Etat de l'art
Séries temporelles

HERMES - DATALAB

Antoine DARGIER

ETE 2023

Ce document a pour vocation d'être un guide introductif à l'étude de séries temporelles. Il couvre un large champ des problèmes rencontrés et la méthodologie globale de ce genre de projet. Il est accompagné de lignes de codes pour pouvoir reproduire rapidement les éléments étudiés, ainsi de quelques exemples pour y voir plus clair. Chaque partie contient un petit résumé pratique, et le document contient des fiches récapitulatives à la fin pour résumer les informations essentielles et pouvoir les retrouver rapidement. Bonne lecture !

Sommaire

1. Introduction

Dans cette partie, nous allons présenter les séries temporelles en en donnant une **définition** pour fixer le cadre de l'étude. Nous verrons ensuite **l'intérêt** de ce type d'analyses et différents **exemples** utiles et appliqués au cas d'Hermès.

a. Définition

Une **série temporelle**, ou série chronologique, est une suite de valeurs numériques représentant **l'évolution d'une quantité spécifique au cours du temps** [1.1]. Chaque observation est associée à un instant précis, ce qui permet d'établir une relation chronologique entre les données. L'analyse des séries temporelles vise à modéliser et/ou à prédire les valeurs futures en fonction des valeurs passées.

Série temporelle : série de données indexée par le temps, quel que soit l'indice de temps (seconde, minute, heure, jour, année, …) et la fréquence des données

b. Intérêt

L'étude des séries temporelles présente plusieurs intérêts majeurs. Tout d'abord, elle permet de **détecter les tendances** et les modèles cachés dans les données, ce qui peut conduire à une meilleure compréhension des phénomènes étudiés. En **identifiant les schémas récurrents, les pics saisonniers ou les fluctuations cycliques**, il devient possible de prendre des **décisions éclairées** et d'anticiper les événements futurs. De plus, l'analyse des séries temporelles joue un rôle crucial dans la **prévision et la planification**. En comprenant les schémas passés, il est possible de faire des prédictions raisonnables sur les valeurs futures. Enfin, les séries temporelles peuvent aussi être utilisées pour des modèles de **classification**, où il peut s'agir par exemple de reconnaître si l'évolution est anormale. Cela peut être utile dans de nombreux domaines, tels que la gestion des stocks, la prévision des ventes, la planification de la production ou même la prévision des tendances économiques.

Applications :

- Compréhension de phénomènes : tendances, saisonnalité, cycles, …
- Régression : prévision et planification
- Classification : détection d'anomalies

c. Exemples

Les séries temporelles sont présentes dans de nombreux domaines, tels que l'économie, la finance, les sciences sociales, la météorologie, la santé et bien d'autres encore. Elles offrent une perspective unique pour comprendre et prédire les phénomènes qui évoluent dans le temps. Voici quelques exemples concrets dans le business de Hermès :

- **Les prévisions de demande** : la demande des entrepôts en matières premières ou des magasins en produits dépend largement du temps, c'est-à-dire des saisons, des jours de la semaine, voire des heures de la journée. Il est donc important de pouvoir les prédire le plus fidèlement possible pour pouvoir anticiper les flux, les besoins en personnels, transports, … ;
- **L'optimisation des prix** : avec une meilleure visibilité et prévision des prix, il devient possible avec les séries temporelles de fixer au mieux les prix des produits dans les différents pays. Cela repose une nouvelle fois sur une meilleure prévision de la demande, et un ajustement des prix en considération ;
- **Les prévisions du trafic** : le trafic est une donnée temporelle typique qui rentre pleinement dans le cadre des séries temporelles. Avec une grande saisonnalité à la journée, à la semaine et avec les jours fériés et vacances, il est intéressant d'avoir des modèles de prédiction du trafic qui pourront ensuite être utilisé pour optimiser les trajets des transporteurs.

2. Cadrage et exploration

Nous allons détailler ici les questions principales à se poser devant une série temporelle, et les premières étapes pour bien comprendre la nature des données. Nous allons commencer par les **questions à se poser** devant une série temporelle avant de se lancer dans le projet. Ensuite, nous verrons les principales étapes d'exploration des données : la **structuration des données, l'analyse de la qualité** des données et **la phase exploratoire**.

a. Les questions à se poser

Dans cette partie, nous allons détailler les grandes questions à se poser au début d'un projet d'analyse de série temporelle, en allant des questions les plus générales aux premières questions de modélisation [2.1].

- **Quelle est la volumétrie des données disponibles ?**
  La taille des données est un facteur très important dans le choix des modèles et les performances. En effet, pour un nombre faible de données (quelques centaines d'observations), les méthodes statistiques fonctionneront très bien, puisque les équations provenant des données ne sont pas très complexes. En revanche, pour de grands volumes de données, les méthodes de machine learning ou deep learning seront probablement plus performantes.

- **Quelle est la maille temporelle des données ?**
  Par maille temporelle, on entend le pas de temps des données. Le pas de temps peut être à la minute, heure, journée, semaine, … Il peut aussi être irrégulier et potentiellement différent entre les données traitées et les prévisions réalisées.
- **Quelle est la maille produit de l'étude ?**
  La maille produit de l'étude est le niveau de précision de la série temporelle : on peut par exemple étudier les ventes au niveau des SKUs, des produits, des maisons, … C'est finalement le niveau d'agrégation des données. Ce niveau peut potentiellement influer sur les choix des modèles ou être remis en question dans l'analyse si les données sont plus ou moins conséquentes.
- **A quelle maille géographique analyse-t-on ces produits ?**
  Maintenant que nous avons fixé la maille temporelle et la maille produit, il faut aussi considérer la maille géographique de l'étude. Est-ce que nous regardons à l'échelle des boutiques, régions, pays, continents, … Une nouvelle fois, cela aura une influence sur la quantité des données dans la modélisation.
- **A quel horizon et quelle fréquence veut-on faire les prévisions ?**
  Il est important de fixer l'horizon de prédiction, car cela aura une influence sur l'évaluation du modèle. Ce ne sont surement pas les mêmes modèles qui seront utilisés pour prédire à deux jours ou à trois mois sur les mêmes données. Et de même, la fréquence de prédiction peut avoir une influence sur le pre-processing et notamment l'agrégation potentielle des données. Il faut donc y répondre rapidement.
- **A-t-on besoin d'un modèle déterministe ou probabiliste ?**
  Cela rejoint en partie le point de l'explicabilité : peut-on se satisfaire d'une unique valeur prédite, ou préfère-t-on un modèle probabiliste qui donne un intervalle de confiance ? Avec les méthodes statistiques, il est souvent possible de renvoyer un intervalle de confiance en plus de la valeur prédite, mais ce n'est pas toujours facile et il faut donc l'anticiper si nécessaire.
- **Y a-t-il des données exogènes ou covariables disponibles ?**
  Face à une série temporelle, il est aussi intéressant de voir quelle autre source d'information pourrait aider à améliorer les prédictions. C'est ce qu'on appelle des données exogènes ou covariables, c'est-à-dire des données extérieures. Il peut y avoir des covariables passées (dont les valeurs passées sont connues), futures (dont les valeurs futures sont connues, par exemple la météo ou des évènements calendaires) ou statiques (séries qui ne changent pas en fonction du temps, par exemple la taille des boutiques)

Données exogènes ou covariables : on parle de données exogènes ou covariables pour décrire des données extérieures supplémentaires utilisées pour améliorer les performances de notre modèle

- **Quelle est l'importance de l'explicabilité pour le cas d'usage ?**
  Il est important de cerner rapidement le besoin d'explicabilité pour le métier pour choisir au mieux le modèle en conséquence. Tous les modèles n'ont pas le même degré d'explicabilité : certains vont renvoyer les features les plus importantes (régression logistique), d'autres seront construits statistiquement avec la tendance et saisonnalité ce qui permet de bien comprendre la prédiction (ARMA), et certains restent des boîtes noires (deep learning).

- **Est-ce que les données sont stationnaires ?**
  C'est une question fondamentale pour l'analyse, puisque la majorité des méthodes d'analyse de séries temporelles repose sur l'hypothèse que les séries sont stationnaires.

Stationnarité : les caractéristiques des données (moyenne, variance, autocorrélation, …) sont constantes en fonction du temps

Il est possible de tester statistiquement si la série est stationnaire ou non, ce qui sera développer dans la partie 3.f. Si elle ne l'est pas, il existe des méthodes pour la rendre stationnaire qui seront aussi développer en 3.f.

- **Y a-t-il une saisonnalité dans les données ?**
  Une saisonnalité oriente aussi dans les choix du modèle. Il est donc important de visualiser la série temporelle pour voir si des motifs se répètent. Des méthodes seront notamment plus performantes pour des données avec une saisonnalité (SARIMA vs. ARIMA). Il faut faire cependant attention à la différence entre saisonnalité et cyclicité.

Saisonnalité : une série temporelle est saisonnière ou périodique si elle est influencée par des facteurs saisonniers (jour, mois, trimestre). C'est toujours pour une période fixée et connue

Cyclicité : une série temporelle est cyclique si elle a des variations à une période non fixée, pour une durée d'au moins 2 ans en général

- **S'agit-il de données univariées ou multivariées ?**
  Il est aussi important d'analyser la typologie des données. Pour les séries temporelles, nous pouvons distinguer les données univariées des données multivariées. Chacun de ces types de données a ses propres modèles, et c'est donc un gain de temps de vite comprendre la typologie de données pour se diriger vers les modèles adéquats.

Données univariées : on parle de données univariées s'il s'agit de l'étude d'une unique série temporelle. On utilisera alors que les valeurs passées pour notre modèle

Données multivariées : des données multivariées sont composées de plusieurs sources de données. Cela peut être d'autres séries utiles, ou des évènements impactants

Les réponses à ces questions influenceront les modèles utilisés qui n'ont pas tous les mêmes possibilités. Nous reviendrons sur le bon choix du modèle selon le cas d'usage dans la partie de modélisation.

Questions pre-processing :

Questions	A qui ?
Quelle est la volumétrie des données disponibles ?	Métier/Data Scientist
Quelle est la maille temporelle des données ?	Métier/Data Scientist
Quelle est la maille produit de l'étude (SKUs, produits) ?	Métier
A quelle maille géographique analyse-t-on ces produits ?	Métier
A quel horizon et quelle fréquence veut-on faire les prévisions ?	Métier
A-t-on besoin d'un modèle déterministe ou probabiliste ?	Métier
Y a-t-il des données exogènes ou covariables disponibles/utiles ?	Métier
Quelle est l'importance de l'explicabilité pour le cas d'usage ?	Métier
Est-ce que les données sont stationnaires ?	Data Scientist
Y a-t-il une saisonnalité dans les données ?	Métier/Data Scientist
S'agit-il de données univariées ou multivariées ?	Data Scientist

b. Structuration des données

Avant de commencer, il est important **d'explorer** les données. Pour cela, il est possible d'utiliser la fonction describe() de pandas :

```
import pandas as pd
df = pd.read_csv(filepath)
df.Date.describe()
```

Les séries temporelles sont souvent dans des **formats non-structurés**, avec **différents types de données** qui se mélangent dans les colonnes. Souvent, la colonne des dates a pour format string et il est nécessaire de la transformer en **datetime**. De plus, il peut arriver que les dates ne soient pas dans l'ordre, et il est alors nécessaire de trier les données par ordre croissant de dates :

```
df['Date'] = pd.to_datetime(df['Date'])
df.sort_values(by=['Date'], inplace=True, ascending=True)
```

c. Evaluation de la qualité des données

L'analyse de séries temporelles peut présenter de nombreux défis, et parfois être frustrant, à cause de la nature même des données. Il est donc important de veiller à la qualité des données, notamment au moment de les extraire. Il existe **4 grandes familles de problèmes de qualité** des données que l'on peut rencontrer dans un problème de série temporelle [2.2] :

- **Validité** : valeurs en dehors de la plage, variations rapides impossibles, ordre des dates inexact
- **Précision** : échantillonnage inconstant, mauvaise précision du capteur

- **Complétude** : champs vides, pas de clés/id, pas de méta data, pas d'information de provenance
- **Temporalité** : décalage entre les données et la réalité

Une première étape dans l'analyse des données est de bien vérifier **l'unité des colonnes** et la validité des données. Ainsi, on peut utiliser des fonctions de pandas pour fixer le format d'une colonne comme pour les dates (2.b.), et parcourir les données pour voir si les valeurs présentes sont **cohérentes, dans la même unité**, … Pour cela, des **visualisations graphiques** et des **analyses statistiques** avec **des box plot** permettent de repérer rapidement des éléments qui semblent incorrectes.

Les différents problèmes rencontrés avec des séries temporelles peuvent permettre d'établir une note sur la qualité des données, et ainsi de mesurer s'il y a des soucis à régler ou non, et par exemple comparer la qualité de différents datasets. Il est aussi possible d'aller plus loin en analysant chaque critère de façon exhaustive. Ainsi, pour la complétude par exemple, on pourra lister les différents points suivants :

- Champs vides
- Unité de mesure manquante
- Dates manquantes

Qualité des données

4 éléments à regarder pour la qualité des données :

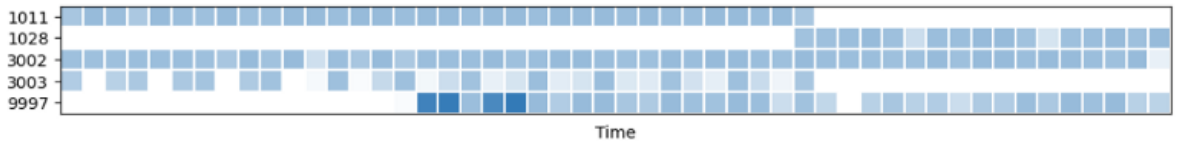
- **Validité** (valeurs hors scope, variations impossibles, …)
- **Précision** (incertitudes sur les données)
- **Complétude** (champs vides, inconstance des unités, …)
- **Temporalité** (décalage données-réalité)

d. Exploratory data analysis (EDA)

L'EDA est une étape cruciale pour comprendre les données utilisées avant de se lancer dans des modèles. Pour commencer, il faut comprendre combien de données nous avons, et notamment le nombre de colonnes : est-ce des données univariées ou multivariées ? Il existe des outils, notamment avec pandas, pour **visualiser rapidement le nombre de données** dans chaque colonne et la **fréquence des mesures** [2.3]. Cela peut être vraiment efficace pour voir rapidement si toutes les colonnes sont **homogènes**. On pourra utiliser pour cela la fonction suivante :

```
from ydata_profiling.visualisation.plot import timeseries_heatmap
timeseries_heatmap(dataframe=df, entity_column='Values', sortby='Date')
```

qui retourne ce genre de heatmap :



Nous pouvons alors voir ici que toutes les données n'ont pas commencé à avoir des enregistrements au même moment, et les couleurs montrent que certaines données peuvent avoir plus de points sur une période donnée. Dans ce cas-ci, il peut être utile d'utiliser une plage de temps dynamique plutôt que prédéterminée pour pallier ces problèmes.

Il est aussi possible de générer avec pandas des **rapports sur les dataframes**. Ces rapports existent aussi pour les séries temporelles grâce au paramètre *tsmode=true*, et montre notamment des résumés du **contenu des différentes colonnes, des études de l'autocorrélation, de la stationnarité et saisonnalité, …**

```
from pandas_profiling import ProfileReport
profile = ProfileReport(dataframe=df, tsmode=True, sortby="Date")
profile.to_file('profile_report.html')
```

Ces fonctions peuvent être très utiles pour repérer des éléments particuliers et propres à la série temporelle. Il faudra compléter ces résultats par des visualisations dépendant du cas d'usage pour bien comprendre les données, les confronter avec le monde réel, … et discuter de ces résultats avec le métier. On pourra aussi **afficher la série temporelle avec différents niveaux d'agrégation des dates** : de la maille la plus fine disponible, puis en agrégant par jours, semaines, mois, année pour voir des comportements particuliers et certaines saisonnalités. Pour cela, la fonction *seasonal_decompose* peut notamment être utile, et retourne **la tendance et saisonnalité de la série** :

```
from statsmodels.tsa.seasonal import seasonal_decompose
sd = seasonal_decompose(df, period=period)
```

L'argument *period* permet d'indiquer différentes périodes d'analyse pour étudier la saisonnalité à la journée, semaine, mois, année, …

3. Pre-processing

Cette partie va permettre de mettre en avant les principales **étapes de pre-processing** de séries temporelles. Dans une analyse de série temporelle, c'est une étape clé qu'il ne faut pas négliger pour utiliser des données fiables et propres. Nous verrons ainsi comment **gérer les valeurs manquantes**, comment **filtrer et débruiter** une série qui serait inexploitable à cause d'un bruit superposé, comment **détecter les outliers** pour les enlever ou les analyser. Enfin, nous verrons les **méthodes de normalisation** de séries temporelles, de **séparation en ensemble de train et de test**, et la **gestion de la stationnarité** qui est un élément très important selon les modèles que l'on veut utiliser.

a. Gestion des valeurs manquantes (irrégularité des mesures)

La gestion des valeurs manquantes de séries temporelles est un véritable défi. Tout d'abord, il faut commencer par se demander s'il est **normal d'avoir des valeurs manquantes**. Il peut y avoir plusieurs raisons à cela :

- **L'enregistrement et le stockage des données** : au moment de l'acquisition des données, il est possible qu'il y ait un **problème sur le capteur ou la mesure**, ce qui enlève une ligne de données. Les données sont alors perdues puisque l'évènement est passé, à moins que d'autres sources ou d'autres capteurs/mesures font les mêmes analyses ;
- **L'extraction des données** : il est possible de perdre une partie des données au moment de l'extraction, que ce soit par exemple **en SQL** ou par un fichier excel. Il faudra alors mettre en place des tests (sur le contenu, le nombre d'éléments, …), vérifier la requête SQL si besoin pour être sûr de ne pas être à l'origine de l'erreur, et vérifier le contenu des tables requêtées ;
- **La nature des données** : certaines données présentent naturellement des données manquantes. Par exemple, il n'y a pas de produits qui sont lancés en entrepôts les week-ends. Les séries temporelles n'auront alors que des valeurs les jours de la semaine. Il faudra alors voir s'il est nécessaire de remplacer ces valeurs ou non, et utiliser les techniques suivantes si besoin.

Nous allons distinguer ici deux types de valeurs manquantes, qui ne peuvent pas se traiter de la même façon : des valeurs manquantes peu nombreuses et étalées, et des blocs de valeurs absents.

Pour peu de valeurs manquantes, les **techniques conventionnelles** [3.1] peuvent fonctionner. La première méthode qui vient à l'esprit peut être de **laisser les valeurs manquantes** et voir les résultats. Pour des modèles de séries temporelles, cette méthode est en général assez peu efficace, et il vaut mieux chercher à remplacer les valeurs manquantes. Il est possible de les remplacer par la **dernière valeur disponible, par la valeur moyenne ou médiane, ou par la valeur la plus donnée dans la colonne.** Pour remplacer les valeurs manquantes d'une colonne, on dispose avec pandas d'une fonction *fillna* qui permet de le faire automatiquement :

```
df.fillna(0) #replace all elements with 0s
df.fillna(method="ffill") #propagate last valid observation to next valid
df.fillna(method="backfill") #use next valid observation to fill gap
df.fillna(value=values, limit=1) #replace the first NaN element with values
```

Nous pouvons aussi utiliser des méthodes plus sophistiquées, comme les **moyennes mobiles, régressions, kNN ou toutes autres méthodes de régression** insensibles aux valeurs manquantes pour remplacer les valeurs manquantes. Il faudra cependant **garder en tête la nature des données de séries temporelles** et l'importance de leur chronologie ! Sur des méthodes de classification par exemple, on créera notamment les clusters selon le jour de la semaine, le mois ou l'année plutôt qu'uniquement les valeurs les plus proches dans le temps. Enfin, il existe aussi des méthodes de Deep Learning avec des librairies toutes faites, qui permettent de compléter les valeurs manquantes. On pourra notamment regarder la librairie datawig. Datawig peut prendre une base de données et ajuster un modèle d'imputation pour chaque colonne contenant des valeurs manquantes, avec toutes les autres colonnes comme entrées. Ces méthodes sont souvent plutôt efficaces, mais peuvent être lentes sur de grands datasets.

Dans le second cas, des blocs entiers de valeurs sont manquants, à cause d'un problème technique ou d'erreurs humaines. **Les techniques conventionnelles ne fonctionnent pas ici** puisque l'ordre dans lequel les valeurs sont attribuées compte. Pour pallier cela, il existe des **méthodes l'interpolation** [3.2] qui utilisent les deux points environnants. Prenons par exemple ce jeu de données avec des valeurs manquantes :

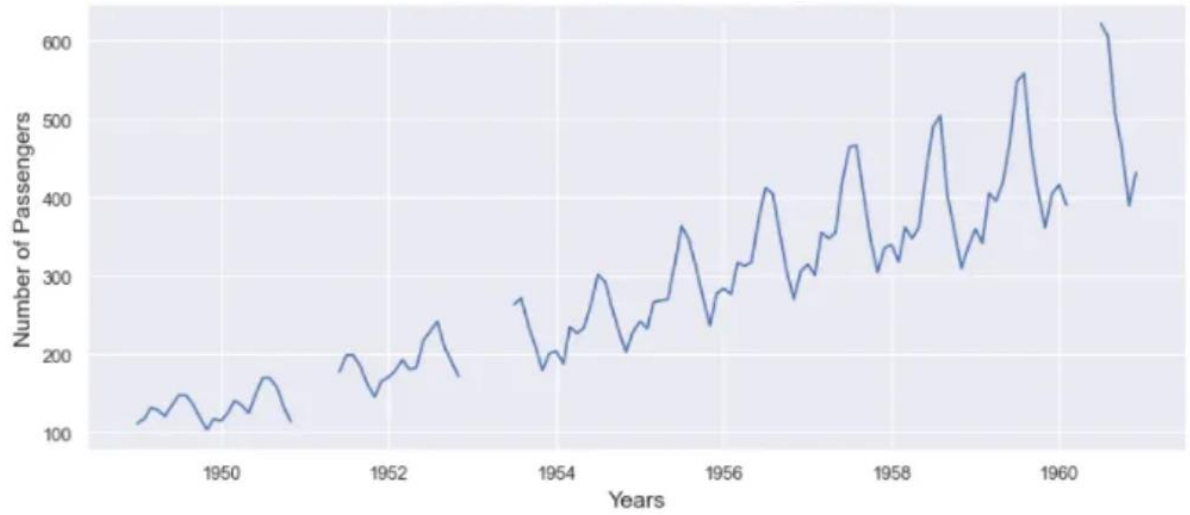


Figure 3.1.1 – Séries temporelles avec valeurs manquantes

L'interpolation linéaire va compléter les valeurs manquantes en traçant la droite entre les deux valeurs environnantes :

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

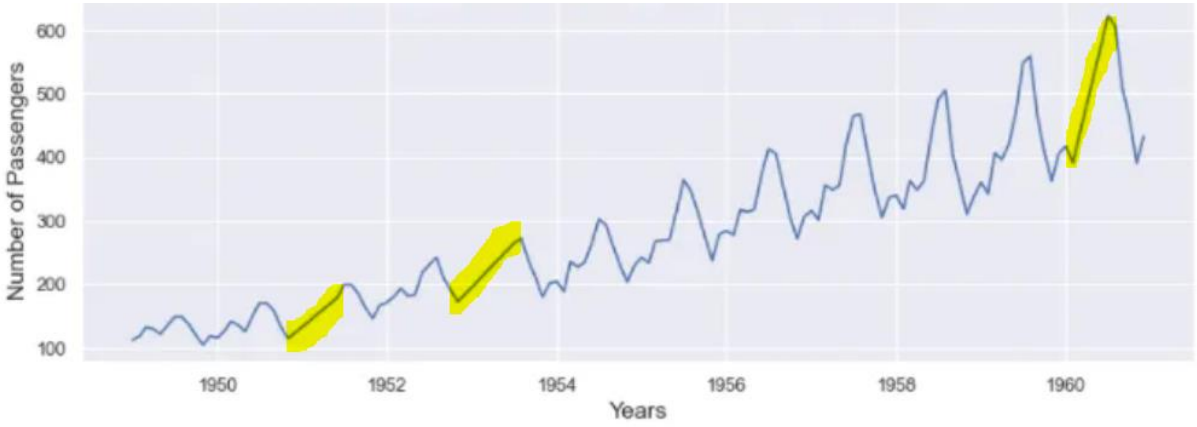


Figure 3.1.2. – Série temporelle interpolée linéairement

Il existe aussi l'interpolation polynomiale, où l'utilisateur peut choisir le degré du polynôme utilisé. Un plus haut degré de polynôme permettra de mieux coller aux données, mais pourra conduire à un overfitting plus important.

```
df['Value'] = df['Value'].interpolate(method='spline', order=3)
```

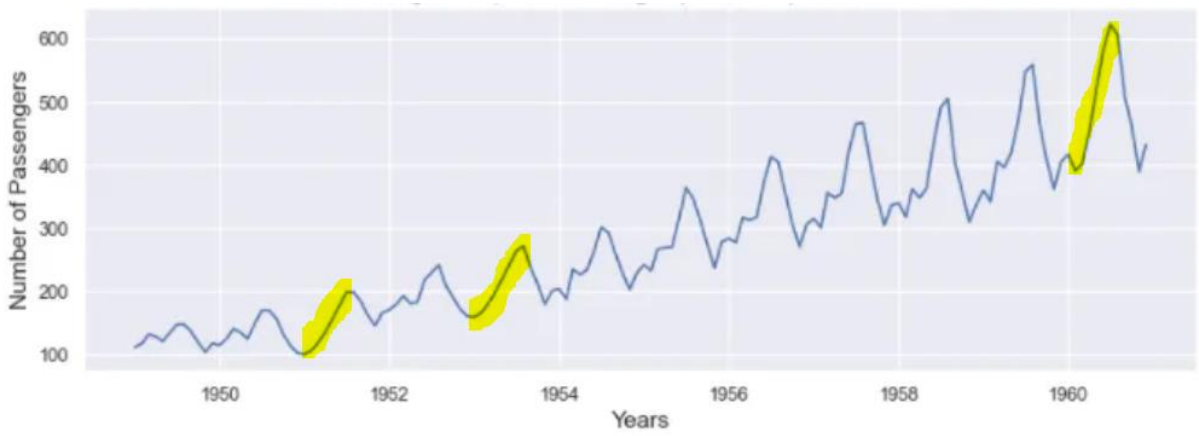


Figure 3.1.3. – Série temporelle interpolée polynomialement

Enfin, il est possible d'utiliser le mode 'time', qui fait une interpolation linéaire en tenant compte des valeurs de temps. Cela est notamment utile si les valeurs manquantes n'ont pas un pas de temps régulier.

```
df['Value'] = df['Value'].interpolate(method='time')
```

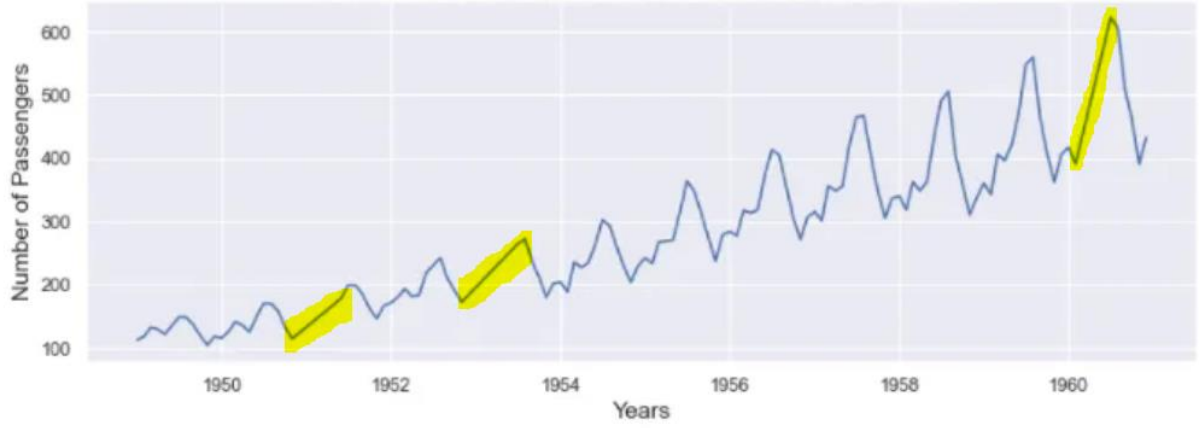


Figure 3.1.4. – Série temporelle interpolée par 'time'

Gestion des valeurs manquantes

2 types de valeurs manquantes :

- **Valeurs éparpillées** : méthodes traditionnelles (dernière valeur disponible, valeur moyenne ou médiane, valeur la plus donnée, modèles moyennes mobiles/kNN/régression)
- **Blocs** : interpolation linéaire, polynomiale

b. Filtrage et débruitage

Le bruit dans les séries temporelles peut causer des problèmes importants, et il est généralement **recommandé de retirer le bruit de la série temporelle** avant les analyses. Ce processus s'appelle le **débruitage** ou « denoising » [3.3].

La première méthode de débruitage est d'utiliser une **moyenne mobile**, en fixant convenablement la taille de la fenêtre : une petite fenêtre retirera peu de bruits, alors qu'une grande fenêtre lissera trop la série. Cet algorithme est simple, mais « violent » : il atténue énormément les fortes variations, il écrête les pics.

```
rolling_df = df['Value'].rolling(20).mean() #rolling window of size 20
```

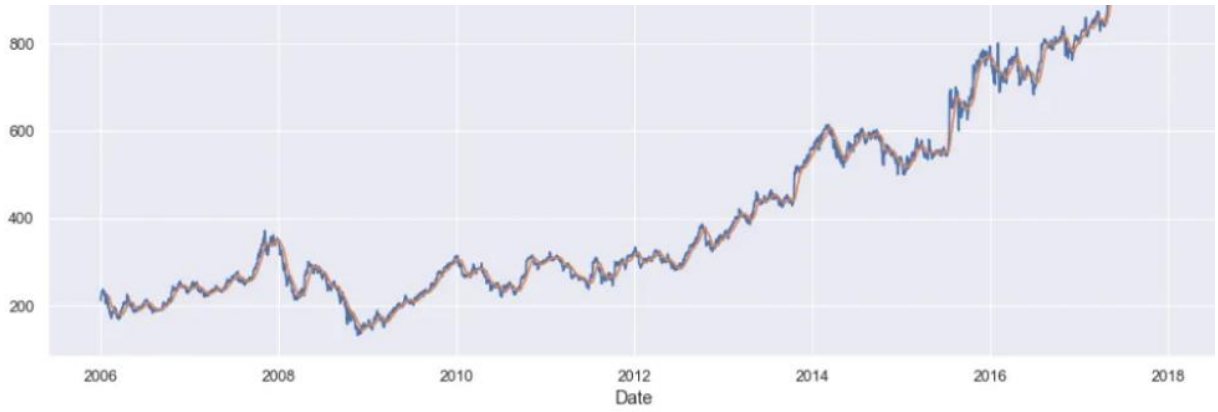


Figure 3.2.1. – Utilisation d'un filtre de moyenne mobile (courbe orange)

Une seconde méthode peut être d'utiliser une **transformée de Fourier** pour passer la série temporelle dans le **domaine fréquentiel**. Une fois dans le domaine fréquentiel, il est possible d'observer la présence et l'impact de chaque fréquence dans la série sur de graphique, et de **fixer un seuil pour enlever les hautes fréquences** correspondant au bruit. Ensuite, nous pouvons appliquer une transformée de Fourier inverse pour récupérer la série filtrée dans le domaine temporel.

```
fhat = np.fft.fft(df['Value'], n) #compute Fourier transform
psd = fhat * np.conj(fhat)/n #power to see the frequencies to keep
psd_idxs = psd > threshold #array of 0 and 1
psd_clean = psd * psd_idxs #zero out all the unnecessary powers
fhat_clean = psd_idxs * fhat #used to retrieve the signal
signal_filtered = np.fft.ifft(fhat_clean) #compute inverse Fourier transform
```

Figure 3.2.2. – Débruitage par transformée de Fourier (courbe orange)

Nous pouvons aussi utiliser les filtres plus traditionnels de traitement du signal. Un **filtre passe-bas** pourra être utile pour laisser passer les fréquences les plus faibles, et retirer les hautes fréquences correspondant à du bruit.

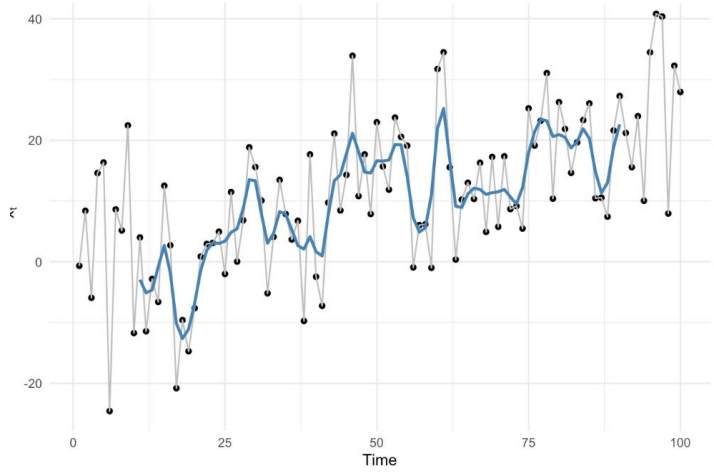


Figure 3.2.3. – Débruitage par filtre passe-bas (courbe bleue)

Un autre type de filtre, les **matching filter** ou filtre de correspondance, permet **d'identifier les sauts** des séries. En réalisant la convolution entre les deux séries, on pourra obtenir un graphe mettant en valeur les sauts importants et leur orientation :

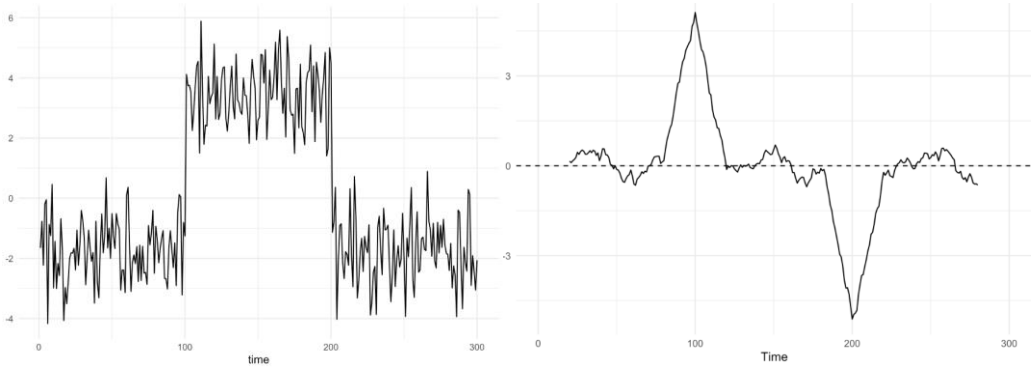


Figure 3.2.4. – Détection de sauts par matching filter

Les modèles d'**exponential smoothing** permettent aussi d'obtenir une série filtrée. Les méthodes de lissages exponentielles, dont il existe plusieurs variantes (simple, double, Holt-Winters), sont des méthodes empiriques de prévision de série temporelle. Elles présentent l'intérêt d'être facilement compréhensibles et leur implémentation récursive en font un outil efficace pour le traitement de gros volumes de données ou dans des systèmes embarqués disposant de peu de mémoire. Ces modèles prennent en compte les dernières valeurs de la série, et les pondèrent par une série géométrique de coefficient λ : $\hat{y}_{t+1} = (1-\lambda)\sum_{i=1}^{\infty}\lambda^i y_{t-i}$. Ainsi, les valeurs les plus proches ont les poids les plus importants, et les valeurs éloignées ont moins de poids. On obtient alors ce genre de figure :

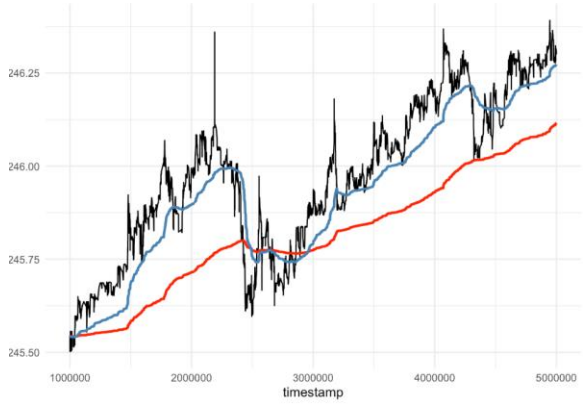


Figure 3.2.5. – Filtrage par exponential smoothing (bleu : λ = 0.999, rouge : λ = 0.995)

Le **filtre de Kalman** est un filtre très utilisé en série temporelle. Ce filtre vient du monde de la physique, où il est important d'évaluer la précision des capteurs et mesures, de les croiser et les comparer pour s'assurer de la précision et l'exactitude des mesures : le filtre de Kalman servait alors à estimer les états d'un système dynamique à partir d'une série de mesures incomplètes ou bruitées. Il a depuis été détourné pour être aussi utilisé pour les séries temporelles, en comparant à chaque étape une prédiction à la valeur réelle pour ajuster les valeurs filtrées : il permet donc de "rattraper" les prédictions par rapport à des valeurs mesurées. La force de ce filtre est sa capacité de prédiction des paramètres et de rectification des erreurs, non seulement des capteurs, mais aussi du modèle lui-même. Dans une méthode d'estimation classique (par exemple, la méthode des moindres carrés), une simple erreur dans la modélisation du système entraine inévitablement une erreur au niveau de l'estimation. Cela peut donc conduire à des erreurs qui s'accumulent, notamment le biais. Ce modèle sous forme discrète peut se résumer dans son cadre le plus simple par l'équation suivante : $\hat{y}_{t+1} = K.y_t + (1-K).\hat{y}_t$. Ici, les valeurs $\hat{y}_t$ sont les valeurs filtrées, alors que y_t est la valeur réelle, et K est un gain de Kalman. Ce filtre permet de débruiter la série temporelle en la lissant, et peut aussi être utilisé lorsqu'il y a des valeurs manquantes.

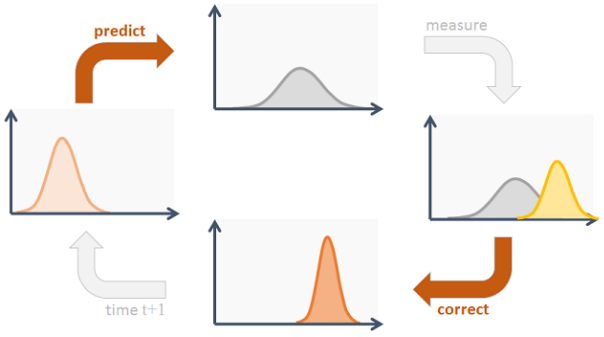


Figure 3.2.6. – Schéma d'un filtrage par filtre de Kalman

Le **filtre de Savitzky–Golay** ou algorithme de Savitsky-Golay est une méthode de traitement du signal utilisée pour lisser des courbes et extraire des dérivées successives. Alors que les méthodes plus traditionnelles comme les moyennes mobiles peuvent entrainer des lissages très violents en atténuant fortement les grandes variations, ce filtre propose une méthode plus souple. On considère alors un polynôme de degré d, que l'on va optimiser sur la fenêtre glissante étudiée : on va pour cela minimiser l'erreur au sens des moindres carrés. On utilisera ensuite ce polynôme pour en déduire la valeur de la série au pas suivant. Dans la pratique, on utilise généralement un polynôme de degré 2 ou 3. Un polynôme de degré 2 permet de prendre en compte la courbure et un polynôme de degré 3 permet de prendre en compte des points d'inflexion. Il est rarement nécessaire d'aller au-delà. Le nombre de points d'un intervalle doit être suffisamment grand devant le degré du polynôme pour que le lissage soit effectif. En effet, s'ils sont égaux, le polynôme passe exactement par tous les points de l'intervalle : il n'y a donc pas de lissage. On prend donc en général un polynôme de degré 3 et une fenêtre glissante d'au moins 5 points. Les valeurs lissées sont des combinaisons linéaires des valeurs réelles de la série. Dans le cas d'un pas constant, les coefficients sont constants, et nous pouvons donc les déterminer par avance, et les stocker dans des tables, ce qui permet ensuite d'avoir un algorithme facile et rapide de mise en œuvre. Nous parlons de coefficients de convolution. Par exemple, pour une fenêtre de 5 points et un polynôme de degré 3 : $\hat{y}_t = \frac{1}{35}(-3y_{t-2} + 12y_{t-1} + 17y_t + 12y_{t+1} - 3y_{t+2})$. Ce filtre rencontrera cependant des difficultés à capturer les dynamiques non linéaires complexes, peut avoir des effets de bord importants et est sensible aux outliers.

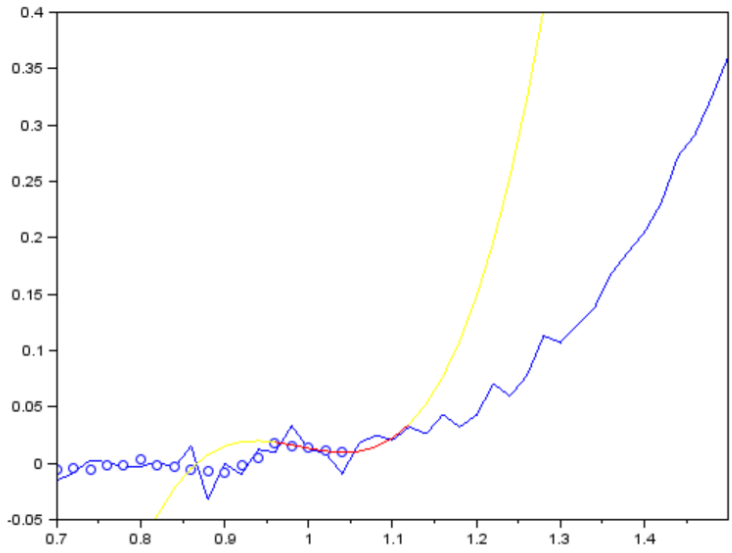


Figure 3.2.7. – Filtrage par algorithme de Savitsky-Golay

Filtrage de séries temporelles

Plusieurs filtres selon les besoins :

- **Moyennes mobiles** : lissage des fluctuations en calculant la moyenne sur une fenêtre glissante, utile pour identifier les tendances à court terme ;
- **Transformée de Fourier** : conversion de la série en fréquentiel, permettant d'analyser les composantes périodiques présentes ;
- **Passe-bas** : atténuation des hautes fréquences pour conserver les variations lentes, efficace pour supprimer le bruit et mettre en évidence les tendances à long terme ;
- **Matching filter** : détection d'un motif spécifique dans une série temporelle, souvent utilisé dans la détection de signaux ou d'événements spécifiques ;
- **Exponential smoothing** : combinaison pondérée des valeurs passées pour donner plus de poids aux observations récentes ;
- **Kalman filter** : correction de la prédiction par comparaison avec les valeurs réelles ;
- **Savitzky-Golay filter** : approximation polynomiale sur une fenêtre glissante.

c. Détection des outliers

Un outlier est une **valeur anormalement haute ou basse**, qui n'a pas un comportement voulu [3.4]. Ces valeurs peuvent avoir de multiples origines, et ont souvent un impact important dans les modèles et prédictions.

Outlier : Une observation qui s'écarte tellement des autres observations qu'elle éveille les soupçons sur le fait qu'elle ait été générée par un mécanisme différent

Pour détecter les outliers, on peut tout d'abord avoir une approche globale en créant des **intervalles de confiance** autour de la série : on pourra par exemple prendre comme limites hautes et basses les valeurs $mean \pm 3.(standard\ deviation)$. Prendre ces valeurs statiques comme des limites globales sur toute la série n'est pas conseillé, notamment pour les séries avec des tendances, saisonnalité, ou variations fortes. Il conviendra d'utiliser ce procédé sur **des fenêtres glissantes**. Cette méthode plutôt simple a de très bons résultats en général.

Il est aussi possible d'utiliser les propriétés statistiques de la série temporelle, avec notamment sa **décomposition saisonnalité-tendance** (STL decomposition). On peut obtenir ce genre de données et graphiques avec la fonction seasonal_decompose de statsmodels :

```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df,model='additive')
```

On peut ensuite **analyser le résidu**, c'est-à-dire la partie de la série temporelle qui n'est pas expliquée par la saisonnalité et la tendance. Ainsi, des valeurs avec des résidus dépassant un seuil déterminé

pourront être classées comme outliers. Cette méthode est assez simple, robuste, et fonctionne dans de nombreuses situations. Son inconvénient est qu'elle peut potentiellement être difficile à adapter : on ne peut jouer que sur le choix du seuil.

Il est aussi possible d'utiliser des **modèles de machine learning** adaptés à la détection d'anomalie. Il s'agit principalement de **modèles de classification** et regression trees. Ces méthodes présentent l'avantage de pouvoir fonctionner de manière **supervisée et non-supervisée** selon la présence d'anomalies labellisées ou non. On peut par exemple analyser le fonctionnement du modèle le plus utilisé pour la détection d'anomalies : **l'Isolation Forest**. C'est un arbre de décision qui fonctionne en isolant des points de données : il prend une partie du dataset et créé des branches jusqu'à ce que des points soient isolés. Pour isoler un point, les partitions sont faites aléatoirement par une division aléatoire entre les valeurs minimale et maximale de la branche. Les **outliers sont plus facilement isolées et donc vont créer des chemins plus courts dans l'arbre**, et on pourra ainsi les repérer. L'inconvénient de ces modèles est que le nombre de features à analyser peut affecter fortement les performances de calcul : il faudra alors les sélectionner par ordre d'importance.

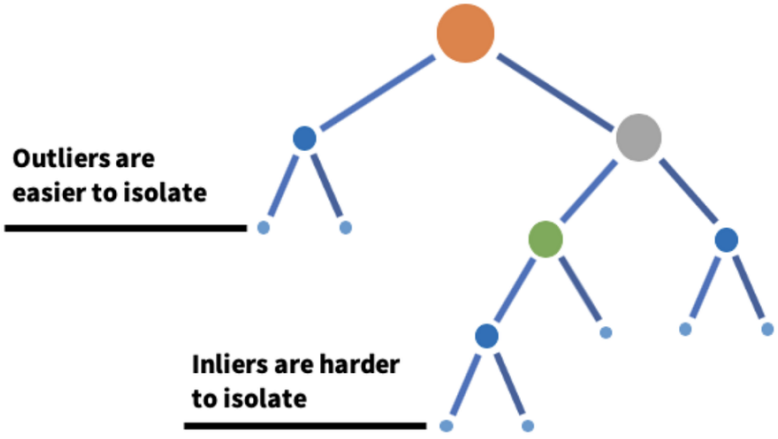


Figure 3.3.1 – Représentation d'un Isolation Forest

Des méthodes de détection fonctionnent aussi sur des **modèles de prédiction** : on utiliser un modèle de prédiction pour **prédire une valeur future** par rapport aux dernières valeurs de la série, et on la **compare à la valeur réelle**. Si l'écart est supérieur à un seuil fixé, nous pourrons considérer la valeur comme une anomalie. Ces méthodes sont performantes dans certains cas car les modèles prennent en compte les différentes saisonnalités, ce qui peut donner de meilleurs résultats que les méthodes précédentes. Cependant, elles seront limitées à des cas précis, notamment s'il y a peu de données ou si l'horizon de prédiction est important.

Enfin, il y a aussi des **méthodes de clustering**, qui ont l'avantage d'être **non-supervisée**. Le **clustering par K-means** est par exemple une méthode non supervisée également très utilisée pour la détection d'anomalies. L'algorithme essaie de regrouper les données selon leur proximité en K clusters. Les anomalies sont alors détectées parce qu'elles vont se situer **loin des clusters**, avec une distance supérieure à un seuil que l'on fixera. Ces méthodes peuvent avoir des performances de calcul qui diminuent quand le nombre de features augmente, et ont des hyperparamètres qu'il faut fine-tuner pour obtenir les résultats souhaités.

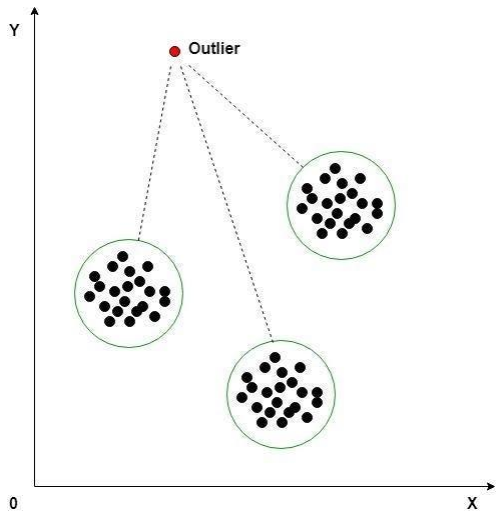


Figure 3.3.2 – K-means clustering pour la détection d'anomalies

Détection d'outliers

Plusieurs méthodes qui peuvent se combiner :

- **Création d'intervalles de confiance** : valeurs hors de l'intervalle de confiance
- **Analyse de résidus** : résidus anormalement importants
- **Méthodes de classification** : Isolation Forest
- **Méthodes de prédiction** : comparaison de valeurs prédites aux valeurs réelles
- **Méthodes de clustering** : kNN

d. Normalisation

La normalisation est un processus classique en machine learning. Elle présente de nombreux avantages, notamment de pouvoir **interpréter les résultats** des modèles en regardant l'importance des différentes features, mais elle permet aussi souvent d'obtenir des **meilleures performances**, si la série temporelle a une envergure ou une distribution consistante. Il existe deux techniques pour faire cela : la **normalisation** et la **standardisation**.

La **normalisation** consiste à ramener la série à des valeurs entre 0 et 1. Cette méthode nécessite d'avoir accès au minimum et maximum des valeurs étudiées. Si la série temporelle varie beaucoup, il peut être difficile d'accéder à ses valeurs, et il vaut mieux se tourner vers d'autres méthodes. Cette méthode est déjà complètement implémentée dans scikit-learn, et peut être utilisée avec les commandes suivantes :

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(values)
normalized = scaler.transform(values)
```

D'un autre côté, la **standardisation** consiste à transformer la distribution des valeurs pour obtenir une série de moyenne 0 et d'écart-type 1. Cela peut se faire notamment en soustrayant la moyenne aux valeurs, puis en divisant par l'écart-type, et nécessite donc d'avoir accès à la moyenne et l'écart-type de la série. Un module est également implémenté dans scikit-learn :

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler = scaler.fit(values)
normalized = scaler.transform(values)
```

Ces deux méthodes semblent assez proches en termes de gain de performances pour les modèles de machine learning. Il est surtout important d'y penser si les performances des modèles sont moyennes ou pour comparer le poids des features dans le modèle.

Normalisation

Pour améliorer les performances et convergence de modèles ML

Pour pouvoir comparer les poids des features et les interpréter

- **Normalisation** : valeurs ramenées entre 0 et 1 ;
- **Standardisation** : moyenne ramenée à 0, écart-type ramené à 1.

e. Train test split

Le train-test-split des séries temporelles est particulier, car nous devons nous assurer que **l'ensemble de test soit plus vieux que l'ensemble d'entrainement**. Il faudra donc conserver les dernières valeurs de la série pour les comparer aux modèles en réalisant le train test split. La cross-validation a un format particulier puisqu'il faut toujours que les valeurs prises pour l'entrainement soient antérieures à celles des prédictions [3.5]. Il ne faut pas non plus utiliser les données de validation pour l'entrainement du modèle pour ne pas créer d'**overfitting** du modèle ! Le caractère aléatoire de la division provient donc uniquement de la taille accordée à l'ensemble d'entrainement et celui de test. Une autre stratégie peut également consister à **conserver une fenêtre d'entrainement de taille constante**, pour éviter d'avoir une trop grande influence de valeurs lointaines. On testera alors à différentes périodes. Cette division peut donc se faire facilement manuellement, en décidant d'un indice limite qui sépare les deux ensembles. Il existe aussi une fonction dans la librairie scikit-learn qui permet de faire cela, en faisant plusieurs (n_splits) séparations différentes en conservant cette contrainte de chronologie entre les deux ensembles.

```
from sklearn.model_selection import TimeSeriesSplit
tss = TimeSeriesSplit(n_splits = 4)
for train_index, test_index in tss.split(X):
    X_train, X_test = X.iloc[train_index, :], X.iloc[test_index,:]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

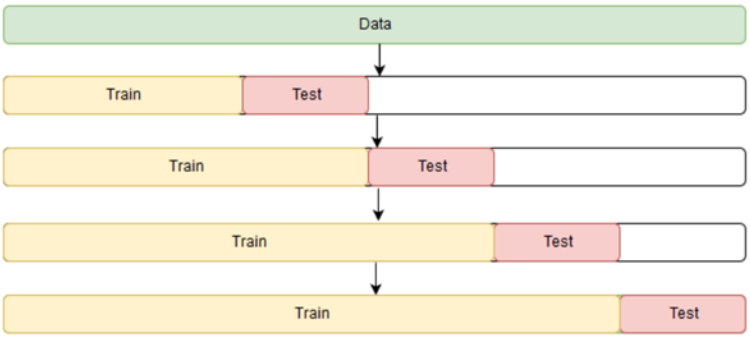


Figure 3.5. – Cross-validation de modèles de séries temporelles

f. Autocorrélation

La **fonction d'autocorrélation** d'une série temporelle traduit la **corrélation de la variable à l'instant t avec les valeurs précédentes de la série** (à t-1, t-2, ..., t-k). Elle est souvent abréviée **ACF** (AutoCorrelation Function). La **fonction d'autocorrélation partielle** mesure la corrélation entre la variable à l'instant t et la variable à un instant t-k **sans tenir compte de l'influence des autres retards** (t-1, t-2, ..., t-k+1). Elle est souvent abréviée **PACF** (Partial AutoCorrelation Function). On pourra les tracer à l'aide des fonctions suivantes :

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(df['values'])
plot_pacf(df['values'])
```

Cela donnera par exemple les graphes suivants, qui peuvent être utilisés pour analyser la saisonnalité des données avec des plus grandes corrélations à des pas réguliers, et également pour déterminer les paramètres des modèles AR ou MA [4.a].

