Avila
Sébastien Yung Hugo Ravé
The goal of this dataset was to predict which copist out of 12 named labels in the dataset wrote the page.
We had multiple features to predict this : such as interlinear spacing which is the space separating two lines. Or lower and upper margin which is the distance separating the last/first line of the text with the bottom/top of the page… etc

We started by importing pandas and loading the train set as well as the test set.
Datasets were .txt files, so we converted them to .csv

```
import pandas as pd

trainset = pd.read_csv("avila-tr.txt",delimiter=',')
trainset.to_csv('avila-tr.csv')
testset = pd.read_csv("avila-ts.txt",delimiter=',')
testset.to_csv('avila-ts.csv')
```

We then import all the necessary libraries that we'll need to deal with the data. Namely numpy for matrix and array, seaborn (for visualisations), matplotlib.pyplot and sklearn (for prediction models).
The problem we are dealing with is obviously a classification problem. Indeed, we are trying to find which copyist wrote those texts using quantitative values.
We are planning to use KNeighborsClassifier, RandomForestClassifier and SVC from sklearn library.

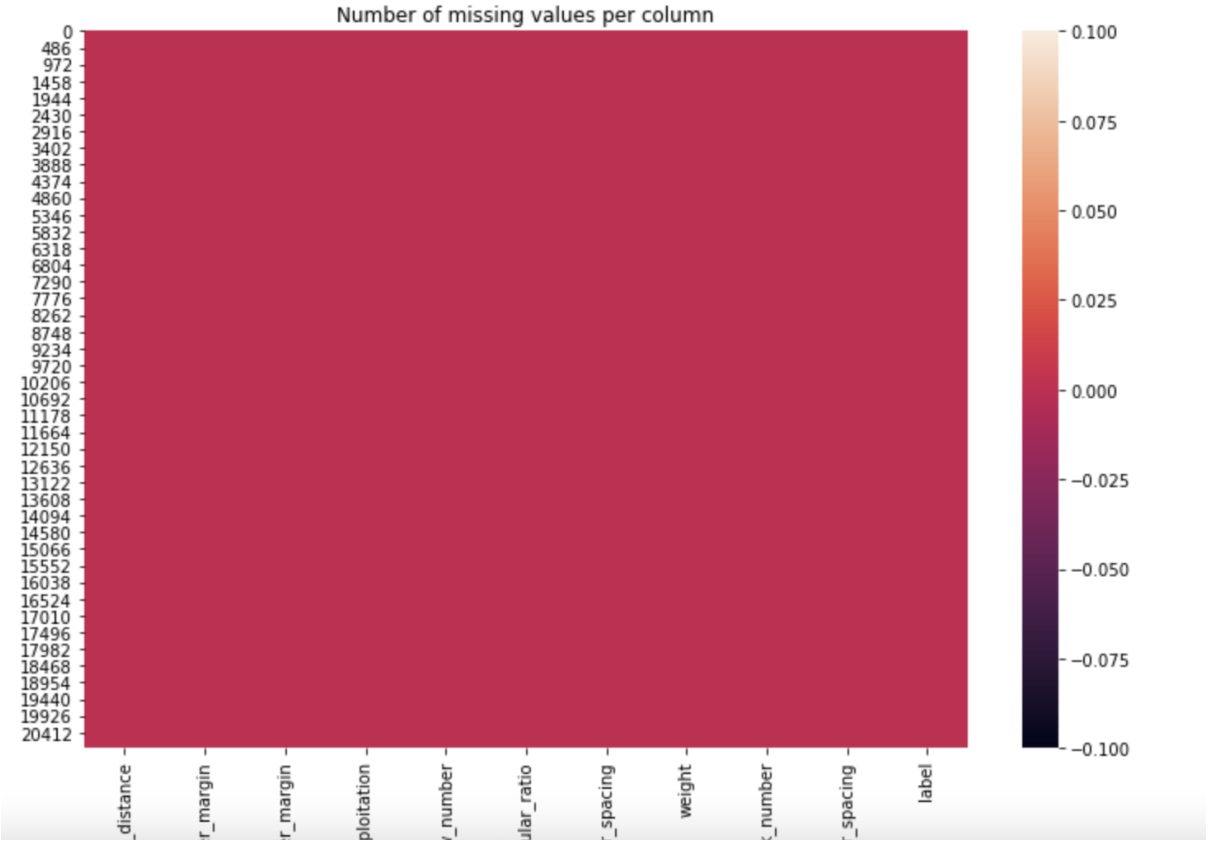
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
```

We then rename all the columns in the train and test sets to make the data set more readable and usable. We also concatenate the train set and test set into a big dataframe.

```
trainset = pd.read_csv('avila-tr.csv', names=['intercolumnar_distance','upper_margin','lower_margin',
                                              'exploitation','row_number','modular_ratio','interlinear_spacing',
                                              'weight','peak_number','modular_ratio/interlinear_spacing','label'])
testset = pd.read_csv('avila-ts.csv', names=['intercolumnar_distance','upper_margin','lower_margin',
                                             'exploitation','row_number','modular_ratio','interlinear_spacing',
                                             'weight','peak_number','modular_ratio/interlinear_spacing','label'])

trainset.to_csv('avila-tr.csv')
testset.to_csv('avila-ts.csv')

print(trainset.shape, testset.shape)
df = pd.concat([trainset, testset], axis=0, ignore_index=True, sort=False)

df.reset_index(inplace=True, drop=True)
df
```

We then want to see where our missing values are. But we can't really see any missing values here.

```
plt.figure(figsize=(12, 8))
sns.heatmap(df.isna(), cbar=True)
plt.title('Number of missing values per column')
plt.show()
```



We then want to see if we have any missing values because we couldn't see any in the graphic just before. As you can see here we don't have any missing values.

```
df.isna().sum()
```

```
intercolumnar_distance                0
upper_margin                          0
lower_margin                          0
exploitation                          0
row_number                            0
modular_ratio                         0
interlinear_spacing                   0
weight                                0
peak_number                           0
modular_ratio/interlinear_spacing     0
label                                 0
dtype: int64
```

We also wanted to see if the data was already standardized which it was because the values are all relatively close to each other.

```
df.head()
```

| | intercolumnar_distance | upper_margin | lower_margin | exploitation | row_number | modular_ratio | interlinear_spacing | weight | peak_number | modular_ratio/in |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.266074 | -0.165620 | 0.320980 | 0.483299 | 0.172340 | 0.273364 | 0.371178 | 0.929823 | 0.251173 | |
| **1** | 0.130292 | 0.870736 | -3.210528 | 0.062493 | 0.261718 | 1.436060 | 1.465940 | 0.636203 | 0.282354 | |
| **2** | -0.116585 | 0.069915 | 0.068476 | -0.783147 | 0.261718 | 0.439463 | -0.081827 | -0.888236 | -0.123005 | |
| **3** | 0.031541 | 0.297600 | -3.210528 | -0.583590 | -0.721442 | -0.307984 | 0.710932 | 1.051693 | 0.594169 | |
| **4** | 0.229043 | 0.807926 | -0.052442 | 0.082634 | 0.261718 | 0.148790 | 0.635431 | 0.051062 | 0.032902 | |

After that, we show the clustermap in order to know if there seems to be related features. It appears that only the modular ratio and the modular ratio/interlinear spacing are strongly correlated as well as the weight and the peak numbers which is slightly less correlated but still much more than any other features in the dataset.
After that we split the data frame back into 2 sets, the training and the testing sets. We also separate our train set as well as our test set into 2 in order to start using prediction models.

```
X_train = trainset.drop('label', axis=1)
y_train = trainset['label']
X_test = testset.drop('label', axis=1)
y_test = testset['label']
```

The first prediction model we used is the linear SVC model. We also used a grid search CV in order to find the best parameters to fit the model.

```
model = LinearSVC(random_state=0)
model
```

```
LinearSVC(random_state=0)
```

```
params = {
    'penalty' : ['l1', 'l2'],
    'loss' : ['hinge', 'squared_hinge'],
    'C' : [0.1, 1, 10],
    'dual': [True, False],
    'multi_class' : ['ovr', 'crammer_singer'],
    'tol' : [0.00001, 0.0001, 0.001]
}
```

```
grid = GridSearchCV(model, param_grid=params, cv = 5)
grid.fit(X_train, y_train)
```

We saved those parameters and used them to fit the model.

