Proyecto de Consultoría – Clustering (Lab 12)

- Sofia Mishell 22049
- Jose Marchena 22398

0) Inicialización de Spark

In [1]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Proyecto Consultoria Clustering - EDA") \
    .getOrCreate()

spark
```

Out[1]:

SparkSession - in-memory

SparkContext

Spark UI

Version	`v4.0.1`
Master	`local[*]`
AppName	`Proyecto Consultoria Clustering - EDA`

1) Carga del Dataset

Cargamos el archivo `hack_data.csv`, que contiene información de las sesiones registradas durante los ataques.
El parámetro `inferSchema=True` permite que Spark determine automáticamente los tipos de datos.

In [2]:
```
hack_df = spark.read.csv("data/hack_data.csv", header=True, inferSchema=True)

print(" Muestra de los datos:")
hack_df.show(5)
```

```
 Muestra de los datos:
+----------------------+-----------------+--------------+-----------------+-------
--------+--------------------+----------------+
|Session_Connection_Time|Bytes Transferred|Kali_Trace_Used|Servers_Corrupted|Pages_C
orrupted|            Location|WPM_Typing_Speed|
+----------------------+-----------------+--------------+-----------------+-------
--------+--------------------+----------------+
|                   8.0|           391.09|             1|             2.96|
7.0|            Slovenia|           72.37|
|                  20.0|           720.99|             0|             3.04|
9.0|British Virgin Is...|           69.08|
|                  31.0|           356.32|             1|             3.71|
8.0|             Tokelau|           70.58|
|                   2.0|           228.08|             1|             2.48|
8.0|             Bolivia|            70.8|
|                  20.0|            408.5|             0|             3.57|
8.0|                Iraq|           71.28|
+----------------------+-----------------+--------------+-----------------+-------
--------+--------------------+----------------+
only showing top 5 rows
```

2) Análisis Exploratorio

Inspección General del dataset

In [3]:

```python
print("Esquema del dataset:")
hack_df.printSchema()

print("\nNúmero de filas y columnas:")
print(f"Filas: {hack_df.count()}, Columnas: {len(hack_df.columns)}")

print("\nResumen estadístico:")
hack_df.describe().show()

# Verificar valores nulos
from pyspark.sql.functions import col, sum as _sum
hack_df.select([_sum(col(c).isNull().cast("int")).alias(c) for c in hack_df.columns
```

```
Esquema del dataset:
root
 |-- Session_Connection_Time: double (nullable = true)
 |-- Bytes Transferred: double (nullable = true)
 |-- Kali_Trace_Used: integer (nullable = true)
 |-- Servers_Corrupted: double (nullable = true)
 |-- Pages_Corrupted: double (nullable = true)
 |-- Location: string (nullable = true)
 |-- WPM_Typing_Speed: double (nullable = true)


Número de filas y columnas:
Filas: 334, Columnas: 7

Resumen estadístico:
+-------+----------------------+-----------------+------------------+-------------
----+------------------+-----------+------------------+
|summary|Session_Connection_Time| Bytes Transferred|   Kali_Trace_Used|Servers_Corru
pted|   Pages_Corrupted|   Location|  WPM_Typing_Speed|
+-------+----------------------+-----------------+------------------+-------------
----+------------------+-----------+------------------+
|  count|                   334|              334|               334|
334|               334|        334|               334|
|   mean|    30.008982035928145| 607.2452694610777|0.5119760479041916|5.25850299401
1977|10.838323353293413|       NULL|57.342395209580864|
| stddev|    14.088200614636158|286.33593163576757|0.5006065264451406| 2.3019069333
9697|  3.06352633036022|       NULL| 13.41106336843464|
|    min|                   1.0|             10.0|                 0|
1.0|               6.0|Afghanistan|              40.0|
|    max|                  60.0|           1330.5|                 1|
10.0|              15.0|   Zimbabwe|              75.0|
+-------+----------------------+-----------------+------------------+-------------
----+------------------+-----------+------------------+

+----------------------+-----------------+--------------+-----------------+-------
--------+--------+---------------+
|Session_Connection_Time|Bytes Transferred|Kali_Trace_Used|Servers_Corrupted|Pages_C
orrupted|Location|WPM_Typing_Speed|
+----------------------+-----------------+--------------+-----------------+-------
--------+--------+---------------+
|                     0|                0|             0|                0|
0|       0|              0|
+----------------------+-----------------+--------------+-----------------+-------
--------+--------+---------------+
```

Resultados

- **Filas:** 334
- **Columnas:** 7
- No hay valores nulos.

Tipos de datos:

- `Session_Connection_Time`, `Bytes Transferred`, `Servers_Corrupted`, `Pages_Corrupted`, `WPM_Typing_Speed`: numéricos
- `Kali_Trace_Used`: binario (0 = no, 1 = sí)
- `Location`: categórica (texto)

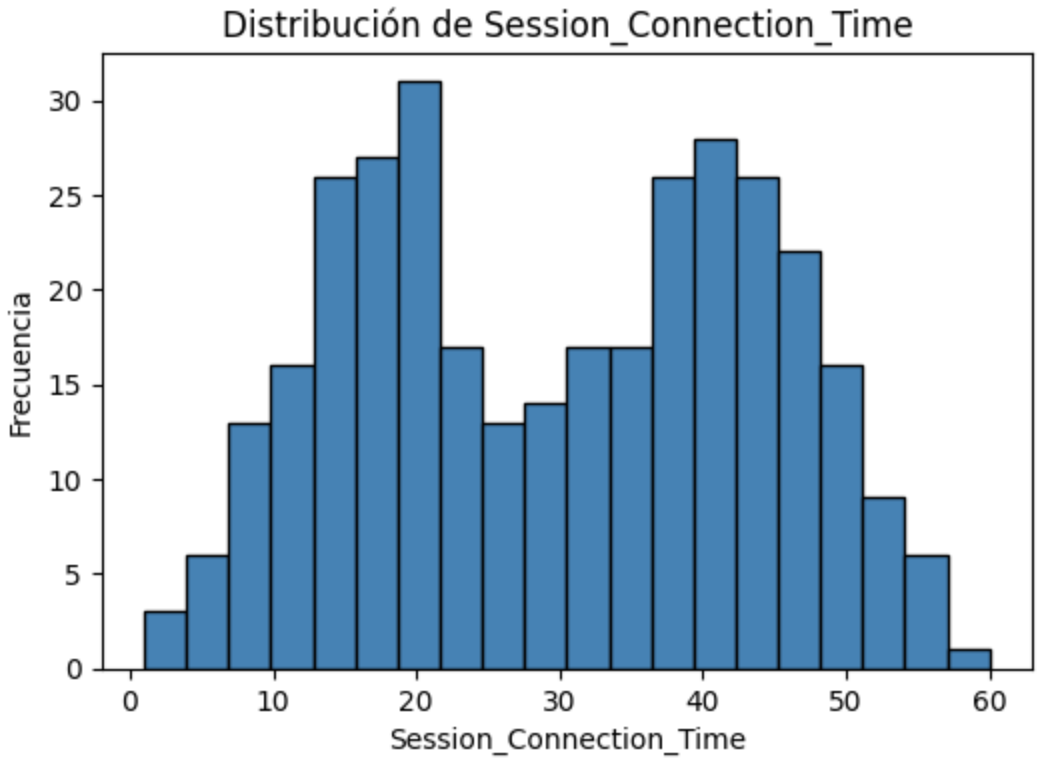
Estadísticas principales:

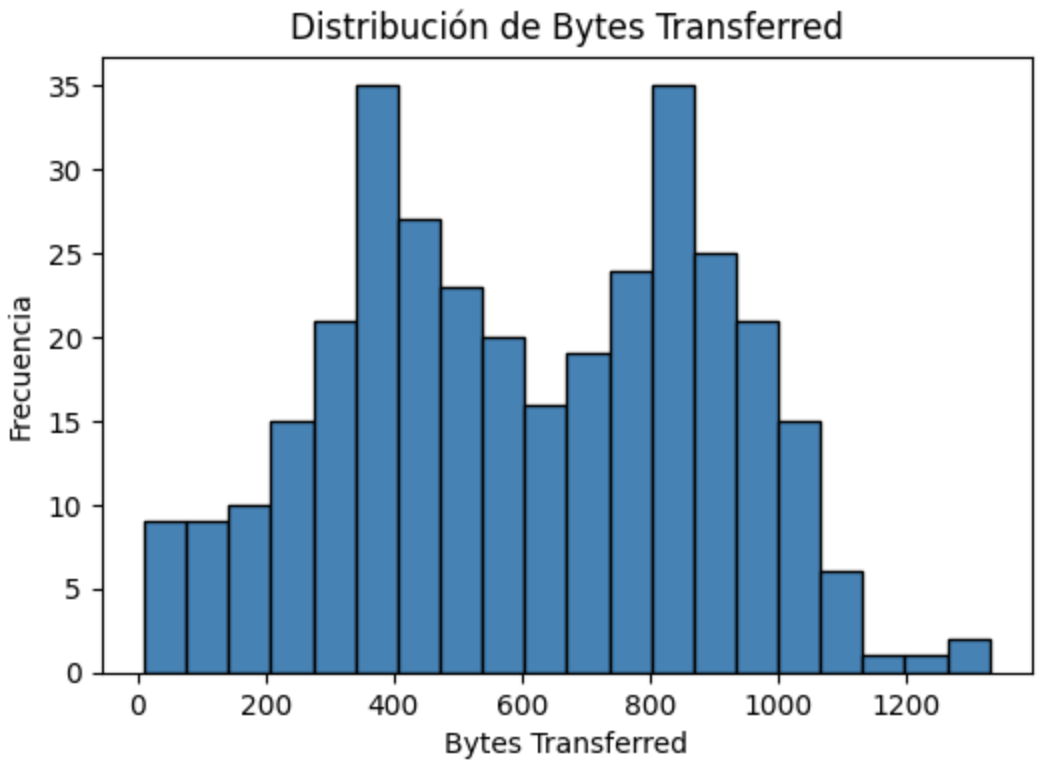
Variable	Media	Desv. Est.	Mín	Máx
Session_Connection_Time	30.0	14.09	1	60
Bytes_Transferred	607.2	286.3	10	1330
Servers_Corrupted	5.26	2.30	1	10
Pages_Corrupted	10.83	3.06	6	15
WPM_Typing_Speed	57.34	13.41	40	75

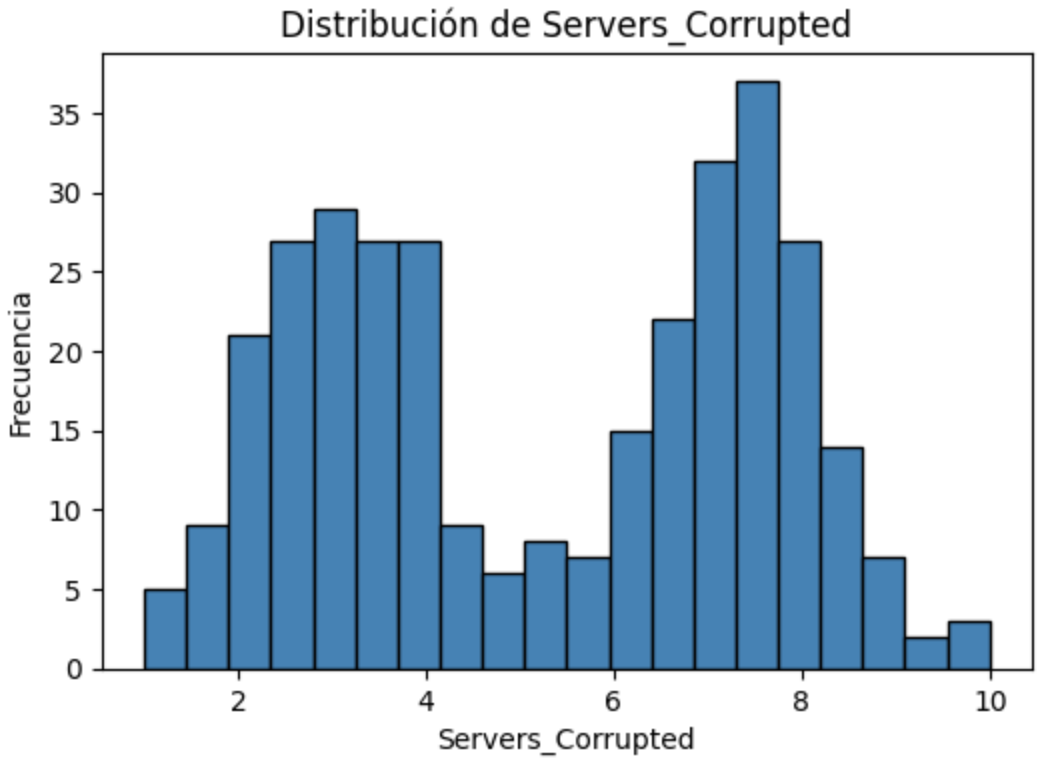
Los datos se distribuyen en rangos amplios, lo que sugiere comportamientos diversos en las sesiones.

Distribuciones númericas

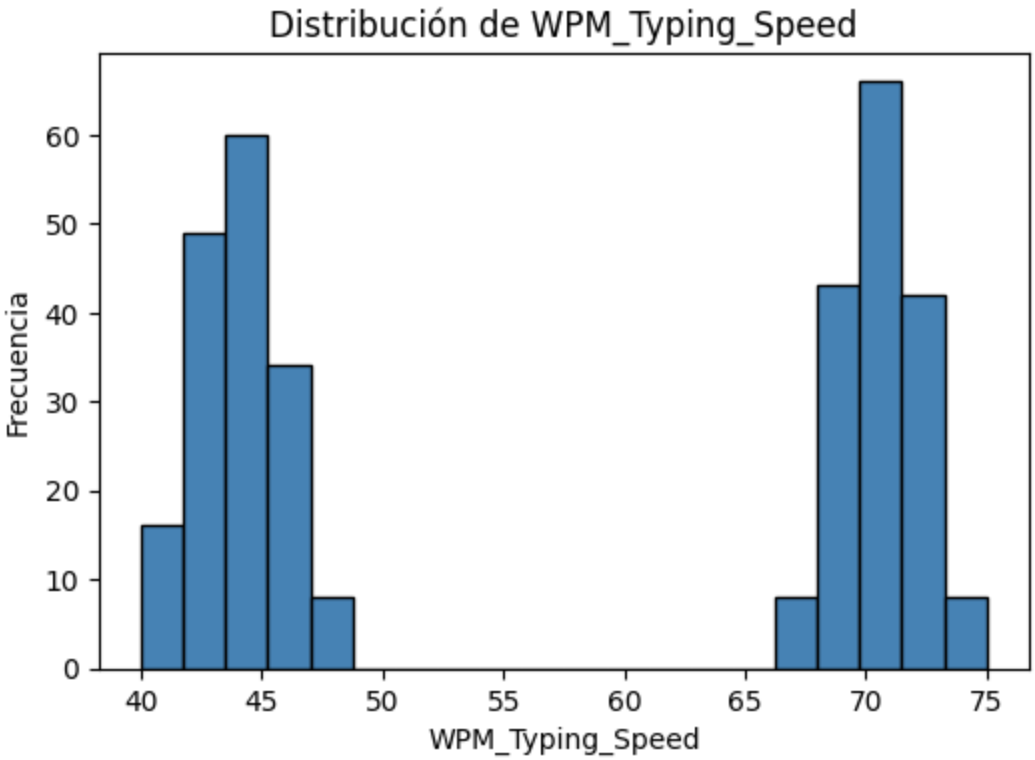
In [4]:
```
import matplotlib.pyplot as plt
import pandas as pd

hack_pd = hack_df.toPandas()
num_cols = ['Session_Connection_Time','Bytes Transferred','Servers_Corrupted','Page

for col_name in num_cols:
    plt.figure(figsize=(6,4))
    plt.hist(hack_pd[col_name], bins=20, color='steelblue', edgecolor='black')
    plt.title(f'Distribución de {col_name}')
    plt.xlabel(col_name)
    plt.ylabel('Frecuencia')
    plt.show()
```


Distribución de Session_Connection_Time
Frecuencia
Session_Connection_Time
0
5
10
15
20
25
30
0
10
20
30
40
50
60


Distribución de Bytes Transferred
Frecuencia
Bytes Transferred
0
5
10
15
20
25
30
35
0
200
400
600
800
1000
1200


Distribución de Servers_Corrupted
Frecuencia
Servers_Corrupted
0
5
10
15
20
25
30
35
2
4
6
8
10


Distribución de Pages_Corrupted
Frecuencia
Pages_Corrupted
0
20
40
60
80
100
6
8
10
12
14



Interpretación

Session_Connection_Time:
Muestra dos concentraciones principales, una entre 10–25 minutos y otra entre 35–50 minutos. Esto sugiere **dos grupos distintos de comportamiento** (posibles hackers con diferentes hábitos de conexión).

Bytes_Transferred:
Presenta también dos picos, uno alrededor de 400 MB y otro cerca de 800 MB, reforzando la posibilidad de múltiples grupos de ataque con distinta intensidad de transferencia.

Servers_Corrupted:
Tiene dos acumulaciones en torno a 3 y 7 servidores comprometidos. Esto indica dos niveles de agresividad en los ataques.

Pages_Corrupted:
Los valores se agrupan fuertemente alrededor de 8 y 14 páginas afectadas, lo que sugiere que los atacantes tienden a causar un número específico de daños, probablemente por patrón de ejecución.

WPM_Typing_Speed:
Muestra dos grupos bien definidos: uno entre 42–48 palabras por minuto y otro entre 68–72. Este indicador es muy relevante, pues **podría representar dos hackers distintos** según su velocidad de tecleo característica.

Análisis de Ubicaciones

In [5]:
```
loc_count = hack_df.groupBy("Location").count().orderBy("count", ascending=False)
loc_count.show(10)
```

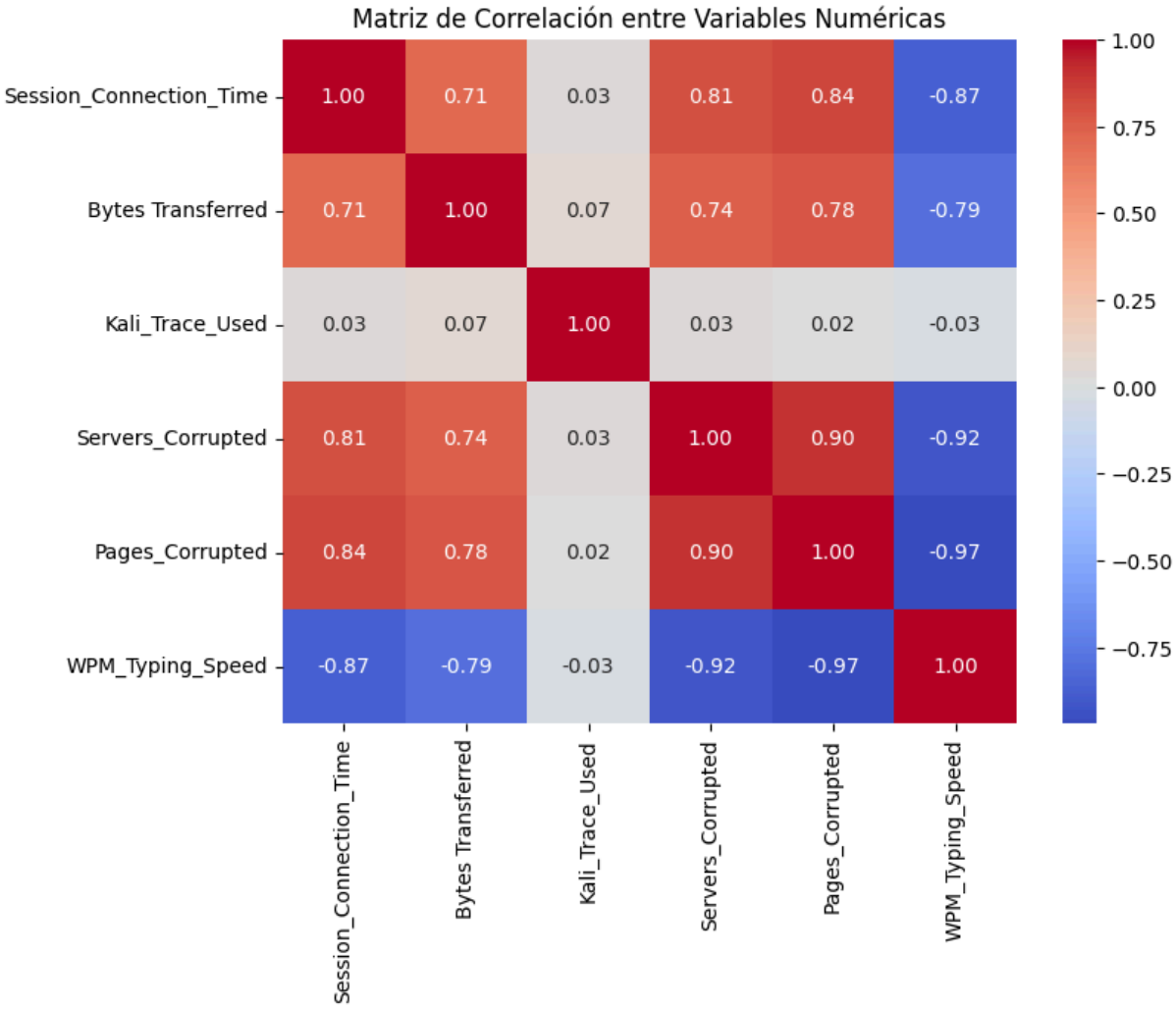
```
+--------------------+-----+
|            Location|count|
+--------------------+-----+
|United States Vir...|    6|
|          Mauritania|    5|
|      Czech Republic|    5|
|       Guinea-Bissau|    5|
|             Reunion|    4|
|             Ukraine|    4|
|           Sri Lanka|    4|
|              Tuvalu|    4|
|Palestinian Terri...|    4|
|    Saint Barthelemy|    4|
+--------------------+-----+
only showing top 10 rows
```

Los ataques provienen de países muy diversos y con baja repetición, lo que confirma que los hackers utilizaron **VPNs o IPs falsas** para ocultar su origen.
Por ello, `Location` **no es una variable confiable** para el clustering.

Matriz de Correlación

Analizamos la relación entre las variables numéricas para entender qué métricas se asocian entre sí.

In [6]:
```
import seaborn as sns

corr = hack_pd.corr(numeric_only=True)
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Matriz de Correlación entre Variables Numéricas")
plt.show()
```



Interpretación de la Matriz de Correlación

- **Servers_Corrupted** y **Pages_Corrupted** presentan una **correlación positiva muy alta (0.90)**, lo que indica que cuando se afectan más servidores también se corrompen más páginas.
- **Session_Connection_Time**, **Bytes_Transferred** y las variables de daño muestran **correlaciones positivas fuertes**, reflejando que las sesiones más largas y con mayor transferencia generan más afectaciones.
- **WPM_Typing_Speed** tiene **correlaciones negativas altas** con el resto, especialmente con **Pages_Corrupted (-0.97)** y **Servers_Corrupted (-0.92)**, lo que sugiere **dos tipos de hackers**:
  - unos más **lentos y destructivos**,
  - y otros **rápidos pero menos dañinos**.
- **Kali_Trace_Used** no guarda relación significativa con las demás variables.

3) Limpieza Preliminar

Eliminamos la variable `Location`, pues no aporta valor analítico confiable, y mantenemos solo las variables relevantes para el clustering.

In [7]:
```
hack_clean_df = hack_df.drop("Location")
hack_clean_df.show(5)
```

```
+----------------------+----------------+--------------+----------------+-------
--------+---------------+
|Session_Connection_Time|Bytes Transferred|Kali_Trace_Used|Servers_Corrupted|Pages_C
orrupted|WPM_Typing_Speed|
+----------------------+----------------+--------------+----------------+-------
--------+---------------+
|                   8.0|          391.09|             1|            2.96|
7.0|          72.37|
|                  20.0|          720.99|             0|            3.04|
9.0|          69.08|
|                  31.0|          356.32|             1|            3.71|
8.0|          70.58|
|                   2.0|          228.08|             1|            2.48|
8.0|           70.8|
|                  20.0|           408.5|             0|            3.57|
8.0|          71.28|
+----------------------+----------------+--------------+----------------+-------
--------+---------------+
only showing top 5 rows
```

4) Preparación de Datos y Clustering (K-Means con Spark)

Aplicar **K-Means** para identificar grupos naturales en los datos. El objetivo es determinar si los ataques se agrupan en **2 o 3 clusters**, lo cual ayudaría a inferir si hubo **dos o tres hackers** distintos.

In [8]:
```
from pyspark.ml.feature import VectorAssembler, StandardScaler

# Ensamblamos las columnas numéricas relevantes en un solo vector
feature_cols = ['Session_Connection_Time', 'Bytes Transferred', 'Kali_Trace_Used',
                'Servers_Corrupted', 'Pages_Corrupted', 'WPM_Typing_Speed']

assembler = VectorAssembler(inputCols=feature_cols, outputCol='features_unscaled')
assembled_df = assembler.transform(hack_clean_df)

# Escalamos los datos para que todas las variables contribuyan por igual
scaler = StandardScaler(inputCol='features_unscaled', outputCol='features', withMea
scaled_df = scaler.fit(assembled_df).transform(assembled_df)

scaled_df.select('features').show(5, truncate=False)
```

```
+--------------------------------------------------------------------------------
-------------------------------------+
|features
|
+--------------------------------------------------------------------------------
-------------------------------------+
|[-1.562228040184432,-0.75490095925522,0.9748653409721156,-0.9985212523861794,-1.252
9101889070722,1.1205379005060365]    |
|[-0.7104514131868532,0.3972422527942127,-1.0227114926762684,-0.9637674581126887,-0.
6000677503813903,0.8752180545239784] |
|[0.07034382822759382,-0.8763317548992295,0.9748653409721156,-0.6727044310722048,-0.
9264889696442312,0.9870660086191414] |
|[-1.9881163536832211,-1.3241973066216277,0.9748653409721156,-1.2070440180271231,-0.
9264889696442312,1.0034703752197651] |
|[-0.7104514131868532,-0.6940982513989585,-1.0227114926762684,-0.7335235710508134,-
0.9264889696442312,1.0392617205302177]|
+--------------------------------------------------------------------------------
-------------------------------------+
only showing top 5 rows
```

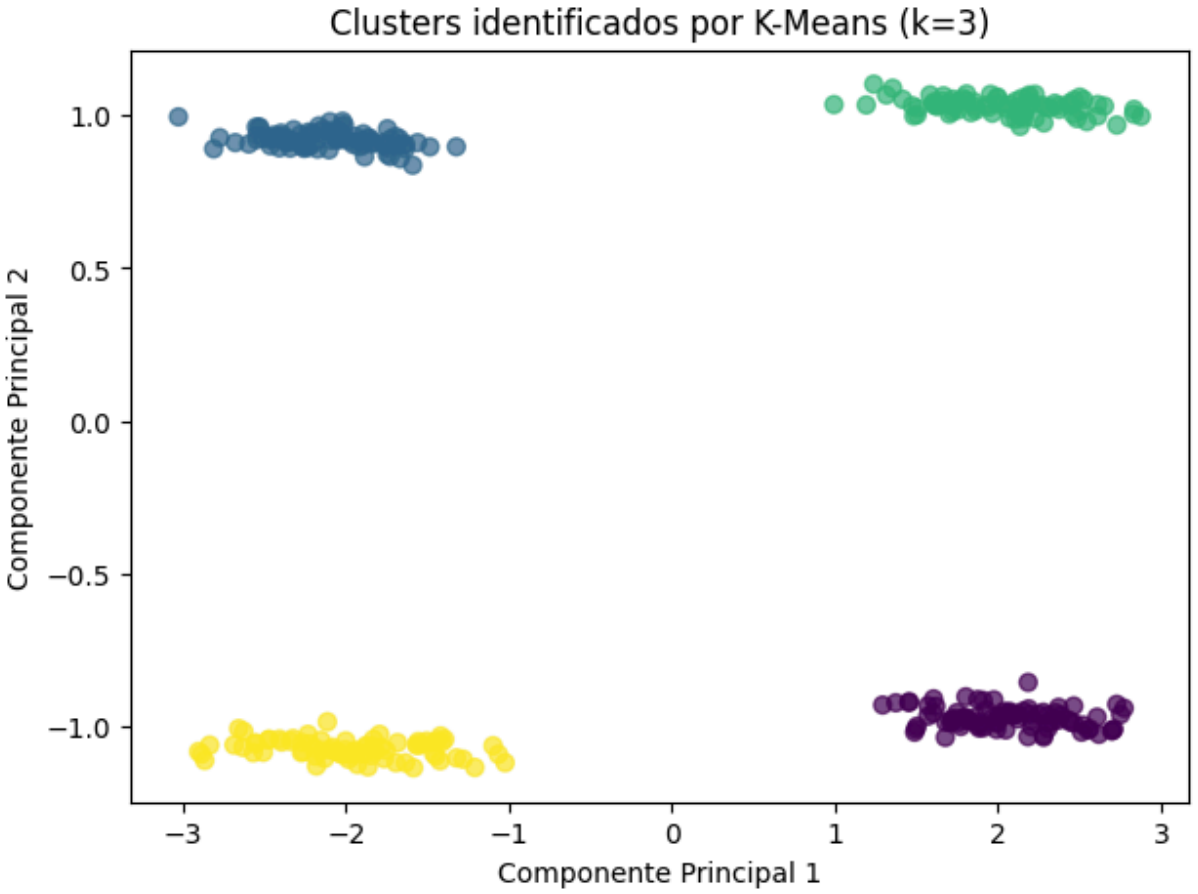
5) Aplicación del Modelo K-Means

In [16]:

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

# Modelo con 2 clusters
kmeans_2 = KMeans(featuresCol='features', k=2, seed=67)
model_2 = kmeans_2.fit(scaled_df)
pred_2 = model_2.transform(scaled_df)

# Modelo con 3 clusters
kmeans_3 = KMeans(featuresCol='features', k=3, seed=67)
model_3 = kmeans_3.fit(scaled_df)
pred_3 = model_3.transform(scaled_df)

# Modelo con 3 clusters
kmeans_4 = KMeans(featuresCol='features', k=4, seed=67)
model_4 = kmeans_4.fit(scaled_df)
pred_4 = model_4.transform(scaled_df)

# Evaluación con Silhouette Score
evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='features',

score_2 = evaluator.evaluate(pred_2)
score_3 = evaluator.evaluate(pred_3)
score_4 = evaluator.evaluate(pred_4)


print(f"Silhouette Score (k=2): {score_2:.4f}")
print(f"Silhouette Score (k=3): {score_3:.4f}")
print(f"Silhouette Score (k=4): {score_4:.4f}")
```

```
Silhouette Score (k=2): 0.8176
Silhouette Score (k=3): 0.7608
Silhouette Score (k=4): 0.7196
```

In [39]:
```
from pyspark.ml.feature import PCA
import matplotlib.pyplot as plt

# Aplicamos PCA para reducir de 6D a 2D
pca = PCA(k=2, inputCol='features', outputCol='pca_features')
pca_model = pca.fit(scaled_df)
pca_df = pca_model.transform(pred_4)  # graficaremos el modelo con k=3

# Convertimos a Pandas para graficar
pca_pd = pca_df.select('pca_features', 'prediction').toPandas()
pca_pd[['x','y']] = pd.DataFrame(pca_pd['pca_features'].tolist(), index=pca_pd.inde

plt.figure(figsize=(7,5))
plt.scatter(pca_pd['x'], pca_pd['y'], c=pca_pd['prediction'], cmap='viridis', s=40,
plt.title('Clusters identificados por K-Means (k=3)')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.show()
```



In [21]:
```
def pca_vis(pred, title):
    pca = PCA(k=2, inputCol='features', outputCol='pca_features')
    pca_model = pca.fit(scaled_df)
    pca_df = pca_model.transform(pred)  # graficaremos el modelo con k=3

    # Convertimos a Pandas para graficar
    pca_pd = pca_df.select('pca_features', 'prediction').toPandas()
    pca_pd[['x','y']] = pd.DataFrame(pca_pd['pca_features'].tolist(), index=pca_pd.
```

```
    plt.figure(figsize=(7,5))
    plt.scatter(pca_pd['x'], pca_pd['y'], c=pca_pd['prediction'], cmap='viridis', s
    plt.title(f'{title} (k=3)')
    plt.xlabel('Componente Principal 1')
    plt.ylabel('Componente Principal 2')
    plt.show()
```

In [38]:
```
from pyspark.ml.feature import PCA
import numpy as np

# Fit PCA
pca = PCA(k=6, inputCol="features", outputCol="pcaFeatures")
pca_model = pca.fit(scaled_df)

# Get the principal components (loadings)
# This tells you HOW each original feature contributes to each PC
pc_matrix = pca_model.pc.toArray()  # Shape: (num_features, k)

print("Shape:", pc_matrix.shape)  # e.g., (25, 10) for 25 features, 10 PCs

# To see which features matter most for PC1:
pc1_loadings = pc_matrix[:, 0]
feature_names = [
    "Session_Connection_Time",
    "Bytes_Transferred",
    "Servers_Corrupted",
    "Pages_Corrupted",
    "WPM_Typing_Speed"
]  # Your feature names

# Create a mapping
for i, feature in enumerate(feature_names):
    print(f"{feature}: {pc1_loadings[i]:.3f}")
```

```
Shape: (6, 6)
Session_Connection_Time: -0.436
Bytes_Transferred: -0.414
Servers_Corrupted: -0.024
Pages_Corrupted: -0.450
WPM_Typing_Speed: -0.464
```

In [47]:
```
from pyspark.sql.functions import count, mean, stddev
def show_profile(pred, title):
    cluster_profiles = pred.groupBy('prediction').agg(
        count('*').alias('count'),
        mean('Session_Connection_Time').alias('avg_Session_Connection_Time'),
        mean('Servers_Corrupted').alias('avg_Servers_Corrupted'),
        mean('Pages_Corrupted').alias('avg_Pages_Corrupted'),
        mean('WPM_Typing_Speed').alias('avg_WPM_Typing_Speed')
    )
    print(f"{title} profiles")
    cluster_profiles.show()

show_profile(pred_2, "(K=2)")
show_profile(pred_3, "(K=3)")
show_profile(pred_4, "(K=4)")
```

```
(K=2) profiles
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+
|prediction|count|avg_Session_Connection_Time|avg_Servers_Corrupted|avg_Pages_Corrup
ted|avg_WPM_Typing_Speed|
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+
|         1|  167|         17.75449101796407|   3.1419161676646703|  7.850299401197
605|    70.6329341317365|
|         0|  167|         42.26347305389221|    7.375089820359279| 13.826347305389
222|   44.05185628742513|
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+

(K=3) profiles
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+
|prediction|count|avg_Session_Connection_Time|avg_Servers_Corrupted|avg_Pages_Corrup
ted|avg_WPM_Typing_Speed|
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+
|         1|   79|        41.379746835443036|    7.364683544303797| 13.873417721518
987|  44.311139240506314|
|         2|   88|         43.05681818181818|   7.3844318181818185| 13.784090909090
908|  43.819090909090896|
|         0|  167|         17.75449101796407|   3.1419161676646703|  7.850299401197
605|    70.6329341317365|
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+

(K=4) profiles
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+
|prediction|count|avg_Session_Connection_Time|avg_Servers_Corrupted|avg_Pages_Corrup
ted|avg_WPM_Typing_Speed|
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+
|         1|   88|         43.05681818181818|   7.3844318181818185| 13.784090909090
908|  43.819090909090896|
|         3|   79|        41.379746835443036|    7.364683544303797| 13.873417721518
987|  44.311139240506314|
|         2|   83|        17.156626506024097|    3.158192771084338|  7.819277108433
735|   70.96493975903616|
|         0|   84|        18.345238095238095|   3.1258333333333326|  7.880952380952
381|   70.30488095238096|
+----------+-----+--------------------------+--------------------+--------------------
---+--------------------+
```

Explicacion

cluster 0 parece referir a los ataques, con un mayor promedio de paginas y servidores crasheados, asimismo, parecen mantener una menor tasa de WPM lo cual se relaciona con la correlacion negativa que se observo en el analisis exploratorio