Práctica 3

Python y minería de datos

Índice

INICIO……3

PREPROCESAMIENTO DE DATOS……..7

CLASIFICACIÓN……10

PREDICCIÓN……12

CLUSTERING……..14

WEKA VS PYTHON……15

INICIO

Primeramente, vamos a importar la siguiente librería para poder ver por pantalla las siguientes tablas.

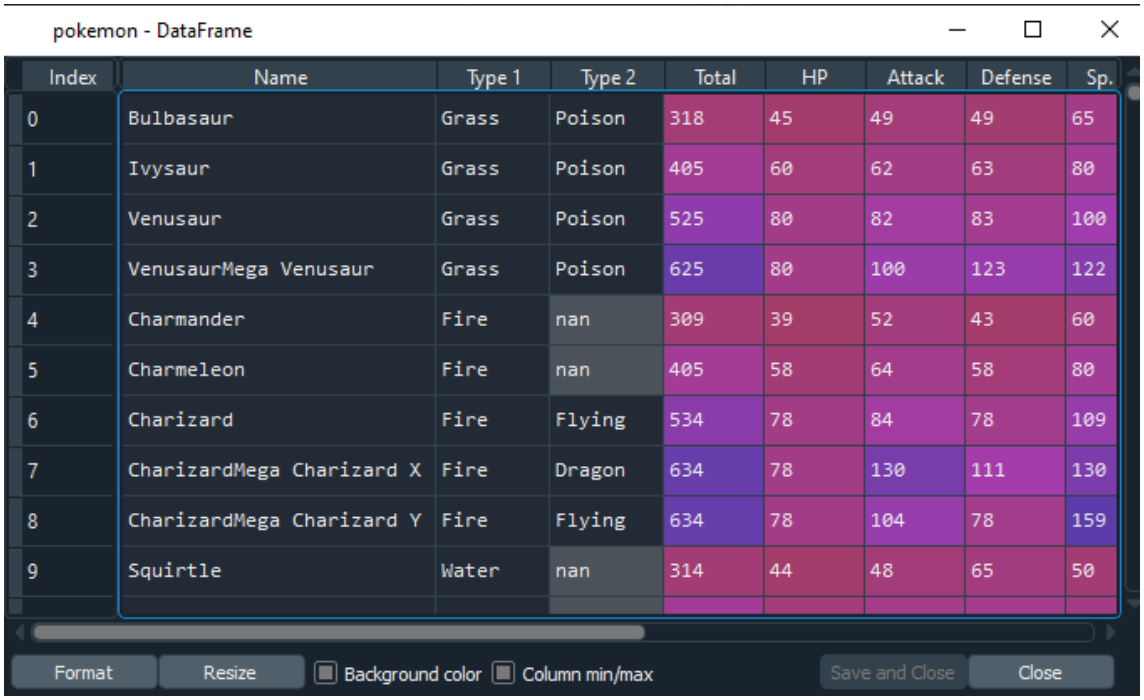
```
import pandas as pd
```

En segundo lugar vamos a importar nuestro fichero csv a Python, y lo vamos a guardar en la variable pokemon.

pokemon = pd.read_csv("C:/Users/vicente candela pere/Desktop/3º de carrera/Mineria de Datos/practicas/Practica 3/Pokemon.csv", sep = ",")

De manera que ahora podemos ver por pantalla una tabla como la siguiente, con todos los valores de nuestro csv de manera que sale en negro los valores nominales y de color lila los valores numéricos.

El entorno de desarrollo lo que hace es enseñarnos las variables, esto es una buena manera de saber con qué variables podemos trabajar y con las que no, según lo que queramos seleccionar. Otra cosa que también puede hacer es ordenarnos las variables.

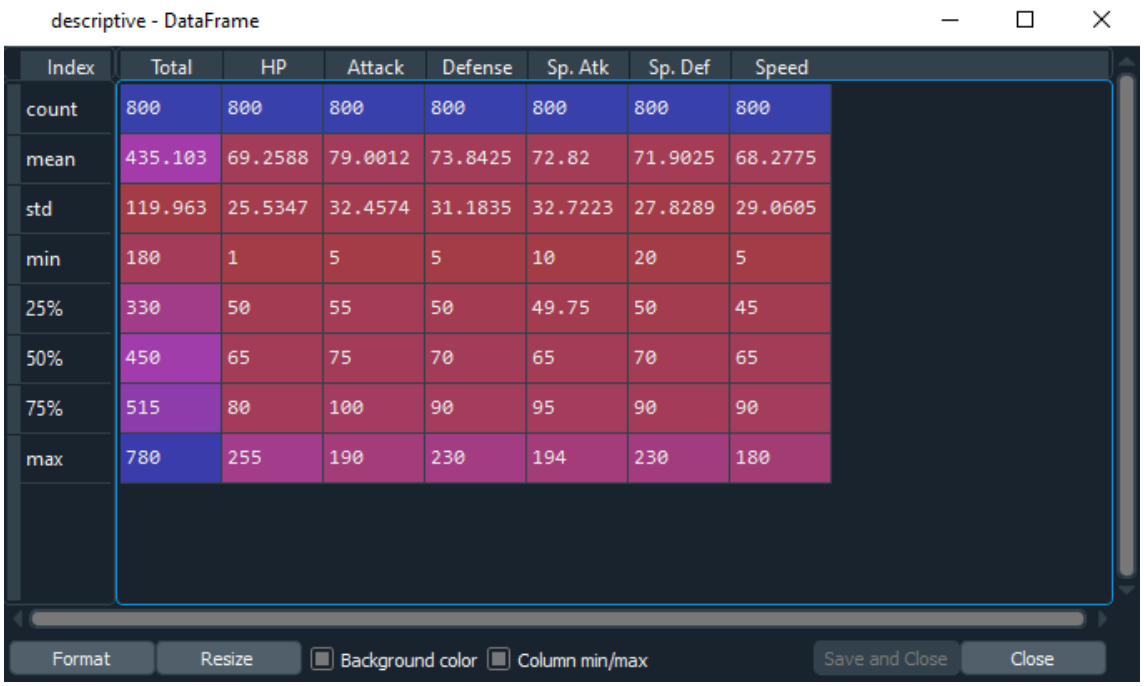


Index	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp.
0	Bulbasaur	Grass	Poison	318	45	49	49	65
1	Ivysaur	Grass	Poison	405	60	62	63	80
2	Venusaur	Grass	Poison	525	80	82	83	100
3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122
4	Charmander	Fire	nan	309	39	52	43	60
5	Charmeleon	Fire	nan	405	58	64	58	80
6	Charizard	Fire	Flying	534	78	84	78	109
7	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130
8	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159
9	Squirtle	Water	nan	314	44	48	65	50

Después, si introducimos la siguiente instrucción

```
descriptive = pokemon.describe()
```

Y se nos mostrara una tabla a modo resumen de nuestros valores numéricos. Como podemos observar a la izquierda tenemos la variable conteo que nos dice cuantas veces ha contado esa variable, en nuestro caso todos tienen el mismo conteo ya que me aseguré de que no hubiese ningún valor nulo en el CSV. También podemos observar que nos muestra la media, el mínimo, el máximo, los cuartiles…



Index	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
count	800	800	800	800	800	800	800
mean	435.103	69.2588	79.0012	73.8425	72.82	71.9025	68.2775
std	119.963	25.5347	32.4574	31.1835	32.7223	27.8289	29.0605
min	180	1	5	5	10	20	5
25%	330	50	55	50	49.75	50	45
50%	450	65	75	70	65	70	65
75%	515	80	100	90	95	90	90
max	780	255	190	230	194	230	180

Aquí podemos ver como con las siguientes líneas de código a través de nosotros indicarle el CSV, la columna que queremos que muestre y el nombre de la función, nos muestra por consola los valores que puede tomar ese atributo.

Para este ejemplo he puesto el atributo generación el cual puede mostrar los valores de G1 a G6 y el atributo Legendario que nos mostrara verdadero o falso según el Pokémon.

```
pokemon["Generation"].unique()
pokemon["Legendary"].unique()
```

```
In [10]: pokemon["Generation"].unique()
Out[10]: array(['G1', 'G2', 'G3', 'G4', 'G5', 'G6'], dtype=object)

In [11]: pokemon["Legendary"].unique()
Out[11]: array(['FALSO', 'VERDADERO'], dtype=object)
```

Con las siguientes líneas de código lo que vamos a hacer es contar los valores que llega a tomar cada valor del atributo.

Esto se hace de una manera muy parecida al comando que habíamos hecho anteriormente. Ponemos el CSV, después que columna queremos que elija y después el nombre de la función

```
pokemon["Generation"].value_counts()
pokemon["Legendary"].value_counts()
```

Aquí podemos apreciar los valores que han tomado nuestras categorías de manera que por ejemplo G1 ha tomado 166, y también podemos observar como falso ha obtenido un valor de 735.

```
In [12]: pokemon["Generation"].value_counts()
Out[12]:
G1    166
G5    165
G3    160
G4    121
G2    106
G6     82
Name: Generation, dtype: int64

In [13]: pokemon["Legendary"].value_counts()
Out[13]:
FALSO        735
VERDADERO     65
Name: Legendary, dtype: int64
```

Lo siguiente que vamos a ver es un gráfico de correlaciones.

El comando pairplot sirve para poner en los ejes nuestras variables y como emparejarlas para ver que relaciones hay entre cada una de ellas.

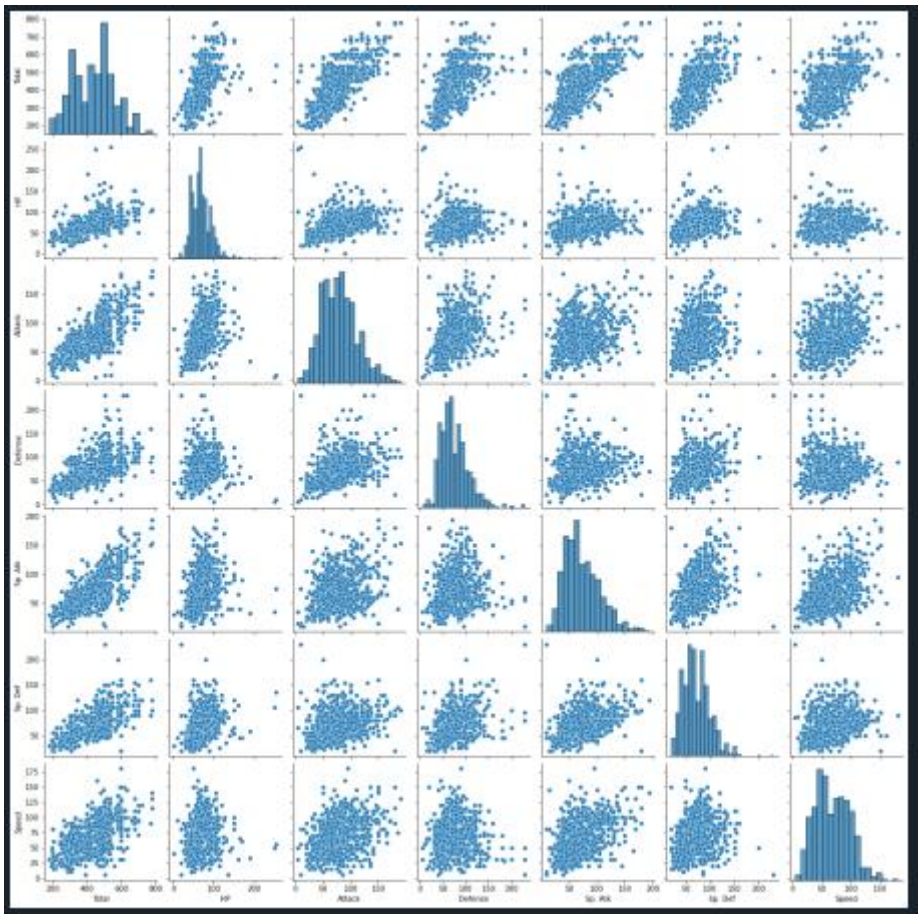
Pero como podemos ver le tenemos que pasar un filtro, ese filtro excluirá a todos los atributos que sean de tipo object, lo podemos hacer de la siguiente forma.

Primeramente, importamos como siempre, la librería que nos va a hacer falta para poder usar las funciones que necesitaremos posteriormente.

Después mediante la segunda línea de código pasamos ese filtro de manera que excluya a los atributos object, es decir, las cadenas de caracteres.

```
import seaborn as sns

sns.pairplot(pokemon.select_dtypes(exclude=[object]))
```

Y como podemos observar nos devuelve el grafico en la ventana de plots

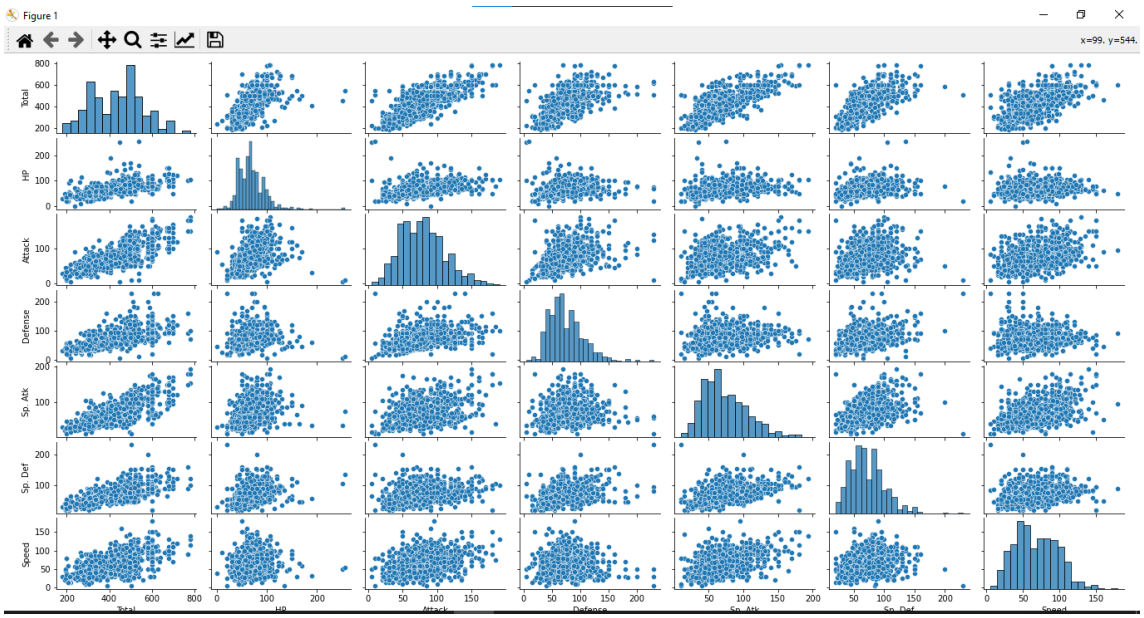


Como no se observan muy bien los datos, vamos a ejecutar las siguientes líneas de código para que se nos habrá en una nueva ventana y además este sea dinámico.

Son las mismas dos líneas de código, pero hemos añadido la línea de mas arriba.

```
%matplotlib auto
import seaborn as sns

sns.pairplot(pokemon.select_dtypes(exclude=[object]))
```

De esta forma se nos abre en una ventana a parte una grafica como la siguiente



Con estos gráficos podemos ver como nuestras variables están relacionadas, y también podemos ver como nuestras variables cambian para cada punto.

Preprocesamiento de datos

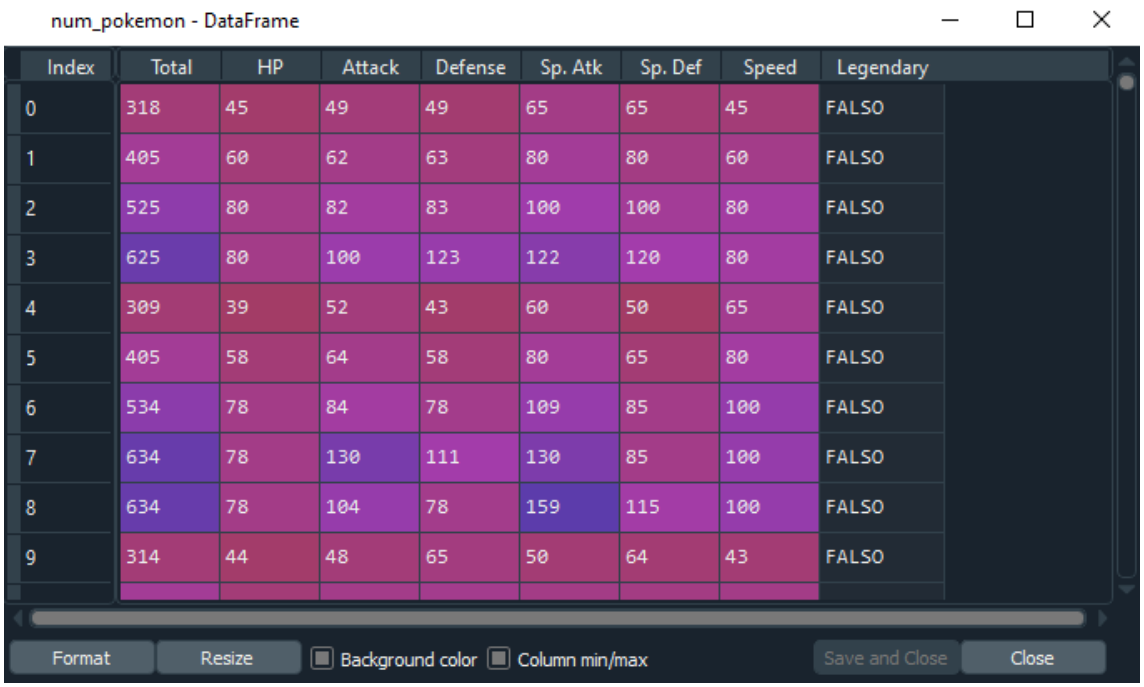
Para los siguientes algoritmos que vamos a usar no debemos utilizar ningún atributo de tipo nominal por lo que usando las siguientes líneas de código hacemos que se quiten dichas columnas, para nuestro ejemplo (ya que nosotros sabemos que columnas debemos eliminar).

Anteriormente habíamos visto una manera de excluir los objects, para este caso vamos a hacerlo de una forma diferente, que es diciendo que columnas queremos quitar.

```
num_pokemon = pokemon.drop(["Name", "Type 1", "Type 2", "Generation"], axis = 1)
```

Con esto le decimos que nos quite las columnas de name, type1, type 2 y generación. Además, también hay que mencionar que el axis lo que hace es quitarnos la columna, si eso fuese un 0 eliminaría filas.

Y se nos quedaría algo como la siguiente imagen



Index	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Legendary
0	318	45	49	49	65	65	45	FALSO
1	405	60	62	63	80	80	60	FALSO
2	525	80	82	83	100	100	80	FALSO
3	625	80	100	123	122	120	80	FALSO
4	309	39	52	43	60	50	65	FALSO
5	405	58	64	58	80	65	80	FALSO
6	534	78	84	78	109	85	100	FALSO
7	634	78	130	111	130	85	100	FALSO
8	634	78	104	78	159	115	100	FALSO
9	314	44	48	65	50	64	43	FALSO

Después deberemos hacer que nuestra variable de clase en este caso Legendario se convierta también a numérico ya que ahora es de tipo nominal con los valores verdadero y falso, como hemos visto antes. Por lo que para hacer eso deberemos poner las siguientes líneas de Código

```
num_pokemon.loc[num_pokemon.loc[:, "Legendary"] == "VERDADERO", "Legendary"] = 1
num_pokemon.loc[num_pokemon.loc[:, "Legendary"] == "FALSO", "Legendary"] = 0

num_pokemon["Legendary"].unique()
```

Que nos mostrará los siguientes mensajes por consola.

```
In [19]: num_pokemon = pokemon.drop(["Name", "Type 1", "Type 2",
"Generation"], axis = 1)

In [20]: num_pokemon.loc[num_pokemon.loc[:, "Legendary"] == "VERDADERO",
"Legendary"] = 1
    ...: num_pokemon.loc[num_pokemon.loc[:, "Legendary"] == "FALSO",
"Legendary"] = 0

In [21]: num_pokemon["Legendary"].unique()
Out[21]: array([0, 1], dtype=object)
```

Con el comando unique se nos muestra por consola como ahora si nuestro atributo de clase ahora es numérico.

Y aquí podemos ver como se nos quedaría.

num_pokemon - DataFrame

Index	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Legendary
0	318	45	49	49	65	65	45	0
1	405	60	62	63	80	80	60	0
2	525	80	82	83	100	100	80	0
3	625	80	100	123	122	120	80	0
4	309	39	52	43	60	50	65	0
5	405	58	64	58	80	65	80	0
6	534	78	84	78	109	85	100	0
7	634	78	130	111	130	85	100	0
8	634	78	104	78	159	115	100	0
9	314	44	48	65	50	64	43	0

Format Resize Background color Column min/max Save and Close Clos

Una vez terminada la fase del preprocesamiento de datos nos disponemos a realizar los diferentes algoritmos de minería de datos.

CLASIFICACIÓN

Lo primero que deberemos hacer es importar las diferentes librerías que podemos observar para así poder llamar a las funciones que vamos a utilizar posteriormente.

```
#clasificacion
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
```

Lo siguiente que debemos hacer es separar nuestro data set en dos conjuntos, en un conjunto todas nuestras variables explicativas, y el otro en nuestra variable objetivo que es el atributo clase (Legendario).

Esto lo hacemos mediante las siguientes líneas de código para así poder dividirlas.

```
X = num_pokemon.values[:, :-1] #Variables explicativas
y = num_pokemon.values[:, -1] #variable objetivo
y = y.astype("int")
```

Donde aquí podemos ver que le decimos que coja todas las filas y todas las columnas hasta el índice -1, lo cual quiere decir que coja todos los datos menos nuestra ultima columna. Esto lo hacemos porque nuestra variable objetivo es la clase que esta la última y es la que queremos analizar.

También cabe mencionar que para que no haya un error le decimos que la " Y " es de tipo numérica ya que antes nos lo devolvía como tipo objeto.

Ahora lo que vamos a hacer es crear el modelo con las siguientes líneas de código donde el criterio "entropy" es uno de los muchos que nos podemos encontrar por internet, pero para este caso usaremos este.

Después, donde dice el max_depth es el número de veces que va a dividirse el árbol en una rama. Eso quiere decir que en este caso nuestro árbol va a tener 4 ramas.

Y posteriormente ajustamos nuestro modelo con la siguiente línea de código.

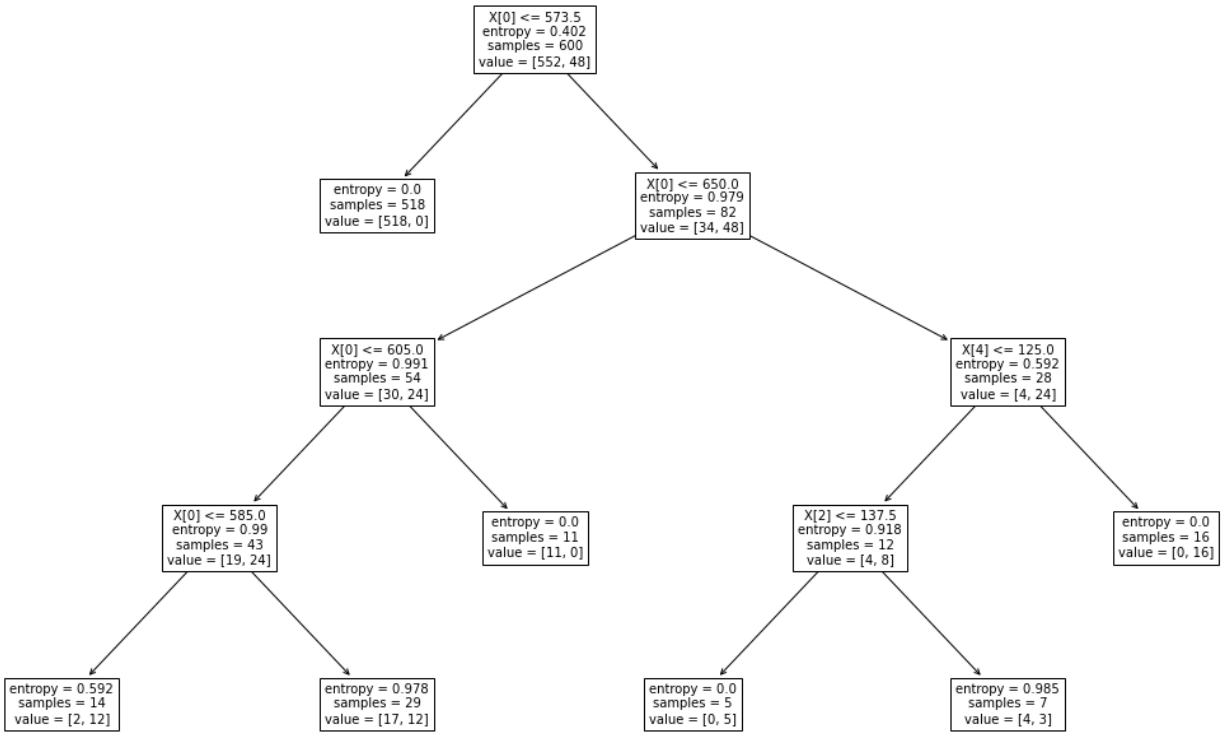
```
modelo = DecisionTreeClassifier(criterion='entropy', max_depth=4)

modelo = modelo.fit(X, y)
```

Por último, con las siguientes líneas de código hacemos que nuestro árbol se muestre por pantalla.

```
fig, ax = plt.subplots(figsize=(20, 12)) #Tamaño del gráfico
tree.plot_tree(modelo, fontsize = 10)
```

Aquí podemos ver nuestro árbol que se separa en cuatro, el nodo de arriba es el nodo raíz, destacar que uno de los fallos de Python es que no nos dice el nombre de la variable.

Una cosa que podemos hacer es mediante la siguiente línea de código ver cuales son los nombres de las columnas para guiarme.



Lo que hace es sacar los nombres de las columnas.

```
num_pokemon.columns
```

Lo que me dice el árbol es que si se cumple mi condición del nodo iremos para la izquierda y si no se cumple iremos a la derecha y así hasta llegar al final de alguna de las ramas.

PREDICCIÓN

Lo primero que haremos será importar nuestras librerías para así después poder utilizar los comandos de estas.

Estas librerías sirven para que nos hagan la separación entre training y test.

```
#prediccion

from sklearn.model_selection import train_test_split #Separar el data set en training
from sklearn.metrics import accuracy_score #Métricas de la predicción del modelo. Pre
from sklearn.metrics import confusion_matrix
```

Hacemos lo mismo de antes, separamos las X y las Y con las siguientes líneas de código.

```
X = num_pokemon.values[:, :-1] #Variables explicativas
y = num_pokemon.values[:, -1] #variable objetivo
y = y.astype("int")
```

Con la siguiente línea de código la función lo que hace es separar nuestras X y nuestras Y en cuatro variables distintas. En las X que son de train, en las X que son de test, en las Y que son de train y en las Y que son de test.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

Así lo que podemos hacer es crear el modelo que hemos hecho como antes con las siguientes líneas de código, pero con la diferencia de que a este modelo le vamos a dar las variables de train para entrenar. Y luego ya para predecir le damos la variable X_test para guardarlo en la variable y_prediccion.

```
modelo = DecisionTreeClassifier(criterion='entropy', max_depth=4)

modelo = modelo.fit(X_train, y_train)

y_prediccion = modelo.predict(X_test)

accuracy_score(y_test, y_prediccion)
```

Y así con la siguiente línea de código lo que hacemos es comparar los resultados que habíamos obtenido de X_test, que recordemos se han guardado en y_prediccion, con la variable y_test la cual habíamos sacado de la separación de antes.

Así pues, ya habríamos obtenido los valores de la matriz de confusión.

```
accuracy_score(y_test, y_prediccion)
```

El cual nos da un valor del 94% de confianza lo cual no está nada mal.

```
In [34]: accuracy_score(y_test, y_prediccion)
Out[34]: 0.945
```

Con las siguientes líneas de código lo que vamos a hacer es dibujar la matriz de confusión por consola.

```
pd.DataFrame(
    confusion_matrix(y_test, y_prediccion),
    columns=['Predicted Not Class', 'Predicted Class'],
    index=['True Not Class', 'True Class']
)
```

La cual nos dará una cosa como esta.

```
Out[35]:
                Predicted Not Class  Predicted Class
True Not Class                  179                4
True Class                        7               10
```

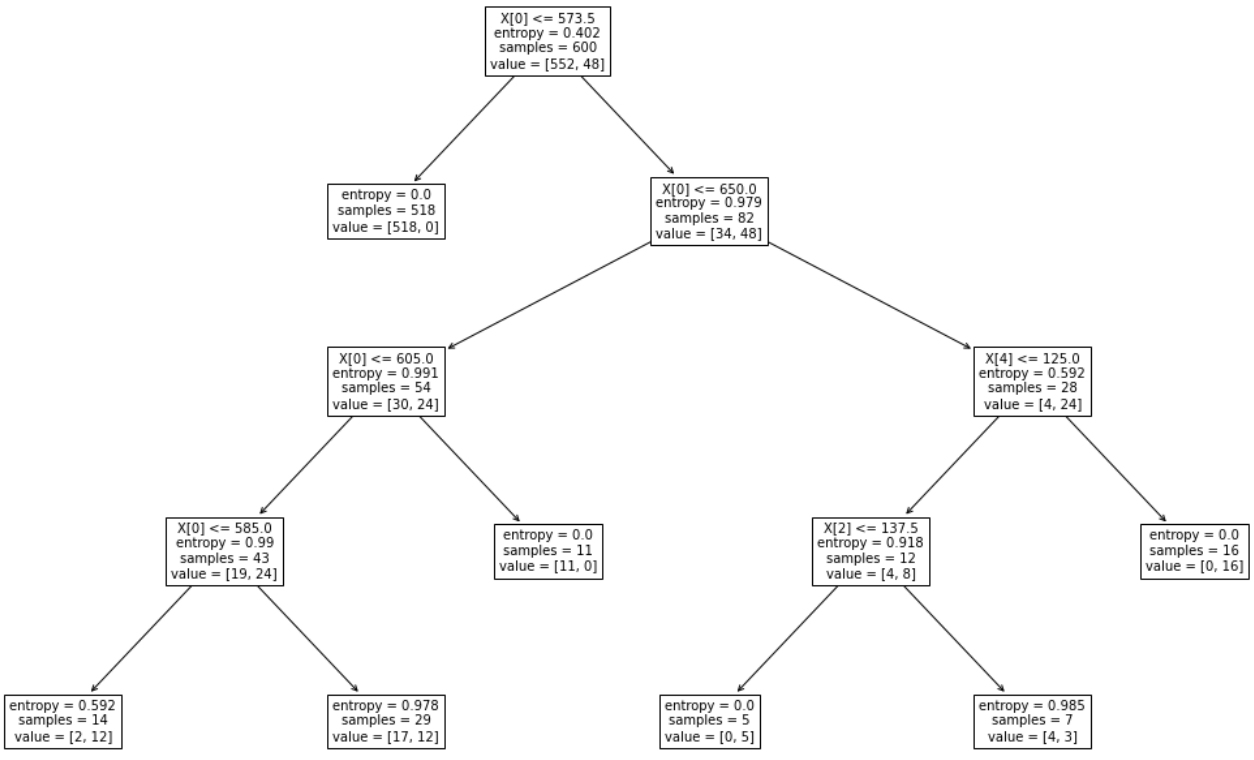
Esta matriz dice que 179 veces se ha dicho que nuestro Pokémon iba a no ser legendaria y efectivamente no era legendario, también podemos ver que ha dicho en 7 veces que el Pokémon iba a no ser legendario y al final sí que lo fue.

Después, podemos ver que dijo que en 10 veces que el Pokémon si iba a ser legendario y si lo fue, y también podemos ver como en 4 ocasiones se dijo que el Pokémon iba a ser legendario pero no lo fue.

Con la siguiente línea de código lo que hace es dibujarnos nuestro árbol por pantalla.

```
fig, ax = plt.subplots(figsize=(20, 12)) #Tamaño del gráfico
tree.plot_tree(modelo, fontsize = 10)
```

Aquí podemos observar el árbol q nos muestra por pantalla.



CLUSTERING

Lo primero que vamos a hacer y como hemos hecho hasta ahora es importar las librerías de las cuales necesitaremos sus funciones más en adelante.

El método que vamos a usar para el clustering es el algoritmo de Kmeans, o como nosotros lo hemos estado viendo a lo largo del curso, Simple Kmeans.

```
#CLUSTERING
from sklearn.cluster import KMeans
```

Después lo que debemos hacer es seleccionar los atributos con los que vamos a querer trabajar.

Eso lo hacemos a través de la siguiente línea de código.

```
K_pokemon = num_pokemon[["HP", "Attack", "Legendary"]]
```

Donde como podemos ver he seleccionado el atributo de HP (vida), el atributo de ataque y el atributo de clase Legendario. He elegido estos atributos para poder comparar con los resultados de la Práctica 2 ya que elegí estos dos mismos atributos para hacer el clúster de simple Kmeans.

Para este ejemplo solo vamos a tener 2 ejes donde uno será vida y otro ataque, no pensemos que por que puse 3 atributos arriba me va a salir una coordenada en 3D ni nada por el estilo.

Después lo que decimos con la siguiente línea de código es en cuantos clusters queremos separarlos. Para nuestro ejemplo he optado por dividirlo en 4 clusters diferentes.

```
kmeans_pokemon = KMeans(n_clusters = 4)
```

Más tarde simplemente tendremos que ajustar el modelo con la siguiente línea de código.

```
#Ajustar modelo
kmeans_pokemon.fit(K_pokemon)
```

Para así después poder dibujar el clúster

```
#Dibujar el clústering
centroids = kmeans_pokemon.cluster_centers_
labels = kmeans_pokemon.labels_
```

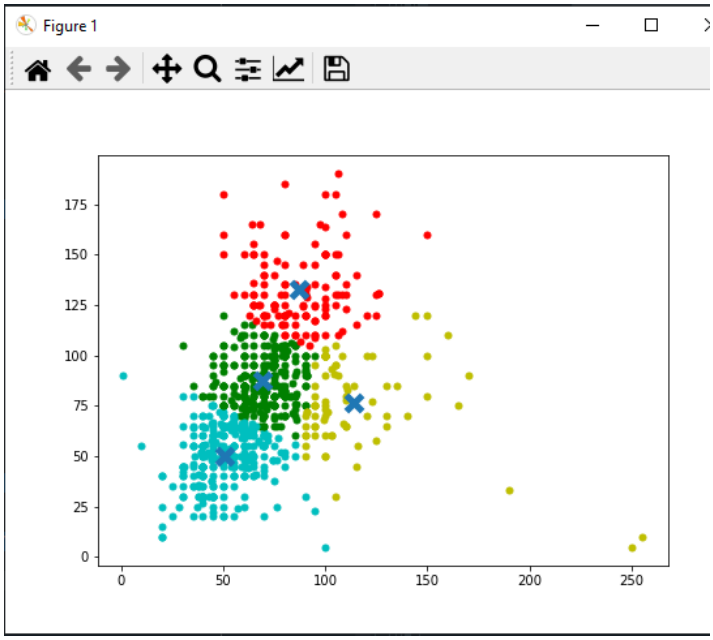
Después, deberemos de asignar unos colores diferentes a nuestros centroides, los cuales para este ejemplo he usado el verde, rojo, azul y amarillo.

Aquí como podemos ver están las funciones que nos van a sacar los centroides, para así finalmente mostrárnoslo por pantalla.

Marker = x será como nosotros lo veremos por pantalla, lo podemos cambiar a la letra que queramos y también mencionar que S es el tamaño que nosotros le queramos dar.

```
#Dibujar los puntos con los colores de cada clúster
colors = ["g.","r.","c.","y."]
for i in range(len(K_pokemon)):
    plt.plot(K_pokemon.iloc[i,0], K_pokemon.iloc[i,1], colors[labels[i]], markersize = 10)

#Dibujar los centroides de cada clúster
plt.scatter(centroids[:, 0],centroids[:, 1], marker = "x", s=150, linewidths = 5, zorder = 10)

plt.show()
```

Este sería el resultado que hemos obtenido tras ejecutar la anterior función donde en el eje X vemos hp y en el eje Y vemos ataque, y los 4 clusters que habíamos pedido que nos mostrase.



Valoración de Python vs Weka

Personalmente creo que Weka ha sido mucho más fácil de entender, la curva de aprendizaje ha sido muy cómoda donde he podido observar que sin saber mucho de minería ni el entorno, era capaz de ejecutar la gran mayoría de algoritmos y poder visualizarlos en pocos minutos.

Mientras que Python lo he visto más pesado y difícil de comprender.

El problema que imagino que habrá sobre Weka vs Python es que Weka estará bastante limitado a la hora de tratar con datos y querer usar los algoritmos que se usen en la vida real.

De ahí que los data science utilicen Python para resolver sus problemas en vez de Weka que será más un juguete para tratar con datos de andar por casa.