```
mirror_object
                    peration == "MIRROR_X":
                   irror_mod.use_x = True
                   mirror_mod.use_y = False
                   "Irror_mod.use_z = False
                    _operation == "MIRROR_Y":
                    lrror_mod.use_x = False
                    irror_mod.use_y = True
                    lrror_mod.use_z = False
                     _operation == "MIRROR_Z"
                     rror_mod.use_x = False
                     __mod.use_y = False
                     _rror_mod.use_z = True
                     melection at the end -add
                      ob.select= 1
                     er ob.select=1
                      ntext.scene.objects.action
                     "Selected" + str(modified
Python for Data
Analysis select exactly Analysis Sor Classes
```

Seqqat Othman - Jullien Maxime - Dobosz Adam

Introduction

We decided to work on the <u>Spambase Data Set</u>. 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
- <u>1 nominal class attribute</u> 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()
```

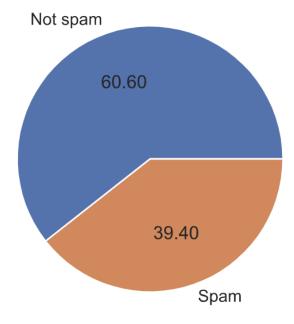
False

Visualization

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

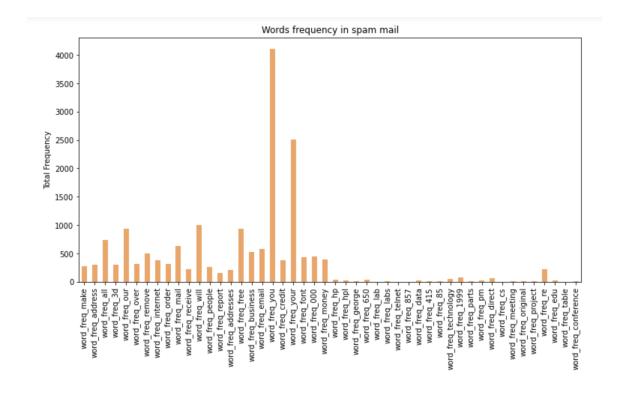
```
spam_prctg_df = spambase_df.spam.value_counts(normalize=True).to_frame()
plt.figure()
plt.pie(spam_prctg_df, labels=['Not spam', 'Spam'], autopct='%.2f')
plt.suptitle("Percentage of spams in the dataset", fontsize=16)
plt.show()
```

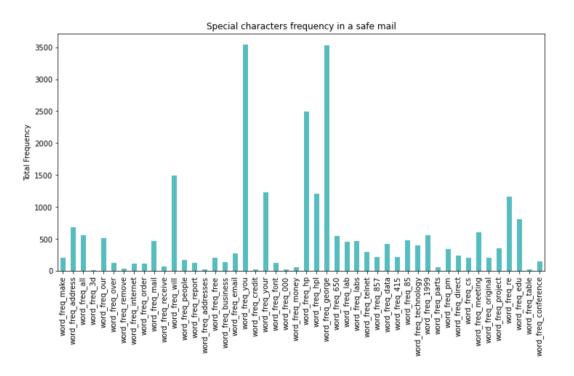
Percentage of spams in the dataset



Words frequency

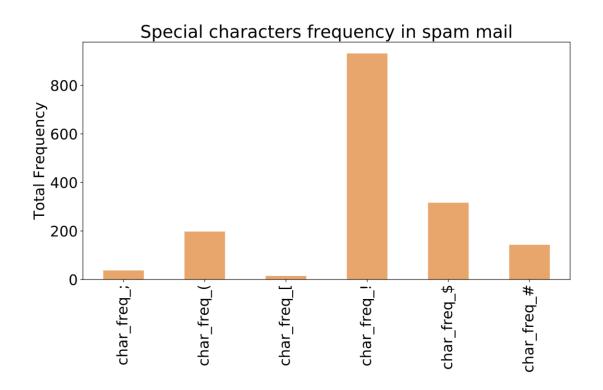
We first displayed the frequency of specific words in safe, and spam email. We can see that words like "your " are much more present in spams, while words like "George" (the donor's name), "hp ", "hpl" are more frequent in safe emails.

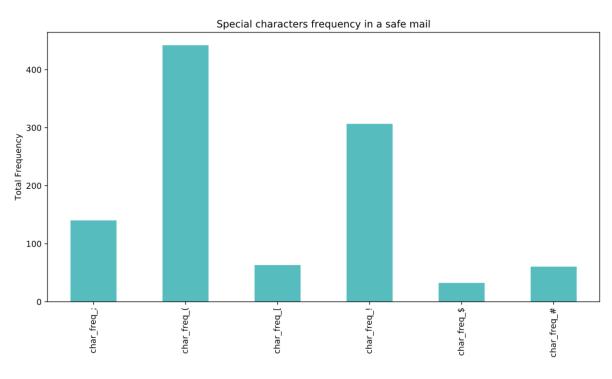




Special characters frequency

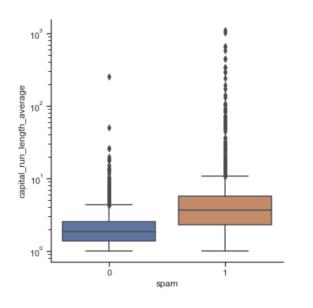
 Then we did the same plots for the special characters. We can see that «!» is way more frequent in spam mails, while parenthesis are more present in safe mails

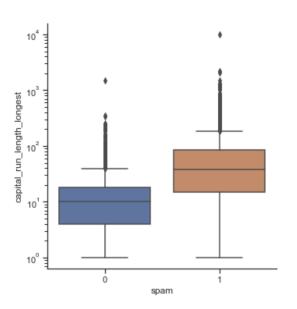


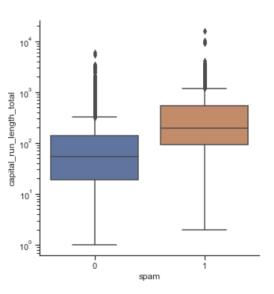


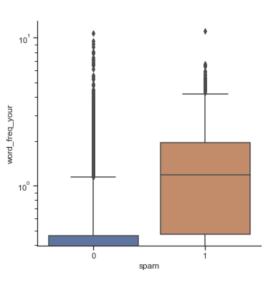
Boxplots

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.

word_freq_000			- 1.00
word_freq_you	word_freq_your -		- 1.00
Char_freq_\$ word_freq_you			
word_freq_vou word_freq_free = 0.26 word_freq_business = 0.26 capital_run_length_total = 0.25 word_freq_our = 0.24 word_freq_our = 0.24 word_freq_our = 0.23 word_freq_over = 0.23 word_freq_order = 0.23 word_freq_internet = 0.21 word_freq_internet = 0.21 word_freq_all = 0.2 word_freq_all = 0.2 word_freq_all = 0.14 word_freq_mail = 0.14 word_freq_mail = 0.14 word_freq_mail = 0.11 word_freq_fred_internet = 0.13 capital_run_length_loged = 0.13 word_freq_all = 0.14 word_freq_addresses = 0.2 word_freq_mail = 0.11 word_freq_fred_internet = 0.065 word_freq_fred_internet = 0.065 word_freq_fred_internet = 0.065 word_freq_fred_internet = 0.0065 word_freq_dinternet = 0.0066 word_freq_dinternet = 0.0066 word_freq_dinternet = 0.0066 word_freq_dinternet =			
