

13

Visualización de datos con matplotlib y seaborn

En este capítulo veremos cómo representar gráficamente los datos que hemos aprendido a manipular en el capítulo 12. La visualización es un pilar del análisis exploratorio: una buena gráfica permite detectar patrones, valores atípicos, distribuciones inesperadas y errores en los datos que pasarían inadvertidos en una tabla. En cualquier proyecto de análisis de datos, las representaciones visuales (mapas de calor de actividad, distribuciones de valores, series temporales, curvas de evaluación de modelos) son a menudo la forma más eficaz de comunicar resultados a un equipo o de descubrir comportamientos inesperados. La ciberseguridad es uno de los dominios donde esto resulta útil, por ejemplo al representar tamaños de paquete, series temporales de eventos o curvas ROC de detectores. Trabajaremos con dos bibliotecas complementarias: `matplotlib`, la *lingua franca* de la visualización en Python, y `seaborn`, una capa de alto nivel construida sobre `matplotlib` que produce gráficos estadísticos elegantes con muy poco código.

13.1. Biblioteca matplotlib

`matplotlib` es la biblioteca de visualización más usada en Python. Su API es muy flexible – permite controlar cualquier detalle del gráfico – pero también puede resultar verbosa. La forma habitual de utilizarla es importando el submódulo `pyplot` con el alias estándar `plt`:

```
import matplotlib.pyplot as plt
```

13.1.1. Anatomía de una figura

Una figura de `matplotlib` tiene dos elementos principales:

- La **Figure** (figura): el lienzo completo donde se dibuja.
- Los **Axes** (ejes): cada subgráfico dentro de la figura. Una figura puede tener uno o varios **Axes** organizados en una rejilla.

Hay dos estilos de uso habituales:

- **Estilo *state-machine*** (heredado de Matlab): se llama directamente a `plt.plot(...)`, `plt.title(...)`, etc. Es cómodo para gráficos rápidos.
↪
- **Estilo orientado a objetos**: se obtiene la figura y los ejes con `fig, ax = plt.subplots()` y se invocan los métodos sobre `ax`. Es más verboso, pero más flexible y recomendable para gráficos complejos. Es el estilo que usaremos en este capítulo.
↪

13.1.2. Primer gráfico: una línea

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 2 * np.pi, 100)
5 y = np.sin(x)
6
7 fig, ax = plt.subplots(figsize=(7, 4))
8 ax.plot(x, y, label="sin(x)")
9 ax.set_xlabel("x")
10 ax.set_ylabel("sin(x)")
11 ax.set_title("Función seno")
12 ax.legend()
13 ax.grid(True)
14 plt.show()
```

Algunos comentarios:

- `plt.subplots` devuelve una tupla (`figura`, `ejes`) que desempaquetamos.
- `figsize=(ancho, alto)` se mide en pulgadas. Es un buen parámetro al que prestar atención: las figuras demasiado pequeñas son ilegibles.
- `ax.plot` acepta opciones de estilo: color (`color="red"`), grosor (`linewidth=2`), tipo de línea (`linestyle="--"`), marcador (`marker="o"`).
- `plt.show()` abre una ventana interactiva. En Jupyter no es necesario; en scripts sí.

13.1.3. Tipos de gráfico habituales

```
rng = np.random.default_rng(0)
x = rng.normal(size=200)
y = 0.5 * x + rng.normal(scale=0.3, size=200)

fig, ax = plt.subplots()
ax.scatter(x, y, s=20, alpha=0.6)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_title("Dispersión x vs y")
```

El parámetro `alpha` da transparencia, útil cuando los puntos se solapan.

Histograma. Para visualizar la distribución de una variable:

```
muestras = rng.normal(loc=50, scale=10, size=10000)

fig, ax = plt.subplots()
ax.hist(muestras, bins=40, edgecolor="black")
ax.set_xlabel("Valor")
ax.set_ylabel("Frecuencia")
ax.set_title("Histograma de muestras normales")
```

El parámetro `bins` controla el número de barras: pocos *bins* ocultan estructura, demasiados producen ruido. Un buen punto de partida es la regla de Sturges (`int(np.log2(n)) + 1`).

Gráfico de barras. Útil para variables categóricas:

```
categorias = ["GET", "POST", "PUT", "DELETE"]
conteos = [1245, 320, 80, 25]

fig, ax = plt.subplots()
ax.bar(categorias, conteos, color="steelblue")
ax.set_ylabel("Peticiones")
ax.set_title("Distribución de métodos HTTP")
```

Boxplot. Resume cinco estadísticos (mínimo, primer cuartil, mediana, tercer cuartil, máximo) y resalta valores atípicos. Muy útil para comparar distribuciones:

```
g1 = rng.normal(50, 5, 200)
g2 = rng.normal(55, 8, 200)
g3 = rng.normal(48, 12, 200)
```

```
fig, ax = plt.subplots()
ax.boxplot([g1, g2, g3], tick_labels=["Servicio A", "Servicio
↪      B", "Servicio C"])
ax.set_ylabel("Latencia (ms)")
ax.set_title("Distribución de latencias por servicio")
```

Mapa de calor (heatmap). Para matrices 2D, donde el color codifica la magnitud:

```
matriz = rng.random((10, 12))

fig, ax = plt.subplots()
im = ax.imshow(matriz, cmap="viridis", aspect="auto")
fig.colorbar(im, ax=ax, label="Intensidad")
ax.set_title("Matriz de actividad")
```

13.1.4. Múltiples subgráficos

`plt.subplots(filas, cols)` devuelve una matriz de `Axes` con la que componer varios gráficos en una misma figura:

```
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

axes[0].plot(x, np.sin(x))
axes[0].set_title("sin(x)")

axes[1].plot(x, np.cos(x), color="red")
axes[1].set_title("cos(x)")

fig.tight_layout()    # Ajusta espacios automáticamente
```

13.1.5. Personalización: ejes, leyenda, anotaciones

- `ax.set_xlim(a, b)` / `ax.set_ylim(a, b)`: rango de los ejes.
- `ax.set_xticks(...)` / `ax.set_yticks(...)`: marcas concretas.
- `ax.set_xscale("log")`: escala logarítmica (muy útil cuando los datos abarcan varios órdenes de magnitud, como tamaños de fichero o tiempos de respuesta).
- `ax.legend(loc="best")`: leyenda automática con las etiquetas `label=` pasadas a `plot/scatter`.
- `ax.annotate("texto", xy=(x, y), ...)`: anotación con flecha en un punto concreto.
- `ax.axvline(x, color="red", linestyle="--")`: línea vertical (útil para marcar umbrales).

13.1.6. Guardar la figura

`fig.savefig("salida.png", dpi=150, bbox_inches="tight")` guarda la figura en disco. Formatos típicos: `.png` (mapa de bits, web), `.pdf` y `.svg` (vectoriales, ideales para publicación). El parámetro `dpi` controla la resolución; `bbox_inches="tight"` recorta los márgenes en blanco.

Reproducibilidad: cierra siempre la figura

En scripts que generan muchas figuras (en bucle, por ejemplo), `matplotlib` acumula todas en memoria. Tras `savefig`, llama a `plt.close(fig)` para liberar la memoria de esa figura.

13.2. Biblioteca *seaborn*

seaborn se construye sobre `matplotlib` y aporta dos cosas: estilos visuales más cuidados por defecto, y una API de alto nivel pensada para datos tabulares (`DataFrame` de *pandas*). Se importa convencionalmente como `sns`:

```
import seaborn as sns
```

13.2.1. Filosofía y datos en formato largo

Casi todas las funciones de *seaborn* aceptan un `DataFrame` y los nombres de las columnas que actúan como ejes, color, tamaño, etc. Para sacarle partido, los datos deben estar en *formato largo*: una fila por observación, con columnas para cada variable. Si tus datos están en formato ancho (una columna por categoría), conviértelos antes con `pd.melt` (cap. 12).

13.2.2. Gráficos estadísticos esenciales

`histplot` y `kdeplot`. Histograma con opción de superponer una estimación de densidad por núcleo (KDE):

```
import seaborn as sns
import numpy as np
import pandas as pd

rng = np.random.default_rng(0)
df = pd.DataFrame({
    "latencia": np.concatenate([rng.normal(50, 5, 1000),
                                rng.normal(70, 10, 500)]),
    "servicio": ["A"] * 1000 + ["B"] * 500
})
```

```
↪ sns.histplot(data=df, x="latencia", hue="servicio", kde=True, bins=40)
```

scatterplot y regplot. Dispersión, opcionalmente con ajuste por regresión:

```
df = pd.DataFrame({
    "x": rng.normal(size=200),
})
df["y"] = 0.5 * df["x"] + rng.normal(scale=0.3, size=200)
sns.regplot(data=df, x="x", y="y", line_kws={"color": "red"})
```

boxplot y violinplot. Comparan distribuciones por categoría. violinplot añade una estimación de densidad lateral:

```
↪ sns.boxplot(data=df_eventos, x="servicio", y="latencia", hue="entorno")
sns.violinplot(data=df_eventos, x="servicio", y="latencia")
```

heatmap. Especialmente útil para matrices de correlación:

```
corr = df.select_dtypes(include="number").corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", center=0)
```

pairplot. Matriz de gráficos de dispersión entre todos los pares de variables numéricas. Muy útil en análisis exploratorio:

```
sns.pairplot(df, hue="clase", diag_kind="kde")
```

13.2.3. Estilos y paletas

seaborn ofrece estilos globales con `sns.set_theme(style=..., palette=...)`. Algunos estilos comunes son `"whitegrid"`, `"darkgrid"` y `"ticks"`. Para paletas de color: `"deep"`, `"colorblind"` (especialmente recomendable en publicaciones por accesibilidad), `"viridis"` (perceptualmente uniforme, ideal para escalas continuas).

```
sns.set_theme(style="whitegrid", palette="colorblind")
```

13.3. Aplicación: visualización de registros de actividad

Mostramos a continuación cuatro patrones recurrentes de visualización en análisis de seguridad. Todos presuponen que los datos están en un `DataFrame` pandas – la salida natural del capítulo anterior.

13.3.1. Distribución de tamaños de paquete

Una distribución muy sesgada o con varios picos suele revelar una mezcla de tráfico heterogéneos (HTTP corto + transferencias largas, por ejemplo). Un histograma en escala logarítmica facilita la lectura:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(7, 4))
ax.hist(df_paquetes["tamaño"], bins=80)
ax.set_xscale("log")
ax.set_xlabel("Tamaño del paquete (bytes, escala log)")
ax.set_ylabel("Frecuencia")
ax.set_title("Distribución de tamaños de paquete")
```

13.3.2. Mapa de calor de actividad por hora y día

Combinando `pandas.pivot` con `seaborn.heatmap` generamos un mapa que revela patrones horarios y semanales (jornadas laborales, actividad fuera de horario, etc.):

```
import seaborn as sns

df_eventos["hora"] = df_eventos["timestamp"].dt.hour
df_eventos["dia"] = df_eventos["timestamp"].dt.day_name()

tabla = (df_eventos.groupby(["dia", "hora"]).size()
         .unstack(fill_value=0))

fig, ax = plt.subplots(figsize=(10, 4))
sns.heatmap(tabla, cmap="YlOrRd", ax=ax)
ax.set_title("Eventos por día y hora")
```

13.3.3. Serie temporal con umbrales

Para detectar visualmente picos en una serie temporal, conviene superponer la media móvil y un umbral (e.g., $\mu + 3\sigma$):

```
ts = df_eventos.set_index("timestamp").resample("h").size()
mm = ts.rolling(24).mean()
umbral = mm + 3 * ts.rolling(24).std()
```

```
fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(ts.index, ts, label="eventos/hora", linewidth=0.8)
ax.plot(mm.index, mm, label="media móvil 24h", color="orange",
↪      )
↪ ax.plot(umbral.index, umbral, label="mu+3sigma", color="red",
      linestyle="--")
ax.legend()
ax.set_title("Eventos por hora con detección de picos")
```

13.3.4. Matriz de correlación de features

Antes de entrenar un modelo de aprendizaje (capítulo siguiente), es muy útil inspeccionar las correlaciones entre las variables de entrada para detectar redundancia:

```
sns.set_theme(style="white")
fig, ax = plt.subplots(figsize=(7, 6))
sns.heatmap(df_features.corr(), annot=True, fmt=".2f",
            cmap="coolwarm", center=0, ax=ax)
ax.set_title("Correlación entre variables")
```

13.4. Ejercicios

EJERCICIO 13.1

Escribe un programa en Python que cargue un fichero CSV con tamaños de paquete (columna `tamano`, en bytes) y muestre dos gráficas en una sola figura: a la izquierda, un histograma con escala lineal; a la derecha, el mismo histograma con escala logarítmica en el eje X. Las dos gráficas deben tener título, etiquetas en los ejes y un número razonable de *bins*.

El programa debe usar `matplotlib.pyplot.subplots` con 1, 2 paneles y guardar la figura resultante en un fichero «`tamanos.png`» con resolución 150 dpi.

Solución (girar para leer)

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Datos sintéticos: tamaños de paquete en bytes (cola
5   larga)
6 rng = np.random.default_rng(0)
7 tamaño = rng.lognormal(mean=6.0, sigma=1.2, size=5000)
8 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
9
10 # Panel izquierdo: histograma en escala lineal
11 ax1.hist(tamaño, bins=40, color="steelblue", edgecolor="
12   white")
13 ax1.set_title("Escala lineal")
14 ax1.set_xlabel("Tamaño (bytes)")
15 ax1.set_ylabel("Frecuencia")
16 # Panel derecho: mismo histograma con eje X logarítmico
17 bins_log = np.logspace(np.log10(tamaño.min()), np.log10
18   (tamaño.max()), 40)
19 ax2.hist(tamaño, bins=bins_log, color="darkorange",
20   edgecolor="white")
21 ax2.set_xscale("log")
22 ax2.set_title("Escala logarítmica")
23 ax2.set_xlabel("Tamaño (bytes)")
24 ax2.set_ylabel("Frecuencia")
25 fig.tight_layout()
26 fig.savefig("tamanos.png", dpi=150)

```

Código 13.1: Solución al Ejercicio 13.1.

EJERCICIO 13.2

Dado un fichero CSV de eventos con una columna de timestamp (formato ISO) y un identificador de usuario, escribe un programa que genere un mapa de calor de actividad con días de la semana en el eje Y y horas del día (0–23) en el eje X. Las celdas deben mostrar el número total de eventos para esa combinación.

Usa `pandas` para agrupar y `seaborn.heatmap` para la visualización. La figura debe guardarse como «heatmap.pdf».

Solución (girar para leer)

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5
6 # Datos sintéticos: eventos con timestamp ISO e id de usuario
7 rng = np.random.default_rng(0)
8 n = 20000
9 inicio = pd.Timestamp("2024-01-01")
10 desplaz = rng.integers(0, 30 * 24 * 60, size=n) # minutos sobre un mes
11 df = pd.DataFrame({
12     "timestamp": inicio + pd.to_timedelta(desplaz, unit="m"),
13     "usuario": rng.integers(1, 50, size=n),
14 })
15
16 # Extraer día de la semana (0=Lunes) y hora del día
17 df["día"] = df["timestamp"].dt.dayofweek
18 df["hora"] = df["timestamp"].dt.hour
19
20 # Agrupar contando eventos por (día, hora)
21 tabla = df.groupby(["día", "hora"]).size().unstack(fill_value=0)
22 días = ["Lun", "Mar", "Mié", "Jue", "Vie", "Sab", "Dom"]
23 tabla.index = días
24
25 fig, ax = plt.subplots(figsize=(10, 4))
26 sns.heatmap(tabla, cmap="viridis", ax=ax, cbar_kws={"label": "Eventos"})
27 ax.set_xlabel("Hora del día")
28 ax.set_ylabel("Día de la semana")
29 ax.set_title("Mapa de calor de actividad")
30 fig.tight_layout()
31 fig.savefig("heatmap.pdf")

```

Código 13.2: Solución al Ejercicio 13.2.

EJERCICIO 13.3

A partir de un fichero CSV con dos columnas (**servicio** y **latencia_ms**), escribe un programa que dibuje un *boxplot* comparando las distribuciones de latencias por servicio. Identifica visualmente los servicios con mayor mediana y los que tienen más valores atípicos.

Genera la figura tanto con **matplotlib** como con **seaborn** (en paneles distintos) para apreciar la diferencia de estilo por defecto.

Solución (girar para leer)

```

1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5
6  # Datos sintéticos: latencia (ms) por servicio, con
   valores atípicos
7  rng = np.random.default_rng(0)
8  servicios = ["auth", "api", "db", "cache"]
9  medias = {"auth": 40, "api": 80, "db": 120, "cache":
   20}
10  filas = []
11  for s in servicios:
12      base = rng.normal(medias[s], 10, size=200)
13      atip = rng.normal(medias[s] * 4, 20, size=8) #
   outliers
14      for v in np.concatenate([base, atip]):
15          filas.append((s, v))
16  df = pd.DataFrame(filas, columns=["servicio", "
   latencia-ms"])
17
18  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(11, 4))
19
20  # Panel izquierdo: boxplot con matplotlib
   datos = [df[df.servicio == s].latencia-ms.values for s
   in servicios]
21
22  ax1.boxplot(datos, tick_labels=servicios)
23  ax1.set_title("matplotlib")
24  ax1.set_xlabel("servicio")
25  ax1.set_ylabel("latencia (ms)")
26
27  # Panel derecho: boxplot con seaborn
   sns.boxplot(data=df, x="servicio", y="latencia-ms", ax=
   ax2)
28
29  ax2.set_title("seaborn")
30  ax2.set_xlabel("servicio")
31  ax2.set_ylabel("latencia (ms)")
32
33  fig.tight_layout()
34  fig.savefig("boxplot.png", dpi=150)

```

Código 13.3: Solución al Ejercicio 13.3.

EJERCICIO 13.4

Dado un fichero CSV con una columna de timestamp (granularidad de minutos) y una columna numérica **eventos**, escribe un programa que:

1. Remuestree los datos a frecuencia horaria sumando.
2. Calcule la media móvil de 24 horas y el umbral $\mu + 3\sigma$ (con σ también en ventana móvil de 24 h).
3. Dibuje la serie original, la media móvil y el umbral en la misma gráfica.
4. Marque con un círculo rojo los puntos que superan el umbral.

Solución (girar para leer)

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 # Datos sintéticos: eventos por minuto durante 20 días,
  con picos
6 rng = np.random.default_rng(0)
7 idx = pd.date_range("2024-01-01", periods=20 * 24 * 60,
  freq="min")
8 base = rng.poisson(5, size=len(idx))
9 base[5000:5060] += 80 # ráfaga anómala
10 serie = pd.Series(base, index=idx, name="eventos")
11
12 # 1. Remuestrear a frecuencia horaria sumando
  hora = serie.resample("h").sum()
13
14 # 2. Media móvil y umbral mu + 3 sigma (ventana de 24 h
  media = hora.rolling(24, min_periods=1).mean()
15 desv = hora.rolling(24, min_periods=1).std().fillna(0)
16 umbral = media + 3 * desv
17
18 # 3-4. Dibujar serie, media móvil, umbral y marcar
  anomalías
19 fig, ax = plt.subplots(figsize=(11, 4))
20 ax.plot(hora.index, hora, label="Eventos", color=
  "steelblue")
21 ax.plot(media.index, media, label="Media móvil 24 h",
  color="green")
22 ax.plot(umbral.index, umbral, label="Umbral mu+3sigma",
  color="orange", ls="--")
23 anom = hora[hora > umbral]
24 ax.scatter(anom.index, anom, color="red", zorder=5,
  label="Anomalías")
25 ax.set_xlabel("Tiempo")
26 ax.set_ylabel("Eventos por hora")
27 ax.set_title("Detección de anomalías en serie temporal")
28
29 ax.legend()
30 fig.tight_layout()
31 fig.savefig("anomalías.png", dpi=150)

```

Código 13.4: Solución al Ejercicio 13.4.

EJERCICIO 13.5

Carga un fichero CSV con varias columnas numéricas y una columna categórica **clase** (con 2 o 3 valores). Genera un pairplot de seaborn coloreando los puntos por la clase, y guárdalo como «pairplot.png». ¿Se aprecia visualmente alguna combinación de variables que separe bien las clases?

Solución (girar para leer)

```

1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4
5 # Datos sintéticos: 3 clases con medias distintas en 4
6   variables
7   rng = np.random.default_rng(0)
8   centros = {"normal": [0, 0, 0, 0], "sospechoso": [3, 3, 3, 3],
9             "malicioso": [6, 1, 4, 2]}
10  for clase, c in centros.items():
11      muestras = rng.normal(c, 1.0, size=(150, 4))
12      for m in muestras:
13          filas.append([*m, clase])
14  df = pd.DataFrame(filas, columns=["v1", "v2", "v3", "v4",
15                                  "clase"])
16  # Pairplot coloreado por la clase
17  g = sns.pairplot(df, hue="clase", diag_kind="kde",
18                  corner=False)
19  g.fig.suptitle("Relaciones entre variables por clase",
20                y=1.02)
21  g.savefig("pairplot.png", dpi=150)

```

Código 13.5: Solución al Ejercicio 13.5.

EJERCICIO 13.6

Dado un fichero CSV con $n \geq 6$ columnas numéricas, calcula la matriz de correlación de Pearson entre todas las columnas y dibújala como un mapa de calor anotado con los valores numéricos. La diagonal debe quedar resaltada. Identifica los pares de variables con correlación absoluta superior a 0.8.

Solución (girar para leer)

```

2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5
6 # Datos sintéticos: 6 variables, algunas fuertemente
7 correladas
8 rng = np.random.default_rng(0)
9 a = rng.normal(size=n)
10 b = rng.normal(size=n)
11 df = pd.DataFrame({
12     "v1": a,
13     "v2": a * 0.95 + rng.normal(0, 0.1, n), # muy
14     "v3": b,
15     "v4": b * -0.9 + rng.normal(0, 0.1, n), #
16     "v5": rng.normal(size=n),
17     "v6": rng.normal(size=n),
18 })
19
20 # Matriz de correlacion de Pearson
21 corr = df.corr()
22
23 # Mapa de calor anotado con diagonal resaltada
24 fig, ax = plt.subplots(figsize=(7, 6))
25 sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm",
26             center=0,
27             linewidths=0.5, linecolor="white", ax=ax)
28 ax.add_patch(plt.Rectangle((1, 1), 1, 1, fill=False),
29             edgecolor="black", lw=2)
30 ax.set_title("Matriz de correlacion de Pearson")
31 fig.tight_layout()
32 fig.savefig("correlacion.png", dpi=150)
33
34 # Pares con correlacion absoluta superior a 0.8
35 for i in range(len(corr)):
36     for j in range(i + 1, len(corr)):
37         if abs(corr.iloc[i, j]) > 0.8:
38             print(f"{corr.columns[i]} - {corr.columns[j]}: {corr.iloc[i, j]:.2f}")

```

EJERCICIO 13.7

Combina varias técnicas para construir un *dashboard* compacto en una única figura, con cuatro paneles dispuestos en una rejilla 2×2:

1. Histograma de la duración de las sesiones.
2. Boxplot de tráfico (bytes) por categoría de usuario.
3. Serie temporal de eventos por hora del día.
4. Mapa de calor de actividad por día y hora.

Usa `plt.subplots(2, 2)` y `fig.tight_layout()` para que los paneles no se solapen. Guarda el resultado como «`dashboard.pdf`».

Solución (girar para leer)

```

7  rng = np.random.default_rng(0)
8  n = 4000
9  ts = pd.Timestamp("2024-01-01") + pd.to_timedelta(
10     rng.integers(0, 7 * 24 * 60, n), unit="m")
11  df = pd.DataFrame({
12     "duracion": rng.lognormal(3, 0.6, n),
13     "bytes": rng.lognormal(8, 1.0, n),
14     "categoria": rng.choice(["nuevo", "habitual", "
15     "timestamp": ts,
16     })
17  df["dia"] = df["timestamp"].dt.dayofweek
18  df["hora"] = df["timestamp"].dt.hour
19
20  fig, axs = plt.subplots(2, 2, figsize=(11, 8))
21
22  # 1. Histograma de duracion de sesiones
23  axs[0, 0].hist(df["duracion"], bins=40, color="
24     steelblue")
25  axs[0, 0].set_title("Duracion de sesiones")
26  axs[0, 0].set_xlabel("Segundos")
27  axs[0, 0].set_ylabel("Frecuencia")
28
29  # 2. Boxplot de trafico por categoria
30  sns.boxplot(data=df, x="categoria", y="bytes", ax=axs
31     [0, 1])
32  axs[0, 1].set_title("Trafico por categoria")
33
34  # 3. Serie de eventos por hora del dia
35  por_hora = df.groupby("hora").size()
36  axs[1, 0].plot(por_hora.index, por_hora.values, marker=
37     "o")
38  axs[1, 0].set_title("Eventos por hora")
39  axs[1, 0].set_xlabel("Hora")
40  axs[1, 0].set_ylabel("Eventos")
41
42  # 4. Mapa de calor de actividad por dia y hora
43  tabla = df.groupby(["dia", "hora"]).size().unstack(
44     fill_value=0)
45  sns.heatmap(tabla, cmap="viridis", ax=axs[1, 1])
46  axs[1, 1].set_title("Actividad dia/hora")
47  fig.tight_layout()
48  fig.savefig("dashboard.pdf")

```