# manejo de datos

import numpy as np

import pandas as pd

# imputación de datos, preprocesamiento

from sklearn.experimental import enable\_iterative\_imputer

from sklearn.impute import IterativeImputer

# transformación, por si nos sirve

from sklearn.preprocessing import MinMaxScaler

# seleccion de modelos (validacion cruzada)

from sklearn.model\_selection import cross\_val\_score

from sklearn import model\_selection

# metricas

from sklearn.metrics import recall\_score, precision\_score, accuracy\_score

from sklearn.metrics import classification\_report, confusion\_matrix

# visualización

import matplotlib.pyplot as plt

import seaborn as sns

from soditools import soditools as u

sns.set(style='white', palette='inferno')

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

data = u.query("SELECT customer\_id\_sod, compra\_aseo, asistencia,compra\_decoracion,compra\_ferreteria,compra\_jardin,compra\_menaje,compra\_organizacion,compra\_pintura,desarrollo,maduro,recencia FROM `sod-cl-bi-sandbox.sbx\_crm\_symphony\_diarios.tbl\_tablon\_seba\_imput`")

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

#REVISAMOS LOS VALORES NULOS POR FILAS

data.isna().sum()

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

#TRANSFORMAR A LOGARITMO PARA EVITAR SESGO

data['log\_venta'] = np.log(data['venta']+1)

data.head(3)

# acá generamos una grilla de (filas, columnas) donde podemos insertar gráficos

#fig, axes = plt.subplots(1,2,figsize=(12,4))

# grafico 1. el parametro ax = axes[0] nos dice donde ubicamos el grafico

#a = sns.histplot(data=data, x='venta',ax=axes[0])

# grafico 2. el parametro ax = axes[1] nos dice donde ubicamos el grafico

#b = sns.histplot(x='log\_venta', data=data, ax=axes[1])

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

columns\_to\_include\_2 = ['recencia','asistencia','venta','rango\_18\_30','rango\_31\_40','rango\_41\_50',

'rango\_51\_60','rango\_61\_70','rango\_71\_80','rango\_81\_90','fuera\_rango\_edad',

'sin\_edad','adquisicion','reactivado','nuevo','desarrollo','potencial','maduro','retencion',

'recuperacion','fugado','navegacion\_anual','preaprobado','cmr','normal\_evaluacion','normal\_blacklist',

'categoria\_loyalty','compra\_pintura','compra\_ferreteria','compra\_pisos','compra\_herramientas',

'compra\_banos','compra\_airelibre','compra\_jardin','compra\_electrohogar','compra\_muebles',

'compra\_decoracion','compra\_iluminacion','compra\_automovil','compra\_menaje','compra\_organizacion',

'compra\_aseo','compra\_casainteligente','numero\_envios','MARCA\_APP']

data\_2 = data[columns\_to\_include\_2]

data\_2.columns.tolist()

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

#HEAT MAP PARA REVISAR VARIABLES ALTAMENTE CORRELACIONADAS

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

plt.figure(figsize=(24,24))

# Crear una matriz de correlación

corr\_matrix = data\_2.corr()

# Crear un mapa de calor para visualizar la correlación

mask = np.triu(np.ones\_like(corr\_matrix, dtype=bool))

# plot

#sns.heatmap(corr\_matrix,mask=mask,annot=True);

sns.heatmap(corr\_matrix, mask=mask, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

plt.show()

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

#Dejamos las variables que creemos que pueden servirnos

#columns\_to\_include\_3 = ['customer\_id\_sod','recencia','asistencia','rango\_18\_30','rango\_31\_40','rango\_41\_50',

# 'rango\_51\_60','rango\_61\_70','rango\_71\_80','rango\_81\_90','fuera\_rango\_edad',

# 'sin\_edad','adquisicion','reactivado','nuevo','desarrollo','potencial',

# 'maduro','retencion','recuperacion','fugado','navegacion\_anual','preaprobado',

# 'cmr','normal\_evaluacion','normal\_blacklist','categoria\_loyalty','compra\_jardin',

# 'numero\_envios','MARCA\_APP']

#data\_3 = data[columns\_to\_include\_3]

#data\_3.columns.tolist()

####################### Eliminamos variables

data = data.drop(['venta','sin\_edad','normal\_evaluacion','fuera\_rango\_edad'], axis=1)

###################### Nos quedamos con las variables más importantes

#columnas = ['customer\_id\_sod','compra\_ferreteria','asistencia','recencia','compra\_herramientas',

# 'compra\_pintura','maduro','compra\_electrohogar','compra\_muebles','navegacion\_anual',

# 'compra\_organizacion','desarrollo','recuperacion','compra\_decoracion','compra\_automovil',

# 'potencial','compra\_aseo','numero\_envios','rango\_31\_40'

# ]

#data =data [columnas]

#data.head()

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

columnas = ['compra\_muebles','customer\_id\_sod','log\_venta','navegacion\_anual','maduro','compra\_pintura','compra\_decoracion','compra\_organizacion',

'rango\_18\_30','compra\_pisos','compra\_electrohogar','recencia','desarrollo','reactivado','categoria\_loyalty',

'cmr','potencial','fugado'

]

data = data[columnas]

data.head()

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

def ecdf(data):

# Computa la distribución empirica acumulada (ECDF por sus siglas en ingles)

x = np.sort(data)

n = x.size

y = np.arange(1, n+1) / n

return x,y

def plot\_kolmogorov(sample1, sample2,variable\_name='',alpha=0.05):

import matplotlib.pyplot as plt

from scipy import stats

# obtenemos la ECDF para cada muestra y las ordenamos

x1,y1 = ecdf(sample1)

x2,y2 = ecdf(sample2)

# plot

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

sns.lineplot(x=x1, y=y1, color='red')

sns.lineplot(x=x2, y=y2, color='blue')

# test

stat, p\_val = stats.ks\_2samp(sample1, sample2)

# interpretamos el p-value

print(f"p-value: {round(p\_val,4)} | alpha: {alpha}")

if p\_val < alpha:

print('Las distribuciones son independientes (Rechaza Hipótesis Nula) | Existe relación entre las varibles')

else:

print('Las distribuciones son dependientes (Acepta Hipótesis Nula) | No existe relación entre las varibles')

plt.ylabel("Cumulative Distribution Function")

plt.xlabel(variable\_name)

plt.show()

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

data.columns

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

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

# La funcion trains test split nos ayuda a dividir de forma aleatoria, el grupo entreno y testeo

# split

#X\_train, X\_test, y\_train, y\_test = train\_test\_split(attributes, target, test\_size=0.33, seed) -- forma general

# debemos entregarle el vector de atributos, etiquetas y nos entrega las filas de atributos y etiquetas, X,Y

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[data.columns.difference(['compra\_aseo','customer\_id\_sod'])], data['compra\_aseo'], test\_size=0.2, random\_state=2000000)

# se tiene el conjunto de entrenamiento con las variables X, que ayudan al modelo a definir y los argumentos Y que son las respuestas a la modelacion, para conocer el porcentaje de acierto

# el primer argumento que se entrega es las variables X, los atributos

# las etiquetas son las que estan en la columna diabetes, que se entrega en la segunda posicion, como segundo argumento

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

#Revisamos el conjunto de entrenamiento y testeo

# Vemos que ocurre para cada variable

#X\_train.columns

X\_test.columns.tolist()

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

from sklearn.tree import DecisionTreeClassifier

class CustomThresholdDecisionTree(DecisionTreeClassifier):

def \_\_init\_\_(self, threshold=0.5, \*\*kwargs):

super().\_\_init\_\_(max\_depth=6, criterion='gini', \*\*kwargs)

self.threshold = threshold

def predict\_proba\_with\_threshold(self, X):

return super().predict\_proba(X)

def predict(self, X):

# Obtener las probabilidades sin aplicar el umbral

raw\_predictions = super().predict\_proba(X)[:, 1]

return raw\_predictions

# Crear una instancia del modelo

TREE\_2 = CustomThresholdDecisionTree(threshold=0.3, random\_state=100)

# Entrenar el modelo

TREE\_2.fit(X\_train, y\_train)

# Realizar predicciones con las probabilidades

y\_probabilities = TREE\_2.predict(X\_test)

# Aplicar el umbral fuera de la clase

y\_pred\_tree\_custom = (y\_probabilities > TREE\_2.threshold).astype(int)

# Imprimir el informe de clasificación

print('Reporte modelo arbol de decision:')

print(classification\_report(y\_test, y\_pred\_tree\_custom, target\_names=['NO', 'SI']))

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

from sklearn.tree import DecisionTreeClassifier

import pandas as pd

# Supongamos que 'X' son tus características y 'y' es tu variable objetivo

# Ajusta según tus datos

#X = pd.X\_({'Feature1': [1, 2, 3, 4], 'Feature2': [5, 6, 7, 8]})

#y = pd.Series([0, 1, 0, 1])

# Entrenar un modelo de árbol de decisión

#modelo\_arbol = DecisionTreeClassifier()

#modelo\_arbol.fit(X, y)

# Obtener la importancia de las características

importancia\_caracteristicas = TREE\_2.feature\_importances\_

# Asociar las importancias con el nombre de las características

df\_importancia = pd.DataFrame({'Caracteristica': X\_train.columns, 'Importancia': importancia\_caracteristicas})

# Ordenar por importancia descendente

df\_importancia = df\_importancia.sort\_values(by='Importancia', ascending=False)

# Imprimir la importancia de las características

print(df\_importancia)

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

data\_pre = u.query("SELECT customer\_id\_sod, asistencia,compra\_decoracion,compra\_ferreteria,compra\_jardin,compra\_menaje,compra\_organizacion,compra\_pintura,desarrollo,maduro,recencia FROM `sod-cl-bi-sandbox.sbx\_crm\_symphony\_diarios.tbl\_tablon\_seba\_output`")

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

data\_pre['log\_venta'] = np.log(data\_pre['venta']+1)

data.head(3)

# acá generamos una grilla de (filas, columnas) donde podemos insertar gráficos

#fig, axes = plt.subplots(1,2,figsize=(12,4))

# grafico 1. el parametro ax = axes[0] nos dice donde ubicamos el grafico

#a = sns.histplot(data=data\_pre, x='venta',ax=axes[0])

# grafico 2. el parametro ax = axes[1] nos dice donde ubicamos el grafico

#b = sns.histplot(x='log\_venta', data=data\_pre, ax=axes[1])

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

columns\_to\_include = ['customer\_id\_sod','asistencia','compra\_decoracion','compra\_ferreteria',

'compra\_jardin','compra\_menaje','compra\_organizacion','compra\_pintura',

'desarrollo','maduro','recencia']

date\_pre\_limpia = data\_pre[columns\_to\_include]

columns\_to\_include\_2 = ['asistencia','compra\_decoracion','compra\_ferreteria',

'compra\_jardin','compra\_menaje','compra\_organizacion','compra\_pintura',

'desarrollo','maduro','recencia']

date\_pre\_limpia\_2 = date\_pre\_limpia[columns\_to\_include\_2]

date\_pre\_limpia\_2.columns.tolist()

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

predicciones\_nuevos\_datos\_2 = TREE\_2.predict(date\_pre\_limpia\_2)

print("Predicciones sobre nuevos datos:")

print(predicciones\_nuevos\_datos\_2)

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

# ya con las predicciones listas procedemos a pegar la prediccion y migrar el dataframe

predicciones\_nuevos\_datos\_2 = pd.Series(predicciones\_nuevos\_datos\_2,name = 'predict')

data\_final = pd.concat([date\_pre\_limpia, predicciones\_nuevos\_datos\_2], axis=1)

data\_final

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

#Finalizacion del experimento

!pip install pandas-gbq

import pandas as pd

# Suponiendo que tienes un DataFrame llamado 'df'

#df=pd.DataFrame()

#df['id'] = [1,2,3]

#df['data'] = [4,5,6]

# sod-cl-bi-sandbox.sbx\_crm\_symphony\_diarios.tbl\_tablon\_seba\_predict\_2023

# Define el nombre del proyecto de GCP y el ID del conjunto de datos y de la tabla en BigQuery

project\_id = 'sod-cl-bi-sandbox'

dataset\_id = 'sbx\_crm\_symphony\_diarios'

table\_name = 'tbl\_tablon\_seba\_predict\_2024\_aseo'

# Carga el DataFrame en BigQuery

pd.io.gbq.to\_gbq(data\_final, f'{dataset\_id}.{table\_name}', project\_id=project\_id, if\_exists='replace')