```python
import tkinter as tk
from tkinter import ttk, messagebox
import requests
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# ----------------------------
# Configuración de opciones UI
# ----------------------------
CITIES = {
    "León, Gto": (21.12, -101.68),
    "CDMX": (19.43, -99.13),
    "Guadalajara": (20.67, -103.35),
    "Monterrey": (25.68, -100.31),
    "Madrid": (40.42, -3.70),
    "Bogotá": (4.71, -74.07),
    "Buenos Aires": (-34.61, -58.38),
}

# variable -> (etiqueta legible, nombre en Open-Meteo)
VARIABLES = {
    "Temperatura (°C)": "temperature_2m",
    "Humedad relativa (%)": "relativehumidity_2m",
    "Viento a 10 m (m/s)": "windspeed_10m",
    "Precipitación (mm)": "precipitation",
}

# ----------------------------
# Lógica de datos
# ----------------------------
def fetch_data(latitude: float, longitude: float, hourly_var: str,
past_days: int = 1):
    """
    Conecta con la API de Open-Meteo y obtiene series horarias para la
variable indicada.
    Devuelve dos listas: horas (str ISO) y valores (float/int).
    """
    try:
        if past_days < 0:
            past_days = 0
```

```
        url = (
            "https://api.open-meteo.com/v1/forecast"
            f"?latitude={latitude}&longitude={longitude}"
            f"&hourly={hourly_var}&past_days={past_days}"
            "&timezone=auto"
        )
        resp = requests.get(url, timeout=15)
        resp.raise_for_status()
        data = resp.json()

        horas = data["hourly"]["time"]
        valores = data["hourly"][hourly_var]

        if not horas or not valores or len(horas) != len(valores):
            raise ValueError("Respuesta incompleta de la API.")

        # Limpiar etiquetas: quedarse con HH:MM y fecha corta para
legibilidad
        # Ejemplo: "2025-10-03T07:00" -> "10-03 07:00"
        etiquetas = []
        for t in horas:
            # Espera formato ISO "YYYY-MM-DDTHH:MM"
            try:
                etiquetas.append(f"{t[5:10]} {t[11:16]}")
            except Exception:
                etiquetas.append(t)

        return etiquetas, valores
