Développement d’un Algorithme de Trading sur le Forex (EUR/USD) avec Gestion Dynamique des Risques

Zouirchi Hamza & El Amrani Mohammed Aymane

Introduction Globale

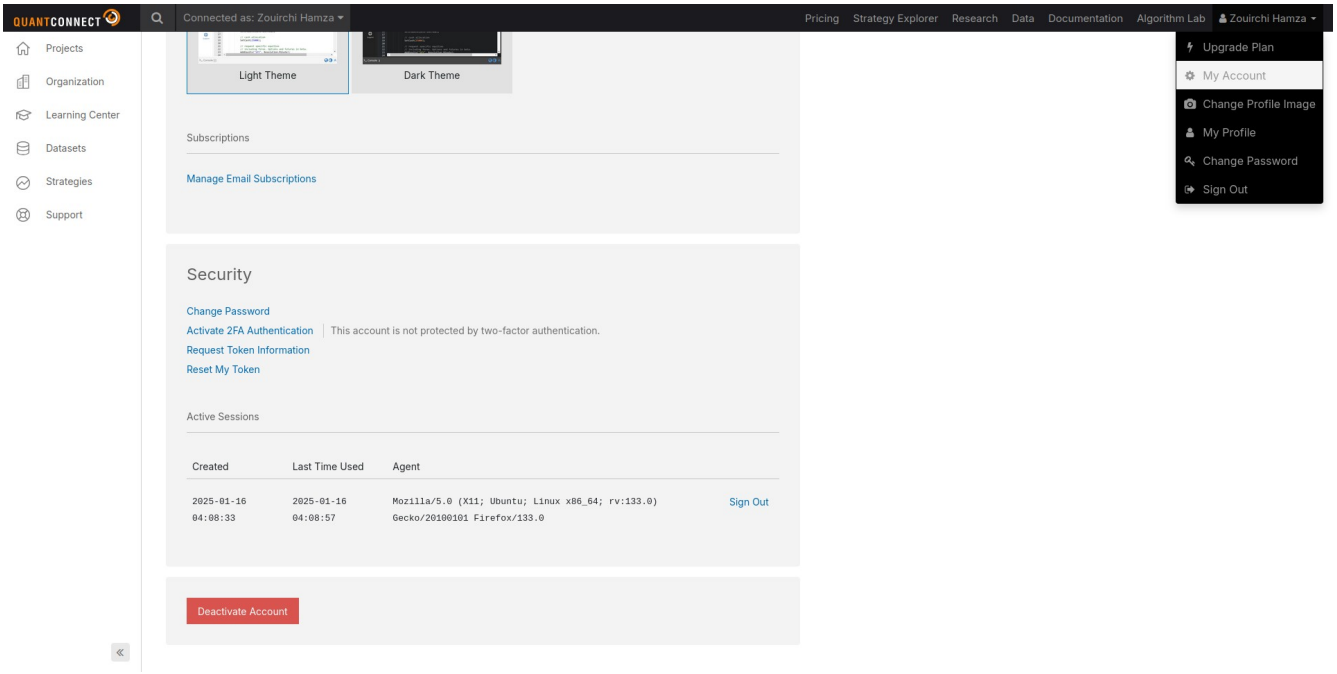
Le marché des changes, ou Forex, est l'un des secteurs les plus dynamiques et liquides du monde financier, avec un volume de transactions quotidien dépassant les 6 trillions de dollars. Parmi les nombreuses paires de devises échangées, l'EUR/USD se distingue par sa popularité, représentant un volume significatif d'échanges en raison de l'importance économique de la zone euro et des États-Unis. Dans ce contexte, le développement d'algorithmes de trading est devenu une pratique courante, permettant aux traders de capitaliser sur les fluctuations des prix avec une efficacité accrue.

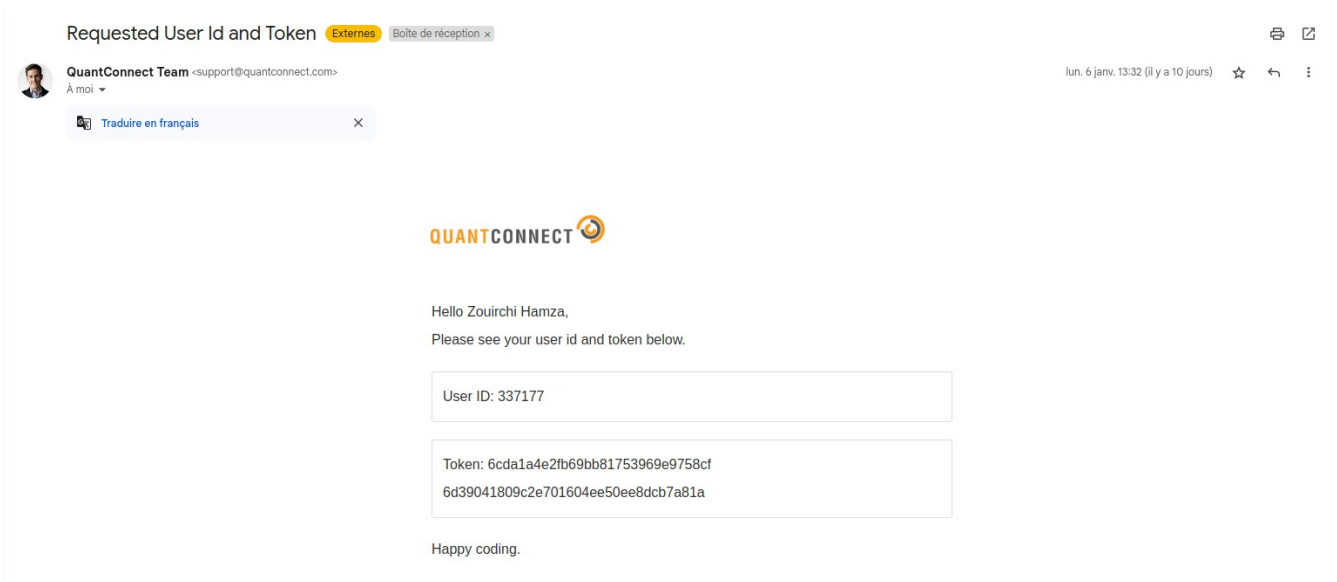
L'algorithme de trading repose sur des modèles mathématiques et des analyses techniques qui permettent d'identifier des opportunités d'achat ou de vente. Cependant, la volatilité inhérente au marché Forex pose des défis importants, rendant cruciale l'intégration d'une gestion dynamique des risques. Cette approche vise à protéger le capital investi tout en maximisant les rendements, en ajustant les positions en fonction des conditions du marché et des performances passées.

L'utilisation d'API comme celle de QuantConnect facilite grandement ce processus. Cette plateforme offre un accès à des données historiques et en temps réel, permettant aux développeurs de tester leurs stratégies algorithmiques dans un environnement contrôlé. Grâce à des outils avancés de backtesting, les traders peuvent simuler les performances de leurs algorithmes sur des périodes passées, évaluant ainsi leur viabilité avant de les déployer sur le marché réel. En combinant des analyses techniques robustes, une gestion des risques réfléchie et des capacités de test approfondies, les algorithmes de trading sur le Forex, et en particulier sur la paire EUR/USD, peuvent devenir des instruments puissants pour naviguer dans cet environnement complexe et en constante évolution.

Applications :

1. Génération d'un jeton API sécurisé





Dedans QuantConnect,
Creer un Compte, naviguer vers My Account, Chercher dans Security ‘Request Token Information’.
Dans votre email, vous allez recevoir les infos qu’on va utiliser dans le code pour se connecter a QuantConnect et Tester vos strategies ‘Algorithmiquement’.
Vous pouvez utiliser la plateforme, et aussi python directement comme suit :

```
import base64
import hashlib
import time

# Obtenir un timestamp
timestamp = str(int(time.time()))
time_stamped_token = "6cda1a4e2fb69bb81753969e9758cf6d39041809c2e701604ee50ee8dcb7a81a"
+ ':' + timestamp

# Obtenir le jeton API haché
hashed_token = hashlib.sha256(time_stamped_token.encode('utf-8')).hexdigest()

# Formatage correct de la chaîne d'authentification
authentication = "{}:{}".format("337177", hashed_token)

# Encoder le jeton d'authentification en base64
api_token = base64.b64encode(authentication.encode('utf-8')).decode('ascii')

# Afficher le résultat
print(api_token)
```

Dans cette première section, nous générons un jeton API sécurisé pour une authentification. Nous commençons par obtenir un timestamp actuel en secondes. Ensuite, nous créons une chaîne qui combine un identifiant fixe et ce timestamp, que nous hachons avec l'algorithme SHA-256 pour produire un jeton sécurisé. Nous formatons ensuite une chaîne d'authentification en utilisant cet identifiant et le jeton haché, puis nous encodons le tout en base64 pour obtenir le jeton final, qui est affiché à l'utilisateur.

```
# Output the result
print(api_token)
```

```
MzM3MTc3OmNlMzBmMGNkYjA1N2VmYTJmY2E2YmRlMjRjODUzMzBjNWM5ZGM3NTljYmM5MzZiOWJkMmNkOGIyNjc4OTYwMzk=
```

2. Téléchargement des données historiques

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import yfinance as yf


# symbole de EUR/USD :
symbol = "EURUSD=X"

# Télécharger les données historiques pour 1 jour avec un intervalle de 5 minutes
stock = yf.Ticker(symbol)
df = stock.history(period="1d", interval="1m")  # Période de 1 jour, intervalle de 1 minute

# Afficher les données
df
```

Datetime	Open	High	Low	Close	Volume	Dividends	Stock Splits
2025-01-16 00:00:00+00:00	1.029866	1.029866	1.029866	1.029866	0	0.0	0.0
2025-01-16 00:01:00+00:00	1.029866	1.029866	1.029866	1.029866	0	0.0	0.0
2025-01-16 00:02:00+00:00	1.029760	1.029760	1.029760	1.029760	0	0.0	0.0
2025-01-16 00:03:00+00:00	1.029760	1.029760	1.029760	1.029760	0	0.0	0.0
2025-01-16 00:04:00+00:00	1.029760	1.029760	1.029760	1.029760	0	0.0	0.0
...	...	...	...	...	...	...	...
2025-01-16 03:14:00+00:00	1.029018	1.029018	1.029018	1.029018	0	0.0	0.0
2025-01-16 03:15:00+00:00	1.028912	1.028912	1.028912	1.028912	0	0.0	0.0
2025-01-16 03:16:00+00:00	1.028912	1.028912	1.028912	1.028912	0	0.0	0.0
2025-01-16 03:17:00+00:00	1.028912	1.028912	1.028912	1.028912	0	0.0	0.0
2025-01-16 03:18:00+00:00	1.028912	1.028912	1.028912	1.028912	0	0.0	0.0

199 rows × 7 columns

Dans cette deuxième section, nous téléchargeons des données historiques pour la paire de devises EUR/USD à l'aide de la bibliothèque `yfinance`. Nous spécifions que nous souhaitons récupérer les données sur une période d'un jour avec un intervalle d'une minute. Les données téléchargées sont stockées dans un DataFrame que nous affichons ensuite pour inspection.

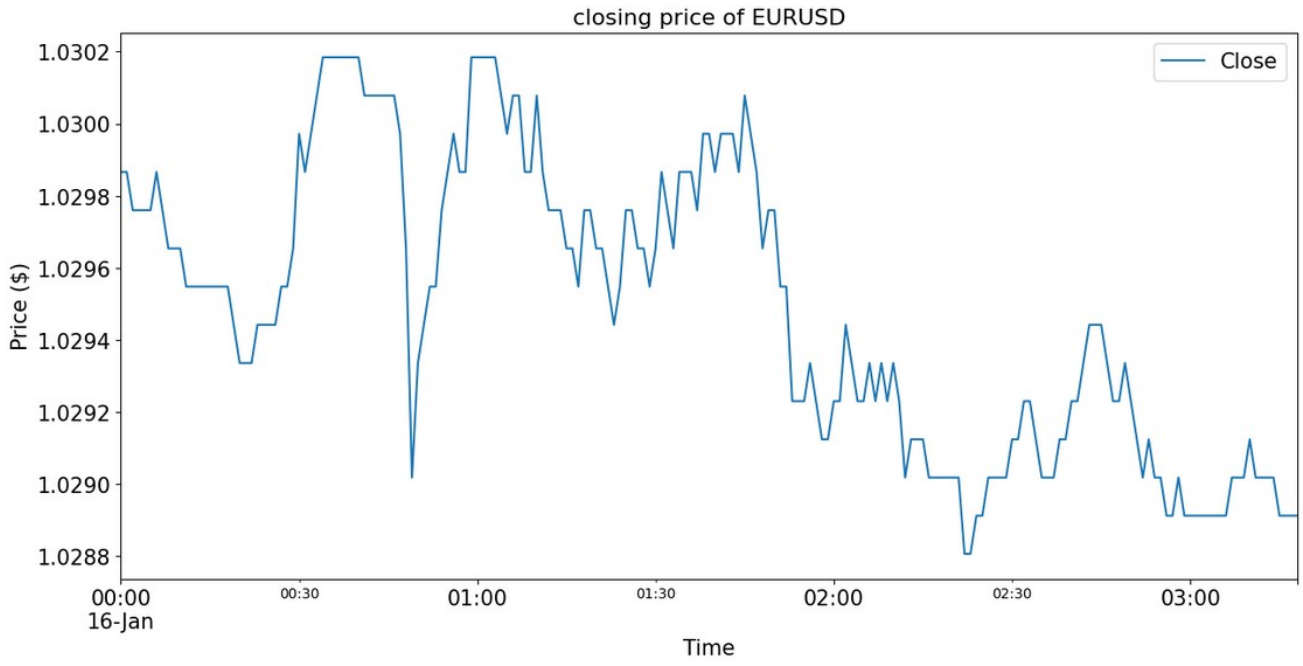
3. Sélection des données de bougie verte et rouge

```
green_df = df[df.Close > df.Open].copy()
green_df["Height"] = green_df["Close"] - green_df["Open"]

red_df = df[df.Close < df.Open].copy()
red_df["Height"] = red_df["Open"] – red_df["Close"]
```

Ici, nous filtrons les données pour créer deux DataFrames distincts : `green_df` pour les bougies vertes (où le prix de clôture est supérieur au prix d'ouverture) et `red_df` pour les bougies rouges (où le prix de clôture est inférieur au prix d'ouverture). Pour chaque DataFrame, nous calculons également la hauteur de la bougie, qui est la différence entre le prix de clôture et le prix d'ouverture.

4. Visualisation du prix de clôture

```
# Visualiser le prix de clôture de l'EUR/USD (chaque minute par jour)

# Tracer le prix de clôture ajusté
plt.figure(figsize=(15, 7))
df['Close'].plot()

# Définir les étiquettes et tailles du titre et des axes
plt.title('Prix de clôture de l\'EUR/USD', fontsize=16)
plt.xlabel('Temps', fontsize=15)
plt.ylabel('Prix ($)', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(['Close'], prop={'size': 15})

# Afficher le graphique
plt.show()
```



Dans cette section, nous visualisons le prix de clôture de l'EUR/USD sur un graphique. Nous configurons la taille de la figure et traçons la série de prix de clôture. Les axes sont étiquetés et le titre est défini pour donner un contexte au graphique. Enfin, nous affichons le graphique à l'utilisateur.

5. Création de moyennes mobiles simples

Création de moyennes mobiles simples

```
window = 3
SMA1 = "SMA-" + str(window)
df[SMA1] = df['Close'].rolling(window).mean()
colnames = ["Close", SMA1]
df2 = df[colnames]
df2.head()
```

Datetime	Close	SMA-3
2025-01-16 00:00:00+00:00	1.029866	NaN
2025-01-16 00:01:00+00:00	1.029866	NaN
2025-01-16 00:02:00+00:00	1.029760	1.029831
2025-01-16 00:03:00+00:00	1.029760	1.029795
2025-01-16 00:04:00+00:00	1.029760	1.029760

Nous créons une moyenne mobile simple (SMA) sur une fenêtre de trois périodes (minutes). Cette moyenne mobile est ajoutée au DataFrame d'origine, et nous extrayons ensuite les colonnes pertinentes pour créer un nouveau DataFrame `df2`, qui contient les prix de clôture et la SMA.

```
np.mean(df['Close'][:3])
```

```
1.029830773671468
```

6. Calcul de la moyenne mobile simple avec périodes minimales

Configurer l'argument min_periods dans la fonction rolling() pour contrôler le comportement aux fenêtres initiales avec des données incomplètes.
Par exemple, en définissant min_periods=1, le code précédent rapportera la valeur moyenne basée sur les données disponibles dans la fenêtre.

```
df['New_SMA'] = df['Close'].rolling(window, min_periods=1).mean()
df[colnames + ['New_SMA']].head()
```

Datetime	Close	SMA-3	New_SMA
2025-01-16 00:00:00+00:00	1.029866	NaN	1.029866
2025-01-16 00:01:00+00:00	1.029866	NaN	1.029866
2025-01-16 00:02:00+00:00	1.029760	1.029831	1.029831
2025-01-16 00:03:00+00:00	1.029760	1.029795	1.029795
2025-01-16 00:04:00+00:00	1.029760	1.029760	1.029760

Dans cette section, nous explorons la possibilité de configurer l'argument `min_periods` dans la méthode `rolling()` pour gérer les cas où les données sont incomplètes. En définissant `min_periods=1`, nous permettons à la moyenne mobile de calculer des valeurs même si toutes les données de la fenêtre ne sont pas disponibles.

7. Visualisation des prix et de la moyenne mobile

```
# Visualiser le prix de clôture et sa SMA
# Visualiser le prix original et la SMA sur trois périodes

# Couleurs pour le tracé de lignes
colors = ['blue', 'red']
# Tracé de la série originale et de la SMA
df2.plot(color=colors, linewidth=1.5, figsize=(12, 6))
# Modifier la taille des ticks
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.legend(labels=colnames, fontsize=13)
# Titre et étiquettes
plt.title('Prix de clôture et sa SMA', fontsize=20)
plt.xlabel('Date', fontsize=16)
plt.ylabel('Prix', fontsize=16)
```

Nous visualisons ici le prix de clôture original et la moyenne mobile sur trois périodes sur le même graphique. Nous utilisons des couleurs distinctes pour chaque ligne afin de faciliter la distinction entre elles. Les axes sont étiquetés, et nous ajoutons un titre pour contextualiser le graphique.

8. Création d'une moyenne mobile simple à 20 périodes

```
# Création de la SMA sur 20 périodes
window = 20
SMA2 = "SMA-" + str(window)
df2["SMA-" + SMA2] = df2['Close'].rolling(window).mean()
colnames = ["Adj Close", SMA1, SMA2]
```

Nous ajoutons une seconde moyenne mobile simple, cette fois sur une fenêtre de 20 périodes. Cela nous permet de comparer les tendances à court terme (SMA-3) et à long terme (SMA-20) dans nos analyses.

9. Superposition des moyennes mobiles sur le graphique

```
# Visualiser les prix avec les deux SMAs
# Visualiser les prix avec la SMA sur 3 périodes et la SMA sur 20 périodes

# Couleurs pour le tracé de lignes
colors = ['blue', 'red', 'green']
# Tracé des prix originaux et des SMAs
df2.plot(color=colors, linewidth=1.5, figsize=(12, 6))
# Modifier la taille des ticks
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.legend(labels=colnames, fontsize=13)
# Titre et étiquettes
plt.title('Prix de clôture et ses SMAs', fontsize=20)
plt.xlabel('Date', fontsize=16)
plt.ylabel('Prix', fontsize=16)
```

Dans cette section, nous superposons les deux moyennes mobiles sur le graphique des prix de clôture. Cela nous permet de visualiser comment les différentes moyennes mobiles réagissent aux fluctuations des prix et d'identifier les tendances.

10. Introduction aux moyennes mobiles exponentielles

Les moyennes mobiles exponentielles (EMA), également connues sous le nom de moyennes mobiles pondérées exponentiellement (EWMA),
sont un autre type de moyenne mobile qui accorde un poids plus élevé et une plus grande signification aux données les plus récentes.
Ceci est une différence clé par rapport à la moyenne mobile simple, qui accorde un poids égal à tous les points de données dans la période.
L'EMA est largement utilisée pour réduire le bruit dans les données et identifier les tendances à long terme.

Nous introduisons ici le concept de moyennes mobiles exponentielles (EMA), qui diffèrent des moyennes mobiles simples par leur capacité à accorder plus de poids aux données récentes. Cela les rend plus réactives aux changements récents du marché et utiles pour identifier les tendances.

11. Calcul de l'EMA

Le facteur de lissage α varie entre zéro et un.
Ce facteur détermine le poids donné à l'observation la plus récente par rapport aux EMA existantes.
Un α plus élevé accentue les prix récents plus fortement.

Pour la première valeur d'EWMA à t = 0, un choix par défaut est de définir EWMA0 = S0.

Cela signifie que l'EMA suppose que les données récentes sont plus pertinentes que les anciennes.

Nous expliquons ici comment calculer l'EMA. Le facteur de lissage α est essentiel pour déterminer la réactivité de l'EMA. Un choix par défaut est d'initialiser la première valeur de l'EMA avec le premier prix.

12. Création de la série EMA

L'EMA peut être calculée en appelant la méthode ewm() sur un objet de série Pandas, suivie de l'extraction de la valeur moyenne via mean().
Nous pouvons définir l'argument alpha dans ewm() pour contrôler directement l'importance de l'observation actuelle par rapport aux historiques.

```
alpha = 0.1
df2['EWM_' + str(alpha)] = df2['Close'].ewm(alpha=alpha, adjust=False).mean()
df2.head()
```

Datetime	Close	SMA-3	SMA-SMA-20	EWM_0.1
2025-01-16 00:00:00+00:00	1.029866	NaN	NaN	1.029866
2025-01-16 00:01:00+00:00	1.029866	NaN	NaN	1.029866
2025-01-16 00:02:00+00:00	1.029760	1.029831	NaN	1.029856
2025-01-16 00:03:00+00:00	1.029760	1.029795	NaN	1.029846
2025-01-16 00:04:00+00:00	1.029760	1.029760	NaN	1.029837

Nous calculons l'EMA en utilisant la méthode `ewm()` sur la série de prix de clôture. Nous définissons un α de 0,1 pour donner plus de poids aux prix récents. La série EMA résultante est ajoutée à notre DataFrame.

13. Création d'une autre série EMA

Créons une autre série EMA avec α = 0.5.
Cela signifie que nous attribuons un poids égal à l'observation actuelle et aux historiques.

```
alpha = 0.5
df2['EWM_' + str(alpha)] = df2['Close'].ewm(alpha=alpha, adjust=False).mean()
```

df2.head()

Datetime	Close	SMA-3	SMA-SMA-20	EWM_0.1	EWM_0.5
2025-01-16 00:00:00+00:00	1.029866	NaN	NaN	1.029866	1.029866
2025-01-16 00:01:00+00:00	1.029866	NaN	NaN	1.029866	1.029866
2025-01-16 00:02:00+00:00	1.029760	1.029831	NaN	1.029856	1.029813
2025-01-16 00:03:00+00:00	1.029760	1.029795	NaN	1.029846	1.029787
2025-01-16 00:04:00+00:00	1.029760	1.029760	NaN	1.029837	1.029773

Nous ajoutons une autre série EMA, cette fois avec un α de 0,5, ce qui signifie que l'observation actuelle et les anciennes observations sont considérées également. Cela nous permet de comparer l'impact de différents facteurs de lissage sur l'EMA.

14. Visualisation de toutes les moyennes mobiles

```
# Visualiser toutes les moyennes mobiles ensemble
# Visualiser les prix de clôture avec les SMA et EMA de différentes configurations

df2.plot(linewidth=1.5, figsize=(12, 6))
plt.title('Prix de clôture avec SMA et EMA', fontsize=20)
plt.xlabel('Date', fontsize=16)
plt.ylabel('Prix', fontsize=16)
```

Nous visualisons toutes les moyennes mobiles (SMA et EMA) sur le même graphique avec les prix de clôture. Cela nous aide à analyser la relation entre les différentes moyennes et les mouvements de prix.

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 199 entries, 2025-01-16 00:00:00+00:00 to 2025-01-16 03:18:00+00:00
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Close       199 non-null    float64
 1   SMA-3       197 non-null    float64
 2   SMA-SMA-20  180 non-null    float64
 3   EWM_0.1     199 non-null    float64
 4   EWM_0.5     199 non-null    float64
dtypes: float64(5)
memory usage: 9.3 KB
```

15. Mise en œuvre de la stratégie de suivi de tendance

La stratégie de suivi de tendance qui repose sur les moyennes mobiles fonctionne ainsi.
Il y aura deux moyennes mobiles : une moyenne mobile à court terme et une moyenne mobile à long terme.
Lorsque la moyenne mobile à court terme croise au-dessus de la moyenne mobile à long terme, cela signale une action d'achat, et le trader entre en position longue sur l'actif.
Lorsque la moyenne mobile à court terme croise en dessous de la moyenne mobile à long terme, cela signale une action de vente, et le trader entre en position courte sur l'actif.

Nous introduisons ici la stratégie de suivi de tendance basée sur les croisements de moyennes mobiles. Cette stratégie repose sur l'idée que les croisements peuvent signaler des opportunités d'achat ou de vente.

16. Création des signaux d'achat et de vente

Nous allons utiliser SMA-3 et SMA-20 comme moyennes mobiles respectives à court terme et à long terme,
dont le croisement générera un signal de trading.

df2['SMA-3'] = df2['SMA-3'].shift(1)
df2['SMA-SMA-20'] = df2['SMA-SMA-20'].shift(1)

Nous décalons les moyennes mobiles pour éviter d'utiliser les informations futures lors de la prise de décisions de trading. Cela garantit que les signaux sont basés uniquement sur les données disponibles jusqu'à présent.

17. Application de la règle de trading

Implémentons la règle de trading : acheter si SMA-3 > SMA-20, et vendre si SMA-3 < SMA-20.
Cette condition peut être créée en utilisant la fonction np.where().

df2['signal'] = np.where(df2['SMA-3'] > df2['SMA-SMA-20'], 1, 0)
df2['signal'] = np.where(df2['SMA-3'] < df2['SMA-SMA-20'], -1, df2['signal'])
df2.dropna(inplace=True)

Nous appliquons la règle de trading en créant une colonne de signaux. Un signal d'achat est représenté par 1, un signal de vente par -1, et aucune action par 0. Nous nettoyons également le DataFrame en supprimant les lignes contenant des valeurs manquantes.

```
# We can check the frequency distribution of the signal column as follows:

df2['signal'].value_counts()

# The result shows that there are more declining days than inclining days, which confirms the downward trending price series shown earlier
```

```
signal
-1    99
 1    80
Name: count, dtype: int64
```

18. Analyse des retours

Nous introduisons une stratégie de référence appelée achat et conservation,
et nous utiliserons le retour logarithmique au lieu du retour brut pour faciliter les calculs.

df2['log_return_buy_n_hold'] = np.log(df2['Close']).diff()

Nous calculons le retour logarithmique pour la stratégie d'achat et de conservation. Ce type de retour est souvent préféré en finance car il permet d'analyser les performances de manière plus précise.

19. Calcul du retour de la stratégie de suivi de tendance

```
# Maintenant, nous allons calculer le retour logarithmique de la stratégie de suivi
de tendance.
# Rappelons que la colonne de signal représente si nous allons long (valeur 1) ou
court (valeur -1) à chaque période.

df2['log_return_trend_follow'] = df2['signal'] * df2['log_return_buy_n_hold']
```

Nous calculons le retour logarithmique pour la stratégie de suivi de tendance en multipliant le signal par le retour logarithmique de la stratégie d'achat et de conservation. Cela nous permet de quantifier les performances de notre stratégie.

20. Actions de trading explicites

```
# La colonne de signal nous indique si nous devrions aller long ou court dans
l'actif sous la stratégie de suivi de tendance.
# Cependant, cela ne signifie pas que nous devons effectuer un trade à chaque
période.
# Si le signal reste le même pour deux périodes consécutives, nous restons dans la
position.

df2['action'] = df2.signal.diff()
```

Nous définissons les actions de trading en utilisant la différence des signaux. Cela nous permet de savoir quand un changement de position est nécessaire, soit pour passer d'une position longue à courte, soit vice versa.

```
# We can produce a frequency count of different trading actions using the value_counts() function:

df2['action'].value_counts()
```

```
action
 0.0    162
 2.0      8
-2.0      8
Name: count, dtype: int64
```

21. Visualisation des actions de trading

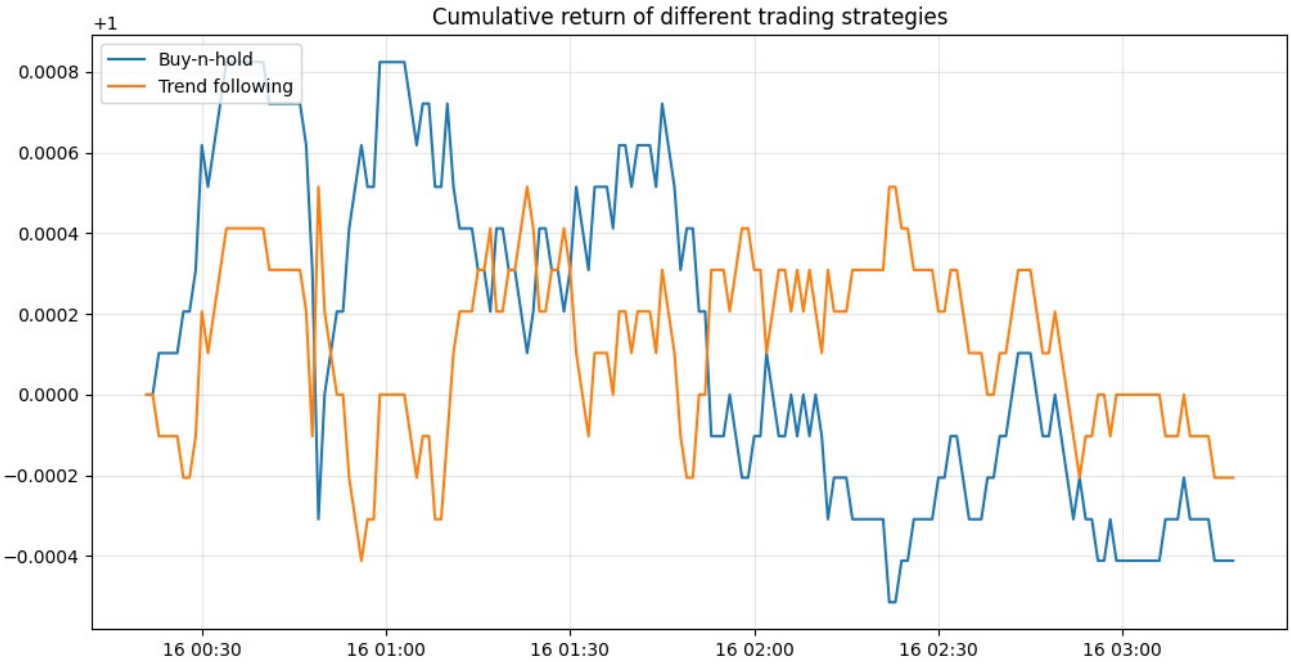
```
# Nous pouvons visualiser ces actions de trading sous forme de triangles sur le
graphique avec les prix des actions et les SMAs.
# Nous indiquons une action d'achat via un triangle vert pointant vers le haut
lorsque la SMA à court terme croise au-dessus de la SMA à long terme.
```

```
plt.rcParams['figure.figsize'] = 12, 6
plt.grid(True, alpha=.3)
plt.plot(df2['Close'], label='Close')
plt.plot(df2['SMA-3'], label='SMA-3')
plt.plot(df2['SMA-SMA-20'], label='SMA-SMA-20')
plt.plot(df2.loc[df2.action == 2].index, df2['SMA-3'][df2.action == 2], '^',
color='g', markersize=12)
plt.plot(df2[df2.action == -2].index, df2['SMA-SMA-20'][df2.action == -2], 'v',
color='r', markersize=12)
plt.legend(loc=1);
```

Nous visualisons les actions de trading sur le graphique en utilisant des triangles verts pour les achats et rouges pour les ventes. Cela permet d'illustrer clairement les points d'entrée et de sortie de notre stratégie.

22. Analyse des rendements cumulés

```
# Analysons les rendements cumulés de chaque période pour les deux stratégies de
trading.
# Nous aimerions obtenir le retour final en pourcentage, comparant les deux
stratégies.

plt.plot(np.exp(df2['log_return_buy_n_hold']).cumprod(), label='Buy-n-hold')
plt.plot(np.exp(df2['log_return_trend_follow']).cumprod(), label='Trend following')
plt.legend(loc=2)
plt.title("Retour cumulatif de différentes stratégies de trading")
plt.grid(True, alpha=.3)
```



Nous comparons les rendements cumulés des deux stratégies de trading sur un graphique. Cela nous permet d'évaluer visuellement la performance relative de la stratégie d'achat et de conservation par rapport à la stratégie de suivi de tendance. En traçant les courbes des rendements cumulés, nous pouvons observer comment chaque stratégie aurait évolué au fil du temps, fournissant ainsi un aperçu clair de leur efficacité respective.

23. Calcul des rendements terminaux

```
# Calcul du retour terminal de la stratégie d'achat et de conservation
np.exp(df2['log_return_buy_n_hold']).cumprod()[-1] - 1

Output : -0.0004115952186884986
```

Nous calculons le retour terminal pour la stratégie d'achat et de conservation. Ce retour est obtenu en exponentiant le retour logarithmique cumulé et en soustrayant 1. Cela nous donne une mesure finale de la performance de cette stratégie sur la période analysée.

24. Calcul des rendements terminaux pour la stratégie de suivi de tendance

```
# Calcul du retour terminal de la stratégie de suivi de tendance
np.exp(df2['log_return_trend_follow']).cumprod()[-1] - 1

Output : -0.00020590451978175217
```

De même, nous calculons le retour terminal pour la stratégie de suivi de tendance en utilisant la même méthode. Cela nous permet de comparer directement les performances des deux stratégies en termes de rendement global sur la période étudiée.

returns = df.Close.pct_change()

returns

```
Datetime
2025-01-16 00:00:00+00:00         NaN
2025-01-16 00:01:00+00:00    0.000000
2025-01-16 00:02:00+00:00   -0.000103
2025-01-16 00:03:00+00:00    0.000000
2025-01-16 00:04:00+00:00    0.000000
                                ...
2025-01-16 03:14:00+00:00    0.000000
2025-01-16 03:15:00+00:00   -0.000103
2025-01-16 03:16:00+00:00    0.000000
2025-01-16 03:17:00+00:00    0.000000
2025-01-16 03:18:00+00:00    0.000000
Name: Close, Length: 199, dtype: float64
```

terminal_return = df.Close.iloc[-1] / df.Close.iloc[0] - 1

terminal_return

Output : -0.0009260177774578215

cum_returns = (1+returns).cumprod() - 1

cum_returns

```
Datetime
2025-01-16 00:00:00+00:00         NaN
2025-01-16 00:01:00+00:00    0.000000
2025-01-16 00:02:00+00:00   -0.000103
2025-01-16 00:03:00+00:00   -0.000103
2025-01-16 00:04:00+00:00   -0.000103
                                ...
2025-01-16 03:14:00+00:00   -0.000823
2025-01-16 03:15:00+00:00   -0.000926
2025-01-16 03:16:00+00:00   -0.000926
2025-01-16 03:17:00+00:00   -0.000926
2025-01-16 03:18:00+00:00   -0.000926
Name: Close, Length: 199, dtype: float64
```

cum_returns.values[-1] == terminal_return

Output : False

log_returns = np.log(1+returns)

log_returns

```
Datetime
2025-01-16 00:00:00+00:00         NaN
2025-01-16 00:01:00+00:00    0.000000
2025-01-16 00:02:00+00:00   -0.000103
2025-01-16 00:03:00+00:00    0.000000
2025-01-16 00:04:00+00:00    0.000000
                                 ...
2025-01-16 03:14:00+00:00    0.000000
2025-01-16 03:15:00+00:00   -0.000103
2025-01-16 03:16:00+00:00    0.000000
2025-01-16 03:17:00+00:00    0.000000
2025-01-16 03:18:00+00:00    0.000000
Name: Close, Length: 199, dtype: float64
```

cum_return2 = np.exp(log_returns.cumsum()) - 1

cum_return2

```
Datetime
2025-01-16 00:00:00+00:00         NaN
2025-01-16 00:01:00+00:00    0.000000
2025-01-16 00:02:00+00:00   -0.000103
2025-01-16 00:03:00+00:00   -0.000103
2025-01-16 00:04:00+00:00   -0.000103
                                 ...
2025-01-16 03:14:00+00:00   -0.000823
2025-01-16 03:15:00+00:00   -0.000926
2025-01-16 03:16:00+00:00   -0.000926
2025-01-16 03:17:00+00:00   -0.000926
2025-01-16 03:18:00+00:00   -0.000926
Name: Close, Length: 199, dtype: float64
```

cum_return2.values[-1] == terminal_return

Output : False

25. Résumé des résultats

```
# Résumé des résultats des stratégies
# Nous avons couvert les bases de la stratégie populaire de suivi de tendance et
son implémentation en Python.
# Nous avons commencé par un exercice sur les rendements logarithmiques puis avons
transitionné vers différentes moyennes mobiles comme indicateurs techniques
couramment utilisés.
# Enfin, nous avons discuté de la génération de signaux de trading et du calcul des
métriques de performance à l'aide de cette stratégie,
# qui servira de bonne base alors que nous explorons d'autres candidats par la
suite.
```

Dans cette dernière section, nous faisons un récapitulatif des concepts et des techniques que nous avons abordés. Nous avons exploré les principes fondamentaux de la stratégie de suivi de tendance, y compris l'utilisation des moyennes mobiles comme indicateurs techniques, la génération de signaux de trading et l'évaluation des performances à travers les rendements logarithmiques. Ce travail constitue une base solide pour le développement et l'exploration de stratégies de trading plus avancées à l'avenir.

En conclusion, ce code présente une approche systématique pour développer une stratégie de trading algorithmique sur le Forex, mettant l'accent sur l'utilisation des données historiques, des indicateurs techniques et des méthodes de gestion des risques. L'analyse des résultats permet d'évaluer l'efficacité des stratégies mises en œuvre et d'orienter les décisions futures en matière de trading.

Introduction

Le code présenté ci-dessous a pour objectif d'identifier les motifs d'engulfing dans les bougies japonaises sur les données de la paire de devises EUR/USD. Ces motifs sont souvent utilisés dans l'analyse technique pour signaler des retournements de tendance potentiels. Le code permet non seulement de détecter ces motifs, mais aussi d'évaluer leur efficacité en calculant le pourcentage de réussite associé à chaque signal. En outre, il intègre des indicateurs de tendance pour affiner les décisions de trading.

Analyse du Code

1. Chargement et préparation des données

```
import pandas as pd
df = pd.read_csv("/home/ab/Downloads/EURUSD_Candlestick_1_D_ASK_05.05.2003-
19.10.2019.csv")
df.tail()
```

	Local time	open	high	low	close	volume
4322	14.10.2019 00:00:00.000 GMT+0300	1.10399	1.10428	1.10129	1.10281	150329.9600
4323	15.10.2019 00:00:00.000 GMT+0300	1.10281	1.10463	1.09915	1.10338	225114.4925
4324	16.10.2019 00:00:00.000 GMT+0300	1.10331	1.10857	1.10226	1.10720	224179.4392
4325	17.10.2019 00:00:00.000 GMT+0300	1.10724	1.11400	1.10653	1.11254	221922.6500
4326	18.10.2019 00:00:00.000 GMT+0300	1.11256	1.11727	1.11149	1.11723	143073.7750

Dans cette première cellule, nous chargeons les données historiques des bougies japonaises pour la paire EUR/USD à partir d'un fichier CSV. Nous utilisons la bibliothèque `pandas` pour manipuler les données et affichons les cinq dernières lignes du DataFrame pour vérifier que les données sont correctement chargées.

2. Vérification des valeurs manquantes

```
# Vérifier s'il y a des valeurs NA dans les données
df.isna().sum()
```

```
Local time    0
open          0
high          0
low           0
close         0
volume        0
dtype: int64
```

Ici, nous vérifions la présence de valeurs manquantes dans le DataFrame. Cela est crucial pour s'assurer que nos analyses ultérieures ne soient pas affectées par des données incomplètes.

3. Identification des motifs d'engulfing

```
# Signaux de motifs engulfing
import random

def Revsignal1(df1):
    length = len(df1)
    high = list(df1['high'])
    low = list(df1['low'])
    close = list(df1['close'])
    open = list(df1['open'])
    signal = [0] * length
    bodydiff = [0] * length

    for row in range(1, length):
        bodydiff[row] = abs(open[row] - close[row])
        bodydiffmin = 0.003
        if (bodydiff[row] > bodydiffmin and bodydiff[row - 1] > bodydiffmin and
            open[row - 1] < close[row - 1] and
            open[row] > close[row] and
            (open[row] - close[row - 1]) >= +0e-5 and close[row] < open[row - 1]):
            signal[row] = 1  # Signal d'engulfing haussier
        elif (bodydiff[row] > bodydiffmin and bodydiff[row - 1] > bodydiffmin and
              open[row - 1] > close[row - 1] and
              open[row] < close[row] and
              (open[row] - close[row - 1]) <= -0e-5 and close[row] > open[row -
1]):
            signal[row] = 2  # Signal d'engulfing baissier
        else:
            signal[row] = 0  # Pas de signal
    return signal

df['signal1'] = Revsignal1(df)
df[df['signal1'] == 1].count()
```

```
Local time    131
open          131
high          131
low           131
close         131
volume        131
signal1       131
dtype: int64
```

Dans cette section, nous définissons une fonction `Revsignal1` qui détecte les motifs d'engulfing dans les données de bougies. Nous calculons la différence entre le prix d'ouverture et le prix de clôture pour chaque bougie. Ensuite, nous appliquons des conditions pour identifier les motifs d'engulfing haussiers et baissiers. Les signaux sont stockés dans une nouvelle colonne `signal1` dans le DataFrame. Enfin, nous comptons le nombre de signaux haussiers détectés.

4. Définition de l'objectif de tendance

```
# Définition de l'objectif
def mytarget(df1, barsfront):
    length = len(df1)
    high = list(df1['high'])
    low = list(df1['low'])
    close = list(df1['close'])
    open = list(df1['open'])
    trendcat = [None] * length

    piplim = 300e-5
    for line in range(0, length - 1 - barsfront):
        for i in range(1, barsfront + 1):
            if ((high[line + i] - max(close[line], open[line])) > piplim) and
((min(close[line], open[line]) - low[line + i]) > piplim):
                trendcat[line] = 3  # Pas de tendance
            elif (min(close[line], open[line]) - low[line + i]) > piplim:
                trendcat[line] = 1  # Tendance baissière
                break
            elif (high[line + i] - max(close[line], open[line])) > piplim:
                trendcat[line] = 2  # Tendance haussière
                break
            else:
                trendcat[line] = 0  # Pas de tendance claire
    return trendcat

df['Trend'] = mytarget(df, 3)
```

Nous définissons la fonction `mytarget`, qui évalue la tendance du marché en fonction des bougies suivantes. Cette fonction détermine si le marché est en tendance haussière, baissière ou sans tendance, en se basant sur des seuils de variation de prix (pip). Les résultats sont stockés dans une nouvelle colonne `Trend` dans le DataFrame.

5. Évaluation des résultats

```
import numpy as np

conditions = [(df['Trend'] == 1) & (df['signal1'] == 1), (df['Trend'] == 2) &
(df['signal1'] == 2)]
values = [1, 2]
df['result'] = np.select(conditions, values)

trendId = 2
print(df[df['result'] == trendId].result.count() / df[df['signal1'] ==
trendId].signal1.count())
df[(df['Trend'] != trendId) & (df['signal1'] == trendId)]  # faux positifs
```

Output : 0.8205128205128205

	Local time	open	high	low	close	volume	signal1	Trend	result
224	11.03.2004 00:00:00.000 GMT+0200	1.22216	1.23841	1.21627	1.23504	1.129471e+06	2	1.0	0
417	03.12.2004 00:00:00.000 GMT+0200	1.32682	1.34604	1.32513	1.34505	1.100705e+06	2	1.0	0
675	28.11.2005 00:00:00.000 GMT+0200	1.17096	1.19015	1.16806	1.18475	1.080417e+06	2	0.0	0
1049	27.04.2007 00:00:00.000 GMT+0300	1.36020	1.36830	1.35860	1.36540	1.024237e+06	2	1.0	0
1609	11.06.2009 00:00:00.000 GMT+0300	1.39863	1.41730	1.39440	1.41095	5.543684e+05	2	1.0	0
1689	30.09.2009 00:00:00.000 GMT+0300	1.45867	1.46739	1.45734	1.46376	7.987590e+04	2	1.0	0
2133	07.06.2011 00:00:00.000 GMT+0300	1.45766	1.46964	1.45646	1.46923	2.083190e+05	2	1.0	0
2174	03.08.2011 00:00:00.000 GMT+0300	1.42040	1.43448	1.41460	1.43243	3.184016e+05	2	1.0	0
2215	29.09.2011 00:00:00.000 GMT+0300	1.35433	1.36784	1.35205	1.35985	2.839137e+05	2	1.0	0
2244	08.11.2011 00:00:00.000 GMT+0200	1.37771	1.38480	1.37252	1.38339	2.502527e+05	2	1.0	0
2292	12.01.2012 00:00:00.000 GMT+0200	1.27072	1.28452	1.26999	1.28151	3.078163e+05	2	1.0	0
2607	21.03.2013 23:00:00.000 GMT+0200	1.28992	1.30077	1.28886	1.29886	2.482676e+05	2	1.0	0
2624	16.04.2013 00:00:00.000 GMT+0300	1.30351	1.32020	1.30298	1.31781	2.836193e+05	2	1.0	0
3082	09.01.2015 00:00:00.000 GMT+0200	1.17929	1.18464	1.17630	1.18425	1.665926e+05	2	1.0	0
3204	29.06.2015 00:00:00.000 GMT+0300	1.10073	1.12786	1.09550	1.12362	2.400577e+05	2	0.0	0
3213	10.07.2015 00:00:00.000 GMT+0300	1.10377	1.12164	1.10322	1.11584	2.122633e+05	2	1.0	0
3228	31.07.2015 00:00:00.000 GMT+0300	1.09335	1.11144	1.09213	1.09861	1.927528e+05	2	1.0	0
3707	02.06.2017 00:00:00.000 GMT+0300	1.12130	1.12852	1.12052	1.12851	1.691527e+05	2	0.0	0
3860	04.01.2018 00:00:00.000 GMT+0200	1.20146	1.20890	1.20046	1.20685	2.100527e+05	2	1.0	0
4034	05.09.2018 00:00:00.000 GMT+0300	1.15828	1.16401	1.15430	1.16303	2.581394e+05	2	1.0	0
4148	12.02.2019 00:00:00.000 GMT+0200	1.12763	1.13398	1.12579	1.13259	4.096074e+05	2	1.0	0

Dans cette section, nous évaluons l'efficacité des signaux d'engulfing en les comparant aux tendances identifiées. Nous créons une nouvelle colonne `result` qui indique si un signal d'engulfing est en accord avec la tendance. Ensuite, nous calculons le pourcentage de signaux corrects par rapport au nombre total de signaux d'engulfing. Nous affichons ce pourcentage et identifions les faux positifs.

6. Visualisation des bougies

```
Copy
dfpl = df[400:450]
import plotly.graph_objects as go
from datetime import datetime
```

```
fig = go.Figure(data=[go.Candlestick(x=dfpl.index,
                open=dfpl['open'],
                high=dfpl['high'],
                low=dfpl['low'],
                close=dfpl['close'])])

fig.show()
```

Enfin, nous visualisons une portion des données sous forme de graphique en chandeliers à l'aide de `plotly`. Cela permet de visualiser les motifs d'engulfing détectés sur un échantillon de données, facilitant ainsi l'analyse visuelle des signaux générés.

Résumé

Le code permet d'identifier et d'analyser les motifs d'engulfing dans les données de bougies japonaises pour la paire EUR/USD. Après avoir chargé et préparé les données, nous avons développé une méthode pour détecter ces motifs, tout en évaluant leur efficacité en fonction des tendances du marché. La visualisation des bougies offre un aperçu pratique des signaux détectés, fournissant ainsi une base solide pour le développement de stratégies de trading basées sur ces motifs.

Introduction

La visualisation des données financières, en particulier à travers des graphiques en chandeliers, est essentielle pour les traders et les analystes. Les graphiques en chandeliers fournissent une représentation visuelle des mouvements de prix sur une période donnée, permettant d'identifier des motifs et des tendances qui peuvent influencer les décisions de trading. Dans ce contexte, nous allons explorer comment créer manuellement des graphiques en chandeliers en utilisant Python et la

bibliothèque Matplotlib. De plus, nous allons également rechercher des motifs spécifiques, tels que les motifs d'engulfing, qui sont souvent utilisés pour prédire les retournements de tendance.

Analyse du Code

1. Chargement des bibliothèques nécessaires

```
import matplotlib
print("Matplotlib Version : {}".format(matplotlib.__version__))

Output : Matplotlib Version : 3.9.2
```

Ici, nous importons la bibliothèque Matplotlib et vérifions sa version. Cela nous assure que nous utilisons une version compatible pour créer nos graphiques.

2. Chargement des données

```
Copy
import pandas as pd

# Chargement des données EUR/USD
eurusd_df =
pd.read_csv("/home/ab/Downloads/EURUSD_Daily_200001030000_201912310000.csv",
index_col=0, parse_dates=True)

# Réinitialisation de l'index pour faciliter l'accès aux colonnes
eurusd_df = eurusd_df.reset_index()

# Affichage des premières lignes pour vérifier la structure des données
eurusd_df.head()
```

	Date	Open	High	Low	Close	Tickvol	Vol	Spread
0	2000-01-03	1.0073	1.0278	1.0054	1.0246	6642	0	50
1	2000-01-04	1.0243	1.0340	1.0213	1.0292	7339	0	50
2	2000-01-05	1.0293	1.0402	1.0284	1.0326	6570	0	50
3	2000-01-06	1.0325	1.0415	1.0272	1.0330	7223	0	50
4	2000-01-07	1.0329	1.0332	1.0260	1.0298	5689	0	50

Nous chargeons les données historiques de la paire de devises EUR/USD à partir d'un fichier CSV. Nous réinitialisons l'index pour faciliter l'accès aux colonnes et affichons les premières lignes du DataFrame pour vérifier que les données sont correctement chargées.

3. Sélection des données

```
# Sélectionner les 30 dernières lignes des données EUR/USD
eurusd_df_small = eurusd_df[-30:]
```

```
# Afficher les 30 dernières lignes
eurusd_df_small
```

	Date	Open	High	Low	Close	Tickvol	Vol	Spread
5169	2019-11-19	1.10718	1.10838	1.10624	1.10780	26644	0	1
5170	2019-11-20	1.10777	1.10813	1.10530	1.10723	34131	0	1
5171	2019-11-21	1.10723	1.10969	1.10521	1.10581	31829	0	1
5172	2019-11-22	1.10580	1.10875	1.10144	1.10182	32645	0	1
5173	2019-11-25	1.10151	1.10320	1.10035	1.10126	27995	0	1
5174	2019-11-26	1.10129	1.10255	1.10072	1.10194	28638	0	1
5175	2019-11-27	1.10194	1.10246	1.09923	1.09989	29482	0	1
5176	2019-11-28	1.09987	1.10178	1.09986	1.10075	20746	0	1
5177	2019-11-29	1.10071	1.10282	1.09810	1.10152	27604	0	1
5178	2019-12-02	1.10241	1.10910	1.10029	1.10784	31480	0	1
5179	2019-12-03	1.10782	1.10934	1.10658	1.10809	32021	0	1
5180	2019-12-04	1.10809	1.11161	1.10667	1.10775	35509	0	1
5181	2019-12-05	1.10772	1.11079	1.10771	1.11042	27845	0	1
5182	2019-12-06	1.11042	1.11097	1.10396	1.10595	27888	0	1
5183	2019-12-09	1.10574	1.10779	1.10533	1.10635	21381	0	1
5184	2019-12-10	1.10635	1.10976	1.10627	1.10919	27293	0	1
5185	2019-12-11	1.10914	1.11447	1.10701	1.11294	33856	0	1
5186	2019-12-12	1.11294	1.11541	1.11030	1.11286	44761	0	1
5187	2019-12-13	1.11294	1.11993	1.11032	1.11173	63101	0	1
5188	2019-12-16	1.11259	1.11580	1.11226	1.11434	30624	0	1
5189	2019-12-17	1.11435	1.11745	1.11291	1.11487	35410	0	1
5190	2019-12-18	1.11483	1.11541	1.11102	1.11127	31490	0	1
5191	2019-12-19	1.11126	1.11442	1.11072	1.11204	34743	0	1
5192	2019-12-20	1.11201	1.11246	1.10662	1.10751	29734	0	1
5193	2019-12-23	1.10750	1.10958	1.10698	1.10895	29202	0	1

```
et ainsi de suite jusqu’a 30 derniers…
```

Nous sélectionnons les 30 dernières lignes des données pour une analyse plus ciblée. Cela nous permet de visualiser un échantillon récent des mouvements de prix.

4. Identification des bougies

```
# Sélectionner les bougies "vertes" (Close > Open)
```

```
green_df = eurusd_df_small[eurusd_df_small.Close > eurusd_df_small.Open].copy()
green_df["Height"] = green_df["Close"] - green_df["Open"]

# Sélectionner les bougies "rouges" (Close < Open)
red_df = eurusd_df_small[eurusd_df_small.Close < eurusd_df_small.Open].copy()
red_df["Height"] = red_df["Open"] – red_df["Close"]
```

```
green_df.head()
```

	Date	Open	High	Low	Close	Tickvol	Vol	Spread	Height
5169	2019-11-19	1.10718	1.10838	1.10624	1.10780	26644	0	1	0.00062
5174	2019-11-26	1.10129	1.10255	1.10072	1.10194	28638	0	1	0.00065
5176	2019-11-28	1.09987	1.10178	1.09986	1.10075	20746	0	1	0.00088
5177	2019-11-29	1.10071	1.10282	1.09810	1.10152	27604	0	1	0.00081
5178	2019-12-02	1.10241	1.10910	1.10029	1.10784	31480	0	1	0.00543

```
red_df.head()
```

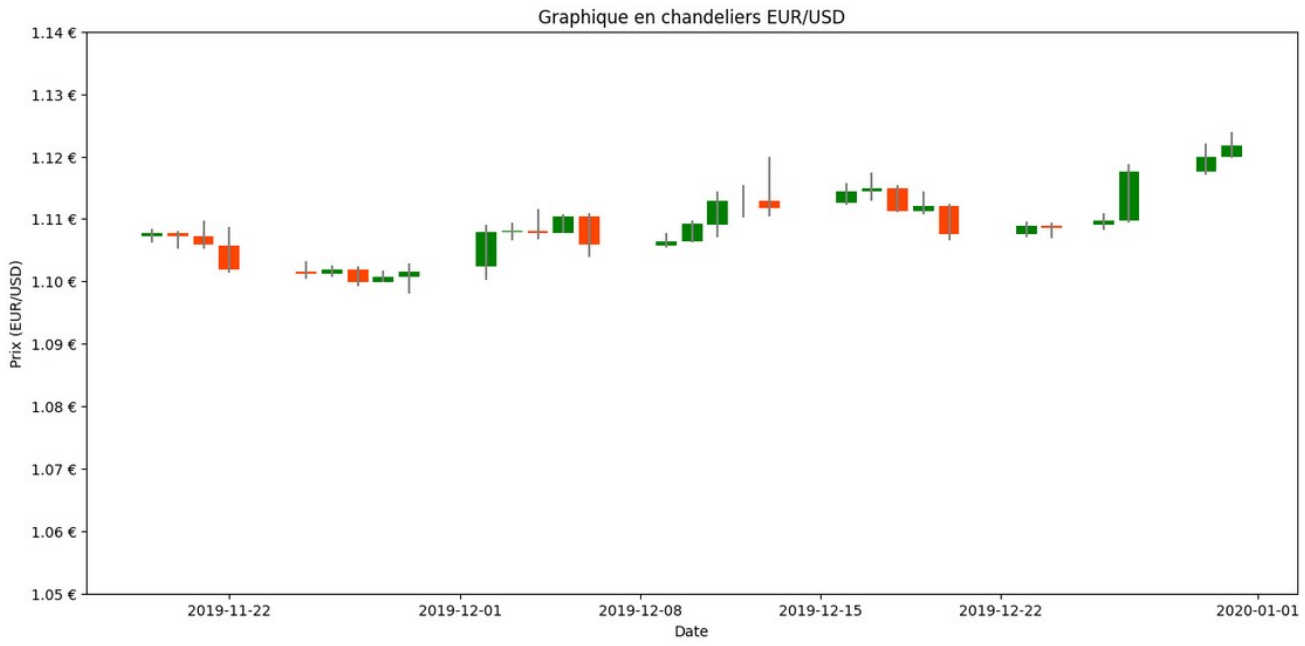
	Date	Open	High	Low	Close	Tickvol	Vol	Spread	Height
5170	2019-11-20	1.10777	1.10813	1.10530	1.10723	34131	0	1	0.00054
5171	2019-11-21	1.10723	1.10969	1.10521	1.10581	31829	0	1	0.00142
5172	2019-11-22	1.10580	1.10875	1.10144	1.10182	32645	0	1	0.00398
5173	2019-11-25	1.10151	1.10320	1.10035	1.10126	27995	0	1	0.00025
5175	2019-11-27	1.10194	1.10246	1.09923	1.09989	29482	0	1	0.00205

Nous identifions les bougies vertes (haussières) et rouges (baissières) en comparant les prix de clôture et d'ouverture. Nous calculons également la hauteur de chaque bougie, qui est essentielle pour la visualisation.

5. Création du graphique en chandeliers

```
import matplotlib.pyplot as plt
import numpy as np

# Créer la figure pour afficher le graphique
fig = plt.figure(figsize=(15, 7))
```

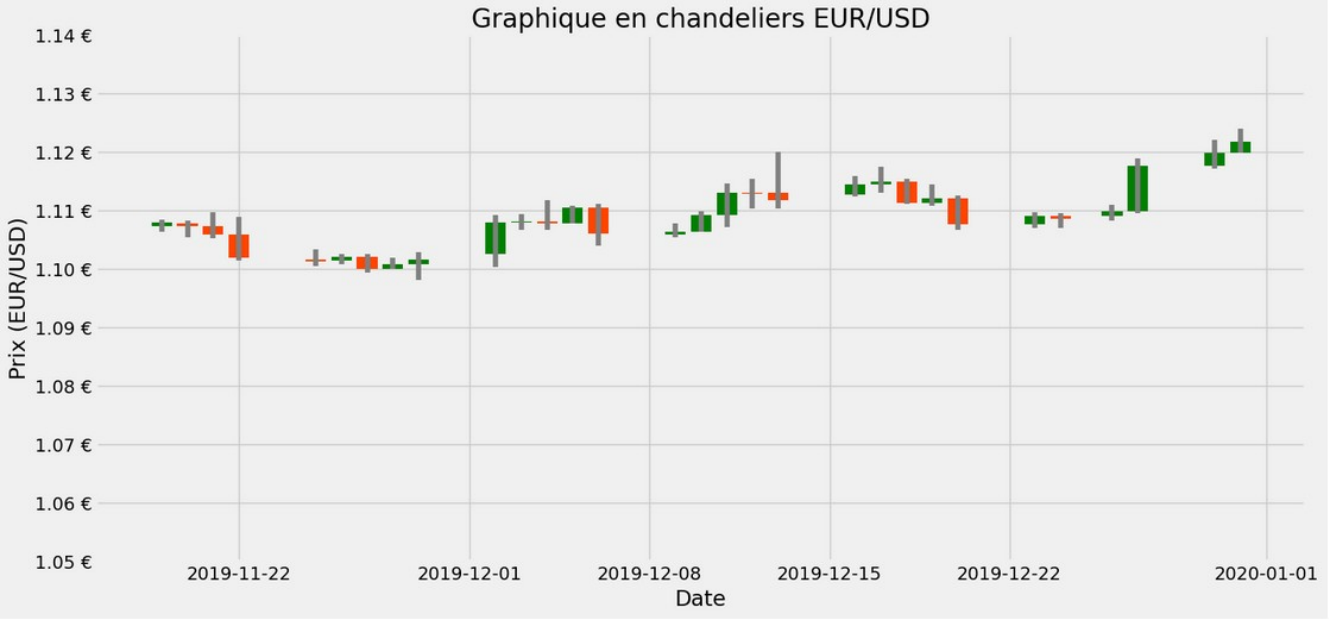
```
# Lignes grises pour les valeurs min et max
plt.vlines(x=eurusd_df_small["Date"], ymin=eurusd_df_small["Low"],
ymax=eurusd_df_small["High"], color="grey")

# Bougies vertes (Close > Open)
plt.bar(x=green_df["Date"], height=green_df["Height"], bottom=green_df["Open"],
color="green")

# Bougies rouges (Close < Open)
plt.bar(x=red_df["Date"], height=red_df["Height"], bottom=red_df["Close"],
color="orangered")

# Personnalisation des ticks de l'axe Y avec des valeurs en euros
plt.yticks(np.arange(1.05, 1.15, 0.01), ["{:.2f} €".format(v) for v in
np.arange(1.05, 1.15, 0.01)])

# Titres et étiquettes
plt.xlabel("Date")
plt.ylabel("Prix (EUR/USD)")
plt.title("Graphique en chandeliers EUR/USD")

# Affichage du graphique
plt.show()
```



Nous créons un graphique en chandeliers en utilisant Matplotlib. Les lignes verticales grises représentent les prix minimums et maximums, tandis que les barres vertes et rouges représentent respectivement les bougies haussières et baissières. Nous personnalisons également l'axe des ordonnées pour afficher les prix en euros.

OR :

```
plt.style.use("fivethirtyeight");
import matplotlib.pyplot as plt
import numpy as np

# Créer la figure pour afficher le graphique
fig = plt.figure(figsize=(15, 7))

## Lignes grises pour les valeurs min et max
plt.vlines(x=eurusd_df_small["Date"], ymin=eurusd_df_small["Low"],
ymax=eurusd_df_small["High"],
        color="grey")

## Bougies vertes (Close > Open)
plt.bar(x=green_df["Date"], height=green_df["Height"], bottom=green_df["Open"], color="green")
## Bougies rouges (Close < Open)
plt.bar(x=red_df["Date"], height=red_df["Height"], bottom=red_df["Close"], color="orangered")
# Personnalisation des ticks de l'axe Y avec des valeurs en EUR/USD
plt.yticks(np.arange(1.05, 1.15, 0.01), ["{:.2f} €".format(v) for v in np.arange(1.05, 1.15, 0.01)])
# Titres et étiquettes
plt.xlabel("Date")
plt.ylabel("Prix (EUR/USD)")
plt.title("Graphique en chandeliers EUR/USD")
# Affichage du graphique
plt.show()
```



OR :

```
import matplotlib.pyplot as plt
import numpy as np

# Créer la figure pour afficher le graphique
fig = plt.figure(figsize=(15, 7))

## Lignes grises pour les valeurs min et max
plt.vlines(x=green_df["Date"], ymin=green_df["Low"], ymax=green_df["High"],
        color="green")
plt.vlines(x=red_df["Date"], ymin=red_df["Low"], ymax=red_df["High"],
        color="orangered")
## Bougies vertes (Close > Open)
plt.bar(x=green_df["Date"], height=green_df["Height"], bottom=green_df["Open"], color="green")
## Bougies rouges (Close < Open)
plt.bar(x=red_df["Date"], height=red_df["Height"], bottom=red_df["Close"], color="orangered")

# Personnalisation des ticks de l'axe Y avec des valeurs en EUR/USD
```

plt.yticks(np.arange(1.05, 1.15, 0.01), ["{:.2f} €".format(v) for v in np.arange(1.05, 1.15, 0.01)])

Titres et étiquettes

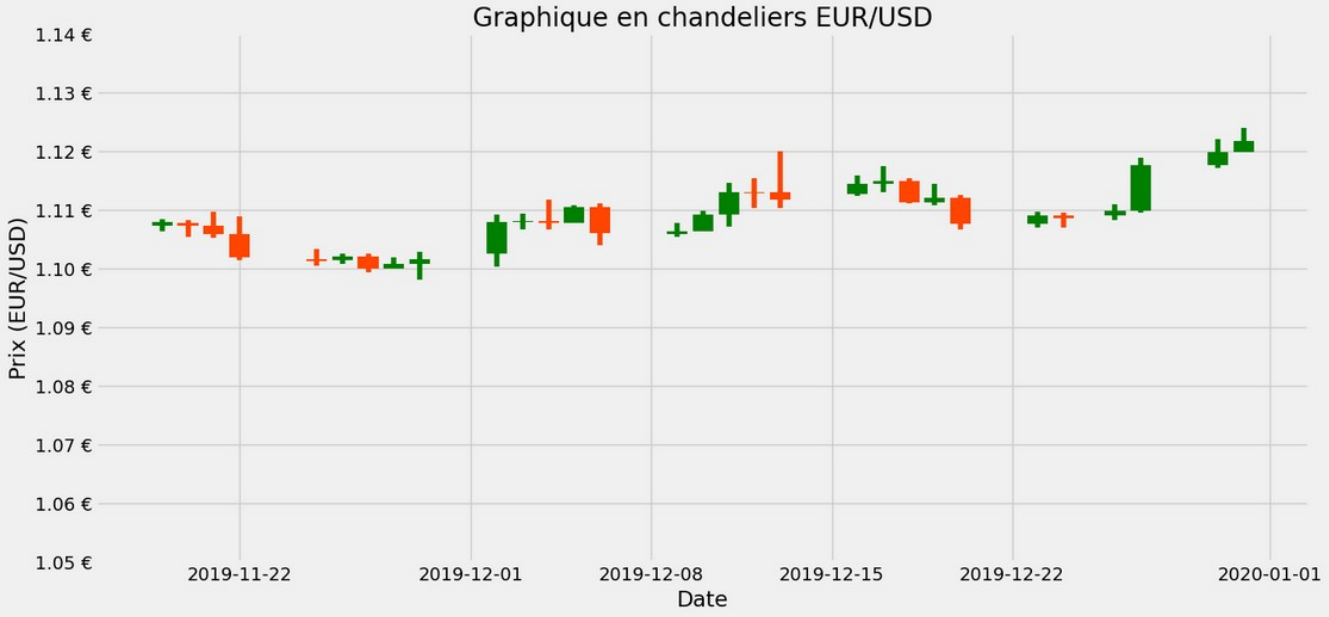
plt.xlabel("Date")

plt.ylabel("Prix (EUR/USD)")

plt.title("Graphique en chandeliers EUR/USD")

Affichage du graphique

plt.show()



6. Affichage des volumes (optionnel)

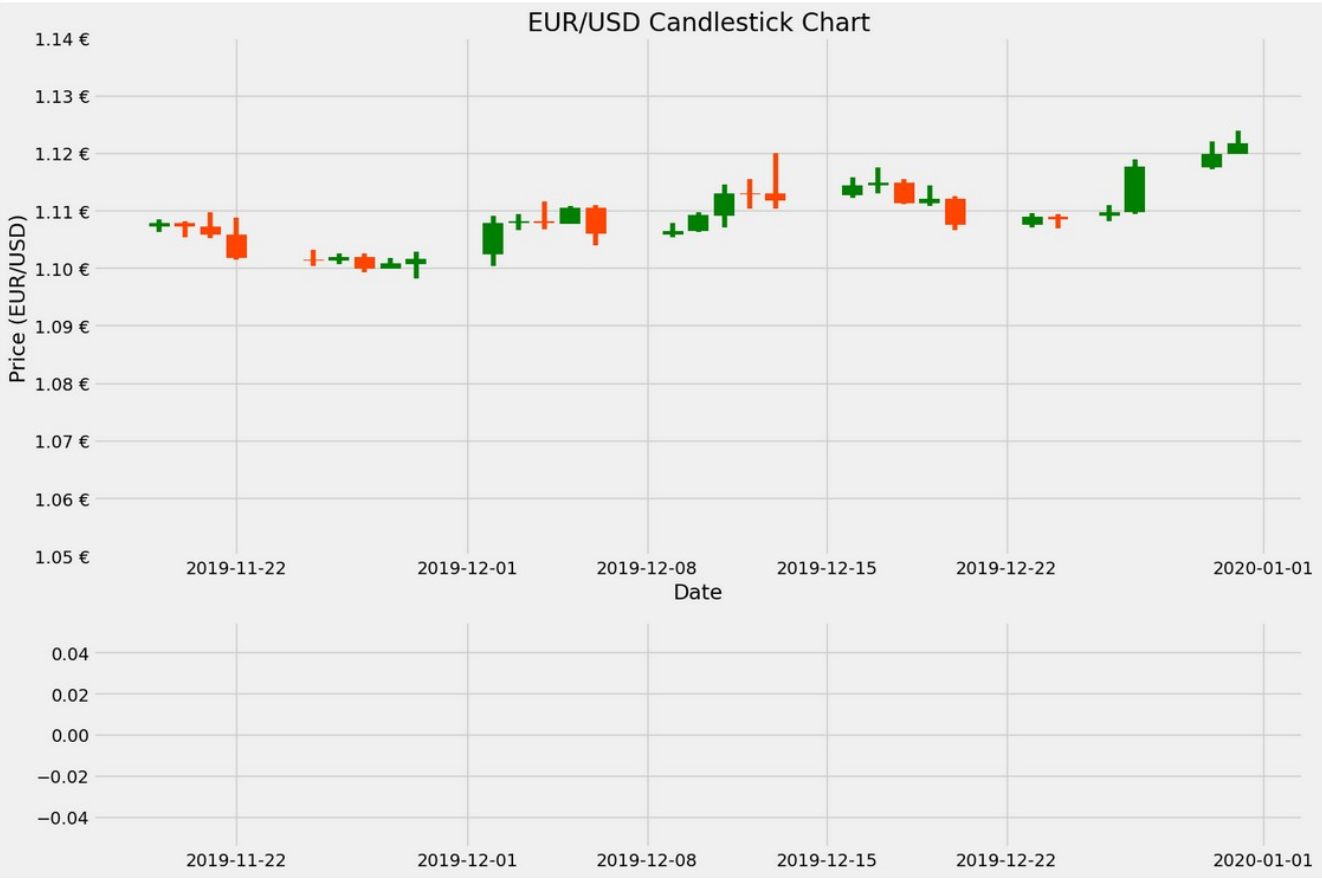
```
# Affichage des volumes
ax2 = fig.add_subplot(grid[2, :])
plt.bar(x=eurusd_df_small["Date"], height=eurusd_df_small["Vol"],
color="dodgerblue")
```

Nous pouvons également afficher les volumes de transactions sous le graphique en chandeliers pour une analyse plus approfondie. Cela peut aider à comprendre la force des mouvements de prix.

Avec Volume Code :

```
import matplotlib.pyplot as plt
```

```
from matplotlib.gridspec import GridSpec

import numpy as np



grid = GridSpec(3,3)

fig = plt.figure(figsize=(15,10))

ax1 = fig.add_subplot(grid[:2, :])

## Lignes grises pour les valeurs min et max

plt.vlines(x=green_df["Date"], ymin=green_df["Low"], ymax=green_df["High"],

           color="green")

plt.vlines(x=red_df["Date"], ymin=red_df["Low"], ymax=red_df["High"],

           color="orangered")

## Bougies vertes (Close > Open)

plt.bar(x=green_df["Date"], height=green_df["Height"], bottom=green_df["Open"],
color="green")

## Bougies rouges (Close < Open)

plt.bar(x=red_df["Date"], height=red_df["Height"], bottom=red_df["Close"],
color="orangered")

# Personnalisation des ticks de l'axe Y avec des valeurs en EUR/USD

plt.yticks(np.arange(1.05, 1.15, 0.01), ["{:.2f} €".format(v) for v in
np.arange(1.05, 1.15, 0.01)])

plt.xlabel("Date")

plt.ylabel("Price (EUR/USD)")

plt.title("EUR/USD Candlestick Chart")

ax2 = fig.add_subplot(grid[2, :])

# Affichage des volumes

plt.bar(x=eurusd_df_small["Date"], height=eurusd_df_small["Vol"],
color="dodgerblue")



plt.tight_layout()
```

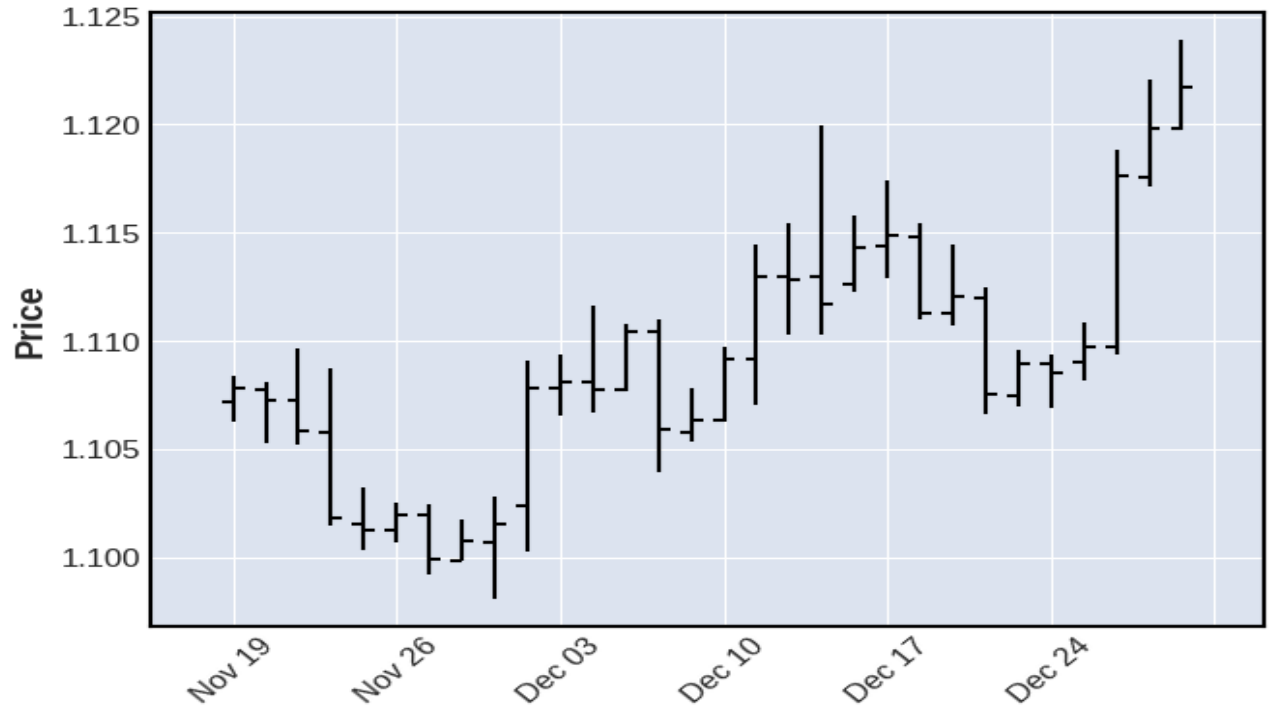


Le Volume n’est pas existe !

Plots par mplfinance :

```
import mplfinance
```

```
mplfinance.plot(eurusd_df_small.set_index("Date"))
```




```
mplfinance.plot(eurusd_df_small.set_index("Date"), type="candle")
```

```
mplfinance.plot(eurusd_df_small.set_index("Date"), type="candle", style="charles",
figscale=1.8)
```

7. Recherche de motifs spécifiques

```
# Engulfing pattern signals
def Revsignal1(df):
    length = len(df)
    high = list(df['High'])
    low = list(df['Low'])
    close = list(df['Close'])
    open = list(df['Open'])
    signal = [0] * length
    bodydiff = [0] * length

    for row in range(1, length):
        bodydiff[row] = abs(open[row] - close[row])
        bodydiffmin = 0.0003  # Seuil pour EUR/USD (valeurs typiques plus petites)
        # Détection des signaux "bearish engulfing"
        if (bodydiff[row] > bodydiffmin and bodydiff[row-1] > bodydiffmin and
            open[row-1] < close[row-1] and
            open[row] > close[row] and
            (open[row] - close[row-1]) >= 0 and close[row] < open[row-1]):
            signal[row] = 1  # Bearish engulfing
        # Détection des signaux "bullish engulfing"
        elif (bodydiff[row] > bodydiffmin and bodydiff[row-1] > bodydiffmin and
              open[row-1] > close[row-1] and
              open[row] < close[row] and
              (open[row] - close[row-1]) <= 0 and close[row] > open[row-1]):
            signal[row] = 2  # Bullish engulfing
        else:
            signal[row] = 0  # Aucun signal

    return signal


# Ajout des signaux dans eurusd_df
eurusd_df['Signal'] = Revsignal1(eurusd_df)

# Compte des signaux détectés
bearish_count = eurusd_df[eurusd_df['Signal'] == 1].count()
bullish_count = eurusd_df[eurusd_df['Signal'] == 2].count()

print("Nombre de signaux Bearish Engulfing :", bearish_count)
print("Nombre de signaux Bullish Engulfing :", bullish_count)
```

```python
# Compte des signaux détectés
bearish_count = eurusd_df[eurusd_df['Signal'] == 1].count()
bullish_count = eurusd_df[eurusd_df['Signal'] == 2].count()

print("Nombre de signaux Bearish Engulfing :", bearish_count)
print("Nombre de signaux Bullish Engulfing :", bullish_count)
```

```
Nombre de signaux Bearish Engulfing : Date       327
Open       327
High       327
Low        327
Close      327
Tickvol    327
Vol        327
Spread     327
Signal     327
dtype: int64
Nombre de signaux Bullish Engulfing : Date       352
Open       352
High       352
Low        352
Close      352
Tickvol    352
Vol        352
Spread     352
Signal     352
dtype: int64
```

Nous définissons une fonction pour détecter les motifs d'engulfing et ajoutons ces signaux au DataFrame. Ensuite, nous comptons le nombre de signaux détectés pour les motifs haussiers et baissiers.

Conclusion

La création de graphiques en chandeliers à l'aide de Python et Matplotlib est une compétence précieuse pour tout trader. Ces graphiques offrent une visualisation claire des mouvements de prix, facilitant l'identification des tendances et des motifs. En combinant cette visualisation avec des analyses de motifs tels que les engulfing patterns, les traders peuvent prendre des décisions éclairées basées sur des données visuelles. Ainsi, la qualité des graphiques et des analyses visuelles joue un rôle crucial dans le succès du trading, renforçant l'importance de maîtriser ces outils pour naviguer efficacement sur les marchés financiers.

Optimisation des Stratégies de Trading par la Visualisation, la Détection de Motifs et l'Intégration Technologique

Dans le domaine du trading, la capacité à visualiser les données financières et à détecter des motifs dans les graphiques en chandeliers est essentielle pour prendre des décisions éclairées. En combinant cette visualisation avec une approche algorithmique pour détecter des motifs, nous pouvons minimiser la probabilité de pertes et maximiser celle des gains. Chaque motif a un pourcentage de réussite associé, et en le confirmant avec des indicateurs techniques tels que les moyennes mobiles (SMA, EMA), nous pouvons renforcer notre stratégie de trading.

1. Lien entre Visualisation, Détection de Motifs et Indicateurs Techniques

La visualisation des données financières permet aux traders d'identifier rapidement des motifs tels que les engulfing patterns, qui signalent souvent des retournements de tendance. En intégrant des indicateurs techniques, nous pouvons valider ces signaux. Par exemple, si un motif haussier est détecté, nous pouvons vérifier si la moyenne mobile exponentielle (EMA) est également en tendance haussière, ce qui renforce la validité du signal.

2. Diversification des Stratégies

Chaque chandelier ou motif identifié peut nécessiter une stratégie de trading distincte. Par exemple, un motif de chandelier haussier peut être associé à une stratégie de suivi de tendance, tandis qu'un motif baissier pourrait nécessiter une approche de couverture. Il est donc crucial de diversifier nos stratégies en fonction des motifs détectés afin d'optimiser les performances globales du portefeuille.

3. Gestion des Risques avec des Stop Loss Dynamiques

La gestion des risques est une composante cruciale de toute stratégie de trading. Les stop loss sont utilisés pour limiter les pertes potentielles. Il existe deux types de stop loss : statique et dynamique.

Stop Loss Statique

Un stop loss statique est fixé à un niveau prédéterminé et ne change pas, peu importe les mouvements du marché. Par exemple, un trader peut décider de placer un stop loss à 30 pips en dessous du prix d'entrée.

```
stop_loss_static = entry_price - 0.0030  # 30 pips pour EUR/USD
```

Stop Loss Dynamique

Un stop loss dynamique, en revanche, s'ajuste en fonction des mouvements du marché. Cela permet de protéger les gains lorsque le marché évolue en notre faveur. Par exemple, si le prix monte de 20 pips, le stop loss peut être déplacé à 10 pips en dessous du nouveau prix.

Exemple de Stop Loss Dynamique Efficace

Voici un exemple simple de code illustrant un stop loss dynamique :

```
entry_price = 1.1000  # Prix d'entrée
stop_loss = entry_price - 0.0030  # Stop loss initial à 30 pips en dessous
trailing_distance = 0.0010  # Trailing stop de 10 pips

# Simulons un mouvement de prix
for current_price in [1.1005, 1.1010, 1.1015, 1.1020, 1.1015, 1.1010]:
    print(f"Prix actuel: {current_price:.4f}, Stop Loss: {stop_loss:.4f}")

    # Mise à jour du stop loss dynamique
    if current_price > entry_price + trailing_distance:
        stop_loss = current_price - trailing_distance  # Ajustement du stop loss
        print(f"Stop Loss ajusté à: {stop_loss:.4f}")
```

Dans cet exemple, si le prix d'entrée est de 1.1000 et que le prix actuel atteint 1.1010, le stop loss est ajusté à 1.1000 (10 pips en dessous du prix actuel), protégeant ainsi les gains.

4. Importance du Backtesting

Le backtesting est une étape cruciale dans le développement de toute stratégie de trading. Il permet aux traders de tester leurs stratégies sur des données historiques pour évaluer leur performance avant de les appliquer sur le marché en temps réel. Grâce au backtesting, les traders peuvent identifier les points forts et les faiblesses de leurs stratégies, ajuster leurs paramètres et s'assurer que leurs approches sont robustes face aux différentes conditions de marché.

5. Intégration de l'Intelligence Artificielle et de Technologies Avancées

L'intégration de l'intelligence artificielle (IA) dans les stratégies de trading ouvre de nouvelles perspectives. Les algorithmes d'apprentissage automatique peuvent analyser des volumes massifs de données pour identifier des motifs complexes et affiner les prédictions. De plus, des techniques avancées telles que les ondelettes (wavelets) peuvent être utilisées pour décomposer les séries temporelles et extraire des caractéristiques pertinentes, améliorant ainsi la précision des modèles prédictifs.

Les circuits quantiques, bien que encore en phase expérimentale, promettent de révolutionner le trading algorithmique en permettant des calculs beaucoup plus rapides et en gérant des ensembles de données bien plus volumineux que les ordinateurs classiques.

Conclusion Globale

En combinant une visualisation claire des données, la détection algorithmique de motifs, l'intégration d'indicateurs techniques, et une gestion des risques efficace grâce à des stop loss dynamiques, nous pouvons créer une stratégie de trading robuste et performante. La diversification des stratégies en fonction des motifs détectés et l'importance du backtesting garantissent que nous prenons des décisions éclairées basées sur des analyses rigoureuses.

L'intégration de technologies avancées telles que l'intelligence artificielle, les ondelettes, et potentiellement les circuits quantiques, représente l'avenir du trading. Ces outils permettent non seulement d'optimiser les performances des stratégies de trading, mais aussi de mieux gérer les risques, augmentant ainsi les chances de succès sur les marchés financiers. En adoptant cette approche systématique et technologique, les traders peuvent naviguer efficacement dans un environnement de marché complexe et en constante évolution.