Page Blocks Classification

A project done by Alix JACQUEMIN and César LEBLANC

Python for Data Analysis

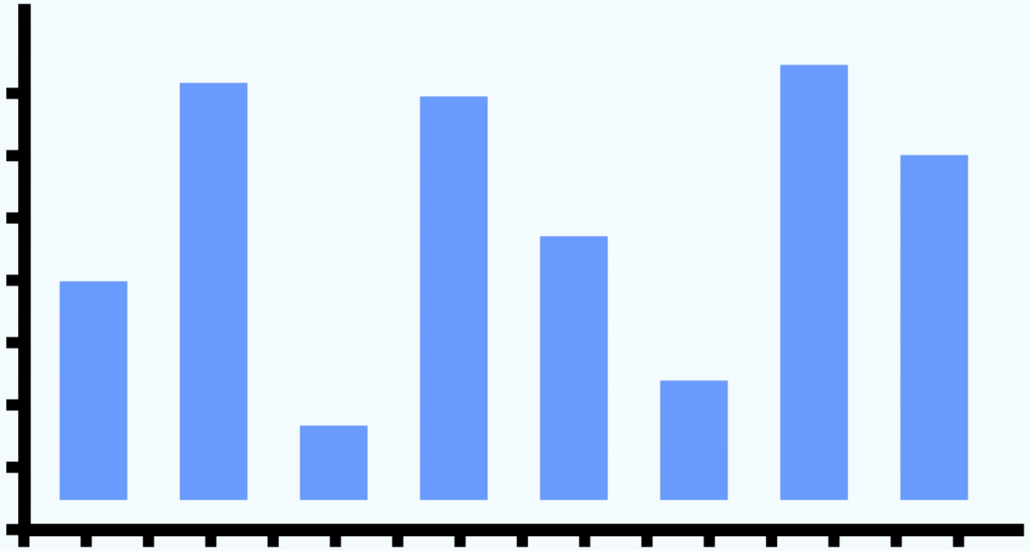
Luc BERTIN

ESILV

January 10th, 2021

Problem

The problem consists in classifying all the blocks of the page layout of a document that has been detected by a segmentation process. This is an essential step in document analysis   in order to separate text from graphic areas. The data set contains five classes.



The five classes

1 – Text

2 – Horizontal line

3 – Picture

4 – Vertical line

5 – Graphic

Class Distribution

Class	Frequency	%	Valid %	Cum %
Text	4913	89.8	89.8	89.8
Horiz. line	329	6.0	6.0	95.8
Graphic	28	0.5	0.5	96.3
Vert. line	88	1.6	1.6	97.9
Picture	115	2.1	2.1	100.0
	TOTAL	5473	100.0	100.0

Attributes of the Dataset

The Dataset is composed of 11 columns, the last one being the class that we have to predict, and of 5473 lines, examples coming from 54 distinct documents.

Each observation concerns one block of page.

All attributes are numeric.

