Programación Sistemas Agrarios

La programación para los sistemas agrarios es una disciplina que combina la tecnología de la información con la agricultura para mejorar la eficiencia, la productividad y la sostenibilidad en la gestión de las explotaciones agrícolas. A medida que la agricultura moderna se vuelve más compleja y orientada hacia la maximización de los rendimientos, la gestión de datos y la automatización desempeñan un papel crucial en la toma de decisiones agrícolas. La programación en este contexto se refiere a la creación y el uso de software y sistemas informáticos para abordar los desafíos específicos de la agricultura.

Los sistemas agrarios pueden abarcar una amplia gama de aplicaciones, desde la monitorización de cultivos y el control de plagas hasta la gestión de la cadena de suministro y la optimización de la maquinaria agrícola. La programación agraria implica el desarrollo de aplicaciones y algoritmos que recopilan, procesan y analizan datos agrícolas, lo que permite a los agricultores tomar decisiones más informadas sobre la siembra, el riego, la fertilización y otros aspectos críticos de la producción agrícola.

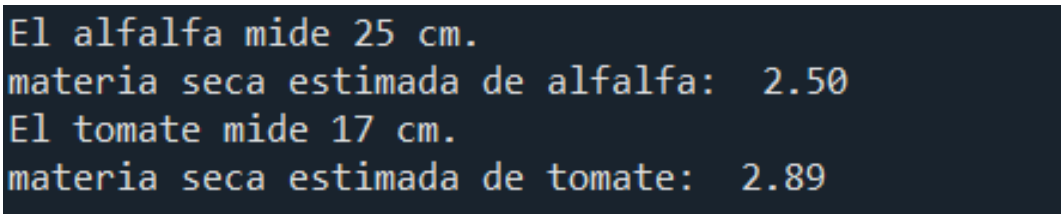
En este marco, la programación aplicada a los sistemas agrarios se abordará desde una perspectiva general, introduciendo los principios básicos de la programación y su utilidad en el tratamiento de datos.

A continuación, se realizará una recopilación y presentación de los distintos códigos y prácticas desarrollados a lo largo del trabajo, los cuales ejemplifican la aplicación de la programación. Estos ejercicios permiten ilustrar el uso de herramientas informáticas para el procesamiento, análisis y visualización de datos agrícolas, así como la implementación de métodos orientados a la automatización y al apoyo en la toma de decisiones. En conjunto, las prácticas presentadas reflejan cómo la programación constituye un recurso clave para abordar problemáticas reales desde una perspectiva tecnológica y analítica.

Agro-Modeling: Simulación de Crecimiento y Biomasa en Python

Desarrollo de un modelo orientado a objetos para automatizar el seguimiento de cultivos, integrando lógica de crecimiento dinámico y estimación de materia seca mediante ecuaciones alométricas personalizables.

```
class planta:
    def __init__ (self, cultivo, altura, biomasa):
        self.cultivo=cultivo
        self.altura=altura
        self.biomasa=biomasa

    def crecer(self, cm):
        self.altura += cm
        print(f"El {self.cultivo} mide {self.altura} cm.")

    def ms (self, a=0.02, b=1.5):
        self.biomasa=a*self.altura**b
        print(f"materia seca estimada de {self.cultivo}: {self.biomasa: .2f}")
        return self.biomasa

    def ms2 (self, a=0.01, b=2):
        self.biomasa=a*self.altura**b
        print(f"materia seca estimada de {self.cultivo}: {self.biomasa: .2f}")
        return self.biomasa
p1=planta ("alfalfa",20,20)
p1.crecer(5)
materia=p1.ms()
p2=planta("tomate",10,10)
p2.crecer (7)
materia=p2.ms2()
```



Smart Greenhouse: Simulación de Control Térmico Automatizado mediante PID

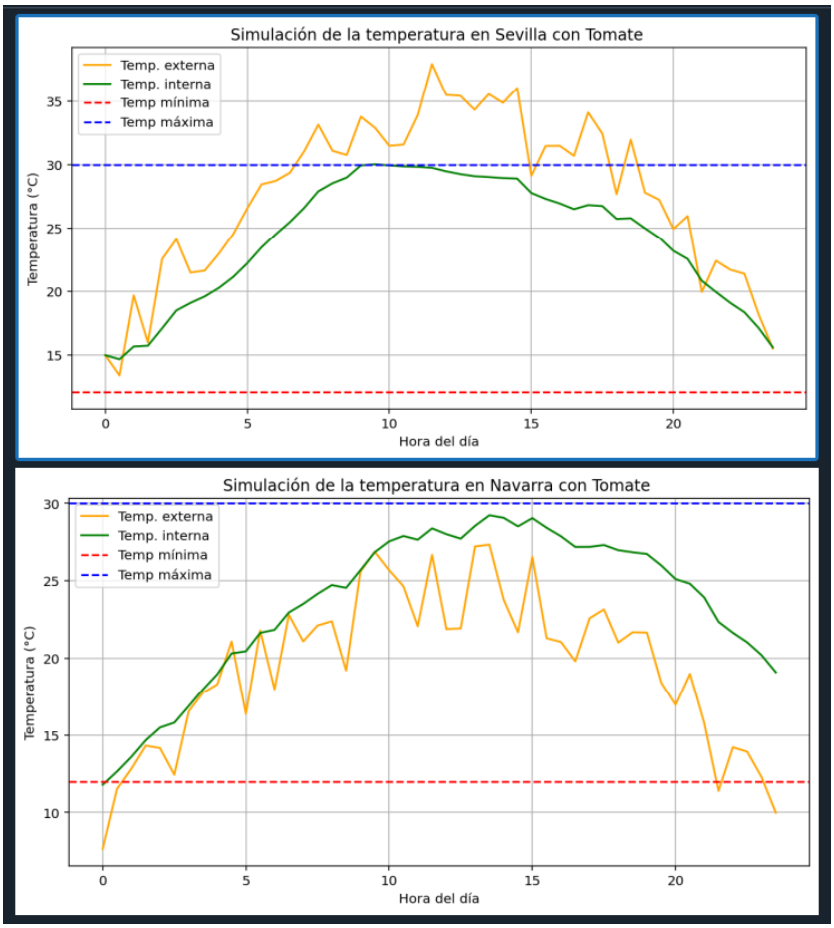
Sistema de control climático en Python que utiliza algoritmos PID para regular la temperatura interna de invernaderos, optimizando el confort térmico de los cultivos y calculando costos operativos en tiempo real.

```
import numpy as np
import matplotlib.pyplot as plt

# Parámetros simulacion temperatura
noise_m = 0         # Media del ruido aleatorio
noise_std = 2    # Desviación estándar del ruido aleatorio

# Clase para los cultivos
class Cultivo:
    def __init__(self, nombre, tempmin, tempmax):

        self.nombre = nombre
        self.tempmin = tempmin
        self.tempmax = tempmax

# crear un objeto de la clase cultivo
cultivo1=Cultivo("Tomate",12,30)
cultivo2=Cultivo("Lechuga", 10,28)

# Clase para los invernaderos
class Invernadero:
    def __init__(self, nombre, cultivo, Kp, Ki, Kd, coste_calentar, coste_enfriar,
max_temp, min_temp):

        self.nombre = nombre
        self.cultivo = cultivo
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.coste_calentar = coste_calentar
        self.coste_enfriar = coste_enfriar
        self.max_temp = max_temp
        self.min_temp = min_temp
        self.ext_temp = []  # Registro de temperaturas exteriores
        self.int_temp = []  # Registro de temperaturas internas
        self.current_int_temp=None  # Inicio de variable para temperatura interna
        self.integral=0  # Inicio de variable para PDI
```

```
        self.previous_error=0 # Inicio de variable para PDI
        self.pid_outputs = []  # Registro de las salidas del PID
        self.costes = []  # Registro del coste de control en cada ciclo


    def generate_temp(self, hour):
        """
        Genera una temperatura externa para el invernadero basada en una onda sinusoidal
con ruido.
        """
        sin_temp = self.min_temp + (self.max_temp - self.min_temp) * np.sin(np.pi * hour
/ 24)
        noise = np.random.normal(noise_m, noise_std)  # Ruido
        return sin_temp + noise


    def apply_pid_control(self, hour):
        """
        Aplica el control PID para ajustar la temperatura interna del invernadero.
        """
        # Generar la temperatura externa
        current_ext_temp = self.generate_temp(hour)
        self.ext_temp.append(current_ext_temp)


        # Inicializar la temperatura interna en el primer ciclo
        if self.current_int_temp is None:
            self.current_int_temp = current_ext_temp


        # Simular que la temperatura interna tiende a la externa
        self.current_int_temp += (current_ext_temp - self.current_int_temp) * 0.2


        # Calcular el error solo si la temperatura está fuera del rango del cultivo
        if self.current_int_temp < self.cultivo.tempmin:
            error = self.cultivo.tempmin - self.current_int_temp  # Elevar la
temperatura
        elif self.current_int_temp > self.cultivo.tempmax:
            error = self.cultivo.tempmax - self.current_int_temp  # Reducir la
temperatura
        else:
            error = 0  # No hacer ajustes si la temperatura está dentro del rango


        # Cálculo de las componentes del PID
        self.integral += error  # Acumular el error para la parte integral
        self.integral = np.clip(self.integral, -10, 10)  # Limitar el valor integral
        derivative = error - self.previous_error  # Diferencia en el error para la parte
derivativa
```

```
        pid_output = self.Kp * error + self.Ki * self.integral + self.Kd * derivative

        # Ajuste de la temperatura interna
        self.current_int_temp += pid_output

        # Calcular el coste
        if pid_output > 0:
            coste = pid_output * self.coste_calentar
        else:
            coste = -pid_output * self.coste_enfriar
        self.costes.append(coste)

        # Guardar los resultados
        self.int_temp.append(self.current_int_temp)
        self.pid_outputs.append(pid_output)
        self.previous_error = error

    def show_invernadero_status(self, hour):
        """
        Muestra el estado actual del invernadero: temperaturas externas e internas.
        """
        print(f"[{self.nombre} - {self.cultivo.nombre}] Hora {hour:.1f}:")
        print(f"  - Temp externa: {self.ext_temp[-1]:.2f}°C")
        print(f"  - Temp interna ajustada: {self.current_int_temp:.2f}°C")
        print(f"  - PID: {self.pid_outputs[-1]:.2f}°C (Coste ciclo: {self.costes[-
1]:.2f}€)\n")



# Crear objetos invernadero
invernadero1=Invernadero("Sevilla",cultivo1,0.7,0.2,0.05,10,5,35,15)
invernadero2=Invernadero("Navarra",cultivo1,0.7,0.2,0.05,10,5,25,10)
invernaderos = [invernadero1,invernadero2]

# Simulación de control PID para todos los invernaderos por 24 horas (48 intervalos de
media hora)
for invernadero in invernaderos:
    for half_hour in range(48):
        hour = half_hour / 2
        invernadero.apply_pid_control(hour)
        invernadero.show_invernadero_status(hour)

# Visualizar las temperaturas de los invernaderos
hours = np.arange(0, 24, 0.5)
```

```
for invernadero in invernaderos:
    plt.figure(figsize=(10, 5))
    plt.plot(hours, invernadero.ext_temp, label="Temp. externa", color='orange')
    plt.plot(hours, invernadero.int_temp, label="Temp. interna", color='green')
    plt.axhline(y=invernadero.cultivo.tempmin, color='r', linestyle='--', label="Temp mínima")
    plt.axhline(y=invernadero.cultivo.tempmax, color='b', linestyle='--', label="Temp máxima")
    plt.title(f'Simulación de la temperatura en {invernadero.nombre} con {invernadero.cultivo.nombre} ')
    plt.xlabel('Hora del día')
    plt.ylabel('Temperatura (°C)')
    plt.legend(loc='upper left')
    plt.grid(True)
    plt.show()
    print(f"{invernadero.nombre} con {invernadero.cultivo.nombre}- Coste total: {sum(invernadero.costes):.2f}€")
print("Simulación completa.")
```

