(table)



    # Affichage de la matrice de confusion
    plt.figure(figsize=(8, 6))
    sns.heatmap(result['Confusion Matrix'], annot=True, fmt='d', cmap='Blues',␣
 ↪cbar=False,
                xticklabels=model.classes_, yticklabels=model.classes_)
    plt.xlabel('Prédit')
    plt.ylabel('Réel')
    plt.title(f'Matrice de Confusion pour DataFrame {result["DataFrame"]}')
    plt.show()
# Affichage des résultats pour chaque dataframe sous forme de tableau
# table_headers = ['DataFrame', 'Accuracy', 'Hamming','Macro-average␣
 ↪Recall','Macro-average F1 Score', 'Micro-average Recall', 'Micro-average F1␣
 ↪Score']
# table_rows = [[result['DataFrame'], result['Accuracy'], result['Hamming'],␣
 ↪result['Macro-average Recall'], result['Macro-average F1␣
 ↪Score'],result['Micro-average Recall'],result['Micro-average F1 Score']] for␣
 ↪result in results]

# # Afficher le tableau
# print(tabulate(table_rows, headers=table_headers, tablefmt='pretty'))
```

```
**********************************************************************
MSFT_df
**********************************************************************
+------------------------+--------+
```

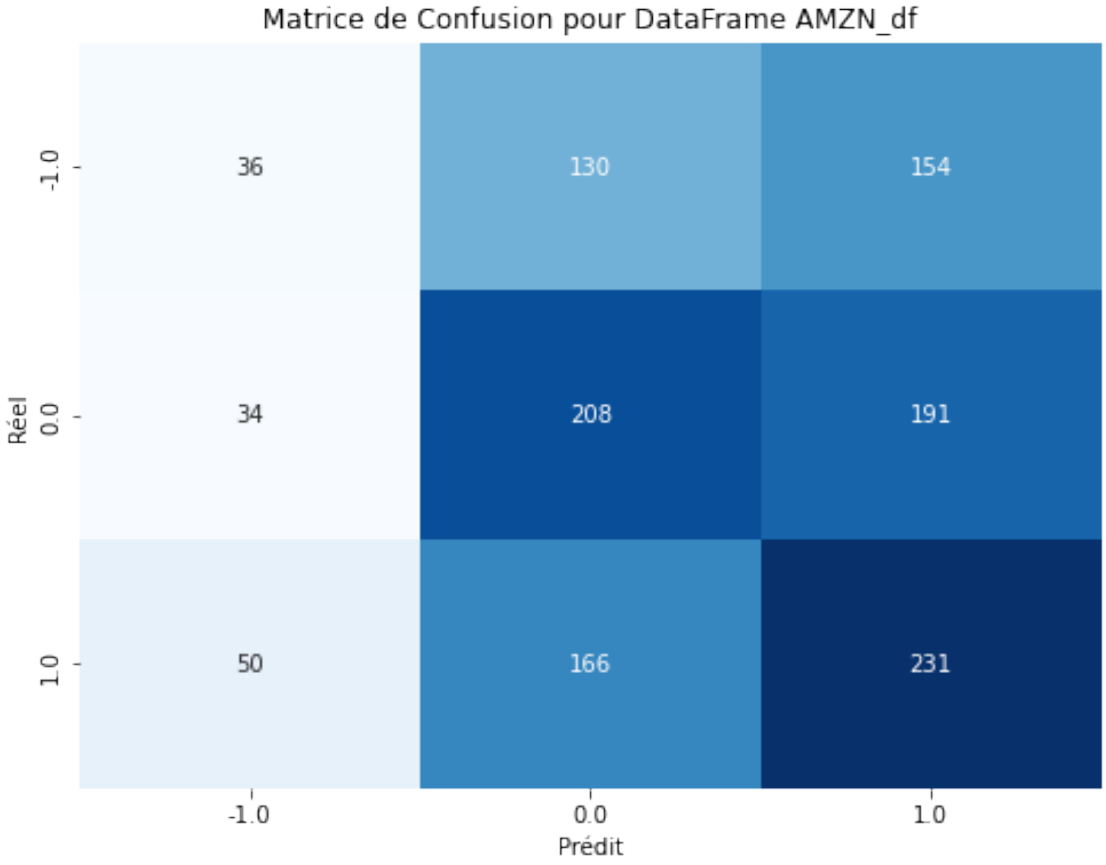
Metric	Value
Accuracy	51.92%
Hamming	48.08%
Macro-average Recall	36.91%
Macro-average F1 Score	30.39%
Micro-average Recall	51.92%
Micro-average F1 Score	51.92%



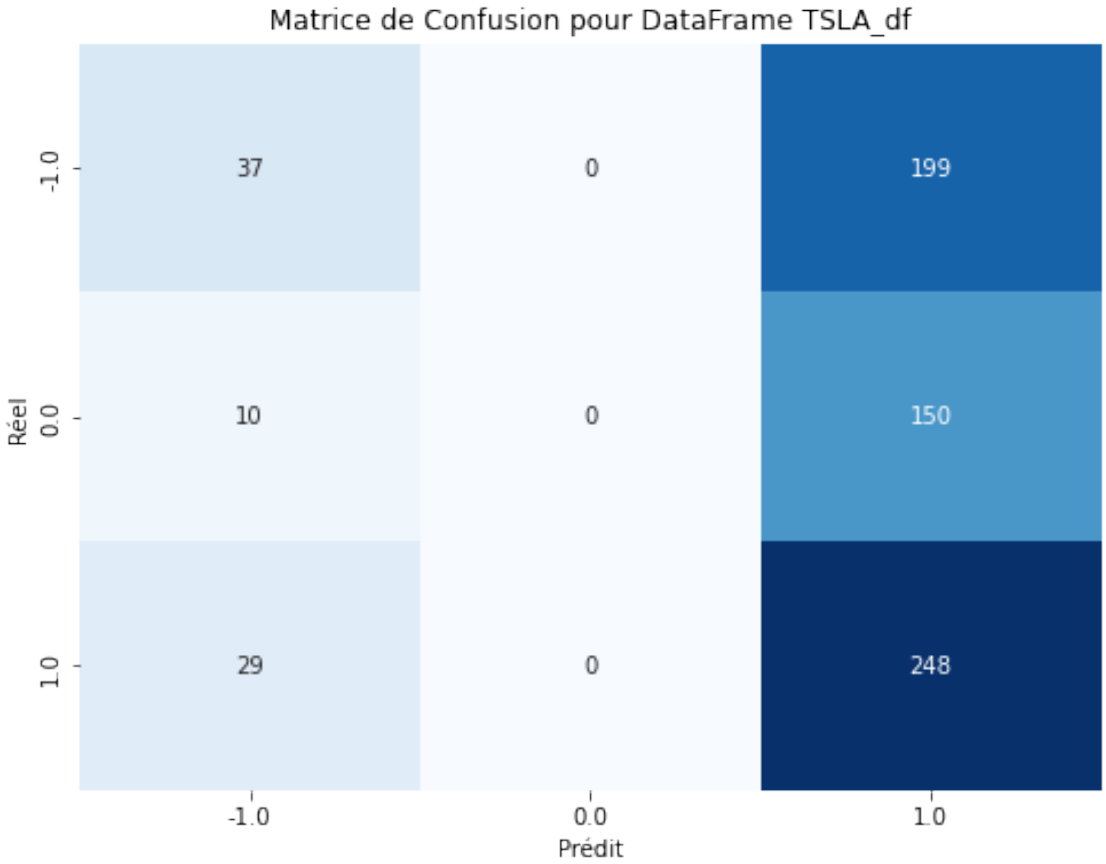
****
AMZN_df
****

Metric	Value
Accuracy	39.58%
Hamming	60.42%
Macro-average Recall	36.99%

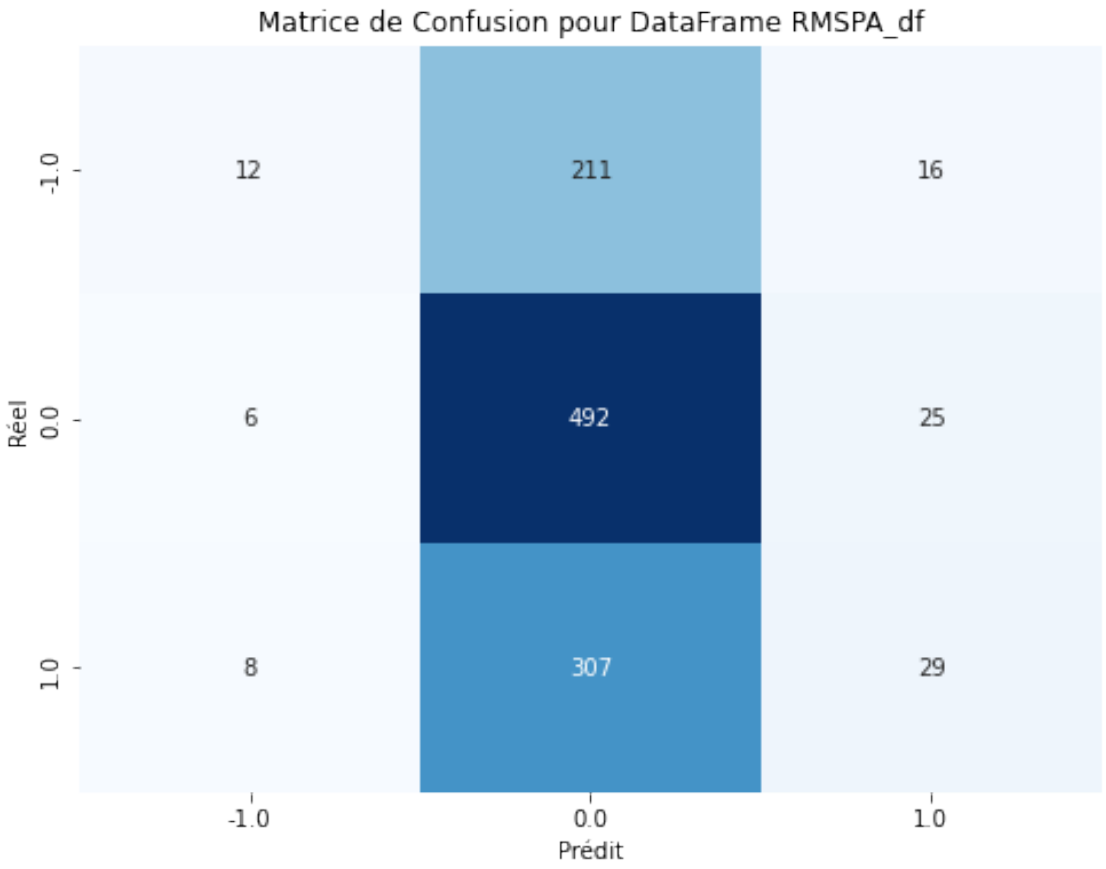
```
| Macro-average F1 Score | 35.31% |
| Micro-average Recall   | 39.58% |
| Micro-average F1 Score | 39.58% |
+------------------------+--------+
```



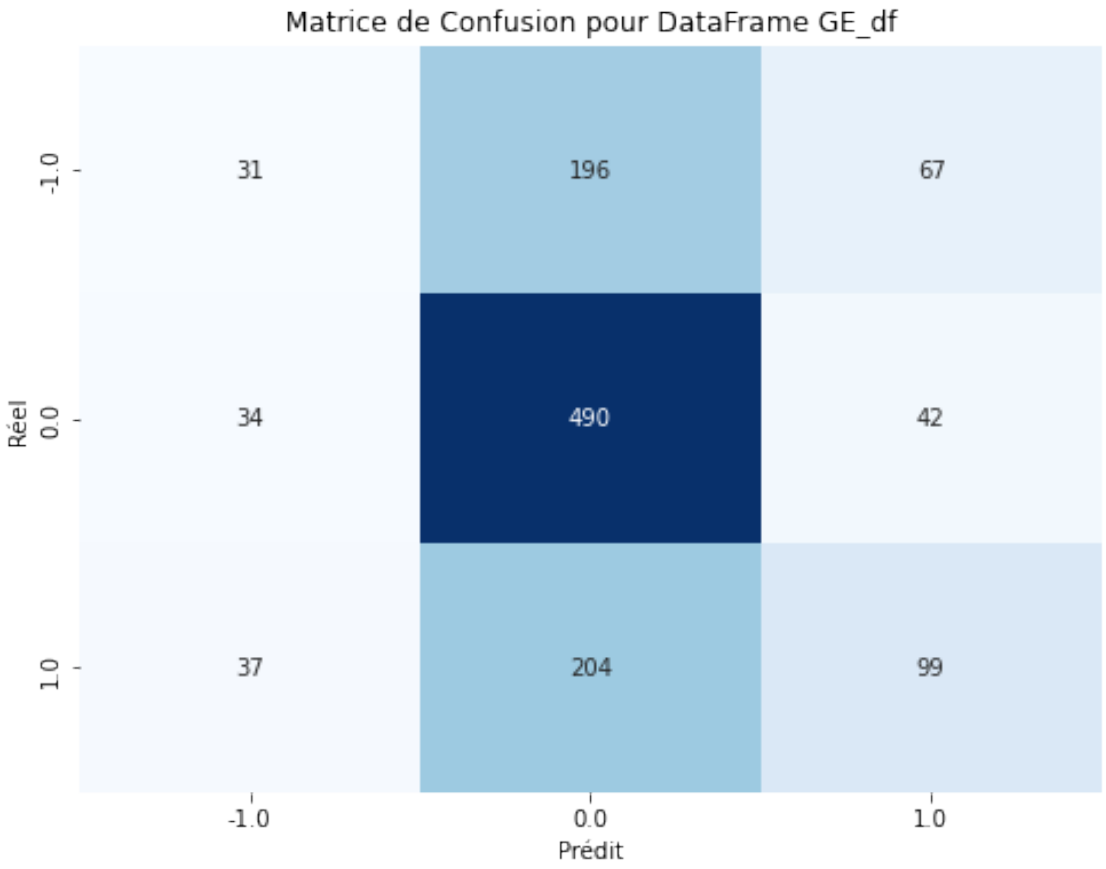

