**IBN ZOHR UNIVERSITY FACULTY OF SCIENCES – AGADIR**

**Center of Excellence**

**Program: Computer Engineering and Embedded Systems**

**Machine Learning Project Report**

**Lab Group: 3**

**Prepared by:** ZGAR LATIFA

OUBOURHAIL CHAYMA

**Breast Cancer Classification Using Machine Learning Algorithms**

**Abstract**

This project focuses on the use of machine learning algorithms to predict the nature of breast tumors based on clinical data. The dataset used is the Breast Cancer Wisconsin (Diagnostic) dataset. The goal is to develop and compare several classification models to identify the most effective one in distinguishing between benign and malignant tumors.

1. **Introduction**

Breast cancer is one of the leading causes of mortality among women. Accurate diagnosis is therefore essential to ensure timely and appropriate treatment. Machine learning can provide crucial support in predicting medical diagnoses. This project aims to evaluate several classification models applied to biopsy data from breast tumors.

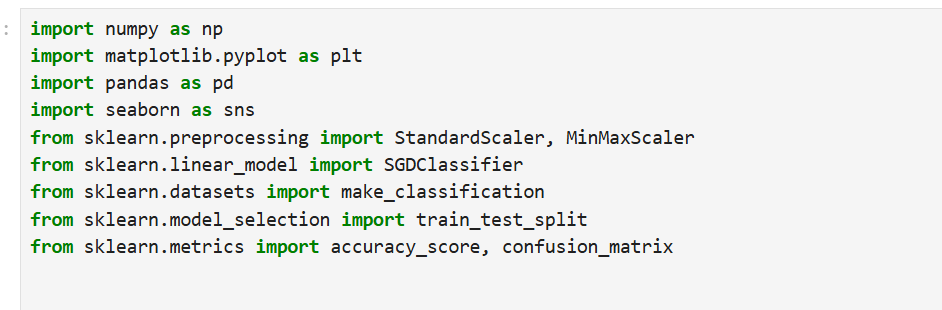
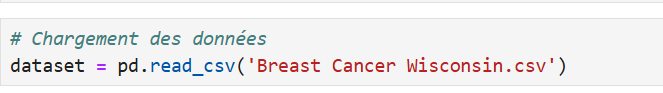
1. **Problem Statement**

With the growing volume of medical data, traditional diagnostic methods are showing their limitations. How can we effectively predict whether a tumor is benign or malignant based on clinical measurements extracted from biopsies?

1. **Dataset Description**

* **Name:** Breast Cancer Wisconsin (Diagnostic)
* **Size:** 569 samples
* **Variables:** 30 features (statistical measurements of cell nuclei) with one target variable ('diagnosis')
* **Classes:** M (malignant), B (benign)
  1. **Models Used:**

