Todas las librerias

In [4]:
```
import datetime
import numpy as np
import pandas as pd
from datetime import date, timedelta, datetime
import os
import missingno as msno

from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import (adjusted_mutual_info_score, homogeneity_score,
                             completeness_score,classification_report, confusion_matr
                             mean_squared_error, mean_absolute_error,
                             mean_absolute_percentage_error,
                             silhouette_score, v_measure_score, adjusted_rand_score)
from sklearn.linear_model import ElasticNet, SGDClassifier, LogisticRegression
from sklearn.cluster import KMeans, DBSCAN, AffinityPropagation
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

import plotly.graph_objects as go
import plotly.express as px
import statsmodels.api as sm

from imblearn.over_sampling import RandomOverSampler
```

Ejercicio 1 (2 puntos): a) Crea una función que calcule y devuelva el factorial de un número entero. (0.6 puntos)

In [5]:
```
def factorial(n):
   if n==0 or n==1:
            resultado=1
   elif n>1:
            resultado=n*factorial(n-1)
   return resultado
print(factorial(4))
```

```
24
```

b) Crea una función que verifique si un número es primo o no. (0.6 puntos)

In [36]:
```
def es_primo(num):
    for n in range(2, num):
        if num % n == 0:
            return 0
    return 1

print(es_primo(4))
```

```
0
```

c) Muestra en un dataframe los 50 primeros números positivos, si es primo y su factorial utilizando las funciones anteriores. (0.6 puntos)

In [43]:
```
df=pd.DataFrame(columns=["numbers","primo","factorial"])
df["numbers"]=range(50)
df["primo"]=df["numbers"].apply(es_primo)
```

```
df["factorial"]=df["numbers"].apply(factorial)
print(df)
```

```
    numbers  primo                                       factorial
0         0      1                                               1
1         1      1                                               1
2         2      1                                               2
3         3      1                                               6
4         4      0                                              24
5         5      1                                             120
6         6      0                                             720
7         7      1                                            5040
8         8      0                                           40320
9         9      0                                          362880
10       10      0                                         3628800
11       11      1                                        39916800
12       12      0                                       479001600
13       13      1                                      6227020800
14       14      0                                     87178291200
15       15      0                                   1307674368000
16       16      0                                  20922789888000
17       17      1                                 355687428096000
18       18      0                                6402373705728000
19       19      1                              121645100408832000
20       20      0                             2432902008176640000
21       21      0                            51090942171709440000
22       22      0                          1124000727777607680000
23       23      1                         25852016738884976640000
24       24      0                        620448401733239439360000
25       25      0                      15511210043330985984000000
26       26      0                     403291461126605635584000000
27       27      0                   10888869450418352160768000000
28       28      0                  304888344611713860501504000000
29       29      1                 8841761993739701954543616000000
30       30      0               265252859812191058636308480000000
31       31      1              8222838654177922817725562880000000
32       32      0            263130836933693530167218012160000000
33       33      0           8683317618811886495518194401280000000
34       34      0         295232799039604140847618609643520000000
35       35      0       10333147966386144929666651337523200000000
36       36      0      371993326789901217467999448150835200000000
37       37      1    13763753091226345046315979581580902400000000
38       38      0   523022617466601111760007224100074291200000000
39       39      0 20397882081197443358640281739902897356800000000
40       40      0 815915283247897734345611269596115894272000000000
41       41      1 33452526613163807108170062053440751665152000000...
42       42      0 14050061177528798985431426062445115699363840000...
43       43      1 60415263063373835637355132068513997507264512000...
44       44      0 26582715747884487680436258110146158903196385280...
45       45      0 11962222086548019456196316149565771506438373376...
46       46      0 55026221598120889498503054288002548929616517529...
47       47      1 25862324151116818064296435515361197996919763238...
48       48      0 12413915592536072670862289047373375038521486354...
49       49      0 60828186403426756087225216332129537688755283137...
```

d) ¿Cómo se podría programar en una clase las tres operaciones anteriores? (0.2 puntos)

In [49]:

```python
class framePrimoFact():
    def __init__(self,n):
        self.df=pd.DataFrame(columns=["numbers","primo","factorial"])
        self.df["numbers"]=range(n)
        self.df["primo"]=self.df["numbers"].apply(es_primo)
        self.df["factorial"]=self.df["numbers"].apply(factorial)
```

```
    def es_primo(x):
        for n in range(2, x):
            if x % n == 0:
                return 0
        return 1

    def factorial(n):
        if n==0 or n==1:
                resultado=1
        elif n>1:
                resultado=n*factorial(n-1)
        return resultado

    def dataframe(self):
         return self.df

framePrimoFact(5).dataframe()
```

Out[49]:

	numbers	primo	factorial
0	0	1	1
1	1	1	1
2	2	1	2
3	3	1	6
4	4	0	24

Ejercicio 2 (4 puntos): a) Extrae de sklearn el conjunto de datos California Housing dataset y transfórmalo a dataframe de pandas (0.25 puntos)

b) Construye una función que muestra la estructura del dataset, el número de NAs, tipos de variables y estadísticas básicas de cada una de las variables. (0.5 puntos)

c) Construye una Regresión lineal y un Random forest que predigan el Median house value según los datos disponibles. (0.75 puntos)

d) Visualiza cuales son las variables (coeficientes) más importantes en cada uno de los modelos. (1.25 puntos)

e) Decide a través de las métricas que consideres oportunas, cuál de los dos modelos es mejor, por qué y explica el proceso que has realizado para responder en los puntos anteriores. (1.25 puntos)

In [6]:

```
# Carga del fichero de datos de clasificación
from sklearn.datasets import fetch_california_housing
dataset = sklearn.datasets.fetch_california_housing()
# Pasamos a formato dataframe
df = pd.DataFrame(data = dataset['data'], columns = dataset['feature_names'])
# Agregamos la variable objetivo
df["target"] = dataset["target"]
df.head()
```

Out[6]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

In [7]:

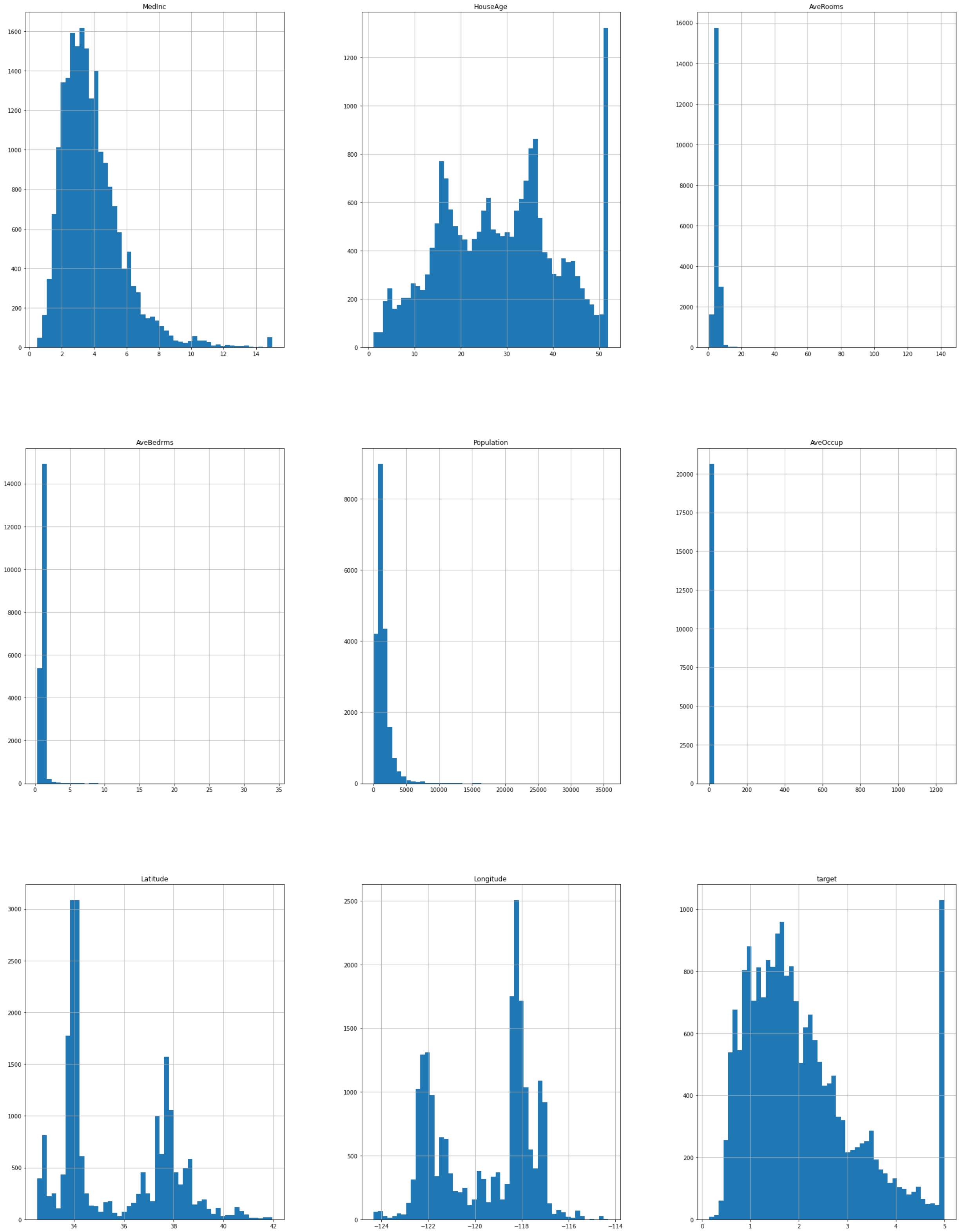
```
print(df.isna().sum())
print(df[df.isna().any(axis=1)]) # Observar las filas que contienen las NA's antes d
df.dropna(inplace=True)

df.hist(bins=50, figsize=(30,40))
df.corr()
```

```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
target        0
dtype: int64
Empty DataFrame
Columns: [MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Lon
gitude, target]
Index: []
```

Out[7]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.079809	-0.0151
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.011173	-0.1081
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.106389	-0.0275
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.069721	0.013
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.108785	0.0997
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.002366	0.002
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.000000	-0.924
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.924664	1.000
target	0.688075	0.105623	0.151948	-0.046701	-0.024650	-0.023737	-0.144160	-0.045



In [62]:

```
# Dividir train-test
X = dataset["data"].copy()
y = dataset["target"].copy()
X_train, X_test, y_train, y_test = train_test_split(X, y ,test_size = 0.3, random_st
reg = ElasticNet()
reg.fit(X_train,y_train)
predictions = reg.predict(X_test)
predictions_train = reg.predict(X_train)
# Metricas de evaluación
rmse_train = np.sqrt(mean_squared_error(y_train,predictions_train))
mae_train = mean_absolute_error(y_train, predictions_train)
mape_train = mean_absolute_percentage_error(y_train, predictions_train)

rmse_test = np.sqrt(mean_squared_error(y_test,predictions))
```

```
mae_test = mean_absolute_error(y_test, predictions)
mape_test = mean_absolute_percentage_error(y_test, predictions)

print("El RMSE de train del modelo es: {}".format(rmse_train))
print(f"El MAE de train del modelo es: {mae_train}")
print(f"El MAPE de train del modelo es: {100 * mape_train} %")

print("")

print("El RMSE de test del modelo es: {}".format(rmse_test))
print(f"El MAE de test del modelo es: {mae_test}")
print(f"El MAPE de test del modelo es: {100*mape_test} %")
```

```
El RMSE de train del modelo es: 0.877980259023968
El MAE de train del modelo es: 0.6807095985024406
El MAPE de train del modelo es: 45.20172523874712 %

El RMSE de test del modelo es: 0.8720194772729226
El MAE de test del modelo es: 0.6785162737312243
El MAPE de test del modelo es: 46.262056633536005 %
```

In [74]:

```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(n_estimators=50,max_depth=5)
clf.fit(X, y)
predictions=clf.predict(X_test)
predictions_train = clf.predict(X_train)
# Metricas de evaluación
rmse_train = np.sqrt(mean_squared_error(y_train,predictions_train))
mae_train = mean_absolute_error(y_train, predictions_train)
mape_train = mean_absolute_percentage_error(y_train, predictions_train)

rmse_test = np.sqrt(mean_squared_error(y_test,predictions))
mae_test = mean_absolute_error(y_test, predictions)
mape_test = mean_absolute_percentage_error(y_test, predictions)

print("El RMSE de train del modelo es: {}".format(rmse_train))
print(f"El MAE de train del modelo es: {mae_train}")
print(f"El MAPE de train del modelo es: {100 * mape_train} %")

print("")

print("El RMSE de test del modelo es: {}".format(rmse_test))
print(f"El MAE de test del modelo es: {mae_test}")
print(f"El MAPE de test del modelo es: {100*mape_test} %")
```

```
El RMSE de train del modelo es: 0.6601541694772718
El MAE de train del modelo es: 0.48105936327330867
El MAPE de train del modelo es: 29.774174449419156 %

El RMSE de test del modelo es: 0.6556572809260707
El MAE de test del modelo es: 0.48032090656151283
El MAPE de test del modelo es: 30.282448271518792 %
```

In [75]:

```
print(reg.coef_)
```

```
[ 2.53912734e-01  1.09735267e-02  0.00000000e+00 -0.00000000e+00
  1.11830749e-05 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00]
```

In [76]:

```
print(clf.feature_importances_)
```

```
[0.75478532 0.0408163  0.01854407 0.00081961 0.00379145 0.14200313
 0.0251567  0.01408342]
```

Elegiría la Regresión lineal porque aunque el RMSE es algo mayor, es más sencilla y utiliza menos coeficientes por lo que es más robusto el modelo.

Para las respuestas b, c, d, e, f y g es imperativo acompañarlas respuestas con una visualización.

a) Lee el fichero en formato dataframe, aplica la función del ejercicio 2.b, elimina NAs y convierte a integer si fuera necesario. (0.25 puntos)

b) ¿Cuántos artistas únicos hay? (0.25 puntos)

c) ¿Cuál es la distribución de reproducciones? (0.5 puntos)

d) ¿Existe una diferencia signitificativa en las reproducciones entre las canciones de un solo artista y las de más de uno? (0.5 puntos)

e) ¿Cuáles son las propiedades de una canción que mejor correlan con el número de reproducciones de una canción? (0.5 puntos)

f) ¿Cuáles son las variables que mejor predicen las canciones que están por encima el percentil 50? (1 puntos)

Nota: Crea una variable binaria (Hit/No Hit) en base a 3.c, crea una regresión logística y visualiza sus coeficientes.

g) Agrupa los 4 gráficos realizados en uno solo y haz una recomendación a un sello discográfico para producir un nuevo hit. (1 puntos)

