



PROJET DE STATISTIQUE APPLIQUÉE

OUTIL DYNAMIQUE DE SUIVI DES PRÉVISIONS DU TRAFIC AÉRIEN

Claire HE
Victor HUYNH
Solène BLASCO LOPEZ
Antonin FALHER

Référent des Aéroports de Paris : Raphaël BOUDRA
Correspondant Ensaïe : Jean-Michel ZAKOIAN

21 mai 2021

Table des matières

Introduction	2
1 Description des données	3
1.1 Présentation des données à notre disposition	3
1.1.1 Données de trafic fournies par les Aéroports de Paris	3
1.1.2 Autres données à notre disposition	4
1.2 Visualisation des données retenues pour notre étude	4
1.2.1 Description des données retenues pour notre étude	4
1.2.2 Observation des tendances et des saisonsnalités du trafic aérien	4
2 Revue de littérature	6
3 Approche empirique	8
3.1 Modèle ARIMA	8
3.1.1 Définition du modèle	8
3.1.2 Stationnarisation de notre série	8
3.1.3 Identification des ordres p_{max} et q_{max}	8
3.1.4 Estimation, test de validité des modèles possibles et choix du meilleur modèle	9
3.2 Modèle SARIMA	9
3.2.1 Définition du modèle	9
3.2.2 Sélection des ordres optimaux	9
3.3 Modèle non paramétrique	10
3.3.1 Définition du modèle	10
3.3.2 Stationnarité	11
3.3.3 Choix des paramètres	11
3.3.4 Intervalles de confiance	11
3.4 Modèle LASSO	12
3.4.1 Définition du modèle	12
3.4.2 Intervalle de prédiction	12
4 Format du code et outil PowerBi	14
5 Résultats	15
5.1 Prévisions du trafic aérien par chacun des modèles implémentés	15
5.1.1 Modèles ARIMA/SARIMA	15
5.1.2 Modèle Non-Paramétrique	16
5.1.3 Modèle Lasso	17
5.2 Comparaison des différents modèles	18
5.2.1 Performances des modèles	18
5.2.2 Temps d'exécution	19
Conclusion	20
Références	21
A Figures et Outil PowerBi	22
B Code des différents modèles	36

Introduction

Ce projet de Statistique Appliquée est proposé et encadré par les Aéroports de Paris. Être en mesure de prédire avec le plus de précision possible le trafic aérien et la fréquentation des aéroports est crucial pour une telle entreprise. L'enjeu est de pouvoir anticiper au mieux la demande afin d'ajuster au mieux l'offre de sièges. Mieux connaître les besoins permet aux aéroports d'améliorer leurs performances et leur efficacité opérationnelle.

L'étude proposée par les Aéroports de Paris a pour objectif d'élaborer plusieurs modèles efficients de prévision du Trafic Aérien à une échelle agrégée. L'entreprise souhaiterait prédire le flux de passagers sur les différents faisceaux qu'elle dessert, et à un niveau plus général que le vol à vol (journalier, mensuel, etc.). Les enjeux sont multiples mais avant tout opérationnels : les Aéroports de Paris souhaitent que les prédictions réalisées soient visualisables sous Power BI. L'outil développé doit être dynamique, c'est-à-dire permettre de mettre à jour de manière interactive les différents indicateurs de suivi désirés en modifiant des paramètres définis par l'utilisateur.

Notre étude a donc été réalisée autour de la problématique suivante :

Comment prédire au mieux le trafic aérien à différents horizons de prévision sur chacun des faisceaux desservis par les Aéroports de Paris ?

Pour mener à bien cette étude, les Aéroports de Paris ont mis à notre disposition des données quotidiennes de trafic aérien dans l'aéroport d'Orly sur plusieurs années ainsi que l'exemple des prédictions que leur entreprise avait réalisées au préalable sur une année. Les prédictions attendues doivent avant tout être réalisées à partir du trafic aérien antérieur, mais nous avons également pu intégrer à certains modèles des données calendaires fournies par notre référent. Notre référent a également mis à notre disposition le travail effectué lors de l'étude qu'il avait proposée l'année dernière, qui visait à implémenter des modèles de prédition du trafic aérien journalier entre deux destinations précises.

Dans un premier temps, nous avons découvert les données et nous les avons visualisées pour comprendre au mieux les spécificités du trafic aérien, en particulier sur chacun des faisceaux desservis par les Aéroports de Paris. Nous avons réalisé par la suite une revue de littérature pour connaître le contexte théorique dans lequel s'inscrivaient nos données, et pour identifier des modèles pertinents à implémenter. Enfin, nous avons implémenté sous Python les modèles retenus sur nos données, avant de visualiser leurs prévisions sous PowerBi. Nous avons commenté les résultats obtenus et comparé les différents modèles de prédition du trafic aérien.

Organisation du travail

Pour réaliser cette étude, chacun a participé aux différentes étapes du projet, réalisées successivement comme présentées. Nous mettions en commun notre travail via un dépôt Github et des réunions fréquentes entre nous. Nous avions également des points avec notre référent à qui nous présentions régulièrement l'avancement de notre projet.

Plus particulièrement, Victor Huynh et Antonin Falher ont travaillé à l'implémentation des modèles ARIMA et SARIMA, tandis que Claire He et Solène Blasco Lopez se sont concentrées sur les modèles non-paramétrique et Lasso.

Le langage informatique utilisé pour l'implémentation des modèles est Python. Nous avons également mis au point un outil de visualisation et de comparaison des performances des modèles dans PowerBi, où la plupart des visuels peuvent être réalisés à partir de codes Python sous-jacents.

1 Description des données

La première étape de notre travail a été de mettre en forme puis de visualiser les données fournies par les Aéroports de Paris, afin de découvrir et comprendre les spécificités du trafic aérien.

1.1 Présentation des données à notre disposition

1.1.1 Données de trafic fournies par les Aéroports de Paris

Pour mener à bien notre étude, les Aéroports de Paris ont mis à notre disposition deux types de données :

- Des historiques détaillés du trafic aérien entre 2008 et 2018 qui recensent tous les vols ayant décollé ou atterri dans les aéroports de Paris entre ces dates.
- Des estimations qui avaient été réalisées par les Aéroports de Paris entre 2015 et 2017. Chaque semaine, l'entreprise a regardé l'historique des vols précédents, et a prédit un taux de remplissage pour chaque jour, suivant le faisceau, le type de mouvement et l'aérogare. Nous avons concaténé ces bases en un seul fichier `fqms_concat`.

Suite à différents problèmes (détaillés dans notre note de mi-parcours) rencontrés initialement pour faire correspondre les réalisations avec les prédictions, notre référent nous a fourni une base `DATABASE` qui contient les réalisations au vol à vol entre 2008 à 2016, à partir de laquelle il était possible de faire correspondre les prédictions pour l'année 2016. Nous y avons ajouté l'estimation du nombre de sièges moyen pour chaque vol, en créant des tables de référence par type d'avion à partir de cette information, qui était présente dans la base `fqms_concat`, mais pas dans les deux autres. Cela nous a également permis de calculer une approximation des taux de remplissages réalisés pour chaque vol.

La table qui suit résume les différentes variables d'intérêt que nous avions alors à notre disposition :

Variables	Interprétations
Date	Date du vol
Faisceau	Faisceau du vol parmi 5 modalités (National, International, Autre UE, Schengen et Dom-Tom)
ArrDep	Type de mouvement parmi 2 modalités (Arrivée ou Départ)
Cie et Code Cie	Informations sur la compagnie pour laquelle chaque vol est réalisé (nom complet et code de 3 lettres)
Sièges CorrectionsICI	Nombre moyen de sièges prévus sur chaque vol
PAX	Nombre de passagers réalisés pour chaque vol
PAX_FQM	Nombre de passagers estimés par les Aéroports de Paris pour chaque vol (non vide seulement pour 2016)
Coeff_Rempl	Taux de remplissage réalisé pour chaque vol (estimation à partir des estimations du nombre de sièges)
Coeff_Rempl_FQM	Taux de remplissage estimé par les Aéroports de Paris pour chaque vol (non vide seulement pour 2016)
Date Equiv	Date équivalente utilisée par les Aéroports de Paris pour réaliser leurs prédictions

TABLE 1 – Variables d'intérêt à notre disposition pour chaque vol

Remarque : nous avons conservé les noms de variables établis par l'entreprise et avons fait au préalable le moins de modifications préalables sur les bases de données, afin que l'entreprise puisse facilement mettre à jour les données et réutiliser notre travail par la suite.

1.1.2 Autres données à notre disposition

Les Aéroports de Paris ont également mis à notre disposition une base **Calendrier** qui contient les informations calendaires pour chaque date entre 2007 et 2024. Cette base de données possède notamment une colonne **JourSem** qui contient des chiffres de 1 à 7 correspondant au jour de la semaine, ainsi que des colonnes pour chaque périodes de vacances de chaque zones, et pour chaque jour de la semaine potentiellement férié. Ces colonnes contiennent des entiers, qui indiquent le cas échéant la position de la date considérée dans la période de vacances, ou la position de la date considérée dans la liste des jours fériés de l'année. Cela est plus précis que de simples indicatrices, et permet d'affiner nos prédictions, en prenant par exemple en compte le fait qu'il y a une période de charge en début de vacances, et de décharge en fin de vacances.

1.2 Visualisation des données retenues pour notre étude

1.2.1 Description des données retenues pour notre étude

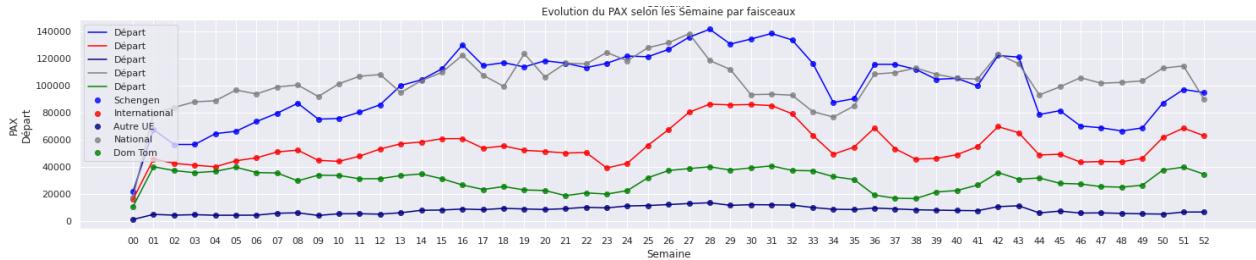
Les prédictions des Aéroports de Paris fournies par notre encadrant ne concernent que les vols survenus dans l'aéroport d'Orly. Ces données ne sont complètes que pour l'année 2016 (pour les autres années, certaines prédictions sont lacunaires ou manquantes).

Dans le cadre de notre étude, nous nous sommes donc restreints au trafic aérien dans l'aéroport d'Orly. Comme nous cherchons à prédire le trafic aérien de manière journalière, nous avons fait une agrégation journalière sur notre base qui était au vol à vol, en distinguant les faisceaux et le type de mouvement. De plus, nous nous sommes concentrés sur la prédition du nombre de passagers, car les taux de remplissages que nous avons approchés ne sont pas suffisamment précis.

Pour des raisons pratiques, nous avons choisi de manière arbitraire d'étudier uniquement les départs, les spécificités du trafic aérien sur chacun des faisceaux étant très similaires entre les départs et les arrivées. Les Aéroports de Paris distinguent également l'aérogare lorsqu'ils réalisent leurs prédictions, mais nous avons choisi de ne pas le faire, car le nombre de prédictions à réaliser aurait été trop grand et nos programmes étaient déjà très longs pour traiter tous les faisceaux et les types de mouvement différents.

Une fois les données récupérées et mises en forme, nous les avons visualisées sur chacun des faisceaux desservis, afin de comprendre les spécificités du trafic aérien. Notons que distinguer les faisceaux dans les prédictions est pertinent, car comme en témoigne la figure 1, le volume de passagers et les tendances diffèrent ostensiblement.

FIGURE 1 – Observations des nombres de passagers hebdomadaires par faisceau, pour les départs de 2016



1.2.2 Observation des tendances et des saisonnalités du trafic aérien

Afin de pouvoir soumettre nos données à une analyse de séries temporelles, nous avons cherché à mettre en avant les éventuelles tendances et saisonnalités présentes dans les taux de remplissages et les nombres de

passagers journaliers en distinguant chacun des faisceaux et des types de mouvement. Il est en effet primordial de bien comprendre nos données pour chercher et choisir par la suite les modèles les mieux adaptés.

En ce qui concerne le nombre de passagers réalisés, voici les principales caractéristiques du trafic aérien mises en avant en annexe dans la figure 8 :

- Alors qu'une tendance globale à la hausse au fil des années peut être observée sur les séries hebdomadaires bleues pour les faisceaux Schengen et Autre UE, d'autres faisceaux ne semblent pas présenter de tendance globale, comme par exemple le faisceau International.
- Dans la colonne centrale, on remarque la présence de saisonnalités annuelles plus ou moins marquées selon les faisceaux, avec notamment une hausse commune du nombre de passagers réalisés en été et en toute fin d'année.
- La saisonnalité mensuelle, non représentée dans la figure 8, est très peu marquée sur chacun des faisceaux, et a été négligée dans nos modélisations.
- Enfin, on peut observer une saisonnalité hebdomadaire dans les séries vertes qui correspondent aux écarts de nombre de passagers journaliers entre le jour considéré et le même jour l'année précédente. En effet, le nombre de passagers est en général bien plus important les vendredis qu'en début de semaine.

En ce qui concerne les coefficients de remplissage moyens réalisés, leurs caractéristiques sont présentées de manière analogue dans la figure 9. Rappelons qu'ils n'étaient pas disponibles initialement dans la base fournie par les Aéroports de Paris. Ils ont été approchés en attribuant à chaque vol une estimation du nombre de sièges disponibles qui correspond en fait au nombre moyen de sièges disponibles sur le type d'avion correspondant.

On constate dans la figure 9 que pour un même faisceau, les coefficients de remplissages peuvent avoir des tendances qui diffèrent de celles du nombre de passagers. On observe également des saisonnalités différentes et moins lisibles. Cela confirme notre choix de nous concentrer sur la prédiction des nombres de passagers.

2 Revue de littérature

En visualisant nos données, nous avons mis en avant les spécificités du trafic aérien, en particulier sur chacun des faisceaux desservis par les Aéroports de Paris. Nous avons ensuite réalisé une revue de littérature, en partie guidée par notre référent, pour mieux connaître le cadre théorique dans lequel s'inscrivent nos données, et identifier des modèles pertinents à implémenter.

Tout d'abord, la prise en compte de la temporalité de nos données étant essentielle, nous nous sommes naturellement orientés vers une approche sur les séries temporelles. Nous nous sommes alors référés à notre cours du deuxième semestre de deuxième année "Séries Temporelles Linéaires", dispensé par M. Francq, ainsi que sur quelques ouvrages complémentaires, tels que *Times Series : Theory and Methods* de Brockwell et Davis [1] ainsi que l'ouvrage *Séries temporelles et modèles dynamiques* de Gouriéroux et Monfort [2].

Très vite, nous avons cherché à exploiter les modèles classiques ARMA, ARIMA et SARIMA, qui sont au cœur de la modélisation de séries temporelles. Le modèle ARMA nous aurait permis de saisir la complexité des séries étudiées. Cependant, la visualisation de nos données nous a permis de constater que les séries du trafic aérien n'étaient pas stationnaires et présentaient des tendances et saisonnalités plus ou moins marquées suivant le faisceau étudié. Pour pallier à ce problème de non-stationnarité, nous nous sommes tournés vers le modèle ARIMA. Ce modèle ne peut néanmoins pas prendre en compte le caractère saisonnier très marqué du trafic aérien, comme par exemple le fait que certains jours de la semaine, ou les périodes de vacances scolaires, suscitent davantage de trafic. C'est pourquoi, comme nous disposons de données sur plusieurs années, nous avons porté notre attention sur le modèle SARIMA, avec une saisonnalité implémentée sur les douze mois de l'année.

Dans son mémoire *Modélisation et prévision de la consommation horaire d'électricité au Québec*, Sylvestre Tatsa admet que le modèle SARIMA se révèle efficace à court-terme. Mais à long-terme, pour améliorer les performances du modèle en termes de prédictions, il suggère l'ajout de variables exogènes ayant une influence directe sur notre variable d'intérêt, qui dans le cadre de notre projet est le nombre de passagers. Un tel apport au modèle lui permettrait de tenir compte de certaines spécificités que le passé de notre variable d'intérêt ignore. Ainsi, sur de plus grands horizons de prévision, il obtient de meilleures prédictions avec des indicateurs MAPE plus petits. Cependant, implémenter une telle méthode se révèle coûteux, puisque l'auteur raconte avoir fait face à des limites computationnelles dans ses modélisations.

Pour aller au-delà des modèles ARIMA et SARIMA, nous aurions pu nous intéresser au modèle de Holt-Winters, décrit en détail dans l'article *Prévisions journalières de séries temporelles saisonnières avec effets calendaires* de Joelle Bouchard et Benoit Montreuil. Ce modèle a l'avantage de pouvoir prendre en compte plusieurs saisonnalités (contrairement au modèle SARIMA qui n'en considère qu'une, et au modèle ARIMA qui n'en tient pas compte du tout) ainsi que des effets calendaires. En effet, cette méthode consiste à diviser l'année en plusieurs saisons, qui reflètent par exemple l'impact d'une fête annuelle (comme Noël ou la Saint Valentin) ou l'effet d'un mois particulier sur la demande des consommateurs. Elle paraît ainsi plus parcimonieuse. Cependant, les résultats de Sylvestre Tatsa dans l'article cité précédemment montrent que le modèle SARIMA (à la fois avec et sans variables exogènes) s'avère plus performant dans ses prédictions que celui de Holt-Winters.

Outre l'approche classique de modélisation de nos séries temporelles par ARIMA/SARIMA, nous nous sommes penchés sur un article proposé par notre encadrant, détaillant une approche de prévision non-paramétrique. Cette approche a initialement été élaborée pour le traitement de la consommation d'électricité et a été présentée dans des articles de Antonialdis et al. en 2012 [5] et 2014 [6]. Elle permet de construire des prévisions en présence de non stationnarités, avec une approche de la saisonnalité plus complexe que l'approche SARIMA qui ne tient compte que d'une saisonnalité. Il nous a paru cohérent de tenter d'appliquer ce modèle à nos données, le trafic aérien présentant plusieurs saisonnalités, notamment hebdomadaire et annuelle, tout comme les données de consommation présentées dans cet article.

L'idée du modèle est de découper nos données fonctionnelles en blocs représentant des trajectoires : les blocs futurs sont alors prédits à partir des blocs passés, auxquels on alloue des poids déterminés par leur pertinence pour la prédition. Pour construire ces poids, Antonialdis et al. proposent également d'intégrer des corrections calendaires, ce qui n'était pas possible avec un modèle SARIMA.

Par ailleurs, l'article évoque différentes opérations pour créer la prévision : le passage par les transformées en ondelettes est présenté comme une manière d'écraser les comportements non stationnaires des trajectoires. Les poids sont ensuite construits en utilisant un noyau de probabilité. Nous avons cherché à comparer les résultats de prédiction obtenus selon l'utilisation de la transformation en ondelettes ou non qui semble difficile à mettre en œuvre concrètement.

