# Random Forest for Sorting and Counting Corrugated Cardboard Sheets

By: David Hernán García Fernández – A01173130

## 1. Abstract

The present work consists of the implementation of a Random Forest for the classification and accounting of corrugated cardboard sheets of type B, BC, C, E and EB. For this, the basic knowledge necessary to understand the work is explained in the theoretical framework, especially those related to the types of flutes, to later explain the composition of the implemented Random Forest code, as well as the results of it.

**Keywords:** Random Forest, Corrugated cardboard, Cardboard flute.

## 2. Introduction

Companies that work with corrugated cardboard have the need to know how many sheets are stored and the most usual thing is that they are counted manually, which is inefficient. Therefore, they have requested the design of an algorithm with the ability to differentiate and count the sheets that are stacked using computer vision.

In previous semesters it was achieved to extract numerical characteristics from the images provided by the company using the MVTec HALCON software for type B and BC flutes, now the same will be done for type C, E and EB flutes. With these databases, the classification algorithm will be assembled.

## 3. Theoretical framework

### 3.1 Corrugated cardboard and types of flutes

Corrugated cardboard is characterized by combining two elements in its structure: One (or several) sheet of corrugated paper, also known as flute or medium, which acts as a central nerve and gives the corrugated cardboard extra strength; and smooth cardboard sheets (or liners), which are placed outside and function as separators for different layers of flutes (Gonzalez, 2019).

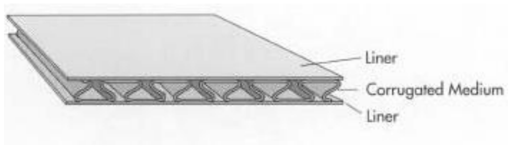


Image 1. Corrugated cardboard components.

Corrugated is classified according to the number of lines or flutes. The flute can be of four types: A, B, C, D and E. According to the construction of the box, the flute can have a horizontal or vertical arrangement, in addition these flutes can be combined with each other (Cantu,2019).

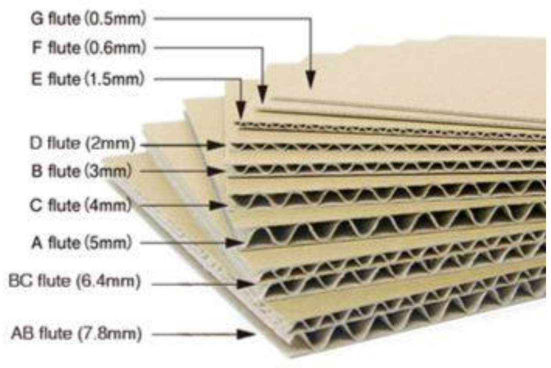


Image 2. Flute types according to industry standards

## 3.2 SciKit Learn Framework

Scikit-Learn is a library for Python that has several clustering, classification, regression and dimensionality reduction algorithms and is compatible with other libraries such as NumPy, SciPy and matplotlib. Its abundant variety of algorithms make it an important tool to start programming Machine Learning systems, data analysis and statistical modeling (Universidad De Alcalá, 2019).

In this project Scikit-Learn is used to create the Random Forest.

## 3.3 Random Forest Classifier

Random forest is a supervised learning algorithm that can be used for both classification and regression problems. What it does is to create an ensemble of decision trees, so that each tree will create its own model and make they own predictions, then the mean or majority of those predictions will be given as the result of the Random Forest. The result obtained will have a higher precision and make a more stable prediction that the one obtained by a single tree (Donges, 2019).

Some of its hyperparameter from Scikit Learn are:

- n_estimators: Number of trees that the algorithm will be built.
- max_features: Maximum number of features that the random forest considers to split a node.
- max_leaf_nodes: Sets the maximum number of leaf nodes.
- n_jobs: How many processors it can use. “-1” means no limit.

(Scikit Learn, s.f.)

The advantages of this algorithm are its versatility, its easy implementation, and its ability to avoid overfitting, as long as there are enough trees, its main disadvantage is that the more trees you have, the slower it will be (Donges, 2019).

