

# TP DE CLASIFICACIÓN

## *Introducción*

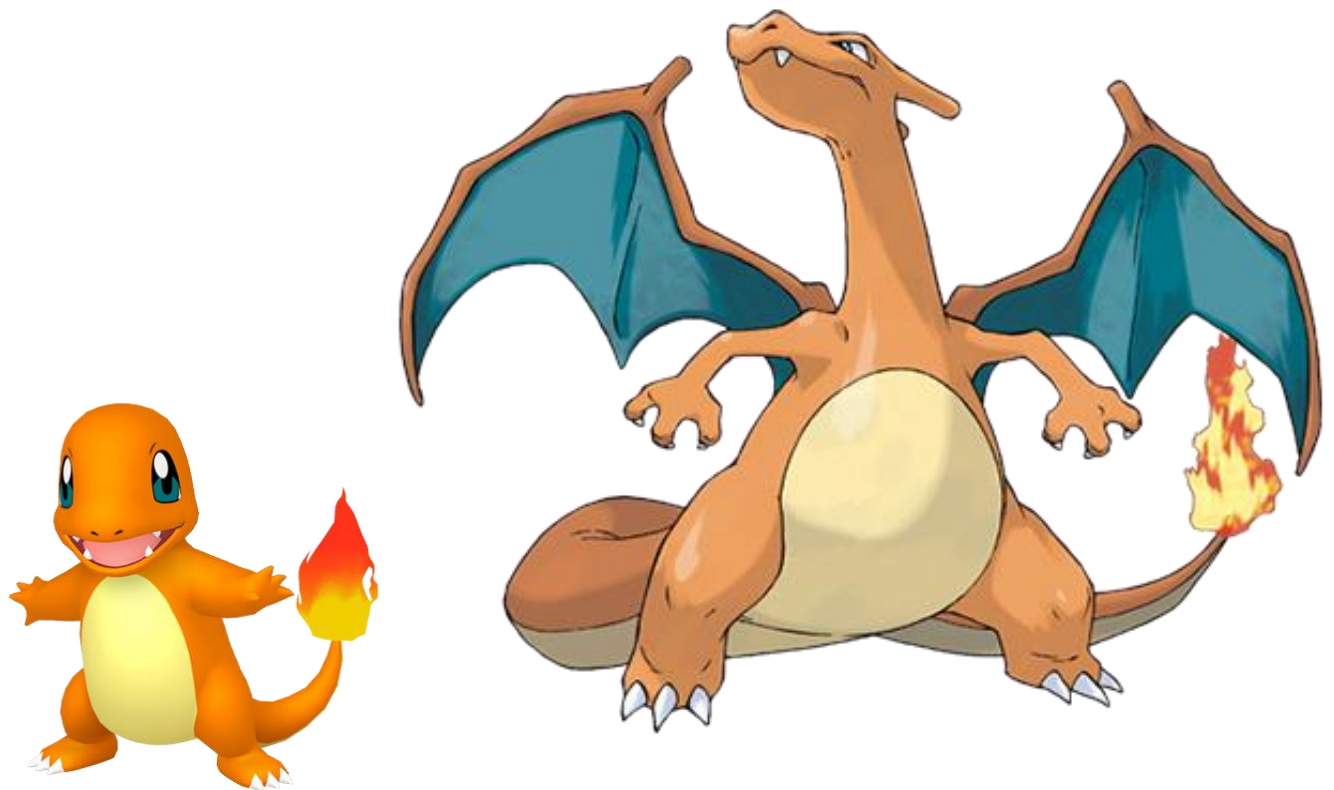
Utilizando los algoritmos de Clasificación y técnicas vistas, elegí tomar como dataset para este trabajo practico a los Pokémon y realizar mi análisis en función a sus diferentes atributos y características.

Los Pokémon y los datos que estos tienen son información que tienen un valor y significado muy importante para mí.

Representan una parte de mi infancia y adolescencia muy valerosa que me acompañó durante muchísimos años. Tengo un amplio conocimiento sobre esta temática y me entusiasma mucho hacer este análisis ya que jugaba sus videojuegos, veía sus películas y series e intercambiaba sus cartas.

Por esta razón decidí que mi trabajo sea en base a un dataset de Pokémon.

Existen diversas clases de criaturas en el mundo Pokémon, las hay de diferentes tamaños, poderes y colores. Están dotados de fuerzas y habilidades, listos para las batallas y con diferentes atributos de poderes y características.



Es importante saber que existen diferentes tipos de Pokémon, ya que pueden tener diferentes clasificaciones así sean de fuego, tierra, agua, psíquicos, etc.

Incluso está la posibilidad de que sean de dos tipos al mismo tiempo.

En total son 18 los tipos que existen, cada uno con sus particularidades, con sus características y sus propios puntajes.

Durante los juegos, los entrenadores realizan estrategias con diferentes tipos de Pokémon, ellos deben conocer estas características para aprovechar al máximo las fortalezas de sus Pokémon y enfrentar eficazmente a sus oponentes.



Otra clasificación importante para mencionar tiene que ver con las generaciones.

Pokémon tiene años en la industria de los juegos y el anime. Desde 1996 que se conoció la primera generación de estas criaturas con sus primeros 151 Pokémon, sumado a sus Mega evoluciones.

De ahí en más, cada cierto tiempo, se incorporan a la franquicia nuevas generaciones de Pokémon con una cierta cantidad de estos mismos. Como la franquicia sigue estando hasta el día de la fecha, es probable que sigan apareciendo nuevas generaciones futuras de estas fascinantes criaturas.

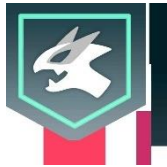


También hay otra clasificación importante que tienen los Pokémon y es que algunos tienen una rareza excepcional, a estos se los conocen como Pokémon Legendarios.

Los Pokémon legendarios son seres extraordinarios y misteriosos que habitan en el universo Pokémon. Son considerados como las criaturas más poderosas y raras de todas. Estos Pokémon, a diferencia de los demás, suelen tener una apariencia única y características especiales que los distinguen del resto.

Además, su historia y mitología suelen estar envueltas en leyendas y narrativas épicas. Se dice que estos Pokémon tienen un poder y una influencia significativos sobre elementos como el tiempo, el espacio, la naturaleza y otros aspectos trascendentales. Debido a su inmenso poder, los Pokémon legendarios son buscados por muchos entrenadores para intentar capturarlos y aprovechar su fuerza en batallas. Sin embargo, su rareza y la responsabilidad que conlleva su existencia los convierten en desafíos formidables. Los Pokémon legendarios a menudo encarnan valores y conceptos profundos, y su presencia en el mundo Pokémon suele estar relacionada con la protección y el equilibrio del universo. Su aparición y participación en la historia de los juegos y series animadas de Pokémon siempre generan emocionantes aventuras y momentos inolvidables para los entrenadores y fanáticos de la franquicia.

Símbolo Legendario.



## Análisis

Luego de esta breve introducción al mundo Pokémon, comenzaré a explicar acerca del análisis de investigación que realicé acerca de estas criaturas partiendo de un dataset con diferentes columnas y registros de datos.

El Dataset elegido muestra a un registro de 800 Pokémon organizados de manera ascendente por el ID de estos y con unas 15 columnas en donde se detallan sus características.

|    | A  | B | C | D | E | F | G | H | I |
|----|--|---|---|---|---|---|---|---|---|
| 1  | #,Name,Type 1,Type 2,Total,HP,Attack,Defense,Sp. Atk,Sp. Def,Speed,Generation,Legendary,Winbattle,Resistance |   |   |   |   |   |   |   |   |
| 2  | 1,Bulbasaur,Grass,Poison,318,45,49,49,65,65,45,1,False,35,450  |   |   |   |   |   |   |   |   |
| 3  | 2,Ivysaur,Grass,Poison,405,60,62,63,80,80,60,1,False,50,730  |   |   |   |   |   |   |   |   |
| 4  | 3,Venusaur,Grass,Poison,525,80,82,83,100,100,80,1,False,75,940   |   |   |   |   |   |   |   |   |
| 5  | 3,VenusaurMega Venusaur,Grass,Poison,625,80,100,123,122,120,80,1,False,86,1110                               |   |   |   |   |   |   |   |   |
| 6  | 4,Charmander,Fire,,309,39,52,43,60,50,65,1,False,40,500  |   |   |   |   |   |   |   |   |
| 7  | 5,Charmeleon,Fire,,405,58,64,58,80,65,80,1,False,60,800  |   |   |   |   |   |   |   |   |
| 8  | 6,Charizard,Fire,Flying,534,78,84,78,109,85,100,1,False,80,1000  |   |   |   |   |   |   |   |   |
| 9  | 6,CharizardMega Charizard X,Fire,Dragon,634,78,130,111,130,85,100,1,False,90,1150                            |   |   |   |   |   |   |   |   |
| 10 | 6,CharizardMega Charizard Y,Fire,Flying,634,78,104,78,159,115,100,1,False,90,1150                            |   |   |   |   |   |   |   |   |
| 11 | 7,Squirtle,Water,,314,44,48,65,50,64,43,1,False,35,450   |   |   |   |   |   |   |   |   |
| 12 | 8,Wartortle,Water,,405,59,63,80,65,80,58,1,False,50,730  |   |   |   |   |   |   |   |   |
| 13 | 9,Blastoise,Water,,530,79,83,100,85,105,78,1,False,75,940  |   |   |   |   |   |   |   |   |
| 14 | 9,BlastoiseMega Blastoise,Water,,630,79,103,120,135,115,78,1,False,86,1110                                   |   |   |   |   |   |   |   |   |
| 15 | 10,Caterpie,Bug,,195,45,30,35,20,20,45,1,False,5,83  |   |   |   |   |   |   |   |   |
| 16 | 11,Metapod,Bug,,205,50,20,55,25,25,30,1,False,25,379   |   |   |   |   |   |   |   |   |
| 17 | 12,Butterfree,Bug,Flying,395,60,45,50,90,80,70,1,False,45,688  |   |   |   |   |   |   |   |   |
| 18 | 13,Weedle,Bug,Poison,195,40,35,30,20,20,50,1,False,5,67  |   |   |   |   |   |   |   |   |
| 19 | 14,Kakuna,Bug,Poison,205,45,25,50,25,25,35,1,False,25,362  |   |   |   |   |   |   |   |   |
| 20 | 15,Beedrill,Bug,Poison,395,65,90,40,45,80,75,1,False,45,671  |   |   |   |   |   |   |   |   |
| 21 | 15,BeedrillMega Beedrill,Bug,Poison,495,65,150,40,15,80,145,1,False,65,823                                   |   |   |   |   |   |   |   |   |
| 22 | 16,Pidgey,Normal,Flying,251,40,45,40,35,35,56,1,False,10,118   |   |   |   |   |   |   |   |   |
| 23 | 17,Pidgeotto,Normal,Flying,349,63,60,55,50,50,71,1,False,30,411  |   |   |   |   |   |   |   |   |
| 24 | 18,Pidgeot,Normal,Flying,479,83,80,75,70,70,101,1,False,50,721   |   |   |   |   |   |   |   |   |
| 25 | 18,PidgeotMega Pidgeot,Normal,Flying,579,83,80,80,135,80,121,1,False,65,815                                  |   |   |   |   |   |   |   |   |
| 26 | 19,Rattata,Normal,,253,30,56,35,25,35,72,1,False,12,131  |   |   |   |   |   |   |   |   |
| 27 | 20,Raticate,Normal,,413,55,81,60,50,70,97,1,False,30,399   |   |   |   |   |   |   |   |   |
| 28 | 21,Spearow,Normal,Flying,262,40,60,30,31,31,70,1,False,15,201  |   |   |   |   |   |   |   |   |
| 29 | 22,Fearow,Normal,Flying,442,65,90,65,61,61,100,1,False,30,412  |   |   |   |   |   |   |   |   |

Primeramente, lo que hice fue crear un archivo tpPokemonNew con extensión ".ipynb".

Esta extensión hace referencia a un archivo en formato Jupyter Notebook, la cual permite crear y compartir documentos interactivos que contienen código en vivo, visualizaciones, texto explicativo y otros elementos multimedia.

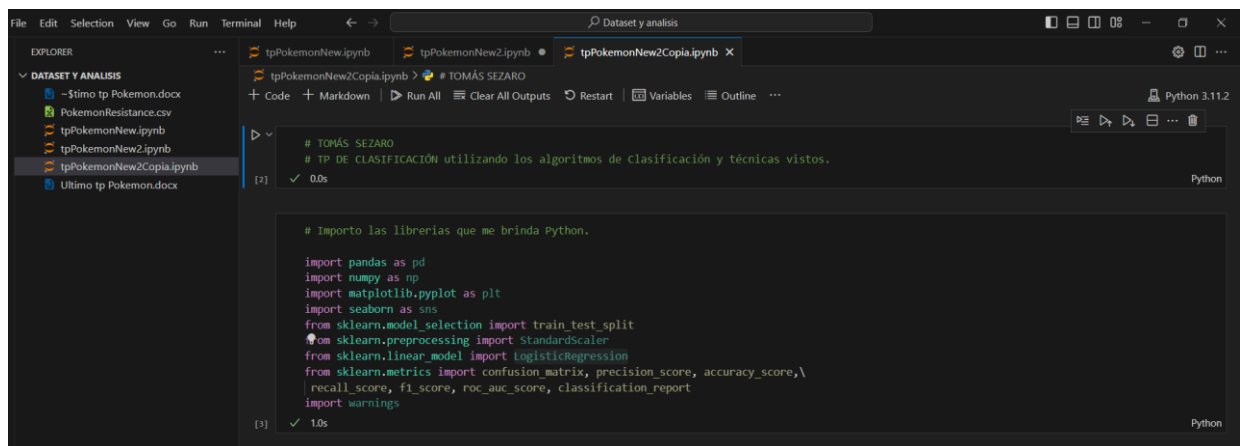
Los archivos con extensión ".ipynb" contienen el contenido y la estructura de un cuaderno de Jupyter.

Un cuaderno de Jupyter consiste en una serie de celdas que pueden contener código en lenguajes como Python, R, Julia, entre otros, así como texto enriquecido, ecuaciones matemáticas, gráficos y más. Las celdas pueden ejecutarse individualmente, lo que permite una ejecución interactiva del código y una visualización instantánea de los resultados.

Luego, dependiendo para cada modelo de clasificación, comienzo a importar las librerías que voy a utilizar para este proyecto como StandardScaler, pandas, numpy, matplotlib, seaborn, LogisticRegression, entre otras.

Estas librerías son herramientas fundamentales en el ecosistema de Python para el análisis de datos y la visualización. Cada una de estas librerías ofrece funcionalidades específicas que son esenciales en el procesamiento y análisis de datos.

Por supuesto, previamente se requiere la instalación de las dependencias de estas librerías por medio del Pip, esta es una herramienta estándar en Python que facilita la instalación de paquetes y librerías adicionales.



```
# TOMÁS SEZARO
# TP DE CLASIFICACIÓN utilizando los algoritmos de Clasificación y técnicas vistos.

# Importo las librerías que me brinda Python.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, \
recall_score, f1_score, roc_auc_score, classification_report
import warnings
```

En este dataset, los Pokémon están representados por un Id en la primera columna y tienen un nombre propio característico en la segunda.

En la Columna 3 se encuentra la clasificación de Pokémon por tipo, hay muchos tipos de Pokémon como puede ser de agua, fuego, eléctrico, etc.

Al mismo tiempo un Pokémon puede tener un segundo tipo, ya que puede ser volador y de fuego, por ejemplo, o solamente tener un tipo solo y esta cuarta columna quedar en null.

La quinta columna representa el valor numérico total del Pokémon.

De la 6ta a la 11va columna se encuentran los valores numéricos de diferentes características, como son la salud, el ataque, la defensa, la velocidad de ataque, la velocidad de defensa, y la velocidad en general.

La suma de todas estas columnas nos da como resultado el valor total de la columna 5.

La columna 12 representa la generación de Pokémon, ya que a medida que pasan los años, se descubren nuevos Pokémon, y ellos se categorizan en generaciones comenzando desde la primera hasta sexta por el momento.

La columna 13 habla de la cuestión de que un Pokémon pueda ser legendario, ya que algunos que, si lo son, los cuales representan una rareza absoluta.

La columna 14 es Winbattle y tiene que ver con un valor numérico que representa las probabilidades de ganar una batalla.

La columna 15 es el valor de Resistencia que tiene el Pokémon.

| #   | Name                  | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | Winbattle | Resistance |
|-----|-----------------------|--------|--------|-------|----|--------|---------|---------|---------|-------|------------|-----------|-----------|------------|
| 0 1 | Bulbasaur             | Grass  | Poison | 318   | 45 | 49     | 49      | 65      | 65      | 45    | 1          | False     | 35        | 450        |
| 1 2 | Ivysaur               | Grass  | Poison | 405   | 60 | 62     | 63      | 80      | 80      | 60    | 1          | False     | 50        | 730        |
| 2 3 | Venusaur              | Grass  | Poison | 525   | 80 | 82     | 83      | 100     | 100     | 80    | 1          | False     | 75        | 940        |
| 3 3 | VenusaurMega Venusaur | Grass  | Poison | 625   | 80 | 100    | 123     | 122     | 120     | 80    | 1          | False     | 86        | 1110       |
| 4 4 | Charmander            | Fire   | NaN    | 309   | 39 | 52     | 43      | 60      | 50      | 65    | 1          | False     | 40        | 500        |



## Regresión Logística - Clasificación

A diferencia de la regresión lineal donde se predice un valor continuo, se utiliza la regresión logística se utiliza para predecir una variable categórica binaria o multiclase.

En lugar de predecir valores numéricos como en la regresión lineal, la regresión logística estima la probabilidad de que una instancia pertenezca a una determinada clase.

Esto se logra utilizando una función logística, también conocida como función sigmoide, que transforma una combinación lineal de las variables de entrada en un valor comprendido entre 0 y 1.

Importo las librerías que me brinda Python.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, \
recall_score, f1_score, roc_auc_score, classification_report
import warnings
```

✓ 1.0s

Luego realizo la importación de los datos.

Ignoro las advertencias generadas por el módulo warnings en Python.

```
# Realizo la importación de los datos.
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()
```

[3] ✓ 0.1s

| # | Name | Type 1                | Type 2 | Total  | HP  | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | Winbattle | Resistance |      |
|---|------|-----------------------|--------|--------|-----|--------|---------|---------|---------|-------|------------|-----------|-----------|------------|------|
| 0 | 1    | Bulbasaur             | Grass  | Poison | 318 | 45     | 49      | 49      | 65      | 65    | 45         | 1         | False     | 35         | 450  |
| 1 | 2    | Ivysaur               | Grass  | Poison | 405 | 60     | 62      | 63      | 80      | 80    | 60         | 1         | False     | 50         | 730  |
| 2 | 3    | Venusaur              | Grass  | Poison | 525 | 80     | 82      | 83      | 100     | 100   | 80         | 1         | False     | 75         | 940  |
| 3 | 3    | VenusaurMega Venusaur | Grass  | Poison | 625 | 80     | 100     | 123     | 122     | 120   | 80         | 1         | False     | 86         | 1110 |
| 4 | 4    | Charmander            | Fire   | NaN    | 309 | 39     | 52      | 43      | 60      | 50    | 65         | 1         | False     | 40         | 500  |

Para este análisis tomaré los valores de Total y Resistance para la variable X independiente. También tomaré los valores de Legendary para la variable Y dependiente.

Consulto si hay algún valor null (NaN), me devolverá un booleano que en caso de ser true es porque si hay valores Nan.

```
df.isnull().values.any()
```

✓ 0.0s

True

Obtuve un valor True por lo que es necesario verificar en donde se encuentran los valores nulos.

Averiguo en que columnas hay valores null.

```
df.isnull().any()
```

[7] ✓ 0.0s

|            |       |
|------------|-------|
| #          | False |
| Name       | False |
| Type 1     | False |
| Type 2     | True  |
| Total      | False |
| HP         | False |
| Attack     | False |
| Defense    | False |
| Sp. Atk    | False |
| Sp. Def    | False |
| Speed      | False |
| Generation | False |
| Legendary  | False |
| Winbattle  | False |
| Resistance | False |
| dtype:     | bool  |

Constato que únicamente hay valores null en la columna Type2, por lo que no hay inconvenientes en tomar los datos de Total, Resistance y Legendary.



Coloco los datos de las columnas Total y Resistance en la variable X.

Coloco los datos de la columna Legendary en la variable y.

```
x = df.iloc[:, [4,14]].values
y = df.iloc[:, 12].values
X,y
```

[9] ✓ 0.0s

```
... (array([[ 318, 450],
[ 405, 730],
[ 525, 940],
...,
[ 600, 1095],
[ 680, 1135],
[ 600, 1030]], dtype=int64),
array([False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, True, True, True, False, False, False,
...,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
```

Utilizo LabelEncoder para convertir a números los datos categóricos de la columna Legendary almacenados en la variable y.

[illegible]



Ajusto el modelo de clasificación con el conjunto de entrenamiento.

Esto se hace para hacer predicciones en los datos de prueba y mostrar una tabla que compara las etiquetas reales con las etiquetas predichas para una muestra de prueba limitada.

```

classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).head(100)

```

✓ 0.0s

|     | Actual | Predicción |
|-----|--------|------------|
| 0   | 0      | 0          |
| 1   | 0      | 0          |
| 2   | 0      | 0          |
| 3   | 0      | 0          |
| 4   | 0      | 0          |
| ... | ...    | ...        |
| 95  | 0      | 1          |
| 96  | 0      | 0          |
| 97  | 1      | 0          |
| 98  | 0      | 0          |
| 99  | 0      | 0          |

100 rows × 2 columns

Construyo la matriz de confusión, la cual es una técnica muy potente.

Se calcula sobre el conjunto de testing y se puede observar si las predicciones que ha realizado el algoritmo son potentes

```

cm = confusion_matrix(y_test, y_pred)
cm

```

✓ 0.0s

```

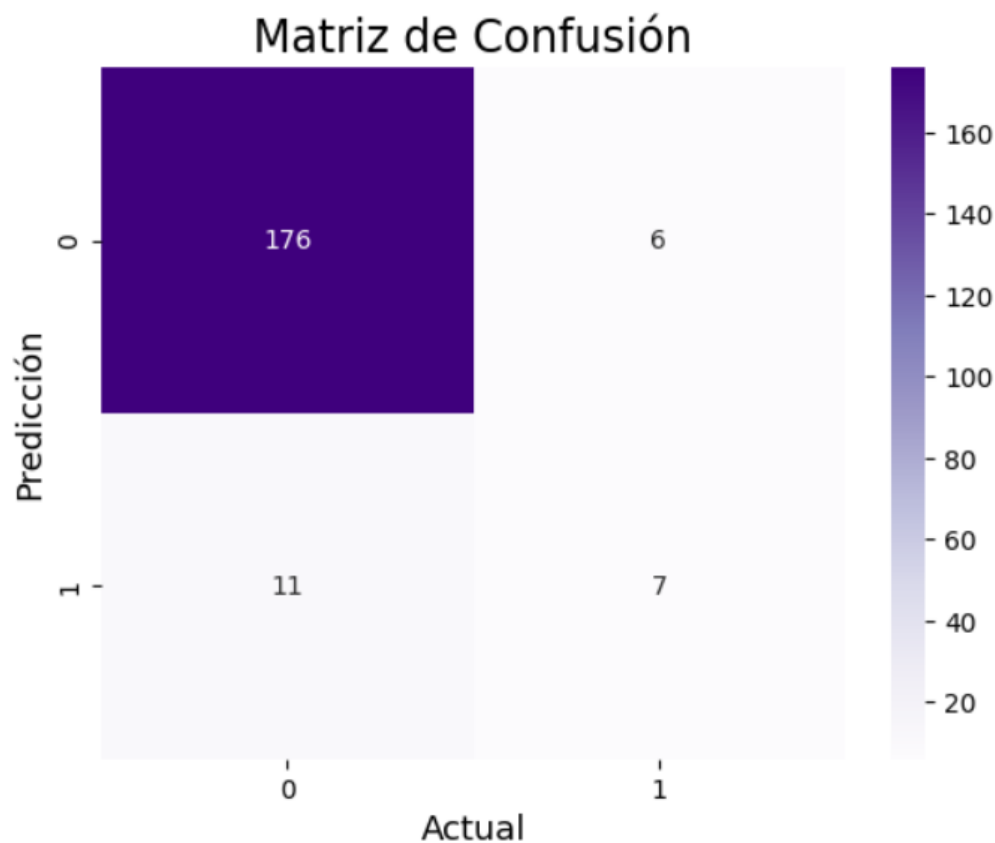
array([[176,  6],
       [ 11,  7]], dtype=int64)

```

Visualizo la matriz de confusión de un modelo de clasificación utilizando la biblioteca Seaborn y Matplotlib en Python.

La matriz de confusión habla de qué tan bien ha evaluado el algoritmo la clasificación, en este caso, de los pokemones legendarios o no legendarios con respecto a las propiedades de Total y Resitence.

```
sns.heatmap(cm,  
    annot=True,  
    fmt='g',  
    cmap='Purples')  
plt.ylabel('Predicción',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Matriz de Confusión',fontsize=17)  
plt.show()  
✓ 0.1s
```



Las predicciones correctas son los valores numéricos que se encuentran en el cuadrado superior a la izquierda junto con las que se encuentran en el cuadrado inferior a la derecha.

Mientras que las predicciones incorrectas son los valores numéricos que se encuentran en el cuadrado superior a la derecha junto con las que se encuentran en el cuadrado inferior a la izquierda.

#### OBESERVACIÓN

El cuadrado superior izquierdo con un valor de 176 representa la cantidad de instancias clasificadas correctamente como clase 0.

El cuadrado superior derecho con un valor de 6 representa la cantidad de instancias clasificadas incorrectamente como clase 1 cuando en realidad son de clase 0.

El cuadrado inferior izquierdo con un valor de 11 representa la cantidad de instancias clasificadas incorrectamente como clase 0 cuando en realidad son de clase 1.

El cuadrado inferior derecho con un valor de 7 representa la cantidad de instancias clasificadas correctamente como clase 1.

Noto que hay un gran número de predicciones calificadas como correctas, 183 aplicando la suma de los valores correctos, a diferencia de los 17 casos incorrectos.

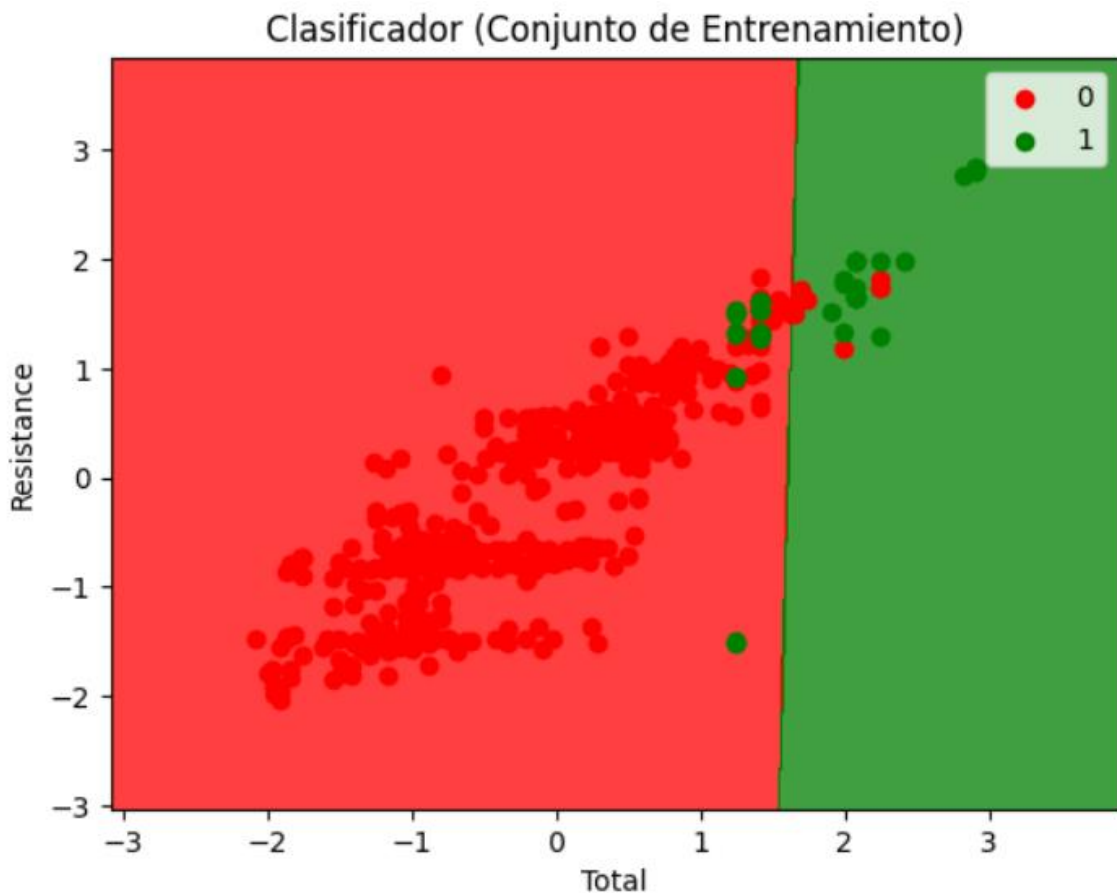
Además, en el caso del cuadrado superior izquierdo, clase 0, que son las predicciones correctas, presenta un color purpura intenso a diferencia de los demás.

Esto podría considerarlo como un buen primer indicio del modelo.

Visualización del modelo de clasificación en el conjunto de Entrenamiento.

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Clasificador (Conjunto de Entrenamiento)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
```

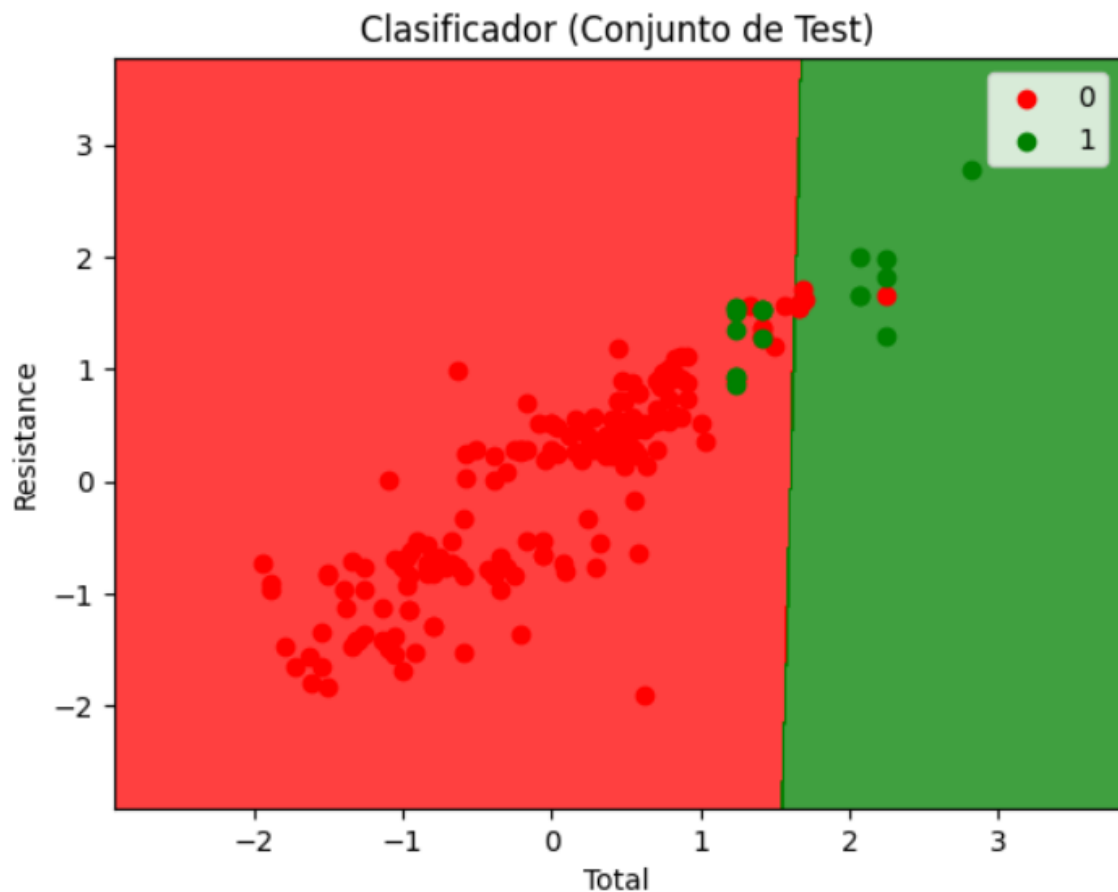
✓ 0.2s



Visualización del modelo de clasificación en el conjunto de Test.

```
x_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Clasificador (Conjunto de Test)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
```

✓ 0.1s





Ahora analizaré diferentes métricas de evaluación del rendimiento de un modelo de clasificación.

**Precisión del modelo:** La precisión mide la proporción de instancias positivas que fueron correctamente clasificadas como positivas.

**Exactitud del modelo:** La exactitud es la proporción total de predicciones correctas realizadas por el modelo.

**Sensibilidad del modelo:** La sensibilidad, también conocida como tasa de verdaderos positivos o recall, mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo.

**Puntaje F1 del modelo:** El puntaje F1 es una medida que combina la precisión y la sensibilidad en un solo valor. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad.

**Curva ROC - AUC del modelo:** La curva ROC, y el área bajo la curva (AUC) son métricas utilizadas para evaluar el rendimiento de un modelo de clasificación en problemas de clasificación binaria. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de clasificación. El AUC mide la capacidad del modelo para distinguir entre las clases.

```
print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
print(f'Exactitud del modelo: {accuracy_score(y_test, y_pred):.2f}')
print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
print(f'Puntaje F1 del modelo: {f1_score(y_test, y_pred):.2f}')
print(f'Curva ROC - AUC del modelo: {roc_auc_score(y_test, y_pred):.2f}')
```

✓ 0.0s

```
Precisión del modelo: 0.54
Exactitud del modelo: 0.92
Sensibilidad del modelo: 0.39
Puntaje F1 del modelo: 0.45
Curva ROC - AUC del modelo: 0.68
```

Imprimo las métricas de evaluación del modelo proporcionan información adicional sobre el rendimiento del modelo en términos de precisión, exactitud, sensibilidad, puntaje F1 y el área bajo la curva ROC-AUC.

```
print(classification_report(y_test, y_pred))
```

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.94      | 0.97   | 0.95     | 182     |
| 1            | 0.54      | 0.39   | 0.45     | 18      |
| accuracy     |           |        | 0.92     | 200     |
| macro avg    | 0.74      | 0.68   | 0.70     | 200     |
| weighted avg | 0.90      | 0.92   | 0.91     | 200     |

### Definiciones de otros datos

Macro avg: Calcula el promedio sin tener en cuenta el desequilibrio en el número de muestras de cada clase.

Para cada métrica, se calcula la media de los valores para todas las clases y se informa como el promedio macro avg.

Se trata de un promedio no ponderado en el que todas las clases tienen el mismo peso.

Este enfoque considera que todas las clases son igualmente importantes y le asigna la misma importancia a cada clase al calcular el promedio.

Weighted avg: Calcula el promedio ponderado teniendo en cuenta el número de muestras en cada clase.

Para cada métrica, se calcula la media ponderada de los valores para todas las clases y se informa como el promedio weighted avg.

En este enfoque, se asigna un peso proporcional al número de muestras de cada clase.

Las clases con más muestras tienen un impacto mayor en el cálculo del promedio.

Este enfoque es útil cuando hay un desequilibrio significativo en el número de muestras entre las clases.

Para la conclusión me centraré en los valores específicos obtenidos de cada clase en lugar de los promedios macro avg y weighted avg.

Los valores específicos de cada clase proporcionan información más detallada y relevante sobre cómo el modelo está clasificando cada clase individualmente.

El enfoque en los valores de cada clase es considerablemente más importante cuando hay un desequilibrio significativo en el número de muestras entre las clases.

En estos casos, los promedios pueden verse afectados por la clase dominante y no reflejar adecuadamente el rendimiento del modelo en las clases minoritarias.

Por lo tanto, los valores específicos de cada clase ofrecen una visión más detallada y proporcionan mayor eficiencia en las conclusiones del rendimiento del modelo de clasificación.

## Conclusión

Dado los resultados obtenidos con las diferentes métricas e informes puedo sacar las siguientes conclusiones adicionales.

El modelo tiene una precisión alta para la clase 0 con un 94% lo cual es algo altamente positivo.

Sin embargo, con respecto a la clase 1 el porcentaje de precisión baja considerablemente teniendo apenas un 54% de precisión.

La exactitud o accuracy, devolvió un porcentaje muy elevado del 92% lo que significa que un muy alto número de instancias se clasificaron correctamente en general sobre el total de instancias.

En general, un mayor valor de exactitud o accuracy indica un mejor rendimiento del modelo, pero también puede ser engañoso en casos de desequilibrio de clases, donde una clase dominante puede influir en el resultado.

Por lo tanto, es un dato que hay que tomarlo con cierto cuidado.

Otro aspecto a tener en cuenta es que la sensibilidad con respecto a la clase 0 es del 97%, por lo que el modelo es capaz de identificar de manera confiable la mayoría de las instancias de esa clase.

Aunque el inconveniente vuelve a surgir en la clase 1, en donde la sensibilidad desciende hasta el valor de 39% lo que es considerablemente bajo.

El Puntaje F1, que es una medida que combina precisión y sensibilidad para evaluar el equilibrio entre ambas, vuelve arrojar una diferenciación muy marcada, 95% para la clase 0 lo cual es muy bueno, pero 45% para la clase 1 siendo un porcentaje bajo.

Finalmente, el porcentaje arrojado por Curva ROC - AUC del modelo es del 68%, el cual es un valor por arriba del 50% (50% indica un modelo que clasifica las instancias de manera aleatoria), aunque lo esperado sería un valor aún más alto del modelo para separar las clases positivas y negativas y evitando cometer errores.

Mi conclusión de este modelo de clasificación de Regresión Logística, para este caso particular, es que posee un rendimiento regular, ya que, pese a tener un buen porcentaje de exactitud y que también presenta buenos valores para la clase 0, se encuentran inconvenientes cuando tiene que clasificar a la clase 1, dado a los porcentajes obtenidos se concluye que no lo hace de forma eficiente ni confiable.

## K-Nearest Neighbors

Es un algoritmo de aprendizaje automático supervisado, parte de tener una serie de datos categorizados.

Para realizar una predicción, se toma una instancia de prueba con características desconocidas y se calcula su similitud con las instancias de entrenamiento.

Se elige el número K de vecinos que se quiere tener en cuenta para el proceso de clasificación.

Una vez elegido se toman los K vecinos más cercanos del nuevo dato utilizando la distancia euclídea. E

Una vez seleccionados esos K vecinos más cercanos, se cuentan el número de puntos que pertenecen a cada una de las categorías.

Finalmente, se asigna el punto a la categoría luego de aplicar la regla del voto por mayoría.

Importo las librerías que me brinda Python.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, \
    recall_score, f1_score, roc_auc_score, classification_report
from matplotlib.colors import ListedColormap
import warnings
✓ 0.0s
```

Realizo la importación de los datos.

Ignoro las advertencias generadas por el módulo warnings en Python.

```
warnings.filterwarnings("ignore")
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()
```

| #   | Name                  | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | Winbattle | Resistance |
|-----|-----------------------|--------|--------|-------|----|--------|---------|---------|---------|-------|------------|-----------|-----------|------------|
| 0 1 | Bulbasaur             | Grass  | Poison | 318   | 45 | 49     | 49      | 65      | 65      | 45    | 1          | False     | 35        | 450        |
| 1 2 | Ivysaur               | Grass  | Poison | 405   | 60 | 62     | 63      | 80      | 80      | 60    | 1          | False     | 50        | 730        |
| 2 3 | Venusaur              | Grass  | Poison | 525   | 80 | 82     | 83      | 100     | 100     | 80    | 1          | False     | 75        | 940        |
| 3 3 | VenusaurMega Venusaur | Grass  | Poison | 625   | 80 | 100    | 123     | 122     | 120     | 80    | 1          | False     | 86        | 1110       |
| 4 4 | Charmander            | Fire   | NaN    | 309   | 39 | 52     | 43      | 60      | 50      | 65    | 1          | False     | 40        | 500        |

Para este análisis tomaré los valores de Total y Resistance para la variable X independiente. También tomaré los valores de Legendary para la variable Y dependiente.

Consulto si hay algún valor null (NaN), me devolverá un booleano que en caso de ser true es porque si hay valores Nan.

```
df.isnull().values.any()
✓ 0.0s
True
```

Obtuve un valor True por lo que es necesario verificar en donde se encuentran los valores nulos.

Averiguo en que columnas hay valores null.

```
df.isnull().any()
[7] ✓ 0.0s
```

|     |            |       |
|-----|------------|-------|
| ... | #          | False |
|     | Name       | False |
|     | Type 1     | False |
|     | Type 2     | True  |
|     | Total      | False |
|     | HP         | False |
|     | Attack     | False |
|     | Defense    | False |
|     | Sp. Atk    | False |
|     | Sp. Def    | False |
|     | Speed      | False |
|     | Generation | False |
|     | Legendary  | False |
|     | winbattle  | False |
|     | Resistance | False |
|     | dtype:     | bool  |

Constato que únicamente hay valores null en la columna Type2, por lo que no hay inconvenientes en tomar los datos de Total, Resistance y Legendary.

Coloco los datos de las columnas Total y Resistance en la variable X.

Coloco los datos de la columna Legendary en la variable y.

```
x = df.iloc[:, [4,14]].values
y = df.iloc[:, 12].values
X,y
```

```
[9] ✓ 0.0s
```

```
... (array([[ 318, 450],
[ 405, 730],
[ 525, 940],
...,
[ 600, 1095],
[ 680, 1135],
[ 600, 1030]], dtype=int64),
array([False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, True, True, True, False, False, False,
...,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
```

Utilizo LabelEncoder para convertir a números los datos categóricos de la columna Legendary almacenados en la variable y.

[illegible]

Imprimo nuevamente mis variables con sus datos.

[illegible]

Utilizo `train_test_split` para dividir un dataset en bloques.

Conjunto de testing por un lado y conjunto de entrenamiento por el otro.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

[12] ✓ 0.0s

## Escales los datos (variables)

El escalado va a transformar los valores de las características de forma que estén confinados en un rango  $[a, b]$ , típicamente  $[0, 1]$  o  $[-1, 1]$ .

```
sc_X = StandardScaler()  
X_train = sc_X.fit_transform(X_train)  
X_test = sc_X.transform(X_test)
```

✓ 0.0s



Ajusto el modelo de clasificación con el conjunto de entrenamiento.

Esto se hace para hacer predicciones en los datos de prueba y mostrar una tabla que compara las etiquetas reales con las etiquetas predichas para una muestra de prueba limitada.

$p=2$  es la Distancia euclidiana

```

classifier = KNeighborsClassifier(n_neighbors = 5, metric = "minkowski", p = 2)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).head(100)

```

✓ 0.0s

|     | Actual | Predicción |
|-----|--------|------------|
| 0   | 0      | 0          |
| 1   | 0      | 0          |
| 2   | 0      | 0          |
| 3   | 0      | 0          |
| 4   | 0      | 0          |
| ... | ...    | ...        |
| 95  | 0      | 0          |
| 96  | 0      | 0          |
| 97  | 1      | 1          |
| 98  | 0      | 0          |
| 99  | 0      | 0          |

100 rows × 2 columns

Construyo la matriz de confusión, la cual es una técnica muy potente.

Se calcula sobre el conjunto de testing y se puede observar si las predicciones que ha realizado el algoritmo son potentes

```

cm = confusion_matrix(y_test, y_pred)
cm

```

✓ 0.0s

```

array([[174,  8],
       [ 2, 16]], dtype=int64)

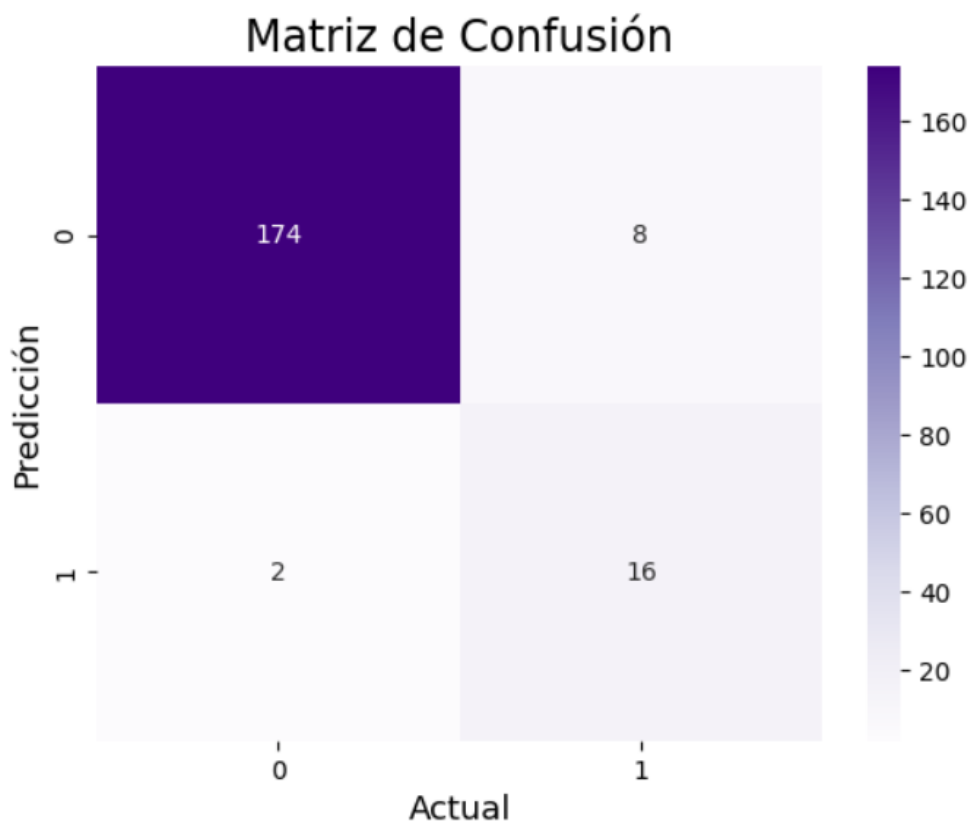
```

Visualizo la matriz de confusión de un modelo de clasificación utilizando la biblioteca Seaborn y Matplotlib en Python.

La matriz de confusión habla de qué tan bien ha evaluado el algoritmo la clasificación, en este caso, de los pokemones legendarios o no legendarios con respecto a las propiedades de Total y Resitence.

```
sns.heatmap(cm,  
            annot=True,  
            fmt='g',  
            cmap='Purples')  
plt.ylabel('Predicción',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Matriz de Confusión',fontsize=17)  
plt.show()
```

✓ 0.1s



Las predicciones correctas son los valores numéricos que se encuentran en el cuadrado superior a la izquierda junto con las que se encuentran en el cuadrado inferior a la derecha.

Mientras que las predicciones incorrectas son los valores numéricos que se encuentran en el cuadrado superior a la derecha junto con las que se encuentran en el cuadrado inferior a la izquierda.

#### OBESERVACIÓN

El cuadrado superior izquierdo con un valor de 174 representa la cantidad de instancias clasificadas correctamente como clase 0.

El cuadrado superior derecho con un valor de 8 representa la cantidad de instancias clasificadas incorrectamente como clase 1 cuando en realidad son de clase 0.

El cuadrado inferior izquierdo con un valor de 2 representa la cantidad de instancias clasificadas incorrectamente como clase 0 cuando en realidad son de clase 1.

El cuadrado inferior derecho con un valor de 16 representa la cantidad de instancias clasificadas correctamente como clase 1.

Noto que hay un gran número de predicciones calificadas como correctas, 190 aplicando la suma de los valores correctos, a diferencia de los 10 casos incorrectos.

Además, en el caso del cuadrado superior izquierdo, clase 0, que son las predicciones correctas, presenta un color purpura intenso a diferencia de los demás.

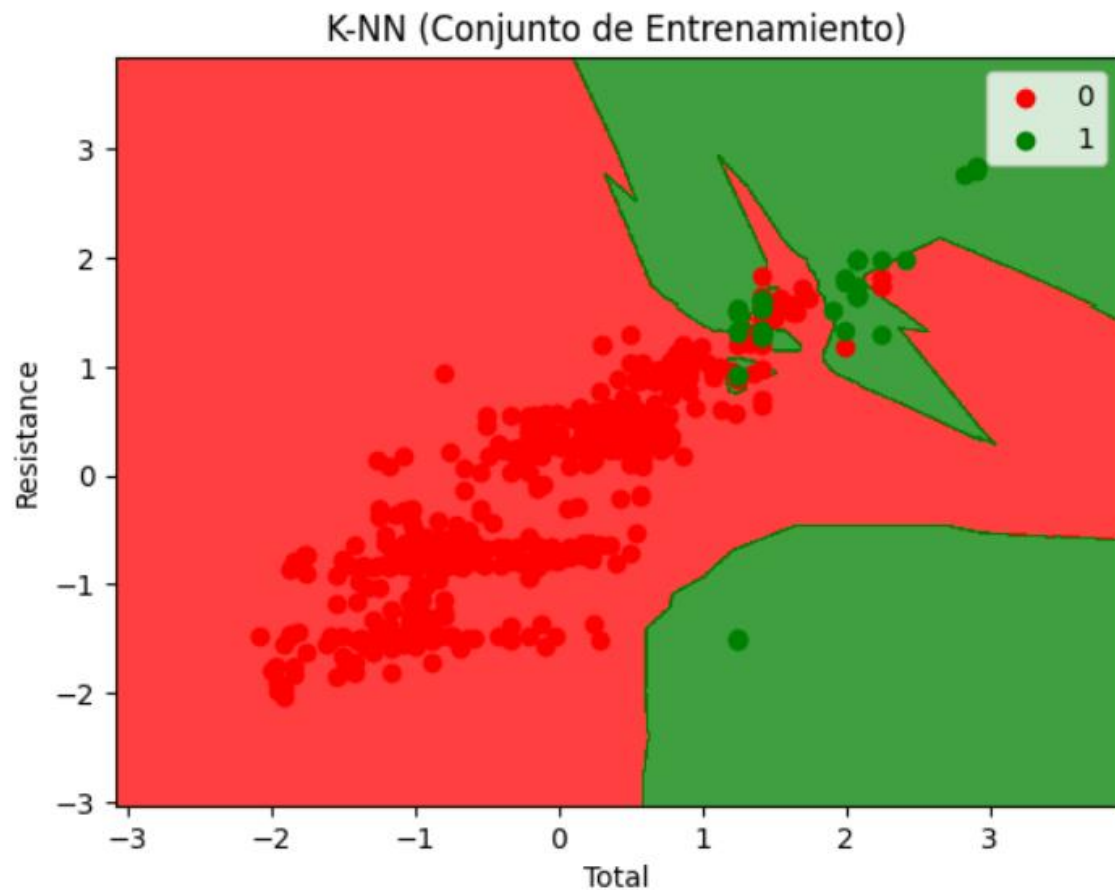
También se detecta una variación del color para el caso del cuadrado inferior derecho, clase 1, que son las predicciones correctas, presentando con un tono más gris y diferenciándose del blanco de las instancias incorrectas.

Estas observaciones son mejores que las vistas en el caso de clasificación de Regresión Logística.

Esto podría considerarlo como un mejor primer indicio del modelo.

Visualización del modelo de clasificación en el conjunto de Entrenamiento.

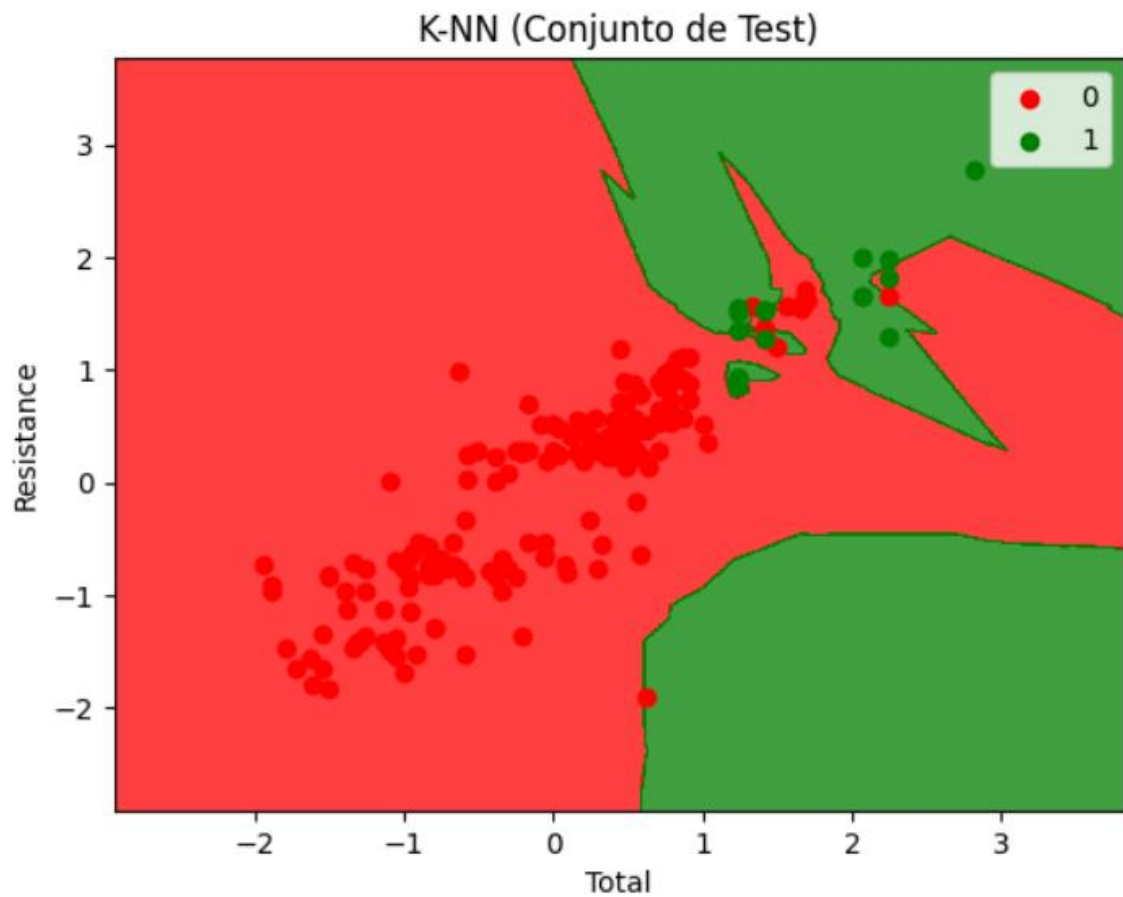
```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Conjunto de Entrenamiento)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
✓ 6.1s
```



Visualización del modelo de clasificación en el conjunto de Test.

```
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Conjunto de Test)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
```

✓ 5.9s



Ahora analizaré diferentes métricas de evaluación del rendimiento de un modelo de clasificación.

**Precisión del modelo:** La precisión mide la proporción de instancias positivas que fueron correctamente clasificadas como positivas.

**Exactitud del modelo:** La exactitud es la proporción total de predicciones correctas realizadas por el modelo.

**Sensibilidad del modelo:** La sensibilidad, también conocida como tasa de verdaderos positivos o recall, mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo.

**Puntaje F1 del modelo:** El puntaje F1 es una medida que combina la precisión y la sensibilidad en un solo valor. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad.

**Curva ROC - AUC del modelo:** La curva ROC, y el área bajo la curva (AUC) son métricas utilizadas para evaluar el rendimiento de un modelo de clasificación en problemas de clasificación binaria. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de clasificación. El AUC mide la capacidad del modelo para distinguir entre las clases.

```
print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
print(f'Exactitud del modelo: {accuracy_score(y_test, y_pred):.2f}')
print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
print(f'Puntaje F1 del modelo: {f1_score(y_test, y_pred):.2f}')
print(f'Curva ROC - AUC del modelo: {roc_auc_score(y_test, y_pred):.2f}')
```

✓ 0.0s

```
Precisión del modelo: 0.67
Exactitud del modelo: 0.95
Sensibilidad del modelo: 0.89
Puntaje F1 del modelo: 0.76
Curva ROC - AUC del modelo: 0.92
```

Imprimo las métricas de evaluación del modelo proporcionan información adicional sobre el rendimiento del modelo en términos de precisión, exactitud, sensibilidad, puntaje F1 y el área bajo la curva ROC-AUC.

```
print(classification_report(y_test, y_pred))
```

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.99      | 0.96   | 0.97     | 182     |
| 1            | 0.67      | 0.89   | 0.76     | 18      |
| accuracy     |           |        | 0.95     | 200     |
| macro avg    | 0.83      | 0.92   | 0.87     | 200     |
| weighted avg | 0.96      | 0.95   | 0.95     | 200     |

## Definiciones de otros datos

Macro avg: Calcula el promedio sin tener en cuenta el desequilibrio en el número de muestras de cada clase.

Para cada métrica, se calcula la media de los valores para todas las clases y se informa como el promedio macro avg.

Se trata de un promedio no ponderado en el que todas las clases tienen el mismo peso.

Este enfoque considera que todas las clases son igualmente importantes y le asigna la misma importancia a cada clase al calcular el promedio.

Weighted avg: Calcula el promedio ponderado teniendo en cuenta el número de muestras en cada clase.

Para cada métrica, se calcula la media ponderada de los valores para todas las clases y se informa como el promedio weighted avg.

En este enfoque, se asigna un peso proporcional al número de muestras de cada clase.

Las clases con más muestras tienen un impacto mayor en el cálculo del promedio.

Este enfoque es útil cuando hay un desequilibrio significativo en el número de muestras entre las clases.

Para la conclusión me centraré en los valores específicos obtenidos de cada clase en lugar de los promedios macro avg y weighted avg.

Los valores específicos de cada clase proporcionan información más detallada y relevante sobre cómo el modelo está clasificando cada clase individualmente.

El enfoque en los valores de cada clase es considerablemente más importante cuando hay un desequilibrio significativo en el número de muestras entre las clases.

En estos casos, los promedios pueden verse afectados por la clase dominante y no reflejar adecuadamente el rendimiento del modelo en las clases minoritarias.

Por lo tanto, los valores específicos de cada clase ofrecen una visión más detallada y proporcionan mayor eficiencia en las conclusiones del rendimiento del modelo de clasificación.



## Conclusión

Dado los resultados obtenidos con las diferentes métricas e informes puedo sacar las siguientes conclusiones adicionales.

El modelo tiene una precisión casi perfecta para la clase 0 con un 99% lo cual es algo altamente positivo.

Sin embargo, con respecto a la clase 1 el porcentaje de precisión baja teniendo un valor de precisión del 67%.

La exactitud o accuracy, devolvió un porcentaje muy elevado del 95% lo que significa que un muy alto número de instancias se clasificaron correctamente en general sobre el total de instancias.

En general, un mayor valor de exactitud o accuracy indica un mejor rendimiento del modelo, pero también puede ser engañoso en casos de desequilibrio de clases, donde una clase dominante puede influir en el resultado.

Por lo tanto, es un dato que hay que tomarlo con cierto cuidado.

Otro aspecto a tener en cuenta es que la sensibilidad con respecto a la clase 0 es del 96%, por lo que el modelo es capaz de identificar de manera confiable la mayoría de las instancias de esa clase.

Mientras que en la clase 1, la sensibilidad es del 86%, es un número bastante bueno también, y aún más teniendo en cuenta el valor de 39% obtenido en el modelo anterior.

El Puntaje F1, que es una medida que combina precisión y sensibilidad para evaluar el equilibrio entre ambas, arroja una diferenciación moderada, 97% para la clase 0 lo cual es muy bueno, y 76% para la clase 1 siendo un porcentaje moderado que no está mal, pero podría ser mejor.

Finalmente, el porcentaje arrojado por Curva ROC - AUC del modelo es del 92%, el cual es un valor muy por arriba del 50% (50% indica un modelo que clasifica las instancias de manera aleatoria), y podemos determinar que separa las clases positivas y negativas de buena manera.

Mi conclusión de este modelo de clasificación de K-Nearest Neighbors, para este caso particular, es que posee un rendimiento moderadamente bueno, ya que pese a tener un buen porcentaje de exactitud, un buen porcentaje de Curva ROC - AUC del modelo, y que también presenta buenos valores para la clase 0. Se encuentran valores que podrían ser mejores cuando tiene que clasificar a la clase 1, como el caso de la precisión con su 67%.

El f1-score de 76% también podría ser mejor.

Dado a los porcentajes obtenidos se concluye que, para este caso, este modelo de clasificación de K-Nearest Neighbors resulta más eficiente y confiable que el de Regresión Logística.

## Máquinas de Soporte Vectorial

Es un método de aprendizaje automático supervisado utilizado para la clasificación de datos.

SVM busca encontrar un hiperplano en un espacio de características que mejor separe las diferentes clases de datos.

Importo las librerías que me brinda Python.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, \
    recall_score, f1_score, roc_auc_score, classification_report
from matplotlib.colors import ListedColormap
import warnings
✓ 0.0s
```

Realizo la importación de los datos.

Ignoro las advertencias generadas por el módulo warnings en Python.

```
warnings.filterwarnings("ignore")
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()
✓ 0.0s
```

| #   | Name                  | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | Winbattle | Resistance |
|-----|-----------------------|--------|--------|-------|----|--------|---------|---------|---------|-------|------------|-----------|-----------|------------|
| 0 1 | Bulbasaur             | Grass  | Poison | 318   | 45 | 49     | 49      | 65      | 65      | 45    | 1          | False     | 35        | 450        |
| 1 2 | Ivysaur               | Grass  | Poison | 405   | 60 | 62     | 63      | 80      | 80      | 60    | 1          | False     | 50        | 730        |
| 2 3 | Venusaur              | Grass  | Poison | 525   | 80 | 82     | 83      | 100     | 100     | 80    | 1          | False     | 75        | 940        |
| 3 3 | VenusaurMega Venusaur | Grass  | Poison | 625   | 80 | 100    | 123     | 122     | 120     | 80    | 1          | False     | 86        | 1110       |
| 4 4 | Charmander            | Fire   | NaN    | 309   | 39 | 52     | 43      | 60      | 50      | 65    | 1          | False     | 40        | 500        |

Para este análisis tomaré los valores de Total y Resistance para la variable X independiente. También tomaré los valores de Legendary para la variable Y dependiente.

Consulto si hay algún valor null (NaN), me devolverá un booleano que en caso de ser true es porque si hay valores Nan.

```
df.isnull().values.any()
```

✓ 0.0s

True

Obtuve un valor True por lo que es necesario verificar en donde se encuentran los valores nulos.

Averiguo en que columnas hay valores null.

```
df.isnull().any()
```

[7] ✓ 0.0s

|            |       |
|------------|-------|
| #          | False |
| Name       | False |
| Type 1     | False |
| Type 2     | True  |
| Total      | False |
| HP         | False |
| Attack     | False |
| Defense    | False |
| Sp. Atk    | False |
| Sp. Def    | False |
| Speed      | False |
| Generation | False |
| Legendary  | False |
| Winbattle  | False |
| Resistance | False |
| dtype:     | bool  |

Constato que únicamente hay valores null en la columna Type2, por lo que no hay inconvenientes en tomar los datos de Total, Resistance y Legendary.

Coloco los datos de las columnas Total y Resistance en la variable X.

Coloco los datos de la columna Legendary en la variable y.

```
x = df.iloc[:, [4,14]].values
y = df.iloc[:, 12].values
X,y
```

```
[9] ✓ 0.0s
```

```
... (array([[ 318, 450],
[ 405, 730],
[ 525, 940],
...,
[ 600, 1095],
[ 680, 1135],
[ 600, 1030]], dtype=int64),
array([False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, True, True, True, False, False, False,
...,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
```

Utilizo LabelEncoder para convertir a números los datos categóricos de la columna Legendary almacenados en la variable y.

[illegible]

Imprimo nuevamente mis variables con sus datos.

[illegible]

Utilizo `train_test_split` para dividir un dataset en bloques.

Conjunto de testing por un lado y conjunto de entrenamiento por el otro.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

[12] ✓ 0.0s

## Escales los datos (variables)

El escalado va a transformar los valores de las características de forma que estén confinados en un rango  $[a, b]$ , típicamente  $[0, 1]$  o  $[-1, 1]$ .

```
sc_X = StandardScaler()  
X_train = sc_X.fit_transform(X_train)  
X_test = sc_X.transform(X_test)
```

✓ 0.0s

Ajusto el modelo de clasificación con el conjunto de entrenamiento.

Esto se hace para hacer predicciones en los datos de prueba y mostrar una tabla que compara las etiquetas reales con las etiquetas predichas para una muestra de prueba limitada.

```

classifier = SVC(kernel = "linear", random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).head(100)

```

✓ 0.0s

|     | Actual | Predicción |
|-----|--------|------------|
| 0   | 0      | 0          |
| 1   | 0      | 0          |
| 2   | 0      | 0          |
| 3   | 0      | 0          |
| 4   | 0      | 0          |
| ... | ...    | ...        |
| 95  | 0      | 1          |
| 96  | 0      | 0          |
| 97  | 1      | 0          |
| 98  | 0      | 0          |
| 99  | 0      | 0          |

100 rows × 2 columns

Construyo la matriz de confusión, la cual es una técnica muy potente.

Se calcula sobre el conjunto de testing y se puede observar si las predicciones que ha realizado el algoritmo son potentes

```

cm = confusion_matrix(y_test, y_pred)
cm

```

✓ 0.0s

```

array([[176,  6],
       [ 11,  7]], dtype=int64)

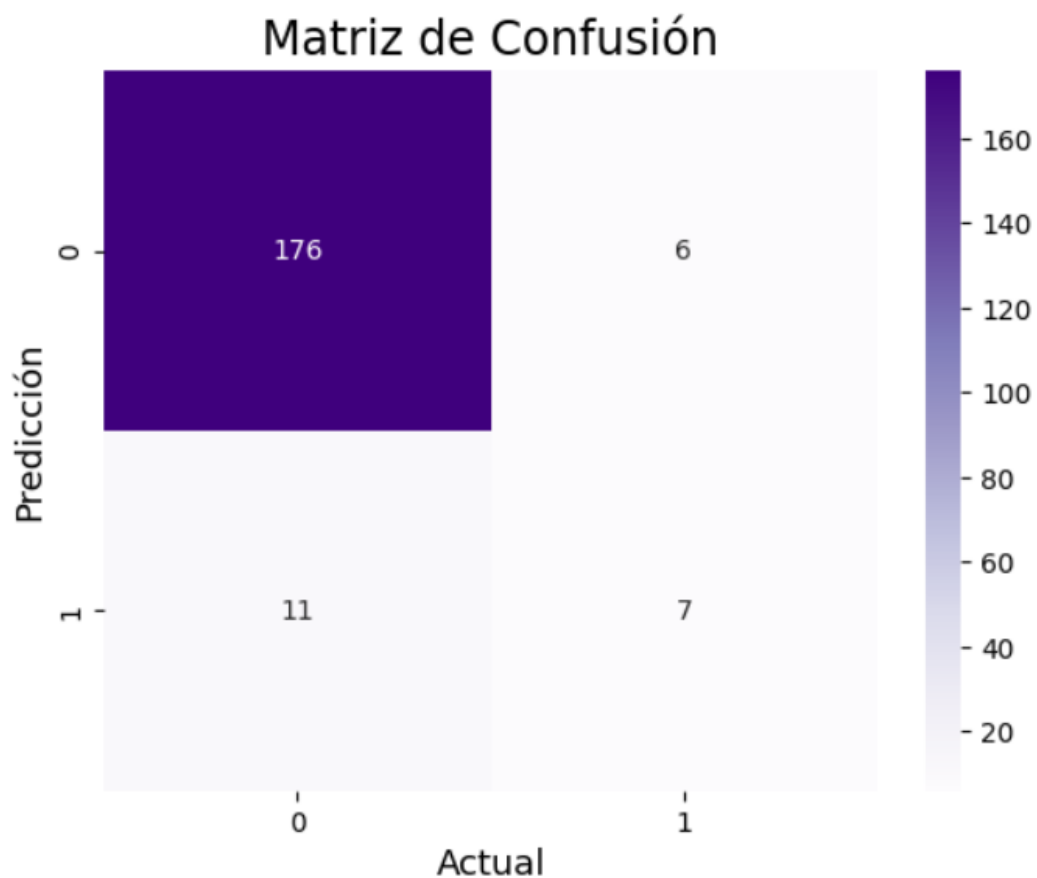
```

Visualizo la matriz de confusión de un modelo de clasificación utilizando la biblioteca Seaborn y Matplotlib en Python.

La matriz de confusión habla de qué tan bien ha evaluado el algoritmo la clasificación, en este caso, de los pokemones legendarios o no legendarios con respecto a las propiedades de Total y Resitence.

```
sns.heatmap(cm,  
            annot=True,  
            fmt='g',  
            cmap='Purples')  
plt.ylabel('Predicción',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Matriz de Confusión',fontsize=17)  
plt.show()
```

✓ 0.1s





Las predicciones correctas son los valores numéricos que se encuentran en el cuadrado superior a la izquierda junto con las que se encuentran en el cuadrado inferior a la derecha.

Mientras que las predicciones incorrectas son los valores numéricos que se encuentran en el cuadrado superior a la derecha junto con las que se encuentran en el cuadrado inferior a la izquierda.

## OBESERVACIÓN

El cuadrado superior izquierdo con un valor de 176 representa la cantidad de instancias clasificadas correctamente como clase 0.

El cuadrado superior derecho con un valor de 6 representa la cantidad de instancias clasificadas incorrectamente como clase 1 cuando en realidad son de clase 0.

El cuadrado inferior izquierdo con un valor de 11 representa la cantidad de instancias clasificadas incorrectamente como clase 0 cuando en realidad son de clase 1.

El cuadrado inferior derecho con un valor de 7 representa la cantidad de instancias clasificadas correctamente como clase 1.

Noto que hay un gran número de predicciones calificadas como correctas, 183 aplicando la suma de los valores correctos, a diferencia de los 17 casos incorrectos.

Además, en el caso del cuadrado superior izquierdo, clase 0, que son las predicciones correctas, presenta un color purpura intenso a diferencia de los demás.

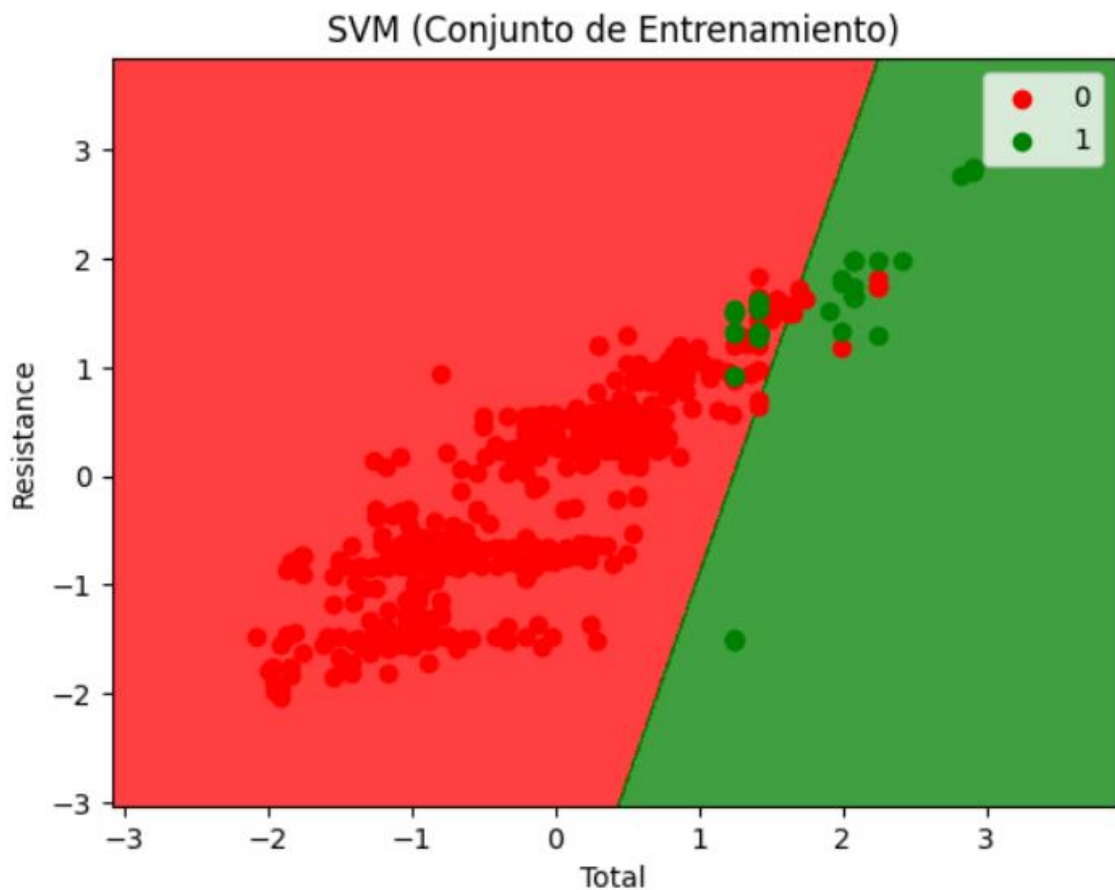
Estas observaciones son idénticas que las vistas en el caso de clasificación de Regresión Logística.

De igual forma, podría considerar estos valores obtenidos en la matriz de confusión como un buen primer indicio del modelo.

Visualización del modelo de clasificación en el conjunto de Entrenamiento.

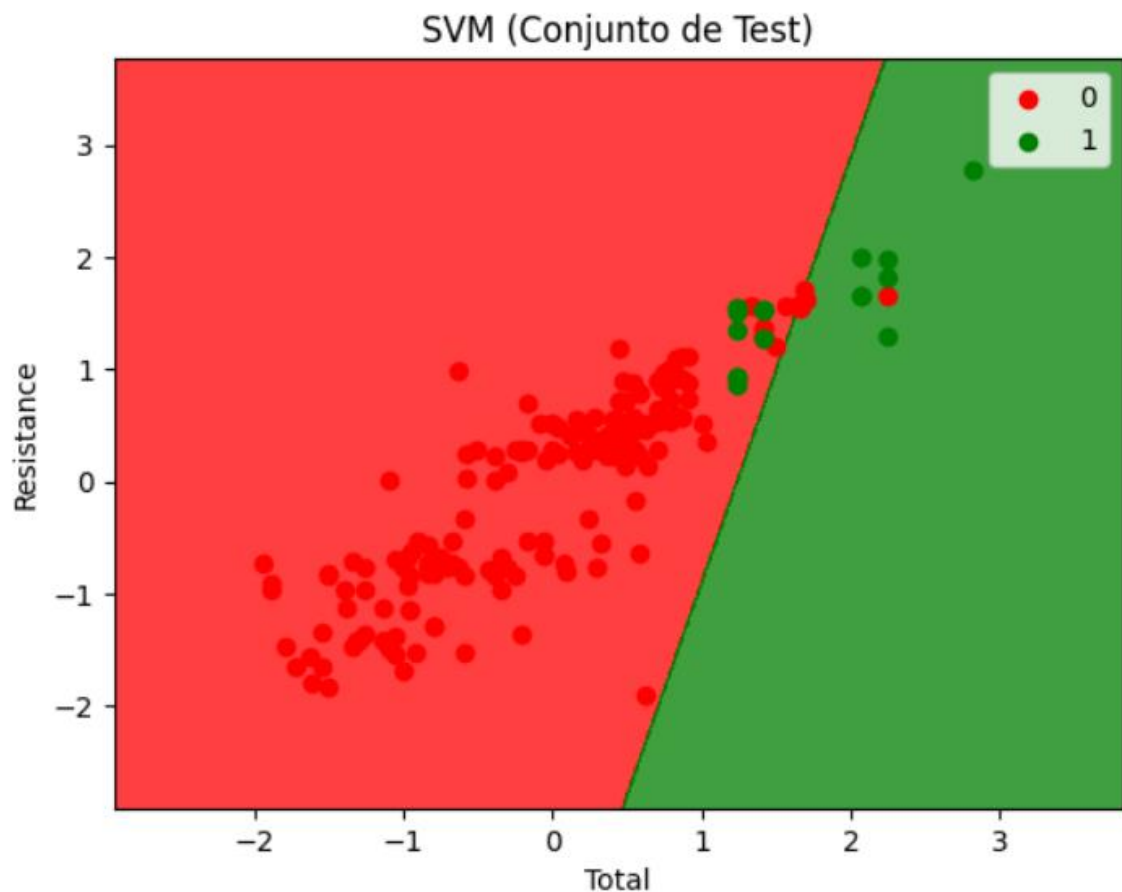
```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Conjunto de Entrenamiento)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
```

✓ 0.5s



Visualización del modelo de clasificación en el conjunto de Test.

```
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Conjunto de Test)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
✓ 0.5s
```



Ahora analizaré diferentes métricas de evaluación del rendimiento de un modelo de clasificación.

**Precisión del modelo:** La precisión mide la proporción de instancias positivas que fueron correctamente clasificadas como positivas.

**Exactitud del modelo:** La exactitud es la proporción total de predicciones correctas realizadas por el modelo.

**Sensibilidad del modelo:** La sensibilidad, también conocida como tasa de verdaderos positivos o recall, mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo.

**Puntaje F1 del modelo:** El puntaje F1 es una medida que combina la precisión y la sensibilidad en un solo valor. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad.

**Curva ROC - AUC del modelo:** La curva ROC, y el área bajo la curva (AUC) son métricas utilizadas para evaluar el rendimiento de un modelo de clasificación en problemas de clasificación binaria. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de clasificación. El AUC mide la capacidad del modelo para distinguir entre las clases.

```
print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
print(f'Exactitud del modelo: {accuracy_score(y_test, y_pred):.2f}')
print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
print(f'Puntaje F1 del modelo: {f1_score(y_test, y_pred):.2f}')
print(f'Curva ROC - AUC del modelo: {roc_auc_score(y_test, y_pred):.2f}')
```

✓ 0.0s

Precisión del modelo: 0.54  
 Exactitud del modelo: 0.92  
 Sensibilidad del modelo: 0.39  
 Puntaje F1 del modelo: 0.45  
 Curva ROC - AUC del modelo: 0.68

Imprimo las métricas de evaluación del modelo proporcionan información adicional sobre el rendimiento del modelo en términos de precisión, exactitud, sensibilidad, puntaje F1 y el área bajo la curva ROC-AUC.

```
print(classification_report(y_test, y_pred))
```

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.94      | 0.97   | 0.95     | 182     |
| 1            | 0.54      | 0.39   | 0.45     | 18      |
| accuracy     |           |        | 0.92     | 200     |
| macro avg    | 0.74      | 0.68   | 0.70     | 200     |
| weighted avg | 0.90      | 0.92   | 0.91     | 200     |

## Definiciones de otros datos

Macro avg: Calcula el promedio sin tener en cuenta el desequilibrio en el número de muestras de cada clase.

Para cada métrica, se calcula la media de los valores para todas las clases y se informa como el promedio macro avg.

Se trata de un promedio no ponderado en el que todas las clases tienen el mismo peso.

Este enfoque considera que todas las clases son igualmente importantes y le asigna la misma importancia a cada clase al calcular el promedio.

Weighted avg: Calcula el promedio ponderado teniendo en cuenta el número de muestras en cada clase.

Para cada métrica, se calcula la media ponderada de los valores para todas las clases y se informa como el promedio weighted avg.

En este enfoque, se asigna un peso proporcional al número de muestras de cada clase.

Las clases con más muestras tienen un impacto mayor en el cálculo del promedio.

Este enfoque es útil cuando hay un desequilibrio significativo en el número de muestras entre las clases.

Para la conclusión me centraré en los valores específicos obtenidos de cada clase en lugar de los promedios macro avg y weighted avg.

Los valores específicos de cada clase proporcionan información más detallada y relevante sobre cómo el modelo está clasificando cada clase individualmente.

El enfoque en los valores de cada clase es considerablemente más importante cuando hay un desequilibrio significativo en el número de muestras entre las clases.

En estos casos, los promedios pueden verse afectados por la clase dominante y no reflejar adecuadamente el rendimiento del modelo en las clases minoritarias.

Por lo tanto, los valores específicos de cada clase ofrecen una visión más detallada y proporcionan mayor eficiencia en las conclusiones del rendimiento del modelo de clasificación.

## Conclusión

Dado los resultados obtenidos con las diferentes métricas e informes puedo sacar las siguientes conclusiones adicionales.

El modelo tiene una precisión alta para la clase 0 con un 94% lo cual es algo altamente positivo.

Sin embargo, con respecto a la clase 1 el porcentaje de precisión baja considerablemente teniendo apenas un 54% de precisión.

La exactitud o accuracy, devolvió un porcentaje muy elevado del 92% lo que significa que un muy alto número de instancias se clasificaron correctamente en general sobre el total de instancias.

En general, un mayor valor de exactitud o accuracy indica un mejor rendimiento del modelo, pero también puede ser engañoso en casos de desequilibrio de clases, donde una clase dominante puede influir en el resultado.

Por lo tanto, es un dato que hay que tomarlo con cierto cuidado.

Otro aspecto a tener en cuenta es que la sensibilidad con respecto a la clase 0 es del 97%, por lo que el modelo es capaz de identificar de manera confiable la mayoría de las instancias de esa clase.

Aunque el inconveniente vuelve a surgir en la clase 1, en donde la sensibilidad desciende hasta el valor de 39% lo que es considerablemente bajo.

El Puntaje F1, que es una medida que combina precisión y sensibilidad para evaluar el equilibrio entre ambas, vuelve arrojar una diferenciación muy marcada, 95% para la clase 0 lo cual es muy bueno, pero 45% para la clase 1 siendo un porcentaje bajo.

Finalmente, el porcentaje arrojado por Curva ROC - AUC del modelo es del 68%, el cual es un valor por arriba del 50% (50% indica un modelo que clasifica las instancias de manera aleatoria), aunque lo esperado sería un valor aún más alto del modelo para separar las clases positivas y negativas y evitando cometer errores.

Mi conclusión de este modelo de clasificación SVM, para este caso, es que posee un rendimiento regular, similar al de Regresión Logística, ya que, pese a tener un buen porcentaje de exactitud y que también presenta buenos valores para la clase 0, se encuentran inconvenientes cuando tiene que clasificar a la clase 1, dado a los porcentajes obtenidos se concluye que no lo hace de forma eficiente ni confiable.

## SVM Soft Margin (hiperplano óptimo)

El caso visto anteriormente tiene poca aplicación práctica porque se encuentran casos en los que las clases sean perfecta y linealmente separables.

El caso anterior de SVM también es muy sensible a variaciones en los datos y presenta problemas de overfitting.

Por esta razón es preferible crear un clasificador basado en un hiperplano que, aunque no separe perfectamente las dos clases, sea más robusto y tenga mayor capacidad predictiva al aplicarlo a nuevas observaciones, es decir, menos problemas de overfitting.

Importo las librerías que me brinda Python.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, \
recall_score, f1_score, roc_auc_score, classification_report
from matplotlib.colors import ListedColormap
import warnings

✓ 0.0s
```

Realizo la importación de los datos.

Ignoro las advertencias generadas por el módulo warnings en Python.

```
warnings.filterwarnings("ignore")
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()
```

✓ 0.0s

| #   | Name                  | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | Winbattle | Resistance |
|-----|-----------------------|--------|--------|-------|----|--------|---------|---------|---------|-------|------------|-----------|-----------|------------|
| 0 1 | Bulbasaur             | Grass  | Poison | 318   | 45 | 49     | 49      | 65      | 65      | 45    | 1          | False     | 35        | 450        |
| 1 2 | Ivysaur               | Grass  | Poison | 405   | 60 | 62     | 63      | 80      | 80      | 60    | 1          | False     | 50        | 730        |
| 2 3 | Venusaur              | Grass  | Poison | 525   | 80 | 82     | 83      | 100     | 100     | 80    | 1          | False     | 75        | 940        |
| 3 3 | VenusaurMega Venusaur | Grass  | Poison | 625   | 80 | 100    | 123     | 122     | 120     | 80    | 1          | False     | 86        | 1110       |
| 4 4 | Charmander            | Fire   | NaN    | 309   | 39 | 52     | 43      | 60      | 50      | 65    | 1          | False     | 40        | 500        |

Para este análisis tomaré los valores de Total y Resistance para la variable X independiente. También tomaré los valores de Legendary para la variable Y dependiente.

Consulto si hay algún valor null (NaN), me devolverá un booleano que en caso de ser true es porque si hay valores Nan.

```
df.isnull().values.any()
```

✓ 0.0s

True

Obtuve un valor True por lo que es necesario verificar en donde se encuentran los valores nulos.

Averiguo en que columnas hay valores null.

```
df.isnull().any()
```

[7] ✓ 0.0s

|            |       |
|------------|-------|
| #          | False |
| Name       | False |
| Type 1     | False |
| Type 2     | True  |
| Total      | False |
| HP         | False |
| Attack     | False |
| Defense    | False |
| Sp. Atk    | False |
| Sp. Def    | False |
| Speed      | False |
| Generation | False |
| Legendary  | False |
| Winbattle  | False |
| Resistance | False |
| dtype:     | bool  |

Constato que únicamente hay valores null en la columna Type2, por lo que no hay inconvenientes en tomar los datos de Total, Resistance y Legendary.



Coloco los datos de las columnas Total y Resistance en la variable X.

Coloco los datos de la columna Legendary en la variable y.

```
x = df.iloc[:, [4,14]].values
y = df.iloc[:, 12].values
X,y
```

[9] ✓ 0.0s

```
... (array([[ 318, 450],
[ 405, 730],
[ 525, 940],
...,
[ 600, 1095],
[ 680, 1135],
[ 600, 1030]], dtype=int64),
array([False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, True, True, True, False, False, False,
...,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
```

Utilizo LabelEncoder para convertir a números los datos categóricos de la columna Legendary almacenados en la variable y.

[illegible]

Imprimo nuevamente mis variables con sus datos.

[illegible]

Utilizo `train_test_split` para dividir un dataset en bloques.

Conjunto de testing por un lado y conjunto de entrenamiento por el otro.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

## Escales los datos (variables)

El escalado va a transformar los valores de las características de forma que estén confinados en un rango  $[a, b]$ , típicamente  $[0, 1]$  o  $[-1, 1]$ .

```
sc_X = StandardScaler()  
X_train = sc_X.fit_transform(X_train)  
X_test = sc_X.transform(X_test)
```

✓ 0.0s

Ajusto el modelo de clasificación con el conjunto de entrenamiento.

Esto se hace para hacer predicciones en los datos de prueba y mostrar una tabla que compara las etiquetas reales con las etiquetas predichas para una muestra de prueba limitada.

```

classifier = SVC(kernel = "rbf", random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).head(100)

```

✓ 0.0s

|     | Actual | Predicción |
|-----|--------|------------|
| 0   | 0      | 0          |
| 1   | 0      | 0          |
| 2   | 0      | 0          |
| 3   | 0      | 0          |
| 4   | 0      | 0          |
| ... | ...    | ...        |
| 95  | 0      | 1          |
| 96  | 0      | 0          |
| 97  | 1      | 0          |
| 98  | 0      | 0          |
| 99  | 0      | 0          |

100 rows × 2 columns

Construyo la matriz de confusión, la cual es una técnica muy potente.

Se calcula sobre el conjunto de testing y se puede observar si las predicciones que ha realizado el algoritmo son potentes

```

cm = confusion_matrix(y_test, y_pred)
cm

```

✓ 0.0s

```

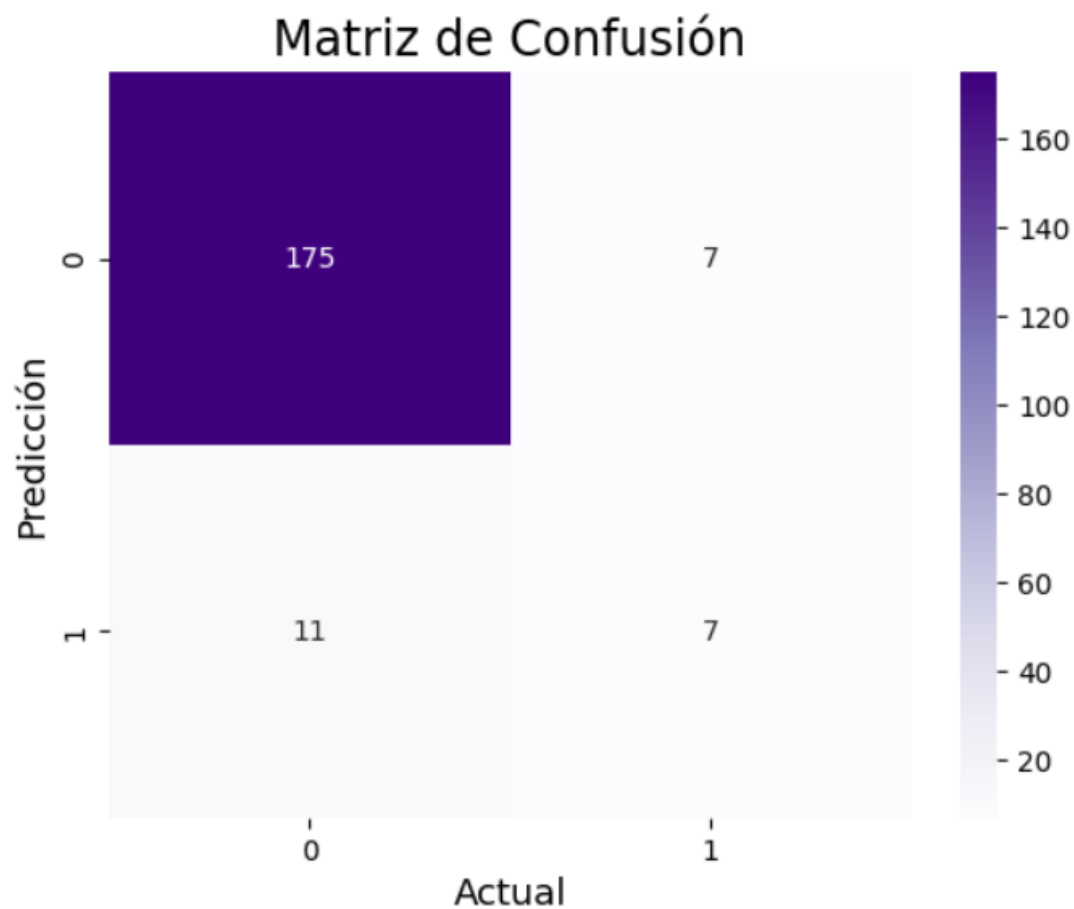
array([[175,  7],
       [ 11,  7]], dtype=int64)

```

Visualizo la matriz de confusión de un modelo de clasificación utilizando la biblioteca Seaborn y Matplotlib en Python.

La matriz de confusión habla de qué tan bien ha evaluado el algoritmo la clasificación, en este caso, de los pokemones legendarios o no legendarios con respecto a las propiedades de Total y Resitence.

```
sns.heatmap(cm,  
            annot=True,  
            fmt='g',  
            cmap='Purples')  
plt.ylabel('Predicción',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Matriz de Confusión',fontsize=17)  
plt.show()  
✓ 0.1s
```



Las predicciones correctas son los valores numéricos que se encuentran en el cuadrado superior a la izquierda junto con las que se encuentran en el cuadrado inferior a la derecha.

Mientras que las predicciones incorrectas son los valores numéricos que se encuentran en el cuadrado superior a la derecha junto con las que se encuentran en el cuadrado inferior a la izquierda.

#### OBESERVACIÓN

El cuadrado superior izquierdo con un valor de 175 representa la cantidad de instancias clasificadas correctamente como clase 0.

El cuadrado superior derecho con un valor de 7 representa la cantidad de instancias clasificadas incorrectamente como clase 1 cuando en realidad son de clase 0.

El cuadrado inferior izquierdo con un valor de 11 representa la cantidad de instancias clasificadas incorrectamente como clase 0 cuando en realidad son de clase 1.

El cuadrado inferior derecho con un valor de 7 representa la cantidad de instancias clasificadas correctamente como clase 1.

Noto que hay un gran número de predicciones calificadas como correctas, 182 aplicando la suma de los valores correctos, a diferencia de los 18 casos incorrectos.

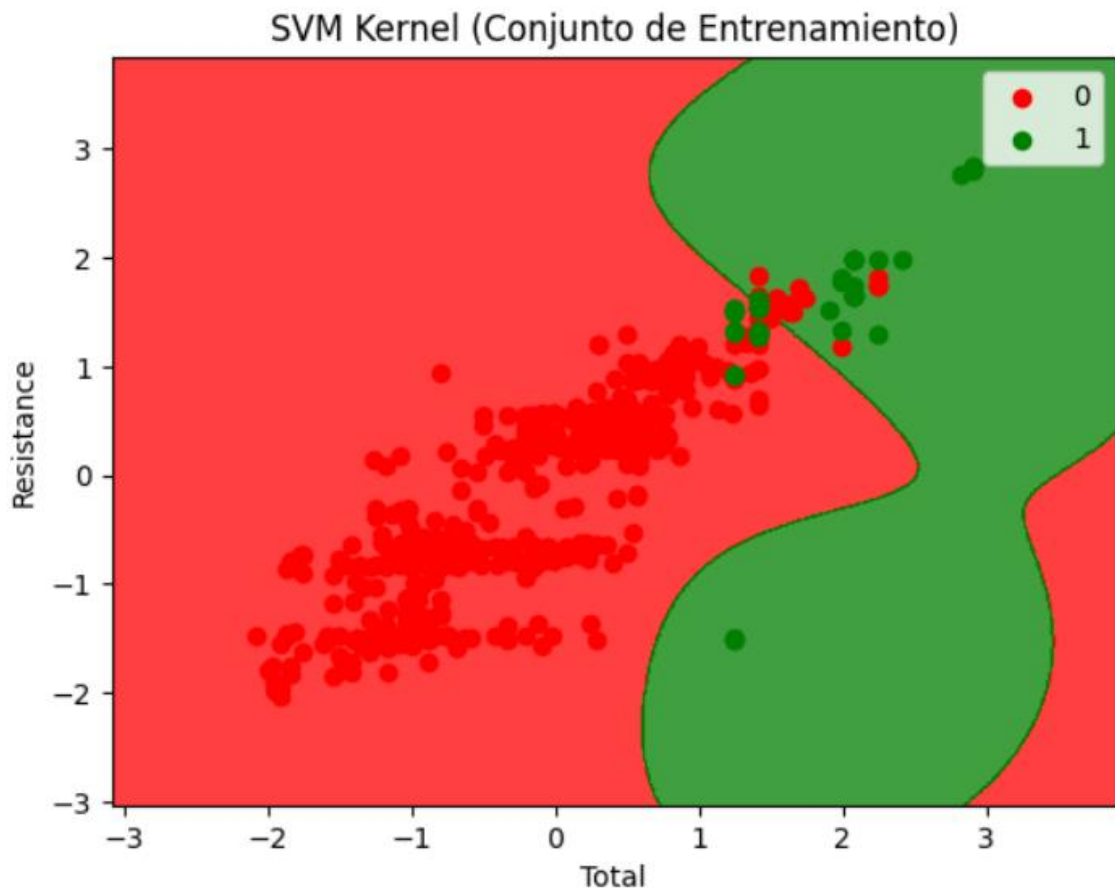
Además, en el caso del cuadrado superior izquierdo, clase 0, que son las predicciones correctas, presenta un color purpura intenso a diferencia de los demás.

Observo resultados muy similares a los de Regresión Logística y SVM sin soft margin.

Esto podría considerarlo como un buen primer indicio del modelo.

Visualización del modelo de clasificación en el conjunto de Entrenamiento.

```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM Kernel (Conjunto de Entrenamiento)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
✓ 2.2s
```

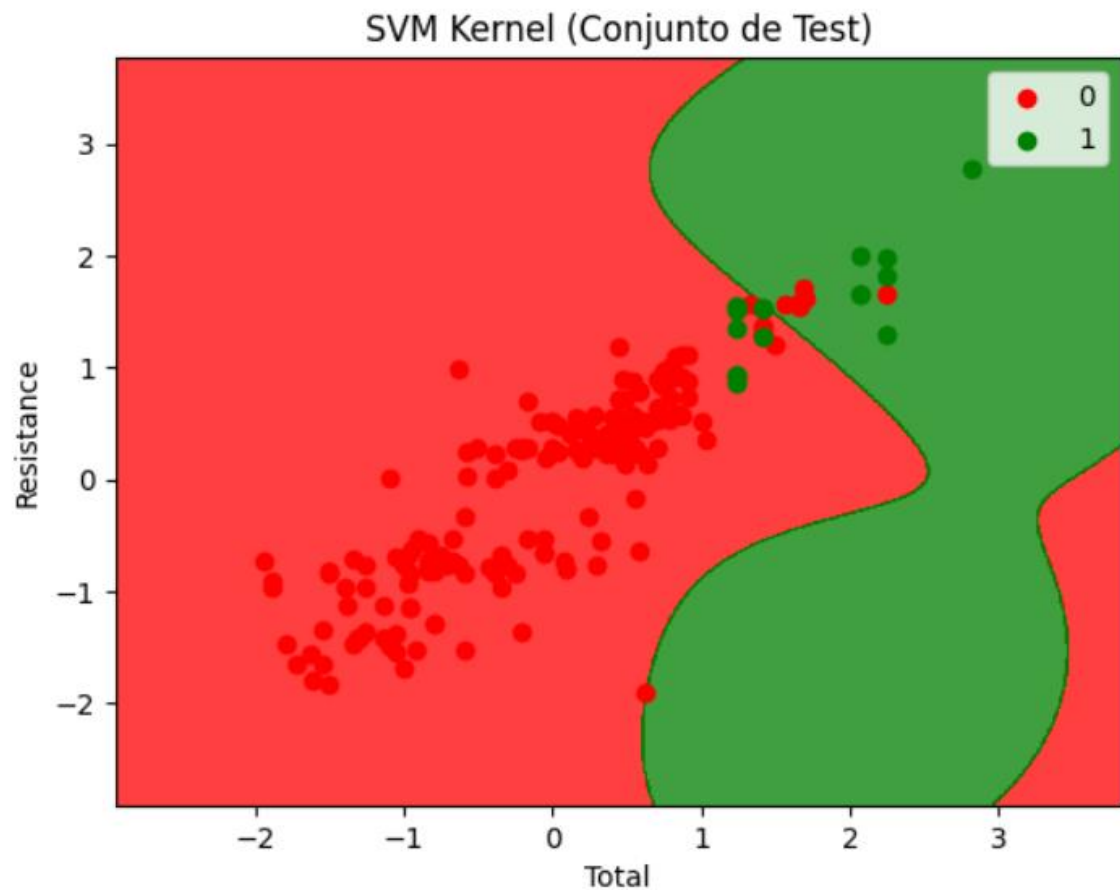


Visualización del modelo de clasificación en el conjunto de Test.

```

X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM Kernel (Conjunto de Test)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
✓ 2.1s

```



Ahora analizaré diferentes métricas de evaluación del rendimiento de un modelo de clasificación.

**Precisión del modelo:** La precisión mide la proporción de instancias positivas que fueron correctamente clasificadas como positivas.

**Exactitud del modelo:** La exactitud es la proporción total de predicciones correctas realizadas por el modelo.

**Sensibilidad del modelo:** La sensibilidad, también conocida como tasa de verdaderos positivos o recall, mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo.

**Puntaje F1 del modelo:** El puntaje F1 es una medida que combina la precisión y la sensibilidad en un solo valor. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad.

**Curva ROC - AUC del modelo:** La curva ROC, y el área bajo la curva (AUC) son métricas utilizadas para evaluar el rendimiento de un modelo de clasificación en problemas de clasificación binaria. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de clasificación. El AUC mide la capacidad del modelo para distinguir entre las clases.

```
print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
print(f'Exactitud del modelo: {accuracy_score(y_test, y_pred):.2f}')
print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
print(f'Puntaje F1 del modelo: {f1_score(y_test, y_pred):.2f}')
print(f'Curva ROC - AUC del modelo: {roc_auc_score(y_test, y_pred):.2f}')
```

✓ 0.0s

```
Precisión del modelo: 0.50
Exactitud del modelo: 0.91
Sensibilidad del modelo: 0.39
Puntaje F1 del modelo: 0.44
Curva ROC - AUC del modelo: 0.68
```

Imprimo las métricas de evaluación del modelo proporcionan información adicional sobre el rendimiento del modelo en términos de precisión, exactitud, sensibilidad, puntaje F1 y el área bajo la curva ROC-AUC.

```
print(classification_report(y_test, y_pred))
```

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.94      | 0.96   | 0.95     | 182     |
| 1            | 0.50      | 0.39   | 0.44     | 18      |
| accuracy     |           |        | 0.91     | 200     |
| macro avg    | 0.72      | 0.68   | 0.69     | 200     |
| weighted avg | 0.90      | 0.91   | 0.90     | 200     |



## Definiciones de otros datos

Macro avg: Calcula el promedio sin tener en cuenta el desequilibrio en el número de muestras de cada clase.

Para cada métrica, se calcula la media de los valores para todas las clases y se informa como el promedio macro avg.

Se trata de un promedio no ponderado en el que todas las clases tienen el mismo peso.

Este enfoque considera que todas las clases son igualmente importantes y le asigna la misma importancia a cada clase al calcular el promedio.

Weighted avg: Calcula el promedio ponderado teniendo en cuenta el número de muestras en cada clase.

Para cada métrica, se calcula la media ponderada de los valores para todas las clases y se informa como el promedio weighted avg.

En este enfoque, se asigna un peso proporcional al número de muestras de cada clase.

Las clases con más muestras tienen un impacto mayor en el cálculo del promedio.

Este enfoque es útil cuando hay un desequilibrio significativo en el número de muestras entre las clases.

Para la conclusión me centraré en los valores específicos obtenidos de cada clase en lugar de los promedios macro avg y weighted avg.

Los valores específicos de cada clase proporcionan información más detallada y relevante sobre cómo el modelo está clasificando cada clase individualmente.

El enfoque en los valores de cada clase es considerablemente más importante cuando hay un desequilibrio significativo en el número de muestras entre las clases.

En estos casos, los promedios pueden verse afectados por la clase dominante y no reflejar adecuadamente el rendimiento del modelo en las clases minoritarias.

Por lo tanto, los valores específicos de cada clase ofrecen una visión más detallada y proporcionan mayor eficiencia en las conclusiones del rendimiento del modelo de clasificación.

## Conclusión

Dado los resultados obtenidos con las diferentes métricas e informes puedo sacar las siguientes conclusiones adicionales.

El modelo tiene una precisión alta para la clase 0 con un 94% lo cual es algo altamente positivo.

Sin embargo, con respecto a la clase 1 el porcentaje de precisión baja bastante teniendo apenas la mitad de precisión, un 50%.

La exactitud o accuracy, devolvió un porcentaje muy elevado del 91% lo que significa que un muy alto número de instancias se clasificaron correctamente en general sobre el total de instancias.

En general, un mayor valor de exactitud o accuracy indica un mejor rendimiento del modelo, pero también puede ser engañoso en casos de desequilibrio de clases, donde una clase dominante puede influir en el resultado.

Por lo tanto, es un dato que hay que tomarlo con cierto cuidado.

Otro aspecto a tener en cuenta es que la sensibilidad con respecto a la clase 0 es del 96%, por lo que el modelo es capaz de identificar de manera confiable la mayoría de las instancias de esa clase.

Aunque el inconveniente vuelve a surgir en la clase 1, en donde la sensibilidad desciende hasta el valor de 39% lo que es considerablemente bajo.

El Puntaje F1, que es una medida que combina precisión y sensibilidad para evaluar el equilibrio entre ambas, vuelve arrojar una diferenciación muy marcada, 95% para la clase 0 lo cual es muy bueno, pero 44% para la clase 1 siendo un porcentaje bajo.

Finalmente, el porcentaje arrojado por Curva ROC - AUC del modelo es del 68%, el cual es un valor por arriba del 50% (50% indica un modelo que clasifica las instancias de manera aleatoria), aunque lo esperado sería un valor aún más alto del modelo para separar las clases positivas y negativas y evitando cometer errores.

Mi conclusión de este modelo de clasificación SVM tipo soft margin, para este caso, es que posee un rendimiento regular, ya que, pese a tener un buen porcentaje de exactitud y que también presenta buenos valores para la clase 0, se encuentran inconvenientes cuando tiene que clasificar a la clase 1, dado a los porcentajes obtenidos se concluye que no lo hace de forma eficiente ni confiable.

Dado a que en algunos valores el rendimiento fue incluso por debajo de los anteriores modelos analizados, por el momento considero que es el modelo menos confiable para este caso.

## Naive Bayes

Es un método de aprendizaje automático supervisado que se basa en el teorema de Bayes y la suposición de independencia condicional entre las características.

El algoritmo Naive Bayes se basa en el teorema de Bayes, que establece la relación entre las probabilidades condicionales inversas.

El teorema de Bayes establece que la probabilidad de una hipótesis dado un conjunto de evidencias se puede calcular utilizando la probabilidad de la evidencia dado la hipótesis y las probabilidades previas de las hipótesis y las evidencias.

Para clasificar una nueva instancia, el algoritmo Naive Bayes calcula la probabilidad posterior de cada clase dada la instancia utilizando el teorema de Bayes. Se selecciona la clase con la probabilidad posterior más alta como la predicción del algoritmo.

Importo las librerías que me brinda Python.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, \
    recall_score, f1_score, roc_auc_score, classification_report
from matplotlib.colors import ListedColormap
import warnings
```

✓ 0.0s

Realizo la importación de los datos.

Ignoro las advertencias generadas por el módulo warnings en Python.

```
warnings.filterwarnings("ignore")
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()
```

✓ 0.0s

| #   | Name                  | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | Winbattle | Resistance |
|-----|-----------------------|--------|--------|-------|----|--------|---------|---------|---------|-------|------------|-----------|-----------|------------|
| 0 1 | Bulbasaur             | Grass  | Poison | 318   | 45 | 49     | 49      | 65      | 65      | 45    | 1          | False     | 35        | 450        |
| 1 2 | Ivysaur               | Grass  | Poison | 405   | 60 | 62     | 63      | 80      | 80      | 60    | 1          | False     | 50        | 730        |
| 2 3 | Venusaur              | Grass  | Poison | 525   | 80 | 82     | 83      | 100     | 100     | 80    | 1          | False     | 75        | 940        |
| 3 3 | VenusaurMega Venusaur | Grass  | Poison | 625   | 80 | 100    | 123     | 122     | 120     | 80    | 1          | False     | 86        | 1110       |
| 4 4 | Charmander            | Fire   | NaN    | 309   | 39 | 52     | 43      | 60      | 50      | 65    | 1          | False     | 40        | 500        |

Para este análisis tomaré los valores de Total y Resistance para la variable X independiente. También tomaré los valores de Legendary para la variable Y dependiente.

Consulto si hay algún valor null (NaN), me devolverá un booleano que en caso de ser true es porque si hay valores Nan.

```
df.isnull().values.any()
```

✓ 0.0s

True

Obtuve un valor True por lo que es necesario verificar en donde se encuentran los valores nulos.

Averiguo en que columnas hay valores null.

```
df.isnull().any()
```

[7] ✓ 0.0s

|            |       |
|------------|-------|
| #          | False |
| Name       | False |
| Type 1     | False |
| Type 2     | True  |
| Total      | False |
| HP         | False |
| Attack     | False |
| Defense    | False |
| Sp. Atk    | False |
| Sp. Def    | False |
| Speed      | False |
| Generation | False |
| Legendary  | False |
| Winbattle  | False |
| Resistance | False |
| dtype:     | bool  |

Constato que únicamente hay valores null en la columna Type2, por lo que no hay inconvenientes en tomar los datos de Total, Resistance y Legendary.

Coloco los datos de las columnas Total y Resistance en la variable X.

Coloco los datos de la columna Legendary en la variable y.

```
x = df.iloc[:, [4,14]].values
y = df.iloc[:, 12].values
X,y
```

```
[9] ✓ 0.0s
```

```
... (array([[ 318, 450],
[ 405, 730],
[ 525, 940],
...,
[ 600, 1095],
[ 680, 1135],
[ 600, 1030]], dtype=int64),
array([False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, True, True, True, False, False, False,
...,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
```

Utilizo LabelEncoder para convertir a números los datos categóricos de la columna Legendary almacenados en la variable y.

[illegible]



Ajusto el modelo de clasificación con el conjunto de entrenamiento.

Esto se hace para hacer predicciones en los datos de prueba y mostrar una tabla que compara las etiquetas reales con las etiquetas predichas para una muestra de prueba limitada.

```

classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).head(100)

```

✓ 0.0s

|     | Actual | Predicción |
|-----|--------|------------|
| 0   | 0      | 0          |
| 1   | 0      | 0          |
| 2   | 0      | 0          |
| 3   | 0      | 0          |
| 4   | 0      | 0          |
| ... | ...    | ...        |
| 95  | 0      | 1          |
| 96  | 0      | 0          |
| 97  | 1      | 0          |
| 98  | 0      | 0          |
| 99  | 0      | 0          |

100 rows × 2 columns

Construyo la matriz de confusión, la cual es una técnica muy potente.

Se calcula sobre el conjunto de testing y se puede observar si las predicciones que ha realizado el algoritmo son potentes

```

cm = confusion_matrix(y_test, y_pred)
cm

```

✓ 0.0s

```

array([[168, 14],
       [ 3, 15]], dtype=int64)

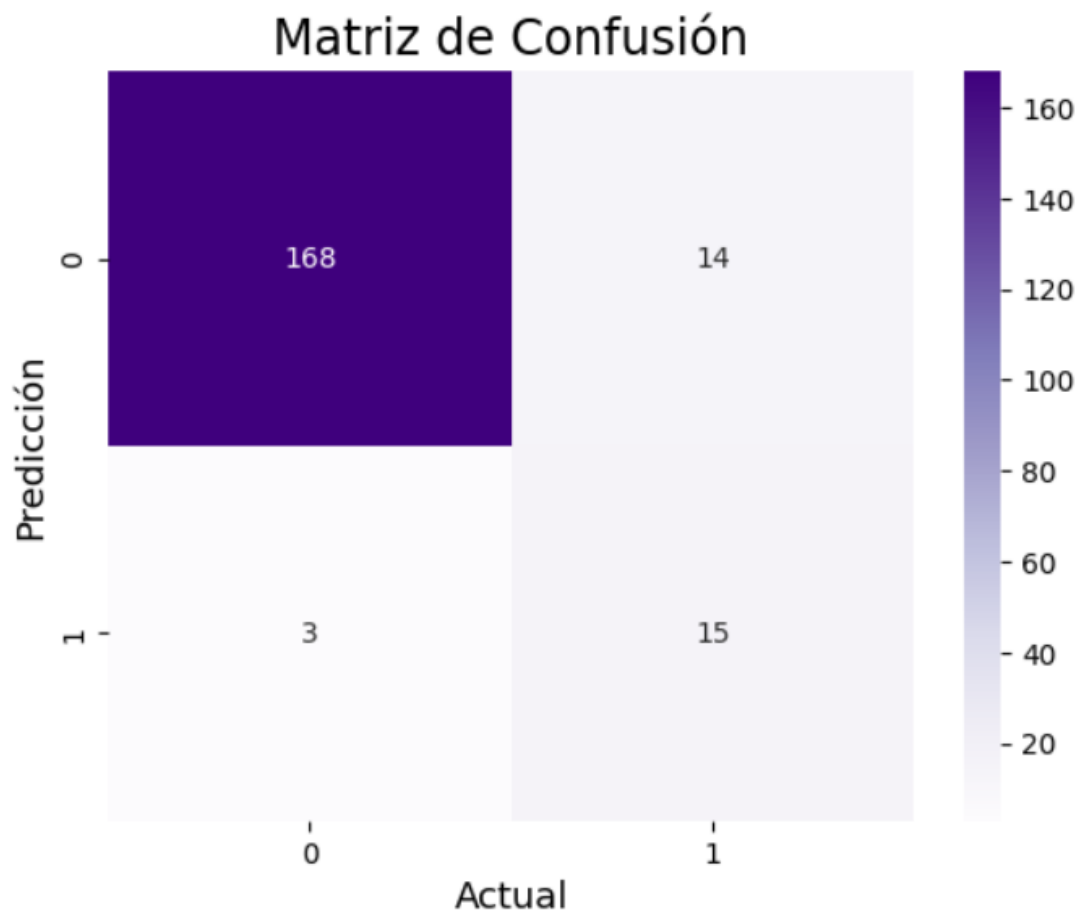
```

Visualizo la matriz de confusión de un modelo de clasificación utilizando la biblioteca Seaborn y Matplotlib en Python.

La matriz de confusión habla de qué tan bien ha evaluado el algoritmo la clasificación, en este caso, de los pokemones legendarios o no legendarios con respecto a las propiedades de Total y Resitence.

```
sns.heatmap(cm,  
            annot=True,  
            fmt='g',  
            cmap='Purples')  
plt.ylabel('Predicción',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Matriz de Confusión',fontsize=17)  
plt.show()
```

✓ 0.1s





Las predicciones correctas son los valores numéricos que se encuentran en el cuadrado superior a la izquierda junto con las que se encuentran en el cuadrado inferior a la derecha.

# Mientras que las predicciones incorrectas son los valores numéricos que se encuentran en el cuadrado superior a la derecha junto con las que se encuentran en el cuadrado inferior a la izquierda.

## OBESERVACIÓN

El cuadrado superior izquierdo con un valor de 168 representa la cantidad de instancias clasificadas correctamente como clase 0.

El cuadrado superior derecho con un valor de 14 representa la cantidad de instancias clasificadas incorrectamente como clase 1 cuando en realidad son de clase 0.

El cuadrado inferior izquierdo con un valor de 3 representa la cantidad de instancias clasificadas incorrectamente como clase 0 cuando en realidad son de clase 1.

El cuadrado inferior derecho con un valor de 15 representa la cantidad de instancias clasificadas correctamente como clase 1.

Noto que hay un gran número de predicciones calificadas como correctas, 183 aplicando la suma de los valores correctos, a diferencia de los 17 casos incorrectos.

Además, en el caso del cuadrado superior izquierdo, clase 0, que son las predicciones correctas, presenta un color purpura intenso a diferencia de los demás.

En el cuadrado superior derecho, y el cuadrado inferior derecho, se observa un color blanco grisáceo, diferenciándose levemente del cuadrado inferior izquierdo por la cantidad de instancias.

En este modelo se observa una disminución para las instancias calificadas correctamente como clase 0 y un aumento en las instancias calificadas correctamente como clase 1 en comparación a otros modelos.

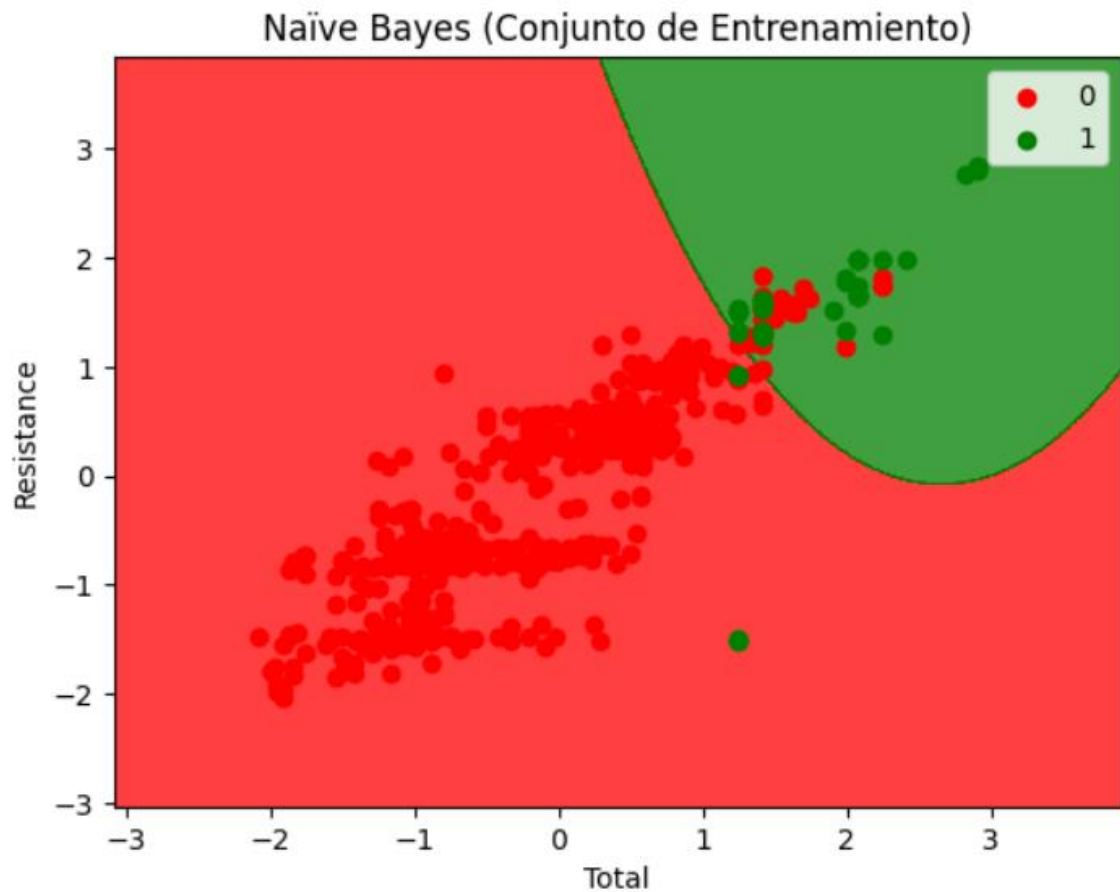
también en este modelo se observa una leve disminución para las instancias calificadas incorrectamente como clase 0, cuando en realidad son clase 1, y un leve aumento en las instancias calificadas incorrectamente como clase 1, cuando en realidad son clase 0, en comparación a otros modelos.

Esto podría considerarlo como un buen primer indicio del modelo.

Visualización del modelo de clasificación en el conjunto de Entrenamiento.

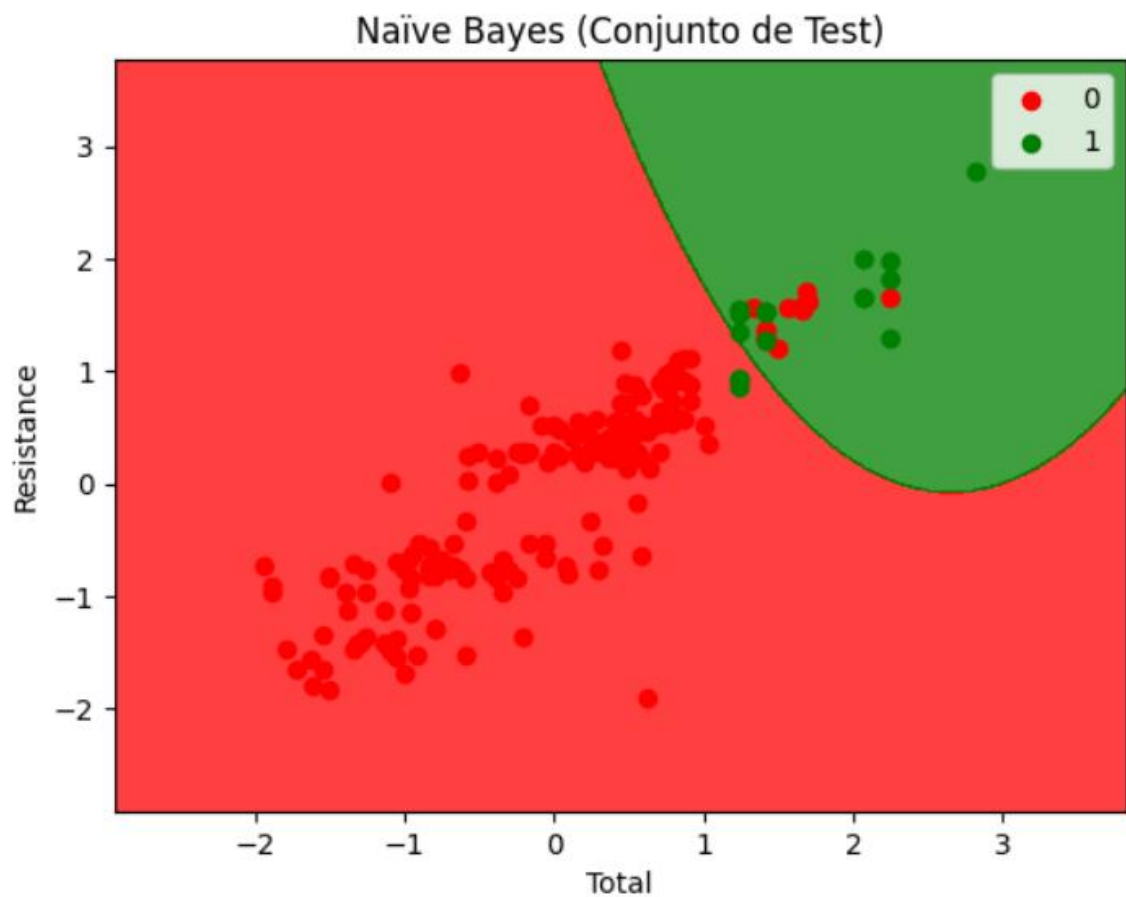
```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naïve Bayes (Conjunto de Entrenamiento)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
```

✓ 0.2s



Visualización del modelo de clasificación en el conjunto de Test.

```
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naïve Bayes (Conjunto de Test)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
✓ 0.1s
```



Ahora analizaré diferentes métricas de evaluación del rendimiento de un modelo de clasificación.

**Precisión del modelo:** La precisión mide la proporción de instancias positivas que fueron correctamente clasificadas como positivas.

**Exactitud del modelo:** La exactitud es la proporción total de predicciones correctas realizadas por el modelo.

**Sensibilidad del modelo:** La sensibilidad, también conocida como tasa de verdaderos positivos o recall, mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo.

**Puntaje F1 del modelo:** El puntaje F1 es una medida que combina la precisión y la sensibilidad en un solo valor. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad.

**Curva ROC - AUC del modelo:** La curva ROC, y el área bajo la curva (AUC) son métricas utilizadas para evaluar el rendimiento de un modelo de clasificación en problemas de clasificación binaria. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de clasificación. El AUC mide la capacidad del modelo para distinguir entre las clases.

```
print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
print(f'Exactitud del modelo: {accuracy_score(y_test, y_pred):.2f}')
print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
print(f'Puntaje F1 del modelo: {f1_score(y_test, y_pred):.2f}')
print(f'Curva ROC - AUC del modelo: {roc_auc_score(y_test, y_pred):.2f}')
```

✓ 0.0s

```
Precisión del modelo: 0.52
Exactitud del modelo: 0.92
Sensibilidad del modelo: 0.83
Puntaje F1 del modelo: 0.64
Curva ROC - AUC del modelo: 0.88
```

Imprimo las métricas de evaluación del modelo proporcionan información adicional sobre el rendimiento del modelo en términos de precisión, exactitud, sensibilidad, puntaje F1 y el área bajo la curva ROC-AUC.

```
print(classification_report(y_test, y_pred))
```

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.98      | 0.92   | 0.95     | 182     |
| 1            | 0.52      | 0.83   | 0.64     | 18      |
| accuracy     |           |        | 0.92     | 200     |
| macro avg    | 0.75      | 0.88   | 0.80     | 200     |
| weighted avg | 0.94      | 0.92   | 0.92     | 200     |

## Definiciones de otros datos

Macro avg: Calcula el promedio sin tener en cuenta el desequilibrio en el número de muestras de cada clase.

Para cada métrica, se calcula la media de los valores para todas las clases y se informa como el promedio macro avg.

Se trata de un promedio no ponderado en el que todas las clases tienen el mismo peso.

Este enfoque considera que todas las clases son igualmente importantes y le asigna la misma importancia a cada clase al calcular el promedio.

Weighted avg: Calcula el promedio ponderado teniendo en cuenta el número de muestras en cada clase.

Para cada métrica, se calcula la media ponderada de los valores para todas las clases y se informa como el promedio weighted avg.

En este enfoque, se asigna un peso proporcional al número de muestras de cada clase.

Las clases con más muestras tienen un impacto mayor en el cálculo del promedio.

Este enfoque es útil cuando hay un desequilibrio significativo en el número de muestras entre las clases.

Para la conclusión me centraré en los valores específicos obtenidos de cada clase en lugar de los promedios macro avg y weighted avg.

Los valores específicos de cada clase proporcionan información más detallada y relevante sobre cómo el modelo está clasificando cada clase individualmente.

El enfoque en los valores de cada clase es considerablemente más importante cuando hay un desequilibrio significativo en el número de muestras entre las clases.

En estos casos, los promedios pueden verse afectados por la clase dominante y no reflejar adecuadamente el rendimiento del modelo en las clases minoritarias.

Por lo tanto, los valores específicos de cada clase ofrecen una visión más detallada y proporcionan mayor eficiencia en las conclusiones del rendimiento del modelo de clasificación.

## Conclusión

Dado los resultados obtenidos con las diferentes métricas e informes puedo sacar las siguientes conclusiones adicionales.

El modelo tiene una precisión muy alta para la clase 0 con un 98% lo cual es algo altamente positivo.

Sin embargo, con respecto a la clase 1 el porcentaje de precisión baja considerablemente teniendo apenas un 52% de precisión.

La exactitud o accuracy, devolvió un porcentaje muy elevado del 92% lo que significa que un muy alto número de instancias se clasificaron correctamente en general sobre el total de instancias.

En general, un mayor valor de exactitud o accuracy indica un mejor rendimiento del modelo, pero también puede ser engañoso en casos de desequilibrio de clases, donde una clase dominante puede influir en el resultado.

Por lo tanto, es un dato que hay que tomarlo con cierto cuidado.

Otro aspecto a tener en cuenta es que la sensibilidad con respecto a la clase 0 es del 92%, por lo que el modelo es capaz de identificar de manera confiable la mayoría de las instancias de esa clase.

Mientras que en la clase 1, la sensibilidad apenas desciende hasta el valor de 83% lo que es un buen valor considerando los otros modelos donde en la misma clase descendía considerablemente.

El Puntaje F1, que es una medida que combina precisión y sensibilidad para evaluar el equilibrio entre ambas, arrojar una diferenciación marcada, 95% para la clase 0 lo cual es muy bueno, y 64% para la clase 1 siendo un porcentaje moderado.

Finalmente, el porcentaje arrojado por Curva ROC - AUC del modelo es del 88%, el cual es un valor muy por arriba del 50% (50% indica un modelo que clasifica las instancias de manera aleatoria), es un valor confiable para el rendimiento de separar las clases positivas y negativas y evitando cometer errores.

Mi conclusión de este modelo de clasificación Naïve Bayes, para este caso, es que posee un rendimiento moderado, algunos valores son relativamente altos lo que garantiza una buena confiabilidad para la clase 0.

El inconveniente se encuentra en la clase 1, ya que pese a tener un buen valor de sensibilidad, el valor de precisión es algo deficiente y valor del Puntaje F1 también podría mejorar.

Por esta razón, para este caso, mi conclusión es que presenta un rendimiento mejor que los modelos de SVM y Regresión Logística, pero aun así está por debajo de la eficiencia de los valores obtenidos en el modelo de K-Nearest Neighbors.

## Árboles de decisión

Es un método de aprendizaje automático supervisado que utiliza una estructura de árbol para tomar decisiones de clasificación.

Un árbol de decisión es una estructura jerárquica compuesta por nodos de decisión y nodos de hoja. Cada nodo de decisión representa una pregunta o una prueba sobre una característica, y cada rama representa una posible respuesta o valor. Los nodos de hoja representan las clases o categorías finales a las que se clasificarán las instancias.

Este algoritmo se basa en clasificar variables categóricas. Las predicciones se basan en combinaciones de valores en los campos de entrada.

A través de iteraciones va realizando cortes en el conjunto de datos.

El criterio mediante el cual se divide una rama en dos nodos hoja, es el índice de Gini. La división se hace de tal modo que se maximice el número de categorías dentro de cada una de estas divisiones que se van a realizar. La división intenta minimizar la entropía.

Cuánto más homogéneo es el grupo, más se reduce la entropía desde el nodo padre al nodo hijo. Si la entropía en un nodo hijo llega a ser cero significa que el grupo es totalmente homogéneo y que, en consecuencia, esa rama del árbol es capaz de clasificar los usuarios, con un 100% de seguridad, en la categoría que corresponda.

Importo las librerías que me brinda Python.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, \
recall_score, f1_score, roc_auc_score, classification_report
import warnings
```

✓ 0.0s

Realizo la importación de los datos.

Ignoro las advertencias generadas por el módulo warnings en Python.

```
warnings.filterwarnings("ignore")
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()
```

✓ 0.0s

| #   | Name                  | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | Winbattle | Resistance |
|-----|-----------------------|--------|--------|-------|----|--------|---------|---------|---------|-------|------------|-----------|-----------|------------|
| 0 1 | Bulbasaur             | Grass  | Poison | 318   | 45 | 49     | 49      | 65      | 65      | 45    | 1          | False     | 35        | 450        |
| 1 2 | Ivysaur               | Grass  | Poison | 405   | 60 | 62     | 63      | 80      | 80      | 60    | 1          | False     | 50        | 730        |
| 2 3 | Venusaur              | Grass  | Poison | 525   | 80 | 82     | 83      | 100     | 100     | 80    | 1          | False     | 75        | 940        |
| 3 3 | VenusaurMega Venusaur | Grass  | Poison | 625   | 80 | 100    | 123     | 122     | 120     | 80    | 1          | False     | 86        | 1110       |
| 4 4 | Charmander            | Fire   | NaN    | 309   | 39 | 52     | 43      | 60      | 50      | 65    | 1          | False     | 40        | 500        |

Para este análisis tomaré los valores de Total y Resistance para la variable X independiente. También tomaré los valores de Legendary para la variable Y dependiente.

Consulto si hay algún valor null (NaN), me devolverá un booleano que en caso de ser true es porque si hay valores Nan.

```
df.isnull().values.any()
```

✓ 0.0s

True

Obtuve un valor True por lo que es necesario verificar en donde se encuentran los valores nulos.

Averiguo en que columnas hay valores null.

```
df.isnull().any()
```

[7] ✓ 0.0s

```
... # False
    Name False
    Type 1 False
    Type 2 True
    Total False
    HP False
    Attack False
    Defense False
    Sp. Atk False
    Sp. Def False
    Speed False
    Generation False
    Legendary False
    Winbattle False
    Resistance False
    dtype: bool
```



Constato que únicamente hay valores null en la columna Type2, por lo que no hay inconvenientes en tomar los datos de Total, Resistance y Legendary.

Coloco los datos de las columnas Total y Resistance en la variable X.

Coloco los datos de la columna Legendary en la variable y.

```
X = df.iloc[:, [4,14]].values
y = df.iloc[:, 12].values
X,y
```

[9] ✓ 0.0s

```
... (array([[ 318, 450],
[ 405, 730],
[ 525, 940],
...,
[ 600, 1095],
[ 680, 1135],
[ 600, 1030]], dtype=int64),
array([False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, True, True, True, False, False, False,
...
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
```

Utilizo LabelEncoder para convertir a números los datos categóricos de la columna Legendary almacenados en la variable y.

[illegible]



Ajusto el modelo de clasificación con el conjunto de entrenamiento.

Esto se hace para hacer predicciones en los datos de prueba y mostrar una tabla que compara las etiquetas reales con las etiquetas predichas para una muestra de prueba limitada.

```

classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).head(100)

```

✓ 0.0s

|     | Actual | Predicción |
|-----|--------|------------|
| 0   | 0      | 0          |
| 1   | 0      | 0          |
| 2   | 0      | 0          |
| 3   | 0      | 0          |
| 4   | 0      | 0          |
| ... | ...    | ...        |
| 95  | 0      | 0          |
| 96  | 0      | 0          |
| 97  | 1      | 0          |
| 98  | 0      | 0          |
| 99  | 0      | 0          |

100 rows × 2 columns

Construyo la matriz de confusión, la cual es una técnica muy potente.

Se calcula sobre el conjunto de testing y se puede observar si las predicciones que ha realizado el algoritmo son potentes

```

cm = confusion_matrix(y_test, y_pred)
cm

```

✓ 0.0s

```

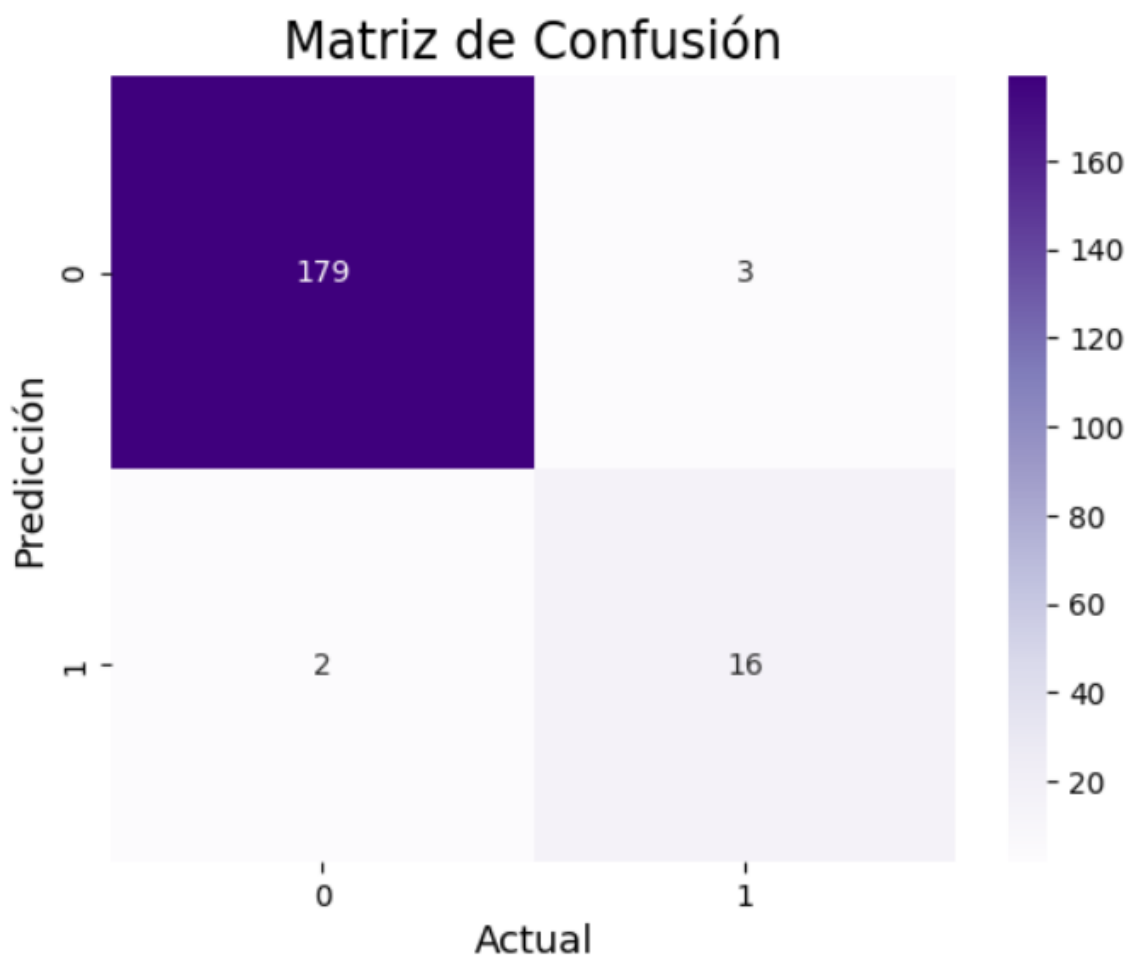
array([[179,  3],
       [ 2, 16]], dtype=int64)

```

Visualizo la matriz de confusión de un modelo de clasificación utilizando la biblioteca Seaborn y Matplotlib en Python.

La matriz de confusión habla de qué tan bien ha evaluado el algoritmo la clasificación, en este caso, de los pokemones legendarios o no legendarios con respecto a las propiedades de Total y Resitence.

```
sns.heatmap(cm,  
            annot=True,  
            fmt='g',  
            cmap='Purples')  
plt.ylabel('Predicción',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Matriz de Confusión',fontsize=17)  
plt.show()  
✓ 0.1s
```



Las predicciones correctas son los valores numéricos que se encuentran en el cuadrado superior a la izquierda junto con las que se encuentran en el cuadrado inferior a la derecha.

Mientras que las predicciones incorrectas son los valores numéricos que se encuentran en el cuadrado superior a la derecha junto con las que se encuentran en el cuadrado inferior a la izquierda.

## OBESERVACIÓN

El cuadrado superior izquierdo con un valor de 179 representa la cantidad de instancias clasificadas correctamente como clase 0.

El cuadrado superior derecho con un valor de 3 representa la cantidad de instancias clasificadas incorrectamente como clase 1 cuando en realidad son de clase 0.

El cuadrado inferior izquierdo con un valor de 2 representa la cantidad de instancias clasificadas incorrectamente como clase 0 cuando en realidad son de clase 1.

El cuadrado inferior derecho con un valor de 16 representa la cantidad de instancias clasificadas correctamente como clase 1.

Noto que hay un interesante y muy alto número de predicciones calificadas como correctas, 195 aplicando la suma de los valores correctos, a diferencia de los apenas 5 casos incorrectos.

Además, en el caso del cuadrado superior izquierdo, clase 0, que son las predicciones correctas, presenta un color purpura intenso a diferencia de los demás.

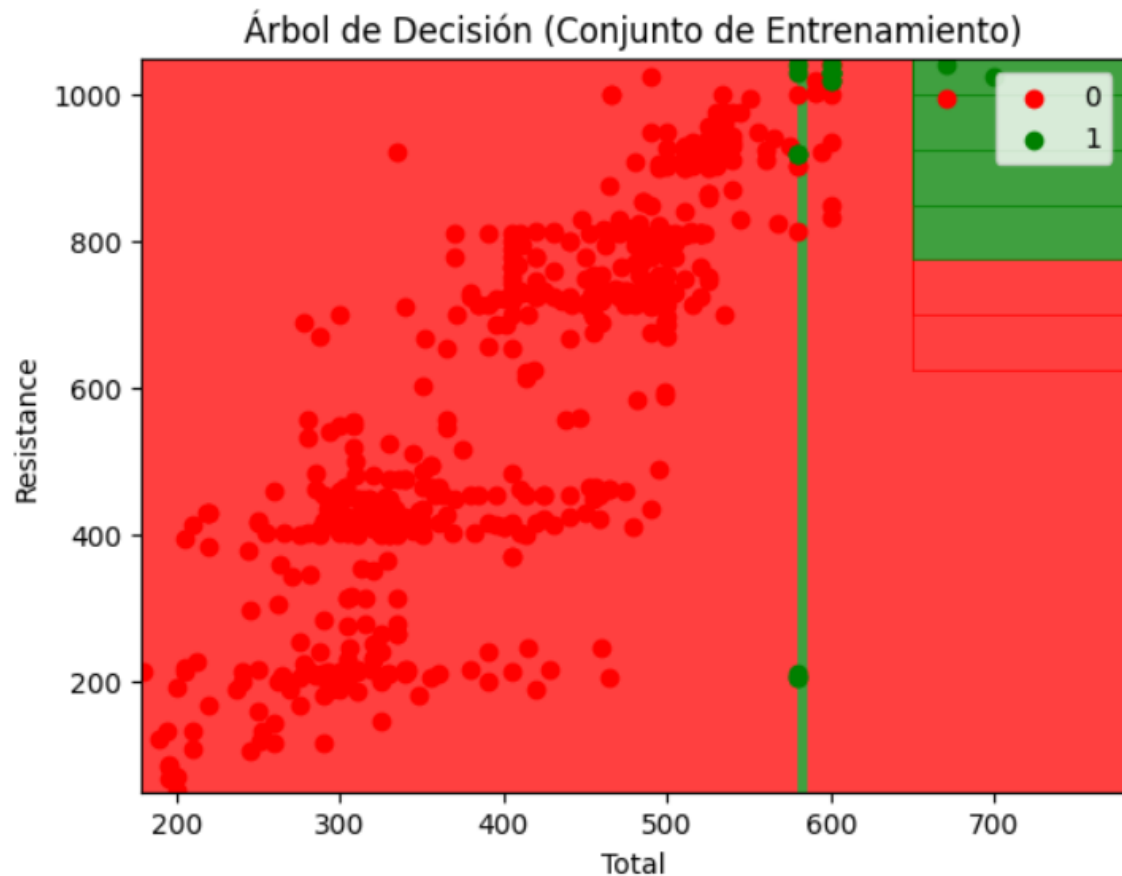
Para el caso del cuadrado inferior derecho, clase 1, se nota un blanco grisáceo que lo diferencia de los blancos para las instancias clasificadas incorrectamente.

Esto podría considerarlo como un excelente primer indicio del modelo.

Visualización del modelo de clasificación en el conjunto de Entrenamiento.

```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 1),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 500))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Árbol de Decisión (Conjunto de Entrenamiento)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
```

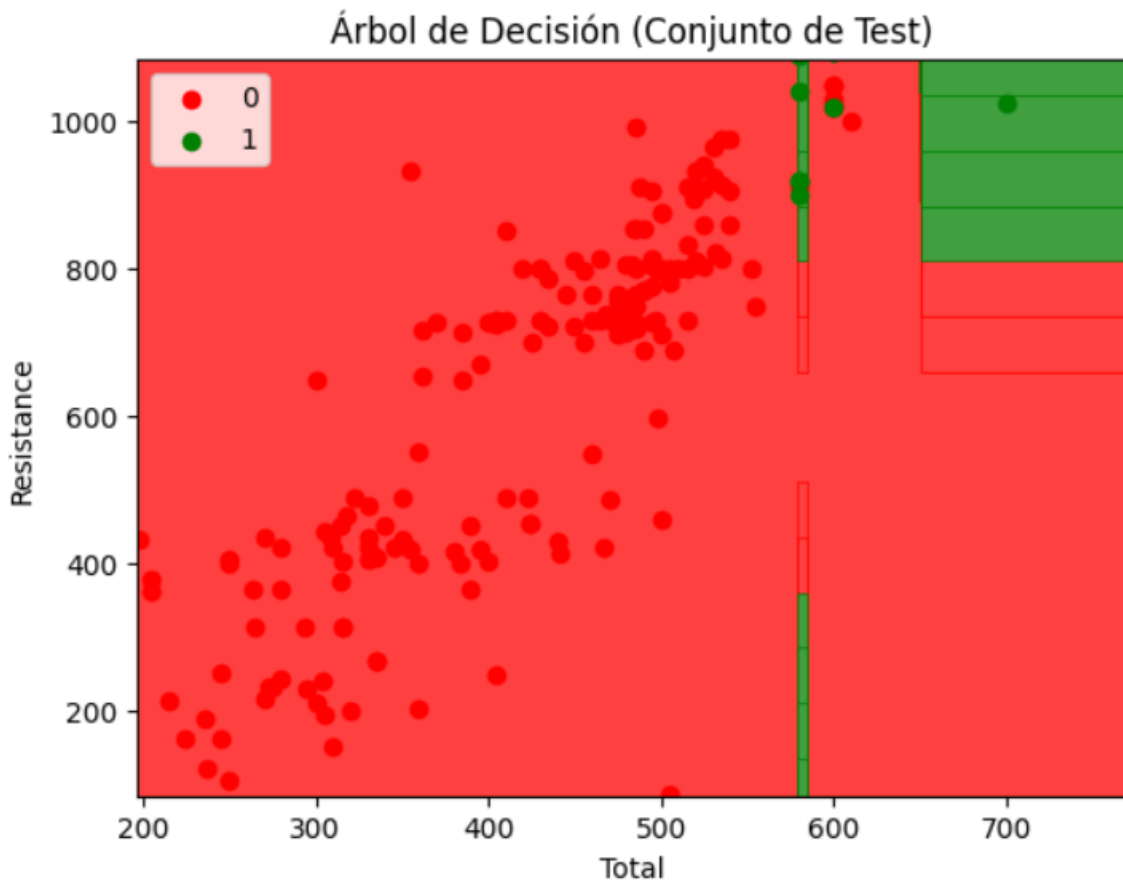
✓ 0.1s



Visualización del modelo de clasificación en el conjunto de Test.

```
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 1),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 500))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Árbol de Decisión (Conjunto de Test)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
```

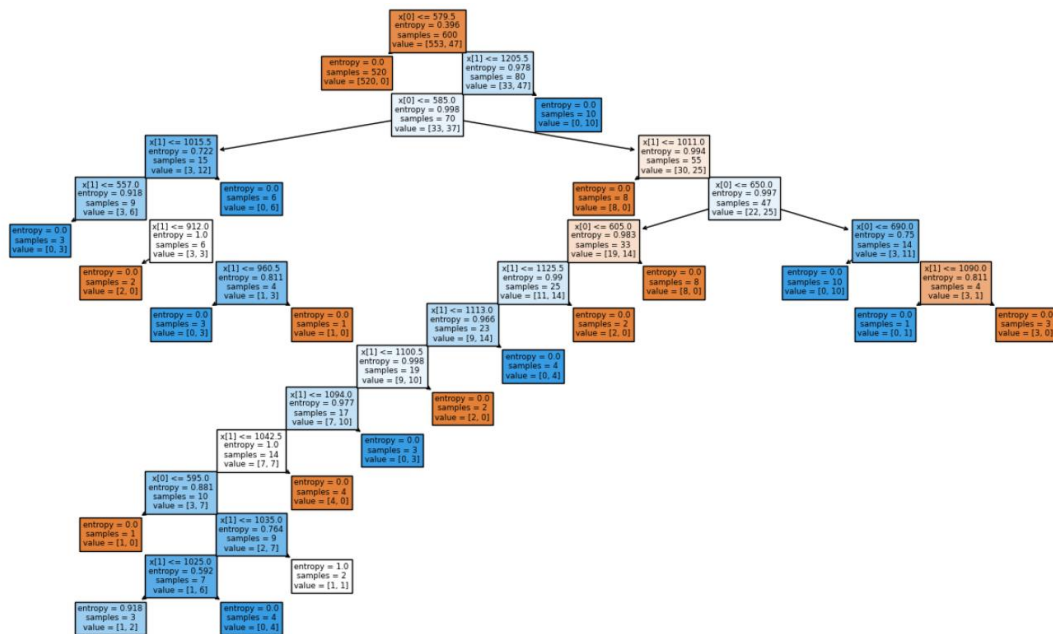
✓ 0.1s



Se obtuvo un límite de predicción compuesto por varias líneas horizontales y líneas verticales.

```
plt.figure(figsize = (16, 9))
plot_tree(classifier, filled=True)
plt.show()
```

✓ 0.6s



Determino profundidad del árbol y números de nodos terminales.

```
print(f"Profundidad del árbol: {classifier.get_depth()}")
print(f"Número de nodos terminales: {classifier.get_n_leaves()}")
```

✓ 0.0s

Profundidad del árbol: 14  
Número de nodos terminales: 21



Ahora analizaré diferentes métricas de evaluación del rendimiento de un modelo de clasificación.

**Precisión del modelo:** La precisión mide la proporción de instancias positivas que fueron correctamente clasificadas como positivas.

**Exactitud del modelo:** La exactitud es la proporción total de predicciones correctas realizadas por el modelo.

**Sensibilidad del modelo:** La sensibilidad, también conocida como tasa de verdaderos positivos o recall, mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo.

**Puntaje F1 del modelo:** El puntaje F1 es una medida que combina la precisión y la sensibilidad en un solo valor. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad.

**Curva ROC - AUC del modelo:** La curva ROC, y el área bajo la curva (AUC) son métricas utilizadas para evaluar el rendimiento de un modelo de clasificación en problemas de clasificación binaria. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de clasificación. El AUC mide la capacidad del modelo para distinguir entre las clases.

```
print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
print(f'Exactitud del modelo: {accuracy_score(y_test, y_pred):.2f}')
print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
print(f'Puntaje F1 del modelo: {f1_score(y_test, y_pred):.2f}')
print(f'Curva ROC - AUC del modelo: {roc_auc_score(y_test, y_pred):.2f}')
```

✓ 0.0s

```
Precisión del modelo: 0.84
Exactitud del modelo: 0.97
Sensibilidad del modelo: 0.89
Puntaje F1 del modelo: 0.86
Curva ROC - AUC del modelo: 0.94
```

Imprimo las métricas de evaluación del modelo proporcionan información adicional sobre el rendimiento del modelo en términos de precisión, exactitud, sensibilidad, puntaje F1 y el área bajo la curva ROC-AUC.

```
print(classification_report(y_test, y_pred))
```

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.99      | 0.98   | 0.99     | 182     |
| 1            | 0.84      | 0.89   | 0.86     | 18      |
| accuracy     |           |        | 0.97     | 200     |
| macro avg    | 0.92      | 0.94   | 0.93     | 200     |
| weighted avg | 0.98      | 0.97   | 0.98     | 200     |

Impresión del accuracy test.

```
accuracy = accuracy_score(
    y_true = y_test,
    y_pred = classifier.predict(X = X_test),
    normalize = True
)
print(f"El accuracy de test es: {100 * accuracy} %")
```

✓ 0.0s

El accuracy de test es: 97.5 %

Definiciones de otros datos

Macro avg: Calcula el promedio sin tener en cuenta el desequilibrio en el número de muestras de cada clase.

Para cada métrica, se calcula la media de los valores para todas las clases y se informa como el promedio macro avg.

Se trata de un promedio no ponderado en el que todas las clases tienen el mismo peso.

Este enfoque considera que todas las clases son igualmente importantes y le asigna la misma importancia a cada clase al calcular el promedio.

Weighted avg: Calcula el promedio ponderado teniendo en cuenta el número de muestras en cada clase.

Para cada métrica, se calcula la media ponderada de los valores para todas las clases y se informa como el promedio weighted avg.

En este enfoque, se asigna un peso proporcional al número de muestras de cada clase.

Las clases con más muestras tienen un impacto mayor en el cálculo del promedio.

Este enfoque es útil cuando hay un desequilibrio significativo en el número de muestras entre las clases.

Para la conclusión me centraré en los valores específicos obtenidos de cada clase en lugar de los promedios macro avg y weighted avg.

Los valores específicos de cada clase proporcionan información más detallada y relevante sobre cómo el modelo está clasificando cada clase individualmente.

El enfoque en los valores de cada clase es considerablemente más importante cuando hay un desequilibrio significativo en el número de muestras entre las clases.

En estos casos, los promedios pueden verse afectados por la clase dominante y no reflejar adecuadamente el rendimiento del modelo en las clases minoritarias.

Por lo tanto, los valores específicos de cada clase ofrecen una visión más detallada y proporcionan mayor eficiencia en las conclusiones del rendimiento del modelo de clasificación.

## Conclusión

Dado los resultados obtenidos con las diferentes métricas e informes puedo sacar las siguientes conclusiones adicionales.

El modelo tiene una precisión casi perfecta para la clase 0 con un 99% lo cual es algo altamente positivo.

Con respecto a la clase 1 el porcentaje de precisión también es alto teniendo un valor de precisión del 84%.

La exactitud o accuracy, devolvió un porcentaje muy elevado del 97% lo que significa que un muy alto número de instancias se clasificaron correctamente en general sobre el total de instancias.

En general, un mayor valor de exactitud o accuracy indica un mejor rendimiento del modelo, pero también puede ser engañoso en casos de desequilibrio de clases, donde una clase dominante puede influir en el resultado.

Por lo tanto, es un dato que hay que tomarlo con cierto cuidado.

Otro aspecto a tener en cuenta es que la sensibilidad con respecto a la clase 0 es del 98%, por lo que el modelo es capaz de identificar de manera excelente y confiable la mayoría de las instancias de esa clase.

En la clase 1, la sensibilidad es del 89%, es un número muy alto también, de los más elevado hasta el momento, entre todos los modelos para este aspecto, junto al modelo K-Nearest Neighbors.

El Puntaje F1, que es una medida que combina precisión y sensibilidad para evaluar el equilibrio entre ambas, arroja una imperceptible diferenciación, 99% para la clase 0 lo cual es casi perfecto, y 86% para la clase 1 siendo un porcentaje muy alto.

Finalmente, el porcentaje arrojado por Curva ROC - AUC del modelo es del 94%, el cual es un valor muy por arriba del 50% (50% indica un modelo que clasifica las instancias de manera aleatoria), y podemos determinar que separa las clases positivas y negativas de manera muy buena y efectiva.

Mi conclusión de este modelo de clasificación de Árbol de Decisión, para este caso particular, es que posee un rendimiento excelente.

Los valores obtenidos son muy altos para la clase 0, y en especial para la clase 1 que es donde se encontraban las mayores dificultades de los anteriores modelos para este caso.

Hasta el momento es el modelo con mayor confiabilidad y eficiencia.

### Pruning (podado del árbol)

Con el objetivo de identificar la profundidad óptima, que consigue reducir la varianza y aumentar la capacidad predictiva del modelo, se somete al árbol a un proceso de pruning por búsqueda por validación cruzada.

Se utiliza el parámetro `ccp_alpha`: flotante no negativo, predeterminado = 0.0, es un parámetro de complejidad utilizado para la poda de costo y complejidad mínima.

```
param_grid = {'ccp_alpha': np.linspace(0, 5, 10)}
param_grid
```

✓ 0.0s

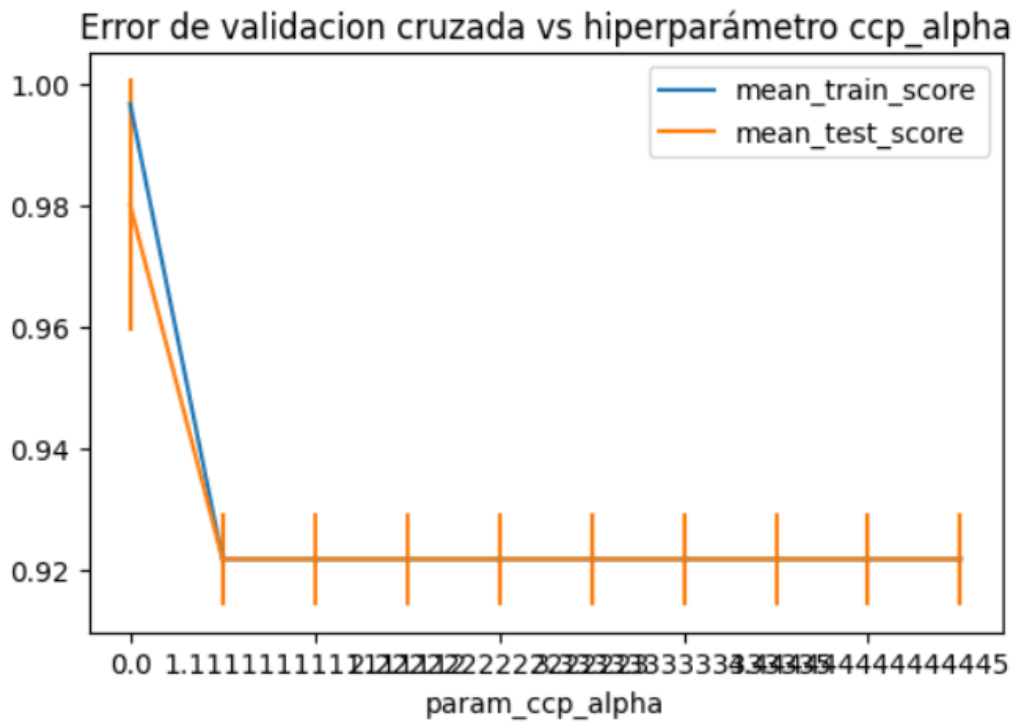
```
{'ccp_alpha': array([0.          , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.          ])}
```

Una vez identificado el valor óptimo de `ccp_alpha`, se reentrena el árbol indicando este valor en sus argumentos. Si en el `GridSearchCV()` se indica `refit=True`, este reentrenamiento se hace automáticamente y el modelo resultante se encuentra almacenado en `.best_estimator_`.

Visualizo el Error de validación cruzada vs hiperparámetro `ccp_alpha`

```
grid = GridSearchCV(
    estimator = DecisionTreeClassifier(random_state=123),
    param_grid = param_grid,
    scoring = 'accuracy',
    cv = 10,
    refit = True,
    return_train_score = True
)
grid.fit(X_train, y_train)
fig, ax = plt.subplots(figsize=(6, 3.84))
scores = pd.DataFrame(grid.cv_results_)
scores.plot(x='param_ccp_alpha', y='mean_train_score', yerr='std_train_score', ax=ax)
scores.plot(x='param_ccp_alpha', y='mean_test_score', yerr='std_test_score', ax=ax)
ax.set_title("Error de validacion cruzada vs hiperparámetro ccp_alpha");
```

✓ 0.2s



Determino el mejor valor ccp\_alpha encontrado.

```
grid.best_params_
✓ 0.0s
{'ccp_alpha': 0.0}
```

Estructura del árbol final

```
dtc_final = grid.best_estimator_
print(f"Profundidad del árbol: {dtc_final.get_depth()}")
print(f"Número de nodos terminales: {dtc_final.get_n_leaves()}")
✓ 0.0s

Profundidad del árbol: 10
Número de nodos terminales: 22
```

Noto que hay una variación ahora en la profundidad de árbol con 10 cuando antes era de 14.

también aumento un Número de nodos terminales obteniendo 22 cuando antes era 21.

Impresión del accuracy test nuevamente.

```

predicc = dtc_final.predict(X = X_test)
accuracy = accuracy_score(
    y_true = y_test,
    y_pred = predicc,
    normalize = True
)
print(f"El accuracy de test es: {100 * accuracy} %")
✓ 0.0s
El accuracy de test es: 97.5 %

```

El accuracy de test no presenta variación.

Imprimo nuevamente las métricas de evaluación del rendimiento de un modelo de clasificación.

**Precisión del modelo:** La precisión mide la proporción de instancias positivas que fueron correctamente clasificadas como positivas.

**Exactitud del modelo:** La exactitud es la proporción total de predicciones correctas realizadas por el modelo.

**Sensibilidad del modelo:** La sensibilidad, también conocida como tasa de verdaderos positivos o recall, mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo.

**Puntaje F1 del modelo:** El puntaje F1 es una medida que combina la precisión y la sensibilidad en un solo valor. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad.

**Curva ROC - AUC del modelo:** La curva ROC, y el área bajo la curva (AUC) son métricas utilizadas para evaluar el rendimiento de un modelo de clasificación en problemas de clasificación binaria. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de clasificación. El AUC mide la capacidad del modelo para distinguir entre las clases.

```

print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
print(f'Exactitud del modelo:{accuracy_score(y_test, y_pred):.2f}')
print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
print(f'Puntaje F1 del modelo:{f1_score(y_test, y_pred):.2f}')
print(f'Curva ROC - AUC del modelo:{roc_auc_score(y_test, y_pred):.2f}')
✓ 0.0s
Precisión del modelo: 0.84
Exactitud del modelo:0.97
Sensibilidad del modelo: 0.89
Puntaje F1 del modelo:0.86
Curva ROC - AUC del modelo:0.94

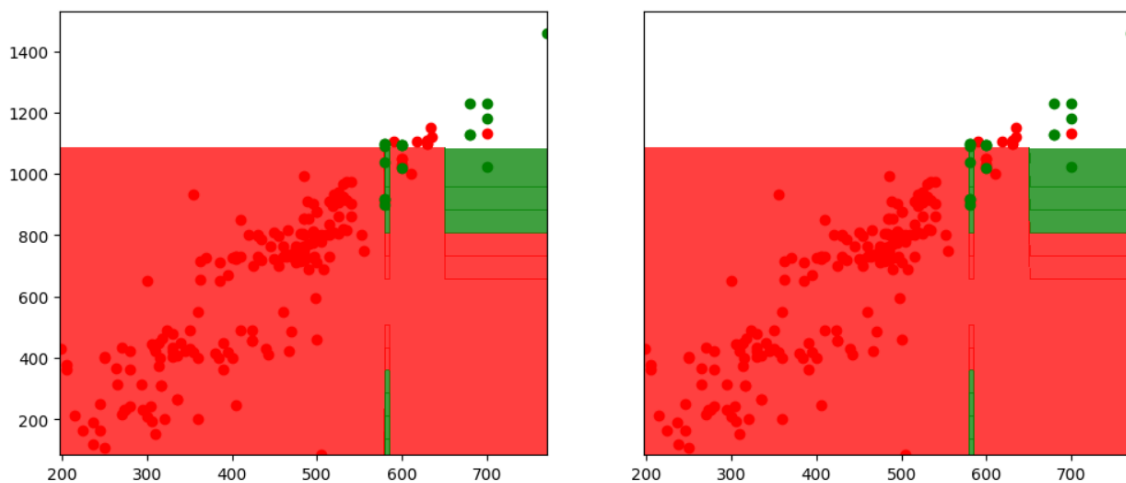
```

comparación de gráficos.

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,5))
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 1),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 500))
ax1.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
for i, j in enumerate(np.unique(y_set)):
    ax1.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
color = ListedColormap(('red', 'green'))(i), label = j)
ax2.contourf(X1, X2, dtc_final.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
for i, j in enumerate(np.unique(y_set)):
    ax2.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
color = ListedColormap(('red', 'green'))(i), label = j)
ax2.yaxis.set_ticks([])
fig.suptitle('Comparando gráficos')
plt.show()
```

✓ 0.1s

Comparando gráficos



Una breve conclusión del método aplicado Pruning, podado del árbol, es que no mejoró los valores de las métricas, por lo que es un rendimiento idéntico.

La comparación de los gráficos muestra la misma observación.

## Random Forest

Es una extensión del algoritmo de árbol de decisión que utiliza la combinación de múltiples árboles de decisión para mejorar la precisión de la clasificación.

Se selecciona un número aleatorio de puntos que pertenecen al conjunto de entrenamiento, y con esos puntos seleccionados de entre el total, se construye un árbol de decisión de manera que se aplica la técnica Random Forest.

La diferencia es que, cada uno de esos árboles, elabora la predicción sobre la categoría a la cual pertenece el punto.

Luego se somete a voto, de modo que la categoría que obtiene más votos por parte de los árboles es la categoría a la cual se clasifica el dato

Importo las librerías que me brinda Python.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, \
recall_score, f1_score, roc_auc_score, classification_report
import warnings

✓ 0.1s
```

Realizo la importación de los datos.

Ignoro las advertencias generadas por el módulo warnings en Python.

```
warnings.filterwarnings("ignore")
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()

✓ 0.0s
```

| #   | Name                  | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | Winbattle | Resistance |
|-----|-----------------------|--------|--------|-------|----|--------|---------|---------|---------|-------|------------|-----------|-----------|------------|
| 0 1 | Bulbasaur             | Grass  | Poison | 318   | 45 | 49     | 49      | 65      | 65      | 45    | 1          | False     | 35        | 450        |
| 1 2 | Ivysaur               | Grass  | Poison | 405   | 60 | 62     | 63      | 80      | 80      | 60    | 1          | False     | 50        | 730        |
| 2 3 | Venusaur              | Grass  | Poison | 525   | 80 | 82     | 83      | 100     | 100     | 80    | 1          | False     | 75        | 940        |
| 3 3 | VenusaurMega Venusaur | Grass  | Poison | 625   | 80 | 100    | 123     | 122     | 120     | 80    | 1          | False     | 86        | 1110       |
| 4 4 | Charmander            | Fire   | NaN    | 309   | 39 | 52     | 43      | 60      | 50      | 65    | 1          | False     | 40        | 500        |

Para este análisis tomaré los valores de Total y Resistance para la variable X independiente. También tomaré los valores de Legendary para la variable Y dependiente.



Consulto si hay algún valor null (NaN), me devolverá un booleano que en caso de ser true es porque si hay valores Nan.

```
df.isnull().values.any()
```

✓ 0.0s

True

Obtuve un valor True por lo que es necesario verificar en donde se encuentran los valores nulos.

Averiguo en que columnas hay valores null.

```
df.isnull().any()
```

[7] ✓ 0.0s

|            |       |
|------------|-------|
| #          | False |
| Name       | False |
| Type 1     | False |
| Type 2     | True  |
| Total      | False |
| HP         | False |
| Attack     | False |
| Defense    | False |
| Sp. Atk    | False |
| Sp. Def    | False |
| Speed      | False |
| Generation | False |
| Legendary  | False |
| Winbattle  | False |
| Resistance | False |
| dtype:     | bool  |

Constato que únicamente hay valores null en la columna Type2, por lo que no hay inconvenientes en tomar los datos de Total, Resistance y Legendary.

Coloco los datos de las columnas Total y Resistance en la variable X.

Coloco los datos de la columna Legendary en la variable y.

```
x = df.iloc[:, [4,14]].values
y = df.iloc[:, 12].values
X,y
```

[9] ✓ 0.0s

```
... (array([[ 318, 450],
[ 405, 730],
[ 525, 940],
...,
[ 600, 1095],
[ 680, 1135],
[ 600, 1030]], dtype=int64),
array([False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, True, True, True, False, False, False,
...,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
```

Utilizo LabelEncoder para convertir a números los datos categóricos de la columna Legendary almacenados en la variable y.

[illegible]

Imprimo nuevamente mis variables con sus datos.

[illegible]

Utilizo `train_test_split` para dividir un dataset en bloques.

Conjunto de testing por un lado y conjunto de entrenamiento por el otro.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

## Escales los datos (variables)

El escalado va a transformar los valores de las características de forma que estén confinados en un rango  $[a, b]$ , típicamente  $[0, 1]$  o  $[-1, 1]$ .

```
sc_X = StandardScaler()  
X_train = sc_X.fit_transform(X_train)  
X_test = sc_X.transform(X_test)
```

✓ 0.0s

Ajusto el modelo de clasificación con el conjunto de entrenamiento.

Esto se hace para hacer predicciones en los datos de prueba y mostrar una tabla que compara las etiquetas reales con las etiquetas predichas para una muestra de prueba limitada.

```

classifier = RandomForestClassifier(n_estimators = 10, criterion = "entropy", random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).head(100)

```

✓ 0.0s

|     | Actual | Predicción |
|-----|--------|------------|
| 0   | 0      | 0          |
| 1   | 0      | 0          |
| 2   | 0      | 0          |
| 3   | 0      | 0          |
| 4   | 0      | 0          |
| ... | ...    | ...        |
| 95  | 0      | 0          |
| 96  | 0      | 0          |
| 97  | 1      | 0          |
| 98  | 0      | 0          |
| 99  | 0      | 0          |

100 rows × 2 columns

Construyo la matriz de confusión, la cual es una técnica muy potente.

Se calcula sobre el conjunto de testing y se puede observar si las predicciones que ha realizado el algoritmo son potentes

```

cm = confusion_matrix(y_test, y_pred)
cm

```

✓ 0.0s

```

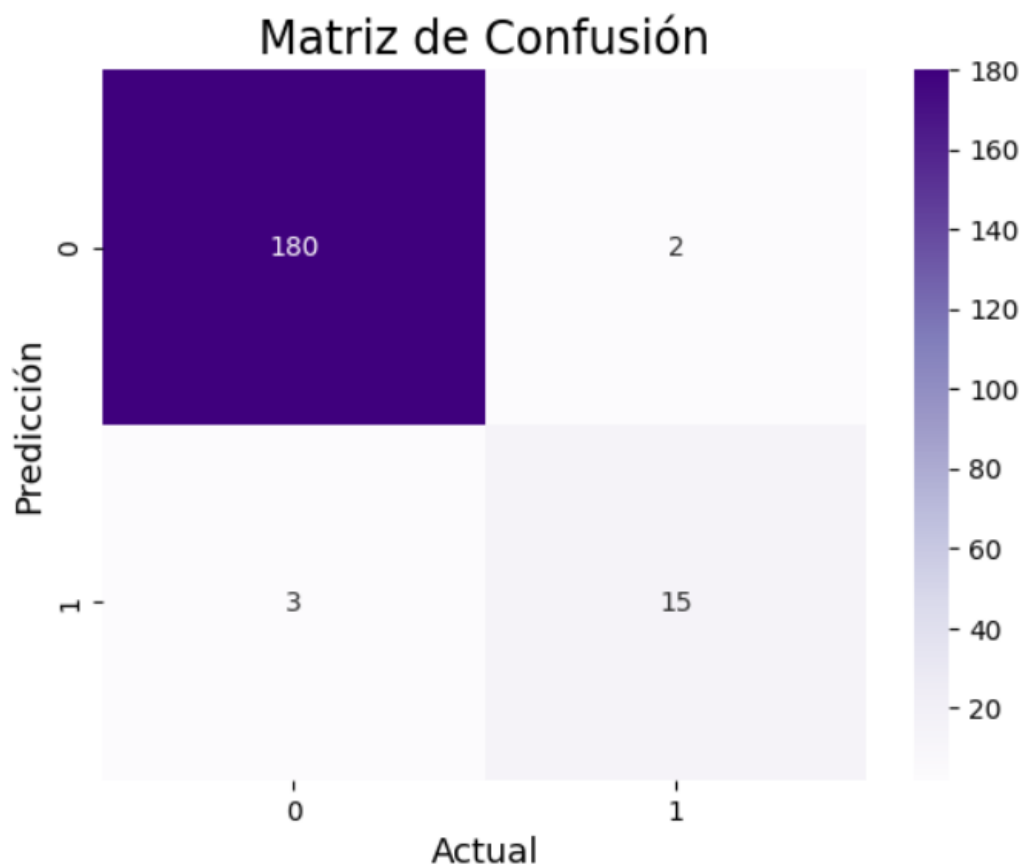
array([[180,  2],
       [ 3, 15]], dtype=int64)

```

Visualizo la matriz de confusión de un modelo de clasificación utilizando la biblioteca Seaborn y Matplotlib en Python.

La matriz de confusión habla de qué tan bien ha evaluado el algoritmo la clasificación, en este caso, de los pokemones legendarios o no legendarios con respecto a las propiedades de Total y Resitence.

```
sns.heatmap(cm,  
            annot=True,  
            fmt='g',  
            cmap='Purples')  
plt.ylabel('Predicción',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Matriz de Confusión',fontsize=17)  
plt.show()  
✓ 0.1s
```



Las predicciones correctas son los valores numéricos que se encuentran en el cuadrado superior a la izquierda junto con las que se encuentran en el cuadrado inferior a la derecha.

Mientras que las predicciones incorrectas son los valores numéricos que se encuentran en el cuadrado superior a la derecha junto con las que se encuentran en el cuadrado inferior a la izquierda.

## OBESERVACIÓN

El cuadrado superior izquierdo con un valor de 180 representa la cantidad de instancias clasificadas correctamente como clase 0.

El cuadrado superior derecho con un valor de 2 representa la cantidad de instancias clasificadas incorrectamente como clase 1 cuando en realidad son de clase 0.

El cuadrado inferior izquierdo con un valor de 3 representa la cantidad de instancias clasificadas incorrectamente como clase 0 cuando en realidad son de clase 1.

El cuadrado inferior derecho con un valor de 15 representa la cantidad de instancias clasificadas correctamente como clase 1.

Noto que hay un interesante y muy alto número de predicciones calificadas como correctas, 195 aplicando la suma de los valores correctos, a diferencia de los apenas 5 casos incorrectos.

Además, en el caso del cuadrado superior izquierdo, clase 0, que son las predicciones correctas, presenta un color purpura intenso a diferencia de los demás.

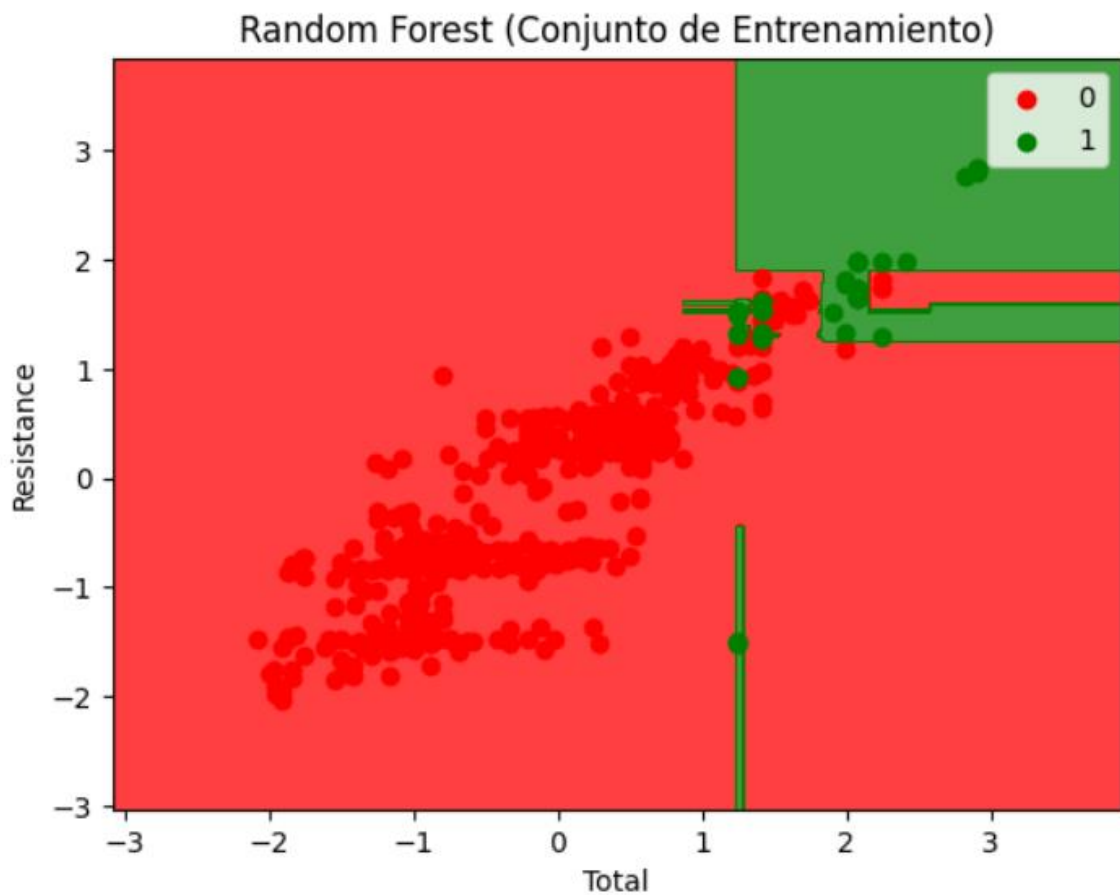
Para el caso del cuadrado inferior derecho, clase 1, se nota un blanco grisáceo que lo diferencia de los blancos para las instancias clasificadas incorrectamente.

Los valores arrojados son muy similares a la matriz de confusión del modelo de Árbol de Decisión.

Esto podría considerarlo como un excelente primer indicio del modelo.

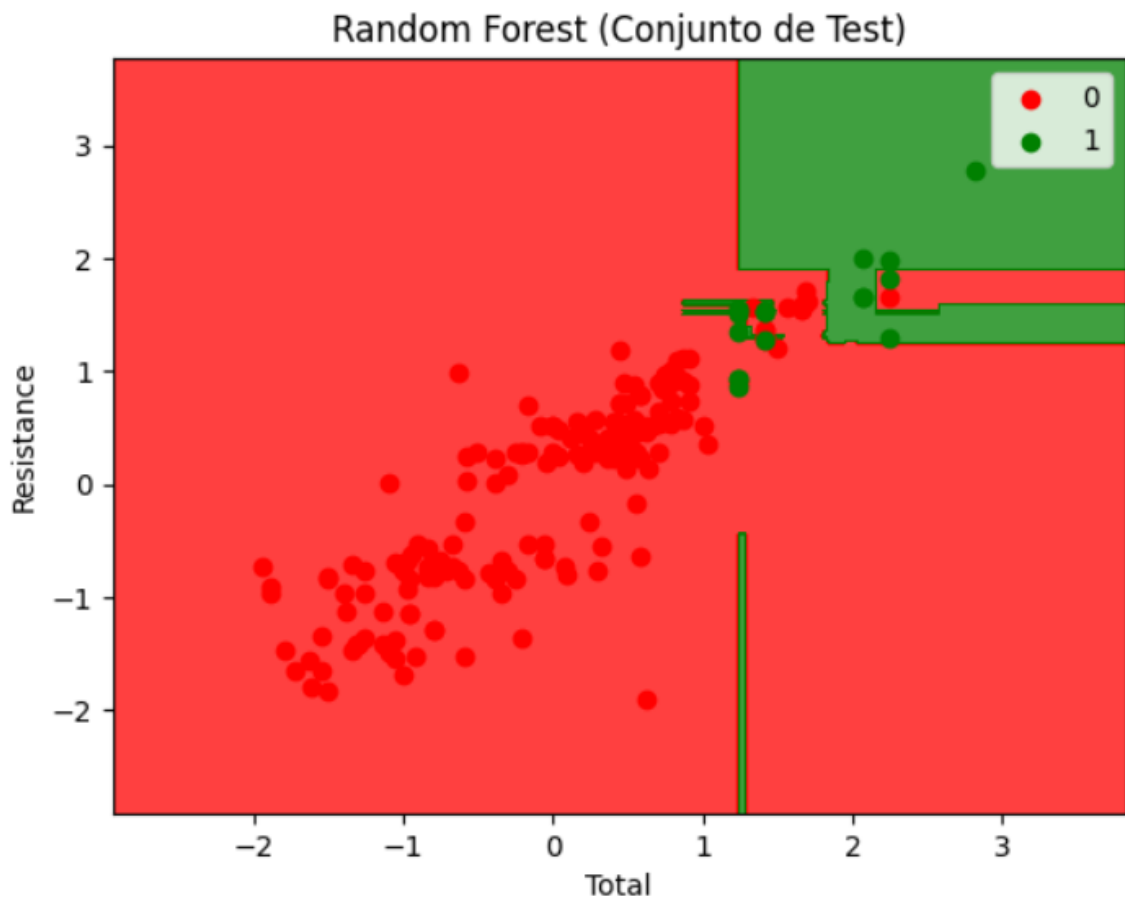
Visualización del modelo de clasificación en el conjunto de Entrenamiento.

```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest (Conjunto de Entrenamiento)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
✓ 0.3s
```



Visualización del modelo de clasificación en el conjunto de Test.

```
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest (Conjunto de Test)')
plt.xlabel('Total')
plt.ylabel('Resistance')
plt.legend()
plt.show()
✓ 0.3s
```





Ahora analizaré diferentes métricas de evaluación del rendimiento de un modelo de clasificación.

**Precisión del modelo:** La precisión mide la proporción de instancias positivas que fueron correctamente clasificadas como positivas.

**Exactitud del modelo:** La exactitud es la proporción total de predicciones correctas realizadas por el modelo.

**Sensibilidad del modelo:** La sensibilidad, también conocida como tasa de verdaderos positivos o recall, mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo.

**Puntaje F1 del modelo:** El puntaje F1 es una medida que combina la precisión y la sensibilidad en un solo valor. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad.

**Curva ROC - AUC del modelo:** La curva ROC, y el área bajo la curva (AUC) son métricas utilizadas para evaluar el rendimiento de un modelo de clasificación en problemas de clasificación binaria. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de clasificación. El AUC mide la capacidad del modelo para distinguir entre las clases.

```
print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
print(f'Exactitud del modelo: {accuracy_score(y_test, y_pred):.2f}')
print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
print(f'Puntaje F1 del modelo: {f1_score(y_test, y_pred):.2f}')
print(f'Curva ROC - AUC del modelo: {roc_auc_score(y_test, y_pred):.2f}')
```

✓ 0.0s

```
Precisión del modelo: 0.88
Exactitud del modelo: 0.97
Sensibilidad del modelo: 0.83
Puntaje F1 del modelo: 0.86
Curva ROC - AUC del modelo: 0.91
```

Imprimo las métricas de evaluación del modelo proporcionan información adicional sobre el rendimiento del modelo en términos de precisión, exactitud, sensibilidad, puntaje F1 y el área bajo la curva ROC-AUC.

```
print(classification_report(y_test, y_pred))
```

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.98      | 0.99   | 0.99     | 182     |
| 1            | 0.88      | 0.83   | 0.86     | 18      |
| accuracy     |           |        | 0.97     | 200     |
| macro avg    | 0.93      | 0.91   | 0.92     | 200     |
| weighted avg | 0.97      | 0.97   | 0.97     | 200     |

## Definiciones de otros datos

Macro avg: Calcula el promedio sin tener en cuenta el desequilibrio en el número de muestras de cada clase.

Para cada métrica, se calcula la media de los valores para todas las clases y se informa como el promedio macro avg.

Se trata de un promedio no ponderado en el que todas las clases tienen el mismo peso.

Este enfoque considera que todas las clases son igualmente importantes y le asigna la misma importancia a cada clase al calcular el promedio.

Weighted avg: Calcula el promedio ponderado teniendo en cuenta el número de muestras en cada clase.

Para cada métrica, se calcula la media ponderada de los valores para todas las clases y se informa como el promedio weighted avg.

En este enfoque, se asigna un peso proporcional al número de muestras de cada clase.

Las clases con más muestras tienen un impacto mayor en el cálculo del promedio.

Este enfoque es útil cuando hay un desequilibrio significativo en el número de muestras entre las clases.

Para la conclusión me centraré en los valores específicos obtenidos de cada clase en lugar de los promedios macro avg y weighted avg.

Los valores específicos de cada clase proporcionan información más detallada y relevante sobre cómo el modelo está clasificando cada clase individualmente.

El enfoque en los valores de cada clase es considerablemente más importante cuando hay un desequilibrio significativo en el número de muestras entre las clases.

En estos casos, los promedios pueden verse afectados por la clase dominante y no reflejar adecuadamente el rendimiento del modelo en las clases minoritarias.

Por lo tanto, los valores específicos de cada clase ofrecen una visión más detallada y proporcionan mayor eficiencia en las conclusiones del rendimiento del modelo de clasificación.

## Conclusión

Dado los resultados obtenidos con las diferentes métricas e informes puedo sacar las siguientes conclusiones adicionales.

El modelo tiene una precisión casi perfecta para la clase 0 con un 98% lo cual es algo altamente positivo.

Con respecto a la clase 1 el porcentaje de precisión también es muy alto teniendo un valor de precisión del 88%, es el valor más alto para la clase 1 entre todos los modelos.

La exactitud o accuracy, devolvió un porcentaje muy elevado del 97% lo que significa que un muy alto número de instancias se clasificaron correctamente en general sobre el total de instancias.

En general, un mayor valor de exactitud o accuracy indica un mejor rendimiento del modelo, pero también puede ser engañoso en casos de desequilibrio de clases, donde una clase dominante puede influir en el resultado.

Por lo tanto, es un dato que hay que tomarlo con cierto cuidado.

Otro aspecto a tener en cuenta es que la sensibilidad con respecto a la clase 0 es del 99%, por lo que el modelo es capaz de identificar de manera excelente y confiable la mayoría de las instancias de esa clase casi de manera perfecta.

En la clase 1, la sensibilidad es del 83%, es un número alto también comparándolo en general con otros modelos.

El Puntaje F1, que es una medida que combina precisión y sensibilidad para evaluar el equilibrio entre ambas, arroja una imperceptible diferenciación, 99% para la clase 0 lo cual es casi perfecto, y 86% para la clase 1 siendo un porcentaje muy alto.

Finalmente, el porcentaje arrojado por Curva ROC - AUC del modelo es del 91%, el cual es un valor muy por arriba del 50% (50% indica un modelo que clasifica las instancias de manera aleatoria), y podemos determinar que separa las clases positivas y negativas de manera muy buena y efectiva.

Mi conclusión de este modelo de clasificación de Random Forest, para este caso particular, es que posee un rendimiento excelente.

Los valores obtenidos son muy altos para la clase 0, y en especial para la clase 1 que es donde se encontraban las mayores dificultades de los anteriores modelos para este caso.

Es el modelo con mayor confiabilidad y eficiencia junto con el modelo de Árbol de Decisión

.

## Conclusión general final

Ya habiendo analizado este caso con diferentes algoritmos de calificación puedo sacar una conclusión final en cuanto a eficiencia y rendimiento.

Analizando Total y Resistance para la variable x, con respecto a los valores de Legendary para la variable Y dependiente, en estos modelos de calificación, puedo concluir que hay dos modelos que sobresalen de los demás dados sus valores de rendimiento devueltos en las métricas de evaluación.

Estos son los modelos de Árbol de Decisión y Random Forest, ya que presentan una alta confiabilidad tanto para la clase 0 como para la clase 1.

Cualquiera de los dos arroja resultados sobresalientes, teniendo valores de precisión, sensibilidad, puntaje F1 y Curva ROC - AUC muy similares y óptimos.

Además, quiero destacar que el modelo Árbol de Decisión tiene como positivo la Interpretabilidad, sin necesidad de escalado de características, y funciona en problemas lineales y no lineales.

El modelo Random Forest es potente y preciso, de excelente rendimiento en muchos problemas, incluidos los no lineales.

Hay otro modelo que presentó resultados relativamente aceptables y buenos para este caso como el modelo K-Nearest Neighbors, al cual ubicaría en tercer lugar dado a que, presenta algunas dificultades en algunos valores obtenidos de las métricas, pero no tan marcadas como otros modelos.

Los demás modelos analizados presentan serias inconsistencias, particularmente en los valores obtenidos para la clase 1, por lo que no los vuelve modelos confiables para este caso particular.