final_paper

November 8, 2020

1 Final Project

2 0.0 Configurar MLFlow

[1]:
```
import mlflow
import os

# you can set your tracking server URI programmatically:
mlflow.set_tracking_uri('https://mlflow-aie3.ai.spglobal.com/')
os.environ['MLFLOW_S3_ENDPOINT_URL'] = 'https://minio-aie3.ai.spglobal.com/'
os.environ['LOGNAME'] = 'oswaldo'
```

3 1.0 Importar datos de entrenamiento

[2]:
```
import pandas as pd
dataset = pd.read_csv("training.csv")
```

```
/Users/oswaldo_gomez/Library/Caches/pypoetry/virtualenvs/time-series-CSagJmyP-
py3.7/lib/python3.7/site-packages/ipykernel/ipkernel.py:287: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future.
Please pass the result to `transformed_cell` argument and any exception that
happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and
above.
  and should_run_async(code)
```

[3]:
```
dataset.dropna(inplace=True)
```

[4]:
```
#check the shape of data
dataset.shape
```

```
/Users/oswaldo_gomez/Library/Caches/pypoetry/virtualenvs/time-series-CSagJmyP-
py3.7/lib/python3.7/site-packages/ipykernel/ipkernel.py:287: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future.
Please pass the result to `transformed_cell` argument and any exception that
happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and
above.
  and should_run_async(code)
```

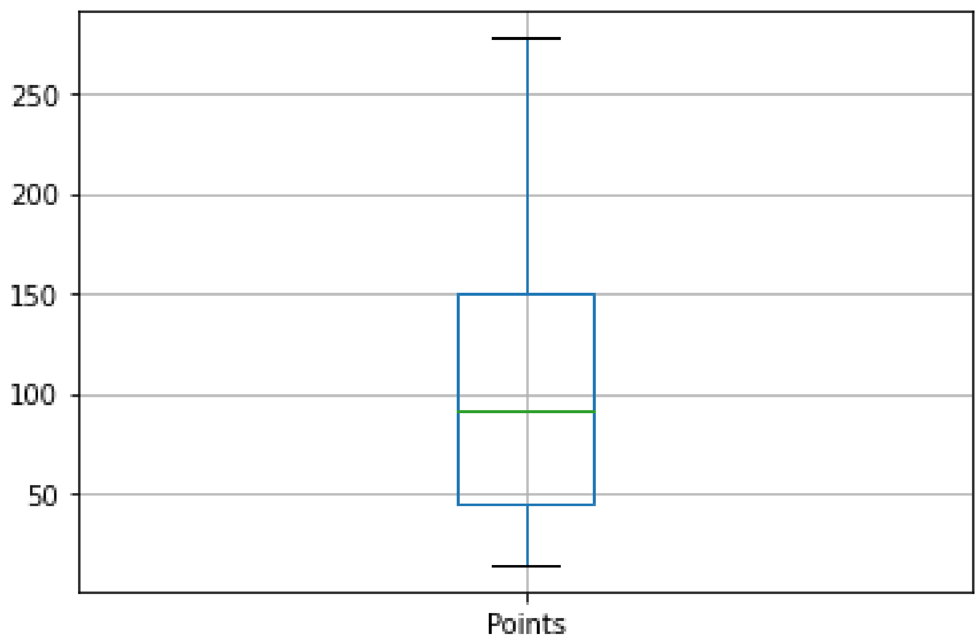
[4]: (85, 24)

[5]:
```
dataset['Points'].describe()
```

[5]:
```
count     85.000000
mean     102.352941
std       63.767753
min       14.000000
25%       45.000000
50%       92.000000
75%      150.000000
max      278.000000
Name: Points, dtype: float64
```

