UNIVERSIDAD COMPLUTENSE DE MADRID

MÁSTER EN
BIG DATA, DATA SCIENCE & ARTIFICIAL INTELLIGENCE

Machine Learning

TAREA DE EVALUACIÓN:

Estilo de Vida y Salud para la Clasificación de Obesidad

Trabajo realizado por

Diego López Escobar

Profesor

INMACULADA GUTIÉRREZ, JUAN ANTONIO GUEVARA, DANIEL GÓMEZ

MADRID – 28/03/2024

Índice

Objetivo:

Este proyecto se centra en la predicción de una variable binaria utilizando diversos algoritmos de clasificación. La variable binaria en cuestión indica si una persona padece obesidad o no. En este contexto, la obesidad se define cuando el valor de la variable "NObesity" corresponde a alguno de los tipos de obesidad, es decir, Obesity type I, Obesity type II u Obesity type III.

Introducción al dataset:

Previo al abordaje del problema en cuestión, resulta imperativo obtener una comprensión exhaustiva del dataset con el que estamos lidiando. A continuación se presentará una tabla en donde se explica y se obtiene una mejor comprensión de cada variable.

Categoría	Nombre	Descripción	Tipo de variable
Variable objetivo	NObesity	Según el ICM	object
Hábitos alimentarios	FAVC	Consumo frecuente de alimentos muy calóricos	object
Hábitos alimentarios	FCVC	Frecuencia de consumo de verduras	object
Hábitos alimentarios	NCP	Número de comidas principales	object
Hábitos alimentarios	CAEC	Consumo de alimentos entre comidas	object
Hábitos alimentarios	CH20	Consumo de agua	float64
Hábitos alimentarios	CALC	Consumo de alcohol	object
Condiciôn Física	SCC	Control del consumo de calorías	object
Condiciôn Física	FAF	Frecuencia de la actividad física	float64
Condiciôn Física	TUE	Tiempo de uso de dispositivos tecnológicos	float64
Condiciôn Física	MTRANS	Uso de transporte	object

Condiciôn Física	SMOKE	¿Fuma?	object
Características de la respuesta	Family History with Overweight	Historial familiar con sobrepeso	object
Características de la respuesta	Gender	¿Hombre o mujer?	object
Características de la respuesta	Age	Edad en años	float64
Características de la respuesta	Height	Altura en metros	float64
Características de la respuesta	Weight	Peso en kilogramos	float64

Al analizar los datos, se evidencia que las columnas pueden ser agrupadas en tres categorías distintas:

1. Columnas con valores susceptibles de agrupación en categorías discretas: En este grupo se encuentran "Age," "Height," y "Weight," cuyos valores representan medidas específicas y son más naturalmente susceptibles de ser clasificados en intervalos o categorías.
2. Columnas con valores numéricos interpretables: Este conjunto incluye "FCVC," "NCP," "CH2O," "FAF," y "TUE." Estas columnas representan datos numéricos que, en relación con su significado subyacente, pueden ser aproximados para brindar un sentido más coherente y comprensible.
3. Columnas con valores cuya variabilidad es limitada: En esta categoría se encuentran columnas como "Gender," "family_history_with_overweight," "FAVC," "CAEC," "SMOKE," "SCC," "CALC," "MTRANS," y "NObeyesdad." Estas columnas exhiben menos de ocho variaciones distintas, excluyendo "nan" como un valor válido, lo que sugiere una relativa simplicidad en su diversidad y facilita su agrupación para análisis posteriores.

Preparación de datos y análisis exploratorio:

Importación de bibliotecas:

Es fundamental destacar que, como parte de los archivos adjuntos a esta tarea, se incluye el archivo requirements.txt, el cual especifica todas las versiones de las bibliotecas utilizadas en el notebook. Esto se realiza con el fin de garantizar la reproducibilidad del análisis. Entre las bibliotecas más relevantes se encuentran:

- Versión de sklearn: 1.2.2
- Versión de pandas: 1.5.3
- Versión de numpy: 1.25.2
- Versión de seaborn: 0.13.1
- Versión de matplotlib: 3.6.3

Esto permite mantener la coherencia y consistencia en el entorno de desarrollo, facilitando la replicación de los resultados en diferentes configuraciones de sistemas.

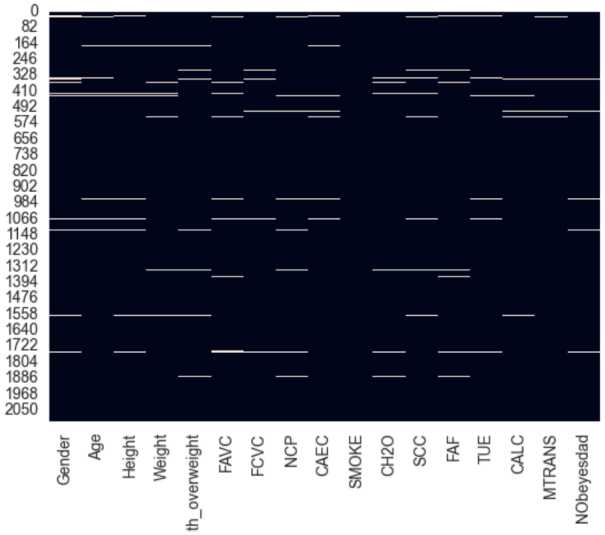
Exploración inicial de datos

Se realizó una exploración inicial de los datos para comprender su estructura y contenido. Se identificaron las siguientes características del conjunto de datos:

- Se verificó el tipo de datos de cada columna mediante el atributo `dtypes`.
- Se eliminaron los datos duplicados, se encontraron 24.
- Se observó un total de 2111 filas y 17 columnas.
- Se mostraron los primeros registros del DataFrame mediante la función `head()`.

Gestión de datos faltantes

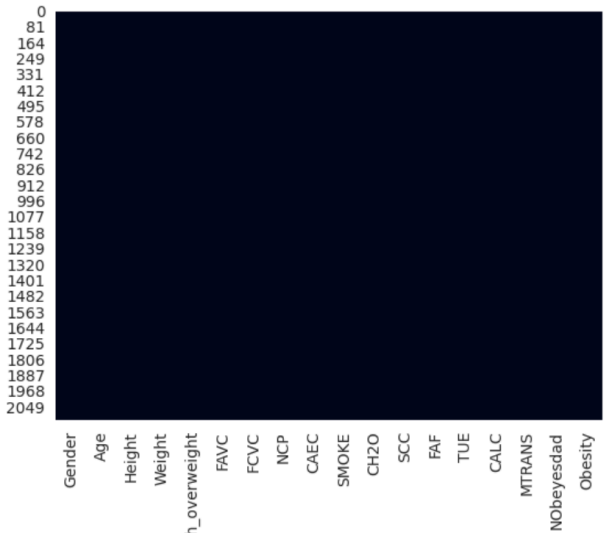
Se identificaron 778 datos faltantes. Distribuidos de la siguiente manera:



Se identificaron y trataron los valores faltantes en el conjunto de datos utilizando diferentes métodos de imputación, incluyendo KNNImputer para algunas columnas y el reemplazo por la media en otras, un ejemplo de uno de los KNN que se utilizó para imputar:

```
#Aplicó KNN para eliminar los NaN de las columnas que ya he tratado

# Combina las listas para incluir todas las columnas relevantes
columnas_imputar = binarias_list + category_cols

# Crea una instancia de KNNImputer con la métrica "nan_euclidean"
imputer_knn = KNNImputer(n_neighbors=5, metric="nan_euclidean")

# Aplica KNNImputer a todas las columnas con NaN
data2[columnas_imputar] = imputer_knn.fit_transform(data2[columnas_imputar])

# Muestra las primeras filas del DataFrame después de la imputación
data2.head()
```

Lo cual permitió que al finalizar con este proceso no hubiera más datos faltantes en el dataset, el mapa de calor se ve visualiza así:



Manipulación de columnas

Se transformaron las siguientes columnas categóricas en variables binarias mediante el uso de Label Encoder y mapeo manual de valores:

- Gender
- family_history_with_overweight
- FAVC
- SMOKE
- SCC

Las categorías 'yes' y 'Male' fueron mapeadas a 1, mientras que 'no' y 'Female' fueron mapeadas a 0.