```
**********************************************************************
TSLA_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 42.35% |
| Hamming                | 57.65% |
| Macro-average Recall   | 35.07% |
| Macro-average F1 Score | 26.82% |
| Micro-average Recall   | 42.35% |
| Micro-average F1 Score | 42.35% |
+------------------------+--------+
```



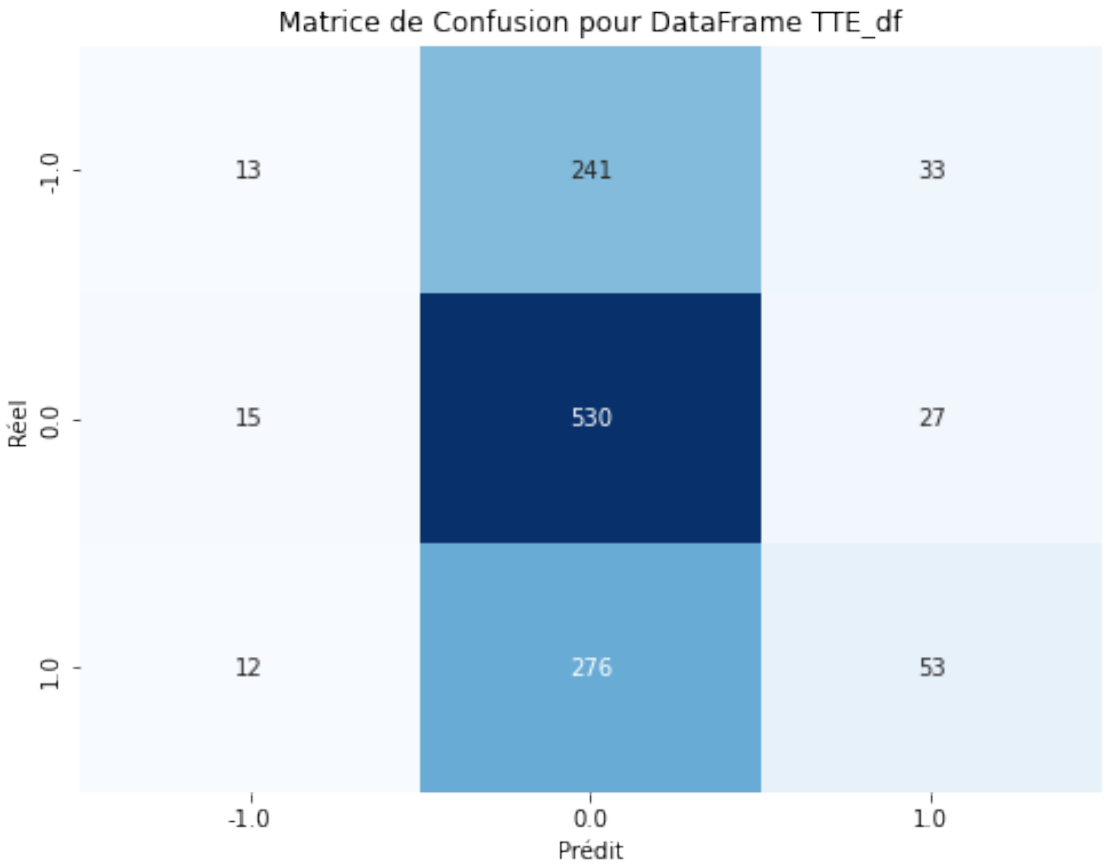

```
**************************************************************************
RMSPA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 48.19% |
| Hamming                | 51.81% |
| Macro-average Recall   | 35.84% |
| Macro-average F1 Score | 29.08% |
| Micro-average Recall   | 48.19% |
| Micro-average F1 Score | 48.19% |
+------------------------+--------+
```




```
**************************************************************************
GE_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 51.67% |
| Hamming                | 48.33% |
| Macro-average Recall   | 42.08% |
| Macro-average F1 Score | 39.70% |
| Micro-average Recall   | 51.67% |
| Micro-average F1 Score | 51.67% |
+------------------------+--------+
```




