

Prueba Intertrimestral

Nombre: Álvaro

Apellidos: Olivié Molina

Tiempo de la prueba: 2 Horas

Asignatura: Desarrollo de Aplicaciones para la Visualización de Datos

Fecha: 18 de octubre de 2023

Instrucciones:

- Escribe código limpio y autoexplicativo.
- Se eliminará 0.5 puntos por usar Seaborn o Matplotlib.
- Se pueden utilizar los materiales de clase.
- Se puede utilizar internet para búsqueda de dudas y documentación.
- No se puede utilizar ningún tipo de LLM.
- No se puede utilizar mensajería instantánea.
- Sube tus resultados a tu repositorio de Github.
- Imprime una versión en PDF en A3 y Portrait del notebook.
- Envialo tus resultados a dmartincorral@icai.comillas.edu adjuntando el PDF y la url del notebook subido al repositorio de Github.

Inicialización de librerías

Carga aquí todas las librerías que vayas a utilizar.

```
In [ ]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
```

Ejercicio 1 (2 puntos):

a) Crea una función que calcule y devuelva el factorial de un número entero. **(0.6 puntos)**

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

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

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

In []:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

In []:

```
def es_primo(n):
    if n < 2:
        return False
    elif n == 2:
        return True
    else:
        for i in range(2, n):
            if n % i == 0:
                return False
        return True
```

In []:

```
list = []
for i in range(1, 51):
    list.append({'Número': i, 'Primo': es_primo(i), 'Factorial': factorial(i)})
df = pd.DataFrame(list, columns=['Número', 'Primo', 'Factorial'])
df
```

Out[]:

	Número	Primo	Factorial
0	1	False	1
1	2	True	2
2	3	True	6
3	4	False	24
4	5	True	120
5	6	False	720
6	7	True	5040
7	8	False	40320
8	9	False	362880
9	10	False	3628800
10	11	True	39916800
11	12	False	479001600
12	13	True	6227020800
13	14	False	87178291200
14	15	False	1307674368000
15	16	False	20922789888000
16	17	True	355687428096000
17	18	False	6402373705728000
18	19	True	121645100408832000
19	20	False	2432902008176640000
20	21	False	51090942171709440000
21	22	False	1124000727777607680000
22	23	True	25852016738884976640000
23	24	False	620448401733239439360000
24	25	False	15511210043330985984000000
25	26	False	403291461126605635584000000
26	27	False	10888869450418352160768000000
27	28	False	304888344611713860501504000000
28	29	True	8841761993739701954543616000000
29	30	False	265252859812191058636308480000000
30	31	True	8222838654177922817725562880000000
31	32	False	263130836933693530167218012160000000
32	33	False	8683317618811886495518194401280000000

Número	Primo		Factorial
33	34	False	295232799039604140847618609643520000000
34	35	False	1033314796638614492966651337523200000000
35	36	False	371993326789901217467999448150835200000000
36	37	True	13763753091226345046315979581580902400000000
37	38	False	523022617466601111760007224100074291200000000
38	39	False	20397882081197443358640281739902897356800000000
39	40	False	815915283247897734345611269596115894272000000000
40	41	True	3345252661316380710817006205344075166515200000...
41	42	False	1405006117752879898543142606244511569936384000...
42	43	True	6041526306337383563735513206851399750726451200...
43	44	False	2658271574788448768043625811014615890319638528...
44	45	False	1196222208654801945619631614956577150643837337...
45	46	False	5502622159812088949850305428800254892961651752...
46	47	True	2586232415111681806429643551536119799691976323...
47	48	False	1241391559253607267086228904737337503852148635...
48	49	False	6082818640342675608722521633212953768875528313...
49	50	False	3041409320171337804361260816606476884437764156...

Creando una donde su función sea generar el dataframe y los métodos internos sean los creados anteriormente. El atributo sería el daataframe y se crearía al instanciar la clase.

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 [ ]: california_housing = fetch_california_housing(as_frame=True)

ch = pd.DataFrame(data = california_housing['data'], columns = california_housing['feature_names'])
y = california_housing['target']
```

```
In [ ]: def estructura(df):
    print('Estructura del dataset')
    print(df.info())
    print('Número de NAs')
    print(df.isna().sum())
    print('Tipos de variables')
    print(df.dtypes)
    print('Estadísticas básicas')
    print(df.describe())
```

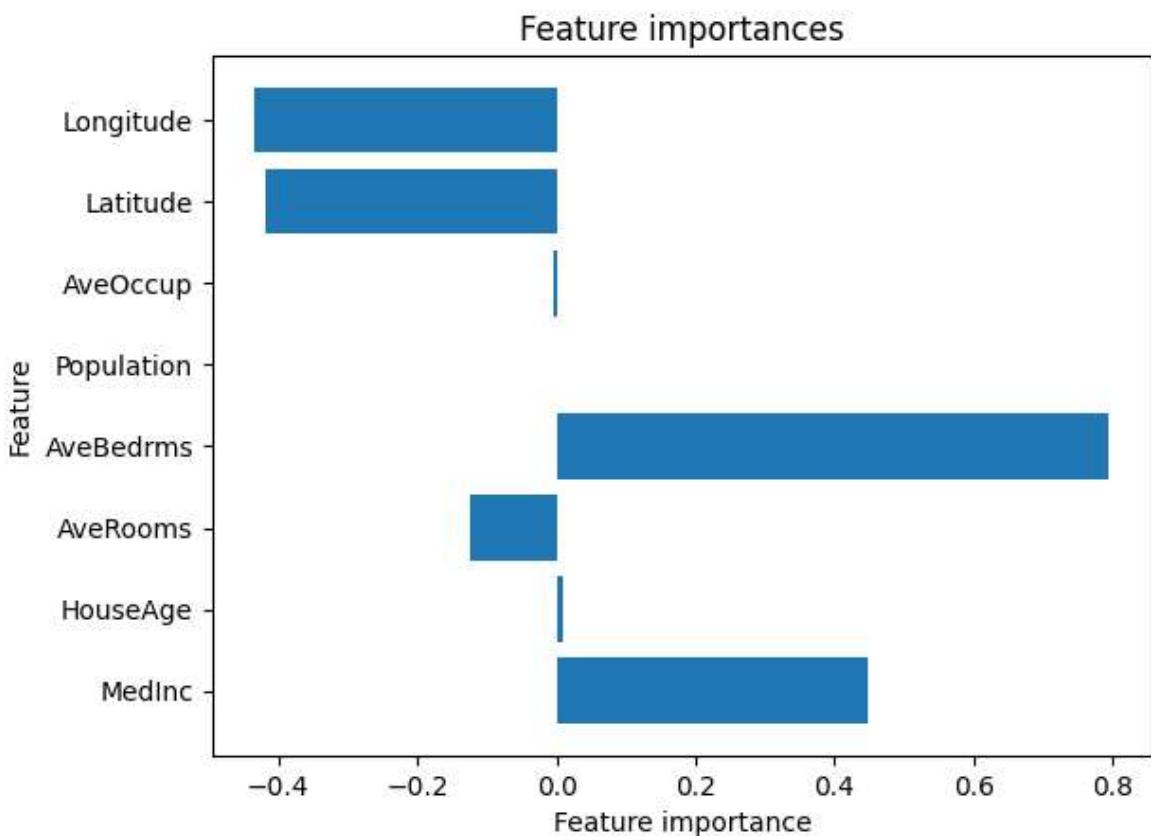
```
In [ ]: train_ch, test_ch, train_y, test_y = train_test_split(ch, y, test_size = 0.25, random_state=42)
```

```
In [ ]: lr = LinearRegression()
lr.fit(train_ch, train_y)
plr = lr.predict(test_ch)
error = mean_squared_error(test_y, plr)
print('El error lr:', error)
r2 = r2_score(test_y, plr)
print('El R2 lr:', r2)

plt.figure()
plt.title("Feature importances")
plt.barh(ch.columns, lr.coef_)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```

El error lr: 0.541128747847069

El R2 lr: 0.591050979549135

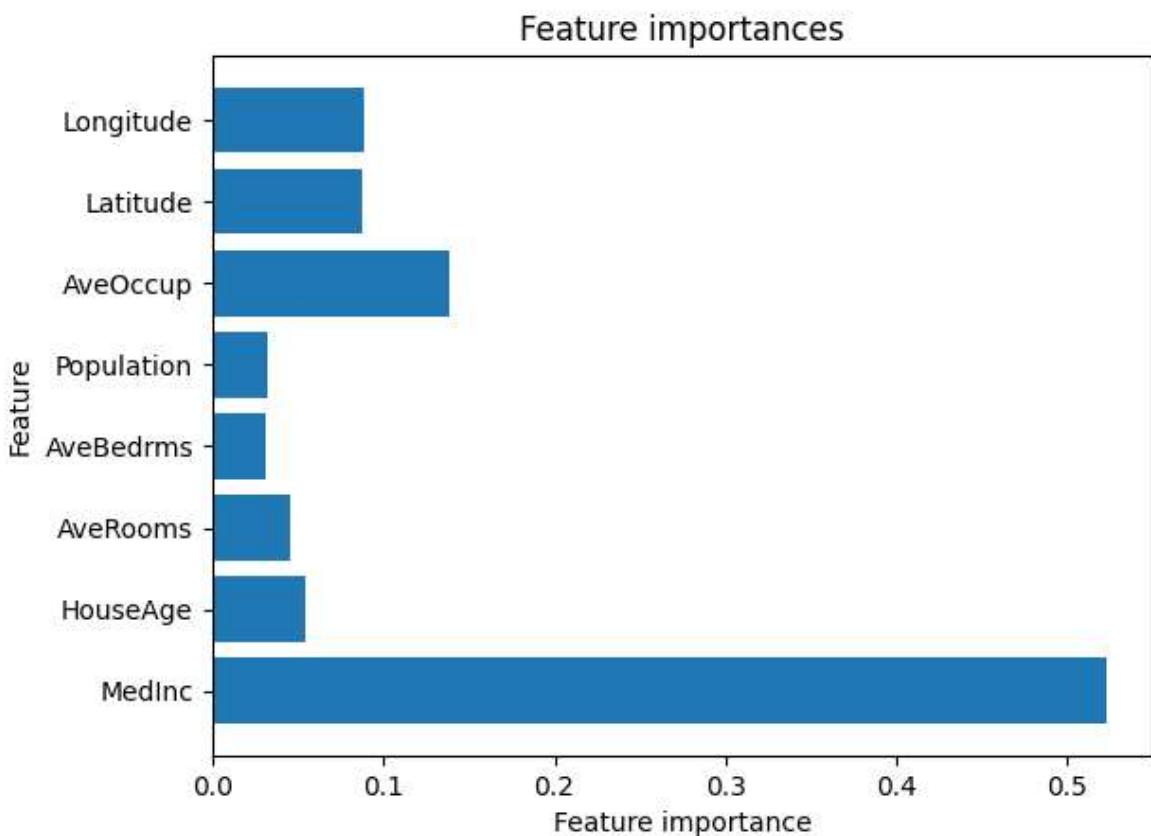


```
In [ ]: rf = RandomForestRegressor()
rf.fit(train_ch, train_y)
prf = rf.predict(test_ch)
error = mean_squared_error(test_y, prf)
print('El error rf:', error)
r2 = r2_score(test_y, prf)
print('El R2 rf:', r2)

plt.figure()
plt.title("Feature importances")
plt.barh(ch.columns, rf.feature_importances_)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```

El error rf: 0.25448766543513524

El R2 rf: 0.8076751939153334



El modelo de random forest es mejor que el modelo de linear regression. Esto se ve en que tiene un R cuadrado mucho mas alto y un error mucho mas bajo. Cabe destacar que el modelo de linear regression es mucho mas rapido

Para realizar los dos ejercicios anteriores he seguido la misma estructura para ambos modelos. He dividido entre train y test los datos, he creado un modelo y lo he entrenado con los datos train. Despues he predecido los resultados con test y he comparado con el test_y. Con estos resultados he calculado el error y R cuadrado que sirven como metricas de un buen modelo.

Para visualizar el peso de las variables de cada modelo he hecho un plot con el peso que le he dado el modelo a cada variable. En ambos casos el ingreso medio ha tenido un peso importante pero en el modelo de regresion lineal tambien ha afectado mucho el numero de cuartos

Ejercicio 3 (4 puntos):

Consideremos el dataset que contiene **The Most Streamed Spotify Songs 2023** que se encuentra en el repositorio.

Información de las variables:

- track_name: Name of the song
- artist(s)_name: Name of the artist(s) of the song
- vartist_count: Number of artists contributing to the song
- released_year: Year when the song was released
- released_month: Month when the song was released

- release_day: Day of the month when the song was released
- in_spotify_playlists: Number of Spotify playlists the song is included in
- in_spotify_charts: Presence and rank of the song on Spotify charts
- streams: Total number of streams on Spotify
- in_apple_playlists: Number of Apple Music playlists the song is included in
- in_apple_charts: Presence and rank of the song on Apple Music charts
- in_deezer_playlists: Number of Deezer playlists the song is included in
- in_deezer_charts: Presence and rank of the song on Deezer charts
- in_shazam_charts: Presence and rank of the song on Shazam charts
- bpm: Beats per minute, a measure of song tempo
- key: Key of the song
- mode: Mode of the song (major or minor)
- danceability_%: Percentage indicating how suitable the song is for dancing
- valence_%: Positivity of the song's musical content
- energy_%: Perceived energy level of the song
- acousticness_%: Amount of acoustic sound in the song
- instrumentalness_%: Amount of instrumental content in the song
- liveness_%: Presence of live performance elements
- speechiness_%: Amount of spoken words in the song

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 significativa 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 [ ]: df = pd.read_csv("spotify-2023.csv", encoding='iso-8859-1')

estructura(df)

df = df.dropna()
```

```
pd.to_numeric(df["streams"], errors='coerce')
df["streams"] = df["streams"].astype('Int64')
df["in_deezer_playlists"] = df["in_deezer_playlists"].str.replace(',', '')
df["in_shazam_charts"] = df["in_shazam_charts"].str.replace(',', '')
df["in_deezer_playlists"] = df["in_deezer_playlists"].astype('Int64')
df["in_shazam_charts"] = df["in_shazam_charts"].astype('Int64')
```

```
Estructura del dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 953 entries, 0 to 952
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   track_name        953 non-null    object  
 1   artist(s)_name   953 non-null    object  
 2   artist_count      953 non-null    int64  
 3   released_year     953 non-null    int64  
 4   released_month    953 non-null    int64  
 5   released_day      953 non-null    int64  
 6   in_spotify_playlists 953 non-null    int64  
 7   in_spotify_charts 953 non-null    int64  
 8   streams           953 non-null    object  
 9   in_apple_playlists 953 non-null    int64  
 10  in_apple_charts   953 non-null    int64  
 11  in_deezer_playlists 953 non-null    object  
 12  in_deezer_charts  953 non-null    int64  
 13  in_shazam_charts  903 non-null    object  
 14  bpm               953 non-null    int64  
 15  key               858 non-null    object  
 16  mode              953 non-null    object  
 17  danceability_%    953 non-null    int64  
 18  valence_%         953 non-null    int64  
 19  energy_%          953 non-null    int64  
 20  acousticness_%   953 non-null    int64  
 21  instrumentalness_% 953 non-null    int64  
 22  liveness_%        953 non-null    int64  
 23  speechiness_%    953 non-null    int64  
dtypes: int64(17), object(7)
memory usage: 178.8+ KB
None
Número de NAs
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
Tipos de variables
```

```

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

Estadísticas básicas
      artist_count released_year released_month released_day \
count      953.000000      953.000000      953.000000      953.000000
mean       1.556139     2018.238195      6.033578     13.930745
std        0.893044     11.116218      3.566435      9.201949
min        1.000000     1930.000000      1.000000      1.000000
25%        1.000000     2020.000000      3.000000      6.000000
50%        1.000000     2022.000000      6.000000     13.000000
75%        2.000000     2022.000000      9.000000     22.000000
max        8.000000     2023.000000     12.000000     31.000000

      in_spotify_playlists in_spotify_charts in_apple_playlists \
count      953.000000      953.000000      953.000000
mean       5200.124869      12.009444      67.812172
std        7897.608990      19.575992      86.441493
min        31.000000      0.000000      0.000000
25%        875.000000      0.000000      13.000000
50%        2224.000000      3.000000      34.000000
75%        5542.000000      16.000000      88.000000
max        52898.000000     147.000000     672.000000

      in_apple_charts in_deezer_charts bpm danceability_% \
count      953.000000      953.000000      953.000000      953.000000
mean       51.908709      2.666317     122.540399      66.96957
std        50.630241      6.035599     28.057802      14.63061
min        0.000000      0.000000     65.000000      23.00000
25%        7.000000      0.000000    100.000000      57.00000
50%        38.000000      0.000000    121.000000      69.00000
75%        87.000000      2.000000    140.000000      78.00000
max        275.000000     58.000000    206.000000      96.00000

      valence_% energy_% acousticness_% instrumentalness_% liveness_% \
count      953.000000      953.000000      953.000000      953.000000      953.000000
mean       51.431270     64.279119     27.057712      1.581322     18.213012
std        23.480632     16.550526     25.996077      8.409800     13.711223

```

```
min      4.000000    9.000000      0.000000      0.000000    3.000000
25%    32.000000   53.000000      6.000000      0.000000  10.000000
50%    51.000000   66.000000     18.000000      0.000000  12.000000
75%    70.000000   77.000000     43.000000      0.000000  24.000000
max    97.000000   97.000000    97.000000    91.000000  97.000000
```

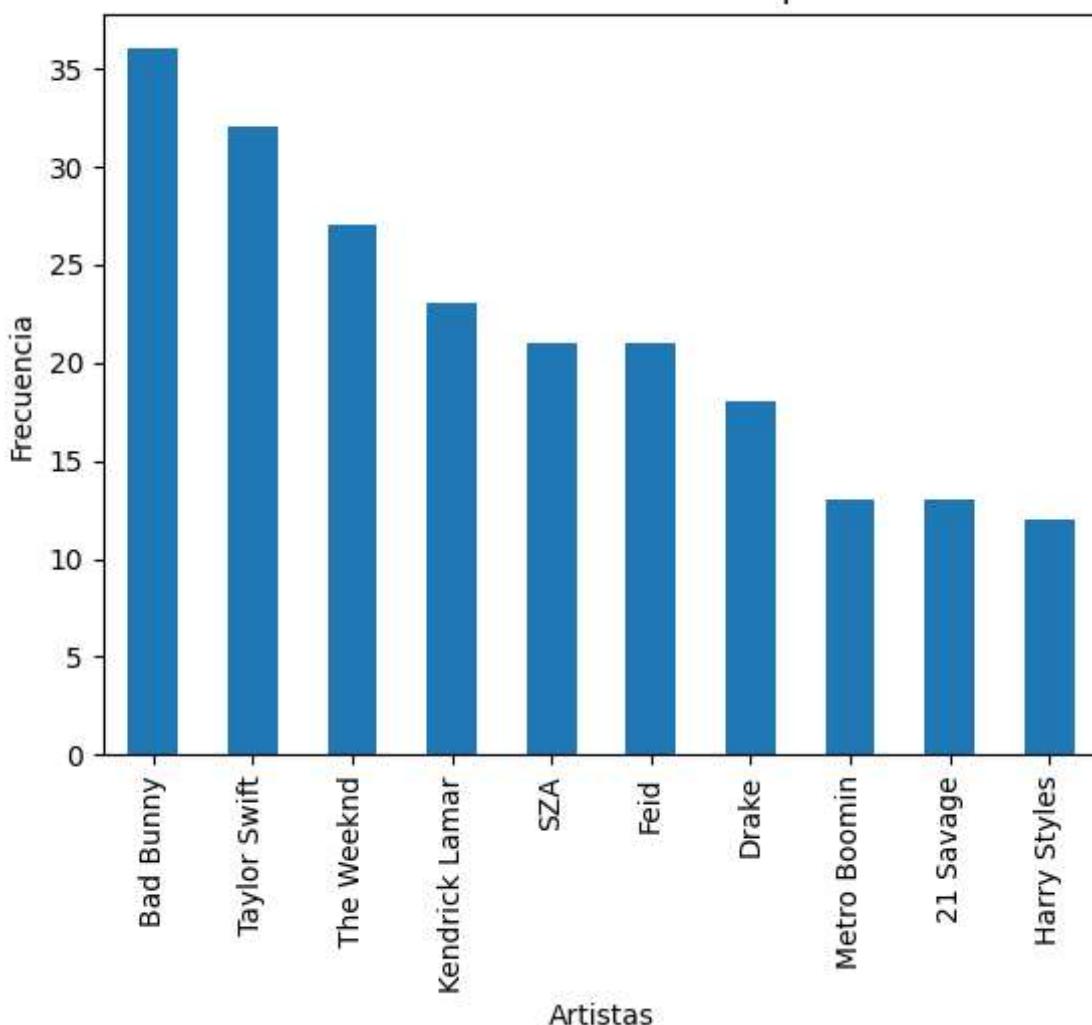
```
speechiness_%
count    953.000000
mean     10.131165
std      9.912888
min      2.000000
25%     4.000000
50%     6.000000
75%    11.000000
max    64.000000
```

```
In [ ]: artists = []
for i in range(len(df)):
    a = df.iloc[i, 1].split(', ')
    for i in a:
        artists.append(i)
artists = pd.Series(artists)
print(len(artists.unique()))


plt.figure()
plt.title("Distribución de artistas Top 10")
artists.value_counts()[:10].plot(kind='bar')
plt.xlabel("Artistas")
plt.ylabel("Frecuencia")
plt.show()
```

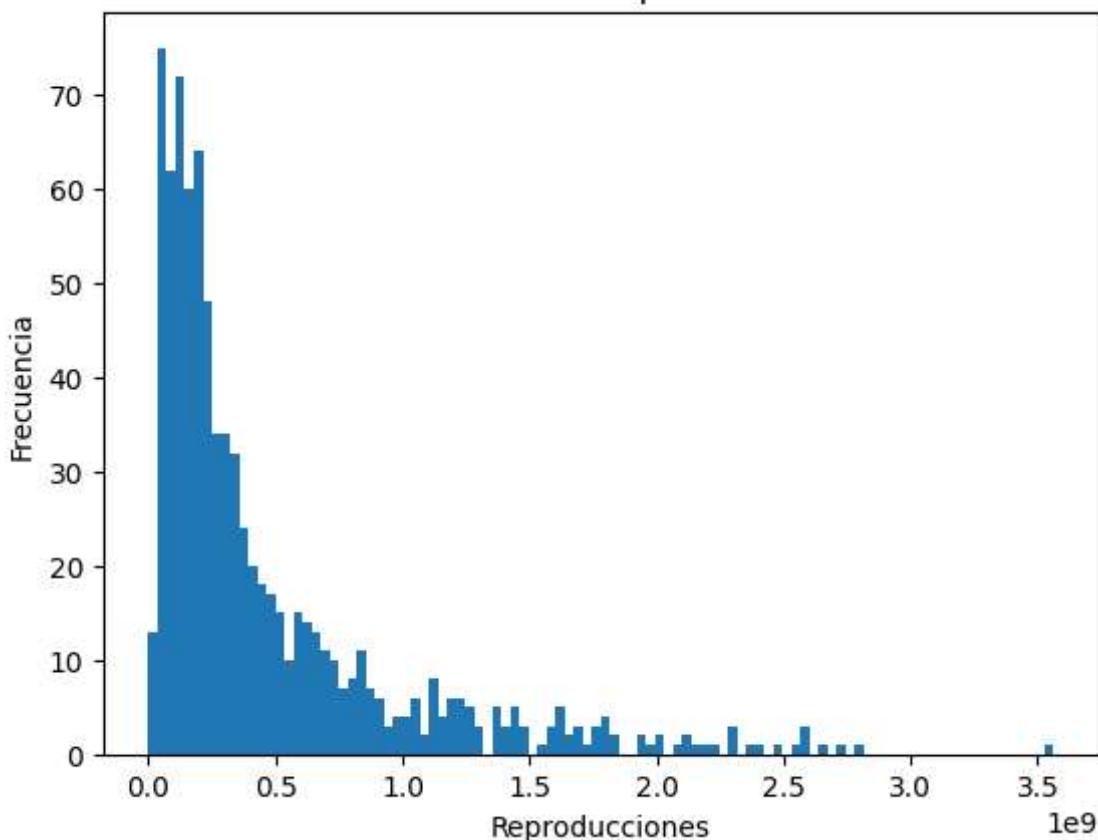
627

Distribución de artistas Top 10



```
In [ ]: plt.figure()
plt.title("Distribución de reproducciones")
plt.hist(df['streams'], bins=100)
plt.xlabel("Reproducciones")
plt.ylabel("Frecuencia")
plt.show()
```

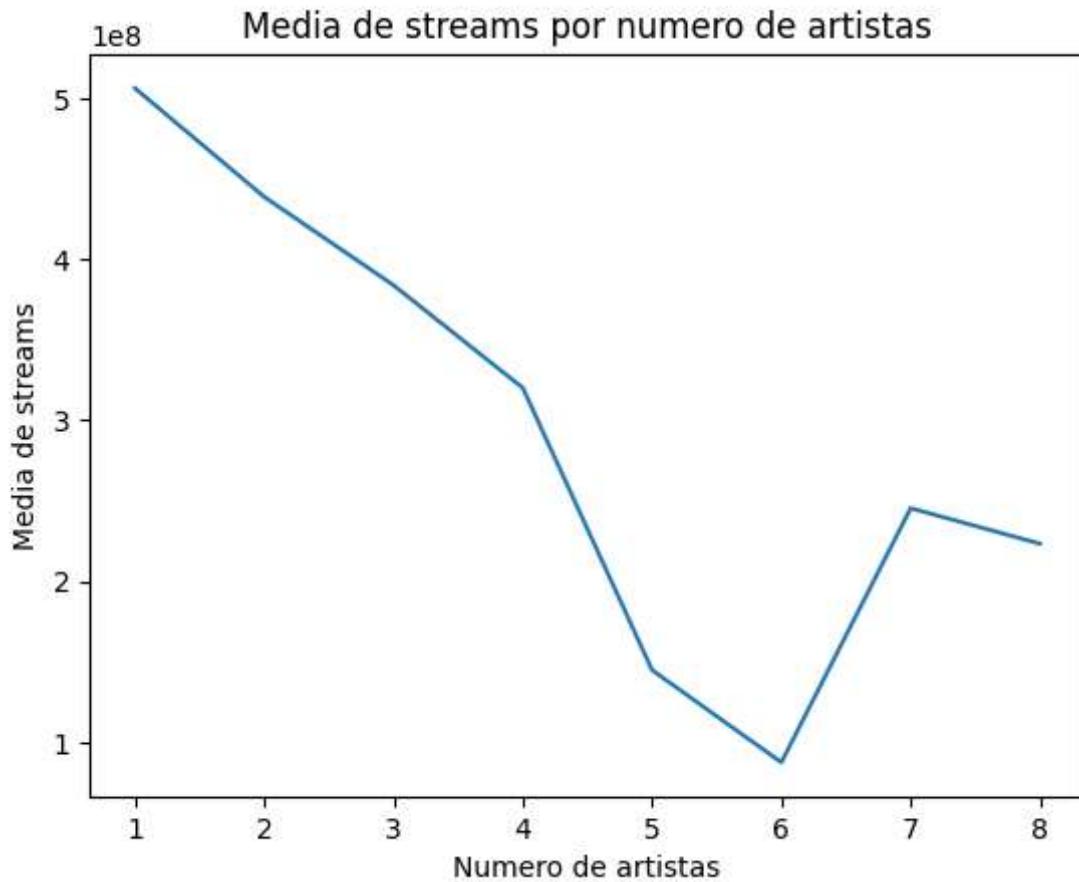
Distribución de reproducciones



```
In [ ]: df1 = df[df['artist_count'] == 1]
df2 = df[df['artist_count'] == 2]
df3 = df[df['artist_count'] == 3]
df4 = df[df['artist_count'] == 4]
df5 = df[df['artist_count'] == 5]
df6 = df[df['artist_count'] == 6]
df7 = df[df['artist_count'] == 7]
df8 = df[df['artist_count'] == 8]

plt.figure()
plt.title("Media de streams por numero de artistas")
plt.plot([1, 2, 3, 4, 5, 6, 7, 8],
         [df1['streams'].mean(),
          df2['streams'].mean(),
          df3['streams'].mean(),
          df4['streams'].mean(),
          df5['streams'].mean(),
          df6['streams'].mean(),
          df7['streams'].mean(),
          df8['streams'].mean()])
plt.xlabel("Numero de artistas")
plt.ylabel("Media de streams")
```

```
Out[ ]: Text(0, 0.5, 'Media de streams')
```

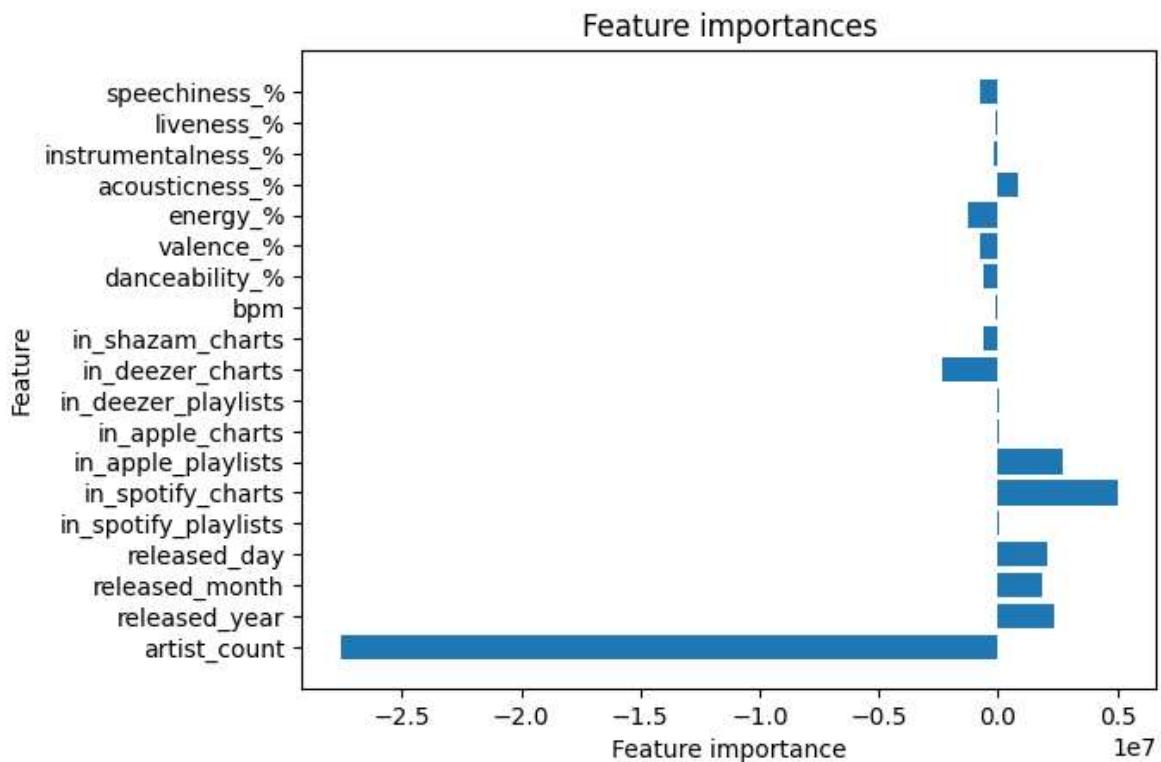


Hay una tendencia a bajar a medida que el numero de cantantes sube. Al final vuelve a subir pero esto es debido a que hay muy pocas canciones con tantos cantantes y suele ser gente famosa que se unen y se hacen virales.

```
In [ ]: target = df['streams']
features = df.drop(['streams', 'track_name', 'artist(s)_name', 'key', 'mode'], axis=1)
lr = LinearRegression()
lr.fit(features, target)
plr = lr.predict(features)
r2 = r2_score(target, plr)
print('El R2 lr:', r2)

plt.figure()
plt.title("Feature importances")
plt.barh(features.columns, lr.coef_)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```

El R2 lr: 0.7229380784007834



```
In [ ]: target = target > target.mean()
lreg = LogisticRegression()
lreg.fit(features, target)
plreg = lreg.predict(features)
r2 = r2_score(target, plreg)
print('El R2 lreg:', r2)

plt.figure()
plt.title("Feature importances")
plt.barh(features.columns, lreg.coef_[0])
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```

El R2 lreg: 0.47633928571428574

```
C:\Users\aolivie\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

