# [NÚMERO NO SE QUE] Predicciones

Tras realizar la limpieza de datos inicial en el apartado anterior, se realizaron los entrenamientos necesarios para poder generar los distintos modelos, y así, predecir con los algoritmos que se mencionarán más adelante.

La idea es poder ser capaces de predecir si una persona padecerá de recidiva bioquímica (RBQ) de cáncer de próstata, a partir de unos datos. La recidiva bioquímica consiste en la repetición de una enfermedad después de haberse recuperado de dicha enfermedad.

Para ello, se ha utilizado un notebook (adjuntos en el anexo de esta memoria, y con el proyecto como fichero aparte) con la limpieza de datos para realizar el análisis de los datos y las distintas pruebas; para así, posteriormente, copiar y pegar el código necesario del notebook en el código que contiene el *backend* de la aplicación.

Para realizar el análisis de los datos se ha utilizado la librería Pandas, y para generar los modelos y realizar las predicciones se ha utilizado scikit-learn, una librería de *Machine Learning* muy sencilla de utilizar. MENCIONAR QUE LOS DATOS CATEGÓRICOS SON NUMÉRICOS… QUE MIREN EL WORD CON EL *MAPEO. QUIZAS SE MENCIONE EN LA PARTE DE “LIMPIEZA”*

## Descripción inicial de los datos

En primer lugar, se visualizaron los datos para tener una idea inicial de cómo es el dataset. Haciendo **df.shape** se obtiene el número de filas y de columnas del dataset tras la primera limpieza previa.

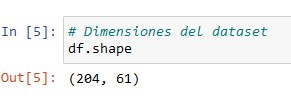


Figura 1. Forma del dataset tras la primera limpieza

Se ha observado que el dataset tiene 204 filas (es decir, 204 casos de seguimiento de cáncer de próstata) y 61 columnas (o variables). Viendo esto, se puede apreciar la gran escasez de datos para realizar el entrenamiento. Al ser casos muy extraños, no existe ninguna forma de obtener más datos.

También ha utilizado **df.describe(include="all",datetime\_is\_numeric=True)** para ver un resumen estadístico.

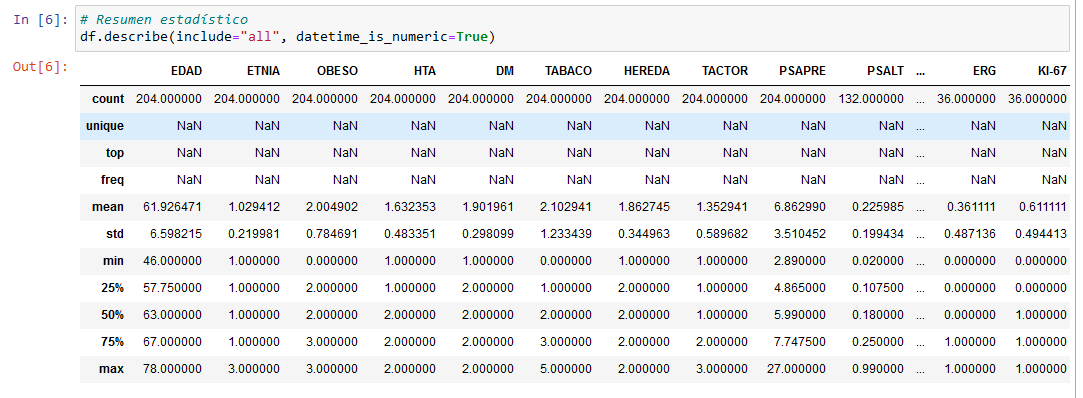


Figura 2. Resumen estadístico

Con este resumen estadístico se ha podido ver que, por ejemplo, la edad media de los pacientes es de unos 62 años, que el más joven tiene 46 años, y que el más mayor tiene 78 años. También aparecen las estadísticas de otras columnas como PSAPRE, PSALT, etc. Explicar e interpretar estos tecnicismos médicos se alejan del ámbito de este Trabajo Final de Grado.

