Rapport Final

Apprentissage



Google Analytics Customer Revenue Prediction

Predict how much GStore customers will spend



Victor DULEBA (Data Science 3), Albéric DUFAURE (Data Science 2), Thomas DUVAL (Data Science 2)

SOMMAIRE

Introduction

Data pre-Processing

Visualisation

Modèles

Conclusion

INTRODUCTION

Le but de ce projet est de répondre à une compétition kaggle : <https://www.kaggle.com/c/ga-customer-revenue-prediction/overview> .

Dans cette compétition, le challenge est d’analyser une boutique en ligne Google, où est vendu différents produits de la marque, grâce au dataset fourni par Google Analytics et de prédire le revenu par consommateur.

On nous rappelle que la plupart des entreprises vérifient la règle du 80/20 : sur tous les visiteurs d’un site marchand, seulement 20% achèteront lors de cette visite.

A l’aide des différents algorithmes, fonctions et méthodes vus en cours, notre objectif est de modéliser les revenus générés par visiteur.

Ainsi dans ce rapport, nous préciserons notre démarche d’étude en expliquant tout d’abord les différents pré-traitements que nous avons fait sur notre dataset, puis nous explorerons les données avec quelques visualisations et enfin nous présenterons nos modèles et les résultats obtenus.

Ce rapport est à lire en parallèle du code R fourni.

DATA PRE-PROCESSING

Pour commencer, on rappellera qu’il a été décidé de travailler uniquement avec le train.csv fourni dans la compétition. La première problématique de ce dataset était la présence de json, ainsi un code était donné afin de désimbriquer ces json et enregistrer chaque feature dans sa propre colonne.

Dans les colonnes de base on retrouve certaines données comme :

* *fullVisitorId*- Un identifiant unique à chaque visiteur de Google Merchandise Store.
* *channelGrouping* – La channel via laquelle le visiteur est arrivé sur le Store.
* *date* – La data à laquelle l’utilisateur à visiter le Store.
* *device* – Les specifications de l’appareil utilisé pour accéder au Store.
* *geoNetwork* – Cette section contient les informations concernant la géographie de l’utilisateur
* *socialEngagementType* – Type d’engagement, "Socially Engaged" ou "Not Socially Engaged".
* *totals* – Cette section contient les valeurs agrégées de la session.
* *trafficSource* – Cette section contient les informations de la source de la session.
* *visitId* – L’identifiant de la session. Il s’agit d’une partie d’une valeure \_utmb cookie qui est unique à chaque visiteur. On retrouve sous *sessionId* l’ensemble identifiant du visiteur et identifiant de la session actuelle.
* *visitNumber* – Le numéro de la session du visiteur.
* *visitStartTime* – Timestanp du début de la visite.
* *hits* – On y retrouve tous les types de hits ainsi qu’un enregistrement de toutes les pages visitées.
* *totals* – Inclus des valeurs aggrégées de haut niveau.

Après avoir exécuté le code fourni, nous nous retrouvons avec un dataframe de taille 903653 lignes x 55 colonnes.

1. Exploration

La première étape à réaliser dans n’importe quel projet c’est l’exploration des données. Afin de savoir ce que l’on a, quelles sont les spécificités de nos données et ce que l’on pourra modifier par la suite.

Ainsi nous avons commencé par analyser le dataframe, comprendre chaque colonne et les variables qui la composait, nous avons également affiché les valeurs manquantes (NA) grâce à la librairie **visdat**.

On remarque tout de suite plusieurs choses, plusieurs colonnes sont peuplées uniquement de la variable ‘﻿not available in demo datase’. Mais on retrouve également un grand nombre de NA dans beaucoup de colonnes ainsi que des NA annotées ‘unknown.unknown’, ‘(not set)’, ‘not available in demo dataset’, ‘(not provided)’, ‘(none)’, ‘<NA>’ dans certaines colonnes. Autre chose que l’on remarque, c’est la présence de colonnes remplies d’une valeur constante sur l’ensemble de celles-ci.

On en conclut que certaines colonnes seront inutilisables pour la suite.

En analysant *transactionRevenue* on remarque les valeurs sont très hautes pour des achats, en effet elles ont été multipliées par 10^6, avec des valeurs qui vont de 10^5 à 2,3\*10^10. On prendra donc le logarithme naturel de cette valeur, comme il est indiqué de faire dans la compétition Kaggle, les valeurs s’étendent alors de 9,21 à 23,86.

Nous avons affiché les premières et dernières dates du jeu de donnée pour constater que celui-ci commence le 1er Août 2016 et se termine le 1er Août 2017.

Pour finir sur l’exploration, on dira que nombreuses sont les colonnes à modifier et traiter, que ce soit sur le type de données, sur les NA ou autre. On détaillera notre démarche dans la partie suivante.

1. Traitements

Suite aux observations précédentes, nous avons procédé à un certain nombre de traitements sur le dataset.

Nous avons choisi de travaillé uniquement sur la moitié du dataset pour des soucis de taille de données.

Tout d’abord, nous avons trouvé les colonnes composées d’une variable constante ou d’une variable ‘not available’, nous les avons donc supprimées.

Ensuite, nous avons converti *date* au format date %Y%m%d sur la timezone UTC, afin d’avoir une variable exploitable. Nous avons également converti *visitStartTime* en POSIXct car il s’agissait d’un timestamp. Ces deux variables offrent des informations sur la date de la visite et également sur la date de début de la visite. Nous avons donc décidé de décomposer *visitStartTime* en 5 colonnes : l’année, le mois, le jour (entier), le jour de la semaine et l’heure. Il est très généralement recommandé de faire ce genre de décomposition afin de pouvoir avoir beaucoup plus de paramètres pour l’étude.

De plus, pour les colonnes *hits, pageviews, page, bounces et newVisits* nous les avons convertis en integer.

Puis, lors de notre exploration nous avons trouvé des variables NA « masquées » en char, tel que ‘unknown.unknown’, ‘(not set)’, ‘not available in demo dataset’, ‘(not provided)’, ‘(none)’, ‘<NA>’. Nous avons donc remplacé chacune de ces variables par un NA normal pour uniformiser et éviter les erreurs.

Pour la feature *newVisits,* ayant pour valeur 1 s’il s’agit d’une nouvelle visite et NA sinon, nous avons changé les NA en 0, puis nous avons converti la colonne en logical afin d’avoir des TRUE FALSE.

Pour la feature *pageviews*, ayant pour valeur le nombre de pages vues, nous avons change les NA en 1 car le visiteur visite au moins une page.

Pour la feature *transactionRevenue,* ayant pour valeur le revenu ou alors NA en cas de non-achat, nous avons remplacé les NA par des 0 puis nous l’avons converti en numeric. Suite à cela, nous avons appliqué un log sur cette feature.

Après cela, nous nous sommes rendu compte que les colonnes *country, browser et operatingSystem* présentaient des variables qui apparaissaient très peu de fois comparé aux autres valeurs. Ainsi, nous avons décidé de remplacer par ‘Other’ chaque variable étant présentes moins de 1000 fois.

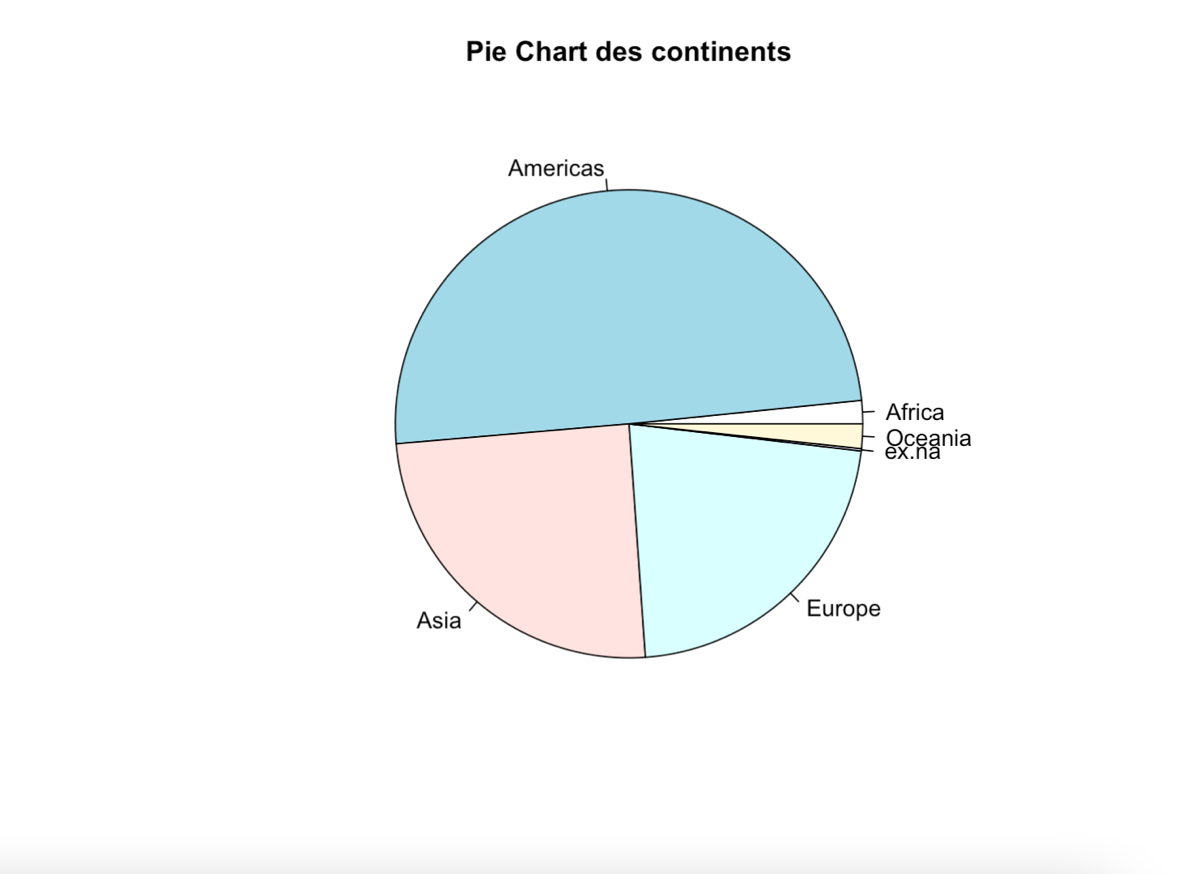
Enfin, ayant de nombreuses colonnes avec un grand nombre de valeurs manquantes, nous avons remplacé les NA dans *continent, subContinent, country, et operatingSystem* par ‘ex.na’.

Pour finir un certain nombre de features ont été converti en factor.

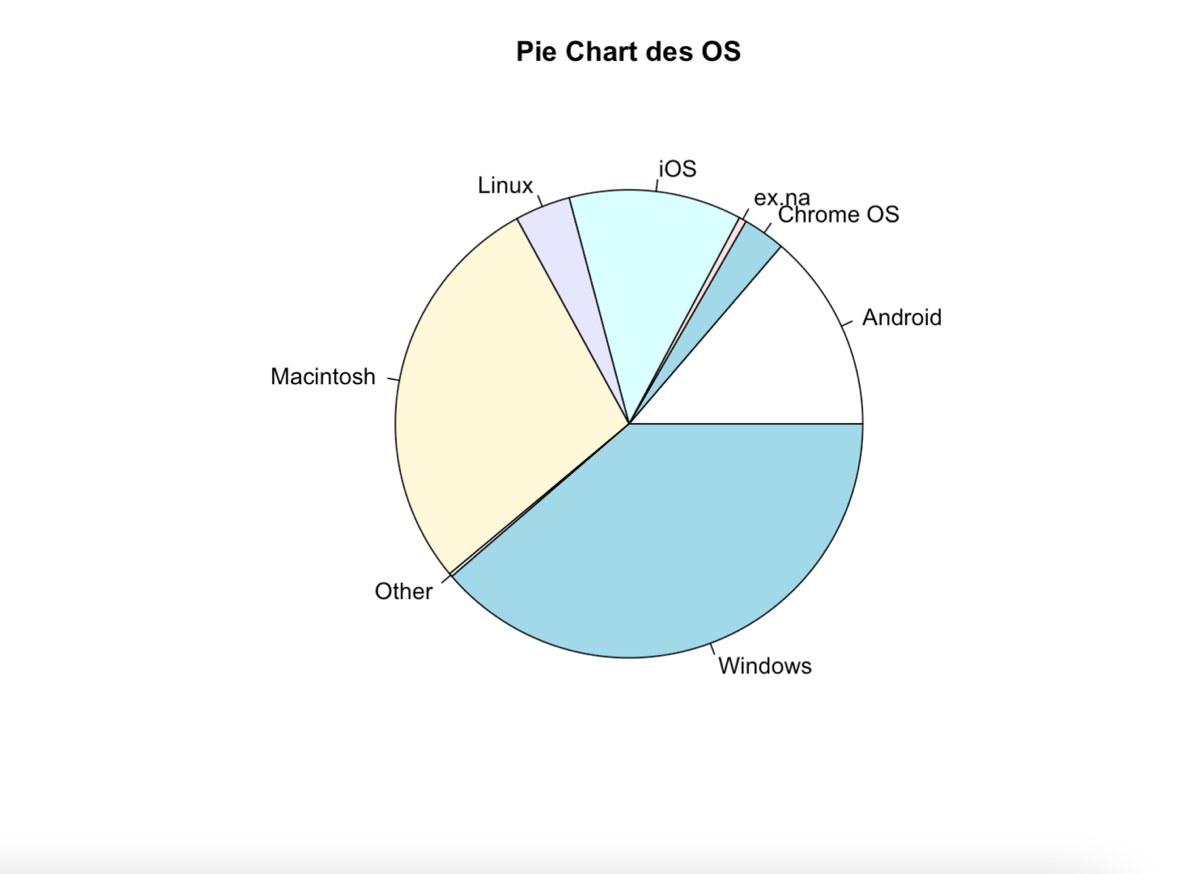
Finalement, on peut dire que le pre-processing des données est autant, voir plus important qu’un bon modèle de prédiction. Il est necessaire de faire cette étape soigneusement afin d’avoir les meilleurs résultats possibles.

DATA VISUALISATION

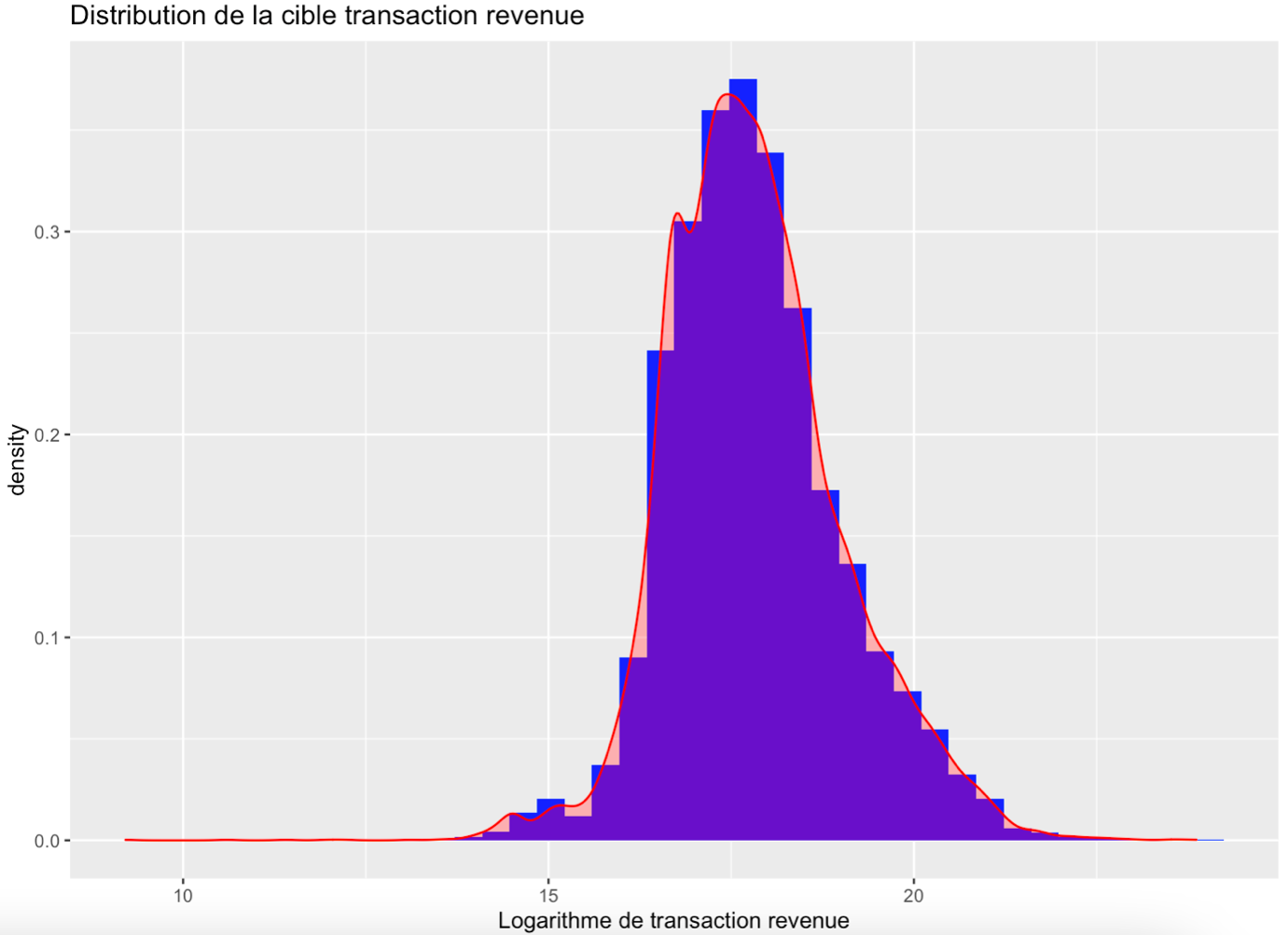
Nous avons procédé à différentes visualisations de notre dataset, cela permet d’interpréter nos données :



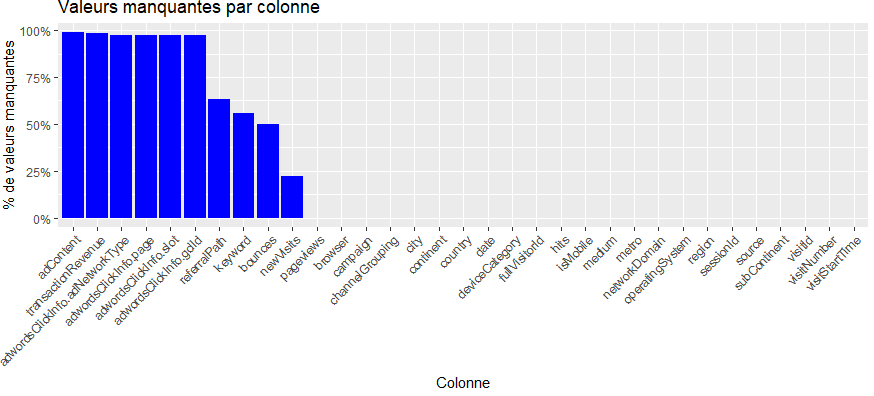
D’après cette première visualisation, on peut voir que quasiment la moitié des visiteurs du site sont du continent américain, puis environ ¼ d’Asie et un peu moins d’Europe.



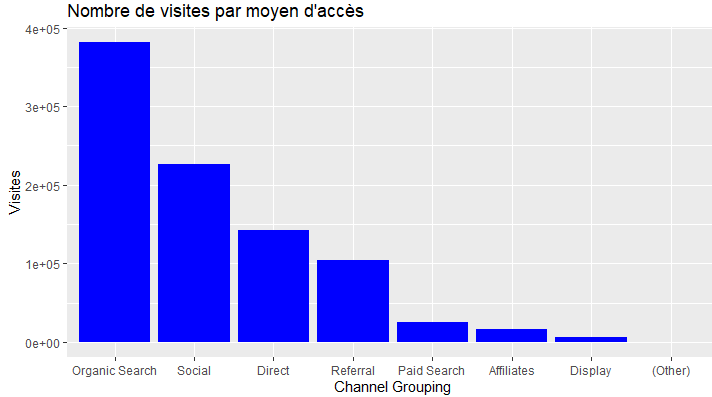
Ici, on peut voir les OS (Operating System) utilisés par les visiteurs pour accéder au site. C’est intéressant de voir la répartition où Windows arrive en premier suivi de MacOS puis iOS et Android. Les 2 plus gros OS d’ordinateur puis les 2 plus gros OS de smartphones.



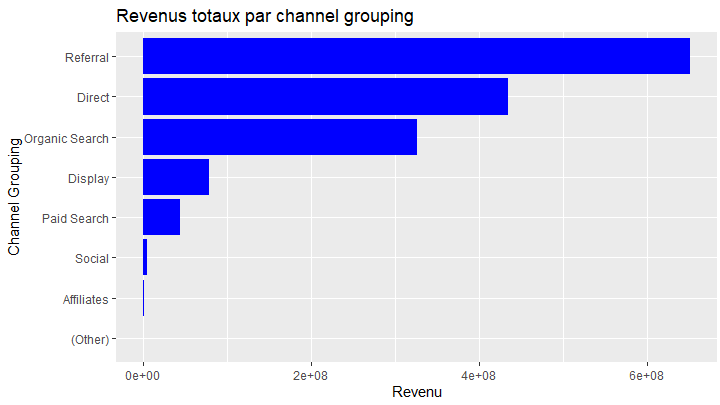
Ici, nous avons la répartition du logarithme des valeurs de notre cible, transaction revenue. Nous pouvons constater que les valeurs sont majoritairement comprises entre 17 et 20.



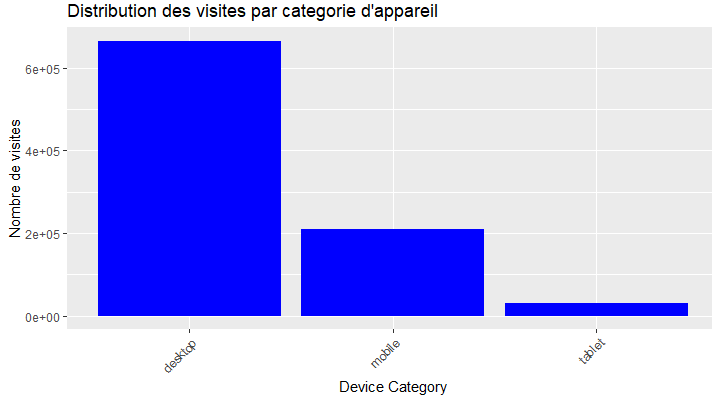
Nous avons également décidé d’afficher le nombre de valeurs manquantes par colonne. On peut donc voir que les colonnes adContent, transactionRevenue, adwordsClickInfo, referalPath, keyword, bounces et newVisits contiennent des valeurs manquantes, avec un pourcentage de valeurs manquantes proche de 100% pour les six premières colonnes. (voir la partie précédente *Traitement* pour une explication de notre démarche quant au traitement de ces valeurs manquantes).



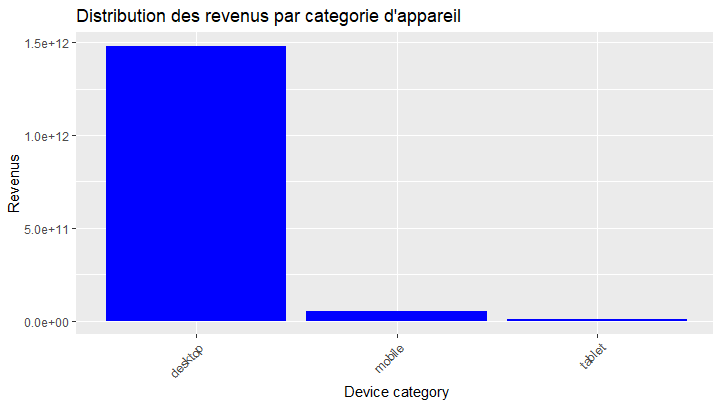
Ci-dessus nous avons le nombre de visite par moyen d’accès sur le site. Organic search signifie que l’utilisateur a effectué la recherche manuellement sur un navigateur, Social qu’il a cliqué sur un lien depuis un réseau social, Direct qu’il a écrit l’adresse dans la barre de recherche, Referral qu’il a cliqué sur un lien depuis un autre site.



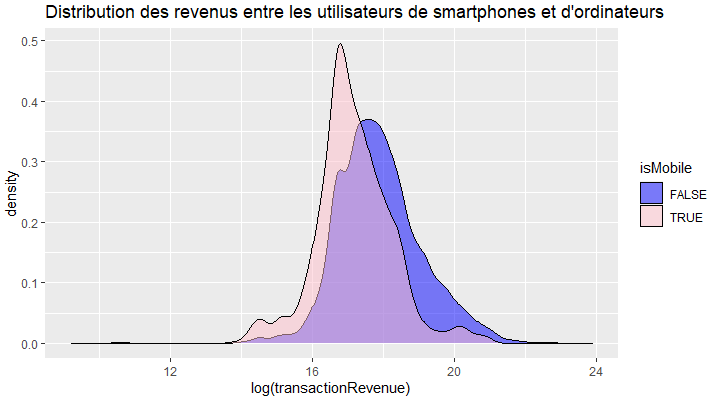
Ici, nous avons les revenus totaux générés par chaque moyen d’accès au site. Bien que le moyen d’accès “Referral” soit le quatrième plus populaire, c’est celui qui génère le plus de bénéfices. “Organic Search” génère quant à lui seulement la moitié des revenus par accès “Referral”.



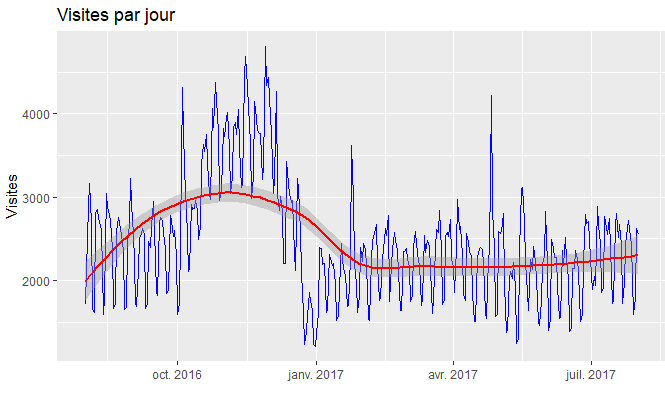
Voici le nombre de visites en fonction de l’appareil utilisé. Il y a donc une majorité d’utilisateurs d’ordinateurs fixes, suivis par les utilisateurs de smartphones, puis ceux de tablettes.



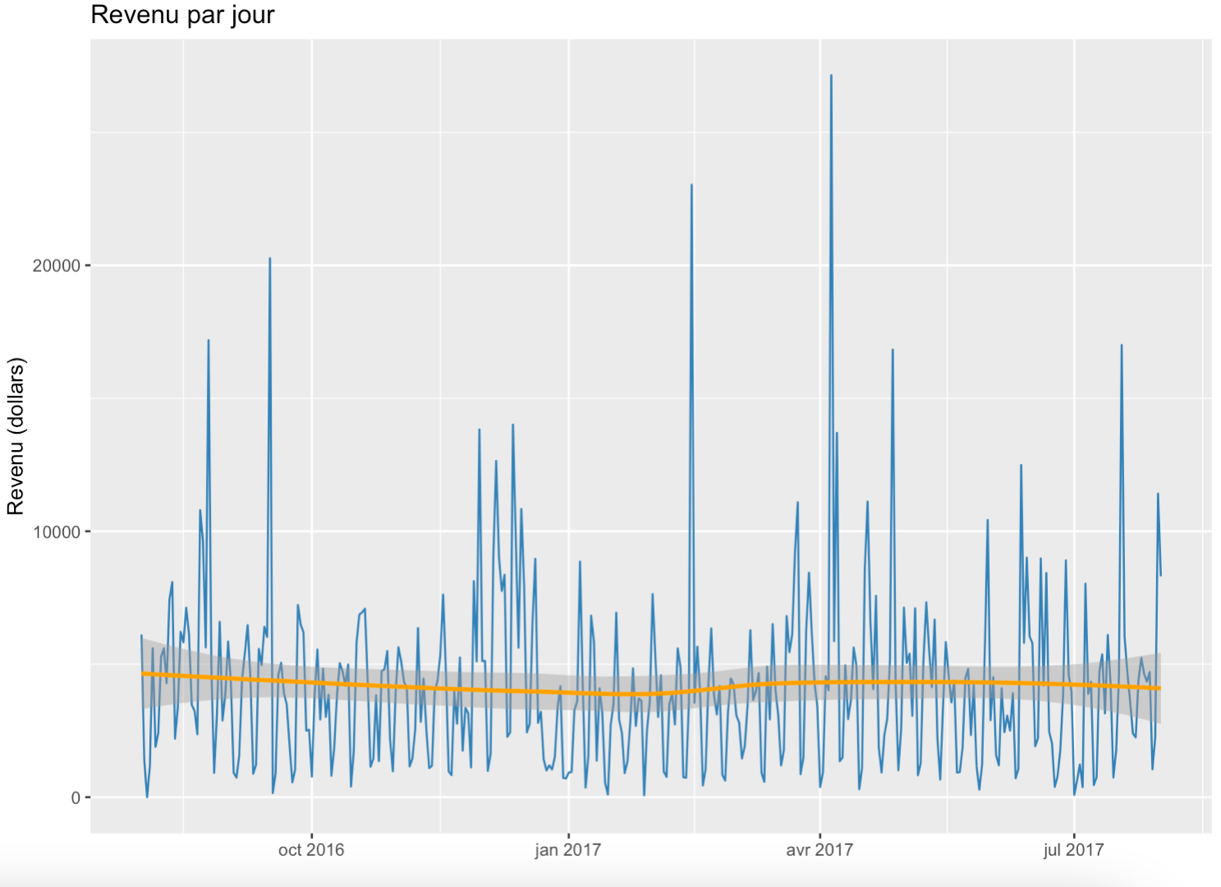
Ici, nous avons là les revenus générés en fonction des appareils utilisés. En comparant avec le graphique précédent, nous pouvons voir que la proportion “mobile” est plus faible, ce qui signifie que les utilisateurs de smartphones sont moins susceptibles de faire des achats que les autres une fois qu’ils sont sur le site.



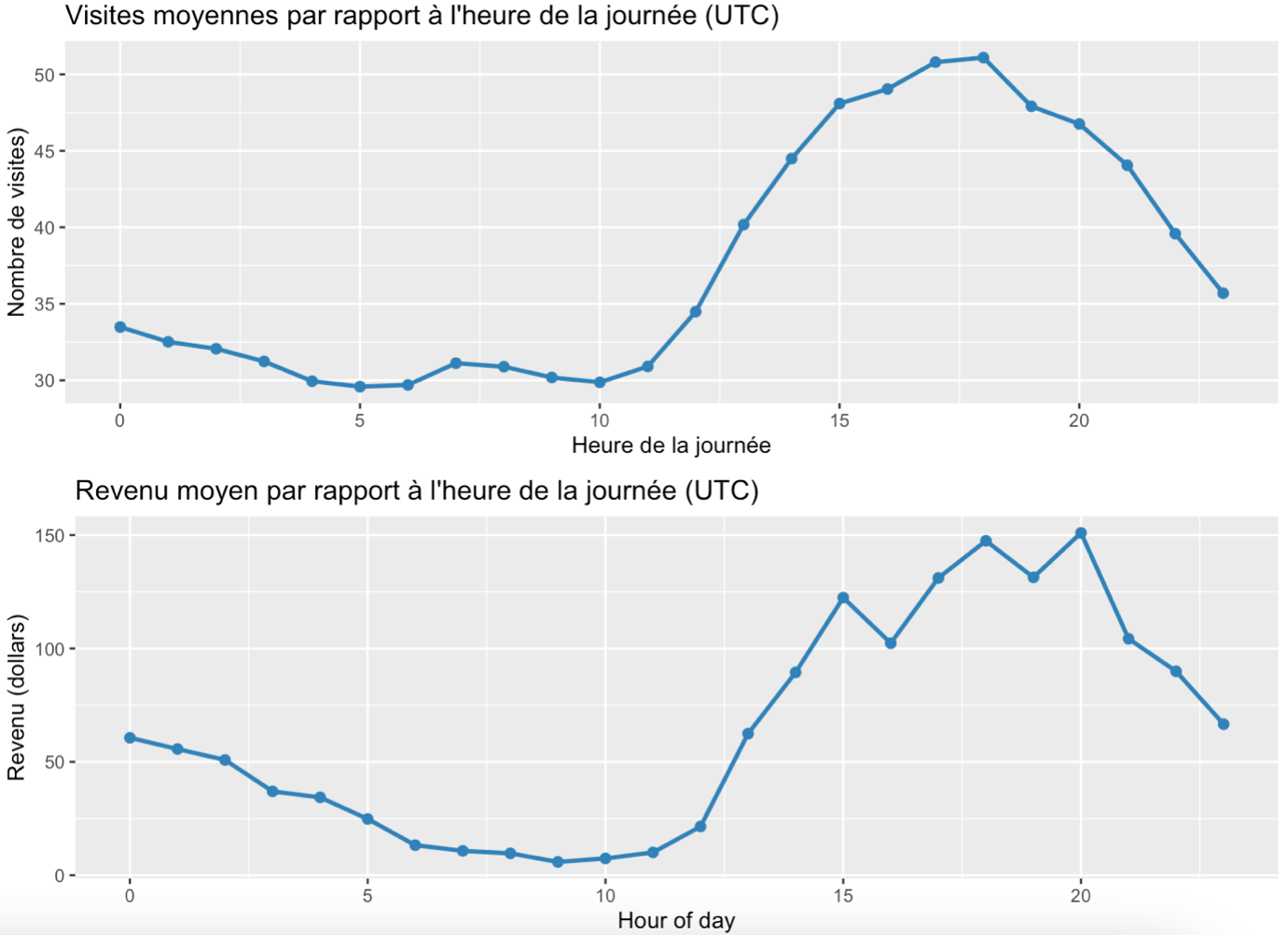
Nous avons donc choisi d’afficher le logarithme de la cible transaction revenue en séparant les utilisateurs de smartphones des autres utilisateurs. Nous pouvons remarquer que les premiers ont tendance à dépenser moins d’argent, mais la différence n’est pas extrêmement significative.



Ici, nous avons le nombre de visites par jour depuis Août 2016 jusqu’à Août 2017 (soit l’ensemble de notre jeu de données). Nous pouvons voir une nette augmentation des visites entre Octobre 2016 et Janvier 2017. Après être retombé, le nombre de visites est resté relativement constant. La ligne rouge (tracée avec la fonction geom\_smooth) sert à voir la tendance générale.



Ci-dessus, nous avons le graphique des revenus par jours d’Août 2016 à Août 2017. Il est intéressant de comparer ce graphique avec le précédent pour remarquer que le pic de fréquentation du site d’Octobre 2016 à Janvier 2017 n’a pas influé sur les revenus.



Ici, nous avons le nombre de visites et les revenus générés en fonction des heures de la journée. Comme nous aurions pu penser, ces deux informations sont fortement corrélées. Les clients font le plus d’achats entre 15 et 20 heures UTC, c’est à dire le matin pour les clients américains qui constituent la majorité des clients de ce jeu de données.

MODELES

Une fois que l’exploration et le pre-processing des données ont été réalisés, il nous reste à imaginer plusieurs modèles pour répondre à notre problématique. On pourra facilement comparer l’efficacité de chacun de ces modèles à l’aide de différentes méthodes d’évaluation.

1. Méthodes d’évaluation

Pour évaluer nos différents modèles, nous avons choisi 3 indices :

* RMSE, Root Mean Squared Error, comme demandé dans le sujet du projet
* RSS, Residual sum of squares
* R2, R-Squared

Pour rappel, il est demandé de prédire le total par utilisateur, ainsi pour chaque modèle nous avons suivi le même schéma dans notre code :

* Découpage train et test set
* Prédiction du revenu par session
* Évaluation de la prédiction
* Regroupement par *visitorId*
* Évaluation de la prédiction une fois regroupé

Pour la création du train set et du test set, nous avons découpé nos données en ¾ et ¼ .

Pour expliquer le regroupement : les prédictions sont réalisées sur chaque ligne de nos données, c’est-à-dire sur une visite sur la plateforme. Une fois la prédiction faite, il nous suffit de faire la somme des prédictions par *visitorId* puis d’évaluer ces résultats avec les données actuelles que nous regroupons également. Ainsi, nous évaluerons la performance de nos modèles sur les données groupées dans ce rapport.

Dans la suite, nous allons présenter nos modèles, nous avons fait une étude graduelle en allant du moins performant au plus performant.

1. Modèle naïf

Pour ce premier modèle, il suffit tout simplement de la moyenne. Ainsi, on obtient :

Une image contenant texte

Description générée automatiquementUne image contenant texte

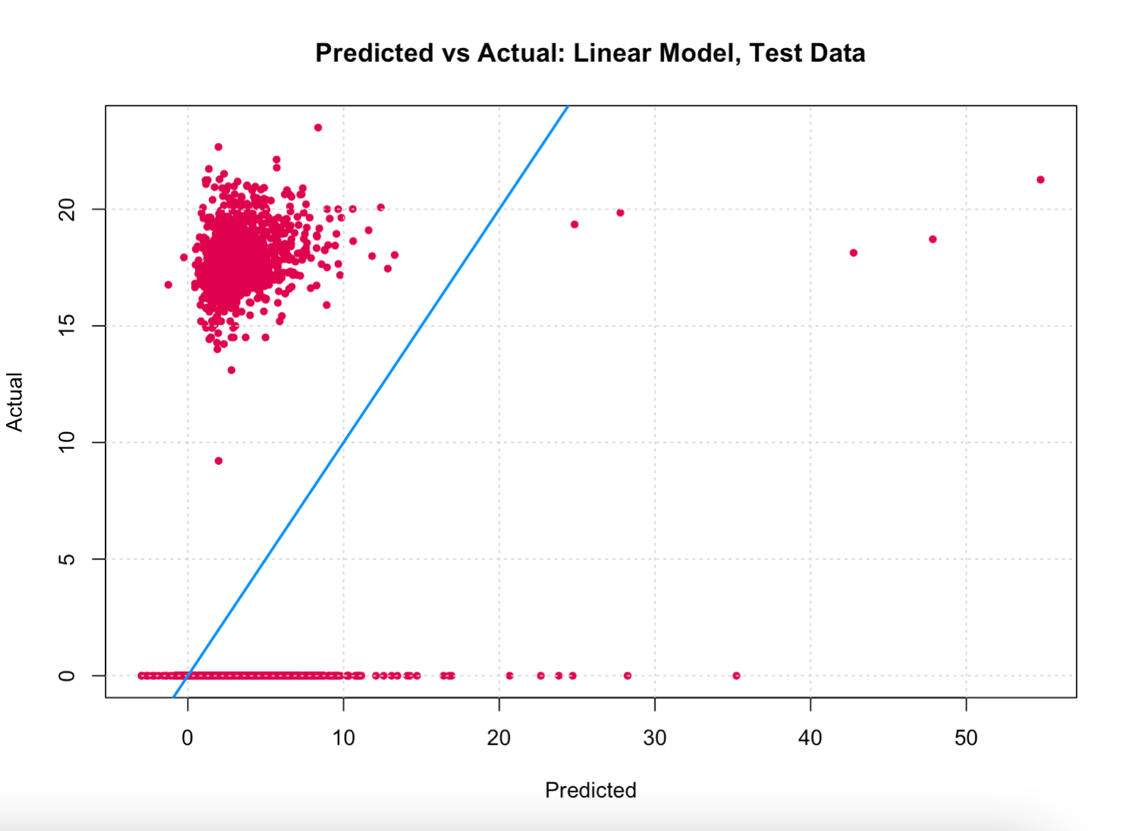
Description générée automatiquement

Le premier étant par session et le deuxième par visiteur. Cela nous permet de partir avec un ordre d’idée pour évaluer le reste de nos modèles.

1. Regression linéaire

Le modèle le plus basique que l’on utilise en prédiction est la régression linéaire. Nous l’avons donc appliqué ici en sélectionnant les colonnes ayant le plus d’impact sur la prédiction.

Les features que nous avons choisi sont : *channelGrouping, visitNumber, hits, pageviews, newVisits, continent, subcontinent, country, browser, operatingSystem, isMobile, deviceCategory, visitStartYear, visitStartMonth, visitStartDay, visitStartWeekDay, visitStartHour.*



Voici à quoi ressemble notre prediction. La régression linéaire ne parait pas être le plus adapté sur nos données, cela se confirme avec les indicateurs d’évaluation :

Une image contenant texte

Description générée automatiquement

Ce modèle reste néanmoins bien mieux qu’une simple moyenne.

Pour ce qui est du temps d’execution :

Une image contenant texte

Description générée automatiquement

1. Arbre simple

Nous avons décidé de faire un arbre simple avec **rplot** en prenant les mêmes features que précedemment.

Le plot de l’arbre n’est pas très parlant, néanmoins les résultats obtenus sont meilleurs que la régression linéaire.

Une image contenant texte

Description générée automatiquement

Pour ce qui est du temps d’exécution :

Une image contenant texte

Description générée automatiquement

1. Random Forest

Création d’un dataset Random Forest :

La mise en place d’un Random Forest nécessite un traitement supplémentaire sur le dataset que l’on va utiliser pour celui-ci.

En effet il nous est impossible de réaliser un random forest avec des données incluant des “Na”. Nous avons adopté la stratégie suivante :

* Elimination des colonnes contenant une très forte proportion de “Na” (+ de 90%)
* Le reste des “Na” sont eliminés grâce au paramètre “na.action=’na.exclude’ ” lors de la création de notre modèle Random Forest. Les lignes comportant au moins un “Na” ne seront pas pris en compte.

Nous sommes donc obligés de retirer certaines colonnes en entier (voir premier point au-dessus) car le “na.exclude” seul aurait retiré la quasi-totalité des lignes (certaines colonnes sont remplies de “Na”.

Recherche des paramètres optimaux :

Pour trouver les meilleurs paramètres de notre modèle (ntree,nodesize,mtry) nous avons réalisé un gridsearch en nous appuyant sur les librairies “caret” et “e1071”. Nous nous sommes inspirés du site [www.janbasktraining.com](http://www.janbasktraining.com).

La fonction trainControl() rentrée en paramètre dans le greedsearch, contrôle le dossier cross validation.

Cela nous a permis de trouver les valeurs qui minimisent le RMSE :

* mtry = 8-9
* nodesize = 25
* Ntree =250

Remarque : Etant donné le volume des données nous avons travaillé pour le random forest avec en moyenne 5000 lignes tirées au hasard. Selon ce tirage, les valeurs renvoyées par le gridsearch peuvent varier (en particulier ntree). Pour résoudre ce problème nous devrions (si nos machines et R nous le permettaient) faire tourner le gridsearch sur l’ensemble de nos données pour obtenir un réglage idéal.

Le modèle :

Voilà les résultats obtenus lorsqu’on lance un random forest grâce à la libraire “randomForest” sur R :

Une image contenant texte

Description générée automatiquement

Pour ce qui est du temps d’exécution :



Remarque : Grâce au paramètre “importance” de la libraire randomForest nous pouvons afficher les colonnes qui ont été le plus impactantes dans le résultat de notre modèle. Cela nous a permis de faire tourner un modèle random forest allégé (avec seulement une petite dizaine de variable). Cela nous a permis d’obtenir un modèle moins gourmand en ressource avec un temps de calcul plus faible tout en obtenant un résultat proche du random forest “complet”.

1. Gradient Boosting

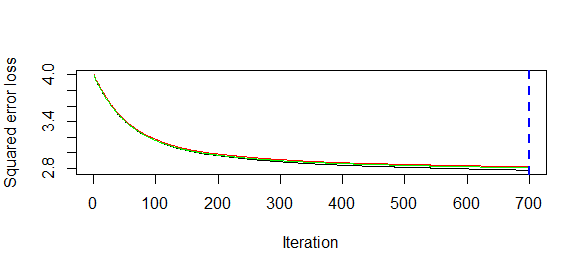
Nous avons remarqué que les compétiteurs de Kaggle ont obtenu les meilleurs scores de RMSE avec des modèles GBM, soit Gradient Boosting Machines.

Cet algorithme est un cas particulier du boosting, qui consiste agréger plusieurs modèles en un seul.

La construction des modèles se fait de manière séquentielle, c’est-à-dire qu’après avoir construit un premier modèle et l’avoir évalué, les individus suivants vont être pondérés en fonction de la performance de la prédiction précédente. Un poids plus important sera donné aux individus pour lesquels la valeur a été mal prédite dans le modèle précédent. Corriger les poids au fur et à mesure permet d’augmenter les performances.

Le gradient boosting utilise la méthode du gradient de la fonction de perte pour calculer les poids des individus. Cette méthode est similaire à la descente de gradient pour les réseaux de neurones.

Ci-dessous nous pouvons voir un graphe de l’errreur quadratique moyenne (RMSE) en fonction du nombre d’arbres. En bleu nous pouvons voir que le nombre d’itérations minimisant la RMSE se situe à 700 itérations (arbres). Cela est logique car à chaque itération l’algorithme GBM augmente ses performances.

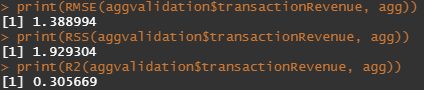


Pour éviter le sur-apprentissage et avoir un temps de calcul relativement court, après avoir testé plusieurs valeurs, nous avons choisi de limiter le nombre d’arbres à 700.

Un autre hyper paramètre important dans les GBM est le taux d’apprenstissage (shrinkage) qui correspond à la distance entre les points lors de la descent de gradient. Plus le learning rate est grand, moins le modèle a besoin d’arbres, mais plus il risque de perdre des informations durant la phase de minimisation de l’erreur. Après expérimentations, nous avons choisi un taux d’apprentissage de 0.1 comme il est recommandé de le faire pour les jeux de données conséquents.

Les autres hyper paramètres sont le nombre minimum d’observations par arbre (minobsinnode = 100), le nombre maximum de branches par arbre (interaction.depth = 3), la fraction du jeu de données d’apprentissage sélectionnée pour l’arbre suivant (bag.fraction = 0.5), la fraction des observations utilisée pour entrainer la GBM, sachant que le reste est utilisé pour avoir des individus extérieurs lors du calcul de la perte (train.fraction = 0.5) et finalement le nombre de validations croisées à réaliser (cv.folds = 10).

En utilisant ces paramètres nous arrivons à un RMSE de 1.39, un RSS de 1.93 et un R2 de 0.30, ce qui est meilleur que les deux autres modèles présentés précédemment.



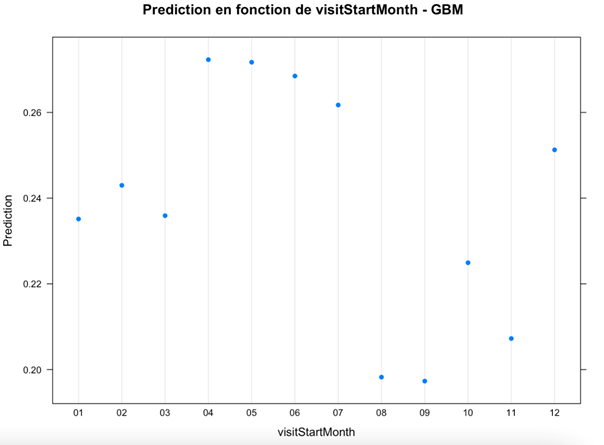
Pour ce qui est du temps d’exécution :

Une image contenant texte

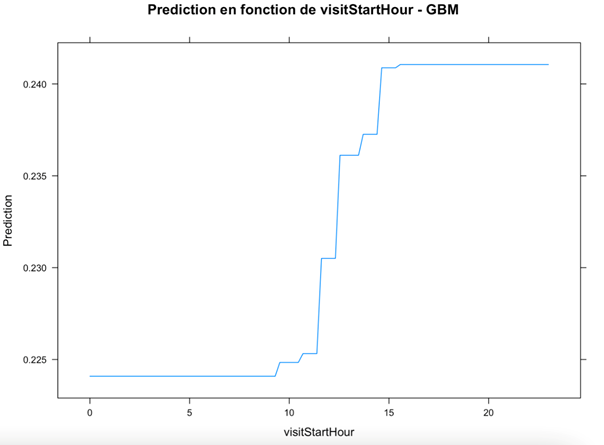
Description générée automatiquement

Grâce à la fonction plot(gbm.model), nous pouvons afficher l’effet des variables que nous avons choisi sur notre cible (les dépenses par utilisateur).

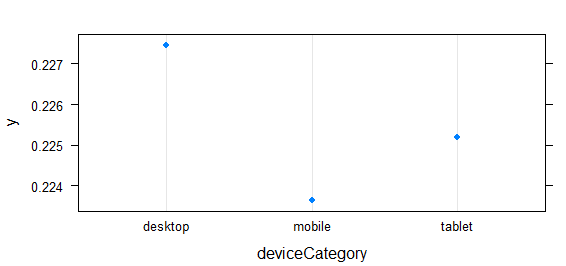
Voici une sélection de l’application de cette fonction sur les colonnes les plus pertinentes (en abscisse la colonne, en ordonnée sa corrélation avec la cible) :



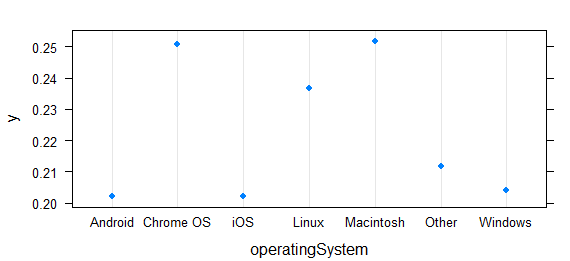
Ci-dessus nous voyons que les mois les plus impactants sur la vente sont les mois de Février à Juillet, avec une exception (cohérente) pour le mois de Décembre.



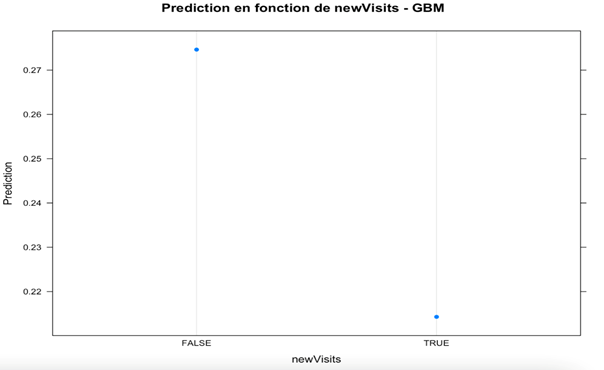
Ci-dessus nous voyons que les heures durant lesquelles les clients dépensent le plus sont entre 14 et 0 heures UTC. Etant donné que la grande majorité des clients vient des Etats Unis, cela correspond là-bas à entre 7 et 19 heures (fourchette grande car les fuseaux horaires sont entre UTC -8 et UTC -5 aux US).



Ci-dessus nous voyons que les appareils avec lesquels les clients font le plus de dépenses sont les ordinateurs, avec un peu d’achats effectués à partir de tablettes, et encore moins d’achats faits à partir de smartphones.



Ci-dessus nous voyons que les utilisateurs de Macintosh et de Chrome OS sont ceux qui dépensent le plus sur le site.



Ci-dessus nous avons l’impact de la variable newVisits sur la cible. Nous constations donc que les utilisateurs se rendant sur le site pour la première fois sont moins susceptibles d’acheter des articles.

CONCLUSION

Pour conclure, dans ce projet nous devions répondre à une compétition Kaggle sur des données Google Analytics et faire une prédiction des revenus.

Pour répondre à cela, nous avons exploré et traiter nos données, nous les avons visualisées pour en savoir plus et nous avons finalement essayé différents modèles et mesuré leurs performances.

Ainsi, nous avons présenté les modèles du moins performant au plus performant, on retiendra donc le GBM (Gradient Boosting Machines) comme étant le meilleur dans notre étude. Il permet d’obtenir un RMSE de 1,39 et un R2 de 0,30 pour un temps d’exécution de 12 min ce qui est nettement meilleur que les autres.

Si nous avions eu plus de temps sur ce projet, nous aurions aimé tester d’autres modèles mais également prendre le temps de pouvoir faire varier les paramètres et hyperparamètres. Certains modèles nécessitent un temps d’exécution très long, également dans la recherche des paramètres, ce qui peut monopoliser notre machine pendant des jours. Nous aurions aimé trouver un modèle ayant un RMSE < 1 ou encore un R2 > 0,5

On notera que, dans les futurs projets, il est intéressant de faire une démarche du modèle le plus naïf au plus « complexe » afin de maîtriser les données et savoir ce que l’on veut prédire et l’ordre de grandeur des valeurs de RMSE, RSS et R2 que l’on doit trouver.

En ouverture, nous aurions aimé aussi prédire d’autres choses qui peuvent être intéressantes dans la gestion d’un e-shop afin de pouvoir guider au mieux une campagne marketing grâce à nos modèles de Machine Learning.