word_freq_pusiness - 0.26 word_freq_pusiness - 0.26 capital_run_length_total - 0.25 word_freq_our - 0.24 word_freq_our - 0.24 word_freq_our - 0.23 word_freq_over - 0.23 word_freq_order - 0.23 word_freq_order - 0.23 word_freq_money - 0.22 capital_run_length_longest - 0.22 word_freq_enail - 0.2 word_freq_enail - 0.2 word_freq_enail - 0.2 word_freq_enail - 0.2 word_freq_enail - 0.14 word_freq_enail - 0.19 word_freq_order - 0.13 capital_run_length_enail - 0.14 word_freq_mail - 0.14 word_freq_mail - 0.14 word_freq_freq_inil - 0.11 word_freq_mail - 0.11 word_freq_freq_font - 0.092 capital_run_length_enail - 0.065 word_freq_total - 0.065 word_freq_will - 0.0077 word_freq_dotal - 0.065 word_freq_dotal - 0.031 word_freq_dotal - 0.031 word_freq_dotal - 0.031 word_freq_dotal - 0.045 char_freq_i - 0.065 word_freq_direct - 0.065 word_freq_direct - 0.065 word_freq_freq_cos - 0.097 word_freq_freq_to - 0.084 char_freq_i - 0.099 word_freq_total - 0.11 word_freq_cos - 0.097 word_freq_data - 0.12 word_freq_data - 0.12 word_freq_data - 0.12 word_freq_data - 0.13 word_freq_total - 0.14 word_freq_total - 0.15 word_freq_total - 0.15 word_freq_total - 0.15 word_freq_dats - 0.15 word_freq_dats - 0.15 word_freq_flab - 0.16 word_freq_lab - 0.17 word_freq_lab - 0.18 word_freq_lab - 0.23 word_freq_hpl - 0.23 word_freq_hpl - 0.23			
word_freq_business -			
Capital_run_length_total			
Capital_run_lengm_total			- 0.75
Char_freq_! -			0.70
word_freq_ever			
word_freq_over			
word_freq_money - 0.22 capital_run_length_longest - 0.22 word_freq_manel - 0.21 word_freq_manel - 0.2 word_freq_manel - 0.2 word_freq_all - 0.2 word_freq_addresses - 0.2 word_freq_amail - 0.19 word_freq_manel - 0.19 word_freq_manel - 0.13 word_freq_manel - 0.13 capital_run_length_average - 0.11 word_freq_freq_will - 0.065 word_freq_address - 0.03 word_freq_address - 0.03 word_freq_address - 0.03 word_freq_bart - 0.03 word_freq_bart - 0.03 word_freq_ton - 0.065 char_freq_i - 0.065 word_freq_ton - 0.084 char_freq_i - 0.065 word_freq_core - 0.094 word_freq_ton - 0.095 word_freq_ton - 0.095 word_freq_ton - 0.095 word_freq_ton - 0.095 word_freq_ton - 0.11 word_freq_ton - 0.12 word_freq_ton - 0.12 word_freq_ton - 0.13 word_freq_ton - 0.14 word_freq_ton - 0.14 word_freq_ton - 0.15 word_freq_dab - 0.16 word_freq_dab - 0.17 word_freq_lab - 0.18 word_freq_lab - 0.18 word_freq_lab - 0.18 word_freq_pi - 0.23 word_freq_hi - 0.23			
capital_run_length_longest			
capital_run_length_longest			
Capital_rull_eligits_longest			- 0.50
word_freq_email			0.00
word_freq_all word_freq_addresses 0.2 word_freq_addresses 0.2 word_freq_mail word_freq_mail 0.14 word_freq_mail 0.14 word_freq_meal 0.13 -0.25 word_freq_meal 0.13 -0.25 word_freq_meal 0.11 word_freq_font 0.092 word_freq_font 0.006 word_freq_font 0.006 word_freq_dill 0.0077 -0.00 word_freq_dill 0.005 word_freq_dill 0.005 word_freq_dill 0.065 word_freq_table 0.045 word_freq_table 0.065 word_freq_table 0.065 word_freq_table 0.065 word_freq_table 0.065 word_freq_table 0.095 word_freq_table 0.095 word_freq_table 0.095 word_freq_table 0.11 word_freq_table 0.11 word_freq_table 0.11 word_freq_table 0.12 word_freq_table 0.12 word_freq_table 0.13 word_freq_table 0.14 word_freq_table 0.14 word_freq_table 0.15 word_freq_table 0.15 word_freq_table 0.15 word_freq_table 0.15 word_freq_table 0.15 word_freq_table 0.16 word_freq_table 0.17 word_freq_table 0.18 word_freq_table 0.23 word_freq_table 0.23 word_freq_table 0.23 word_freq_table 0.26 word_freq_table			
word_freq_credit			
word_freq_credit			
word_freq_mail word_freq_maik word_freq_make word_freq_make word_freq_make word_freq_font word_freq_font word_freq_font word_freq_sid word_freq_sid word_freq_sid word_freq_sid word_freq_will word_freq_will word_freq_barts word_freq_barts word_freq_table word_freq_			
word_freq_people			
word_freq_make			- 0.25
capital_run_length_average -			0.20
word_freq_font			
Char_freq_# - 0.065 Word_freq_ad - 0.057 Word_freq_add - 0.057 Word_freq_address - 0.03 Word_freq_adss - 0.031 Word_freq_table - 0.045 Char_freq_i - 0.06 Char_freq_i - 0.065 Word_freq_direct - 0.065 Word_freq_conference - 0.094 Word_freq_conference - 0.095 Word_freq_conference - 0.097 Word_freq_direct - 0.097 Word_freq_direct - 0.097 Word_freq_direct - 0.11 Word_freq_direct - 0.11 Word_freq_direct - 0.12 Word_freq_direct - 0.13 Word_freq_direct - 0.13 Word_freq_direct - 0.13 Word_freq_direct - 0.14 Word_freq_direct - 0.14 Word_freq_meeting - 0.14 Word_freq_edu - 0.15 Word_freq_freq_edu - 0.15 Word_freq_freq_freq - 0.16 Word_freq_labs - 0.17 Word_freq_labs - 0.18 Word_freq_porge - 0.18 Word_freq_hpi - 0.26 Word_freq_hpi -			
word_freq_report 0.06 word_freq_ad 0.057 word_freq_will 0.0077 word_freq_dadress 0.03 word_freq_parts 0.031 word_freq_table 0.045 char_freq_i 0.065 word_freq_direct 0.065 word_freq_onference 0.084 char_freq_i 0.09 word_freq_project 0.095 word_freq_table 0.11 word_freq_data 0.12 word_freq_data 0.12 word_freq_table 0.13 word_freq_lab 0.14 word_freq_tence 0.14 word_freq_edu 0.14 word_freq_edu 0.15 word_freq_dab 0.16 word_freq_freq_tence 0.16 word_freq_freq_freq 0.16 word_freq_freq_freq 0.17 word_freq_labs 0.17 word_freq_labs 0.17 word_freq_labs 0.17 word_freq_labs 0.18 word_freq_ledip 0.23 word_freq_pore 0.18 word_freq_pore 0.26 word_freq_hp 0.26 word_f			
word_freq_will			
word_freq_will = 0.0077 = 0.00 word_freq_adress = 0.03 word_freq_table = 0.045 dar_freq_! = 0.065 word_freq_liele = 0.065 word_freq_conference = 0.084 dar_freq_(= 0.09 word_freq_conference = 0.097 word_freq_conference = 0.097 word_freq_data = 0.11 word_freq_data = 0.12 word_freq_data = 0.12 word_freq_telnet = 0.13 word_freq_liele = 0.13 word_freq_liele = 0.13 word_freq_liele = 0.14 word_freq_telnet = 0.14 word_freq_telnet = 0.15 word_freq_ender = 0.14 word_freq_ender = 0.14 word_freq_ender = 0.15 word_freq_freq_ender = 0.15 word_freq_freq_ender = 0.15 word_freq_freq_ender = 0.16 word_freq_freq_freq_ender = 0.17 word_freq_freq_freq = 0.18 word_freq_labs = 0.17 word_freq_hpi = 0.23 word_freq_hpi = 0.26 word_freq_hpi = 0.26			
word_freq_address =			- 0.00
word_freq_parts -			0.00
