

Projet velo Data Mining - Geoffrey HARRAZI

Le projet consiste à prévoir le nombre de vélos loués à chaque heure dans des bornes libres-services de la ville (système type Vélib'). La variable cible est ici la variable **count**.

Voici un descriptif de l'ensemble des variables :

- *datetime* - date et heure du relevé
- *season* - 1 = printemps , 2 = été, 3 = automne, 4 = hiver
- *holiday* – indique si le jour est un jour de vacances scolaires
- *workingday* - indique si le jour est travaillé (ni week-end ni vacances)
- *weather* - 1: Dégagé à nuageux, 2 : Brouillard, 3 : Légère pluie ou neige, 4 : Fortes averses ou neiges
- *temp* – température en degrés Celsius
- *atemp* – température ressentie en degrés Celsius
- *humidity* – taux d'humidité
- *windspeed* – vitesse du vent
- *casual* - nombre de locations d'usagers non abonnés
- *registered* – nombre de locations d'usagers abonnés
- *count* – nombre total de locations de vélos

L'objectif du projet est de mener à bien la création d'un modèle qui pourrait théoriquement être déployé en production. Les étapes d'exploration des données, de traitement et de preprocessing ne sont bien entendu pas à négliger. Il ne s'agit pas d'une compétition de type *Kaggle*, le projet ne sera pas uniquement noté sur la performance du modèle, mais plutôt sur votre approche complète et la justification de chacun de vos choix.

Comme vu durant le cours, soyez faites attention à certains points :

- quel type de problème dois-je traiter ?
- feature engineering : est-ce que j'utilise les données correctement, toutes les données ?
- data leakage : est-ce qu'une de mes features n'est pas trop explicative ?
- ai-je bien traité toutes les données correctement ?
- est-ce que mon modèle est adapté ?
- etc, etc, etc

Soyez vigilant à expliquer et justifier votre démarche à l'aide de visualisation, de commentaires dans vos codes (pensez aux cellules markdown), etc

Librairies

In [1211]:

```
#Import des librairies utilisés
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

#import statsmodels.formula.api as sm
import statsmodels.api as sm

from IPython.display import Image
from IPython.core.display import HTML
%matplotlib inline
```

In [1212]:

```
#Import 2
from scipy import stats

from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler

#metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_log_error

#models
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNet

#model selection
from sklearn.model_selection import train_test_split,cross_validate
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV

sns.set_style(style = 'whitegrid')
```

In [1213]:

```
import pycaret
from pycaret.regression import *
```

Premier contact et mise en forme

Première contact avec les données. J'en affiche des extraits, j'en regarde le nombre d'entrée et de paramètres. Et je met en forme le DataFrame

In [1214]:

```
df = pd.read_csv("../data/input/velo.csv")
df.head()
```

Out[1214]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	cas
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	

In [1215]:

```
nrow, ncol = df.shape
df.shape
```

Out[1215]:

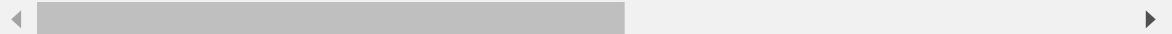
(10886, 12)

In [1216]:

```
df.describe(include='all')
```

Out[1216]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
count	10886	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
unique	10886	NaN							
top	2012-07-17 09:00:00	NaN							
freq	1	NaN							
mean	NaN	2.506614	0.028569	0.680875	1.418427	20.23086	14.52222	80.0	NaN
std	NaN	1.116174	0.166599	0.466159	0.633839	7.79159	7.50000	82.000	NaN
min	NaN	1.000000	0.000000	0.000000	1.000000	0.82000	13.94000	1.00000	NaN
25%	NaN	2.000000	0.000000	0.000000	1.000000	20.50000	14.37500	82.000	NaN
50%	NaN	3.000000	0.000000	1.000000	1.000000	20.50000	14.52222	82.000	NaN
75%	NaN	4.000000	0.000000	1.000000	2.000000	26.24000	17.18750	82.000	NaN
max	NaN	4.000000	1.000000	1.000000	4.000000	41.00000	28.50000	82.000	NaN



On constate que casual + registered = count Il est évident que c'est donnée ne doivent pas servir au training

In [1217]:

```
df = df.drop(["casual", "registered"], axis=1)
df.head()
```

Out[1217]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	cou
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	1
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	1
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	1
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	1



On va maintenant affecter le bon type de donnée pour chaque paramètres du dataframe

In [1218]:

```
df.dtypes
```

Out[1218]:

datetime	object
season	int64
holiday	int64
workingday	int64
weather	int64
temp	float64
atemp	float64
humidity	int64
windspeed	float64
count	int64
dtype:	object

On va faire les changements de type suivants (on se base sur le descriptifs des variables disponible et le bon sens):

```
datetime -> Date
season -> Catégorie
holiday -> Catégorie
workingday -> Catégorie
weather -> Catégorie
```

Quand on ne sait pas on peut vérifier de cette façon. Si une catégorie n'a pas "beaucoup" de valeur différente on peut en déduire que c'est une catégorie

In [1219]:

```
for colname, serie in df.iteritems():
    print(colname + " has " + str(serie.drop_duplicates().shape[0]) + " unique values.")
)
```

```
datetime has 10886 unique values.
season has 4 unique values.
holiday has 2 unique values.
workingday has 2 unique values.
weather has 4 unique values.
temp has 49 unique values.
atemp has 60 unique values.
humidity has 89 unique values.
windspeed has 28 unique values.
count has 822 unique values.
```

In [1220]:

```
df_no_type = df.copy()
df["datetime"] = pd.to_datetime(df["datetime"])
df["season"] = pd.Categorical(df["season"], ordered=True)
df["holiday"] = pd.Categorical(df["holiday"], ordered=False)
df["workingday"] = pd.Categorical(df["workingday"], ordered=False)
df["weather"] = pd.Categorical(df["weather"], ordered=False)
```

Valeurs manquantes

On vérifie les changement. On remarque au même moment que le dataset est plutôt propre et qu'on aura pas à gerer les valeurs NULL

In [1221]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   datetime    10886 non-null   datetime64[ns]
 1   season      10886 non-null   category
 2   holiday     10886 non-null   category
 3   workingday  10886 non-null   category
 4   weather     10886 non-null   category
 5   temp         10886 non-null   float64
 6   atemp        10886 non-null   float64
 7   humidity    10886 non-null   int64  
 8   windspeed   10886 non-null   float64
 9   count        10886 non-null   int64  
dtypes: category(4), datetime64[ns](1), float64(3), int64(2)
memory usage: 553.5 KB
```

In [1222]:

```
df.isnull().sum()
```

Out[1222]:

```
datetime      0
season        0
holiday       0
workingday   0
weather       0
temp          0
atemp         0
humidity     0
windspeed    0
count         0
dtype: int64
```

Désormais on peut maintenant voir les valeur min et max pour datetime par exemple.

In [1223]:

df.describe(include='all')

Out[1223]:

	datetime	season	holiday	workingday	weather	temp	atemp	hu
count	10886	10886.0	10886.0	10886.0	10886.0	10886.00000	10886.00000	10886.00000
unique	10886	4.0	2.0	2.0	4.0	NaN	NaN	NaN
top	2011-06-09 04:00:00	4.0	0.0	1.0	1.0	NaN	NaN	NaN
freq	1	2734.0	10575.0	7412.0	7192.0	NaN	NaN	NaN
first	2011-01-01 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
last	2012-12-19 23:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	20.23086	23.655084	61.8
std	NaN	NaN	NaN	NaN	NaN	7.79159	8.474601	19.2
min	NaN	NaN	NaN	NaN	NaN	0.82000	0.760000	0.0
25%	NaN	NaN	NaN	NaN	NaN	13.94000	16.665000	47.0
50%	NaN	NaN	NaN	NaN	NaN	20.50000	24.240000	62.0
75%	NaN	NaN	NaN	NaN	NaN	26.24000	31.060000	77.0
max	NaN	NaN	NaN	NaN	NaN	41.00000	45.455000	100.0

Exploration des données

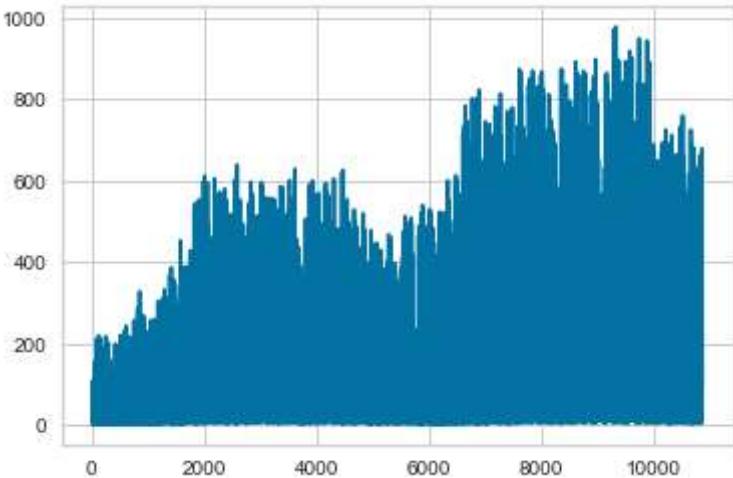
Count

In [1224]:

```
plt.plot(df["count"])
```

Out[1224]:

```
[<matplotlib.lines.Line2D object at 0x000001C2526D1580>]
```



On peut voir des décrochages à certain moment. J'emet l'hypothèse que c'est décrochage (des moments avec peu de location) sont corrélé avec les jours de vacances. Et affichant les graphiques des deux variables l'une au dessus de l'autre on voit que la corrélation est flagrante. En étant attentif on peut aussi repérer qu'il y a 2 fois le même pattern de jours de vacances. Ce qui confirme que le dataset est étallé sur un peu près 2ans.

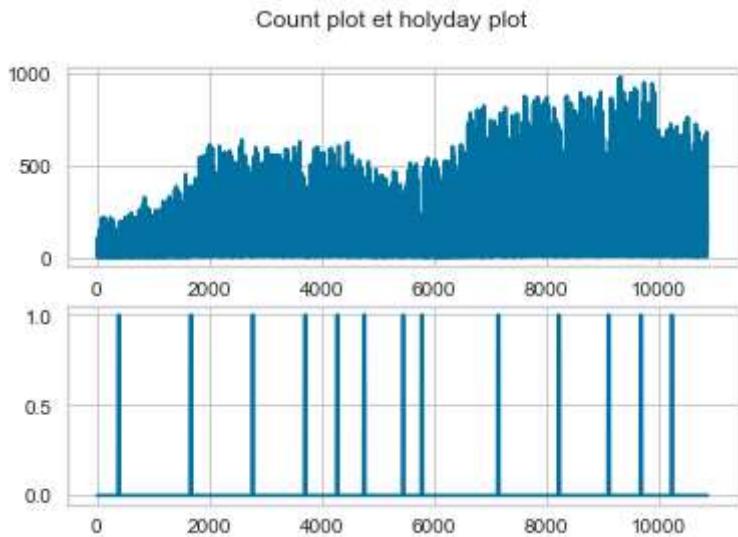
On voit aussi que le service a gagné en popularité au cours des deux ans. Une tendance qu'il sera difficile à prédire avec nos données actuelle.

In [1225]:

```
fig, (ax1, ax2) = plt.subplots(2)
fig.suptitle('Count plot et holyday plot')
ax1.plot(df["count"])
ax2.plot(df["holiday"])
#ax3.plot(df.workingday.values[::-1])
#Il y a pour moi une redondance entre les collums holiday et workingday
#Je verrais ça plus tard.
#Ajouter une collumns weekend ? et delete workingday ?
```

Out[1225]:

[<matplotlib.lines.Line2D object at 0x000001C252539EE0>]



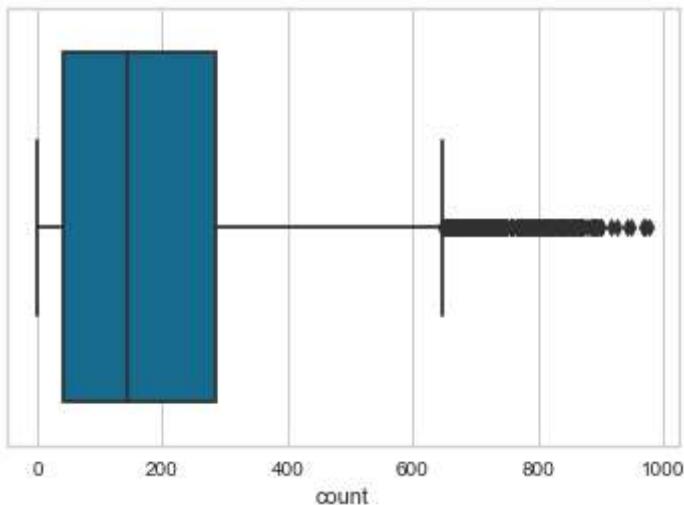
On peut voir qu'il y a des outliers sur cette variable. C'est outlier vont fausser les conclusion de notre modèle parcequ'elles ne représentent pas la réalité générale. Je garde cette infos et nous allons supprimer les outliers à la fin de l'exploration.

In [1226]:

```
sns.boxplot(x=df['count'])
```

Out[1226]:

<AxesSubplot:xlabel='count'>



Date

On a seulement deux an d'historique. Il va donc être compliqué de ce basé sur la date pour faire un modèle.
Je pense qu'il sera intéressant d'explorer la date en features plus représentatif et plus corrélé avec count

Experimentation:

In [1227]:

```
df['dayofweek'] = df.datetime.dt.dayofweek
df['hour'] = df.datetime.dt.hour
df['month'] = df.datetime.dt.month
df['year'] = df.datetime.dt.year
```

In [1228]:

```
df["dayofweek"] = pd.Categorical(df["dayofweek"], ordered=False)
df["hour"] = pd.Categorical(df["hour"], ordered=False)
df["month"] = pd.Categorical(df["month"], ordered=False)
df["year"] = pd.Categorical(df["year"], ordered=False)
```

Mois

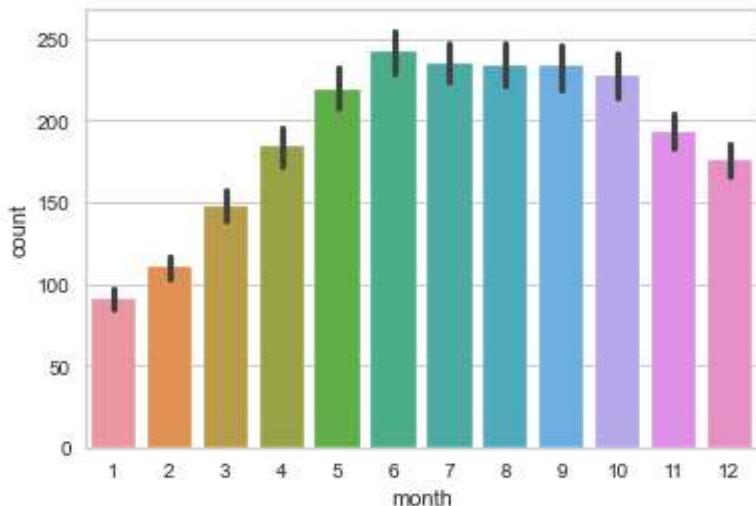
On a ici les valeurs moyenne de la variable count par mois (1= Janvier, 2 = Février ...). Cette variable est très intéressante parce qu'on voit une variation évidente des locations selon le mois.

In [1229]:

```
sns.barplot(data=df,x="month",y="count")
```

Out[1229]:

```
<AxesSubplot:xlabel='month', ylabel='count'>
```



Jour de la semaine

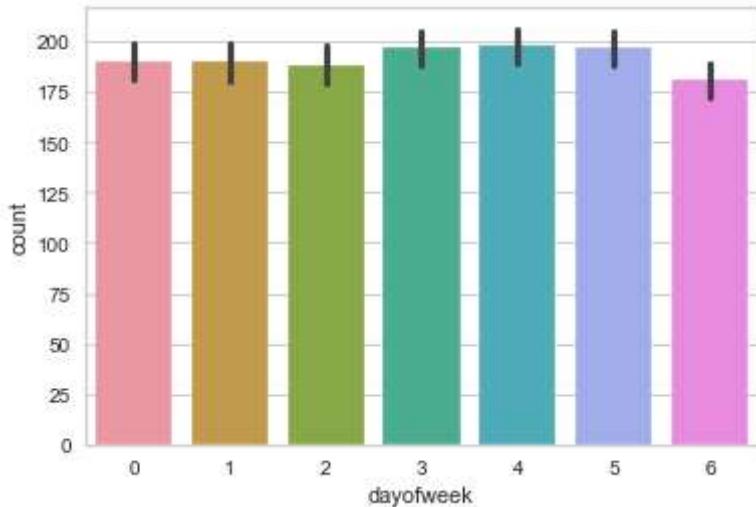
Ici c'est moins évident à voir. On pourrait croiser le jours de la semaine et les heures. Les habitudes devraient être différentes. Par exemple des plages horaires de forte activité différentes le lundi et le dimanche

In [1230]:

```
sns.barplot(data=df,x="dayofweek",y="count")
```

Out[1230]:

```
<AxesSubplot:xlabel='dayofweek', ylabel='count'>
```



Heure

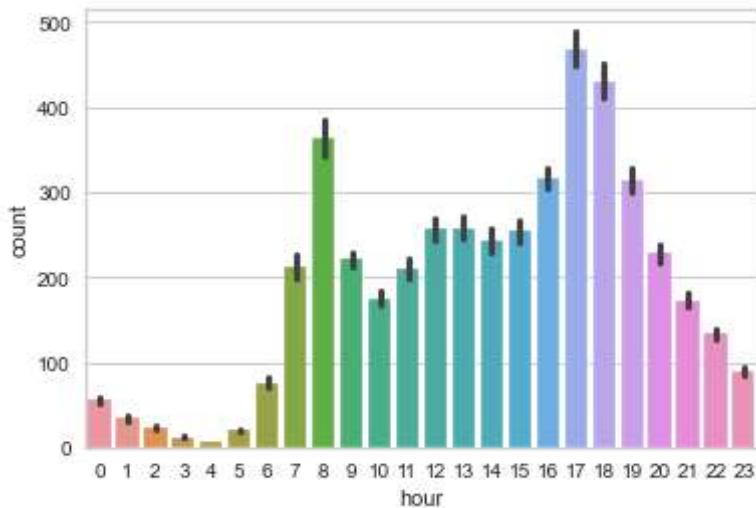
Pareil pour la variable heure. On peut voir des distributions fortement corrélées avec l'heure de location

In [1231]:

```
sns.barplot(data=df,x="hour",y="count")
```

Out[1231]:

```
<AxesSubplot:xlabel='hour', ylabel='count'>
```

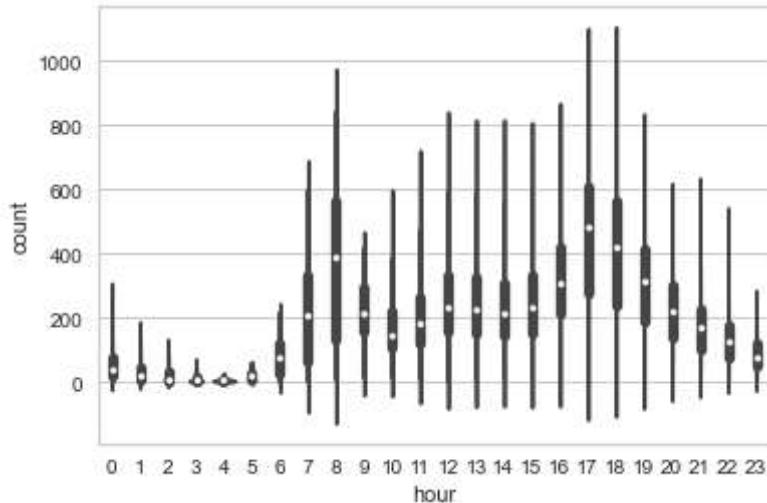


In [1232]:

```
sns.violinplot(data = df, x = 'hour', y = 'count')
```

Out[1232]:

```
<AxesSubplot:xlabel='hour', ylabel='count'>
```



La variable hour est très correlé à la variable count

In [1233]:

```
df['hour2'] = df.datetime.dt.hour
df['hour2'].corr(df['count'])
```

Out[1233]:

```
0.40060119414684725
```

In [1234]:

```
df = df.drop(["hour2"],axis=1)
```

Relation des variables date

On observe comme supposé une vrai différence de distribution entre les jours de weekend et les jours de semaine (0:4 = Lundi:Vendredi, 5 = samedi, 6 = dimanche)

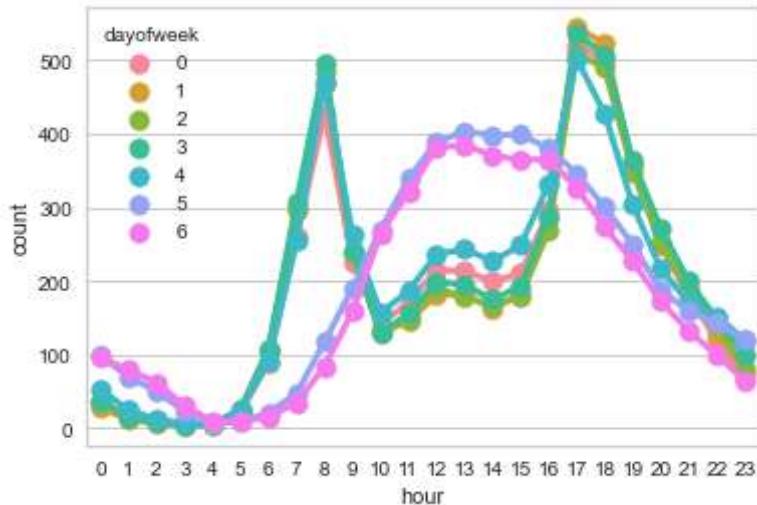
On peut voir des différences de proportion lorsqu'on croise hour avec season, weather. Ce sont des informations que notre modèle utilisera pour sa prédiction.

In [1235]:

```
hourGroupbyDayOfWeek = pd.DataFrame(df.groupby(["hour", "dayofweek"], sort=True)[["count"]].mean()).reset_index()
sns.pointplot(x=hourGroupbyDayOfWeek["hour"], y=hourGroupbyDayOfWeek["count"], hue=hourGroupbyDayOfWeek["dayofweek"], data=hourGroupbyDayOfWeek, join=True,)
```

Out[1235]:

```
<AxesSubplot:xlabel='hour', ylabel='count'>
```



On pourrait peut-être simplifier le modèle en ayant une variable isWeekend. Vu qu'on identifie clairement une courbe de location pour les jours de semaine et une pour le weekend. Mais j'ai peur que ce soit redondant avec la variable workingday. On peut vérifier le même graphique avec workingday =0 et workingday =1

In [1236]:

df

Out[1236]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000
...
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981

10886 rows × 14 columns

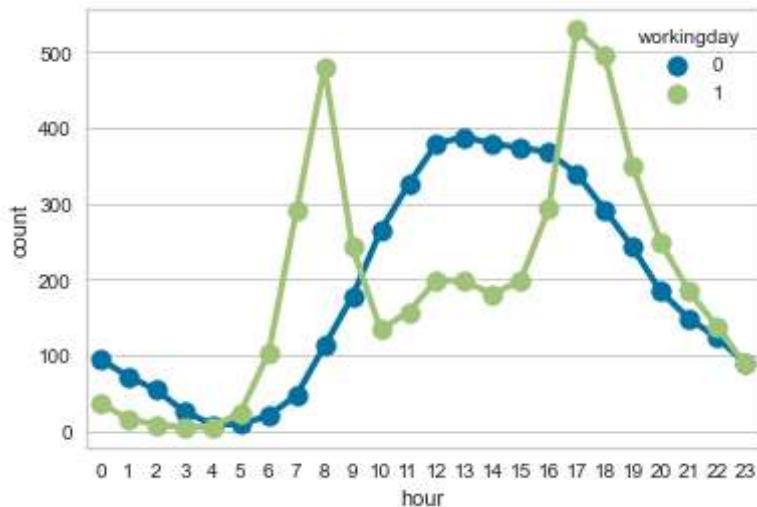


In [1237]:

```
hourGroupbyWorkingday = pd.DataFrame(df.groupby([ "hour", "workingday"], sort=True)[ "count"].mean()).reset_index()
sns.pointplot(x=hourGroupbyWorkingday[ "hour"], y=hourGroupbyWorkingday[ "count"], hue=hourGroupbyWorkingday[ "workingday"], data=hourGroupbyDayOfWeek, join=True,)
```

Out[1237]:

```
<AxesSubplot:xlabel='hour', ylabel='count'>
```



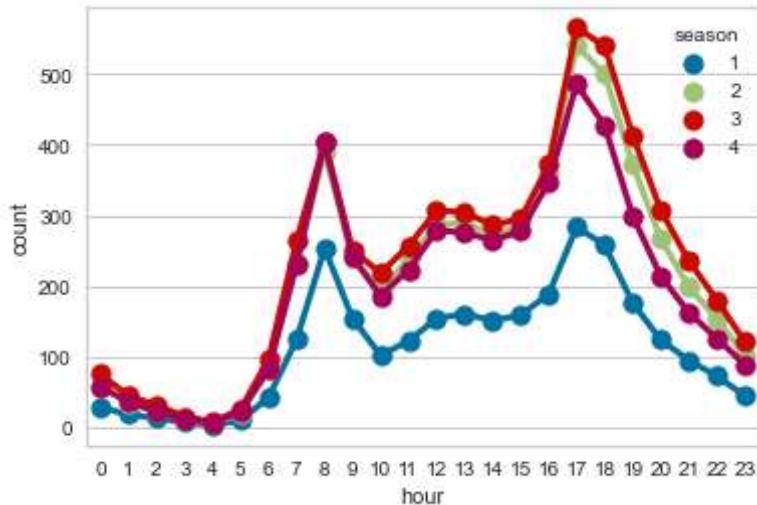
Comme on pouvait si attendre. Les courbes ne corresponde pas vraiment à jour de semaine et jour de weekend. Mais la répartition des locations dépend si le jour est un jour de travail ou de repos. Donc workingday "possède" déjà cette information. On va garder les jours de semaine pour enlever les différences de proportions entre les jours. Par exemple: à 14h et 15h on a une différence de location entre samedi et dimanche.

In [1238]:

```
hourGroupbySeason = pd.DataFrame(df.groupby(["hour", "season"], sort=True)[["count"]].mean()
()).reset_index()
sns.pointplot(x=hourGroupbySeason[ "hour" ], y=hourGroupbySeason[ "count" ], hue=hourGroupby
Season[ "season" ], data=hourGroupbySeason, join=True,)
```

Out[1238]:

```
<AxesSubplot:xlabel='hour', ylabel='count'>
```

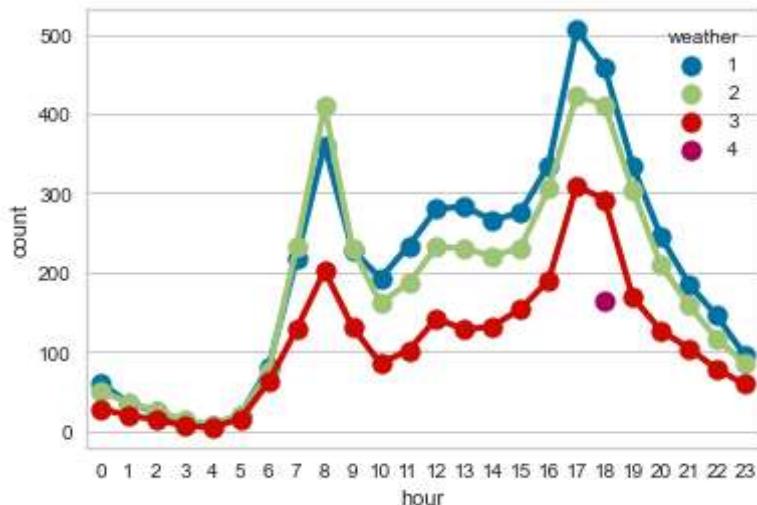


In [1239]:

```
hourGroupbyWeather = pd.DataFrame(df.groupby(["hour", "weather"], sort=True)[["count"]].mean()).reset_index()
sns.pointplot(x=hourGroupbyWeather["hour"], y=hourGroupbyWeather["count"], hue=hourGroupbyWeather["weather"], data=hourGroupbyWeather, join=True,)
```

Out[1239]:

```
<AxesSubplot:xlabel='hour', ylabel='count'>
```



Je supprime maintenant la column datetime qui est devenu inutile

In [1240]:

```
df = df.drop(["datetime"], axis=1)
```

In [1241]:

df

Out[1241]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	date
0	1	0	0	1	9.84	14.395	81	0.0000	16	
1	1	0	0	1	9.02	13.635	80	0.0000	40	
2	1	0	0	1	9.02	13.635	80	0.0000	32	
3	1	0	0	1	9.84	14.395	75	0.0000	13	
4	1	0	0	1	9.84	14.395	75	0.0000	1	
...
10881	4	0	1	1	15.58	19.695	50	26.0027	336	
10882	4	0	1	1	14.76	17.425	57	15.0013	241	
10883	4	0	1	1	13.94	15.910	61	15.0013	168	
10884	4	0	1	1	13.94	17.425	61	6.0032	129	
10885	4	0	1	1	13.12	16.665	66	8.9981	88	

10886 rows × 13 columns

Windspeed

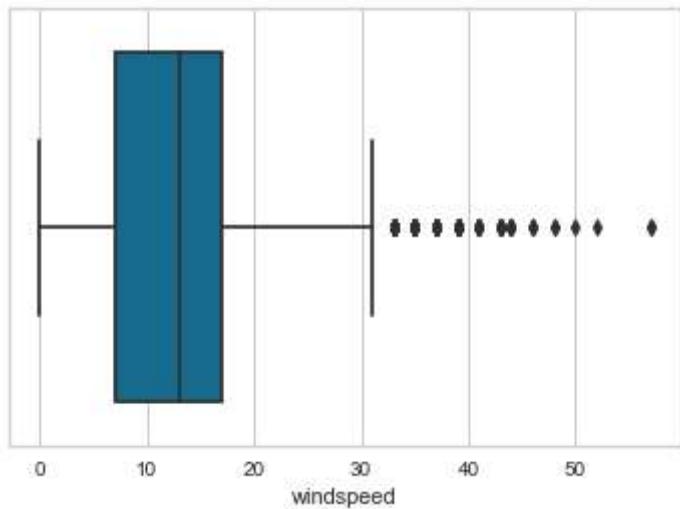
On peut voir ici qu'on a des valeurs de où windspeed est égale à zero. Ce qui semble plutot dû à un manque de données. Car la vitesse du vent ne peut pas être à zero.

In [1242]:

```
sns.boxplot(x=df['windspeed'])
```

Out[1242]:

```
<AxesSubplot:xlabel='windspeed'>
```



In [1243]:

```
df[df["windspeed"] == 0]
```

Out[1243]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	date
0	1	0	0	1	9.84	14.395	81	0.0	16	2010-01-01 00:00:00
1	1	0	0	1	9.02	13.635	80	0.0	40	2010-01-02 00:00:00
2	1	0	0	1	9.02	13.635	80	0.0	32	2010-01-03 00:00:00
3	1	0	0	1	9.84	14.395	75	0.0	13	2010-01-04 00:00:00
4	1	0	0	1	9.84	14.395	75	0.0	1	2010-01-05 00:00:00
...
10826	4	0	1	2	16.40	20.455	87	0.0	232	2010-01-26 00:00:00
10829	4	0	1	2	17.22	21.210	88	0.0	211	2010-01-29 00:00:00
10846	4	0	1	1	15.58	19.695	94	0.0	662	2010-01-46 00:00:00
10860	4	0	1	1	13.94	16.665	49	0.0	132	2010-01-60 00:00:00
10862	4	0	1	1	12.30	15.910	61	0.0	41	2010-01-62 00:00:00

1313 rows × 13 columns

In [1244]:

```
windspeedGroupByMonths = pd.DataFrame(df.groupby(["windspeed", "month"]))
```

In [1245]:

```
df_windspeed = df.copy()
df_windspeed["month"] == pd.Categorical(df_windspeed["month"])
```

Out[1245]:

```
0      True
1      True
2      True
3      True
4      True
...
10881  True
10882  True
10883  True
10884  True
10885  True
Name: month, Length: 10886, dtype: bool
```

In [1246]:

```
df_windspeed.groupby("month")["windspeed"].mean()
```

Out[1246]:

```
month
1    14.582959
2    13.963707
3    15.363249
4    15.581090
5    12.293956
6    12.348930
7    11.019928
8    11.931179
9    11.575698
10   11.226457
11   13.126587
12   10.682489
Name: windspeed, dtype: float64
```

Ici on la moyenne de la valeurs windspeed par mois. Je vais donc remplacer les valeurs égale à zero avec la valeur moyenne du mois.

A default de réussir à faire cette opération de manière "propre" j'ai fais comme ceci:

In [1247]:

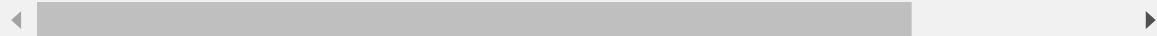
```
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 1) ), 14.58, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 2) ), 13.96, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 3) ), 15.36, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 4) ), 15.58, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 5) ), 12.29, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 6) ), 12.34, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 7) ), 11.01, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 8) ), 11.93, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 9) ), 11.57, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 10) ), 11.22, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 11) ), 13.12, df['windspeed'])
df['windspeed'] = np.where( ( (df['windspeed'] == 0) & (df['month'] == 12) ), 10.68, df['windspeed'])
```

In [1248]:

```
df[df["windspeed"] == 0]
```

Out[1248]:

season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	dayofwe
--------	---------	------------	---------	------	-------	----------	-----------	-------	---------



Exploration des variables catégorie

In [1249]:

```
#Petit rappel
for colname, serie in df.iteritems():
    print(colname + " has " + str(serie.drop_duplicates().shape[0]) + " unique values.")
)
```

```
season has 4 unique values.
holiday has 2 unique values.
workingday has 2 unique values.
weather has 4 unique values.
temp has 49 unique values.
atemp has 60 unique values.
humidity has 89 unique values.
windspeed has 39 unique values.
count has 822 unique values.
dayofweek has 7 unique values.
hour has 24 unique values.
month has 12 unique values.
year has 2 unique values.
```

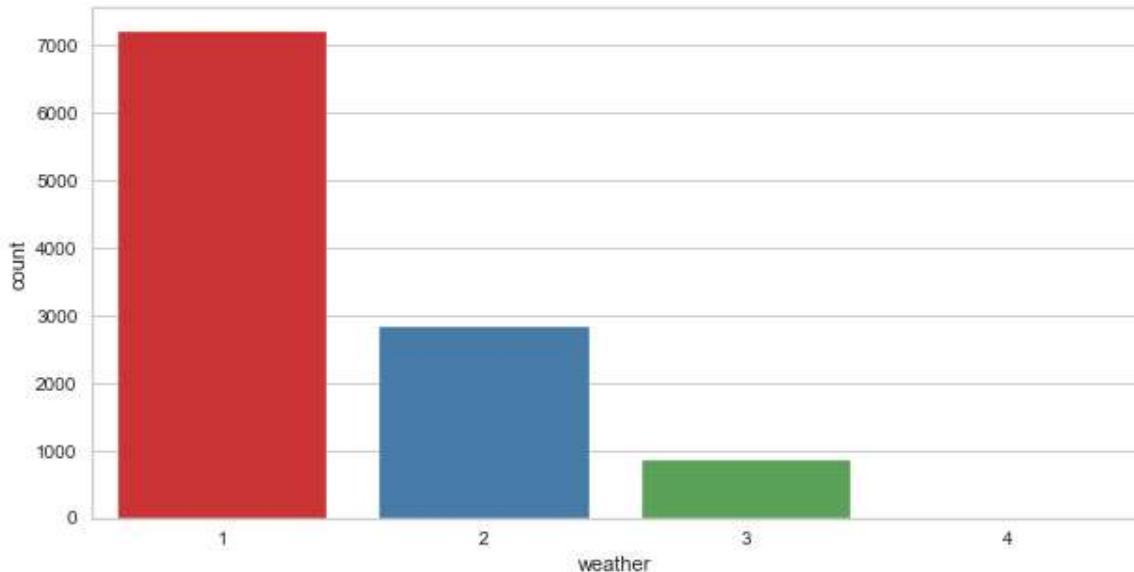
In [1250]:

```
plt.figure(figsize=(10,5))
chart = sns.countplot(x = 'weather', data = df, palette='Set1')
chart.set_xticklabels(chart.get_xticklabels())

#weather - 1: Dégagé à nuageux, 2 : Brouillard, 3 : Légère pluie ou neige, 4 : Fortes averses ou neiges
```

Out[1250]:

```
[Text(0, 0, '1'), Text(1, 0, '2'), Text(2, 0, '3'), Text(3, 0, '4')]
```



Je remarque qu'il n'y a que 1 exemple de jour avec une forte averse. Cette exemple sera soit dans le train soit dans le test donc on peut supprimer cette enregistrement

In [1251]:

```
df[df["weather"]==4]
```

Out[1251]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	day
5631	1	0		1	4	8.2	11.365	86	6.0032	164

In [1252]:

```
df = df.drop(5631)
```

On enlève carrément la catégorie 4

In [1253]:

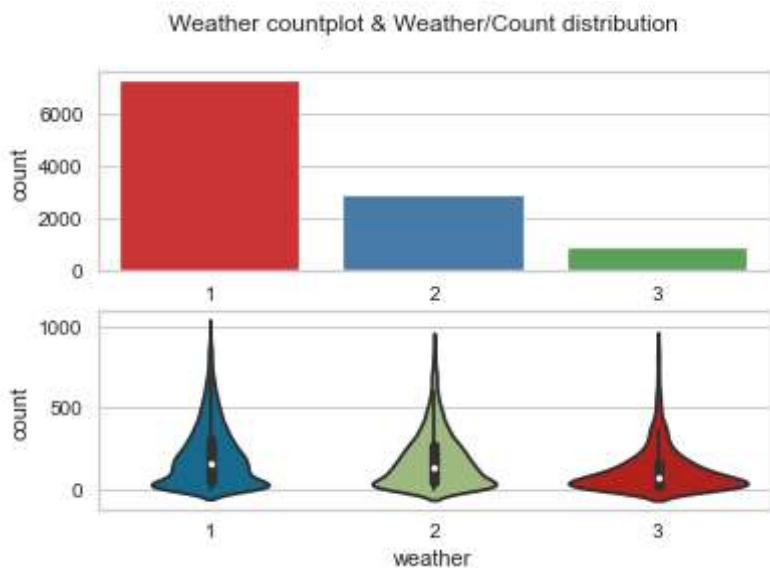
```
df.weather = df.weather.cat.remove_unused_categories()
```

In [1254]:

```
fig, axes = plt.subplots(2, 1)
fig.suptitle('Weather countplot & Weather/Count distribution')
sns.countplot(ax=axes[0], x = 'weather', data = df, palette='Set1')
sns.violinplot(ax=axes[1], data = df, x = 'weather', y = 'count')
```

Out[1254]:

```
<AxesSubplot:xlabel='weather', ylabel='count'>
```

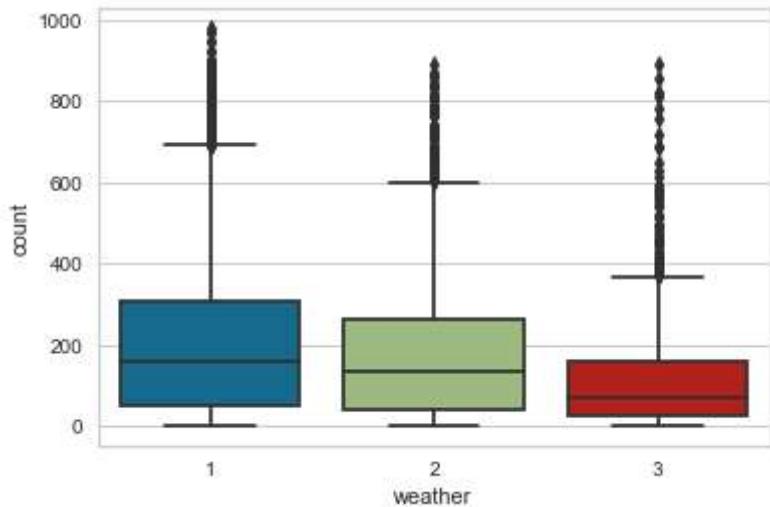


In [1255]:

```
sns.boxplot( data = df, x = 'weather', y = 'count')
```

Out[1255]:

```
<AxesSubplot:xlabel='weather', ylabel='count'>
```



Je remarque holiday et workday sont redondant. Et j'ai l'impression qu'il y a surement une meilleur façon de représenter les Jours de vacances, les week-end et les jours travaillé. Je reprendrais ce raisonnement dans la suite de l'analyse.

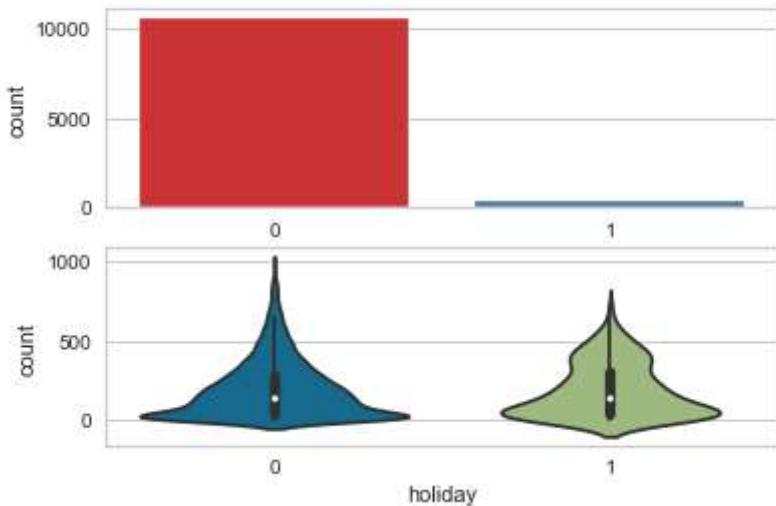
In [1256]:

```
fig, axes = plt.subplots(2, 1)
fig.suptitle('Holiday countplot & Holiday/Count distribution')
sns.countplot(ax=axes[0], x = 'holiday', data = df, palette='Set1')
sns.violinplot(ax=axes[1], data = df, x = 'holiday', y = 'count')
```

Out[1256]:

<AxesSubplot:xlabel='holiday', ylabel='count'>

Holiday countplot & Holiday/Count distribution

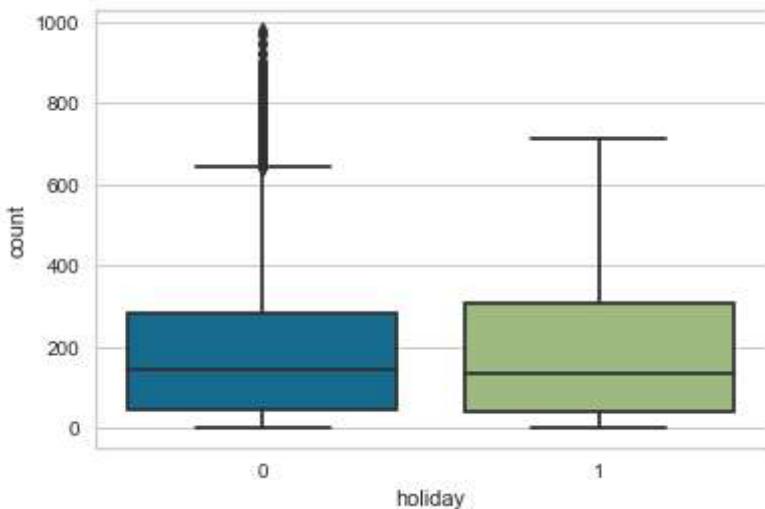


In [1257]:

```
sns.boxplot( data = df, x = 'holiday', y = 'count')
```

Out[1257]:

<AxesSubplot:xlabel='holiday', ylabel='count'>

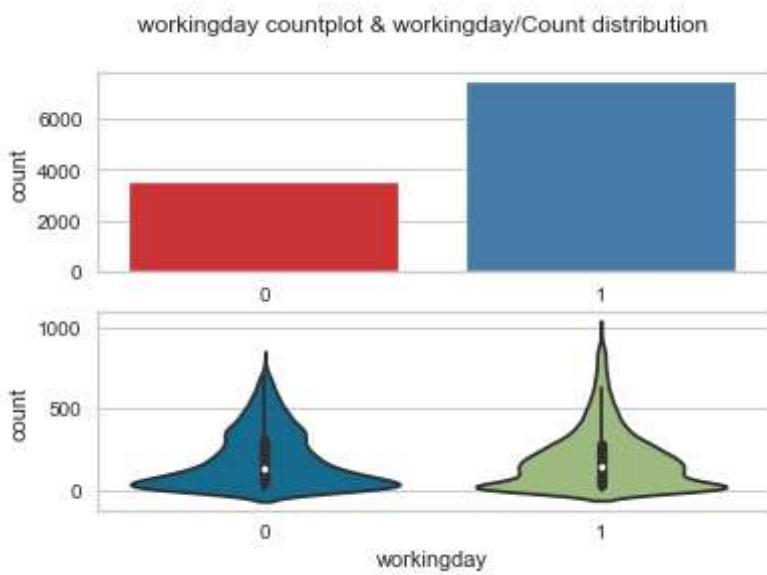


In [1258]:

```
fig, axes = plt.subplots(2, 1)
fig.suptitle('workingday countplot & workingday/Count distribution')
sns.countplot(ax=axes[0], x = 'workingday', data = df, palette='Set1')
sns.violinplot(ax=axes[1], data = df, x = 'workingday', y = 'count')
```

Out[1258]:

<AxesSubplot:xlabel='workingday', ylabel='count'>

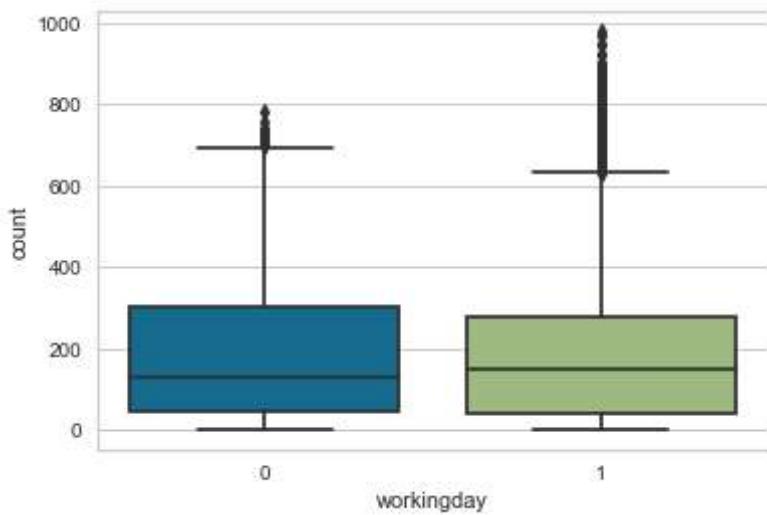


In [1259]:

```
sns.boxplot( data = df, x = 'workingday', y = 'count')
```

Out[1259]:

<AxesSubplot:xlabel='workingday', ylabel='count'>



On peut voir avec les graphiques précédent la corrélation entre la variable weather et count. Pour les variables workingday et holiday, j'ai l'impression qu'il y a une meilleure façon de traduire cette aspect en donnée et qu'elles pourraient être redondant entre elles. Mais je ne vois pas comment faire mieux à l'heure actuelle.

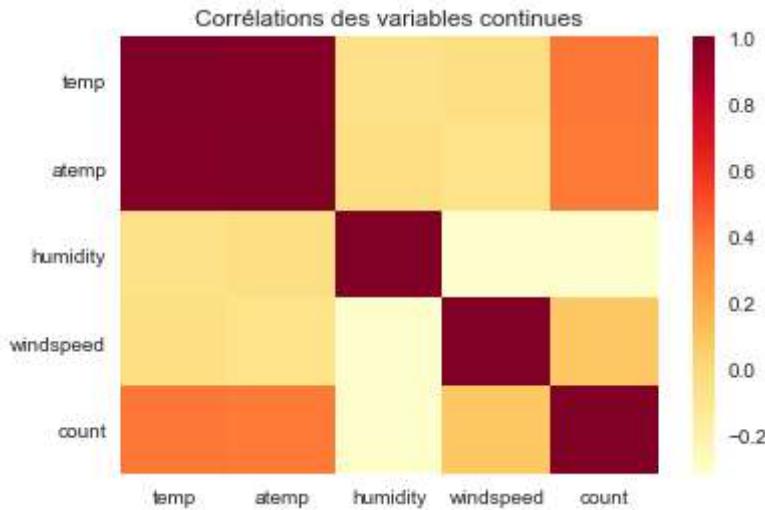
On peut observer des outliers lorsque workingday=1 et holiday=0

Exploration des variables continues

Pour m'attaquer à l'exploration des variables continue je vais d'abord dégrossir avec une heatmap de correlation générale:

In [1260]:

```
sns.heatmap(df.corr(), cmap="YlOrRd")
plt.title("Corrélations des variables continues")
plt.show()
```



Observations:

- Les variables temp et atemp ont une bonne corrélation avec notre variable cible (count). Elles pourraient être utile à notre modèle, Mais elles sont trop corrélé entre elle. Et ensemble elles n'apportent pas plus d'information que toute seul. Je pense conservé uniquement temp. Etant donné que la température ressentie (atemp) est calculé à partir de la température et de l'humidité. ET que l'on a déjà l'humidité dans notre dataset -Humidité semble aussi avoir une corrélation mais plus faible. -Pour windspeed ça me semble proche de zero. ON va regarder ça de plus près.

In [1261]:

```
df.corr()
```

Out[1261]:

	temp	atemp	humidity	windspeed	count
temp	1.000000	0.984945	-0.064783	-0.044975	0.394476
atemp	0.984945	1.000000	-0.043376	-0.075156	0.389802
humidity	-0.064783	-0.043376	1.000000	-0.318922	-0.317377
windspeed	-0.044975	-0.075156	-0.318922	1.000000	0.082135
count	0.394476	0.389802	-0.317377	0.082135	1.000000

Je trouve toutes les variables intéressante à utiliser à part atemp. L'humidité et la vitesse du vent semble avoir une petite corrélation qui pourraient aider notre modèle.

In [1262]:

```
df.drop(['atemp'], axis=1)
```

Out[1262]:

	season	holiday	workingday	weather	temp	humidity	windspeed	count	dayofweek
0	1	0	0	1	9.84	81	14.5800	16	5
1	1	0	0	1	9.02	80	14.5800	40	5
2	1	0	0	1	9.02	80	14.5800	32	5
3	1	0	0	1	9.84	75	14.5800	13	5
4	1	0	0	1	9.84	75	14.5800	1	5
...
10881	4	0	1	1	15.58	50	26.0027	336	2
10882	4	0	1	1	14.76	57	15.0013	241	2
10883	4	0	1	1	13.94	61	15.0013	168	2
10884	4	0	1	1	13.94	61	6.0032	129	2
10885	4	0	1	1	13.12	66	8.9981	88	2

10885 rows × 12 columns

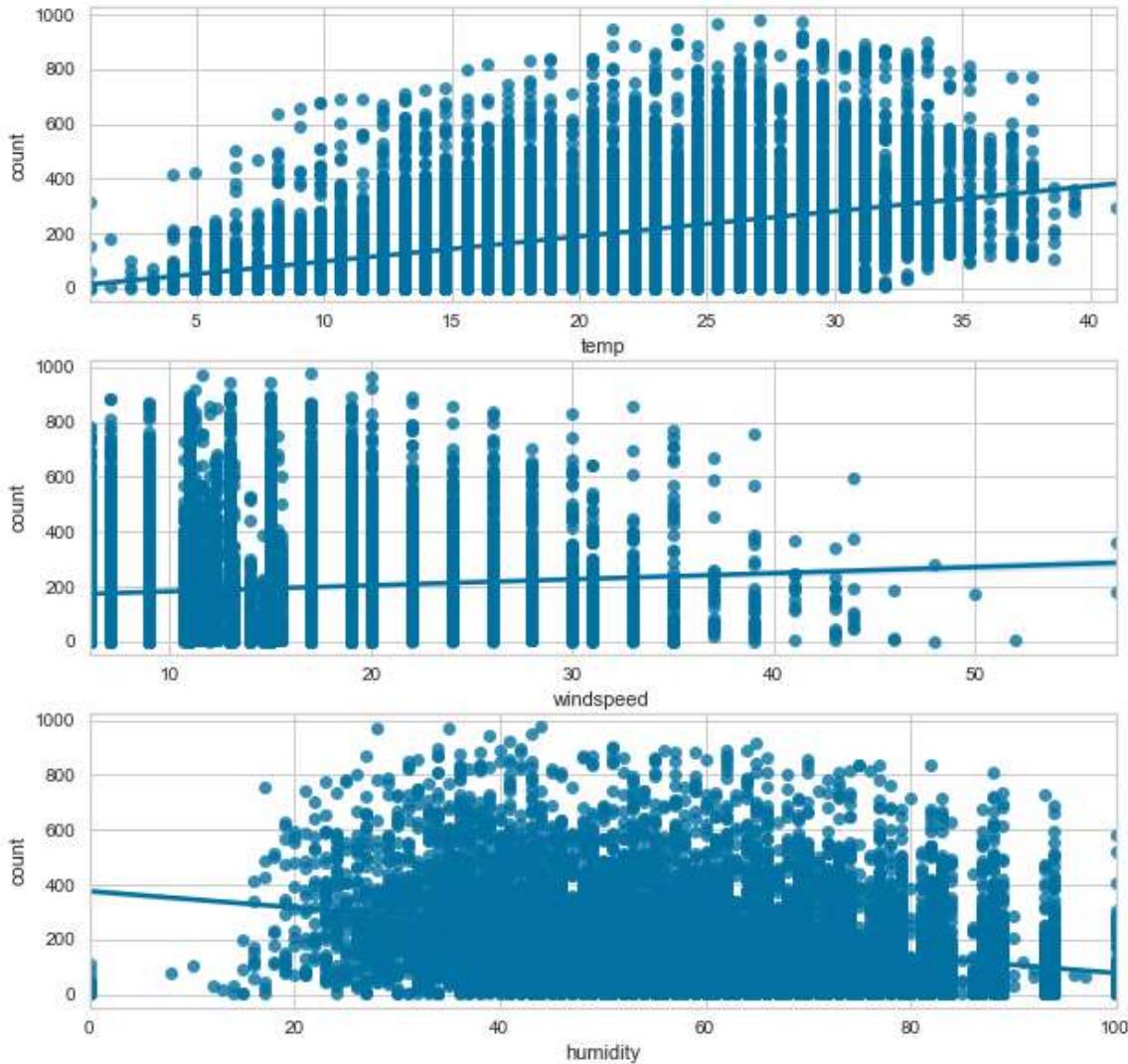
On confirme la corrélation positive de "temp" et la corrélation négative de "humidity". Enfin "windspeed" n'est pas vraiment corrélé comme plus haut.

In [1263]:

```
fig, axes = plt.subplots(3, 1)
fig.set_size_inches(10,10)
sns.regplot(ax=axes[0], x="temp", y="count", data=df)
sns.regplot(ax=axes[1], x="windspeed", y="count", data=df)
sns.regplot(ax=axes[2], x="humidity", y="count", data=df)
```

Out[1263]:

<AxesSubplot:xlabel='humidity', ylabel='count'>



Outlier

Après quelque recherche j'ai voulu utiliser le z-score sur la colonnes count pour detecter et supprimer les outliers identifié sur les boxplots. J'avais avant cela utilisé le z-score sur toutes les collonnes mais je perdais plus de 800 lignes donc je pense que c'était contre productif.

In [1264]:

```
df_noOutlier = df.copy()
df_noOutlier.head()
```

Out[1264]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	dayofw
0	1	0	0	1	9.84	14.395	81	14.58	16	
1	1	0	0	1	9.02	13.635	80	14.58	40	
2	1	0	0	1	9.02	13.635	80	14.58	32	
3	1	0	0	1	9.84	14.395	75	14.58	13	
4	1	0	0	1	9.84	14.395	75	14.58	1	

◀ ▶

In [1265]:

```
#Calcul des z-score par ligne et columns count
z = np.abs(stats.zscore(df_noOutlier["count"]))
print(z)
```

[0.9692643 0.83677321 0.88093691 ... 0.13015408 0.3454521 0.57179104]

In [1266]:

```
#affichage des index avec un z-score supérieur à 3
threshold = 3
print(np.where(z > 3))
```

```
(array([ 6657,  6658,  6682,  6778,  6848,  6849,  6872,  6873,  6896,
       7016,  7087,  7183,  7184,  7231,  7232,  7256,  7280,  7399,
       7472,  7495,  7615,  7616,  7639,  7640,  7654,  7663,  7783,
       7784,  7807,  7808,  7832,  7855,  7856,  7879,  7951,  7952,
       7999,  8000,  8023,  8024,  8047,  8143,  8144,  8191,  8359,
      8360,  8383,  8384,  8407,  8408,  8431,  8432,  8455,  8527,
      8528,  8551,  8552,  8599,  8600,  8623,  8624,  8638,  8647,
      8648,  8671,  8743,  8744,  8767,  8768,  8782,  8791,  8792,
      8815,  8816,  8911,  8912,  8935,  8936,  8959,  8960,  8983,
      8984,  9007,  9151,  9152,  9175,  9176,  9199,  9200,  9223,
      9224,  9266,  9295,  9296,  9310,  9319,  9320,  9334,  9343,
      9344,  9358,  9367,  9368,  9382,  9391,  9392,  9414,  9463,
      9464,  9502,  9511,  9512,  9526,  9535,  9574,  9583,  9584,
      9598,  9607,  9608,  9622,  9631,  9632,  9650,  9727,  9728,
      9742,  9751,  9752,  9766,  9775,  9777,  9790,  9799,  9862,
      9871,  9886,  9895,  9896,  9910,  9919,  9920,  9934,  9943,
      9944, 10518, 10533], dtype=int64),)
```

In [1267]:

```
#On regarde un exemple. Ligne 6657. On peut voir que c'est la columns count qui à un z-score supérieur à 3 pour cette exemple
print(z[6657])
```

3.259409553377711

In [1268]:

```
df_noOutlier.iloc[np.where(z > 3)]
```

Out[1268]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	date
6658	1	0	1	1	28.70	31.820	28	6.0032	782	
6659	1	0	1	1	28.70	31.820	32	15.3600	749	
6683	1	0	1	1	27.06	31.060	44	19.0012	746	
6779	1	0	1	1	26.24	31.060	57	16.9979	801	
6849	2	0	1	1	25.42	30.305	17	12.9980	757	
...
9935	4	0	1	1	18.86	22.725	82	16.9979	834	
9944	4	0	1	2	23.78	27.275	64	22.0028	890	
9945	4	0	1	2	22.96	26.515	64	22.0028	788	
10519	4	0	1	1	21.32	25.000	59	15.0013	743	
10534	4	0	1	2	18.04	21.970	58	19.0012	759	

147 rows × 13 columns

In [1269]:

```
#On a 147 outlier. Pour un dataset de + 10000 lignes c'est bon
df_noOutlier.iloc[np.where(z > 3)].shape[0]
```

Out[1269]:

147

In [1270]:

```
#On supprime les outliers
df_noOutlier = df_noOutlier[(z < 3)]
```

In [1271]:

```
#Nombre de ligne avec outlier supprimé
print(df.shape[0] - df_noOutlier.shape[0])
```

147

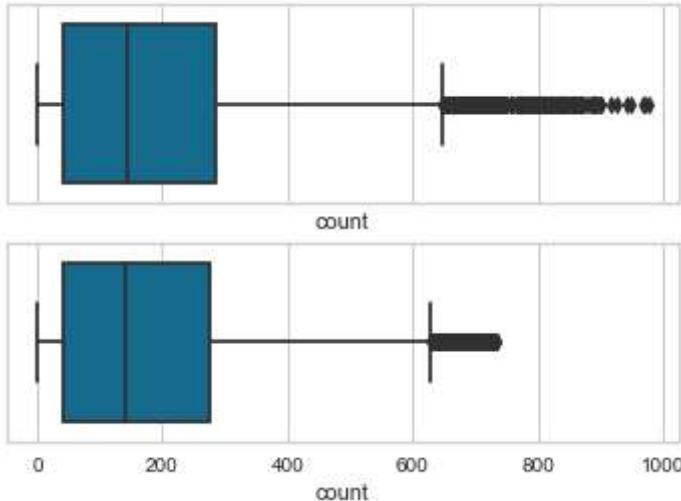
On vérifie le résultat

In [1272]:

```
fig, axes = plt.subplots(2, 1, sharex=True)
sns.boxplot(ax=axes[0], x=df['count'])
sns.boxplot(ax=axes[1], x=df_noOutlier['count'])
```

Out[1272]:

<AxesSubplot:xlabel='count'>



Distribution des variables

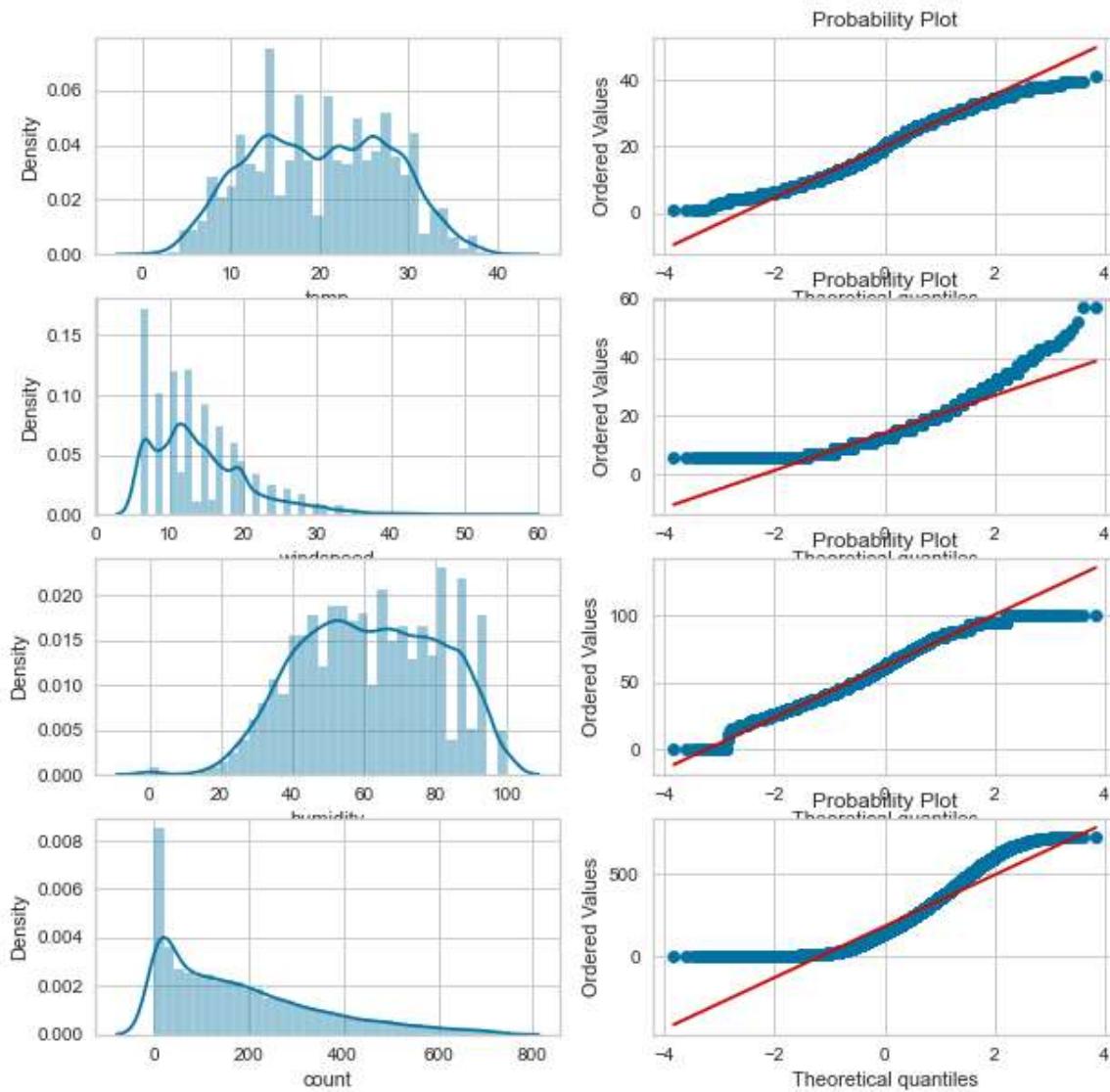
On va vérifier la distribution normal des variables. QQ plot est utilisé pour avoir une vue plus précise que l'histogramme de la distribution des variable avec la loi normal.

In [1273]:

```
fig, axes = plt.subplots(4, 2)
fig.set_size_inches(10,10)
sns.distplot(df_noOutlier["temp"], ax=axes[0,0])
stats.probplot(df_noOutlier["temp"], dist='norm', fit=True, plot=axes[0,1])
sns.distplot(df_noOutlier["windspeed"], ax=axes[1,0])
stats.probplot(df_noOutlier["windspeed"], dist='norm', fit=True, plot=axes[1,1])
sns.distplot(df_noOutlier["humidity"], ax=axes[2,0])
stats.probplot(df_noOutlier["humidity"], dist='norm', fit=True, plot=axes[2,1])
sns.distplot(df_noOutlier["count"], ax=axes[3,0])
stats.probplot(df_noOutlier["count"], dist='norm', fit=True, plot=axes[3,1])
```

Out[1273]:

```
((array([-3.82817384, -3.60399556, -3.48096514, ..., 3.48096514,
       3.60399556, 3.82817384]), array([ 1,  1,  1, ..., 732, 733, 73
4], dtype=int64)), (157.65362658175596, 183.00558763270624, 0.945521243132
703))
```



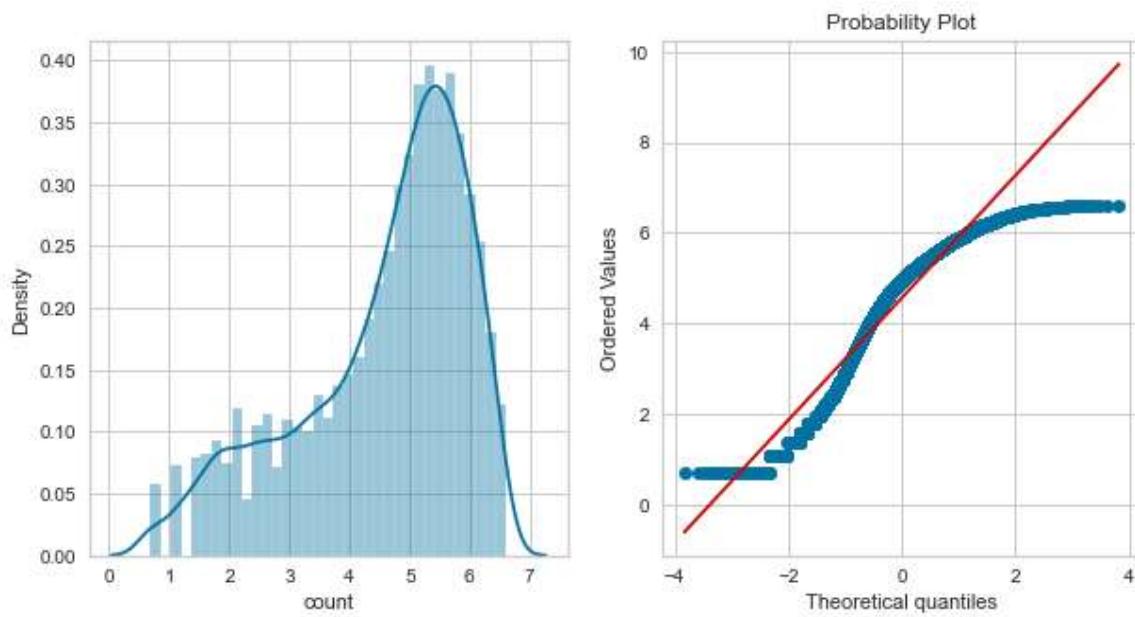
On peut voir que la variable count est écrasé vers la gauche. On peut la normaliser avec la fonction log. pour notre modèle. On peut voir qu'ensuite la distribution fit mieux la loi normal.

In [1274]:

```
fig, axes = plt.subplots(1, 2)
fig.set_size_inches(10,5)
sns.distplot(np.log1p(df_noOutlier["count"]), ax=axes[0])
stats.probplot(np.log1p(df_noOutlier["count"]), dist='norm', fit=True, plot=axes[1])
```

Out[1274]:

```
((array([-3.82817384, -3.60399556, -3.48096514, ..., 3.48096514,
       3.60399556, 3.82817384]), array([0.69314718, 0.69314718, 0.693147
18, ..., 6.5971457 , 6.59850903,
       6.5998705 ])), (1.3487595031437705, 4.562373251436122, 0.9581226674
861731))
```



In [1275]:

```
df_noOutlier_norm = df_noOutlier.copy()
df_noOutlier_norm["count"] = np.log1p(df_noOutlier_norm["count"])
df_norm = df.copy()
df_norm["count"] = np.log1p(df["count"])
```

Models

Multi Linear Regression

Ici on va préparer nos données pour pouvoir utiliser le modèle de Linear Regression de scikit-learn. Notre principale problème sont les variables de type catégorie. On va utiliser différentes méthodes selon que notre catégorie possède un grand nombre de catégories ou non.

In [1276]:

df_noOutlier_norm.head()

Out[1276]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	day
0	1	0	0	1	9.84	14.395	81	14.58	2.833213	
1	1	0	0	1	9.02	13.635	80	14.58	3.713572	
2	1	0	0	1	9.02	13.635	80	14.58	3.496508	
3	1	0	0	1	9.84	14.395	75	14.58	2.639057	
4	1	0	0	1	9.84	14.395	75	14.58	0.693147	

◀ ▶

In [1277]:

df_noOutlier_norm.dtypes

Out[1277]:

```

season          category
holiday         category
workingday      category
weather          category
temp            float64
atemp            float64
humidity        int64
windspeed        float64
count            float64
dayofweek        category
hour             category
month            category
year             category
dtype: object

```

In [1278]:

```

for colname, serie in df_noOutlier_norm.select_dtypes(include=['category']).iteritems():
    print(colname + " has " + str(serie.drop_duplicates().shape[0]) + " unique values.")
)

```

```

season has 4 unique values.
holiday has 2 unique values.
workingday has 2 unique values.
weather has 3 unique values.
dayofweek has 7 unique values.
hour has 24 unique values.
month has 12 unique values.
year has 2 unique values.

```

In [1279]:

```
df_copy = df.copy()
df_norm_copy = df_norm.copy()
df_noOutlier_copy = df_noOutlier.copy()
df_noOutlier_norm_copy = df_noOutlier_norm.copy()
```

In [1280]:

```
categorical_features = ["season", "holiday", "workingday", "weather", "month", "year",
"hour"]
categorical_features_less_dummies = ["season", "holiday", "workingday", "weather"]
```

In [1281]:

```
df_dummies_noOutlier = pd.get_dummies(df_noOutlier_norm_copy[categorical_features])
df_dummies = pd.get_dummies(df_copy[categorical_features])

df_less_dummies_noOutlier = pd.get_dummies(df_noOutlier_norm_copy[categorical_features_
less_dummies])
```

In [1282]:

```
# on concatène ensuite avec le dataset original
df = pd.concat([df_copy.drop(categorical_features, axis=1), df_dummies], axis=1)
df_norm = pd.concat([df_norm_copy.drop(categorical_features, axis=1), df_dummies], axis
=1)
df_noOutlier = pd.concat([df_noOutlier_copy.drop(categorical_features, axis=1), df_dumm
ies_noOutlier], axis=1)
df_noOutlier_norm = pd.concat([df_noOutlier_norm_copy.drop(categorical_features, axis=1
), df_dummies_noOutlier], axis=1)

df_less_dummies_noOutlier = pd.concat([df_noOutlier_copy.drop(categorical_features, axi
s=1), df_dummies_noOutlier], axis=1)
df_less_dummies_noOutlier_norm = pd.concat([df_noOutlier_norm_copy.drop(categorical_fea
tures, axis=1), df_dummies_noOutlier], axis=1)
```

In [1283]:

```
df.head()
```

Out[1283]:

	temp	atemp	humidity	windspeed	count	dayofweek	season_1	season_2	season_3	se
0	9.84	14.395	81	14.58	16	5	1	0	0	0
1	9.02	13.635	80	14.58	40	5	1	0	0	0
2	9.02	13.635	80	14.58	32	5	1	0	0	0
3	9.84	14.395	75	14.58	13	5	1	0	0	0
4	9.84	14.395	75	14.58	1	5	1	0	0	0

5 rows × 55 columns

Multi Linear regression Model

Après de nombreux essaie avec 4 types de datasets:

- Dataframe normal
- Dataframe sans Outlier
- DataFrame sans Outlier et normalisé
- Dataframe normalisé

J'ai regardé les métrics suivantes: MSE, R², RMSE et RMSLE. Mais je vais me concentrer sur MSE est R2. Le meilleur modèles que j'ai entraîné et la version avec le dataframe sans outlier et normalisé. Il semble donc que le modèle de regression linéaire est mieux entraîné avec un dataset sans outlier et normalisé.

Après différente test avec les variables catégoriel, le modèle préfère avoir les variables catégoriques traité avec get_dummies pour Season, Workingday, holydays, mais n'a pas de préférence pour les variable Dates. Traité avec ou sans get_dummies les résultats sont similaires.

On valide: Datasets df_less_dummies_noOutlier_norm MSE 8095.79 RMSE 89.9766 R2 0.696031

Je ne pense pas que ce modèle est adapté à ce problème. En effet sans normalisation des données le modèle prédit des valeurs négatives. En effet comme expliqué ici:

<https://stats.stackexchange.com/questions/145383/getting-negative-predicted-values-after-linear-regression>
[\(https://stats.stackexchange.com/questions/145383/getting-negative-predicted-values-after-linear-regression\)](https://stats.stackexchange.com/questions/145383/getting-negative-predicted-values-after-linear-regression)

"Linear regression does not respect the bounds of 0. It's linear, always and everywhere. It may not be appropriate for values that need to be close to 0 but are strictly positive.".

Etant donnée mes résultats mes nombreux essais peu convaincant j'ai décidé d'arrêter d'abandonner ce modèle. Je pense également que le problème persistera pour les autres modèles de regression linéaire tel que Ridge ou Lasso.

In [1311]:

```
def evaluate(y_test, y_train, y_pred_test, y_pred_train):
    print("MSE test: ", mean_squared_error(y_test, y_pred_test, squared=True))
    print("MSE train: ", mean_squared_error(y_train, y_pred_train, squared=True))
    print("R2 test: ", r2_score(y_test, y_pred_test))
    print("R2 train: ", r2_score(y_train, y_pred_train))
```

In [1312]:

```
#ICI LES RESULTATS SONT MIS A L'ECHELLE.
#Parce que nous avons des datasets avec et sans normalisation. Pour comparer les résultats j'ai dénormalisé y_pred, y_train et
#y_test des dataframe normalisé pour l'évaluation
datasets = [df, df_norm, df_noOutlier, df_noOutlier_norm, df_less_dummies_noOutlier, df_less_dummies_noOutlier_norm]
datasetsLabels = ["df", "df_norm", "df_noOutlier", "df_noOutlier_norm", "df_less_dummies_noOutlier", "df_less_dummies_noOutlier_norm"]
mse = []
r2 = []
d = {}
for i in range(len(datasets)):
    model = LinearRegression()
    X = datasets[i].drop(["count", "atemp"], axis=1)
    y = datasets[i]["count"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=777)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    #Mise à l'échelle des résultats
    if "norm" in datasetsLabels[i]:
        y_test = np.expm1(y_test)
        y_pred = np.expm1(y_pred)
    mse.append(mean_squared_error(y_test, y_pred, squared=True))
    r2.append(r2_score(y_test, y_pred))
d = {'Datasets': datasetsLabels, 'MSE': mse, 'R2': r2}
res_frame = pd.DataFrame(d)
res_frame
```

Out[1312]:

	Datasets	MSE	R2
0	df	10152.449009	0.683313
1	df_norm	9629.286079	0.699632
2	df_noOutlier	8377.452752	0.685455
3	df_noOutlier_norm	8095.793764	0.696031
4	df_less_dummies_noOutlier	8377.452752	0.685455
5	df_less_dummies_noOutlier_norm	8095.793764	0.696031

On valide celui là pour LinearRegression (On cherche à minimiser MSE et maximiser R2):

In [1313]:

res_frame.loc[5]

Out[1313]:

```
Datasets      df_less_dummies_noOutlier_norm
MSE                  8095.79
R2                  0.696031
Name: 5, dtype: object
```

Final result (sans la dénormalisation):

In [1316]:

```
X = df_less_dummies_noOutlier_norm.drop(["count", "atemp"], axis=1)
y = df_less_dummies_noOutlier_norm["count"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=77)
model.fit(X_train,y_train)
y_pred_test=model.predict(X_test)
y_pred_train=model.predict(X_train)
evaluate(y_test, y_train, y_pred_test, y_pred_train)
```

MSE test: 0.3711791898894495
MSE train: 0.3318896388357093
R2 test: 0.8205940488212254
R2 train: 0.8304490688449889

Lasso

On essaye LASSO ici pour mesurer l'apport d'une regularisation en regression linéaire. Nous allons train avec différente valeur d'alpha et en Cross validation.

In [1317]:

```
X = df_less_dummies_noOutlier_norm.drop(["count", "atemp"], axis=1)
y = df_less_dummies_noOutlier_norm["count"]

# Create a list of alphas to cross-validate against
alphas = np.logspace(-10, 1, 400)
```

In [1318]:

#On veut savoir pour quelles valeurs de Alpha avec 5-Fold de validation croisé
#le modèle est plus performant

```
lcv = LassoCV(alphas=alphas, normalize=False, fit_intercept=False, random_state=0, cv=5)
lcv.fit(X, y)
```

Out[1318]:

```
LassoCV(alphas=array([1.0000000e-10, 1.06553795e-10, 1.13537112e-10, 1.20
978102e-10,
1.28906759e-10, 1.37355044e-10, 1.46357012e-10, 1.55948950e-10,
1.66169525e-10, 1.77059935e-10, 1.88664080e-10, 2.01028738e-10,
2.14203749e-10, 2.28242224e-10, 2.43200751e-10, 2.59139630e-10,
2.76123110e-10, 2.94219653e-10, 3.13502206e-10, 3.34048498e-10,
3.55941352e-10, 3.79269019e-1...
4.97441317e+00, 5.30042602e+00, 5.64780507e+00, 6.01795064e+00,
6.41235480e+00, 6.83260739e+00, 7.28040247e+00, 7.75754513e+00,
8.26595874e+00, 8.80769273e+00, 9.38493086e+00, 1.00000000e+01]),
copy_X=True, cv=5, eps=0.001, fit_intercept=False, max_iter=1000,
n_alphas=100, n_jobs=None, normalize=False, positive=False,
precompute='auto', random_state=0, selection='cyclic', tol=0.0001,
verbose=False)
```

In [1319]:

```
#Valeurs d'alpha testé
print(lcv.alphas_)
```

[1.00000000e+01 9.38493086e+00 8.80769273e+00 8.26595874e+00
 7.75754513e+00 7.28040247e+00 6.83260739e+00 6.41235480e+00
 6.01795064e+00 5.64780507e+00 5.30042602e+00 4.97441317e+00
 4.66845237e+00 4.38131027e+00 4.11182940e+00 3.85892347e+00
 3.62157300e+00 3.39882122e+00 3.18977022e+00 2.99357729e+00
 2.80945159e+00 2.63665090e+00 2.47447864e+00 2.32228110e+00
 2.17944475e+00 2.04539383e+00 1.91958797e+00 1.80152004e+00
 1.69071410e+00 1.58672350e+00 1.48912903e+00 1.39753730e+00
 1.31157910e+00 1.23090791e+00 1.15519857e+00 1.08414587e+00
 1.01746340e+00 9.54882369e-01 8.96150502e-01 8.41031051e-01
 7.89301826e-01 7.40754307e-01 6.95192796e-01 6.52433633e-01
 6.12304454e-01 5.74643497e-01 5.39298949e-01 5.06128335e-01
 4.74997943e-01 4.45782286e-01 4.18363594e-01 3.92631340e-01
 3.68481798e-01 3.45817620e-01 3.24547446e-01 3.04585534e-01
 2.85851418e-01 2.68269580e-01 2.51769146e-01 2.36283603e-01
 2.21750528e-01 2.08111337e-01 1.95311051e-01 1.83298071e-01
 1.72023972e-01 1.61443309e-01 1.51513429e-01 1.42194306e-01
 1.33448373e-01 1.25240375e-01 1.17537227e-01 1.10307874e-01
 1.03523178e-01 9.71557865e-02 9.11800339e-02 8.55718314e-02
 8.03085722e-02 7.53690398e-02 7.07333228e-02 6.63827344e-02
 6.22997373e-02 5.84678728e-02 5.48716944e-02 5.14967058e-02
 4.83293024e-02 4.53567162e-02 4.25669645e-02 3.99488019e-02
 3.74916744e-02 3.51856773e-02 3.30215148e-02 3.09904634e-02
 2.90843356e-02 2.72954479e-02 2.56165892e-02 2.40409918e-02
 2.25623046e-02 2.11745669e-02 1.98721847e-02 1.86499079e-02
 1.75028096e-02 1.64262658e-02 1.54159369e-02 1.44677502e-02
 1.35778836e-02 1.27427499e-02 1.19589826e-02 1.12234225e-02
 1.05331045e-02 9.88524571e-03 9.27723476e-03 8.70662068e-03
 8.17110332e-03 7.66852397e-03 7.19685673e-03 6.75420029e-03
 6.33877027e-03 5.94889208e-03 5.58299409e-03 5.23960135e-03
 4.91732965e-03 4.61487988e-03 4.33103286e-03 4.06464440e-03
 3.81464066e-03 3.58001389e-03 3.35981829e-03 3.15316623e-03
 2.95922471e-03 2.77721193e-03 2.60639420e-03 2.44608294e-03
 2.29563192e-03 2.15443469e-03 2.02192206e-03 1.89755988e-03
 1.78084683e-03 1.67131243e-03 1.56851516e-03 1.47204064e-03
 1.38149996e-03 1.29652816e-03 1.21678272e-03 1.14194217e-03
 1.07170483e-03 1.00578757e-03 9.43924684e-04 8.85866790e-04
 8.31379858e-04 7.80244249e-04 7.32253834e-04 6.87215161e-04
 6.44946677e-04 6.05277998e-04 5.68049216e-04 5.33110262e-04
 5.00320295e-04 4.69547138e-04 4.40666743e-04 4.13562692e-04
 3.88125727e-04 3.64253312e-04 3.41849215e-04 3.20823125e-04
 3.01090284e-04 2.82571150e-04 2.65191071e-04 2.48879987e-04
 2.33572147e-04 2.19205845e-04 2.05723170e-04 1.93069773e-04
 1.81194647e-04 1.70049924e-04 1.59590678e-04 1.49774748e-04
 1.40562565e-04 1.31916996e-04 1.23803188e-04 1.16188436e-04
 1.09042044e-04 1.02335205e-04 9.60408821e-05 9.01337039e-05
 8.45898580e-05 7.93869969e-05 7.45041477e-05 6.99216276e-05
 6.56209641e-05 6.15848211e-05 5.77969288e-05 5.42420181e-05
 5.09057590e-05 4.77747029e-05 4.48362284e-05 4.20784904e-05
 3.94903723e-05 3.70614414e-05 3.47819065e-05 3.26425788e-05
 3.06348345e-05 2.87505804e-05 2.69822209e-05 2.53226278e-05
 2.37651111e-05 2.23033925e-05 2.09315797e-05 1.96441428e-05
 1.84358922e-05 1.73019574e-05 1.62377674e-05 1.52390324e-05
 1.43017266e-05 1.34220715e-05 1.25965213e-05 1.18217482e-05
 1.10946289e-05 1.04122326e-05 9.77180827e-06 9.17077451e-06
 8.60670847e-06 8.07733640e-06 7.58052437e-06 7.11426971e-06
 6.67669294e-06 6.26603016e-06 5.88062599e-06 5.51892683e-06
 5.17947468e-06 4.86090118e-06 4.56192215e-06 4.28133240e-06
 4.01800086e-06 3.77086603e-06 3.53893170e-06 3.32126293e-06
 3.11698230e-06 2.92526634e-06 2.74534223e-06 2.57648471e-06
 2.41801308e-06 2.26928856e-06 2.12971163e-06 1.99871964e-06

```

1.87578456e-06 1.76041084e-06 1.65213341e-06 1.55051578e-06
1.45514834e-06 1.36564666e-06 1.28164995e-06 1.20281961e-06
1.12883789e-06 1.05940656e-06 9.94245730e-07 9.33092744e-07
8.75701089e-07 8.21839418e-07 7.71290612e-07 7.23850907e-07
6.79329072e-07 6.37545637e-07 5.98332173e-07 5.61530608e-07
5.26992593e-07 4.94578905e-07 4.64158883e-07 4.35609903e-07
4.08816882e-07 3.83671818e-07 3.60073348e-07 3.37926348e-07
3.17141542e-07 2.97635144e-07 2.79328525e-07 2.62147890e-07
2.46023982e-07 2.30891806e-07 2.16690364e-07 2.03362408e-07
1.90854214e-07 1.79115361e-07 1.68098528e-07 1.57759306e-07
1.48056018e-07 1.38949549e-07 1.30403192e-07 1.22382494e-07
1.14855124e-07 1.07790740e-07 1.01160864e-07 9.49387718e-08
8.90993810e-08 8.36191530e-08 7.84759970e-08 7.36491807e-08
6.91192469e-08 6.48679353e-08 6.08781089e-08 5.71336843e-08
5.36195677e-08 5.03215936e-08 4.72264677e-08 4.43217134e-08
4.15956216e-08 3.90372033e-08 3.66361454e-08 3.43827692e-08
3.22679912e-08 3.02832867e-08 2.84206552e-08 2.66725884e-08
2.50320398e-08 2.34923963e-08 2.20474515e-08 2.06913808e-08
1.94187178e-08 1.82243324e-08 1.71034100e-08 1.60514320e-08
1.50641580e-08 1.41376081e-08 1.32680475e-08 1.24519708e-08
1.16860886e-08 1.09673133e-08 1.02927477e-08 9.65967258e-09
9.06553593e-09 8.50794280e-09 7.98464550e-09 7.49353460e-09
7.03263041e-09 6.60007502e-09 6.19412478e-09 5.81314328e-09
5.45559478e-09 5.12003798e-09 4.80512025e-09 4.50957214e-09
4.23220227e-09 3.97189257e-09 3.72759372e-09 3.49832094e-09
3.28315001e-09 3.08121359e-09 2.89169765e-09 2.71383825e-09
2.54691844e-09 2.39026535e-09 2.24324750e-09 2.10527227e-09
1.97578347e-09 1.85425913e-09 1.74020937e-09 1.63317447e-09
1.53272295e-09 1.43844989e-09 1.34997528e-09 1.26694246e-09
1.18901674e-09 1.11588399e-09 1.04724941e-09 9.82836333e-10
9.22385104e-10 8.65652043e-10 8.12408458e-10 7.62439721e-10
7.15544407e-10 6.71533479e-10 6.30229527e-10 5.91466054e-10
5.55086803e-10 5.20945127e-10 4.88903400e-10 4.58832461e-10
4.30611093e-10 4.04125533e-10 3.79269019e-10 3.55941352e-10
3.34048498e-10 3.13502206e-10 2.94219653e-10 2.76123110e-10
2.59139630e-10 2.43200751e-10 2.28242224e-10 2.14203749e-10
2.01028738e-10 1.88664080e-10 1.77059935e-10 1.66169525e-10
1.55948950e-10 1.46357012e-10 1.37355044e-10 1.28906759e-10
1.20978102e-10 1.13537112e-10 1.06553795e-10 1.00000000e-10]

```

In [1320]:

```
#valeurs des MSE en validation croisée
print(lcv.mse_path_)
```

```

[[2.95030792 2.50685844 3.689098 3.9611766 3.27061375]
 [2.84918822 2.43045027 3.67014343 3.75658626 3.18708884]
 [2.72625975 2.36664634 3.58493879 3.57535009 3.11351296]
 ...
 [0.46621246 0.36408057 0.41490026 0.34868193 0.3302701 ]
 [0.46621246 0.36408057 0.41490026 0.34868193 0.3302701 ]
 [0.46621246 0.36408057 0.41490026 0.34868193 0.3302701 ]]

```

In [1321]:

```
#moyenne mse en validation croisée pour chaque alpha
avg_mse = np.mean(lcv.mse_path_,axis=1)
#alphas vs. MSE en cross-validation
lasso_df = pd.DataFrame({'alpha':lcv.alphas_,'MSE':avg_mse})
print(lasso_df)
```

	alpha	MSE
0	1.000000e+01	3.275611
1	9.384931e+00	3.178691
2	8.807693e+00	3.073342
3	8.265959e+00	2.978186
4	7.757545e+00	2.894696
..
395	1.289068e-10	0.384829
396	1.209781e-10	0.384829
397	1.135371e-10	0.384829
398	1.065538e-10	0.384829
399	1.000000e-10	0.384829

[400 rows x 2 columns]

In [1322]:

```
lasso_df[lasso_df.MSE <=0.39]
```

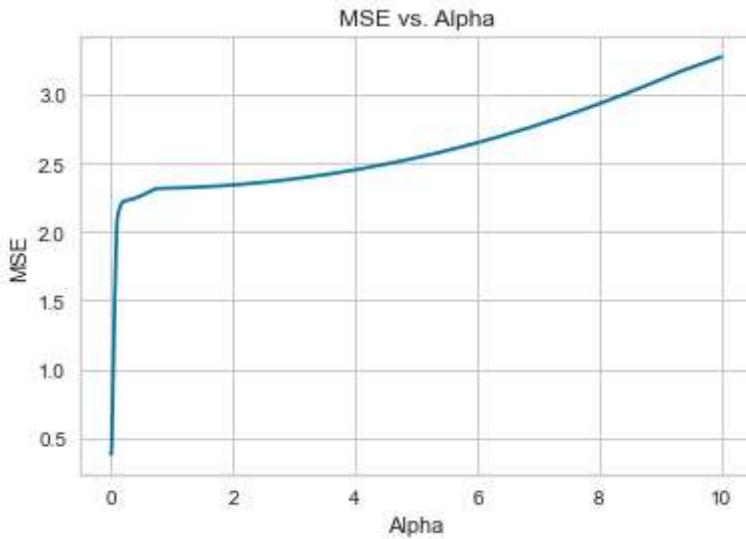
Out[1322]:

	alpha	MSE
120	4.917330e-03	0.388004
121	4.614880e-03	0.386127
122	4.331033e-03	0.384551
123	4.064644e-03	0.383194
124	3.814641e-03	0.382084
..
395	1.289068e-10	0.384829
396	1.209781e-10	0.384829
397	1.135371e-10	0.384829
398	1.065538e-10	0.384829
399	1.000000e-10	0.384829

280 rows x 2 columns

In [1323]:

```
#sous-forme graphique
plt.plot(lcv.alphas_,avg_mse)
plt.xlabel('Alpha')
plt.ylabel('MSE')
plt.title('MSE vs. Alpha')
plt.show()
```



In [1324]:

```
#best alpha
print(lcv.alpha_) #0.01
```

0.0022956319242369095

In [1326]:

```
y_pred_test=lcv.predict(X_test)
y_pred_train=lcv.predict(X_train)
evaluate(y_test, y_train, y_pred_test, y_pred_train)
```

MSE test: 0.37775841903722646
MSE train: 0.3388249532207634
R2 test: 0.8174140406326424
R2 train: 0.8269060567281789

On constate que les performances sont sensiblement similaire à notre modèles de regression linéaire sans régularisation. Je pense chercher à explorer d'autre type d'algorythme. Sans amélioration avec ce modèle je pense que essayer Ridge ou ElasticNet (qui utilise les deux régularization L1 et L2) est une perte de temps.

Random Forest

Je pense que ce seraient une bonne idée de tester les forets aléatoires. Car cette algo n'est pas de la même famille que LinearRegression, Lasso et Ridge et il est populaire pour les problèmes de regression.

In [1327]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=77)
```

In [1328]:

```
regressor = RandomForestRegressor(n_estimators=20, random_state=0)
regressor.fit(X_train, y_train)
```

Out[1328]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=
None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=20, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [1330]:

```
y_pred_test=regressor.predict(X_test)
y_pred_train=regressor.predict(X_train)
evaluate(y_test, y_train, y_pred_test, y_pred_train)
```

```
MSE test:  0.123688954774135
MSE train:  0.017845564384944043
R2 test:  0.9402161421011062
R2 train:  0.9908833187168228
```

En quelques lignes le modèle affiche déjà de très bonne performance comparé à nos modèle précédents. Je vais aller plus en profondeur en tunant et utilisant la cross validation. (Même si les random forest sont moins sujet au problème d'overfitting)

On regarde les paramètres de notre modèles:

In [1331]:

```
from pprint import pprint
```

In [1332]:

```
pprint(model.get_params())
```

```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normalize': False}
```

n_estimators = number of trees in the forest

max_features = max number of features considered for splitting a node

max_depth = max number of levels in each decision tree

min_samples_split = min number of data points placed in a node before the node is split

min_samples_leaf = min number of data points allowed in a leaf node

bootstrap = method for sampling data points (with or without replacement)

Je vais créer une liste de paramètre à essayer

In [1333]:

```
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 150, num = 10)]
max_features = ['auto', 'sqrt']

max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)

min_samples_split = [2, 5, 10]

min_samples_leaf = [1, 2, 4]

bootstrap = [True, False]
```

In [1334]:

```
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [1335]:

```
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [10, 25, 41, 56, 72, 87, 103, 118, 134, 150]}
```

On va essayer toutes c'est combinaison pour espérer trouver les meilleurs paramètres avec notre randomgrid

In [1336]:

```
X = df_less_dummies_noOutlier_norm.drop(["count", "atemp"], axis=1)
y = df_less_dummies_noOutlier_norm["count"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=77)
```

In [1337]:

```
rf = RandomForestRegressor()
# Random search de paramètre avec 3-Fold Cross validation
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit
rf_random.fit(X_train, y_train)
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 25 tasks      | elapsed:  14.5s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  2.4min finished
```

Out[1337]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                    estimator=RandomForestRegressor(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    criterion='mse',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs=None, oob_score=False...),
                    iid='deprecated', n_iter=100, n_jobs=-1,
                    param_distributions={'bootstrap': [True, False],
                                         'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
                                         'max_features': ['auto', 'sqrt'],
                                         'min_samples_leaf': [1, 2, 4],
                                         'min_samples_split': [2, 5, 10],
                                         'n_estimators': [10, 25, 41, 56, 72, 87, 103, 118, 134, 150]},
                    pre_dispatch='2*n_jobs', random_state=42, refit=True,
                    return_train_score=False, scoring=None, verbose=2)
```

In [1338]:

```
#Meilleurs paramètres  
rf_random.best_params_
```

Out[1338]:

```
{'n_estimators': 118, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 90, 'bootstrap': True}
```

In [1340]:

```
best_grid = grid_search.best_estimator_  
y_pred_test = best_grid.predict(X_test)  
y_pred_train = best_grid.predict(X_train)  
evaluate(y_test, y_train, y_pred_test, y_pred_train)
```

```
MSE test:  0.12224363050522728  
MSE train:  0.018052085948676997  
R2 test:  0.9409146972574921  
R2 train:  0.9907778139967678
```

Nous avons une petite améliorations négligeable et une légère diminution du MSE. On peut tenter de raffiner les paramètres avec une GridSearch centré autour des best_parameters précédents. Mais dans notre cas je pense pas que nous allons améliorer le model.

In [1341]:

```
param_grid = {  
    'bootstrap': [True],  
    'max_depth': [80, 90, 100, 110],  
    'max_features': ['auto'],  
    'min_samples_leaf': [1, 2, 3],  
    'min_samples_split': [1, 2, 3],  
    'n_estimators': [90, 95, 103, 110, 115]  
}  
  
rf = RandomForestRegressor()  
  
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,  
                           cv = 3, n_jobs = -1, verbose = 2)
```

In [1342]:

```
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Fitting 3 folds for each of 180 candidates, totalling 540 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  23.7s
[Parallel(n_jobs=-1)]: Done 162 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 365 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done 540 out of 540 | elapsed:  6.0min finished
```

Out[1342]:

```
{'bootstrap': True, 'max_depth': 100, 'max_features': 'auto', 'min_samples
_leaf': 1, 'min_samples_split': 2, 'n_estimators': 110}
```

In [1343]:

```
best_grid = grid_search.best_estimator_
y_pred_test = best_grid.predict(X_test)
y_pred_train = best_grid.predict(X_train)
evaluate(y_test, y_train, y_pred_test, y_pred_train)
```

MSE test: 0.12086349711954356

MSE train: 0.01519840026182826

R2 test: 0.9415817716774937

R2 train: 0.9922356632599334

Malgrés que le fine hyperparameter tuning n'a pas donné de meilleur résultat on peut considerer que ce modèle seraient le modèle à envoyer en production étant donnée sa bonne performance et le fait qu'il n'overfit pas nos données de train.

J'ai cru aussi comprendre que XGBOOST était une version améliorée et très utilisée de RandomForest il serait donc intéressant d'essayer.

Autres méthodes et modèles

Ici je veux essayer une librairie que j'ai trouvée par la suite qui permet de facilement tester, tuner et évaluer un grand nombre de modèles sur un dataset. Afin de faire une première passe qui va nous faciliter notre travail de recherche.

<https://medium.com/@shekharshashank1/2-words-code-to-compare-20-ml-regression-models-with-pycaret-8ed70c62a6b7> (<https://medium.com/@shekharshashank1/2-words-code-to-compare-20-ml-regression-models-with-pycaret-8ed70c62a6b7>)

In [1349]:

```
x_train,x_test,y_train,y_test=train_test_split(df_less_dummies_noOutlier_norm.drop('cou
nt',axis=1),df_less_dummies_noOutlier['count'],test_size=0.25,random_state=42)
```

In [1350]:

```
caret_df = setup(data = df_less_dummies_noOutlier_norm, target = "count", session_id=55 )
```

	Description	Value
0	session_id	55
1	Target	count
2	Original Data	(10738, 55)
3	Missing Values	False
4	Numeric Features	53
5	Categorical Features	1
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(7516, 57)
10	Transformed Test Set	(3222, 57)
11	Shuffle Train-Test	True
12	Stratify Train-Test	False
13	Fold Generator	KFold
14	Fold Number	10
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False
18	Experiment Name	reg-default-name
19	USI	939f
20	Imputation Type	simple
21	Iterative Imputation Iteration	None
22	Numeric Imputer	mean
23	Iterative Imputation Numeric Model	None
24	Categorical Imputer	constant
25	Iterative Imputation Categorical Model	None
26	Unknown Categoricals Handling	least_frequent
27	Normalize	False
28	Normalize Method	None
29	Transformation	False
30	Transformation Method	None
31	PCA	False
32	PCA Method	None
33	PCA Components	None
34	Ignore Low Variance	False
35	Combine Rare Levels	False
36	Rare Level Threshold	None
37	Numeric Binning	False

	Description	Value
38	Remove Outliers	False
39	Outliers Threshold	None
40	Remove Multicollinearity	False
41	Multicollinearity Threshold	None
42	Clustering	False
43	Clustering Iteration	None
44	Polynomial Features	False
45	Polynomial Degree	None
46	Trigonometry Features	False
47	Polynomial Threshold	None
48	Group Features	False
49	Feature Selection	False
50	Features Selection Threshold	None
51	Feature Interaction	False
52	Feature Ratio	False
53	Interaction Threshold	None
54	Transform Target	False
55	Transform Target Method	box-cox

In [1351]:

```
#Comparer 25models de base  
compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
catboost	CatBoost Regressor	0.1962	0.0858	0.2926	0.9562	0.0799	0.0671
lightgbm	Light Gradient Boosting Machine	0.2271	0.1047	0.3234	0.9465	0.0848	0.0753
xgboost	Extreme Gradient Boosting	0.2245	0.1065	0.3261	0.9456	0.0871	0.0746
rf	Random Forest Regressor	0.2354	0.1214	0.3479	0.9381	0.0907	0.0785
et	Extra Trees Regressor	0.2283	0.1222	0.3492	0.9377	0.0952	0.0785
gbr	Gradient Boosting Regressor	0.3646	0.2304	0.4797	0.8824	0.1126	0.1122
dt	Decision Tree Regressor	0.3204	0.2393	0.4887	0.8781	0.1251	0.1034
lr	Linear Regression	0.4373	0.3432	0.5855	0.8247	0.1387	0.1354
ridge	Ridge Regression	0.4375	0.3432	0.5856	0.8247	0.1387	0.1357
br	Bayesian Ridge	0.4374	0.3432	0.5855	0.8247	0.1387	0.1356
huber	Huber Regressor	0.4597	0.4132	0.6422	0.7889	0.1478	0.1437
par	Passive Aggressive Regressor	0.5510	0.5322	0.7166	0.7258	0.1675	0.1764
omp	Orthogonal Matching Pursuit	0.7013	0.8320	0.9117	0.5754	0.1960	0.2106
ada	AdaBoost Regressor	0.9171	1.1145	1.0554	0.4318	0.2198	0.2565
knn	K Neighbors Regressor	0.9270	1.4880	1.2193	0.2415	0.2739	0.3122
en	Elastic Net	0.9650	1.5174	1.2314	0.2267	0.2808	0.3412
lasso	Lasso Regression	0.9711	1.5291	1.2362	0.2208	0.2823	0.3449
llar	Lasso Least Angle Regression	1.1379	1.9633	1.4009	-0.0007	0.3123	0.4015
lar	Least Angle Regression	4245.5833	416401738.6108	6455.4682	-203872308.7183	1.1242	1176.41


Out[1351]:

```
<catboost.core.CatBoostRegressor object at 0x000001C23141F1C0>
```

In [1352]:

```
#Selection du meilleur modèle et fine tuning automatique
catboost_reg = create_model("catboost")
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	0.2044	0.0888	0.2979	0.9576	0.0853	0.0734
1	0.1895	0.0786	0.2804	0.9608	0.0767	0.0641
2	0.1975	0.0885	0.2974	0.9537	0.0805	0.0678
3	0.1805	0.0675	0.2598	0.9641	0.0692	0.0590
4	0.1972	0.0888	0.2981	0.9538	0.0822	0.0705
5	0.1962	0.0852	0.2920	0.9583	0.0813	0.0693
6	0.2037	0.0880	0.2966	0.9569	0.0862	0.0735
7	0.1921	0.0897	0.2994	0.9546	0.0772	0.0605
8	0.2021	0.0888	0.2980	0.9515	0.0772	0.0630
9	0.1989	0.0941	0.3068	0.9506	0.0835	0.0698
Mean	0.1962	0.0858	0.2926	0.9562	0.0799	0.0671
SD	0.0069	0.0071	0.0126	0.0040	0.0048	0.0049

In [1353]:

```
#Paramètres utilisé  
evaluate_model(catboost_reg)
```

Parameters	
nan_mode	Min
eval_metric	RMSE
iterations	1000
sampling_frequency	PerTree
leaf_estimation_method	Newton
grow_policy	SymmetricTree
penalties_coefficient	1
boosting_type	Plain
model_shrink_mode	Constant
feature_border_type	GreedyLogSum
bayesian_matrix_reg	0.10000000149011612
l2_leaf_reg	3
random_strength	1
rsm	1
boost_from_average	True
model_size_reg	0.5
subsample	0.800000011920929
use_best_model	False
random_seed	55
depth	6
posterior_sampling	False
border_count	254
classes_count	0
auto_class_weights	None
sparse_features_conflict_fraction	0
leaf_estimation_backtracking	AnyImprovement
best_model_min_trees	1
model_shrink_rate	0
min_data_in_leaf	1
loss_function	RMSE
learning_rate	0.05588899925351143
score_function	Cosine
task_type	CPU
leaf_estimation_iterations	1
bootstrap_type	MVS
max_leaves	64

On retient donc CatBoost Regressor ici (je ne connaissait pas ce modèle).