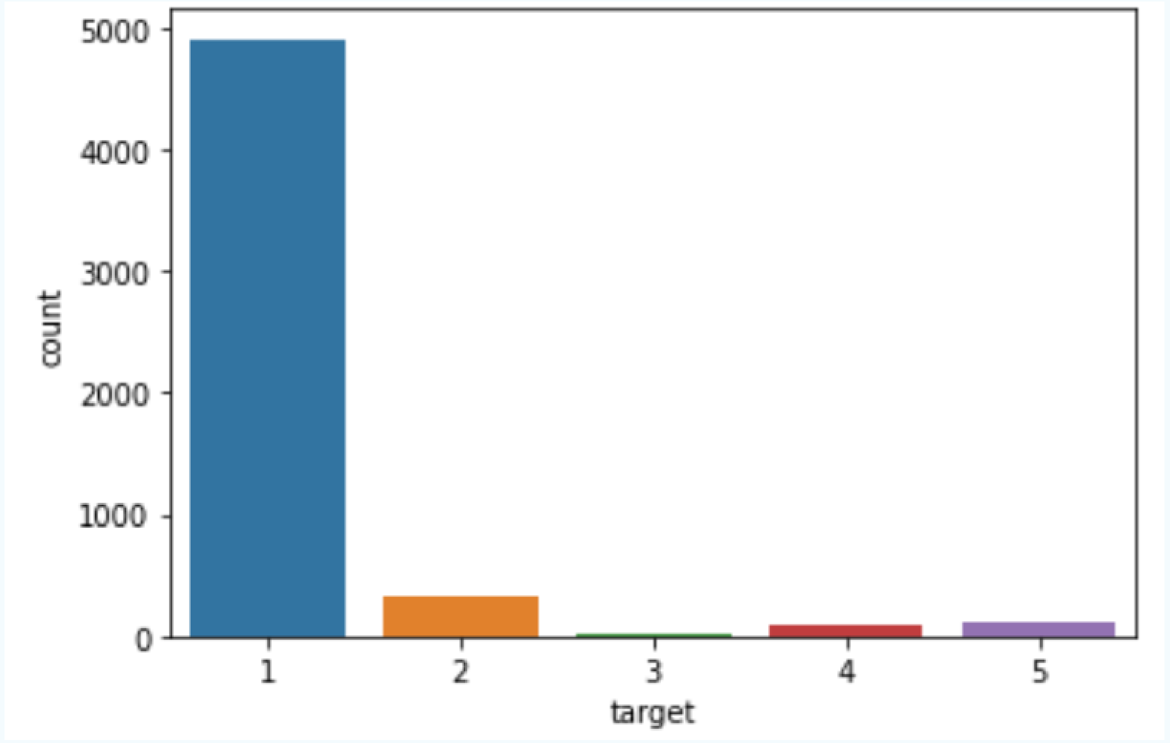
Data are in a format readable by C4.5.

Name of the Column	Type	
Height	Integer	Height of the block.
Lenght	Integer	Length of the block.
Area	Integer	Area of the block (height * lenght).
Eccen	Continuous	Eccentricity of the block (lenght / height).
P_black	Continuous	Percentage of black pixels within the block (blackpix / area).
P_and	Continuous	Percentage of black pixels after the application of the Run Length Smoothing Algorithm (RLSA) (blackand / area)
Mean_tr	Continuous	Mean number of white-black transitions (blackpix / wb_trans)
Blackpix	Integer	Total number of black pixels in the original bitmap of the block.
Blackand	Integer	Total number of black pixels in the bitmap of the block after the RLSA.
Wb_trans	Integer	Number of white-black transitions in the original bitmap of the block.

Health Checks

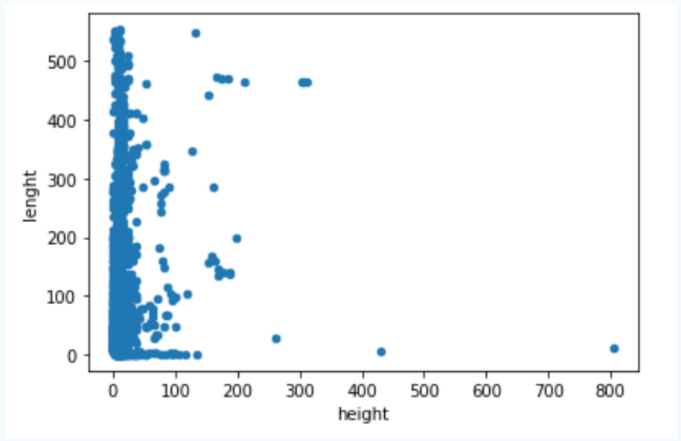
First, we checked if the Dataset had any Missing Values. There are no null values and no NA values.

We also renamed the 'class' column to 'target' since 'class' is already used in Python.



We can also see that the target 1 (which represents text) is over-represented in our Dataset, representing around 90% of the observations

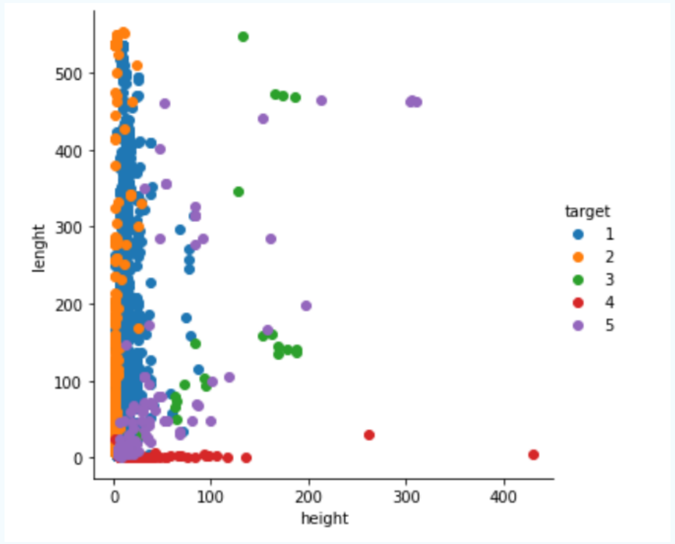
Data Visualization 1/3



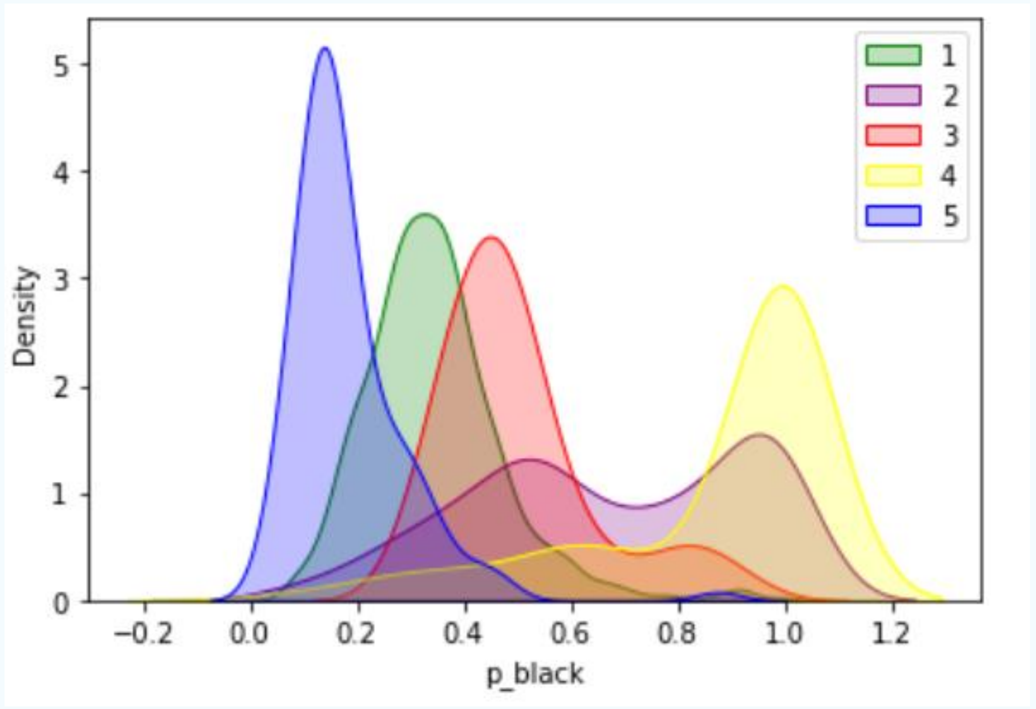
By Plotting the height versus the length, we can see that we have one outliner in the 'height' features (of value 804), that doesn't go along the other observations. Thus we decided to remove it

Then we color our plot according to the target. We can see that the class 4 has a low length, whereas classes 1 and 2 have low heights.

We can even see a small linearity between height and lenght for class 3 and 5.



Data Visualization 2/3

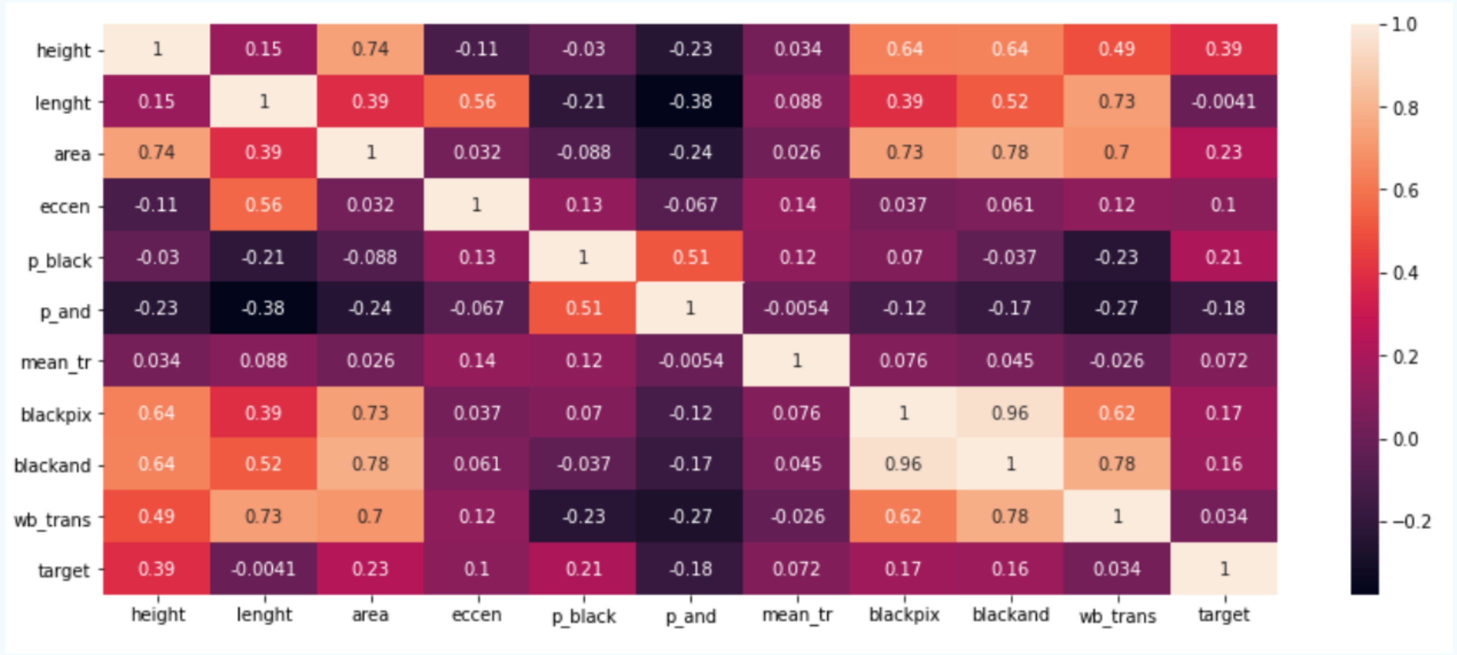


This plot is useful for looking at univariate relations between the classes and p_black (the percentage of black pixels within the block).

The plot creates a kernel density estimate for p_black.

We can see that, for example, the class 5 (which represents a graphic) has in general the lowest p_black among all classes (and on the contrary 4 has the highest amount of p_black).

Data Visualization 3/3



	height	lenght	area	eccen	p_black	p_and	mean_tr	blackpix	blackand	wb_trans	target
height	1	0.15	0.74	-0.11	-0.03	-0.23	0.034	0.64	0.64	0.49	0.39
lenght	0.15	1	0.39	0.56	-0.21	-0.38	0.088	0.39	0.52	0.73	-0.0041
area	0.74	0.39	1	0.032	-0.088	-0.24	0.026	0.73	0.78	0.7	0.23
eccen	-0.11	0.56	0.032	1	0.13	-0.067	0.14	0.037	0.061	0.12	0.1
p_black	-0.03	-0.21	-0.088	0.13	1	0.51	0.12	0.07	-0.037	-0.23	0.21
p_and	-0.23	-0.38	-0.24	-0.067	0.51	1	-0.0054	-0.12	-0.17	-0.27	-0.18
mean_tr	0.034	0.088	0.026	0.14	0.12	-0.0054	1	0.076	0.045	-0.026	0.072
blackpix	0.64	0.39	0.73	0.037	0.07	-0.12	0.076	1	0.96	0.62	0.17
blackand	0.64	0.52	0.78	0.061	-0.037	-0.17	0.045	0.96	1	0.78	0.16
wb_trans	0.49	0.73	0.7	0.12	-0.23	-0.27	-0.026	0.62	0.78	1	0.034
target	0.39	-0.0041	0.23	0.1	0.21	-0.18	0.072	0.17	0.16	0.034	1

We can see on this heatmap that the two most correlated features are blackpix and blackand (the total number of black pixels in the original bitmap of the block and the total number of black pixels in the bitmap of the block after the RLSA) with a positive correlation of 0.96 or 96%.

We can also see that the two least correlated features are the target and the lenght (the type of the block and its lenght).

Modelling step 1/5: Raw Data

At first, we used 7 different models on our raw data without any preprocessing.

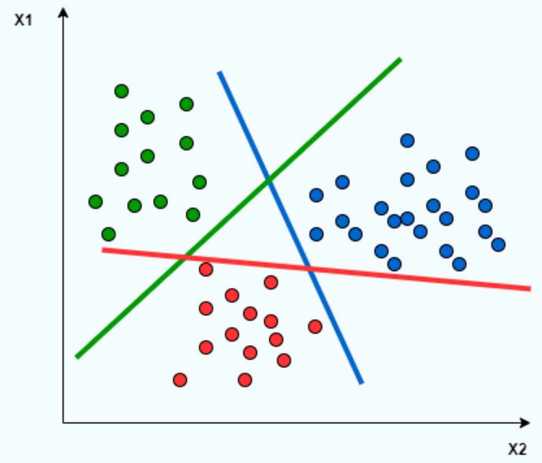
We create a train set and a test set. We put 80% of our observations into training and the rest for testing.

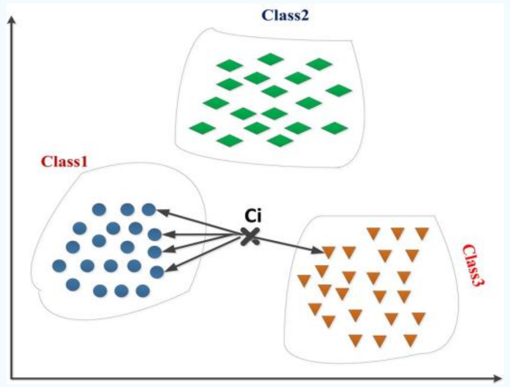
The models that we tried are the following ones :

Multiclass Logistic Regression
y=2
y=1
y=0

Logistic Regression

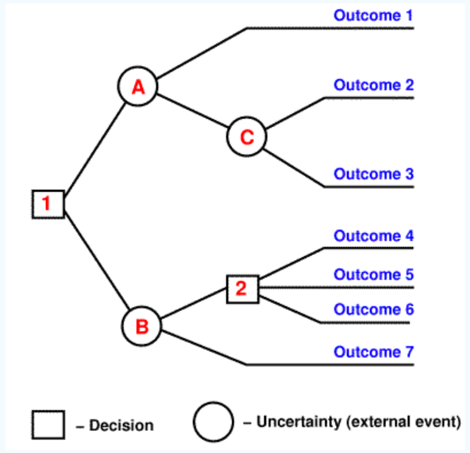
Support Vector Machine





K-Nearest Neighbours

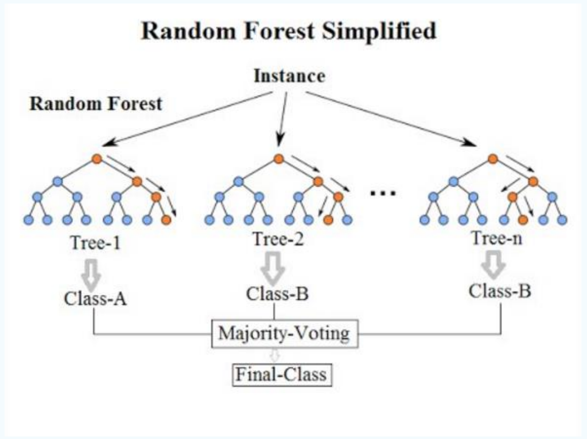
Decision Tree



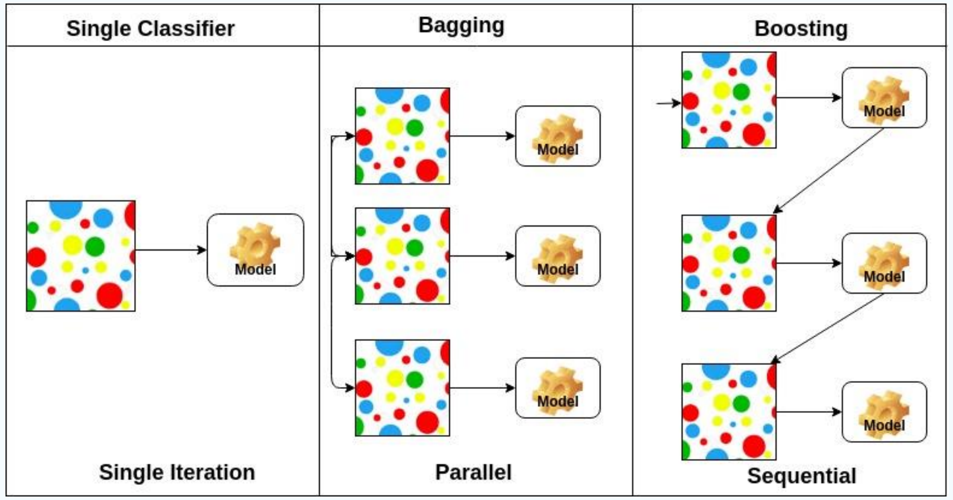
Modelling step 1/5: Raw Data

ESILV LÉONARD DE VINCI — ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

Random Forest



Bagging & ADA Boosting

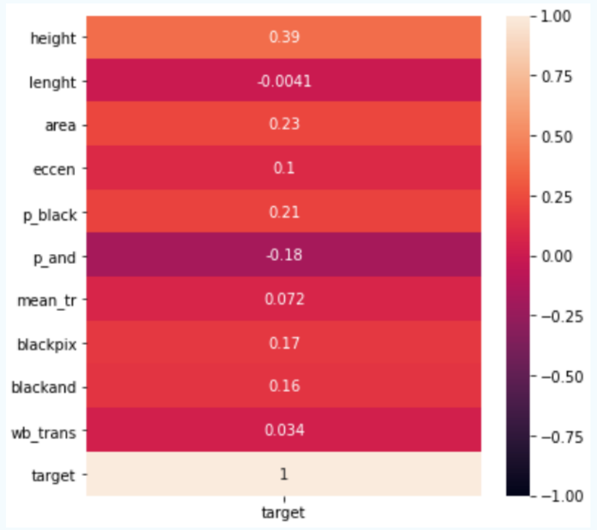


Model	Logistic Regression	SVM	KNN	Decision Tree	Random Forest	Bagging	ADA Boosting
Accuracy	0.961	0.909	0.964	0.974	0.97	0.921	0.973

In this step, our best accuracy was obtained with Random Forest and Ada Boosting (between 0.96 and 0.97).

Modelling step 2/5 : with Preprocessing

For this part we used the same techniques of modelling while removing the least correlated features to the target. We only used the 3 methods which had the best accuracies in the previous part: Decision Tree, Random Forest and ADA Boosting.



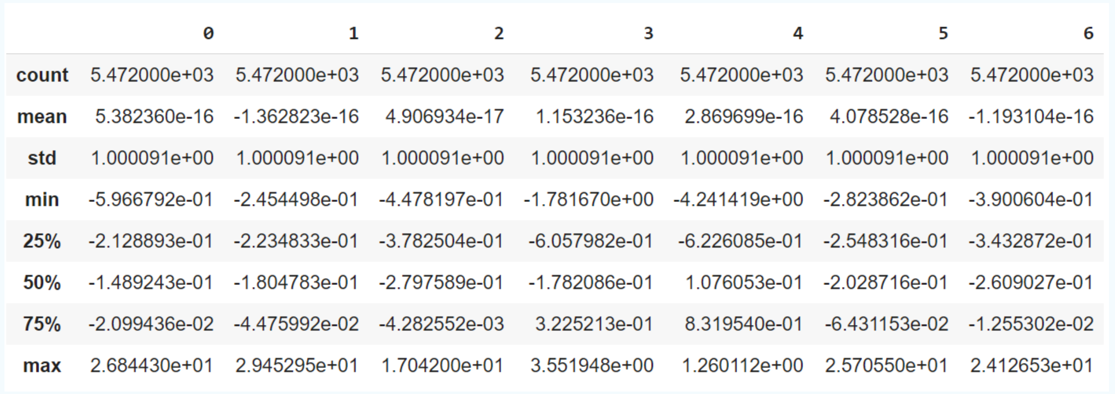
Model	Accuracy
Decision Tree	0.97
Random Forest	0.972
ADA Boosting	0.967

We can see that the least correlated features are : lenght, mean_tr and wb_trans. Thus we remove them.

For this step, we obtain our best accuracy with (Random Forest) and we improve our overall acuracy.
We conclude that it was a good idea to remove the least correlated variables to the target.

Modelling step 3: scaling

We scale our Data, still using the same columns and models as in the previous step.



	0	1	2	3	4	5	6
count	5.472000e+03	5.472000e+03	5.472000e+03	5.472000e+03	5.472000e+03	5.472000e+03	5.472000e+03
mean	5.382360e-16	-1.362823e-16	4.906934e-17	1.153236e-16	2.869699e-16	4.078528e-16	-1.193104e-16
std	1.000091e+00	1.000091e+00	1.000091e+00	1.000091e+00	1.000091e+00	1.000091e+00	1.000091e+00
min	-5.966792e-01	-2.454498e-01	-4.478197e-01	-1.781670e+00	-4.241419e+00	-2.823862e-01	-3.900604e-01
25%	-2.128893e-01	-2.234833e-01	-3.782504e-01	-6.057982e-01	-6.226085e-01	-2.548316e-01	-3.432872e-01
50%	-1.489243e-01	-1.804783e-01	-2.797589e-01	-1.782086e-01	1.076053e-01	-2.028716e-01	-2.609027e-01
75%	-2.099436e-02	-4.475992e-02	-4.282552e-03	3.225213e-01	8.319540e-01	-6.431153e-02	-1.255302e-02
max	2.684430e+01	2.945295e+01	1.704200e+01	3.551948e+00	1.260112e+00	2.570550e+01	2.412653e+01

Now we can see that the mean of every features is now equal to 0 and their standard deviations are equal to 1 : our variables are standardized.
We also removed the same features than before.

For this step we achieve our best accuracy with Random Forest.
However, our overall accuracy decreased so it was not worth it to scale the features.

Model	Accuracy
Decision Tree	0.968
Random Forest	0.973
ADA Boosting	0.974

Modelling step 4 : Hypertuning

In this step, we tuned our hyperparameters for the three models that worked best for us so far : Decision Tree, Random Forest and ADA Boosting.
We removed the least correlated variables to the target as in step 2.
We also scaled the variables since it increased our accuracy in step 3.

In order to define our hyperparameters, we used Grid Search, using corresponding parameters for each model, for example 'max_depth' and 'min_samples_split' are the parameters that we chose for the Decision Tree.

We used the parameters found with Grid Search in our previous models and ended up with
the following results.

Model	Accuracy
Decision Tree	0.969
Random Forest	0.973
ADA Boosting	0.942

Modelling step 5 : Cross Validation

For this last step, we used cross validation by splitting our dataset, fitting a model and then computing the accuracy 10 different times with different splits each time.
We also removed the least correlated variables, scaled our features and tuned the hyperparameters as seen in our previous steps

The results obtained with this step are the following:

Model	Accuracy
Decision Tree	0.976
Random Forest	0.984
ADA Boosting	0.951

Conclusion

Among all the steps and preprocessing that we have done, we obtained our best accuracy with a Random Forest model (0.984).
However, by running this notebook multiple times, the accuracies might change a little (for example we obtained respectively 0.991 and 0.992 once while using a Decision Tree and a Random Forest model), but that's because there is a part of random in the algorithms.

Final Results after step 5 of Modelling:

Model	Accuracy
Decision Tree	0.976
Random Forest	0.984
ADA Boosting	0.951