1. Procesamiento de datos

September 3, 2024

1 Datos en formato cvs

Los datos en formato cvs (comma separated values) son un formato de archivo que se utiliza para almacenar datos tabulares. Cada línea del archivo corresponde a una fila de la tabla, y los valores de cada fila están separados por comas.

La biblioteca `pandas` de Python permite leer y escribir archivos en formato cvs.

Instalación de la biblioteca `pandas`:

```
conda install pandas
```

1.1 Lectura de datos

Para leer un archivo en formato cvs se utiliza la función `read_csv` de la biblioteca `pandas`. Por ejemplo, para leer un archivo llamado `datos.csv` se utiliza la siguiente instrucción:

```
pd.read_csv('datos.csv')
```

1.1.1 Ejemplos de los parámetros de la función read_csv

```
read.csv(filepath="../datasets/customer-churn-model/Customer Churn Model.csv/titanic3.csv",
        sep = ",",
        dtype={"ingresos":np.float64, "edad":np.int32},
        header=0,names={"ingresos", "edad"},
        skiprows=12,
        index_col=None,
        skip_blank_lines=False,
        na_filter=False
        )
```

1. **sep** separado por comas
2. **dtype** columna ingresos de tipo flotante. El valor determinado es None, quiere decir que panda asignara el tipo que mas le convenga
3. **header** donde esta la cabecera, y cuales quieres utilizar
4. **skiperows** saltar filas
5. **index_col** cambiar la columna del index
6. **skipe_blank** saltar lineas en blancos
7. **na_filter** elimina NaN

[ ]:
```
# Importar bibliotecas
import pandas as pd # Biblioteca para manejo de datos
import numpy as np # Biblioteca para manejo de arreglos
import os # Biblioteca para manipulacion de archivos
```

[ ]:
```
# Datos desde URL
sample_url = "https://drive.google.com/uc?
 ↪id=1zO8ekHWx9U7mrbx_0Hoxxu6od7uxJqWw&export=download"
sample_data = pd.read_csv(sample_url)
sample_data.head()
```

[ ]:

	Index	Customer Id	First Name	Last Name	Company	City	Country	Phone 1	Phone 2	Email	Subscription Date	Website
0	1	DD37Cf93aecA6Dc	Sheryl	Baxter	Rasmussen Group	East Leonard	Chile	229.077.5154	397.884.0519x718	zunigavanessa@smith.info	2020-08-24	http://www.stephenson.com/
1	2	1Ef7b82A4CAAD10	Preston	Lozano	Vega-Gentry	East Jimmychester	Djibouti	5153435776	686-620-1820x944	vmata@colon.com	2021-04-23	http://www.hobbs.com/
2	3	6F94879bDAfE5a6	Roy	Berry	Murillo-Perry	Isabelborough	Antigua and Barbuda	+1-539-402-0259	(496)978-3969x58947	beckycarr@hogan.com	2020-03-25	http://www.lawrence.com/
3	4	5Cef8BFA16c5e3c	Linda	Olsen	Dominguez, Mcmillan and Donovan	Bensonview	Dominican Republic	001-808-617-6467x12895	+1-813-324-8756	stanleyblackwell@benson.org	2020-06-02	http://www.good-lyons.com/
4	5	053d585Ab6b3159	Joanna	Bender	Martin, Lang and Andrade	West Priscilla	Slovakia (Slovak Republic)	001-234-203-0635x76146	001-199-446-3860x3486	colinalvarado@miles.net	2021-04-17	https://goodwin-ingram.com/

1.2 Escritura de datos

Para escribir un archivo en formato cvs se utiliza la función `to_csv` de la biblioteca `pandas`. Por ejemplo, para escribir un archivo llamado `datos.csv` se utiliza la siguiente instrucción:

```
df.to_csv('datos.csv', index=False)
```

```
df = pd.DataFrame(data=datos, columns=nombres_columnas)
```

Donde `df` es un objeto de tipo `DataFrame` de la biblioteca `pandas`.

[ ]:
```
# Ejemplo: Medidas se almacenan en una lista de listas
measurements = [['Sun', 146, 152],
                                ['Moon', 0.36, 0.41],
                                ['Mercury', 82, 217],
                                ['Venus', 38, 261],
                                ['Mars', 56, 401],
                                ['Jupiter', 588, 968],
                                ['Saturn', 1195, 1660],
                                ['Uranus', 2750, 3150],
                                ['Neptune', 4300, 4700],
                                ['Halley\'s comet', 6, 5400]]

# los nombres de las columnas se almacenan en la variable header
header = ['Celestial bodies ','MIN', 'MAX']

# guardar el DataFrame en la variable celestial
celestial = pd.DataFrame(data=measurements, columns=header)
celestial
```

[ ]:

	Celestial bodies	MIN	MAX
0	Sun	146.00	152.00
1	Moon	0.36	0.41
2	Mercury	82.00	217.00
3	Venus	38.00	261.00
4	Mars	56.00	401.00
5	Jupiter	588.00	968.00
6	Saturn	1195.00	1660.00
7	Uranus	2750.00	3150.00
8	Neptune	4300.00	4700.00
9	Halley's comet	6.00	5400.00

Ejemplo: Preferencias musicales de las ciudades de Springfield y Shelbyville

https://www.kaggle.com/code/cirilobomaye/limpieza-de-datos-musica/input

- `' 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.

[ ]:
```
# Cargar datos
mainpath = "./datasets/" #carpeta global
filename = "music_project_en.csv" #dataset
fullpath = os.path.join(mainpath, filename)
data = pd.read_csv (fullpath)
```

[ ]:
```
# Ver los primeros registros
data.head()
```

[ ]:
```
     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

         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
```

[ ]:
```
# Ver los ultimos registros
data.tail()
```

[ ]:
```
         userID                             Track         artist       genre  \
65074  729CBB09                           My Name         McLean         rnb
65075  D08D4A55  Maybe One Day (feat. Black Spade)    Blu & Exile         hip
65076  C5E3A0D5                          Jalopiina            NaN  industrial
65077  321D0506                      Freight Train  Chas McDevitt        rock
65078  3A64EF84       Tell Me Sweet Little Lies   Monica Lopez     country

            City        time        Day
65074  Springfield  13:32:28  Wednesday
65075  Shelbyville  10:00:00     Monday
65076  Springfield  20:09:26     Friday
65077  Springfield  21:43:59     Friday
65078  Springfield  21:59:46     Friday
```

[ ]:
```
# Ver registros random
data.sample(5)
```

[ ]:
```
         userID                                              Track  \
20154  F35E3084                                                NaN
60143  94B9E05B                  Liar (as made famous by Madcon)
15409  4E4F397C               Ethereal Glyph (Solarstone Retouch)
36387  85A5EDB5  Symphony No. 5 in B-Flat Major D. 485: III. Me…
1936   A7A9C4DF                                       Going Down

                           artist      genre       City        time  \
20154                         NaN        NaN  Springfield  08:31:19
60143  Hit Singles Incorporated        pop  Shelbyville  20:30:03
```

```
15409                    Outer Pulse      dance  Springfield  21:41:22
36387  Slovak Philharmonic Orchestra  classical  Springfield  14:19:29
1936                           Bugsy        hip  Springfield  08:57:37

            Day
20154     Monday
60143  Wednesday
15409     Friday
36387     Friday
1936      Friday
```

[ ]:
```
# Ver dimensiones
data.shape # (filas, columnas)
```

[ ]:
```
(65079, 7)
```

[ ]:
```
#cabeceras de las columnas
data.columns.values
```

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

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. Nombres con dos palabras sin guión bajo de separación (snake_case)

1.2.1 Descripciones numéricas y describe()

Una comprensión aún más avanzada de los datos se puede obtener con el método **describe()** , que muestra las principales medidas estadísticas del conjunto de datos, incluida la desviación estándar, los valores mínimos y máximos para *valores numéricos*

[ ]:
```
# Resumen estadístico de las variables numéricas
data.describe()
```

[ ]:

	userID	Track	artist	genre	City	time	Day
count	65079	63736	57512	63881	65079	65079	65079
unique	41748	39666	37806	268	2	20392	3
top	A8AE9169	Brand	Kartvelli	pop	Springfield	08:14:07	Friday
freq	76	136	136	8850	45360	14	23149

1.3 Tipos de datos

Los tipos de datos en pandas son similares a los tipos de datos en Python. Algunos de los tipos de datos más importantes son:

1. **object**: texto

2. **int64**: números enteros
3. **float64**: números decimales
4. **datetime64**: fecha y hora
5. **bool**: valores booleanos

[ ]:
```
# #tipo de datos (objetc=string)
data.dtypes
```

[ ]:
```
  userID    object
Track       object
artist      object
genre       object
  City      object
time        object
Day         object
dtype: object
```

1.4 Convertir tipos de datos

Para convertir los tipos de datos de una columna se utiliza la función `astype`. Por ejemplo, para convertir la columna `edad` a tipo `float` se utiliza la siguiente instrucción:

```
df['edad'] = df['edad'].astype(float)
```

1.4.1 Conversión de los valores de string a números

números: to_numeric() . Convierte los valores de columna al tipo float64 o int64 (número entero), según el valor de entrada.

El método to_numeric() tiene un parámetro errors . Este parámetro determina qué hará **to_numeric** al encontrar un valor no válido:

1. errors='raise' (predeterminado): se genera una excepción cuando se encuentra un valor incorrecto, lo que detiene la conversión a números.
2. errors='coerce' : los valores incorrectos se reemplazan por NaN .
3. errors='ignore' : los valores incorrectos no se modifican.

```
pd.to_numeric(df['column'], errors='raise' )
pd.to_numeric(df['column'], errors='coerce')
pd.to_numeric(df['column'], errors='ignore')
```

1.4.2 Conversión de strings a horas y fechas

datetime: datos especial que utilizamos cuando trabajamos con fechas y horas:

Para convertir los strings a fechas y horas, usamos el método **to_datetime()** de pandas. Los parámetros del método incluyen el nombre de la columna que contiene strings y el formato de fecha en un string.

`pd.to_datetime(df['column'], format='%d/%m/%Y %H:%M:%S')` Establecemos el formato de fecha utilizando un sistema de designación especial:

- %d : día del mes (01 a 31);
- %m : mes (01 a 12);
- %Y : año en cuatro dígitos (por ejemplo, 1994);
- %y : año en dos dígitos (por ejemplo, 94);
- Z o T : separador estándar para la fecha y la hora;
- %H : hora (00 a 23);
- %M : minuto (00 a 59);
- %S : segundo (00 a 59).

1.4.3 Conversión de strings a categorías

categorías: datos especiales que utilizamos cuando trabajamos con datos que tienen un número limitado de valores únicos.

Para convertir los strings a categorías, usamos el método **astype()** de pandas.

```
df['column'] = df['column'].astype('category')
```

Preprocesamiento de datos

El preprocesamiento de datos es una etapa importante en el análisis de datos. En esta etapa, se realizan tareas como la limpieza de datos, la transformación de datos y la reducción de datos.

[ ]:
```
'''
Renombrar columnas: Bucle en los encabezados para :
1. Poner todo en minúsculas
2. Eliminar los espacios
3. Remplazar espacio entre palabras por "_"
'''

new_col_names = []

for name in data.columns:
    # Luego, pon todas las letras en minúsculas
    name_lowered = name.lower()
    # Elimina los espacios al principio y al final
    name_stripped = name_lowered.strip()
    # Por último, reemplaza los espacios entre palabras por guiones bajos
    name_no_spaces = name_stripped.replace(' ', '_')
    # Agrega el nuevo nombre a la lista de nuevos nombres de columna
    new_col_names.append(name_no_spaces)

# Reemplaza los nombres anteriores por los nuevos
data.columns = new_col_names
data.head()
```

[ ]:
```
     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

          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
```

1.4.4 Valores faltantes

Los valores faltantes (NaN) pueden ser causados por errores en la recopilación de datos, errores en la entrada de datos o errores en la manipulación de datos.

Para manejar los valores faltantes, se pueden utilizar diferentes estrategias, como eliminar los valores faltantes, reemplazar los valores faltantes con un valor predeterminado o reemplazar los valores faltantes con un valor calculado.

En pandas se pueden encontrar con:

```
pd.notnull(data["artist"]).values.ravel().sum()
```

```
pd.isnull(data["artist"]).values.ravel().sum()
```

crea una lista de datos no nulos/nulos y luego suma los valores de la lista.

Con el método **isnull()** se puede encontrar los valores faltantes en un DataFrame. El método **isnull()** devuelve un DataFrame de valores booleanos, donde los valores True indican valores faltantes.

```
df.isnull()
```

y el método **sum()**:

```
df.isnull().sum()
```

Para eliminar los valores faltantes, usa el método **dropna()**:

```
df.dropna()
```

Para reemplazar los valores faltantes con un valor predeterminado, usa el método **fillna()**:

```
df.fillna(0)
```

Para reemplazar los valores faltantes con un valor calculado, usa el método **fillna()** con un valor calculado:

```
df.fillna(df.mean())
```

Para reemplazar los valores faltantes con un valor calculado, usa el método **fillna()** con un valor anterior:

```
df.fillna(method='ffill')
```

o posterior

```
df.fillna(method='bfill')
```

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

[ ]:
```
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 `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, pero ya que solo hay 1198 NaN y los valores no nulos son 61253, remplazarlos por `'unknown'` no deberia de afectar a los resultados ya que solo es el 1.9% del total de la muestra

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

for col in columns_to_replace:
    data[col].fillna('unknow', inplace=True)
```

[ ]:
```
# Contar valores ausentes
print(data.isna().sum())
```

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

1.4.5 Valores duplicados

Los valores duplicados pueden ser causados por errores en la recopilación de datos, errores en la entrada de datos o errores en la manipulación de datos.

Para encontrar los valores duplicados, se puede utilizar el método **duplicated()**:

```
df.duplicated()
```

Para eliminar los valores duplicados, se puede utilizar el método **drop_duplicates()**:

`df.drop_duplicates()`

[ ]:
```
# Contar duplicados explícitos
data.duplicated().sum()
```

[ ]:
```
3826
```

Cuando eliminas filas, a menudo también es importante actualizar el índice. Para hacerlo, llama al método reset_index(). Esto creará un nuevo DataFrame en el que:

1. Los índices del DataFrame original se ubicarán en una nueva columna llamada 'index'.
2. Los nuevos índices se establecerán en orden para todas las filas en el DataFrame.

[ ]:
```
# Eliminar duplicados explícitos
data = data.drop_duplicates().reset_index(drop=True)
```

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

[ ]:
```
0
```

Para descubrir duplicados menos evidentes, se deben buscar valores que pueden incluir:

1. Ortografías alternativas de la misma palabra (por ejemplo, 'jazz' y 'jass');
2. Faltas de ortografía (por ejemplo, 'jazzz' en lugar de 'jazz');
3. Mayúsculas inconsistentes.

unique() y nunique() son métodos que te ayudarán a encontrar valores únicos y contarlos.

`df['column'].nunique()`

[ ]:
```
data['genre'].nunique()# Número de valores únicos en la columna 'genre'
```

[ ]:
```
269
```

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.

[ ]:
```
# Inspeccionar los nombres de géneros no únicos
data.groupby('genre')['genre'].count()
```

[ ]:
```
genre
acid            1
acoustic        5
action          4
```

```
adult          24
africa         16
               …
vocal          93
western        97
world        1947
worldbeat       2
ïîï             1
Name: genre, Length: 269, dtype: int64
```

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 `data`, 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.

[ ]:
```python
# Función para reemplazar duplicados implícitos
def replace_wrong_genres(df, column, wrong_genres, correct_genres):
    for wrong in wrong_genres: #
        df[column] = df[column].replace(wrong_genres, correct_genres)
    return df
```

[ ]:
```python
# Eliminar duplicados implícitos
wrong_genres = ['hip', 'hop', 'hip-hop'] #lista de nombres mal escritos
name = 'hiphop' #genero correcto
data = replace_wrong_genres(data, 'genre', wrong_genres, name)
```

[ ]:
```python
# Comprobación de duplicados implícitos
print(data['genre'].sort_values().unique())
```

```
['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'
'unknow' 'urban' 'uzbek' 'variété' 'vi' 'videogame' 'vocal' 'western'
'world' 'worldbeat' 'ïîï']
```

[ ]:
```python
# Conversión de strings a horas
data['time'] = pd.to_datetime(data['time'], format='%H:%M:%S').dt.time
# Conversión de strings a categorías
data['city'] = data['city'].astype('category')
```

1.4.6 Variables dummy

Las variables dummy son variables binarias que representan categorías. Se utilizan para representar variables categóricas en modelos de regresión.

Para crear variables dummy en pandas, se utiliza la función **get_dummies()**:

```
pd.get_dummies(df['column'])
```

Para agregar las variables dummy al DataFrame, se utiliza la función **concat()**:

```
pd.concat([df, pd.get_dummies(df['column'])], axis=1)
```

El parámetro **axis=1** se utiliza para concatenar las columnas horizontalmente.

[ ]:
```
# Variables dummies

# Crear variables dummies
city_dummy  = pd.get_dummies(data['city'], prefix='city')
#city_dummy  = pd.get_dummies(data['city'], prefix='city', drop_first=True)
city_dummy.head()
```

[ ]:

	city_Shelbyville	city_Springfield
0	1	0
1	0	1
2	1	0
3	1	0
4	0	1

[ ]:
```
# Concatenar el DataFrame original con el DataFrame de variables dummies
data_dummy = pd.concat([data, city_dummy], axis=1)
data_dummy.head()
```

[ ]:

	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	

	city	time	day	city_Shelbyville	city_Springfield
0	Shelbyville	20:28:33	Wednesday	1	0
1	Springfield	14:07:09	Friday	0	1
2	Shelbyville	20:58:07	Wednesday	1	0
3	Shelbyville	08:37:09	Monday	1	0
4	Springfield	08:34:34	Monday	0	1

Todo definido en una función

[ ]:
```
def createDummies(df, var_name):
    dummy = pd.get_dummies(df[var_name], prefix=var_name)
    df = df.drop(var_name, axis = 1)
    df = pd.concat([df, dummy ], axis = 1)
    return df
```

[ ]:
```
# mostrar información del DataFrame
def info(df):
    display(df.head(10))
    print()
    print(df.info())
    print()
    print(df.describe())
    print()
```

```
    print('Duplicated: ',df.duplicated().sum())
    print()
    print('Null values %:')
    print(100*df.isnull().sum()/len(df))

info(data)
```

```
     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            unknow   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


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61253 entries, 0 to 61252
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   userid  61253 non-null  object
 1   track   61253 non-null  object
 2   artist  61253 non-null  object
 3   genre   61253 non-null  object
 4   city    61253 non-null  category
 5   time    61253 non-null  object
 6   day     61253 non-null  object
dtypes: category(1), object(6)
memory usage: 2.9+ MB
None
```

```
          userid   track  artist  genre         city      time     day
count      61253   61253   61253  61253        61253     61253   61253
unique     41748   39667   37807    266            2     20392       3
top     A8AE9169  unknow  unknow    pop  Springfield  08:14:07  Friday
freq          71    1262    7097   8323        42741        13   21840

Duplicated:  0

Null values %:
userid    0.0
track     0.0
artist    0.0
genre     0.0
city      0.0
time      0.0
day       0.0
dtype: float64
```

[ ]:
```
data.columns.values.tolist()
```

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

1.5 Crear un subconjunto de datos

[ ]:
```
desired_columns = ['genre', 'city', 'time', 'day'] #crear una lista de columnas
subset = data[desired_columns] #crear el subset de columnas de la lista
subset.head()
```

[ ]:
```
   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
```

[ ]:
```
subset_first_20 = data[["genre", "city"]][:20]
subset_first_20
```

[ ]:
```
          genre         city
0          rock  Shelbyville
1          rock  Springfield
2           pop  Shelbyville
3          folk  Shelbyville
4         dance  Springfield
5        rusrap  Shelbyville
6         dance  Springfield
7         dance  Springfield
```

```
8        ruspop  Springfield
9         dance  Shelbyville
10        world  Springfield
11   electronic  Springfield
12          pop  Springfield
13        dance  Springfield
14          pop  Shelbyville
15       unknow  Springfield
16  alternative  Shelbyville
17         folk  Shelbyville
18     children  Shelbyville
19          rnb  Springfield
```

1.5.1 Filtrado con loc e iloc: Filtrado de filas y columnas a la vez

[ ]:
```
data.iloc[1:10, 3:6]#10 filas y columnas de la 3 a la 6
#loc para etiquetas y iloc para posiciones
```

[ ]:
```
    genre         city      time
1    rock  Springfield  14:07:09
2     pop  Shelbyville  20:58:07
3    folk  Shelbyville  08:37:09
4   dance  Springfield  08:34:34
5  rusrap  Shelbyville  13:09:41
6   dance  Springfield  13:00:07
7   dance  Springfield  20:47:49
8  ruspop  Springfield  09:17:40
9   dance  Shelbyville  21:20:49
```

1.5.2 Filtrar por condición booleana

Para filtrar un DataFrame por una condición booleana, se puede utilizar la siguiente sintaxis:

```
df[df['column'] > value]
```

```
df[df['column'] == value]
```

```
df[df['column'] != value]
```

```
df[df['column'].isin([value1, value2])]
```

```
df[df['column'].notnull()]
```

```
df[df['column'].isnull()]
```

```
df[df['column'].str.contains('value')]
```

```
df[df['column'].str.startswith('value')]
```

```
df[df['column'].str.endswith('value')]
```

```
df[df['column'].str.len() > value]
```

[ ]:
```
#filtrar por genero pop, ciudad New York y dia Lunes

data_filter = data[(data['genre'] == 'pop') & (data['day'] == 'Monday')]
data_filter
```

[ ]:
```
         userid                      track               artist genre  \
12     FF3FD2BD                      Truth               Bamboo   pop
26     982219FD    We Not Speak Americano          Genio Band   pop
57     E35653A8               Yurak Ezilar      Shoxruz Abadiya   pop
60     E2F8BE12                      Banga              unknow   pop
66     B103C670       When I Was Your Man        Kriss Ramirez   pop
…         …                         …                  …   …
61096  2705A54D            Confidence Man  The Jeff Healey Band   pop
61113  1301E0AE                  Hold On       Wilson Phillips   pop
61147  21D8803B  Total Eclipse of the Heart         Nicki French   pop
61180   E13B82C                  No Love        Salt Cathedral   pop
61197  F82168F2               These Nights     Langston Francis   pop

              city      time     day
12     Springfield  09:19:49  Monday
26     Shelbyville  20:03:03  Monday
57     Springfield  20:07:56  Monday
60     Springfield  08:45:42  Monday
66     Shelbyville  14:42:11  Monday
…            …       …     …
61096  Springfield  14:07:39  Monday
61113  Springfield  09:41:06  Monday
61147  Springfield  21:04:47  Monday
61180  Springfield  08:30:33  Monday
61197  Springfield  08:32:10  Monday

[2886 rows x 7 columns]
```

2 Insertar nuevas filas en el dataframe

Para insertar nuevas filas en un DataFrame, se utiliza el método **append()**. El método **append()** agrega filas al final del DataFrame.

```
df.append({'column1': value1, 'column2': value2}, ignore_index=True)
```

El parámetro **ignore_index=True** se utiliza para restablecer los índices de las filas agregadas.

[ ]:
```
#conaenar dos dataframes
data1 = data.iloc[0:10, :]
data2 = data.iloc[10:20, :]
data_concat = pd.concat([data1, data2], axis=0)
data_concat
```

[ ]:

	userid	track	\
0	FFB692EC	Kamigata To Boots	
1	55204538	Delayed Because of Accident	
2	20EC38	Funiculì funiculà	
3	A3DD03C9	Dragons in the Sunset	
4	E2DC1FAE	Soul People	
5	842029A1	Chains	
6	4CB90AA5	True	
7	F03E1C1F	Feeling This Way	
8	8FA1D3BE	L'estate	
9	E772D5C0	Pessimist	
10	BC5A3A29	Gool la Mita	
11	8B5192C0	Is There Anybody Out There? (Panoramic Paralys…	
12	FF3FD2BD	Truth	
13	CC782B0F	After School Special	
14	94EB25C2	Make Love Whenever You Can	
15	E3C5756F	unknow	
16	81D05C7D	SLAVES OF FEAR	
17	39DE290E	Hallo Hallo	
18	58AE138A	Pat-a-Cake	
19	772F5B59	Sweetback	

	artist	genre	city	time	day
0	The Mass Missile	rock	Shelbyville	20:28:33	Wednesday
1	Andreas Rönnberg	rock	Springfield	14:07:09	Friday
2	Mario Lanza	pop	Shelbyville	20:58:07	Wednesday
3	Fire + Ice	folk	Shelbyville	08:37:09	Monday
4	Space Echo	dance	Springfield	08:34:34	Monday
5	Obladaet	rusrap	Shelbyville	13:09:41	Friday
6	Roman Messer	dance	Springfield	13:00:07	Wednesday
7	Polina Griffith	dance	Springfield	20:47:49	Wednesday
8	Julia Dalia	ruspop	Springfield	09:17:40	Friday
9	unknow	dance	Shelbyville	21:20:49	Wednesday
10	Shireen Abdul Wahab	world	Springfield	14:08:42	Monday
11	Pink Floyd Floydhead	electronic	Springfield	13:47:49	Monday
12	Bamboo	pop	Springfield	09:19:49	Monday
13	Detroit Grand Pubahs	dance	Springfield	20:04:12	Friday
14	Arabesque	pop	Shelbyville	13:22:08	Wednesday
15	unknow	unknow	Springfield	09:24:51	Monday
16	HEALTH	alternative	Shelbyville	20:54:48	Monday
17	Die Klima Hawaiians	folk	Shelbyville	14:36:47	Monday
18	Mother Goose Club	children	Shelbyville	20:04:56	Monday
19	The Fabulous Three	rnb	Springfield	13:56:42	Wednesday

[ ]:
```
# crear una nueva columna con el nombre 'weekend' que contenga True si el día␣
↪es 'Saturday' o 'Sunday' y False en caso contrario
```

```
data['weekend'] = data['day'].apply(lambda x: True if x in ['Saturday',␣
 ↪'Sunday'] else False)
data.head()
```

[ ]:

	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	

	city	time	day	weekend
0	Shelbyville	20:28:33	Wednesday	False
1	Springfield	14:07:09	Friday	False
2	Shelbyville	20:58:07	Wednesday	False
3	Shelbyville	08:37:09	Monday	False
4	Springfield	08:34:34	Monday	False

2.1 Referencias

https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html