

TP DE REGRESIÓN

Introducción

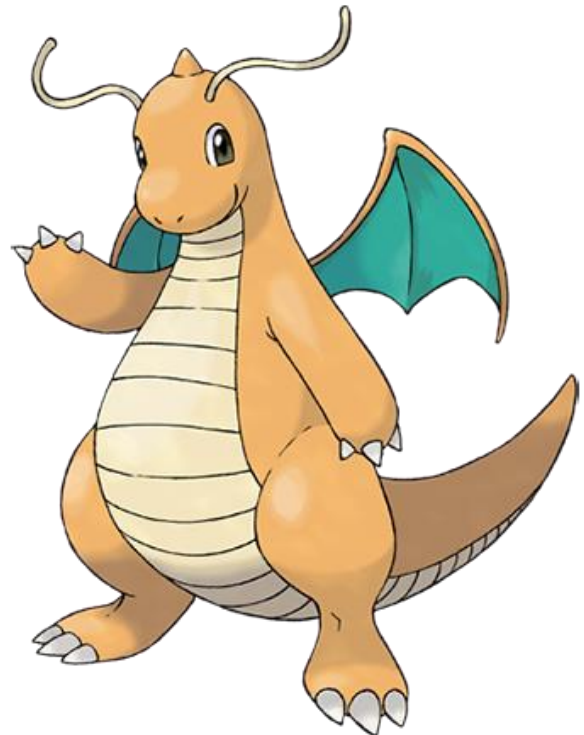
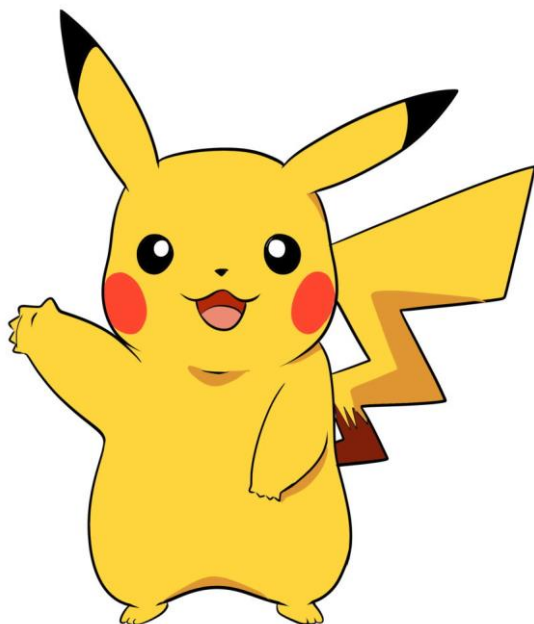
Utilizando los algoritmos de regresión y técnicas vistos, elegí tomar como dataset para este trabajo practico a los Pokémon y realizar mi análisis en función a sus diferentes atributos y características.

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.

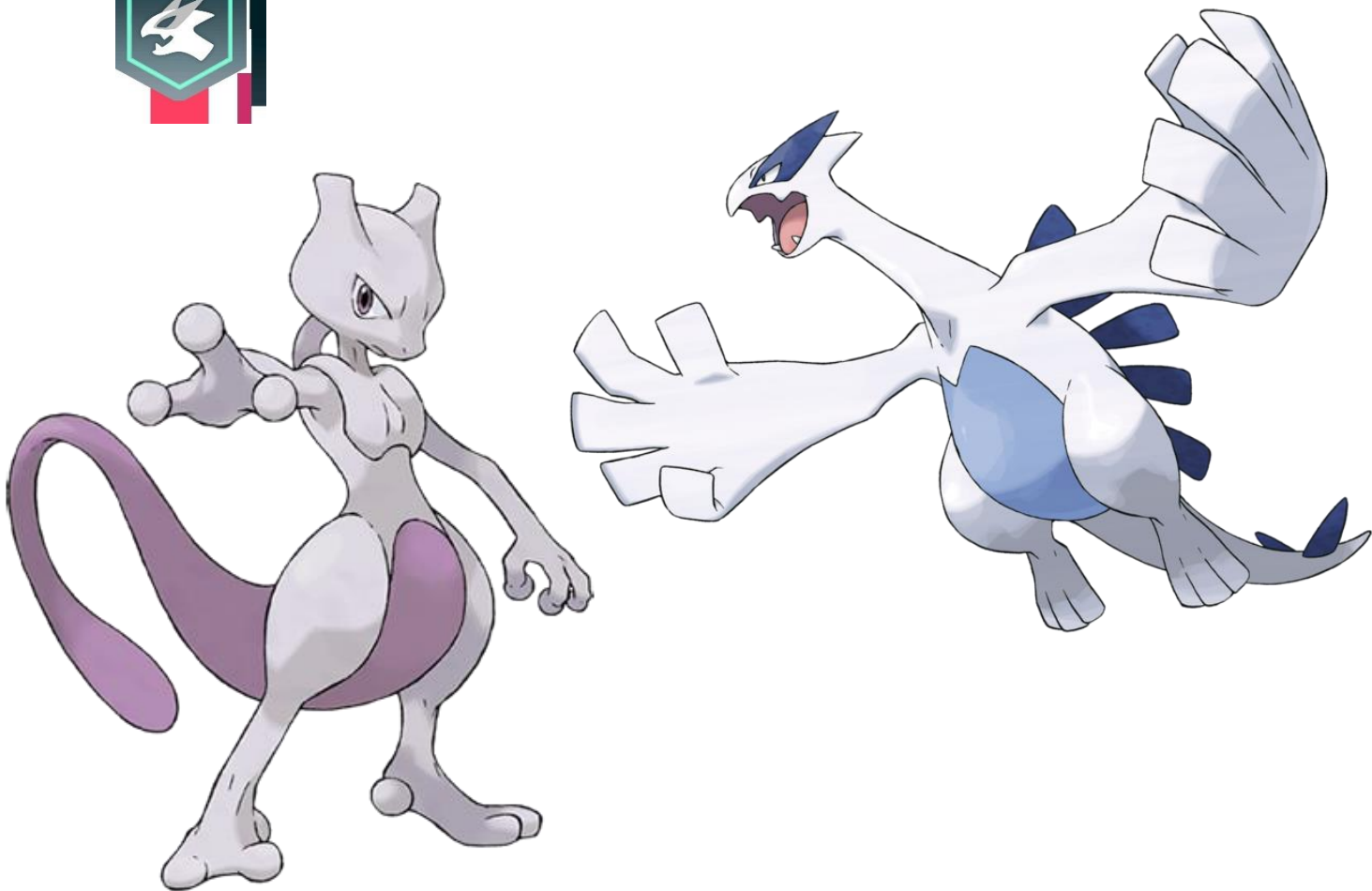
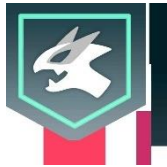


También hay otra clasificación importante que tienen los Pokémon y es que algunos tienen una rareza excepcional, a estos se los conocen como Pokémon Legendarios.

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 comienzo a importar las librerías que voy a utilizar para este proyecto como pandas, numpy, matplotlib, seaborn, 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 REGRESION utilizando los algoritmos de regresión 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 seabornInstance
import seaborn as sns
import sklearn.model_selection
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline

```

```

# Importo las librerías que me brinda Python.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
import seaborn as sns
import sklearn.model_selection
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline

```

Luego realizo la importación de los datos.

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.

```
# 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

Utilizo algunas funciones más en mi variable donde tengo alojado el dataset.

```
# Realizo la función shape para comprobar los registros y columnas de mi dataset.

df.shape
```

[137]

... (800, 15)

```
# Utilizo describe para obtener más información acerca de mi dataset.

df.describe()
```

[138]

...

	#	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Winbattle	Resistance
count	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000
mean	362.813750	435.10250	69.258750	79.001250	73.842500	72.820000	71.902500	68.277500	3.32375	49.913750	655.573750
std	208.343798	119.96304	25.534669	32.457366	31.183501	32.722294	27.828916	29.060474	1.66129	21.974301	291.629751
min	1.000000	180.00000	1.000000	5.000000	5.000000	10.000000	20.000000	5.000000	1.00000	5.000000	50.000000
25%	184.750000	330.00000	50.000000	55.000000	50.000000	49.750000	50.000000	45.000000	2.00000	30.000000	420.000000
50%	364.500000	450.00000	65.000000	75.000000	70.000000	65.000000	70.000000	65.000000	3.00000	50.000000	723.000000
75%	539.250000	515.00000	80.000000	100.000000	90.000000	95.000000	90.000000	90.000000	5.00000	65.000000	861.000000
max	721.000000	780.00000	255.000000	190.000000	230.000000	194.000000	230.000000	180.000000	6.00000	99.000000	1481.000000

En este punto es donde decido hacer un análisis de Regresión Lineal Simple entre la relación que tienen la columna Winbattle con Resistance.

Por lo cual, deseo quedarme únicamente con esas dos columnas, así que elimino las demás.

```
df.drop(["#"], axis = 1, inplace = True)
df.drop(["Name"], axis = 1, inplace = True)
df.drop(["Type 1"], axis = 1, inplace = True)
df.drop(["Type 2"], axis = 1, inplace = True)
df.drop(["Total"], axis = 1, inplace = True)
df.drop(["Legendary"], axis = 1, inplace = True)
df.drop(["HP"], axis = 1, inplace = True)
df.drop(["Attack"], axis = 1, inplace = True)
df.drop(["Defense"], axis = 1, inplace = True)
df.drop(["Sp. Atk"], axis = 1, inplace = True)
df.drop(["Sp. Def"], axis = 1, inplace = True)
df.drop(["Speed"], axis = 1, inplace = True)
df.drop(["Generation"], axis = 1, inplace = True)
df.head()
```

[7] ✓ 0.0s

...

	Winbattle	Resistance
0	35	450
1	50	730
2	75	940
3	86	1110
4	40	500

Verifico los datos para ver si hay valores null.

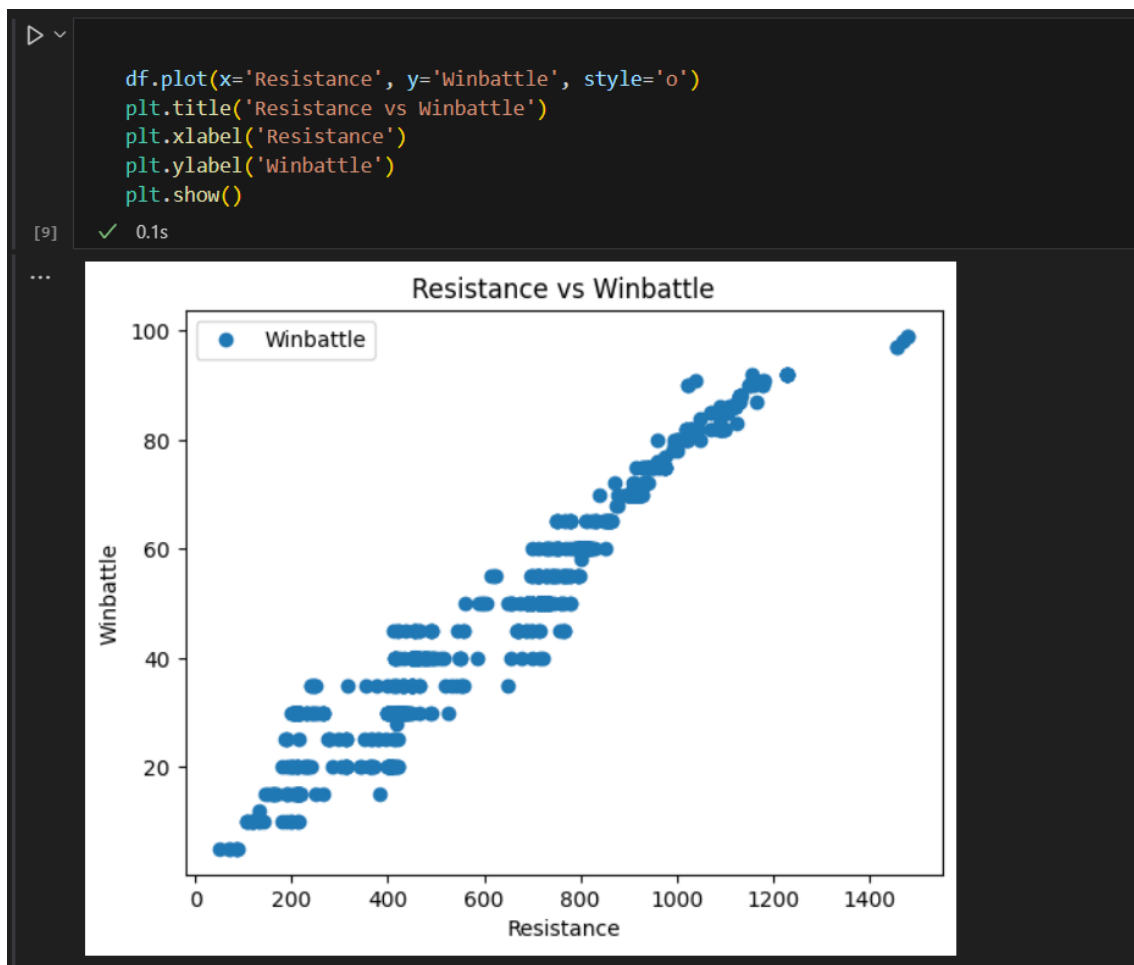
Como obtuve un false significa que no hay valores null y puedo continuar con el análisis.

```
# Busco si hay algún valor null.  
  
df.isnull().values.any()  
[8] ✓ 0.0s  
... False
```

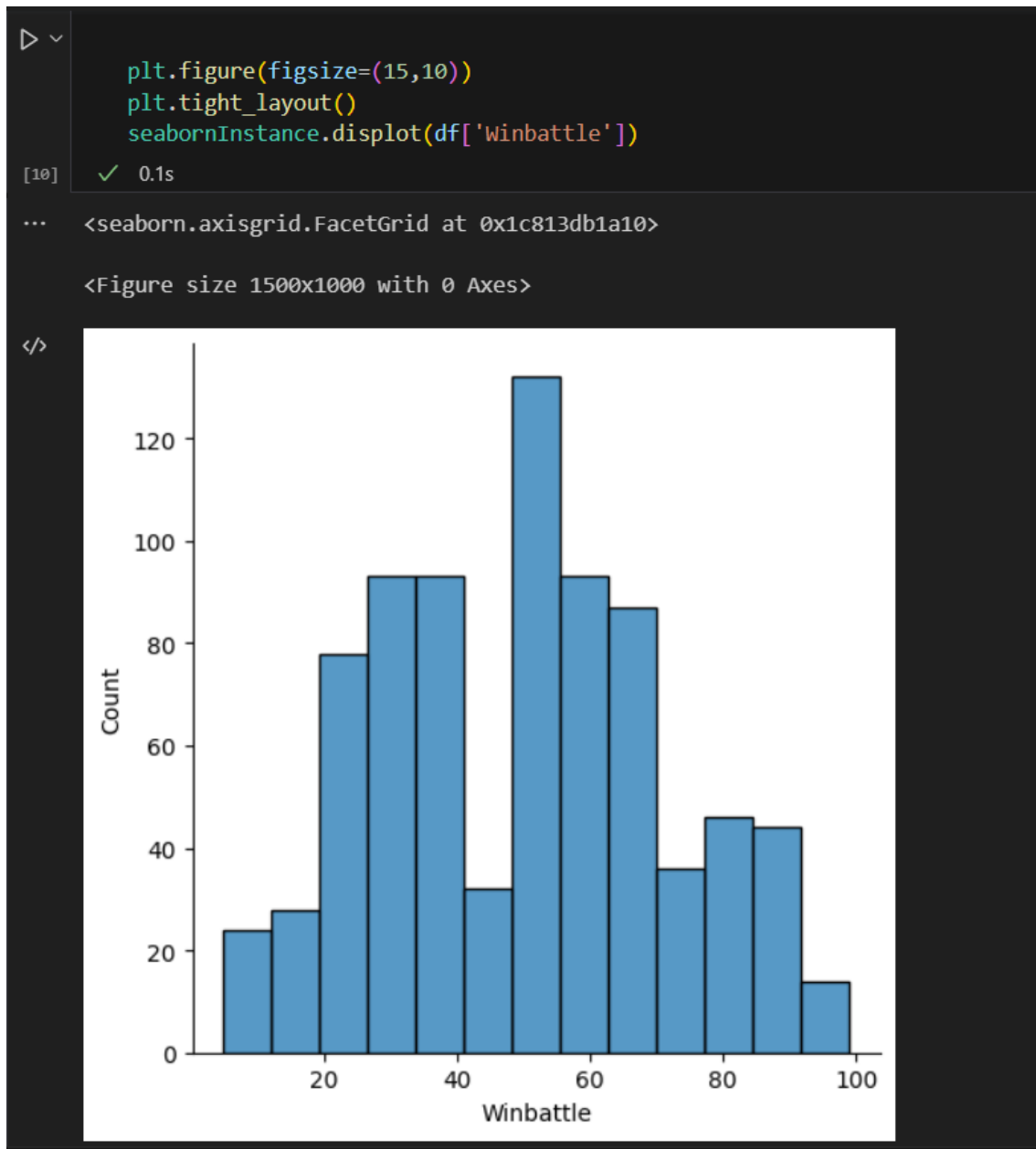
Gráfico los puntos de datos en un diagrama en dos dimensiones para ilustrar el dataset.

Verifico si manualmente puedo encontrar alguna relación entre los datos.

Vemos una proyección ascendente en este caso.



Busco el valor de Winbattle promedio y lo represento con un gráfico de barras.



La media de Winbattle se encuentra entre los valores de 45 a 55 aprox.

Luego dividido los datos en variable independiente y variable dependiente cuyos valores se deben predecir.

```
x = df['Resistance'].values.reshape(-1,1)
y = df['winbattle'].values.reshape(-1,1)
df_aux = pd.DataFrame({'x': x.flatten(), 'y': y.flatten()})
df_aux
```

[11] ✓ 0.0s

	x	y
0	450	35
1	730	50
2	940	75
3	1110	86
4	500	40
...
795	1030	82
796	1180	90
797	1095	85
798	1135	88
799	1030	82

800 rows × 2 columns

Utilizo `train_test_split` para dividir un dataset en bloques.

Conjunto de testing y conjunto de entrenamiento.

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
df_aux = pd.DataFrame({'X_train': X_train.flatten(), 'y_train': y_train.flatten()})
df_aux.head()

```

[12] ✓ 0.0s

	X_train	y_train
0	450	35
1	450	35
2	750	65
3	760	50
4	248	35

```

df_aux = pd.DataFrame({'X_test': X_test.flatten(), 'y_test': y_test.flatten()})
df_aux.head()

```

[13] ✓ 0.0s

	X_test	y_test
0	216	15
1	804	60
2	431	30
3	490	40
4	364	20

Realizo el entrenamiento del algoritmo.

```

regressor = LinearRegression()
regressor.fit(X_train, y_train)

```

[148]

... ▾ LinearRegression
LinearRegression()

```

# obtengo el interceptor.

print(regressor.intercept_)

[149]
... [2.63686166]

# obtengo la pendiente.

print(regressor.coef_)

[150]
... [[0.07241806]]

```

El resultado debe ser aproximadamente 2.63686166 y 0.07241806 respectivamente.

Esto significa que, por cada unidad de cambio en la Resistencia,

El cambio en el Winbattle es de alrededor de 0.072%.

Utilizo el método `predict()` para tomar este conjunto de características y devolver las predicciones correspondientes para la variable dependiente.

Percibo que el valor que obtengo en la columna Predicted se acerca bastante al Actual.

```

y_pred = regressor.predict(X_test)

[18] ✓ 0.0s

df_aux = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
df_aux

[19] ✓ 0.0s

```

	Actual	Predicted
0	15	18.279163
1	60	60.860985
2	30	33.849047
3	40	38.121713
4	20	28.997037
...
155	20	28.997037
156	30	19.293016
157	55	58.036680
158	60	64.264634
159	40	37.252696

160 rows × 2 columns

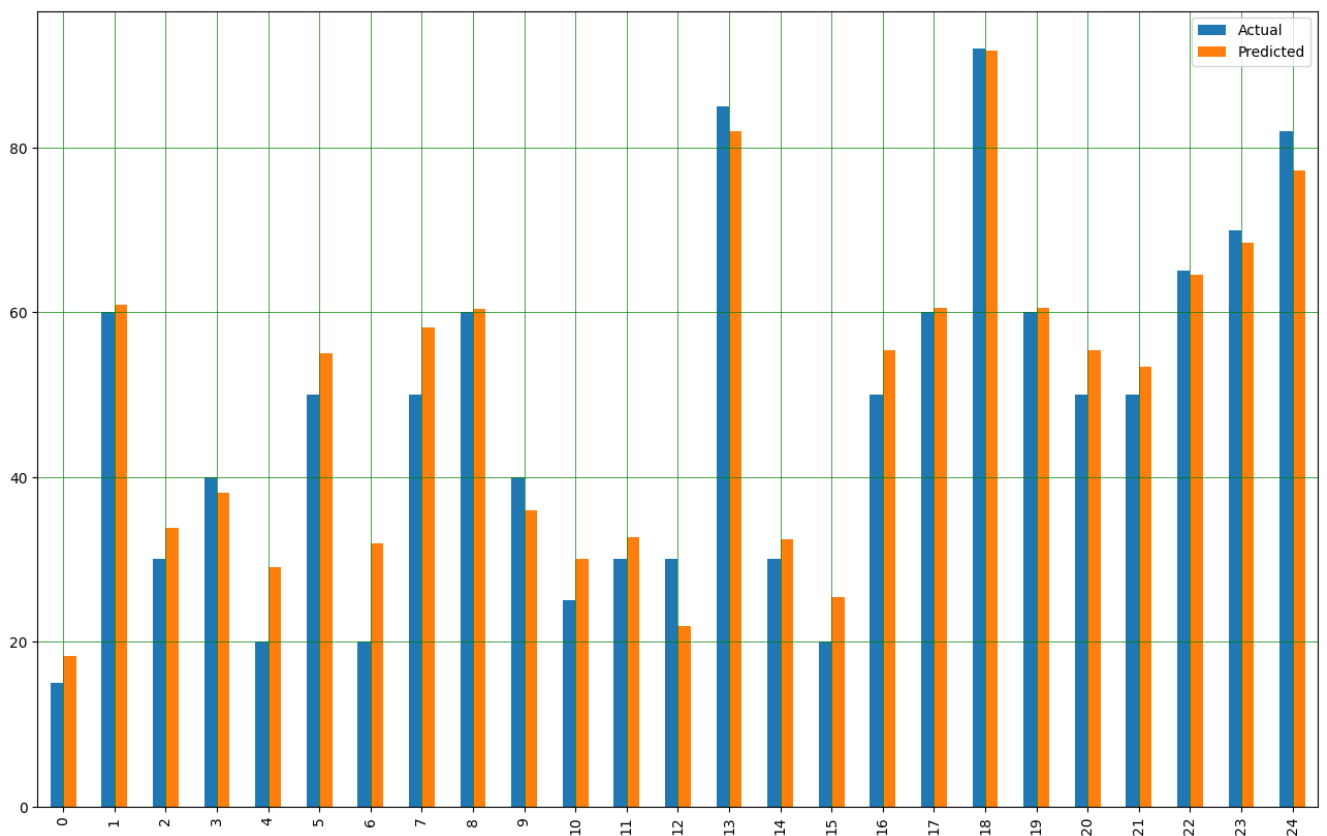
Realizo un gráfico de barras mostrando la comparación de valores reales y predichos.

Tomo unos 25 registros.

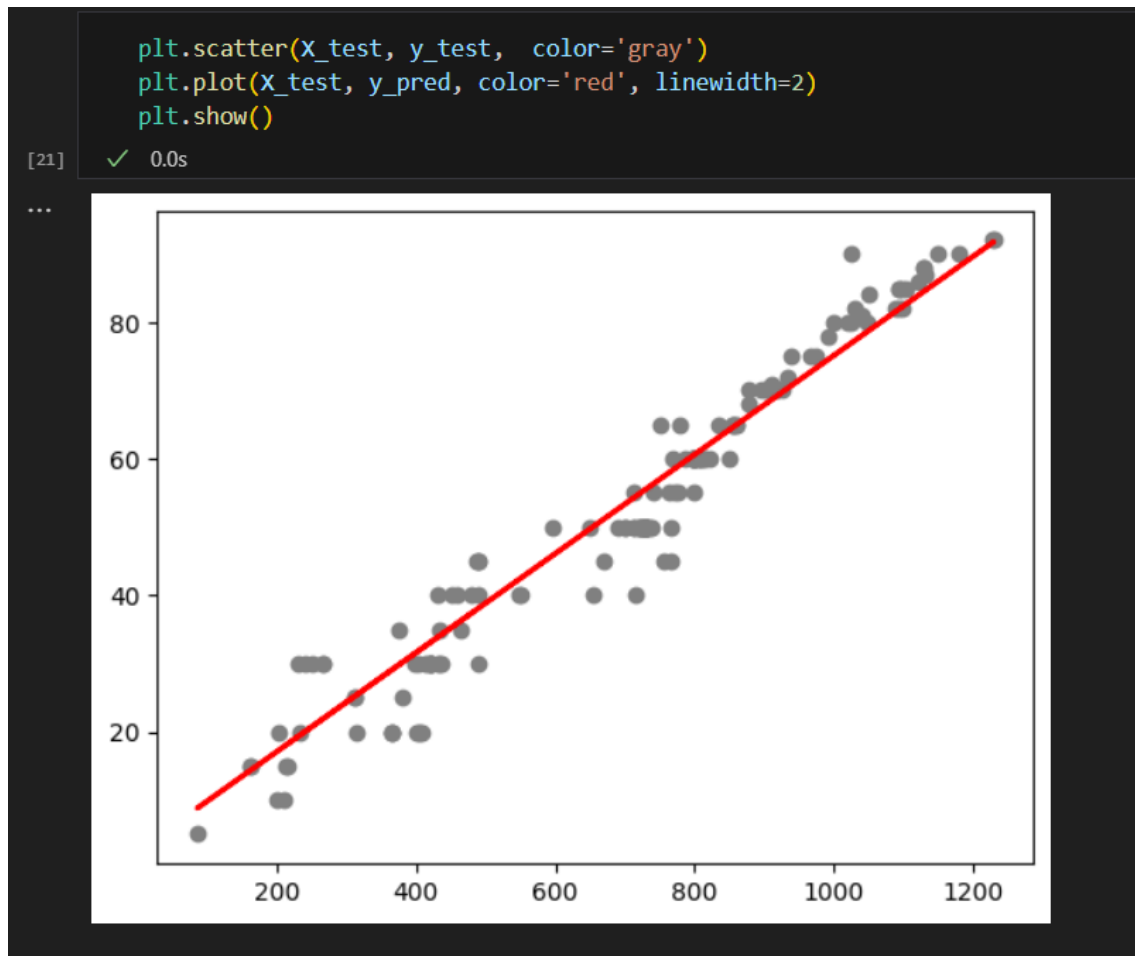
```
df1 = df_aux.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

[20] ✓ 0.1s

Gráfico Actual & Predicted.



Los porcentajes predichos se acercan a los reales, finalmente trazo una línea recta con los datos de la prueba.



Dan como resultado una línea ascendente.

Por último, analizaré el Error Absoluto Medio, el Error cuadrático Medio y la Raíz del error cuadrático medio.

El Error Absoluto Medio mide la magnitud promedio de los errores en las predicciones del modelo. Se calcula tomando la diferencia absoluta entre cada valor predicho y el valor real, y luego promediando esos errores.

El Error Cuadrático Medio calcula el promedio de los errores al cuadrado entre las predicciones y los valores reales. Al elevar los errores al cuadrado, esta métrica da más peso a los errores más grandes.

