TD Projet DATA - Prédiction des Prix et Actifs du S&P 500 grâce à du Machine Learning simple

Décembre 2024

Jules Besson, Antoine Delcambre, Loïc Blocquel, Thomas Pham, Aubin Maillard

Introduction et objectifs

Les marchés financiers sont des environnements complexes influencés par une multitude de facteurs économiques, techniques et comportementaux.

Dans le cadre de ce projet, nous allons développer un projet de machine learning pour à terme prédire les prix et les rendements des actifs du S&P 500.

Dans un premier temps, nous collecterons les données historiques relatives au S&P500, que nous traiterons afin de développer des modèles prédictifs permettant de prédire les variations des actifs du S&P 500.

Cette étude vise à développer une approche robuste de prévision permettant d'éclairer les stratégies d'investissement tout en intégrant des indicateurs financiers et macroéconomiques pertinents.

Objectifs du projet :

1. Collecter et préparer des données historiques sur le S&P 500, incluant des variables exogènes comme le VIX, les taux d'intérêt et d'autres indicateurs économiques.
2. Mettre en œuvre plusieurs modèles de machine learning adaptés aux séries temporelles et aux relations non linéaires.
3. Analyser les performances des modèles à l'aide de métriques d'évaluation pertinentes.
4. Interpréter les résultats pour dégager des recommandations d'investissement basées sur les prédictions.

Méthodologie

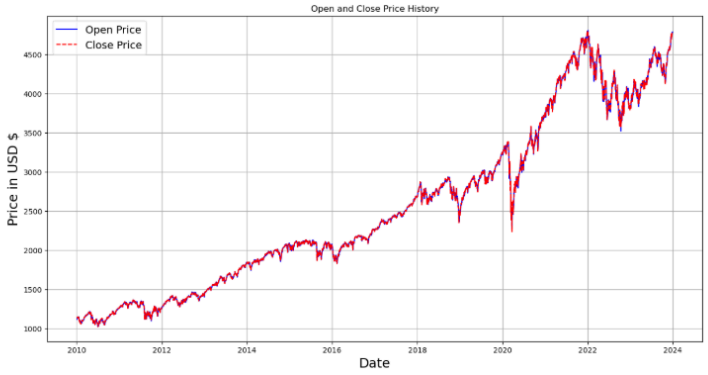
On séparera la méthodologie suivant 3 grandes parties, la première étant la collecte des données, la préparation des données, puis le choix et le développement du modèle prédictif.

I) Collectes des données

La première étape a consisté à collecter et à préparer les données nécessaires pour notre analyse :

```
df = yf.download('^GSPC', start=start_date, end=end_date)
```

Le but étant de télécharger tous les prix historiques du S&P 500 via Yahoo Finance. Elles incluent par exemple, le prix d'ouverture, de clôture, le prix minimum et maximum de la journée.



On ajoute également d'autres données comme des variables exogènes pour permettre une évaluation complète. Ici nous avons rajouté :

- L'indice de volatilité (VIX)
- Taux d'intérêt à 10 ans (Reflète les conditions de marché à long terme)
- Taux de chômage (Représentent les données macroéconomiques)
- Indice sur l'inflation

II) Prétraitement des données

Cette étape est essentielle pour permettre de les utiliser prochainement correctement dans les modèles prédictifs. En effet, étant donné le nombre important de données, il est essentiel de les prétraitées pour corriger les valeurs manquantes, supprimer les doublons et transformer les prix en rendements logarithmiques.

En illustration, on remplit les valeurs manquantes (NaN) par des interpolations ou suppressions des lignes, par exemple avec ce genre de code :

```
df['Treasury_Yield_10y'].interpolate(method='linear')
```

.

On vérifiera ensuite la qualité de notre data, voir si elle doit être de nouveau soumise à de potentielles modifications.

De par cette étape, on pourra observer si certaines colonnes possèdent encore des valeurs manquantes, ou NaN, et les remplacer par une valeur réelle pour pouvoir les exploiter.

```
# Vérification des corrections

df.isnull().sum()
```

```
Price
,Close                    0
,High                     0
,Low                      0
,Open                     0
,Volume                   0
,VIX                      0
,VIX_diff                 0
,Treasury_Yield_10y       0
,Treasury_Yield_10y_diff  0
,Unemployment_Rate        0
,Unemployment_Rate_diff   0
,CPI                      0
,CPI_diff                 0
,Close J+1                0
,Log_Returns              0
,dtype: int64
```

Close J+1	Log_Returns
1136.520020	NaN
1137.140015	0.003111
1141.689941	0.000545
1144.979980	0.003993
1146.979980	0.002878
...	...
4774.750000	0.001659
4781.580078	0.004223
4783.350098	0.001429
4769.830078	0.000370
NaN	-0.002830

Enfin, on vérifie une dernière fois nos corrections, nous permettant d'avoir l'ensemble de nos données traitées.

Il nous faut également crée une **nouvelle colonne J+1** pour calculer notre **rendement logarithmique** permettant de vérifier si notre modèle nous donne de bon résultats :

(A savoir qu'ici la colonne J+1 est créé avant les différentes corrections d'où l'apparition des différentes valeur NaN dans les deux colonnes)

Enfin, la dernière étape du prétraitement des données et de faire une analyse de la stationnarité (ADF) afin que nos modèles puissent être appliqués (comme par exemple le modèle ARIMA) .

Ici, nous avons effectué deux tests de stationnarité, un test du processus par rapport au rendement logarithmique, l'autre test du processus close.

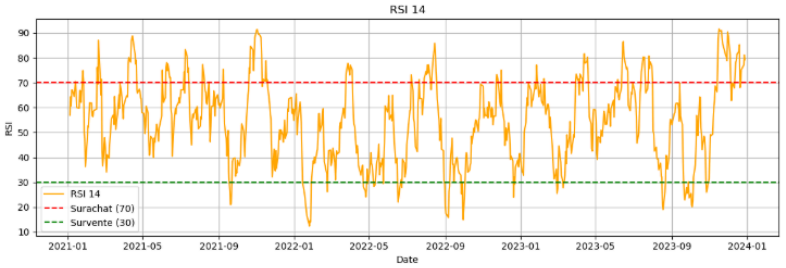
On observe alors que la p-value du test de l'ADF du processus par rapport au rendement logarithmique est largement inférieur à 0.05 (P-value : 4.266429641242552e-24). L'hypothèse nulle est donc rejetée, le processus est donc stationnaire par rapport au rendement logarithmique.

Cependant, le processus(close) a une p-value largement supérieur à 0.05. Le processus a donc une racine unitaire et n'est donc pas stationnaire, on ne pourra donc pas appliquer des modèles AMIRA

III) Feature Engineering

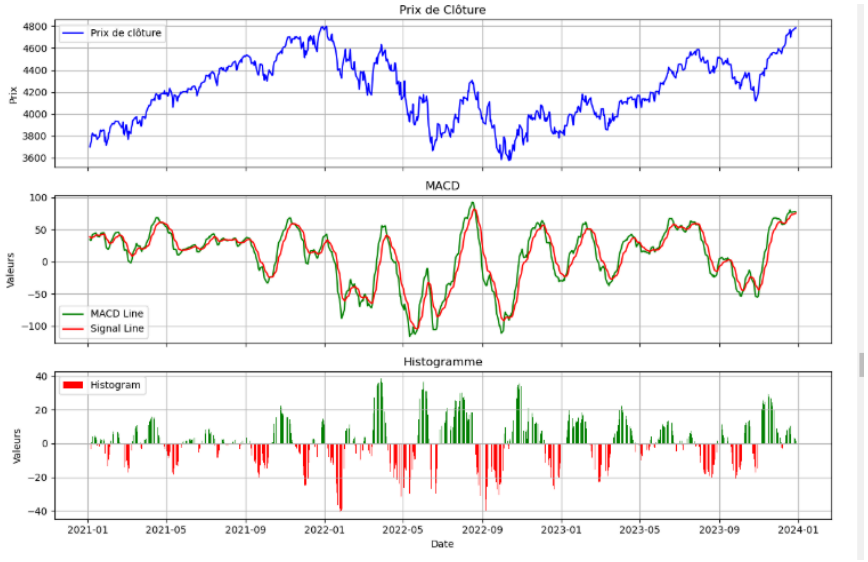
Enfin, un travail de feature engineering a été effectué pour enrichir les données avec des indicateurs techniques comme :

- **RSI** (Relative Strength Index) : indicateur qui permet d'identifier les zones de surachat ou de survente. Communément, lorsque sa valeur est supérieure à 70, on parle souvent d'une situation de surachat et en dessous de 30, on parle de situation



- **Moyennes mobiles (MA)** et moyennes mobiles exponentielles (EMA) : une moyenne mobile donne la valeur moyenne des cours sur une période donnée. Cet indicateur permet de s'affranchir des aberrations des cours en les « lissant ». Une moyenne mobile à 20 jours donne le cours moyen sur les vingt dernières séances. Une moyenne mobile exponentielle accorde une plus forte pondération aux cours les plus récents ce qui la rend plus réactive. On trace plusieurs moyennes mobiles sur différentes périodes.

- **MACD** (Moving Average Convergence Divergence).

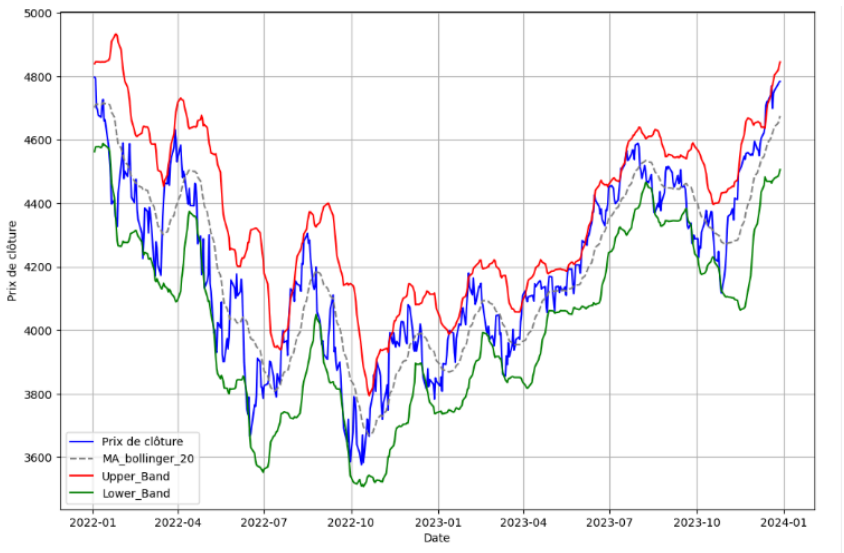


Le MACD est un indicateur technique qui mesure la différence entre deux moyennes mobiles exponentielles pour détecter les tendances du marché et les signaux d'achat ou de vente.

Lorsque la MACD Line croise la Signal Line à la hausse : signal d'achat. Lorsque la MACD Line croise la Signal Line à la baisse : signal de vente.

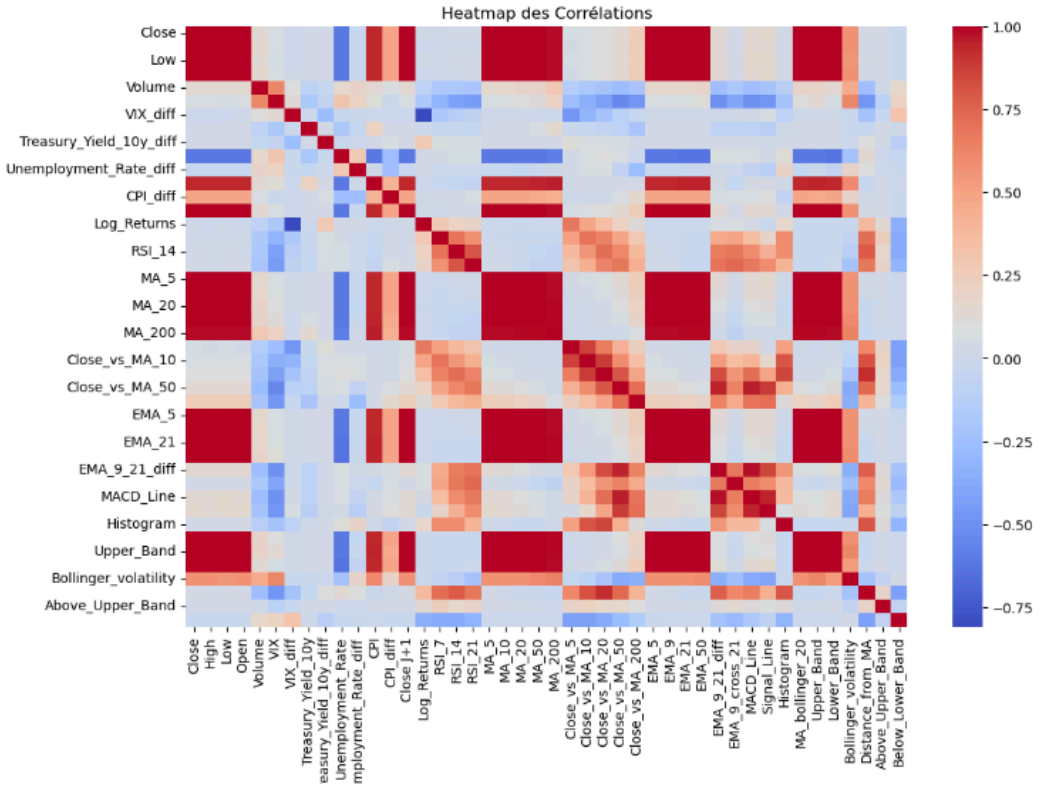
HISTOGRAMME : Positif : tendance haussière. Négatif : tendance baissière.

- **Bandes de Bollinger** : Les bandes de Bollinger sont composées de trois lignes sur le graphique d'un trader. La ligne centrale de l'indicateur est la moyenne mobile individuelle (SMA) du prix de l'instrument. La bande supérieure est la SMA plus deux écarts-types. La bande inférieure est la SMA moins deux écarts-types. C'est un outil d'analyse technique très utilisé. L'écartement de la bande supérieur et inférieur reflète la volatilité.



Comme précédemment, on fait un retraitement des données pour enlever et traiter les valeur nulles / Nan pour permettre une analyse complète de ces indicateurs techniques

Enfin, afin de sélectionner les variables les plus pertinente, il est intéressant de faire une analyse de corrélation et des test de multicolinéarité (VIF) par exemple avec une Heatmap des corrélations :



On voit alors que certaines variables sont fortement corrélées entre elle tels que Close, High, Low, et Open, ainsi que les moyennes mobiles (MA_*) et exponentielles (EMA*)

On peut alors réduire la multicolinéarité en gardant seulement quelques indicateurs pertinents parmi eux comme Close et une ou deux moyennes mobiles (par exemple, MA_ 50 et EMA 21).

On se doit tout de même de conserver les indicateurs macroéconomiques pertinents : comme CPI, CPI_diff, et Unemployment_Rate, car ils apportent une information complémentaire (faible colinéarité avec les prix) et peuvent influencer la direction du marché.

On peut également explorer des relations non linéaires : Certaines variables comme Volume, MACD_Line, RSI, et Bollinger Volatility pourraient avoir des relations non linéaires avec Close J+1. Des modèles comme les forêts aléatoires ou le gradient boosting peuvent exploiter ces relations.

En faisant alors un test de multicolinéarité (VIF) on en tire alors seulement certaines variables les plus pertinentes pour nos futures modèles :

	feature	VIF
4	Volume	17.086621
5	RSI_14	15.835945
0	Close	14.216241
2	Bollinger_volatility	5.493725
3	Distance_from_MA	3.132855
1	MACD_Line	2.822538
6	CPI_diff	2.115283
8	Unemployment_Rate_diff	1.251751
7	VIX_diff	1.194962
9	Treasury_Yield_10y_diff	1.067096

Partie 2 : Développement du Modèle Prédictif

Choix des modèles

Dans ce projet, nous avons exploré plusieurs approches de modélisation pour prédire les rendements et les mouvements des actifs du S&P 500. À travers une analyse empirique, nous avons testé des modèles allant de la régression linéaire à des approches non linéaires et basées sur les séries temporelles. Voici un résumé des méthodes employées et du fil de réflexion fait :

Tout d'abord**, Les modèles** ont été **évalués** à l'aide **des métriques suivantes :**

- **MAE (Mean Absolute Error)** : Mesure la moyenne des erreurs absolues.
- **RMSE (Root Mean Squared Error)** : Privilégie les grandes erreurs en les pondérant davantage.
- **Direction Accuracy** : Mesure la capacité du modèle à prévoir correctement les hausses et baisses.

C'est à l'aide de ces métriques que nous évaluerons la performance des modèles choisis.

Voici maintenant les différents **modèles et les métriques d'évaluation** qui en découle**:**

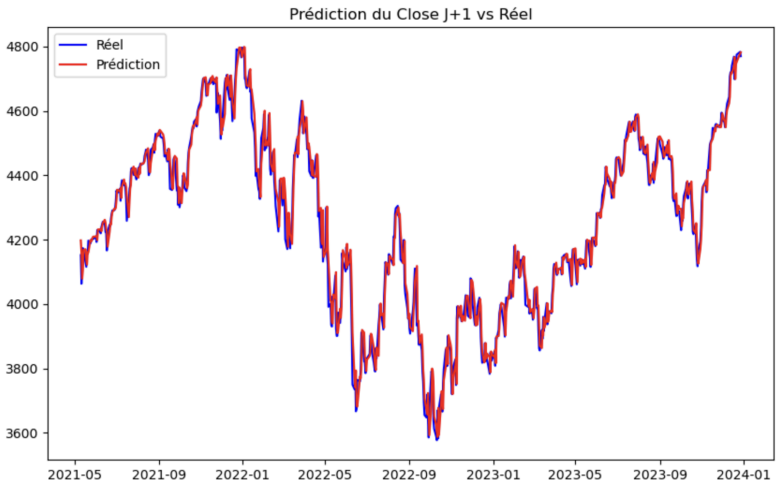
- **Régression Linéaire :** Ce modèle a été utilisé pour prédire le prix de Close J+1 et a servi de baseline pour évaluer la performance des autres approches.

```python
# 1. Définir X (features) et y (target)
X = df[['Close']]  # Utilise uniquement la colonne 'Close'
y = df['Close J+1']  # La cible est le 'Close' du jour suivant

# 2. Séparation des données en train (80%) et test (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# 3. Initialiser et entraîner le modèle de régression linéaire
model = LinearRegression()
model.fit(X_train, y_train)

# 4. Prédire les valeurs sur l'ensemble de test
y_pred = model.predict(X_test)

# 5. Évaluation du modèle
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f'Mean Absolute Error (MAE): {mae}')
print(f'Root Mean Squared Error (RMSE): {rmse}')

# 6. Visualisation des prédictions vs réels
plt.figure(figsize=(10,6))
plt.plot(y_test.index, y_test, label='Réel', color='blue')
plt.plot(y_test.index, y_pred, label='Prédiction', color='red')
plt.legend()
plt.title('Prédiction du Close J+1 vs Réel')
plt.show()
```

Dans ce premier essai, on a utilisé un modèle de régression linéaire pour prédire le prix de fermeture du jour suivant (Close J+1) en sebasant uniquement sur le prix de fermeture du jour actuel (Close). Nous avons choisi cette approche simple pour avoir une première estimation rapide, mais on sait que ce modèle est assez basique. En effet, il ne prend en compte que la relation linéaire directe entre ces deux valeurs, sans intégrer d'autres facteurs comme les tendances sous-jacentes, la saisonnalité ou les événements extérieurs qui pourraient influencer le marché. C'est pour cela que l'erreur de prédiction est assez élevée. Le modèle peut être qualifié de "naïf" car il ne capture pas la complexité des données financières. Malgré cela, il permet d'obtenir une première approximation, mais nous sommes conscient qu'une approche plus sophistiquée sera nécessaire pour améliorer les résultats. C’est ce qu’on va essayer de faire par la suite.

Analyse et interprétations:

```
Mean Absolute Error (MAE): 35.542932690916516
Root Mean Squared Error (RMSE): 47.19075718529133
```



empire les choses en ajoutant des variable

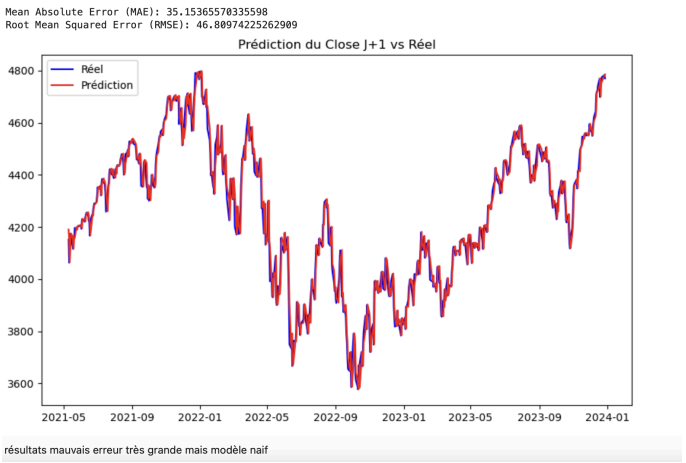
```python
# Sélectionner les variables pour X (features) et y (target)
X = df[['Close', 'Volume', 'RSI_14', 'Bollinger_volatility', 'MACD_Line', 'VIX_diff', 'CPI_diff', 'Distance_from_MA']]
y = df['Close J+1']

# Diviser les données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# Standardiser les données (important pour les modèles linéaires)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Entraîner le modèle
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Faire des prédictions
y_pred = model.predict(X_test_scaled)

# Évaluer le modèle
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f'Mean Absolute Error (MAE): {mae}')
print(f'Root Mean Squared Error (RMSE): {rmse}')

# 6. Visualisation des prédictions vs réels
plt.figure(figsize=(10,6))
plt.plot(y_test.index, y_test, label='Réel', color='blue')
plt.plot(y_test.index, y_pred, label='Prédiction', color='red')
plt.legend()
plt.title('Prédiction du Close J+1 vs Réel')
plt.show()
```

Dans ce deuxième essai, nous avons ajouté plusieurs variables supplémentaires pour améliorer la prédiction du prix de fermeture du jour suivant (Close J+1). En plus du prix de fermeture actuel (Close), nous avons inclus des variables telles que le volume des transactions, le RSI (Relative Strength Index) à 14 jours, la volatilité des bandes de Bollinger, la ligne MACD, les différences avec l'indice VIX et le CPI, ainsi que la distance par rapport à la moyenne mobile. Ces nouvelles variables sont censées capturer davantage d'informations qui pourraient influencer le prix du marché, ce qui théoriquement devrait améliorer la précision du modèle.

Cependant, après avoir standardisé les données pour que toutes les variables aient une échelle comparable, nous avons constaté que l'ajout de ces variables a finalement empiré les résultats. L'erreur de prédiction a en réalité augmenté, comme en témoignent les nouvelles valeurs de l'Erreur Absolue Moyenne (MAE) et de l'Erreur Quadratique Moyenne (RMSE), qui sont légèrement plus élevées que dans le premier modèle. Cela pourrait être dû à plusieurs facteurs, comme la multicolinéarité entre certaines des nouvelles variables, ou simplement le fait que ces variables n'ajoutent pas de valeur prédictive réelle pour ce modèle spécifique.

En résumé, bien que l'ajout de ces variables ait l'intention d'améliorer le modèle, cela semble avoir eu un effet négatif. Il est possible qu'une analyse plus approfondie des relations entre ces variables et le prix de fermeture soit nécessaire, ou qu'un autre type de modèle, plus complexe, soit mieux adapté pour exploiter pleinement ces informations.

Analyse et interprétations:



- Dans ce modèle **Random Forest Regressor**, l'objectif est de prédire le prix de fermeture du jour suivant (Close J+1) à partir du prix de clôture actuel (Close). Contrairement au modèle de régression linéaire, le modèle Random Forest utilise une approche basée sur l'agrégation de plusieurs arbres décisionnels, ce qui lui permet de capturer des relations non linéaires et d'améliorer les performances par rapport à un simple modèle linéaire.

  Analyse et interprétations:

  ```
  Mean Absolute Error (MAE) - XGBoost: 62.85284545236212
  Root Mean Squared Error (RMSE) - XGBoost: 87.14112762756005
  ```

  Le modèle **Random Forest** a montré une erreur de prédiction relativement élevée, avec un **MAE de 56.39** et un **RMSE de 79.12**. Bien que ces résultats soient moins bons que ceux attendus, le modèle Random Forest reste robuste et capable de capturer des relations non linéaires dans les données. Cependant, la performance pourrait encore être améliorée par des ajustements de paramètres, l'intégration de plus de variables explicatives, ou même par l'exploration d'autres modèles plus complexes. Le fait que les erreurs restent relativement élevées pourrait indiquer qu'il existe d'autres facteurs non capturés par le modèle actuel, ou que la nature des données nécessite une approche plus fine

- Le modèle **XGBoost** a été entraîné en utilisant les mêmes variables explicatives que dans les modèles précédents, avec une validation croisée temporelle pour maintenir l'ordre des données.

  L'ajout de l'algorithme XGBoost, plus complexe et capable de gérer des relations non linéaires et a pour objectifs d' améliorer les prédictions par rapport aux modèles plus simples.

Analyse et **interprétations:**

```
Mean Absolute Error (MAE) - XGBoost: 62.85284545236212
Root Mean Squared Error (RMSE) - XGBoost: 87.14112762756005
```

Bien que XGBoost soit un modèle robuste et souvent performant pour des données complexes, ici l'erreur est relativement élevée. Cela pourrait être dû à un manque d'optimisation des hyperparamètres ou à la nature non linéaire des données, qui nécessite peut-être plus de features ou un ajustement des paramètres pour de meilleures performances.

- Un **Voting Regressor** a été créé pour combiner les prédictions de trois modèles différents : la régression linéaire, Random Forest et XGBoost. L'idée était d'exploiter les forces de chaque modèle pour obtenir des prédictions plus fiables.

  La combinaison de plusieurs modèles permet souvent de réduire le sur-apprentissage (overfitting) tout en améliorant la généralisation en théorie bien sûr.

  Analyse et **interprétations:**

```
MAE (Voting): 48.581835891430266
RMSE (Voting): 64.89686954436961
```

  L'agrégation a réussi à réduire l'erreur par rapport aux modèles individuels, ce qui démontre que le **Voting Regressor** améliore la performance en combinant plusieurs techniques. Cependant, l'erreur reste encore relativement importante, ce qui suggère que l'ajout d'autres variables ou une meilleure gestion des hyperparamètres serait nécessaire pour de meilleurs résultats.

- Une nouvelle variable **'Movement'** a été créée en calculant la différence entre le prix de clôture du jour suivant et le prix de clôture actuel. Ensuite, une colonne catégorielle 'Direction' a été ajoutée pour indiquer si le prix a augmenté, diminué ou est resté stable.

```python
df['Movement'] = df['Close J+1'] - df['Close']

# Ajouter une colonne catégorielle 'Direction' pour indiquer le mouvement
# 'Up' si Close J+1 > Close, 'Down' si Close J+1 < Close, 'No Change' sinon
df['Direction'] = df['Movement'].apply(lambda x: 1 if x > 0 else (0 if x <= 0 else 'No Change'))

# Vérification des premières lignes
print(df[['Close', 'Close J+1', 'Movement', 'Direction']].head())
```

  Analyse et **interprétations:**

```
Mean Absolute Error (MAE): 0.4990237324583611
Root Mean Squared Error (RMSE): 0.5015320254420413
Accuracy: 0.51
```

  Bien que l'ajout des nouvelles variables ait permis d'améliorer légèrement l'exactitude des prédictions, les erreurs (MAE et RMSE) restent élevées, ce qui

indique que le modèle pourrait ne pas encore capturer efficacement les dynamiques sous-jacentes du marché. Le fait que l'**accuracy** soit à peine au-dessus de 50% montre que le modèle n'arrive pas à faire mieux qu'une prédiction aléatoire dans de nombreux cas.

- Le modèle **Random Forest a été mis à jour** pour inclure des variables supplémentaires comme le volume des transactions, le RSI (Relative Strength Index), la volatilité des bandes de Bollinger, le MACD, la distance par rapport à la moyenne mobile (Distance_from_MA), ainsi que d'autres indicateurs économiques. La validation croisée temporelle a été maintenue pour garantir une gestion correcte des séries temporelles.

```
# Sélectionner les variables pour X (features) et y (target)
X = df[['Close', 'Volume', 'RSI_14', 'Bollinger_volatility', 'MACD_Line', 'VIX_diff', 'CPI_diff', 'Distance_from_MA']]
y = df['Direction']

tscv = TimeSeriesSplit(n_splits=5)

for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

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

# Créer le modèle RandomForest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Entraîner le modèle
rf_model.fit(X_train_scaled, y_train)

# Faire des prédictions
y_pred_rf = rf_model.predict(X_test_scaled)

# 6. Convertir les prédictions en catégories
y_pred_categories = np.where(y_pred_rf > 0.5, 1, 0)

# 7. Évaluation du modèle
mae = mean_absolute_error(y_test, y_pred_rf)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_rf))
accuracy = accuracy_score(y_test, y_pred_categories)

print(f'Mean Absolute Error (MAE): {mae}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'Accuracy: {accuracy:.2f}')
```

L'objectif était d'ajouter des variables pertinentes pour capter des informations supplémentaires qui pourraient mieux expliquer les variations du marché.

Analyse et **interprétations:**

```
Mean Absolute Error (MAE): 0.5077396021699818
Root Mean Squared Error (RMSE): 0.5245102770598594
Accuracy: 0.47
```

L'ajout de plus de variables explicatives a conduit à une légère augmentation de l'erreur de prédiction (MAE et RMSE), et l'**accuracy** a diminué par rapport au modèle de Voting. Cela suggère que l'ajout de trop de variables peut entraîner de la **multicolinéarité** (ce qui peut perturber le modèle), ou que les nouvelles variables ne sont pas assez informatives pour le modèle. Cela peut aussi signifier qu'un autre modèle plus adapté à ces données serait nécessaire.

Analyse de la multicolinéarité (VIF)

Objectif : Vérifier la présence de multicolinéarité parmi les variables explicatives pour éviter qu'elles n'interfèrent entre elles, ce qui pourrait altérer la performance du modèle.

Le **Variance Inflation Factor (VIF)** a été calculé pour chaque variable afin d'identifier les variables hautement corrélées et potentiellement problématiques.

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Sélection des features sans la cible
X = df.drop(columns=['Direction','Close J+1','Movement'])

# Calcul du VIF
vif_data = pd.DataFrame()
vif_data['feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Affichage des résultats
print(vif_data.sort_values(by='VIF', ascending=False))
```

```
Corrélation des variables à Direction Direction                1.000000
Movement                    0.618378
Unemployment_Rate           0.037257
Below_Lower_Band            0.023560
VIX_diff                    0.019796
VIX                         0.012309
Volume                      0.006240
Close_vs_MA_200             0.004912
Signal_Line                 0.001175
Unemployment_Rate_diff     -0.003304
MACD_Line                  -0.003882
Close J+1                  -0.005859
CPI_diff                   -0.006469
EMA_9_21_diff              -0.006765
EMA_9_cross_21             -0.007375
Close_vs_MA_50             -0.010587
Histogram                  -0.015159
RSI_21                     -0.015435
Bollinger_volatility       -0.018557
Close_vs_MA_20             -0.019418
RSI_14                     -0.022025
Lower_Band                 -0.022727
RSI_7                      -0.022803
MA_bollinger_20            -0.023134
MA_20                      -0.023134
MA_10                      -0.023266
EMA_21                     -0.023310
Upper_Band                 -0.023443
EMA_9                      -0.023444
MA_50                      -0.023489
EMA_50                     -0.023527
MA_5                       -0.023559
EMA_5                      -0.023655
```

Des variables telles que 'Close', 'MA_10', 'MA_50' ont montré des VIF infinis ou très élevés, indiquant une forte multicolinéarité. D'autres variables comme 'Close_vs_MA' et plusieurs moyennes mobiles ont également montré des VIF élevés.

La présence de multicolinéarité peut être la raison pour laquelle l'ajout de ces variables n'a pas amélioré le modèle. La multicolinéarité rend difficile pour les modèles d'identifier des relations claires entre les variables et la cible. Cela suggère que l'on pourrait réduire le nombre de variables ou appliquer une régularisation pour atténuer cet effet.