```
grid.best_params_
#{'C': 0.1,
# 'dual': True,
# 'loss': 'hinge',
# 'multi_class': 'crammer_singer',
# 'penalty': 'l1',
# 'tol': 0.001}
```

```
grid.best_score_
```

```
model = LinearSVC(C=0.1, dual=True, loss='hinge', multi_class='crammer_singer', penalty='l1', tol=0.001)
model
```

```
LinearSVC(C=0.1, loss='hinge', multi_class='crammer_singer', penalty='l1',
          tol=0.001)
```

```
model.fit(X_train, y_train)
```

```
/Users/sebastienyung/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/_base.py:986: ConvergenceWarni
ear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
LinearSVC(C=0.1, loss='hinge', multi_class='crammer_singer', penalty='l1',
          tol=0.001)
```

```
predictions = model.predict(X_test)
predictions[:20]
```

```
array(['A', 'X', 'I', 'A', 'A', 'A', 'Y', 'A', 'A', 'A', 'A', 'A', 'A',
       'A', 'A', 'A', 'A', 'G', 'I', 'I'], dtype=object)
```

We then put our predictions and the actual labels in a confusion matrix to see how the model performed. We got 54% accuracy on this model, which is a pretty bad result so we decided to test different parameters in the next grid search CV for the linear SVC model but we

ended up having even worse accuracy, 51% so we decided to use a different model.

```
pd.DataFrame(confusion_matrix(y_test, predictions))
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4068 | 2 | 24 | 8 | 27 | 18 | 52 | 52 | 20 | 0 | 12 | 3 |
| **1** | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 67 | 0 | 2 | 1 | 11 | 5 | 4 | 10 | 2 | 0 | 0 | 1 |
| **3** | 283 | 0 | 6 | 0 | 9 | 6 | 7 | 35 | 4 | 0 | 3 | 0 |
| **4** | 651 | 0 | 30 | 10 | 84 | 12 | 36 | 163 | 8 | 0 | 85 | 16 |
| **5** | 1741 | 0 | 28 | 3 | 1 | 32 | 67 | 40 | 33 | 2 | 0 | 15 |
| **6** | 328 | 0 | 7 | 1 | 7 | 7 | 14 | 73 | 2 | 0 | 8 | 0 |
| **7** | 258 | 0 | 17 | 4 | 16 | 2 | 41 | 141 | 34 | 0 | 6 | 1 |
| **8** | 32 | 0 | 2 | 0 | 2 | 2 | 0 | 0 | 771 | 0 | 9 | 14 |
| **9** | 34 | 0 | 6 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 3 | 0 |
| **10** | 56 | 0 | 2 | 1 | 2 | 0 | 8 | 13 | 19 | 0 | 392 | 29 |
| **11** | 35 | 0 | 4 | 1 | 0 | 0 | 0 | 0 | 40 | 0 | 34 | 153 |

```
# % valeurs exactes
(y_test == predictions).sum()/len(y_test)
```

```
0.5425888665325285
```

```
model2 = make_pipeline(PolynomialFeatures(degree=3), StandardScaler(), LinearSVC(C=10, loss='hinge',
                                                                                 multi_class='crammer_singer',
                                                                                 penalty='l2', tol=0.001))
model2
```

```
# % valeurs exactes
(y_test == predictions).sum()/len(y_test)
#0.7499281402701926
```

```
0.5142282265018684
```

We then did the same thing with the KNN and SVC using base parameters at first and then using a grid search CV to find better parameters which boosted our accuracy but we didn't get anything better than 77% for both models.
Then we used a random forest and got 98% accuracy with base parameters !

# Random Forest

```
model7 = RandomForestClassifier()
```

```
model7.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
predictions = model7.predict(X_test)
```

```
# % valeurs exactes
(y_test == predictions).sum()/len(y_test)
```

```
0.9823704129539139
```

We then used a grid search to find better parameters and our accuracy went up a little.

? Getting rid of weird lines ????

Now onto the features selection, we used a feature selector from sklearn. The 2 worst features were weight and modular ratio/interlinear spacing as indicated here. So we got rid of those to have a better model fitting.

```
from sklearn.feature_selection import f_classif, SelectKBest
from sklearn.feature_selection import mutual_info_classif
```

```
X_train.shape, y_train.shape
```

```
((10430, 10), (10430,))
```

```
f_classif(X_train, y_train)
```

```
(array([ 69.36415416,   6.46480319,  12.54638585, 143.4966501 ,
         81.99985336,  91.02632928,  95.6229215 ,  30.03858907,
        214.79469083,  63.52088942]),
 array([7.37511869e-151, 8.22972652e-011, 6.14425216e-024, 7.88522494e-309,
        1.66345550e-178, 4.48034117e-198, 5.54745993e-208, 3.78304949e-063,
        0.00000000e+000, 5.62240920e-138]))
```

```
X_train.columns
```

```
Index(['intercolumnar_distance', 'upper_margin', 'lower_margin',
       'exploitation', 'row_number', 'modular_ratio', 'interlinear_spacing',
       'weight', 'peak_number', 'modular_ratio/interlinear_spacing'],
      dtype='object')
```

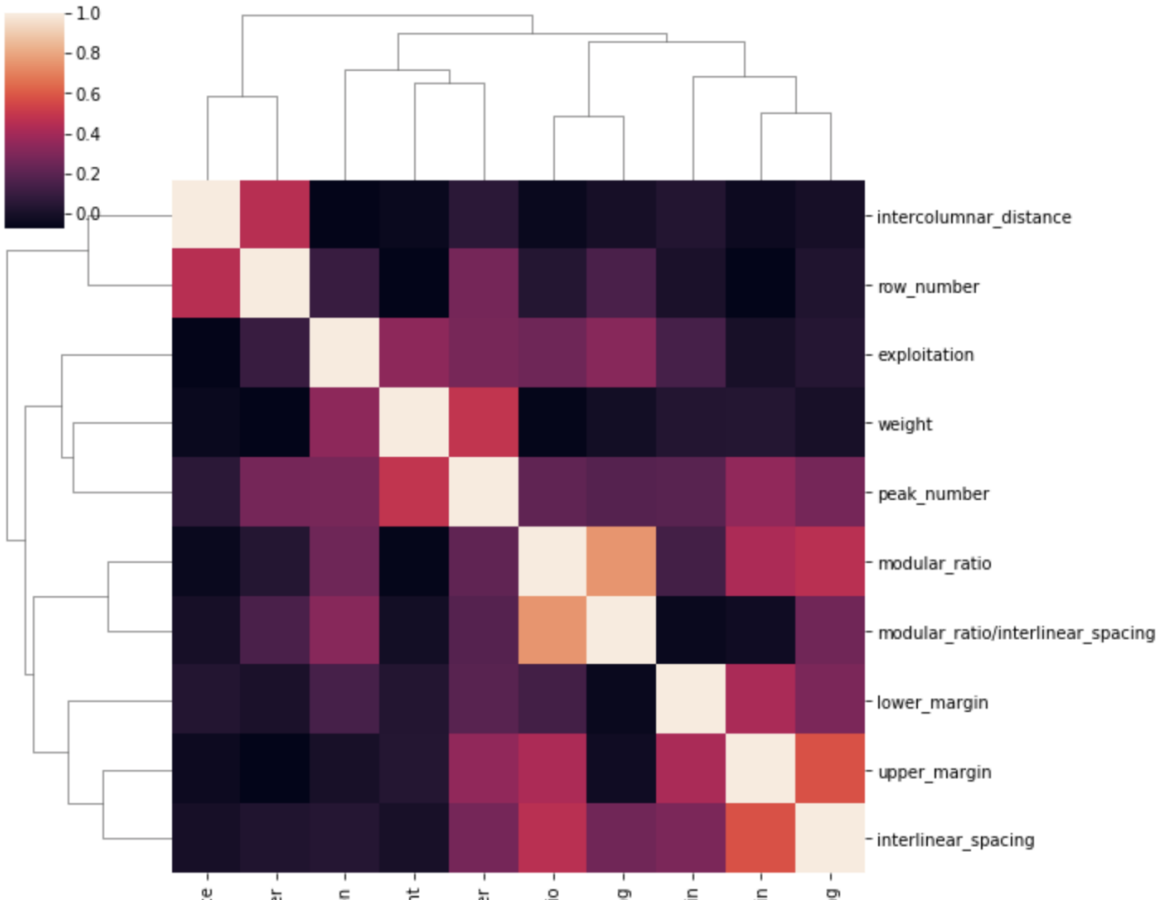
```
selector = SelectKBest(mutual_info_classif, k=8)
selector.fit_transform(X_train, y_train)
X_train.columns[selector.get_support()==False]
```

```
Index(['weight', 'modular_ratio/interlinear_spacing'], dtype='object')
```

Indeed as we can see here most of the features are not related at all with those two features (black cases meaning no correlation)

```
sns.clustermap(trainset.corr())
```

```
<seaborn.matrix.ClusterGrid at 0x7fc731211eb8>
```




Using those new sets where we got rid of those 2 features we got our best accuracy yet, 99.87%, with the random forest model.

We also used a voting classifier, a bagging classifier and a stacking classifier but we didn't get a better accuracy than the random forest.

This is where we made the django part, using our best model, the random forest.

```
import joblib
```

```
filename = 'final_model.sav'
joblib.dump(model9, filename)
```