Maestría en Inteligencia Artificial Aplicada

Curso: Inteligencia Artificial y Aprendizaje Automático

Tecnológico de Monterrey

Prof Luis Eduardo Falcón Morales

Adtividad de la Semana 6

Árboles de decisión y bosque aleatorio.

Nombres y matrículas de los integrantes del equipo:

- Esperón Carreón José Eduardo A01372413
- Fernández Lara Jorge A01793062
- Hernández Ramos Joel Orlando A00759664
- Romo Cárdenas Juan Carlos A00260430
- Torres Cantú David Alejandro A00818002

En cada sección deberás incluir todas las líneas de código necesarias para responder a cada uno de los ejercicios.

```
# Incluye aquí todos módulos, librerías y paquetes que requieras.
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import classification_report, make_scorer
from sklearn.model_selection import  cross_validate,  RepeatedStratifiedKFold

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import validation_curve
from sklearn.model_selection import learning_curve

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder


import matplotlib.pyplot as plt
```

Objetivo: Predecir si una persona es clasificada como confiable a la asignación de un crédito, o no lo es tomando la base de datos históricos del South_German_Credit_Data_Set.

Ejercicio-1.

Carga los datos y sustituye los nombres de las columnas del alemán al inglés de acuerdo a como se indica en la página de la UCI.

NOTA: Si lo deseas, puedes traducirlos y ponerlas en español.

Para un manejo de los datos de forma directa, se descargó la base de datos y fue incluida en un repositorio público en "GitHub", de tal forma que cualquiera pueda acceder a él, sin necesidad de incorporar el archivo al directorio

```
filename='https://raw.githubusercontent.com/JoelOrlandoHR/RANDOM/main/SouthGermanCredit.asc'
```

Para obtener una primera percepción, creamos el dataframe y visualizamos las primeras 5 filas.

```
from pandas.io.parsers.readers import read_csv
```

```
df=pd.read_csv(filename, sep=' ')
df.head()
```

	laufkont	laufzeit	moral	verw	hoehe	sparkont	beszeit	rate	famges	buerge	...	verm	alter	weitkred	wohn	bi
0	1	18	4	2	1049	1	2	4	2	1	...	2	21	3	1	
1	1	9	4	0	2799	1	3	2	3	1	...	1	36	3	1	
2	2	12	2	9	841	2	4	2	2	1	...	1	23	3	1	
3	1	12	4	0	2122	1	3	3	3	1	...	1	39	3	1	
4	1	12	4	0	2171	1	3	4	3	1	...	2	38	1	2	

5 rows × 21 columns

Considerando que los nombres de las columnas están en Alemán, para una mejor comprensión, los traduciremos a inglés y español utilizando la descripción del conjunto de datos que se encuentra en el link:

https://archive.ics.uci.edu/ml/datasets/South+German+Credit#

```
renamed_cols= ["status", "duration", "credit_history", "purpose", "amount", "savings",
                  "employment_duration", "installment_rate", "personal_status_sex",
                  "other_debtors", "present_residence", "property", "age", "other_installment_plans", "housing",
                  "number_credits", "job", "people_liable", "telephone", "foreign_worker", "credit_risk"]

renamed_cols_esp = ["Estado", "Duracion", "Historial_crediticio", "Proposito", "Importe", "Ahorros", "Duracion_Empleo",
                    "tasa", "Genero", "Otros_Acreedores", "Residencia_Actual", "Bienes", "Edad", "Otros_planes_de_credito",
                    "Tipo_de_vivienda", "Numero_de_creditos", "trabajo", "Dependientes", "Telefono", "Trabajador_extranjero"

df.columns = renamed_cols_esp

df.columns #Mostramos los nombres de las columnas para verificar que se haya realizado el cambio de nombre de forma correcta

     Index(['Estado', 'Duracion', 'Historial_crediticio', 'Proposito', 'Importe',
            'Ahorros', 'Duracion_Empleo', 'tasa', 'Genero', 'Otros_Acreedores',
            'Residencia_Actual', 'Bienes', 'Edad', 'Otros_planes_de_credito',
            'Tipo_de_vivienda', 'Numero_de_creditos', 'trabajo', 'Dependientes',
            'Telefono', 'Trabajador_extranjero', 'Riesgo_de_credito'],
           dtype='object')
```

Ejercicio-2.

Realiza una partición de los datos en el conjunto de entrenamiento del 85% y el de prueba de 15%. Los modelos se estarán entrenando con el método de validación cruzada, así que no es necesario en este paso generar el conjunto de validación. Define como la variable X a todas las variables de entrada y a la variable Y como la variable de salida.

Generamos el modelo conforme a lo especificado en las instrucciones, separando la variable resultado.

```
X = df.copy().drop(columns='Riesgo_de_credito')
y = df.copy()['Riesgo_de_credito']
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.85, random_state=5)
```

Ejercicio-3.

Como una primera aproximación (baseline) realizarás las siguientes transformaciones mínimas para generar los primeros modelos. En la misma página de la UCI se indica el tipo de variable de cada uno de los factores. Definen cuatro tipo de variables: categórica (categorical), ordinal (discretized quantitative), numérica (quantitative) y binaria (binary). Con base a dicha información realiza un Pipeline que incluya al menos las siguientes transformaciones:

a) Imputación a todas las variables de entrada, diferenciando entre el tipo de cada variable (decide y justifica que tipo de imputación realizas en cada caso).

b) Realiza un análisis de las variables numéricas (quantitative) de entrada y aplica una transformación que escale a todas ellas en un rango equiparable

c) Aplica la transformación One-Hot encoding a las variables de entrada de tipo categórico y binaria. En particular, justifica por qué una variable binaria requeriría que se le aplique la transformación one-hot encoding. Por el momento dejar las variables ordinales sin

transformar.

ANALIZANDO LOS DATOS ORIGINALES

Antes de resolver el ejercicio 3 es necesario entender los datos. Por esto, primero vamos a analizar el conjunto de datos que se nos ha proporcionado.

Tomando en cuenta los tipos de datos, valores nulos y su distribución. Verificamos si existen valores nulos.

```
df.isna().any()
```

```
Estado                   False
Duracion                 False
Historial_crediticio     False
Proposito                False
Importe                  False
Ahorros                  False
Duracion_Empleo          False
tasa                     False
Genero                   False
Otros_Acreedores         False
Residencia_Actual        False
Bienes                   False
Edad                     False
Otros_planes_de_credito  False
Tipo_de_vivienda         False
Numero_de_creditos       False
trabajo                  False
Dependientes             False
Telefono                 False
Trabajador_extranjero    False
Riesgo_de_credito        False
dtype: bool
```

Una vez que se conoce que no se tienen valores nulos, procederemos a dividir nuestras variables o características de acuerdo al tipo de dato.

Primero analizamos la variable de salida. Esto para conocer si necesita algún tipo de ajuste.

```
df['Riesgo_de_credito'].describe()
```

```
count    1000.000000
mean        0.700000
std         0.458487
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: Riesgo_de_credito, dtype: float64
```

Cómo podemos ver, el método describre nos dice que tenemos una variable sin datos nulos ya que count logró encontrar 1000 datos de tipo float. Los datos parecen ser solamente 1 y 0. Lo confirmamos de la siguiente manera.

```
df['Riesgo_de_credito'].unique()
```

```
array([1, 0])
```

De esta manera, podemos concluir que la variable "Y" no necesita ni imputaciones para sustituir variable nulos, ni transformaciones, pues los datos ya se encuentran en un rango equiparable a comparación de lo que se nos indican en las instrucciones de este ejercicio.

De acuerdo a nuestra documentación, nuestra entrada o variable "X" se encuentra dividida de la siguiente forma:

- **Variables Categóricas:**

  1. 'status'/'Estado'
  2. 'credit_history'/'Historial_crediticio'
  3. 'purpose'/'Proposito'
  4. 'savings'/'Ahorros'
  5. 'personal_status_sex'/'Genero'
  6. 'other_debtors'/'Otros_acreedores'

7. 'other_installment_plans'/'Otros_planes_de_credito'
8. 'housing'/'Tipo_de_vivienda'

- **Variables Numéricas:**

1. 'duration'/'Duracion'
2. 'amount'/'Cantidad'
3. 'age'/'Edad'

- **Variables binarias:**

1. 'people_liable'/'Personas_dependientes'
2. 'telephone'/'Telefono'
3. 'foreign_worker'/'Trabajador_extranjero'

- **Variables Ordinales:**

1. 'employment_duration'/'Duracion_del_empleo'
2. 'installment_rate'/'Tasa'
3. 'present_residence'/'Residencia_actual_'
4. 'property'/'Bienes'
5. 'number_credits'/'Numero_de_creditos'
6. 'job'/'Trabajo'

```
Categorical_Variables= ['Estado', 'Historial_crediticio', 'Proposito', 'Ahorros', 'Genero','Otros_Acreedores', 'Otros_planes_

Numerical_Variables= ['Duracion', 'Importe', 'Edad']

Binary=['Dependientes', 'Telefono', 'Trabajador_extranjero']

Ordinal= ['Duracion_Empleo','tasa','Residencia_Actual', 'Bienes', 'Numero_de_creditos', 'trabajo']
```

De acuerdo al análisis inicial, no se necesita realizar una imputación, ya que los datos no contienen valores nulos. Sin embargo, en las instrucciones se indica definir las imputaciones que se usarían para cada caso. Esto puede ser de utilidad en caso de que nuestra base de datos se actualice y comiencen a haber casos de datos nulos en nuestras variables.

Primero analizaremos cada tipo de dato para comparar diferencias antes y después de las imputaciones y transformaciones.

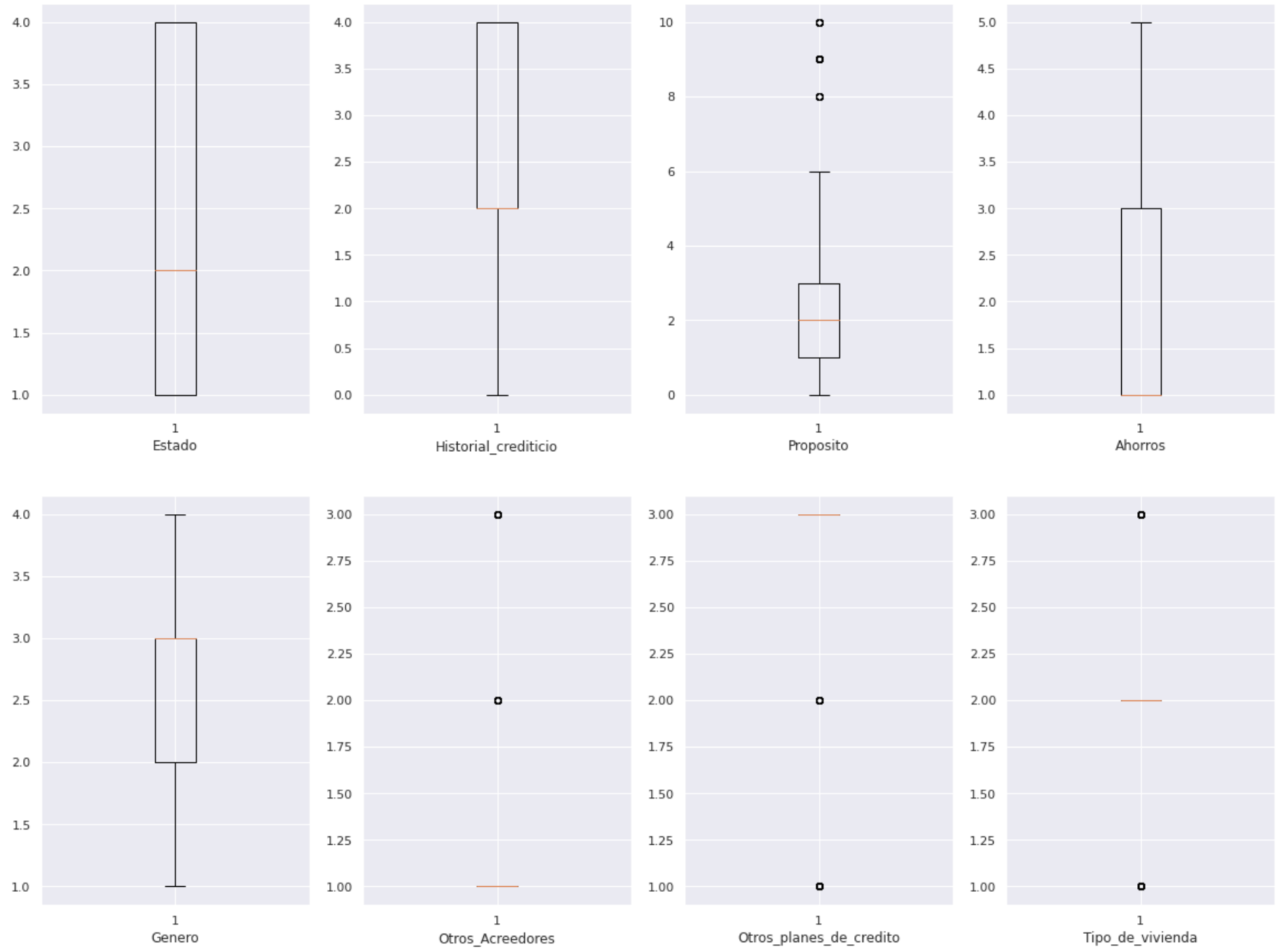
Nota: Se omitirán las variables ordinales conforme las instrucciones.

1. **Variables Categóricas**

```
X_train_categ=X_train[Categorical_Variables].copy()
X_train_categ.describe()
```

	Estado	Historial_crediticio	Proposito	Ahorros	Genero	Otros_Acreedores	Otros_planes_de_credito
count	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000
mean	2.590588	2.532941	2.800000	2.136471	2.677647	1.148235	2.681176
std	1.257030	1.073055	2.704531	1.593630	0.711404	0.484534	0.700495
min	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000
25%	1.000000	2.000000	1.000000	1.000000	2.000000	1.000000	3.000000
50%	2.000000	2.000000	2.000000	1.000000	3.000000	1.000000	3.000000
75%	4.000000	4.000000	3.000000	3.000000	3.000000	1.000000	3.000000
max	4.000000	4.000000	10.000000	5.000000	4.000000	3.000000	3.000000

```
sns.set(rc={'figure.figsize':(20,15)})

figure_1, axes = plt.subplots(2, 4)
for i in range(0,8):
  plt.subplot(2, 4, (i+1))
  plt.boxplot(X_train_categ[X_train_categ.columns[i]])
  plt.xlabel(X_train_categ.columns[i])
plt.show()
```

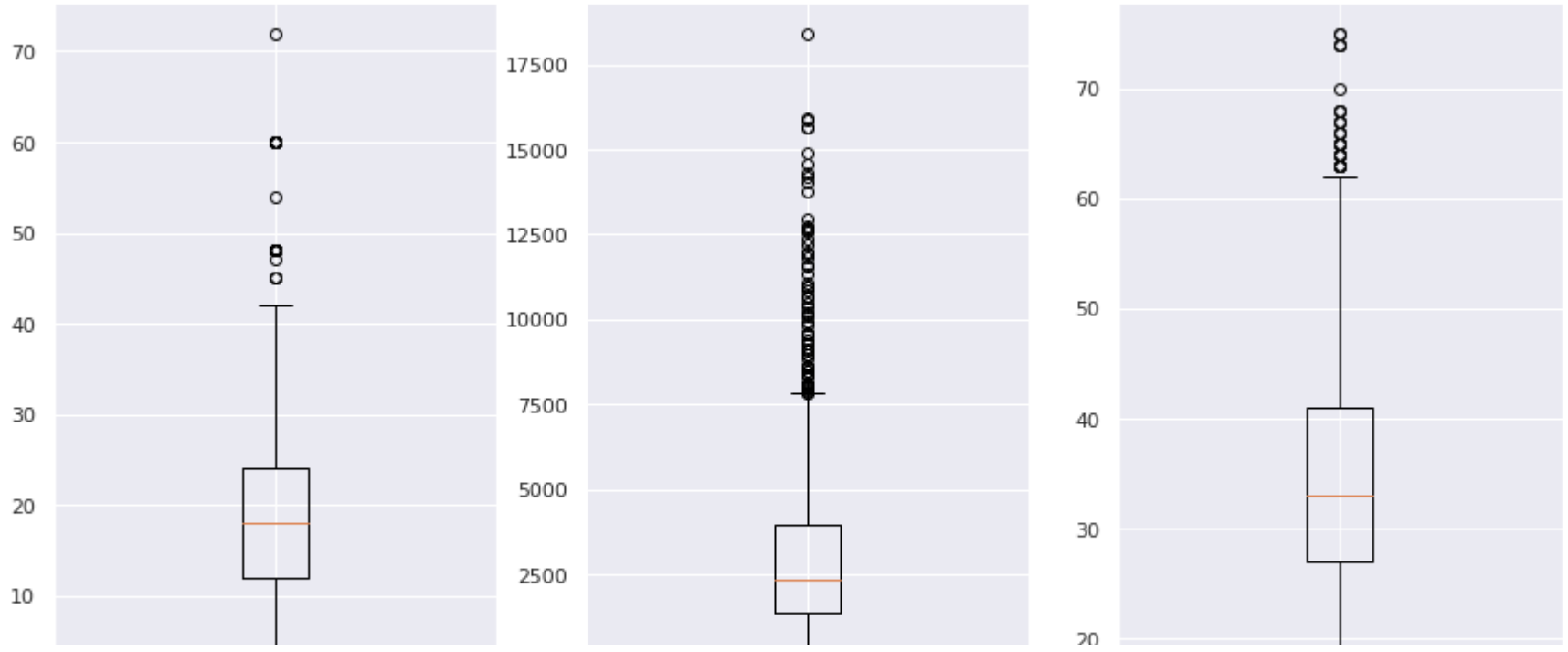


2. **Variables Numéricas**

```
X_train_num=X_train[Numerical_Variables].copy()
X_train_num.describe()
```

	Duracion	Importe	Edad
count	850.000000	850.000000	850.000000
mean	20.738824	3276.667059	35.208235
std	11.955651	2826.845733	11.173972
min	4.000000	276.000000	19.000000
25%	12.000000	1367.250000	27.000000
50%	18.000000	2328.000000	33.000000
75%	24.000000	3959.000000	41.000000
max	72.000000	18424.000000	75.000000

```
sns.set(rc={'figure.figsize':(20,15)})

figure_2, axes = plt.subplots(1, 3)
for i in range(0,3):
  plt.subplot(2, 4, (i+1))
  plt.boxplot(X_train_num[X_train_num.columns[i]])
  plt.xlabel(X_train_num.columns[i])
plt.show()
```

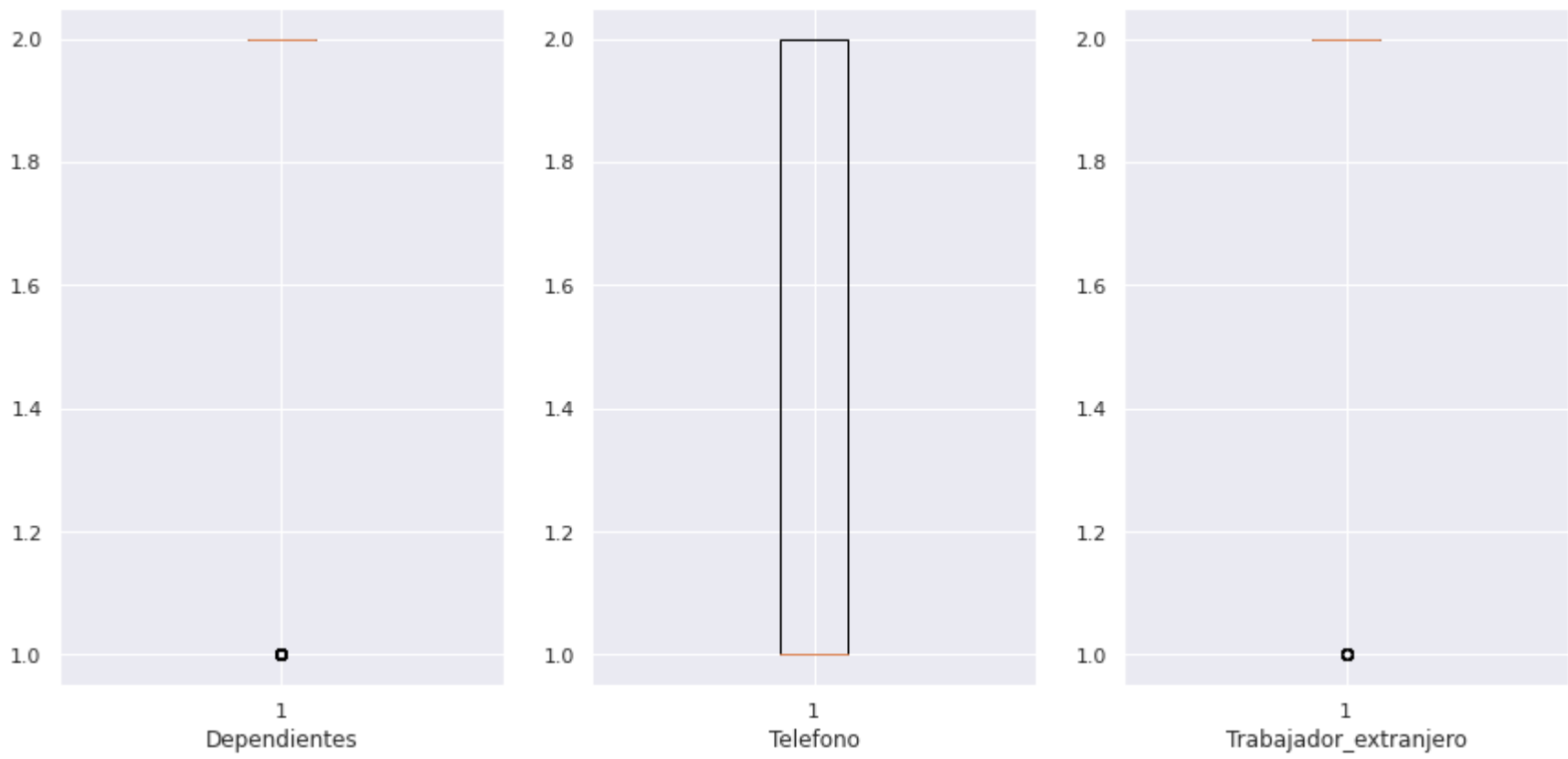


3. **Variables Binarias**

```
X_train_bin=X_train[Binary].copy()
X_train_bin.describe()
```