word_freq_table - -0.045 dhar_freq_i - -0.06 word_freq_direct - -0.065 word_freq_direct - -0.084 word_freq_conference - -0.094 dhar_freq_(- -0.09 word_freq_conference - -0.095 word_freq_conference - -0.095 word_freq_conference - -0.095 word_freq_star -0.11 word_freq_star -0.11 word_freq_data - -0.12 word_freq_bala - -0.12 word_freq_lab - -0.13 word_freq_lab - -0.13 word_freq_original - -0.14 word_freq_ebin - -0.14 word_freq_edu - -0.15 word_freq_edu - -0.15 word_freq_fed5 - -0.15 word_freq_fed5 - -0.16 word_freq_labs - -0.17 word_freq_hpi - -0.23 word_freq_hpi - -0.26			
Char_freq_i - - - - - - - - - -			
Char_freq_i			
word_freq_direct =			
thar_freq_(= -0.09 word_freq_cs = -0.095 word_freq_cs = -0.097 word_freq_415 = -0.11 word_freq_857 = -0.11 word_freq_data = -0.12 word_freq_telnet = -0.13 word_freq_lab = -0.13 word_freq_loriginal = -0.14 word_freq_lechnology = -0.14 word_freq_tendeting = -0.15 word_freq_ed = -0.15 word_freq_ed = -0.15 word_freq_85 = -0.15 word_freq_85 = -0.15 word_freq_1abs = -0.17 word_freq_labs = -0.17 word_freq_labs = -0.18 word_freq_laps = -0.18 word_freq_leorge = -0.18 word_freq_hpl = -0.23 word_freq_hp = -0.26		-0.065	
Char_freq_(word_freg_conference -	-0.084	0.25
word_freq_cs	char_freq_(-	-0.09	3.20
word_freq_415 -	word_freq_project =	-0.095	
word_freq_857 -	word_freq_cs -	-0.097	
word_freq_data =	word_freq_415 -	-0.11	
word_freq_pm -	word_freq_857 =	-0.11	
word_freq_telnet = -0.13 word_freq_lab = -0.14 word_freq_technology = -0.14 word_freq_meeting = -0.14 word_freq_ree = -0.15 word_freq_ged = -0.15 word_freq_650 = -0.15 word_freq_labs = -0.17 word_freq_labs = -0.17 word_freq_labs = -0.18 word_freq_laps = -0.18 word_freq_peorge = -0.18 word_freq_hpl = -0.23 word_freq_hp = -0.26	word_freq_data -		
word_freq_lab = -0.13 word_freq_original0.14 word_freq_technology = -0.14 word_freq_meeting0.14 word_freq_re = -0.14 word_freq_edu0.15 word_freq_6500.15 word_freq_6500.16 word_freq_labs0.17 word_freq_labs0.18 word_freq_laps0.18 word_freq_freq_freq_freq_freq_freq_freq_freq	word_freq_pm -		0.50
word_freq_original = -0.14 word_freq_technology = -0.14 word_freq_meeting = -0.14 word_freq_re = -0.14 word_freq_edu = -0.15 word_freq_650 = -0.16 word_freq_labs = -0.17 word_freq_labs = -0.17 word_freq_legorge = -0.18 word_freq_teq_hpl = -0.23 word_freq_hpl = -0.26 word_freq_hp = -0.26	word_freq_telnet =		
word_freq_technology -			
word_freq_meeting =			
word_freq_re			
word_freq_edu = -0.15 = -0.75 word_freq_85 = -0.15 word_freq_650 = -0.16 word_freq_labs = -0.17 word_freq_1999 = -0.18 word_freq_peorge = -0.18 word_freq_hpl = -0.23 word_freq_hp = -1.00			
word_freq_85 = -0.15 word_freq_650 = -0.16 word_freq_labs = -0.17 word_freq_1999 = -0.18 word_freq_peorge = -0.18 word_freq_hpl = -0.23 word_freq_hp = -1.00			
word_freq_650 = -0.16 word_freq_labs = -0.17 word_freq_1999 = -0.18 word_freq_george = -0.18 word_freq_hpl = -0.23 word_freq_hp = -1.00			0.75
word_freq_labs = -0.17 word_freq_1999 = -0.18 word_freq_george = -0.18 word_freq_hpl = -0.23 word_freq_hp = -1.00			
word_freq_19990.18 word_freq_george0.18 word_freq_hpl0.23 word_freq_hp1.00			
word_freq_peorge0.18 word_freq_hpl0.23 word_freq_hp1.00			
word_freq_hpl0.23 word_freq_hp1.00			
word_freq_hp			
-1.00			
spam	word_ireq_np =		1.00
		spam	

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

```
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 : {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 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

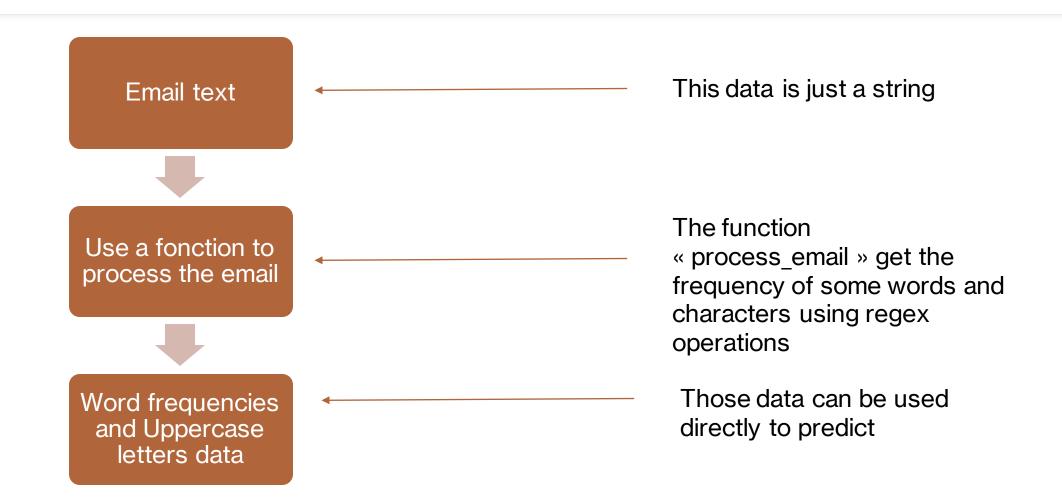
Then we used this grid as our parameter for the GridSearchCV function, which gave us the same results.

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



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:

```
JSON Response with the following content:

1 {
2     "prediction": true,
3 }

true = SPAM
false = valid
```

Interface

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

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

