Laboratorio 12 - Proyecto de Consultoría - SparkML

- Mónica Salvatierra 22249
- Derek Arreaga 22537

Link del repositorio: https://github.com/alee2602/LAB12-DS

Importación de librerías

In [1]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import (
    col, avg, stddev, min as min_, max as max_,
    sum as _sum, when, count, isnan
)
from pyspark.ml.feature import (
    Imputer, VectorAssembler, StandardScaler
)
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.ml.feature import PCA
```

Iniciar sesión en Spark

In [2]:
```
spark = (
    SparkSession.builder
    .appName("Análisis de Hackeo")
    .getOrCreate()
)

spark
```

Out[2]:

SparkSession - in-memory

SparkContext

Spark UI

Version	`v3.5.1`
Master	`local[*]`
AppName	`Análisis de Hackeo`

Carga de datos

In [3]:
```
df = (
    spark.read.csv(
        "hack_data.csv",
        header=True,
        inferSchema=True
    )
)

df.printSchema()
df.show(5)
```

```
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)

+----------------------+-----------------+---------------+-----------------+-------
--------+--------------------+----------------+
|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
```

El dataset brindado consiste de **7** columnas con la siguiente descripción:

- `Session_Connection_Time` - **double**: Representa la duración de la sesión de conexión en minutos.
- `Bytes Transferred` - **double**: Indica la cantidad total de megabytes transferidos durante la sesión.
- `Kali_Trace_Used` - **integer**: Variable binaria (0 o 1) que señala si el atacante utilizó la distribución de seguridad Kali Linux durante el ataque. (1 si se detectó y 0 si no)
- `Servers_Corrupted` - **double**: Indica el número de servidores que fueron comprometidos durante la sesión de ataque.
- `Pages_Corrupted` - **double**: Registra la cantidad de páginas o recursos internos que fueron accedidos o modificados de manera ilegal durante el ataque.

- `Location` - **string**: Ubicación del ataque. Sin embargo, esta información puede ser poco confiable, ya que los atacantes suelen utilizar VPNs o proxies para ocultar su verdadera localización.
- `WPM_Typing_Speed` - **double**: Velocidad de typeo medido en la cantidad de palabras por minuto

Verificar la existencia de valores nulos

In [4]:
```
nulls = df.select([
    count(
        when(
            col(c).isNull() | isnan(c) | (col(c) == ""), c
        )
    ).alias(c)
    for c in df.columns
])

nulls.show()
```

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

Obtener estadísticas descriptivas

In [5]:
```
df.describe().show()
```

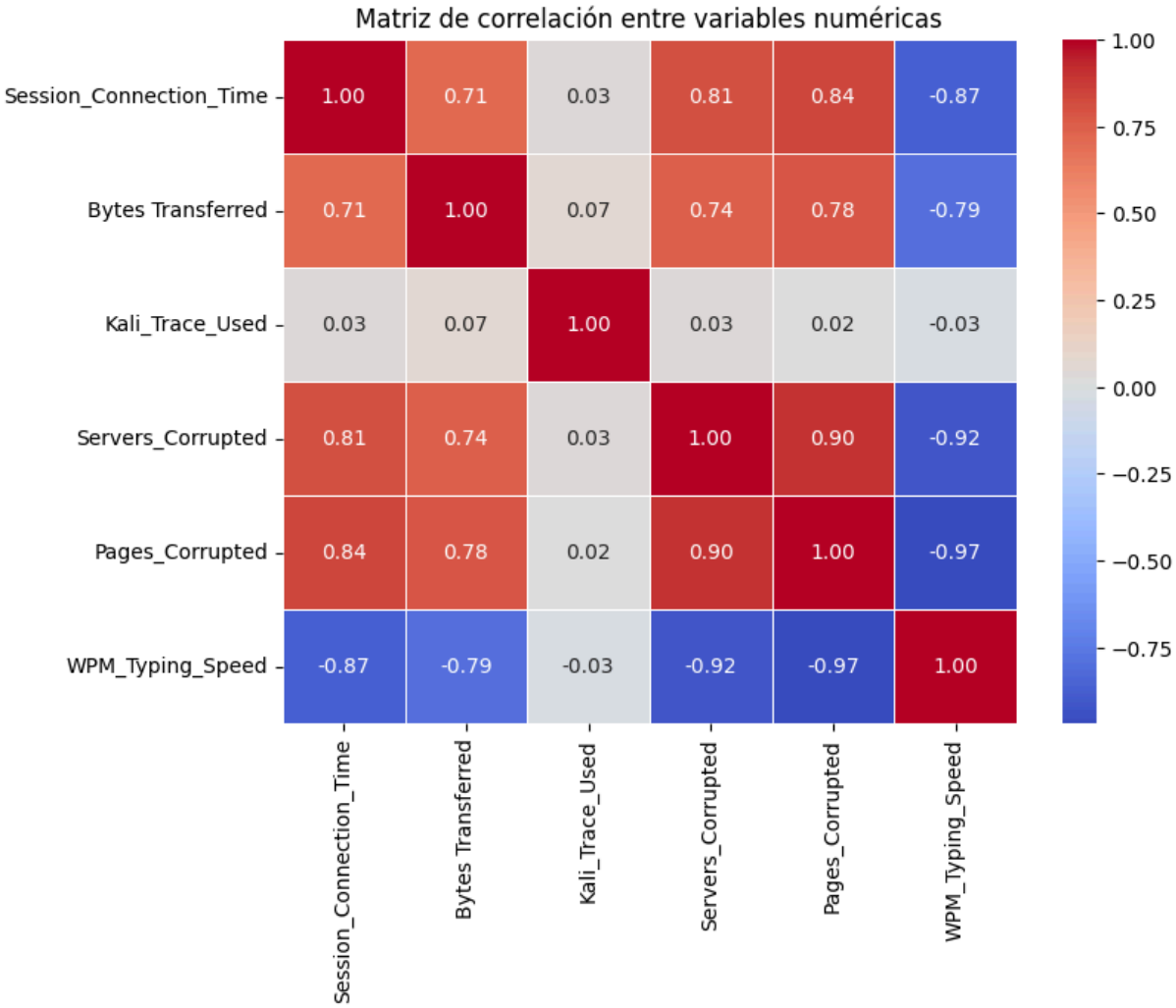
```
+-------+----------------------+-----------------+-----------------+-------------
----+-----------------+-----------+-----------------+
|summary|Session_Connection_Time| Bytes Transferred|   Kali_Trace_Used|Servers_Corru
pted|   Pages_Corrupted|   Location|  WPM_Typing_Speed|
+-------+----------------------+-----------------+-----------------+-------------
----+-----------------+-----------+-----------------+
|  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|
+-------+----------------------+-----------------+-----------------+-------------
----+-----------------+-----------+-----------------+
```

Al observar las estadísticas descriptivas del conjunto de datos, se puede notar que contiene 334 registros en total. El tiempo promedio de conexión de las sesiones es de aproximadamente 30 minutos, aunque existen sesiones muy cortas de solo un minuto y otras que alcanzan hasta una hora completa. En cuanto a la cantidad de bytes transferidos, el promedio es cercano a 607 MB, pero con una desviación estándar bastante alta, lo que indica que hubo sesiones con un volumen de transferencia muy diferente entre sí.

La mayoría de los atacantes parece haber corrompido en promedio 5 servidores y unas 10 páginas por sesión, aunque también se registran casos en los que solo se afectó un servidor o una página, mientras que otros alcanzaron valores máximos de 10 servidores y 15 páginas comprometidas.

Matriz de Correlación

In [6]:
```
num_cols = ["Session_Connection_Time", "Bytes Transferred", "Kali_Trace_Used", "Ser
pdf = df.select(num_cols).toPandas()

corr_matrix = pdf.corr()

plt.figure(figsize=(8,6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Matriz de correlación entre variables numéricas")
plt.show()
```



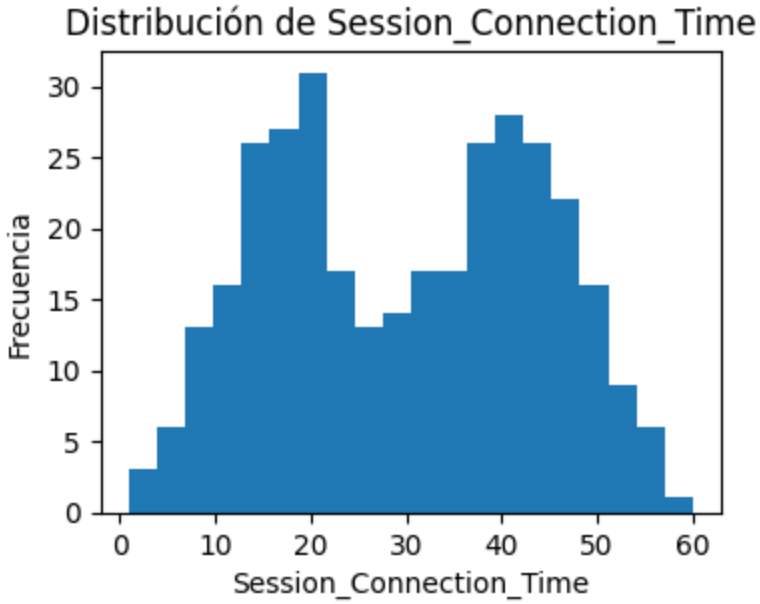
Al analizar la matriz de correlación, se observa una relación bastante fuerte entre varias de las variables numéricas. En particular, el tiempo de conexión, la cantidad de bytes transferidos, los servidores corrompidos y las páginas comprometidas presentan correlaciones positivas altas entre sí, lo que significa que cuando una de estas variables aumenta, las demás tienden a hacerlo también. Esto tiene sentido, ya que un ataque más largo suele implicar más transferencia de datos y un mayor daño a los servidores o páginas afectadas.

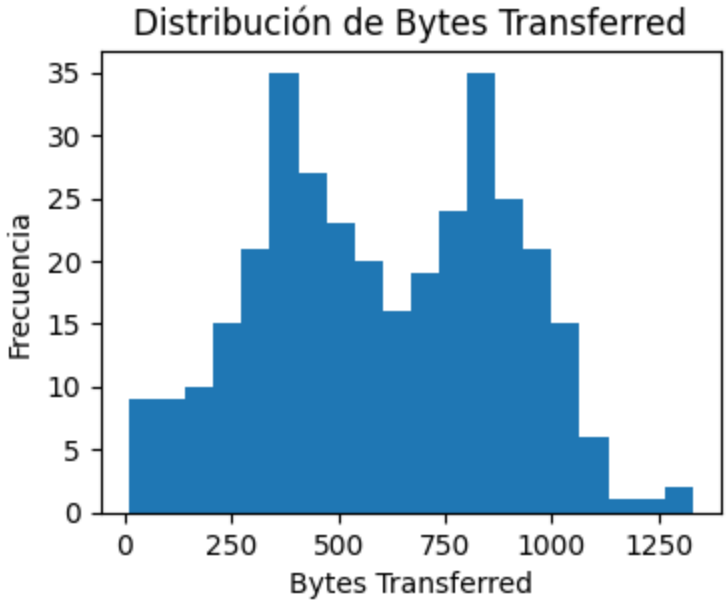
Por otro lado, la variable `WPM_Typing_Speed` muestra correlaciones negativas muy marcadas con el resto de las variables, especialmente con el tiempo de conexión y con las páginas corrompidas. Esto podría interpretarse como que los atacantes que escriben más rápido tienden a realizar ataques más breves o con menor nivel de daño, mientras que aquellos con una velocidad de tipeo más lenta suelen mantener sesiones más largas y afectar un mayor número de recursos.

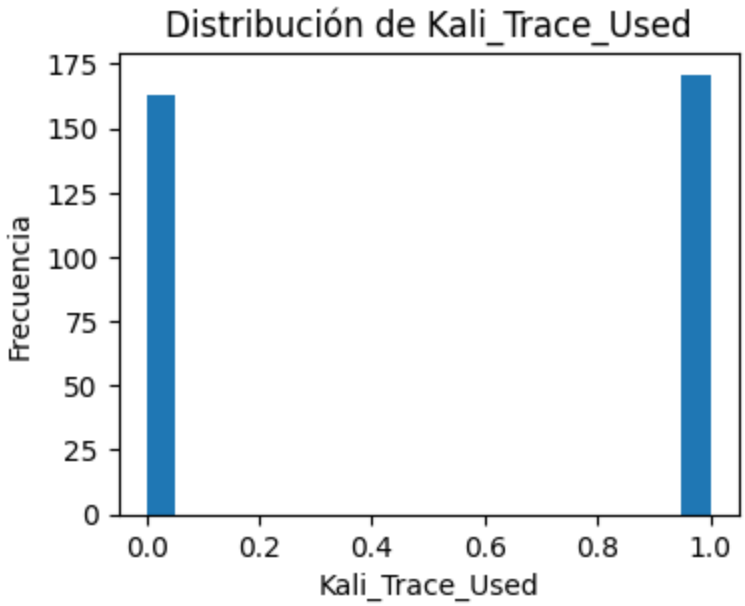
En cambio, la variable `Kali_Trace_Used` no muestra una relación clara con las demás, lo que podría sugerir que el uso de Kali Linux no está directamente asociado con el tiempo de ataque ni con la magnitud del daño causado.

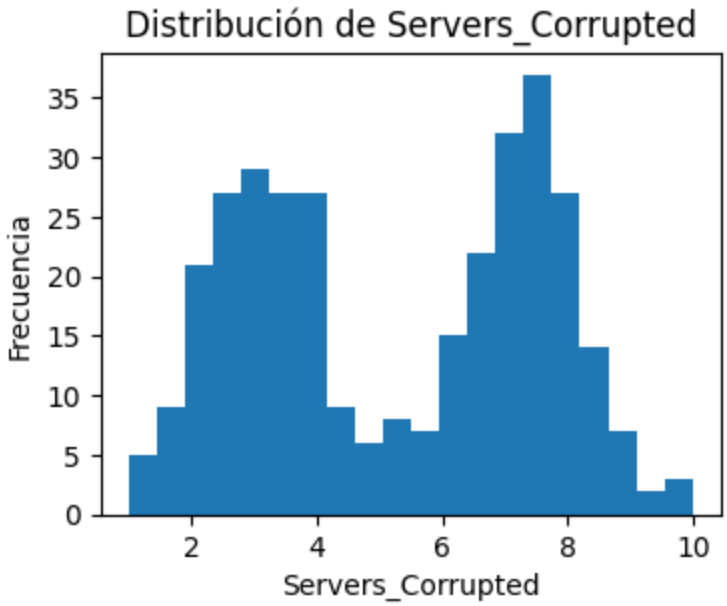
Obtener la distribución de las variables

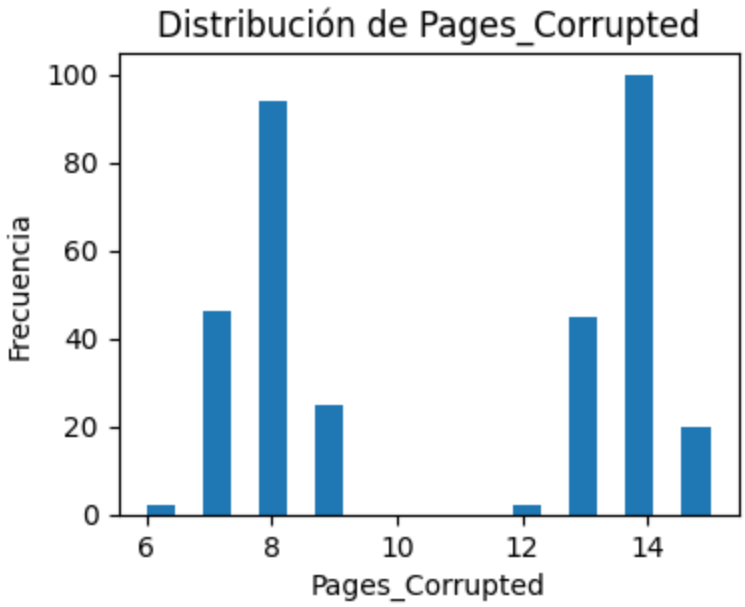
In [7]:
```
pdf = df.select(num_cols).toPandas()

import matplotlib.pyplot as plt

for col in pdf.columns:
    plt.figure(figsize=(4,3))
    plt.hist(pdf[col], bins=20)
    plt.title(f"Distribución de {col}")
    plt.xlabel(col)
    plt.ylabel("Frecuencia")
    plt.show()
```

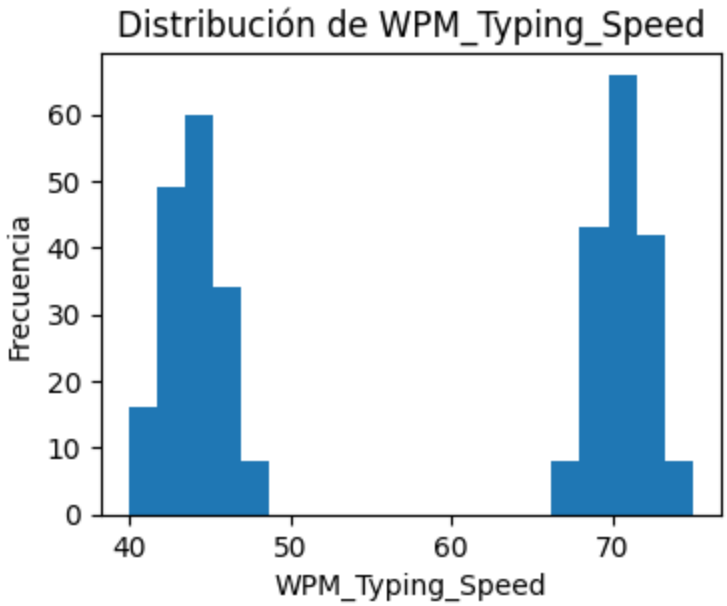





Distribución de Kali_Trace_Used
Frecuencia
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0
Kali_Trace_Used


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





Al revisar las distribuciones de las variables numéricas se pueden identificar varios patrones interesantes sobre el comportamiento de los ataques. La variable `Session_Connection_Time` muestra dos grupos principales de valores, lo que indica que hubo sesiones tanto cortas como largas. Esto demuestra la existencia de distintos tipos de atacantes o estrategias, ya que algunos mantuvieron conexiones breves mientras que otros permanecieron activos por más tiempo. De forma similar, la variable `Bytes Transferred` también presenta una distribución bimodal, con dos concentraciones bien marcadas. Esto refuerza la idea de que existen al menos dos comportamientos distintos entre las sesiones, posiblemente asociadas con diferentes niveles de intensidad o con distintos volúmenes de información intercambiada durante el ataque.

En el caso de `Kali_Trace_Used`, la distribución es completamente binaria, ya que solo toma los valores 0 y 1. Esto confirma que en unas sesiones se detectó el uso de la

herramienta Kali Linux mientras que en otras no, aunque no necesariamente esté relacionado con la gravedad del ataque.

La variable `Servers_Corrupted` también muestra una tendencia bimodal, con grupos concentrados entre 2 y 4 servidores y otro entre 6 y 8. Esto sugiere que algunos atacantes lograron comprometer más infraestructura que otros, lo cual podría ayudar a diferenciarlos en el análisis de clústeres.

Por otro lado, `Pages_Corrupted` tiene una distribución muy concentrada alrededor de ciertos valores, especialmente en 8 y 14, lo que insinúa que los atacantes afectaron cantidades similares de páginas en varios casos. Finalmente, `WPM_Typing_Speed` presenta dos picos claros, uno cercano a las 45 palabras por minuto y otro alrededor de 70, lo que refuerza la hipótesis de que existen al menos dos perfiles distintos de atacantes con estilos o niveles de habilidad diferentes al escribir comandos durante los ataques.

Armar el vector de características

In [8]:
```
feature_cols = [
    "Session_Connection_Time",
    "Bytes Transferred",
    "Kali_Trace_Used",
    "Servers_Corrupted",
    "Pages_Corrupted",
    "WPM_Typing_Speed"
]

assembler = VectorAssembler(
    inputCols=feature_cols,
    outputCol="features_noscaled"
)

df_feats = assembler.transform(df)
df_feats.select("features_noscaled").show(5, truncate=False)
```

```
+--------------------------------+
|features_noscaled               |
+--------------------------------+
|[8.0,391.09,1.0,2.96,7.0,72.37] |
|[20.0,720.99,0.0,3.04,9.0,69.08]|
|[31.0,356.32,1.0,3.71,8.0,70.58]|
|[2.0,228.08,1.0,2.48,8.0,70.8]  |
|[20.0,408.5,0.0,3.57,8.0,71.28] |
+--------------------------------+
only showing top 5 rows
```

Escalar los valores

In [9]:
```
scaler = StandardScaler(
    inputCol="features_noscaled",
    outputCol="features",
```

```
    withMean=True,
    withStd=True
)

scaler_model = scaler.fit(df_feats)
df_scaled = scaler_model.transform(df_feats)

df_scaled.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
```

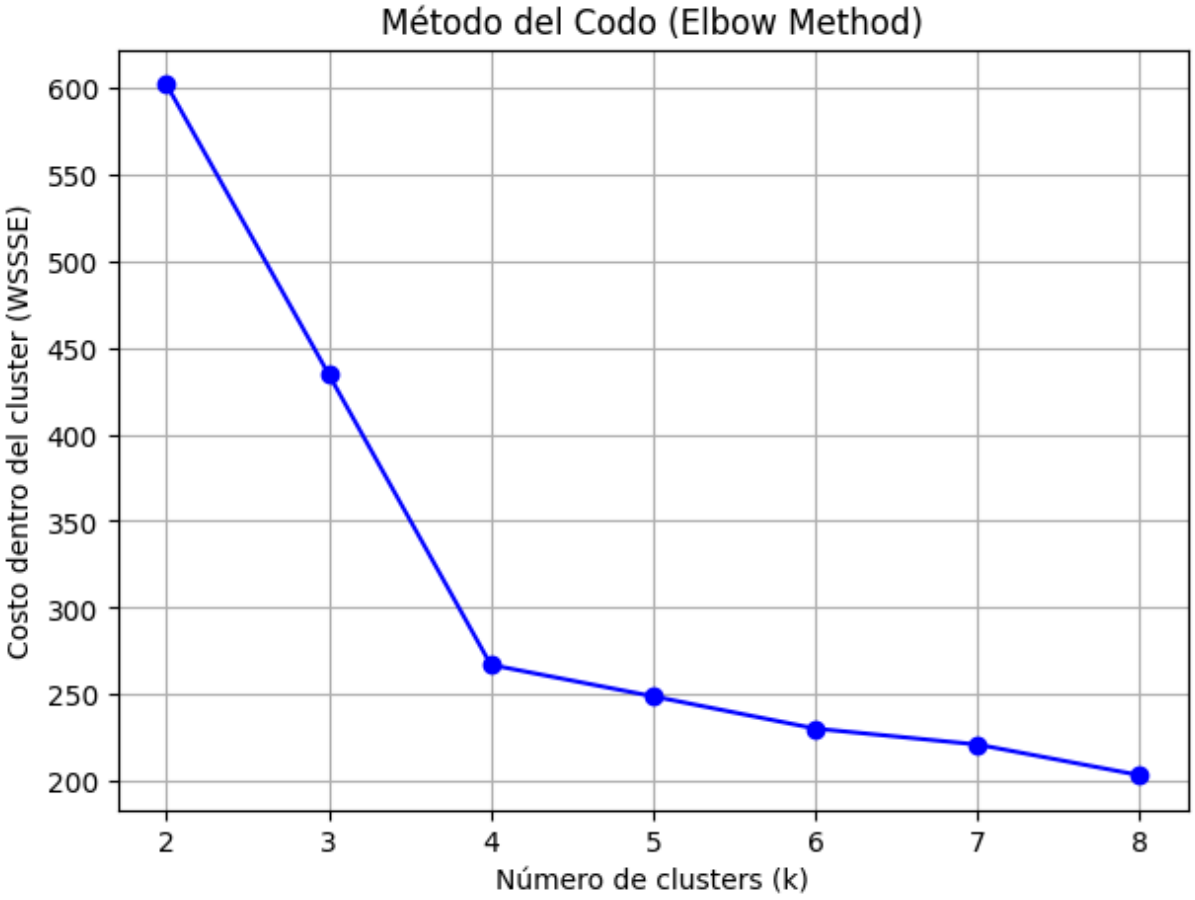
Utilizar el método del codo para obtener un valor óptimo de k

In [10]:

```
from pyspark.ml.clustering import KMeans
import matplotlib.pyplot as plt

# Lista de k a probar
ks = list(range(2, 9))
costs = []

# Calcular el costo (WSSSE) para cada k
for k in ks:
    kmeans = KMeans(featuresCol="features", k=k, seed=42)
    model = kmeans.fit(df_scaled)
    cost = model.summary.trainingCost
    costs.append(cost)
    print(f"k={k}, cost={cost}")

plt.figure(figsize=(7, 5))
plt.plot(ks, costs, marker='o', linestyle='-', color='b')
plt.title('Método del Codo (Elbow Method)')
plt.xlabel('Número de clusters (k)')
plt.ylabel('Costo dentro del cluster (WSSSE)')
plt.xticks(ks)
```

```
plt.grid(True)
plt.show()
```

```
k=2, cost=601.7707512676687
k=3, cost=434.7550730848759
k=4, cost=267.13361168878936
k=5, cost=248.973058822869
k=6, cost=230.40044634373197
k=7, cost=221.19542695456494
k=8, cost=203.4961972545083
```



La gráfica del método del codo permite identificar el número de clústeres que logra un buen equilibrio entre la reducción del costo interno y la simplicidad del modelo. En este caso, se observa que el costo disminuye de forma muy pronunciada desde **k = 2 hasta k = 4**, pero después de ese punto la curva comienza a aplanarse y las mejoras se vuelven mucho más pequeñas. Esto indica que **k = 4** es el punto donde se encuentra el "codo" de la gráfica, ya que representa el valor a partir del cual añadir más clústeres no aporta una ganancia significativa en la compactación de los grupos. Por lo tanto, este número de clústeres se considera el más adecuado para describir la estructura del conjunto de datos, manteniendo un equilibrio razonable entre precisión y simplicidad del modelo.

Entrenar el modelo con distintos valores de k

In [11]:
```
evaluator = ClusteringEvaluator(
    featuresCol="features",
```

```
    predictionCol="prediction",
    metricName="silhouette",
    distanceMeasure="squaredEuclidean"
)

for k in [2, 3, 4]:
    kmeans = KMeans(
        featuresCol="features",
        predictionCol="prediction",
        k=k,
        seed=42
    )
    model = kmeans.fit(df_scaled)
    result = model.transform(df_scaled)

    print(f"\n=== k = {k} ===")
    result.groupBy("prediction").count().show()

    sil = evaluator.evaluate(result)
    print("Silhouette:", sil)
```

```
=== k = 2 ===
+----------+-----+
|prediction|count|
+----------+-----+
|         1|  167|
|         0|  167|
+----------+-----+

Silhouette: 0.817646009401246

=== k = 3 ===
+----------+-----+
|prediction|count|
+----------+-----+
|         1|   88|
|         2|   79|
|         0|  167|
+----------+-----+

Silhouette: 0.7608455651454932

=== k = 4 ===
+----------+-----+
|prediction|count|
+----------+-----+
|         1|   79|
|         3|   83|
|         2|   88|
|         0|   84|
+----------+-----+

Silhouette: 0.7195901966635173
```

Al comparar los resultados de los distintos valores de k, se observa que el índice de Silhouette presenta su valor más alto cuando **k = 2**, con un puntaje de aproximadamente

0.82, lo que indica una separación muy clara entre los grupos. A medida que se incrementa el número de clústeres, el valor del silhouette va disminuyendo gradualmente, alcanzando **0.76** para **k = 3** y **0.72** para **k = 4**. Esta tendencia sugiere que, aunque aumentar el número de grupos permite una división más específica de los datos, también genera solapamientos que reducen la coherencia interna de los clústeres.

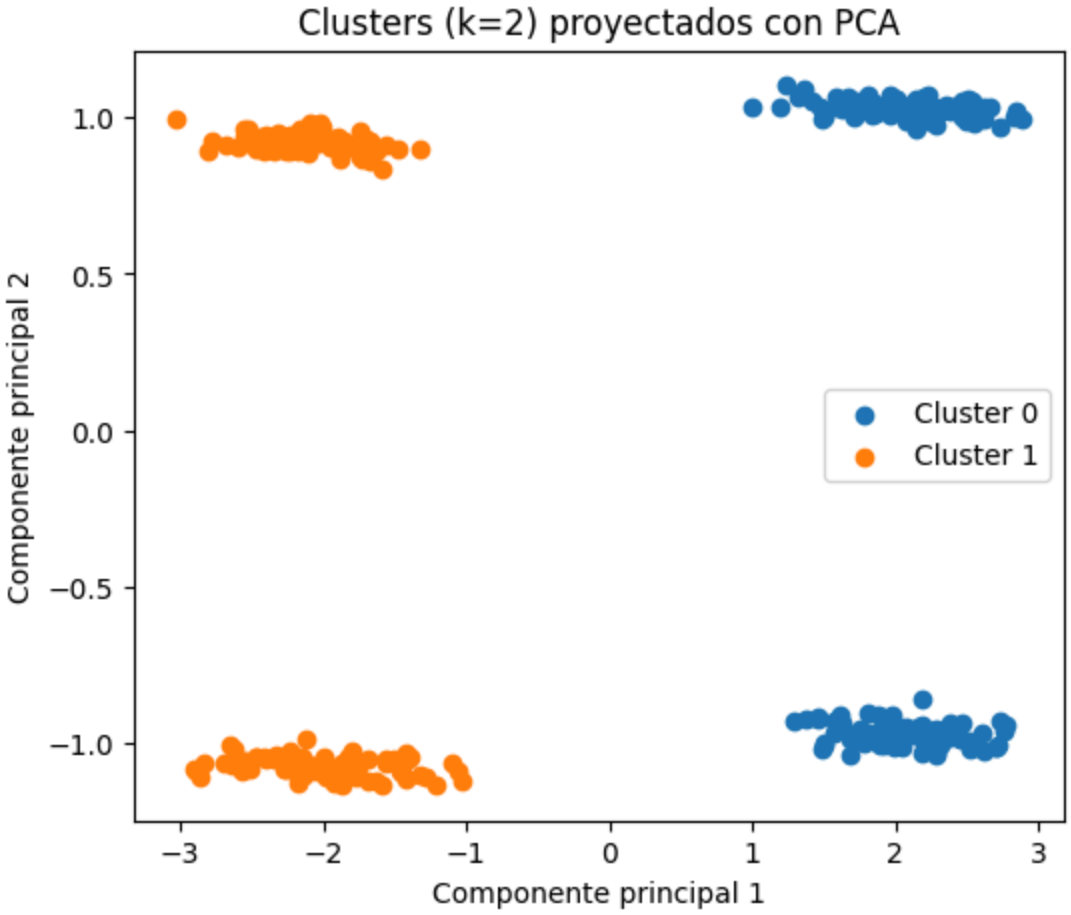
En cuanto a la distribución de los registros, es evidente que para **k = 2** los grupos son perfectamente equilibrados, con 167 elementos en cada uno, lo que refuerza la buena separación observada en el silhouette. Cuando se incrementa a **k = 3** y **k = 4**, los grupos se vuelven más pequeños y su tamaño se distribuye de forma menos uniforme, con algunos clústeres más poblados que otros.

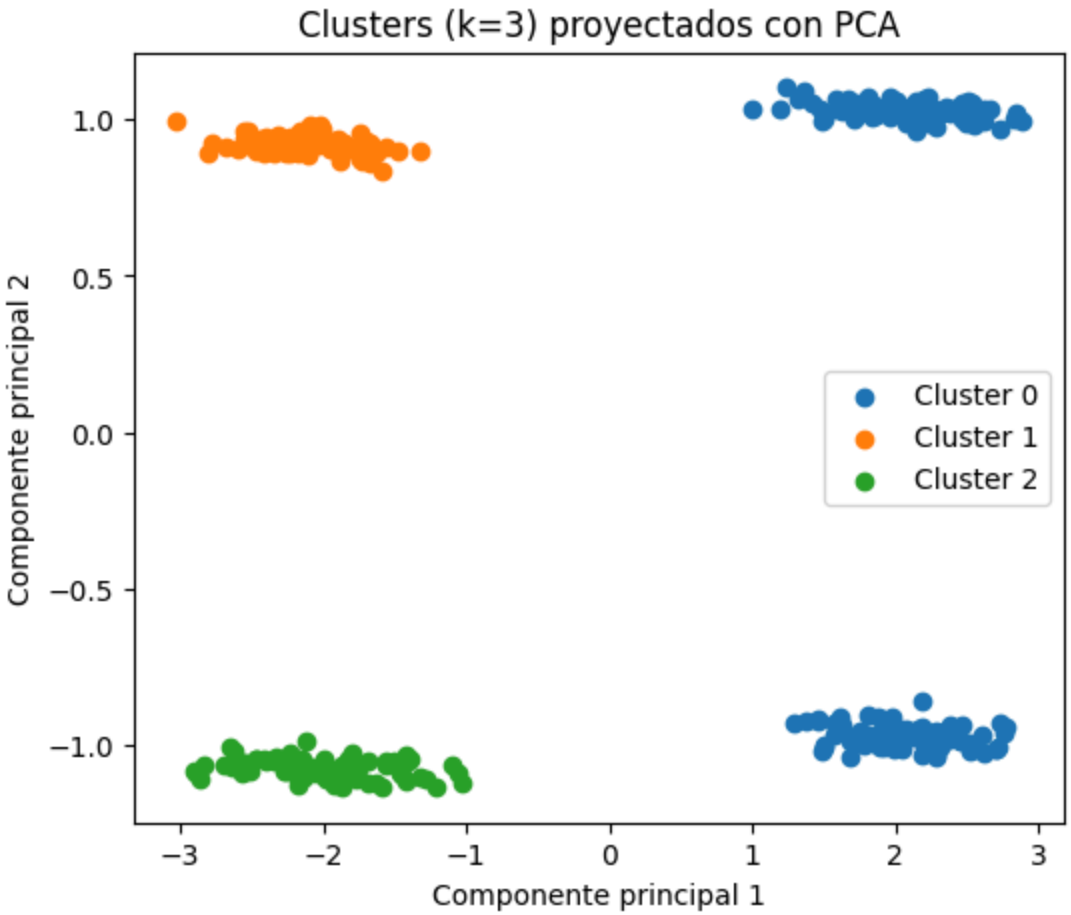
Estos resultados demuestran que aunque el método del codo sugería **k = 4** como una posible opción, el análisis del silhouette indica que el modelo con **k = 2** ofrece una mejor calidad de agrupamiento. Esto significa que los datos pueden dividirse de forma natural en dos grupos bien diferenciados, lo que facilita su interpretación y reduce la posibilidad de ruido o sobresegmentación.

Graficar los clústers

In [12]:
```
# Reducimos a 2 componentes principales
pca = PCA(k=2, inputCol="features", outputCol="pca2")
pca_model = pca.fit(df_scaled)
df_pca = pca_model.transform(df_scaled)

kmeans2 = KMeans(featuresCol="features", k=2, seed=42)
model_k2 = kmeans2.fit(df_scaled)
res_k2 = model_k2.transform(df_pca).select("pca2", "prediction")

kmeans3 = KMeans(featuresCol="features", k=3, seed=42)
model_k3 = kmeans3.fit(df_scaled)
res_k3 = model_k3.transform(df_pca).select("pca2", "prediction")

pdf_k2 = res_k2.toPandas()
pdf_k3 = res_k3.toPandas()

# Función auxiliar para graficar
def plot_clusters(pdf, k):
    x = [vec[0] for vec in pdf["pca2"]]
    y = [vec[1] for vec in pdf["pca2"]]
    labels = pdf["prediction"]

    plt.figure(figsize=(6,5))
    for c in sorted(labels.unique()):
        plt.scatter(
            [x[i] for i in range(len(x)) if labels[i] == c],
            [y[i] for i in range(len(y)) if labels[i] == c],
            label=f"Cluster {c}"
        )
    plt.xlabel("Componente principal 1")
```

```
    plt.ylabel("Componente principal 2")
    plt.title(f"Clusters (k={k}) proyectados con PCA")
    plt.legend()
    plt.show()

plot_clusters(pdf_k2, 2)
plot_clusters(pdf_k3, 3)
```





Al observar las proyecciones de los clústeres en el plano PCA, se puede notar una clara separación entre los grupos, especialmente en el caso de **k = 2**. En esa configuración, los puntos se dividen en dos conjuntos bien definidos que no presentan superposición visible, demostrando que los datos pueden agruparse de manera natural en dos categorías principales. Esta separación tan marcada coincide con el valor más alto del índice de silhouette, reflejando que los clústeres son compactos internamente y están bien distanciados entre sí.

Por otro lado, al aumentar el número de clústeres a **k = 3**, se observa la aparición de un tercer grupo, aunque su inclusión no mejora significativamente la separación global. Si bien el nuevo clúster agrupa un subconjunto de puntos de forma coherente, los límites entre los tres grupos se vuelven menos definidos, lo que puede indicar una segmentación artificial o forzada de los datos. Gracias a la visualización de los clusters, se observa que la opción **k = 2** ofrece la mejor estructura de agrupamiento, ya que logra una división equilibrada, clara y consistente con los resultados cuantitativos obtenidos previamente.

Obtener los centroides

In [16]:
```
centers = model_k2.clusterCenters()
for i, c in enumerate(centers):
    print(f"Centro {i}: {c}")
```

```
Centro 0: [-0.86984075 -0.80244614 -0.02990384 -0.91949279 -0.97535442  0.99101306]
Centro 1: [ 0.86984075  0.80244614  0.02990384  0.91949279  0.97535442 -0.99101306]
```

In [17]:
```
spark.stop()
```

Conclusión

Después de realizar el análisis completo, se concluye que los datos pueden agruparse de manera coherente en dos perfiles principales de atacantes. Aunque el método del codo sugería una posible división en cuatro grupos, los resultados del índice de silhouette, las proyecciones PCA y la observación de los centroides confirman que **k = 2** ofrece una mejor separación, más equilibrada y con menor superposición entre los datos. Esto indica que durante los ataques analizados participaron **dos hackers distintos**, cada uno con un patrón de comportamiento bien definido. Uno de ellos ejecutó ataques más prolongados y con mayor volumen de transferencia de datos, mientras que el otro realizó acciones más rápidas y precisas, afectando un número menor de servidores y páginas. Aunque el sistema haya sido comprometido por dos atacantes con estrategias diferentes, el análisis permitió conlcluir que ambos tenían características lo suficientemente consistentes como para ser identificados de manera clara mediante el modelo de clustering.