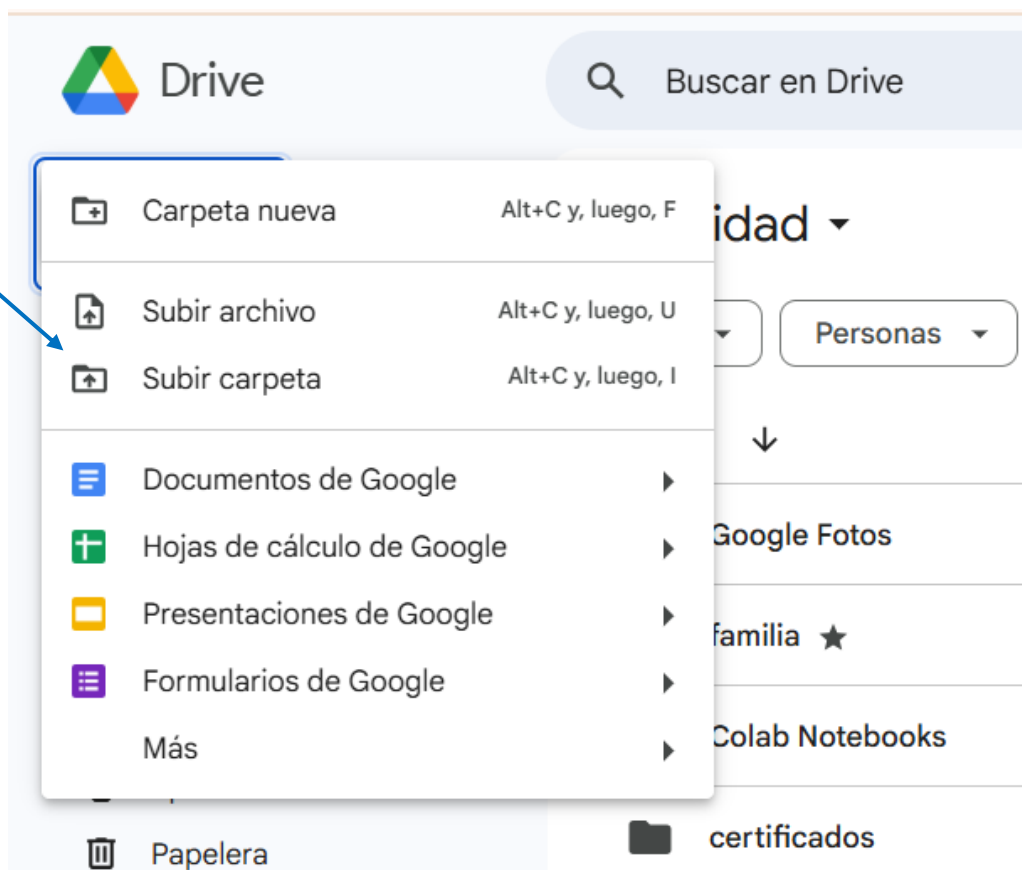
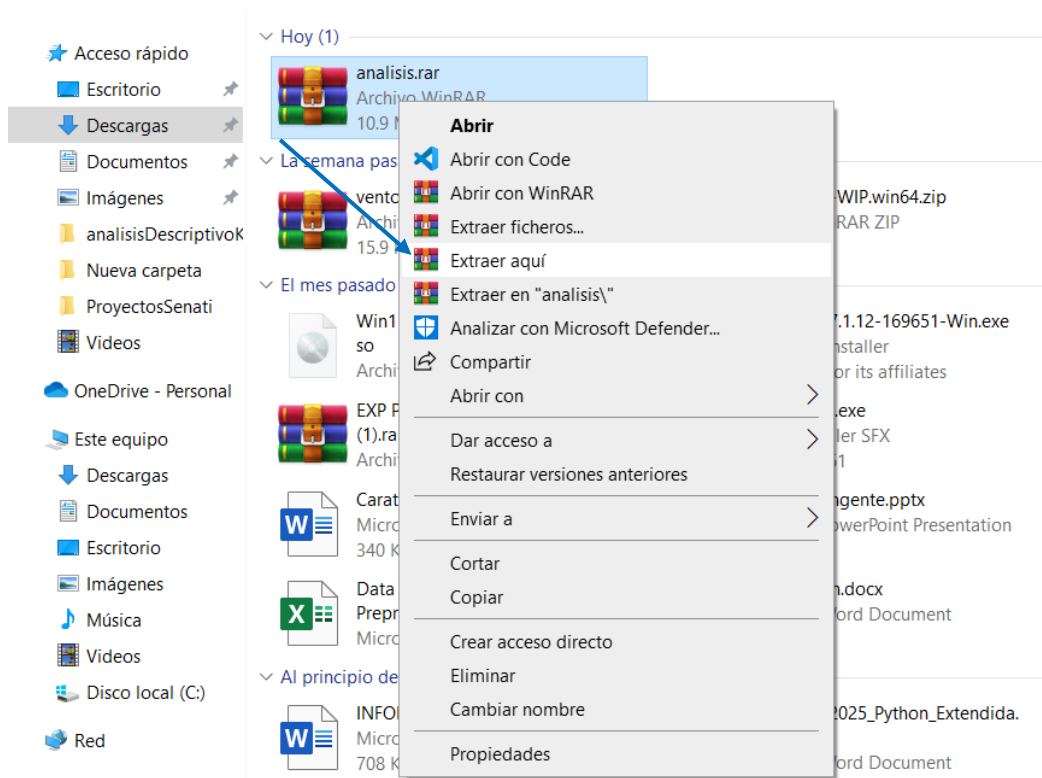
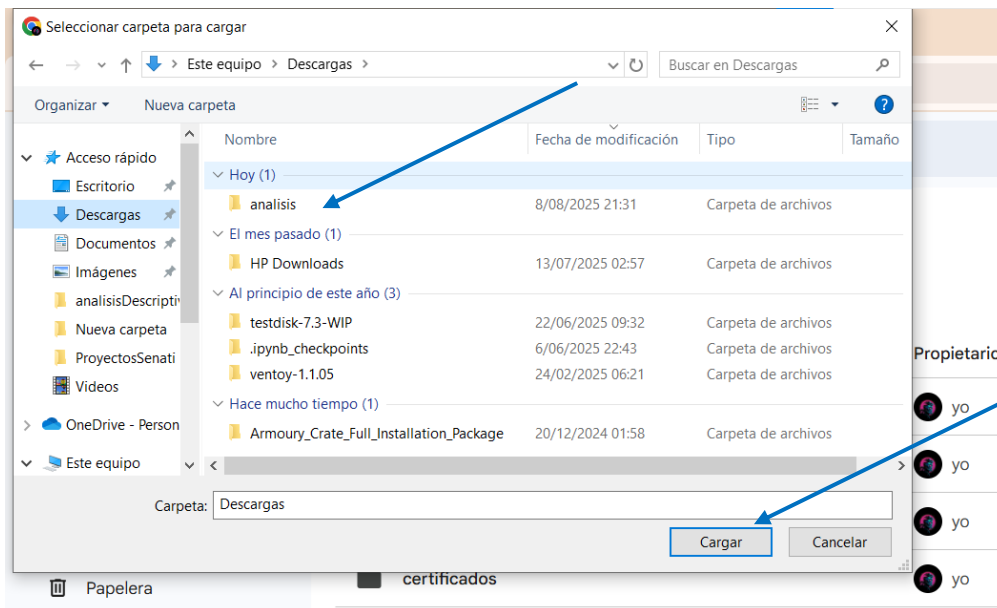


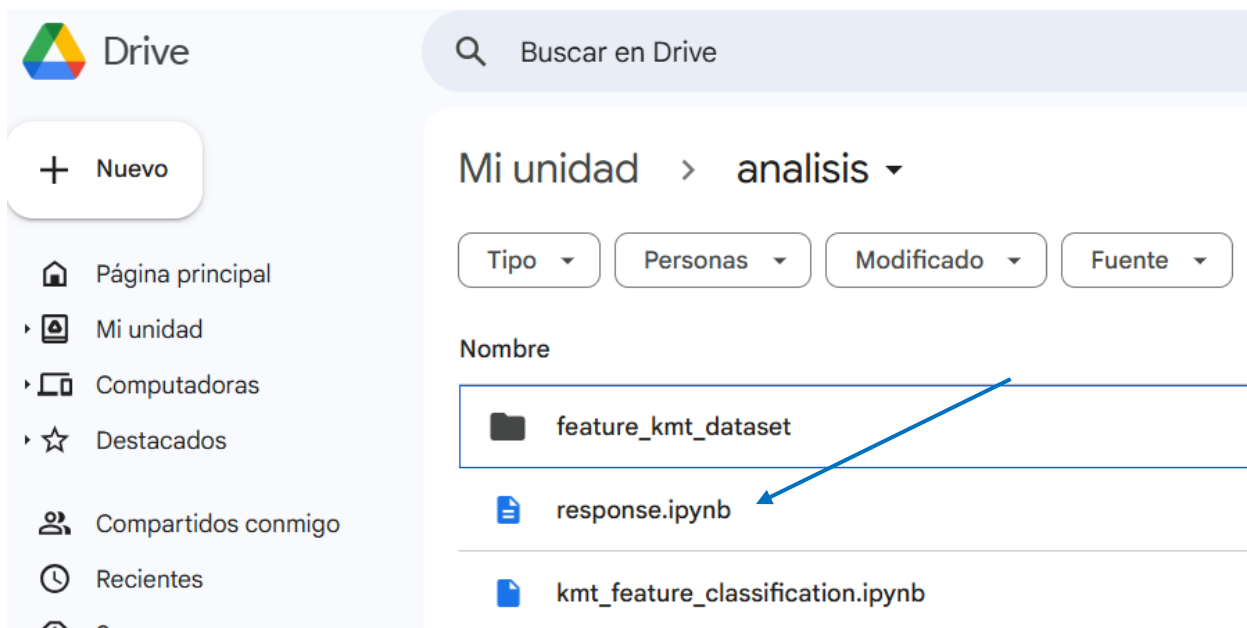
## Obtención de gráficos con Python, paso a paso.

1. Descomprimir el archivo proporcionado, ingrese a su drive, seleccione y cargue la carpeta analisis





Ingresar a ese archivo



2. **Primera celda:** carga las librerías necesarias para hacer el análisis, las librerías usadas son:
  - a. Pandas, Biblioteca principal para manipulación y análisis de datos en Python.
  - b. Seaborn: Biblioteca para visualización estadística de alto nivel; facilita histogramas, boxplots, pairplots, etc. Se integra con pandas y matplotlib.
  - c. Matplotlib: Biblioteca base para gráficos en Python.
  - d. scipy.stats: Funciones estadísticas:
    - i. mannwhitneyu: prueba no paramétrica para comparar dos distribuciones (como alternativa a t-test si no hay normalidad).

- ii. skew: asimetría de la distribución (si cola está a la derecha/izquierda).
- iii. kurtosis: curtosis (concentración/colas; ojo con la definición — ver nota más abajo).
- iv. spearmanr: correlación de rango de Spearman (no paramétrica, mide relación monotónica).



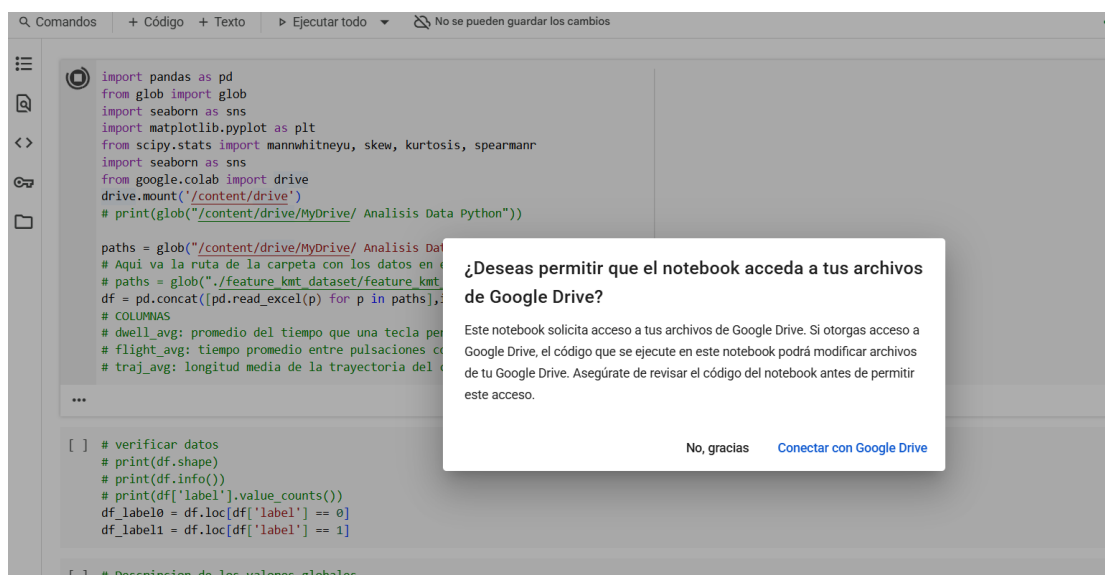
```
import pandas as pd
from glob import glob
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import mannwhitneyu, skew, kurtosis, spearmanr
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')
# print(glob("/content/drive/MyDrive/ Analisis Data Python"))

paths = glob("/content/drive/MyDrive/analisis/feature_kmt_dataset/feature_kmt_xlsx/feature_kmt_user *.xlsx")
# Aqui va la ruta de la carpeta con los datos en excel(los excels contiene el promedio de los datos)
# paths = glob("./feature_kmt_dataset/feature_kmt_xlsx/feature_kmt_user *.xlsx")
df = pd.concat([pd.read_excel(p) for p in paths],ignore_index=True)
# COLUMNAS
# dwell_avg: promedio del tiempo que una tecla permanece presionada.
# flight_avg: tiempo promedio entre pulsaciones consecutivas.
# traj_avg: longitud media de la trayectoria del cursor del ratón durante la sesión.
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

Presione play para cargar los archivos y ejecutar el primer script

3. Si Google le pide permisos del drive, concédalos. Presione Conectar con Google Drive



```
import pandas as pd
from glob import glob
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import mannwhitneyu, skew, kurtosis, spearmanr
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')
# print(glob("/content/drive/MyDrive/ Analisis Data Python"))

paths = glob("/content/drive/MyDrive/ Analisis Data Python/feature_kmt_dataset/feature_kmt_xlsx/feature_kmt_user *.xlsx")
# Aqui va la ruta de la carpeta con los datos en excel(los excels contiene el promedio de los datos)
# paths = glob("./feature_kmt_dataset/feature_kmt_xlsx/feature_kmt_user *.xlsx")
df = pd.concat([pd.read_excel(p) for p in paths],ignore_index=True)
# COLUMNAS
# dwell_avg: promedio del tiempo que una tecla permanece presionada.
# flight_avg: tiempo promedio entre pulsaciones consecutivas.
# traj_avg: longitud media de la trayectoria del cursor del ratón durante la sesión.
```

¿Deseas permitir que el notebook acceda a tus archivos de Google Drive?

Este notebook solicita acceso a tus archivos de Google Drive. Si otorgas acceso a Google Drive, el código que se ejecute en este notebook podrá modificar archivos de tu Google Drive. Asegúrate de revisar el código del notebook antes de permitir este acceso.

No, gracias [Conectar con Google Drive](#)

```
[ ] # verificar datos
# print(df.shape)
# print(df.info())
# print(df['label'].value_counts())
df_label0 = df.loc[df['label'] == 0]
df_label1 = df.loc[df['label'] == 1]

[ ] # Descripción de los valores globales
```

4. En la primera celda, esto crea un DataFrame para manejar toda la data y hacer mas sencillo el análisis, todo se almacena en la variable 'df'

```
df = pd.concat([pd.read_excel(p) for p in paths],ignore_index=True)
```

5. Aquí se verifico los datos, si desea probarlo, borre '#' y ejecute la celda presionando play.

También se separo en 2 variables los datos de usuarios legítimos e ilegítimos para posteriores análisis



```

# verificar datos
# print(df.shape)
# print(df.info())
# print(df['label'].value_counts())
df_label0 = df.loc[df['label'] == 0]
df_label1 = df.loc[df['label'] == 1]

```

6. En esta celda se hace una descripción de los datos(ignore unnamed, count y label). Se obtiene la media, desviación estándar, mínimo, máximos y los cuartiles. Al ejecutar la celda, debería salir una tabla igual a esta, recuerde que las celdas se ejecutando dando botón de **play**:



```

[4] # Descripción de los valores globales
df.describe()

```

	Unnamed: 0	dwell_avg	flight_avg	traj_avg	label
count	1760.00000	1760.000000	1760.000000	1760.000000	1760.000000
mean	9.50000	0.115514	0.952100	471.587845	0.500000
std	5.76792	0.026339	0.701359	212.388368	0.500142
min	0.00000	0.061586	0.188908	179.164520	0.000000
25%	4.75000	0.095207	0.555139	322.691455	0.000000
50%	9.50000	0.117266	0.749423	411.584994	0.500000
75%	14.25000	0.130498	1.115375	559.801943	1.000000
max	19.00000	0.214586	9.905352	1860.326693	1.000000

El método **.describe()** genera la descripción de la data

7. Aquí se hace una descripción de cada data separado por usuario legítimos e ilegítimos, se obtiene 2 tablas

```

▶ # Descripción de los valores según label: 1->usuario legítimo, 0->usuario falso
# description = df.groupby('label')[['dwell_avg', 'flight_avg', 'traj_avg']].describe()
# Mostrar resultados en dos tablas claras
print("=== Análisis descriptivo para LABEL 0 (usuarios FALSOS) ===")
display(df_label0.describe())
print("\n=== Análisis descriptivo para LABEL 1 (usuarios LEGÍTIMOS) ===")
display(df_label1.describe())

```

8. Aquí se obtiene 3 histogramas(histplot) que están separados por clase para comparaciones entre usuarios legítimos e ilegítimos.

En los bucles **for**:

- Este bucle recorre una lista de nombres de columnas del DataFrame df.
- En cada vuelta, la variable col toma uno de esos nombres.
- Esto permite repetir el mismo gráfico para cada variable sin tener que copiar el código tres veces.

En plt.figure(): Genera la figura pero aun vacía

En sns.histplot(): dibuja el histograma con ciertos parámetros;

**data=df**: Indica que los datos provienen del DataFrame df.

**x=col**: El nombre de la columna actual del bucle ('dwell\_avg', 'flight\_avg' o 'traj\_avg'). Esa será la variable que se graficará en el eje horizontal.

**hue='label'**: Seaborn usará la columna 'label' para separar y colorear los datos según su categoría (por ejemplo: “legítimo” vs “impostor”).

**kde=True**: Además del histograma, dibuja la curva de densidad suavizada (Kernel Density Estimate).

**stat='density'**: En lugar de mostrar el conteo de datos por barra, muestra densidad de probabilidad (área total = 1). Esto facilita la comparación entre grupos con diferente cantidad de datos.

**element='step'**: El histograma se dibuja con **contornos** tipo escalón, no con barras rellenas. Esto hace más fácil ver varias distribuciones superpuestas.

**common\_norm=False**

Por defecto, Seaborn normaliza todas las series juntas para que la suma total sea 1.

Con False, normaliza cada clase (hue) de forma independiente, de modo que puedas comparar la forma de las distribuciones sin que una clase con más datos “aplane” la otra.

```
# Distribuciones separadas por clase
for col in ['dwell_avg', 'flight_avg', 'traj_avg']:
    plt.figure()
    sns.histplot(data=df, x=col, hue='label', kde=True, stat='density', element='step', common_norm=False)
    plt.title(f'Distribución de {col} por tipo de usuario')
    plt.show()
```

9. Se obtiene otros 3 histogramas pero esta vez por variable(no es de mayor relevancia)

```
# Histogramas por variable
for col in ['dwell_avg', 'flight_avg', 'traj_avg']:
    plt.figure()
    sns.histplot(df[col], kde=True)
    plt.title(f'Histograma de {col}')
    plt.show()
```

- ☐ **for col in [...]:** recorre las tres columnas y repite el gráfico para cada una.
- ☐ **plt.figure():** crea una nueva figura vacía para no sobrescribir gráficos anteriores.
- ☐ **sns.histplot(df[col], kde=True):**

Grafica el histograma de la columna.

kde=True añade una curva de densidad suavizada sobre el histograma.

- ☐ **plt.title(...):** pone un título con el nombre de la variable.
- ☐ **plt.show():** muestra el gráfico en pantalla.

10. Boxplot, Diagrama de cajas utilizado para visualizar la distribución de los datos, identificar su tendencia central y detectar posibles valores atípicos (*outliers*). Permite observar la concentración de los datos en torno a la mediana y la dispersión de los mismos.

```
# Boxplots
for col in ['dwell_avg', 'flight_avg', 'traj_avg']:
    plt.figure()
    sns.boxplot(x='label', y=col, data=df)
    plt.title(f'Boxplot de {col} por clase')
    plt.show()
```

11. Matriz de correlación, muestra como las variables están correlacionadas. Este grafico nos sugiere que no hay una relación directa completamente entre las variables ya que miden comportamientos distintos, nos da a entender que esto es perfecto para un modelo de machine learning como el SVM; porque significa que cada variable podría aportar **información única** al modelo.

```
# Correlacion
corr = df[['dwell_avg', 'flight_avg', 'traj_avg']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Matriz de correlación')
plt.show()
```

12. Test de U de Mann-Whitney : Prueba estadística no paramétrica, esto sirve para comparar las 2 datas de usuarios legítimos e ilegítimos, para ver si hay una diferencia significativa entre sus distribuciones. Se analiza cada variable.

```
# test U de Mann-whitney
for col in ['dwell_avg', 'flight_avg', 'traj_avg']:
    u_stat, p_val = mannwhitneyu(df[df['label']==0][col], df[df['label']==1][col])
    print(f"{col}: U={u_stat:.2f}, p={p_val:.4f}")
    print("-----")
    for var in ['dwell_avg', 'flight_avg', 'traj_avg']:
        group0 = df[df['label'] == 0][var]
        group1 = df[df['label'] == 1][var]

        stat, p = mannwhitneyu(group0, group1, alternative='two-sided')
        print(f"{var} → U-statistic = {stat:.2f}, p-value = {p:.5f}")
```

- **Recorre cada variable** (dwell\_avg, flight\_avg, traj\_avg).
- **Separa los datos en dos grupos** según la columna label (0 y 1, por ejemplo, usuarios legítimos vs impostores).
- **mannwhitneyu(...):**

Aplica la **prueba U de Mann–Whitney**, que compara si dos grupos provienen de distribuciones diferentes (prueba no paramétrica).


Devuelve:

U o *U-statistic*: medida de la diferencia en rangos entre grupos.

p: valor p para evaluar significancia estadística.

- **alternative='two-sided'**: prueba bilateral (busca diferencias en cualquier dirección).
- **print(...)**: muestra el nombre de la variable, el estadístico U y el valor p con formato reducido de decimales.

13. Pair plot: Ayuda a entender las relaciones entre las variables de los datos mostrando sus interacciones.

```
 sns.pairplot(df, hue='label', vars=['dwell_avg', 'flight_avg', 'traj_avg'])
```

**sns.pairplot(...)**: Genera una matriz de gráficos de dispersión (*scatter plots*) para explorar las relaciones bivariadas entre las variables seleccionadas (dwell\_avg, flight\_avg, traj\_avg). El parámetro `hue='label'` colorea los puntos según la clase (label), facilitando la identificación de patrones o separaciones entre grupos. También incluye histogramas o KDE en la diagonal para mostrar la distribución univariada de cada variable.



14. Asimetría y curtosis: genera una tabla con los datos(explicado mas detalladamente en el otro documento)

```
# Skewness (Asimetría)
# Kurtosis (Curtosis)
vars_bio = ['dwell_avg', 'flight_avg', 'traj_avg']

# Crear tabla de resultados
resultados = []
for var in vars_bio:
    skew_real = skew(df_label1[var])
    kurt_real = kurtosis(df_label1[var], fisher=False) # fisher=False → compara contra la normal
    skew_fake = skew(df_label0[var])
    kurt_fake = kurtosis(df_label0[var], fisher=False)

    resultados.append({
        'Variable': var,
        'Skew_real': skew_real,
        'Kurtosis_real': kurt_real,
        'Skew_fake': skew_fake,
        'Kurtosis_fake': kurt_fake
    })
resultadosSkKU = pd.DataFrame(resultados)
resultadosSkKU
```

Este código evalúa la forma de la distribución de cada variable biométrica (dwell\_avg, flight\_avg, traj\_avg) de manera separada para usuarios legítimos (df\_label1) y no legítimos (df\_label0).

- **skew():** mide la asimetría de la distribución (cola hacia la derecha o izquierda).
- **kurtosis(fisher=False):** mide la curtosis en su definición de Pearson (normal = 3), indicando el grado de concentración y el peso de las colas.

Los resultados se almacenan en una lista de diccionarios y luego se convierten en un DataFrame (resultadosSkKU) para facilitar su visualización y análisis comparativo entre ambos grupos.

15. Correlación de spearman:

```
print("Correlación de Spearman con la etiqueta (label):")
for var in ['dwell_avg', 'flight_avg', 'traj_avg']:
    rho, p_val = spearmanr(df[var], df['label'])
    print(f"{var}: ρ = {rho:.4f}, p = {p_val:.5f}")

print("\n" + "-"*50)

# --- 2. Matriz de correlaciones de Spearman entre todas las variables ---
corr_matrix = df[['dwell_avg', 'flight_avg', 'traj_avg'] + ['label']].corr(method='spearman')
print("Matriz de correlaciones de Spearman:")
print(corr_matrix)
```

Toda la data se analizó con Python en Google colab por la ventaja que no es necesario instalar nada en el computador para que los scripts funcionen.