COMILLAS
UNIVERSIDAD PONTIFICIA

Facultad de Ciencias Económicas y Empresariales

EXPLICACIÓN Y PREDICCIÓN DEL DEFAULT EN CRÉDITOS, CON LA IMPLEMENTACIÓN DE MODELOS DE MACHINE LEARNING

Autor: Luis Ramiro López Blanco
Director: María Coronado Vaca

MADRID | Junio de 2022

RESUMEN

Este trabajo analiza los datos de la Base de Datos de *Lending Club*, una plataforma de créditos localizada en Estados Unidos. A través de estos datos, este trabajo analiza las cualidades de los prestatarios que pueden llegar a provocar un *Default* en dichos créditos. Para ello, se lleva a cabo un análisis exploratorio de los datos, analizando todas las variables y sus diferentes valores según las observaciones que pertenecen a cada clase de la variable *target*. Además, usamos ciertos modelos con capacidad explicativa con la finalidad de averiguar cuales son las variables que más influyen y cuales no debemos tener tan en cuenta. Comprobamos, que de las numerosas variables que teníamos en un principio, tan solo unas 7 de ellas son verdaderamente relevantes y describen las cualidades de las personas con mayor probabilidad de generar un *Default* (*int_rate*, *emp_length*, *dti*, *inq_last_6mths*, *Prestamo_Largo*, *mths_since_last_delinq* y *RENT*).

Por otra parte, haciendo uso de distintos algoritmos de *Machine Learning*, intentamos generar y entrenar modelos que tengan capacidad predictiva a la hora de saber en qué créditos se podría dar un *Default*. Usamos modelos de *Logistic Regression*, KNN, *Decision Tree Classifier*, *Random Forest Classifier*, e incluso *AutoML*. Este último modelo usa internamente la optimización bayesiana de hiperparámetros, lo cual también explicamos dentro de este trabajo.

PALABRAS CLAVE

Riesgo de Créditos

Default

Machine Learning

Optimización Bayesiana

Python

ABSTRACT

In this project, we will analyze data from the source Lending Club, a platform that gives loans and is settled in the United States. With this data, we will analyze the main and most important characteristics of borrowers that might lead to causing a Default in said loans. To do this, we will carry out an exploratory analysis of the data, considering all the variables and their different values when the observations are split according to the target variable. Additionally, we will use models that have an explicative capacity in order to see which variables are the most influential, and which are not worth it to take into account. As an outcome, we discover that, of all the variables that we had in the beginning, only 7 of them are actually relevant and describe the qualities of a person with higher probability of causing Default (*int_rate, emp_length, dti, inq_last_6mths, Prestamo_Largo, mths_since_last_delinq* y *RENT*).

The other part of the project is to generate and train models that might have a predictive capacity when it comes to classifying between Defaults and non-Defaults. For this, we will use different kinds of Machine Learning models: Logistic Regression, KNN, Decision Tree Classifier, Random Forest Classifier and AutoML. This last one model uses, internally, Bayesian optimization of hiperparametres, which is also explained in this project.

KEYWORDS

Credit risk

Default

Machine Learning

Bayesian Optimization

Python

INDICE

1.- INTRODUCCIÓN

1.1 Introducción

Tras la crisis financiera de 2008, y debido en parte a un descenso en la confianza del público general sobre la estabilidad de las entidades bancarias, surgió un modelo de préstamo que, aunque podría parecer moderno, se basa en las prácticas más iniciales de esta actividad. De esta forma, aparecieron actores que se dedicaban a poner en contacto a inversores (prestamistas) y prestatarios, sin la necesidad de tener un banco de por medio. Si bien es cierto, que también este modelo ha ido evolucionando con el tiempo, ya que, hoy en día, ya no es necesario que un inversor concreto preste dinero a un cliente concreto, si no que los inversores pueden aplicar su dinero a una parte de un préstamo o dividirlo en varios.

Muchos de estos "nuevos" intermediarios fueron Fintechs. Estos, movidos por la innovación, usan nuevas formas de análisis para discriminar entre clientes, entre las que se encuentra el Big Data, datos no tradicionales y técnicas de *Machine Learning*. Estas nuevas tecnologías están marcando la diferencia en el entorno financiero. De hecho, (Jagtiani and Lemieux, 2019b) los ratings entre instituciones tradicionales como el FICO score y los nuevos análisis han pasado de tener una coincidencia del 80% en 2007 al 35% en 2015. Es por esto que nuestro análisis tendrá más sentido cuanto más nos acerquemos a la actualidad (teniendo en cuenta que nuestros datos van de 2007 a 2018).

Dicho esto, es claro que la discriminación entre clientes que aportan beneficios o que generan pérdidas es un análisis crucial para las empresas financieras, más aún teniendo en cuenta la posibilidad de mejora en esta área gracias a las nuevas tecnologías. Esto, sin embargo, no hace si no aumentar la competencia dentro de este mercado. Además, nos encontramos en un entorno económico en el que los tipos de interés no ayudan a dar beneficios a bancos y similares.

Tal y como venimos diciendo, analizar el riesgo de crédito es muy relevante, ya que supone la primera barrera para defender la solvencia de las entidades financieras, pero también para proveer de liquidez a la sociedad de forma adecuada y sin ineficiencias. Este intento de "búsqueda de la verdad" en cuanto a análisis crediticio está llevándose a cabo tanto por las entidades tradicionales como por los nuevos actores financieros. Sin embargo, en este trabajo realizaremos un análisis por medio de la Base de Datos de *Lending Club*, la compañía más importante de préstamos entre particulares de EEUU.

1.2 Objetivo

El objetivo es analizar las características de los individuos a los que se les ha otorgado la posibilidad de solicitar crédito, con la finalidad de crear modelos explicativos que nos permitan entender qué variables de una persona son las determinantes a la hora de que un crédito resulte o no en impago, además de entrenar un algoritmo de clasificación que sea lo más preciso posible a la hora de predecir posibles *Defaults*. Este algoritmo estará basado en las variables que contengan una información especialmente relevante de entre todas las disponibles. Todo ello basándonos en las bases de datos de la plataforma *Lending Club*, que comprenden datos de préstamos otorgados entre los años 2007 y 2018. Lo novedoso de este trabajo es el uso, no solo de técnicas de *Machine Learning*, si no de la optimización bayesiana de los hiperparámetros de los distintos modelos que se entrenen, ya que no es algo en lo que se haya profundizado mucho hasta el momento.

1.3 Justificación del tema objeto de estudio

Este tema de estudio tiene un gran interés principalmente debido a dos factores clave. En primer lugar, la importancia del riesgo de crédito. El riesgo de crédito resulta de la imposibilidad del deudor de hacer frente a sus obligaciones como prestatario. Esto se traduce en un riesgo para el prestamista de no recuperar el capital que invirtió. Por tanto, se relaciona directamente con la solvencia o liquidez del deudor. Tradicionalmente, esta solvencia se relacionaba sobre todo con el flujo de caja de la entidad prestataria, en el caso de una persona física estaríamos hablando de sus ingresos, ya sean por rentas o por su actividad laboral.

Sin embargo, la irrupción de la tecnología nos permite tener muchos otros factores en cuenta. No solo por la capacidad de obtener esos datos, si no más bien por la capacidad que otorga a la hora de saber cuáles de esos datos “no tradicionales” tienen peso a la hora

de hablar de riesgo de crédito, y de ser el caso, cuál es el peso que debiera tener al tomar la decisión de discriminar potenciales clientes. Es por esto que hacer un análisis del riesgo de crédito con tecnologías de *Machine Learning* es algo novedoso y justifica el tema de estudio.

En segundo lugar, y desde un punto de vista más técnico, abordamos este tema no solo con las nuevas tecnologías, las cuales son novedosas, pero ya se usan en la gran mayoría de entidades financieras, si no que también incorporamos a nuestro análisis de *Machine Learning* la Optimización Bayesiana. Esta técnica, es mucho menos conocida y, por lo tanto, utilizada, pero permite, en teoría, obtener algoritmos que produzcan mejores resultados. Esto se hace a través de la optimización de hiperparámetros por medio de un modelo probabilístico que orienta la búsqueda de estos hacia las áreas que más probabilidad tienen de contenerlos.

1.4 Metodología

Para el desarrollo de lo anteriormente explicado se realizará la siguiente metodología:

Se obtendrán los datos de la web de *Lending Club*, una plataforma online que ofrece préstamos a particulares y negocios a la vez que ofrece posibilidades de inversión. Estos datos serán lo más actualizados posible, y con magnitud razonable dentro de las posibilidades del Trabajo. En estos datos se reflejan un gran número de variables relacionadas con personas a las que se les ha otorgado la posibilidad de solicitar créditos (y que deben estar a fecha de admisión de las solicitudes) en *Lending Club*.

Posteriormente, realizaré un tratamiento de los datos, que consistirá en analizar las variables para descartar aquellas que no sean útiles para el estudio, o que no tengan información relevante. Estos datos serán posteriormente trasladados a herramientas de Python para su análisis y estudio. En concreto, haré uso de la herramienta Spyder (Python 3.9) dentro de la plataforma Anaconda, haciendo uso de paquetes como Autosklearn, así como la herramienta de Google colab para el modelo de *AutoML*.

El estudio se realizará en varias etapas. Primero con algoritmos de *Logit*, KNN, *Decision Tree* o *Random Forest*, con un tiempo de computación bajo. A medida que avance el

trabajo, se irán introduciendo algoritmos de mayor complejidad, y a su vez mayor tiempo de computación, como el algoritmo de *AutoML*.

En la parte final del análisis estaremos trabajando con técnicas de optimización bayesiana de hiperparámetros, que consiste en crear un modelo probabilístico en el que el valor de la función objetivo es una métrica de validación del modelo, dirigiendo la búsqueda hacia áreas de valores que suscitan mayor interés.

1.5 Estructura

En primer lugar, realizaremos una revisión de la literatura para entender en qué punto se encuentra este tema y cuales han sido los avances de los últimos tiempos a través de diversos estudios. Tras esto, empezaremos con el trabajo propiamente dicho.

Comenzamos con un análisis exhaustivo de los datos de los que disponemos, para después proceder a tratarlos. Es decir, analizar variable por variable cuales tienen un peso real sobre el riesgo de crédito, eliminando las que no lo tengan. Con las variables que resten, se procederá a tratar los valores que queden como *Missing Values*, los cuales recibirán un trato distinto dependiendo del caso.

Una vez tengamos los datos limpios, se pueden empezar a entrenar los algoritmos, haciendo un análisis de sus resultados y explicando la relación de los resultados con las características que reflejan las variables. Tras hacer esto con cada modelo que entrenemos, explicaremos las conclusiones del trabajo.

1.6 Revisión de la literatura

El análisis de riesgo de créditos es tan antiguo como el propio préstamo, ya que, desde siempre, el precio de los préstamos y la decisión de otorgarlo o no se ha basado en el riesgo de impago por parte del prestatario. Por tanto, el análisis de riesgo de créditos ha sido un tema extensamente estudiado. Sin embargo, con la entrada de las nuevas

tecnologías, como la aparición de las *Fintechs* y del *Machine Learning*, el estudio de este tema vuelve a ser necesario. Una persona que ha aportado mucho en esta área es Julapa Jagtiani con estudios que tratan sobre la influencia del *Machine Learning* sobre el precio de los créditos, y el creciente uso del *Machine Learning* por parte de las Fintech (Jagtiani, J., Lemieux, C., 2019a); o sobre el papel que juegan plataformas con características similares a *Lending Club* a la hora de ofrecer acceso a créditos a personas que no tienen la capacidad de pedir créditos en los bancos y otras instituciones tradicionales (Jagtiani, J., Lemieux, C. , 2019b).

También encontramos estudios similares al que vamos a llevar a cabo en este trabajo, aunque con ciertas diferencias, como puede ser el de Don Carmichael, que trata de extraer cuales son las variables otorgadas por la base de datos de *Lending Club* que más influencian la aparición de un *Default* (Charmichael, D., 2014). En la misma línea, hay otros estudios que también analizan los datos de la plataforma *Lending Club*, legando a la conclusión de que el riesgo que suponen los *Defaults* no está siendo compensado por los tipos de interés más altos, y que deben intentar tener clientes con más ingresos y mejor nota de crédito (Emekter, R., Tu, Y., Jirasakuldech, B., Lu, M., 2015)

Por otro lado, también ha habido trabajos que analizan, a través de algoritmos de regresión, los préstamos de plataformas de *peer-to-peer lending* para pequeños negocios (Mach, T., Carter, C., Slattery, C., 2014), averiguando que estos suelen tener tipos de interés más altos, ya que tienen más probabilidades de generar *Default*.

Vemos, por tanto, que es un tema sobre el que se ha investigado, ya que la entrada de plataformas como *Lending Club*, el modelo de *peer-to-peer lending*, y el *Machine Learning* han supuesto una revolución para el mundo financiero y, sobre todo, para los créditos personales.

2.- METODOLOGÍA DE DATOS

2.1 Descripción de la Base de Datos

Lending Club es una plataforma de préstamos nacida en 2007. Desde ese momento, más de tres millones de miembros se han unido a ella. Se definen como una empresa *Fintech*. Es decir, que hacen uso de tecnología más avanzada que la que cabría esperar de una institución bancaria más tradicional. Esta plataforma tiene como valores la innovación, como base para crear soluciones que aporten más valor; la simplificación de la experiencia del préstamo; el desarrollo ético y responsable de la industria; y la reducción de los costes de intermediación a través de un "marketplace" que permita oportunidades y rentabilidades más atractivas. (Web *Lending Club*, 2022)

La Base de Datos en la que se basa este trabajo ha sido obtenida indirectamente de la página web de la plataforma *Lending Club*, y a través de Kaggle, un *hub* online dónde miles de usuarios comparten datos que pueden ser relevantes para estudios, especialmente de *Machine Learning*. Esta Base de Datos, antes de ser tratada ni modificada, presentaba todos los créditos otorgados por la plataforma entre los años 2007 y 2018. En total, esta Base de Datos, albergaba en torno a 1.500.000 observaciones y alrededor de unas 120 variables, con un peso total de en torno a 1,6 GB de memoria.

De esas 120 variables, hay contenidas variables de todo tipo, tanto cualitativas como cuantitativas, binarias, e incluso de tipo temporal. Muchas de estas variables fueron introducidas en el momento en el que se aprobó el crédito y este fue desembolsado en la cuenta bancaria del prestatario, ya que fueron usadas para tomar la decisión sobre si dicho crédito se otorgaba o no, como pueden ser la variable *home_ownership*, o *anual_ inc*, que reflejan la situación de habitabilidad de las personas que pretenden acceder a créditos, o el salario anual de dicha persona. Otras variables son el resultado de un análisis de riesgo por parte de la plataforma para poner los términos de un crédito que se pretende otorgar, como puede ser la variable *int_rate*, que refleja el tipo de interés que impone la plataforma para cada préstamo individual. Además, también encontramos variables que no son a fecha de admisión, si no que son *to date*, las cuales consideraremos más adelante.

Finalmente, tenemos nuestra variable dependiente *loan_status*, que refleja el estado del crédito en el momento en el que se extrajo esa Base de Datos. Para que nuestro análisis tenga sentido, hemos seleccionado únicamente aquellas observaciones en las que el plazo de devolución del crédito esté cerrado. Por tanto, para nuestra variable dependiente, solo caben 2 opciones: o bien el préstamo ha sido *Fully Paid*, en cuyo caso el préstamo fue devuelto dentro del plazo otorgado, o *Charged Off,* en cuyo caso el préstamo sufrió un impago total o parcial. De esta forma, podemos concluir que la variable dependiente es de binario.

Antes de continuar con el análisis de las variables, es necesario hacer varias puntualizaciones sobre la Base de Datos que vamos a usar. En primer lugar, a pesar de que la Base de Datos original contenía otro fichero adicional con los préstamos que habían sido rechazados, nosotros usamos únicamente la Base de Datos con los créditos que sí fueron concedidos, lo cual puede llevar a distintos sesgos durante el análisis. Sin duda conduce a una reducción radical del poder predictivo de los algoritmos que entrenemos con estos datos, ya que todas las observaciones han sido previamente filtradas a través de los requisitos que impone la plataforma para solicitar el crédito. Esto puede dar lugar a que, al analizar que variables son relevantes y cuales no, queden "descalificadas" algunas que, si tomásemos una muestra de solicitantes no pre-filtrados serían de vital importancia.

Teniendo en cuenta la gran cantidad de variables obtenidas de la Base de Datos original de *Lending Club*, hacemos un análisis preliminar sobre cuales pueden tener interés para nuestro estudio. Basándome en mi propio análisis y el de la Profesora María Coronado Vaca, selecciono las variables que tienen potencial para influir finalmente sobre la variable dependiente. Esta discriminación de variables se hace por los siguientes motivos.

En primer lugar, las que tienen todos los registros como *missing values* (NAs). Aquí encontramos las variables que por motivos de confidencialidad hemos de eliminar, que son *number_id* y *member_id*. También son variables vacías las siguientes 51:

url, tot_coll_amt, tot_cur_bal, open_acc_6m, open_act_il, open_il_12m, open_il_24m, mths_since_rcnt_il, total_bal_il, il_util, open_rv_12m, open_rv_24m, max_bal_bc, all_util, total_rev_hi_lim, inq_fi, total_cu_tl, inq_last_12m, acc_open_past_24mths, avg_cur_bal, bc_open_to_buy, bc_util, mo_sin_old_il_acct, mo_sin_old_rev_tl_op, mo_sin_rcnt_rev_tl_op, mo_sin_rcnt_tl, mort_acc, mths_since_recent_bc, mths_since_recent_bc_dlq, mths_since_recent_inq, mths_since_recent_revol_delinq,

num_accts_ever_120_pd, num_actv_bc_tl, num_actv_rev_tl, num_bc_sats, num_bc_tl, num_il_tl, num_op_rev_tl, num_rev_accts, num_rev_tl_bal_gt_0, num_sats, num_tl_120dpd_2m, num_tl_30dpd, num_tl_90g_dpd_24m, num_tl_op_past_12m, pct_tl_nvr_dlq, percent_bc_gt_75, tot_hi_cred_lim, total_bal_ex_mort, total_bc_limit, total_il_high_credit_limit.

Las siguientes variables (28) se refieren a préstamos que no son individuales o que tienen que ver con el *hardship_flag.* Además, también son variables completamente vacías, por lo que las eliminamos.

annual_inc_joint, dti_joint, verification_status_joint, revol_bal_joint, sec_app_earliest_cr_line, sec_app_inq_last_6mths, sec_app_mort_acc, sec_app_open_acc, sec_app_revol_util, sec_app_open_act_il, sec_app_num_rev_accts, sec_app_chargeoff_within_12_mths, sec_app_collections_12_mths_ex_med, sec_app_mths_since_last_major_derog, hardship_type, hardship_reason, hardship_status, deferral_term, hardship_amount, hardship_start_date, hardship_end_date, payment_plan_start_date, hardship_length, hardship_dpd, hardship_loan_status, orig_projected_additional_accrued_interest, hardship_payoff_balance_amount, hardship_last_payment_amount

Por otra parte, eliminamos la variable *mths_since_last_*record, ya que, aunque no tiene todos sus valores como NAs, sí casi todos, no logramos entender correctamente el significado de esta, por lo que sería un gran problema a la hora de interpretar su influencia.

También descartamos las variables que tienen en todos los registros valor 0 (como las variables *mths_since_last_major_derog, delinq_amnt, acc_now_delinq, out_prncp out_prncp_inv,* ya que tratamos con una Base de Datos de préstamos ya finalizados), o que tienen un porcentaje de valores distintos a 0 inferiores al 0.01% de las observaciones de la Base de Datos.

En otras variables, como dijimos antes, observamos que no son a fecha de admisión del crédito, por lo que también hemos de eliminarlas. Este es el caso de las variables *total_rec_prncp, total_rec_int, total_rec_late_fee, recoveries, collection_recovery_fee, last_pymnt_d, last_pymnt_amnt, next_pymnt_d.*

Otras variables tienen todos sus valores iguales, por lo que no aportan ninguna información para nuestro análisis, y en consecuencia las eliminamos. Aquí encontramos variables como*: disbursement_method, policy_code, initial_list_status, pymnt_plan*.

Otras variables como *desc*, *total_pymnt* son resultado de operaciones aritméticas entre otras variables, por lo que no nos aportarían información en un análisis futuro. Por otra parte, las variables *grade* y *subgrade* han de ser eliminadas, ya que presentan el propio método de *scoring* de préstamos de la plataforma *Lending Club*, por lo que no tendría sentido incluirlas a la hora de hacer nuestro análisis y algoritmo de *scoring*. Hay también variables relativas al FICO *score*, que eliminamos por el mismo motivo. También eliminamos las variables que se refieren a las refinanciaciones, ya que no sabemos si estas variables son a fecha de admisión.

Por último, la variable *mths_since_last_delinq* necesita de una aclaración adicional. Cabe la razonable duda sobre si esta variable se refiere al número de meses desde el último impago dentro del préstamo al que la observación hace referencia, o si bien se refiere a otros préstamos que el prestatario haya podido obtener con anterioridad. La importancia de este detalle es clave, puesto que, si se refiere a los impagos sobre el propio crédito objeto de estudio, no sería una variable a fecha de admisión y contaría con sesgo. A pesar de esta duda, no tiene sentido que esta variable fuese *to date*, ya que, en ese caso, todas las observaciones con *Charged Off* deberían tener un valor de la variable como 0, lo cual no es así.

Una vez hecha esta selección preliminar de variables, procedo a explicar el significado de las 19 variables (Tabla 1) que sí van a estar incluidas dentro de nuestro análisis. De estas 19 variables, 15 son cuantitativas, mientras que 4 son cualitativas. En la siguiente tabla se especifica más aún su tipo y su definición:

Tabla 1: Listado de las variables que se incluirán en los modelos, así como el tipo de variable y su descripción

Variable	Tipo de variable	Descripción
Loan_amount	Cuantitativa Discreta	Monto total solicitado ($)

Term	Cuantitativa Discreta	Plazo del préstamo en meses (3 o 5 años)
Int_rate	Cuantitativa Continua	Tipo de interés (%)
Installment	Cuantitativa Continua	Cuota mensual ($)
Emp_length	Cuantitativa Discreta	Antigüedad en el trabajo (años)
Home_ownership	Cualitativa No Ordenable	Situación de propiedad sobre la vivienda (own, mortgage o rent)
Anual_inc	Cuantitativa Discreta	Ingreso anual ($)
Loan_status	Cualitativa Binaria	Variable Dependiente. Estado del crédito
Purpose	Cualitativa No Ordenable	Finalidad del préstamo. (13 posibles finalidades)
Addr_state	Cualitativa No Ordenable	Estado de E.E.U.U
Dti	Cuantitativa Continua	Pagos mensuales por deudas / Ingresos mensuales (%). No computan los pagos de deudas por hipoteca

Delinq_2yrs	Cuantitativa Discreta	Número de Retrasos de más de 30 días en los últimos 2 años
Inq_last_6mths	Cuantitativa Discreta	Número de Solicitudes de préstamos excluyendo para casa o coche en los últimos 6 meses
Mths_since_last_delinq	Cuantitativa Discreta	Número de meses desde el último impago
Open_acc	Cuantitativa Discreta	Número de Líneas de crédito abiertas
Pub_rec	Cuantitativa Discreta	Número de Retrasos en los ficheros públicos
Revol_bal	Cuantitativa Discreta	Balance total de sus créditos revolving
Revol_util	Cuantitativa Continua	Porcentage de utilización de sus créditos revolving
Pub_rec_bankrupcies	Cuantitativa Discreta	Número de Quiebras en el registro público

Fuente: Elaboración propia

Así quedaría configurada, en una primera instancia nuestra Base de Datos, donde tenemos un total de 19 variables. De estas, 18 serán parte de la X y la variable dependiente será nuestro *target* o Y. Haciendo una primera exploración sobre esta variable dependiente *loan_status,* observamos ciertas características de la Base de Datos que vale la pena comentar. La Base de Datos no contiene una proporción balanceada de observaciones para cada una de las dos posibilidades de la variable objetivo, si no más bien todo lo contrario. Alrededor de un 20,5% de las observaciones tienen valor *Charged Off* en la variable *loan_status,* mientras que un 79,5% de estas tienen valor *Fully Paid*. Esto tendrá

implicaciones y requerirá un tratamiento distinto a la hora de realizar procesos como la Cross-Validation, o la separación entre *Training set* y *Test set*.

2.2 Tratamiento de la Base de Datos

En primer lugar, debemos tratar la Base de Datos para que cada variable sea del tipo que debe ser. Esto quiere decir que aunque la herramienta Spyder (Python 3.9), que será la que usemos principalmente a lo largo del trabajo, nos diga que una variable es de un tipo habrá que revisar si hay algún problema en esa clasificación.

Las variables *loan_amount*, *anual_inc*, *delinq_2yrs*, *inq_last_6mths*, *mths_since_last_delinq*, *open_acc*, *pub_rec*, *revol_bal* y *pub_rec_bankruptcies*, aparecían en un principio como variables de tipo *object*, es decir, cualitativas, ya que los decimales no estaban correctamente escritos. Para ello, hacemos un tratamiento previo para convertir los puntos en comas, y que así tome las variables como cuantitativas. Por otra parte, la variable *term* estaba siendo tratada como una variable cualitativa, ya que al número de meses estaba añadida la palabra "meses". Al separar esa variable y dejarla asignada solo al valor numérico, la convertimos en una variable cuantitativa. La variable *emp_length*, referida a la antigüedad en el puesto de trabajo, tenía un formato que le llevaba a comportarse como una variable cualitativa, ya que los valores referidos a antigüedades de menos de un año o de más de 10 años estaban escritas como <1 y 10+ respectivamente, mientras que el resto de valores intermedios sí eran simplemente números. Para remediarlo, simplemente quitamos los símbolos sobrantes de forma que todas las observaciones de clientes que tengan menos de un año o más de diez años de antigüedad quedarían como 1 y 10 respectivamente. En cuanto a problemas de formato en la Base de Datos, no encontramos ninguno más.

Tras esto, debemos tratar con el problema de los *missing values* y de las posibles observaciones que se deban eliminar. No todas las variables presentan este problema, pero iremos una por una para asegurarnos. La primera variable en la que advertimos NAs es la variable *emp_length*. Para esta variable, encontramos que en torno a un 6% de las observaciones tienen como valor NA en esta variable. Lo importante en este caso es ver qué significado pueden tener estos huecos y como tratarlos. Para ello, comparamos esta

variable con una de las variables que no estará incluida posteriormente *emp_title.* Al comparar los *missing values* de ambas variables, observamos que hay más NAs en la variable referida al título que en la variable referida a la antigüedad, si bien lo que nos interesa realmente es si todas las observaciones con NA en *emp_length* también tienen NA en la variable *emp*_title. Observamos que más del 95% de las observaciones con NA en *emp_length* tienen NA en *emp*_title, por lo que lo lógico será que las observaciones con valor NA en *emp_length* simplemente debieran tener valor 0. Esto se sostiene por el hecho de que las observaciones con NAs en *emp_length* también tienen la variable de *emp_title* como NA, por lo que probablemente no tengan trabajo en ese momento. Además, de estos NAs, el porcentaje de *Charged Offs* es mayor que en la Base de Datos en general, llegando a en torno al 27% de las observaciones, lo cual tendría sentido, pues las personas que carecen de trabajo tendrían más dificultades para responder a las obligaciones sobrevenidas a consecuencia de la obtención de un crédito. Procedemos, por tanto, a sustituir los NAs por 0.

También se presenta el problema de los NAs en la variable *revol_util*, con un porcentaje de NAs en la Base de Datos inferior al 0,1%. Comparamos esta variable con la variable *revol_bal*, tras lo cual advertimos que todas las observaciones que tienen valor NA en la variable *revol_util*, tienen valor 0 en la variable *revol_bal*. Esto tiene sentido, ya que si *revol_bal* es 0, no hay nada de lo que disponer, por lo que el porcentaje de utilización, *revol_util*, debe ser 0. Comprobamos además la consistencia de nuestra argumentación al observar que para todas las observaciones con valor 0 en la variable *revol_bal*, no observamos ningún porcentaje distinto de 0 en la variable asociada. Procedemos, por tanto, a sustituir los NAs de la variable *revol_util* por 0.

Para terminar con el tratamiento de los *missing values*, observamos la variable *moths_since_last_delinq*, con, aproximadamente, un 48,4% de los valores como NAs. Aquí el tema de interés vuelve a ser el significado de esos NAs. Pues bien, teniendo en cuenta que la variable se refiere a cuantos meses han pasado desde la última vez que el prestatario incurrió en un impago (de un crédito distinto al que está aquí reflejado, pues como ya dijimos antes, esta variable es a fecha de admisión), los 0 son radicalmente distintos a los NAs. Esto se debe a que un 0 significa que el prestatario ha cometido un impago hace 0 meses, es decir, en el último mes; mientras que un NA significa que nunca ha cometido un impago. Por tanto, sustituiremos los NAs por su significado: "Sin Impagos".

El siguiente paso lógico, como ya podemos intuir por la configuración de la variable *mths_since_last_delinq* (Sin Impagos, 0, 1, 2. 3…104), es modificar las variables para que tengan un significado más pleno y tengan sentido a la hora de tratar con ellas. De las 19 variables, las 12 siguientes no necesitan tratamiento previo adicional: *loan_amnt, int_rate, installment, emp_length, anual_inc, dti, delinq_2yrs, inq_last_6mths, open_acc, pub_rec, revol_bal* y *pub_rec_bankruptcies*. Sobre las otras 7 variables iremos una por una depurando la Base de Datos y creando variables utilizables. Sin embargo, debido a limitaciones computacionales, debemos reducir el tamaño de la Base de Datos por medio de una muestra aleatoria creada a través de un programa de Spyder (Python 3.9) antes de poder tratar las variables. El siguiente programa de elaboración propia (Apéndice 1) toma la Base de Datos original ("ACCEPTED_FULL.csv") fila por fila tomando la primera, *header,* para luego ir discriminando de forma aleatoria. Esta discriminación se realiza a través de la asignación de un número aleatorio entre 0 y 1 a cada fila, haciendo uso del paquete *random*. Tras esto ponemos un "tope" con el cual controlamos indirectamente el tamaño de la muestra que vamos a extraer. Por ejemplo, si ponemos como tope 0,1 significa que solo se tomarán las filas cuyo número aleatorio asignado esté por debajo de ese tope, por lo que, basándonos en la estadística, y al tener una Base de Datos tan grande, se cogerán el 10% de los datos de forma completamente aleatoria. En nuestro caso, hemos usado un "tope" o *threshold* de 0,02, por lo que tomaremos una muestra del 2% de las observaciones de la Base de Datos original. Estas filas que va leyendo el programa y cumplen con la condición se van escribiendo en otro fichero, por lo que en ningún momento se carga a memoria la Base de Datos entera (lo cual daría bastantes problemas), si no solo las filas elegidas aleatoriamente.

Como veníamos diciendo, hay 7 de las 19 variables que necesitan de un tratamiento previo antes de poder usarlas. En primer lugar, la variable *revol_util*, que representa el porcentaje utilizado de la variable *revol_bal*, y por lo tanto debería tener valores de entre 0 y 100, tiene un 0,5% de observaciones con valores superiores a 100. Puesto que esto no debería ser así, ya que, en principio, no se podría usar un porcentaje superior al 100%, sustituimos esos valores por el valor máximo, es decir, 100. Cabe destacar que entre este 0,5% antes mencionado, el porcentaje de *Charged Offs* sube en torno al 27%, un 6% más que en la Base de Datos. Además, transformamos la variable *mths_since_last_delinq* en una variable completamente categórica, ya que teníamos tanto valores numéricos como "Sin Impagos". Para solucionarlo, agrupamos los valores en las siguientes 4 categorías: A: Sin

Impagos (No ha incurrido nunca en impago), B: >12meses (la última vez que tuvo un impago fue hace más de un año), C: 3-12meses (La última vez que tuvo un impago fue hace 3 a 12 meses) y D: <3meses (Tuvo un impago hace menos de 3 meses). Otra modificación a realizar es convertir la variable *term*, teóricamente cuantitativa, en una variable con valores "Préstamo Corto" (préstamos concedidos a 36 meses) o "Préstamo largo" (préstamos concedidos a 60 meses). Tras esto, aún quedarían 6 variables que tratar más en profundidad: *term, home_ownership, porpuse, addr_state, mths_since_last_delinq* y la variable *target: loan_status*.

Como apunte, dentro de la variable *porpuse*, una de las categorías no entraría dentro del espectro de los créditos personales al consumo: *small business*, que representa en torno a un 1% de la muestra. Si bien es cierto que no es una cantidad abrumadora de datos, es importante averiguar si estas observaciones van a distorsionar los resultados de un análisis en el que, ya desde un principio, no deberían tener parte. Por tanto, analizamos si la proporción de *Charged Offs* dentro de este subsegmento es distinta a la proporción general de la muestra. Efectivamente, el porcentaje de *Charged Offs* es de 27,3% cuando hablamos de *small business*, mientras que la media de toda la muestra es del 20,5%. Por tanto, debemos eliminar estas observaciones.

El tratamiento de las 6 variables que lo requieren lo hacemos a través de un programa de elaboración propia en Spyder (Python 3.9) en el que convertimos las variables categóricas en variables *dummies*. Este programa (Apéndice 2) lee el archivo con las 19 variables originales e introduce directamente las 13 variables que, como hemos dicho, ya no necesitan más tratamiento. Después, va tomando una a una las 5 variables que no son *target* y las va convirtiendo en variables binarias, de forma que, por cada variable categórica con N categorías, saca N-1 variables binarias o *dummies*. Esto quiere decir que para la variable *home_ownership* (MORTGAGE, OWN, RENT) ahora tenemos dos variables binarias *OWN* y *RENT*, y si ambas son 0, entonces significaría que es MORTGAGE.

Para la variable *purpose*, ahora tenemos 10 variables binarias que son *credit_card, debt_consolidation, home_improvement, house, major_purchase, medical, moving, other, renewable_energy* y *vacation*. Si todas ellas tienen valores iguales a 0 en una observación, entonces quiere decir que el *purpose* es *car*. Cabe destacar que en este tipo

de conjuntos de variables lógicamente solo una de ellas puede tener valor 1 en cada observación.

Para la variable *addr_state,* ahora tenemos 49 variables binarias correspondientes a 49 de los 50 estados de Estados Unidos que estaban reflejados en la muestra. En el caso de que esas 49 variables tuviesen valores 0 en una observación, querría decir que la dirección del prestatario es la correspondiente a Arkansas (AK).

Para la variable *mths_since_last_delinq*, la cual habíamos transformado en 4 categorías de la A a la D según el tiempo que había pasado desde el último impago, ahora tenemos 3 variables llamadas *B, C* y *D.* Como vemos, la categoría A, referente a los prestatarios que nunca habían tenido un impago ha desaparecido, por lo que se dará cuando las 3 nuevas variables tengan valor igual a 0.

Para la variable *term* ahora tenemos una sola variable binaria llamada *Prestamo_Largo* que hace referencia a si el préstamo es de 60 meses o no. En caso de que el valor sea 0, y por lo tanto no sea un préstamo largo, significaría que se trata de un préstamo corto, es decir, de 36 meses.

Por último, para la variable *target*, referida al *loan_status*, ahora tenemos una variable llamada *Y*. Esta variable binaria muestra un 0 cuando el préstamo está en *Fully Paid*, y un 1 cuando el préstamo está en *Charged Off*. Por lo tanto, de aquí en adelante, cuando se prediga un negativo o un positivo, nos estaremos refiriendo a que se predice un No *default* o un *Default* respectivamente. Con todo esto, el *dataframe* que usaremos de aquí en adelante contiene 11.094 filas, 78 variables independientes (X), y 1 variable dependiente (Y). De las 78 variables dependientes: 8 son variables cuantitativas discretas, 5 son variables cuantitativas continuas, y 65 son variables binarias (que son realmente las otras 6 variables cualitativas después de convertirlas en *dummies*). Todo esto lo observamos al ejecutar el comando *datos.info()* en nuestro programa de Spyder (Python 3.9), y lo vemos en los Elementos 2 y 3 que se muestran a continuación.

Elemento 2 (Izquierda): Categoría de las 79 variables . Excluyendo las variables de Estado y la variable *Target*

Elemento 3 (Derecha): Categoría de las variables originales

```
RangeIndex: 11094 entries, 0 to 11093
Data columns (total 79 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   loan_amnt             11094 non-null  int64
 1   int_rate              11094 non-null  float64
 2   installment           11094 non-null  float64
 3   emp_length            11094 non-null  int64
 4   annual_inc            11094 non-null  float64
 5   dti                   11094 non-null  float64
 6   delinq_2yrs           11094 non-null  int64
 7   inq_last_6mths        11094 non-null  int64
 8   open_acc              11094 non-null  int64
 9   pub_rec               11094 non-null  int64
 10  revol_bal             11094 non-null  int64
 11  revol_util            11094 non-null  float64
 12  pub_rec_bankruptcies  11094 non-null  int64
 13  OWN                   11094 non-null  uint8
 14  RENT                  11094 non-null  uint8
 15  credit_card           11094 non-null  uint8
 16  debt_consolidation    11094 non-null  uint8
 17  home_improvement      11094 non-null  uint8
 18  house                 11094 non-null  uint8
 19  major_purchase        11094 non-null  uint8
 20  medical               11094 non-null  uint8
 21  moving                11094 non-null  uint8
 22  other                 11094 non-null  uint8
 23  renewable_energy      11094 non-null  uint8
 24  vacation              11094 non-null  uint8
 25  B                     11094 non-null  uint8
 26  C                     11094 non-null  uint8
 27  D                     11094 non-null  uint8
 28  Prestamo_Largo        11094 non-null  uint8
```

```
RangeIndex: 11094 entries, 0 to 11093
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   loan_amnt             11094 non-null  int64
 1   term                  11094 non-null  object
 2   int_rate              11094 non-null  float64
 3   installment           11094 non-null  float64
 4   emp_length            11094 non-null  int64
 5   home_ownership        11094 non-null  object
 6   annual_inc            11094 non-null  float64
 7   loan_status           11094 non-null  object
 8   purpose               11094 non-null  object
 9   addr_state            11094 non-null  object
 10  dti                   11094 non-null  float64
 11  delinq_2yrs           11094 non-null  int64
 12  inq_last_6mths        11094 non-null  int64
 13  mths_since_last_delinq 11094 non-null  object
 14  open_acc              11094 non-null  int64
 15  pub_rec               11094 non-null  int64
 16  revol_bal             11094 non-null  int64
 17  revol_util            11094 non-null  float64
 18  pub_rec_bankruptcies  11094 non-null  int64
```

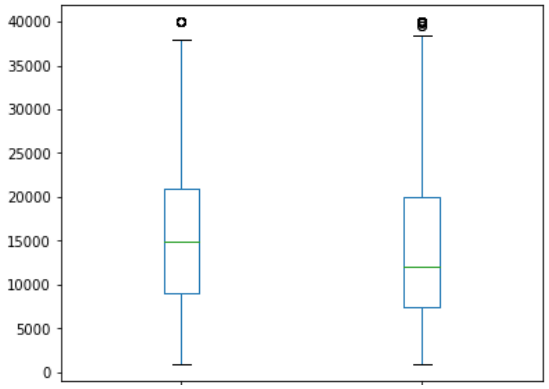
Fuente: Elaboración propia

2.3 Análisis exploratorio de los datos

Tras este primer paso por la Base de Datos, procedemos a realizar un análisis exploratorio (Apéndice 2) más exhaustivo. Este análisis se llevará acabo haciendo uso de la Base de Datos que refleja el Elemento 3, pues las variables del Elemento 2, *dummies*, dificultan mucho la realización de este análisis. Compararemos el efecto de cada variable en el porcentaje de "positivos" (De aquí en adelante, nos referiremos a las observaciones con variable y=0 como negativos, es decir, que han sido *No Default*. Llamaremos positivos a las observaciones con variable $y=1$, es decir, *Defaults*) que contiene la variable *Y*, antes

loan_status. Empezamos con la variable *loan_amnt*, para ver si hay una diferencia significativa en la cuantía entre positivos y negativos (Gráfico 4).

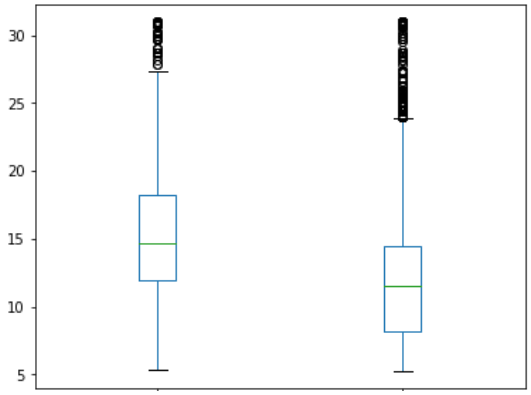
Gráfico 4: Boxplot de la variable cuantía del préstamo respecto de la variable estatus del crédito



Fuente: Elaboración propia

Como podemos observar, sí hay una pequeña variación, puesto que los créditos con estatus *Charged Off* (izquierda) tienen una mediana y percentil 75 ligeramente superior a los créditos *Fully Paid* (derecha). Sin embargo, la envergadura de esta no es excesivamente grande, por lo que ya veremos más adelante si es significativa esta variable a la hora de determinar los positivos y negativos. A continuación, veremos el efecto de la variable *int_rate*:

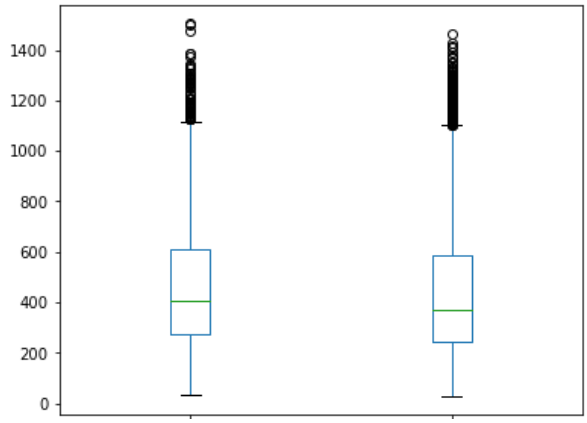
Gráfico 5: Boxplot de la variable tipo de interés respecto de la variable estatus del crédito



Fuente: Elaboración propia

En este gráfico (Gráfico 5) sí que observamos una diferencia más marcada que antes. Tanto la mediana como la caja están en torno a un 4% por encima en los *Default*. Si lo pensamos, esto tiene una explicación muy evidente, la plataforma *Lending Club* otorga tipos de interés más elevados a los individuos solicitantes que suponen, según sus criterios, un riesgo superior de impago. Por tanto, simplemente observamos la realización de un riesgo que ya se había tenido en cuenta y que tenía, como medida, ese tipo de interés superior.

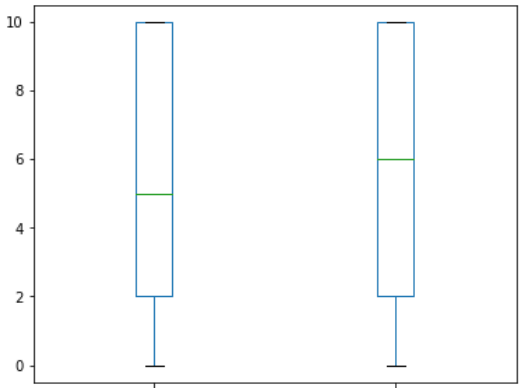
Gráfico 6: Boxplot de la variable cuota mensual respecto de la variable estatus del crédito



Fuente: Elaboración propia

La variable de cuota mensual no es significativa a la hora de determinar el estatus del crédito, ya que los *No Default* (derecha) y los *Default* (izquierda) se distribuyen de forma similar según esta variable (Gráfico 6). Es importante recordar que esta variable es muy similar a una construcción a partir de las variables de longitud del préstamo y de la cuantía total de este, aunque se añaden también los intereses. Por tanto, aunque la variable de cuota mensual y cuantía del préstamo no estén completamente correlacionadas, sí que veremos una relación fuerte.

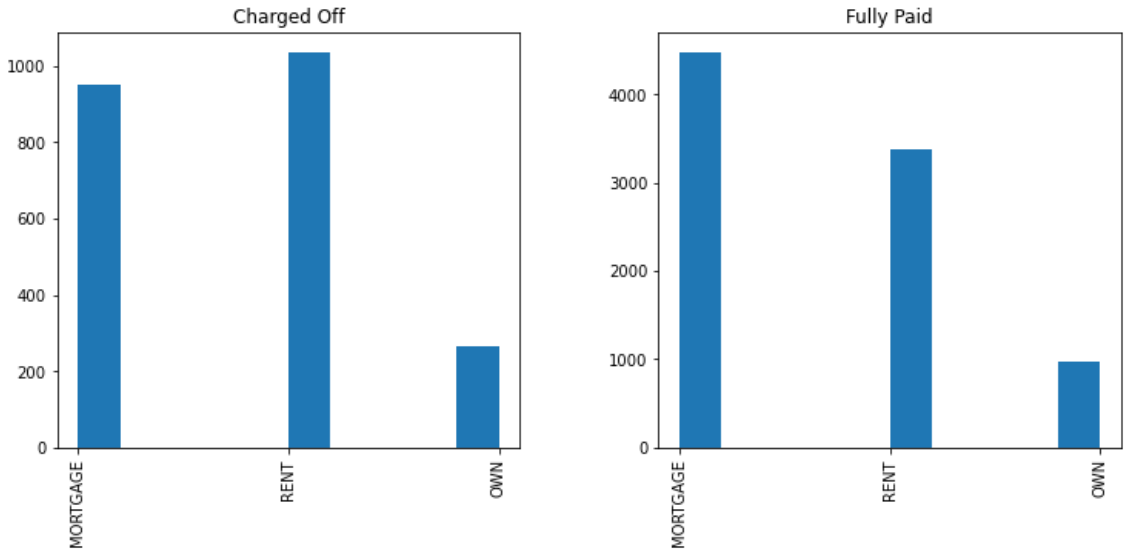
Gráfico 7: Boxplot de la variable atigüedad laboral respecto de la variable estatus del crédito



Fuente: Elaboración propia

En este gráfico (Gráfico 7) observamos que la caja del *Boxplot* es muy similar para los *No Default* (derecha) y *Default* (izquierda). Sin embargo, esto no nos debe confundir, ya que el hecho de que todos los valores por encima de 10 años tengan asignados un 10 nos distorsiona la percepción. La mediana, por el contrario, si nos indica que las personas *No Default* tienen una antigüedad laboral superior. Y es que, es muy probable que, a mayor antigüedad laboral, mayor seguridad económica y capacidad para hacer frente a las obligaciones del préstamo.

Gráfico 8: Distribución de las posibilidades habitacionales según estatus del crédito

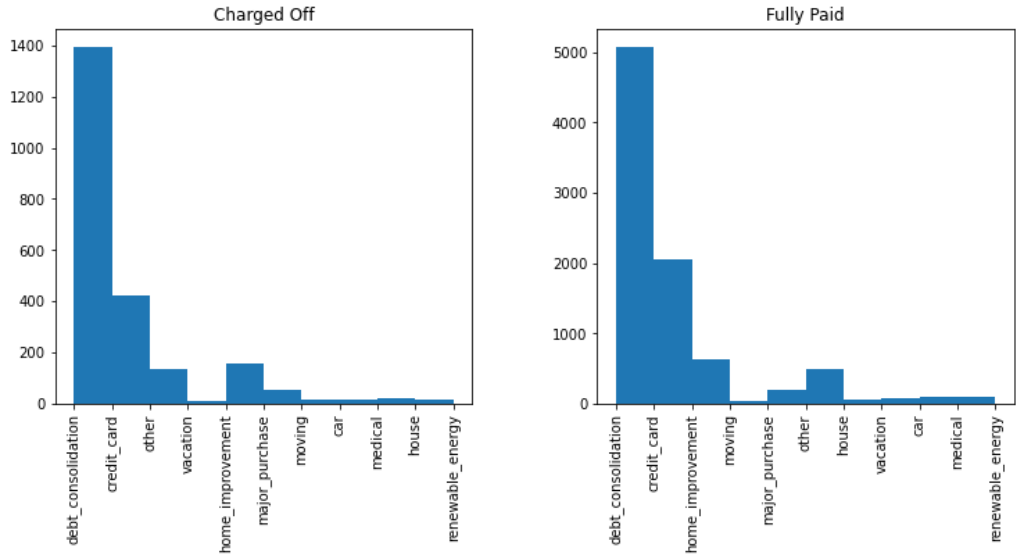


Fuente: Elaboración propia

La variable de *home_ownership* se distribuye de esta manera (Gráfico 8). Vemos que en este caso sí hay diferencias significativas, ya que, aunque hay una proporción similar de personas que son propietarias de sus viviendas, vemos que hay mayor cantidad de *Defaults*, en proporción, que *No Defaults* que habitan viviendas alquiladas. Este se puede deber a que las personas que tienen una casa en propiedad o están en proyecto de tenerla (hipotecados), tienen mayor seguridad económica y capacidad de pagar sus deudas. Las personas que sólo se pueden permitir, por sus medios económicos, vivir en una casa alquilada, tendrán menor calidad como prestatarios.

En cuanto a la variable de ingreso anual, las características de sus valores no nos permiten visualizar de forma correcta su distribución en un gráfico de *Boxplot*. Sin embargo, sí observamos que los *Default* tienen valores más bajos en esta variable, por lo que podría ser significativa.

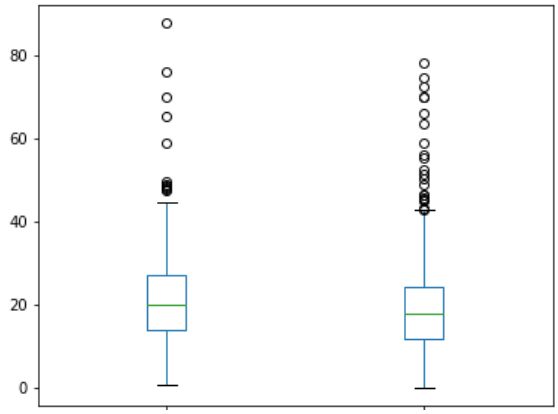
Gráfico 9: Distribución de la variable propósito de crédito respecto de la variable estatus del crédito



Fuente: Elaboración propia

La variable propósito (Gráfico 9) se distribuye con similitud tanto para los positivos como los negativos (El gráfico se ha diseñado proporcionalmente a pesar de que haya más número de observaciones para los *No Defaults*, ya que sólo queremos fijarnos en las proporciones). Lo cual nos indica que la relevancia de las variables *dummies* creadas a partir de esta variable pueden no tener una relevancia muy grande.

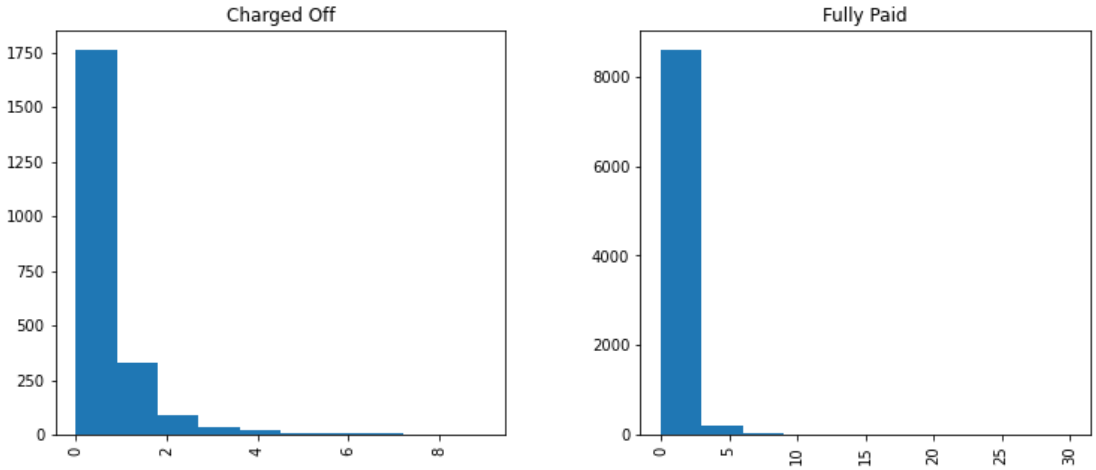
Gráfico 10: Boxplot de DTI (pagos deuda mensual / ingreso mensual) según estatus del crédito



Fuente: Elaboración propia

En el anterior gráfico (Gráfico 10) se aprecia una ligera diferencia de distribución entre los *Defaults* y los *No Defaults*. Los *Defaults* (izquierda) dedican más porcentaje de sus ingresos al pago de deudas que los *No Defaults*. Esto puede ser causado, bien porque tengan una deuda más grande, o bien porque tengan unos ingresos más pequeños que los otros. Quiere decir, en cualquiera de los dos casos, que los *Defaults* tienen una capacidad más reducida de hacer frente a las deudas.

Gráfico 11: Distribución de la variable nº retrasos en últimos dos años respecto de la variable estatus del crédito

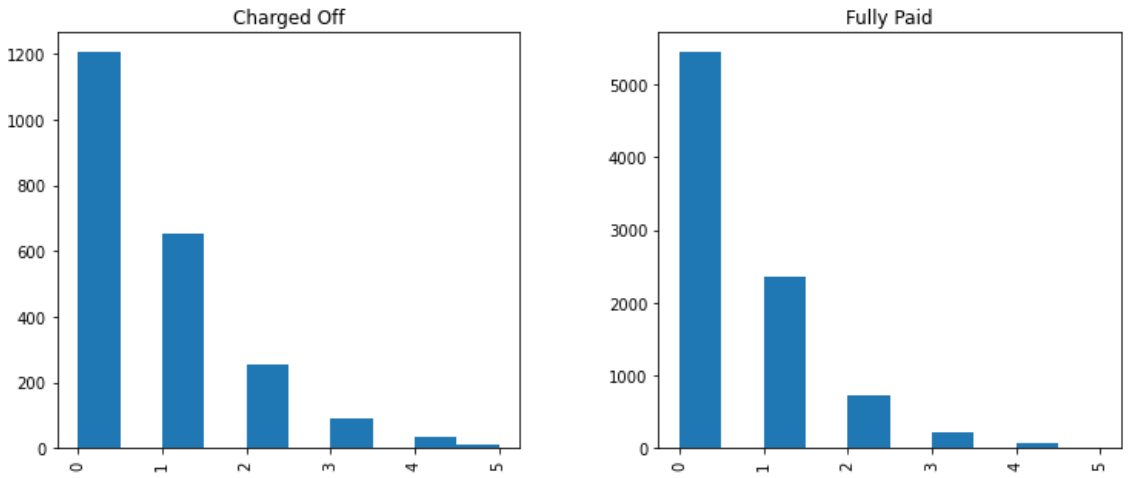


Fuente: Elaboración propia

La anterior variable presenta diferencias en su distribución según los casos sean positivos o negativos (Gráfico 11). Vemos que aquellas personas con estatus *Fully Paid* tienen, en

su mayoría, 0 retrasos en los últimos 2 años, con algunas excepciones entre los valores 1 y 5. Por otra parte, en el gráfico de la izquierda, más disgregado para mostrar mejor la distribución, observamos que hay un número significativo de observaciones en las que ha habido retrasos en los últimos 2 años. Por tanto, esta variable si puede ser relevante a la hora de implementar nuestros modelos.

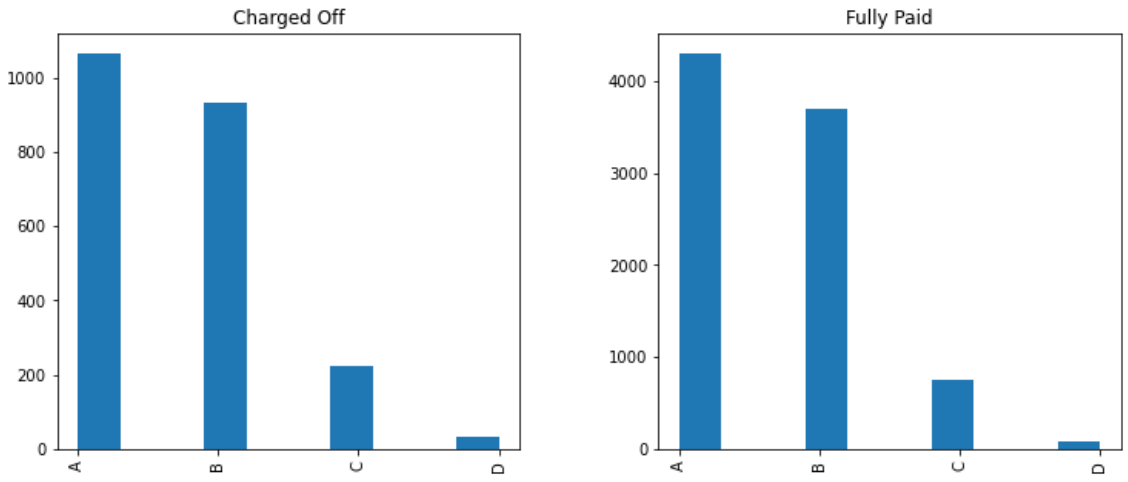
Gráfico 12: Distribución de la variable nº solicitudes de préstamo en últimos 6 meses respecto de la variable estatus del crédito



Fuente: Elaboración propia

En el (Gráfico 12) se aprecia un número progresivamente menor de solicitudes de préstamo anteriores para los negativos. Por tanto, las personas que tienen estatus *Charged Off* solicitan un mayor número de préstamos. Bien sea porque los solicitan y se los rechazan o bien porque necesitan recibir dinero más a menudo. Puede que esta variable sea relevante en los modelos, aunque las diferencias son sutiles.

