¡Hola, Carlos!

Mi nombre es Carlos Ortiz, soy code reviewer de Practicum y voy a revisar el proyecto que acabas de desarrollar.

Cuando vea un error la primera vez, lo señalaré. Deberás encontrarlo y arreglarlo. La intención es que te prepares para un espacio real de trabajo. En un trabajo, el líder de tu equipo hará lo mismo. Si no puedes solucionar el error, te daré más información en la próxima ocasión.

Encontrarás mis comentarios más abajo - **por favor, no los muevas, no los modifiques ni los borres.**

¿Cómo lo voy a hacer? Voy a leer detenidamente cada una de las implementaciones que has llevado a cabo para cumplir con lo solicitado. Verás los comentarios de esta forma:

Comentario del revisor Si todo está perfecto.

Comentario del revisor Si tu código está bien pero se puede mejorar o hay algún detalle que le hace falta.

Comentario del revisor Si de pronto hace falta algo o existe algún problema con tu código o conclusiones.

Puedes responderme de esta forma:

Respuesta del estudiante

¡Empecemos!

Y.Music

Contents

- Introducción
- Etapa 1. Descripción de los datos
  - Conclusions
- Etapa 2. Data preprocessing
  - 2.1 Estilo del encabezado
  - 2.2 Valores ausentes
  - 2.3 Duplicados

- 2.4 Conclusiones
- Etapa 3. Prueba de hipótesis
  - 3.1 Hipótesis 1: comparar el comportamiento del usuario en las dos ciudades
  - 3.2 Hipótesis 2: música al principio y al final de la semana
  - 3.3 Hipótesis 3: preferencias de género en Springfield y Shelbyville
- Conclusiones

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, compararás las preferencias musicales de las ciudades de Springfield y Shelbyville. Estudiarás datos reales de Y.Music para probar las hipótesis de abajo y comparar el comportamiento del usuario de esas dos ciudades.

Objetivo:

Prueba 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 necesitarás examinarlos antes de probar las 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

Volver a Contenidos

Comentario del revisor Bien. Ten presente en el futuro siempre incluir esta sección: un título, una introducción y una tabla de contenidos. El objetivo es que puedas incluir este proyecto en tu

portafolio y que las personas no familiarizadas con la programación puedan navegar más fácilmente.

Etapa 1. Descripción de los datos

Abre los datos en Y.Music y examínalos.

Necesitarás `pandas` así que impórtalo.

In [1]:
```
# importando pandas
import pandas as pd
import copy
```

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

In [2]:
```
# leyendo el archivo y almacenándolo en df
df = pd.read_csv('/datasets/music_project_en.csv')
```

Imprime las 10 primeras filas de la tabla:

In [3]:
```
# obteniendo las 10 primeras filas de la tabla df
df.head(10)
```

Out[3]:

	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
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
9	E772D5C0	Pessimist	NaN	dance	Shelbyville	21:20:49	Wednesday

Respuesta del estudiante Esta vez omiti el $pr\int$ como me recomendaste, para dejar que se ejecute automaticamente.

Comentario del revisor No es necesario que utilices la función $pr\int$ en este caso ya que por defecto $jupyter$ imprime la última línea de cada celda.

Obtener la información general sobre la tabla con un comando:

In [4]:
```
# obteniendo información general sobre los datos en df
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
```

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

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

Podemos ver tres problemas con el estilo en los nombres de las columnas:

1. Algunos nombres están en mayúsculas, otros en minúsculas.
2. Hay algunos espacios en algunos nombres.
3. `En la columna 'artist' falta el dato de la cancion 'Pessimist'.`

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.

Comentario del revisor ¡Buen trabajo!

Volver a Contenidos

Etapa 2. Preprocesamiento de datos

Corrige el formato en los encabezados de las columnas y ocúpate de los valores ausentes. Después, comprueba si hay duplicados en los datos.

Estilo del encabezado

Imprime el encabezado de la columna:

In [5]:
```
# la lista de los nombres de las columnas en la tabla df
column_headers = df.columns.tolist()
print(column_headers)
```

```
['  userID', 'Track', 'artist', 'genre', '  City  ', 'time', 'Day']
```

Cambia los nombres de las columnas de acuerdo con las reglas del buen estilo:

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

In [6]:
```
# renombra las columnas
new_column_headers = [column.strip().lower().replace(' ', '_') for column in column_hea

columns_map = {column_headers[i]: new_column_headers[i] for i in range(len(column_heade

df = df.rename(columns=columns_map)
```

Comprueba el resultado. Imprime los nombres de las columnas una vez más:

In [7]:
```
# comprobando el resultado: la lista de los nombres de las columnas
column_headers = df.columns.tolist()
print(column_headers)
```

```
['userid', 'track', 'artist', 'genre', 'city', 'time', 'day']
```

Comentario del revisor Buen trabajo.

Volver a Contenidos

Valores ausentes

Primero encuentra el número de valores ausentes en la tabla. Para ello, utiliza dos métodos pandas:

In [8]:
```
# calculando valores ausentes
info_summary = df.info()
null_counts = df.shape[0] - df.describe().loc['count']
print(null_counts)
```

```
null_counts = df.isnull().sum()
print(null_counts)
```

```
<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
userid       0
track     1343
artist    7567
genre     1198
city         0
time         0
day          0
Name: count, dtype: object
userid       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 puedes reemplazarlos por marcadores claros.

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

Reemplazar los valores ausentes en `'track'`, `'artist'`, y `'genre'` con la string `'unknown'`. Para ello, crea la lista `columns_to_replace`, recórrela con un bucle `for` y reemplaza los valores ausentes en cada una de las columnas:

In [9]:

```
# recorriendo los nombres de las columnas y reemplazando los valores ausentes con 'unkn
columns_to_replace = ['track', 'artist', 'genre']

for column in columns_to_replace:
    df[column].fillna('unknown', inplace=True)

print(df[columns_to_replace])
```

```
                                  track            artist       genre
0                      Kamigata To Boots  The Mass Missile        rock
```

```
1            Delayed Because of Accident  Andreas Rönnberg        rock
2                      Funiculì funiculà       Mario Lanza         pop
3                  Dragons in the Sunset        Fire + Ice        folk
4                            Soul People        Space Echo       dance
...                                  ...               ...         ...
65074                            My Name            McLean         rnb
65075  Maybe One Day (feat. Black Spade)       Blu & Exile         hip
65076                          Jalopiina           unknown  industrial
65077                       Freight Train     Chas McDevitt        rock
65078          Tell Me Sweet Little Lies      Monica Lopez     country

[65079 rows x 3 columns]
```

Asegúrate de que la tabla no contiene más valores ausentes. Cuenta de nuevo los valores ausentes.

In [10]:

```
# contando valores ausentes
null_counts = df.isnull().sum()
print(null_counts)
```

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

Comentario del revisor ¡Gran trabajo en esta sección!

Volver a Contenidos

Duplicados

Encuentra el número de duplicados obvios en la tabla utilizando un comando:

In [11]:

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

```
3826
```

Llama al método `pandas` para deshacerte de los duplicados obvios:

In [12]:

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

Respuesta del estudiante Gracias por el aclarando, hice la correccion de lugar, utilizando la recomendacion que has proporcionado, para evitar los saltos de indice.

Comentario del revisor ~Cuando eliminamos los duplicados también eliminamos los índices asociados a cada fila. Esto puede generar problemas de búsqueda por filas porque tendremos saltos en el índice. Lo mejor será, además de lo ya realizado, agregar el método $reset_{\in}dex$ y pasarle el argumento $drop = True$: $df.drop_duplicates().reset_{\in}dex(drop = True)$.~

Cuenta los duplicados obvios una vez más para asegurarte de que todos han sido eliminados:

In [13]:
```
# comprobando duplicados
print(df.duplicated().sum())
```

```
0
```

Ahora deshazte 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 a resultado.

Imprime 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

In [14]:
```
# inspeccionando los nombres de géneros únicos
unique_genres = df['genre'].unique()

print(unique_genres)
```

```
['rock' 'pop' 'folk' 'dance' 'rusrap' 'ruspop' 'world' 'electronic'
 'unknown' 'alternative' 'children' 'rnb' 'hip' 'jazz' 'postrock' 'latin'
 'classical' 'metal' 'reggae' 'triphop' 'blues' 'instrumental' 'rusrock'
 'dnb' 'türk' 'post' 'country' 'psychedelic' 'conjazz' 'indie'
 'posthardcore' 'local' 'avantgarde' 'punk' 'videogame' 'techno' 'house'
 'christmas' 'melodic' 'caucasian' 'reggaeton' 'soundtrack' 'singer' 'ska'
 'salsa' 'ambient' 'film' 'western' 'rap' 'beats' "hard'n'heavy"
 'progmetal' 'minimal' 'tropical' 'contemporary' 'new' 'soul' 'holiday'
 'german' 'jpop' 'spiritual' 'urban' 'gospel' 'nujazz' 'folkmetal'
 'trance' 'miscellaneous' 'anime' 'hardcore' 'progressive' 'korean'
 'numetal' 'vocal' 'estrada' 'tango' 'loungeelectronic' 'classicmetal'
 'dubstep' 'club' 'deep' 'southern' 'black' 'folkrock' 'fitness' 'french'
 'disco' 'religious' 'hiphop' 'drum' 'extrememetal' 'türkçe'
 'experimental' 'easy' 'metalcore' 'modern' 'argentinetango' 'old' 'swing'
 'breaks' 'eurofolk' 'stonerrock' 'industrial' 'funk' 'middle' 'variété'
 'other' 'adult' 'christian' 'thrash' 'gothic' 'international' 'muslim'
 'relax' 'schlager' 'caribbean' 'nu' 'breakbeat' 'comedy' 'chill' 'newage'
 'specialty' 'uzbek' 'k-pop' 'balkan' 'chinese' 'meditative' 'dub' 'power'
 'death' 'grime' 'arabesk' 'romance' 'flamenco' 'leftfield' 'european'
 'tech' 'newwave' 'dancehall' 'mpb' 'piano' 'top' 'bigroom' 'opera'
 'celtic' 'tradjazz' 'acoustic' 'epicmetal' 'hip-hop' 'historisch'
 'downbeat' 'downtempo' 'africa' 'audiobook' 'jewish' 'sängerportrait'
 'deutschrock' 'eastern' 'action' 'future' 'electropop' 'folklore'
 'bollywood' 'marschmusik' 'rnr' 'karaoke' 'indian' 'rancheras'
 'afrikaans' 'rhythm' 'sound' 'deutschspr' 'trip' 'lovers' 'choral'
 'dancepop' 'retro' 'smooth' 'mexican' 'brazilian' 'ïîï' 'mood' 'surf'
 'gangsta' 'inspirational' 'idm' 'ethnic' 'bluegrass' 'broadway'
 'animated' 'americana' 'karadeniz' 'rockabilly' 'colombian' 'self' 'hop'
 'sertanejo' 'japanese' 'canzone' 'lounge' 'sport' 'ragga' 'traditional'
 'gitarre' 'frankreich' 'emo' 'laiko' 'cantopop' 'glitch' 'documentary'
 'oceania' 'popeurodance' 'dark' 'vi' 'grunge' 'hardstyle' 'samba'
 'garage' 'art' 'folktronica' 'entehno' 'mediterranean' 'chamber' 'cuban'
 'taraftar' 'gypsy' 'hardtechno' 'shoegazing' 'bossa' 'latino' 'worldbeat'
 'malaysian' 'baile' 'ghazal' 'arabic' 'popelectronic' 'acid' 'kayokyoku'
 'neoklassik' 'tribal' 'tanzorchester' 'native' 'independent' 'cantautori'
 'handsup' 'punjabi' 'synthpop' 'rave' 'französisch' 'quebecois' 'speech'
 'soulful' 'jam' 'ram' 'horror' 'orchestral' 'neue' 'roots' 'slow'
```

```
 'jungle' 'indipop' 'axé' 'fado' 'showtunes' 'arena' 'irish' 'mandopop'
 'forró' 'dirty' 'regional']
```

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, declara 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` .

In [15]:

```
# función para reemplazar duplicados implícitos
def replace_wrong_genres(wrong_genres, correct_genre):
    for wrong_genre in wrong_genres:
        df['genre'] = df['genre'].replace(wrong_genre, correct_genre)

wrong_genres = ['hip', 'hop', 'hip-hop']
correct_genre = 'hiphop'

replace_wrong_genres(wrong_genres, correct_genre)

print(df['genre'])
```

```
0              rock
1              rock
2               pop
3              folk
4             dance
            ...
61248           rnb
61249        hiphop
61250    industrial
61251          rock
61252       country
Name: genre, Length: 61253, dtype: object
```

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

In [16]:

```
# eliminando duplicados implícitos
df['genre'].replace(['hip', 'hop', 'hip-hop'], 'hiphop', inplace=True)
```

Asegúrate que los nombres duplicados han sido eliminados. Imprime la lista de valores únicos de la columna `'genre'` :

In [17]:
```
# revisando en busca de duplicados implícitos
duplicate_count = df.duplicated().sum()

print("Número de duplicados: ", duplicate_count)
```

```
Número de duplicados:  0
```

Comentario del revisor ¡Muy buen trabajo en esta sección!

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

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. Comprueba esto utilizando los datos de tres días de la semana: lunes, miércoles y viernes.

- Divide a los usuarios en grupos por ciudad.
- Compara cuántas pistas reprodujo cada grupo el lunes, el miércoles y el viernes.

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

Evalúa la actividad del usuario en cada ciudad. Agrupa los datos por ciudad y encuentra el número de canciones reproducidas en cada grupo.

In [18]:
```
# contando las pistas reproducidas en cada ciudad
tracks_played_per_city = df.groupby('city')['track'].count()
```

Springfield ha reproducido más pistas que 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 agrupa los datos por día de la semana y encuentra el número de pistas reproducidas el lunes, miércoles y viernes.

In [19]:
```
# calculando las pistas reproducidas en cada uno de los tres días
filtered_df = df[df['day'].isin(['Monday', 'Wednesday', 'Friday'])]
tracks_played_per_day = filtered_df.groupby('day')['track'].count()
print(tracks_played_per_day)
```

```
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 has visto cómo funciona el agrupar por ciudad o día. Ahora escribe la función que agrupará ambos.

Crea 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, utiliza 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

Aplica un filtrado consecutivo con indexación lógica.

Después, calcula los valores de la columna `'user_id'` en la tabla resultante. Almacena el resultado en la nueva variable. Recupera esta variable de la función.

In [20]:
```
# <creando la función number_tracks()>
# declararemos la función con dos parámetros: day=, city=.
# deja que la variable track_list almacene las filas df en las que
# el valor en la columna 'day' es igual al parámetro day= y, al mismo tiempo,
# el valor de la columna 'city' es igual al parámetro city= (aplica el filtrado consecu
# con indexación lógica).
# deja que la variable track_list_count almacene el número de valores de la columna 'us
# (encontrado con el método count()).
# permite que la función devuelva un número: el valor de track_list_count.
```

```
# la función cuenta las pistas reproducidas en un cierto día y ciudad.
# 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 tabla filtrada,
# y devuelve ese número.
# para ver lo que devuelve, envuelve la llamada de la función en print().
def number_tracks(day, city):
    track_list = df[(df['day'] == day) & (df['city'] == city)]
    track_list_count = track_list['userid'].count()
    return track_list_count
```

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 [21]:

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

```
15740
```

In [22]:

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

```
5614
```

In [23]:

```
# el número de canciones reproducidas en Springfield el miércoles
print(number_tracks(day='Wednesday', city='Springfield'))
```

```
11056
```

In [24]:

```
# el número de canciones reproducidas en Shelbyville el miércoles
print(number_tracks(day='Wednesday', city='Shelbyville'))
```

```
7003
```

In [25]:

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

```
15945
```

In [26]:

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

```
5895
```

Utiliza `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()`

In [27]:

```
springfield_monday = number_tracks(day='Monday', city='Springfield')
springfield_wednesday = number_tracks(day='Wednesday', city='Springfield')
springfield_friday = number_tracks(day='Friday', city='Springfield')

shelbyville_monday = number_tracks(day='Monday', city='Shelbyville')
```

```
shelbyville_wednesday = number_tracks(day='Wednesday', city='Shelbyville')
shelbyville_friday = number_tracks(day='Friday', city='Shelbyville')
```

In [28]:

```
# tabla con los resultados
data = {
    'city': ['Springfield', 'Shelbyville'],
    'Monday': [springfield_monday, shelbyville_monday],
    'Wednesday': [springfield_wednesday, shelbyville_wednesday],
    'Friday': [springfield_friday, shelbyville_friday]
}

table = pd.DataFrame(data)

print(table)
```

```
          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 Shelbyville, al contario, los usuarios escuchan más música los miércoles. La actividad de los usuarios los lunes y viernes es menor.

Así que la primera hipótesis parece ser correcta.

Comentario del revisor ¡Buen trabajo!

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.

Obtén tablas (asegúrate de que el nombre de tu tabla combinada encaja con el DataFrame dado en los dos bloques de código de abajo):

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

In [29]:

```
# 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']
```

In [30]:

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

Escribe 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.

In [31]:

```
# declarando la función genre_weekday() con los parámetros day=, time1= y time2=. Deber
# devolver información sobre los géneros más populares de un determinado día a una dete

# 1) Deja que la variable genre_df almacene las filas que cumplen varias condiciones:
#    - el valor de la columna 'day' es igual al valor del argumento day=
#    - el valor de la columna 'time' es mayor que el valor del argumento time1=
#    - el valor en la columna 'time' es menor que el valor del argumento time2=
#    Utiliza un filtrado consecutivo con indexación lógica.

# 2) Agrupa genre_df por la columna 'genre', toma una de sus columnas,
#    y utiliza el método count() para encontrar el número de entradas por cada uno de
#    los géneros representados; almacena los Series resultantes en
#    la variable genre_df_count

# 3) Ordena genre_df_count en orden descendente de frecuencia y guarda el resultado
#    en la variable genre_df_sorted

# 4) Devuelve un objeto Series con los primeros 15 valores de genre_df_sorted - los 15
#    géneros más populares (en un determinado día, en un determinado periodo de tiempo)

# Escribe tu función aquí
#from datetime import datetime, time
def genre_weekday(df, day, time1, time2):

    # filtrado consecutivo
    # genre_df solo almacenará aquellas filas df en las que el día sea igual a day=
    genre_df = copy.deepcopy(df.loc[df['day'] == day])

    # genre_df solo almacenará aquellas filas df en las que el tiempo sea menos que tim
    # genre_df solo almacenará aquellas filas df en las que el tiempo sea mayor que tim
    genre_df = genre_df.loc[(genre_df['time'] < time2) & (genre_df['time'] > time1)]

    # agrupa el DataFrame filtrado por la columna con los nombres de los géneros, toma
    genre_df_grouped = genre_df.groupby('genre')['genre'].count()

    # ordenaremos el resultado en orden descendente (por lo que los géneros más popular
    genre_df_sorted = genre_df_grouped.sort_values(ascending=False)

    # devolveremos el objeto Series que almacena los 15 géneros más populares en un día
    return genre_df_sorted[:15]
```

Respuesta del estudiante Aca he modificado la parte comparativa de las variables 'time' para que sean mas simples, eso si, por unos Warnings que estuve teniendo, tuve que importar una

copia de la tabla, para que todo funcione correctamente, aparte modifique el 'groupby' para que solo acceda a 'genre' como recomendaste.

Respuesta del estudiante Como puedes ver, modifique la parte en la que no se estaba usando el filtro correctamente, por la que has recomendado, para que se filtren los valores de manera adecuada, aparte en la parte inferior modifique las variables para que representen correctamente los valores presentados dia de la semana, parte del dia y de que ciudad.

Compara los resultados de la función `genre_weekday()` para Springfield y Shelbyville el lunes por la mañana (de 7 a 11) y el viernes por la tarde (de 17:00 a 23:00):

In [32]:
```
# llamando a la función para el lunes por la mañana en Springfield (utilizando spr_gene
result_monday_morning_spr = genre_weekday(spr_general, 'Monday', '07:00:00', '11:00:00'
```

In [33]:
```
# llamando a la función para el lunes por la mañana en Shelbyville (utilizando shel_gen
result_monday_morning_shel = genre_weekday(shel_general, 'Monday', '07:00:00', '11:00:0
```

In [34]:
```
# llamando a la función para el viernes por la tarde en Springfield
result_friday_afternoon_spr = genre_weekday(spr_general, 'Friday', '17:00:00', '23:00:0
```

In [35]:
```
# llamando a la función para el viernes por la tarde en Shelbyville
result_friday_afternoon_shel = genre_weekday(shel_general, 'Friday', '17:00:00', '23:00
```

Respuesta del estudiante Aca he modificado el codigo para que use los valores STR como recomendaste y todo sea mas user friendly.

Respuesta del estudiante Mas arriba se puede observar la modificacion en la que cambio la libreria de sitio para importarla al principo del documento como me has sugerido.

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 similar. Los cinco géneros más populares son los mismos, solo rock y electrónica han intercambiado posiciones.

2. En Springfield el número de valores ausentes resultaron ser tan altos que el valor `'unknown'` llegó al décimo. Esto significa que los valores ausentes forman una parte considerable de los datos, lo que podría ser la base de la cuestión sobre la fiabilidad de nuestras conclusiones.

Para el viernes por la tarde, la situación es similar. Los géneros individuales varían algo pero, en general, los 15 más populares son parecidos en las dos ciudades.

De esta forma, la segunda hipótesis ha sido parcialmente demostrada:

- Los usuarios escuchan música similar al principio y al final de la semana.
- No hay una gran diferencia entre Springfield y Shelbyville. En ambas ciudades, el pop es el género más popular.

Sin embargo, el número de valores ausentes hace este resultado un tanto cuestionable. En Springfield, hay tantos que afectan a nuestros 15 más populares. De no faltarnos esos valores, las cosas podrían parecer diferentes.

Comentario del revisor Excelente trabajo, Carlos.

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.

Agrupa la tabla `spr_general` por género y encuentra el número de canciones reproducidas de cada género con el método `count()` . Después, ordena el resultado en orden descendente y guárdalo en `spr_genres` .

In [36]:

```
# en una línea: agrupa la tabla spr_general por la columna 'genre',
# cuenta los valores 'genre' con count() en la agrupación,
# ordena el Series resultante en orden descendiente, y almacénalo en spr_genres
#spr_genres = spr_general.groupby('genre').count()['track'].sort_values(ascending=False
spr_genres = spr_general.groupby('genre')['genre'].count().sort_values(ascending=False)
```

Respuesta del estudiante Aca he modificado el codigo para que acceda a la columna 'genre' en lugar de 'track'.

Imprime las 10 primeras filas de `spr_genres` :

In [37]:

```
# imprimiendo las 10 primeras filas de spr_genres
print(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: genre, dtype: int64
```

Ahora haz lo mismo con los datos de Shelbyville.

Agrupa la tabla `shel_general` por género y encuentra el número de canciones reproducidas de cada género. Después, ordena el resultado en orden descendente y guárdalo en la tabla

`shel_genres` :

In [38]:
```
# en una línea: agrupa la tabla shel_general por la columna 'genre',
# cuenta los valores 'genre' en el agrupamiento con count(),
# ordena el Series resultante en orden descendente y guárdalo en shel_genres
shel_genres = shel_general.groupby('genre')['genre'].count().sort_values(ascending=Fals
```

Respuesta del estudiante Aca al igual que la anterior vez he modificado el codigo para que acceda a la columna 'genre' en lugar de 'track'.

Imprime las 10 primeras filas de `shel_genres` :

In [39]:
```
# imprimiendo las 10 primeras filas de shel_genres
print(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: genre, 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 ciudades, Springfield y Shelbyville, se prefiere el pop.

Tras analizar los datos, concluimos:

1. La actividad del usuario en Springfield y Shelbyville depende del día de la semana aunque las ciudades varían de diferentes formas.

La primera hipótesis ha sido aceptada completamente.

1. Las preferencias musicales no varían significativamente en el transcurso de la semana en Springfield y Shelbyville. Podemos observar pequeñas diferencias en el orden los lunes, pero:

- En Springfield y Shelbyville la gente lo que más escucha es la música pop.

Así que no podemos aceptar esta hipótesis. También debemos tener en cuenta que el resultado podría haber sido diferente si no fuera por los valores ausentes.

1. Resulta que las preferencias musicales de los usuarios de Springfield y Shelbyville son bastante parecidas.

La tercera hipótesis es rechazada. Si hay alguna diferencia en las preferencias no se puede observar en los datos.

Nota

En los proyectos reales, la investigación supone el estudio de hipótesis estadísticas que es más preciso y cuantitativo. También ten en cuenta que no siempre podemos sacar conclusiones sobre una ciudad entera basándonos en datos de una sola fuente.

Analizarás el estudio de hipótesis en el sprint de análisis estadístico de datos.

Comentarios generales **Comentario del revisor** ¡Gran trabajo! Has demostrado un muy buen manejo del lenguaje y uso de la librería pandas. Nos quedan algunos detalles de forma y de fondo pendientes por resolver para poder aprobar el proyecto. He dejado comentarios a lo largo del documento para ayudarte con ello.

Comentarios generales #2 **Comentario del revisor** Buen trabajo, Carlos. Nos queda trabajar un poco más en la sección de la función $\geq nre_weekday$ y estaremos listos para aprobar tu proyecto.

Comentarios generales #3 **Comentario del revisor** Gran trabajo, Carlos. Todo ha sido corregido y has aprobado tu primer proyecto. ¡Felicitaciones!

Volver a Contenidos