Grupo 4

CARLOTA SALAZAR, YUSHAN YANG, JULIA YI

PD1 - GIDIA

Grupo 4

EVENTPEAK – E2



Contenido

[1. Extracción de los datos 2](#_Toc160527521)

[Fuente 1: Ticketmaster 2](#_Toc160527522)

[Fuente 2: Spotify 2](#_Toc160527523)

[Fuente 3: LastFM 3](#_Toc160527524)

[Fuente 4: Google Trends 3](#_Toc160527525)

[2. Transformación de los datos 3](#_Toc160527526)

[Evaluación inicial 3](#_Toc160527527)

[Limpieza de los datos 4](#_Toc160527528)

[Integración de datos 5](#_Toc160527529)

[Creación de nuevas variables 5](#_Toc160527530)

[3. Carga de los datos 6](#_Toc160527531)

[4. Exploración de los datos 6](#_Toc160527532)

[5. Otras consideraciones 6](#_Toc160527533)

# Extracción de los datos

## Fuente 1: Ticketmaster

Para obtener los datos de Ticketmaster, buscamos en internet la página de Ticketmaster Developer (enlace: <https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/> ), donde explica cómo o qué opciones podemos usar de API para obtener los datos. En ese enlace, hay un apartado para crear una URL personalizada a partir de los parámetros especificados (enlace: <https://developer.ticketmaster.com/api-explorer/v2/>). Entre los parámetros que incluimos estaban *Country Code*, que lo pusimos en España (por tanto, la unidad de moneda siempre es el euro), y *classification name* que escogimos “music”. También vamos especificando manualmente la página que queremos obtener. Así, quedó el siguiente enlace que podemos ver en el código:

'https://app.ticketmaster.com/discovery/v2/events?**apikey=4uZ7cKblFpFckdrfHMGrT2coHBKCiAjs**&**countryCode=ES**&**classificationName=music**&locale=\*&**page**=' + str(pag)

Tras especificar esa URL, (con la API key resaltada), para extraer los datos hemos necesitado la librería request, que le pregunta a la API si podemos obtener sus datos y nos devuelve la respuesta. Para guardar los datos obtenidos en un dataframe usaremos la librería pandas.

Un aspecto relevante de la captura es que Ticketmaster devuelve un fichero en formato JSON, formado por 3 diccionarios cada uno con una información específica. Puede verse en el segundo enlace anterior, al pie de la web tras filtrar por los elementos seleccionados. Aquí se describe:

* *pages*: nos dice el número de páginas totales -39-, el número de página actual -0-, el número de eventos que hay por página -20- y el número de eventos totales que hay -772-\*. Cabe destacar que Ticketmaster tiene un límite de elementos totales para ofrecer, pero como hemos filtrado por País=España, este límite no llega a superarse.

Para obtener los eventos de todas las páginas, leemos en bucle desde página=0 hasta página=total\_páginas. Cada página leída la pasamos a formato dataframe y la juntamos con el resto de páginas ya leídas y pasadas a dataframe para tener un único conjunto.

* *\_embedded*: es el diccionario con los eventos (información en bruto que tratamos más adelante).
* *\_links*: devuelve las URLs a: la primera página, la página actual, la página siguiente y la última página.

\* Cada semana, la API va ofreciéndonos nuevos datos, por lo que estas cantidades se actualizan.

## Fuente 2: Spotify

En cuanto a Spotify, si queremos acceder al número de oyentes mensuales de un artista, primero necesitamos algunos parámetros concretos (client\_id, client\_secret) para obtener el token de acceso y hacer la solicitud de búsqueda, pues Spotify es muy restrictivo a la hora de ofrecer datos a ajenos. De nuevo, usaremos la librería request.

A continuación, necesitamos obtener el id del artista del que queremos buscar su número de oyentes mensuales. Para ello, especificamos el nombre del artista y a través de la URL <https://api.spotify.com/v1/search> lo podremos obtener.

Por último, este id lo incorporamos al enlace https://api.spotify.com/v1/artists/**{artist \_id}** , que nos devuelve el valor que queríamos obtener (el número de oyentes mensuales del artista).

La idea es realizar estas 3 operaciones en bucle para cada artista de nuestro dataframe.

## Fuente 3: LastFM

El proceso para la extracción de datos de LastFM es mucho más sencillo, por eso decidimos incluirlo, de momento, en la lista inicial de fuentes empleadas. Sin embargo, valoramos la opción de excluir esta fuente por redundancia con Spotify, ya que ambos nos aportan el número de oyentes de un artista, aunque la ventaja de LastFM es que podemos obtener el número de oyentes en cualquier fecha pasada.

Como decimos, el proceso de obtención de número de oyentes en LastFM es muy sencillo y pasa por solicitar los datos a través del enlace siguiente, dado el nombre del artista:

f"http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&**artist={ARTIST\_NAME}**&api\_key={LASTFM\_API\_KEY}&format=json"

Esto devuelve un dataframe y ahora filtramos por la fecha específica.

## Fuente 4: Google Trends

Google Trends nos dará la popularidad de cada artista en cada comunidad autónoma de España. Para extraer estos datos, es requerida la librería pytreds.request.

Solo debemos especificar el nombre del artista y el período de tiempo y realizar la búsqueda filtrando por país (España) y por tipo de datos devueltos (por comunidad autónoma). Este proceso no requiere de ninguna URL ya que se hace todo a través de la librería mencionada.

Sin embargo, Google Trends es muy restrictivo, con un límite bastante bajo de requests por unidad de tiempo, y además requiere de un tiempo alto de espera para obtener los datos solicitados. Por ello, puede que no lleguemos a obtener todos los datos que queremos a tiempo. Si esto ocurriera, otra opción sería recopilar el número de seguidores en una red social como Instagram para cada artista (también se ha probado con código para asegurar esta extracción).

# Transformación de los datos

## Evaluación inicial

Nuestro conjunto de datos tiene aproximadamente 900 observaciones de partida correspondientes a los datos que nos ofrece Ticketmaster, pero, como dijimos anteriormente, esta cifra va en aumento cada semana. Vamos a pararnos en esta cifra inicial para tratar el ciclo 2 del Proyecto (a día 02/03/2024, con 5 dataframes).

Los datos que obtenemos cada semana los guardamos porque a la siguiente semana salen datos nuevos y algunas observaciones pueden desaparecer de las obtenciones anteriores. Por eso, además de ir guardándolos, vamos integrando los datos antiguos con los nuevos. Estimamos unas 1000 observaciones en bruto.

El conjunto de datos estará formado por multitud de variables obtenidas de distintas fuentes. Analicemos estas variables ofrecidas por cada fuente:

Como decimos, TicketMaster es el núcleo de nuestro conjunto de datos. En primer lugar, tenemos el campo “name”, que contiene el nombre de los eventos. Vemos que vamos a tener que trabajar con esta columna para obtener el nombre del artista (por ejemplo, el nombre del evento puede ser Taylor Swift On Tour, pero el nombre del artista es solo Taylor Swift).

Las siguientes 6 columnas, las eliminaremos por irrelevancia o invariación (no son útiles). Estas columnas son: “type” (adquiere únicamente el valor “event”), “id” (identificador único del evento), “test” (es booleana y siempre toma el valor “False”), “url” (enlace a la página de ticketmaster para comprar las entradas de dicho evento), “locale” (marca “es” pues hemos seleccionado únicamente los eventos de España para realizar este Proyecto) e “images” (diccionario con datos numéricos y strings que describen una imagen).

Las variables que más nos interesan son “dates”, “priceRanges” y “\_embedded” que son diccionarios en los que podemos encontrar la información más relevante para predecir nuestro objetivo como horas de inicio y fin del evento (números en formato fecha), el género y el subgénero de música (strings), los rangos de los precios del evento (números decimales), información geográfica completa del evento (strings)… En “priceRanges”, vemos que hay varios tipos de precios (estándar, con tarifas como gastos de envíos…). En “\_embedded”, encontramos *VenueName* que es el nombre del lugar donde se realiza el evento, el cual puede tomar muchos valores distintos, pero los clasificaremos mejor como “estadio”, “teatro” o “sala”, según su capacidad. La idea, además, es desglosar la información más importante de todos estos diccionarios para crear columnas nuevas.

También nos servirán las columnas “classifications”, “sales” y “promoter” que son también diccionarios para indicar tipo del evento, días y horas de inicio y final del evento, nombre del promotor o promotores… Quizá nos sea útil este tipo de información.

Por último, tampoco nos interesan, por inutilidad, las variables “\_links” y “seatmap”.

Otra variable de nuestro conjunto de datos inicial es el número de oyentes en Spotify (“Spotify\_Mensuales”), de tipo entero, que obtendremos de la fuente Spotify. Es una variable muy útil que guiará al sistema de AA a aprender qué artistas tienen mayor importancia o repercusión. Tendremos que obtener todos los artistas de nuestro conjunto y buscar el número de oyentes para cada uno de ellos. Lo mismo ocurre con el número de oyentes de LastFM (“Last.fm\_Oyentes”). Ambas expresan lo mismo, pero no queríamos limitarnos únicamente a una plataforma porque podíamos estar dejándonos fuera a muchos oyentes, pues LastFM es una de las alternativas a Spotify más populares, con gran uso (y menos restrictivo) en varios países.

De momento, no hemos añadido la variable de “popularidad” de Google Trends porque, como ya dijimos, es muy lento y molesto por lo que no conseguimos obtener los datos que queríamos.

## Limpieza de los datos

En primer lugar, hemos hecho limpieza con las filas que trataban sobre film, festivales, musicales, servicios de autobuses, entradas meet, vip o meet&greet, ticket y package (recordemos que cada fila representaba un evento, que nosotras filtramos como conciertos, pero dentro de esta categoría se incluyen variaciones que no nos interesan para el modelo por ambigüedad, como las mencionadas). Para descartar estas filas, primero nos fijamos en el campo “name”, que podía contener estas palabras clave para referirse al tipo de evento, y también en el campo “classifications”, que contiene información sobre el tipo y subtipo del evento.

En segundo lugar, atendiendo a las columnas, hemos quitado aquellas que no nos eran útiles (especificadas en los puntos anteriores) y hemos analizado y desglosado las que sí eran útiles pero que concentraban multitud de información (en forma de diccionario).

Otra cosa que queremos comentar es que los diccionarios en realidad son un string, por lo que se encuentran entre comillas simples y queremos quitárselas. Además, los valores booleanos vienen como *True* y *False* y tendrían que empezar con minúsculas para que Python los reconozca como booleanos. Aunque fue mucho más complicado para \_embedded por ser un valor muy grande, al final logramos todas estas transformaciones para todas las variables involucradas.

## Integración de datos

Hemos tenido que realizar varios procesos de integración de datos, el primero de ellos con los datos de Ticketmaster pues, como decíamos, cada semana obtenemos unos nuevos y los tenemos que ir recopilándolos. Cuando llegamos al límite de datos que queríamos, viendo el conjunto en general, nos percatamos de que había observaciones que aparecían nuevas en cada extracción, así como observaciones que se eliminaban. Por lo que teníamos que combinar en uno todos nuestros conjuntos parciales obtenidos (que son 5), usando concat(). Luego eliminamos las observaciones duplicadas usando drop\_duplicates(inplace=True).

Por otro lado, hemos integrado datos de distintas fuentes: todas las variables de Ticketmaster con la de Spotify y con la de LastFM. Al añadir estas variables de las fuentes secundarias al conjunto de datos principal con la función merge(), aparecen nuevas filas con valores nulos que hemos tenido que eliminar.

## Creación de nuevas variables

Entre las variables de TicketMaster que nos interesaban (“name”, “dates”, “classifications”, “priceRanges”, “\_embedded”, “sales” y “promoter”), ya comentamos anteriormente que estaban en formato de diccionario y que contenían información que podía ser desglosada. En esta etapa del proyecto, nos dedicamos a crear nuevas variables derivadas de estas, de la siguiente manera:

nameArtist: la columna “name” era el nombre del concierto / evento, por lo que muchas veces no concuerda (sino que contiene) el nombre del artista. Por eso, hemos creado una nueva columna “nameArtist”, de tipo string, que tenga únicamente el nombre del artista, el cual podemos obtener bien filtrando los elementos de “name”, bien extrayéndolo de un campo de dentro de “\_embedded”.

startDateTime: es la fecha y hora de inicio para la que el evento está fijado. Lo hemos obtenido de la columna “dates”. Está en formato fecha.

Tipo y subtipo: el tipo y subtipo del evento (evento, festival, ópera, etc.), lo hemos obtenido de la columna “classifications”. Estas variables, de tipo string, nos sirven para limpiar aquellas filas que contengan un tipo o subtipo de los que mencionamos anteriormente que íbamos a descartar (festival, musical...). Cuando terminamos con esta tarea, estas variables las eliminamos.

Min\_price y max\_price: contienen los precios mínimos y máximos, respectivamente, obtenidos del diccionario “PriceRanges” solo para el tipo de precio “estándar”. Son de formato decimal.

VenueName, VenueCity y VenueState: son los datos geográficos (lugar concreto que clasificaremos como “estadio”, “teatro” o “sala”; ciudad; estado/ comunidad autónoma) de la realización del evento, los obtendremos del diccionario “\_embedded”. El formato es string.

Genre y subgenre: es el género y el subgénero de música del concierto -pop, rock, etc.-, que también obtendremos de “\_embedded”. Son de tipo string.

StartTimeSale y EndTimeSale: son las fechas y horas de inicio y final de la venta de entradas para el evento, que podemos encontrar en “sales”. Están en formato fecha.

Promotor: es el nombre del promotor (string) obtenido de “promoter”. Esta columna decidimos dejarla de momento porque puede aportar información útil, pero pendiente de evolución.

# Carga de los datos

De momento, hemos sacado el Parquet de todos los datos (“EventPeak.parquet”).

Para organizar los datos de conciertos en distintos archivos Parquet, hemos pensado hacerlo de una de las siguientes maneras:

* Por año o mes: para facilitar el análisis por períodos temporales.
* Por ubicación: hemos pensado hacerlo por ciudades, así podremos realizar análisis específicos para cada ciudad donde se llevan a cabo los eventos (para decidir hasta dónde nos interesa llevar EventPeak), o también por comunidades autónomas. Esta última, nos parece la opción más interesante, pero tenemos un problema con la distribución, pues en Madrid por ejemplo hay 376 observaciones para el segundo caso, mientras que en Toledo hay 1, entonces, ¿vamos a hacer ficheros que tengan únicamente una entrada? Otra idea para paliar esto es: las comunidades autónomas con número aceptable de participaciones las pasamos a Parquet individualmente y el resto (con entre 1 y 10 participaciones, más o menos) las agrupamos en otro Parquet aparte.

En cualquier caso, no hicimos la separación de Parquets porque nos gustaría recibir primero alguna recomendación u orientación.

# Exploración de los datos

Cada variable, su descripción y su tipo ya ha sido explicado a lo largo del documento. A continuación, podemos ver la media, moda, desviación estándar, cuartiles, mínimos y máximos para las variables numéricas. Debajo se encuentra la matriz de correlación.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Queda pendiente un análisis descriptivo para las variables categóricas y la interpretación de todas las medidas realizadas.

# Otras consideraciones

Al final no vamos a trabajar con Google Trends porque no hemos conseguido obtener los datos que queríamos ni tampoco pensamos que los vayamos a conseguir a tiempo (por las características descritas en el apartado 1.4).

La clasificación de la variable VenueName en “estadio”, “teatro” o “sala” queda pendiente para la siguiente entrega, pues para esta ya eran muchas cosas y no daba tiempo.

Esta semana la dedicaremos a revisar código y, sobre todo, ordenarlo y comentarlo, además de organizar el repositorio de Git.