



TÉLÉCOM SUDPARIS

CSC 8609
RAPPORT DE PROJET
DÉPÔT GITHUB

Prédiction de la demande de partage de vélos

Élève :
Yann MILLET

Enseignant :
Julien ROMERO

12 janvier 2025



Table des matières

1	Description des données	2
1.1	Présentation du dataset	2
1.2	Description globale	2
1.3	Taille	3
1.4	Relations évidentes entre attributs	3
1.5	Défis anticipés	8
2	Data Cleaning	9
2.1	Valeurs manquantes	9
2.2	Données aberrantes	9
2.3	Données redondantes	13
2.4	Normalisation des données	14
2.5	Outils de nettoyage	14
3	Feature engineering	14
3.1	Caractéristiques pertinentes	14
3.2	Nouvelles caractéristiques	15
3.3	Encodage des catégories/Transformation et mise à l'échelle des caractéristiques	15
4	Model Selection	16
4.1	Modèles testés	16
4.2	Paramètres et Hyperparamètres	18
4.3	Modèle supplémentaire	20
4.4	Gestion des données déséquilibré	23
4.5	Division en train/validation/test	23
5	Évaluation	24
5.1	Métriques	24
5.2	Étude d'ablation	24
5.3	Analyse des erreurs	25
5.4	Cross-validation	26
5.5	Comparaison des performances	26
6	Discussion et conclusion	27
6.1	Parce que je sais pas noter mes devoirs	27
6.2	Défis importants	27
6.3	Fiabilité	28
6.4	Travaux futurs	29
6.5	Analyse critique	30
6.6	Leçons apprises	30

1 Description des données

1.1 Présentation du dataset

De quoi parle le jeu de données et quelle est la tâche de prédiction ?

Ce jeu de données s'appelle "Bike Sharing" et contient des données relatives à la location de vélo. Les données ont été récoltées en 2011 et 2012. Le dataset contient des données liées aux conditions météorologiques (`weathersit`, `temp`, `atemp`, `hum`, `windspeed`), des données calendaires (`instant`, `dteday`, `season`, `yr`, `mnth`) et des données chiffrées sur le nombre de locations (`casual`, `registered` et `cnt`).

Le dataset est composé de 2 fichiers, `day.csv` et `hour.csv`, contenant chacun les mêmes informations à deux niveaux de granularité différents, respectivement journalier et horaire. Le fichier `hour.csv` contient un attribut supplémentaire `hr`.

En utilisant l'attribut `cnt` comme target, une tâche possible avec ce dataset est de tenter de prédire la demande de location de vélos en fonction des conditions météorologiques et calendaires. C'est donc une tâche de regression même si les prédictions seront arrondies parce qu'on ne peut pas louer une fraction de vélo.

1.2 Description globale

Combien de variables (colonnes) contient le jeu de données ? Quels types de données contiennent-elles (catégoriques, numériques, etc).

Le dataset contient 16 (respectivement 17) variables pour le fichier `day.csv` (respectivement `hour.csv`). Les features catégoriques sont pour chaque échantillon :

- `season` : **encodé ordinalement** (1 hiver, 2 printemps, 3 été, 4 automne). Représente la saison de l'échantillon.
- `yr` : **encodé ordinalement** (0 2011, 1 2012), représente l'année de l'échantillon.
- `mnth` : **encodé ordinalement** (de 1 à 12, de janvier à décembre), représente le mois de l'année.
- `hr` : **encodé ordinalement** (de 0 à 23), représente le créneau horaire dans lequel on se trouve.
- `holiday` : attribut *binaire* qui découle de l'**encodage ordinal** d'un attribut catégorique "holiday"/"not-holiday". 1 signifie que le jour considéré est un jour de vacances, 0 que ce n'est pas le cas.
- `weekday` : **encodé ordinalement** (de 0 à 6 de lundi à dimanche), représente le jour de la semaine.
- `workingday` : attribut *binaire* qui résulte de l'**encodage ordinal** d'un attribut catégorique "working day"/"not workingday". 1 signifie que l'échantillon correspond à un jour travaillé.
- `weathersit` : **encodé ordinalement** pour discrétiser les conditions météorologiques (de 1 à 4, par degré de difficulté croissant).

Les features numériques sont :

- `instant` : index d'enregistrement.

- `temp` : température en degrés Celsius, **normalisée** par $\frac{temp-t_{min}}{t_{max}-t_{min}}$ où $t_{min} = -8$ et $t_{max} = +39$.
- `atemp` : température **ressentie** en degrés Celsius, normalisée par $\frac{temp-t_{min}}{t_{max}-t_{min}}$ où $t_{min} = -16$ et $t_{max} = +50$.
- `hum` : taux d'humidité dans l'air, **normalisé** en divisant par 100 (le maximum de cette valeur puisque c'est un pourcentage).
- `windspeed` : vitesse du vent (probablement en *miles per hour*), **normalisé** en divisant par 67 (maximum enregistré, même si je n'ai repéré de valeur que jusque 57mph).
- `casual` : nombre d'utilisateurs occasionnels non enregistrés/inscrits qui ont loué un vélo pendant sur le créneau de l'échantillon.
- `registered` : nombre d'utilisateurs enregistrés/inscrits ayant loué un vélo sur le créneau de l'échantillon.
- `cnt` : nombre total de vélos loués lors de l'échantillon (`casual` + `registered`).

Le dernier attribut est `dtoday`, qui représente la date du jour de chaque échantillon, de type `object`.

1.3 Taille

Quelle est la taille du jeu de données (nombre de lignes et de colonnes) ?

Le dataset est donc constitué de 2 fichiers : `day.csv` et `hour.csv`. Le premier contient 731 lignes pour 16 colonnes, le second contient 17379 lignes pour 17 colonnes, ce qui correspond environ à 24 fois la taille du premier fichier.

```
print(
    f"The dataset DAY contains {day_df.shape[0]} samples and "
    f"{day_df.shape[1]} columns."
)
print(
    f"The dataset HOUR contains {hour_df.shape[0]} samples and "
    f"{hour_df.shape[1]} columns."
)
✓ 0.0s
```

The dataset DAY contains 731 samples and 16 columns.
The dataset HOUR contains 17379 samples and 17 columns.

FIGURE 1 – Taille des deux fichiers

1.4 Relations évidentes entre attributs

Y a-t-il des motifs ou des relations évidentes dans les données ? Fournissez quelques visualisations (par exemple, histogrammes, tableau de corrélation).

Certains attributs font l'objet de relations évidentes. On remarque notamment que l'attribut `dtoday` rassemble les informations des colonnes `yr`, `mnth`, `weekday`, et même `hr` pour le fichier

hour.csv. En convertissant la colonne en format `datetime` puis en `int64` (car `scikit-learn` ne peut pas traiter des données `datetime`), on perd le détails périodiques de l'information (cf. FIG.2) sur les jours, mois, ...

```

day_df['dteday'] = pd.to_datetime(day_df['dteday']).astype(np.int64)
hour_df['dteday'] = pd.to_datetime(hour_df['dteday']).astype(np.int64)

#hour_df.info()
#day_df.info()

hour_df['dteday']
✓ 0.0s

0      1293840000000000000
1      1293840000000000000
2      1293840000000000000
3      1293840000000000000
4      1293840000000000000
...
17374   1356912000000000000
17375   1356912000000000000
17376   1356912000000000000
17377   1356912000000000000
17378   1356912000000000000
Name: dteday, Length: 17379, dtype: int64

```

FIGURE 2 – Perte d'informations périodiques à la conversion du `datetime`

Une autre relation évidente se trouve entre `casual` et `registered`. Ces 2 attributs se complètent pour former l'attribut `cnt`, qui sera notre target (FIG.3).

```

# vérif que casual + registered = cnt
verif = hour_df.apply((lambda row: row['casual']+row['registered']==row['cnt']), axis=1)
verif.value_counts()
✓ 0.1s

True      17379
dtype: int64

```

FIGURE 3 – Relation `casual+registered=cnt`

Au niveau des motifs repérables dans les données, on en remarque plusieurs également.

Premièrement, l'évolution temporelles du nombre de location sur 2 ans (FIG.4), on y remarque très clairement 2 pics au niveau de l'été, qui illustre la saisonnalité de ce genre d'activités en plein air.

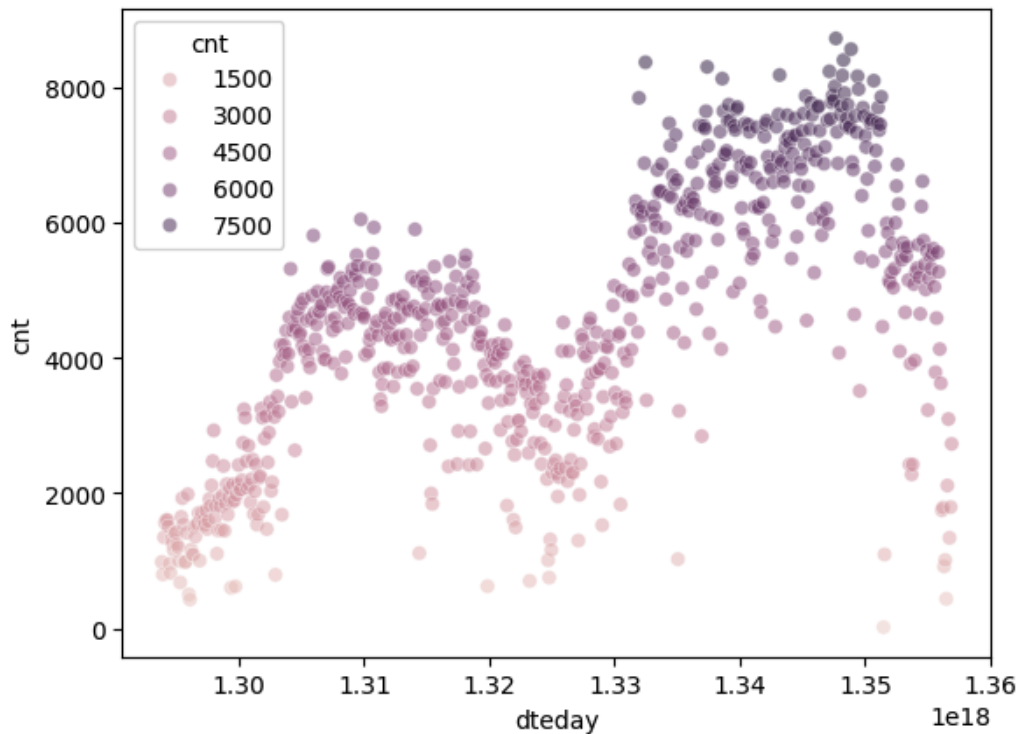


FIGURE 4 – Évolution dans le temps des locations de vélos

Deuxièmement, la **température** en fonction de la **température ressentie** illustre l'erreur de perception lorsque les températures s'éloignent de la zone auquel un humain est habitué. On remarque sur la FIG.5 (graphe en bas à gauche ou d'en haut à droite) que la courbe s'écarte de $y = x$ sur les bords, ce qui se comprend car on ressent toujours plus froid lorsqu'on a très froid, et toujours plus chaud lorsqu'on a trop chaud (c'est ainsi qu'est estimée la température ressentie relativement à la température réelle).

On remarque aussi quelques relations météorologiques, telles d'une plus grande densité de jours avec une température plutôt élevée et une humidité basse, caractéristique des jours plus chauds (on y remarque également un plus grand nombre de locations).