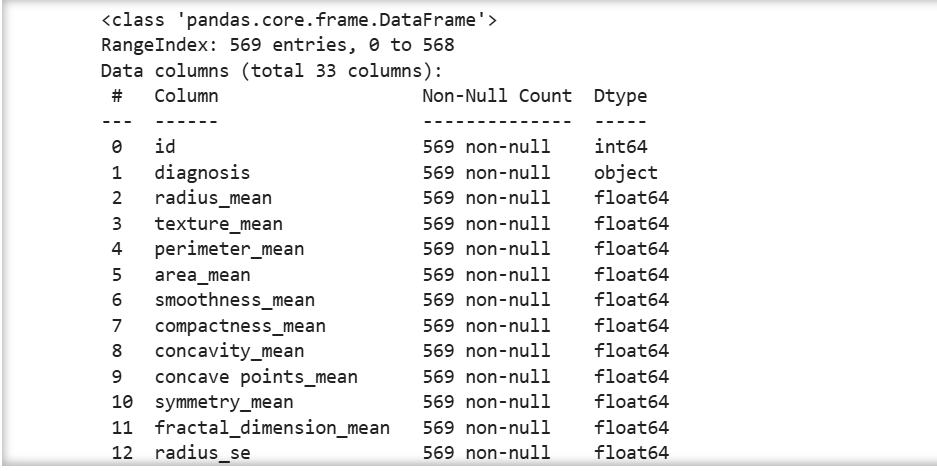
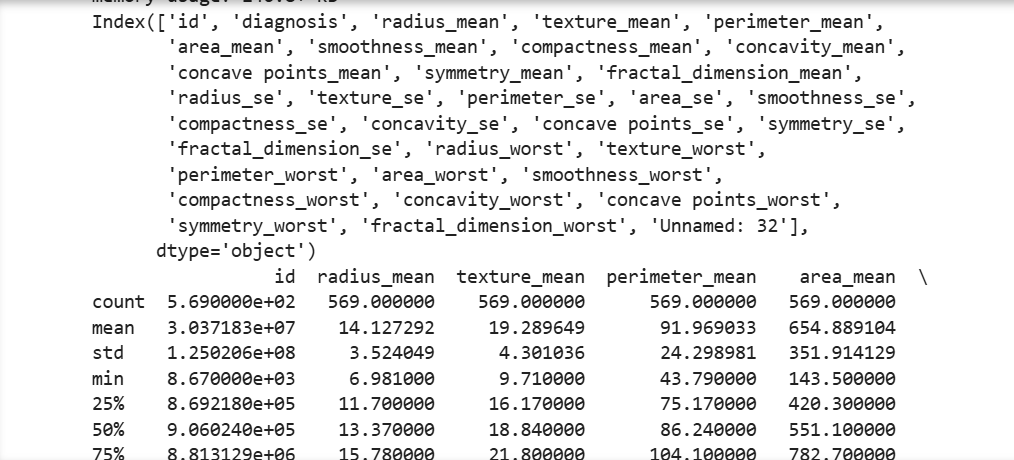
**Logistic Regression**

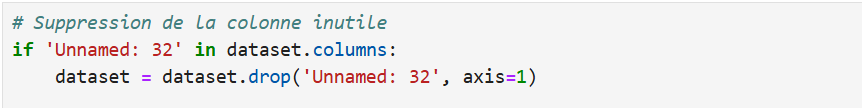
Logistic regression is a supervised classification algorithm used to predict binary categorical variables. In this report, we apply logistic regression using the SGDClassifier on the Breast Cancer Wisconsin dataset to predict whether a tumor is malignant (M) or benign (B).  

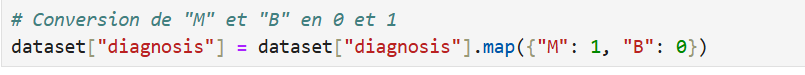
The command is used to load data from a CSV file. 

These commands allow for quick exploration of the dataset’s structure and content. Each command displays the following information:

* Column data types
* Column names
* Descriptive statistics
* Dataset dimensions (rows and columns)
* 5 random samples



Delete an empty column if it exists (such as "Unnamed: 32", which is often generated automatically). 

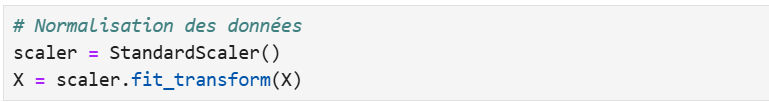
And the following command is used to encode the target variable:

* M (malignant) → 1
* B (benign) → 0

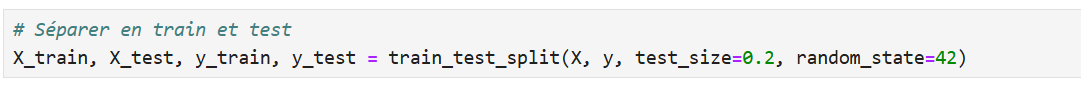


The command displays the number of missing values per column.

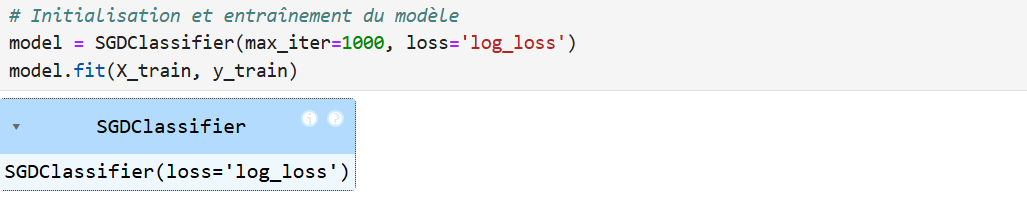
* + - Selects all columns starting from index 3
    - values converts the data into a NumPy array
    - y = dataset["diagnosis"].values retrieves the **diagnosis** column and converts it into a NumPy array