La Raíz del Error Cuadrático Medio es simplemente la raíz cuadrada del MSE. Proporciona una medida del error promedio, similar al MAE, pero está en la misma escala que los valores originales.


```
print('Error Absoluto Medio:', metrics.mean_absolute_error(y_test, y_pred))
print('Error Cuadratico Medio:', metrics.mean_squared_error(y_test, y_pred))
print('Raíz del error cuadrático medio:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

[22] ✓ 0.0s

```
... Error Absoluto Medio: 3.848300608049468
Error Cuadratico Medio: 24.84009195088455
Raíz del error cuadrático medio: 4.9839835423970404
```

Teniendo en cuenta estos valores, puedo sacar algunas conclusiones:

El Error Absoluto Medio de aproximadamente 3.85 indica que, en promedio, las predicciones difieren del valor real en aproximadamente 3.85 unidades en la escala de WinBattle.

El Error Cuadrático Medio de aproximadamente 24.84 muestra que los errores entre las predicciones y los valores reales pueden ser mayores en magnitud debido al efecto del cuadrado.

La Raíz del Error Cuadrático Medio de aproximadamente 4.98 indica que, en promedio, las predicciones difieren del valor real en aproximadamente 4.98 unidades en la escala de WinBattle.

Por lo tanto, los resultados indican que el modelo tiene un error promedio MODERADO en las predicciones de WinBattle con respecto a Resistance.

El error Cuadrático se encuentra además entre los valores acordes con relación al 10% del valor de la media de WinBattle que se encuentra entre los 45 y 55

Igualmente, en líneas generales y a la vista de los gráficos, la conclusión en general es que a mayor valor de Resistance del Pokémon, es altamente probable que haya un mayor número de WinBattle.

Ahora me gustaría realizar otro análisis, pero esta vez aplicando Regresión Lineal múltiple.

Realizo la importación de los datos.

```
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()
```

[23] ✓ 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

Realizo la función shape para comprobar los registros y columnas de mi dataset.

```
df.shape
```

[25] ✓ 0.0s

... (800, 15)

Para este análisis la columna de Name no será necesaria, por lo que decido eliminarla.

```
df.drop(["Name"], axis = 1, inplace = True)
```

[26] ✓ 0.0s

Compruebo que se eliminó la columna Name.

```
df.head()
```

[27] ✓ 0.0s

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

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

```
[28] df.isnull().values.any()
... True
```

En este caso el resultado devolvió true, por lo que hay valores null.

Consulto cuantos valores null hay.

```
[29] df.isnull().sum().sum()
... 386
```

Averiguo en que columnas hay valores null.

```
[30] df.isnull().any()
... # 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
```

Comprobé que únicamente tuve valores null en la columna Type 2, por lo que decido eliminarla para simplificar de manera resolutive.

```
df.drop(["Type 2"], axis = 1, inplace = True)
df.head()
```

[31] ✓ 0.0s

...

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

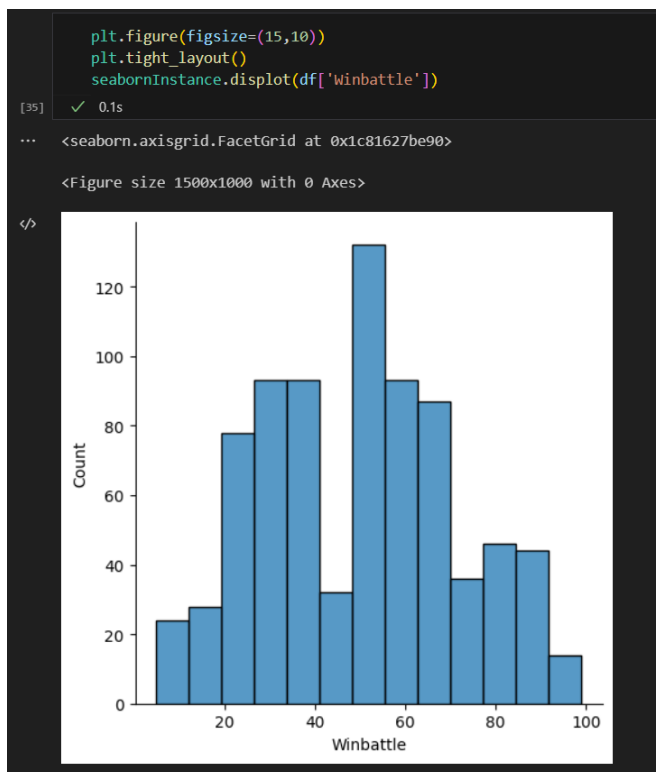
Constato que ahora no se encuentran valores null en el dataset.

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

[34] ✓ 0.0s

... False

Quiero encontrar el valor promedio de Winbattle.



La media de Winbattle se encuentra entre los valores de 45 a 55 aprox.

Ahora es cuando decido separar el dataset entre variables dependientes e independientes.

X tendrá las variables independientes.

```

x = df.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,12]].values
x

[36] ✓ 0.0s

... array([[1, 'Grass', 318, ..., 1, False, 450],
          [2, 'Grass', 405, ..., 1, False, 730],
          [3, 'Grass', 525, ..., 1, False, 940],
          ...,
          [720, 'Psychic', 600, ..., 6, True, 1095],
          [720, 'Psychic', 680, ..., 6, True, 1135],
          [721, 'Fire', 600, ..., 6, True, 1030]], dtype=object)

```

Mientras que Y almacenará la variable dependiente.

```

y = df.iloc[:, 11].values
y

[37] ✓ 0.0s

... array([35, 50, 75, 86, 40, 60, 80, 90, 90, 35, 50, 75, 86, 5, 25, 45, 5,
          25, 45, 65, 10, 30, 50, 65, 12, 30, 15, 30, 20, 45, 40, 65, 40, 60,
          20, 45, 65, 20, 45, 65, 40, 65, 25, 60, 20, 50, 10, 40, 10, 30, 60,
          15, 35, 15, 35, 20, 50, 10, 45, 15, 55, 15, 45, 20, 65, 20, 45, 60,
          30, 50, 70, 85, 20, 40, 65, 10, 35, 50, 20, 50, 30, 50, 70, 20, 50,
          20, 60, 80, 30, 60, 45, 15, 35, 30, 60, 15, 55, 30, 60, 30, 55, 75,
          85, 50, 35, 60, 15, 50, 30, 55, 20, 65, 30, 50, 50, 50, 55, 15, 50,
          40, 60, 55, 50, 55, 78, 25, 50, 20, 50, 20, 50, 60, 50, 60, 60, 60,
          45, 65, 55, 5, 70, 87, 70, 45, 30, 65, 65, 65, 40, 40, 70, 40, 70,
          75, 83, 72, 82, 82, 82, 40, 65, 88, 92, 99, 99, 91, 35, 55, 75, 40,
          60, 80, 35, 55, 75, 15, 55, 20, 55, 15, 35, 10, 30, 60, 30, 55, 25,
          30, 10, 25, 55, 30, 60, 20, 50, 70, 80, 55, 25, 55, 50, 60, 20, 30,
          50, 15, 10, 50, 40, 20, 50, 65, 65, 60, 55, 60, 40, 60, 40, 10, 30,
          35, 60, 72, 82, 30, 65, 60, 70, 82, 5, 40, 65, 60, 40, 68, 20, 50,
          30, 60, 30, 20, 50, 40, 50, 68, 40, 70, 82, 70, 30, 60, 71, 60, 15,
          10, 50, 20, 40, 30, 55, 65, 82, 82, 82, 40, 60, 80, 90, 88, 88, 85,
          35, 50, 75, 85, 35, 50, 75, 85, 35, 50, 75, 86, 30, 50, 15, 25, 5,
          15, 40, 15, 40, 30, 40, 60, 15, 40, 65, 15, 40, 30, 50, 30, 50, 70,
          85, 10, 40, 30, 55, 15, 35, 80, 30, 45, 25, 10, 30, 50, 10, 40, 10,
          40, 10, 40, 50, 70, 50, 60, 35, 55, 70, 85, 30, 60, 75, 30, 55, 75,
          20, 20, 35, 35, 50, 30, 45, 20, 50, 70, 30, 50, 30, 50, 70, 50, 30,
          60, 30, 30, 50, 70, 20, 50, 15, 65, 80, 40, 45, 60, 60, 20, 50, 20,
          50, 30, 60, 35, 60, 30, 60, 5, 70, 35, 40, 30, 50, 75, 30, 60, 40,
          30, 60, 75, 40, 30, 60, 80, 20, 45, 77, 30, 50, 50, 50, 30, 35, 60,
          80, 90, 30, 50, 75, 87, 81, 81, 83, 82, 92, 82, 92, 91, 97, 91, 97,
          ...
          50, 15, 20, 50, 20, 50, 30, 45, 60, 30, 60, 20, 60, 50, 72, 79, 79,
          35, 65, 72, 72, 30, 55, 30, 55, 20, 40, 30, 60, 25, 50, 20, 50, 20,

```

Ahora utilizo LabelEncoder para convertir a números los datos categóricos de la columna Type1.

```
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
X[:, 1] = labelencoder_X.fit_transform(X[:, 1])
X
```

[38] ✓ 0.0s

```
... array([[1, 9, 318, ..., 1, False, 450],
        [2, 9, 405, ..., 1, False, 730],
        [3, 9, 525, ..., 1, False, 940],
        ...,
        [720, 14, 600, ..., 6, True, 1095],
        [720, 14, 680, ..., 6, True, 1135],
        [721, 6, 600, ..., 6, True, 1030]], dtype=object)
```

Emito ambos grupos de datos para comparar y ver como quedaron convertidos los datos categóricos.

```
pd.concat([pd.DataFrame(X[:, 1]), df.iloc[:, 1]], axis=1)
```

[39] ✓ 0.0s

```
... 
```

	0	Type 1
0	9	Grass
1	9	Grass
2	9	Grass
3	9	Grass
4	6	Fire
...
795	15	Rock
796	15	Rock
797	14	Psychic
798	14	Psychic
799	6	Fire

800 rows × 2 columns

Mismo proceso con la columna Legendary, utilizo LabelEncoder para convertir a números los datos categóricos.

```

from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
X[:, 10] = labelencoder_X.fit_transform(X[:, 10])
X

```

[40] ✓ 0.0s

```

... array([[1, 9, 318, ..., 1, 0, 450],
          [2, 9, 405, ..., 1, 0, 730],
          [3, 9, 525, ..., 1, 0, 940],
          ...,
          [720, 14, 600, ..., 6, 1, 1095],
          [720, 14, 680, ..., 6, 1, 1135],
          [721, 6, 600, ..., 6, 1, 1030]], dtype=object)

```

Emito ambos grupos de datos para comparar y ver como quedaron convertidos los datos categóricos.

```

pd.concat([pd.DataFrame(X[:, 9]), df.iloc[:, 10]], axis=1)

```

[41] ✓ 0.0s

```

...

```

	0	Legendary
0	1	False
1	1	False
2	1	False
3	1	False
4	1	False
...
795	6	True
796	6	True
797	6	True
798	6	True
799	6	True

800 rows × 2 columns

Utilizo OneHotEncoder para codificar características categóricas como una matriz numérica.

make_column_transformer permite aplicar transformaciones de datos de forma selectiva a diferentes columnas del conjunto de datos.

Esto calcula los datos categóricos y sobrescribe.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
onehotencoder = make_column_transformer((OneHotEncoder(), [1]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)
X
```

[42] ✓ 0.0s

```
... array([[0.0, 0.0, 0.0, ..., 1, 0, 450],
          [0.0, 0.0, 0.0, ..., 1, 0, 730],
          [0.0, 0.0, 0.0, ..., 1, 0, 940],
          ...,
          [0.0, 0.0, 0.0, ..., 6, 1, 1095],
          [0.0, 0.0, 0.0, ..., 6, 1, 1135],
          [0.0, 0.0, 0.0, ..., 6, 1, 1030]], dtype=object)
```

Chequeo el resultado observando una porción.

```
pd.concat([pd.DataFrame(X),df.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,12]]], axis=1).head()
```

[43] ✓ 0.0s

	0	1	2	3	4	5	6	7	8	9	...	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Resistance
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	318	45	49	49	65	65	45	1	False	450
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	405	60	62	63	80	80	60	1	False	730
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	525	80	82	83	100	100	80	1	False	940
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	625	80	100	123	122	120	80	1	False	1110
4	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	309	39	52	43	60	50	65	1	False	500

5 rows × 41 columns

Para evitar las trampas de las variables ficticias elimino una de las columnas.

Elimino la columna 1 de las variables dummy.

Chequeo el resultado observando una porción.

```
X = X[:,1:]
pd.concat([pd.DataFrame(X),df.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,12]]], axis=1).head()
```

[44] ✓ 0.0s

	0	1	2	3	4	5	6	7	8	9	...	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Resistance
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	318	45	49	49	65	65	45	1	False	450
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	405	60	62	63	80	80	60	1	False	730
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	525	80	82	83	100	100	80	1	False	940
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	625	80	100	123	122	120	80	1	False	1110
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	309	39	52	43	60	50	65	1	False	500

5 rows × 40 columns

Utilizo fit transform y muestro la matriz Y.

```

labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
y

```

[45] ✓ 0.0s

```

... array([ 8, 11, 20, 31,  9, 14, 25, 34, 34,  8, 11, 20, 31,  0,  5, 10,  0,
           5, 10, 15,  1,  7, 11, 15,  2,  7,  3,  7,  4, 10,  9, 15,  9, 14,
           4, 10, 15,  4, 10, 15,  9, 15,  5, 14,  4, 11,  1,  9,  1,  7, 14,
           3,  8,  3,  8,  4, 11,  1, 10,  3, 12,  3, 10,  4, 15,  4, 10, 14,
           7, 11, 17, 30,  4,  9, 15,  1,  8, 11,  4, 11,  7, 11, 17,  4, 11,
           4, 14, 25,  7, 14, 10,  3,  8,  7, 14,  3, 12,  7, 14,  7, 12, 20,
          30, 11,  8, 14,  3, 11,  7, 12,  4, 15,  7, 11, 11, 11, 12,  3, 11,
           9, 14, 12, 11, 12, 23,  5, 11,  4, 11,  4, 11, 14, 11, 14, 14, 14,
          10, 15, 12,  0, 17, 32, 17, 10,  7, 15, 15, 15,  9,  9, 17,  9, 17,
          20, 28, 19, 27, 27, 27,  9, 15, 33, 36, 39, 39, 35,  8, 12, 20,  9,
          14, 25,  8, 12, 20,  3, 12,  4, 12,  3,  8,  1,  7, 14,  7, 12,  5,
           7,  1,  5, 12,  7, 14,  4, 11, 17, 25, 12,  5, 12, 11, 14,  4,  7,
          11,  3,  1, 11,  9,  4, 11, 15, 15, 14, 12, 14,  9, 14,  9,  1,  7,
           8, 14, 19, 27,  7, 15, 14, 17, 27,  0,  9, 15, 14,  9, 16,  4, 11,
           7, 14,  7,  4, 11,  9, 11, 16,  9, 17, 27, 17,  7, 14, 18, 14,  3,
           1, 11,  4,  9,  7, 12, 15, 27, 27, 27,  9, 14, 25, 34, 33, 33, 30,
           8, 11, 20, 30,  8, 11, 20, 30,  8, 11, 20, 31,  7, 11,  3,  5,  0,
           3,  9,  3,  9,  7,  9, 14,  3,  9, 15,  3,  9,  7, 11,  7, 11, 17,
          30,  1,  9,  7, 12,  3,  8, 25,  7, 10,  5,  1,  7, 11,  1,  9,  1,
           9,  1,  9, 11, 17, 11, 14,  8, 12, 17, 30,  7, 14, 20,  7, 12, 20,
           4,  4,  8,  8, 11,  7, 10,  4, 11, 17,  7, 11,  7, 11, 17, 11,  7,
          14,  7,  7, 11, 17,  4, 11,  3, 15, 25,  9, 10, 14, 14,  4, 11,  4,
          11,  7, 14,  8, 14,  7, 14,  0, 17,  8,  9,  7, 11, 20,  7, 14,  9,
           7, 14, 20,  9,  7, 14, 25,  4, 10, 22,  7, 11, 11, 11,  7,  8, 14,
          25, 34,  7, 11, 20, 32, 26, 26, 28, 27, 36, 27, 36, 35, 37, 35, 37,
          ...
          11,  3,  4, 11,  4, 11,  7, 10, 14,  7, 14,  4, 14, 11, 19, 24, 24,
           8, 15, 19, 19,  7, 12,  7, 12,  4,  9,  7, 14,  5, 11,  4, 11,  4,
          11,  9, 17,  9, 17, 20, 18, 11, 17,  9, 14, 27, 14,  4, 14,  7,  7,
           7,  7, 14, 14, 14, 14,  7, 14,  7, 14, 33, 33, 30, 27, 34, 30, 33,

```

Utilizo train_test_split para dividir un dataset en bloques.

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

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

```

[46] ✓ 0.0s

Chequeo las dimensiones.

```

len(X_train), len(X_test), len(y_train), len(y_test)
[47] ✓ 0.0s
... (640, 160, 640, 160)

```

Imprimo en pantalla X_train que me devolverá una matriz.

```

X_train
[48] ✓ 0.0s
... array([[0.0, 0.0, 0.0, ..., 2, 0, 450],
          [0.0, 0.0, 0.0, ..., 1, 0, 450],
          [0.0, 0.0, 0.0, ..., 1, 0, 750],
          ...,
          [0.0, 0.0, 0.0, ..., 5, 0, 366],
          [0.0, 0.0, 0.0, ..., 5, 0, 910],
          [0.0, 0.0, 0.0, ..., 5, 0, 824]], dtype=object)

```

Imprimo en pantalla X_test que me devolverá una matriz.

```

X_test
[49] ✓ 0.0s
... array([[0.0, 0.0, 0.0, ..., 3, 0, 216],
          [0.0, 0.0, 0.0, ..., 4, 0, 804],
          [0.0, 0.0, 0.0, ..., 3, 0, 431],
          ...,
          [0.0, 0.0, 0.0, ..., 3, 0, 765],
          [0.0, 0.0, 0.0, ..., 2, 0, 851],
          [0.0, 0.0, 0.0, ..., 2, 0, 478]], dtype=object)

```

Imprimo en pantalla `y_train` que me devolverá una matriz.

```

y_train
[50] ✓ 0.0s
... array([ 8,  8, 15, 11,  8, 12, 14, 10, 20, 14,  4, 10, 30, 10, 11, 31, 27,
          4,  1,  7,  7,  1,  7,  7,  4, 37,  8, 19, 17, 10,  8,  3, 27, 11,
        14, 11, 30,  7,  5,  5, 28,  7, 17, 30, 11, 11, 11, 33, 14, 14,  7,
          8, 36, 11,  7,  7, 10,  4,  4, 14, 11, 17,  5,  7, 11,  4, 17,  7,
        36,  7, 12, 11,  9,  7, 14, 27, 10, 11,  9,  5, 11, 34, 17, 10, 12,
        10, 31, 20,  7,  4,  8,  8,  1,  9, 15,  9, 11, 14, 15, 12, 35,  7,
          1, 25, 37,  4, 11, 19, 30,  7, 17,  7, 10,  1, 14, 34, 33, 11,  4,
        12, 11, 30, 11,  7,  6, 27, 12, 11,  7,  4,  7, 31, 10, 16,  7, 25,
        17, 11,  8,  4, 12, 14, 10,  9,  5,  9,  9,  9,  4,  3, 20, 12, 26,
        12, 11, 17, 11, 14, 12, 12, 25, 10, 17,  7,  1,  7, 12, 14, 15,  7,
        14,  5,  3,  8,  1, 25,  9, 20, 11, 14, 14, 19, 10, 17, 11, 30, 17,
          7, 14, 17,  7,  7,  9,  9,  7,  5,  4, 11, 20,  7, 27,  3, 14, 17,
          7, 14, 31, 14,  9, 11, 25,  5,  3, 14, 17, 17,  8,  9, 12,  7,  7,
          8, 14, 14, 17, 36, 20, 27, 11,  4, 17, 11, 11, 17,  1,  7, 33, 17,
          1,  4, 15, 11,  7,  7, 12, 11, 17, 20,  9,  7, 15, 14, 34,  8,  3,
          7, 17, 27,  8, 19, 14, 12, 27, 11,  7, 20,  8,  9, 10, 34, 12, 14,
        30, 14, 11,  9, 11, 22, 12,  9, 27,  4, 27,  4, 10,  9,  7, 20, 20,
        12, 10,  4, 14, 11,  9, 14, 17, 13, 14, 39, 23, 28, 20,  3, 10, 11,
        24, 14,  9,  8, 12, 14,  3, 30, 20, 34, 27,  3,  7,  4, 27, 17,  4,
          7,  1,  7,  9, 15, 20, 11, 11,  7,  9,  9, 17, 11,  4, 19, 17, 14,
        32, 11, 14, 14, 36, 14, 30, 11, 12, 14,  8,  7, 11, 38, 11, 14,  7,
          0, 17,  7,  4, 15, 14,  7, 11,  4, 17,  7, 39,  3,  0,  4, 14, 10,
        27,  5,  7, 17,  7, 11,  4, 15,  4, 26, 17,  7,  4,  1, 14, 11, 14,
          7,  4,  8, 14, 15,  3, 10, 11,  4,  5, 14, 12, 20,  4, 14,  8, 24,
          7,  7,  5, 10,  1, 20,  3, 33,  9, 19, 10, 15, 14,  7,  4,  4,  3,
        ...
          9,  1,  7, 11,  3,  7,  7,  4,  3, 17, 31,  9,  3,  1,  9,  4,  9,
          7,  5, 14,  7, 14, 11,  3, 27,  3,  8, 11,  4, 14, 10, 14, 15,  9,
        11, 17, 32, 36, 14,  9,  9,  0, 35, 15, 11, 11,  4,  7,  4, 31, 27,
        12, 11,  4,  7,  5, 14, 20, 25, 27,  4, 17,  7, 14,  5,  7, 17,  9,
          7, 11, 17,  8,  7, 33, 11, 14,  5, 17, 14,  dtype=int64)

```

Imprimo en pantalla `y_test` que me devolverá una matriz.

```

y_test
[51] ✓ 0.0s
... array([ 3, 14,  7,  9,  4, 11,  4, 11, 14,  9,  5,  7,  7, 30,  7,  4, 11,
        14, 36, 14, 11, 11, 15, 17, 27,  7, 27, 17, 17, 17, 15, 10,  4, 14,
        25, 15, 20, 10,  9,  7,  4, 34, 14, 15,  7,  7, 14,  7, 11,  9, 23,
          5,  9,  1,  7, 11, 31, 14,  7, 14, 29, 34, 11,  4, 15,  0,  7, 17,
        15, 16,  7, 17, 11, 12,  7, 11, 14, 11,  8, 30, 14,  7,  9, 17, 11,
        14, 11, 11,  4, 11, 19, 11, 30, 20, 14, 32, 11, 17, 14, 18, 11, 25,
        11, 11, 10, 14, 12, 14, 25,  4, 34, 14, 15,  9, 14, 17, 11, 15, 11,
        11, 12,  3, 11, 27,  7, 17, 11, 10, 11, 26,  5,  3, 20, 33, 33, 17,
        25, 12, 14, 10,  3, 10,  8, 11, 27, 36, 17, 30,  7,  9, 11, 12,  4,
          8,  1,  4,  7, 12, 14,  9], dtype=int64)

```

Escalo 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 $[-1, 1]$.

es un proceso comúnmente utilizado en el preprocesamiento de datos para asegurarse de que todas las variables o características tengan un rango de valores similar. Esto es especialmente importante cuando se trabaja con algoritmos que son sensibles a la escala de las variables, como muchas técnicas de aprendizaje automático.

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

[52] ✓ 0.0s

Ajusto el modelo de RLM con el conjunto de entrenamiento.

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)
```

[53] ✓ 0.0s

...
▼ LinearRegression
LinearRegression()

Realizo la predicción de los resultados en el conjunto de testing y muestro los datos que quedaron en `y_pred`

```
y_pred = regression.predict(X_test)
y_pred
```

[55] ✓ 0.0s

... array([2.57669037, 16.44455308, 6.2165486 , 8.78210228, 6.59930921,
12.83454755, 5.26409866, 15.04953637, 15.45062507, 10.08141936,
 4.88237945, 6.67972161, 4.41413013, 29.97567563, 9.11327772,
 3.90528759, 12.67821078, 14.49110541, 33.92869301, 16.39860874,
13.21703582, 13.10598674, 18.01854743, 17.23399692, 22.76620145,
 6.0441095 , 27.82028193, 17.42821948, 17.92906216, 17.42315666,
17.57753532, 14.46964248, 5.52974211, 15.16689211, 21.78118177,
13.92421187, 19.37629474, 15.20550187, 6.98557668, -6.08961174,
 5.04116875, 31.9306887 , 8.17054649, 17.9765059 , 5.06541122,
 6.44438195, 16.42446788, 7.70489124, 11.79017827, 6.89518994,
20.38713978, 4.0361082 , 8.00381385, 2.81357234, 2.86184579,
16.12538741, 22.74109809, 15.04876 , 4.42657498, 16.05603782,
21.98424031, 24.63671595, 12.2940396 , 3.80761208, 16.53221771,
 3.68167203, 5.19149225, 17.35450942, 16.80637174, 17.59336194,
 7.98971644, 23.60002516, 13.07757777, 15.44731159, 6.74080941,
13.78338151, 15.21634457, 14.02456612, 6.37568467, 29.76054732,
16.11368246, 6.47868598, 10.26272571, 16.71075246, 14.19290723,
15.17954 , 13.33510826, 13.85027695, 6.51944988, 13.9929802 ,
16.59084895, 13.841981 , 23.9130477 , 18.79715487, 16.45811517,
25.07402614, 13.28339037, 24.08759922, 15.93492548, 18.64124142,
13.86184207, 23.41370433, 13.32415112, 13.74202098, 12.32144318,
15.28974967, 15.45347188, 16.06496443, 22.97489781, 4.32699049,
30.85697762, 15.42845897, 16.09492097, 11.85238856, 14.07399157,
17.72116369, 13.7146043 , 15.81525918, 13.25857957, 12.83691067,
14.3339481 , 1.95875022, 12.62004316, 30.5787119 , 8.44316437,

Comparo los valores de las columnas Real del `y_test` con la predicción de `y_pred`

Observo que en general son valores relativamente cercanos.

```
df = pd.DataFrame({'Real': y_test.flatten(), 'Predicción': y_pred.flatten()})
df1.head()
```

[56] ✓ 0.0s

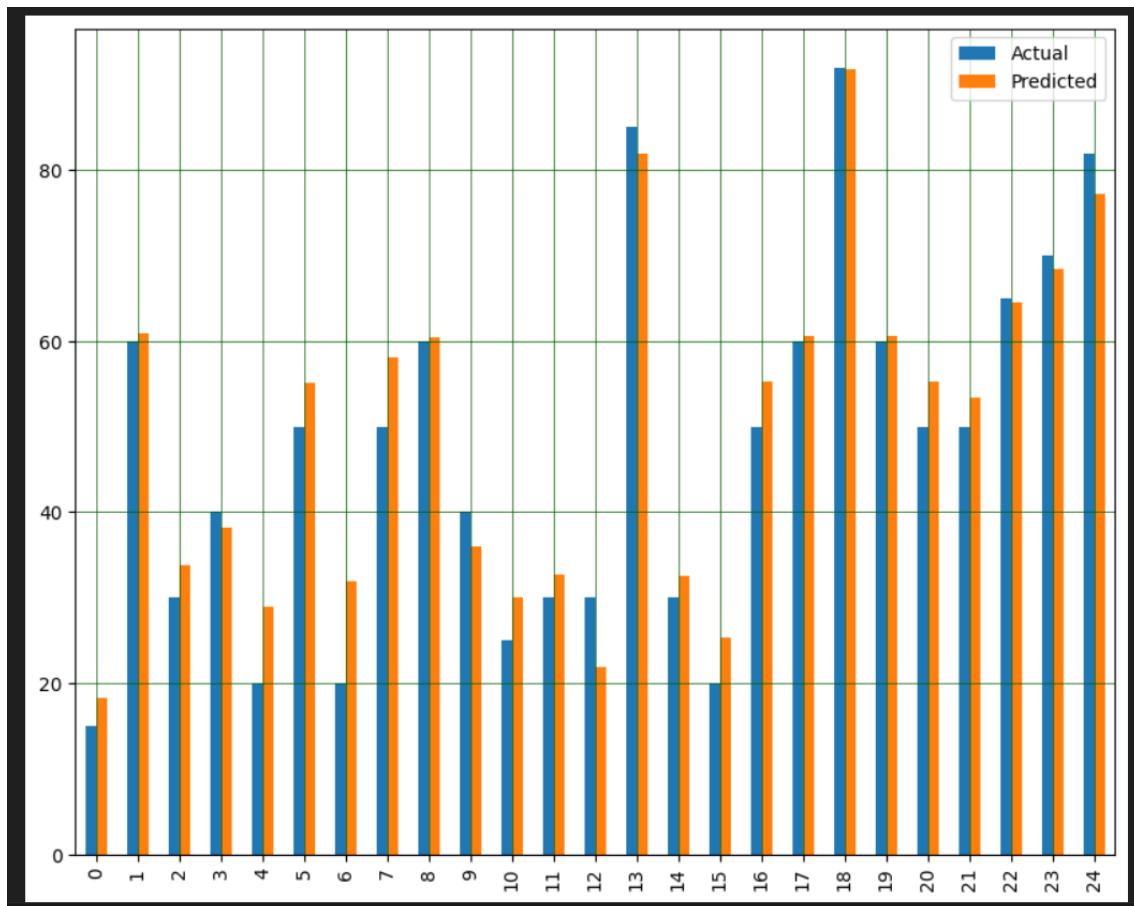
...

