TOULOUSE LAUTREC

APRENDIZAJE AUTOMATICO CON PYTHON

ARBOL DE DECISION

Ing. Alexander Valdez

Curso 2290, Clases Lunes y Miercoles 20:00-22:30pm

Tercera Clase

Arbol de Decisión: ¿Quién llegará al número uno en Billboard 100?

In [2]:
```
# Imports necesarios
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
%matplotlib inline
#plt.rcParams['figure.figsize'] = (16, 9)
#plt.style.use('ggplot')
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from IPython.display import Image as PImage
from subprocess import check_call
from PIL import Image, ImageDraw, ImageFont
```

Cargamos los datos de entrada

In [3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
t("/content/drive", force_remount=True).
```

In [4]:
```
artists_billboard = pd.read_csv(r"/content/drive/MyDrive/DATASET_TOULOUSE_C3/artists_bil
```

In [5]:
```
artists_billboard.shape
```

Out[5]:
```
(635, 11)
```

In [ ]:
```
artists_billboard.head()
```

Out[ ]:

	id	title	artist	mood	tempo	genre	artist_type	chart_date	durationSeg	top	a
0	0	Small Town Throwdown	BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...	Brooding	Medium Tempo	Traditional	Male	20140628	191.0	0	
1	1	Bang Bang	JESSIE J, ARIANA GRANDE & NICKI MINAJ	Energizing	Medium Tempo	Pop	Female	20140816	368.0	0	
2	2	Timber	PITBULL featuring KE$HA	Excited	Medium Tempo	Urban	Mixed	20140118	223.0	1	
3	3	Sweater Weather	THE NEIGHBOURHOOD	Brooding	Medium Tempo	Alternative & Punk	Male	20140104	206.0	0	
4	4	Automatic	MIRANDA LAMBERT	Yearning	Medium Tempo	Traditional	Female	20140301	232.0	0	

In [ ]:
```
```

¿Cuántos alcanzaron el número 1?

In [6]:
```
pip install -U seaborn
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.
2)
Collecting seaborn
  Downloading seaborn-0.13.0-py3-none-any.whl (294 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 294.6/294.6 kB
6.6 MB/s eta 0:00:00
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/python3.10/dist-pa
ckages (from seaborn) (1.23.5)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-packages (f
rom seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.3 in /usr/local/lib/python3.10/dist
-packages (from seaborn) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib!=3.6.1,>=3.3->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages
 (from matplotlib!=3.6.1,>=3.3->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib!=3.6.1,>=3.3->seaborn) (4.44.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib!=3.6.1,>=3.3->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-package
s (from matplotlib!=3.6.1,>=3.3->seaborn) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages
 (from matplotlib!=3.6.1,>=3.3->seaborn) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib!=3.6.1,>=3.3->seaborn) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-pa
ckages (from matplotlib!=3.6.1,>=3.3->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages
 (from pandas>=1.2->seaborn) (2023.3.post1)
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
 python-dateutil>=2.7->matplotlib!=3.6.1,>=3.3->seaborn) (1.16.0)
Installing collected packages: seaborn
  Attempting uninstall: seaborn
    Found existing installation: seaborn 0.12.2
    Uninstalling seaborn-0.12.2:
      Successfully uninstalled seaborn-0.12.2
ERROR: pip's dependency resolver does not currently take into account all the packages t
hat are installed. This behaviour is the source of the following dependency conflicts.
lida 0.0.10 requires fastapi, which is not installed.
lida 0.0.10 requires kaleido, which is not installed.
lida 0.0.10 requires python-multipart, which is not installed.
lida 0.0.10 requires uvicorn, which is not installed.
Successfully installed seaborn-0.13.0
```

In [7]:
```
artists_billboard.groupby('top').size()
```

Out[7]:
```
top
0    494
1    141
dtype: int64
```

In [34]:
```
sb.factorplot('top',data=artists_billboard,kind="count")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-34-db8a178ee4fd> in <cell line: 1>()
----> 1 sb.factorplot('top',data=artists_billboard,kind="count")

AttributeError: module 'seaborn' has no attribute 'factorplot'
```

Visualicemos los Atributos de entrada

In [ ]:
```
sb.factorplot('artist_type',data=artists_billboard,kind="count")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-11-37dbac29b9be> in <cell line: 1>()
----> 1 sb.factorplot('artist_type',data=artists_billboard,kind="count")

AttributeError: module 'seaborn' has no attribute 'factorplot'
```

In [ ]:
```
sb.factorplot('top',data=artists_billboard,hue='artist_type',kind="count")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-12-34c99a1d6fda> in <cell line: 1>()
----> 1 sb.factorplot('top',data=artists_billboard,hue='artist_type',kind="count")

AttributeError: module 'seaborn' has no attribute 'factorplot'
```

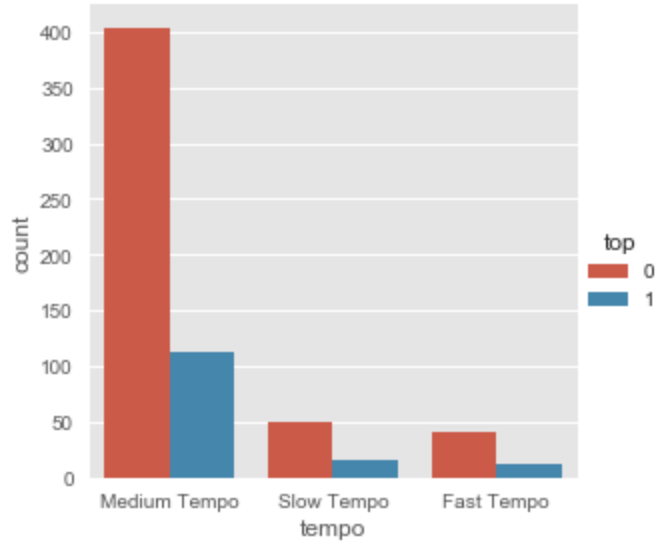
In [ ]:
```
sb.factorplot('mood',data=artists_billboard,kind="count", aspect=3)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-13-69cb70aee6cf> in <cell line: 1>()
----> 1 sb.factorplot('mood',data=artists_billboard,kind="count", aspect=3)

AttributeError: module 'seaborn' has no attribute 'factorplot'
```

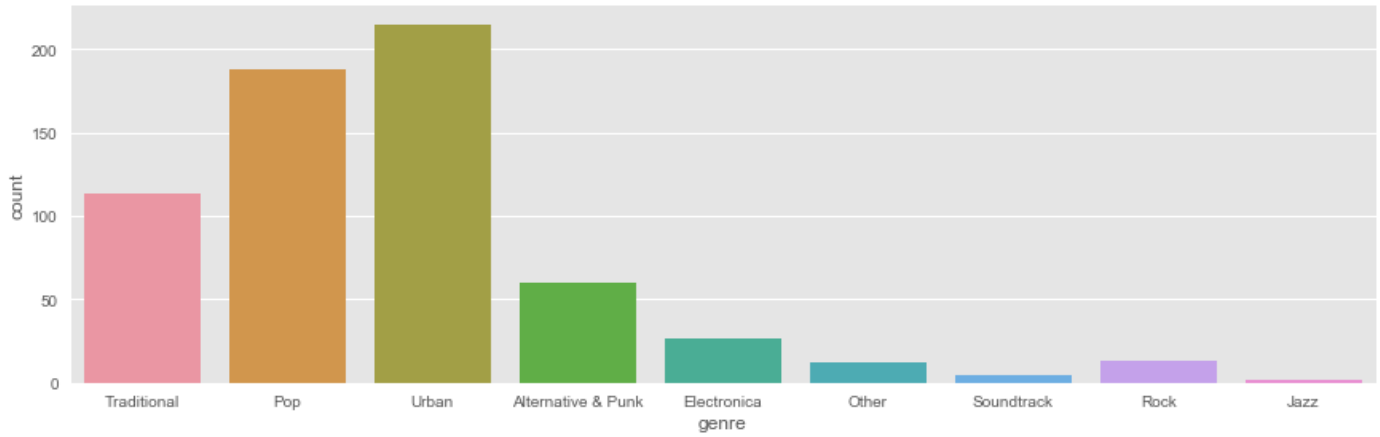
In [ ]:
```
sb.factorplot('tempo',data=artists_billboard,hue='top',kind="count")
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x118e97950>
```



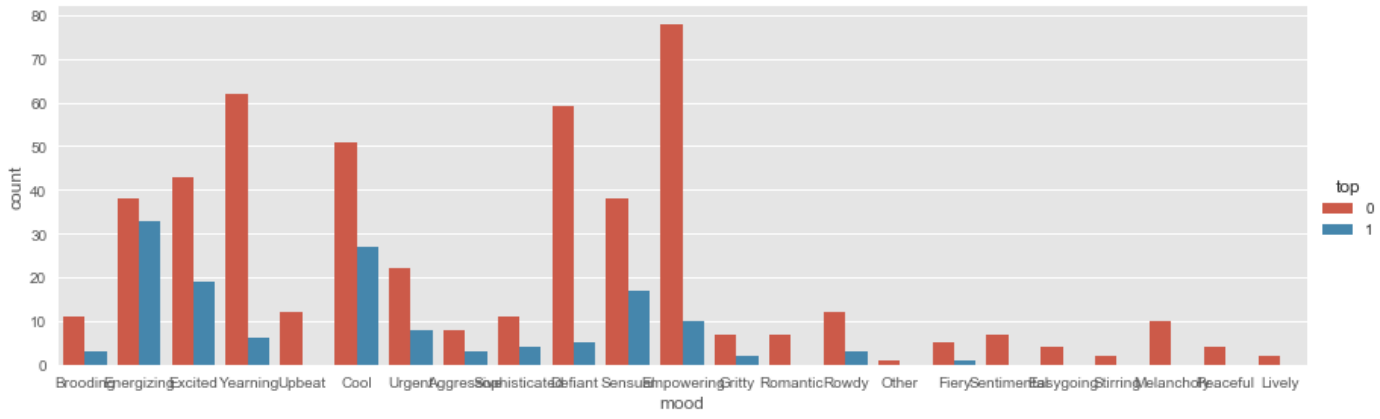
In [ ]:
```
sb.factorplot('genre',data=artists_billboard,kind="count", aspect=3)
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x1198d4e90>
```



In [ ]:
```
sb.factorplot('mood',data=artists_billboard,hue='top',kind="count", aspect=3)
```

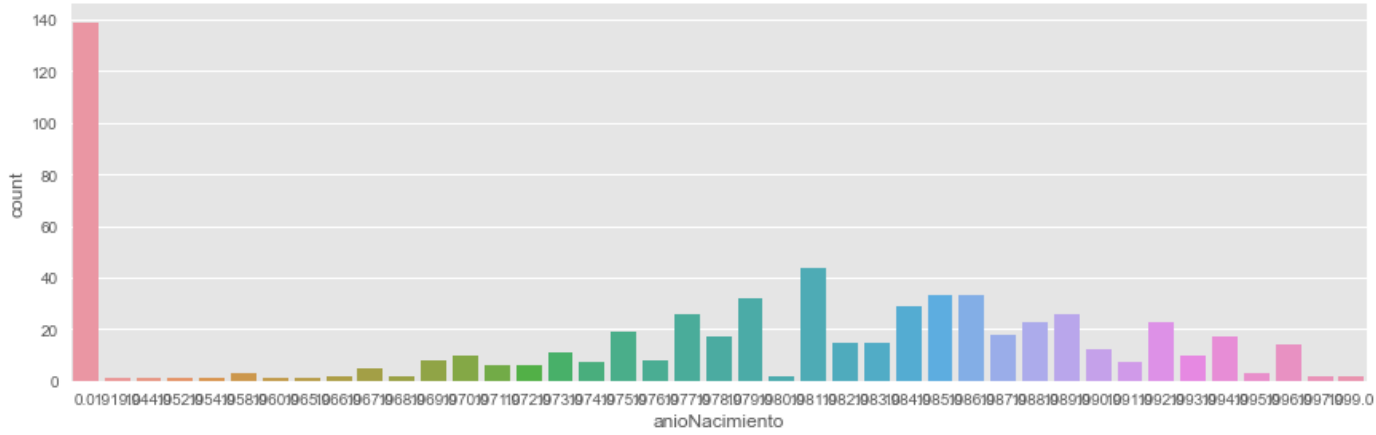
Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x119849ad0>
```



Visualizamos los años de nacimiento de los artistas

In [ ]:
```
sb.factorplot('anioNacimiento',data=artists_billboard,kind="count", aspect=3)
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x11952c150>
```



In [8]:
```
#artists_billboard[['anioNacimiento', 'top']].groupby(['anioNacimiento'], as_index=False
nacimientosPorAnio = artists_billboard['anioNacimiento']
len(nacimientosPorAnio[nacimientosPorAnio<=0])
```

Out[8]:
```
139
```

Notamos que tenemos 139 registros de canciones de las que desconocemos el año de nacimiento del artista. Deberemos tratar estos datos para poder utilizar el árbol.

Comparemos los Top y los No-top

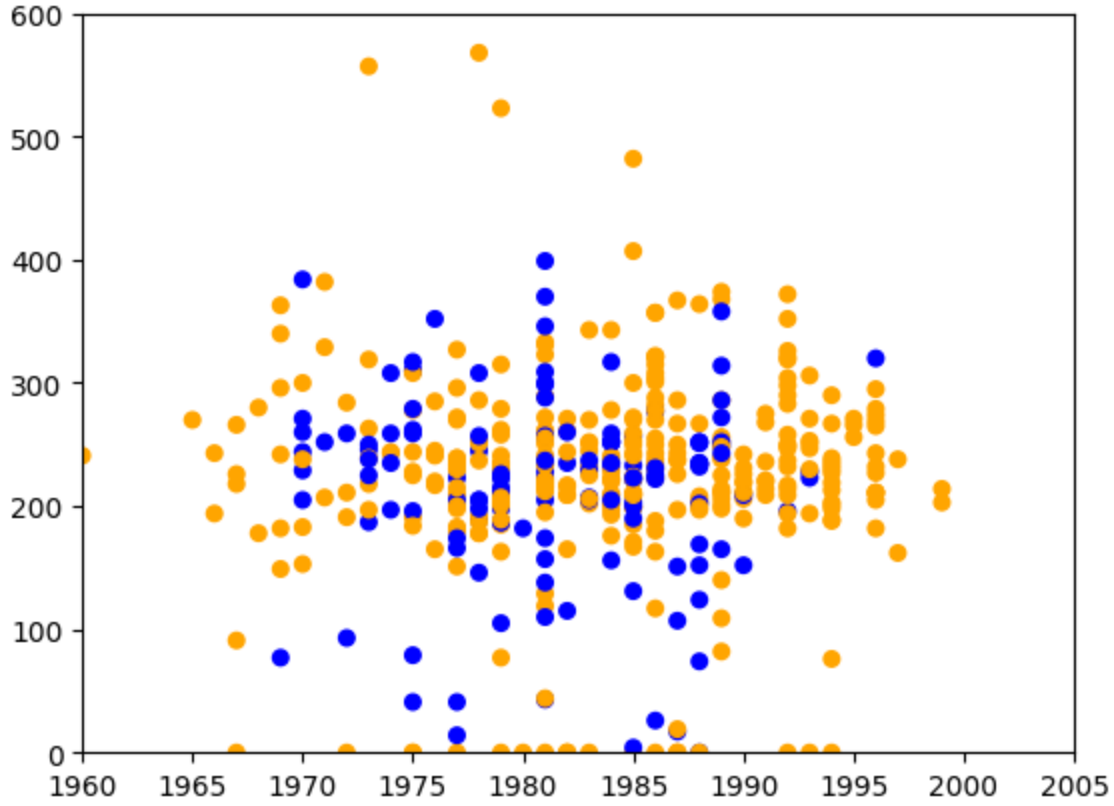
Buscamos si hay alguna relación evidente entre Año y duración de Canción

In [9]:
```
colores=['orange','blue']
tamanios=[60,40]

f1 = artists_billboard['anioNacimiento'].values
f2 = artists_billboard['durationSeg'].values

asignar=[]
for index, row in artists_billboard.iterrows():
    asignar.append(colores[row['top']])

plt.scatter(f1, f2, c=asignar, s=30)
plt.axis([1960,2005,0,600])
plt.show()
```

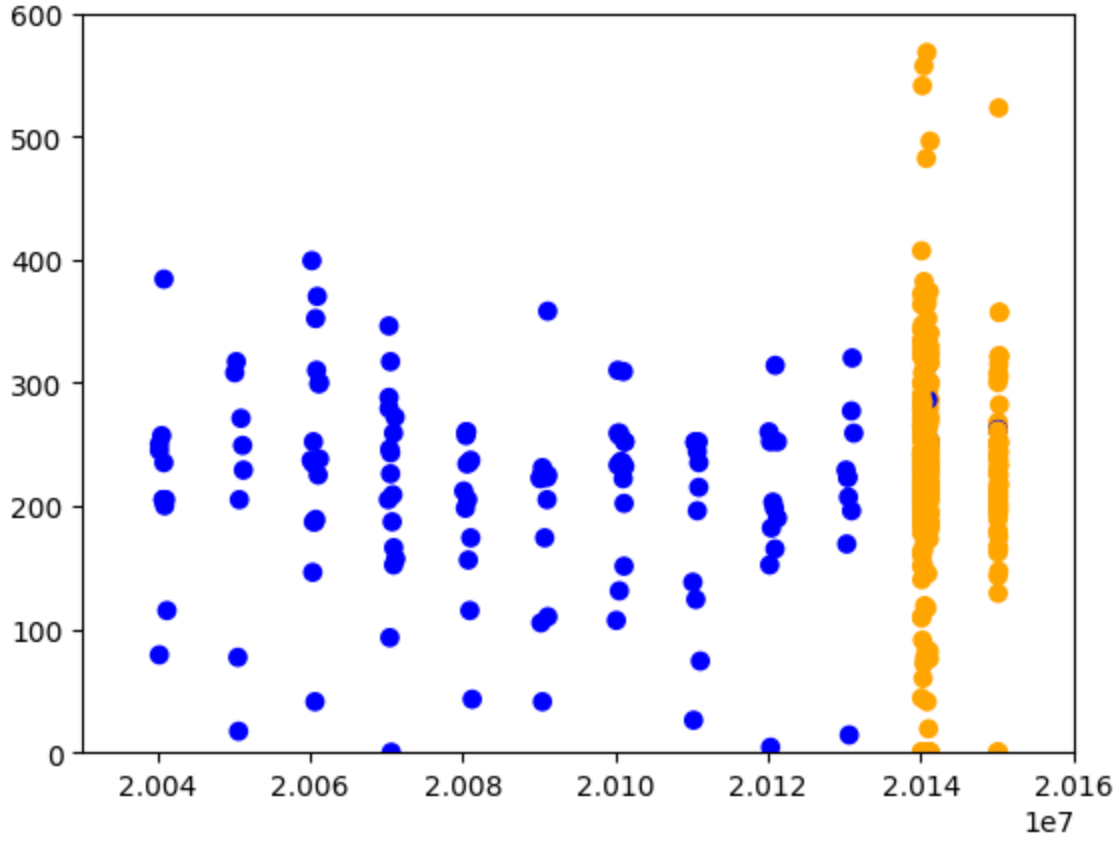


No perece haber ningún patron a la vista, están bastante mezclados los top de los no-top.

veamos en que años tenemos Top

In [15]:
```
f1 = artists_billboard['chart_date'].values
f2 = artists_billboard['durationSeg'].values
#print(f1)
#print(f2)
asignar=[]
asignar2=[]
for index, row in artists_billboard.iterrows():
    asignar.append(colores[row['top']])
    asignar2.append(tamanios[row['top']])

plt.scatter(f1.astype(float), f2, c=asignar)
plt.axis([20030101,20160101,0,600])
plt.show()
```



Esto es porque inicialmente tomé información de 2014 y 2015 y había casi 500 no-top contra sólo 11 registros top. Entonces tomé a los artistas que alcanzaron el top entre 2004 y 2013 para sumar casos positivos y equilibrar un poco nuestros datos de entrada. Así y todo, sigue estando desbalanceado.

Arreglar las Edades de Artistas

In [16]:
```
def edad_fix(anio):
    if anio==0:
        return None
    return anio

artists_billboard['anioNacimiento']=artists_billboard.apply(lambda x: edad_fix(x['anioNa
```

In [17]:
```
def calcula_edad(anio,cuando):
    cad = str(cuando)
    momento = cad[:4]
    if anio==0.0:
        return None
    return int(momento) - anio

artists_billboard['edad_en_billboard']=artists_billboard.apply(lambda x: calcula_edad(x[
```

In [18]:
```
artists_billboard.head()
```

Out[18]:

	id	title	artist	mood	tempo	genre	artist_type	chart_date	durationSeg	top	a
0	0	Small Town Throwdown	BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...	Brooding	Medium Tempo	Traditional	Male	20140628	191.0	0	
1	1	Bang Bang	JESSIE J, ARIANA GRANDE & NICKI MINAJ	Energizing	Medium Tempo	Pop	Female	20140816	368.0	0	

2	2	Timber	PITBULL featuring KE$HA	Excited	Medium Tempo	Urban	Mixed	20140118	223.0	1
3	3	Sweater Weather	THE NEIGHBOURHOOD	Brooding	Medium Tempo	Alternative & Punk	Male	20140104	206.0	0
4	4	Automatic	MIRANDA LAMBERT	Yearning	Medium Tempo	Traditional	Female	20140301	232.0	0

Calculamos promedio de edad y asignamos a los registros Nulos

In [19]:
```
age_avg = artists_billboard['edad_en_billboard'].mean()
age_std = artists_billboard['edad_en_billboard'].std()
age_null_count = artists_billboard['edad_en_billboard'].isnull().sum()
age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std, size=age_

conValoresNulos = np.isnan(artists_billboard['edad_en_billboard'])

artists_billboard.loc[np.isnan(artists_billboard['edad_en_billboard']), 'edad_en_billboa
artists_billboard['edad_en_billboard'] = artists_billboard['edad_en_billboard'].astype(i
print("Edad Promedio: " + str(age_avg))
print("Desvió Std Edad: " + str(age_std))
print("Intervalo para asignar edad aleatoria: " + str(int(age_avg - age_std)) + " a " +
```

```
Edad Promedio: 30.10282258064516
Desvió Std Edad: 8.40078832861513
Intervalo para asignar edad aleatoria: 21 a 38
```

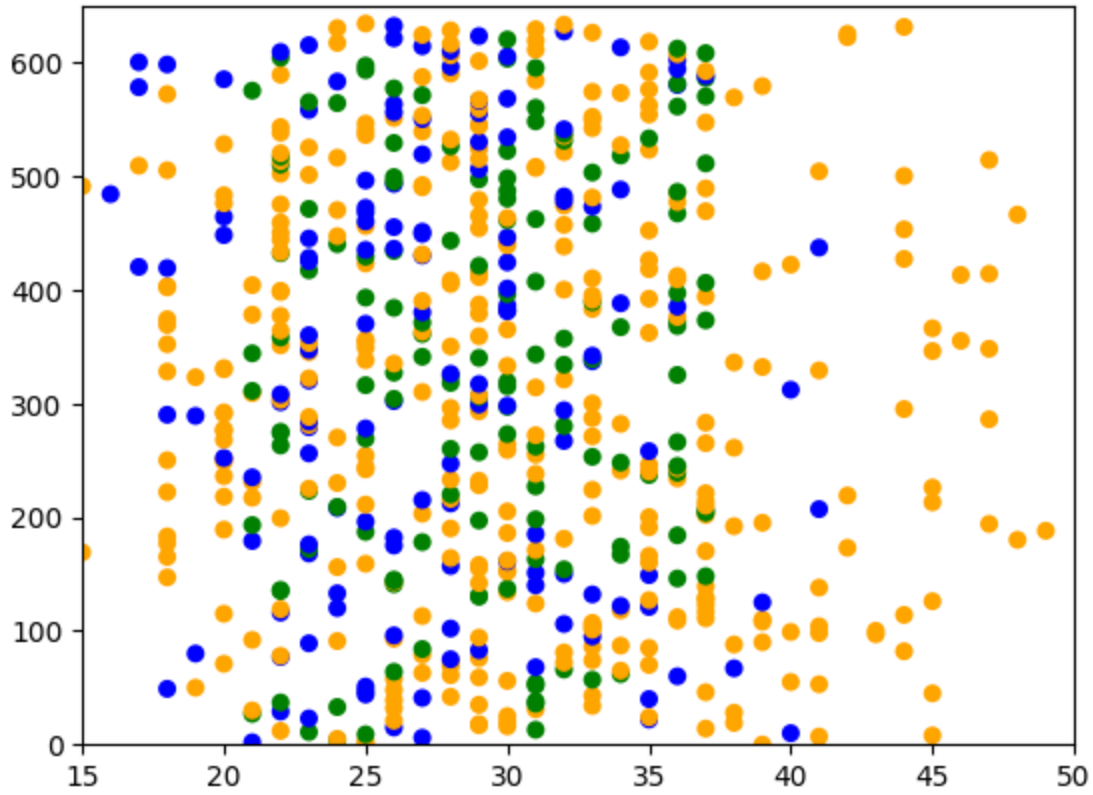
Visualizamos las edades que agregamos

In [20]:
```
f1 = artists_billboard['edad_en_billboard'].values
f2 = artists_billboard.index

colores = ['orange','blue','green']

asignar=[]
for index, row in artists_billboard.iterrows():
    if (conValoresNulos[index]):
        asignar.append(colores[2]) # verde
    else:
        asignar.append(colores[row['top']])

plt.scatter(f1, f2, c=asignar, s=30)
plt.axis([15,50,0,650])
plt.show()
```



Mapeo de Atributos

Realizaremos un mapeo de los atributos de entrada para poder transformarlos a categorias que podamos utilzar en nuestro árbol de decisión

In [21]:
```
separador = "### ### ###"
grouped11 = artists_billboard.groupby('mood').size()#.sum().reset_index()
neworder11 = grouped11.sort_values(ascending=False)
print(neworder11)
print(separador)
print("Tempos de Canción: " + str(artists_billboard['tempo'].unique()))
print(separador)
print("Tipos de Artista: " + str(artists_billboard['artist_type'].unique()))
print(separador)
grouped11 = artists_billboard.groupby('genre').size()#.sum().reset_index()
neworder11 = grouped11.sort_values(ascending=False)
print(neworder11)
```

```
mood
Empowering       88
Cool             78
Energizing       71
Yearning         68
Defiant          64
Excited          62
Sensual          55
Urgent           30
Rowdy            15
Sophisticated    15
Brooding         14
Upbeat           12
Aggressive       11
Melancholy       10
Gritty            9
Romantic          7
Sentimental       7
```

```
Fiery              6
Peaceful           4
Easygoing          4
Lively             2
Stirring           2
Other              1
dtype: int64
### ### ###
Tempos de Canción: ['Medium Tempo' 'Slow Tempo' 'Fast Tempo']
### ### ###
Tipos de Artista: ['Male' 'Female' 'Mixed']
### ### ###
genre
Urban                 215
Pop                   188
Traditional           113
Alternative & Punk     60
Electronica            27
Rock                   13
Other                  12
Soundtrack              5
Jazz                    2
dtype: int64
```

In [22]:

```
# Mood Mapping
artists_billboard['moodEncoded'] = artists_billboard['mood'].map( {'Energizing': 6,
                                        'Empowering': 6,
                                        'Cool': 5,
                                        'Yearning': 4, # anhelo, deseo, ansia
                                        'Excited': 5, #emocionado
                                        'Defiant': 3,
                                        'Sensual': 2,
                                        'Gritty': 3, #coraje
                                        'Sophisticated': 4,
                                        'Aggressive': 4, # provocativo
                                        'Fiery': 4, #caracter fuerte
                                        'Urgent': 3,
                                        'Rowdy': 4, #ruidoso alboroto
                                        'Sentimental': 4,
                                        'Easygoing': 1, # sencillo
                                        'Melancholy': 4,
                                        'Romantic': 2,
                                        'Peaceful': 1,
                                        'Brooding': 4, # melancolico
                                        'Upbeat': 5, #optimista alegre
                                        'Stirring': 5, #emocionante
                                        'Lively': 5, #animado
                                        'Other': 0,'':0} ).astype(int)
# Tempo Mapping
artists_billboard['tempoEncoded'] = artists_billboard['tempo'].map( {'Fast Tempo': 0, 'M
# Genre Mapping
artists_billboard['genreEncoded'] = artists_billboard['genre'].map( {'Urban': 4,
                                        'Pop': 3,
                                        'Traditional': 2,
                                        'Alternative & Punk': 1,
                                       'Electronica': 1,
                                        'Rock': 1,
                                        'Soundtrack': 0,
                                        'Jazz': 0,
                                        'Other':0,'':0}
                                      ).astype(int)
# artist_type Mapping
artists_billboard['artist_typeEncoded'] = artists_billboard['artist_type'].map( {'Female
```

```
# Mapping edad en la que llegaron al billboard
artists_billboard.loc[ artists_billboard['edad_en_billboard'] <= 21, 'edadEncoded']
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 21) & (artists_billboard
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 26) & (artists_billboard
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 30) & (artists_billboard
artists_billboard.loc[ artists_billboard['edad_en_billboard'] > 40, 'edadEncoded'] = 4

# Mapping Song Duration
artists_billboard.loc[ artists_billboard['durationSeg'] <= 150, 'durationEncoded']
artists_billboard.loc[(artists_billboard['durationSeg'] > 150) & (artists_billboard['dur
artists_billboard.loc[(artists_billboard['durationSeg'] > 180) & (artists_billboard['dur
artists_billboard.loc[(artists_billboard['durationSeg'] > 210) & (artists_billboard['dur
artists_billboard.loc[(artists_billboard['durationSeg'] > 240) & (artists_billboard['dur
artists_billboard.loc[(artists_billboard['durationSeg'] > 270) & (artists_billboard['dur
artists_billboard.loc[ artists_billboard['durationSeg'] > 300, 'durationEncoded'] = 6
```

In [24]:
```
drop_elements = ['id','title','artist','mood','tempo','genre','artist_type','chart_date'
artists_encoded = artists_billboard.drop(drop_elements, axis = 1)
```

Analizamos nuestros datos de Entrada Categóricos

In [25]:
```
artists_encoded.head()
```

Out[25]:

	top	moodEncoded	tempoEncoded	genreEncoded	artist_typeEncoded	edadEncoded	durationEncoded
0	0	4	2	2	3	3.0	2.0
1	0	6	2	3	2	1.0	6.0
2	1	5	2	4	1	0.0	3.0
3	0	4	2	1	3	1.0	2.0
4	0	4	2	2	2	1.0	3.0

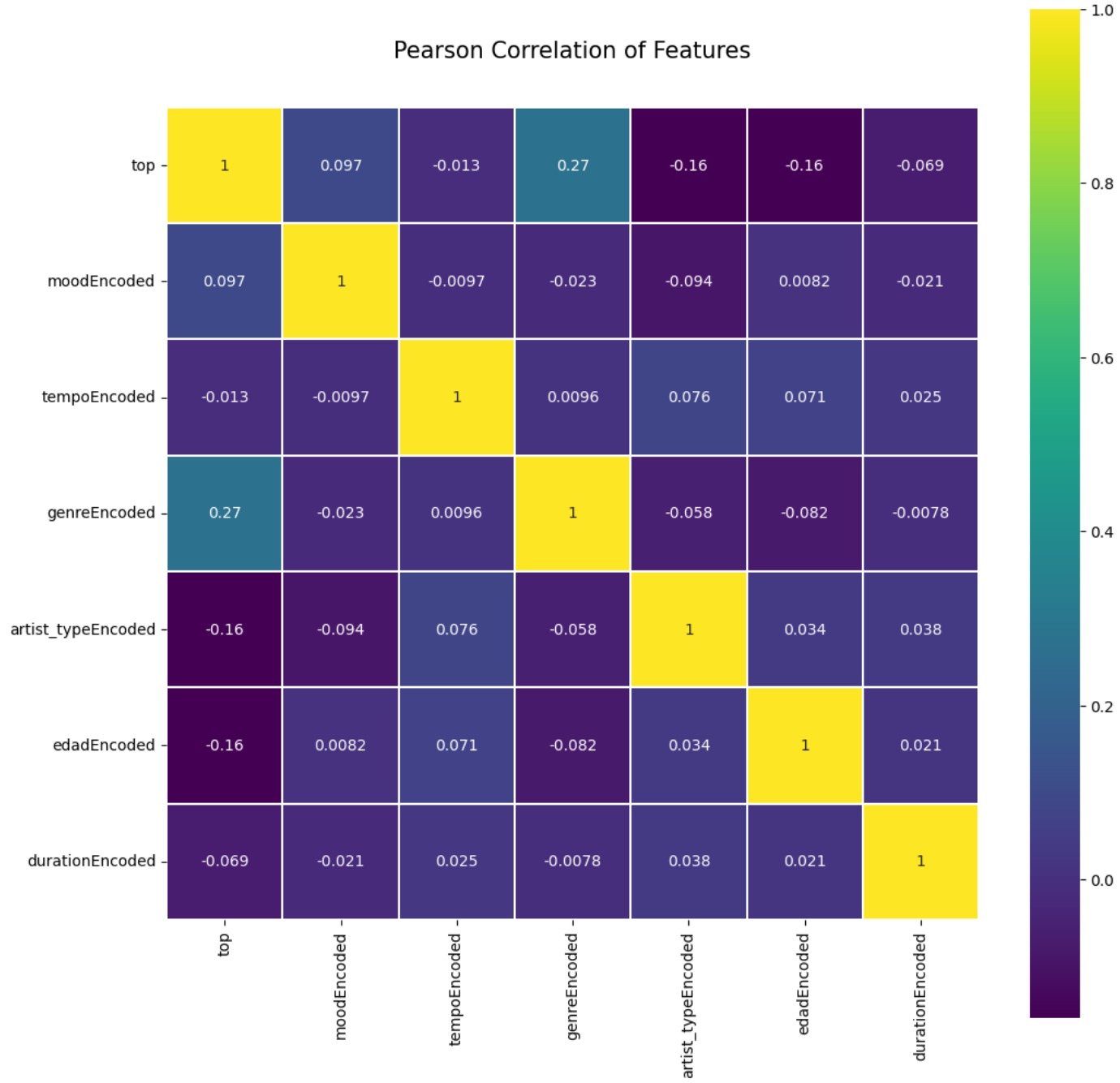
In [26]:
```
artists_encoded.describe()
```

Out[26]:

	top	moodEncoded	tempoEncoded	genreEncoded	artist_typeEncoded	edadEncoded	durationEncoc
count	635.000000	635.000000	635.000000	635.000000	635.000000	635.000000	635.000
mean	0.222047	4.344882	1.730709	2.755906	2.459843	2.040945	3.179
std	0.415950	1.350003	0.603553	1.165463	0.740583	1.137915	1.775
min	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000
25%	0.000000	3.000000	2.000000	2.000000	2.000000	1.000000	2.000
50%	0.000000	4.000000	2.000000	3.000000	3.000000	2.000000	3.000
75%	0.000000	5.500000	2.000000	4.000000	3.000000	3.000000	4.000
max	1.000000	6.000000	2.000000	4.000000	3.000000	4.000000	6.000

In [27]:
```
colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sb.heatmap(artists_encoded.astype(float).corr(),linewidths=0.1,vmax=1.0, square=True, cm
```

Out[27]:
```
<Axes: title={'center': 'Pearson Correlation of Features'}>
```



In [ ]:
```
artists_encoded[['moodEncoded', 'top']].groupby(['moodEncoded'], as_index=False).agg(['m
```

Out[ ]:

	top		
	mean	count	sum
moodEncoded			
0	0.000000	1	0
1	0.000000	8	0
2	0.274194	62	17
3	0.145631	103	15
4	0.136986	146	20
5	0.294872	156	46
6	0.270440	159	43

In [ ]:
```
artists_encoded[['artist_typeEncoded', 'top']].groupby(['artist_typeEncoded'], as_index=
```

Out[ ]:

	top		
	mean	count	sum
artist_typeEncoded			
1	0.305263	95	29
2	0.320261	153	49
3	0.162791	387	63

In [ ]:
```
artists_encoded[['genreEncoded', 'top']].groupby(['genreEncoded'], as_index=False).agg([
```

Out[ ]:

	top		
	mean	count	sum
genreEncoded			
0	0.105263	19	2
1	0.070000	100	7
2	0.008850	113	1
3	0.319149	188	60
4	0.330233	215	71

In [ ]:
```
artists_encoded[['tempoEncoded', 'top']].groupby(['tempoEncoded'], as_index=False).agg([
```

Out[ ]:

	top		
	mean	count	sum
tempoEncoded			
0	0.226415	53	12
1	0.246154	65	16
2	0.218569	517	113

In [ ]:
```
artists_encoded[['durationEncoded', 'top']].groupby(['durationEncoded'], as_index=False)
```

Out[ ]:

	top		
	mean	count	sum
durationEncoded			
0.0	0.295775	71	21
1.0	0.333333	30	10
2.0	0.212963	108	23
3.0	0.202381	168	34
4.0	0.232143	112	26
5.0	0.145455	55	8
6.0	0.208791	91	19

In [ ]:
```
artists_encoded[['edadEncoded', 'top']].groupby(['edadEncoded'], as_index=False).agg(['m
```

Out[ ]:

	top		
	mean	count	sum
edadEncoded			
0.0	0.257576	66	17
1.0	0.300613	163	49
2.0	0.260563	142	37
3.0	0.165899	217	36
4.0	0.042553	47	2

Buscamos nuestro Arbol de Decisión

In [29]:
```
cv = KFold(n_splits=10) # Numero deseado de "folds" que haremos
accuracies = list()
max_attributes = len(list(artists_encoded))
depth_range = range(1, max_attributes + 1)

# Testearemos la profundidad de 1 a cantidad de atributos +1
for depth in depth_range:
    fold_accuracy = []
    tree_model = tree.DecisionTreeClassifier(criterion='entropy',
                                             min_samples_split=20,
                                             min_samples_leaf=5,
                                             max_depth = depth,
                                             class_weight={1:3.5})
    for train_fold, valid_fold in cv.split(artists_encoded):
        f_train = artists_encoded.loc[train_fold]
        f_valid = artists_encoded.loc[valid_fold]

        model = tree_model.fit(X = f_train.drop(['top'], axis=1),
                               y = f_train["top"])
        valid_acc = model.score(X = f_valid.drop(['top'], axis=1),
                                y = f_valid["top"]) # calculamos la precision con el seg
        fold_accuracy.append(valid_acc)

    avg = sum(fold_accuracy)/len(fold_accuracy)
    accuracies.append(avg)

# Mostramos los resultados obtenidos
df = pd.DataFrame({"Max Depth": depth_range, "Average Accuracy": accuracies})
df = df[["Max Depth", "Average Accuracy"]]
print(df.to_string(index=False))
```

```
 Max Depth  Average Accuracy
         1          0.556101
         2          0.556126
         3          0.567163
         4          0.648884
         5          0.612798
         6          0.628373
         7          0.626761
```

Creamos el Arbol de Decisión

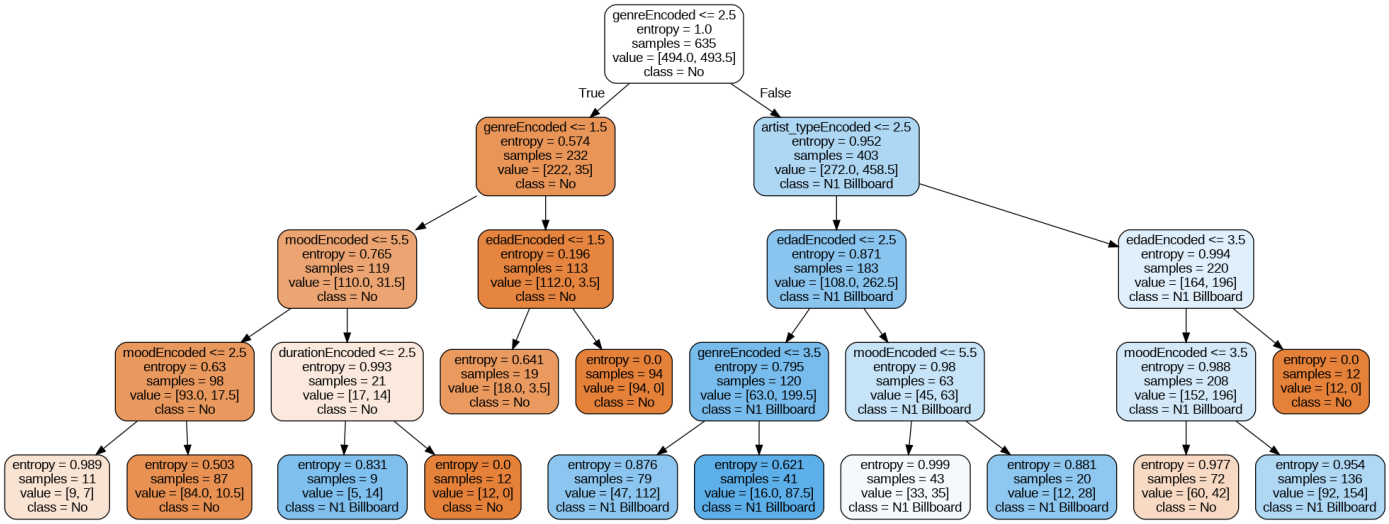
In [30]:
```
# Crear arrays de entrenamiento y las etiquetas que indican si llegó a top o no
y_train = artists_encoded['top']
x_train = artists_encoded.drop(['top'], axis=1).values

# Crear Arbol de decision con profundidad = 4
decision_tree = tree.DecisionTreeClassifier(criterion='entropy',
                                            min_samples_split=20,
                                            min_samples_leaf=5,
                                            max_depth = 4,
                                            class_weight={1:3.5})
decision_tree.fit(x_train, y_train)

# exportar el modelo a archivo .dot
with open(r"tree1.dot", 'w') as f:
     f = tree.export_graphviz(decision_tree,
                              out_file=f,
                              max_depth = 7,
                              impurity = True,
                              feature_names = list(artists_encoded.drop(['top'], axis=1)
                              class_names = ['No', 'N1 Billboard'],
                              rounded = True,
                              filled= True )

# Convertir el archivo .dot a png para poder visualizarlo
check_call(['dot','-Tpng',r'tree1.dot','-o',r'tree1.png'])
PImage("tree1.png")
```

Out[30]:



Precisión del árbol

In [31]:
```
acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
print(acc_decision_tree)
```

```
64.88
```

Predicción del árbol de decisión

In [32]:
```
#predecir artista CAMILA CABELLO featuring YOUNG THUG
# con su canción Havana llego a numero 1 Billboard US en 2017

x_test = pd.DataFrame(columns=('top','moodEncoded', 'tempoEncoded', 'genreEncoded','arti
x_test.loc[0] = (1,5,2,4,1,0,3)
y_pred = decision_tree.predict(x_test.drop(['top'], axis = 1))
print("Prediccion: " + str(y_pred))
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
print("Probabilidad de Acierto: " + str(round(y_proba[0][y_pred][0]* 100, 2))+"%")
```

```
Prediccion: [1]
Probabilidad de Acierto: 84.54%
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature
names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature
names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(
```

In [33]:
```
#predecir artista Imagine Dragons
# con su canción Believer llego al puesto 42 Billboard US en 2017

x_test = pd.DataFrame(columns=('top','moodEncoded', 'tempoEncoded', 'genreEncoded','arti
x_test.loc[0] = (0,4,2,1,3,2,3)
y_pred = decision_tree.predict(x_test.drop(['top'], axis = 1))
print("Prediccion: " + str(y_pred))
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
print("Probabilidad de Acierto: " + str(round(y_proba[0][y_pred][0]* 100, 2))+"%")
```

```
Prediccion: [0]
Probabilidad de Acierto: 88.89%
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature
names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature
names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(
```

FINAL

In [ ]:
```
```