SERVER.PY

In the last document we made the following:

First of all, to create web routes in a very simple way we use Flask:
"From flask import Flask" to which you must subsequently quote: app = Flask (__ name__, static_folder = '/ static');

To import the classes, already defined in the previous files: "Flask, render_template, send_from_directory, request and jsonify", we use the keyword "import".

The "numpy" and "pandas" classes, such as "np" and "pd" respectively, are also imported.

Subsequently, we import the classes "svm" from "sklearn", "cross_val_score" from "sklearn.model_selection" and frequency, instances, pca Standardizatio and preprocessing. On this last occasion, as we can see, we have not declared any package for our class since we have only made the import, however, this import as such is something particular for us, because we have imported ALL the classes belonging to the package.

To leave "clf" blank, we equate it to "none". "Last_id" we equate to the figure 30000.

Next, the web applications are defined:

```
@app.route("/")
def index():
    return render_template("charts.html")
```

First of all, we must bear in mind that Chart.html is a javascript library to create simple and clean graphics. All of them are based on HTML5, responsive, modular, interactive and there are a total of 6 graphics.
It must also be taken into account that, in the previous files, the web application has been created, which when called, shows the corresponding graphic and is called index, We simply pass the arrays to render_template(). This means that most of the magic occurs in the template. Chart.js is a client-side javascript library which is why our app.py is very minimal.

Something similar we do to send a file from a given directory with send_file (). This is a safe way to quickly expose static files from a loading folder or something similar. In our project:

```
@app.route('/<path:path>')
def send_js(path):
    return send_from_directory('static', path)
```

Parameters:

- **directory** – the directory where all the files are stored: static
- **filename** – the filename relative to that directory to download: path
- **options** – optional keyword arguments that are directly forwarded to send_file().

The last web application serves to obtain the different values, that is the reason why it has been called: "getValues"

The "Request.Form" command is used to collect values in a form with "method" = "post" "(how to interact with the user).

The information sent from a form with the POST method is invisible to others and has no limits on the amount of information that must be sent. In this case the forms are "key" and "value".

Subsequently, the global variable "last_id" is defined and, by calling the different previous routes, values are given to the variables. Ends by returning the string on the screen with the declaration of "return".

Finally: *"argv"* represents all the items that come along via the command-line input, but counting starts at zero (0) not one (1), len(sys.argv)!= 2 just checks that is not the same as two.
Display by screen: "You have to specify if you want the accuracy or not! Use 0 if not, 1 if yes".

Exit (-1) is generally used to indicate an incorrect termination.

Once this condition has been introduced, the learning phase begins by calling the corresponding actions already defined above.

The characteristics of the data are "SEX", "EDUCATION", "MARRIAGE", "AGE", "BILL_AMT1", "BILL_AMT2", "BILL_AMT3", "BILL_AMT4", "BILL_AMT5", "BILL_AMT6", "PAY_AMT1", "PAY_AMT2", "PAY_AMT3", "PAY_AMT4", "PAY_AMT5", "PAY_AMT6".

Read_csv: is an important pandas function to read csv files and do operations on it.
Opening a CSV file through this is easy. But there are many others thing one can do through this function only to change the returned object completely. In this case, the delimiter is ";".The header is where you tell Java what value type (0, in this case), if any, the method will return (an int value, a double value, a string value, etc).

In *x* we enter the numeric data of the database where the characteristics are but in matrix format with np.array ().

In *y* enter the values of the data "default payment nex month" in list format.
As for clf = svm.SVC (kernel = "rbf", verbose = True) the fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples.

kernel: ***string, optional (default='rbf')***
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used.

verbose : ***bool, default: False***
Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

The clf (for classifier) estimator instance is first fitted to the model; that is, it must *learn* from the model. This is done by passing our training set to the fit method

As to sys.argv, is automatically a list of strings representing the arguments (as separated by spaces) on the command-line. The name comes from the C programming convention in which argv and argc represent the command line arguments.

To use it, you will first have to import it (import sys) The first argument, sys.argv[0], is always the name of the program as it was invoked, and sys.argv[1] is the first argument you pass to the program.

When it is satisfied that "sys.argv [1] ==" 1 "":

1. The following message is printed on the screen: "Evaluating performances".

2. By means of cross_val_score" we evaluate a score by cross-validation.
   The parameters are:

   - **estimator :** ***estimator object implementing 'fit' (CLF)***
     The object to use to fit the data.

   - **X :** ***array-like***
     The data to fit. Can be for example a list, or an array.

   - **y :** ***array-like, optional, default: None***
     The target variable to try to predict in the case of supervised learning.

   - **Scoring :** ***string, callable or None, optional, default: None***
     A string (see model evaluation documentation) or a scorer callable object / function with signature scorer(estimator, X, y) which should return only a single value.
     Similar to cross_validate but only a single metric is permitted.
     If None, the estimator's default scorer (if available) is used.