Dado el propósito del proyecto, la columna categórica "NObeyesdad" se ha convertido en la columna "Obesity". En esta transformación, los valores correspondientes a Obesity type I, Obesity type II y Obesity type III se han asignado como 1, mientras que las demás opciones han sido designadas como 0.

```
# Función para mapear los valores
def map_obesity_category(value):
    if value in ['Insufficient_Weight', 'Normal_Weight', 'Overweight_Level_I',
'Overweight_Level_II']:
        return 0
    elif value in ['Obesity_Type_I', 'Obesity_Type_II', 'Obesity_Type_III']:
        return 1
    else:
        return np.nan

# Crear la nueva columna Obesity usando apply y la función map_obesity_category
data2['Obesity'] = data2['NObeyesdad'].apply(lambda x: map_obesity_category(x))
```

Transformación de datos numéricos

Para las columnas CAEC, CALC y MTRANS, que contienen datos categóricos con un máximo de 4, 4 y 5 categorías respectivamente, se empleó el codificador de etiquetas (label encoder) para transformar las categorías en números. Los valores NaN se mantuvieron como NaN, sin asignarles ningún número asociado, ya que el propósito es aplicar posteriormente un algoritmo de vecinos más cercanos (KNN) para eliminar los datos nulos.

```
# A cada categoria le asigno un número con LabelEncoder, dejando a los NaN intactos.
category_cols = ['CAEC', 'CALC', 'MTRANS']
#category_cols = [4, 4, 5]

# Crea una instancia de LabelEncoder
label_encoder = LabelEncoder()

# Itera sobre las columnas categóricas
for columna_categorica in category_cols:
   # Identifica las categorías existentes antes de la transformación
   categorias_existentes = data2[columna_categorica].dropna().unique()

   # Aplica LabelEncoder solo a las categorías existentes y deja los NaN como NaN
   data2[columna_categorica] = data2[columna_categorica].apply(lambda x:
label_encoder.fit(categorias_existentes).transform([x])[0] if pd.notna(x) and x in
categorias_existentes else x)
```

Ya una vez con las columnas CAEC, CALC y MTRANS con datos numéricos se procedió a utilizar un algoritmo de KNN para imputar los datos NaN y así eliminarlos por completo.

Al explorar las columnas FCVC, NCP, CH2O, FAF y TUE, se observó inicialmente que parecían contener solo valores enteros. Sin embargo, tras una inspección más detallada, se descubrió la presencia de valores decimales. Esta situación planteó la necesidad de decidir si mantener los datos tal como están o modificarlos para que consistan únicamente en números enteros.

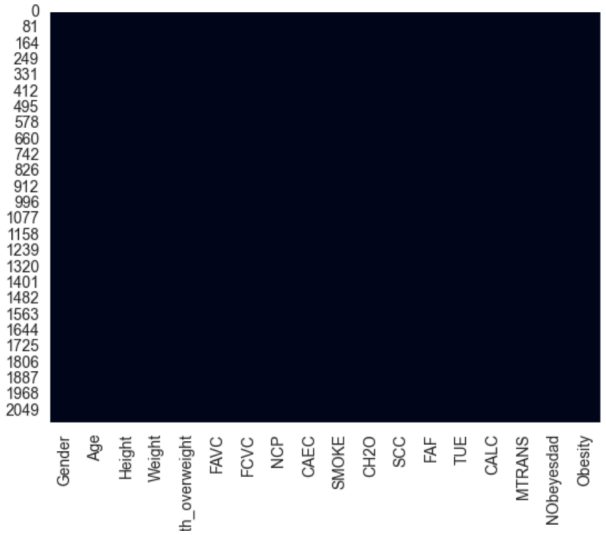
Para tomar decisiones informadas, se consultó la descripción de cada columna para obtener una comprensión de la naturaleza de los datos que deberían contener. Además, se analizó el porcentaje de valores enteros y decimales en cada variable. A continuación, se presentan ejemplos de los hallazgos:

- Para la columna FCVC, que representa el nivel de consumo de vegetales del individuo, se observó que el 60% de los datos son números enteros. Por lo tanto, se decidió transformar la columna a datos enteros.
- En cuanto a la columna NCP, que indica la cantidad de comidas principales que consume el individuo al día, se identificó que casi el 70% de los datos son números enteros. Dado que no tiene sentido tener fracciones de comidas, se optó por transformar esta columna también a valores enteros.

En este punto del proceso, todas las columnas FCVC, NCP, CH2O, FAF y TUE han sido transformadas para contener únicamente datos enteros. Ahora que no hay valores nulos en estas columnas, se procede a utilizar el algoritmo KNN para eliminar cualquier dato faltante que pueda quedar en el conjunto de datos.

Creación de nuevas características

Hasta este punto, se han abordado todas las variables, excepto Age, Height y Weight, las cuales aún contenían valores NaN en algunas de sus entradas. Para abordar este problema, se optó por utilizar un algoritmo KNN con un parámetro de vecinos igual a 3 y una métrica de distancia "nan_euclidean". Esto permitió eliminar los valores NaN de manera efectiva en estas columnas. En este punto, el mapa de calor de las variables null de los datos se ve así:



Dado el amplio rango de valores presentes en las variables Age, Height y Weight, se decidió agrupar estas variables en cuartiles para simplificar su representación. Los cuartiles seleccionados fueron 0.25, 0.50, 0.75 y 1.0. Para cada una de estas variables, se creó una nueva columna con el sufijo "_new". Por ejemplo, la variable Age ahora cuenta con una columna llamada Age_new, la cual asigna un valor entre 1 y 4 dependiendo del cuartil al que pertenezca el valor original de la variable.

```
# Define los cuantiles para cada categoría
quantiles = [0, 0.25, 0.5, 0.75, 1.0]

# Define las etiquetas para cada categoría
labels = [1, 2, 3, 4]

# Aplica la categorización a las columnas Age, Height y Weight y crea nuevas columnas
data2['Age_new'] = pd.qcut(data2['Age'], q=quantiles, labels=labels)
data2['Height_new'] = pd.qcut(data2['Height'], q=quantiles, labels=labels)
data2['Weight_new'] = pd.qcut(data2['Weight'], q=quantiles, labels=labels)
```

Hasta el momento, estas son las variables y sus tipos:

```
Obesity                          int64
NObeyesdad                      object
Gender                           int64
Age                              int64
Age_new                          int64
Height                         float64
Height_new                       int64
Weight                         float64
Weight_new                       int64
FAVC                             int64
FCVC                             int64
NCP                              int64
CAEC                             int64
SMOKE                            int64
CH2O                             int64
SCC                              int64
FAF                              int64
TUE                              int64
CALC                             int64
MTRANS                           int64
family_history_with_overweight   int64
```

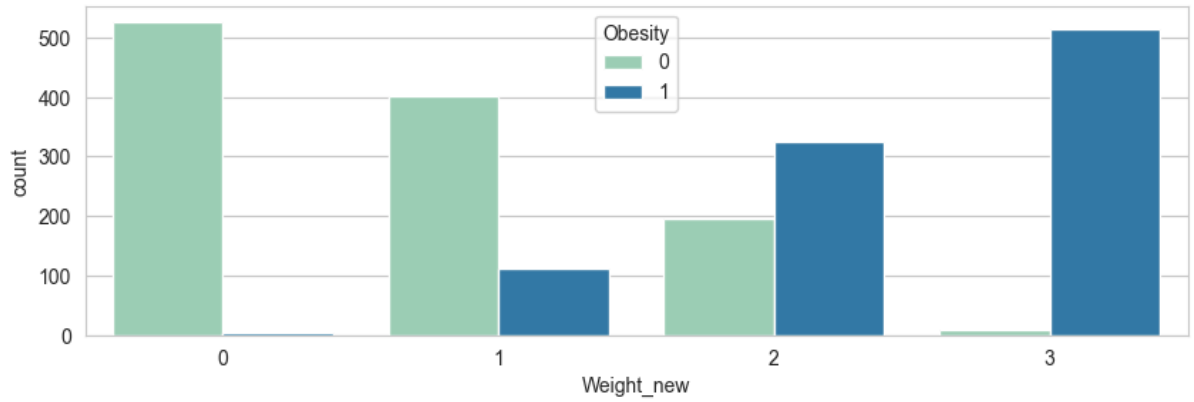
Eliminación de columnas

A partir de acá se creará un nuevo data frame llamado data3 a partir del data frame anterior (data2), pero se eliminarán las siguientes columnas: Age, NObeyesdad, Weight y Height.

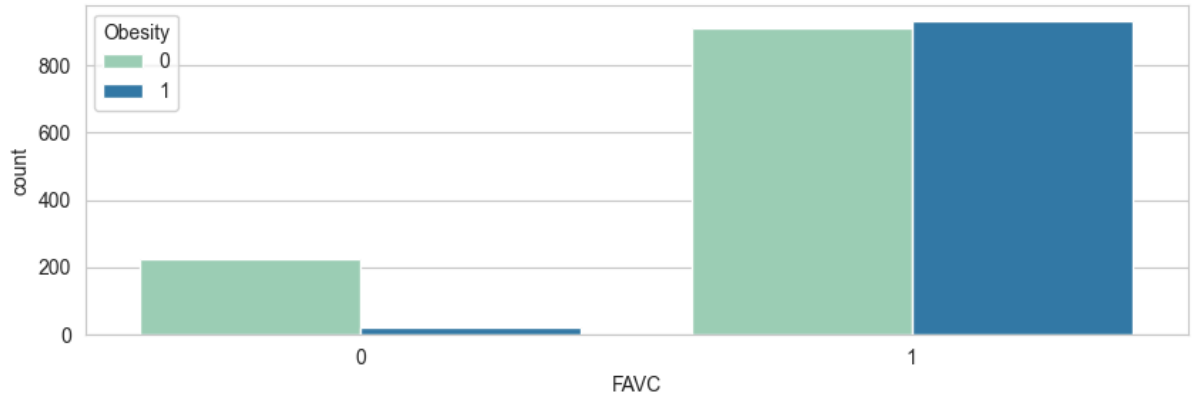
Visualización de datos

Se identificaron 5 variables que muestran una relación significativa con la variable objetivo de Obesity=1. Antes de comenzar con el análisis se debe de entender que de color azul se presenta donde Obesity=1.

