Python for Data Analysis

BRUNNER Nicolas DIA2

1) Discovery of the dataset

The dataset is named **Facebook Comment Volume Dataset Data Set**, and is composed of 54 integeer/float attributes and the associated task is **regression.**

In fact, the main goal of the dataset is trying to **predict the exact number of comments** in the **next day** under a **Facebook publication**.

To do so, the author provides us multiple features.

	likes	Checkins	Returns	Category
0	634995	0	463	1
1	634995	0	463	1
2	634995	0	463	1
3	634995	0	463	1
4	634995	0	463	1
5	634995	0	463	1
6	634995	0	463	1
7	634995	0	463	1
8	634995	0	463	1
9	634995	0	463	1

- The **like** columns represent the number of like on a specific Facebook page (it's actually the number of people that follows the page).

- **Checkins** is the number of people that visited the place so far (only for real places like theater, institutions etc..).

- **Returns** is the number of people that came back to the page after liking it.

- **Category** is the category of the Facebook page (brand, place, institution). From 1 to 9.

Here, we have the same values for each row because it's the same page, but different post from the page.

Then, the features from 5 to 29 are features aggregated by page, by calculating min, max, average, median and standard deviation of essential features of the dataset.

X	X.1	X.2	X.3	X.4	X.5	X.6	X.7
0.0	806.0	11.291045	1.0	70.495138	0.0	806.0	7.574627
0.0	806.0	11.291045	1.0	70.495138	0.0	806.0	7.574627
0.0	806.0	11.291045	1.0	70.495138	0.0	806.0	7.574627
0.0	806.0	11.291045	1.0	70.495138	0.0	806.0	7.574627

Here, I only show the 7 first features.

So, for now, we only have features that describes the page, and not the posts. But what interested us is the number of commentaires under each posts of the page.

Then, we finally have features that allows us to differentiate the post and the Facebook page associated.

commBase	comm24	comm48	comm24_1	diff_24.48	baseTime	length	shares	promoted	hrs
0	0	0	0	0	65	166	2	0	24
0	0	0	0	0	10	132	1	0	24
0	0	0	0	0	14	133	2	0	24
7	0	3	7	-3	62	131	1	0	24
1	0	0	1	0	58	142	5	0	24
0	0	0	0	0	60	166	1	0	24
0	0	0	0	0	68	145	2	0	24
1	0	1	1	-1	32	157	2	0	24

- **commBase** is the total number of comments before selected base date/time.
- **comm24** the number of comments in last 24 hours, relative to base date/time.
- **comm48** the number of comments in last 48 to last 24 hours relative to base date/time.
- **comm24_1** the number of comments in the first 24h after the publication.
- **diff_24.48** the difference between **comm24** and **comm48**.
- **baseTime** is the selected time in order to simulate the scenario in hours.
- Then, we have the **length** and **shares** of the post.
- **Promoted** is a boolean, 0 if the post wasn't promoted by Facebook, and 1 if it was.
- **hrs** is the number of hour for which we have the comments received.

After that we have features that give us the exact day of the publication.

sun_pub	mon_pub	tue_pub	wed_pub	thu_pub	fri_pub	sat_pub
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	1	0
0	1	0	0	0	0	0

And, just after that we have the day where the data were collect.

sun_base	mon_base	tue_base	wed_base	thu_base	fri_base	sat_base
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	0	0	1

5
0
0
0
0
2

And finally we have the **output** !
The output is the number of comments under the selected Facebook post.

So, we indeed have a **regression** problem here because we want to predict the exact amount of comments.

We can now start working on it !

2) Quick information/sanity check on the dataset

The shape of the dataset is : **(40949, 54).**
So we have **40949 samples**, which is great to train a good model. But we also have **54 features**, which is maybe a little bit too exagereted, maybe we could delete some later.

After that, I checked if there were any missing values in the dataset with the line : **df.isnull().any().**
Good news, the dataset is clean.

Then, I checked the type of each features, there are all **continuous** features, non of them is **categorical**. Here, we already have information about what type of model we are going to select.

Now that the dataset is usable, let's push things a little bit further.

3) Updating the dataset

Here, I will check if all of the 54 features are usable, or if we can get rid of them.

First, the **promoted** features, after checking it, non of the 41000 posts are promoted.
So, wan can clearly delete these feature because it will not help us to predict something.

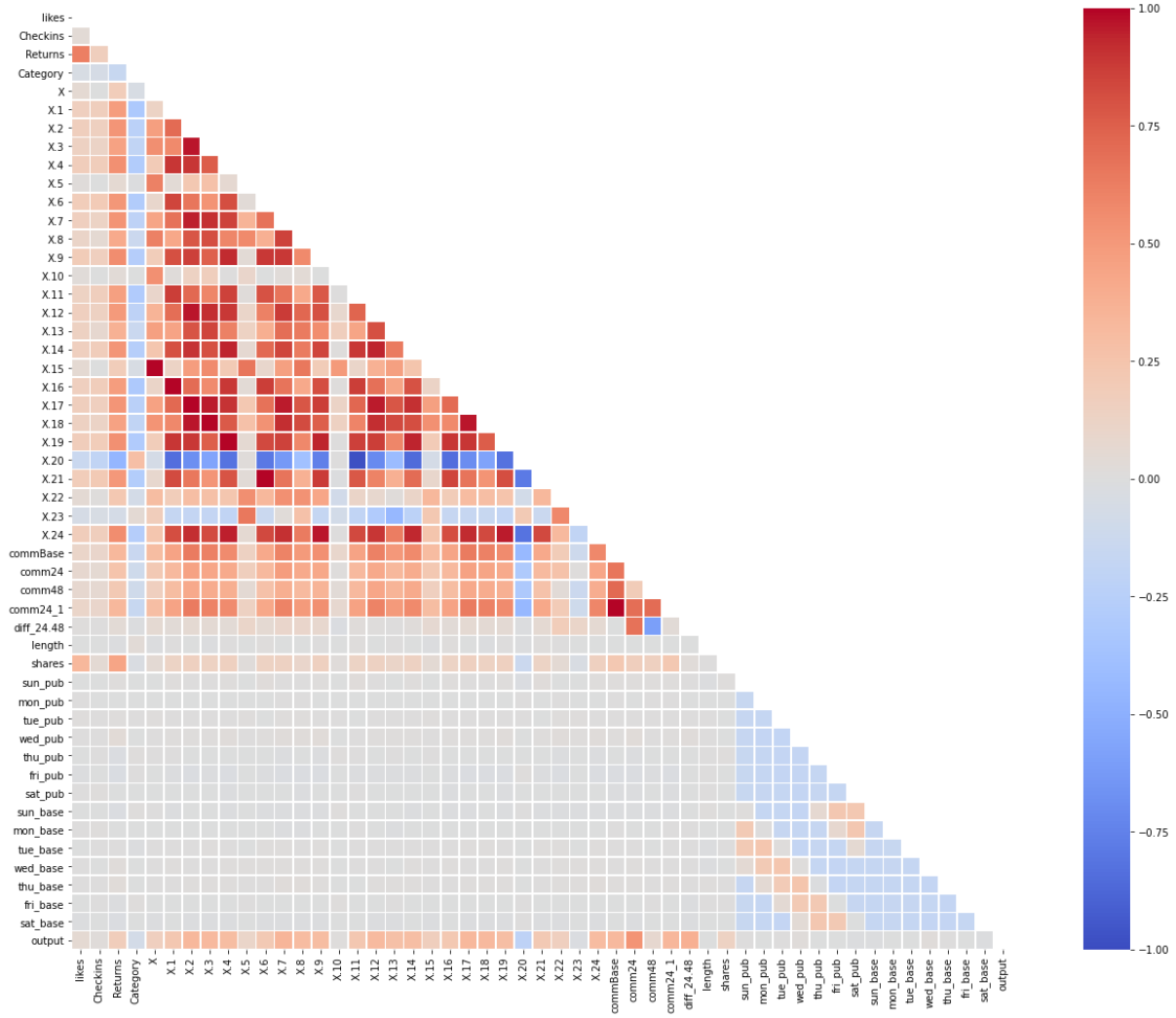
After that, the **hrs** variable is almost always at the value '24', except for 778 of the 41000 samples, which correspond to less than 2% of the values. It will mean that the author of the dataset record the number of comment under this specific post for 3 hours of example, instead of 24 hours. Obviously, except if we normalize our dataset, it has no sense to compare those results. And finally, the correlation between **hrs** and the **output** is 0.0125.
This is why I decided to delete the variable **hrs** from the original dataset.

The final feature that I decided to remove is the **baseTime** one.
I didn't really understand the aim of it. It is range for values from 0 to 71. According to the website where we are downloading the dataset, it is says that it is "*Selected time in order to simulate the scenario*".
I tried to contact the author of the dataset for a more precise description but he/she didn't respond.
Finally, **baseTime** and **output** are only -0.22 correlated.

So, I have decided to delete this features because I don't understand it, and it is not the most important one.

Now that all the ‘weird’ features are gone, we can plot the **correlation matrix** in order to see which feature is important, and which is not.

This, is what I get



It is basically all of the X features that are highly correlated between them, because they are probably linear combination or logarithmic regression. For example, maybe X24=log(X21)-X5.

And, if I zoom on the **output** line, this is what we get :

We can clearly see that all the last features about the day of publication and the baseTime day aren’t correlated at all with the output. So, I removed them because it was 14 features

which would have taken too much importance in our futur models, and add too much time complexity for few results.

So, this is the final output line in our correlation matrix of the updated dataset.

We can see that the most correlated feature is **comm24**, which is explainable because the more comments the post receive at it publication, the more trendy the post will be.

All of the X features are a little bit correlated, so for now I will keep them, maybe I will try to remove them after to see if it impact the results.

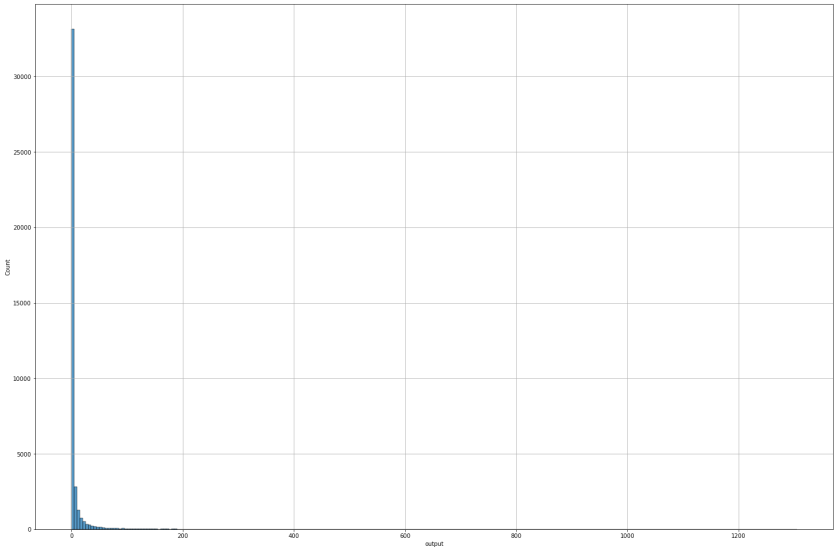
Likes aren't that correlated but it is because the number of likes correponds to the number of follower on the page, and not the number of likes of the post.
Same for **Category, Checkins, Returns**, all of the page-based features are few correlated with the **output**.

But, the **post-based** features like **commeBase, comm24, etc..** are well correlated.

4) Data Visualization with Seaborn

Here, we will see the link between the **output** and other features.

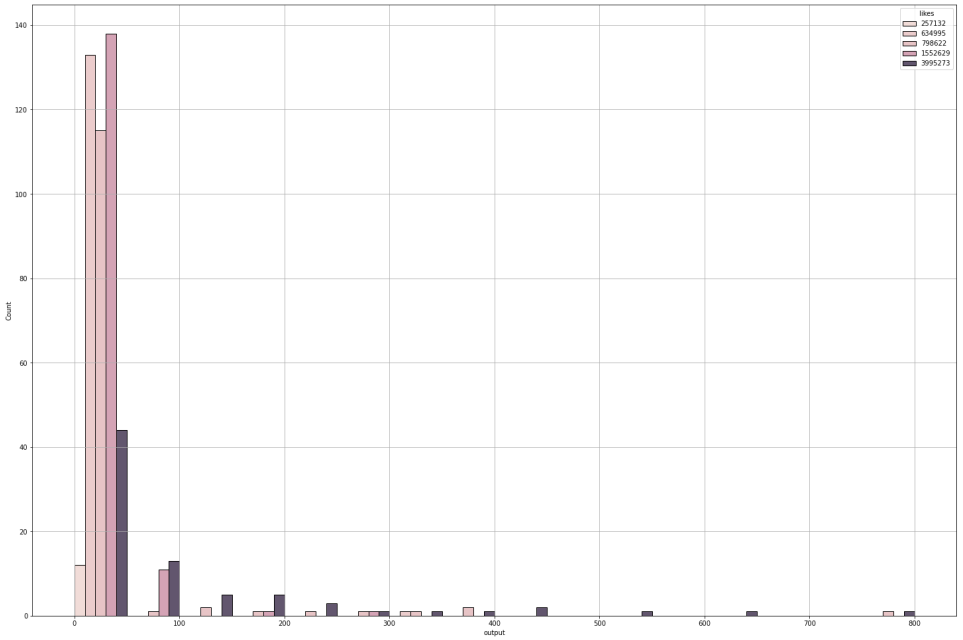
First, I did a quick **histogram** of the **output** :



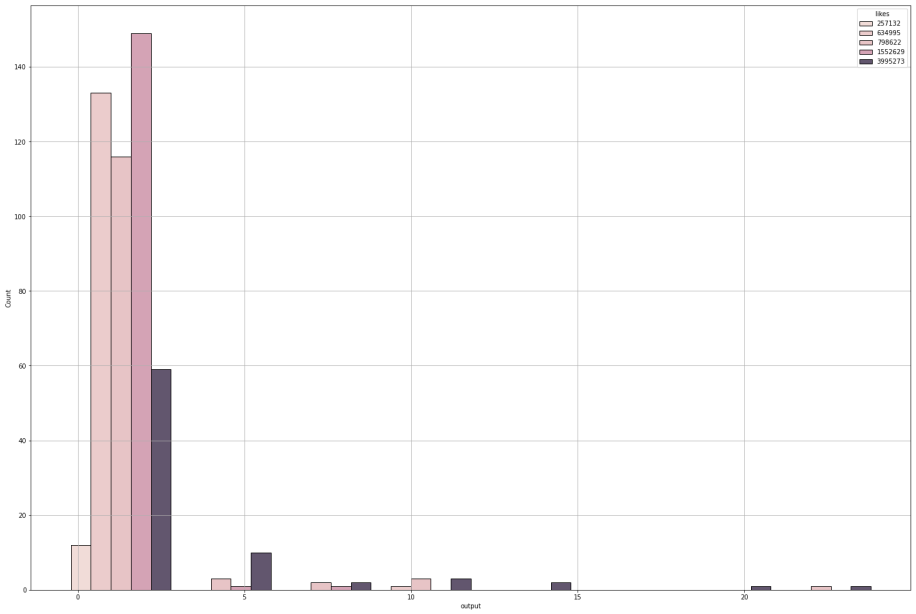
And we can clearly see that most of the post didn't receive any comments.
In fact, if we use the **value_count()** function on our dataset, we can see that 22000 posts aren't commented at all. And the rest of them hava a very few comments.

So, it is very probable that our model will predict wrong number of comments for higly commented posts because most of the post are few commented. Indeed, with the **describe()** function, we can see that 75% of the posts have 3 or less comments.

To see the disparity of comments between different pages we can plot histograms of differents pages on the same graph.
Here, I only represented 3 pages because if we do more, the results is unreadble.
But, we can see that globally, pages have the same distributions of comments under there post. Most of there publications have no comments, but a few of them are highly commentated.

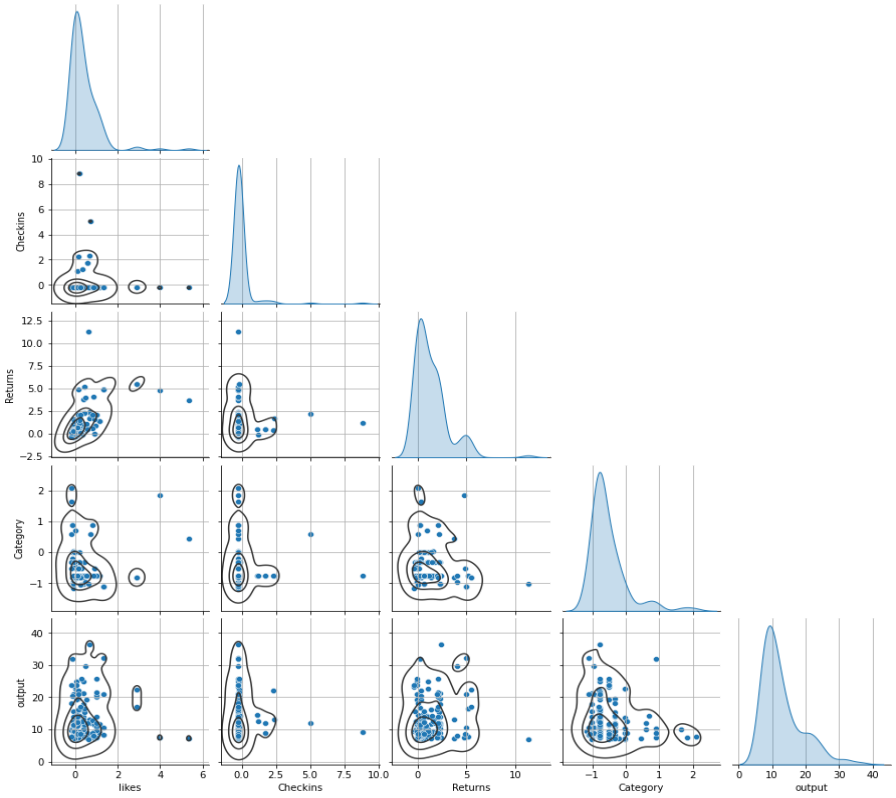


This plot will be more significant if we normalize the data, and if we do that, this is what we get :



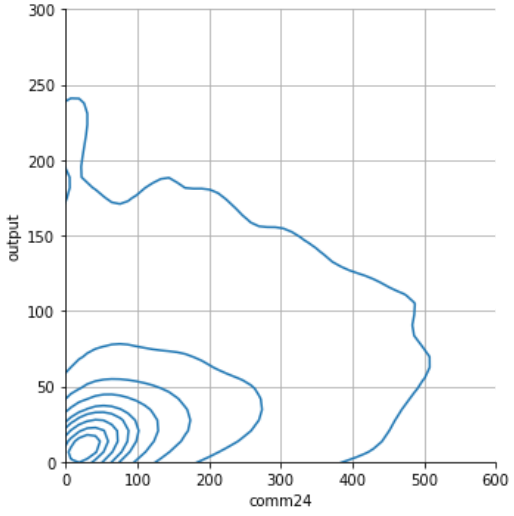
So, we do have the same tendency than above.
Posts are indeed few commentated.

Then, we can try to visualize the link between the **page-related features** and the **output.**
So, we define **page_related_df**=df[["likes","Checkins","Returns","Category"]].
And we can do a pairplot.

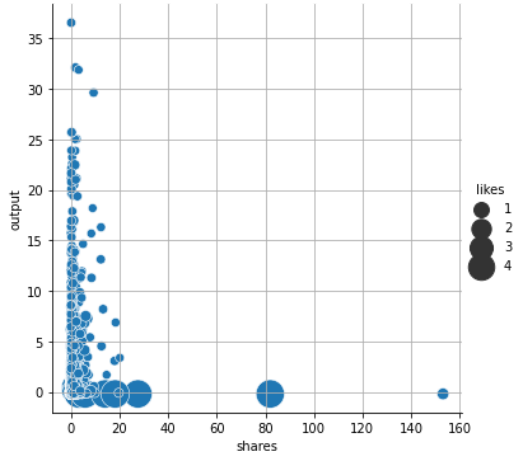


What we can see here if we look at the last line is that more **likes** on the page doesn't mean more **comments** at all.
Same observation for **checkins, returns and category.**
We can conclude that those **page-related features** are, in agreement with the correlation matrix, not very important for our model.

Now, we can check the relation between **post-related features** and the **output**.



First, with the most correlated feature, **comm24** (the number of comment in the first 24h), we can see here that globaly, the more **comm24** is high, the more the output tends to be higher, even if it is not blatant.

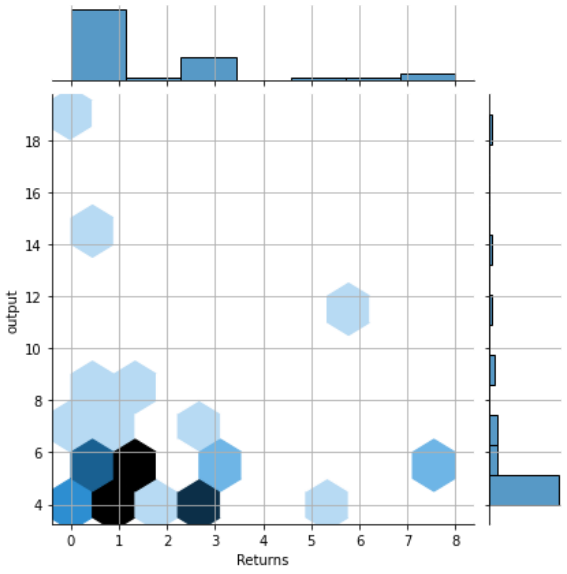


Then, for the **shares,** I plot :

The bigger the 'bubbles' are, the more **likes** the publication receive. And with this graph we can see that **shares** and **likes** can't explain the output well.

Same observation here with **returns**, we can't make any assumptions here.

What I can assume now is that none of the features will be a **decisive feature** in the prediction because they are all few correlated.

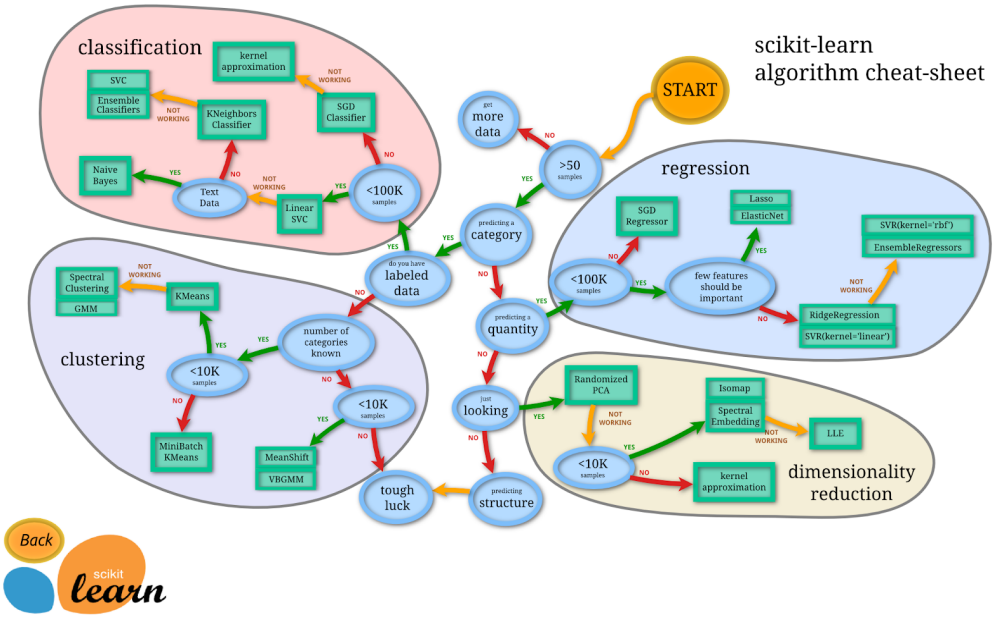


5) Model selection

Now, we will try some models from **sklearn**, test them, look a the result they give and try to adjust them.

But first, let's define **X** and **y**.
X will be the 36 features that we kept and **y** will be our output. Then, by using the **train_test_split()** function from skearn, we can define **X_train, X_test, y_train, y_test**.
I decided to kept 30% of the data to measure the efficiency of each model.

To help me find the right model for my problem, sklearn post this 'help' picture.
So, we are in the top right corner.



First, I will try all the models without tuning them, just to see if the model is usable or not. Then, I will do a **GridSearchCV** to select the best of the best.

So, the first model we could try is **SVR**(Epsilon-Support Vector Regression).
The issue with this model is the fit time complexity, which is more than quadratic. If our dataset, is too big, sklearn tell us to use **LinearSVR** or **SGDRegressor**, but after trying them, the results were extremelly bad. This is why I kept **SVR**.
So, without tuning anything, the **SVR** give us a **MSE** equal to **1000** approximately and a **RMSE** around **32**.

The second model is **Ridge**, this model solves a regression model where the loss function is the linear least squares function. The **Ridge** model give us better result as the **MSE** is arround the **730** and the **RMSE** arround **26-27**.

The third one is **ElasticNet**, it gives similar result because the model is like the **Ridge** one. So, we get a **MSE** arround **725** and a **RMSE** arround **26**.

I will then try the **RandomForestRegressor**, we have **MSE** arround the 1600 and a **RMSE** arround the **40**.

Finally, I will try **KNNRegressor,** we obrain a **RMSE** around 30 also.

So, without tuning any model, we can see that best ones are **Ridge** and **ElasticNet**. But, we get a **RMSE** arround **26**, which is not that great. Indeed, the **maximum** output in our dataset is **1305** comments, the **minimum** is **0**, the **mean** is 7**,** the **3rd quartile** is **3** and the **std** is **35**. So, our model give us result that are globally 26 comments away from the real result. This can be explain because 85% of the dataset has low values (under 10 comments), and for the few high values, they are only a few of them, so it is difficult to predict precisely the exact number.

6) GridSearch

Here, I will create a **GridSearchCV**, a grid search that also apply **cross-validation** in order to avoid overfitting because it is very easy to overfit here. For example, I ran a **RandomForest** with max_depth set to 20000 and I had a **RMSE** arround **0,5**. This is why we also apply **cross-validation**.
So, first I enter all the parameters of each model in Python dictionnary.
I decided to put **cross-validation fold** to **5**, and the **scoring** will be **neg_mean_squared_error.** I also put **n_jobs=-1** to use all the processors to have quicker results.

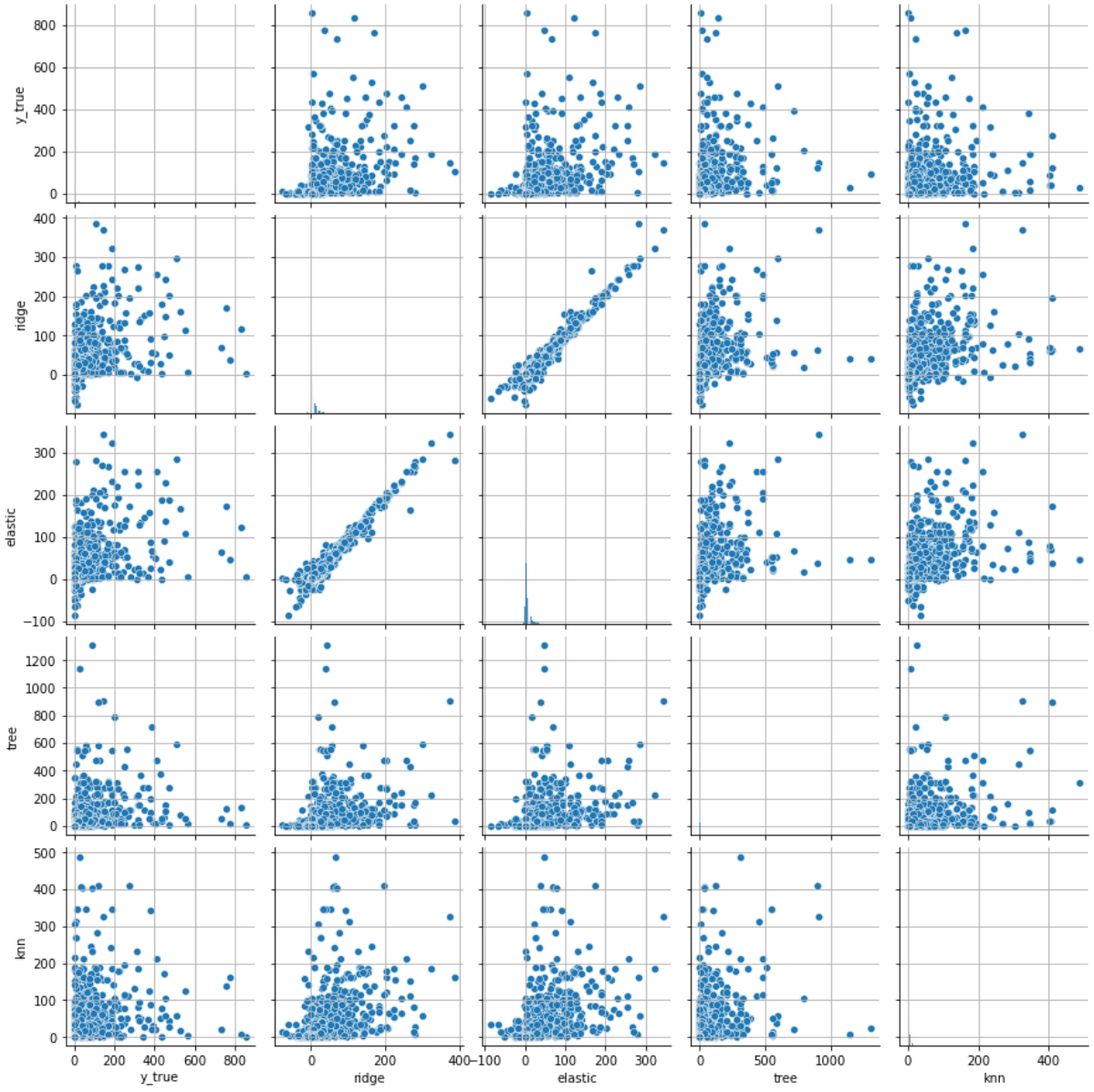
Then, launch the GridSearch and wait for the results !
The results shows that the best models are **Ridge** and **Elastic** as before, the score are slightly less good (**MSE** arround 980 instead of 730) but at least we are not overfitting. Just after them we have the **KNNRegressor** at 1050 of **MSE**

Then, **DecisionTree** and **SVR** are arround 1200-1300 of **MSE.**

So for now, the best model is **Ridge({'alpha': 0.0, 'max_iter': 1000.0, 'normalize': True, 'solver': 'svd', 'tol': 1}).**

7) Graph of the prediction

We can make a paiprplot of our predictions.



We can see that ridge and elastic net give almsot the same results.
But the prediction are not that good, we do not obtain a line in the dedicated graphs.

8) TensorFlow

We can also try TensorFlow to predict our dataset.

Since we are predicting a single continuous value, the output layer will only have 1 node.
Our model consists of four dense layers with 100, 50, 25, and 1 node, respectively.
For regression problems, one of the most commonly used loss functions is mean_squared_error obviously so this what we are going to use here.

After training the model on 40 epochs, we get a **MSE** of **1046,** which is almost like **Ridge and Elastic,** but they are still better.

But, if we look closer, TensorFlow return almost every time the same value, corresponding to the mean of the output. Probably because my manipulation are false.
We will obviously not use this model at all, let's forget this mistake...

9) Deleting features in order to do some improvement

I created a new dataset without all the **X features**. I will try all the previous model on it to see if it works better like this.

The result is not very convicing, I obtain almost the same but with slightly more **MSE.** But for the **API** I will keep the **KNN** model on the new dataset because it is going to be very boring to manually enter 50 features and the difference between the two **MSE** isn't a big deal !

10) Conclusion

Non of the model that I have tested can be considered as good, maybe except for the **KNNRegressor** one.
Indeed, we still get a high **RMSE**, which is probably caused by the too few samples where the output is different from 0.

I have tried to **normalize** the dataset and training the model on it but the **result** were the **same**. I didn't put in the .ipynb script because it would have become too long for nothing really interesting.

The **KNNRegressor** model will tend to surestimate the number of comments under a post.

11) Flask API

First, we import **pickle** and **dump** our model.

For the **flask API,** we need to create 2 python files : **app.py** and **request.py**.

In the **app.py** file :

- Create the Flask application : **app = Flask(__name__)**
- Load our model with pickle : **model = pickle.load(open('model.pkl', 'rb'))**
- Define the home page of our app with the **home()** function.
- Define the predict page with the **predict()** function.
- In this function, we first **collect our data** that the user will manually enter, we **convert** it into float, then put it in a np.array, and call the **predict()** function on this specific array.
- Render the result by **printing** the final **result**.

In the **request.py** file:

- define the url of our Flask app
- can manually enter all the data

The simplest solution is to create a page.html where we can easily enter the data and have the result.

To do so, I created a file name **templates** where Flask will automatically search for templates. In it, I have created the **index.html** file that will be our final render.
All of the style references at the beginning of the file were taken on the internet at the following adress : https://github.com/krishnaik06/Deployment-flask

For the rest of the file, it is just to **require** all the **inputs** from the user.
Then, I created a "Predict" **button** that will **call** the **API** and return the associated prediction.

To correctly run everything, we need to run the **app.py** application in background. I'm using **Anaconda**, so from the **Anaconda Prompt** I'm running the app.py, click on the URL and everything work !