[6]:
```
dataset.boxplot(column=['Points'])
```

[6]: <AxesSubplot:>



3.1 Convertimos la columna de punto de numérica a categórica, con dos categorías. Bueno y Malo.

3.1.1 Malo se define entre el valor mínimo teórico y la media. Bueno es entre la media y el valor máximo observado

[7]:
```
bins=[0,92,278]
names=['Bad','Good']
dataset['Points_range']=pd.cut(dataset['Points'],bins,labels=names)
```

```
/Users/oswaldo_gomez/Library/Caches/pypoetry/virtualenvs/time-series-CSagJmyP-
py3.7/lib/python3.7/site-packages/ipykernel/ipkernel.py:287: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future.
Please pass the result to `transformed_cell` argument and any exception that
happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and
above.
  and should_run_async(code)
```

[8]:
```
dataset.sort_values(by='Points_range')
```

[8]:
```
     duration  key  loudness  mode    tempo  artist_hotttnesss  \
0   219.96263    7    -6.869     1  129.991               0.00
45  209.26694    1    -3.828     1  124.843               0.01
46  173.46613    6    -5.819     1  139.682               0.29
47  244.20000   10    -3.871     1   91.996               0.24
48  179.00000   11    -7.907     0   86.175               0.01
..        …  …       …   …       …                 …
61  179.13538    2    -3.978     1  167.893               0.15
62  182.95084    6    -6.376     1  121.963               0.20
64  181.74525   10    -6.991     1   85.915               0.04
67  186.74667    7   -18.336     1  181.599               0.67
49  235.17332   11    -4.950     0  124.461               0.71

    end_of_fade_in  start_of_fade_out  mode_confidence  key_confidence  …  \
0          0.00000          207.69089            0.755           0.954  …
45         0.58054          204.60263            0.790           0.617  …
46         0.00000          168.33305            0.678           0.742  …
47         0.40685          240.97668            0.430           0.470  …
48         0.45274          173.83038            0.718           0.659  …
..             …                …               …             …  …
61         0.00000          175.10748            0.613           0.433  …
62         3.54685          174.65470            0.510           0.594  …
64         2.86186          176.55873            0.526           0.451  …
67         0.75456          178.16672            0.818           1.000  …
49         0.28526          230.85860            0.641           0.858  …

    energy  speechiness  acousticness  instrumentalness  liveness  valence  \
0    0.870       0.0456      0.000592           0.82700    0.1620    0.175
```

```
45    0.809        0.0384      0.546000            0.00000     0.1230     0.423
46    0.529        0.0292      0.205000            0.00000     0.0927     0.386
47    0.833        0.0310      0.076500            0.00000     0.2390     0.519
48    0.442        0.0328      0.829000            0.00000     0.2460     0.130
..      …            …           …              …       …        …
61    0.846        0.0516      0.112000            0.00000     0.2190     0.358
62    0.692        0.0588      0.007530            0.00152     0.0301     0.574
64    0.280        0.0313      0.615000            0.00000     0.1880     0.259
67    0.258        0.0605      0.361000            0.00000     0.0349     0.934
49    0.909        0.0670      0.036500            0.00407     0.1100     0.544

    duration_ms  Points      Country  Points_range
0        219963    45.0      Belgium           Bad
45       209267    34.0        Italy           Bad
46       173466    35.0  Netherlands           Bad
47       244200    64.0    Australia           Bad
48       179000    36.0      Ireland           Bad
..          …        …           …          …
61       179135   202.0    Australia          Good
62       182951   174.0  Netherlands          Good
64       181745   239.0      Georgia          Good
67       186747   125.0        Spain          Good
49       235173   176.0      Armenia          Good

[85 rows x 25 columns]
```

4 2.0 Vamos a comenzar el experimento, en donde sólo utilizaremos las columnas no ignoradas (precedidas por un #, por ejemplo #year no es ignorada. O lo que es lo mismo, es considerada)

4.1 El objetivo es encontrar un modelo de aprendizaje automático que logre predecir la categoría de bueno/malo con base en año (categórico), bailable, energía y acustica de las canciones utilizando Spotify API get-audio-features y get-audio-analysis