	Dependientes	Telefono	Trabajador_extranjero
count	850.000000	850.000000	850.000000
mean	1.854118	1.408235	1.960000
std	0.353196	0.491796	0.196075
min	1.000000	1.000000	1.000000
25%	2.000000	1.000000	2.000000
50%	2.000000	1.000000	2.000000
75%	2.000000	2.000000	2.000000
max	2.000000	2.000000	2.000000

```
sns.set(rc={'figure.figsize':(20,15)})

figure_3, axes = plt.subplots(1, 3)
for i in range(0,3):
  plt.subplot(2, 4, (i+1))
  plt.boxplot(X_train_bin[X_train_bin.columns[i]])
  plt.xlabel(X_train_bin.columns[i])
plt.show()
```

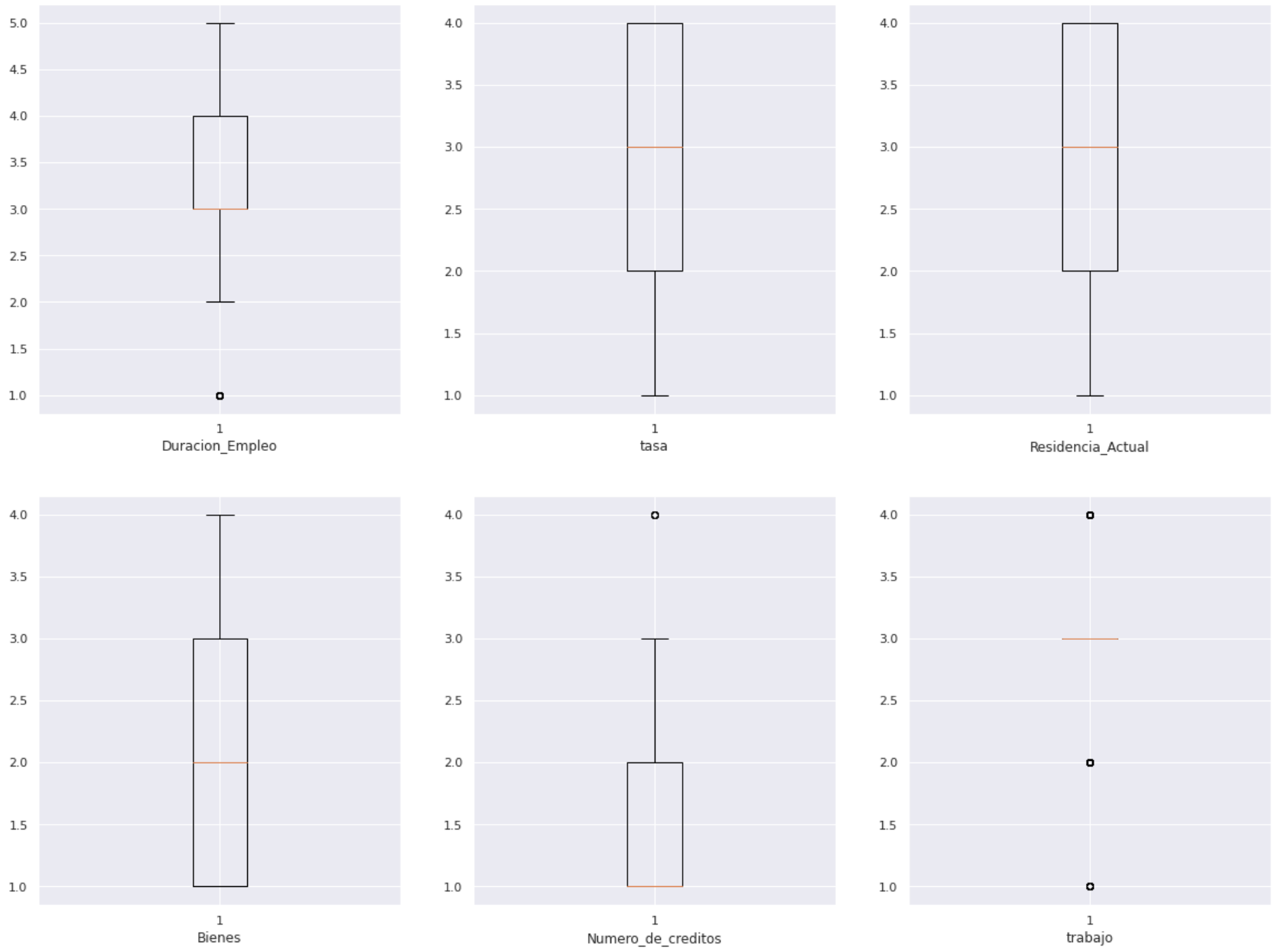


4. **Variables Ordinales**

```
X_train_ord=X_train[Ordinal].copy()
X_train_ord.describe()
```

	Duracion_Empleo	tasa	Residencia_Actual	Bienes	Numero_de_creditos	trabajo
count	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000
mean	3.363529	2.943529	2.844706	2.337647	1.388235	2.900000
std	1.209829	1.133747	1.097970	1.036493	0.567974	0.658817
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	3.000000	2.000000	2.000000	1.000000	1.000000	3.000000
50%	3.000000	3.000000	3.000000	2.000000	1.000000	3.000000
75%	4.000000	4.000000	4.000000	3.000000	2.000000	3.000000
max	5.000000	4.000000	4.000000	4.000000	4.000000	4.000000

```
sns.set(rc={'figure.figsize':(20,15)})

figure_3, axes = plt.subplots(2, 3)
for i in range(0,6):
  plt.subplot(2, 3, (i+1))
  plt.boxplot(X_train_ord[X_train_ord.columns[i]])
  plt.xlabel(X_train_ord.columns[i])
plt.show()
```



IMPUTACION Y TRANSFORMACIÓN

Después de analizar nuestros datos y su distribución, empezaremos con el proceso de imputación y transformación.

JUSTIFICACIÓN PARA LA IMPUTACIÓN Y TRANSFORMACIONES

A) Imputación a todas las variables de entrada, diferenciando entre el tipo de cada variable (decide y justifica que tipo de imputación realizas en cada caso).

> **Valores numéricos:** Para el caso de valores ausentes decidimos utilizar la mediana como el valor imputado, dado que estos valores pueden ser influenciados si se usa la media para la imputación, en cuyo caso puede afectar el modelo y hacer inferencias incorrectas con relación a la población de la que se está tomando el dato. Además, de acuerdo con otros artículos revisados, la imputación a la media afecta a la desviación estándar, por lo tanto a la distribución de los datos, lo cual termina reflejando algo distinto a la realidad que se enfrenta el modelo.

> En la gráfica de cajas (bloxplot) se evidencia que hay muchos outliers para los tres valores numéricos, no obstante se utilizará la transformación MinMax pues en las instrucciones se nos indica que todas las variables númericas deben de quedar transformadas en un rango equiparable. En este sendi, MinMax se caracteriza porque nos permite dejar los valores dentro de un rango específico.

> Referencia:

> Jadhav, A., Pramod, D., & Ramanathan, K. (2019). Comparison of performance of data imputation methods for numeric dataset. Applied Artificial Intelligence, 33(10), 913-933. Recuperado el 23 de octubre de 2022 en: https://www.tandfonline.com/doi/full/10.1080/08839514.2019.1637138

> Géron, Aurélien (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition. O'Reilly. Chapter 2. End-to-End Machine Learning Project.


```
#Transformación e imputación para valores Numéricos.
#Decidimos utilizar la imputación mediana y transformación MinMax.
num_pipeline = Pipeline(steps = [('impMediana', SimpleImputer(strategy='median')),
                                 ('escalaNum', MinMaxScaler(feature_range=(1,2)))])
num_pipeline_nombres = Numerical_Variables
```

> **Valores categóricos:** Decidimos utilizar la moda para la imputación de los datos ausentes, así como la transformación de One Hot Encoding, la cual tiene la particularidad de que cuanto mayores sean las clasificaciones de los datos categóricos, se generan muchas más columnas en la transformación. Lo que provocaría un mayor costo computacional; sin embargo, revisando referencias de uso, se ha demostrado que es más efectivo hacer esta transformación para datos categóricos, porque permite tener una mejor trazabilidad de las variables y determinar cómo afecta al modelo. No obstante, se espera que esta aplicación, sea posterior a una limpieza considerando evitar la redundancia de datos, así como quitar columnas cuyos pesos no afectan el resultado de un modelo. También se encontró que de acuerdo a otros investigadores, la utilización de One Hot Encoding genera buen desempeño en los problemas de clasificación de crédito cuando existe ausencia de datos.

> Finalmente, la imputación de moda es la más utilizada para variables categóricas debido a las características de las mismas. Por ejemplo, en nuestro caso, las variables categóricas tienen un rango de valores enteros definido, por ende, usar imputaciones que puedan arrojar un número con decimales cómo la media o mediana, afectaría la integridad de nuestros datos.

> Referencias:

> Caro Puerta, L. C., & Rodas Zuluaga, L. J. (2022). Modelos de aprendizaje supervisado para la clasificación de riesgo crediticio en la entidad financiera Home Credit. Recuperado el 23 de octubre de 2022 en: https://bibliotecadigital.udea.edu.co/handle/10495/29124

> Seger, C. (2018). An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing. Recuperado el 23 de octubre de 2022 en https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1259073&dswid=-1263

> Yu, L., Zhou, R., Chen, R., & Lai, K. K. (2022). Missing data preprocessing in credit classification: One-hot encoding or imputation?. Emerging Markets Finance and Trade, 58(2), 472-482. Recuperado el 23 de octubre de 2022 en: https://www.tandfonline.com/doi/abs/10.1080/1540496X.2020.1825935


```
#Transformación e imputación para valores Categóricos.
#Decidimos utilizar el modo de imputación y Transformación de One Hot Encoding.
cat_pipeline = Pipeline(steps = [('impModa', SimpleImputer(strategy='most_frequent')), ('OneHotE', OneHotEncoder(drop='first
cat_pipeline_nombres = Categorical_Variables
```

> **Valores binarios:** Decidimos hacer la imputación usando la moda basándonos en que las variables binarias tienen tambien un rango especifico de valores enteros (solo 0 y 1), adicionalmente, para datos que hablan acerca del

> comportamiento de una población, es de las imputaciones más sencillas. También se decidio por ella para no afectar en demasiado la desviación estándard y la media con las imputaciones de mediana y media. Como se menciono anteriormente, es altamente recomendable usar one hot encoder por la practicidad que genera tener las clasificaciones de un elemento como vector independiente.

> Referencias:

> Jadhav, A., Pramod, D., & Ramanathan, K. (2019). Comparison of performance of data imputation methods for numeric dataset. Applied Artificial Intelligence, 33(10), 913-933. Recuperado el 23 de octubre de 2022 en: https://www.tandfonline.com/doi/full/10.1080/08839514.2019.1637138

> Xu, X., Xia, L., Zhang, Q., Wu, S., Wu, M., & Liu, H. (2020). The ability of different imputation methods for missing values in mental measurement questionnaires. BMC Medical Research Methodology, 20(1), 1-9. Recuperado el 24 de octubre de 2022 en: https://link.springer.com/article/10.1186/s12874-020-00932-0


```
#Transformación e imputación para valores Binarios.
#Decidimos utilizar el modo de imputación y Transformación de One Hot Encoding.
bin_pipeline = Pipeline(steps = [('impModa', SimpleImputer(strategy='most_frequent')), ('OneHotE', OneHotEncoder(drop='first
bin_pipeline_nombres = Binary
```

> **Valores ordinales:** No habrá transformación para este tipo de variables. Decidimos utilizar la moda como imputación basados en que para datos que hablan acerca del comportamiento de una población, es de las imputaciones más sencillas, también se decide para no afectar en demasía la desviación estandar y la media con las imputaciones de mediana y media.

> Referencias:

> Jadhav, A., Pramod, D., & Ramanathan, K. (2019). Comparison of performance of data imputation methods for numeric dataset. Applied Artificial Intelligence, 33(10), 913-933. Recuperado el 23 de octubre de 2022 en: https://www.tandfonline.com/doi/full/10.1080/08839514.2019.1637138

> Xu, X., Xia, L., Zhang, Q., Wu, S., Wu, M., & Liu, H. (2020). The ability of different imputation methods for missing values in mental measurement questionnaires. BMC Medical Research Methodology, 20(1), 1-9. Recuperado el 24 de octubre de 2022 en: https://link.springer.com/article/10.1186/s12874-020-00932-0


```
# Imputación para valores ordinales. No habrá transformación para este tipo de variables.
# Decidimos utilizar el modo de imputación
ord_pipeline = Pipeline(steps = [('impModa', SimpleImputer(strategy='most_frequent'))])
ord_pipeline_nombres = Ordinal
# Conjuntamos las transformaciones numéricas y categóricas que se estarán aplicando a los datos de entrada:
columnasTransformer = ColumnTransformer(transformers = [('numpipe', num_pipeline, num_pipeline_nombres),
                                                         ('catpipe', cat_pipeline, cat_pipeline_nombres),
                                                         ('binpipe', bin_pipeline, bin_pipeline_nombres),
                                                         ('ordpipe', ord_pipeline, ord_pipeline_nombres)],
                                                remainder='passthrough')
```

Ejercicio-4.

Llevarás un entrenamiento usando validación cruzada entre los siguientes tres modelos de aprendizaje automático: Regresión Logística, Árbol de Decisión y Bosque Aleatorio. Deberás llevar a cabo el entrenamiento de los tres de manera conjunta usando un ciclo FOR. Recuerda aplicar las transformaciones que definiste en tu Pipeline. El entrenamiento debe ser con las siguientes características:

1. Usa los parámetros predeterminados de cada modelo.
2. En cada iteración deben calcularse todas las siguientes métricas: accuracy, precision,recall, f1-score y Gmean. Todas estas métricas deben ser funciones que tú mismo debes definir (Es decir, no usar las funciones de dichas métricas que te proporciona scikit-learn. Sin embargo, sí puedes usar la información regresada por el método confusion_matrix() de scikit-learn para definir las métricas).
3. Usar validación cruzada estratificada con 5 particiones y con 3 repeticiones.
4. Imprimir el valor de todas estas métricas, tanto para los datos de entrenamiento, como para los de validación. Así como los diagramas de caja y bigotes de los tres modelos con la métrica “recall”. ¿Alguno de los modelos está subentrenado o sobreentrenado? Justifica tu respuesta.

5. En particular obtengamos algunas de las llamadas curvas de aprendizaje para algunos de estos casos. En dada gráfico debes incluir tus comentarios sobre el modelo generado:

> 1. Obtener las curvas de aprendizaje (learning_curve) en la cual se va incrementando el tamaño de la muestra para el modelo de regresión Logística con su hiperparámetros predeterminados. Utilizar al menos 20 puntos en la partición de los conjuntos de entrenamiento y la métrica “f1-score”, como evaluación del desempeño de dicha función “learning_curve()”.
> 2. Obtener las curvas de validación (validation_curve) en la cual se va incrementando la complejidad del hiperparámetro “max_depth” para el modelo de árbol de decisión con sus hiperparámetros predeterminados. Utilizar valores de máxima profundidad desde 1 hasta 20 y con la métrica “f1-score” para la evaluación del desempeño del modelo.
> 3. Obtener las curvas de aprendizaje (learning_curve) en la cual se va incrementando el tamaño de la muestra para el modelo de regresión bosque aleatorio (random forest) con su hiperparámetros predeterminados. Utilizar al menos 20 puntos en la partición de los conjuntos de entrenamiento y la

En el siguiente bloque definiremos las funciones para el cálculo de las métricas: Accuracy, Precision, Recall, Gmean

```
def calculate_accuracy(yreal, ypred):

    results = confusion_matrix(yreal, ypred)
    vn, fp, fn, vp = results.ravel()

    acc = (vp + vn)/ (vp+vn+fp+fn)

    return acc

def calculate_precision(yreal, ypred):

    results = confusion_matrix(yreal, ypred)
    vn, fp, fn, vp = results.ravel()

    prec = (vp)/ (vp+fp)

    return prec

def calculate_recall(yreal, ypred):

    results = confusion_matrix(yreal, ypred)
    vn, fp, fn, vp = results.ravel()

    recall = vp/(vp + fn)

    return recall

def calculate_f1score(yreal, ypred):

    results = confusion_matrix(yreal, ypred)
    vn, fp, fn, vp = results.ravel()

    f1score = (2*vp)/(2*vp+ fp + fn)

    return f1score

def calculate_gmean(yreal, ypred):

    results = confusion_matrix(yreal, ypred)
    vn, fp, fn, vp = results.ravel()

    specificity = vn/(vn + fp)
    recall = vp/(vp + fn)

    gmean = np.sqrt(recall * specificity)

    return gmean
```

A continuación definiremos nuestro ciclo FOR en donde se realizará el entrenamiento de los tres modelos. Cómo se especifica en las instrucciones se utlizan las funciones de las métricas definidas por nosotros, se define el Kfold y se imprimen los resultados tanto de nuestro modelo con los datos de entrenamiento y validación.

```
models = [LogisticRegression(max_iter=3000), DecisionTreeClassifier(), RandomForestClassifier()]
model_names = ["Logistic_Regression", "Decision_Tree", "Random_Forest"]
results= list()

for i in range(len(models)):
```

```
#
pipeline = Pipeline(steps = [('ct', columnasTransformer), (model_names[i], models[i])])

#
metrics= {'accuracy':make_scorer(calculate_accuracy), 'precision':make_scorer(calculate_precision), 'recall':make_sc
kfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=3)
scores = cross_validate( estimator=pipeline, X=X_train, y=np.ravel(y_train),
    scoring=metrics, cv=kfold, return_train_score=True)

#El score de cada iteración es almacenado en la lista "scores"
results.append(scores)


#Imprimos el resultado para cada modelo
print('TRAIN METRIC SCORES\n'+'%s:\nmean Accuracy: %.3f (%.4f)\nmean Precision: %.3f (%.4f)\nmean Recall: %.3f (%.4f
                                                                np.mean(scores['train_accuracy']),
                                                                np.std(scores['train_accuracy']),
                                                                np.mean(scores['train_precision']),
                                                                np.std(scores['train_precision']),
                                                                np.mean(scores['train_recall']),
                                                                np.std(scores['train_recall']),
                                                                np.mean(scores['train_f1score']),
                                                                np.std(scores['train_f1score']),
                                                                np.mean(scores['train_gmean']),
                                                                np.std(scores['train_gmean']),
                                                                ))
```

```
print('VALIDATION METRIC SCORES\n'+'%s:\nmean Accuracy: %.3f (%.4f)\nmean Precision: %.3f (%.4f)\nmean Recall: %.3f
                                                                np.mean(scores['test_accuracy']),
                                                                np.std(scores['test_accuracy']),
                                                                np.mean(scores['test_precision']),
                                                                np.std(scores['test_precision']),
                                                                np.mean(scores['test_recall']),
                                                                np.std(scores['test_recall']),
                                                                np.mean(scores['test_f1score']),
                                                                np.std(scores['test_f1score']),
                                                                np.mean(scores['test_gmean']),
                                                                np.std(scores['test_gmean']),
                                                                ))
```

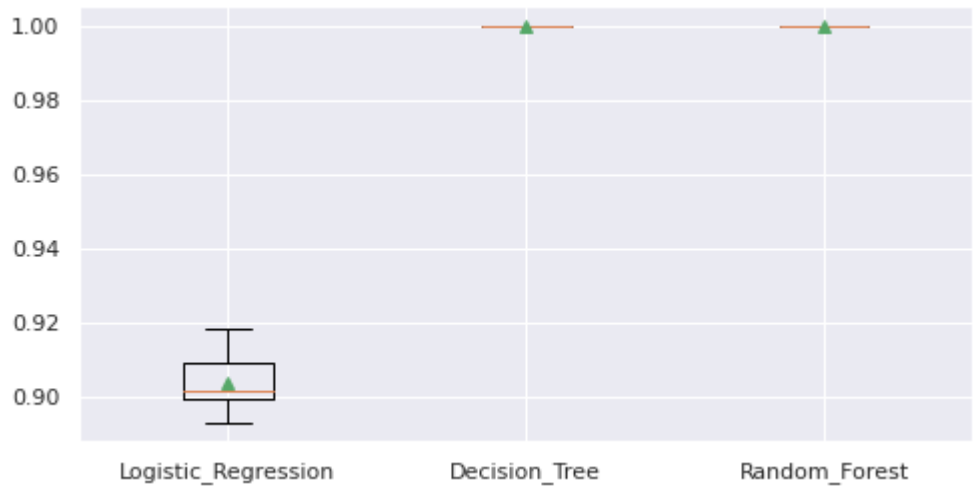
```
TRAIN METRIC SCORES
Logistic_Regression:
mean Accuracy: 0.791 (0.0075)
mean Precision: 0.819 (0.0067)
mean Recall: 0.904 (0.0068)
mean f1_Score: 0.859 (0.0049)
Gmean: 0.688 (0.0145)

VALIDATION METRIC SCORES
Logistic_Regression:
mean Accuracy: 0.752 (0.0255)
mean Precision: 0.793 (0.0181)
mean Recall: 0.877 (0.0244)
mean f1_Score: 0.832 (0.0175)
Gmean: 0.631 (0.0406)

TRAIN METRIC SCORES
Decision_Tree:
mean Accuracy: 1.000 (0.0000)
mean Precision: 1.000 (0.0000)
mean Recall: 1.000 (0.0000)
mean f1_Score: 1.000 (0.0000)
Gmean: 1.000 (0.0000)

VALIDATION METRIC SCORES
Decision_Tree:
mean Accuracy: 0.682 (0.0361)
mean Precision: 0.775 (0.0237)
mean Recall: 0.773 (0.0449)
mean f1_Score: 0.773 (0.0292)
Gmean: 0.598 (0.0470)

TRAIN METRIC SCORES
Random_Forest:
mean Accuracy: 1.000 (0.0000)
mean Precision: 1.000 (0.0000)
mean Recall: 1.000 (0.0000)
mean f1_Score: 1.000 (0.0000)
Gmean: 1.000 (0.0000)

VALIDATION METRIC SCORES
```

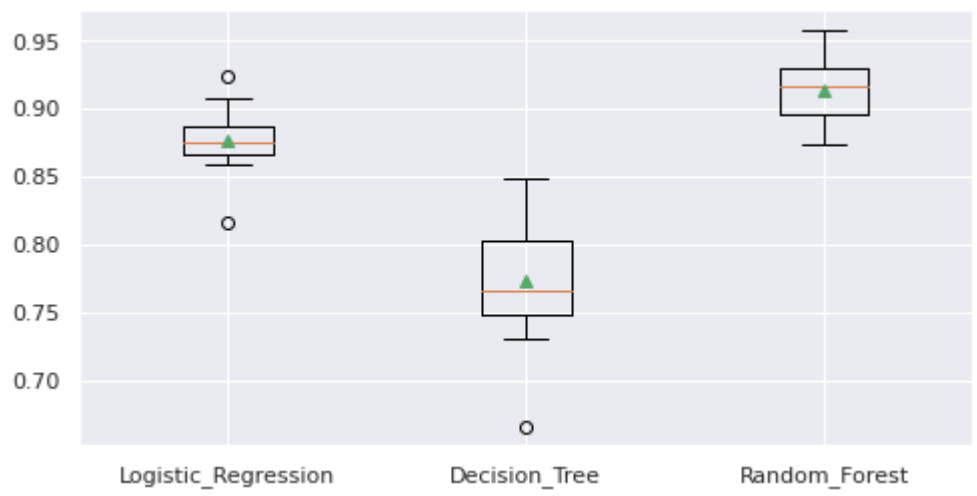
```
Random_Forest:
mean Accuracy: 0.756 (0.0206)
mean Precision: 0.778 (0.0159)
mean Recall: 0.914 (0.0251)
mean f1_Score: 0.841 (0.0139)
Gmean: 0.588 (0.0427)
```

Ahora se definiran las gráficas de caja y bigotes para cada uno de nuestros modelos utilizando la métrica recall.

```
sns.set(rc={'figure.figsize':(8,4)})
bprecall_train = list()
for i in range(len(results)):
  rr = results[i]['train_recall']
  bprecall_train.append(rr)

plt.boxplot(bprecall_train, labels=model_names, showmeans=True)
plt.show()
```

```
bprecall_test = list()
for i in range(len(results)):
  rr = results[i]['test_recall']
  bprecall_test.append(rr)

plt.boxplot(bprecall_test, labels=model_names, showmeans=True)
plt.show()
```





PREGUNTA

1. ¿Son los modelos sobre entrenados o subentrenados? Justifica

> En el procesamiento de los datos de entrenamiento, para la técnica de árbol de decisión, así también como en el caso del bosque aleatorio, son tan efectivos los dos modelos que nos hacen pensar que ambos modelos están queriendo solamente memorizar los datos de prueba, en vez de hacer el proceso de aprender. Lo cual nos hace pensar que hay un sobreentrenamiento involucrado, pues obtenemos un mucho menor rendimiento de las métricas cuando el modelo trabaja con los datos de validación. La alta efectividad la podemos ver reflejada tanto en los boxplots como en las métricas de de los datos de entrenamiento donde se afirma unos desempeños del 100%. Mientras que , cuando el modelo trabaja con los datos de validación, podemos observar como los gráficos y los porcentajes disminuyen y se dispersan.

> En el caso de los datos de prueba, existe una dispersión homogénea y sin presentar tantos outliers, sin embargo tienen una alta desviación entre los resultados de cada modelo.

> Con lo que respecta a la regresión logística, los resultados del modelo utilizando los datos de entrenamiento nos hacen inferir está sub-entrenada. Su desempeño en las métricas no es tan bueno (no llega al 90% en ninguna de las métricas). Adicionalmente, debido a que hay una gran similitud en los resultados de las métricas utlizando los datos de validacion, podemos también hipotetizar que ha alcansado su máxima eficiencia.

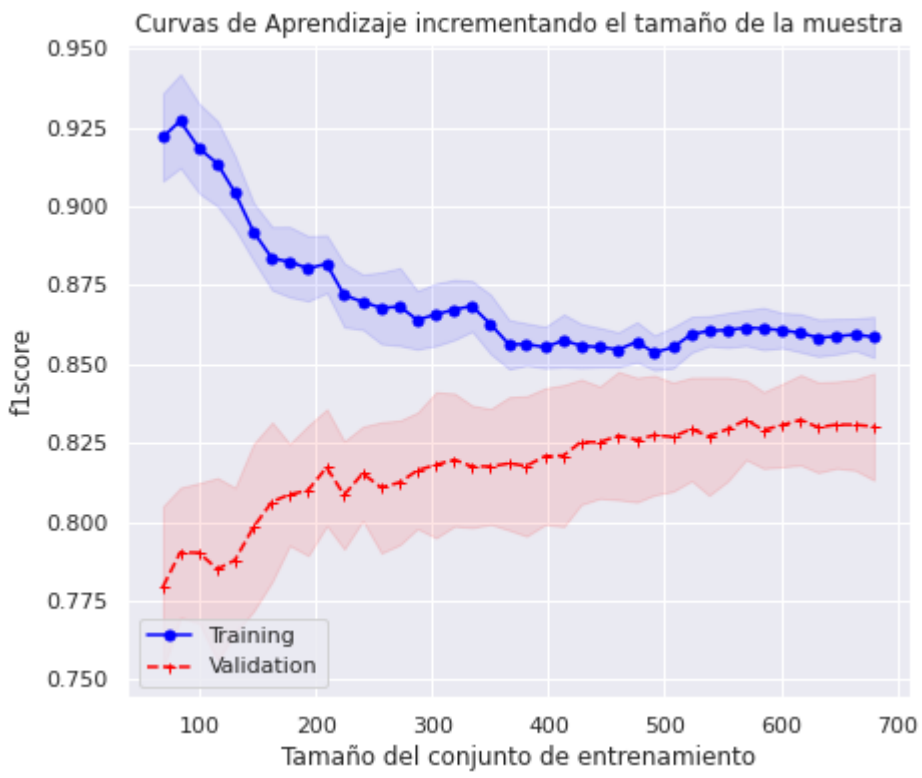
> A continuación, nos basaremos en las curvas de aprendizaje para hacer una mejor conclusión con cada uno de los modelos:

```
def my_LearningCurvePlot(train_sizes, train_scores, val_scores, y_axis_metric):

        train_mean = np.mean(train_scores, axis=1)
        train_std = np.std(train_scores, axis=1)
        val_mean = np.mean(val_scores, axis=1)
        val_std = np.std(val_scores, axis=1)

        # Graficamos las curvas de aprendizaje incluyendo una región indicando la desviación estándar.
        plt.figure(figsize=(7,6))
        plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training')
        plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, alpha=0.1, color='blue')

        plt.plot(train_sizes, val_mean, color='red', marker='+', markersize=5, linestyle='--', label='Validation')
        plt.fill_between(train_sizes, val_mean + val_std, val_mean - val_std, alpha=0.1, color='red')

        plt.title('Curvas de Aprendizaje incrementando el tamaño de la muestra')
        plt.xlabel('Tamaño del conjunto de entrenamiento')
        plt.ylabel(y_axis_metric)
        plt.grid(b=True)
        plt.legend(loc='lower left')
        plt.show()
```

```
def my_ModelnLearningCurve(Xin, models, model_names, y_axis_metric):
  Xc_train=columnasTransformer.fit(Xin)
  Xx_train=Xc_train.transform(Xin)

  kfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=3)
  delta_train_sz = np.linspace(0.1, 1, num=40)

  tr_sizes, tr_scores, val_scores = learning_curve(estimator = models,
                                                   X = Xx_train,
                                                   y = y_train.values.ravel(),
                                                   cv = kfold,
                                                   train_sizes =delta_train_sz,
                                                   random_state=11, scoring=metrics[y_axis_metric])
  print('Curvas de aprendizaje del ' + model_names)
  my_LearningCurvePlot(tr_sizes, tr_scores, val_scores, y_axis_metric)
```

```
my_ModelnLearningCurve(X_train, models[0], model_names[0], y_axis_metric='f1score')
```

```
Curvas de aprendizaje del Logistic_Regression
```



Observaciones

En la gráfica de Logistic Regression, podemos ver que tanto nuestros datos de entrenamiento como los de prueba estan lejos de un desempeño óptimo; ni si quiera alcanzamos el 90% de rendimiento en esta métrica. Por ende podríamos conclui que nuestro modelo esta subentrenado. De igual manera, también hay una distancia considerable entre los datos de entrenamiento y de validación (4%), dando pie a que haya lígera varianza y, por ende, a un lígero sobre-entrenamiento.

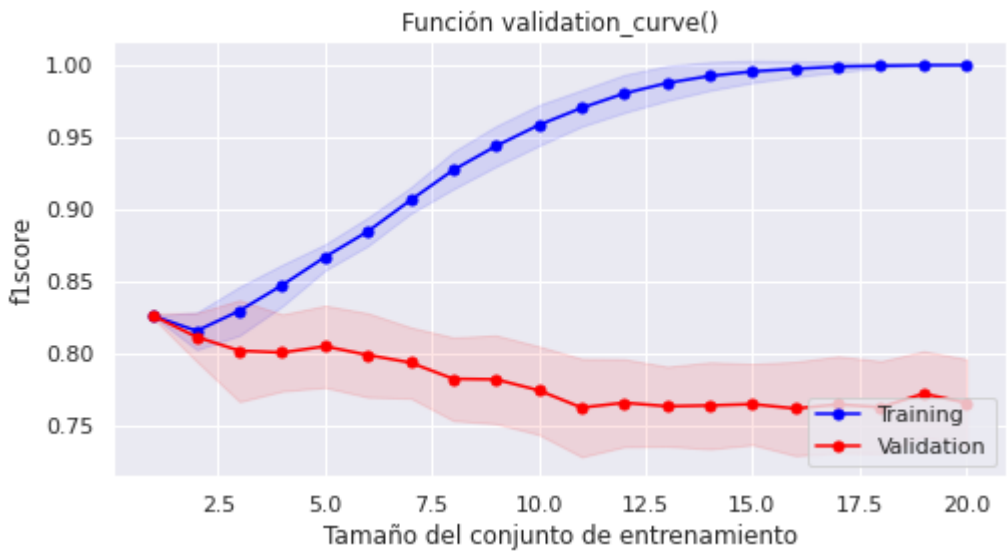
Referencias:

Géron, Aurélien (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition. O'Reilly. Chapter 4 Training

```
def my_ModelnValidationCurve(Xin, models, model_names, y_axis_metric):
  Xc_train=columnasTransformer.fit(Xin)
  Xx_train=Xc_train.transform(Xin)

  kfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=3)
  delta_train_sz = np.linspace(1, 20, num=20, dtype='int')



  train_scores, valid_scores = validation_curve(models,
                                                Xx_train,
                                                np.ravel(y_train),
                                                param_name="max_depth",
                                                param_range=delta_train_sz,
                                                cv=kfold,
                                                scoring=metrics[y_axis_metric])

  train_mean = np.mean(train_scores, axis=1)
  train_std = np.std(train_scores, axis=1)
  valid_mean = np.mean(valid_scores, axis=1)
  valid_std = np.std(valid_scores, axis=1)

  print('Curvas de validación del ' + model_names)
  # Curva de entrenamiento con la métrica de exactitud (accuracy):
  plt.plot(np.linspace(1, 20, num=20, dtype='int'), train_mean, color='blue', marker='o', markersize=5, label='Training')
  plt.fill_between(np.linspace(1, 20, num=20, dtype='int'), train_mean + train_std, train_mean - train_std, alpha=0.1, color=



  # Curva de validación:
  plt.plot(np.linspace(1, 20, num=20, dtype='int'), valid_mean, color='red', marker='o', markersize=5, label='Validation')
  plt.fill_between(np.linspace(1, 20, num=20, dtype='int'), valid_mean + valid_std, valid_mean - valid_std, alpha=0.1, color=

  plt.title('Función validation_curve()')
  plt.xlabel('Tamaño del conjunto de entrenamiento')
  plt.ylabel(y_axis_metric)
  plt.grid(b=True)
  plt.legend(loc='lower right')
  plt.show()
```

```
my_ModelnValidationCurve(X_train, models[1], model_names[1], y_axis_metric='f1score')
```

```
Curvas de validación del Decision_Tree
```



Observaciones

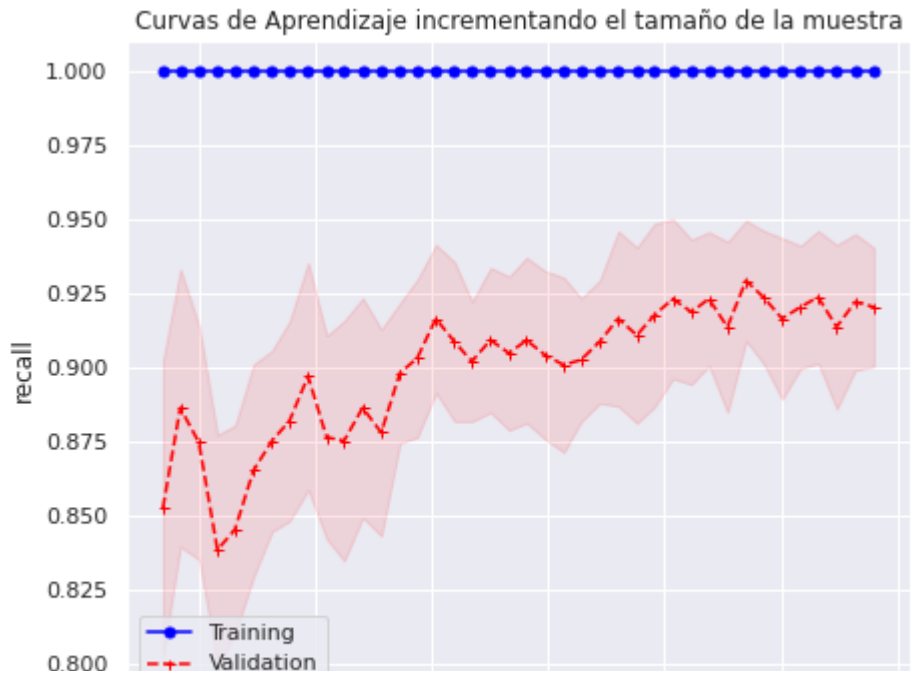
Podemos ver que los datos de entrenamiento se comportan de manera casi perfecta, sin embargo, al llegar a la validación podemos notar una caída en la efectividad del modelo. Una diferencia de más de 20% entre los rendimientos de los modelos sugiere un serio sobre-entrenamiento de nuestros datos. Posiblemente esto sucede porque en nuestro modelo no hemos especificado el número de splits y hojas que deseamos. Por ejemplo, scikitlearn nos dice que si no especificamos ciertos parametros el arbol se seguira extendiendo hasta conseguir hojas "puras". Esto pudo haber dado pie al sobre-entrenamiento tan marcado.

Referencias:

Géron, Aurélien (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition. O'Reilly. Chapter 4 Training Models.

```
my_ModelnLearningCurve(X_train, models[2], model_names[2], y_axis_metric='recall')
```

```
Curvas de aprendizaje del Random_Forest
```



Observaciones

Nuevamente esta gráfica complementa nuestra hipótesis obtenida a partir de los resultados obtenidos en los gráficos de caja y bigotes (boxplots) y anteriores incisos; tenemos un sobre entrenamiento. Debido a que nuestro modelo muestra una línea recta en el desempeño máximo (100%), podemos inferir que nuestro modelo se aprendio por completo los datos y, por ende, cuando recibe el conjunto de validación, nuestro modelo es incapaz de hacer predicciones tan acertadas. Ahora, un 90% de desempeño en recall para la validación no es malo, no obstante si es una variación significativa (~10%) si lo comparamos con el conjunto de entrenamiento.

Referencias:

Géron, Aurélien (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition. O'Reilly. Chapter 4 Training Models.

Ejercicio-5.

Finalmente, veamos la manera de mejorar los valores de los hiperparámetros de cada modelo, así como el problema del sobreentrenamiento de algunos de ellos. Para ello deberás usar el método GridSearchCV() de scikit-learn. Recuerda que este método hace una búsqueda de los mejores hiperparámetros de un modelo mediante el llamada formato de malla y aplicando validación cruzada. En cada caso puedes incrementar el máximo de iteraciones,"max_iter" para que tengas la convergencia adecuada para todas las combinaciones en cada modelo. Recuerda también aplicar las transformaciones que definiste en tu Pipeline. Para fines de este ejercicio se ha seleccionado para cada modelo una métrica diferente, que permita irte familiarizando con ellas.

1. Para el modelo de regresión logística realizar el entrenamiento buscando sus mejores hiperparámetros con GridSearchCV(). Los hiperparámetros que debes incluir en su búsqueda deben ser al menos los siguientes: C, solver, class_weight y penalty. En este caso deberás usar la métrica (scoring) "f1-score". Imprime la mejor combinación de parámetros obtenidos, así como el valor del mejor desempeño (score) obtenido con la métrica f1. ¿Cuál es la utilidad de la métrica "f1-score"? Incluye tus conclusiones.

   > **NOTA:** Toma en cuenta que no todas las combinaciones de "solver" y "penalty" son posibles, para que lo tomes en cuenta al momento de realizar la búsqueda. Revisa la documentación.

2. Con los mejores valores de los hiperparámetros encontrados con la métrica "f1-score" para el modelo de regresión logística, obtener las curvas de aprendizaje (learning curve), incrementando el tamaño del conjunto de entrenamiento al menos 20 veces. Si lo crees adecuado, puedes hacer los ajustes que consideres adecuados para mejorar el resultado y evitar el sobreentrenamiento o el subentrenamiento.
3. Para el modelo de árbol de decisión (decision tree) realizar el entrenamiento buscando sus mejores hiperparámetros con GridSearchCV(). Los hiperparámetros que debes incluir en su búsqueda deben ser al menos los siguientes: ccp_alpha, criterion, max_depth, min_samples_split y class_weight. En este caso deberás usar la métrica (scoring) "precision". Imprime la mejor combinación de parámetros obtenidos, así como el valor del mejor desempeño (score) obtenido con la métrica "precision". ¿Cuál es la utilidad de la métrica "precision"? Incluye tus conclusiones.
4. Con los mejores valores de los hiperparámetros encontrados con la métrica "precision" para el modelo de árbol de decisión, obtener las curvas de aprendizaje (learning curve), incrementando el tamaño del conjunto de entrenamiento al menos 20 veces. Si lo crees adecuado, puedes hacer los ajustes que consideres adecuados para mejorar el resultado y evitar el sobreentrenamiento o el subentrenamiento.
5. Para el modelo de bosque aleatorio (random forest) realizar el entrenamiento buscando sus mejores hiperparámetros con GridSearchCV(). Los hiperparámetros que debes incluir en su búsqueda deben ser al menos los siguientes: ccp_alpha, criterion, max_depth, min_samples_split y class_weight. En este caso deberás usar la métrica (scoring) "recall". Imprime la mejor

combinación de parámetros obtenidos, así como el valor del mejor desempeño (score) obtenido con la métrica “recall”. ¿Cuál es la utilidad de la métrica “recall”? Incluye tus conclusiones.

> **NOTA:** Toma en cuenta que el método de random forest pude tardar varios minutos en llevar a cabo.

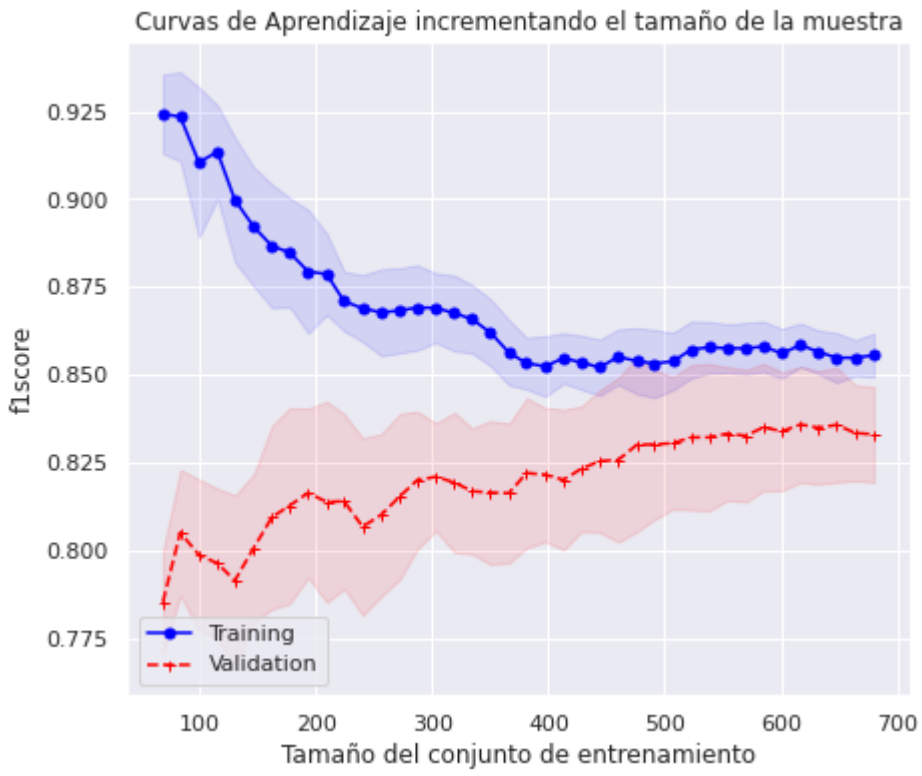
6. Con los mejores valores de los hiperparámetros encontrados con la métrica “recall” para el modelo de bosque aleatorio, obtener las curvas de validación (validation curve), incrementando la complejidad del modelo a través del hiperparámetro “max_depth” con al menos 10 valores. Si lo crees adecuado, puedes hacer los ajustes que consideres adecuados para mejorar el resultado y

```
evaluating_model = LogisticRegression(max_iter=5000)

dicc_grid = {'C':[0.0001,0.01,0.1,1.0,10.,100.], 'class_weight':['balanced', None],
             'solver':['liblinear','saga'],
             'penalty': ['l2','l1']}  #liblinear was removed because it doesn't work with the same penalty as the others

cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3)

grid_LR = GridSearchCV(estimator=evaluating_model,
                    param_grid=dicc_grid,
                    cv=cv,
                    scoring=metrics['f1score'],
                    n_jobs = -1)


# Transformamos los datos de entrada:
Xc_train=columnasTransformer.fit(X_train)
Xx_train=Xc_train.transform(X_train)

# Llevamos a cabo el proceso de etrenamiento con validación-cruzada y búsqueda de malla.
# Observa que de acuerdo a las opciones incluidas en la malla, se estarán realizando (6)(5)=30
# combinaciones diferentes, además de las (10)(5)=50 particiones de la validación-cruzada,
# lo cual implica también un mayor tiempo de entrenamiento.

grid_LR.fit(Xx_train, np.ravel(y_train))

print('Mejor valor de exactitud obtenido con la mejor combinación:', grid_LR.best_score_)
print('Mejor combinación de valores encontrados de los hiperparámetros:', grid_LR.best_params_)
print('Métrica utilizada: F1-Score a traves de la función ', grid_LR.scoring)
```

```
Mejor valor de exactitud obtenido con la mejor combinación: 0.8367398458343998
Mejor combinación de valores encontrados de los hiperparámetros: {'C': 1.0, 'class_weight': None, 'penalty': 'l2', 'sol
Métrica utilizada: F1-Score a traves de la función  make_scorer(calculate_f1score)
```

```
my_ModelnLearningCurve(X_train, models=LogisticRegression(C=1.0, class_weight=None, penalty= 'l2', solver= 'liblinear'), mod
```

```
Curvas de aprendizaje del LogisticRegression
```



¿Cuál es la utilidad de la métrica “f1-score”? Incluye tus conclusiones.

En el caso de la evaluación de los modelos, el uso de f1-score, denominada como media armónica, es útil para estos casos donde deseamos que las variables de menor peso estén consideradas dentro del resultado. En otras palabras, cuando el costo de falsos positivos o falsos negativos tienen la misma importancia. Creemos que este podría ser una buena métrica de desempeño para este ejercicio del riesgo crediticio que representa una persona donde no tenemos claro que es más costoso para la empresa: perder la oportunidad de darle un crédito a un cliente (falso negativo) o darle un crédito a un cliente que potencialmente nos va quedar mal con sus pagos (falso positivo).

Ahora, en cuanto a la gráfica, podemos ver que hubo un lígero cambio en el sobreentrenamiento del modelo. Es decir, podemos darnos cuenta de que hay una menor diferencia entre las curvas de entrenamiento y validacion. Esto principalmente por la incoporación la de la penalización. No obstante, no hubo cambio significativo respecto al subentrenamiento del modelo. Es decir, no vemos mayor desempeño en la métrica de f1-score todavía despues de escoger los "mejores" parámetros. Esto le da sustento a nuestra hipotesis inicial de que el modelo no es suficiente para describir la complejidad de nuestros datos.

Referencias:

Lipton, Z. C., Elkan, C., & Narayanaswamy, B. (2014). Thresholding classifiers to maximize F1 score. arXiv preprint arXiv:1402.1892. Recuperado el 24 de octubre de 2022 en: https://arxiv.org/abs/1402.1892

Géron, Aurélien (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition. O'Reilly. Chapter 3 Classification.

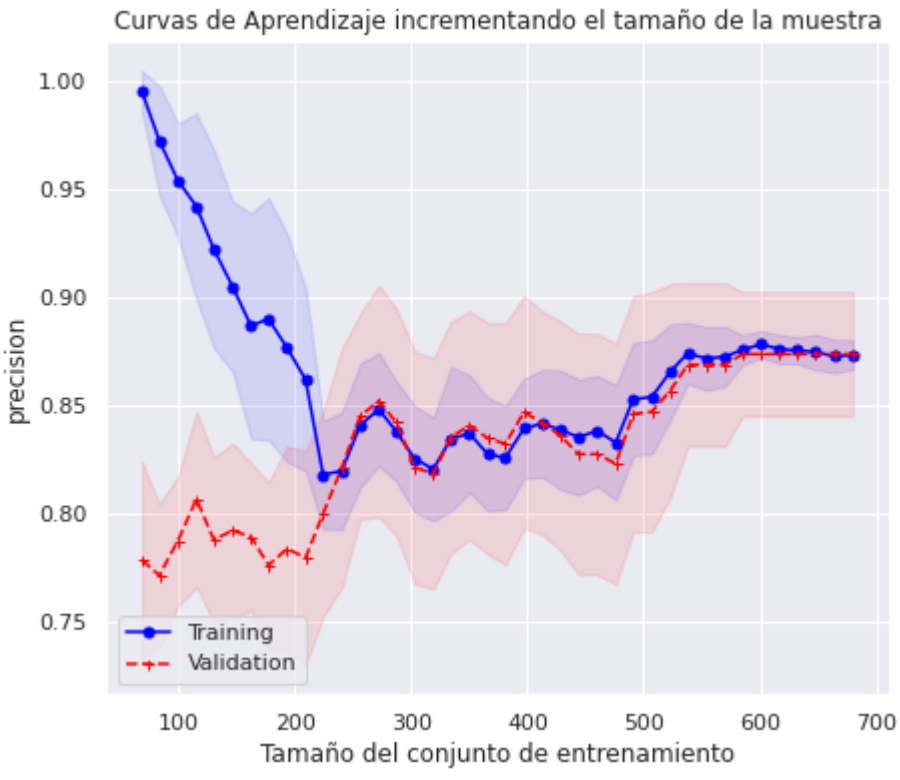
```
evaluating_model = models[1]

dicc_grid = {'ccp_alpha':[0,0.005,0.01,0.015, 0.02], 'criterion':['gini', 'entropy'],
             'max_depth':[5,10,15,20,25],
             'min_samples_split': [2,4,6,8], 'class_weight':['balanced', None]}

cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3)
```

```
grid_DecisionTree = GridSearchCV(estimator=evaluating_model,
                     param_grid=dicc_grid,
                     cv=cv,
                     scoring=metrics['precision'],
                     n_jobs = -1)


grid_DecisionTree.fit(Xx_train, np.ravel(y_train))

print('Mejor valor de exactitud obtenido con la mejor combinación:', grid_DecisionTree.best_score_)
print('Mejor combinación de valores encontrados de los hiperparámetros:', grid_DecisionTree.best_params_)
print('Métrica utilizada: Precision a traves de la función ', grid_DecisionTree.scoring)
```

```
Mejor valor de exactitud obtenido con la mejor combinación: 0.8733594082281685
Mejor combinación de valores encontrados de los hiperparámetros: {'ccp_alpha': 0.02, 'class_weight': 'balanced', 'crite
Métrica utilizada: Precision a traves de la función  make_scorer(calculate_precision)
```

```
my_ModelnLearningCurve(X_train, models=DecisionTreeClassifier(ccp_alpha= 0.02, class_weight= 'balanced', criterion= 'gini', r
```

```
Curvas de aprendizaje del DecisionTree
```



¿Cuál es la utilidad de la métrica "precision"? Incluye tus conclusiones.

El uso de la métrica de precision en el modelo de árbol de decisión nos permite identificar cómo es que el modelo hace las clasificaciones correctamente con relación a sus verdaderos positivos y todos los valores positivos encontrados en el modelo. Especificamente, nuestra gráfica nos esta diciendo que el modelo de Árbol de clasificación tiene un ~88% de probabilidad de no etiquetar como positiva una muestra que es negativa.

A comparación de la gráfica anterior, donde no definimos parámetros, el modelo parecer haber limitado la profundidad del árbol y la pureza de sus hojas. Esto disminuyo el desempeño de nuestro modelo de un 100% a un 88%, pero nos ayudo a combatir el sobre

entrenamiento. Ahora, nuestro modelo tiene practicamente la misma efectividad al usar el conjunto de validación que el de entrenamiento.

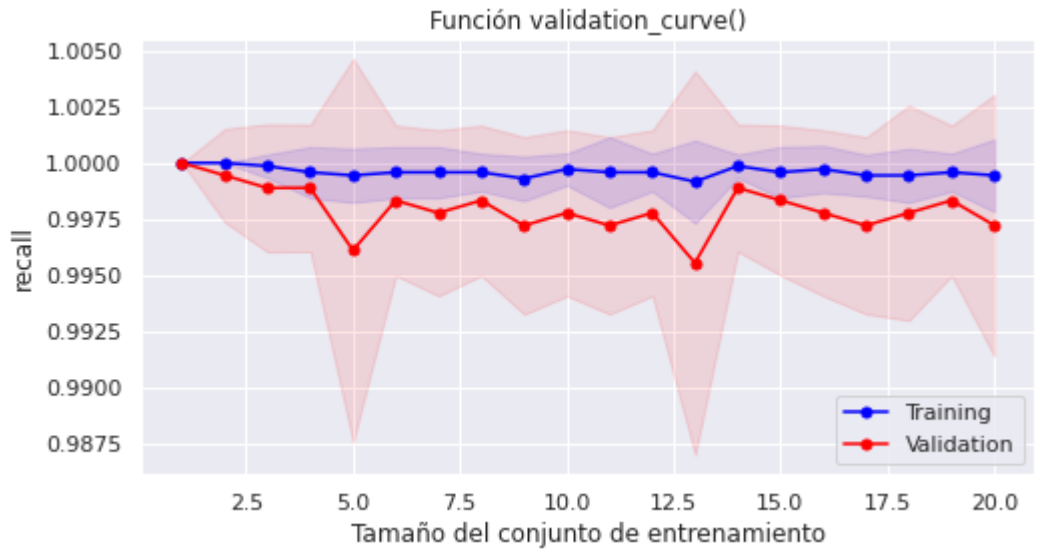
Alam, T. M., Shaukat, K., Hameed, I. A., Luo, S., Sarwar, M. U., Shabbir, S., Li, J. & Khushi, M. (2020). An investigation of credit card default prediction in the imbalanced datasets. IEEE Access, 8, 201173-201198. Disponible en:

```
evaluating_model = models[2]

dicc_grid = {'ccp_alpha':[0,0.005,0.01,0.015, 0.02], 'criterion':['gini', 'entropy'],
             'max_depth':[3,5,7,10,13,15,17,20,23,25],
             'min_samples_split': [2,4,6,8], 'class_weight':['balanced', None]}

cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3)

grid_RandomForest = GridSearchCV(estimator=evaluating_model,
                    param_grid=dicc_grid,
                    cv=cv,
                    scoring=metrics['recall'],
                    n_jobs = -1)


grid_RandomForest.fit(Xx_train, np.ravel(y_train))

print('Mejor valor de exactitud obtenido con la mejor combinación:', grid_RandomForest.best_score_)
print('Mejor combinación de valores encontrados de los hiperparámetros:', grid_RandomForest.best_params_)
print('Métrica utilizada: Recall a traves de la función ', grid_RandomForest.scoring)
```

```
Mejor valor de exactitud obtenido con la mejor combinación: 1.0
Mejor combinación de valores encontrados de los hiperparámetros: {'ccp_alpha': 0.01, 'class_weight': None, 'criterion':
Métrica utilizada: Recall a traves de la función  make_scorer(calculate_recall)
```

```
my_ModelnValidationCurve(X_train, models=RandomForestClassifier(ccp_alpha= 0.01, class_weight= None, criterion= 'gini', max_
```

```
Curvas de validación del RandomForest
```



¿Cuál es la utilidad de la métrica “recall”? Incluye tus conclusiones.

El uso de la métrica de recall en el modelo de árboles aleatorios es similar al de precisión, ya que nos permite identificar cómo es que el modelo hace las clasificaciones correctamente con relación a sus verdaderos positivos y todos los valores reales positivos que entran al modelo. A diferencia de la precisión, el calculo de recall no considera los falsos positivos, pero sí los falsos negativos. Por ende, recall es la capacidad del clasificador para encontrar todas las muestras positivas.

En nuestra gráfica podemos darnos cuenta que nuestro modelo es muy bueno para encontrar todas las muestras positivas. En este caso, la definición de parametros nos ayudo combatir el sobre-entrenamiento dado por una varianza del 10% entre nuestros conjuntos de datos. Además conservamos un muy alto desempeño. Parece ser un modelo perfecto, habría que confirmar este resultado con el siguiente ejercicio (Ejercicio 6) para poder corroborar si con datos completamente nuevos, el modelo nos seguirá dando este desempeño.

Alam, T. M., Shaukat, K., Hameed, I. A., Luo, S., Sarwar, M. U., Shabbir, S., Li, J. & Khushi, M. (2020). An investigation of credit card default prediction in the imbalanced datasets. IEEE Access, 8, 201173-201198. Disponible en: https://ieeexplore.ieee.org/document/9239944

Ejercicio-6.

Para cada uno de estos tres modelos, con las métricas que se consideraron en cada caso y usando el conjunto de Prueba que no has utilizado hasta ahora, obtener los modelos finales como se te indica a continuación. Deberás usar además como conjunto de

entrenamiento el llamado modelo de entrenamiento “aumentado” que consiste en las datos que estuviste utilizando para entrenamiento y validación:

1. Obtener el modelo de regresión logística con los mejores parámetros que hayas encontrado con la métrica f1-score utilizada. Imprimir el valor de dicha métrica e incluye tus conclusiones finales para este caso. Incluir un gráfico del árbol de decisión final obtenido.
2. Obtener el modelo de árbol de decisiones con los mejores parámetros que hayas encontrado con la métrica “precision” utilizada. Imprimir el valor de dicha métrica e incluye tus conclusiones finales para este caso.
3. Obtener el modelo de bosque aleatorio con los mejores parámetros que hayas encontrado con la métrica “recall” utilizada. Imprimir el valor de dicha métrica e incluye tus conclusiones finales para este caso

```
#Definimos nuestro modelo
final_model_LR= LogisticRegression(C=1.0, class_weight=None, penalty= 'l2', solver= 'liblinear')

#Lo entrenamos por ultima vez
Xc_train=columnasTransformer.fit(X_train)
Xx_train=Xc_train.transform(X_train)
final_model_LR.fit(Xx_train, np.ravel(y_train))

#Transformamos los datos de prueba y los utilizamos por primera vez para obtener sus predicciones
# y desempeño del modelo:
Xx_test = Xc_train.transform(X_test)
print('Accuracy final del modelo: ', final_model_LR.score(Xx_test, np.ravel(y_test)))

pp = final_model_LR.predict(Xx_test)
print('f1_Score final del modelo: ', calculate_f1score(y_test,pp) )
```

```
Accuracy final del modelo:  0.74
f1_Score final del modelo:  0.8202764976958525
```

Conclusión - Modelo de Regresión Logística:

El modelo de regresión logística parece balancear los efectos de la precisión (precision) y el sensibilidad (recall), ya que el valor "f1-score" es alto. La exactitud del modelo es mejor que la del modelo de árbol de decisión y la del modelo de arboles aleatorios, o sea que produce más predicciones de los tipos positivos verdaderos y negativos verdaderos.

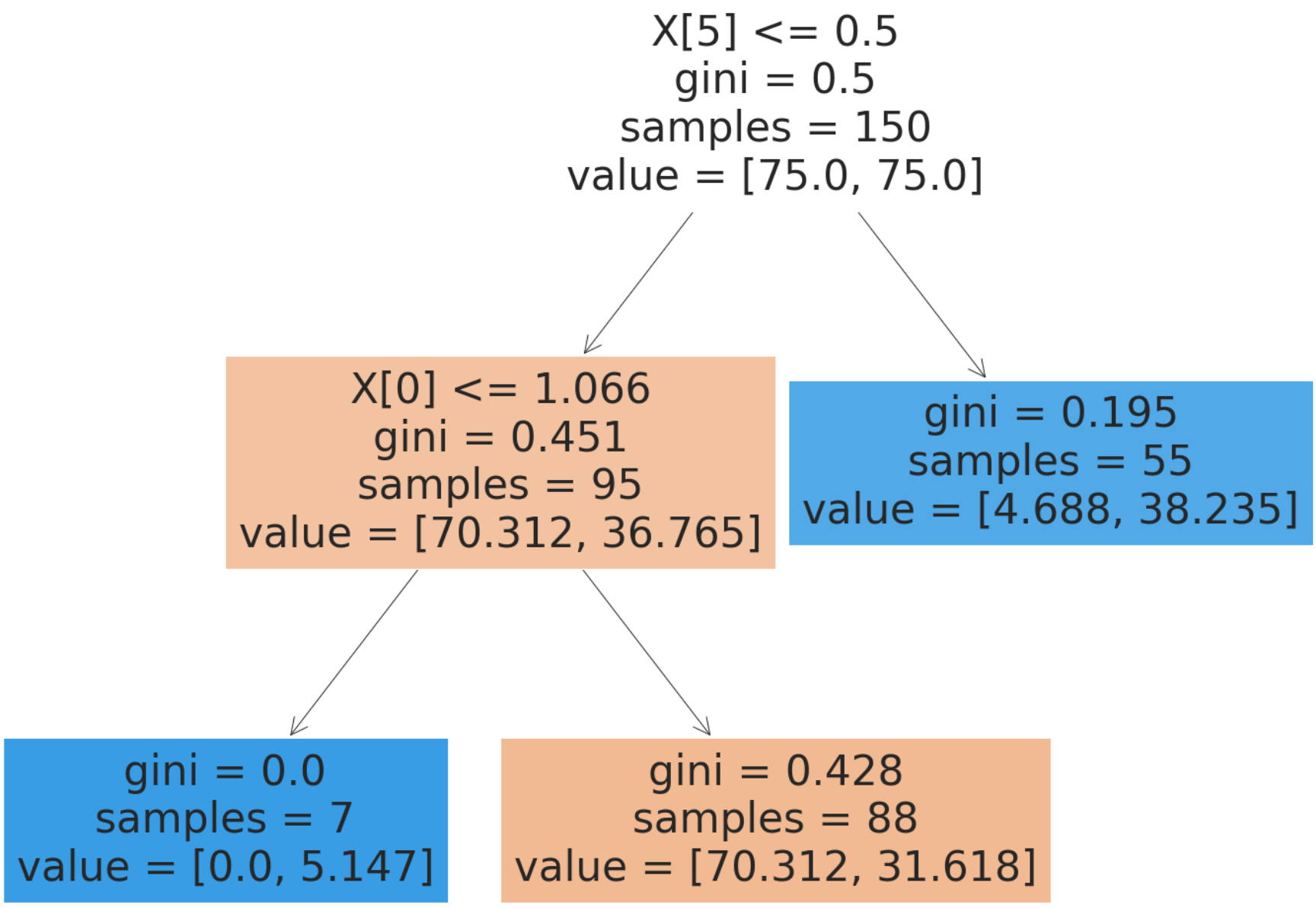
```
#Definimos nuestro modelo
final_model_DTC= DecisionTreeClassifier(ccp_alpha= 0.02, class_weight= 'balanced', criterion= 'gini', max_depth= 5, min_samp

#Lo entrenamos por ultima vez
Xc_train=columnasTransformer.fit(X_train)
Xx_train=Xc_train.transform(X_train)
final_model_DTC.fit(Xx_train, np.ravel(y_train))

#Transformamos los datos de prueba y los utilizamos por primera vez para obtener sus predicciones
# y desempeño del modelo:
Xx_test = Xc_train.transform(X_test)
print('Accuracy final del modelo: ', final_model_DTC.score(Xx_test, np.ravel(y_test)))

pp_DTC = final_model_DTC.predict(Xx_test)
print('Precision final del modelo: ', calculate_precision(y_test,pp_DTC) )
```

```
Accuracy final del modelo:  0.6466666666666666
Precision final del modelo:  0.9454545454545454
```

Conclusión - Modelo de Arbol de Decisiones:

El modelo árbol de decisiones por otro lado parece ser un modelo "más optimista" ya que la precisión (precision) es alta. En este caso trata de minimizar los incidentes donde buenos clientes sean tratados como malos. Como la exactitud (accuracy) no es alta (64.66 %) parece que el modelo produce muchas predicciones del tipo negativos falsos.

```
from sklearn import tree
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(final_model_DTC.fit(Xx_test, np.ravel(y_test)),
                   filled=True)
```




```
#Definimos nuestro modelo
final_model_RF= RandomForestClassifier(ccp_alpha= 0.01, class_weight= None, criterion= 'gini', max_depth= 3, min_samples_spl

#Lo entrenamos por ultima vez
Xc_train=columnasTransformer.fit(X_train)
Xx_train=Xc_train.transform(X_train)
final_model_RF.fit(Xx_train, np.ravel(y_train))

#Transformamos los datos de prueba y los utilizamos por primera vez para obtener sus predicciones
# y desempeño del modelo:
Xx_test = Xc_train.transform(X_test)
print('Accuracy final del modelo: ', final_model_RF.score(Xx_test, np.ravel(y_test)))

pp_RF = final_model_RF.predict(Xx_test)

print('Recall final del modelo: ', calculate_recall(y_test,pp_RF) )

     Accuracy final del modelo:  0.68
     Recall final del modelo:  1.0
```

Conclusión - Modelo de Árboles Aleatorios:

Con lo que respecta al modelo de arboles aleatorios la sensibilidad (recall) igual a 1, lo que indica que es un modelo "más pesimista" ya que no produce decisiones del tipo negativos falsos, o sea casos en que un cliente con riesgo de crédito alto o malo se le trate como un cliente con un buen historial. Como la exactitud (accuracy) no es alta (68%) esto parece indicar que este modelo tiende a incrementar las predicciones del tipo positivos falsos.

Referencias para las conclusiones finales

Kanstrén, T., (11 Septiembre, 2020), "*A Look at Precision, Recall and F1-Score*", Towards Data Science, https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec

Fin de la Actividad de la semana 6.