# SPRINT 8: INTEGRACIÓN DE POWER BI CON PYTHON

**Utiliza los scripts de Python creados previamente en Sprint 8.1 para generar visualizaciones personalizadas con las bibliotecas Seaborn y Matplotlib. Estas visualizaciones se integrarán en el informe de Power BI para ofrecer una comprensión más profunda de la capacidad del lenguaje de programación en la fuente Power BI.**

## PROBLEMA 1: IMPORTACIÓN DE DATAFRAMES

Al intentar obtener datos de los dataframes utilizados en Python, no me aparecía nada en la vista previa y no me daba ningún error, lo cual quiere decir que el script se ejecutaba bien, pero por algún motivo no aparecían los dataframes.

Script ejecutado:

Script de Python

Script

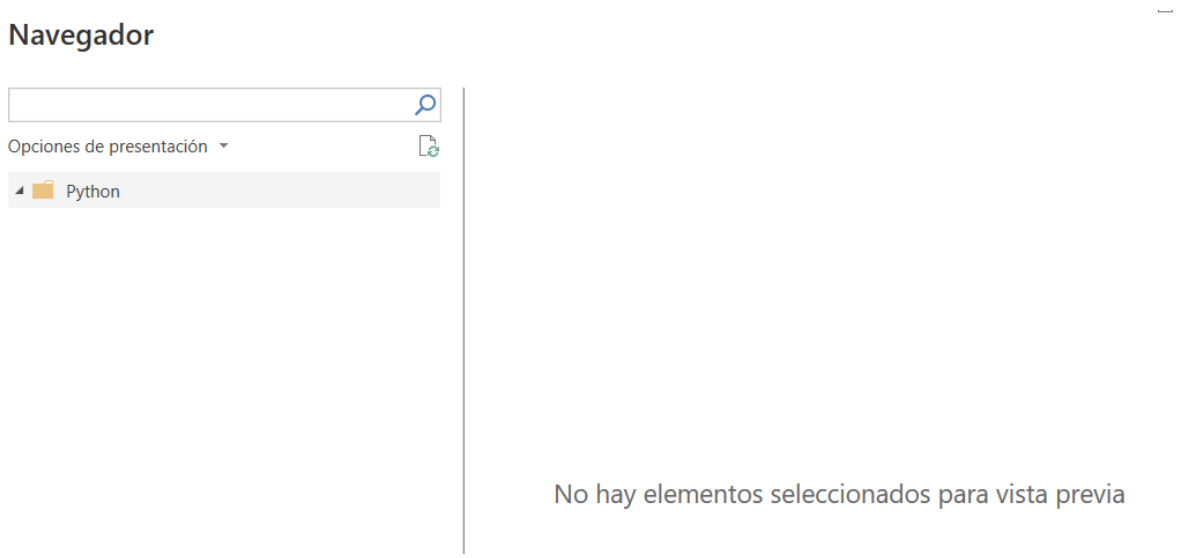
```
import pandas as pd
from sqlalchemy import create_engine

conexion = create_engine(f'mysql+mysqlconnector://root:joonie@localhost/operations')
tablas_db = ['transaction','company','credit_card','user','transaction_product','product']
dataframe = {}

for tabla in tablas_db:
        query = f'SELECT * FROM {tabla}'
        dataframe[tabla] = pd.read_sql(query, conexion)

conexion = conexion.dispose()
```

Vista previa:




**SOLUCIÓN: asignar los diferentes dataframes a una variable.**

En el Sprint 8.01, al importar la base de datos de MySQL en Python, almacené las diferentes tablas en un diccionario llamado *dataframe*. A lo largo de ese sprint trabajé con la nomenclatura *dataframe*['tabla'] para acceder a cada dataframe.

Cuando ejecuté el script de la imagen, el bucle funcionaba y se importaban los dataframes, pero en ningún momento llamaba explícitamente a los dataframes almacenados. Por tanto, mi solución fue asignar los dataframes del diccionario a diferentes variables.

Para simplificar los nombres de los dataframes, en lugar de ponerles la nomenclatura utilizada en el Sprint 8.01, simplemente usé el nombre de la 'tabla' para simplificar.

Además, en el Sprint 8.01 creé dos columnas con datos nuevas que he utilizado en varios ejercicios: columnas *age* y *continents*. Consideré que la mejor opción era guardarlas en el modelo en lugar de ejecutar repetidamente el código para crearlas en los diversos ejercicios. Asimismo, cambié los nombres de algunas columnas.

Script de Python modificado:

Script

```
import pandas as pd
from sqlalchemy import create_engine

conexion = create_engine(f'mysql+mysqlconnector://root:joonie@localhost/operations')
tablas_db = ['transaction','company','credit_card','user','transaction_product','product']
dataframe = {}

for tabla in tablas_db:
        query = f'SELECT * FROM {tabla}'
        dataframe[tabla] = pd.read_sql(query, conexion)

conexion = conexion.dispose()
```

```
#cambio dtype de objeto a datetime
dataframe['user']['birth_date'] = pd.to_datetime(dataframe['user']['birth_date'])
```

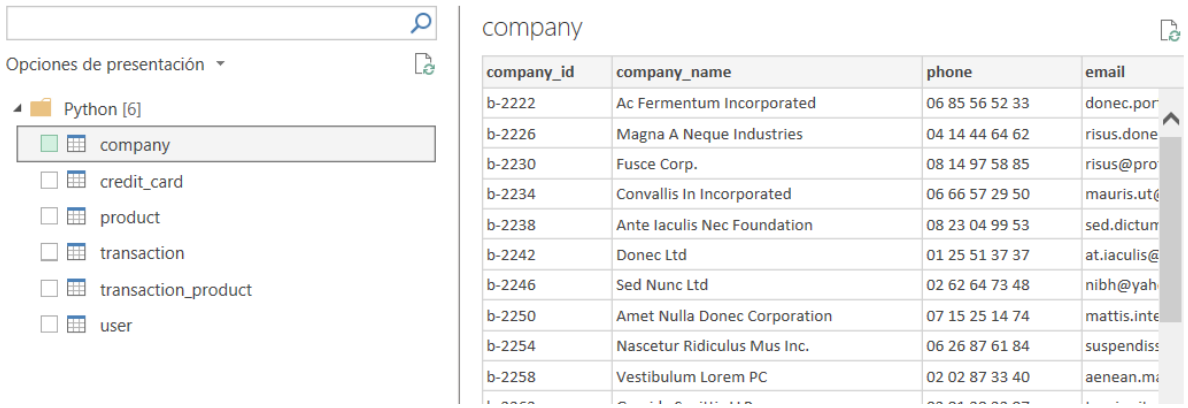
```
#función para calcular la edad
def calculate_age(birth_date):
    today = pd.Timestamp.now()
    age = today.year - birth_date.year - ((today.month, today.day) < (birth_date.month, birth_
    return age
dataframe['user']['age'] = dataframe['user']['birth_date'].apply(calculate_age)

#creación de una nueva columna para clasificar los países de las empresas por continente
Continents = {'Europe': ['Germany','Italy','United Kingdom','Sweden','Norway','Ireland','Nethe
              'Asia': ['China'],
              'North America': ['Canada','United States'],
              'Oceania': ['New Zealand','Australia']}
```

```
def assign_continent(country):
    if country in Continents['Europe']:
         return 'Europe'
    elif country in Continents['Asia']:
        return 'Asia'
    elif country in Continents['North America']:
        return 'North America'
    elif country in Continents['Oceania']:
        return 'Oceania'
    else:
        return 'Undefined Continent'
dataframe['company']['Continent'] = dataframe['company']['country'].apply(assign_continent)
```

```
#simplifico los nombres de dataframes para usar en PowerBI; cambio nombre a algunas columnas
transaction = dataframe['transaction'].rename(columns={'id':'transaction_id'})
company = dataframe['company'].rename(columns={'country':'company_country'})
credit_card = dataframe['credit_card'].rename(columns={'id':'card_id','user_id':'card_owner'}
user = dataframe['user'].rename(columns={'id':'user_id','country':'user_country'})
transaction_product = dataframe['transaction_product']
product = dataframe['product'].rename(columns={'id':'product_id'})
```

Vista previa al ejecutar el Script:

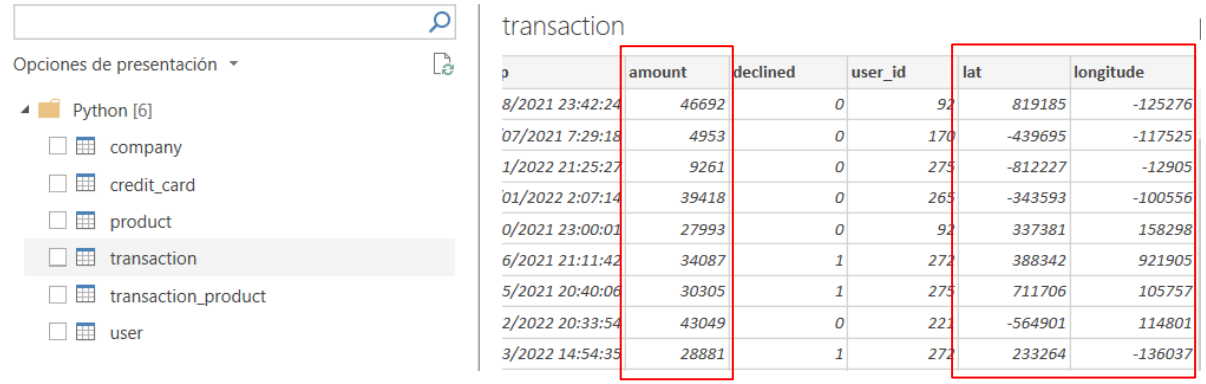



| company_id | company_name | phone | email |
|---|---|---|---|
| b-2222 | Ac Fermentum Incorporated | 06 85 56 52 33 | donec.por |
| b-2226 | Magna A Neque Industries | 04 14 44 64 62 | risus.done |
| b-2230 | Fusce Corp. | 08 14 97 58 85 | risus@pro |
| b-2234 | Convallis In Incorporated | 06 66 57 29 50 | mauris.ut@ |
| b-2238 | Ante Iaculis Nec Foundation | 08 23 04 99 53 | sed.dictum |
| b-2242 | Donec Ltd | 01 25 51 37 37 | at.iaculis@ |
| b-2246 | Sed Nunc Ltd | 02 62 64 73 48 | nibh@yah |
| b-2250 | Amet Nulla Donec Corporation | 07 15 25 14 74 | mattis.inte |
| b-2254 | Nascetur Ridiculus Mus Inc. | 06 26 87 61 84 | suspendiss |
| b-2258 | Vestibulum Lorem PC | 02 02 87 33 40 | aenean.ma |

## PROBLEMA 2: INCORRECTA DETECCIÓN DEL SEPARADOR DE DECIMALES

Al importar los datos de los dataframes, no se detectaron bien los puntos como separadores de comas, a pesar de que en Python no tuve ningún problema en utilizarlos. Sigo sin entender muy bien el motivo de este error, ya que mi sistema operativo tiene configurado el punto como separador de decimales.

Vista previa de la tabla 'transaction':

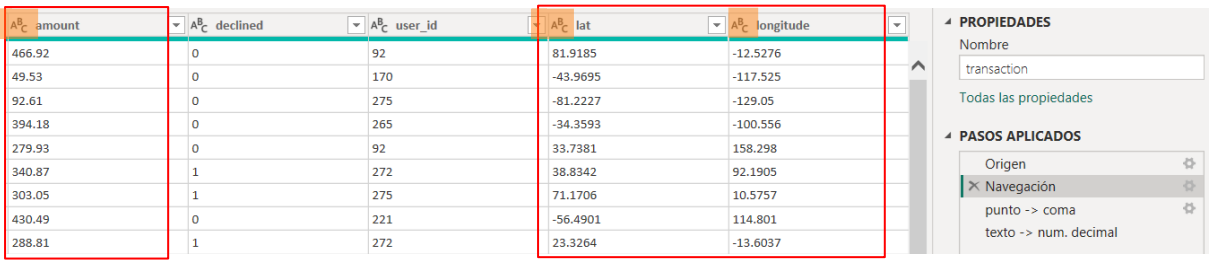



| p | amount | declined | user_id | lat | longitude |
|---|---|---|---|---|---|
| 8/2021 23:42:24 | 46692 | 0 | 92 | 819185 | -125276 |
| 07/2021 7:29:18 | 4953 | 0 | 170 | -439695 | -117525 |
| 1/2022 21:25:27 | 9261 | 0 | 275 | -812227 | -12905 |
| 01/2022 2:07:14 | 39418 | 0 | 265 | -343593 | -100556 |
| 0/2021 23:00:01 | 27993 | 0 | 92 | 337381 | 158298 |
| 6/2021 21:11:42 | 34087 | 1 | 272 | 388342 | 921905 |
| 5/2021 20:40:06 | 30305 | 1 | 275 | 711706 | 105757 |
| 2/2022 20:33:54 | 43049 | 0 | 221 | -564901 | 114801 |
| 3/2022 14:54:35 | 28881 | 1 | 272 | 233264 | -136037 |

**SOLUCIÓN: utilizar PowerQuery para transformar puntos a comas y cambiarles el tipo de dato a decimal.**

Ante este problema, en lugar de cargar los datos directamente, opté por entrar en editor PowerQuery y ver en qué punto surgió el problema.

Averigüé que todos los datos de origen se cargaron como tipo Texto, con el punto como separador de decimales.




| amount | declined | user_id | lat | longitude |
|---|---|---|---|---|
| 466.92 | 0 | 92 | 81.9185 | -12.5276 |
| 49.53 | 0 | 170 | -43.9695 | -117.525 |
| 92.61 | 0 | 275 | -81.2227 | -129.05 |
| 394.18 | 0 | 265 | -34.3593 | -100.556 |
| 279.93 | 0 | 92 | 33.7381 | 158.298 |
| 340.87 | 1 | 272 | 38.8342 | 92.1905 |
| 303.05 | 1 | 275 | 71.1706 | 10.5757 |
| 430.49 | 0 | 221 | -56.4901 | 114.801 |
| 288.81 | 1 | 272 | 23.3264 | -13.6037 |

Al cargar los datos, PowerBI hace una detección automática del tipo de dato de cada columna y los cambia según ese criterio (en la imagen de vista previa comentada al inicio de este problema se ha aplicado ese cambio automático).

Sabiendo esto, lo primero que hice es reemplazar puntos por comas en todas las columnas que contenían decimales ('transaction': amount, lat, longitude; 'product': price, weight). Y, finalmente, modifiqué el tipo de dato de esas columnas por número decimal. Adicionalmente, modifiqué el tipo de dato de otras columnas de las tablas afectadas, ya que no se aplicó la detección automática.

Paso 1: cambio punto por coma

```
= Table.ReplaceValue(transaction1,".",",",Replacer.ReplaceText,{"amount", "lat", "longitude"})
```

| estamp | amount | declined | user_id | lat | longitude |
|---|---|---|---|---|---|
| 8-28 23:42:24 | 466,92 | 0 | 92 | 81,9185 | -12,5276 |
| 7-26 07:29:18 | 49,53 | 0 | 170 | -43,9695 | -117,525 |
| 1-06 21:25:27 | 92,61 | 0 | 275 | -81,2227 | -129,05 |
| 1-26 02:07:14 | 394,18 | 0 | 265 | -34,3593 | -100,556 |
| 0-26 23:00:01 | 279,93 | 0 | 92 | 33,7381 | 158,298 |

Paso 2: cambio tipo de dato

```
= Table.TransformColumnTypes(#"punto -> coma",{{"amount", type number}, {"lat", type number}, {"longitude", type number}, {"declined",
```

| estamp | 1.2 amount | 1²3 declined | 1²3 user_id | 1.2 lat | 1.2 longitude |
|---|---|---|---|---|---|
| *28/08/2021 23:42:24* | *466.92* | *0* | *92* | *81.9185* | *-12.5276* |
| *26/07/2021 7:29:18* | *49.53* | *0* | *170* | *-43.9695* | *-117.525* |
| *06/01/2022 21:25:27* | *92.61* | *0* | *275* | *-81.2227* | *-129.05* |
| *26/01/2022 2:07:14* | *394.18* | *0* | *265* | *-34.3593* | *-100.556* |
| *26/10/2021 23:00:01* | *279.93* | *0* | *92* | *33.7381* | *158.298* |

Curiosamente, al modificar el tipo de dato a decimal, los valores aparecen con punto, ya que es como lo tengo definido en mi sistema operativo.

## PROBLEMA 3: FALTA DE VALORES AL EJECUTAR OBJETOS VISUALES DE PYTHON

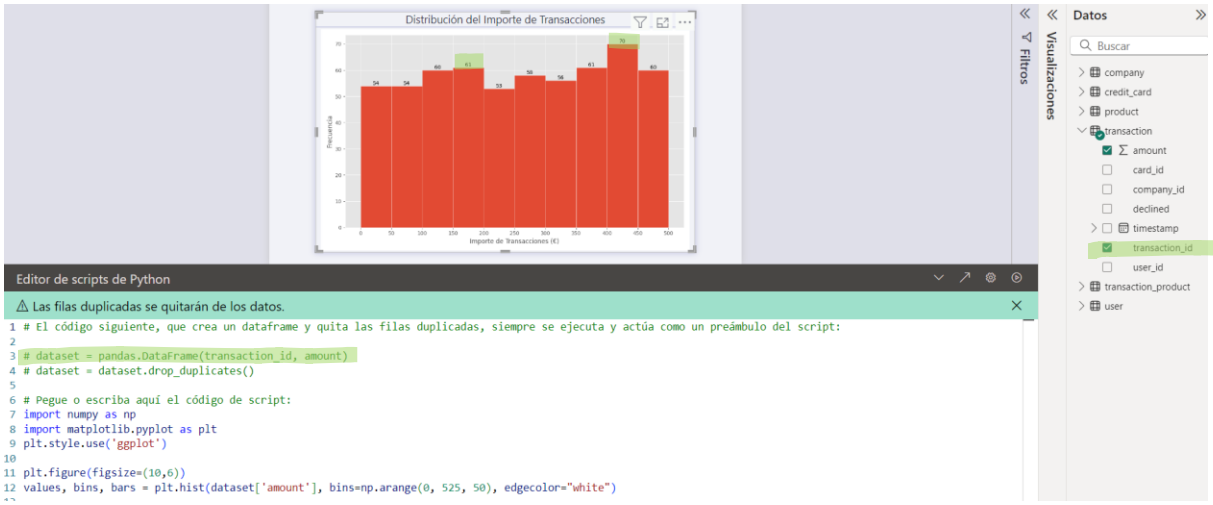
Al ejecutar el código del ejercicio 1.1, me di cuenta de que el gráfico resultado era incorrecto, ya que en el rango de 150-200€ tendrían que haber 61 registros, y de 400-450€ tendrían que ser 70 registros (que son los resultados obtenidos en Python).

Descubrí que la causa del problema estaba en que, tal y como aparece escrito en el editor de scripts de Python, las filas duplicadas se quitan de los datos.

Realmente sabemos que en la tabla 'transaction' no tenemos duplicados, ya que el id de transacción es *primary key*. En el ejemplo anterior se ve que únicamente seleccioné la columna del importe, que es la que necesito para ejecutar el código, y en caso de importe si que puede haber cantidades duplicadas.

**<u>SOLUCIÓN:</u> seleccionar una columna que haga de identificador único para evitar que se eliminen datos aparentemente duplicados.**

Como solución, lo único que tuve que hacer es seleccionar transaction_id para que también forme parte del dataset definido, a pesar de que no lo utilice en ninguna parte del código.




Para evitar este problema, al final terminé seleccionando la columna de transaction_id en todas las visualizaciones, just in case.

## PROBLEMA 4: NO ENTENDER BIEN COMO FUNCIONAN LOS OBJETOS VISUALES DE PYTHON

En el ejercicio 1.3, yo quería representar la distribución geográfica tanto de la totalidad de los compradores como de los recurrentes, en una misma visualización. Me encontré con que, en Python, yo realizaba cálculo de transacciones por usuario y después aplicaba un filtro y merge para obtener los compradores recurrentes. Al principio pensé que para integrar ese código en PowerBI, tenía que crear: o bien medidas DAX para calcular o filtrar, o bien hacer una tabla nueva en el modelo.

De hecho, después de varios intentos sin éxito con medidas DAX, opté por hacer tabla nueva con los usuarios filtrados. Sin embargo, a pesar de que la relación con el resto del modelo creado aparentemente era correcta, cuando ejecutaba el código no obtenía el resultado esperado, que era el de tener dos "pie charts" en una misma visualización, uno para todos los compradores y otro para los compradores filtrados. La tabla creada generaba algún conflicto en el modelo y los países se filtraban para ambos "pie chart".

Mi siguiente propuesta fue hacer dos visualizaciones por separado, pero esta solución también tenía sus desventajas. La principal es que la visualización de la nueva tabla no se veía afectada por un segmentado de datos. En estos casos, el segmentado de datos puede ser muy útil para aplicar un filtro adicional a la visualización sin tener que modificar nada del código de Python. Por tanto, esta

propuesta tampoco me convencía y tuve que investigar más a fondo sobre como representar la visualización con tan solo mi código de Python.

**<u>SOLUCIÓN:</u> ejecutar todas las partes del código que influyen en la construcción de la visualización.**

Tal y como he comentado antes, para llegar a identificar los usuarios recurrentes tuve que hacer varias modificaciones con mi dataframe y crear nuevas variables.

Sabía que no hacía falta trasladar el código de merge a PowerBI porque al importar los dataframes ya se establecían las relaciones en el modelo y las operaciones merge eran redundantes. Con tan solo seleccionar las columnas que tocan de diferentes tablas, ya se hacía JOIN/merge.

Lo que no sabía es que, para hacer operaciones de agregación o filtrado, no hacía falta ningún tipo de fórmula DAX. Únicamente se tiene que escoger correctamente las columnas y adaptar ligeramente el código de Python.

Por ejemplo, mi código original de Python era:

```
#cálculo de mediana de gasto por usuario y 'merge' con dataframe user
user_median_amount = pd.merge(dataframe['transaction'].groupby('user_id')['amount'].median().reset_index(name='median_amount
                              dataframe['user'],
                              on='user_id')
```

```
#recuento de transacciones por usuario y filtrado de los recurrentes
user_trans_count = dataframe['transaction']['user_id'].value_counts().reset_index(name='trans_count')
rec_users_filtered = user_trans_count[user_trans_count['trans_count'] > 2].reset_index(drop=True)
recurrent_users = pd.merge(rec_users_filtered, user_median_amount, on='user_id')

#recuento de usuarios por país
trans_users_country = user_median_amount.groupby('user_country').size().reset_index(name='user_count')
rec_users_country = recurrent_users.groupby('user_country').size().reset_index(name='rec_user_count')
```

- Merge entre la mediana de importe por usuario y todos los demás datos del dataframe user.
- Recuento de transacciones por usuario y filtrado de los usuarios recurrentes.
- Merge entre los identificadores de usuarios recurrentes y la tabla con datos de usuario y mediana de gastos.
- Recuento de los usuarios por país, en total y de recurrentes.

Para pasar correctamente este código a script en PowerBI, tuve que hacer las siguientes modificaciones:

```
#mediana de gasto y país de cada comprador
user_median_amount = dataset.groupby('user_id').agg({'amount': 'median', 'user_country': 'first'}).reset_index()

#recuento de transacciones por usuario y filtrado de los recurrentes
user_trans_count = dataset['user_id'].value_counts().reset_index(name='trans_count')
rec_users_filtered = user_trans_count[user_trans_count['trans_count'] > 2].reset_index(drop=True)
recurrent_users = pd.merge(rec_users_filtered, user_median_amount, on='user_id')

#recuento de usuarios por país
trans_users_country = user_median_amount.groupby('user_country').size().reset_index(name='user_count')
rec_users_country = recurrent_users.groupby('user_country').size().reset_index(name='rec_user_count')
```

- Seleccionar estas variables:
  - transaction: amount, user_id y transaction_id
  - user: user_country
- Agrupar el dataset por id del usuario y así calcular la mediana de importe e identificar el país del usuario → datos guardados en la variable user_median_amount
- Calcular la cantidad de transacciones por usuario y filtrado de los recurrentes.

- Merge entre los usuarios recurrentes filtrados y user_median_amount; en este caso sí que hay que hacer merge porque los datos específicos de usuarios recurrentes estaban guardados en una variable, no una tabla del modelo.
- Recuento de los usuarios por país, en total y de recurrentes.

Por tanto, una vez entendida esta lógica, simplemente tuve que copiar y pegar mi código original en todos los ejercicios y adaptarlo ligeramente. Otra cosa que me ayudó mucho a resolver estos ejercicios fue leer y entender los errores que me aparecían al ejecutar el script en PowerBI.

Gráfico final conseguido: