```
def display_advanced_maps_tab(df_long, gdf_stations, **kwargs):
    """
    Versión Corrección Final:
    - Solución KeyError 'df_raw' (blindaje de memoria).
    - Solución Mapas Vacíos (Fallback inteligente si el recorte falla).
    - Restauración total de Popups, FDC y Textos.
    """
    import plotly.graph_objects as go
    import plotly.express as px
    from scipy.interpolate import Rbf, griddata
    import numpy as np
    from streamlit_folium import st_folium
    import folium
    from folium.plugins import LocateControl
    import geopandas as gpd
    import matplotlib
    import matplotlib.pyplot as plt
    from matplotlib import path as mpath
    from shapely.geometry import LineString
    import tempfile
    import os
    import shutil

    matplotlib.use('Agg')

    # Recuperar datos
    # 1. Recuperar Coberturas (Si no vienen en kwargs, intentar cargar por defecto)
    gdf_coberturas = kwargs.get("gdf_coberturas", None)
    if gdf_coberturas is None:
        try:
            # Ruta por defecto en tu carpeta data
            path_cob = "data/coberturas_antioquia.geojson" # Ajusta si tu archivo tiene otro nombre
            if os.path.exists(path_cob):
                gdf_coberturas = gpd.read_file(path_cob)
        except:
            pass # Si falla, se quedará como None y no mostrará esos mapas

    df_trends_global = kwargs.get("df_trends", None)

    # Configuración Títulos
    selected_months = kwargs.get("selected_months", [])
    titulo_meses = ""
    if selected_months and len(selected_months) < 12:
        nombres_meses = ["Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov", "Dic"]
        meses_str = ", ".join([nombres_meses[m - 1] for m in selected_months])
        titulo_meses = f" ({meses_str})"
```

```
    st.subheader(f"🌎 Superficies de Interpolación{titulo_meses} y Análisis Hidrológico")

    mode = st.radio("Modo de Análisis:", ["Regional (Comparación)", "Por Cuenca (Detallado)"],
horizontal=True)
    user_loc = kwargs.get("user_loc", None)

    # --- HELPERS ---
    # --- HELPER: INTERPOLACIÓN ROBUSTA (CON EXTRAPOLACIÓN) ---
    def run_interp(df_puntos, metodo, bounds_box):
        try:
            # 1. Extraer límites
            minx, maxx, miny, maxy = bounds_box

            # 2. Generar grilla con mayor resolución (1500x150)
            # Usamos números complejos (150j) para incluir el punto final exacto
            gx, gy = np.mgrid[minx:maxx:150j, miny:maxy:150j]

            # 3. Preparar datos
            df_unique = df_puntos.drop_duplicates(subset=["longitude", "latitude"])
            pts = df_unique[["longitude", "latitude"]].values
            vals = df_unique[Config.PRECIPITATION_COL].values

            if len(pts) < 3: return None, None, None

            # 4. Interpolación Principal
            if "Kriging" in metodo:
                try:
                    # Rbf (Thin Plate Spline) suele extrapolar bien suavemente
                    rbf = Rbf(pts[:, 0], pts[:, 1], vals, function="thin_plate")
                    gz = rbf(gx, gy)
                except:
                    gz = griddata(pts, vals, (gx, gy), method="linear")
            else:
                # IDW / Lineal genera NaNs afuera.
                gz = griddata(pts, vals, (gx, gy), method="linear")

            # 5. RELLENO DE BORDES (EXTRAPOLACIÓN) - EL SECRETO
            # Buscamos dónde quedó vacío (NaN) y lo llenamos con el vecino más cercano
            mask_nan = np.isnan(gz)
            if np.any(mask_nan):
                gz_nearest = griddata(pts, vals, (gx, gy), method="nearest")
                gz[mask_nan] = gz_nearest[mask_nan]

            return gx, gy, gz
```

```
        except Exception as e:
            print(f"Error interpolación: {e}")
            return None, None, None

    def calcular_promedios_reales(df_datos):
        if df_datos.empty: return pd.DataFrame()
        conteo = df_datos[df_datos[Config.PRECIPITATION_COL] >= 0].groupby([Config.STATION_NAME_COL,
Config.YEAR_COL]).size()
        anos_validos = conteo[conteo >= 5].index
        df_filtrado = df_datos.set_index([Config.STATION_NAME_COL,
Config.YEAR_COL]).loc[anos_validos].reset_index()
        suma_anual = df_filtrado.groupby([Config.STATION_NAME_COL,
Config.YEAR_COL])[Config.PRECIPITATION_COL].sum().reset_index()
        return
suma_anual.groupby(Config.STATION_NAME_COL)[Config.PRECIPITATION_COL].mean().reset_index()

    def generate_isohyets_gdf(gx, gy, gz, levels=10, crs="EPSG:4326"):
        try:
            fig, ax = plt.subplots()
            contour = ax.contour(gx, gy, gz, levels=levels)
            plt.close(fig)
            lines, values = [], []
            for collection in contour.collections:
                z_val = 0
                try: z_val = collection.level
                except: pass
                for path in collection.get_paths():
                    if len(path.vertices) >= 2:
                        lines.append(LineString(path.vertices))
                        values.append(z_val)
            if not lines: return None
            return gpd.GeoDataFrame({"valor": values, "geometry": lines}, crs=crs)
        except: return None

    def create_zipped_shapefile(gdf, filename_base):
        with tempfile.TemporaryDirectory() as tmpdir:
            path = os.path.join(tmpdir, f"{filename_base}.shp")
            gdf.to_file(path, driver="ESRI Shapefile")
            base_zip = os.path.join(tmpdir, filename_base)
            shutil.make_archive(base_zip, 'zip', tmpdir)
            with open(f"{base_zip}.zip", "rb") as f: return f.read()

    # --- HELPER MÁSCARA (MEJORADO PARA EVITAR MAPAS VACÍOS) ---
    # --- HELPER OPTIMIZADO: MÁSCARA RÁPIDA (CON CORRECCIÓN DE GIRO) ---
    def mask_grid_with_geometries(gx, gy, grid_values, gdf_mask):
        if gdf_mask is None or gdf_mask.empty or grid_values is None:
```

```
    return np.zeros_like(grid_values) if grid_values is not None else None

try:
    from rasterio import features
    from rasterio.transform import from_bounds

    # 1. Asegurar proyección Lat/Lon
    if gdf_mask.crs is None: gdf_mask.set_crs("EPSG:3116", inplace=True)
    if gdf_mask.crs.to_string() != "EPSG:4326":
        gdf_mask = gdf_mask.to_crs("EPSG:4326")

    # 2. Configurar dimensiones
    # OJO: grid_values viene de np.mgrid que es (X, Y)
    # Rasterio genera imágenes (Filas=Y, Columnas=X)
    rows, cols = gx.shape # Aquí rows es X, cols es Y en la lógica de mgrid

    minx, maxx = gx.min(), gx.max()
    miny, maxy = gy.min(), gy.max()

    # 3. Crear transformación
    # IMPORTANTE: Intercambiamos filas/cols en from_bounds para engañar a rasterio
    # y que genere la matriz en la orientación que coincide con mgrid (X, Y)
    transform = from_bounds(minx, miny, maxx, maxy, rows, cols)

    # 4. Rasterizar
    shapes = [(geom, 1) for geom in gdf_mask.geometry if not geom.is_empty]

    if not shapes: return np.zeros_like(grid_values)

    mask_arr = features.rasterize(
        shapes=shapes,
        out_shape=(rows, cols), # Dimensiones invertidas
        transform=transform,
        fill=0,
        default_value=1,
        dtype='uint8'
    )

    # 5. CORRECCIÓN DE GIRO (TRANSPOSE)
    # Como mgrid es (X, Y) y rasterio suele ser (Y, X), a veces quedan rotados.
    # Esta línea asegura que la máscara encaje si quedó rotada.
    if mask_arr.shape != grid_values.shape:
        mask_arr = mask_arr.T
    else:
        # Si las dimensiones son cuadradas (150x150), el shape no nos avisa.
        # Aplicamos la transpuesta porque sabemos que mgrid vs imagen siempre invierte ejes.
```

```
            mask_arr = mask_arr.T

            # 6. Aplicar máscara
            result = grid_values.copy()
            # Donde no hay máscara, ponemos NaN (transparente)
            result[mask_arr == 0] = np.nan

            return result

        except Exception as e:
            print(f"Error máscara rápida: {e}")
            return np.zeros_like(grid_values)

# ==============================================================================
    # MODO 1: REGIONAL (COMPARACIÓN) - CON DESCARGAS
    # ==============================================================================
    if mode == "Regional (Comparación)":
        st.markdown("#### 🆚 Comparación de Periodos Climáticos")
        st.info("Visualice cambios en el patrón de lluvias entre dos periodos.")

        c1, c2 = st.columns(2)
        with c1:
            st.markdown("###### Periodo 1 (Referencia)")
            r1 = st.slider("Rango P1:", 1980, 2024, (1990, 2000), key="r1")
            m1 = st.selectbox("Método P1:", ["Kriging (RBF)", "IDW (Lineal)", "Spline"], key="m1")
        with c2:
            st.markdown("###### Periodo 2 (Reciente)")
            r2 = st.slider("Rango P2:", 1980, 2024, (2010, 2020), key="r2")
            m2 = st.selectbox("Método P2:", ["Kriging (RBF)", "IDW (Lineal)", "Spline"], key="m2")

        # Botón de cálculo con PERSISTENCIA
        if st.button("🚀 Generar Comparación"):
            st.session_state["regional_done"] = True
            st.session_state["reg_params"] = {"r1": r1, "m1": m1, "r2": r2, "m2": m2}

        if st.session_state.get("regional_done"):
            p = st.session_state["reg_params"]

            # Función interna plot_panel corregida y con descargas
            def plot_panel(rng, meth, col, tag, u_loc):
                mask = (df_long[Config.YEAR_COL] >= rng[0]) & (df_long[Config.YEAR_COL] <= rng[1])
                df_sub = df_long[mask]
                df_avg = calcular_promedios_reales(df_sub)

                if df_avg.empty:
```

```
            col.warning(f"Sin datos válidos para {rng}")
            return

        if Config.STATION_NAME_COL not in df_avg.columns:
            df_avg = df_avg.reset_index()

        df_m = pd.merge(df_avg, gdf_stations, on=Config.STATION_NAME_COL).dropna(subset=["latitude",
"longitude"])

        if len(df_m) > 2:
            bounds = [
                df_m.longitude.min() - 0.1,
                df_m.longitude.max() + 0.1,
                df_m.latitude.min() - 0.1,
                df_m.latitude.max() + 0.1,
            ]
            gx, gy, gz = run_interp(df_m, meth, bounds)

            if gz is not None:
                # 1. Mapa Plotly (Isoyetas Visuales)
                fig = go.Figure(
                    go.Contour(
                        z=gz.T,
                        x=gx[:, 0],
                        y=gy[0, :],
                        colorscale="Viridis",
                        colorbar=dict(title="mm/año", len=0.5),