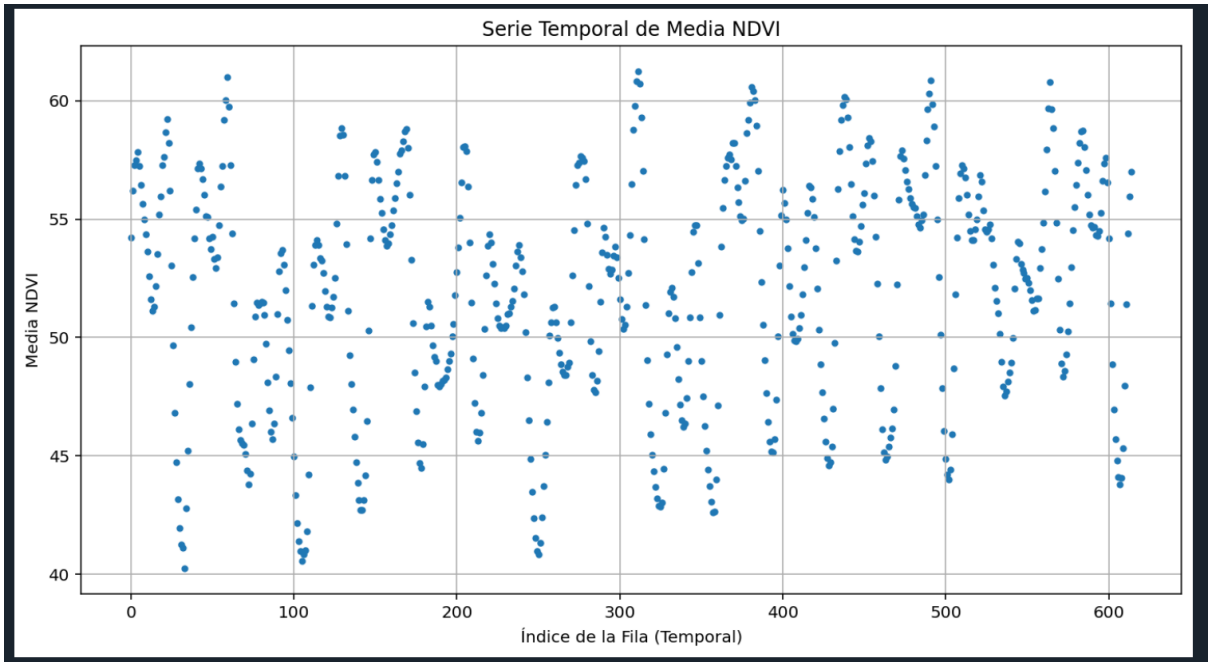


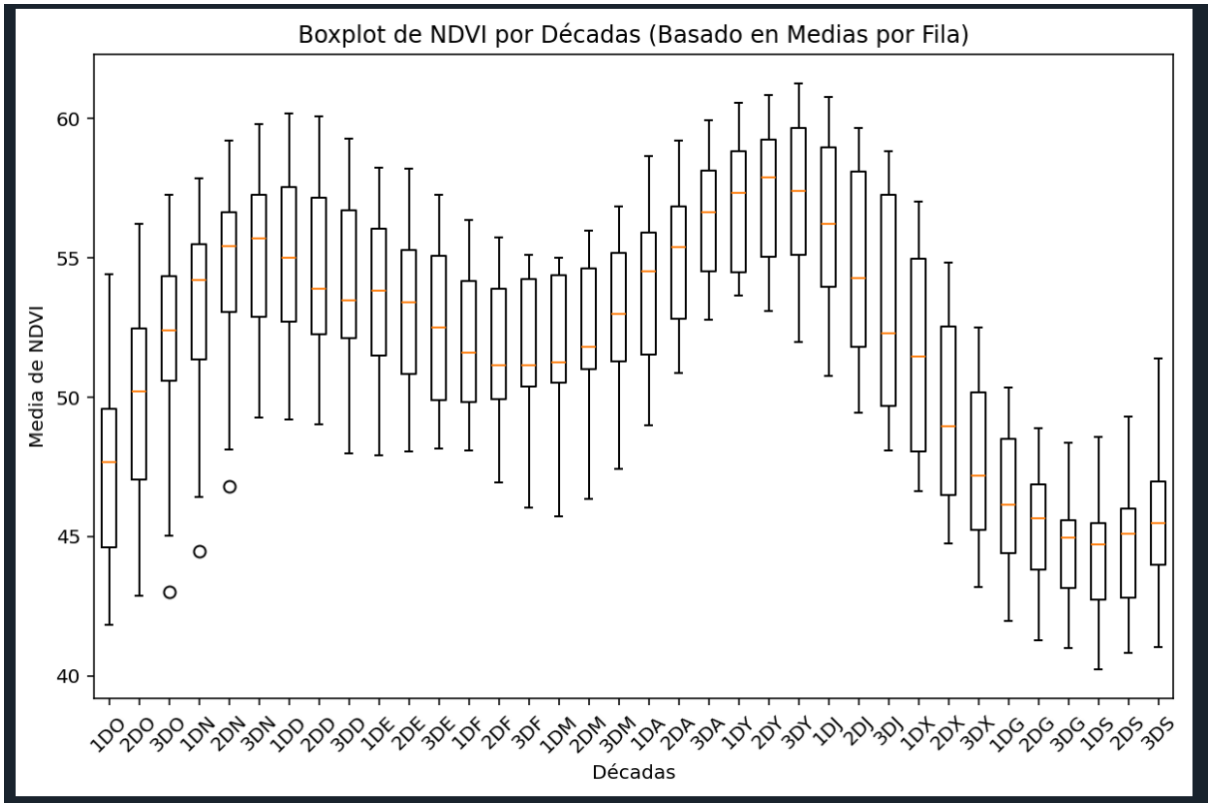
Remote Sensing: Análisis Espacial y Temporal de Índices NDVI

Procesamiento de datos satelitales con Pandas para calcular dinámicas de vegetación, aplicando estadísticas descriptivas, filtrado por cuantiles y visualización avanzada de series temporales y distribución decenal de píxeles.

```
import pandas as pd
import matplotlib.pyplot as plt


# RUTA ARCHIVO EXCEL
direc="C:/Users/Facun/OneDrive/Escritorio/Master AP/5.Programacion/Bloque 2/22.10.2025/"
nombre='Lozoya_NDVI.xlsx'
archivo_excel = direc+nombre


# LECTURA Y PROCESAMIENTO DE DATOS
df = pd.read_excel(archivo_excel)
# Asegúrate de que "Decenal" sea una columna de tipo cadena
df['Decenal'] = df['Decenal'].astype(str)
# Filtrar las columnas de los píxeles (asumiendo que las columnas de píxeles tienen nombres de coordenadas XY)
columnas_pixeles = [col for col in df.columns if col not in ["Season","Decenal"]]
# Calcular la media por fila de los valores NDVI de todos los píxeles
df['Media_NDVI'] = df[columnas_pixeles].mean(axis=1)
#valores superiores
q2 = df['Media_NDVI'].quantile(0.5)
valores_superiores = df[df['Media_NDVI'] > q2]


# GRAFICO PARA MOSTRAR LA SERIE TEMPORAL DE LA Media_NDVI
plt.figure(figsize=(12, 6))
plt.scatter(df.index, df['Media_NDVI'], marker='o', s=10)  # Usamos df.index en el eje x para mostrar todos los valores sin agrupar
plt.title('Serie Temporal de Media NDVI')
plt.xlabel('Índice de la Fila (Temporal)')
plt.ylabel('Media NDVI')
plt.grid(True)
plt.show()


# ESTADISTICA DESCRIPTIVA
media = df['Media_NDVI'].mean()
mediana = df['Media_NDVI'].median()
desviacion_estandar = df['Media_NDVI'].std()
minimo = df['Media_NDVI'].min()
maximo = df['Media_NDVI'].max()
```

```
print(f'Media: {media}')
print(f'Mediana: {mediana}')
print(f'Desviación Estándar: {desviacion_estandar}')
print(f'Mínimo: {minimo}')
print(f'Máximo: {maximo}')


# BOXPLOT DE LAS MEDIAS AGRUPADAS POR COLUMNA "Decenal"
plt.figure(figsize=(10, 6))
plt.title("Boxplot de NDVI por Décadas (Basado en Medias por Fila)")
plt.xlabel("Décadas")
plt.ylabel("Media de NDVI")
plt.grid(False)
plt.xticks(rotation=45)


plt.boxplot([df[df['Decenal'] == decada]['Media_NDVI'] for decada in
df['Decenal'].unique()], labels=df['Decenal'].unique())
plt.show()


print("Q2 (mediana):", "Media_NDVI")
print("Valores por encima del Q2:")
print(valores_superiores)


columnas_pixeles = [col for col in df.columns if col not in ["Season", "Decenal"]]
q2_global = df[columnas_pixeles].stack().median()
pixel = '24711529'
contador = (df[pixel] < q2_global).sum()
total = len(df)
porcentaje = (contador / total) * 100
print(f"Píxel {pixel} menor a Q2 {contador} veces ({porcentaje:.2f}%).")
```


Serie Temporal de Media NDVI
Media NDVI
Índice de la Fila (Temporal)
60
55
50
45
40
0
100
200
300
400
500
600


Boxplot de NDVI por Décadas (Basado en Medias por Fila)
Media de NDVI
Décadas
60
55
50
45
40
1DO
2DO
3DO
1DN
2DN
3DN
1DD
2DD
3DD
1DE
2DE
3DE
1DF
2DF
3DF
1DM
2DM
3DM
1DA
2DA
3DA
1DY
2DY
3DY
1DJ
2DJ
3DJ
1DX
2DX
3DX
1DG
2DG
3DG
1DS
2DS
3DS

Advanced Imputation & Anomaly Detection: Análisis de Z-Score en Series NDVI

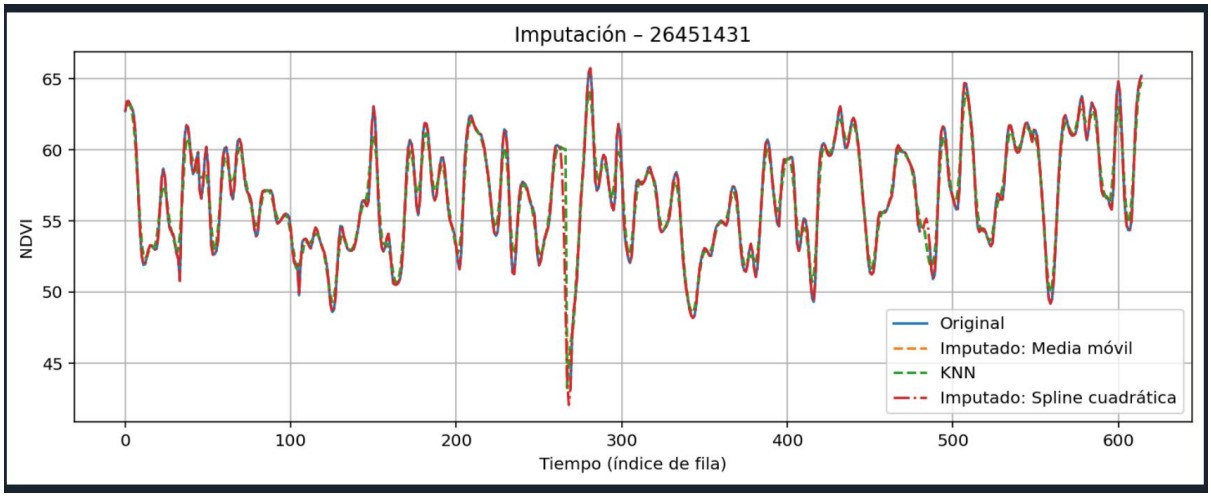
Implementación de modelos avanzados de imputación (KNN, Spline y Media Móvil) validados estadísticamente mediante el cálculo de Z-Scores para identificar anomalías y garantizar la precisión en datos satelitales.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.impute import KNNImputer

# -------------------- 1) LECTURA --------------------
direc  = r'C:/dataframe B2_4/'
nombre = 'Lozoya_NDVI2.xlsx'
df = pd.read_excel(direc + nombre)

# Serie de 3 píxeles
ts = df[['26451431','25601485', '25561499']].astype(float)

# -------------------- RESUMEN DE HUECOS  --------------------
nan_mask = ts.isna()  # dónde faltaba dato originalmente
print("\nHuecos originales por columna:")
print(nan_mask.sum())

# -------------------- IMPUTACIÓN: Media móvil vs Spline --------------------
# Media móvil centrada (ventana=5) + ffill/bfill
mm = ts.rolling(window=5, center=True, min_periods=1).mean()
ts_mm = mm.fillna(method='ffill').fillna(method='bfill')

# Spline polinómica de orden 2 (cuadrática)
ts_spline2 = ts.interpolate(method='polynomial', order=2, limit_direction='both')

# KNN (k=3): aprovecha la similitud entre filas/fechas usando la otra columna como pista
imputer = KNNImputer(n_neighbors=3, weights='distance')
ts_knn = pd.DataFrame(imputer.fit_transform(ts), index=ts.index, columns=ts.columns)

# -------------------- GRÁFICOS (solo líneas) --------------------
# Tomamos los 3 nombres de columna del DataFrame ts
c1, c2, c3 = ts.columns[:3]

# --- Píxel 1 ---
plt.figure(figsize=(12,4))
plt.plot(ts.index,     ts[c1],        linestyle='-',  label='Original')
plt.plot(ts_mm.index,  ts_mm[c1],     linestyle='--', label='Imputado: Media móvil')
```

```
plt.plot(ts_knn.index,  ts_mm[c1],     linestyle='--', label='KNN')
plt.plot(ts_spline2.index, ts_spline2[c1], linestyle='-.', label='Imputado: Spline cuadrática')
plt.title(f'Imputación – {c1}')
plt.xlabel('Tiempo (índice de fila)'); plt.ylabel('NDVI')
plt.grid(True); plt.legend(); plt.show()


# --- Píxel 2 ---
plt.figure(figsize=(12,4))
plt.plot(ts.index,     ts[c2],        linestyle='-',  label='Original')
plt.plot(ts_mm.index,  ts_mm[c2],     linestyle='--', label='Imputado: Media móvil')
plt.plot(ts_knn.index,  ts_mm[c2],     linestyle='--', label='KNN')
plt.plot(ts_spline2.index, ts_spline2[c2], linestyle='-.', label='Imputado: Spline cuadrática')
plt.title(f'Imputación – {c2}')
plt.xlabel('Tiempo (índice de fila)'); plt.ylabel('NDVI')
plt.grid(True); plt.legend(); plt.show()


# --- Píxel 3 ---
plt.figure(figsize=(12,4))
plt.plot(ts.index,     ts[c3],        linestyle='-',  label='Original')
plt.plot(ts_mm.index,  ts_mm[c3],     linestyle='--', label='Imputado: Media móvil')
plt.plot(ts_knn.index,  ts_mm[c3],     linestyle='--', label='KNN')
plt.plot(ts_spline2.index, ts_spline2[c3], linestyle='-.', label='Imputado: Spline cuadrática')
plt.title(f'Imputación – {c3}')
plt.xlabel('Tiempo (índice de fila)'); plt.ylabel('NDVI')
plt.grid(True); plt.legend(); plt.show()


#-----Bucle------


for col in ts.columns:
    plt.figure(figsize=(12,4))
    plt.plot(ts.index,     ts[col],        linestyle='-',  label='Original')
    plt.plot(ts_mm.index,  ts_mm[col],     linestyle='--', label='Imputado: Media móvil')
    plt.plot(ts_knn.index,  ts_mm[col],     linestyle='--', label='KNN')
    plt.plot(ts_spline2.index, ts_spline2[c3], linestyle='-.', label='Imputado: Spline cuadrática')
    plt.title(f'Imputación – {col}')
    plt.xlabel('Tiempo (índice de fila)'); plt.ylabel('NDVI')
    plt.grid(True); plt.legend(); plt.show()
```

```
#---- media,desvio y zscore-----
# Media por columna (cada píxel)
media = ts.mean()
print("Media por píxel:")
print(media)

# Desvío estándar por columna
desvio = ts.std()
print("\nDesvío estándar por píxel:")
print(desvio)

# -------------------- Z-SCORE --------------------
# (valor - media) / desvío estándar
zscore = (ts - media) / desvio
print("\nZ-score por píxel:")
print(zscore.head())  # muestra las primeras filas

for col in ts.columns:
    plt.figure(figsize=(10, 4))
    plt.plot(zscore.index, zscore[col], label=f'Z-score {col}')
    plt.axhline(0, color='gray', linestyle='--')
    plt.title(f'Z-score – {col}')
    plt.xlabel('Tiempo')
    plt.ylabel('Z-score')
    plt.legend()
    plt.grid(True)
    plt.show()

# -------------------- 1) ESTADÍSTICAS Y Z-SCORE ORIGINAL --------------------
media = ts.mean()
desvio = ts.std()
zscore = (ts - media) / desvio
# -------------------- 2) MÁSCARA DE VALORES FALTANTES --------------------
nan_mask = ts.isna()
# -------------------- 3) Z-SCORES DE LAS SERIES IMPUTADAS --------------------
z_mm      = (ts_mm - media) / desvio
z_spline2 = (ts_spline2 - media) / desvio
z_knn     = (ts_knn - media) / desvio
# -------------------- 4) COMPARACIÓN DE Z-SCORE EN LOS PUNTOS IMPUTADOS ---------------
-----
comparacion = pd.DataFrame(index=ts.index)
```

```
for col in [c1, c2, c3]:
    # diferencias absolutas
    dif_mm = abs(z_mm[col] - zscore[col])
    dif_spline = abs(z_spline2[col] - zscore[col])
    dif_knn = abs(z_knn[col] - zscore[col])

    # guardar solo donde había NaN
    comparacion.loc[nan_mask[col], f'{col}_mm_diff'] = dif_mm
    comparacion.loc[nan_mask[col], f'{col}_spline_diff'] = dif_spline
    comparacion.loc[nan_mask[col], f'{col}_knn_diff'] = dif_knn

# -------------------- 5) RESULTADOS RESUMIDOS --------------------
print("\nPromedio de diferencia de z-score (solo donde se imputó):")
print(comparacion.mean().round(4))

# -------------------- 6) GRÁFICOS DE COMPARACIÓN --------------------
for col in [c1, c2, c3]:
    plt.figure(figsize=(10, 4))
    plt.plot(zscore.index, zscore[col], label='Z-score original', color='black')
    plt.plot(z_mm.index, z_mm[col], '--', label='Media móvil')
    plt.plot(z_spline2.index, z_spline2[col], '-.', label='Spline')
    plt.plot(z_knn.index, z_knn[col], ':', label='KNN')
    plt.title(f'Comparación de Z-score – {col}')
    plt.xlabel('Tiempo')
    plt.ylabel('Z-score')
    plt.legend()
    plt.grid(True)
    plt.show()
```

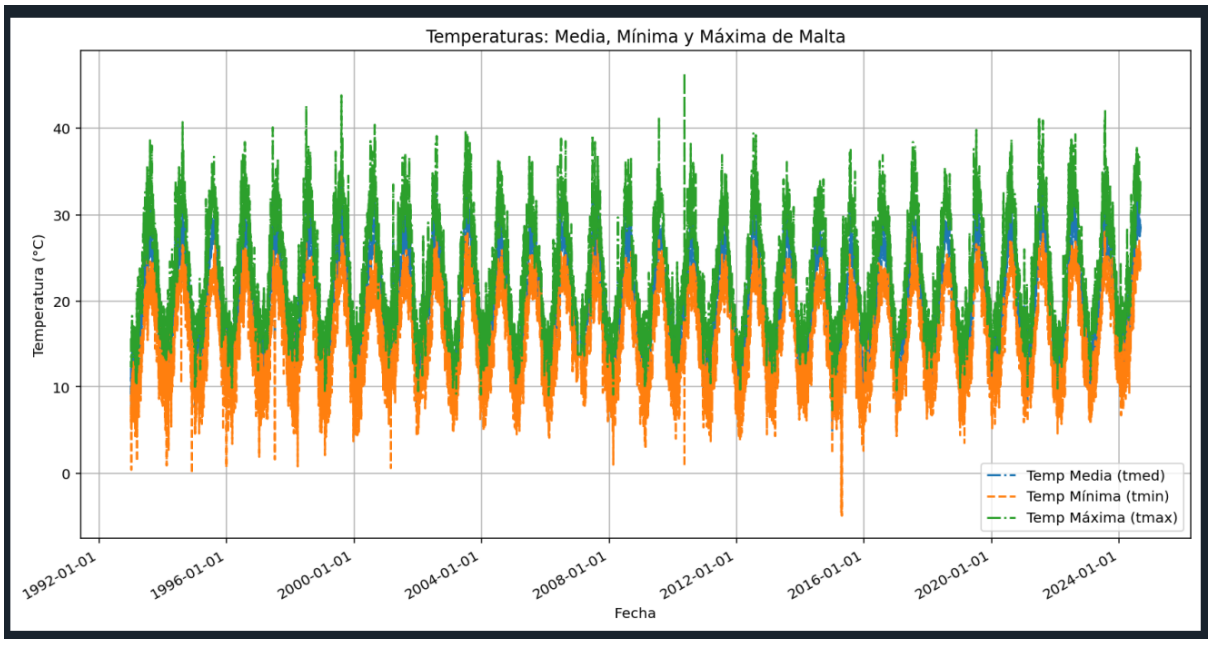


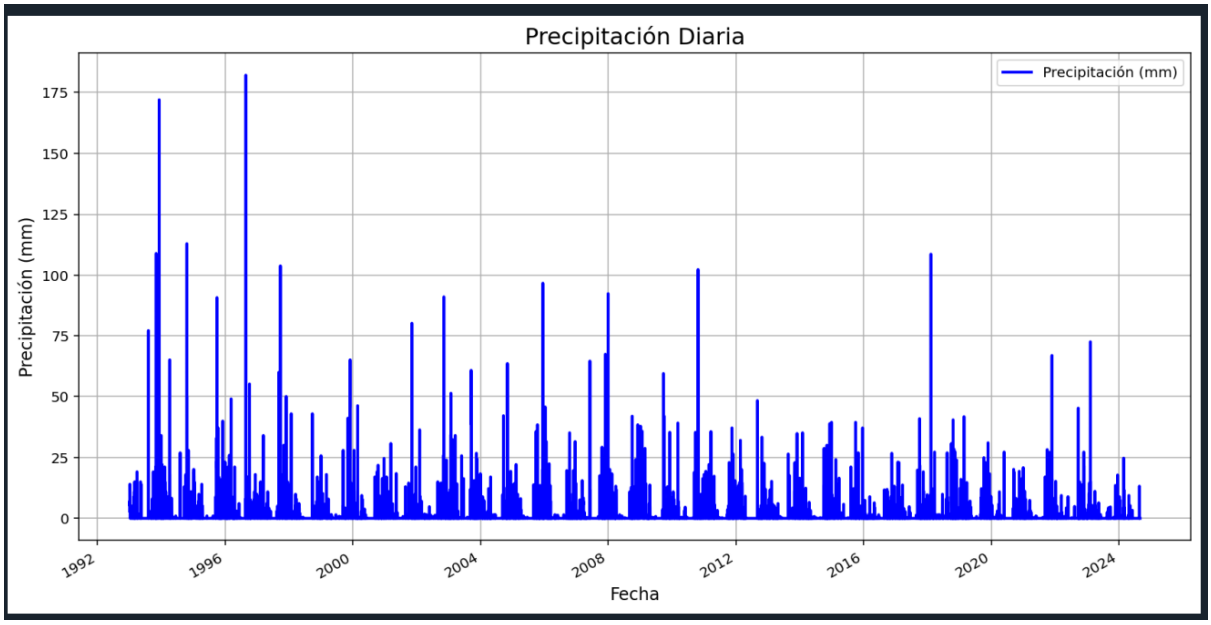
Climate Data: Análisis Temporal de Variables Agrometeorológicas en Malta

Implementación en Python para el procesamiento de series climáticas históricas. El sistema transforma registros crudos en objetos temporales, gestiona la limpieza de datos numéricos y genera visualizaciones profesionales de la evolución diaria de temperaturas (máxima, media, mínima) y precipitaciones para el análisis de tendencias meteorológicas.

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
# Leer el archivo CSV
ruta = "C:/Users/Facun/OneDrive/Escritorio/Master AP/5.Programacion/Bloque
3/Clase_1_5.11.2025/"
archivo = "Malta_Climate.csv"
df = pd.read_csv(ruta + archivo, delimiter=';')
est = 'Malta'

# Mostrar las primeras filas y tipos de datos
print(df.head())
print(df.info())

# Crear una columna de fecha a partir de las columnas day, month y year
df['fecha'] = pd.to_datetime(df[['year', 'month', 'day']])

# Asegurarse de que las columnas numéricas estén en el formato correcto
columnas_numericas = ['tmax', 'tmin', 'tmed', 'preci']
df[columnas_numericas] = df[columnas_numericas].apply(pd.to_numeric, errors='coerce')

# Verificar las primeras filas para confirmar que todo esté correcto
print(df.head())

# La fecha como índice
df.set_index('fecha', inplace=True)

# Plot de las tres temperaturas: tmed, tmin, tmax
# Configurar el formato de fecha para el eje X
plt.figure(figsize=(14, 7))
plt.plot(df.index, df['tmed'], label='Temp Media (tmed)', linestyle='-.')
plt.plot(df.index, df['tmin'], label='Temp Mínima (tmin)', linestyle='--')
plt.plot(df.index, df['tmax'], label='Temp Máxima (tmax)', linestyle='-.')
plt.title('Temperaturas: Media, Mínima y Máxima de ' + est)
plt.xlabel('Fecha')
plt.ylabel('Temperatura (°C)')
plt.legend()
```

```
plt.grid()


# Formatear las fechas en el eje X
date_format = DateFormatter('%Y-%m-%d')  # Formato de fecha 'YYYY-MM-DD'
plt.gca().xaxis.set_major_formatter(date_format)
plt.gcf().autofmt_xdate()  # Rotar las etiquetas de fecha para legibilidad
plt.show()


# Gráfico Precipitación
plt.figure(figsize=(14, 7))
plt.plot(df.index, df['preci'], label='Precipitación (mm)', color='blue', linestyle='-',
linewidth=2)
plt.title('Precipitación Diaria', fontsize=16)
plt.xlabel('Fecha', fontsize=12)
plt.ylabel('Precipitación (mm)', fontsize=12)
plt.legend()
plt.grid()
plt.gcf().autofmt_xdate()  # Formato para las fechas en el eje X
plt.show()
```



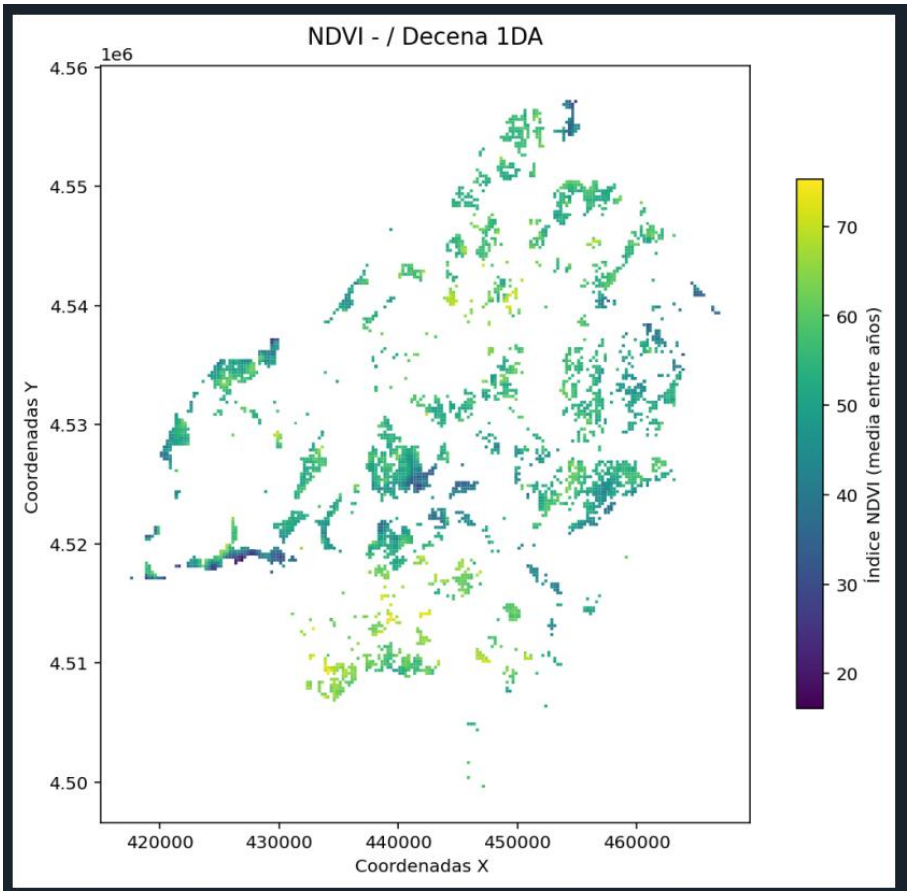


Geospatial Analytics: Mapeo de Dinámicas de Vegetación con GeoPandas y NDVI

Automatización de cartografía digital mediante el procesamiento de datos multiespectrales y archivos shapefile, generando mapas coropléticos que visualizan la evolución espacial y temporal del vigor vegetal por periodos decenales.

```
import os, re
import pandas as pd
import matplotlib.pyplot as plt
import geopandas as gpd

direc = "C:/Users/Facun/OneDrive/Escritorio/Master AP/5.Programacion/Bloque
3/Clase_2_7.11.2025/"
xls = os.path.join(direc, 'Lozoya_NDVI.xlsx')
shp= os.path.join(direc, 'Comarca_Lozoya.shp')

# Leer datos
df = pd.read_excel(xls)
df.columns = df.columns.astype(str)
gdf = gpd.read_file(shp)

# Ajustar columna para unir con el Excel
gdf["colrow"] = gdf["colrow"].apply(lambda x: str(x).replace('-', ''))



for decena in sorted(df["Decenal"].unique()):
    subset = df[df["Decenal"] == decena]
```

```
    row = subset.drop(columns=["Season", "Decenal"]).mean()
    row = pd.to_numeric(row, errors='coerce')
    vals = row.rename_axis("colrow").reset_index(name="NDVI")

    # shapefile
    gplot = gdf.merge(vals, on="colrow", how="left")

    if gplot["NDVI"].isna().all():
        print(f"Decena {decena}: sin datos")
        continue

    vmin, vmax = gplot["NDVI"].min(), gplot["NDVI"].max()

    fig, ax = plt.subplots(figsize=(8, 7))
    gplot.plot(
        ax=ax,
        column="NDVI",
        cmap="viridis",
        vmin=vmin,
        vmax=vmax,
        legend=True,
        legend_kwds={
            "label": "Índice NDVI (media entre años)",
            "orientation": "vertical",
            "shrink": 0.7})

    # Título y ejes
    ax.set_title(f"NDVI - / Decena {decena}", fontsize=13, pad=12)
    ax.set_xlabel("Coordenadas X ", fontsize=10)
    ax.set_ylabel("Coordenadas Y ", fontsize=10)
    plt.tight_layout()
    plt.show()
```



Climate Analytics: Descomposición de Series Temporales y Reportes Automatizados en PDF

Desarrollo de un flujo de trabajo en Python para el análisis climático de Malta, integrando pruebas de normalidad de Shapiro-Wilk, descomposición estacional STL y detección de eventos extremos, con exportación automática a informes profesionales en PDF.

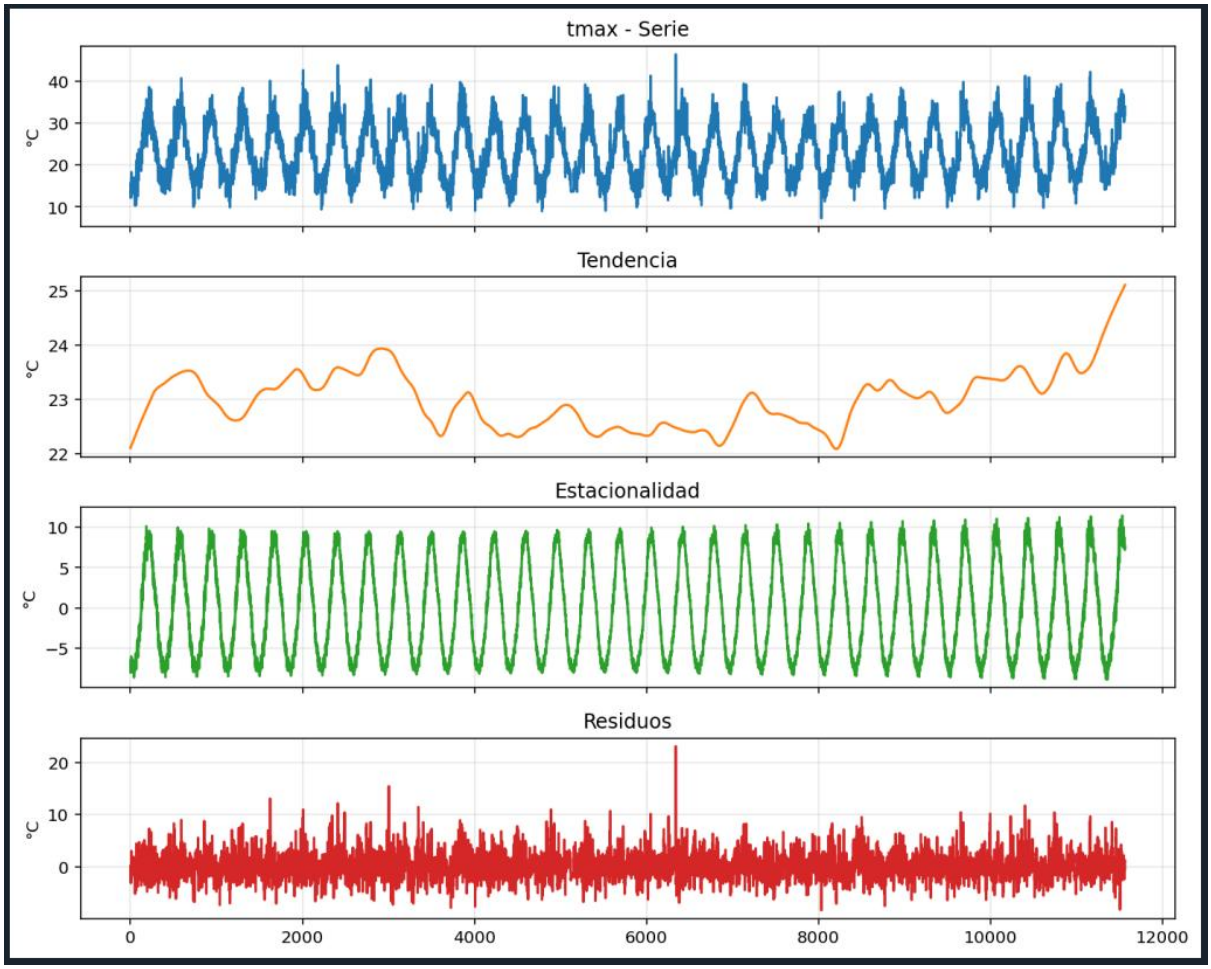
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from statsmodels.tsa.seasonal import STL
from scipy.stats import shapiro

# ==========================
# 1) FUNCIÓN: lectura + fecha
# ==========================
def leer_malta(ruta, archivo, sep=';', dec='.', freq='D'):
    df = pd.read_csv(ruta + archivo, delimiter=sep, decimal=dec)
    df['fecha'] = pd.to_datetime(df[['year', 'month', 'day']], errors='coerce')
```

```
    return df


# ----------------- PARÁMETROS -----------------
ruta = "C:/Users/Facun/OneDrive/Escritorio/Master AP/5.Programacion/Bloque 3/Clase_3_14.11.2025/"
archivo = "Malta_Climate.csv"
periodo_estacional = 365               # diario -> 365 (si mensual, usar 12)
salida_pdf = "C:/Users/Facun/OneDrive/Escritorio/Master AP/5.Programacion/Bloque 3/Clase_3_14.11.2025/Informe_preliminar_Malta_Finalizado.pdf"


# ==========================
# 2) LECTURA
# ==========================
data = leer_malta(ruta, archivo)


# ==========================
# 3) SHAPIRO-WILK tmax/tmin
# ==========================
tmax = pd.to_numeric(data['tmax'], errors='coerce').astype(float)
tmin = pd.to_numeric(data['tmin'], errors='coerce').astype(float)
Pp = pd.to_numeric(data['preci'], errors='coerce').astype(float)

# (Shapiro tiene límite de tamaño. Muestreamos si hay >5000 observaciones)
def shapiro_safe(x, max_n=5000, seed=0):
    x = pd.Series(x).dropna()
    if len(x) < 3:
        return np.nan, np.nan
    if len(x) > max_n:
        x = x.sample(max_n, random_state=seed)
    W, p = shapiro(x.values)
    return W, p

W_tmax, p_tmax = shapiro_safe(tmax)
W_tmin, p_tmin = shapiro_safe(tmin)

tabla_shapiro = pd.DataFrame({
    'Serie': ['tmax', 'tmin'],
    'W': [W_tmax, W_tmin],
    'p-valor': [p_tmax, p_tmin],
    'Conclusión (α=0.05)': ['No normal' if (pd.notna(p_tmax) and p_tmax < 0.05) else 'Normal',
                            'No normal' if (pd.notna(p_tmin) and p_tmin < 0.05) else 'Normal']
})
```

```
print("\n=== Shapiro–Wilk (tmax/tmin) ===")
print(tabla_shapiro.round(4))

# ==========================
# 4) STL de tmax (gráficos)
# ==========================
#ponemos como son series diarias, la estacionalidad es 365
stl = STL(tmax, period=periodo_estacional, robust=True, seasonal=365).fit()

# Plot serie tmax
plt.figure(figsize=(10, 3.2))
plt.plot(tmax, color='C0')
plt.title("Serie tmax")
plt.ylabel("ºC"); plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Plot descomposición, usamos de stl.
fig, axes = plt.subplots(4, 1, figsize=(10, 8), sharex=True)
axes[0].plot(tmax, color='C0'); axes[0].set_title("tmax - Serie")
axes[1].plot(stl.trend, color='C1'); axes[1].set_title("Tendencia")
axes[2].plot(stl.seasonal, color='C2'); axes[2].set_title("Estacionalidad")
axes[3].plot(stl.resid, color='C3'); axes[3].set_title("Residuos")
for ax in axes: ax.grid(alpha=0.3); ax.set_ylabel("ºC")
plt.tight_layout(); plt.show()

# ==========================
# 5) Fechas con tmin < p05 (tabla en consola)
# ==========================
q05_tmin = tmin.quantile(0.05)
fechas_tmin_bajas = tmin[tmin < q05_tmin].dropna()
tabla_tmin_fechas = pd.DataFrame({
    'fecha': pd.to_datetime(fechas_tmin_bajas.index).date,
    'tmin (ºC)': fechas_tmin_bajas.values
})
print(f"\n=== Fechas con tmin < p05 ({q05_tmin:.2f} ºC) ===")
print(tabla_tmin_fechas.head(20))  # muestrario; el alumno puede guardar a CSV si quiere

# ==========================
# 6) Informe preliminar en PDF (A4)
#     - Tabla normalidad tmax/tmin
```

```
#     - Gráfico serie tmax
#     - Descomposición STL tmax
#     - Texto “explicación”
#     - Tabla de fechas con precipitación < p05
# ==========================
preci = pd.to_numeric(data['preci'], errors='coerce').astype(float)
q97_preci = preci[preci > 0].quantile(0.97)
fechas_preci_altas = preci[preci > q97_preci]
tabla_preci_fechas = pd.DataFrame({'fecha':
pd.to_datetime(fechas_preci_altas.index).date,
                                   'precip (mm/día)': fechas_preci_altas.values})


print(f"\n=== Fechas con preci > p97 ({q97_preci:.2f} mm) ===")
print(tabla_preci_fechas.head(20))  # muestrario; el alumno puede guardar a CSV si
quiere


A4 = (8.27, 11.69)


with PdfPages(salida_pdf) as pdf:
    # Portada
    fig, ax = plt.subplots(figsize=A4)
    ax.axis('off')
    ax.text(0.5, 0.80, "Informe preliminar — Malta", ha='center', fontsize=18,
weight='bold')
    ax.text(0.5, 0.68, "Normalidad (tmax/tmin) • Serie y descomposición (tmax) • Fechas
precip < p05",
            ha='center', fontsize=11)
    pdf.savefig(fig); plt.close(fig)


    # Tabla Shapiro
    fig, ax = plt.subplots(figsize=A4)
    ax.axis('off'); ax.set_title("Shapiro–Wilk — tmax/tmin (α=0.05)", fontsize=14)
    tbl = ax.table(cellText=tabla_shapiro.round(4).values,
                   colLabels=tabla_shapiro.columns, loc='center')
    tbl.auto_set_font_size(False); tbl.set_fontsize(10); tbl.scale(1, 1.2)
    pdf.savefig(fig); plt.close(fig)



    # Descomposición tmax
    fig, axes = plt.subplots(4, 1, figsize=A4, sharex=True)
    axes[0].plot(tmax, color='C0'); axes[0].set_title("tmax - Serie")
    axes[1].plot(stl.trend, color='C1'); axes[1].set_title("Tendencia")
    axes[2].plot(stl.seasonal, color='C2'); axes[2].set_title("Estacionalidad")
    axes[3].plot(stl.resid, color='C3'); axes[3].set_title("Residuos")
```

```
    for ax in axes: ax.grid(alpha=0.3); ax.set_ylabel("°C")
    fig.tight_layout(); pdf.savefig(fig); plt.close(fig)


    # Texto de descripción (edítalo tú)
    fig, ax = plt.subplots(figsize=A4)
    ax.axis('off')
    ax.text(0.02, 0.92, "Descripción/explicación (edítame):", fontsize=14,
weight='bold')
    ax.text(0.02, 0.78, "• Normalidad: interpreta la tabla de Shapiro para tmax y
tmin.", fontsize=11)
    ax.text(0.02, 0.64, "• tmax: comenta la tendencia observada y la amplitud de la
estacionalidad.", fontsize=11)
    ax.text(0.02, 0.50, "• Residuos: ¿parecen ruido blanco? ¿hay outliers?",
fontsize=11)
    ax.text(0.02, 0.36, "• Nota: si precip tiene muchos 0, el p05 puede ser 0 mm/día.",
fontsize=11)
    pdf.savefig(fig); plt.close(fig)


    # Tabla de fechas precip < p05
    fig, ax = plt.subplots(figsize=A4)
    ax.axis('off')
    ax.set_title(f"Fechas con precipitación > p97 (umbral = {q97_preci:.2f} mm/día)",
fontsize=14)
    if len(tabla_preci_fechas) == 0:
        ax.text(0.02, 0.6, "No se han detectado fechas por encima del percentil 0.97.",
fontsize=12)
    else:
        # si es muy larga, se puede cortar o sugerir exportar a CSV
        muestra = tabla_preci_fechas.head(40)  # muestra para el informe
        tbl = ax.table(cellText=muestra.values,
                       colLabels=muestra.columns, loc='center')
        tbl.auto_set_font_size(False); tbl.set_fontsize(9); tbl.scale(1, 1.2)
        ax.text(0.02, 0.06, "(*solo se muestran las primeras 40 filas; exporta a CSV si
necesitas todas*)",
                fontsize=9)
    pdf.savefig(fig); plt.close(fig)


print(f"\nInforme PDF guardado en: {salida_pdf}")
```



Agro-Tech: Automatización de Evapotranspiración mediante API de AEMET y Hargreaves

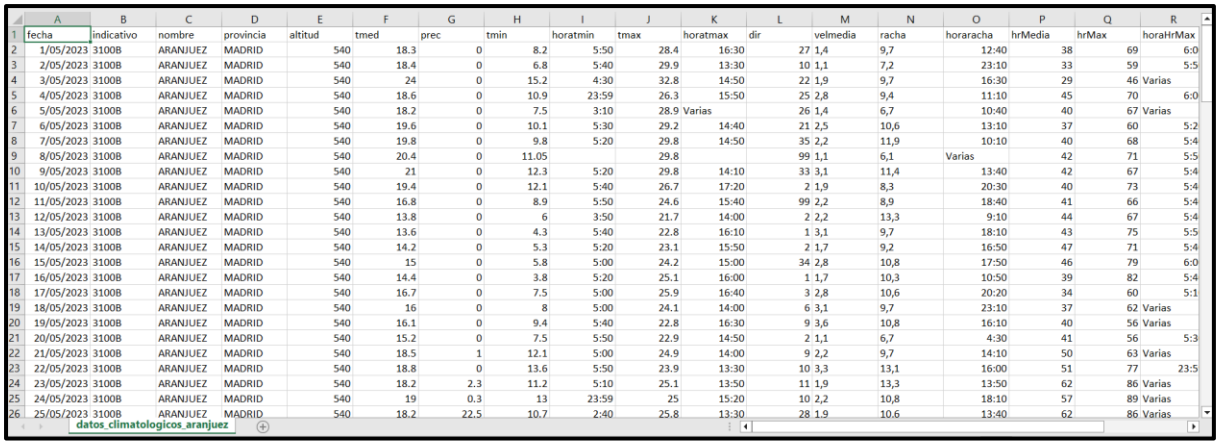
Desarrollo de un sistema que integra peticiones a APIs climáticas con el modelo matemático de Hargreaves para calcular la ETP diaria. El flujo incluye limpieza de datos, imputación por interpolación lineal y mapeo dinámico de radiación solar según latitud geográfica.

```
import requests
import pandas as pd
import numpy as np


# Diccionario para mapear números de mes a formato 'Ene', 'Feb', etc.
month_dict = {1: 'Ene', 2: 'Feb', 3: 'Mar', 4: 'Abr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Ago', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dic'}


def cargar_radiacion(ruta_tabla,latitud_est):
    # Cargar la tabla de la radiación
    tabla_radiacion = pd.read_csv(ruta_tabla, index_col=0) # 'Latitud'
```

```
    # Verificar si la latitud_est está presente en la columna "Latitud"
    if latitud_est in tabla_radiacion.index:
        # Obtener los valores de Rs para la latitud_est
        valores_rs = tabla_radiacion.loc[latitud_est].values
        # Crear un diccionario para mapear meses a valores de Rs
        diccionario_rs = dict(zip(tabla_radiacion.columns, valores_rs))
        return diccionario_rs
    else:
        print(f"Error: Latitud {latitud_est} no encontrada en el archivo.")
        return None

def calcular_etp(row):

   # Obtener el mes de la fecha y formatearlo como 'Ene', 'Feb', etc.
    mes_numero = int(row['fecha'].split('-')[1])
    mes_abreviado = month_dict.get(mes_numero, np.nan)

    # Obtener la radiación correspondiente al mes
    radiacion = diccionario_rs.get(mes_abreviado, np.nan)

    # Fórmula de Hargreaves para calcular ETP
    etp = 0.0023 * (row['tmed'] + 17.8) * np.sqrt(row['tmax'] - row['tmin']) * radiacion
    return etp

# Input: estacion, fecha de inicio y fin, latitud, ruta
est='3100B' #aranjuez             #menorca 'B893'  #Ibiza 'B954'
fini='2023-05-01'
ffin='2023-08-31'
latitud_est=40

ruta="C:/Users/Facun/OneDrive/Escritorio/Master AP/5.Programacion/Bloque
3/Clase_4_19.11.2025/"
resultado="datos_climatologicos_aranjuez.csv"
tabla="Latitud.csv"
ruta_tabla=ruta+tabla



# Cargar la información de radiación solar para la latitud dada
diccionario_rs = cargar_radiacion(ruta_tabla, latitud_est)

# Verificar si se obtuvo la información correctamente
if diccionario_rs:
```

```
    # Imprimir el diccionario para verificar
    print(diccionario_rs)
else:
    # Manejar el error según sea necesario
    print("No se pudo obtener la información de radiación solar.")


# Peticion informacion
url =
"https://opendata.aemet.es/opendata/api/valores/climatologicos/diarios/datos/fechaini/"+
fini+"T00%3A00%3A00UTC/fechafin/"+ffin+"T23%3A59%3A59UTC/estacion/"+est


querystring =
{"api_key":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhbmFtYXJpYS50YXJxdWlzQHVwbS5lcyIsImp0aSI6ImY
3YjUxNDhiLTViMTgtNDIwYy04NGVmLWIyNTk4OTY0ZmJlYiIsImlzcyI6IkFFTUVUIiwiaWF0IjoxNjk5NTU1ODc
5LCJ1c2VySWQiOiJmN2I1MTQ4Yi01YjE4LTQyMGMtODRlZi1iMjU5ODk2NGZiZWIiLCJyb2xlIjoiIn0.2BtWZ91
A8NZI0D3wgcF0lhZlCFGiRsfkMBKsBWcS3Hw"}


headers = {
    'cache-control': "no-cache"
    }


response = requests.request("GET", url, headers=headers, params=querystring)


print(response.text)


# Verifica si la solicitud fue exitosa (código de estado 200)
if response.status_code == 200:
    # Convierte la respuesta a un diccionario de Python
    data = response.json()

    # Accede a la URL de los datos
    datos_url = data.get('datos', '')

    # Descarga los datos
    datos_response = requests.get(datos_url)
    datos_json = datos_response.json()

    # Convierte a DataFrame
    df = pd.DataFrame(datos_json)

    # Reemplazar comas por puntos en columnas específicas
    columnas_numericas = ['tmed', 'prec', 'tmin', 'tmax']
    df[columnas_numericas] = df[columnas_numericas].replace({',': '.'}, regex=True)
```

```
    # Convertir columnas numéricas a valores numéricos
    df[columnas_numericas] = df[columnas_numericas].apply(lambda x:
pd.to_numeric(x.str.replace(',', '.'), errors='coerce'))



    #Rellenar datos faltantes
    df['fecha'] = pd.to_datetime(df['fecha'])

    df.set_index('fecha', inplace=True)

    missing_values = df [columnas_numericas].isna().sum()
    print("valores faltantes antes de la impputacion")
    print(missing_values)



    for col in ['tmed','tmin','tmax']:
        df[col] = df[col].interpolate(method='linear')
    df['prec']= df['prec'].ffill().bfill()
    df.reset_index(inplace=True)

    final_missing_values = df[columnas_numericas].isna().sum()
    print("\nValores faltantes despues de la imputacion: ")
    print(final_missing_values)



    df['fecha'] = df['fecha'].astype(str)
# Aplicar la función para calcular ETP y agregarla como una nueva columna 'ETP'
    df['ETP'] = df.apply(calcular_etp, axis=1)

    # Guarda el DataFrame en un archivo CSV
    df.to_csv(ruta+resultado, index=False)

    print("Datos guardados correctamente en 'datos_climatologicos_aranjuez.csv'")
else:
    print(f"Error al hacer la solicitud. Código de estado: {response.status_code}")
```



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	fecha	indicativo	nombre	provincia	altitud	tmed	prec	tmin	horatmin	tmax	horatmax	dir	velmedia	racha	horaracha	hrMedia	hrMax	horaHrMax
2	1/05/2023	3100B	ARANJUEZ	MADRID	540	18.3	0	8.2	5:50	28.4	16:30	27	1,4	9,7	12:40	38	69	6:0
3	2/05/2023	3100B	ARANJUEZ	MADRID	540	18.4	0	6.8	5:40	29.9	13:30	10	1,1	7,2	23:10	33	59	5:5
4	3/05/2023	3100B	ARANJUEZ	MADRID	540	24	0	15.2	4:30	32.8	14:50	22	1,9	9,7	16:30	29	46	Varias
5	4/05/2023	3100B	ARANJUEZ	MADRID	540	18.6	0	10.9	23:59	26.3	15:50	25	2,8	9,4	11:10	45	70	6:0
6	5/05/2023	3100B	ARANJUEZ	MADRID	540	18.2	0	7.5	3:10	28.9	Varias	26	1,4	6,7	10:40	40	67	Varias
7	6/05/2023	3100B	ARANJUEZ	MADRID	540	19.6	0	10.1	5:30	29.2	14:40	21	2,5	10,6	13:10	37	60	5:2
8	7/05/2023	3100B	ARANJUEZ	MADRID	540	19.8	0	9.8	5:20	29.8	14:50	35	2,2	11,9	10:10	40	68	5:4
9	8/05/2023	3100B	ARANJUEZ	MADRID	540	20.4	0	11.05		29.8		99	1,1	6,1	Varias	42	71	5:5
10	9/05/2023	3100B	ARANJUEZ	MADRID	540	21	0	12.3	5:20	29.8	14:10	33	3,1	11,4	13:40	42	67	5:4
11	10/05/2023	3100B	ARANJUEZ	MADRID	540	19.4	0	12.1	5:40	26.7	17:20	2	1,9	8,3	20:30	40	73	5:4
12	11/05/2023	3100B	ARANJUEZ	MADRID	540	16.8	0	8.9	5:50	24.6	15:40	99	2,2	8,9	18:40	41	66	5:4
13	12/05/2023	3100B	ARANJUEZ	MADRID	540	13.8	0	6	3:50	21.7	14:00	2	2,2	13,3	9:10	44	67	5:4
14	13/05/2023	3100B	ARANJUEZ	MADRID	540	13.6	0	4.3	5:40	22.8	16:10	1	3,1	9,7	18:10	43	75	5:5
15	14/05/2023	3100B	ARANJUEZ	MADRID	540	14.2	0	5.3	5:20	23.1	15:50	2	1,7	9,2	16:50	47	71	5:4
16	15/05/2023	3100B	ARANJUEZ	MADRID	540	15	0	5.8	5:00	24.2	15:00	34	2,8	10,8	17:50	46	79	6:0
17	16/05/2023	3100B	ARANJUEZ	MADRID	540	14.4	0	3.8	5:20	25.1	16:00	1	1,7	10,3	10:50	39	82	5:4
18	17/05/2023	3100B	ARANJUEZ	MADRID	540	16.7	0	7.5	5:00	25.9	16:40	3	2,8	10,6	20:20	34	60	5:1
19	18/05/2023	3100B	ARANJUEZ	MADRID	540	16	0	8	5:00	24.1	14:00	6	3,1	9,7	23:10	37	62	Varias
20	19/05/2023	3100B	ARANJUEZ	MADRID	540	16.1	0	9.4	5:40	22.8	16:30	9	3,6	10,8	16:10	40	56	Varias
21	20/05/2023	3100B	ARANJUEZ	MADRID	540	15.2	0	7.5	5:50	22.9	14:50	2	1,1	6,7	4:30	41	56	5:3
22	21/05/2023	3100B	ARANJUEZ	MADRID	540	18.5	1	12.1	5:00	24.9	14:00	9	2,2	9,7	14:10	50	63	Varias
23	22/05/2023	3100B	ARANJUEZ	MADRID	540	18.8	0	13.6	5:50	23.9	13:30	10	3,3	13,1	16:00	51	77	23:5
24	23/05/2023	3100B	ARANJUEZ	MADRID	540	18.2	2.3	11.2	5:10	25.1	13:50	11	1,9	13,3	13:50	62	86	Varias
25	24/05/2023	3100B	ARANJUEZ	MADRID	540	19	0.3	13	23:59	25	15:20	10	2,2	10,8	18:10	57	89	Varias
26	25/05/2023	3100B	ARANJUEZ	MADRID	540	18.2	22.5	10.7	2:40	25.8	13:30	28	1.9	10.6	13:40	62	86	Varias

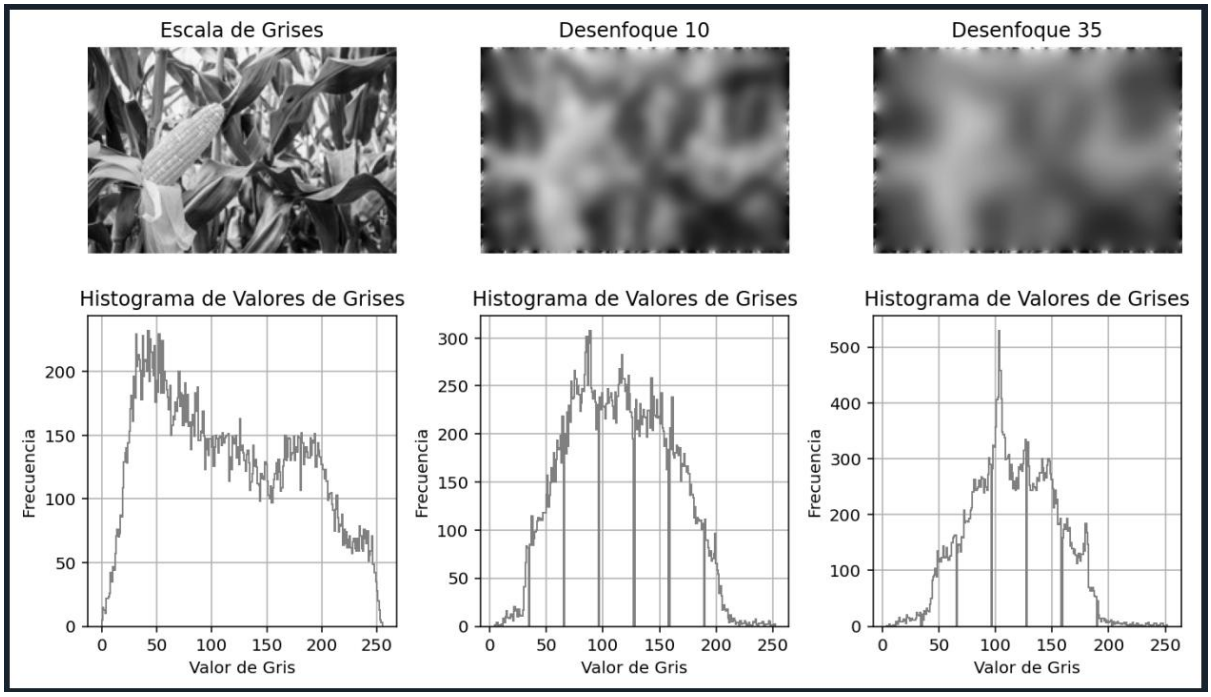
datos_climatologicos_aranjuez

Digital Processing: Efectos de Desenfoque y Análisis de Histogramas

Descripción: Visualización del impacto de filtros de desenfoque progresivo sobre una imagen en escala de grises, analizando la redistribución de frecuencias de píxeles mediante histogramas comparativos.

```
from PIL import Image, ImageFilter
import numpy as np
import matplotlib.pyplot as plt

imagen_pil = Image.open("maiz.jpg")

plt.figure(figsize=(10, 6))

# Escala de grises
imagen_gris = imagen_pil.convert("L")
plt.subplot(2, 3, 1)
plt.imshow(imagen_gris, cmap="gray")
plt.title("Escala de Grises")
plt.axis("off")

# Desenfoque 25
intensidad_desenfoque = 10
imagen_desenfocada = imagen_gris.copy()
for _ in range(intensidad_desenfoque):
    imagen_desenfocada = imagen_desenfocada.filter(ImageFilter.BLUR)

plt.subplot(2, 3, 2)
plt.imshow(imagen_desenfocada,cmap="gray")
plt.title(f"Desenfoque {intensidad_desenfoque}")
plt.axis("off")
```

```
# Desenfoque 35
intensidad_desenfoque = 35
imagen_desenfocada2 = imagen_gris.copy()
for _ in range(intensidad_desenfoque):
    imagen_desenfocada2 = imagen_desenfocada2.filter(ImageFilter.BLUR)

plt.subplot(2, 3, 3)
plt.imshow(imagen_desenfocada2,cmap="gray")
plt.title(f"Desenfoque {intensidad_desenfoque}")
plt.axis("off")

# Histogramas

plt.subplot(2, 3, 4)
plt.hist(np.array(imagen_gris).ravel(), bins=256, color='gray', histtype='step')
plt.title("Histograma de Valores de Grises")
plt.xlabel("Valor de Gris")
plt.ylabel("Frecuencia")
plt.grid(True)

plt.subplot(2, 3, 5)
plt.hist(np.array(imagen_desenfocada).ravel(), bins=256, color='gray', histtype='step')
plt.title("Histograma de Valores de Grises")
plt.xlabel("Valor de Gris")
plt.ylabel("Frecuencia")
plt.grid(True)

plt.subplot(2, 3, 6)
plt.hist(np.array(imagen_desenfocada2).ravel(), bins=256, color='gray', histtype='step')
plt.title("Histograma de Valores de Grises")
plt.xlabel("Valor de Gris")
plt.ylabel("Frecuencia")
plt.grid(True)

# Mostrar todo junto
plt.tight_layout()
plt.show()
```



Machine Learning: Compresión y Análisis de Imágenes mediante PCA

Implementación de un flujo de trabajo en Python que utiliza el Análisis de Componentes Principales (PCA) para reducir la dimensionalidad de datos visuales, permitiendo la reconstrucción de imágenes y el análisis de varianza explicada por canal de color.

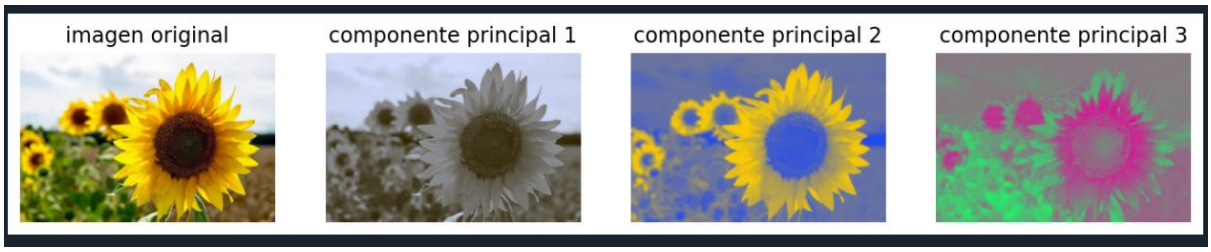
```
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from PIL import Image
import pandas as pd

def pca_image (image_path, num_components=3):
    image = Image.open(image_path)

    image_array = np.array(image)



    height, width, channels = image_array.shape

    flattened_array = image_array.reshape((height*width, channels))

    pca= PCA(n_components = num_components)
    transformed_array = pca.fit_transform(flattened_array)
```

```
    reconstructed_array = pca.inverse_transform(transformed_array)
    reconstructed_image_array =  reconstructed_array.reshape (( height, width,
channels))



    reconstructed_image = Image.fromarray(reconstructed_image_array.astype(np.uint8))

    plt.figure(figsize =(8,4))

    plt.subplot(1,2,1)
    plt.imshow(image)
    plt.title ("imagen original")
    plt.axis("off")



    plt.subplot(1,2,2)
    plt.imshow(reconstructed_image)
    plt.title(f"imagen reconstruida ({num_components} componentes principales)")
    plt.axis("off")

    plt.show()



imagen_path = "C:/Users/Facun/Downloads/girasoles.jpg.webp"
num_componentes_principales = 3
pca_image(imagen_path, num_components=num_componentes_principales)

imagen_path = "C:/Users/Facun/Downloads/girasoles.jpg.webp"
num_componentes_principales = 2
pca_image(imagen_path, num_components=num_componentes_principales)

imagen_path = "C:/Users/Facun/Downloads/girasoles.jpg.webp"
num_componentes_principales = 1
pca_image(imagen_path, num_components=num_componentes_principales)

def show_pca_components (image_path, num_components=3):
    image = Image.open(image_path)

    image_array = np.array(image)

    height, width = image_array.shape[:2]
```

```
    flattened_array = image_array.reshape ((-1, image_array.shape[-1]))


    pca= PCA(n_components = num_components)
    transformed_array = pca.fit_transform(flattened_array)


    plt.figure(figsize =(12,4))


    plt.subplot(1,num_components + 1,1)
    plt.imshow(image)
    plt.title ("imagen original")
    plt.axis("off")


    for i in range(num_components):


        component_image_array = np.dot(pca.transform(flattened_array)[:,i:i+1],
pca.components_[i:i+1])


        component_image_array = (component_image_array -
np.min(component_image_array))/(np.max(component_image_array)-
np.min(component_image_array))*255


        component_image =
Image.fromarray(component_image_array.astype(np.uint8).reshape((height,width,
image_array.shape[-1])))


        plt.subplot(1, num_components+1,i+2)
        plt.imshow(component_image)
        plt.title(f"componente principal {i+1}")
        plt.axis("off")


    plt.show()


imagen_path = "C:/Users/Facun/Downloads/girasoles.jpg.webp"
num_componentes_principales = 3
show_pca_components(imagen_path, num_components=num_componentes_principales)


#TABLA VARIANZA


def pca_varianza_matriz(image_path, num_components=3):
    image = Image.open(image_path)
    image_array = np.array(image)



    flat = image_array.reshape((-1, image_array.shape[-1]))
```

```
    # PCA
    pca = PCA(n_components=num_components)
    pca.fit(flat)

    # Tabla de varianza
    var_table = pd.DataFrame({
        "Componente": [f"PC{i+1}" for i in range(num_components)],
        "Varianza": pca.explained_variance_,
        "Varianza Explicada (%)": pca.explained_variance_ratio_ * 100,
        "Varianza Acumulada (%)": np.cumsum(pca.explained_variance_ratio_ * 100)
    })

    # Matriz PCA (componentes principales)
    pca_matrix = pd.DataFrame(
        pca.components_,
        columns=["Canal R", "Canal G", "Canal B"],
        index=[f"PC{i+1}" for i in range(num_components)]
    )

    return var_table, pca_matrix


imagen_path = "C:/Users/Facun/Downloads/girasoles.jpg.webp"
tabla_varianza, matriz_pca = pca_varianza_matriz(imagen_path, num_components=3)

print("TABLA DE VARIANZA:\n", tabla_varianza)
print("\nMATRIZ PCA (LOADINGS):\n", matriz_pca)



def plot_pca(pca_matrix):
    plt.figure(figsize=(8,5))

    loadings = pca_matrix.values
    components = pca_matrix.index
    channels = pca_matrix.columns

    for i in range(loadings.shape[0]):
        plt.plot(channels, loadings[i], marker="o", label=components[i])

    plt.title("Peso de cada Componente PCA por canal (Loadings)")
```

```
    plt.xlabel("Canales (R, G, B)")
    plt.ylabel("Peso / Contribución")
    plt.grid(True, linestyle="--", alpha=0.5)
    plt.legend()
    plt.tight_layout()
    plt.show()

plot_pca(matriz_pca)
```

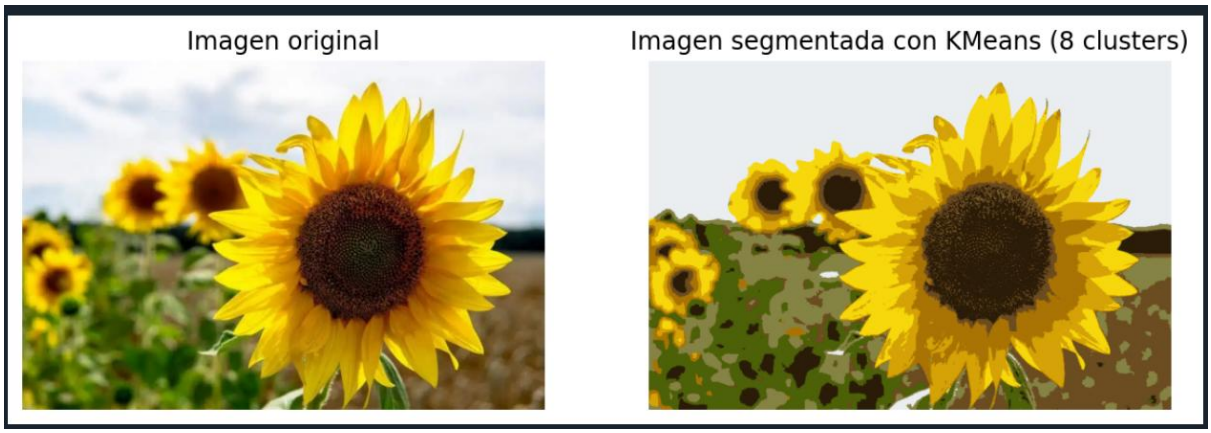


Computer Vision: Segmentación de Color y Cuantificación mediante K-Means

Implementación de un algoritmo de aprendizaje no supervisado para agrupar píxeles por similitud cromática, logrando una reducción inteligente de la paleta de colores y la segmentación de objetos basada en sus propiedades visuales.

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from PIL import Image

def aplicar_kmeans (image_path,num_clusters =8):
    image = Image.open(image_path)
    image_array = np.array(image)
    height, width = image_array.shape [:2]
    flattened_array = image_array.reshape ((-1, image_array.shape[-1]))

    kmeans = KMeans (n_clusters=num_clusters, random_state=42)
    kmeans.fit(flattened_array)

    segmented_image_array = kmeans.cluster_centers_[kmeans.labels_]
    segmented_image_array = segmented_image_array.reshape((height, width,
image_array.shape[-1]))
```

```
    plt.figure(figsize = (10,5))

    plt.subplot(1,2,1)
    plt.imshow(image)
    plt.title("Imagen original")
    plt.axis ("off")



    plt.subplot(1,2,2)
    plt.imshow(segmented_image_array.astype(np.uint8))
    plt.title(f"Imagen segmentada con KMeans ({num_clusters} clusters)")
    plt.axis ("off")

    plt.show()

imagen_path = "C:/Users/Facun/Downloads/girasoles.jpg.webp"
num_clusters_kmeans = 8
aplicar_kmeans(imagen_path, num_clusters=num_clusters_kmeans)
```



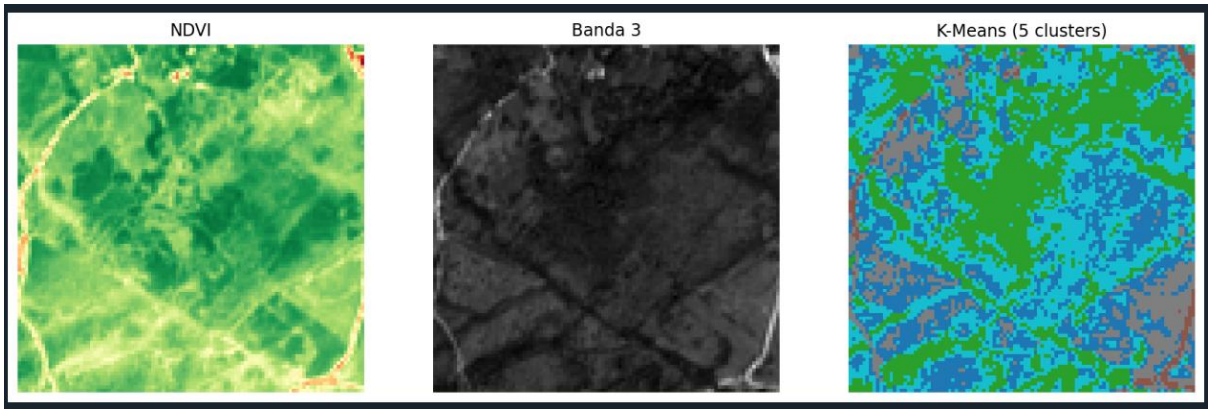
Remote Sensing ML: Clasificación No Supervisada de Suelos y Cultivos con K-Means

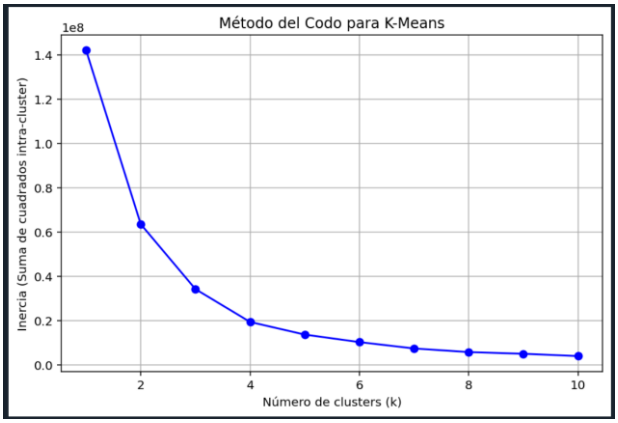
Implementación de un flujo de trabajo para segmentar imágenes satelitales utilizando el algoritmo K-Means. El sistema integra datos de NDVI y bandas espectrales para clasificar coberturas vegetales, optimizando la agrupación mediante el método del codo.

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import rasterio

def calculate_elbow(ndvi_path, b3_path, max_clusters=10):
    """Genera el gráfico del codo para encontrar el número óptimo de clusters"""
    # Leer NDVI
    with rasterio.open(ndvi_path) as src:
        ndvi = src.read(1).astype(np.float32)

    # Leer Banda 3
    with rasterio.open(b3_path) as src:
        b3 = src.read(1).astype(np.float32)

    # Validar tamaño
    if ndvi.shape != b3.shape:
        raise Exception("ERROR: NDVI y B3 tienen dimensiones distintas.")

    # Crear matriz multivariable
    multiband = np.stack([ndvi, b3], axis=-1)
    flat_pixels = multiband.reshape(-1, 2)

    # Calcular inercia para diferentes k
    inertias = []
    cluster_range = range(1, max_clusters + 1)
    for k in cluster_range:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(flat_pixels)
        inertias.append(kmeans.inertia_)

    # Graficar codo
    plt.figure(figsize=(8,5))
    plt.plot(cluster_range, inertias, 'bo-')
    plt.xlabel('Número de clusters (k)')
    plt.ylabel('Inercia (Suma de cuadrados intra-cluster)')
```

```
    plt.title('Método del Codo para K-Means')
    plt.grid(True)
    plt.show()


def apply_kmeans_ndvi_b3(ndvi_path, b3_path, num_clusters):
    """Aplica K-Means usando NDVI + Banda 3"""
    # Leer NDVI
    with rasterio.open(ndvi_path) as src:
        ndvi = src.read(1).astype(np.float32)

    # Leer Banda 3
    with rasterio.open(b3_path) as src:
        b3 = src.read(1).astype(np.float32)

    # Validar tamaño
    if ndvi.shape != b3.shape:
        raise Exception("ERROR: NDVI y B3 tienen dimensiones distintas.")

    height, width = ndvi.shape

    # Crear matriz multivariable
    multiband = np.stack([ndvi, b3], axis=-1)
    flat_pixels = multiband.reshape(-1, 2)

    # Aplicar K-Means
    print(f"Aplicando K-Means con {num_clusters} clusters...")
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)
    kmeans.fit(flat_pixels)

    labels = kmeans.labels_.reshape(height, width)

    # Mostrar resultados
    plt.figure(figsize=(15,5))

    plt.subplot(1,3,1)
    plt.imshow(ndvi, cmap="RdYlGn")
    plt.title("NDVI")
    plt.axis("off")

    plt.subplot(1,3,2)
    plt.imshow(b3, cmap="gray")
```

```
    plt.title("Banda 3")
    plt.axis("off")


    plt.subplot(1,3,3)
    plt.imshow(labels, cmap="tab10")
    plt.title(f"K-Means ({num_clusters} clusters)")
    plt.axis("off")


    plt.show()



#  USO
ndvi_path = r"C:\Users\Facun\OneDrive\Escritorio\Master AP\5.Programacion\Bloque 4\Descargas Sentinel\2024-04-12_NDVI.tif"
b3_path   = r"C:\Users\Facun\OneDrive\Escritorio\Master AP\5.Programacion\Bloque 4\Descargas Sentinel\2024-04-12_B3.B3.tif"


#Graficar codo para decidir el número óptimo de clusters
calculate_elbow(ndvi_path, b3_path, max_clusters=10)


#K-Means con el número elegido
num_clusters = 5
apply_kmeans_ndvi_b3(ndvi_path, b3_path, num_clusters)
```





Comentarios finales

A lo largo de este trabajo, se ha evidenciado cómo la integración de competencias en programación, tecnologías de la computación y gestión de sistemas productivos permite enfrentar problemas complejos de manera eficiente. Gracias a este desarrollo, se consiguieron las siguientes competencias:

- Integrar conocimientos avanzados de programación y otras tecnologías de computación que permitan un aprendizaje continuo, así como una capacidad de adaptación a nuevas situaciones o entornos cambiantes.
- Poseer conocimiento avanzado y ser capaz de desarrollar tecnología en gestión de equipos e instalaciones que se integren en los procesos y sistemas de producción agroalimentaria.
- Poseer conocimiento avanzado y ser capaz de desarrollar tecnología en sistemas de producción vegetal y en sistemas integrados de protección de cultivos.
- Resolución de problemas: capacidad para describir, organizar y analizar los elementos constitutivos de un problema complejo y diseñar estrategias que permitan alcanzar