[9]:
```python
from pycaret.classification import *
exp_mclf101 = setup(data = dataset,
                    target = 'Points_range',
                    session_id=123,
                    ignore_features=[#'duration',
                                     'key',
                                     #'loudness',
                                     'mode',
                                     #'tempo',
```

```python
                                    'artist_hotttnesss',
                                    'end_of_fade_in',
                                    'start_of_fade_out',
                                    'mode_confidence',
                                    'key_confidence',
                                    'time_signature',
                                    'time_signature_confidence',
                                    'year',
                                    'popularity',
                                    #'danceability',
                                    #'energy',
                                    #'speechiness',
                                    #'acousticness',
                                    'instrumentalness',
                                    'liveness',
                                    #'valence',
                                    'duration_ms',
                                    'Points',
                                    'Country',
                                    #'Points_range',
                                   ],
                    log_experiment=True,
                    experiment_name="Final_paper_final",
                    log_plots=True,
                    profile=True,
                    use_gpu=True)
```

```
HBox(children=(HTML(value='Summarize dataset'), FloatProgress(value=0.0, max=40.
↪0), HTML(value='')))
```

```
HBox(children=(HTML(value='Generate report structure'), FloatProgress(value=0.0,␣
↪max=1.0), HTML(value='')))
```

```
HBox(children=(HTML(value='Render HTML'), FloatProgress(value=0.0, max=1.0),␣
↪HTML(value='')))
```

```
<IPython.core.display.HTML object>
```

5 3.0 Comparando múltiples modelos

5.1 Vamos a ordenarlos de mayor a menor precisión

[10]:
```
best = compare_models(sort='Precision')
```

```
<pandas.io.formats.style.Styler at 0x143216240>
```

6 3.0 Crearemos 3 objetos modelo que presentaron las métricas más altas de Precisión

6.0.1 Gradient Boosting Classifier

[11]:
```
gbc = create_model('gbc')
```

```
<pandas.io.formats.style.Styler at 0x16e92df60>
```

[12]:
```
#trained model object is stored in the variable 'dt'.
print(gbc)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=123, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

6.0.2 Random Forest

[13]:
```
rf = create_model('rf')
```

```
<pandas.io.formats.style.Styler at 0x16ec3ba20>
```

6.0.3 Ada Boost Classifier

[14]:
```
ada = create_model('ada')
```

```
<pandas.io.formats.style.Styler at 0x1702e5240>
```

7 4.0 Vamos a afinar los hiperparámetros buscando maximizar la precisión

7.0.1 Gradient Boosting Classifier

[15]:
```
tuned_gbc = tune_model(gbc,optimize = 'Precision',choose_better=True)
```

```
<pandas.io.formats.style.Styler at 0x12141ada0>
```

[16]:
```
#tuned model object is stored in the variable 'tuned_dt'.
print(tuned_gbc)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.408, loss='deviance', max_depth=10,
                           max_features=1.0, max_leaf_nodes=None,
                           min_impurity_decrease=0.0005,
                           min_impurity_split=None, min_samples_leaf=3,
                           min_samples_split=10, min_weight_fraction_leaf=0.0,
                           n_estimators=170, n_iter_no_change=None,
                           presort='deprecated', random_state=123,
                           subsample=0.95, tol=0.0001, validation_fraction=0.1,
                           verbose=0, warm_start=False)
```

7.0.2 Random Forest

[17]:
```
import numpy as np
tuned_rf = tune_model(rf,optimize =␣
↪'Precision',choose_better=True)#,n_iter=1000)
```