Enfin, notre référent nous a proposé une dernière approche : passer par une régression de type Lasso, en utilisant les données du calendrier à notre disposition. En passant par une régression, le trafic aérien serait alors prédit uniquement en fonction d'indicatrices temporelles construites à partir du calendrier. Pour préciser notre choix de modélisation, nous nous sommes référés à l'ouvrage théorique très complet de Hastie et al [7].

Comme il nous est possible, grâce au calendrier fourni par les Aéroports de Paris, de prendre en compte le jour de la semaine, le mois, la semaine, mais aussi chaque jour férié ou chaque position d'un jour dans les vacances scolaires, nous avons potentiellement beaucoup de variables possibles. Cela rend légitime le choix de l'approche Lasso, qui permettrait de réduire ce nombre en fixant un grand nombre de coefficients à zéro, contrairement à d'autres approches comme Ridge, qui pénalise seulement la norme du vecteur des coefficients, laissant de nombreux coefficients faibles mais différents de zéro.

Finalement, nous avons donc retenu quatre approches possibles pour prédire le trafic aérien à partir de nos données : une approche en séries temporelles avec les modèles ARIMA et SARIMA, une approche non-paramétrique, et une approche de type régression, avec une régression pénalisée de type Lasso.

3 Approche empirique

Dans cette partie, nous allons présenter plus en détail les modèles précédemment sélectionnés. Nous préciserons le cadre théorique dans lequel ils s'appliquent et les méthodes employées pour les mettre en oeuvre sur nos données.

3.1 Modèle ARIMA

3.1.1 Définition du modèle

Nous nous sommes d'abord intéressés au modèle ARIMA (Auto Regressive Integrated Moving Average).

Soit (X_t) une série temporelle. On appelle opérateur de retard l'opérateur B qui à toute observation associe la précédente : $\forall t, BX_t = X_{t-1}$.

On dit que la série (X_t) suit un modèle ARIMA(p, d, q) si $Y_t = (1 - B)^d X_t$ suit un modèle ARMA(p, q), c'est-à-dire, Y_t est faiblement stationnaire et satisfait $\forall t$:

$$Y_t - \sum_{m=1}^p \phi_m Y_{t-m} = \varepsilon_t - \sum_{m=1}^q \psi_m \varepsilon_{t-m} \quad (1)$$

où $\varepsilon_t \sim BB(0, \sigma^2)$.

En cours, nous avons étudié la méthodologie de Box-Jenkins pour implémenter un modèle ARIMA(p, q), qui est la suivante :

- Etudier la stationnarité de la série, si besoin la rendre stationnaire, par différenciation par exemple.
- Identifier a priori les ordres p_{max} et q_{max} .
- Estimer les paramètres (méthode des moindres carrés, du maximum de vraisemblance...) des modèles candidats et tester leur validité.
- Choisir le meilleur modèle (à l'aide des critères AIC ou BIC par exemple) parmi tous les candidats.

3.1.2 Stationnarisation de notre série

Tout d'abord, nous vérifions que notre série n'est pas déjà stationnaire. Si elle ne l'est effectivement pas, afin de la stationnariser, nous la différencions. On fixe alors d en fonction du nombre de différenciations que l'on a effectuées. En pratique, on a $d \leq 2$: par ailleurs, il faut veiller à ne pas sur-différencier notre série, sous peine de rendre le modèle ARMA associé non-inversible.

Pour vérifier la stationnarité de la nouvelle série obtenue, on peut passer par le test augmenté de Dickey-Fuller (ADF), ou bien le test KPSS (Kwiatkowski-Phillips-Schmidt-Shin) :

- Pour le premier test, l'hypothèse nulle est "la série présente une racine unitaire, i.e. la série n'est pas stationnaire", elle repose donc sur la non-stationnarité.
- Pour le second test, l'hypothèse nulle est "la variance de la partie non-stationnaire de la série est nulle", elle repose donc sur la stationnarité.

3.1.3 Identification des ordres p_{max} et q_{max}

Une fois que l'on a stationnarisé notre série, on cherche à déterminer p et q afin de corriger les auto-corrélations résiduelles. Pour cela, nous devons déterminer une borne supérieure pour ces deux ordres. Pour p , il s'agit d'observer la fonction d'auto-corrélation partielle alors que pour q , il s'agit d'observer la fonction d'auto-corrélation.

Rappelons la définition de ces deux fonctions. L'auto-corrélation partielle r et l'auto-corrélation ρ sont données respectivement $\forall h$ par :

$$r(h) = \text{Corr}(Y_t, Y_{t-h}|Y_{t-1}, \dots, Y_{t-h+1}) \quad \rho(h) = \text{Corr}(Y_t, Y_{t-h}) \quad (2)$$

Pour déterminer les ordres p_{max} et q_{max} maximaux, il suffit de prendre les décalages à partir desquels respectivement l'auto-corrélation partielle et l'auto-corrélation ne sont plus significatifs. Cela peut se faire en lisant l'autocorrélogramme partiel et l'autocorrélogramme.

3.1.4 Estimation, test de validité des modèles possibles et choix du meilleur modèle

Une fois que l'on a déterminé p_{max} et q_{max} , on teste tous les modèles ARMA(p, q) pour $p \in \llbracket 0, p_{max} \rrbracket$ et $q \in \llbracket 0, q_{max} \rrbracket$, et on sélectionne le plus précis parmi ceux qui sont valides. Un modèle est dit valide si d'une part ses résidus suivent un bruit blanc i.e. ses résidus ne sont pas auto-correlés, et d'autre part si les coefficients estimés des ordres AR et MA les plus élevés sont statistiquement significatifs.

Le test de Ljung-Box a pour hypothèse nulle "la nullité jointe des auto-corrélations jusqu'à un rang k donné" et peut être utilisé pour tester la blancheur des résidus. On effectue ce test sur nos résidus, des rangs 1 à 24 (on a une périodicité annuelle, on va donc tester sur deux périodicités).

Les estimations de nos modèles sur Python affichent les p-valeurs associées aux coefficients AR et MA : on considère qu'ils sont significatifs si leurs p-valeurs sont suffisamment petites.

Enfin, afin de départager les candidats, nous pouvons utiliser un critère d'information, comme AIC (Critère d'Information d'Akaike) ou BIC (Critère d'Information Bayésien). Il s'agit de minimiser ces critères afin de trouver le meilleur modèle. Dans le cadre de notre projet, nous faisons appel au critère BIC, car ce dernier pénalise les modèles trop complexes, i.e. ayant un trop grand nombre de paramètres.

3.2 Modèle SARIMA

3.2.1 Définition du modèle

Puisque nos données présentent une saisonnalité, nous avons étendu notre étude au modèle SARIMA (Seasonal Autoregressive Integrated Moving Average), adapté à ce genre de situation.

On dit que la série (X_t) suit un modèle SARIMA(p, d, q), $(P, D, Q)_s$ de période s si $Y_t = (I - B)^d(I - B^s)^D X_t$ suit un modèle ARMA causal, c'est-à-dire si :

$$\phi(B)\Phi(B^s)Y_t = \psi(B)\Psi(B^s)\varepsilon_t + c \quad (3)$$

où ϕ, Φ, ψ et Ψ sont des polynômes de degrés respectifs p, P, q et Q , c est une constante et $\varepsilon_t \sim BB(0, \sigma^2)$.

3.2.2 Sélection des ordres optimaux

On a implémenté le modèle SARIMA de la façon suivante :

- s correspond au paramètre de saisonnalité : puisque notre série présente une saisonnalité annuelle (12 mois), il est pertinent de prendre $s = 12$.
- Les ordres d et D sont à choisir de sorte que (Y_t) définie comme précédemment soit stationnaire.
- On détermine les ordres p_{max} et q_{max} de la même façon que pour le modèle ARIMA.
- Pour P_{max} et Q_{max} , on étudie également l'autocorrélogramme partiel et l'autocorrélogramme respectivement : pour chacun, il faut compter les premiers retards multiples de s à partir desquels l'auto-corrélation partielle r et auto-corrélation ρ ne sont plus significatifs.
- On teste également la validité de nos modèles, en veillant à ce que l'on ait bien des résidus qui correspondent à des bruits blancs, et que l'on ait bien des coefficients statistiquement significatifs.
- Finalement, parmi tous les modèles possibles, on minimise selon le critère AIC ou le critère BIC.

3.3 Modèle non paramétrique

Le modèle non paramétrique présenté dans cette section s'inspire des articles d'Antoniadis et al. (en 2012 [5] et 2014[6]) sur la prévision de consommation électrique.

Son principe général est de trouver dans le passé des contextes similaires à la situation présente afin de prévoir le futur par une combinaison linéaire des futurs des passés les plus semblables au présent.

3.3.1 Définition du modèle

On cherche à prédire un processus stochastique qui est continu $X = (X(t), t \in [0, T])$, mais que l'on n'observe que de manière discrète, à intervalles de temps réguliers.

Pour chercher dans la trajectoire observée des contextes similaires au présent, on segmente la trajectoire de X en n blocs de taille δ : on dispose alors d'un processus stochastique $Z = (Z_k(t))_{1 \leq k \leq n, 0 \leq t \leq \delta}$, où $Z_k(t) = X((\delta - 1)(k - 1) + t)$. Il s'agit en fait d'une série temporelle de fonctions, et l'on cherche à prédire Z_{n+1} pour pouvoir prédire la trajectoire future à un certain horizon h_{Prev} .

L'idée de ce modèle est de prévoir Z_{n+1} par une combinaison linéaire de l'échantillon Z_1, \dots, Z_n . L'équation principale du modèle s'écrit alors pour un processus stationnaire :

$$\hat{Z}_{n+1}(t) = \sum_{m=1}^{n-1} w_{n,m} Z_{m+1}(t) \quad (4)$$

Dans cette équation, on fait apparaître un vecteur de poids $(w_{n,m})_{1 \leq m \leq n-1}$, afin de pondérer les futurs des blocs passés considérés. Pour le construire, on cherche à comparer les différents blocs et à évaluer leur similarité. Le processus de construction de ce vecteur peut être résumé en trois étapes :

- On utilise une mesure de dissimilarité D pour pouvoir comparer les blocs passés au bloc présent (tout dernier bloc de l'historique). Nous avons retenu comme mesure de dissimilarité une distance proche de la distance euclidienne, définie par :

$$\forall 1 \leq i, j \leq n, D(Z_i, Z_j) = \sum_{t=0}^{\delta} (Z_i(t) - Z_j(t))^2 \quad (5)$$

- La dissimilarité calculée pour chacun des blocs passés est ensuite transformée à l'aide d'un noyau de probabilité gaussien. Les poids construits correspondent alors intuitivement à la renormalisation de la distance renormalisée du bloc témoin dans l'ensemble des blocs témoins au bloc futur :

$$\forall 1 \leq m \leq n-1, w_{n,m} = \frac{K_h(D(Z_n, Z_m))}{\sum_{i=1}^{n-1} K_h(D(Z_i, Z_n))} \text{ avec } K_h(x) = \frac{1}{h} \left(\frac{1}{\sqrt{2\pi}} \right)^{\delta} \exp \left(-\frac{x}{2h} \right) \quad (6)$$

où h est un hyper-paramètre appelé largeur de la fenêtre.

- Enfin, pour chaque date t que l'on souhaite prédire, on réalise une correction calendaire des poids précédents, en ne retenant que les poids placés sur les blocs dont les futurs sont similaires à la date à prédire :

$$\forall 1 \leq m \leq n-1, \tilde{w}_{n,m} = \frac{w_{n,m} \mathbb{1}_{(gr(n)=gr(m))}}{\sum_{i=1}^{n-1} w_{n,i} \mathbb{1}_{(gr(n)=gr(i))}} \quad (7)$$

Dans notre cas, les groupes sont déterminés à partir de la base **Calendrier** à notre disposition : nous gardons les dates qui correspondent aux mêmes jours de vacances ou jours fériés, afin de capter en priorité les périodes de charge et décharge importantes au trafic aérien. On prend également en compte le jour de la semaine, lorsque cela est possible (dans certains rares cas cela peut rendre tous les poids nuls...).

3.3.2 Stationnarité

Notons que l'on a supposé le processus Z stationnaire dans l'équation (4) du modèle. Cela n'est à priori pas vérifié dans notre cas.

Une solution apportée par l'article d'Antoniadis [5] est d'introduire une transformée en ondelettes discrète (DWT) : la décomposition permet d'obtenir d'une part une approximation de la trajectoire et d'autre part le détail dû aux non stationnarités et au bruit (équation (8)).

$$Z_i(t) = \sum_k^{2^{j_0}-1} c_{j_0,k}^{(i)} \phi_{j_0,k}(t) + \sum_{j=j_0+1}^J \sum_k^{2^j-1} d_{j,k}^{(i)} \psi_{j,k}(t) \quad (8)$$

Seule la partie "détail" présente dans le terme de droite de l'équation (8) est utilisée dans la construction d'une "dissimilitude" en prenant la distance entre les coefficients des blocs témoins.

Cependant, la décomposition en ondelettes s'est avérée en pratique très coûteuse en mémoire comme en temps d'exécution. Afin d'obtenir tout de même de bonnes prédictions, d'autres corrections sont adoptées pour pallier au problème d'instationnarité. Dans toutes les équations de la section précédente, nous travaillons sur des blocs préalablement centrés réduits afin d'éviter les problèmes dus au fait qu'initialement les blocs présentent des niveaux moyens très différents. La prédition est en toute fin remise à niveau, avec la moyenne et l'écart-type du tout dernier bloc témoin. De plus, passer la dissimilitude dans un noyau gaussien (équation (6)) permet d'écraser les non stationnarités du prédicteur. Enfin, l'existence de classes de segments est une autre source potentielle d'instationnarité qui est corrigée grâce aux ajustements calendaires (équation (7)).

3.3.3 Choix des paramètres

Dans ce modèle, deux variables sont à déterminer : la taille des blocs considérés δ , et la largeur de fenêtre h (qui intervient dans l'équation (6)).

En ce qui concerne la taille des blocs δ , la structure de nos données et leur saisonnalité annuelle nous a naturellement orienté vers des blocs de $\delta = 365$ jours.

L'hyper-paramètre h est primordial dans l'estimation par noyau, car il définit l'importance attribuée aux valeurs extrêmes : un bloc très similaire au présent sauf en quelques points sera plus fortement pénalisé pour une grande valeur de h . Comme nos données ont une "structure" vis à vis du calendrier, il nous a paru plus cohérent de tester différentes valeurs de h sur des parties précises de l'historique à notre disposition plutôt que sur des parties choisies aléatoirement. Nous avons pris le parti de déterminer h comme moyenne des paramètres h qui permettent de prédire au mieux (au sens de la minimisation de la RMSE) les périodes similaires de l'historique à notre disposition. Par exemple, si l'on veut prédire la première semaine de janvier, on prend la moyenne des h qui permettrait de prédire au mieux les premières semaines de janvier présentes dans l'historique (en ne considérant pour chacune que l'historique antérieur).

Nous nous sommes aussi intéressées à une sélection jointe de la taille des blocs et du paramètre h (non présentée ici). La méthode par *gridsearch* adoptée sur les données n'a toutefois pas donné de résultats plus performants que la sélection sur l'hyper-paramètre h à taille de blocs fixée à 365 jours, et ce même pour des prévisions de court terme.

3.3.4 Intervalles de confiance

Afin d'enrichir nos prévisions, nous avons également calculé des intervalles de confiance en nous inspirant de la méthode adoptée par Antoniadis et al. en 2014 [6].

Les intervalles de confiance que nous avons construits sont des intervalles ponctuels, calculés séparément pour chaque date prédictive, et non une bande de confiance. De plus, comme précédemment, les blocs sont centrés réduits avant le calcul des intervalles, puis remis à niveau par la suite, pour palier aux problèmes d'instationnarités.

Les bornes des intervalles de confiance que nous souhaitons déterminer sont identifiées comme les quantiles α et $1 - \alpha$ de la distribution conditionnelle de Z_{n+1} sachant Z_1, \dots, Z_n . Les poids $(\tilde{w}_{n,m})_{1 \leq m \leq n-1}$ induisant une loi de probabilité discrète que les observations, nous les avons utilisés pour réaliser un échantillonnage bootstrap $Z_{n+1}^{(1)}, \dots, Z_{n+1}^{(B)}$ de $B = 1000$ pseudo réalisations par tirage aléatoire des segments du passé. On obtient alors un intervalle de confiance en prenant les quantiles α et $1 - \alpha$ de cet échantillon.

3.4 Modèle LASSO

3.4.1 Définition du modèle

En dernier lieu, nous avons travaillé sur le modèle Lasso (Tibshirani, 1996 [7]), appliqué aux différentes indicatrices temporelles (jour, semaine, mois). Pour nos données, y la quantité à prédire est le nombre de passagers ou le taux de remplissage sur les différents faisceaux, X décrit les régresseurs considérés - indicatrices temporelles, et t est un paramètre qui contrôle le niveau de régularisation des coefficients estimés par le modèle.

$$\begin{aligned} \min_{\beta \in \mathbb{R}^p} & ||y - X\beta||_2^2 \\ \text{s.c. } & ||\beta||_1 \leq t \end{aligned} \tag{9}$$

Le Lasso est donc une régression linéaire, dite pénalisée en raison de la contrainte définie dans l'équation 9. Cette méthode permet d'effectuer à la fois la sélection des régresseurs significatifs et l'estimation de la série. Elle a l'avantage de pouvoir capter les saisonnalités multiples et identifier les régresseurs les plus pertinents.

Le problème du Lasso se réécrit avec le lagrangien :

$$\min_{\beta \in \mathbb{R}^p} \{ ||y - X\beta||_2^2 + \alpha ||\beta||_1 \}$$

où α est appelé paramètre de régularisation du modèle.

Sur le plan théorique, la résolution de la contrainte ne peut pas toujours être exprimée explicitement, puisque la norme 1 n'est pas différentiable. On utilisera en pratique le Lasso-CV du package `scikit-learn`, qui atteint la fonction-objectif minimale pour l'erreur de prédiction par validation croisée¹.

3.4.2 Intervalle de prédiction

Du fait de la définition du Lasso, qui est une régression linéaire pénalisée dans laquelle on choisit de diminuer la variance en pénalisant les "mauvais" régresseurs² l'estimateur est biaisé. La construction d'un intervalle de confiance est alors moins pertinente que pour nos modèles précédents. En revanche, cette caractéristique du modèle Lasso en fait un modèle de sélection parcimonieuse. Néanmoins, de la même manière que dans le modèle paramétrique, nous pouvons chercher à estimer un intervalle de prédiction par une méthode bootstrap inspirée de (Kumar and Srivastava, 2012 [8]).

L'idée ici est d'échantillonner et de prédire un grand nombre de fois sur nos données pour estimer le bruit du modèle, les résidus et le biais. On utilise les prédictions centrées $\bar{y}_{b,n}$ sur chaque échantillon du bootstrap comme estimation du bruit du modèle et on estime le biais et les résidus à l'aide de l'erreur entre le réalisé

1. On peut se poser la question de la pertinence d'une démarche par validation croisée sur une série temporelle. L'article "On the use of cross-validation for time series predictor evaluation" (Bergmeir, Benitez (2012)) se penche sur ce problème et évoque une perte de stabilité du Lasso pour des séries temporelles. Il existe des améliorations théoriques (Roberts, Nowak (2014)) mais difficiles à mettre en place d'un point de vue computationnel. En pratique une validation croisée standard est satisfaisante.