```
**************************************************************************
TTE_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 49.67% |
| Hamming                | 50.33% |
| Macro-average Recall   | 37.58% |
| Macro-average F1 Score | 32.26% |
| Micro-average Recall   | 49.67% |
| Micro-average F1 Score | 49.67% |
+------------------------+--------+
```



En général, l'analyse des performances des modèles appliqués à différents DataFrames révèle des résultats variables. Pour l'ensemble des DataFrames, les accuracies varient de 39.58% à 51.92%. La Hamming Loss, mesurant la discordance entre prédictions et étiquettes réelles, affiche des valeurs entre 48.08% et 60.42%. Les métriques macro-average, telles que le rappel et le score F1, démontrent des performances modérées, avec des valeurs allant de 35.07% à 42.08% et de 26.82% à 39.70%, respectivement. Les métriques micro-average présentent une performance globalement plus modérée, avec des valeurs entre 39.58% et 51.92%. Ces résultats soulignent la diversité des performances des modèles sur différents ensembles de données et suggèrent la nécessité d'une analyse approfondie et d'ajustements potentiels pour améliorer la robustesse du modèle.

0.4.2 2. SVM

[21]:
```
# from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
# from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, hamming_loss

# Liste pour stocker les résultats de chaque modèle
result_SVM = []
```

```
# Parcourir chaque dataframe dans list_choice
for key, df in list_choice.items():
    # Créez un DataFrame avec les colonnes sélectionnées
    data = df[selected_features + [target_column]]
    # Supprimer les lignes avec des valeurs manquantes
    data.dropna(inplace=True)

    # Divisez les données en ensembles d'entraînement et de test
    X_train, X_test, y_train, y_test = train_test_split(data.
 ↪drop(target_column, axis=1), data[target_column], test_size=0.2,␣
 ↪random_state=42)

    # Standardisez les données
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Entraînez le modèle SVM
    model = SVC(random_state=42)
    model.fit(X_train_scaled, y_train)

    # Prédiction sur l'ensemble de test
    y_pred = model.predict(X_test_scaled)

    # Évaluation de la performance
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)
    hamming = hamming_loss(y_test, y_pred)
    macro_recall = recall_score(y_test, y_pred, average='macro')
    macro_f1 = f1_score(y_test, y_pred, average='macro')
    micro_recall = recall_score(y_test, y_pred, average='micro')
    micro_f1 = f1_score(y_test, y_pred, average='micro')


    # Stocker les résultats dans la liste
    result_SVM.append({
        'Model':'SVM',
        'DataFrame': key,
        'Accuracy': accuracy,
        'Hamming': hamming,
        'Confusion Matrix': conf_matrix,
        'Classification Report': class_report,
        'Macro-Recall': macro_recall,
        'Macro-F1' :macro_f1,
        'Micro-Recall': micro_recall,
```

```
        'Micro-F1':micro_f1

    })
```

[22]:
```
for result_rf in result_SVM :


    print(f"\n{'*' * 70}\nResults for {result_rf['DataFrame']}\n{'*' * 70}")

    # Afficher les résultats sous forme de tableau
#     table_headers1 = [ 'Accuracy', 'Hamming Loss', 'Macro Recall','Macro F1␣
 ↪Score','Micro Recall','Micro F1 Score' ]
#     table_headers2 = [ 'Best Hyperparameters']
    table_rows = []
    field_names = ['Metric', 'Value']
        # Create a PrettyTable object
    table = PrettyTable(field_names)
#     t=PrettyTable(field_names)
    # Add each metric value to the table

    table.add_row(['Accuracy', f"{result_rf['Accuracy']:.2%}"])
    table.add_row(['Hamming', f"{result_rf['Hamming']:.2%}"])
    table.add_row(['Macro-average Recall', f"{result_rf['Macro-Recall']:.2%}"])
    table.add_row(['Macro-average F1 Score', f"{result_rf['Macro-F1']:.2%}"])
    table.add_row(['Micro-average Recall', f"{result_rf['Micro-Recall']:.2%}"])
    table.add_row(['Micro-average F1 Score', f"{result_rf['Micro-F1']:.2%}"])

    # Style the table
    table.align['Metric'] = 'l'
    table.align['Value'] = 'r'

    # Print the table
    print(table)
```

```
**********************************************************************
Results for MSFT_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 53.33% |
| Hamming                | 46.67% |
| Macro-average Recall   | 38.50% |
| Macro-average F1 Score | 33.10% |
| Micro-average Recall   | 53.33% |
| Micro-average F1 Score | 53.33% |
```

```
+------------------------+--------+

**************************************************************************
Results for AMZN_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 43.67% |
| Hamming                | 56.33% |
| Macro-average Recall   | 41.09% |
| Macro-average F1 Score | 39.82% |
| Micro-average Recall   | 43.67% |
| Micro-average F1 Score | 43.67% |
+------------------------+--------+

**************************************************************************
Results for TSLA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 44.13% |
| Hamming                | 55.87% |
| Macro-average Recall   | 36.60% |
| Macro-average F1 Score | 28.33% |
| Micro-average Recall   | 44.13% |
| Micro-average F1 Score | 44.13% |
+------------------------+--------+

**************************************************************************
Results for RMSPA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 49.73% |
| Hamming                | 50.27% |
| Macro-average Recall   | 37.85% |
| Macro-average F1 Score | 32.65% |
| Micro-average Recall   | 49.73% |
| Micro-average F1 Score | 49.73% |
+------------------------+--------+

**************************************************************************
Results for GE_df
**************************************************************************
+------------------------+--------+
```

```
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 52.17% |
| Hamming                | 47.83% |
| Macro-average Recall   | 42.74% |
| Macro-average F1 Score | 40.95% |
| Micro-average Recall   | 52.17% |
| Micro-average F1 Score | 52.17% |
+------------------------+--------+

**********************************************************************
Results for TTE_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 51.17% |
| Hamming                | 48.83% |
| Macro-average Recall   | 39.65% |
| Macro-average F1 Score | 35.86% |
| Micro-average Recall   | 51.17% |
| Micro-average F1 Score | 51.17% |
+------------------------+--------+
```

[ ]:

0.4.3 3. K plus proches voisins (KNN)

[23]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, hamming_loss, recall_score, f1_score

# Liste pour stocker les résultats de chaque modèle k-NN
result_knn = []

def train_knn_model(X_train, y_train, X_test, y_test, key):
    # Standardisation des features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Définition des hyperparamètres à rechercher
    param_grid = {'n_neighbors': [3, 5, 7, 9, 11]}

    # Recherche par grille pour trouver les meilleurs hyperparamètres
```

```
    grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
    grid_search.fit(X_train_scaled, y_train)

    # Affichage des meilleurs hyperparamètres et du meilleur score
    print("Meilleurs hyperparamètres :", grid_search.best_params_)
    print("Meilleur score :", grid_search.best_score_)

    # Entraînement du modèle k-NN avec les meilleurs hyperparamètres
    best_knn_model = grid_search.best_estimator_
    best_knn_model.fit(X_train_scaled, y_train)

    # Prédictions sur l'ensemble de test
    y_pred_knn = best_knn_model.predict(X_test_scaled)

    # Évaluation du modèle
    accuracy = accuracy_score(y_test, y_pred_knn)
    conf_matrix = confusion_matrix(y_test, y_pred_knn)
    class_report = classification_report(y_test, y_pred_knn)
    hamming = hamming_loss(y_test, y_pred_knn)
    macro_recall = recall_score(y_test, y_pred_knn, average='macro')
    macro_f1 = f1_score(y_test, y_pred_knn, average='macro')
    micro_recall = recall_score(y_test, y_pred_knn, average='micro')
    micro_f1 = f1_score(y_test, y_pred_knn, average='micro')

    # Stocker les résultats dans la liste
    result_knn.append({
        'Model':' K plus proches voisins(KNN)',
        'DataFrame': key,
        'Accuracy': accuracy,
        'Hamming': hamming,
        'Confusion Matrix': conf_matrix,
        'Classification Report': class_report,
        'Macro-Recall': macro_recall,
        'Macro-F1': macro_f1,
        'Micro-Recall': micro_recall,
        'Micro-F1': micro_f1
    })

# Parcourir chaque dataframe dans list_choice
for key, df in list_choice.items():
    # Créez un DataFrame avec les colonnes sélectionnées
    data = df[selected_features + [target_column]]
    # Supprimer les lignes avec des valeurs manquantes
    data.dropna(inplace=True)

    # Divisez les données en ensembles d'entraînement et de test
```

```
    X_train, X_test, y_train, y_test = train_test_split(data.
↪drop(target_column, axis=1), data[target_column], test_size=0.2,␣
↪random_state=42)

    print(f"\nTraining k-NN model for DataFrame {key}:")
    train_knn_model(X_train, y_train, X_test, y_test, key)
```

```
Training k-NN model for DataFrame MSFT_df:
Meilleurs hyperparamètres : {'n_neighbors': 11}
Meilleur score : 0.4842676833507126

Training k-NN model for DataFrame AMZN_df:
Meilleurs hyperparamètres : {'n_neighbors': 3}
Meilleur score : 0.45884428223844276

Training k-NN model for DataFrame TSLA_df:
Meilleurs hyperparamètres : {'n_neighbors': 3}
Meilleur score : 0.4897807591396509

Training k-NN model for DataFrame RMSPA_df:
Meilleurs hyperparamètres : {'n_neighbors': 5}
Meilleur score : 0.5246496408211263

Training k-NN model for DataFrame GE_df:
Meilleurs hyperparamètres : {'n_neighbors': 5}
Meilleur score : 0.5536555005213764

Training k-NN model for DataFrame TTE_df:
Meilleurs hyperparamètres : {'n_neighbors': 3}
Meilleur score : 0.5148978971150504
```

[24]:
```
for result_rf in result_knn :


    print(f"\n{'*' * 70}\nResults for {result_rf['DataFrame']}\n{'*' * 70}")
    table_rows = []
    field_names = ['Metric', 'Value']
    table = PrettyTable(field_names)
    table.add_row(['Accuracy', f"{result_rf['Accuracy']:.2%}"])
    table.add_row(['Hamming', f"{result_rf['Hamming']:.2%}"])
    table.add_row(['Macro-average Recall', f"{result_rf['Macro-Recall']:.2%}"])
    table.add_row(['Macro-average F1 Score', f"{result_rf['Macro-F1']:.2%}"])
    table.add_row(['Micro-average Recall', f"{result_rf['Micro-Recall']:.2%}"])
    table.add_row(['Micro-average F1 Score', f"{result_rf['Micro-F1']:.2%}"])

    # Style the table
```

```
    table.align['Metric'] = 'l'
    table.align['Value'] = 'r'

    # Print the table
    print(table)
```

```
**************************************************************************
Results for MSFT_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 52.08% |
| Hamming                | 47.92% |
| Macro-average Recall   | 43.80% |
| Macro-average F1 Score | 43.21% |
| Micro-average Recall   | 52.08% |
| Micro-average F1 Score | 52.08% |
+------------------------+--------+

**************************************************************************
Results for AMZN_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 47.08% |
| Hamming                | 52.92% |
| Macro-average Recall   | 47.94% |
| Macro-average F1 Score | 47.07% |
| Micro-average Recall   | 47.08% |
| Micro-average F1 Score | 47.08% |
+------------------------+--------+

**************************************************************************
Results for TSLA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 48.59% |
| Hamming                | 51.41% |
| Macro-average Recall   | 44.88% |
| Macro-average F1 Score | 44.11% |
| Micro-average Recall   | 48.59% |
| Micro-average F1 Score | 48.59% |
```

```
+------------------------+--------+

**************************************************************************
Results for RMSPA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 50.00% |
| Hamming                | 50.00% |
| Macro-average Recall   | 46.65% |
| Macro-average F1 Score | 46.85% |
| Micro-average Recall   | 50.00% |
| Micro-average F1 Score | 50.00% |
+------------------------+--------+

**************************************************************************
Results for GE_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 54.83% |
| Hamming                | 45.17% |
| Macro-average Recall   | 51.12% |
| Macro-average F1 Score | 51.31% |
| Micro-average Recall   | 54.83% |
| Micro-average F1 Score | 54.83% |
+------------------------+--------+

**************************************************************************
Results for TTE_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 54.67% |
| Hamming                | 45.33% |
| Macro-average Recall   | 53.39% |
| Macro-average F1 Score | 52.98% |
| Micro-average Recall   | 54.67% |
| Micro-average F1 Score | 54.67% |
+------------------------+--------+
```

0.4.4 4 . Random forest

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, hamming_loss
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Liste pour stocker les résultats de chaque modèle avec Grid Search, y compris␣
 ↪Hamming Loss
results_rf_grid_search_hamming = []


# Définir la grille des hyperparamètres à rechercher pour Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [ None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
key, df = next(iter(list_choice.items()))
# Parcourir chaque dataframe dans list_choice
for key, df in list_choice.items():

    # Créez un DataFrame avec les colonnes sélectionnées
    data = df[selected_features + [target_column]]
    # Supprimer les lignes avec des valeurs manquantes
    data.dropna(inplace=True)

    # Divisez les données en ensembles d'entraînement et de test
    X_train, X_test, y_train, y_test = train_test_split(data.
 ↪drop(target_column, axis=1), data[target_column], test_size=0.2,␣
 ↪random_state=42)

    # Création d'un modèle Random Forest
    rf_model = RandomForestClassifier()

    # Utiliser GridSearchCV pour trouver les meilleurs hyperparamètres
    grid_search_rf = GridSearchCV(rf_model, param_grid_rf, cv=5,␣
 ↪scoring='accuracy')
    grid_search_rf.fit(X_train, y_train)

    # Prédictions sur l'ensemble de test
    y_pred_rf = grid_search_rf.predict(X_test)

    # Évaluation du modèle
    accuracy_rf = accuracy_score(y_test, y_pred_rf)
```

```
    class_report_rf = classification_report(y_test, y_pred_rf)
    hamming_rf = hamming_loss(y_test, y_pred_rf)
    macro_recall_rf = recall_score(y_test, y_pred_rf, average='macro')
    macro_f1_rf = f1_score(y_test, y_pred_rf, average='macro')
    micro_recall_rf = recall_score(y_test, y_pred_rf, average='micro')
    micro_f1_rf = f1_score(y_test, y_pred_rf, average='micro')

    # Add the macro and micro averages to your results dictionary
    results_rf_grid_search_hamming.append({
        'Model':'Random Forest',
        'DataFrame': key,
        'Best Hyperparameters': grid_search_rf.best_params_,
        'Accuracy': accuracy_rf,
        'Classification Report': class_report_rf,
        'Hamming': hamming_rf,

        'Macro-Recall': macro_recall_rf,
        'Macro-F1': macro_f1_rf,

        'Micro-Recall': micro_recall_rf,
        'Micro-F1': micro_f1_rf,
    })
```

Evaluation

[39]:
```
for result_rf in results_rf_grid_search_hamming :


    print(f"\n{'*' * 70}\nResults for {result_rf['DataFrame']}\n{'*' * 70}")

    # Afficher les résultats sous forme de tableau
#     table_headers1 = [ 'Accuracy', 'Hamming Loss', 'Macro Recall','Macro F1␣
 ↪Score','Micro Recall','Micro F1 Score' ]
#     table_headers2 = [ 'Best Hyperparameters']
    table_rows = []
    field_names = ['Metric', 'Value']
        # Create a PrettyTable object
    table = PrettyTable(field_names)
#     t=PrettyTable(field_names)
    # Add each metric value to the table
    print(result_rf['Best Hyperparameters'])
    table.add_row(['Accuracy', f"{result_rf['Accuracy']:.2%}"])
    table.add_row(['Hamming', f"{result_rf['Hamming']:.2%}"])
    table.add_row(['Macro-average Recall', f"{result_rf['Macro-Recall']:.2%}"])
```

```
    table.add_row(['Macro-average F1 Score', f"{result_rf['Macro-F1']:.2%}"])
    table.add_row(['Micro-average Recall', f"{result_rf['Micro-Recall']:.2%}"])
    table.add_row(['Micro-average F1 Score', f"{result_rf['Micro-F1']:.2%}"])

    # Style the table
    table.align['Metric'] = 'l'
    table.align['Value'] = 'r'

    # Print the table
    print(table)
```

```
**********************************************************************
Results for MSFT_df
**********************************************************************
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 200}
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 63.83% |
| Hamming                | 36.17% |
| Macro-average Recall   | 57.34% |
| Macro-average F1 Score | 59.11% |
| Micro-average Recall   | 63.83% |
| Micro-average F1 Score | 63.83% |
+------------------------+--------+

**********************************************************************
Results for AMZN_df
**********************************************************************
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 200}
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 59.25% |
| Hamming                | 40.75% |
| Macro-average Recall   | 58.66% |
| Macro-average F1 Score | 58.73% |
| Micro-average Recall   | 59.25% |
| Micro-average F1 Score | 59.25% |
+------------------------+--------+

**********************************************************************
Results for TSLA_df
**********************************************************************
```

```
{'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2,
'n_estimators': 100}
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 58.54% |
| Hamming                | 41.46% |
| Macro-average Recall   | 53.49% |
| Macro-average F1 Score | 52.79% |
| Micro-average Recall   | 58.54% |
| Micro-average F1 Score | 58.54% |
+------------------------+--------+

**********************************************************************
Results for RMSPA_df
**********************************************************************
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 200}
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 59.95% |
| Hamming                | 40.05% |
| Macro-average Recall   | 53.83% |
| Macro-average F1 Score | 55.15% |
| Micro-average Recall   | 59.95% |
| Micro-average F1 Score | 59.95% |
+------------------------+--------+

**********************************************************************
Results for GE_df
**********************************************************************
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 200}
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 61.17% |
| Hamming                | 38.83% |
| Macro-average Recall   | 56.19% |
| Macro-average F1 Score | 57.35% |
| Micro-average Recall   | 61.17% |
| Micro-average F1 Score | 61.17% |
+------------------------+--------+

**********************************************************************
Results for TTE_df
**********************************************************************
```

```
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 200}
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 57.25% |
| Hamming                | 42.75% |
| Macro-average Recall   | 51.66% |
| Macro-average F1 Score | 52.85% |
| Micro-average Recall   | 57.25% |
| Micro-average F1 Score | 57.25% |
+------------------------+--------+
```

[ ]:

0.4.5 5. Gradient boosting

[ ]:
```python
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪hamming_loss, confusion_matrix, recall_score, f1_score
from sklearn.preprocessing import StandardScaler

# Initialize result list
result_gradientb = []

def optimize_gb_model(X_train, y_train, X_test, y_test, key):
    # Standardisation des features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Définir les hyperparamètres à optimiser
    param_dist = {
        'n_estimators': [50, 100, 150, 200],
        'learning_rate': [0.01, 0.1, 0.2, 0.3],
        'max_depth': [3, 4, 5, 6]
    }

    # Initialiser le modèle Gradient Boosting
    gb_model = GradientBoostingClassifier(random_state=42)

    # Initialiser la recherche d'hyperparamètres
    random_search = RandomizedSearchCV(gb_model, param_dist, n_iter=10, cv=5,␣
 ↪random_state=42)

    # Effectuer la recherche d'hyperparamètres sur les données d'entraînement
```

```
    random_search.fit(X_train_scaled, y_train)

    # Afficher les meilleurs hyperparamètres et le meilleur score
    print("Meilleurs hyperparamètres après recherche aléatoire :",␣
 ↪random_search.best_params_)
    print("Meilleur score après recherche aléatoire :", random_search.
 ↪best_score_)

    # Prédictions sur l'ensemble de test avec les meilleurs hyperparamètres
    y_pred_gb_optimized = random_search.predict(X_test_scaled)

    # Calculer les métriques
    accuracy = accuracy_score(y_test, y_pred_gb_optimized)
    hamming = hamming_loss(y_test, y_pred_gb_optimized)
    conf_matrix = confusion_matrix(y_test, y_pred_gb_optimized)
    class_report = classification_report(y_test, y_pred_gb_optimized)
    macro_recall = recall_score(y_test, y_pred_gb_optimized, average='macro')
    macro_f1 = f1_score(y_test, y_pred_gb_optimized, average='macro')
    micro_recall = recall_score(y_test, y_pred_gb_optimized, average='micro')
    micro_f1 = f1_score(y_test, y_pred_gb_optimized, average='micro')

    # Append results to the list
    result_gradientb.append({
        'Model':'Gradient boosting',
        'DataFrame': key,
        'Accuracy': accuracy,
        'Hamming': hamming,
        'Confusion Matrix': conf_matrix,
        'Classification Report': class_report,
        'Macro-Recall': macro_recall,
        'Macro-F1': macro_f1,
        'Micro-Recall': micro_recall,
        'Micro-F1': micro_f1
    })

# Parcourir chaque dataframe dans list_choice
for key, df in list_choice.items():
    # Créez un DataFrame avec les colonnes sélectionnées
    data = df[selected_features + [target_column]]
    # Supprimer les lignes avec des valeurs manquantes
    data.dropna(inplace=True)

    # Divisez les données en ensembles d'entraînement et de test
    X_train, X_test, y_train, y_test = train_test_split(data.
 ↪drop(target_column, axis=1), data[target_column], test_size=0.2,␣
 ↪random_state=42)
```

```
    print(f"\nOptimizing Gradient Boosting model for DataFrame {key}:")
    optimize_gb_model(X_train, y_train, X_test, y_test, key)
```

[40]:
```
for result_rf in result_gradientb :


    print(f"\n{'*' * 70}\nResults for {result_rf['DataFrame']}\n{'*' * 70}")
    table_rows = []
    field_names = ['Metric', 'Value']
    table = PrettyTable(field_names)
    table.add_row(['Accuracy', f"{result_rf['Accuracy']:.2%}"])
    table.add_row(['Hamming', f"{result_rf['Hamming']:.2%}"])
    table.add_row(['Macro-average Recall', f"{result_rf['Macro-Recall']:.2%}"])
    table.add_row(['Macro-average F1 Score', f"{result_rf['Macro-F1']:.2%}"])
    table.add_row(['Micro-average Recall', f"{result_rf['Micro-Recall']:.2%}"])
    table.add_row(['Micro-average F1 Score', f"{result_rf['Micro-F1']:.2%}"])

    # Style the table
    table.align['Metric'] = 'l'
    table.align['Value'] = 'r'

    # Print the table
    print(table)
```

```
**********************************************************************
Results for MSFT_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 57.08% |
| Hamming                | 42.92% |
| Macro-average Recall   | 50.57% |
| Macro-average F1 Score | 51.44% |
| Micro-average Recall   | 57.08% |
| Micro-average F1 Score | 57.08% |
+------------------------+--------+

**********************************************************************
Results for AMZN_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 54.33% |
| Hamming                | 45.67% |
```

```
| Macro-average Recall   | 53.67% |
| Macro-average F1 Score | 53.77% |
| Micro-average Recall   | 54.33% |
| Micro-average F1 Score | 54.33% |
+------------------------+--------+

**************************************************************************
Results for TSLA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 54.83% |
| Hamming                | 45.17% |
| Macro-average Recall   | 50.29% |
| Macro-average F1 Score | 49.66% |
| Micro-average Recall   | 54.83% |
| Micro-average F1 Score | 54.83% |
+------------------------+--------+

**************************************************************************
Results for RMSPA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 56.60% |
| Hamming                | 43.40% |
| Macro-average Recall   | 50.58% |
| Macro-average F1 Score | 51.42% |
| Micro-average Recall   | 56.60% |
| Micro-average F1 Score | 56.60% |
+------------------------+--------+

**************************************************************************
Results for GE_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 56.67% |
| Hamming                | 43.33% |
| Macro-average Recall   | 51.18% |
| Macro-average F1 Score | 51.81% |
| Micro-average Recall   | 56.67% |
| Micro-average F1 Score | 56.67% |
+------------------------+--------+
```

```
**********************************************************************
Results for TTE_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 52.83% |
| Hamming                | 47.17% |
| Macro-average Recall   | 46.33% |
| Macro-average F1 Score | 46.71% |
| Micro-average Recall   | 52.83% |
| Micro-average F1 Score | 52.83% |
+------------------------+--------+
```

[ ]:

0.4.6 6.LSTM

[ ]:

[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import hamming_loss, precision_recall_fscore_support
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Liste pour stocker les résultats de chaque modèle LSTM
results_LSTM = []

def train_lstm_model(df, features, key):
    # Définir les caractéristiques (X) et la cible (y)
    X = df[features].values
    y = df['Trend'].values

    # Encoder les étiquettes en utilisant LabelEncoder
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    # Normaliser les données
    scaler = MinMaxScaler(feature_range=(0, 1))
    X_scaled = scaler.fit_transform(X)
```

```
    # Diviser les données en ensembles d'entraînement et de test avec␣
↪stratification
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded,␣
↪test_size=0.2, random_state=42, stratify=y_encoded)

    # Remodeler les données pour les rendre compatibles avec LSTM
    X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
    X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

    # Convertir les étiquettes en encodage one-hot (si nécessaire)
    num_classes = len(np.unique(y))
    y_train_onehot = to_categorical(y_train, num_classes=num_classes)
    y_test_onehot = to_categorical(y_test, num_classes=num_classes)

    # Créer le modèle LSTM
    model = Sequential()
    model.add(LSTM(300, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    # Compiler le modèle
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])

    # Entraîner le modèle et enregistrer l'historique de l'entraînement
    history = model.fit(X_train, y_train_onehot, epochs=10, batch_size=32,␣
↪validation_data=(X_test, y_test_onehot))

    # Évaluation sur l'ensemble de test
    test_loss, test_accuracy = model.evaluate(X_test, y_test_onehot)
    print(f'\nTest Accuracy: {test_accuracy}')

    # Prédictions sur l'ensemble de test
    y_pred = model.predict(X_test)

    # Convertir les prédictions en classe binaire (0, 1, 2) plutôt qu'en␣
↪encodage one-hot
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_test_classes = np.argmax(y_test_onehot, axis=1)

    # Calcul de l'indice de Hamming
    hamming = hamming_loss(y_test_classes, y_pred_classes)
    print(f'\nHamming Loss: {hamming}')

    # Calcul du micro-average (précision, rappel, f1-score)
    micro_avg = precision_recall_fscore_support(y_test_classes, y_pred_classes,␣
↪average='micro')
```

```
    # Calcul du macro-average (précision, rappel, f1-score)
    macro_avg = precision_recall_fscore_support(y_test_classes, y_pred_classes,␣
 ↪average='macro')


    results_LSTM.append({
        'Model':'LSTM',
        'DataFrame': key,
        'Accuracy': test_accuracy,
        'Hamming': hamming,

        'Micro-Recall': micro_avg[1],
        'Micro-F1': micro_avg[2],

        'Macro-Recall': macro_avg[1],
        'Macro-F1': macro_avg[2]
    })

# Appliquer la fonction à chaque dataframe de la liste
for key, df in list_choice.items():
    print(f'\nTraining LSTM Model for DataFrame {key}...')
    train_lstm_model(df, selected_features, key)
```

[41]:
```
for result_rf in results_LSTM :


    print(f"\n{'*' * 70}\nResults for {result_rf['DataFrame']}\n{'*' * 70}")
    table_rows = []
    field_names = ['Metric', 'Value']
    table = PrettyTable(field_names)
    table.add_row(['Accuracy', f"{result_rf['Accuracy']:.2%}"])
    table.add_row(['Hamming', f"{result_rf['Hamming']:.2%}"])
    table.add_row(['Macro-average Recall', f"{result_rf['Macro-Recall']:.2%}"])
    table.add_row(['Macro-average F1 Score', f"{result_rf['Macro-F1']:.2%}"])
    table.add_row(['Micro-average Recall', f"{result_rf['Micro-Recall']:.2%}"])
    table.add_row(['Micro-average F1 Score', f"{result_rf['Micro-F1']:.2%}"])


    # Style the table
    table.align['Metric'] = 'l'
    table.align['Value'] = 'r'

    # Print the table
    print(table)
```

```
**********************************************************************
Results for MSFT_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 49.42% |
| Hamming                | 50.58% |
| Macro-average Recall   | 34.85% |
| Macro-average F1 Score | 27.57% |
| Micro-average Recall   | 49.42% |
| Micro-average F1 Score | 49.42% |
+------------------------+--------+

**********************************************************************
Results for AMZN_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 40.83% |
| Hamming                | 59.17% |
| Macro-average Recall   | 38.41% |
| Macro-average F1 Score | 32.27% |
| Micro-average Recall   | 40.83% |
| Micro-average F1 Score | 40.83% |
+------------------------+--------+

**********************************************************************
Results for TSLA_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 44.13% |
| Hamming                | 55.87% |
| Macro-average Recall   | 35.09% |
| Macro-average F1 Score | 26.44% |
| Micro-average Recall   | 44.13% |
| Micro-average F1 Score | 44.13% |
+------------------------+--------+

**********************************************************************
Results for RMSPA_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
```

```
| Accuracy                | 47.38% |
| Hamming                 | 52.62% |
| Macro-average Recall    | 33.91% |
| Macro-average F1 Score | 24.88% |
| Micro-average Recall    | 47.38% |
| Micro-average F1 Score | 47.38% |
+-------------------------+--------+

**********************************************************************
Results for GE_df
**********************************************************************
+-------------------------+--------+
| Metric                  |  Value |
+-------------------------+--------+
| Accuracy                | 51.50% |
| Hamming                 | 48.50% |
| Macro-average Recall    | 38.96% |
| Macro-average F1 Score | 33.45% |
| Micro-average Recall    | 51.50% |
| Micro-average F1 Score | 51.50% |
+-------------------------+--------+

**********************************************************************
Results for TTE_df
**********************************************************************
+-------------------------+--------+
| Metric                  |  Value |
+-------------------------+--------+
| Accuracy                | 47.92% |
| Hamming                 | 52.08% |
| Macro-average Recall    | 37.12% |
| Macro-average F1 Score | 31.37% |
| Micro-average Recall    | 47.92% |
| Micro-average F1 Score | 47.92% |
+-------------------------+--------+
```

[ ]:

0.4.7 7. Réseaux de neurones récurrents (RNN)

[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import hamming_loss, precision_recall_fscore_support
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Liste pour stocker les résultats de chaque modèle RNN
results_RNN = []

def train_rnn_model(df, features, key):
    # Définir les caractéristiques (X) et la cible (y)
    X = df[features].values
    y = df['Trend'].values

    # Encoder les étiquettes en utilisant LabelEncoder
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    # Normaliser les données
    scaler = MinMaxScaler(feature_range=(0, 1))
    X_scaled = scaler.fit_transform(X)

    # Diviser les données en ensembles d'entraînement et de test avec␣
 ↪stratification
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded,␣
 ↪test_size=0.2, random_state=42, stratify=y_encoded)

    # Remodeler les données pour les rendre compatibles avec RNN
    X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
    X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

    # Convertir les étiquettes en encodage one-hot (si nécessaire)
    num_classes = len(np.unique(y))
    y_train_onehot = to_categorical(y_train, num_classes=num_classes)
    y_test_onehot = to_categorical(y_test, num_classes=num_classes)

    # Créer le modèle RNN
    model = Sequential()
    model.add(SimpleRNN(300, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    # Compiler le modèle
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

    # Entraîner le modèle et enregistrer l'historique de l'entraînement
    history = model.fit(X_train, y_train_onehot, epochs=10, batch_size=32,␣
 ↪validation_data=(X_test, y_test_onehot))
```

```
    # Évaluation sur l'ensemble de test
    test_loss, test_accuracy = model.evaluate(X_test, y_test_onehot)
    print(f'\nTest Accuracy: {test_accuracy}')

    # Prédictions sur l'ensemble de test
    y_pred = model.predict(X_test)

    # Convertir les prédictions en classe binaire (0, 1, 2) plutôt qu'en␣
↪encodage one-hot
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_test_classes = np.argmax(y_test_onehot, axis=1)

    # Calcul de l'indice de Hamming
    hamming = hamming_loss(y_test_classes, y_pred_classes)


    # Calcul du micro-average (précision, rappel, f1-score)
    micro_avg = precision_recall_fscore_support(y_test_classes, y_pred_classes,␣
↪average='micro')


    # Calcul du macro-average (précision, rappel, f1-score)
    macro_avg = precision_recall_fscore_support(y_test_classes, y_pred_classes,␣
↪average='macro')
    print(f'\nMacro-average Precision: {macro_avg[0]}')
    print(f'Macro-average Recall: {macro_avg[1]}')
    print(f'Macro-average F1 Score: {macro_avg[2]}')

    results_RNN.append({
        'Model':'RNN',
        'DataFrame': key,
        'Accuracy': test_accuracy,
        'Hamming': hamming,

        'Micro-Recall': micro_avg[1],
        'Micro-F1': micro_avg[2],

        'Macro-Recall': macro_avg[1],
        'Macro-F1': macro_avg[2]
    })

# Appliquer la fonction à chaque dataframe de la liste
for key, df in list_choice.items():
    print(f'\nTraining RNN Model for DataFrame {key}...')
    train_rnn_model(df, selected_features, key)
```

[42]:
```
for result_rf in results_RNN :


    print(f"\n{'*' * 70}\nResults for {result_rf['DataFrame']}\n{'*' * 70}")
    table_rows = []
    field_names = ['Metric', 'Value']
    table = PrettyTable(field_names)
    table.add_row(['Accuracy', f"{result_rf['Accuracy']:.2%}"])
    table.add_row(['Hamming', f"{result_rf['Hamming']:.2%}"])
    table.add_row(['Macro-average Recall', f"{result_rf['Macro-Recall']:.2%}"])
    table.add_row(['Macro-average F1 Score', f"{result_rf['Macro-F1']:.2%}"])
    table.add_row(['Micro-average Recall', f"{result_rf['Micro-Recall']:.2%}"])
    table.add_row(['Micro-average F1 Score', f"{result_rf['Micro-F1']:.2%}"])


    # Style the table
    table.align['Metric'] = 'l'
    table.align['Value'] = 'r'

    # Print the table
    print(table)
```

```
**********************************************************************
Results for MSFT_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 49.50% |
| Hamming                | 50.50% |
| Macro-average Recall   | 35.36% |
| Macro-average F1 Score | 28.89% |
| Micro-average Recall   | 49.50% |
| Micro-average F1 Score | 49.50% |
+------------------------+--------+

**********************************************************************
Results for AMZN_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 41.67% |
| Hamming                | 58.33% |
| Macro-average Recall   | 39.27% |
| Macro-average F1 Score | 33.81% |
```

```
| Micro-average Recall   | 41.67% |
| Micro-average F1 Score | 41.67% |
+------------------------+--------+

**************************************************************************
Results for TSLA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 42.35% |
| Hamming                | 57.65% |
| Macro-average Recall   | 34.35% |
| Macro-average F1 Score | 27.91% |
| Micro-average Recall   | 42.35% |
| Micro-average F1 Score | 42.35% |
+------------------------+--------+

**************************************************************************
Results for RMSPA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 48.28% |
| Hamming                | 51.72% |
| Macro-average Recall   | 35.45% |
| Macro-average F1 Score | 28.41% |
| Micro-average Recall   | 48.28% |
| Micro-average F1 Score | 48.28% |
+------------------------+--------+

**************************************************************************
Results for GE_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 52.08% |
| Hamming                | 47.92% |
| Macro-average Recall   | 39.79% |
| Macro-average F1 Score | 35.59% |
| Micro-average Recall   | 52.08% |
| Micro-average F1 Score | 52.08% |
+------------------------+--------+

