UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Electronică,
Telecomunicații și

# Random Forest Algoritm

Cureu Denisa-Geanina

## Objective

In this project we will study 100 women, using the data listed below to try to understand whether a pregnant woman has a *low*, *medium* or *high risk* of mortality.

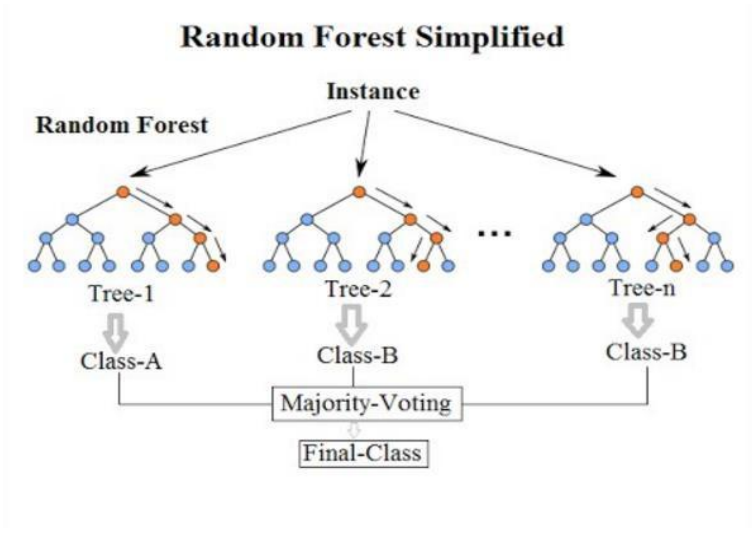
- **SCENARIO DATA**
- Age: ages in years.
- SystolicBP: upper blood pressure value in mmHg, a significant attribute during pregnancy.
- DiastolicBP: lower blood pressure value in mmHg, another significant attribute during pregnancy.
- BS: blood glucose in terms of concentration, mmol/L.
- HeartRate: heart rate in beats per minute.
- RiskLevel: the level of risk during pregnancy.
- BodyTemp: body temperature.

## Contained

## Theoretical foundation

Random forest is a machine learning technique that is used to solve regression and classification problems.

For classification tasks, the result of radom forest is the class selected by most trees. For regression tasks, the mean or mean prediction of the individual trees is returned.



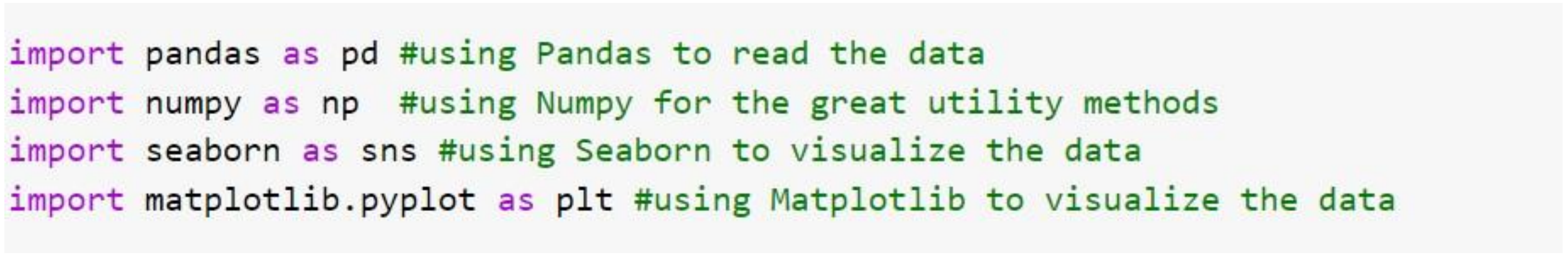
Random forest uses many trees and makes a prediction by averaging the predictions of each component tree. Overall, it has much better predictive accuracy than a single decision tree and works well with the parameters involved.

- **General characteristics**

- ✓ It is based on the vote of m random trees, where m is a Input parameter
- ✓ Generate m drive sets
- ✓ A random tree is trained for each set
- ✓ For a test instance, each tree gives a class membership vote
- ✓ The class with the most votes is the answer to Random Forest

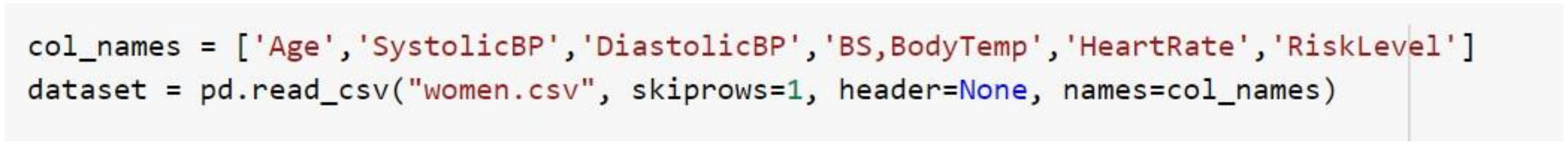
## Implementation of the adopted solution

❖ Importing Libraries:



```
import pandas as pd #using Pandas to read the data
import numpy as np  #using Numpy for the great utility methods
import seaborn as sns #using Seaborn to visualize the data
import matplotlib.pyplot as plt #using Matplotlib to visualize the data
```

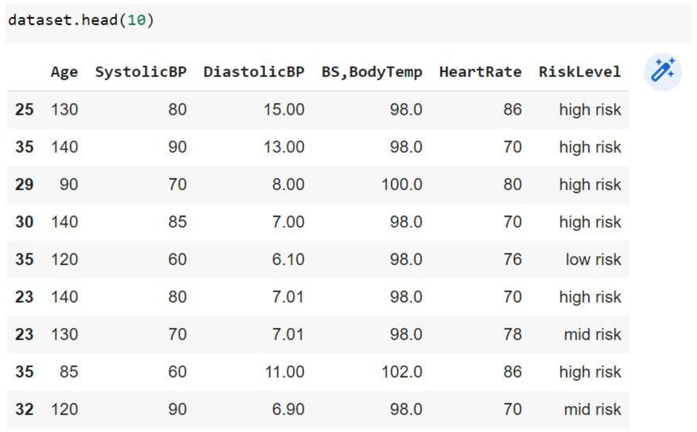
❖ Importing the database:



```
col_names = ['Age','SystolicBP','DiastolicBP','BS,BodyTemp','HeartRate','RiskLevel']
dataset = pd.read_csv("women.csv", skiprows=1, header=None, names=col_names)
```

❖ Data viewing:

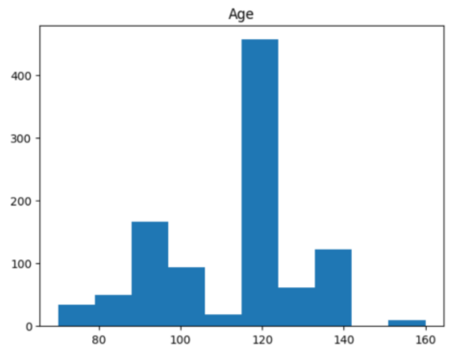
➢ Table:

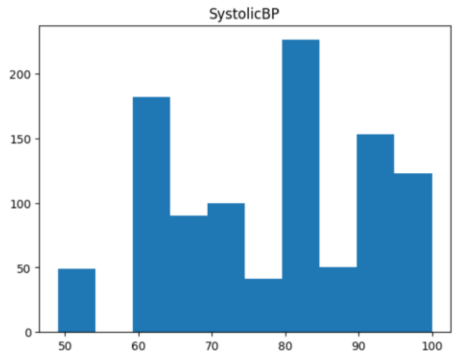


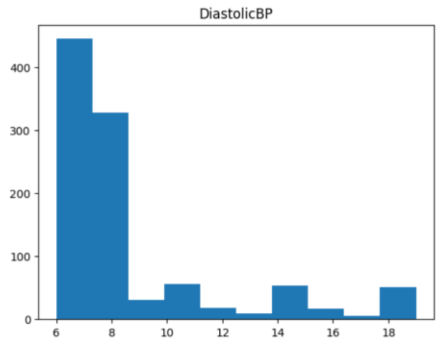
```
dataset.head(10)
```

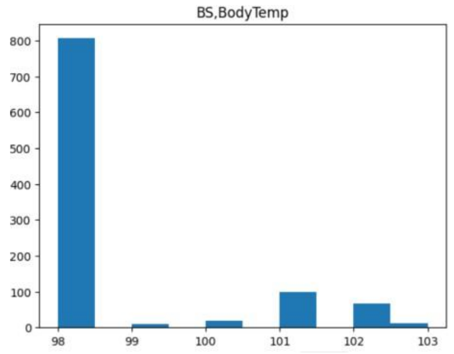
| | Age | SystolicBP | DiastolicBP | BS,BodyTemp | HeartRate | RiskLevel |
|---|---|---|---|---|---|---|
| **25** | 130 | 80 | 15.00 | 98.0 | 86 | high risk |
| **35** | 140 | 90 | 13.00 | 98.0 | 70 | high risk |
| **29** | 90 | 70 | 8.00 | 100.0 | 80 | high risk |
| **30** | 140 | 85 | 7.00 | 98.0 | 70 | high risk |
| **35** | 120 | 60 | 6.10 | 98.0 | 76 | low risk |
| **23** | 140 | 80 | 7.01 | 98.0 | 70 | high risk |
| **23** | 130 | 70 | 7.01 | 98.0 | 78 | mid risk |
| **35** | 85 | 60 | 11.00 | 102.0 | 86 | high risk |
| **32** | 120 | 90 | 6.90 | 98.0 | 70 | mid risk |

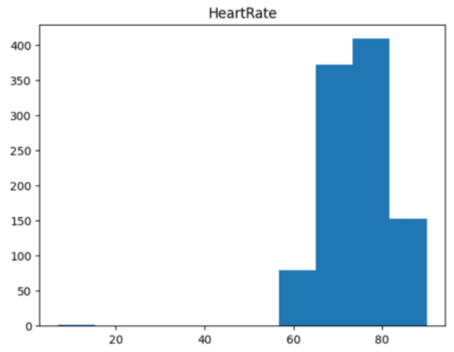
➢ Histograme:

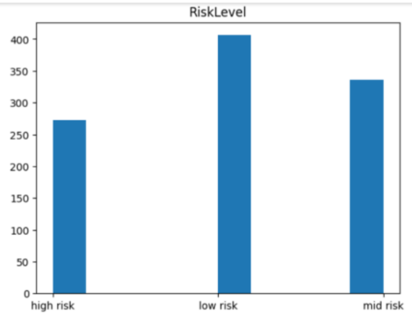




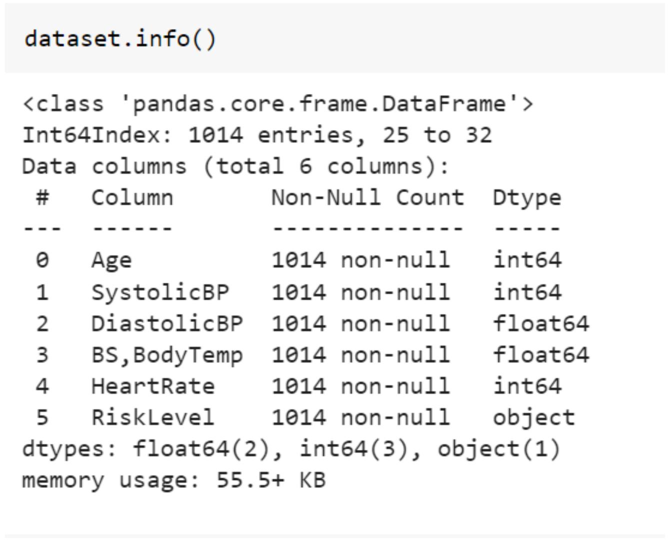








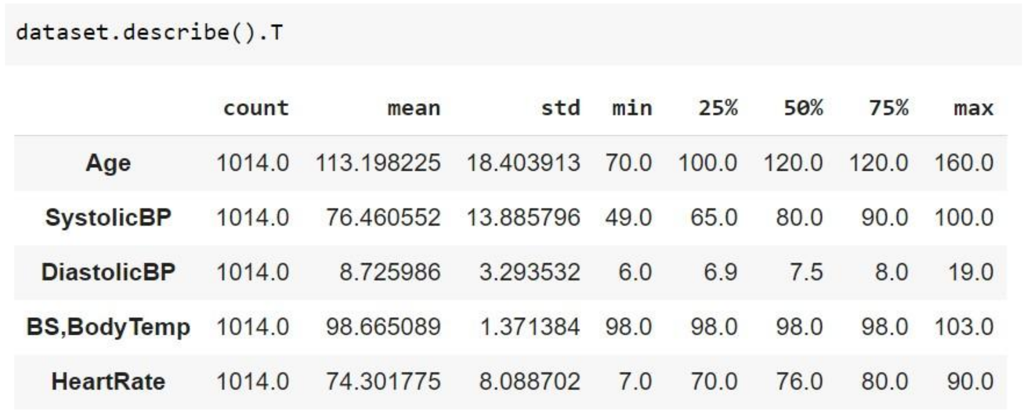
❖ Data types:



```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1014 entries, 25 to 32
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          1014 non-null   int64  
 1   SystolicBP   1014 non-null   int64  
 2   DiastolicBP  1014 non-null   float64
 3   BS,BodyTemp  1014 non-null   float64
 4   HeartRate    1014 non-null   int64  
 5   RiskLevel    1014 non-null   object 
dtypes: float64(2), int64(3), object(1)
memory usage: 55.5+ KB
```

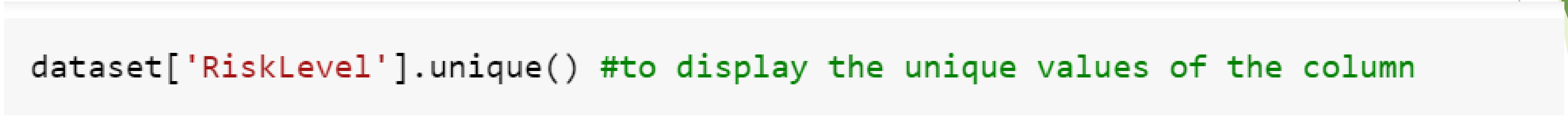
❖ Descriptive statistics: average values of each column, standard deviation, values minimal/maxim etc.



```
dataset.describe().T
```

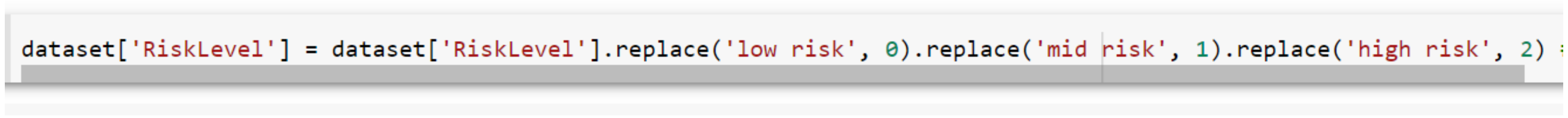
| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Age** | 1014.0 | 113.198225 | 18.403913 | 70.0 | 100.0 | 120.0 | 120.0 | 160.0 |
| **SystolicBP** | 1014.0 | 76.460552 | 13.885796 | 49.0 | 65.0 | 80.0 | 90.0 | 100.0 |
| **DiastolicBP** | 1014.0 | 8.725986 | 3.293532 | 6.0 | 6.9 | 7.5 | 8.0 | 19.0 |
| **BS,BodyTemp** | 1014.0 | 98.665089 | 1.371384 | 98.0 | 98.0 | 98.0 | 98.0 | 103.0 |
| **HeartRate** | 1014.0 | 74.301775 | 8.088702 | 7.0 | 70.0 | 76.0 | 80.0 | 90.0 |

❖ Ensuring that there are only 3 unique classes:



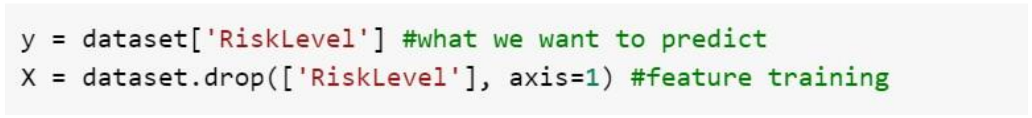
```
dataset['RiskLevel'].unique() #to display the unique values of the column
```

❖ Labeling each category with a digit:



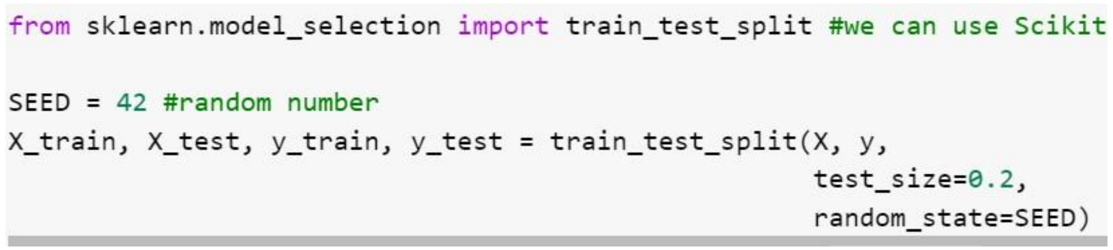
```
dataset['RiskLevel'] = dataset['RiskLevel'].replace('low risk', 0).replace('mid risk', 1).replace('high risk', 2)
```

❖ Splitting data into features/X and labels/Y:



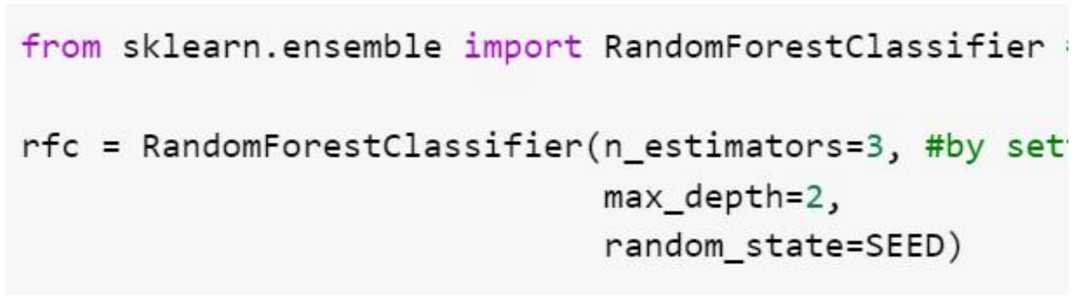
```
y = dataset['RiskLevel'] #what we want to predict
X = dataset.drop(['RiskLevel'], axis=1) #feature training
```

- ❖ Using the Scikit-Learn method to divide data into test sets (20%) and training sets (80%) :



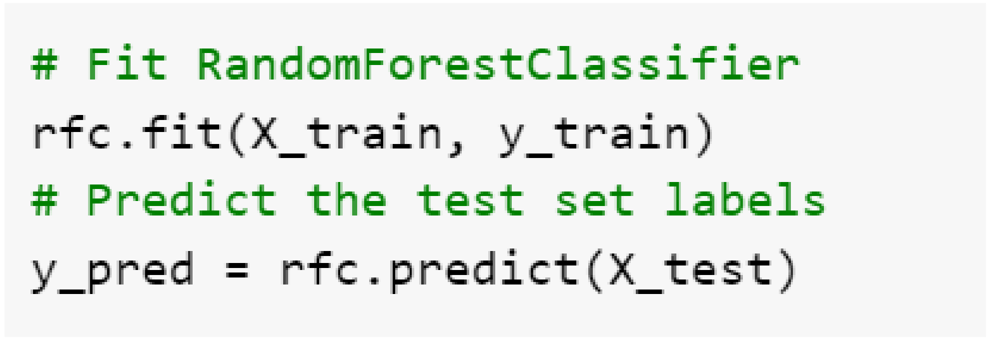
```
from sklearn.model_selection import train_test_split #we can use Scikit

SEED = 42 #random number
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=SEED)
```

- ❖ Creating the model: 3 trees and 3 levels



```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=3, #by set
                             max_depth=2,
                             random_state=SEED)
```

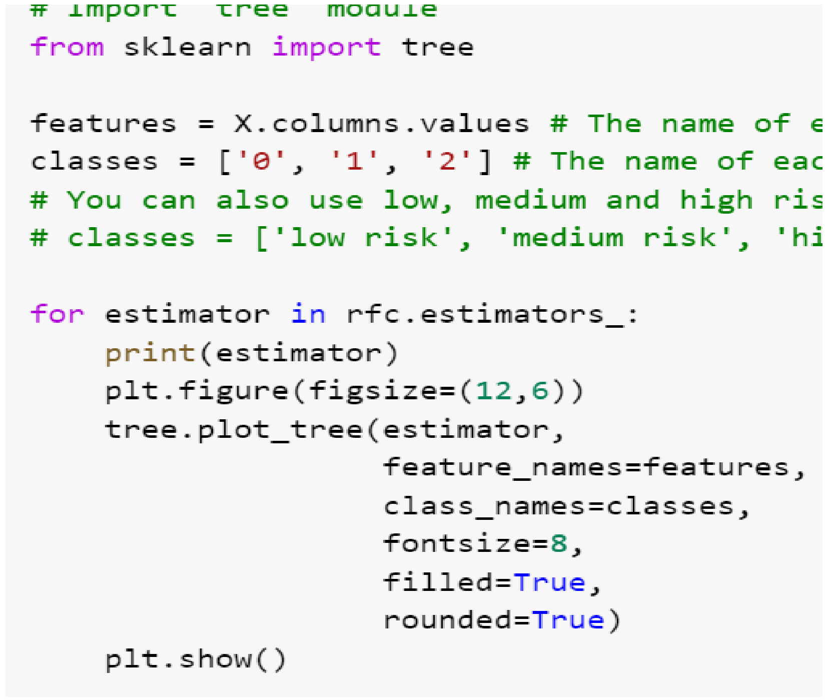
❖ To fit the pattern around the data - we call the fit() method, Moving on to the training features and tags:



```
# Fit RandomForestClassifier
rfc.fit(X_train, y_train)
# Predict the test set labels
y_pred = rfc.predict(X_test)
```

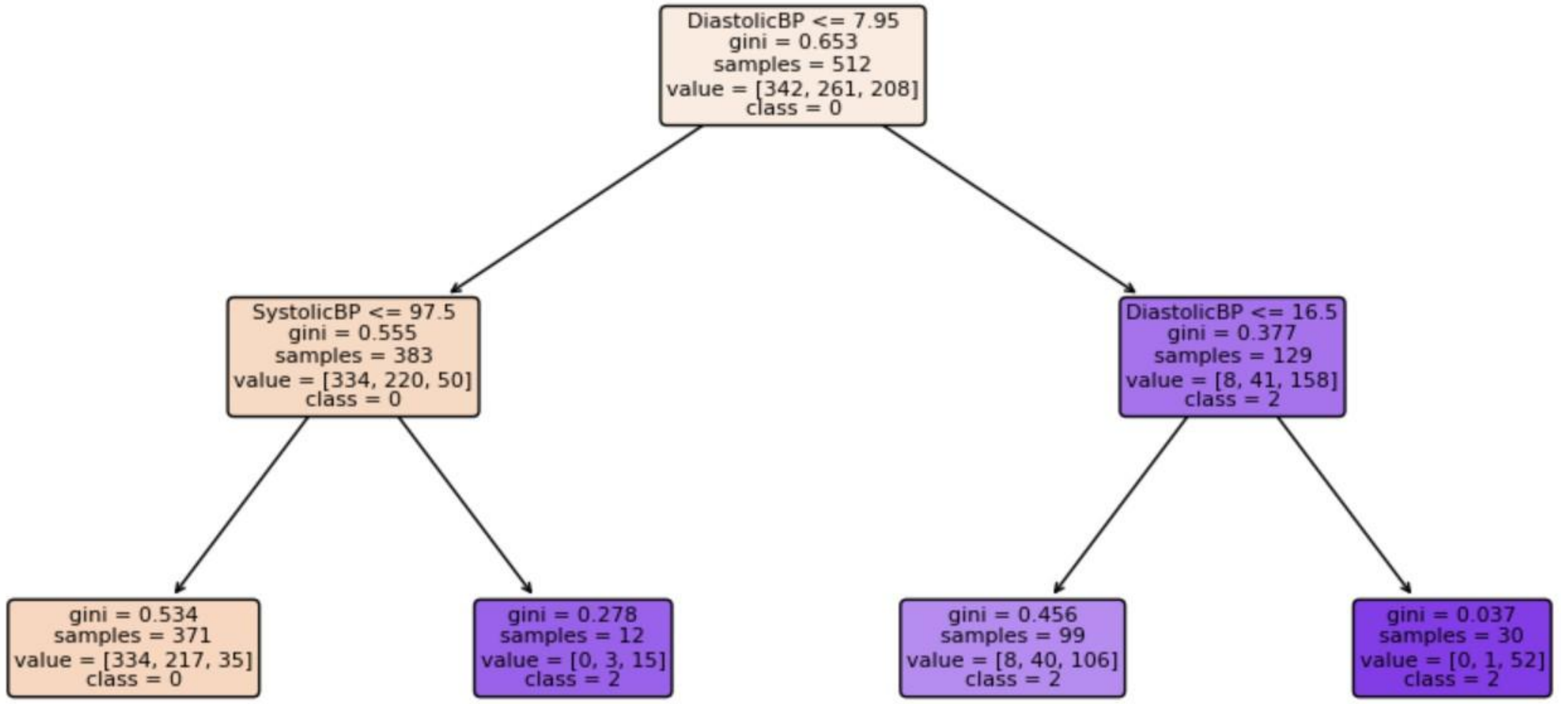
## Experimental results

❖ Comparison of predicted labels with real labels:

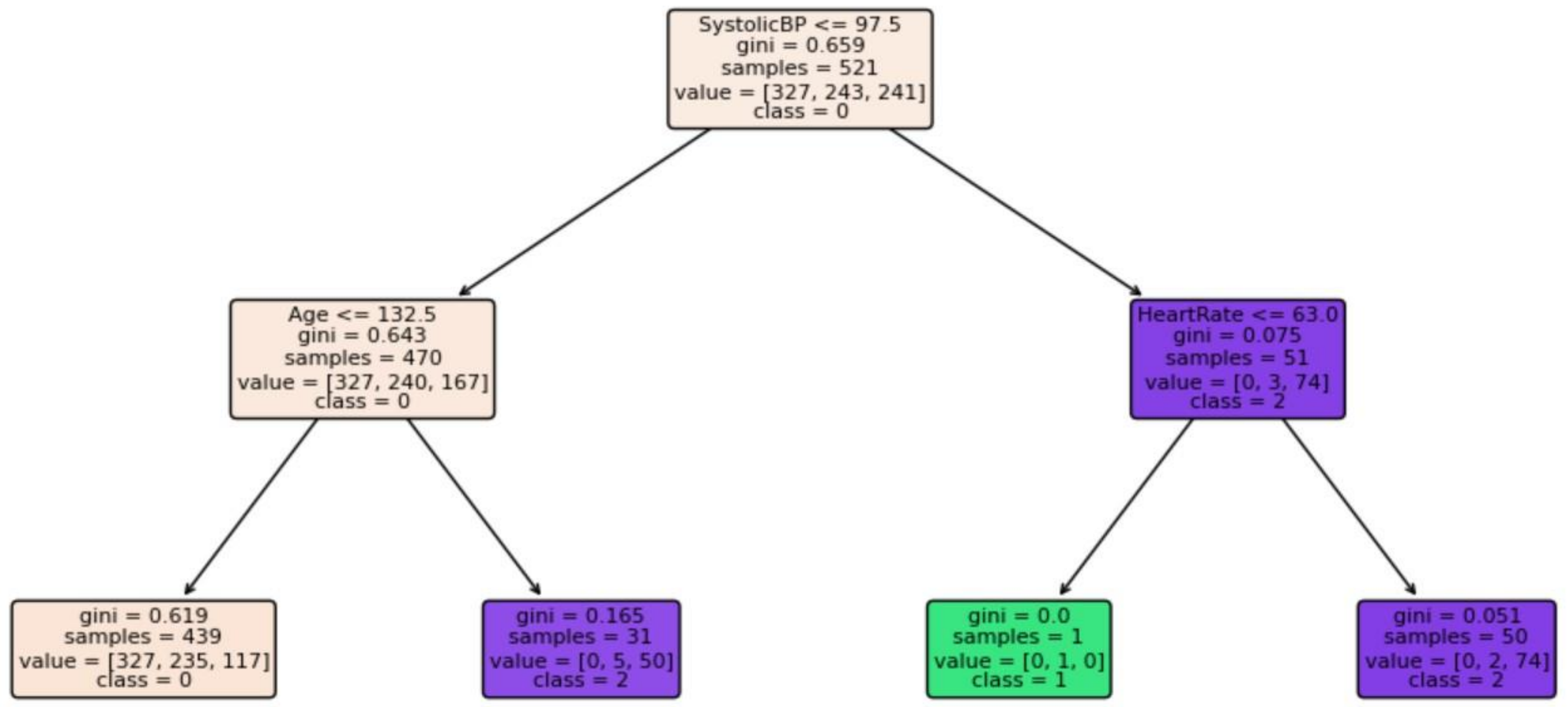


```
# import  tree  module
from sklearn import tree

features = X.columns.values # The name of e
classes = ['0', '1', '2'] # The name of eac
# You can also use low, medium and high ris
# classes = ['low risk', 'medium risk', 'hi

for estimator in rfc.estimators_:
    print(estimator)
    plt.figure(figsize=(12,6))
    tree.plot_tree(estimator,
                   feature_names=features,
                   class_names=classes,
                   fontsize=8,
                   filled=True,
                   rounded=True)
    plt.show()
```

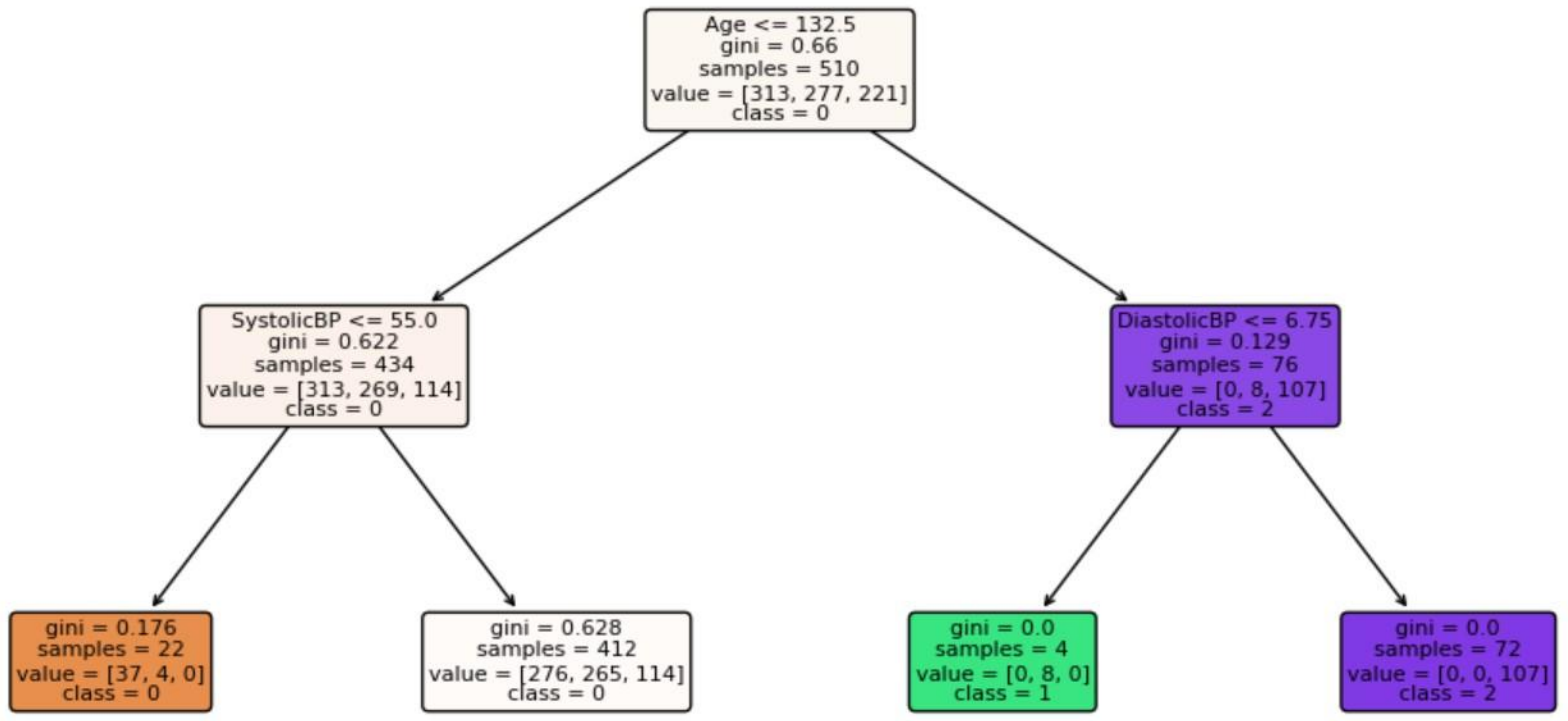
❖ Tree 1


DiastolicBP <= 7.95
gini = 0.653
samples = 512
value = [342, 261, 208]
class = 0
SystolicBP <= 97.5
gini = 0.555
samples = 383
value = [334, 220, 50]
class = 0
DiastolicBP <= 16.5
gini = 0.377
samples = 129
value = [8, 41, 158]
class = 2
gini = 0.534
samples = 371
value = [334, 217, 35]
class = 0
gini = 0.278
samples = 12
value = [0, 3, 15]
class = 2
gini = 0.456
samples = 99
value = [8, 40, 106]
class = 2
gini = 0.037
samples = 30
value = [0, 1, 52]
class = 2

## ❖ Tree 2


SystolicBP <= 97.5
gini = 0.659
samples = 521
value = [327, 243, 241]
class = 0
Age <= 132.5
gini = 0.643
samples = 470
value = [327, 240, 167]
class = 0
HeartRate <= 63.0
gini = 0.075
samples = 51
value = [0, 3, 74]
class = 2
gini = 0.619
samples = 439
value = [327, 235, 117]
class = 0
gini = 0.165
samples = 31
value = [0, 5, 50]
class = 2
gini = 0.0
samples = 1
value = [0, 1, 0]
class = 1
gini = 0.051
samples = 50
value = [0, 2, 74]
class = 2

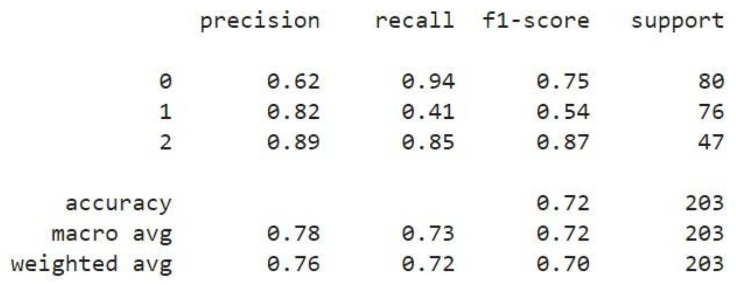
❖ Tree 3


Age <= 132.5
gini = 0.66
samples = 510
value = [313, 277, 221]
class = 0
SystolicBP <= 55.0
gini = 0.622
samples = 434
value = [313, 269, 114]
class = 0
DiastolicBP <= 6.75
gini = 0.129
samples = 76
value = [0, 8, 107]
class = 2
gini = 0.176
samples = 22
value = [37, 4, 0]
class = 0
gini = 0.628
samples = 412
value = [276, 265, 114]
class = 0
gini = 0.0
samples = 4
value = [0, 8, 0]
class = 1
gini = 0.0
samples = 72
value = [0, 0, 107]
class = 2