2. par exemple, si le paramètre de régularisation λ choisi est presque nul, la régression n'est pas pénalisée, en revanche, si λ tend vers $+\infty$ alors les coefficients de β seront presque tous nuls

et la prédiction sur chaque échantillon bootstrap. Il apparaît alors un problème potentiel d'overfitting sur les résidus d'entraînement comme de validation. On introduit alors une mesure de cet overfitting : la *relative overfitting rate* définie comme $\hat{R} = \frac{\text{validation error} - \text{train error}}{\gamma - \text{train error}}$ avec γ la *no-information error rate* qui est la loss calculée si les données sont toutes indépendantes. \hat{R} est une mesure qui vaut donc 0 s'il n'y a pas d'overfitting et tend vers 1 si γ et l'erreur d'entraînement sont proches. A partir de \hat{R} , on construit des poids permettant de corriger l'overfitting de l'estimation du biais et des résidus sur l'ensemble des données :

$$\hat{\varepsilon}_i = (1 - \text{weight}) \times \text{train error}_i + \text{weight} \times \text{validation error}_i$$

On peut à présent construire l'intervalle de prédiction suivant (avec B le nombre d'échantillons bootstraps, n la taille de l'échantillon total) :

$$C = \{\bar{y}_{b,n} + \hat{\varepsilon} \mid b < B, i < n\}$$

On construit enfin les bornes de l'intervalle de prédiction à l'ordre α en prenant pour borne inférieure le $\frac{\alpha}{2}$ -quantile de C et le $1 - \frac{\alpha}{2}$ -quantile de C en borne supérieure.

4 Format du code et outil PowerBi

Avant de visualiser les prévisions réalisées par les différents modèles présentés dans la section précédente, nous allons présenter rapidement le format de notre code, et l'outil PowerBi que nous avons créé pour les Aéroports de Paris.

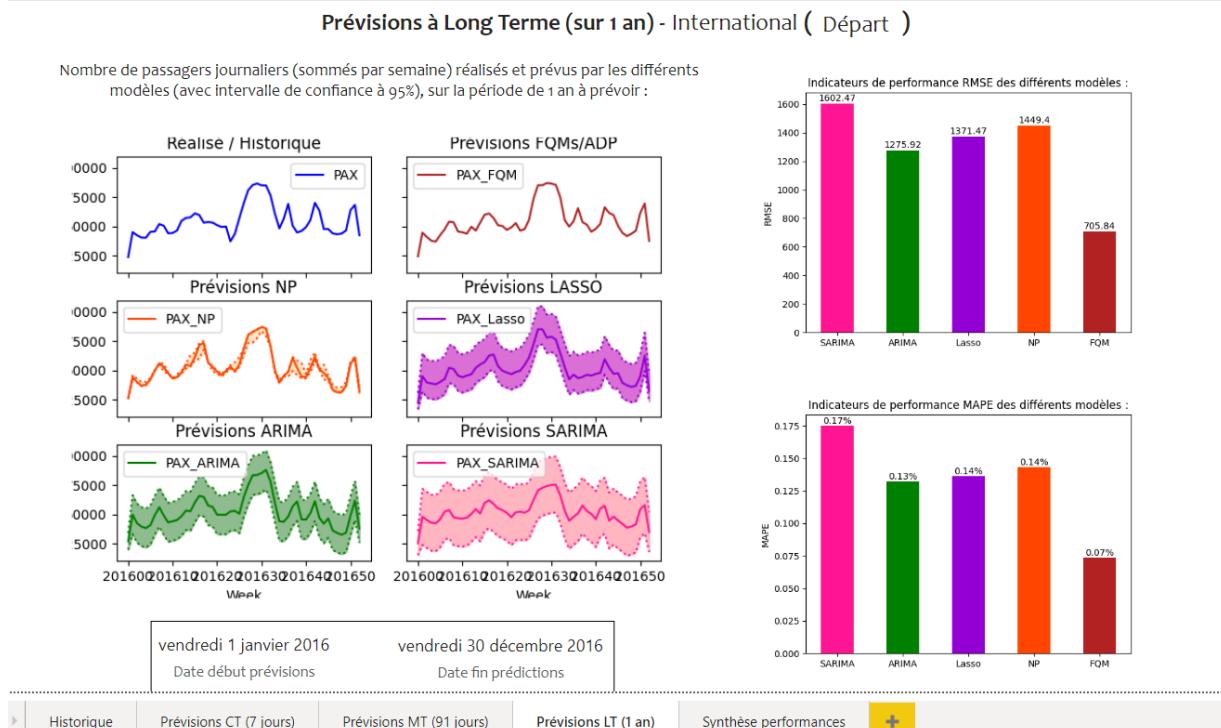
Chacun des quatre modèles présentés dans la section précédente a été implémenté sous Python dans un fichier dédié. Il se présente sous la forme finale d'une fonction `previsions_modèle` qui réalise des prévisions à partir d'un historique passé en argument (déjà agrégé de manière journalière et filtré sur un faisceau et type de mouvement).

Ces quatre fichiers s'accompagnent d'un fichier `__main__` qui centralise l'appel de nos modèles. L'utilisateur spécifie le chemin vers l'historique et le calendrier, choisit les mouvements, les horizons de prévision et les faisceaux sur lesquels il souhaite obtenir des prévisions, ainsi que l'intervalle de confiance souhaité. L'exécution de ce fichier réalise les prévisions suivant nos quatre modèles et crée des bases de données par horizon de prévision.

L'outil PowerBi que nous avons implémenté récupère les bases de données ainsi créées, dans notre cas avec des prévisions à court terme (sur 7 jours), moyen terme (sur 3 mois) et long terme (sur 1 an). Il propose des visualisations par faisceau et type de mouvement. Une fois ceux-ci choisis à travers les paramètres centralisés, différents onglets sont présentés proposant : une visualisation de l'historique, une visualisation des prévisions pour chacun des horizons de prévision ainsi qu'un onglet de synthèse des indicateurs RMSE et MAPE sur chaque horizon de prévision, faisceau et modèle (sur le mouvement considéré).

Des visualisation des différents onglets de l'outil sont disponibles en annexe dans la section A.4. Voici par exemple l'onglet de visualisation des prévisions de long terme des départs du faisceau International :

FIGURE 2 – Visualisation des prévisions à Long Terme (année 2016) dans l'outil PowerBi, sur les départs du faisceau International



5 Résultats

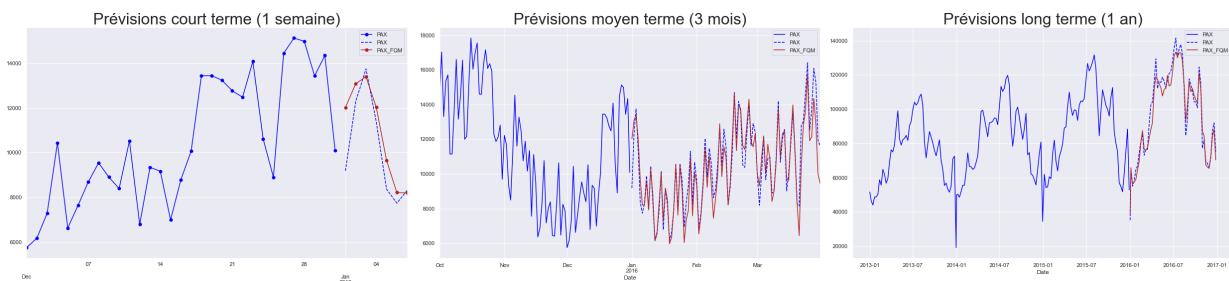
Une fois nos quatre modèles implémentés, nous avons réalisé des prévisions à différents horizons sur chacun de nos faisceaux, afin de comparer leurs performances. Nous avons réalisé des prévisions de court terme, sur la première semaine de 2016, à moyen terme, sur les trois premiers mois de 2016, et à long terme, sur toute l'année 2016, et ce à partir d'un même historique allant de 2008 à 2015. La totalité des prévisions sur les départs est disponible en annexe, dans les figures 10, 11 et 12.

5.1 Prévisions du trafic aérien par chacun des modèles implémentés

Pour comprendre au mieux l'apport de chacune des différentes approches de prévisions que nous avons retenues, nous allons commencer par regarder de plus près les prévisions réalisées par chacun des modèles sur le faisceau Schengen. Nous l'avons choisi car son historique n'est pas dégradé (contrairement à celui du faisceau Autre UE), et il présente différentes caractéristiques que l'on peut retrouver dans d'autres faisceaux : une tendance à la hausse, des saisonnalités annuelle et hebdomadaire marquées, ainsi qu'une augmentation progressive de la "hauteur" des motifs d'une année sur l'autre (cf. courbe bleu de la deuxième ligne de la figure 8).

La figure 3 propose un aperçu du trafic aérien sur les différentes plages de prévisions étudiées (en pointillés bleus), ainsi qu'une partie de l'historique qui les précède (en trait plein bleu). On y a également représenté les prévisions réalisées par les aéroports de Paris (en rouge). On remarque leur grande capacité à prévoir le trafic aérien. N'oublions pas que ces prévisions sont réalisées d'une semaine sur l'autre, contrairement à nos modèles, qui ne disposent que de l'historique qui précède la période de prévision (trait plein bleu).

FIGURE 3 – Nombre de passagers journaliers réalisés (bleu), et prédis par les aéroports de Paris (rouge) pour les départs de Schengen, sur les différents horizons de prévision étudiés



5.1.1 Modèles ARIMA/SARIMA

Intéressons-nous pour commencer aux prévisions réalisées par les modèles ARIMA et SARIMA : les figures 4 et 5 présentent les prévisions du modèle ARIMA en vert et SARIMA en rose sur le faisceau Schengen pour les différents horizons de prévision, avec les intervalles de confiance associés.

On remarque que ces modèles, qui ne basent leurs prévisions que sur l'historique, sans aucune correction calendaire, n'arrivent pas à saisir la spécificité du trafic aérien sur le mois de janvier, et donnent de mauvaises prévisions à court terme comme à moyen terme dans notre exemple. En effet, les motifs reproduits à court et moyen termes ressemblent davantage aux motifs qui les précèdent, et n'arrivent pas à rendre compte de la particularité du mois de janvier, visible dans le réalisé.

Pour ce qui est des prévisions à long terme, les deux modèles arrivent à produire une prévision dont la forme semble cohérente. Cependant, on remarque que les prévisions sont très lissées et perdent en précision

FIGURE 4 – Nombre de passagers journaliers réalisés (bleu), et prédis par le modèle ARIMA (vert), pour les départs de Schengen, sur les différents horizons de prévision étudiés, avec intervalles de confiance à 95%

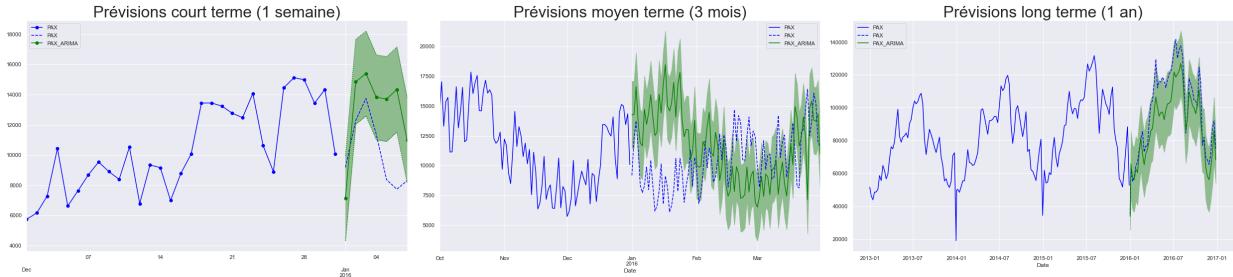
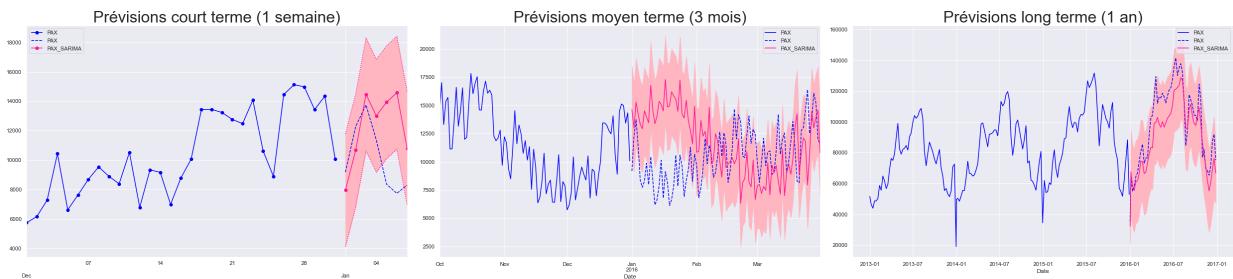


FIGURE 5 – Nombre de passagers journaliers réalisés (bleu), et prédis par le modèle SARIMA (rose), pour les départs de Schengen, sur les différents horizons de prévision étudiés, avec intervalles de confiance à 95%



sur de gros horizons de prévisions. Les deux modèles n'arrivent visiblement pas à rendre compte du fait que sur ce faisceau, les motifs annuels ont tendance à s'«élargir» au fur et à mesure des années.

Enfin, on obtient sur nos graphiques des intervalles de confiance plus larges avec le modèle SARIMA qu'avec le modèle ARIMA, ce qui s'explique par le fait que le premier prend en compte plus de paramètres (les paramètres saisonniers) que le dernier. En outre, on peut notamment remarquer à court et moyen termes que le réalisé n'est pas compris dans les intervalles de confiance à bien des endroits. Il semblerait alors que les intervalles de confiance trouvés ne soient pas toujours pertinents.

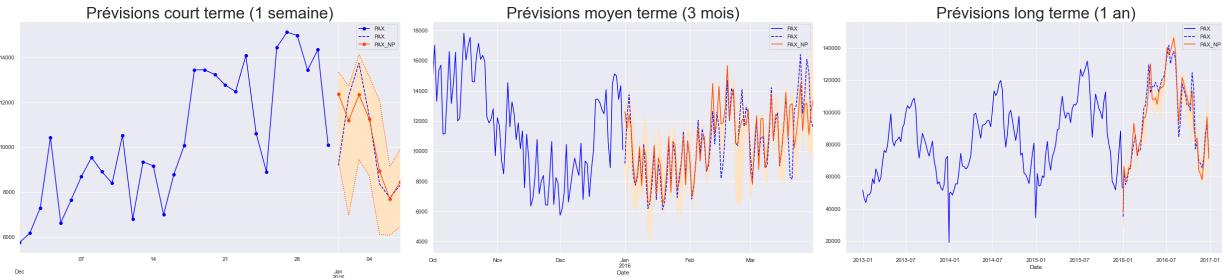
En somme, bien qu'ils perdent en précision sur de gros horizons de prévision, les modèles ARIMA et SARIMA ont l'avantage d'être relativement simples (notamment par rapport aux deux suivants), et de ne nécessiter qu'une série stationnaire qui peut être obtenue par différentiation.

5.1.2 Modèle Non-Paramétrique

D'un point de vue général, les prévisions réalisées par le modèle non-paramétrique sont plutôt satisfaisantes sur chacun des faisceaux et des horizons de prévision. Leur qualité de prévision est donc assez régulière. On remarque que les hyper-paramètres h choisis par le modèle, et présentés en annexe dans la table 8, ont tendance à diminuer sur un même faisceau lorsque l'horizon de prévision augmente. Cela signifie que plus l'horizon de prévision est petit, plus le modèle est exigeant avec la similarité aux blocs (de 365 jours) de l'historique avec le tout dernier bloc du passé.

Intéressons-nous de manière plus particulière aux prévisions réalisées par ce modèle sur le faisceau Schengen : la figure 6 présente ces prévisions en orange, pour les différents horizons retenus, et avec des intervalles de confiance à 95%.

FIGURE 6 – Nombre de passagers journaliers réalisés (bleu), et prédis par le modèle non-paramétrique (orange), pour les départs de Schengen, sur les différents horizons de prévision étudiés, avec intervalles de confiance à 95%



Rappelons que ce modèle, contrairement aux modèles ARIMA et SARIMA précédemment évoqués, utilise en plus de l'historique du trafic aérien les données calendaires pour réaliser ses prédictions. L'avantage de cette correction calendaire est de capturer plusieurs saisons : bien que la taille des blocs soit de 365 jours (pour prendre en compte la tendance annuelle, beaucoup plus marquée), la correction calendaire prend entre autre en compte le jour de la semaine et le fait qu'un jour soit férié ou pendant les vacances scolaires. A court et moyen termes, les prévisions de début janvier sont plus proches du réalisé que pour les modèles ARIMA et SARIMA. Seule la prévision du 1er janvier est un peu moins en accord avec le réalisé.

Au regard des prévisions de la figure 6, le modèle non-paramétrique semble offrir des prévisions satisfaisantes non seulement à court et moyen terme, mais aussi à long terme : ces prévisions se rapprochent du réalisé, en captant à la fois la tendance à la hausse, mais aussi le fait que les motifs annuels s'agrandissent au fil des années. Cela est certainement dû au fait que le modèle non-paramétrique calcule ses poids avec une mesure de dissimilarité qui doit défavoriser les blocs les plus passés.

En revanche, les intervalles de confiance proposés par bootstrap dans le modèle non-paramétrique ne semblent pas très satisfaisants. On remarque pour le faisceau Schengen ici étudié qu'à moyen et à long terme, ces intervalles s'amenuisent et n'incluent pas beaucoup le réalisé. Sur d'autres faisceaux, comme par exemple sur les prévisions à moyen terme du faisceau national (figure 11), certains points n'ont pas d'intervalle de confiance. Nous pensons que cela est dû à la correction calendaire qui est très forte : pour certaines prédictions, ne choisir dans l'historique que des jours ayant les mêmes caractéristiques (jour de la semaine, semaine, même jour des vacances, ...) conduit à ne choisir que très peu de points (voire un seul). La méthode bootstrap génère alors forcément des intervalles de confiance très fins (voire pas d'intervalle s'il n'y a qu'un point avec lequel faire des simulations). Le modèle non-paramétrique n'est donc pas satisfaisant si l'on souhaite avant tout avoir des intervalles de confiance (ce qui n'est par exemple pas le cas des Aéroports de Paris, qui s'intéressent plutôt à la valeur de prévision).

5.1.3 Modèle Lasso

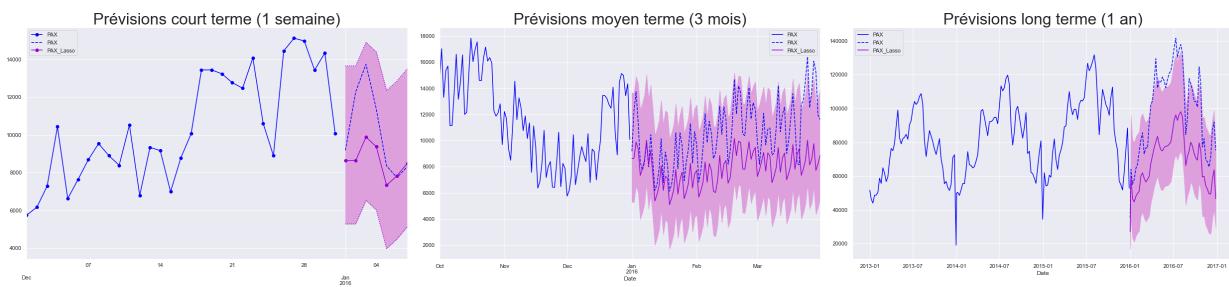
D'un point de vue général, les prévisions réalisées par le modèle Lasso sont souvent, au moins visuellement, plus proches de celles du modèle non paramétrique que de celles des modèles ARIMA et SARIMA dans les figures 10 à 12. On rappelle que, comme le modèle non-paramétrique, le modèle Lasso utilise en plus de l'historique les informations contenues dans le *calendrier* pour réaliser ses prévisions.

Les hyper-paramètres α choisis par le modèle, résumés dans la table 9, varient énormément d'un faisceau à l'autre. On peut d'ailleurs remarquer un grand point faible du modèle Lasso : ce modèle ne parvient pas

à réaliser des prévisions satisfaisantes sur le faisceau Autre UE, dont l'historique est "dégénéré" car anormalement aplati sur les premières années. Le coefficient α qui est choisi utilise la quasi totalité des variables explicatives disponibles (295 sur 300), et ne parvient pourtant pas à réaliser des prévisions satisfaisantes : comme en témoignent les figures 10 à 12, les prévisions du modèle Lasso sont anormalement aplatis, avec des intervalles de confiance extrêmement importants et donc imprécis.

Intéressons-nous de manière plus particulière aux prévisions réalisées par ce modèle sur le faisceau Schengen : la figure 7 présente ces prévisions en violet, pour les différents horizons retenus, et avec des intervalles de confiance à 95%.

FIGURE 7 – Nombre de passagers journaliers réalisés (bleu), et prédis par le modèle Lasso (violet), pour les départs de Schengen, sur les différents horizons de prévision étudiés, avec intervalles de confiance à 95%



On remarque que les prévisions obtenues avec le modèle Lasso sont proches de celles du modèle non-paramétrique dans leur forme : elles captent par exemple plutôt bien à moyen terme la particularité du mois de janvier, contrairement aux modèles ARIMA et SARIMA. En revanche, les prévisions du modèle Lasso s'écartent davantage du réalisé, et semblent trop aplatis bien qu'ayant la bonne "forme". Cela est particulièrement visible sur le long terme : le modèle Lasso ne parvient pas à capter le fait que les motifs annuels "s'élargissent" avec le temps.

Pour remédier à ce problème, et aussi faire face à un historique "dégénéré" comme celui du faisceau Autre UE, il serait peut-être pertinent de chercher un moyen de pénaliser de manière "progressive" les valeurs de l'historique à mesure qu'elles sont lointaines.

Enfin, contrairement au modèle non-paramétrique, la méthode bootstrap utilisée dans le modèle Lasso permet d'aboutir à des intervalles de confiance plus satisfaisants, qui ont une largeur régulière, et englobent par exemple plus souvent le réalisé du faisceau Schengen sur la figure 7, au moins à court et moyen terme.

5.2 Comparaison des différents modèles

Dans la section précédente, nous avons regardé en détail les prévisions réalisées par chacun de nos modèles, ainsi que leur capacité à rendre compte des différentes particularités du trafic aérien. Nous allons désormais comparer nos modèles de manière plus générale, en s'intéressant à leurs performances globales de prévisions, et de manière plus pratique à leurs temps d'exécution.

5.2.1 Performances des modèles

Pour comparer la qualité des prévisions de nos différents modèles, nous avons calculé les indicateurs de performance RMSE et MAPE, par faisceau, par modèle et par horizon de prévision. Ces indicateurs sont

résumés dans la figure 13.

On remarque de manière assez nette que les prévisions réalisées par les Aéroports de Paris sont les plus performantes, et ce pour tous les faisceaux et horizons de prévision. Rappelons que ces prévisions sont réalisées d'une semaine sur l'autre, et ne peuvent donc pas réellement être comparées aux autres modèles à moyen et long terme.

Le modèle non-paramétrique apparaît comme le modèle le plus performant de manière générale sur tous les horizons de prévision, excepté peut-être sur les prévisions à court terme du faisceau Autre UE. Mais dans l'ensemble, les prévisions de ce modèle sont très satisfaisantes, et ce modèle se montre robuste à des historiques dégénérés comme celui du faisceau Autre UE. En revanche, comme nous l'avons montré dans la section précédente, c'est le modèle aux intervalles de confiance les moins pertinents.

En ce qui concerne les trois autres modèles, il n'y a pas de modèle qui se distingue particulièrement. Le modèle Lasso semble réaliser de moins bonnes prévisions seulement sur les faisceaux Autre UE et Schengen, pour lesquels l'historique a la particularité de s'aplatir progressivement dans le passé. De plus, si l'on ne dispose pas du calendrier, et que l'on compare seulement les modèles ARIMA et SARIMA, le modèle ARIMA semble meilleur tant en terme de RMSE que de MAPE pour les prévisions de long terme. En ce qui concerne les prévisions de court et moyen terme, SARIMA est, tout comme le modèle Lasso, moins bon seulement sur les faisceaux Autre UE et Schengen.

5.2.2 Temps d'exécution

Au-delà de la qualité de prévision de chacun de nos modèles, il est également important de prendre en compte leur temps d'exécution. En effet, nous cherchons à proposer des modèles qui peuvent être utilisés de manière pratique par les Aéroports de Paris. Il est donc important de donner quelques indications sur les temps d'exécution de nos modèles.

Pour ce faire, nous avons proposé en annexe quelques ordres de grandeur du temps d'exécution par modèle. Il ne s'agit pas de mesures absolues, mais de mesures qui ont été faites sur un même ordinateur et avec la même taille d'historique. Ces valeurs sont donc surtout présentées dans un but comparatif.

Les modèles les plus réguliers en terme d'exécution sont certainement les modèles ARIMA et SARIMA. Pour ces modèles, le temps d'exécution ne dépend pas de l'horizon de prévision souhaité et le calcul des intervalles de confiance est très rapide, le plus long étant de trouver les paramètres qui s'ajustent au mieux à l'historique. Comme présenté dans les tables 5 et 7, le temps moyen de calcul pour un faisceau et un mouvement donné est de moins d'une minute pour ARIMA et presque 7 minutes pour SARIMA.

En ce qui concerne le modèle non-paramétrique, comme dans les deux modèles précédents, les intervalles de confiance par bootstrap n'affectent quasiment pas le temps de calcul. En revanche, le temps de calcul dépend de l'horizon de prévision souhaité, le modèle recalculant des poids avec une correction calendaire adaptée pour chaque jour à prévoir. La figure 14 représente le temps d'exécution moyen par faisceau pour différents horizons de prévisions (14 points sur le graphique). On obtient une tendance clairement linéaire, ce qui peut poser problème pour des horizons de prévision très élevés. Cela dit, même pour des prévisions sur un an, nous avons obtenu un temps d'exécution moyen par faisceau de 132.07 secondes, soit un peu plus de 2 minutes. Cela reste moins important que le temps d'exécution des modèles SARIMA et Lasso.

Enfin, le modèle Lasso est le seul dont le calcul des intervalles de confiance a un réel impact sur le temps d'exécution, comme en témoigne la table 10. Dans le cas où l'on n'est pas intéressé par le calcul de tels intervalles (ce qui est notamment le cas des aéroports de Paris), le modèle Lasso se révèle être le modèle le plus rapide, avec un temps moyen d'exécution de 1.37 secondes. En revanche, si l'on souhaite obtenir des intervalles de confiance, le modèle Lasso devient le plus coûteux en temps d'exécution, avec un temps moyen d'exécution par faisceau de presque 25 minutes.

Conclusion

Pour répondre au mieux à la problématique des Aéroports de Paris, à savoir comment prédire au mieux le trafic aérien pour différents faisceaux et horizons de prévision, nous avons implémenté quatre modèles aux approches différentes.

Il semble que les approches les plus classiques, à savoir ARIMA et SARIMA, soient globalement moins performantes que les modèles non-paramétrique et Lasso, qui ont la particularité d'intégrer des corrections calendaires. Cependant, nous n'avons pas pu sélectionné un modèle qui se distingue des autres en terme de performances. Alors que le modèle non-paramétrique est meilleur en terme de prévision, il est le seul à ne pas fournir d'intervalles de confiance pertinents. Parmi les trois modèles restants, le modèle Lasso semble meilleur, mais n'est pas robuste à des historiques dégénérés et est coûteux en temps si l'on souhaite obtenir des intervalles de prévisions.

Un utilisateur souhaitant réaliser des prévisions devra donc adapter le choix de son modèle au faisceau qu'il a choisi, mais aussi à l'utilisation qu'il souhaite faire de ces prévisions et à leur temps de calcul. Pour aider ce choix, nous avons résumé les atouts et les inconvénients de chacun de nos modèles dans la table 2.

En revanche, pour un utilisateur tel que les Aéroports de Paris, qui dispose d'informations calendaires précises, et dont l'enjeu est de réaliser une prévision la plus précise possible du trafic aérien, sans réellement être intéressé par des intervalles de confiance, le modèle non-paramétrique semble le mieux adapté. En effet, ce modèle arrive à capter des saisons multiples, ici annuelles et hebdomadaires, est performant sur tous les horizons de prévision, et semble robuste à des historiques dégénérés comme celui du faisceau Autre UE. Bien qu'il ne soit pas aussi performant que les prévisions réalisées "à la main" par les Aéroports de Paris à court terme, il offre une alternative automatisée satisfaisante et un moyen d'obtenir des prévisions sur des horizons plus longs.

Modèle Critère	ARIMA	SARIMA	Lasso	Non-paramétrique
Précision à court et moyen terme	--	-	+	++
Précision à long terme	-	--	+	++
Qualité des intervalles de confiance	-	-	+	--
Robustesse face à un historique qui s'aplatis dans le passé	+	-	--	++
Interprétabilité du modèle	+	+	++	--
Temps d'exécution	++	-	-- (avec IC) ++ (sans IC)	+

TABLE 2 – Tableau récapitulatif comparant la qualité des différents modèles, selon divers critères

Bilan personnel

D'un point de vue personnel, ce projet nous a permis de tester les performances des modèles ARIMA et SARIMA vus en cours sur des données réelles, mais aussi d'explorer et d'implémenter par nous-mêmes de nouvelles approches, en intégrant notamment des corrections calendaires. Ce projet a été une très bonne expérience de travail en groupe qui nous a permis de mettre nos connaissances au service d'une entreprise pour répondre à des problématiques concrètes, d'améliorer notre maîtrise du langage Python, et de découvrir un nouvel outil de visualisation de données, PowerBi.

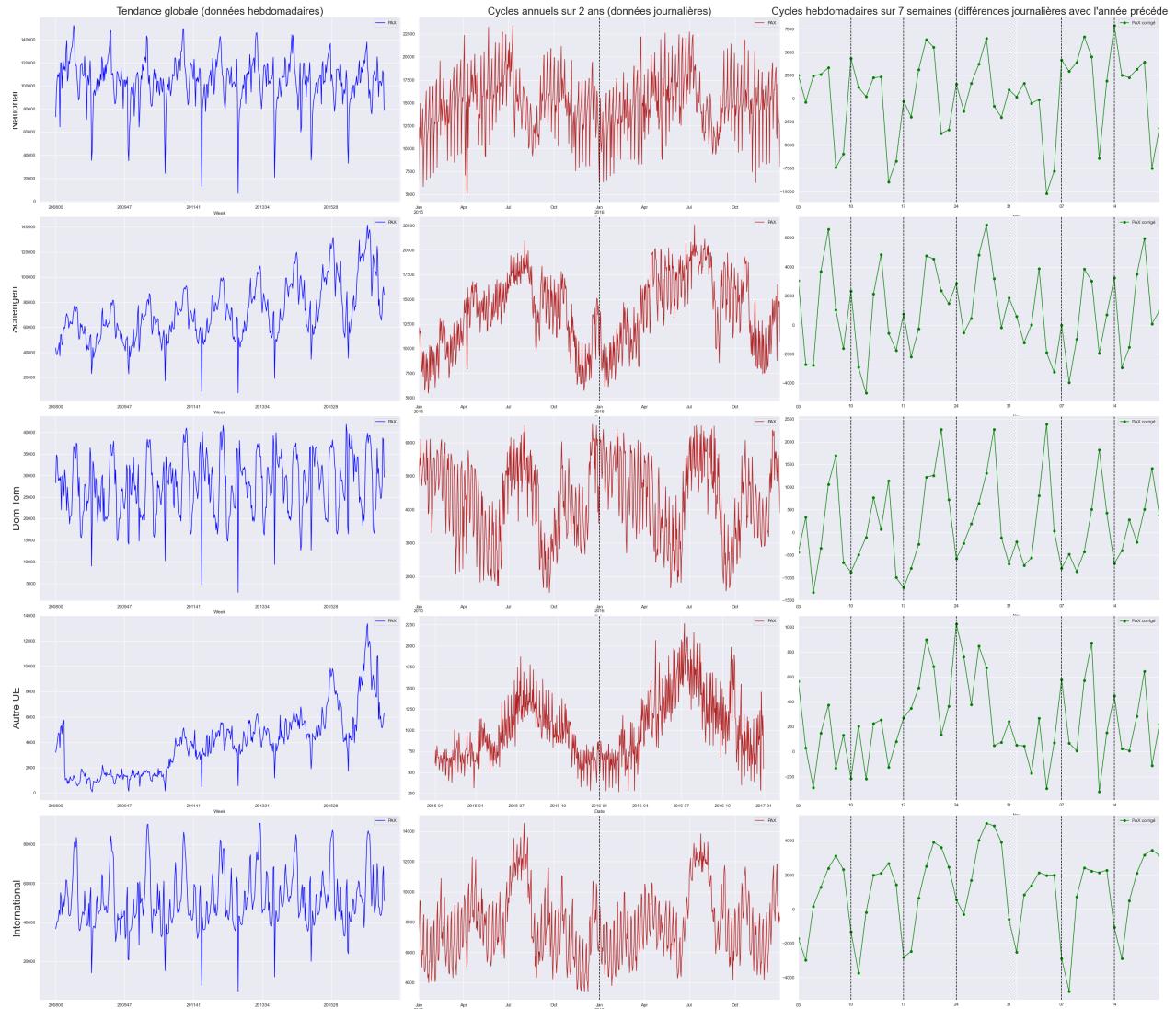
Références

- [1] Peter J. Brockwell and Richard A. Davis. *Time Series : Theory and Methods*. Springer Series of Statistics, 2009.
- [2] Christian Gourieroux et Alain Montfort. *Séries temporelles et modèles dynamiques*. Economica, 2002.
- [3] Sylvestre Tatsa. *Modélisation et prévision de la consommation horaire d'électricité au Québec*. 2013.
<https://corpus.ulaval.ca/jspui/bitstream/20.500.11794/24781/1/30329.pdf>
- [4] Joëlle Bouchard et Benoit Montreuil. *Prévisions journalières de séries temporelles saisonnières avec effets calendaires*. 2011.
http://www.simagi.polymtl.ca/congresgi/cigi2011/articles/_Bouchard-Previsions.pdf
- [5] Anestis Antoniadis, Xavier Brossat, Jairo Cugliari et Jean-Michel Poggi. *Prévision d'un processus à valeurs fonctionnelles en présence de non stationnarités. Application à la consommation d'électricité*. Journal de la Société Française de Statistique, Vol. 153 No. 2, 2012.
<https://hal.archives-ouvertes.fr/hal-00814530v1/document>
- [6] Anestis Antoniadis, Xavier Brossat, Jairo Cugliari et Jean-Michel Poggi. *Une approche fonctionnelle pour la prévision non-paramétrique de la consommation d'électricité*. Journal de la Société Française de Statistique, Vol. 155 No. 2, 2014.
<http://journal-sfds.fr/article/view/285>
- [7] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning. Data Mining, Inference and Prediction*. Springer Series of Statistics, 2009.
<https://web.stanford.edu/~hastie/Papers/ESLII.pdf>
- [8] S. Kumar, Ashok N. Srivastava. *Bootstrap prediction intervals in non-parametric regression with applications to anomaly detection*. 2012.
<https://ntrs.nasa.gov/api/citations/20130014367>

A Figures et Outil PowerBi

1. Statistiques descriptives sur les données à notre disposition :

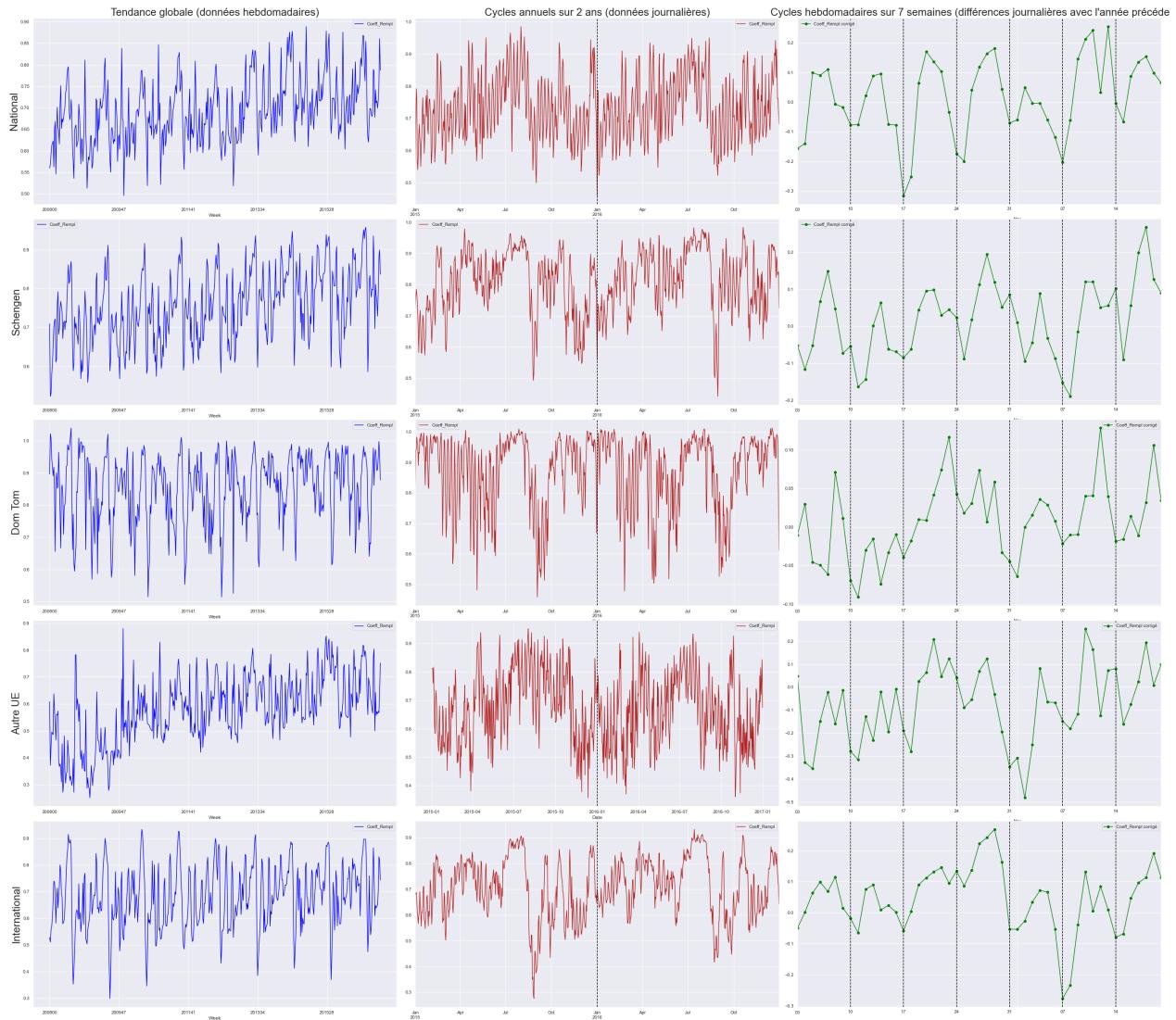
FIGURE 8 – Caractéristiques du trafic aérien en terme de nombre de passagers réalisés, sur les départs de chaque faisceau



Chaque ligne représente les caractéristiques du trafic aérien pour un faisceau (dans l'ordre National, Schengen, Dom-Tom, Autre UE et International).

