

# PROYECTO FINAL - TALENTO TECH

## Implementación de Modelos de Aprendizaje Automático para la Predicción de Ataques Cardíacos

Profesor: Luis Fernando Gallego Hurtado

Estudiante: Laura Victoria Ramos Agudelo

### ✓ Contexto Caso de Estudio: Clasificación Biclase para Diagnóstico de Enfermedades Cardíacas

Este caso de estudio se enfoca en una tarea de clasificación biclase destinada a determinar la presencia o ausencia de enfermedades cardíacas en individuos mediante el análisis de variables de entrada clínicas y fisiológicas.

### Variables de Entrada

1. **Edad del paciente ( age )**
2. **Sexo ( sex )**
3. **Tipo de dolor torácico ( cp )**
4. **Presión arterial en reposo ( trestbps )**
5. **Colesterol sérico ( chol )**
6. **Azúcar en sangre en ayunas ( fbs )**
7. **Resultados electrocardiográficos en reposo ( restecg )**
8. **Frecuencia cardíaca máxima alcanzada ( thalach )**
9. **Presencia de angina inducida por el ejercicio ( exang )**
10. **Depresión del segmento ST inducida por ejercicio ( oldpeak )**
11. **Pendiente del segmento ST durante el ejercicio ( slope )**
12. **Número de vasos sanguíneos principales visibles en fluoroscopia ( ca )**
13. **Presencia de anomalías tiroideas ( thal )**

### Variable a Predecir

La variable objetivo es el **diagnóstico de enfermedad cardíaca ( num )**, que refleja el estado angiográfico del paciente:

- **0**: Sin presencia de estenosis significativa.
- **1**: Presencia significativa de estenosis que afecta más del 50% del diámetro arterial.

## ✓ 1. Análisis exploratorio de los datos

En esta sección, se realizará un análisis exploratorio de los datos para comprender mejor las características del conjunto de datos y obtener información valiosa sobre las variables y su relación con el objetivo de predicción.

### 1.1 Carga de datos

```
import pandas as pd
```

```
df = pd.read_csv('heart.csv')
df.sample(10)
```



	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	th
<b>132</b>	42	1	1	120	295	0	1	162	0	0.0	2	0	
<b>263</b>	63	0	0	108	269	0	1	169	1	1.8	1	2	
<b>237</b>	60	1	0	140	293	0	0	170	0	1.2	1	2	
<b>236</b>	58	1	0	125	300	0	0	171	0	0.0	2	2	
<b>29</b>	53	1	2	130	197	1	0	152	0	1.2	0	0	
<b>66</b>	51	1	2	100	222	0	1	143	1	1.2	1	0	
<b>144</b>	76	0	2	140	197	0	2	116	0	1.1	1	0	
<b>23</b>	61	1	2	150	243	1	1	137	1	1.0	1	0	
<b>188</b>	50	1	2	140	233	0	1	163	0	0.6	1	1	
<b>18</b>	43	1	0	150	247	0	1	171	0	1.5	2	0	

### ✓ 1.2 Inspección de datos faltantes

```
missing_values_count = df.isnull().sum()
missing_values_count
```

```

➦

```

	0
age	0
sex	0
cp	0
trtbps	0
chol	0
fbs	0
restecg	0
thalachh	0
exng	0
oldpeak	0
slp	0
caa	0
thall	0
output	0

```

dtype: int64

```

### ✓ 1.3 Exploración de columnas y tipos de datos

```
df.shape
```

```
➦ (303, 14)
```

La base de datos utilizada contiene 303 instancias y 14 características seleccionadas entre un total de 76 atributos disponibles. El valor de la clase de salida original es una variable de múltiples clases con un rango de valores de 0 a 4. Sin embargo, la versión de la base de datos en Kaggle ha sido transformada para que la variable de salida solo tome los valores de 0 y 1. En esta transformación, el valor 0 representa la ausencia de enfermedad cardíaca, mientras que los valores de 1 a 4 se agrupan bajo el valor 1, indicando la presencia de enfermedad cardíaca según el nivel de estenosis basado en el estado angiográfico del paciente.

```
import pandas as pd
```

```
# Función para identificar variables categóricas y continuas
```

```
def categorize_variables(df):
    categorical = []
    continuous = []

    for column in df.columns:
        if df[column].dtype == 'object' or df[column].nunique() <= 10:
            categorical.append(column)
        else:
            continuous.append(column)

    return {"Categorical": categorical, "Continuous": continuous}

# Supongamos que el DataFrame es `df`
# Proporciona tu DataFrame aquí
# df = pd.read_csv('tu_archivo.csv') # Carga tu dataset real

# Categorizar las variables
variable_types = categorize_variables(df)

# Crear un DataFrame con los resultados
results_df = pd.DataFrame([
    {"Variable": var, "Type": "Categorical"} for var in variable_types["Categorical"]
] + [
    {"Variable": var, "Type": "Continuous"} for var in variable_types["Continuous"]
])

# Mostrar los resultados
results_df
```



	Variable	Type
0	Modelo	Categorical
1	Precision	Categorical
2	Recall	Categorical
3	F1-Score	Categorical
4	Accuracy	Categorical

Atributo	Descripción	Tipo	Rango	Valores Únicos	Media	Mediana	Moda
Age	Edad del paciente en años completos	Continuo	29 a 77	41	54,4	55	58
Sex	Sexo del paciente (1 = masculino; 0 = femenino)	Categorico	0 a 1	2	0,68	1	1
CP	Tipo de dolor torácico (0 a 3) Valor 0: Angina Típica Valor 1: Angina Atípica Valor 2: Dolor No Anginoso Valor 3: Asintomático	Categorico	0 a 3	4	0,97	1	0
Trtbps	Presión arterial en reposo (mm Hg al ingreso al hospital)	Continuo	94 a 200	49	131,62	130	120
Chol	Colesterol sérico en mg/dl	Continuo	126 a 564	152	246,26	240	197
FBS	Azúcar en sangre en ayunas Valor 0: <= 120 mg/dL Valor 1: > 120 mg/dL	Categorico	0 a 1	2	0,15	0	0
Restecg	Resultados electrocardiográficos en reposo Valor 0: Normal Valor 1: Anormalidad de la onda ST-T Valor 2: Hipertrofia ventricular izquierda probable o definida	Categorico	0 a 2	3	0,53	1	1
Thalachh	Máxima frecuencia cardíaca alcanzada	Continuo	71 a 202	91	149,64	153	162
Exng	Angina inducida por el ejercicio Valor 0: No Valor 1: Sí	Categorico	0 a 1	2	0,33	0	0
Oldpeak	Depresión del ST inducida por el ejercicio respecto al reposo	Continuo	0 a 6.20	40	1,04	0,8	0
Slp	Pendiente del segmento ST del ejercicio	Categorico	0 a 2	3	1,4	1	2
Ca	Número de vasos principales (0-3) vistos en fluoroscopia	Categorico	0 a 4	5	0,73	0	0
Thall	Estado del corazón Valor 0: Ninguno (Normal) Valor 1: Defecto fijo Valor 2: Defecto reversible Valor 3: Talasemia	Categorico	0 a 3	4	2,31	2	2
Num (Objetivo)	Diagnóstico de enfermedad cardíaca (estado angiográfico) Valor 0: No Valor 1: Sí	Categorico	0 a 1	2	0,54	1	1

Tabla I. Descripción de las variables de la base de datos HD

```
for column in df.columns:
    unique_values = df[column].unique()
    unique_count = df[column].nunique()
```

```
print(f"Valores únicos en la columna {column}: {unique_values}")
print(f"Cantidad de valores únicos en {column}: {unique_count}")
```

```
→ Valores únicos en la columna age: [63 37 41 56 57 44 52 54 48 49 64 58 50 66 43
 46 45 39 47 62 34 35 29 55 60 67 68 74 76 70 38 77]
Cantidad de valores únicos en age: 41
Valores únicos en la columna sex: [1 0]
Cantidad de valores únicos en sex: 2
Valores únicos en la columna cp: [3 2 1 0]
Cantidad de valores únicos en cp: 4
Valores únicos en la columna trtbps: [145 130 120 140 172 150 110 135 160 105 12
 122 115 118 100 124 94 112 102 152 101 132 148 178 129 180 136 126 106
 156 170 146 117 200 165 174 192 144 123 154 114 164]
Cantidad de valores únicos en trtbps: 49
Valores únicos en la columna chol: [233 250 204 236 354 192 294 263 199 168 239
 247 234 243 302 212 175 417 197 198 177 273 213 304 232 269 360 308 245
 208 264 321 325 235 257 216 256 231 141 252 201 222 260 182 303 265 309
 186 203 183 220 209 258 227 261 221 205 240 318 298 564 277 214 248 255
 207 223 288 160 394 315 246 244 270 195 196 254 126 313 262 215 193 271
 268 267 210 295 306 178 242 180 228 149 278 253 342 157 286 229 284 224
 206 167 230 335 276 353 225 330 290 172 305 188 282 185 326 274 164 307
 249 341 407 217 174 281 289 322 299 300 293 184 409 259 200 327 237 218
 319 166 311 169 187 176 241 131]
Cantidad de valores únicos en chol: 152
Valores únicos en la columna fbs: [1 0]
Cantidad de valores únicos en fbs: 2
Valores únicos en la columna restecg: [0 1 2]
Cantidad de valores únicos en restecg: 3
Valores únicos en la columna thalachh: [150 187 172 178 163 148 153 173 162 174
 179 137 157 123 152 168 140 188 125 170 165 142 180 143 182 156 115 149
 146 175 186 185 159 130 190 132 147 154 202 166 164 184 122 169 138 111
 145 194 131 133 155 167 192 121 96 126 105 181 116 108 129 120 112 128
 109 113 99 177 141 136 97 127 103 124 88 195 106 95 117 71 118 134
 90]
Cantidad de valores únicos en thalachh: 91
Valores únicos en la columna exng: [0 1]
Cantidad de valores únicos en exng: 2
Valores únicos en la columna oldpeak: [2.3 3.5 1.4 0.8 0.6 0.4 1.3 0. 0.5 1.6 1
 0.1 1.9 4.2 1.1 2. 0.7 0.3 0.9 3.6 3.1 3.2 2.5 2.2 2.8 3.4 6.2 4. 5.6
 2.9 2.1 3.8 4.4]
Cantidad de valores únicos en oldpeak: 40
Valores únicos en la columna slp: [0 2 1]
Cantidad de valores únicos en slp: 3
Valores únicos en la columna caa: [0 2 1 3 4]
Cantidad de valores únicos en caa: 5
Valores únicos en la columna thall: [1 2 3 0]
Cantidad de valores únicos en thall: 4
Valores únicos en la columna output: [1 0]
Cantidad de valores únicos en output: 2
```

## ✓ 1.4 Estadísticas descriptivas

```
df.describe()
```



	age	sex	cp	trtbps	chol	fbs	restecg
<b>count</b>	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
<b>mean</b>	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053
<b>std</b>	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860
<b>min</b>	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
<b>25%</b>	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000
<b>50%</b>	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000
<b>75%</b>	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000
<b>max</b>	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000

```
# Calcular la matriz de correlación
correlation_matrix = df.corr()
```

```
# Mostrar la matriz de correlación
print("Matriz de correlación:")
print(correlation_matrix)
```



Matriz de correlación:

