Data Mining: Second Project

Patient Survival Prediction

Ricard Guixaró Trancho

Ramon Ribas Domingo

David Daniel Streuli

Mario Font Blanc

Xavier Marti Llull

Facultad de Informática de Barcelona

Universitat Politècnica de Catalunya

**Supervisor**:

Mario Martín Muñoz

December 21, 2022

**Index**

[**Introduction**](#_pe1urjm5gitv) **3**

[Statement of Problem](#_xcls9arwrm5e) 3

[Methodology](#_k1ng059w7ql) 3

[Motivation](#_x7dkmulzw7gv) 3

[**Description of the Original Data**](#_vvybbkaxnfwq) **4**

[Description of the Dataset](#_g0xzahpuo48a) 4

[Description of Selected Features](#_pojo37eo0ena) 5

[Feature Correlation: Bivariate Analysis](#_uibv551hl2tv) 7

[**Description of the Pre-processing of the Data**](#_o0bqjpnesp26) **8**

[Deletion of Columns](#_uxj93ri8grhs) 8

[Outlier and Error Detection](#_kgubik38nopq) 8

[Data Imputation](#_wocfbjar3616) 9

[Description of the Pre-processed Dataset](#_cu0e7shm8nnt) 9

[**Evaluation Criteria of Data Mining Models**](#_8y8s46pcwgtz) **11**

[**Execution of Different Machine Learning Methods**](#_90myp2eqkiog) **12**

[Naïve Bayes](#_c6wrjn7qgt2v) 12

[K-NN](#_hj13ajht8bj1) 15

[Decision Trees](#_8y9jf4x5t0ts) 17

[Support Vector Machine](#_wgs70wfb1pd9) 21

[Linear SVM](#_q6wzo2yrbu98) 21

[Polynomial Kernel](#_1kz17cqt3p33) 22

[RBF Kernel](#_n459nkpg5gl4) 23

[Meta-learning algorithms](#_6qkjri8qj0m7) 24

[Voting Scheme](#_43ri5rmsw554) 24

[Bagging](#_yzbhsfp6n2x9) 26

[RandomForest](#_oco508h9o9zs) 30

[Adaboost](#_64jixc57shg5) 33

[Conclusions Meta-Learning Algorithms](#_1jgq3hdef79l) 34

[**Comparison and conclusions**](#_7pplfwq28ayh) **35**

[**Attachments**](#_6ybon05jz4if) **36**

# Introduction

In this chapter, an introduction to this paper is given.

## Statement of Problem

In this section, we state the problem. This is the second practical work for the data mining course at UPC Barcelona. In this work, we were tasked with finding a non-trivial dataset for a classification task (more than 20 columns, more than 1.000 examples and, preferably, with missing, noisy or heterogeneous data). The data mining process includes a description of the data, pre-processing and various machine learning algorithms.

## Methodology

In this section, the methodology for the description of the data mining process is stated.

In the first section, we describe the original dataset, then in the second section we describe the preprocessing of the data set. In the third section, we describe the evaluation criterias and in the fourth section, we describe the executions of the different machine learning algorithms. In the last section, we do a comparison and conclusion. The name of the files with the source code can be found in the Attachments.

## Motivation

In this section, the motivation for writing this paper will be stated.

In this project, we aim to apply various data mining methods learned in the *Data Mining* course on the chosen dataset and draw meaningful conclusions. We should get a better understanding of the complete data mining process, the current data mining technologies and how to apply them with Python.

# 

# **Description of the** O**riginal** D**ata**

In this chapter, we describe the original dataset that we have chosen and its structure. First, we give a literary description of the dataset and its relevance. Then we describe the dataset statistically.

## Description of the Dataset

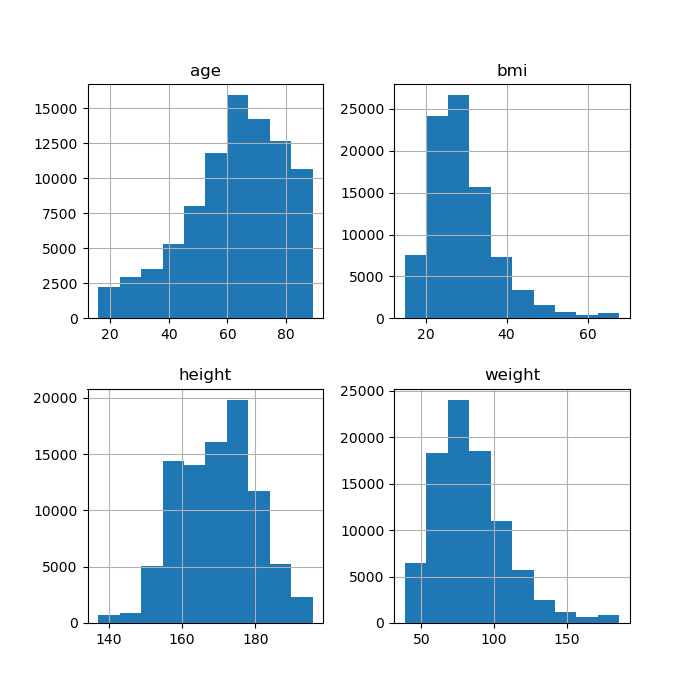
In this section we describe the dataset. The dataset that we have chosen is called *Patient Survival Prediction* and was obtained from Kaggle, an online community for data scientists. It contains information on hospitalised patients. The data is intended to make predictions about the survival of those patients. Each row in the dataset represents a patient and the columns represent various health measures such as blood pressure and heart rate, as well as personal data such as gender or ethnicity.

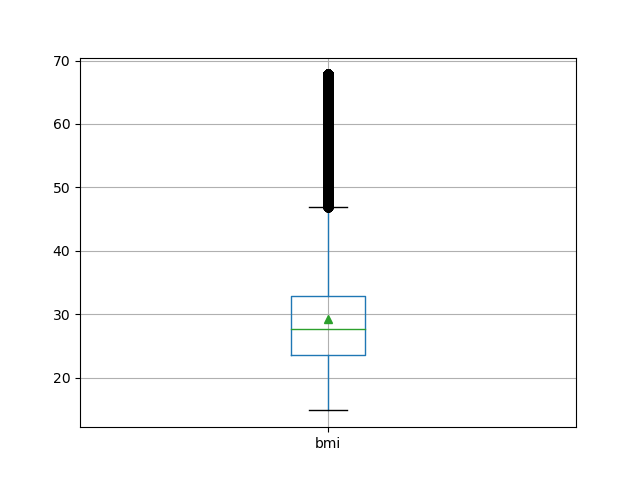
| **Author** | Mitisha Agarwal |
| --- | --- |
| **URL** | https://www.kaggle.com/dsv/2972359 |
| **DOI** | 10.34740/KAGGLE/DSV/2972359 |
| **Year** | 2021 |
| **Number of rows** | 91713 |
| **Number of columns** | 85 |
| **Integer** | 60 |
| **Decimal** | 13 |
| **String** | 7 |
| **Other** | 5 |

Below is a list of all features. A full description with metadata for each feature is included in **df\_des.csv**.

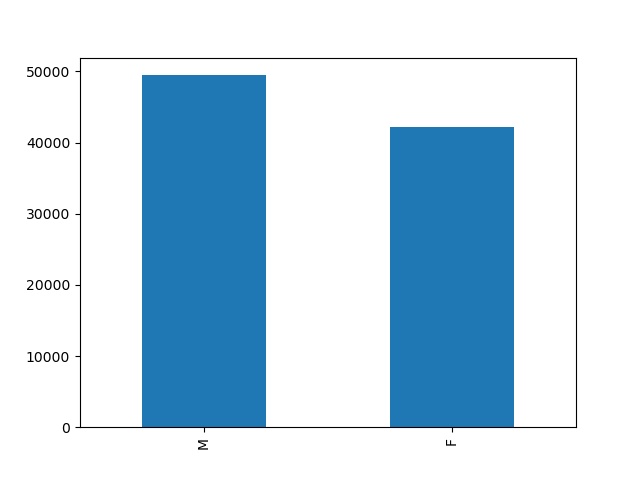
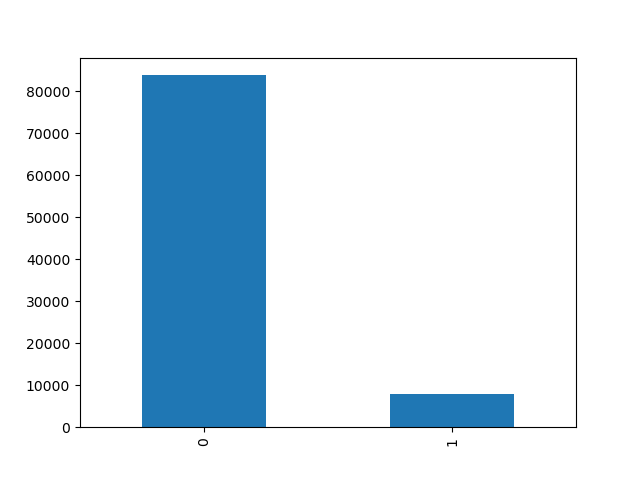
**Features**: 'encounter\_id', 'patient\_id', 'hospital\_id', 'age', 'bmi', 'elective\_surgery', 'ethnicity', 'gender', 'height', 'icu\_admit\_source', 'icu\_id', 'icu\_stay\_type', 'icu\_type', 'pre\_icu\_los\_days', 'weight', 'apache\_2\_diagnosis', 'apache\_3j\_diagnosis', 'apache\_post\_operative', 'arf\_apache', 'gcs\_eyes\_apache', 'gcs\_motor\_apache', 'gcs\_unable\_apache', 'gcs\_verbal\_apache', 'heart\_rate\_apache', 'intubated\_apache', 'map\_apache', 'resprate\_apache', 'temp\_apache', 'ventilated\_apache', 'd1\_diasbp\_max', 'd1\_diasbp\_min', 'd1\_diasbp\_noninvasive\_max', 'd1\_diasbp\_noninvasive\_min', 'd1\_heartrate\_max', 'd1\_heartrate\_min', 'd1\_mbp\_max', 'd1\_mbp\_min', 'd1\_mbp\_noninvasive\_max', 'd1\_mbp\_noninvasive\_min', 'd1\_resprate\_max', 'd1\_resprate\_min', 'd1\_spo2\_max', 'd1\_spo2\_min', 'd1\_sysbp\_max', 'd1\_sysbp\_min', 'd1\_sysbp\_noninvasive\_max', 'd1\_sysbp\_noninvasive\_min', 'd1\_temp\_max', 'd1\_temp\_min', 'h1\_diasbp\_max', 'h1\_diasbp\_min', 'h1\_diasbp\_noninvasive\_max', 'h1\_diasbp\_noninvasive\_min', 'h1\_heartrate\_max', 'h1\_heartrate\_min', 'h1\_mbp\_max', 'h1\_mbp\_min', 'h1\_mbp\_noninvasive\_max', 'h1\_mbp\_noninvasive\_min', 'h1\_resprate\_max', 'h1\_resprate\_min', 'h1\_spo2\_max', 'h1\_spo2\_min', 'h1\_sysbp\_max', 'h1\_sysbp\_min', 'h1\_sysbp\_noninvasive\_max', 'h1\_sysbp\_noninvasive\_min', 'd1\_glucose\_max', 'd1\_glucose\_min', 'd1\_potassium\_max', 'd1\_potassium\_min', 'apache\_4a\_hospital\_death\_prob', 'apache\_4a\_icu\_death\_prob', 'aids', 'cirrhosis', 'diabetes\_mellitus', 'hepatic\_failure', 'immunosuppression', 'leukemia', 'lymphoma', 'solid\_tumor\_with\_metastasis', 'apache\_3j\_bodysystem', 'apache\_2\_bodysystem', 'Unnamed: 83', 'hospital\_death'

## Description of Selected Features

In this section, we describe some interesting observations in the original dataset. First we show the histograms for some personal data of the patients, to get a good overview of the domain. We see that the people in the dataset are rather old, with the mean age at 62.3 years old. This can be explained by the fact that health generally decreases with age. Also the BMI with a mean at 29.2 is rather high, considering that a BMI above 25 is considered as being unhealthy.

The boxplot of the BMI also shows that the variance is quite high for the BMI.

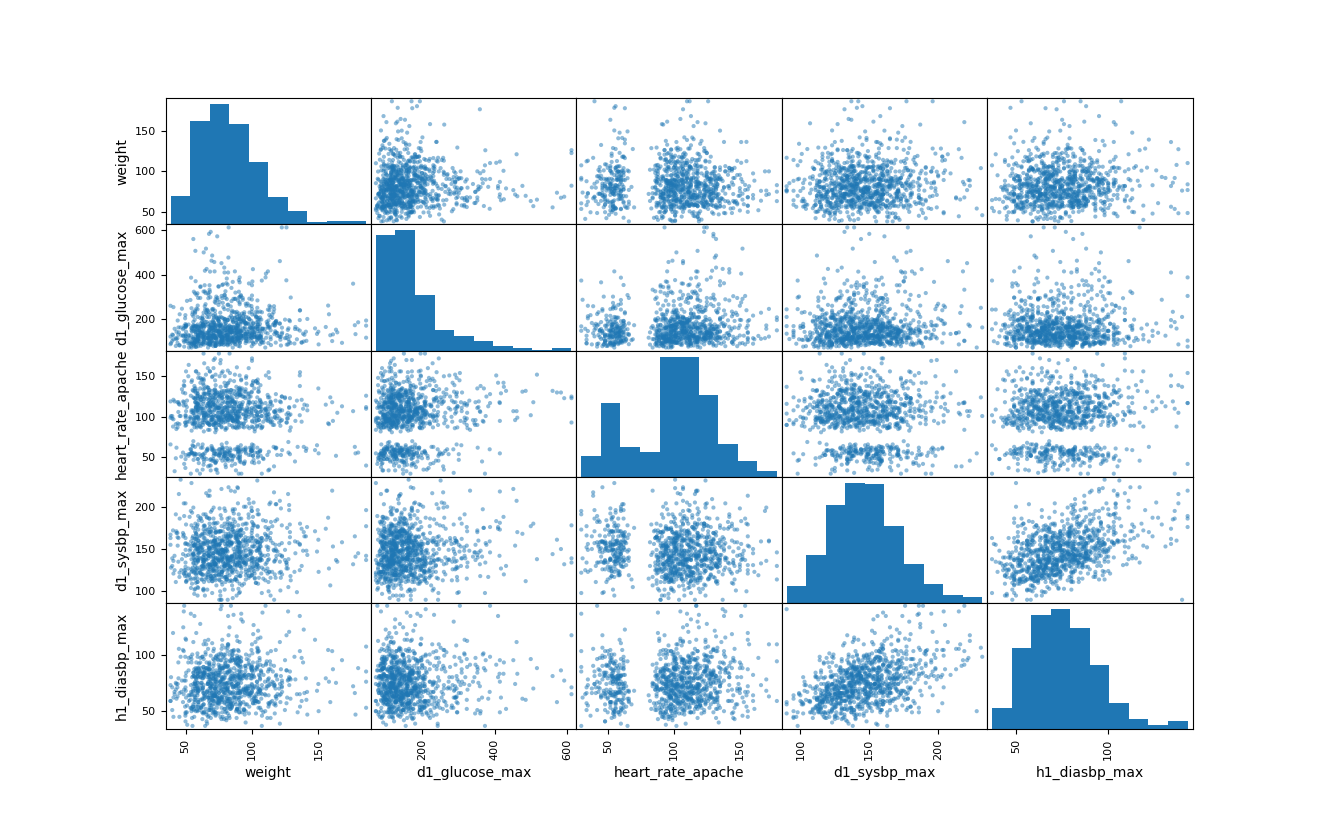
The distribution of the genders in patients tends slightly towards more male patients.

Finally we observe that the numbers of hospital deaths is 7915.

## Feature Correlation: Bivariate Analysis

In this section, we compare features by the means of a bivariate analysis and see how they are correlated.

The diastolic and systolic blood pressure correlate as can be seen in the graph below. The other features shown in the graph do not correlate in any significant way. However, interestingly there is a gap in the heart rate.



# **Description of the** P**re-processing of** the D**ata**

In this section, we describe the pre-processing steps that we have applied to our data set. The pre-processing of the data is a very important step in data mining pipeline, as the quality of the machine learning algorithm’s output highly depends on it’s input. Thus it is often a big part of any data mining project. We first do a feature selection and remove any columns that we will not need for further processing. Then we treat errors and outliers. Finally, we do data imputation to fill in missing features. The new pre-processed data set will then be used for the further processing of the data.

## Deletion of Columns

First, we noticed that the dataset contains redundant columns. Specifically, there is an empty column in the dataset. Since this column does not contain any information, we can discard it and delete it from the dataset.

The *id*’s do not contain any valuable information, since they are arbitrary and there is no correlation between *id* and any other feature. Therefore we also drop the columns encounter\_id and patient\_id. However, we keep the column with the hospital\_id, as it is possible that some hospitals have a high number of deaths. After the column deletions, we are left with 82 columns.

There are some columns representing a feature with very few occurrences, that might be too specific for the analysis of the whole population. We delete all columns with one hot encoding with less than 100 occurrences. To do this we simply sum up the number of occurrences. As only very few people have AIDS, we delete the corresponding column aids as it might cause unwanted interference in the machine learning algorithms. This leaves us with 81 columns.

## Outlier and Error Detection

In this section, we describe how we found outliers and errors in our dataset and how we treated them.

Since it is unclear whether our outliers are errors in the sampling of the data or if they are accurate measurements which happen to be anomalies, we decided to drop all the outliers and focus on the “normal” data. Although this will drastically reduce the size of our dataset and shrink our rows, we can afford this mesure due to the large size of our dataset. However, since our goal is to learn about hospital deaths, we do not want to delete too many rows with hospital\_death = 1, so we only delete rows with hospital\_death = 0.

Specifically we dropped every row that is an outlier in at least one column. We defined an outlier to be two standard deviations away from the mean.

Since we still have more than 26’691 rows in our dataset after dropping the outliers, we conform to the “10 times rule” that states that the number of rows should be at least 10 times higher than the number of columns, i.e. the number of parameters. In our case, with 81 columns we would need at least 8’100 rows to follow this rule.

## Data Imputation

In this subsection, we describe how we treated missing data. In order to impute missing data, it is necessary to understand the data domain, since the chosen imputation technique has implications for further data processing. Thus we need to decide how we treat missing data for each feature individually.

First we notice the correlation between height and weight. Thus we do a regression to fill in missing data for the columns height and weight. Since some rows are missing both values, we fill in the remaining ones with the mean. We then calculate the missing values for the column bmi according to the usual calculation of the body mass index.

The missing values for the remaining columns are filled in with the mean for the column. However, since we have some columns with one hot encoding, we do not want to fill in missing values for those columns with the mean. Thus we first convert those columns from 1/0 to True/False.

## Description of the Pre-processed Dataset

In this section, we describe the final dataset after preprocessing.

The final dataset after preprocessing contains no NaN for numeric columns and was reduced in both dimensions. It still is large enough for the machine learning algorithms we will apply on it in the following sections and due to the removed outliers, the dropped columns and the data imputation should yield better results than the original dataset.

| **Number of rows** | 26691 |
| --- | --- |
| **Number of columns** | 81 |
| **Integer / Boolean** | 57 |
| **Decimal** | 13 |
| **String** | 7 |
| **Other** | 4 |

# **Evaluation** C**riteria of** D**ata** M**ining** M**odels**

In this chapter, we will decide how the models are to be evaluated. We will use 30% of the dataset as test data, leaving the rest as training data. With a test dataset large enough the variance should not be affected.

To define a metric of evaluation, it was decided to use accuracy, as the objective of a predictive model is to make the correct prediction. The dataset has unbalanced classes, so accuracy is not the perfect metric, but it’s still a good metric nonetheless. This is due to the fact that the model can predict wrongly a minoritarian class and still achieve a good accuracy.

As to calculate the accuracy, we mainly used 10 fold cross validation which had good performance without too much computational cost.

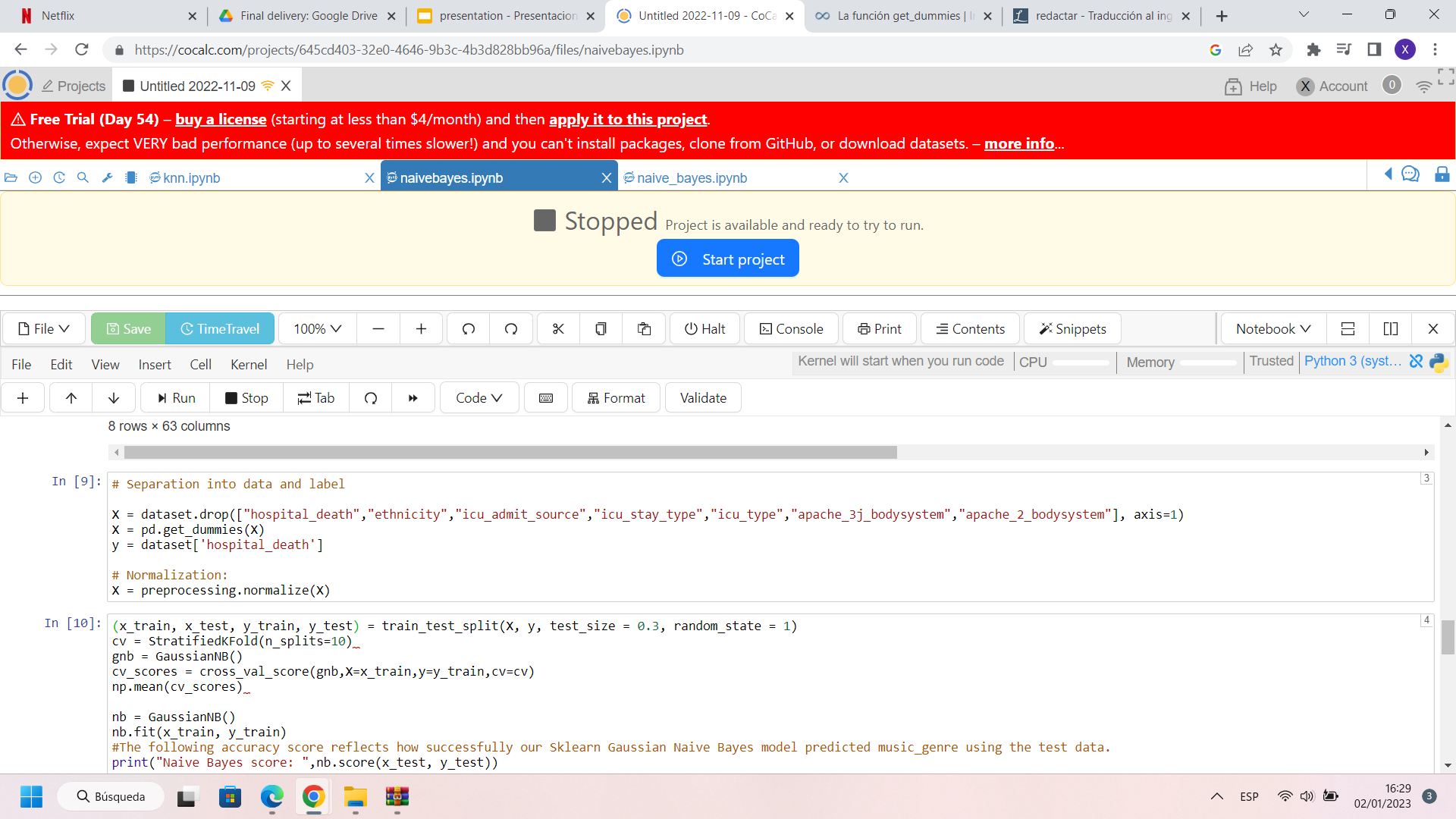
# **Execution of** D**ifferent** M**achine** L**earning** M**ethods**

In this section, we will write a report about the execution of different Machine Learning methods that we will be using with the target to continue studying them, get more familiar with them and discuss and compare their performance. Concretely, we will deal with Naïve Bayes, K-NN, Decision Trees, Support Vector Machines and Meta-learning algorithms, and we will focus on hospital\_death variable, which indicates whether the patient died during his hospitalization or not.

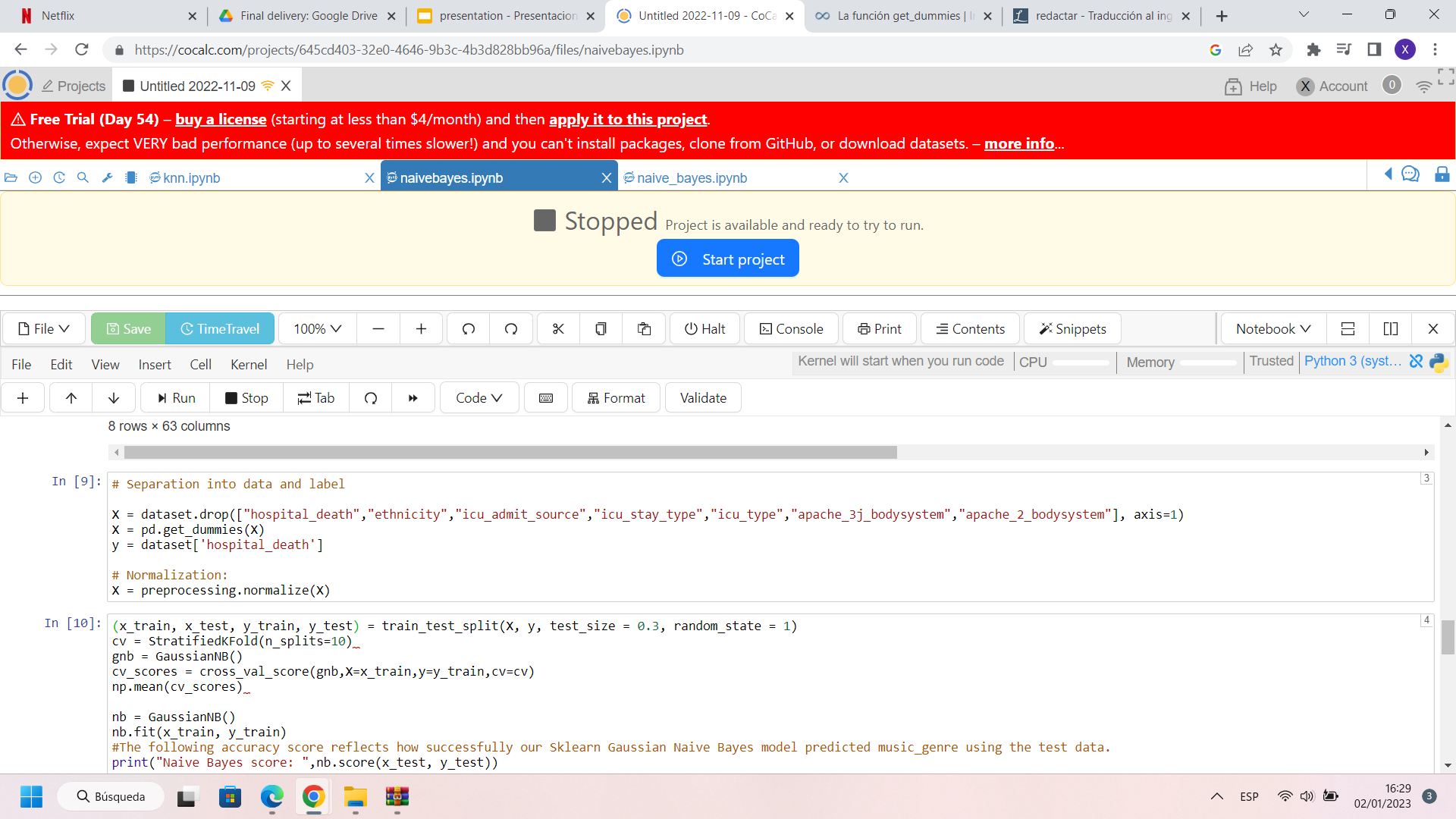
## Naïve Bayes

Naïve Bayes is a probabilistic classifier that is based on the theorem of Bayes and the assumption that the explanatory variables are independent. Bayes theorem is the following one: P(A|B) = P(B|A) \* P(A) / P(B), and it tells that the probability of event A occurring given that B is true equals to the probability of event B occurring given that A is true multiplied by the probability of A and divided by the probability of B. This theorem can be applied to obtain the probability of a label y given the values of explanatory variables X: P(y|x1,...,xn) = P(x1|y)\*...\*P(xn|y)\*P(y) / (P(x1)\*...\*P(xn)).

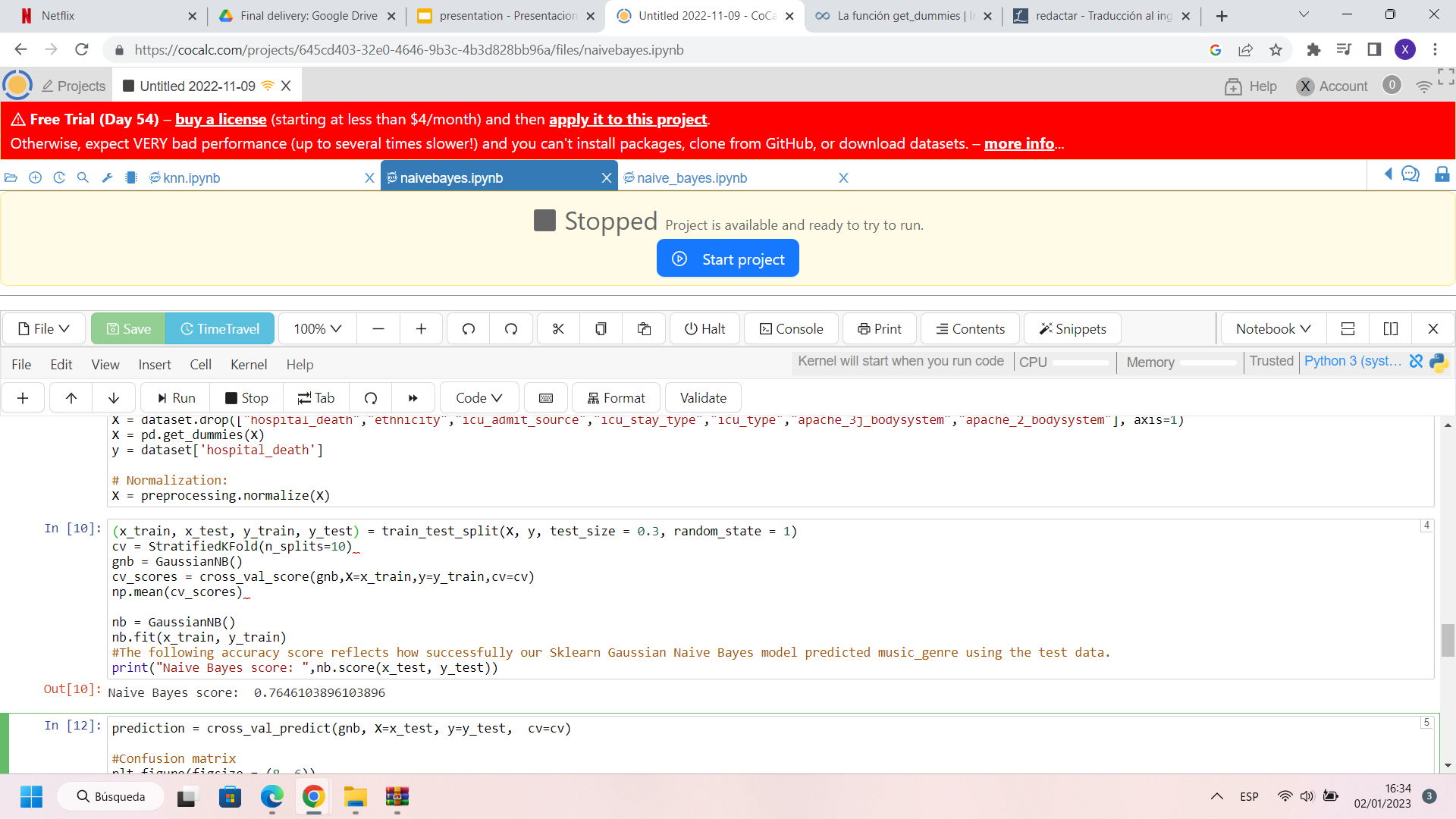
The first thing that we do is to divide our preprocessed dataset into the explanatory variables X and the response variable y. As we think that the variable gender can be important for the prediction and it is a categorical variable, we use the get\_dummies function from pandas to convert it to a numerical one.



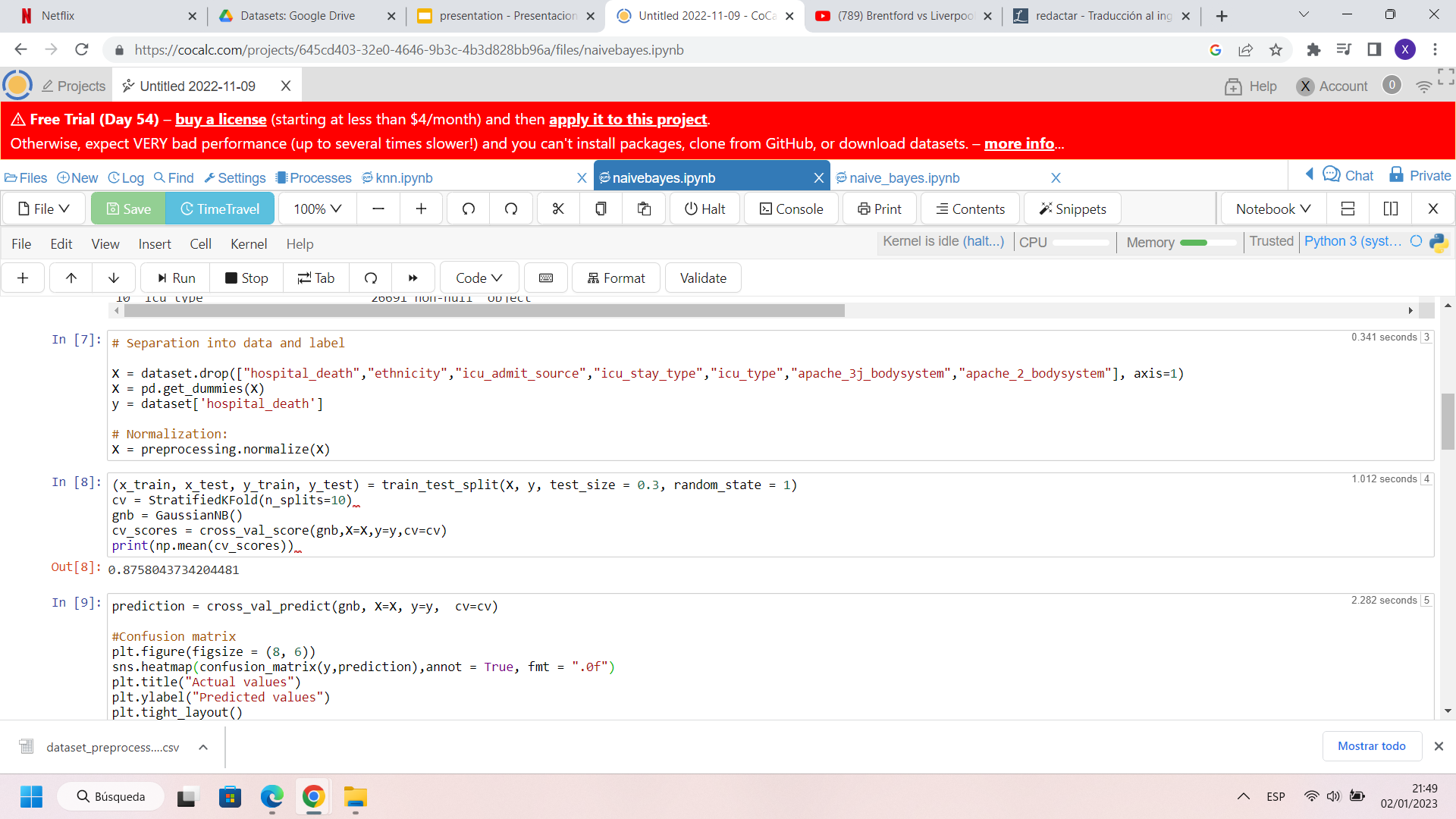
Then we normalize the explanatory data in order to be able to apply Gaussian Naïve Bayes correctly.



After that, we divide the data in a training dataset and a testing dataset so we can use one to train the model and the other to test how well it performs. Concretely, a 70% of the original is taken for training and the remaining 30% goes for testing, as it is usually done.

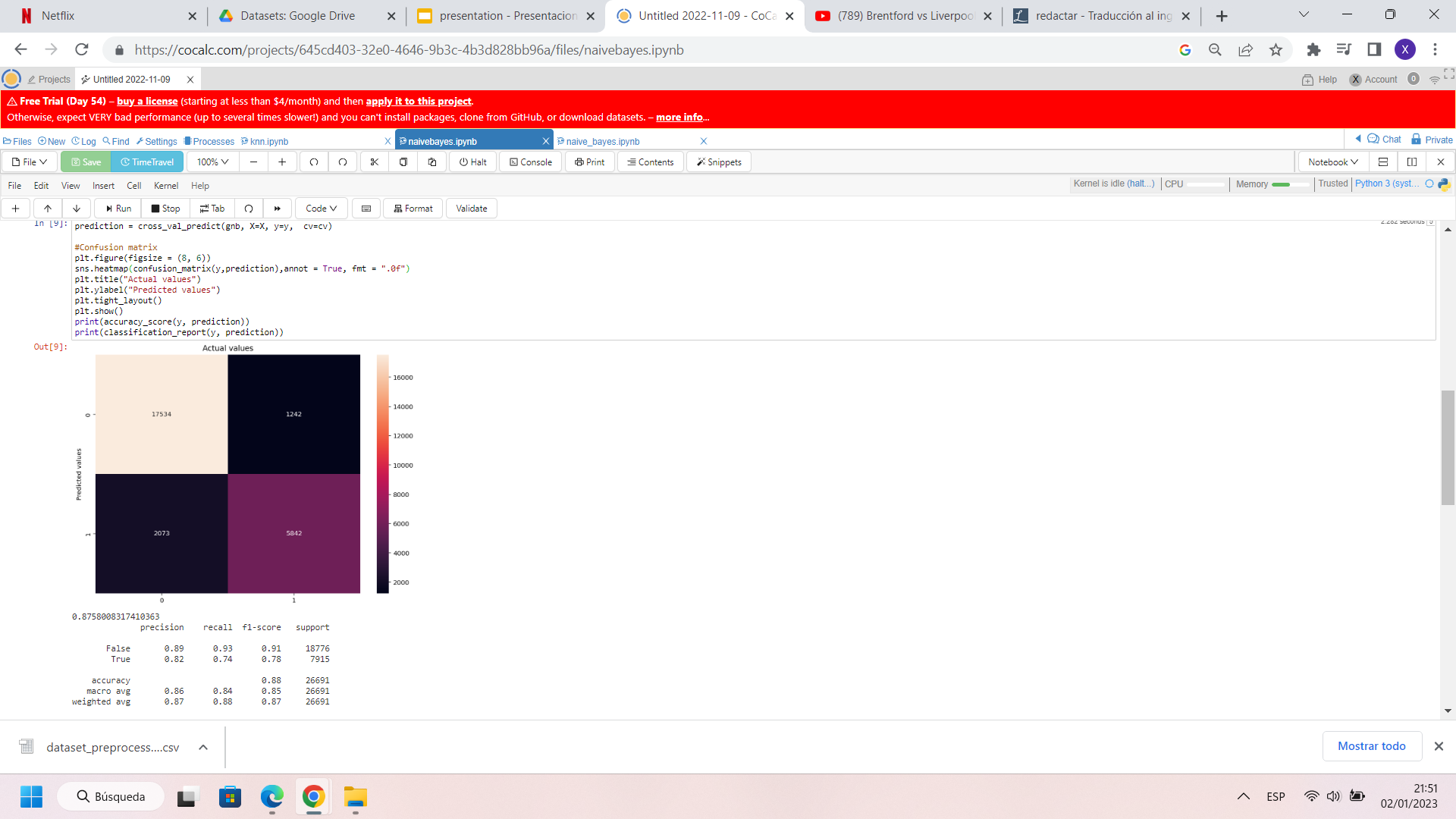


With cross validation we can use our training data in order to estimate how well our model will perform on data not seen before. Before using simple cross-validation, we check the performance of 10-fold cross validation with cross\_val\_score which divides the dataset in cv parts and uses one to test the model and the others to train it. So our dataset will be split in 10 parts, and one will be used to test the model and nine to train it.



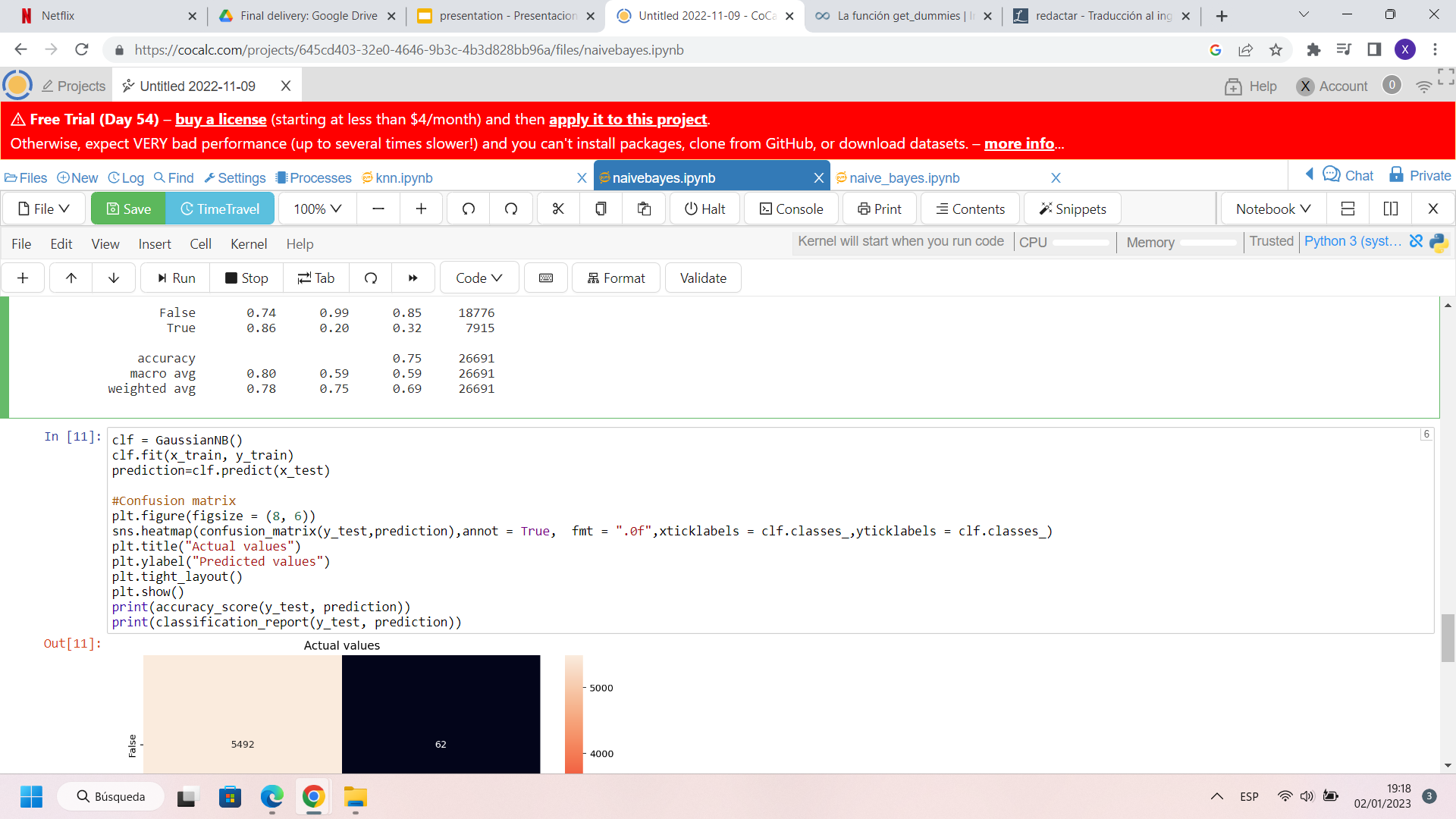
The mean of the accuracy of the 10 different models generated by the cross\_val\_score is 0.8758.

To finish, we use the cross\_val\_predict function to predict the response variable and discuss the goodness of the model.

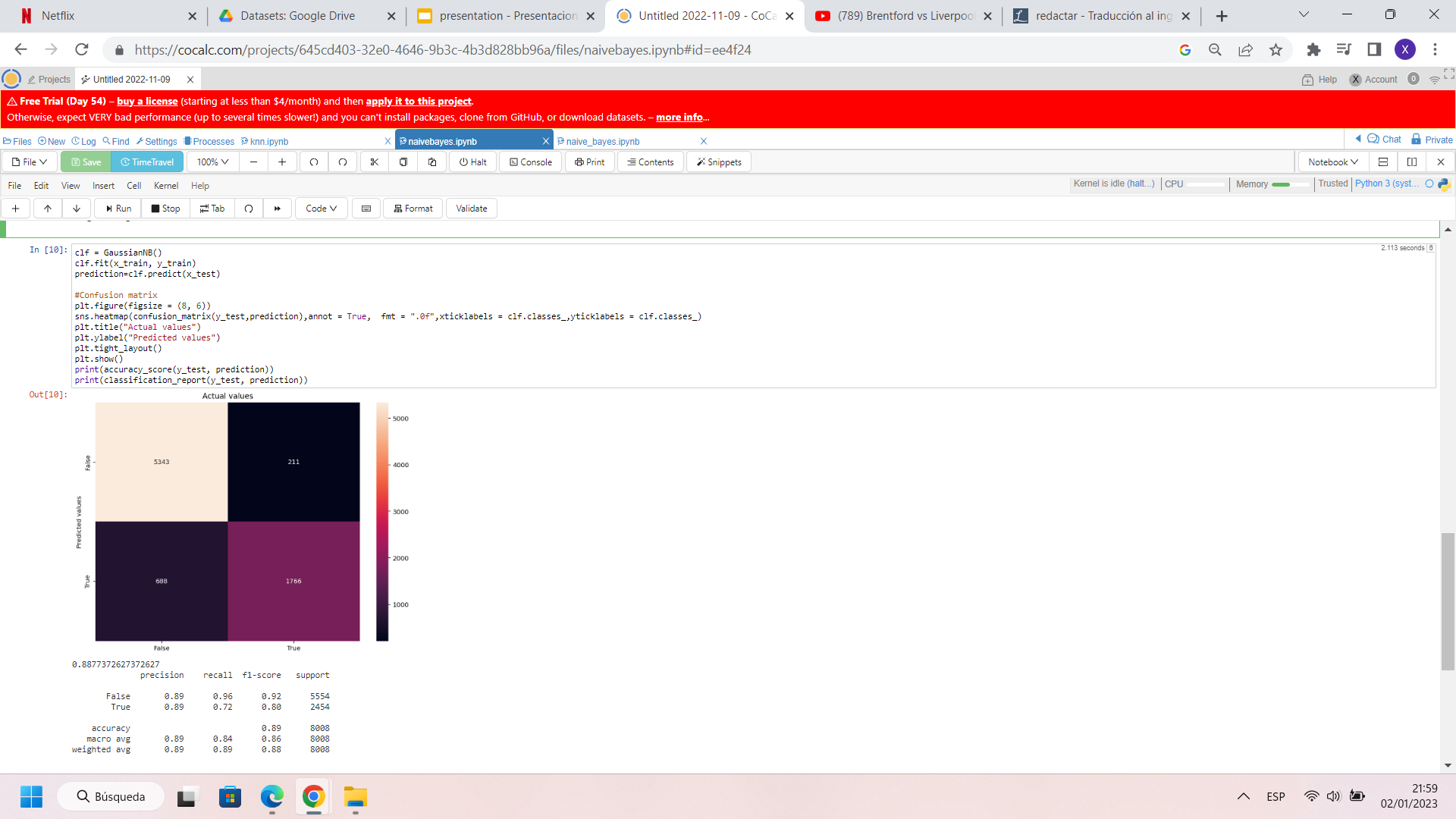


Precision equals to the number of true positives divided by the sum of true and false positives, so high values of precision mean that the values have been well predicted. On the other hand, recall equals to the number of true positives divided by the sum of true positives and false negatives, so low values mean that we have a lot of false negative predictions, something bad but it is not our case. As we can see, a lot of true negative cases are detected, and positives are also more or less well predicted.

Now we go with the simple cross-validation to see if we get even better results:



We get an accuracy of 0.8877 and this classification report:



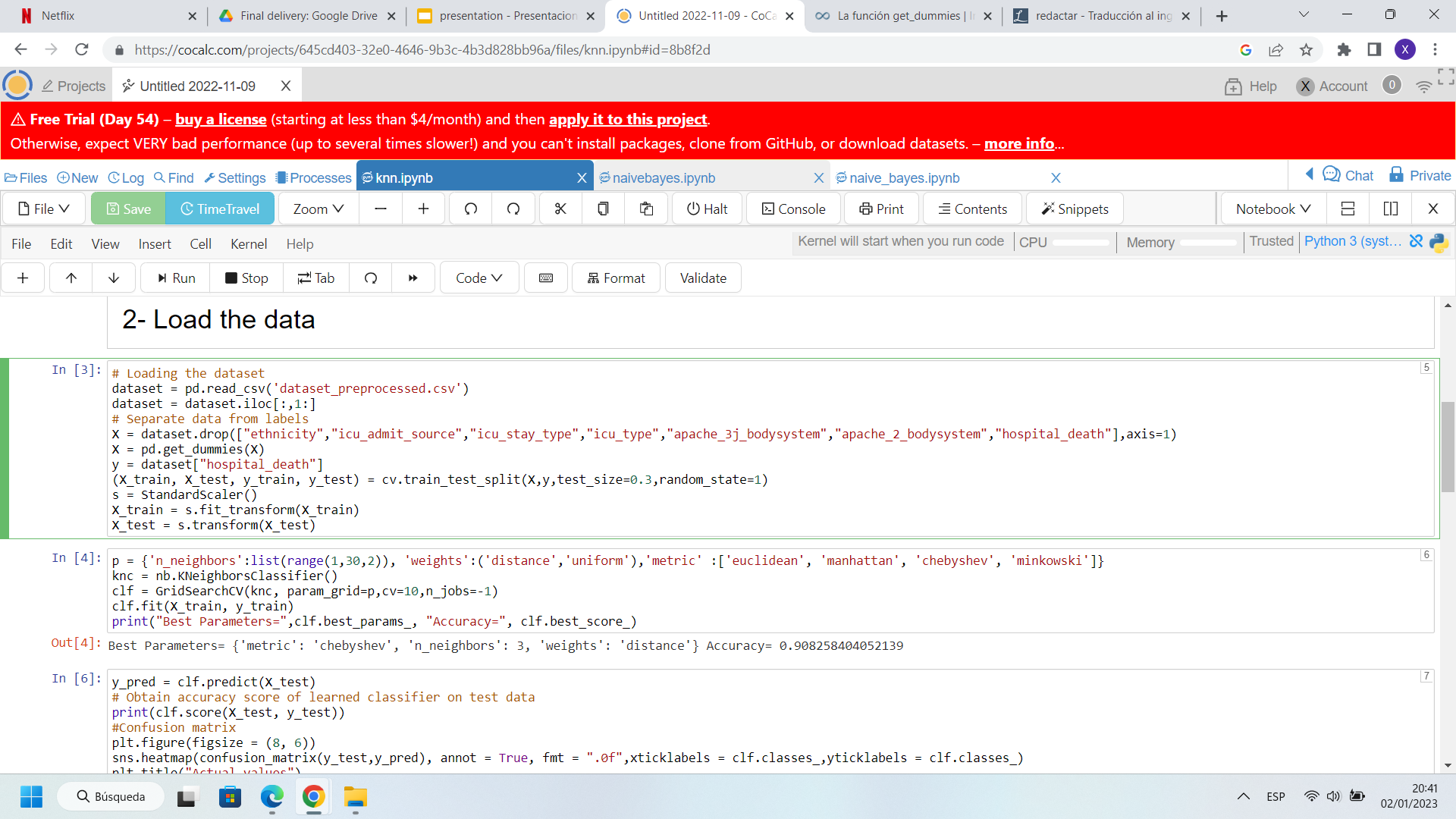
As f1 is the harmonic mean of precision and recall, high values indicate that the model is expected to perform well. As we can see, f1 has only increased one hundredth, as the confusion matrix is similar but with lower magnitudes.

In conclusion, Naïve Bayes is a classifier that offers good results despite assuming a false hypothesis as it is that the explanatory variables are independent. In our case we have obtained a very good model, a model that predicts well whether a patient with certain characteristics will have hospital death or not.

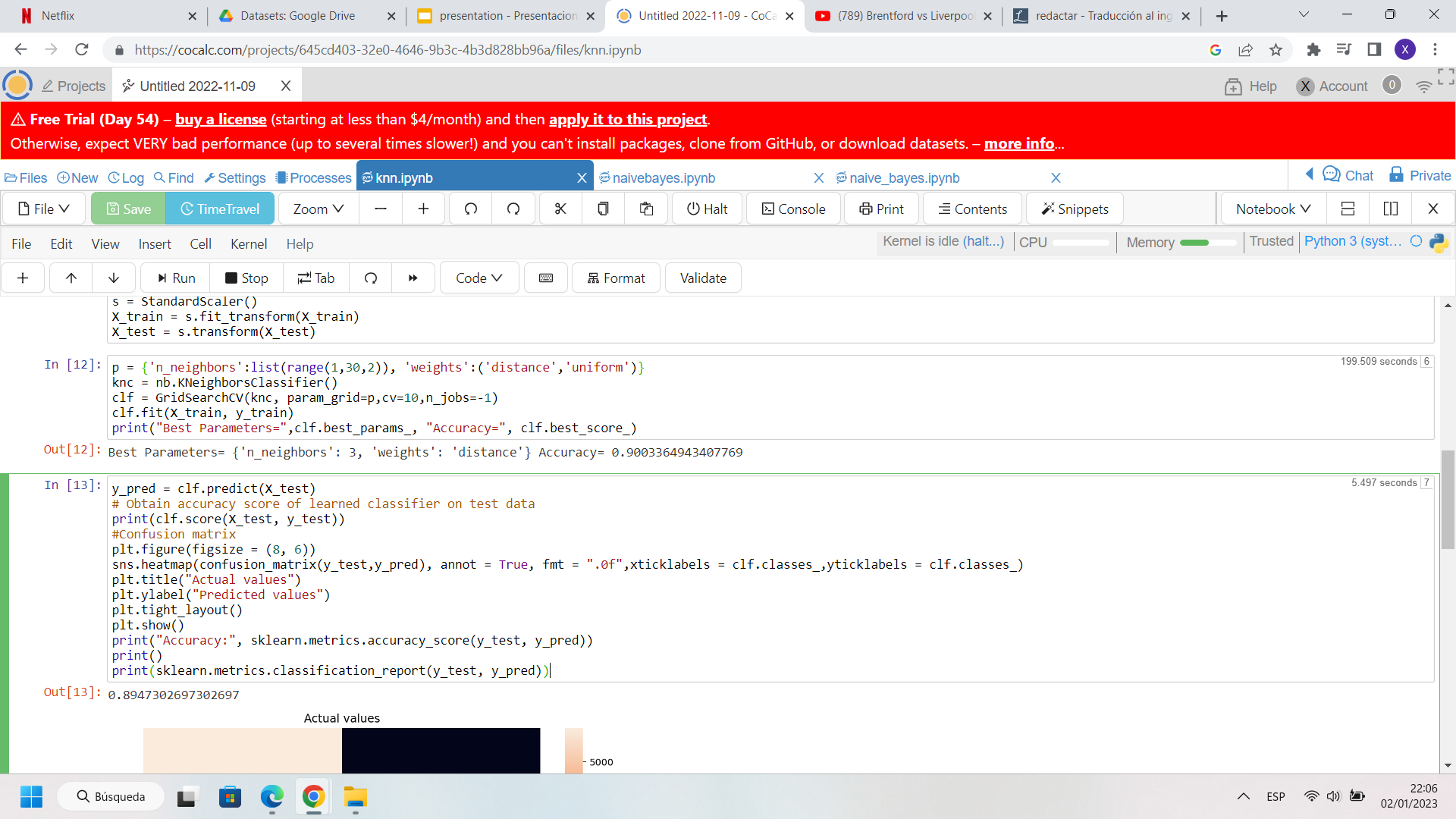
## K-NN

K-NN (K Nearest Neighbors) is an algorithm that is mainly used for classification too. It classifies the data points based on the classification of its neighbors and it can also take into account the distance between them to give more importance, more weight to those that are closer to the point being classified. Then we say that we are using the weighted K-NN.

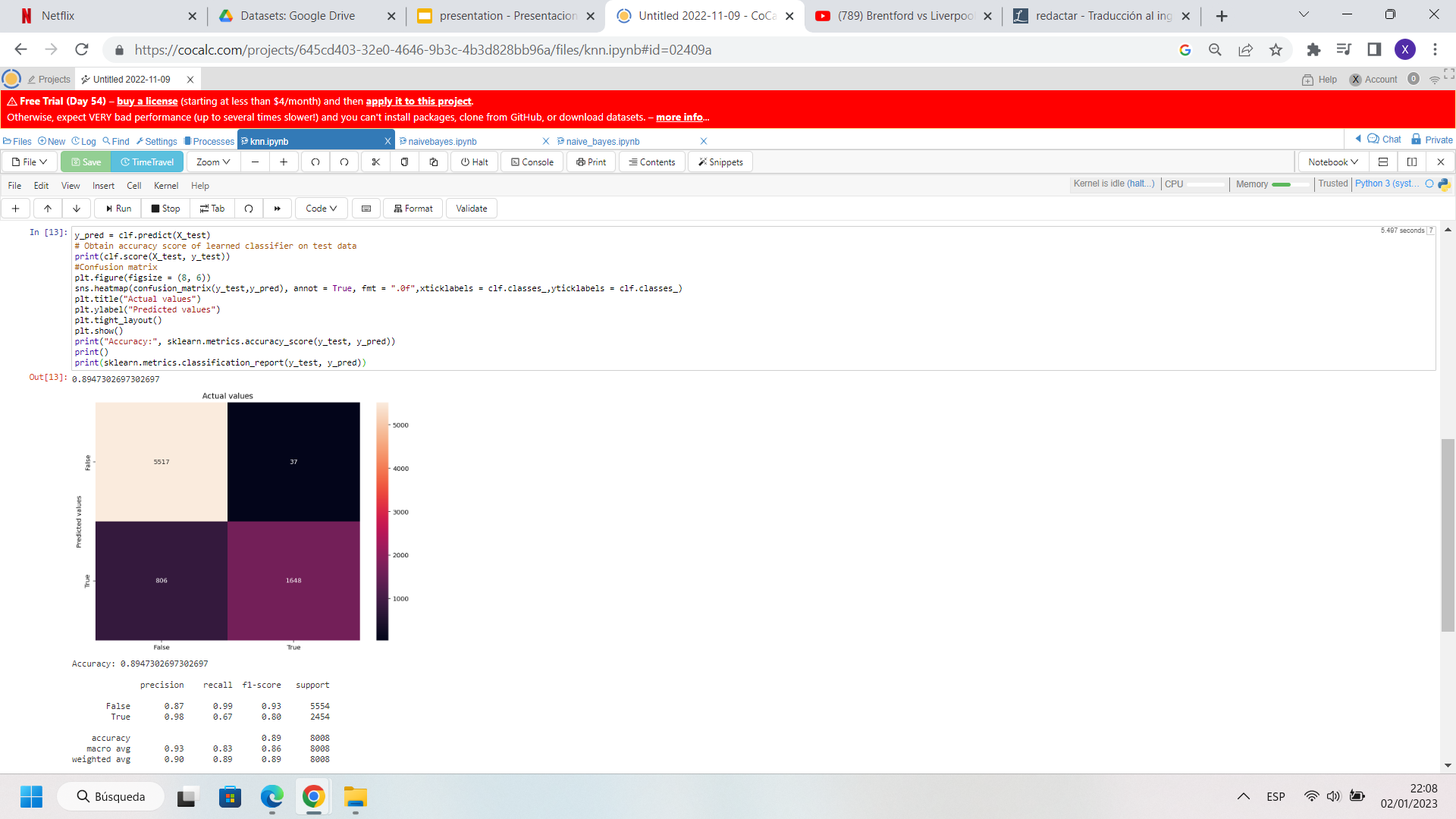
For this method, what we first do is to remove all the categorical variables in our preprocessed dataset because it can only work with numerical ones. Then, we divide the dataset into training (70% of the dataset) and testing (30% of the dataset) data, do feature scaling and use GridSearchCV function as 10-fold cross-validation in order to find the best parameters for knn.



The parameters of k-nn are two: k (number of neighbors to take into account), and weight.



As we can see, the best is to use weighted knn by distance, and a value for k equal to 3.

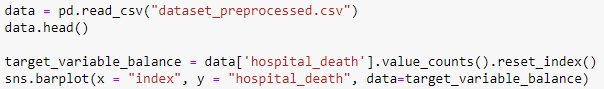


The confusion matrix obtained using those parameters shows an average accuracy of 0.8947. The performance of the model is good, as with the case of Naïve Bayes we achieve good precision and recall values for true and false, the only thing to say is that the number of false negatives could be lower.

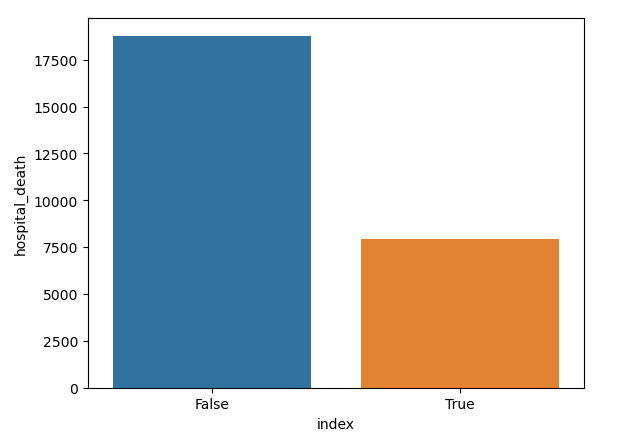
In conclusion, with k-nn we have obtained a model that performs so similar to the one created using Naïve Bayes, with the only difference that this time we have less false positives and more false negatives. One thing to mention too is that the knn method has had the disadvantage of being much less efficient, and after all we have not achieved a notorious improvement, something difficult because the performance was already really good.

## Decision Trees

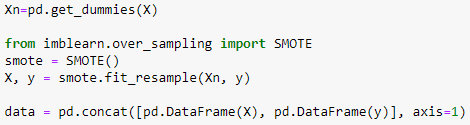
First task we need to execute to perform decision trees on our dataset is check if our data is unbalanced. To do this we simply check the difference between results on our target variable. The following code prints us a barplot to check the distribution of examples:



And the result barplot is the following:



As we can clearly see on the plot our dataset is an unbalanced one since the *True* results are less than half of the *False* ones. After checking for this information we decide to balance our dataset so our classifier works better. We decided to use an oversampling method, SMOTE (Synthetic Minority Oversampling Technique), that will help us to generate rows for the minority class. The following code oversamples our dataset:



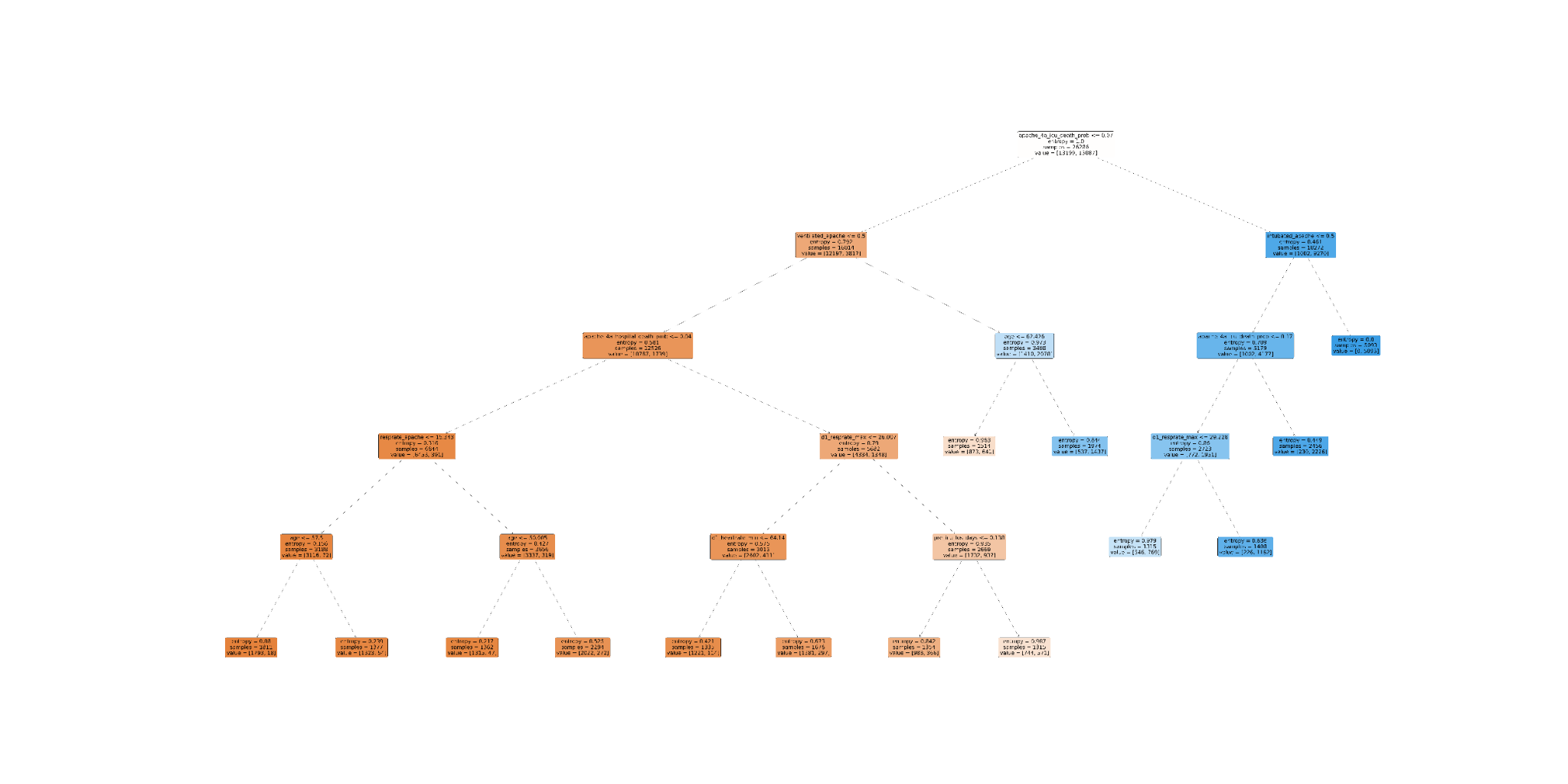
We then split our dataset into training and test and now we can start to execute decision tree learning. After executing the *sklearn* function *DecisionTreeClassifier* and getting an enormous tree that could not be read we decided to use *GridSearchCV* to choose the best parameters.

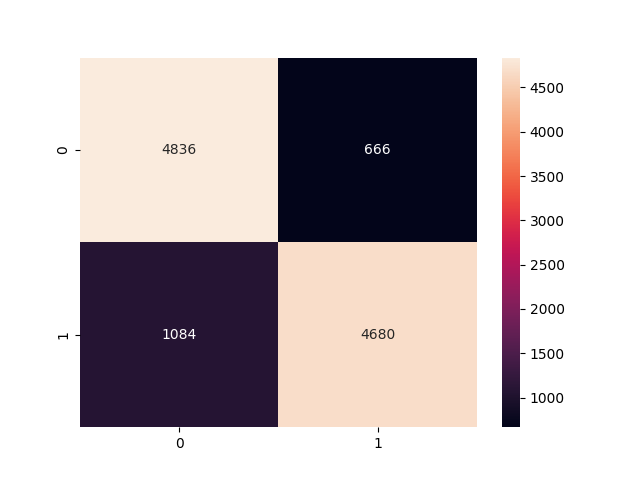
The first parameter we decide to parameterize is *criterion* that can be “gini” for the Gini impurity or “entropy” for the Shannon information gain. The second one *min\_samples\_split,* that are the minimum samples required to split an internal node, this parameter can help to not overfit the tree. Third one is *min\_impurity\_decrease* that helps reduce the splitting of the nodes if it decreases the impurity. And lastly we have *min\_samples\_leaf* that is the minimum number of samples required to split an internal node.

After executing the function we get for best parameters:

* criterion: entropy
* min\_samples\_split: 2
* min\_impurity\_decrease: 0.0
* min\_samples\_leaf: 4

Since the result tree with these parameters was too big to comprehend we decided to change the *min\_samples\_leaf* parameter to get a better look at the most significant variables. These were our results:



****

|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.82 | 0.88 | 0.85 | 5502 |
| **True** | 0.88 | 0.81 | 0.84 | 5764 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.85 | 11266 |
| **Macro Avg** | 0.85 | 0.85 | 0.85 | 11266 |
| **Weighted Avg** | 0.85 | 0.85 | 0.85 | 11266 |

Here we can see the resulting decision tree, the confusion matrix and the classification report. We finally got an accuracy of **0.849**, and we can see that the results are pretty good overall.

As we can check in the decision tree the most important variable in our dataset is *apache\_4a\_icu\_death\_prob*  and as we can see in the target variable (*hospital\_death*) values in the next depth of the tree, it does a pretty good job at separating the positive and the negative. The next most important variables would be *ventilated\_apache* and *intubated\_apache*. Through the tree we can see a good ratio between positive and negative examples on the tree leaves, which means the rules are working quite well to select a difference between classes.

## Support Vector Machine

Next we will be executing an analysis on our dataset with different kernels and parameters. We’ll start with the Linear SVM and see if we obtain good specs despite not using a Kernel. Then we will start implementing the SVM with kernels, first the Polynomial one, and then the default SKLearn version, the RBF Kernel.

### Linear SVM

First, we tried the Linear SVM with the default parameters, and we obtained the following accuracy and confusion matrix when used on the test set.

Accuracy = **0.9205**

| 0 | 5466 | 88 |
| --- | --- | --- |
| 1 | 548 | 1906 |
|  | 0 | 1 |

And, despite being the default parameters, the algorithm provided a high 0.92 of accuracy. With the number of supports: 3921 ( 3885 of them have slacks) and prop. of supports: 0.2098.

This process has a huge computational cost, so we adjusted the C parameter (used in the GridSearch) trying to reduce it. The value assigned is *Cs = np.logspace(-3, 5, num=9, base=10.0)*. Keeping the cross-validation value at 10 proved to be an unwise decision, as the process took even longer.

Next, we used the GridSearch algorithm to find the optimal value of C. After 24h of execution, the algorithm didn't finish, so we decided to execute it with a 3 fold cross validation for quicker performance, but after 12h hours it still didn’t finish. After that, it was obvious that with our possibilities Grid search algorithm was not viable with our dataset.

### 

### Polynomial Kernel

Our first execution of SVC with degree 2 with the default parameters had these confusion matrix and accuracy:

Accuracy: **0.9244**

| 0 | 5556 | 77 |
| --- | --- | --- |
| 1 | 528 | 1847 |
|  | 0 | 1 |

The default value for the *C* parameter is **1.0**, that is a regularization parameter. So we apply *GridSearchCV* to obtain the best value for this parameter. After the execution we get the best value *C*=**10.0**.

After executing SVC with the new *C* value we get the following results:

Accuracy: **0.9307**

| 0 | 5544 | 100 |
| --- | --- | --- |
| 1 | 455 | 1920 |
|  | 0 | 1 |

As we can see, there is not a huge difference between both values. The resulting model has 3234 supports (2343 of them have slacks), with a proportion of supports of 0.1731. With such a lower amount of supports, performance of the model is improved.

Next, we executed the same functions with degree 3, first with the default parameters:

Accuracy: **0.9286**

| 0 | 5556 | 77 |
| --- | --- | --- |
| 1 | 495 | 1880 |
|  | 0 | 1 |

This is the confusion matrix with *C* value at **1.0**. Now we execute *GridSearchCV* to find the best value: and it is once again *C=***10.0*.*** We execute with the new value and these are the results

:

Accuracy: **0.9334**

| 0 | 5513 | 120 |
| --- | --- | --- |
| 1 | 413 | 1962 |
|  | 0 | 1 |

And as we can see with the confusion matrix and the accuracy the results are, very similar with the change of the parameter. The resulting model has 3356 supports (1511 of them have slacks), with a proportion of supports of 0.1796.

### RBF Kernel

Finally, we used the radial basis function (RBF) with default parameters

Accuracy: **0.9360**

| 0 | 5512 | 42 |
| --- | --- | --- |
| 1 | 470 | 1984 |
|  | 0 | 1 |

It is observable that with default parameters it already has the best performance so far. Resulting in 3415

supports (3251 with slacks) and a proportion of 0.1827 supports. Afterwards we execute GridSearch again to find the optimal c and gamma. But like the linear kernel, the execution was too costly for our computers to handle.

## 

## Meta-learning algorithms

Next we will analyse the following four different Meta-learning algorithms: Voting Scheme, Bagging, RandomForest and Adaboost. The file that contains these scrips is named *“meta\_algorithms.ipynb.”*.

### Voting Scheme

With this method we will use all three classifiers explained before: Naïve Bayes, KNN and Decision Tree. We will start with the simple majority algorithm, and then we’ll check if a weighted majority gives us different values.

The first algorithm, Naïve Bayes, gave us an accuracy of **0.876**.

For the KNN method, we used the GridSearchCV function in order to find the best possible parameters, which turned out to be the next ones: **‘n\_neighbor’ = 9** and **‘weights = 'distance'**. The accuracy obtained, though, is not that great, **0.797**.

Finally, the Decision Tree classifier provided an accuracy of **0.878**, which is slightly higher than the one obtained with Naïve Bayes.

Once we have these results, we are free to start using meta-learning techniques.



**Majority Voting**

Accuracy = **0.901**

|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.88 | 0.99 | 0.93 | 5554 |
| **True** | 0.96 | 0.70 | 0.81 | 2454 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.90 | 8008 |
| **Macro Avg** | 0.92 | 0.85 | 0.87 | 8008 |
| **Weighted Avg** | 0.91 | 0.90 | 0.90 | 8008 |

For the weighted majority algorithm, we decided to give higher weights to Naïve Bayes and Decision Tree, and lower one to the KNN method. So the final weight distribution is the following:

| **Naïve Bayes** (accuracy = **0.876**) | **KNN** (accuracy = **0.797**) | **Decision Tree** (accuracy = **0.878**) |
| --- | --- | --- |
| **3** | **1** | **3** |



**Weighted Voting**

Accuracy = **0.906**

|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.89 | 0.99 | 0.94 | 5554 |
| **True** | 0.96 | 0.72 | 0.83 | 2454 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.91 | 8008 |
| **Macro Avg** | 0.93 | 0.866 | 0.88 | 8008 |
| **Weighted Avg** | 0.91 | 0.91 | 0.90 | 8008 |

And as the accuracy above shows, it is slightly better than using the simple majority algorithm.

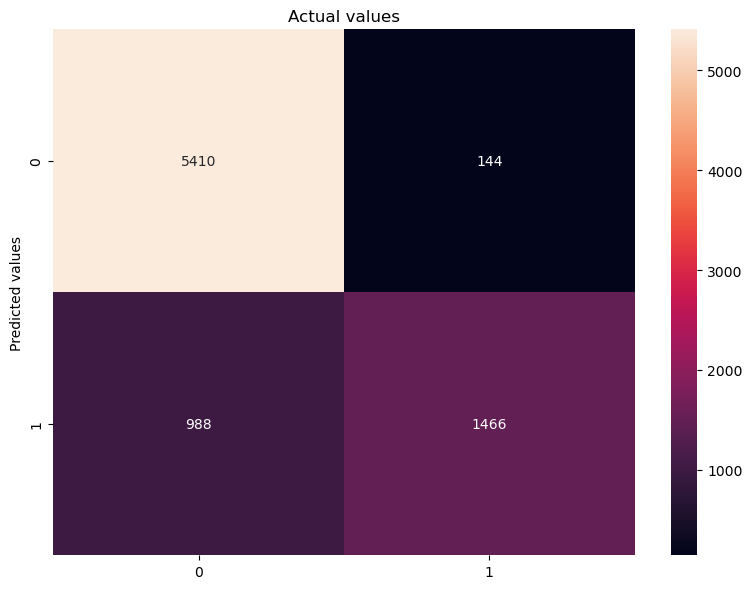
### Bagging

Next we’ll check how our classifiers do with the Bagging algorithm. The parameters used in this method are: **n\_estimators = [1, 2, 5, 10, 20, 50, 100, 200]** and **max\_features = 0.35**. In the case of the KNN algorithm, we reused the same parameters obtained while running the Majority Vote.

Here are the results obtained.

| **n\_estimators** | **Naïve Bayes** (accuracy) | **KNN** (accuracy) | **Decision Tree** (accuracy) |
| --- | --- | --- | --- |
| **1** | **0.876** | **0.777** | **0.812** |
| **2** | **0.876** | **0.796** | **0.843** |
| **5** | **0.876** | **0.810** | **0.891** |
| **10** | **0.876** | **0.810** | **0.906** |
| **20** | **0.876** | **0.812** | **0.912** |
| **50** | **0.876** | **0.814** | **0.921** |
| **100** | **0.876** | **0.815** | **0.924** |
| **200** | **0.876** | **0.815** | **0.924** |

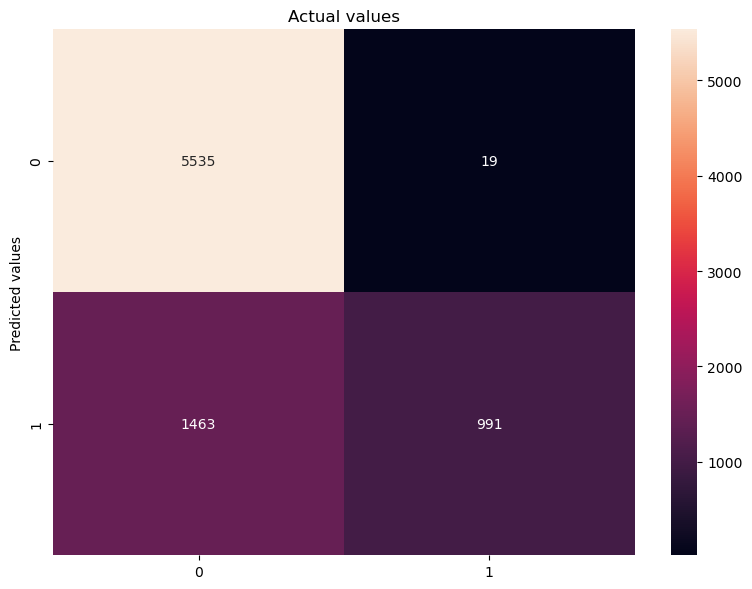
Next, we tested these algorithms with the test dataset (we decided to set the **n\_estimators** parameter to 100), and we obtained the following results:

**Naive Bayes**

Accuracy = **0.8586**

### 

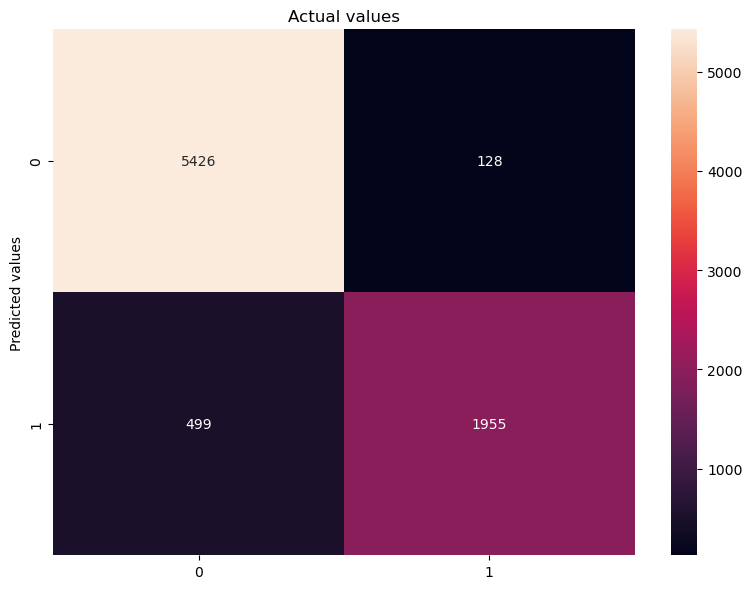
|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.85 | 0.97 | 0.91 | 5554 |
| **True** | 0.91 | 0.60 | 0.72 | 2454 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.86 | 8008 |
| **Macro Avg** | 0.88 | 0.79 | 0.81 | 8008 |
| **Weighted Avg** | 0.87 | 0.86 | 0.85 | 8008 |

**KNN**

Accuracy = **0.8149**

|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.79 | 1.00 | 0.88 | 5554 |
| **True** | 0.98 |  | 0.57 | 2454 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.81 | 8008 |
| **Macro Avg** | 0.89 | 0.70 | 0.73 | 8008 |
| **Weighted Avg** | 0.85 | 0.81 | 0.79 | 8008 |

**Decision Tree**



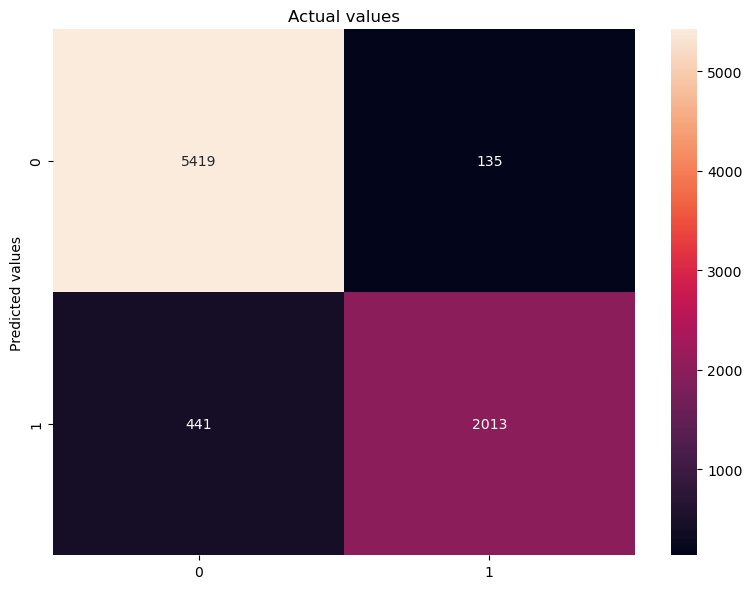
Accuracy = **0.9217**

|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.92 | 0.98 | 0.95 | 5554 |
| **True** | 0.94 | 0.80 | 0.86 | 2454 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.92 | 8008 |
| **Macro Avg** | 0.93 | 0.89 | 0.90 | 8008 |
| **Weighted Avg** | 0.92 | 0.92 | 0.92 | 8008 |

### 

### RandomForest

Now we will analyse our classifiers with the Random Forest algorithm. For this method we won’t change the parameters used in the Bagging algorithm, so **n\_estimators** will stay at **100**, and **max\_features** at **0.35**.

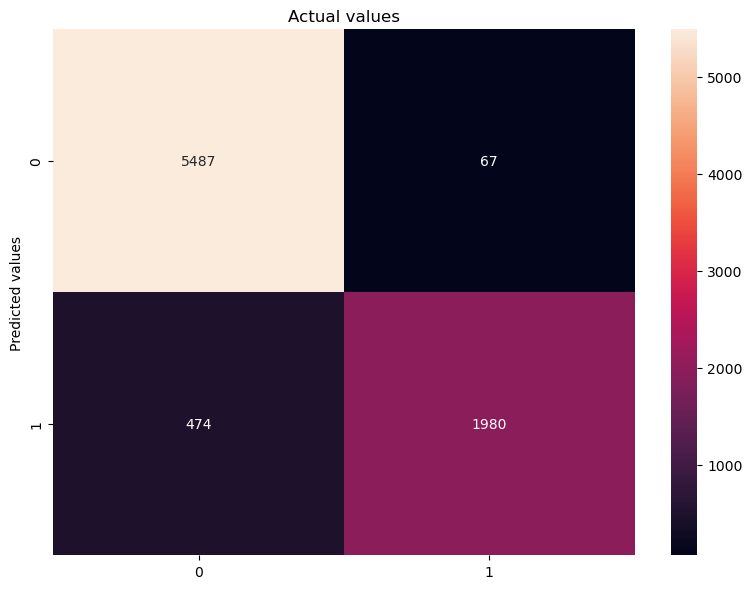


Accuracy = **0.9280**

|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.92 | 0.98 | 0.95 | 5554 |
| **True** | 0.94 | 0.82 | 0.87 | 2454 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.93 | 8008 |
| **Macro Avg** | 0.93 | 0.90 | 0.91 | 8008 |
| **Weighted Avg** | 0.93 | 0.93 | 0.93 | 8008 |

To check if the value of **n\_estimators** affects significantly at the results, we made the same analyses but changing this value in order to obtain its accuracy.

| **n\_estimators** | **Random Forest** (accuracy) |
| --- | --- |
| **1** | **0.849** |
| **2** | **0.870** |
| **5** | **0.904** |
| **10** | **0.916** |
| **20** | **0.924** |
| **50** | **0.927** |
| **100** | **0.928** |
| **200** | **0.928** |

We made the same tests using the ExtraTreesClassifier algorithm. Though this time we obtained a slightly better results, as the overall accuracy has increased from **0.928** to **0.934**.

Accuracy = **0.93244**

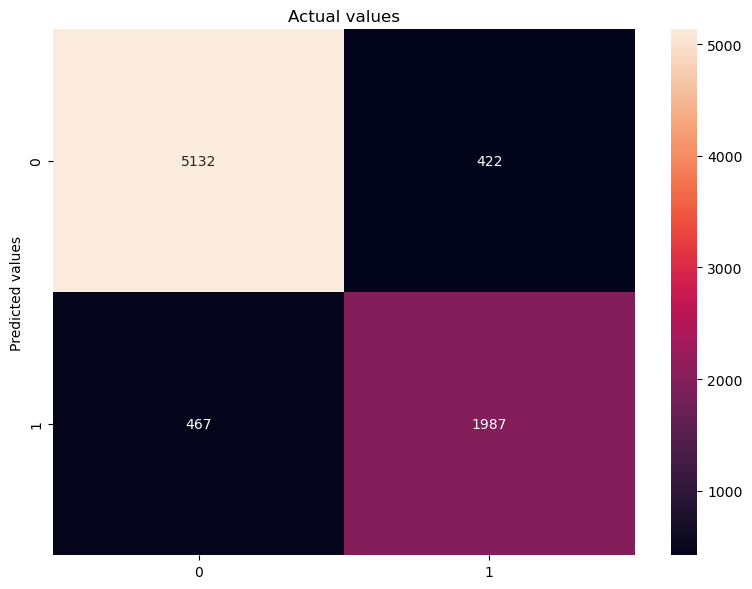
### 

|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.92 | 0.99 | 0.95 | 5554 |
| **True** | 0.97 | 0.81 | 0.88 | 2454 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.93 | 8008 |
| **Macro Avg** | 0.94 | 0.90 | 0.92 | 8008 |
| **Weighted Avg** | 0.93 | 0.93 | 0.93 | 8008 |

| **n\_estimators** | **Random Forest Extra** (accuracy) |
| --- | --- |
| **1** | **0.856** |
| **2** | **0.877** |
| **5** | **0.911** |
| **10** | **0.919** |
| **20** | **0.925** |
| **50** | **0.928** |
| **100** | **0.930** |
| **200** | **0.930** |

### Adaboost

Finally, we will check how our classifiers do with the Adaboost classifier.



Accuracy = **0.8889**

### 

|  | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| **False** | 0.92 | 0.92 | 0.92 | 5554 |
| **True** | 0.82 | 0.81 | 0.82 | 2454 |
|  |  |  |  |  |
| **Accuracy** |  |  | 0.89 | 8008 |
| **Macro Avg** | 0.87 | 0.87 | 0.87 | 8008 |
| **Weighted Avg** | 0.89 | 0.89 | 0.89 | 8008 |

| **n\_estimators** | **Random Forest Extra** (accuracy) |
| --- | --- |
| **1** | **0.889** |
| **2** | **0.900** |
| **5** | **0.908** |
| **10** | **0.912** |
| **20** | **0.912** |
| **50** | **0.910** |
| **100** | **0.903** |
| **200** | **0.900** |

### Conclusions Meta-Learning Algorithms

We definitely obtained decent results. The “best” algorithm for our dataset is clearly Random Forest, though the RandomForest Extra fits better, and the results are even better, having an overall accuracy of **93.2%**.

The Boosting algorithm (Adaboost) didn’t actually work out, and provided poor results compared to the others. The Bagging method didn’t fare much better and gave us low results for both KNN and Naïve Bayes algorithms, though Decision Tree’s results were quite better.

For the Voting Scheme, both Majority Voting and Weighted Voting provided almost similar results (**0.901** and **0.906** respectively).

# 

# Comparison and c**onclusions**

Although this project was centered in prediction models, preprocessing was still the most important and time consuming part. The dataset needed to be simplified because of an important number of outliers and very minoritarian classes in some specific attributes; this could have marred our results in a real world scenario, for example, with a patient with aids, but it made relationships between variables simple enough to be learned by the models used in the course. The dataset started being significantly unbalanced, but after the preprocessing the difference was reduced to a ratio barely below 1/2.

The prediction results were highly satisfactory. We can remark that even the baseline algorithms, such as K-NN and Naïve Bayes already did a good job with an accuracy slightly under 0.90. Even with Naive Bayes’ assumption of independent variables, which is specially wrong in medical circumstances. Decision Trees being the worst, had an accuracy of **0.849**, which is already outstanding.

Meta-learning algorithms gave even better results, surpassing the 0.90 mark. Achieving an accuracy of **0.932** in the case of Random Forest, being the best one. Adaboost, on the other hand, didn’t have such a good performance, as it is made to improve weak learners, and all our models were already pretty good.

With such a big dataset, such high accuracy is expected, but it comes with an equally high computational cost. Two grid search algorithms weren’t able to finish its execution during SVM in 12h with worse versions of the cross validation algorithms. Even with default values, the model with the highest accuracy was the SVM with RBF kernel, with a score of **0.9360.**

# Attachments

In the attachments we provide the following files with the code to obtain our results.

| **Name of the file** | **Section** |
| --- | --- |
| bivariate\_analysis.py | Description of the Original Dataset |
| boxplot.py | Description of the Original Dataset |
| create\_df\_corr.py | Description of the Original Dataset |
| top\_features.py | Description of the Original Dataset |
| dataset.csv | Description of the Original Dataset |
| df\_corr.csv | Description of the Original Dataset |
| df\_des.csv | Description of the Original Dataset |

| **Name of the file** | **Section** |
| --- | --- |
| dataset\_preprocessed\_des.csv | Description of the Pre-processing of the Data |
| dataset\_preprocessed.csv | Description of the Pre-processing of the Data |
| deleting\_columns.py | Description of the Pre-processing of the Data |
| error\_and\_outliers.py | Description of the Pre-processing of the Data |
| missing\_data.py | Description of the Pre-processing of the Data |

| **Name of the file** | **Section** |
| --- | --- |
| knn.ipynb | Execution of different machine learning methods |
| naivebayes.ipynb | Execution of different machine learning methods |

| **Name of the file** | **Section** |
| --- | --- |
| SVM.ipynb | Execution of Support Vector Machines |

| **Name of the file** | **Section** |
| --- | --- |
| meta-methods.ipynb | Execution of meta-learning algorithms |