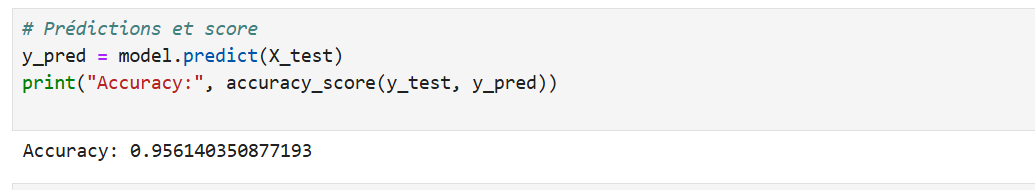


We normalize the data to improve the model’s convergence. Each variable is standardized (mean of 0, standard deviation of 1). 

We flatten y to obtain a one-dimensional vector, which is required for certain models.  We split the dataset into two parts:

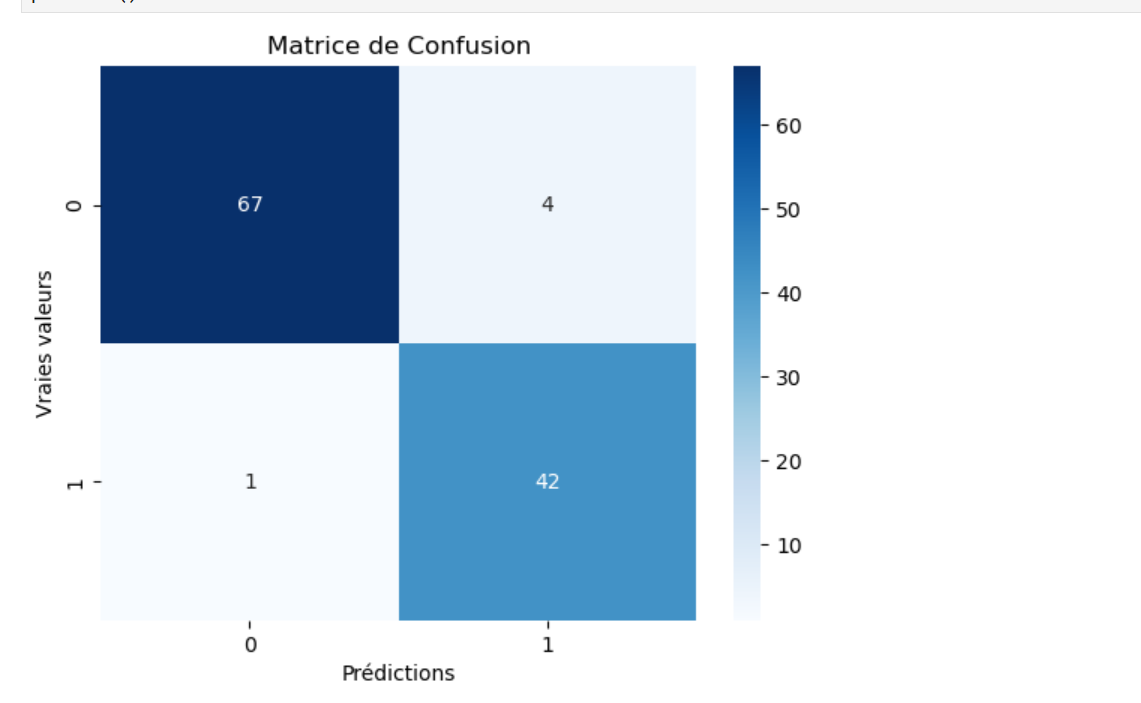
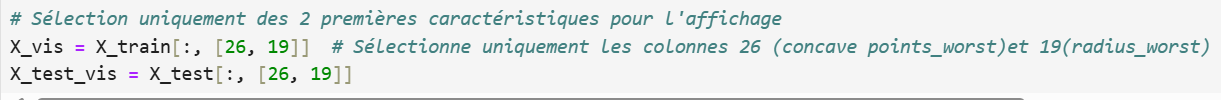
* 80% to train the model (X\_train, y\_train)
* 20% to evaluate it (X\_test, y\_test)



We initialize a classifier using logistic regression (with the log\_loss function), trained via stochastic gradient descent, and then fit it to the data. 

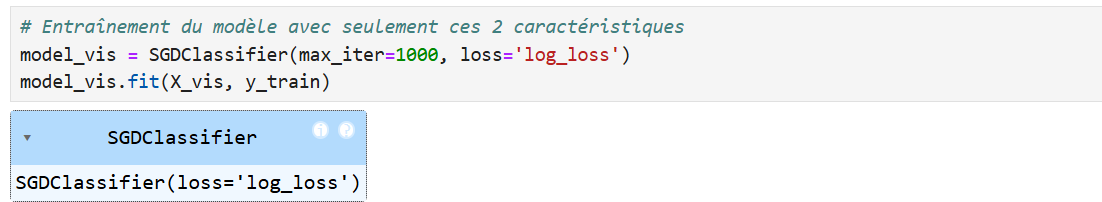
We use the .predict() method to generate predictions from the model trained on the test data. Next, we measure the model’s accuracy, which is the proportion of correct predictions out of all predictions made.

**Accuracy = 0.956140350877193**

This means the model correctly predicted approximately **95.61%** of the cases in the test dataset.  We generate a confusion matrix to visualize the model’s performance.  

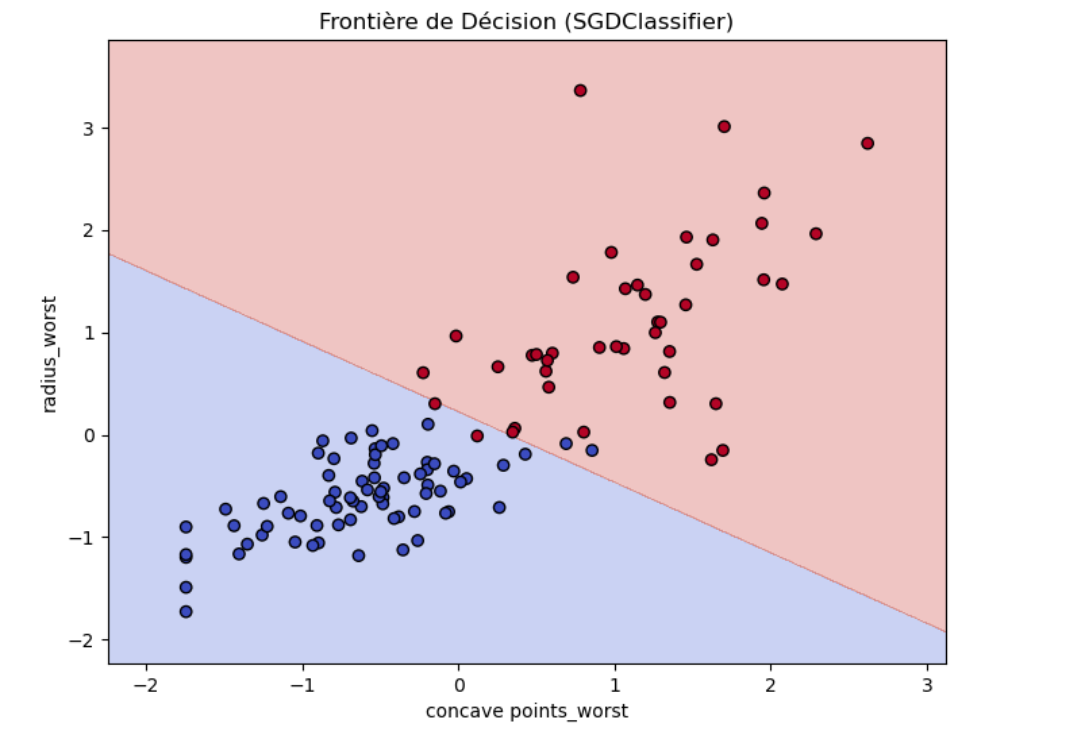
We select two highly discriminative variables:

* concave points\_worst (index 26)
* radius\_worst (index 19)

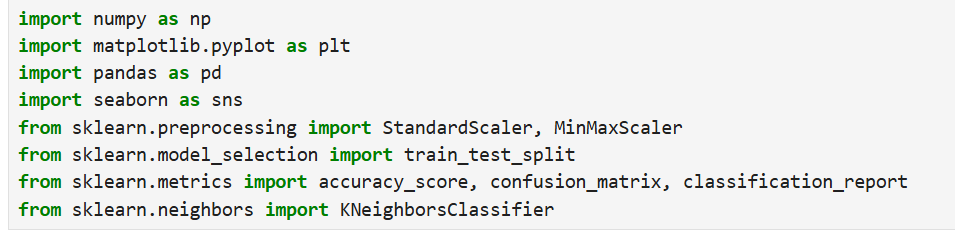
This allows us to visualize the decision boundary in 2D. 

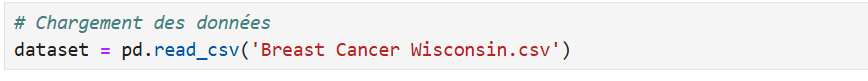
We retrain a model using only these two variables. 

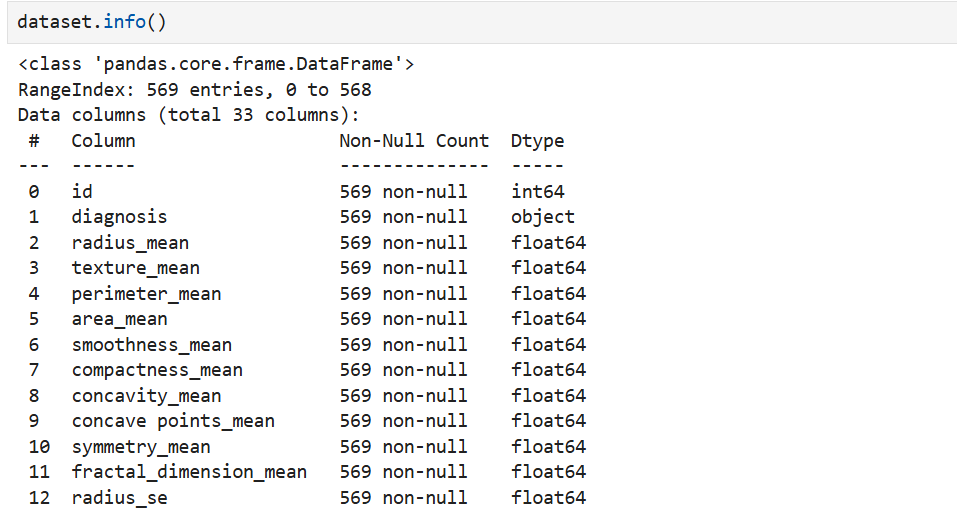
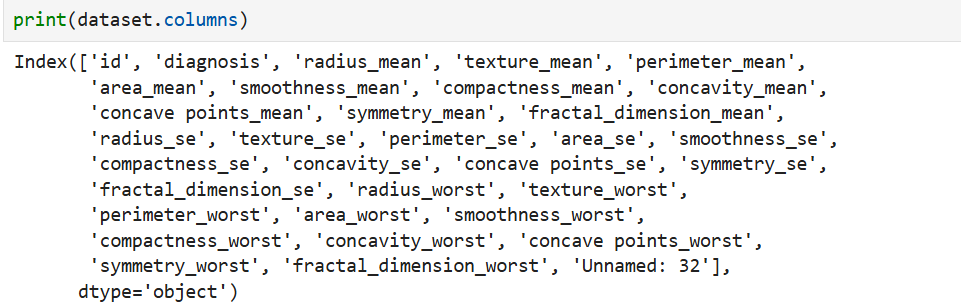
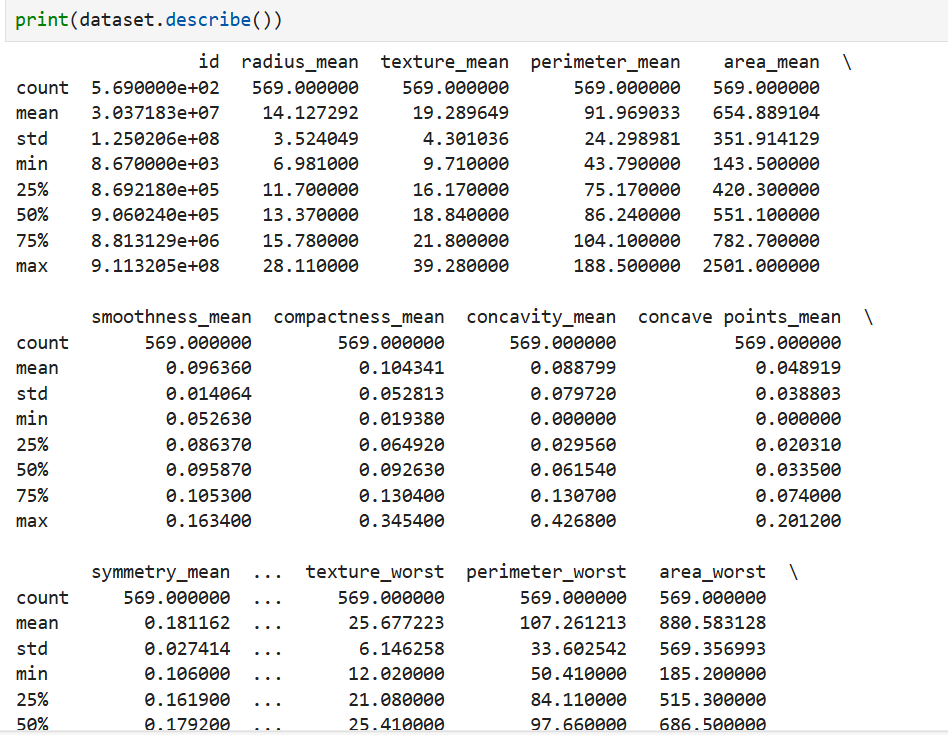
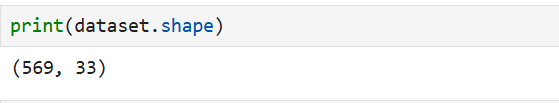
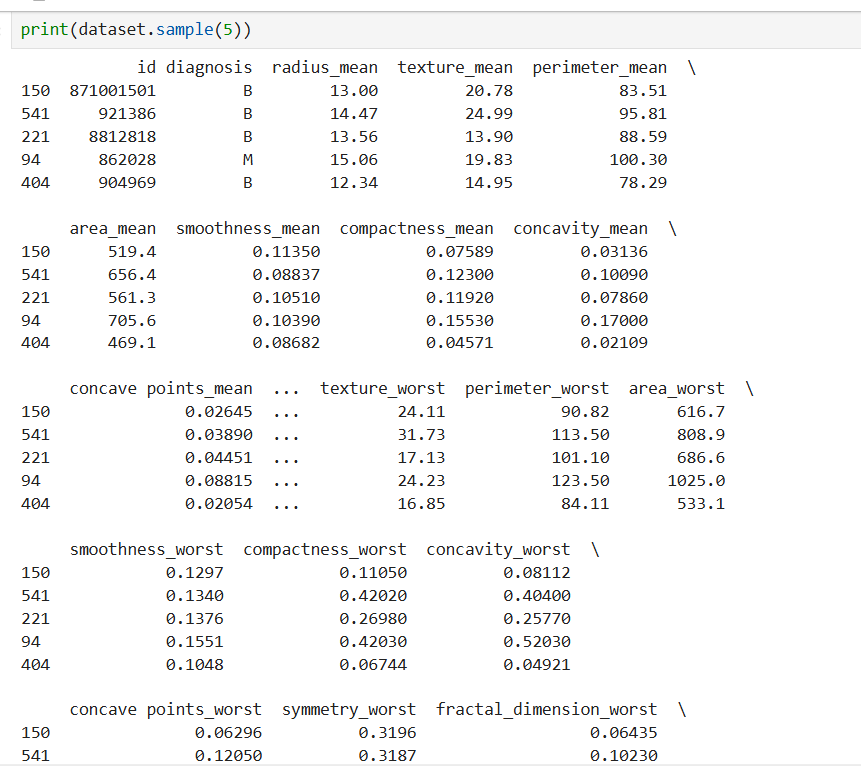
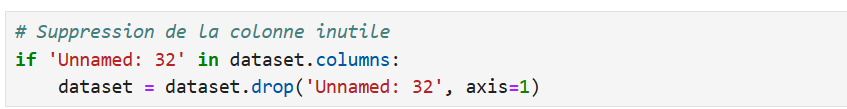
As part of our study on tumor classification, we implemented a plot\_decision\_boundary() function to visualize how our model separates benign from malignant cases.

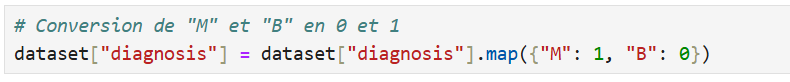
* Thousands of points spaced 0.01 units apart are generated to cover the entire plot.
* The model predicts the class (0 or 1) for each point in the grid.
* contourf(): Colors the regions based on the predicted class (blue for benign, orange for malignant).
* scatter(): Overlays the actual data points to compare with the predictions.
* The result shows that blue points (true benign cases) fall within the blue region, and orange points (true malignant cases) fall within the orange region.
* 

K-Nearest Neighbors (KNN) The KNN

algorithm is a classification model used to predict whether a tumor is benign or malignant. The principle is simple: a new tumor is compared to the K closest tumors in the dataset (based on their features). Then, the model checks the majority class among these neighbors: if most are malignant, it predicts “malignant”; otherwise, it predicts “benign.” ». 

* We import **NumPy** for mathematical operations.
* **matplotlib.pyplot** and **seaborn** are used to display visualizations (such as the confusion matrix or error plots).
* **pandas** is used to read and manipulate the dataset.
* We use **StandardScaler** and **MinMaxScaler** to normalize the data.
* **train\_test\_split** is used to divide the data into training and testing sets.
* **accuracy\_score**, **confusion\_matrix**, and **classification\_report** are used to evaluate the model’s performance.
* Finally, **KNeighborsClassifier** is the KNN model we will train. 

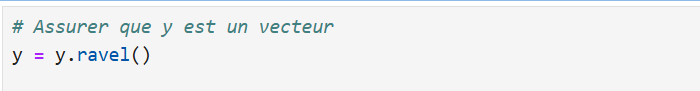
We load the CSV file containing breast cancer data into a DataFrame named dataset.      

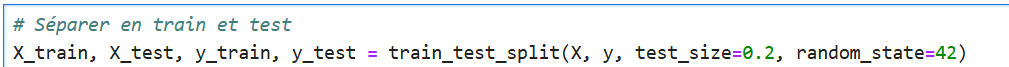
We remove the column "Unnamed: 32" if it exists, as it is an empty and unnecessary column. 

We replace "M" (malignant) with 1 and "B" (benign) with 0 to convert the classes into numeric format, which is required for the algorithm. 

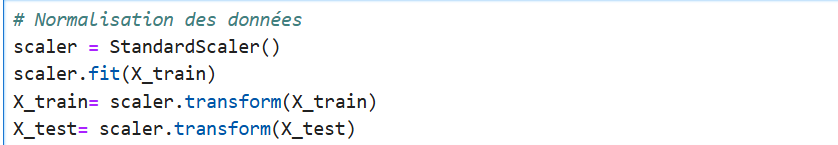
We check that there are no missing values in the dataset. The output confirms that all columns contain 0 missing values. 

* We select the features starting from column 3 to the end (excluding ID and name).
* y contains the target variable (0 or 1).

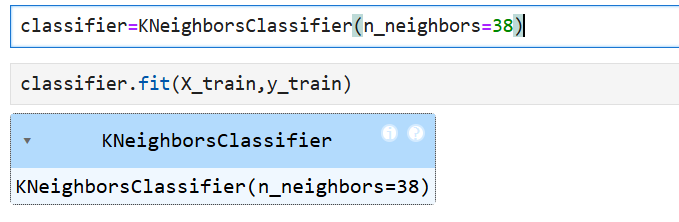


ravel() transforms y into a 1D array, which is necessary for training.

* **We split the data into 80% for training and 20% for testing.**
* **Setting** random\_state=42 **ensures reproducible results.**



**Before normalizing the data with** StandardScaler**, you must first train the scaler by calling** .fit() **on the training data. Then, you can apply** .transform() **to normalize the values. This step is important to avoid the** NotFittedError**.**



**We create a KNN model with 38 neighbors.**

We train the model using the training data.