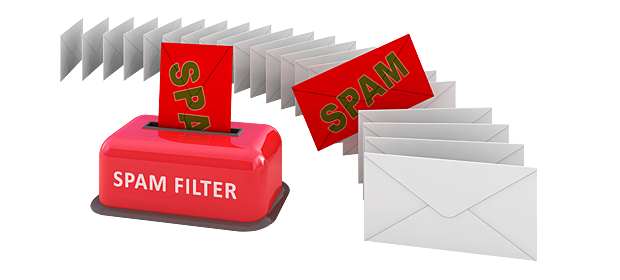
Documentation Technique



**Projet : Spam Classifier**

Steve et Jérémy

**2022**

**Informations**

|  |  |
| --- | --- |
| **Nom du projet** | Spam Classifier |
| **Type de document** | Documentation technique |
| **Date** | 20/01/2020 |
| **Version** | 1.0 |
| **Mots-clés** | Spam– Classifieurs – Machine Learning – Algorithme – Application Web |
| **Auteurs** | Steve Lacroix  Jérémy Ruellan |

**Table des matières**

Contenu

[1 – Enoncé du Projet 4](#_Toc94569547)

[2 –Application Web 5](#_Toc94569548)

[2.1 – Streamlit 5](#_Toc94569549)

[2.2 – Les Fichiers Python de l’app 5](#_Toc94569551)

[2.3 – Les Colonnes Features 5](#_Toc94569552)

[2.4 – Les Différents Classifiers Utilisés 7](#_Toc94569553)

[2.5 – Recherche des Hyperparamètres des Classifiers 8](#_Toc94569554)

[2.6 – Preprocessing 10](#_Toc94569555)

[2.7 – Evalutation et Score des Classifiers 11](#_Toc94569556)

[2.8 – Prédiction d’un message simple 14](#_Toc94569557)

# 1 – Enoncé du Projet

Concevoir un classifieur de détection automatique de SPAM.

La collection SMS Spam est un ensemble de messages SMS marqués qui ont été collectés pour la recherche sur les SMS Spam. Elle contient un ensemble de messages SMS en anglais de 5 574 messages, étiquetés selon qu'ils sont ham (légitimes) ou spam.

Attention : il s'agit d'un jeu de données qui ne doit pas être travaillé en NLP (on fera du NLP un peu plus tard). En terme de préparation des données, il s'agit d'extraire une multitude d'infos à partir du message texte, afin d'obtenir un ensemble de descripteurs (features) comme :

* nombre et proportion des caractères "!, ?, €, #" etc
* longueur du message
* nombre et proportion de majuscules
* présence d'URL
* et d'autres descripteurs

# 2 –Application Web

## 2.1 – Streamlit .

## Nous avons créé notre application sur Steamlit.

## C’est le framework d’application spécifiquement pour les équipes d’apprentissage automatique et de science des données.

## Il est open source en langage python et on peut créer de belles applications Web pour le machine learning en peu de temps. Il est compatible avec les principales bibliothèques python telles que scikit-learn, keras, pytorch, latex, numpy, pandas, matplotlib, etc.

## 2.2 – Les Fichiers Python de l’app

Notre code se résume en 3 fichiers python.

* « ergonomie.py » : qui contient la classe affichage. Elle est utilisée pour l’affichage de l’ensemble des éléments sur l’interface de l’application web.
* « model\_prediction.py » : qui contient la class prediction. Elle est utilisée pour toutes les actions liées au machine learning.
* « app\_spam.py » : qui instancie les 2 classes et appelle leurs méthodes et attributs.

|  |  |
| --- | --- |
| **Ergonomie.py** | **Model\_prediction.py** |

## 2.3 – Les Colonnes Features

Dans un premier temps, nous avons réalisés des recherches sur Jupyter pour identifier les caractéristiques d’un message Spam.

Suite à cela, nous avons utilisé 7 colonnes features : Taille\_Message, Ponctuations, Majuscules, Emoticônes, URL, motSpam, Telephone\_5.

Pour la colonne ‘Taille\_Message’, nous indiquons la longueur du message.

Pour la colonne ‘Ponctuations’, nous indiquons un 1 si il y a la présence de caractères spéciaux dans le message ("!, $, £, ?, #, %, \_, \*, =, &, -").

Pour la colonne ‘Majuscules’, nous indiquons le nombre de majuscules dans le message.

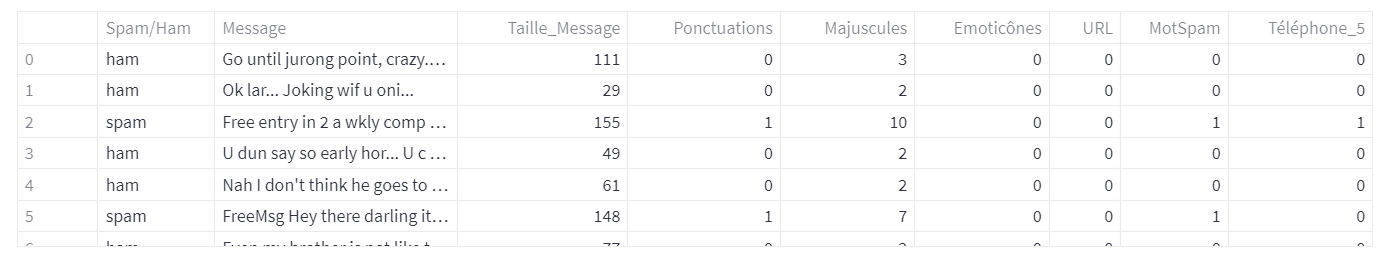
Pour la colonne ‘Emoticônes’, nous indiquons un 1 si il y a la présence d’émoticônes (":), :-), ;), ;-), :(, :-(, :D, :-D, ;D, ;-D, :P, :-P, ;P, ;-P").

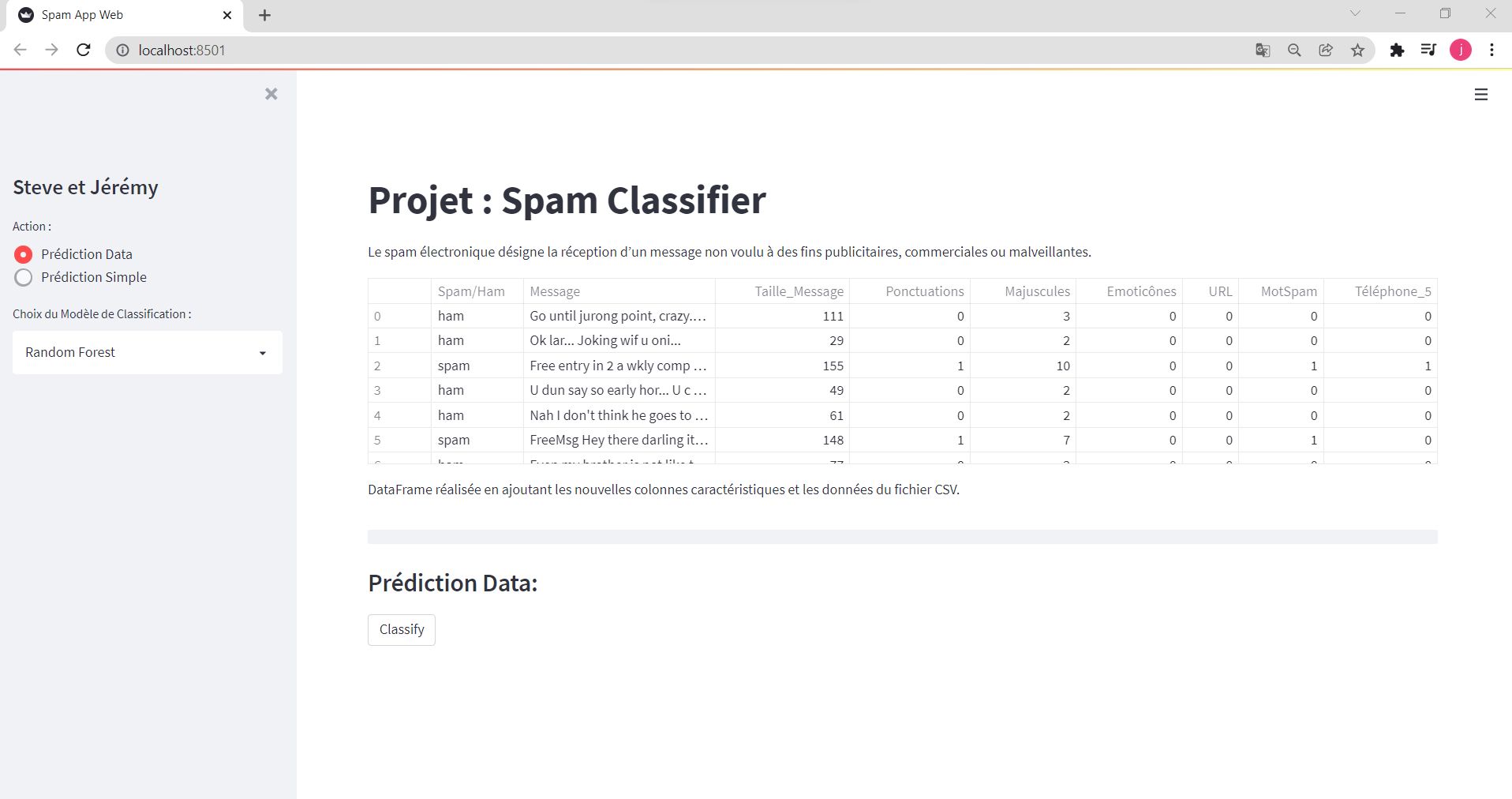
Pour la colonne ‘URL’, nous indiquons un 1 si il y a la présence d’URL dans le message ("www, @, http, https").

Pour la colonne ‘motSpam’, nous indiquons un 1 si il y a la présence de certains mots, souvent présents dans les spam ("free, call, price, win, won, new, now, cash, text, txt, mobile, urgent, 150p").

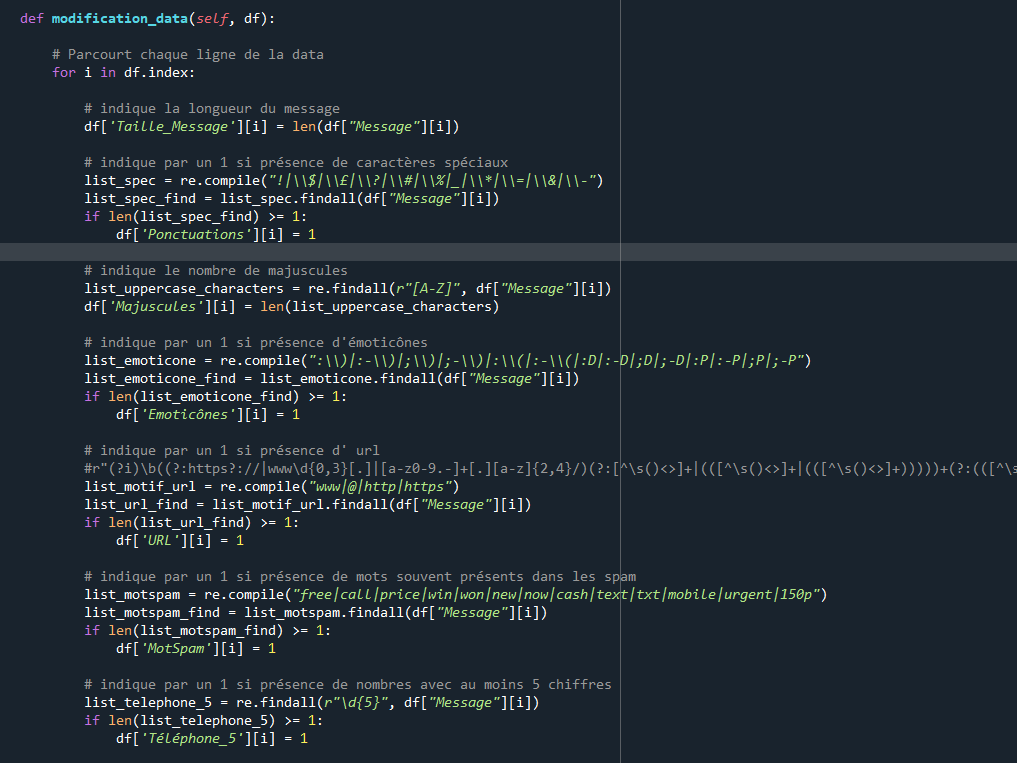
Pour la colonne ‘Telephone\_5’, nous indiquons un 1 si il y a la présence d’un ou plusieurs nombres avec au moins 5 chiffres dans le message.

Sur l’interface de l’application web, nous affichons la Dataframe avec les colonnes features.





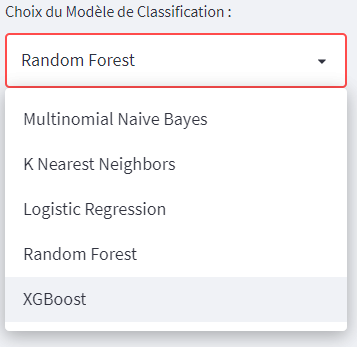
Méthode « modification\_data » dans Model\_prediction.py

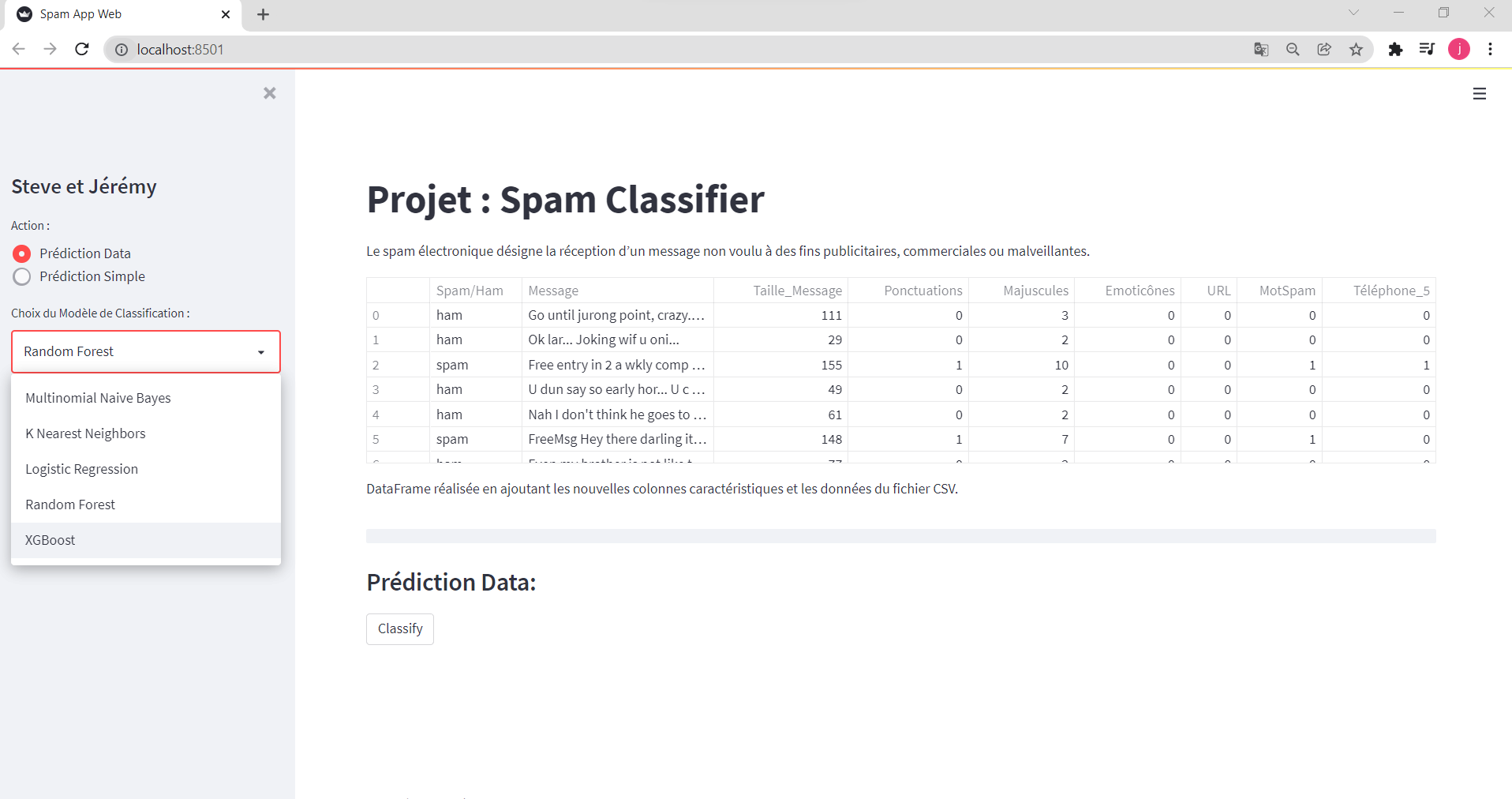


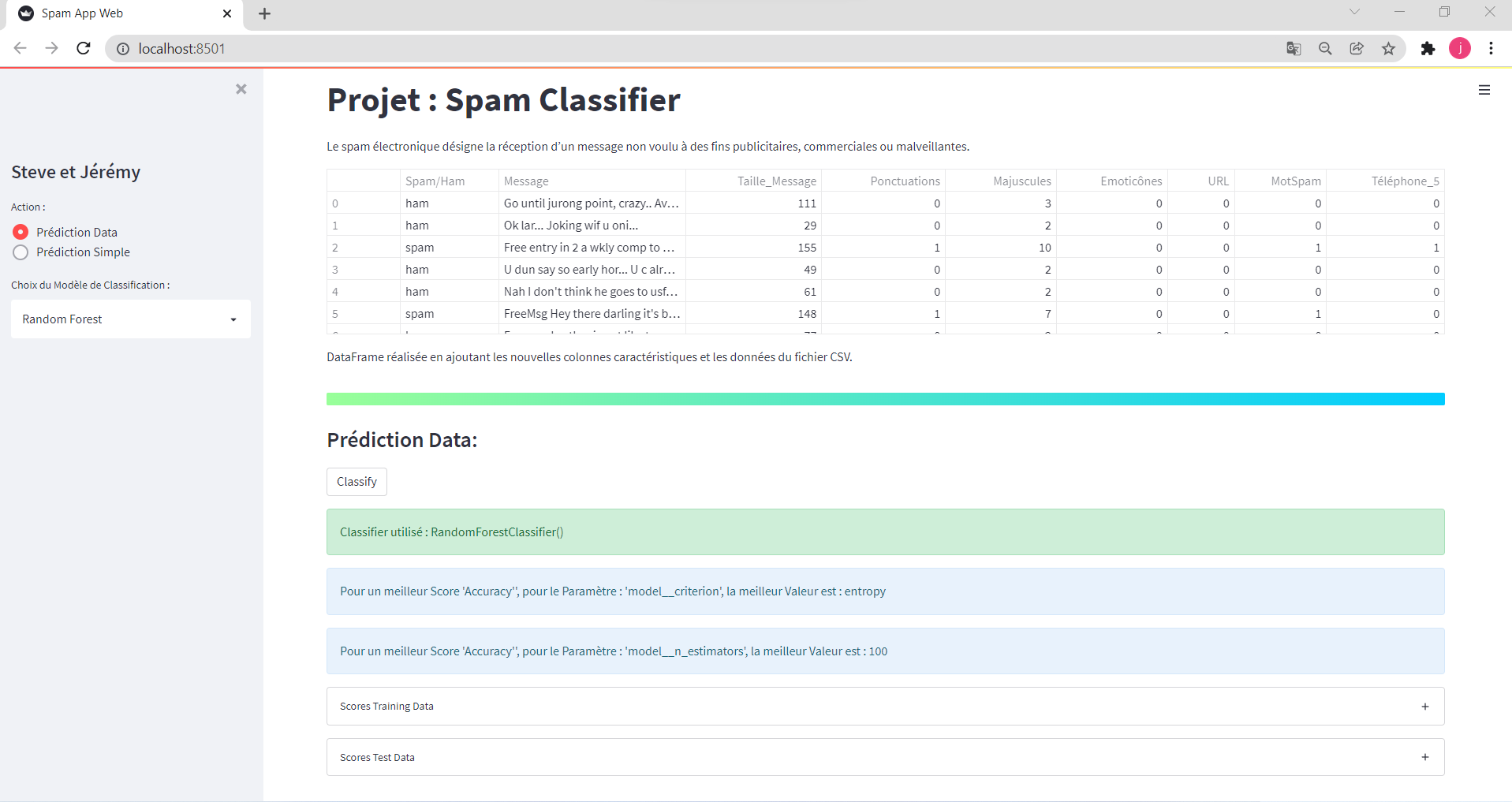
## 2.4 – Les Différents Classifiers Utilisés

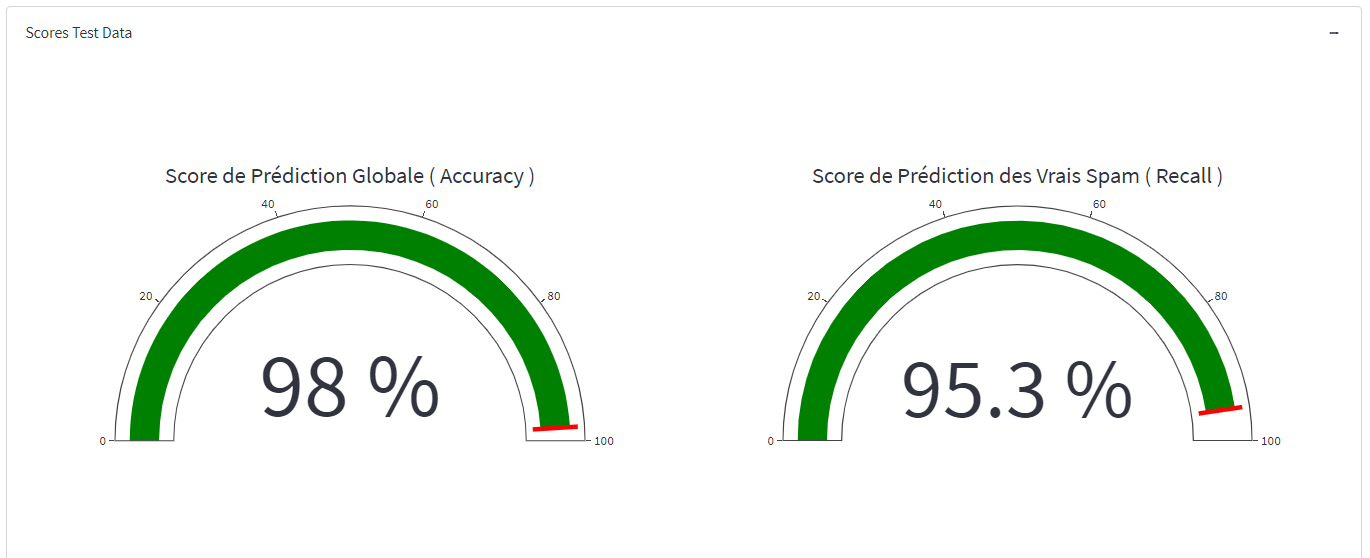
Nous avons testé 5 modèles de classifier : Multinomial Naive Bayes, K Nearest Neighbors, Logistic Regression, Random Forest, XGBoost.

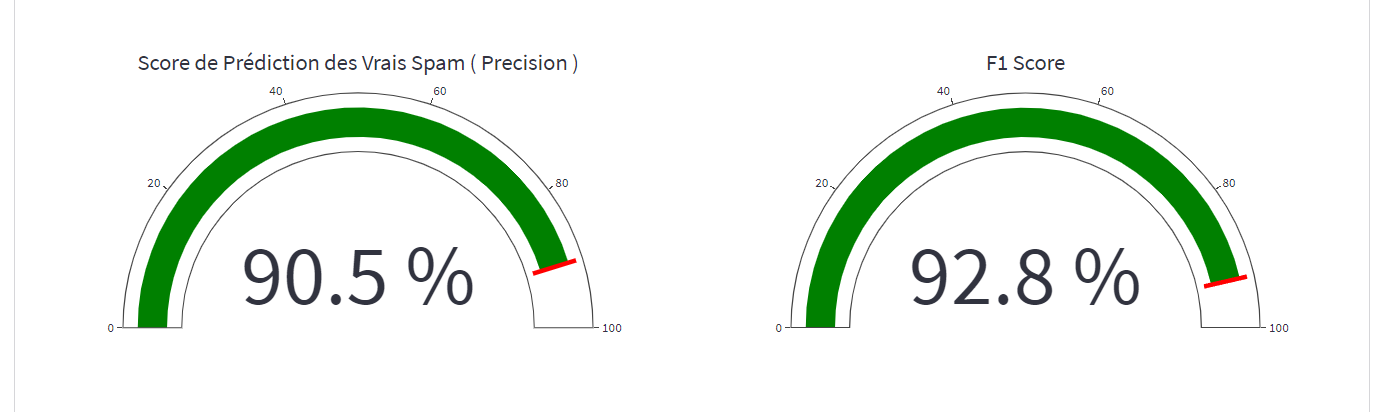
Nous pouvons tous les selectionner sur l’interface de l’application web et ensuite voir celui qui donne les meilleurs scores de performance, après le tuning de leurs hyperparamètres et du preprocessing.

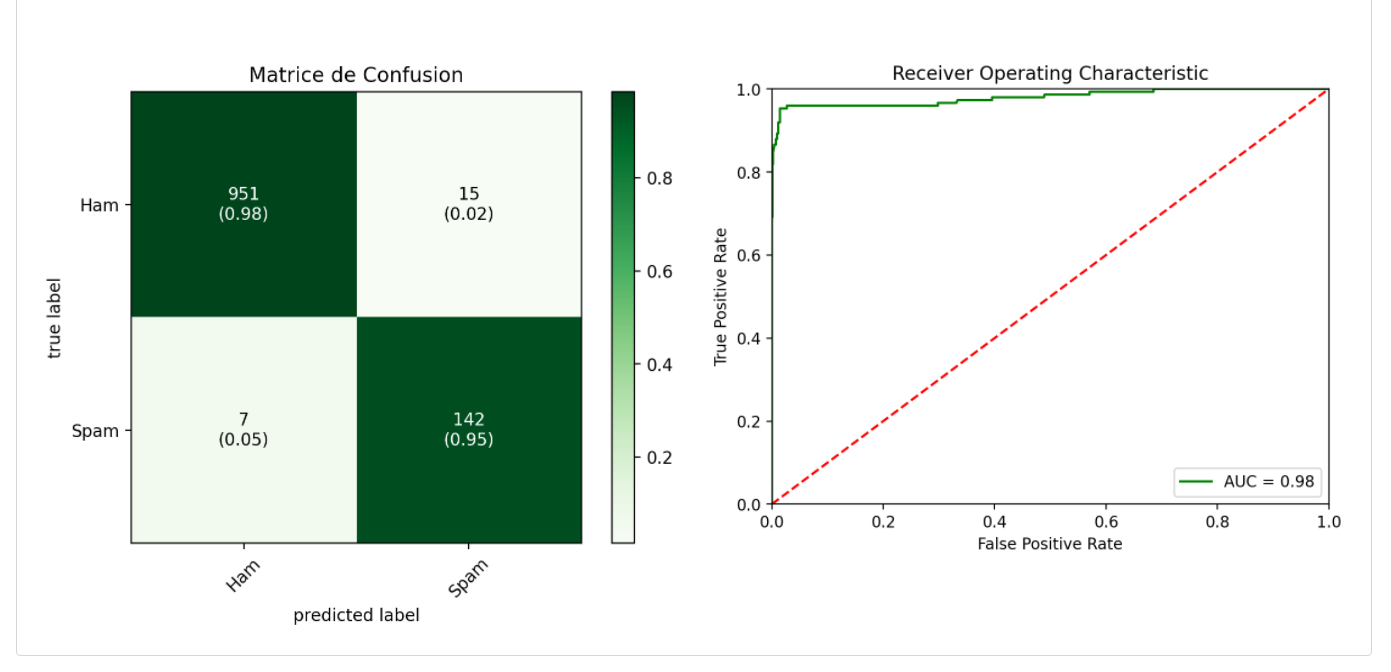
**

**

**

**

**

**

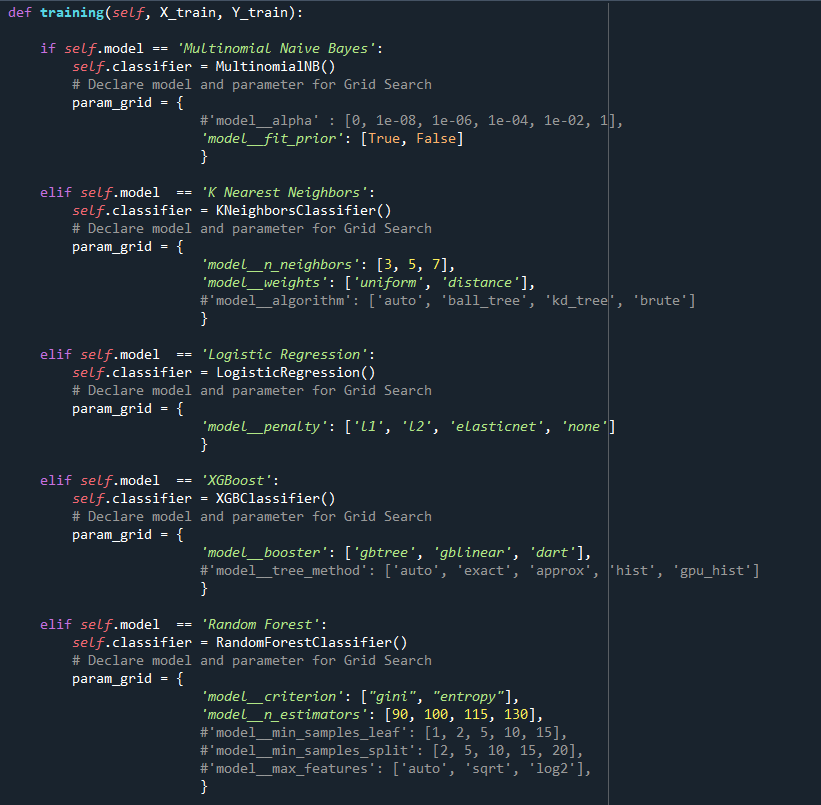
## 2.5 – Recherche des Hyperparamètres des Classifiers

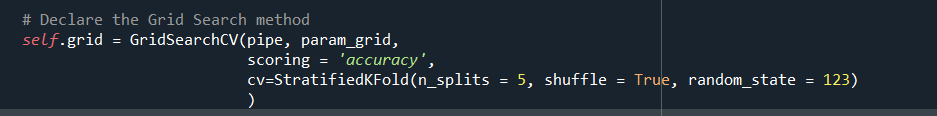
Grâce à la fonction GridSearchCV(), nous recherchons les hyperparamètres.

Nous avons indiqué au paramètre de scoring de GridSearchCV() le score ‘accuracy’.

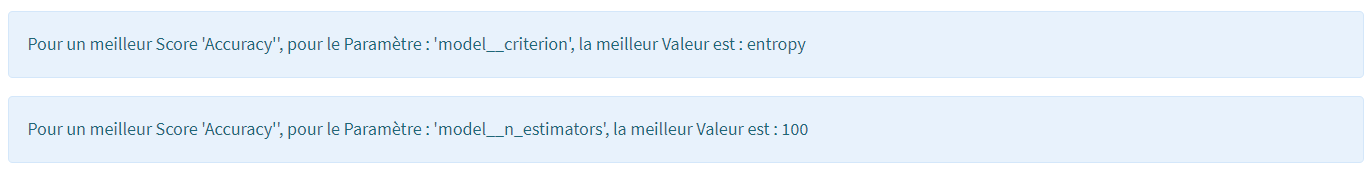
Par contre, nous avons pu remarquer que l’on a une baisse des scores de prédiction lorsque l’on ajoute plus de paramètres avec plusieurs valeurs possibles, malgré le fait de garder possible les valeurs par défaut.

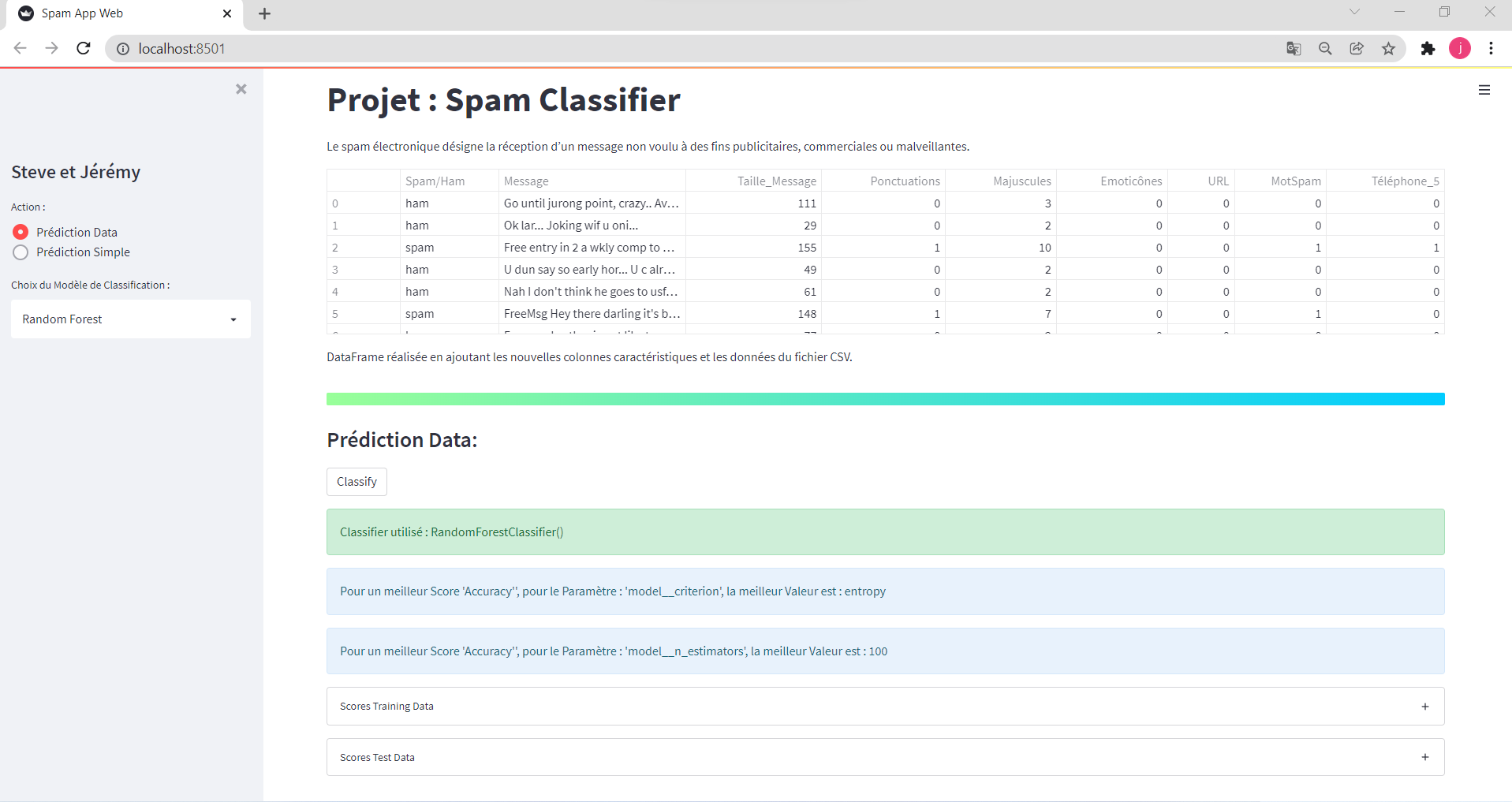
Méthode « training » dans Model\_prediction.py

**

**

Voici ce que nous obtenons par exemple sur l’interface de l’application web suite à la recherche des hyperparamètres pour le classifier Random Forest.

**

**

## 2.6 – Preprocessing

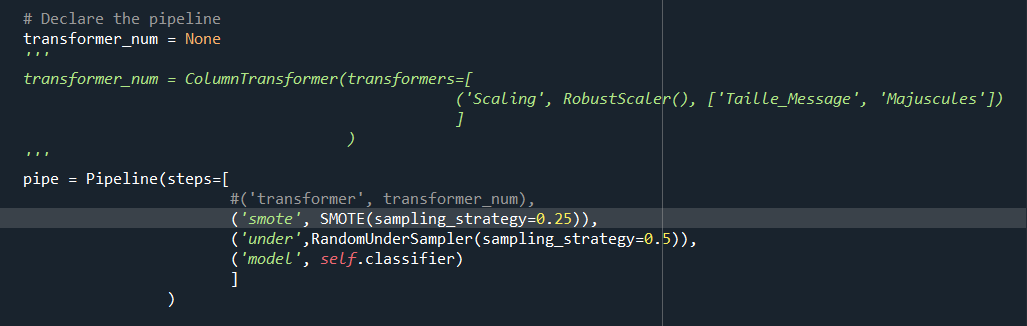
Pour le preprocessing, nous utilisons le Synthetic Minority Oversampling Technique. Il produit des observations minoritaires ressemblantes mais distinctes de celles déjà existantes. Nous obtenons un meilleur score recall, mais le score accuracy diminue car le modèle fait plus d’erreurs concernant les vrais ham qu’il considère spam pour certains. Cependant au final, il prédit mieux les spam.

Ensuite, dans le Pipeline, nous avons rajouté aussi un random undersampling. Il retire aléatoirement x% des observations majoritaires. Ce qui nous a permis de re-augmenter le score ‘accuracy’.

Au final pour résumer en détails, nous suréchantillonnons la classe minoritaire avec SMOTE à un rapport d'environ 2,5:10, puis nous sous-échantillonnons la classe majoritaire pour obtenir un rapport d'environ 1:2.

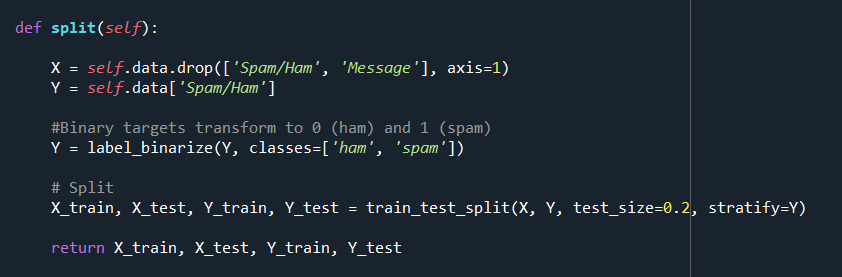
Ensuite par contre, nous avons essayé d’appliquer un RobustScaler() sur les colonnes ‘Majuscule’ et ‘Taille\_Message’ de notre dataframe. Et nous avons observé une baisse des scores de performance pour chaque modèle que l’on utilisait. C’est pour cette raison que nous ne l’avons pas retenu au final.

Méthode « training » dans Model\_prediction.py

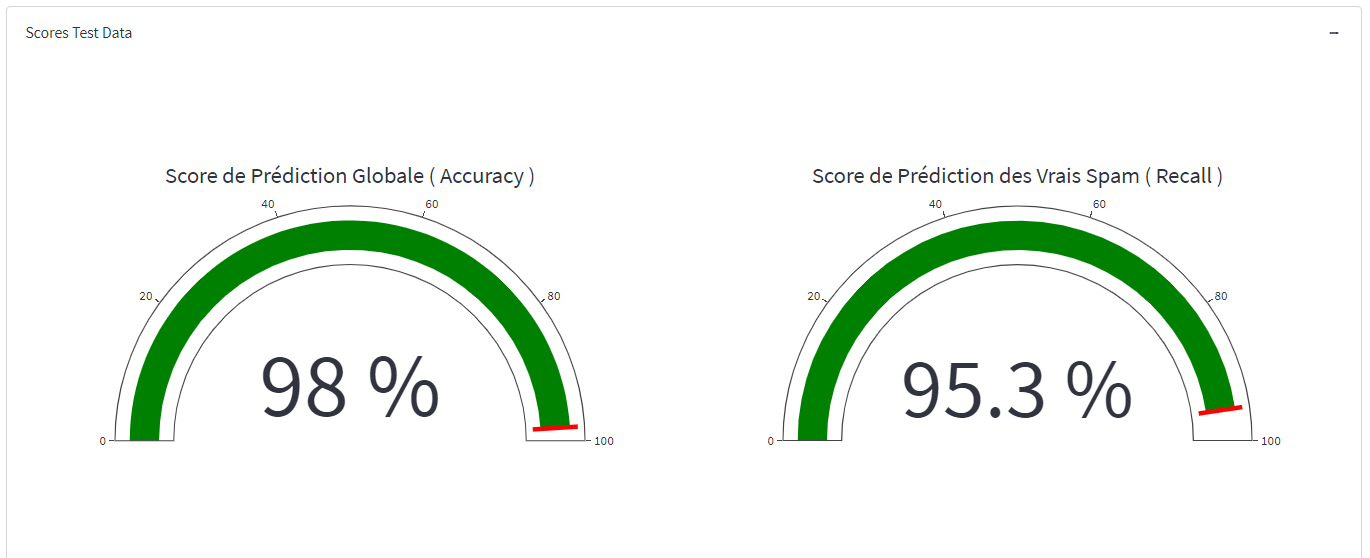


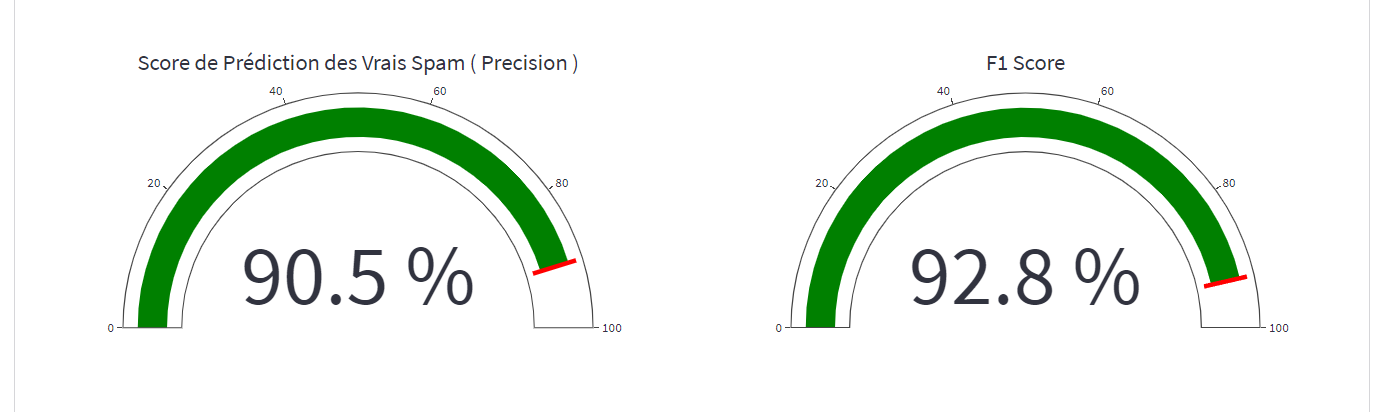
Aussi, on encode la target avant le train\_test\_split() avec le label\_binarize() de la librairie sklearn. Dans la colonne Y, le Spam est tranformé en 1 et le ham en 0.

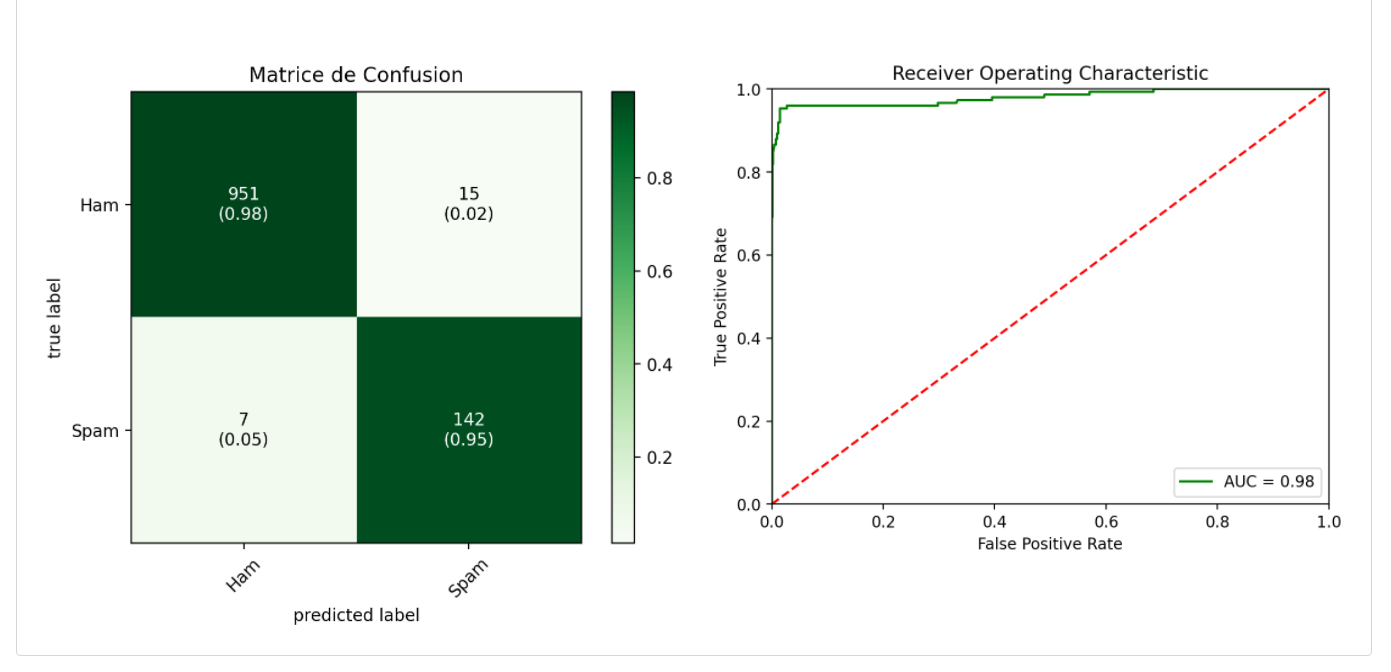
Méthode « split » dans Model\_prediction.py

**

Voici un exemple des résultats que nous affichons sur l’application web suite au préprocessing :

**

**

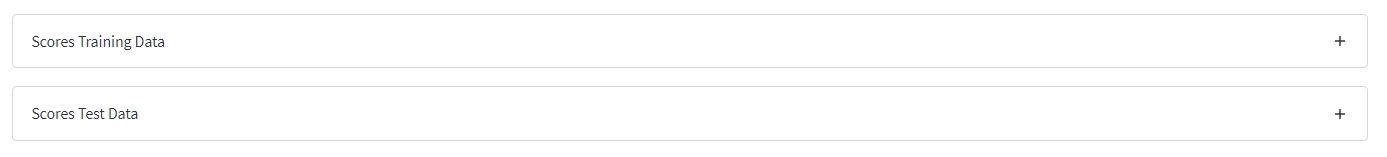
**

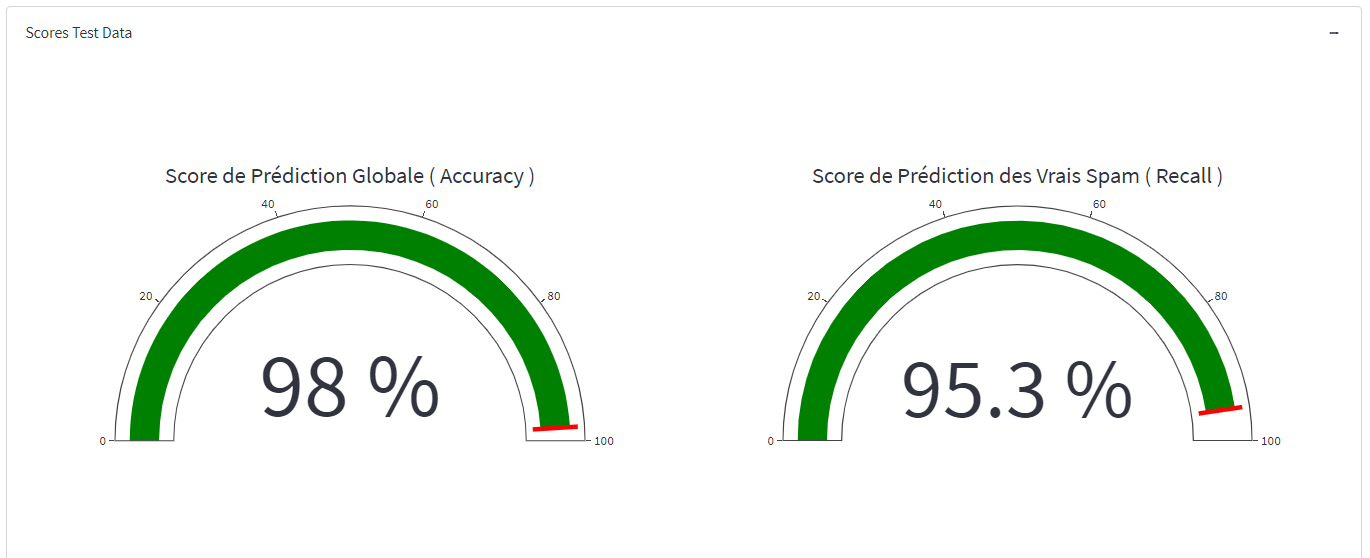
## 2.7 – Evalutation et Score des Classifiers

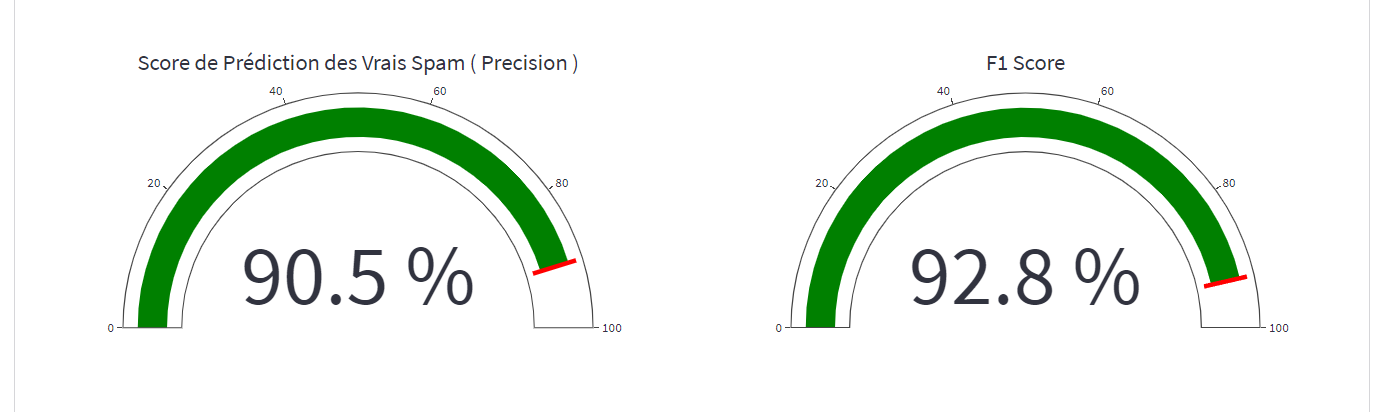
Pour évaluer nos modèles, nous affichons plusieurs scores sur l’application web pour le training data et le test data :

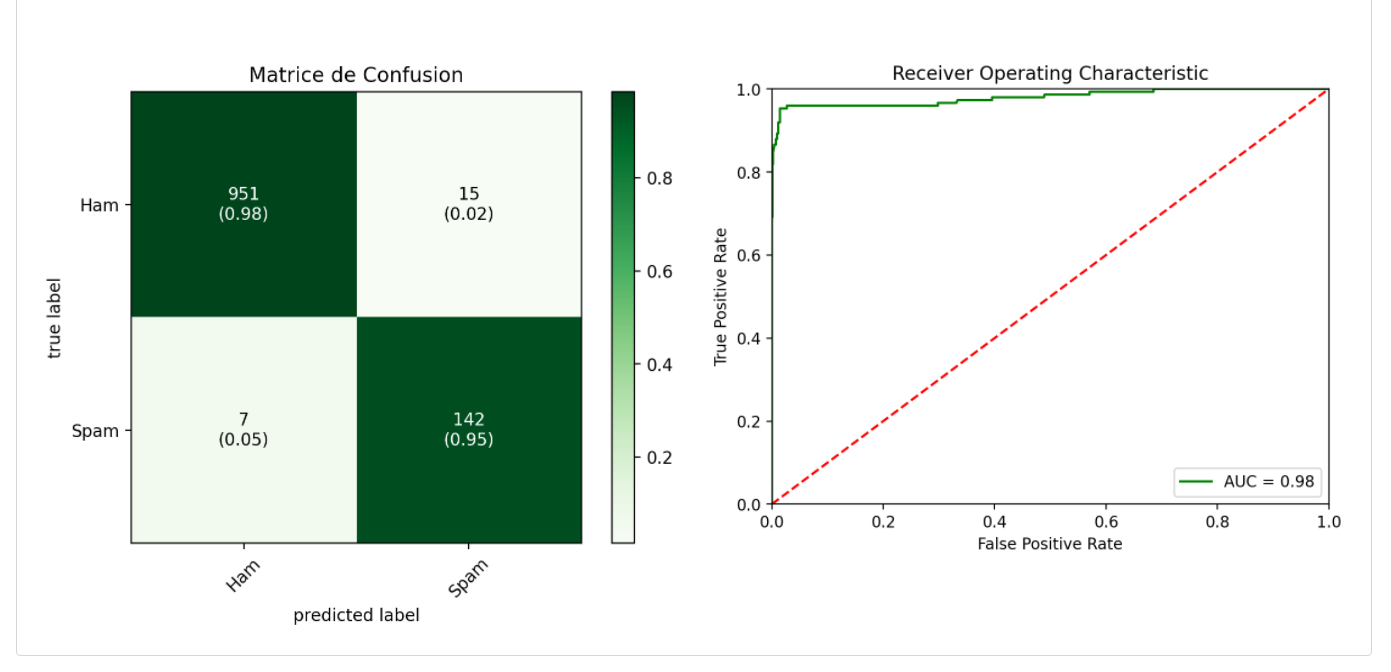
* la Confusion matrix
* le score de prediction (accuracy)
* le score du recall
* Le score de precision
* le F1-Score
* le [Receiver Operating Characteristic](http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html)

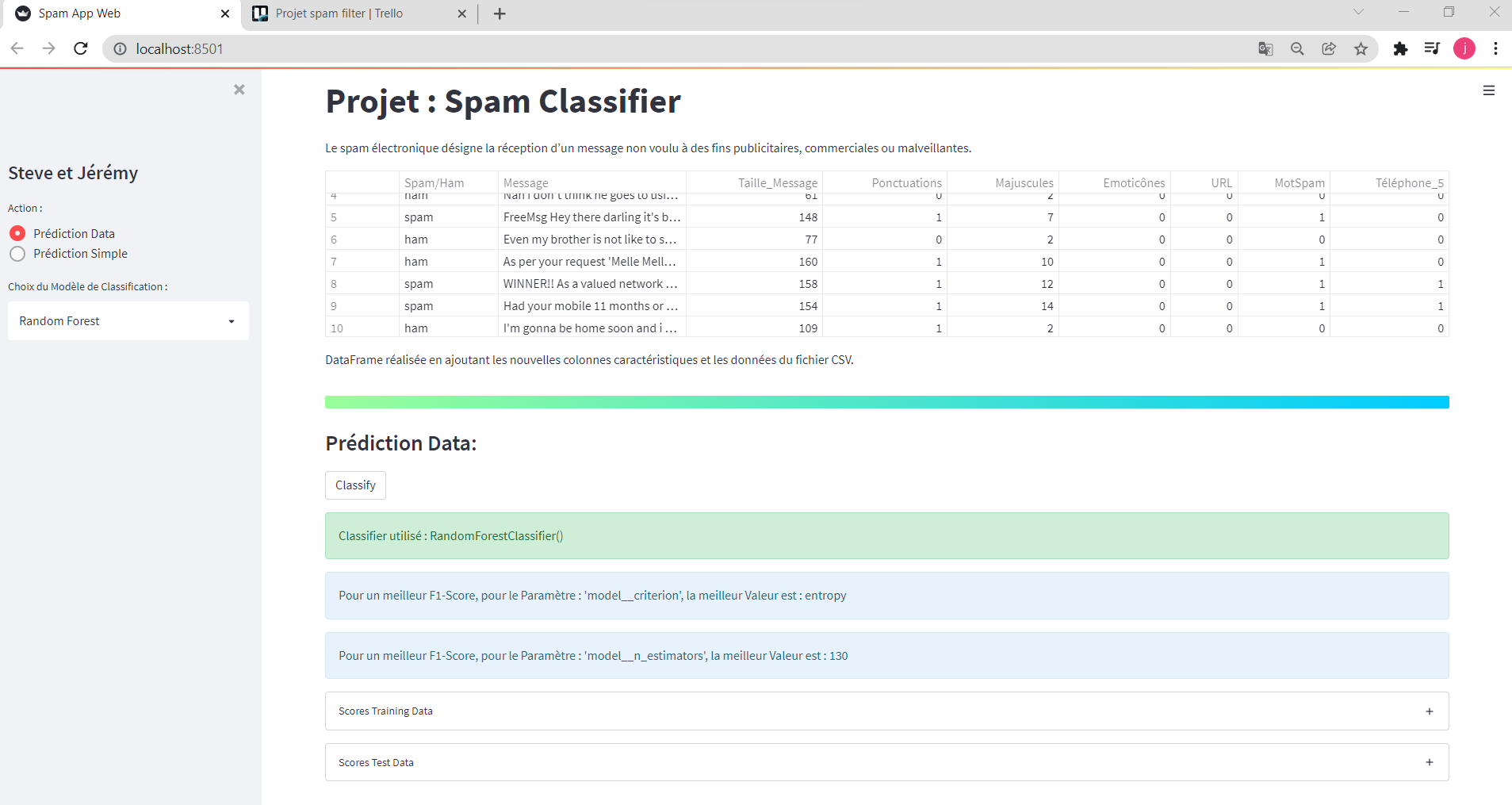
Sur l’application, pour pouvoir les observer, il faut pour cela, après avoir réalisé une ‘Prédiction Data’ suivant un classifier, dérouler les containers ‘Scores Training Data’ et ‘Scores Test Data’ qui sont contenus dans l’expander.





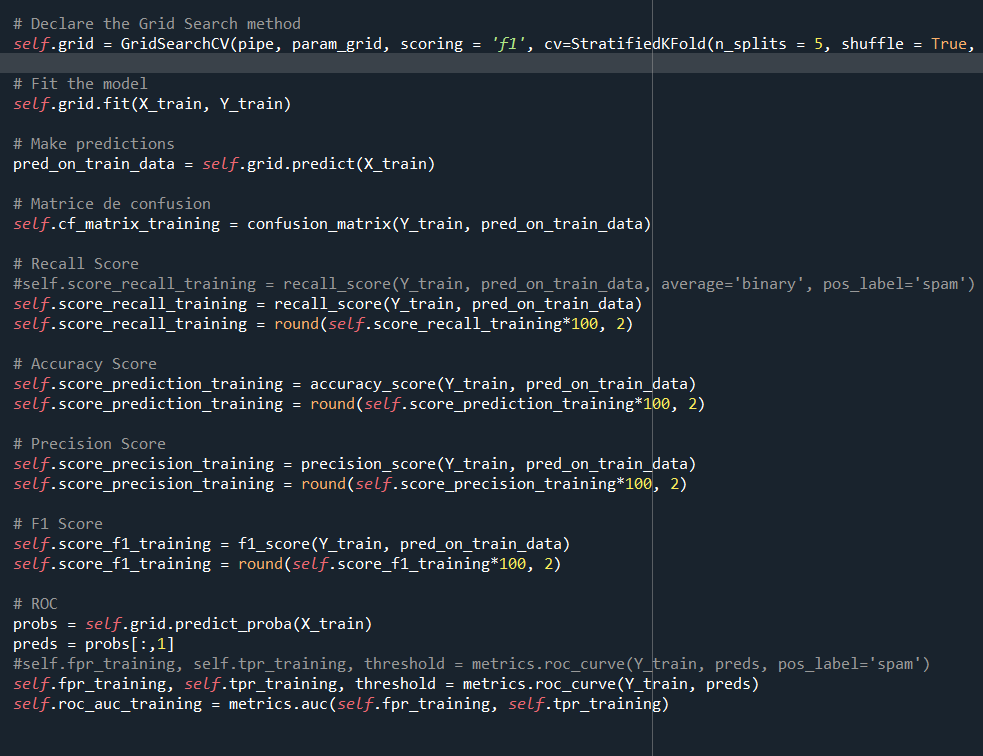






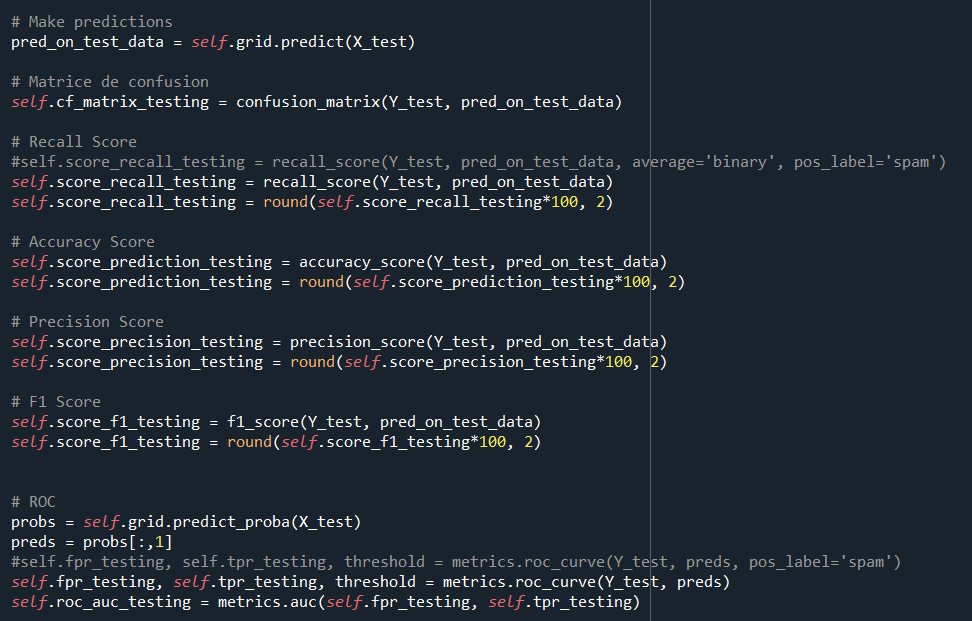
Voici ci-dessous la partie du code qui créé les scores de performance du training data.

Méthode « training » dans Model\_prediction.py



Voici ci-dessous la partie du code qui créé les scores de performance du test data.

Méthode « testing » dans Model\_prediction.py



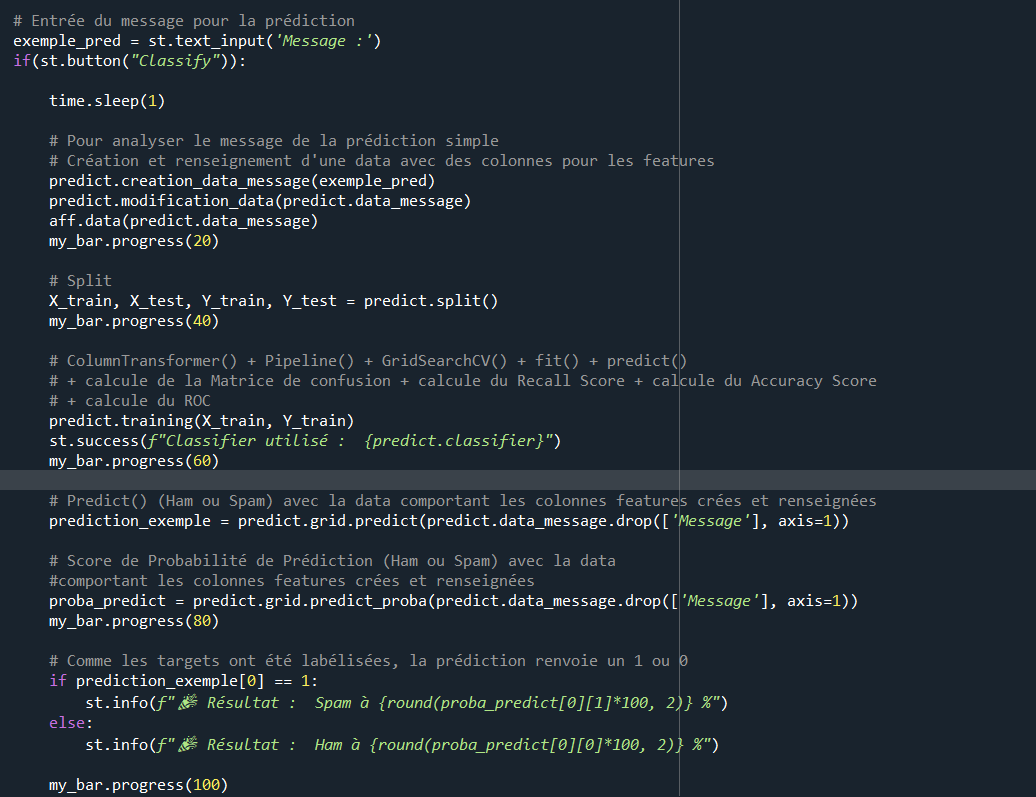
## 2.8 – Prédiction d’un message simple

Dans l’application web, le modèle pré-entrainé peut faire une prédiction sur un message qu’on lui indique.

En plus de sa réponse, à savoir un spam ou un ham, il nous affichera aussi un ‘indice de confiance’ que l’on calcule avec la fonction ‘proba\_predict’.

Sachant que le modèle s’est entrainé sur les colonnes features que l’on a créé (Taille\_Message, Ponctuations, Majuscules, Emoticônes, URL, motSpam, Telephone\_5.), il ne peut faire une prédiction depuis le message renseigné. Alors nous remplissons ces colonnes suivant ce message, puis nous effectuons la prédiction depuis ces colonnes remplies.

Dans app\_spam.py



Voici un exemple sur l’application web pour la partie prédiction simple :