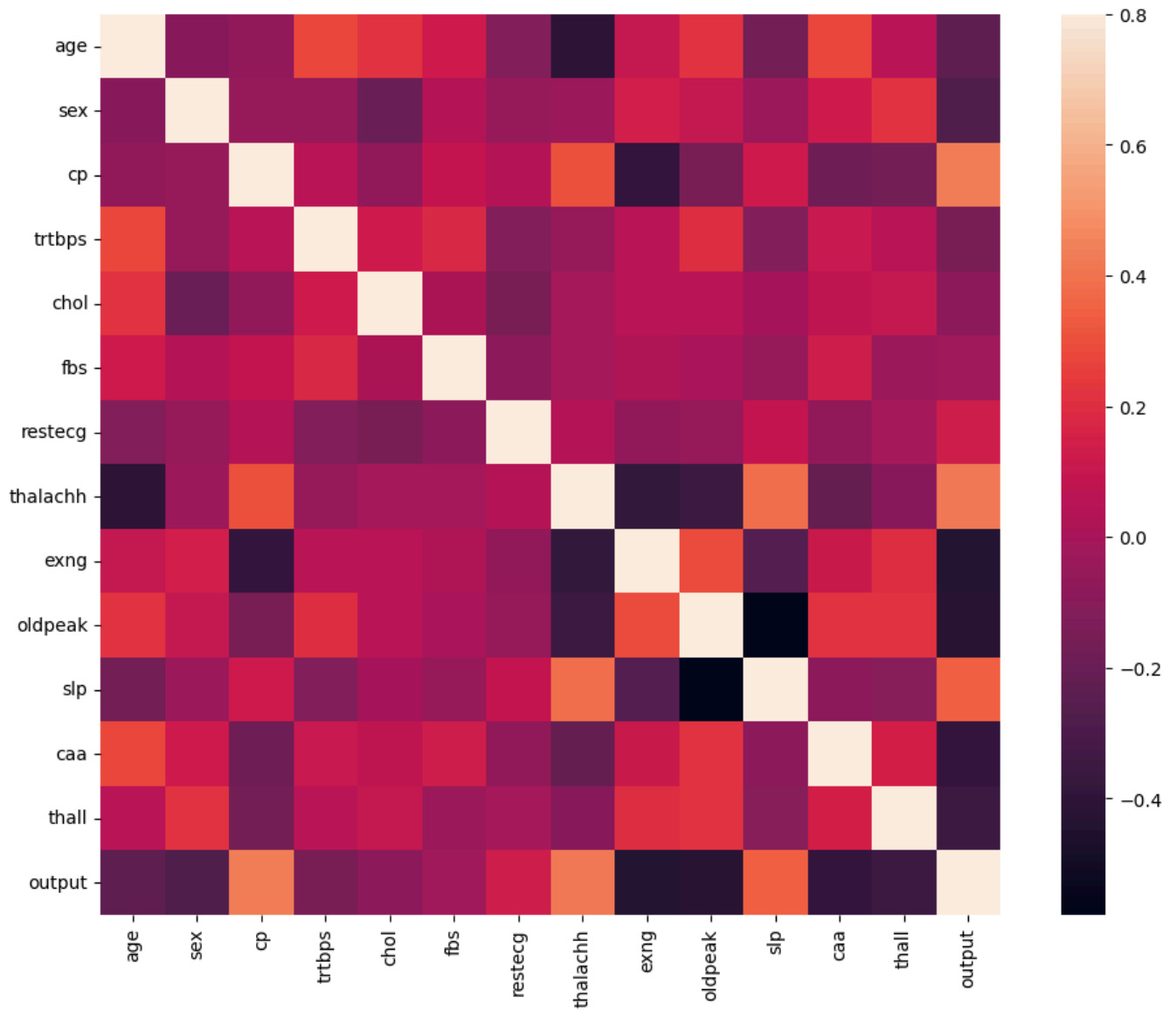
	age	sex	cp	trtbps	chol	fbs	\
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	
trtbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	
thalachh	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	
exng	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	
slp	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	
caa	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	
thall	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	
output	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	
	restecg	thalachh	exng	oldpeak	slp	caa	\
age	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	
sex	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	
cp	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	
trtbps	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	
chol	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	
fbs	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	
restecg	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	
thalachh	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	
exng	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	

oldpeak	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682
slp	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155
caa	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000
thall	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832
output	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391724

	thall	output
age	0.068001	-0.225439
sex	0.210041	-0.280937
cp	-0.161736	0.433798
trtbps	0.062210	-0.144931
chol	0.098803	-0.085239
fbs	-0.032019	-0.028046
restecg	-0.011981	0.137230
thalachh	-0.096439	0.421741
exng	0.206754	-0.436757
oldpeak	0.210244	-0.430696
slp	-0.104764	0.345877
caa	0.151832	-0.391724
thall	1.000000	-0.344029
output	-0.344029	1.000000

```
import seaborn as sns
import matplotlib.pyplot as plt
#correlation matrix
corrmat = df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```





## ✓ 2. Preparación de los datos

```
import numpy as np
from sklearn.preprocessing import StandardScaler
```

```
X = df.iloc[:, :-1].values
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Contar el número de muestras por cada valor de la columna 'output'
clases_count = df['output'].value_counts()

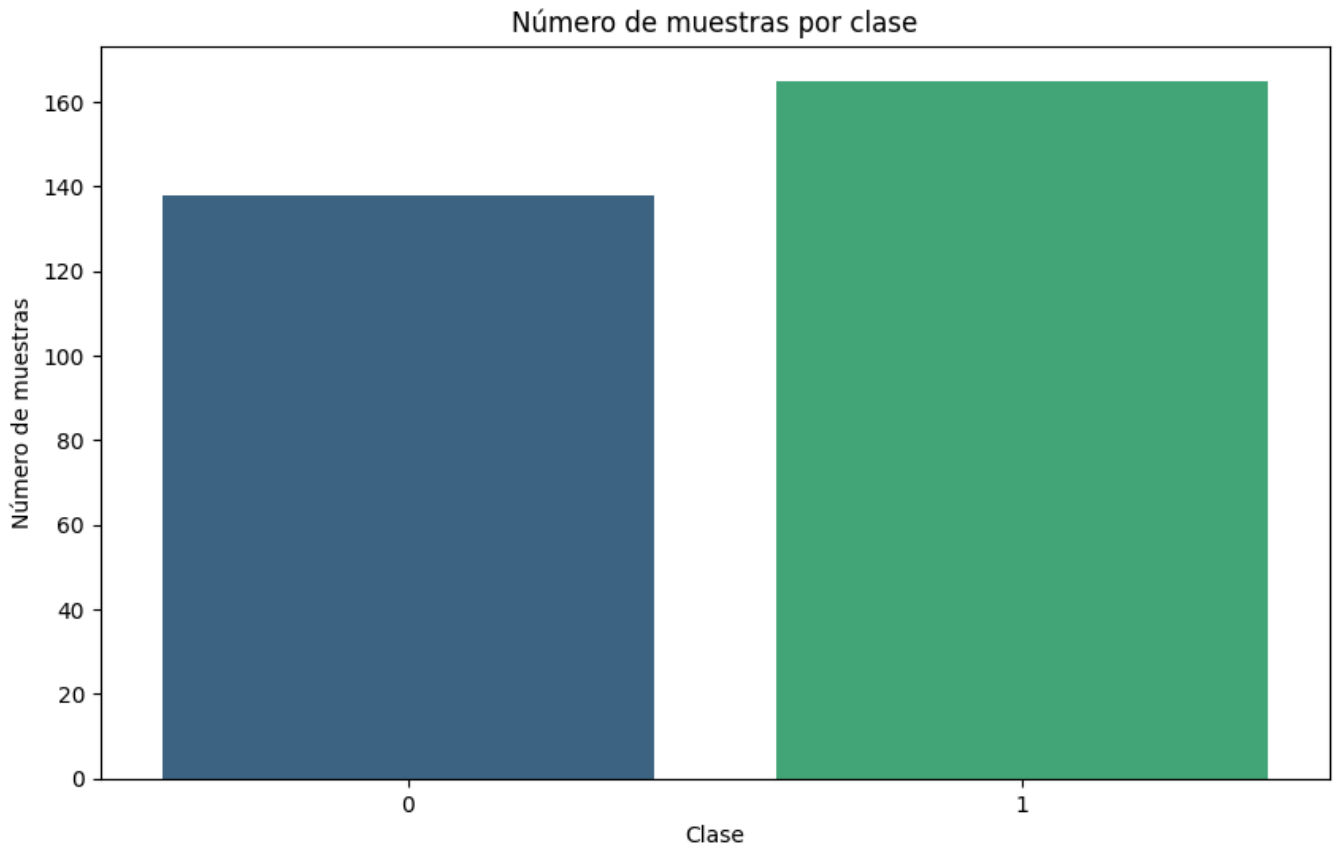
# Crear el gráfico de barras
plt.figure(figsize=(10, 6))
sns.barplot(x=clases_count.index, y=clases_count.values, palette='viridis')
plt.title('Número de muestras por clase')
plt.xlabel('Clase')
plt.ylabel('Número de muestras')
plt.xticks(rotation=0) # Ajusta la rotación según necesidad
plt.show()

# Imprimir el conteo de muestras por clase
print(clases_count)
```

 <ipython-input-55-1b84c5f7dab8>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v

```
sns.barplot(x=clases_count.index, y=clases_count.values, palette='viridis')
```



output

```
1    165
```

```
0    138
```

Name: count, dtype: int64

## ✓ Normalización de los datos

## ✓ Variables que necesitan normalización

En este proyecto, las variables con escalas diferentes que requieren normalización son:

1. **age (Edad del paciente):** Rango de 29 a 77.
2. **trtbps (Presión arterial en reposo):** Rango de 94 a 200.
3. **cho1 (Colesterol sérico):** Rango de 126 a 564.

4. **thalachh (Frecuencia cardíaca máxima alcanzada):** Rango de 71 a 202.

5. **oldpeak (Depresión del segmento ST inducida por ejercicio):** Rango de 0.0 a 6.2.

Estas variables tienen rangos amplios y podrían dominar a otras variables en el modelo si no se normalizan.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Asumiendo que 'df' es tu DataFrame original
# Separar la columna de salida 'output' del resto de las características
output = df['output']
features = df.drop('output', axis=1)

# Identificar variables categóricas (con hasta 4 categorías distintas) y continuas
categorical_columns = [col for col in features.columns if features[col].nunique() <= 4]
continuous_columns = [col for col in features.columns if col not in categorical_columns]

# 1. Normalizar las variables continuas
scaler = MinMaxScaler()
features[continuous_columns] = scaler.fit_transform(features[continuous_columns])

# 2. Aplicar One-Hot Encoding a las variables categóricas
features_encoded = pd.get_dummies(features, columns=categorical_columns)

# 3. Reconstruir el DataFrame final con la columna 'output'
normalized_df = pd.concat([features_encoded, output], axis=1)

# Mostrar el DataFrame normalizado y codificado
print(normalized_df.head())
```

```
⇒
```

	age	trtbps	chol	thalachh	oldpeak	sex_0	sex_1	cp_0	\
0	0.708333	0.481132	0.244292	0.603053	0.370968	False	True	False	
1	0.166667	0.339623	0.283105	0.885496	0.564516	False	True	False	
2	0.250000	0.339623	0.178082	0.770992	0.225806	True	False	False	
3	0.562500	0.245283	0.251142	0.816794	0.129032	False	True	False	
4	0.583333	0.245283	0.520548	0.702290	0.096774	True	False	True	

	cp_1	cp_2	...	caa_0	caa_1	caa_2	caa_3	caa_4	thall_0	thall_1	\
0	False	False	...	True	False	False	False	False	False	True	
1	False	True	...	True	False	False	False	False	False	False	
2	True	False	...	True	False	False	False	False	False	False	
3	True	False	...	True	False	False	False	False	False	False	
4	False	False	...	True	False	False	False	False	False	False	

	thall_2	thall_3	output
0	False	False	1
1	True	False	1
2	True	False	1

3	True	False	1
4	True	False	1

[5 rows x 31 columns]

```
# Calcular los rangos (mínimo y máximo) de cada columna en normalized_df
ranges = normalized_df.aggregate(['min', 'max']).T
ranges.columns = ['Min', 'Max']
ranges
```



	Min	Max
<b>age</b>	0.0	1.0
<b>trtbps</b>	0.0	1.0
<b>chol</b>	0.0	1.0
<b>thalachh</b>	0.0	1.0
<b>oldpeak</b>	0.0	1.0
<b>sex_0</b>	False	True
<b>sex_1</b>	False	True
<b>cp_0</b>	False	True
<b>cp_1</b>	False	True
<b>cp_2</b>	False	True
<b>cp_3</b>	False	True
<b>fbs_0</b>	False	True
<b>fbs_1</b>	False	True
<b>restecg_0</b>	False	True
<b>restecg_1</b>	False	True
<b>restecg_2</b>	False	True
<b>exng_0</b>	False	True
<b>exng_1</b>	False	True
<b>slp_0</b>	False	True
<b>slp_1</b>	False	True
<b>slp_2</b>	False	True
<b>caa_0</b>	False	True
<b>caa_1</b>	False	True
<b>caa_2</b>	False	True
<b>caa_3</b>	False	True
<b>caa_4</b>	False	True
<b>thall_0</b>	False	True
<b>thall_1</b>	False	True
<b>thall_2</b>	False	True
<b>thall_3</b>	False	True

output      0      1

## ✓ Proceso de Aumento de Datos Sintéticos

El aumento de datos sintéticos es una técnica empleada para incrementar la cantidad de datos disponibles en un conjunto inicial, permitiendo mejorar la capacidad de generalización de los modelos de aprendizaje automático. En este caso, se utilizó un enfoque basado en regresión logística para garantizar que los datos sintéticos sigan los patrones y relaciones presentes en los datos originales.

