# Development project in Machine Learning

***Students:***

- **ABDALLAOUI Nada**
- **KERDAD Fatima-Zahra**
- **LEGUET Emmanuel**
- **ZRIAA Imane**

***To the Attention of :***

- **Mrs DUPRAZ Elsa**

**IMT Atlantique**
Bretagne-Pays de la Loire
École Mines-Télécom

# CONTENT:

## I – Introduction:

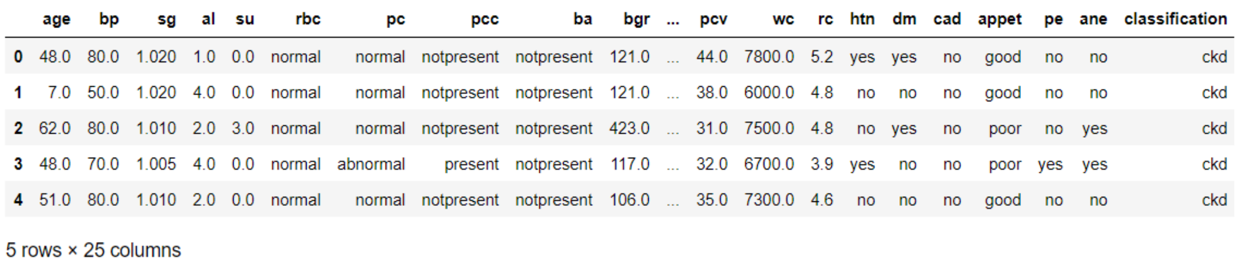
This project is about applying a machine learning model for a supervised classification onto two different datasets. The idea is to implement a unique Machine Learning workflow and apply it to : Kidney Disease and BankNote Authentication data sets.
In the following we will explain in detail the implementation of an automated pipeline that allows the extraction of the data to transform it into a usable form so that we can train our models and use them for new predictions.

## II – Datasets:

To implement our machine learning model, two datasets were provided:

### Chronic Kidney disease:

This data was taken over a 2-month period in India with 25 features. The target is the ‘classification’, which is either ‘ckd’ or ‘notckd’ with ‘ ckd’ stands for  chronic kidney disease.
The model we implemented will enable us to predict if a patient is suffering from a chronic kidney disease or not.



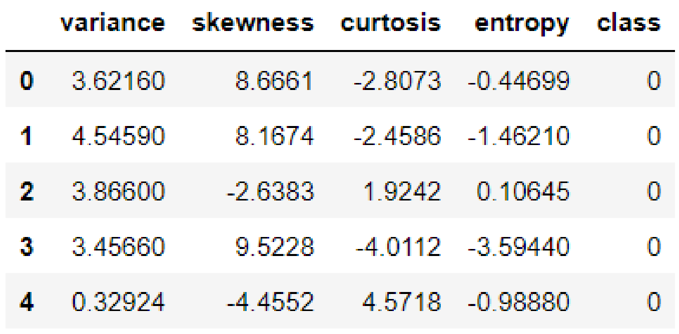
| | age | bp | sg | al | su | rbc | pc | pcc | ba | bgr | ... | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 48.0 | 80.0 | 1.020 | 1.0 | 0.0 | normal | normal | notpresent | notpresent | 121.0 | ... | 44.0 | 7800.0 | 5.2 | yes | yes | no | good | no | no | ckd |
| **1** | 7.0 | 50.0 | 1.020 | 4.0 | 0.0 | normal | normal | notpresent | notpresent | 121.0 | ... | 38.0 | 6000.0 | 4.8 | no | no | no | good | no | no | ckd |
| **2** | 62.0 | 80.0 | 1.010 | 2.0 | 3.0 | normal | normal | notpresent | notpresent | 423.0 | ... | 31.0 | 7500.0 | 4.8 | no | yes | no | poor | no | yes | ckd |
| **3** | 48.0 | 70.0 | 1.005 | 4.0 | 0.0 | normal | abnormal | present | notpresent | 117.0 | ... | 32.0 | 6700.0 | 3.9 | yes | no | no | poor | yes | yes | ckd |
| **4** | 51.0 | 80.0 | 1.010 | 2.0 | 0.0 | normal | normal | notpresent | notpresent | 106.0 | ... | 35.0 | 7300.0 | 4.6 | no | no | no | good | no | no | ckd |

5 rows × 25 columns

*Figure 1: A sample of 5 rows of chronic kidney disease dataset*

### Bank Note Authentication:

Data were extracted from images that were taken from genuine and forged banknotes with 4 features. The dataset has 1372 samples and the target is the ‘class’ which can be either 0 or 1.

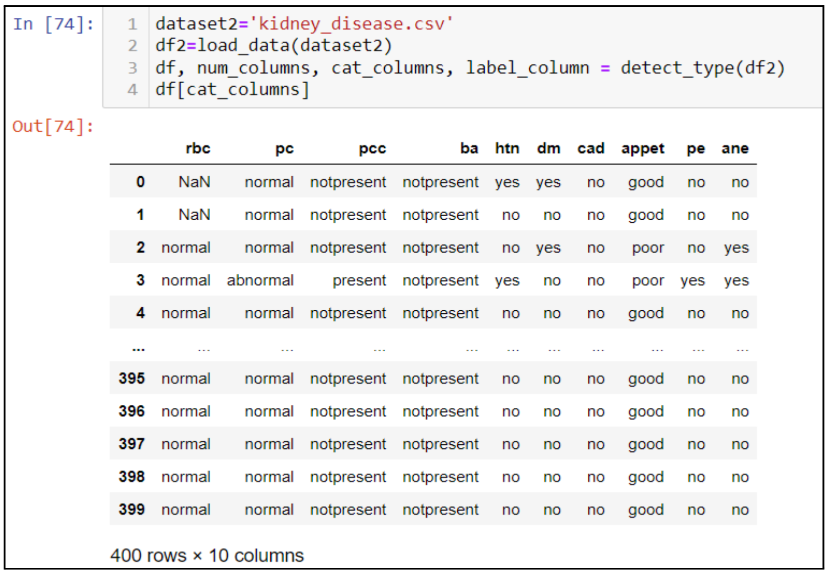


| | variance | skewness | curtosis | entropy | class |
|---|---|---|---|---|---|
| **0** | 3.62160 | 8.6661 | -2.8073 | -0.44699 | 0 |
| **1** | 4.54590 | 8.1674 | -2.4586 | -1.46210 | 0 |
| **2** | 3.86600 | -2.6383 | 1.9242 | 0.10645 | 0 |
| **3** | 3.45660 | 9.5228 | -4.0112 | -3.59440 | 0 |
| **4** | 0.32924 | -4.4552 | 4.5718 | -0.98880 | 0 |

*Figure 2: A sample of 5 rows of Bank Note Authentication dataset*

## III – Cleaning data & Pre-processing:

To clean our data, we should identify and correct the errors in the dataset that may negatively impact our predictive model.
First, we should detect the type of our columns and split it into categorical and numerical columns.
We used the function **detect_type** that takes as input the dataframe and returns the categorical, the numerical and the label columns.
For example, for the kidney disease dataset, the split of our columns will be as follow:



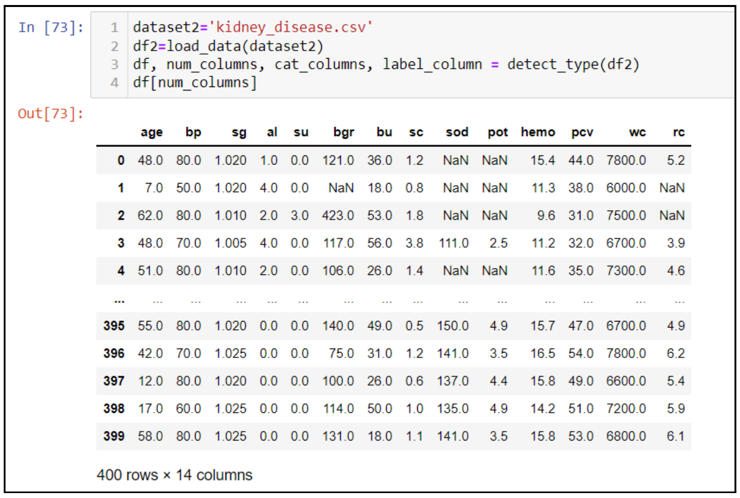
```
In [74]:
dataset2='kidney_disease.csv'
df2=load_data(dataset2)
df, num_columns, cat_columns, label_column = detect_type(df2)
df[cat_columns]
```

Out[74]:

| | rbc | pc | pcc | ba | htn | dm | cad | appet | pe | ane |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | normal | notpresent | notpresent | yes | yes | no | good | no | no |
| **1** | NaN | normal | notpresent | notpresent | no | no | no | good | no | no |
| **2** | normal | normal | notpresent | notpresent | no | yes | no | poor | no | yes |
| **3** | normal | abnormal | present | notpresent | yes | no | no | poor | yes | yes |
| **4** | normal | normal | notpresent | notpresent | no | no | no | good | no | no |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **395** | normal | normal | notpresent | notpresent | no | no | no | good | no | no |
| **396** | normal | normal | notpresent | notpresent | no | no | no | good | no | no |
| **397** | normal | normal | notpresent | notpresent | no | no | no | good | no | no |
| **398** | normal | normal | notpresent | notpresent | no | no | no | good | no | no |
| **399** | normal | normal | notpresent | notpresent | no | no | no | good | no | no |

400 rows × 10 columns

*Figure 3: Categorical columns of the kidney chronic disease dataset*



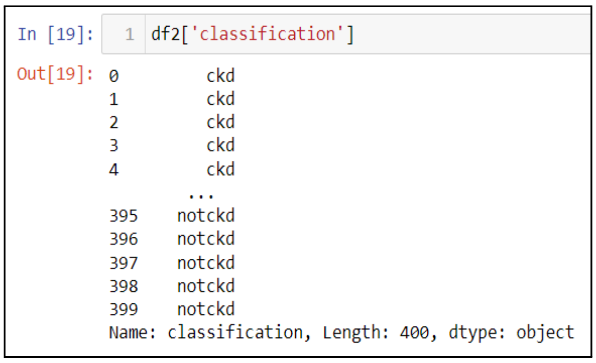
```
In [73]: dataset2='kidney_disease.csv'
         df2=load_data(dataset2)
         df, num_columns, cat_columns, label_column = detect_type(df2)
         df[num_columns]
Out[73]:
```

| | age | bp | sg | al | su | bgr | bu | sc | sod | pot | hemo | pcv | wc | rc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 48.0 | 80.0 | 1.020 | 1.0 | 0.0 | 121.0 | 36.0 | 1.2 | NaN | NaN | 15.4 | 44.0 | 7800.0 | 5.2 |
| **1** | 7.0 | 50.0 | 1.020 | 4.0 | 0.0 | NaN | 18.0 | 0.8 | NaN | NaN | 11.3 | 38.0 | 6000.0 | NaN |
| **2** | 62.0 | 80.0 | 1.010 | 2.0 | 3.0 | 423.0 | 53.0 | 1.8 | NaN | NaN | 9.6 | 31.0 | 7500.0 | NaN |
| **3** | 48.0 | 70.0 | 1.005 | 4.0 | 0.0 | 117.0 | 56.0 | 3.8 | 111.0 | 2.5 | 11.2 | 32.0 | 6700.0 | 3.9 |
| **4** | 51.0 | 80.0 | 1.010 | 2.0 | 0.0 | 106.0 | 26.0 | 1.4 | NaN | NaN | 11.6 | 35.0 | 7300.0 | 4.6 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **395** | 55.0 | 80.0 | 1.020 | 0.0 | 0.0 | 140.0 | 49.0 | 0.5 | 150.0 | 4.9 | 15.7 | 47.0 | 6700.0 | 4.9 |
| **396** | 42.0 | 70.0 | 1.025 | 0.0 | 0.0 | 75.0 | 31.0 | 1.2 | 141.0 | 3.5 | 16.5 | 54.0 | 7800.0 | 6.2 |
| **397** | 12.0 | 80.0 | 1.020 | 0.0 | 0.0 | 100.0 | 26.0 | 0.6 | 137.0 | 4.4 | 15.8 | 49.0 | 6600.0 | 5.4 |
| **398** | 17.0 | 60.0 | 1.025 | 0.0 | 0.0 | 114.0 | 50.0 | 1.0 | 135.0 | 4.9 | 14.2 | 51.0 | 7200.0 | 5.9 |
| **399** | 58.0 | 80.0 | 1.025 | 0.0 | 0.0 | 131.0 | 18.0 | 1.1 | 141.0 | 3.5 | 15.8 | 53.0 | 6800.0 | 6.1 |

400 rows × 14 columns

***Figure 4: Numerical columns of the kidney chronic disease dataset***

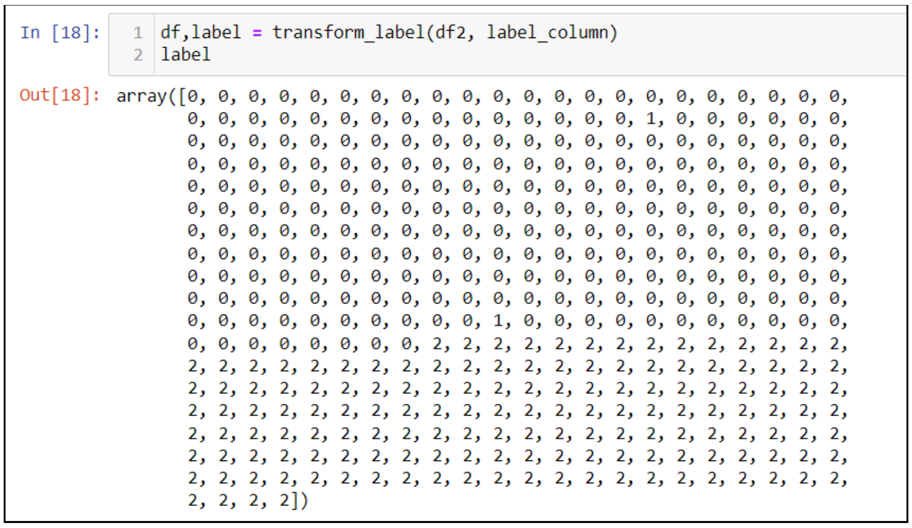
After that, we should identify and correct the errors in the dataset that may negatively impact our predictive model.
At first, we did label encoding to convert the labels into a numeric form.



```
In [19]: df2['classification']

Out[19]: 0         ckd
         1         ckd
         2         ckd
         3         ckd
         4         ckd
                 ...
         395    notckd
         396    notckd
         397    notckd
         398    notckd
         399    notckd
         Name: classification, Length: 400, dtype: object
```

***Figure 5: Label column before encoding for kidney chronic dataset***



```
In [18]: df,label = transform_label(df2, label_column)
         label

Out[18]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2])
```

***Figure 6: Label column after encoding for kidney chronic dataset***

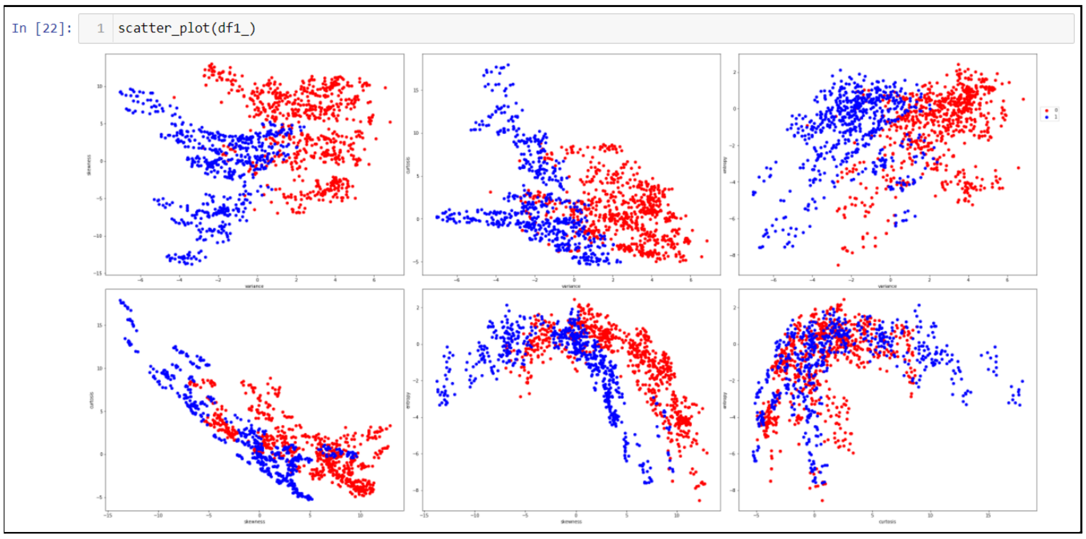
We fill the missing values to make our dataset usable. For the numerical columns, we replace the missing values with the mean of all the values that we have, with the most frequent category for the categorical columns.
We used the function **fill_na** that takes as input the data frame, the categorical columns and the numerical columns and returns the data frame with no missing values.

To make sure that the data is internally consistent, we normalize our features by using the Standardization method.
We use the function **transform_data** that takes as input the dataframe, the categorical columns, and the numerical columns. The function encodes the categorical columns and normalizes the numerical ones.

## IV – Data Visualization

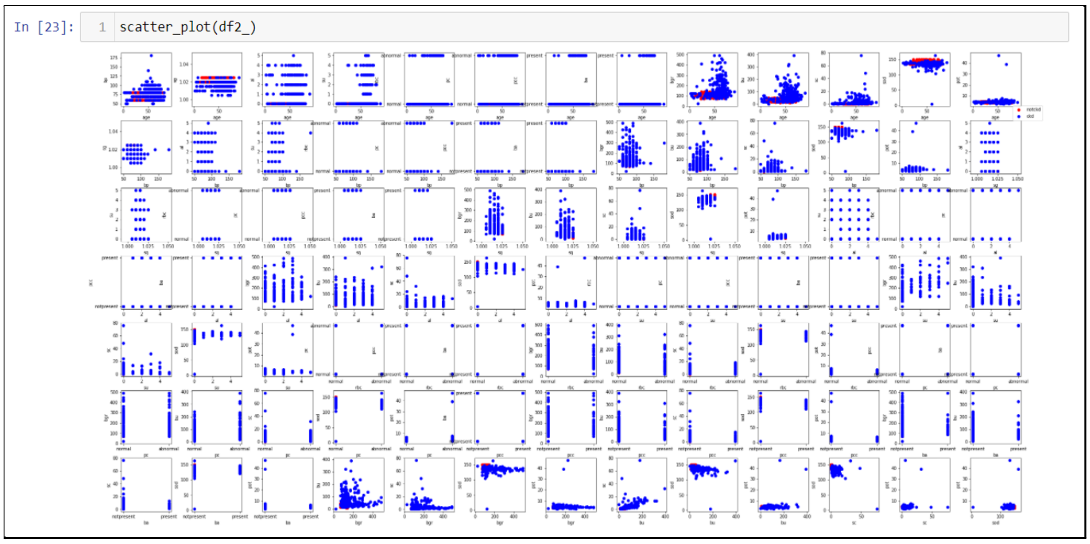
To get a general understanding of the kind of data we are manipulating, it may be wise to visualize it in order to detect the possible relationships that may exist between our variables.

For this reason, we added the function **scatter_plot** that takes as argument the dataframe and returns a simple scatter of 2 different features one plotted along the x-axis and the other plotted along the y-axis.



```
In [22]: scatter_plot(df1_)
```

*Figure 7: scatter plot of bank note authentication dataset*



```
In [23]:  scatter_plot(df2_)
```

*Figure 8: scatter plot of chronic kidney dataset*

## V – Split the dataset:

One of the most important steps when implementing a machine learning model is to split our data into training and testing sets to evaluate the performance of our model when we provide it with new data.
For this reason, we use the function **split_data** that takes as argument the cleaned dataset and returns the split with a test_size. We set the size 0.7 so that to train our models with 70% of our data and evaluate its performance with

## VI – Feature selection & Training models:

First, we select features through Principal Components Analysis (PCA) with the function **apply_pca**. It allows us to only keep directions that maximize variance in the data. We choose a certain level of explained variance as a threshold to retrieve only the most relevant components, or directly the number of components to keep. Below, we can visualize data projected on the 2 or 3 main components.

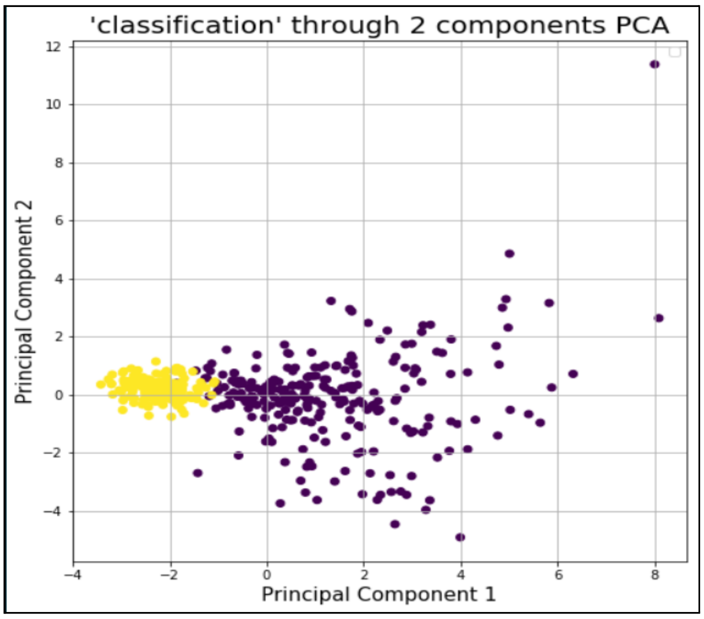


***Figure 9: PCA (2d) on Chronic Kidney Disease Dataset***

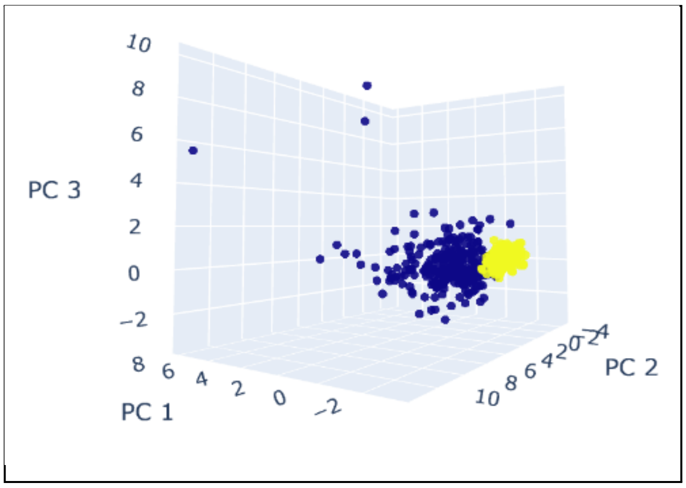
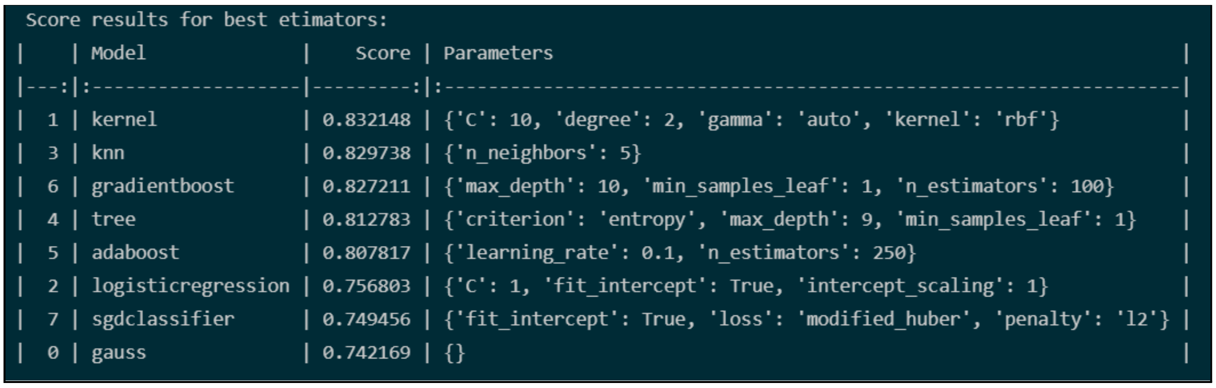


***Figure 10: PCA (3d) on Chronic Kidney Disease Dataset***

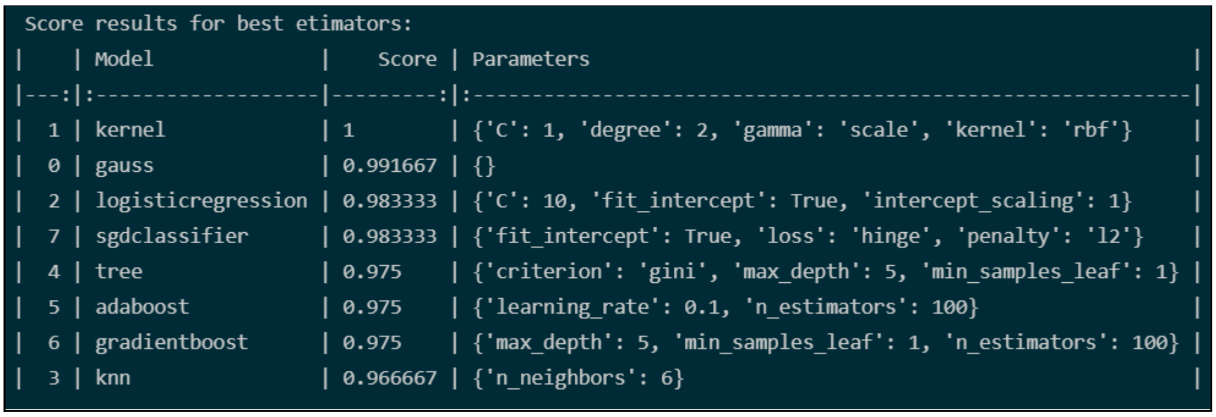
After preparing our data for Machine Learning algorithms, all that remains to do is to pick a model and train it. However, there are numerous models able to classify between such data. To maximize our chances, we first try multiple relevant classification models, and then we use a grid search procedure to optimize each model parameters from ranges of possible values. This is done using the function **train_models** that take in input the models, with for each one the parameters to try and their range of values to test. Then, cross-validation is used to keep only the best parameters for each model, which are those maximizing the mean score over the cross-validation. The best parametrized models are then trained on all train data. The scores calculated with cross-validation are relevant to compare the different models with optimized parameters. Thus, in the perspective of the end to end pipeline to create the better model, we would only keep the model with the higher score (which is the accuracy by default). To go further, we can use multiple score metrics, as in the following section, but the selection of the best model should be done manually in this case, based on the different score results.

Below, the results using 8 different models (Gaussian Naive Bayes, SVM, Logistic Regression, KNN,  Decision Tree, Random Forest, AdaBoost, Gradient Boosting and Stochastic Gradient Descent), comparing their mean cross-validation score:



```
Score results for best etimators:
|    | Model              |    Score | Parameters                                                       |
|---:|:-------------------|---------:|:-----------------------------------------------------------------|
|  1 | kernel             | 0.832148 | {'C': 10, 'degree': 2, 'gamma': 'auto', 'kernel': 'rbf'}         |
|  3 | knn                | 0.829738 | {'n_neighbors': 5}                                               |
|  6 | gradientboost      | 0.827211 | {'max_depth': 10, 'min_samples_leaf': 1, 'n_estimators': 100}    |
|  4 | tree               | 0.812783 | {'criterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 1}  |
|  5 | adaboost           | 0.807817 | {'learning_rate': 0.1, 'n_estimators': 250}                      |
|  2 | logisticregression | 0.756803 | {'C': 1, 'fit_intercept': True, 'intercept_scaling': 1}          |
|  7 | sgdclassifier      | 0.749456 | {'fit_intercept': True, 'loss': 'modified_huber', 'penalty': 'l2'} |
|  0 | gauss              | 0.742169 | {}                                                               |
```

***Figure 11: training results on Banknote Authentication Dataset***



```
Score results for best etimators:
|    | Model              |    Score | Parameters                                                     |
|---:|:-------------------|---------:|:---------------------------------------------------------------|
|  1 | kernel             | 1        | {'C': 1, 'degree': 2, 'gamma': 'scale', 'kernel': 'rbf'}       |
|  0 | gauss              | 0.991667 | {}                                                             |
|  2 | logisticregression | 0.983333 | {'C': 10, 'fit_intercept': True, 'intercept_scaling': 1}       |
|  7 | sgdclassifier      | 0.983333 | {'fit_intercept': True, 'loss': 'hinge', 'penalty': 'l2'}      |
|  4 | tree               | 0.975    | {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 1}   |
|  5 | adaboost           | 0.975    | {'learning_rate': 0.1, 'n_estimators': 100}                    |
|  6 | gradientboost      | 0.975    | {'max_depth': 5, 'min_samples_leaf': 1, 'n_estimators': 100}   |
|  3 | knn                | 0.966667 | {'n_neighbors': 6}                                             |
```

***Figure 12: training results on Chronic Kidney Disease Dataset***

Therefore, if we have to select only one model from the training, it would be an SVM model (named here ‘kernel’) for both datasets, with their respective parameters.

## VII – Model Validation:

In order to validate the model, we created the function **evaluate_model** that computes the F1-score, the accuracy,  the precision, and the recall of the predictions of the input model on the test data. Using several metrics allow us to get a better and more general overview of the performance of our model.

Below the results, evaluating all the previous models from training:

| Model | F1-score | Accuracy | Precision | Recall |
|---|---|---|---|---|
| **kernel** | 82.14 | 82.41 | 81.99 | 78.01 |
| **knn** | 82.51 | 82.73 | 81.52 | 79.63 |
| **gradientboost** | 82.43 | 82.62 | 80.89 | 80.32 |
| **tree** | 79.44 | 79.50 | 74.74 | 82.18 |
| **adaboost** | 78.28 | 78.26 | 73.73 | 80.56 |
| **logisticregression** | 71.73 | 72.42 | 72.03 | 63.19 |
| **sgdclassifier** | 59.32 | 60.67 | 57.63 | 47.22 |
| **gauss** | 69.71 | 70.97 | 72.98 | 56.25 |

***Table 1: validation results on Banknote Authentication Dataset***

Therefore, in this case, we observe that the 'best' model depends on the metric we use, and the best model based on the cross-validation procedure during the training part won't necessarily be the same on new data.

## VIII – Conclusion:

To conclude, we have built an end to end pipeline which allows a user to train a model and get its score on one of the two given datasets. We've used different coding techniques and collaboration techniques. Git has been a very important tool in our project, and we have learnt how to manage a Machine Learning project remotely during this exercise.

Besides, this project has shed light on practices that help to write better code as a group. First, concerning Git, some good practices are to give clear and meaningful commit titles, to only commit code that runs, to pull frequently and before committing. Moreover, it also appears that it helps to create another branch than the main if you have doubts about your code. Then, about writing Python code, it is better if everyone uses the same code style, add information on variable types and comments, and favour readability rather than brevity (*import this*). Finally, it is relevant to organize our functions in a meaningful manner in our file, and to separate function definitions from the main file.