In [163…

```
df = pd.read_csv("spotify.csv", encoding = "ISO-8859-1")
print(df.isna().sum())
print(df[df.isna().any(axis=1)]) # Observar las filas que contienen las NA's antes d
df.dropna(inplace=True)
df.head()
```

```
track_name               0
artist(s)_name           0
artist_count             0
released_year            0
released_month           0
released_day             0
in_spotify_playlists     0
in_spotify_charts        0
streams                  0
in_apple_playlists       0
in_apple_charts          0
in_deezer_playlists      0
in_deezer_charts         0
in_shazam_charts        50
bpm                      0
key                     95
mode                     0
danceability_%           0
valence_%                0
energy_%                 0
acousticness_%           0
instrumentalness_%       0
liveness_%               0
speechiness_%            0
dtype: int64
```

```
                                             track_name  \
12                                              Flowers   
14                                            As It Was   
17   What Was I Made For? [From The Motion Picture ...   
22                                      I Wanna Be Yours   
35                                      Los del Espacio   
..                                                  ...   
901                                          After LIKE   
903             B.O.T.A. (Baddest Of Them All) - Edit   
927               I Really Want to Stay at Your House   
938                                           Labyrinth   
940                                       Sweet Nothing   

                                        artist(s)_name  artist_count  \
12                                        Miley Cyrus             1   
14                                       Harry Styles             1   
17                                      Billie Eilish             1   
22                                     Arctic Monkeys             1   
35   Big One, Duki, Lit Killah, Maria Becerra, FMK,...             8   
..                                                 ...           ...   
901                                                IVE             1   
903                Interplanetary Criminal, Eliza Rose             2   
927                        Rosa Walton, Hallie Coggins             2   
938                                       Taylor Swift             1   
940                                       Taylor Swift             1   

     released_year  released_month  released_day  in_spotify_playlists  \
12            2023               1            12                 12211   
14            2022               3            31                 23575   
17            2023               7            13                   873   
22            2013               1             1                 12859   
35            2023               6             1                  1150   
..             ...             ...           ...                   ...   
901           2022               8            22                   767   
903           2022               6            15                  5153   
927           2020              12            18                   668   
938           2022              10            21                  1597   
940           2022              10            21                  1747   

     in_spotify_charts     streams  in_apple_playlists  ...  bpm  key   mode  \
12                 115  1316855716                 300  ...  118  NaN  Major   
14                 130  2513188493                 403  ...  174   F#  Minor   
17                 104    30546883                  80  ...   78  NaN  Major   
22                 110  1297026226                  24  ...  135  NaN  Minor   
35                  31   123122413                  22  ...  120  NaN  Major   
..                 ...         ...                 ...  ...  ...  ...    ...   
901                 12   265548837                  20  ...  125  NaN  Major   
903                  6   244585109                 102  ...  137  NaN  Major   
927                  1   140430339                   0  ...  125   D#  Minor   
938                  0   187339835                   6  ...  110  NaN  Major   
940                  0   186104310                   9  ...  177  NaN  Major   

    danceability_%  valence_% energy_% acousticness_%  instrumentalness_%  \
12              71         65       68              6                   0   
14              52         66       73             34                   0   
17              44         14        9             96                   0   
22              48         44       42             12                   2   
35              81         63       68             11                   0   
..             ...        ...      ...            ...                 ...   
901             68         80       92             10                   0   
903             74         71       89             24                  61   
927             49         13       74              0                   0   
938             48         15       31             80                  22   
940             34         39       16             97                   0   
```

```
     liveness_%  speechiness_%
12            3              7
14           31              6
17           10              3
22           11              3
35           11              4
..          ...            ...
901           9             12
903          15              5
927           9              4
938          12              4
940          12              5

[136 rows x 24 columns]
```

Out[163…

	track_name	artist(s)_name	artist_count	released_year	released_month	released_day	in_spotify_p
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	14	
1	LALA	Myke Towers	1	2023	3	23	
2	vampire	Olivia Rodrigo	1	2023	6	30	
3	Cruel Summer	Taylor Swift	1	2019	8	23	
4	WHERE SHE GOES	Bad Bunny	1	2023	5	18	

5 rows × 24 columns

In [164…

```
df[['streams']] = df[['streams']].apply(pd.to_numeric, errors='coerce')
df.describe()
```

Out[164…

	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_charts
count	817.000000	817.000000	817.000000	817.000000	817.000000	817.00000(
mean	1.567931	2018.457772	6.018360	13.696450	4849.898409	11.722154
std	0.876211	10.829267	3.572554	9.299663	7741.126455	18.61766ε
min	1.000000	1930.000000	1.000000	1.000000	31.000000	0.00000(
25%	1.000000	2021.000000	3.000000	5.000000	829.000000	0.00000(
50%	1.000000	2022.000000	5.000000	13.000000	2040.000000	3.00000(
75%	2.000000	2022.000000	9.000000	22.000000	4890.000000	16.00000(
max	8.000000	2023.000000	12.000000	31.000000	52898.000000	147.00000(

In [165…

```
len(df["artist(s)_name"].unique())
```

Out[165…

```
571
```

In [166…

```
import plotly.express as px
fig = px.histogram(df, x="artist(s)_name", y="streams")
fig.show()
```

La proporcion total de reproducciones partido de numero de artistas totales, es mejor para solos

In [167…

```
df_solo=df[df["artist_count"]==1]
df_varios=df[df["artist_count"]!=1]


print((df_solo["streams"].sum()/len(df_solo["artist(s)_name"])))

print((df_varios["streams"].sum()/len(df_varios["artist(s)_name"])))
```

```
505369297.91093117
411888391.91640866
```

Correla con las playlist de spotify y apple con lo que mejor

In [182…

```
df_corr=df.corr()
df_corr["streams"]
```

Out[182…

	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_charts
count	817.000000	817.000000	817.000000	817.000000	817.000000	817.00000C
mean	1.567931	2018.457772	6.018360	13.696450	4849.898409	11.72215‹
std	0.876211	10.829267	3.572554	9.299663	7741.126455	18.61766ξ
min	1.000000	1930.000000	1.000000	1.000000	31.000000	0.00000C
25%	1.000000	2021.000000	3.000000	5.000000	829.000000	0.00000C
50%	1.000000	2022.000000	5.000000	13.000000	2040.000000	3.00000C
75%	2.000000	2022.000000	9.000000	22.000000	4890.000000	16.00000C
max	8.000000	2023.000000	12.000000	31.000000	52898.000000	147.00000C

In [200…

```
df["Hits"]=(df['streams'].gt(df['streams'].median())).astype(int)
# Dividir juego de datos en entrenamiento y test
# Realizar siempre antes de crear un modelo.

dfX = df.loc[:, df.columns !="Hits"].select_dtypes([np.number])
X = dfX.replace('NaN',0)
y = df["Hits"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_s
# Inicializar modelos
# Crear un objeto de sobremuestreo
ros = RandomOverSampler(random_state=0)

# Aplicar el sobremuestreo a tus datos
X_resampled, y_resampled = ros.fit_resample(X, y)

# X_resampled y y_resampled son tus datos rebalanceados

# Regresión logística
```

```
clf_log = LogisticRegression(max_iter=10000, tol=0.1)
clf_log.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11100/1071989684.py in <module>
     19 # Regresión logística
     20 clf_log = LogisticRegression(max_iter=10000, tol=0.1)
---> 21 clf_log.fit(X_train, y_train)
     22 

c:\Anaconda\lib\site-packages\sklearn\base.py in wrapper(estimator, *args, **kwargs)
   1150                 )
   1151             ):
-> 1152                 return fit_method(estimator, *args, **kwargs)
   1153 
   1154         return wrapper

c:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py in fit(self, X, y, s
ample_weight)
   1206             _dtype = [np.float64, np.float32]
   1207 
-> 1208         X, y = self._validate_data(
   1209             X,
   1210             y,

c:\Anaconda\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, v
alidate_separately, cast_to_ndarray, **check_params)
    620                 y = check_array(y, input_name="y", **check_y_params)
    621             else:
--> 622                 X, y = check_X_y(X, y, **check_params)
    623             out = X, y
    624 

c:\Anaconda\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_
sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_
nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
   1144         )
   1145 
-> 1146     X = check_array(
   1147         X,
   1148         accept_sparse=accept_sparse,

c:\Anaconda\lib\site-packages\sklearn\utils\validation.py in check_array(array, acce
pt_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, all
ow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
    955 
    956         if force_all_finite:
--> 957             _assert_all_finite(
    958                 array,
    959                 input_name=input_name,

c:\Anaconda\lib\site-packages\sklearn\utils\validation.py in _assert_all_finite(X, a
llow_nan, msg_dtype, estimator_name, input_name)
    120         return
    121 
--> 122     _assert_all_finite_element_wise(
    123         X,
    124         xp=xp,

c:\Anaconda\lib\site-packages\sklearn\utils\validation.py in _assert_all_finite_elem
ent_wise(X, xp, allow_nan, msg_dtype, estimator_name, input_name)
    169                 "#estimators-that-handle-nan-values"
    170             )
```

```
--> 171         raise ValueError(msg_err)
    172
    173

ValueError: Input X contains NaN.
LogisticRegression does not accept missing values encoded as NaN natively. For super
vised learning, you might want to consider sklearn.ensemble.HistGradientBoostingClas
sifier and Regressor which accept missing values encoded as NaNs natively. Alternati
vely, it is possible to preprocess the data, for instance by using an imputer transf
ormer in a pipeline or drop samples with missing values. See https://scikit-learn.or
g/stable/modules/impute.html You can find a list of all estimators that handle NaN v
alues at the following page: https://scikit-learn.org/stable/modules/impute.html#est
imators-that-handle-nan-values
```

In [ ]:

```
predictions = clf_log.predict(X_test)

print("Classification report")
print(classification_report(y_test, predictions, target_names=dataset["target_names"

print("Confusion matrix")
print(confusion_matrix(y_test, predictions))

predictions = clf_sgd.predict(X_test)

print("Classification report")
print(classification_report(y_test, predictions, target_names=dataset["target_names"

print("Confusion matrix")
print(confusion_matrix(y_test, predictions))
```