P2_Python_basico_2

July 24, 2024

1 Déjame escuchar la música

2 Contenido

2.1 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.

2.1.1 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.

2.1.2 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.

Volver a Contenidos

2.2 Etapa 1. Descripción de los datos

Abre los datos y examínalos.

Necesitarás `pandas`, así que impórtalo.

[1]:
```
# Importar pandas
import pandas as pd
```

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

[2]:
```
# Leer el archivo y almacenarlo en df
df = pd.read_csv('/datasets/music_project_en.csv')
df.to_csv('music_project_en.csv', index=False)
```

Muestra las 10 primeras filas de la tabla:

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

[3]:
```
     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.

[4]:
```
# Obtener la información general sobre nuestros datos
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
```

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. `No se utiliza el estilo de escritura snake_case en userID, debería ser user_id`.

2.2.1 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?`

R: dentro del DataFrame tenemos datos como identificadores únicos de usuario, nombre de la canción, nombre del artista, género de la canción, ciudad donde fue escuchada, la hora y el día en la que fue escuchada

para entender los datos que almacenan las columnas debemos realizar preprocesamiento de datos mediante normalización de columnas (ej. mayúsculas, snake_case, etc), tratamiento de datos NaN, agrupaciones, sumas, cuentas y ordenamientos de datos

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

R: Si hay suficientes datos para responder nuestra hipótesis, principalmente necesitaremos datos como la ciudad, los días en los que se escuchó la canción en cuestión, así como el Track

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

R: si,al menos al imprimir las primeras 10 filas se observan valores ausentes

Volver a Contenidos

2.3 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.

2.3.1 Estilo del encabezado

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

[5]:
```
# Muestra los nombres de las columnas
df.columns
```

[5]:
```
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:

[6]:
```
# Bucle en los encabezados poniendo todo en minúsculas
new_col_names=[]
for col in df.columns:
    lowered_column= col.lower()
    new_col_names.append(lowered_column)

df.columns=new_col_names
print (new_col_names)
```

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

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:

[7]:
```
# Bucle en los encabezados eliminando los espacios
new_col_names=[]
for col in df.columns:
    stripped_column= col.strip()
    new_col_names.append(stripped_column)

df.columns=new_col_names
print (df.columns)
```

```
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.

[8]:
```
# Cambiar el nombre de la columna "userid"
df.rename(columns={'userid':'user_id'},inplace=True)
```

Comprueba el resultado. Muestra los encabezados una vez más:

[9]:
```
# Comprobar el resultado: la lista de encabezados
print(df.columns)
```

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

Volver a Contenidos

2.3.2 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.

[10]:
```
# Calcular el número de valores ausentes
df.isna().sum()
```

[10]:
```
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 `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.

[11]:
```
# Bucle en los encabezados reemplazando los valores ausentes con 'unknown'
#Generamos una lista con los nombres de las columnas a modificar (reemplazar␣
 ↪vacíos)
#replace_col = ['track','artist','genre']
replace_col = ['track','artist','genre']

#generamos un bucle que recorra cada columna de la lista anterior y reemplace␣
 ↪los NaN por "unknown"
for col in replace_col:
    df[col].fillna('unknown',inplace=True)

#imprimimos la cabecera de nuestro df para verificar el resultado
print(df.head(10))
```

```
    user_id                        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           unknown   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
```

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.

[12]:
```
# Contar valores ausentes
df.isna().sum()
```

[12]:
```
user_id    0
track      0
artist     0
genre      0
city       0
time       0
day        0
dtype: int64
```

Volver a Contenidos

2.3.3 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.

[13]:
```
# Contar duplicados explícitos
print(df.duplicated().sum())
```

```
3826
```

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

[15]:
```
# Eliminar duplicados explícitos
#eliminamos los duplicados mediante la función "drop_duplicates()"
df = df.drop_duplicates().reset_index(drop=True)
```

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

[16]:
```
# Comprobar de nuevo si hay duplicados
print(df.duplicated().sum())
```

```
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.

[17]:
```
# Inspeccionar los nombres de géneros únicos
print(df['genre'].sort_values().unique())
print()
```

```
print('la cantidad de géneros únicos es: ',df['genre'].nunique())
```

```
['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']

la cantidad de géneros únicos es:  269
```

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.

[18]:
```
# Función para reemplazar duplicados implícitos
#generamos una lista con los valores dubplicados implicitos a reemplazar
wrong_genres=['hip','hop','hip-hop']
#generamos un string con el valor correcto
correct_genre='hiphop'
#definimos la columna sobre la cual vamos a trabajar
col=['genre']

#definimos la función con que recibe como parámetros la lista de valores␣
 ↪erróneos y el string correcto
def replace_wrong_genres(wrong_genres,correct_genre):
    #iteramos la lista de valores incorrectos
    for wrong_value in wrong_genres:
        #por cada valor incorrecto lo reemplazamos con el string correcto
        df[col]=df[col].replace(wrong_value, correct_genre)
    return df
```

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`:

[19]:
```
# Eliminar duplicados implícitos
print(replace_wrong_genres(wrong_genres, correct_genre))
```

```
        user_id                                track            artist  \
0      FFB692EC                  Kamigata To Boots  The Mass Missile
1      55204538        Delayed Because of Accident  Andreas Rönnberg
2        20EC38                  Funiculì funiculà       Mario Lanza
3      A3DD03C9              Dragons in the Sunset        Fire + Ice
4      E2DC1FAE                         Soul People        Space Echo
…         …                                    …                …
61248  729CBB09                           My Name            McLean
61249  D08D4A55  Maybe One Day (feat. Black Spade)       Blu & Exile
61250  C5E3A0D5                          Jalopiina           unknown
61251  321D0506                       Freight Train     Chas McDevitt
61252  3A64EF84          Tell Me Sweet Little Lies      Monica Lopez

            genre         city      time        day
0            rock  Shelbyville  20:28:33  Wednesday
1            rock  Springfield  14:07:09     Friday
```

```
2             pop  Shelbyville  20:58:07  Wednesday
3            folk  Shelbyville  08:37:09     Monday
4           dance  Springfield  08:34:34     Monday
…             …           …         …         …
61248         rnb  Springfield  13:32:28  Wednesday
61249      hiphop  Shelbyville  10:00:00     Monday
61250  industrial  Springfield  20:09:26     Friday
61251        rock  Springfield  21:43:59     Friday
61252     country  Springfield  21:59:46     Friday

[61253 rows x 7 columns]
```

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

[20]:
```
# Comprobación de duplicados implícitos
print(df['genre'].sort_values().unique())
print()
print('la cantidad de géneros únicos es: ',df['genre'].nunique())
```

```
['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']

la cantidad de géneros únicos es:  266
```

Volver a Contenidos

2.3.4 Tus observaciones

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

- Podemos observar que originalmente teníamos 3826 filas con valores duplicados explícitos, estos valores nos pueden generar sesgos en nuestro análisis final, por lo tanto fueron eliminados.
- se tenían 1198 campos de la columna 'genre' nulos, así cómo 7567 campos en la columna 'artist' nulos, estos campos, principalmente el del género, son escenciales para detectar tendencias de comportamiento en nuestros usuarios.
- dentro de los duplicados implícitos detectamos anomalías en el género "hiphop" que fueron regularizadas y nos retornaron 266 géneros únicos, cuando previamente teníamos 269.

Volver a Contenidos

2.4 Etapa 3. Prueba de hipótesis

2.4.1 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.

Realiza cada cálculo por separado.

El primer paso es evaluar la actividad del usuario en cada ciudad. Recuerda las etapas dividir-aplicar-combinar de las que hablamos anteriormente en la lección. Tu objetivo ahora es agrupar los datos por ciudad, aplicar el método apropiado para contar durante la etapa de aplicación y luego encontrar la cantidad de canciones reproducidas en cada grupo especificando la columna para obtener el recuento.

A continuación se muestra un ejemplo de cómo debería verse el resultado final: `df.groupby(by='....')['column'].method()`Realiza cada cálculo por separado.

Para evaluar la actividad de los usuarios y las usuarias en cada ciudad, agrupa los datos por ciudad y encuentra la cantidad de canciones reproducidas en cada grupo.

[21]:
```
# Contar las canciones reproducidas en cada ciudad
print(df.groupby('city')['track'].count())
```

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

`Comenta tus observaciones aquí` * observamos que Springfield escucha más canciones en general que Shelbyville, exactamente una diferencia de 24,229 reproducciones más

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.

[22]:
```
# Calcular las canciones reproducidas en cada uno de los tres días
print(df.groupby('day')['track'].count())
```

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

`Comenta tus observaciones aquí` * el día Viernes es el día con mayor concurrencia de reproducciones en la agrupación de 3 días observados, hace sentido que sea el día con mayor cantidad de reproducciones debido a que es el último día de la semana laboral y se podría percibir como un día más social.

- por otro lado, el Miércoles es el día con menor cantidad de reproducciones, posiblemente sea porque es mitad de semana y las personas estén menos animadas a socializar o estén más enfocados en sus ocupaciones.

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.

[23]:
```
# Declara la función number_tracks() con dos parámetros: day= y city=.
def number_tracks(day, city):

    # Almacena las filas del DataFrame donde el valor en la columna 'day' es␣
 ↪igual al parámetro day=
    #day_filter = df[df['day']==day]

    # Filtra las filas donde el valor en la columna 'city' es igual al␣
 ↪parámetro city=
    #city_filter = day_filter[day_filter['city']==city]


    city_filter = df[ (df['city']==city) & (df['day']==day) ]

    # Extrae la columna 'user_id' de la tabla filtrada y aplica el método␣
 ↪count()

    return city_filter['user_id'].count()
    # Devolve el número de valores de la columna 'user_id'

    #### COMENTARIOS DEL ESTUDIANTE####
    #utilicé una doble indexación lógica para reducir mi cantidad de código,␣
 ↪sin embargo dejé documentado cómo lo habría ejecutado siguiendo los pasos␣
 ↪explícitos que solicitan
```

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.

[24]:
```
# El número de canciones reproducidas en Springfield el lunes
print("El número de canciones reproducidas en Springfield el Lunes fue:␣
 ↪",number_tracks('Monday','Springfield'))
```

```
El número de canciones reproducidas en Springfield el Lunes fue:  15740
```

[25]:
```
# El número de canciones reproducidas en Shelbyville el lunes
print("El número de canciones reproducidas en Shelbyville el Lunes fue:␣
 ↪",number_tracks('Monday','Shelbyville'))
```

```
El número de canciones reproducidas en Shelbyville el Lunes fue:  5614
```

[26]:
```
# El número de canciones reproducidas en Springfield el miércoles
print("El número de canciones reproducidas en Springfield el Miércoles fue:␣
 ↪",number_tracks('Wednesday','Springfield'))
```

```
El número de canciones reproducidas en Springfield el Miércoles fue:  11056
```

[27]:
```
# El número de canciones reproducidas en Shelbyville el miércoles
print("El número de canciones reproducidas en Shelbyville el Miércoles fue:␣
 ↪",number_tracks('Wednesday','Shelbyville'))
```

```
El número de canciones reproducidas en Shelbyville el Miércoles fue:  7003
```

[28]:
```
# El número de canciones reproducidas en Springfield el viernes
print("El número de canciones reproducidas en Springfield el Viernes fue:␣
 ↪",number_tracks('Friday','Springfield'))
```

```
El número de canciones reproducidas en Springfield el Viernes fue:  15945
```

[29]:
```
# El número de canciones reproducidas en Shelbyville el viernes
print("El número de canciones reproducidas en Shelbyville el Viernes fue:␣
 ↪",number_tracks('Friday','Shelbyville'))
```

```
El número de canciones reproducidas en Shelbyville el Viernes fue:  5895
```

Conclusiones

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

La hipótesis es correcta.

- Observamos que en general Springfield tiene un comportamiento más activo que Shelbyville
- En lo particular observamos que en Shelbyville el día más activo es el Miércoles, superando al 2do día más activo (Viernes) por un 18.796% , el día menos activo de esta ciudad fue el Lunes, cayendo respecto al Miércoles (día más activo) por 24.74%
- El día más activo de Springfield es el Viernes, superando a su 2do día más activo (Lunes) por apenas 1.302% , en contraparte el día menos activo de esta ciudad es el Miércoles, cayendo respecto al Viernes (día más activo) por 44.22%

Con estos datos podemos determinar que si existe una tendencia de comportamiento diferente para cada ciudad y para cada día de la semana

Volver a Contenidos

3 Conclusiones

Resume aquí tus conclusiones sobre la hipótesis.

Observamos que Springfield en general tiene un comportamiento más activo que Shelbyville, el día más activo para Springfield es el Viernes, por otro lado el día más activo de Shelbyville es el Miércoles.

Con la comprobación de esta hipótesis podríamos llegar a nuevas investigaciones, como determinar cual es el mejor día para lanzar una campaña de marketing de un artista musical, conforme a los días con mayor cantidad de reproducciones, así como el género más escuchado por día, con estos datos podríamos asegurar un mejor engagement por parte de los usuarios con la campaña planeada.

Para el alcance de nuestro dataset en este proyecto nuestra hipótesis es correcta, sin embargo, si quisieramos ampliar más el estudio, podríamos verificar la cantidad de pobladores de cada ciudad y compararlo en medida de porcentajes tomando en cuenta la cantidad de reproducciones, por ejemplo, encontrar que si bien Shelbyville tiene menor cantidad de reproducciones en un día dado, podríamos determinar que tiene un mayor porcentaje de usuarios activos en comparación a Springfield, esto en dado caso de que la población de Shelbyville sea menor a la de Springfield.

Volver a Contenidos