🎼 Music Streaming Analytics - Análisis de Comportamiento de Usuarios

- Notebook by Julio C. Martínez I.
- Supported by Francisco Alfaro & Alfonso Tobar
- Code Review by Carlos Ortíz

Licencia

Copyright @2023 by Julio César Martínez Izaguirre

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

```
http://www.apache.org/licenses/LICENSE-2.0
```

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License

Tabla de Contentido

Introducción

Siempre que investiguemos, necesitamos formular hipótesis que después podamos probar. A veces aceptamos estas hipótesis; otras, las rechazamos. Para tomar las decisiones correctas, una empresa debe ser capaz de entender si está haciendo las suposiciones correctas.

En este proyecto, vamos a comparar las preferencias musicales de las ciudades de Springfield y Shelbyville. Estudiaremos los datos reales de **Yandex.Music** para probar las hipótesis de abajo y comparar el comportamiento del usuario de esas dos ciudades.

Objetivo:

Probarémos tres hipótesis:

1. La actividad de los usuarios difiere según el día de la semana y dependiendo de la ciudad.
2. Los lunes por la mañana, los habitantes de Springfield y Shelbyville escuchan diferentes géneros. Lo mismo ocurre con los viernes por la noche.
3. Los oyentes de Springfield y Shelbyville tienen preferencias distintas. En Springfield prefieren el pop mientras que en Shelbyville hay más aficionados al rap.

Etapas

Los datos del comportamiento del usuario se almacenan en el archivo `/datasets/music_project_en.csv`. No hay ninguna información sobre la calidad de los datos así que necesitaremos examinarlos antes de probar las hipótesis.

Primero, evaluaremos la calidad de los datos y veremos si los problemas son significativos. Entonces, durante el preprocesamiento de datos, tomaremos en cuenta los problemas más críticos.

Este proyecto consistirá en tres etapas:

1. Descripción de los datos
2. Preprocesamiento de datos
3. Prueba de hipótesis

Volver a Contenidos

Etapa 1. Descripción de los datos

Comenzamos esta etapa abriendo los datos de **Yandex.Music** y luego vamos a examinarlos, para ello vamos a llamar a la librería de **Pandas** que nos ayudará rápidamente con la lectura de los archivos.

```
# importando pandas
import pandas as pd
```

En el siguiente paso vamos a leer el archivo `music_project_en.csv` de la carpeta `/datasets/` y vamos a guárdarlo en la variable `df`:

```
# leyendo el archivo y almacenándolo en df
df = pd.read_csv('/content/music_project_en.csv')
```

Vamos a imprimir las 10 primeras filas de la tabla:

```
# obteniendo las 10 primeras filas de la tabla df
df.head(10)
```

	userID	Track	artist	genre	City	time	Day
0	FFB692EC	Kamigata To Boots	The Mass Missile	rock	Shelbyville	20:28:33	Wednesday
1	55204538	Delayed Because of Accident	Andreas Rönnberg	rock	Springfield	14:07:09	Friday
2	20EC38	Funiculì funiculà	Mario Lanza	pop	Shelbyville	20:58:07	Wednesday
3	A3DD03C9	Dragons in the Sunset	Fire + Ice	folk	Shelbyville	08:37:09	Monday
4	E2DC1FAE	Soul People	Space Echo	dance	Springfield	08:34:34	Monday
5	842029A1	Chains	Obladaet	rusrap	Shelbyville	13:09:41	Friday
6	4CB90AA5	True	Roman Messer	dance	Springfield	13:00:07	Wednesday
		Feeling This	Polina				

```
# Últimas 10 filas de la tabla
df.tail(10)
```

	userID	Track	artist	genre	City	time	Day
65069	BE1AAD74	Waterwalk	Eduardo Gonzales	electronic	Springfield	20:38:59	Monday
65070	49F35D53	Ass Up	Rameez	dance	Springfield	14:08:58	Friday
65071	92378E24	Swing it Like You Mean it	OJOJOJ	techno	Springfield	21:12:56	Friday
65072	C532021D	We Can Not Be Silenced	Pänzer	extrememetal	Springfield	08:38:24	Friday
65073	83A474E7	I Worship Only What You Bleed	The Black Dahlia Murder	extrememetal	Springfield	21:07:12	Monday
65074	729CBB09	My Name Maybe	McLean	rnb	Springfield	13:32:28	Wednesday

Podemos apreciar algunos inconvenientes con nuestra tabla pero antes de tratarlos vamos a obtener un poco más de información general con un comando:

```
1 # obteniendo información general sobre los datos en df
2 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65079 entries, 0 to 65078
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0     userID  65079 non-null  object
 1   Track     63736 non-null  object
 2   artist    57512 non-null  object
 3   genre     63881 non-null  object
 4     City    65079 non-null  object
 5   time      65079 non-null  object
 6   Day       65079 non-null  object
dtypes: object(7)
memory usage: 3.5+ MB
```

OBSERVACIONES

La tabla contiene siete columnas. Todas almacenan el mismo tipo de datos: `object`.

De acuerdo con la documentación:

- `'userID'` — identificador del usuario
- `'Track'` — título de la pista
- `'artist'` — nombre del artista

- `'genre'` — género
- `'City'` — ciudad del usuario
- `'time'` — el periodo de tiempo exacto en que se reprodujo la pista
- `'Day'` — día de la semana

Después de una breve visualización podemos ver algunos problemas con el estilo en los nombres de las columnas como son:

1. Algunos nombres están en mayúsculas, otros en minúsculas.
2. Hay algunos espacios en algunos nombres.
3. Presentación inadecuada de los datos.
4. El número de valores de las columnas es diferente. Esto significa que los datos contienen valores ausentes.

Conclusiones

Cada fila de la tabla almacena datos de la pista que fue reproducida. Algunas columnas describen la pista en sí: su título, el artista y el género. El resto transmite la información del usuario: la ciudad de la que viene, el tiempo que ha reproducido la pista.

Está claro que los datos son suficientes para probar la hipótesis. Sin embargo, hay valores ausentes.

Para continuar, necesitamos preprocesar los datos.

Volver a Contenidos

Etapa 2. Preprocesamiento de datos

Durante esta etapa vamos a realizar un **preprocesamiento** de los datos de tal forma que vamos a corregir el **formato** en los encabezados de las columnas y acontinuación vamos a ocuparnos de los **valores ausentes**, después, comprobarémos si hay **valores duplicados** en los datos.

Estilo del encabezado

Vamos a visualizar los nombres de las columnas.

```
# la lista de los nombres de las columnas en la tabla df
df.columns
```

```
Index(['  userID', 'Track', 'artist', 'genre', '  City  ', 'time',
'Day'], dtype='object')
```

Como hemos visto anteriormente el formato de los nombres es erroneo, vamos a corregirlos a la forma **"snake_case"** de acuerdo con las reglas del buen estilo de programación:

- Si el nombre tiene varias palabras, utilizamos snake_case
- Todos los caracteres deben ser minúsculas
- Eliminamos los espacios

```
# renombra las columnas
df = df.rename(
    columns={
        '  userID' : 'user_id',
        'Track'    : 'track',
        'artist'   : 'artist',
        'genre'    : 'genre',
        '  City  ' : 'city',
        'time'     : 'time',
        'Day'      : 'day'
    }
)
```

Después de aplicar las correcciones debemos comprobar el resultado. Vamos a imprimir los nombres de las columnas una vez más:

```
# comprobando el resultado: la lista de los nombres de las colu
df.columns
```

```
Index(['user_id', 'track', 'artist', 'genre', 'city', 'time', 'day'],
dtype='object')
```

Volver a Contenidos

Valores ausentes

El siguiente paso consiste en encontrar el número de **valores ausentes** en la tabla. Para ello, vamos a utilizar dos métodos de pandas:

```
# calculando valores ausentes
df.isna().sum()
```

```
user_id       0
track      1343
artist     7567
genre      1198
city          0
time          0
day           0
dtype: int64
```

No todos los **valores ausentes** afectan a la investigación. Por ejemplo, los valores ausentes en la **pista y artista** no son cruciales. Simplemente podemos reemplazarlos por marcadores claros.

Pero los valores ausentes en `'genre'` pueden afectar la comparación entre las **preferencias musicales** de Springfield y Shelbyville. Normalmente sería útil saber las razones por las cuales hay datos ausentes e intentar recuperarlos. Pero no tenemos esa oportunidad en este proyecto. Así que tendremos que:

- Rellenar esos valores ausentes con marcadores
- Evaluar cuánto podrían afectar los valores ausentes a tus cómputos.

Para reemplazar los valores ausentes en `'track'`, `'artist'`, y `'genre'` con la string `'unknown'` vamos a crear la lista `columns_to_replace`, luego vamos a recorrerla con un bucle `for` y remplazaremos los **valores ausentes** en cada una de las columnas:

```
# recorriendo los nombres de las columnas y reemplazando los va
columns_to_replace=['track','artist','genre']

for column in columns_to_replace:
    df[column]=df[column].fillna('unknown')
    print(column)
```

```
track
artist
genre
```

Ahora vamos a comprobar que la tabla no contiene más valores ausentes, para ello vamos a contar de nuevo los valores ausentes.

```
# contando valores ausentes
df.isna().sum()
```

```
user_id    0
track      0
artist     0
genre      0
city       0
time       0
day        0
dtype: int64
```

Volver a Contenidos

Duplicados

Una vez que hemos tratado los valores ausentes es momento de encontrar el número de **valores duplicados** obvios en la tabla:

```
# contando duplicado obvios
df.duplicated().sum()
```

```
3826
```

Tenemos un total de **3826** valores duplicados. Para resolver esto vamos a llamar a un método `pandas` para deshacernos de los duplicados obvios:

```
# eliminando duplicados obvios
df = df.drop_duplicates().reset_index(drop=True)
```

Contamos los duplicados obvios una vez más para asegurar que todos han sido eliminados:

```
# comprobando duplicados
df.duplicated().sum()
```

```
0
```

Ahora vamos a deshacernos de los **duplicados implícitos** en la columna `genre`. Por ejemplo, el nombre de un género se puede escribir de varias formas. Dichos errores también pueden afectar al resultado.

Para resolver esto vamos a imprimir una lista de nombres únicos de géneros, ordenados en orden alfabético. Cómo se hace:

- Recupera la deseada columna DataFrame
- Aplícale un método de orden
- Para la columna ordenada, llama al método que te devolverá todos los valores de columna únicos

```
# inspeccionando los nombres de géneros únicos
df['genre'].sort_values().unique()
```

```
array(['acid', 'acoustic', 'action', 'adult', 'africa', 'afrikaans',
       'alternative', 'ambient', 'americana', 'animated', 'anime',
       'arabesk', 'arabic', 'arena', 'argentinetango', 'art',
'audiobook',
       'avantgarde', 'axé', 'baile', 'balkan', 'beats', 'bigroom',
       'black', 'bluegrass', 'blues', 'bollywood', 'bossa', 'brazilian',
       'breakbeat', 'breaks', 'broadway', 'cantautori', 'cantopop',
       'canzone', 'caribbean', 'caucasian', 'celtic', 'chamber',
       'children', 'chill', 'chinese', 'choral', 'christian',
'christmas',
       'classical', 'classicmetal', 'club', 'colombian', 'comedy',
       'conjazz', 'contemporary', 'country', 'cuban', 'dance',
       'dancehall', 'dancepop', 'dark', 'death', 'deep', 'deutschrock',
       'deutschspr', 'dirty', 'disco', 'dnb', 'documentary', 'downbeat',
       'downtempo', 'drum', 'dub', 'dubstep', 'eastern', 'easy',
       'electronic', 'electropop', 'emo', 'entehno', 'epicmetal',
       'estrada', 'ethnic', 'eurofolk', 'european', 'experimental',
       'extrememetal', 'fado', 'film', 'fitness', 'flamenco', 'folk',
       'folklore', 'folkmetal', 'folkrock', 'folktronica', 'forró',
       'frankreich', 'französisch', 'french', 'funk', 'future',
'gangsta',
       'garage', 'german', 'ghazal', 'gitarre', 'glitch', 'gospel',
       'gothic', 'grime', 'grunge', 'gypsy', 'handsup', "hard'n'heavy",
       'hardcore', 'hardstyle', 'hardtechno', 'hip', 'hip-hop',
'hiphop',
       'historisch', 'holiday', 'hop', 'horror', 'house', 'idm',
       'independent', 'indian', 'indie', 'indipop', 'industrial',
       'inspirational', 'instrumental', 'international', 'irish', 'jam',
       'japanese', 'jazz', 'jewish', 'jpop', 'jungle', 'k-pop',
       'karadeniz', 'karaoke', 'kayokyoku', 'korean', 'laiko', 'latin',
       'latino', 'leftfield', 'local', 'lounge', 'loungeelectronic',
       'lovers', 'malaysian', 'mandopop', 'marschmusik', 'meditative',
       'mediterranean', 'melodic', 'metal', 'metalcore', 'mexican',
       'middle', 'minimal', 'miscellaneous', 'modern', 'mood', 'mpb',
       'muslim', 'native', 'neoklassik', 'neue', 'new', 'newage',
       'newwave', 'nu', 'nujazz', 'numetal', 'oceania', 'old', 'opera',
       'orchestral', 'other', 'piano', 'pop', 'popelectronic',
       'popeurodance', 'post', 'posthardcore', 'postrock', 'power',
       'progmetal', 'progressive', 'psychedelic', 'punjabi', 'punk',
       'quebecois', 'ragga', 'ram', 'rancheras', 'rap', 'rave',
'reggae',
```

```
          'reggaeton', 'regional', 'relax', 'religious', 'retro', 'rhythm',
          'rnb', 'rnr', 'rock', 'rockabilly', 'romance', 'roots', 'ruspop',
          'rusrap', 'rusrock', 'salsa', 'samba', 'schlager', 'self',
          'sertanejo', 'shoegazing', 'showtunes', 'singer', 'ska', 'slow',
          'smooth', 'soul', 'soulful', 'sound', 'soundtrack', 'southern',
          'specialty', 'speech', 'spiritual', 'sport', 'stonerrock',
    'surf',
          'swing', 'synthpop', 'sängerportrait', 'tango', 'tanzorchester',
          'taraftar', 'tech', 'techno', 'thrash', 'top', 'traditional',
          'tradjazz', 'trance', 'tribal', 'trip', 'triphop', 'tropical',
          'türk', 'türkçe', 'unknown', 'urban', 'uzbek', 'variété', 'vi',
          'videogame', 'vocal', 'western', 'world', 'worldbeat', 'ïîï'],
         dtype=object)
```

Buscando en la lista para encontrar duplicados implícitos del género `hiphop`. Estos pueden ser nombres escritos incorrectamente o nombres alternativos para el mismo género.

Vemos los siguientes duplicados implícitos:

- `hip`
- `hop`
- `hip-hop`

Para deshacernos de ellos, declaramos la función `replace_wrong_genres()` con dos parámetros:

- `wrong_genres=` — la lista de duplicados
- `correct_genre=` — la string con el valor correcto

La función debería corregir los nombres en la columna `'genre'` de la tabla `df`, es decir, remplaza cada valor de la lista `wrong_genres` con el valor en `correct_genre`.

```
# función para reemplazar duplicados implícitos

wrong=['hip','hop','hip-hop']
correct='hiphop'

def replace_wrong_genres(wrong_genres, correct_genre):
    for wrong in wrong_genres:
        df['genre']=df['genre'].replace(wrong, correct)

```

Ahora que ha quedado definida nuestra función vamos a llamarla y pasarle argumentos para que retire los duplicados implícitos (`hip`, `hop` y `hip-hop`) y los reemplace por `hiphop`:

```
# eliminando duplicados implícitos
replace_wrong_genres(wrong,correct)
```

Por último vamos asegurarnos de que los nombres duplicados han sido eliminados, para ello imprimimos la lista de valores únicos de la columna `'genre'`:

```
# revisando en busca de duplicados implícitos
df['genre'].sort_values().unique()
```

```
array(['acid', 'acoustic', 'action', 'adult', 'africa', 'afrikaans',
       'alternative', 'ambient', 'americana', 'animated', 'anime',
       'arabesk', 'arabic', 'arena', 'argentinetango', 'art',
'audiobook',
       'avantgarde', 'axé', 'baile', 'balkan', 'beats', 'bigroom',
       'black', 'bluegrass', 'blues', 'bollywood', 'bossa', 'brazilian',
       'breakbeat', 'breaks', 'broadway', 'cantautori', 'cantopop',
       'canzone', 'caribbean', 'caucasian', 'celtic', 'chamber',
       'children', 'chill', 'chinese', 'choral', 'christian',
'christmas',
       'classical', 'classicmetal', 'club', 'colombian', 'comedy',
       'conjazz', 'contemporary', 'country', 'cuban', 'dance',
       'dancehall', 'dancepop', 'dark', 'death', 'deep', 'deutschrock',
       'deutschspr', 'dirty', 'disco', 'dnb', 'documentary', 'downbeat',
       'downtempo', 'drum', 'dub', 'dubstep', 'eastern', 'easy',
       'electronic', 'electropop', 'emo', 'entehno', 'epicmetal',
       'estrada', 'ethnic', 'eurofolk', 'european', 'experimental',
       'extrememetal', 'fado', 'film', 'fitness', 'flamenco', 'folk',
       'folklore', 'folkmetal', 'folkrock', 'folktronica', 'forró',
       'frankreich', 'französisch', 'french', 'funk', 'future',
'gangsta',
       'garage', 'german', 'ghazal', 'gitarre', 'glitch', 'gospel',
       'gothic', 'grime', 'grunge', 'gypsy', 'handsup', "hard'n'heavy",
       'hardcore', 'hardstyle', 'hardtechno', 'hiphop', 'historisch',
       'holiday', 'horror', 'house', 'idm', 'independent', 'indian',
       'indie', 'indipop', 'industrial', 'inspirational',
'instrumental',
       'international', 'irish', 'jam', 'japanese', 'jazz', 'jewish',
       'jpop', 'jungle', 'k-pop', 'karadeniz', 'karaoke', 'kayokyoku',
       'korean', 'laiko', 'latin', 'latino', 'leftfield', 'local',
       'lounge', 'loungeelectronic', 'lovers', 'malaysian', 'mandopop',
       'marschmusik', 'meditative', 'mediterranean', 'melodic', 'metal',
       'metalcore', 'mexican', 'middle', 'minimal', 'miscellaneous',
       'modern', 'mood', 'mpb', 'muslim', 'native', 'neoklassik',
'neue',
       'new', 'newage', 'newwave', 'nu', 'nujazz', 'numetal', 'oceania',
       'old', 'opera', 'orchestral', 'other', 'piano', 'pop',
       'popelectronic', 'popeurodance', 'post', 'posthardcore',
       'postrock', 'power', 'progmetal', 'progressive', 'psychedelic',
       'punjabi', 'punk', 'quebecois', 'ragga', 'ram', 'rancheras',
'rap',
       'rave', 'reggae', 'reggaeton', 'regional', 'relax', 'religious',
```

```
          'retro', 'rhythm', 'rnb', 'rnr', 'rock', 'rockabilly', 'romance',
          'roots', 'ruspop', 'rusrap', 'rusrock', 'salsa', 'samba',
          'schlager', 'self', 'sertanejo', 'shoegazing', 'showtunes',
          'singer', 'ska', 'slow', 'smooth', 'soul', 'soulful', 'sound',
          'soundtrack', 'southern', 'specialty', 'speech', 'spiritual',
          'sport', 'stonerrock', 'surf', 'swing', 'synthpop',
          'sängerportrait', 'tango', 'tanzorchester', 'taraftar', 'tech',
          'techno', 'thrash', 'top', 'traditional', 'tradjazz', 'trance',
          'tribal', 'trip', 'triphop', 'tropical', 'türk', 'türkçe',
          'unknown', 'urban', 'uzbek', 'variété', 'vi', 'videogame',
    'vocal',
          'western', 'world', 'worldbeat', 'ïîï'], dtype=object)
```

Volver a Contenidos

Conclusiones

Detectamos tres problemas con los datos:

- Estilos de encabezados incorrectos
- Valores ausentes
- Duplicados obvios e implícitos

Los encabezados han sido eliminados para conseguir que el procesamiento de la tabla sea más sencillo.

Todos los valores ausentes han sido reemplazados por `'unknown'`. Pero todavía tenemos que ver si los valores ausentes en `'genre'` afectan a nuestros cálculos.

La ausencia de duplicados hará que los resultados sean mas precisos y fáciles de entender.

Ahora ya podemos continuar probando las hipótesis.

Volver a Contenidos

Etapa 3. Prueba de hipótesis

En nuestra tercer y última etapa vamos a comprobar las hipótesis planteadas de un inicio. Vamos a ello.

Hipótesis 1: comparar el comportamiento del usuario en las dos ciudades

De acuerdo con la primera hipótesis, los usuarios de Springfield y Shelbyville escuchan música de forma distinta. Comprobaremos esto utilizando los datos de tres días de la semana: lunes, miércoles y viernes.

- Dividiremos a los usuarios en grupos por ciudad.
- Compararemos cuántas pistas reprodujeron cada grupo el lunes, el miércoles y el viernes.

Por el bien del ejercicio, realizaremos cada cálculo de forma separada.

Evaluaremos la actividad del usuario en cada ciudad. Luego agruparemos los datos por ciudad y finalmente encontraremos el número de canciones reproducidas en cada grupo.

```
# contando las pistas reproducidas en cada ciudad
city_group = df.groupby('city')['track'].count()
city_group
```

```
city
Shelbyville    18512
Springfield    42741
Name: track, dtype: int64
```

- **Springfield** ha reproducido más pistas que los ciudadanos en **Shelbyville** pero eso no implica que los ciudadanos de Springfield escuchen música más a menudo. Esta ciudad es simplemente más grande y hay más usuarios.

Ahora **agruparemos** los datos por día de la semana y encontraremos el número de pistas reproducidas el lunes, miércoles y viernes.

```
# calculando las pistas reproducidas en cada uno de los tres dí
day_group = df.groupby('day')['track'].count()
day_group
```

```
day
Friday       21840
Monday       21354
Wednesday    18059
Name: track, dtype: int64
```

- El miércoles fue el día más **silencioso** de todos. Pero si consideramos las dos ciudades por separado podríamos llegar a una conclusión diferente.

Ya hemos visto cómo funciona el agrupar por ciudad o día. Ahora escribiremos una **función** que agrupará ambos.

Creamos la función `number_tracks()` para calcular el número de canciones reproducidas en un determinado **día y ciudad**. Requerirá dos parámetros:

- Día de la semana
- Nombre de la ciudad

En la función, utilizaremos una variable para almacenar las filas de la tabla original, donde:

- el valor de la columna `'day'` es igual al parámetro de día
- el valor de la columna `'city'` es igual al parámetro de ciudad

Aplicaremos un **filtrado consecutivo** con **indexación lógica**.

Después, calcularemos los valores de la columna `'user_id'` en la tabla resultante. Almacenaremos el resultado en la nueva variable. Recuperaremos esta variable de la función.

```
# <creando la función number_tracks()>
# declararemos la función con dos parámetros: day=, city=.
# dejamos que la variable track_list almacene las filas df en l
# el valor en la columna 'day' es igual al parámetro day= y, al
# el valor de la columna 'city' es igual al parámetro city= (ap
# con indexación lógica).
# dejamos que la variable track_list_count almacene el número d
# (encontrado con el método count()).
# la función devuelve un número: el valor de track_list_count.

def number_tracks(day, city):
    track_list = df[df['day'] == day]
    track_list = track_list[track_list['city'] == city]
    track_list_count = track_list['user_id'].count()
    return track_list_count


# la función cuenta las pistas reproducidas en un cierto día y
# primero recupera las filas del día deseado de la tabla,
# después filtra las filas de la ciudad deseada del resultado,
# entonces, encuentra el número de valores de 'user_id' en la t
# y devuelve ese número.
# para ver lo que devuelve, hacemos la llamada de la función en
```

Llamamos a `number_tracks()` hasta **seis veces**, cambiando los valores de los parámetros, para que recuperemos los datos de ambas ciudades para cada uno de los tres días.

```
# el número de canciones reproducidas en Springfield el lunes
print(number_tracks('Monday','Springfield'))
```

```
15740
```

```
# el número de canciones reproducidas en Shelbyville el lunes
print(number_tracks('Monday', 'Shelbyville'))
```

```
5614
```

```
# el número de canciones reproducidas en Springfield el miércol
print(number_tracks('Wednesday', 'Springfield'))
```

```
11056
```

```
# el número de canciones reproducidas en Shelbyville el miércol
print(number_tracks('Wednesday', 'Shelbyville'))
```

```
7003
```

```
# el número de canciones reproducidas en Springfield el viernes
print(number_tracks('Friday', 'Springfield'))
```

```
15945
```

```
# el número de canciones reproducidas en Shelbyville el viernes
print(number_tracks('Friday','Shelbyville'))
```

```
5895
```

Ahora vamos a organizar nuestros resultados utilizando `pd.DataFrame` para crear una tabla, donde:

- Los nombres de las columnas son: `['city', 'monday', 'wednesday', 'friday']`
- Los datos son los resultados que conseguiste de `number_tracks()`

```
# Nombres de las columnas y sus valores
header = ['city','monday','wednesday', 'friday']
cities = [['Springfield', 15740, 11056, 15945],['Shelbyville',
```

```
# Tabla con los resultados
table_by_days = pd.DataFrame(data=cities, columns=header)
table_by_days
```

	city	monday	wednesday	friday
0	Springfield	15740	11056	15945
1	Shelbyville	5614	7003	5895

Conclusiones

Los datos revelan las diferencias en el comportamiento de los usuarios:

- En Springfield, el número de canciones reproducidas alcanzan el punto máximo los lunes y viernes mientras que los miércoles hay un descenso de la actividad.
- En el caso de Shelbyville, ocurre lo contario, los usuarios de esta ciudad escuchan más música los miércoles mientras que la actividad de los usuarios los lunes y viernes es menor.

Así que la primera hipótesis parece ser correcta.

Volver a Contenidos

Hipótesis 2: música al principio y al final de la semana

De acuerdo con la **segunda hipótesis**, los lunes por la mañana y los viernes por la noche los ciudadanos de **Springfield** escuchan géneros que difieren de aquellos que los usuarios de **Shelbyville** disfrutan.

Vamos a obtener dos tablas y asegurarnos de que el nombre de las tablas combinadas encaja con el DataFrame dado en los dos bloques:

- Para Springfield — `spr_general`
- Para Shelbyville — `shel_general`

```
# obteniendo la tabla spr_general de las filas de df,
# donde los valores en la columna 'city' es 'Springfield'
```

```
spr_general = df[df['city'] == 'Springfield']
spr_general
```

	user_id	track	artist	genre	city	time	day
1	55204538	Delayed Because of Accident	Andreas Rönnberg	rock	Springfield	14:07:09	Friday
4	E2DC1FAE	Soul People	Space Echo	dance	Springfield	08:34:34	Monday
6	4CB90AA5	True	Roman Messer	dance	Springfield	13:00:07	Wednesday
7	F03E1C1F	Feeling This Way	Polina Griffith	dance	Springfield	20:47:49	Wednesday
8	8FA1D3BE	L'estate	Julia Dalia	ruspop	Springfield	09:17:40	Friday
...	...	...	...	...	...	...	...
61247	83A474E7	I Worship Only What	The Black Dahlia	extrememetal	Springfield	21:07:12	Monday

```
# obteniendo shel_general de las filas df,
# donde el valor de la columna 'city' es 'Shelbyville'
shel_general = df[df['city'] == 'Shelbyville']
shel_general
```

	user_id	track	artist	genre	city	time	
0	FFB692EC	Kamigata To Boots	The Mass Missile	rock	Shelbyville	20:28:33	Wedne
2	20EC38	Funiculì funiculà	Mario Lanza	pop	Shelbyville	20:58:07	Wedne
3	A3DD03C9	Dragons in the Sunset	Fire + Ice	folk	Shelbyville	08:37:09	Mo
5	842029A1	Chains	Obladaet	rusrap	Shelbyville	13:09:41	F
9	E772D5C0	Pessimist	unknown	dance	Shelbyville	21:20:49	Wedne
...	...	...	...	...	...	...	
61239	D94F810B	Theme from the Walking Dead	Proyecto Halloween	film	Shelbyville	21:14:40	Mo
61240	BC8EC5CF	Red Lips: Gta (Rover	Rover	electronic	Shelbyville	21:06:50	Mo

Ahora que ya tenemos nuestras nuevas tablas vamos a escribir la función `genre_weekday()` con cuatro parámetros:

- Una tabla para los datos (`df`)
- El día de la semana (`day`)
- La marca de fecha y hora en formato 'hh:mm' (`time1`)
- La marca de fecha y hora en formato 'hh:mm' (`time2`)

La función debería devolver información de los 15 géneros más populares de un día determinado en un período entre dos marcas de fecha y hora.

```
# declarando la función genre_weekday() con los parámetros day=
# devolver información sobre los géneros más populares de un de

# 1) Dejamos que la variable genre_df almacene las filas que cu
#    - el valor de la columna 'day' es igual al valor del argum
#    - el valor de la columna 'time' es mayor que el valor del
#    - el valor en la columna 'time' es menor que el valor del
#    Utilizaremos un filtrado consecutivo con indexación lógica

# 2) Agrupamos genre_df por la columna 'genre', tomamos una de
#    y utilizamos el método count() para encontrar el número de
#    los géneros representados; almacenamos los Series resultan
#    la variable genre_df_count

```

```
# 3) Ordenamos genre_df_count en orden descendente de frecuenci
#    en la variable genre_df_sorted

# 4) Devolvemos un objeto Series con los primeros 15 valores de
#    géneros más populares (en un determinado día, en un determ

def genre_weekday(df, day, time1, time2):

    # filtrado consecutivo
    # genre_df solo almacenará aquellas filas df en las que el
    genre_df = df[df['day'] == day]

    # genre_df solo almacenará aquellas filas df en las que el
    genre_df = genre_df[df['time'] < time2]

    # genre_df solo almacenará aquellas filas df en las que el
    genre_df = genre_df[df['time'] > time1]

    # agrupamos el DataFrame filtrado por la columna con los no
    genre_df_grouped = genre_df.groupby('genre')['time'].count(

    # ordenamos el resultado en orden descendente (por lo que l
    genre_df_sorted = genre_df_grouped.sort_values(ascending=Fa

    # devolvemos el objeto Series que almacena los 15 géneros m
    return pd.Series(genre_df_sorted[:15])
```

Una vez definida nuestra función **comparamos** los resultados de la función `genre_weekday()` para Springfield y Shelbyville el lunes por la mañana (de 7:00 a 11:00) y el viernes por la tarde (de 17:00 a 23:00):

```
# llamando a la función para el lunes por la mañana en Springfi
genre_weekday(spr_general, 'Monday', '07:00', '11:00')
```

```
<ipython-input-32-655f7a1b3c7e>:28: UserWarning: Boolean Series key will
  genre_df = genre_df[df['time'] < time2]
<ipython-input-32-655f7a1b3c7e>:31: UserWarning: Boolean Series key will
  genre_df = genre_df[df['time'] > time1]
genre
pop            781
dance          549
electronic     480
rock           474
hiphop         286
ruspop         186
world          181
rusrap         175
alternative    164
unknown        161
```

```
classical      157
metal          120
jazz           100
folk            97
soundtrack      95
Name: time, dtype: int64
```

```
# llamando a la función para el lunes por la mañana en Shelbyvi
genre_weekday(shel_general, 'Monday', '07:00', '11:00')
```

```
<ipython-input-32-655f7a1b3c7e>:28: UserWarning: Boolean Series key will
  genre_df = genre_df[df['time'] < time2]
<ipython-input-32-655f7a1b3c7e>:31: UserWarning: Boolean Series key will
  genre_df = genre_df[df['time'] > time1]
genre
pop            218
dance          182
rock           162
electronic     147
hiphop          80
ruspop          64
alternative     58
rusrap          55
jazz            44
classical       40
world           36
rap             32
soundtrack      31
rnb             27
metal           27
Name: time, dtype: int64
```

```
# llamando a la función para el viernes por la tarde en Springf
genre_weekday(spr_general, 'Friday', '17:00', '23:00')
```

```
<ipython-input-32-655f7a1b3c7e>:28: UserWarning: Boolean Series key will
  genre_df = genre_df[df['time'] < time2]
<ipython-input-32-655f7a1b3c7e>:31: UserWarning: Boolean Series key will
  genre_df = genre_df[df['time'] > time1]
genre
pop            713
rock           517
dance          495
electronic     482
hiphop         273
world          208
ruspop         170
classical      163
alternative    163
rusrap         142
jazz           111
unknown        110
soundtrack     105
```

```
rnb            90
metal          88
Name: time, dtype: int64
```

```
# llamando a la función para el viernes por la tarde en Shelbyv
genre_weekday(shel_general, 'Friday', '17:00', '23:00')
```

```
<ipython-input-32-655f7a1b3c7e>:28: UserWarning: Boolean Series key will
  genre_df = genre_df[df['time'] < time2]
<ipython-input-32-655f7a1b3c7e>:31: UserWarning: Boolean Series key will
  genre_df = genre_df[df['time'] > time1]
genre
pop            256
rock           216
electronic     216
dance          210
hiphop          97
alternative     63
jazz            61
classical       60
rusrap          59
world           54
unknown         47
ruspop          47
soundtrack      40
metal           39
rap             36
Name: time, dtype: int64
```

Conclusión

Habiendo comparado los 15 géneros más populares del lunes por la mañana podemos concluir lo siguiente:

1. Los usuarios de Springfield y Shelbyville escuchan música diferente los lunes por la mañana.
2. Los 3 géneros de música más populares entre los usuarios de Springfield son:

- pop
- dance
- electrónica

3. Los 3 géneros musicales más populares entre los usuarios de Shelbyville fueron:

- pop
- dance
- rock

Para el **viernes** por la tarde la situación cambia.

4. Los 3 géneros más populares en Springfield son:

- pop
- rock
- dance

5. Los 3 géneros más populares en Shelbyville son:

- pop
- rock
- electrónica

De esta forma, fallamos en aceptar la segunda hipótesis. Los usuarios de Springfield escuchan música diferente a lo que escuchan los usuarios de Shelbyville.

Volver a Contenidos

Hipótesis 3: preferencias de género en Springfield y Shelbyville

Hipótesis: Shelbyville ama la música rap. A los ciudadanos de Springfield les gusta más el pop.

Para comprobar esto vamos agrupar la tabla `spr_general` por género y encontraremos el número de canciones reproducidas de cada género con el método `count()`. Después, ordenaremos el resultado en orden descendente y lo guardaremos en la variable `spr_genres`.

```
# en una línea: agrupamos la tabla spr_general por la columna '
# contamos los valores 'genre' con count() en la agrupación,
# ordenamos el objeto series resultante en orden descendiente,

spr_genres = spr_general.groupby('genre').count()
spr_genres = spr_genres['track'].sort_values(ascending = False)
```

Visualizamos las 10 primeras filas de `spr_genres`:

```
# imprimiendo las 10 primeras filas de spr_genres
spr_genres.head(10)
```

```
genre
pop            5892
dance          4435
rock           3965
```

```
electronic     3786
hiphop         2096
classical      1616
world          1432
alternative    1379
ruspop         1372
rusrap         1161
Name: track, dtype: int64
```

Ahora vamos hacer lo mismo con los datos de **Shelbyville**.

Agrupamos la tabla `shel_general` por género y encontramos el número de canciones reproducidas de cada género. Después, ordenamos el resultado en orden descendente y lo guardamos en la tabla `shel_genres`:

```
# Agrupamos, contamos y ordenamos la tabla
shel_genres = shel_general.groupby('genre').count()
shel_genres = shel_genres['track'].sort_values(ascending = Fals
```

Imprimimos las 10 primeras filas de `shel_genres`:

```
# imprimiendo las 10 primeras filas de shel_genres
shel_genres.head(10)
```

```
genre
pop            2431
dance          1932
rock           1879
electronic     1736
hiphop          960
alternative     649
classical       646
rusrap          564
ruspop          538
world           515
Name: track, dtype: int64
```

Conclusión

La hipótesis ha sido **parcialmente** demostrada:

- La música **pop** es el género más popular en Springfield, tal como se esperaba.
- Sin embargo, la música **pop** ha resultado ser igual de popular en Springfield que en Shelbyville y el **rap** no estaba entre los 5 más populares en ninguna de las ciudades.

Volver a Contenidos

Conclusiones

Hemos probado las siguientes tres hipótesis:

1. La actividad de los usuarios difiere dependiendo del día de la semana y de las distintas ciudades.
2. Los lunes por la mañana los residentes de Springfield y Shelbyville escuchan géneros distintos. Lo mismo ocurre con los viernes por la noche.
3. Los oyentes de Springfield y Shelbyville tienen distintas preferencias. En ambas