UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Master 2 Traitement de l'information et data-science en entreprise (TIDE)

Cours : Grande Dimension

Analyse prédictive des annulations de réservations hôtelières

GROUPE 5

BOUDCHICH Soumia

DADDIO Melanie

TCHABO MBIENKEU Yves Gaël

Année universitaire 2024/2025

Introduction

Le secteur hôtelier fait face à des annulations de réservations régulières, facilitées par les politiques souples d'annulation et les plateformes en ligne. Ce phénomène occasionne des dépenses considérables dues aux pertes et perturbe à la fois la gestion opérationnelle et l'usage optimal des ressources.

Face à ce défi, l'utilisation de méthodes d'apprentissage statistique s'est révélée comme un moyen efficace de prévoir les annulations et d'en atténuer les conséquences. Dans le cadre d'apprentissage statistique nous avons s'inspirer de nombreuses recherches récentes qui soulignent l'importance de ces méthodes : Antonio et al. (2017)[1] dans leur *'Predicting hotel booking cancellations to decrease uncertainty and increase revenue'* ont démontré l'efficacité des modèles d'arbres de décision pour réduire les coûts opérationnels grâce à une prédiction précise des annulations. Les auteurs de l'article *'Forecasting cancellation rates for services booking revenue management using data mining'* Morales et Wang (2018)[2] ont introduit une autre stratégie pour mettre en évidence les avantages économiques de l'utilisation de modèles prédictifs intégrant des variables temporelles et comportementales afin d'améliorer l'attribution des chambres. Enfin, Santos et al. (2020)[3] dans leur article *'Leveraging predictive analytics to manage hotel cancellations and no-shows'* montrent que l'utilisation d'algorithmes de classification comme le XGBoost aide les hôtels à prévoir plus efficacement les annulations.

Dans cette optique, le groupe INN Hotels, envisage d'exploiter l'apprentissage statistique pour prévoir les annulations de réservations. Par conséquent, notre projet vise principalement à élaborer un modèle prédictif qui permettra de déterminer l'état des réservations. Ce modèle sera chargé de traiter deux types d'erreurs : anticiper à tort la continuité d'une réservation annulée, ce qui entraînerait des dégâts financiers, et estimer à tort une annulation, ce qui affecterait la réputation de l'hôtel et la satisfaction clientèle. Afin de trouver un équilibre optimal entre ces erreurs, le modèle sera optimisé à l'aide du *F1-score*, combinant la précision et le rappel.

Pour atteindre cet objectif, notre démarche comprendra trois étapes essentielles: Nous débuterons par une analyse exploratoire pour comprendre les données et adapter nos données pour l'étape modélisation. Ensuite, nous passerons à la modélisation pour anticiper les annulations de réservations, à travers différents modèles de classification tels que l'Arbre de Décision, la Régression Logistique et le XGBoost. En nous basant sur les résultats des modèles, nous proposerons diverses recommandations pour optimiser la gestion des réservations et réduire le taux d'annulation au sein du Groupe INN Hotels.[1]

---

[1] [1] Antonio, N., de Almeida, A., & Nunes, L. (2017). Predicting hotel booking cancellations to decrease uncertainty and increase revenue. *Tourism & Management Studies* : Redalyc
[2] Morales, D. R., & Wang, J. (2010). Forecasting cancellation rates for services booking revenue management using data mining. *European Journal of Operational Research* : ScienceDirect+3SpringerLink+3IDEAS/RePEc+3
[3]Santos, F., et al. (2020). Leveraging predictive analytics to manage hotel cancellations and no-shows. *Journal of Revenue and Pricing Management* : ScienceDirect

I. Identifier les principaux déterminants des annulations de réservation **(Melanie & Soumia)**

1. Analyse exploratoire des données

1.1 Présentation de la base de l'étude et des variables

La base de données analysée comprend 36,275 observations et 18 variables détaillées ci-dessous :

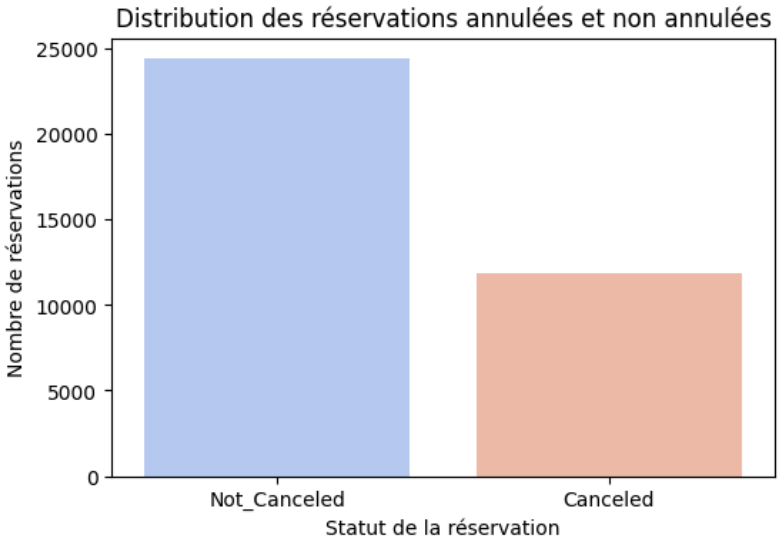
- **Booking_ID :** Identifiant unique de chaque réservation.
- **no_of_adults** : Nombre d'adultes.
- **no_of_children** : Nombre d'enfants.
- **no_of_weekend_night**s : Nombre de nuits réservées pendant le week-end.
- **no_of_week_night**s : Nombre de nuits réservées en semaine.
- **type_of_meal_plan** : Type de pension choisi par le client :
  - Not Selected : Aucun plan de repas sélectionné.
  - Meal Plan 1 : Petit-déjeuner.
  - Meal Plan 2 : Demi-pension (petit-déjeuner + un autre repas).
  - Meal Plan 3 : Pension complète (petit-déjeuner, déjeuner, dîner).
- **required_car_parking_space** : Besoin d'un parking (0 : Non, 1 : Oui).
- **room_type_reserved** : Type de chambre réservée.
- **lead_time** : Nombre de jours entre la réservation et l'arrivée.
- **arrival_year** : Année d'arrivée.
- **arrival_month** : Mois d'arrivée.
- **arrival_date** : Jour d'arrivée dans le mois.
- **market_segment_type** : Segment de marché de la réservation.
- **repeated_guest** : Client habitué ? (0 : Non, 1 : Oui).
- **no_of_previous_cancellations** : Nombre de réservations précédentes annulées par le client.
- **no_of_previous_bookings_not_canceled** : Nombre de réservations précédentes maintenues par le client.
- **avg_price_per_room** : Prix moyen par jour pour la réservation.
- **no_of_special_requests** : Nombre total de demandes spéciales formulées par le client.
- **booking_status** : Statut indiquant si la réservation a été annulée ou non.

Après une analyse préliminaire des données, les variables catégorielles required_car_parking_space et repeated_guest ont été converties au type catégoriel. De même, pour les variables arrival_year, arrival_month et arrival_date afin de mieux capturer les effets saisonniers et non linéaires dans les données.

Enfin, aucune valeur manquante n'a été détectée dans l'ensemble des données analysées.

1.2 Présentation de la variable cible et des variables explicatives

La variable cible booking_status se répartit comme suit :



- Réservations maintenues : 24,390 (67,3 %)
- Réservations annulées : 11,885 (32,7 %)

Cette distribution montre un taux élevé d'annulations, ce qui constitue un enjeu significatif pour la rentabilité de l'hôtel.

Concernant les variables explicatives, les statistiques descriptives mettent en évidence les éléments suivants :

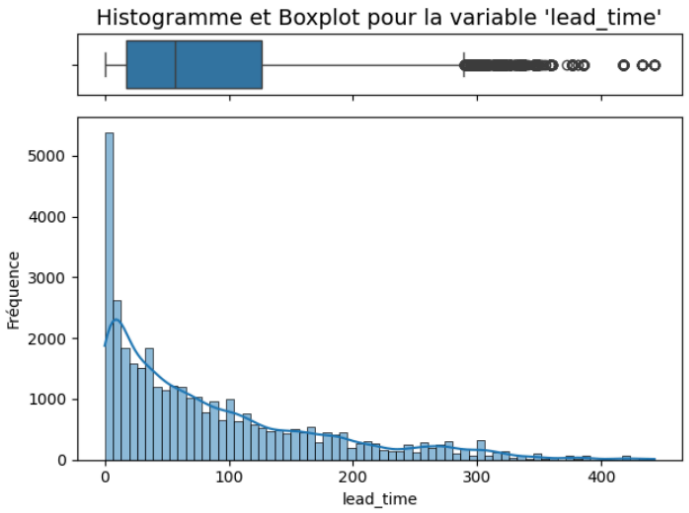
- Variables numériques :
- Délai moyen avant arrivée (**lead_time**) : 85,23 jours.
- Moyenne des nuits réservées en semaine : 2,20.
- Prix moyen par chambre : 103,42 euros.
- Variables catégorielles :
- Type de repas : Dominance du plan “Meal Plan 1” avec 27,835 occurrences.
- Besoin d'une place de parking : Majoritairement “Non” avec 35,151 occurrences.
- Type de chambre réservé : Majoritairement “Room_Type 1” avec 28,130 occurrences.
- Segment de marché : Principalement “ Online” avec 23,214 occurrences.
- Client régulier : Majoritairement “Non” avec 35,345 occurrences.

2. Statistiques descriptives

2.1 Analyse univarié

Nous avons d'abord réalisé une analyse univariée des variables quantitatives via des histogrammes et des boxplots, révélant plusieurs valeurs aberrantes : **lead_time** avec des valeurs élevées atypiques, **avg_price_per_room** avec des tarifs extrêmes ou proches de zéro, et **no_of_children** avec quelques valeurs inhabituelles. Ces anomalies feront l'objet d'un traitement spécifique. Ensuite, l'analyse des variables qualitatives a montré que certaines modalités peuvent être regroupées.

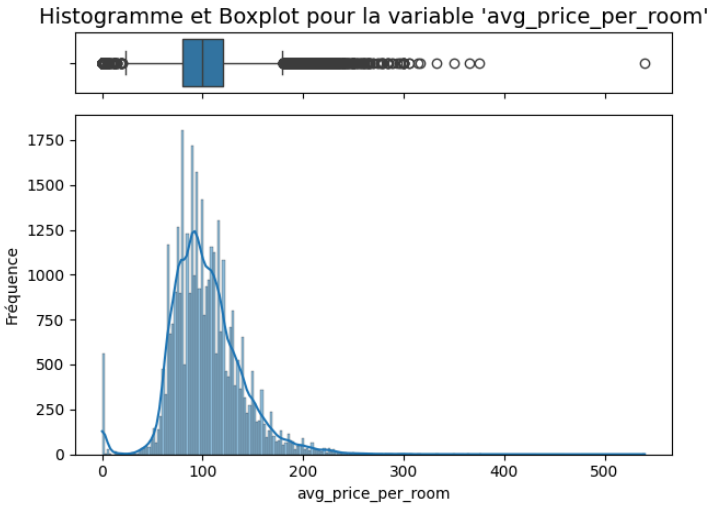
2.1.1 La Variable lead_time



La variable lead_time correspond au délai en jours entre la réservation et l'arrivée, et présente une distribution fortement asymétrique vers la droite. La majorité des réservations concerne des délais courts, mais certaines observations avec des délais très longs (plus de 300 jours) sont présentes.

Sachant qu'un modèle logistique est sensible à la forme (l'asymétrie) des données et compte tenu de ces observations, une transformation logarithmique serait plus appropriée qu'un simple traitement des valeurs aberrantes afin de limiter l'influence des valeurs extrêmes.

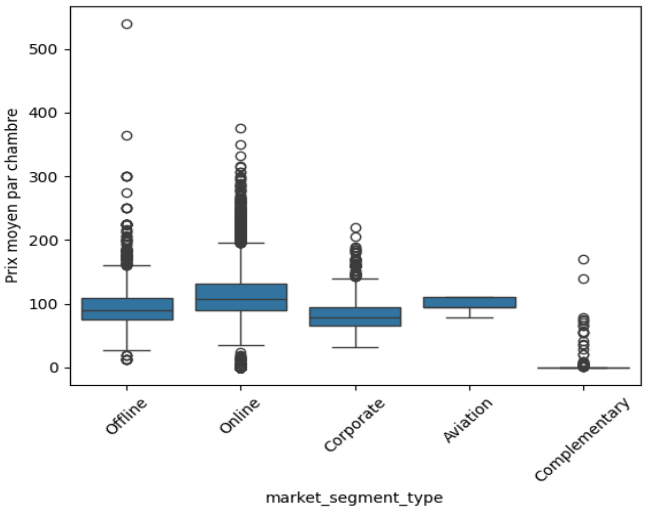
2.1.2 La Variable avg_price_per_room



La distribution de la variable avg_price_per_room est fortement asymétrique vers la droite, avec une majorité de prix entre 50 et 200 euros, accompagnée d'une faible proportion de prix plus élevés (200 à 400 euros). On note aussi quelques valeurs aberrantes, notamment une réservation à 500 euros ainsi que des réservations à 0 euro. Pour analyser plus précisément ces anomalies, nous étudierons cette variable selon deux critères : le segment de marché et le type de chambre réservée.

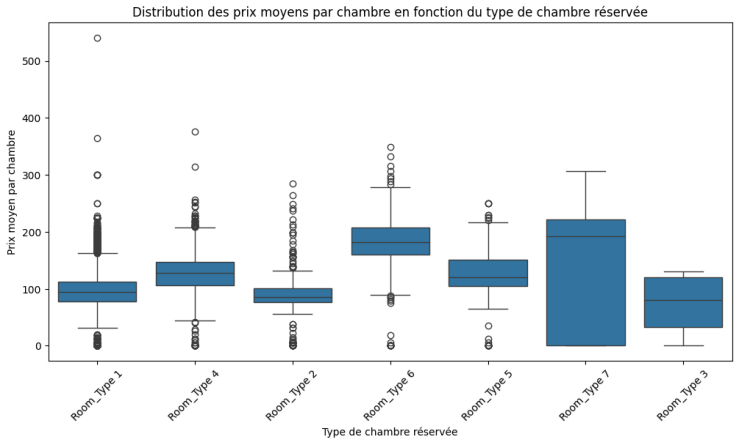
- Le segment de marché :

Les réservations Online affichent un prix moyen par chambre légèrement supérieur et une dispersion plus élevée, probablement due aux frais associés aux plateformes numériques. Le segment Complementary présente logiquement des prix très faibles ou proches de zéro, indiquant des chambres offertes ou à tarifs réduits. Enfin, le segment Offline montre une dispersion notable avec des valeurs extrêmes, notamment une réservation exceptionnelle à 540 euros



- Le type de chambre réservée :

Chaque type de chambre présente une distribution des prix distincte. En particulier, les types Room_Type 4 et Room_Type 1 affichent une forte dispersion, incluant des valeurs aberrantes comme la réservation à 540 euros que nous examinerons plus en détail par la suite.



Pour la variable avg_price_per_room, nous avons décidé de conserver les observations à prix nul (0€). Ces réservations correspondent à des séjours offerts (Complementary) ou à des promotions spécifiques dans le segment Online, ce qui est cohérent avec la stratégie commerciale de l'hôtel. En revanche, l'observation exceptionnelle (Booking_ID : INN33115) avec un prix très élevé (540€) a été traitée en tant que valeur aberrante à l'aide de la méthode de l'intervalle interquartile (IQR), en la remplaçant par la valeur maximale calculée (179.55€).

2.1.3 La variable no_of_children

Une analyse approfondie révèle que 93 % des clients voyagent sans enfants, et ceux accompagnés voyagent généralement avec 1 à 3 enfants. Toutefois, quelques rares observations indiquant un nombre élevé d'enfants (9 ou 10) semblent irréalistes dans ce contexte. Ces valeurs aberrantes seront donc remplacées par la valeur maximale réaliste, fixée à 3 enfants.

2.1.4 La variable type_of_meal_plan

On constate que les prestations avec petits déjeuners sont majoritairement choisies par les clients. Tant dis que les formules telles que demie pensions ou pension complète le sont beaucoup moins. Ainsi, nous avons eu l'intuition de regrouper "Meal Plan 2" et "Meal Plan 3" en une seule catégorie ("Other"). Nous verrons par la suite que ce regroupement n'est pas forcément utile au vu de la stratégie d'encodage sélectionnée.

2.1.5 La variable room_type_reserved

La catégorie "Room_Type 1" domine largement les réservations, tant dis que les types "Room_Type 3" et "Room_Type 7" sont extrêmement rares. Cependant, dans ce cas nous n'avons pas envisagé de faire un regroupement dans une catégorie "Autres" du fait que les prix changent en termes de type.

2.1.6 La variable market_segment_type

Pour finir, suite à notre analyse on a pu noter que les réservations sont généralement faites en ligne ou hors ligne. Certaines modalités, telles que Corporate, Complementary, Aviation, beaucoup plus rares sont tout de même présentes.

2.2 Analyse bivarié

Pour approfondir l'analyse descriptive, nous avons examiné la répartition de chaque variable en fonction de la variable cible. Cette approche graphique permet d'identifier les variables influençant potentiellement les annulations.

Pour les variables continues, les réservations annulées sont généralement associées à un délai plus long entre la réservation et l'arrivée, tandis que les réservations proches de la date d'arrivée sont plus souvent maintenues. De plus, les réservations à des prix élevés (100-150 €) ont tendance à être plus annulées que celles à prix modéré.

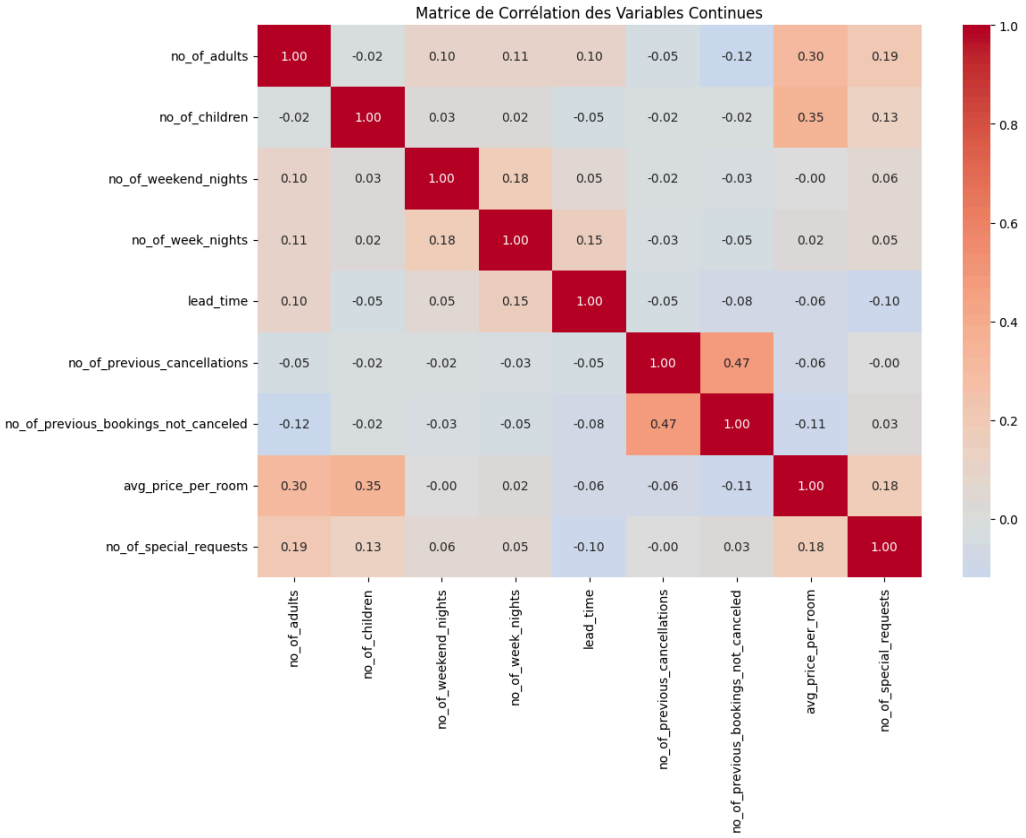
Pour les variables catégorielles, les annulations sont plus fréquentes en fin d'année (août-octobre) qu'en début d'année (décembre-mars). Les clients réguliers, bien que peu nombreux, semblent moins enclins à annuler leur réservation par rapport aux nouveaux clients.

2.3 Analyse de la corrélation et liaison avec la variable cible

Afin d'améliorer la précision du modèle et de réduire la redondance dans les données, nous avons analysé la dépendance entre les variables explicatives.

2.3.1 Corrélation entre les variables numériques

Pour les variables quantitatives nous avons regardé la matrice de corrélation :



Dans l'ensemble, les variables quantitatives présentent des corrélations faibles, proches de zéro. Certaines, comme avg_price_per_room avec no_of_adults et no_of_children, montrent une légère corrélation. En revanche, une corrélation plus élevée de 0,47 a été observée entre no_of_previous_cancellations et no_of_previous_bookings_not_canceled. Avant de

supprimer l'une de ces deux variables, nous évaluerons leur importance lors de la sélection des variables.

2.3.2 Dépendance entre les variables catégorielles

Pour les variables qualitatives, nous avons effectué un test du Khi-deux pour évaluer leur indépendance et calculé le V de Cramer pour mesurer la force de l'association. Une liaison est considérée significative si le V de Cramer dépasse 0,3. L'analyse a révélé une corrélation entre market_segment_type et repeated_guest, ainsi qu'entre arrival_year et arrival_month. Afin d'éviter la multicolinéarité et d'améliorer la stabilité du modèle, nous avons décidé de ne conserver qu'une seule variable dans chaque paire, en fonction de sa relation avec la variable cible.

2.3.3 Liaison des variables avec la variable cible

En complément de l'analyse de corrélation, nous avons utilisé le test de Welch pour évaluer la relation entre la variable cible et les variables continues, révélant une liaison significative pour toutes les variables (p-value < 0,05). Pour les variables qualitatives, le test du Khi-deux confirme leur pertinence. Les variables ayant le plus d'influence sont arrival_month, arrival_year et market_segment_type. Pour éviter la redondance, nous avons conservé arrival_month et market_segment_type.

II. Approche de modélisation et Ajustement

1. Préparation des données pour la modélisation **(Melanie & Soumia)**

Après cette analyse descriptive, nous passons maintenant à la mise en place du modèle prédictif. Cette étape inclut plusieurs traitements préparatoires : encodage de la variable cible, transformation logarithmique des variables asymétriques, suppression des variables redondantes (V de Cramer), encodage des variables catégorielles, normalisation avec la fonction RobustScaler et séparation des données en ensembles train et test tout en maintenant la répartition des classes grâce à la méthode stratify.

Pour ce faire, nous allons commencer par appliquer des traitements basés sur les résultats de l'analyse descriptive : la variable cible booking_status a été encodée en binaire (Canceled = 1, Not_Canceled = 0). Ensuite pour corriger l'asymétrie des données, une transformation logarithmique a été appliquée aux variables lead_time et avg_price_per_room. Enfin, pour éviter la redondance, les variables repeated_guest et arrival_year ont été supprimées sur la base de la corrélation mesurée par le V de Cramér.

Ensuite, nous avons adopté une stratégie d'encodage adaptée à la nature des variables catégorielles :

- Pour les variables à plus de 4 modalités (*type_of_meal_plan*, *room_type_reserved*, *market_segment_type*), chaque modalité des variables dans les données d'entraînement a été remplacée par la moyenne de la variable cible associée, permettant de réduire la dimensionnalité par rapport à un one-hot encoding tout en conservant l'information liée à la variable cible.
- La variable binaire (*required_car_parking_space*) a été directement encodée en binaire (0 ou 1).
- Pour les variables temporelles (*arrival_date* et *arrival_month*), nous avons utilisé un encodage cyclique pour prendre en compte leur nature périodique. Un one-hot encoding aurait artificiellement éloigné des valeurs proches (exemple : 01 et 12). L'encodage cyclique a donc été réalisé en transformant ces valeurs en coordonnées sur un cercle à l'aide des fonctions sinus et cosinus.

Pour préparer les variables continues à la modélisation, nous avons choisi de les normaliser à l'aide de RobustScaler() afin de les ramener à la même échelle. Ce choix s'explique par plusieurs raisons : Premièrement, la régression logistique s'adapte mieux avec les lois normales et deuxièmement, ce transformateur standardise les données en utilisant les quantiles plutôt que la moyenne et l'écart-type, ce qui le rend plus résistant aux distributions asymétriques et aux valeurs aberrantes.

Notre objectif est d'identifier les principaux facteurs influençant les annulations de réservation. Pour cela, nous utiliserons notre meilleur modèle afin de sélectionner les

variables les plus pertinentes sur la base de leurs contributions dans la prédiction de l'annulation de réservation.

2. Elaboration et comparaison des modèles de classification

Notre étude vise à anticiper si un client va annuler sa réservation d'hôtel. Pour cela, nous avons opté pour l'élaboration d'un modèle de classification. Par conséquent, nous allons évaluer différentes méthodes de classification supervisée.

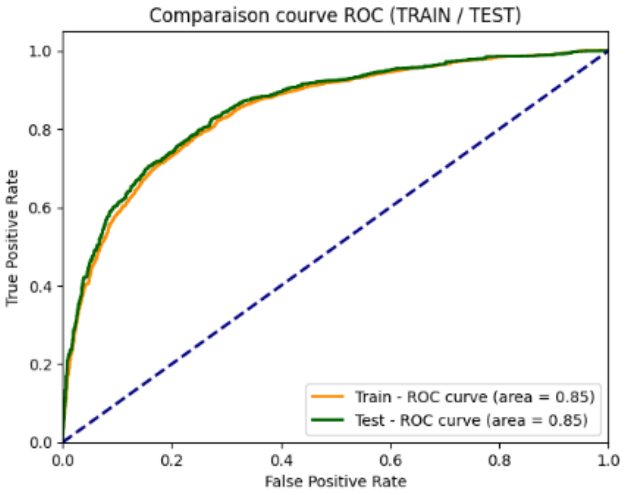
Nous commencerons par mettre en œuvre le modèle traditionnel de classification : la régression logistique, pour obtenir une première estimation des performances. Nous approfondirons notre recherche en nous focalisant sur un modèle d'arbre de décision, une approche plutôt simple qui présente un intérêt notable pour nos recommandations. Pour finir, nous allons mettre à l'essai un modèle XGBoost dans le but d'estimer son potentiel d'amélioration de nos résultats.

La performance de ces modèles sera évaluée à l'aide de plusieurs métriques afin de sélectionner celui offrant le meilleur compromis entre précision et rappel, et en privilégiant le F1-Score.

2.1 Modèle de Régression Logistique (Scikit-Learn) **(Soumia)**

Nous avons commencé avec la mise en place d'un modèle de régression logistique, spécifiquement ajusté à ce problème binaire du fait de sa capacité à délivrer une interprétation directe des coefficients et à représenter des relations non linéaires simples entre les variables.

Nous avons commencé par établir un modèle initial qui dans un premier temps a montré une bonne performance avec une AUC de 0,85 sur l'entraînement et le test. Nous avons également effectué une analyse des coefficients de ce modèle qui nous a montré que le lead_time est le facteur le plus influent avec odds ratio de 4,35, ce qui indique qu'un délai de réservation plus long augmente la probabilité d'annulation. La réservation d'une place de parking réduit significativement ce risque avec un odds ratio de 0,25, ce résultat peut se traduire par un engagement plus fort du client. Le prix moyen par chambre est également positivement corrélé au risque d'annulation avec un odds ratio de 2,27.



Comme notre métrique de performance est le F1-score nous avons généré une matrice de confusion qui révèle une précision globale de 81% et un F1-Score de 0,86 pour la classe "non annulé" mais de seulement 0,67 pour la classe "Annulé". Ce résultat montre que le modèle a plus de difficultés à prédire cette classe.

Pour améliorer le F1-score, nous avons d'abord testé le Lasso pour une sélection de variables. En effet, Lasso va pénaliser certains coefficients (des variables les moins explicatives) à 0 avec la pénalité L1, ce qui va nous permettre de sélectionner un ensemble de variables explicatives. De plus, cette méthode permet de gérer la multicolinéarité. Malgré que nous avons précédemment tenté de gérer la redondance des variables, cette caractéristique de Lasso peut s'avérer intéressante.

Tout d'abord, nous avons généré un chemin Lasso qui identifiait 12 variables, soit le modèle complet. Ainsi, malgré cette sélection, le score F1 pour la catégorie Annulé stagne à 0,67. Puis en prenant plus de recul sur notre choix nous nous sommes aperçus que la sélection de variables par lasso n'était pas forcément la méthode la plus pertinente dans notre cas. En effet, le Lasso repose sur une relation linéaire entre les variables explicatives et la variable cible. Or, dans notre cas on s'aperçoit que certaines variables, telles que qualitatives, ont un impact non linéaire sur le fait d'annuler sa réservation d'hôtel ou non.

Sur la base de ce constat, nous avons décidé d'utiliser un modèle de régression logistique avec statsmodels pour la sélection des variables. Nous avons choisi le modèle de régression logistique pour la sélection de variables à l'aide de la librairie statsmodels plutôt que Scikit-Learn car cette dernière ne donne pas d'information sur la pertinence des coefficients. Ainsi, pour aller plus loin et détecter la pertinence des coefficients, nous allons utiliser le package statsmodels. Cependant même avec cette nouvelle sélection de variables notre modèle n'est pas amélioré.

Enfin, nous avons optimisé le modèle avec une Grid Search pour ajuster les hyperparamètres. Cela a permis d'atteindre une précision de 80,5% mais le F1-Score de la classe "Annulé" est resté inchangé à 0,67.

Pour corriger ce problème, nous avons introduit une pondération automatique des classes pour réajuster le poids de la classe minoritaire. Cette approche a entraîné une amélioration notable : le recall de la classe "Annulé" est passé de 60% à 77%, et le F1-Score est monté à 0,69.

	precision	recall	f1-score	support
0	0.87	0.78	0.82	4878
1	0.63	0.77	0.69	2377
accuracy			0.78	7255
macro avg	0.75	0.77	0.76	7255
weighted avg	0.79	0.78	0.78	7255

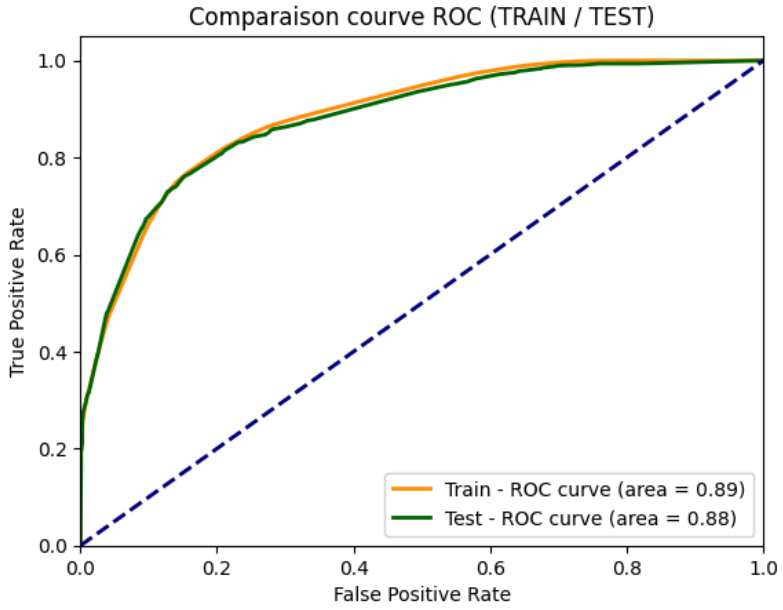
Cette correction a aidé le modèle à ajuster légèrement l'équilibre entre la détection des deux classes. Néanmoins les performances de ce modèle restent discutables. Le fait de sélectionner des variables ou non n'est pas forcément pertinent pour la logistique. En effet, selon nous, la logistique ne fonctionne pas bien car elle a du mal à capter les relations non linéaires entre les variables et la variable cible. Suite à ces constatations, nous considérons l'idée d'optimiser la performance en essayant un modèle non linéaire, tel qu'un arbre de décision.

2.2 Modèle d'arbre de décision optimisé par Grid Search **(Melanie)**

L'objectif de notre étude n'est pas seulement de prédire si un client va annuler ou non. Nous devons aussi proposer des recommandations afin d'optimiser la gestion des réservations et de réduire le taux d'annulation. Ainsi, le modèle d'arbre de décision nous a paru pertinent.
Dans un premier temps, ce modèle nous a paru pertinent pour ce type de problème où les relations entre les variables ne sont pas toujours linéaires.
Dans un second temps ce modèle s'avère intéressant dans notre cas car il est capable d'identifier les variables les plus pertinentes et les classer par importance. Ainsi, en déterminant quelles variables ont un impact significatif sur la probabilité d'annulation, nous pourrions recommander à l'hôtel de se concentrer sur ces facteurs.

Nous avons donc commencé par établir un "modèle d'initialisation". Afin de rendre le modèle plus performant nous avons tenté d'optimiser ses paramètres. Pour ce faire nous

avons testé trois approches : la méthode aléatoire, la méthode Grid Search et la méthode Optimisation Bayesienne. Il s'est avéré que la méthode par Grid Search a obtenu le meilleur score pour la meilleure combinaison d'hyper paramètres trouvée.
En appliquant cette combinaison de d'hyper paramètres, nous avons pu obtenir de meilleurs résultats de F1-score que ce soit au niveau des "annulations" (0.72) ou des 'non-annulations" (0.87).



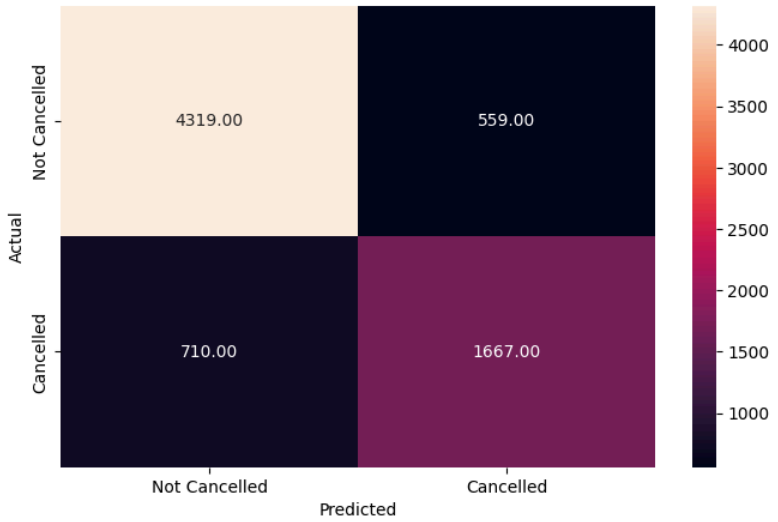
Notre modèle semble bien ajusté avec des AUC (Area Under Curve) sur l'ensemble d'apprentissage et de test assez proches et plutôt proches de 1.

De plus, en appliquant notre sélection d'hyper paramètres nous avons nettement amélioré les résultats de la logarithmic loss. En effet, la logarithmic loss de l'ensemble de test (0.47) et celle de l'ensemble de train (0.38) sont bien plus proches qu'avec notre modèle d'initialisation. Ainsi, en optimisant nos hyperparamètres nous avons pu limiter le sur apprentissage même si même si le modèle semble tout de même mieux prédire sur l'ensemble d'apprentissage que sur l'ensemble de test.

Concernant l'importance des variables, nous avons obtenu un classement similaire entre le modèle initial et le modèle optimisé. Néanmoins, la répartition de l'importance des variables est plus équilibrée : le modèle optimisé dépend moins d'une seule variable et donne plus d'importance à plus de variables. Ainsi les résultats nous indiquent que les variables telles que lead_time, no_of_special_requests et avg_price_per_room ont un impact important sur l'annulation de chambre d'hôtel.

Enfin, afin d'améliorer nos résultats nous avons cherché le seuil 0.357143 qui maximise le F1-score plutôt que d'utiliser le seuil par défaut.

	precision	recall	f1-score	support
0	0.86	0.89	0.87	4878
1	0.75	0.70	0.72	2377
accuracy			0.83	7255
macro avg	0.80	0.79	0.80	7255
weighted avg	0.82	0.83	0.82	7255



2.3 Modèle XGBoost optimisé (Yves)

Notre modélisation à l'aide de l'arbre de décision a montré une amélioration du F1-Score par rapport à la régression logistique. Cependant, pour aller plus loin et maximiser la performance du modèle, nous avons décidé d'implémenter le modèle XGBoost. Contrairement à un arbre de décision classique, XGBoost construit plusieurs arbres de manière séquentielle, en corrigeant progressivement les erreurs des arbres précédents. De plus, il intègre automatiquement la régularisation L1 et L2, ce qui permet de mieux contrôler le surajustement et d'améliorer la capacité de généralisation du modèle.

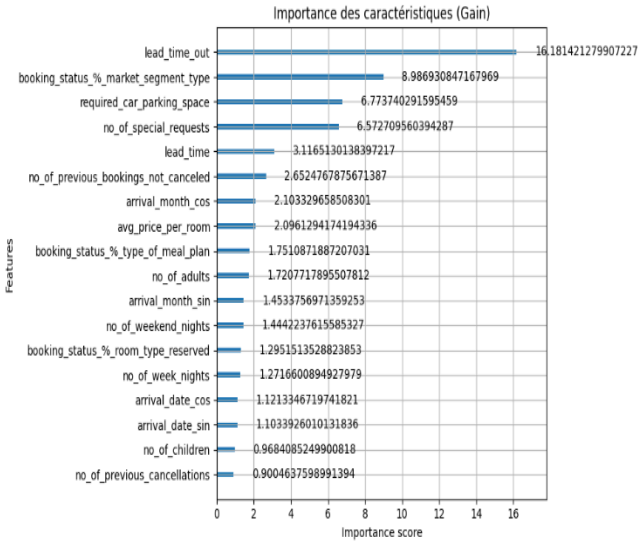
Nous avons commencé par établir un modèle d'initialisation avec la configuration : Classification multiclasse, Nombre de classes : 2 , Fonction de perte logarithmique (mlogloss), booster : gbtree. Nous avons commencé par implémenter un modèle d'initialisation qui a pu avoir une précision globale de 90% avec une macro-moyenne du F1-Score de 0,88%. A travers sa matrice de confusion on peut voir que le modèle détecte efficacement les réservations non annulées avec une précision de 91%, un rappel de 94% et un F1-Score pour la classe "Annulée" de 0,84%, indiquant que le modèle avait encore des difficultés à identifier correctement les annulations.

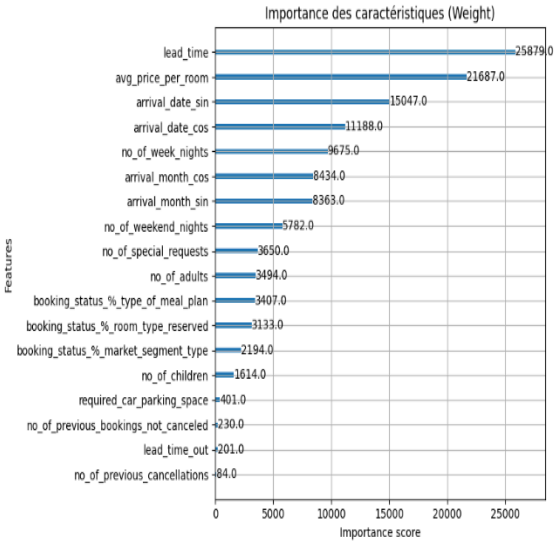
Pour améliorer cette performance, nous avons procédé à une optimisation des hyperparamètres du modèle en utilisant deux méthodes : RandomSearchCV et Optuna. Après avoir comparé les résultats, nous avons constaté que le modèle optimisé avec les meilleurs paramètres renvoyés par les deux méthodes fournit des performances à peu près similaires mais avec une légère préférence pour les paramètres de RandomSearchCV. Nous tenons à préciser que la recherche de meilleurs de paramètres à été nettement plus rapide avec RandomSearchCV. Les paramètres optimaux renvoyés par cette méthode ont conduit à une légère amélioration. La précision globale est restée à 90%, la moyenne du F1-Score a atteint 89%. La nouvelle matrice de confusion montre que le modèle optimisé détecte efficacement les réservations non annulées avec une précision de 91% et un rappel de 95%. Le F1-Score pour la classe "Anulée" est passé à 81% indiquant une légère amélioration.

	precision	recall	f1-score	support
0	0.91	0.95	0.93	4878
1	0.89	0.81	0.85	2377
accuracy			0.90	7255
macro avg	0.90	0.88	0.89	7255
weighted avg	0.90	0.90	0.90	7255

La matrice de confusion après optimisation montre une forte amélioration dans la détection des annulations. Le nombre de faux négatifs a diminué, traduisant une meilleure séparation entre les deux classes. L'augmentation du F1-Score pour la classe "Annulé" confirme une capacité du modèle à mieux prédire les annulations, tout en maintenant une bonne capacité de classification pour la classe "Non annulée".

La figure suivante montre que les variables les plus utilisées dans la construction du modèle (Weight) sont : lead time, avg_price_per, les variables de date (date and month), no_of_week_nights, no_of_weekend_nights et no_of_special_requests. Cependant, les variables qui contribuent le plus à l'efficacité du modèle (Gain) classés par ordre de contribution sont les variables lead_time_out, booking_status_%_market_segment_type, required_car_parking_space, no_of_special_request et no_of_previous_bookings_not_canceled.





III. Interprétation, recommandations et limitation

1. Interprétation des résultats: **(Melanie, Soumia et Yves)**

Cette étude repose sur deux objectifs principaux : proposer un modèle performant permettant à INN Hotels Group d'anticiper les clients susceptibles d'annuler leur réservation et émettre des recommandations afin d'optimiser la gestion des réservations et de réduire le taux d'annulation.

Concernant la prédiction, le modèle le plus performant est le modèle XGBoost optimisé à l'aide de RandomSearchCV, qui offre le meilleur F1-Score. Grâce à cette optimisation, le modèle atteint une précision globale de 90%, avec un F1-Score de 0,93 pour la classe "Non annulé" et de 0,85% pour la classe "Annulé".

Concernant les recommandations pour le groupe hôtelier, notre analyse descriptive détaillée a permis de mieux comprendre l'influence de certaines variables sur les annulations. Cette analyse, accompagnée par l'étude de l'importance des variables fournit des éléments concrets pour optimiser la gestion des réservations.

Lors de l'analyse descriptive, plusieurs constats ont émergé. Tout d'abord, les réservations effectuées très en avance, en particulier lorsque le prix de la chambre est supérieur à la moyenne, ont tendance à être plus souvent annulées. Ce phénomène est particulièrement marqué en fin d'année, notamment entre août et septembre.

Les résultats de la modélisation confirment ces tendances. L'analyse de l'importance des variables lors de la prédiction des annulations met en évidence quatre facteurs déterminants :

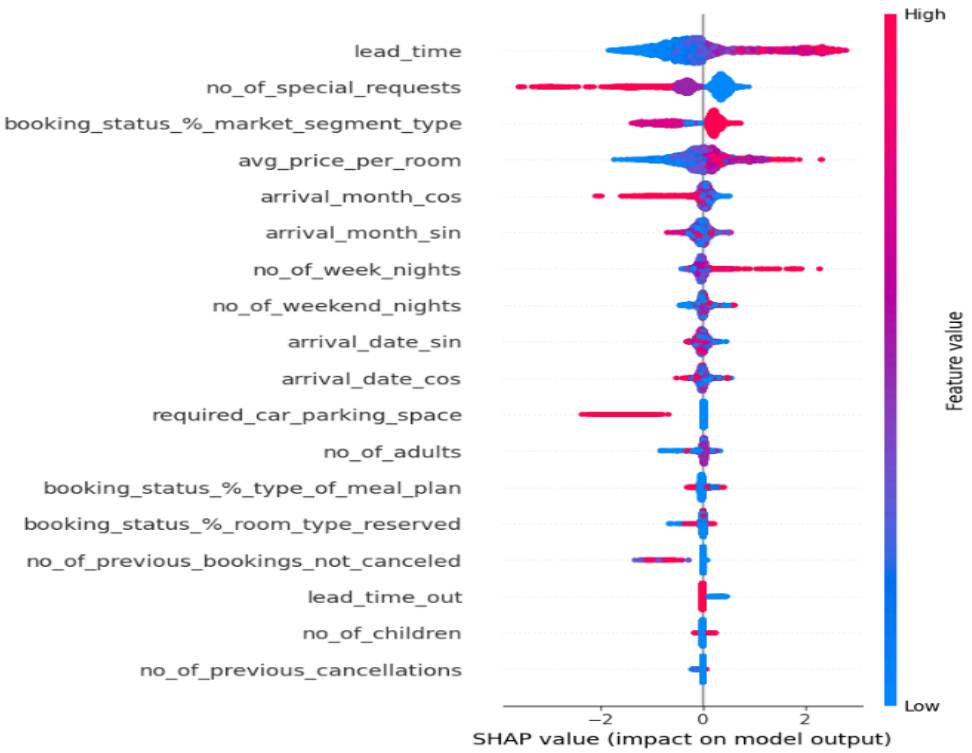
- le délai de réservation
- nombre de demande spéciale

- le type de segment du marché
- espace parking

L'approche basée sur l'arbre de décision optimisé permet d'aller plus loin en identifiant des combinaisons de facteurs influençant le risque d'annulation. Par exemple, lorsque le prix moyen par chambre est élevé (avg_price_per_room > 0.36) et que le délai de réservation est long (lead_time > 0.53), le taux d'annulation augmente fortement. En revanche, la présence de demandes spéciales (required_car_parking_space = 1) combinée à un délai de réservation court (lead_time ≤ 0.44) réduit significativement le risque d'annulation.

Pour approfondir notre compréhension de ces caractéristiques, nous avons complété notre analyse en utilisant un modèle XGBoost interprété à l'aide des Shapley Additive Explanations (SHAP). En agrégeant les valeurs SHAP sur l'ensemble des données, on identifie les caractéristiques ayant le plus d'influence sur le modèle. La figure ci-dessous nous permet de recueillir plusieurs informations. Premièrement les variables les plus importantes sont **lead_time**, **no_of_special_requests**, **booking_status_%_market_segment_type**, **avg_price_per_room**, **arrival_month_cos**, **no_of_week_nights**, **required_car_parking_space** et **no_of_previous_booking_not_canceled**. Secondement, l'influence de chaque variable sur la prédiction. Nous y avons relevé les informations suivantes :

- Lorsque pour un individu donné, les valeurs des variables **lead_time, avg_price_per_room, no_of_weekend_nights, no_of_adults** et **no_of_previous_cancellations** sont très basses, ou alors lorsque les valeurs des variables **no_of_special_requests, arrival_month_cos et required_car_parking_space** sont trés hautes, on peut alors considérer avec une forte probabilité que cet individu maintiendra sa réservation.
- Par contre, lorsque les valeurs des variables **lead_time, booking_status_%_market_segment_type, avg_price_per_room** ou encore **no_of_week_nights** sont très hautes, on peut cette fois supposer avec une forte chance que l'individu annulera sa réservation.
- On note aussi qu'une valeur moyenne de la variable **no_of__special_requests** contribue négativement à la probabilité d'être classé "Annulé" par le modèle.



Ces résultats montrent que certains facteurs clés sont déterminants dans la prédiction des annulations. Par conséquent, des actions ciblées pourraient être mises en place pour réduire le taux d'annulation.

2. Recommandations **(Melanie)**

A présent, en nous basant sur notre analyse nous allons émettre des propositions et idées afin d'optimiser la gestion des réservations et limiter les annulations.

Pour commencer, il serait intéressant que le groupe INNhotel mette en place un programme de fidélité. En effet, on a constaté que les personnes ayant déjà effectué une réservation auront moins tendance à annuler. Ainsi, en ciblant cette population et en essayant de l'agrandir nous pourrions atténuer le risque d'annulation. Pour ce faire, il serait intéressant de créer une carte spéciale pour les clients réguliers. Cette carte leur donnerait la possibilité de cumuler des points afin d'accéder divers avantages, comme des promotions ou des prestations spéciales (places de parking, surclassement, offre de petit déjeuners).

Ensuite, il serait pertinent de mettre en place une politique d'annulation en fonction du délai de réservation. En effet, le délai de réservation est le facteur influençant le plus le risque d'annulation. Par exemple, INN Hotel Group pourrait mettre en place une politique où les clients qui réservent très en avance ne peuvent annuler gratuitement, sauf en payant un supplément. Néanmoins, il est possible que INN Hotel Group ne souhaite pas pénaliser ses clients. Dans ce cas, nous pourrions adopter la méthode inverse : encourager les réservations de dernière minute. Pour ce faire, l'hôtel pourrait proposer des offres attractives de dernière

minute en cas de disponibilité. Cela permettrait au passage à l'hôtel d'optimiser l'occupation de ses chambres qui n'allaient pas l'être et limiter les pertes financières.

De plus, nous avons constaté que la période hivernale était moins sujette à d'éventuelles annulations. Il serait donc judicieux d'adapter la stratégie tarifaire selon la saison. Rendre les offres hivernales plus attractives, avec des promotions ou des services supplémentaires pour attirer davantage de réservations. À l'inverse, augmenter les prix pour les périodes à fortes demandes, comme l'été par exemple, pour compenser les éventuelles annulations.

Enfin, les clients ayant formulé des demandes spécifiques ont un taux d'annulation plus faible, cette stratégie permettrait de renforcer leur engagement. Ainsi il serait pertinent de pousser les clients à effectuer des demandes spéciales. Pour ce faire, l'hôtel pourrait mieux mettre en avant ces services, en les mettant davantage en avant sur le site pour les réservations en ligne et en demandant au personnel de les proposer systématiquement pour les réservations hors ligne. De plus, il serait judicieux de rendre ces options plus accessibles.

Si l'hôtel souhaite aller encore plus loin que ces recommandations émises, il pourrait envisager d'exploiter notre modèle prédictif XGBoost optimisé. À chaque nouvelle réservation, soumettre au modèle le profil du client ainsi que ses informations relatives à sa réservation afin d'analyser le risque d'annulation du client. Si un client est identifié comme à risque, lui proposer des avantages spécifiques, comme un petit-déjeuner offert ou un surclassement, afin de l'inciter à maintenir sa réservation.

3. Limitations et améliorations possibles **(Soumia)**

Bien que le modèle XGBoost optimisé ait montré des performances solides avec une précision globale de 90% et un F1-Score de 0,85 pour la classe "Annulé", certaines limites subsistent.

Tout d'abord, le déséquilibre des classes reste un défi, le modèle ayant une capacité de prédiction légèrement inférieure pour la classe "Annulé" en raison de la faible fréquence de cette classe dans l'ensemble d'entraînement.

La complexité du modèle est une autre limite. Bien que l'optimisation par RandomSearchCV ait amélioré les performances, le modèle reste coûteux en temps de calcul, surtout avec un volume de données élevé.

De plus, bien qu'on ait tenté de l'intégrer avec notre encodage cyclique, la nature temporelle des données n'a pas été pleinement exploitée. L'intégration de caractéristiques temporelles plus avancées telle que la tendance saisonnière via des modèles comme Prophet permettrait de mieux capter la dynamique des annulations.

Conclusion

L’objectif principal de notre étude était de développer un modèle permettant d’anticiper les annulations de réservations afin d’aider INN Hotels Group à mieux gérer ses coûts et optimiser son taux d’occupation. Nous pensons avoir atteint cet objectif dans une certaine mesure.

Notre analyse descriptive et notre modélisation nous ont permis d’identifier les principales variables influençant l’annulation d’une réservation avec notamment le délai de réservation, le prix moyen par chambre, le nombre de demandes spéciales, ainsi que la saisonnalité. Ces résultats sont importants non seulement pour améliorer la prédiction des annulations, mais aussi pour guider l’hôtel dans ses décisions. En effet, ces facteurs nous ont permis de formuler des recommandations concrètes à INN Hotels Group afin de réduire son taux d’annulation.

Par ailleurs, bien que notre modèle ne soit pas parfait, il pourrait tout de même aider l’hôtel. En effet, notre modèle pourrait être utilisé afin d’analyser chaque nouvelle réservation et estimer son risque d’annulation. Ainsi, l’hôtel pourrait anticiper les annulations potentielles, limiter ses pertes financières et améliorer la satisfaction des clients.

Néanmoins, nous sommes conscients que notre modèle peut tout de même être amélioré afin d'être plus performant. Certaines limites, comme le déséquilibre des classes et la complexité du modèle XGBoost limitent encore sa précision.

De ce fait, bien que optimisable, notre étude propose tout de même une approche intéressante pour INN Hotels Group, avec la proportions d’un outil prédictif ainsi que des recommandations pour réduire son taux d’annulation et améliorer sa rentabilité.

ANNEXE : Code

1 Librairies

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import scipy.stats as st
from scipy.stats import chi2_contingency
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, RobustScaler
from sklearn.model_selection import GridSearchCV
from xgboost import plot_importance
```

2 Aperçu Préliminaire des Données

[ ]:
```
data = pd.read_csv("INNHotelsGroup.csv")
data = data.set_index("Booking_ID")
data.head()
```

[ ]:
```
data.shape
```

[ ]:
```
data.info()
```

[ ]:
```
data[["required_car_parking_space", "repeated_guest"]] =␣
↪data[["required_car_parking_space", "repeated_guest"]].astype("category")
```

[ ]:
```
data.isnull().sum()
```

2.1 Aperçu de la variable “booking_status”

[ ]:
```
print(data["booking_status"].value_counts())

plt.figure(figsize=(6,4))
sns.countplot(x=data["booking_status"], palette="coolwarm")
plt.title("Distribution des réservations annulées et non annulées")
plt.xlabel("Statut de la réservation")
plt.ylabel("Nombre de réservations")
plt.show()
```

2.2 Aperçu des variables explicatives

[ ]:
```
# Transformer les variables temporelles en type 'category'
data['arrival_year'] = data['arrival_year'].astype('category')
data['arrival_month'] = data['arrival_month'].astype('category')
data['arrival_date'] = data['arrival_date'].astype('category')
```

[ ]:
```
var_num = data.select_dtypes(include=["int64", "float64"]).columns
var_num = [col for col in var_num ]
var_num
```

[ ]:
```
data[var_num].describe()
```

[ ]:
```
var_char = data.select_dtypes(include=["object", "category"]).columns
var_char= [col for col in var_char ]
var_char
```

[ ]:
```
occurrences = {var: data[var].value_counts(dropna=False) for var in var_char}

for var, counts in occurrences.items():
    print(f"Occurrences pour la variable '{var}':")
    print(counts)
    print("-" * 40)
```

3 Statistiques Descriptives

3.1 Analyse univarié

3.1.1 Analyse des variables numeriques

[ ]:
```
n_cols = 3
n_rows = (len(var_num) + n_cols - 1) // n_cols

plt.figure(figsize=(n_cols * 5, n_rows * 5))

for i, col in enumerate(var_num):
```

```
    plt.subplot(n_rows, n_cols, i + 1)
    sns.histplot(data[col], kde=True, color='skyblue', bins=20)
    plt.title(f'Distribution de {col}')
    plt.xlabel(col)
    plt.ylabel('Fréquence')

plt.tight_layout()
plt.show()
```

[ ]:
```
n_cols = 3
n_rows = (len(var_num) + n_cols - 1) // n_cols

plt.figure(figsize=(n_cols * 5, n_rows * 5))

for i, col in enumerate(var_num):
    plt.subplot(n_rows, n_cols, i + 1)
    sns.boxplot(y=data[col], palette='Set2')
    plt.title(f'Distribution de {col}')
    plt.ylabel(col)

plt.tight_layout()

plt.show()
```

- **Analyse de la variable lead_time**

[ ]:
```
def hist_box(data, var):
    fig, (ax_box, ax_hist) = plt.subplots(nrows=2, sharex=True,␣
 ↪gridspec_kw={"height_ratios": (.15, .85)})

    # Boxplot
    sns.boxplot(data=data, x=var, ax=ax_box)
    ax_box.set(xlabel='')
    ax_box.set_title(f"Histogramme et Boxplot pour la variable '{var}'",␣
 ↪fontsize=14)

    # Histogramme
    sns.histplot(data=data, x=var, kde=True, ax=ax_hist)
    ax_hist.set(xlabel=var, ylabel='Fréquence')

    plt.tight_layout()
    plt.show()

hist_box(data, 'lead_time')
```

[ ]:
```
summary_400 = data.loc[data["lead_time"] > 400, "booking_status"].value_counts()
summary_300 = data.loc[data["lead_time"] > 300, "booking_status"].value_counts()
```

```
summary_df = pd.DataFrame({
    'lead_time > 400': summary_400,
    'lead_time > 300': summary_300
}).fillna(0).astype(int)
print(summary_df)
```

Analyse de la variable avg_price_per_room

[ ]:
```
hist_box(data, 'avg_price_per_room')
```

[ ]:
```
sns.boxplot(x='market_segment_type', y='avg_price_per_room', data=data)
plt.xlabel('market_segment_type')
plt.ylabel('Prix moyen par chambre')
plt.xticks(rotation=45)
plt.show()
```

[ ]:
```
print(data[data["avg_price_per_room"] == 0].shape[0]/data.shape[0] *100)
data.loc[data["avg_price_per_room"] == 0, "market_segment_type"].value_counts()
```

[ ]:
```
data[data["avg_price_per_room"] > 500]
```

[ ]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='room_type_reserved', y='avg_price_per_room', data=data)
plt.title('Distribution des prix moyens par chambre en fonction du type de␣
 ↪chambre réservée')
plt.xlabel('Type de chambre réservée')
plt.ylabel('Prix moyen par chambre')
plt.xticks(rotation=45)
plt.show()
```

Gestion des valeurs aberrantes à l'aide de l'intervalle interquartile (IQR)

[ ]:
```
Q1 = data["avg_price_per_room"].quantile(0.25)
Q3 = data["avg_price_per_room"].quantile(0.75)
IQR = Q3 - Q1
Upper_Whisker = Q3 + 1.5 * IQR
data.loc[data["avg_price_per_room"] >= 500, "avg_price_per_room"] =␣
 ↪Upper_Whisker

data.loc["INN33115"]
```

- ***Analyse de la variable no_of_children**

[ ]:
```
hist_box(data, 'no_of_children')
```

[ ]:
```
data['no_of_children'].value_counts(normalize=True)
```

[ ]:
```
data["no_of_children"] = data["no_of_children"].replace([9, 10], 3)
data['no_of_children'].value_counts(normalize=True)
```

3.1.2 Analyse des variables categorielles

[ ]:
```
var_char = [col for col in var_char if col != "booking_status"]
for col in var_char:

    plt.figure(figsize=(8,4))
    sns.countplot(y=col, data=data, order=data[col].value_counts().index,␣
 ↪palette='viridis')
    plt.title(f'Distribution de {col}')
    plt.xlabel('Nombre d\'observations')
    plt.ylabel(col)
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.show()
```

3.2 Analyse multivariré

3.2.1 Analyse des variables numeriques en fonction de la variable cible :

[ ]:
```
num_vars = len(var_num)
rows = (num_vars // 3) + (num_vars % 3 > 0)
plt.figure(figsize=(21, 7 * rows))

for i, var in enumerate(var_num[:-1], start=1):
    plt.subplot(rows, 3, i)

    # kdeplot
    sns.kdeplot(
        data=data,
        x=var,
        hue="booking_status",
        fill=True,
        palette="Purples",
        common_norm=True
    )
    sns.despine(top=True, right=True, bottom=True, left=True)
    plt.tick_params(axis="both", which="both", bottom=False, top=False,␣
 ↪left=False)
    plt.xlabel("")
    plt.title(var, fontsize=14)

plt.tight_layout()
plt.show()
```

3.2.2 Analyse des variables catégorielles en fonction de la variable cible :

[ ]:
```
var_char = [col for col in var_char if col != "booking_status"]

for col in var_char:
    plt.figure(figsize=(10, 5))
    sns.countplot(y=col, hue='booking_status', data=data,
                  order=data[col].value_counts().index, palette='Set2')

    plt.title(f'Répartition de {col} selon booking_status', fontsize=15)
    plt.xlabel('Nombre d\'observations')
    plt.ylabel(col)
    plt.legend(title='booking_status')
    plt.grid(axis='x', linestyle='--', alpha=0.7)

    plt.show()
```

3.3 Analyse de correlation et liaison avec la variable cible

3.3.1 Analyse de la correlation entre les variables

Variables Quantitatives : Matrice de Correlation

[ ]:
```
corr_matrix = data[var_num].corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', center=0)
plt.title('Matrice de Corrélation des Variables Continues')
plt.show()
```

Variables Qualitatives : test de Khi-deux et V de cramer

[ ]:
```
results = {}

for i in range(len(var_char)):
    for j in range(i + 1, len(var_char)):
        var1, var2 = var_char[i], var_char[j]
        contingency_table = pd.crosstab(data[var1], data[var2])

        chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

        n = contingency_table.sum().sum()
        k = min(contingency_table.shape)
        cramer_v = np.sqrt(chi2_stat / (n * (k - 1))) if k > 1 else 0

        results[f'{var1} & {var2}'] = {
            "chi2_statistic": chi2_stat,
            "p_value": p_value,
            "cramer_v": cramer_v
```

```
        }

results_df = pd.DataFrame(results).T
results_df.columns = ['Chi2 Statistic', 'P-Value', 'Cramer V']

results_df
```

3.3.2 Liens entre "booking_status" et les variables explicatives :

Variables Quantitatives : test de Student

[ ]:
```
results = []
for var in var_num:

    groupe_annule = data[data['booking_status'] == 'Canceled'][var]
    groupe_non_annule = data[data['booking_status'] == 'Not_Canceled'][var]
    t_stat, p_value = st.ttest_ind(groupe_annule, groupe_non_annule,␣
 ↪equal_var=False)

    results.append({
        "Variable": var,
        "Statistique t": t_stat,
        "P-Value": p_value
    })


results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by="P-Value")

print(results_df.round(4))
```

Variables catégorielles : test de Chi2

[ ]:
```
var_char
results_chi2 = []

for col in var_char:
    contingency_table = pd.crosstab(data[col], data['booking_status'])
    chi2_stat, p_val, dof, expected = chi2_contingency(contingency_table)

    results_chi2.append({
        "Variable": col,
        "Chi2 Statistique": chi2_stat,
        "P-Value": p_val
    })

chi2_results_df = pd.DataFrame(results_chi2).sort_values(by="P-Value")
```

```
print(chi2_results_df.round(4))
```

4 Préparation des données pour la modélisation

4.1 Préparation variables catégorielles et de la variable cible

4.1.1 Preparation et Encodage des variables categorielles

[ ]:
```
# X['lead_time'] = np.log1p(X['lead_time'])
# X['avg_price_per_room'] = np.log1p(X['avg_price_per_room'])
```

[ ]:
```
df = data.copy()
```

[ ]:
```
df.info()
```

[ ]:
```
# Recodage de la variable cible :
df['booking_status'] = df['booking_status'].map({'Canceled': 1, 'Not_Canceled':␣
↪0})
```

[ ]:
```
q1 = np.quantile(df['lead_time'], 0.25)
q3 = np.quantile(df['lead_time'], 0.75)
diff = q3 - q1
df['lead_time_out'] = df['lead_time'] > q3 + (1.5 * diff)

freq_map = df['lead_time_out'].value_counts(normalize=True)
df['lead_time_out'] = df['lead_time_out'].map(freq_map)
```

[ ]:
```
# Definir la variable cible
target = 'booking_status'
```

[ ]:
```
df.drop(['repeated_guest', 'arrival_year'], axis=1, inplace=True)
```

[ ]:
```
df.info()
```

- **Encodage des variables temporelles : Cyclical encoding**

[ ]:
```
df['arrival_date'] = df['arrival_date'].astype('int64')
df['arrival_month'] = df['arrival_month'].astype('int64')

# we assum that the madx is  31 days
df['arrival_date_sin'] = np.sin(2 * np.pi * df['arrival_date'] / 31)
df['arrival_date_cos'] = np.cos(2 * np.pi * df['arrival_date'] / 31)

#
df['arrival_month_sin'] = np.sin(2 * np.pi * df['arrival_month'] / 12)
df['arrival_month_cos'] = np.cos(2 * np.pi * df['arrival_month'] / 12)
```

[ ]:
```
df.drop(['arrival_date','arrival_month'], axis=1,inplace=True)
```

[ ]:
```
df.columns
```

- **Encodage des variables avec plus de 4 modalités:**

[ ]:
```
categorical_columns = df.select_dtypes(include=['object', 'category']).columns
categorical_columns
```

[ ]:
```
var_cat_plus_4 = []
var_cat_moins_4 = []

for var in categorical_columns :
    print(df[var].value_counts())
    print(len(df[var].value_counts()))
    if len(df[var].value_counts()) >= 4 :
        var_cat_plus_4.append(var)
    else :
        var_cat_moins_4.append(var)

print(var_cat_plus_4,var_cat_moins_4)
```

[ ]:
```
# Les variables catégorielles a plus de 4 modalités sont remplacées par la␣
 ↪variable moyenne de la variable cible par modalité

for v in var_cat_plus_4:
    tmp = pd.DataFrame(df.groupby(by=[v])[target].mean())
    df= df.join(tmp, on=v, how='left', lsuffix='', rsuffix= "_%_"+ v ,␣
 ↪sort=False)
```

Encodage des variables binaires

[ ]:
```
df["required_car_parking_space"] = df["required_car_parking_space"].
 ↪astype('int64')
```

[ ]:
```
df.info()
```

4.1.2 Separtaion et encodage de la variable cible

[ ]:
```
var_continues = list(df.select_dtypes(include=['int64', 'float64']).columns)
len(var_continues)
```

[ ]:
```
var_continues
```

[ ]:
```
X_var_continues = var_continues.remove(target)
```

[ ]:
```
Y = df[target]
```

[ ]:
```
colonnes_presentes = [col for col in var_continues if col in df.columns]
X = df[colonnes_presentes]
```

[ ]:
```
X.columns
```

[ ]:
```
Y.head()
```

[ ]:
```
X.head()
```

4.2 Standardisation des variables

[ ]:
```
scaler = RobustScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
X_scaled.head()
```

[ ]:
```
X_scaled.shape
```

4.3 Séparation en train/test

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, Y,
    test_size=0.2,
    stratify=Y,
    random_state=42
)

# Vérification des dimensions
print("X_train shape :", X_train.shape)
print("y_train shape :", y_train.shape)
print("X_test shape :", X_test.shape)
print("y_test shape :", y_test.shape)
```

5 Modélisation

[ ]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import log_loss
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.metrics import roc_curve, auc

# from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.linear_model import lasso_path, LassoCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

import random
from skopt import BayesSearchCV
from skopt.space import Real, Integer,Categorical
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from xgboost import XGBClassifier
import optuna
```

5.1 Arbre de décision :

5.1.1 Modèle d'initialisation

[ ]:
```
tree = DecisionTreeClassifier(criterion='gini',
                              splitter='best',
                              min_samples_leaf=20, # augmentation de ce␣
 ↪parametre pour forcer l'arbre à ne pas trop se spécialiser sur des petits␣
 ↪groupes d'observations.
                              max_depth=5, # imiter la profondeur maximale de␣
 ↪l'arbre pour éviter qu'il apprenne trop de détails spécifiques à␣
 ↪l'échantillon d'entraînement.
                              random_state=42)
# il faut optimiser ces parametres
```

[ ]:
```
tree.fit(X_train,y_train)
print(tree)
```

[ ]:
```
y_train_predict = tree.predict(X_train)
print(y_train_predict)
```

[ ]:
```
y_test_predict = tree.predict(X_test)
```

[ ]:
```
y_train_predict_proba = tree.predict_proba(X_train)[:,1]# On conserve en␣
 ↪mémoire uniquement la probabilité de l'événement cible pour nos graphiques
print(y_train_predict_proba)
y_test_predict_proba = tree.predict_proba(X_test)[:,1]
```

[ ]:
```
plt.figure(figsize=(100,100))
plot_tree(tree, feature_names = var_continues,  max_depth=5, filled = True,␣
 ↪fontsize=50)
plt.show()
```

[ ]:
```
# Importance des variables
importance_variable = pd.DataFrame()
importance_variable["Variable"]= var_continues
importance_variable["Feature Importance"] = tree.feature_importances_
importance_variable.sort_values(by = "Feature Importance", axis=0,␣
 ↪ascending=False, inplace=True)

print("Les 5 variables les plus importantes : ")
importance_variable.head(5)
```

[ ]:
```
importance_variable
```

[ ]:
```
select_var = ["lead_time",
"booking_status_%_market_segment_type",
"no_of_special_requests",
"avg_price_per_room"]
```

Évaluation du modele :

[ ]:
```
fpr_train, tpr_train, _ = roc_curve(y_train, y_train_predict_proba)
roc_auc_train = auc(fpr_train, tpr_train)
fpr_test, tpr_test, _ = roc_curve(y_test, y_test_predict_proba)
roc_auc_test = auc(fpr_test, tpr_test)

plt.figure()
lw = 2
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw, label='Train - ROC curve (area = %0.2f)' % roc_auc_train)

plt.plot(fpr_test, tpr_test, color='darkgreen',
         lw=lw, label='Test - ROC curve (area = %0.2f)' % roc_auc_test)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Comparaison courve ROC (TRAIN / TEST)')
plt.legend(loc="lower right")
plt.show()
```

[ ]:
```
print("log loss app : " + str(log_loss(y_train, y_train_predict_proba)))
print("log loss test : " + str(log_loss(y_test, y_test_predict_proba)))
```

[ ]:
```
def metrics_score(actual, predicted):
    print(classification_report(actual, predicted))
```

```
    cm = confusion_matrix(actual, predicted)
    plt.figure(figsize=(8,5))

    sns.heatmap(cm, annot=True,  fmt='.2f', xticklabels=['Not Cancelled',␣
 ↪'Cancelled'], yticklabels=['Not Cancelled', 'Cancelled'])
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
```

[ ]:
```
y_train_predict = tree.predict(X_train)
y_test_predict = tree.predict(X_test)
```

[ ]:
```
metrics_score(y_train, y_train_predict)
```

[ ]:
```
metrics_score(y_test, y_test_predict)
```

5.1.2 Optimisation du modele

[ ]:
```
# Création du dictionnaire des indicateurs que nous souhaitons testés pour la␣
 ↪méthode Random ou GridSearch
param_dict = {
    'criterion': ['gini', 'entropy'], # Le critère de split des arbres
    'splitter': ['best','random'],    # Est-ce que l'on teste un échantillon de␣
 ↪variable (random)
                                      #ou toutes les variables (best) à␣
 ↪chaque neoud
    'max_depth': [3,4,10],  # Profondeur maximum de l'arbre
    'min_samples_split': [2,4], # # Nombre d'observations mimnimum pour créer␣
 ↪un split
    'min_samples_leaf': [1,5,10],     # Nombre d'observations minimum dans une␣
 ↪feuille
    'min_weight_fraction_leaf': [0,0.01], # Proportion minimum des observations␣
 ↪dans une feuille
    'max_features': ['log2',"sqrt"]}

#Création  du dictionnaire de recherche pour la méthode d'optimisation␣
 ↪bayesienne
clf = DecisionTreeClassifier()
param_dict_bayes = {
    'criterion': Categorical(['gini', 'entropy']),
    'splitter': Categorical(['best','random']),
    'max_depth': Integer(3,30),
    'min_samples_split': Integer(2,50),
    'min_samples_leaf': Integer(1,20),
    'min_weight_fraction_leaf': Real(0,0.5, prior='uniform')}
```

```
NB_ITER = 5



def random_parameter(clf,param_dict,n_iter,X_train,y_train,nb_cv) :
    res = pd.DataFrame()
    compt = 0
    num_iter = []
    auc=[]
    param = []
    while compt <n_iter :
        compt = compt +1
        params = {key: random.sample(value, 1)[0] for key, value in param_dict.
 ↪items()}
        clf.set_params(**params)
        scores = cross_validate(clf, X_train, y_train, cv=5,
                        scoring = ['roc_auc'])
        num_iter.append(compt)
        param.append(params)
        auc.append(scores['test_roc_auc'].mean())

    res["Num_ITER"] = num_iter
    res["Param"] = param
    res["Auc"] = auc

    return res

Random_Res_Tree = random_parameter(DecisionTreeClassifier()␣
 ↪,param_dict,NB_ITER,X_train,y_train,5)
print(" #### RECHERCHE ALEATOIRE #### ")
Random_Res_Tree.sort_values('Auc', ascending = False, inplace = True)
Random_Res_Tree.head()
best_param_random_search = list(Random_Res_Tree["Param"])[0]
print("\n Paramètres recherche aléatoire : ")
print(best_param_random_search)

print("\n Résultats recherche aléatoire : " + str(Random_Res_Tree['Auc'].max()))




Grid_Search =␣
 ↪GridSearchCV(DecisionTreeClassifier(),param_dict,scoring='roc_auc',cv=5)
Grid_Search.fit(X_train,y_train)
print(" #### RECHERCHE GRID SEARCH #### ")
print("\n Paramètres grid search : ")
```

```
best_param_gid_search = Grid_Search.best_params_
print(best_param_gid_search)
best_score_grid_search = Grid_Search.best_score_
print("\n Résultats grid search  : " + str(best_score_grid_search))



opt = BayesSearchCV(clf,param_dict_bayes , n_iter=NB_ITER,cv=5,scoring =␣
 ↪'roc_auc')
opt.fit(X_train, y_train)
print(" #### RECHERCHE OPTIMISATION #### ")
print("\n Paramètres grid search : ")
best_param_opti_bayes =opt.best_params_
print(best_param_opti_bayes)
best_score_opti_bayes = opt.best_score_
print("\n Résultats grid search  : " + str(best_score_opti_bayes))
```

[ ]:
```
tree = DecisionTreeClassifier(criterion='entropy',
                              splitter='best',
                              max_depth=10,
                              min_samples_leaf=5,
                              min_samples_split=2,
                              min_weight_fraction_leaf=0,
                              max_features='sqrt',
                              random_state=42)

tree.fit(X_train, y_train)
```

[ ]:
```
y_train_predict = tree.predict(X_train)
print(y_train_predict)
```

[ ]:
```
y_test_predict = tree.predict(X_test)
```

[ ]:
```
y_train_predict_proba = tree.predict_proba(X_train)[:,1]
print(y_train_predict_proba)
y_test_predict_proba = tree.predict_proba(X_test)[:,1]
```

[ ]:
```
plt.figure(figsize=(100,100))
plot_tree(tree, feature_names = var_continues,  max_depth=5, filled = True,␣
 ↪fontsize=50)
plt.show()
```

[ ]:
```
# Importance des variables
importance_variable = pd.DataFrame()
importance_variable["Variable"]= var_continues
importance_variable["Feature Importance"] = tree.feature_importances_
```

```
importance_variable.sort_values(by = "Feature Importance", axis=0,␣
 ↪ascending=False, inplace=True)

print("Les 5 variables les plus importantes :")
importance_variable.head(5)
```

[ ]:
```
importance_variable
```

Évaluation du modele :

[ ]:
```
fpr_train, tpr_train, _ = roc_curve(y_train, y_train_predict_proba)
roc_auc_train = auc(fpr_train, tpr_train)
fpr_test, tpr_test, _ = roc_curve(y_test, y_test_predict_proba)
roc_auc_test = auc(fpr_test, tpr_test)

plt.figure()
lw = 2
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw, label='Train - ROC curve (area = %0.2f)' % roc_auc_train)

plt.plot(fpr_test, tpr_test, color='darkgreen',
         lw=lw, label='Test - ROC curve (area = %0.2f)' % roc_auc_test)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Comparaison courve ROC (TRAIN / TEST)')
plt.legend(loc="lower right")
plt.show()
```

[ ]:
```
print("log loss app : " + str(log_loss(y_train, y_train_predict_proba)))
print("log loss test : " + str(log_loss(y_test, y_test_predict_proba)))
```

[ ]:
```
def metrics_score(actual, predicted):
    print(classification_report(actual, predicted))

    cm = confusion_matrix(actual, predicted)
    plt.figure(figsize=(8,5))

    sns.heatmap(cm, annot=True,  fmt='.2f', xticklabels=['Not Cancelled',␣
 ↪'Cancelled'], yticklabels=['Not Cancelled', 'Cancelled'])
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
```

[ ]:
```
y_train_predict = tree.predict(X_train)
y_test_predict = tree.predict(X_test)
```

[ ]:
```
metrics_score(y_train, y_train_predict)
```

[ ]:
```
metrics_score(y_test, y_test_predict)
```

Seuils

[ ]:
```
#Choix du seuil
precision_train, recall_train, thresholds_train =␣
 ↪precision_recall_curve(y_train,
                                                                  ␣
 ↪y_train_predict_proba)
precision_test, recall_test, thresholds_test = precision_recall_curve(y_test,
                                                                  ␣
 ↪y_test_predict_proba)
table_choix_seuil = pd.DataFrame()
table_choix_seuil["SEUIL"] = [0] + list(thresholds_train)
table_choix_seuil["Precision_train"] = precision_train
table_choix_seuil["Recall_train"] = recall_train

f1_scores = 2 * (precision_train * recall_train) / (precision_train +␣
 ↪recall_train)
table_choix_seuil["f1_scores"] = f1_scores


table_choix_seuil.sort_values(by = "SEUIL", axis=0, ascending=False,␣
 ↪inplace=True)
print(table_choix_seuil)
```

[ ]:
```
max_f1_score_row = table_choix_seuil.loc[table_choix_seuil['f1_scores'].
 ↪idxmax()]
print(max_f1_score_row)
```

[ ]:
```
table_choix_seuil
```

[ ]:
```
# seuil optimal
y_proba = tree.predict_proba(X_test)[:, 1]
y_pred = (y_proba >= 0.409).astype(int)
```

[ ]:
```
metrics_score(y_test, y_test_predict)
```

5.2 Régression logistique

5.2.1 Model d'initialisation

[ ]:
```
regLog1 = LogisticRegression(max_iter=500, solver='lbfgs')
regLog1.fit(X_train,y_train)

y_train_predict_proba = regLog1.predict_proba(X_train)[:,1]
y_test_predict_proba = regLog1.predict_proba(X_test)[:,1]
```

[ ]:
```
fpr_train, tpr_train, _ = roc_curve(y_train, y_train_predict_proba)
roc_auc_train = auc(fpr_train, tpr_train)
fpr_test, tpr_test, _ = roc_curve(y_test, y_test_predict_proba)
roc_auc_test = auc(fpr_test, tpr_test)

plt.figure()
lw = 2
plt.plot(fpr_train, tpr_train, color='darkorange',
         lw=lw, label='Train - ROC curve (area = %0.2f)' % roc_auc_train)

plt.plot(fpr_test, tpr_test, color='darkgreen',
         lw=lw, label='Test - ROC curve (area = %0.2f)' % roc_auc_test)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Comparaison courve ROC (TRAIN / TEST)')
plt.legend(loc="lower right")
plt.show()
```

[ ]:
```
y_predict = regLog1.predict(X_test)
metrics_score(y_test, y_predict)
```

[ ]:
```
table_coeff = pd.DataFrame()
table_coeff["Variable"]=X_train.columns
table_coeff["Coefficient"] = regLog1.coef_[0]
table_coeff["Odds Ratio"] = np.exp(table_coeff["Coefficient"])

print(table_coeff.sort_values(by='Coefficient', key=abs, ascending=False))

print("Intercept : " + str(regLog1.intercept_))
```

5.2.2 Selection de variable avec Lasso

[ ]:
```
alphas_lasso, coefs_lasso, _ = lasso_path(X_train.values, y_train)
```

[ ]:
```
feature_names = X_train.columns

plt.figure(figsize=(12, 8))

for i, coef in enumerate(coefs_lasso):
    plt.plot(alphas_lasso, coef, label=feature_names[i])

plt.xlabel('Alpha')
plt.ylabel('Coefficients')
plt.title('LASSO Path')
plt.xscale('log')
plt.gca().invert_xaxis()


plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize='small')
plt.tight_layout()

plt.show()
```

[ ]:
```
lasso_cv = LassoCV(alphas=np.logspace(-4, 0, 50), cv=5)
lasso_cv.fit(X_train, y_train)
```

[ ]:
```
threshold = 0.01

selected_features = X_train.columns[np.abs(lasso_cv.coef_) > threshold]
coefficients = lasso_cv.coef_[np.abs(lasso_cv.coef_) > threshold]

print(f"Variables sélectionnées : {selected_features.tolist()}")
```

[ ]:
```
selected_features = ['no_of_children', 'no_of_weekend_nights',␣
 ↪'no_of_week_nights', 'required_car_parking_space', 'lead_time',␣
 ↪'no_of_previous_cancellations', 'no_of_previous_bookings_not_canceled',␣
 ↪'avg_price_per_room', 'no_of_special_requests', 'lead_time_out',␣
 ↪'arrival_month_sin', 'arrival_month_cos',␣
 ↪'booking_status_%_type_of_meal_plan', 'booking_status_%_market_segment_type']
X_train_lasso = X_train[selected_features]
X_test_lasso = X_test[selected_features]
```

[ ]:
```
coef_df = pd.Series(coefficients, index=selected_features).sort_values(key=abs,␣
 ↪ascending=False)

plt.figure(figsize=(8, 5))
sns.barplot(x=coef_df.values, y=coef_df.index, palette="viridis")
```

```
plt.title("Importance des variables issues de la régression Lasso")
plt.xlabel("Coefficient")
plt.ylabel("Variables")
plt.axvline(0, color='gray', linestyle='--', lw=1)  # Ligne verticale pour␣
 ↪séparer les coefficients positifs et négatifs
plt.show()
```

[ ]:
```
regLog_lasso = LogisticRegression()
regLog_lasso.fit(X_train_lasso, y_train)

y_pred = regLog_lasso.predict(X_test_lasso)

conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

metrics_score(y_test, y_pred)
```

5.2.3 Sélection de variable par regression logistique statsmodels

[ ]:
```
X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
```

[ ]:
```
import statsmodels as sm

from statsmodels.api import Logit

X_train["const"] =1
X_test["const"]=1 # pour ajouter l'intercept
lr = Logit(endog=y_train,exog=X_train)

reg = lr.fit()
print(reg.summary())
```

[ ]:
```
p_values = reg.pvalues
```

[ ]:
```
variables_significatives = p_values[p_values < 0.05].index.tolist()
variables_non_significatives = p_values[p_values >= 0.05].index.tolist()
```

[ ]:
```
variables_significatives
```

[ ]:
```
variables_non_significatives
```

[ ]:
```
X_train_sel = X_train[variables_significatives]
X_test_sel = X_test[variables_significatives]
```

[ ]:
```
regLog_sel = LogisticRegression()
regLog_sel.fit(X_train_sel, y_train)

y_pred = regLog_sel.predict(X_test_sel)

conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

metrics_score(y_test, y_pred)
```

5.2.4 Optimisation du modele par Grid Search et Poids

[ ]:
```
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization strength
    'penalty': ['l1', 'l2'],  # Type de régularisation
    'solver': ['liblinear']  # Solver adapté à l1/l2
}

log_reg = LogisticRegression(max_iter=500, random_state=42)

grid_search = GridSearchCV(log_reg, param_grid, cv=5, scoring='accuracy',␣
 ↪verbose=1)
grid_search.fit(X_train_lasso, y_train)

best_params = grid_search.best_params_
print(f"Best Params: {best_params}")


best_log_reg = grid_search.best_estimator_
y_pred = best_log_reg.predict(X_test_lasso)
```

[ ]:
```
metrics_score(y_test, y_pred)
```

Ajouter des poids pour les classes pour mieux gerer le déséquilibre de classes

[ ]:
```
log_reg_balanced = LogisticRegression(
    C=best_params['C'],
    penalty=best_params['penalty'],
    solver=best_params['solver'],
    class_weight='balanced',
    max_iter=500,
    random_state=42
)


log_reg_balanced.fit(X_train_lasso, y_train)
```

```
y_pred_balanced = log_reg_balanced.predict(X_test_lasso)
```

[ ]:
```
metrics_score(y_test, y_pred_balanced)
```

5.3 XGBoost

[ ]:
```
from xgboost import XGBClassifier
```

5.3.1 Modèle d'initialisation

[ ]:
```
xgb_model = XGBClassifier(
                objective="multi:softmax",
                num_class=2,
                booster="gbtree",
                eval_metric="mlogloss"
)
```

[ ]:
```
xgb_model.fit(X_train, y_train)
```

[ ]:
```
y_pred_proba = xgb_model.predict_proba(X_test)
```

[ ]:
```
initial_log_loss = log_loss(y_test, y_pred_proba)
print("Log Loss initiale :", initial_log_loss)
```

[ ]:
```
y_pred = xgb_model.predict(X_test)
```

[ ]:
```
metrics_score(y_test, y_pred)
```

5.3.2 Optimisation du modèle

Optimisation du modèle par GridSearch

[ ]:
```
params = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [3, 5, 7, 10],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.6, 0.7, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.7, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2, 0.5],
    'min_child_weight': [1, 3, 5, 7]
}

#random_search = RandomizedSearchCV(
#    XGBClassifier(objective="multi:softmax", num_class=2,␣
 ↪eval_metric="mlogloss", random_state=42),
#    param_distributions=params,
```

```
#    n_iter=30,
#    scoring='f1_weighted',
#    cv=5,
#    verbose=1,
#    n_jobs=-1,
#    random_state=42
#)

#random_search.fit(X_train, y_train)

print("Best parameters:", random_search.best_params_)
print("Best accuracy:", random_search.best_score_)
```

[ ]:
```
#Best model
optimized_xgb = XGBClassifier(
    objective="multi:softmax",
    num_class=2,
    eval_metric="mlogloss",
    subsample=0.8,
    n_estimators=300,
    min_child_weight=1,
    max_depth=10,
    learning_rate=0.05,
    gamma=0.1,
    colsample_bytree=0.7,
    random_state=42,
    use_label_encoder=False
)


optimized_xgb.fit(X_train, y_train)

y_pred = optimized_xgb.predict(X_test)
metrics_score(y_test, y_pred)
```

[ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(18, 6))  # 1 ligne, 2 colonnes

# Premier graphique : importance selon le 'gain'
plot_importance(optimized_xgb, ax=axes[0], importance_type='gain')
axes[0].set_title("Importance des caractéristiques (Gain)")

# Deuxième graphique : importance selon le 'weight'
plot_importance(optimized_xgb, ax=axes[1], importance_type='weight')
axes[1].set_title("Importance des caractéristiques (Weight)")

# Ajuster l'espace entre les graphiques
plt.tight_layout()
```

```
plt.show()
```

[ ]:
```
import shap
shap.initjs()
```

[ ]:
```
explainer = shap.TreeExplainer(optimized_xgb)
shap_values = explainer.shap_values(X_test)
```

[ ]:
```
classe_index = 1  # Classe positive
observation_index = 0  # Première observation

# Correction de force_plot
shap.force_plot(
    explainer.expected_value[classe_index],  # Base value pour la classe 1
    shap_values[observation_index, :, classe_index],  # SHAP values pour cette␣
 ↪observation et classe
    X_test.iloc[observation_index]  # Caractéristiques de l'observation
)
```

[ ]:
```
shap.summary_plot(shap_values[..., 1], X_test)  # Classe positive (1)
```

Optimisation du modèle avec Optuna

[ ]:
```
def objective(trial):
    selected_features = trial.suggest_categorical('features', [list(X.columns)])
    X_train_selected = X_train[selected_features]
    X_test_selected = X_test[selected_features]

    param = {
        'objective': 'multi:softmax',
        'num_class': 2,
        'booster': 'gbtree',
        'eval_metric': 'mlogloss',
        'n_estimators': trial.suggest_int('n_estimators', 50, 1000),
        'max_depth': trial.suggest_int('max_depth', 3, 30),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.1),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 15),
        'subsample': trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.01, 1.0),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.0, 1),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.0, 10)
    }

    mod = XGBClassifier(**param, random_state=42)
    scores = cross_validate(mod, X_train_selected, y_train, cv=3,␣
 ↪scoring='accuracy')
```

```
    return np.mean(scores['test_score'])  # Fixed line

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

print("Meilleurs paramètres:", study.best_params)
```

[ ]:
```
xgb_tunned = XGBClassifier(
                objective="multi:softmax",
                num_class=2,
                booster="gbtree",
                eval_metric="mlogloss",
                n_estimators=876,
                max_depth=14,
                learning_rate=0.012036591105960953,
                min_child_weight=1,
                subsample=0.9303558848762992,
                colsample_bytree=0.7479507299406605,
                reg_alpha=0.022053370844735434,
                reg_lambda=7.723870704304475,
                random_state=42
            )
```

[ ]:
```
xgb_tunned.fit(X_train, y_train)
```

[ ]:
```
y_pred_proba = xgb_tunned.predict_proba(X_test)
```

[ ]:
```
final_log_loss = log_loss(y_test, y_pred_proba)
print("Log Loss finale :", final_log_loss)
```

[ ]:
```
y_pred = xgb_tunned.predict(X_test)
```

[ ]:
```
metrics_score(y_test, y_pred)
```

[ ]:
```

```