

# Laboratorio 7 - Data Science

- Mónica Salvatierra - 22249
- Derek Arreaga - 22537

Link del repositorio: <https://github.com/alee2602/LAB7-DS>

## 1. Carga de Conjunto de datos

### Importación de librerías

```
In [4]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

### Construir un dataframe con el conjunto de datos

```
In [5]: df = pd.read_csv('diabetes.csv')
print(df.head())
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

### Verificar el tipo de datos de cada observación del dataset

```
In [6]: print(df.dtypes)
```

Pregnancies	int64
Glucose	int64
BloodPressure	int64
SkinThickness	int64
Insulin	int64
BMI	float64
DiabetesPedigreeFunction	float64
Age	int64
Outcome	int64
dtype:	object

### Descripción de variables


Variable	Tipo	Descripción
Pregnancies	int64	Cantidad de embarazos
Glucose	int64	Concentración de Glucosa
BloodPressure	int64	Presión de sangre diastólica (mmHg)
SkinThickness	int64	Grosor de pliegue cutáneo del Triscep (mm)
Insulin	int64	Suero de insulina 2-Horas (mu U/ml)
BMI	float64	Índice de masa corporal (Peso en Kg/(estatura en mts)) <sup>2</sup>
DiabetesPedigreeFunction	float64	Función de pedigree de diabetes
Age	int64	Edad en años
Outcome	int64	<b>1:</b> Diabetes , <b>0:</b> No Diabetes

## 2. Análisis Exploratorios de Datos

In [7]: `df.describe()`

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
<b>count</b>	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
<b>mean</b>	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.347135
<b>std</b>	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.476953
<b>min</b>	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
<b>25%</b>	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.000000
<b>50%</b>	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.000000
<b>75%</b>	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.000000
<b>max</b>	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	0.000000



### Obtener valores nulos

In [8]: `null_counts = df.isnull().sum()  
null_counts`

```
Out[8]: Pregnancies      0
        Glucose          0
        BloodPressure    0
        SkinThickness    0
        Insulin          0
        BMI              0
        DiabetesPedigreeFunction  0
        Age              0
        Outcome          0
        dtype: int64
```

### Contar la cantidad de ceros por variable

```
In [9]: zero_counts = (df == 0).sum()
        zero_counts
```

```
Out[9]: Pregnancies      111
        Glucose          5
        BloodPressure    35
        SkinThickness    227
        Insulin          374
        BMI              11
        DiabetesPedigreeFunction  0
        Age              0
        Outcome          500
        dtype: int64
```

Algunas de las variables contienen '0', lo cual es un poco realista, lo que podría tomarse como un valor faltante dentro del dataset. Para ello, se reemplazaran los valores sospechosos por 'Nan' para que AutoGluon los pueda manejar correctamente.

```
In [10]: import numpy as np

        df_clean = df.copy()

        # Columnas donde los ceros no son fisiológicos
        cols_with_invalid_zeros = ["Glucose", "BloodPressure", "SkinThickness", "Insulin",

        # Reemplazar ceros por NaN en esas columnas
        df_clean[cols_with_invalid_zeros] = df_clean[cols_with_invalid_zeros].replace(0, np

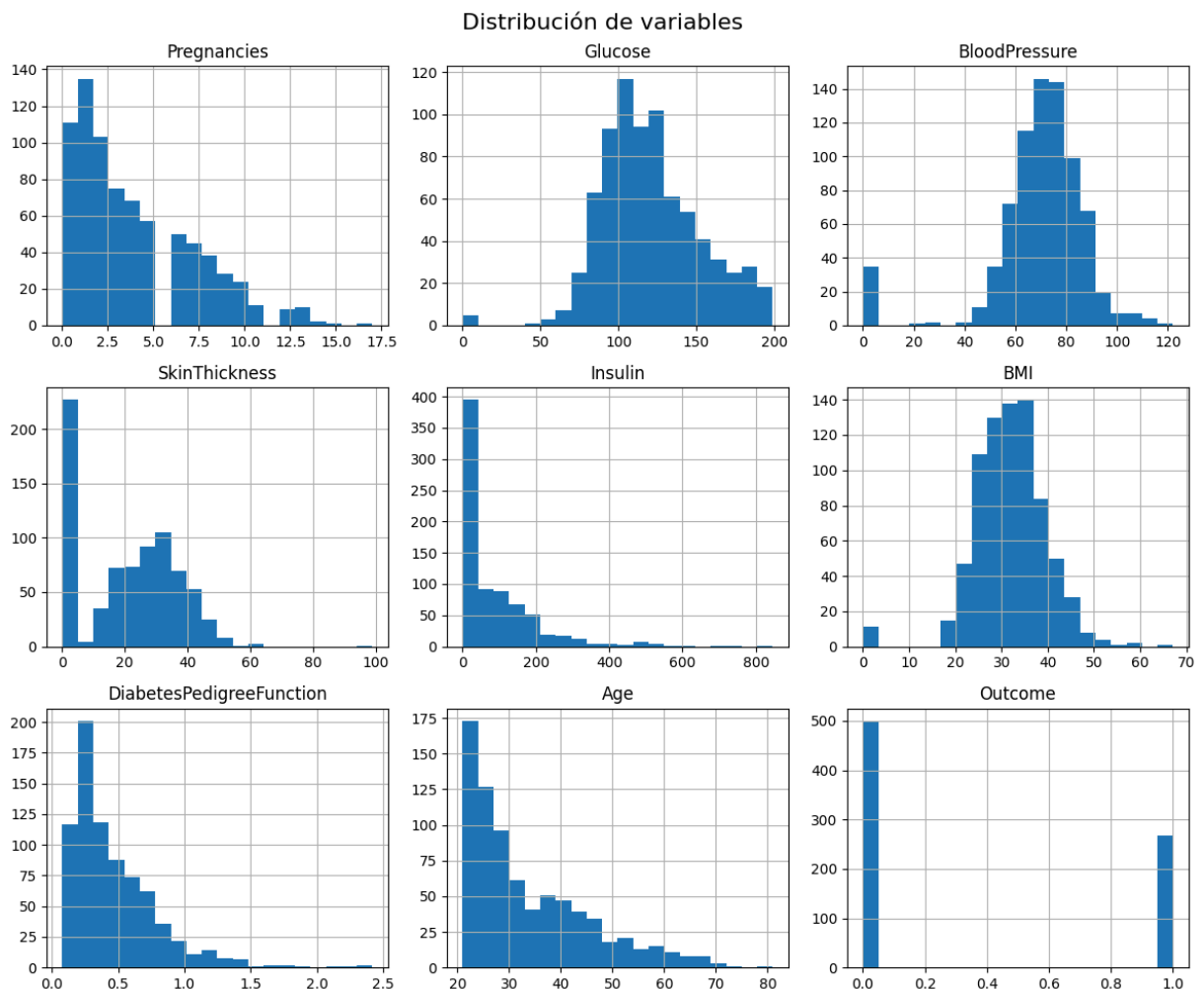
        missing_percent = df_clean.isnull().mean() * 100

        missing_percent
```

```
Out[10]: Pregnancies      0.000000
          Glucose         0.651042
          BloodPressure    4.557292
          SkinThickness    29.557292
          Insulin          48.697917
          BMI             1.432292
          DiabetesPedigreeFunction 0.000000
          Age             0.000000
          Outcome          0.000000
          dtype: float64
```

## Distribución de las variables utilizando histogramas

```
In [11]: df.hist(figsize=(12, 10), bins=20)
          plt.suptitle("Distribución de variables", fontsize=16)
          plt.tight_layout()
          plt.show()
```



### 1. **Pregnancies** (número de embarazos)

- La mayoría de los valores están entre 0 y 6, con pocos casos extremos (hasta 17 embarazos).
- La variable se encuentra sesgada hacia valores bajos.

- Es esperable porque la mayor parte de la población tiene pocos embarazos.

## 2. **Glucose** (nivel de glucosa en sangre)

- Distribución aproximadamente normal pero con un pico extraño en 0, lo que indica datos faltantes.
- Valores altos ( $>150$ ) están asociados con casos de diabetes.

## 3. **BloodPressure** (presión arterial)

- Centrada alrededor de 70–80 mmHg, que es un rango normal.
- Igual que en glucosa, aparece un grupo en 0 que no es fisiológicamente posible.

## 4. **SkinThickness** (grosor de pliegue cutáneo)

- Gran parte de los registros están en 0, indicando muchos valores faltantes.
- En los valores válidos, se concentra entre 15 y 40 mm, lo cual es clínicamente razonable.

## 5. **Insulin**

- Distribución muy sesgada, con muchísimos valores faltantes.
- En los valores válidos, varía ampliamente (hasta más de 800), lo que introduce outliers muy extremos.

## 6. **BMI** (Índice de Masa Corporal)

- Distribución con forma de campana, centrada en 30, lo que indica tendencia al sobrepeso/obesidad en la muestra.
- Aparecen algunos valores en 0, probablemente faltantes.

## 7. **DiabetesPedigreeFunction**

- Variable sesgada hacia la derecha
- La mayoría de las personas tienen valores bajos ( $<1$ ), pero hay casos extremos ( $>2$ ).
- Refleja que en pocas familias la predisposición genética es muy alta.

## 8. **Age**

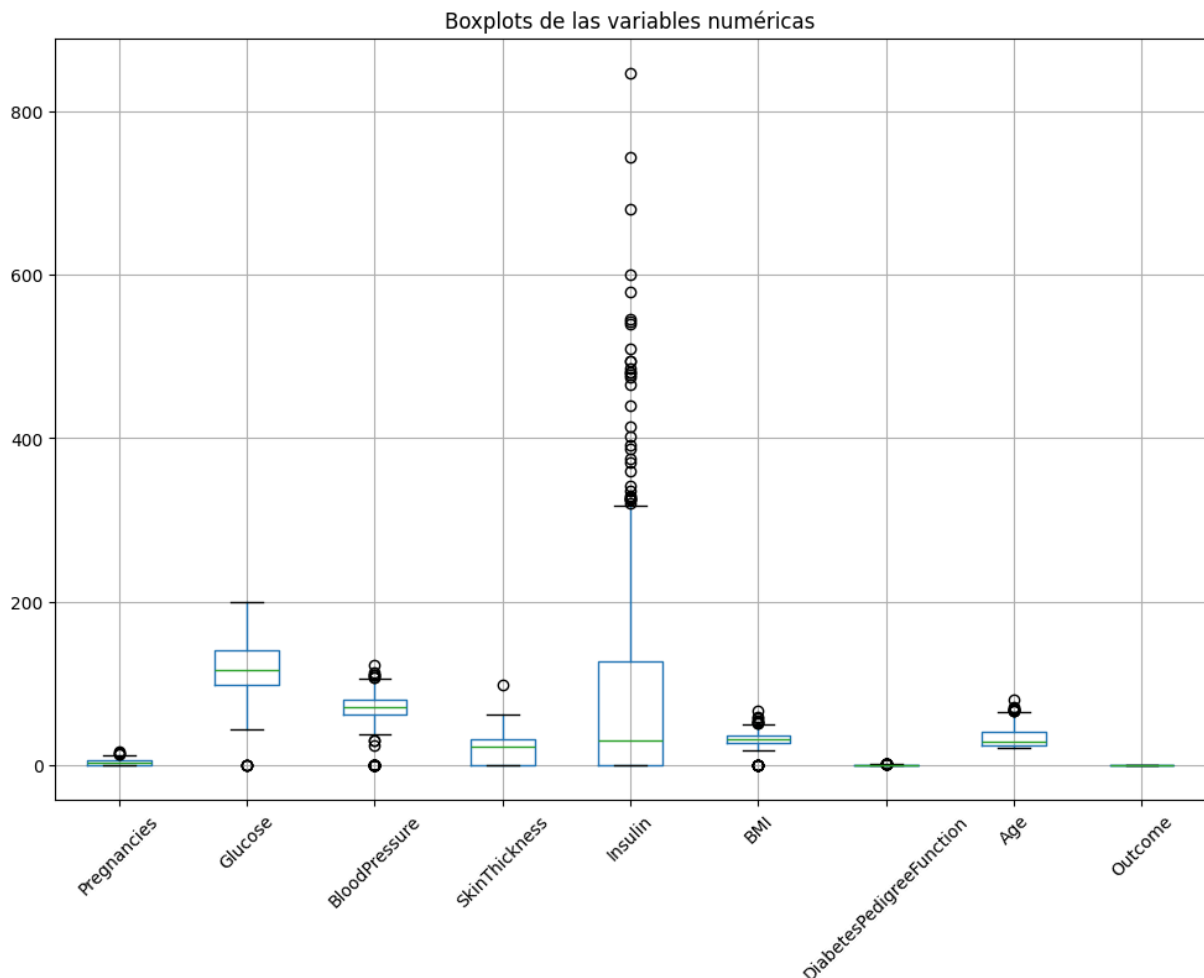
- Distribución sesgada hacia valores bajos dado que la mayoría de pacientes tienen entre 20 y 40 años.
- Sin embargo, también hay adultos mayores (hasta 81).

## 9. **Outcome** (variable objetivo)

- Desbalance claro, puesto que la mayoría son 0 (no diabéticos) y menos casos son 1 (diabéticos).
- Este desbalance debe considerarse en el modelado.

### Distribución de las variables utilizando boxplots para detectar valores atípicos

```
In [12]: plt.figure(figsize=(12, 8))
df.boxplot()
plt.title("Boxplots de las variables numéricas")
plt.xticks(rotation=45)
plt.show()
```



En el caso de **Glucose**, el boxplot evidencia una distribución relativamente concentrada entre 90 y 150, pero con presencia de valores atípicos en los extremos altos. Esto es consistente con la naturaleza clínica de la glucosa, donde niveles elevados se asocian con diabetes. Sin embargo, también se observa un grupo de valores en cero, que no son fisiológicamente posibles y deben tratarse como datos faltantes.

El boxplot de **BloodPressure** muestra una dispersión moderada, con la mayoría de valores entre 60 y 90, rango considerado normal. Al igual que con la glucosa, aparecen ceros

que indican registros incorrectos o faltantes, lo cual resalta la necesidad de una limpieza previa antes del modelado.

En **SkinThickness** y especialmente en **Insulin**, los boxplots revelan un gran número de valores atípicos. En el caso de la insulina, los outliers se extienden hasta valores extremadamente altos (más de 800), lo que genera una distribución muy dispersa. Además, ambos indicadores presentan acumulación de ceros, nuevamente señal de datos faltantes. Esta situación puede afectar el entrenamiento de modelos, ya que introduce ruido en las predicciones.

El **BMI** presenta una distribución más estable, con la mayoría de valores entre 25 y 40, lo que indica una tendencia general hacia el sobrepeso y la obesidad en la población estudiada. No obstante, algunos ceros vuelven a aparecer, lo que confirma la necesidad de tratarlos como nulos.

Finalmente, el boxplot de **Pregnancies** y **Age** muestran pocos valores atípicos y una distribución más compacta, por lo que pueden considerarse variables más confiables dentro del conjunto de datos. Esto las convierte en buenos candidatos para aportar información estable a los modelos predictivos.

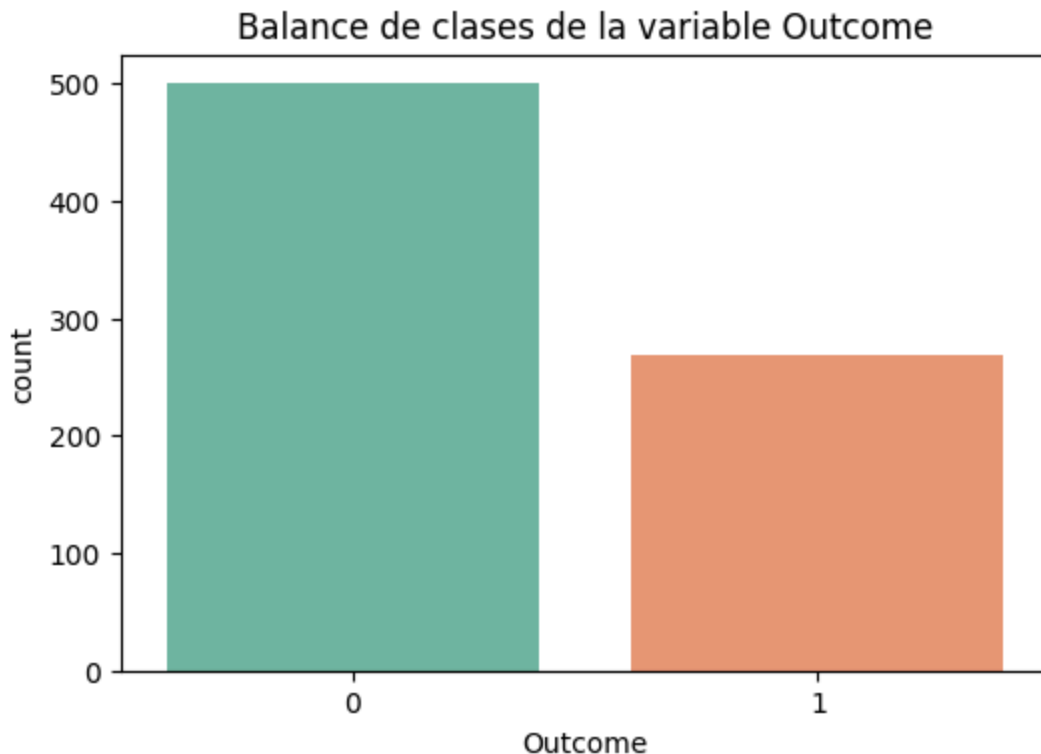
### Distribución de clases para la variable "Outcome"

```
In [13]: plt.figure(figsize=(6,4))
sns.countplot(x="Outcome", data=df, palette="Set2")
plt.title("Balance de clases de la variable Outcome")
plt.show()
```