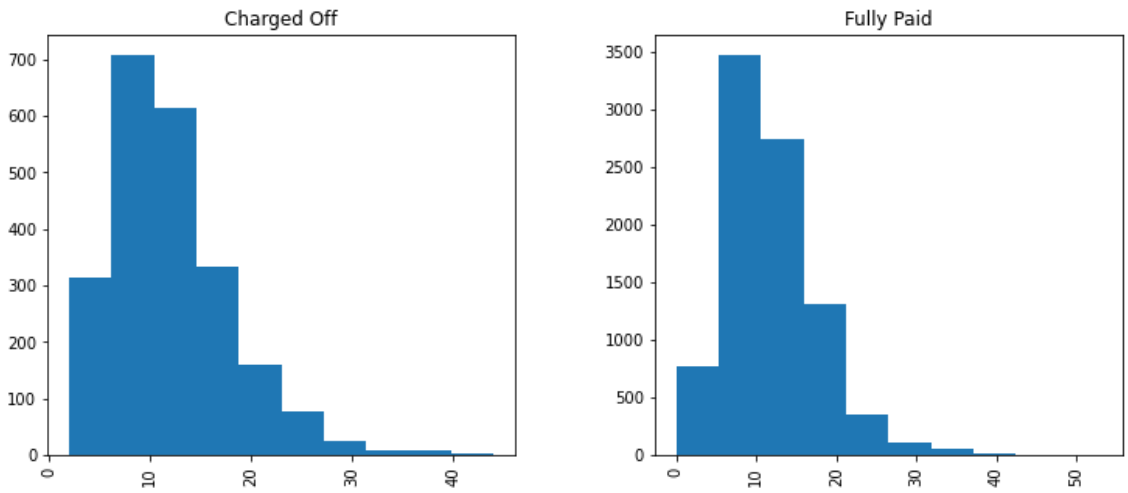
Gráfico 13: Distribución de la variable meses desde el último retraso respecto de la variable estatus del crédito



Fuente: Elaboración propia

Según este gráfico (Gráfico 13), no se observan diferencias de distribución entre los positivos y los negativos (excepto por una pequeña mayor proporción de C y D en los que tienen estatus *Charged Off*), por lo que es posible que esta variable no sea relevante a la hora de modelar. Aunque nunca hay que descartar esta opción, pues las combinaciones entre variables podrían jugar un papel relevante.

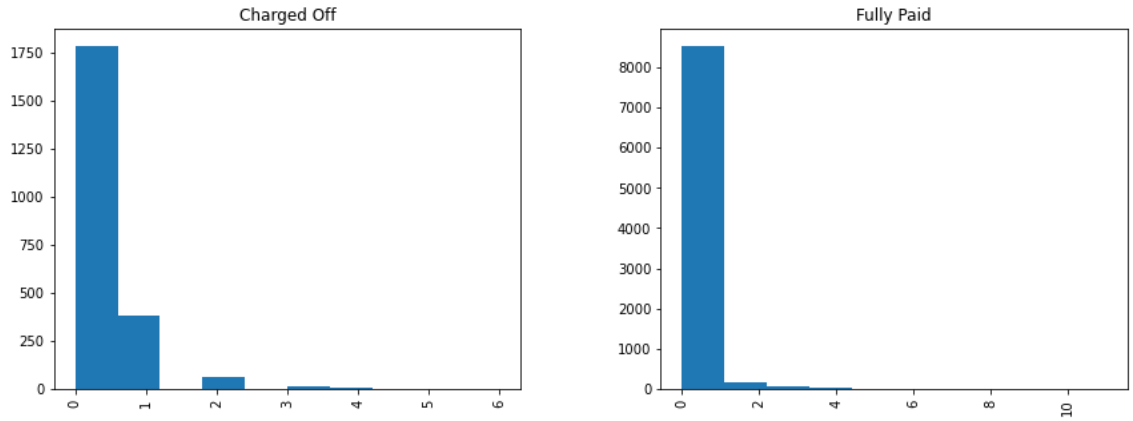
Gráfico 14: Distribución de la variable nº de líneas de crédito abiertas respecto de la variable estatus del crédito



Fuente: Elaboración propia

El gráfico anterior (Gráfico 14) no muestra diferencias muy significativas entre los positivos y los negativos, por lo que entendemos que el número de líneas de crédito abiertas no será muy relevante. Es importante tener en cuenta que este análisis preliminar de la relación entre las variables y la variable *target* puede no ser muy exacto.

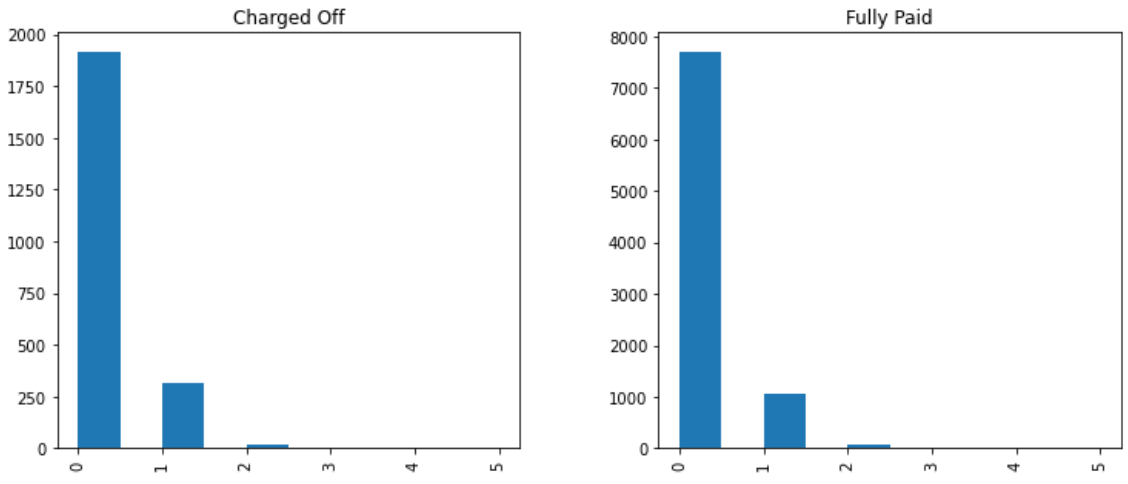
Gráfico 15: Distribución de la variable nº de impagos en los ficheros públicos respecto de la variable estatus del crédito



Fuente: Elaboración propia

A continuación, vemos el gráfico (Gráfico 15) que representa los impagos reconocidos públicamente. A pesar de que la mayoría de los valores son 0 tanto para positivos como para negativos, vemos que los *Defaults* tienen una mayor tasa de impagos para los valores del 1 al 4. Esto indica que los prestatarios que han tenido impagos en el pasado tienen más probabilidad de caer en impago en el futuro. Esta variable tiene potencial para ser relevante.

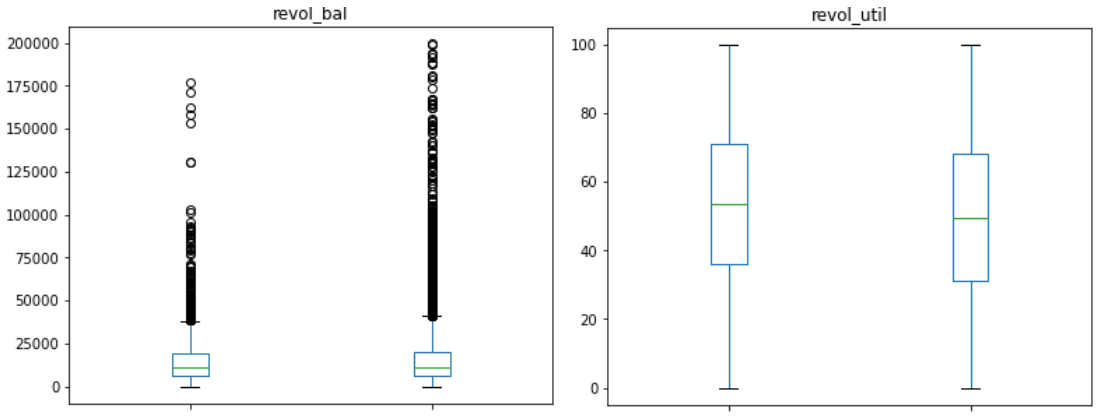
Gráfico 16: Distribución de la variable nº de declaraciones públicas de quiebra respecto de la variable estatus del crédito



Fuente: Elaboración propia

Aunque en principio están bastante relacionadas las variables de nº de declaraciones públicas de quiebra y nº de impagos en los ficheros públicos según estatus del crédito, pues toda declaración de quiebra es también una declaración de impago en el registro público, vemos que para la segunda de ellas (Gráfico 16) las distribuciones son mucho más similares que para la primera. Es posible que no estén tan correlacionadas entre si como se pudiera pensar en un principio.

Gráfico 17: Distribución de las variables cantidad en créditos *revolving* y porcentaje de utilización respecto de la variable estatus del crédito

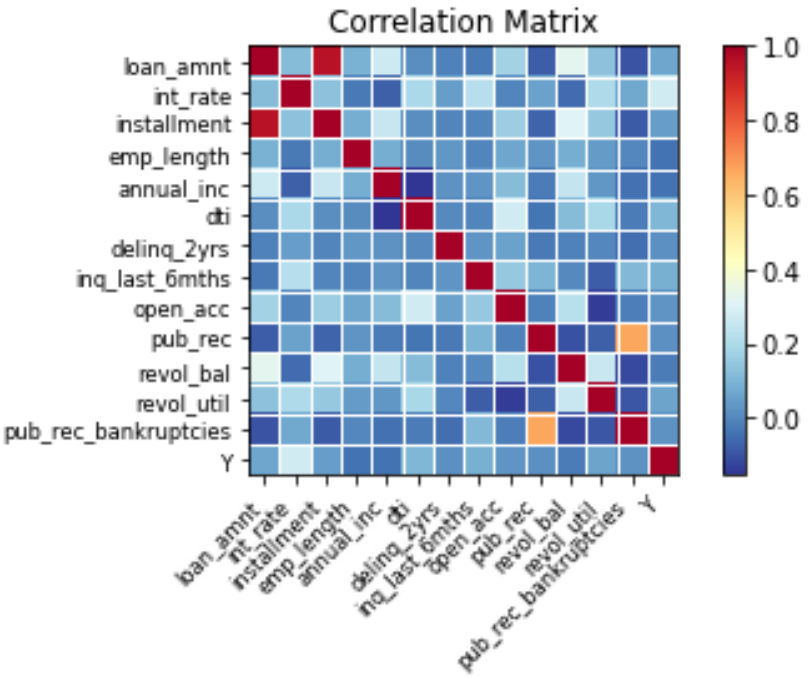


Fuente: Elaboración propia

Por último, procedemos a analizar las variables de créditos *revolving*. La primera de ellas, a la izquierda (Gráfico 17) se refiere al total de sus créditos *revolving*; mientras que la segunda, a la derecha (Gráfico 17), se refiere al porcentaje utilizado sobre la cuantía total. Hacemos un inciso para explicar lo que es un crédito *revolving*. Pues bien, este tipo de créditos son los otorgados por parte de una entidad financiera como una línea de crédito. En este préstamo, se establece una cuantía máxima de la que se puede disponer a voluntad del prestatario. Cuando se realiza una retirada de efectivo, el importe de la línea de crédito se reduce en dicha cuantía; y cuando se produce un ingreso, el tope de la línea de crédito se amplía. Por este tipo de créditos se paga una cuota fija mensual además de un porcentaje de interés por la cuantía utilizada. Este tipo suele tener un mínimo de entre el 3% y el 5% (Anónimo, Periódico El Economista, 2022). Dicho esto, vemos que la distribución de la cuantía total (izquierda) no presenta variaciones significativas entre los positivos y los negativos; aunque sí vemos que el porcentaje utilizado con respecto de estas líneas de crédito es más alto para los *Default*. Esto nos indica que las personas con un crédito en estatus *Charged Off* suelen llevar más al límite sus líneas de crédito resultando en un endeudamiento mayor y, por lo tanto, una mayor probabilidad de hacer *Default*.

En relación con el análisis anterior, presentamos la matriz de correlaciones (Tabla 18) entre todas las variables numéricas del *dataframe*, sin variables *dummies*, y la variable objetivo (Esto se debe a que el *dataframe* preparado para usarse en modelos tiene unas dimensiones que no permiten su correcta visualización en este tipo de gráficos). Se puede apreciar que no hay correlaciones muy fuertes entre las variables, con alguna excepción. Como mencionamos antes, hay una correlación muy fuerte entre la cuantía total del crédito y la cuota mensual del mismo. También es importante mencionar la correlación, también mencionada en párrafos anteriores, entre el número de impagos declarados en documento público y el número de declaraciones públicas de quiebra. Encontramos también una correlación positiva entre la variable *target* y el tipo de interés. Esto se debe a que cuanto más crece el valor del tipo de interés, más probabilidad hay de que la variable *target* tenga valor positivo (1). Aparte de las mencionadas, no hay correlaciones muy fuertes entre las variables, habiendo algunas de ellas cercanas al 0.

Tabla 18: Tabla de correlaciones entre las variables



Fuente: Elaboración propia

2.4 Aplicación de modelos

Comenzamos ya la sección del trabajo dedicada a la explicación, implementación y extracción de conclusiones de cada uno de los modelos. Recordamos, como se explicó al comienzo del proyecto, que nuestro problema es de Clasificación, por lo tanto, nuestros modelos deberán ser Modelos Predictivos Supervisados de Clasificación. El primero que desarrollamos es la Regresión Logística (*Logit*), puesto que además de los resultados del modelo, aporta datos muy útiles para determinar la relevancia de las variables (*P-Values*).

2.4.1 Modelo de Regresión Logística (*Logit*)

Hoy en día, los modelos de la familia *Logit* se usan en gran número de ciencias, como la economía. Esta función se empezó a usar en torno al Siglo XIX, en los ámbitos de la

química y la demografía, para el estudio de ciertas reacciones y el crecimiento de la población respectivamente. Esta función goza de numerosas ventajas, entre las que están la flexibilidad de su interpretación, al depender esta de la definición de las propias variables; y la simplicidad en su cálculo y aplicación. Sin embargo, es cierto que no tener una interpretación específica del modelo nos conduce a otras implicaciones, como el hecho de que esta función no tiene asociada una distribución reconocida de probabilidad como puede tener el modelo *Probit*. (Elena Martínez Rodríguez, 2008)

La función logística tiene como propiedad ser una función monótona creciente acotada entre 0 y 1. Gráficamente, representa una curva sinusoidal, donde la función presenta un crecimiento lento para valores cercanos a 0, para luego comenzar un crecimiento similar al exponencial. Por último, se vuelve a ralentizar su crecimiento hasta alcanzar su cota máxima, 1. (Elena Martínez Rodríguez, 2008)

El modelo de Regresión Logística, a pesar de su nombre, es un modelo lineal de Clasificación. Este modelo sólo sirve para cuando la variable *target* es binaria, como es nuestro caso, siendo los 1 préstamos con estatus *Charged Off* y los 0 préstamos con estatus *Fully Paid*. Es decir, la Regresión Logística es un método de regresión que permite predecir la probabilidad de una variable cualitativa binaria a partir de una o varias variables cuantitativas. Esta probabilidad es usada para clasificar las observaciones dependiendo del valor que tome. (Joaquín Amat Rodrigo, 2016)

El *Logit* no modela directamente la variable binaria, Y, si no el logaritmo de los *odds* de Y usando la función logística. Los *odds* son una función que calcula la probabilidad de que ocurra un evento, en nuestro caso que el préstamo sea *Charged Off*, entre 1 menos la probabilidad de que ocurra el evento, es decir, la probabilidad de que no ocurra el evento. La razón por la que no se usa la variable dependiente, si no una función de esta es que, si así fuera, no estaría garantizado que el resultado de la parte derecha de la ecuación estuviese en el intervalo [0,1]. Todo esto conforma el lado izquierdo de la ecuación utilizada en los modelos *Logit* (Elemento 19). Por otra parte, tenemos los predictores, es decir, las variables independientes. Al haber más de una variable predictora, estamos hablando de un análisis multivariante, Regresión Logística Múltiple.

Elemento 19: Fórmula de la ecuación *Logit*

$$\ln\left(\frac{P(Y)}{1-P(Y)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + K + \beta_k X_k$$

Fuente: Elaboración propia

A continuación, procedemos a aplicar este modelo para nuestros datos. La creación del modelo se hace a través de un programa en Spyder (Python 3.9) en el que se llevan a cabo distintos pasos (Apéndice 3). En primer lugar, es importante establecer que el uso que vamos a hacer de este modelo es más explicativo que predictivo, pues lo vamos a usar principalmente para dar una idea de la relevancia de las variables según los *p-values* que se extraigan del modelo. Lo llevamos a cabo a través de un paquete de *Python* llamado *statsmodels*. Dentro de este paquete usamos la función *logit* para realizar nuestro modelo, que será un *Full-model*, es decir, que tendrá en cuenta todas las variables de nuestro *dataframe* (recordemos que estamos usando el *dataframe* tratado con las variables cualitativas convertidas en variables *dummies*). Tras ajustar el modelo, estos son los valores que extraemos (Las variables que se extrajeron a partir de la variable *addr_state* no están incluidas en la ilustración):

Tabla 20: Resumen del modelo *Logit* donde se pueden observar tanto los *p-values* como los coeficientes

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-4.7026	0.662	-7.103	0.000	-6.000	-3.405
loan_amnt	1.054e-05	1.96e-05	0.536	0.592	-2.8e-05	4.91e-05
int_rate	0.0999	0.008	12.457	0.000	0.084	0.116
installment	-5.218e-06	0.001	-0.008	0.993	-0.001	0.001
emp_length	-0.0215	0.007	-3.151	0.002	-0.035	-0.008
annual_inc	-2.139e-06	7.8e-07	-2.742	0.006	-3.67e-06	-6.1e-07
dti	0.0157	0.003	4.916	0.000	0.009	0.022
delinq_2yrs	0.0059	0.030	0.200	0.842	-0.052	0.064
inq_last_6mths	0.1314	0.029	4.582	0.000	0.075	0.188
open_acc	0.0064	0.005	1.255	0.209	-0.004	0.016
pub_rec	0.0372	0.055	0.674	0.500	-0.071	0.145
revol_bal	-3.013e-06	1.68e-06	-1.798	0.072	-6.3e-06	2.72e-07
revol_util	0.0033	0.001	2.712	0.007	0.001	0.006
pub_rec_bankruptcies	0.0542	0.084	0.644	0.520	-0.111	0.219
OWN	0.2545	0.084	3.029	0.002	0.090	0.419
RENT	0.3798	0.059	6.413	0.000	0.264	0.496
credit_card	0.2765	0.293	0.943	0.346	-0.298	0.851
debt_consolidation	0.3132	0.289	1.084	0.279	-0.253	0.880
home_improvement	0.4955	0.303	1.638	0.101	-0.097	1.088
house	-0.4296	0.490	-0.877	0.380	-1.390	0.530
major_purchase	0.4756	0.329	1.444	0.149	-0.170	1.121
medical	0.2288	0.375	0.611	0.541	-0.506	0.963
moving	0.5704	0.405	1.410	0.159	-0.223	1.363
other	0.3911	0.304	1.285	0.199	-0.205	0.988
renewable_energy	2.1838	0.740	2.949	0.003	0.733	3.635
vacation	-0.0516	0.432	-0.119	0.905	-0.899	0.796
B	-0.0156	0.055	-0.283	0.777	-0.124	0.092
C	0.1936	0.105	1.851	0.064	-0.011	0.399
D	0.4356	0.229	1.903	0.057	-0.013	0.884
Prestamo_Largo	0.5528	0.136	4.062	0.000	0.286	0.819

Fuente: Elaboración propia

Los efectos marginales en modelos lineales representan el cambio que produce en la variable dependiente Y un cambio de una unidad en la variable independiente Xi. En la tabla que vemos arriba estos valores están representados en la columna *coef* (Tabla 20) por lo que, si tienen un valor negativo, esto significará que, cuanto más alto sea el valor de esa variable Xi, más baja será la probabilidad de *Default*. Sin embargo, al no estar en un modelo lineal, el efecto marginal depende de los distintos valores que tomen las variables explicativas. Por tanto, estos coeficientes son orientativos del efecto que tiene cada variable Xi en la variable Y, pero no son valores exactos.

La columna más importante (Tabla 20) es la referente a los *p-values*, que corresponde con la columna $P>|z|$. El nivel de significatividad mínimo para considerar a una variable como relevante varía dependiendo del estudio, pero nosotros tomaremos el estándar (0.05). Además, se consideran más significativos progresivamente a medida que bajan de los umbrales de 0.05 (C), 0.01 (B) y 0.001 (A). Antes de entrar a los valores concretos, explicamos que significan estos *p-values* y su importancia. Pues bien, lo primero que hay que comprender es que el *p-value* es el resultado de la realización de un contraste de hipótesis para cada una de las variables. Al hacer este contraste de hipótesis asumimos que la hipótesis nula, Ho, es que la variable Xi no tiene ninguna influencia o correlación con la variable Y; y que la hipótesis alternativa H es que la variable Xi sí condiciona a la variable Y. De esta forma, el *p-value* expresa la probabilidad, por lo que siempre estará en el rango [0,1], de obtener el valor de la prueba estadística utilizada aceptando la hipótesis nula. Esto supone que, cuando decimos que aceptamos un nivel de significación mínimo de 0.05, lo que realmente afirmamos es que rechazamos la hipótesis nula con una confianza del 95%. Por tanto, cuanto menor sea el *p-value*, más confianza tenemos de que esa variable es significativa.

Una vez aclarado el significado, procedemos a analizar los valores que nos ha dejado nuestro modelo en *Python*. La variable *loan_amnt*, yendo por orden, tiene un *p-value* de 0.592, un valor muy superior al máximo permitido, por lo que no tenemos la suficiente confianza, ni de lejos, como para afirmar que esta variable es significativa a la hora de predecir Y. En relación con la anterior, la variable *installment*, lejos de tener confianza suficiente como para afirmar su significación, el *p-value* llega al 0.99, indicando que esta variable será claramente no relevante.

A diferencia de estas, la variable *int_rate* presenta un *p-value* de 0.00 (A), es decir, estamos completamente seguros de que esta variable influencia la variable dependiente. Además, aunque, como ya dijimos, los efectos marginales no se pueden tomar de forma exacta para algoritmos de este tipo, vemos que hay una relación positiva (0.099) entre esta variable y la Y. Por tanto, cuanto más alto sea el tipo de interés, más probabilidad de tener estatus *Charged Off*. Esto confirma que los prestatarios a los que se les carga un tipo de interés inferior tienen realmente menos riesgo de *Default*

La variable *emp_length* tiene un *p-value* de 0.002 (B), por lo que tenemos prueba suficiente como para considerarla relevante. Además, como era de esperar, vemos un coeficiente negativo, por lo que corroboramos que cuanta mayor es la antigüedad laboral, más probabilidad hay de *Fully Paid*, ya que esto otorga un plus de estabilidad financiera.

Las dos siguientes variables *anual_inc* y *dti* tienen *p-values* de 0.006 (B) y 0.00 (A) respectivamente. Ambas variables se refieren de forma diferente a los ingresos del prestatario, indicando que esto es un dato relevante a la hora de predecir impagos. La primera tiene una relación negativa, mientras que *dti* tiene un coeficiente positivo, ya que el *debt to income* influye negativamente a la capacidad de hacer frente a las obligaciones del préstamo. Además, observamos que el coeficiente de *anual_inc* es muy próximo a 0, ya que, aunque la variable si es relevante, una variación de 1 euro no supone un gran cambio en la probabilidad de *Default*.

La variable *inq_last_6mths*, con un *p-value* de 0.00 (A), es claramente significativa, con un impacto positivo en la probabilidad de tener estatus *Charged Off*. Por tanto, aceptamos que los prestatarios que solicitan un mayor número de préstamos en el periodo previo a recibir uno, tiene una probabilidad más alta de *Default*.

Por otra parte, las variables de *pub_rec* y *pub_rec_bankruptcies* no son significativas, ya que tienen *p-values* de 0.50 y 0.52 respectivamente. En las variables de *revol_bal* y *revol_util* observamos un fenómeno curioso, puesto que la primera de ellas no es significativa (*p-value* = 0.072), mientras que la segunda sí (*p-value* = 0.07 (B)). Esto significa que lo relevante a la hora de predecir *Defaults* no es la cantidad recibida en préstamos *revolving*, si no si dichos prestatarios llevan muy al límite a esas líneas de crédito. El signo positivo de su coeficiente nos confirma que, a mayor porcentaje utilizado de las líneas de crédito, menor probabilidad de *Fully Paid*, y mayor de *Default*.

De las variables *dummies* derivadas de la variable *home_ownership*, *OWN* y *RENT*, ambas son significativas, con *p-values* de 0.002 (B) y 0.00 (A) respectivamente. Esto, unido a que ambas tienen coeficientes positivos, quiere decir que las tres opciones (*mortgage*, *own* y *rent*) presentan relevancia. De hecho, los coeficientes son bastante grandes (además de positivos), por lo que tendrán un papel importante a la hora de predecir los *Defaults* y *No Defaults*.

De las variables *dummies* procedentes de la variable *purpose* tan solo una tiene un *p_value* dentro de los límites aceptables establecidos: *renewable_energy*. Esto simplemente nos indica que el propósito para el cual sea requerido el dinero no es un factor fundamental en general. Por otra parte, y en cuanto a las variables *dummies* derivadas de *addr_state*, vemos que, aunque la mayoría no son significativas, hay algunas excepciones. En concreto: *AR*, *HI* y *MD* (siendo flexibles y aceptando *p-values* de hasta 0.06). Estas tres variables corresponden a prestatarios en los estados de Arkansas, Hawái y Maryland. No logro encontrar, a pesar de su significación, razones que justifiquen la importancia de estas tres variables, ni en términos de PIB per cápita (donde ocupan los puestos 45, 18 y 13 respectivamente en el ranking de Estados Unidos), ni en términos de PIB.

Las siguientes variables, *B*, *C* y *D* son el resultado de la descomposición de la variable *mths_since_last_delinq*. De ellas, tan solo *D* (la referente a personas con impagos previos al préstamo en los 3 meses anteriores) tiene un *p-value* que podemos considerar como dentro de los límites válidos (0.057 (C)). Como era de esperar, el coeficiente es positivo, lo que significa que los prestatarios que cometieron impagos en los 3 meses previos al crédito objeto de estudio tienen mayor probabilidad de un *Default*.

Por último, tenemos una variable con mucho peso y un *p-value* de 0.00 (A): *Prestamo_Largo* (variable binaria derivada de la variable *term*). El marcado carácter positivo de su coeficiente nos indica que los prestatarios con una maduración del crédito de 60 meses en vez de 36 tienen una probabilidad mayor de tener estatus *Charged Off*, y por tanto suele haber mayor porcentaje de *Defaults*.

2.4.2 Modelo K-Nearest-Neighbours (KNN)

El KNN es una técnica de aprendizaje supervisado que puede utilizarse en tareas tanto de regresión como de clasificación, aunque en este caso será, como ya hemos dicho, de clasificación. Este método, de forma muy básica, dado un nuevo individuo, busca entre el conjunto de *train* cuales son los individuos más cercanos, clasificando al nuevo en función de la mayoría de sus vecinos. Por tanto, este modelo no intenta construir un modelo interno general, si no que basa su predicción en un simple voto. En este voto, toman partido los K vecinos más cercanos. La elección optima del valor de K depende por completo en el *dataset* del que dispongamos. (Scikit Learn Nearest Neighbours Classification, 2022)

Comenzamos por tanto con el modelo, que se llevará a cabo a través de un programa de Python (Apéndice 3). En primer lugar, debemos separar los datos entre el conjunto de entrenamiento y el conjunto de *test*. Para ello, realizamos un *Split* aleatorio, a través de una semilla aleatoria, en el que el 30% de los datos quedarán asignados al conjunto de *test*. Debemos parar y hacer un inciso sobre esto, pues al tener entre manos un problema al borde de ser no balanceado, las cosas se complican. Cuando hablamos de un problema no balanceado, nos referimos a que la variable *target* está muy lejos de ser equitativa. Por ejemplo, un problema en el que la variable objetivo sea si una moneda saca cara o no será un problema balanceado, pues la proporción de unos y ceros será bastante 50 / 50. Sin embargo, si la variable es si la moneda cae de canto o no, estaremos ante un problema no balanceado, pues el porcentaje de unos será muy bajo. En nuestro caso, estamos más bien en un escenario de 80% / 20%. Para curarnos en salud, lo tratamos como un problema no balanceado a la hora de hacer la partición. Esto significa que vamos a hacer un *Split* aleatorio estratificado, es decir, que asegure guardar la proporción que tiene la variable *target* en los datos antes de la partición.

Cabe en este punto preguntarse si es necesaria esta partición entre *Train* y *Test*, ya que al realizar el *K-folds*, se podría evitar. Sin embargo, realizar esta partición tiene sus ventajas. Principalmente, el hecho de separar una parte de los datos que no se incluyen en el proceso de entrenamiento hace que, al medir la capacidad predictiva de los modelos, se evite un *bias* innecesario, pues que los datos sobre los que se mide la predicción hayan sido usados

para entrenar el modelo trastoca los resultados. Por tanto, realizaremos esta separación para entrenar tanto este, como los modelos que vienen a continuación.

Tras esto, creamos un *Pipeline*, una sucesión de N pasos definidos previamente. En nuestro caso, estos pasos consisten en: Primero, preprocesamos los datos que van a entrar al modelo. Esto es especialmente relevante aquí, pues este modelo basa toda su predicción en las distancias entre observaciones, por lo que las escalas de las distintas variables distorsionan la información. Por ejemplo, con el resto de las condiciones iguales, dos variables X2 y X3, con una unidad más en ingreso mensual y una unidad más en tipo de interés respectivamente, estarán a la misma distancia de la variable X1. Para solucionar esto, escalamos las variables. Segundo, hacemos una selección de variables. En este caso, vamos a empezar haciendo una simple discriminación de las variables que menos información aporten al modelo. Tercero, introducimos el clasificador de KNN dentro del modelo gracias al paquete *sklearn*.

Nuestro modelo es una búsqueda aleatoria para encontrar los parámetros que mejoren una medida concreta de *performance*. Dentro de esta búsqueda aleatoria introducimos el *pipeline* y las distribuciones de los hiperparámetros que son, en este caso, el porcentaje de variables que se van a conservar dentro del modelo, y el número de *K* vecinos que se van a usar en el modelo. Para encontrar los valores óptimos de estos parámetros le damos a la búsqueda aleatoria una serie de valores. Primero, el número de iteraciones que se van a realizar de esa búsqueda aleatoria. Segundo, el método de *scoring*, que determinará como se establece que modelos son mejores que otros. Dos buenos métodos de *scoring* son el *f1* y el *auc*. El *f1* se utiliza para combinar las medidas de precisión y *recall* en un solo número, siendo muy útil en modelos donde la variable *target* es binaria. Se computa calculando la media armónica entre estas dos medidas, asumiendo que ambas nos importan de igual manera. La precisión se refiere al porcentaje de casos que ha acertado el modelo, mientras que el *recall* se refiere al porcentaje de casos positivos (*Defaults*) que se han predicho correctamente. El método *auc* utiliza la curva *ROC* para después calcular la zona entre la línea y = x y la curva *ROC*.

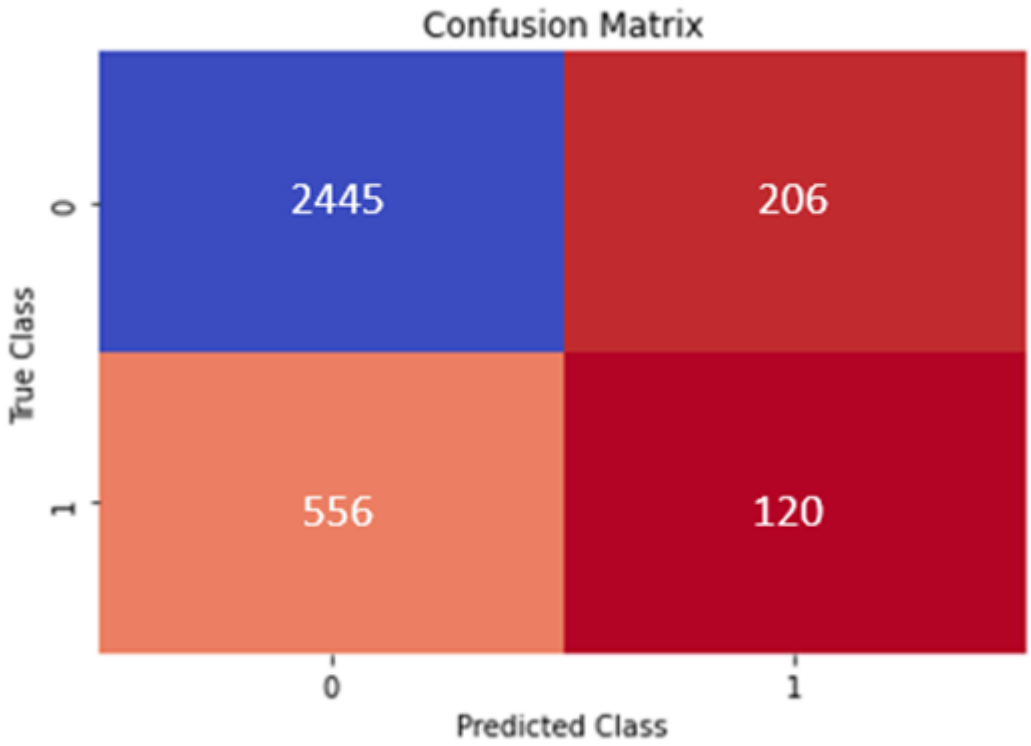
En nuestra búsqueda aleatoria de los parámetros que mejoren el modelo vamos a usar una estrategia de *Cross-Validation*. En concreto, vamos a hacer uso de *stratified K-folds*. Este método parte, para cada iteración, el conjunto de datos en *K* partes haciendo *trainings* en los que cada vez una de las *K* partes hace de conjunto de *test*. En nuestro caso, además,

estas *K* particiones deben estratificarse para que se conserven las proporciones de *Defaults* que tienen los datos en su conjunto.

En nuestro caso, usaremos como valores 20 iteraciones y *K-folds* = 5. En total, se llevarán a cabo un total de 100 *fits*. Este proceso nos devuelve el resultado de los hiperparámetros. En este caso, se usarán los 3 vecinos más cercanos a un punto para clasificarlo. Además, descartamos el 62% de las variables, ya que el algoritmo considera que hay muchas variables que no aportan información relevante (puede ser debido a las dos variables categóricas desglosadas en variables *dummies*, causando que 50 de las 77 variables sean derivadas de la variable *addr_state*). Sacamos los *scores* del modelo para ver cuales considera que son las variables más relevantes. La variable *int_rate* es sin duda la más relevante, con un *score* de 642.25, lo cual cuadra con lo obtenido en el modelo *Logit*; por otra parte, la variable *Prestamo_largo* tiene un *score* de 296.12, por lo que será bastante relevante también; en cuanto a la variable *dti*, tiene el tercer mayor *score*, 108.21; la variable *inq_last_6mths* tiene un *score* de 52.04; *loan_amnt* de 36.52; *RENT* 34.27; *revol_util* 28.70; *installment* 20.11; *emp_length* 16.33; *credit_card* 13.24; *debt_consolidation* 11.48; *anual_inc* 10.71. Realmente, las únicas contradicciones entre estos *scores* y los *p-values* extraídos del *Logit* son para la pareja de variables *loan_amnt* y *Installment* que, como ya dijimos, están muy interrelacionadas (en el *Logit* ambas tenían *p-values* superiores a 0.5).

Después de esto, usamos el conjunto de *test* para hacer predicciones y así comprobar el desempeño de nuestro modelo. Para ello, usamos diversas medidas de *performance*. En primer lugar, la precisión de nuestro modelo ha sido del 72%. En cuanto al *recall*, este fue del 77%, aunque el *recall* de los *Defaults* es bastante bajo, siendo este cercano al 18%. Esto es un problema, ya que el modelo es capaz de identificar de forma bastante precisa a los *No Defaults*, pero tiene dificultades para identificar los *Defaults* correctamente.

Tabla 21: Matriz de confusión resultante de nuestro modelo de KNN



Fuente: Elaboración propia

En la matriz (Tabla 21), vemos como, de las 3327 observaciones que había en el conjunto de *test*, 2445 (en torno a un 73.5%) fueron negativos bien predichos, es decir, se predijo bien el 92.2% de los negativos. Sin embargo, solo 120 *Defaults* (18%) se pudieron llegar a predecir correctamente, mientras que 556 *Defaults* se pasaron por alto. Como vemos, el modelo no predice de forma adecuada, ya que un 18% de precisión a la hora de predecir *Defaults* no es suficiente. Sin embargo, viendo este modelo como un método de filtraje posterior a la decisión de aceptación del crédito (ya que estamos usando los datos de los créditos que efectivamente fueron aceptados por *Lending Club*), podemos otorgarle algo de utilidad. Como veíamos al principio de este trabajo, la proporción de *Defaults* ronda el 20,3%, y nuestro modelo, según la matriz de confusión vista (Tabla 21), estaría descartando 326 créditos con un 37% de *Defaults*, por lo que estaría descartando una parte de los créditos ya aceptados según el método de *Lending Club*, bajando el porcentaje de *Defaults* totales de 20,3% a 18,5%. Es decir, aplicar este modelo como segundo filtro, habría bajado los *Defaults* en 1,8 puntos porcentuales.

2.4.3 Modelo de Decission Tree Classifier

Los árboles de decisión son técnicas de aprendizaje supervisado. Para un conjunto de datos con N variables, el algoritmo va creando particiones recursivas de ese espacio de N dimensiones en espacios que no se solapan. Son un método muy generalizado, ya que son muy sencillos de utilizar y más aún de entender. Se basan en un conjunto sucesivo de condiciones que se aplican de forma jerárquica. De esta manera, la decisión final sobre un individuo se forma siguiendo estas condiciones. El algoritmo comienza construyendo el árbol desde la "raíz", para después ir añadiendo particiones que mejoren la homogeneidad de los nodos que crea, dividiendo las observaciones hasta que todas las observaciones de un nodo sean de la misma clase. Esta homogeneidad que determina los *splits* que se van llevando a cabo se puede medir según el índice de Gini (Elemento 22). El valor de este índice va de 0, cuando todas las observaciones son de la misma clase, hasta (m-1)/m siendo m el número de categorías de la variable *target*. En nuestro caso, la variable *target* tiene 2 clases, por lo que el índice de Gini estaría entre 0 y 0.5. Por tanto, el árbol de decisión irá realizando los *splits* de acuerdo con las particiones que tengan un índice de Gini más cercano a 0.

Elemento 22: Fórmula del Índice de Gini

$$G = |1 - (\Sigma_{k=1}^{k=n-1} (X_{k+1} - X_k)(Y_{k+1} + Y_k)|$$

Fuente: Elaboración propia

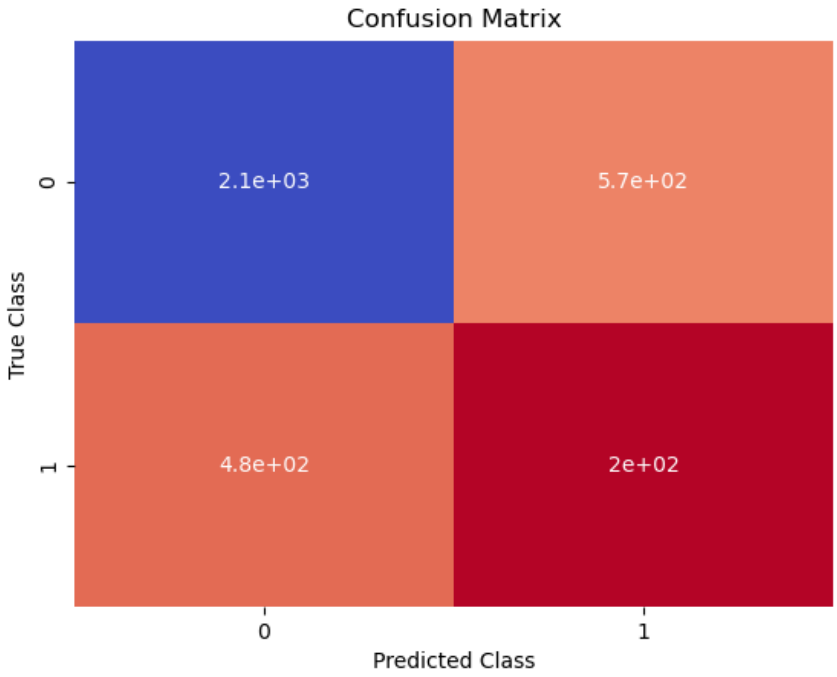
Comenzamos por tanto con el modelo, que se llevará a cabo a través de un programa de Python (Apéndice 3). En primer lugar, debemos separar los datos entre el conjunto de entrenamiento y el conjunto de *test*. Para ello, realizamos un *Split* aleatorio, a través de una semilla aleatoria, en el que el 30% de los datos quedarán asignados al conjunto de *test*. Nuestro modelo es una búsqueda aleatoria para encontrar los parámetros que mejoren una medida concreta de *performance*. Dentro de esta búsqueda aleatoria introducimos el *pipeline* y las distribuciones de los hiperparámetros que son, en este caso, el porcentaje de variables que se van a conservar dentro del modelo, y el número de *estimators* que se van a usar en el modelo. Para encontrar los valores óptimos de estos parámetros le damos a la búsqueda aleatoria una serie de valores. Primero, el número de iteraciones que se van

a realizar de esa búsqueda aleatoria. Segundo, el método de *scoring*, que determinará como se establece qué modelos son mejores que otros. De forma muy similar al modelo anterior, mantenemos *f1 y auc* como posibles medidas de precisión de nuestro modelo. También mantenemos el uso de *Cross-Validation* y los *stratified K-folds*.

Tras este proceso, obtenemos el resultado de hiperparámetros que se han considerado óptimos. En este caso, descartaremos el 13% de las variables, ya que el algoritmo considera que hay algunas variables que no aportan información relevante

Tras esto, usamos el conjunto de *test* para hacer predicciones y así comprobar el desempeño de nuestro modelo. Para ello, usamos las medidas de *performance* mencionadas previamente. En primer lugar, usando *f1* como medida de precisión de nuestro modelo a la hora de encontrar el mejor modelo, la precisión de este ha sido, para los casos positivos (*Default*), del 25,8%, mientras que al *recall* fue del 29,4%. En cuanto a los casos de *No Default*, estas medidas suben hasta el 81% y 78% respectivamente. Esto se ve claramente representado en la matriz de confusión (Tabla 23). De las 3327 observaciones del conjunto de *Test*, 2651 observaciones corresponden a créditos *No Default*, habiéndose clasificado correctamente 2080; mientras que de las 676 observaciones que son *Default*, tan solo 199 se han clasificado correctamente. Esto es un problema, ya que el modelo es capaz de identificar de forma bastante precisa a los *No Defaults*, pero tiene dificultades para identificar los *Defaults* correctamente. Si hacemos uso de la medida de precisión *auc*, obtendríamos mayor efectividad a la hora de detectar *No Defaults*, pero a costa de mucha menor efectividad a la hora de detectar *Defaults*, por lo que no nos compensa.

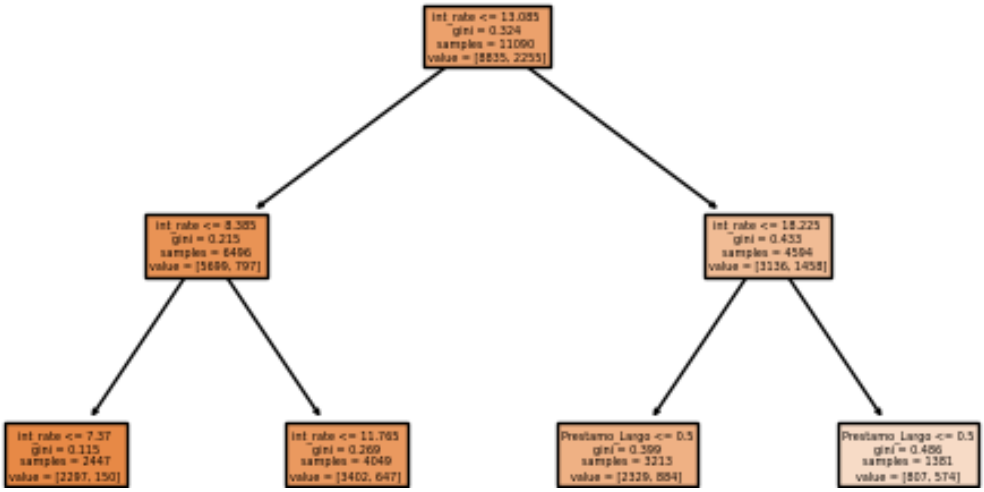
Tabla 23: Matriz de confusión resultante de nuestro modelo de Árbol de decisión



Fuente: Elaboración propia

A pesar de que el modelo no es especialmente efectivo a la hora de predecir, especialmente en el caso de los *Defaults*, sí tiene utilidad como modelo descriptivo (Gráfico 24). Como podemos observar en el gráfico del árbol de decisión, simplificado para mostrarnos las primeras divisiones, los nodos finales van de izquierda a derecha de un color naranja más oscuro a un color más claro. Estos colores representan la proporción de *Defaults* según las divisiones realizadas, es decir, los nodos de color más oscuro tienen una menor proporción de *Defaults* que los nodos más claros. Por otra parte, vemos que las variables según las cuales se realizan estas divisiones son principalmente 2 en estas primeras divisiones: por una parte, el *int_rate* según el cual se hacen las dos primeras divisiones, indicando que las observaciones con valores pequeños en esta variable suelen tener un porcentaje menor de posibilidades de ser *Default*. En conceto, la primera división se realiza según si la observación tiene un valor menor o mayor a 13 en la variable *int_rate*. En el lado derecho, la tercera división se lleva a cabo a través de la variable *Prestamo_Largo*, indicando que el hecho de que el préstamo sea a 3 o a 5 años es también determinante.

Gráfico 24: Gráfico del Árbol de decisión que separa las observaciones en módulos según el índice de



Gini

Fuente: Elaboración propia

2.4.4 Modelo de Random Forest Classifier

El *Random Forest* es un tipo de *ensemble* de modelos, esto es, una combinación de modelos con el fin de mejorar la precisión y el rendimiento de las predicciones, ya que los errores producidos por uno de los modelos incluidos en dicho *ensemble* se van mitigando y diluyendo al mezclarse con los resultados del resto de los modelos. De esta forma, se debería apreciar una disminución de la varianza en la estimación del *ensemble*. Esto se debe a que los árboles por sí solos tienen la tendencia de producir *overfitting* y suelen tener una varianza alta. Sin embargo, al combinar un número de árboles, se logra reducir dicha varianza, y muchos de los errores se van compensando, dando lugar a un modelo con mejor rendimiento y precisión. El *RandomForestClassifier* de la librería *sklearn* lleva a cabo un *fit* de varios árboles de decisión con distintas sub-muestras de los datos, para así reducir el *overfitting* y mejorar la precisión predictiva.

Existen varias formas de realizar un modelo de clasificación con un *Random forest*, ya que las decisiones se pueden tomar mediante un sistema de votación entre los diferentes árboles, o mediante la realización de una media aritmética de dichos modelos. En nuestro

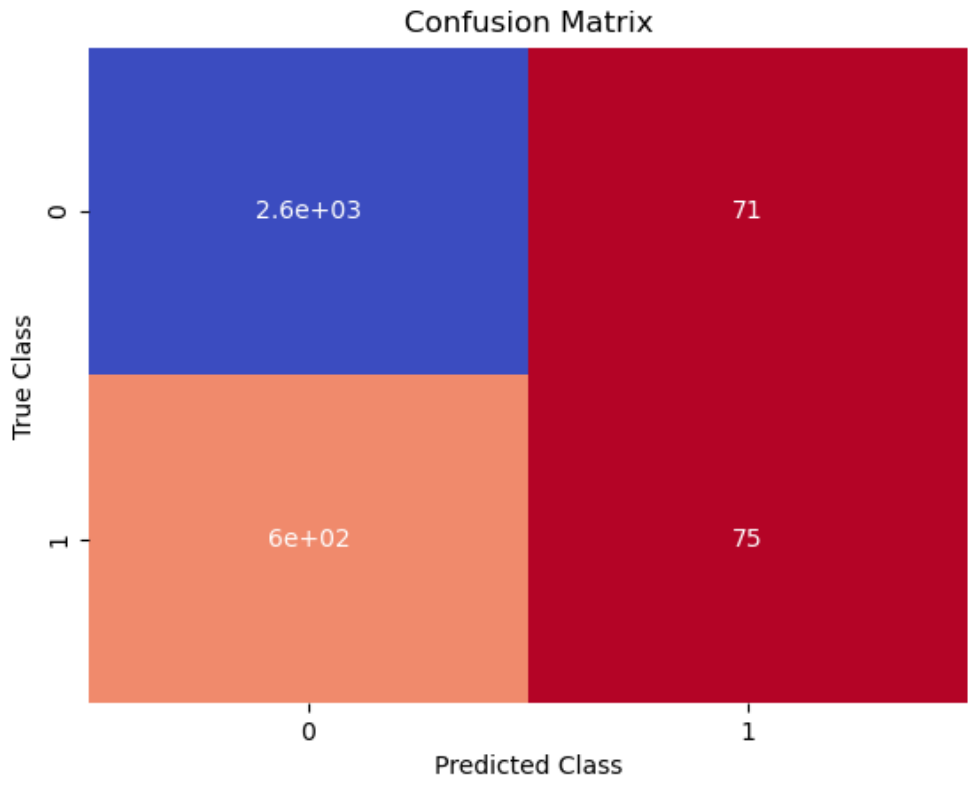
caso, al usar la librería de *sklearn*, se lleva a cabo haciendo la media aritmética de las probabilidades de cada predicción.

A la hora de entrenar a nuestro modelo, seguimos usando la medida *f1*, ya que es ideal para medir el desempeño de modelos con una variable *target* binaria. Dentro de nuestro *Pipeline*, y de forma similar a la que habíamos descrito anteriormente, incluiremos como hiperparámetros: *classifier__n_estimators* el cual determina el número de árboles de decisión que optimizaría el desempeño de nuestro *ensemble*; el *classifier__class_weight=('balanced',),* para intentar que la muestra sea balanceada; y el *classifier__max_features*, que fijará el número máximo de variables que pueden integrarse dentro de cada uno de los árboles de decisión que conforman el modelo. En este caso, se han seleccionado el 56% de las variables, y un número de árboles de decisión de 30.

Al fijarnos en el desempeño de nuestro modelo, observamos varias cosas. En primer lugar, vemos que la *accuracy* del modelo llega al 80%, lo que nos muestra que, de las observaciones cuya predicción ha sido *Default*, casi el 52% han sido clasificadas correctamente. Sin embargo, el *recall* del modelo es inferior al que habíamos obtenido con el árbol de decisión, ya que solo se han logrado predecir correctamente el 11% de los *Defaults*. Por otra parte, para los casos de *No Default*, se ha obtenido una precisión del 81%, con un *recall* del 97%. Para un mayor análisis nos fijamos en la matriz de confusión obtenida (Tabla 25). Como podemos observar, de las 2651 observaciones No *Default*, 2580 se predijeron correctamente; aunque de las 676 observaciones *Default*, tan solo se lograron clasificar como tal 75. Estos resultados pueden ser poco alentadores, ya que superan por poco la regla de la mayoría (regla por la que un modelo no es válido si se obtiene mejor *accuracy* prediciendo todas las observaciones como si fuesen de la clase mayoritaria), en concreto, con la regla de la mayoría, obtendríamos un *accuracy* de entorno al 79%, mientras que usando este modelo obtenemos un 80%. Sin embargo, debemos tener en consideración el tipo de *dataset* que estamos utilizando. Todas las observaciones incluidas corresponden a créditos que efectivamente fueron otorgados por la entidad *Lending Club*, incluyendo aquellos que acabaron en *Default*. Por tanto, si se hubiera usado este modelo predictivo antes de otorgar los créditos, se podría haber mejorado la proporción de *No Defaults*. Esto se debe a que, simplemente no otorgando los créditos clasificados por el modelo como *Defaults*, estaríamos eliminando una parte de la población en la que la proporción de *Defaults* supera el 51%, es decir, unos 31

puntos porcentuales por encima de la proporción de *Defaults* en nuestro *dataset*. Por tanto, aunque los resultados son sin duda mejorables, podrían llegar a tener una utilidad práctica para disminuir los impagos.

Tabla 25: Matriz de confusión resultante de nuestro modelo de *Random Forest*



Fuente: Elaboración propia

2.4.5 Modelo de *AutoML*

Llegamos finalmente a una parte crucial del trabajo. En las páginas anteriores hemos ido probando y midiendo distintos modelos, algunos incluso con un enfoque más explicativo que predictivo. Sin embargo, el modelo de *AutoML* intentará obtener el modelo que minimice el error y, por tanto, tenga una mejor capacidad predictiva. Los procesos de *AuroML* tienen la peculiaridad de que, para llegar a estos modelos optimizados, usan una

herramienta de búsqueda, la Optimización Bayesiana, mucho más potente y eficiente que los métodos simples más comunes, como pueden ser el *RandomSearch* o el *GridSearch*.

La optimización bayesiana es un método de optimización que tiene su principal utilidad a la hora de mejorar la precisión cuando estamos en un modelo de caja negra, donde lo que nos importa es simplemente la performance del modelo. En líneas generales, es un proceso que se va alimentando a sí mismo, y cuyo objetivo es minimizar un *target*, que puede ser, por ejemplo, una medida de *performance* del modelo, como el *f1* que hemos estado usando en los modelos anteriores, y que es una buena medida para *datasets* con una variable *target* binaria.

Más en concreto, es un proceso iterativo en el que se realizan una serie de pasos repetidamente hasta que se está satisfecho con el resultado (Gráfico 26). Estas iteraciones van añadiendo información que, tras varias iteraciones, hace que el proceso sea más preciso. De esta forma, no te verías obligado a sacrificar una enorme cantidad de tiempo y de precisión como en el caso del *RandomSearch*.

Gráfico 26: Gráfico explicativo de los pasos del proceso de Optimización Bayesiana

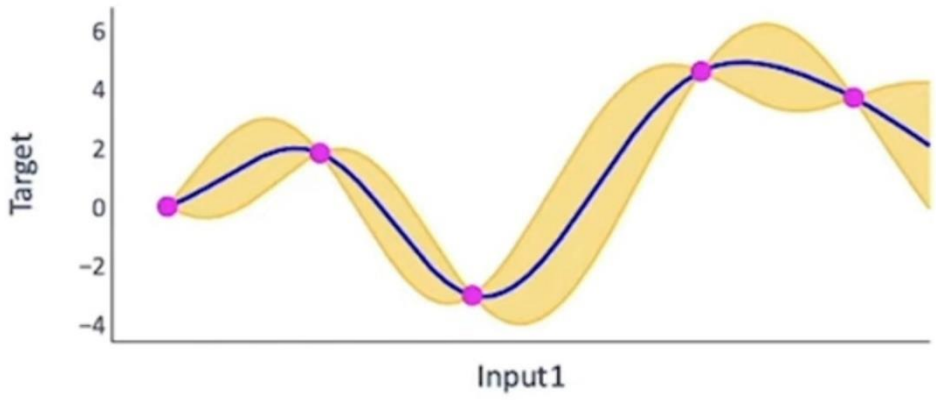
Nº Paso	Paso	Breve explicación
1.-	Conjunto inicial de *input*	Tomamos varios valores aleatorios del input para tener unos puntos de referencia
2.-	*Gausin Process Regressor*	Crea un rango de incertidumbre por el que puede pasar la función
3.-	Función de Adquisición	Crea una distribución de probabilidad de los posibles valores que optimizan la función
4.-	Identificar el vector de *input* que optimiza la función de adquisición o minimiza la entropía de esta	Proceso de decisión interno para elegir el siguiente punto de *input*

5.-	Repetimos 2, 3 y 4 hasta que obtengamos el punto óptimo o se acabe el *budget*	Vamos aumentando los datos y mejorando los resultados del proceso

Fuente: Elaboración Propia

En primer lugar, se lleva a cabo un proceso de *sampling* inicial. A partir de estas muestras se entrenan modelos con distintos valores de los hiperparámetros, que nos servirán para entrenar un *Gaussian Process Regressor*. Este es el resultado de entrenar un gran número de funciones de regresión en las que cada una tiene distintos hiperparámetros propios, como los *kernels*, dando lugar, no a una línea de la función, si no a un área que representa la incertidumbre del modelo a través de la desviación típica de todas las funciones de regresión. Al tener solo unos pocos puntos de nuestra función, no podemos predecir con perfecta exactitud la función que corresponde a esos puntos, pero sí un rango de funciones entre la que estaría contenida (Gráfico 27).

Gráfico 27: Gráfico ejemplificados de un *Gaussian Process Regressor*

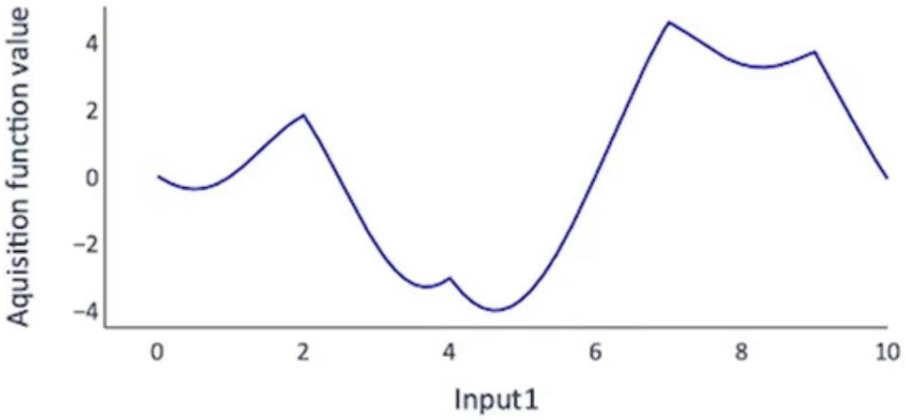


Fuente: Youtube Video Bayesian Optimization (Bayes Opt): Easy explanation of popular hyperparameter tuning method

A partir de esto, se calcula la función de adquisición, que se usará para identificar el siguiente valor de los hiperparámetros que será evaluado, el cual minimizará dicha función de adquisición. Esta función representa la distribución de probabilidad de encontrar valores en los que se podría encontrar un mejor resultado. Por ejemplo (Gráfico 28), si lo que queremos es minimizar nuestro *target*, la función de adquisición tomará una forma parecida a la frontera inferior de nuestra área de incertidumbre, ya que es un caso

muy básico. Decimos parecida porque la función de adquisición depende también de un hiperparámetro, *kappa*, que regula el rango de incertidumbre permitido en dicha función. En este caso, vemos que la función de adquisición (Gráfico 28) concuerda perfectamente con la parte baja del rango de incertidumbre del Gráfico 27. Esto se debe a que se ha tomado una *kappa* de 1, por lo que la función de adquisición (Elemento 29) quedaría como la media del *Gaussian Process Regressor* menos su desviación típica, es decir, la frontera inferior del rango de incertidumbre. Sin embargo, este valor de *kappa* podría ser modificado para tener una función de adquisición con mayor incertidumbre, pero resultados potencialmente mejores.

Gráfico 28: Gráfico ejemplificados de una función de adquisición que buscaría minimizar el *target* del Gráfico 27



Fuente: Youtube Video Bayesian Optimization (Bayes Opt): Easy explanation of popular hyperparameter tuning method

Elemento 29: Fórmula de la función de adquisición

$$a_{(x)} = f_{(x)} \pm (\mathrm{K} * \sigma_{f(x)})$$

Fuente: Elaboración propia

De esta forma, se añade un nuevo vector de valores de hiperparámetros al proceso, haciendo que, en la siguiente iteración, dicho nuevo *input* esté incluido en la función de adquisición que determinará el siguiente *input*. A medida que se añaden puntos al

Gaussian Process Regressor, el rango de incertidumbre va cambiando, y por tanto también la función de adquisición, de forma que el modelo se va retroalimentando. Es muy importante entender que el proceso no solo saca los puntos en los que hay una mayor posibilidad de optimizar el *target*, si no que intenta también sacar puntos que reduzcan de manera significativa la entropía en nuestro *Gaussian Process Regressor*, para ir acotando los lugares en los que encontrar esos puntos que sí optimizan.

Estos pasos descritos se repetirán hasta que la función de adquisición no ofrezca alternativas que minimicen más el target, o hasta que se acabe el *Budget* de evaluación. Llegados a este punto, tendríamos el modelo que minimiza cierto *target*, que, como hemos dicho antes, puede ser una medida de *performance*. Esto quiere decir que ya tenemos los valores óptimos de los hiperparámetros de nuestro modelo.

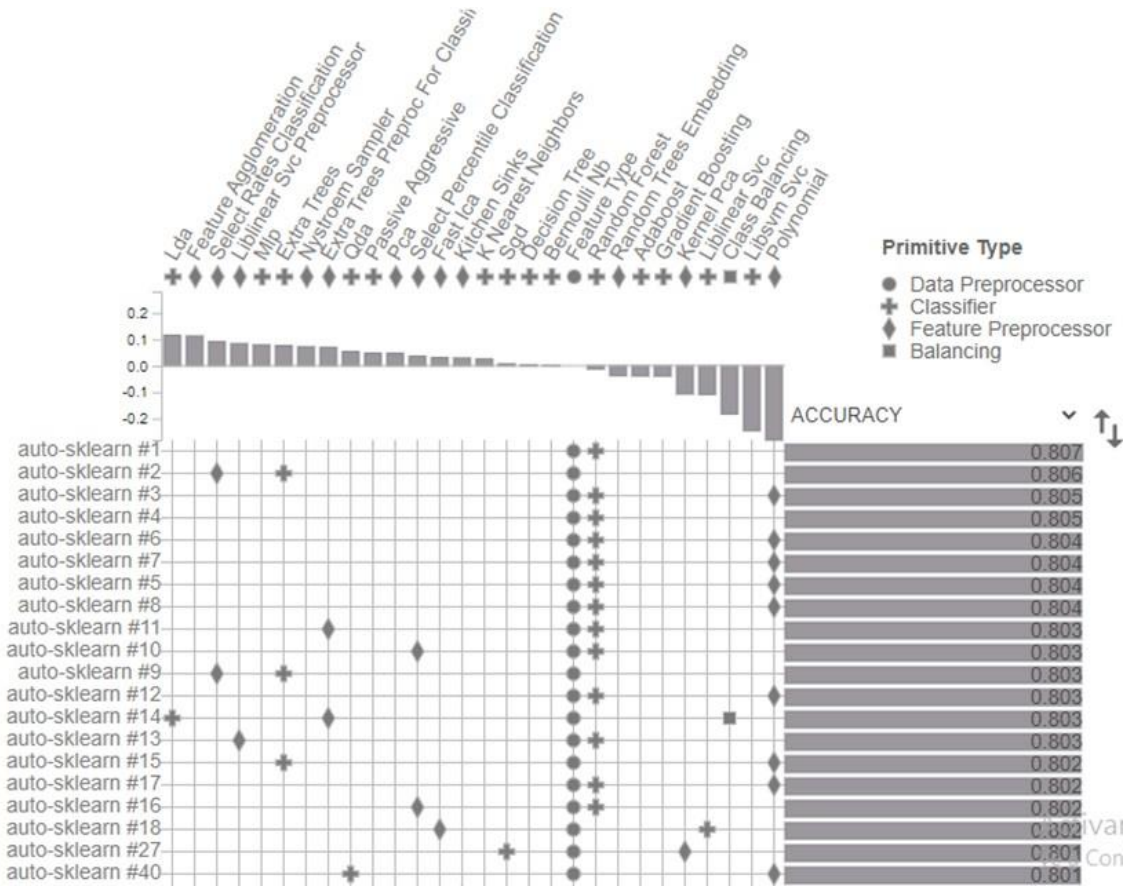
Una vez entendemos el proceso de optimización que se lleva a cabo dentro de un proceso de modelado de *AutoML*¸ procedemos a explicar cómo funcionan este tipo de modelos. En concreto, vamos a hacer uso de la librería *auto-sklearn* en Python, que pertenece al conjunto de librerías de *scikit learn*. De forma muy básica, el *AutoML* es el proceso por el que se automatizan tareas propias del desarrollo de modelos de *Machine Learning*, y que, de otra forma, implicarían el uso de una gran cantidad de tiempo y un componente alto de prueba y error a la hora de comparar un gran número de modelos. Estos modelos de *AutoML* se caracterizan por tener una gran eficiencia y productividad, manteniendo la calidad del modelo. En definitiva, hace el *Machine Learning* más accesible a las personas ya que es mucho más sencillo de usar (realiza las tareas de preprocesamiento de datos, extracción de características y selección iterativa de modelos óptimos con gran eficiencia y con muy pocas líneas de código), y mejora la eficiencia del proceso de búsqueda de modelos óptimos, y, por tanto, acelera la investigación y aplicaciones en temas de *ML*.

Esta vez realizaremos nuestro modelo a través de un cuaderno de Jupyter en Google Colab. (Apéndice 4). Google Colab. aporta importantes ventajas a la hora de programar y ejecutar cuadernos de código. En primer lugar, Google Colab. permite instalar todos los paquetes y librerías necesarios para ejecutar modelos de *AutoML* con *scikit-learn*, lo cual suponía un problema en nuestro entorno Spyder (Python 3.9). Además, permite el uso del servidor de Google, habilitando, como poco, 13GB de memoria RAM y unos 110GB de disco para ejecutar el código. Pues bien, en este cuaderno de Jupyter, comenzamos instalando los paquetes de *AutoML* de la librería de *scikit-learn*. Tras esto, importamos el

dataset y lo procesamos de igual manera que en el resto de modelos, realizando posteriormente un *Split* entre *train* y *test*. A la hora de programar el modelo propiamente dicho, usamos *autosklearn.classification.AutoSklearnClassifier()*, otorgándole un tiempo de búsqueda de 60 minutos.

Durante este tiempo, se buscan, a través de la optimización bayesiana de hiperparámetros, los modelos que tengan una mayor *accuracy*, así como los modelos que puedan formar un ensemble lo más preciso posible. En nuestro caso, hemos obtenido los siguientes resultados en cuanto a cuáles son los mejores modelos en términos de *accuracy* (Gráfico 30): nuestro mejor modelo obtenido tiene una *accuracy* del 80,7%, y consiste en un modelo de *Random Forest*. Vemos que, aunque haya mucha variedad de modelos, dentro de los 20 mejores, la gran mayoría comparte el hecho de ser *Random forest*.

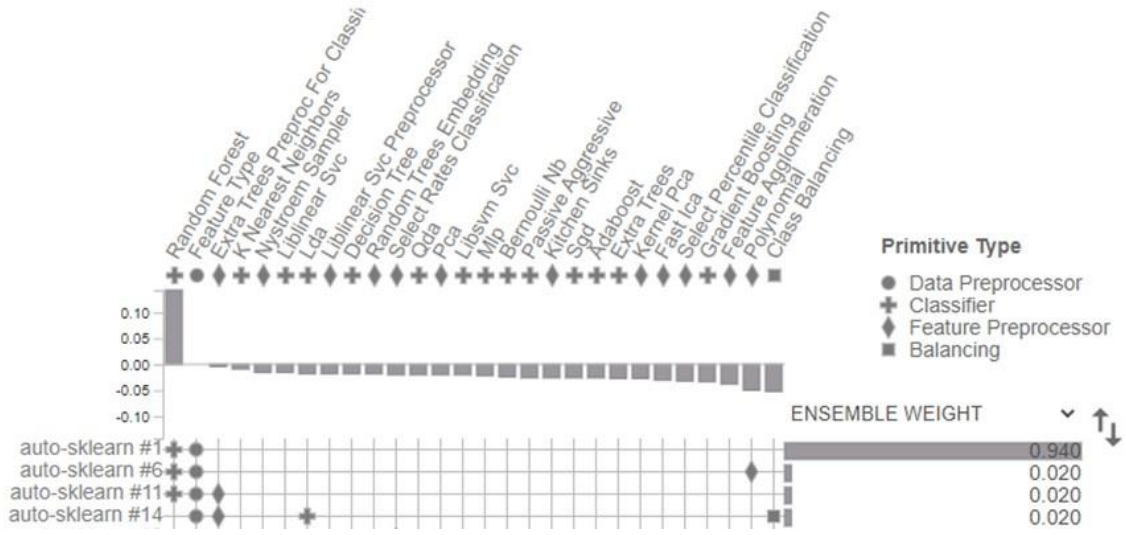
Gráfico 30: Gráfico que muestra los 20 mejores modelos encontrados por nuestro algoritmo de *AutoML*, así como el tipo de algoritmo



Fuente: Elaboración propia

Esta *accuracy* obtenida por los modelos vistos arriba se podría mejorar a través de la creación de *ensembles* de modelos. El *AutoML* nos enseña también cual sería el mejor *ensemble* de modelos (Gráfico 31). En este caso, se trata de la unión de 4 modelos con pesos muy distintos, en concreto, haríamos uso de los modelos 1, 6, 11 y 14, los cuales también salen representados entre los 20 mejores modelos. El modelo 1 tiene un peso del 94%, mientras que los otros 3 modelos tienen un peso de 2% cada uno. Es importante destacar que, de los 4 modelos que forman este ensemble, 3 de ellos son modelos de *Random* Forest y uno de LDA. Sin embargo, tras aplicar el *AutoML* a nuestros datos tras entrenar el modelo, obtenemos una *accuracy* algo inferior al 80%.

Gráfico 31: Gráfico que muestra los modelos que forman parte del *ensemble*, así como sus pesos dentro de este



Fuente: Elaboración propia

3.- ANÁLISIS DEL RESULTADO

Tras realizar varios modelos, toca ahora el análisis de los resultados obtenidos. Nuestro análisis tenía una vertiente explicativa, con el fin de comprender que características de los prestatarios eran más relevantes e influyentes para generar un *Default*.

En el modelo de *Logit*, encontrábamos con valores de *p-value* que indicaban representatividad en las variables *int_rate*, *emp_length*, *dti*, *inq_last_6mths*, *Prestamo_Largo*, *mths_since_last_delinq* y *RENT*. Es decir, los prestatarios que tenían un crédito otorgado con un mayor tipo de interés tendrán más proporción de *Defaults*, lo cual tiene bastante sentido, ya que, un mayor tipo de interés representa un mayor riesgo de impago. La variable referente a la antigüedad en el trabajo, *emp_length*, tiene un impacto negativo, en cuanto a que a mayor antigüedad menor posibilidad de *Default*, queriendo esto decir que las personas con seguridad en el ámbito laboral son mejores pagadores. La variable *dti*, *debt to income*, influye de forma positiva, puesto que tener un mayor porcentaje de deuda respecto de tus ingresos causa una subida en la probabilidad de *Default*. Esto es comprensible, ya que el hecho de tener más deuda con respecto a tus ingresos reduce la capacidad de hacer frente a las obligaciones de dicha deuda. La variable *inq_last_6mths* indica el número de peticiones de préstamos solicitados en los últimos 6 meses, por lo que, como hemos observado, tiene una influencia positiva, ya que las personas que solicitan más préstamos suelen tener problemas económicos y, por tanto, caer más en *Default*. La variable *Prestamo_Largo* indica si la longitud del préstamo es de 5 años (y no de 3 años). Esto nos demuestra que los créditos otorgados a mayor plazo tienen mayor posibilidad de recaer en *Default*. La variable *mths_since_last_delinq* también aparece como relevante, lo que nos quiere decir que las personas que han tenido incidentes previos de impago tienen más posibilidad de *Default* en el futuro. Por último, la variable *RENT* tienen un coeficiente positivo, lo que quiere decir que los prestatarios que necesitan solicitar créditos para hacer frente a los pagos de su alquiler suelen recaer en *Default* con mayor probabilidad.

En el modelo de KNN, las variables con *scores* más altos fueron *int_rate*, *emp_length* y *dti*. Esto nos cuadra bastante con lo que habíamos observado en el modelo de *Logit*, indicando que nuestro análisis explicativo es consistente. En el modelo de *Decision Tree Classifier* las variables que se consideraron más relevantes fueron la de *int_rate* y *Prestamo_Largo*, coincidiendo en parte con el modelo de *Logit* y en parte con el modelo

de KNN. Comprobamos, por tanto, que de las numerosas variables que teníamos en un principio, tan solo 7 de ellas son verdaderamente relevantes y describen las cualidades de las personas con mayor probabilidad de generar un *Default*.

Por otra parte, nuestro análisis también pretendía tener una vertiente predictiva. Por este lado, los resultados son menos eficaces, ya que, a la hora de predecir *Defaults* (positivos), nuestros modelos muestran una dificultad extra. Aun así, con varios de ellos hemos logrado superar la regla de la mayoría, la cual, en este caso, presenta un listón muy alto, ya que, con la regla de la mayoría, el *accuracy* ya rozaría el 80%, siendo este un valor muy elevado. En cualquier caso, el mejor modelo en términos de *accuracy* sería el de *Random Forest*, lo cual coincide con el análisis de *AutoML* realizado al final. El modelo de *Random Forest* alcanzó una precisión de en torno al 80%, superando ligeramente la regla de la mayoría, a diferencia de otros como el KNN que tiene una calidad predictiva claramente inferior, aunque logra predecir un número mayor de positivos a costa del *accuracy*.

4.- CONCLUSIONES

Como conclusiones a este trabajo, empezamos diciendo que la parte explicativa del mismo ha sido exitosa, llegando a conclusiones claras de cuáles son las variables más determinantes para segregar entre positivos y negativos (visto en los párrafos superiores), mientras que la parte predictiva necesita de una explicación adicional a los resultados. Con esto queremos decir que, aunque los resultados en términos de *accuracy* y *recall* no son excelentes, sí tienen, desde mi punto de vista, una utilidad práctica.

Para explicar esta utilidad debemos reiterar la procedencia y cualidades de los datos usados para el análisis. Nuestros datos proceden de los préstamos otorgados en los últimos años por parte de la plataforma *Lending Club*. Es decir, los préstamos analizados son únicamente aquellos que ya habían sido aprobados por los algoritmos y analistas de dicha plataforma. Por tanto, cuando nos referimos a superar la regla de la mayoría, realmente estamos haciendo referencia a refinar aún más la discriminación realizada por esta plataforma.

Sin embargo, con algunos de los modelos hemos conseguido superar esta regla de la mayoría, por lo que, en esos casos, los resultados son positivos. De hecho, si se hubiera realizado un segundo filtro de los créditos ya aceptados por *Lending Club* con nuestro modelo de *Random Forest* se habría reducido la proporción de *Defaults*. Si bien también es cierto que el *int_rate* establecido es acorde al riesgo estimado de *Default*.

Queda mucho camino por recorrer en cualquier investigación, ya que ninguna es exhaustiva. Esto quiere decir que hay futuras líneas de continuación en esta línea de investigación. Una de ellas sería la inferencia de los denegados dentro del análisis, o realizar un análisis con la base de datos con, no sólo los prestatarios aprobados por la plataforma, si no también con los denegados, llevando a cabo una clasificación multiclase en vez de binaria, donde las clases no sean solo *Default* o no *Default*, si no que añadiríamos una que fuera Denegado.

5.- BIBLIOGRAFÍA

Alarcón Flores, J. B., (2014), *Modelos de minería de datos: random forest y adaboost, para identificar los factores asociados al uso de las TIC (internet, telefonía Fija y televisión de paga) en los hogares del Perú*, Lima: Universidad Nacional mayor de San Marcos.

Amat Rodrigo, J., (2016), Regresión logística simple y múltiple, *Cienciadedatos*.

Barral Varela, G., (2022), Crédito Revolving, *Expansión*.

Berástegui Arbeloa, G., (2018), *Implementación del algoritmo de los k vecinos más cercanos (k-NN) y estimación del mejor valor local de k para su cálculo*, Pamplona: Universidad Pública de Navarra.

Cantalapiedra, M., (2019), *Crowdlending: Análisis del caso español*, Madrid

Carmichael, D., (2014), *Modeling default for peer-to-peer loans,* Houston: C.T. Bauer College of Business.

Nunes, C., (2019), *Towards the improvement of decision tree learning: a perspective on search and evaluation*, Barcelona: Universitat Pompeu Fabra

Croux, C., Jagtiani, J., Korivi, T., Vulanovic, M., (2020), Important factors determining Fintech loan default: Evidence from a lendingclub consumer platform, *Journal of Economic Behavior and Organization*, 173, pp. 270-296

Emekter, R., Tu, Y., Jirasakuldech, B., Lu, M., (2015), Evaluating Credit Risk and Loan Performance in Online Peer-to-Peer (P2P) Lending, *Applied Economics*, 47, pp. 54–70.

Fernández Sánchez, D., (2019), *Creación de una Herramienta de Optimización Bayesiana en Python*, Madrid: Universidad Autónoma de Madrid.

Jagtiani, J., Lemieux, C., (2019), Do fintech lenders penetrate areas that are underserved by traditional banks?, *Journal of Economic Behavior and Organization*, 100, pp. 43-54

Jagtiani, J., Lemieux, C., (2019), The roles of alternative data and machine learning in fintech lending: evidence from the LendingClub consumer platform, *Working papers Federal Reserve Bank of Philadelphia*, 18-15

Kuhn, M., (2016), Bayesian Optimization of Machine Learning Models, *Milestones in AI, Machine Learning, Data Science, and visualization with R and Python since 2008*.

Lutkevich, B., (Año no disponible), Automated machine learning (AutoML), *TechTarget*
Mach, T., Carter, C., Slattery, C., 2014, Peer-to-Peer Lending to Small Businesses, *Finance and Economics Discussion Series 2014*, 10.

Martínez Rodríguez, E., (2008), *Logit Model como modelo de elección discreta: origen y evolución*.

Mullainathan, S., Spiess, J., (2017), Machine Learning: An Applied Econometric Approach, *Journal of Economic Perspectives*, 31.2, pp. 87-106.

Molina Morales, X., Martínez Chafer, L., Del Corte Lore, V., (Año no disponible), *Análisis del fenómeno del crowdfunding: el caso de la plataforma Verkami*, Castellón: Universitat Jaume I de Castelló.

6.- APÉNDICES

APÉNDICE 1: Programa Spyder (Python 3.9): MUESTRA1.py. Elaboración Propia

```
from random import random
# probabilidad de coger el 2%
THRESHOLD = 0.02
# abro archivo destino
with open('ACCEPTED_SAMPLE_PRE_FINAL.csv', 'wt') as target:
    # abro archivo origen
    with open('ACCEPTED_FULL.csv') as source:
        # lee y graba la cabecera
        header = source.readline()
        target.write(header)
        # voy copiando las lineas de uno a otro si se cumple la condición aleatoria
        while line := source.readline():
            if random() < THRESHOLD:
                # print(line)
                target.write(line)
```

APÉNDICE 2: Programa Spyder (Python 3.9): ANALISIS EXPLORATORIO.py. Elaboración Propia

```
path = 'C:/Users/comillas/Desktop/TFG/ANALYTICS/SCRIPTS/ACCEPTED_SAMPLE_PRE_FINAL.xlsx'

df = pd.read_excel(path)

#Creamos X con las variables que no necesitan modificación

#y le añadimos las variables que necesitan el proceso dummy

SOME_INPUTS = ['loan_amnt', 'int_rate', 'installment',

        'emp_length', 'annual_inc', 'dti', 'delinq_2yrs',

        'inq_last_6mths', 'open_acc', 'pub_rec', 'revol_bal',

        'revol_util', 'pub_rec_bankruptcies']

X = df[SOME_INPUTS]

X

dummies1 = pd.get_dummies(df['home_ownership'], drop_first=True)

X = X.join(dummies1)

dummies2 = pd.get_dummies(df['purpose'], drop_first=True)

X = X.join(dummies2)

dummies3 = pd.get_dummies(df['mths_since_last_delinq'], drop_first=True)

X = X.join(dummies3)

dummies4 = pd.get_dummies(df['term'], drop_first=True)

X = X.join(dummies4)

dummies5 = pd.get_dummies(df['addr_state'], drop_first=True)

X = X.join(dummies5)

X
```

```
#Creamos Y

TARGET = 'loan_status'

y = pd.get_dummies(df[TARGET], drop_first=False)

y = y['Charged Off']

y

#Así juntamos las X y las Y en un solo dataframe llamado df2

df2 = X.copy()

df2['Y'] = y

df2

#Ahora tenemos 2 dataframe:

#df tiene las 19 variables originales como el que tenemos en el excell

#df2 tiene las variables categóricas transformadas a traves de dummies. Total de 80 variables

# computing the correlation matrix

C = np.corrcoef(df2.D,df2.Y)

print(C)

# computing and drawing the correlation matrix

df3 = df.copy()

df3['Y'] = y

df3

corr = df3.set_index('loan_status').corr()

sm.graphics.plot_corr(corr, xnames=list(corr.columns))

plt.show()

#Aqui para hacer boxplots de una variable segun otra

df2.boxplot('loan_amnt', 'Y', grid = False, figsize = (6,5))
```

```
df2.boxplot('int_rate', 'Y', grid = False, figsize = (6,5))

df2.boxplot('installment', 'Y', grid = False, figsize = (6,5))

df2.boxplot('emp_length', 'Y', grid = False, figsize = (6,5))

df2.boxplot('annual_inc', 'Y', grid = False, figsize = (6,5))

df2.boxplot('dti', 'Y', grid = False, figsize = (6,5))

df2.boxplot('open_acc', 'Y', grid = False, figsize = (6,5))

df2.boxplot('pub_rec', 'Y', grid = False, figsize = (6,5))

df2.boxplot('revol_bal', 'Y', grid = False, figsize = (6,5))

df2.boxplot('revol_util', 'Y', grid = False, figsize = (6,5))

df2.boxplot('pub_rec_bankrupcies', 'Y', grid = False, figsize = (6,5))

#Las demas variables, se estudian con histogramas

df.hist('term', 'loan_status', grid = False, figsize = (12,5))

df.hist('home_ownership', 'loan_status', grid = False, figsize = (14,5))

df.hist('purpose', 'loan_status', grid = False, figsize = (12,5))

df.hist('addr_state', 'loan_status', grid = False, figsize = (12,5))

df.hist('delinq_2yrs', 'loan_status', grid = False, figsize = (12,5))

df.hist('inq_last_6mths', 'loan_status', grid = False, figsize = (12,5))

df.hist('mths_since_last_delinq', 'loan_status', grid = False, figsize = (12,5))

df.hist('pub_rec', 'loan_status', grid = False, figsize = (12,5))

df.hist('pub_rec_bankruptcies', 'loan_status', grid = False, figsize = (12,5))
```

APÉNDICE 3: Programa Spyder (Python 3.9): MODELOS.py. Elaboración Propia

```
import pandas as pd

from sklearn.model_selection import StratifiedKFold, train_test_split,
RandomizedSearchCV, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import f_classif, SelectPercentile

import sklearn.metrics as metrics

from sklearn.neighbors import KNeighborsClassifier

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier

from scipy.stats import randint, uniform

from sklearn.linear_model import LogisticRegression

import matplotlib.pyplot as plt

#Asi leemos el fichero csv que contiene nuestra muestra

path =
'C:/Users/comillas/Desktop/TFG/ANALYTICS/SCRIPTS/ACCEPTED_SAMPLE_PR
E_FINAL.xlsx'

df = pd.read_excel(path)

#########################################################################

#Comenzamos con el algoritmo LOGIT

from statsmodels.formula.api import logit, probit

data = X.copy()

data['Y'] = y

data
```

```
model = logit('Y ~
loan_amnt+int_rate+installment+emp_length+annual_inc+dti+delinq_2yrs+inq_last_6
mths+open_acc+pub_rec+revol_bal+revol_util+pub_rec_bankruptcies+OWN+RENT+c
redit_card+debt_consolidation+home_improvement+house+major_purchase+medical+
moving+other+renewable_energy+vacation+AL+AR+AZ+CA+CO+CT+DC+DE+FL+
GA+HI+ID+IL+IN+KS+KY+LA+MA+MD+ME+MI+MN+MO+MS+MT+NC+ND+N
E+NH+NJ+NM+NV+NY+OH+OK+OR+PA+RI+SC+SD+TN+TX+UT+VA+VT+WA
+WI+WV+WY+B+C+D+Prestamo_Largo', df2)

logit_res = model.fit()

logit_res.summary()

#########################################################################

#TEST Y TRAINING SETS

#Creamos los test y training sets decidiendo que porcentaje va a cada uno

#Estratificamos por si acaso estamos ante un dataset unbalanced

TEST_SIZE = 0.3

RANDOM_SEED = 42

X_train, X_test, y_train, y_test = train_test_split(X,
                                    y,
                                    test_size=TEST_SIZE,
                                    stratify=y,
                                    random_state=RANDOM_SEED)

#########################################################################

#MODELO KNN

#Construimos el modelo

scaler = StandardScaler()

selector = SelectPercentile(f_classif)
```

```
classifier = KNeighborsClassifier()
pipeline = Pipeline(steps=[
        ('scaler', scaler),
        ('selector', selector),
        ('classifier', classifier)])
#Nuestro modelo ES una busqueda aleatoria:
   #un pipeline con los pasos necesarios
   #la distribucion de los hiperparametros
   #numero de iteraciones
   #un metodo para medir el performance: "f1" o "auc"
   #Estrategia de CV, en este ejemplo stratified K-Fold
   #n_jobs: esto es para correr en paralelo. Si pones -1 usa todos los cores de tu ordenador. 1 solo un core (si te da problemas).
SEARCH_BUDGET = 20
K_FOLDS = 5
SCORING_METHOD = 'f1'
#SCORING_METHOD = 'roc_auc'
stratified_kfold = StratifiedKFold(n_splits=K_FOLDS)
param_dist = {
        'selector__percentile': randint(20, 100+1),
        'classifier__n_neighbors': randint(2, 10+1),
}
model = RandomizedSearchCV(pipeline,
                    param_distributions=param_dist,
                    n_iter=SEARCH_BUDGET,
```

```
                    scoring=SCORING_METHOD,
                    cv=stratified_kfold,
                    n_jobs=-1,
                    verbose=1)
model.fit(X_train, y_train);
model.best_estimator_
model.best_params_
#Extraigo del pipeline, del paso "selector" los scores que asigna a cada una de las features.
#Lo pongo en una serie con los nombres correspondientes
scores = model.best_estimator_['selector'].scores_
nombres = X.columns
pd.Series(scores, index=nombres)
#Predecimos la y cogiendo el conjunto de test
y_pred = model.predict(X_test)
#vemos metricas de la comparación entre nuestra predicción y la Y del conjunto de test
precision = metrics.precision_score(y_test, y_pred)
recall = metrics.recall_score(y_test, y_pred)
f1 = metrics.f1_score(y_test, y_pred)
cm = metrics.confusion_matrix(y_test, y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)
print(f'precision={precision:.1%} recall={recall:.1%} f1={f1:.2}')
print(f'AUC={auc:.2}')
print(cm)
report = metrics.classification_report(y_test, y_pred)
```

```
print(report)
#con esto realizamos un heatmap que nos representa la matriz de confusion
import seaborn as sns
sns.heatmap(cm, annot=True, cbar=None, cmap="coolwarm_r")
plt.title("Confusion Matrix"),
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
#########################################################################
#MODELO ARBOL DECISION
#Construimos el modelo
scaler = StandardScaler()
selector = SelectPercentile(f_classif)
classifier = DecisionTreeClassifier(ccp_alpha=0.5)
pipeline = Pipeline(steps=[
        ('scaler', scaler),
        ('selector', selector),
        ('classifier', classifier)])
SEARCH_BUDGET = 20
K_FOLDS = 5
SCORING_METHOD = 'f1'
#SCORING_METHOD = 'roc_auc'
stratified_kfold = StratifiedKFold(n_splits=K_FOLDS)
param_dist = {
        'selector__percentile': randint(50, 100+1),
```

```
        'classifier__ccp_alpha': uniform(loc=0,scale=1)
}
model = RandomizedSearchCV(pipeline,
                           param_distributions=param_dist,
                           n_iter=SEARCH_BUDGET,
                           scoring=SCORING_METHOD,
                           cv=stratified_kfold,
                           n_jobs=-1,
                           verbose=1)
model.fit(X_train, y_train);
model.best_estimator_
model.best_params_
#Predecimos la y cogiendo el conjunto de test
y_pred = model.predict(X_test)
#vemos metricas de la comparacion entre nuestra prediccion y la Y del conjunto de test
precision = metrics.precision_score(y_test, y_pred)
recall = metrics.recall_score(y_test, y_pred)
f1 = metrics.f1_score(y_test, y_pred)
cm = metrics.confusion_matrix(y_test, y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)
print(f'precision={precision:.1%} recall={recall:.1%} f1={f1:.2}')
print(f'AUC={auc:.2}')
print(cm)
report = metrics.classification_report(y_test, y_pred)
```

```
print(report)
#Con esto sacamos la matriz de confusion
import seaborn as sns
sns.heatmap(cm, annot=True, cbar=None, cmap="coolwarm_r")
plt.title("Confusion Matrix"),
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
#Con esto, sacamos un grafico del arbol de decision
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
tree.plot_tree(clf, filled=True, max_depth=2, feature_names=X.columns)
#########################################################################
#MODELO RANDOM FOREST
#Construimos el modelo
#scaler = StandardScaler()
#selector = SelectPercentile(f_classif)
classifier = RandomForestClassifier()
pipeline = Pipeline(steps=[
    #        ('scaler', scaler),
 #           ('selector', selector),
           ('classifier', classifier)])
SEARCH_BUDGET = 20
K_FOLDS = 5
```

```
#SCORING_METHOD = 'balanced_accuracy'
SCORING_METHOD = 'f1'
#SCORING_METHOD = 'roc_auc'
stratified_kfold = StratifiedKFold(n_splits=K_FOLDS)
param_dist = {
        'classifier__n_estimators': [100, 200, 300],
        'classifier__class_weight':('balanced',),
        'classifier__max_features': uniform(0.25, 0.75)
}
model = RandomizedSearchCV(pipeline,
                    param_distributions=param_dist,
                    n_iter=SEARCH_BUDGET,
                    scoring=SCORING_METHOD,
                    cv=stratified_kfold,
                    n_jobs=-1,
                    verbose=2)
model.fit(X_train, y_train);
model.best_estimator_
model.best_params_
#Predecimos la y cogiendo el conjunto de test
y_pred = model.predict(X_test)
#vemos metricas de la comparacion entre nuestra prediccion y la Y del conjunto de test
precision = metrics.precision_score(y_test, y_pred)
recall = metrics.recall_score(y_test, y_pred)
```

```
f1 = metrics.f1_score(y_test, y_pred)

cm = metrics.confusion_matrix(y_test, y_pred)

auc = metrics.roc_auc_score(y_test, y_pred)

print(f'precision={precision:.1%} recall={recall:.1%} f1={f1:.2}')

print(f'AUC={auc:.2}')

print(cm)

report = metrics.classification_report(y_test, y_pred)

print(report)

#Con esto sacamos la matriz de confusion

import seaborn as sns

sns.heatmap(cm, annot=True, cbar=None, cmap="coolwarm_r")

plt.title("Confusion Matrix"),

plt.ylabel("True Class"), plt.xlabel("Predicted Class")

plt.show()
```

APÉNDICE 4: Programa en Google Colab (Python): *AUTOML*.ipynb. Elaboración Propia

```
!pip3 install scipy

!pip3 install sklearn

!pip3 install auto-sklearn

!pip3 install PipelineProfiler

#IMPORTAMOS LOS PAQUETES

import pandas as pd

from sklearn.model_selection import StratifiedKFold, train_test_split,
RandomizedSearchCV, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import f_classif, SelectPercentile

import sklearn.metrics as metrics

from sklearn.neighbors import KNeighborsClassifier

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier

from scipy.stats import randint, uniform

from sklearn.linear_model import LogisticRegression

import matplotlib.pyplot as plt

from scipy.stats import uniform

#EL DATASET Y EL SPLIT YA LO HEMOS HECHO EN SCRIPTS ANTERIORES

#CREAMOS EL MODELO, LO ENTRENAMOS Y SACAMOS LOS RESULTADOS

!pip install --upgrade scikit-learn

import sklearn.model_selection
```

```
import sklearn.metrics
import autosklearn.classification
import PipelineProfiler
if __name__ == "__main__":
    X_train, X_test, y_train, y_test = \
        sklearn.model_selection.train_test_split(X, y, random_state=1)
    automl =
autosklearn.classification.AutoSklearnClassifier(time_left_for_this_task=3600,
metric=autosklearn.metrics.balanced_accuracy)
    automl.fit(X_train, y_train)
    y_hat = automl.predict(X_test)
    print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))
    print(automl.leaderboard())
    print(automl.show_models())
    profiler_data=PipelineProfiler.import_autosklearn(automl)
    PipelineProfiler.plot_pipeline_matrix(profiler_data)
```