Python for Data Analysis

Seqqat Othman - Jullien Maxime - Dobosz Adam

Introduction

We decided to work on the Spambase Data Set. We had to **visualize** the data and create a **model** which predicts if a given email is a spam or not, based on several attributes.

Attributes description

- 48 float attributes which represent the frequency of a **specific word**
- 6 float attributes of type which represent the frequency of a **specific character**
- 1 float attribute which represents the **average** length of uninterrupted **sequences of capital letters**
- 1 integer attribute which represents the length of **longest** uninterrupted **sequence of capital letters**
- 1 integer attribute wich represents the total number of **capital letters in the e-mail**
- 1 nominal class attribute which denotes whether the e-mail was considered **spam** (1) **or not** (0)

Data importation

We had to store all attributes names from « document.name » in a list, in order to use it as the columns' names in our dataframe.

```
import pandas as pd
import numpy as np

with open('spambase/spambase.names','r') as f:
    features = []
    for line in f.readlines()[33:]:
        features.append(line.split(':')[0])
    features.append('spam')
    f.close()

spambase_df = pd.read_csv('spambase/spambase.data', names=features)
```

First look

- The dataframe contains **4601 observations** and **58 attributes.**

- We don’t have **any missing value** in the dataset.

```
print(f"Shape of the original spambase Data Frame : {spambase_df.shape}")
```

```
Shape of the original spambase Data Frame : (4601, 58)
```

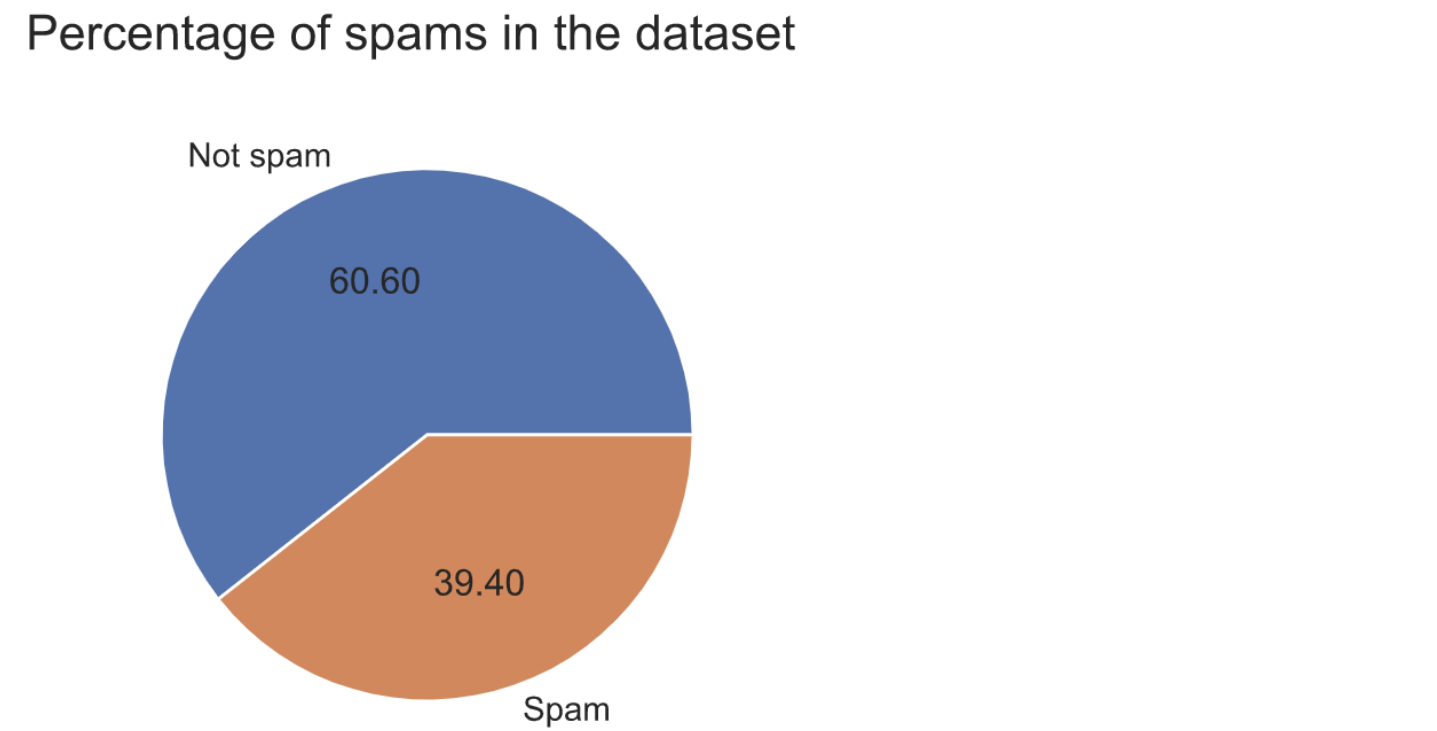
```
spambase_df.isnull().values.any()
```

```
False
```