                        contours=dict(start=0, end=5000, size=200),
                    )
                )

                # Puntos Estaciones
                fig.add_trace(
                    go.Scatter(
                        x=df_m.longitude,
                        y=df_m.latitude,
                        mode="markers",
                        marker=dict(color="black", size=7, line=dict(width=1, color="white")),
                        text=df_m.apply(lambda x: f"<b>{x[Config.STATION_NAME_COL]}</b><br>Ppt:
{x[Config.PRECIPITATION_COL]:.0f} mm", axis=1),
                        hoverinfo="text",
                        showlegend=False,
                    )
                )

                # Capa Usuario
```

```
if u_loc:
    fig.add_trace(
        go.Scatter(
            x=[u_loc[1]],
            y=[u_loc[0]],
            mode="markers+text",
            marker=dict(color="red", size=12, symbol="star"),
            text=[" 📍 TÚ"],
            textposition="top center",
        )
    )

fig.update_layout(
    title=f"Ppt Media Anual ({rng[0]}-{rng[1]})",
    margin=dict(l=0, r=0, b=0, t=40),
    height=400,
)
col.plotly_chart(fig, use_container_width=True)

# 2. Generación de Vectores para Descarga
gdf_iso = generate_isohyets_gdf(gx, gy, gz, levels=12, crs=gdf_stations.crs)

# 3. Zona de Descargas
if gdf_iso is not None:
    with col.expander("💾 Descargar Capas (GIS)"):
        # GeoJSON (Rápido)
        col.download_button(
            label="🗺️ Descargar GeoJSON",
            data=gdf_iso.to_json(),
            file_name=f"isoyetas_{tag}_{rng[0]}_{rng[1]}.geojson",
            mime="application/json"
        )

        # Shapefile (Manual para evitar bloqueos)
        if col.button(f"📦 Generar Shapefile ({tag})", key=f"btn_shp_{tag}"):
            try:
                zip_shp = create_zipped_shapefile(gdf_iso, f"isoyetas_{tag}")
                col.download_button(
                    label="📥 Descargar ZIP (.shp)",
                    data=zip_shp,
                    file_name=f"isoyetas_{tag}_{rng[0]}_{rng[1]}.zip",
                    mime="application/zip"
                )
            except Exception as e:
                col.error(f"Error generando ZIP: {e}")
```

```
        # 4. Mapa Interactivo (Folium) con Popups
        with col.expander(f"🔎 Ver Mapa Interactivo Detallado ({tag})", expanded=True):
            col.write("Mapa navegable con detalles por estación.")
            center_lat = (bounds[2] + bounds[3]) / 2
            center_lon = (bounds[0] + bounds[1]) / 2
            m = folium.Map(
                location=[center_lat, center_lon],
                zoom_start=8,
                tiles="CartoDB positron",
            )

            # Añadimos isoyetas al folium también para referencia
            if gdf_iso is not None:
                folium.GeoJson(
                    gdf_iso,
                    name="Isoyetas",
                    style_function=lambda x: {'color': '#2c3e50', 'weight': 1, 'dashArray': '5, 5'}
                ).add_to(m)

            for _, row in df_m.iterrows():
                nombre = row[Config.STATION_NAME_COL]
                lluvia = row[Config.PRECIPITATION_COL]
                html = f"<b>{nombre}</b><br>{lluvia:.0f} mm"
                folium.CircleMarker(
                    [row["latitude"], row["longitude"]],
                    radius=6,
                    color="blue",
                    fill=True,
                    fill_color="cyan",
                    fill_opacity=0.9,
                    tooltip=html,
                ).add_to(m)

            LocateControl(auto_start=False).add_to(m)
            st_folium(
                m,
                height=350,
                use_container_width=True,
                key=f"folium_comp_{tag}",
            )

# Renderizar paneles
plot_panel(p["r1"], p["m1"], c1, "A", user_loc)
plot_panel(p["r2"], p["m2"], c2, "B", user_loc)
```

```
    # ================================================================================
    # MODO 2: CUENCA (COMPLETO E INTEGRADO)
    # ================================================================================
    else:
        gdf_subcuencas = kwargs.get("gdf_subcuencas")
        if gdf_subcuencas is None or gdf_subcuencas.empty:
            st.warning(" ⚠️ No se ha cargado la capa de Cuencas.")
            return

        col_name = next((c for c in gdf_subcuencas.columns if "nombre" in c.lower() or "cuenca" in c.lower()),
gdf_subcuencas.columns[0])
        default_cuencas = st.session_state.get("last_sel_cuencas", [])
        avail_opts = sorted(gdf_subcuencas[col_name].unique().astype(str))
        valid_defaults = [x for x in default_cuencas if x in avail_opts]
        sel_cuencas = st.multiselect("Seleccionar Cuenca(s):", avail_opts, default=valid_defaults)

        with st.expander(" ⚙️ Configuración Avanzada", expanded=False):
            buffer_km = st.slider("Radio búsqueda (km):", 0, 50, 15, step=5)

        if sel_cuencas:
            c_p1, c_p2 = st.columns(2)
            rng_c = c_p1.slider("Periodo:", 1980, 2025, (2000, 2020))
            meth_c = c_p2.selectbox("Método Interpolación:", ["Kriging (RBF)", "IDW"])

            # --- LÓGICA DE CÁLCULO ---
            if st.button(" ⚡ Analizar Cuenca"):
                st.session_state["last_sel_cuencas"] = sel_cuencas
                if "basin_res" in st.session_state: del st.session_state["basin_res"]

                with st.spinner("Procesando hidrología, IVC y tendencias..."):
                    # 1. Geometría
                    sub = gdf_subcuencas[gdf_subcuencas[col_name].isin(sel_cuencas)]
                    try:
                        geom_union = sub.geometry.unary_union
                        gdf_union = gpd.GeoDataFrame({"geometry": [geom_union]}, crs=gdf_subcuencas.crs)
                    except:
                        gdf_union = gpd.GeoDataFrame({"geometry": [sub.geometry.iloc[0]]}, crs=gdf_subcuencas.crs)
                        geom_union = sub.geometry.iloc[0]

                    gdf_vis = gdf_union.copy()
                    gdf_vis["geometry"] = gdf_vis.geometry.simplify(0.005)

                    buf = geom_union.buffer(buffer_km / 111.32)
                    gdf_buf = gpd.GeoDataFrame({"geometry": [buf]}, crs=gdf_stations.crs)
                    stns = gpd.sjoin(gdf_stations, gdf_buf, predicate="intersects")
```

```
            if not stns.empty:
                # 2. Datos
                mask = (df_long[Config.STATION_NAME_COL].isin(stns[Config.STATION_NAME_COL].unique()))
& \
                    (df_long[Config.YEAR_COL] >= rng_c[0]) & (df_long[Config.YEAR_COL] <= rng_c[1])
                df_raw = df_long[mask].copy()
                df_avg = calcular_promedios_reales(df_raw)

                if Config.STATION_NAME_COL not in df_avg.columns: df_avg = df_avg.reset_index()
                df_int = pd.merge(df_avg, gdf_stations,
on=Config.STATION_NAME_COL).dropna(subset=["latitude", "longitude"])
                if Config.ALTITUDE_COL not in df_int.columns: df_int[Config.ALTITUDE_COL] = 1500
                gdf_pts = gpd.GeoDataFrame(df_int, geometry=gpd.points_from_xy(df_int.longitude,
df_int.latitude), crs=gdf_stations.crs)

                if len(df_int) >= 3:
                    # 3. Malla Base e Interpolación (CORREGIDO)
                    # Usamos los límites del buffer para asegurar cobertura total
                    minx, miny, maxx, maxy = gdf_buf.total_bounds

                    # A. Interpolación de PRECIPITACIÓN (gz)
                    # Llamamos al helper run_interp que ya incluye Extrapolación (relleno de bordes)
                    gx, gy, gz = run_interp(df_int, meth_c, [minx, maxx, miny, maxy])

                    # B. Interpolación de ALTITUD (gz_alt)
                    # Necesaria para el cálculo de IVC. La hacemos sobre la misma malla gx, gy.
                    gz_alt = None
                    if gx is not None:
                        # Definimos los puntos y valores aquí para evitar errores de variables no definidas
                        pts = df_int[["longitude", "latitude"]].values
                        vals_alt = df_int[Config.ALTITUDE_COL].values

                        # 1. Interpolación Lineal (precisa adentro)
                        gz_alt = griddata(pts, vals_alt, (gx, gy), method='linear')

                        # 2. Extrapolación Nearest (para rellenar los bordes vacíos NaN)
                        mask_nan_alt = np.isnan(gz_alt)
                        if np.any(mask_nan_alt):
                            gz_alt_near = griddata(pts, vals_alt, (gx, gy), method='nearest')
                            gz_alt[mask_nan_alt] = gz_alt_near[mask_nan_alt]

                    # B. IVC (Física y Normalización)
                    gz_t = np.maximum(28 - (0.006 * gz_alt), 0)
                    I_t = 300 + (25 * gz_t) + (0.05 * gz_t**3)
                    with np.errstate(divide='ignore', invalid='ignore'):
```

```
                    gz_etr = gz / np.sqrt(0.9 + (gz / l_t)**2)
                gz_etr = np.minimum(gz_etr, gz)
                gz_esd = gz - gz_etr

                t_max = np.nanmax(gz_t)
                gz_it = 100 * (gz_t / t_max) if t_max > 0 else gz_t * 0
                esd_max = np.nanmax(gz_esd)
                if esd_max <= 0: esd_max = 1
                gz_iesd = 100 * ((esd_max - gz_esd) / esd_max)
                gz_ivc = (gz_it + gz_iesd) / 2

                # C. Tendencias (In-Situ Backup)
                gz_iv_var = None
                df_slopes = df_trends_global # Intentar usar el global

                # Si no hay global, calcular localmente
                if df_slopes is None and len(df_raw) > 0:
                    try:
                        import scipy.stats
                        slopes = []
                        for s in df_int[Config.STATION_NAME_COL].unique():
                            d = df_raw[df_raw[Config.STATION_NAME_COL] ==
s].groupby(Config.YEAR_COL)[Config.PRECIPITATION_COL].sum()
                            if len(d) > 5:
                                sl, _, _, _, _ = scipy.stats.linregress(d.index, d.values)
                                slopes.append({Config.STATION_NAME_COL: s, 'slope': sl})
                        if slopes: df_slopes = pd.DataFrame(slopes)
                    except: pass

                if df_slopes is not None:
                    try:
                        df_tm = pd.merge(df_int, df_slopes, on=Config.STATION_NAME_COL, how="left")
                        if 'slope' in df_tm.columns:
                            gz_slope = griddata(pts, df_tm['slope'].fillna(0).values, (gx, gy), method='linear')
                            gz_tr = np.clip(50 - (gz_slope * 25), 0, 100)
                            gz_iv_var = (gz_iesd + gz_tr) / 2
                    except: pass

                # D. Recorte (Masking) para Cultivos e Incendios (CÓDIGO SIMPLIFICADO)
                # Inicializamos con ceros por defecto
                gz_cult = np.zeros_like(gz_ivc)
                gz_inc = np.zeros_like(gz_ivc)

                if gdf_coberturas is not None:
                    try:
                        # Identificar columna de texto
```

```
                txt_cols = gdf_coberturas.select_dtypes(include=['object']).columns
                if len(txt_cols) > 0:
                    c = txt_cols[0]
                    # Filtros flexibles (case-insensitive)
                    gdf_a = gdf_coberturas[gdf_coberturas[c].astype(str).str.contains("Cult|Past|Agri",
case=False, na=False)]
                    gdf_b = gdf_coberturas[gdf_coberturas[c].astype(str).str.contains("Bosq|For|Selv",
case=False, na=False)]

                    # Aplicar la nueva función de máscara corregida
                    if not gdf_a.empty:
                        # gz_cult tendrá valores donde hay cultivo, y NaN donde no
                        gz_cult = mask_grid_with_geometries(gx, gy, gz_ivc, gdf_a)

                    if not gdf_b.empty:
                        # gz_inc tendrá valores donde hay bosque, y NaN donde no
                        gz_inc = mask_grid_with_geometries(gx, gy, gz_ivc, gdf_b)

            except Exception as e:
                print(f"Error en filtrado de coberturas: {e}")

        # 5. Hidrología Completa (ORDEN CORREGIDO)
        # A. Primero generamos isoyetas (para que exista gdf_iso)
        gdf_iso = generate_isohyets_gdf(gx, gy, gz, levels=12, crs=gdf_stations.crs)

        # B. Calculamos precipitación media
        ppt_med = np.nanmean(gz) if gz is not None else 0

        # C. Morfometría
        try: morph = analysis.calculate_morphometry(gdf_union)
        except: morph = {"area_km2": 0, "alt_prom_m": 1500}
        area_km2 = morph.get("area_km2", 100)

        # D. Balance de Turc y Caudales
        tm = max(0, 28 - 0.006 * morph.get("alt_prom_m", 1500))
        try: _, q_mm = analysis.calculate_water_balance_turc(ppt_med, tm)
        except: q_mm = 0

        vol_hm3 = (q_mm * area_km2) / 1000
        q_m3s = (vol_hm3 * 1_000_000) / 31536000

        # --- INTERVENCIÓN 2: CÁLCULO DE CURVAS (CORREGIDO) ---

        # A. CURVA HIPSOMÉTRICA (Usando DEM Real)
        hyp_data = None
        try:
```

```
                    # Pasamos la geometría Y la ruta del archivo DEM definida en Config
                    hyp_data = analysis.calculate_hypsometric_curve(
                        gdf_union,
                        dem_path=getattr(Config, "DEM_FILE_PATH", None) # Usa None si no está en Config
                    )
                except Exception as e:
                    print(f"Error hipsometría: {e}")

                # B. CURVA DE DURACIÓN DE CAUDALES (FDC)
                fdc_data = None
                try:
                    # 1. Serie de Precipitación Mensual de la Cuenca
                    ppt_series = df_raw.groupby(Config.DATE_COL)[Config.PRECIPITATION_COL].mean()

                    # 2. Coeficiente de Escorrentía (Estimado simple)
                    # Si hay mucha lluvia (>2000mm) asumimos mayor escorrentía (0.5), si no 0.3
                    c_est = 0.5 if ppt_med > 2000 else 0.3

                    # 3. Calcular usando la función de analysis.py
                    fdc_data = analysis.calculate_duration_curve(
                        ppt_series,
                        runoff_coeff=c_est,
                        area_km2=area_km2
                    )
                except Exception as e:
                    print(f"Error FDC: {e}")

                # E. Índices Climáticos
                idx_c = {}
                try:
                    idx_c = analysis.calculate_climatic_indices(
                        df_raw.groupby(Config.DATE_COL)[Config.PRECIPITATION_COL].mean(),
                        morph.get("alt_prom_m", 1500)
                    )
                except: pass

                # GUARDADO FINAL EN SESSION STATE
                st.session_state["basin_res"] = {
                    "ready": True, "names": ", ".join(sel_cuencas), "bounds": [minx, maxx, miny, maxy],
                    "gz": gz, "gx": gx, "gy": gy, "gz_ivc": gz_ivc, "gz_iv_var": gz_iv_var,
                    "gz_cult": gz_cult, "gz_inc": gz_inc,
                    "gdf_union": gdf_union, "gdf_vis": gdf_vis, "gdf_pts": gdf_pts, "gdf_buf": gdf_buf,
"gdf_iso": gdf_iso,
                    "df_int": df_int, "df_raw": df_raw,
                    "bal": {"P": ppt_med, "Q": q_mm, "Q_m3s": q_m3s, "Vol": vol_hm3},
                    "morph": morph,
```

```
                    "idx": idx_c,
                    "fdc": fdc_data,     # <--- Guardamos el resultado limpio
                    "hyp": hyp_data      # <--- Guardamos el resultado limpio
                }
            else: st.error("Insuficientes estaciones (<3).")
        else: st.error("Sin estaciones cercanas.")

    # --- VISUALIZACIÓN ---
    res = st.session_state.get("basin_res")
    if res and res.get("ready"):
        st.markdown(f"##### 📊 Resultados: {res['names']}")

        opts = ["Precipitación (Isoyetas)", "IVC (Vulnerabilidad Climática)", "Vulnerabilidad Variación Clima
(ESD + Tendencias)"]
        if gdf_coberturas is not None:
            opts += ["Vulnerabilidad Cultivos (IVC + Agric)", "Vulnerabilidad Incendios (IVC + Bosque)"]

        sel = st.selectbox("Seleccionar Mapa Temático:", opts)

        z, tit, colors = res["gz"], "Precipitación (mm)", "Blues"
        zmin, zmax = 0, np.nanmax(z)

        if "IVC" in sel:
            colors = "RdYlGn_r"
            zmin, zmax = 0, 100
            if sel == "IVC (Vulnerabilidad Climática)":
                z, tit = res["gz_ivc"], "IVC Global"
            elif "Cultivos" in sel:
                z, tit = res.get("gz_cult", res["gz_ivc"]), "Amenaza en Cultivos (Recortado)"
            elif "Incendios" in sel:
                z, tit = res.get("gz_inc", res["gz_ivc"]), "Amenaza Incendios (Recortado)"
        elif "Variación" in sel:
            z, tit, colors = res.get("gz_iv_var", res["gz_ivc"]), "Vulnerabilidad Variación (Tendencia)",
"RdYlGn_r"
            zmin, zmax = 0, 100

        # Mapa Principal Plotly
        # --- CORRECCIÓN 1: BLINDAJE CONTRA MAPAS VACÍOS ---

        # Verificamos si z tiene datos antes de intentar graficar
        if z is not None:
            # Mapa Principal Plotly
            try:
                # Nota: z.T es la transpuesta, necesaria porque Plotly interpreta x/y distinto a numpy
                fig = go.Figure(go.Contour(
                    z=z.T,
```

```
                    x=res["gx"][:,0],
                    y=res["gy"][0,:],
                    colorscale=colors,
                    zmin=zmin,
                    zmax=zmax,
                    contours=dict(start=zmin, end=zmax, size=(zmax-zmin)/15 if zmax>zmin else 1),
                    colorbar=dict(title=tit)
                ))

                # Agregar contorno de la cuenca (Si existe)
                try:
                    # Manejo robusto de geometrías (Polygon vs MultiPolygon)
                    geoms = res["gdf_vis"].geometry
                    if not geoms.empty:
                        geom_list = geoms.iloc[0].geoms if geoms.iloc[0].geom_type == 'MultiPolygon' else
[geoms.iloc[0]]
                        for g in geom_list:
                            xs, ys = g.exterior.xy
                            fig.add_trace(go.Scatter(x=list(xs), y=list(ys), mode="lines", line=dict(color="black",
width=2), name="Cuenca"))
                except Exception as e:
                    print(f"No se pudo dibujar el contorno de la cuenca: {e}")

                # Agregar puntos de estaciones
                fig.add_trace(go.Scatter(
                    x=res["gdf_pts"].geometry.x,
                    y=res["gdf_pts"].geometry.y,
                    mode="markers",
                    marker=dict(color="black", size=4),
                    name="Estaciones"
                ))

                fig.update_layout(title=tit, height=500, margin=dict(l=20, r=20, t=40, b=20))
                st.plotly_chart(fig, use_container_width=True)

            except Exception as e:
                st.error(f"Error al renderizar el gráfico: {e}")

        else:
            # Si z es None, mostramos una advertencia amigable en lugar de crashear
            st.warning(f" ⚠️ No se pudieron generar datos para: **{sel}**.")
            st.info("Posible causa: La capa de cobertura (cultivos/bosques) no se superpone con la zona
seleccionada o hay un error de proyección.")

        if "IVC" in sel or "Variación" in sel:
```

```
            with st.expander("ℹ️ Interpretación del IVC (Semáforo de Riesgo)", expanded=True):
                st.markdown("""
                El **Índice de Vulnerabilidad Climática (IVC)** identifica zonas críticas por condiciones biofísicas:
                * 🔴 **Rojo (80-100): Vulnerabilidad Crítica.** Coincidencia de altas temperaturas y bajo balance hídrico (déficit).
                * 🟠 **Naranja (60-80): Vulnerabilidad Alta.**
                * 🟡 **Amarillo (40-60): Vulnerabilidad Media.**
                * 🟢 **Verde (0-40): Vulnerabilidad Baja.** Zonas con superávit hídrico y temperaturas moderadas.
                """)

            # Descargas
            col_d1, col_d2 = st.columns(2)
            with col_d1:
                if st.button("📥 Generar GeoJSON"):
                    gdf_out = generate_isohyets_gdf(res["gx"], res["gy"], z, levels=15)
                    if gdf_out is not None: col_d1.download_button("Descargar", gdf_out.to_json(), "mapa.geojson")

            # Métricas y Balance (Restaurado)
            st.markdown("---")
            st.subheader("💧 Balance Hídrico y Morfometría")
            m, b = res["morph"], res["bal"]
            c1, c2, c3, c4 = st.columns(4)
            c1.metric("Área", f"{m.get('area_km2', 0):.1f} km²")
            c2.metric("Altitud Media", f"{m.get('alt_prom_m', 0):.0f} m")
            c3.metric("Ppt Media", f"{b.get('P', 0):.0f} mm")
            c4.metric("Caudal (Q)", f"{(b.get('Q', 0)*m.get('area_km2',0)/1000 * 1e6 / 31536000):.2f} m³/s")

            with st.expander("ℹ️ Detalle Metodológico: Balance de Turc"):
                st.markdown("""
                **Fórmula de Turc (1954):** Estima la escorrentía anual ($Q$) en función de la Precipitación ($P$) y Temperatura ($T$).
                $$ E = \\frac{P}{\\sqrt{0.9 + \\frac{P^2}{L(T)^2}}} \\quad \\text{donde} \\quad L(T) = 300 + 25T + 0.05T^3 $$
                El caudal específico se deriva como $Q = P - E$. Es ideal para estimar oferta hídrica media a largo plazo.
                """)

            # ÍNDICES, FDC Y HIPSOMETRÍA COMPLETOS

            # A. ÍNDICES CLIMÁTICOS (Con Interpretación Numérica)
            st.markdown("---")
            st.subheader("🌡️ Índices Climáticos")
```

```
            idx = res["idx"]

            c_idx1, c_idx2 = st.columns(2)
            with c_idx1:
                st.metric("Aridez (Martonne)", f"{idx.get('martonne_val', 0):.1f}",
delta=idx.get("martonne_class", ""))
            with c_idx2:
                st.metric("Erosividad (Fournier)", f"{idx.get('fournier_val', 0):.1f}", delta=idx.get("fournier_class",
""))

            with st.expander("ℹ️ Interpretación y Rangos Numéricos", expanded=False):
                st.markdown("""
                **1. Índice de Aridez de Martonne ($I_M$):**
                $$ I_M = \\frac{P}{T + 10} $$
                * **0 - 10:** Desértico / Árido 🔴
                * **10 - 20:** Semiárido 🟠
                * **20 - 30:** Mediterráneo 🟡
                * **30 - 60:** Húmedo 🟢
                * **> 60:** Perhúmedo 🔵

                **2. Índice de Fournier ($I_F$):**
                $$ I_F = \\sum \\frac{p_i^2}{P} $$
                * **< 60:** Erosividad Muy Baja 🟢
                * **60 - 90:** Moderada 🟡
                * **90 - 120:** Alta 🟠
                * **> 120:** Muy Alta (Riesgo Erosión) 🔴
                """)

            # B. CURVA DE DURACIÓN DE CAUDALES (FDC) - Restaurada
            if res.get("fdc"):
                st.markdown("---")
                st.subheader("📉 Curva de Duración de Caudales (FDC)")

                # Recuperar datos
                fdc_data = res["fdc"]
                df_fdc = fdc_data.get("data") if isinstance(fdc_data, dict) else fdc_data

                if df_fdc is not None and not df_fdc.empty:
                    col_f1, col_f2 = st.columns([3, 1])
                    with col_f1:
                        # Gráfico de Área
                        fig_f = go.Figure()
                        fig_f.add_trace(go.Scatter(
                            x=df_fdc["Probabilidad Excedencia (%)"],
                            y=df_fdc["Caudal (m³/s)"],
```

```
                    fill='tozeroy',
                    mode='lines',
                    line=dict(color='#3498db')
                ))
                fig_f.update_layout(
                    xaxis_title="Probabilidad Excedencia (%)",
                    yaxis_title="Caudal (m³/s)",
                    margin=dict(l=20, r=20, t=20, b=20),
                    height=350
                )
                st.plotly_chart(fig_f, use_container_width=True)

            with col_f2:
                st.markdown("**Ecuación Ajustada:**")
                if isinstance(fdc_data, dict) and "equation" in fdc_data:
                    eq_latex = fdc_data["equation"].replace("P", "P_{exc}")
                    st.latex(eq_latex)
                    st.caption(f"$R^2 = {fdc_data.get('r_squared', 0):.4f}$")
                else:
                    st.info("Ecuación no disponible")

            with st.expander("ℹ️ Metodología e Interpretación FDC"):
                st.markdown("""
                **Definición:** La curva FDC representa la relación entre la magnitud del caudal y la frecuencia con la que es superado.

                **Metodología:**
                1. Se genera la serie de caudales medios diarios/mensuales a partir del balance hídrico ($Q = P - ETR$).
                2. Se ordenan los datos de mayor a menor.
                3. Se calcula la probabilidad: $P(\%) = \\frac{m}{n+1} \\times 100$.

                **Interpretación:**
                * **Q95 (Caudal Ecológico):** Caudal superado el 95% del tiempo. Indica la oferta hídrica en estiaje.
                * **Pendiente:** Una curva con mucha pendiente indica una cuenca con respuesta rápida y poca regulación (acuíferos pobres).
                """)

        # C. CURVA HIPSOMÉTRICA - Restaurada
        if hasattr(analysis, "calculate_hypsometric_curve"):
            try:
                hyp = analysis.calculate_hypsometric_curve(res["gdf_cuenca"])
                if hyp:
                    st.markdown("---")
```

```
                st.subheader("🏔️ Curva Hipsométrica")

                col_h1, col_h2 = st.columns([3, 1])
                with col_h1:
                  fig_h = go.Figure()
                  fig_h.add_trace(go.Scatter(
                    x=hyp["area_percent"],
                    y=hyp["elevations"],
                    fill='tozeroy',
                    line=dict(color='green')
                  ))
                  fig_h.update_layout(
                    xaxis_title="% Área Acumulada sobre la cota",
                    yaxis_title="Elevación (msnm)",
                    margin=dict(l=20, r=20, t=20, b=20),
                    height=350
                  )
                  st.plotly_chart(fig_h, use_container_width=True)

                with col_h2:
                  st.markdown("**Modelo Altitudinal:**")
                  if hyp.get("equation"):
                    # Limpiamos la ecuación para LaTeX
                    eq_clean = hyp["equation"].replace("x", "A")
                    st.latex(eq_clean)

                with st.expander("ℹ️ Interpretación Geomorfológica"):
                  st.markdown("""
                  **Concepto:** Muestra la distribución del área de la cuenca en función de la altura.

                  **Interpretación (Integral Hipsométrica):**
                  * **Curva Convexa (Integral > 0.6):** Cuenca en fase de **Juventud**. Gran potencial erosivo, laderas inestables.
                  * **Curva en 'S' (Integral 0.35 - 0.60):** Cuenca en fase de **Madurez**. Equilibrio entre erosión y sedimentación.
                  * **Curva Cóncava (Integral < 0.35):** Cuenca en fase de **Vejez**. Predomina la sedimentación, relieves suaves.
                  """)
              except Exception as e:
                pass # Silenciar errores no críticos aquí

            # Mapa de Contexto (Restaurado con Popups)
            st.markdown("---")
            st.subheader("📍 Mapa de Contexto")
            if "bounds" in res:
```

```
            m_ctx = folium.Map([(res["bounds"][2]+res["bounds"][3])/2,
(res["bounds"][0]+res["bounds"][1])/2], zoom_start=10, tiles="CartoDB positron")
            if res.get("gdf_buf") is not None:
                folium.GeoJson(res["gdf_buf"], style_function=lambda x: {"color": "gray", "dashArray": "5,5",
"fill": False}).add_to(m_ctx)
            if "gdf_vis" in res:
                folium.GeoJson(res["gdf_vis"], style_function=lambda x: {"color": "blue", "weight": 2,
"fillOpacity": 0.1}).add_to(m_ctx)

            # --- POPUPS DETALLADOS Y SEGUROS ---
            # 1. Pre-cálculo seguro de años de registro (Evita KeyError)
            df_raw_safe = res.get("df_raw", pd.DataFrame())
            if not df_raw_safe.empty:
                years_per_station =
df_raw_safe.groupby(Config.STATION_NAME_COL)[Config.YEAR_COL].nunique()
            else:
                years_per_station = {}

            # 2. Iteración sobre estaciones interpoladas
            for _, r in res["df_int"].iterrows():
                nombre = r[Config.STATION_NAME_COL]
                ppt_val = r[Config.PRECIPITATION_COL]

                # 3. Recuperación inteligente de Municipio
                # Convertimos índices a string y buscamos 'muni' o 'ciud' sin importar mayúsculas
                cols_r = [str(c) for c in r.index.tolist()]
                col_muni = next((c for c in cols_r if "muni" in c.lower() or "municipio" in c.lower()), None)
                muni_val = r[col_muni] if col_muni else "N/A"

                # 4. Recuperación de Años y Altitud
                n_anos = years_per_station.get(nombre, 0) if isinstance(years_per_station, dict) else
years_per_station.get(nombre, 0)
                alt_val = r.get(Config.ALTITUDE_COL, 0)

                # 5. Construcción del HTML
                html_content = f"""
                <div style='font-family:sans-serif; font-size:12px; min-width:150px'>
                    <h5 style='margin:0; color:#2c3e50; border-bottom:1px solid #ccc; padding-
bottom:3px'>{nombre}</h5>
                    <div style='margin-top:5px'>
                        📍 <b>Mun:</b> {muni_val}<br>
                        ⛰️ <b>Alt:</b> {alt_val:.0f} msnm<br>
                        🌧️ <b>Ppt:</b> {ppt_val:.0f} mm<br>
                        📅 <b>Reg:</b> {n_anos} años
                    </div>
```

```
            </div>
            """

            # 6. Creación del Popup y Marcador
            iframe = folium.IFrame(html_content, width=220, height=130)
            popup = folium.Popup(iframe, max_width=220)

            folium.CircleMarker(
                [r.latitude, r.longitude],
                radius=5,
                color="darkred",
                fill=True,
                fill_color="#e74c3c",
                fill_opacity=0.9,
                popup=popup
            ).add_to(m_ctx)

        st_folium(m_ctx, height=450, width="100%")

# --- INTERVENCIÓN 3: VISUALIZACIÓN CORREGIDA (CON PROTECCIÓN DE ERROR) ---

# 1. Intentamos recuperar los resultados
res = st.session_state.get("basin_res")

# 2. VERIFICACIÓN CRÍTICA: Solo intentamos dibujar si 'res' tiene datos
if res is not None:

    st.markdown("---")
    st.subheader("📊 Análisis Hidrológico Detallado")

    # Recuperamos los resultados del diccionario de forma segura
    hyp = res.get("hyp")
    fdc = res.get("fdc")

    col_curvas_1, col_curvas_2 = st.columns(2)

    # --- A. VISUALIZACIÓN CURVA HIPSOMÉTRICA ---
    with col_curvas_1:
        st.markdown("**⛰️ Curva Hipsométrica**")
        if hyp:
            import plotly.graph_objects as go # Aseguramos importación

            fig_h = go.Figure()
            fig_h.add_trace(go.Scatter(
                x=hyp["area_percent"],
```

```
            y=hyp["elevations"],
            fill='tozeroy',
            mode='lines',
            line=dict(color='#2E86C1'),
            name='Terreno'
        ))
        fig_h.update_layout(
            xaxis_title="% Área Acumulada",
            yaxis_title="Elevación (msnm)",
            height=300,
            margin=dict(l=0,r=0,t=30,b=0),
            hovermode="x unified"
        )
        st.plotly_chart(fig_h, use_container_width=True)

        st.caption(f"📐 **Modelo:** `${hyp.get('equation', 'N/A')}$`")
        if hyp.get("source") == "Simulado":
            st.caption("⚠️ *Datos simulados (DEM no detectado)*")
    else:
        st.warning("⚠️ Datos de elevación no disponibles.")

# --- B. VISUALIZACIÓN CURVA FDC ---
with col_curvas_2:
    st.markdown("**🌊 Curva de Duración de Caudales (FDC)**")
    if fdc and fdc.get("data") is not None:
        df_fdc = fdc["data"]

        fig_f = go.Figure()
        fig_f.add_trace(go.Scatter(
            x=df_fdc["Probabilidad Excedencia (%)"],
            y=df_fdc["Caudal (m³/s)"],
            mode='lines',
            line=dict(color='#27AE60', width=2),
            name='Caudal'
        ))
        fig_f.update_layout(
            xaxis_title="% Tiempo excedencia",
            yaxis_title="Caudal (m³/s)",
            yaxis_type="log",
            height=300,
            margin=dict(l=0,r=0,t=30,b=0)
        )
        st.plotly_chart(fig_f, use_container_width=True)

        try:
```

```
            q_vals = df_fdc["Caudal (m³/s)"].values
            q95 = np.percentile(q_vals, 5)
            st.caption(f"🐟 **Caudal Ecológico (Q95):** {q95:.2f} m³/s")
            st.caption(f"📐 **R² Ajuste:** {fdc.get('r_squared', 0):.2f}")
        except: pass
    else:
        st.info("⚠️ Faltan datos de lluvia para generar la curva FDC.")

else:
    # Si res es None (aún no se ha analizado nada), mostramos esto:
    st.info("👈 Seleccione una cuenca y haga clic en '⚡ Analizar Cuenca' para ver los detalles.")
```