C:\Users\Ale\AppData\Local\Temp\ipykernel\_13444\1934927133.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.countplot(x="Outcome", data=df, palette="Set2")
```



La gráfica de balance de clases de la variable **Outcome** evidencia un desbalance claro en el conjunto de datos. La clase 0, que representa a los pacientes **sin diagnóstico de diabetes**, tiene una frecuencia considerablemente mayor, con alrededor de 500 casos. En contraste, la clase 1, correspondiente a los pacientes **con diabetes**, cuenta con aproximadamente 260 observaciones.

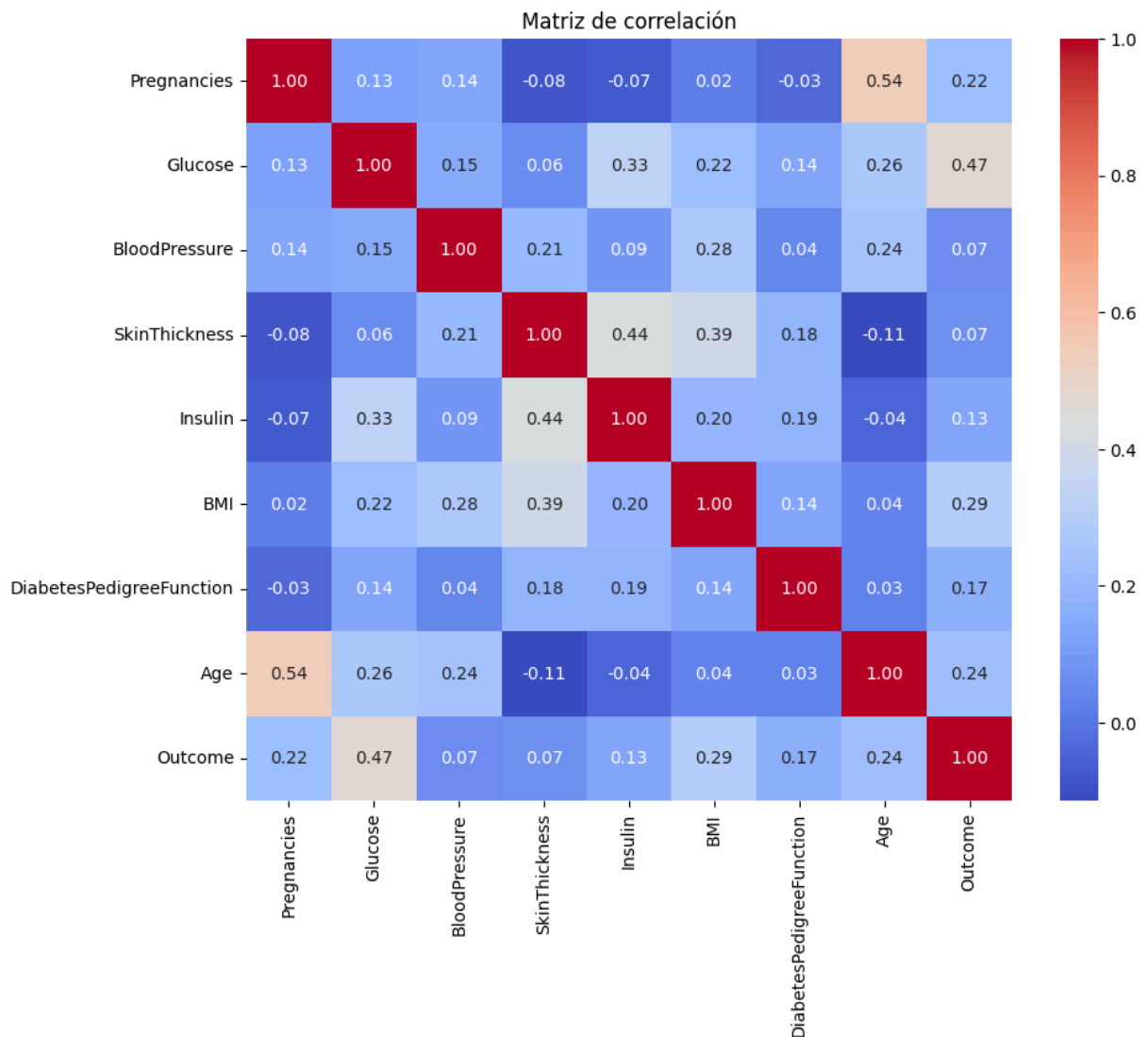
Este desbalance implica que el modelo podría inclinarse a predecir con mayor frecuencia la clase mayoritaria (no diabéticos), alcanzando una aparente alta precisión, pero con bajo desempeño en la identificación de los casos positivos de diabetes. En contextos médicos, este sesgo es especialmente delicado, ya que los falsos negativos (personas con diabetes clasificadas como sanas) representan un riesgo significativo.

Por esta razón, al evaluar el modelo no basta con utilizar solo la métrica de accuracy, sino que es fundamental complementar con precisión, recall y F1-score, que permiten medir mejor el rendimiento en la detección de la clase minoritaria. Además, puede ser recomendable aplicar técnicas de balanceo, como SMOTE (Synthetic Minority Over-sampling Technique) o ponderación de clases, para mitigar este problema y mejorar la capacidad predictiva hacia los casos positivos de diabetes.

### Matriz de correlación de todas las variables

```
In [14]: plt.figure(figsize=(10, 8))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Matriz de correlación")
plt.show()
```





El mapa de calor de correlaciones muestra relaciones importantes entre las variables y la variable objetivo **Outcome**. La correlación más fuerte se observa con Glucose, lo que confirma que los niveles altos de glucosa en sangre son el indicador principal para predecir la presencia de diabetes. Otras variables que presentan correlaciones moderadas son el **BMI** (índice de masa corporal) y la **Edad**, lo cual refleja que el sobrepeso y el envejecimiento son factores de riesgo relevantes en el desarrollo de la enfermedad.

Además, se destaca la relación entre **Pregnancies** y **Age**, lo cual es lógico porque a mayor edad es más probable que una mujer haya tenido más embarazos. También se observa cierta correlación entre **Insulin** y **SkinThickness**, dos medidas clínicas relacionadas con el metabolismo. Sin embargo, la mayoría de las correlaciones entre variables no son muy altas, lo que sugiere que cada característica aporta información complementaria para la predicción.

### 3. Entrenamiento con AutoGluon

```
In [17]: from autogluon.tabular import TabularDataset, TabularPredictor
        from sklearn.model_selection import train_test_split
```

c:\Users\Ale\OneDrive - UVG\Escritorio\Cuarto Año\Semestre2\Data Science\LAB7-DS\env\_autogluon\lib\site-packages\tqdm\auto.py:21: TqdmWarning: IPProgress not found. Please update jupyter and ipywidgets. See [https://ipywidgets.readthedocs.io/en/stable/user\\_install.html](https://ipywidgets.readthedocs.io/en/stable/user_install.html)

```
from .autonotebook import tqdm as notebook_tqdm
```

```
In [18]: df = df_clean.copy()

        train_data, test_data = train_test_split(df, test_size=0.2, random_state=8)

        # Convertir a formato AutoGluon
        train_data = TabularDataset(train_data)
        test_data = TabularDataset(test_data)

        # Variable objetivo
        label = "Outcome"
```

```
In [19]: predictor = TabularPredictor(
            label=label,
            eval_metric="accuracy"
        ).fit(
            train_data,
            presets="best_quality",
            time_limit=300
        )
```

```

No path specified. Models will be saved in: "AutogluonModels\ag-20250927_010818"
Verbosity: 2 (Standard Logging)
===== System Info =====
AutoGluon Version: 1.4.0
Python Version: 3.10.11
Operating System: Windows
Platform Machine: AMD64
Platform Version: 10.0.19045
CPU Count: 16
Memory Avail: 2.84 GB / 15.78 GB (18.0%)
Disk Space Avail: 225.43 GB / 931.89 GB (24.2%)
=====
Presets specified: ['best_quality']
Using hyperparameters preset: hyperparameters='zeroshot'
Setting dynamic_stacking from 'auto' to True. Reason: Enable dynamic_stacking when use_bag_holdout is disabled. (use_bag_holdout=False)
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8, num_bag_sets=1
DyStack is enabled (dynamic_stacking=True). AutoGluon will try to determine whether the input data is affected by stacked overfitting and enable or disable stacking as a consequence.
    This is used to identify the optimal `num_stack_levels` value. Copies of AutoGluon will be fit on subsets of the data. Then holdout validation data is used to detect stacked overfitting.
    Running DyStack for up to 75s of the 300s of remaining time (25%).
2025-09-26 19:08:22,931 INFO util.py:154 -- Missing packages: ['ipywidgets']. Run `pip install -U ipywidgets`, then restart the notebook server for rich notebook output.
    Running DyStack sub-fit in a ray process to avoid memory leakage. Enabling ray logging (enable_ray_logging=True). Specify `ds_args={'enable_ray_logging': False}` if you experience logging issues.
2025-09-26 19:08:36,784 INFO worker.py:1843 -- Started a local Ray instance. View the dashboard at 127.0.0.1:8265
    Context path: "c:\Users\Ale\OneDrive - UVG\Escritorio\Cuarto Año\Semestre2\Data Science\LAB7-DS\AutogluonModels\ag-20250927_010818\ds_sub_fit\sub_fit_ho"
(_dystack pid=21144) Running DyStack sub-fit ...
(_dystack pid=21144) Beginning AutoGluon training ... Time limit = 54s
(_dystack pid=21144) AutoGluon will save models to "c:\Users\Ale\OneDrive - UVG\Escritorio\Cuarto Año\Semestre2\Data Science\LAB7-DS\AutogluonModels\ag-20250927_010818\ds_sub_fit\sub_fit_ho"
(_dystack pid=21144) Train Data Rows: 545
(_dystack pid=21144) Train Data Columns: 8
(_dystack pid=21144) Label Column: Outcome
(_dystack pid=21144) Problem Type: binary
(_dystack pid=21144) Preprocessing data ...
(_dystack pid=21144) Selected class <--> label mapping: class 1 = 1, class 0 = 0
(_dystack pid=21144) Using Feature Generators to preprocess the data ...
(_dystack pid=21144) Fitting AutoMLPipelineFeatureGenerator...
(_dystack pid=21144) Available Memory: 3180.47 MB
(_dystack pid=21144) Train Data (Original) Memory Usage: 0.03 MB (0.0% of available memory)
(_dystack pid=21144) Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify special dtypes of the features.
(_dystack pid=21144) Stage 1 Generators:
(_dystack pid=21144) Fitting AsTypeFeatureGenerator...

```

```

(_dystack pid=21144) Stage 2 Generators:
(_dystack pid=21144) Fitting FillNaFeatureGenerator...
(_dystack pid=21144) Stage 3 Generators:
(_dystack pid=21144) Fitting IdentityFeatureGenerator...
(_dystack pid=21144) Stage 4 Generators:
(_dystack pid=21144) Fitting DropUniqueFeatureGenerator...
(_dystack pid=21144) Stage 5 Generators:
(_dystack pid=21144) Fitting DropDuplicatesFeatureGenerator...
(_dystack pid=21144) Types of features in original data (raw dtype, special dtype
s):
(_dystack pid=21144) ('float', []) : 6 | ['Glucose', 'BloodPressure', 'Sk
inThickness', 'Insulin', 'BMI', ...]
(_dystack pid=21144) ('int', []) : 2 | ['Pregnancies', 'Age']
(_dystack pid=21144) Types of features in processed data (raw dtype, special dtype
s):
(_dystack pid=21144) ('float', []) : 6 | ['Glucose', 'BloodPressure', 'Sk
inThickness', 'Insulin', 'BMI', ...]
(_dystack pid=21144) ('int', []) : 2 | ['Pregnancies', 'Age']
(_dystack pid=21144) 0.0s = Fit runtime
(_dystack pid=21144) 8 features in original data used to generate 8 features in p
rocessed data.
(_dystack pid=21144) Train Data (Processed) Memory Usage: 0.03 MB (0.0% of availa
ble memory)
(_dystack pid=21144) Data preprocessing and feature engineering runtime = 0.05s ...
(_dystack pid=21144) AutoGluon will gauge predictive performance using evaluation me
tric: 'accuracy'
(_dystack pid=21144) To change this, specify the eval_metric parameter of Predict
or()
(_dystack pid=21144) Large model count detected (110 configs) ... Only displaying th
e first 3 models of each family. To see all, set `verbosity=3`.
(_dystack pid=21144) User-specified model hyperparameters to be fit:
(_dystack pid=21144) {
(_dystack pid=21144) 'NN_TORCH': [{}, {'activation': 'elu', 'dropout_prob': 0.100
77639529843717, 'hidden_size': 108, 'learning_rate': 0.002735937344002146, 'num_laye
rs': 4, 'use_batchnorm': True, 'weight_decay': 1.356433327634438e-12, 'ag_args': {'n
ame_suffix': '_r79', 'priority': -2}}, {'activation': 'elu', 'dropout_prob': 0.11897
478034205347, 'hidden_size': 213, 'learning_rate': 0.0010474382260641949, 'num_layer
s': 4, 'use_batchnorm': False, 'weight_decay': 5.594471067786272e-10, 'ag_args': {'n
ame_suffix': '_r22', 'priority': -7}}],
(_dystack pid=21144) 'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'X
T'}}, {}, {'learning_rate': 0.03, 'num_leaves': 128, 'feature_fraction': 0.9, 'min_d
ata_in_leaf': 3, 'ag_args': {'name_suffix': 'Large', 'priority': 0, 'hyperparameter_
tune_kwargs': None}}],
(_dystack pid=21144) 'CAT': [{}, {'depth': 6, 'grow_policy': 'SymmetricTree', 'l2
_leaf_reg': 2.1542798306067823, 'learning_rate': 0.06864209415792857, 'max_ctr_compl
exity': 4, 'one_hot_max_size': 10, 'ag_args': {'name_suffix': '_r177', 'priority': -
1}}, {'depth': 8, 'grow_policy': 'Depthwise', 'l2_leaf_reg': 2.7997999596449104, 'le
arning_rate': 0.031375015734637225, 'max_ctr_complexity': 2, 'one_hot_max_size': 3,
'ag_args': {'name_suffix': '_r9', 'priority': -5}}],
(_dystack pid=21144) 'XGB': [{}, {'colsample_bytree': 0.6917311125174739, 'enable
_categorical': False, 'learning_rate': 0.018063876087523967, 'max_depth': 10, 'min_c
hild_weight': 0.6028633586934382, 'ag_args': {'name_suffix': '_r33', 'priority': -
8}}, {'colsample_bytree': 0.6628423832084077, 'enable_categorical': False, 'learning
_rate': 0.08775715546881824, 'max_depth': 5, 'min_child_weight': 0.6294123374222513,
'ag_args': {'name_suffix': '_r89', 'priority': -16}}],
(_dystack pid=21144) 'FASTAI': [{}, {'bs': 256, 'emb_drop': 0.5411770367537934,

```

```

'epochs': 43, 'layers': [800, 400], 'lr': 0.01519848858318159, 'ps': 0.2378294656660
4385, 'ag_args': {'name_suffix': '_r191', 'priority': -4}}, {'bs': 2048, 'emb_drop':
0.05070411322605811, 'epochs': 29, 'layers': [200, 100], 'lr': 0.08974235041576624,
'ps': 0.10393466140748028, 'ag_args': {'name_suffix': '_r102', 'priority': -11}}],
(_dystack pid=21144) 'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gin
i', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}}, {'criterion':
'squared_error', 'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression',
'quantile']}}],
(_dystack pid=21144) 'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gin
i', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}}, {'criterion':
'squared_error', 'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression',
'quantile']}}],
(_dystack pid=21144) }
(_dystack pid=21144) AutoGluon will fit 2 stack levels (L1 to L2) ...
(_dystack pid=21144) Fitting 108 L1 models, fit_strategy="sequential" ...
(_dystack pid=21144) Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 3
5.71s of the 53.57s of remaining time.
(_dystack pid=21144) Fitting 8 child models (S1F1 - S1F8) | Fitting with Parallel
LocalFoldFittingStrategy (8 workers, per: cpus=2, gpus=0, memory=0.15%)
(_dystack pid=21144) 0.7927 = Validation score (accuracy)
(_dystack pid=21144) 4.35s = Training runtime
(_dystack pid=21144) 0.02s = Validation runtime
(_dystack pid=21144) Fitting model: LightGBM_BAG_L1 ... Training model for up to 3.6
6s of the 21.51s of remaining time.
(_dystack pid=21144) Fitting 8 child models (S1F1 - S1F8) | Fitting with Parallel
LocalFoldFittingStrategy (8 workers, per: cpus=2, gpus=0, memory=0.21%)
(_dystack pid=21144) 0.8018 = Validation score (accuracy)
(_dystack pid=21144) 2.03s = Training runtime
(_dystack pid=21144) 0.01s = Validation runtime
(_dystack pid=21144) Fitting model: WeightedEnsemble_L2 ... Training model for up to
53.58s of the 13.37s of remaining time.
(_dystack pid=21144) Ensemble Weights: {'LightGBM_BAG_L1': 1.0}
(_dystack pid=21144) 0.8018 = Validation score (accuracy)
(_dystack pid=21144) 0.02s = Training runtime
(_dystack pid=21144) 0.0s = Validation runtime
(_dystack pid=21144) Fitting 108 L2 models, fit_strategy="sequential" ...
(_dystack pid=21144) Fitting model: LightGBMXT_BAG_L2 ... Training model for up to 1
3.30s of the 13.26s of remaining time.
(_dystack pid=21144) Fitting 8 child models (S1F1 - S1F8) | Fitting with Parallel
LocalFoldFittingStrategy (8 workers, per: cpus=2, gpus=0, memory=0.22%)
(_dystack pid=21144) 0.8037 = Validation score (accuracy)
(_dystack pid=21144) 1.8s = Training runtime
(_dystack pid=21144) 0.02s = Validation runtime
(_dystack pid=21144) Fitting model: LightGBM_BAG_L2 ... Training model for up to 5.7
7s of the 5.73s of remaining time.
(_dystack pid=21144) Fitting 8 child models (S1F1 - S1F8) | Fitting with Parallel
LocalFoldFittingStrategy (8 workers, per: cpus=2, gpus=0, memory=0.23%)
(_dystack pid=21144) 0.8183 = Validation score (accuracy)
(_dystack pid=21144) 2.07s = Training runtime
(_dystack pid=21144) 0.02s = Validation runtime
(_dystack pid=21144) Fitting model: WeightedEnsemble_L3 ... Training model for up to
53.58s of the -2.50s of remaining time.
(_dystack pid=21144) Ensemble Weights: {'LightGBM_BAG_L2': 1.0}
(_dystack pid=21144) 0.8183 = Validation score (accuracy)

```

```

(_dystack pid=21144) 0.02s = Training runtime
(_dystack pid=21144) 0.0s = Validation runtime
(_dystack pid=21144) AutoGluon training complete, total runtime = 56.19s ... Best model: WeightedEnsemble_L3 | Estimated inference throughput: 1988.5 rows/s (69 batch size)
(_dystack pid=21144) Disabling decision threshold calibration for metric `accuracy` due to having fewer than 10000 rows of validation data for calibration, to avoid overfitting (545 rows).
(_dystack pid=21144) `accuracy` is generally not improved through threshold calibration. Force calibration via specifying `calibrate_decision_threshold=True`.
(_dystack pid=21144) TabularPredictor saved. To load, use: predictor = TabularPredictor.load("c:\Users\Ale\OneDrive - UVG\Escritorio\Cuarto Año\Semestre2\Data Science\LAB7-DS\AutogluonModels\ag-20250927_010818\ds_sub_fit\sub_fit_ho")
(_dystack pid=21144) Deleting DyStack predictor artifacts (clean_up_fits=True) ...
Leaderboard on holdout data (DyStack):
      model score_holdout score_val eval_metric pred_time_test pred_time_val fit_time pred_time_test_marginal pred_time_val_marginal fit_time_marginal
1 stack_level can_infer fit_order
0 LightGBMXT_BAG_L2 0.768116 0.803670 accuracy 0.788948
0.036492 3.827686 0.349344 0.022205 1.7976
31 2 True 4
1 LightGBM_BAG_L1 0.753623 0.801835 accuracy 0.439604
0.014287 2.030055 0.439604 0.014287 2.0300
55 1 True 2
2 WeightedEnsemble_L2 0.753623 0.801835 accuracy 0.471605
0.015286 2.049055 0.032001 0.000999 0.0190
00 2 True 3
3 LightGBM_BAG_L2 0.753623 0.818349 accuracy 0.821555
0.034447 4.098598 0.381951 0.020160 2.0685
43 2 True 5
4 WeightedEnsemble_L3 0.753623 0.818349 accuracy 0.850707
0.036448 4.121598 0.029152 0.002001 0.0230
00 3 True 6
5 LightGBMXT_BAG_L1 0.753623 0.792661 accuracy 1.044777
0.015012 4.352777 1.044777 0.015012 4.3527
77 1 True 1
0 = Optimal num_stack_levels (Stacked Overfitting Occurred: True)
104s = DyStack runtime | 196s = Remaining runtime
Starting main fit with num_stack_levels=0.
For future fit calls on this dataset, you can skip DyStack to save time: `predictor.fit(..., dynamic_stacking=False, num_stack_levels=0)`
Beginning AutoGluon training ... Time limit = 196s
AutoGluon will save models to "c:\Users\Ale\OneDrive - UVG\Escritorio\Cuarto Año\Semestre2\Data Science\LAB7-DS\AutogluonModels\ag-20250927_010818"
Train Data Rows: 614
Train Data Columns: 8
Label Column: Outcome
Problem Type: binary
Preprocessing data ...
Selected class <--> label mapping: class 1 = 1, class 0 = 0
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
Available Memory: 3563.25 MB
Train Data (Original) Memory Usage: 0.04 MB (0.0% of available memory)
Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify special dtypes of the features.

```

```

Stage 1 Generators:
    Fitting AsTypeFeatureGenerator...
Stage 2 Generators:
    Fitting FillNaFeatureGenerator...
Stage 3 Generators:
    Fitting IdentityFeatureGenerator...
Stage 4 Generators:
    Fitting DropUniqueFeatureGenerator...
Stage 5 Generators:
    Fitting DropDuplicatesFeatureGenerator...
Types of features in original data (raw dtype, special dtypes):
    ('float', []) : 6 | ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', ...]
    ('int', [])   : 2 | ['Pregnancies', 'Age']
Types of features in processed data (raw dtype, special dtypes):
    ('float', []) : 6 | ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', ...]
    ('int', [])   : 2 | ['Pregnancies', 'Age']
0.1s = Fit runtime
8 features in original data used to generate 8 features in processed data.
Train Data (Processed) Memory Usage: 0.04 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.09s ...
AutoGluon will gauge predictive performance using evaluation metric: 'accuracy'
To change this, specify the eval_metric parameter of Predictor()
Large model count detected (110 configs) ... Only displaying the first 3 models of each family. To see all, set `verbosity=3`.
User-specified model hyperparameters to be fit:
{
    'NN_TORCH': [{}, {'activation': 'elu', 'dropout_prob': 0.10077639529843717, 'hidden_size': 108, 'learning_rate': 0.002735937344002146, 'num_layers': 4, 'use_batchnorm': True, 'weight_decay': 1.356433327634438e-12, 'ag_args': {'name_suffix': '_r79', 'priority': -2}}, {'activation': 'elu', 'dropout_prob': 0.11897478034205347, 'hidden_size': 213, 'learning_rate': 0.0010474382260641949, 'num_layers': 4, 'use_batchnorm': False, 'weight_decay': 5.594471067786272e-10, 'ag_args': {'name_suffix': '_r22', 'priority': -7}}],
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}], {}, {'learning_rate': 0.03, 'num_leaves': 128, 'feature_fraction': 0.9, 'min_data_in_leaf': 3, 'ag_args': {'name_suffix': 'Large', 'priority': 0, 'hyperparameter_tune_kwargs': None}},
    'CAT': [{}, {'depth': 6, 'grow_policy': 'SymmetricTree', 'l2_leaf_reg': 2.1542798306067823, 'learning_rate': 0.06864209415792857, 'max_ctr_complexity': 4, 'one_hot_max_size': 10, 'ag_args': {'name_suffix': '_r177', 'priority': -1}}, {'depth': 8, 'grow_policy': 'Depthwise', 'l2_leaf_reg': 2.7997999596449104, 'learning_rate': 0.031375015734637225, 'max_ctr_complexity': 2, 'one_hot_max_size': 3, 'ag_args': {'name_suffix': '_r9', 'priority': -5}}],
    'XGB': [{}, {'colsample_bytree': 0.6917311125174739, 'enable_categorical': False, 'learning_rate': 0.018063876087523967, 'max_depth': 10, 'min_child_weight': 0.6028633586934382, 'ag_args': {'name_suffix': '_r33', 'priority': -8}}, {'colsample_bytree': 0.6628423832084077, 'enable_categorical': False, 'learning_rate': 0.08775715546881824, 'max_depth': 5, 'min_child_weight': 0.6294123374222513, 'ag_args': {'name_suffix': '_r89', 'priority': -16}}],
    'FASTAI': [{}, {'bs': 256, 'emb_drop': 0.5411770367537934, 'epochs': 43, 'layers': [800, 400], 'lr': 0.01519848858318159, 'ps': 0.23782946566604385, 'ag_args': {'name_suffix': '_r191', 'priority': -4}}, {'bs': 2048, 'emb_drop': 0.05070411322605811, 'epochs': 29, 'layers': [200, 100], 'lr': 0.08974235041576624, 'ps': 0.10393466140748028, 'ag_args': {'name_suffix': '_r102', 'priority': -11}}],

```



```

'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression', 'quantile']}}],
'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression', 'quantile']}}],
}
Fitting 108 L1 models, fit_strategy="sequential" ...
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 196.38s of the 196.22s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting Strategy (8 workers, per: cpus=2, gpus=0, memory=0.13%)
    0.798    = Validation score    (accuracy)
    1.79s    = Training    runtime
    0.01s    = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 183.65s of the 183.50s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting Strategy (8 workers, per: cpus=2, gpus=0, memory=0.18%)
    0.8013   = Validation score    (accuracy)
    1.91s    = Training    runtime
    0.02s    = Validation runtime
Fitting model: RandomForestGini_BAG_L1 ... Training model for up to 175.86s of the 175.70s of remaining time.
    0.7752   = Validation score    (accuracy)
    2.29s    = Training    runtime
    0.09s    = Validation runtime
Fitting model: RandomForestEntr_BAG_L1 ... Training model for up to 173.41s of the 173.25s of remaining time.
    0.7752   = Validation score    (accuracy)
    0.86s    = Training    runtime
    0.09s    = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 172.39s of the 172.23s of remaining time.
    Memory not enough to fit 8 folds in parallel. Will train 4 folds in parallel instead (Estimated 10.91% memory usage per fold, 43.65%/80.00% total).
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting Strategy (4 workers, per: cpus=4, gpus=0, memory=10.91%)
    0.7915   = Validation score    (accuracy)
    16.68s   = Training    runtime
    0.01s    = Validation runtime
Fitting model: ExtraTreesGini_BAG_L1 ... Training model for up to 151.60s of the 151.44s of remaining time.
    0.7622   = Validation score    (accuracy)
    0.94s    = Training    runtime
    0.11s    = Validation runtime
Fitting model: ExtraTreesEntr_BAG_L1 ... Training model for up to 150.47s of the 150.31s of remaining time.
    0.7443   = Validation score    (accuracy)
    0.93s    = Training    runtime
    0.1s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 149.35s of the 149.19s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting

```



```

Strategy (8 workers, per: cpus=2, gpus=0, memory=0.01%)
    0.8078 = Validation score (accuracy)
    17.52s = Training runtime
    0.13s = Validation runtime
Fitting model: XGBoost_BAG_L1 ... Training model for up to 126.19s of the 126.03s of
remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting
Strategy (8 workers, per: cpus=2, gpus=0, memory=0.46%)
    0.7932 = Validation score (accuracy)
    3.85s = Training runtime
    0.04s = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 115.60s of the 11
5.44s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting
Strategy (8 workers, per: cpus=2, gpus=0, memory=0.01%)
    0.7997 = Validation score (accuracy)
    29.2s = Training runtime
    0.1s = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 80.38s of the 80.23
s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting
Strategy (8 workers, per: cpus=2, gpus=0, memory=0.46%)
    0.8078 = Validation score (accuracy)
    3.96s = Training runtime
    0.02s = Validation runtime
Fitting model: CatBoost_r177_BAG_L1 ... Training model for up to 70.62s of the 70.46
s of remaining time.
    Memory not enough to fit 8 folds in parallel. Will train 4 folds in parallel
instead (Estimated 12.52% memory usage per fold, 50.08%/80.00% total).
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting
Strategy (4 workers, per: cpus=4, gpus=0, memory=12.52%)
    0.7964 = Validation score (accuracy)
    9.78s = Training runtime
    0.02s = Validation runtime
Fitting model: NeuralNetTorch_r79_BAG_L1 ... Training model for up to 56.31s of the
56.16s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting
Strategy (8 workers, per: cpus=2, gpus=0, memory=0.01%)
    0.8029 = Validation score (accuracy)
    16.77s = Training runtime
    0.1s = Validation runtime
Fitting model: LightGBM_r131_BAG_L1 ... Training model for up to 34.10s of the 33.94
s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting
Strategy (8 workers, per: cpus=2, gpus=0, memory=0.25%)
    0.7964 = Validation score (accuracy)
    2.03s = Training runtime
    0.02s = Validation runtime
Fitting model: NeuralNetFastAI_r191_BAG_L1 ... Training model for up to 26.57s of th
e 26.42s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting
Strategy (8 workers, per: cpus=2, gpus=0, memory=0.02%)
    0.8094 = Validation score (accuracy)
    14.89s = Training runtime
    0.14s = Validation runtime
Fitting model: CatBoost_r9_BAG_L1 ... Training model for up to 5.66s of the 5.51s of

```

```

remaining time.
    Memory not enough to fit 8 folds in parallel. Will train 2 folds in parallel
    instead (Estimated 21.94% memory usage per fold, 43.88%/80.00% total).
    Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFitting
    Strategy (2 workers, per: cpus=8, gpus=0, memory=21.94%)
    0.7378 = Validation score (accuracy)
    14.19s = Training runtime
    0.01s = Validation runtime
Fitting model: WeightedEnsemble_L2 ... Training model for up to 196.38s of the -13.3
9s of remaining time.
    Ensemble Weights: {'LightGBMLarge_BAG_L1': 0.286, 'NeuralNetTorch_r79_BAG_L
1': 0.286, 'NeuralNetFastAI_r191_BAG_L1': 0.286, 'ExtraTreesGini_BAG_L1': 0.143}
    0.8192 = Validation score (accuracy)
    0.11s = Training runtime
    0.0s = Validation runtime
AutoGluon training complete, total runtime = 210.02s ... Best model: WeightedEnsembl
e_L2 | Estimated inference throughput: 283.9 rows/s (77 batch size)
Disabling decision threshold calibration for metric `accuracy` due to having fewer t
han 10000 rows of validation data for calibration, to avoid overfitting (614 rows).
    `accuracy` is generally not improved through threshold calibration. Force ca
libration via specifying `calibrate_decision_threshold=True`.
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("c:\Users\Ale
OneDrive - UVG\Escritorio\Cuarto Año\Semestre2\Data Science\LAB7-DS\AutogluonModel
s\ag-20250927_010818")

```

```
In [20]: performance = predictor.evaluate(test_data)
```

```
In [21]: predictor_lr = TabularPredictor(label=label).fit(
    train_data,
    hyperparameters={'LR': {}},
    time_limit=60
)
performance_lr = predictor_lr.evaluate(test_data)
```

```

No path specified. Models will be saved in: "AutogluonModels\ag-20250927_011433"
Verbosity: 2 (Standard Logging)
===== System Info =====
AutoGluon Version: 1.4.0
Python Version: 3.10.11
Operating System: Windows
Platform Machine: AMD64
Platform Version: 10.0.19045
CPU Count: 16
Memory Avail: 3.89 GB / 15.78 GB (24.6%)
Disk Space Avail: 228.04 GB / 931.89 GB (24.5%)
=====
No presets specified! To achieve strong results with AutoGluon, it is recommended to
use the available presets. Defaulting to `medium`...
Recommended Presets (For more details refer to https://auto.gluon.ai/stable/tutorials/tabular/tabular-essentials.html#presets):
presets='extreme' : New in v1.4: Massively better than 'best' on datasets <3
0000 samples by using new models meta-learned on https://tabarena.ai: TabPFNv2, TabI
CL, Mitra, and TabM. Absolute best accuracy. Requires a GPU. Recommended 64 GB CPU m
emory and 32+ GB GPU memory.
presets='best' : Maximize accuracy. Recommended for most users. Use in co
mpetitions and benchmarks.
presets='high' : Strong accuracy with fast inference speed.
presets='good' : Good accuracy with very fast inference speed.
presets='medium' : Fast training time, ideal for initial prototyping.
Beginning AutoGluon training ... Time limit = 60s
AutoGluon will save models to "c:\Users\Ale\OneDrive - UVG\Escritorio\Cuarto Año\Sem
estre2\Data Science\LAB7-DS\AutogluonModels\ag-20250927_011433"
Train Data Rows: 614
Train Data Columns: 8
Label Column: Outcome
AutoGluon infers your prediction problem is: 'binary' (because only two unique label
-values observed).
2 unique label values: [np.int64(0), np.int64(1)]
If 'binary' is not the correct problem_type, please manually specify the pro
blem_type parameter during Predictor init (You may specify problem_type as one of:
['binary', 'multiclass', 'regression', 'quantile'])
Problem Type: binary
Preprocessing data ...
Selected class <--> label mapping: class 1 = 1, class 0 = 0
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
Available Memory: 3964.15 MB
Train Data (Original) Memory Usage: 0.04 MB (0.0% of available memory)
Inferring data type of each feature based on column values. Set feature_meta
data_in to manually specify special dtypes of the features.
Stage 1 Generators:
Fitting AsTypeFeatureGenerator...
Stage 2 Generators:
Fitting FillNaFeatureGenerator...
Stage 3 Generators:
Fitting IdentityFeatureGenerator...
Stage 4 Generators:
Fitting DropUniqueFeatureGenerator...
Stage 5 Generators:
Fitting DropDuplicatesFeatureGenerator...

```

```

Types of features in original data (raw dtype, special dtypes):
  ('float', []) : 6 | ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', ...]
  ('int', [])   : 2 | ['Pregnancies', 'Age']
Types of features in processed data (raw dtype, special dtypes):
  ('float', []) : 6 | ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', ...]
  ('int', [])   : 2 | ['Pregnancies', 'Age']
0.1s = Fit runtime
8 features in original data used to generate 8 features in processed data.
Train Data (Processed) Memory Usage: 0.04 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.08s ...
AutoGluon will gauge predictive performance using evaluation metric: 'accuracy'
To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with holdout_frac=0.2, Train Rows: 491, Val Rows: 123
User-specified model hyperparameters to be fit:
{
  'LR': [{}],
}
Fitting 1 L1 models, fit_strategy="sequential" ...
Fitting model: LinearModel ... Training model for up to 59.92s of the 59.92s of remaining time.
  Fitting with cpus=16, gpus=0, mem=0.0/3.9 GB
  0.6992 = Validation score (accuracy)
  12.78s = Training runtime
  0.01s  = Validation runtime
Fitting model: WeightedEnsemble_L2 ... Training model for up to 59.92s of the 47.09s of remaining time.
  Ensemble Weights: {'LinearModel': 1.0}
  0.6992 = Validation score (accuracy)
  0.03s  = Training runtime
  0.0s   = Validation runtime
AutoGluon training complete, total runtime = 13.01s ... Best model: WeightedEnsemble_L2 | Estimated inference throughput: 12241.6 rows/s (123 batch size)
Disabling decision threshold calibration for metric `accuracy` due to having fewer than 10000 rows of validation data for calibration, to avoid overfitting (123 rows).
`accuracy` is generally not improved through threshold calibration. Force calibration via specifying `calibrate_decision_threshold=True`.
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("c:\Users\Ale\OneDrive - UVG\Escritorio\Cuarto Año\Semestre2\Data Science\LAB7-DS\AutogluonModels\ag-20250927_011433")

```

```

In [22]: feature_importance = predictor.feature_importance(test_data)
         print(feature_importance)

```

```

Computing feature importance via permutation shuffling for 8 features using 154 rows with 5 shuffle sets...
  22.57s = Expected runtime (4.51s per shuffle set)
  3.99s  = Actual runtime (Completed 5 of 5 shuffle sets)

```

	importance	stddev	p_value	n	p99_high \
Glucose	0.102597	0.028821	0.000675	5	0.161941
Age	0.012987	0.020014	0.110207	5	0.054197
Pregnancies	0.002597	0.008712	0.270735	5	0.020535
BloodPressure	-0.001299	0.008466	0.625566	5	0.016134
Insulin	-0.002597	0.011796	0.675869	5	0.021691
SkinThickness	-0.007792	0.012491	0.882252	5	0.017926
DiabetesPedigreeFunction	-0.011688	0.015503	0.916448	5	0.020233
BMI	-0.016883	0.007404	0.996506	5	-0.001639

	p99_low
Glucose	0.043254
Age	-0.028223
Pregnancies	-0.015341
BloodPressure	-0.018731
Insulin	-0.026886
SkinThickness	-0.033510
DiabetesPedigreeFunction	-0.043609
BMI	-0.032128

## 4. Evaluación del Modelo

```
In [ ]: from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc

# Tabla de mejores modelos

leaderboard = predictor.heartbeat(test_data, silent=True)
print(leaderboard)
```

	model	score_test	score_val	eval_metric	\
0	CatBoost_r9_BAG_L1	0.785714	0.737785	accuracy	
1	CatBoost_BAG_L1	0.772727	0.791531	accuracy	
2	ExtraTreesGini_BAG_L1	0.766234	0.762215	accuracy	
3	LightGBMXT_BAG_L1	0.766234	0.798046	accuracy	
4	ExtraTreesEntr_BAG_L1	0.766234	0.744300	accuracy	
5	LightGBM_r131_BAG_L1	0.766234	0.796417	accuracy	
6	CatBoost_r177_BAG_L1	0.753247	0.796417	accuracy	
7	NeuralNetTorch_BAG_L1	0.746753	0.799674	accuracy	
8	RandomForestEntr_BAG_L1	0.746753	0.775244	accuracy	
9	NeuralNetFastAI_BAG_L1	0.746753	0.807818	accuracy	
10	XGBoost_BAG_L1	0.746753	0.793160	accuracy	
11	LightGBMLarge_BAG_L1	0.740260	0.807818	accuracy	
12	NeuralNetTorch_r79_BAG_L1	0.740260	0.802932	accuracy	
13	WeightedEnsemble_L2	0.740260	0.819218	accuracy	
14	NeuralNetFastAI_r191_BAG_L1	0.720779	0.809446	accuracy	
15	RandomForestGini_BAG_L1	0.720779	0.775244	accuracy	
16	LightGBM_BAG_L1	0.720779	0.801303	accuracy	

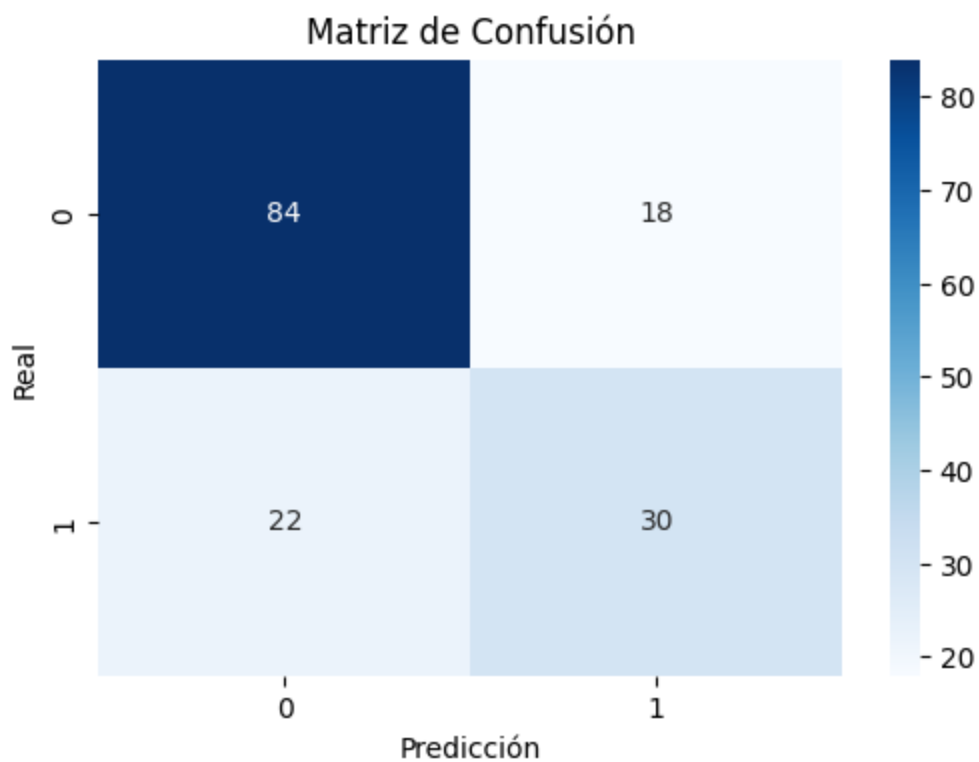
	pred_time_test	pred_time_val	fit_time	pred_time_test_marginal	\
0	0.213178	0.011168	14.194748	0.213178	
1	0.271153	0.010993	16.678583	0.271153	
2	0.086194	0.107566	0.937180	0.086194	
3	0.419027	0.014990	1.791476	0.419027	
4	0.482115	0.098994	0.934117	0.482115	
5	0.504240	0.015002	2.027767	0.504240	
6	0.295259	0.015004	9.780375	0.295259	
7	0.436313	0.095003	29.201660	0.436313	
8	0.537454	0.092000	0.863780	0.537454	
9	0.539728	0.130042	17.521769	0.539728	
10	1.409449	0.038986	3.847193	1.409449	
11	0.087992	0.017998	3.963068	0.087992	
12	0.122998	0.097991	16.770660	0.122998	
13	0.471341	0.366180	36.670855	0.002998	
14	0.171159	0.141623	14.892685	0.171159	
15	0.389651	0.094000	2.291869	0.389651	
16	0.526731	0.015015	1.908682	0.526731	

	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer	\
0	0.011168	14.194748	1	True	
1	0.010993	16.678583	1	True	
2	0.107566	0.937180	1	True	
3	0.014990	1.791476	1	True	
4	0.098994	0.934117	1	True	
5	0.015002	2.027767	1	True	
6	0.015004	9.780375	1	True	
7	0.095003	29.201660	1	True	
8	0.092000	0.863780	1	True	
9	0.130042	17.521769	1	True	
10	0.038986	3.847193	1	True	
11	0.017998	3.963068	1	True	
12	0.097991	16.770660	1	True	
13	0.001001	0.107263	2	True	
14	0.141623	14.892685	1	True	
15	0.094000	2.291869	1	True	
16	0.015015	1.908682	1	True	

	fit_order
0	16
1	5
2	6
3	1
4	7
5	14
6	12
7	10
8	4
9	8
10	9
11	11
12	13
13	17
14	15
15	3
16	2

```
In [24]: y_true = test_data[label]
y_pred = predictor.predict(test_data)
y_proba = predictor.predict_proba(test_data)[1]
```

```
In [25]: cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=[0,1], yticklabels=[
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.title("Matriz de Confusión")
plt.show()
```



El modelo con mejor desempeño en las pruebas fue **CatBoost\_r9\_BAG\_L1**, alcanzando una accuracy de 0.78 en el conjunto de prueba. Este resultado lo ubica por encima de otras variantes como CatBoost, LightGBM o RandomForest, mostrando que CatBoost es especialmente eficaz para este problema tabular.

La matriz de confusión del mejor modelo refleja un comportamiento equilibrado en la clasificación. El modelo identificó correctamente 84 casos negativos (verdaderos negativos) y 30 casos positivos (verdaderos positivos). Sin embargo, cometió 18 falsos positivos (predijo diabetes cuando no la había) y 22 falsos negativos (clasificó como sanos a pacientes con diabetes).

Este patrón evidencia que el modelo es relativamente confiable al reconocer a los pacientes sin diabetes, pero aún tiene limitaciones al identificar a los casos positivos, lo cual es crítico en un contexto médico, ya que los falsos negativos pueden tener consecuencias graves.

```
In [26]: print("Reporte de clasificación:")
print(classification_report(y_true, y_pred, digits=3))
```

Reporte de clasificación:

	precision	recall	f1-score	support
0	0.792	0.824	0.808	102
1	0.625	0.577	0.600	52
accuracy			0.740	154
macro avg	0.709	0.700	0.704	154
weighted avg	0.736	0.740	0.738	154

El reporte de clasificación muestra un accuracy del **74%**, lo que indica que el modelo acierta aproximadamente tres de cada cuatro predicciones. Si bien es un valor aceptable, conviene profundizar en las métricas por clase debido al desbalance del dataset.

Para la clase 0 (no diabéticos), el modelo alcanzó una precisión de **0.79** y un recall de **0.82**, lo que significa que identifica correctamente a la mayoría de los pacientes sanos y comete pocos falsos positivos. El F1-score de **0.81** confirma que esta clase está bien representada en las predicciones.

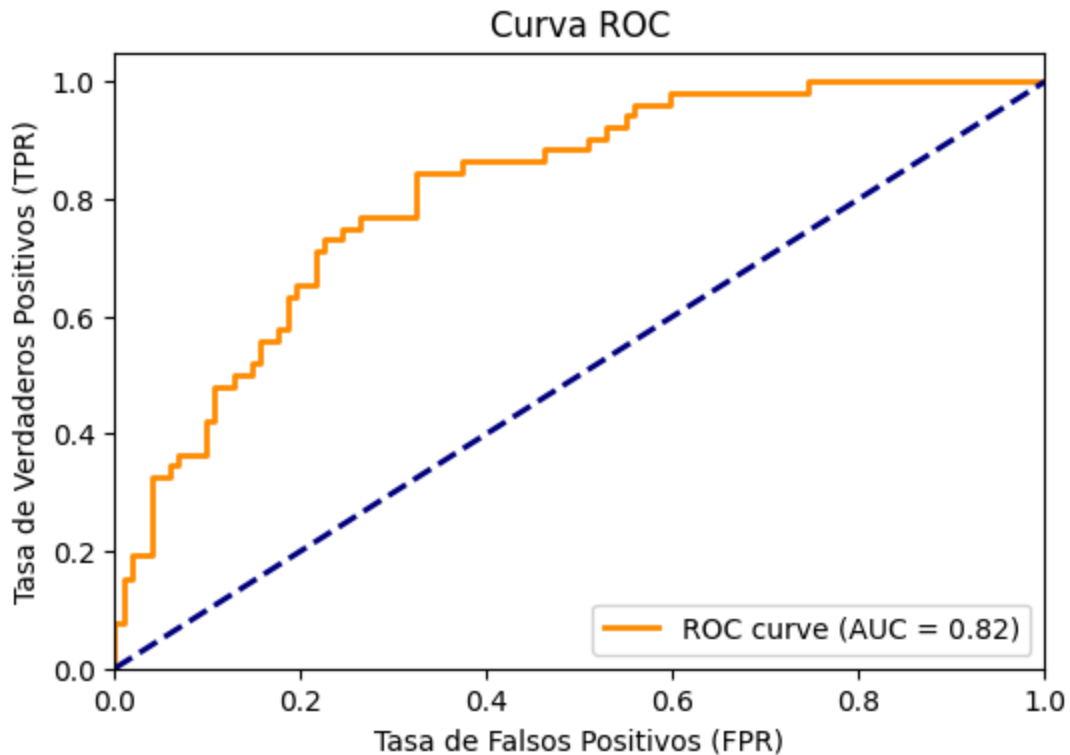
En contraste, la clase 1 (diabéticos) presenta un desempeño más limitado, con precisión de **0.62** y recall de **0.58**. Esto implica que, aunque más de la mitad de los pacientes con diabetes fueron identificados, el modelo todavía deja escapar un número significativo de casos positivos, lo que se refleja en un F1-score de apenas **0.60**.

```
In [28]: fpr, tpr, thresholds = roc_curve(y_true, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, color="darkorange", lw=2, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0,1], [0,1], color="navy", lw=2, linestyle="--")
```



```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("Tasa de Falsos Positivos (FPR)")
plt.ylabel("Tasa de Verdaderos Positivos (TPR)")
plt.title("Curva ROC")
plt.legend(loc="lower right")
plt.show()
```



La curva ROC ilustra la capacidad del modelo para distinguir entre pacientes con y sin diabetes en diferentes umbrales de decisión. En el eje horizontal se encuentra la tasa de falsos positivos (FPR) y en el eje vertical la tasa de verdaderos positivos (TPR). Un modelo ideal se acercaría a la esquina superior izquierda (TPR alto y FPR bajo), mientras que un modelo sin capacidad de discriminación seguiría la diagonal azul.

En este caso, la curva se ubica significativamente por encima de la diagonal, lo que confirma que el modelo logra una buena capacidad de separación entre clases. El valor de AUC = **0.82** respalda esta conclusión, ya que un AUC superior a **0.8** suele considerarse un resultado sólido en problemas de clasificación médica.

Sin embargo, aunque el AUC refleja un buen desempeño global, la curva también muestra que aún existe un compromiso entre la sensibilidad y la especificidad. Esto significa que dependiendo del umbral elegido, el modelo puede priorizar la detección de más casos positivos (mejor recall) o reducir los falsos positivos (mejor precisión). En un contexto clínico, convendría ajustar este límite para minimizar los falsos negativos, pues no detectar a un paciente con diabetes puede tener consecuencias más graves que clasificar erróneamente a un paciente sano.

## 5. Reflexión

### 1. ¿Qué ventajas y desventajas encontró al usar AutoGluon y AutoML en general?

- Una de las ventajas más grandes es la rapidez, puesto que en muy poco tiempo pudimos entrenar, comparar y seleccionar entre varios modelos sin tener que programar todo desde cero. También nos gustó que AutoGluon facilita la evaluación y que se adapta bien a problemas de clasificación tabular. La desventaja principal es que se pierde cierto control sobre el proceso, ya que no siempre es claro qué hace cada modelo por dentro ni cómo se ajustan todos los hiperparámetros. Además, puede consumir bastantes recursos de la computadora, lo que hace que a veces no sea tan ligero de usar.

### 2. ¿Qué métricas considera más relevantes en este problema?

- En este caso, lo más importante no es solo la accuracy, sino el recall y el F1-score de la clase positiva (pacientes diabéticos). El recall es fundamental porque mide qué tan bien el modelo detecta a los pacientes que realmente tienen diabetes. Un falso negativo en salud puede ser mucho más grave que un falso positivo. El F1-score, al equilibrar precisión y recall, también ayuda a evaluar el desempeño global sin que el desbalance de clases afecte demasiado.

### 3. ¿Qué preocupaciones deberían tomarse al aplicar este tipo de herramientas en salud?

- En un ámbito médico hay que ser especialmente cuidadosos. Primero, asegurar que los datos estén bien limpios y representen de manera justa a la población, porque un sesgo en los datos puede traer consecuencias graves. Segundo, recordar que un modelo de AutoML no debe reemplazar la decisión clínica, por lo que debe usarse como un apoyo para los profesionales de la salud. Finalmente, es importante evaluar con varias métricas y no confiarse únicamente en la accuracy, porque los falsos negativos pueden poner en riesgo la vida de los pacientes.

### 4. ¿Cómo compara esta experiencia con construir un modelo manualmente?

- Usar AutoGluon es mucho más rápido y práctico, porque en poco tiempo se obtienen resultados sólidos sin necesidad de invertir tanto esfuerzo en pruebas manuales. Sin embargo, construir un modelo manualmente ayuda a aprender y comprender a fondo qué pasa en cada etapa, desde la selección de características hasta el ajuste de hiperparámetros. Personalmente, creo que AutoML es excelente para tener resultados rápidos o para proyectos prácticos, pero el enfoque manual sigue siendo valioso para

entender de verdad cómo funcionan los algoritmos y poder explicarlos con mayor claridad.