    except Exception as e:
        messagebox.showerror("Error", f"No se pudieron obtener los
datos:\n{e}")
        return [], []

# ----------------------------
# Gráficas
# ----------------------------
def create_line_chart(x_labels, y_vals, titulo, y_label):
    fig, ax = plt.subplots(figsize=(7, 3))
    ax.plot(x_labels, y_vals, linestyle="-", marker="o", markersize=3)
    ax.set_title(titulo)
```

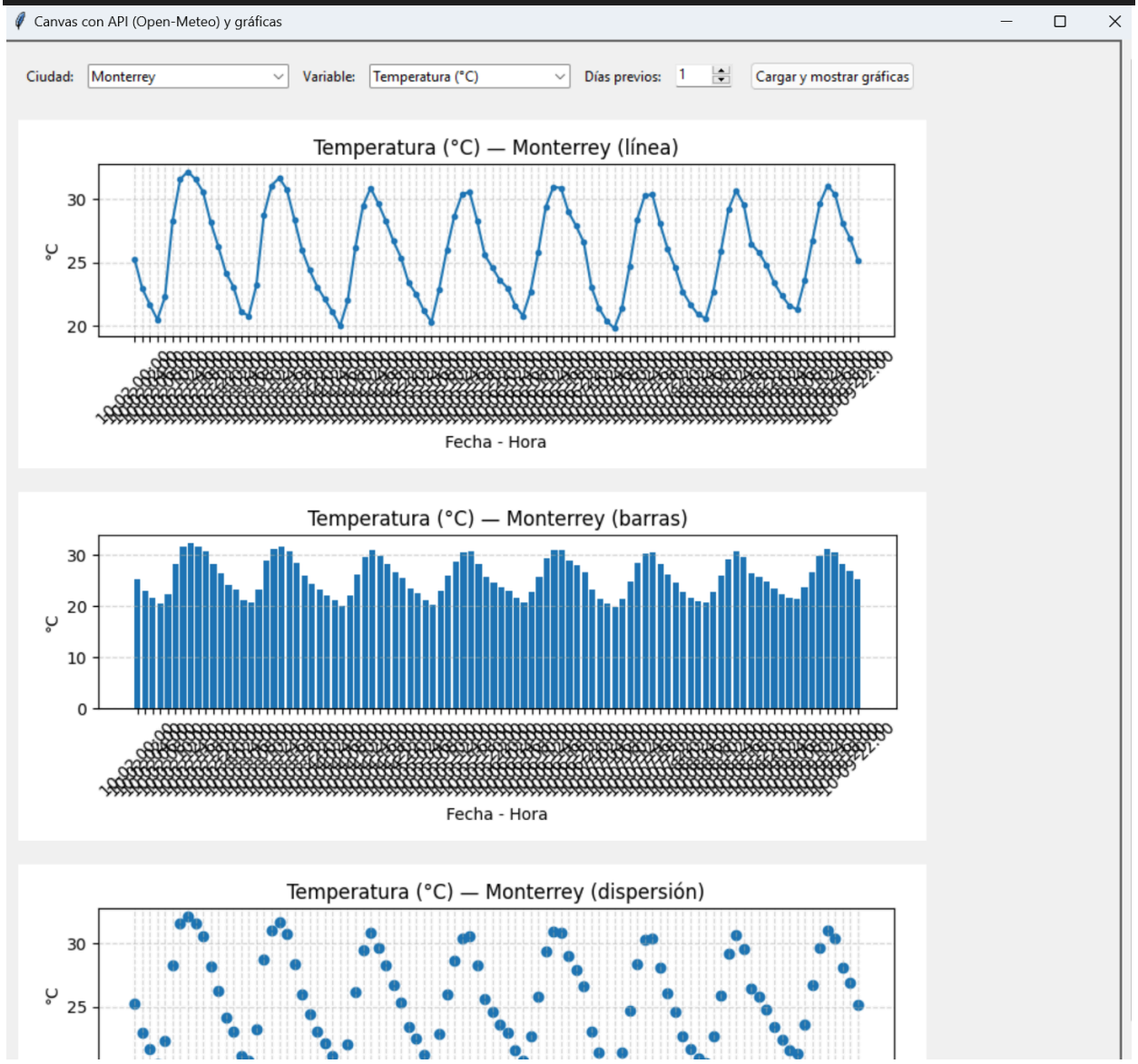
```
    ax.set_xlabel("Fecha - Hora")
    ax.set_ylabel(y_label)
    ax.grid(True, linestyle="--", alpha=.5)
    ax.tick_params(axis="x", rotation=45)
    fig.tight_layout()
    return fig

def create_bar_chart(x_labels, y_vals, titulo, y_label):
    fig, ax = plt.subplots(figsize=(7, 3))
    ax.bar(x_labels, y_vals)
    ax.set_title(titulo)
    ax.set_xlabel("Fecha - Hora")
    ax.set_ylabel(y_label)
    ax.grid(True, linestyle="--", alpha=.5, axis="y")
    ax.tick_params(axis="x", rotation=45)
    fig.tight_layout()
    return fig

def create_scatter_chart(x_labels, y_vals, titulo, y_label):
    fig, ax = plt.subplots(figsize=(7, 3))
    ax.scatter(x_labels, y_vals)
    ax.set_title(titulo)
    ax.set_xlabel("Fecha - Hora")
    ax.set_ylabel(y_label)
    ax.grid(True, linestyle="--", alpha=.5)
    ax.tick_params(axis="x", rotation=45)
    fig.tight_layout()
    return fig

# ----------------------------
# Utilidades UI
# ----------------------------
def _clear_frame_widgets(frm: ttk.Frame):
    """Elimina todo menos el header de controles (si existe)."""
    for w in frm.winfo_children():
        # Conserva el marco de controles si lo marcamos con atributo
        if getattr(w, "_is_controls", False):
            continue
        w.destroy()
```

```python
def _downsample(x, y, step=2):
    """Reduce densidad de etiquetas para que no se encimen."""
    if step <= 1:
        return x, y
    return x[::step], y[::step]

# ----------------------------
# Vista principal de la ventana
# ----------------------------
def open_win_canvas(parent: tk.Tk):
    """
    Crea la ventana secundaria con gráficas de la API, controles y scroll.
    """
    win = tk.Toplevel(parent)
    win.title("Canvas con API (Open-Meteo) y gráficas")
    win.geometry("980x1100")

    # Contenedor con scroll
    container = ttk.Frame(win)
    container.pack(fill="both", expand=True)

    canvas = tk.Canvas(container)
    scrollbar = ttk.Scrollbar(container, orient="vertical",
command=canvas.yview)
    scrollable = ttk.Frame(canvas, padding=12)

    scrollable.bind("<Configure>", lambda e:
canvas.configure(scrollregion=canvas.bbox("all")))
    canvas.create_window((0, 0), window=scrollable, anchor="nw")
    canvas.configure(yscrollcommand=scrollbar.set)

    canvas.pack(side="left", fill="both", expand=True)
    scrollbar.pack(side="right", fill="y")

    # ---------- Controles (ciudad, variable, past_days, botón) ----------
    controls = ttk.Frame(scrollable)
    controls._is_controls = True  # Marcador para no eliminarlo al
refrescar
    controls.pack(fill="x", pady=(0, 10))
```

```
    ttk.Label(controls, text="Ciudad:").grid(row=0, column=0, padx=5,
pady=5, sticky="w")
    cb_city = ttk.Combobox(controls, values=list(CITIES.keys()),
state="readonly", width=25)
    cb_city.set("León, Gto")
    cb_city.grid(row=0, column=1, padx=5, pady=5, sticky="w")

    ttk.Label(controls, text="Variable:").grid(row=0, column=2, padx=5,
pady=5, sticky="w")
    cb_var = ttk.Combobox(controls, values=list(VARIABLES.keys()),
state="readonly", width=25)
    cb_var.set("Temperatura (°C)")
    cb_var.grid(row=0, column=3, padx=5, pady=5, sticky="w")

    ttk.Label(controls, text="Días previos:").grid(row=0, column=4,
padx=5, pady=5, sticky="w")
    sp_days = ttk.Spinbox(controls, from_=0, to=7, width=5)
    sp_days.set("1")
    sp_days.grid(row=0, column=5, padx=5, pady=5, sticky="w")

    def cargar():
        _clear_frame_widgets(scrollable)

        city = cb_city.get()
        var_label = cb_var.get()
        try:
            days = int(sp_days.get())
        except Exception:
            days = 1

        lat, lon = CITIES[city]
        hourly_var = VARIABLES[var_label]

        x, y = fetch_data(lat, lon, hourly_var, past_days=days)
        if not x or not y:
            return

        # Si hay demasiados puntos, muestrea para etiquetas más limpias
(cada 2)
        step = 2 if len(x) > 36 else 1
```

```
        x_ds, y_ds = _downsample(x, y, step=step)

        # Etiquetas de ejes dependientes de variable
        y_axis = {
            "temperature_2m": "°C",
            "relativehumidity_2m": "%",
            "windspeed_10m": "m/s",
            "precipitation": "mm",
        }.get(hourly_var, "valor")

        titulo_base = f"{var_label} — {city}"

        # Línea
        fig1 = create_line_chart(x_ds, y_ds, f"{titulo_base} (línea)",
y_axis)
        cv1 = FigureCanvasTkAgg(fig1, master=scrollable); cv1.draw()
        cv1.get_tk_widget().pack(pady=10, fill="x")

        # Barras
        fig2 = create_bar_chart(x_ds, y_ds, f"{titulo_base} (barras)",
y_axis)
        cv2 = FigureCanvasTkAgg(fig2, master=scrollable); cv2.draw()
        cv2.get_tk_widget().pack(pady=10, fill="x")

        # Dispersión
        fig3 = create_scatter_chart(x_ds, y_ds, f"{titulo_base}
(dispersión)", y_axis)
        cv3 = FigureCanvasTkAgg(fig3, master=scrollable); cv3.draw()
        cv3.get_tk_widget().pack(pady=10, fill="x")

    ttk.Button(controls, text="Cargar y mostrar gráficas",
command=cargar).grid(
        row=0, column=6, padx=10, pady=5
    )

# ----------------------------
# Ejecución independiente
# ----------------------------
if __name__ == "__main__":
    root = tk.Tk()
```

```
    root.title("Prueba win_canvas")
    ttk.Button(root, text="Abrir ventana Canvas", command=lambda:
open_win_canvas(root)).pack(pady=20)
    root.mainloop()
```



En el código que recibimos originalmente solo se mostraban dos gráficas básicas (línea y barras) con datos fijos de León, Gto. Yo lo modifiqué para que ahora también incluya una tercera gráfica de dispersión, ofreciendo así tres formas distintas de visualizar la información. Además, añadí una función que limpia el frame antes de graficar de nuevo para evitar que se acumulen varias gráficas al presionar el botón varias veces. Otra mejora fue la integración de un scroll vertical, lo cual permite desplazarse cuando las gráficas no caben completas en la ventana. También mejoré la presentación de los datos al rotar las etiquetas de tiempo, aplicar

`tight_layout()` para que no se recorten títulos ni ejes, y activar rejillas para mayor claridad. Finalmente, incorporé controles interactivos (combobox y spinbox) que permiten elegir la ciudad, la variable meteorológica (temperatura, humedad relativa, viento o precipitación) y la cantidad de días previos a consultar, haciendo que el programa sea más dinámico, personalizable y fácil de usar para distintos escenarios.