PROJECT REPORT
MACHINE LEARNING / DATA SCIENCE

SUBJECT: STROKE PREDICTION

AMRANI-HANCHI LINA
BENMOUSSA DINA
BENNANI NADA

Project supervised by Guilhem PIAT

TABLE OF CONTENTS

Associated dataset: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

I. Introduction

For this research our dataset was taken from Kaggle. The dataset is publicly available but contains information on individuals regarding health features. Due to the fact that medical information requires specific privacy and protection, the source of the dataset used for this project remains confidential. This dataset is aimed at predicting whether an individual is more likely or not to have a stroke based on the given characteristics.

Having a major interest in the medical field, we chose to work on this specific subject that is predicting strokes based on multiple medical attributes. Strokes consist of a serious medical condition that affects millions of people worldwide with significant impacts on individuals such as disabilities and often death. Thus early detection and prevention are crucial elements in the reduction of the consequences occurring following a stroke.

Detecting symptoms early can provide valuable information for the prediction of strokes and promoting a healthy lifestyle. This research employs Machine Learning algorithms to develop and evaluate several models, thus creating a framework for predicting long-term stroke-risk.

The main challenging aspects regarding the use of this dataset were the presence of categorical variables and the fact that it was highly unbalanced.

II. Motivations

Based on this dataset, our goal would be to develop a classification model that accurately predicts the likelihood of an individual having a stroke based on various risk factors.

The first step would be to explore and clean the dataset. We will identify features type, missing values, outliers and unnecessary features.This would involve analyzing the distribution of data and addressing any class imbalance issues that may exist.

The next step would be to analyze the correlations between variables and to extract information related to feature importance to perform feature selection if needed.

After that, we will start testing models such as SVC, Logistic Regression, RandomForm, XGB, models that seem to suit our classification task. We will also use some hyperparameter tuning to optimize the performance of our models using the grid search technique.

Finally, we will evaluate models that seem to perform best using accuracy, precision, recall, F1 score to compare them and conclude on the overall best performing model for this type of classification task.

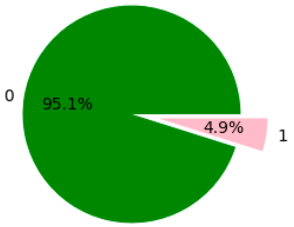
III. Brief description of the dataset

The dataset we used contains information on 5110 patients with 12 attributes for each one, ‘stroke’ being the target variable in this case. The features are: [id: unique identifier; gender: "Male", "Female" or "Other"; age: age of the patient; hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension; heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease; ever_married: "No" or "Yes"; work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"; Residence_type: "Rural" or "Urban";

avg_glucose_level: average glucose level in blood; bmi: body mass index; smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*; stroke: 1 if the patient had a stroke or 0 if not]

IV. Data analysis and visualization

We first analyzed the distribution of the numerical features that were available. Then we studied the correlation between features and the stroke, and between features themselves in order to establish a link between variables and to know if there was any direct link to the probability of having a stroke.

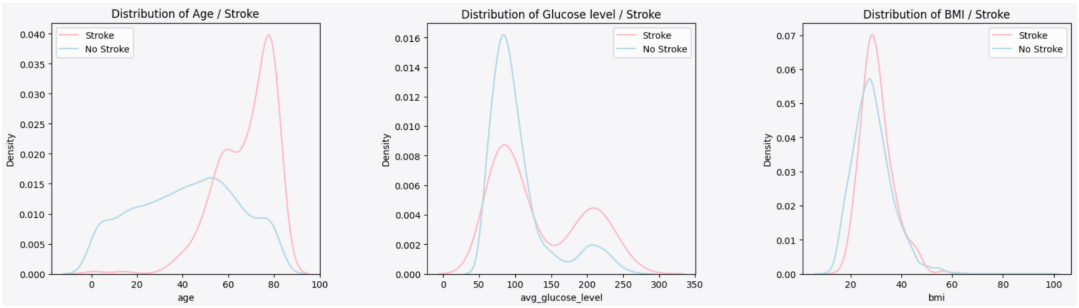


0: non stroke
1: stroke

Plot 1 : Number of stroke in the dataset

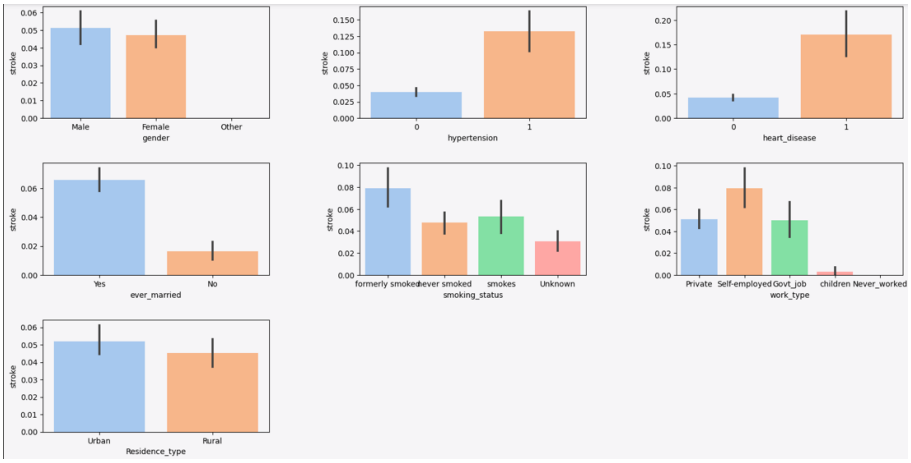
The first observations we made during our data analysis is that the dataset is **very imbalanced** in favor of the negative class(0: 'non stroke') *[plot 1]*. This is a common situation in Machine Learning but it can lead to a biased model that can prioritize the majority class and fail to accurately predict the minority class. However, in our case, the minority class (1: 'stroke') is the one that is the most important regarding the original problem. To address this issue we decided to oversample the minority class, which we will explain more in detail further.

To gain a better understanding of our dataset, we decided to establish the correlation between the different features and stroke.



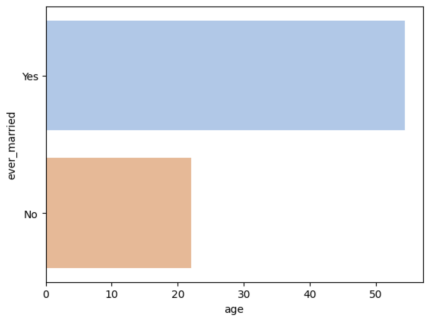
Plot 2: Correlation between numerical features and strokes

The graph shows the correlation between stroke and three numerical attributes (age, BMI, and average glucose) using density curves. The density curves for each attribute are shown separately for individuals with and without stroke. The curves reveal that the distribution of age and average glucose levels for individuals with stroke is shifted towards higher values compared to those without stroke. In contrast, the distribution of BMI appears to be similar for individuals with and without stroke. **Overall, the graph suggests that age and average glucose levels may be important factors in predicting the risk of stroke, while BMI may not be as strongly associated with stroke risk.**



Plot 3: Correlation between categorical features and strokes

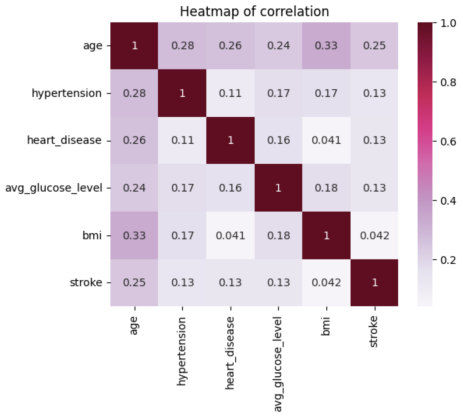
The graph presents the correlation between stroke and several categorical attributes using bar plots. Each bar represents the proportion of individuals with and without stroke for a particular category of the attribute. The graph reveals that the proportion of males and females with stroke is roughly the same, but the proportion of individuals with hypertension or heart disease is significantly higher in the stroke group compared to the non-stroke group. Similarly, the proportion of ever-married individuals and smokers is higher in the stroke group. Regarding work type, individuals in private jobs have a higher proportion of stroke compared to other categories, while residence type does not seem to be strongly associated with stroke risk. **Overall, the graph suggests that hypertension, heart disease, ever-married status, smoking status, and work type may be important predictors of stroke risk.** Upon studying all of these correlations, the correlation between age and ever-married piqued our interest and may prove to be useful later on.



Plot 4: correlation between the age and the marital status

The diagram compares the age and ever-married attributes in the dataset. It shows that **people who have ever been married tend to be older than those who have never been married.** This indicates that the correlation between age and ever-married is likely due to the age of the individuals rather than their marital status. Therefore, age may be a more important factor in predicting the risk of stroke than ever-married status. *Does this correlation alone warrant considering the removal of the 'ever-married' feature from the predictive model?*

We chose to end with a heatmap of feature correlation as it provides a comprehensive and visual representation of the strength and direction of the correlations between different features, allowing us to identify potential patterns and relationships in the data.



Plot 5: heatmap of features correlation

This correlation matrix confirms the observations made earlier. In fact, we can observe that there is a positive correlation between age and stroke , glucose level and stroke and no significant correlation between BMI and stroke as explained earlier.

After observing the correlation between all the features, we concluded that in general there is no significant correlation. However, this doesn't mean that they don't have an impact on the prediction of strokes.

At this point of the study, it seems that the variables that have the biggest influence on the prediction of stroke are **age, hypertension, heart_disease and self_employment**.

We will see if these assumptions change later in the process.

V. Data Preprocessing

By following the next preprocessing steps, the stroke prediction dataset can be transformed into a format suitable for machine learning algorithms.

a.Feature selection:

Since our data consists of 5110 records and 12 features, it doesn't seem necessary to do a dimensionality reduction (2^12=4096). Still it may be possible to remove 1 or 2 features if necessary.

After analyzing our data very carefully and studying the correlation between variables it seems that overall the correlations between features are not really significant. One of the only significant correlations was between stroke and age.

Also, as we have just mentioned earlier, the ever_married feature was correlated with age, and age with stroke so we wanted to remove that feature from our dataset. In fact, we assumed that the older a person is, the more likely it is that this person has already been married. Thus it seems that removing the ever_married feature could help us reduce dimension to keep only relevant features. However, eliminating the ever-married attribute is not feasible as it provides additional information on stroke, such as the impact of long-term companionship and social support on stroke risk. Therefore, while age may be a more important factor in predicting the risk of stroke, the ever-married attribute should be considered in any predictive model as it may contribute to a more accurate risk assessment and provide insight into the psychological and social factors associated with stroke risk. That is why we

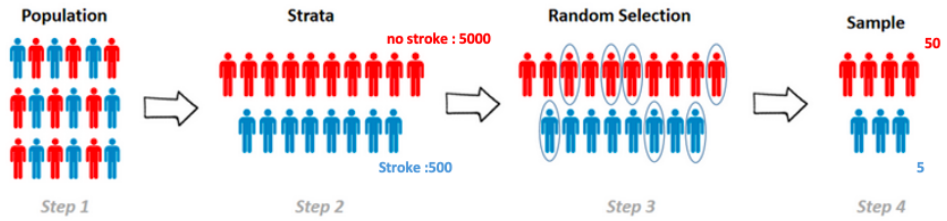
finally decided to keep that feature even if it was one of the only one that we thought was very unnecessary due to the parallel correlation.
We decided to drop the id column since it wasn't giving any relevant information..

b.Handling categorical data:

We decided to replace binary categorical data for the features : gender, hypertension, heart_disease, ever_married and residence type by binary numerical values (0 and 1) to encode categorical data into numerical data.
For features like work_type, smoking_status, we decided to perform a one-hot-encoding since we can have more than 2 values for these features. We thought it was wiser to choose this encoding and not another one like binary encoding to avoid creating an ordering relationship between level or even target encoding to avoid causing data leakage.
For the smoking status we finally decided to go for a one hot encoding and forget about the idea of the order encoding since our features (formally_smokes , never_smokes, smokes, unknown_smoker) do not allow us to sort the unknown smoker with the others. Even if we create a separate feature for the unknown and do an ordering encoding for the other features this is not working since we have to put a value inside of it anyway for the unknown individual.

c.Stratification of data :

As we have an unbalanced dataset, the risk of doing a normal train/test split is that we could have a train or a test set containing only samples of the majority class, thus leading to unreliable learning and predictions from our models. That is why we decided to go for a stratified approach for the train/test split that consists in having the same proportion of each in both our train and test set (we use the parameter stratify of the train_test split function already implemented in scikit-learn). Explanation of stratified sampling:



d.Dealing with missing/unique values :

The bmi column contains some NaN values (201 out of 5110 records). One approach for handling those missing values is to impute it with the mean or the median BMI value of the dataset. We decided to replace these values by the mean as we didn't want to remove 201 rows of our dataset. We did this operation after the train and test split to avoid data leakage.
We also decided to drop one record that had the value "Other" for the gender feature as it was unique and prevented us from having a binary feature.

e.Scaling Data:

Scaling data is an important preprocessing step in machine learning that involves transforming the data to have a consistent scale or range. This is important because many machine learning algorithms are sensitive to the scale of the input features. Here we have different features with different units, and if not scaled, the algorithm will give more weight to features with larger scales. We scaled with StandardScaler that scales the feature to have zero mean and unit variance. StandardScalier is a good choice because it is actually less sensitive to outliers.

f.Oversampling :

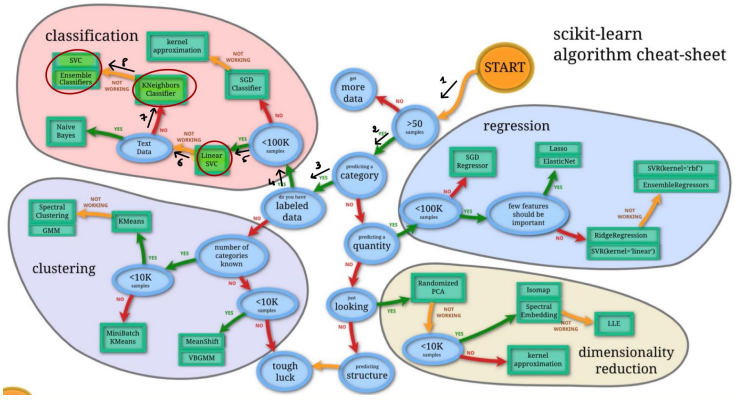
As we have seen before, 95,1 % of the samples are not having strokes against 4,9 % of the samples having strokes.

Oversampling is a technique used to address class imbalance, where one class of data is underrepresented in the dataset. In our dataset, only a small percentage of records belong to the positive class, which can cause machine learning algorithms to be biased towards the majority class. That is why we decided to use the SMOTE oversampling technique that creates samples of the minority class by interpolation between existing samples in the feature space. By oversampling the minority class, the algorithms can better learn and distinguish the minority class from the majority class, leading to improved performance in predicting the positive class. However, it is important to note that oversampling can cause some overfitting and increased computation time.

Another approach is to undersample the dataset, but since the number of our data is not huge, we decided to go for the oversampling method.

VI. Modeling

In this section, we present the models we decided to use in the classification for stroke occurrence.There's a handy flowchart produced by Scikit-learn that can help decide which estimator to choose:



- I have more than 50 samples
- I'm predicting a category (stroke or no stroke)
- I have labeled data

- I have less than 100K samples
  - ➔ Linear SVC will go on our list.
- I don't have text data
  - ➔ KNeighborsClassifier will go on our list.
  - ➔ Ensemble Classifier will go on our list (Random Forest, Decision Tree)

The last ones are Logistic Regression which, despite its name, is a linear model for classification, and XGBoost classifier. We found those ones by searching for "best machine learning models for classification problems" online.

We started training the models on the training set and generated metrics and their results for cross-validation before oversampling, after oversampling our dataset and after hyperparametrization.

K-fold cross-validation is a technique that helps to evaluate the generalization performance of a model by splitting the dataset into k subsets and training the model on k-1 subsets while evaluating on the remaining subset. This process is repeated k times with each subset used as the validation set once. By taking the average performance of the model across these k-folds, it provides a more reliable estimate of the model performance.

For the hyperparametriaztion we decide to consider the F1 score as the metric to optimize. In fact, since in our special case we are predicting stroke, we decided that it is more important to prioritize true positives rather than distinguishing between true and false positives.

$$F_1\text{−score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \qquad \boldsymbol{Accuracy} = \frac{\boldsymbol{TP + TN}}{\boldsymbol{TP + TN + FP + FN}}$$

Since it is more important to diagnose a patient as positive when it is really the case, we decide to choose F1 as our metric to compare model performances on the training set.

Accuracy is a metric that is often used to evaluate the performance of machine learning models, but it can be misleading in the case of imbalanced datasets. In the case where the majority class is very much larger than the minority one, a model that simply predicts the majority class for all samples will achieve high accuracy, even though it may not be able to classify effectively minority class.

Here are some results of one of our executions and some comments about it.

a.Support Vector Classifier(SVC):

First, we considered the Support Vector Classifier, which is a supervised machine learning algorithm used for classification tasks. This algorithm works by finding a hyperplane in a high-dimensional space to separate the classes. The optimal hyperplane is one that maximizes the margin to allow better generalization. The margin maximization is what makes this algorithm efficient for binary classification because it will make the classifier less likely to make mistakes when predicting the class of a new data point. The kernel tricks allow the classifier to work well even when the data is not necessarily linearly separable. With appropriate tuning of the hyperparameters, it achieves a good performance.

Accuracy with k-fold cross-validation before oversampling: `0.9513093868354188`

F1 score with k-fold cross-validation before oversampling : `0.0`

`0.8886018758915364`

F1 with k-fold cross-validation after oversampling:
Hyperparameters: `{'C': 1, 'kernel': 'rbf', 'random_state': 0}`
F1 of the oversampled data after tuning: `0.8886018758915364`

The support vector classifier has an F1-score of 0 before oversampling and 0.8 after. This is because as the data before oversampling is highly unbalanced, the model seems to be choosing the compromise to always predict 0. In fact, by predicting 0, the model has 95% chances of being right since 95% of the data is people not having strokes. Thus by trying to learn to predict 1, the model would have more chances to make an error. This is why when oversampling the dataset, the F1-score becomes 0.87. Hyperparametrization doesn't seem to change the result of F1 score.

b. Logistic Regression:

The second model we included in our classification framework is Logistic Regression. It works by finding a relationship between two of the data features and then uses this relationship to predict the target feature. Logistic regression is efficient for binary classification in our case because it can be easily interpreted. It is a model that can handle linear and non-linear relationships between input variables and the target variable.
Accuracy with k-fold cross-validation before oversampling: `0.9513093868354188`
F1 score with k-fold cross-validation before oversampling : `0.0`
F1 with k-fold cross-validation after oversampling: `0.7986938166350146`
Hyperparameters: `{'C': 0.25, 'random_state': 0}`
F1 of the oversampled data after tuning: `0.7990484838702672`

We notice the same phenomenon for logistic regression where there is an F1-score of 0 before oversampling and 0.8 after also. There is, before oversampling, a high accuracy which kind of contradicts the null F1-score, justifying the fact that we focus on F1-score.

c. K-Nearest-Neighbors:

K-Nearest-Neighbors is a non-parametric supervised machine learning algorithm that takes as an input the k closest training examples in the dataset. The data point we want to classify is assigned to the class that is most common among its k neighbors. This algorithm relies on distance (Euclidian distance by default) so if the features are on different units or scales, normalizing the train set data can improve accuracy. It is efficient for binary classification because it is intuitive and does not require assumptions about the underlying distribution of the data.
Accuracy with k-fold cross-validation before oversampling: `0.9478846061652044`
F1 score with k-fold cross-validation before oversampling : `0.9169266729086087`

F1 with k-fold cross-validation after oversampling: `0.017786561264822136`

`{'metric': 'manhattan', 'n_neighbors': 5}`

Hyperparameters:
F1 of the oversampled data after tuning: 0.9291636410152527

The K-nearest-neighbors classifier has a F1-score that increases significantly after oversampling. It seems that this model cannot reliably predict the minority class since the scores are all very low before oversampling. This is why oversampling is necessary to obtain good results. As we see here, accuracy is high before oversampling even though the F1-score is really low, justifying the use of F1-score as the principal metric for our k-fold cross validation. There is an increase of the F1-score when using the best hyperparameters for the model.

d. Decision Tree:

Decision Tree is a supervised learning algorithm working by recursively partitioning the feature space into subsets based on a set of rules, with the aim of minimizing the impurity of the resulting subsets. It's efficient for binary classification in the case of predicting stroke because they can handle both numerical and categorical features. They are also able to capture non-linear relationships between the features and the target variable, making them suitable for complex classification tasks.
Accuracy with k-fold cross-validation before oversampling: 0.9082452418620258
F1 score with k-fold cross-validation before oversampling : 0.1368240790524459
F1 with k-fold cross-validation after oversampling: 0.9209975549348226
Hyperparameters: {'criterion': 'gini', 'random_state': 0}
F1 of the oversampled data after tuning: 0.9209975549348226

In the case of the Decision Tree classifier, we have the same effect as k-nearest neighbors occurring. We also notice a slight increase in the F1 score after the hyperparametrization.

e. Random Forest:

Random Forest is a Classification model where multiple decision trees are created using different subsets of the data and features and then combining their predictions to make a final prediction. Random Forest is efficient for binary classification as it has a good accuracy and it can handle feature selection and in our case it can give us a good estimate of the feature importance which is a good way to understand the underlying relationships between input features and target variables.
Accuracy with k-fold cross-validation before oversampling:
F1 score with k-fold cross-validation before oversampling : 0.009523809523809523
F1 with k-fold cross-validation after oversampling: 0.9627713597065259
Hyperparameters: {'criterion': 'entropy', 'n_estimators': 150, 'random_state': 0}
F1 of the oversampled data after tuning: 0.9643676033480233

For the Random Forest classifier we again have the F1-score that increases significantly after oversampling.

f. XGBoost:

Gradient Boost is an algorithm that is efficient and flexible. It can be used for both classification and regression tasks. It provides pretty accurate results even if hyperparametrization can be complex. It works by building different decision trees where each tree is built recursively to correct the errors of the previous ones. The model focuses on data points that are hard to predict thus improving accuracy. It uses regularization techniques to prevent overfitting.

Accuracy with k-fold cross-validation before oversampling: 0.9444580277098614
F1 score with k-fold cross-validation before oversampling : 0.09922312470138558
F1 with k-fold cross-validation after oversampling: 0.9543852582217911
Hyperparameters: {'eval_metric': 'error', 'learning_rate': 0.1}
F1 of the oversampled data after tuning: 0.9400598953950311

We notice here that in opposition to the other models, the F1 score decreases slightly after the hyperparametrization.

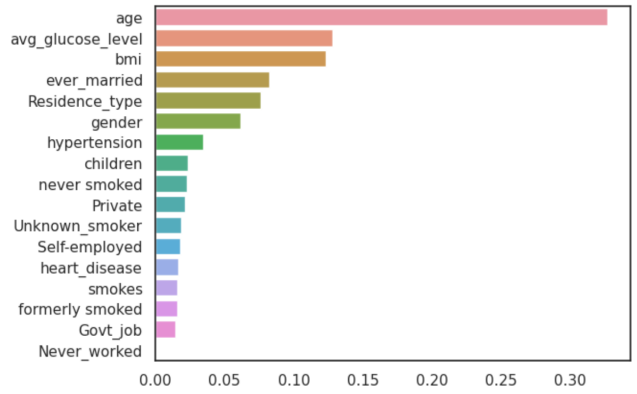
VII. Model evaluation

It seems that the Random Forest classifier and XGB (both hyperparametrized) have the best performance with cross validation on the training set for the oversampled data with respectively 0.96 and 0.94 for F1 comparaison metric.
Random forest seems to perform the best on the training set.
Let's evaluate our model to see the final results on the test set :

```
Accuracy Score:  0.9334637964774951
ROC AUC Score: 0.57
Precision: 0.24
Recall: 0.16
F1: 0.19
```

Based on the results, it appears that RandomForestClassifier has achieved a high accuracy score of 0.93 which indicates that it is able to correctly classify a large proportion of the data points.
However, the low ROC AUC score of 0.57 suggests that the model is not performing well in terms of distinguishing between positive and negative instances.
The precision score of 0.24 suggests that the model is correctly identifying only 24 % of the positive instances.
While the recall score of 0.16 suggests that it is missing a significant proportion of the positive instances.
The F1 score of 0.19 reflects the balance between precision and recall, and indicates that the model may not be performing as well as we wanted in terms of correctly identifying positive instances.
In summary, while the high accuracy score is promising, the low ROC AUC and low precision and recall score suggest that the model may not be performing well for this specific problem of stroke prediction.

For this classifier it seems that the most predictive features are age, bmi, and avg_glucose_level:



VIII. Conclusion

After the whole process, the exploitation of these Machine Learning models allowed us to choose the most efficient model for stroke prediction for unknown individuals and to get the most relevant risk factors for stroke occurrence. This is an important way of helping in early prevention.

We came to the conclusion that tree-based models performed the best, more especially the RandomForestClassifier that performs with an accuracy of 0.93 and F1 score of 0.19 for the prediction of stroke on unknown individuals. The most relevant predictive features were age, average glucose level and bmi which confirmed a part of our assumptions and also refuted some others.

We could suggest some improvements that we didn't have time to achieve especially regarding filling missing values. In fact, for the bmi feature, we could have predicted the missing values instead of filling them with the mean as the bmi depends on some characteristics that are specific to some individuals(gender, age, avg_glucose_level).

One thing that could have helped with our predictions is having a dataset with more individuals having a stroke in order to have a balanced dataset.