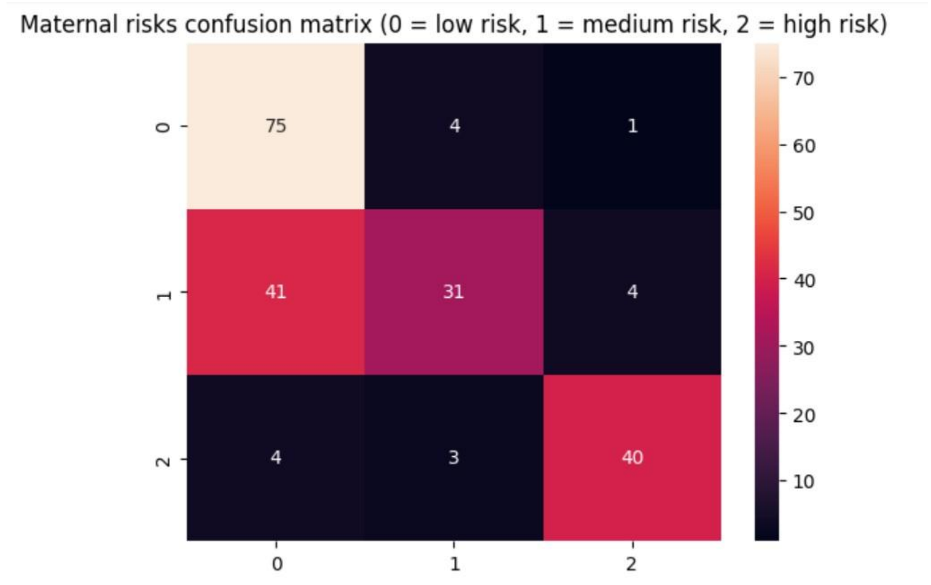
## ❖ General characteristics of trees

- **Accuracy -** when the goal is to understand which correct prediction values were considered correct by the classifier.
- **Recall –** typically calculated along with accuracy to understand how many of the true positives have been identified by the classifier.
- **F1 score -** is the balanced or *harmonic* means of accuracy and recalculation. The smallest value is 0, and the highest is 1.
- **Confusion matrix -** when we need to know how much evidence we have obtained well, or bad for *every class.*
- **Accuracy -** describes how many predictions were correct made by the classifier.

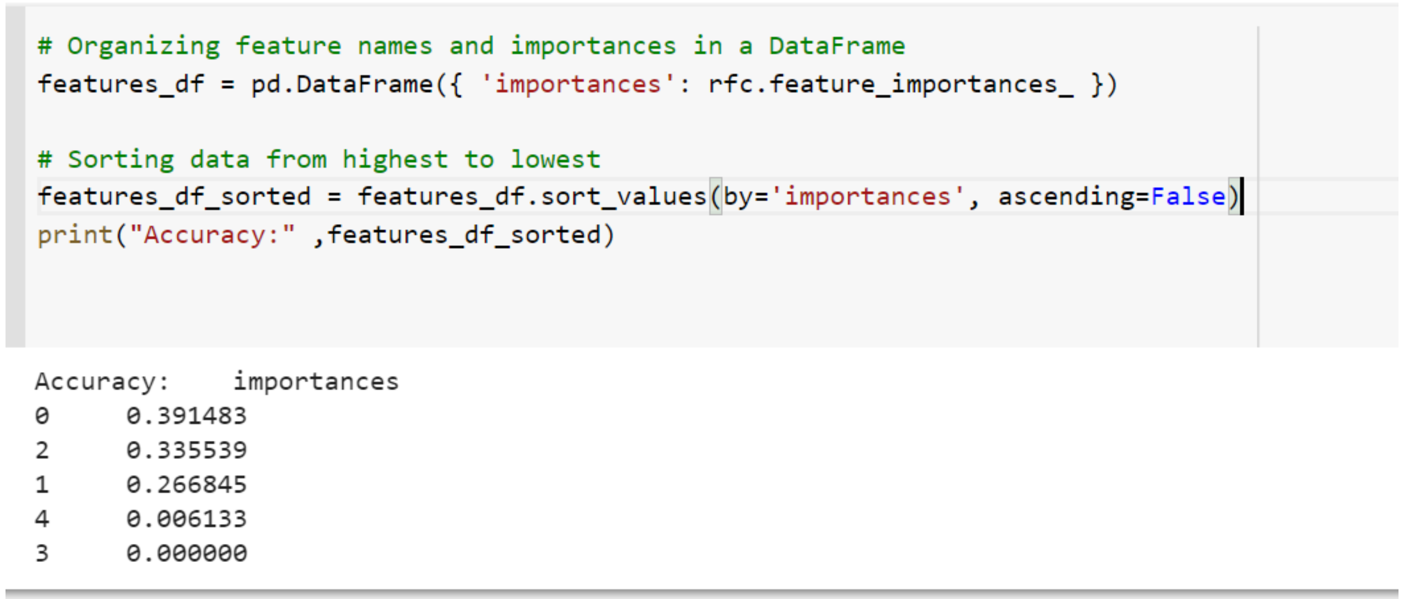
- Relationship between recall and accuracy for all three classes in individually is captured in the score *F1*, which is the harmonic mean between recall and accuracy - the model does *well* for class 0, quite bad for class 1 and decent for class 2.
- Looking at the confusion matrix, we can see that most mistakes are when we classify 52 records of medium risk as low risk, which provides additional insight into low class 1 recalls.



```
              precision    recall  f1-score   support

           0       0.62      0.94      0.75        80
           1       0.82      0.41      0.54        76
           2       0.89      0.85      0.87        47

    accuracy                           0.72       203
   macro avg       0.78      0.73      0.72       203
weighted avg       0.76      0.72      0.70       203
```



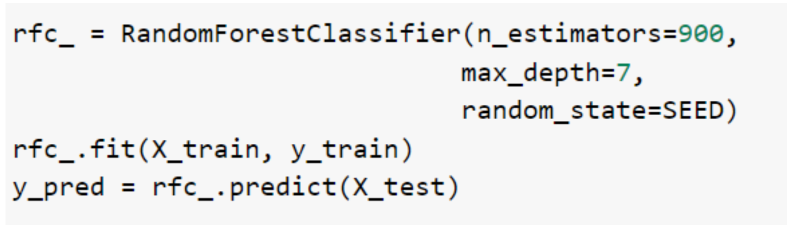
The classifier mostly takes *into account blood glucose*, then a little bit of diastolic pressure, body temperature and just a little age to make a decision, this could also be related to Class 1 low recall, perhaps the medium-risk data relates to characteristics that are largely not considered by the model.



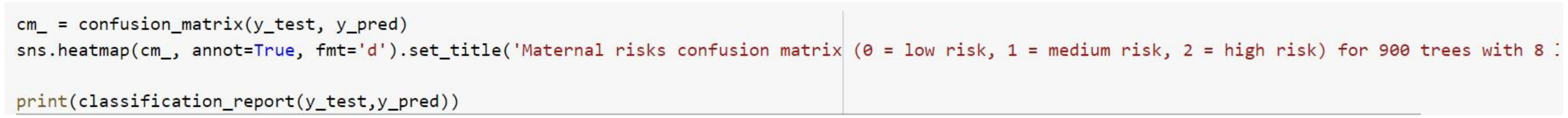
```
# Organizing feature names and importances in a DataFrame
features_df = pd.DataFrame({ 'importances': rfc.feature_importances_ })

# Sorting data from highest to lowest
features_df_sorted = features_df.sort_values(by='importances', ascending=False)
print("Accuracy:" ,features_df_sorted)
```

```
Accuracy:     importances
0      0.391483
2      0.335539
1      0.266845
4      0.006133
3      0.000000
```

## ❖ Creating the model: 900 trees and 8 levels

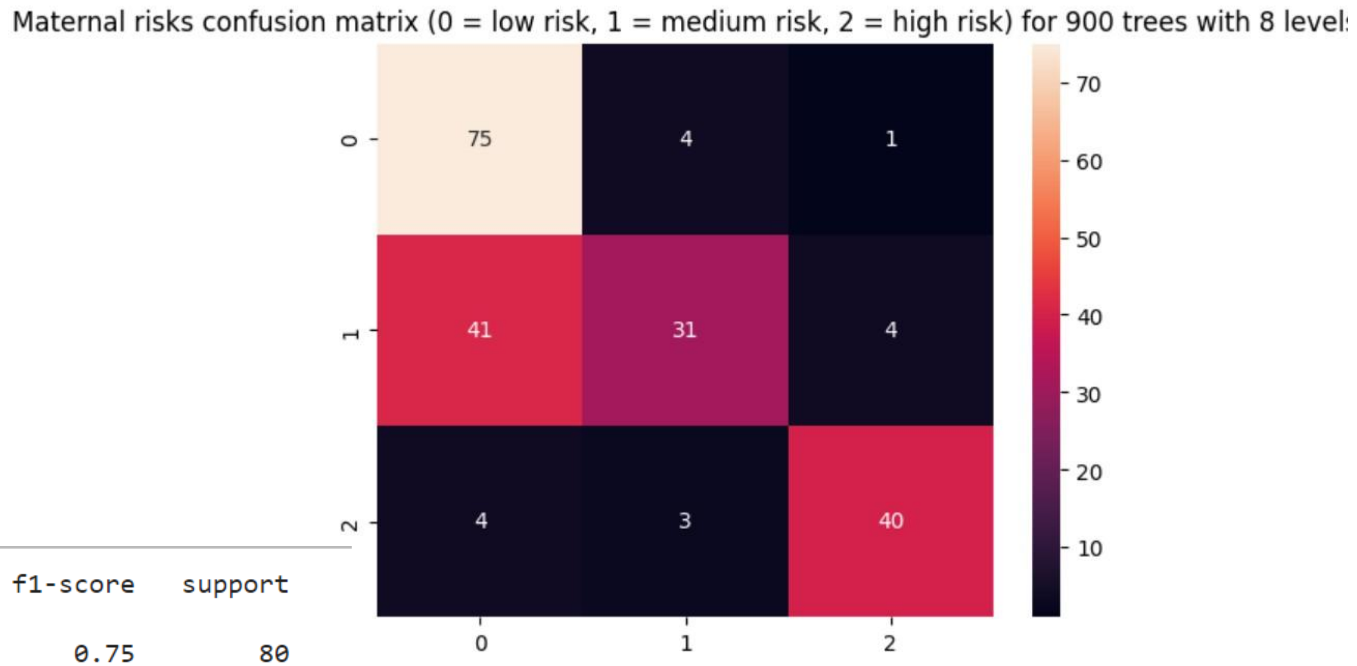


```python
rfc_ = RandomForestClassifier(n_estimators=900,
                              max_depth=7,
                              random_state=SEED)
rfc_.fit(X_train, y_train)
y_pred = rfc_.predict(X_test)
```



```python
cm_ = confusion_matrix(y_test, y_pred)
sns.heatmap(cm_, annot=True, fmt='d').set_title('Maternal risks confusion matrix (0 = low risk, 1 = medium risk, 2 = high risk) for 900 trees with 8 

print(classification_report(y_test,y_pred))
```

❖ The fact that we have increased the number of trees and that we have increased the tree levels underlines a noticeable improvement in the measurements.



```
              precision    recall  f1-score   support

           0       0.62      0.94      0.75        80
           1       0.82      0.41      0.54        76
           2       0.89      0.85      0.87        47

    accuracy                           0.72       203
   macro avg       0.78      0.73      0.72       203
weighted avg       0.76      0.72      0.70       203
```

## Conclusions

The Randam Forest is a classification algorithm consisting of many decision trees. It uses packaging and randomness when constructing each individual tree to try to create a collection of trees whose prediction is more accurate than that of any individual tree.

The advantage of this algorithm is that it is very stable. Even if a new data feature is introduced into the dataset, the overall algorithm is not affected too much because the new data may have a impact on a tree, but it is very difficult for it to have a impact on all trees.

A disadvantage of this algorithm would be its complexity.

## Bibliography

- https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/
- https://en.wikipedia.org/wiki/Random_forest
- https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/
- https://www.kaggle.com/code/faressayah/decision-trees-random-forest-for-beginners
- https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/

Facultatea de Electronică, Telecomunicații și

Thank you!