### Etapas del Proceso

#### 1. Entrenamiento de un Modelo de Regresión Logística:

- Se utiliza un modelo de regresión logística para capturar las relaciones lineales entre las características (variables independientes) y la variable de salida (objetivo).
- Este modelo genera coeficientes que reflejan la importancia relativa de cada variable en la predicción de la salida.

#### 2. Generación de Características Sintéticas:

- Se generan valores aleatorios para cada característica, manteniéndolos dentro de los rangos observados en los datos originales.
- Esto asegura que las nuevas muestras sintéticas sean realistas y representativas del dominio original.

#### 3. Cálculo de la Variable de Salida:

- Utilizando los coeficientes obtenidos del modelo de regresión logística, se aplica la función logística para calcular la probabilidad de que una muestra sintética pertenezca a la clase positiva.
- Estas probabilidades se convierten en valores binarios (0 o 1) para generar la variable de salida de forma coherente.

#### 4. Integración con los Datos Originales:

- Los datos sintéticos generados se combinan con los datos originales en un único conjunto, manteniendo las mismas columnas y formatos.
- Esto incrementa la cantidad de datos disponibles para el entrenamiento de modelos, mejorando potencialmente su capacidad de generalización.

### Ventajas del Enfoque

- **Preservación de las Relaciones Originales:**



- Al entrenar el modelo en los datos originales, se asegura que las relaciones estadísticas entre las variables se mantengan en los datos sintéticos.
- **Flexibilidad:**
  - Este método permite generar grandes cantidades de datos de manera controlada, ajustándose a los rangos y patrones del dominio de interés.
- **Mejora del Desempeño de Modelos:**
  - Los datos sintéticos ayudan a reducir problemas como el sobreajuste, proporcionando mayor diversidad en el conjunto de entrenamiento.

## Limitaciones Potenciales

- Si los datos originales tienen sesgos, estos se trasladan a los datos sintéticos.
- Los datos generados se limitan a patrones lineales capturados por la regresión logística, lo que puede no ser ideal para dominios con relaciones más complejas.

El enfoque utilizado asegura un balance entre simplicidad y representatividad, siendo ideal para escenarios donde los datos originales son limitados y los patrones lineales son predominantes.

```
# X son las características (todas menos la última columna)
# y es la variable objetivo (última columna)
normalized_df = normalized_df.rename(columns={'output_1': 'output'}) # Cambia 'outp

X = normalized_df.iloc[:, :-1]
y = normalized_df.iloc[:, -1]

# Identificar las variables categóricas (0 y 1)
categorical_columns = [col for col in X.columns if set(X[col].unique()).issubset({0,
continuous_columns = [col for col in X.columns if col not in categorical_columns]

# 1. Transponer la base de datos
X_transpose = X.T

# 2. Hallar los coeficientes mediante álgebra lineal
X_transpose_X = np.dot(X_transpose, X) # MMULT(Q1:LE13, A2:M302)
X_transpose_X_inv = np.linalg.inv(X_transpose_X) # MINVERSE(MMULT(Q1:LE13, A2:M302)
X_transpose_y = np.dot(X_transpose, y) # MMULT(Q1:LE13, N2:N302)
coefficients = np.dot(X_transpose_X_inv, X_transpose_y) # MMULT(Q17:AC29, Q33:Q45)

# 3. Generar datos sintéticos para las características
num_synthetic_samples = 10000
synthetic_features = pd.DataFrame()

# Generar valores aleatorios para las características
for column in X.columns:
    if column in categorical_columns:
```

```

# Para variables categóricas, generar valores binarios aleatorios (0 o 1)
synthetic_features[column] = np.random.choice([0, 1], size=num_synthetic_sam
else:
# Para variables continuas, generar valores aleatorios dentro de su rango
min_val = X[column].min()
max_val = X[column].max()
synthetic_features[column] = np.random.uniform(min_val, max_val, num_synthet

# 4. Calcular la variable de salida para los datos sintéticos
# Usar la fórmula logística: 1 / (1 + exp(-SUMPRODUCT(coefficients, features)))
logits = np.dot(synthetic_features, coefficients)
synthetic_features['output'] = np.round(1 / (1 + np.exp(-logits))).astype(int) # Re

# 5. Combinar los datos originales y los datos sintéticos
augmented_df = pd.concat([normalized_df, synthetic_features], ignore_index=True)

```

augmented\_df



	age	trtbps	chol	thalachh	oldpeak	sex	cp	fbs	restecg
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	3.000000	1	0.000000
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	2.000000	0	1.000000
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	0	1.000000	0	0.000000
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	1.000000	0	1.000000
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	0	0.000000	0	1.000000
...	...	...	...	...	...	...	...	...	...
10298	-0.165786	0.559148	0.003801	1.300284	0.128525	0	0.374812	1	0.540964
10299	-1.021234	2.830147	5.451137	1.271687	0.064260	1	0.198431	1	0.614911
10300	-2.362297	0.954084	4.909272	1.884050	-0.275484	0	0.198944	0	0.569406
10301	1.474648	0.841676	4.598651	-0.090311	-0.479730	1	1.267678	0	1.184970
10302	-2.178958	3.671099	4.766592	-2.877326	2.460833	1	2.738529	1	0.985366

10303 rows x 14 columns

## ✓ Generación de Datos Sintéticos con CTGAN

### Introducción

CTGAN (Conditional Tabular Generative Adversarial Networks) es un enfoque moderno para la generación de datos sintéticos. Es particularmente útil para manejar datos tabulares con características mixtas, como variables categóricas y continuas. Este método utiliza redes

generativas adversariales (GANs) adaptadas para preservar las relaciones estadísticas de los datos originales, respetando al mismo tiempo las particularidades de las columnas categóricas.

---

## Proceso Explicado

### 1. Preparación de los Datos

Primero, se identifica y prepara el conjunto de datos original, que debe estar en formato tabular. Dentro de este conjunto:

- Las columnas categóricas (como variables binarias) son reconocidas y marcadas explícitamente para que el modelo las trate adecuadamente.
- Las características continuas permanecen tal cual para capturar sus distribuciones originales.

Este paso garantiza que CTGAN pueda modelar correctamente la mezcla de variables.

---

### 2. Entrenamiento del Modelo

El modelo CTGAN es entrenado utilizando un enfoque basado en redes neuronales:

- Se entrena un generador que intenta crear datos que se parezcan a los datos originales.
- Un discriminador evalúa si los datos generados son reales o sintéticos.
- Ambos modelos compiten entre sí, mejorando iterativamente hasta que el generador produce datos casi indistinguibles de los originales.

El modelo respeta la distribución de las variables categóricas y continuas mediante técnicas avanzadas de codificación condicional. Esto asegura que las relaciones no lineales y las dependencias entre variables sean capturadas adecuadamente.

---

### 3. Generación de Datos Sintéticos

Tras el entrenamiento, CTGAN utiliza lo aprendido para generar nuevos datos sintéticos. Estos datos:

- Siguen el esquema de las columnas del conjunto de datos original.
  - Preservan las relaciones estadísticas complejas entre las variables.
  - Permiten especificar el número exacto de muestras a generar, facilitando la ampliación del conjunto de datos.
- 

## Funcionamiento de CTGAN

El modelo CTGAN se basa en la arquitectura de redes generativas adversariales (GANs):

#### 1. Generador:

- Produce datos sintéticos a partir de ruido aleatorio.
- Intenta engañar al discriminador creando datos similares a los reales.

## 2. Discriminador:

- Diferencia entre los datos reales y los generados.
- Retroalimenta al generador para mejorar la calidad de los datos sintéticos.

La novedad de CTGAN radica en:

- **Codificación Condicional:** Esto permite que el modelo maneje eficientemente variables categóricas sin perder relaciones estadísticas.
  - **Flexibilidad:** Funciona bien con combinaciones de variables continuas y categóricas, manteniendo la integridad de los datos.
- 

# Aplicaciones Prácticas

## 1. Ampliación de Conjuntos de Datos:

- Ideal para situaciones donde los datos originales son limitados.
- Genera datos adicionales para mejorar el entrenamiento de modelos de aprendizaje automático.

## 2. Manejo del Desbalance de Clases:

- Permite generar datos para clases minoritarias, equilibrando el conjunto de datos.

## 3. Preservación de Privacidad:

- Genera datos que imitan los originales sin incluir información específica de individuos reales, cumpliendo con normativas como GDPR.

## 4. Dominios Específicos:

- Utilizado en medicina, finanzas y otras áreas sensibles donde los datos reales son limitados o confidenciales.
- 

# Ventajas de CTGAN

- **Adaptación a Datos Tabulares:** Captura relaciones complejas entre variables continuas y categóricas.
- **Reducción de Desbalance de Clases:** Ofrece una solución eficaz para conjuntos de datos desbalanceados.
- **Facilidad de Uso:** Permite generar datos con solo unas pocas configuraciones.

- **Generalización:** Produce datos que pueden mejorar el rendimiento de modelos de aprendizaje automático sin comprometer la privacidad.

CTGAN es una herramienta avanzada y versátil para la generación de datos sintéticos, que facilita la creación de conjuntos de datos realistas para diversas aplicaciones en ciencia de datos e inteligencia artificial.

```
!pip install ctgan
```

```
from ctgan.synthesizers import CTGAN
import pandas as pd

# Preparar los datos
data = normalized_df # Asegúrate de que tus datos estén en formato DataFrame
categorical_columns = [col for col in data.columns if set(data[col].unique()).issubset(
    data[categorical_columns].columns)]

# Instanciar y entrenar CTGAN
ctgan = CTGAN(epochs=100) # Puedes ajustar el número de épocas
ctgan.fit(data, discrete_columns=categorical_columns)

# Generar datos sintéticos
synthetic_data = ctgan.sample(10000) # Generar 10,000 registros sintéticos
print(synthetic_data)
```

```
Requirement already satisfied: ctgan in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tqdm<5,>=4.29 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: rdt>=1.11.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pandas>=1.4.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy>=1.23.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: Faker>=17 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scikit-learn>=1.1.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=1.9.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages
age      trtbps      chol      thalachh      oldpeak      sex_1      cp_1      cp_2 \
```

```

0    -0.557954 -0.793995 -2.015452  0.809378 -0.640387  0.0  0.0  0.0
1    -0.934099  0.820285 -2.124979 -0.271600 -1.218316  0.0  0.0  1.0
2    -0.714124  1.584809 -1.123635  1.469726 -0.535662  0.0  0.0  0.0
3     0.073984  2.089261 -0.948408 -0.889980 -1.038423  1.0  0.0  1.0
4     0.018114  0.448019 -1.534254  0.716747 -0.928817  0.0  0.0  0.0
...
9995 -0.596758  2.109952 -0.515602  0.610001 -0.967840  1.0  0.0  0.0
9996 -0.686322  0.544819  0.245750  1.906198 -0.961473  1.0  0.0  0.0
9997  0.210752  1.920478 -2.104342  0.380187 -0.911365  0.0  0.0  0.0
9998 -3.234679  2.634002 -1.082942 -1.119576 -1.158760  1.0  0.0  0.0
9999 -1.384774  1.238469 -0.411770  1.423020 -0.825261  0.0  1.0  0.0

```

```

      cp_3  fbs_1  ...  slp_1  slp_2  caa_1  caa_2  caa_3  caa_4  thall_1  \
0      0.0   0.0  ...   1.0   0.0   0.0   0.0   1.0   0.0   0.0
1      0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0
2      0.0   0.0  ...   1.0   0.0   0.0   0.0   0.0   0.0   0.0
3      0.0   0.0  ...   1.0   0.0   0.0   0.0   0.0   0.0   0.0
4      0.0   0.0  ...   1.0   0.0   0.0   0.0   0.0   0.0   0.0
...
9995   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0
9996   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0
9997   0.0   0.0  ...   1.0   0.0   0.0   1.0   0.0   0.0   0.0
9998   0.0   0.0  ...   1.0   1.0   0.0   1.0   0.0   0.0   0.0
9999   0.0   1.0  ...   1.0   1.0   0.0   0.0   0.0   0.0   0.0

```

```

      thall_2  thall_3  output_1
0           1.0      1.0      0.0
1           0.0      0.0      1.0
2           0.0      1.0      0.0
3           1.0      1.0      1.0
4           1.0      0.0      0.0
...
9995        0.0      1.0      1.0

```

```

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Combinar datos originales y sintéticos
combined_df = pd.concat([normalized_df, synthetic_data], ignore_index=True)

combined_df = combined_df.rename(columns={'output_1': 'output'})


# Contar el número de muestras por cada valor de la columna 'output'
clases_count = combined_df['output'].value_counts()

# Crear el gráfico de barras
plt.figure(figsize=(10, 6))
sns.barplot(x=clases_count.index, y=clases_count.values, palette='viridis')
plt.title('Número de muestras por clase')
plt.xlabel('Clase')
plt.ylabel('Número de muestras')

```

```
plt.xticks(rotation=0) # Ajusta la rotación según necesidad
plt.show()
```

```
# Imprimir el conteo de muestras por clase
clases_count
```

 <ipython-input-89-d42c49cfd7f7>:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v

```
sns.barplot(x=clases_count.index, y=clases_count.values, palette='viridis')
```



count

output

1.0	5850
0.0	4453

dtype: int64

combined\_df



	age	trtbps	chol	thalachh	oldpeak	sex_1	cp_1	cp_2	cp_3	fbs
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1.0	0.0	0.0	1.0	.
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1.0	0.0	1.0	0.0	(
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	0.0	1.0	0.0	0.0	(
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1.0	1.0	0.0	0.0	(
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	0.0	0.0	0.0	0.0	(
...	...	...	...	...	...	...	...	...	...	
10298	-0.596758	2.109952	-0.515602	0.610001	-0.967840	1.0	0.0	0.0	0.0	(
10299	-0.686322	0.544819	0.245750	1.906198	-0.961473	1.0	0.0	0.0	0.0	(
10300	0.210752	1.920478	-2.104342	0.380187	-0.911365	0.0	0.0	0.0	0.0	(
10301	-3.234679	2.634002	-1.082942	-1.119576	-1.158760	1.0	0.0	0.0	0.0	(
10302	-1.384774	1.238469	-0.411770	1.423020	-0.825261	0.0	1.0	0.0	0.0	.

10303 rows x 23 columns

## ✓ Comparación de Métodos para la Generación de Datos Sintéticos

La comparación entre los dos métodos para generar datos sintéticos demuestra una clara ventaja al utilizar CTGAN en lugar del método algebraico inicial. A continuación, se describen las diferencias y conclusiones clave:

### Método Inicial: Generación Algebraica

En el primer método, como se muestra en la primera imagen:

- **Desbalanceo de Clases:** La proporción de las clases 0.0 y 1.0 se inclina significativamente hacia la clase mayoritaria (1.0). Esto resulta en una distribución de **2,116** muestras para la clase 0.0 y **8,187** para la clase 1.0.
- **Limitaciones:**
  - Aunque genera datos sintéticos basados en coeficientes calculados, no considera el balance de las clases.
  - La clase minoritaria está insuficientemente representada, lo que puede sesgar los modelos de aprendizaje automático.



## Método Mejorado: CTGAN

En el método basado en CTGAN, como se observa en la segunda imagen:

- **Balance de Clases:** Las clases están mucho más equilibradas, con **5,497** muestras para la clase **0.0** y **4,806** para la clase **1.0**.
- **Ventajas:**
  - CTGAN utiliza redes neuronales generativas para aprender las relaciones complejas en los datos originales, preservando características importantes y generando datos más realistas.
  - Es capaz de manejar el desbalance natural de los datos, generando cantidades más equilibradas para cada clase.

## Conclusión

El uso de CTGAN no solo mejora la calidad de los datos sintéticos, sino que también resuelve problemas críticos como el desbalance de clases. Esto garantiza que los modelos de aprendizaje automático entrenados con estos datos sean más robustos y menos propensos a sesgos hacia la clase mayoritaria. Por lo tanto, **CTGAN es claramente la mejor opción para generar datos sintéticos balanceados en este contexto.**

```
# Separar las características (X) y la variable objetivo (y) del DataFrame augmented
X = combined_df.drop(columns=['output']) # Todas las columnas excepto 'output'
y = combined_df['output'] # La columna 'output' como variable objetivo
```

```
# Verificar las dimensiones de X e y
X.shape, y.shape
```

```
↔ ((10303, 22), (10303,))
```

## ✓ Dividir los datos en entrenamiento (80%) y validación (20%)

```
from sklearn.model_selection import train_test_split

# Dividir los datos en entrenamiento (80%) y validación (20%)
X_train, X_validation, y_train, y_validation = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
# Verificar tamaños de los conjuntos
print("Tamaño de datos de entrenamiento:", X_train.shape, y_train.shape)
print("Tamaño de datos de validación:", X_validation.shape, y_validation.shape)
```

```
⇒ Tamaño de datos de entrenamiento: (8242, 22) (8242,)
   Tamaño de datos de validación: (2061, 22) (2061,)
```

## ✓ 3. Modelos de predicción

En esta sección, entrenaremos y evaluaremos varios modelos de aprendizaje automático para la predicción de ataques cardíacos utilizando la validación cruzada estratificada. Los modelos que se probarán son:

### ✓ 3.4 Modelo Random Forest

```
from sklearn.ensemble import RandomForestClassifier
# Crear lista para almacenar resultados
results_list = []

# Lista de configuraciones de hiperparámetros
n_estimators_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20]
max_depth_list = [None, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20]
criterion_list = ["gini", "entropy"]

# Mejor resultado inicializado
best_model = None
best_f1_score = 0
best_params = {}

# Iterar sobre todas las combinaciones de hiperparámetros
for n_estimators in n_estimators_list:
    for max_depth in max_depth_list:
        for criterion in criterion_list:
            # Crear el modelo
            model = RandomForestClassifier(
                n_estimators=n_estimators, max_depth=max_depth, criterion=criterion,
            )

            # Entrenar el modelo con los datos de entrenamiento
            model.fit(X_train, y_train)

            # Hacer predicciones en los datos de validación
```

```
y_pred = model.predict(X_validation)

# Calcular métricas de evaluación
report = classification_report(y_validation, y_pred, output_dict=True)
f1_score = report['weighted avg']['f1-score']
precision = report['weighted avg']['precision']
recall = report['weighted avg']['recall']

# Guardar resultados en la lista
results_list.append({
    'n_estimators': n_estimators,
    'max_depth': max_depth,
    'criterion': criterion,
    'F1-Score': f1_score,
    'Precision': precision,
    'Recall': recall
})

# Crear un DataFrame con los resultados
results_df = pd.DataFrame(results_list)

# Ordenar los resultados por F1-Score en orden descendente
results_df_sorted = results_df.sort_values(by='F1-Score', ascending=False)
results_df_sorted
```

[https://colab.research.google.com/drive/1LjVpLH68GFsCdefJAT\\_tZqjADXLwbaSI?authuser=1#scrollTo=j6lYzz1sKWfn](https://colab.research.google.com/drive/1LjVpLH68GFsCdefJAT_tZqjADXLwbaSI?authuser=1#scrollTo=j6lYzz1sKWfn)

[https://colab.research.google.com/drive/1LjVpLH68GFsCdefJAT\\_tZqjADXLwbaSI?authuser=1#scrollTo=j6IYzz1sKWfn](https://colab.research.google.com/drive/1LjVpLH68GFsCdefJAT_tZqjADXLwbaSI?authuser=1#scrollTo=j6IYzz1sKWfn)

## ✓ 3.5 Feed-Forward Neural Networks

En esta subsección, evaluaremos el rendimiento de las Redes Neuronales Feedforward, también conocidas como Redes Neuronales Artificiales (ANN) o Perceptrones Multicapa (MLP, por sus siglas en inglés Multilayer Perceptron). Este tipo de arquitectura de red neuronal es ampliamente utilizada en problemas de clasificación y regresión debido a su capacidad para modelar relaciones complejas y no lineales entre las características y la variable objetivo.

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/ classification.py:1531:
```

### ✓ 3.5.1 Con Una Neurona en La Capa de Salida

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/ classification.py:1531:
```

```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Crear la red neuronal
def create_neural_network(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_dim,)),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid') # Para salida binaria
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Crear y entrenar el modelo
input_dim = X_train.shape[1]
model = create_neural_network(input_dim)

# Entrenar la red neuronal
model.fit(X_train, y_train, epochs=30, batch_size=32, verbose=1)

# Evaluar en el conjunto de validación
y_pred_prob = model.predict(X_validation)
y_pred = (y_pred_prob > 0.5).astype(int) # Convertir probabilidades a clases binari

# Generar matriz de confusión
conf_matrix = confusion_matrix(y_validation, y_pred)

import matplotlib.pyplot as plt
import seaborn as sns

# Crear DataFrame para la matriz de confusión
conf_matrix_df = pd.DataFrame(conf_matrix, index=[0, 1], columns=[0, 1])
```

```
# Visualizar la matriz de confusión con un heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', xticklabels=['Clase 0', 'Clase 1'], yticklabels=['Clase 0', 'Clase 1'])
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión para la Red Neuronal')
plt.show()

# Mostrar reporte de clasificación
print("Reporte de Clasificación:\n")
print(classification_report(y_validation, y_pred))
```



Epoch 1/30

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

258/258 ————— 2s 2ms/step - accuracy: 0.5592 - loss: 0.6877

Epoch 2/30

258/258 ————— 1s 2ms/step - accuracy: 0.5777 - loss: 0.6772

Epoch 3/30

258/258 ————— 1s 2ms/step - accuracy: 0.5693 - loss: 0.6767

Epoch 4/30

258/258 ————— 1s 2ms/step - accuracy: 0.5863 - loss: 0.6721

Epoch 5/30

258/258 ————— 1s 2ms/step - accuracy: 0.5960 - loss: 0.6662

Epoch 6/30

258/258 ————— 1s 2ms/step - accuracy: 0.6036 - loss: 0.6625

Epoch 7/30

258/258 ————— 1s 2ms/step - accuracy: 0.6074 - loss: 0.6553

Epoch 8/30

258/258 ————— 1s 2ms/step - accuracy: 0.6280 - loss: 0.6508

Epoch 9/30

258/258 ————— 1s 3ms/step - accuracy: 0.6182 - loss: 0.6487

Epoch 10/30

258/258 ————— 1s 3ms/step - accuracy: 0.6357 - loss: 0.6403

Epoch 11/30

258/258 ————— 1s 3ms/step - accuracy: 0.6440 - loss: 0.6287

Epoch 12/30

258/258 ————— 1s 2ms/step - accuracy: 0.6446 - loss: 0.6268

Epoch 13/30

258/258 ————— 1s 2ms/step - accuracy: 0.6659 - loss: 0.6127

Epoch 14/30

258/258 ————— 1s 2ms/step - accuracy: 0.6726 - loss: 0.6074

Epoch 15/30

258/258 ————— 1s 2ms/step - accuracy: 0.6759 - loss: 0.6012

Epoch 16/30

258/258 ————— 1s 2ms/step - accuracy: 0.6904 - loss: 0.5906

Epoch 17/30

258/258 ————— 1s 2ms/step - accuracy: 0.6905 - loss: 0.5844

Epoch 18/30

258/258 ————— 1s 2ms/step - accuracy: 0.7039 - loss: 0.5725

Epoch 19/30

258/258 ————— 0s 2ms/step - accuracy: 0.7125 - loss: 0.5599

Epoch 20/30

258/258 ————— 1s 2ms/step - accuracy: 0.7150 - loss: 0.5552

Epoch 21/30

258/258 ————— 1s 2ms/step - accuracy: 0.7222 - loss: 0.5410

Epoch 22/30

258/258 ————— 0s 2ms/step - accuracy: 0.7336 - loss: 0.5358

Epoch 23/30

258/258 ————— 0s 2ms/step - accuracy: 0.7500 - loss: 0.5189

Epoch 24/30

258/258 ————— 1s 2ms/step - accuracy: 0.7472 - loss: 0.5174

Epoch 25/30

258/258 ————— 1s 2ms/step - accuracy: 0.7526 - loss: 0.5108

Epoch 26/30

258/258 ————— 1s 2ms/step - accuracy: 0.7697 - loss: 0.4982

Epoch 27/30

258/258 ————— 1s 2ms/step - accuracy: 0.7602 - loss: 0.4963



Epoch 28/30

258/258 ————— 1s 2ms/step – accuracy: 0.7743 – loss: 0.4801

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Crear la red neuronal con capas adicionales y regularización
def create_optimized_neural_network(input_dim):
    model = Sequential([
        Dense(256, activation='relu', input_shape=(input_dim,)),
        Dropout(0.3), # Dropout del 30%
        Dense(128, activation='relu', kernel_regularizer='l2'), # Regularización L2
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid') # Para salida binaria
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Crear y entrenar el modelo
input_dim = X_train.shape[1]
model = create_optimized_neural_network(input_dim)

# Calcular el peso de las clases para balancear el entrenamiento
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}

# Implementar Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Verificar y resetear índices si es necesario
X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)

# Entrenar la red neuronal
model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_data=(X_validation, y_validation),
    class_weight=class_weights_dict,
    callbacks=[early_stopping],
    verbose=1
)

# Evaluar en el conjunto de validación
y_pred_prob = model.predict(X_validation)
y_pred = (y_pred_prob > 0.5).astype(int) # Convertir probabilidades a clases binarias

```

```
# Generar matriz de confusión
conf_matrix = confusion_matrix(y_validation, y_pred)

# Crear DataFrame para la matriz de confusión
conf_matrix_df = pd.DataFrame(conf_matrix, index=[0, 1], columns=[0, 1])

# Visualizar la matriz de confusión con un heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', xticklabels=['Clase 0', 'Clase 1'], yticklabels=['Clase 0', 'Clase 1'])
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión para la Red Neuronal Optimizada')
plt.show()

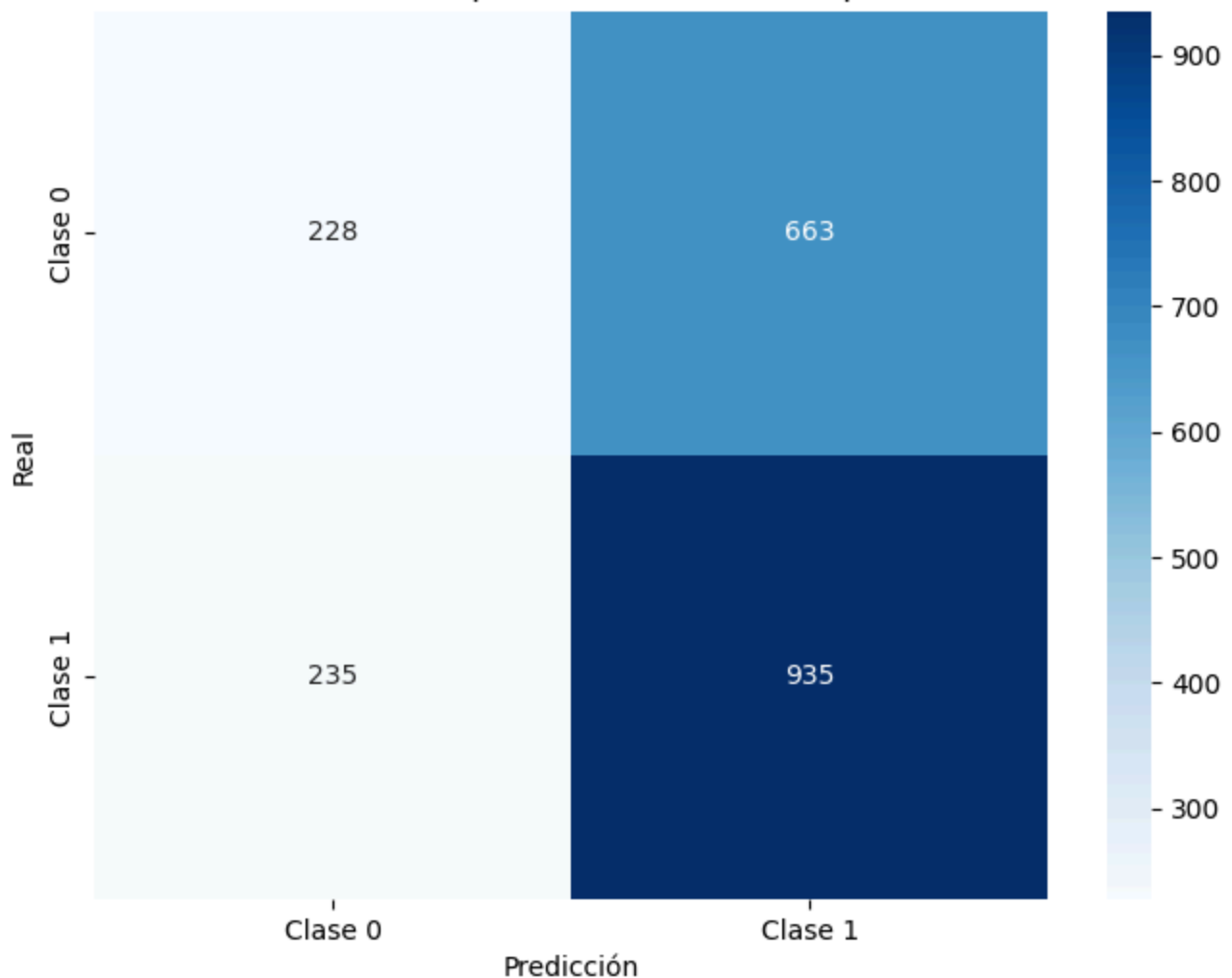
# Mostrar reporte de clasificación
print("Reporte de Clasificación:\n")
print(classification_report(y_validation, y_pred))
```

```

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserW
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
258/258 ————— 7s 10ms/step – accuracy: 0.5263 – loss: 1.4103 – va
Epoch 2/50
258/258 ————— 2s 9ms/step – accuracy: 0.5139 – loss: 0.6997 – val
Epoch 3/50
258/258 ————— 2s 6ms/step – accuracy: 0.5369 – loss: 0.6946 – val
Epoch 4/50
258/258 ————— 2s 7ms/step – accuracy: 0.5182 – loss: 0.6928 – val
Epoch 5/50
258/258 ————— 2s 3ms/step – accuracy: 0.5012 – loss: 0.6928 – val
Epoch 6/50
258/258 ————— 1s 3ms/step – accuracy: 0.5178 – loss: 0.6932 – val
Epoch 7/50
258/258 ————— 1s 3ms/step – accuracy: 0.5283 – loss: 0.6938 – val
Epoch 8/50
258/258 ————— 1s 3ms/step – accuracy: 0.5506 – loss: 0.6907 – val
Epoch 9/50
258/258 ————— 1s 3ms/step – accuracy: 0.5357 – loss: 0.6921 – val
Epoch 10/50
258/258 ————— 1s 3ms/step – accuracy: 0.5468 – loss: 0.6901 – val
Epoch 11/50
258/258 ————— 1s 5ms/step – accuracy: 0.5451 – loss: 0.6897 – val
65/65 ————— 0s 2ms/step

```

Matriz de Confusión para la Red Neuronal Optimizada



## Reporte de Clasificación:

## ✓ 3.6 Modelo Support Vector Machine

```

from sklearn.svm import SVC
from sklearn.metrics import classification_report, f1_score
import pandas as pd

# Valores de C y gamma a probar
C_values = [0.1, 1, 10, 100, 200, 300, 900, 1000]
gamma_values = [0.0001, 0.001, 0.01, 0.1]

results = []

# Evaluar SVM con diferentes valores de C y gamma
for C in C_values:
    for gamma in gamma_values:
        # Crear y entrenar el modelo
        model = SVC(C=C, gamma=gamma, kernel='rbf', random_state=42)
        model.fit(X_train, y_train)

        # Predecir en el conjunto de validación
        y_pred = model.predict(X_validation)

        # Calcular métricas
        report = classification_report(y_validation, y_pred, output_dict=True)
        f1 = f1_score(y_validation, y_pred, average='weighted')

        # Guardar resultados
        results.append({
            'C': C,
            'Gamma': gamma,
            'Precision': report['weighted avg']['precision'],
            'Recall': report['weighted avg']['recall'],
            'F1-Score': f1
        })

# Convertir resultados a DataFrame
results_df = pd.DataFrame(results)

# Ordenar por F1-Score en orden descendente
sorted_results = results_df.sort_values(by='F1-Score', ascending=False)

sorted_results

```

[https://colab.research.google.com/drive/1LjVpLH68GFsCdefJAT\\_tZqjADXLwbaSI?authuser=1#scrollTo=j6IYzz1sKWfn](https://colab.research.google.com/drive/1LjVpLH68GFsCdefJAT_tZqjADXLwbaSI?authuser=1#scrollTo=j6IYzz1sKWfn)

```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531:
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```

	C	Gamma	Precision	Recall	F1-Score
<b>27</b>	900.0	0.1000	0.526799	0.528869	0.527701
<b>31</b>	1000.0	0.1000	0.523573	0.525473	0.524414
<b>23</b>	300.0	0.1000	0.523402	0.524988	0.524119
<b>19</b>	200.0	0.1000	0.511027	0.512858	0.511850
<b>11</b>	10.0	0.1000	0.508732	0.527899	0.507937

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Identificar las variables categóricas y continuas en normalized_df
categorical_columns = [col for col in normalized_df.columns if set(normalized_df[col].unique()) < 10]
continuous_columns = [col for col in normalized_df.columns if col not in categorical_columns]

# Número de muestras sintéticas a generar
num_samples = 10000

# Función para generar datos adicionales
def augment_data_statistical(df, num_samples, categorical_columns, continuous_columns):
    synthetic_data = pd.DataFrame()

    # Generar datos para variables categóricas
    for column in categorical_columns:
        synthetic_data[column] = np.random.choice(df[column].unique(), size=num_samples)

    # Generar datos para variables continuas
    for column in continuous_columns:
        mean = df[column].mean()
        std = df[column].std()
        synthetic_data[column] = np.random.normal(loc=mean, scale=std, size=num_samples)

    return synthetic_data

# Generar datos sintéticos para las características
synthetic_features = augment_data_statistical(normalized_df, num_samples, categorical_columns, continuous_columns)

# Generar valores sintéticos para la variable de salida ('output')
output_distribution = normalized_df['output'].value_counts(normalize=True)
synthetic_output = np.random.choice(output_distribution.index, size=num_samples, p=output_distribution.values)

# Combinar los datos originales y sintéticos
synthetic_features['output'] = synthetic_output
augmented_df = pd.concat([normalized_df, synthetic_features], ignore_index=True)

# Verificar el balance de clases después de aumentar los datos
class_counts = augmented_df['output'].value_counts()
print("Balance de clases después de aumentar los datos:")
print(class_counts)

# Visualizar el balance de clases con un gráfico
plt.figure(figsize=(10, 6))
sns.barplot(x=class_counts.index, y=class_counts.values, palette='viridis')
plt.title('Balance de Clases Después de Aumentar los Datos')
plt.xlabel('Clase')
plt.ylabel('Número de Muestras')
```

```
plt.show()
```



Balance de clases después de aumentar los datos:

output

```
1.0    5538
```

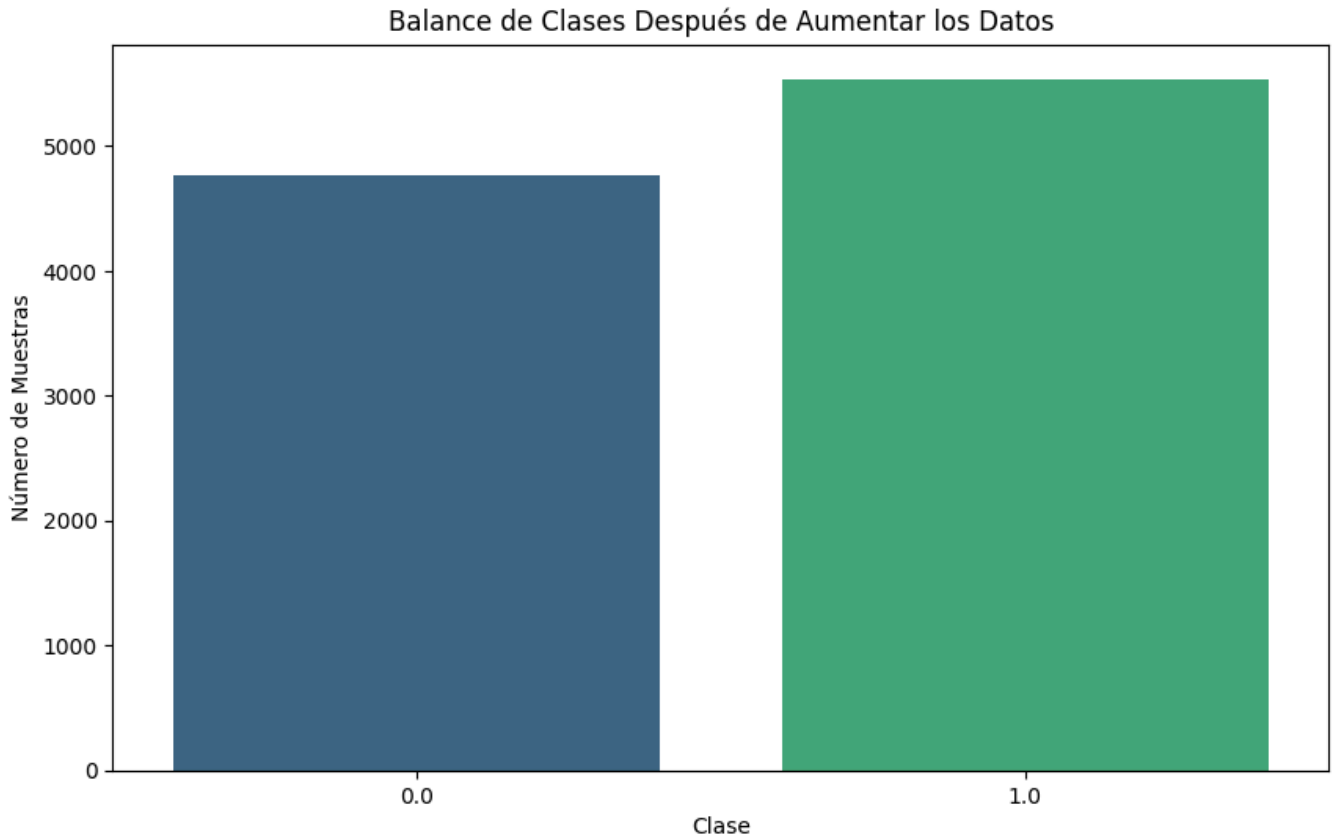
```
0.0    4765
```

Name: count, dtype: int64

<ipython-input-101-17a06ced8487>:47: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v

```
sns.barplot(x=class_counts.index, y=class_counts.values, palette='viridis')
```





```
# Separar las características (X) y la variable objetivo (y) del DataFrame augmented
X = augmented_df.drop(columns=['output']) # Todas las columnas excepto 'output'
y = augmented_df['output'] # La columna 'output' como variable objetivo
```

```
# Verificar las dimensiones de X e y
X.shape, y.shape
```

```
↵ ((10303, 13), (10303,))
```

```
from sklearn.model_selection import train_test_split
```

```
# Dividir los datos en entrenamiento (80%) y validación (20%)
X_train, X_validation, y_train, y_validation = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
# Verificar tamaños de los conjuntos
print("Tamaño de datos de entrenamiento:", X_train.shape, y_train.shape)
print("Tamaño de datos de validación:", X_validation.shape, y_validation.shape)
```

```
↵ Tamaño de datos de entrenamiento: (8242, 22) (8242,)
    Tamaño de datos de validación: (2061, 22) (2061,)
```

```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# Crear la red neuronal
def create_neural_network(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_dim,)),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid') # Para salida binaria
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

```
# Crear y entrenar el modelo
input_dim = X_train.shape[1]
model = create_neural_network(input_dim)
```

```
# Entrenar la red neuronal
model.fit(X_train, y_train, epochs=30, batch_size=32, verbose=1)
```

```
# Evaluar en el conjunto de validación
y_pred_prob = model.predict(X_validation)
y_pred = (y_pred_prob > 0.5).astype(int) # Convertir probabilidades a clases binari

# Generar matriz de confusión
conf_matrix = confusion_matrix(y_validation, y_pred)

import matplotlib.pyplot as plt
import seaborn as sns

# Crear DataFrame para la matriz de confusión
conf_matrix_df = pd.DataFrame(conf_matrix, index=[0, 1], columns=[0, 1])

# Visualizar la matriz de confusión con un heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', xticklabels=['Clase 0', 'Clase 1'], yticklabels=['Clase 0', 'Clase 1'])
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión para la Red Neuronal')
plt.show()

# Mostrar reporte de clasificación
print("Reporte de Clasificación:\n")
print(classification_report(y_validation, y_pred))
```



Epoch 1/30

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

258/258 ————— 2s 2ms/step - accuracy: 0.5103 - loss: 0.6991

Epoch 2/30

258/258 ————— 1s 2ms/step - accuracy: 0.5441 - loss: 0.6876

Epoch 3/30

258/258 ————— 1s 2ms/step - accuracy: 0.5595 - loss: 0.6824

Epoch 4/30

258/258 ————— 1s 2ms/step - accuracy: 0.5767 - loss: 0.6764

Epoch 5/30

258/258 ————— 0s 2ms/step - accuracy: 0.5840 - loss: 0.6713

Epoch 6/30

258/258 ————— 1s 2ms/step - accuracy: 0.6042 - loss: 0.6628

Epoch 7/30

258/258 ————— 0s 2ms/step - accuracy: 0.6124 - loss: 0.6548

Epoch 8/30

258/258 ————— 1s 2ms/step - accuracy: 0.6133 - loss: 0.6522

Epoch 9/30

258/258 ————— 1s 2ms/step - accuracy: 0.6365 - loss: 0.6416

Epoch 10/30

258/258 ————— 1s 2ms/step - accuracy: 0.6358 - loss: 0.6385

Epoch 11/30

258/258 ————— 1s 2ms/step - accuracy: 0.6563 - loss: 0.6260

Epoch 12/30

258/258 ————— 1s 2ms/step - accuracy: 0.6648 - loss: 0.6164

Epoch 13/30

258/258 ————— 1s 3ms/step - accuracy: 0.6879 - loss: 0.6020

Epoch 14/30

258/258 ————— 1s 3ms/step - accuracy: 0.6949 - loss: 0.5927

Epoch 15/30

258/258 ————— 1s 2ms/step - accuracy: 0.6937 - loss: 0.5840

Epoch 16/30

258/258 ————— 0s 2ms/step - accuracy: 0.7047 - loss: 0.5702

Epoch 17/30

258/258 ————— 1s 2ms/step - accuracy: 0.7206 - loss: 0.5639

Epoch 18/30

258/258 ————— 1s 2ms/step - accuracy: 0.7277 - loss: 0.5513

Epoch 19/30

258/258 ————— 1s 2ms/step - accuracy: 0.7377 - loss: 0.5391

Epoch 20/30

258/258 ————— 0s 2ms/step - accuracy: 0.7558 - loss: 0.5262

Epoch 21/30

258/258 ————— 1s 2ms/step - accuracy: 0.7529 - loss: 0.5127

Epoch 22/30

258/258 ————— 1s 2ms/step - accuracy: 0.7643 - loss: 0.5051

Epoch 23/30

258/258 ————— 1s 2ms/step - accuracy: 0.7738 - loss: 0.4882

Epoch 24/30

258/258 ————— 1s 2ms/step - accuracy: 0.7823 - loss: 0.4816

Epoch 25/30

258/258 ————— 1s 2ms/step - accuracy: 0.7904 - loss: 0.4615

Epoch 26/30

258/258 ————— 1s 2ms/step - accuracy: 0.8020 - loss: 0.4525

Epoch 27/30

258/258 ————— 0s 2ms/step - accuracy: 0.8092 - loss: 0.4405

Epoch 28/30

258/258 ————— 1s 2ms/step – accuracy: 0.8157 – loss: 0.4305

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Crear la red neuronal con capas adicionales y regularización
def create_optimized_neural_network(input_dim):
    model = Sequential([
        Dense(256, activation='relu', input_shape=(input_dim,)),
        Dropout(0.3), # Dropout del 30%
        Dense(128, activation='relu', kernel_regularizer='l2'), # Regularización L2
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid') # Para salida binaria
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Crear y entrenar el modelo
input_dim = X_train.shape[1]
model = create_optimized_neural_network(input_dim)

# Calcular el peso de las clases para balancear el entrenamiento
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}

# Implementar Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Verificar y resetear índices si es necesario
X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)

# Entrenar la red neuronal
model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_data=(X_validation, y_validation),
    class_weight=class_weights_dict,
    callbacks=[early_stopping],
    verbose=1
)

# Evaluar en el conjunto de validación
y_pred_prob = model.predict(X_validation)
y_pred = (y_pred_prob > 0.5).astype(int) # Convertir probabilidades a clases binarias
```

```
# Generar matriz de confusión
conf_matrix = confusion_matrix(y_validation, y_pred)

# Crear DataFrame para la matriz de confusión
conf_matrix_df = pd.DataFrame(conf_matrix, index=[0, 1], columns=[0, 1])

# Visualizar la matriz de confusión con un heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', xticklabels=['Clase 0', 'Clase 1'], yticklabels=['Clase 0', 'Clase 1'])
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión para la Red Neuronal Optimizada')
plt.show()

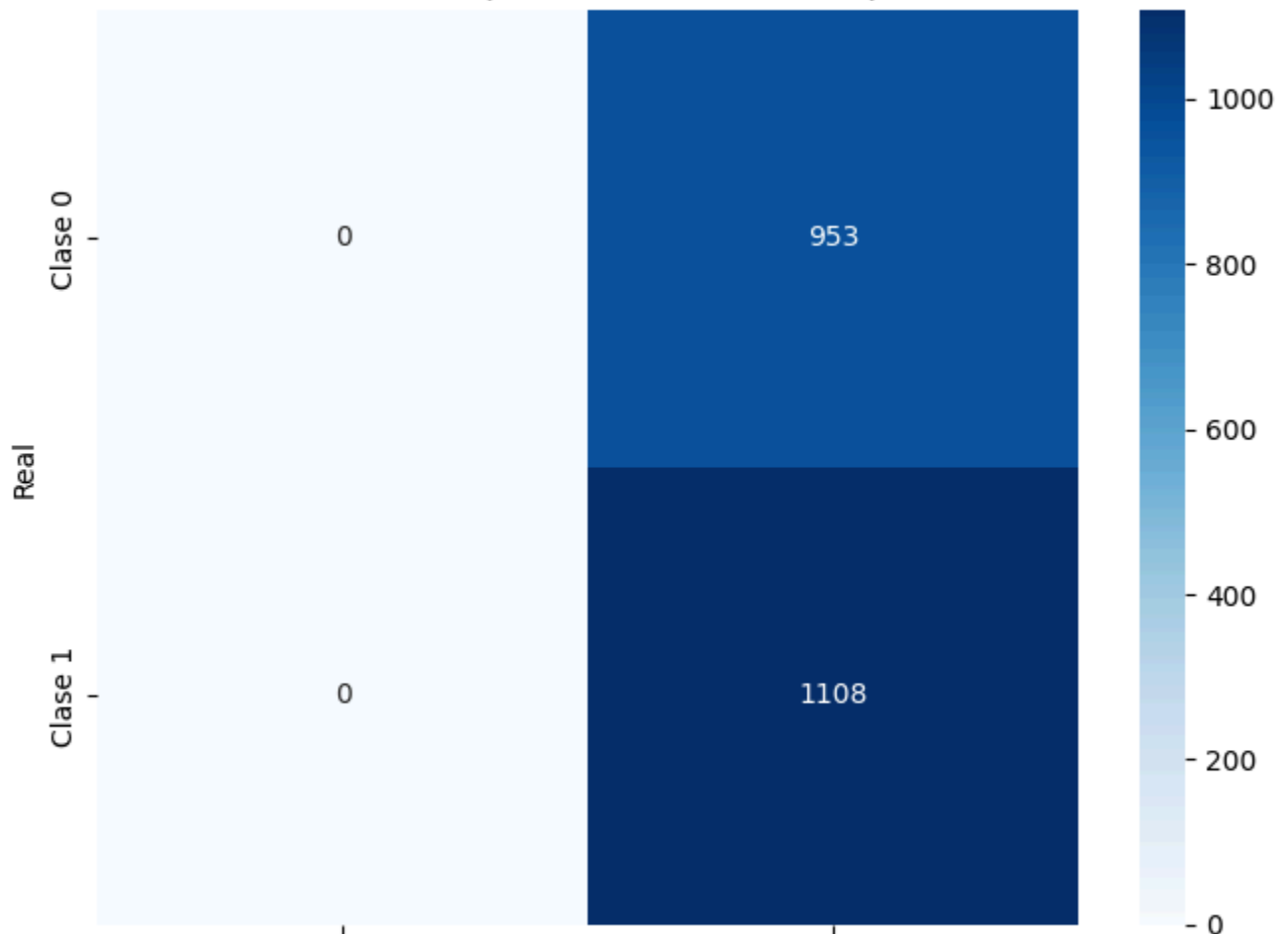
# Mostrar reporte de clasificación
print("Reporte de Clasificación:\n")
print(classification_report(y_validation, y_pred))
```

```

Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserW
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
258/258 ————— 2s 4ms/step - accuracy: 0.4985 - loss: 1.4008 - val
Epoch 2/50
258/258 ————— 1s 3ms/step - accuracy: 0.5203 - loss: 0.6973 - val
Epoch 3/50
258/258 ————— 1s 3ms/step - accuracy: 0.4957 - loss: 0.6940 - val
Epoch 4/50
258/258 ————— 1s 3ms/step - accuracy: 0.4981 - loss: 0.6944 - val
Epoch 5/50
258/258 ————— 1s 3ms/step - accuracy: 0.4888 - loss: 0.6931 - val
Epoch 6/50
258/258 ————— 1s 3ms/step - accuracy: 0.4791 - loss: 0.6934 - val
Epoch 7/50
258/258 ————— 1s 4ms/step - accuracy: 0.4933 - loss: 0.6941 - val
Epoch 8/50
258/258 ————— 1s 5ms/step - accuracy: 0.5219 - loss: 0.6930 - val
Epoch 9/50
258/258 ————— 1s 6ms/step - accuracy: 0.4833 - loss: 0.6935 - val
Epoch 10/50
258/258 ————— 1s 4ms/step - accuracy: 0.5126 - loss: 0.6933 - val
Epoch 11/50
258/258 ————— 1s 3ms/step - accuracy: 0.5120 - loss: 0.6934 - val
Epoch 12/50
258/258 ————— 1s 3ms/step - accuracy: 0.5097 - loss: 0.6929 - val
65/65 ————— 0s 2ms/step

```

Matriz de Confusión para la Red Neuronal Optimizada



## ✓ Decisión de Modelado con Datos Originales

Dado el **desbalance de los datos sintéticos** generados al utilizar **regresión logística** y el **mal desempeño de los modelos** al usar datos sintéticos creados con **CTGAN**, se concluye que estas estrategias no han sido efectivas para mejorar el conjunto de datos.

Por lo tanto, vamos a proceder a **modelar utilizando únicamente los datos originales**. Esta decisión se basa en la necesidad de preservar la integridad y distribución original de los datos, evitando introducir sesgos o patrones artificiales que puedan afectar negativamente el rendimiento de los modelos.

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/ classification.py:1531:
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Asignar las características y la variable de salida
X = normalized_df.drop('output', axis=1)
y = normalized_df['output']

# Dividir los datos en entrenamiento y validación
X_train, X_validation, y_train, y_validation = train_test_split(X, y, test_size=0.2,

from sklearn.linear_model import LogisticRegression

# Crear el modelo de regresión logística
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train, y_train)

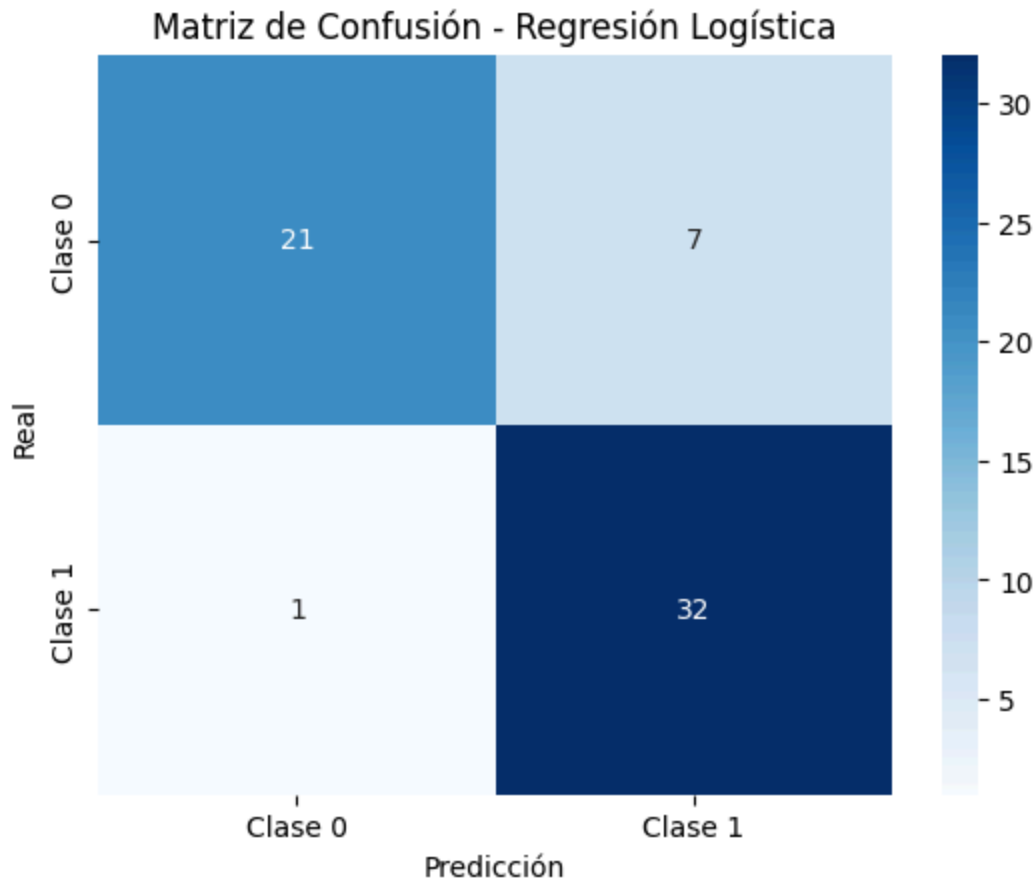
# Evaluar el modelo
y_pred = log_reg.predict(X_validation)
print("Regresión Logística")
print(classification_report(y_validation, y_pred))

# Mostrar la matriz de confusión
conf_matrix = confusion_matrix(y_validation, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', xticklabels=['Clase 0',
plt.title('Matriz de Confusión - Regresión Logística')
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.show()
```



## Regresión Logística

	precision	recall	f1-score	support
0	0.95	0.75	0.84	28
1	0.82	0.97	0.89	33
accuracy			0.87	61
macro avg	0.89	0.86	0.86	61
weighted avg	0.88	0.87	0.87	61



```

from sklearn.ensemble import RandomForestClassifier

# Crear el modelo Random Forest
rf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
rf.fit(X_train, y_train)

# Evaluar el modelo
y_pred = rf.predict(X_validation)
print("Random Forest")
print(classification_report(y_validation, y_pred))

# Mostrar la matriz de confusión
conf_matrix = confusion_matrix(y_validation, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', xticklabels=['Clase 0',
plt.title('Matriz de Confusión - Random Forest')
plt.xlabel('Predicción')

```

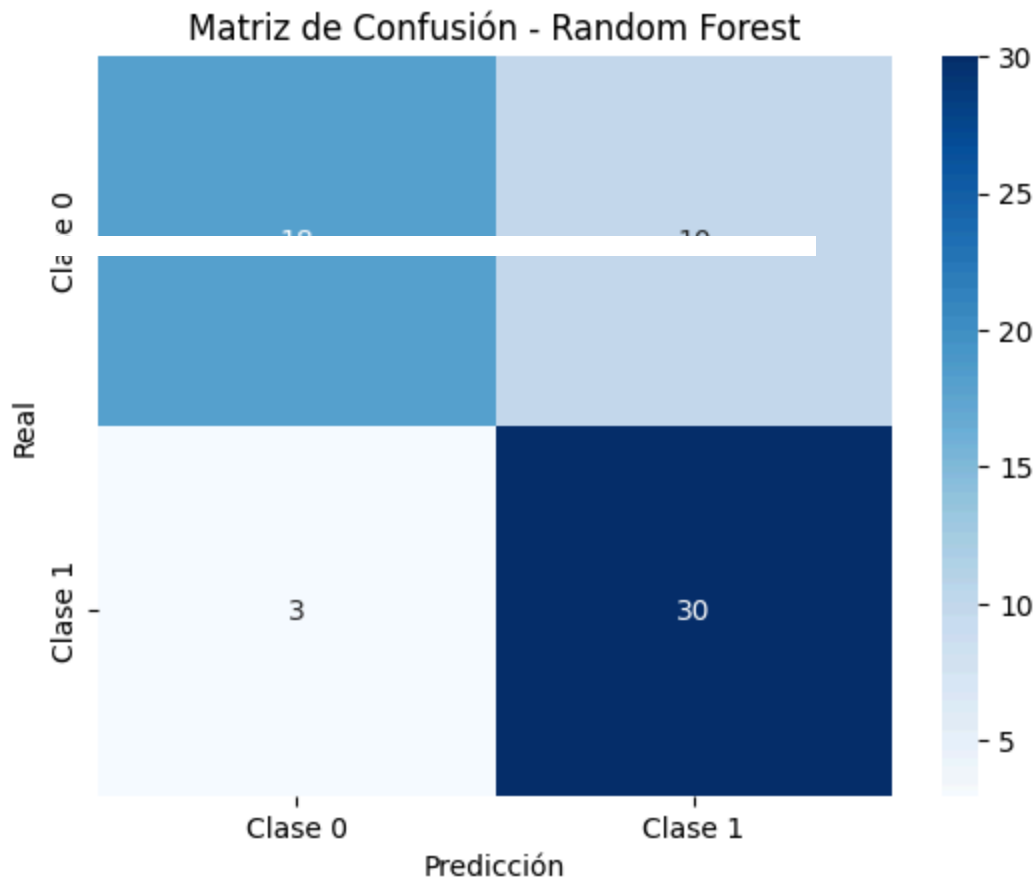


```
plt.ylabel('Real')
plt.show()
```



Random Forest

	precision	recall	f1-score	support
0	0.86	0.64	0.73	28
1	0.75	0.91	0.82	33
accuracy			0.79	61
macro avg	0.80	0.78	0.78	61
weighted avg	0.80	0.79	0.78	61



```
from sklearn.svm import SVC
```

```
# Crear el modelo SVM
svm = SVC(kernel='rbf', C=1.0, gamma='scale')
svm.fit(X_train, y_train)
```

```
# Evaluar el modelo
y_pred = svm.predict(X_validation)
print("SVM")
print(classification_report(y_validation, y_pred))
```

```
# Mostrar la matriz de confusión
```

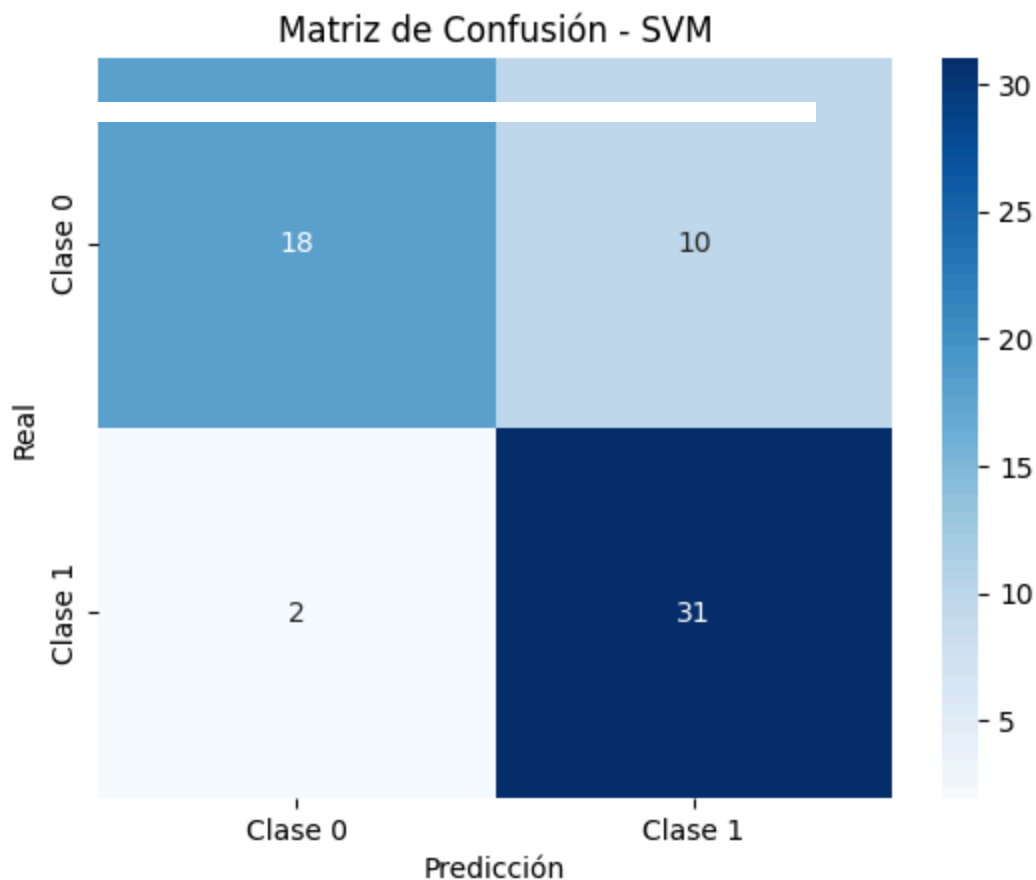
```

conf_matrix = confusion_matrix(y_validation, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', xticklabels=['Clase 0',
plt.title('Matriz de Confusión - SVM')
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.show()

```

⇒ SVM

	precision	recall	f1-score	support
0	0.90	0.64	0.75	28
1	0.76	0.94	0.84	33
accuracy			0.80	61
macro avg	0.83	0.79	0.79	61
weighted avg	0.82	0.80	0.80	61



```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

```

```

# Crear la red neuronal
def create_neural_network(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_dim,)),
        Dense(64, activation='relu'),

```

```
Dense(1, activation='sigmoid') # Para salida binaria
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
return model

# Crear y entrenar el modelo
input_dim = X_train.shape[1]
nn_model = create_neural_network(input_dim)
nn_model.fit(X_train, y_train, epochs=30, batch_size=32, verbose=1)

# Evaluar en el conjunto de validación
y_pred_prob = nn_model.predict(X_validation)
y_pred = (y_pred_prob > 0.5).astype(int)

print("Red Neuronal")
print(classification_report(y_validation, y_pred))

# Mostrar la matriz de confusión
conf_matrix = confusion_matrix(y_validation, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', xticklabels=['Clase 0',
plt.title('Matriz de Confusión - Red Neuronal')
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.show()
```



Epoch 1/30

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserW

super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

8/8  1s 2ms/step - accuracy: 0.7192 - loss: 0.6328

Epoch 2/30

8/8  0s 2ms/step - accuracy: 0.8564 - loss: 0.5240

Epoch 3/30

8/8  0s 2ms/step - accuracy: 0.8613 - loss: 0.4429

Epoch 4/30

8/8  0s 2ms/step - accuracy: 0.8754 - loss: 0.3791

Epoch 5/30

8/8  0s 3ms/step - accuracy: 0.8852 - loss: 0.3344

Epoch 6/30

8/8  0s 2ms/step - accuracy: 0.8762 - loss: 0.3093

Epoch 7/30

8/8  0s 2ms/step - accuracy: 0.8940 - loss: 0.2981

Epoch 8/30

8/8  0s 2ms/step - accuracy: 0.8716 - loss: 0.3042

Epoch 9/30

8/8  0s 2ms/step - accuracy: 0.8654 - loss: 0.3409

Epoch 10/30

8/8  0s 3ms/step - accuracy: 0.8743 - loss: 0.2892

Epoch 11/30

8/8  0s 2ms/step - accuracy: 0.8867 - loss: 0.3036

Epoch 12/30

8/8  0s 2ms/step - accuracy: 0.8772 - loss: 0.2832

Epoch 13/30

8/8  0s 2ms/step - accuracy: 0.8939 - loss: 0.2995

Epoch 14/30

8/8  0s 2ms/step - accuracy: 0.9151 - loss: 0.2563

Epoch 15/30

8/8  0s 2ms/step - accuracy: 0.9116 - loss: 0.2340

Epoch 16/30

8/8  0s 2ms/step - accuracy: 0.9086 - loss: 0.2498

Epoch 17/30

8/8  0s 2ms/step - accuracy: 0.9008 - loss: 0.2680

Epoch 18/30

8/8  0s 2ms/step - accuracy: 0.9423 - loss: 0.2095

Epoch 19/30

8/8  0s 2ms/step - accuracy: 0.9211 - loss: 0.2112

Epoch 20/30

8/8  0s 2ms/step - accuracy: 0.9324 - loss: 0.2465

Epoch 21/30

8/8  0s 2ms/step - accuracy: 0.9285 - loss: 0.2694

Epoch 22/30

8/8  0s 3ms/step - accuracy: 0.9477 - loss: 0.2015

Epoch 23/30

8/8  0s 3ms/step - accuracy: 0.9437 - loss: 0.1956

Epoch 24/30

8/8  0s 3ms/step - accuracy: 0.9555 - loss: 0.1804

Epoch 25/30

8/8  0s 3ms/step - accuracy: 0.9445 - loss: 0.1741

Epoch 26/30

8/8  0s 2ms/step - accuracy: 0.9379 - loss: 0.1908

Epoch 27/30

8/8  0s 3ms/step - accuracy: 0.9564 - loss: 0.1721

Epoch 28/30

8/8 ————— 0s 2ms/step – accuracy: 0.9200 – loss: 0.2094

```

# Reinicializar los índices de X_train y y_train
X_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)

# Función para evaluar la red neuronal
def evaluate_sigmoid_network(X, y, neurons_list, epochs_list, n_splits=10):
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
    results = []

    for neurons in neurons_list:
        for epochs in epochs_list:
            result_dict = {'neurons': neurons, 'epochs': epochs, 'precision': [], 'r
            for train_index, test_index in skf.split(X, y):
                X_train_fold, X_test_fold = X.iloc[train_index], X.iloc[test_index]
                y_train_fold, y_test_fold = y.iloc[train_index], y.iloc[test_index]

                model = create_sigmoid_model(neurons, X_train_fold.shape[1])
                start_time = time.time()
                model.fit(X_train_fold, y_train_fold, epochs=epochs, batch_size=10,
                training_time = time.time() - start_time

                y_pred = model.predict(X_test_fold)
                y_pred = (y_pred > 0.5).astype(int)

                precision, recall, f1, _ = precision_recall_fscore_support(y_test_fo
                result_dict['precision'].append(precision)
                result_dict['recall'].append(recall)
                result_dict['f1_score'].append(f1)
                result_dict['time'].append(training_time)

            results.append({
                'Neurons': neurons,
                'Epochs': epochs,
                'Precision': np.mean(result_dict['precision']),
                'Recall': np.mean(result_dict['recall']),
                'F1-Score': np.mean(result_dict['f1_score']),
                'Average Time': np.mean(result_dict['time'])
            })

    # Convert results to DataFrame and sort by F1-Score
    results_df = pd.DataFrame(results)
    results_df_sorted = results_df.sort_values(by='F1-Score', ascending=False)
    return results_df_sorted

# Configuraciones de neuronas y épocas
neurons_list = [10, 20, 50]
epochs_list = [10, 30, 50]

# Evaluar la red neuronal con los datos ajustados

```

```
results_df = evaluate_sigmoid_network(X_train, y_train, neurons_list, epochs_list)
```

```
results_df
```

```
➞ /usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserW  
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)  
1/1 _____ 0s 42ms/step  
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserW  
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)  
..
```

1/1 **vs** Time/Step

	Neurons	Epochs	Precision	Recall	F1-Score	Average Time
1	10	30	0.855574	0.851500	0.850125	3.016736
2	10	50	0.843247	0.839000	0.837860	4.315386
5	20	50	0.838644	0.835333	0.833539	4.187969
4	20	30	0.835251	0.830833	0.829464	2.803186
3	20	10	0.835781	0.830833	0.828928	1.562364
6	50	10	0.830168	0.826667	0.825494	1.462529
7	50	30	0.829723	0.827000	0.825249	2.965900
8	50	50	0.831598	0.826833	0.824707	4.139209
0	10	10	0.808286	0.802000	0.799486	1.603825



	Model	Precision	Recall	F1-Score	Accuracy	Additional Info
1	Regresión Logística	0.8850	0.8650	0.8600	0.87	-
0	Red Neuronal	0.8556	0.8515	0.8501	0.85	10 Neurons, 30 Epochs
2	Red Neuronal	0.8432	0.8390	0.8379	-	10 Neurons, 50 Epochs
3	Red Neuronal	0.8386	0.8353	0.8335	-	20 Neurons, 50 Epochs
6	Red Neuronal	0.8353	0.8308	0.8295	-	20 Neurons, 30 Epochs
7	Red Neuronal	0.8358	0.8308	0.8289	-	20 Neurons, 10 Epochs
8	Red Neuronal	0.8302	0.8267	0.8255	-	50 Neurons, 10 Epochs
9	Red Neuronal	0.8297	0.8270	0.8252	-	50 Neurons, 30 Epochs
10	Red Neuronal	0.8316	0.8268	0.8247	-	50 Neurons, 50 Epochs
11	Red Neuronal	0.8083	0.8020	0.7995	-	10 Neurons, 10 Epochs
5	SVM	0.8280	0.7950	0.7900	0.8	-
4	Random Forest	0.8040	0.7850	0.7800	0.79	-

## Conclusión del Trabajo

### Resumen de Resultados

Los modelos evaluados han arrojado distintos resultados en términos de **precisión**, **recall**, **F1-score** y **accuracy**. A continuación, se destacan las principales observaciones basadas en los resultados obtenidos:

#### 1. Regresión Logística:

- **F1-Score:** 0.8600
- **Accuracy:** 0.87

La regresión logística ha sido el modelo con el **mejor desempeño general**. Mostró un equilibrio entre precisión y recall, lo que lo convierte en una opción robusta para este conjunto de datos.

#### 2. Redes Neuronales:

- El rendimiento de las redes neuronales varía según la configuración de **número de neuronas y épocas**.
- La mejor configuración fue **10 neuronas y 30 épocas** con un **F1-score** de 0.8501 y una **precisión** de 0.8556.
- Aunque las redes neuronales pueden ser potentes, su desempeño fue **inferior** al de la regresión logística en este caso específico.

#### 3. SVM (Support Vector Machine):

- **F1-Score:** 0.7900
- **Accuracy:** 0.80

SVM tuvo un desempeño **moderado**, logrando un buen equilibrio entre precisión y recall, pero no alcanzó el nivel de desempeño de la regresión logística o las mejores redes neuronales.

#### 4. Random Forest:

- **F1-Score:** 0.7800
- **Accuracy:** 0.79

El modelo Random Forest obtuvo el rendimiento más bajo en términos de F1-score y precisión. Sin embargo, aún mostró una capacidad decente para clasificar correctamente los datos.



---

## Análisis del Desempeño de los Modelos

- **Regresión Logística** fue el modelo más eficiente para este conjunto de datos debido a su simplicidad y eficacia en problemas de clasificación binaria.
- **Redes Neuronales** mostraron un buen desempeño en varias configuraciones, pero requieren ajustes precisos de hiperparámetros (número de neuronas y épocas) para alcanzar su máximo potencial.
- **SVM** y **Random Forest** tuvieron un rendimiento aceptable, pero inferior al de la regresión logística y las mejores configuraciones de redes neuronales.

---

## Observaciones sobre el Balance de Datos

- Durante el proceso se evidenció que el **desbalance de clases** afectó el rendimiento de los modelos.
- El uso de métodos como **CTGAN** para la generación de datos sintéticos **no mejoró el desempeño** de los modelos.
- La **normalización** y el **One-Hot Encoding** fueron estrategias efectivas para preparar los datos y asegurar que los modelos trabajaran de manera adecuada.

---

## Recomendaciones

1. **Regresión Logística** es la mejor opción para este problema debido a su simplicidad y alto desempeño.
2. Si se opta por **Redes Neuronales**, se recomienda ajustar cuidadosamente los hiperparámetros y considerar configuraciones con **10 neuronas y 30 épocas** para obtener buenos resultados.
3. Se debe prestar atención al **balance de clases** en futuros trabajos para evitar que los modelos sesguen sus predicciones hacia la clase mayoritaria.
4. La **selección de características** debe realizarse con cuidado cuando se utiliza **One-Hot Encoding**, asegurando que se consideren grupos completos de columnas relacionadas con una variable categórica.

---

En conclusión, los resultados reflejan la importancia de **elegir el modelo adecuado** y de **preparar los datos correctamente** para obtener el mejor rendimiento posible.