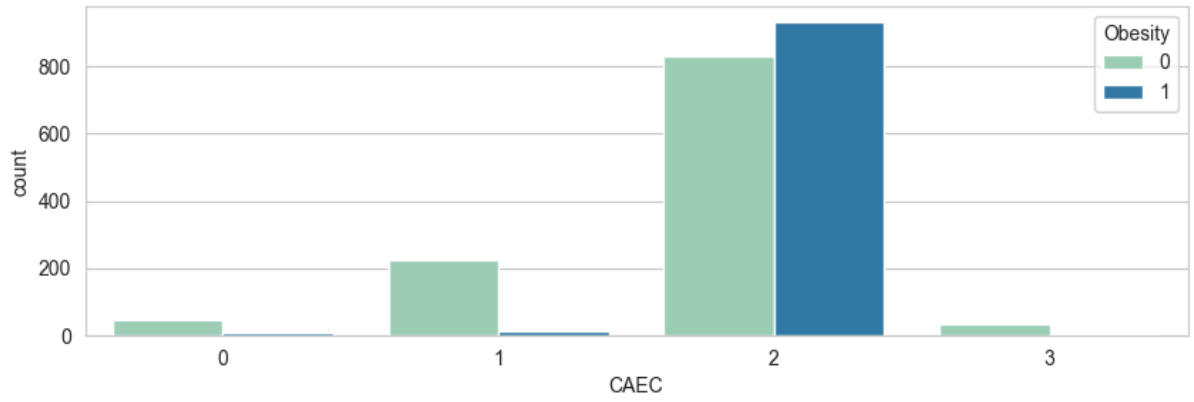
Weight: Peso de las personas clasificado por los cuartiles.



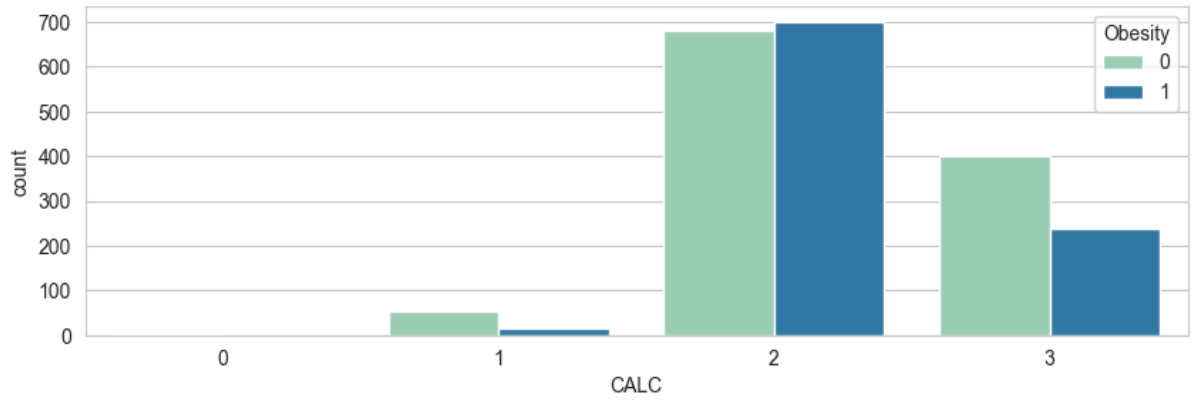
FAVC: Consumo frecuente de alimentos hipercalóricos, que son alimentos que se suelen consumir para subir de peso.



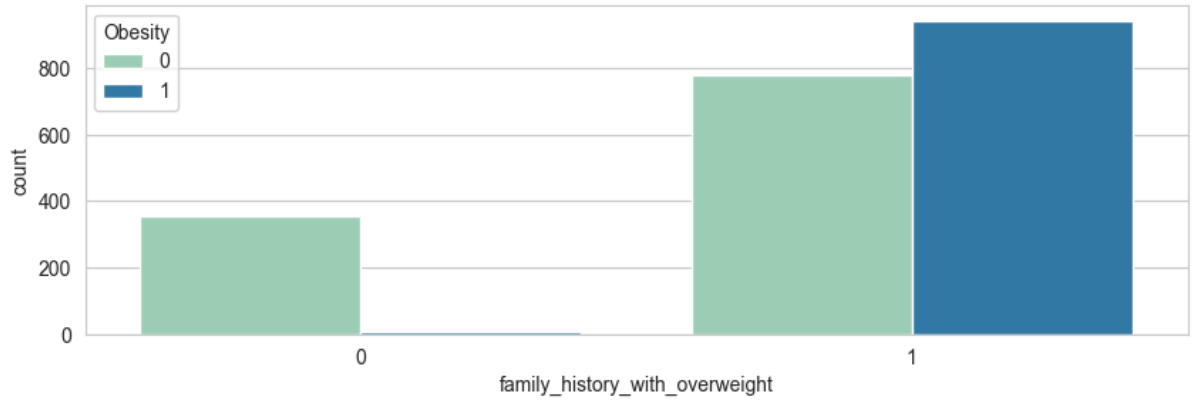
CAEC: Consumo de comidas entre alimentos.



CALC: Consumo de alcohol.



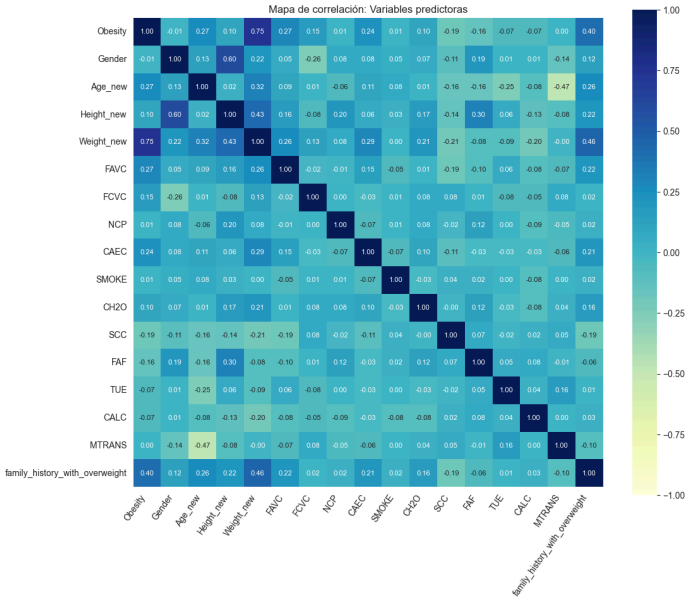
Historial familiar con sobrepeso:



Después de analizar la relación entre las variables y la variable objetivo de Obesity, se observa que ciertos factores como el peso, el consumo frecuente de alimentos hipercalóricos (FAVC), el patrón de consumo de comidas entre alimentos (CAEC), el consumo de alcohol (CALC) y el historial familiar con sobrepeso muestran una asociación significativa con la presencia de obesidad.

Análisis de correlación

Se calculó la matriz de correlación entre las variables predictoras y se eliminaron características altamente correlacionadas para evitar problemas de multicolinealidad.



Después de examinar las correlaciones entre las variables predictoras y la variable objetivo Obesity=1 y para seguir con la línea de las variables que tomamos en cuenta antes tenemos:

Peso (Weight_new): Presenta una correlación positiva significativa con la obesidad (0.75), lo que sugiere que el peso podría ser un indicador relevante para predecir la obesidad.

Consumo frecuente de alimentos hipercalóricos (FAVC): Exhibe una correlación positiva considerable con la obesidad (0.27), lo que indica que el consumo frecuente de este tipo de alimentos podría estar asociado con un mayor riesgo de obesidad.

Consumo de comidas entre alimentos (CAEC): Muestra una correlación moderada con la obesidad (0.24), lo que sugiere que los hábitos alimenticios podrían desempeñar un papel en la predicción de la obesidad.

Consumo de alcohol (CALC): Tiene una correlación negativa moderada con la obesidad (-0.07), lo que podría indicar que el consumo de alcohol está inversamente relacionado con la obesidad.

Historial familiar con sobrepeso (family_history_with_overweight): Exhibe una correlación positiva notable con la obesidad (0.40), lo que sugiere que el historial familiar con sobrepeso puede ser un factor predictor importante de la obesidad en los individuos.

Selección de muestra aleatoria:

Se seleccionó aleatoriamente una muestra de 1000 instancias del conjunto de datos utilizando la semilla específica (2313) para garantizar la reproducibilidad de los resultados.

Apartado 1: Obtención del Mejor Modelo de Regresión Logística y Mejor Red Neuronal

En este apartado, se procedió a explorar y comparar el rendimiento de un modelo de regresión logística y una red neuronal para la tarea de clasificación de la presencia de obesidad.