On pourrait également se dire que les distributions des températures (réelles et ressenties) suivent une sorte de double gaussienne, qui pourrait laisser penser qu'il s'agit d'un pic pour la valeur moyenne basse en hiver, et la valeur moyenne haute en été, comme on peut le voir sur l'histogramme basique de **pandas** (FIG.6). Cependant lorsque l'on augmente le nombre de bins sur l'histogramme (FIG.7) que ce n'est qu'une impression et que la distribution est plus complexe que cela. Cependant, sur les courbes de densité (FIG.8), les pics sont centrés sur des valeurs qui se corrélaient aux moyennes saisonnières calculées : pour le premier pic, environ 22 degrés qui est la moyenne de température pour la saison 1 (hiver), et pour le second pic, environ 36 degrés pour une moyenne estivale de 41 degrés (41 degrés étant la moyenne calculée pour la température ressentie pendant la saison 3).

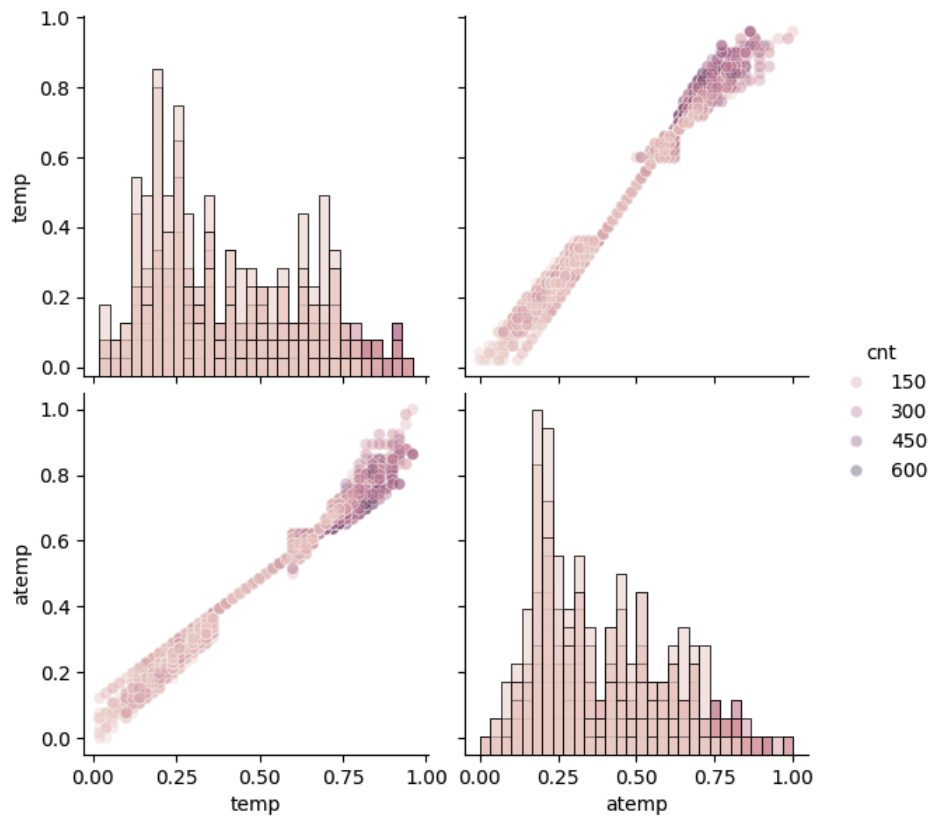


FIGURE 5 – Divergences entre température réelle et température ressentie

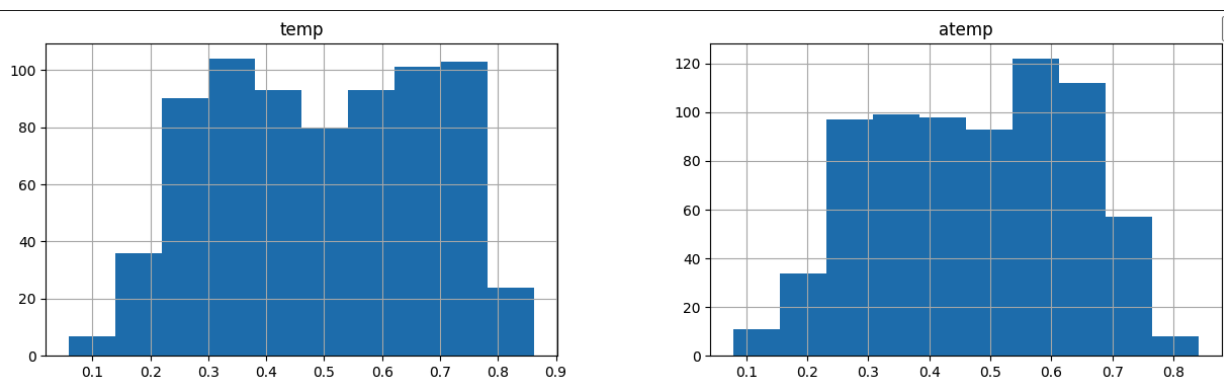


FIGURE 6 – Histogrammes avec faible nombre de bins (10)

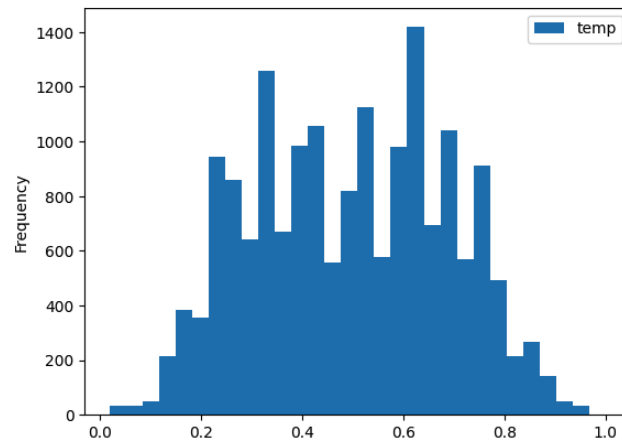


FIGURE 7 – Histogramme avec 3x plus de bins (30)

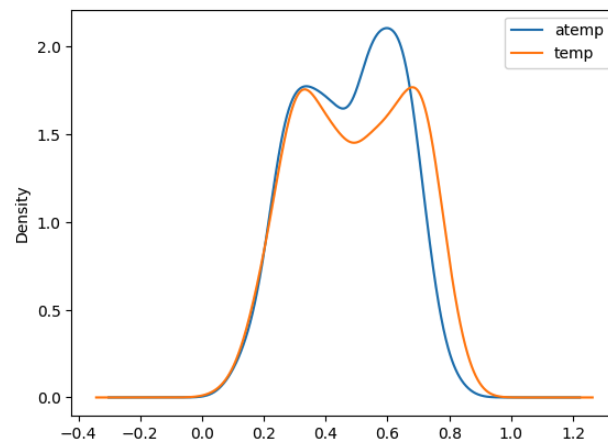


FIGURE 8 – Densité des différents attributs de température

```
# moyenne hivernale
print(hour_df.loc[hour_df["season"]==1]["temp"].mean()*47 +8)
# moyenne estivale
print(hour_df.loc[hour_df["season"]==3]["temp"].mean()*47 +8)
✓ 0.0s

22.05989156058463
41.20127669039145
```

FIGURE 9 – Calcul des moyennes saisonnières de température

1.5 Défis anticipés

Quels sont les défis potentiels que vous anticipez en travaillant avec ce jeu de données (par exemple, données manquantes, déséquilibre, haute dimensionnalité) ?

Parmi les défis anticipés, l'apparente simplicité du dataset peut devenir un problème. Il s'agit de ne pas sursimplifier les relations car on risque de finir avec des interprétations trop simples ou et de se perdre avec un modèle mal adapté.

Un autre challenge sera de réussir à mitiger les déséquilibres dans différents attributs. Parmi les attributs à surveiller il y a le nombre de jours de vacances versus le nombre de jours hors vacances (différent des jours travaillés car comprend les weekends), le nombre de jours de travail versus le nombre de jours de repos, et enfin les différentes conditions météo. Pour ce dernier attribut, on remarque un grand déséquilibre entre les classes de conditions 1 et 2 par rapport aux 3 et 4. La catégorie 4 n'est même pas représentée dans le fichier `day.csv` et n'est présente qu'à 4 reprises dans `hour.csv`.

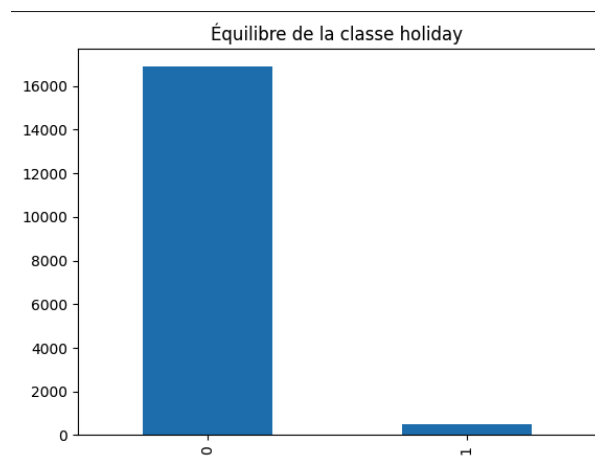


FIGURE 10 – Déséquilibre flagrant de la classe `holiday` dans le fichier `hour.csv`

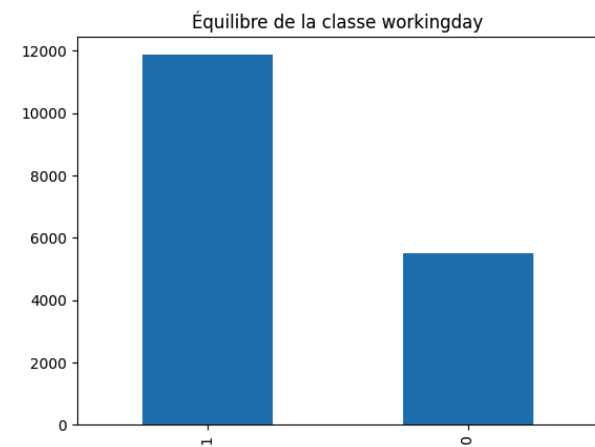
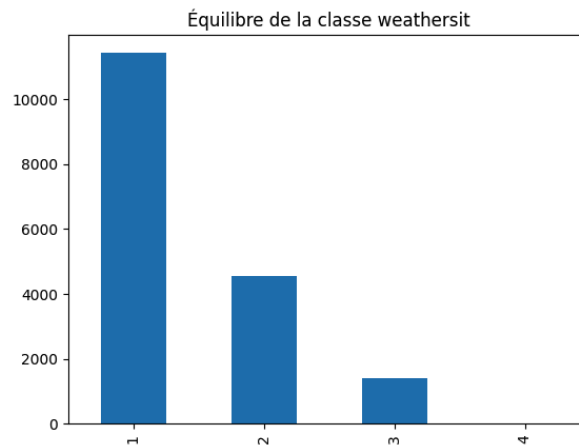


FIGURE 11 – Déséquilibre de la classe `workingday` dans le fichier `hour.csv` (moins flagrant que 'holiday')

FIGURE 12 – Déséquilibre flagrant de la classe `weathersit` dans le fichier `hour.csv`

2 Data Cleaning

2.1 Valeurs manquantes

Y a-t-il des valeurs manquantes dans le jeu de données ? Si oui, comment les avez-vous traitées ?

Le dataset n'affiche aucune valeur manquante sur le site de téléchargement (FIG.13), ce que j'ai pu vérifier en affichant les informations du dataset avec `pandas` (FIG.14). Il n'y avait donc rien à signaler de ce côté là. On peut remarquer qu'il manque quelques échantillons dans le dataset `hour.csv` si on veut matcher parfaitement 24 fois le nombre de jours dans `day.csv`, mais l'important est d'avoir des échantillons complets et diversifiés et non pas d'avoir une chronologie complète d'un dataset à l'autre.

2.2 Données aberrantes


Y a-t-il des valeurs aberrantes ou des données anormales ? Comment les avez-vous identifiées et traitées ?

En observant les pairplots suivants (FIG.15 et FIG.16), il semble y avoir peu ou pas d'**outliers** dans le dataset.

Aucun point de données ne semble isolé radicalement et tout semble dans des ordres de grandeurs acceptables à première vue, mais ici je n'ai tracé que 5000 points par plot pour aller plus vite et ne pas surcharger les tracés, donc ça ne représente pas tous les points. En traçant les **boxplots** (FIG.17) pour les données numériques que nous allons utiliser pour l'apprentissage (`windspeed`, `temp`, `atemp` et `hum`), on remarque la présence de plusieurs **points aberrants** (outliers).

On parle d'outliers lorsqu'un point dépasse de plus de $3/2$ le dernier quantile (par le haut) ou est $3/2$ fois inférieur au premier quantile (par le bas). Sur la FIGURE 17 ces points sont mis en valeur par des cercles noirs.

Il n'y a ici aucune raison valable d'enlever des outliers, car ce sont des points de données



Bike Sharing

Donated on 12/19/2013

This dataset contains the hourly and daily count of rental bikes between years 2011 and 2012 in Capital bikeshare system with the corresponding weather and seasonal information.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Social Science	Regression
Feature Type	# Instances	# Features
Integer, Real	17389	13

Dataset Information

Additional Information

Bike sharing systems are new generation of traditional bike rentals where whole process from membership, rental and return back has become automatic. Through these systems, user is able to easily rent a bike from a particular position and return back at another position. Currently, there are about over 500 bike-sharing programs around the world which is composed of over 500 thousands bicycles. Today, there exists great interest in these systems due to their important role in traffic, environmental and health issues.

Apart from interesting real world applications of bike sharing systems, the characteristics of data being generated by these systems make them attractive for the research. Opposed to other transport services such as bus or subway, the duration of travel, departure and arrival position is explicitly recorded in these systems. This feature turns bike sharing system into a virtual sensor network that can be used for sensing mobility in the city. Hence, it is expected that most of important events in the city could be detected via monitoring these data.

[SHOW LESS](#)

Has Missing Values?
No

FIGURE 13 – Page web de téléchargement du dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 17 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   instant     17379 non-null  int64
1   dteday      17379 non-null  int64
2   season      17379 non-null  int64
3   yr          17379 non-null  int64
4   mnth        17379 non-null  int64
5   hr          17379 non-null  int64
6   holiday     17379 non-null  int64
7   weekday     17379 non-null  int64
8   workingday  17379 non-null  int64
9   weathersit   17379 non-null  int64
10  temp        17379 non-null  float64
11  atemp       17379 non-null  float64
12  hum         17379 non-null  float64
13  windspeed   17379 non-null  float64
14  casual      17379 non-null  int64
15  registered  17379 non-null  int64
16  cnt         17379 non-null  int64
dtypes: float64(4), int64(13)
memory usage: 2.3 MB
```

FIGURE 14 – Vérification de l'absence de données manquantes (hour.csv)

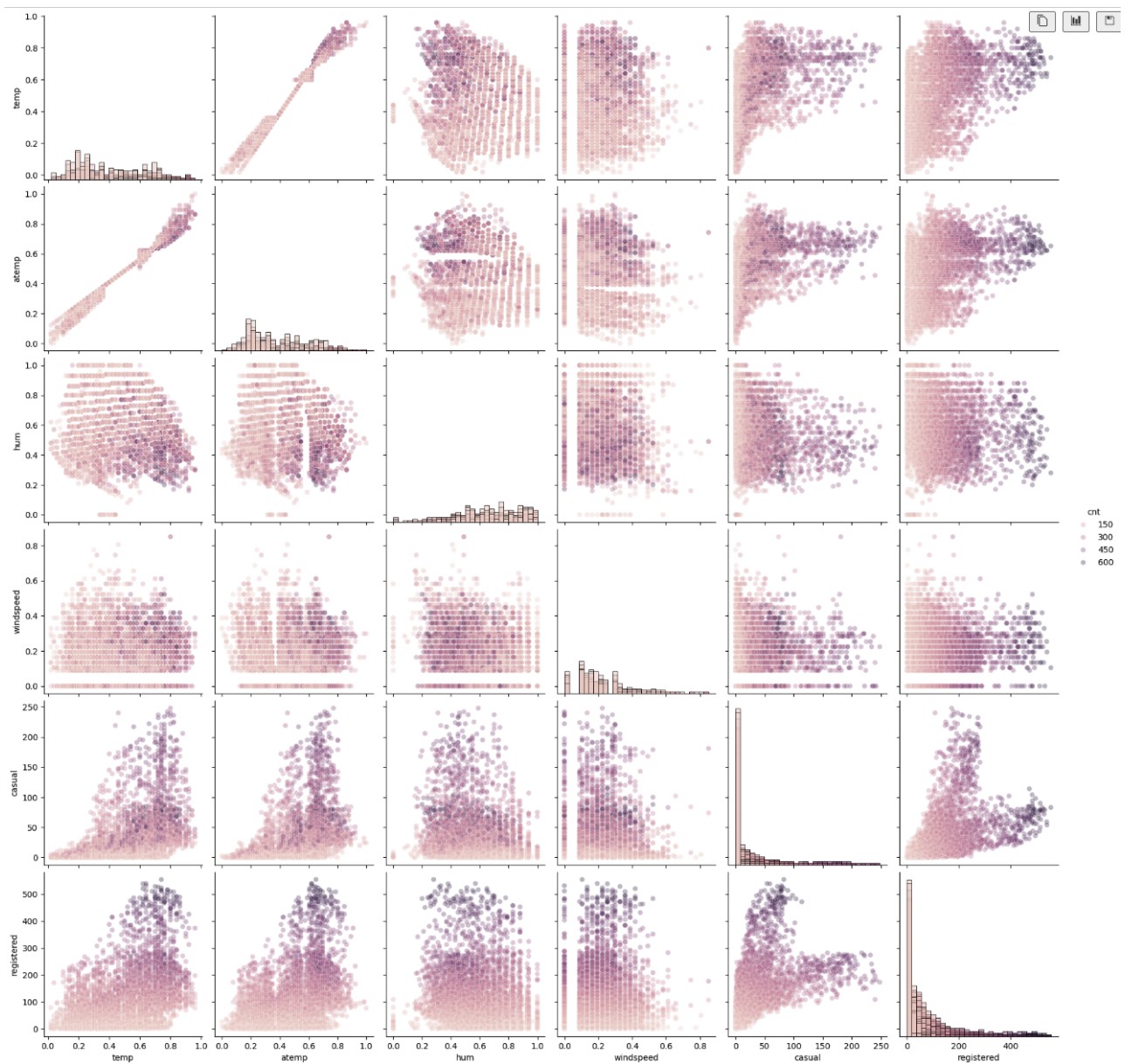


FIGURE 15 – Pairplot des valeurs numériques entre elles

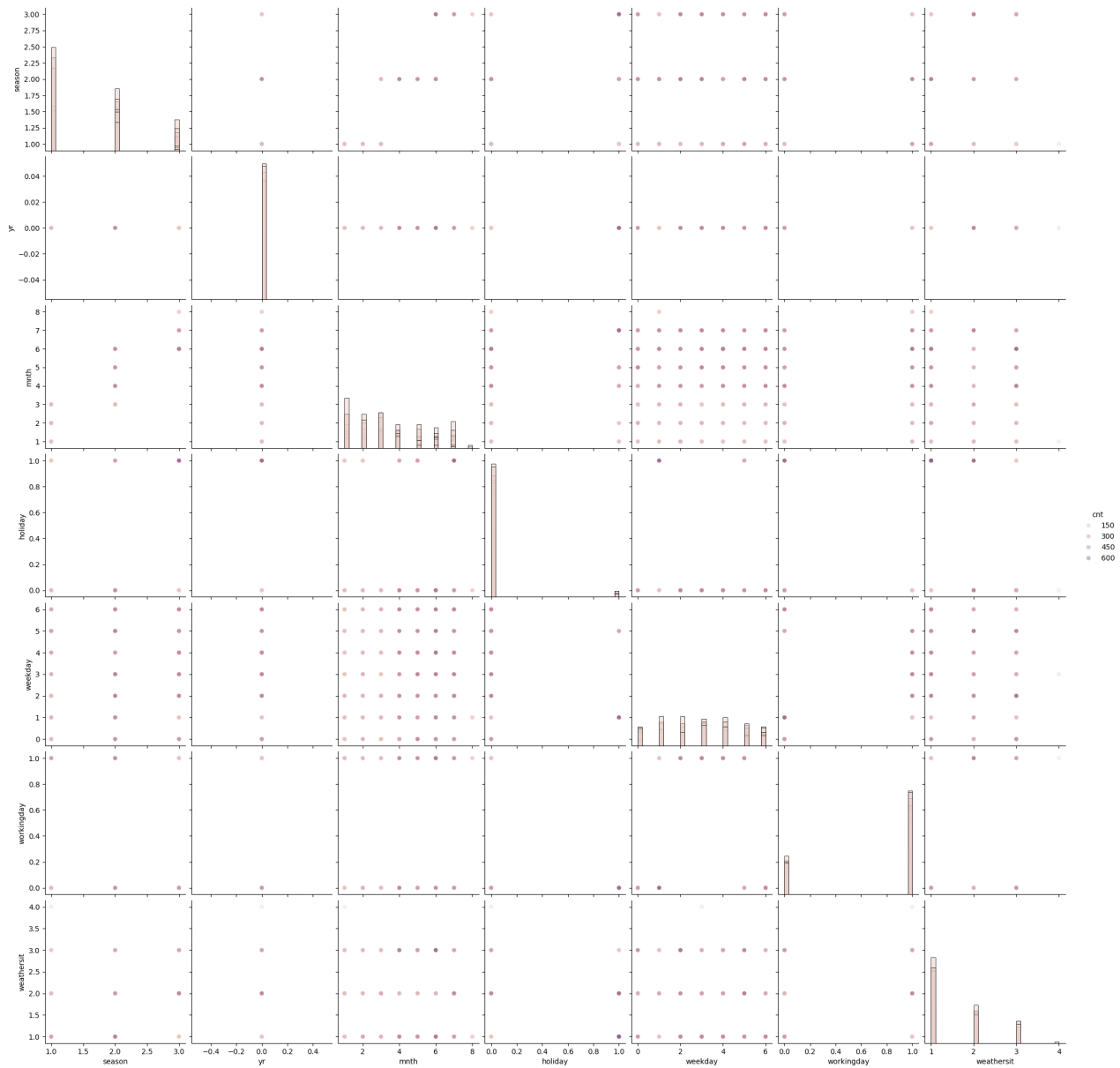


FIGURE 16 – Pairplot des valeurs catégoriques entre elles

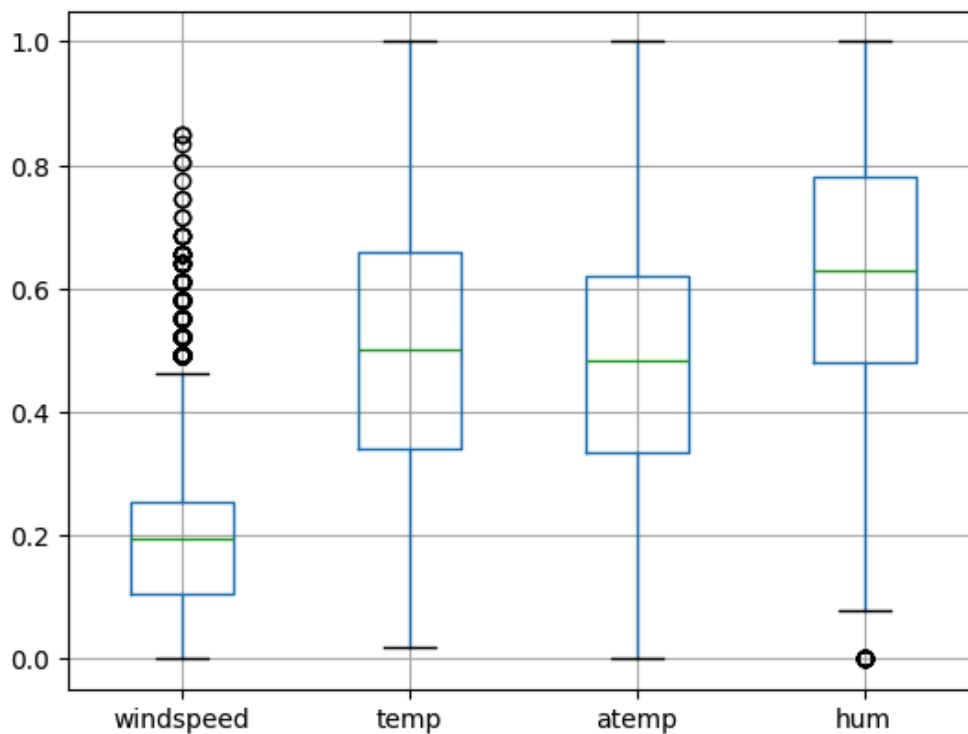


FIGURE 17 – Boxplots des attributs numériques révélant des outliers

tout à fait corrects. Il s'agira tout de même d'essayer de déterminer leur influence sur l'apprentissage du modèle, notamment pour les points de **windspeed** puisqu'ils sont plus nombreux et très écartés de des quartiles centraux. On pourra notamment évaluer les performances de nos modèles en les ré-entraînant avec et sans ces outliers. On mesurera aussi les performances avec des métriques peu sensibles aux outliers (qui sont moins nombreux que la masse) comme la **médiane**.

2.3 Données redondantes

Avez-vous rencontré des données incohérentes ou redondantes ? Comment avez-vous résolu ces problèmes ?

Les données de ce dataset rendent compte d'une série temporelle, l'attribut **instant** indexant les échantillons de manière unique (puis vérification FIG.18 de l'absence de lignes identiques) on sait que les valeurs sont uniques. S'il venait à y avoir 2 échantillons identiques par des conditions météo similaires, ils seraient forcément distingués par leur **instant** et leurs données calendaires.

Cependant, certaines colonnes sont redondantes entre elles. Comme nous l'avons vu plus haut, l'attribut **dteday** est un condensé moins précis de toutes les autres données calendaires, il ne servira ici à rien et on l'écarte du dataset. De même, la température ressentie **atemp** est très corrélée à la température réelle **temp**. Je veux éviter tout problème de multicolinéarité entre ces 2 attributs (ce qui pourrait compliquer l'explicabilité de ces variables dans la prédiction finale) alors j'écarte tout simplement **atemp** des données.

Les attributs **casual** et **registered**, nous l'avons aussi vu, permettent de faire le lien avec

```
print(day_df.duplicated().value_counts())  
print(hour_df.duplicated().value_counts())  
  
✓ 0.0s  
  
False      731  
dtype: int64  
False     17379  
dtype: int64
```

FIGURE 18 – Vérification de l'absence de doublons

la target si nous en disposons. C'est justement ce que l'on cherche à prédire, même si nous ne cherchons pas dans ce projet à prédire la distinction (une analyse de données pourra être effectuée en conclusion pour tenter d'estimer la répartition de la prédiction globale). Ces deux colonnes sont également supprimées du dataset.

2.4 Normalisation des données

Avez-vous normalisé ou standardisé les données ? Expliquez pourquoi et comment.

Comme indiqué dans la section 1.2 sur la description des données, toutes les données numériques sont déjà normalisées (selon une normalisation MinMax). Les données catégoriques sont déjà encodées, et bien que j'ai commencé par penser que j'allais changer l'encodage de certains attributs (par exemple les saisons, car un printemps (encodage 2) ne vaut pas deux hivers (encodage 1)), j'ai finalement gardé tous les encodages ordinaux.

2.5 Outils de nettoyage

Quels outils avez-vous utilisés pour le nettoyage des données ? Fournissez des exemples de votre code.

Tout ce qui précède présente à quel point les données étaient déjà presque prêtes à l'emploi. Ainsi, à l'exception de la suppression de quelques colonnes, aucun nettoyage n'a été effectué sur les données.

3 Feature engineering

3.1 Caractéristiques pertinentes

Quelles caractéristiques avez-vous sélectionnées comme les plus pertinentes pour votre modèle ? Décrivez votre processus de sélection.

Les caractéristiques pertinentes pour notre tâche sont les données météorologiques, les données calendaires qui rendent compte des **schémas cycliques** de notre calendrier et des données spécifiques au type de jour (travaillé/repos, vacances). J'ai déjà quelque peu développé plus haut, mais les attributs `season`, `mnth`, `weekday`, `yr` (et `hr` pour `hour.csv`) suffisent à donner une bonne représentation du placement des échantillons dans une chronologie. Les attributs `holiday` et `workingday` permettent d'ajouter une couche d'information importante sur les jours et de venir compléter les données calendaires. Les données météorologiques sont évidemment incontournables.

En dehors de ces attributs, tout a été écarté du dataset d'entraînement. L'attribut `instant` est supprimé car redondant avec l'indexage automatique des dataframe `pandas`, l'attribut `dteday` est supprimé car redondant avec les autres informations calendaires. Enfin, les attributs `casual` et `registered` font partie de la réponse et ne peuvent pas être utilisés pour la tâche de régression du nombre de vélos loués par échantillon. Ils pourraient servir pour apprendre à prédire des métriques plus spécifiques mais ce n'est pas notre but ici.

3.2 Nouvelles caractéristiques

Avez-vous créé de nouvelles caractéristiques ? Si oui, expliquez ce que vous avez créé et pourquoi.

Les premières caractéristiques que j'ai décidé d'ajouter sont des caractéristiques polynomiales en fonction des données numériques météo. J'ai ajouté des polynômes de degré 2, ce qui rajoute donc 6 attributs. J'ai déjà eu l'occasion de travailler sur des données météorologiques en Machine Learning, et j'ai pu observer que la plupart des métriques météorologiques étaient en réalité des combinaisons polynomiales de données basiques. Il m'a semblé naturel d'essayer de recréer des combinaisons de manière similaire sur ces données.

Ensuite, j'ai voulu rajouter la moyenne glissante sur les 7 derniers jours du nombre de location de vélos (ou les dernières 24 heures), afin d'ajouter des données plus tendanciennes. Cependant cela empêchera l'utilisation du modèle final à des fins de prédiction sur le moyen terme puisqu'il faudra toujours fournir au modèle les valeurs des jours d'avant. On testera donc des modèles avec et sans ces nouvelles caractéristiques. Il est toutefois important de se souvenir que les données ne correspondent pas forcément à 2 ans de récolte sans aucun trou, les 7 échantillons précédents un point de données ne sont pas automatiquement les 7 jours qui le précèdent. De même pour les échantillons de `hour.csv` (où la moyenne est faite sur 24 valeurs. Aussi, il faut au minimum 1 valeur précédente disponible pour que cette attribut soit rempli. En cas de manque de données (moins que 7 ou moins que 24), les données disponibles sont tout de même utilisées (mais la moyenne se fait sur moins que 7 jours/24h).

3.3 Encodage des catégories/Transformation et mise à l'échelle des caractéristiques

Comment avez-vous encodé les variables catégoriques (par exemple, one-hot encoding, label encoding) ? Justifiez votre choix.

Avez-vous transformé ou mis à l'échelle certaines caractéristiques ? Pourquoi cela était-il nécessaire pour votre projet ?

Comme expliqué précédemment, je n'ai pas eu à mettre à l'échelle aucune données puisqu'elles étaient toutes normalisées et encodées d'une manière qui me convenait bien. Si j'ai au départ envisagé de normaliser les attributs `casual` et `registered`, je me suis vite rendu compte qu'ils constituaient une partie de la réponse que notre modèle était censé prédire et donc que ces attributs n'avaient pas leur place dans les données d'entraînement. Ils ont donc été écartés et le problème de leur normalisation ne se posait plus.

Concernant ces deux colonnes, j'ai un moment pensé à les utiliser pour raffiner les prédictions, et faire prédire au modèle le nombre d'utilisateurs occasionnels relativement au nombre d'abonnés enregistrés au programme de location. Hors pour que cela soit utile il aurait fallu posséder le nombre d'abonnées à la date de chaque échantillon en addition du nombre de vélos loués par des personnes abonnées, et le dataset ne disposait que de cet dernier attribut sans préciser le nombre total d'abonnements.

4 Model Selection

4.1 Modèles testés

Quels modèles de machine learning avez-vous testés et pourquoi avez-vous choisi ces modèles ?

Les modèles testés sont :

- Régression Linéaire
- KNN
- Arbre de décision
- Random Forest
- Bagging (Bootstrap Aggregating)
- Boosting

J'ai choisi ces modèles car ce sont des modèles simples de Machine Learning, qui s'adaptent bien à une tâche de regression (des algorithmes tels que Naive Bayes, K-Means ou K-Medoids se prêtent très bien à de la classification mais moins de la régression par exemple). Ce sont donc des algorithmes variés avec des forces et faiblesses différentes, il est donc probable d'en trouver au moins qui soit suffisamment prometteur pour notre projet.

L'idée était de tester ces modèles sans **fine-tuning** afin de voir les architectures prometteuses sans fabriquer un modèle robuste et précis dès à présent. Aussi, en cas d'architecture prometteuse, plusieurs variations de cette architecture seraient ensuite envisagée. Par exemple pour les algorithmes de **Boosting**, qui ont plutôt bien performé dans les tests, plusieurs modèles ont été envisagés comme AdaBoost, XGBoost ou encore Stochastic Gradient Boosting.

Voici les performances des modèles sans aucune paramétrisation spéciale :

day.csv :

- Régression Linéaire : $RMSE = 635.88$, $R^2 = 0.89$. Performance solide pour un modèle non affiné. Le R^2 est bon même si la $RMSE$ reste encore trop élevée. Il semble que les données et leurs caractéristiques non linéaires ajoutées soient bien capturées par le modèle. On conserve ce modèle pour la suite.

- **KNN** : $RMSE = 931.62$, $R^2 = 0.76$. $RMSE$ trop élevée malgré un R^2 qui plutôt acceptable. Les données non linéaires rajoutées semblent augmenter la dimensionalité du dataset dans une mesure trop importante et les algorithmes de plus proches voisins ont souvent du mal en grande dimension. Ce modèle est écarté pour la suite du projet.
- **Arbre de décision** : $RMSE = 922.94$, $R^2 = 0.76$. Mêmes ordres de valeurs que les plus proches voisins, ce genre de modèle tend à overfit facilement. Il serait intéressant d'affiner ce modèle pour tester ses performances lorsque l'overfit est mitigé, mais les meilleures performances d'autres modèles suggèrent que les arbres de décisions ne sont pas dans les meilleurs modèles ici. On écarte ce modèle pour la suite.
- **Random Forest** : $RMSE = 678.20$, $R^2 = 0.87$. Deux valeurs correctes pour un algorithme non affiné. Par rapport à un arbre seul, l'overfitting est réduit en mélangeant les résultats d'un grand nombre d'arbres. On ne conserve pas ce modèle pour la suite car 2 autres sont meilleurs à première vue. Il en s'agit pas d'un choix totalement éclairé et justifié mais il faut choisir, et j'ai envie de voir jusqu'où une regression linéaire peut aller.
- **Bagging** : $RMSE = 706.54$, $R^2 = 0.86$. Ce modèle est une version similaire aux **Random Forest**, en légèrement moins capable de généralisation ici. On l'écarte.
- **Boosting** : $RMSE = 645.10$, $R^2 = 0.88$. Deuxième meilleur performeur sur le dataset des jours, les ajustements en cascade caractéristiques d'un modèle de **Boosting** semblent avoir plutôt bien intégré les données non-linéaires, ce qui fait ressortir ses performances par rapport aux autres. On garde ce modèle pour la suite.

hour.csv :

- **Régression Linéaire** : $RMSE = 139.92$, $R^2 = 0.41$. Performances moyennes, les données horaires semblent devenir trop compliquées pour être expliquées efficacement par un simple modèle linéaire avec ajout d'attributs quadratiques. On écarte ce modèle pour la suite.
- **KNN** : $RMSE = 91.99$, $R^2 = 0.74$. Performances correctes, la $RMSE$ est haute, mais le R^2 est acceptable. On écarte cependant le modèle de par les performances meilleures des autres modèles.
- **Arbre de décision** : $RMSE = 60.04$, $R^2 = 0.89$. Cette fois-ci l'arbre n'a pas autant d'imprécision qu'avec l'autre dataset. On ne garde pas ce modèle pour la suite même s'il aurait été intéressant de voir si une gestion de l'overfitting potentiel permet d'augmenter significativement les performances.
- **Random Forest** : $RMSE = 42.46$, $R^2 = 0.95$. C'est le meilleur modèle, le R^2 est bon et la $RMSE$ est au plus bas de cette expérience. On garde ce modèle.
- **Bagging** : $RMSE = 45.27$, $R^2 = 0.94$. Très bon mais moins que **Random Forest** pour un principe similaire, on ne le garde pas pour ne pas faire doublon.
- **Boosting** : $RMSE = 66.66$, $R^2 = 0.87$. Semblent capturer plutôt bien les relations complexes, ses métriques sont bonnes. On le garde également.

N.B : Ces tests ont été faits par une **K-Folds cross-validation** à 5 splits, avec 20% des données dans le test set.

N.B : Ces tests ont aussi été menés sur des datasets où les données glissantes sur 7 jours ou 24 heures avaient été enlevées car j'avais un gros doute sur l'utilité de ces nouvelles colonnes et je me demandais si ça ne pouvait pas même pénaliser l'apprentissage de nos modèles. Sans rentrer dans les détails, les résultats étaient légèrement moins bons sur les 2 datasets en enlevant les colonnes, de l'ordre de quelques dixièmes pour le R^2 et de quelques dizaines pour la $RMSE$.

Seuls 1 modèle sur chaque dataset avait de très légères meilleures performances mais non significatives. Les colonnes ont donc été laissées pour la suite.

4.2 Paramètres et Hyperparamètres

Décrivez les hyperparamètres des modèles que vous avez entraînés. Quels paramètres avez-vous ajustés et comment avez-vous décidé de leurs valeurs ?

J'ai choisi les modèles suivants : **GBDT** (Gradient Boosting Decision Tree) et **Linear-Regression** pour le dataset `day.csv`, et **RandomForestRegressor** et **LightGBM** pour le dataset `hour.csv`. La raison pour laquelle j'ai changé l'algorithme de Boosting d'un dataset à l'autre est le fait que LightBGM est censé mieux gérer les grands datasets grâce à sa rapidité. `hour.csv` est un plus gros dataset que `day.csv` mais je doute que 17000 lignes soient considérées comme un gros dataset. Je voulais cependant tester la différence entre ces deux implémentations.

Au niveau des méthodes de recherches des meilleurs **hyperparamètres**, on utilisera une **GridSearch** pour la Regression Linéaire et la Random Forest, et des **RandomSearch** pour les modèles de Boosting. J'aurais bien aimé essayer d'utiliser de l'**optimisation bayésienne**, mais je ne maîtrise pas suffisamment cette technique pour la mettre en oeuvre dans un temps raisonnable.

D'après la FIG.19, les coefficients de la régression linéaire (dans une double validation croisée) ont des amplitudes très variées (au passage on remarque que la nouvelle feature des valeurs glissantes ne sert à rien pour ce modèle). On va donc régulariser ce modèle avec une **Regression Ridge**, qui pénalise par rapport au carré des amplitudes, et comme on le voit sur la FIG.19 les attributs à gros coefficients sont majoritairement ceux qui sont des carrés d'attributs initiaux.

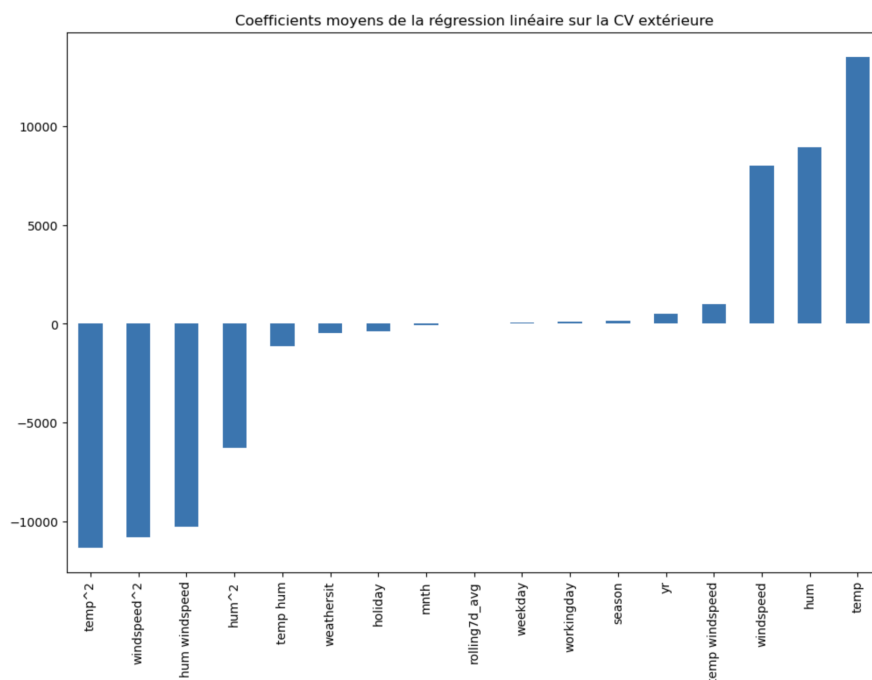


FIGURE 19 – Coefficients moyens de la régression linéaire sur plusieurs folds de cross-validation

Ainsi pour la Régression Linéaire on va affiner l'hyperparamètre **alpha** qui dirige la force de régularisation.

Pour la Random Forest on va affiner le **nombre d'estimateurs**, la **profondeur maximale** des arbres ainsi que le nombre **minimum de splits** pour créer un nouveau noeud dans un arbre.

Pour les algorithmes de Boosting, on va affiner les mêmes hyperparamètres mais pour des valeurs différentes (j'ai déjà passé un tour de nested-CV et j'ai adapté pour voir dans quelles intervalles ces hyperparamètres avaient tendance à se déplacer). On va ajuster la **profondeur maximale**, la **vitesse d'apprentissage** et le **nombre d'estimateurs**.

Comme dit juste au dessus, j'ai d'abord fait un premier tour en **nested-CV** et j'ai recentré les valeurs des paramètres à tester en fonction des valeurs desquelles se rapprochaient les meilleurs modèles. J'ai ensuite refait un passage en étoffant un peu plus.

Ainsi j'obtiens mes hyperparamètres pour les modèles que j'ai choisi. J'en profite pour restreindre leur nombre à 1 par dataset. J'en avais volontairement gardés 2 par dataset pour vérifier certaines hypothèses, mais le but est d'implémenter un seul modèle donc je garde le boosting pour les deux datasets, car ils performant mieux que leurs concurrents. La régression linéaire est écartée également car sa **learning curve** montre des trajectoires pour l'erreur d'entraînement (qui augmente) et l'erreur de validation (qui ne diminue pas vraiment) qui semblent indiquer un underfitting (voir FIG.20).

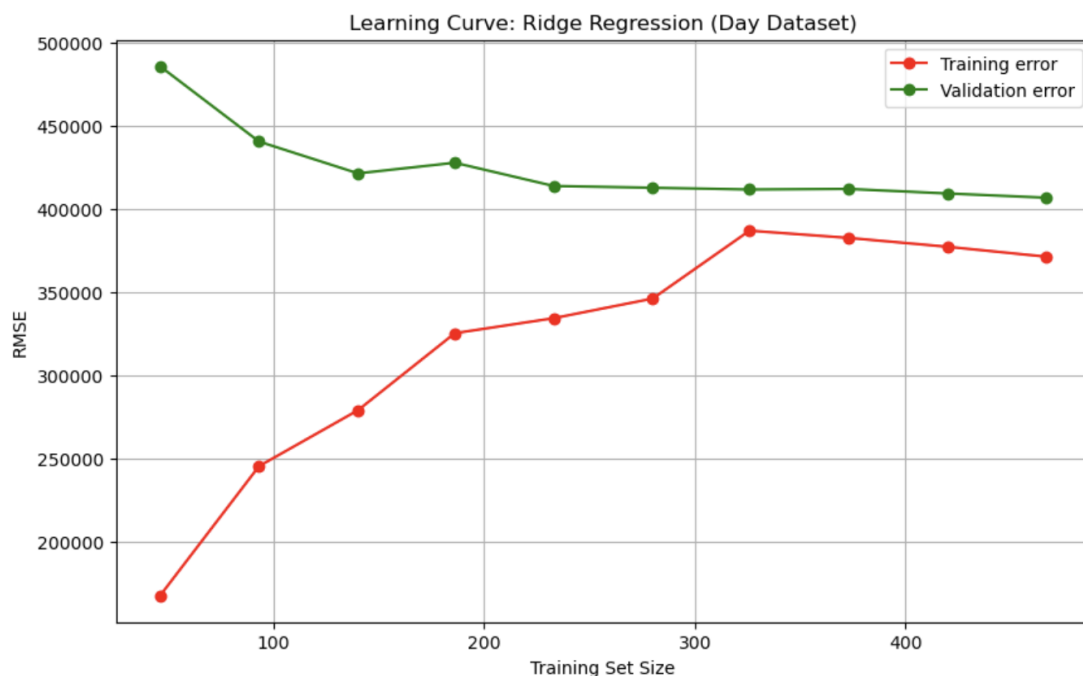


FIGURE 20 – Learning et validation curves de la Régression Linéaire

À l'inverse le boosting sur le dataset des heures semble indiquer de l'overfitting, comme il s'agissait de le vérifier. Les paramètres donnés par l'affinage ne permettent pas au modèle de bien généraliser sur des données non vues (FIG.21). Je dois cependant avouer que je ne

comprends pas quelle est l'unité de la RMSE ici. C'est normalement homogène à un nombre de location, mais que ce soit pour la régression Ridge ou le LGBM, les valeurs sont tellement grandes qu'elles ne rentrent même pas dans le cadre d'un prédicteur Dummy qui prédirait la moyenne de toutes les valeurs.

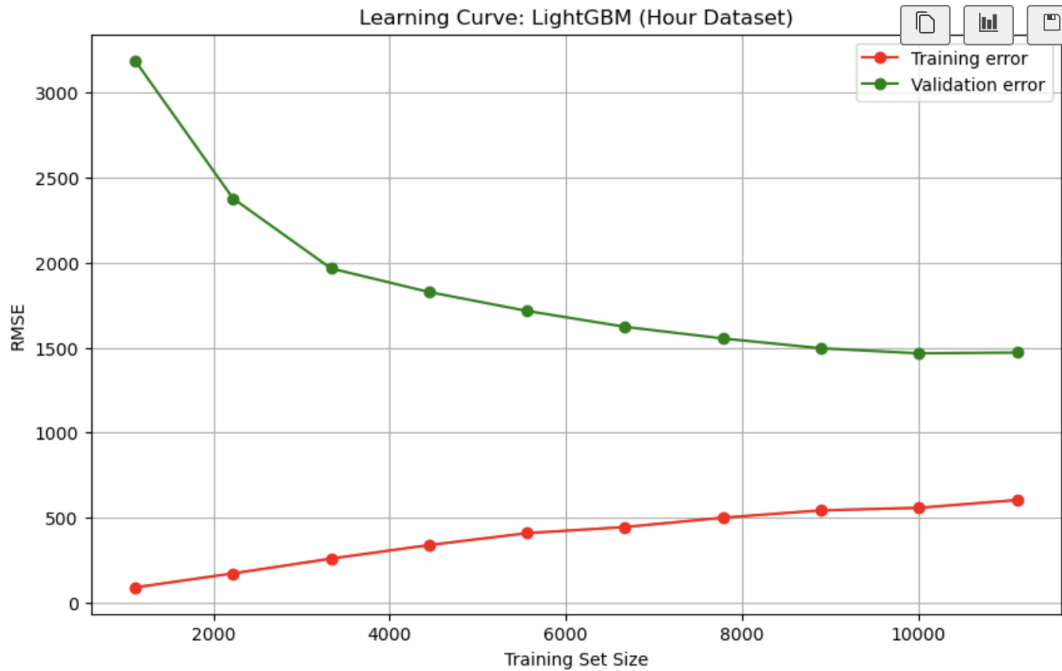


FIGURE 21 – Learning et validation curves de l'algorithme LightGBM

De même pour l'algorithme de Gradient Boosting, il semble overfit mais l'échelle de valeurs de la RMSE est très étrange (voir FIG.22).

4.3 Modèle supplémentaire

Utilisez un modèle de machine learning qui n'a pas été vu dans le MOOC. Expliquez comment il fonctionne et comparez ses performances avec les autres modèles que vous avez utilisés.

Dans cette partie, j'ai choisi d'entraîner un **SVR** (Support Vector Machine for Regression) comme modèle additionnel. C'est une implémentation des machines à vecteurs de support pour la régression. Il fonctionne sur le même principe que la régression linéaire, à ceci près qu'il cherche à caser un maximum de données d'entraînement dans une marge ε proche de son hyperplan (une droite en dimension n). Tant qu'un échantillon se trouve à l'intérieur de la marge, on ne considère pas de son erreur. C'est une sorte de régularisation car ça permet de simplifier le modèle en ne se préoccupant pas d'erreurs minimales.

La fonction de coût à minimiser est donc de la forme :

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n (\xi_i + \xi_i^*) \quad (1)$$

Avec ε la marge de que l'on se donne (hyperparamètre), C le paramètre de régularisation, w les

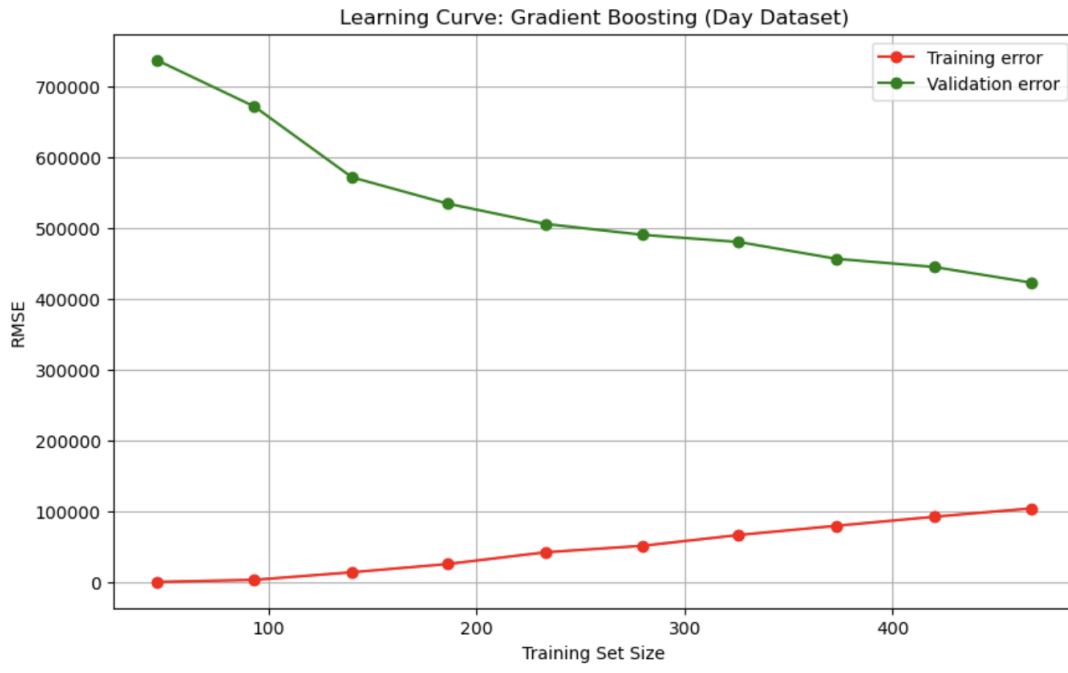


FIGURE 22 – Learning et validation curves de l’algorithme de Gradient Boosting sur day.csv

pois de la régression et ξ et ξ^* les erreurs d’un point (ξ si l’erreur est au dessus, ξ^* si l’erreur est en dessous).

L’équation de la régression est la simple :

$$f(x) = w^T x + b \quad (2)$$

À noter que puisque c’est un algorithme basé sur des calculs de distances (comme dans un KNN par exemple), la normalisation des données est ici primordiale. C’est le cas de nos données et ce dès le départ, c’est une des raisons qui m’a fait choisir ce modèle en tant que modèle supplémentaire (et aussi parce qu’en terme de modèles pas trop compliqués que l’on n’a pas vu en cours il ne reste plus beaucoup de choix).

Pour les hyperparamètres, j’affine **C** et **gamma**. Gamma est un paramètre du noyau. Sans trop rentrer dans les détails de ce qu’est un noyau ici, c’est une autre manière de d’augmenter les données avec des features non linéaires. Ça nous permettra de voir si les **Polynomial Features** rajoutées dans le preprocessing suffisaient ou non. Ce genre d’amélioration nécessite un paramètre **gamma**.

Afin de limiter la durée de l’affinage, j’utilise le théorème de Vapnik et les définitions suivantes pour trouver la valeur de **gamma** :

Théorème 1 *Si la dimension VC d’un modèle est une petite fraction du nombre de données N , alors*

$$\text{erreur_test} \leq \text{erreur_train} + \sqrt{\frac{VC}{N}} \quad (3)$$

Définition 1 la dimension VC d'un modèle est le nombre de point maximal que ce modèle peut pulvériser, i.e. qu'il peut mathématiquement séparer de manière parfaite peu importe la configuration initiale des points.

Théorème 2 La dimension VC d'un SVR pour un jeu de données de m attributs est $VC = m + 1$.

Théorème 3 Corrolaire du théorème de Vapnik : pour N points de données et une dimension VC donnée :

$$\varepsilon = \sqrt{\frac{VC}{N}} \quad (4)$$

Pour le dataset des jours, on trouve $\varepsilon \approx 0.15$ ($VC=17$, $N=731$) et pour le dataset des heures on trouve $\varepsilon \approx 0.03$ ($VC=18$, $N=17379$).

Je choisis également un kernel `poly`, qui est un kernel simple, je n'ai pas envie de rajouter trop de complexité puisque j'ai déjà ajouté des features polynomiales en début de projet.

TABLE 1 – Caractéristiques des SVR

	day.csv	hour.csv
kernel	rbf	rbf
C	100	100
ε	0.15	0.03
γ	0.1	0.1
RMSE	1271.77	50.89
R^2	0.596	0.918

Je ne vais pas inventer des résultats là où je n'ai que des résultats préliminaires, cela fait 5 jours (de lundi 6 janvier après-midi à samedi 11 janvier soir) que je n'arrive pas à faire une cross-validation potable pour un SVR avec un kernel `poly`. Ça tourne en dans le vide pendant des heures. J'ai même laissé tourner une nuit entière en me disant que ça serait largement suffisant mais toujours rien 10 heures plus tard. Je n'avais plus que ça à faire en dehors de ce que j'ai écrit dans la section 6.1, mais 5 jours n'ont pas suffi à sortir quelque chose qui me satisfasse. Je pense donc qu'il y a une problème avec mon code mais je en trouve pas quoi (je n'ai pas d'erreur pour m'orienter), je ferai ça plus tard par curiosité après le rendu. Je me suis donc résolu en dernier recours à utiliser un kernel `rbf` même si je pense qu'un kernel `poly` aurait pu être beaucoup mieux (explications en fin de section). Rbf peut être plutôt correct aussi mais je ne vois qu'un faible intérêt à garder toujours les paramètres par défaut, surtout quand les tests préliminaires ont montré qu'un SVR Vanilla obtenait un R^2 de 0.15 et une RMSE de 1700 (resp. 150) pour `day.csv` (resp. `hour.csv`).

Sur le `day.csv` : On peut voir que la variance expliquée est moyenne, tout comme la RMSE qui est le double de notre modèle ensembliste final. Je ne sais pas si un kernel différent aurait

suffit à améliorer significativement, mais c'est une option à tester.

Sur le `hour.csv` : Ici les résultats sont déjà bien plus encourageants. La RMSE est à 50, qui est plutôt proche des 38 de notre modèle final, avec une variance expliquée supérieure à 0.90 ce qui est plutôt satisfaisant. À noter que $\varepsilon = 0.03$ n'est pas ce qui est donné par la cross-validation croisée (qui propose 0.1), mais en entraînant un modèle sur chaque valeur, les résultats ne diffèrent presque pas. J'ai gardé 0.03 pour corroborer la théorie, mais si les résultats avaient été significativement moindres pour 0.03, je n'aurais pas eu le temps (ni la capacité) de chercher une explication théorique à pourquoi cela ne fonctionne pas.

Pour revenir sur pourquoi je pense qu'un kernel poly serait meilleur qu'un kernel rbf (radial basis function), les données calendaires se prêtent par nature à une modélisation polynomiale (sur un intervalle de temps comme le nôtre, ces schémas sinusoïdaux sont approximables par des polynômes). On peut le voir FIG.4 par exemple. Ainsi un kernel polynomial se prête parfaitement à cette modélisation, et j'ai peur que le kernel rbf overfit un peu en intégrant des relations trop complexes. Ce genre de relations polynomiales se retrouvent aussi d'après moi dans les relations de type `cnt` en fonction de `temp`, où les locations atteignent un pic à une certaine température et diminuent lorsqu'il fait trop froid ou trop chaud.

4.4 Gestion des données déséquilibré

Comment avez-vous géré les jeux de données déséquilibrés (si applicable) ?

Comme présenté plus haut, certaines features présentent des déséquilibres. Cependant ça ne concerne pas la target, puisque mon dataset s'applique à une tâche de régression (du moins dans ce que j'ai choisi d'en faire). Je ne peux pas augmenter les données à l'aide d'un `RandomOversampling` ou d'un algorithme type `SMOTE/SMOTE-NC`. Je pourrais augmenter manuellement les groupes sous représentés, mais ça me prendrait beaucoup de temps alors je vais adopter une position plus simple pour l'attribut `weathersit` et concaténer les classes 3 et 4, qui représenteront donc les situations météo difficiles, avec moins de nuances. Cela permettra d'éviter aussi que le dataset des jours ne manque complètement la classe 4 de cet attribut. Jje n'ai pas testé si l'absence totale d'une classe dans un dataset posait problème si on la reconstruit lorsque le modèle est déployé (ici je ne pense pas car les données sont encodées ordinalement, donc aucune erreur ne se lèverait, mais les précisions du modèle sur de tels samples pourraient être affectées). Cela permettra de réduire un peu le déséquilibre même si c'est très marginal voire inutile (la classe 4 compte 4 élément dans `hour.csv` et aucun dans `day.csv`).

De l'autre côté, pour `holiday` et `workingday` (qui sont des attributs binaires) je ne peux pas faire ça. Je ne vais donc pas y toucher car je ne sais pas comment faire autrement que rajouter des données synthétiques, et je manque d'idées pour cela.

4.5 Division en train/validation/test

Comment avez-vous divisé le jeu de données en ensembles d'entraînement, de validation et de test ? Fournissez une justification pour votre choix.

J'ai opté pour des cross-validations imbriquées afin de pouvoir pratiquer plusieurs Search sur différents sous ensemble des données. Pour chaque modèle à entraîner, j'ai divisé le dataset utilisé en 5 folds pour réaliser une cross-validation externe. Chaque **test set** comprend donc 20% des données (1 fold sur les 5). Ensuite pour chacun de ces folds, j'ai effectué l'affinage des hyperparamètres au travers d'une cross-validation interne en 3 folds sur une **GridSearch** ou une **RandomSearch**. Pour un modèle précis, on effectue donc 3 GridSearch différentes avec à chaque fois 2/3 des données pour entraîner, c'est donc le **train set** et 1/3 des données pour tester ces paramètres là, c'est donc le **validation set**.

Diviser le dataset ainsi en 3 types de sets de données permet d'éviter les fuites de données vers le modèle, qui si on l'entraînait uniquement avec train et test sets en cross-validation pourrait faire fuiter des données d'un test set dans le modèle (puisque chaque test set est également utilisé dans une autre itération dans le train set) et donc biaiser les scores de test puisque le modèle aurait déjà pu apprendre les données de test qui lui sont présentées. À l'inverse ici, lorsque la search se termine, on teste chaque instance du modèle sur les données test que l'on a séparé dans la cross-validation externe, qui sont des données que les modèles n'ont jamais vu.

5 Évaluation

5.1 Métriques

Quelles métriques avez-vous utilisées pour évaluer votre modèle (par exemple, précision, rappel, F1, RMSE) ? Pourquoi avez-vous choisi ces métriques ?

Pour l'évaluation des modèles, j'ai choisi d'utiliser conjointement le coefficient de détermination R^2 et la **RMSE** (Racine de l'Erreur Moyenne Quadratique). Le R^2 mesure la proportion de la variance de **target** qui est expliquée par le modèle (il a une valeur maximale de 1). Plus simplement, plus il est élevé et proche de 1, plus le modèle permet d'expliquer la variabilité des données, et donc est un bon modèle. Mais cette métrique ne permet pas de donner une idée d'à quel point le modèle se trompe point par point, c'est pourquoi j'ai choisi d'utiliser également la RMSE. La RMSE est homogène à l'unité de **target** et mesure l'éloignement moyen de chaque prédiction du modèle à sa vraie valeur. Par rapport à la **MAE** (Erreur Moyenne Absolue), le carré permet d'amplifier les grosses erreurs de prédiction. Un des problèmes est qu'il n'est pas possible de voir si notre modèle sous-estime ou sur-estime ses prédictions. Mais l'association de ces deux métriques est déjà suffisante et je ne vais pas en rajouter une troisième.

5.2 Étude d'ablation

Réalisez une étude d'ablation en supprimant une ou plusieurs caractéristiques. Comment cela a-t-il affecté les performances de votre modèle ? Que révèle cela sur l'importance de ces caractéristiques ?

Première ablation : valeurs glissantes

Le premier attribut que j'ai enlevé est les données glissantes que j'avais moi-même rajouté.

J'avais un doute sur leur utilité, et on voit lors des tests des deux algorithmes de boosting les choses suivantes :

- `day.csv` : La RMSE de test passe de 650 à 611, ce qui montre que le modèle avait sur-appris à cause de cet attribut. Les résultats sont meilleurs mais toujours pas à la hauteur (on en discutera plus tard).
- `hour.csv` : La RMSE de test reste plutôt constante en dessous de 40 (environ 38). Il n'y a pas d'amélioration notable. Ainsi, quitte à choisir, je choisirais de ne pas garder cet attribut car il n'apporte rien de significatif. Ainsi on ne garde pas ces colonnes de pour les ablations suivantes car elles pourraient fausser leur intérêt.

Deuxième ablation : données météorologiques Ensuite j'ai voulu supprimer les données météorologiques afin de voir ce que si des schémas calendaires pouvaient suffire à dégager un bon ordre de grandeur de prédiction.

- `day.csv` : La RMSE augmente à 1066, ça devient vraiment problématique comme scores.
- `hour.csv` : La RMSE augmente à 58, c'est également problématique compte tenu de l'intervalle de valeur de `cnt` sur le dataset des heures (moyenne à 190).

Troisième ablation : données calendaires Enfin, j'ai fait l'inverse et retiré les données calendaires pour voir si les données météo permettent d'obtenir de bons résultats à elles seules. Je m'attends à ce que les résultats soient encore pire que sur les données calendaires uniquement, car je suppose que les gens ne louent pas de vélos en semaine, même quand il fait beau (c'est une intuition grossière).

- `day.csv` : La RMSE explose à 1511, le modèle ne sert donc plus à rien (déjà à +1000 d'ailleurs). Le modèle fait à peine mieux qu'un modèle Dummy (qui a une RMSE de 2000).
- `hour.csv` : La RMSE monte à 150, ce qui est presque tout aussi mauvais que les 178 du Dummy.

TABLE 2 – RMSE pour les différentes études d'ablation

Données supprimées	<code>day.csv</code>	<code>hour.csv</code>
Valeurs glissantes	611	38
Données météo	1066	58
Données calendaires	1511	150

En conclusion de cette petite étude d'ablation, on confirme que l'entièreté du dataset est utile, ce qui était prévisible vu la taille de celui-ci. Il est grossièrement constitué de 2 types de métriques (calendaires et météo) et le but de ce projet est justement de tirer le maximum de cette association, alors il n'est pas surprenant de voir qu'en retirer la moitié ne donne pas de bonnes performances.

5.3 Analyse des erreurs

Effectuez une analyse des erreurs. Analysez les cas où le modèle a mal performé et décrivez les raisons possibles des erreurs.

A propos du modèle sur `hour.csv`, le choix du boosting a été une erreur, le fait de se rendre compte après qu'il overfit probablement un peu fait réfléchir sur la pertinence. Tout le rapport est déjà écrit mais si c'était à refaire je prendrai probablement la Random Forest à la place.

A propos du modèle sur `day.csv`, les résultats ne sont pas acceptables pour une véritable mise en production. Je ne sais pas précisément les étapes les plus significatives qui pourraient être mieux faites sur ce modèle, mais je crois qu'il y eu une accumulation de petites erreurs tout au long du processus, comme sur la feature engineering, repérer d'autres relations entre les features, mieux mitiger l'overfitting et diversifier les mesures de performances. Je ne sais pas identifier lesquelles sont plus importantes, c'est une suite d'imprécisions qui coûte à la fin sur les performances des modèles.

5.4 Cross-validation

Avez-vous effectué une validation croisée ? Si oui, quelle méthode avez-vous utilisée et quelles informations en avez-vous tirées ?

Question déjà traitée dans la sous-section 4.5. La nested cross-validation m'a permis de tester un grand nombre de combinaisons d'hyperparamètres pour chacun des modèles tout en garantissant une étanchéité dans les informations apprises par ces modèles. C'est plus technique à mettre en oeuvre (j'ai écrit une fonction que j'appelle pour chaque modèle à entraîner, mais elle m'a donné du fil à retordre) mais bien plus pratique et surtout fiable sur l'apprentissage par rapport à des bidouilles copiées collées pour les faire sur plusieurs modèles.

En ce qui concerne la stratégie de **cross-validation**, je n'ai pas mélangé les données lors des splits (en utilisant `'Shuffle=True'`) car je pensais que ça aurait pu révéler des problèmes similaires à ceux des séries temporelles. Au final je ne pense pas que ce soit le cas car j'ai enlevé les dates et les index qui conservaient la chronologie, mais également car on ne loue pas un vélo parce qu'il a fait beau la veille (d'où mon scepticisme face à la colonne de valeurs glissantes sur `7j/24h`, je pense que ça a une importance minime dans l'apprentissage, comme on a pu partiellement le vérifier sur la régression linéaire).

5.5 Comparaison des performances

Comparez les performances de votre modèle et pipeline avec un ou plusieurs modèles de base (dont le modèle aléatoire). Quelle était ces modèles de base et combien votre modèle a-t-il amélioré les résultats ?

J'avais fait des petits tests en début de projet pour voir quels modèles j'allais pouvoir utiliser (voir section 4.1), mais j'en ai refait après coup, notamment pour intégrer le modèle **Dummy**.

Pour chaque dataset, la meilleur performance est écrite en gras, la deuxième meilleure est soulignée.

Ces résultats montrent que les modèles entraînés améliorent les prédictions par rapport aux modèles baseline. Évidemment ceux-ci ne sont pas affinés mais cela montre que les modèles de boosting choisis sont préférables aux modèles de base. Les performances sont meilleurs que les modèles de bases d'au moins 118 unités sur `day.csv` et de 6 unités sur `hour.csv`. Ainsi, sans compter **Dummy** qui ne compte pas vraiment comme un candidat, les modèles entraînés sont meilleurs de 19% à 209% sur `day.csv`, et de 16% à 367% sur `hour.csv`.

En terme de pourcentage d'erreur moyen (MAPE, Mean Absolute Percentage Error), le modèle sur `day.csv` a une MAPE de 1.05, ce qui est très mauvais, et le modèle sur `hour.csv` a une MAPE de

TABLE 3 – Comparaison des **RMSE** des modèles baseline et des modèles affinés

Modèle	day.csv	hour.csv
Dummy Regressor	2022.17	178.03
Linear Regression	758.50	140.82
K-Nearest Neighbors	1008.94	57.76
Decision Tree	944.39	63.17
Random Forest	<u>727.70</u>	<u>44.68</u>
Bagging	739.56	47.04
SVR	1271.77	50.89
GradientBoosting	611.72	-
LightGBM	-	38.37

0.36, ce qui est raisonnable. Cette métrique diminue grandement mon appréciation des résultats, ainsi que la fiabilité des modèles obtenus.

N.B. : Ces valeurs ont été calculées après avoir décidé de retirer la feature des valeurs glissantes de locations. Autrement dit les valeurs pour ces modèles présentées en section 4.1 diffèrent car elles ont été calculées avec les données glissantes. Je ne les ai pas mises à jour car ce sont celles qui m'ont permis de faire mes choix de modèles. Une autre chose à noter est la suivante : pour le dataset des heures, le boosting n'est pas le meilleur modèle, il faisait néanmoins parti des meilleurs et s'est révélé le meilleur lors de l'affinage des hyperparamètres. J'ai discuté plus longuement de ce point dans la section 5.3.

6 Discussion et conclusion

6.1 Parce que je sais pas noter mes devoirs

Choses que j'avais prévu de faire mais que je n'ai pas fait parce qu'il faut finir et que je pensais que j'avais une semaine de plus.

- Faire un calcul de **features importance** sur **lightGBM** et **Random Forest**, ça m'aurait permis de voir quels attributs contribuent le plus à la précision des modèles et donc mieux gérer certains attributs inutiles ou qui prennent trop l'ascendant.
- Évaluer la précision des modèles avec des métriques moins sensibles aux outliers (comme la médiane), ça m'aurait permis de mieux comprendre l'importance de ces valeurs extrêmes dans le modèle.
- Analyser les données des attributs **casual** et **registered** pour voir si on peut tirer des informations de leurs distributions et de leur corrélation avec la target **cnt**.
- Évaluer plus sérieusement qu'avec un simple graphe des poids de régression linéaire l'importance des données glissantes de location pour vérifier si elles servent à quelque chose ou non. Les garder alors qu'elles ne servent à rien peut diminuer les performances du modèle.
- Donner plus de détails et d'analyses sur mes choix et ce qu'il serait possible de faire autrement que moi.
- Rendre le projet à l'heure.

6.2 Défis importants

Quels ont été les défis les plus importants que vous avez rencontrés pendant le projet et comment les avez-vous surmontés ?

Durant ce projet, le principale défi que j'ai eu à rencontrer est probablement la gestion des différents modèles. Lorsqu'il n'y a qu'un modèle à tester (qu'on l'a déjà défini à l'avance ou bien qu'on nous demande explicitement de l'utiliser) la marche à suivre est presque directe, mais lorsqu'il faut tester plusieurs modèles on se perd vite, que ce soit dans le code qui est plus fourni et où on ne sait plus quels variables vont pour quels modèles, alors on rallonge les nombres de ces variables mais on les écrit mal, ou tout simplement qu'on essaie de faire un beau code bien optimisé et pratique et que l'on se perd à écrire des fonctions réutilisables alors qu'on aurait tout simplement pu coller 4 blocs de code à la suite et ça aurait tout aussi bien marché.

D'un point de vue données, comme prévu l'apparente simplicité du dataset m'a poussé à vouloir rajouter des features pour l'enrichir, et surtout me poser la question de si elles sont vraiment utiles ou non, de si j'ai bien calculé ces nouvelles features et si je ne suis pas en train de tout détériorer à vouloir rajouter une feature que j'ai trouvé intéressante au premier abord.

Cependant, ces difficultés finissent par s'estomper, notamment avec l'habitude du projet qui fait que l'on s'y retrouve mieux.

Une dernière légère difficulté a été l'utilisation de modèles que je n'avais jamais vraiment utilisé sérieusement avant, et qui ralentissaient mes avancées par rapport aux modèles que je maîtrisais. C'était un peu un projet à 2 vitesses de ce point de vue là même si c'était tout sauf handicapant.

6.3 Fiabilité

Sur la base de vos résultats, à quel point votre modèle est-il fiable pour la tâche de prédiction ? Quelles limitations présente-t-il ?

Au niveau de la fiabilité, les déséquilibres de certaines classes dans les features peuvent entraîner des imprécisions plus accentuées lorsque de telles classes se retrouveront dans des données fournies pour une prédiction. Les attributs `workingday` et `holiday` ont une importance dans le compte total de vélos loués : il semble que beaucoup plus de vélos sont loués les jours travaillés (qui sont donc des jours de non vacances). Ne pas réussir à équilibrer les données d'entraînement sur ces attributs rend donc les 2 modèles entraînés moins fiables. Au niveau de `weathersit` également, la quasi absence de classe 4 dans les données limite l'incidence de cet attribut car sa variance est donc moindre. Les prédictions seront donc moins fiables sous des conditions météorologiques compliquées. Cependant, si on imagine vouloir prédire les locations de vélos au jour le jour (ou heure) afin de pouvoir adapter les quantités de vélos dans les stations de location et de ne pas se retrouver à court de matériel, alors des conditions météorologiques difficiles moins bien prédites auraient de plus faible conséquences sur le business puisque l'on loue beaucoup moins de vélo par temps orageux que par grand soleil. Ainsi on a bien moins de risque de se trouver à court de vélos sous ces conditions.

Aussi le modèle ne contient pas tant de samples à des températures élevées, surtout vu l'évolution du climat actuellement et sachant que les données ont été récoltées en 2011/2012. Pour moi c'est une grande limitation du dataset car les jours aux températures élevées vont augmenter en fréquence et en intensité. Aussi on peut relativement douter de la linéarité de l'évolution des locations en fonction de la température. Il semble improbable de voir doubler les locations à 25°C lorsque la température atteindra les 50°C.

Le modèle n'est adapté qu'à la ville d'où proviennent les données. Les habitudes et mentalités des habitants sont très spécifiques à l'endroit où ils vivent et il serait incohérent d'appliquer les prédictions

d'un modèle entraîné sur une ville par exemple américaine (population fortement habituée à la voiture, vivant dans des conditions météo précises) à une ville hollandaise où l'utilisation du vélo est beaucoup plus répandue, et ce même par un temps moins clément que ce sur quoi a été entraîné le modèle.

Le modèle est également limité par la relative simplicité du dataset, qui ne fournit que des informations génériques alors que d'autres attributs (dont je parle plus en détails dans la sous-section suivante) auraient pu venir compléter cette représentation et permettre des résultats plus précis bien que plus spécifiques et moins généralisables à d'autres villes.

6.4 Travaux futurs

Si vous aviez plus de temps ou de ressources, quelles étapes supplémentaires prendriez-vous pour améliorer votre modèle ?

Parmi les choses qui auraient pu être faites ou améliorées, je peux citer :

- Data Analysis : tracer des correlation heatmap entre les attributs numériques, tracer des patterns horaires pour voir l'évolution des locations par heure en fonction d'autres attributs (jours de repos/travail/vacances, saisons), tracer les locations en fonction d'une condition météo à la fois. Cela n'a pas été fait pour ne pas surcharger un rapport déjà conséquent et ne pas passer trop de temps non plus sur la partie analyse du dataset.
- Data Mining : récolter plus de données diversifiées afin de réduire les déséquilibres de classes comme **weathersit** et permettre de mieux répartir les jeux de données train/test/validation selon ces attributs.
- Dataset Augmentation : récolter des données dans d'autres villes, ou alors augmenter la dimensionnalité du dataset sur cette ville. Les patterns de location sont très spécifiques aux habitudes des habitants, ainsi des informations supplémentaires sur la zone et les locations auraient pu aider le modèle à mieux apprendre sur cette ville. Exemples : nombre de personnes abonnées au service (qu'elles louent ou non), superficie de la ville, distance moyenne parcourue par un vélo loué, division de la ville en quartier pour mieux segmenter, ...
- Model Choice : utiliser une approche du dataset sous forme de **Séries Temporelles**, ce qui aurait nécessité des modèles différents tels que **LSTM** ou **ARIMA**. Ce n'est pas un domaine avec lequel je suis très familier et cela ne rentre pas dans le cadre de ce cours, mais il aurait pu être intéressant de tester un autre paradigme d'apprentissage puisque les données peuvent s'y prêter.
- Solution Architecture : utiliser une approche hybride avec des modèles en cascades pour utiliser les deux fichiers du dataset. Prédire les comptes journaliers de vélos loués à l'aide de **day.csv** et ensuite prédire la distribution du compte total de vélo loués par créneau horaire sur la journée. On aurait ainsi eu un modèle global qui aurait pu prédire que tel jour seraient loués N vélos, et pour chaque heure de la journée quelle proportion de N serait louée cette heure là. Cela pourrait potentiellement permettre de révéler des schémas qui sortent du cadre classique de "je loue un vélo à cette heure-ci car il fait beau et chaud", et intégrer des comportements du genre "aujourd'hui il va faire beau et chaud, je loue mon vélo ce matin à 8h alors qu'il fait encore froid pour l'avoir à disposition pour la journée", en prenant donc en compte la distribution de la location sur la journée. Comportement qui pourrait expliquer une partie des vélos loués à de basses températures. Ce point est une supposition et je n'ai pas cherché à le vérifier ou ne serait-ce que me donner une intuition sur le sujet, c'est purement spéculatif.
- Features Engineering : ajouter les données polynomiales **avant** de normaliser les données et non l'inverse. Les variables finales sont censées être des versions normalisées de ce qu'elles représentent et non être une version polynomiale de variables normalisées qui dépendent elles-mêmes déjà des données.

- Hyperparameters tuning : mieux affiner les hyperparamètres avec des recherches plus fines et détaillées aurait pu améliorer les résultats, même si ce ne sont que des améliorations très succinctes.

6.5 Analyse critique

Comment les résultats de votre projet se comparent-ils à vos attentes (par exemple, basées sur la littérature, l'intuition ou la connaissance du domaine) ?

Initialement je n'avais pas vraiment de connaissances dans le domaine si ce n'est mon expérience d'utilisateur de vélos partagés qui est plutôt limitée. Je n'ai pas fait de recherche de littérature à ce sujet car les enjeux étant majoritairement de progresser moi-même (d'avoir une bonne note également, mais je pense valider ce cours même avec une note moyenne au projet et donc c'est un objectif secondaire) je ne voulais pas me donner une orientation ou me biaiser initialement parce qu'untel ou untel a utilisé tel ou tel modèle avec tels paramètres. C'est peut-être une erreur de ma part, mais j'irai explorer certains autres projets dans les jours qui viennent et cela fera office d'une sorte de correction.

Je n'avais pas d'attentes spécifiques en ce qui concerne les métriques d'évaluation, mais j'espérais initialement atteindre un pourcentage d'erreur moyen en deça de 15/20%. Je voulais également faire le projet le plus propre possible et aller au bout de tout ce que j'avais en tête de faire (sections 6.1 et 6.4 montrent que ce n'est pas le cas). Compte tenu des résultats, qui sont de 611 (resp. 38) de RMSE pour le modèle sur `day.csv` (resp. `hour.csv`), avec un pourcentage d'erreur moyen de 105% (resp. 36%), je considère que l'objectif n'est pas atteint dans le sens où il ne permet pas une mise en production (même si ce n'est pas le but ici). Un objectif lors de ce projet était également de comprendre le dataset et de s'exercer, et de ce point de vue là c'est déjà beaucoup mieux. En m'y mettant plus régulièrement plutôt qu'un gros coup toutes les semaines (erreur d'organisation du temps de travail de ma part), je pense que j'aurais également pu lisser les améliorations sur plus de sessions de travail et donc éviter la tunnel vision liée à des sessions de plusieurs heures d'affilée.

Dans l'absolu, une erreur de 38 sur le dataset des heures n'est pas rédhibitoire pour l'intérêt du modèle puisque sur une journée les erreurs peuvent se compenser d'une heure à une autre. En effet elles peuvent aussi s'accumuler, mais sur un compte total de vélo dans la ville, une quarantaine de vélo n'est pas très importante par rapport aux milliers qui sont loués chaque jour (une moyenne de 4500 vélos loués par jour)(même si aucune information n'est disponible sur le nombre de vélos que compte la ville). A propos des résultats du dataset des jours, qui sont d'une granularité moins fine, on peut plus facilement douter de leur importance. En effet, une erreur de 600 vélos sur la journée peut poser des problèmes car on arrive sur des nombres conséquents. D'après moi, le modèle journalier de prédictions peut être utilisé pour se donner une idée de l'affluence pour une semaine à venir (sur du plus long terme la fiabilité des prévisions météorologiques peut laisser à désirer), mais l'erreur résiduelle limite tout de même les utilisations.

6.6 Leçons apprises

Résumez les leçons les plus importantes que vous avez apprises en réalisant ce projet.

Tout au long de ce projet, j'ai appris à tester des hypothèses, des intuitions et vérifier des idées plus vite. J'avais l'habitude de prendre mon temps et de faire des recherches sur internet soigneusement et j'avais toujours tendance à perdre du temps sur des tâches qui devraient être rapides voire immédiates.

Je n'aime pas utiliser **chat-GPT** pour mon travail hors cas extrême où les recherches classiques

ne peuvent pas m'aider, et c'était un très bon exercice pour mettre toutes les connaissances que j'ai apprises (durant ce cours et ailleurs) en pratique, le faire rapidement et corriger vite mes erreurs si j'en voyais, même si ça m'a certainement pris beaucoup plus de temps que ce que j'aurais pu y passer autrement.

Je pense que ce projet m'a surtout fait gagner en rapidité d'analyse et en "prototypage" d'idées/intuitions, ce qui m'a permis d'essayer beaucoup plus que je ne le pouvais avant (au niveau des graphes tracés et du features engineering je n'ai même pas mis la moitié de ce que j'ai essayé sinon le rapport aurait fait 60 pages). Je suis également plutôt content de mes progrès sur ce point, ainsi que sur ma capacité de concentration que je me suis forcé à améliorer pendant le projet.

Enfin, j'ai appris que j'ai encore besoin de progresser sur mon organisation lorsque j'ai plusieurs projets à gérer en même temps.