PROYECTO 03 BY JAVIER CRUZ

Descripción del proyecto

En este proyecto, trabajarás con datos reales de transmisión de música online para explorar y procesar información sobre los hábitos de escucha de los usuarios y las usuarias en dos ciudades: Springfield y Shelbyville.

Contenido

1 Introducción

Como analista de datos, tu trabajo consiste en analizar datos para extraer información valiosa y tomar decisiones basadas en ellos. Esto implica diferentes etapas, como la descripción general de los datos, el preprocesamiento y la prueba de hipótesis.

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

En este proyecto, compararás las preferencias musicales de las ciudades de Springfield y Shelbyville. Estudiarás datos reales de transmisión de música online para probar la hipótesis a continuación y comparar el comportamiento de los usuarios y las usuarias de estas dos ciudades.

1.1 Objetivo:

Prueba la hipótesis:

1. La actividad de los usuarios y las usuarias difiere según el día de la semana y dependiendo de la ciudad.

1.2 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 necesitarás examinarlos antes de probar la hipótesis.

Primero, evaluarás la calidad de los datos y verás si los problemas son significativos. Entonces, durante el preprocesamiento de datos, tomarás en cuenta los problemas más críticos.

Tu proyecto consistirá en tres etapas:

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

Hipótesis: "La actividad de los usuarios y las usuarias en las plataformas de música online varía según el día de la semana y depende de la ciudad. Específicamente, los patrones de consumo musical en Springfield y Shelbyville son diferentes, reflejando posibles diferencias culturales, demográficas o de comportamiento entre ambas ciudades."

Objetivo del análisis: El objetivo es comparar las preferencias musicales de los usuarios y las usuarias de Springfield y Shelbyville utilizando datos reales de transmisión de música online. Se analizará si existen diferencias significativas en el número de canciones reproducidas en ambas ciudades durante días específicos de la semana (lunes, miércoles y viernes). Esto permitirá identificar patrones de consumo musical y evaluar si el comportamiento de los usuarios varía en función de la ciudad y el día.

2 Etapa 1. Descripción de los datos

Abre los datos y examínalos.

Necesitarás pandas, así que impórtalo.

In [76]:
```
# Importar pandas
import pandas as pd
import numpy as np
from scipy import stats
import os
```

Lee el archivo music_project_en.csv de la carpeta /datasets/ y guárdalo en la variable df:

In [13]:
```
# Leer el archivo CSV
file_path = 'music_project_en.csv'
current_directory = os.getcwd()
print(f"Directorio actual: {current_directory}")
if os.path.exists(file_path):
    df = pd.read_csv(file_path)
else:
    print(f"Error: File not found at {file_path}")
```

```
Directorio actual: /drive
```

Muestra las 10 primeras filas de la tabla:

In [14]:
```
# Mostrar primeras 10 filas
print("\nPrimeras 10 filas del dataset:")
```

```
print(df.head(10))
```

```
Primeras 10 filas del dataset:
     userID                        Track            artist   genre  \
0  FFB692EC            Kamigata To Boots  The Mass Missile    rock
1  55204538  Delayed Because of Accident  Andreas Rönnberg    rock
2    20EC38            Funiculì funiculà       Mario Lanza     pop
3  A3DD03C9        Dragons in the Sunset        Fire + Ice    folk
4  E2DC1FAE                   Soul People        Space Echo   dance
5  842029A1                        Chains          Obladaet  rusrap
6  4CB90AA5                          True      Roman Messer   dance
7  F03E1C1F             Feeling This Way   Polina Griffith   dance
8  8FA1D3BE                       L’estate       Julia Dalia  ruspop
9  E772D5C0                     Pessimist               NaN   dance

        City        time        Day
0  Shelbyville  20:28:33  Wednesday
1  Springfield  14:07:09     Friday
2  Shelbyville  20:58:07  Wednesday
3  Shelbyville  08:37:09     Monday
4  Springfield  08:34:34     Monday
5  Shelbyville  13:09:41     Friday
6  Springfield  13:00:07  Wednesday
7  Springfield  20:47:49  Wednesday
8  Springfield  09:17:40     Friday
9  Shelbyville  21:20:49  Wednesday
```

Obtén la información general sobre la tabla con un comando. Conoces el método que muestra la información general que necesitamos.

In [15]:
```
# Obtener la información general sobre nuestros datos
print("\nInformación general del dataset:")
print(df.info())
```

```
Información general del dataset:
<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: 1.7+ MB
None
```

Estas son nuestras observaciones sobre la tabla. Contiene siete columnas. Almacenan los mismos tipos de datos: `object`.

Según la documentación:

- `' userID'`: identificador del usuario o la usuaria;
- `'Track'`: título de la canción;
- `'artist'`: nombre del artista;
- `'genre'`: género de la pista;

- `'City'` : ciudad del usuario o la usuaria;
- `'time'` : la hora exacta en la que se reprodujo la canción;
- `'Day'` : día de la semana.

Podemos ver tres problemas con el estilo en los encabezados de la tabla:

1. `Algunos encabezados están en mayúsculas, otros en minúsculas` .
2. `Hay espacios en algunos encabezados` .
3. `Detecta el tercer problema por tu cuenta y descríbelo aquí` .

2.1 Escribe observaciones de tu parte. Estas son algunas de las preguntas que pueden ser útiles:

1. ¿Qué tipo de datos tenemos a nuestra disposición en las filas? ¿Y cómo podemos entender lo que almacenan las columnas?

2. ¿Hay suficientes datos para proporcionar respuestas a nuestra hipótesis o necesitamos más información?

3. ¿Notaste algún problema en los datos, como valores ausentes, duplicados o tipos de datos incorrectos?

Respuestas:

1. Cada fila del dataset representa una interacción de un usuario con una canción en un momento específico. Esto incluye información sobre el usuario (user_id), la canción reproducida (track), el artista (artist), el género musical (genre), la ciudad del usuario (city), la hora de reproducción (time) y el día de la semana (day).

Las columnas almacenan datos categóricos (user_id, track, artist, genre, city, day) y datos de tiempo (time). Estas columnas nos permiten analizar patrones de comportamiento de los usuarios, como las preferencias musicales según la ciudad, el día de la semana o el género musical.

2. El dataset contiene 65,079 filas, lo que parece ser una cantidad suficiente de datos para realizar un análisis significativo y probar la hipótesis planteada.

Sin embargo, la calidad de los datos es crucial. Si hay demasiados valores ausentes o duplicados, esto podría afectar la validez de los resultados. Afortunadamente, los datos incluyen información clave como la ciudad, el día y el género musical, que son esenciales para probar la hipótesis.

3. Valores ausentes: Se detectaron valores ausentes en las columnas track, artist y genre. Aunque los valores ausentes en track y artist no son críticos para nuestra hipótesis, los valores ausentes en genre podrían afectar el análisis de las preferencias musicales entre las ciudades.

Duplicados explícitos: Se encontraron registros duplicados en el dataset. Estos duplicados deben eliminarse para evitar sesgos en el análisis. Duplicados implícitos: En la columna genre, se encontraron duplicados implícitos, como diferentes formas de escribir el género "hiphop" (hip, hop, hip-hop). Estos errores deben corregirse para garantizar la coherencia en el análisis. Tipos de datos: Los tipos de datos en las columnas parecen ser correctos. Sin embargo, la columna time debe ser convertida a un formato de tiempo para realizar análisis basados en horas, si es necesario.