## Limpieza de columnas con filas vacías

Una cosa que se tuvo muy en cuenta (ya que es necesario para poder realizar los entrenamientos), es no tener ninguna columna con datos nulos. Para detectar estos nulos se ha utilizado **df.info().**

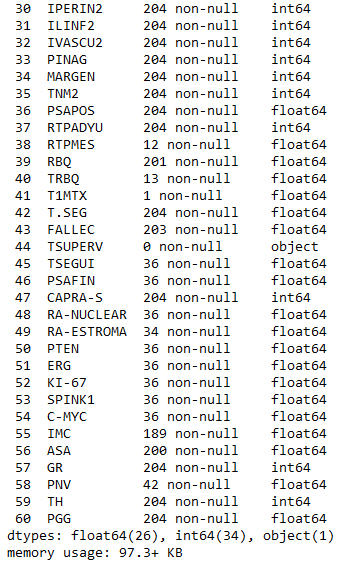
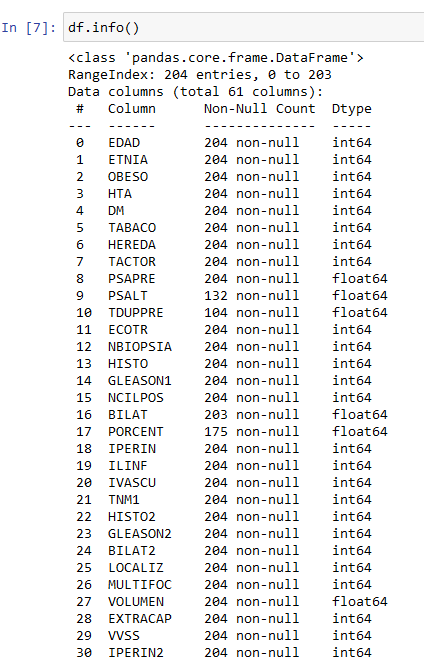


Figura 3. Visualización para localizar columnas vacías

Como se puede observar en las capturas anteriores, hay algunas columnas con entradas nulas. Algunas ni siquiera llegan a las 100 entradas no nulas, por lo que fue necesario hacer más limpieza para no tener ninguna columna con datos vacíos, y que scikit-learn pueda hacer sus entrenamientos correctamente. Para hacer esta limpieza, se rellenaron aquellos nulos con la mediana de los datos de cada columna.

Antes de hacer el rellenado, se eliminaron aquellas columnas con más del 50% de filas vacías, ya que rellenar con la mediana una columna prácticamente vacía sería falsear demasiado el dataset, haciendo que las predicciones no sean certeras y reflejen una realidad que no es. A continuación, se muestra la función para eliminar las columnas casi vacías.

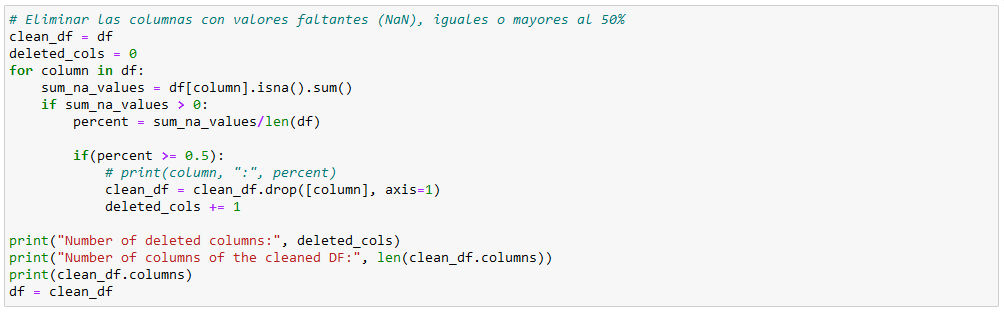


Figura 4. Función para eliminar columnas con valores faltantes mayores o iguales al 50%

Una vez eliminadas las columnas casi vacías, se rellenaron las demás columnas con la mediana a través de la siguiente función:

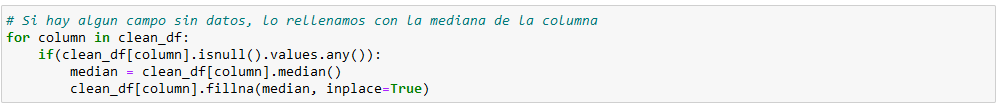


Figura 5. Función para rellenar columnas con filas vacías con la mediana

Una vez rellenadas las columnas se puede confirmar que, efectivamente, no hay ninguna columna con filas vacías.

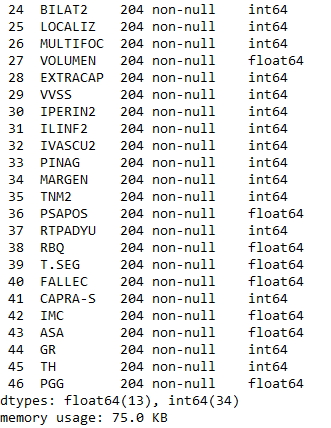
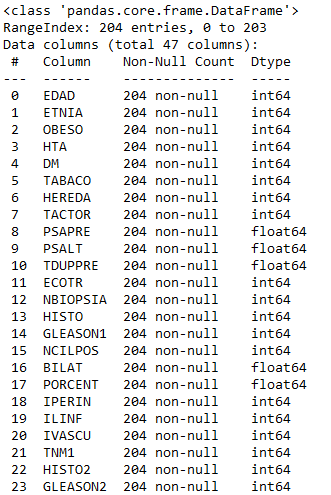


Figura 6. Confirmación de que todas las columnas están totalmente rellenas

Tras esta limpieza, se puede comprobar que todas las columnas están rellenas. Sin embargo, en vez de tener 61 columnas, el dataset tiene 47 columnas. 14 columnas tenían más del 50% de filas vacías.

## Visualización general y estudio de los datos

Después de detectar y limpiar aquellas columnas con datos vacíos, se visualizaron los datos a través de histogramas para entender un poco mejor el conjunto de datos.

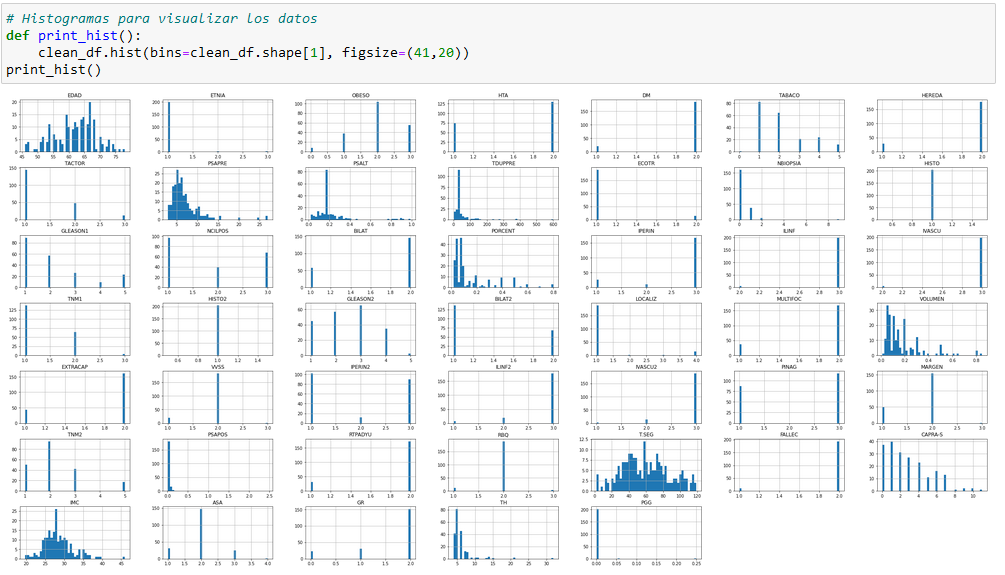


Figura 7. Histogramas del dataset

Tras visualizar estos datos, se puede estudiar cada columna de manera más sencilla. A continuación, se detallará el análisis realizado sobre algunas columnas cuyo estudio es interesante:

* **ETNIA.** Esta columna no es muy relevante a la hora de predecir la recidiva bioquímica. Además, estos estudios se realizaron en España, por lo que la mayoría de la población es caucásica, y no aportaría mucha información. Puede producir *overfitting* si dejamos esta columna, ya que el hecho de ser caucásico influenciaría demasiado sobre la predicción.
* **IPERIN.** Hay tres categorías (“si”, “no”, “nc”). La mayoría de los casos no se conocen (“nc”). Al tener menos de 30 casos conocidos (sobre los más de 200), no tiene mucho sentido utilizar esta variable, ya que solo añadiría ruido a los modelos.
* **LINF, LINF2, IVASCU e IVASCU2.** Hay tres categorías (“si”, “no”, “nc”). La mayoría de los casos no se conocen (“nc”). No tiene mucho sentido utilizar estas variables, ya que, al igual que el caso anterior, solo añadirían ruido a los modelos.
* **FALLEC.** Esta variable se tiene que eliminar, ya que el fallecimiento no influye en nada en la predicción de la recidiva bioquímica. Además, el fallecimiento es una consecuencia de la recidiva, no una causa, por lo que usarlo en los modelos no tiene ningún tipo de sentido.
* **IPERIN2.** Hay tres variables (“si”, “no”, “nc”). La mayoría de la población está repartida entre el “sí” y el “nc”, siendo el “sí” el que tiene mayor peso. Para este caso se ha decidido reasignar todos los casos de “nc” al “si” o al “no” de manera aleatoria.

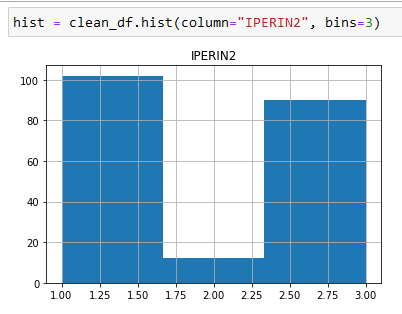
****

Figura 8. Casos de IPERIN en el dataset

Para realizar la reasignación aleatoria de los “nc”, se ha utilizado la función descrita a continuación:



Figura 9. Función para reasignar los casos de "nc"

Una vez realizadas las reasignaciones, se obtuvo un nuevo IPERIN2 sin los casos “nc”, tal y como se muestra en el histograma de la *Figura 10*. Nótese que, al ser aleatorio, el histograma no será el mismo en todas las ejecuciones, pero en la mayoría de casos habrá un mayor balance entre el “si” y el “no”.

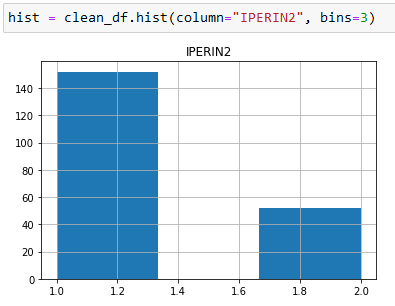


Figura 10. IPERIN2 tras las reasignaciones aleatorias

* **RBQ.** Esta es la variable a predecir. Tiene tres categorías: “SI (CASOS)”, “NO (CONTROLES)” y “PERSISTENCIA PSA”. Después de realizar el análisis de esta columna, se preguntó al codirector del proyecto sobre la relevancia de “PERSISTENCIA PSA”, ya que son casos muy escasos (menos de 10 casos). La respuesta fue que la “PERSISTENCIA PSA” no tiene mucho impacto sobre la RBQ y que se puede descartar. Nuestro predictor se convierte entonces en un clasificador binario, el cual se limita a predecir “SI” o “NO”.

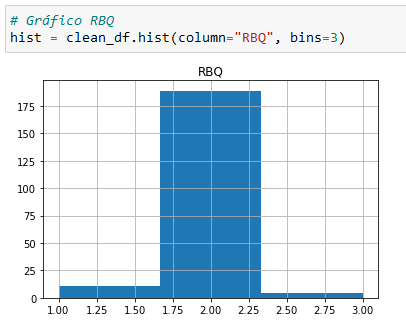


Figura 11. Histograma de RBQ (con PERSISTENCIA PSA)

La función para eliminar los casos de “PERSISTENCIA PSA” mediante reasignaciones a “SI” o “NO” de manera aleatoria de describe en la captura siguiente:

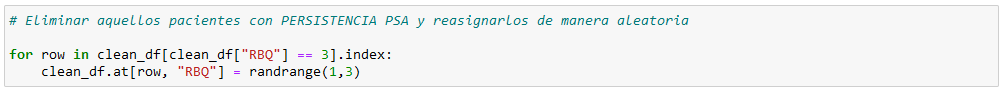


Figura 12. Función para reasignar los casos de PERSISTENCIA PSA

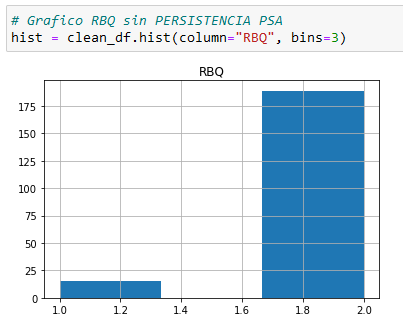


Figura 13. Histograma de RBQ (sin PERSISTENCIA PSA)

Observando el histograma, se puede comprobar que esta variable está muy desbalanceada, dando mayor peso al “NO”. Este desbalanceo y la escasez de datos limita bastante la capacidad predictiva de los modelos, y no se puede hacer mucho para mejorar dicha capacidad, ya que solo se disponen de los más de 200 (y escasos) casos de cáncer de próstata en seguimiento.

Además, a través de estos histogramas se puede saber de antemano que los modelos estarán muy sesgados y que en la mayoría de las predicciones el resultado será “NO”. Existe una probabilidad muy alta de que existan muchos falsos negativos de casos con recidiva bioquímica.

En el apartado siguiente se mostrarán los pasos utilizados para poder obtener modelos de predicción con la mayor efectividad y certeza posible.

## Generación de los modelos de predicción

Los modelos de predicción que se utilizan actualmente en la unidad de urooncología se generan a partir de regresiones lineales y de curvas de supervivencia (o regresiones de Cox). Para este TFG, se han planteado utilizar distintos modelos de *Machine Learning* para realizar las predicciones, entre las cuales se encuentran *Random Forest*, *K-nearest neighbours* y regresión logística, cuyos detalles se explicarán más adelante.

A continuación, se presentará el proceso seguido para generar los distintos modelos de predicción; desde la separación de los datos de entrenamiento y de test, hasta la evaluación de la efectividad de los modelos mediante matrices de confusión y curvas ROC.

### Separación de los datos de entrenamiento y de test

El primer reto fue generar los datos de entrenamiento, que se utilizarán para generar los modelos; y los datos de test, para evaluar los modelos. La literatura recomienda tener aproximadamente un 70% de los datos dedicados al entrenamiento, y el 30% restante a los test. Sin embargo, tal y como se ha mencionado anteriormente, el dataset tiene muy pocos datos y la variable a predecir está muy desbalanceada (véase la Figura 13), siendo el “NO” la categoría predominante, por lo que se han planteado distintas propuestas para seleccionar la mejor relación entre los datos de entrenamiento y de test.

El dataset tiene 204 entradas de datos. Observando las frecuencias del “SI” y del “NO” de la variable a predecir, se puede comprobar que hay (aproximadamente, depende de cómo se haga la redistribución aleatoria de los casos de “PERSISTENCIA PSA”) unos 190 “NO” y unos 10 (“SI”).

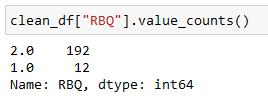


Figura 14. Frecuencias de las distintas categorías (2.0 corresponde al “NO”. 1.0 corresponde al “SI”)

Utilizando la relación estándar de 70/30, los modelos tendrían disponibles (haciendo los cálculos pertinentes) 142 entradas dedicadas al entrenamiento y 61 entradas a los test. A priori esta distribución de los datos puede parecer buena. Sin embargo, resultó interesante reflexionar en si utilizar más datos de entrenamiento (un 80/20, por ejemplo), o más datos de test (un 60/40), para compensar de alguna forma la escasez de datos y el desbalanceo de las distintas categorías.

Si se utilizan más datos dedicados al entrenamiento, con un 80/20 o incluso un 90/10, es muy probable que la calidad de los modelos aumente. El problema de esto es que no habría suficientes datos para testear, y se perdería la capacidad de evaluar correctamente la calidad de los modelos (sobre todo, para evaluar las predicciones de los casos de “SI”, que son muy escasos).

Por otra parte, si se utilizan más datos dedicados a los test, con un 60/40, se obtendría una mejor capacidad de valorar la calidad de los modelos (y más casos de “SI” para testear), a cambio de una peor calidad de los modelos.

En un caso real donde se necesite predecir la recidiva bioquímica, la calidad de los modelos de predicción es muy importante, por lo que en ese caso es muy probable que se acabe utilizando un 80% del dataset para entrenar y el otro 20% restante para evaluar.

Sin embargo, de cara a este TFG se ha optado por tener más datos de test, utilizando un 60% para entrenar y un 40% para testear, ya que de este modo habrá más casos de “SI” para poder mostrar por pantalla y obtener mejores valores de *accuracy*, *precision* y *recall*. Es verdad que estos modelos son un poco peores, pero de esta forma se puede compensar la escasez de “SI” y mostrar algo más “vistoso” y equilibrado en la aplicación.

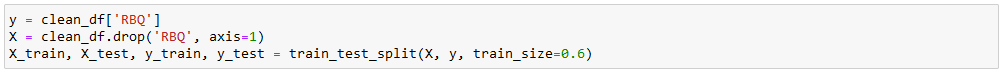


Figura 15. Generación de datos de entrenamiento y de test

De esta forma, hay 122 entradas dedicadas al entrenamiento y 82 entradas para testear.

### Random Forest

Los *Random Forests* (o Bosques Aleatorios en castellano), son una combinación de varios Árboles de Decisión, a los cuales, aleatoriamente, se les pasa un subconjunto de los valores de un conjunto de datos. Este método de aprendizaje entra dentro del grupo de los *Ensamble Learning Methods* (o Métodos de Aprendizaje Combinado), ya que utiliza varios estimadores para generar los modelos de predicción.



Figura 16. Visualización gráfica de Random Forests

En este caso, no ha sido necesario realizar un normalizado ni un escalado para que los datos del dataset tengan un rango similar, ya que éstos métodos no lo requieren. Aplicar estas técnicas en Árboles Aleatorios no aportarían ningún tipo de beneficio.

En scikit-learn, un Random Forest Classifier se puede crear con **RandomForestClassifier()**, y se han utilizado los parámetros **class\_weight**, **bootstrap**, n\_**estimators** y **max\_samples**. Las reflexiones realizadas sobre cada parámetro se describirán a continuación:

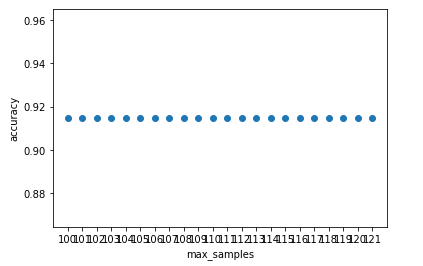
* **class\_weight**: Indica los pesos de cada clase de la variable a predecir (en este caso, RBQ). Por defecto todas las clases tienen peso 1. Se ha utilizado **balanced**, de tal forma que el peso de una clase es inversamente proporcional a la frecuencia de dicha clase. De esta manera, se consigue compensar el desbalanceo del “SI” sobre el “NO” predominante en este modelo.
* **bootstrap**: Establecido a **true**.Este parámetro indica al clasificador que tiene que utilizar subconjuntos del dataset. El tamaño de cada subconjunto es indicado por **max\_samples**.
* **max\_samples**. Indica el número de entradas máximo de cada Árbol de Decisión del *Random Forest*. Para seleccionar este valor, se ha iterado sobre un rango entre 100 y 122 (teniendo el 60% de datos para entrenar tenemos, como máximo, 122 entradas). El rango establecido podría haber sido mayor. Sin embargo, utilizar tan pocas entradas para cada árbol podría perjudicar el entrenamiento de cada árbol.

Se ha ejecutado la función siguiente para encontrar este valor:



Figura 17. Función para encontrar el mejor valor de max\_samples

Los resultados se muestran a continuación:



Se puede observar que cualquier valor ofrece un *accuracy* de aproximadamente 0,92. Cualquier valor de **max\_sample** puede servir. Se ha utilizado, en este caso, utilizar 100 valores para cada Árbol.

* **n\_estimators**: Este parámetro indica el número de árboles de decisión que contendrá el bosque aleatorio. Para seleccionar este valor, se han realizado entrenamientos en varias iteraciones y se ha dibujado un gráfico con los resultados. El número de estimadores mínimo es 2, ya que un Bosque debe de tener más de un Árbol. En cuanto al número máximo, la teoría recomienda utilizar la mayor cantidad posible. Usar muchos estimadores hará que los entrenamientos tarden más. En este caso se ha establecido el máximo a 100. La búsqueda de este valor se ha realizado utilizando la siguiente función:

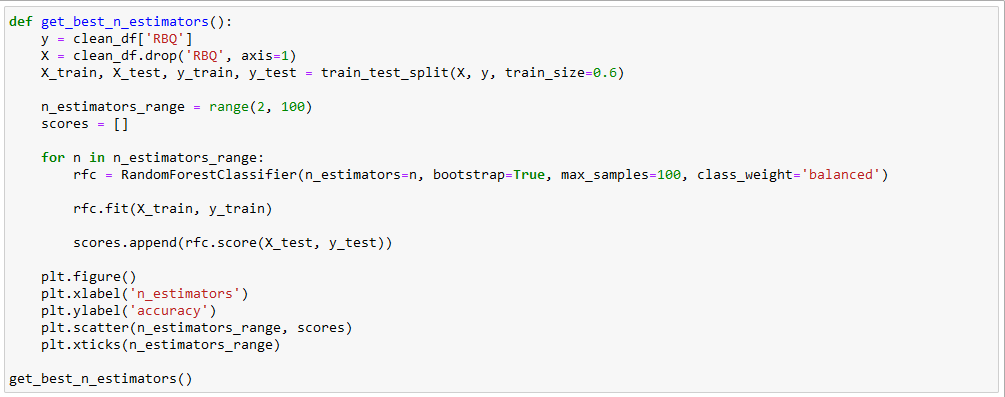


Figura 18. Función para encontrar el mejor valor de n\_estimators

Los resultados se muestran en la figura siguiente:

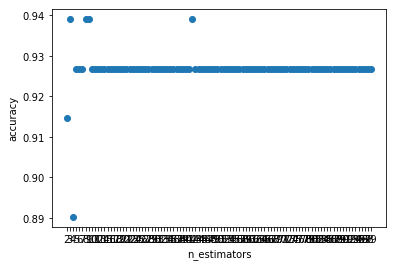


Figura 19. Accuracy para cada valor de n\_estimators

Se puede observar que la precisión es aproximadamente 0,9 para cualquier valor **n\_estimators**, por lo que todos los valores actuarán de manera similar. Existen algunos picos; sin embargo, eso no será siempre así en todas las ejecuciones de la función mencionada anteriormente.

Una vez seleccionados los parámetros, se procede a entrenar y evaluar el modelo con los parámetros seleccionados. En la celda siguiente se muestra el entrenamiento y la generación de los resultados de la evaluación (que no siempre serán iguales).

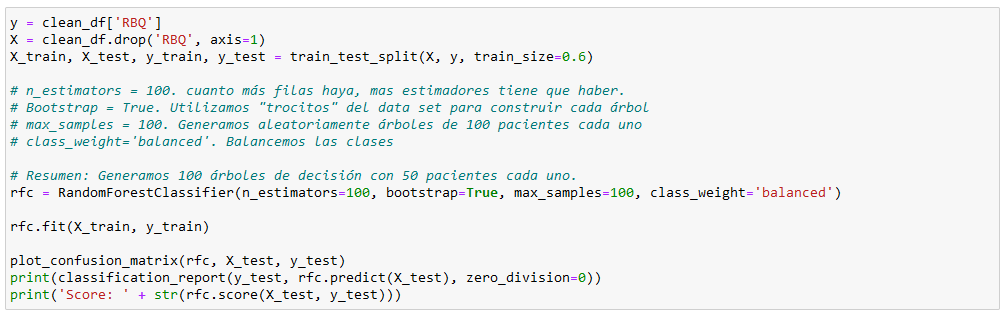


Figura 20. Celda con el entrenamiento y generación de los resultados del entrenamiento

Los resultados del entrenamiento se describen en el informe y en la matriz de confusión que se muestran a continuación.

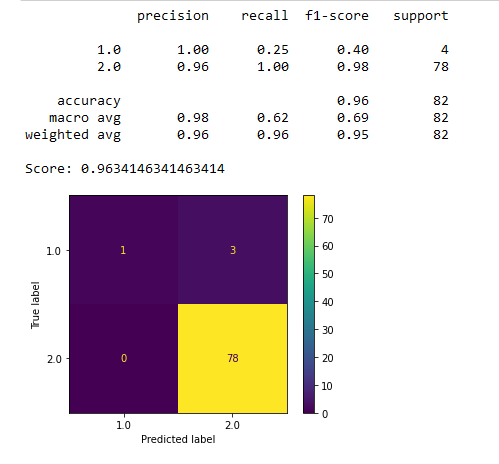


Figura 21. Resultados del entrenamiento

Se obtiene un *score* de 0,96. Este resultado puede parecer bueno. Sin embargo, hay que tener en cuenta otros valores, como *precision* y *recall*.

Se puede comprobar que el modelo es capaz de predecir correctamente la mayoría los casos de “NO”, pero el predictor es muy malo prediciendo los casos de “SI” (solo ha acertado 1 caso de 4). Cabe destacar que los casos de “SI” son muy escasos, por lo que se necesitan más datos para tener un predictor mejor.

Con este primer estimador se ha visto el gran problema que surge al tener tan pocos datos y al tener una variable a predecir muy desbalanceada. Los modelos que se generan están muy sesgados y evaluar la efectividad de un predictor es muy complicado.

Con este dataset, los modelos que se generan tienden a predecir “NO” y en un caso real, es muy probable que los predictores den varios falsos negativos. Este fenómeno se podrá observar también en los estimadores que se mencionarán más adelante.

// TODO curvas ROC y AUC

## k Nearest Neigbours (kNN)