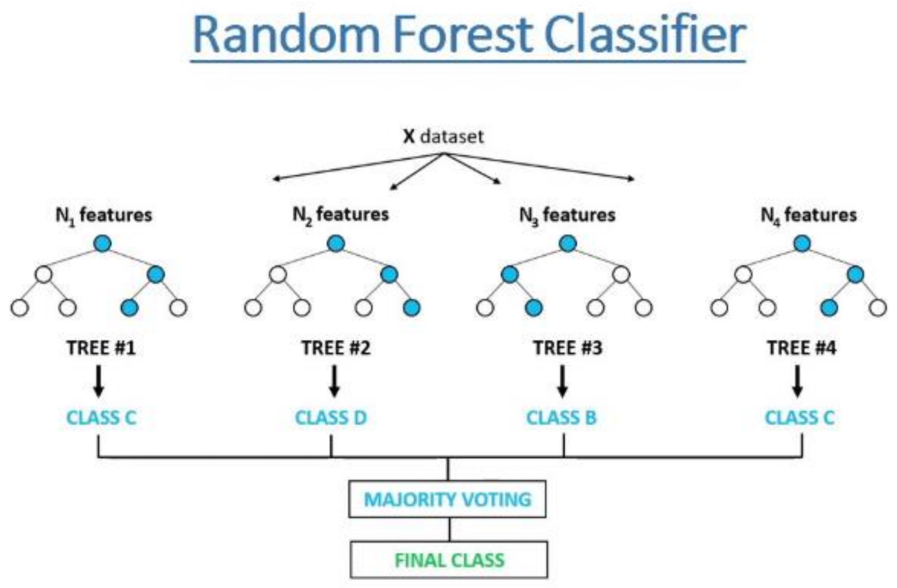


Image 3. Graphic representation of the Random Forest algorithm

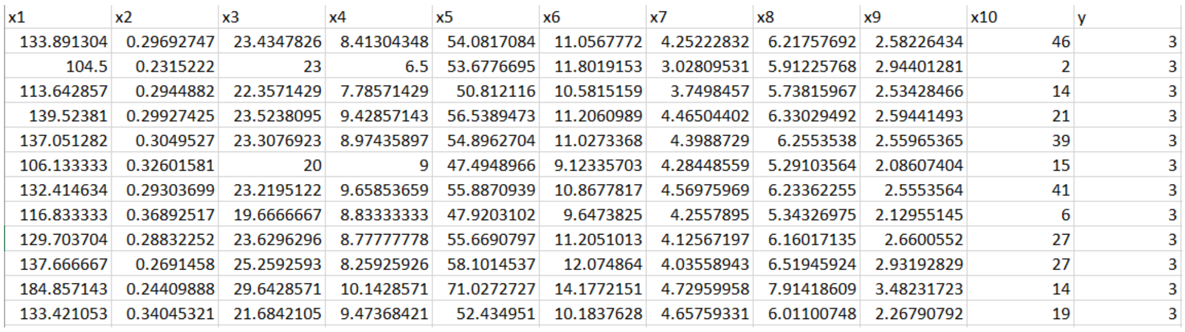
## 4. Implementation

### 4.1 Database

The images of the five types of flutes were provided by the company, those images were used by the code developed in the previous semesters in MVTec Halcon software to obtain five files with a .csv extension, each of which containing the numerical characteristics of each of the flutes, which are: B, BC, C, E and EB.

Each of these files has a similar structure: It has 10 columns, the first nine columns have float numbers, while the last one has only integers. In addition, there were several rows with many zeros in them, so they were manually removed as they affected the efficiency of the program.

To carry out the classification process, a column called "y" was added to each of the five datasets, this column contains a different number for each type of flute: 1 for E, 2 for EB, 3 for B, 4 for C and 5 for BC. Image 4 shows how the datasets used in the program look like.



| x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 | y |
|---|---|---|---|---|---|---|---|---|---|---|
| 133.891304 | 0.29692747 | 23.4347826 | 8.41304348 | 54.0817084 | 11.0567772 | 4.25222832 | 6.21757692 | 2.58226434 | 46 | 3 |
| 104.5 | 0.2315222 | 23 | 6.5 | 53.6776695 | 11.8019153 | 3.02809531 | 5.91225768 | 2.94401281 | 2 | 3 |
| 113.642857 | 0.2944882 | 22.3571429 | 7.78571429 | 50.812116 | 10.5815159 | 3.7498457 | 5.73815967 | 2.53428466 | 14 | 3 |
| 139.52381 | 0.29927425 | 23.5238095 | 9.42857143 | 56.5389473 | 11.2060989 | 4.46504402 | 6.33029492 | 2.59441493 | 21 | 3 |
| 137.051282 | 0.3049527 | 23.3076923 | 8.97435897 | 54.8962704 | 11.0273368 | 4.3988729 | 6.2553538 | 2.55965365 | 39 | 3 |
| 106.133333 | 0.32601581 | 20 | 9 | 47.4948966 | 9.12335703 | 4.28448559 | 5.29103564 | 2.08607404 | 15 | 3 |
| 132.414634 | 0.29303699 | 23.2195122 | 9.65853659 | 55.8870939 | 10.8677817 | 4.56975969 | 6.23362255 | 2.5553564 | 41 | 3 |
| 116.833333 | 0.36892517 | 19.6666667 | 8.83333333 | 47.9203102 | 9.6473825 | 4.2557895 | 5.34326975 | 2.12955145 | 6 | 3 |
| 129.703704 | 0.28832252 | 23.6296296 | 8.77777778 | 55.6690797 | 11.2051013 | 4.12567197 | 6.16017135 | 2.6600552 | 27 | 3 |
| 137.666667 | 0.2691458 | 25.2592593 | 8.25925926 | 58.1014537 | 12.074864 | 4.03558943 | 6.51945924 | 2.93192829 | 27 | 3 |
| 184.857143 | 0.24409888 | 29.6428571 | 10.1428571 | 71.0272727 | 14.1772151 | 4.72959958 | 7.91418609 | 3.48231723 | 14 | 3 |
| 133.421053 | 0.34045321 | 21.6842105 | 9.47368421 | 52.434951 | 10.1837628 | 4.65759331 | 6.01100748 | 2.26790792 | 19 | 3 |

