Make Us Rich

Outil de prédiction de l’évolution des cours des crypto-monnaies

Thomas Chaigneau

May 12, 2022

Contents

1 Introduction **5**
1.1 Contexte . . . 5
1.2 Enoncé du problème . . . 5
1.3 Objectifs . . . 5
1.4 Approche de la solution . . . 5
1.5 Contributions et réalisations . . . 6
1.6 Organisation du rapport . . . 7

2 Etat de l'art **8**
2.1 Aperçu et historique rapide . . . 8
2.2 Fondamentaux des séries temporelles . . . 9
2.3 Différents modèles de prédiction . . . 10
2.4 Réseaux de neurones récurrents (*RNN*) . . . 12
2.4.1 Architecture du réseau de neurones récurrents . . . 12
2.4.2 Avantages et inconvénients . . . 13
2.4.3 Gestion des gradients . . . 14
2.4.3.1 Cas de gradient qui explose . . . 14
2.4.3.2 Cas de gradient qui disparaît . . . 15
2.4.4 Fonctions d'activation . . . 16
2.4.5 GRU et LSTM . . . 16
2.4.5.1 Gated Recurrent Unit (GRU) . . . 17
2.4.5.2 Long Short-Term Memory (LSTM) . . . 17
2.5 L'avènement des modèles Transformers . . . 18

3 Make Us Rich **20**
3.1 Les données . . . 20
3.1.1 Récupération des données . . . 20
3.1.2 Description des données . . . 21
3.1.3 Préparation des données . . . 22
3.1.3.1 Extraction des données utiles . . . 22
3.1.3.2 Séparation des jeux de données . . . 22
3.1.3.3 Mise à l'échelle des données . . . 22
3.1.3.4 Préparation des séquences de données . . . 23
3.2 Modélisation . . . 23
3.3 Service des modèles . . . 23
3.4 Interface utilisateur . . . 24

3.5 Packaging du projet 24

4 Conclusion **25**

Je, Thomas Chaigneau, de l'école Microsoft IA by Simplon à Brest, en alternance en tant que Développeur IA au Crédit Mutuel Arkéa, confirme que c'est mon propre travail et les figures, les tableaux, les extraits de code et les illustrations dans ce rapport sont originaux et n'ont pas ́eté tirés de l'œuvre d'aucune autre personne, sauf lorsque les œuvres d'autres ont ́eté explicitement reconnues, citées et référencées. Je comprends que cela sera considéré comme un cas de plagiat, sinon. Le plagiat est une forme d'inconduite académique et sera pénalisé en conséquence.

Je donne mon consentement a ce qu'une copie de mon rapport soit communiquée aux futurs étudiants comme exemple.

Je donne mon consentement pour que mon travail soit rendu plus largement accessible au public avec un intérêt pour l'enseignement, l'apprentissage et la recherche.

Thomas Chaigneau, April 22, 2022

Tuteur : *Jean-Marie Prigent, Machine Learning Engineer @ Crédit Mutuel Arkéa*

Un rapport soumis en réponse aux exigences de la formation *Développeur IA* de l'école *Microsoft IA by Simplon à Brest* pour le titre RNCP34757.

1 Introduction

1.1 Contexte

Aujourd'hui, il existe plusieurs solutions de surveillance de portefeuilles financiers en ligne, qui permettent le suivi de l'évolution du cours de différents actifs. Dans le cas des crypto-monnaies, ces outils offrent à l'utilisateur une interface gratuite et simple lui permettant d'ajouter manuellement ses actifs et de les suivre en temps réel. Aucun, à ma connaissance, n'intègre des outils d'analyses et de prédictions des cours des crypto-monnaies.

1.2 Enoncé du problème

Puisqu'aucun outil, disponible gratuitement, n'offre la possibilité de simuler une prédiction de l'évolution des cours des crypto-monnaies, je souhaite créer un outil permettant de façon automatisée et transparente pour l'utilisateur de suivre l'évolution des cours des crypto-monnaies. Comment permettre à l'utilisateur de disposer d'informations de prédictions automatiques sur ses actifs qui soient un minimum fiables ?

1.3 Objectifs

Développer une architecture permettant l'entraînement automatique de modèles de prédiction de l'évolution des cours des crypto-monnaies, le service de ces modèles via une *API REST* ainsi qu'une interface web permettant de visualiser les prédictions et l'évolution des cours des crypto-monnaies par l'utilisateur.

1.4 Approche de la solution

Le projet se décompose en trois composants distincts et qui interagissent entre eux pour former la solution complète :

- **Interface** : l'interface web permettant à l'utilisateur de visualiser les prédictions et l'évolution des cours des crypto-monnaies.
- **Serving** : le serveur web qui met à disposition les modèles de prédiction via une API REST.
- **Training** : pipeline automatisé d'entraînement des modèles, de leur validation et de leur stockage.

Voici un schéma de l'architecture du projet :

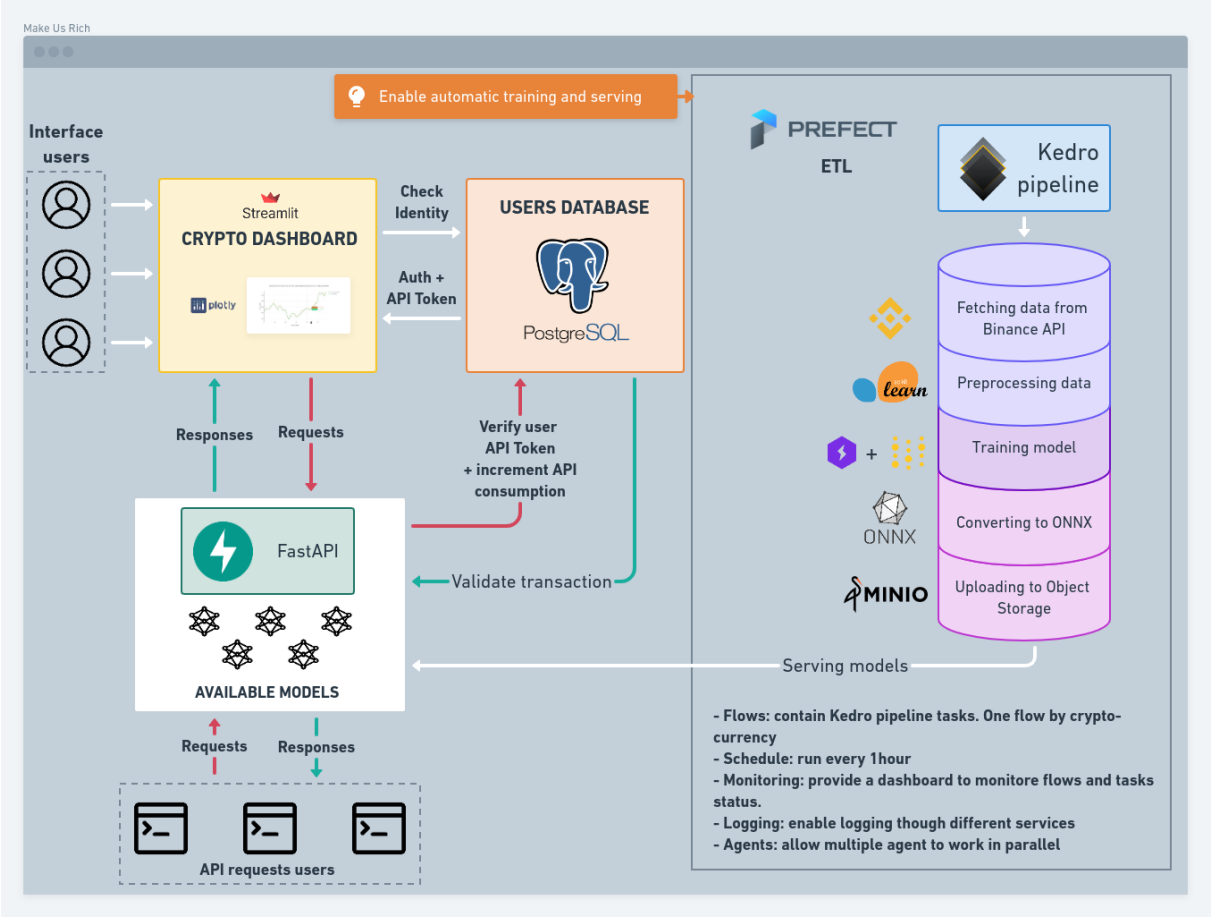


Figure 1: Architecture du projet Make Us Rich

Ce schéma très complet reprend tous les composants du projet et présente leurs interactions. Nous détaillerons les différents composants et leurs fonctions dans la seconde partie de ce rapport.

1.5 Contributions et réalisations

Tout ce qui est présenté dans le schéma d'architecture ci-dessus est fonctionnel et déployé. Les composants sont développés en utilisant le langage de programmation Python et différents outils et librairies très utiles comme *FastAPI*, *Pytorch-Lightning*, *Scikit-learn*, etc.

Le projet est open-source et est accessible sur *GitHub* : Make Us Rich.

Il est possible d'y contribuer et de l'améliorer. Il est également possible de simplement l'utiliser

et de déployer localement tous les composants du projet. Toutes les étapes de déploiement sont détaillées dans la documentation associée qui est également disponible sur *GitHub* : Documentation.

1.6 Organisation du rapport

Le rapport s'organise en 3 grandes parties :

- **Etat de l'art** : présentation des avancées des modélisations IA et des algorithmes de prédiction sur des séries temporelles.
- **Make Us Rich** : présentation et détails de la solution et de son architecture.
- **Conclusion** : retour sur le projet et ouverture sur les différents axes d'amélioration du projet.

2 Etat de l'art

Dans cette première section, nous allons décrire les avancées de la modélisation IA et des algorithmes de prédiction sur des données temporelles. Ce ne sera malheureusement pas une liste exhaustive de toutes les options disponibles, ni un historique complet des différentes évolution des algorithmes de prédiction sur des séries temporelles, car cela est trop riche pour tenir dans ce rapport.

Nous allons donc nous concentrer sur les avancées les plus récentes et celles qui ont un rapport direct avec la solution envisagée dans ce projet. Commençons par un bref aperçu et historique de la tâche de prédiction à l'aide de données temporelles.

2.1 Aperçu et historique rapide

Les séries temporelles, ainsi que leur analyse, sont de plus en plus importantes en raison de la production massive de données dans le monde. Il y a donc un besoin, en constante augmentation, dans l'analyse de ces séries chronologiques avec des techniques statistiques et plus récemment d'apprentissage automatique.

L'analyse des séries chronologiques consiste à extraire des informations récapitulatives et statistiques significatives à partir de points classés par ordre chronologique. L'intérêt étant de diagnostiquer le comportement passé pour prédire le comportement futur.

Aucune des techniques ne s'est développée dans le vide ou par intérêt purement théorique. Les innovations dans l'analyse des séries chronologiques résultent de nouvelles méthodes de collecte, d'enregistrement et de visualisation des données (Nielsen 2019).

Il existe énormément de domaine d'application tels que la médecine, la météorologie, l'astronomie ou encore ce qui va nous intéresser ici, les marchés financiers et notamment celui des crypto-monnaies.

Pour revenir à l'aspect historique, les organisations privées et notamment bancaires ont commencé à collecter des données par imitation du gouvernement américain qui collectait des données économiques publiques. Les premiers pionniers de l'analyse des données chronologiques des cours de la bourse ont fait ce travail mathématique à la main, alors que de nos jours ce travail est réalisé avec l'assistance de méthodes analytiques et des algorithmes de machine learning (Nielsen 2019).

Richard Dennis, dans les années 80, a été le premier à développer un algorithme de prédiction des cours de la bourse qui ne comprenait que quelques règles de base, permettant à quiconque les connaissant de prévoir le prix d'une action et d'en retirer des bénéfices par spéculation.

Progressivement, avec l'accumulation de personnes utilisant ces règles, elles sont devenues de plus en plus inefficaces. Il aura donc fallu développer de nouvelles méthodes, notamment statistiques, plus complexes pour toujours mieux prévoir l'évolution des cours des marchés financiers.

C'est ainsi que des méthodes historiques telles que *SARIMA* ou *ARIMA* se sont démocratisées. Elles présentent néanmoins un inconvénient : elles nécessitent des données stationnaires pour fonctionner. De plus, ces techniques statistiques dites historiques ont des résultats médiocres sur le long terme, et ainsi se sont développés d'autres méthodes d'apprentissage automatique utilisant la puissance des réseaux de neurones, comme *RNN* (Recurrent Neural Network) (Rumelhart and Williams 1985).

2.2 Fondamentaux des séries temporelles

Comme évoqué précédemment, la stationnarité d'une série est une propriété essentielle pour l'analyse statistique. Une série chronologique est dite stationnaire si ces propriétés telles que la moyenne, la variance ou la covariance sont constantes au cours du temps. Or, cela n'est pas vrai pour toutes les séries temporelles et notamment les données issues des marchés financiers, qui ne sont stationnaires que sur une période de temps fixée (souvent courte).

Il existe troise composantes qui constituent une série temporelle :

- **Tendance (T = Trend)** : correspond à une augmentation ou à une diminution sur le long terme des données et qui peut assumer une grande variété de modèles. Nous utilisons la tendance pour estimer le niveau, c'est-à-dire la valeur ou la plage typique de valeurs, que la variable doit avoir au cours du temps. On parle de tendance à la hausse ou à la baisse.
- **Saisonnalité (S = Seasonal)** : est l'apparition de schémas de variations cycliques qui se répètent à des taux de fréquence relativement constants.
- **Résidu (R = Remainder)** : correspondent aux fluctuations à court terme qui ne sont ni systématiques ni prévisibles. Au quotidien, des événements imprévus provoquent de telles instabilités. Concrètement, la composante résiduelle est ce qui reste après l'estimation de la tendance et de la saisonnalité, et leur suppression d'une série chronologique.

Voici une représentation de la décomposition des composantes d'une série temporelle :

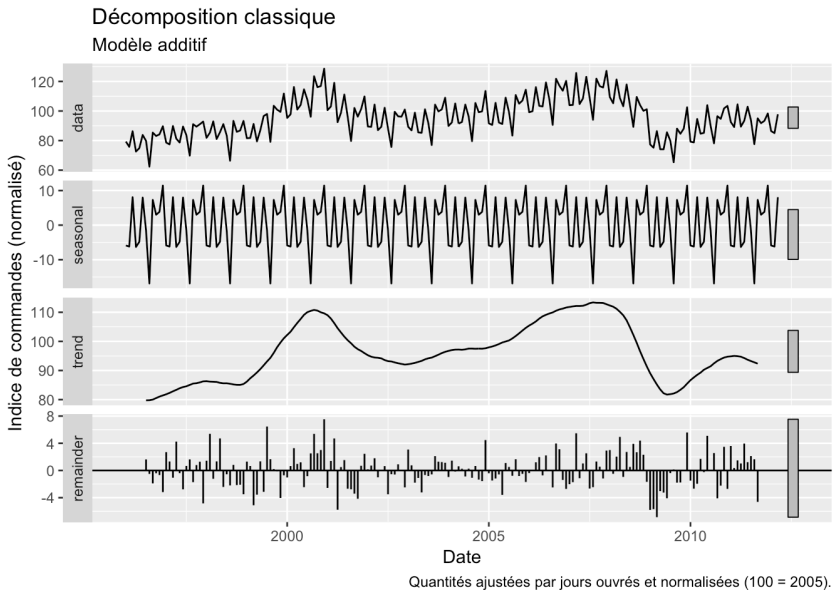


Figure 2: Graphique présentant la décomposition classique d'un modèle additif d'une série temporelle (Chailan and Palacios-Rodríguez 2018)

2.3 Différents modèles de prédiction

Nous allons voir ici les différentes techniques et modèles qui existent pour la prédiction à l'aide de données temporelles.

Nous pouvons d'ores et déjà distinguer deux catégories de modèles de prédiction :

- **Modèles statistiques** : dits traditionnels, ils regroupent les modèles univariés et multivariés comprenant respectivement *ARIMA*, *SARIMA* et *VAR*.

Un processus stationnaire X_t admet une représentation $ARIMA(p, d, q)$ dite minimale s'il existe une relation (Goude 2020) :

$$\Phi(L)(1-L)^d X_t = \Theta(L)\epsilon_t, \forall_t \in Z$$

avec pour conditions :

- $\phi_p \neq 0$ et $\theta_q \neq 0$
- Φ et Θ doivent être des polynômes de degrés respectifs p et q, n'ont pas de racines communes et leurs racines sont de modules > 1
- ϵ_t est un BB de variance σ^2

De même, un processus stationnaire X_t admet une représentation $SARIMA(p, d, q)$ dite minimale si la relation suivante est vraie (Goude 2020) :

$$(1 - L)^d \Phi_p(L)(1 - L^s)^D \Phi_P(L^s) X_t = \theta_q(L)\theta_Q(Ls)\epsilon_t, \forall_t \in Z$$

avec les mêmes conditions que pour les modèles ARIMA.

- **Modèles d'apprentissage automatique** : ils regroupent les modèles de régression par amplification de gradient et les modèles par apprentissage profond comprenant les réseaux de neurones récurrents et convolutionnels.

Voici une représentation des différents modèles de prédiction :

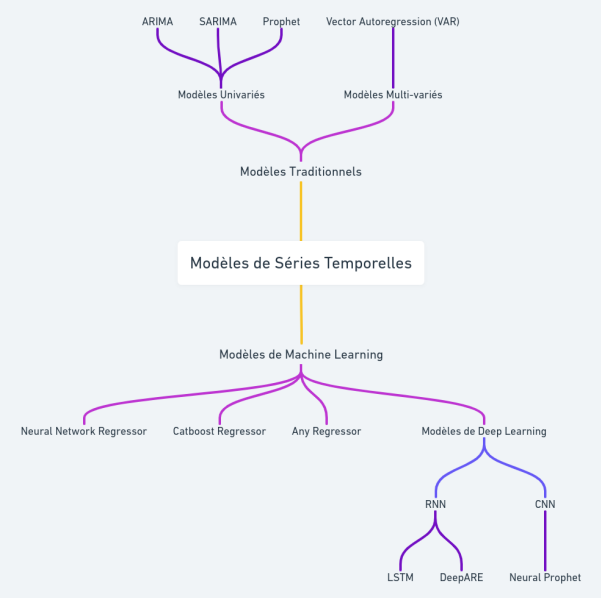


Figure 3: Liste non-exhaustive des modèles utilisés pour la prédiction de séries chronologiques.

Nous pourrions également ajouter à cette liste les très récents modèles basés sur l'architecture *Transformers*, comme **Temporal Fusion Transformers** (TFT) qui est un modèle de Google (Lim et al. 2019) qui permet de combiner des données temporelles avec des données non temporelles, des données statiques comme des informations de localisation dans le cas de prédictions météorologiques (Kafritsas 2021).

Dans notre projet, nous allons nous concentrer sur les modèles de Machine Learning les plus récents, qui sont les modèles de Deep Learning tels que les architectures réseaux de neurones convolutionnels et réseaux de neurones récurrents.

2.4 Réseaux de neurones récurrents (*RNN*)

Les réseaux de neurones récurrents, ou *Recurrent Neural Network*, sont des architectures de neurones qui sont utilisés dans beaucoup de cas d'usage. Ils sont appelés réseaux de neurones récurrents car ils sont capables de se réguler en fonction de la sortie des neurones précédents (Sherstinsky 2020). Ils sont notamment utilisés pour la prédiction de séries temporelles, car ils permettent de prédire la valeur d'une variable à partir de ses valeurs précédentes.

2.4.1 Architecture du réseau de neurones récurrents

C'est par le biais d'états cachés (en anglais *hidden states*) qu'un modèle RNN est capable de réaliser la prédiciton d'une variable. Ainsi, un modèle RNN prend en entrée des séquences de vecteurs de données, et non pas des vecteurs de données individuels.

Une architecture traditionnelle d'un RNN se présente comme suit (Amidi and Amidi 2020) :

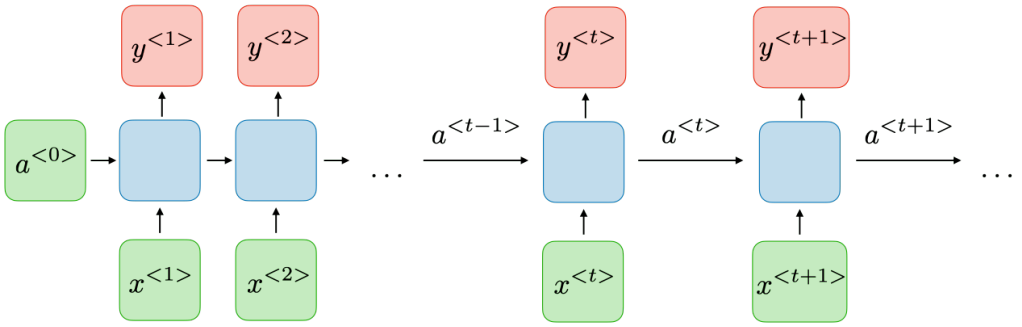


Figure 4: Architecture d'un réseau de neurones récurrents (RNN)

À l'instant *t*, l'activation $a^{<t>}$ d'un neurone est définie par la fonction d'activation suivante :

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

et la sortie $y^{<t>}$ est de la forme :

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

où $W_{ax}, W_{aa}, W_{ya}, b_a$ et b_y sont des coefficients de poids partagés temporellement entre les fonctions d'activation g_1 et g_2.

Il est également intéressant de s'intéresser à l'architecture d'une cellule (en anglais *block*) qui compose un réseau de neurones récurrents. Cela permet de comprendre les mécanismes qui

ont lieu à chaque étape de la propagation de données dans un réseau RNN. Ce sera également utile dans un second temps pour pouvoir comparer les différences majeures avec des architectures plus intéressantes que celles dites classiques, que nous verrons juste après.

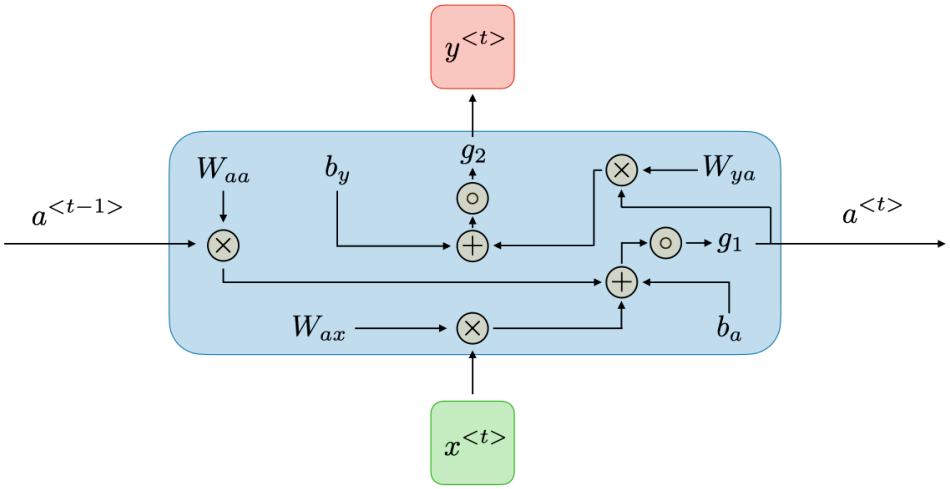


Figure 5: Architecture d'une cellule d'un réseau de neurones récurrents (RNN) (Amidi and Amidi 2020)

Nous pouvons voir que chaque cellule va prendre un état de la donnée précédente, et qu'elle va produire une sortie en fonction de son état et de la fonction d'activation associée.

2.4.2 Avantages et inconvénients

Voici un tableau récapitulatif des avantages et inconvénients d'utiliser ce genre de modélisation (Amidi and Amidi 2020) :

Table 1: Avantages et inconvénients des modèles RNN

Avantages	Inconvénients
- Possibilité de traiter une entrée de n'importe quelle longueur	- Le calcul est plus consommateur en ressources (par rapport à d'autres modèles)
- La taille du modèle n'augmente pas avec la taille de l'entrée	- Difficulté pour accéder aux informations trop lointaines
- Le calcul prend en compte les informations historiques	- Impossibilité d'envisager une entrée future pour l'état actuel

Avantages	Inconvénients
- Les pondérations sont réparties dans le temps	

Nous pouvons constater que comme attendu lors de l'utilisation de modèles de *Deep Learning*, les modèles RNN sont plus consommateurs en ressources que d'autres modèles de *Machine Learning*. Ils présentent néanmoins des avantages non négligeables, en dehors d'un gain de performances, pour notre cas d'usage dans le cadre de la prédiction de séries temporelles financières.

2.4.3 Gestion des gradients

Il existe également un autre inconvénient des modèles RNN qui sont les phénomènes de gradients qui disparaissent et qui explosent lors de l'apprentissage. En anglais, on parle de *vanishing gradient* et *exploding gradient*.

Cela est expliqué par le fait que sur le long terme il est très difficile de capturer les dépendances à cause du gradient multiplicatif qui peut soit décroître, soit augmenter de manière exponentielle en fonction du nombre de couches du modèle.

2.4.3.1 Cas de gradient qui explose

Pour contrer les phénomènes de gradient qui explose, il est possible d'utiliser une technique de *gradient clipping* (Sherstinsky 2020) (en français *coupure de gradient*) qui permet de limiter le gradient à une valeur fixée. Puisque la valeur du gradient est plafonnée les phénomènes néfastes de gradient sont donc maîtrisés en pratique.

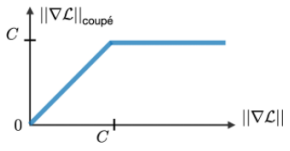


Figure 6: Technique de gradient clipping

Grâce à cette technique, nous pouvons donc éviter que le gradient devienne trop important en le remettant à une échelle plus petite.

2.4.3.2 Cas de gradient qui disparaît

Concernant les phénomènes de gradient qui disparaissent, il est possible d'utiliser des *portes* de différents types, souvent notées Γ et sont définies par :

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

où W, U et b sont des coefficients spécifiques à la porte et σ est une fonction sigmoïde.

Les portes sont utilisées dans les architectures plus spécifiques comme *GRU* et *LSTM* que nous verrons juste après.

Table 2: Comparaison des différents types de portes et leurs rôles

Type de porte	Rôle	Utilité
Porte d'actualisation Γ_u	Décide si l'état de la cellule doit être mis à jour avec la valeur d'activation en cours	GRU, LSTM
Porte de pertinence Γ_r	Décide si l'état de la cellule antérieure est important ou non	GRU, LSTM
Porte d'oubli Γ_f	Contrôle la quantité d'information qui est conservé ou oublié de la cellule antérieure	LSTM
Porte de sortie Γ_o	Détermine le prochain état caché en contrôlant quelle quantité d'information est libérée par la cellule	LSTM

Ces différents types de portes permettent de corriger les erreurs de calcul du gradient en fonction de la mesure de l'importance du passé, et ainsi de s'affranchir en partie des phénomènes de gradient qui disparaissent. Il est important de noter que les portes d'oubli et de sortie sont utilisées uniquement dans les architectures LSTM. GRU dispose donc de deux portes, alors que LSTM dispose de quatre portes.

2.4.4 Fonctions d'activation

Il existe trois fonctions d'activation qui sont utilisées dans les modèles RNN :

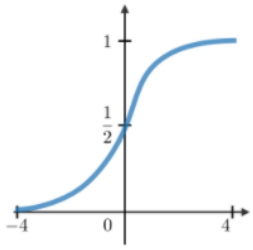
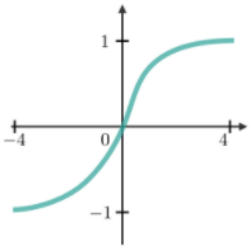
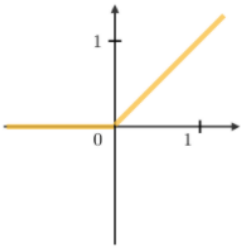
Sigmoïde	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

Figure 7: Fonctions d'activation communément utilisées et leurs représentations (Amidi and Amidi 2020)

- La *fonction d'activation sigmoïde* représente la fonction de répartition de la loi logistique, souvent utilisée dans les réseaux de neurones, car elle est dérivable.
- La *fonction d'activation tanh*, ou *tangente hyperbolique*, représente la fonction de répartition de la loi hyperbolique.
- La *fonction d'activation RELU*, ou *rectified linear unit,* représente la fonction de répartition de la loi linéaire.

Le rôle d'une fonction d'activation est de modifier de manière non-linéaire les valeurs de sortie des neurones, ce qui permet de modifier spatialement leur représentation. Une fonction d'activation est donc définie et spécifique pour chaque couche du réseau de neurones. Il ne faut pas confondre avec les fonctions de *loss* qui sont utilisées pour déterminer la qualité de l'apprentissage et sont quant à elles uniques, c'est-à-dire que l'on doit définir une unique fonction de loss pour chaque modèle.

2.4.5 GRU et LSTM

Nous pouvons distinguer les unités de porte récurrente (en anglais *Gated Recurrent Unit*) (GRU) et les unités de mémoire à long/court terme (en anglais *Long Short-Term Memory*) (LSTM). Ces deux architectures sont très similaires et visent à atténuer le problème de gradient qui disparaît, rencontré avec les RNNs traditionnels lors de l'apprentissage. *LSTM* peut être vu comme étant une généralisation de *GRU* en utilisant des cellules de mémoire à long ou court terme.

Pour comprendre les différences fondamentales entre les deux architectures, il est nécessaire de regarder en détails les différentes équations utilisées par chacune d'elles.

2.4.5.1 Gated Recurrent Unit (GRU)

Comme nous l'avons vu précédemment, l'architecture GRU comporte deux portes : une porte d'actualisation Γ_u (en anglais *update gate*) et une porte de pertinence Γ_r (en anglais *reset gate*).

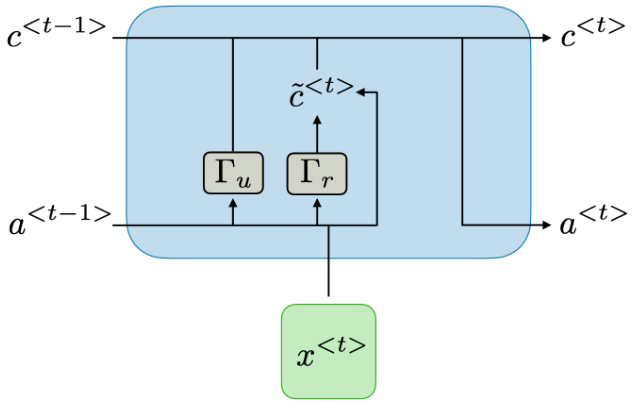


Figure 8: Architecture d'une unité de GRU (Amidi and Amidi 2020)

Il faut discerner trois composantes importantes pour la structure de l'unité de GRU :

- La cellule candidate $c^{<t>}$, où $c^{<t>} = tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
- L'état final de la cellule $c^{<t>}$, où $c^{<t>} = \Gamma_u \star c^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$

  L'état final de la cellule est calculé par la somme des produits de la porte d'actualisation Γ_u et de la valeur de la cellule candidate $c^{<t>}$ et de l'inverse de la porte d'actualisation $1 - \Gamma_u$ multiplié par la valeur de l'état final de la cellule antérieure c^{t-1}.

  Cet état final de la cellule est donc dépendant de la porte d'actualisation Γ_u et peut soit être mis à jour avec la valeur de la cellule candidate $c^{<t>}$ ou soit conservé la valeur de l'état final de la cellule antérieure.
- La fonction d'activation $a^{<t>}$, où $a^{<t>} = c^{<t>}$

2.4.5.2 Long Short-Term Memory (LSTM)

Maintenant que nous avons vu plus en détails l'architecture générale des RNNs, ainsi que les particularités de GRU, il est temps d'aborder en détails les particularités de l'architecture de LSTM, qui sera l'architecture choisie par le projet final.

En plus des deux portes d'actualisation et de pertinence, LSTM intègre une porte d'oubli Γ_f et une porte de sortie Γ_o.

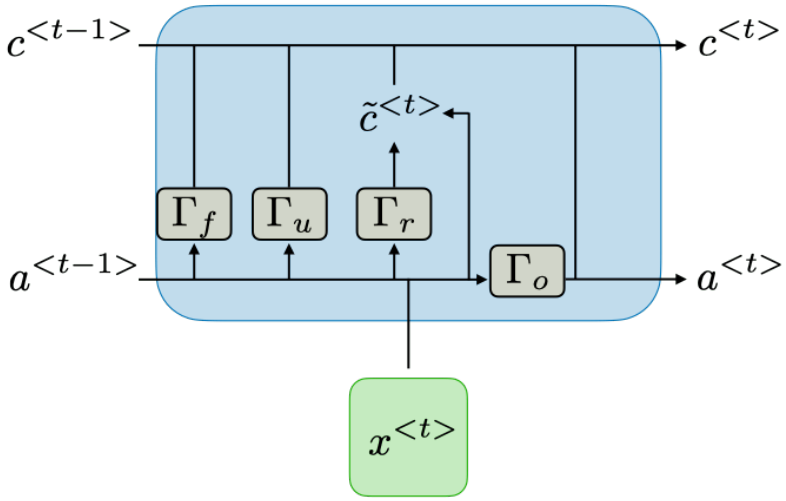


Figure 9: Architecture d'une unité de LSTM (Amidi and Amidi 2020)

L'architecture LSTM est similaire à l'architecture GRU, mais permet de gérer le problème de gradient qui disparaît. Ainsi, aux trois composantes de l'unité de GRU que nous avons vu précédemment, il y a deux différences :

- La fonction d'activation $a^{<t>}$ est désormais multipliée par la porte de sortie Γ_o, ce qui permet de contrôler la quantité d'information qui est libérée par la cellule. On a donc : $a^{<t>} = \Gamma_o \star c^{<t>}$
- L'état final de la cellule $c^{<t>}$ est désormais influencé par la porte d'oubli Γ_f qui devient un facteur de la valeur de l'état final de la cellule antérieure c^{t-1}. En agissant ainsi, la porte d'oubli permet de réguler la quantité d'information retenue de la cellule antérieure. Il y a donc un choix sur ce qui est conservé et oublié. On a ainsi : $c^{<t>} = \Gamma_f \star c^{t-1} + \Gamma_f \star c^{<t-1>}$

2.5 L'avènement des modèles Transformers

Le premier papier de recherche qui présente les modèles Transformers est *Attention Is All You Need* (Vaswani et al. 2017), présenté en juin 2017. L'objectif initial de ce genre d'architecture, qui vient ajouter un mécanisme d'attention aux RNNs, était d'améliorer les performances des modèles existants sur les tâches de traduction en *NLP*.

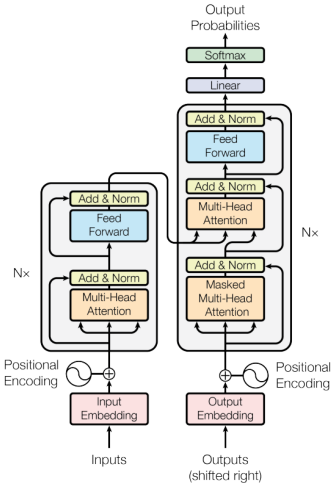


Figure 10: Architecture d'un modèle Transformer (Vaswani et al. 2017)

Nous ne détaillerons pas ici les différentes composantes de cette architecture, mais ce qu'il faut retenir c'est que par l'utilisation d'un encodeur et d'un décodeur, couplés au mécanisme d'attention, il est possible d'entraîner des modèles de manière non-supervisée et surtout d'obtenir des performances encore jamais atteintes par les modèles existants.

L'entraînement de ces modèles de manière non-supervisée est très simple et permet dans un second temps de *finetuner* les modèles avec beaucoup moins de données sur des tâches précises. Ainsi, un même modèle de base peut être entraîné sur différentes tâches et produire des modèles finaux à la pointe des performances des modèles existants.

L'essor de cette architecture a principalement eu lieu dans le domaine du traitement du langage naturel avec des modèles très célèbres tels que *BERT* (Devlin et al. 2018), *GPT-2* (Radford et al. 2019) ou encore *T5* (Raffel et al. 2019). Mais aujourd'hui, il existe également des applications dans les domaines de la vision par ordinateur (*Computer Vision*), des séries temporelles (*Time Series*) ou encore dans le traitement du signal audio (*Audio Processing*).

Cette architecture passionnante est en train de devenir un modèle de référence pour énormément de projet de recherche et cela dans quasiment tous les domaines du Machine Learning. Nous ne détaillerons pas plus ici puisque nous n'avons pas fait usage de cette architecture dans **Make Us Rich**, et cela nécessiterait un dossier à part entière tellement il y a choses à préciser.

Maintenant que nous avons un meilleur aperçu des modèles de référence, et des détails de l'architecture des modèles RNN, et plus particulièrement des modèles LSTM, nous allons pouvoir présenter le projet **Make Us Rich** qui vise à automatiser l'entraînement et le service de modèles LSTM pour de la prédiction sur des données financières.

3 Make Us Rich

Il existe beaucoup de ressources, notamment sur les modèles de prédiction appliqués à des séries temporelles, qui sont très utiles sur des données stationnaires telles que les données météorologiques ou les vols d'avions. En revanche, en ce qui concerne les marchés financiers et les crypto-monnaies, il est difficile de trouver des méthodes dites *classiques* qui soient aussi efficaces et fiables.

Étant passionné par le *Deep Learning* et surveillant d'un oeil positif l'évolution des crypto-monnaies, j'ai souhaité créer un outil permettant de lier les deux. En plus de créer cet outil de prédiction, j'ai voulu aller plus loin et développer le projet pour un faire une base solide à déployer pour quiconque souhaiterait comprendre les enjeux du déploiement de modèles de *Deep Learning* et les outils d'automatisation et de supervision des modèles.

C'est donc avant tout un projet de passioné, mais aussi un formidable outil (en tous cas je l'espère) pour apprendre à automatiser, déployer et mettre à disposition des modèles de *Deep Learning*. Sans plus attendre, je vous invite à découvrir plus en détails le projet et son architecture présentée en introduction de ce dossier.

Pour conserver une certaine lisibilité, les différents blocs de code sont numérotés entre parenthèses et leur détails sont disponibles dans les annexes de ce rapport.

3.1 Les données

TODO : - préparation des données (= feature engineering) - formatage des données (= model inputs)

3.1.1 Récupération des données

Cette étape de récupération des séries temporelles correspond à l'étape de *fetching* sur le schéma présenté en introduction. Les données que j'utilise sont des données publiques collectées par beaucoup de plateformes de marchés financiers. Pour ce projet, j'ai choisi de récupérer les données de la plateforme *Binance*. *Binance* est une plateforme très connue dans le monde de l'échange de crypto-monnaies et elle dispose d'une API de trading qui permet de récupérer des données très simplement et rapidement. Il suffit de disposer d'un compte sur *Binance* et de se générer un token d'accès pour pouvoir utiliser cette API de façon permanente et sans frais.

Binance dispose également d'un package Python (`python-binance`) qui facilite les appels à son API. J'ai donc coder une classe, `BinanceClient`(1), qui permet de gérer les interactions

avec l'API de *Binance* et qui inclut des méthodes telles que la récupération de données sur cinq jours, un an ou une période à définir par l'utilisateur. Ces méthodes requièrent en argument un symbole de crypto- monnaie et une monnaie de comparaison dans tous les cas et renvoient les données sous forme de `pandas.DataFrame`.

3.1.2 Description des données

Les données récupérées sont des séries temporelles de la crypto-monnaie cible comparée à une monnaie. Par exemple, je peux récupérer les données de la crypto-monnaie *BTC* par rapport à la monnaie *EUR* sur les 5 dernières jours avec une intervalle de 1 heure.

On compte 12 colonnes de données :

- `timestamp` : date et heure de la série temporelle.
- `open` : valeur d'ouverture de la crypto-monnaie cible sur l'intervalle.
- `high` : valeur haute de la crypto-monnaie cible sur l'intervalle.
- `low` : valeur basse de la crypto-monnaie cible sur l'intervalle.
- `close` : valeur de clôture de la crypto-monnaie cible sur l'intervalle.
- `volume` : volume d'échange de la crypto-monnaie cible sur l'intervalle.
- `close_time` : date et heure de la clôture de la crypto-monnaie cible sur l'intervalle.
- `quote_av` (quote asset volume) : correspond au volume d'échange de la monnaie cible sur l'intervalle.
- `trades` : correspond au nombre de trades effectués sur l'intervalle.
- `tb_base_av` (taker buy base asset volume) : correspond au volume d'acheteur de la crypto-monnaie cible sur l'intervalle.
- `tb_quote_av` (taker buy quote asset volume) : correspond au volume d'acheteur de la monnaie cible sur l'intervalle.
- `ignore` : correspond à une colonne qui ne sera pas utilisée.

Ce sont des informations classiques que l'on retrouve souvent sur les plateformes de marchés financiers. Nous allons voir que pour notre cas d'usage, toutes ces données ne seront pas utilisées. De plus, ces données ne seront pas stockées puisqu'elles ne sont plus valables après la fin de l'intervalle de récupération et que le projet prévoit un ré-entrainement toutes les heures des modèles. Ainsi nous n'avons pas besoin de stocker les données pour une utilisation ultérieure, un simple appel à l'API suffit pour récupérer les nouvelles données utiles à un entraînement de modèle.

3.1.3 Préparation des données

Il est nécessaire de préparer les données pour qu'elles soient utilisables par le modèle. Pour cela, nous allons utiliser plusieurs fonctions définies dans l'étape de `preprocessing` de ce projet. C'est à cette étape qu'intervient un choix des *features* à utiliser pour l'entraînement du modèle leur *engineering*. Le terme *feature engineering* est un terme désignant les différentes étapes de "rafinement" des données pour qu'elles soient utilisables par le modèle.

3.1.3.1 Extraction des données utiles

Tout d'abord, nous allons extraire les données utiles à partir de la série temporelle grâce à la fonction `extract_features_from_dataset`(2). Pour cela, nous allons utiliser uniquement les colonnes `open`, `high`, `low`, `close` et `timestamp`. Nous stockons également la différence entre la valeur de clôture et la valeur d'ouverture pour chaque intervalle sous le nom `close_change`.

Ainsi à l'issu de cette étape, nous obtenons un nouveau `pandas.DataFrame` qui contient les features spécialement sélectionnées pour l'entraînement. Nous n'incluons pas les colonnes relatives aux volumes d'échange et aux trades, car ce qui nous intéressera sera la prédiction de la valeur de clôture sur l'intervalle suivant. Il serait néanmoins possible d'inclure les notions de volumes dans l'entraînement, mais cela complexifierait le modèle et l'alourdirait pour un gain probablement peu significatif.

3.1.3.2 Séparation des jeux de données

Une fois nos *features* sélectionnées, nous allons séparer les données en trois jeux de données distincts grâce à la fonction `split_data`(3). La séparation consiste à diviser les données en deux jeux de données :

- `training_set` (90%)
- `test_set` (10%)

Pour des données temporelles il est important de ne pas mélanger la chronologie des données puisque cela peut créer des problèmes de cohérence. En effet, nous voulons que le modèle puisse prédire les valeurs de clôture sur l'intervalle suivant et non pas sur des intervalles passés.

3.1.3.3 Mise à l'échelle des données

Il est important de mettre à l'échelle les données pour que le modèle puisse les utiliser correctement. Pour cela, nous allons utiliser la fonction `scale_data`(4). Cette fonction va permettre de

normaliser les données pour que les données de nos deux jeux de données soient comprises entre -1 et 1. C'est une technique de normalisation qui permet de réduire les écarts entre les données et ainsi les rendre plus facile à manipuler par le modèle lors de l'entraînement.

On utilise ici la méthode de normalisation `MinMaxScaler` de la librairie `sklearn`. Il est important de noter que nous sauvegardons également cet objet de normalisation dans un fichier pickle pour pouvoir l'utiliser plus tard lors de l'inférence via l'API. En effet, puisque le modèle est entraîné sur des données normalisées, il est primordial qu'elles le soient également lors des prédictions postérieures. De plus, nous avons besoin de cet objet pour pouvoir inverser la normalisation des données prédites et obtenir des valeurs de clôture réelles, c'est-à-dire des valeurs de clôture non normalisées.

3.1.3.4 Préparation des séquences de données

Il ne reste plus qu'à préparer les données pour qu'elles soient utilisables par le modèle. Pour cela, nous allons devoir créer des séquences de données. Pour cela, nous allons utiliser la fonction `create_sequences`(5). Cette fonction va utiliser les données préalablement normalisées pour créer des séquences de données de taille `sequence_length`.

C'est à cette étape que nous construisons les *features* d'entrée du modèle et la *target* de sortie, aussi appelé *label*. Dans notre cas, nous utiliserons la colonne `close` comme *target* et le reste des colonnes comme *features*.

Il est à noter que nous alons séparer les données du `training_set` en deux séquences de données distinctes pour avoir également des séquences de données pour la validation du modèle, grâce à la fonction `split_sequences`(6). Nous utiliserons comme taille `val_size=0.2` pour la validation du modèle, ce qui représente 18% des données totales attribuées pour la validation du modèle.

3.2 Modélisation

TODO : - présentation du modèle - description du modèle (avec code) - choix des hyperparamètres - monitoring des entraînements - validation du modèle - conversion vers ONNX - stockage objet des modèles + features engineering

3.3 Service des modèles

TODO : - contexte global sur serving de modèles (docker, kubernetes, …) - présentation API REST - détails des endpoints de l'API REST

3.4 Interface utilisateur

TODO : - présentation de l'interface utilisateur - base de données relationnelle pour authentification - tokens pour appels API

3.5 Packaging du projet

TODO : - ETL - Prefect - Déploiement - Docker, Cloud ou Local - Documentation - Mkdocs material - Alerting

4 Conclusion

TODO : - retour sur projet : ce qui a été fait, reste à faire, mieux, moins bien, … - ouverture : monétisation, modélisation, fonctionnalités utilisateurs, …

```python
# https://github.com/ChainYo/make-us-rich/blob/master/make_us_rich/client/binance_client.py
import pandas as pd

from binance.client import Client
from os import getenv
from typing import Optional

from make_us_rich.utils import load_env


class BinanceClient:
    def __init__(self):
        """
        Initializes the client for connecting to the Binance API.
        """
        try:
            self._config = load_env("binance")
        except:
            self._config = {"API_KEY": getenv("API_KEY"), "SECRET_KEY": getenv("SECRET_KEY")}
        self.client = Client(self._config["API_KEY"], self._config["SECRET_KEY"])
        self.columns = ["timestamp", "open", "high", "low", "close", "volume", "close_time",
            "quote_av", "trades", "tb_base_av", "tb_quote_av", "ignore"]


    def get_five_days_data(self, symbol: str) -> pd.DataFrame:
        """
        Gets the data for the last five days.
        Parameters
        ----------
        symbol: str
            Symbol to get the data for.

        Returns
        -------
        pd.DataFrame
            Dataframe for the last five days.
        """
        symbol = symbol.upper()
        klines = self.client.get_historical_klines(symbol, "1h", "5 day ago UTC")
        data = pd.DataFrame(klines, columns=self.columns)
        data["timestamp"] = pd.to_datetime(data["timestamp"], unit="ms")
        return data


    def get_one_year_data(self, symbol: str) -> pd.DataFrame:
        """
        Gets the data for the last year.
        Parameters
        ----------
        symbol: str
            Symbol to get the data for.

        Returns
        -------
        pd.DataFrame
            Dataframe for the last year.
        """
        klines = self.client.get_historical_klines(symbol, "1h", "1 year ago UTC")
        data = pd.DataFrame(klines, columns=self.columns)
        data["timestamp"] = pd.to_datetime(data["timestamp"], unit="ms")
        return data


    def get_data(
      self, symbol: str, interval: str, start_time: str, end_time: Optional[str] = None
    ) -> pd.DataFrame:
        """
        Gets the data for the given symbol, interval, and time range.
        Parameters
        ----------
        symbol: str
            Symbol to get the data for.
        interval: str
            Interval to get the data for.
        start_time: str
            Start time of the data.
        end_time: Optional[str]
            End time of the data.

        Returns
        -------
        pd.DataFrame
            Dataframe for the given symbol, interval, and time range.
        """
        klines = self.client.get_historical_klines(symbol, interval, start_time, end_time)
        data = pd.DataFrame(klines, columns=self.columns)
        data["timestamp"] = pd.to_datetime(data["timestamp"], unit="ms")
        return data
```

Annexe 1 **Annexe 2**

```python
import pandas as pd


def extract_features_from_dataset(data: pd.DataFrame) -> pd.DataFrame:
    """
    Extract features from dataset.

    Parameters
    ----------
    data: pd.DataFrame
        Market chart data.

    Returns
    -------
    pd.DataFrame
        Pandas dataframe of features.
    """
    rows = []
    for _, row in data.iterrows():
        row_data = dict(
            day_of_week=row["timestamp"].dayofweek,
            day_of_month=row["timestamp"].day,
            week_of_year=row["timestamp"].week,
            month_of_year=row["timestamp"].month,
            open=row["open"],
            high=row["high"],
            low=row["low"],
            close=row["close"],
            close_change=row["close"] - row["open"],
        )
        rows.append(row_data)
    return pd.DataFrame(rows)
```

Annexe 3

```python
import pandas as pd


def split_data(data: pd.DataFrame) -> pd.DataFrame:
    """
    Split data into training and test sets.

    Parameters
    ----------
    data: pd.DataFrame
        Market chart data.

    Returns
    -------
    pd.DataFrame
        Pandas dataframe of training and test data.
    """
    train_size = int(len(data) * 0.9)
    train_df, test_df = data[:train_size], data[train_size + 1:]
    return train_df, test_df
```

```python
import pandas as pd

from pickle import dump
from sklearn.preprocessing import MinMaxScaler


def scale_data(
    train_df: pd.DataFrame, test_df: pd.DataFrame, dir_path: str,
) -> pd.DataFrame:
    """
    Scale data to have a mean of 0 and a standard deviation of 1.

    Parameters
    ----------
    train_df: pd.DataFrame
        Training data.
    test_df: pd.DataFrame
        Test data.
    dir_path: str
        Directory path to save the scaler.

    Returns
    -------
    pd.DataFrame
        Scaled training and test data.
    """
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train_df)

    scaled_train_df = pd.DataFrame(
        scaler.transform(train_df),
        index=train_df.index,
        columns=train_df.columns,
    )
    scaled_test_df = pd.DataFrame(
        scaler.transform(test_df),
        index=test_df.index,
        columns=test_df.columns,
    )
    dump(scaler, open(f"{dir_path}/scaler.pkl", "wb"))
    return scaled_train_df, scaled_test_df
```

Annexe 4

```python
import pandas as pd

from typing import List, Tuple


def create_sequences(
    input_data: pd.DataFrame,
    target_column: str,
    sequence_length: int,
    ) -> List[Tuple[pd.DataFrame, float]]:
    """
    Create sequences from the input data.

    Parameters
    ----------
    input_data: pd.DataFrame
        Pandas dataframe of input data.
    target_column: str
        Name of the column to predict.
    sequence_length: int
        Length of the sequence.

    Returns
    -------
    List[Tuple[pd.DataFrame, float]]
        List of sequences.
    """
    sequences = []
    size = len(input_data)
    for i in range(size - sequence_length):
        sequence = input_data[i: i + sequence_length]
        label_position = i + sequence_length
        label = input_data.iloc[label_position][target_column]
        sequences.append([sequence, label])
    return sequences
```

Annexe 5

Annexe 6 Annexe 6 **Annexe 7** Annexe 7 **Annexe 8** Annexe 8 **Annexe 9** Annexe 9 **Annexe 10** Annexe 10

Amidi, Afshine, and Shervine Amidi. 2020. “Recurrent Neural Networks Cheatsheet Star.” *CS 230 - Recurrent Neural Networks Cheatsheet*. Stanford.edu. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

Chailan, Romain, and F. Palacios-Rodríguez. 2018. “TP-Timeseries Novembre 2018.” *Rchailan.github.io*. https://rchailan.github.io/assets/lectures/timeseries/tp_timeseries.html.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. “BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” arXiv. https://doi.org/10.48550/ARXIV.1810.04805.

Goude, Yannig. 2020. “Les Processus Arima.” Paris-Saclay. https://www.imo.universite-paris-saclay.fr/~goude/Materials/time_series/cours6_ARIMA.pdf.

Kafritsas, Nikos. 2021. *Temporal Fusion Transformer: Time Series Forecasting with Interpretability*. https://towardsdatascience.com/temporal-fusion-transformer-googles-model-for-interpretable-time-series-forecasting-5aa17beb621.

Lim, Bryan, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. 2019. “Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting.” arXiv. https://doi.org/10.48550/ARXIV.1912.09363.

Nielsen, Aileen. 2019. *Practical Time Series Analysis*. O’Reilly Media.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. *Language Models Are Unsupervised Multitask Learners*. OpenAI. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.” arXiv. https://doi.org/10.48550/ARXIV.1910.10683.

Rumelhart, Geoffrey E, David E; Hinton, and Ronald J Williams. 1985. “Learning Internal Representations by Error Propagation,” September. https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf.

Sherstinsky, Alex. 2020. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network.” *Physica D: Nonlinear Phenomena* 404 (March): 132306. https://doi.org/10.1016/j.physd.2019.132306.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. “Attention Is All You Need.” arXiv. https://doi.org/10.48550/ARXIV.1706.03762.