

# Python for data analysis

## Seoul Rented bike prediction

07/01/2021

Kenny GUILLOUCHE, Clément GENOT

## Seoul Bike Sharing Demand Data Set

Download: [Data Folder](#) [Data Set Description](#)

**Abstract:** The dataset contains count of public bikes rented at each hour in Seoul Bike haring System with the corresponding Weather data

Data Set Characteristics:	Multivariate	Number of Instances:	8760	Area:	Computer
Attribute Characteristics:	Integer, Real	Number of Attributes:	14	Date Donated	2020-03-01
Associated Tasks:	Regression	Missing Values?	N/A	Number of Web Hits:	13861



L'étude a pour objectif de prédire le nombre de location de vélo à Seoul pour une heure donnée. Ce dataset nous a été fourni par des chercheurs. (<https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand>)

Ceux-ci ont récupéré les données du gouvernement de Seoul sur les locations de vélos auquel ils ont ajouté les informations météorologiques et les informations sur les vacances scolaires. Toutes ces informations ont été obtenu via du datamining.

Le but des chercheurs dans leur article « Using data mining techniques for bike sharing demand prediction in metropolitan city » était dans un premier de temps d'aider les villes à fournir aux citoyens un approvisionnement en vélos de locations adaptés aux nombres de clients.

Dans un seconde temps, il était de tester différentes méthodes de datamining et également de déterminer différentes approches de machine learning pour faire une sorte d'état de l'art des méthodes de résolution d'un problème de régression.

## Attribute Information:

Date : year-month-day  
Rented Bike count - Count of bikes rented at each hour  
Hour - Hour of the day  
Temperature-Temperature in Celsius  
Humidity - %  
Windspeed - m/s  
Visibility - 10m  
Dew point temperature - Celsius  
Solar radiation - MJ/m2  
Rainfall - mm  
Snowfall - cm  
Seasons - Winter, Spring, Summer, Autumn  
Holiday - Holiday/No holiday  
Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours)  
Day  
Month  
Year  
DayOfWeek  
isWeekEnd

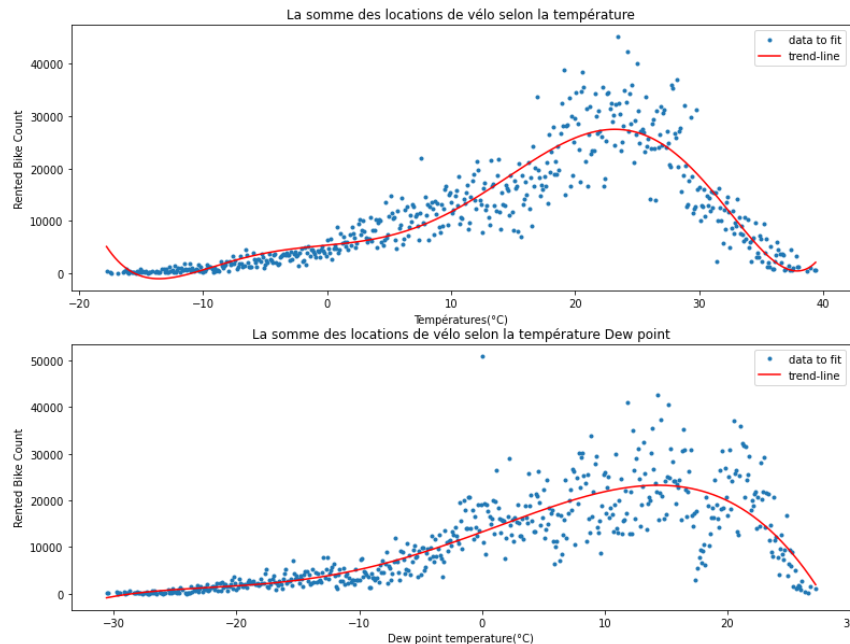
Le jeu de données sur lequel va se baser notre étude comprend donc différentes informations. On a d'abord le nombre de vélos loués chaque heure à Seoul, on est donc dans la résolution d'un problème de régression de façon supervisée. On a ensuite des informations temporels avec la date, l'heure, vacances scolaires et les saisons. Puis des informations météorologiques avec la température, l'humidité, la vitesse du vent, la visibilité (brouillard), la température à l'aube, le niveau de radiation solaire, le niveau de pluie tombé et le niveau de neige tombé. Et pour terminer, nous avons une information liée à la disponibilité de la station de vélo.

Se basant là-dessus, nous avons commencé par décomposer la date en jour, mois, année afin d'étudier l'impact des différents aspects de la date sur le nombre de vélos loués. Le jour nous l'avons découpé en deux formats : le jour du mois (1-31) et le jour de la semaine (0-6, 0 étant lundi et 6 étant dimanche).

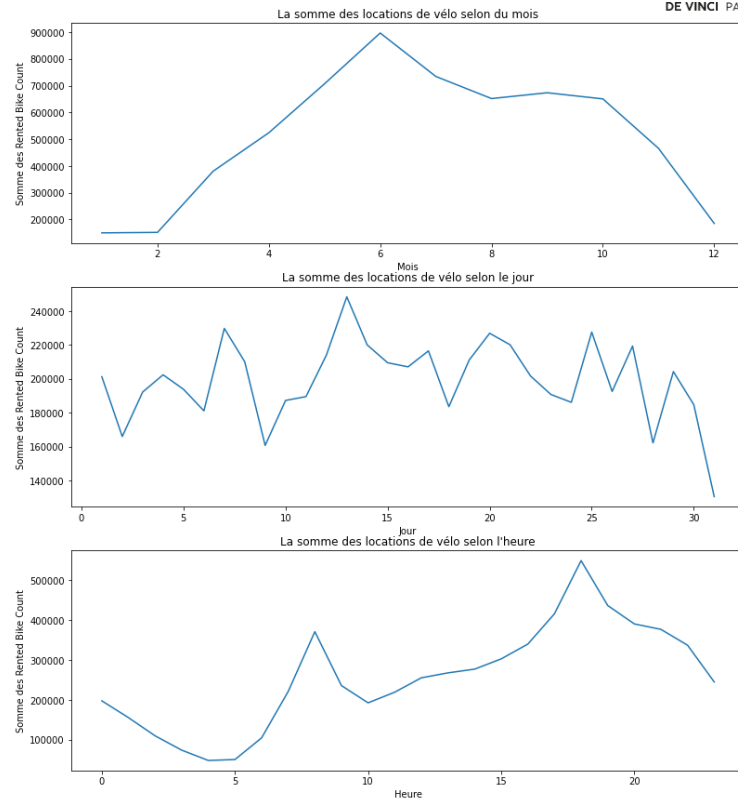
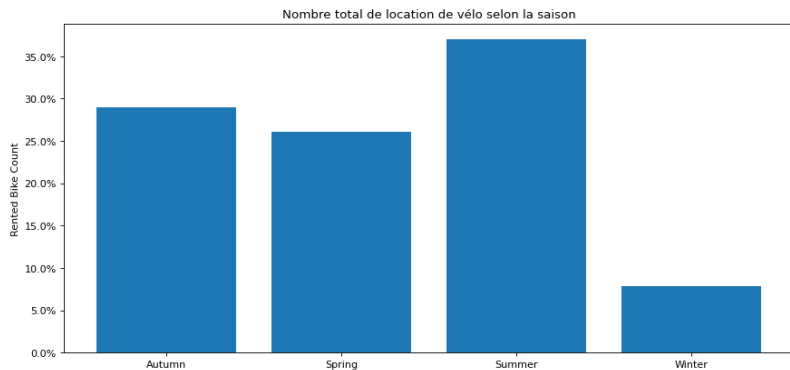
De plus, nous pensons que le week-end doit avoir un impact sur la location de vélo, nous avons donc ajouté une colonne isWeekEnd.

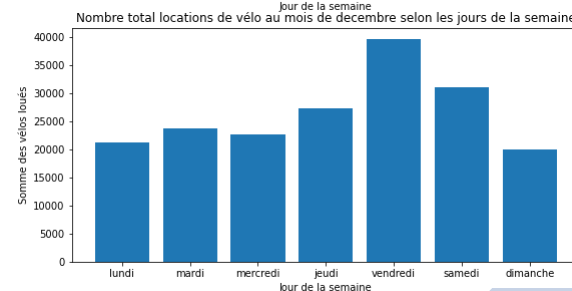
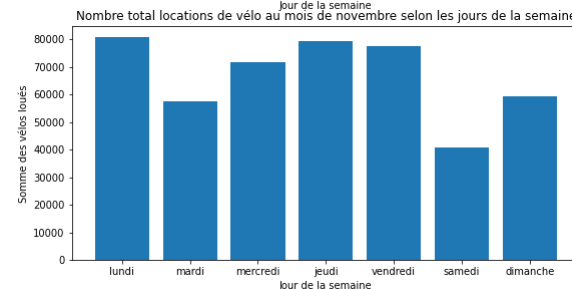
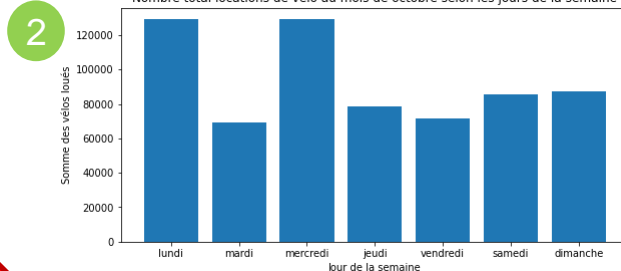
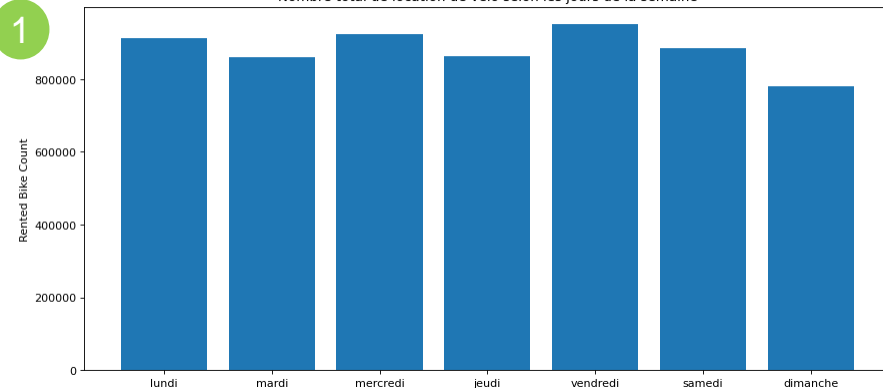
Maintenant que nous connaissons notre jeu de données, nous allons analyser plus en détails les colonnes afin de trouver les variables intéressantes pour prédire.

Pour commencer nous avons regardé l'impact de la température sur le nombre de vélo loué, on a donc tracé ceux-ci sous forme de nuage de point et tracer la courbe de tendance associé. On peut observer que le nombre de vélo augmente drastiquement lorsque les températures sont plus clémentes. Cela ne nous surprend pas, en effet les températures douces amènent généralement les gens à passer plus de temps dehors et à privilégier le vélo à d'autres formes de transports urbains.



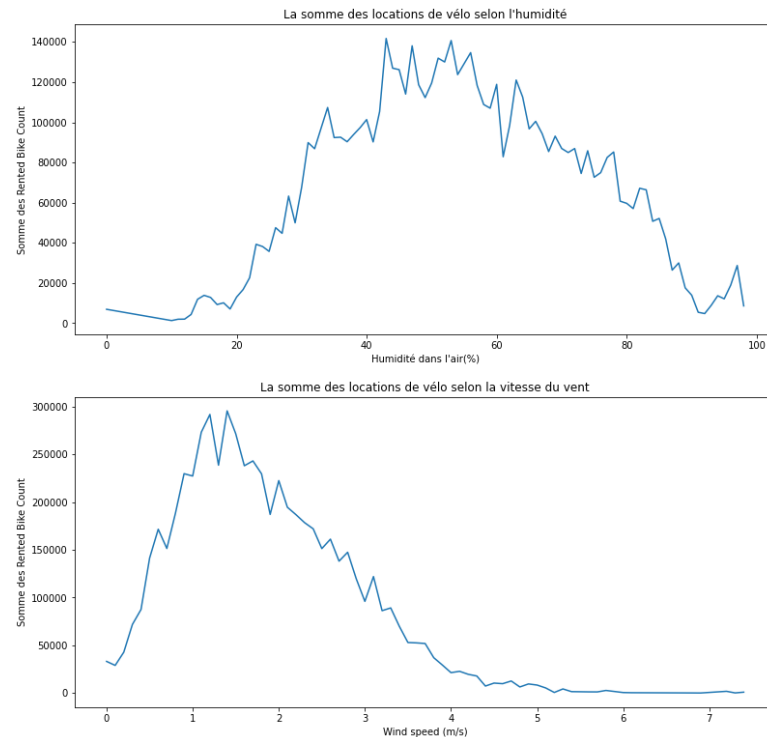
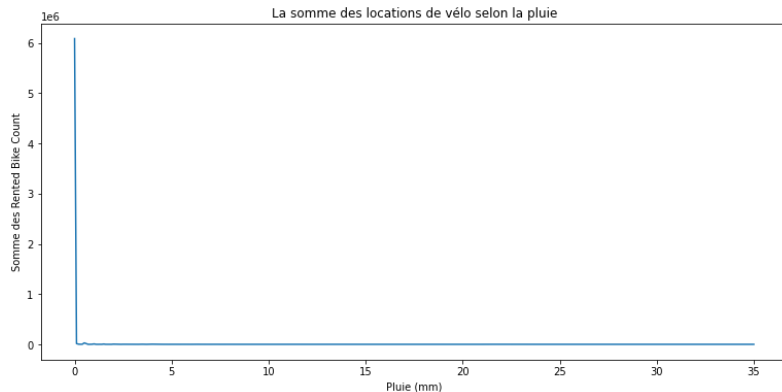
Nous sommes ensuite parti sur nos données temporelles, nous avons tracé l'évolution des locations de vélos au cours des saisons, mois, jour et heures. On a pu mettre en évidence des périodes propices à leur location, en effet on remarque que l'hiver est la période avec la demande la plus faible, cela pourrait servir à la ville pour une étude de maintenance des stations afin de garantir la sureté et la disponibilité des vélos lors des périodes de plus forte demande. De plus, on a pu mettre en évidence des habitudes dans les horaires de locations avec des pic à 8h et 18h cela correspond probablement aux horaires moyennes des travailleurs à Seoul.





Dans la continuité de l'étude des variables temporelles, nous avons voulu voir l'impact des jours de la semaine sur la location de vélo, nous avons tout d'abord fait la somme des vélos loués en fonction du jour de la semaine (1), nous n'avons pas trouvé de différence notable sur cette représentation, nous avons alors voulu faire ressortir les différents en zoomant sur les différents mois (2, nous avons montrés le résultat sur 3 mois, les 9 autres sont disponibles dans le notebook), sur cette nouvelle représentation nous voyons apparaitre des patterns sur certains mois, cela nous permet de dire que la variable contenant les jours de la semaine est intéressant allié aux mois.

Nous nous sommes ensuite intéresser aux variables météorologiques avec l'humidité, la vitesse du vent, la quantité de pluie, la quantité de neige. On observe ainsi que la location de vélo est très impacté par les conditions météorologiques diverses que ce soit par l'humidité, de l'air, la vitesse du vent ou encore la pluie. De plus, on peut voir que dès qu'il pleut très peu de personne loue des vélos, en se basant sur le graphiques ci-dessous nous pouvons voir que cette variable sous cette forme n'apporte pas de plus-value car on peut assimiler la pluie à une location de vélo proche de 0, on décide donc de la passer en booléen.



Maintenant que l'analyse des données est faite et que nous avons fait ressortir les variables intéressantes, nous allons réfléchir aux modèles à employer.

Il s'agit d'un problème de régression, nous allons donc employer les différents algorithmes associés à ce genre de problème. Ainsi dans cette partie nous allons aborder les algorithmes suivants :

- Régression linéaire simple
- Régression polynomiale
- Random Forest
- LGB
- XGBoost
- CatBoost

Pour déterminer lequel de ces algorithmes nous donnera les meilleurs résultats nous allons tout d'abord découper notre dataset en train et test afin d'évaluer notre modèle sur des données auxquels il n'a pas pu s'entraîner.

Ensuite au niveau de l'évaluation des modèles nous allons utiliser le **R<sup>2</sup> et RMSE** qui permettent de donner une indication de l'erreur de notre modèle. R<sup>2</sup> est compris entre 0 et 1, plus il est proche de 1, plus notre algorithme est précis. RMSE c'est l'inverse, plus il est faible et plus il signifie que l'erreur est faible et donc que notre algorithme est performant.



## Transformation des données:

Les données en string ne sont pas interprétable tel quel par un algorithme, il faut les transformer soit en énumérations, soit en vecteur. Ici nous allons transformer les données suivantes :

- Season : string to enum de 1 à 4
- Functionning day : string to bool
- Holiday : string to bool
- Date : suppression car on l'a déjà décomposé en année, mois, jour

## Séparation des données :

Dans un contexte de mission lié au machine learning, on décompose le dataset en 2 parties :

- Train : partie servant à entraîner le modèle (70% dataset)
- Test : partie servant à évaluer le modèle (30%), cette partie est censé permettre d'évaluer le modèle sur des données jamais touché par le modèle

Il arrive également souvent que le jeu test soit garder par le client (ex : concours kaggle) et que nous devions nous même le créer comme ici afin d'évaluer notre modèle avant de le soumettre au client.

Nos données sont maintenant prêtes pour notre modèle.

# Vers des premiers modèles simples

## Regression linéaire :

Le premier algorithme que nous allons utiliser est une régression linéaire simple. Cela consiste à essayer de résoudre l'équation  $y = a \cdot X + b$  avec  $y$  = le nombre de vélos loués par heure (à déterminer),  $X$  = les autres données de notre dataset et  $a$  et  $b$  = coefficient à déterminer

Nous obtenons comme résultat : **R2 de 0.53 et un RMSE de 441.**

Ce premier algorithme nous donne des résultats assez médiocre cela vient du fait qu'il a du mal à trouver une droite permettant de représenter les données. Essayons donc de l'améliorer en ne passant plus par une droite mais par un polynôme de degré  $N$ .

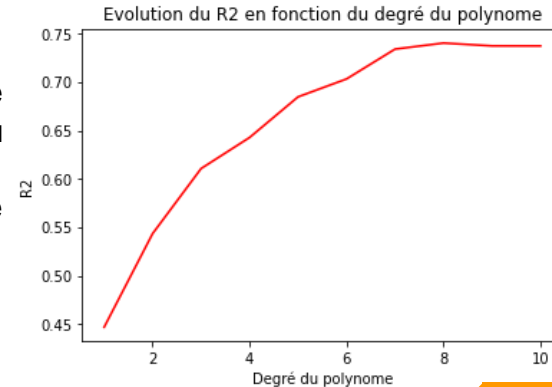
## Régression polynomiale de degré $N$ :

En passant par une régression polynomiale de degré  $N$ , notre nouvelle formule est  $y = aX_1 + bX_2 + cX_n + \dots + \text{constante}$ , avec  $N$  le degré du polynôme.

En bouclant sur différents degrés de polynômes, nous trouvons que le degré optimal est 8. (cf : la figure à droite)

Ainsi nous obtenons de meilleur résultat avec :

- **R2 : 0.74**
- **RMSE : 329**



## Random Forest :

"L'algorithme des forêts d'arbres décisionnels effectue un apprentissage sur de multiples arbres de décision entraînés sur des sous-ensembles de données légèrement différents.". Globalement on va construire différents arbres avec différentes colonnes de notre dataset que l'on va entraîner sur nos données et ensuite on va essayer de déterminer le meilleur arbre parmi tous ceux créés.

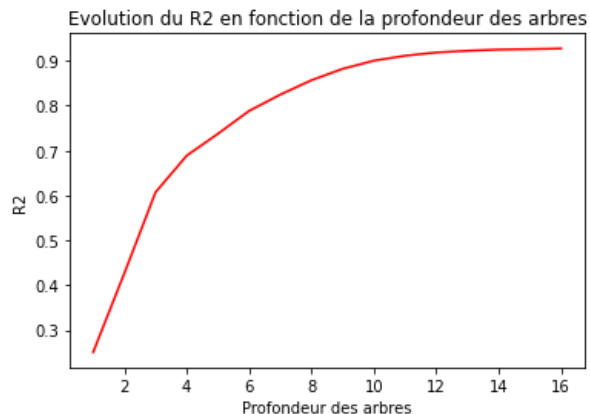
RandomForest a deux paramètres très important :

- max\_depth = la profondeur des arbres
- n\_estimators = le nombre d'arbres

Nous allons boucler différentes valeurs sur ces paramètres afin de trouver le meilleur modèle de RandomForest pour nos données.

Ainsi une fois les paramètres établis, nous obtenons de bien meilleur résultat avec :

- R2 : 0.927
- RMSE : 174



## Gradient boosting:

Le principe du gradient boosting ressemble au bagging vu précédemment avec randomForest. C'est-à-dire que l'on va créer plusieurs modèles que nous agrégeons ensuite pour n'avoir qu'un seul résultat. Pour chaque modèle créé nous allons calculer la différence entre ce que nous avons obtenu et le résultat attendu (appelée la "fonction de perte"). A partir de celui-ci nous allons imputer des poids à chaque node des modèles. Puis on recommence encore et encore X fois, ce qui va permettre au fur et à mesure d'apprendre des erreurs passées et d'obtenir le meilleur modèle.

Dans ce genre d'approche, le plus dur est d'éviter le surapprentissage des données, c'est-à-dire que notre modèle ce soit trop adapté aux données de train et peine à être suffisamment généraliste pour obtenir de bon résultat sur les tests.

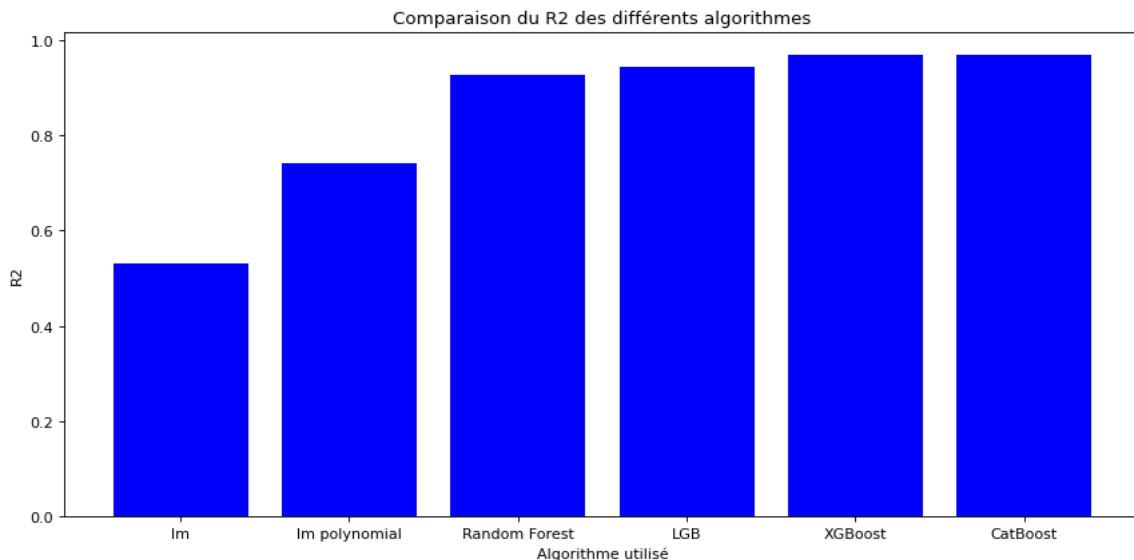
Nous avons étudié 3 algorithmes de boosting différents dans lequel nous avons optimiser les différents paramètres afin d'obtenir les meilleurs modèles pour chaque algorithme.(cf : partie « Gradient boosting » du notebook)

Ainsi nous obtenons les résultats suivants :

- LGB :  $R^2 = 0.942$  , RMSE = 152
- XGBoost :  $R^2 = 0.949$  , RMSE = 146
- CatBoost :  $R^2 = 0.959$  , RMSE = 130

Catboost est l'algorithme de boosting nous offrant le meilleur score.

Pour répondre à cette problématique de régression linéaire supervisé, nous avons utilisé différents algorithmes afin dans un premier temps d'étudier ceux-ci et également dans l'optique d'améliorer nos prédictions afin d'obtenir le meilleur modèle pour notre API. Ci-contre on peut voir la comparaison entre les différents résultats que nous avons pu avoir. On remarque une domination des algorithmes de boosting parmi les meilleurs résultats. En l'occurrence CatBoost est celui ayant retourné les meilleures prédictions, cela s'explique en grande partie par le fait qu'il s'adapte aux données très rapidement par rapport à ses concurrents sur des données moins abondantes.



Si nous devions aller plus loin, nous aurions pu tester d'autres approches tel que le SVM ou encore une approche temporelle avec un LSTM.

# Un concours kaggle

En regardant notre dataset sur internet, nous sommes tombés sur un concours kaggle lié (<https://www.kaggle.com/c/seoul-bike-rental-prediction/overview>). Celui-ci n'a pas eu beaucoup de participants et aucune personne n'a publié d'étude dessus. Nous avons vu que nous pouvions toujours participer à ce concours. Nous avons alors décidé de participer pour tester notre modèle.

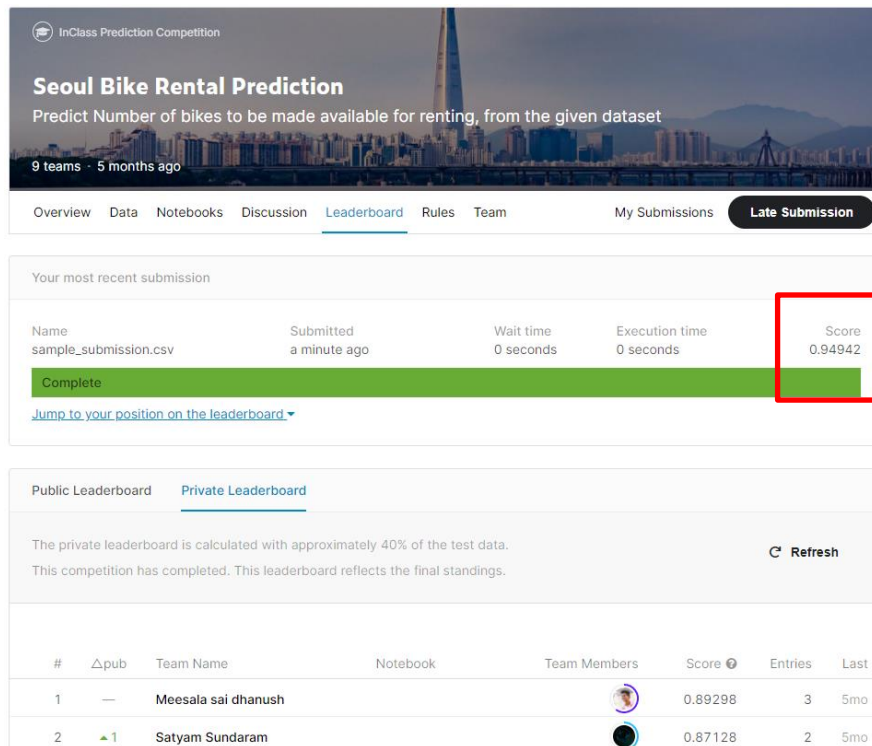
Ce concours se base sur le R2 comme évaluation.

Après upload sur le concours kaggle, nous avons obtenu un R2 de :

- **Private Score : 0.93549**
- **Public Score : 0.94942**

Ce qui nous place sur la première place du concours Kaggle (de 10 participants).

**On en conclut donc que notre modèle est performant par rapport à la problématique demandée**



InClass Prediction Competition

## Seoul Bike Rental Prediction

Predict Number of bikes to be made available for renting, from the given dataset

9 teams · 5 months ago

Overview Data Notebooks Discussion **Leaderboard** Rules Team My Submissions **Late Submission**

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
sample_submission.csv	a minute ago	0 seconds	0 seconds	0.94942

Complete

[Jump to your position on the leaderboard](#)

Public Leaderboard **Private Leaderboard**

The private leaderboard is calculated with approximately 40% of the test data. This competition has completed. This leaderboard reflects the final standings. Refresh

#	△pub	Team Name	Notebook	Team Members	Score	Entries	Last
1	—	Meesala sai dhanush			0.89298	3	5mo
2	▲ 1	Satyam Sundaram			0.87128	2	5mo

Dans la partie précédente, nous avons étudié différents modèles afin de prédire le nombre vélo loué à Seoul sur une certaine heure. Le modèle ayant retourné les meilleurs résultats a été enregistré et nous allons le réutiliser sous forme d'une API afin de réaliser des prédictions.

Une API fonctionne généralement via un système de Client-Serveur. Le serveur va contenir notre modèle et va prédire le nombre de vélos loués à partir des données fournis par le client lors d'une requête.

Les étapes de fonctionnement sont :

1. On lance le serveur sur une adresse IP et un port, on met en place un point d'accès au serveur via /api
2. Le client va émettre une requête sur notre serveur via son url "<http://IP:Port/api>" (la requête contient les données nécessaire pour prédire)
3. Le serveur reçoit la requête avec les données (GET) , ils utilisent le modèle pour prédire un résultat, le serveur retourne la prédiction (POST)
4. Le client reçoit la réponse de la prédiction

Pour mettre en place notre API, plusieurs solutions s'offraient à nous, parmi celles-ci Flask est celle que nous avons retenu (cf : notebook API\_velos\_loues\_seoul)

L'API retourne ainsi le résultat de la prédiction à partir des données transmises comme-ça :

```
Le nombre de vélo loué pour cette heure et ce jour là sera de : "234"
```

Au terme de ce projet, nous avons vu que la première étape était tout d'abord d'avoir une bonne connaissance des données sur les lesquels nous travaillons. Cela a pour but de nous permettre d'aborder dans les meilleures conditions la problématique posée et les besoin métiers que cette dernière implique. Il est également selon nous primordial d'identifier les données réellement utiles que nous allons ensuite apporter aux différents algorithmes implémentés. Ainsi, nous avons pu déterminer les principales variables, ayant une incidence sur le nombre de locations de vélos à Seoul.

Puis il a été question de mettre en place les algorithmes de prédiction du nombre de vélos loués par heure. Pour ce faire, nous avons fait le choix de réaliser une régression linéaire simple, une régression polynomial, randomForest, des méthodes de Gradient Boosting LGB, XGBoost et enfin CatBoost.

Au travers de ces différentes stratégies, nous avons pu apprécier les résultats obtenus et les interpréter afin d'en tirer nos conclusions. De même, cet exercice nous a permis de dégager des pistes de réflexion et d'améliorations essentielles dans notre démarche d'arriver vers des modèles plus performants, de telle manière à finalement pouvoir proposer des résultats des plus probants.