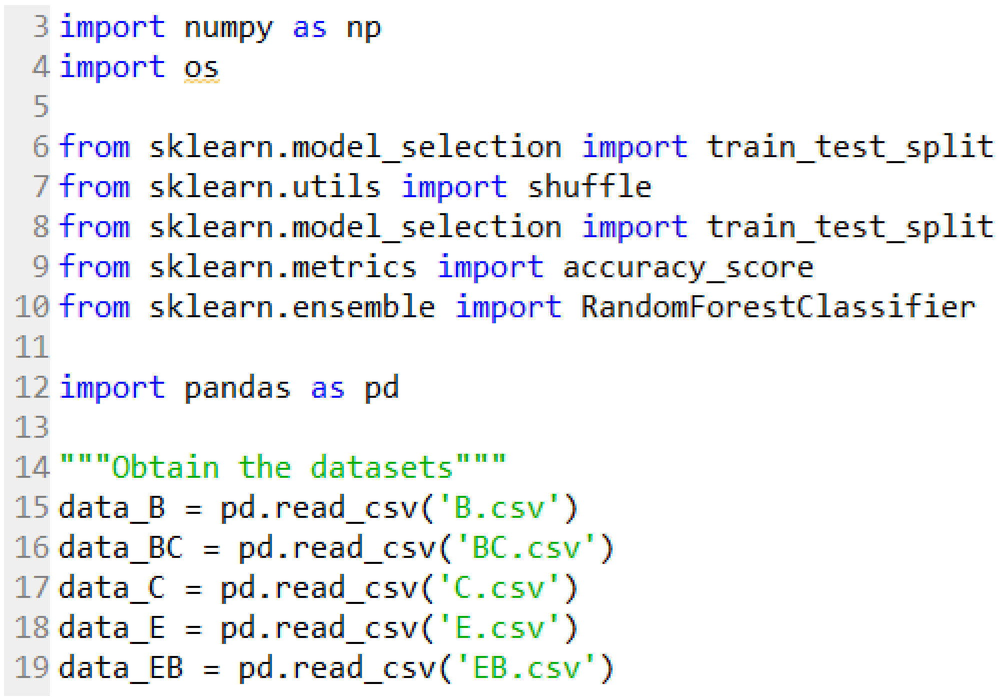
Image 4. Flute Type B Dataset.

The size of each dataset, defined by the number of rows, are the following: 862 for B, 31486 for BC, 10183 for C, 975 for E and 560 for EB.

## 4.1 Coding

The main reason I selected the Random Forest is because of its "resistance" to overfitting and because it is quite useful when you have a small dataset, as in this project.

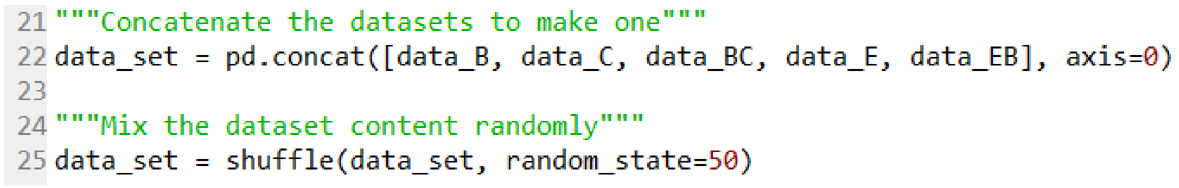
First, I imported all the needed libraries and the datasets.



```python
import numpy as np
import os

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

import pandas as pd

"""Obtain the datasets"""
data_B = pd.read_csv('B.csv')
data_BC = pd.read_csv('BC.csv')
data_C = pd.read_csv('C.csv')
data_E = pd.read_csv('E.csv')
data_EB = pd.read_csv('EB.csv')
```

Image 5. Imports

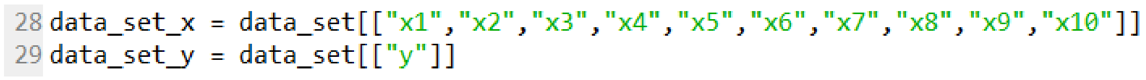
Then I put the five datasets together using the pandas function “concat” to get just one to work with and I save it in “data_set”. Then I mixed randomly the new dataset using the “shuffle” function, this to prevent the training and test sets from consisting of only one type of flute. It should be said that the scaling algorithms worsen the final result a bit.



```python
"""Concatenate the datasets to make one"""
data_set = pd.concat([data_B, data_C, data_BC, data_E, data_EB], axis=0)

"""Mix the dataset content randomly"""
data_set = shuffle(data_set, random_state=50)
```

Image 6. Concatenation and shuffle.

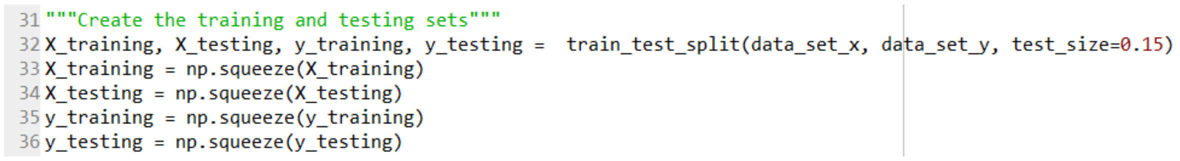
I continued by assigning the first ten columns of the dataset as the X's and the last one as the target or "y".



```
data_set_x = data_set[["x1","x2","x3","x4","x5","x6","x7","x8","x9","x10"]]
data_set_y = data_set[["y"]]
```

Image 7. Target and attributes designation

Then I did the division of the dataset assigning 15% of it for testing purposes. The “squeeze” functions are fixing an error.

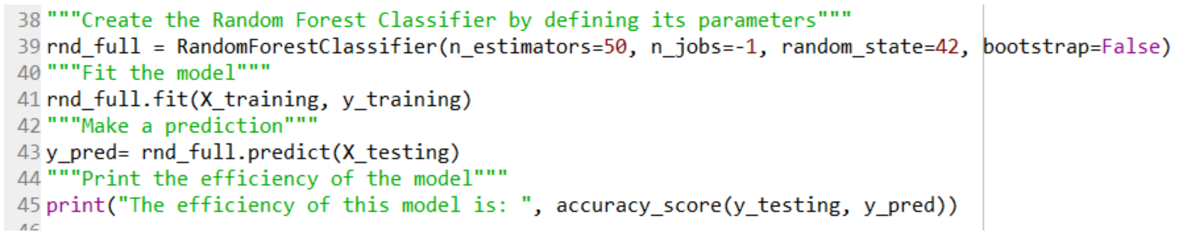


```
"""Create the training and testing sets"""
X_training, X_testing, y_training, y_testing =  train_test_split(data_set_x, data_set_y, test_size=0.15)
X_training = np.squeeze(X_training)
X_testing = np.squeeze(X_testing)
y_training = np.squeeze(y_training)
y_testing = np.squeeze(y_testing)
```

Image 8. Creation of test and training groups

First, I established the Random Forest hyperparameters, which I tested and modified until I achieved the highest possible accuracy which ended up being between 92% and 93% depending on how the shuffle ordered the dataset. Limiting the number of leaf nodes reduces accuracy, as does setting the bootstrap to True.

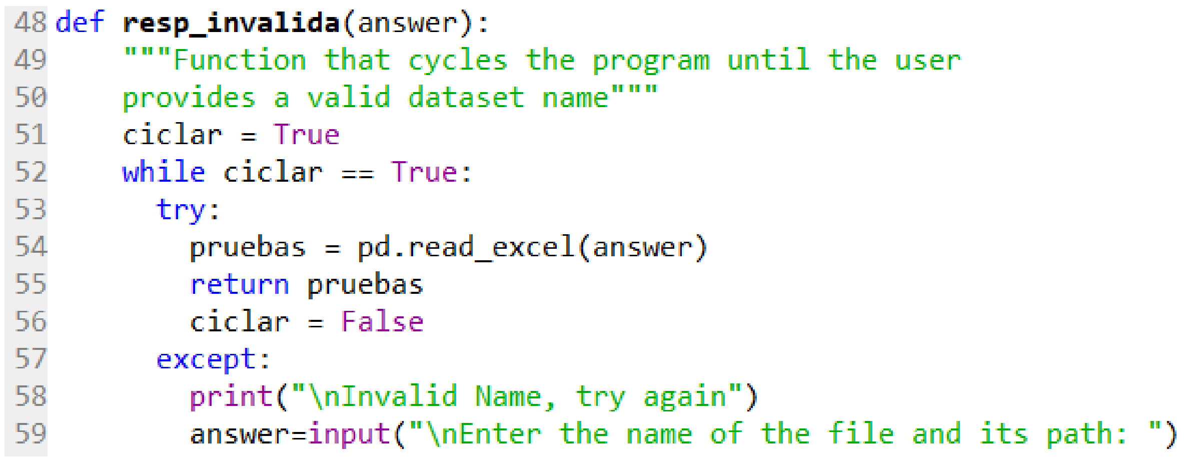
Then the defined Random Forest is fed, and an accuracy test is performed. The result of it is printed so that the user knows it.



```
"""Create the Random Forest Classifier by defining its parameters"""
rnd_full = RandomForestClassifier(n_estimators=50, n_jobs=-1, random_state=42, bootstrap=False)
"""Fit the model"""
rnd_full.fit(X_training, y_training)
"""Make a prediction"""
y_pred= rnd_full.predict(X_testing)
"""Print the efficiency of the model"""
print("The efficiency of this model is: ", accuracy_score(y_testing, y_pred))
```

Image 9. Random Forest creation and accuracy test

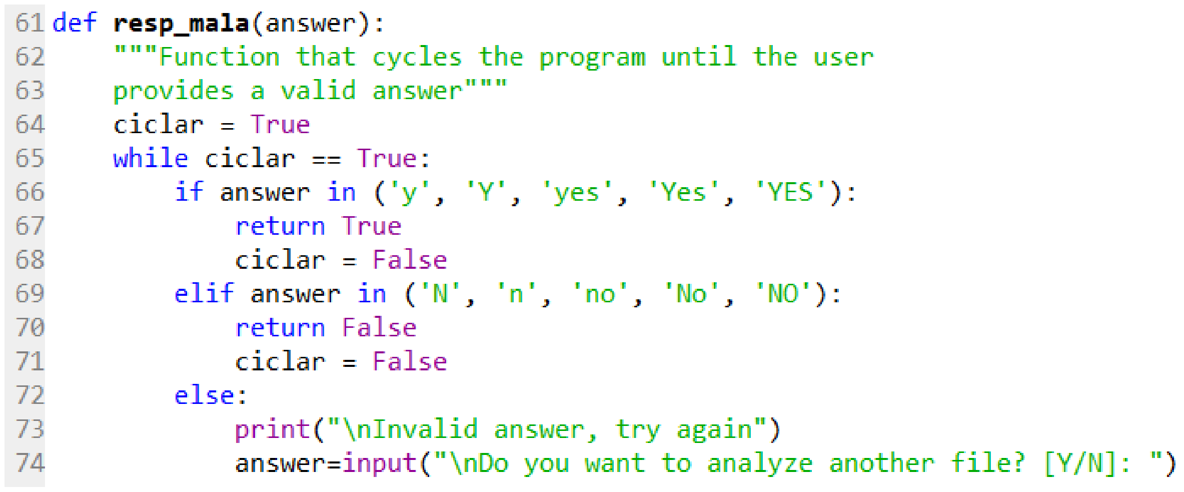
Once the Random Forest is totally defined and created, is time to implement the user interface section. This function oversees if the user has provided an existing file name by sending a warning when not and asking for the name again. The function ends when a valid file name is provided and returns the file information.



```python
def resp_invalida(answer):
    """Function that cycles the program until the user
    provides a valid dataset name"""
    ciclar = True
    while ciclar == True:
      try:
        pruebas = pd.read_excel(answer)
        return pruebas
        ciclar = False
      except:
        print("\nInvalid Name, try again")
        answer=input("\nEnter the name of the file and its path: ")
```

Image 10. Function that validates if the user has provided an existing file.

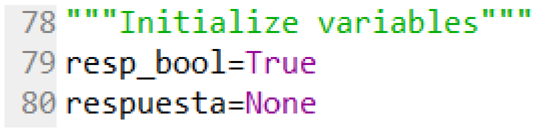
This function checks if the user's answer is valid by asking again until it is. Prevents the program from breaking.



```python
def resp_mala(answer):
    """Function that cycles the program until the user
    provides a valid answer"""
    ciclar = True
    while ciclar == True:
        if answer in ('y', 'Y', 'yes', 'Yes', 'YES'):
            return True
            ciclar = False
        elif answer in ('N', 'n', 'no', 'No', 'NO'):
            return False
            ciclar = False
        else:
            print("\nInvalid answer, try again")
            answer=input("\nDo you want to analyze another file? [Y/N]: ")
```

Image 11. Function that validates if the user’s answer is valid.

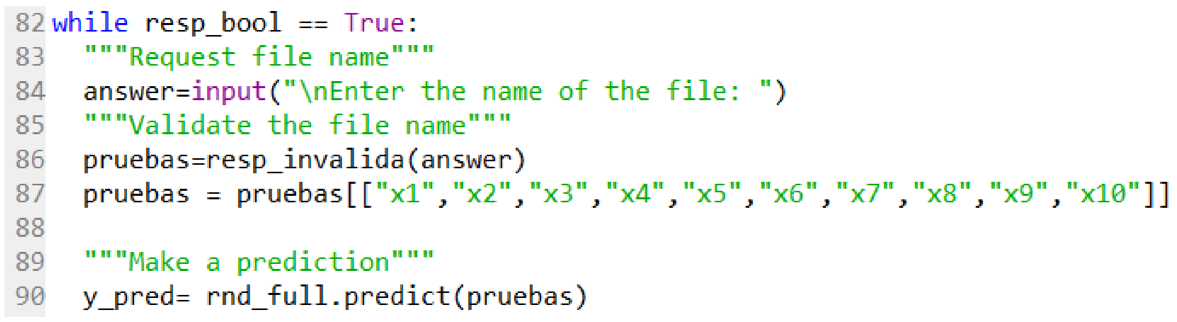
Now the variables used in the while cycle are initialized.



```python
"""Initialize variables"""
resp_bool=True
respuesta=None
```

Image 12. Variable initialization

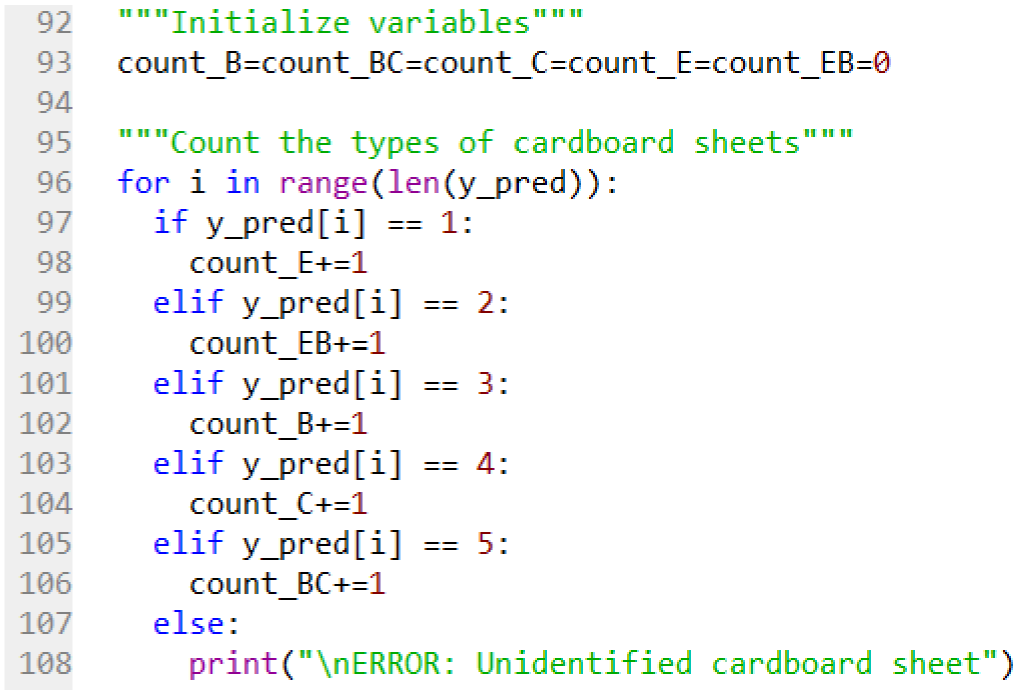
Image 13 shows the part of the code where the prediction is made. It requests the file name and provides this data to the “resp_invalida” function which return the data that is inside the file, then the data is used for making a prediction which is stores into an array.



```python
while resp_bool == True:
  """Request file name"""
  answer=input("\nEnter the name of the file: ")
  """Validate the file name"""
  pruebas=resp_invalida(answer)
  pruebas = pruebas[["x1","x2","x3","x4","x5","x6","x7","x8","x9","x10"]]

  """Make a prediction"""
  y_pred= rnd_full.predict(pruebas)
```

Image 13. Read the data and make a prediction.

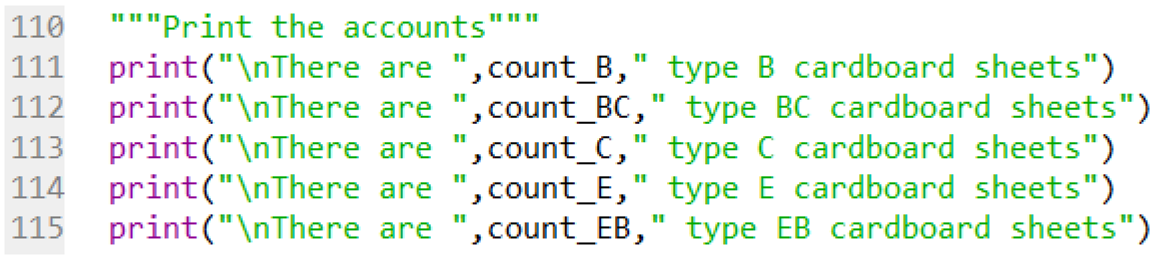
Once the prediction data is done, the number of sheets that were predicted is counted, for this, the array is traversed and the times that each type of sheet appears is counted.



```python
  """Initialize variables"""
  count_B=count_BC=count_C=count_E=count_EB=0

  """Count the types of cardboard sheets"""
  for i in range(len(y_pred)):
    if y_pred[i] == 1:
      count_E+=1
    elif y_pred[i] == 2:
      count_EB+=1
    elif y_pred[i] == 3:
      count_B+=1
    elif y_pred[i] == 4:
      count_C+=1
    elif y_pred[i] == 5:
      count_BC+=1
    else:
      print("\nERROR: Unidentified cardboard sheet")
```

Image 14. Count the cardboard sheets.

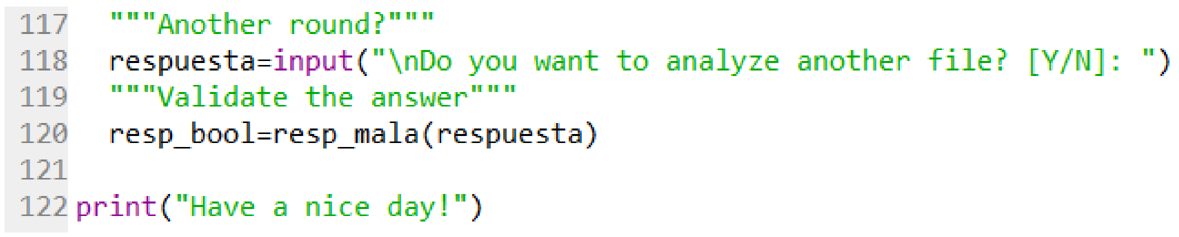
Then, the results of the previous count are printed.



```python
  """Print the accounts"""
  print("\nThere are ",count_B," type B cardboard sheets")
  print("\nThere are ",count_BC," type BC cardboard sheets")
  print("\nThere are ",count_C," type C cardboard sheets")
  print("\nThere are ",count_E," type E cardboard sheets")
  print("\nThere are ",count_EB," type EB cardboard sheets")
```

Image 15. Print the account.

Finally, the user is asked if he has another file with data to classify, its answer is evaluated in the “resp_mala” function. If the user types "yes" or something similar, the while loop is executed again, otherwise the program finish and prints a friendly message.

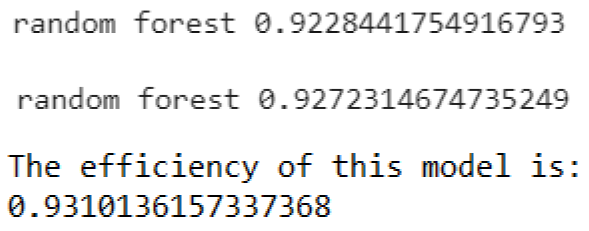


```
    """Another round?"""
    respuesta=input("\nDo you want to analyze another file? [Y/N]: ")
    """Validate the answer"""
    resp_bool=resp_mala(respuesta)

print("Have a nice day!")
```

Image 16. Continue or finish.

## 5. Results

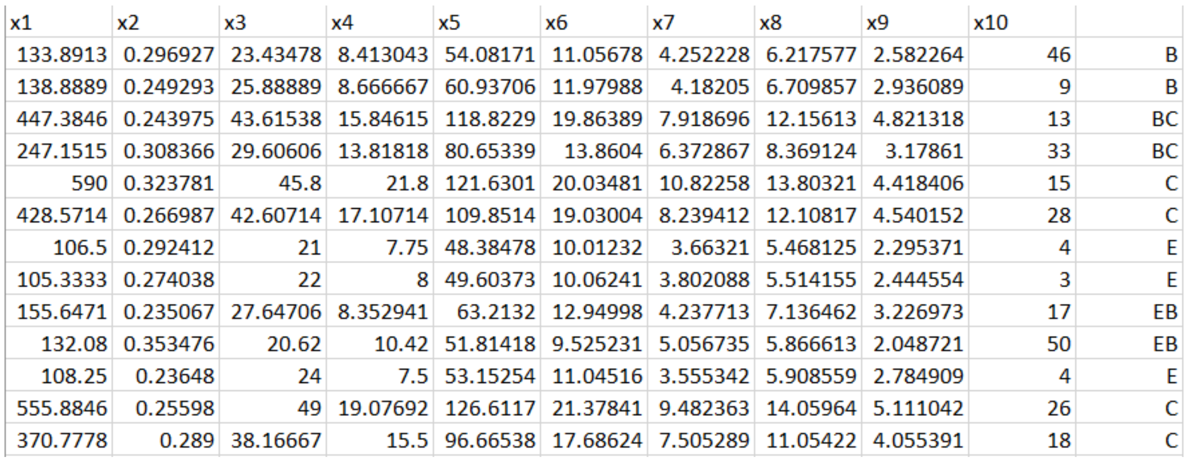
As mentioned above, the efficiency of the algorithm ranges from 92% to 93% accuracy depending on how the shuffle algorithm orders the data.



```
random forest 0.9228441754916793

random forest 0.9272314674735249

The efficiency of this model is:
0.9310136157337368
```

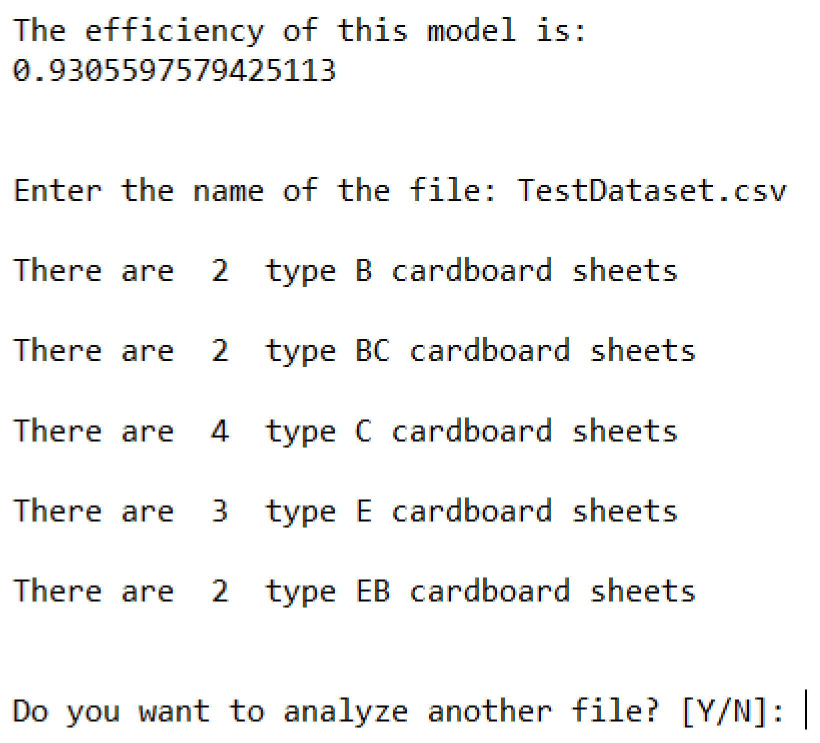
Image 17. Different accuracy scores obtained by the program.

However, the program makes successful predictions when entering a test file.



| x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 133.8913 | 0.296927 | 23.43478 | 8.413043 | 54.08171 | 11.05678 | 4.252228 | 6.217577 | 2.582264 | 46 | B |
| 138.8889 | 0.249293 | 25.88889 | 8.666667 | 60.93706 | 11.97988 | 4.18205 | 6.709857 | 2.936089 | 9 | B |
| 447.3846 | 0.243975 | 43.61538 | 15.84615 | 118.8229 | 19.86389 | 7.918696 | 12.15613 | 4.821318 | 13 | BC |
| 247.1515 | 0.308366 | 29.60606 | 13.81818 | 80.65339 | 13.8604 | 6.372867 | 8.369124 | 3.17861 | 33 | BC |
| 590 | 0.323781 | 45.8 | 21.8 | 121.6301 | 20.03481 | 10.82258 | 13.80321 | 4.418406 | 15 | C |
| 428.5714 | 0.266987 | 42.60714 | 17.10714 | 109.8514 | 19.03004 | 8.239412 | 12.10817 | 4.540152 | 28 | C |
| 106.5 | 0.292412 | 21 | 7.75 | 48.38478 | 10.01232 | 3.66321 | 5.468125 | 2.295371 | 4 | E |
| 105.3333 | 0.274038 | 22 | 8 | 49.60373 | 10.06241 | 3.802088 | 5.514155 | 2.444554 | 3 | E |
| 155.6471 | 0.235067 | 27.64706 | 8.352941 | 63.2132 | 12.94998 | 4.237713 | 7.136462 | 3.226973 | 17 | EB |
| 132.08 | 0.353476 | 20.62 | 10.42 | 51.81418 | 9.525231 | 5.056735 | 5.866613 | 2.048721 | 50 | EB |
| 108.25 | 0.23648 | 24 | 7.5 | 53.15254 | 11.04516 | 3.555342 | 5.908559 | 2.784909 | 4 | E |
| 555.8846 | 0.25598 | 49 | 19.07692 | 126.6117 | 21.37841 | 9.482363 | 14.05964 | 5.111042 | 26 | C |
| 370.7778 | 0.289 | 38.16667 | 15.5 | 96.66538 | 17.68624 | 7.505289 | 11.05422 | 4.055391 | 18 | C |

Image 18. Structure of the test file, the last column serves as a reference and is ignored by the program.



```
The efficiency of this model is:
0.9305597579425113


Enter the name of the file: TestDataset.csv

There are  2  type B cardboard sheets

There are  2  type BC cardboard sheets

There are  4  type C cardboard sheets

There are  3  type E cardboard sheets

There are  2  type EB cardboard sheets


Do you want to analyze another file? [Y/N]: 
```

Image 19. Program response when receiving the test file as input.

## 6. Conclusion

The Random Forest is a very powerful method to perform classification and regression tasks with a very good efficiency which works quite well when you have a limited dataset. Regarding the project, the MVTec HALCON algorithm that extracts the data must be improved since the numbers obtained are not easy to differentiate and improve the methods that have been used to extract the images of the flutes.

## 7. References

Cantu, E. (2019). *Cartón corrugado y microcorrugado,* from Franja Industrias | Etiquetas en rollo. Web Site: https://www.etiquetasenrollo.mx/2015/04/carton-corrugado-y-microcorrugado/

Donges, N. (2019). *A COMPLETE GUIDE TO THE RANDOM FOREST ALGORITHM*, from BuiltIn. Web Site: https://builtin.com/data-science/random-forest-algorithm

Gonzales, B. (2019). *Cartón corrugado: tipos y usos en el embalaje,* from Tu blog de Embalaje, Logística y más | RAJA®. Web Site: https://www.rajapack.es/blog-es/productos/carton-corrugado-tipos-usos-embalaje/

Scikit Learn. (s.f.). *sklearn.ensemble.RandomForestClassifier*. Web Site: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Universidad De Alcalá. (2019). *Scikit-Learn, herramienta básica para el Data Science en Python*, from Máster en Data Science. Web Site: https://www.master-data-scientist.com/scikit-learn-data-science/