### MODELO RANDOM FOREST #####

import matplotlib.pyplot as plt

#import seaborn as sns

from sklearn.metrics import RocCurveDisplay

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (

confusion\_matrix, classification\_report,

mean\_absolute\_error, mean\_squared\_error, roc\_auc\_score

)

Import shop

#csv

df = pd.read\_csv("BankChurners.csv")

df = df.drop(columns=[

'CLIENTNUM',

'Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_1',

'Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_2'

])

# Target Binaria

df['Attrition\_Flag'] = df['Attrition\_Flag'].apply(lambda x: 1 if x == 'Attrited Customer' else 0)

# encoding de variables categóricas

X = pd.get\_dummies(df.drop(columns='Attrition\_Flag'), drop\_first=True)

y = df['Attrition\_Flag']

#### TRAIN #####

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, test\_size=0.2, random\_state=42)

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_proba = model.predict\_proba(X\_test)[:, 1]

### DESEMPEÑO DEL MODELO ######

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

mape = np.mean(np.abs((y\_test - y\_pred) / np.maximum(y\_test, 1))) \* 100

auc = roc\_auc\_score(y\_test, y\_proba)

print("Matriz de Confusión:\n", conf\_matrix)

print("\nReporte de Clasificación:\n", report)

print(f"MAE: {mae:.4f}")

print(f"RMSE: {rmse:.4f}")

print(f"MAPE: {mape:.2f}%")

print(f"AUC ROC: {auc:.4f}")

resultados = X\_test.copy()

resultados['Real'] = y\_test.values

resultados['Predicho'] = y\_pred

resultados['Probabilidad\_Attrited'] = y\_proba

resultados.to\_csv("predicciones\_random\_forest.csv", index=False)

# Matriz de confusión

#plt.figure(figsize=(6, 5))

#sns.heatmap(conf\_matrix\_rf, annot=True, fmt="d", cmap="Blues",

# xticklabels=["Existing", "Attrited"], yticklabels=["Existing", "Attrited"])

#plt.title("Matriz de Confusión")

#plt.xlabel("Predicción")

#plt.ylabel("Real")

#plt.tight\_layout()

#plt.show()

# Curva ROC

RocCurveDisplay.from\_predictions(y\_test, y\_proba)

plt.title("Curva ROC - Random Forest")

plt.grid(True)

plt.show()

#### MODELO DE LIGHTGBM ######

import pandas as pd

import numpy as np

import lightgbm as lgb

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import (

confusion\_matrix, classification\_report,

mean\_absolute\_error, mean\_squared\_error, roc\_auc\_score,

RocCurveDisplay

)

import matplotlib.pyplot as plt

#import seaborn as sns

df = pd.read\_csv("BankChurners.csv")

df = df.drop(columns=[

'CLIENTNUM',

'Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_1',

'Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_2'

])

# target

df['Attrition\_Flag'] = df['Attrition\_Flag'].apply(lambda x: 1 if x == 'Attrited Customer' else 0)

X = pd.get\_dummies(df.drop(columns='Attrition\_Flag'), drop\_first=True)

y = df['Attrition\_Flag']

### TRAIN ####

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, test\_size=0.2, random\_state=42)

model = lgb.LGBMClassifier(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_proba = model.predict\_proba(X\_test)[:, 1]

####### DESEMPEÑO #######

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

mape = np.mean(np.abs((y\_test - y\_pred) / np.maximum(y\_test, 1))) \* 100

auc = roc\_auc\_score(y\_test, y\_proba)

print("Matriz de Confusión:\n", conf\_matrix)

print("\nReporte de Clasificación:\n", report)

print(f"MAE: {mae:.4f}")

print(f"RMSE: {rmse:.4f}")

print(f"MAPE: {mape:.2f}%")

print(f"AUC ROC: {auc:.4f}")

# Matriz de confusión

plt.figure(figsize=(6, 5))

#sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues",

# xticklabels=["Existing", "Attrited"], yticklabels=["Existing", "Attrited"])

plt.title("Matriz de Confusión - LightGBM")

plt.xlabel("Predicción")

plt.ylabel("Real")

plt.tight\_layout()

plt.show()

#ROC

RocCurveDisplay.from\_predictions(y\_test, y\_proba)

plt.title("Curva ROC - LightGBM")

plt.grid(True)

plt.show()

importances = model.feature\_importances\_ #IMPORTANCIAS DE VARIABLES

features = X.columns

importances\_df = pd.DataFrame({

'Feature': features,

'Importance': importances

}).sort\_values(by='Importance', ascending=False)

importances = model.feature\_importances\_

features = X.columns

importances\_df = pd.DataFrame({

'Feature': features,

'Importance': importances

}).sort\_values(by='Importance', ascending=False)

top\_features = importances\_df.head(20)

plt.figure(figsize=(10, 8))

plt.barh(top\_features['Feature'][::-1], top\_features['Importance'][::-1])

plt.xlabel("Importancia")

plt.title("Top 20 Feature Importances - LightGBM")

plt.tight\_layout()

plt.show()

# Guardar predicciones

resultados = X\_test.copy()

resultados['Real'] = y\_test.values

resultados['Predicho'] = y\_pred

resultados['Probabilidad\_Attrited'] = y\_proba

resultados.to\_csv("predicciones\_lightgbm.csv", index=False)

# Inicializar explicador SHAP

explainer = shap.TreeExplainer(model)

shap\_values = explainer.shap\_values(X\_test)

# Gráfico resumen

shap.summary\_plot(shap\_values[1], X\_test, plot\_type="bar") # para la clase Attrited (1)

shap.summary\_plot(shap\_values[1

], X\_test) # gráfico de dispersión

import matplotlib.pyplot as plt

import seaborn as sns

# Copiar y codificar la target

df\_viz = df.copy()

df\_viz['Attrition\_Flag'] = df\_viz['Attrition\_Flag'].apply(lambda x: 'Attrited' if x == 'Attrited Customer' else 'Existing')

# Seleccionar algunas variables numéricas clave

numeric\_features = [

'Customer\_Age', 'Dependent\_count', 'Credit\_Limit', 'Total\_Revolving\_Bal',

'Avg\_Open\_To\_Buy', 'Total\_Trans\_Amt', 'Total\_Trans\_Ct',

'Total\_Amt\_Chng\_Q4\_Q1', 'Total\_Ct\_Chng\_Q4\_Q1', 'Avg\_Utilization\_Ratio'

]

# Crear boxplots para comparar entre clases

for feature in numeric\_features:

plt.figure(figsize=(8, 5))

sns.boxplot(data=df\_viz, x='Attrition\_Flag', y=feature)

plt.title(f'{feature} por clase de cliente')

plt.xlabel('Clase (Attrition\_Flag)')

plt.ylabel(feature)

plt.tight\_layout()

plt.show()

sin seaborn

import matplotlib.pyplot as plt

#csv

df = pd.read\_csv("BankChurners.csv")

# Copiar y codificar la target

df\_viz = df.copy()

df\_viz['Attrition\_Flag'] = df\_viz['Attrition\_Flag'].apply(lambda x: 'Attrited' if x == 'Attrited Customer' else 'Existing')

# Variables numéricas a comparar

numeric\_features = [

'Customer\_Age', 'Dependent\_count', 'Credit\_Limit', 'Total\_Revolving\_Bal',

'Avg\_Open\_To\_Buy', 'Total\_Trans\_Amt', 'Total\_Trans\_Ct',

'Total\_Amt\_Chng\_Q4\_Q1', 'Total\_Ct\_Chng\_Q4\_Q1', 'Avg\_Utilization\_Ratio'

]

# Crear boxplots con matplotlib

for feature in numeric\_features:

existing = df\_viz[df\_viz['Attrition\_Flag'] == 'Existing'][feature]

attrited = df\_viz[df\_viz['Attrition\_Flag'] == 'Attrited'][feature]

plt.figure(figsize=(8, 5))

plt.boxplot([existing, attrited], labels=['Existing', 'Attrited'], patch\_artist=True)

plt.title(f'{feature} por clase de cliente')

plt.xlabel('Clase (Attrition\_Flag)')

plt.ylabel(feature)

plt.grid(True, linestyle='--', alpha=0.6)

plt.tight\_layout()

plt.show()