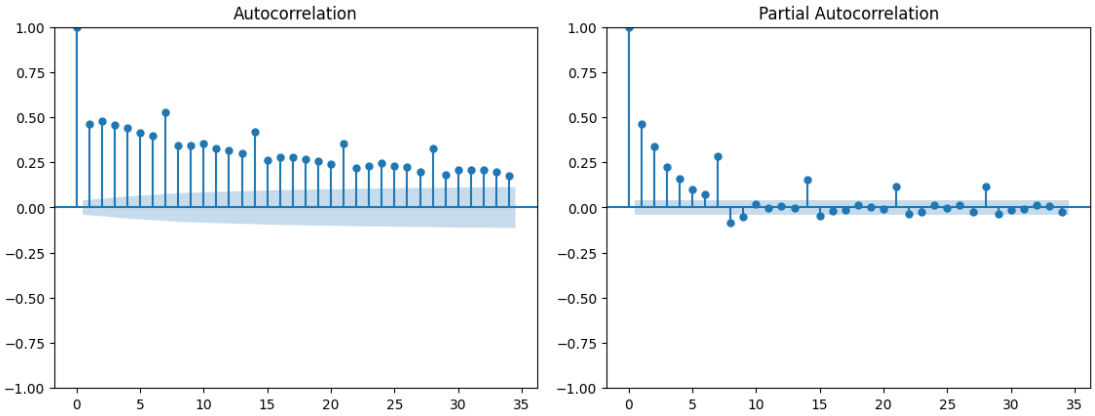


Figure 3.6. – Graphes d'autocorrélation et autocorrélation partielle

A partir de ces graphes, on peut notamment voir une saisonnalité hebdomadaire, avec une corrélation plus importante pour les valeurs espacées de 7 pas.

g. Stationnarité

Une série temporelle est considérée comme stationnaire si ses propriétés statistiques (variance, moyenne, covariance, ...) ne dépendent pas du temps. La plupart des modèles statistiques que nous détaillerons dans la partie suivante nécessite que la série temporelle soit stationnaire. Pour cela, il faut tout d'abord être capable de **savoir si la série est stationnaire ou non** [3.6]. Une première méthode est plutôt **exploratoire**, en traçant la série, pour voir si nous sommes capables de détecter des comportements non-stationnaires. Il est aussi intéressant de regarder la décomposition de la série et ses composantes de tendance, saisonnalité et résidus. Il est souvent possible de remarquer les comportements non-stationnaires les plus marqués, mais nous ne pourrons jamais dire avec certitude que la série est stationnaire.

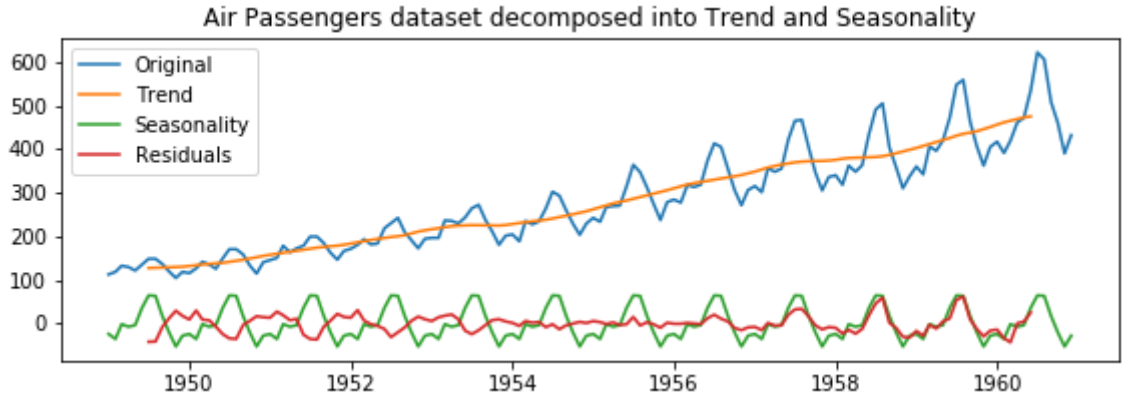


Figure 3.7. – Décomposition d'une série temporelle

Nous voyons ici que la tendance de la série augmente avec le temps : la série n'est donc clairement pas stationnaire.

Des **tests statistiques** ont donc été définis pour cela, et nous donne la réponse avec une précision que l'on peut choisir. Le test le plus utilisé est le **test augmenté de Dickey-Fuller (ADF test).** Ce test fonctionne en testant les hypothèses suivantes :

- Hypothèse nulle : la série temporelle n'est pas stationnaire et présente une racine unitaire, ce qui signifie qu'elle est fortement dépendante du temps ;
- Hypothèse alternative : la série temporelle est stationnaire, c'est-à-dire qu'elle ne présente pas de racine unitaire et est faiblement dépendante du temps.

L'hypothèse nulle sera rejetée si le score est plus petit qu'un score critique fixé, c'est-à-dire si la p-value est inférieure au niveau toléré. En général, on fixe le seuil à 95%, et on veut donc que la **p-value renvoyée par le test soit inférieure à 0,05**. Ce test est implémenté dans la librairie statsmodels, et peut être utilisé de la manière suivante :

```python
from statsmodels.tsa.stattools import adfuller
def adf_test(timeseries):
    # Perform Augmented Dickey-Fuller test
    result = adfuller(timeseries, autolag='AIC')
    # Extract and print test results
    print("ADF Statistic:", result[0])
    print("p-value:", result[1])
    print("Critical Values:")
    for key, value in result[4].items():
        print(f"\t{key}: {value}")
    # Interpret the results
    if result[1] <= 0.05:
        print("The series is likely stationary (reject the null hypothesis)")
    else:
        print("The series is likely non-stationary")
```

Il existe également d'autres tests de stationnarité comme le **Kwiatkowski Phillips Schmidt Shin (KPSS) test**, que l'on peut aussi trouver dans la librairie statsmodels, mais qui reste moins utilisé.

Si la série temporelle n'est pas stationnaire, il existe différentes méthodes pour la rendre stationnaire. Toutes ces méthodes vont transformer les données, mais elles sont **inversibles** pour que nous puissions ensuite **reconstruire les données** une fois les prédictions réalisées. Tout d'abord, plusieurs transformations peuvent être essayées, en passant par exemple au **logarithme**, en prenant la **racine carré ou cubique** des données, ou tout autre **transformation proportionnelle**. La méthode la plus efficace et la plus utilisée est la **différentiation**. Pour cela, chaque terme est soustrait au précédent, ce qui permet d'effacer les tendances. Cela peut se faire très simplement avec numpy, et peut être répété plusieurs fois pour atteindre la stationnarité :

```python
differentiated_data = np.diff(data)
reconstruction = [data[0]]
reconstruction.extend(differentiated_data)
reconstruction = np.array(reconstruction)
reconstructed_data = reconstruction.cumsum()
```

Il ne faut pas oublier de remettre le premier terme pour reconstruire les données originales. De plus, si l'opération est effectuée dans un DataFrame, il faut penser à utiliser la fonction dropna() car la différentiation va enlever la première valeur qui restera non définie. Il faudra donc la remplacer par un 0 ou une autre valeur pour ne pas créer d'erreurs dans la modélisation.

Stationnarité

- Tester la stationnarité avec un test avancé de **test augmenté de Dickey-Fuller (ADF test)**
- Rendre la série stationnaire par :
    - **Transformation** : log, racine carré/cubique, proportionnelle, …
    - **Différentiation** : soustraire à chaque terme le précédent

4. Modélisation

Dans cette partie, nous allons évoquer les différents types de modèles utilisés pour les prévisions de séries temporelles. Pour chaque modèle, nous évoquerons rapidement la **théorie** associée, ses **avantages/inconvénients** et son **implémentation**. Il existe trois grandes familles de modèles pour les prévisions de séries temporelles : les **méthodes statistiques**, les **méthodes de machine learning**, et les **méthodes de deep learning**.

a. Modèles statistiques

i. Données univariées

Dans un premier temps, nous allons regarder les principaux modèles avec des **données univariées**. Comme établi dans la [2.a], on parle de **données univariées s'il s'agit de l'étude d'une unique série temporelle**. Ce sont les méthodes les plus classiques autour de la modélisation de séries temporelles.

Il existe tout d'abord des modèles statistiques qui permettent de réaliser des prédictions de séries temporelles. Ces méthodes peuvent souvent se **combiner** pour obtenir des modèles plus complexes et performants, et nous allons donc les traiter par ordre de complexité.

Pour commencer, le modèle le plus classique est le **modèle autorégressif AR** : dans un modèle AR(p), nous considérons que chaque valeur est une **combinaison linéaire des p valeurs précédentes**, plus un terme d'erreurs aléatoires et une constante :

$$\hat{y}_t = c + \sum_{i=1}^{p} A_i y_{t-i} + \varepsilon_t$$

Ce type de modèles utilise uniquement les valeurs passées pour prédire de nouvelles valeurs. Cette méthode statistique est généralement utilisée pour analyser des phénomènes naturels, des processus économiques, … Les méthodes des moindres carrés ou de maximum de vraisemblance sont généralement utilisées pour déterminer les paramètres du modèle c et A_i. Le modèle AR a pour avantage d'être :

- assez **simple à comprendre et à mettre en place** ;
- **efficace en termes de complexité computationnelle**, ce qui peut être très important pur de larges datasets.

Cependant, il y a aussi quelques inconvénients à l'utilisation de ce modèle :

- il ne s'utilise qu'avec des données **stationnaires** : il faut donc tout d'abord vérifier la stationnarité de nos données, et les rendre stationnaires si besoin [3.g] ;
- étant donné que les modèles autorégressifs ne prévoient les valeurs que sur la base d'informations passées, ils supposent que les **comportements fondamentaux resteront inchangés**. Par conséquent, les prévisions peuvent être inexactes ou surprenantes si les forces

sous-jacentes changent, en particulier lorsqu'un secteur subit des transformations technologiques rapides.

Il existe plusieurs librairies pour mettre en place ce genre de modèle, et on pourra par exemple utiliser la fonction suivante :

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(list(df_train['values']), order=(p, 0, 0))
model = model.fit()
prediction = model.forecast(steps=len(df_test))
```

La seconde méthode classique est le modèle **moving average MA** : dans un modèle MA(q), nous considérons que chaque valeur est une **combinaison linéaire des q erreurs précédentes**, plus une constante. Ce que nous appelons l'erreur est la **différence entre la valeur réelle et la moyenne mobile locale**. Ainsi, un grand écart entre la moyenne mobile et la valeur réelle (correspondant à un pic important éloigné de la moyenne) aura un grand impact dans la prédiction de la valeur future. Ce modèle se formalise de la manière suivante :

$$\hat{y}_t = c + \sum_{j=1}^{q} B_j e_{t-j} + e_t$$

Les modèles MA ont pour avantages qu'ils sont :

- plutôt **faciles à comprendre et à mettre en place** ;
- **peu couteux computationnellement**.

Cependant, les modèles MA nécessitent également des données **stationnaires**, et ont tendance être e**n retard lors de changements rapides** dans les données car le pic met du temps à être pris en compte avec la moyenne mobile. De même, on pourra utiliser la fonction suivante pour mettre en place ce genre de modèles :

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(list(df_train['values']), order=(0, 0, q))
model = model.fit()
prediction = model.forecast(steps=len(df_test))
```

Une fois ces deux modèles présentés, il est possible de les **combiner** pour obtenir une modélisation plus complète et plus performante. Chaque terme sera donc exprimé comme **combinaison linéaire des p dernières valeurs (partie AR(p)) et comme combinaison linéaire des q dernières erreurs (partie MA(q))**. On parle alors de modèle **ARMA(p, q)**.

Mais comment choisir les valeurs de p et q ? Pour cela, il faut utiliser les graphes d'autocorrélation et d'autocorrélation partielle [3.f]. Il existe des règles selon la forme des graphes :

- Si l'ACF s'annule à un certain rang q, et que la PACF décroit exponentiellement, on peut choisir un modèle MA(q) ;
- Si la PACF s'annule à un certain range p, et que l'ACF décroit exponentiellement, on peut choisir un modèle AR(p) ;

- Si la PACF et l'ACF décroissent exponentiellement sans s'annuler il faut alors choisir un modèle ARMA(p, q). Le **calcul d'un critère d'information** [7.d] devient alors nécessaire pour sélectionner les paramètres p et q les plus pertinents.

Nous pouvons donc résumer ces règles dans le tableau suivant :

ACF\PACF	***PACF s'annule au pas p***	***PACF décroit exponentiellement***
ACF s'annule au pas q	-	MA(q)
ACF décroit exponentiellement	AR(p)	ARMA(p, q)

La contrainte de stationnarité est souvent l'étape la plus critique de ce genre de modèle, car il est difficile en pratique de satisfaire cette hypothèse avec certitude. Pour intégrer cette étape directement dans le modèle, le modèle **ARIMA** présente un **paramètre d** qui correspond au **nombre d'intégration/différentiation souhaité pour obtenir une série stationnaire**. Ainsi, l'utilisateur n'a plus besoin de différencier lui-même la série, mais peut utiliser directement la fonction ARIMA(p, d, q). Il lui reste cependant à déterminer le nombre d'intégration requis pour obtenir la stationnarité.

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(list(df_train['values']), order=(p, d, q))
model = model.fit()
prediction = model.forecast(steps=len(df_test))
```

Toujours pour obtenir des modèles plus performants et complexes, sont apparus les modèles **SARIMA**. Ce sont les mêmes modèles que les modèles ARIMA, avec des **paramètres supplémentaires pour prendre en compte la saisonnalité** de la série. Ainsi, ce genre de modèle s'écrit sous la forme **SARIMA(p, d, q, P, D, Q, m)**, avec les paramètres :

- p : le nombre de termes de la composante autoregressive (voir AR(p)) ;
- d : le nombre d'intégration nécessaire pour rendre la série stationnaire ;
- q : le nombre de termes de la composante moving average (voir MA(q)) ;
- P : le nombre de termes de la composante saisonnale autoregressive. On prendre ici les P derniers termes avec un pas de m pour cette composante ;
- D : le nombre d'intégration nécessaire pour rendre stationnaire la série à l'ordre m ;
- Q : le nombre de termes de la composante saisonnale moving average. On prendre ici les Q dernières erreurs avec un pas de m pour cette composante ;
- m : la saisonnalité des données. Pour une saisonnalité hebdomadaire avec une donnée par jour, on prendra m = 7. Si on a une donnée par heure, on prendra m = 24x7.

On pourra utiliser le code suivant pour mettre en place ce genre de modèle :

```
import statsmodels.api as sm
model = sm.tsa.SARIMAX(list(df_train['values']), order=(p,d,q),
seasonal_order=(P,D,Q,m))
model = model.fit()
prediction = model.forecast(steps=len(df_test))
```

Que ce soit pour un modèle ARIMA ou SARIMA, des fonctions existent pour **rechercher automatiquement de manière optimisée les meilleurs paramètres**. On pourra par exemple utiliser :

```
import pmdarima as pm
pm.auto_arima(df_train['values'], start_p=1, start_q=1,
        max_p=7, max_q=7,
        m=m,
        d=d,
        seasonal=True,
        start_P=0,
        trace=True,
        error_action='ignore')
```

Le deuxième type de modèle statistique est le lissage exponentiel ou **Exponential Smoothing**. Comme son nom l'indique, c'est une méthode de lissage des données qui peut servir comme méthode de prédiction. Commençons par le **lissage exponentiel simple**. Cette méthode va considérer que chaque valeur s'exprime comme la somme pondérée des valeurs précédentes. Les valeurs les plus proches auront les poids les plus importants, et les poids vont décroitre exponentiellement. Cela peut se formaliser de la manière suivante :

$$\hat{y}_t = \alpha y_t + (1-\alpha)\hat{y}_{t-1}$$

Ou encore :

$$\hat{y}_t = \sum_{i=0}^{t-1} \alpha(1-\alpha)^i y_{t-i}$$

Le paramètre α dans [0, 1] est à estimer et correspond à la mémoire de la prédiction : plus α est proche de 1 plus les observations récentes influent sur la prévision, à l'inverse un α proche de 0 conduit à une prévision très stable prenant en compte un passé lointain.

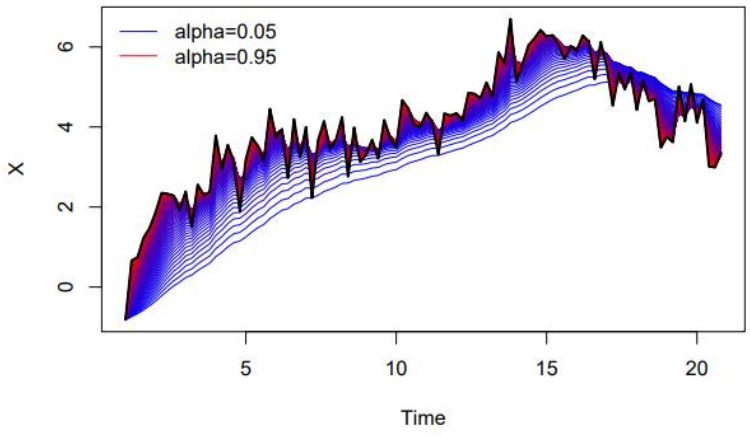


Figure 4.1. – Lissage exponentiel simple

Le modèle peut ensuite est complexifié en utilisant une droite au lieu d'une simple valeur dans l'approximation locale de la série : on parle alors de **lissage exponentiel double** ou de **Holt.** Cela est notamment utilisé quand il y a une tendance dans les données. On ne rentrera pas ici dans les équations correspondantes, mais comme dans le cas simple, il est possible d'exprimer tous les termes sous forme d'équation. On aura alors ce type de modèles :

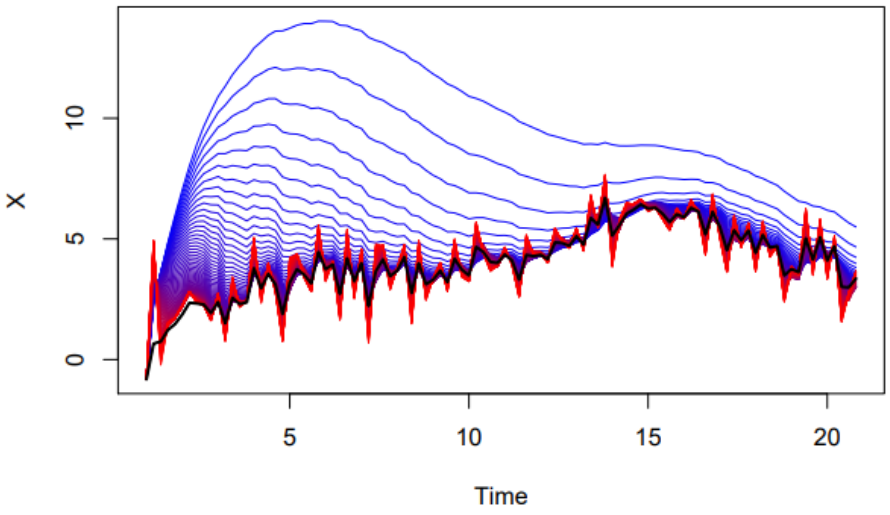


Figure 4.2. – Lissage exponentiel double

Enfin, le modèle le plus complexe établi permet de prendre en compte la tendance ainsi que la saisonnalité. On parle alors de **Triple Exponential Smoothing** ou **méthode de Holt-Winters.** Cette approche est une généralisation du lissage double, qui permet entre autres de proposer les modèles suivants :

- tendance linéaire locale, comme pour le lissage exponentiel double ;
- tendance linéaire locale + saisonnalité (modèle additif) ;
- tendance linéaire locale * saisonnalité (modèle multiplicatif).

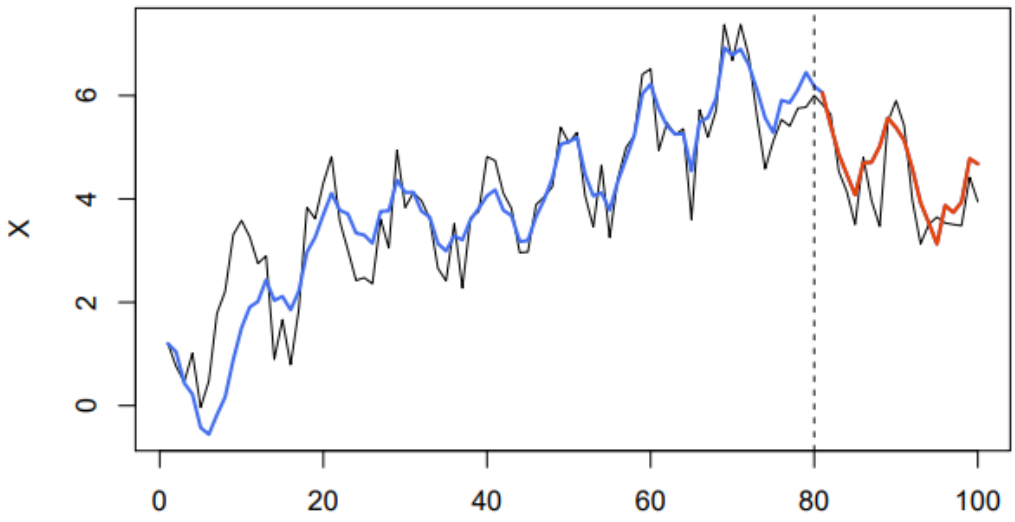


Figure 4.3. – Lissage exponentiel triple

Les modèles d'Exponential Smoothing présentent de nombreux avantages :

- ils sont plutôt **faciles à appliquer** et permettent de se placer à **différents niveaux de complexité** ;
- ils sont **robustes aux outliers et au bruit**, ce qui peut être très utile dans certaines conditions ;
- ils peuvent **répondre assez rapidement à des changements** dans les données car ils donnent plus de poids aux valeurs récentes.
-

Ils ont tout de même quelques inconvénients :

- ils ne pourront pas prédire des changements brutaux, car les valeurs prédites ne dépendent que des valeurs passées. Les pics pourront apparaître mais avec un certain **retard** ;
- pour des **saisonnalités** importantes, ces modèles peuvent mettre **beaucoup de temps pour converger**. Par exemple, pour une saisonnalité annuelle avec des données journalières, il sera quasiment impossible d'utiliser ce modèle.

Pour l'utiliser on pourra par exemple utiliser les librairies et fonctions suivantes :

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
model = ExponentialSmoothing(df_train,
        trend="add",
        seasonal="add",
        seasonal_periods=seasonal_periods).fit(smoothing_level=alpha,
                                               smoothing_trend=gamma,
                                               smoothing_seasonal=delta)
prediction = model.forecast(steps = len(df_test))
```

Il reste cependant à estimer les trois paramètres α, γ et δ. Pour cela, nous devons effectuer une **recherche d'hyperparamètres avec une validation croisée** sur un ensemble de test. Cette étape peut être relativement couteuse selon la taille des données, et il existe des librairies qui présentent des modèles avec cette recherche intégrée.

D'autres modèles statistiques existent également pour les données univariées mais sont moins classiques que ceux présentés ici. Le lecteur pourra faire ses propres recherches si besoin sur les modèles :

- **ARCH** et **GARCH** : ce sont des modèles principalement utilisés en finance et économie, qui utilisent la variance des données pour réaliser les prédictions, notamment quand elle n'est pas constante ;
- La **méthode Theta** : c'est une méthode qui combine deux prédictions réalisées par des Simple Exponential Smoother, et qui les utilisent pour réaliser la prédiction finale ;
- Des **modèles bayésiens** : comme les techniques bayésiennes, ils intègrent des informations a priori et des incertitudes dans les prédictions, et estiment les paramètres du modèle pour faire les prédictions.

En dehors des modèles, certaines méthodologies peuvent être utilisées également pour faire face à une nature spécifique des données. Par exemple, dans le luxe, pour la prévision des ventes, il peut y avoir des ventes intermittentes, c'est-à-dire avec des journées sans vente. Pour cela, la **méthode Croston** (1972) [4.1] peut par exemple être intéressante, et se résume par les éléments suivants :

- Evaluer les ventes moyennes quand il y a des ventes ;
- Evaluer le temps moyen entre deux ventes ;
- Prédire les ventes comme produit du niveau de ventes par la probabilité de ventes.

ii. Données multivariées

Comme nous l'avons vu dans la partie [2.a], des **données multivariées** sont composées de plusieurs sources de données. Cela peut être d'autres séries utiles, ou des évènements impactants.

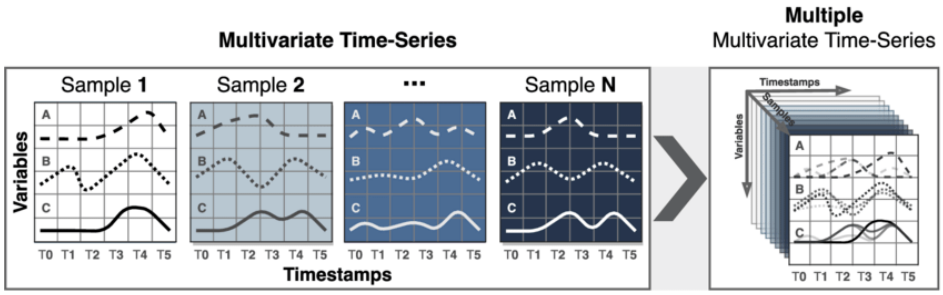


Figure 4.4. – Données multivariées

Traiter toutes ces données d'un bloc présente deux avantages :

- **Améliorer les performances globales** car les données d'une série peuvent améliorer les prédictions d'une autre ;
- **Gagner en temps de calcul**, car nous n'avons plus qu'un seul modèle, et pas un modèle par série.

Il existe donc les modèles **VAR** et **VARMA**, avec le V pour Vector, qui prennent en entrée des données multivariées. La forme des modèles restent identiques, seulement toutes les expressions sont remplacées par des **vecteurs et matrices**. Pour un modèle VAR, on aura donc par exemple :

$$\hat{Y}_t = C + \sum_{i=1}^{p} A_i Y_{t-i} + E_t$$

On peut donc les utiliser avec les fonctions suivantes :

```
from statsmodels.tsa.api import VAR
model = VAR(train_multi)
x= model.select_order()
result = model.fit(x)
lagged_Values = train_multi.values[-x:]
pred = result.forecast(y=lagged_Values, steps=len(df_test))
```

La question principale autour de ces méthodes est de savoir **quelles séries apportent de la valeur** pour les prédictions. En effet, il ne suffit pas de mettre toutes les séries, ce qui peut devenir un souci en rendant le modèle lent et couteux. Pour cela, le **test de causalité de Granger** (1969) a été mis en place. C'est un test venant de l'économétrie pour savoir si une variable est pertinente pour en prédire une autre. C'est une approche de la causalité qui renvoie non pas au caractère théorique de la causalité (cause-effet) mais au caractère prédictif de l'éventuelle cause sur l'effet. En effet, selon Granger, une variable y_1 cause une autre variable y_2, si la connaissance des valeurs passées de y_1 rend meilleure la prévision de y_2 : on dit que y_1 Granger-cause y_2.

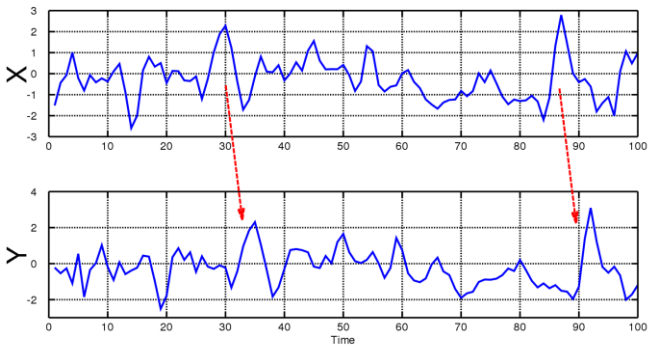


Figure 4.5. – X Granger-cause Y

b. Modèles de Machine Learning

Des approches de Machine Learning ou de régression sont aussi possibles pour des séries temporelles. La démarche de ce type de solution est de **transformer les données en tableau** avec différentes features pertinentes pour ensuite pouvoir utiliser tous les modèles classiques de régression. On pourra ainsi créer des features pertinentes à partir des dates et des données. On peut notamment les regrouper selon leur nature :

- **Features à partir des dates** : on peut transformer les dates en données exploitables (heure, jour de la semaine, semaine, mois, année, …). Parfois, selon les modèles, il peut être intéressant de traduire la cyclicité en variables nominales. Cela permet de conserver la continuité du temps. Par exemple, l'heure d'une journée peut être encodée avec des cosinus et sinus pour traduire la périodicité. On peut également ajouter des features intéressantes à l'aide des dates (vacances, jours fériés, jours ouvrés, weekend ou non, jour ou nuit, …) ;
- **Features à partir des valeurs** : on peut tout d'abord prendre directement certaines valeurs passées comme features, en considérant que les valeurs passées expliquent les valeurs futures comme dans un modèle AR. On peut ensuite les transformer pour en extraire des statistiques pertinentes : moyenne mobile, minimum, maximum, taux de croissance, …

Une fois le pre-processing réalisé, les méthodes de machine learning sont applicables : **régression linéaire ou polynomiale, Decision Tree, RandomForest, kNN, XGBoost, …**

Un modèle de Machine Learning adapté aux séries temporelles a été conçue par Meta en 2019 : **Prophet**. Ce modèle est particulièrement bon pour gérer des saisonnalités multiples. Ce modèle peut se décomposer comme la somme de trois composantes (tendance, saisonnalité, vacances) plus un terme d'erreur :

$$y_t = g_t + s_t + h_t + \varepsilon_t$$

avec :

- g_t **la tendance** : comme pour une régression linéaire ou logistique, ce paramètre donne la tendance générale des données. La nouvelle idée apparue avec Prophet est que cette tendance **peut changer à des points de cassure** : les "changepoints". Prophet peut automatiquement détecter ces points de cassure ou alors cela peut être établi manuellement (ce qui peut être très intéressant pour intégrer des connaissances métiers). Il est aussi possible

d'ajuster à quel point la tendance peut changer aux changepoints, et combien de points sont pris en compte pour la détection automatique des changepoints. Entre deux changepoints, la tendance **peut être linéaire** (valeur par défaut) ou **logistic** si notre modèle a un maximum C. Dans ce cas, cette fonction s'écrit comme : $g_t = \frac{C_t}{1+x^{-k(t-m)}}$ avec C_t la capacité qui peut varier en fonction du temps, k et m respectivement le taux de croissance et le décalage qui peuvent changer aux changepoints. g_t peut aussi être nul ou "plat" s'il n'y a pas de tendance ;

- $\boldsymbol{s_t}$ **la saisonnalité** : c'est une série de Fourier, somme de sinus et cosinus : $s_t = \sum_{n=1}^{N}(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right))$. Le modèle va automatiquement détecter le nombre optimal de termes dans la série, aussi appelé l'ordre de la transformée de Fourier. Mais il est également possible de fixer ce paramètre manuellement selon les besoins. Il est également possible de choisir entre saisonnalité additive ou multiplicative ;
- $\boldsymbol{h_t}$ **les vacances** : cette fonction permet de **prendre en compte les vacances ou des évènements majeurs**. Elle prend une liste de dates (il y a une liste de dates de vacances déjà établie pour de nombreux pays), et quand le modèle arrive sur ces dates, il ajoute ou soustrait une valeur selon l'historique des données de vacances. Il est aussi possible d'intégrer des dates autour de celles déjà rentrés, pour prendre en compte par exemple le temps entre Noël et le jour de l'an.

Lors des prédictions, le modèle renvoie un dataframe avec plusieurs colonnes :

- **yhat** : qui contient les valeurs prédites ;
- **yhat_lower** : qui contient la valeur minimale de l'intervalle de confiance ;
- **yhat_upper** : qui contient la valeur maximale de l'intervalle de confiance.

Ce modèle présente de nombreux avantages :

- le modèle est très **facile d'utilisation**, et peut renvoyer des prédictions correctes en quelques lignes, **sans features engineering** ;
- Prophet ne nécessite **pas** de travailler avec **des séries stationnaires** ;
- Prophet fonctionne plutôt bien pour des **prédictions à moyen terme** (6 mois), même si l'intervalle de confiance peut devenir important ;

Mais il a également quelques inconvénients :

- Il peut être **difficile de fine-tuner le modèle** si beaucoup d'évènements extérieurs perturbent le signal ;
- La tendance explique aussi une grande partie de la prédiction du modèle. Si la **tendance n'est pas bien estimée**, la **performance** peut **décroître** très fortement ;
- Par défaut, Prophet utilise seulement les premiers **80% de l'historique** pour ajuster notamment les changepoints. Il faut donc souvent augmenter cette valeur pour prendre en compte les dernières valeurs qui ont plus d'impact pour les prévisions ;
- Un autre point d'attention est la **sensibilité aux outliers**. La documentation officielle recommande de les retirer pour ne pas fausser les prédictions ;
- Parfois, **rajouter plus de données peut faire perdre en performance**. En effet, Prophet ne pondère pas plus les valeurs les plus récentes. Lui donner des vieilles valeurs peut donc être contre-productif.

On pourra utiliser Prophet avec les commandes suivantes. Il peut cependant être assez difficile d'installer la librairie de Prophet :

```
from prophet import Prophet
df_train.columns = ['ds', 'y']
df_train['ds'] = pd.to_datetime(df_train['ds'])
model = Prophet(interval_width=0.95)
model.fit(df_train)
future_dates = model.make_future_dataframe(periods=len(df_test), freq='D')
prediction = model.predict(future_dates)
```

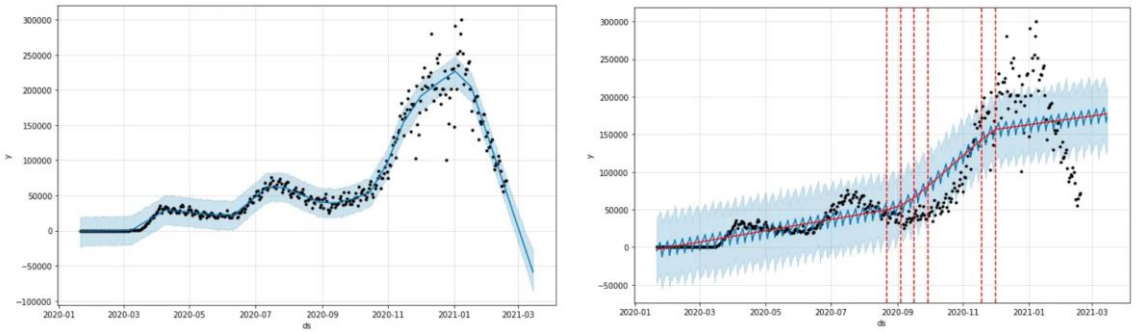


Figure 4.6. – Prédictions avec Prophet, avec intervalle de confiance et points de rupture

Pour traiter les données multivariées avec les modèles de Machine Learning, nous pourrons **utiliser les autres séries temporelles comme features** avec pre-processing ou non. Contrairement à un modèle VAR par exemple, nous conserverons alors **un modèle par série**. Il faut tout de même s'assurer que nous **avons toutes les données sur l'horizon de prédictions**, car nous aurons besoin de toutes les features à ce moment-là.

c. Modèles de Deep Learning

Des modèles de Deep Learning peuvent aussi être intéressants et s'appliquer au cas des séries temporelles. Il existe tout d'abord plusieurs typologies de modèles de Deep Learning :

- **Modèles de séries chronologiques en profondeur (Deep Time Series Models)** : modèles avec des architectures comme les **réseaux de neurones récurrents (RNN)** et les **réseaux de neurones à mémoire à long terme (LSTM)** ;
- **Apprentissage en profondeur par renforcement (Deep Reinforcement Learning)** : les **agents** apprennent à prendre des décisions en fonction des états de la série chronologique ;
- **Modèles de séries chronologiques causales (Causal Time Series Models)** : permettent de comprendre les **relations de causalité** entre les variables d'une série chronologique, ce qui peut être utile pour identifier les causes de certaines tendances ou anomalies ;
- **Modèles de séries chronologiques basés sur l'attention (Attention-based Time Series Models)** : mettent l'accent sur les **parties les plus pertinentes de la série chronologi**que, ce qui peut améliorer la précision des prévisions ;
- **Modèles de séries chronologiques basés sur les GAN (Generative Adversarial Networks)** : permettent **génération de données**, ce qui peut être utile pour simuler des scénarios futurs ou pour la génération de données de test.

Faisons une rapide description des méthodes classiques de Deep Learning :

- **RNN** : Les RNN sont un type de réseau de neurones conçu pour fonctionner avec des données séquentielles, telles que des séries chronologiques. Il dispose d'une boucle de rétroaction qui permet aux informations de persister, ce qui le rend adapté à la modélisation de données de séries chronologiques. Cependant, les RNN peuvent souffrir du problème du vanishing gradient, ce qui peut rendre difficile l'apprentissage des dépendances à long terme ;
- **LSTM** : Les LSTM sont un type de RNN conçu pour résoudre le problème du vanishing gradient. Il dispose d'une cellule mémoire capable de stocker des informations pendant de longues périodes, lui permettant d'apprendre des dépendances à long terme ;
- **NBEATS** (Neural Basis Expansion Analysis for Time Series) : NBEATS est un modèle d'apprentissage en profondeur qui utilise des doubles couches entièrement connectées pour recréer les mécanismes des modèles statistiques. Il s'agit d'un modèle rapide et interprétable qui s'est avéré plutôt performant en comparaison des méthodes statistiques ;
- **Transformers** : Les transformers, développés à l'origine pour le NLP, peuvent également être utilisés pour les séries temporelles. Il y a cependant des contraintes de temps et de mémoire du mécanisme d'attention qui peuvent empêcher le modèle de prendre en compte tout l'historique.

Ces modèles présentent tout de même un inconvénient majeur dans leur utilisation pour l'industrie et des métiers : leur **explicabilité**. Même si certaines techniques d'explicabilité apparaissent, il est toujours **très difficile de justifier les résultats proposés** par ce genre de technique et de les comprendre. Cela est donc souvent difficile à accepter par des métiers qui préféreront avoir des insights sur les éléments expliquant le résultat que le résultat lui-même.

De plus, les modèles de Deep Learning sont très efficaces mais il faut faire attention à ne pas les **utiliser à tort**. En effet, dans cet article "*Do really need Deep learning model for times series forecasting?*" [4.2], de nombreuses architectures complexes de Deep Learning sont comparées avec un modèle Gradient Boosted Regression Tree. Les modèles sont implémentés sur une dizaine de datasets de séries temporelles uni et multivariées. Les résultats montrent **qu'un modèle de Gradient Boosting bien calibré arrive totalement à rivaliser avec les lourdes architectures sur l'ensemble des datasets étudiés**.

Ces modèles peuvent **aussi bien prendre en entrée des données univariées et multivariées**, car il suffit d'ajuster la taille des premières couches du réseau pour correspondre à la taille de nos données. Il n'y a donc pas tellement de distinctions à faire entre les deux types de données ici.

Modélisation

- **Statistiques** : vérifier la stationnarité des données, et utiliser les graphes ACF/PACF pour déterminer les hyperparamètres des modèles
  - AR, MA, ARMA, ARIMA, SARIMA, Exponential Smoothing, …
- **Machine Learning** : transformer des données en tableau et création de features de dates et des valeurs
  - Régression linéaire ou polynomiale, Decision Tree, RandomForest, kNN, XGBoost, Prophet, …
- **Deep Learning** : méthodes rarement privilégiées par manque d'explicabilité
  - RNN, LSTM, NBEATS, Transformers

Famille	Modèles	Nécessite stationnarité	Hyperparam fine-tuning	Données uni/multi	Prend var exogènes	Prend cov passé/ future/statique
Modèle statistique	**ARIMA**	✓	GridSearch	✓/✗	✓	✗/✓/✗
	SARIMA	✓	GridSearch	✓/✗	✓	✗/✓/✗
	VARIMA	✓	GridSearch	✗/✓	✗	✗/✓/✗
	Exponential Smoothing	✗	GridSearch	✓/✗	✗	✗/✗/✗
	Prophet	✗	-	✓/✗	✗	✗/✓/✗
	Croston method	✗	-	✓/✗	✗	✗/✗/✗
Modèle de Machine Learning	**LinearRegression**	✗	-	✓/✓	✓	✓/✓/✓
	RandomForest	✗	-	✓/✓	✓	✓/✓/✓
	DecisionTree	✗	-	✓/✓	✓	✓/✓/✓
	XGBoost	✗	-	✓/✓	✓	✓/✓/✓
Modèle de Deep Learning	**RNN**	✗	GridSearch	✓/✓	✓	✗/✓/✗
	NBEATS	✗	GridSearch	✓/✓	✓	✓/✗/✗
	Transformer	✗	GridSearch	✓/✓	✓	✓/✗/✗

Figure 4.7. – Résumé des modèles et leurs particularités [4.3]

5. Variables exogènes et covariables

Pour améliorer les performances des modèles, il est souvent nécessaire d'ajouter des **données exogènes**, extérieures, qui ont une grande valeur ajoutée. Tous les modèles ne prennent pas en compte ce type de données, mais il est souvent possible d'ajouter ces variables.

Les modèles statistiques présentent souvent une version avec X qui permet d'ajouter des variables exogènes (**ARIMAX, SARIMAX, …**). Dans ces modèles, il est alors possible d'ajouter des colonnes du DataFrame étudié en tant que variables exogènes dans la partie **exog**. Ces variables doivent avoir la même longueur que la variable endogène (la variable à prédire). Il faut cependant aussi les ajouter dans la partie prédiction : **il faut donc les connaître dans le futur**. C'est pour cela qu'il s'agit souvent de valeurs calendaires (jours fériés, vacances, évènements spéciaux, …) que l'on peut donc prédire, ou de données météorologiques. Nous pourrons par exemple utiliser les fonctions ci-dessous pour ce genre de modèles :

```
import statsmodels.api as sm
model = sm.tsa.SARIMAX(endog = df_train['value'],
                       exog = df_train[['holidays', 'covid']],
                       order=(p, d, q), seasonal_order=(P, D, Q, m))
model_fit = model.fit()
prediction = model_fit.forecast(steps = len(df_test),
                                exog = df_test[['holidays', 'covid']])
```

En ce qui concerne les modèles de Machine Learning, nous sommes dans la même situation : les données exogènes peuvent être ajoutées en **features**, mais doivent également être connues sur l'horizon de la prédiction.

Pour les méthodes de Deep Learning, il existe une terminologie propre aux données exogènes : on parle alors de **covariables**. C'est exactement la même chose, mais les covariables sont divisées en 3 types :

- Les **covariables statiques** : elles représentent une valeur qui ne change pas en fonction du temps, mais permet de repérer plusieurs éléments. Par exemple, si un modèle de Deep Learning est utilisé pour prédire les ventes de 3 SKU, les SKU peuvent servir de covariables statiques pour que le modèle comprenne de quel modèle il s'agit. Il s'agit donc souvent **d'identifiants** ;
- Les **covariables passées** : ce sont des données exogènes dont on n'a les **valeurs que dans le passé**. Contrairement aux modèles statistiques ou de Machine Learning, les modèles de Deep Learning peuvent utiliser ce genre de variables pour mieux comprendre le passé et donc le futur. Par exemple, une covariable passée peut être la présence de covid ou non. Cette covariable permettra au modèle de comprendre que la chute des ventes en mars 2020 est due au covid, et donc de ne pas reproduire ce comportement ensuite ;
- Les **covariables futures** : pour ces covariables, nous avons les valeurs dans le passé et dans le futur. Comme pour les modèles précédents, ces variables sont donc souvent des évènements calendaires prévisibles, des informations météorologiques, …

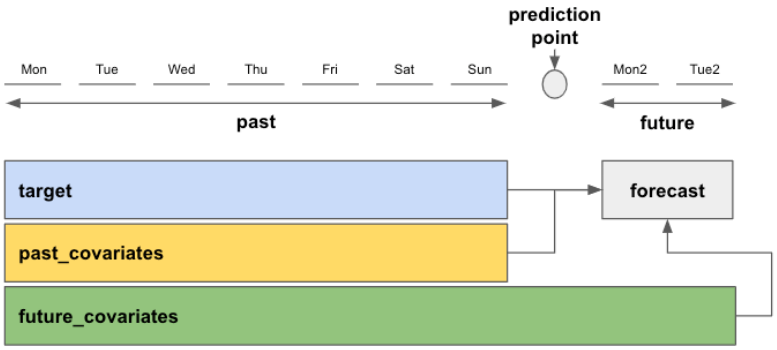


Figure 5.1. – Utilisation de covariables passées et/ou futures

Prenons un exemple pour mieux se rendre compte de l'impact de variables exogènes. Nous allons pour cela regarder le débit d'une rivière au cours du temps [5.1], en ajoutant comme données exogènes les fontes des glaces passées et les précipitations passées et futures. Pour commencer, voilà les données que nous possédons :

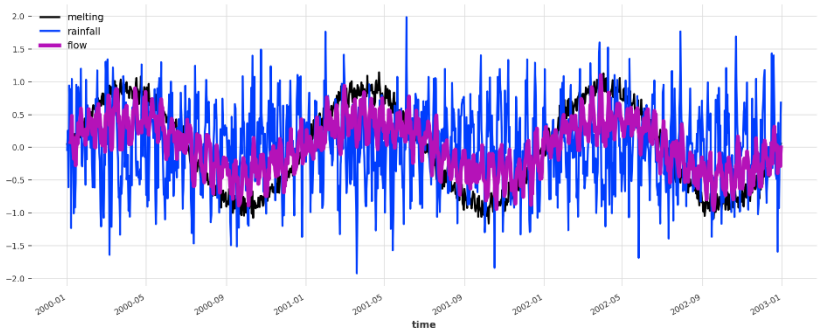


Figure 5.2. – Données du débit du cours d'eau, des précipitations et fonte de glaciers

Commençons tout d'abord par réaliser des prédictions en utilisant que les valeurs passées du débit de la rivière. Nous utiliserons comme modèle un BlockRNN de la librairie Darts. Nous obtenons alors une RMSE de 0,194.

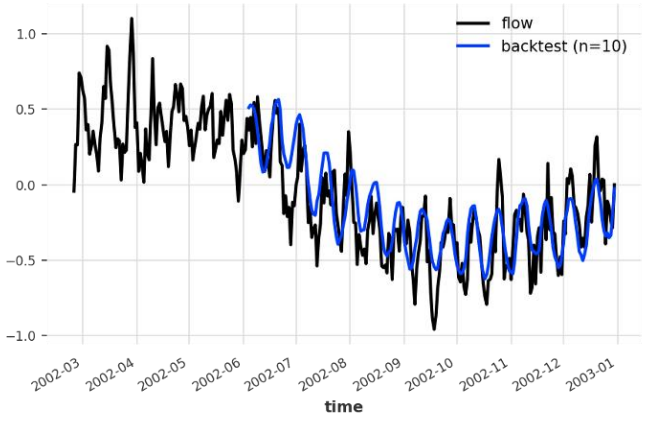


Figure 5.3. – Prédiction sans covariable, RMSE = 0,194

Ajoutons désormais les valeurs passées des fontes des glaciers. On utilisera à nouveau un BlockRNN, qui permet d'ajouter des covariables. Nous constatons que les performances se sont améliorées avec une RMSE de 0,172.

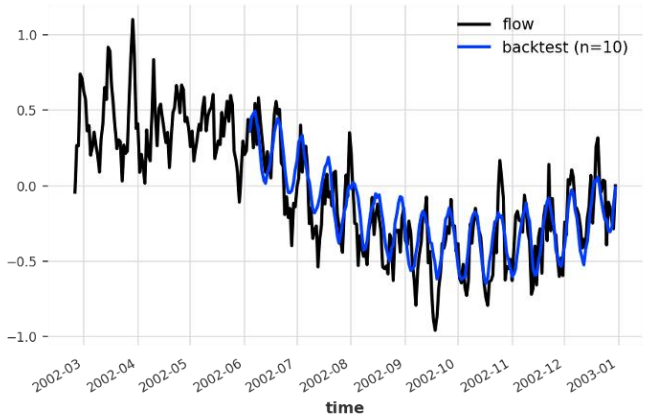


Figure 5.4. – Prédiction avec fonte de glaciers passé, RMSE = 0,172

Enfin, en prenant en compte les valeurs passées des fontes des glaciers et les valeurs passées et futures des précipitations, nous obtenons les meilleures performances avec une RMSE de 0,102. Nous avons utilisé un modèle de régression de Darts appelé RegressionModel, qui permet de prendre en compte des covariables passées et futures.

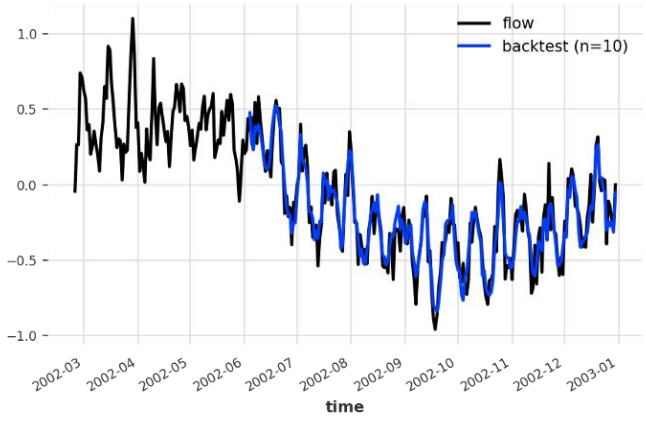


Figure 5.5. – Prédiction avec fonte de glaciers passé et précipitations futures, RMSE = 0,102

Nous constatons donc qu'il est important de prendre en compte des covariables dans les analyses, car elles apportent des valeurs et données importantes, et permettent souvent d'intégrer des connaissances métiers. Il faut cependant bien vérifier si nous avons accès aux valeurs passées et futures, et choisir un modèle en conséquence. Tous les modèles ne permettent pas de prendre en compte tous les types de covariables.

Covariables

Les performances de modèles de séries temporelles **dépendent souvent des données exogènes** que l'on peut ajouter !

Différentes approches selon le type de modèles :

- **Statistiques** : intégration de colonnes de données dans les paramètres exog de modèles ARIMAX, SARIMAX, …
- **Machine Learning** : ajout des données exogènes comme features dans les modèles
- **Deep Learning** : intégration de covariables dans les modèles les prenant en compte :
  - **Cov. statiques** : données pour différencier des séries temporelles
  - **Cov. passées** : données exogènes dont on n'a que les valeurs passées
  - **Cov. futures** : données exogènes dont on a les valeurs passées et futures

6. Evaluation

a. Cross-validation

L'évaluation de la performance des modèles de séries temporelles est très importante, pas seulement d'un point de vue technique pour trouver les meilleurs paramètres, mais aussi d'un point de vue business quand les résultats sont de véritables appuis à la prise de décision. Il est donc **vital de connaître les performances de modèles** pour savoir à quel point nous pouvons **faire confiance** à ses prédictions. Pour cela, on utilisera l'ensemble de test créé pour la cross-validation.

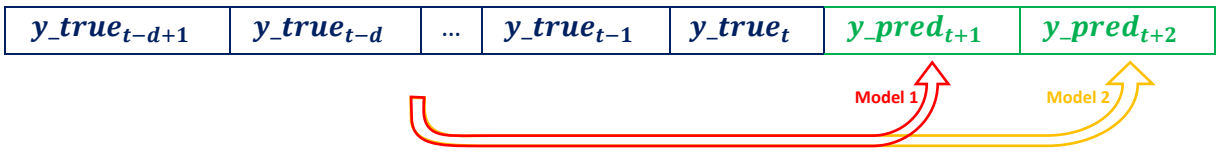
Même si la mesure de la capacité d'un modèle repose sur ses performances, il ne faut pas oublier que d'autres éléments peuvent influencer le choix d'un modèle :

- Complexité et nombre de paramètres
- Temps d'entrainement
- Explicabilité (de la décision, des causes sous-jacentes, …)
- Intervalles de confiance

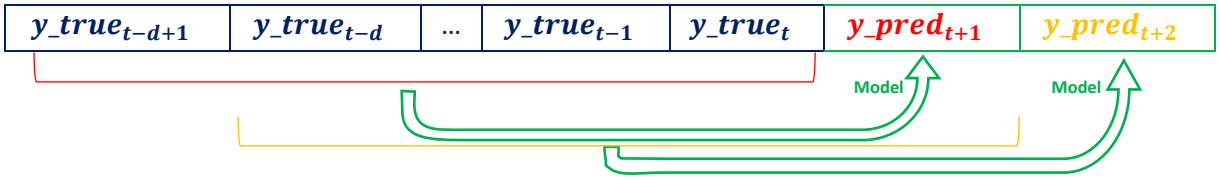
b. Stratégies de prédictions

Les modèles de séries temporelles sont parfois utilisés quand il est nécessaire de prédire plusieurs valeurs dans le futur : on parle de « multi-step time series forecasting » [6.1]. Il existe 4 grandes stratégies de prédictions dans ce cas-là :

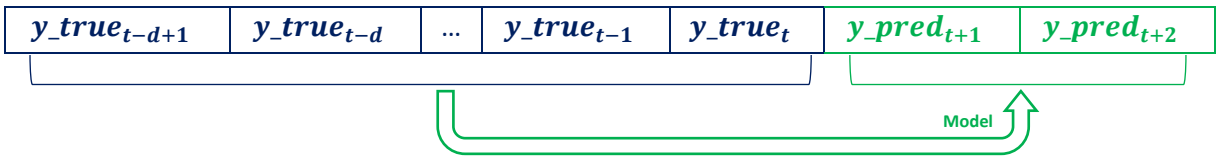
- **Stratégie de prédiction directe** : on va entrainer **un modèle pour chaque prédiction**, reposant sur les d dernières valeurs. Ainsi, un modèle sera développé pour prédire la valeur à t+1, un autre modèle pour prédire la valeur t+2, … Cela ajoute de la complexité et de la maintenance, notamment si le nombre de valeurs à prédire est important. De plus, comme les prédictions sont réalisées de manière indépendante, nous n'avons pas de manière de comprendre leurs relations, alors que cela est au cœur des séries temporelles ;



- **Stratégie de prédiction récursive** : pour la stratégie récursive, les **dernières prédictions peuvent être utilisées** pour la nouvelle prédiction. Par exemple, à t+1, on utilisera les d valeurs réelles à t, t-1, … t-d+1, et à t+2, on utilisera la prédiction de t+1 et les d-1 valeurs réelles à t, t-1, … t-d+2. Ainsi, nous avons toujours d valeurs utilisées pour la prédiction en entrée du modèle. Dans cette méthode, les erreurs de prédiction peuvent s'accumuler et la performance peut donc décliner quand l'horizon de prédiction augmente ;



- **Stratégie de prédiction hybride directe-récursive** : cette stratégie **combine les deux précédentes** pour tirer le meilleur de chacune. Par exemple, il est possible de construire un modèle pour chaque pas de prédiction, tout en utilisant les prédictions des modèles précédents comme input. Cette stratégie permet en général de contourner les limites de chaque stratégie prise séparément ;
- **Stratégie de prédiction à sortie multiple** : cette stratégie consiste en le développement d'un modèle capable de **prédire une séquence entière** d'un seul coup. Ces modèles sont souvent plus complexes, aussi parce qu'ils apprennent les dépendantes entre inputs et outputs, et entre les outputs eux-mêmes. Cela signifie qu'ils seront souvent plus lents, et auront besoin de plus de données pour éviter l'overfitting.



Stratégie de prédiction

4 stratégies pour des prédictions multiples :

- **Stratégie directe** : utiliser différents modèles pour chaque horizon de prédiction
- **Stratégie récursive** : utiliser un seul modèle, et les précédentes prédictions comme inputs
- **Stratégie hybride** : combiner les deux précédentes
- **Stratégie de prédiction à sortie multiple** : prédire une séquence entière d'un seul coup

c. Métriques

Nous allons voir ici les principales métriques [6.2] d'évaluation des modèles de séries temporelles consacrés à la prédiction :

- **R^2** : $R^2 = 1 - \frac{\sum_{i=1}^{n}(y\_true_i - y\_pred_i)^2}{\sum_{i=1}^{n}(y_{true_i} - \overline{y_{true}})^2}$, où $\overline{y_{true}}$ est la moyenne empirique des données

  Le R^2 ne permet pas d'affirmer que le modèle va réaliser de bonnes prédictions, mais il montre si le modèle est un bon ajustement des données et la qualité de cet ajustement par rapport à la moyenne des données. Un R^2 proche de 1 correspondra à un très bon ajustement, alors qu'une valeur proche de 0 ou négative indiquera que la prédiction a des performances moins bonnes que la moyenne.

  Pour l'implémenter, on pourra utiliser la fonction :

```
from sklearn.metrics import r2_score
r2_score(y_true, y_pred)
```

  avec y_true et y_pred des arrays-like de taille (n_samples,) ou (n_samples, n_outputs)

- **Mean Absolute Error (MAE)** : $MAE = \frac{1}{n}\sum_{i=1}^{n}|y\_true_i - y\_pred_i|$

  La MAE correspond à la moyenne de la différence absolue entre les valeurs prédites et celles réelles. Les petites valeurs de MAE correspondent à des bonnes prédictions, alors que des valeurs importantes correspondent à de moins bonnes prédictions. En comparant différents modèles, on va donc choisir celui avec la plus faible MAE. Cependant, il peut être difficile de distinguer de larges et petites erreurs, puisque la MAE n'est pas normalisée.

  Pour l'implémenter, on pourra utiliser la fonction :

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_true, y_pred)
```

  avec y_true et y_pred des arrays-like de taille (n_samples,) ou (n_samples, n_outputs)

- **Mean Absolute Percentage Error (MAPE)** : $MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_{true_i} - y_{pred_i}}{y_{true_i}}\right|$

  La MAPE est la moyenne proportionnelle de la différence absolue entre les valeurs prédites et celles réelles. L'objectif est d'avoir une MAPE faible. Cette métrique a l'avantage d'être proportionnelle donc de pouvoir être utilisée pour comparer des modèles et des valeurs très différentes. Cependant, avec la forme du dénominateur, il faut faire attention aux données qui ont des zéros ou des valeurs extrêmes, car elles peuvent retourner des erreurs ou fausser la métrique. De plus, comme la MAE, la MAPE sous-estime l'impact d'erreurs importantes mais rares avec l'effet de la moyenne.

  Pour l'implémenter, on pourra utiliser la fonction :

```
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_true, y_pred)
```

  avec y_true et y_pred des arrays-like de taille (n_samples,) ou (n_samples, n_outputs)

- **Symmetric Mean Absolute Percentage Error (sMAPE)** : $sMAPE = \frac{100}{n}\sum_{i=1}^{n}\frac{|y_{true_i}-y_{pred_i}|}{(|y_{true_i}|+|y_{pred_i}|)/2}$

  Cette métrique est comme son nom l'indique symétrique, résolvant l'asymétrie de la MAPE qui peut parfois être un problème. En effet, si l'on prédit 10 au lieu de 100, on obtient MAPE = 90% ; et si l'on prédit 100 au lieu de 10, MAPE = 900%. La sMAPE règle ce problème avec une symétrie des valeurs réelles et prédites. Cependant, il n'y a pas de symétrie parfaite et on peut faire face à de l'over-forecasting ou under-forecasting :
  - over-forecasting : on prédit 110 au lieu de 100, sMAPE = 9,52%
  - under-forecasting : on prédit 90 au lieu de 100, sMAPE = 10,53%

  Avec cette nouvelle définition de métrique, nous avons désormais des valeurs contraintes dans l'intervalle 0-200%, ce qui évite les explosions de la métrique.

- **Mean Squared Error (MSE)** : $MSE = \frac{1}{n}\sum_{i=1}^{n}(y\_true_i - y\_pred_i)^2$

  Il s'agit de la moyenne des erreurs quadratiques. Cette métrique peut se décomposer à la fois comme la mesure du biais et de la variance entre les deux séries. L'objectif est d'avoir des modèles avec les valeurs les plus faibles possibles de MSE. Du fait de l'utilisation du carré, cette métrique pénalise beaucoup plus les larges erreurs que les petites. Cette métrique permet de résoudre les problèmes de larges valeurs et de zéros de la MAPE.
  Pour l'implémenter, on pourra utiliser la fonction :

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_true, y_pred)
```

  avec y_true et y_pred des arrays-like de taille (n_samples,) ou (n_samples, n_outputs)

- **Root Mean Squared Error (RMSE)** : $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y\_true_i - y\_pred_i)^2}$

  Une nouvelle fois, des petites valeurs de RMSE indiquent de bonnes performances. Cette métrique, comme la MSE, pénalise davantage les grandes erreurs. L'avantage de la RMSE contrairement à la MSE est qu'elle est dans la même unité que les données, ce qui permet de mieux comparer et interpréter ses valeurs. Il peut être intéressant de comparer la RMSE et la MAE pour voir la nature des erreurs : plus l'écart entre la RMSE et MAE est grand, plus la taille des erreurs est importante. Ce genre de situation peut notamment arriver pour des faibles volumes de données.
  Pour l'implémenter, on pourra utiliser les fonctions :

```
import numpy as np
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_true, y_pred))
```

  avec y_true et y_pred des arrays-like de taille (n_samples,) ou (n_samples, n_outputs)

- **Normalized Root Mean Squared Error (NRMSE)** : $NRMSE = \frac{RMSE}{y\_true_{max}-y\_true_{min}}$ ou $NRMSE = \frac{RMSE}{\overline{y\_true}}$, avec $\overline{y\_true}$ la moyenne des valeurs réelles

  C'est une extension de la RSME qui permet de la normaliser. Ici, les formules montrent les deux principales méthodes de normalisation, avec l'intervalle des valeurs et la moyenne. La NRMSE est souvent utilisée pour comparer différents datasets ou modèles avec des tailles différentes. Les meilleurs modèles auront les valeurs de NRMSE les plus faibles. Cependant, cette métrique peut toujours induire en erreur en traitant de faibles quantités de données.

Pour l'implémenter, on pourra utiliser la fonction :

```
import numpy as np
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_true,y_pred))/(np.max(y_true)-np.min(y_true))
```

avec y_true et y_pred des arrays-like de taille (n_samples,) ou (n_samples, n_outputs)

- **Weighted Mean Absolute Percentage Error (WMAPE)** : $WMPAE = \frac{\sum_{i=1}^{n}|y_{true_i}-y_{pred_i}|}{\sum_{i=1}^{n}|y_{true_i}|}$

  C'est une version de la MAPE dans laquelle les prédictions sont pondérées par les valeurs réelles. Cette métrique évite les problèmes de divergence vers les infinis de la MAPE. Les meilleurs modèles auront les valeurs de WMPAE les plus faibles. Cette métrique peut notamment être utile quand il y a peu de données et que les différentes observations n'ont pas le même niveau de priorité.
  Pour l'implémenter, on pourra utiliser le code :

```
import torch
import torchmetrics
wmape = torchmetrics.WeightedMeanAbsolutePercentageError()
wmape(y_true, y_pred)
```

avec y_true et y_pred des tenseurs

Les métriques

Métrique	Utilisation	Avantages	Inconvénients
R^2	Quand relation entre prédictions et valeurs réelles connue	-Compare les performances à la moyenne	-Donne que des informations sur la qualité de l'ajustement
MAE	Quand peu de valeurs extrêmes	-Interprétable -Facile à comprendre	-Inefficace avec des valeurs extrêmes
MAPE	Quand peu de valeurs extrêmes	-Facile à comprendre -Utile pour comparer des modèles et datasets	-Inefficace avec des valeurs extrêmes
sMAPE	Quand peu de valeurs extrêmes	-A des valeurs limites minimales (0%) et maximales (200%)	-Ne règle pas complètement les problèmes de symétrie
MSE	Pour des valeurs étendues	-Utile pour des valeurs très étendues -Plus sévère avec les grandes valeurs	-Difficilement interprétable
RMSE	Pour des valeurs étendues et interprétabilité	-Utile pour des valeurs très étendues -Plus sévère avec les grandes valeurs -Interprétable car dans la même unité	-Sensible aux outliers
NRMSE	Pour des valeurs étendues et comparer des modèles ou datasets	-Utile pour des valeurs très étendues -Sévère avec les grandes valeurs	- Perte de l'unité de mesure et moins interprétable

d. Critères de sélection de modèles

Il existe aussi des méthodes de sélection de modèles qui ne prennent pas uniquement en compte la performance pour la sélection du meilleur modèle. Par exemple, en comparant différents modèles statistiques, il est important de prendre en compte dans **l'évaluation de la performance** et **du nombre de paramètres**. En effet, le nombre de paramètres a une grande influence sur le temps d'entrainement des modèles. Avec deux modèles presque équivalents, il faudra donc privilégier celui avec le plus petit nombre de paramètres. Il existe pour cela des critères, dont les deux principaux sont :

- **Akaike Information Criterion (AIC)** : $AIC = n.log(\sum_{i=1}^{n}(y\_true_i - y\_pred_i)^2) + 2k$, où n est le nombre d'observations dans la série, et k le nombre de paramètres du modèle
  L'objectif est de sélectionner le modèle avec le score AIC le plus faible possible.
- **Bayesian Information Criterion (BIC)** : $BIC = n.log(\sum_{i=1}^{n}(y\_true_i - y\_pred_i)^2) + k.log(n)$
  L'objectif est de sélectionner le modèle avec le score BIC le plus faible possible.

Pour obtenir ces informations, on pourra par exemple utiliser les fonctions suivantes pour un modèle ARIMA :

```
model = ARIMA(df.value, order=(i, j, k)).fit()
model.aic
model.bic
```

ou la fonction *summary()*, disponible pour tous les modèles statistiques, qui retourne ce genre d'information :

```
model.summary()
```

Par rapport au BIC, **le critère AIC pénalise moins les modèles complexes** : de plus fortes pénalités sont données aux modèles complexes par BIC que par AIC. Ces deux critères sont très intéressants et peuvent être utilisés conjointement pour comparer différents modèles et sélectionner le plus pertinent.

7. Mise en production et maintenance

La maintenance des modèles est une étape cruciale dans la pipeline CI/CD. Un problème souvent rencontré est que le modèle performe correctement sur l'ensemble de développement, validation ou test, mais a des performances mauvaises en production. Cela est particulièrement vrai pour les séries temporelles, où des **changements dans les données peuvent être rapides**. Il y a trois étapes de monitoring de séries temporelles pendant la phase de développement et la phase de production.

Tout d'abord, comme nous l'avons vu dans la partie précédente, il faut **choisir les métriques** les plus adéquates et les **mesurer** en **phase de développement**. Cela permet de voir la qualité des modèles, de les comparer, d'améliorer les performances, et éventuellement de détecter un overfitting entre l'ensemble de test et celui de validation. Pour les méthodes de Deep learning, il est important de **visualiser les courbes d'apprentissage** (learning curves), car c'est sur celles-ci que nous pourrons

voir de l'overfitting avec une bonne performance en entrainement et mauvaise en validation, et mettre en place des solutions pour y remédier comme la **régularisation** ou **l'early stopping**. Cela permet également de suivre la convergence du modèle, et notamment détecter s'il converge vers une solution sous-optimale.

Ensuite, lors de la **phase d'entrainement** du modèle, il est important de suivre les **performances hardware**. Le contrôle des performances hardware pendant l'entrainement d'un modèle DL peut aider à **identifier les goulots d'étranglement** dans le système. Par exemple, le contrôle de l'utilisation de la mémoire du GPU permet de s'assurer que le modèle n'est pas à court de mémoire et que l'entrainement ne s'arrête pas brusquement. Cela permet également de s'assurer que le matériel est utilisé de manière **efficace**.

Enfin, une fois en **production**, il est nécessaire de **suivre les performances du modèle continuellement**. Il est également important de suivre la qualité des données. En suivant régulièrement l'évolution des métriques, en général, il y a uniquement des petites fluctuations qui ne nécessitent pas un réentrainement du modèle. Cependant, si la performance chute brusquement, une raison peut être un **changement dans la distribution des données**. C'est ce qu'on appelle le **drift**. Pour le détecter et y remédier, une solution peut être d'utiliser deux jeux de données pour comparer les performances : un jeu de données avec uniquement des valeurs anciennes qui servira de référence, et un jeu de données où on ajoute des nouvelles valeurs. On peut ensuite mesurer les performances sur les deux jeux de données, ou utiliser d'autres techniques comme des méthodes d'interprétabilité. En effet, en comparant les features les plus importantes, on peut par exemple s'assurer que les features principales sont inchangées entre les deux tests.

Monitoring

Quelques conseils pour gérer la production :

- **Mesurer** et comparer les **performances régulièrement** ;
- Garder un œil sur des **changements dans les caractéristiques statistiques** des données ;
- Utiliser le **feature engineering** ou **data augmentation** pour améliorer la robustesse ;
- **Réentraîner le modèle** sur des données récentes avec la nouvelle distribution ;
- Créer des **alertes** quand le drift est détecté.

8. Retours d'expérience

Dans cette partie, nous allons revenir sur les **grandes étapes d'un projet du DataLab**, à la fois pour mettre plus en avant la **méthodologie générale**, et pour retenir des **points d'attention**. Le projet étudié est celui de prévision de la demande mondiale pour chaque SKU permanents (PES) pour chaque semestre. Suivons les étapes principales du projet :

- **Données** : pour représenter la demande, nous avons utiliser les **données des ventes**, que l'on peut retrouver dans la table FACT_SALES_TRANSACTIONS. Nous avons les ventes de chaque jour pour chaque SKU depuis le 01/01/2017. A ces données, nous pouvons ajouter des

données **d'informations sur les articles** provenant notamment de DIM_ARTICLE (nom des produits, métiers, genre, famille, classification PES). Enfin, d'autres données peuvent sembler pertinentes pour expliquer les ventes et nous allons les ajouter par import de fichiers excels :

- Données d'ouverture de magasins ;
- Jours fériés ;
- Jours particuliers (fête des mères, fête des pères, black Friday, nouvel an chinois, saint valentin) ;
- Soldes d'été et d'hiver en France ;
- Dates de la fashion week et défilé haute couture ;
- Dates des confinements covid en France.

Avant de passer à l'étape d'exploration, il est nécessaire de regarder la **quantité de données** que nous avons. Ainsi, nous avons un **historique** de ventes depuis 2017, ce qui est suffisant pour nos modélisations. Il ne faut pas oublier qu'un historique limité entrainera une limitation dans nos performances de prédiction.

- **Exploration** : dans cette partie, nous avons regardé **l'évolution des données générales, les tendances et périodicités** :

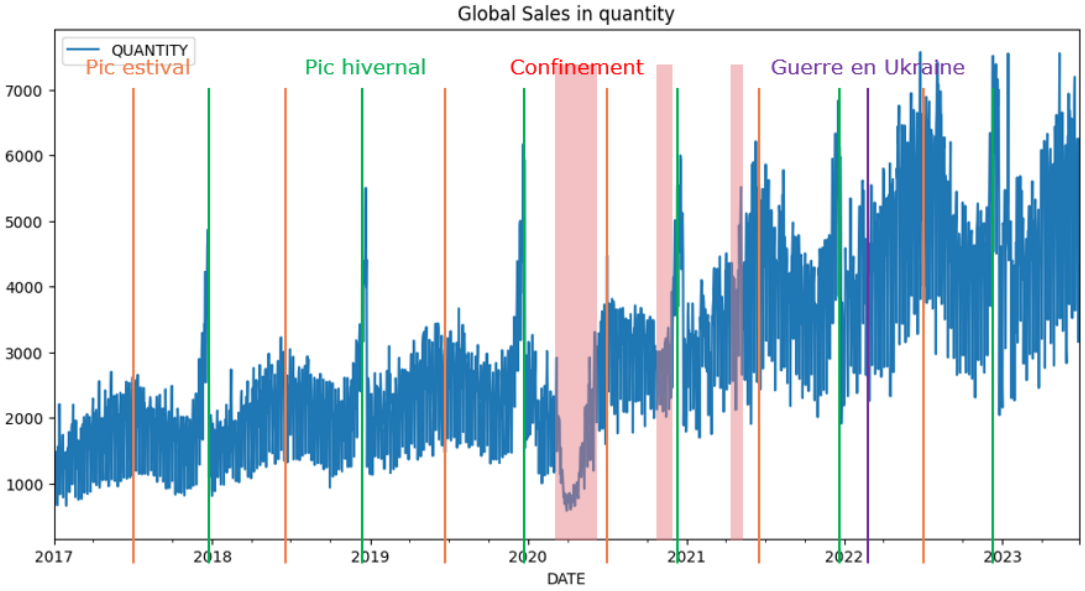


Figure 8.1. – Ventes globales de tous les SKU PES

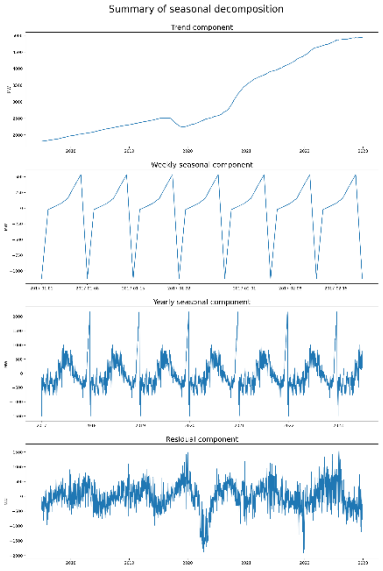


Figure 8.2. – Etude de la saisonnalité des données

Avec cette brève décomposition en tendance-saisonnalité, nous voyons apparaître une **tendance haussière**, ainsi qu'une **double saisonnalité hebdomadaire et annuelle**. Il sera donc important de prendre en compte ces informations dans la modélisation.

Ensuite, il a fallu **conserver les données pertinentes** et **nettoyer les données**. Un tri a donc été fait pour ne conserver que les SKU avec suffisamment d'historique de vente, et classés PES depuis un certain moment. N'ayant pas forcément de ventes chaque jour, il a également fallu **ajouter des lignes** les jours où il n'y avait pas de ventes pour que le modèle explique également ces données-là.

- **Priorisation** : à ce stade, il y avait 2052 SKU à traiter dans les données. Cela peut paraître beaucoup, surtout s'il est nécessaire de faire un modèle par SKU. De plus, certains SKU, bien que classifiés PES, ne représentent pas une part suffisante des ventes. Nous avons donc réalisé **des graphes de Pareto** pour ne sélectionner que les SKU représentant 80% des quantités vendues :

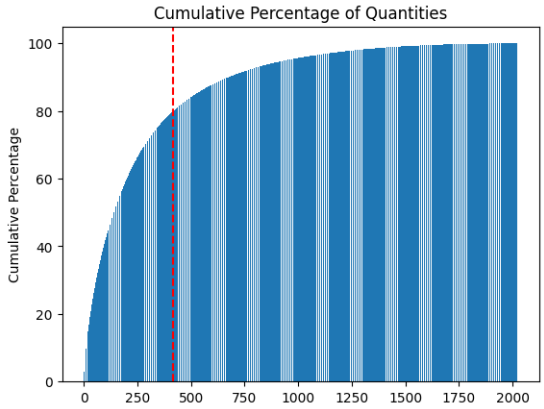


Figure 8.3. – Pareto du pourcentage cumulé des quantités vendues

On se rend compte que cette priorisation était très importante, car **20,7% des SKU représentent 80% des ventes**. En croisant ces valeurs avec une priorisation en CA, nous avons retenu 237 SKU prioritaires sur lesquels nous avons entrainé nos modèles, avant d'essayer de généraliser à tous les SKU.

- **Métriques** : Le choix des métriques étudiés était fondamental. Nous avons tout d'abord retenu la **MAE**, qui permet de se rendre compte de l'erreur, et ensuite de mesurer ce qu'une erreur peut coûter en termes de stockage par exemple. Nous avons mesuré la MAE pour chaque SKU, et pris ensuite la **moyenne et le quartile 90** pour avoir une vision assez complète de la répartition. Le problème de cette métrique est que nous ne pouvions pas comparer des erreurs pour deux SKU avec des ordres de grandeur de ventes différentes. Pour cela, nous avons voulu utiliser la MAPE, mais avec des valeurs de ventes proches de 0, celle-ci exposait. Nous avons donc utilisé la **sMAPE**, et avons pris comme pour la MAE la **moyenne et le quartile 90**. Enfin, pour avoir une idée des performances de notre modélisation par rapport à une moyenne, nous avons ajouté **le coefficient R^2.**
- **Cadre** : Afin de coller à ce que font les métiers, nous avions un certain cahier des charges à respecter : prédictions à la maille SKU, pour un semestre, avec un mois d'avance (en novembre pour S1 suivant, en mai pour S2 suivant). Dans un premier temps, nous avons essayé de faire des **prédictions journalières**, mais les résultats n'étaient pas concluants. Nous sommes donc passé à des **prédictions mensuelles**, comme le font les métiers, ce qui a amélioré les performances.
- **Normalisation** : Avant la modélisation, nous avons normalisé les données. Cela permet d'éviter d'avoir des valeurs extrêmes, et améliore les performances des modèles.
- **Méthodologie** : Dans un premier temps, nous avons réalisé des prédictions journalières. Nous avons d'abord essayé des **modèles statistiques univariés** (ARIMA, SARIMA, …) **sans données exogènes, puis avec** en essayant d'ajouter le plus de données pertinentes possibles. Ensuite, nous avons essayé des **méthodes multivariées** (VARMA), où chaque SKU est une série, et elles

servent toutes à prédire les autres. Ensuite, nous sommes passés à des **méthodes de Machine Learning**, en ajoutant comme features les données exogènes pertinentes. Les résultats n'étant pas convaincants, nous avons ensuite réalisé les prédictions à la maile mensuelle. Pour cela, nous avons repris les mêmes modèles avec deux stratégies : agréger les données mensuellement et réaliser la modélisation ensuite, ou conserver un modèle journalier pour ensuite agréger par mois. C'est cette dernière option qui a été la meilleure.

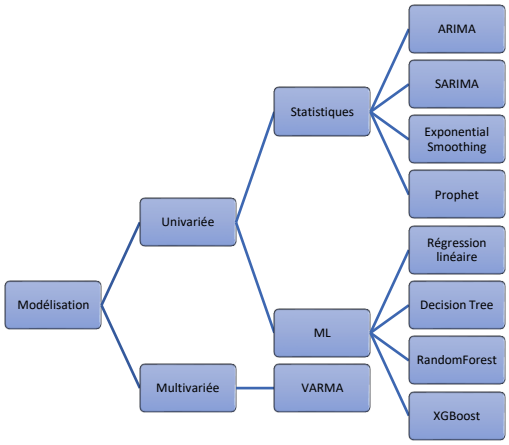


Figure 8.4. – Méthodologie utilisée et modèles essayés

- **Modélisation** : Pour chaque modèle, nous avons tout d'abord **préparé les données** sous le format nécessaire, puis **recherché les meilleurs hyperparamètres**. Cela est très couteux et ne pouvait pas être fait pour chaque SKU. Nous avons donc **agrégé tous les SKU** dans une série en **normalisant** les données pour que chaque SKU ait le même poids, et nous avons ensuite trouver les meilleurs hyperparamètres sur cette série. Selon les modèles, nous avons soit utilisé les fonctions de recherche d'hyperparamètres déjà implémentées, soit effectué cette recherche manuellement. Nous avons ensuite **généralisé ces hyperparamètres à chaque SKU**. En ce qui concerne la saisonnalité, certains modèles ne permettaient pas d'utiliser une saisonnalité annuelle (m=365) car les temps de convergence devenaient trop importants, et nous avons donc utilisé une saisonnalité hebdomadaire. Pour trouver les meilleures features, notamment pour les modèles avec données exogènes ou les modèles de Machine Learning, nous avons tout d'abord réalisé une **matrice de corrélation** pour voir celles qui étaient les plus corrélées à la quantité vendue, et celles qui étaient très corrélées entre elles. Après un premier tri, nous avons essayé de les **ajouter progressivement** pour voir si les performances étaient améliorées.

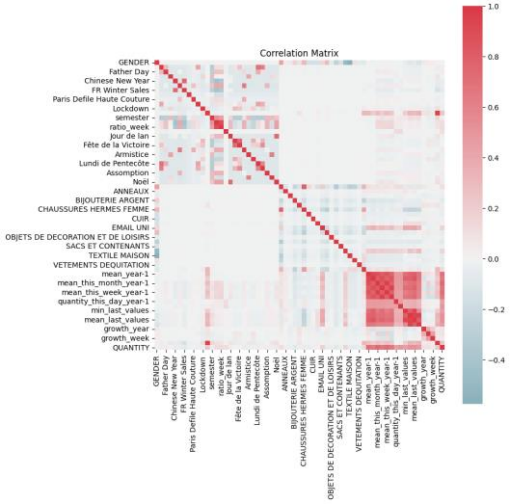


Figure 8.5. – Matrice de corrélation de nos features

- **Résultats** : Finalement, notre meilleur modèle a été un modèle **XGBoost général entrainé sur toutes les données,** et qui réalisait des **prédictions journalières**. Nous avons ensuite **agrégé** ces prédictions **de manière mensuelle**. Comme **features** nous utilisions : la moyenne de l'année précédente, la quantité vendue ce jour l'année précédente et celle d'avant, la moyenne des dernières ventes connues, la croissance de ce mois l'année dernière, la croissance de cette semaine l'année dernière, la date des soldes d'été et d'hiver, la date du défilé haute couture de Paris, la fête du travail, Noël et les éléments classifiés « Sacs et contenants ». Nous avons finalement atteint une sM**APE moyenne de 35% sur les SKU priorisés.**

9. Conclusion

Ce rapport a **exploré** avec différents niveaux de profondeur **divers aspects liés à l'analyse des séries temporelles**, du cadrage initial à la mise en production et à la maintenance continue. Nous avons abordé les différentes étapes clés de ce processus, y compris l'exploration des données, le pre-processing, la modélisation, l'utilisation de données exogènes, l'évaluation des performances, ainsi que la mise en production. En comprenant les méthodes et les bonnes pratiques que nous avons discutées, ce rapport à vocation à **aider les data scientists juniors ou confirmés** dans leur étude de séries temporelles.

Au-delà du pre-processing et de la modélisation, deux points semblent particulièrement cruciaux pour avoir de bons modèles de séries temporelles. Tout d'abord, **l'intégration de données exogènes** dans le processus de modélisation a montré son importance pour améliorer la précision des prévisions, en tenant compte des facteurs externes qui peuvent influencer les séries temporelles. D'un autre côté, **l'évaluation** rigoureuse des performances des modèles est également cruciale pour garantir leur pertinence dans des applications réelles. Le **choix de métriques pertinentes** est donc une étape à ne pas négliger, et il ne faut pas hésiter à créer ses propres métriques selon le cas étudié.

Enfin, une « cheat sheet » est fourni pour résumer les points clés à retenir lors de l'analyse des séries temporelles, offrant ainsi une ressource pratique pour retrouver une information rapidement.

En résumé, ce rapport vise à servir de **premier guide** pour ceux qui s'intéressent à l'analyse des séries temporelles. Les sources de la bibliographie ou des recherches plus pointues permettront ensuite de cibler une étape et trouver des informations plus détaillées. Les séries temporelles sont un domaine en constante évolution, et en continuant à explorer et à expérimenter, nous pouvons exploiter leur potentiel pour résoudre des problèmes complexes et prendre des décisions informées.

10. Fiches recap'

Un projet data

Echanges permanents avec le métier

Travailler avec les dates

- **Importer les librairies** : from datetime import datetime, timedelta
- **Date du jour/ jour et heure actuel** : datetime.date.today() / datetime.datetime.now()
- **Ouvrir un csv avec des dates** : pd.read_csv(path, parse_dates = ['date'])
- **Changer le format des dates** : pd.to_datetime(df['date'], dayfirst= True), pd.to_datetime(df['date'], format ="%Y-%m-%d %H:%M:%S")
- **Extraire année/mois/jour** : df['date'].dt.year/df['date'].dt.month/df['date'@].dt.day
- **Ajouter une période** : df['date'] + timedelta(days=10)

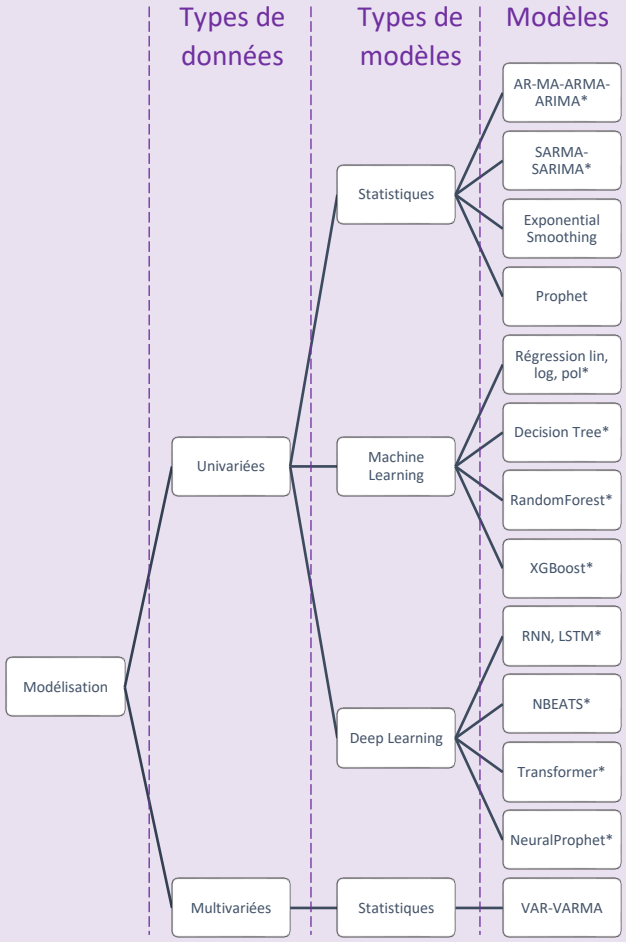
Pre-processing

- **Valeurs manquantes :** imputation avec df['value'].interpolate(method=) ou méthode d'imputation comme kNN, moving average, …
- **Filtrage** : Moyennes mobiles, Transformée de Fourier, Passe-bas, Matching filter, Exponential smoothing, Kalman filter, Savitzky-Golay
- **Outliers** : Création d'intervalles de confiance, Analyse de résidus, Méthodes de classification (Isolation Forest), Méthodes de prédiction, Méthodes de clustering
- **Normalisation :** from sklearn.preprocessing import MinMaxScaler, StandardScaler
- **Train test split :** conserver les valeurs de train antérieures au test

Famille	Modèles	Nécessite stationnarité	Hyperparam fine-tuning	Données uni/multi	Prend var exogènes	Prend cov passé/ future/statique
Modèle statistique	**ARIMA**	✓	GridSearch	✓/✗	✓	✗/✓/✗
	SARIMA	✓	GridSearch	✓/✗	✓	✗/✓/✗
	VARIMA	✓	GridSearch	✗/✓	✗	✗/✓/✗
	Exponential Smoothing	✗	GridSearch	✓/✗	✗	✗/✗/✗
	Prophet	✗	-	✓/✗	✗	✗/✓/✗
	Croston method	✗	-	✓/✗	✗	✗/✗/✗
	LinearRegression	✗	-	✓/✓	✓	✓/✓/✓
	RandomForest	✗	-	✓/✓	✓	✓/✓/✓

Modèle de Machine Learning	**DecisionTree**	✗	-	✓/✓	✓	✓/✓/✓
	XGBoost	✗	-	✓/✓	✓	✓/✓/✓
Modèle de Deep Learning	**RNN**	✗	GridSearch	✓/✓	✓	✗/✓/✗
	NBEATS	✗	GridSearch	✓/✓	✓	✓/✗/✗
	Transformer	✗	GridSearch	✓/✓	✓	✓/✗/✗

Méthodologie



*modèle pouvant prendre des données exogènes

Covariables

- **Covariables statiques :** données pour différencier des séries temporelles
- **Covariables passées :** données exogènes dont on n'a que les valeurs passées
- **Covariables futures :** données exogènes dont on a les valeurs passées et futures

Stratégie d'évaluation

4 stratégies pour des prédictions multiples :

- **Stratégie directe :** utiliser différents modèles pour chaque horizon de prédiction
- **Stratégie récursive :** utiliser un seul modèle, et les précédentes prédictions comme inputs
- **Stratégie hybride :** combiner les deux précédentes
- **Stratégie de prédiction à sortie multiple :** prédire une séquence entière d'un seul coup

Métriques

- **R^2 :** pour comparer les performances à la moyenne ;
- **MAE :** écart relatif moyen, + interprétabilité en donnant l'erreur moyenne
- **MAPE :** écart relatif en pourcentage, + interprétabilité en comparant des modèles
- **sMAPE :** symétrique, pour limiter des explosions de la MAPE avec des faibles valeurs
- **MSE :** écart quadratique moyen
- **RMSE :** racine carrée de l'écart relatif moyen, + interprétabilité car conserve l'unité

Critères de sélection de modèle : BIC, AIC

Cas d'usages

- **Prévision de la demande** pour optimiser l'inventaire, les stocks, les ressources, …
- **Optimisation de la supply chain** avec gestion des stocks, du personnel, …
- **Optimisation des prix** selon la demande, pays, …
- **Détection de fraudes**
- **Détection d'anomalies** (chaîne de production, produits, données, …)
- **Analyse des tendances** saisonnières pour guider les créations, lancements et prix
- **Analyse de sentiments** sur les réseaux sociaux
- **Evaluation des performances de ventes**

11. Bibliographie

[1.1] https://en.wikipedia.org/wiki/Time_series

[2.1] Towards Data Science, Things That You Should Check Before Creating Univariate Time Series Model, Raden Aurelius Andhika Viadinugroho, 2021, https://towardsdatascience.com/things-that-you-should-check-before-creating-univariate-time-series-model-ba5fe381f68e

[2.2] Medium, Discovering the Keys to Solving for Data Quality Analysis in Streaming Time Series Datasets, Chris Herrera, 2018, https://medium.com/hashmapinc/discovering-the-keys-to-solving-for-data-quality-analysis-in-streaming-time-series-datasets-8d8780fa7ecb

[2.3] Towards Data Science, How to Do an EDA for Time-Series, Fabiana Clemente, 2022, https://towardsdatascience.com/how-to-do-an-eda-for-time-series-cbb92b3b1913

[3.1] Towards Data Science, 7 Ways to Handle Missing Values in Machine Learning, Satyam Kumar, 2020, https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e

[3.2] Towards Data Science, 4 Techniques to Handle Missing values in Time Series Data, Satyam Kumar, 2022, https://towardsdatascience.com/4-techniques-to-handle-missing-values-in-time-series-data-c3568589b5a8

[3.3] Medium, Pre-processing of Time Series Data, Shashank Gupta, 2022, https://medium.com/enjoy-algorithm/pre-processing-of-time-series-data-c50f8a3e7a98

[3.4] NeptuneAI MLOps Blog, Anomaly Detection in Time Series, Aayush Bajaj, 2023, https://neptune.ai/blog/anomaly-detection-in-time-series

[3.5] Medium, Cross Validation in Time Series, Soumya Shrivastava, 2020, https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4

[3.6] Analytics India Mag, How to make a time series stationary?, Sourabh Mehta, 2022, https://analyticsindiamag.com/how-to-make-a-time-series-stationary/

[4.1] Medium, Forecasting Intermittent Demand with the Croston Model, Nicolas Vandeput, 2019, https://towardsdatascience.com/croston-forecast-model-for-intermittent-demand-360287a17f5f

[4.2] arXiv, Do We Really Need Deep Learning Models for Time Series Forecasting?, Shereen Elsayed and Daniela Thyssens and Ahmed Rashed and Hadi Samer Jomaa and Lars Schmidt-Thieme, 2021, 2101.02118

[4.3] Darts, https://unit8co.github.io/darts/

[5.1] Unit8, Time Series Forecasting Using Past and Future External Data with Darts, Julien Herzen, https://unit8.com/resources/time-series-forecasting-using-past-and-future-external-data-with-darts/

[6.1] Machine Learning Mastery, 4 Strategies for Multi-Step Time Series Forecasting, Jason Brownlee, 2017, https://machinelearningmastery.com/multi-step-time-series-forecasting/

[6.2] Analytics India Mag, A Guide to Different Evaluation Metrics for Time Series Forecasting Models, Vijaysinh Lendave, 2021, https://analyticsindiamag.com/a-guide-to-different-evaluation-metrics-for-time-series-forecasting-models/