Universidad Nacional Autonoma de México

Por mi raza hablara el espiritu

Facultad de Estudios Superior de Acatlan
Matematicas Aplicadas Y Computación

Identificación de Patrones de Éxito Musical en Spotify mediante Análisis de Datos y Clasificación Predictiva

Solis Vilchis Roberto Atonatiuh
Ortiz Garcia Miguel Angel

Índice general

1. Resumen Ejecutivo (Abstract)

Este proyecto presenta un análisis y modelación de datos sobre el éxito de canciones en plataformas de streaming como Spotify, Apple Music, Shazam y Deezer. El objetivo principal es identificar las características que influyen en el éxito de las canciones y predecir su popularidad utilizando diversas técnicas de análisis y aprendizaje supervisado.

Inicialmente, se realiza una exploración de datos para entender mejor las características musicales como el BPM (beats per minute), la danzabilidad y otros atributos relevantes. A continuación, se aplica el Análisis de Componentes Principales (PCA) para la reducción de dimensiones, permitiendo simplificar los datos sin perder información crítica.

Se implementan técnicas de clustering para categorizar las canciones en grupos según su potencial de éxito. Posteriormente, se utilizan varios modelos de clasificación supervisada, incluyendo Regresión Logística, Random Forest, Redes Neuronales, Análisis Discriminante y Support Vector Machine, para predecir la categoría de éxito de cada canción.

Para evaluar el rendimiento de los modelos, se emplean métricas como la precisión, la recall, el F1-score y el área bajo la curva ROC (ROC-AUC). Estos análisis permiten determinar cuál es el modelo más eficaz para la clasificación de canciones.

Los resultados de este estudio pueden ayudar a identificar canciones con alto potencial de éxito en las diferentes plataformas de streaming, ofreciendo valiosas recomendaciones tanto para productores musicales como para plataformas de streaming en la toma de decisiones estratégicas.

2. Introducción

Vivimos en una época donde la música se ha vuelto cada vez más relevante. En un mundo caracterizado por altos niveles de estrés, la música ofrece un medio para relajarse y aliviar esos momentos de tensión. Por esta razón, las plataformas de streaming han proliferado, convirtiéndose en medios cruciales para la difusión de canciones.

Dada la importancia de la música, surge la pregunta de cómo saber si una canción será relevante o, como ocurre con la mayoría, pasará desapercibida sin alcanzar el éxito. Este proyecto busca abordar esta cuestión utilizando técnicas de machine learning para predecir la posibilidad de que una canción

tenga éxito en alguna plataforma de streaming.

El objetivo principal de este proyecto es desarrollar un modelo que pueda predecir el éxito de una canción, proporcionando así una herramienta útil para discográficas y estrategias de marketing en plataformas como Spotify, Apple Music y otras. Además, el análisis de los datos permitirá observar cómo varían los rankings de popularidad entre diferentes plataformas.

Para alcanzar este objetivo, es fundamental seguir un camino metodológico claro y riguroso. Este proyecto se divide en varias etapas: exploración de datos, análisis, reducción de dimensiones, clustering, modelación con algoritmos de clasificación y evaluación de los modelos creados. Cada una de estas etapas será explicada en detalle a lo largo del documento.

Este documento está organizado de la siguiente manera: primero, se describe el análisis de los datos y las características musicales utilizadas. Luego, se detalla la metodología empleada para la reducción de dimensiones y el clustering. Posteriormente, se presentan los modelos de clasificación utilizados y sus evaluaciones. Finalmente, se discuten los resultados y se presentan las conclusiones y posibles trabajos futuros.

3. Preparación de Datos (Data Preparation)

Nuestro conjunto de datos proviene de la plataforma Kaggle y contiene información sobre las canciones en Spotify. El dataset se puede encontrar en Kaggle. Incluye datos sobre las canciones más populares del año 2023, así como de años anteriores como 2022 y 2021. Además, contiene el número de reproducciones y la cantidad de listas de reproducción en otras plataformas donde se encuentra cada canción.

El conjunto de datos se carga utilizando la biblioteca pandas y se realiza una descripción inicial de las variables.

```
import pandas as pd

data = pd.read_csv('./spotify-2023.csv', encoding='ISO-8859-1')

print(data.info())
print(data.describe())
```

Listing 1: Carga de datos y descripción inicial

Para entender mejor las variables presentes en nuestro conjunto de datos, se ha creado un diccionario de datos que describe el significado y el tipo de cada columna. La siguiente tabla muestra las variables, sus significados y sus categorías.

Variable	Descripción	Tipo
track_name	Nombre de la canción	Categórica
artist(s)_name	Nombre del/los artista(s) de la canción	Categórica
artist_count	Número de artistas que contribuyen a la canción	Numérica
released_year	Año de lanzamiento de la canción	Numérica
released_month	Mes de lanzamiento de la canción	Numérica
released_day	Día del mes de lanzamiento de la canción	Numérica
in_spotify_playlists	Número de listas de reproducción de Spotify en las que está incluida la canción	Numérica
in_spotify_charts	Presencia y rango de la canción en las listas de Spotify	Numérica
streams	Número total de reproducciones en Spotify	Numérica
in_apple_playlists	Número de listas de reproducción de Apple Music en las que está incluida la canción	Numérica
in_apple_charts	Presencia y rango de la canción en las listas de Apple Music	Numérica
in_deezer_playlists	Número de listas de reproducción de Deezer en las que está incluida la canción	Numérica
in_deezer_charts	Presencia y rango de la canción en las listas de Deezer	Numérica
in_shazam_charts	Presencia y rango de la canción en las listas de Shazam	Numérica
bpm	Beats por minuto, una medida del tempo de la canción	Numérica
key	Clave de la canción	Categórica
mode	Modo de la canción (mayor o menor)	Categórica
danceability %	Porcentaje que indica cuán adecuada es la canción para bailar	Numérica
valence %	Positividad del contenido musical de la canción	Numérica
energy %	Nivel de energía percibido de la canción	Numérica
acousticness %	Cantidad de sonido acústico en la canción	Numérica
instrumentalness %	Cantidad de contenido instrumental en la canción	Numérica
liveness %	Presencia de elementos de actuación en vivo	Numérica
speechiness %	Cantidad de palabras habladas en la canción	Numérica

Cuadro 1: Diccionario de datos

La limpieza de datos es crucial para garantizar la calidad y precisión de los análisis posteriores. Incluye la identificación y corrección de valores faltantes, incorrectos y atípicos.

1. Valores Incorrectos Se identificó un error en la columna streams, donde debería haber un valor numérico pero se encontró la cadena:

'BPM110KeyAModeMajorDanceability53Valence75Energy69Acousticness7 Instrumentalness0Liveness17Speechiness3'. Este valor se cambió a NaN (valor faltante). Tambien se identificaron errores en el registro de datos en ciertas columnas, como el agregar coma ",.[en] valores que deberian ser solo numericos.

```
vi = spo[spo['streams'] == '
    BPM110KeyAModeMajorDanceability53Valence75
    Energy69Acousticness7Instrumentalness0
    Liveness17Speechiness3'].index[0]

spo.loc[vi, ['streams']] = np.nan

spo['streams'] = spo['streams'].astype('Int64')

spo['in_deezer_playlists'] = spo['
    in_deezer_playlists'].astype(str)

spo['in_deezer_playlists'] = spo['
    in_deezer_playlists'].apply(lambda x: int(x.
    replace(',', '')))

spo['in_deezer_playlists'] = spo['
    in_deezer_playlists'].astype(int)

spo['in_shazam_charts'] = spo['in_shazam_charts'].
    astype(str)

spo['in_shazam_charts'].replace('nan', np.nan,
    inplace=True)

spo['in_shazam_charts'] = spo['in_shazam_charts'].
    apply(lambda x: int(str(x).replace(',', '')) if
     pd.notnull(x) else np.nan )

spo['in_shazam_charts'] = spo['in_shazam_charts'].
    astype('Int64')
```

Listing 2: Valores incorrectos

2. Valores Faltantes Se encontraron valores faltantes en las siguientes columnas:

```
nulos = spo.isnull().sum()
print(f'Columnas con al menos un nulo: \n{nulos[
    nulos > 0]}')
```

Listing 3: Valores nulos

```
Columnas con al menos un nulo:
streams              1
```

```
in_shazam_charts    50
key                 95
dtype: int64
```

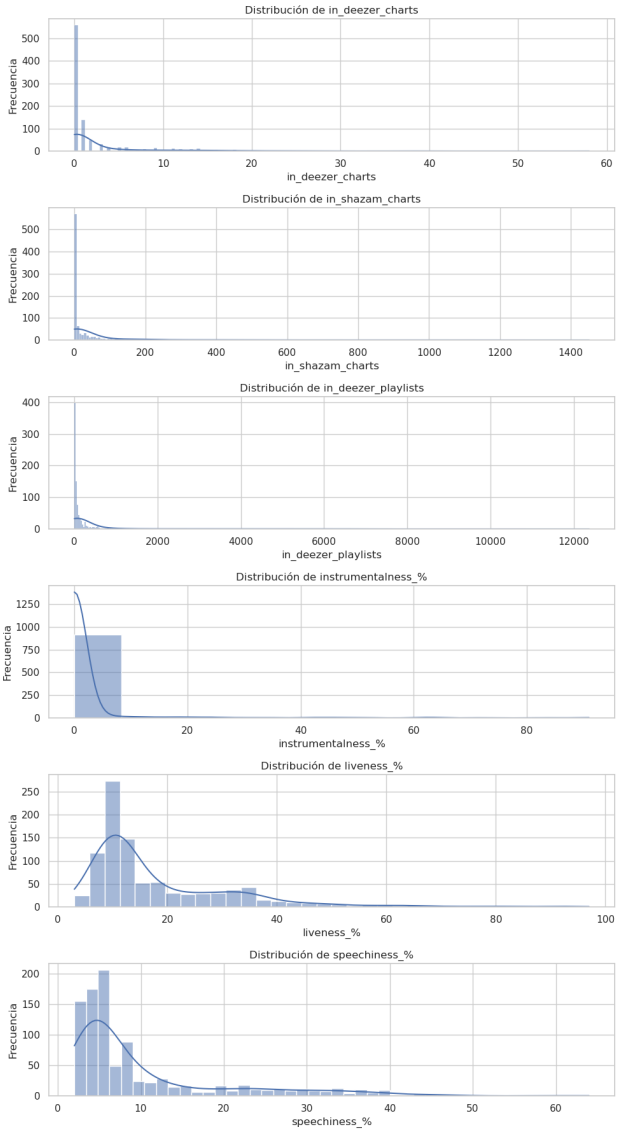
Para imputar estos valores faltantes, se utilizaron las siguientes técnicas:

- streams: Mediana
- in_shazam_charts: Media
- key: Moda

```
imputar_moda_cat = SimpleImputer(strategy='most_frequent')

spo['key'] = imputar_moda_cat.fit_transform(spo[['key']]).ravel()

spo['key'] = spo['key'].astype(str)

imput_streams = SimpleImputer(strategy='mean')

spo['streams'] = imput_streams.fit_transform(spo[['streams']])

imput_shazam_charts = SimpleImputer(strategy='median')

spo['in_shazam_charts'] = imput_shazam_charts.fit_transform(spo[['in_shazam_charts']])
```

Listing 4: Metodos de imputacion

3. Valores Atípicos (Outliers) Para identificar valores atípicos, se utilizaron gráficos de distribución. Las columnas con valores potencialmente atípicos fueron tratadas con técnicas de Capping y Flooring.



Estas técnicas se aplicaron a las siguientes columnas:

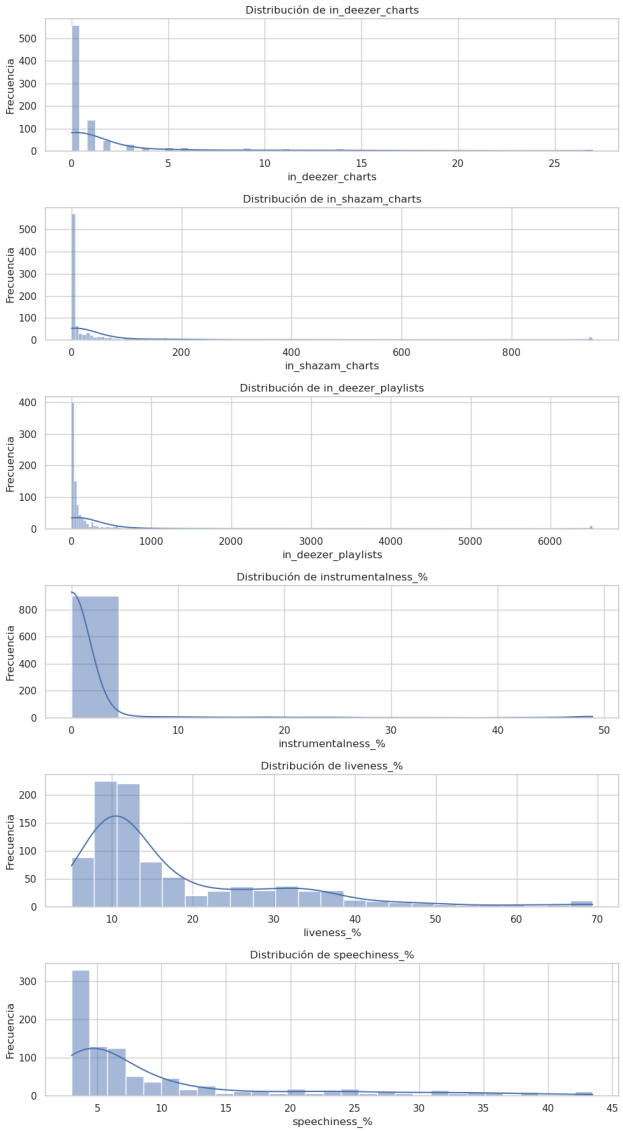
- in_deezer_charts
- in_shazam_charts
- in_deezer_playlists
- instrumentalness_
- liveness_
- speechiness_

```
        change_column = ['in_deezer_charts', '
            in_shazam_charts', 'in_deezer_playlists', '
            instrumentalness_%', 'liveness_%', '
            speechiness_%']

        def cap_floor(df, column, lower_quantile=0.01,
            upper_quantile=0.99):
            lower_bound = df[column].quantile(
                lower_quantile)
            upper_bound = df[column].quantile(
                upper_quantile)
            df[column] = np.where(df[column] < lower_bound
                , lower_bound, df[column])
            df[column] = np.where(df[column] > upper_bound
                , upper_bound, df[column])
            return df

        for column in change_column:
            spo = cap_floor(spo, column)
```

Listing 5: Tratamiento de valores atipicos

```
        sns.set(style="whitegrid")

        num_columns = len(change_column)
        fig, axes = plt.subplots(num_columns, 1, figsize
            =(10, 3 * num_columns))

        for i, column in enumerate(change_column):
            ax = axes[i] if num_columns > 1 else axes
            if pd.api.types.is_numeric_dtype(spo[column]):
                sns.histplot(spo[column], kde=True, ax=ax)
                ax.set_xlabel(column)
                ax.set_ylabel('Frecuencia')
            else:
                sns.countplot(y=spo[column], ax=ax)
                ax.set_xlabel('Frecuencia')
                ax.set_ylabel(column)
            ax.set_title(f'Distribuci n de {column}')

    plt.tight_layout()
    plt.show()
```

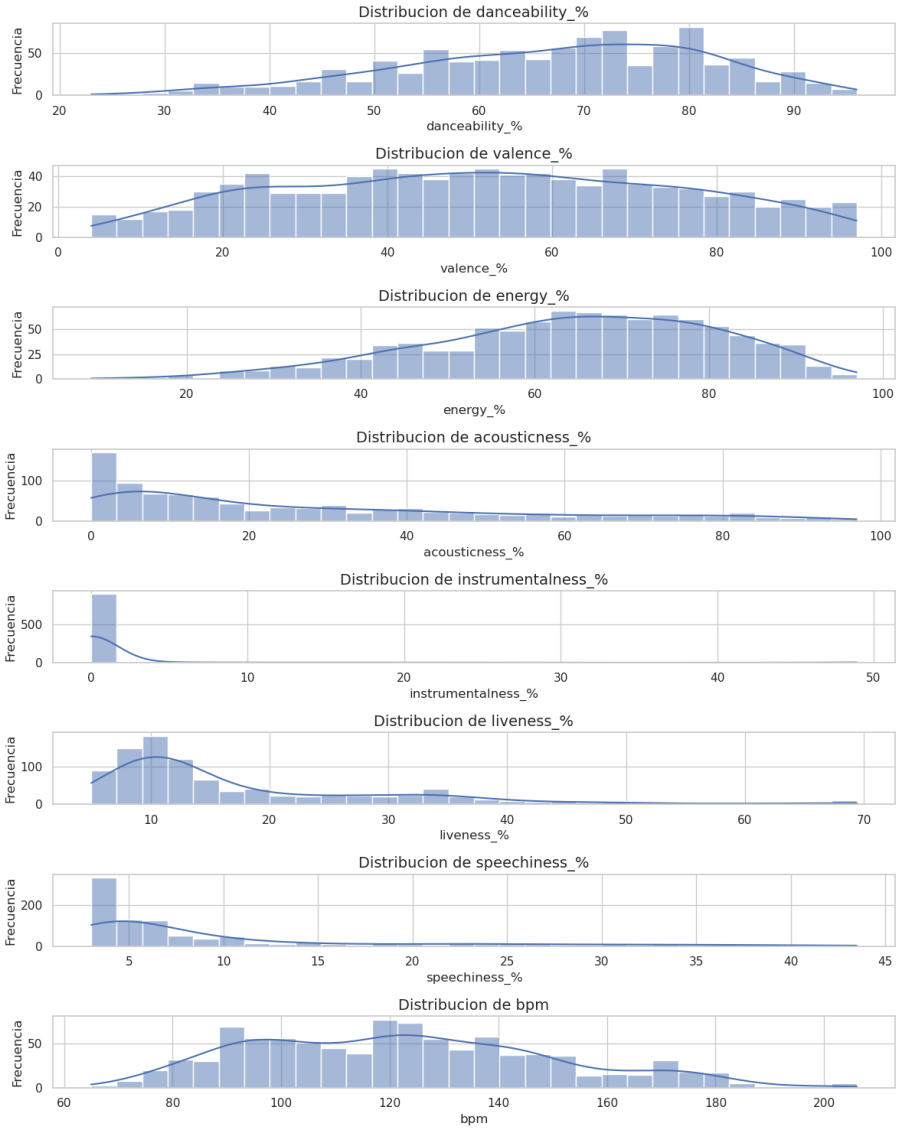
Listing 6: Codigo para las visualizaciones



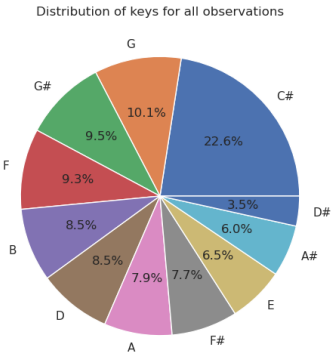
Al comparar las gráficas, podemos notar un cambio significativo en varias de ellas, pero también observamos que algunas se mantienen similares. Es evidente que el uso de la técnica de capping y flooring resulta muy útil para tratar valores atípicos, lo que nos permite continuar con un análisis más detallado de los datos.

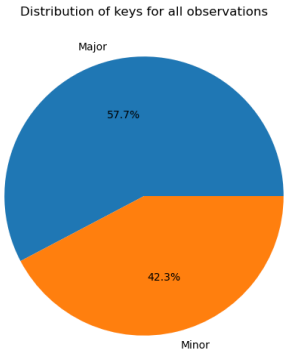
4. Análisis de Datos (Data Analysis)

En esta sección, analizamos las distribuciones de las variables que representan los atributos de las canciones, como el porcentaje de energía, dancibilidad, entre otros. El objetivo es entender el comportamiento de estas variables y sus tipos de distribución, considerando que los datos ya han sido tratados.

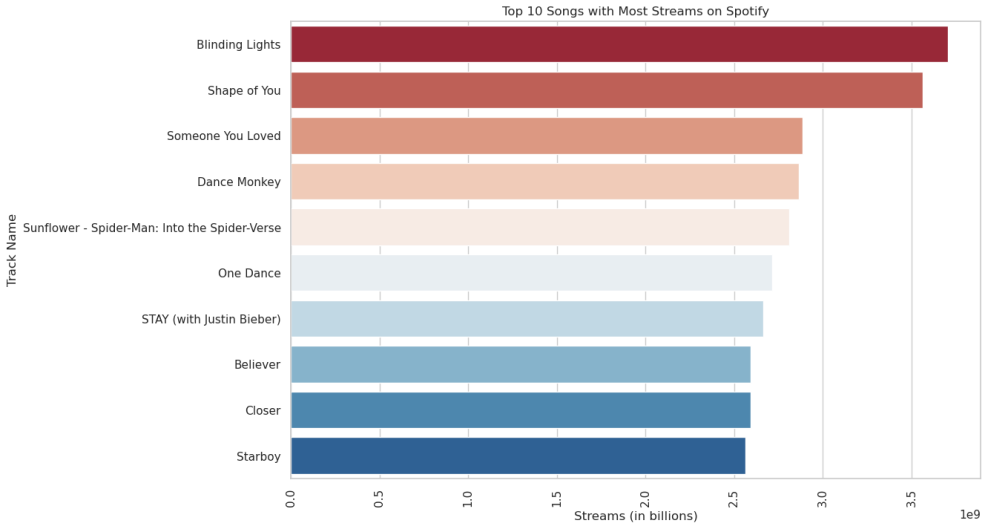


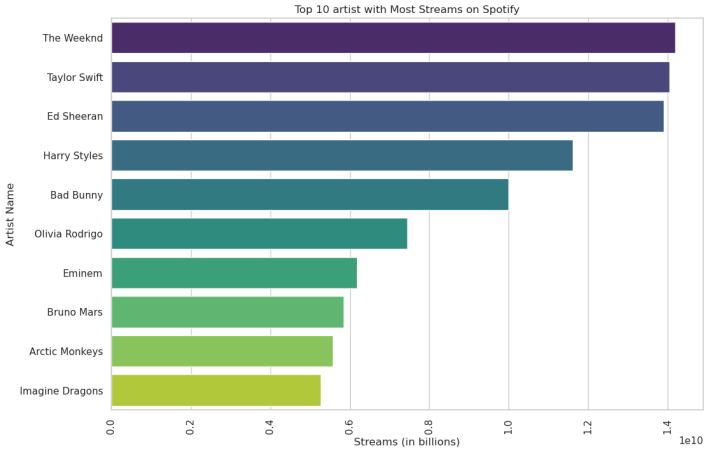
Además de las variables continuas, también se analizaron las variables discretas, particularmente 'key' y 'mode'. Se utilizaron gráficas de pastel para visualizar los porcentajes de cada valor único, proporcionando una visión clara de la distribución de estas variables.





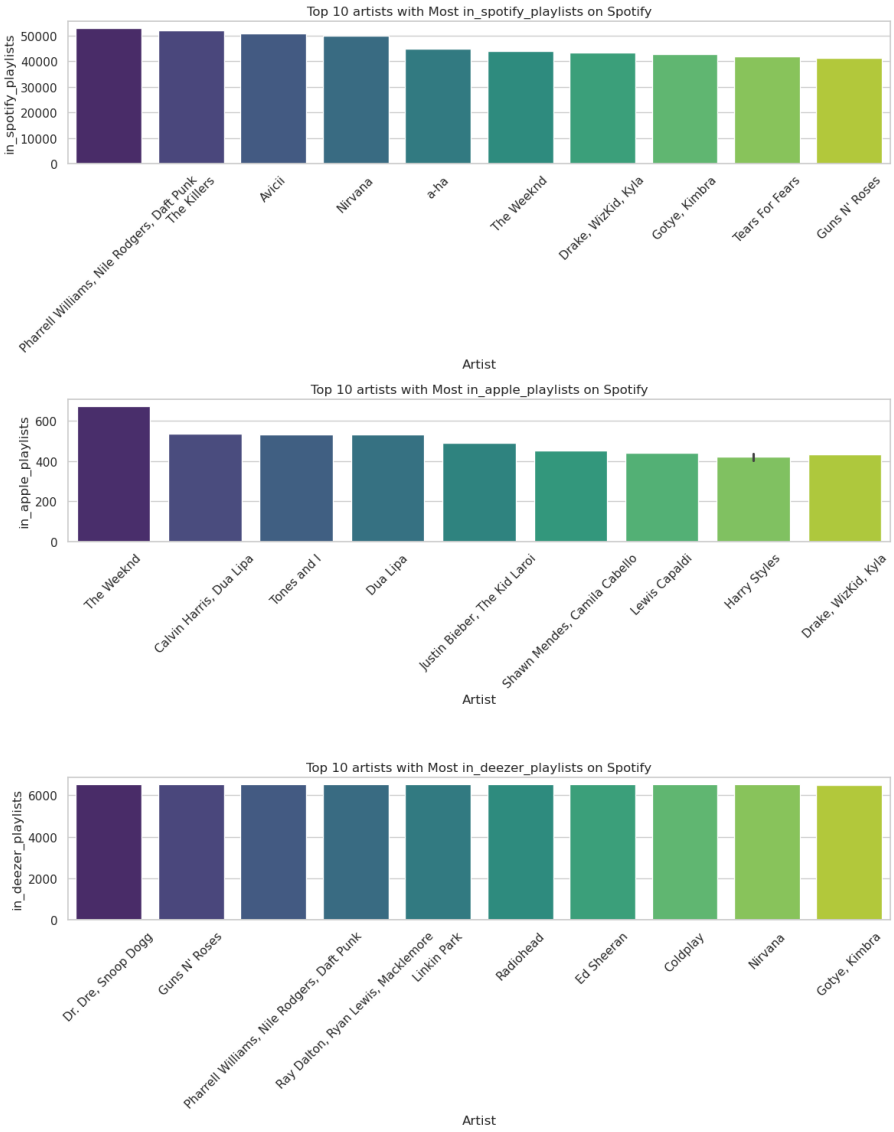
Conociendo las frecuencias y porcentajes de las variables, procedemos a analizar las variables relacionadas con streams, playlists, charts, artistas y canciones. Esto nos permitirá identificar a los mejores artistas, las mejores canciones en diferentes plataformas, entre otros aspectos destacados.



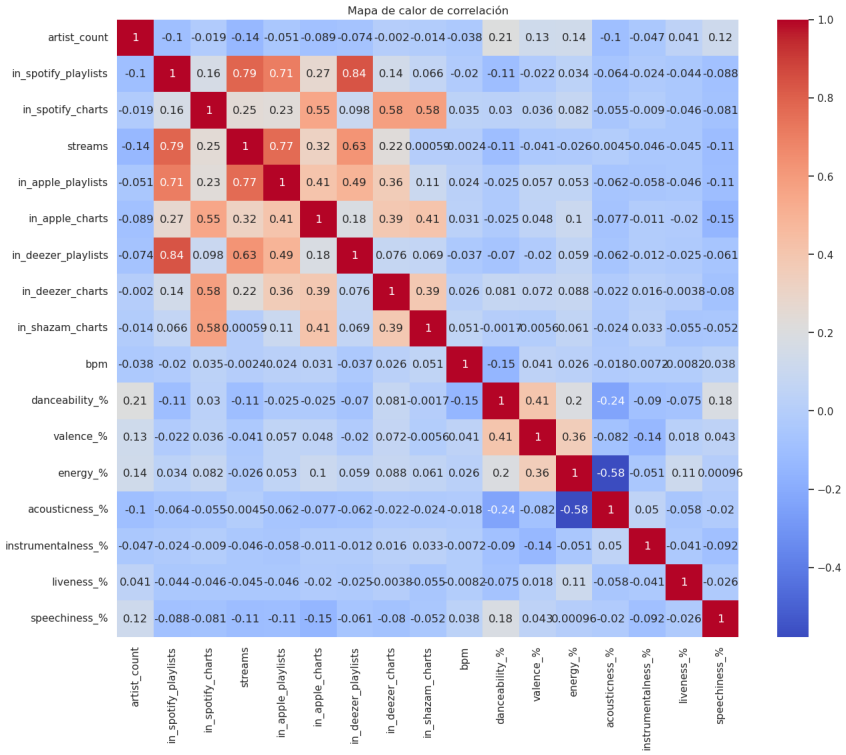


Algunos artistas muestran consistencia en su popularidad a través de múltiples plataformas. Por ejemplo, The Weeknd aparece frecuentemente en las listas de reproducción tanto de Spotify como de Apple Music.

Existen diferencias notables entre las plataformas. Artistas que son populares en Spotify pueden no serlo en Deezer y viceversa. Esto puede deberse a las diferencias en la base de usuarios y algoritmos de recomendación de cada plataforma.



Finalmente, examinamos las correlaciones entre las columnas utilizando mapas de calor. El mapa de calor de correlaciones proporciona una representación visual de las relaciones entre las variables del conjunto de datos. En este mapa, cada celda representa el coeficiente de correlación entre dos variables, donde un valor de 1 indica una correlación positiva perfecta, -1 indica una correlación negativa perfecta, y 0 indica que no hay correlación lineal entre las variables.



1. Correlación Alta:

   Las variables relacionadas con playlists y charts en las diferentes plataformas tienen correlaciones significativamente altas. Por ejemplo, in_spotify_playlists tiene una alta correlación con in_spotify_charts (0.79) y in_apple_playlists (0.71). Esto sugiere que las canciones que son populares en una plataforma tienden a ser populares en otras también.

2. Correlación Moderada:

   streams tiene una correlación moderada con in_spotify_charts (0.63) y in_apple_playlists (0.77), indicando que el número de reproducciones está relacionado con la aparición en listas de éxitos. Las características de la canción como energy_% y acousticness_% tienen una correlación inversa moderada (-0.58), sugiriendo que las canciones con alta energía tienden a tener baja acústica.

3. Correlación Baja:

   Variables como bpm, danceability_%, y speechiness_% muestran baja correlación con la mayoría de las otras variables, indicando que estos atributos individuales de las canciones no están fuertemente relacionados con su popularidad o presencia en playlists y charts.

5. Reducción de Dimensiones (Dimensionality Reduction)

El uso de múltiples variables en un conjunto de datos ocasiona alta dimensionalidad, lo cual implica varios problemas en el momento de la modelación, un fenómeno conocido como la "maldición de la dimensionalidad". Para evitar estos problemas, en este proyecto se decidió realizar una reducción de dimensiones.

Primero, se normalizaron los datos, pero no todos los datos, sino los importantes para crear los clusters. Nos enfocamos en datos que representen si una canción tuvo éxito o no. El siguiente código muestra este proceso y las variables utilizadas:

```
from sklearn.preprocessing import StandardScaler

# Variables relevantes
rank = ['in_spotify_charts', 'in_apple_charts', '
    in_deezer_charts', 'in_shazam_charts', '
    in_spotify_playlists', 'in_apple_playlists', '
    in_deezer_playlists', 'streams']
X1 = spo[rank]

# Normalizar datos
scaler = StandardScaler()
X1_scaled = scaler.fit_transform(X1)
```

Listing 7: Normalizacion

La normalización es un paso crucial porque ajusta los valores de las variables a una escala común sin distorsionar las diferencias en los rangos de valores. Esto es especialmente importante para técnicas de reducción de dimensiones.

Para la reducción de dimensiones, utilizamos la técnica de PCA (Análisis de Componentes Principales). El PCA es una técnica que ayuda a mantener los componentes principales de cada variable, reduciéndolos a un menor número de variables que aún retienen la mayor parte de la variabilidad presente en los datos originales.

```
from sklearn.decomposition import PCA

# Aplicar PCA
pca = PCA(n_components=3)
X1_pca = pca.fit_transform(X1_scaled)

principal_components = pd.DataFrame(X1_pca,
```

```
                                                columns=['componente_1'
                                                    , 'componente_2', '
                                                    componente_3'])
```

Listing 8: Reduccion de dimensiones

El PCA es muy útil ya que permite reducir los datos a componentes principales, eliminando los datos redundantes o menos significativos. En este caso, reducimos las dimensiones a tres componentes principales, lo cual facilita el análisis posterior y mejora la eficiencia de los modelos de clasificación.

6. Clustering

El clustering es una técnica de aprendizaje no supervisado que se utiliza para agrupar objetos similares en clusters. En el contexto de nuestro análisis de canciones en Spotify, queremos identificar grupos de canciones que comparten características similares, lo cual puede ser útil para entender patrones de popularidad y éxito.

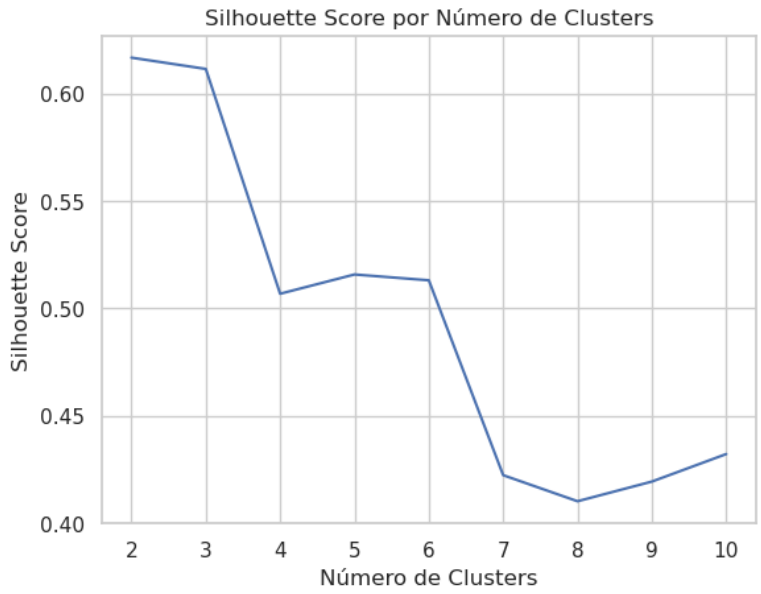
Para determinar el número óptimo de clusters, utilizamos el método del Silhouette Score. Este método evalúa la coherencia dentro de los clusters (qué tan similares son los puntos dentro de un mismo cluster) y la separación entre los clusters (qué tan diferentes son los puntos de diferentes clusters).

El siguiente gráfico muestra el Silhouette Score para diferentes números de clusters:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

silhouette_scores = []
range_n_clusters = list(range(2, 11))

for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, random_state=4,
        n_init=10, max_iter=1000)
    kmeans.fit(principal_components[['componente_1', '
        componente_2', 'componente_3']])
    cluster_labels = kmeans.labels_
    silhouette_avg = silhouette_score(principal_components
        [['componente_1', 'componente_2', 'componente_3']],
         cluster_labels)
    silhouette_scores.append(silhouette_avg)

plt.plot(range_n_clusters, silhouette_scores)
plt.xlabel('N mero de Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score por N mero de Clusters')
```

```
18      plt.show()
```

Listing 9: Números de clusters



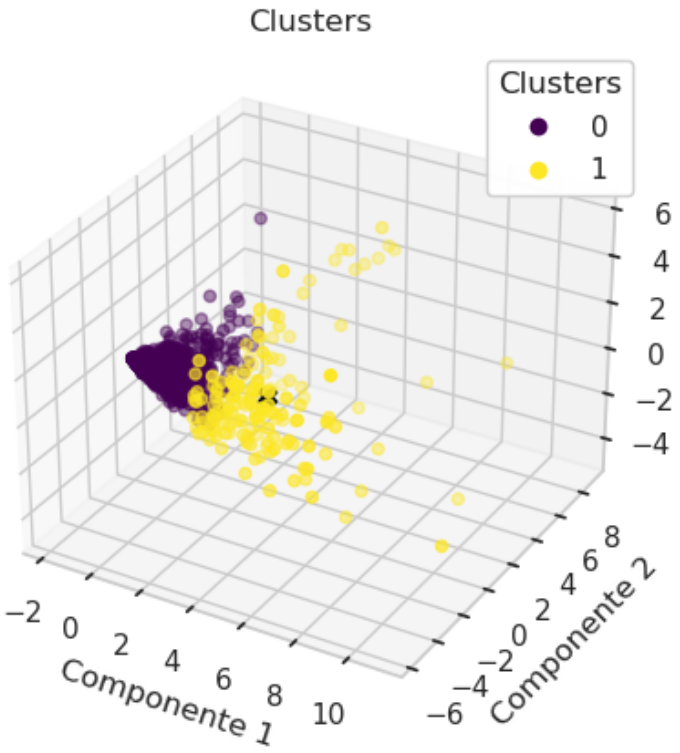
El gráfico sugiere que el número óptimo de clusters es 2, ya que muestra el mayor Silhouette Score en ese punto.

Aplicamos el algoritmo K-Means con 2 clusters y visualizamos los resultados en un gráfico tridimensional:

```
1       from sklearn.cluster import KMeans
2
3       kmeans = KMeans(n_clusters=2, random_state=4, n_init=10,
            max_iter=1000)
4       kmeans.fit(principal_components)
5
6       clusters = kmeans.labels_
7       centroids = kmeans.cluster_centers_
8
9       principal_components['cluster'] = clusters
10
11      fig = plt.figure()
12      ax = fig.add_subplot(111, projection='3d')
13
14      scatter = ax.scatter(principal_components['componente_1'],
15                           principal_components['componente_2'],
16                           principal_components['componente_3'],
17                           c=clusters, cmap='viridis')
18
19      ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:,
            2],
```

```
                s=50, color='black', marker='X', label='
                    Centroids')

    ax.set_xlabel('Componente 1')
    ax.set_ylabel('Componente 2')
    ax.set_zlabel('Componente 3')
    ax.set_title('Clusters')

    # A adir una leyenda
    legend1 = ax.legend(*scatter.legend_elements(), title="
        Clusters")
    ax.add_artist(legend1)

    plt.show()
```

Listing 10: Clusters



El Silhouette Score del modelo K-Means con 2 clusters es 0.6166, lo que indica una buena cohesión y separación entre los clusters:

```
    from sklearn.metrics import silhouette_score

    silhouette_avg = silhouette_score(principal_components[['
        componente_1', 'componente_2', 'componente_3']],
        clusters)
    print(f'Silhouette Score: {silhouette_avg}')
```

Listing 11: Evaluacion de clusters

En resumen, mediante el clustering, hemos podido identificar grupos de canciones con características similares. Este análisis proporciona una base sólida para explorar más a fondo los patrones de éxito de las canciones en Spotify y otras plataformas.

7. Modelación de Clasificación (Classification Modeling)

En esta sección, se describen los pasos seguidos para el modelado de clasificación y se presentan brevemente los modelos utilizados.

Preparación de Datos Primero, los datos fueron estandarizados y divididos en conjuntos de entrenamiento y prueba.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

scaler = StandardScaler()
X = scaler.fit_transform(spo[per])
y = spo['clusters']
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
```

Listing 12: Preparacion de datos

Búsqueda de Mejores Parámetros Para optimizar los modelos, se utilizaron las técnicas de validación cruzada 'GridSearchCV' y 'RandomizedSearchCV'.

```
from sklearn.model_selection import GridSearchCV,
    RandomizedSearchCV

def entrenar(param, modelo, X, y):
    grid = RandomizedSearchCV(param_distributions=param,
                              n_jobs=-1,
                              n_iter=10,
                              cv=4,
                              estimator=modelo,
                              error_score='raise')
    grid.fit(X, y)
    return grid, grid.best_estimator_, grid.best_score_,
        grid.best_params_

def entrenar_g(param, modelo, X, y):
    grid = GridSearchCV(param_grid=param,
```

```
                                n_jobs=-1,
                                cv=4,
                                estimator=modelo,
                                error_score='raise')
        grid.fit(X, y)
        return grid, grid.best_estimator_, grid.best_score_,
            grid.best_params_
```

Listing 13: Validacion cruzada

Parámetros de los Modelos Se definieron los siguientes parámetros para cada modelo:

```
    # SVM (M quinas de vectores de soporte)
    svm_param = {'C': [0.1, 1, 10, 100], 'gamma': [0.1, 0.01,
        0.001, 0.0001], 'kernel': ['linear', 'rbf']}

    # Random Forest
    rf_param = dict(n_estimators=list(range(1, 100, 25)),
                    criterion=['gini', 'entropy'],
                    max_depth=[x for x in list(range(2, 5))] +
                        [None],
                    min_samples_split=[x for x in list(range(2,
                         4))],
                    min_samples_leaf=[x for x in list(range(2,
                        4))],
                    max_features=[None] + [i * .05 for i in
                        list(range(2, 4))],
                    max_leaf_nodes=list(range(2, 10)) + [None],
                    min_impurity_decrease=[x * .10 for x in
                        list(range(2, 4))],
                    oob_score=[True,False],
                    warm_start=[True, False],
                    class_weight=[None, 'balanced'],
                    max_samples=[None])

    # Regresi n Log stica
    rl_param = {'C': [0.1, 1, 10, 100]}

    # Red Neuronal
    rn_param = dict(activation = ['identity', 'logistic', 'tanh
        ', 'relu'],
                    solver = ['lbfgs', 'sgd', 'adam'],
                    alpha = np.arange(0.0001, 0.001, 0.0001),
                    learning_rate = ['constant', 'invscaling',
                        'adaptive'])

    # Ada Boost
    ab_param = dict(n_estimators = range(2,10),
                    learning_rate = np.arange(0.1,1,0.1),
                    algorithm = ['SAMME.R'])

    # An lisis Discriminante
    ad_param = dict(solver = ['svd', 'lsqr', 'eigen'])
```

Listing 14: Parametros

Balanceo de Clases Se utilizó la técnica 'SMOTETomek' para balancear las clases en los datos de entrenamiento.

```
from imblearn.combine import SMOTETomek

os_us = SMOTETomek()
X_train_res, y_train_res = os_us.fit_resample(X_train,
    y_train)
```

Listing 15: Balance de clases

Descripción de los Modelos de Clasificación A continuación, se describen brevemente los modelos de clasificación utilizados en el análisis.

1. Support Vector Machine (SVM)
   - Descripción: Encuentra un hiperplano óptimo que separa las clases de datos.
   - Ventajas: Eficaz en espacios de alta dimensión y con un margen de separación claro.
   - Desventajas: No adecuado para conjuntos de datos muy grandes.
   - Mejores Parámetros:

```
{'C': 0.1, 'gamma': 0.1, 'kernel': 'rbf'}
```

   - Precisión: 0.5771375464684014

2. Random Forest
   - Descripción: Utiliza un conjunto de árboles de decisión para mejorar la precisión.
   - Ventajas: Maneja bien grandes conjuntos de datos y es robusto ante el overfitting.
   - Desventajas: Puede ser lento y consumir mucha memoria.
   - Mejores Parámetros:

```
{'warm_start': True,
 'oob_score': True,
 'n_estimators': 1,
 'min_samples_split': 2,
 'min_samples_leaf': 3,
 'min_impurity_decrease': 0.2,
 'max_samples': None,
 'max_leaf_nodes': 5,
 'max_features': None,
 'max_depth': 2,
 'criterion': 'entropy',
 'class_weight': 'balanced'}
```

- Precisión: 0.5018587360594795

3. Regresión Logística
   - Descripción: Modela la probabilidad de una clase específica utilizando una función logística.
   - Ventajas: Simple de implementar y coeficientes fácilmente interpretables.
   - Desventajas: No funciona bien con relaciones no lineales.
   - Mejores Parámetros:

```
{'C': 1}
```

   - Precisión: 0.5585501858736059

4. Red Neuronal (Neural Network)
   - Descripción: Modela relaciones complejas utilizando capas de neuronas.
   - Ventajas: Capaz de modelar relaciones no lineales complejas.
   - Desventajas: Requiere grandes cantidades de datos y es computacionalmente intensiva.
   - Mejores Parámetros:

```
{'solver': 'lbfgs',
 'learning_rate': 'constant',
 'alpha': 0.0006000000000000001,
 'activation': 'relu'}
```

   - Precisión: 0.8596654275092936

5. AdaBoost
   - Descripción: Combina varios modelos débiles para crear un clasificador fuerte.
   - Ventajas: Mejora la precisión de clasificadores débiles.
   - Desventajas: Sensible al ruido y a los datos atípicos.
   - Mejores Parámetros:

```
{'n_estimators': 6, 'learning_rate': 0.6,
    'algorithm': 'SAMME.R'}
```

   - Precisión: 0.6078066914498141

6. Análisis Discriminante
   - Descripción: Clasifica utilizando combinaciones lineales de características.

- Ventajas: Rápido y eficiente para conjuntos de datos pequeños.
- Desventajas: No adecuado si no se cumplen las suposiciones de normalidad y homogeneidad de varianza.
- Mejores Parámetros:

```
{'solver': 'svd'}
```

- Precisión: 0.5548327137546468

El análisis realizado demuestra que la Red Neuronal es el modelo más efectivo para este conjunto de datos, logrando una precisión de 0.8596654275092936. Los resultados indican que la Red Neuronal no solo superó a los demás modelos, sino que también mostró una notable diferencia en precisión en comparación con AdaBoost, que fue el segundo mejor modelo con una precisión de 0.6078066914498141. Esta diferencia sugiere que la Red Neuronal es mucho más eficaz para capturar las complejidades inherentes en los datos.

8. Evaluación del Modelo (Model Evaluation)

Para determinar la confiabilidad de los modelos, se utilizarán las siguientes herramientas de evaluación:

1. **Métricas de Evaluación**: Para obtener una comprensión numérica del rendimiento de cada modelo.
2. **Matriz de Confusión**: Para visualizar las predicciones correctas e incorrectas.
3. **Curva de Aprendizaje**: Para analizar el rendimiento del modelo en función de la cantidad de datos de entrenamiento.

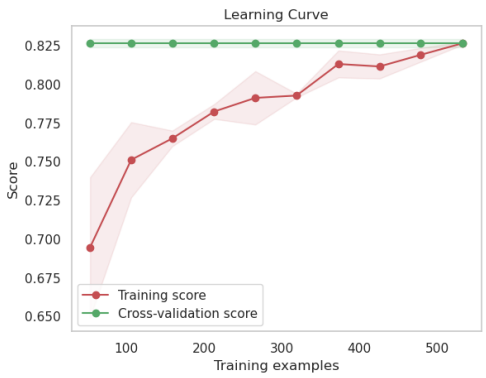
Se han definido funciones para facilitar la implementación de estas evaluaciones:

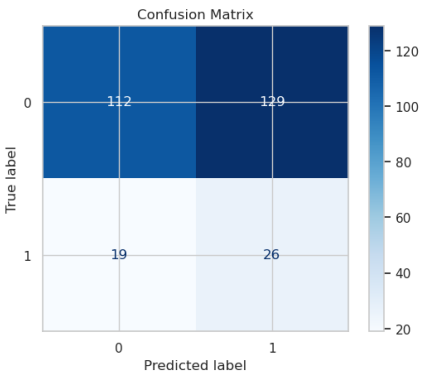
```
def plot_learning_curve(model, X, y):
train_sizes, train_scores, test_scores = learning_curve(
    model, X, y, cv=5, n_jobs=-1, train_sizes=np.linspace
    (0.1, 1.0, 10))
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(train_sizes, train_mean, 'o-', color='r', label='
    Training score')
```

```
    plt.plot(train_sizes, test_mean, 'o-', color='g', label='
        Cross-validation score')
    plt.fill_between(train_sizes, train_mean - train_std,
        train_mean + train_std, alpha=0.1, color='r')
    plt.fill_between(train_sizes, test_mean - test_std,
        test_mean + test_std, alpha=0.1, color='g')

    plt.title('Learning Curve')
    plt.xlabel('Training examples')
    plt.ylabel('Score')
    plt.legend(loc='best')
    plt.grid()
    plt.show()

def plot_confusion_matrix(model, X_test, y_test):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap='Blues')
    plt.title('Confusion Matrix')
    plt.show()

def print_classification_report(model, X_test, y_test):
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred)
    print("Classification Report:\n", report)
```

Listing 16: Evaluacion de modelos

- Support Vector Machine (SVM)

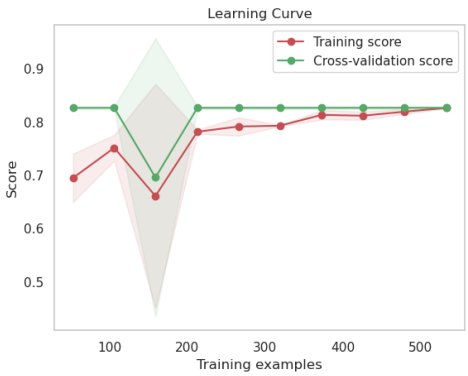


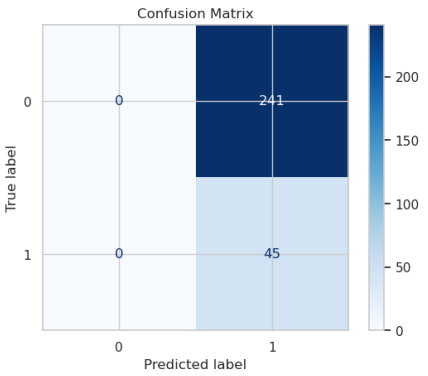



```
        Classification Report:
                      precision    recall  f1-score   support

                   0       0.85      0.46      0.60       241
                   1       0.17      0.58      0.26        45

            accuracy                           0.48       286
```

macro avg	0.51	0.52	0.43	286
weighted avg	0.75	0.48	0.55	286

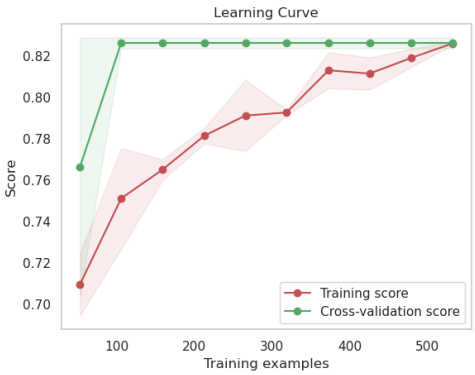
- Random Forest

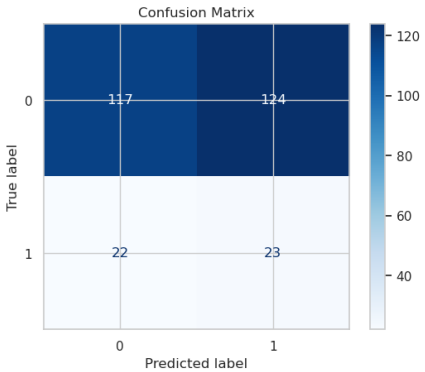




	precision	recall	f1-score	support
0	0.00	0.00	0.00	241
1	0.16	1.00	0.27	45
accuracy			0.16	286
macro avg	0.08	0.50	0.14	286
weighted avg	0.02	0.16	0.04	286

- Regresión Logística

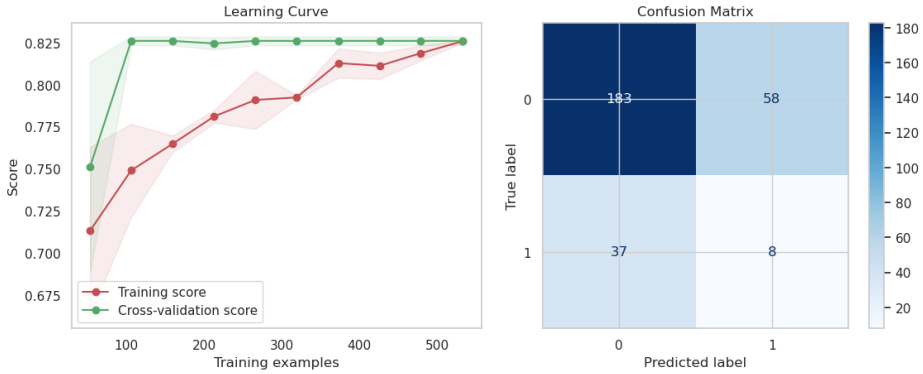




	precision	recall	f1-score	support
0	0.84	0.49	0.62	241
1	0.16	0.51	0.24	45

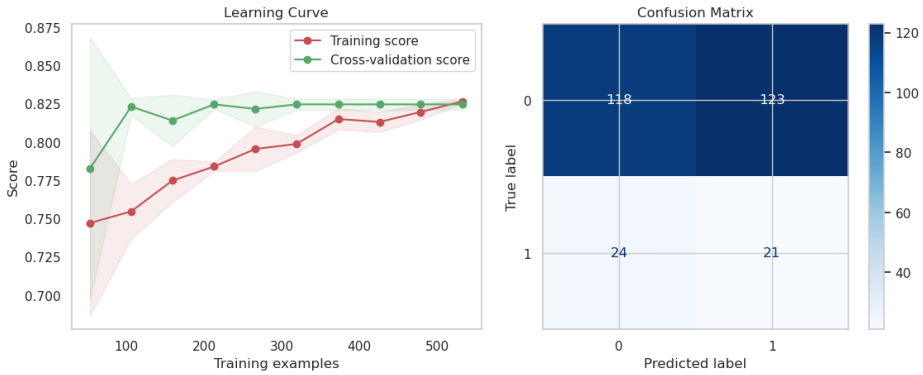
```
                                                                    
    accuracy                           0.49       286
   macro avg       0.50      0.50      0.43       286
weighted avg       0.73      0.49      0.56       286
```

- Red Neuronal (Neural Network)




```
        Classification Report:
                      precision    recall  f1-score   support

                   0       0.83      0.76      0.79       241
                   1       0.12      0.18      0.14        45

            accuracy                           0.67       286
           macro avg       0.48      0.47      0.47       286
        weighted avg       0.72      0.67      0.69       286
```

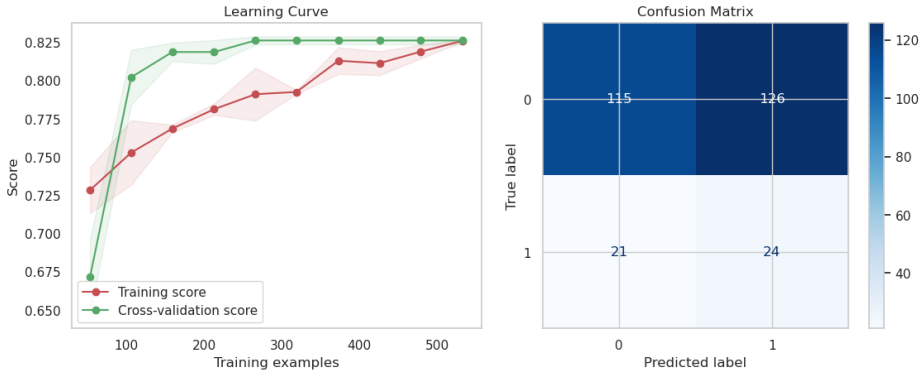
- AdaBoost




```
        Classification Report:
                      precision    recall  f1-score   support

```

```
                0       0.83      0.49      0.62       241
                1       0.15      0.47      0.22        45

         accuracy                           0.49       286
        macro avg       0.49      0.48      0.42       286
     weighted avg       0.72      0.49      0.55       286
```

- Análisis Discriminante




```
        Classification Report:
                      precision    recall  f1-score   support

                   0       0.85      0.48      0.61       241
                   1       0.16      0.53      0.25        45

            accuracy                           0.49       286
           macro avg       0.50      0.51      0.43       286
        weighted avg       0.74      0.49      0.55       286
```

Teniendo en cuenta los resultados obtenidos, se observa que el modelo de Red Neuronal es el que presenta la mayor precisión, alcanzando una precisión de 0.67. Además, es importante destacar que, aunque el modelo de AdaBoost tuvo un rendimiento aceptable con una precisión de 0.49, otros modelos como el SVM, Random Forest, Regresión Logística y Análisis Discriminante no superaron esta precisión.

9. Conclusiones

El tratamiento, análisis y modelación de datos es un proceso exhaustivo y complejo que requiere una serie de habilidades especializadas. Primero,

en el tratamiento de datos, es fundamental asegurar que los datos estén en condiciones óptimas. Los datos crudos a menudo no son útiles sin un proceso adecuado de limpieza y preparación, lo cual exige experiencia y el uso de diversas herramientas.

En el análisis de datos, es crucial tener una comprensión clara de los objetivos y necesidades del cliente o del proyecto. No se trata solo de generar gráficos, sino de plantear preguntas significativas y encontrar respuestas concretas utilizando los datos disponibles. Esta fase requiere una combinación de creatividad y análisis crítico para interpretar correctamente la información.

La modelación de datos es quizás la etapa más laboriosa y técnica. Implica no solo la creación de modelos predictivos, sino también la búsqueda de formas de maximizar el valor de los datos, ya sea mediante la creación de clusters, la reducción de dimensiones, o la selección de características adecuadas. Cada modelo tiene sus propias ventajas y desventajas:

- Red Neuronal: Alta precisión y capacidad para modelar relaciones no lineales complejas, aunque requiere muchos datos y es computacionalmente intensiva.
- Support Vector Machine (SVM): Eficaz en espacios de alta dimensión, pero no adecuado para conjuntos de datos muy grandes.
- Random Forest: Robusto ante el overfitting y maneja bien grandes conjuntos de datos, pero puede ser lento y consumir mucha memoria.
- Regresión Logística: Simple de implementar y fácil de interpretar, aunque no maneja bien relaciones no lineales.
- AdaBoost: Mejora la precisión de clasificadores débiles, aunque es sensible al ruido y a los datos atípicos.
- Análisis Discriminante: Rápido y eficiente para conjuntos de datos pequeños, pero no adecuado si no se cumplen ciertas suposiciones estadísticas.

En resumen, el trabajo de un científico de datos no se limita a la creación de modelos predictivos, sino que también incluye la preparación, análisis e interpretación de datos de manera efectiva. Este proyecto ha demostrado la importancia de cada una de estas etapas y ha subrayado la necesidad de un enfoque meticuloso para obtener resultados precisos y útiles.