**************************************************************************
Results for TTE_df
```

```
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 48.08% |
| Hamming                | 51.92% |
| Macro-average Recall   | 37.52% |
| Macro-average F1 Score | 32.95% |
| Micro-average Recall   | 48.08% |
| Micro-average F1 Score | 48.08% |
+------------------------+--------+
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

0.4.8 8.Réseaux de neurones convolutifs (CNN)

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import hamming_loss, precision_recall_fscore_support
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,␣
↪Dropout
from tensorflow.keras.utils import to_categorical

# Liste pour stocker les résultats de chaque modèle CNN
results_CNN = []

def train_cnn_model(df, features, key):
    # Définir les caractéristiques (X) et la cible (y)
    X = df[features].values
    y = df['Trend'].values

    # Encoder les étiquettes en utilisant LabelEncoder
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    # Normaliser les données
```

```
    scaler = MinMaxScaler(feature_range=(0, 1))
    X_scaled = scaler.fit_transform(X)

    # Diviser les données en ensembles d'entraînement et de test avec␣
↪stratification
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded,␣
↪test_size=0.2, random_state=42, stratify=y_encoded)

    # Remodeler les données pour les rendre compatibles avec CNN
    X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
    X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

    # Convertir les étiquettes en encodage one-hot (si nécessaire)
    num_classes = len(np.unique(y))
    y_train_onehot = to_categorical(y_train, num_classes=num_classes)
    y_test_onehot = to_categorical(y_test, num_classes=num_classes)

    # Créer le modèle CNN
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu',␣
↪input_shape=(X_train.shape[1], 1)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    # Compiler le modèle
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])

    # Entraîner le modèle et enregistrer l'historique de l'entraînement
    history = model.fit(X_train, y_train_onehot, epochs=10, batch_size=32,␣
↪validation_data=(X_test, y_test_onehot))

    # Afficher les courbes d'apprentissage (loss et accuracy) au fil des époques
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()

    # Évaluation sur l'ensemble de test
    test_loss, test_accuracy = model.evaluate(X_test, y_test_onehot)
    print(f'\nTest Accuracy: {test_accuracy}')

    # Prédictions sur l'ensemble de test
    y_pred = model.predict(X_test)

    # Convertir les prédictions en classe binaire (0, 1, 2) plutôt qu'en␣
↪encodage one-hot
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_test_classes = np.argmax(y_test_onehot, axis=1)

    # Calcul de l'indice de Hamming
    hamming = hamming_loss(y_test_classes, y_pred_classes)

    # Calcul du micro-average (précision, rappel, f1-score)
    micro_avg = precision_recall_fscore_support(y_test_classes, y_pred_classes,␣
↪average='micro')

    # Calcul du macro-average (précision, rappel, f1-score)
    macro_avg = precision_recall_fscore_support(y_test_classes, y_pred_classes,␣
↪average='macro')



    results_CNN.append({
        'Model':'CNN',
        'DataFrame': key,
        'Accuracy': test_accuracy,
        'Hamming': hamming,

        'Micro-Recall': micro_avg[1],
        'Micro-F1': micro_avg[2],

        'Macro-Recall': macro_avg[1],
        'Macro-F1': macro_avg[2]
    })

# Appliquer la fonction à chaque dataframe de la liste
```

```
for key, df in list_choice.items():
    print(f'\nTraining CNN Model for DataFrame {key}...')
    train_cnn_model(df, selected_features, key)
```

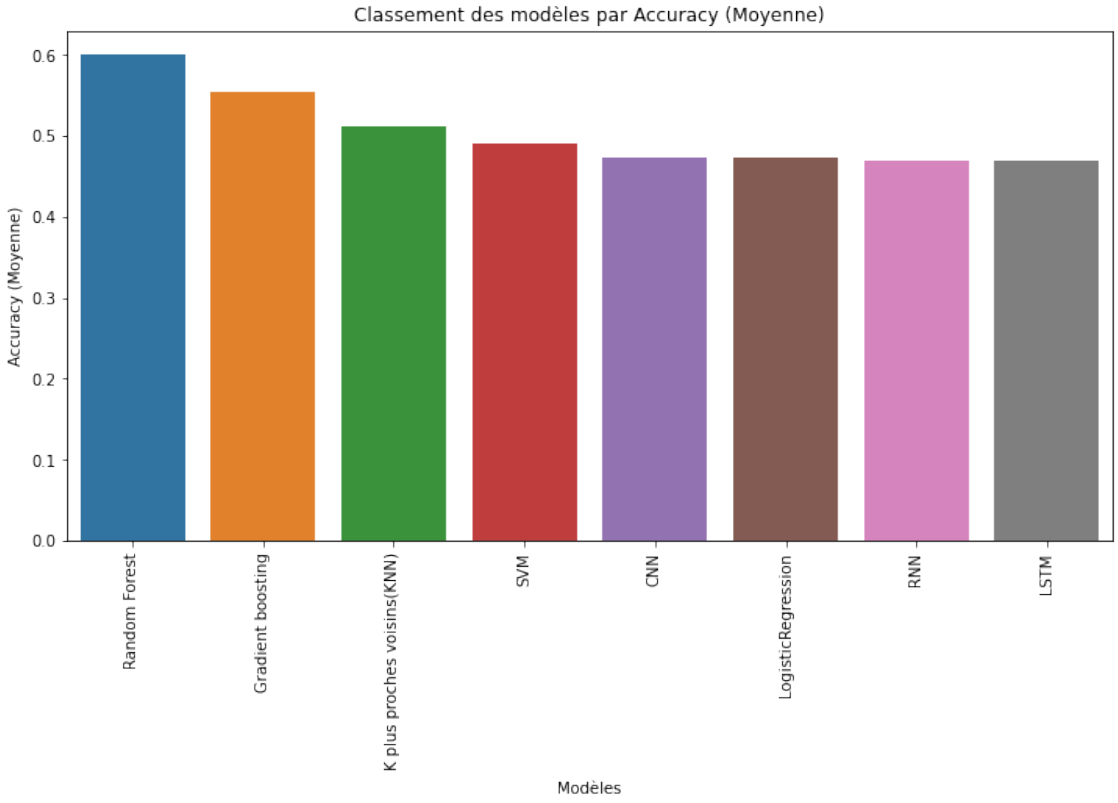
[50]:
```
for result_rf in results_CNN :


    print(f"\n{'*' * 70}\nResults for {result_rf['DataFrame']}\n{'*' * 70}")
    table_rows = []
    field_names = ['Metric', 'Value']
    table = PrettyTable(field_names)
    table.add_row(['Accuracy', f"{result_rf['Accuracy']:.2%}"])
    table.add_row(['Hamming', f"{result_rf['Hamming']:.2%}"])
    table.add_row(['Macro-average Recall', f"{result_rf['Macro-Recall']:.2%}"])
    table.add_row(['Macro-average F1 Score', f"{result_rf['Macro-F1']:.2%}"])
    table.add_row(['Micro-average Recall', f"{result_rf['Micro-Recall']:.2%}"])
    table.add_row(['Micro-average F1 Score', f"{result_rf['Micro-F1']:.2%}"])


    # Style the table
    table.align['Metric'] = 'l'
    table.align['Value'] = 'r'

    # Print the table
    print(table)
```

```
**********************************************************************
Results for MSFT_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 49.67% |
| Hamming                | 50.33% |
| Macro-average Recall   | 34.40% |
| Macro-average F1 Score | 25.65% |
| Micro-average Recall   | 49.67% |
| Micro-average F1 Score | 49.67% |
+------------------------+--------+

**********************************************************************
Results for AMZN_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
```

```
| Accuracy               | 41.08% |
| Hamming                | 58.92% |
| Macro-average Recall   | 39.82% |
| Macro-average F1 Score | 39.10% |
| Micro-average Recall   | 41.08% |
| Micro-average F1 Score | 41.08% |
+------------------------+--------+

**************************************************************************
Results for TSLA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 44.58% |
| Hamming                | 55.42% |
| Macro-average Recall   | 35.63% |
| Macro-average F1 Score | 27.58% |
| Micro-average Recall   | 44.58% |
| Micro-average F1 Score | 44.58% |
+------------------------+--------+

**************************************************************************
Results for RMSPA_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 47.74% |
| Hamming                | 52.26% |
| Macro-average Recall   | 33.62% |
| Macro-average F1 Score | 22.55% |
| Micro-average Recall   | 47.74% |
| Micro-average F1 Score | 47.74% |
+------------------------+--------+

**************************************************************************
Results for GE_df
**************************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 52.08% |
| Hamming                | 47.92% |
| Macro-average Recall   | 40.46% |
| Macro-average F1 Score | 36.64% |
| Micro-average Recall   | 52.08% |
| Micro-average F1 Score | 52.08% |
```

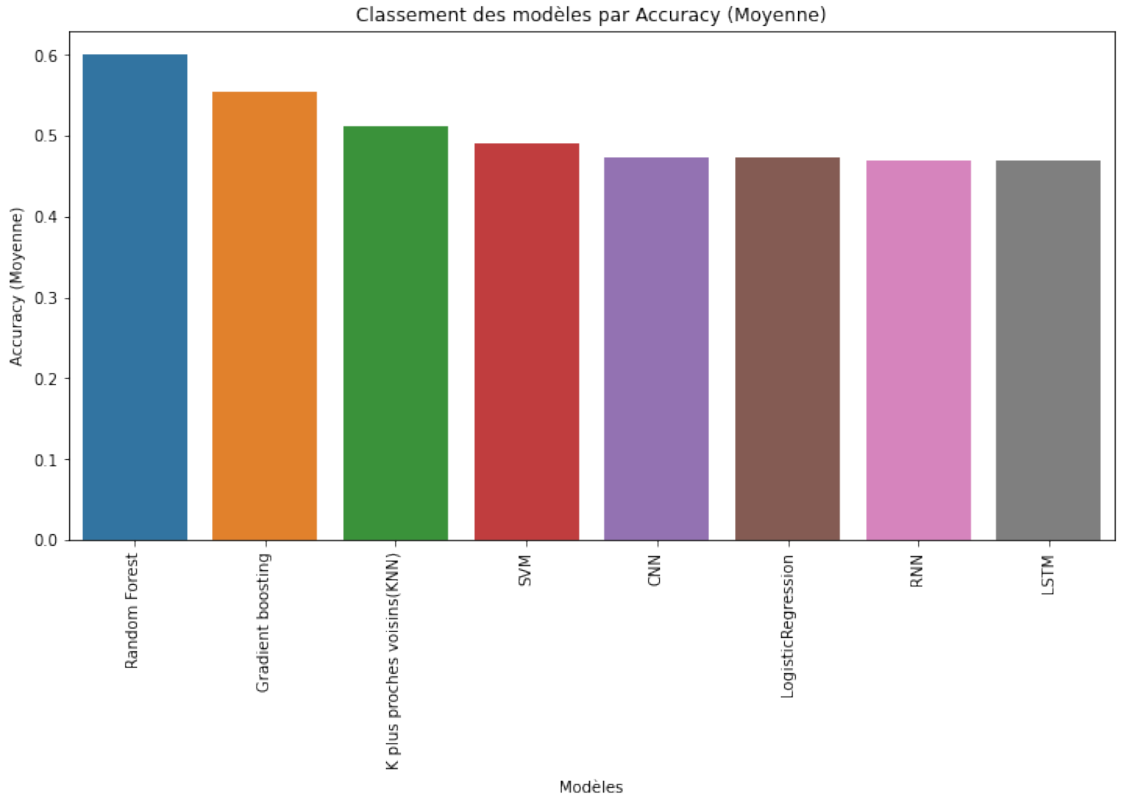
```
+------------------------+--------+

**********************************************************************
Results for TTE_df
**********************************************************************
+------------------------+--------+
| Metric                 |  Value |
+------------------------+--------+
| Accuracy               | 48.33% |
| Hamming                | 51.67% |
| Macro-average Recall   | 38.46% |
| Macro-average F1 Score | 34.52% |
| Micro-average Recall   | 48.33% |
| Micro-average F1 Score | 48.33% |
+------------------------+--------+
```

[53]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Liste pour stocker les résultats de chaque modèle
df_results = [results_regression, result_gradientb,␣
 ↪results_rf_grid_search_hamming, result_SVM, result_knn, results_LSTM,␣
 ↪results_RNN, results_CNN]

# Convertir les résultats en DataFrames
dfs = [pd.DataFrame(result) for result in df_results]

# Concaténer les DataFrames en un seul
df_concatenated = pd.concat(dfs, ignore_index=True)

# Calculer la moyenne de l'Accuracy pour chaque modèle
df_model_mean = df_concatenated.groupby('Model')['Accuracy'].mean().
 ↪reset_index()

df_model_mean_sorted = df_model_mean.sort_values(by='Accuracy', ascending=False)

# Visualisation du classement trié
plt.figure(figsize=(12, 6))
sns.barplot(x='Model', y='Accuracy', data=df_model_mean_sorted)
plt.title('Classement des modèles par Accuracy (Moyenne)')
plt.xlabel('Modèles')
plt.ylabel('Accuracy (Moyenne)')
plt.xticks(rotation=90)
plt.show()
```



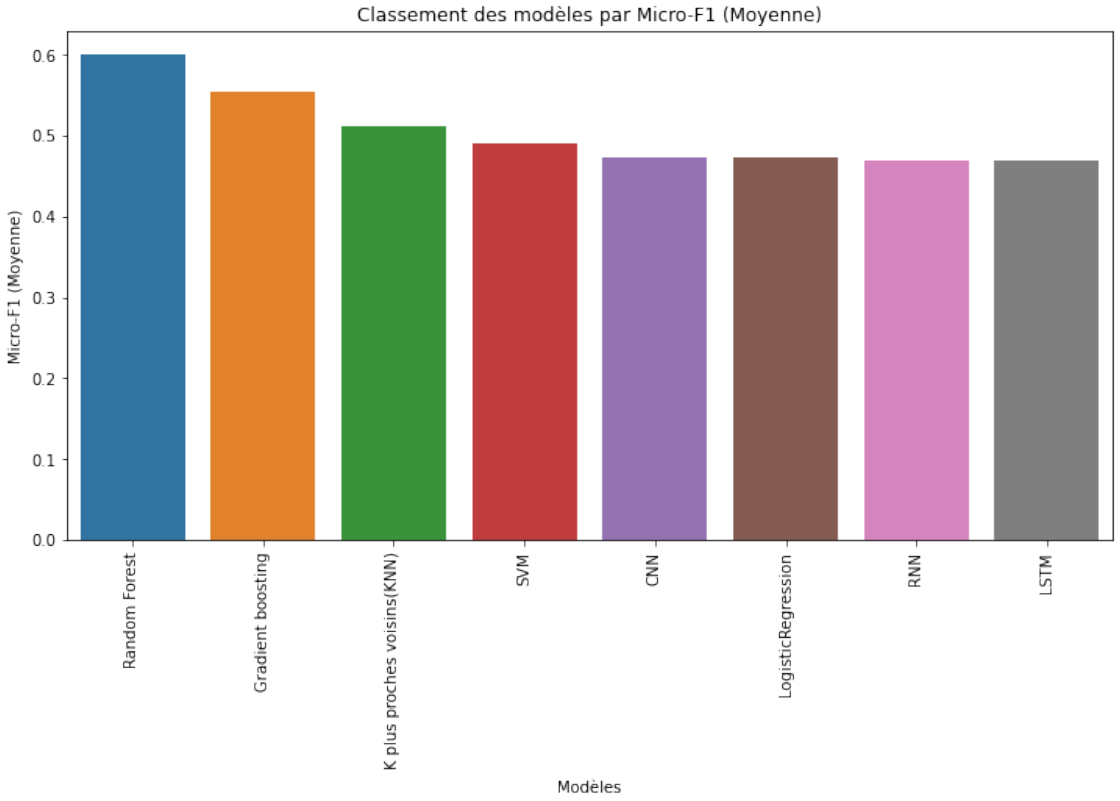
[56]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Liste pour stocker les résultats de chaque modèle
df_results = [results_regression, result_gradientb,␣
 ↪results_rf_grid_search_hamming, result_SVM, result_knn, results_LSTM,␣
 ↪results_RNN, results_CNN]

# Convertir les résultats en DataFrames
dfs = [pd.DataFrame(result) for result in df_results]

# Calculer la moyenne de l'Accuracy pour chaque modèle
df_model_mean = df_concatenated.groupby('Model')['Accuracy'].mean().
 ↪reset_index()
df_model_mean_sorted = df_model_mean.sort_values(by='Accuracy', ascending=False)

# Visualisation du classement trié
plt.figure(figsize=(12, 6))
sns.barplot(x='Model', y='Accuracy', data=df_model_mean_sorted)
plt.title('Classement des modèles par Accuracy (Moyenne)')
```

```
plt.xlabel('Modèles')
plt.ylabel('Accuracy (Moyenne)')
plt.xticks(rotation=90)
plt.show()
```



[55]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Liste pour stocker les résultats de chaque modèle
df_results = [results_regression, result_gradientb,␣
 ↪results_rf_grid_search_hamming, result_SVM, result_knn, results_LSTM,␣
 ↪results_RNN, results_CNN]

# Convertir les résultats en DataFrames
dfs = [pd.DataFrame(result) for result in df_results]

# Calculer la moyenne de l'Accuracy pour chaque modèle
df_model_mean = df_concatenated.groupby('Model')['Micro-F1'].mean().
 ↪reset_index()
df_model_mean_sorted = df_model_mean.sort_values(by='Micro-F1', ascending=False)
```

```
# Visualisation du classement trié
plt.figure(figsize=(12, 6))
sns.barplot(x='Model', y='Micro-F1', data=df_model_mean_sorted)
plt.title('Classement des modèles par Micro-F1 (Moyenne)')
plt.xlabel('Modèles')
plt.ylabel('Micro-F1 (Moyenne)')
plt.xticks(rotation=90)
plt.show()
```



[ ]:

[ ]:

[ ]: