Projet de traitement de données massives

Quatrième Partie : Rendu Final au 18/04/2018
Sujet : Prévision de la fréquentation d'une piscine

Equipe 2
Chloé PELLETIER
Maxime SURMONT

INTRODUCTION

Travailler sur la fréquentation quotidienne des piscines est une problématique intéressante car beaucoup de paramètres peuvent entrer en jeu comme la saison, le jour de la semaine ou encore la météo. Et pour qu'une piscine puisse être gérée au mieux, connaitre à l'avance combien de personnes fréquenteront les lieux permet de pouvoir prévoir le nombre de personnel à employer (maître nageur, réceptionniste, ...), le nombre de bassins à ouvrir ou encore savoir s'il faut renforcer la sécurité pour un jour particulier. Cela peut aussi être intéressant, pour détecter si la piscine a des jours de creux pour décaler le jour de fermeture hebdomadaire et ainsi devenir plus rentable.

Notre projet porte plus précisément sur la fréquentation de la piscine publique de Nettebad à Osnabrück en Allemagne, sur la période s'étalant du 20/03/2005 au 31/12/2012. Pour ce faire nous avons à notre disposition plusieurs informations à propos de cette piscine et des conditions météorologiques de la région.
Il s'agit donc d'un projet de traitement de données numériques permettant de prédire la fréquentation de la piscine sur un an. Le projet Kaggle relié à notre étude se trouve à cette adresse : https://www.kaggle.com/c/swimming-pool-visitor-forecasting.
Lien git vers notre projet : https://github.com/chloepelletier/kaggle_project

I. DESCRIPTION ET PRETRAITEMENT DES DONNEES

Les données que nous avions à notre disposition pour ce projet Kaggle étaient sous forme de fichiers csv. Nous avions trois fichiers différents :

- un fichier contenant des données propres à la piscine avec :
  - **visitors_pool_total** (Numérique): nombre de visiteurs quotidien
  - **sportbad_closed** (Binaire): précise si le bassin sportif est ouvert
  - **freizeitbad_closed** (Binaire): précise si le bassin de loisir est ouvert
  - **sauna_closed** (Binaire): précise si le sauna est ouvert
  - **kursbecken_closed** (Binaire): précise si le bassin olympique est ouvert
  - **event** (Binaire): précise s'il y a un événement
  - **price_adult_90min** (Numérique): prix adulte pour un accès de moins de 90 mins
  - **price_adult_max** (Numérique): prix adulte sans limite de temps
  - **price_reduced_90min** (Numérique): prix réduit pour un accès de moins de 90 mins
  - **price_reduced_max** (Numérique): prix réduit sans limite de temps
  - **sloop_dummy** (Binaire): précise si le toboggan est ouvert
  - **sloop_days_since_opening** (Numérique): jours depuis l'ouverture du toboggan
  - **school_holiday** (Ordinal): indique les vacances scolaires dans les environs
  - **bank_holiday** (Ordinal): indique les jours fériés dans les environs de la piscine

- deux fichiers contenant des données météo à deux positions différentes proches de la piscine avec :
  - **air_humidity** (Numérique): pourcentage humidité
  - **air_pressure** (Numérique): pression atmosphérique
  - **air_temperature_daily_max** (Numérique): température maximale
  - **air_temperature_daily_mean** (Numérique): température moyenne
  - **air_temperature_daily_min** (Numérique): température minimale
  - **precipitation** (Numérique) : quantité de pluie par jour en mm
  - **snow_height** (Numérique): hauteur de neige en cm
  - **sunshine_hours** (Numérique): nombre d'heures de soleil par jour
  - **global_solar_radiation** (Numérique): radiation solaire en W/m²
  - **wind_speed_max** (Numérique) : vent maximum (m/s)
  - **wind_speed_avg** (Numérique) : vent moyen (m/s)
  - **wind_direction_category** (Ordinal): direction du vent

Dans chacun de ces fichiers, l'horodatage de la donnée est présent grâce à l'attribut '**date**' (String).

Nous avons analysé chacun de ces attributs afin d'améliorer la qualité de nos données. Pour cela, nous avons fait plusieurs prétraitements sur ces données "brutes". Nous avons essayé de nettoyer au maximum la donnée mais aussi de la rendre plus facilement exploitable en faisant de la réduction de numérosité (création de catégories pour certains attributs par exemple catégories de prix au lieu d'une valeur numérique) et de dimensionnalité (réduction de certains paramètres traduisant la même évolution par exemple l'évolution de la vitesse moyenne et maximale du vent). Nous avons également fait de l'intégration des données en allégeant le jeu de données grâce à l'élimination d'attributs dupliqués (notamment avec les données des deux météos disponibles) ou inutiles (par exemple la direction du vent).

A partir de ce premier prétraitement, nous avons cherché à éliminer les attributs n'apportant que peu d'informations pour la détermination du nombre de visiteurs. Pour cela, nous avons utilisé le

test Kruskal-Wallis[1], permettant d'affirmer ou non l'hypothèse suivante: les variables testées offrent des populations similaires. A partir de ce test, nous avons les duos d'attributs trop corrélés entre eux par exemple le déroulement d'un événement est inclus dans la date car ceux-ci se déroulent en fin de semaine.
Pour les attributs plus tendancieux (température, vent et précipitation de pluie) , nous avons étudié leur corrélation (de Pearson[2]) avec le nombre de visiteurs total. Cela nous a permis de mettre en avant des cas particuliers sur la vitesse du vent ou encore des cas extrêmes de températures qui nous seront utiles dans la suite de notre étude.

La sélection finale de nos attributs est la suivante :

- **day** (Numérique): numéro du jour de la semaine (0 à 3)
- **month** (Numérique): numéro du mois de l'année (1 à 12)
- **visitors_pool_total** (Numérique): nombre de visiteurs quotidien
- **sportbad_closed** (Binaire): précise si le bassin sportif est ouvert (0) / fermé (1)
- **freizeitbad_closed** (Binaire): précise si le bassin de loisir est ouvert (0) / fermé (1)
- **kursbecken_closed** (Binaire): précise si le bassin olympique est ouvert (0) / fermé (1)
- **school_holiday** (Nominal): catégorie de vacances scolaires dans les environs de la piscine
- **bank_holiday** (Binaire): précise s'il s'agit d'un jour férié (1) ou non (0)
- **temperature** (Numérique): température moyenne (°C)
- **snow_height** (Binaire): indique s'il y a de la neige (1) ou non (0)
- **sunshine_radition** (Ordinal): catégorie de radiation solaire en W/m²

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kruskal.html
[2] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.corr.html

II. DÉMARCHES VERS LA SOLUTION

A. TESTS

Nous avons découpé notre jeu de données en trois sous-ensembles :

- un ensemble d'entraînement : ensemble de données pour créer les différents modèles
- un ensemble de validation : ensemble de données nous permettant de choisir entre nos modèles et d'ajuster les différents paramètres des algorithmes si besoin (en faisant attention au risque du sur-apprentissage/sur-ajustage).
- un ensemble de tests : ensemble de données pour tester et évaluer la performance de notre algorithme final

Pour comparer nos différents modèles et les évaluer, nous avons utilisé le critère de performance RMSE (qui est également celui utilisé pour le calcul du score Kaggle). Il s'agit de la racine carrée de l'écart quadratique moyen. Plus cette valeur est faible, moins il y a d'écart entre les valeurs prédites et réelles.
Nous avons également comparé les valeurs de ce critère pour les différents modèles utilisés sur les différents ensembles de données, pour vérifier qu'il n'y a pas eu de sur-apprentissage.

B. SOLUTIONS ENVISAGÉES

Nous avons testé différents types d'algorithmes à partir de notre sélection d'attributs et le prétraitement effectué :

- Nous avons testé 2 algorithmes de partitionnement : le **K-Mean[3] et le K-Nearest Neighboor[4]**. L'utilisation de ces algorithmes n'a pas été concluante car ils ont pour but de former des groupes d'éléments avec des similarités maximales et en étudiant de plus près notre jeu de données nous nous sommes rendu compte que nos paramètres n'étaient pas assez différenciants (beaucoup d'attributs binaires) pour avoir des groupes de données similaires qui se forment.
  Le test de ces algorithmes, nous a tout de même permis de trouver que les algorithmes supervisés sont plus adaptés à notre jeu de données.

- Nous avons appliqué des algorithmes faisant appel à des forêts d'arbres décisionnels à notre jeu de données **Random Forest Classifier**[5] et **Random Forest Regression**[6]. Ces algorithmes sont des méthodes d'apprentissage automatique, créant des arbres de décision et apprenant à partir de ces derniers. Les scores RMSE obtenus par le jeu de données de validation sont meilleurs que pour les algorithmes de partitionnement (350 contre 500).
  De ces deux algorithmes, Random Forest Regression est le plus performant car il permet d'obtenir des valeurs numériques (ce qui est plus adapté à notre jeu de données) et non des classes de données comme fait Random Forest Classifier.

- Une autre technique d'apprentissage automatique que nous avons pu tester a été les réseaux de neurones. Nous avons donc appliqué à notre jeu de données aux modèles : **MLP**

[3] http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
[4] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[5] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[6] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Classifier[7] (Multi Layer Perceptron) ainsi que **MLP Regressor**[8]. Ce type d'algorithmes s'est également montré performant, avec des scores RMSE légèrement supérieur à ceux des Random Forest (aux environs de 370). Cependant les réseaux de neurones demandent bien plus de ressources CPU que les Random Forest et sont plus longs à exécuter.
Nous avons pu confirmer que les algorithmes se basant sur la régression semblent être les plus performants pour notre jeu de données.

- Nous avons aussi essayé d'autres modèles (par exemple basé sur les SVM (Machine à Vecteur de Support) appliqué à la régression : l'algorithme **SVR**[9]) mais nos tests n'ont pas été concluants.

C. ALGORITHME SÉLECTIONNÉ

Notre algorithme final va se baser sur l'utilisation de deux techniques d'apprentissage automatique différentes : un **Random Forest Regression** ainsi qu'un **MLP Regressor.**

Tout d'abord, au lancement de l'algorithme, celui-ci va effectuer les étapes de prétraitement de données pour obtenir l'ensemble des données regroupé sous un seul tableau avec les attributs définis dans la partie I. Un jeu de données d'entraînement, de validation et de test est alors créé.

Ensuite, l'algorithme va utiliser les deux méthodes d'apprentissage automatique :

- Il commence avec le **Random Forest Regression**, utilisant l'ensemble des données d'entraînement. Cependant une sélection plus poussée des attributs a été effectuée par tests successifs pour obtenir le meilleur score possible.
  Pour rappel, le Random Forest ou algorithme des forêts d'arbres décisionnels se base sur un apprentissage basé sur de multiples arbres de décision entraînés sur des sous-ensembles de données différents. La différence entre un arbre de classification standard et celui-ci se basant sur la régression est que :

  > "*Un arbre de classification standard est construit du haut vers le bas à partir d'un nœud racine et implique le partitionnement des données en sous-ensembles homogènes.*
  > *Dans un arbre de régression, un modèle de régression va être utilisé pour la variable cible en utilisant chacune des variables indépendantes. Ensuite , pour chaque variable indépendante, les données sont réparties en plusieurs points de division. On calcule la somme de l'erreur quadratique (SSE) au niveau de chaque point de séparation entre la valeur prédite et les valeurs réelles. La variable résultant en SSE minimum est sélectionné pour le noeud. Ensuite , ce processus est reproduit et poursuivi jusqu'à couvrir l'ensemble des données.*"[10]

  Toujours par tests successifs, nous avons sélectionné les paramètres du modèle Random Forest nous offrant le plus de performances sur notre jeu de données de validation, risquant le sur-apprentissage.

---

[7] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
[8] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
[9] http://scikit-learn.org/stable/modules/svm.html
[10] https://www.quora.com/How-does-random-forest-work-for-regression-1

- Ensuite, l'algorithme utilise un réseau de neurones se basant sur la régression (**MLP Regressor**). A nouveau sur l'ensemble des données d'entraînement, avec une sélection d'attributs différents de celle pour Random Forest Regression.
  Le perceptron multicouche est un type de réseaux neuronaux formels organisés en plusieurs couches (au moins 3). L'information circulera de la couche d'entrée à la couche de sortie. Ici il va permettre de résoudre un problème d'analyse de régression pour prédire une valeur. Le fonctionnement général suit le schéma suivant[11]:

  1. Présentation d'un motif d'entraînement au réseau.
  2. Comparaison de la sortie du réseau avec la sortie ciblée.
  3. Calcul de l'erreur en sortie de chacun des neurones du réseau.
  4. Calcul, pour chacun des neurones, de la valeur de sortie qui aurait été correcte.
  5. Définition de l'augmentation ou de la diminution nécessaire pour obtenir cette valeur (erreur locale).
  6. Ajustement du poids de chaque connexion vers l'erreur locale la plus faible.
  7. Attribution d'un blâme à tous les neurones précédents.
  8. Recommencer à partir de l'étape 4, sur les neurones précédents en utilisant le blâme comme erreur.

  Comme précédemment, nous avons sélectionné les paramètres de ce modèle nous offrant le plus de performances. Nous avons cependant fait ici un compromis entre performance et temps d'exécution. En effet plus on augmente le nombre de neurones par couches, plus l'algorithme demande de ressources CPU et plus il est long à s'exécuter.

Notre approche a donc été d'entraîner, voir sur-entrainer chaque modèle le plus possible (par la sur-sélection des attributs, la personnalisation des variables de chaque algorithme, etc.), en utilisant les modèles qui se sont avéré être les plus performants. Puis de nuancer le sur-apprentissage de ces algorithmes en les combinants. En effet, en observant les erreurs de chaque algorithme, nous nous sommes rendu compte que les écarts avec les valeurs réelles étaient différents pour chacun d'eux. Et, en utilisant les deux algorithmes, nous réduisons le taux d'erreur globale. Nous avons donc atténué les erreurs d'un algorithme par les erreurs moins importantes de l'autre et inversement. Il s'agit là de la dernière étape de notre algorithme.
Nous avions dans un premier temps pris la moyenne des deux algorithmes pour chaque jour. Mais, au final la méthode qui s'est avéré la plus performante fut de prendre la valeur minimale prédite entre les deux algorithmes, car les erreurs de prédiction étaient souvent des prédictions prévues supérieures à la réalité.

D. PROBLÈMES RENCONTRÉS

Lors de notre développement, nous avons rencontré plusieurs difficultés :

- Tout d'abord, il est difficile de savoir par où débuter lorsque l'on débute un projet d'analyse de données et qu'il s'agit d'un de nos premiers. Et cette difficulté se retrouve aux différentes étapes de développement : Avant même de commencer à coder, voir avant même de

[11] https://fr.wikipedia.org/wiki/Perceptron_multicouche

regarder les données, il faut faire de nombreuses recherches sur les technologies envisageables ainsi que sur les différentes librairies disponibles. Nous avons choisi de développer en python car nous avions déjà fait du R et que nous voulions découvrir le deuxième grand langage utilisé en étude de données. Nous avons ensuite utilisé les bibliothèques les plus reconnues et conseillées. Ensuite pour le prétraitement il faut savoir passer de la théorie à la pratique et sans méthodologie précise on peut s'y perdre. Pour cela nous avons essayé de nous appuyer au mieux sur le cours et nous avons défini de nombreuses courtes étapes facilement réalisables. Enfin la recherche d'algorithmes performants pour nos données est compliquée car on ne connaît pas forcément tout ce qui existe et que l'amélioration des résultats est toujours possible.

- La plus grande difficulté que nous avons rencontrée dans lorsque nous codions fut la fusion des différents tableaux d'informations, notamment pour ceux sur la météo. Nous avons dû faire des choix dans la manière dont nous transformions les données et, même si nous avons essayé d'être les plus cohérents possible, nous n'avions pas la possibilité d'être certains que nos changements gardaient les données intègres.

III. NOS RÉSULTATS

A. PREMIERS RÉSULTATS

Nos premiers résultats avaient des scores très élevé de RMSE, aux alentours de 600 - 700. Cela venait notamment d'un trop grand nombre de variables en entrée. C'est en testant que l'on a pu se rendre compte qu'il existait une sélection optimale de variables qui était assez similaire entre les différents algorithmes testés.
Le premier algorithme que l'on a soumis à KAGGLE était un algorithme utilisant uniquement le Random Forest Regression. Il a obtenu un score de 360 nous plaçant alors vers la quinzième place du classement.

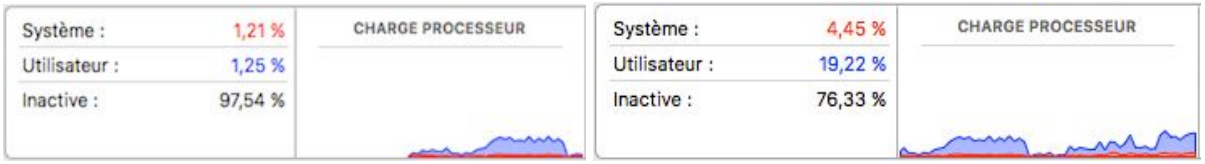
B. NOTRE RÉSULTAT FINAL

Après avoir soumis le tableau des prédictions de notre algorithme final à KAGGLE, nous avons eu un score RMSE de 297 (le private score est le score calculé sur l'ensemble des données) qui vaut celui du groupe arrivé 9ème à la compétition. En moyenne, sur les données de validation, l'erreur quotidienne est d'environ 178 personnes.

Submission and Description	Private Score	Public Score	Use for Final Score
submission.csv 7 days ago by maximeSurmontKGL submission	297.32155	316.25429	☐

score kaggle de notre algorithme final

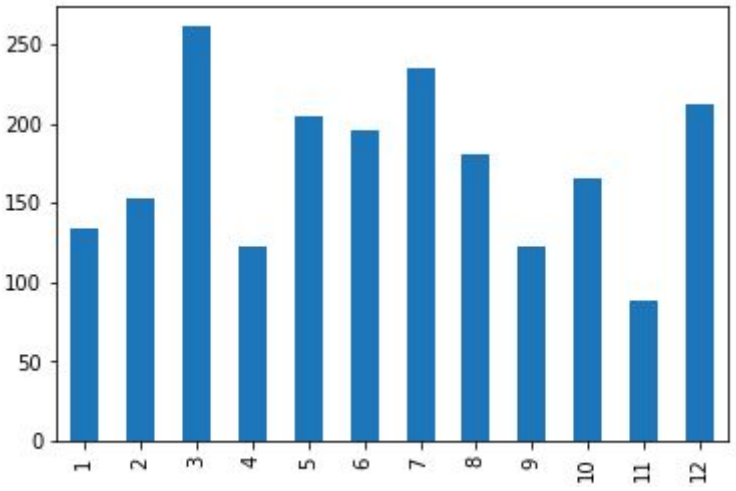
L'algorithme a besoin de 1 min 22 sec pour s'exécuter dont 1 min 16 sec uniquement pour performer le modèle **MLP Regressor**. En terme d'activité CPU, en utilisant nos propres ordinateurs (intel core i7, 2.6 GHz), on obtient le benchmark/étalon de comparaison suivant :



Captures d'écran de l'activité CPU avant et pendant l'activité de l'algorithme

On peut donc approximer l'augmentation de l'activité CPU à 18% (19.22 - 1.25) de ses capacités.

Une limitation de notre algorithme ou axe d'amélioration est l'intervention des vacances scolaires. En effet, comme on peut le voir dans le schéma ci-dessous, on se rend compte ques les mois de mars, décembre et ceux des vacances d'été sont les mois ou l'écart prédiction/réalité est le plus important. Après vérification, c'est bien le paramètre des vacances scolaires qui rend la prédiction plus difficile. Nous avons essayé de mettre en place des modèles pour gérer en cas particulier les vacances mais sans obtenir de résultats vraiment plus performants.



histogramme de l'écart prédiction/réalité en fonction des mois

Les conditions optimales, là où l'écart avec la réalité est le plus faible sont les jours où "il se passe le moins de choses possible". En effet, les meilleurs scores apparaissent aux journées les plus communes, en semaine, sans bassin fermé, hors vacances scolaires ou jour fériés, sans évènement, le temps importe peu à condition de n'avoir aucun extrême.

IV. RÉTROSPECTIVES

A. RETOUR SUR NOS CHOIX

a. Prétraitement des données

Ayant beaucoup de données dans notre jeu de données initial que ce soit au niveau de la météo ou des données propres à la piscine, nous avons consacré beaucoup de temps à l'étude et au prétraitement des données. Ce qui selon nous a été bénéfique pour la suite du projet, car avec un prétraitement efficace au début le reste de l'analyse et du traitement est plus facile à gérer.
Néanmoins, nous nous sommes attardés sur certaines variables qui se sont révélées peu intéressantes lors du traitement des données. Après réflexion, nous aurions dû faire évoluer le prétraitement tout au long du projet c'est à dire faire les prétraitements les plus affinés uniquement sur les données réellement intéressantes.
Nous voulions aussi enrichir nos données avec par exemple d'autres données sur la piscine mais nous n'avons pas pu obtenir les informations désirées.

b. Traitement des données

Pour trouver le modèle le plus efficace, nous avons commencé par tester plusieurs types d'algorithmes de traitement. Ceci était un bon choix car même si certains algorithmes donnaient des résultats complètement erronés, cela permettait d'obtenir des informations sur notre jeu de données et nous amenait à nous poser des questions sur celui-ci, nous permettant de découvrir des liens entre celles-ci.
Nous avons choisi un score RMSE pour évaluer notre algorithme. Mais ce score augmente autant pour une prédiction trop grande que pour une trop petite. Or un score trop petit est plus nocif pour notre étude car si un nombre de visiteurs trop bas est prévu pour une journée, les gérants de la piscine pourraient faire certains choix pénalisant le bien-être des clients (par exemple : choisir de fermer certains bassins, avoir un nombre d'employés insuffisant...). Mais, le score RMSE est celui utilisé pour l'évaluation du projet dans Kaggle, c'est donc le choix d'évaluation le plus logique.

c. Gestion de projet

Nous avons décidé d'utiliser GitHub comme gestionnaire de version et d'utiliser le système des "issues" proposé sur la plateforme pour gérer au mieux notre avancée, en gardant des objectifs à court terme relativement simples.
Nous avons également réfléchi rapidement à mettre en place une structure dans notre répertoire de projet permettant de bien séparer les fichiers de données (initiaux, prétraités et traités), les scripts portant sur le prétraitement, ceux portant sur des modèles, etc..

B. ET SI L'ON DEVAIT TOUT REFAIRE

Si nous devions refaire ce projet depuis le début, on débuterait de la même manière avec travail conséquent sur la compréhension des données mais aussi sur un premier prétraitement avec juste un premier nettoyage de la donnée. Ensuite, nous commencerions le traitement et ferions évoluer le prétraitement des données au fur et à mesure avec les avancées sur le traitement.
Au niveau de la procédure de test, nous aurions également aimé utilisé deux façons d'évaluer notre méthode :

- le RMSE, comme nous l'avons fait car c'est en lien avec la compétition kaggle
- une autre technique s'adaptant plus à nos données accordant moins d'importance aux majorations du nombre de visiteurs qu'aux minorations du nombre de visiteurs.

CONCLUSION

Au travers de ce projet nous avons pu prendre conscience de la difficulté d'un projet d'analyse de données. Il n'a pas été aisé de comprendre et de traiter notre jeu de données. En effet, dans notre cas nous avions une multitude d'attributs différents à analyser pour comprendre nos données. Il a donc fallu appliquer le cours et compléter nos connaissances (apprendre le langage python que nous ne connaissions pas, apprendre à utiliser des librairies telles que scikit-learn ou pandas) pour produire un projet avec un cheminement intéressant et un résultat performant.

En conclusion, pour un premier projet d'analyse de données, nos résultats sont plutôt intéressants et nous permettent d'avoir un bon aperçu du nombre de visiteurs par jour à la piscine de Nettebad avec une différence d'environ 18% par rapport à la réalité. Nous sommes donc satisfaits de ces résultats même si nous savons que certains points pourraient être améliorés.

L'intégralité de notre code se trouve au lien GIT suivant :
https://github.com/chloepelletier/kaggle_project
Le README présent sur GIT permet de comprendre la segmentation de notre projet.