- Les séries bleues correspondent aux nombres de passagers hebdomadaires entre 2008 et 2016.
- Les séries rouges correspondent aux nombres de passagers journaliers entre 2015 et 2016.
- Les séries vertes correspondent aux différences des nombres de passagers journaliers entre 2016 et 2015, représentées sur la période du 3 octobre au 20 novembre.

FIGURE 9 – Caractéristiques du trafic aérien en terme de coefficients de remplissage, sur les départs de chaque faisceau



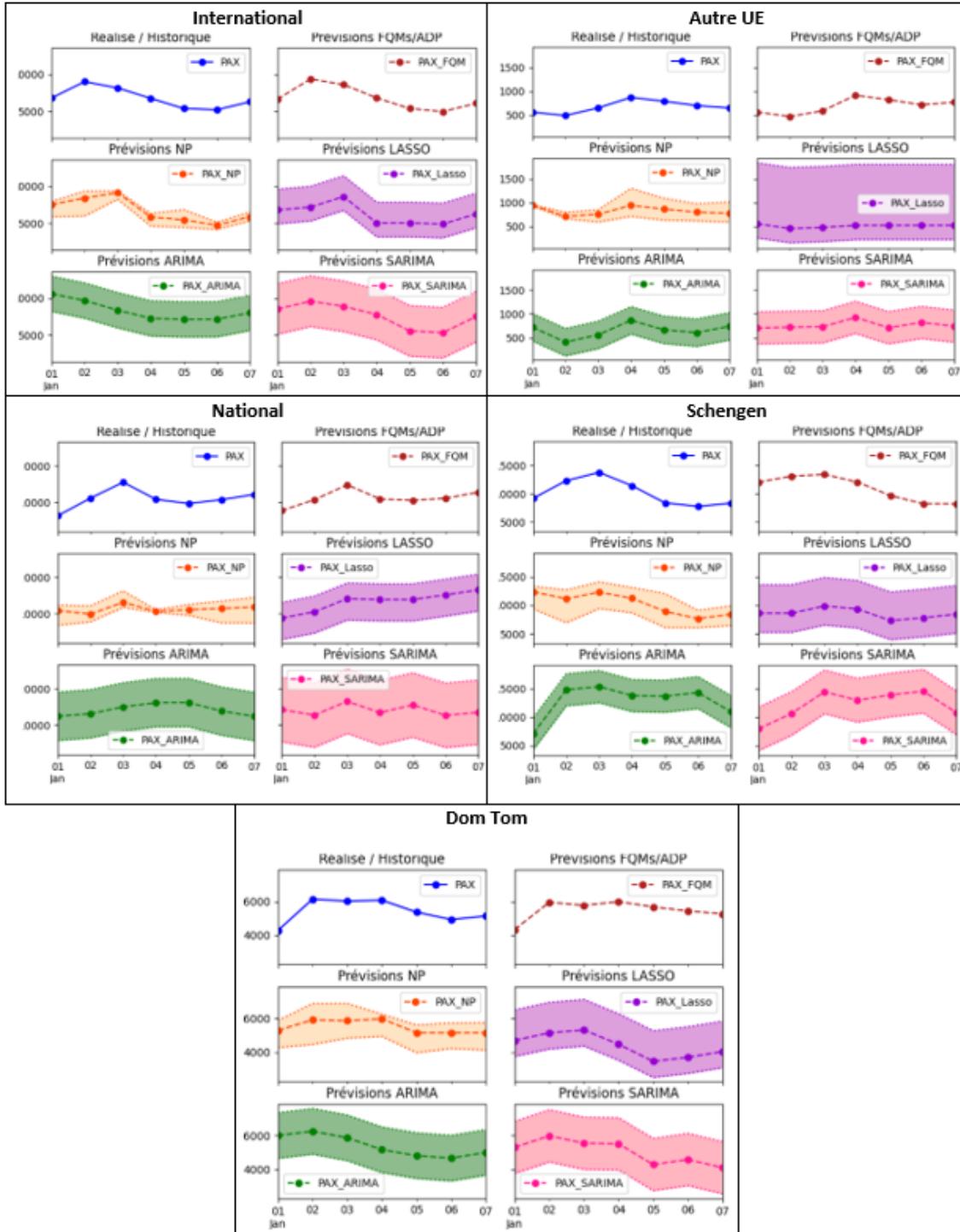
Comme précédemment, chaque ligne représente les caractéristiques du trafic aérien pour un faisceau (dans l'ordre National, Schengen, Dom-Tom, Autre UE et International).

- Les séries bleues correspondent aux taux de remplissages moyens hebdomadaires entre 2008 et 2016.
- Les séries rouges correspondent aux taux de remplissages moyens journaliers entre 2015 et 2016.
- Les séries vertes correspondent aux différences des taux de remplissages moyens journaliers entre 2016 et 2015, représentés sur la période du 3 octobre au 20 novembre.

2. Visualisation des prévisions et de leurs performances

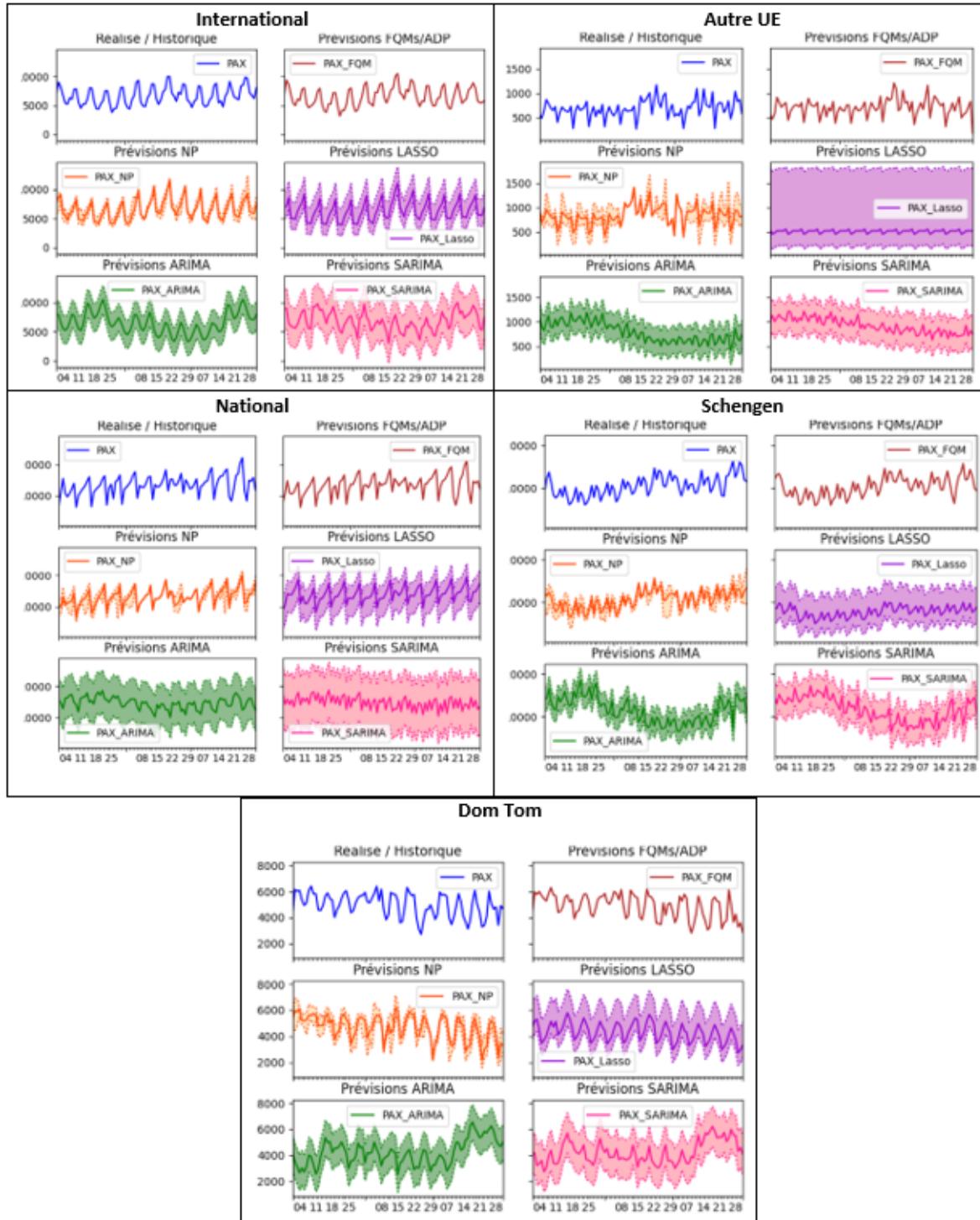
a) Prévisions de Court Terme (1 semaine) – du vendredi 1 au jeudi 7 janvier 2016

FIGURE 10 – Prévisions à court terme du nombre de passagers journaliers, par faisceau et modèles (sur les départs), avec intervalles de confiance à 95%



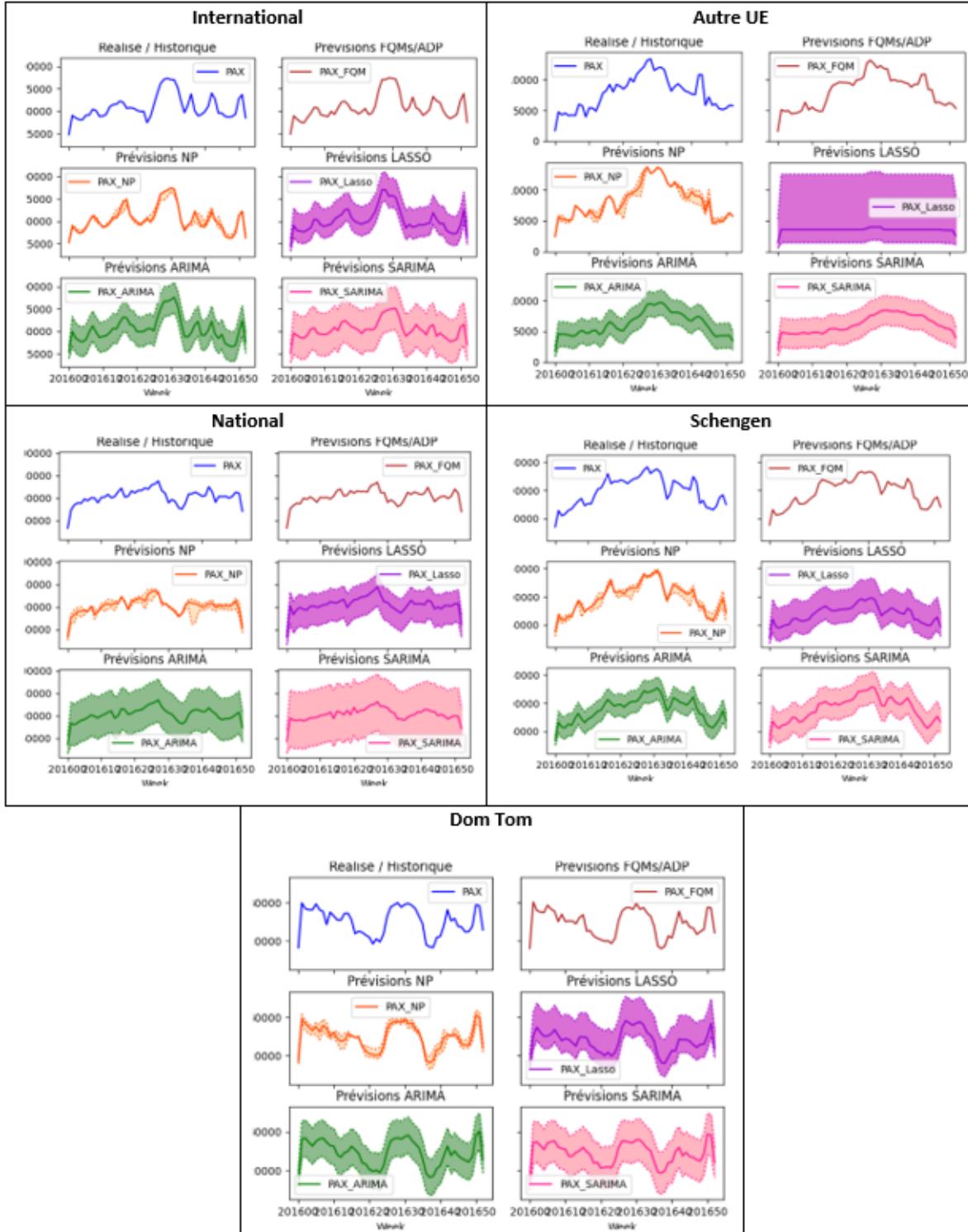
b) Prévisions de Moyen Terme (3 mois) – du 1 janvier au 31 mars 2016

FIGURE 11 – Prévisions à moyen terme du nombre de passagers journaliers, par faisceau et modèles (sur les départs), avec intervalles de confiance à 95%



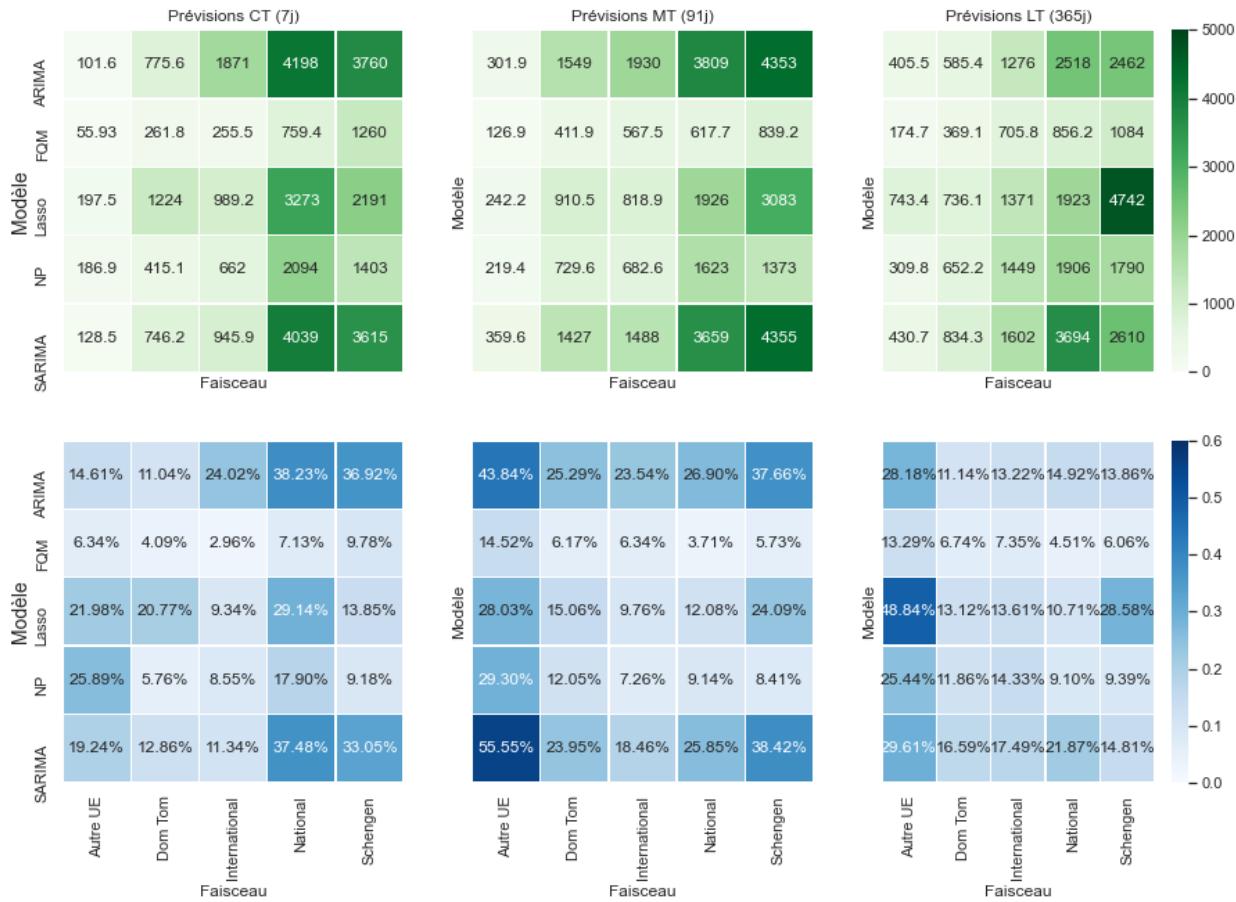
c) Prévisions de Long Terme (1 an) – du 1 janvier au 30 décembre 2016

FIGURE 12 – Prévisions à long terme du nombre de passagers journaliers (sommés par semaine), par faisceau et modèles (sur les départs), avec intervalles de confiance à 95%



d) Indicateurs de performance RMSE et MAPE des différents modèles

FIGURE 13 – Indicateurs de performances RMSE (vert) et MAPE (bleu) des différents modèles, par faisceau et horizon de prévision (sur les départs)



3. Paramètres des différents modèles :

Dans cette section, nous présentons les paramètres adoptés par les différents modèles pour réaliser les prévisions de la section précédente (on est donc toujours sur les départs).

On propose aussi quelques ordres de grandeur du temps d'exécution des différents modèles : ce sont des mesures qui n'ont été faites que sur une seule exécution, sans modifier la taille de l'historique, mais sur le même ordinateur sur lequel rien d'autre ne tournait. Ces chiffres peuvent donc surtout permettre de comparer les modèles entre eux.

Modèle	Paramètre	Description
Modèle ARIMA	p	Ordre d'autorégression : lié à la fonction d'autocorrélation totale
	d	Ordre d'intégration de la série : nombre de différenciations avant d'obtenir une série stationnaire
	q	Ordre de moyenne mobile : lié à la fonction d'autocorrélation partielle
Modèle SARIMA	P	Ordre d'autorégression saisonnier : lié à la fonction d'autocorrélation totale
	D	Ordre d'intégration saisonnier
	Q	Ordre de moyenne mobile saisonnier : lié à la fonction d'autocorrélation partielle
	s	Paramètre de saisonnalité : choisie à 12 puisque nos séries ont une saisonnalité annuelle ie 12 mois
Modèle non paramétrique	h	Hyper-paramètre de largeur de fenêtre de l'estimateur par noyau (permet d'accorder plus moins de poids aux queues de distribution)
	δ	Taille des blocs de trajectoire : comme saisonnalité à l'année, on prend 365 jours
Modèle Lasso	α	Paramètre de régularisation

TABLE 3 – Table explicative des paramètres des modèles

a) Modèle ARIMA

— Ordres ARIMA optimaux par faisceaux :

Faisceau	(p, d, q)
National	(3, 0, 2)
Schengen	(2, 1, 3)
Autre UE	(2, 1, 3)
International	(5, 1, 2)
Dom Tom	(5, 1, 1)

TABLE 4 – Ordres (p, d, q) choisis pour chaque faisceau (sur les départs) selon le critère BIC

On peut remarquer que pour tous les faisceaux, on a $d = 1$, i.e. une seule différenciation suffit à rendre notre série stationnaire, sauf pour le faisceau "National". En effet, rien qu'en le représentant graphiquement, on peut voir qu'il ne semble pas avoir de tendance.

Pour chacun de ces modèles, en regardant les p-values associées aux paramètres ϕ_p et ψ_q , souvent proches de 0, nous avons pu déduire que nos paramètres sont bien significatifs.

Également, en effectuant un test de Ljung-Box jusqu'au décalage 24, on trouve des p-valeurs supérieures à 0.05 pour nos modèles pour chaque faisceau : ainsi, nos résidus sont bien décorrélés, on peut

donc conclure que nos modèles sont valides.

— Temps d'exécution :

Dans le cas du modèle ARIMA, l'horizon de prévision n'a que très peu d'impact sur le temps d'exécution. Le temps de calcul des prévisions et des intervalles de confiance est négligeable, le plus long pour le modèle étant de trouver les paramètres adéquats pour s'ajuster au mieux sur l'historique.

Voici, à titre indicatif, les temps d'exécution avec un horizon de prévision de 7 jours :

National	Schengen	Autre UE	International	Dom Tom	Moyenne
58,54 s	60,81 s	52,87 s	44,51 s	59,69 s	55,33 s

TABLE 5 – Temps d'exécution du modèle ARIMA sur chaque faisceau (avec ic, sur les départs), sur un horizon de prévision de 7 jours

b) Modèle SARIMA

— Ordres SARIMA optimaux par faisceaux :

Faisceau	$(p, d, q), (P, D, Q)_s$
National	$(3, 0, 0), (2, 1, 0)_{12}$
Schengen	$(3, 0, 2), (2, 1, 0)_{12}$
Autre UE	$(0, 0, 2), (1, 1, 1)_{12}$
International	$(0, 0, 3), (0, 1, 1)_{12}$
Dom Tom	$(2, 0, 1), (2, 1, 0)_{12}$

TABLE 6 – Ordres $(p, d, q), (P, D, Q)_s$ choisis pour chaque faisceau (sur les départs) selon le critère BIC

Tous les meilleurs modèles sélectionnés affichent bien un paramètre de saisonnalité $s = 12$, ce qui correspond bien à ce à quoi nous nous attendions. En effet, cela confirme notre analyse de départ selon laquelle la saisonnalité annuelle est la plus marquée et la plus importante à prendre en compte (puisque dans le cadre du modèle SARIMA, il ne faut en choisir qu'une).

Toujours comme pour nos modèles ARIMA, en étudiant la significativité de nos coefficients et en effectuant un test de Ljung-Box, on conclut que nos modèles SARIMA sélectionnés sont également valides.

— Temps d'exécution :

Dans le cas du modèle SARIMA, comme pour le modèle ARIMA, l'horizon de prévision n'a que très peu d'impact sur le temps d'exécution. Le temps de calcul des prévisions et des intervalles de confiance est négligeable, le plus long pour le modèle étant de trouver les paramètres adéquats pour s'ajuster au mieux sur l'historique.

Voici, à titre indicatif, les temps d'exécution avec un horizon de prévision de 7 jours :

National	Schengen	Autre UE	International	Dom Tom	Moyenne
232,78 s = 3 min 53 s	594,92 s = 9 min 55 s	257,85 s = 4 min 17 s	661,87 s = 11 min 2 s	257,74 s = 4 min 18 s	401,03 s = 6 min 41 s

TABLE 7 – Temps d'exécution du modèle SARIMA sur chaque faisceau (avec ic, sur les départs), sur un horizon de prévision de 7 jours

c) Modèle non-paramétrique

- Taille des blocs par défaut (paramètre `tailleBlocs`) : 365 jours
- Nombre de simulations Bootstrap pour la réalisation des intervalles de confiance : 1000
- Hyper-paramètres h choisis :

Faisceau \ hPrev	7 jours	91 jours (3 mois)	365 jours (1 an)
National	14.2	13.3	10
Schengen	18	13	7
Autre UE	29	19	5
International	21	17	16
Dom Tom	24	15	8

TABLE 8 – Hyper-paramètres h choisis pour chaque faisceau et horizon de prévision (pour les départs)

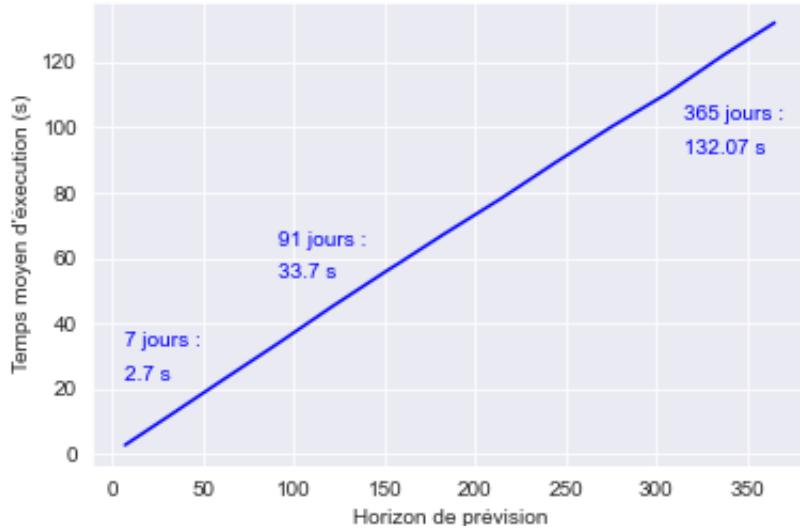
- Temps d'exécution :

Dans le modèle non-paramétrique, le temps de calcul varie très peu d'un faisceau à l'autre. Le calcul des intervalles de confiance par bootstrap n'a également que peu d'impact.

En revanche, contrairement aux trois autres modèles, l'horizon de prévision a un grand impact sur le temps de calcul, car pour chaque date de prévision le modèle calcule des corrections calendaires.

Voici, à titre indicatif, les temps d'exécution moyennés sur les faisceaux, avec différents horizons de prévision (il y a 14 points sur le graphique) :

FIGURE 14 – Temps d'exécution moyen du modèle non-paramétrique (avec ic, sur les départs), en fonction de l'horizon de prévision :



d) Modèle LASSO

- Nombre de variables explicatives disponibles au total : 300
- Nombre de simulations Bootstrap pour la réalisation des intervalles de confiance : 1000
- Hyper-paramètre α choisis :

Faisceau	α	Nombre de variables sélectionnées (sur 300)
National	0.58	109
Schengen	0.17	73
Autre UE	0.47	295
International	0.07	27
Dom Tom	0.14	104

TABLE 9 – Hyper-paramètres α choisis, et nombre de variables explicatives sélectionnées pour chaque faisceau (pour les départs)

- Temps d'exécution :

Dans le cas du modèle LASSO, comme dans les modèles ARIMA et SARIMA, l'horizon de prévision n'a pas de réel impact sur le temps d'exécution, le plus long pour le modèle étant de trouver les paramètres adaptés à l'historique passé en argument.

En revanche, contrairement à nos autres modèles, le calcul des intervalles de confiance par bootstrap tel qu'il a été implémenté est très coûteux, et change drastiquement le temps d'exécution de nos modèles.

Voici, à titre indicatif, les temps d'exécution, avec un horizon de prévision de 7 jours, en distinguant l'exécution avec et sans calcul des intervalles de confiance :

	National	Schengen	Autre UE	International	Dom Tom	Moyenne
Sans IC	1,37 s	1,31 s	2,17 s	0,93 s	1,07 s	1,37 s
Avec IC	1 423,8 s = 23min 44s	1 236,1 s = 20min 36s	2 292,4 s = 38min 12s	1 058,6 s = 17min 39s	1 304,4 s = 21min 44s	1 463,1 s = 24min 23s

TABLE 10 – Temps d'exécution du modèle Lasso sur chaque faisceau (sur les départs), sur un horizon de prévision de 7 jours

4. Outil PowerBi

Dans cette section, nous présentons plus en détail l'outil PowerBi de visualisation des prévisions, déjà présenté dans la section 4.

Notre outil PowerBi permet de créer des visuels autour des bases de données de prévisions, créés par nos programmes python. Il est possible d'actualiser les bases de données directement dans PowerBi si l'on fait d'autres prévisions.

Notre outil PowrBi propose de visualiser les prévisions faites à différents horizons, sur un même faisceau et type de mouvement. Une fenêtre permet de régler ces paramètres de manière centralisée :

FIGURE 15 – Fenêtre de réglage des paramètres de l'outil PowerBi :



Notre outil PowerBi est composé de 5 onglets. On présente ici des captures d'écran avec pour paramètres les départs du faisceau International :

— Onglet 1 : Historique

Le premier onglet permet de visualiser l'intégralité de l'historique qui a été utilisé pour réaliser les prévisions (partie gauche). Dans la partie droite, un curseur, pouvant être réglé entre 7 et 365 jours, affiche les derniers jours de cet historique, afin de pouvoir faire des comparaisons avec les prévisions présentées dans les onglets qui suivent.

— Onglet 2 à 4 : Visualisation des prévisions

Dans les trois onglets qui suivent, qui sont tous construits de la même manière, les différentes prévisions, à court, moyen et long terme, sont présentées. Dans la partie gauche, le réalisé et les différentes prévisions (y compris les prévisions des Aéroports de Paris) sont représentées sur la période de prévision considérée. Dans la partie droite, différents indicateurs de performances ont été calculés, à savoir le RMSE et le MAPE, et présentés dans un histogramme.

— Onglet 5 : Performance des prévisions tous faisceaux compris

Dans ce dernier onglet, le paramètre Faisceau n'a plus d'importance, seul le mouvement choisi importe. Cet onglet résume les différents indicateurs RMSE et MAPE qui ont été calculés par horizon de prévision, modèle et faisceau. Cela permet d'avoir un résumé des performances des modèles, permettant une comparaison directe de nos modèles, en distinguant les faisceaux et horizons de prévisions.

FIGURE 16 – Onglet 1 : Historique

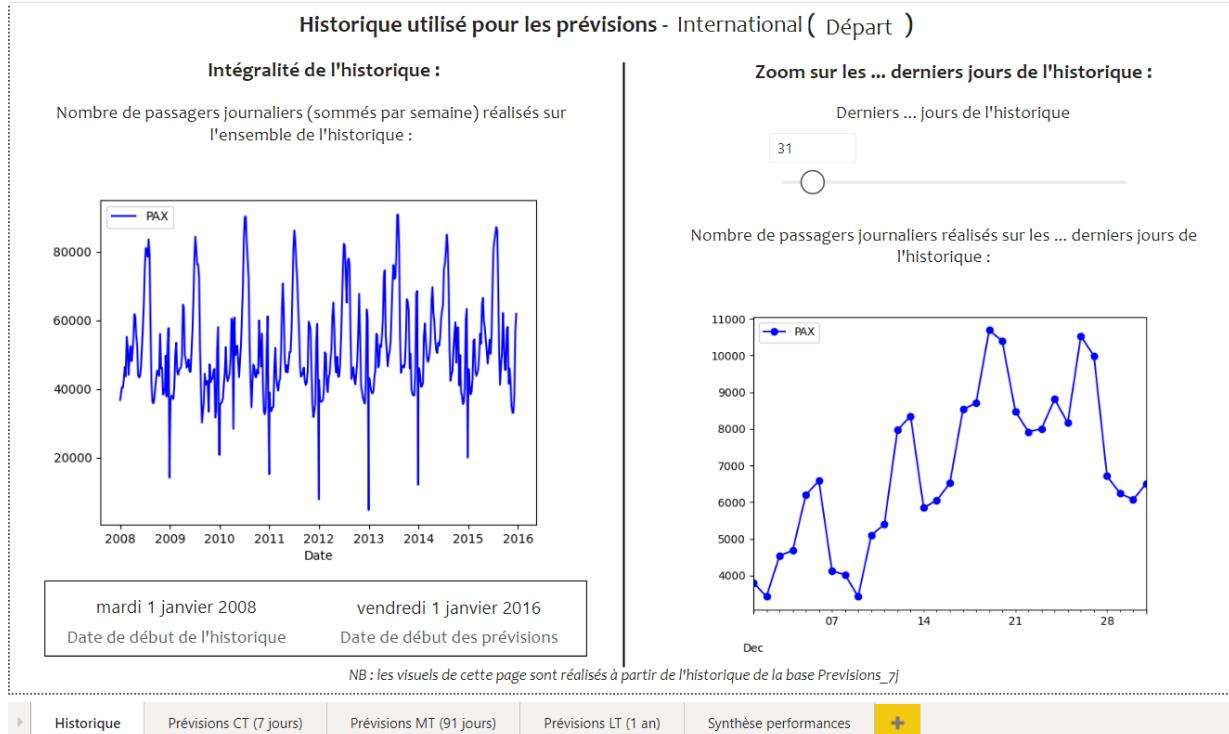


FIGURE 17 – Onglet 2 : Visualisation des prévisions à Court Terme (7 jours)

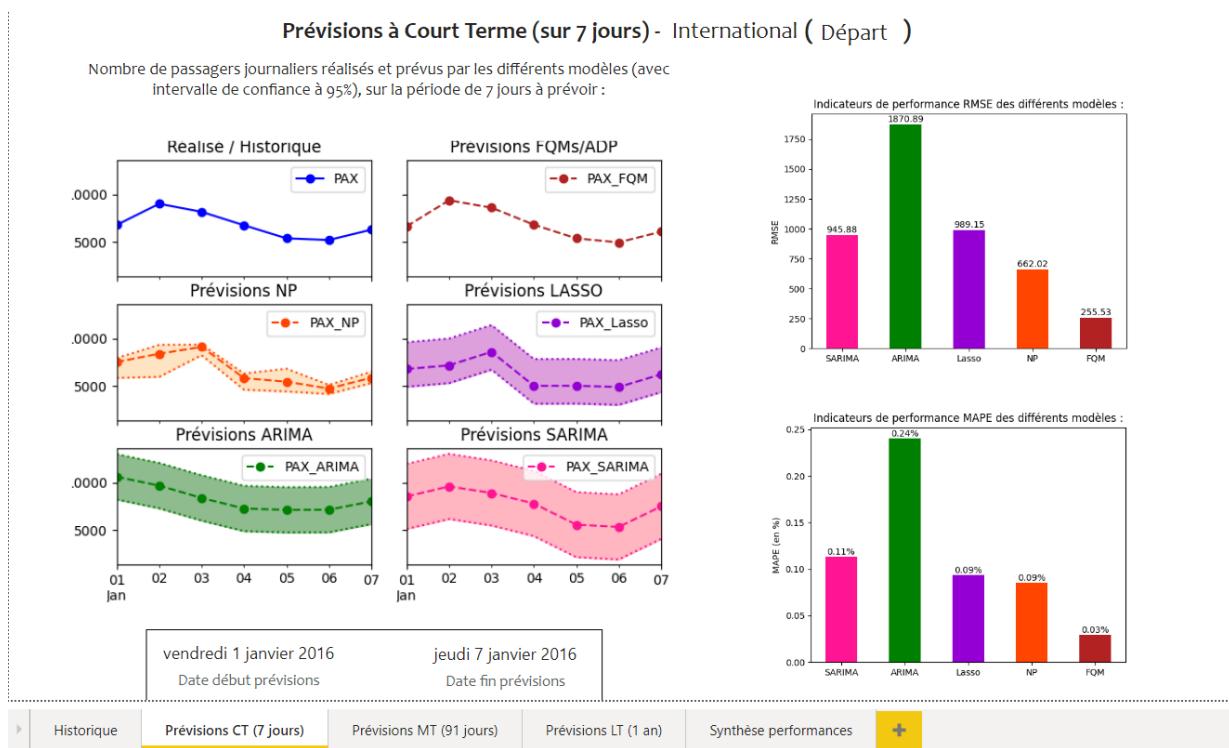


FIGURE 18 – Onglet 3 : Visualisation des prévisions à Moyen Terme (91 jours)

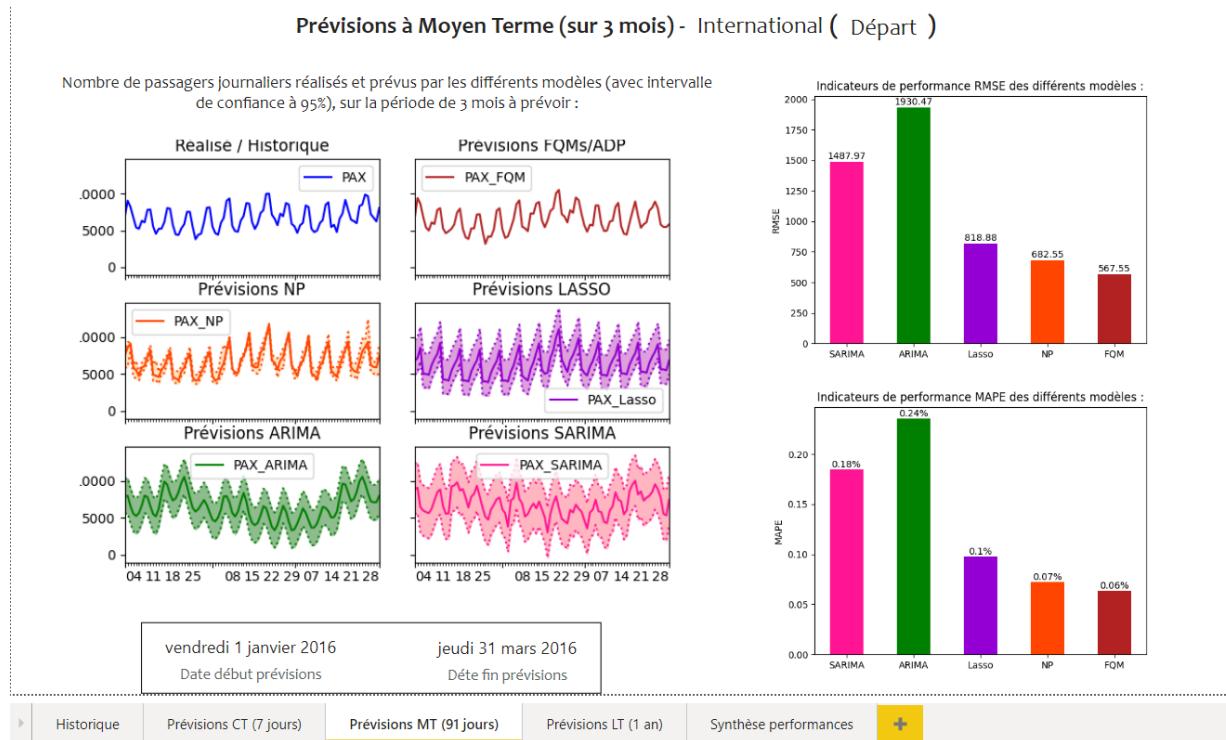


FIGURE 19 – Onglet 4 : Visualisation des prévisions à Long Terme (1 an)

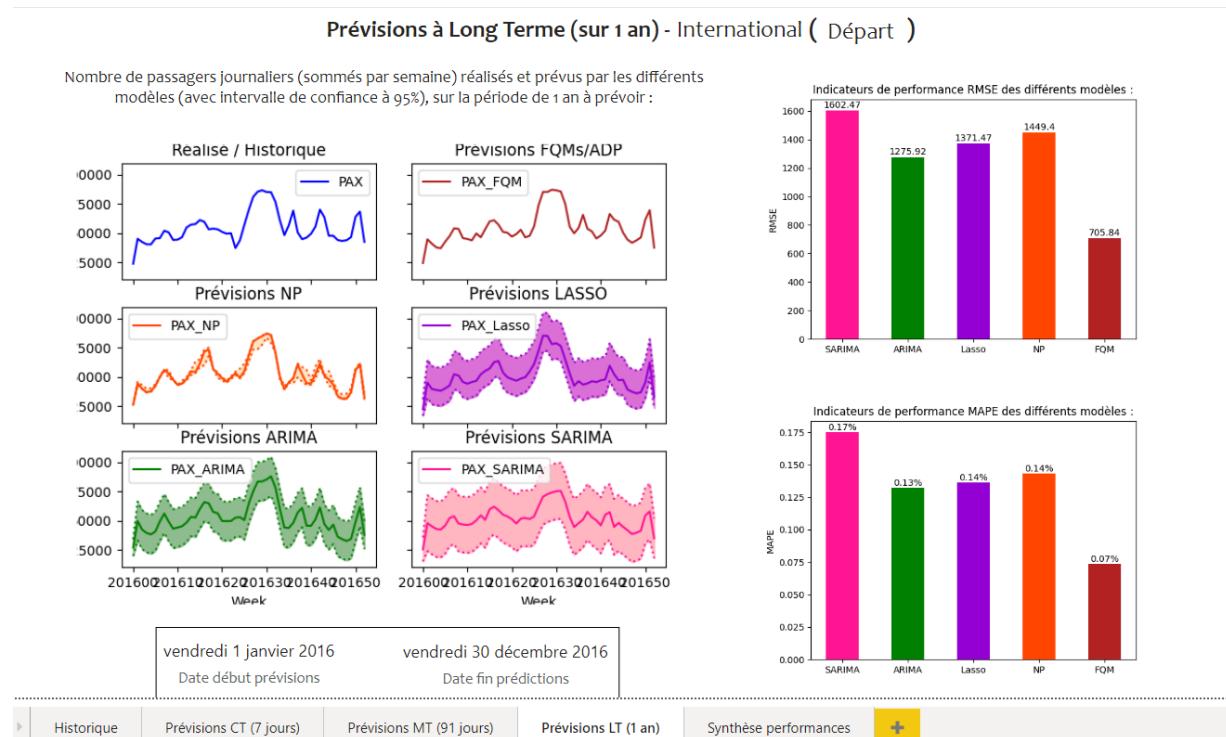
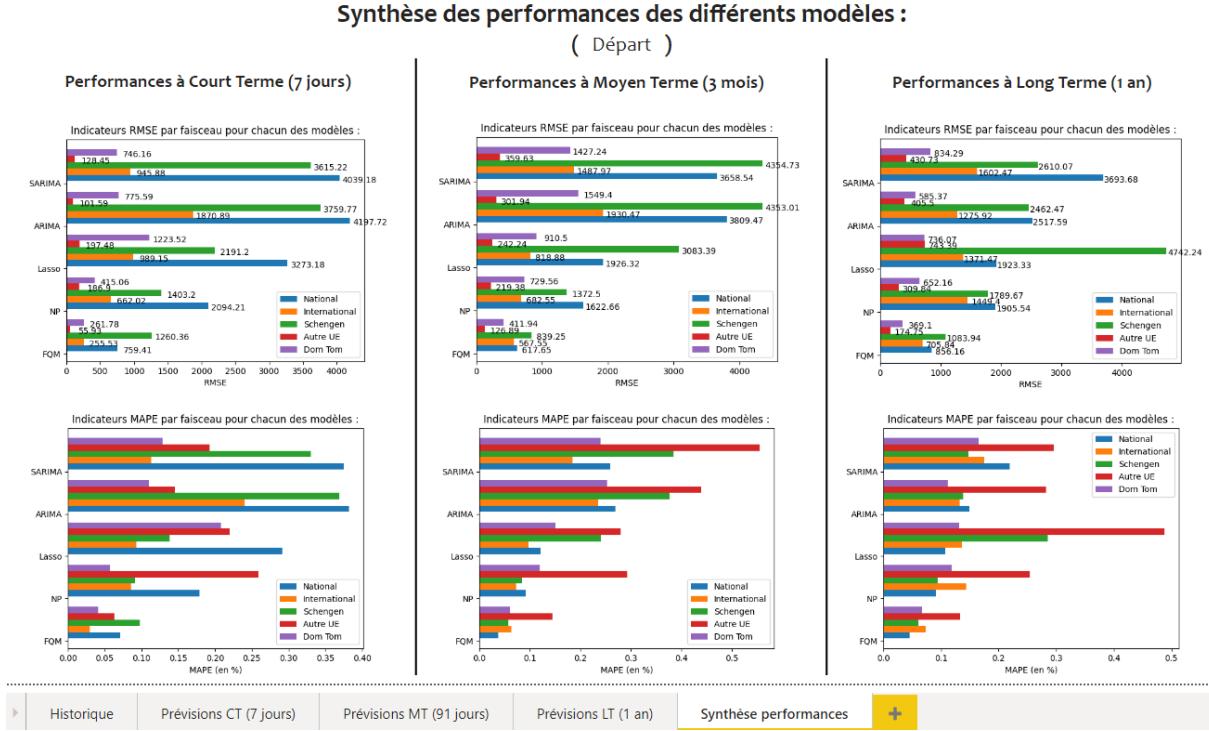


FIGURE 20 – Onglet 5 : Synthèse des performances tous faisceaux compris



B Code des différents modèles

Chacun des modèles a été implémenté sous Python dans un fichier dédié, sous la forme finale d'une fonction `previsions_modele`.

Cette fonction réalise des prévisions sur un seul faisceau, type de mouvement, et pour un seul horizon de prévisions. Elle prend en argument un historique (sur un seul faisceau et type de mouvement), un horizon de prévision, ainsi que d'autres paramètres suivant les besoins du modèle, comme la base `Calendrier`, ...

Nous n'allons présenter ici que les fonctions précédemment décrites (avec leurs fonctions intermédiaires le cas échéant). Ces fonctions ont ensuite été appelées dans un même fichier, à exécuter par l'utilisateur, pour obtenir toutes les prévisions qu'il souhaite, une fois qu'il a spécifié les horizons de prévision ainsi que les dates de début et de fin de son historique voulu.

Import des packages nécessaires à l'implémentation des modèles :

```
import math
from datetime import timedelta

import pandas as pd
import numpy as np
```

1. Modèle ARIMA

Pour implémenter les modèles ARIMA et SARIMA, nous nous sommes aidés de quelques fonctions pré-existantes sur Python : `auto_arima` qui sélectionne le meilleur modèle valide par "grid search" en tenant en compte le critère d'information qu'on lui demande (AIC par défaut si aucun critère n'est spécifié), et `SARIMAX`, qui permet de simuler un ARIMA ou un SARIMA une fois que l'on rentre les ordres souhaités.

```
# !pip install pmdarima
from pmdarima import auto_arima
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Nous avons ainsi pu créer les fonctions suivantes :

- `ordre_ARIMA` : renvoie les ordres ARIMA optimaux lorsque l'on rentre en argument l'historique étudié, et les dates de fin et de début de la période visualisée :

```
def ordre_ARIMA(histoMod, dateDebMod, dateFinMod):

    histoMod = histoMod.reset_index(drop = True)
    coupe = np.where(histoMod['Date'] == dateFinMod)[0][0]
    train = histoMod[histoMod.index <= coupe]

    stepwise_fit = auto_arima(train['PAX'],
                              error_action='ignore',
                              suppress_warnings=True,
                              stepwise=True,
                              information_criterion = 'bic')

    return(stepwise_fit.order)
```

- `previsions_ARIMA` : une fois que l'on a rentré les mêmes arguments que ceux de la fonction précédente, ainsi que l'horizon de prévision souhaité et le niveau de précision de l'intervalle de confiance voulu, on obtient en retour un nouveau dataframe de `hPrev` lignes contenant les colonnes du dataframe initial ainsi que trois nouvelles colonnes, l'une avec la prévision du nombre de passagers, et deux contenant les bornes de l'intervalle de confiance :

```

def previsions_ARIMA(histoMod, dateDebMod, dateFinMod, hPrev, ic = 0.95) :

    histoMod = histoMod.reset_index(drop = True)
    coupe = np.where(histoMod['Date'] == dateFinMod)[0][0]
    train = histoMod[histoMod.index <= coupe]

    ordres = ordre_ARIMA(histoMod, dateDebMod, dateFinMod)
    model = SARIMAX(train['PAX'], order = ordres)
    model_fit = model.fit(disp = -1)

    prediction = model_fit.get_prediction(start = len(train) - hPrev + 1, end =
len(train), freq = 'A')
    PrevisionsARIMA = pd.DataFrame(histoMod[['ArrDep', 'Faisceau']].head(hPrev))
    new_dates = pd.date_range(start = dateFinMod + timedelta(1), end = dateFinMod
+ timedelta(hPrev))
    PrevisionsARIMA['Date'] = new_dates
    PrevisionsARIMA['PAX_ARIMA'] = prediction.predicted_mean.values

    if ic == 0 :
        PrevisionsARIMA['IC'+str(int(ic*100))+'_low_ARIMA'] = [0 for k in range(
hPrev)]
        PrevisionsARIMA['IC'+str(int(ic*100))+'_up_ARIMA'] = [0 for k in range(
hPrev)]
    else :
        PrevisionsARIMA['IC'+str(int(ic*100))+'_low_ARIMA'] = prediction.conf_int(
alpha = 1-ic)['lower PAX'].values
        PrevisionsARIMA['IC'+str(int(ic*100))+'_up_ARIMA'] = prediction.conf_int(
alpha = 1-ic)['upper PAX'].values

    return(PrevisionsARIMA)

```

En outre, pour tester la validité de nos modèles ARIMA, nous avons fait appel à quelques fonctions et modules :

```

import statsmodels.api as sm
from scipy import stats

```

Munis de ces derniers, nous avons pu construire les deux fonctions suivantes :

— `validite_residus_ARIMA` : une fois que l'on rentre l'historique, ses dates de début et de fin, et les ordres ARIMA du modèle que l'on souhaite tester, on obtient en retour un message nous disant si les résidus du modèle sont corrélés ou non :

```

def validite_residus_ARIMA(histoMod, dateDebMod, dateFinMod, p, d, q) :

    bool = True
    histoMod = histoMod.reset_index(drop = True)
    coupe = np.where(histoMod['Date'] == dateFinMod)[0][0]
    train = histoMod[histoMod.index <= coupe]
    model = sm.tsa.ARIMA(histoMod["PAX"], (p,d,q)).fit(disp=0)

    lj = sm.stats.acorr_ljungbox(model.resid, lags = 24)
    pval_corrigee = stats.chi2.sf(lj[-1], 24 - p - q)

    for i in pval_corrigee :
        if i < 0.05 :
            bool = False

    if bool == True :
        print('Les résidus ne sont pas corrélés')
    else :
        print('Les résidus sont corrélés')

```

— `validite_param_ARIMA` : une fois que l'on a rentré les mêmes arguments que ceux de la fonction précédente, on obtient un message nous indiquant si les paramètres du modèle sont significatifs ou non :

```

def validite_param_ARIMA(histoMod, dateDebMod, dateFinMod, p, d, q) :

    bool = True
    histoMod = histoMod.reset_index(drop = True)
    coupe = np.where(histoMod['Date'] == dateFinMod)[0][0]
    train = histoMod[histoMod.index <= coupe]
    model = sm.tsa.ARIMA(histoMod["PAX"], (p,d,q)).fit(disp=0)

    if p == 0 :
        if model.pvalues[q] > 0.05 :
            bool = False
    elif q == 0 :
        if model.pvalues[p] > 0.05 :
            bool = False
    else :
        if model.pvalues[p] > 0.05 :
            bool = False
        if model.pvalues[p+q] > 0.05 :
            bool = False

    if bool == True :
        print('Les paramètres sont significatifs')
    else :
        print('Les paramètres ne sont pas significatifs')

```

2. Modèle SARIMA

Pour le modèle SARIMA, nous avons construit les fonctions ci-dessous, semblables à celles que nous avons créées pour le modèle ARIMA :

- ordre_SARIMA : renvoie les ordres SARIMA optimaux lorsque l'on rentre en argument l'historique étudié, et les dates de fin et de début de la période visualisée :

```

def ordre_SARIMA(histoMod, dateDebMod, dateFinMod):

    histoMod = histoMod.reset_index(drop = True)
    coupe = np.where(histoMod['Date'] == dateFinMod)[0][0]
    train = histoMod[histoMod.index <= coupe]

    stepwise_fit = auto_arima(train['PAX'], start_p=0, start_q=0,
                             max_p=3, max_q=3, m=12,
                             start_P=0, seasonal=True,
                             d=None, D=1, trace=True,
                             error_action='ignore',
                             suppress_warnings=True,
                             stepwise=True,
                             information_criterion = 'bic')

    return stepwise_fit.order, stepwise_fit.seasonal_order

```

- previsions_SARIMA : comme avec previsions_ARIMA, on rentre l'historique des vols qui nous intéresse, ses dates de fin et de début, ainsi que l'horizon de prévision et le niveau de précision de l'intervalle de confiance souhaités, et on obtient à la sortie un nouveau jeu de données de hPrev lignes contenant les colonnes de l'historique initial ainsi que trois nouvelles colonnes, l'une avec la prévision du nombre de passagers, et deux contenant les bornes de l'intervalle de confiance :

```

def previsions_SARIMA (histoMod, dateDebMod, dateFinMod, hPrev, ic) :

    histoMod = histoMod.reset_index(drop = True)
    coupe = np.where(histoMod['Date'] == dateFinMod)[0][0]
    train = histoMod[histoMod.index <= coupe]

    ordres, ordres_sais = ordre_SARIMA(histoMod, dateDebMod, dateFinMod)

```

```

model = SARIMAX(train['PAX'], order = ordres, seasonal_order = ordres_sais)
model_fit = model.fit(disp = -1)

prediction = model_fit.get_prediction(start = len(train) - hPrev + 1, end =
len(train), freq = 'A')
PrevisionsSARIMA = pd.DataFrame(histoMod[['ArrDep', 'Faisceau']]).head(hPrev)
new_dates = pd.date_range(start = dateFinMod + timedelta(1), end = dateFinMod +
timedelta(hPrev))
PrevisionsSARIMA['Date'] = new_dates
PrevisionsSARIMA['PAX_SARIMA'] = prediction.predicted_mean.values

if ic == 0 :
    PrevisionsSARIMA['IC'+str(int(ic*100))+'_low_SARIMA'] = [0 for k in range(
hPrev)]
    PrevisionsSARIMA['IC'+str(int(ic*100))+'_up_SARIMA'] = [0 for k in range(
hPrev)]
else :
    PrevisionsSARIMA['IC'+str(int(ic*100))+'_low_SARIMA'] = prediction.
conf_int(alpha = 1-ic)['lower PAX'].values
    PrevisionsSARIMA['IC'+str(int(ic*100))+'_up_SARIMA'] = prediction.conf_int(
alpha = 1-ic)['upper PAX'].values

return(PrevisionsSARIMA)

```

3. Modèle Non-Paramétrique

L'implémentation du modèle non-paramétrique se fait à l'aide de plusieurs fonctions intermédiaires :

- `infos_blocs` : crée les blocs à partir de l'historique, et renvoie toutes les données nécessaires par la suite (les blocs, leur version centrée réduite, le dernier bloc seul centré réduit, les moyennes et écarts-types des blocs, et leurs distances avec le dernier bloc) :

```

def infos_blocs (histoMod,tailleBlocs) :
"""
Parameters
-----
histoMod : DataFrame
    Historique du trafic sur lequel faire les blocs
tailleBlocs : int
    taille des blocs à réaliser

Returns
-----
Bloc , Blocs_CR , LastBlocs_CR : DataFrame
    Blocs , blocs centrés réduits et dernier bloc centré réduit
Stats : DataFrame
    Moyennes et Ecart-types de chacun des blocs
distances : array
    Distances entre chacun des blocs centrés réduits avec le dernier bloc centré r
éduit

"""
as_strided = np.lib.stride_tricks.as_strided

Bloc = pd.DataFrame(as_strided(histoMod["PAX"], (len(histoMod["PAX"])-(tailleBlocs-1) , tailleBlocs) , (histoMod["PAX"].values.strides * 2)))

Stats = pd.DataFrame()
Stats['Mean'] = Blocs.mean(axis=1)
Stats['Stds'] = Blocs.std(axis=1)

Bloc_CR = (Bloc - np.array(Stats['Mean']).reshape(-1,1)) / np.array(Stats['Stds'])
Bloc_CR .reshape(-1,1)

```

```

LastBloc_CR = pd.DataFrame(np.array(Blocs_CR)[-1].reshape(1,-1))

# Calcul des distances entre chaque bloc (utiles ensuite pour le calcul des poids)
:
distances = np.sum((np.array>LastBloc_CR)-np.array(Blocs_CR[:-1]))**2, axis=1)

return (Blocs, Blocs_CR, LastBloc_CR, Stats, distances)

```

— `previsions_NP_h_fixe` : réalise des prédictions pour un hyper-paramètre `h` donné

```

def previsions_NP_h_fixe (histoMod, Calendrier, dateFinMod, infosBlocs , hPrev, h,
ic = 0, tailleBlocs = 365) :
"""
Parameters
-----
histoMod, Calendrier, dateFinMod, hPrev, tailleBlocs, ic :
    idem que dans la fonction previsions_NP
infosBlocs : tuple
    Contient toutes les infos des blocs déjà calculées : Blocs, Blocs_CR,
LastBloc_CR, Stats, distances
h : int
    hyper paramètre h : largeur de la fenêtre

Returns
-----
PrevisionsNP : DataFrame
    Prévisions journalières du modèle (contient 'PAX_NP', 'Date', 'Faisceau', ,
ArrDep')
    + intervalle de confiance 'ICic_low_NP' et 'ICic_up_NP' si ic != 0

"""

```

```

Blocs, Blocs_CR, LastBloc_CR, Stats, distances = infosBlocs

# Calcul des poids en comparant la similarité des motifs des blocs (noyau gaussien
):
weights = pd.DataFrame( (1/math.sqrt(2*math.pi)**tailleBlocs) * np.exp(- distances
/ (2*h)))

# Calcul des prévisions une par une :
PrevisionsNP = pd.DataFrame()
datePrev = dateFinMod

for horizonPrev in range(1,hPrev+1) :
    datePrev += timedelta(days=1)

    # Correction des poids en utilisant le calendrier :
    CalPrev = np.array(Calendrier[Calendrier['Date']==datePrev])
    histoPrev = histoMod[tailleBlocs - 1 + horizonPrev : ]

    indJourPrev = np.array(histoPrev["Pont_LunF"]) == CalPrev[:,5]
    colHistoPrev = ["Pont_MarF", "Pont_Mer1F", "Pont_Mer2F", "Pont_JeuF", "Pont_VenF",
"Vac_Toussaint", "Vac_Noel", "Vac_Hiver_A", "Vac_Hiver_B", "Vac_Hiver_C", "
Vac_Printemps_A", "Vac_Printemps_B", "Vac_Printemps_C", "Vac_Ete"]
    indCalPrev = [6,7,8,9,10,13,14,15,16,17,18,19,20,21]
    for k in range(len(indCalPrev)) :
        indJourPrev = indJourPrev & (np.array(histoPrev[colHistoPrev[k]]) ==
CalPrev[:,indCalPrev[k]])

        # On ajoute une correction avec le jour de la semaine uniquement si cela n'
annule pas tous les poids :
        indJourPrev2 = indJourPrev & (np.array(histoPrev["JourSem"]) == CalPrev[:,2])
        if sum(indJourPrev2) != 0 :
            indJourPrev = indJourPrev2

    indJourPrev = indJourPrev.reshape(-1,1)

```

```

weightsPrev = indJourPrev * np.array(weights)[:1+len(weights)-horizonPrev]

weightsPrev = np.nan_to_num(weightsPrev) #permet de remplacer les éventuels
Nan par 0 (nécessaire pour l'échantillonnage)
weightsPrev = pd.DataFrame(weightsPrev)

# Normalisation des poids :
s = np.sum(weightsPrev , axis = 0)
Sim = weightsPrev / s

# Calcul de la prévision :
histoMod_CR = (np.array(histoMod['PAX'][tailleBlocs-1+horizonPrev:]).reshape
(-1,1) - np.array(Stats['Mean'][ : - horizonPrev]).reshape(-1,1)) / np.array(Stats
['Stds'][ : - horizonPrev]).reshape(-1,1)
histoMod_RS = histoMod_CR * np.array(Stats['Stds'])[-1].reshape(-1,1) + np.
array(Stats['Mean'])[-1].reshape(-1,1)
UnePrev = np.sum(histoMod_RS*np.array(Sim))

# si ic != 0 : calcul d'un intervalle de confiance par méthode bootstrap :
if ic != 0 :
    # Tirage aléatoire avec remise en prenant pour probas : weightsPrev
    B = 1000 #Taille de l'échantillon Bootstrap

    echantillon = np.random.choice(histoMod_CR.reshape(1,-1)[0] , size = B,
replace = True, p=np.array(Sim).reshape(1,-1)[0])
    echantillon = echantillon * np.array(Stats['Stds'])[-1].reshape(1,-1)[0] +
np.array(Stats['Mean'])[-1].reshape(1,-1)[0]

    residus = echantillon - UnePrev

    # On prend ensuite simplement les quantiles de l'échantillon des résidus
simulé par bootstrap
    p_l = ((1 - ic)/2) * 100
    lower = np.percentile(residus , p_l)
    p_u = ((1 + ic)/2) * 100
    upper = np.percentile(residus , p_u)

    # Ajout de la prévision et de son intervalle de confiance à la table
finale :
    UnePrev = pd.DataFrame(data={"PAX_NP" : [UnePrev] , 'IC'+str(int(ic*100))+',
_low_NP' : [UnePrev+lower] , 'IC'+str(int(ic*100))+'_up_NP' : [UnePrev+upper] })
    PrevisionsNP = pd.concat([PrevisionsNP , pd.concat([UnePrev , pd.DataFrame
([datePrev]) , pd.DataFrame(histoMod[["ArrDep" , "Faisceau"]]).head(1).reset_index
().drop(columns = ['index'])] , axis = 1)])
```

else :

```

        # Ajout de la prévision à la table finale :
        UnePrev = pd.DataFrame(data=[UnePrev],columns = ["PAX_NP"])
        PrevisionsNP = pd.concat([PrevisionsNP , pd.concat([UnePrev , pd.DataFrame
([datePrev]) , pd.DataFrame(histoMod[["ArrDep" , "Faisceau"]]).head(1).reset_index
().drop(columns = ['index'])] , axis = 1)])
```

```

return PrevisionsNP.rename(columns = {0 : "Date"})
```

— rmse : calcule l'erreur rmse entre deux listes de valeurs (utilisé pour le choix de l'hyper-paramètre h)

```

def rmse (serie1 , serie2) :
"""
Parameters
-----
serie1, serie2 : list
    Séries de valeurs (prédites et réelles ici) utilisées pour calculer le RMSE

Returns
```

```

-----
rmse : float
    valeur de l'erreur RMSE entre les deux séries passées en argument
"""
rmse = 0
n = len(serie1)

for i in range(n) :
    rmse += (serie1[i]-serie2[i])**2

return math.sqrt(rmse/n)

```

— `meilleur_h` : détermine le meilleur paramètre h. Pour cela, on commence par sélectionner des périodes de test, qui sont les périodes présentes dans l'historique, qui commencent aux mêmes dates que la période à prédire, mais pour des années antérieures (de sorte qu'il y ait toujours au moins un an de dispo dans l'histo). Ensuite, pour chaque période de test, on cherche le meilleur h parmi les différents candidats. On prend enfin la moyenne des meilleures h sélectionnées.

```

def meilleur_h (histoMod, Calendrier, dateFinMod, hPrev, tailleBlocs) :
"""
Parameters
-----
histoMod, Calendrier, dateFinMod, hPrev, tailleBlocs :
    idem que dans la fonction previsions_NP et previsions_NP_h_fixe

Returns
-----
h : int
    Meilleur h qui permet de faire les meilleures prédictions
"""
candidats_h = [i for i in range(5,55,5)]

meilleurs_h = [] # Liste qui contiendra les meilleurs h retenus pour chaque période testée

nb = len(histoMod)//365 - 1 # Nombre de périodes testées

for i in range(1,nb) :

    # On choisit d'essayer de prédire la même période k années avant :
    dateFinMod2 = dateFinMod - timedelta(days=i*365)

    histoMod2 = histoMod[histoMod['Date']<=dateFinMod2]
    infosBlocs2 = infos_blocs (histoMod2,tailleBlocs)

    realise = histoMod[(histoMod['Date']>dateFinMod2)&(histoMod['Date']<=
dateFinMod2+timedelta(days=hPrev))]
    realise = list(realise['PAX'])

    # Test de chacun des candidats h, et choix de celui avec la plus petite erreur
    RMSE :

    meilleur_h = candidats_h[0]
    previsions = previsions_NP_h_fixe (histoMod2, Calendrier, dateFinMod2,
infosBlocs2 , hPrev, meilleur_h)
    meilleure_erreur = rmse(realise, list(previsions['PAX_NP']))

    for k in range(1,len(candidats_h)) :
        h = candidats_h[k]
        previsions = previsions_NP_h_fixe (histoMod2, Calendrier, dateFinMod2,
infosBlocs2 , hPrev, h)
        erreur = rmse(realise, list(previsions['PAX_NP']))

        if erreur < meilleure_erreur :

```

```

        meilleur_h = h
        meilleure_erreur = erreur
        meilleurs_h.append(meilleur_h)

    return sum(meilleurs_h)/len(meilleurs_h)

```

Enfin, voici la fonction finale `previsions_NP`, qui réalise les prédictions souhaitées après avoir choisir le meilleur hyper-paramètre `h` :

```

def previsions_NP (histoMod, Calendrier, dateDebMod, dateFinMod, hPrev, ic = 0.95,
tailleBlocs = 365) :
"""
Parameters
-----
histoMod : DataFrame
    Historique du trafic journalier, sur un seul faisceau et un seul type de mouvement,
    contenant : 'ArrDep', 'Faisceau', 'Date', 'PAX'
Calendrier : DataFrame
    Calendrier à utiliser pour déterminer les groupes de blocs
dateDebMod : datetime64[ns]
    Date de début de l'historique
dateFinMod : datetime64[ns]
    Date de fin de l'historique
hPrev : int
    Nombre de jours pour lesquels faire une prédiction du trafic
tailleBlocs : int, optional
    Taille des blocs du modèle. The default is 365 days.
ic : float, optional
    Correspond au seuil de l'intervalle de confiance souhaité (mettre 0 pour ne pas
    calculer d'intervalle de confiance). The default is 0.95.

Returns
-----
PrevisionsNP : DataFrame
    Prévisions journalières du modèle (contient 'PAX_NP', 'Date', 'Faisceau', 'ArrDep')
+ intervalle de confiance 'IC_ic_inf_NP' et 'IC_ic_sup_NP' si ic != 0
"""

# Sélection colonnes intérêt Calendrier + dans les colonnes Pont_LunF, ..., Pont_VenF,
# on remplace par 0 si c'est les vacances en même temps
colonnesCalendrier = ["Date", "Mois", "JourSem", "Semaine", "Semaine_Perso", "Pont_LunF", "Pont_MarF", "Pont_Mer1F", "Pont_Mer2F", "Pont_JeuF", "Pont_VenF", "Pont_SamF", "Pont_DimF", "Vac_Toussaint", "Vac_Noel", "Vac_Hiver_A", "Vac_Hiver_B", "Vac_Hiver_C", "Vac_Printemps_A", "Vac_Printemps_B", "Vac_Printemps_C", "Vac_Ete"]
Calendrier = np.array(Calendrier[colonnesCalendrier])
vacances = Calendrier[:,13:].sum(axis=1) == 0 #Contient True si on n'est pas en vacances
et False si on est en vacances
for i in range(5,11) :
    Calendrier[:,i] *= vacances
Calendrier = pd.DataFrame(Calendrier, columns=colonnesCalendrier)

# Augmentation de l'historique avec le calendrier :
histoMod = pd.merge(histoMod, Calendrier, left_on = ['Date'], right_on = ['Date'], how =
    'left')
histoMod = histoMod.sort_values(by='Date')

# Création des blocs et des informations utiles :
infosBlocs = infos_blocs (histoMod,tailleBlocs)

# Choix de la meilleure largeur de la fenêtre : on fait appel à une recherche de type
cross validation
h = meilleur_h (histoMod, Calendrier, dateFinMod, hPrev, tailleBlocs)

# Réalisation des prévisions avec le h sélectionné précédemment :
PrevisionsNP = previsions_NP_h_fixe (histoMod, Calendrier, dateFinMod, infosBlocs ,

```

```

    hPrev, h, ic, tailleBlocs)

    return PrevisionsNP

```

4. Modèle LASSO

Pour le modèle LASSO, nous avons implémenté la fonction `previsions_Lasso` qui permet de réaliser les prédictions souhaitées avec les intervalles de prédiction demandés. Celle-ci appelle une seconde fonction détaillée ci-après.

```

def previsions_Lasso (histoMod, Calendrier, dateDebMod, dateFinMod, hPrev, ic=0.95) :
    """
    Fonction qui réalise les prédictions selon le modèle de régression Lasso
    sur des indicatrices temporelles créées à partir du Calendrier,
    à partir de l'historique donné (déjà filtré pour un faisceau et un type de mouvement)

    Parameters
    -----
    histoMod : DataFrame
        Historique du trafic journalier, sur un seul faisceau et un seul type de mouvement,
        contenant : 'ArrDep', 'Faisceau', 'Date', 'PAX'
    Calendrier : DataFrame
        Calendrier à utiliser pour créer les features de la régression
    dateDebMod : datetime64[ns]
        Date de début de l'historique
    dateFinMod : datetime64[ns]
        Date de fin de l'historique
    hPrev : int
        Nombre de jours pour lesquels faire une prédiction du trafic

    Returns
    -----
    PrevisionsLasso : DataFrame
        Prévisions journalières du modèle (contient 'PAX_NP', 'Date', 'Faisceau', 'ArrDep')
    """

    # Sélection des variables d'intérêt du Calendrier :
    colonnesCalendrier = ["Mois", "JourSem", "Semaine", "LunF", "MarF",
                           "Mer1F", "Mer2F", "JeuF", "VenF", "SamF", "DimF",
                           "Vac_Toussaint", "Vac_Noel",
                           "Vac_Hiver_A", "Vac_Hiver_B", "Vac_Hiver_C",
                           "Vac_Printemps_A", "Vac_Printemps_B", "Vac_Printemps_C", "Vac_Ete"]
    Calendrier = Calendrier[["Date"]+colonnesCalendrier]
    Calendrier = Calendrier.fillna(0)

    # Idée : on transforme les colonnes de vacances en indicatrices qui valent 1 si on est
    # en première semaine de vacances (variable <= med(variable))
    # for k in range(1,10) :
    #     mediane_k = Calendrier[colonnesCalendrier[-k]].median()
    #     Calendrier[colonnesCalendrier[-k]] = Calendrier[colonnesCalendrier[-k]] <=
    #     mediane_k
    #     sur le test Schengen/Arrivées/hPrev=7j, ca diminue le score du modèle...

    # Augmentation histo & Récupération données utilisées pour l'entraînement :
    histoMod = pd.concat([histoMod.set_index('Date'), Calendrier.set_index('Date')], axis=1,
                         join="inner")
    histoMod = histoMod.reset_index()
    y_mod = histoMod['PAX']
    X_mod = histoMod[colonnesCalendrier]

```

```

# Récupération données utilisées pour les prédictions :
X_pred = Calendrier[(Calendrier['Date']>dateFinMod)&(Calendrier['Date']<=dateFinMod+
timedelta(days=hPrev))]
X_pred = X_pred[colonnesCalendrier]

# Création des indicatrices pour chaque colonne du calendrier ayant plus de 2 modalités
# (JourSem, Mois, ...):
encodeur = OneHotEncoder(drop='first').fit(X_mod)
X_mod = encodeur.transform(X_mod).toarray()
X_pred = encodeur.transform(X_pred).toarray()
    # la commande drop='first' permet d'enlever l'une des indicatrices à chaque fois
    # (la première arbitrairement)

# On peut désormais entraîner la régression Lasso et calculer les prédictions :
modele = LassoCV(cv=5,eps=0.001, n_alphas=100, normalize=True)
    # L'argument normalize=True va normaliser toutes les colonnes de X
modele.fit(X_mod,y_mod)

predictions = modele.predict(X_pred)
intervals = prediction_interval(modele, X_mod, y_mod, X_pred, 1-ic)

# Mise en forme des prédictions :
faisceau = histoMod['Faisceau'][0]
mvt = histoMod['ArrDep'][0]

PrevisionsLasso=[]
for k in range(hPrev) :
    PrevisionsLasso.append([dateFinMod + timedelta(days=k+1), faisceau, mvt,
predictions[k]])

PrevisionsLasso = pd.DataFrame(data=PrevisionsLasso,columns = ["Date", "Faisceau", "ArrDep", "PAX_Lasso"])
PrevisionsLasso["IC"+str(int(ic*100))+"_low_LASSO"] = PrevisionsLasso["PAX_Lasso"]+
intervals[0]
PrevisionsLasso["IC"+str(int(ic*100))+"_up_LASSO"] = PrevisionsLasso["PAX_Lasso"]+
intervals[2]

# print((faisceau,mvt,modele.alpha_))
# print("Score entrainement : "+str(modele.score(X_mod,y_mod)))

return PrevisionsLasso

```

La fonction `prediction_interval` implémentée ci-après calcule les bornes de l'intervalle de prédiction selon la méthode décrite dans l'approche empirique.

```

def prediction_interval(model, X_train, y_train, x0, alpha):
    ''' Fonction qui réalise un intervalle de prédiction de niveau alpha pour un modèle entré
    sur un ensemble d'entraînement et un ensemble de prédiction.

    Parameters
    -----
    model :
        Un modèle de prédiction avec les méthodes 'fit' et 'predict' (par ex. LassoCV(),
        LinearRegression() ou RandomForest())
    X_train : numpy array (n_samples, n_features)
        Array numpy contenant l'ensemble d'entraînement
    y_train : numpy array (n_samples,)
        Array numpy du réalisé correspondant au set d'entraînement
    x0 : numpy array (n_features,)
        Array numpy contenant l'ensemble à prédire
    alpha : float
        Niveau de l'intervalle de prédiction

```

```

    Returns
    -----
    percentiles[0] : numpy array
        borne inférieure de l'intervalle de prédiction de niveau alpha
        model.predict(x0) : numpy array
    Prédition
        percentiles[1] : numpy array
        borne inférieure de l'intervalle de prédiction de niveau alpha
    ,

# Echantillons d'entraînement
n = X_train.shape[0]

# On fait 1000 bootstraps
nbootstraps=1000

# Estimation des résidus validation et entraînement par bootstraps
bootstrap_preds, val_residuals = pd.DataFrame(), []
for b in range(nbootstraps):
    train_idxs = np.random.choice(range(n), size = n, replace = True)
    val_idxs = np.array([idx for idx in range(n) if idx not in train_idxs])
    model.fit(X_train[train_idxs, :], y_train[train_idxs])
    preds = model.predict(X_train[val_idxs])
    val_residuals.append(y_train[val_idxs] - preds)

# Estimation par la prédiction centrée du bruit
    bootstrap_preds[b] = model.predict(x0)
    bootstrap_preds -= np.mean(bootstrap_preds)
val_residuals = np.concatenate(val_residuals)

# Prédiction sur le modèle et résidus d'entraînement
model.fit(X_train, y_train)
preds = model.predict(X_train)
train_residuals = y_train - preds

# Correction de l'overfitting : arbitrage résidus de validation et d'entraînement
val_residuals = np.percentile(val_residuals, q = np.arange(100))
train_residuals = np.percentile(train_residuals, q = np.arange(100))

# Estimation des résidus corrigés : choix du cadre .632+ bootstrap
no_information_error = np.mean(np.abs(np.random.permutation(y_train) - \
    np.random.permutation(preds)))

generalisation = np.abs(val_residuals.mean() - train_residuals.mean())
no_information_val = np.abs(no_information_error - train_residuals)
relative_overfitting_rate = np.mean(generalisation / no_information_val)
weight = .632 / (1 - .368 * relative_overfitting_rate)
residuals = (1 - weight) * train_residuals + weight * val_residuals

# Construction de l'intervalle de prédiction et percentiles d'ordre alpha

C = np.array([m + o for m in bootstrap_preds for o in residuals])
qs = [100 * alpha / 2, 100 * (1 - alpha / 2)]
percentiles = np.percentile(C, q = qs)

return percentiles[0], model.predict(x0), percentiles[1]

```