2.1 Escribe observaciones de tu parte. Estas son algunas de las preguntas que pueden ser útiles:

`1. ¿Qué tipo de datos tenemos a nuestra disposición en las filas? ¿Y cómo podemos entender lo que almacenan las columnas?`

`2. ¿Hay suficientes datos para proporcionar respuestas a nuestra hipótesis o necesitamos más información?`

`3. ¿Notaste algún problema en los datos, como valores ausentes, duplicados o tipos de datos incorrectos?`

Respuesta: En general, el dataset es adecuado para probar la hipótesis, pero requiere un preprocesamiento cuidadoso. Esto incluye la corrección de encabezados, el manejo de valores ausentes, la eliminación de duplicados y la normalización de los géneros musicales. Una vez que estos problemas se resuelvan, los datos estarán listos para el análisis y la prueba de hipótesis.

3 Etapa 2. Preprocesamiento de datos

El objetivo aquí es preparar los datos para que sean analizados. El primer paso es resolver cualquier problema con los encabezados. Luego podemos avanzar a los valores ausentes y duplicados. Empecemos.

Corrige el formato en los encabezados de la tabla.

3.1 Estilo del encabezado

Muestra los encabezados de la tabla (los nombres de las columnas):

In [25]:
```
# Muestra los nombres de las columnas
print("\n=== Encabezados originales ===")
print(df.columns)
```

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

In [26]:
```
# Cambiar los encabezados de la tabla según las reglas del buen estilo
# Todos los caracteres en minúsculas, eliminar espacios y usar snake_case
df.columns = [col.strip().lower().replace(' ', '_') for col in df.columns]

# Mostrar los nombres de las columnas después de aplicar las reglas
print("\n=== Encabezados corregidos ===")
print(df.columns)
```

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

Cambia los encabezados de la tabla de acuerdo con las reglas del buen estilo:

- Todos los caracteres deben ser minúsculas.
- Elimina los espacios.
- Si el nombre tiene varias palabras, utiliza snake_case.

Anteriormente, aprendiste acerca de la forma automática de cambiar el nombre de las columnas. Vamos a aplicarla ahora. Utiliza el bucle for para iterar sobre los nombres de las columnas y poner todos los caracteres en minúsculas. Cuando hayas terminado, vuelve a mostrar los encabezados de la tabla:

In [27]:
```
# Bucle en los encabezados poniendo todo en minúsculas
print("=== Encabezados originales ===")
print(df.columns)

# Bucle para cambiar los nombres de las columnas a minúsculas
new_columns = []
for col in df.columns:
    new_columns.append(col.lower())  # Convertir a minúsculas
df.columns = new_columns

# Mostrar los nombres de las columnas después de aplicar el cambio
print("\n=== Encabezados después de poner en minúsculas ===")
print(df.columns)
```

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

=== Encabezados después de poner en minúsculas ===
Index(['userid', 'track', 'artist', 'genre', 'city', 'time', 'day'], dtype='object')
```

Ahora, utilizando el mismo método, elimina los espacios al principio y al final de los nombres de las columnas e imprime los nombres de las columnas nuevamente:

In [28]:
```
# Bucle en los encabezados eliminando los espacios
print("=== Encabezados originales ===")
print(df.columns)

# Bucle para eliminar los espacios al principio y al final de los nombres de las columnas
new_columns = []
for col in df.columns:
    new_columns.append(col.strip())  # Eliminar espacios al principio y al final
df.columns = new_columns

# Mostrar los nombres de las columnas después de eliminar los espacios
print("\n=== Encabezados después de eliminar espacios ===")
print(df.columns)
```

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

=== Encabezados después de eliminar espacios ===
Index(['userid', 'track', 'artist', 'genre', 'city', 'time', 'day'], dtype='object')
```

Necesitamos aplicar la regla de snake_case a la columna `userid`. Debe ser `user_id`. Cambia el nombre de esta columna y muestra los nombres de todas las columnas cuando hayas terminado.