Visualization

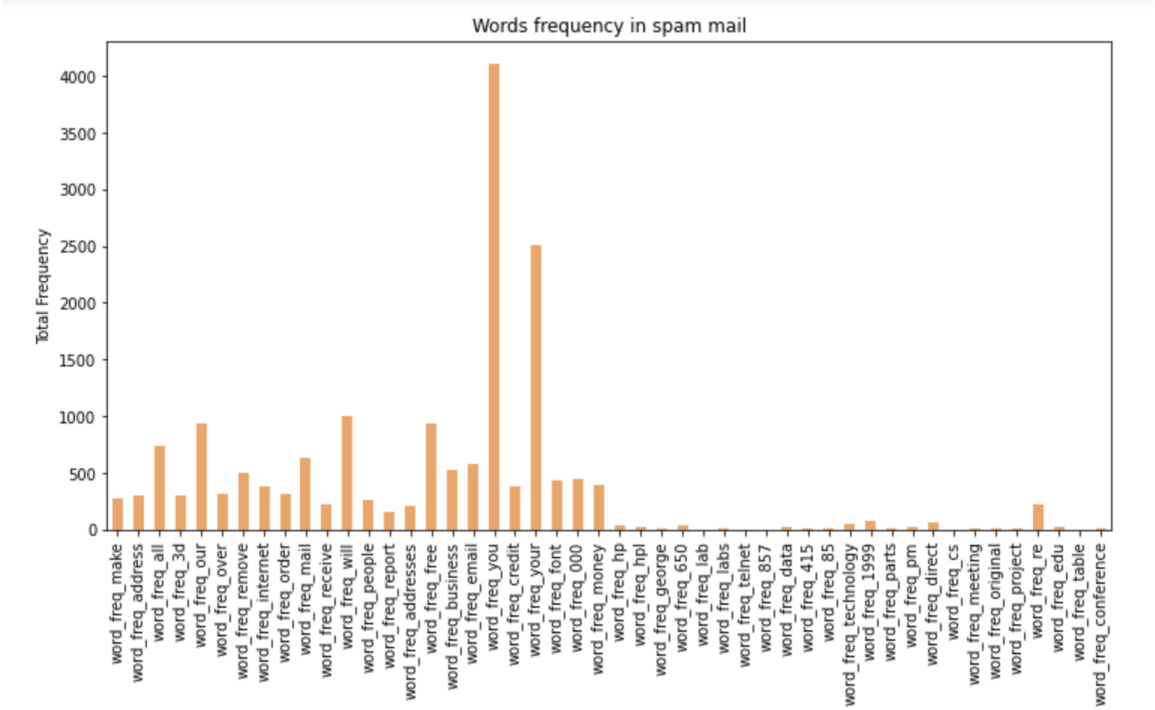
- Among the 4601 instances, **39.4% are spam** emails and **60.6% are not**.

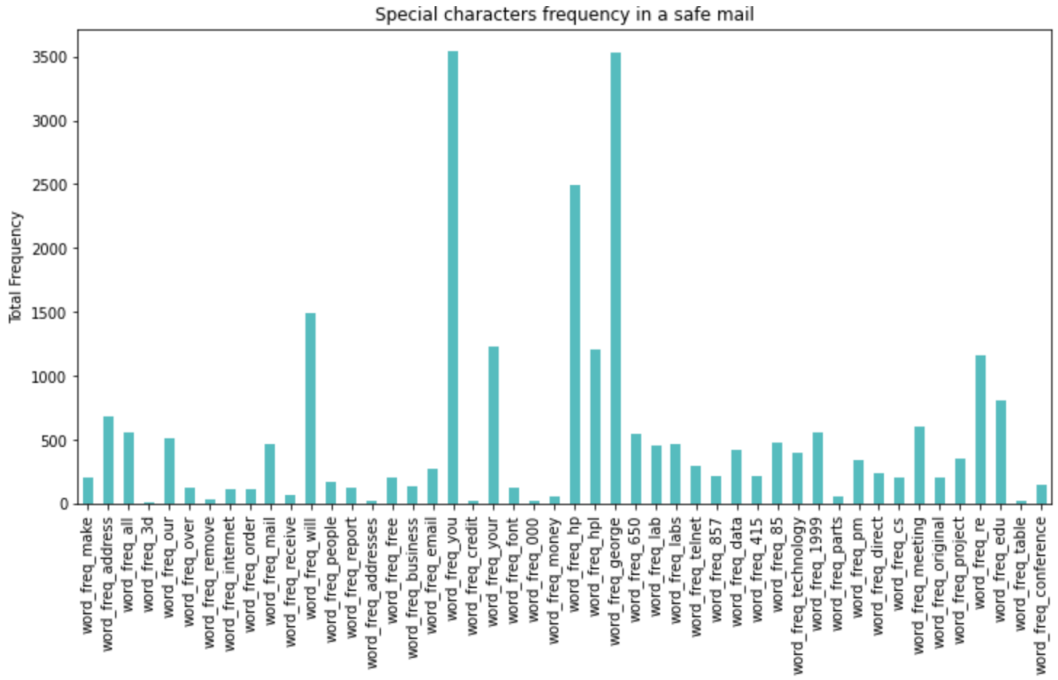
```
spam_prctg_df = spambase_df.spam.value_counts(normalize=True).to_frame()

plt.figure()
plt.pie(spam_prctg_df, labels=['Not spam', 'Spam'], autopct='%.2f')
plt.suptitle("Percentage of spams in the dataset", fontsize=16)

plt.show()
```



Words frequency

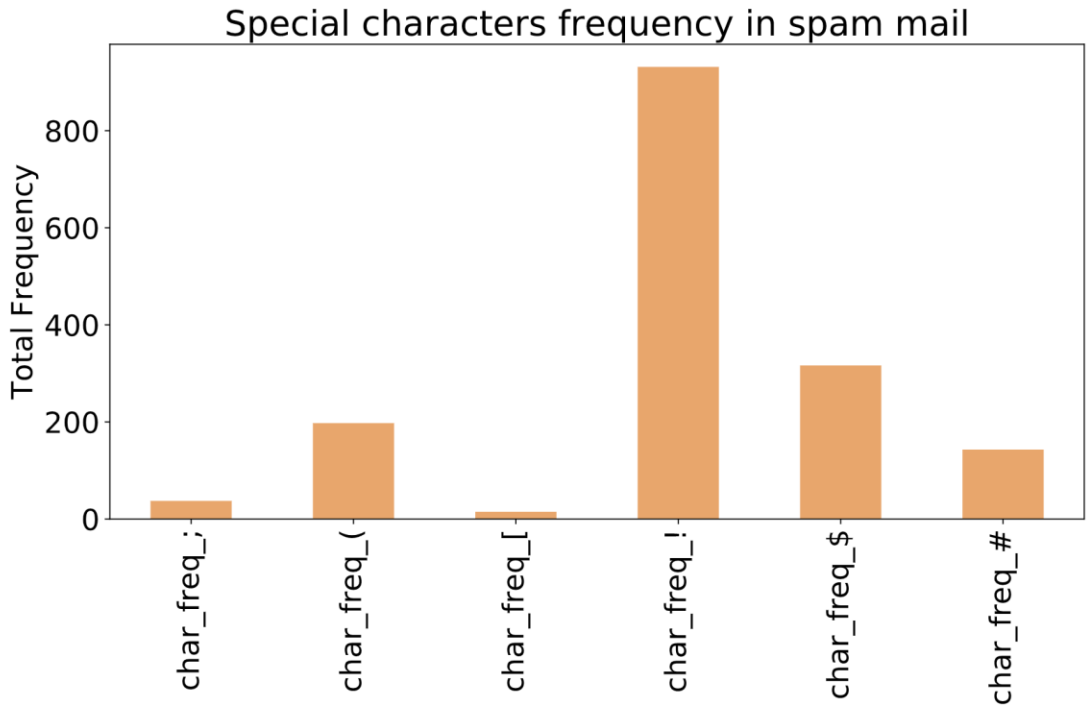
We first displayed the frequency of specific words in safe, and spam email. We can see that words like « your » are much more present in spams, while words like « George » (the donor's name), « hp », « hpl » are more frequent in safe emails.

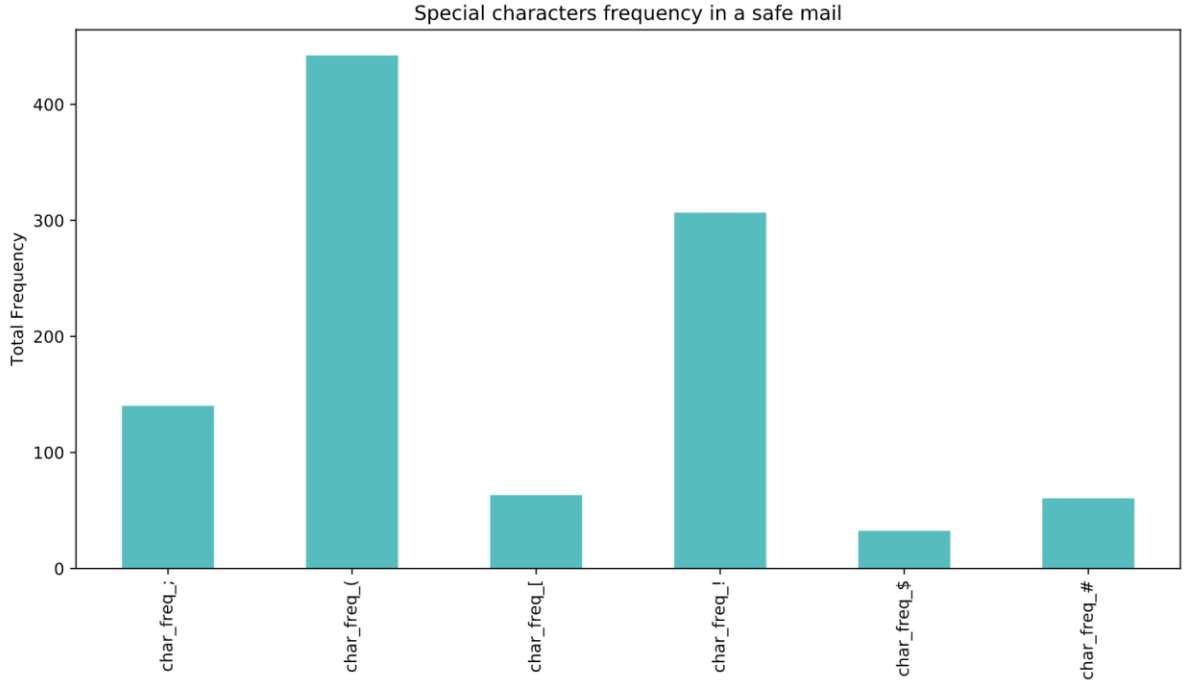




Special characters frequency

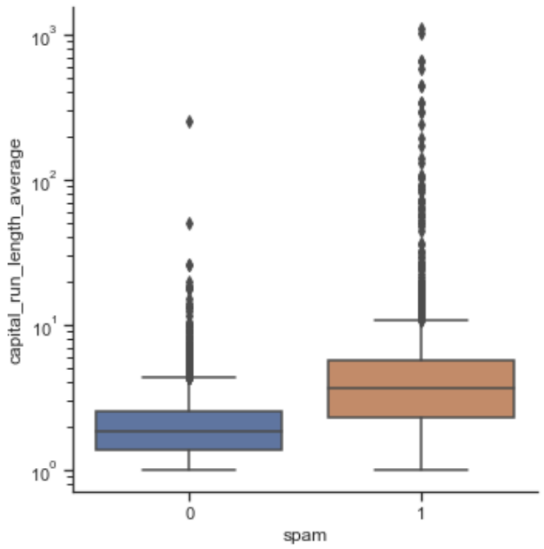
- Then we did the same plots for the special characters. We can see that « ! » is way more frequent in spam mails, while parenthesis are more present in safe mails

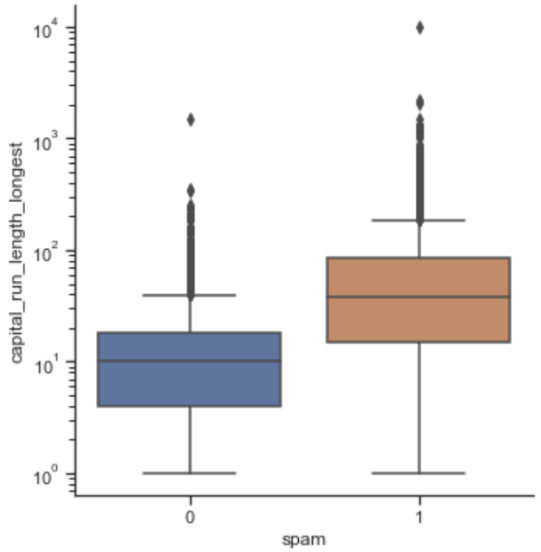


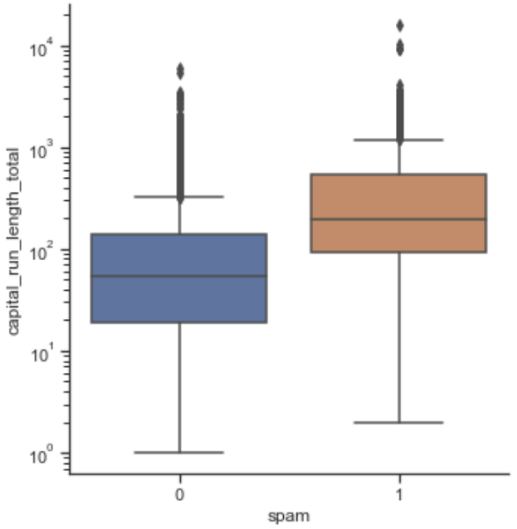


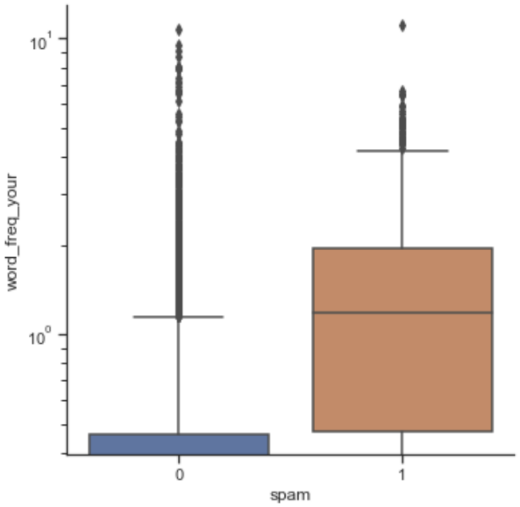
Boxplots

We decided to display some boxplots using the capital letter variables, and some frequency variables



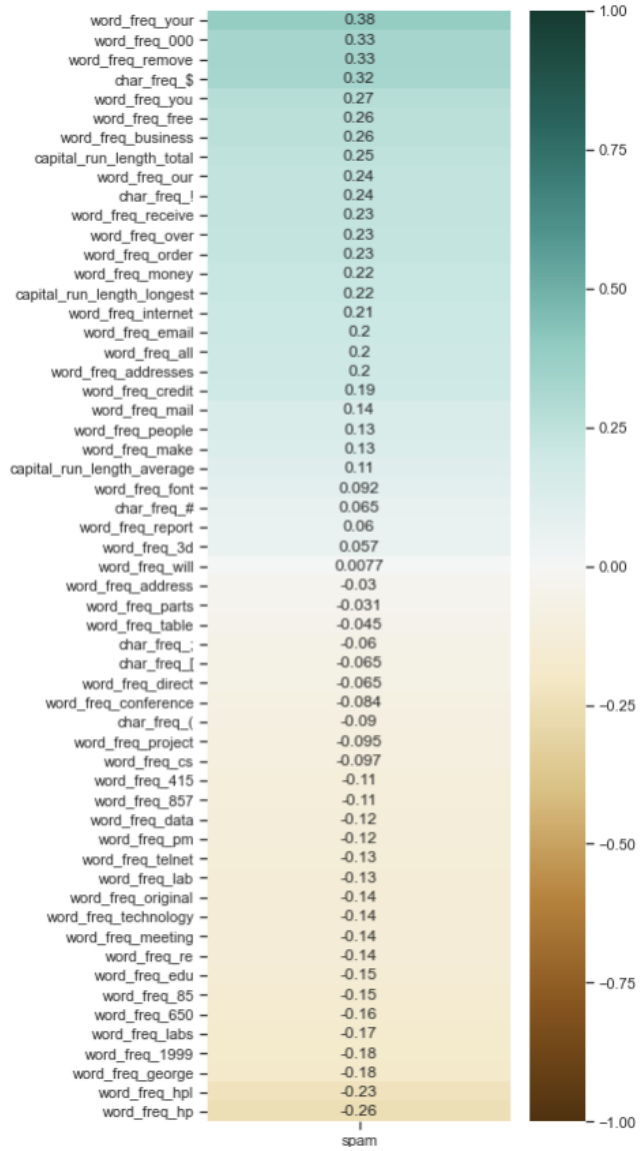






Correlation heatmap

- We also displayed a **correlation heatmap** to see the correlation between the features and wether the email is a spam or not. The higher the correlation, the more significant the feature is to say that the email is a spam.
- We can see here, like in the previous plots, that the frenquency of words like « your », and on the other side « hp », are a **great indicators** to know the nature of the email.



Modelization

- We first had to **split** our data set in a **training** (75%) and a **test** set (25%) by using the train_test_split function from scikit-learn library

```python
from sklearn.model_selection import train_test_split

X = spambase_df.iloc[:,:-1]     # Removing the spam column from the original dataset
Y = spambase_df.spam            # choosing only the spam column from the dataset

# Splitting : we're train with 75% and keep 25% for testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)

print(f"Shape of the train data values : {X_train.shape}")
print(f"Shape of the test data values : {X_test.shape}\n")

print(f"Shape of the train label values : {Y_train.shape}")
print(f"Shape of the test label values :  {Y_test.shape}")
```

```
Shape of the train data values : (3450, 57)
Shape of the test data values : (1151, 57)

Shape of the train label values : (3450,)
Shape of the test label values :  (1151,)
```

Models

We fitted 4 different models using all the attributes :

- Linear discriminant analysis
- Logistic regression
- Gradient boosting classifier
- Random Forest classifier

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
```

Model evaluation

- We had to discuss about the **best metric** to use to evaluate our model. The **accuracy** was the first metric we tought about, but it doesn't take into account the **cost of False Positive** cases. Indeed, if the model classifies safe emails as spams, it will be a real problem for the user.
- So we decided to focus on the **precision** (= True Positive/Total Predicted Positive), which is a far better metric when the **cost of False Positive is high.**

First results

Using the metrics we just described, we selected the **Gradient Boosting** and **Random Forest** classifiers as our best models

```
comparison_df = pd.DataFrame(data=zip([models[model]['name'] for model in models],
 [models[model]['accuracy'] for model in models],
 [models[model]['precision'] for model in models]),
 columns=['model', 'accuracy', 'precision'])

comparison_df.sort_values(by=['precision'], ascending=False)
```

	model	accuracy	precision
3	Random Forest Classifier	0.94005	0.93917
2	Gradient Boosting Classifier	0.93397	0.92344
1	Logistic Regression	0.91573	0.90511
0	Linear Discriminant Analysis	0.88705	0.90107

Tuning

As the Random Forest model had the best precision and accuracy, we decided to do some tuning on it

```
: randForest.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 '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,
 'random_state': 10,
 'verbose': 0,
 'warm_start': False}
```

Randomized Search

First, we created a grid with large vectors of possible values in order to have a first idea of the parameters value that we want, using the RandomizedGridCV function. It gave us the following parameters. However, the new model didn't bring much more improvement (+ 0.005% to 0.001% depending on the test/train sets)

```
{'n_estimators': 1800,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 30,
 'bootstrap': False}
```

Grid Search

Based on the results from the random research, we created a new grid with a shorter range for each hyperparameter in order to optimize again our model

```
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [30, 60, 80, 100],
    'max_features': ['sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [1000, 1800, 2000]
}
```

Final model

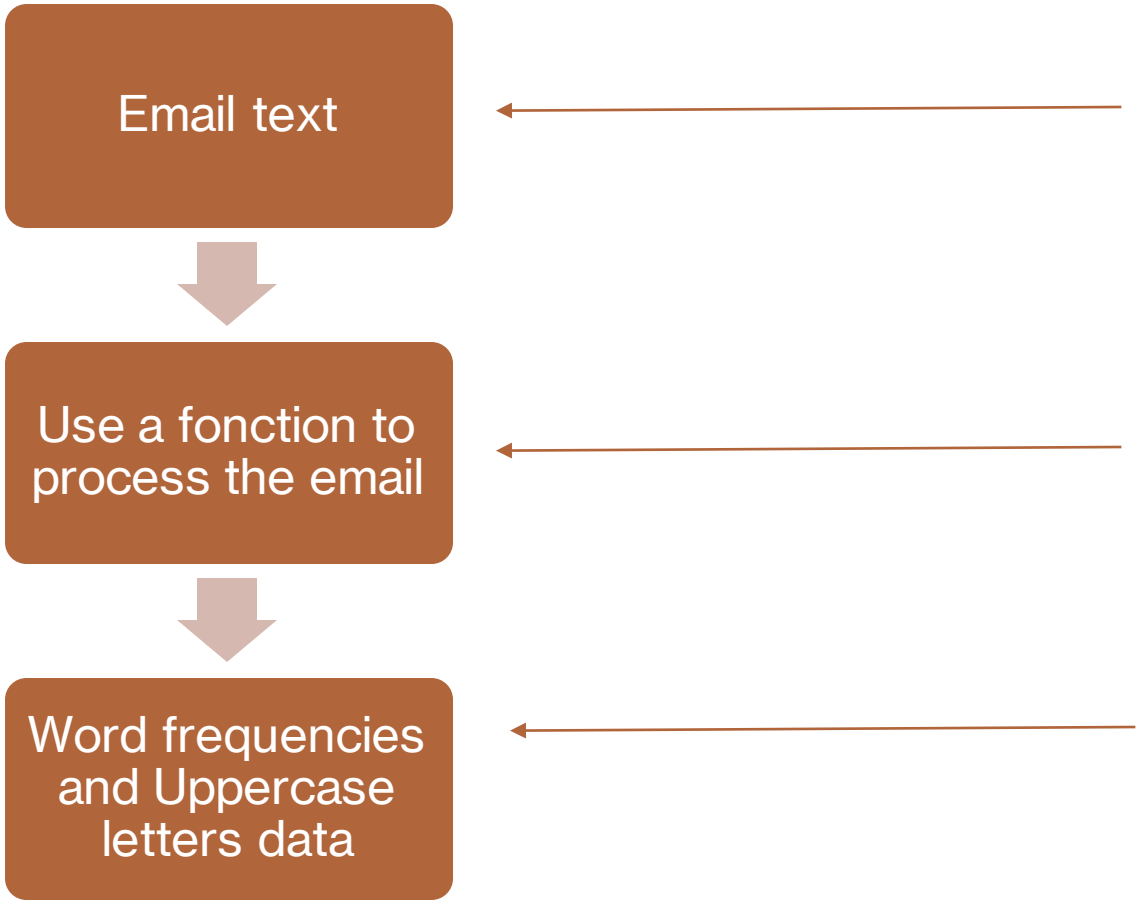
Then we used this grid as our parameter for the GridSearchCV function, which gave us the same results.

The tuning results were different depending on the train/test sets split. Even if we expected more, it enhanced our model precision.

```
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [30, 60, 80, 100],
    'max_features': ['sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [1000, 1800, 2000]
}
```

Flask API

To predict with our API, we need to process an input email to get some characteristics



This data is just a string

The function « process_email » get the frequency of some words and characters using regex operations

Those data can be used directly to predict

Flask API

Using the Random Forest model, we created a Flask API with a route to predict if an email is a SPAM or valide.

POST Request to the API on /api/predict

JSON Body with the following content :

```
{
    "content": "example email content"
}
```

JSON Response with the following content :

```
{
    "prediction": true,
}
```

true = SPAM
false = valid

Interface

To use the API, we created an interface using ReactJS.

Working interface at https://python-data.linkable.tech/