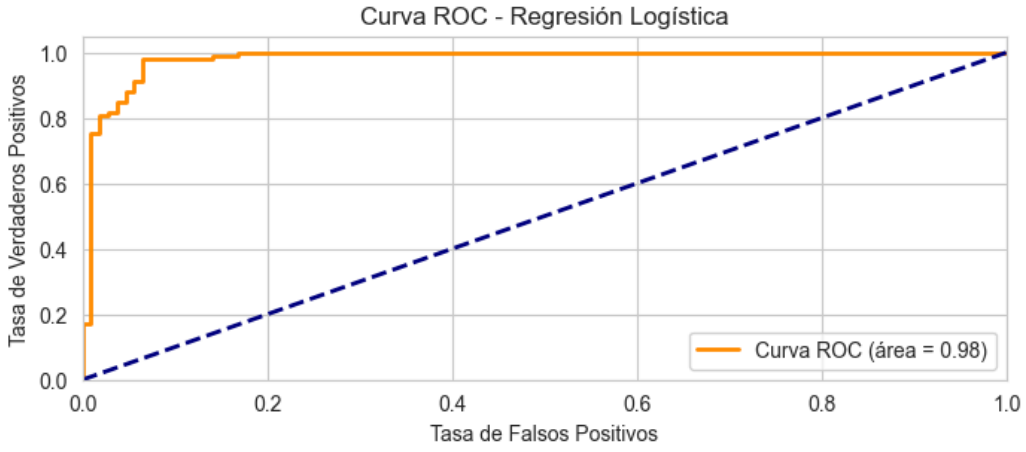
Mejor modelo de Regresión Logística:

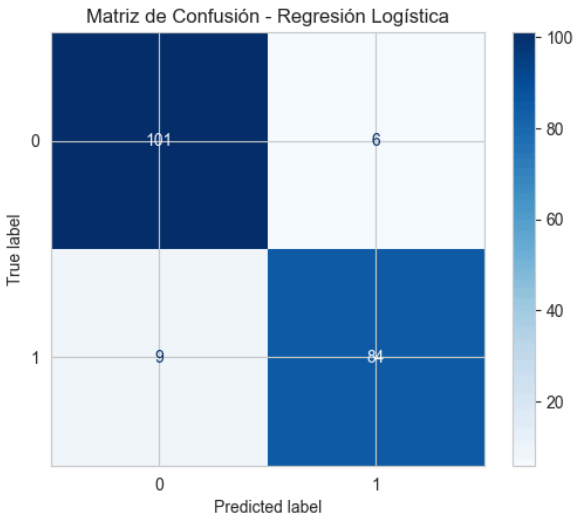
Para encontrar el mejor modelo de regresión logística, se utilizó un proceso de búsqueda de hiperparámetros mediante GridSearchCV. Se consideraron los siguientes hiperparámetros:

- Regularización (parámetro C).
- Tipo de regularización (penalty).
- Solver para la optimización del problema.

```
# Define el pipeline
pipeline = Pipeline([
   ('scaler', StandardScaler()),  # Escala los datos
   ('logreg', LogisticRegression()),  # Regresión Logística
])

# Define los parámetros para la búsqueda de hiperparámetros
param_grid_logreg = {
   'logreg__C': [0.001, 0.01, 0.1, 1, 10, 100],
   'logreg__penalty': ['l1', 'l2'],
   'logreg__solver': ['liblinear'],
}

# Aplica GridSearchCV para encontrar los mejores parámetros para la regresión logística
grid_search_logreg = GridSearchCV(pipeline, param_grid_logreg, cv=5, scoring='accuracy')
grid_search_logreg.fit(X_train, Y_train)
```

El modelo se evaluó utilizando la métrica de precisión (Accuracy) en un conjunto de prueba. Además, se realizaron visualizaciones de la matriz de confusión y la curva ROC para evaluar el rendimiento del modelo.





Los mejores parámetros encontrados para la regresión logística fueron:

- Regularización (C): 1
- Tipo de regularización: L1
- Solver: liblinear

El modelo de regresión logística obtuvo un Accuracy de **92.50%**en el conjunto de prueba.

Mejor Red Neuronal:

Posteriormente, se procedió a encontrar la mejor red neuronal utilizando como entrada las variables seleccionadas por el mejor modelo de regresión logística. Esto se realizó con el objetivo de comparar el rendimiento de ambos modelos.

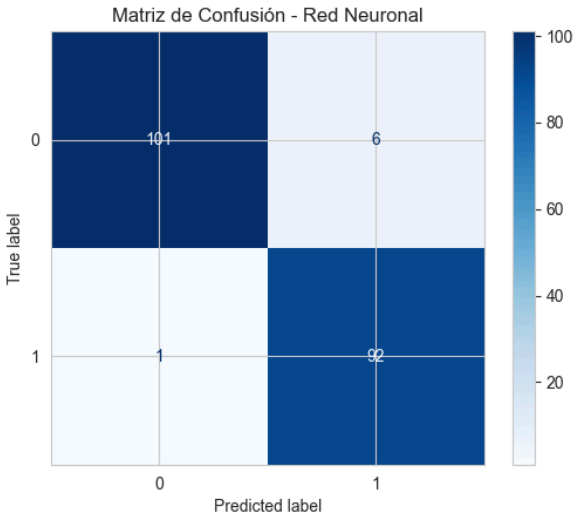
Se consideraron los siguientes hiperparámetros para la búsqueda de la mejor red neuronal:

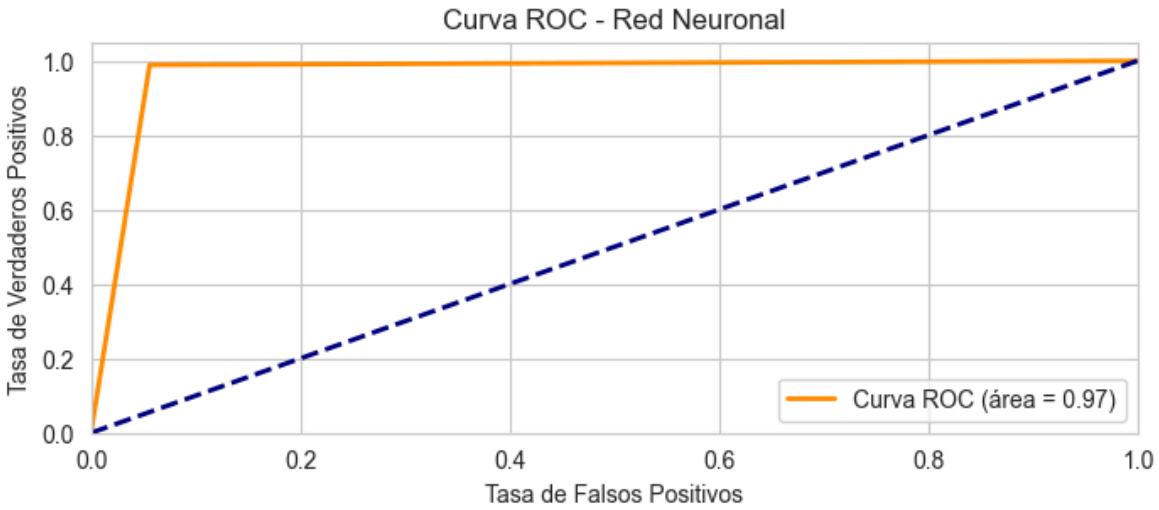
- Tamaño de las capas ocultas.
- Función de activación.
- Algoritmo de optimización.
- Tasa de regularización (alpha).

```
# Obtiene las variables seleccionadas por la regresión logística
selected_features_train =
SelectFromModel(grid_search_logreg.best_estimator_.named_steps['logreg']).transform(X_tr
ain)
selected_features_test =
SelectFromModel(grid_search_logreg.best_estimator_.named_steps['logreg']).transform(X_te
st)

# Define el nuevo pipeline con la red neuronal y las variables seleccionadas
pipeline_neural = Pipeline([
   ('scaler', StandardScaler()),
   ('neuralNetwork1', MLPClassifier())])
# Define los parámetros para la búsqueda de hiperparámetros de la red neuronal
```

```
param_grid_neural = {
    'neuralNetwork1__hidden_layer_sizes': [(15,), (17,), (20,), (25,)],
    'neuralNetwork1__activation': ['logistic', 'tanh', 'relu'],
    'neuralNetwork1__solver': ['adam', 'sgd', 'lbfgs'],
    'neuralNetwork1__alpha': [0.01, 0.01, 0.1]
}

# Aplica GridSearchCV para encontrar los mejores parámetros para la red neuronal
grid_search_neural = GridSearchCV(pipeline_neural, param_grid_neural, cv=5,
scoring='accuracy')
grid_search_neural.fit(selected_features_train, Y_train)
```

Al igual que con el modelo de regresión logística, se evaluó el rendimiento de la red neuronal utilizando la métrica de precisión (Accuracy) en un conjunto de prueba. Además, se generaron visualizaciones de la matriz de confusión y la curva ROC.





Los mejores parámetros encontrados para la red neuronal fueron:

- Tamaño de las capas ocultas: 20
- Función de activación: tanh
- Algoritmo de optimización: lbfgs
- Tasa de regularización (alpha): 0.01

La red neuronal alcanzó un Accuracy del **96.50%** en el conjunto de prueba y un área bajo la Curva ROC (AUC) del 97%.

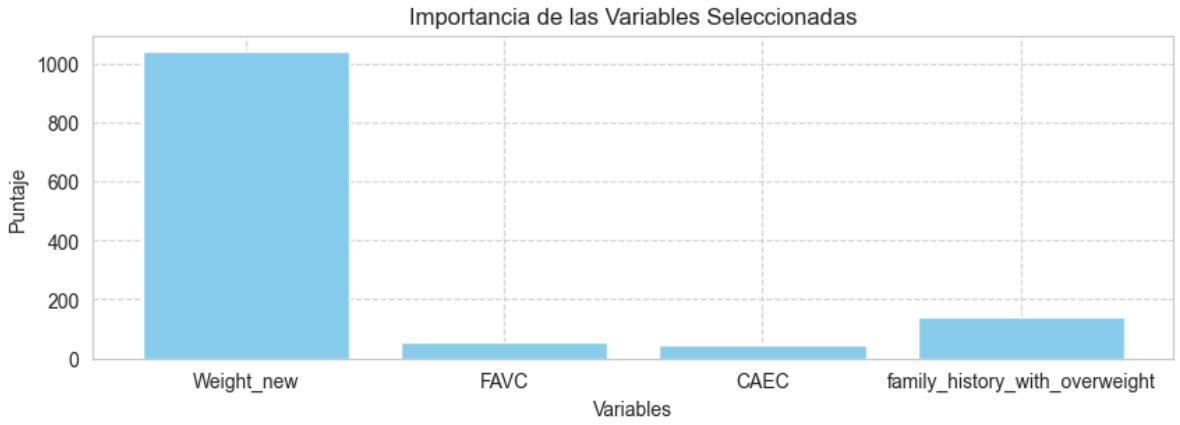
La regresión logística ofrece una interpretación más directa de los coeficientes y proporciona un rendimiento competitivo en comparación con la red neuronal en este conjunto de datos específico. Sin embargo, la red neuronal demuestra una mayor capacidad para capturar relaciones no lineales entre las características, lo que se refleja en su mayor accuracy en el conjunto de prueba.

Estos hallazgos respaldan **la elección de la red neuronal como el modelo preferido** para la clasificación de la obesidad en este contexto particular.

Apartado 2: Red Neuronal con Select K Best=4

El proceso de selección de variables mediante SelectKBest con k=4 permitió identificar las características más relevantes para el modelo. Las variables seleccionadas y sus respectivos pesos porcentuales en el modelo se muestran a continuación:

- Weight_new: Indica el peso de la persona, un factor fundamental en la determinación del estado de salud y el riesgo de obesidad. Representa un 81.16%
- family_history_with_overweight: Los antecedentes familiares de sobrepeso son un factor de riesgo bien establecido para el desarrollo de obesidad, lo que sugiere una influencia genética y ambiental en los hábitos alimenticios y el estilo de vida. Representa un 10.83%
- FAVC: Este hábito puede reflejar patrones dietéticos poco saludables asociados con un mayor riesgo de obesidad. Representa un 4.42%
- CAEC: El hábito de picar entre comidas puede contribuir al aumento de la ingesta calórica diaria y, en última instancia, al aumento de peso. Representa un 3.60%



Estas variables representan las características más influyentes para la predicción de la variable objetivo en el conjunto de datos, según el modelo de red neuronal con la configuración óptima encontrada.

Los parámetros seleccionados para el modelo de red neuronal fueron determinados mediante un proceso de búsqueda exhaustiva que tuvo como objetivo maximizar el Área bajo la Curva (AUC), una métrica crucial para evaluar el rendimiento de modelos de clasificación en conjuntos de datos desbalanceados o con diferentes costos de errores. Se evaluaron varias combinaciones de parámetros para encontrar aquellas que optimizan esta métrica.

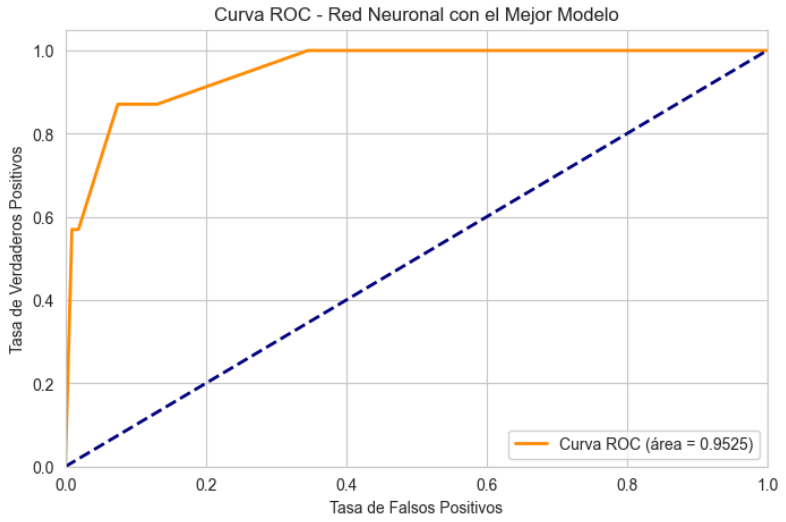
La arquitectura de la red neuronal se configuró con una sola capa oculta, con diferentes números de neuronas en esa capa, activadas por distintas funciones de activación, y utilizando distintos algoritmos de optimización para el proceso de aprendizaje. Además, se aplicó una tasa de regularización (alpha) para evitar el sobreajuste del modelo.

```
# Crea el pipeline para el SelectKBest y la Red Neuronal
pipeline = Pipeline([
   ('scaler2', StandardScaler()),
   ('k_best', SelectKBest(score_func=f_classif, k=4)),
   ('neuralNetwork2', MLPClassifier(random_state=seed))
])

# Combinaciones de parámetros a probar
param_grid_neural = {
   'neuralNetwork2__hidden_layer_sizes': [(15,), (20), (25)],
   'neuralNetwork2__activation': ['logistic', 'tanh', 'relu'],
   'neuralNetwork2__solver': ['adam', 'sgd', 'lbfgs'],
   'neuralNetwork2__alpha': [0.00001, 0.0001, 0.01]
}

# Combina el pipeline con el GridSearchCV
grid_search_neural = GridSearchCV(pipeline, param_grid_neural, cv=5, scoring='roc_auc')
grid_search_neural.fit(X_train, Y_train)
```

Los mejores parámetros encontrados fueron los siguientes:

- Tamaño de las capas ocultas: 20 neuronas.
- Función de activación: ReLU.
- Algoritmo de optimización: Adam.
- Tasa de regularización (alpha): 0.00001.

En cuanto a los resultados obtenidos, se observó una **precisión del modelo de 88.50% en el conjunto de prueba**, lo que indica que el modelo es capaz de generalizar bien para datos no vistos. Además, se calculó el AUC del modelo en el conjunto de prueba, obteniendo un valor del 95.25%, lo que sugiere una buena capacidad del modelo para distinguir entre las clases positiva y negativa.



Apartado 3: Red Neuronal vs Red Neuronal con Select K Best=4

	Red Neuronal	Red Neuronal con Select K Best=4
Accuracy	96.50%	88.50%
AUC	97.00%	95.25%
Tamaño de las capas ocultas	20	20
Función de activación	tanh	ReLU
Algoritmo de optimización	lbfgs	Adam
Alpha	0.01	0.00001

La precisión del modelo de Red Neuronal es notablemente superior a la del modelo de Red Neuronal con Select K Best=4, siendo su AUC también ligeramente mayor. Estas diferencias resaltan la influencia significativa de las variables seleccionadas en el rendimiento del modelo, así como la importancia de los parámetros de configuración. Por lo tanto, **el modelo ganador es el del primer apartado, la Red Neuronal.**

Apartado 4: Búsqueda Paramétrica para el Mejor Árbol de Decisión

En este apartado, se realizó una búsqueda paramétrica exhaustiva para encontrar el mejor modelo de árbol de decisión, evaluando su rendimiento según cuatro métodos diferentes de validación de la bondad de la clasificación. El objetivo fue determinar el árbol ganador.

Se evaluaron cuatro métodos de validación de la bondad de la clasificación:

- Accuracy
- Precision
- Recall
- Área bajo la Curva ROC (AUC)

```
# Crear el pipeline para el árbol de decisiones
tree_pipeline = Pipeline([
   ('scaler', StandardScaler()),
   ('tree', DecisionTreeClassifier(random_state=seed))
])

# Parámetros a probar para el árbol de decisiones
param_grid_tree = {
   'tree__criterion': ['gini', 'entropy'],
   'tree__max_depth': [None, 5, 10, 15],
   'tree__min_samples_split': [2, 5, 10],
   'tree__min_samples_leaf': [1, 2, 4, 6, 7]
}

# Métodos de validación de la bondad de la clasificación
scoring_methods = ['accuracy', 'precision', 'recall', 'roc_auc']
```

Los resultados obtenidos para cada método de validación en el conjunto de prueba fueron los siguientes:

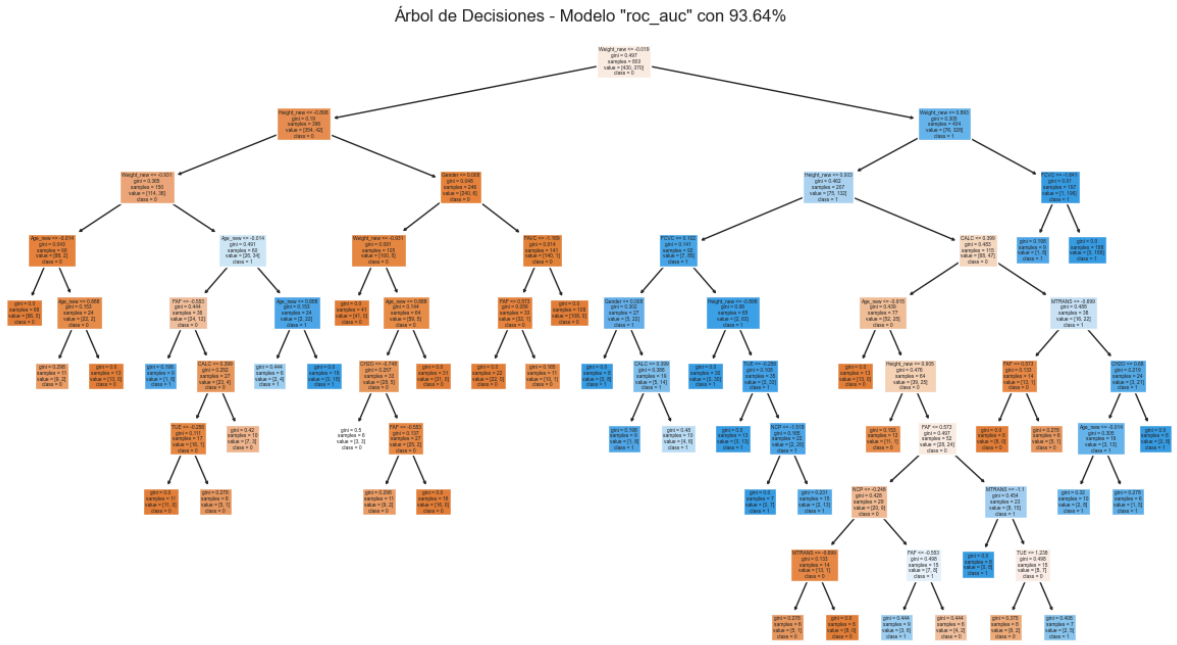
- Accuracy: 91.50%
- Precision: 88.89%
- Recall: 93.55%
- Área bajo la Curva ROC (AUC): 93.64%

El método que mostró el mejor rendimiento fue **AUC fue seleccionado como el ganador.**

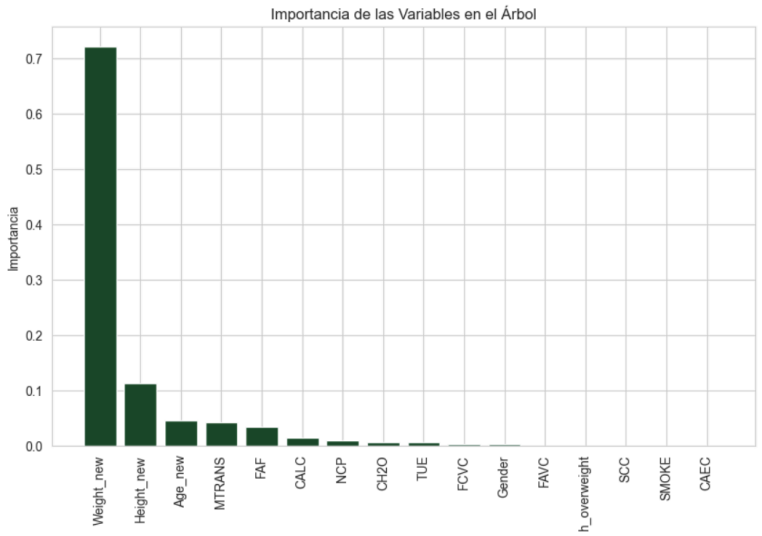
Modelo Ganador:

El modelo ganador corresponde al método de validación "Área bajo la Curva ROC (AUC)" con un valor del 93.64%. A continuación, se presenta el árbol de decisión correspondiente, junto con sus reglas en formato de texto y la importancia de sus variables.

Árbol de Decisión - Modelo "Área bajo la Curva ROC (AUC)



Importancia de las Variables en el Árbol



Reglas del Árbol

```
 |--- Weight_new <= -0.02
|   |--- Height_new <= -0.90
|   |   |--- Weight_new <= -0.93
|   |   |   |--- Age_new <= -0.01
|   |   |   |   |--- class: 0
|   |   |   |--- Age_new >  -0.01
|   |   |   |   |--- Age_new <= 0.89
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- Age_new >  0.89
|   |   |   |   |   |--- class: 0
|   |   |--- Weight_new >  -0.93
|   |   |   |--- Age_new <= -0.01
|   |   |   |   |--- FAF <= -0.55
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- FAF >  -0.55
|   |   |   |   |   |--- CALC <= 0.40
|   |   |   |   |   |   |--- TUE <= -0.26
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- TUE >  -0.26
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- CALC >  0.40
|   |   |   |   |   |--- class: 0
|   |   |   |--- Age_new >  -0.01
|   |   |   |   |--- Age_new <= 0.89
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- Age_new >  0.89
|   |   |   |   |   |--- class: 1
|   |--- Height_new >  -0.90
|   |   |--- Gender <= 0.01
|   |   |   |--- Weight_new <= -0.93
|   |   |   |   |--- class: 0
|   |   |   |--- Weight_new >  -0.93
|   |   |   |   |--- Age_new <= 0.89
|   |   |   |   |   |--- CH2O <= -0.75
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- CH2O >  -0.75
|   |   |   |   |   |   |--- FAF <= -0.55
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- FAF >  -0.55
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- Age_new >  0.89
|   |   |   |   |   |--- class: 0
|   |   |--- Gender >  0.01
|   |   |   |--- FAVC <= -1.17
|   |   |   |   |--- FAF <= 0.57
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- FAF >  0.57
|   |   |   |   |   |--- class: 0
|   |   |   |--- FAVC >  -1.17
|   |   |   |   |--- class: 0
|--- Weight_new >  -0.02
|   |--- Weight_new <= 0.89
|   |   |--- Height_new <= 0.00
|   |   |   |--- FCVC <= 0.10
|   |   |   |   |--- Gender <= 0.01
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- Gender >  0.01
|   |   |   |   |   |--- CALC <= 0.40
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- CALC >  0.40
|   |   |   |   |   |   |--- class: 1
|   |   |   |--- FCVC >  0.10
|   |   |   |   |--- Height_new <= -0.90
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- Height_new >  -0.90
|   |   |   |   |   |--- TUE <= -0.26
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- TUE >  -0.26
|   |   |   |   |   |   |--- NCP <= -1.52
```

```
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- NCP >  -1.52
|   |   |   |   |   |   |   |--- class: 1
|   |   |--- Height_new >  0.00
|   |   |   |--- CALC <= 0.40
|   |   |   |   |--- Age_new <= -0.92
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- Age_new >  -0.92
|   |   |   |   |   |--- Height_new <= 0.91
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- Height_new >  0.91
|   |   |   |   |   |   |--- FAF <= 0.57
|   |   |   |   |   |   |   |--- NCP <= -0.25
|   |   |   |   |   |   |   |   |--- MTRANS <= -0.70
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |--- MTRANS >  -0.70
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- NCP >  -0.25
|   |   |   |   |   |   |   |   |--- FAF <= -0.55
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- FAF >  -0.55
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- FAF >  0.57
|   |   |   |   |   |   |   |--- MTRANS <= -1.10
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- MTRANS >  -1.10
|   |   |   |   |   |   |   |   |--- TUE <= 1.24
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |--- TUE >  1.24
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |--- CALC >  0.40
|   |   |   |   |--- MTRANS <= -0.70
|   |   |   |   |   |--- FAF <= 0.57
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- FAF >  0.57
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- MTRANS >  -0.70
|   |   |   |   |   |--- CH2O <= 0.68
|   |   |   |   |   |   |--- Age_new <= -0.01
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- Age_new >  -0.01
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- CH2O >  0.68
|   |   |   |   |   |   |--- class: 1
|   |--- Weight_new >  0.89
|   |   |--- FCVC <= -1.64
|   |   |   |--- class: 1
|   |   |--- FCVC >  -1.64
|   |   |   |--- class: 1
```

Con esta evaluación detallada, se logra identificar el mejor árbol de decisión y comprender su estructura y características importantes para la clasificación.

