Guion Teleprompter (5 minutos)

00:00–00:10 Hoy te mostraré, de principio a fin, cómo analizamos tráfico de red con Python: cargaremos datos, estimaremos parámetros, generaremos gráficas y detectaremos anomalías.

00:10–00:25 Trabajamos con un archivo CSV que trae marcas de tiempo, tamaño de paquete y protocolo. El proceso produce tablas en formato CSV, figuras en PNG y un informe listo para exportar a PDF.

00:25–00:40 Usaremos dos modelos base: Poisson para el número de paquetes por segundo y Exponencial para los tiempos entre llegadas. Además, revisaremos la relación entre protocolo y tamaño, y cerraremos con detección de anomalías.

00:40–00:55 Arrancamos leyendo el CSV. La función de carga interpreta el formato de fecha y hora, ordena los registros por tiempo y deja los datos listos para agrupar y calcular.

Notas de apoyo (código a mostrar):

- Archivo `src/data/loaders.py` → `read_network_csv(...)` (parsing robusto):

```
def read_network_csv(path: str):
        # Lee CSV, intenta distintos formatos de fecha/hora
        # y retorna una lista de registros tipados
        ...
```

- Explica que devuelve lista de objetos con `timestamp`, `packet_size`, `protocol`.

00:55–01:10 Para la parte discreta, contamos cuántos paquetes llegan en cada segundo. Construimos un vector continuo de segundos y rellenamos con ceros donde no hubo llegadas.

Notas de apoyo (código a mostrar):

- Archivo `src/analysis/statistics.py`:
  - `group_counts_per_second(timestamps_sec)` agrupa por segundo.
  - `expand_counts_with_zeros(counts_per_sec)` completa el rango con ceros.

```
counts_abs = group_counts_per_second(ts)
counts_vector_full = expand_counts_with_zeros(counts_abs)
```

01:10–01:25 Con esos conteos estimamos lambda como el promedio de paquetes por segundo en la ventana analizada. Calculamos también el índice de dispersión, que compara la varianza con la media.

Notas de apoyo (código a mostrar):

- Archivo `analysis_cli.py` (manejo de `--seconds-range` 1-based):

```
s_ini_label, s_fin_label = map(int, args.seconds_range.split('-'))
s_ini, s_fin = s_ini_label - 1, s_fin_label - 1
counts_vector = counts_vector_full[s_ini:s_fin+1]
counts_rel = {label: counts_vector[idx]
                            for label, idx in zip(range(s_ini_label,
s_fin_label+1),

range(len(counts_vector)))}
```

- Cálculo de λ e índice Var/Media:

```
lam_counts = sum(counts_vector) / len(counts_vector)
iod = index_of_dispersion(counts_vector)
```

01:25–01:40 Si el proceso se comporta como Poisson, la media y la varianza deberían ser similares. Un índice cercano a uno sugiere coherencia; muy por debajo o por encima puede indicar sub o sobredispersión.

01:40–01:55 A partir de lambda generamos la distribución teórica de Poisson y la superponemos al histograma de los datos. Esto nos permite evaluar visualmente el ajuste del modelo.

Notas de apoyo (código a mostrar):

- Archivo `src/analysis/statistics.py` → `poisson_pmf(k, λ)`.
- En `analysis_cli.py` (gráfico):

```
xs = np.arange(0, max(obs)+5)
pmf = [poisson_pmf(int(k), lam_counts) for k in xs]
plt.hist(obs, bins=range(0, max(obs)+2), density=True,
                 alpha=0.6, edgecolor="#333", linewidth=0.8)
plt.plot(xs, pmf, 'o-', label=f"Poisson(lambda={lam_counts:.3f})")
```

01:55–02:10 Además, exportamos una tabla para Excel con las frecuencias observadas, las probabilidades teóricas de Poisson y las frecuencias esperadas. Con eso puedes replicar el gráfico directamente en Excel.

Notas de apoyo (código a mostrar):

- En `analysis_cli.py` (tabla Poisson para Excel):

```
rows.append({
        "k_paquetes_por_seg": k,
        "frecuencia_observada": observed,
        "frecuencia_relativa": rel_freq,
        "poisson_pmf_teorica": p_theory,
        "frecuencia_esperada": expected
```

```
})
df_excel.to_csv(out/"poisson_histogram_table.csv", index=False)
```

02:10–02:25 Pasamos a los tiempos entre llegadas. Calculamos las diferencias entre marcas de tiempo consecutivas y estimamos lambda como el inverso de la media de esos intervalos.

Notas de apoyo (código a mostrar):

- `src/analysis/statistics.py`:

```
def interarrival_times(ts):
        times = sorted(ts)
        return [t2 - t1 for t1, t2 in zip(times[:-1], times[1:])]
def estimate_lambda_from_interarrivals(deltas):
        return 1.0 / (sum(deltas)/len(deltas))
```

02:25–02:40 Graficamos el histograma de interarribos y dibujamos encima la densidad teórica de la distribución Exponencial con el lambda estimado. Esto sirve como primera comprobación visual del modelo.

Notas de apoyo (código a mostrar):

- En `analysis_cli.py` (gráfico continuo):

```
xs_cont = np.linspace(0, max(deltas), 100)
pdf = [exponential_pdf(x, lam_inter) for x in xs_cont]
plt.hist(deltas, bins=30, density=True, alpha=0.6,
                 edgecolor="#333", linewidth=0.8)
plt.plot(xs_cont, pdf, label=f"Exp(lambda={lam_inter:.3f})")
```

02:40–02:55 Opcionalmente, realizamos una prueba de Kolmogórov–Smirnov. Con datos discretizados por segundos y empates, el p-valor puede ser muy bajo, por eso interpretamos esa prueba con cautela.

Notas de apoyo (código a mostrar):

- En `analysis_cli.py` (KS):

```
from scipy.stats import kstest
ks = kstest(deltas, 'expon', args=(0, 1.0/lam_inter))
print(ks.statistic, ks.pvalue)
```

02:55–03:10 Para analizar la relación entre protocolo y tamaño, discretizamos el tamaño con un umbral en bytes y construimos una tabla de contingencia con probabilidades conjuntas, marginales y condicionales.

Notas de apoyo (código a mostrar):

- `src/analysis/statistics.py` → `contingency_protocol_size(...)` (mapea 6→TCP, 17→UDP; tamaño ≤500→Pequeño, >500→Grande).

03:10–03:25 Calculamos métricas como la probabilidad de TCP, la probabilidad de paquetes grandes y la probabilidad de grandes condicionada a TCP. Con eso verificamos si las variables parecen independientes.

Notas de apoyo (código a mostrar):

- En `analysis_cli.py` (tabla completa y resumen):

```
rows_full.append({
        'protocolo': prot,
        'tam_categoria': sc,
        'conteo': count,
        'probabilidad': prob
})
pd.DataFrame(rows_full).to_csv("contingency_full.csv", index=False)
```

03:25–03:40 Representamos la tabla conjunta en un mapa de calor con las probabilidades en cada celda. Esto ayuda a identificar patrones o concentraciones por protocolo y tamaño.

Notas de apoyo (código a mostrar):

- En `analysis_cli.py` (heatmap):

```
mat = np.array([[p_tcp_peq, p_tcp_gra], [p_udp_peq, p_udp_gra]])
im = ax.imshow(mat, cmap='Blues')
ax.set_xticks([0,1], labels=['Pequeño','Grande'])
ax.set_yticks([0,1], labels=['TCP','UDP'])
```

03:40–03:55 Cerramos con detección de anomalías. Usamos una regla simple: marcamos como anómalo un segundo si el conteo supera la media más tres veces la raíz de la media.

Notas de apoyo (código a mostrar):

- `src/analysis/statistics.py` → `poisson_anomaly_threshold(lam, z=3.0)`.
- `analysis_cli.py` (detección y figura):

```
k_thresh = poisson_anomaly_threshold(lam_counts, z=3.0)
anomalies = [(sec, cnt) for sec, cnt in counts_rel.items() if cnt > k_thresh]
plt.bar(secs, vals); plt.axhline(y=k_thresh, linestyle='--')
plt.savefig('anomalies_plot.png')
```

03:55–04:10 Generamos un gráfico con las barras de conteos por segundo y una línea horizontal con el umbral. Si algún segundo cruza esa línea, se reporta como anomalía en la tabla correspondiente.

04:10–04:25 Si no hay anomalías, significa que ningún segundo excede el umbral. Para hacer el detector más sensible puedes ampliar la ventana de análisis o reducir el multiplicador del umbral.

04:25–04:40 Todo el proceso está encapsulado en un script de análisis que recibe la ruta del CSV, la carpeta de salida y opciones como la ventana temporal o la exportación de tablas para Excel.

Notas de apoyo (código a mostrar):

- Ejecución desde PowerShell (opcional en overlay):

```
.\.venv\Scripts\python.exe analysis_cli.py .\network_traffic.csv \
    --out out --seconds-range 1-7 --excel-table --excel-compact
```

04:40–04:55 El informe final integra las figuras y tablas clave, y puede exportarse a PDF. Incluye una portada, la metodología, los resultados, la discusión y los anexos con todas las salidas.

04:55–05:00 Con esto tienes un pipeline reproducible para analizar tráfico de red: desde la carga de datos hasta las gráficas y la detección de anomalías, listo para adaptar a tus propios escenarios.