

▼ Prueba Intertrimestral

**Nombre:** Jorge

**Apellidos:** González Cardelús

**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.
  - Envíalo tus resultados a [dmartincorral@icai.comillas.edu](mailto: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.

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.express as px

from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from statsmodels.regression.linear_model import OLS
from sklearn.metrics import r2_score, mean_squared_error
```

▼ 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)**

```
class Operations:
    def fact(self, n):
        if n == 0:
            return 1



        if n>1:
            return self.fact(n-1)*n
        elif n==1:
            return 1

    def is_prime(self, n):
        check = 2
        while check < n:
            if n % check == 0:
                # Es divisble entre check, no puede ser primo
                return False
            else:
                check += 1
        return True

    def create_dataframe(self, n=50):
        data = []
        for i in range(n):
            data.append([i, self.fact(i), self.is_prime(i)])

        df = pd.DataFrame(data, columns=["number", "factorial", "is_prime"])
        return df

operations = Operations()
operations_df = operations.create_dataframe(50)
operations_df.head(20)
```

	number	factorial	is_prime	
0	0	1	True	
1	1	1	True	
2	2	2	True	
3	3	6	True	
4	4	24	False	
5	5	120	True	
-	-	---	-	

▼ 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)**



15151307674368000False

housing = datasets.fetch\_california\_housing()  
print(housing.keys())

dict\_keys(['data', 'target', 'frame', 'target\_names', 'feature\_names', 'DESCR'])

housing\_dataframe=pd.DataFrame(housing["data"], columns=housing["feature\_names"])  
housing\_dataframe["target"] = housing["target"]

housing\_dataframe.head(5)

	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	

```
def describe_df(df):  
    print("Dataframe information:\n\n")  
    print(df.info())  
    print("\n\nDataframe statistics:\n\n")  
    print(df.describe())  
  
    print("\n\nNAs:\n\n")  
    for column in df.columns:  
        print(column + ": " + str(df[column].isna().sum()))
```

describe\_df(housing\_dataframe)

Dataframe information:

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 20640 entries, 0 to 20639  
Data columns (total 9 columns):  
# Column Non-Null Count Dtype  
--- -  
0 MedInc 20640 non-null float64  
1 HouseAge 20640 non-null float64  
2 AveRooms 20640 non-null float64  
3 AveBedrms 20640 non-null float64  
4 Population 20640 non-null float64  
5 AveOccup 20640 non-null float64  
6 Latitude 20640 non-null float64  
7 Longitude 20640 non-null float64  
8 target 20640 non-null float64  
dtypes: float64(9)  
memory usage: 1.4 MB  
None

Dataframe statistics:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	\
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	
std	1.899822	12.585558	2.474173	0.473911	1132.462122	
min	0.499900	1.000000	0.846154	0.333333	3.000000	
25%	2.563400	18.000000	4.440716	1.006079	787.000000	
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	
max	15.000100	52.000000	141.909091	34.066667	35682.000000	

	AveOccup	Latitude	Longitude	target
count	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.070655	35.631861	-119.569704	2.068558
std	10.386050	2.135952	2.003532	1.153956
min	0.692308	32.540000	-124.350000	0.149990
25%	2.429741	33.930000	-121.800000	1.196000
50%	2.818116	34.260000	-118.490000	1.797000
75%	3.282261	37.710000	-118.010000	2.647250
max	1243.333333	41.950000	-114.310000	5.000010

NAs:

MedInc: 0  
HouseAge: 0  
AveRooms: 0  
AveBedrms: 0  
Population: 0  
AveOccup: 0  
Latitude: 0  
Longitude: 0  
target: 0

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

```
X = housing["data"]
y = housing["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y ,test_size = 0.2, random_state = 123)

lr_model = OLS(y_train, X_train)
rf_model = RandomForestRegressor()

lr_result = lr_model.fit()
rf_result = rf_model.fit(X_train, y_train)
```

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

```
lr_result.summary()
```

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.893			
Model:	OLS	Adj. R-squared (uncentered):	0.893			
Method:	Least Squares	F-statistic:	1.716e+04			
Date:	Wed, 18 Oct 2023	Prob (F-statistic):	0.00			
Time:	17:22:12	Log-Likelihood:	-19248.			
No. Observations:	16512	AIC:	3.851e+04			
Df Residuals:	16504	BIC:	3.857e+04			
Df Model:	8					
Covariance Type: nonrobust						
	coef	std err	t	P> t	[0.025	0.975]
x1	0.5168	0.005	107.955	0.000	0.507	0.526
x2	0.0153	0.001	29.508	0.000	0.014	0.016
x3	-0.1879	0.007	-27.171	0.000	-0.201	-0.174
x4	0.8780	0.033	26.472	0.000	0.813	0.943
x5	5.02e-06	5.71e-06	0.879	0.380	-6.18e-06	1.62e-05
x6	-0.0045	0.001	-7.845	0.000	-0.006	-0.003
x7	-0.0665	0.004	-16.580	0.000	-0.074	-0.059
x8	-0.0173	0.001	-13.550	0.000	-0.020	-0.015
Omnibus:	3585.633	Durbin-Watson:	1.996			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	12700.311			
Skew:	1.070	Prob(JB):	0.00			
Kurtosis:	6.726	Cond. No.	1.03e+04			

Notes:

[1] R² is computed without centering (uncentered) since the model does not contain a constant.

[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[3] The condition number is large, 1.03e+04. This might indicate that there are strong multicollinearity or other numerical problems.

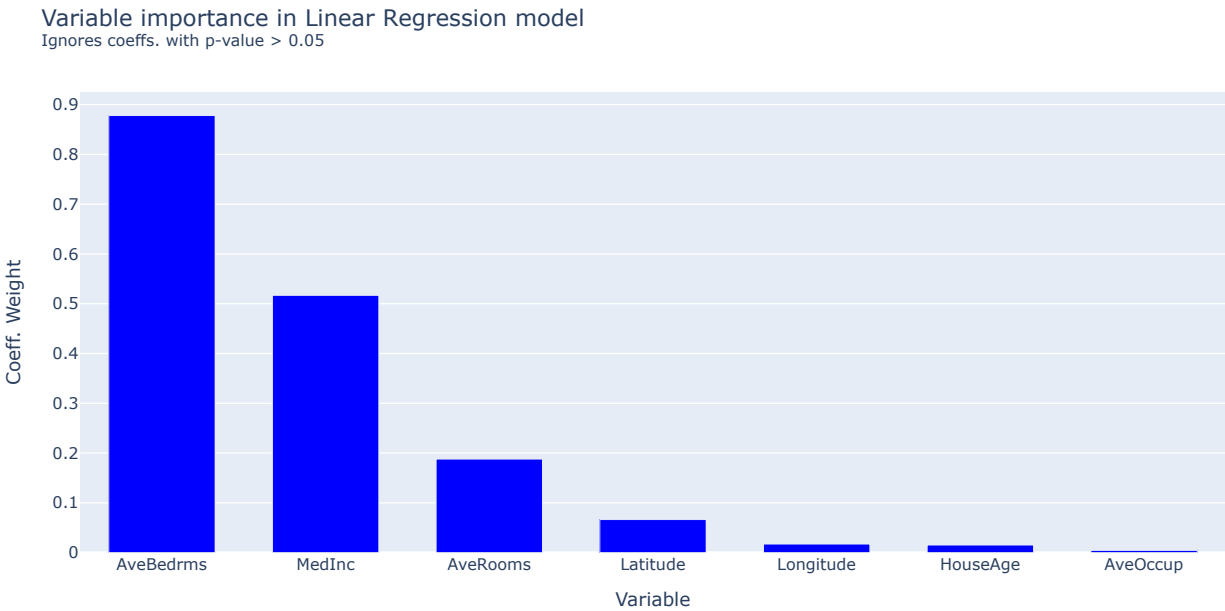
```
def plot_coeff_importance(df, title):
    coeff_importance_figure = go.Figure()
    feature_importance_trace = go.Bar(x=df["variable"], y=df["weight"], marker_color="blue", width=np.repeat(0.65, len(df[:])))
    coeff_importance_figure.add_trace(
        feature_importance_trace
    )
    coeff_importance_figure.update_layout(title=title, xaxis_title="Variable", yaxis_title="Coeff. Weight")
    coeff_importance_figure.show()
```

```
lr_coeff_importance = pd.DataFrame()
lr_coeff_importance["variable"] = housing["feature_names"]
lr_coeff_importance["weight"] = abs(lr_result.params)
lr_coeff_importance["pvalue"] = lr_result.pvalues

lr_coeff_importance = lr_coeff_importance[lr_coeff_importance["pvalue"] < 0.05]
lr_coeff_importance = lr_coeff_importance.sort_values(by="weight", ascending=False)
lr_coeff_importance.head()
```

	variable	weight	pvalue
3	AveBedrms	0.877981	2.883904e-151
0	MedInc	0.516778	0.000000e+00
2	AveRooms	0.187936	4.479569e-159
6	Latitude	0.066454	3.042622e-61
7	Longitude	0.017265	1.314939e-41

```
plot_coeff_importance(
    lr_coeff_importance,
    "Variable importance in Linear Regression model<br><sup>Ignores coeffs. with p-value > 0.05</sup>"
)
```

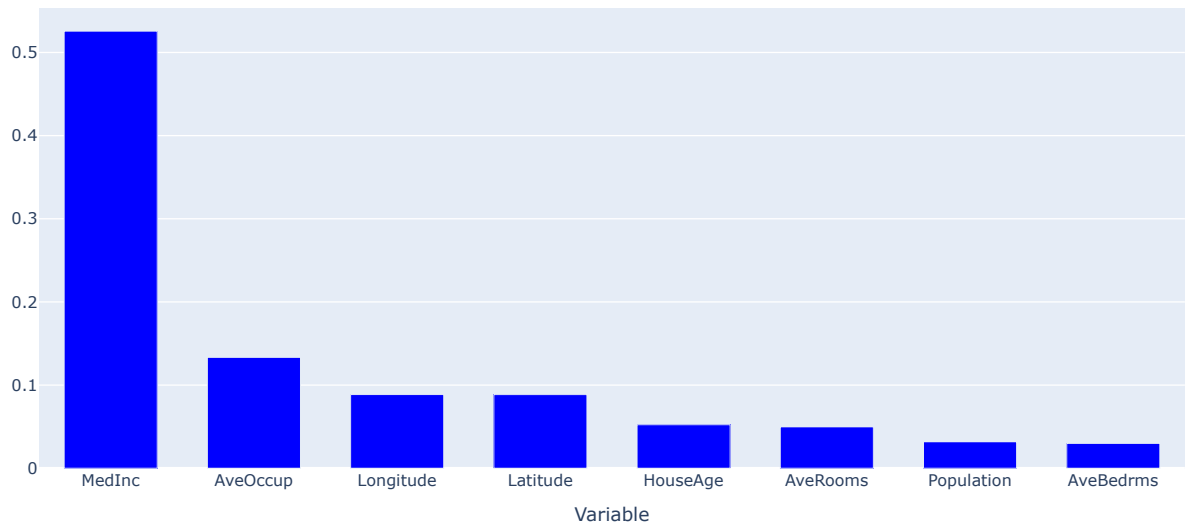


```
rf_coeff_importance = pd.DataFrame()
rf_coeff_importance["variable"] = housing["feature_names"]
rf_coeff_importance["weight"] = rf_model.feature_importances_
rf_coeff_importance = rf_coeff_importance.sort_values(by="weight", ascending=False)
rf_coeff_importance.head()
```

	variable	weight
0	MedInc	0.525620
5	AveOccup	0.133315
7	Longitude	0.088621
6	Latitude	0.088566
1	HouseAge	0.052345

```
plot_coeff_importance(rf_coeff_importance, "Variable importance in Random Forest model")
```

Variable importance in Random Forest model



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)**

```
lr_predictions = lr_result.predict(X_test)
rf_predictions = rf_model.predict(X_test)
```

```
lr_r2 = r2_score(y_test, lr_predictions)
rf_r2 = r2_score(y_test, rf_predictions)
```

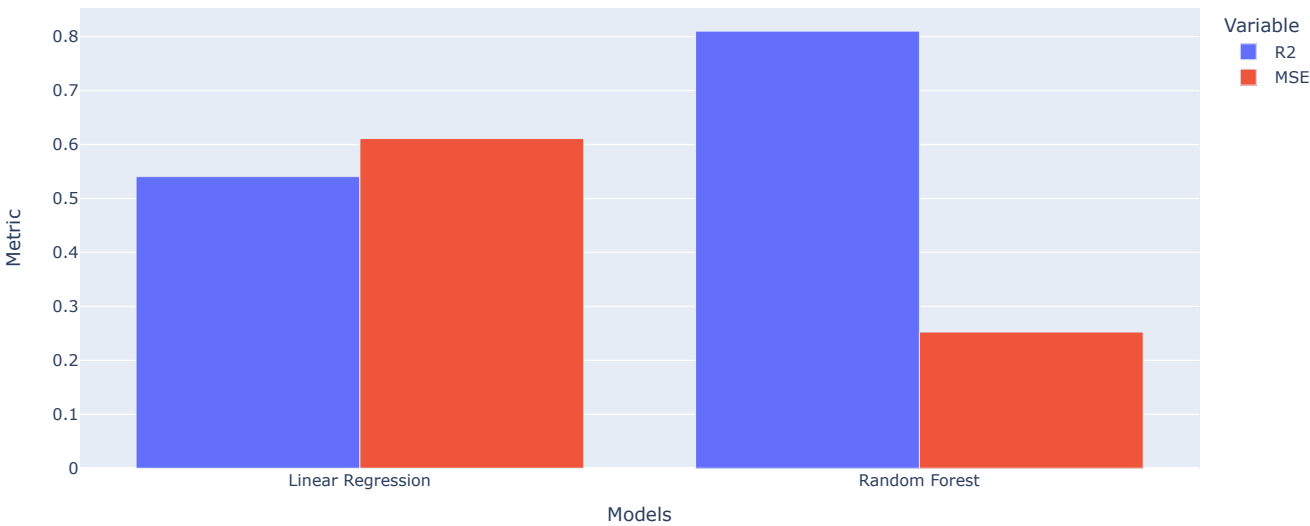
```
lr_mse = mean_squared_error(y_test, lr_predictions)
rf_mse = mean_squared_error(y_test, rf_predictions)
```

```
df_metrics = pd.DataFrame()
df_metrics["R2"] = [lr_r2, rf_r2]
df_metrics["MSE"] = [lr_mse, rf_mse]
```

```
df_metrics.index = ["Linear Regression", "Random Forest"]
```

```
fig = px.bar(
    df_metrics,
    barmode="group",
    title="Permformance Linear Regression vs. Random Forest",
    labels={
        "value": "Metric",
        "index": "Models",
        "variable": "Variable"
    }
)
fig.show()
```

Permformance Linear Regression vs. Random Forest



En el Random Forest el R2 es mayory el MSE es menor que en el modelo de Linear Regression, lo que lo hace un modelo superior.

▼ 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 signitficativa 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)**

```
spotify = pd.read_csv("./spotify-2023.csv", encoding="ISO-8859-1")
spotify["streams"] = pd.to_numeric(spotify["streams"], errors="coerce")
spotify["in_deezer_playlists"] = pd.to_numeric(spotify["in_deezer_playlists"], errors="coerce")
spotify["in_shazam_charts"] = pd.to_numeric(spotify["in_shazam_charts"], errors="coerce")
spotify = spotify.dropna()
describe_df(spotify)
```

	in_deezer_charts	in_shazam_charts	bpm	danceability_%	\
count	748.000000	748.000000	748.000000	748.000000	
mean	2.368984	48.909091	123.143048	67.592246	
std	5.316258	124.329225	28.333491	14.585603	
min	0.000000	0.000000	65.000000	23.000000	
25%	0.000000	0.000000	100.000000	58.000000	
50%	0.000000	2.000000	121.000000	70.000000	
75%	1.000000	33.000000	142.000000	79.000000	
max	45.000000	953.000000	206.000000	96.000000	

	valence_%	energy_%	acousticness_%	instrumentalness_%	liveness_%	\
count	748.000000	748.000000	748.000000	748.000000	748.000000	
mean	51.129679	64.137701	26.700535	1.681818	18.304813	
std	23.607864	15.933631	25.190809	8.838448	13.722069	
min	4.000000	14.000000	0.000000	0.000000	3.000000	
25%	32.000000	53.000000	6.000000	0.000000	10.000000	
50%	51.000000	65.000000	18.000000	0.000000	12.000000	
75%	70.000000	76.000000	41.250000	0.000000	24.000000	
max	97.000000	97.000000	97.000000	91.000000	97.000000	

	speechiness_%
count	748.000000
mean	10.711230
std	10.332208
min	2.000000
25%	4.000000
50%	6.000000
75%	12.000000
max	64.000000

NAs:

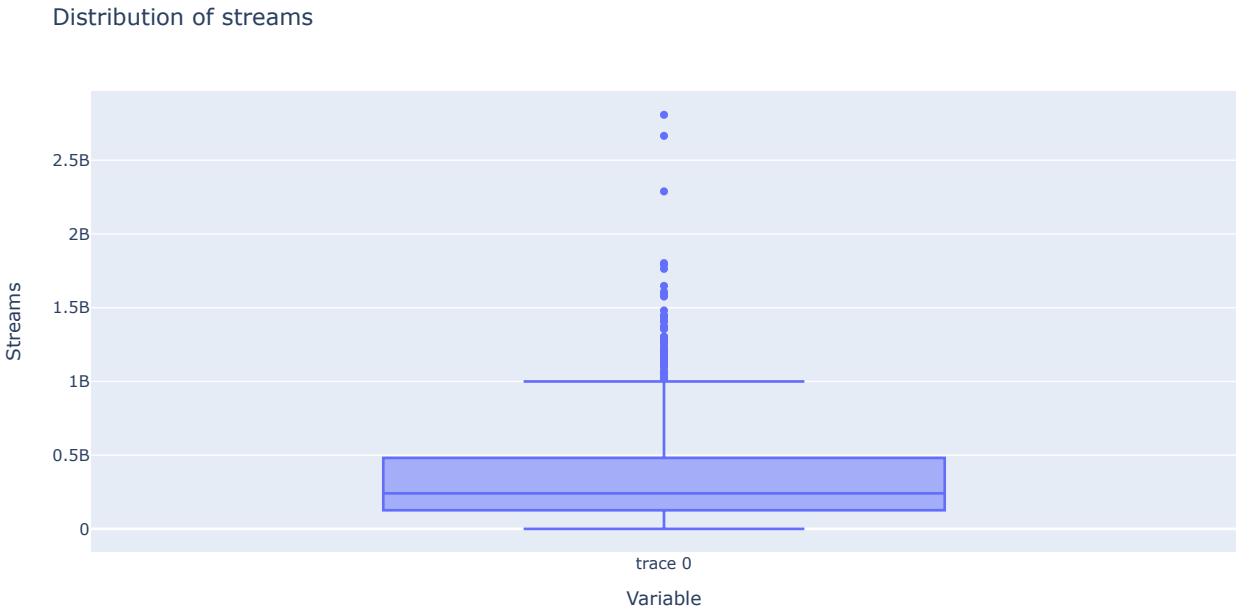
```
track_name: 0
artist(s)_name: 0
artist_count: 0
released_year: 0
released_month: 0
released_day: 0
in_spotify_playlists: 0
in_spotify_charts: 0
streams: 0
in_apple_playlists: 0
in_apple_charts: 0
in_deezer_playlists: 0
in_deezer_charts: 0
in_shazam_charts: 0
bpm: 0
key: 0
mode: 0
danceability_%: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

- b) ¿Cuántos artistas únicos hay? **(0.25 puntos)**
- c) ¿Cuál es la distribución de reproducciones? **(0.5 puntos)**

```
unique_artists = len(spotify["artist(s)_name"].unique())
print("Number of unique artists: " + str(unique_artists))
```

Number of unique artists: 531

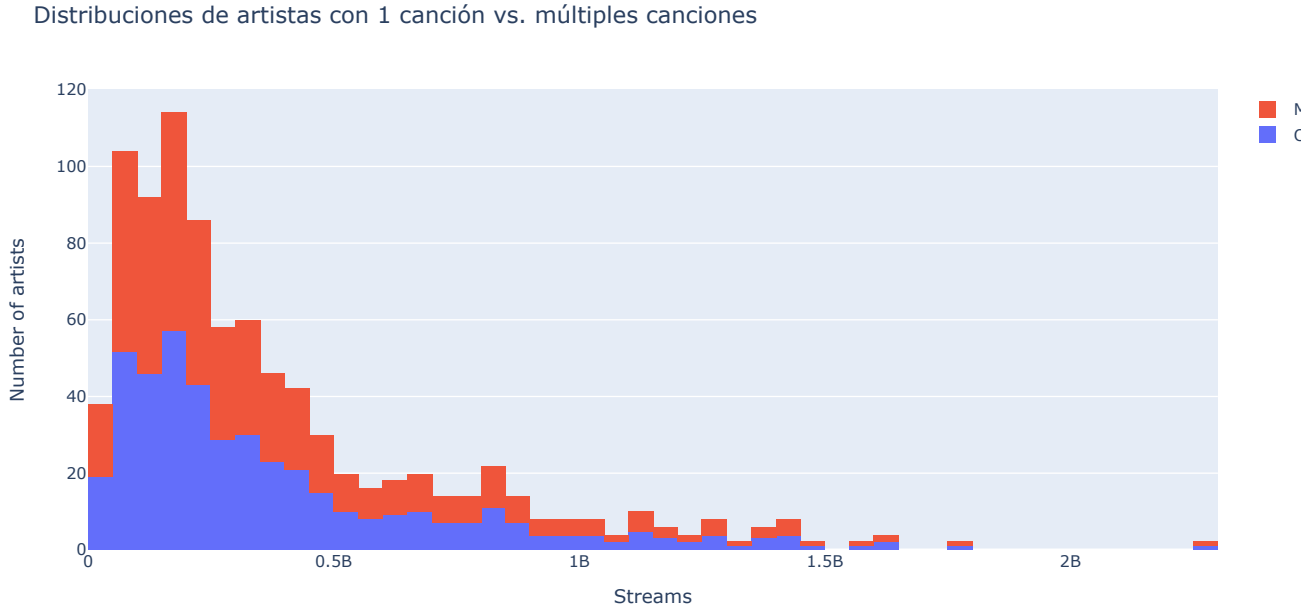
```
reproduction_distribution_trace = go.Box(
    y=spotify["streams"],
)
reproduction_distribution = go.Figure(
    reproduction_distribution_trace
)
reproduction_distribution.update_layout(title="Distribution of streams", xaxis_title="Variable", yaxis_title="Streams")
reproduction_distribution.show()
```



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

```
one_songs = spotify[spotify["artist_count"] == 1]
multiple_songs = spotify[spotify["artist_count"] != 1]

fig = go.Figure(
)
song_count_traces = [
    go.Histogram(
        x=one_songs["streams"],
        name="One song",
    ),
    go.Histogram(
        x=multiple_songs["streams"],
        name="Multiple songs",
    ),
]
fig.add_traces(song_count_traces)
fig.update_layout(title="Distribuciones de artistas con 1 canción vs. múltiples canciones",barmode='stack', yaxis_title="Number of artists", xaxis_title="Streams")
fig.show()
```



No hay una diferencia significativa entre ambas

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

```
spotify_numbers = spotify.select_dtypes(include='number')

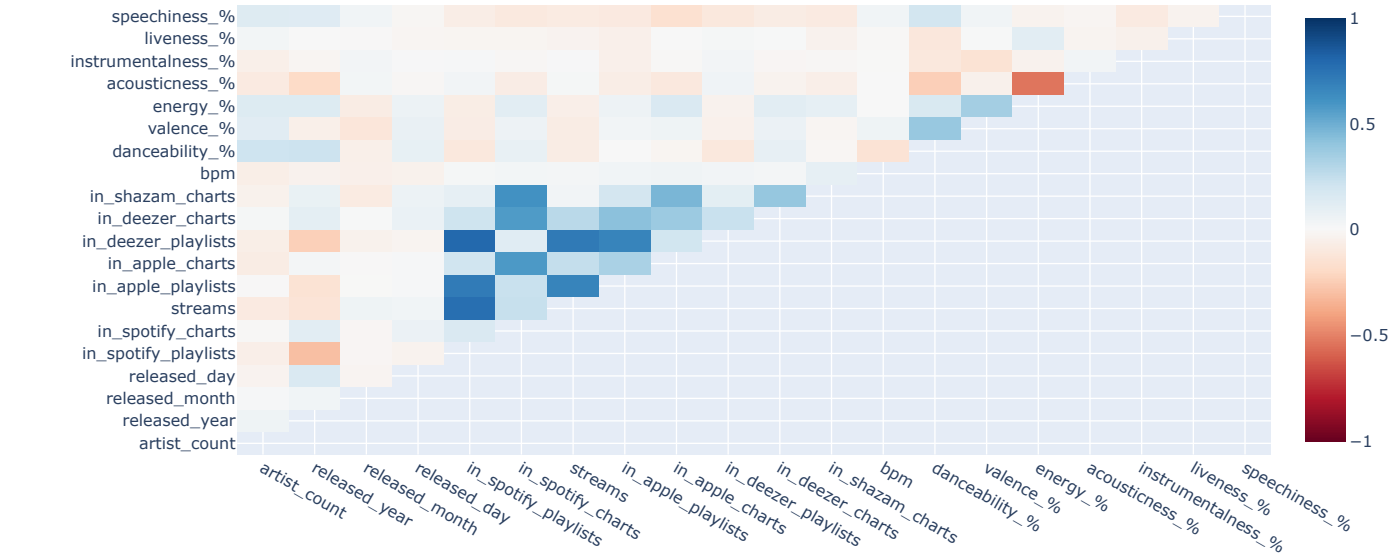
stream_corr = spotify_numbers[spotify_numbers.columns].corr()
stream_corr.head()
```

```

    artist count  released year  released month  released day  in spotify playlists  in spotify charts  streams  in apple playlists  in apple charts  in deezer playlis
mask = np.triu(np.ones_like(stream_corr, dtype=bool))
}

stream_corr_heatmap = go.Heatmap(
    z=stream_corr.mask(mask),
    x=stream_corr.columns,
    y=stream_corr.columns,
    colorscale=px.colors.diverging.RdBu,
    zmin=-1,
    zmax=1
)

corr_matrix = go.Figure(
    stream_corr_heatmap
)
corr_matrix.show()
```



The variables that most correlate with streams are:

- In spotify playlists
- In apple playlists
- In apple charts
- In spotify charts
- In deezer playlists
- In deezer charts

(The most near to 1, and -1 of the corr matrix)

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

```

popular_cutoff = spotify["streams"].quantile(q=0.5)
print(popular_cutoff)

popular_songs = spotify[spotify["streams"] > popular_cutoff]
popular_songs.size

240844044.5
8976

popular_songs.columns

Index(['track_name', 'artist(s)_name', 'artist_count', 'released_year',
      'released_month', 'released_day', 'in_spotify_playlists',
      'in_spotify_charts', 'streams', 'in_apple_playlists', 'in_apple_charts',
      'in_deezer_playlists', 'in_deezer_charts', 'in_shazam_charts', 'bpm',
      'key', 'mode', 'danceability_%', 'valence_%', 'energy_%',
      'acousticness_%', 'instrumentalness_%', 'liveness_%', 'speechiness_%'],
      dtype='object')

# Remove un-dummiabale variables
popular_songs_dataset = popular_songs.drop(["track_name", "artist(s)_name"], axis=1)
popular_songs_dataset.columns

dummies = pd.get_dummies(popular_songs_dataset[["key", "mode"]], prefix="dummy")
popular_songs_dataset = popular_songs_dataset.drop(["key", "mode"], axis=1)
popular_songs_dataset = pd.concat([
    popular_songs_dataset,
    dummies
], axis=1)

x_variables = list(popular_songs_dataset.columns)
x_variables.remove("streams")

# Divide data in X,Y
X_popular_songs = popular_songs_dataset[x_variables]
y_popular_songs = popular_songs_dataset["streams"]

X_popular_songs.head()
```

	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_charts	in_apple_playlists	in_apple_charts	in_deezer_playlists	in_deezer_charts	..
3	1	2019	8	23	7858	100	116	207	125.0	12	
4	1	2023	5	18	3133	50	84	133	87.0	15	
6	2	2023	3	16	3090	50	34	222	43.0	13	
9	2	2023	3	17	2953	44	49	110	66.0	13	
10	2	2023	4	17	2876	40	41	205	54.0	12	
5 rows x 32 columns											

```
spotify_model = RandomForestRegressor(500)
spotify_model.fit(X_popular_songs, y_popular_songs)
```

▼

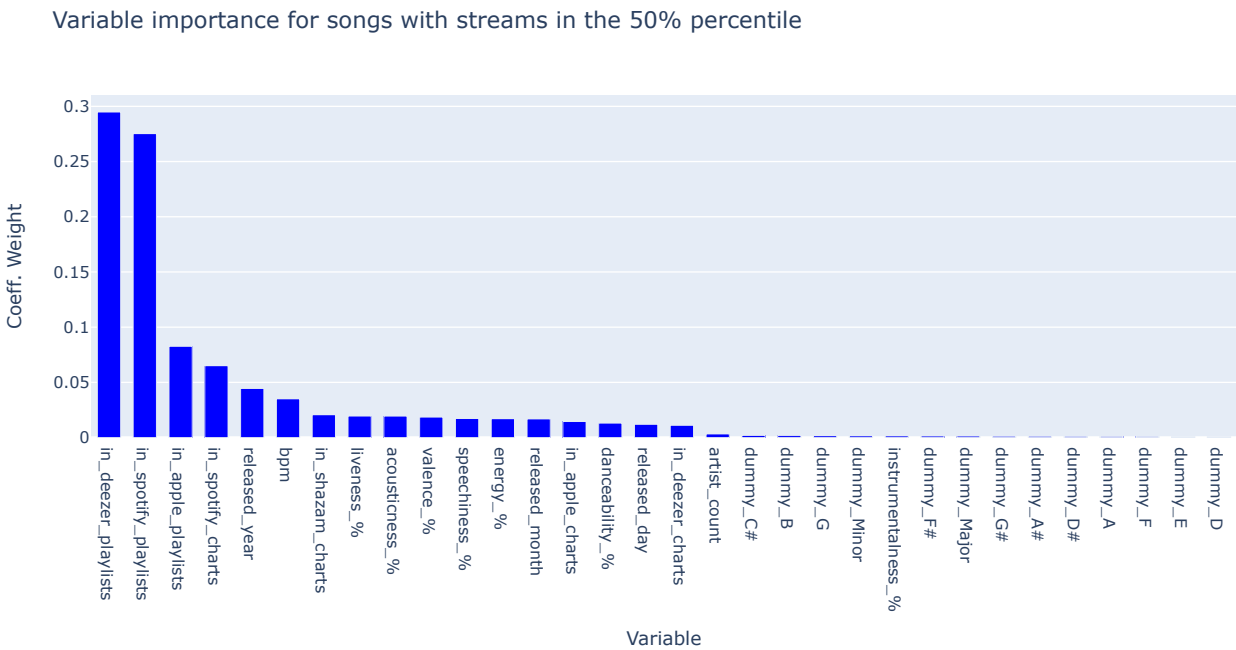
RandomForestRegressor

RandomForestRegressor(n\_estimators=500)

```
popular_songs_coeff_importance = pd.DataFrame()
popular_songs_coeff_importance["variable"] = X_popular_songs.columns
popular_songs_coeff_importance["weight"] = spotify_model.feature_importances_
popular_songs_coeff_importance = popular_songs_coeff_importance.sort_values(by="weight", ascending=False)
popular_songs_coeff_importance.head()
```

	variable	weight	
8	in_deezer_playlists	0.294879	
4	in_spotify_playlists	0.275302	
6	in_apple_playlists	0.082658	
5	in_spotify_charts	0.065120	
1	released_year	0.044497	

```
plot_coeff_importance(popular_songs_coeff_importance, "Variable importance for songs with streams in the 50% percentile")
```



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

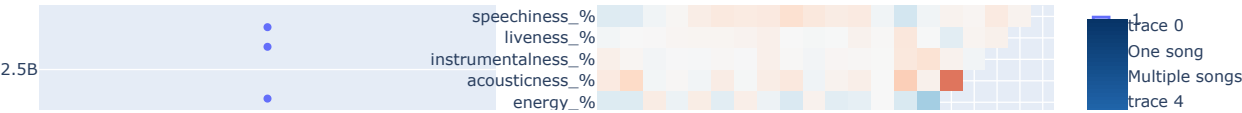
```
fig = make_subplots(rows=2, cols=2)

feature_importance_trace = go.Bar(
    x=popular_songs_coeff_importance["variable"],
    y=popular_songs_coeff_importance["weight"],
    marker_color="blue",
    width=np.repeat(0.65, len(popular_songs_coeff_importance[:]))
)

fig.add_trace(reproduction_distribution_trace, row=1, col=1)
fig.add_trace(stream_corr_heatmap, row=1, col=2)
fig.add_traces(song_count_traces, rows=2, cols=1)
fig.add_trace(feature_importance_trace, row=2, col=2)
fig.update_layout(width=1000, height=1000, bargap=0.1, title="Dash")
fig.show()
```



Dash



El éxito de la canción está muy correlado con el hecho de que se publique en las principales plataformas (Deezer, Spotify, Apple).

También es importante que sea algo acústica, enérgica y que se pueda bailar.

Y no hay una diferencia significativa entre sacar una canción o sacar múltiples (en cuanto al número de reproducciones).

