```python
# modules/interpolation.py

import gstools as gs
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import streamlit as st
from scipy.interpolate import griddata
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import LeaveOneOut

from modules.config import Config


# ---------------------------------------------------------------------------
# FUNCIÓN _perform_loocv (ESTA ES LA FUNCIÓN QUE FALTABA)
# ---------------------------------------------------------------------------
def _perform_loocv(method, lons, lats, vals, elevs=None):
    """
    Función auxiliar interna que realiza el cálculo de Leave-One-Out Cross-Validation.
    """
    loo = LeaveOneOut()
    y_true = []
    y_pred = []

    if len(vals) < 4:
        # No se puede interpolar con menos de 4 puntos de manera fiable
        return {"RMSE": np.nan, "MAE": np.nan}

    for train_index, test_index in loo.split(lons):
        # Datos de entrenamiento
        lons_train, lats_train = lons[train_index], lats[train_index]
        vals_train = vals[train_index]

        # Datos de prueba (un solo punto)
        lons_test, lats_test = lons[test_index], lats[test_index]
        vals_test = vals[test_index]

        # Asegurarse de que hay suficientes puntos para entrenar
        if len(vals_train) < 3:
            continue

        pred_val = np.nan
```

```
try:
    if method in ["Kriging Ordinario", "Kriging con Deriva Externa (KED)"]:
        model = gs.Spherical(dim=2)
        bin_center, gamma = gs.vario_estimate(
            (lons_train, lats_train), vals_train
        )
        model.fit_variogram(bin_center, gamma, nugget=True)

        if method == "Kriging Ordinario":
            krig = gs.krige.Ordinary(
                model, (lons_train, lats_train), vals_train
            )
            pred_val, _ = krig([lons_test[0], lats_test[0]])

        elif method == "Kriging con Deriva Externa (KED)" and elevs is not None:
            elevs_train = elevs[train_index]
            elevs_test = elevs[test_index]
            krig = gs.krige.ExtDrift(
                model,
                (lons_train, lats_train),
                vals_train,
                drift_src=elevs_train,
            )
            pred_val, _ = krig(
                [lons_test[0], lats_test[0]], drift_tgt=[elevs_test[0]]
            )

        else:  # Fallback si KED no tiene elevs
            points_train = np.column_stack((lons_train, lats_train))
            pred_val = griddata(
                points_train,
                vals_train,
                (lons_test[0], lats_test[0]),
                method="linear",
            )

    elif method == "IDW":
        points_train = np.column_stack((lons_train, lats_train))
        pred_val = griddata(
            points_train,
            vals_train,
            (lons_test[0], lats_test[0]),
            method="linear",
        )

    elif method == "Spline (Thin Plate)":
```

```
            points_train = np.column_stack((lons_train, lats_train))
            pred_val = griddata(
                points_train,
                vals_train,
                (lons_test[0], lats_test[0]),
                method="cubic",
            )

        if not np.isnan(pred_val):
            y_true.append(vals_test[0])
            y_pred.append(pred_val)

    except Exception as e:
        print(f"Advertencia en LOOCV (Método: {method}): {e}")
        continue  # Saltar este punto si la interpolación falla

  if not y_true:
      return {"RMSE": np.nan, "MAE": np.nan}

  rmse = np.sqrt(mean_squared_error(y_true, y_pred))
  mae = mean_absolute_error(y_true, y_pred)
  return {"RMSE": rmse, "MAE": mae}


def interpolate_idw(lons, lats, vals, grid_lon, grid_lat, method="cubic"):
  """
  Realiza una interpolación espacial utilizando scipy.griddata.
  """
  points = np.column_stack((lons, lats))
  grid_x, grid_y = np.meshgrid(grid_lon, grid_lat)
  grid_z = griddata(points, vals, (grid_x, grid_y), method=method)
  grid_z = np.nan_to_num(grid_z)
  return grid_z


# ---------------------------------------------------------------------------
# PESTAÑA DE VALIDACIÓN
# ---------------------------------------------------------------------------


# --- CORRECCIÓN ---
# Eliminamos @st.cache_data de esta función interna
# para evitar problemas de "caché anidado".
# El caché en 'perform_loocv_for_all_methods' es suficiente.
def perform_loocv_for_year(year, method, gdf_metadata, df_anual_non_na):
  """
```

```python
    Realiza una Validación Cruzada Dejando Uno Afuera (LOOCV) para un año y método dados.
    """
    df_year = pd.merge(
        df_anual_non_na[df_anual_non_na[Config.YEAR_COL] == year],
        gdf_metadata,
        on=Config.STATION_NAME_COL,
    )

    clean_cols = [Config.LONGITUDE_COL, Config.LATITUDE_COL, Config.PRECIPITATION_COL]
    if (
        method == "Kriging con Deriva Externa (KED)"
        and Config.ELEVATION_COL in df_year.columns
    ):
        clean_cols.append(Config.ELEVATION_COL)

    df_clean = df_year.dropna(subset=clean_cols).copy()
    df_clean = df_clean[np.isfinite(df_clean[clean_cols]).all(axis=1)]
    df_clean = df_clean.drop_duplicates(
        subset=[Config.LONGITUDE_COL, Config.LATITUDE_COL]
    )

    if len(df_clean) < 4:
        return {"RMSE": np.nan, "MAE": np.nan}

    lons = df_clean[Config.LONGITUDE_COL].values
    lats = df_clean[Config.LATITUDE_COL].values
    vals = df_clean[Config.PRECIPITATION_COL].values
    elevs = (
        df_clean[Config.ELEVATION_COL].values
        if Config.ELEVATION_COL in df_clean
        else None
    )

    # --- CORRECCIÓN ---
    # Llamamos a la función _perform_loocv que ahora sí existe
    return _perform_loocv(method, lons, lats, vals, elevs)


@st.cache_data
def perform_loocv_for_all_methods(_year, _gdf_metadata, _df_anual_non_na):
    """Ejecuta LOOCV para todos los métodos de interpolación para un año dado."""
    methods = ["Kriging Ordinario", "IDW", "Spline (Thin Plate)"]
    if Config.ELEVATION_COL in _gdf_metadata.columns:
        methods.insert(1, "Kriging con Deriva Externa (KED)")

    results = []
```

```python
    for method in methods:
        metrics = perform_loocv_for_year(_year, method, _gdf_metadata, _df_anual_non_na)
        if metrics:
            results.append(
                {
                    "Método": method,
                    "Año": _year,
                    "RMSE": metrics.get("RMSE"),
                    "MAE": metrics.get("MAE"),
                }
            )
    return pd.DataFrame(results)


# ---------------------------------------------------------------------------
# PESTAÑA DE SUPERFICIES DE INTERPOLACIÓN
# ---------------------------------------------------------------------------
@st.cache_data
def create_interpolation_surface(
    df_period_mean, period_name, method, variogram_model, gdf_bounds, gdf_metadata
):
    """
    Crea una superficie de interpolación para un DataFrame de precipitación promedio de un período.

    Args:
        df_period_mean (pd.DataFrame): DataFrame con [Config.STATION_NAME_COL,
Config.PRECIPITATION_COL]
                                   conteniendo la precipitación media del período.
        period_name (str): Nombre del período (ej. "1990-2000") para el título del gráfico.
        method (str): "Kriging Ordinario", "IDW", etc.
        variogram_model (str): Modelo de variograma a usar.
        gdf_bounds (list): Límites [minx, miny, maxx, maxy] para la grilla.
        gdf_metadata (gpd.GeoDataFrame): Metadatos de las estaciones (para unir geometría y elevación).
    """

    fig_var = None
    error_msg = None

    # --- Data Preparation (Modificada) ---
    # Unir los datos de precipitación promedio con los metadatos (geometría, etc.)
    df_clean = pd.merge(
        df_period_mean, gdf_metadata, on=Config.STATION_NAME_COL, how="inner"
    )

    clean_cols = [Config.LONGITUDE_COL, Config.LATITUDE_COL, Config.PRECIPITATION_COL]
    if (
```

```
        method == "Kriging con Deriva Externa (KED)"
        and Config.ELEVATION_COL in df_clean.columns
    ):
        clean_cols.append(Config.ELEVATION_COL)

    df_clean = df_clean.dropna(subset=clean_cols).copy()
    df_clean[Config.LONGITUDE_COL] = pd.to_numeric(
        df_clean[Config.LONGITUDE_COL], errors="coerce"
    )
    df_clean[Config.LATITUDE_COL] = pd.to_numeric(
        df_clean[Config.LATITUDE_COL], errors="coerce"
    )
    df_clean[Config.PRECIPITATION_COL] = pd.to_numeric(
        df_clean[Config.PRECIPITATION_COL], errors="coerce"
    )
    df_clean = df_clean.dropna(
        subset=[Config.LONGITUDE_COL, Config.LATITUDE_COL, Config.PRECIPITATION_COL]
    )

    df_clean = df_clean[np.isfinite(df_clean[clean_cols]).all(axis=1)]
    df_clean = df_clean.drop_duplicates(
        subset=[Config.LONGITUDE_COL, Config.LATITUDE_COL]
    )

    if len(df_clean) < 4:
        error_msg = f"Se necesitan al menos 4 estaciones con datos válidos para el período {period_name} para
interpolar (encontradas: {len(df_clean)})."
        fig = go.Figure().update_layout(
            title=error_msg, xaxis_visible=False, yaxis_visible=False
        )
        return fig, None, error_msg

    lons = df_clean[Config.LONGITUDE_COL].values
    lats = df_clean[Config.LATITUDE_COL].values
    vals = df_clean[Config.PRECIPITATION_COL].values
    elevs = (
        df_clean[Config.ELEVATION_COL].values
        if Config.ELEVATION_COL in df_clean
        else None
    )

    # --- CÁLCULO DE RMSE ELIMINADO ---
    # El RMSE basado en LOOCV solo tiene sentido para un año específico,
    # no para un promedio de período.

    # Define grid based on bounds
```

```
if gdf_bounds is None or len(gdf_bounds) != 4 or not all(np.isfinite(gdf_bounds)):
    error_msg = "Límites geográficos (bounds) inválidos o no proporcionados."
    fig = go.Figure().update_layout(
        title=error_msg, xaxis_visible=False, yaxis_visible=False
    )
    return fig, None, error_msg

grid_lon = np.linspace(gdf_bounds[0] - 0.1, gdf_bounds[2] + 0.1, 150)
grid_lat = np.linspace(gdf_bounds[1] - 0.1, gdf_bounds[3] + 0.1, 150)
z_grid, error_message = None, None

# --- Interpolation Calculation ---
try:
    if method in ["Kriging Ordinario", "Kriging con Deriva Externa (KED)"]:
        model_map = {
            "gaussian": gs.Gaussian(dim=2),
            "exponential": gs.Exponential(dim=2),
            "spherical": gs.Spherical(dim=2),
            "linear": gs.Linear(dim=2),
        }
        model = model_map.get(variogram_model, gs.Spherical(dim=2))
        bin_center, gamma = gs.vario_estimate((lons, lats), vals)
        try:
            model.fit_variogram(bin_center, gamma, nugget=True)
        except ValueError as e_fit:
            raise ValueError(
                f"Error ajustando variograma: {e_fit}. Datos insuficientes o sin varianza espacial?"
            )

        try:
            fig_variogram_plt, ax = plt.subplots(figsize=(6, 4))
            ax.plot(bin_center, gamma, "o", label="Experimental")
            model.plot(ax=ax, label="Modelo Ajustado")
            ax.set_xlabel("Distancia")
            ax.set_ylabel("Semivarianza")
            ax.set_title(f"Variograma para {period_name}")
            ax.legend()
            plt.tight_layout()
            plt.close(fig_variogram_plt)
        except Exception as e_plot:
            print(
                f"Warning: No se pudo generar el gráfico del variograma: {e_plot}"
            )
            fig_var = None

        if method == "Kriging Ordinario":
```

```
            krig = gs.krige.Ordinary(model, (lons, lats), vals)
            z_grid, _ = krig.structured([grid_lon, grid_lat])
        else:  # KED
            if elevs is None:
                raise ValueError(
                    "Datos de elevación necesarios para KED no encontrados."
                )
            grid_x_elev, grid_y_elev = np.meshgrid(grid_lon, grid_lat)
            drift_grid = griddata(
                (lons, lats), elevs, (grid_x_elev, grid_y_elev), method="linear"
            )
            nan_mask_elev = np.isnan(drift_grid)
            if np.any(nan_mask_elev):
                fill_values_elev = griddata(
                    (lons, lats),
                    elevs,
                    (grid_x_elev[nan_mask_elev], grid_y_elev[nan_mask_elev]),
                    method="nearest",
                )
                drift_grid[nan_mask_elev] = fill_values_elev
            drift_grid = np.nan_to_num(drift_grid)

            krig = gs.krige.ExtDrift(model, (lons, lats), vals, drift_src=elevs)
            z_grid, _ = krig.structured(
                [grid_lon, grid_lat], drift_tgt=drift_grid.T
            )

    elif method == "IDW":
        grid_x, grid_y = np.meshgrid(grid_lon, grid_lat)
        z_grid = griddata((lons, lats), vals, (grid_x, grid_y), method="linear")
        nan_mask = np.isnan(z_grid)
        if np.any(nan_mask):
            fill_values = griddata(
                (lons, lats),
                vals,
                (grid_x[nan_mask], grid_y[nan_mask]),
                method="nearest",
            )
            z_grid[nan_mask] = fill_values

        if z_grid is not None:
            z_grid = np.nan_to_num(z_grid)

    elif method == "Spline (Thin Plate)":
        grid_x, grid_y = np.meshgrid(grid_lon, grid_lat)
        z_grid = griddata((lons, lats), vals, (grid_x, grid_y), method="cubic")
```

```
        nan_mask = np.isnan(z_grid)
        if np.any(nan_mask):
            fill_values = griddata(
                (lons, lats),
                vals,
                (grid_x[nan_mask], grid_y[nan_mask]),
                method="nearest",
            )
            z_grid[nan_mask] = fill_values

    if z_grid is not None:
        z_grid = np.nan_to_num(z_grid)

except Exception as e:
    error_message = f"Error al calcular {method}: {e}"
    import traceback

    print(traceback.format_exc())
    fig = go.Figure().update_layout(
        title=error_message, xaxis_visible=False, yaxis_visible=False
    )
    return fig, None, error_message

# --- Plotting Section ---
if z_grid is not None:
    fig = go.Figure(
        data=go.Contour(
            z=z_grid.T,
            x=grid_lon,
            y=grid_lat,
            colorscale=px.colors.sequential.YlGnBu,
            colorbar_title="Precipitación (mm)",
            contours=dict(
                coloring="heatmap",
                showlabels=True,
                labelfont=dict(size=10, color="white"),
                labelformat=".0f",
            ),
            line_smoothing=0.85,
            line_color="black",
            line_width=0.5,
        )
    )

    hover_texts = [
        f"<b>{row[Config.STATION_NAME_COL]}</b><br>"
```

```
            + f"Municipio: {row.get(Config.MUNICIPALITY_COL, 'N/A')}<br>"
            + f"Altitud: {row.get(Config.ALTITUDE_COL, 'N/A')} m<br>"
            + f"Ppt. Promedio: {row[Config.PRECIPITATION_COL]:.0f} mm"
            for _, row in df_clean.iterrows()
        ]

        fig.add_trace(
            go.Scatter(
                x=lons,
                y=lats,
                mode="markers",
                marker=dict(color="red", size=5, line=dict(width=1, color="black")),
                name="Estaciones",
                hoverinfo="text",
                text=hover_texts,
            )
        )

        # Anotación de RMSE eliminada

        fig.update_layout(
            title=f"Precipitación Promedio en {period_name} ({method})",  # <-- Título Modificado
            xaxis_title="Longitud",
            yaxis_title="Latitud",
            height=600,
            legend=dict(x=0.01, y=0.01, bgcolor="rgba(0,0,0,0)"),
        )
        return fig, None, None

    return (
        go.Figure().update_layout(title="Error: No se pudo generar la superficie Z"),
        None,
        "Superficie Z es None",
    )


@st.cache_data
def create_kriging_by_basin(gdf_points, grid_lon, grid_lat, value_col="Valor"):
    """
    Realiza Kriging. Si falla, usa un respaldo de interpolación lineal y relleno
    para asegurar una superficie con gradiente y sin vacíos.
    """
    lons = gdf_points.geometry.x
    lats = gdf_points.geometry.y
    vals = gdf_points[value_col].values
```

```
valid_indices = ~np.isnan(vals)
lons, lats, vals = lons[valid_indices], lats[valid_indices], vals[valid_indices]

if len(vals) < 3:
    st.error(
        "Se necesitan al menos 3 puntos con datos para realizar la interpolación."
    )
    ny, nx = len(grid_lat), len(grid_lon)
    return np.zeros((ny, nx)), np.zeros((ny, nx))

try:
    st.write("🛰️ Intentando interpolación con Kriging Ordinario...")
    bin_center, gamma = gs.vario_estimate((lons, lats), vals)
    model = gs.Spherical(dim=2)
    model.fit_variogram(bin_center, gamma, nugget=True)
    kriging = gs.krige.Ordinary(model, cond_pos=(lons, lats), cond_val=vals)
    grid_z, variance = kriging.structured([grid_lon, grid_lat], return_var=True)
    st.success("✅ Interpolación con Kriging completada con éxito.")
except (RuntimeError, ValueError) as e:
    st.warning(
        f"⚠️ El Kriging falló: '{e}'. Usando interpolación de respaldo (lineal + vecino cercano)."
    )
    points = np.column_stack((lons, lats))
    grid_x, grid_y = np.meshgrid(grid_lon, grid_lat)

    grid_z = griddata(points, vals, (grid_x, grid_y), method="linear")
    nan_mask = np.isnan(grid_z)
    if np.any(nan_mask):
        fill_values = griddata(
            points, vals, (grid_x[nan_mask], grid_y[nan_mask]), method="nearest"
        )
        grid_z[nan_mask] = fill_values

    grid_z = np.nan_to_num(grid_z)
    variance = np.zeros_like(grid_z)

return grid_z, variance
```