	Actual	Predicted
0	15	18.279163
1	60	60.860985
2	30	33.849047
3	40	38.121713
4	20	28.997037

Realizo un gráfico de barras mostrando las diferencias entre los valores reales y los de la predicción.

```
df1.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='darkgreen')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

[57] ✓ 0.2s



Finalmente analizaré el Error Absoluto Medio, el Error Cuadrático Medio y la Raíz del error cuadrático medio.

El Error Absoluto Medio mide la magnitud promedio de los errores en las predicciones del modelo. Se calcula tomando la diferencia absoluta entre cada valor predicho y el valor real, y luego promediando esos errores.

El Error Cuadrático Medio calcula el promedio de los errores al cuadrado entre las predicciones y los valores reales. Al elevar los errores al cuadrado, esta métrica da más peso a los errores más grandes.

La Raíz del Error Cuadrático Medio es simplemente la raíz cuadrada del MSE. Proporciona una medida del error promedio, similar al MAE, pero está en la misma escala que los valores originales.

```
print('Error Absoluto Medio:', metrics.mean_absolute_error(y_test, y_pred))
print('Error Cuadrático Medio:', metrics.mean_squared_error(y_test, y_pred))
print('Raíz del error cuadrático medio:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

[58] ✓ 0.0s

```
... Error Absoluto Medio: 2.158770970318563
Error Cuadrático Medio: 8.06719521699956
Raíz del error cuadrático medio: 2.8402808341781203
```

Teniendo en cuenta estos valores, puedo sacar algunas conclusiones:

El Error Absoluto Medio de aproximadamente 2.15 indica que, en promedio, las predicciones difieren del valor real en aproximadamente 3.85 unidades en la escala de WinBattle.

El Error Cuadrático Medio de aproximadamente 8.06 muestra que los errores entre las predicciones y los valores reales pueden ser mayores en magnitud debido al efecto del cuadrado.

La Raíz del Error Cuadrático Medio de aproximadamente 2.84 indica que, en promedio, las predicciones difieren del valor real en aproximadamente 2.84 unidades en la escala de WinBattle.

Por lo tanto, los resultados indican que el modelo tiene un error promedio MODERADAMENTE BAJO en las predicciones de WinBattle con respecto a Resistance.

Bajaron considerablemente los errores absoluto medio, cuadrático medio y la raíz del error cuadrático medio en la regresión múltiple comparado a la regresión lineal simple.

El error cuadrático se encuentra además debajo de los valores acordes con relación al 10% del valor de la media de WinBattle que se encuentra entre los 45 y 55.

Puedo intentar hacer una variante a esto con la eliminación hacia atrás para eliminar algunas variables que no influyen tanto con la variable dependiente.

Chequeo del dataset la porción X

```
pd.DataFrame(X).head()
```

[59] ✓ 0.0s

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25	26	27
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	318	45	49	49	65	65	45	1	0	450
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	405	60	62	63	80	80	60	1	0	730
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	525	80	82	83	100	100	80	1	0	940
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	625	80	100	123	122	120	80	1	0	1110
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	309	39	52	43	60	50	65	1	0	500

5 rows × 28 columns

Importo statsmodels, que es un módulo que proporciona clases y funciones para la estimación de muchos modelos estadísticos diferentes, para realizar pruebas estadísticas y exploración de datos estadísticos.

Construyo el modelo óptimo de RLM utilizando la Eliminación hacia atrás.

Agrego la columna de 1 al conjunto X original.

Tupla de 800 filas por 1 columna de tipo entero.

```
import statsmodels.api as sm

X = np.append(arr = np.ones((800,1)).astype(int), values = X, axis = 1)
pd.DataFrame(X).head()
```

[60] ✓ 0.7s

	0	1	2	3	4	5	6	7	8	9	...	19	20	21	22	23	24	25	26	27	28
0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	318	45	49	49	65	65	45	1	0	450
1	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	405	60	62	63	80	80	60	1	0	730
2	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	525	80	82	83	100	100	80	1	0	940
3	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	625	80	100	123	122	120	80	1	0	1110
4	1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	309	39	52	43	60	50	65	1	0	500

5 rows × 29 columns

X_opt, al inicio, tomará todas las filas y cada una de las columnas del conjunto de datos.

```
X_opt = X[:, [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28]]
pd.DataFrame(X_opt).head()
```

[253]

	0	1	2	3	4	5	6	7	8	9	...	19	20	21	22	23	24	25	26	27	28
0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	318	45	49	49	65	65	45	1	0	450
1	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	405	60	62	63	80	80	60	1	0	730
2	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	525	80	82	83	100	100	80	1	0	940
3	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	625	80	100	123	122	120	80	1	0	1110
4	1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	309	39	52	43	60	50	65	1	0	500

5 rows × 29 columns

Aplico la técnica OLS.

Es una técnica de regresión lineal utilizada para estimar los parámetros de un modelo de regresión.

Consiste en encontrar la línea o superficie que mejor se ajusta a los datos observados minimizando la suma de los residuos al cuadrado.

Observar el p valor en el sumario.

```
SL = 0.05
regression_OLS = sm.OLS(endog = y, exog = X_opt.tolist()).fit()
regression_OLS.summary()
```

[63] ✓ 0.0s

OLS Regression Results						
Dep. Variable:	y	R-squared:	0.909			
Model:	OLS	Adj. R-squared:	0.906			
Method:	Least Squares	F-statistic:	285.2			
Date:	Thu, 25 May 2023	Prob (F-statistic):	0.00			
Time:	23:27:02	Log-Likelihood:	-1853.2			
No. Observations:	800	AIC:	3762.			
Df Residuals:	772	BIC:	3894.			
Df Model:	27					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-5.4978	0.598	-9.194	0.000	-6.672	-4.324
x1	-0.1307	0.562	-0.233	0.816	-1.233	0.972
x2	2.2249	0.584	3.807	0.000	1.078	3.372
x3	-1.3286	0.505	-2.633	0.009	-2.319	-0.338
x4	0.4112	0.698	0.589	0.556	-0.959	1.782
x5	0.1809	0.579	0.312	0.755	-0.955	1.317
x6	-0.6444	0.482	-1.337	0.182	-1.591	0.302
x7	-3.4074	1.316	-2.588	0.010	-5.992	-0.823
x8	0.1192	0.558	0.214	0.831	-0.977	1.215

El funcionamiento es el siguiente:

Si este p valor supera el nivel de significación (SL), entonces se sigue con el siguiente paso y se procede a eliminar esa variable.

Si todas las variables tienen un p valor inferior 0.05 (SL) entonces se da por finalizado el modelo.

En este caso hay varias variables que superan dicho valor, elimino la más grande que es la 12.

Saco columna 12 y vuelvo a aplicar la técnica OLS.

```
X_opt = X[:, [0,1,2,3,4,5,6,7,8,9,10,11,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28]]
regression_OLS = sm.OLS(endog = y, exog = X_opt.tolist()).fit()
regression_OLS.summary()
```

[64] ✓ 0.0s

...

OLS Regression Results						
Dep. Variable:	y	R-squared:	0.909			
Model:	OLS	Adj. R-squared:	0.906			
Method:	Least Squares	F-statistic:	296.5			
Date:	Thu, 25 May 2023	Prob (F-statistic):	0.00			
Time:	23:32:02	Log-Likelihood:	-1853.2			
No. Observations:	800	AIC:	3760.			
Df Residuals:	773	BIC:	3887.			
Df Model:	26					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-5.4813	0.552	-9.930	0.000	-6.565	-4.398
x1	-0.1477	0.510	-0.290	0.772	-1.148	0.852
x2	2.2075	0.532	4.146	0.000	1.162	3.253
x3	-1.3452	0.449	-2.996	0.003	-2.227	-0.464
x4	0.3944	0.658	0.600	0.549	-0.897	1.686
x5	0.1637	0.528	0.310	0.756	-0.872	1.200
x6	-0.6614	0.421	-1.569	0.117	-1.489	0.166
x7	-3.4250	1.293	-2.649	0.008	-5.963	-0.887
x8	0.1029	0.510	0.202	0.840	-0.899	1.105

Saco columna 8 y vuelvo a aplicar la técnica OLS.

```

X_opt = X[:, [0,1,2,3,4,5,6,7,9,10,11,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28]]
regression_OLS = sm.OLS(endog = y, exog = X_opt.tolist()).fit()
regression_OLS.summary()

```

[65] ✓ 0.0s

OLS Regression Results

Dep. Variable:	y	R-squared:	0.909
Model:	OLS	Adj. R-squared:	0.906
Method:	Least Squares	F-statistic:	308.8
Date:	Thu, 25 May 2023	Prob (F-statistic):	0.00
Time:	23:33:01	Log-Likelihood:	-1853.2
No. Observations:	800	AIC:	3758.
Df Residuals:	774	BIC:	3880.
Df Model:	25		

Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
const	-5.4569	0.538	-10.139	0.000	-6.513	-4.400
x1	-0.1708	0.496	-0.344	0.731	-1.145	0.803
x2	2.1815	0.516	4.225	0.000	1.168	3.195
x3	-1.3671	0.435	-3.140	0.002	-2.222	-0.512
x4	0.3720	0.648	0.574	0.566	-0.900	1.644
x5	0.1482	0.522	0.284	0.777	-0.876	1.173
x6	-0.6814	0.409	-1.665	0.096	-1.485	0.122
x7	-3.4509	1.286	-2.684	0.007	-5.975	-0.927
x8	-1.4191	0.363	-3.913	0.000	-2.131	-0.707
x9	-0.2827	0.497	-0.569	0.570	-1.258	0.693
x10	-0.1888	0.558	-0.338	0.735	-1.285	0.907

Saco columna 5 y vuelvo a aplicar la Técnica OLS

```

X_opt = X[:, [0,1,2,3,4,6,7,9,10,11,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28]]
regression_OLS = sm.OLS(endog = y, exog = X_opt.tolist()).fit()
regression_OLS.summary()

```

[66] ✓ 0.0s

OLS Regression Results

Dep. Variable:	y	R-squared:	0.909
Model:	OLS	Adj. R-squared:	0.906
Method:	Least Squares	F-statistic:	322.0
Date:	Thu, 25 May 2023	Prob (F-statistic):	0.00
Time:	23:33:42	Log-Likelihood:	-1853.3
No. Observations:	800	AIC:	3757.
Df Residuals:	775	BIC:	3874.
Df Model:	24		

Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
const	-5.4390	0.534	-10.181	0.000	-6.488	-4.390
x1	-0.1901	0.491	-0.387	0.699	-1.154	0.774
x2	2.1609	0.511	4.230	0.000	1.158	3.164
x3	-1.3807	0.432	-3.192	0.001	-2.230	-0.532
x4	0.3568	0.645	0.553	0.581	-0.910	1.624
x5	-0.6982	0.405	-1.725	0.085	-1.493	0.096
x6	-3.4644	1.284	-2.698	0.007	-5.985	-0.944
x7	-1.4351	0.358	-4.008	0.000	-2.138	-0.732

Así continuamente hasta cumplir con el criterio.

Otra posibilidad que se puede hacer es aplicar una fórmula para no hacerlo de manera manual.

Se lo conoce como Eliminación automática hacia atrás utilizando solamente p-valores.

```
def EliminacionBackward(x, sl):
    numVars = len(x[0])
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x.tolist()).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    x = np.delete(x, j, 1)
            regressor_OLS.summary()
    return x

SL = 0.05
X_opt = X[:, [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28]]
X_Modelado = EliminacionBackward(X_opt, SL)

pd.DataFrame(X_Modelado).head()
```

[67] ✓ 0.0s

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0.0	0.0	0.0	1.0	0.0	0.0	1	318	65	0	450
1	1	0.0	0.0	0.0	1.0	0.0	0.0	2	405	80	0	730
2	1	0.0	0.0	0.0	1.0	0.0	0.0	3	525	100	0	940
3	1	0.0	0.0	0.0	1.0	0.0	0.0	3	625	122	0	1110
4	1	0.0	0.0	0.0	0.0	0.0	0.0	4	309	60	0	500

Entreno nuevamente los datos, en este caso X.

```
X = onehotencoder.fit_transform(X_Modelado)
X
```

[68] ✓ 0.0s

```
... array([[1.0, 0.0, 1, ..., 65, 0, 450],
        [1.0, 0.0, 1, ..., 80, 0, 730],
        [1.0, 0.0, 1, ..., 100, 0, 940],
        ...,
        [1.0, 0.0, 1, ..., 150, 1, 1095],
        [1.0, 0.0, 1, ..., 170, 1, 1135],
        [1.0, 0.0, 1, ..., 130, 1, 1030]], dtype=object)
```

Entreno nuevamente los datos, en este caso Y.

```

y = labelencoder_y.fit_transform(y)
y
[69] ✓ 0.0s
... array([ 8, 11, 20, 31,  9, 14, 25, 34, 34,  8, 11, 20, 31,  0,  5, 10,  0,
          5, 10, 15,  1,  7, 11, 15,  2,  7,  3,  7,  4, 10,  9, 15,  9, 14,
          4, 10, 15,  4, 10, 15,  9, 15,  5, 14,  4, 11,  1,  9,  1,  7, 14,
          3,  8,  3,  8,  4, 11,  1, 10,  3, 12,  3, 10,  4, 15,  4, 10, 14,
          7, 11, 17, 30,  4,  9, 15,  1,  8, 11,  4, 11,  7, 11, 17,  4, 11,
          4, 14, 25,  7, 14, 10,  3,  8,  7, 14,  3, 12,  7, 14,  7, 12, 20,
        30, 11,  8, 14,  3, 11,  7, 12,  4, 15,  7, 11, 11, 11, 12,  3, 11,
          9, 14, 12, 11, 12, 23,  5, 11,  4, 11,  4, 11, 14, 11, 14, 14, 14,
        10, 15, 12,  0, 17, 32, 17, 10,  7, 15, 15, 15,  9,  9, 17,  9, 17,
        20, 28, 19, 27, 27, 27,  9, 15, 33, 36, 39, 39, 35,  8, 12, 20,  9,
        14, 25,  8, 12, 20,  3, 12,  4, 12,  3,  8,  1,  7, 14,  7, 12,  5,
          7,  1,  5, 12,  7, 14,  4, 11, 17, 25, 12,  5, 12, 11, 14,  4,  7,
        11,  3,  1, 11,  9,  4, 11, 15, 15, 14, 12, 14,  9, 14,  9,  1,  7,
          8, 14, 19, 27,  7, 15, 14, 17, 27,  0,  9, 15, 14,  9, 16,  4, 11,
          7, 14,  7,  4, 11,  9, 11, 16,  9, 17, 27, 17,  7, 14, 18, 14,  3,
          1, 11,  4,  9,  7, 12, 15, 27, 27, 27,  9, 14, 25, 34, 33, 33, 30,
          8, 11, 20, 30,  8, 11, 20, 30,  8, 11, 20, 31,  7, 11,  3,  5,  0,
          3,  9,  3,  9,  7,  9, 14,  3,  9, 15,  3,  9,  7, 11,  7, 11, 17,
        30,  1,  9,  7, 12,  3,  8, 25,  7, 10,  5,  1,  7, 11,  1,  9,  1,
          9,  1,  9, 11, 17, 11, 14,  8, 12, 17, 30,  7, 14, 20,  7, 12, 20,
          4,  4,  8,  8, 11,  7, 10,  4, 11, 17,  7, 11,  7, 11, 17, 11,  7,
        14,  7,  7, 11, 17,  4, 11,  3, 15, 25,  9, 10, 14, 14,  4, 11,  4,
        11,  7, 14,  8, 14,  7, 14,  0, 17,  8,  9,  7, 11, 20,  7, 14,  9,
          7, 14, 20,  9,  7, 14, 25,  4, 10, 22,  7, 11, 11, 11,  7,  8, 14,
        25, 34,  7, 11, 20, 32, 26, 26, 28, 27, 36, 27, 36, 35, 37, 35, 37,
        ...
        11,  3,  4, 11,  4, 11,  7, 10, 14,  7, 14,  4, 14, 11, 19, 24, 24,
          8, 15, 19, 19,  7, 12,  7, 12,  4,  9,  7, 14,  5, 11,  4, 11,  4,
        11,  9, 17,  9, 17, 20, 18, 11, 17,  9, 14, 27, 14,  4, 14,  7,  7,
          7,  7, 14, 14, 14, 14,  7, 14,  7, 14, 33, 33, 30, 27, 34, 30, 33,
        27]. dtype=int64)

```

Utilizo train_test_split para dividir un dataset en bloques.

Conjunto de testing y conjunto de entrenamiento.

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
[70] ✓ 0.0s

```

Escala 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 $[-1, 1]$.

es un proceso comúnmente utilizado en el preprocesamiento de datos para asegurarse de que todas las variables o características tengan un rango de valores similar. Esto es especialmente importante cuando se trabaja con algoritmos que son sensibles a la escala de las variables, como muchas técnicas de aprendizaje automático.

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

[71] ✓ 0.0s

Ajusto el modelo de RLM con el conjunto de entrenamiento.

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)
```

[72] ✓ 0.0s

...

- ▼ LinearRegression

LinearRegression()

Realizo la predicción de los resultados en el conjunto de testing y muestro los datos que quedaron en `y_pred`.

Noto que obtengo datos muy similares a los anteriores, aunque con mayor exactitud dada la aplicación de la Eliminación hacia atrás automática.

```

y_pred = regression.predict(X_test)
y_pred
[73] ✓ 0.0s
... array([ 2.47901816, 16.25867019,  6.05655597,  8.30922026,  6.69938551,
          13.28482297,  5.76754959, 14.79555565, 15.07689141, 10.14495529,
           4.30812932,  6.70397775,  4.08205743, 30.23314392,  8.88213134,
           4.25015483, 12.85148408, 14.56310283, 33.8013795 , 16.3345872 ,
          12.8275583 , 13.11776646, 18.4463679 , 17.10172235, 23.57995458,
           6.02342565, 28.43505894, 17.74195909, 17.71590633, 17.52801807,
          17.13631406, 14.25307518,  5.32851567, 15.73481348, 21.86943153,
          13.86777839, 19.62861119, 14.58471342,  7.18411306, -5.98829795,
           4.67808692, 31.85782241,  8.26169039, 17.87117259,  5.3525534 ,
           6.26436028, 16.02415704,  7.38948175, 11.46687563,  7.00645251,
          20.24525801,  4.07204888,  8.22319942,  2.73864563,  3.05885811,
          15.78940329, 22.42986887, 15.31056615,  4.08205743, 15.49506739,
          21.8517367 , 25.0016629 , 12.76611103,  3.50015716, 16.79025605,
           3.7608326 ,  5.18230284, 17.53515175, 17.67599824, 17.34547471,
           8.04096807, 24.29438146, 12.99443378, 14.63762407,  6.68409637,
          14.204883 , 15.66027079, 13.45023361,  6.85741002, 29.82933533,
          15.90343486,  6.61599661, 10.49059176, 16.82807287, 13.74862584,
          15.17042943, 13.43809565, 14.40039079,  6.45316554, 13.72695092,
          16.52891605, 13.44393358, 22.71991269, 18.39444341, 15.86681376,
          24.7761508 , 13.18632831, 24.50257981, 16.04863542, 18.44779704,
          14.28210526, 23.77970925, 13.34259975, 13.76441043, 12.20417556,
          15.03108824, 15.13237444, 15.96316476, 23.27633891,  4.56300738,
          31.13908425, 15.25918942, 16.73884837, 12.10706716, 14.39562236,
          17.32523845, 13.49550561, 16.11309805, 13.07646503, 12.60997065,
          14.39930945,  1.72778674, 12.4026369 , 31.10814392,  8.48976044,
          ...
          18.25692671, 20.59537711, 14.71054708, 16.93029033,  9.17622197,
           1.11449356,  9.51899541,  4.6374643 , 14.27319474, 28.12851358,
          34.94914293, 17.4810478 , 24.73500178,  7.81086371, 10.84310705,
          13.17863252, 12.83111917,  3.1257312 ,  6.60926312,  2.96139715,
           4.01872144,  1.95329884, 13.9325769 , 15.40265601,  8.11753771])

```

Comparo los valores de las columnas Real del y_test con la predicción de y_pred.

Nuevamente veo que los valores son similares.

```
[74] ✓ 0.0s
pd.DataFrame({'Real': y_test.flatten(), 'Predicción': y_pred.flatten()})
```

	Real	Predicción
0	3	2.479018
1	14	16.258670
2	7	6.056560
3	9	8.309220
4	4	6.699386
...
155	4	4.018721
156	7	1.953299
157	12	13.932577
158	14	15.402656
159	9	8.117538

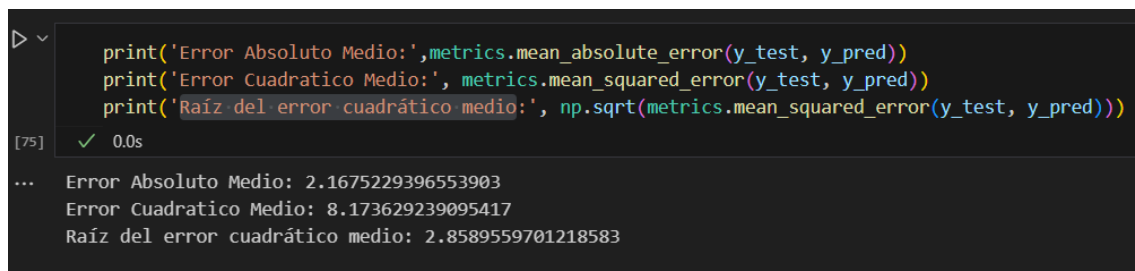
160 rows × 2 columns

Finalmente analizaré el Error Absoluto Medio, el Error Cuadrático Medio y la Raíz del error cuadrático medio.

El Error Absoluto Medio mide la magnitud promedio de los errores en las predicciones del modelo. Se calcula tomando la diferencia absoluta entre cada valor predicho y el valor real, y luego promediando esos errores.

El Error Cuadrático Medio calcula el promedio de los errores al cuadrado entre las predicciones y los valores reales. Al elevar los errores al cuadrado, esta métrica da más peso a los errores más grandes.

La Raíz del Error Cuadrático Medio es simplemente la raíz cuadrada del MSE. Proporciona una medida del error promedio, similar al MAE, pero está en la misma escala que los valores originales.



```

print('Error Absoluto Medio:', metrics.mean_absolute_error(y_test, y_pred))
print('Error Cuadrático Medio:', metrics.mean_squared_error(y_test, y_pred))
print('Raíz del error cuadrático medio:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

```

[75] ✓ 0.0s

```

... Error Absoluto Medio: 2.1675229396553903
Error Cuadrático Medio: 8.173629239095417
Raíz del error cuadrático medio: 2.8589559701218583

```

Los valores obtenidos terminan siendo muy similares a los anteriores de la regresión lineal múltiple sin hacer la eliminación hacia atrás.

La conclusión de estos datos es que los atributos que se eliminaron durante el proceso de regresión hacia atrás no aportaban una mejora significativa en la capacidad predictiva del modelo.

Ya que, cuando se realiza una regresión hacia atrás OLS, el objetivo es eliminar gradualmente los atributos menos relevantes del modelo hasta llegar a una configuración óptima.

Si los atributos que se eliminaron durante la regresión hacia atrás tenían un impacto insignificante en la predicción de la variable dependiente, es probable que su exclusión no haya afectado significativamente la capacidad del modelo para explicar la variabilidad de los datos. Como resultado, el error absoluto medio y la raíz del error cuadrático medio obtenidos después de la regresión hacia atrás OLS serán similares a los obtenidos mediante una regresión lineal múltiple.

Ahora me gustaría realizar algunos Análisis Exploratorio de Datos (EDA).

Realizo la importación de los datos.

```
df = pd.read_csv('pokemonResistance.csv', low_memory=False)
df.head()
```

[76] ✓ 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

Anteriormente, comprobé que únicamente tuve valores null en la columna Type 2, por lo que decido eliminarla para simplificar de manera resolutive.

```
df.drop(["Type 2"], axis = 1, inplace = True)
df.head()
```

[77] ✓ 0.0s

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

Analizaré los datos que tienen que ver con los tipos de Pokémon.

Calculo el promedio de cada Tipo de Pokémon según su valor Total.

Son 18 tipos.

```
print("Promedio por tipo:")
promedio_por_tipo = df.groupby("Type 1")["Total"].mean()
print(promedio_por_tipo)
```

[78] ✓ 0.0s

... Promedio por tipo:

Type 1	
Bug	378.927536
Dark	445.741935
Dragon	550.531250
Electric	443.409091
Fairy	413.176471
Fighting	416.444444
Fire	458.076923
Flying	485.000000
Ghost	439.562500
Grass	421.142857
Ground	437.500000
Ice	433.458333
Normal	401.683673
Poison	399.142857
Psychic	475.947368
Rock	453.750000
Steel	487.703704
Water	430.455357

Name: Total, dtype: float64

Calculo el porcentaje representativo de cada Tipo de Pokémon.

```
total = promedio_por_tipo.sum()
porcentaje_por_tipo = (promedio_por_tipo / total) * 100
print("\nPorcentaje por tipo:\n", porcentaje_por_tipo)
```

[79] ✓ 0.0s

...

Porcentaje por tipo:

Type 1	
Bug	4.753437
Dark	5.591586
Dragon	6.906110
Electric	5.562322
Fairy	5.183071
Fighting	5.224066
Fire	5.746322
Flying	6.084057
Ghost	5.514069
Grass	5.283005
Ground	5.488196
Ice	5.437495
Normal	5.038900
Poison	5.007027
Psychic	5.970497
Rock	5.692043
Steel	6.117974
Water	5.399825

Name: Total, dtype: float64

Determino cual es el Tipo con mayor porcentaje de todos.

```

tipo_mayor = porcentaje_por_tipo.idxmax()
print("\nTipo con mayor porcentaje:", tipo_mayor)

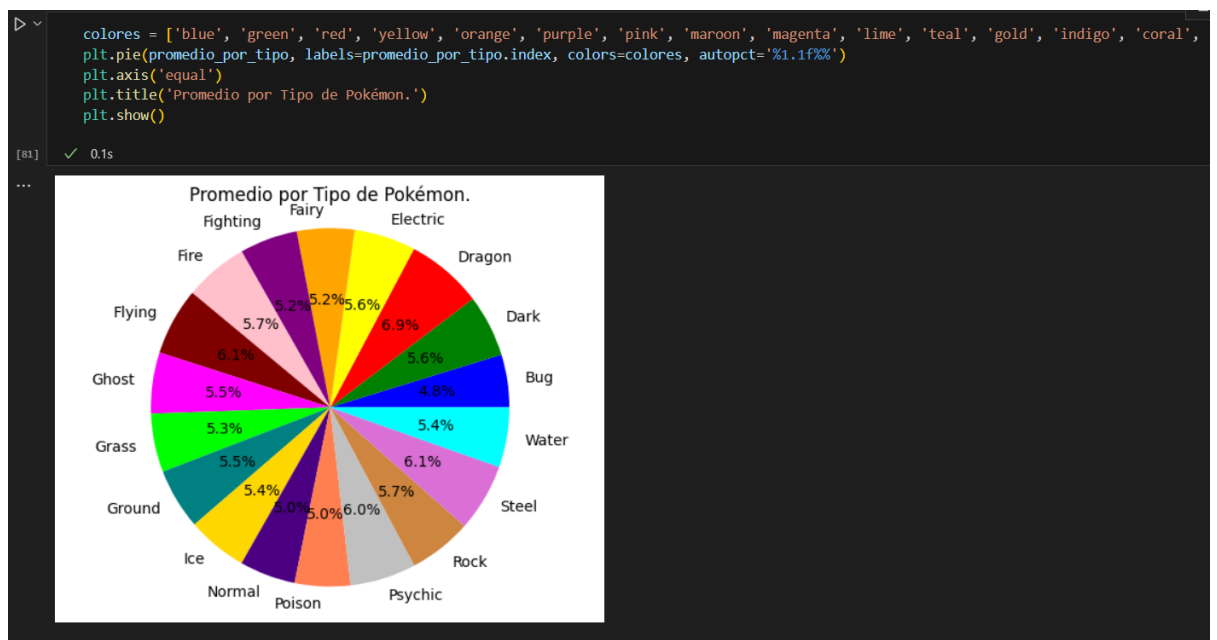
```

[80] ✓ 0.0s

...

Tipo con mayor porcentaje: Dragon

Vuelco los valores en un gráfico representativo.



Para los datos de los Pokémon Legendarios.

Cuento la cantidad de ocurrencias de cada valor en la columna "Legendary".

Obtengo el porcentaje de Pokémon que son legendarios.

```

legendary_counts = df['Legendary'].value_counts()
percentage_legacy = (legendary_counts[True] / len(df)) * 100

print(f"Porcentaje de pokémon legendarios: {percentage_legacy:.1f}%")

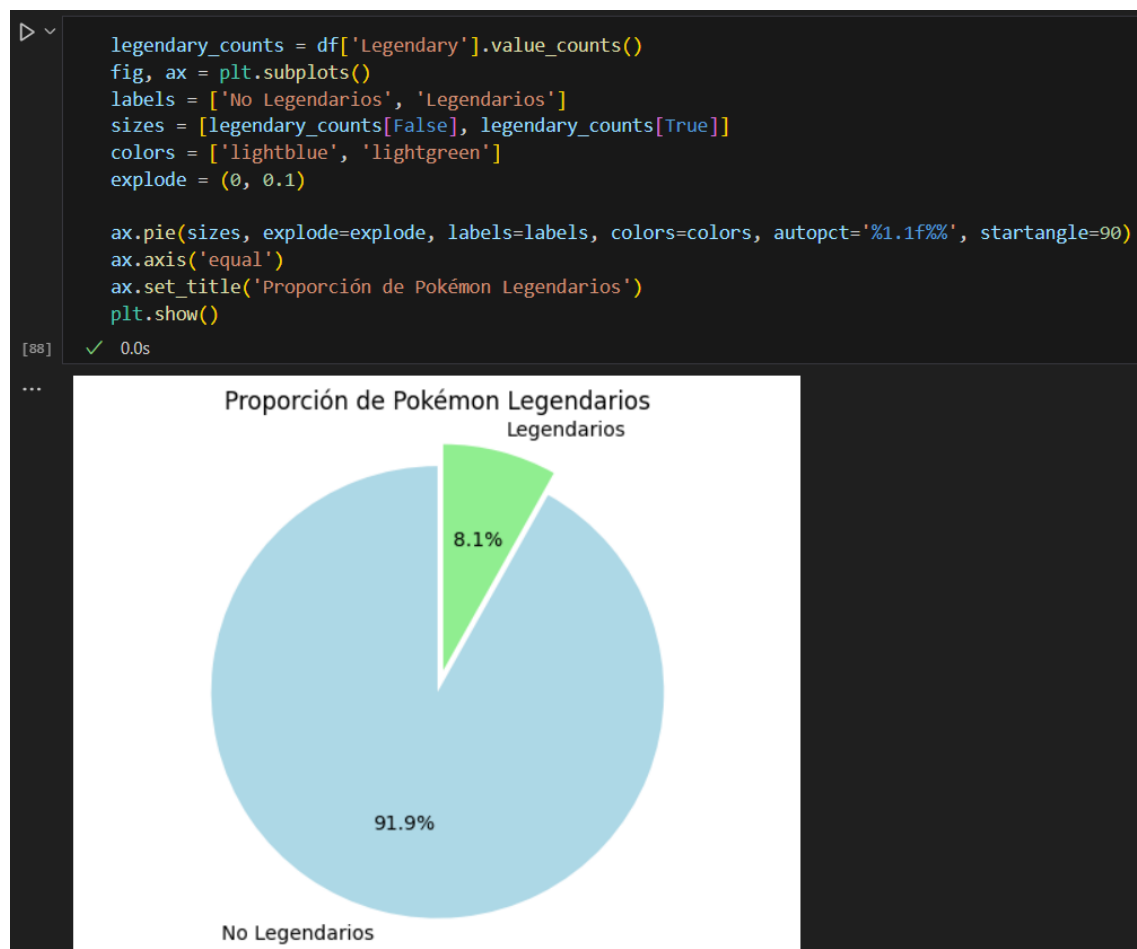
```

[87] ✓ 0.0s

... Porcentaje de pokémon legendarios: 8.1%

Cuento la cantidad de ocurrencias de cada valor en la columna "Legendary".

Defino los datos para el gráfico circular.



Realizo otro Análisis Exploratorio de Datos (EDA).

En esta parte, decido tomar como parte de los datos el ataque, la defensa, la salud (HP) y la velocidad.

```
[89] df[['Attack', 'Defense', 'HP', 'Speed']]
```

✓ 0.0s

...

	Attack	Defense	HP	Speed
0	49	49	45	45
1	62	63	60	60
2	82	83	80	80
3	100	123	80	80
4	52	43	39	65
...
795	100	150	50	50
796	160	110	50	110
797	110	60	80	70
798	160	60	80	80
799	110	120	80	70

800 rows × 4 columns

Realizo un gráfico de distribución para cada variable numérica.

De esta manera puedo percibir la cantidad de registros que tienen determinado valor en las variables.

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(9, 5))
axes = axes.flat
columnas_numeric = df[['Attack', 'Defense', 'HP', 'Speed']].select_dtypes(include=['float64', 'int']).columns
for i, column in enumerate(columnas_numeric):
    sns.histplot(
        data = df[['Attack', 'Defense', 'HP', 'Speed']],
        x = column,
        stat = "count",
        kde = True,
        color = (list(plt.rcParams['axes.prop_cycle'])*2)[i]["color"],
        line_kws= {'linewidth': 2},
        alpha = 0.3,
        ax = axes[i]
    )
    axes[i].set_title(column, fontsize = 10, fontweight = "bold")
    axes[i].tick_params(labelsize = 8)
    axes[i].set_xlabel("")

fig.tight_layout()
plt.subplots_adjust(top = 0.9)
fig.suptitle('Distribución de variables numéricas', fontsize = 10, fontweight = "bold")
```

[98] ✓ 0.6s

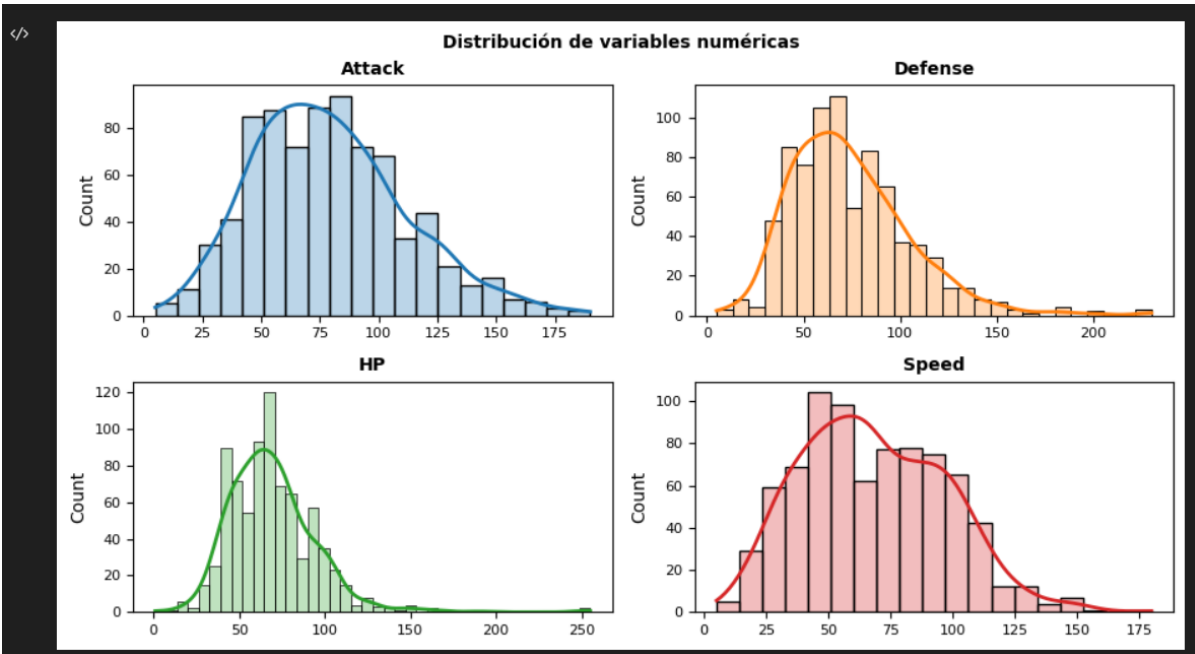
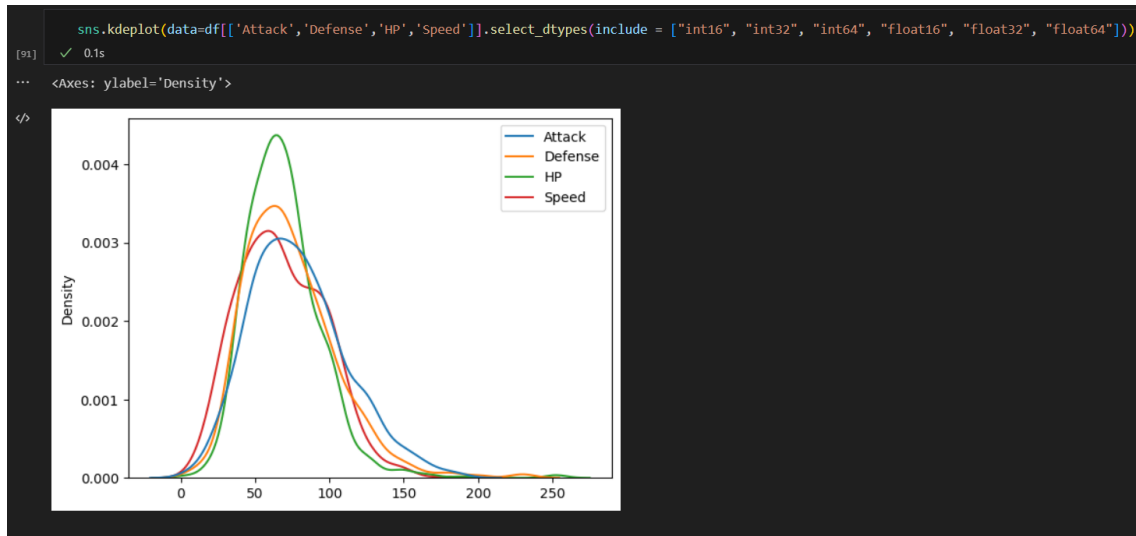


Gráfico de densidad en donde coloco los 4 atributos juntos.



Ahora trabajaré con otro sector de los datos.

Decido crear un concepto nuevo denominado Calificación partiendo de los valores de Total y etiquetándolos según un criterio.

```
df['Calificacion'] = pd.cut(df.Total, bins=[0,450 ,650, float('inf')], labels=['Debil','Moderado','Fuerte'])
```

[92] ✓ 0.0s

Noto que ya se reconoce a Calificación como una columna más.

```
df[['Generation','#','Calificacion','Total']].shape
```

[93] ✓ 0.0s

... (800, 4)

Imprimo los datos obtenidos.

```
[94] df[['Generation','#','Calificacion','Total']]
```

	Generation	#	Calificacion	Total
0	1	1	Debil	318
1	1	2	Debil	405
2	1	3	Moderado	525
3	1	3	Moderado	625
4	1	4	Debil	309
...
795	6	719	Moderado	600
796	6	719	Fuerte	700
797	6	720	Moderado	600
798	6	720	Fuerte	680
799	6	721	Moderado	600

800 rows × 4 columns

Obtengo la cantidad de registros por Calificación.

```
[95] df.Calificacion.value_counts()
```

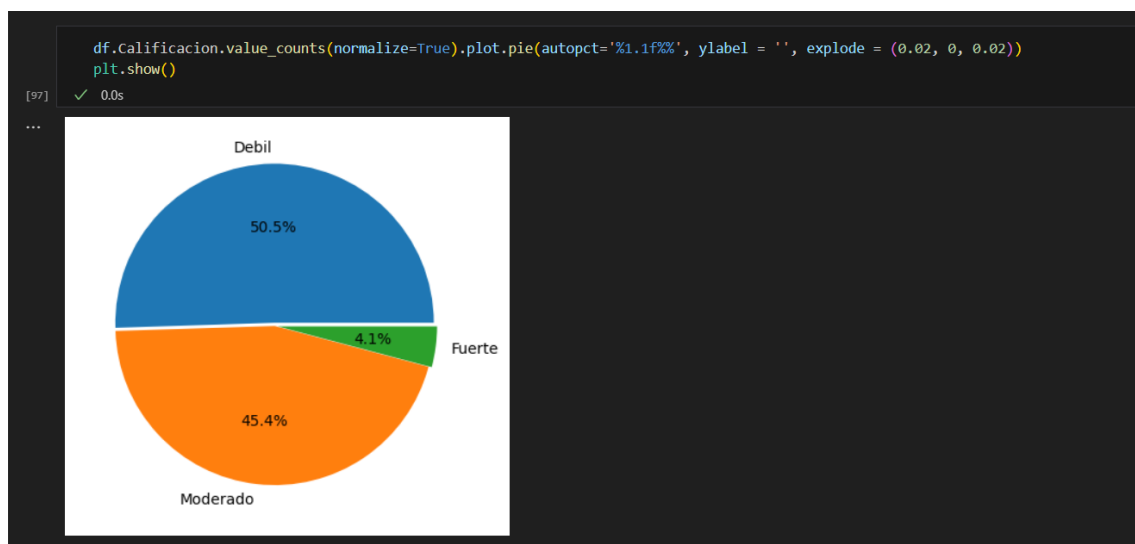
```
Calificacion
Debil      404
Moderado   363
Fuerte     33
Name: count, dtype: int64
```

Outliers normalizados.