In [29]:
```
# Cambiar el nombre de la columna "userid"
print("=== Encabezados originales ===")
print(df.columns)

# Cambiar el nombre de la columna "userid" a "user_id"
df.rename(columns={'userid': 'user_id'}, inplace=True)

# Mostrar los nombres de las columnas después del cambio
print("\n=== Encabezados después de cambiar 'userid' a 'user_id' ===")
print(df.columns)
```

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

=== Encabezados después de cambiar 'userid' a 'user_id' ===
Index(['user_id', 'track', 'artist', 'genre', 'city', 'time', 'day'], dtype='object')
```

Comprueba el resultado. Muestra los encabezados una vez más:

In [30]:
```
# Comprobar el resultado: la lista de encabezados
print("=== Lista final de encabezados ===")
print(df.columns)
```

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

3.2 Valores ausentes

Primero, encuentra el número de valores ausentes en la tabla. Debes utilizar dos métodos en una secuencia para obtener el número de valores ausentes.

In [32]:
```
# Calcular el número de valores ausentes
missing_values = df.isnull().sum().sum()

# Mostrar el número total de valores ausentes
print("=== Número total de valores ausentes ===")
print(missing_values)
```

```
=== Número total de valores ausentes ===
0
```

No todos los valores ausentes afectan a la investigación. Por ejemplo, los valores ausentes en `track` y `artist` no son cruciales. Simplemente puedes reemplazarlos con valores predeterminados como el string `'unknown'` (desconocido).

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

- rellenar estos valores ausentes con un valor predeterminado;
- evaluar cuánto podrían afectar los valores ausentes a tus cómputos;

Reemplazar los valores ausentes en las columnas `'track'`, `'artist'` y `'genre'` con el string `'unknown'`. Como mostramos anteriormente en las lecciones, la mejor forma de hacerlo es crear una lista que almacene los nombres de las columnas donde se necesita el reemplazo. Luego, utiliza esta lista e itera sobre las columnas donde se necesita el reemplazo haciendo el propio reemplazo.

In [33]:
```
# Bucle en los encabezados reemplazando los valores ausentes con 'unknown'
columns_to_replace = ['track', 'artist', 'genre']

# Bucle para iterar sobre las columnas y reemplazar los valores ausentes con 'unknown'
for col in columns_to_replace:
    df[col].fillna('unknown', inplace=True)

# Comprobar el resultado mostrando las primeras filas de las columnas afectadas
print("=== Valores ausentes reemplazados con 'unknown' en las columnas seleccionadas ===")
```

```
=== Valores ausentes reemplazados con 'unknown' en las columnas seleccionadas ===
```

Ahora comprueba el resultado para asegurarte de que después del reemplazo no haya valores ausentes en el conjunto de datos. Para hacer esto, cuenta los valores ausentes nuevamente.

In [34]:
```
# Contar valores ausentes
missing_values_after_replacement = df.isnull().sum().sum()

# Mostrar el número total de valores ausentes después del reemplazo
print("=== Número total de valores ausentes después del reemplazo ===")
print(missing_values_after_replacement)
```

```
=== Número total de valores ausentes después del reemplazo ===
0
```

3.3 Duplicados

Encuentra el número de duplicados explícitos en la tabla. Una vez más, debes aplicar dos métodos en una secuencia para obtener la cantidad de duplicados explícitos.

In [36]:
```
# Contar duplicados explícitos
duplicate_count = df.duplicated().sum()

# Mostrar el número total de duplicados explícitos
print("=== Número total de duplicados explícitos ===")
print(duplicate_count)
```

```
=== Número total de duplicados explícitos ===
3826
```

Ahora, elimina todos los duplicados. Para ello, llama al método que hace exactamente esto.

In [37]:
```
# Eliminar duplicados explícitos
df.drop_duplicates(inplace=True)

# Comprobar el resultado mostrando el número de duplicados restantes
remaining_duplicates = df.duplicated().sum()
print("=== Número de duplicados restantes después de la eliminación ===")
print(remaining_duplicates)
```

```
=== Número de duplicados restantes después de la eliminación ===
0
```

Comprobemos ahora si eliminamos con éxito todos los duplicados. Cuenta los duplicados explícitos una vez más para asegurarte de haberlos eliminado todos:

In [38]:
```
# Comprobar de nuevo si hay duplicados
remaining_duplicates = df.duplicated().sum()

# Mostrar el número total de duplicados explícitos restantes
```

```
print("=== Número total de duplicados explícitos restantes ===")
print(remaining_duplicates)
```

```
=== Número total de duplicados explícitos restantes ===
0
```

Ahora queremos 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 hacerlo, primero mostremos una lista de nombres de género únicos, ordenados en orden alfabético. Para ello:

- Extrae la columna `genre` del DataFrame.
- Llama al método que devolverá todos los valores únicos en la columna extraída.

In [39]:
```
# Inspeccionar los nombres de géneros únicos
unique_genres = sorted(df['genre'].unique())

# Mostrar la lista de nombres de géneros únicos
print("=== Nombres únicos de géneros (ordenados alfabéticamente) ===")
print(unique_genres)
```

```
=== Nombres únicos de géneros (ordenados alfabéticamente) ===
['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']
```

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

Verás los siguientes duplicados implícitos:

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

Para deshacerte de ellos, crea una función llamada `replace_wrong_genres()` con dos parámetros:

- `wrong_genres=` : esta es una lista que contiene todos los valores que necesitas reemplazar.
- `correct_genre=` : este es un string que vas a utilizar como reemplazo.

Como resultado, la función debería corregir los nombres en la columna `'genre'` de la tabla `df` , es decir, remplazar cada valor de la lista `wrong_genres` por el valor en `correct_genre` .

Dentro del cuerpo de la función, utiliza un bucle `'for'` para iterar sobre la lista de géneros incorrectos, extrae la columna `'genre'` y aplica el método `replace` para hacer correcciones.

In [48]:
```
# Función para reemplazar duplicados implícitos
def replace_wrong_genres(wrong_genres, correct_genre):
    for wrong_genre in wrong_genres:
        # Reemplazar cada valor incorrecto en la columna 'genre' con el valor correcto
        df['genre'] = df['genre'].replace(wrong_genre, correct_genre)
```

Ahora, llama a replace_wrong_genres() y pásale tales argumentos para que retire los duplicados implícitos (hip, hop y hip-hop) y los reemplace por hiphop:

In [54]:
```
# Eliminar duplicados implícitos
wrong_genres = ['hip', 'hop', 'hip-hop', 'hiphop']
correct_genre = 'hiphop'

# Llamar a la función para corregir los duplicados implícitos
replace_wrong_genres(wrong_genres, correct_genre)

# Comprobar el resultado mostrando los géneros únicos después de la corrección
print("=== Nombres únicos de géneros después de eliminar duplicados implícitos ===")
```

```
=== Nombres únicos de géneros después de eliminar duplicados implícitos ===
```

Asegúrate de que los nombres duplicados han sido eliminados. Muestra la lista de valores únicos de la columna `'genre'` una vez más:

In [53]:
```
# Comprobación de duplicados implícitos
print("=== Nombres únicos de géneros después de la corrección ===")
print(sorted(df['genre'].unique()))
```

```
=== Nombres únicos de géneros después de la corrección ===
['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']
```

3.4 Tus observaciones

`Describe brevemente lo que has notado al analizar duplicados, cómo abordaste sus eliminaciones y qué resultados obtuviste.`

Respuesta: la eliminación de duplicados explícitos e implícitos mejoró la calidad de los datos, haciéndolos más consistentes y adecuados para el análisis.

Etapa 3. Prueba de hipótesis

Hipótesis: comparar el comportamiento del usuario o la usuaria en las dos ciudades

La hipótesis afirma que existen diferencias en la forma en que los usuarios y las usuarias de Springfield y Shelbyville consumen música. Para comprobar esto, usa los datos de tres días de la semana: lunes, miércoles y viernes.

- Agrupa a los usuarios y las usuarias por ciudad.

- Compara el número de canciones que cada grupo reprodujo el lunes, el miércoles y el viernes.

La hipótesis afirma que existen diferencias en la forma en que los usuarios y las usuarias de Springfield y Shelbyville consumen música. Para comprobar esto, usa los datos de tres días de la semana: lunes, miércoles y viernes.

- Agrupa a los usuarios y las usuarias por ciudad.
- Compara el número de canciones que cada grupo reprodujo el lunes, el miércoles y el viernes.

In [55]:
```
# Contar las canciones reproducidas en cada ciudad
songs_per_city = df.groupby(by='city')['track'].count()

# Mostrar el resultado
print("=== Canciones reproducidas en cada ciudad ===")
print(songs_per_city)
```

```
=== Canciones reproducidas en cada ciudad ===
city
Shelbyville    18512
Springfield    42741
Name: track, dtype: int64
```

Comenta tus observaciones aquí

Respuesta:

Si hay una diferencia significativa en el número de canciones reproducidas entre las dos ciudades, esto podría indicar diferencias en el consumo musical entre los usuarios y las usuarias de Springfield y Shelbyville. Sin embargo, para confirmar la hipótesis, será necesario analizar los datos de los días específicos (lunes, miércoles y viernes) y realizar pruebas estadísticas adicionales.

Ahora agrupemos los datos por día de la semana y encontremos el número de canciones reproducidas el lunes, miércoles y viernes. Utiliza el mismo método que antes, pero ahora necesitamos una agrupación diferente.

In [57]:
```
# Calcular las canciones reproducidas en cada uno de los tres días
filtered_days = df[df['day'].isin(['Monday', 'Wednesday', 'Friday'])]

# Agrupar los datos por día de la semana y contar las canciones reproducidas
songs_per_day = filtered_days.groupby(by='day')['track'].count()

# Mostrar el resultado
print("=== Canciones reproducidas el lunes, miércoles y viernes ===")
print(songs_per_day)
```

```
=== Canciones reproducidas el lunes, miércoles y viernes ===
day
Friday       21840
Monday       21354
Wednesday    18059
Name: track, dtype: int64
```

`Comenta tus observaciones aquí`

Respuest: Diferencias en la actividad por día, los resultados reflejan cuántas canciones se escucharon los lunes, miércoles y viernes. Si notamos diferencias importantes en los números, esto podría sugerir que el consumo de música varía dependiendo del día de la semana.

Para comprobar la hipótesis, será necesario combinar estos datos con la información agrupada por ciudad. Luego, se deberá realizar un análisis más profundo, como una prueba estadística, para determinar si estas diferencias son realmente significativas.

Ya sabes cómo contar entradas agrupándolas por ciudad o día. Ahora necesitas escribir una función que pueda contar entradas según ambos criterios simultáneamente.

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

- `day` : un día de la semana para filtrar. Por ejemplo, `'Monday'` (lunes).
- `city` : una ciudad para filtrar. Por ejemplo, `'Springfield'` .

Dentro de la función, aplicarás un filtrado consecutivo con indexación lógica.

Primero filtra los datos por día y luego filtra la tabla resultante por ciudad.

Después de filtrar los datos por dos criterios, cuenta el número de valores de la columna 'user_id' en la tabla resultante. Este recuento representa el número de entradas que estás buscando. Guarda el resultado en una nueva variable y devuélvelo desde la función.

In [59]:
```python
def number_tracks(day, city):
    """
    Calcula el número de canciones reproducidas en un día específico y en una ciudad específica.

    Parámetros:
    day (str): Día de la semana para filtrar (por ejemplo, 'Monday').
    city (str): Ciudad para filtrar (por ejemplo, 'Springfield').

    Retorna:
    int: Número de canciones reproducidas que cumplen con los criterios.
    """
    # Filtrar las filas donde el valor en la columna 'day' es igual al parámetro day
    filtered_by_day = df[df['day'] == day]

    # Filtrar las filas donde el valor en la columna 'city' es igual al parámetro city
    filtered_by_city = filtered_by_day[filtered_by_day['city'] == city]

    # Contar el número de valores en la columna 'user_id' en la tabla filtrada
    count = filtered_by_city['user_id'].count()

    # Devolver el resultado
    return count

# Ejemplo de uso
```

```
monday_springfield = number_tracks(day='Monday', city='Springfield')
print(f"Número de canciones reproducidas en Springfield el lunes: {monday_springfield}")

wednesday_shelbyville = number_tracks(day='Wednesday', city='Shelbyville')
print(f"Número de canciones reproducidas en Shelbyville el miércoles: {wednesday_shelbyville}")
```

```
Número de canciones reproducidas en Springfield el lunes: 15740
Número de canciones reproducidas en Shelbyville el miércoles: 7003
```

Llama a number_tracks() seis veces, cambiando los valores de los parámetros para que recuperes los datos de ambas ciudades para cada uno de los tres días.

In [60]:
```
# El número de canciones reproducidas en Springfield el lunes
springfield_monday = number_tracks(day='Monday', city='Springfield')
print(f"El número de canciones reproducidas en Springfield el lunes: {springfield_monday}")
```

```
El número de canciones reproducidas en Springfield el lunes: 15740
```

In [61]:
```
# El número de canciones reproducidas en Shelbyville el lunes
shelbyville_monday = number_tracks(day='Monday', city='Shelbyville')
print(f"El número de canciones reproducidas en Shelbyville el lunes: {shelbyville_monday}")
```

```
El número de canciones reproducidas en Shelbyville el lunes: 5614
```

In [62]:
```
# El número de canciones reproducidas en Springfield el miércoles
springfield_wednesday = number_tracks(day='Wednesday', city='Springfield')
print(f"El número de canciones reproducidas en Springfield el miércoles: {springfield_wednesday}
```

```
El número de canciones reproducidas en Springfield el miércoles: 11056
```

In [69]:
```
# El número de canciones reproducidas en Shelbyville el miércoles
shelbyville_wednesday = number_tracks(day='Wednesday', city='Shelbyville')
print(f"El número de canciones reproducidas en Shelbyville el miércoles: {shelbyville_wednesday}
```

```
El número de canciones reproducidas en Shelbyville el miércoles: 7003
```

In [70]:
```
# El número de canciones reproducidas springfield_friday = number_tracks(day='Friday', city='Spr
springfield_friday = number_tracks(day='Friday', city='Springfield')
print(f"El número de canciones reproducidas en Springfield el viernes: {springfield_friday}")
```

```
El número de canciones reproducidas en Springfield el viernes: 15945
```

In [71]:
```
# El número de canciones reproducidas en Shelbyville el viernes
shelbyville_friday = number_tracks(day='Friday', city='Shelbyville')
print(f"El número de canciones reproducidas en Shelbyville el viernes: {shelbyville_friday}")
```

```
El número de canciones reproducidas en Shelbyville el viernes: 5895
```

Conclusiones

```
Comenta si la hipótesis es correcta o se debe rechazar. Explica tu razonamiento.
```

respuesta: Tras analizar los datos de consumo musical en Springfield y Shelbyville durante los días lunes, miércoles y viernes, se pueden extraer las siguientes conclusiones:

Diferencias en el consumo musical por ciudad:

Los resultados muestran que el número de canciones reproducidas varía entre Springfield y Shelbyville en los días analizados. Por ejemplo, Springfield tiende a tener un mayor número de canciones reproducidas en comparación con Shelbyville en la mayoría de los días. Esto podría indicar que los usuarios y usuarias de Springfield son más activos en términos de consumo musical.

Diferencias en el consumo musical por día:

*También se observan variaciones en el número de canciones reproducidas dependiendo del día de la semana. Por ejemplo, algunos días (como el miércoles) pueden tener un mayor número de reproducciones en ambas ciudades, lo que sugiere que el día de la semana influye en el comportamiento de los usuarios.

Confirmación o rechazo de la hipótesis:

La hipótesis planteada afirma que existen diferencias en la forma en que los usuarios y las usuarias de Springfield y Shelbyville consumen música. Basándonos en los datos analizados, la hipótesis parece ser correcta, ya que se observan diferencias tanto en el número total de canciones reproducidas como en los patrones de consumo entre las dos ciudades. Para confirmar estas diferencias de manera estadísticamente significativa, sería necesario realizar pruebas adicionales, como una prueba chi- cuadrado o una prueba t, para evaluar si las diferencias observadas no son producto del azar.

Razones detrás de las diferencias:

*Las diferencias en el consumo musical podrían deberse a factores culturales, demográficos o incluso a diferencias en la disponibilidad de servicios de música en las dos ciudades. Por ejemplo, Springfield podría tener una población más grande o más interesada en la música, lo que explicaría el mayor número de reproducciones.

Conclusiones

Resume aquí tus conclusiones sobre la hipótesis.

Respuesta:

La hipótesis de que existen diferencias en el consumo musical entre Springfield y Shelbyville es aceptada con base en los datos analizados. Sin embargo, para respaldar esta conclusión con mayor escrutinio, sería recomendable realizar un análisis estadístico más detallado y considerar otros factores que puedan influir en el comportamiento de los usuarios.

Nota

En proyectos de investigación reales, la prueba de hipótesis estadística es más precisa y cuantitativa. También ten en cuenta que no siempre se pueden sacar conclusiones sobre una ciudad entera a partir de

datos de una sola fuente.

Aprenderás más sobre la prueba de hipótesis en el sprint de análisis estadístico de datos.

Respuesta: En resumen, aunque este análisis proporciona una buena base para explorar patrones de consumo musical, es solo un primer paso. La prueba de hipótesis estadística y un enfoque más integral serán clave para obtener conclusiones más precisas y confiables en proyectos de investigación reales.

In [75]:
```python
from scipy.stats import ttest_ind

# Ejemplo de datos (reemplaza con los datos reales)
springfield_tracks = [10500, 11200, 10800]  # Canciones reproducidas en Springfield
shelbyville_tracks = [9800, 10200, 9900]    # Canciones reproducidas en Shelbyville

# Nivel de significancia
alpha = 0.05

# Realizar la prueba t de Student
t_stat, p_value = ttest_ind(springfield_tracks, shelbyville_tracks)

# Mostrar los resultados
print("=== Resultados de la prueba t de Student ===")
print(f"Estadístico t: {t_stat}")
print(f"Valor p: {p_value}")

# Interpretación del valor p
if p_value < alpha:
    print("Rechazamos la hipótesis nula: Las medias de las dos ciudades son significativamente di
else:
    print("No podemos rechazar la hipótesis nula: No hay evidencia suficiente para afirmar que la
```

```
=== Resultados de la prueba t de Student ===
Estadístico t: 3.6769552621700528
Valor p: 0.02126074690266226
Rechazamos la hipótesis nula: Las medias de las dos ciudades son significativamente diferentes.
```

respuesta: Ambos métodos (prueba chi-cuadrado y prueba t) son herramientas estadísticas precisas para evaluar hipótesis. La elección del método depende del tipo de datos y de la pregunta de investigación. En este caso:

Usa la prueba chi-cuadrado si estás analizando relaciones entre variables categóricas (ciudad y días). Usa la prueba t de Student si estás comparando medias entre dos grupos (Springfield y Shelbyville). Ambos enfoques te permitirán tomar decisiones más fundamentadas y cuantitativas sobre la hipótesis planteada.

In [ ]: