Université Abdelmalek Essaadi
Faculté des sciences et techniques Tanger
Logiciels & Systèmes Intelligents

Projet de Machine Learning.

Réalisé par :

Soukaina BOUCHANE

Nassima EL JAZOULI

Encadré par :

Pr. Lotfi EL AACHAK

Remerciements :

L'année universitaire touchant à sa fin, nous tenions à vous dire que vous avez été un excellent professeur ! Nous vous remercions d'avoir partagé vos connaissances avec nous, d'avoir toujours été juste dans votre éducation monsieur Lotfi EL AACHAK et de nous avoir toujours soutenus et aidés. Merci encore de la part de tous vos étudiants.

Tables des matières :

Table des figures :

I. Introduction :

Selon la capitalisation boursière, le JPX TOKYO est considérée comme la quatrième bourse mondialement. Le covid19 a affecté tous les marchés financiers et afin que le marché financier peut ressentir la fraicheur, cette compétition était mise à notre disposition afin de prédire les rendements futurs des actions.

Il existe de nombreux efforts de trading quantitatifs utilisés pour analyser les marchés financiers et formuler des stratégies d'investissement. Pour créer et exécuter une telle stratégie, il faut à la fois des données historiques et en temps réel, ce qui est difficile à obtenir, en particulier pour les investisseurs particuliers. Ce concours fournira des données financières pour le marché japonais, permettant aux investisseurs particuliers d'analyser le marché dans toute sa mesure.

Japan Exchange Group, Inc. (JPX) est une société holding exploitant l'une des plus grandes bourses du monde, la Bourse de Tokyo (TSE), et les bourses de dérivés Osaka Exchange (OSE) et Tokyo Commodity Exchange (TOCOM). JPX organise ce concours et est soutenu par la société de technologie AI AlpacaJapan Co.,Ltd.

II. Objectif de la compétition :

Cet ensemble de données contient des données historiques pour une variété d'actions et d'options japonaises. Notre objectif est de prédire les rendements futurs des actions.

III. Vue d'ensemble sur nos données :

Comme les cours historiques des actions ne sont pas confidentiels, il s'agira d'un concours de prévisions utilisant l'API de séries chronologiques. Les données de la période de classement public sont incluses dans l'ensemble de données de la compétition.

Voici une vue d'ensemble de notre hiérarchie de données :

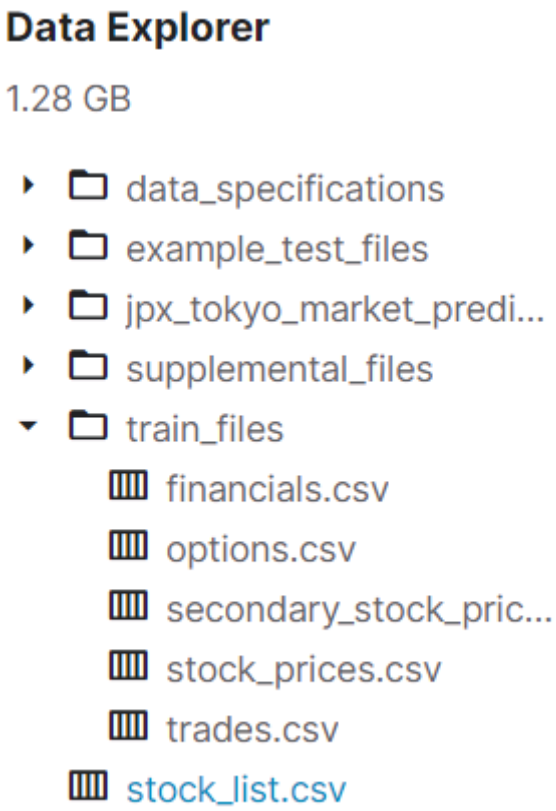


Figure 1 Vue d'ensemble de notre hiérarchie de données.

1. Dossiers :

data_specifications/ - Définitions pour les colonnes individuelles.

jpx_tokyo_market_prediction/ Fichiers qui activent l'API. Attendez-vous à ce que l'API fournisse toutes les lignes en moins de cinq minutes et réserve moins de 0,5 Go de mémoire.

Des copies des fichiers de données existent dans plusieurs dossiers qui couvrent différentes fenêtres temporelles et servent à des fins différentes.

train_files/ Dossier de données couvrant la période de formation principale.

supplemental_files/ Dossier de données contenant une fenêtre dynamique de données d'entraînement supplémentaires. Celui-ci sera mis à jour avec de nouvelles données pendant la phase principale du concours début mai, début juin et environ une semaine avant le verrouillage des soumissions.

example_test_files/ Dossier de données couvrant la période de test public. Destiné à faciliter les tests hors ligne. Inclut les mêmes colonnes fournies par l'API (c'est-à-dire aucune **Targetcolonne**). Vous pouvez calculer la **Targetcolonne** à partir de la **Closecolonne** ; c'est le rendement de l'achat d'une action le lendemain et de la vente le surlendemain. Ce dossier comprend également un exemple d'exemple de fichier de soumission qui sera fourni par l'API.

2. Fichiers :

stock_prices.csv Le fichier principal qui vous intéresse. Comprend le cours de clôture quotidien de chaque action et la colonne cible.

options.csv Données sur le statut d'une variété d'options basées sur le marché plus large. De nombreuses options incluent des prédictions implicites du prix futur du marché boursier et peuvent donc être intéressantes même si les options ne sont pas notées directement.

Secondary_stock_prices.csv L'ensemble de données de base contient les 2 000 actions les plus couramment négociées, mais de nombreux titres moins liquides sont également négociés sur le marché de Tokyo. Ce fichier contient des données pour ces titres, qui ne sont pas notés mais qui peuvent être intéressants pour évaluer le marché dans son ensemble.

trades.csv Résumé agrégé des volumes de transactions de la semaine ouvrée précédente.

financials.csv Résultats des rapports trimestriels sur les résultats.

stock_list.csv - Mappage entre les **SecuritiesCode** noms de société et, ainsi que des informations générales sur le secteur d'activité de la société.

Le fichier qui va nous intéresser le plus c’est le fichier ***stock_prices.csv*** qui est notre fichier principal dans cette compétition.

3. Stock_prices.csv :

Ce fichier est composé de 16 colonnes qui sont les suivant :

RowId, Date, SecuritiesCode, Open, High, Low, Close, Volume, AdjustmentFactor, ExpectedDividend, SupervisionFlag, Target.

RowId: ID unique des enregistrements de prix.

Date: Date de transaction.

SecuritiesCode: Code des valeurs mobilières local.

Open: Le premier prix négocié sur une journée.

High: Le prix négocié le plus élevé sur une journée.

Low: Le prix négocié le plus bas sur une journée.

Close: Le dernier prix négocié un jour.

Volume: Le nombre d'actions échangées sur une journée.

AdjustmentFactor: Un facteur utilisé pour calculer le prix/volume théorique en cas de fractionnement/split inversé (n'incluant PAS le dividende/l'attribution d'actions).

ExpectedDividend: La valeur attendue du dividende à la date ex. Cette valeur est constatée 2 jours ouvrés avant la date de détachement.

SupervisionFlag: Un drapeau des titres sous surveillance et des titres à radier.

Designation Date	Issue Name	Code	Market Segment*	Removal Date	Details	Remarks
May 20, 2022	Mutual Corporation	2773	Standard	-	Designation of Securities Under Supervision (Confirmation)	
May 18, 2022	Arte Salon Holdings,Inc.	2406	Standard	-	Delisting decision & designation of Securities to Be Delisted	
May 16, 2022	KK DI-NIKKO ENGINEERING	6635	Standard	-	Designation of Securities Under Supervision (Confirmation)	
May 16, 2022	KITO CORPORATION	6409	Prime	-	Designation of Securities Under Supervision (Confirmation)	
May 13, 2022	JALUX Inc.	2729	Standard	-	Delisting decision & designation of Securities to Be Delisted	

Figure 2 Drapeau des titres sous surveillance et des titres à radier.

Target: Le taux de variation du cours de clôture ajusté entre t+2 et t+1 où t+0 est la date de transaction.

IV. Outils utilisés :

1. Google Colab :

Google Colab ou Colaboratory est un service cloud, offert par Google (gratuit), basé sur Jupyter Notebook et destiné à la formation et à la recherche dans l'apprentissage automatique. Cette plateforme permet d'entraîner des modèles de Machine Learning directement dans le cloud. Sans donc avoir besoin d'installer quoi que ce soit sur notre ordinateur à l'exception d'un navigateur.

Google colab est un outil complet pour entraîner rapidement et tester rapidement des modèles d'apprentissage automatique sans avoir de contrainte matérielle. Il offre 107 Go de stockage et 12 Go de RAM.

Figure 3 Google Colab.

2. Connexion entre Google Colab et Kaggle :

Cette étape s'effectue en installant d'abord kaggle et en uploadant le fichier kaggle.json généré dans mon profile kaggle, puis on a créé le dossier .kaggle et on copie ce fichier en dedans de lui, afin de lui donner les droits et télécharger nos données en moins de 5 minutes et en allouant 0,5 Go de mémoire.

Figure 4 Connexion avec Kaggle.

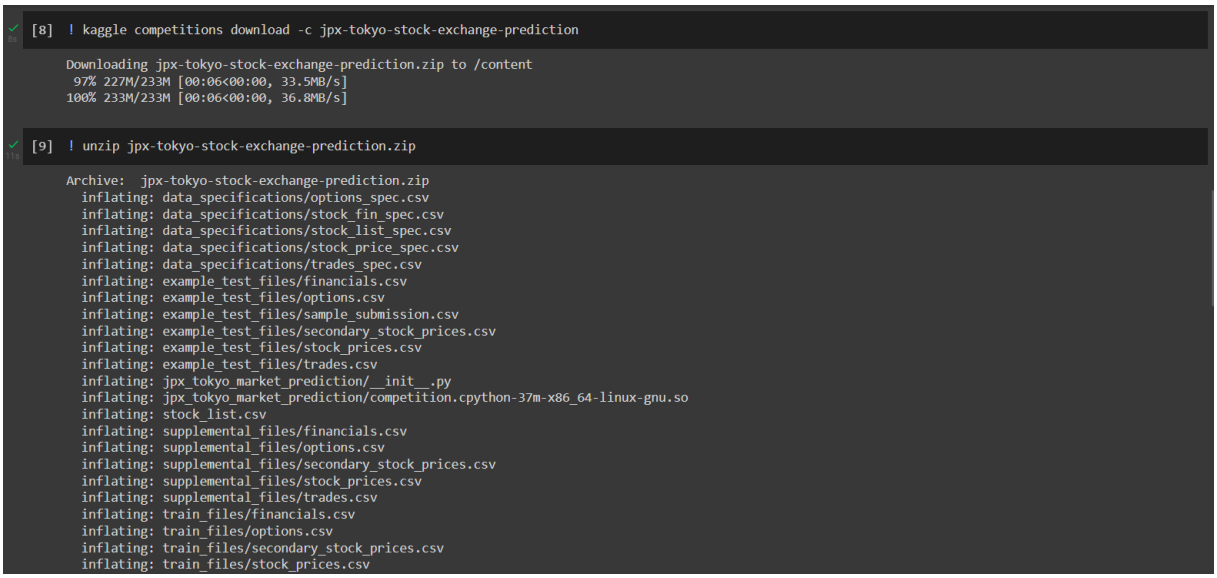


Figure 5 Téléchargement des données de la compétition.

Voici l'arborescence des données après tous ces bouts de code :

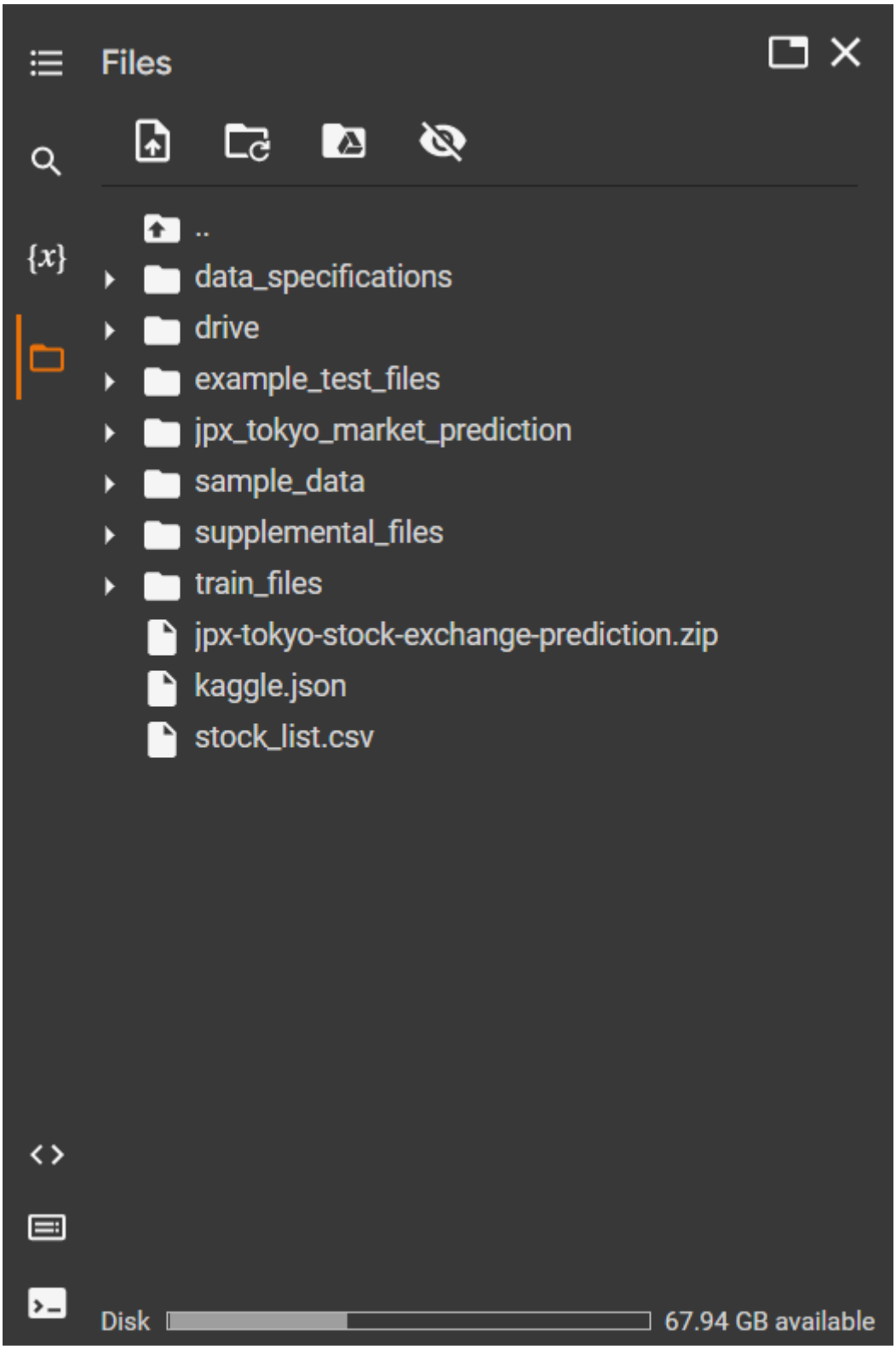


Figure 6 L'arborescence des données.

V. JPX Tokyo Stock Exchange Prediction en utilisant la régression linéaire :

1. Le traitement initial des données :

L'importation des bibliothèques dont on a besoin :

```
[10] import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from pandas.plotting import scatter_matrix
```

Figure 7 L'importation des bibliothèques.

Les données sur lesquelles on travaille sont de type timeseries, qui sont mis à jour pendant cette compétition 3 fois, la première c'était au début mai, la deuxième ça sera au début juin et la troisième, ça sera une semaine avant la fin de la compétition.

```
[11] url = '/content/train_files/stock_prices.csv'
     url1 = '/content/supplemental_files/stock_prices.csv'
     dataset = pd.read_csv(url)
     dataset_supp = pd.read_csv(url1)
     df = pd.concat([dataset, dataset_supp])
```

Figure 8 L'importation des données depuis stock_prices.csv.

D'abord on supprime les valeurs dupliquées par rapport la Date et le SecuritiesCode :

```
[12] df.drop_duplicates(['Date','SecuritiesCode'],keep='first',inplace=True)
```

```
[13] df
```

	RowId	Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	SupervisionFlag	Target
0	20170104_1301	2017-01-04	1301	2734.0	2755.0	2730.0	2742.0	31400	1.0	NaN	False	0.000730
1	20170104_1332	2017-01-04	1332	568.0	576.0	563.0	571.0	2798500	1.0	NaN	False	0.012324
2	20170104_1333	2017-01-04	1333	3150.0	3210.0	3140.0	3210.0	270800	1.0	NaN	False	0.006154
3	20170104_1376	2017-01-04	1376	1510.0	1550.0	1510.0	1550.0	11300	1.0	NaN	False	0.011053
4	20170104_1377	2017-01-04	1377	3270.0	3350.0	3270.0	3330.0	150800	1.0	NaN	False	0.003026
...	...	...	...	...	...	...	...	...	...	...	...	...
195988	20220428_9990	2022-04-28	9990	497.0	504.0	494.0	504.0	51900	1.0	NaN	False	0.024048
195989	20220428_9991	2022-04-28	9991	776.0	786.0	774.0	784.0	23400	1.0	NaN	False	0.011378
195990	20220428_9993	2022-04-28	9993	1520.0	1530.0	1510.0	1517.0	8100	1.0	NaN	False	-0.005941
195991	20220428_9994	2022-04-28	9994	2471.0	2489.0	2457.0	2485.0	7800	1.0	NaN	False	0.005602
195992	20220428_9997	2022-04-28	9997	717.0	731.0	714.0	731.0	149800	1.0	NaN	False	-0.013680

2528524 rows × 12 columns

Figure 9 Suppression des valeurs dupliquées.

On passe pour afficher le résumé statistique de nos données :

```
df.describe()
```

	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	Target
count	2.528524e+06	2.520325e+06	2.520325e+06	2.520325e+06	2.520325e+06	2.528524e+06	2.528524e+06	20821.000000	2.528282e+06
mean	5.893926e+03	2.596964e+03	2.629022e+03	2.563411e+03	2.596341e+03	6.866332e+05	1.000464e+00	23.019841	3.689221e-04
std	2.403049e+03	3.624991e+03	3.667853e+03	3.579982e+03	3.623962e+03	3.819717e+06	6.506888e-02	33.240956	2.343145e-02
min	1.301000e+03	1.400000e+01	1.500000e+01	1.300000e+01	1.400000e+01	0.000000e+00	1.000000e-01	0.000000	-5.785414e-01
25%	3.891000e+03	1.021000e+03	1.034000e+03	1.008000e+03	1.021000e+03	3.050000e+04	1.000000e+00	5.500000	-1.062888e-02
50%	6.237000e+03	1.810000e+03	1.832000e+03	1.788000e+03	1.809500e+03	1.064000e+05	1.000000e+00	15.000000	0.000000e+00
75%	7.962000e+03	3.020000e+03	3.060000e+03	2.987000e+03	3.020000e+03	4.003000e+05	1.000000e+00	30.000000	1.056052e-02
max	9.997000e+03	1.099500e+05	1.105000e+05	1.072000e+05	1.095500e+05	6.436540e+08	2.000000e+01	1080.000000	1.119512e+00

Figure 10 Le résumé statistique de nos données.

Passant maintenant pour afficher s'il y a des valeurs nulles dans chaque colonne :

```
[85] df.isnull().sum()

RowId                   0
Date                    0
SecuritiesCode          0
Open                 8199
High                 8199
Low                  8199
Close                8199
Volume                  0
AdjustmentFactor        0
ExpectedDividend  2507703
SupervisionFlag         0
Target                242
dtype: int64
```

Figure 11 Valeurs nulles.

On constate qu'il y a des valeurs nulles au niveau de Open, High, Low, Close, Target qu'on va les remédier en les remplaçant avec la moyenne de chaque feature d'entre eux. Par contre, la propriété ExpectedDividend, qui signifie le taux de profit, on va remplacer ses valeurs nulles par des zéros.

On va remédier le problème des valeurs nulles toute de suite, mais avant on va afficher les types de nos colonnes :

```
[86] df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 2528524 entries, 0 to 195992
Data columns (total 12 columns):
 #   Column            Dtype
---  ------            -----
 0   RowId             object
 1   Date              object
 2   SecuritiesCode    int64
 3   Open              float64
 4   High              float64
 5   Low               float64
 6   Close             float64
 7   Volume            int64
 8   AdjustmentFactor  float64
 9   ExpectedDividend  float64
 10  SupervisionFlag   bool
 11  Target            float64
dtypes: bool(1), float64(7), int64(2), object(2)
memory usage: 233.9+ MB
```

Figure 12 Les types de nos colonnes.

On va supprimer la colonne RowId qui ne sera pas utile dans notre traitement :

```
[87] df = df.drop(['RowId'], axis = 1)
```

Figure 13 Suppression de RowId.

Comme on a déjà cité lors de l'explication des données, SupervisionFlag représente un drapeau des titres sous surveillance et des titres à radier. On va se débarrasser des données qui ont un SupervisionFlag égal à True (960 enregistrements), et on va supprimer toute la colonne.

```
[88] df = df[df['SupervisionFlag'] == False]

[89] df = df.drop(['SupervisionFlag'], axis = 1)
```

Figure 14 Traitement sur la colonne SupervisionFlag.

C'est le tour des valeurs nulles :

```
[90] df['ExpectedDividend'] = df['ExpectedDividend'].replace(np.nan, 0)

[91] df['Open'] = df['Open'].replace(np.nan, df['Open'].mean())

[92] df['Close'] = df['Close'].replace(np.nan, df['Close'].mean())

[93] df['High'] = df['High'].replace(np.nan, df['High'].mean())

[94] df['Low'] = df['Low'].replace(np.nan, df['Low'].mean())

[95] df['Target'] = df['Target'].replace(np.nan, df['Target'].mean())
```

Figure 15 Traitement des valeurs nulles.

Les données s'affichent comment cela après cette partie :

```
df
```

	Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	Target
0	2017-01-04	1301	2734.0	2755.0	2730.0	2742.0	31400	1.0	0.0	0.000730
1	2017-01-04	1332	568.0	576.0	563.0	571.0	2798500	1.0	0.0	0.012324
2	2017-01-04	1333	3150.0	3210.0	3140.0	3210.0	270800	1.0	0.0	0.006154
3	2017-01-04	1376	1510.0	1550.0	1510.0	1550.0	11300	1.0	0.0	0.011053
4	2017-01-04	1377	3270.0	3350.0	3270.0	3330.0	150800	1.0	0.0	0.003026
...	...	...	...	...	...	...	...	...	...	...
195988	2022-04-28	9990	497.0	504.0	494.0	504.0	51900	1.0	0.0	0.024048
195989	2022-04-28	9991	776.0	786.0	774.0	784.0	23400	1.0	0.0	0.011378
195990	2022-04-28	9993	1520.0	1530.0	1510.0	1517.0	8100	1.0	0.0	-0.005941
195991	2022-04-28	9994	2471.0	2489.0	2457.0	2485.0	7800	1.0	0.0	0.005602
195992	2022-04-28	9997	717.0	731.0	714.0	731.0	149800	1.0	0.0	-0.013680

2527551 rows × 10 columns

Figure 16 Les données après le remplacement des valeurs nulles.

On dessine notre boite à moustache pour afficher nos valeurs aberrantes (outliers) et un histogramme pour chaque colonne pour mieux visualiser la distribution des données :

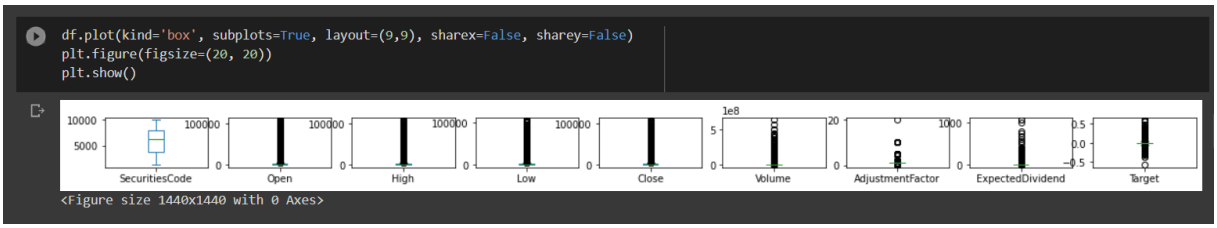


Figure 17 BoxPlot des données.

Figure 18 Histogramme des données.

Il est bien clair que les valeurs aberrantes sont énormes, mais c'est normal dans le cas d'un marché financier. Pour cela, on ne va rien faire à ce niveau.

On visualise maintenant les prix de nos actions selon la date :

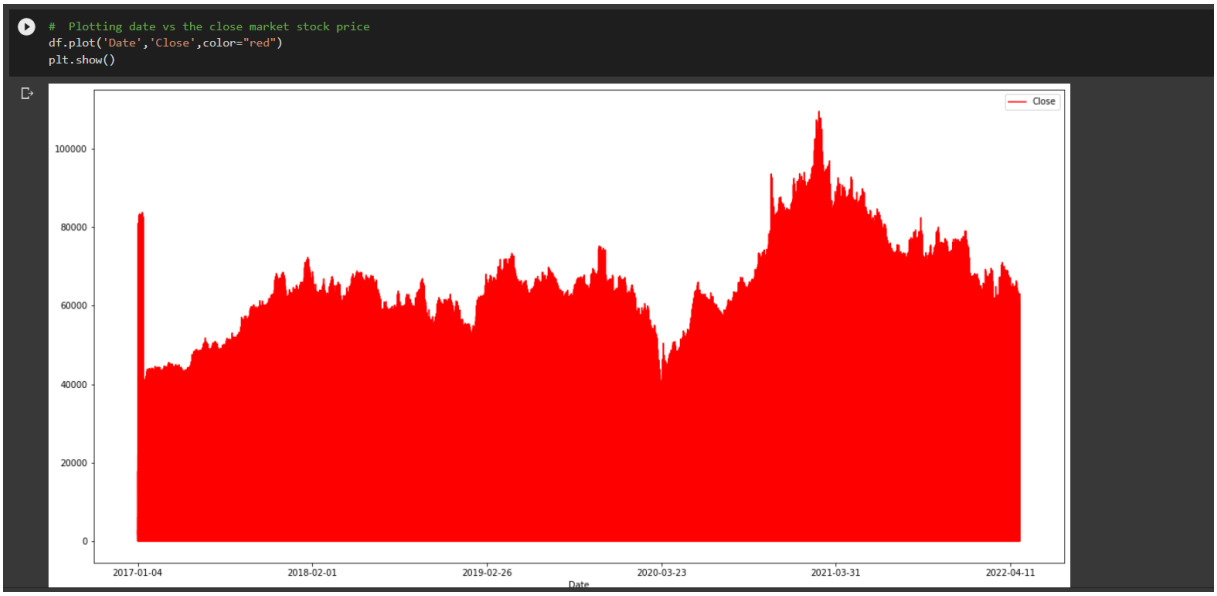


Figure 19 Les prix des actions selon la date.

Comme vous remarquez, pendant la période du confinement au Japan, les actions ont subi une grande diminution dans leurs prix (généralement il y a des secteurs qui ont connu des grandes pertes et d'autres qui ont connu un succès incontournable), dans notre cas on va supprimer les enregistrements de 01-03-2020 jusqu'à 01-09-2020 afin de ne pas affecter les résultats finaux.

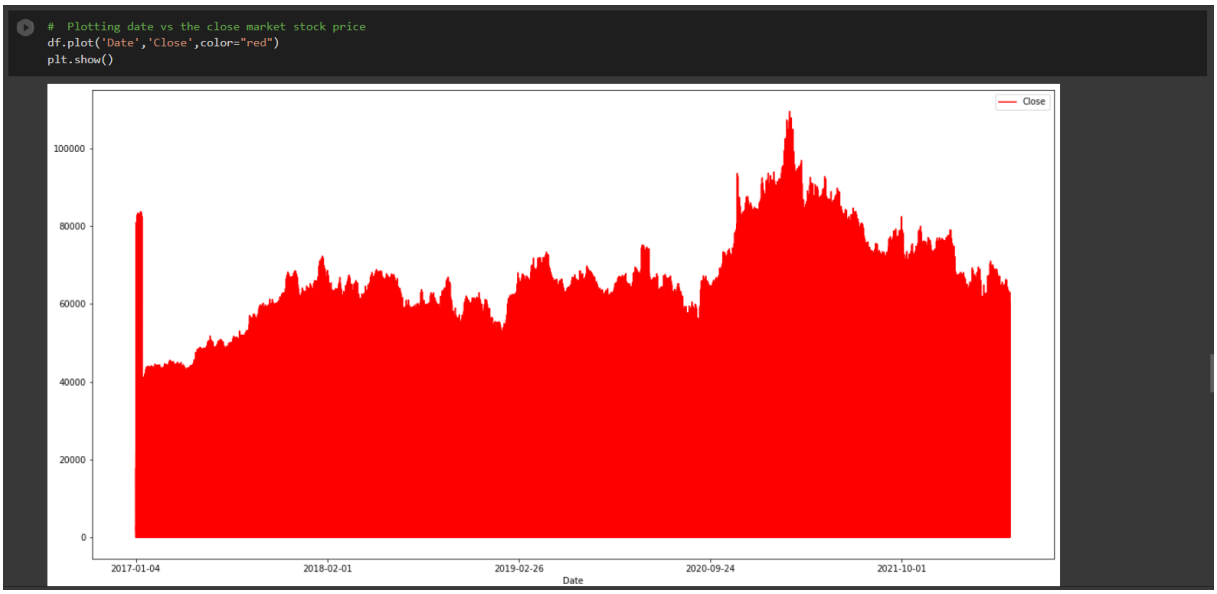


Figure 20 les prix des actions selon la date après le traitement.

Afin d'éviter les problèmes par la suite, on va procéder par convertir notre feature Date en datetime puis en int :

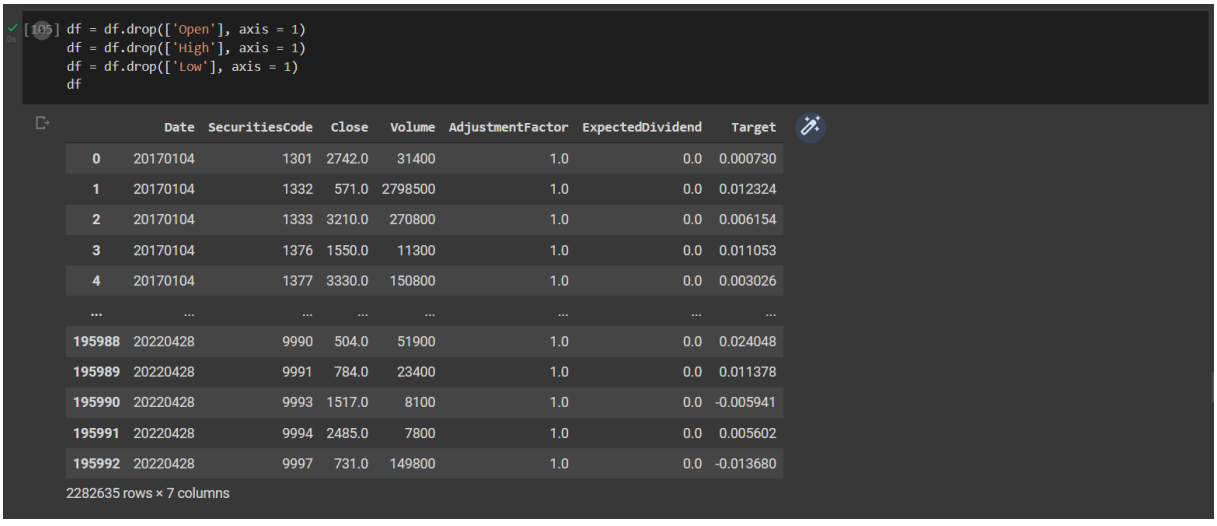
```
[102] df['Date'] = pd.to_datetime(df['Date'])
      df['Date'] = df['Date'].dt.strftime("%Y%m%d").astype(int)
```

Figure 21 Conversion du type de Date.

L'affichage du Heatmap, afin de visualiser les données corrélées et les traiter :

Figure 22 Heatmap.

On constate que les features Open, High, Low et Close sont corrélés entre eux, d'où la suppression des features qui nous donnent les mêmes informations, on va procéder par supprimer les features Open, High et Low et garder juste Close et voir ce que le modèle qu'on va adopter va produire par la suite :



```
[105] df = df.drop(['Open'], axis = 1)
      df = df.drop(['High'], axis = 1)
      df = df.drop(['Low'], axis = 1)
      df
```

	Date	SecuritiesCode	Close	Volume	AdjustmentFactor	ExpectedDividend	Target
0	20170104	1301	2742.0	31400	1.0	0.0	0.000730
1	20170104	1332	571.0	2798500	1.0	0.0	0.012324
2	20170104	1333	3210.0	270800	1.0	0.0	0.006154
3	20170104	1376	1550.0	11300	1.0	0.0	0.011053
4	20170104	1377	3330.0	150800	1.0	0.0	0.003026
...	...	...	...	...	...	...	...
195988	20220428	9990	504.0	51900	1.0	0.0	0.024048
195989	20220428	9991	784.0	23400	1.0	0.0	0.011378
195990	20220428	9993	1517.0	8100	1.0	0.0	-0.005941
195991	20220428	9994	2485.0	7800	1.0	0.0	0.005602
195992	20220428	9997	731.0	149800	1.0	0.0	-0.013680

2282635 rows × 7 columns

Figure 23 Suppression des colonnes à cause de la corrélation.

La remarque est bien claire que les features Close et Volume ont des valeurs grandes tant que les valeurs de la feature Target par exemple est très petite, pour cela, on va normaliser nos données en utilisant la méthode MinMaxScaler() :

```
[ ] from sklearn.preprocessing import MinMaxScaler

[107] scaler = MinMaxScaler()
      print(scaler.fit(df))
      MinMaxScaler(copy=True, feature_range=(0, 1))
      print(scaler.data_max_)
      print(scaler.transform(df))

      MinMaxScaler()
      [2.02204280e+07 9.99700000e+03 1.09550000e+05 6.43654000e+08
       2.00000000e+01 1.08000000e+03 6.18238022e-01]
      [[0.         0.         0.02490505 ... 0.04522613 0.         0.48402557]
       [0.         0.00356486 0.00508509 ... 0.04522613 0.         0.49371281]
       [0.         0.00367985 0.02917762 ... 0.04522613 0.         0.48855722]
       ...
       [1.         0.99954002 0.01372152 ... 0.04522613 0.         0.4784514 ]
       [1.         0.99965501 0.02255879 ... 0.04522613 0.         0.48809632]
       [1.         1.         0.00654579 ... 0.04522613 0.         0.47198463]]
```

Figure 24 La normalisation des données.

2. Modèle d'entrainement :

On passe maintenant pour diviser notre data en train et test afin d'appliquer notre modèle.

```
[108] from sklearn.model_selection import train_test_split

[109] features = ['SecuritiesCode', 'Close', 'Volume', 'AdjustmentFactor']

[110] X = df[features]
      Y = df['Target']

[111] X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=42)
```

Figure 25 La division des données en entrainement et test.

Passant maintenant à l'entrainement de notre modèle en utilisant la régression linéaire :

```
[112] lm = LinearRegression()
      lm.fit(X_train, y_train)

      LinearRegression()
```

Figure 26 l'entrainement du modèle.

Prédictions les valeurs de notre target :

```
[113] y_pred = lm.predict(X_test)
      df_ = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

[114] df_1 = df_.head(25)
      print(df_1)

             Actual  Predicted
767188   0.003856   0.000322
860393   0.017790   0.000346
2185396 -0.016626   0.000308
1379402  0.000000   0.000272
691183  -0.033099   0.000165
417760   0.032662   0.000242
309873  -0.004467   0.000310
850015   0.004145   0.000237
741869  -0.037037   0.000363
142676   0.026128   0.000241
126475  -0.004049   0.000266
428470   0.013744   0.000294
419678   0.000000   0.000256
43796   -0.010471   0.000152
1286034 -0.005038   0.000176
1222098 -0.025758   0.000389
134586   0.005299   0.000316
281650   0.026362   0.000295
274545   0.011673   0.000271
79769    0.006728   0.000268
2146802  0.008785   0.000291
1775726  0.020378   0.000258
146996  -0.021053   0.000303
554238  -0.004573   0.000214
2062885 -0.021579   0.000336
```

Figure 27 Prédiction de la target.

Calculant le score de notre modèle :

```
[115] lm.score(X_test,y_test)

      2.7521391326001954e-05
```

Figure 28 Score du modèle.

Utilisant les mesures de performances pour évaluer notre modèle (MAE, MSE, RMSE) :

```
[117] print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
      print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
      print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

      Mean Absolute Error: 0.014789498980545913
      Mean Squared Error: 0.0004904753013745926
      Root Mean Squared Error: 0.022146676982667007
```

Figure 29 Les mesures de performances.

VI. JPX Tokyo Stock Exchange Prediction en utilisant LSTM :

1. C’est quoi LSTM ?

Les réseaux de mémoire longue à court terme (LSTM) sont un type de réseau neuronal récurrent capable d'apprendre la dépendance d'ordre dans les problèmes de prédiction de séquence.

Il s'agit d'un comportement requis dans des domaines problématiques complexes tels que la traduction automatique, la reconnaissance vocale, etc.

Les LSTM sont un domaine complexe d'apprentissage en profondeur. Il peut être difficile de comprendre ce que sont les LSTM et comment des termes tels que bidirectionnel et séquence à séquence se rapportent au domaine.

Le réseau de neurones récurrent se base sur 3 exigences :

- Que le système soit capable de stocker des informations pour une durée arbitraire.
- Que le système soit résistant au bruit (c'est-à-dire aux fluctuations des entrées qui sont aléatoires ou non pertinentes pour prédire une sortie correcte).
- Que les paramètres du système soient entraînables (dans un délai raisonnable).

Les deux problèmes techniques surmontés par les LSTM sont les gradients de fuite et les gradients d'explosion, tous deux liés à la façon dont le réseau est entraîné.

Réseaux LSTM :

Les réseaux à mémoire longue et courte durée – généralement simplement appelés « LSTM » – sont un type particulier de RNN, capable d'apprendre des dépendances à long terme. Ils ont été introduits par Hochreiter & Schmidhuber (1997), et ont été raffinés et popularisés par de nombreuses personnes dans les travaux suivants. Ils fonctionnent extrêmement bien sur une grande variété de problèmes et sont maintenant largement utilisés.

Les LSTM sont explicitement conçus pour éviter le problème de dépendance à long terme. Se souvenir d'informations pendant de longues périodes est pratiquement leur comportement par défaut, pas quelque chose qu'ils ont du mal à apprendre !

Tous les réseaux de neurones récurrents ont la forme d'une chaîne de modules répétitifs de réseau de neurones. Dans les RNN standard, ce module répétitif aura une structure très simple, telle qu'une seule couche tanh.

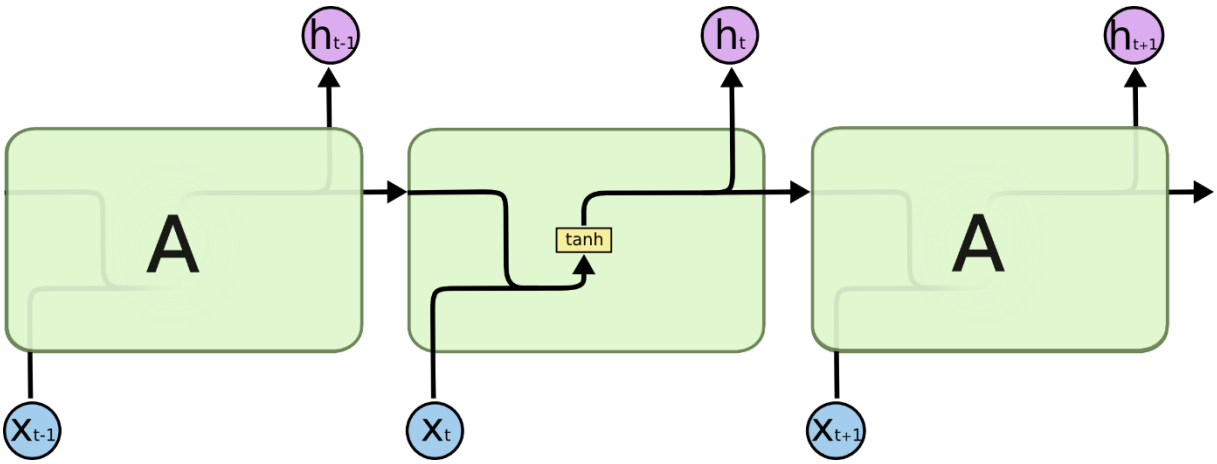


Figure 30 Le module répétitif dans un RNN standard contient une seule couche.

Les LSTM ont également cette structure en forme de chaîne, mais le module répétitif a une structure différente. Au lieu d'avoir une seule couche de réseau de neurones, il y en a quatre, qui interagissent d'une manière très spéciale.

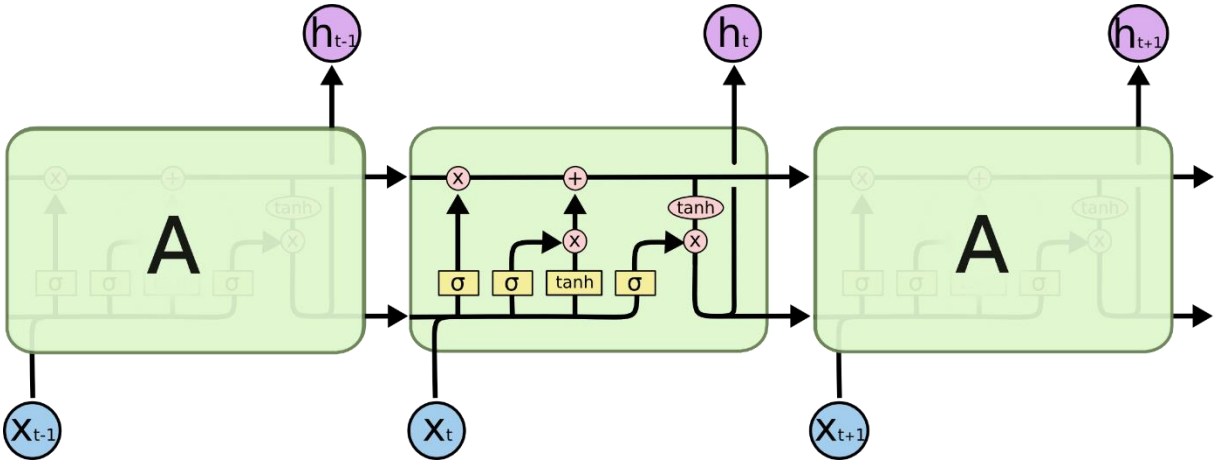


Figure 31 Le module répétitif dans un LSTM contient quatre couches en interaction.

Avant d'entrer dans le vif du sujet, essayant d'abord de se familiariser avec les notions du LSTM :

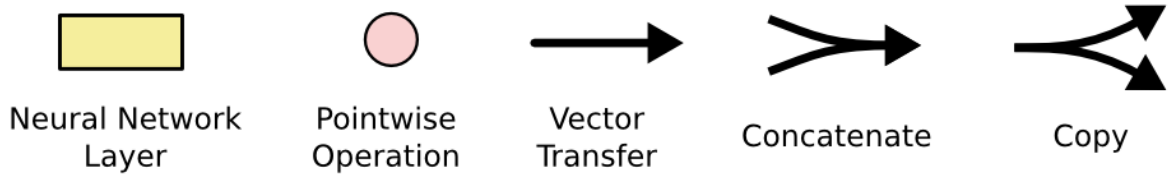


Figure 32 Des notions essentielles.

Dans le diagramme ci-dessus, chaque ligne porte un vecteur entier, de la sortie d'un nœud aux entrées des autres. Les cercles roses représentent des opérations ponctuelles, comme l'addition de vecteurs, tandis que les cases jaunes sont des couches de réseau neuronal apprises. La fusion de lignes indique une concaténation, tandis qu'une bifurcation de ligne indique que son contenu est copié et que les copies vont à des emplacements différents.

L'idée centrale derrière les LSTM :

La clé des LSTM est l'état de la cellule, la ligne horizontale qui traverse le haut du diagramme.

L'état cellulaire est un peu comme un tapis roulant. Il parcourt toute la chaîne, avec seulement quelques interactions linéaires mineures. Il est très facile pour les informations de circuler sans changement.

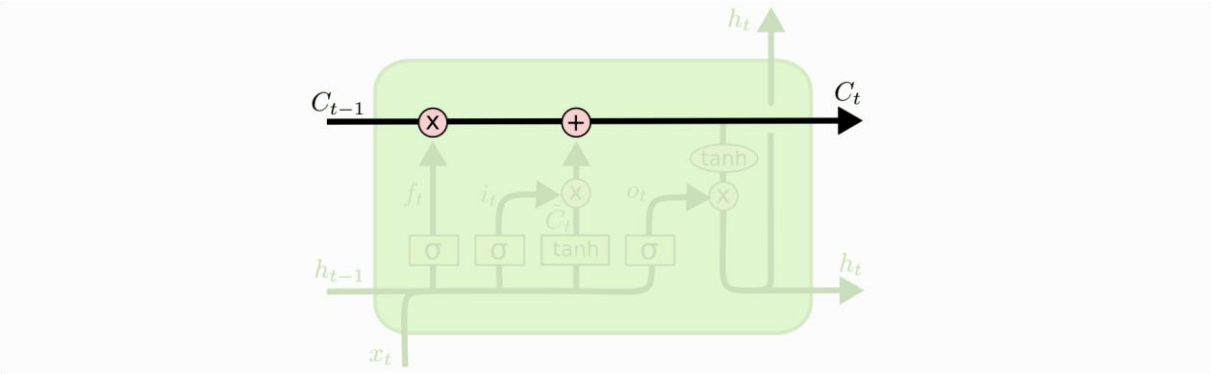


Figure 33 La circulation des informations.

Le LSTM a la capacité de supprimer ou d'ajouter des informations à l'état de la cellule, soigneusement régulées par des structures appelées portes.

Les portes sont un moyen de laisser éventuellement passer des informations. Ils sont composés d'une couche de réseau neuronal sigmoïde et d'une opération de multiplication ponctuelle.

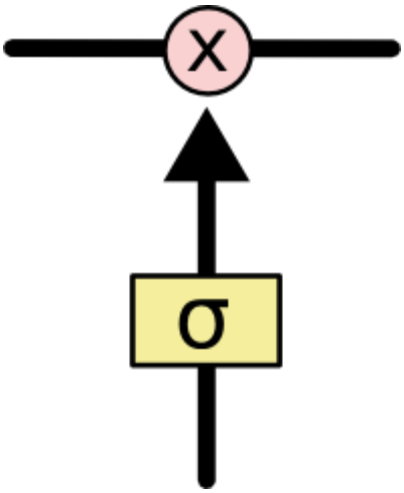


Figure 34 Les portes.

La couche sigmoïde produit des nombres entre zéro et un, décrivant la quantité de chaque composant qui doit être laissée passer. Une valeur de zéro signifie « ne rien laisser passer », tandis qu'une valeur de un signifie « tout laisser passer ! ».

Un LSTM a trois de ces portes, pour protéger et contrôler l'état de la cellule.

2. Le traitement initial des données :

On commence d'abord par les bibliothèques qu'on doit importer :

```
In [12]: import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sns
         import numpy as np
         import warnings
         warnings.filterwarnings('ignore')
```

Figure 35 Importation des bibliothèques.

On passe par la suite pour charger le fichier csv principal qu'on va utiliser, dans cette partie, on va se restreindre sur le dossier train_files et on ne va pas s'occuper du dossier supplemental_files :

```
In [13]: # Load data
         df_stock_prices=pd.read_csv('jpx-tokyo-stock-exchange-prediction/train_files/stock_prices.csv')

In [14]: df_stock_prices.shape
Out[14]: (2332531, 12)

In [15]: df_stock_prices.head()
```

Out[15]:

	RowId	Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	SupervisionFlag	Target
0	20170104_1301	2017-01-04	1301	2734.0	2755.0	2730.0	2742.0	31400	1.0	NaN	False	0.000730
1	20170104_1332	2017-01-04	1332	568.0	576.0	563.0	571.0	2798500	1.0	NaN	False	0.012324
2	20170104_1333	2017-01-04	1333	3150.0	3210.0	3140.0	3210.0	270800	1.0	NaN	False	0.006154
3	20170104_1376	2017-01-04	1376	1510.0	1550.0	1510.0	1550.0	11300	1.0	NaN	False	0.011053
4	20170104_1377	2017-01-04	1377	3270.0	3350.0	3270.0	3330.0	150800	1.0	NaN	False	0.003026

Figure 36 Chargement des données.

Là on supprime le RowId dont on n'aura pas besoin par la suite et on affiche les types des données.

```
In [16]: # We do not need RowId
         df_stock_prices.drop('RowId', axis=1, inplace=True)

In [17]: df_stock_prices.head()
```

Out[17]:

	Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	SupervisionFlag	Target
0	2017-01-04	1301	2734.0	2755.0	2730.0	2742.0	31400	1.0	NaN	False	0.000730
1	2017-01-04	1332	568.0	576.0	563.0	571.0	2798500	1.0	NaN	False	0.012324
2	2017-01-04	1333	3150.0	3210.0	3140.0	3210.0	270800	1.0	NaN	False	0.006154
3	2017-01-04	1376	1510.0	1550.0	1510.0	1550.0	11300	1.0	NaN	False	0.011053
4	2017-01-04	1377	3270.0	3350.0	3270.0	3330.0	150800	1.0	NaN	False	0.003026

```
In [7]: df_stock_prices.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2332531 entries, 2017-01-04 to 2021-12-03
Data columns (total 10 columns):
 #   Column            Dtype
---  ------            -----
 0   SecuritiesCode    int64
 1   Open              float64
 2   High              float64
 3   Low               float64
 4   Close             float64
 5   Volume            int64
 6   AdjustmentFactor  float64
 7   ExpectedDividend  float64
 8   SupervisionFlag   bool
 9   Target            float64
dtypes: bool(1), float64(7), int64(2)
memory usage: 180.2 MB
```

Figure 37 Suppression du RowId et affichage des types des données.

Passant maintenant pour afficher le résumé statistique des données :

```
In [8]: df_stock_prices.describe()
```

Out[8]:

	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	Target
count	2.332531e+06	2.324923e+06	2.324923e+06	2.324923e+06	2.324923e+06	2.332531e+06	2.332531e+06	18865.000000	2.332293e+06
mean	5.894835e+03	2.594511e+03	2.626540e+03	2.561227e+03	2.594023e+03	6.919366e+05	1.000508e+00	22.017730	4.450964e-04
std	2.404161e+03	3.577192e+03	3.619363e+03	3.533494e+03	3.576538e+03	3.911256e+06	6.773040e-02	29.882453	2.339879e-02
min	1.301000e+03	1.400000e+01	1.500000e+01	1.300000e+01	1.400000e+01	0.000000e+00	1.000000e-01	0.000000	-5.785414e-01
25%	3.891000e+03	1.022000e+03	1.035000e+03	1.009000e+03	1.022000e+03	3.030000e+04	1.000000e+00	5.000000	-1.049869e-02
50%	6.238000e+03	1.812000e+03	1.834000e+03	1.790000e+03	1.811000e+03	1.071000e+05	1.000000e+00	15.000000	0.000000e+00
75%	7.965000e+03	3.030000e+03	3.070000e+03	2.995000e+03	3.030000e+03	4.021000e+05	1.000000e+00	30.000000	1.053159e-02
max	9.997000e+03	1.099500e+05	1.105000e+05	1.072000e+05	1.095500e+05	6.436540e+08	2.000000e+01	1070.000000	1.119512e+00

Figure 38 Le résumé statistique des données.

Le tour maintenant est des valeurs dupliquées dont on veut se débarrasser :

Duplicated data

```
In [6]: df_stock_prices.duplicated().value_counts()
```

```
Out[6]: False    2332531
        dtype: int64
```

```
In [7]: duplicateRows = df_stock_prices[df_stock_prices.duplicated()]
        duplicateRows.head()
```

Out[7]:

Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	ExpectedDividend	SupervisionFlag	Target

```
In [8]: df_stock_prices = df_stock_prices.drop_duplicates()
        df_stock_prices.duplicated().value_counts()
```

```
Out[8]: False    2332531
        dtype: int64
```

Figure 39 Les valeurs dupliquées.

On affiche les valeurs nulles :

Missing values

```
In [114]: df_stock_prices.isnull().sum()  # df_stock_prices.isnull().sum()*100/df_stock_prices.shape[0]  Pourcentage
```

```
Out[114]: SecuritiesCode            0
          Open                 5308
          High                 5308
          Low                  5308
          Close                5308
          Volume                  0
          AdjustmentFactor        0
          ExpectedDividend    2311022
          SupervisionFlag         0
          Target                  7
          dtype: int64
```

Figure 40 les valeurs nulles.

On supprime la colonne ExpectedDividend toute entière et on affiche les valeurs nulles de la colonne Target :

```
In [37]: df_stock_prices.drop(['ExpectedDividend'], axis = 1, inplace = True, errors = 'ignore')

In [94]: df_stock_prices.isnull().sum()

Out[94]: SecuritiesCode        0
         Open                5308
         High                5308
         Low                 5308
         Close               5308
         Volume                 0
         AdjustmentFactor       0
         SupervisionFlag        0
         Target                 7
         dtype: int64

In [72]: m = df_stock_prices[df_stock_prices['Target'].isnull()]
         m
```

Out[72]:

Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	SupervisionFlag	Target
2017-01-04	3540	NaN	NaN	NaN	NaN	0	1.0	False	NaN
2017-06-26	3540	NaN	NaN	NaN	NaN	0	1.0	False	NaN
2017-11-02	3540	NaN	NaN	NaN	NaN	0	0.2	False	NaN
2017-11-13	3540	NaN	NaN	NaN	NaN	0	1.0	True	NaN
2018-04-20	4382	NaN	NaN	NaN	NaN	0	1.0	False	NaN
2020-08-20	4056	NaN	NaN	NaN	NaN	0	1.0	False	NaN
2020-10-02	2987	NaN	NaN	NaN	NaN	0	1.0	False	NaN

Figure 41 Suppression de la colonne ExpectedDividend.

On constate que quand la Target est nulle, les colonnes Open, High, Low et Close sont aussi nulles. On procède par supprimer tous les enregistrements, dont il y a des valeurs nulles au niveau de ces colonnes :

```
In [38]: # Drop missing values
         df_stock_prices = df_stock_prices.dropna(axis=0)
         df_stock_prices.tail()
```

Out[38]:

Date	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	SupervisionFlag	Target
2021-12-03	9990	514.0	528.0	513.0	528.0	44200	1.0	False	0.034816
2021-12-03	9991	782.0	794.0	782.0	794.0	35900	1.0	False	0.025478
2021-12-03	9993	1690.0	1690.0	1645.0	1645.0	7200	1.0	False	-0.004302
2021-12-03	9994	2388.0	2396.0	2380.0	2389.0	6500	1.0	False	0.009098
2021-12-03	9997	690.0	711.0	686.0	696.0	381100	1.0	False	0.018414

```
In [121]: df_stock_prices.isnull().sum()

Out[121]: SecuritiesCode      0
          Open                0
          High                0
          Low                 0
          Close               0
          Volume              0
          AdjustmentFactor    0
          SupervisionFlag     0
          Target              0
          dtype: int64
```

Figure 42 Suppression des missings values.

Passant pour re-afficher le résumé statistique des données :

```
In [119]: pd.set_option('display.float_format', lambda x: '%.5f' % x)
          df_stock_prices.describe()
```

Out[119]:

	SecuritiesCode	Open	High	Low	Close	Volume	AdjustmentFactor	Target
count	2324576.00000	2324576.00000	2324576.00000	2324576.00000	2324576.00000	2324576.00000	2324576.00000	2324576.00000
mean	5895.53465	2594.00490	2626.03855	2560.71659	2593.51787	694304.33589	1.00051	0.00043
std	2403.15965	3575.84401	3618.04174	3532.12118	3575.19307	3917732.84198	0.06779	0.02339
min	1301.00000	14.00000	15.00000	13.00000	14.00000	100.00000	0.10000	-0.57854
25%	3891.00000	1022.00000	1035.00000	1009.00000	1022.00000	30700.00000	1.00000	-0.01053
50%	6240.00000	1811.00000	1834.00000	1790.00000	1811.00000	108000.00000	1.00000	0.00000
75%	7965.00000	3030.00000	3065.00000	2995.00000	3030.00000	404200.00000	1.00000	0.01053
max	9997.00000	109950.00000	110500.00000	107200.00000	109550.00000	643654000.00000	20.00000	0.61824

Figure 43 Le résumé statistique des données après les traitements.

Commençant d'abord par la heatmap qui montre les features qui sont corrélées :

```
In [102]: import matplotlib.pyplot as plt
          import seaborn as sns
          corr=df_stock_prices.corr()
          top_features=corr.index
          plt.figure(figsize=(10,10))
          sns.heatmap(df_stock_prices[top_features].corr(),annot=True)
```

Out[102]: <AxesSubplot:>

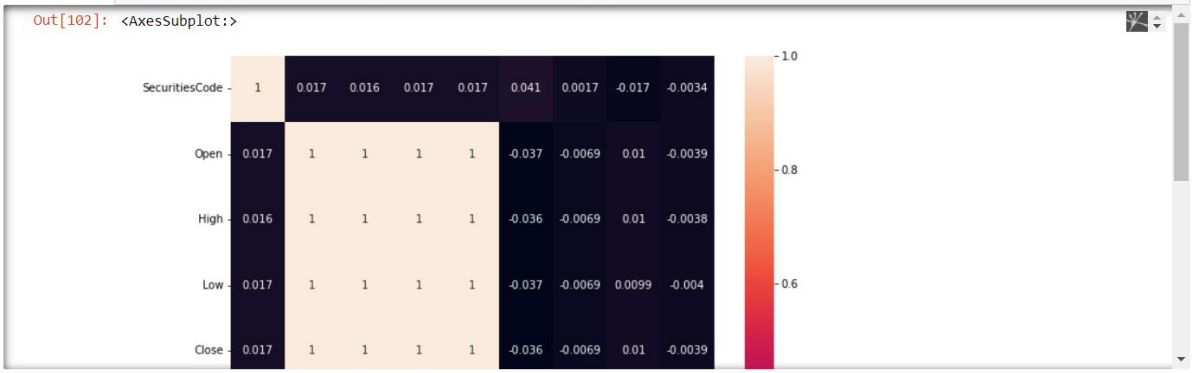


Figure 44 Heatmap des données.

Affichant les données les prix d'ouverture selon la date :

```
In [120]: df_stock_prices['Open'].plot(figsize=(16,5))
```

Out[120]: <AxesSubplot:xlabel='Date'>

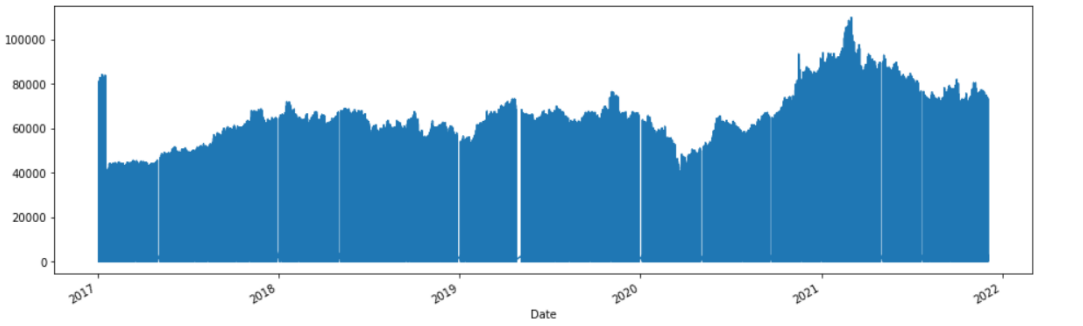


Figure 45 Les prix d'ouverture selon la date.

Affichant les données les prix d'ouverture et de clôture selon la date :

```
In [123]: df_stock_prices.plot.line(y=['Open','Close'], figsize=(10,7))
Out[123]: <AxesSubplot:xlabel='Date'>
```

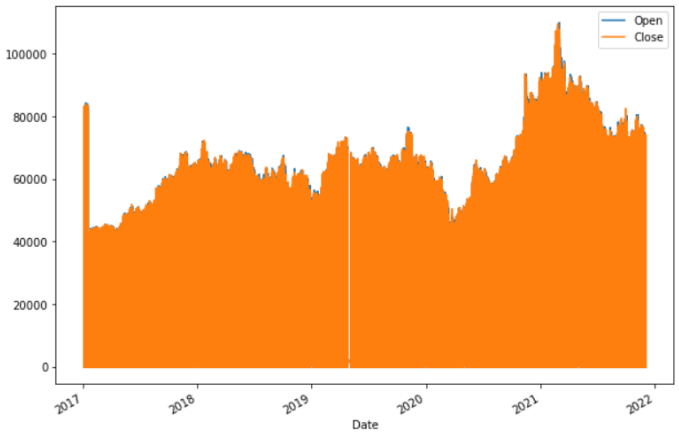


Figure 46 Les prix d'ouverture et de clôture selon la date.

On veut afficher le nombre des outliers dans nos données :

```
In [158]: def get_num_outliers (column):
              q1 = np.percentile(column, 25)
              q3 = np.percentile(column, 75)
              return sum((column<q1) | (column>q3))

In [162]: df_stock_prices.groupby('Open').agg([get_num_outliers]).sum()*100/df_stock_prices.shape[0]
Out[162]: SecuritiesCode    get_num_outliers   49.40991
          High              get_num_outliers   46.64408
          Low               get_num_outliers   46.61211
          Close             get_num_outliers   48.08361
          Volume            get_num_outliers   50.15947
          AdjustmentFactor  get_num_outliers    0.03136
          Target            get_num_outliers   50.14334
          dtype: float64

In [164]: df_stock_prices.groupby('Open').agg([get_num_outliers]).sum()
Out[164]: SecuritiesCode    get_num_outliers    1148571
          High              get_num_outliers    1084277
          Low               get_num_outliers    1083534
          Close             get_num_outliers    1117740
          Volume            get_num_outliers    1165995
          AdjustmentFactor  get_num_outliers        729
          Target            get_num_outliers    1165620
          dtype: int64
```

Figure 47 Affichage du nombre des outliers.

Le nombre est assez énorme. Pour rendre les choses plus claires, on affiche notre boite à moustache :

```
In [153]: plt.figure(facecolor='yellow')
          df_stock_prices.plot(kind= 'box' , subplots=True, layout=(3,3), sharex=False, sharey=False, figsize=(15,15))
          sns.set(font_scale = 1.5)
```

```
<Figure size 432x288 with 0 Axes>
```

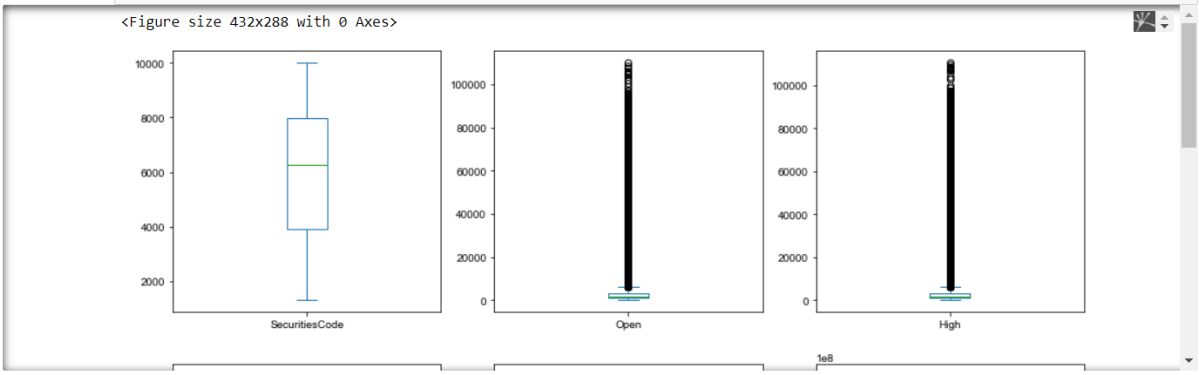


Figure 48 BoxPlot des données.

3. Modèle d'entrainement :

Comme on a déjà montré, le modèle qu'on va utiliser par la suite pour entrainer notre modèle est LSTM :

```
In [40]: import torch
         import torch.nn as nn
         import torch.nn.functional as F

         class LSTM(nn.Module):
             def __init__(self, input_size=8, sequence_num=31, lstm_dim=128,
                          num_layers=2, output_size=1):
                 super().__init__()

                 self.lstm = nn.LSTM(input_size, lstm_dim, num_layers, batch_first=True, bidirectional=True)
                 self.linear1 = nn.Linear(lstm_dim*sequence_num*2, 1)
                 self.bn1 = nn.BatchNorm1d(lstm_dim*sequence_num*2)

             def forward(self, x):
                 lstm_out, _ = self.lstm(x)
                 x = lstm_out.reshape(lstm_out.shape[0], -1)
                 x = self.linear1(self.bn1(x))
                 return x
```

Figure 49 Le modèle LSTM.

Notre modèle effectue son entrainement sur 10 itérations :

In [50]:
```
from tqdm import tqdm
epochs = 10
batch_size = 512
# Check wheter GPU is available
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
# Model Instantiation
model = LSTM(input_size=5, sequence_num=31, lstm_dim=128, num_layers=2, output_size=1)
model.to(device)
model.train()
# setting optimizer
lr = 0.0001
weight_decay = 1.0e-05
optimizer = torch.optim.Adagrad(model.parameters(), lr=lr, weight_decay=weight_decay)
# setting criterion
criterion = nn.MSELoss()
# set iteration counter
iteration = 0
#
log_train = [[0], [np.inf]]
for epoch in range(epochs):
    epoch_loss = 0.0
    for sc in tqdm(df_stock_prices['SecuritiesCode'].unique()):
        X, y = dataset_dict[str(sc)][:, :-1], dataset_dict[str(sc)][:, -1]
        dataset = MyDataset(X, y=y, sequence_num=31, mode='train')
        dataloader = create_dataloader(dataset, X.shape[0], sequence_num=31, input_size=8, batch_size=batch_size, shuffle=True)
        for data, targets in dataloader:
            data, targets = data.to(device), targets.to(device)

            optimizer.zero_grad()

            data = data.to(torch.float32)
            output = model.forward(data)
            targets = targets.to(torch.float32)

            loss = criterion(output.view(1,-1)[0], targets)

            loss.backward()

            optimizer.step()

            epoch_loss += loss.item()

            iteration += 1
    epoch_loss /= iteration
    print('epoch_loss={}'.format(epoch_loss))
    log_train[0].append(iteration)
    log_train[1].append(epoch_loss)
```

```
100%|██████████| 2000/2000 [1:31:56<00:00,  2.76s/it]
epoch_loss=0.002539817585388118
100%|██████████| 2000/2000 [1:50:59<00:00,  3.33s/it]
epoch_loss=0.0006849390439615749
100%|██████████| 2000/2000 [1:49:31<00:00,  3.29s/it]
```

Figure 50 L'entrainement du modèle LSTM.

Meme si on a tenté d'entrainer le modèle plusieurs fois, ce dernier s'arrête à une itération donnée à cause des limites du matériel :

```
100%|██████████| 2000/2000 [1:31:56<00:00,  2.76s/it]
epoch_loss=0.002539817585388118
100%|██████████| 2000/2000 [1:50:59<00:00,  3.33s/it]
epoch_loss=0.0006849390439615749
100%|██████████| 2000/2000 [1:49:31<00:00,  3.29s/it]
epoch_loss=0.0004000207986177636
100%|██████████| 2000/2000 [1:28:49<00:00,  2.66s/it]
epoch_loss=0.00027323903511376513
100%|██████████| 2000/2000 [1:27:33<00:00,  2.63s/it]
epoch_loss=0.00020615823317126597
100%|██████████| 2000/2000 [1:27:32<00:00,  2.63s/it]
epoch_loss=0.0001642580378414675
100%|██████████| 2000/2000 [1:27:14<00:00,  2.62s/it]
epoch_loss=0.00013669496170126005
100%|██████████| 2000/2000 [1:28:01<00:00,  2.64s/it]
epoch_loss=0.00011626183456155245
100%|██████████| 2000/2000 [1:28:03<00:00,  2.64s/it]
epoch_loss=9.933321823787577e-05
100%|██████████| 2000/2000 [2:21:18<00:00,  4.24s/it]
epoch_loss=8.760346144642368e-05
```

Figure 51 Les itérations de l'entrainement.

La fonction de perte diminue lorsque le nombre d'itération augmente.

```
In [52]: import matplotlib.pyplot as plt
         plt.plot(log_train[0][1:], log_train[1][1:])
         plt.xlabel('epoch')
         plt.ylabel('loss')
         plt.show()
```

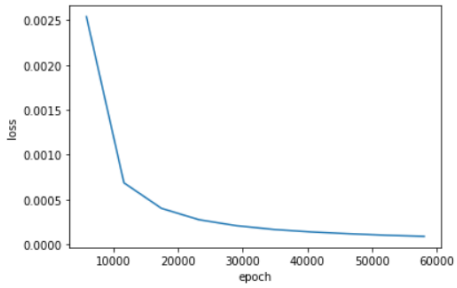


Figure 52 La fonction d'erreur.

Voici la fonction responsable de la prédiction de notre Target :

In [ ]:
```python
from datetime import datetime
def predict(model, X_df, sequence=31):
    pred_df = X_df[['Date', 'SecuritiesCode']]
    # groupby
    code_group = X_df.groupby('SecuritiesCode')
    X_all = np.array([])
    for sc, group in code_group:
        group_std = stdsc.transform(group[columns])
        X = dataset_dict[str(sc)][-1*(sequence-1):, :-1]
        X = np.vstack((X, group_std))
        X_all = np.append(X_all, X)
    X_all = X_all.reshape(-1, sequence, X.shape[1])
    y_pred = np.array([])
    for it in range(X_all.shape[0]//512+1):
        data = X_all[it*512:(it+1)*512]
        data = torch.from_numpy(data.astype(np.float32)).clone()
        data = data.to(torch.float32)
        data = data.to(device)
        output = model.forward(data)
        output = output.view(1, -1)
        output = output.to('cpu').detach().numpy().copy()
        y_pred = np.append(y_pred, output[0])
    pred_df['target'] = y_pred
    pred_df['Rank'] = pred_df["target"].rank(ascending=False,method="first") -1
    pred_df['Rank'] = pred_df['Rank'].astype(int)
    pred_df = pred_df.drop('target', axis=1)
    return pred_df
#df_stock_prices['Date'] = pd.to_datetime(df_stock_prices['Date'])
#stock_prices.query('SecuritiesCode == @code')['Date']

test_X_df = df_stock_prices[df_stock_prices.drop('Target', axis=1)['Date'] == datetime(2021, 12, 3)]
y_pred = predict(model, test_X_df)
print(y_pred.shape)
print(y_pred)
```

Figure 53 La prédiction de la Target.

VII. Conclusion :

Dans une première approche, l'utilisation de la régression linéaire pour prédire les cours des actions est une tâche simple en Python lorsque l'on exploite la puissance des bibliothèques d'apprentissage automatique telles que scikit-learn. La commodité de pandas, la bibliothèque ne peut pas non plus être surestimée, ce qui permet d'ajouter l'un des dizaines d'indicateurs techniques dans une seule ligne de code.

Ce projet nous a permis de voir comment charger des données, tester et diviser les données, ajouter des indicateurs, former un modèle linéaire et enfin appliquer ce modèle pour prédire les cours boursiers futurs de JPX Tokyo, avec un certain succès !

Dans une deux approche, nous avons utilisé le LSTM, les réseaux de mémoire longue à court terme (LSTM) qui sont un type de réseau neuronal récurrent capable d'apprendre la dépendance d'ordre dans les problèmes de prédiction de séquence. Cette deuxième approche était perturbée à la fin de ses itérations par la limitation de la RAM, car le modèle s’arrête à chaque fois dans ses dernières itérations, meme Google Colab avec ses caractéristiques n’était pas suffisant, car lui aussi s’arrête pour se reconnecter meme s’il nous offre 4 Go de RAM de plus.

Le problème qu’on a choisi pour résoudre dans ce projet accepte plusieurs solutions que ça soit du machine learning ou du deep Learning, mais à cause des limites du temps et du matériel, on était s’arrêté à ce point-là.