```
[96] df.Calificacion.value_counts(normalize=True)
```

```
... Calificacion
     Debil      0.50500
     Moderado  0.45375
     Fuerte    0.04125
     Name: proportion, dtype: float64
```

Graficamos y notamos que una pequeña parte de los datos es apenas calificada como Fuerte, predominando el Débil, y de cerca los Moderados.



Obtengo el total de la suma de sus Total.

```
[98] df.groupby(['Calificacion'])['Total'].sum()
```

```
... Calificacion
     Debil      135869
     Moderado  189133
     Fuerte    23080
     Name: Total, dtype: int64
```


Determino la media.

```
df.groupby(['Calificacion'])['Total'].mean()
```

[99] ✓ 0.0s

```
... Calificacion
     Debil      336.309406
     Moderado  521.027548
     Fuerte    699.393939
     Name: Total, dtype: float64
```

Decido agrupar mis datos por Generación, además quiero obtener el número de ID de cada Pokémon por generación.

También la suma de sus totales.

Noto que hay más cantidad de Pokémon en la primera generación.

También hay más cantidad de valor Total en la quinta generación.

```
grouped = df.groupby("Generation").agg({"#": "count", "Total": "sum"})
print(grouped)
```

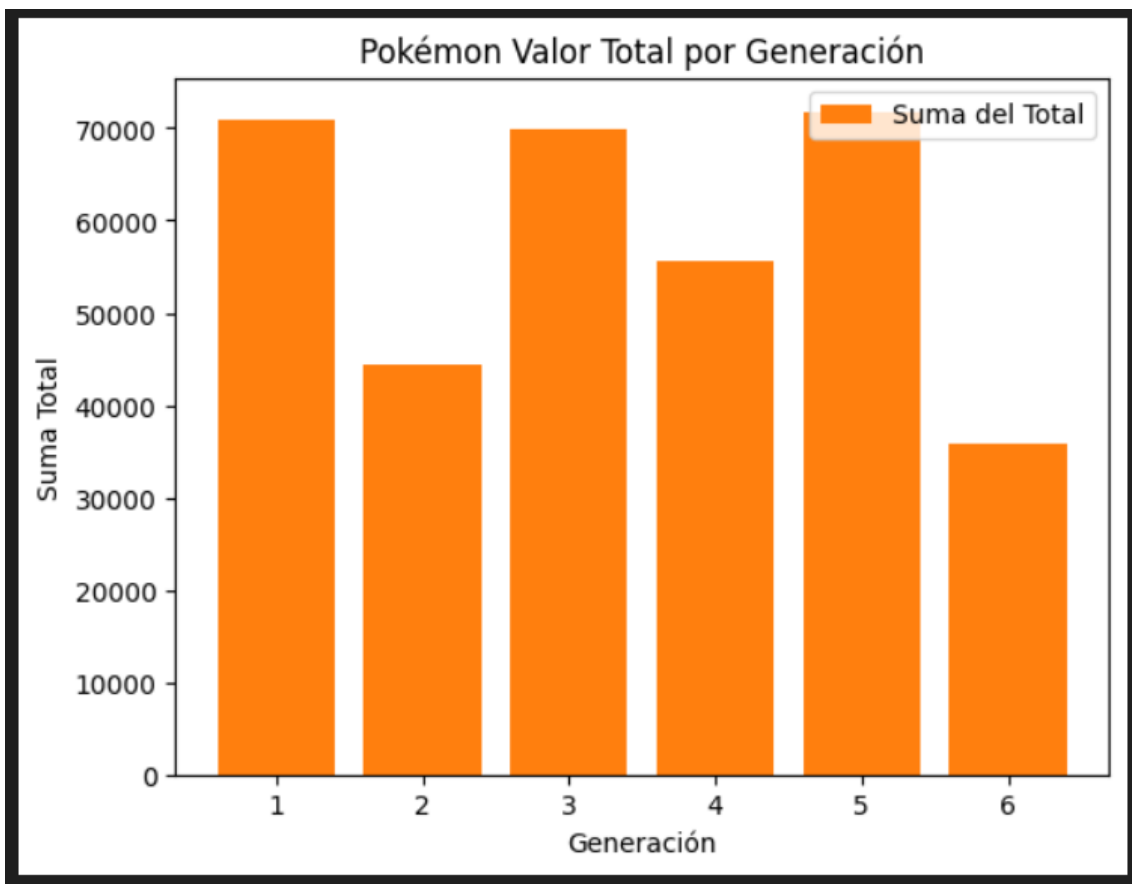
[100] ✓ 0.0s

```
...      #  Total
Generation
1      166  70851
2      106  44338
3      160  69796
4      121  55541
5      165  71773
6       82  35783
```

Armo una gráfica con las generaciones y el Total.

```
fig, ax = plt.subplots()
generaciones = grouped.index
cantidad_pokemones = grouped["#"]
suma_total = grouped["Total"]
ax.bar(generaciones, cantidad_pokemones)
ax.bar(generaciones, suma_total, label="Suma del Total")
ax.set_xlabel("Generación")
ax.set_ylabel("Suma Total")
ax.set_title("Pokémon Valor Total por Generación")
ax.legend()
plt.show()
```

[101] ✓ 0.1s



Aplicaré un análisis más para las características de HP, Attack, Defense, Sp. Atk, Sp. Def y Speed como características particulares.

Con el Total como columna objetivo aplicando una regresión de Random Forest.

Importo las librerías que utilizaré.

```
[102] import pandas as pd
      from sklearn.ensemble import RandomForestRegressor
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import mean_absolute_error, mean_squared_error
      ✓ 0.2s
```

Importo nuevamente el dataset.

```
[103] 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

Selecciono las columnas de características y el objetivo.

```
[104] columnas_caracteristicas = ['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']
      columna_objetivo = 'Total'
      ✓ 0.0s
```

Divido los datos en conjuntos de entrenamiento y prueba.

```
[105] X = df[columnas_caracteristicas]
      y = df[columna_objetivo]
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
      ✓ 0.0s
```

Creo el modelo de regresión con Random Forest.

```
[106] regresor_rf = RandomForestRegressor(random_state=42)
      ✓ 0.0s
```

Entreno el modelo.

```
[107] ✓ 0.2s
```

```
regresor_rf.fit(X_train, y_train)
```

```
... RandomForestRegressor
```

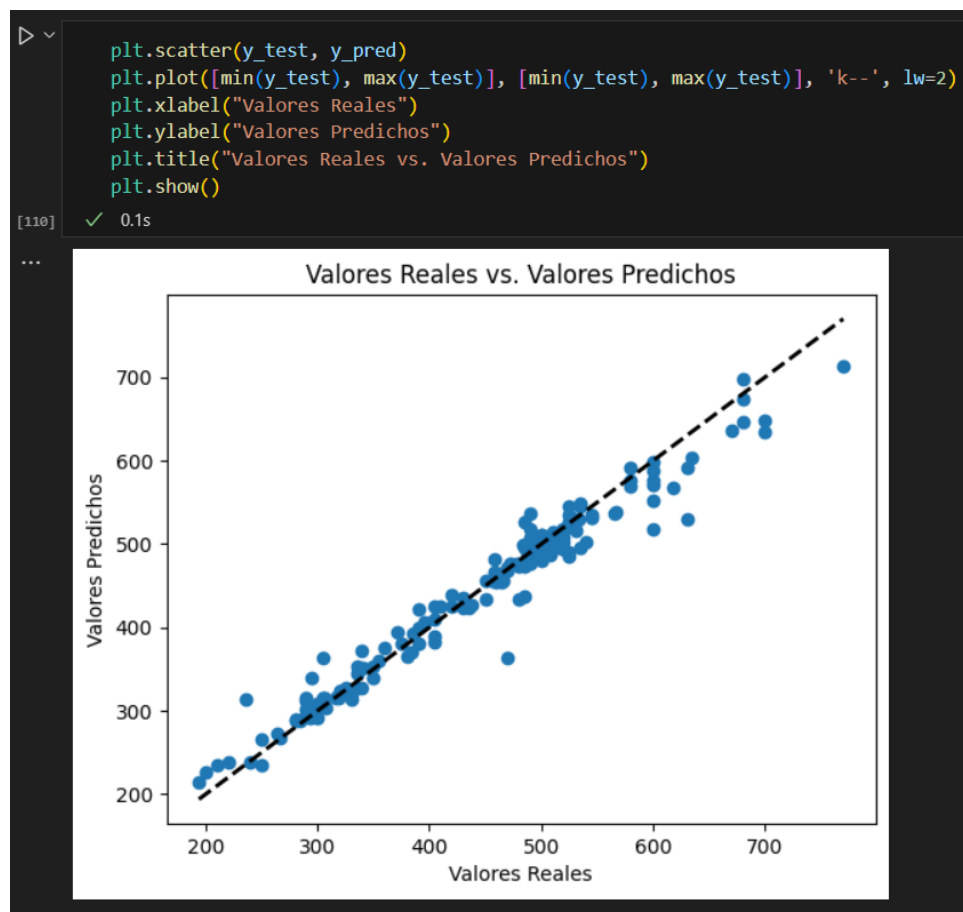
```
RandomForestRegressor(random_state=42)
```

Realizo las predicciones en el conjunto de prueba.

```
[109] ✓ 0.0s
```

```
y_pred = regresor_rf.predict(X_test)
```

Gráfico de predicción vs valores reales.

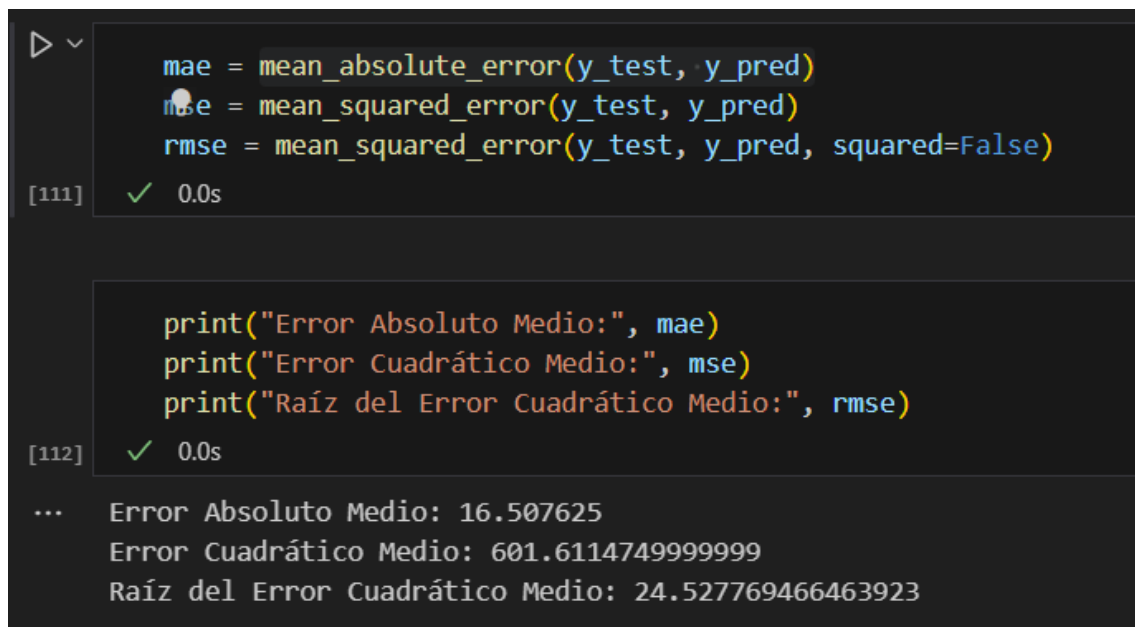


Finalmente analizaré el Error Absoluto Medio, el Error Cuadrático Medio y la Raíz del error cuadrático medio.

El Error Absoluto Medio mide la magnitud promedio de los errores en las predicciones del modelo. Se calcula tomando la diferencia absoluta entre cada valor predicho y el valor real, y luego promediando esos errores.

El Error Cuadrático Medio calcula el promedio de los errores al cuadrado entre las predicciones y los valores reales. Al elevar los errores al cuadrado, esta métrica da más peso a los errores más grandes.

La Raíz del Error Cuadrático Medio es simplemente la raíz cuadrada del MSE. Proporciona una medida del error promedio, similar al MAE, pero está en la misma escala que los valores originales.



```

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

[111] ✓ 0.0s

print("Error Absoluto Medio:", mae)
print("Error Cuadrático Medio:", mse)
print("Raíz del Error Cuadrático Medio:", rmse)

[112] ✓ 0.0s

... Error Absoluto Medio: 16.507625
Error Cuadrático Medio: 601.6114749999999
Raíz del Error Cuadrático Medio: 24.527769466463923

```

Algunas conclusiones que destaco son las siguientes.

La Raíz del Error Cuadrático Medio dio 24.52 y es un valor consistente ya que está por debajo del valor 50 que representa el 10% del valor de la media de Total que es 500, recordando que total es la columna objetivo.

Sin embargo, lo que complica al análisis es el valor arrojado en el Error Cuadrático Medio que es de 601.61.

Al tener un MSE alto, significa que los errores individuales de las predicciones del modelo están siendo penalizados de manera considerable al elevarse al cuadrado.

Por lo que el modelo de regresión con Random Forest, para este caso, puede tener un rendimiento deficiente en términos de la precisión de la predicción de la variable objetivo.

Algunas posibles conclusiones generales basadas en el análisis son:

El modelo de regresión con Random Forest puede no estar capturando adecuadamente las relaciones y patrones en los datos del conjunto de entrenamiento. Esto puede deberse a una configuración incorrecta de los hiperparámetros o a una falta de características relevantes.

Los datos utilizados para entrenar el modelo pueden contener ruido, valores atípicos o falta de representatividad en relación con la variable objetivo. Esto puede afectar la capacidad del modelo para realizar predicciones precisas.

El modelo puede estar sobre ajustando los datos de entrenamiento, lo que significa que está capturando demasiado los detalles y patrones específicos de los datos de entrenamiento, pero no generaliza bien a nuevos datos.