Apartado 5: Búsqueda de Hiperparámetros para Bagging y Random Forest

Modelo de Bagging:

Se realizó una búsqueda paramétrica exhaustiva para encontrar el mejor modelo de Bagging con respecto a la métrica de precisión (Accuracy). Se probaron diversas combinaciones de hiperparámetros, y se evaluaron utilizando el conjunto de prueba.

```
# Parámetros a probar para Bagging
param_grid_bagging = {
   'n_estimators': [10, 50, 100],
   'max_samples': [0.5, 0.7, 1.0],
   'max_features': [0.5, 0.7, 1.0]
}
```

Se realizaron tres mejores modelos de Bagging encontrados junto con sus precisión y parámetros correspondientes, en anexos se encontraron los otros 2, acá solo se verá el que tiene mejor precisión

Modelo 1:

```
   Precisión: 97.50%
   Parámetros:
      base_estimator__ccp_alpha: 0.0
      base_estimator__class_weight: None
      base_estimator__criterion: gini
      base_estimator__max_depth: None
      base_estimator__max_features: None
      base_estimator__max_leaf_nodes: None
      base_estimator__min_impurity_decrease: 0.0
      base_estimator__min_samples_leaf: 1
      base_estimator__min_samples_split: 2
      base_estimator__min_weight_fraction_leaf: 0.0
      base_estimator__random_state: 2313
      base_estimator__splitter: best
      base_estimator: DecisionTreeClassifier(random_state=2313)
      bootstrap: True
      bootstrap_features: False
      estimator: None
      max_features: 0.5
      max_samples: 0.5
      n_estimators: 100
      n_jobs: None
      oob_score: False
      random_state: 2313
      verbose: 0
      warm_start: False
```

Modelo de Random Forest

De manera similar, se llevó a cabo una búsqueda de hiperparámetros para encontrar el mejor modelo de Random Forest en términos de precisión. Se evaluaron varias configuraciones de hiperparámetros utilizando el conjunto de prueba.

```
# Parámetros a probar para Random Forest
param_grid_rf = {
   'n_estimators': [50, 100, 150],
   'max_depth': [None, 5, 10],
```

```
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

Aquí se presenta solo el Modelo #1, mientras que los Modelos #2 y #3 estarán disponibles en los anexos. Es notable mencionar que tanto el Modelo #1 como el Modelo #2 lograron una precisión del 98% en los datos de prueba.

Modelo 1:

```
Precisión: 98.00%
Parámetros:
    bootstrap: True
    ccp_alpha: 0.0
    class_weight: None
    criterion: gini
    max_depth: None
    max_features: auto
    max_leaf_nodes: None
    max_samples: None
    min_impurity_decrease: 0.0
    min_samples_leaf: 2
    min_samples_split: 5
    min_weight_fraction_leaf: 0.0
    n_estimators: 50
    n_jobs: None
    oob_score: False
    random_state: 2313
    verbose: 0
    warm_start: False
```

Después de analizar los resultados de los diferentes modelos evaluados en este estudio, se observa que el Modelo de Random Forest (#1) sobresale con una precisión del 98% en los datos de prueba, superando ligeramente al Modelo de Bagging. Esta mayor precisión puede atribuirse a la capacidad del Random Forest para reducir la varianza y el sobreajuste mediante la combinación de múltiples árboles de decisión. Además, el Random Forest es capaz de capturar relaciones no lineales y gestionar eficazmente conjuntos de datos complejos, lo que lo convierte en una opción sólida para problemas de clasificación como el que estamos abordando. Por lo tanto, basado en estos resultados, se puede concluir que **el Modelo de Random Forest es la mejor opción** para predecir la variable objetivo en nuestro conjunto de datos.

Apartado 6: Búsqueda Paramétrica para Gradient Boosting y XG Boost

Modelo de Gradient Boosting

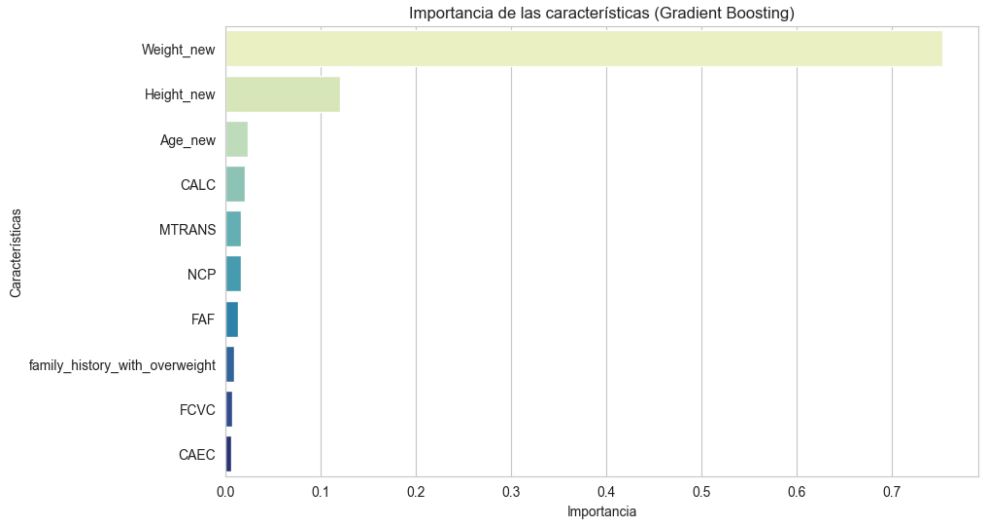
Se realizó una búsqueda paramétrica para determinar el mejor modelo de Gradient Boosting con respecto a la precisión. Los parámetros probados fueron:

```
# Parámetros a probar para Gradient Boosting
param_grid_gb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Crear el modelo de Gradient Boosting
gb_model = GradientBoostingClassifier(random_state=seed)

# Configurar el GridSearchCV
grid_search_gb = GridSearchCV(gb_model, param_grid_gb, cv=5, scoring='accuracy')

# Ajustar el modelo
grid_search_gb.fit(X_train, Y_train)
```

El mejor modelo de Gradient Boosting obtenido fue con los siguientes parámetros:

- Learning rate: 0.2
- Max depth: 3
- Min samples leaf: 4
- Min samples split: 10
- Número de estimadores: 50
- Subsample: 1.0

La precisión del modelo en el conjunto de prueba fue del 93.00%. A continuación, se muestra la importancia de las características en el mejor modelo de Gradient Boosting:



XG Boost según Accuracy

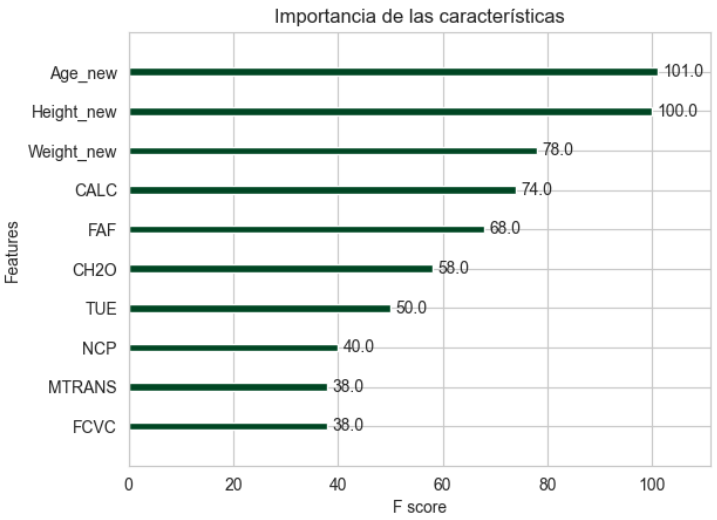
Se realizó una búsqueda paramétrica similar para el modelo XG Boost, utilizando los siguientes parámetros:

```
# Parámetros a probar para XGBoost
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.2]
}

# Crear el modelo de XGBoost
xgb_model = XGBClassifier(random_state=seed)

# Configurar el GridSearchCV
grid_search_xgb = GridSearchCV(xgb_model, param_grid_xgb, cv=5, scoring='accuracy')

# Ajustar el modelo
grid_search_xgb.fit(X_train, Y_train)
```

El mejor modelo de XG Boost obtenido fue con los siguientes parámetros:

- Gamma: 0.1
- Learning rate: 0.2
- Max depth: 7
- Min child weight: 3
- Número de estimadores: 100
- Subsample: 0.8

La precisión del modelo en el conjunto de prueba fue del 96.00%. A continuación, se muestra la importancia de las características en el mejor modelo de XG Boost:



Aunque ambos modelos lograron una precisión considerablemente alta, con Gradient Boosting alcanzando el 93.00% y XG Boost llegando al 96.00%, la diferencia de 3 puntos porcentuales en la precisión puede ser significativa dependiendo del contexto del problema. El modelo XG Boost pudo haber superado al Gradient Boosting debido a su capacidad para lidiar con problemas de clasificación más complejos mediante la optimización de los hiperparámetros, incluyendo la capacidad de manejar mejor la regularización y el sobreajuste. Además, el XG Boost puede ser más rápido de entrenar debido a su implementación optimizada y paralelización.

El modelo XG Boost sería preferible debido a su mayor precisión.

Apartado 7: Modelo SVM

En este apartado, se llevó a cabo una búsqueda paramétrica exhaustiva para determinar el mejor modelo de Support Vector Machine (SVM) utilizando al menos dos kernels diferentes. Los parámetros probados incluyeron la penalización del error de clasificación (C), el coeficiente de kernel (**γ**), y el tipo de kernel.

Para llevar a cabo esta búsqueda, se empleó la validación cruzada con 5 divisiones (5-fold cross-validation) y la métrica de precisión (accuracy) como criterio de evaluación. Los parámetros utilizados en la búsqueda paramétrica fueron los siguientes:

```
# Definir los parámetros a probar para SVM con diferentes kernels
param_grid = {
    'C': [0.1, 0.5, 1, 5, 10, 50],
    'gamma': [0.001, 0.01, 0.1, 1, 10],
```

```
    'kernel': ['rbf', 'linear', 'poly', 'sigmoid']
}
```

Se creó un modelo SVM utilizando la clase SVC de la biblioteca scikit-learn, y se configuró un objeto GridSearchCV para buscar los mejores parámetros dentro del espacio paramétrico definido.

Los resultados obtenidos para SVM fueron los siguientes:

- C: 10
- Gamma: 0.1
- Kernel: rbf

El modelo SVM óptimo, con los parámetros especificados, logró una precisión del 95.88% en el conjunto de prueba.

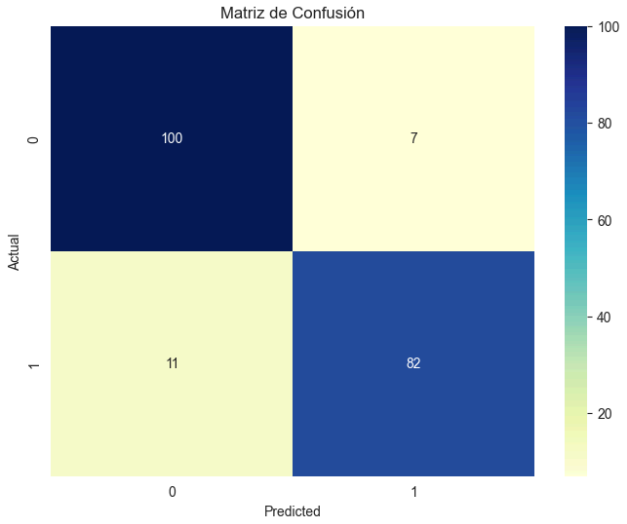
Apartado 8: Ensamblado de Bagging

En este apartado, se implementó un método de ensamblado de Bagging utilizando un clasificador base SVM en lugar de un árbol. El objetivo fue evaluar el rendimiento de un ensamblado de Bagging con un clasificador lineal.

```
# Define el pipeline con un escalador y el clasificador Bagging con SVM como
clasificador base
bagging_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('bagging', BaggingClassifier(base_estimator=SVC(kernel='linear'), n_estimators=10,
random_state=seed))
])

# Entrena el pipeline en los datos de entrenamiento
bagging_pipeline.fit(X_train, Y_train)
```

Se construyó un pipeline que consta de un escalador estándar y un clasificador Bagging con SVM como clasificador base. El clasificador Bagging se configuró con 10 estimadores y una semilla aleatoria para la reproducibilidad.

El modelo entrenado se evaluó utilizando el conjunto de prueba, y se calculó la precisión del modelo, que resultó ser del 91%. Además, se construyó y visualizó la matriz de confusión para evaluar el rendimiento del modelo en la clasificación de las clases.



Apartado 9: Método de Stacking

En este apartado, se implementó un método de Stacking utilizando varios algoritmos de entrada, incluyendo Regresión Logística, Random Forest, Máquinas de Vectores de Soporte (SVM) y Redes Neuronales. El modelo de ensamblaje seleccionado fue una Regresión Logística, que actuó como clasificador final en la composición del Stacking.

```
# Define los modelos base (algoritmos de entrada)
base_models = [
    ('logistic', LogisticRegression()),
    ('random_forest', RandomForestClassifier()),
    ('svm', SVC()),
    ('neural_network', MLPClassifier())
]

# Define el modelo de ensamblaje
stacking_model = StackingClassifier(
    estimators=base_models,
    final_estimator=LogisticRegression()
)

# Parámetros de búsqueda de cuadrícula para el modelo de Stacking
param_grid_stacking = {
    'final_estimator__C': [0.1, 1.0, 10.0],
    'final_estimator__penalty': ['l1', 'l2']
}

# Realizar la búsqueda de cuadrícula para el modelo de Stacking
grid_search_stacking = GridSearchCV(stacking_model, param_grid_stacking, cv=5)
grid_search_stacking.fit(X_train_scaled, Y_train)
```

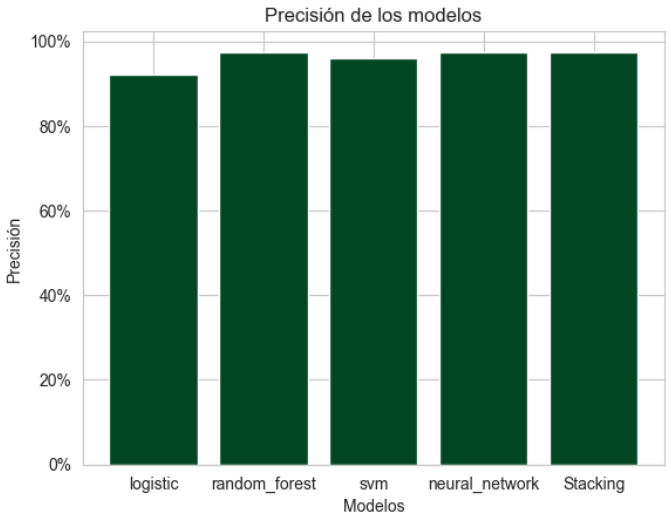
Los modelos base considerados fueron los siguientes:

- Regresión Logística
- Random Forest
- SVM
- Redes Neuronales

El modelo de ensamblaje Stacking se configuró con una Regresión Logística como clasificador final.

Se realizó una búsqueda de hiperparámetros para el modelo de Stacking utilizando la validación cruzada con 5 divisiones. Los parámetros probados incluyeron el parámetro de regularización C y el tipo de penalización (l1, l2) para el clasificador final (Regresión Logística). Después de la evaluación de los modelos base, se seleccionó el modelo de Random Forest como el clasificador base para el Stacking debido a su capacidad para ofrecer una mayor explicabilidad en sus predicciones.

Durante la evaluación de los modelos base, se observó que tanto el modelo de Random Forest como el de Redes Neuronales alcanzaron una precisión del 97.5% en el conjunto de prueba, superando ligeramente al resto de los modelos evaluados, que obtuvieron una precisión del 92% y 96%, respectivamente.



Si bien tanto los modelos de Random Forest como los de Redes Neuronales obtuvieron resultados similares en términos de precisión, se decidió seleccionar el modelo de Random Forest como el clasificador base para el Stacking.

Esta decisión se fundamenta en la capacidad del modelo de Random Forest para ofrecer una mayor explicabilidad en sus predicciones. A diferencia de las Redes Neuronales, que a menudo se consideran "cajas negras" debido a su complejidad y falta de interpretabilidad, el modelo de Random Forest permite analizar la importancia relativa de cada característica en el proceso de toma de decisiones.

Anexos:

Apartado 5: Modelo Bagging

Modelo 2:

```
Precisión: 97.50%
   Parámetros:
      base_estimator__ccp_alpha: 0.0
      base_estimator__class_weight: None
      base_estimator__criterion: gini
      base_estimator__max_depth: None
      base_estimator__max_features: None
      base_estimator__max_leaf_nodes: None
      base_estimator__min_impurity_decrease: 0.0
      base_estimator__min_samples_leaf: 1
      base_estimator__min_samples_split: 2
      base_estimator__min_weight_fraction_leaf: 0.0
      base_estimator__random_state: 2313
      base_estimator__splitter: best
      base_estimator: DecisionTreeClassifier(random_state=2313)
      bootstrap: True
      bootstrap_features: False
      estimator: None
      max_features: 0.5
      max_samples: 1.0
      n_estimators: 100
      n_jobs: None
      oob_score: False
      random_state: 2313
      verbose: 0
      warm_start: False
```

Modelo 3:

```
Precisión: 97.00%
   Parámetros:
      base_estimator__ccp_alpha: 0.0
      base_estimator__class_weight: None
      base_estimator__criterion: gini
      base_estimator__max_depth: None
      base_estimator__max_features: None
      base_estimator__max_leaf_nodes: None
      base_estimator__min_impurity_decrease: 0.0
      base_estimator__min_samples_leaf: 1
      base_estimator__min_samples_split: 2
      base_estimator__min_weight_fraction_leaf: 0.0
      base_estimator__random_state: 2313
      base_estimator__splitter: best
      base_estimator: DecisionTreeClassifier(random_state=2313)
      bootstrap: True
      bootstrap_features: False
      estimator: None
      max_features: 0.5
```

```
      max_samples: 0.7
      n_estimators: 50
      n_jobs: None
      oob_score: False
      random_state: 2313
      verbose: 0
      warm_start: False
```

Apartado 5: Modelo Random Forest

Modelo 2:

```
Precisión: 98.00%
   Parámetros:
      bootstrap: True
      ccp_alpha: 0.0
      class_weight: None
      criterion: gini
      max_depth: None
      max_features: sqrt
      max_leaf_nodes: None
      max_samples: None
      min_impurity_decrease: 0.0
      min_samples_leaf: 2
      min_samples_split: 5
      min_weight_fraction_leaf: 0.0
      n_estimators: 50
      n_jobs: None
      oob_score: False
      random_state: 2313
      verbose: 0
      warm_start: False
```

Modelo 3:

```
Precisión: 98.00%
   Parámetros:
      bootstrap: True
      ccp_alpha: 0.0
      class_weight: None
      criterion: gini
      max_depth: None
      max_features: log2
      max_leaf_nodes: None
      max_samples: None
      min_impurity_decrease: 0.0
      min_samples_leaf: 2
      min_samples_split: 5
      min_weight_fraction_leaf: 0.0
      n_estimators: 50
      n_jobs: None
      oob_score: False
```

```
    random_state: 2313
    verbose: 0
    warm_start: False
```