```
<pandas.io.formats.style.Styler at 0x16b8b5828>
```

7.0.3 Ada Boost Classifier

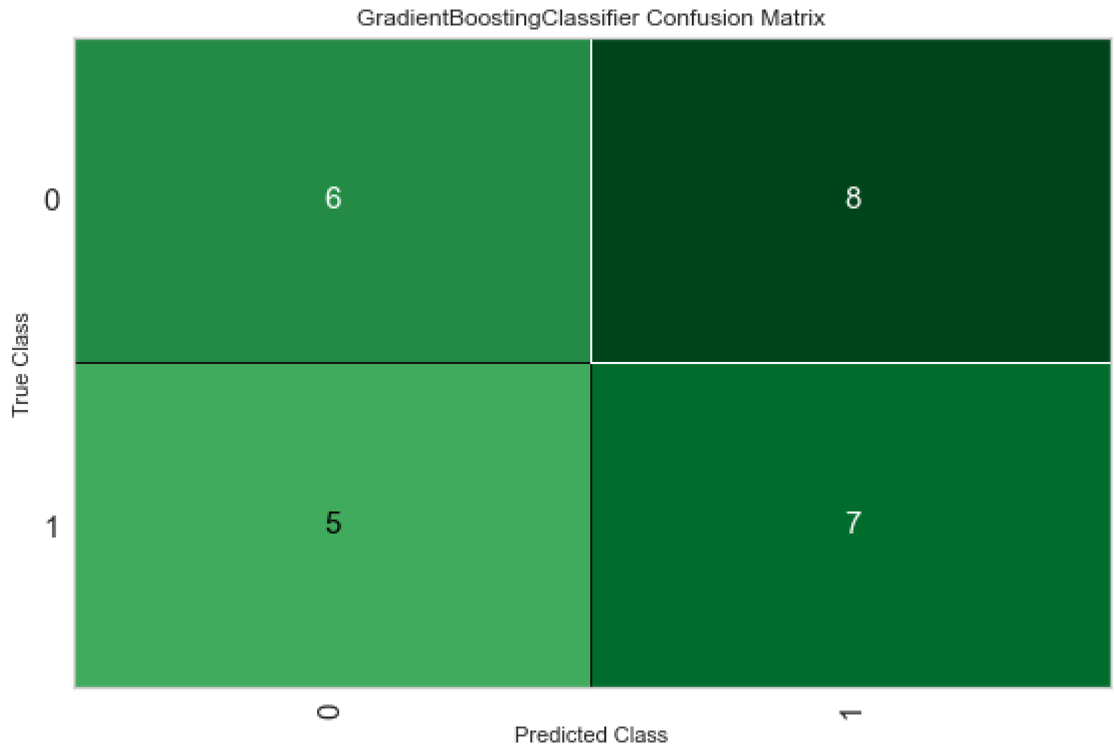
[18]:
```
tuned_ada = tune_model(ada,optimize = 'Precision',choose_better=True)
```

```
<pandas.io.formats.style.Styler at 0x16e2042b0>
```

8 5.0 Gráficas de los modelos afinados

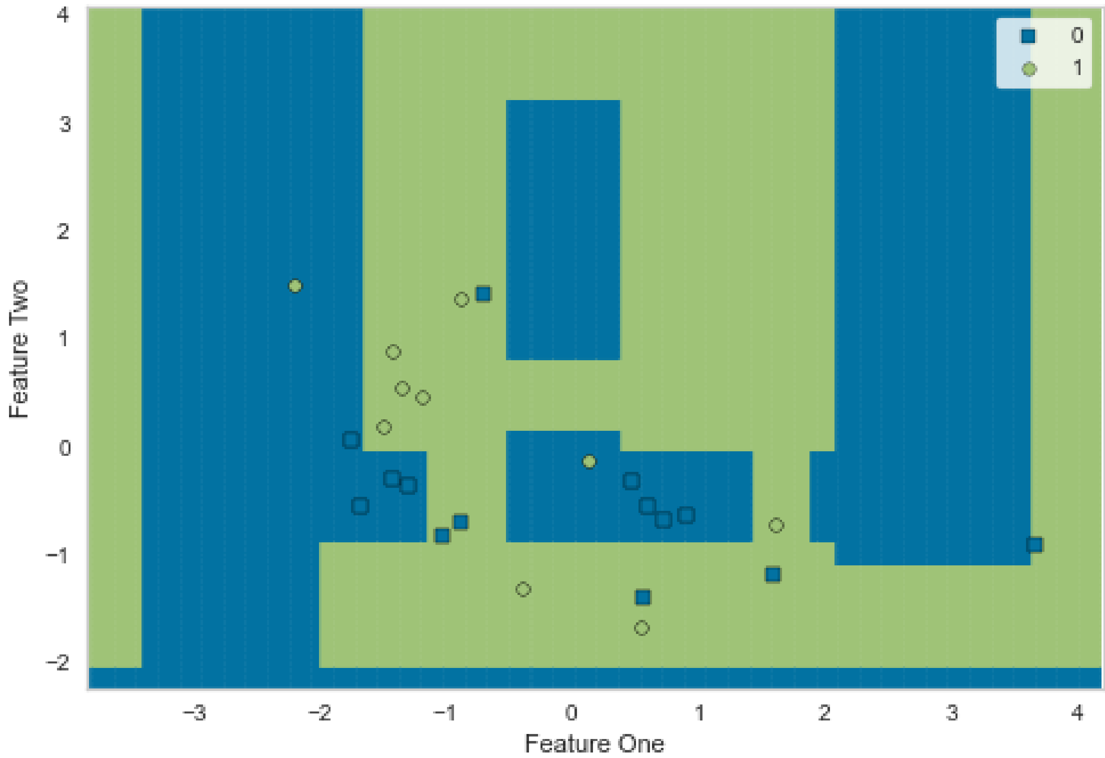
8.0.1 Matriz de confusión

[19]:
```
plot_model(tuned_gbc, plot = 'confusion_matrix')
```



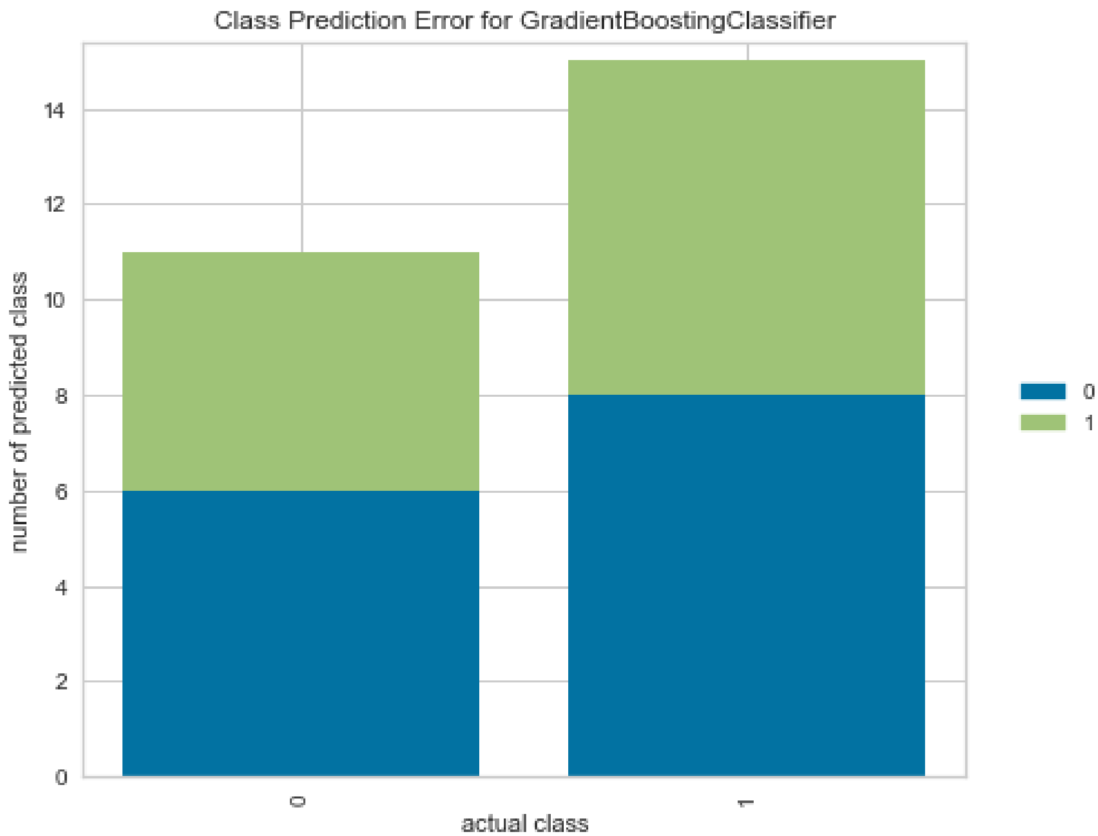
8.0.2 Mapa de decisión

[21]:
```
plot_model(tuned_ada, plot='boundary')
```



8.0.3 Error

[22]:
```
plot_model(tuned_g, plot = 'error')
```



8.0.4 Seleccionar dinámicamente las gráficas

[23]:
```
evaluate_model(tuned_ada)
```

```
interactive(children=(ToggleButtons(description='Plot Type:', icons=('',),␣
↪options=(('Hyperparameters', 'param…
```

9 6.0 Predicción en el conjunto de entrenamiento

[24]:
```
predict_model(tuned_gbc);
```

```
<pandas.io.formats.style.Styler at 0x16e1fce10>
```

10 7.0 Entrenaremos sobre el 100% de los datos ya que el modelo fue afinado y está listo para producción

[25]:
```
final_gbc = finalize_model(tuned_gbc)
```

[26]:
```
#Final K Nearest Neighbour parameters for deployment
print(final_gbc)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.408, loss='deviance', max_depth=10,
                           max_features=1.0, max_leaf_nodes=None,
                           min_impurity_decrease=0.0005,
                           min_impurity_split=None, min_samples_leaf=3,
                           min_samples_split=10, min_weight_fraction_leaf=0.0,
                           n_estimators=170, n_iter_no_change=None,
                           presort='deprecated', random_state=123,
                           subsample=0.95, tol=0.0001, validation_fraction=0.1,
                           verbose=0, warm_start=False)
```

11 8.0 Serializamos el modelo

[29]:
```
save_model(final_gbc,'Final KNN Model 08Feb2020')
```

```
Transformation Pipeline and Model Succesfully Saved
```

[29]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True,
                                       features_todrop=['key', 'mode',
                                                        'artist_hotttnesss',
                                                        'end_of_fade_in',
                                                        'start_of_fade_out',
                                                        'mode_confidence',
                                                        'key_confidence',
                                                        'time_signature',
'time_signature_confidence',
                                                        'year', 'popularity',
                                                        'instrumentalness',
                                                        'liveness',
                                                        'duration_ms', 'Points',
                                                        'Country…
                                             loss='deviance', max_depth=10,
                                             max_features=1.0,
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0005,
                                             min_impurity_split=None,
                                             min_samples_leaf=3,
                                             min_samples_split=10,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=170,
                                             n_iter_no_change=None,
```

```
                                                  presort='deprecated',
                                                  random_state=123, subsample=0.95,
                                                  tol=0.0001,
validation_fraction=0.1,

                                                  verbose=0, warm_start=False)]],
          verbose=False),
 'Final KNN Model 08Feb2020.pkl')
```

(TIP : It's always good to use date in the filename when saving models, it's good for version control.)

12 9.0 Cargamos el modelo serializado

[34]:
```python
saved_final_gbc = load_model('Final KNN Model 08Feb2020')
```

```
Transformation Pipeline and Model Successfully Loaded
```

13 10.0 Vamos a traer datos que no ha visto nunca el modelo ya que son los que buscamos predecir. Los datos de JESC 2020 analizados por Spotify API

Evidentenemtnte estos datos no contienen la puntuación, ya que es lo que buscamos predecir

[36]:
```python
data_unseen=pd.read_csv("final.csv")
data_unseen
```

[36]:

	duration	key	loudness	mode	tempo	artist_hotttnesss	end_of_fade_in	\
0	181.99773	7	-9.342	0	95.001	0.23	0.00000	
1	179.98036	0	-3.938	1	113.932	0.26	0.00000	
2	182.60023	2	-7.322	1	113.981	0.17	0.24989	
3	173.16830	6	-6.834	1	101.021	0.35	0.15116	
4	180.54675	0	-2.799	1	122.028	0.31	0.00000	
5	177.33333	8	-6.671	1	180.020	0.14	0.00000	
6	167.98611	6	-6.184	0	100.040	0.30	2.61805	
7	157.90765	11	-7.370	0	153.369	0.19	0.53991	

	start_of_fade_out	mode_confidence	key_confidence	…	popularity	\
0	178.39311	0.873	0.876	…	35	
1	173.35439	0.636	0.742	…	38	
2	176.25687	0.687	0.619	…	28	
3	168.11247	0.768	0.742	…	46	
4	173.85940	0.490	0.546	…	42	
5	169.28508	0.311	0.507	…	26	
6	162.60934	0.379	0.062	…	42	
7	152.74086	0.655	0.696	…	31	

```
   danceability  energy  speechiness  acousticness  instrumentalness  \
0         0.565   0.563       0.0296        0.1310          0.000002
1         0.758   0.647       0.0419        0.4330          0.000000
2         0.667   0.405       0.0292        0.5470          0.000000
3         0.611   0.623       0.0367        0.0569          0.000045
4         0.523   0.851       0.0373        0.0148          0.000001
5         0.258   0.499       0.0377        0.4550          0.000000
6         0.744   0.574       0.1670        0.0555          0.000000
7         0.359   0.497       0.0439        0.4360          0.000000

   liveness  valence  duration_ms      Country
0    0.2530    0.114       181998      Belarus
1    0.1720    0.597       179980       France
2    0.1920    0.329       182600      Germany
3    0.0930    0.428       173168  Netherlands
4    0.2920    0.181       180547       Poland
5    0.0773    0.428       177333       Russia
6    0.0817    0.353       167986        Spain
7    0.0787    0.328       157908      Ukraine

[8 rows x 23 columns]
```

13.0.1 Vemos las columnas para tener mayor transparencia en este conjunto de datos que deseamos predecir

[38]:
```
data_unseen.columns
```

[38]:
```
Index(['duration', 'key', 'loudness', 'mode', 'tempo', 'artist_hotttnesss',
       'end_of_fade_in', 'start_of_fade_out', 'mode_confidence',
       'key_confidence', 'time_signature', 'time_signature_confidence', 'year',
       'popularity', 'danceability', 'energy', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'duration_ms', 'Country'],
      dtype='object')
```

[39]:
```
new_prediction = predict_model(saved_final_gbc, data=data_unseen)
```

13.1 Esta función predice la etiqueta y el "Score" (probabilidad de la clase predicha) utilizando un modelo entrenado.

[48]:
```
pd.merge(new_prediction.
 ↪sort_values(by='Label'),data_unseen['Country'],left_index=True,right_index=True).
 ↪sort_values(by=['Label','Score'],ascending=False)
```

[48]:
```
    duration  loudness    tempo  danceability  energy  speechiness  \
3  173.16830    -6.834  101.021         0.611   0.623       0.0367
1  179.98036    -3.938  113.932         0.758   0.647       0.0419
```

```
0  181.99773    -9.342   95.001         0.565   0.563       0.0296
4  180.54675    -2.799  122.028         0.523   0.851       0.0373
2  182.60023    -7.322  113.981         0.667   0.405       0.0292
5  177.33333    -6.671  180.020         0.258   0.499       0.0377
6  167.98611    -6.184  100.040         0.744   0.574       0.1670
7  157.90765    -7.370  153.369         0.359   0.497       0.0439

   acousticness  valence Label   Score      Country
3        0.0569    0.428  Good  0.9652  Netherlands
1        0.4330    0.597  Good  0.9250       France
0        0.1310    0.114  Good  0.8749      Belarus
4        0.0148    0.181  Good  0.8209       Poland
2        0.5470    0.329   Bad  0.9823      Germany
5        0.4550    0.428   Bad  0.9635       Russia
6        0.0555    0.353   Bad  0.9127        Spain
7        0.4360    0.328   Bad  0.7647      Ukraine
```

- Vemos entonces que la mejor canción según el modelo es Holanda, Francia, Bielorusia y Polonia.
- Las peores serían Alemania, Rusia, España y Ukrania

[ ]: