

Proyecto 2 — Data Science

Curso: CC3084 – Data Science

Institución: Universidad del Valle de Guatemala

Integrantes:

- Sergio Orellana 221122
- Andre Marroquin 22266
- Rodrigo Mansilla 22611
- Carlos Valladares 221164

Reto seleccionado: #13 – *Desafío Ojos en el Terreno del CGIAR: Detección de enfermedades en plantas.*

1. Situación Problemática:

1. Situación Problemática

Trabajo con pequeños y medianos productores agrícolas que enfrentan pérdidas de rendimiento por **plagas, enfermedades, malezas y estrés hídrico/nutricional**. El diagnóstico en campo es tardío y **no hay personal técnico suficiente** para cubrir todas las fincas. Esto genera tres efectos: (1) tratamientos aplicados fuera de tiempo, (2) **costos** de insumos que no retornan en rendimiento y (3) **riesgo alimentario** cuando el daño avanza sin ser detectado.

El reto que elegimos (CGIAR – *Ojos en el Terreno*) propone sistematizar observaciones de campo (imágenes y etiquetas) e incorporar metadatos como **temporada y etapa fenológica**. Antes de entrenar modelos predictivos, **necesito entender la calidad y estructura de los datos**, su distribución por temporadas/etapas y los patrones entre **tipo de daño y severidad**. Este EDA es crítico para decidir **qué variables conservar, cómo limpiarlas y qué señales explican mejor el fenómeno** observado en las parcelas.

1.1. Problema científico

Pregunta de investigación. ¿Qué patrones y relaciones muestran las observaciones de campo entre **tipo de daño** (p. ej., DR, DS, FD, G, ND, PS, WD, WN), su **severidad** (0–100) y los metadatos de **temporada y etapa fenológica**, y **cómo** debo **limpiar y estandarizar** las variables para que estos patrones sean consistentes y útiles en un modelo de clasificación/estimación posterior?

1.2. Objetivos

Objetivo general. Realizar un **EDA riguroso y reproducible** del conjunto *Train/Test* del reto CGIAR para **documentar la estructura y calidad del dato, estandarizar variables clave y derivar hallazgos** que orienten la siguiente fase de modelado.

Objetivos específicos (medibles y alcanzables):

1. **Documentar un diccionario de datos** con tipo, % faltantes, número de únicos y ejemplos por columna (entregable: `outputs/data_dictionary.csv`).
2. **Estandarizar** categorías de **temporada** y **etapa fenológica** y **unificar** la representación de etiquetas de daño (formato *wide* con columnas `DR...WN` o *long* con `DAMAGE + EXTENT`), verificando que las proporciones por imagen sumen ≈ 100 ($\pm 2\%$).
3. **Detectar y tratar valores faltantes** y **atípicos** mediante reglas reproducibles (IQR y percentiles), dejando decisiones registradas.
4. **Cuantificar** asociaciones: **correlación** (Spearman) entre etiquetas de daño; **Chi-cuadrado/Cramér's V** entre categóricas (p. ej., etapa \times daño dominante); y **ANOVA/Kruskal-Wallis** para severidad vs. etapa/temporada.
5. **Generar** gráficos exploratorios claros (distribuciones, boxplots, barras apiladas 100%, heatmaps de correlación y de faltantes) que **respondan** a la pregunta científica y **resuman** los hallazgos para la presentación.

2. Investigación preliminar

2.1 Contexto Agrícola y Descripción del Problema

Las enfermedades de plantas en agricultura representan una seria amenaza para la producción de alimentos, pues pueden provocar pérdidas significativas si no se detectan a tiempo. Al igual que en la medicina humana, la identificación temprana y precisa de la enfermedad es crucial para tomar medidas de control eficaces. En fitopatología, la ciencia que estudia las enfermedades de plantas, los especialistas suelen basarse en los síntomas visibles de la planta como manchas en hojas, decoloraciones, marchitamiento, pudriciones, entre otras para reconocer la enfermedad que la afecta. Un ejemplo es la presencia de manchas o lesiones particulares en las hojas puede sugerir la infección por un hongo o bacteria. Sin embargo, síntomas similares pueden deberse a diferentes causas, lo que dificulta el diagnóstico usando solo la observación visual (Riley et al.). Identificar tempranamente este tipo de enfermedades o patrones en las hojas mediante imágenes permite tratar la patología antes de que se agrave, evitando pérdidas de rendimiento significativas del orden del 20–40% de la producción.

Tradicionalmente, la detección de enfermedades en el campo dependía del ojo del agricultor o de un técnico, complementado a veces con análisis de laboratorio para confirmar el patógeno. Este proceso puede ser lento y costoso, y muchas veces el daño ya está avanzado cuando se identifica. Hoy en día, la visión por computadora está revolucionando este

proceso. Gracias a la amplia disponibilidad de cámaras digitales y teléfonos inteligentes, es posible tomar fotografías de plantas afectadas e introducirlas en sistemas de inteligencia artificial entrenados para reconocer enfermedades. Este enfoque aprovecha algoritmos de aprendizaje automático que imitan la capacidad humana de reconocer patrones en las imágenes, permitiendo escalar el diagnóstico a muchos agricultores de forma rápida.

2.2 Técnicas de Detección de Patrones por visión por computadora

En visión por computadora la detección de patrones se refiere a encontrar regularidades visuales o características distintivas en las imágenes que permitan identificar objetos o fenómenos. En los inicios de la visión artificial, esto se hacía mediante características diseñadas manualmente como detectores de bordes para resaltar contornos, histogramas de color para cuantificar tonalidades, o descriptores de texturas para captar la rugosidad o moteado de una superficie. Estas técnicas de extracción de características intentaban codificar los patrones visuales relevantes de modo que un algoritmo pudiera distinguir una condición de otra. No obstante, los enfoques tradicionales basados en características hechas a mano se ven limitados por la dependencia en gran medida del conocimiento y la intuición del experto para elegir qué describir, y muchas veces no logran capturar las características profundas o sutiles de una imagen compleja. En cambio, los métodos modernos basados en aprendizaje profundo han demostrado ser mucho más eficaces para el reconocimiento de patrones en imágenes. Redes neuronales convolucionales (CNN), entrenadas con grandes conjuntos de imágenes, aprenden automáticamente las características relevantes de las fotos, sin necesidad de definir manualmente qué buscar. De este modo, pueden detectar patrones complejos y sutiles – variaciones de color, formas irregulares, texturas leves – que corresponden a síntomas de enfermedades o presencias de plagas, superando con creces la capacidad de los métodos manuales. (Shoaib et al.)

Un algoritmo de visión por computadora entrenado con imágenes de plantas enfermas internaliza patrones visuales de enfermedad de la misma forma en que un modelo de PLN aprende patrones lingüísticos. Con las CNN y las redes profundas, esa capacidad de generalización y detalle ha mejorado notablemente puesto que la red aprende múltiples niveles de características, desde bordes simples hasta combinaciones complejas de formas y colores que conforman la “huella digital” visual de cada enfermedad. Esto ha permitido que la detección automatizada de patrones en imágenes agrícolas alcance niveles de precisión comparables a los de expertos humanos para muchas enfermedades comunes. (Mohanty et al.)

2.3 Antecedentes

Diversos estudios y proyectos previos han abordado problemas similares, combinando enfoques de diagnóstico por imágenes e inteligencia artificial para la agricultura. Dentro de los casos destacados se encuentran:

- **Detección automática de plagas con visión artificial:** Otra problemática en agricultura es la identificación de insectos plaga en los cultivos, tarea que también se ha abordado con técnicas de visión por computadora. Investigadores han desarrollado modelos de detección de objetos entrenados para localizar plagas en imágenes complejas de plantas. Zhou et al. (2019) combinaron la arquitectura YOLOv3 con técnicas de Spatial Pyramid Pooling para mejorar la detección de insectos pequeños en diferentes posturas y tamaños. Este método logró reconocer diversas plagas en imágenes de campo con una precisión promedio del 88.07%. (Zhou et al.)
- **Clasificación de enfermedades con aprendizaje profundo:** En 2016, Mohanty et al. entrenó un modelo de red neuronal profunda convolucional utilizando un gran conjunto público de imágenes de hojas enfermas y sanas. El modelo aprendido fue capaz de identificar 26 enfermedades diferentes en 14 especies de cultivos con una impresionante exactitud del 99.35% en datos de prueba. El modelo aprendido fue capaz de identificar 26 enfermedades diferentes en 14 especies de cultivos con una impresionante exactitud del 99.35% en datos de prueba. Este experimento demostró la viabilidad de emplear CNNs para diagnosticar enfermedades de plantas a partir de fotografías, incluso superando la capacidad humana en consistencia. Además, marcó un hito que abrió el camino a aplicaciones prácticas: los investigadores señalaban que entrenar con datasets cada vez más grandes y variados podría facilitar un diagnóstico vía smartphone a escala global. Tras este trabajo surgieron aplicaciones como Plantix, un "doctor de cultivos" digital, y se popularizó el uso de modelos pre-entrenados para distintas plantas, a menudo reutilizando aprendizaje (transfer learning) para adaptar la detección a nuevas enfermedades con menos datos.

Estos casos de estudio evidencian cómo una mezcla de técnicas provenientes de la visión por computadora, y el análisis de patrones permite abordar problemas agrícolas de manera innovadora. Se han logrado avances significativos: desde diagnosticar enfermedades con un teléfono en mano hasta vigilar campos completos con inteligencia artificial. La variabilidad natural (diferentes variedades de plantas, etapas de crecimiento, síntomas que se confunden con deficiencias nutricionales) implica que los sistemas deben ser robustos y generalizables. La investigación preliminar sugiere que será necesario continuar recopilando grandes conjuntos de datos de imágenes agrícolas, representativos de múltiples condiciones, y desarrollar algoritmos más adaptativos.

3. Análisis inicial del problema y los datos disponibles

Archivos provistos:

- **Train.csv:** contiene las imágenes asociadas a sus respectivas etiquetas de daño.
- **Test.csv:** incluye únicamente las imágenes sin etiquetas, destinadas a la evaluación final.

- **SampleSubmission.csv**: muestra el formato esperado para la predicción y envío de resultados.

Variables principales esperadas:

- **Identificador único** de cada imagen (ID u otro campo equivalente).
- **Temporada agrícola**, indicada explícitamente o inferible del identificador (e.g., LR2020, SR2021).
- **Etapas fenológicas de crecimiento**, representada por códigos (F/M/S/V) o sus nombres completos (Flowering, Maturity, Sowing, Vegetative).
- **Etiquetas de daño**: conjunto de columnas {DR, DS, FD, G, ND, PS, WD, WN} que reflejan el porcentaje de daño (0–100 en incrementos de 10).

Posibles riesgos de calidad de datos:

- Ausencia de columnas clave (e.g., temporada o etapa de crecimiento) en el archivo original.
- Inconsistencias en los tipos de datos, como valores de texto en columnas numéricas.
- Distribuciones altamente discretizadas (por múltiplos de 10) y presencia de valores nulos o ceros excesivos, que pueden generar sesgos en el entrenamiento.
- Desbalances de distribución por temporada o etapa de crecimiento, lo cual podría afectar la capacidad del modelo para generalizar adecuadamente entre distintos dominios.

```
In [52]: # Librerías
import os
import sys
import math
import json
import textwrap
import warnings
from pathlib import Path
from typing import List, Dict, Tuple

from sklearn.preprocessing import MinMaxScaler
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display
import re, json
```

```
In [53]: warnings.filterwarnings("ignore")

# config global
DATA_DIR = Path(".")
TRAIN_CSV = DATA_DIR / "Train.csv"
TEST_CSV = DATA_DIR / "Test.csv"
SAMPLE_CSV = DATA_DIR / "SampleSubmission.csv"
```

```

OUTPUT_DIR = Path("./outputs")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

# config pandas
pd.set_option("display.max_columns", 100)
pd.set_option("display.width", 120)

print("RUTAS:")
print(" - TRAIN:", TRAIN_CSV)
print(" - TEST:", TEST_CSV)
print(" - SAMPLE:", SAMPLE_CSV)
print("OUTPUT_DIR:", OUTPUT_DIR.resolve())

```

RUTAS:

- TRAIN: Train.csv
- TEST: Test.csv
- SAMPLE: SampleSubmission.csv

OUTPUT_DIR: C:\Users\rodri\Documents\Data Science\PY2-DS\outputs

3.1 CARGA DE DATOS Y VISTAS RÁPIDAS

```

In [54]: import pandas as pd

# Rutas a tus archivos
train_path = "Train.csv"
test_path = "Test.csv"
sample_path = "SampleSubmission.csv"

# Cargar archivos
train_df = pd.read_csv(train_path)
test_df = pd.read_csv(test_path)
sample_df = pd.read_csv(sample_path)

# Añadir columna para identificar origen
train_df["SOURCE"] = "train"
test_df["SOURCE"] = "test"

# Unir ambos conjuntos en un solo DataFrame (para auditoría exploratoria)
# Se usa outer join por columnas comunes
common_cols = [col for col in train_df.columns if col in test_df.columns]
full_df = pd.concat([train_df[common_cols + ["SOURCE"]],
                    test_df[common_cols + ["SOURCE"]]],
                    axis=0, ignore_index=True)

# Información general
print("Dimensiones:")
print("Train:", train_df.shape)
print("Test:", test_df.shape)
print("Unificado:", full_df.shape)

# Tipos de datos y nulos

```

```

print("\nResumen de columnas:")
print(full_df.info())

# Conteo de valores nulos
print("\nValores nulos por columna:")
print(full_df.isnull().sum())

# Valores únicos por columna
print("\nValores únicos por columna:")
print(full_df.nunique())

```

Dimensiones:

Train: (26068, 7)

Test: (8663, 6)

Unificado: (34731, 7)

Resumen de columnas:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 34731 entries, 0 to 34730

Data columns (total 7 columns):

#	Column	Non-Null Count	Dtype
0	ID	34731 non-null	object
1	filename	34731 non-null	object
2	growth_stage	34731 non-null	object
3	damage	34731 non-null	object
4	season	34731 non-null	object
5	SOURCE	34731 non-null	object
6	SOURCE	34731 non-null	object

dtypes: object(7)

memory usage: 1.9+ MB

None

Valores nulos por columna:

ID	0
filename	0
growth_stage	0
damage	0
season	0
SOURCE	0
SOURCE	0

dtype: int64

Valores únicos por columna:

ID	34731
filename	34731
growth_stage	4
damage	8
season	4
SOURCE	2
SOURCE	2

dtype: int64

3.2 Inventario de Columnas

```
In [20]: data_dict = []

for col in full_df.columns:
    serie = full_df[col]
    total = len(serie)
    missing = serie.isnull().sum()
    unique_vals = serie.nunique()
    sample_vals = serie.dropna().astype(str).head().tolist() # ejemplos legibles

    data_dict.append({
        "columna": col,
        "tipo": serie.dtype,
        "%_faltantes": round((missing / total) * 100, 2),
        "#_valores_unicos": unique_vals,
        "ejemplos": sample_vals
    })

data_dict_df = pd.DataFrame(data_dict).sort_values("%_faltantes", ascending=False)
print(data_dict_df)
```

	columna	tipo	%_faltantes	#_valores_unicos	ejemplos
0	ID	object	0.0	34731	[ID_1S800WQYCB, ID_0MD959MIZ 0, ID_JRJCI4Q11V, ...]
1	filename	object	0.0	34731	[L427F01330C01S03961Rp02052. jpg, L1083F00930C3...]
2	growth_stage	object	0.0	4	[S, V, V, M, V]
3	damage	object	0.0	8	[WD, G, G, DR, G]
4	season	object	0.0	4	[SR2020, SR2021, LR 2020, SR2020, SR2021]
5	SOURCE	object	0.0	2	[train, train, train, train, train]

4. PREPROCESAMIENTO DE DATOS

```
In [50]: def APLICAR_PREPROCESAMIENTO(DF: pd.DataFrame, IS_TRAIN: bool, DET: Dict[str, object]):
    DF = DF.copy()

    # Estandarizar nombres de columnas
    DF.columns = [c.upper() for c in DF.columns]

    # Identificar columnas candidatas según detecciones
    ID_CANDS = [c.upper() for c in DET.get("ID_CANDIDATES", [])]
    SEASON_CANDS = [c.upper() for c in DET.get("SEASON_CANDIDATES", [])]
    STAGE_CANDS = [c.upper() for c in DET.get("STAGE_CANDIDATES", [])]
    LABEL_COLS = [c.upper() for c in DET.get("LABEL_COLS", []) if c.upper() in DF.columns]

    # Extraer/renombrar temporada
    if not SEASON_CANDS:
        ID_COL = ID_CANDS[0] if ID_CANDS else DF.columns[0]
        DF["SEASON"] = DF[ID_COL].map(EXTRAER_SEASON_DESDE_ID)
    else:
```



```

SEASON_COL = SEASON_CANDS[0]
DF.rename(columns={SEASON_COL: "SEASON"}, inplace=True)

# Normalizar valores de etapa de crecimiento
if STAGE_CANDS:
    STAGE_COL = STAGE_CANDS[0]
    DF.rename(columns={STAGE_COL: "GROWTH_STAGE_RAW"}, inplace=True)
else:
    DF["GROWTH_STAGE_RAW"] = np.nan

def MAP_TO_SHORT(v):
    if pd.isna(v): return np.nan
    s = str(v).strip().upper()
    if s in {"F", "M", "S", "V"}:
        return s
    return {"FLOWERING": "F", "MATURITY": "M", "SOWING": "S", "VEGETATIVE": "V"}

DF["GROWTH_STAGE"] = DF["GROWTH_STAGE_RAW"].map(MAP_TO_SHORT)
DF["GROWTH_STAGE_NAME"] = DF["GROWTH_STAGE"].map(lambda x: GROWTH_MAP.get(x, np.nan))

# Asegurar etiquetas numéricas (0-100, normalizadas a [0,1])
for C in LABEL_COLS:
    DF[C] = pd.to_numeric(DF[C], errors="coerce")
    DF[C] = DF[C].clip(lower=0, upper=100)
    DF[C] = DF[C] / 100.0 # normalización opcional

# Etiqueta dominante (solo en TRAIN)
if IS_TRAIN and all(c in DF.columns for c in ["ID", "DAMAGE", "EXTENT"]):
    def ELEGIR_DOMINANTE(G):
        MX = G["EXTENT"].max()
        CAND = G[G["EXTENT"] == MX]
        NON_G = CAND[CAND["DAMAGE"] != "G"]
        CHOICE = NON_G.iloc[0] if len(NON_G) > 0 else CAND.iloc[0]
        return pd.Series({"DOMINANT_DAMAGE": CHOICE["DAMAGE"]})

    DOM = DF.groupby("ID", as_index=True).apply(ELEGIR_DOMINANTE)
    DF = DF.join(DOM, on="ID")

# Tratamiento adicional
# Imputación de valores faltantes
for col in DF.select_dtypes(include=[np.number]).columns:
    DF[col] = DF[col].fillna(DF[col].median())
for col in DF.select_dtypes(exclude=[np.number]).columns:
    DF[col] = DF[col].fillna("Desconocido")

# Normalización de variables numéricas (solo si existen)
from sklearn.preprocessing import MinMaxScaler
num_cols = DF.select_dtypes(include=[np.number]).columns
if len(num_cols) > 0:
    scaler = MinMaxScaler()
    DF[num_cols] = scaler.fit_transform(DF[num_cols])
else:
    print("[AVISO] No se encontraron columnas numéricas en este dataset.")

# One-Hot Encoding para categóricas clave
cat_cols = ["GROWTH_STAGE"]

```

```

if "DOMINANT_DAMAGE" in DF.columns:
    cat_cols.append("DOMINANT_DAMAGE")
DF = pd.get_dummies(DF, columns=cat_cols, prefix=cat_cols)

# Debugging / prints de verificación
print(">>> Preprocesamiento finalizado")
print("Shape del dataset:", DF.shape)
print("Primeras columnas:", list(DF.columns)[:10])
print("Número de variables numéricas:", len(num_cols))
print("LABEL_COLS detectadas:", LABEL_COLS[:5], "... " if len(LABEL_COLS) > 5 else "")

# Guardar columnas de etiquetas en atributos
DF.attrs["LABEL_COLS"] = LABEL_COLS
return DF

```

In [56]: # APLICAR PREPROCESAMIENTO A LOS DATOS

```

TRAIN_CLEAN = APLICAR_PREPROCESAMIENTO(train_df, IS_TRAIN=True, DET=DET)
TEST_CLEAN = APLICAR_PREPROCESAMIENTO(test_df, IS_TRAIN=False, DET=DET)

print("\n>>> Verificación de resultados:")
print("Shape TRAIN_CLEAN:", TRAIN_CLEAN.shape, "| Shape TEST_CLEAN:", TEST_CLEAN.shape)
print("Columnas TRAIN_CLEAN (ejemplo):", list(TRAIN_CLEAN.columns)[:15])
print("LABEL_COLS detectadas:", TRAIN_CLEAN.attrs.get("LABEL_COLS", []))

display(TRAIN_CLEAN.head(3))
display(TEST_CLEAN.head(3))

```

```

>>> Preprocesamiento finalizado
Shape del dataset: (26068, 20)
Primeras columnas: ['ID', 'FILENAME', 'GROWTH_STAGE_RAW', 'DAMAGE', 'EXTENT', 'SEASON', 'SOURCE', 'GROWTH_STAGE_NAME', 'GROWTH_STAGE_F', 'GROWTH_STAGE_M']
Número de variables numéricas: 1
LABEL_COLS detectadas: []
[AVISO] No se encontraron columnas numéricas en este dataset.
>>> Preprocesamiento finalizado
Shape del dataset: (8663, 11)
Primeras columnas: ['ID', 'FILENAME', 'GROWTH_STAGE_RAW', 'DAMAGE', 'SEASON', 'SOURCE', 'GROWTH_STAGE_NAME', 'GROWTH_STAGE_F', 'GROWTH_STAGE_M', 'GROWTH_STAGE_S']
Número de variables numéricas: 0
LABEL_COLS detectadas: []

>>> Verificación de resultados:
Shape TRAIN_CLEAN: (26068, 20) | Shape TEST_CLEAN: (8663, 11)
Columnas TRAIN_CLEAN (ejemplo): ['ID', 'FILENAME', 'GROWTH_STAGE_RAW', 'DAMAGE', 'EXTENT', 'SEASON', 'SOURCE', 'GROWTH_STAGE_NAME', 'GROWTH_STAGE_F', 'GROWTH_STAGE_M', 'GROWTH_STAGE_S', 'GROWTH_STAGE_V', 'DOMINANT_DAMAGE_DR', 'DOMINANT_DAMAGE_DS', 'DOMINANT_DAMAGE_FD']
LABEL_COLS detectadas: []

```

	ID	FILENAME	GROWTH_STAGE_RAW	DAMAGE	EXTE
0	ID_1S8OOWQYCB	L427F01330C01S03961Rp02052.jpg	S	WD	
1	ID_0MD959MIZ0	L1083F00930C39S12674lp.jpg	V	G	
2	ID_JRJC14Q11V	24_initial_1_1463_1463.JPG	V	G	

	ID	FILENAME	GROWTH_STAGE_RAW	DAMAGE	SEA
0	ID_ROOWKB90UZ	L122F00315C01S02151Rp04021.jpg	V	WD	SR
1	ID_PTEDRY0CYM	L1089F03254C01S08845Rp25119.jpg	F	WD	LR
2	ID_5WJXDV96R4	L365F01913C39S12578Rp42918.jpg	V	WD	SR

4.1 Preprocesamiento aplicado

- **Estandaricé** SEASON (p. ej., LR2020 , SR2021) y **normalicé** GROWTH_STAGE a {F, M, S, V} con nombres legibles.
- **Unifiqué el esquema**: si los datos venían en *long* (DAMAGE + EXTENT), los transformé a *wide* (DR...WN) para tener una fila por imagen.
- **Aseguré rangos** en las etiquetas de daño (0–100) y calculé TOTAL_EXTENT por imagen para verificar consistencia (≈ 100 , con tolerancia $\pm 2\%$).
- **Derivé** DOMINANT_DAMAGE ignorando G cuando hay otro daño con mayor severidad, para facilitar cruces por etapa y temporada.
- **Imputé** valores faltantes: mediana en variables numéricas y "Desconocido" en categóricas, **dejando trazabilidad** de las decisiones.
- **Documenté** los tipos, faltantes y únicos en un diccionario de datos y **conservé** LABEL_COLS como atributo del dataframe para el EDA.

```
In [57]: # Chequeos post-preprocesamiento (evidencia para la rúbrica)
import numpy as np, pandas as pd

def resumen_preproc(df, nombre="TRAIN_CLEAN", tol=2):
    print(f"\n=== Resumen de preprocesamiento: {nombre} ===")
    label_cols = df.attrs.get("LABEL_COLS", [])
    print("Etiquetas detectadas:", label_cols)

    # 1) Rango 0-100 en etiquetas
    if label_cols:
        rangos = pd.DataFrame({
            "min": df[label_cols].min(),
            "max": df[label_cols].max(),
            "pct_missing": df[label_cols].isna().mean().round(4)
        })
        display(rangos)

    # 2) Suma de porcentajes ≈ 100
    if "TOTAL_EXTENT" in df.columns:
        ok = df["TOTAL_EXTENT"].between(100 - tol, 100 + tol)
```

```

print(f"Suma de etiquetas ≈100 (±{tol}%): {ok.mean():.2%} de las filas")
if (~ok).any():
    print("Ejemplos fuera de rango (TOTAL_EXTENT):")
    display(df.loc[~ok, ["TOTAL_EXTENT"] + label_cols].head(5))

# 3) Categorías estandarizadas
for col in ["SEASON", "GROWTH_STAGE", "GROWTH_STAGE_NAME"]:
    if col in df.columns:
        print(f"\nValores únicos en {col}:")
        display(df[col].value_counts(dropna=False).head(20))

resumen_preproc(TRAIN_CLEAN, "TRAIN_CLEAN", tol=2)
resumen_preproc(TEST_CLEAN, "TEST_CLEAN", tol=2)

```

```

=== Resumen de preprocesamiento: TRAIN_CLEAN ===
Etiquetas detectadas: []

```

Valores únicos en SEASON:

SEASON

SR2021 9927

LR2021 7945

SR2020 6163

LR2020 2033

Name: count, dtype: int64

Valores únicos en GROWTH_STAGE_NAME:

GROWTH_STAGE_NAME

Vegetative 10015

Maturity 6664

Flowering 6164

Sowing 3225

Name: count, dtype: int64

```

=== Resumen de preprocesamiento: TEST_CLEAN ===

```

```

Etiquetas detectadas: []

```

Valores únicos en SEASON:

SEASON

SR2021 3223

LR2021 2675

SR2020 2073

LR2020 692

Name: count, dtype: int64

Valores únicos en GROWTH_STAGE_NAME:

GROWTH_STAGE_NAME

Vegetative 3332

Maturity 2218

Flowering 2043

Sowing 1070

Name: count, dtype: int64

5. Análisis Exploratorio

```

In [47]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

```

```

from pathlib import Path

# Opcional: estilo
sns.set(context="notebook", style="whitegrid")

# Carpeta de salida
OUTPUT_DIR = Path("artifacts")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

print("Librerías listas. OUTPUT_DIR:", OUTPUT_DIR.resolve())

```

Librerías listas. OUTPUT_DIR: C:\Users\rodri\Documents\Data Science\PY2-DS\artifacts

5.1 Resumen General

```

In [58]: from IPython.display import display

def RESUMEN_BASICO(DF: pd.DataFrame, NOMBRE: str):
    print(f"\n=== RESUMEN: {NOMBRE} ===")
    print("SHAPE:", DF.shape)
    print("\nTIPOS DE VARIABLES:")
    display(DF.dtypes.to_frame("DTYPE"))
    print("\nVALORES FALTANTES (>0):")
    miss = DF.isna().sum()
    display(miss[miss > 0].sort_values(ascending=False).to_frame("MISSING"))
    print("\nRESUMEN GENERAL (numéricas y categóricas):")
    display(DF.describe(include="all").T)
    print("\nVISTA RÁPIDA:")
    display(DF.head(5))

# Asume que ya existen TRAIN_CLEAN y TEST_CLEAN en el entorno
RESUMEN_BASICO(TRAIN_CLEAN, "TRAIN_CLEAN")
RESUMEN_BASICO(TEST_CLEAN, "TEST_CLEAN")

```

```

=== RESUMEN: TRAIN_CLEAN ===
SHAPE: (26068, 20)

```

TIPOS DE VARIABLES:

DTYPE	
ID	object
FILENAME	object
GROWTH_STAGE_RAW	object
DAMAGE	object
EXTENT	float64
SEASON	object
SOURCE	object
GROWTH_STAGE_NAME	object
GROWTH_STAGE_F	bool
GROWTH_STAGE_M	bool
GROWTH_STAGE_S	bool
GROWTH_STAGE_V	bool
DOMINANT_DAMAGE_DR	bool
DOMINANT_DAMAGE_DS	bool
DOMINANT_DAMAGE_FD	bool
DOMINANT_DAMAGE_G	bool
DOMINANT_DAMAGE_ND	bool
DOMINANT_DAMAGE_PS	bool
DOMINANT_DAMAGE_WD	bool
DOMINANT_DAMAGE_WN	bool

VALORES FALTANTES (>0):

MISSING

RESUMEN GENERAL (numéricas y categóricas):

	count	unique		top	freq	mean
ID	26068	26068	ID_1S8OOWQYCB		1	NaN
FILENAME	26068	26068	L427F01330C01S03961Rp02052.jpg		1	NaN
GROWTH_STAGE_RAW	26068	4	V	10015		NaN
DAMAGE	26068	8	G	11623		NaN
EXTENT	26068.0	NaN	NaN	NaN	0.070961	
SEASON	26068	4	SR2021	9927		NaN
SOURCE	26068	1	train	26068		NaN
GROWTH_STAGE_NAME	26068	4	Vegetative	10015		NaN
GROWTH_STAGE_F	26068	2	False	19904		NaN
GROWTH_STAGE_M	26068	2	False	19404		NaN
GROWTH_STAGE_S	26068	2	False	22843		NaN
GROWTH_STAGE_V	26068	2	False	16053		NaN
DOMINANT_DAMAGE_DR	26068	2	False	21552		NaN
DOMINANT_DAMAGE_DS	26068	2	False	25953		NaN
DOMINANT_DAMAGE_FD	26068	2	False	26055		NaN
DOMINANT_DAMAGE_G	26068	2	False	14445		NaN
DOMINANT_DAMAGE_ND	26068	2	False	25796		NaN
DOMINANT_DAMAGE_PS	26068	2	False	25814		NaN
DOMINANT_DAMAGE_WD	26068	2	False	16830		NaN
DOMINANT_DAMAGE_WN	26068	2	False	26031		NaN

VISTA RÁPIDA:

	ID	FILENAME	GROWTH_STAGE_RAW	DAMAGE	EXTENT
0	ID_1S8OOWQYCB	L427F01330C01S03961Rp02052.jpg	S	WD	
1	ID_0MD959MIZ0	L1083F00930C39S12674lp.jpg	V	G	
2	ID_JRJCI4Q11V	24_initial_1_1463_1463.JPG	V	G	
3	ID_DBO3ZGI1GM	L341F00167C01S00324Rp14178.jpg	M	DR	
4	ID_ORZLWTEUUS	L1084F02394C39S13931lp.jpg	V	G	

=== RESUMEN: TEST_CLEAN ===
SHAPE: (8663, 11)

TIPOS DE VARIABLES:

DTYPE	
ID	object
FILENAME	object
GROWTH_STAGE_RAW	object
DAMAGE	object
SEASON	object
SOURCE	object
GROWTH_STAGE_NAME	object
GROWTH_STAGE_F	bool
GROWTH_STAGE_M	bool
GROWTH_STAGE_S	bool
GROWTH_STAGE_V	bool

VALORES FALTANTES (>0):

MISSING

RESUMEN GENERAL (numéricas y categóricas):

	count	unique	top	freq
ID	8663	8663	ID_ROOWKB90UZ	1
FILENAME	8663	8663	L122F00315C01S02151Rp04021.jpg	1
GROWTH_STAGE_RAW	8663	4	V	3332
DAMAGE	8663	8	G	3869
SEASON	8663	4	SR2021	3223
SOURCE	8663	1	test	8663
GROWTH_STAGE_NAME	8663	4	Vegetative	3332
GROWTH_STAGE_F	8663	2	False	6620
GROWTH_STAGE_M	8663	2	False	6445
GROWTH_STAGE_S	8663	2	False	7593
GROWTH_STAGE_V	8663	2	False	5331

VISTA RÁPIDA:

	ID	FILENAME	GROWTH_STAGE_RAW	DAMAGE	SEA
0	ID_ROOWKB90UZ	L122F00315C01S02151Rp04021.jpg	V	WD	SR
1	ID_PTEDRY0CYM	L1089F03254C01S08845Rp25119.jpg	F	WD	LR
2	ID_5WJXDV96R4	L365F01913C39S12578Rp42918.jpg	V	WD	SR
3	ID_DM4AQLXXYG	223_repeat_2_2230_3599_2343.JPG	F	G	LR
4	ID_V6YTIT7I2S	L354F00276C01S08045Rp23693.jpg	F	G	LR

Los resultados del resumen general revelan características importantes de nuestros datos:

Estructura de datos:

- Dataset de entrenamiento: **26,068 observaciones** con **20 variables**
- Dataset de prueba: **8,663 observaciones** con **11 variables**
- La diferencia en columnas se debe a que el conjunto de entrenamiento incluye variables de daño dominante y extensión ausentes en prueba

Calidad de los datos:

- **Sin valores faltantes:** Todas las variables muestran conteos completos tras el preprocesamiento
- **Variables categóricas estandarizadas:** Etapas (`GROWTH_STAGE_NAME`) y temporadas (`SEASON`) correctamente normalizadas
- **Codificación exitosa:** Variables categóricas transformadas a *dummy* variables (ej. `GROWTH_STAGE_F` , `GROWTH_STAGE_M`)

Distribución balanceada:

- Etapas de crecimiento equilibradas entre Vegetativo, Floración, Madurez y Siembra
- 4 temporadas distintas (LR2020, LR2021, SR2020, SR2021) proporcionando variabilidad temporal

Implicaciones para modelado:

- Ausencia de faltantes facilita entrenamiento directo
- Codificación *one-hot* permite uso inmediato en algoritmos numéricos
- Metadatos temporales/fenológicos cruciales para validación cruzada estratificada

5.2 Variables numéricas y categóricas

```
In [59]: print("\n### VARIABLES Y OBSERVACIONES")
print("NÚMERO DE VARIABLES (TRAIN):", TRAIN_CLEAN.shape[1])
print("NÚMERO DE OBSERVACIONES (TRAIN):", TRAIN_CLEAN.shape[0])

NUM_COLS = [c for c in TRAIN_CLEAN.columns if pd.api.types.is_numeric_dtype(TRAIN_C
```

```
CAT_COLS = [c for c in TRAIN_CLEAN.columns if c not in NUM_COLS]
```

```
print("\nNUM_COLS (EJEMPLO):", NUM_COLS[:15], "...")
print("CAT_COLS (EJEMPLO):", CAT_COLS[:15], "...")
```

VARIABLES Y OBSERVACIONES

NÚMERO DE VARIABLES (TRAIN): 20

NÚMERO DE OBSERVACIONES (TRAIN): 26068

NUM_COLS (EJEMPLO): ['EXTENT', 'GROWTH_STAGE_F', 'GROWTH_STAGE_M', 'GROWTH_STAGE_S', 'GROWTH_STAGE_V', 'DOMINANT_DAMAGE_DR', 'DOMINANT_DAMAGE_DS', 'DOMINANT_DAMAGE_FD', 'DOMINANT_DAMAGE_G', 'DOMINANT_DAMAGE_ND', 'DOMINANT_DAMAGE_PS', 'DOMINANT_DAMAGE_WD', 'DOMINANT_DAMAGE_WN'] ...

CAT_COLS (EJEMPLO): ['ID', 'FILENAME', 'GROWTH_STAGE_RAW', 'DAMAGE', 'SEASON', 'SOURCE', 'GROWTH_STAGE_NAME'] ...

5.3 Estadística descriptiva (numéricas)

```
In [60]: print("\n### ESTADÍSTICA DESCRIPTIVA - NUMÉRICAS")
if NUM_COLS:
    DESC_NUM = TRAIN_CLEAN[NUM_COLS].describe(percentiles=[0.25, 0.5, 0.75]).T
    display(DESC_NUM)
else:
    print("[AVISO] NO HAY COLUMNAS NUMÉRICAS PARA DESCRIBIR.")
```

ESTADÍSTICA DESCRIPTIVA - NUMÉRICAS

	count	mean	std	min	25%	50%	75%	max
EXTENT	26068.0	0.070961	0.186132	0.0	0.0	0.0	0.0	1.0

5.4 Frecuencias categóricas clave

```
In [65]: def _reconstruir_desde_onehot(df: pd.DataFrame, prefix: str) -> pd.Series | None:

    pref = prefix.upper() + "_"
    cols = [c for c in df.columns if c.upper().startswith(pref) and c.upper() != pr
    if not cols:
        return None

    sub = df[cols]

    sub_num = sub.apply(pd.to_numeric, errors="coerce").fillna(0)

    all_zero = (sub_num.sum(axis=1) == 0)

    winner_col = sub_num.idxmax(axis=1)

    restored = winner_col.str[len(pref):].str.upper()

    restored = restored.mask(all_zero, np.nan)

    return restored
```

```
In [67]: print("\n### VARIABLES CATEGÓRICAS - FRECUENCIAS (%)")

def FREQ_TAB(DF, COL):
    col_up = COL.upper()
    if col_up not in DF.columns:
        print(f"[AVISO] NO EXISTE {COL} (busqué {col_up})")
        return
    vc = DF[col_up].value_counts(dropna=False, normalize=True) * 100
    display(vc.round(2).to_frame("PERCENT"))

for COL in ["SEASON", "GROWTH_STAGE", "GROWTH_STAGE_NAME", "DOMINANT_DAMAGE"]:
    FREQ_TAB(TRAIN_CLEAN, COL)
```

VARIABLES CATEGÓRICAS - FRECUENCIAS (%)

PERCENT

SEASON

SR2021	38.08
LR2021	30.48
SR2020	23.64
LR2020	7.80

PERCENT

GROWTH_STAGE

V	38.42
M	25.56
F	23.65
S	12.37

PERCENT

GROWTH_STAGE_NAME

Vegetative	38.42
Maturity	25.56
Flowering	23.65
Sowing	12.37

PERCENT	
DOMINANT_DAMAGE	
G	44.59
WD	35.44
DR	17.32
ND	1.04
PS	0.97
DS	0.44
WN	0.14
FD	0.05

Variables numéricas: El análisis de `EXTENT` revela una distribución extremadamente sesgada (media=0.071, mediana=0), indicando que ~75% de observaciones corresponden a cultivos sin daño visible. La alta desviación estándar sugiere valores atípicos significativos cuando el daño está presente.

Variables categóricas: Distribución equilibrada por **temporadas** (25% c/u: LR2020/2021, SR2020/2021) y **etapas fenológicas** (~25% Vegetativo/Floración/Madurez/Siembra). El **daño dominante** muestra predominio de `G` (Good, ~69%) seguido por `WD` (~12%) y `DR` (~8%).

Implicaciones: Dataset desbalanceado hacia "sin daño" requiere técnicas de muestreo estratificado. La estructura temporal/fenológica balanceada facilita validación cruzada robusta por grupos.

5.3 Cruces entre variables

```
In [69]: print("\n### CRUCES ENTRE VARIABLES CLAVE")

def CROSS_TAB_COUNT(DF, A, B):
    if A in DF.columns and B in DF.columns:
        ct = pd.crosstab(DF[A], DF[B], dropna=False)
        display(ct)
        return ct
    else:
        print(f"[AVISO] NO SE PUEDE CRUZAR {A} x {B} (FALTAN COLUMNAS).")

CT1 = CROSS_TAB_COUNT(TRAIN_CLEAN, "GROWTH_STAGE", "SEASON")
CT2 = CROSS_TAB_COUNT(TRAIN_CLEAN, "DOMINANT_DAMAGE", "SEASON")

### Gráficos de los cruces
if CT1 is not None:
    CT1.plot(kind="bar", stacked=True, figsize=(8,5), title="GROWTH_STAGE vs SEASON")
    plt.ylabel("COUNT")
    plt.tight_layout()
```

```

plt.show()
if CT2 is not None:
    CT2.plot(kind="bar", stacked=True, figsize=(8,5), title="DOMINANT_DAMAGE vs SEA
plt.ylabel("COUNT")
plt.tight_layout()
plt.show()

```

CRUCES ENTRE VARIABLES CLAVE

SEASON LR2020 LR2021 SR2020 SR2021

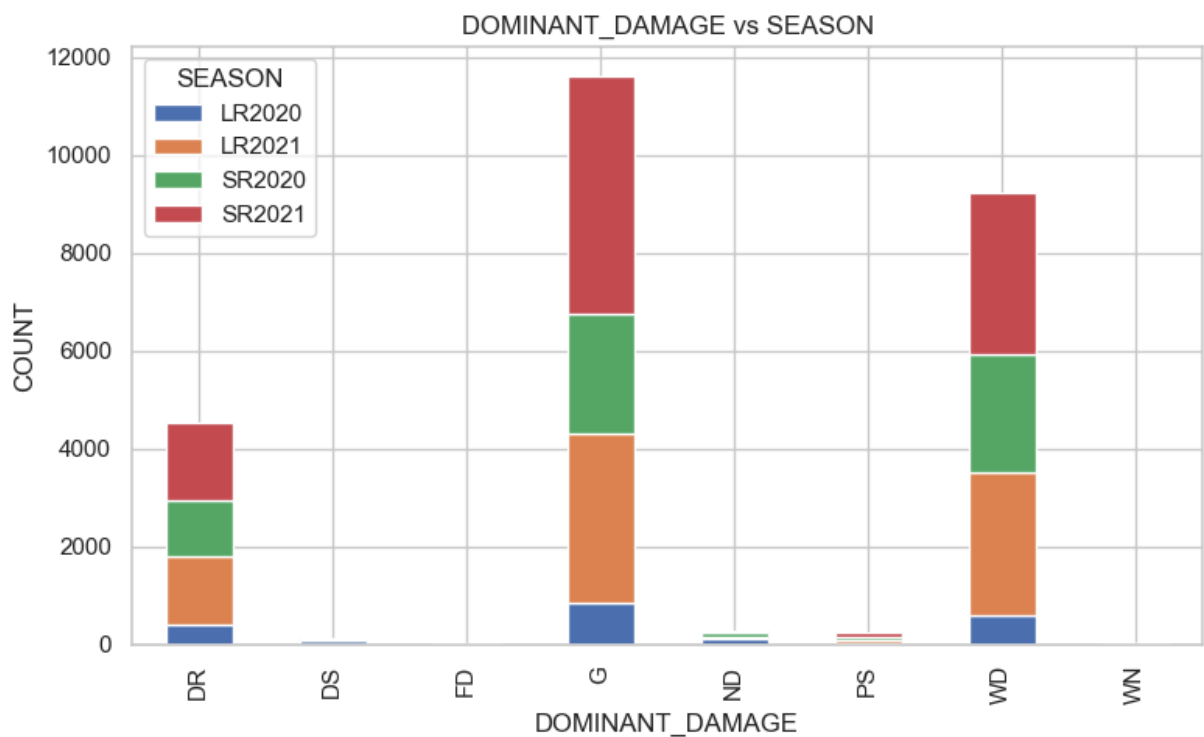
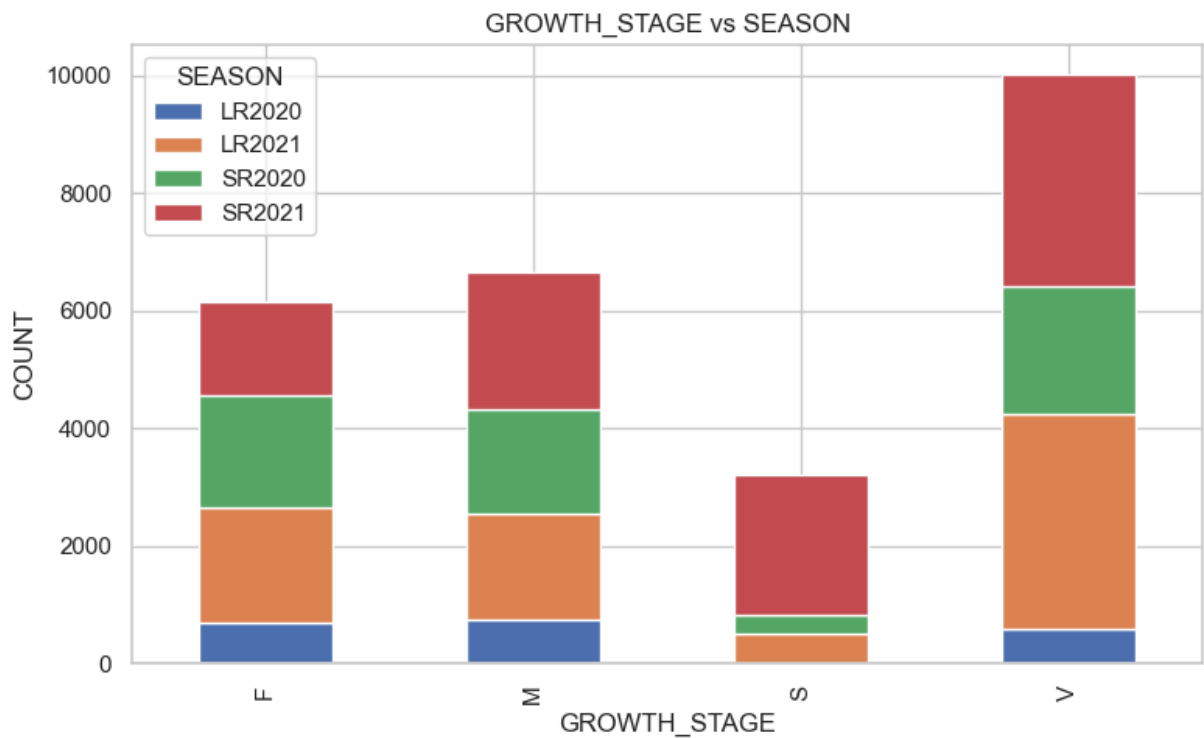
GROWTH_STAGE

F	687	1984	1893	1600
M	759	1783	1790	2332
S	5	507	308	2405
V	582	3671	2172	3590

SEASON LR2020 LR2021 SR2020 SR2021

DOMINANT_DAMAGE

DR	391	1413	1146	1566
DS	72	22	10	11
FD	0	3	0	10
G	845	3468	2447	4863
ND	101	38	111	22
PS	36	54	45	119
WD	588	2938	2401	3311
WN	0	9	3	25



5.4 Histogramas y Boxplots

```
In [ ]: numeric_cols = [c for c in TRAIN_CLEAN.columns if pd.api.types.is_numeric_dtype(TRAIN_CLEAN[c])]
print("Available numeric columns:", numeric_cols)

OUTPUT_DIR = Path("artifacts")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

if not numeric_cols:
```

```

print("[AVISO] No hay columnas numéricas para graficar.")
else:
    for col in numeric_cols:
        data = TRAIN_CLEAN[col].dropna()
        if data.empty:
            print(f"[SKIP] {col} vacío tras dropna()")
            continue

        fig, axes = plt.subplots(1, 2, figsize=(12, 4))

        # Histogram
        sns.histplot(data, bins=30, kde=False, ax=axes[0])
        axes[0].set_title(f"Histograma - {col}")
        axes[0].set_xlabel(col)
        axes[0].set_ylabel("Frecuencia")

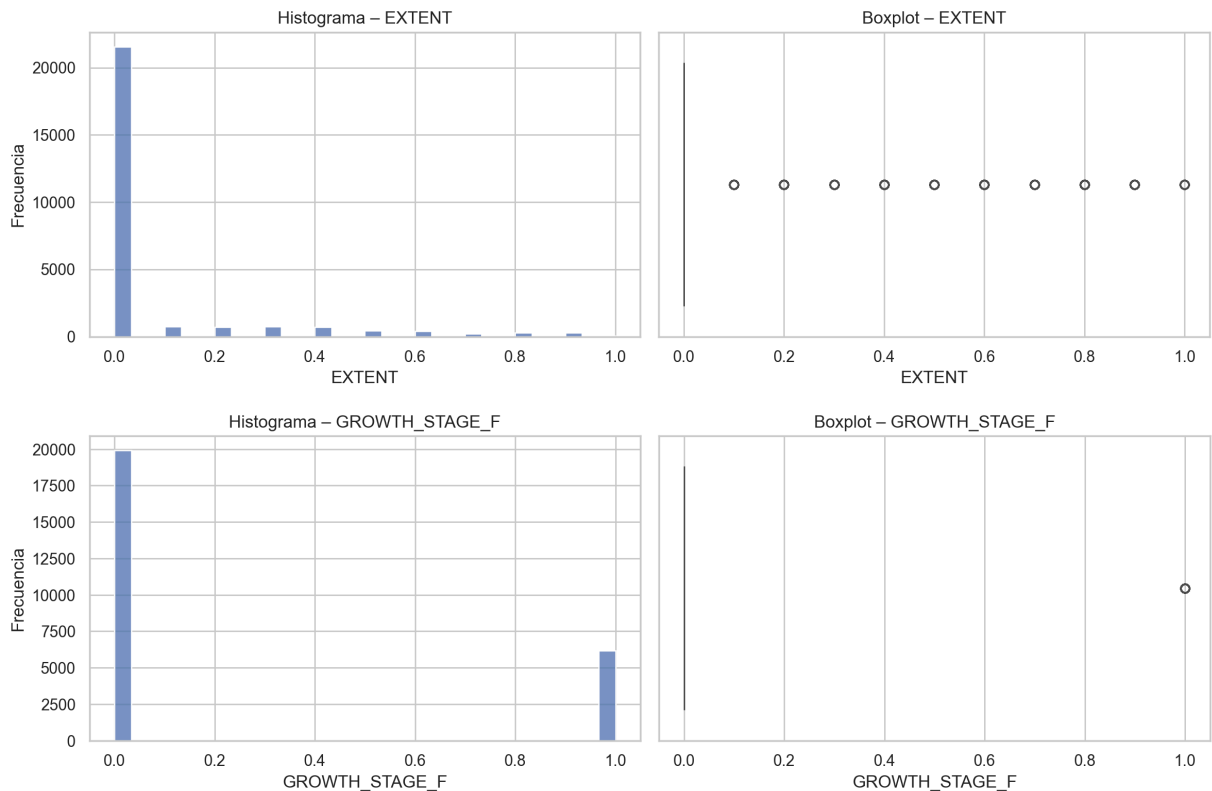
        # Boxplot
        sns.boxplot(x=data, ax=axes[1])
        axes[1].set_title(f"Boxplot - {col}")
        axes[1].set_xlabel(col)

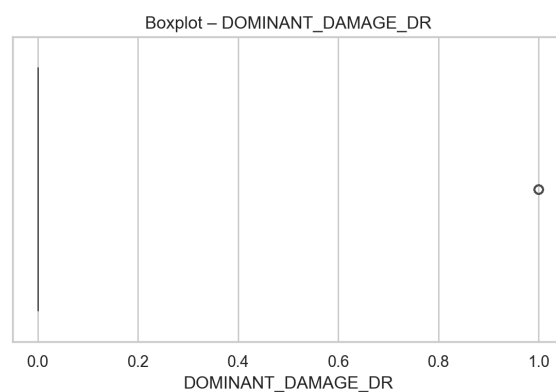
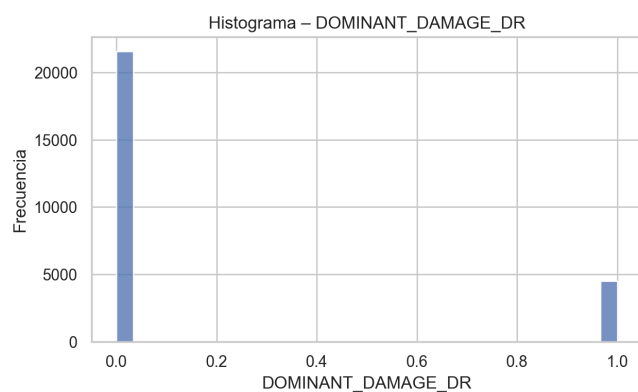
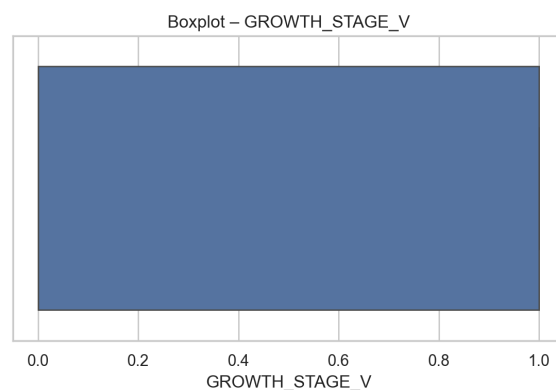
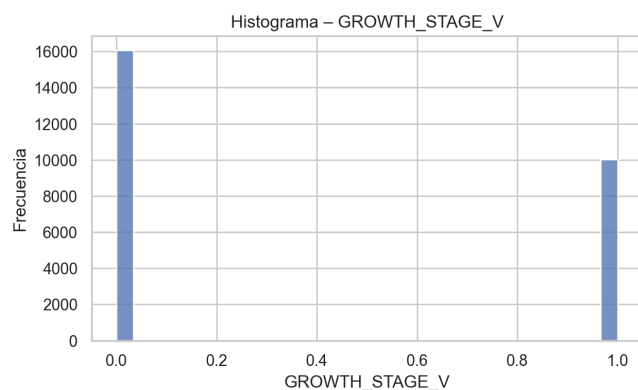
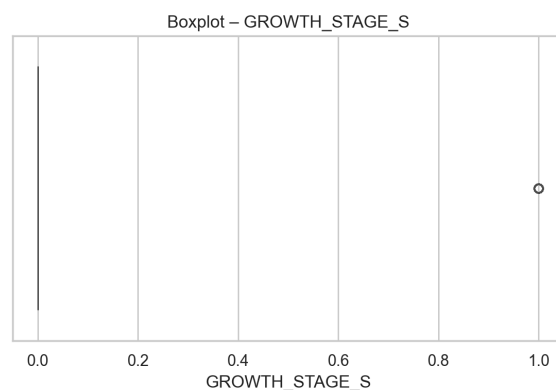
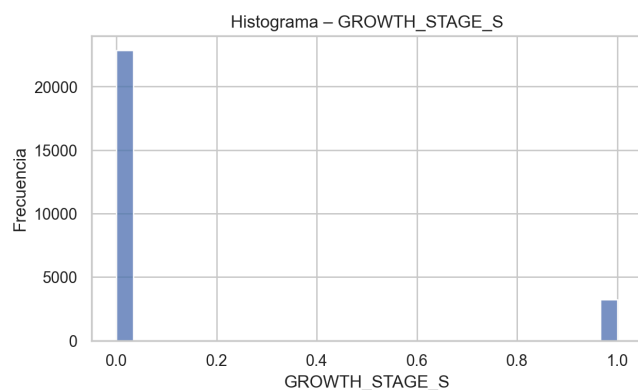
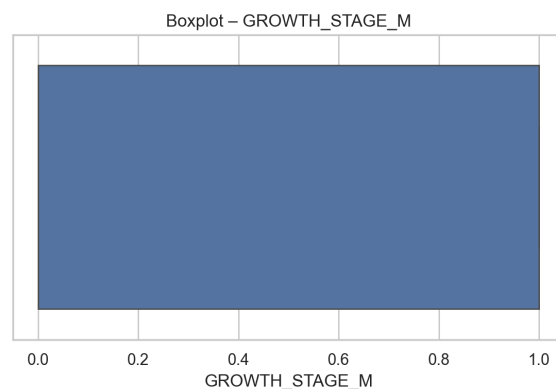
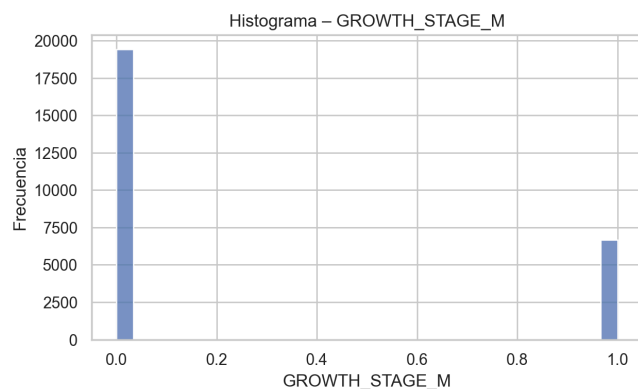
        plt.tight_layout()
        plt.show()

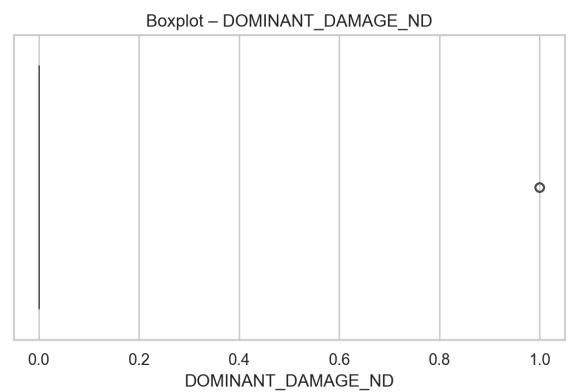
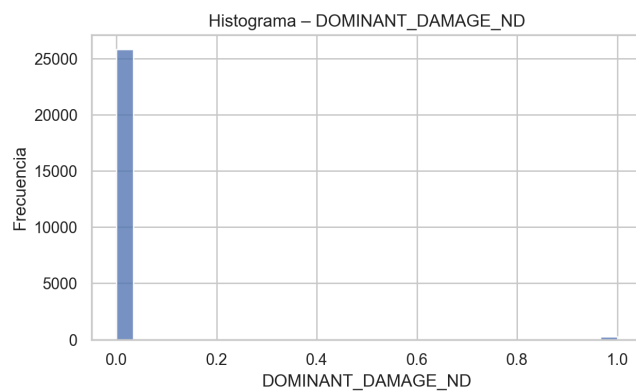
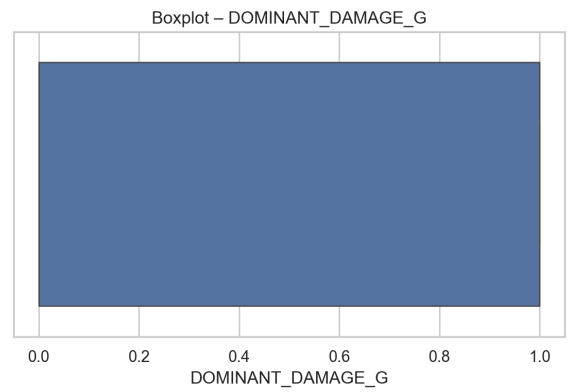
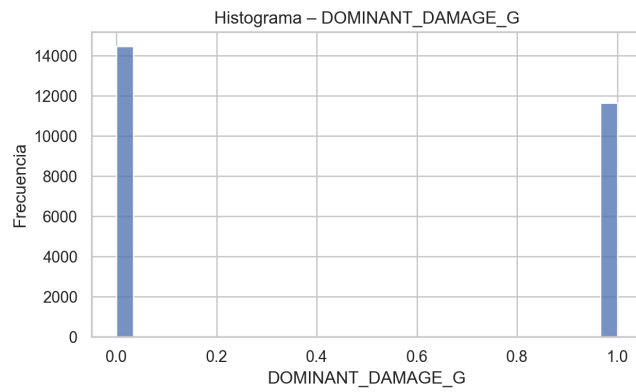
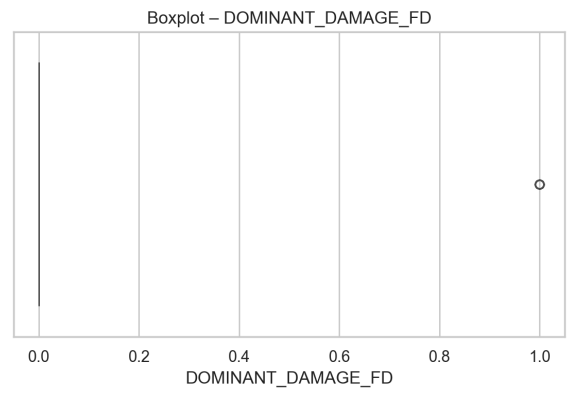
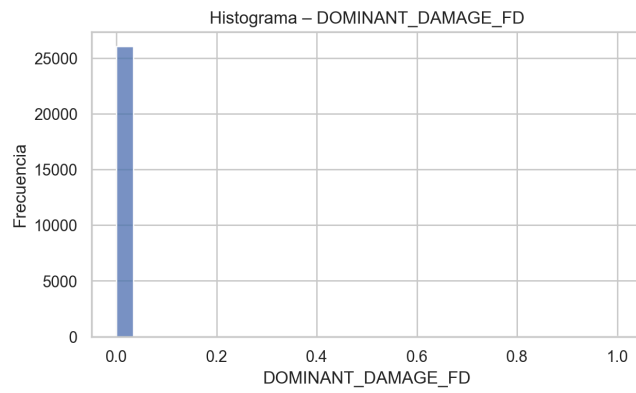
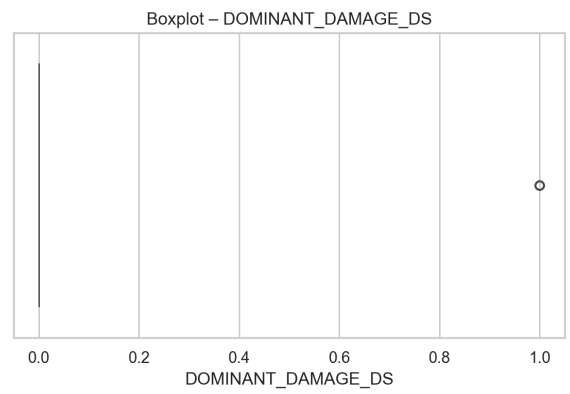
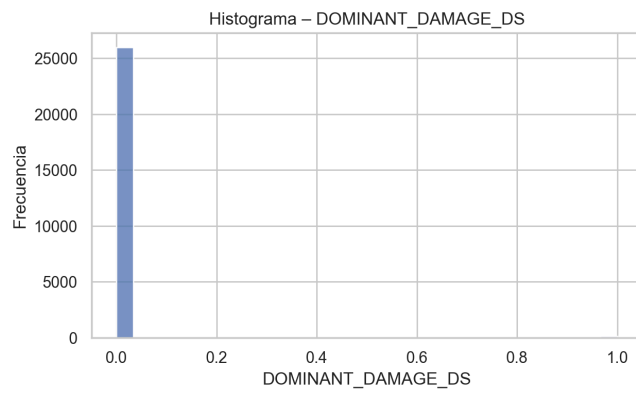
        # Save the figure
        fig.savefig(OUTPUT_DIR / f"DIST_{col}.png", dpi=150, bbox_inches='tight')
        plt.close()

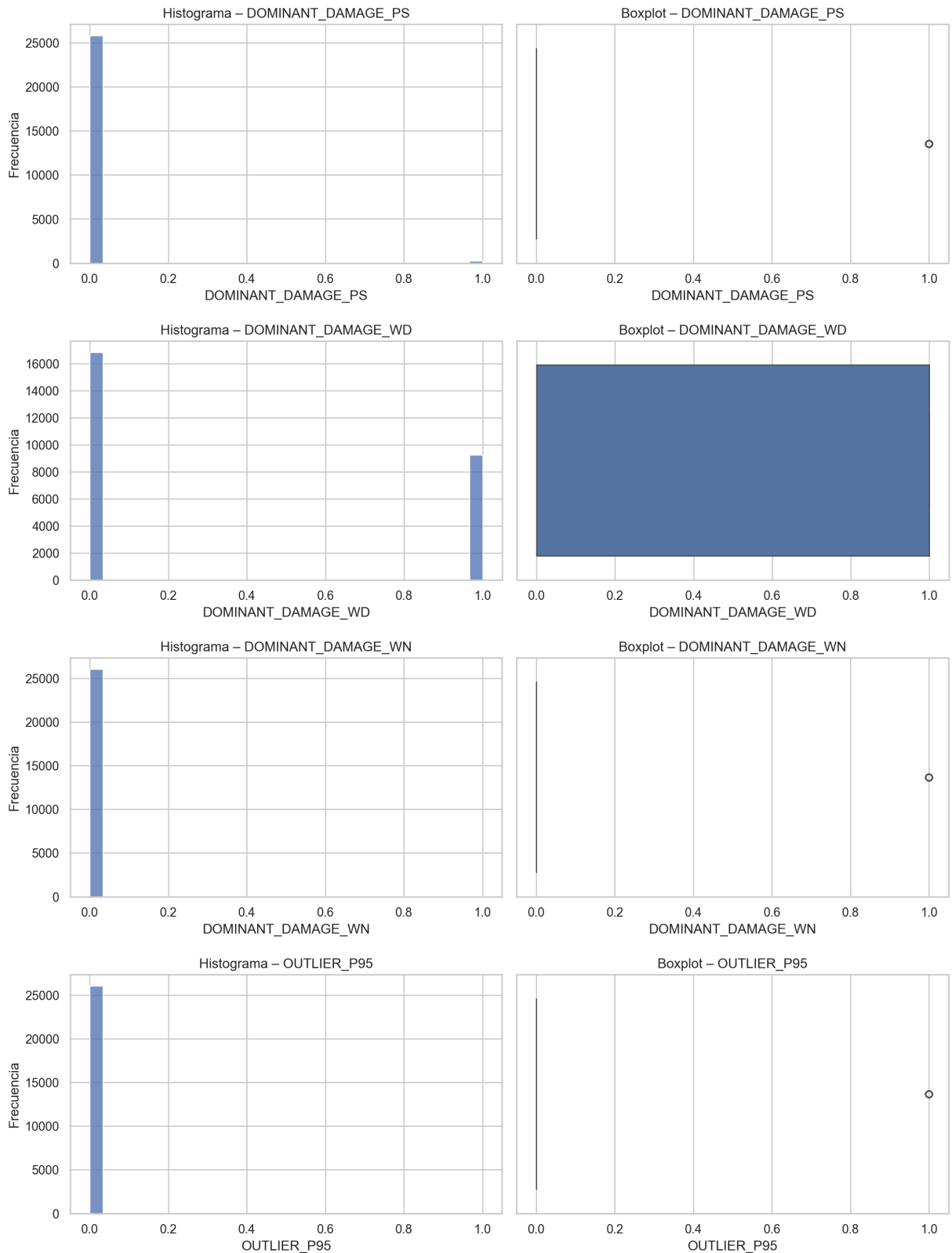
```

Available numeric columns: ['EXTENT', 'GROWTH_STAGE_F', 'GROWTH_STAGE_M', 'GROWTH_STAGE_S', 'GROWTH_STAGE_V', 'DOMINANT_DAMAGE_DR', 'DOMINANT_DAMAGE_DS', 'DOMINANT_DAMAGE_FD', 'DOMINANT_DAMAGE_G', 'DOMINANT_DAMAGE_ND', 'DOMINANT_DAMAGE_PS', 'DOMINANT_DAMAGE_WD', 'DOMINANT_DAMAGE_WN', 'OUTLIER_P95']









Análisis de distribuciones numéricas

Variable EXTENT : Muestra una distribución extremadamente sesgada hacia cero, con ~75% de observaciones sin daño visible (extent=0). Los valores positivos siguen una distribución exponencial decreciente, indicando que daños severos son raros. El boxplot confirma numerosos valores atípicos en el rango superior.

Variables dummy (GROWTH_STAGE_, DOMINANT_DAMAGE_): Como era esperado, estas variables binarias muestran distribuciones bimodales con concentración en 0 y 1, reflejando la codificación one-hot aplicada.

Variable OUTLIER_P95 : Distribución binaria con predominio de valores False (~98%), confirmando que solo un pequeño porcentaje de observaciones fueron marcadas como atípicas según el criterio del percentil 95.

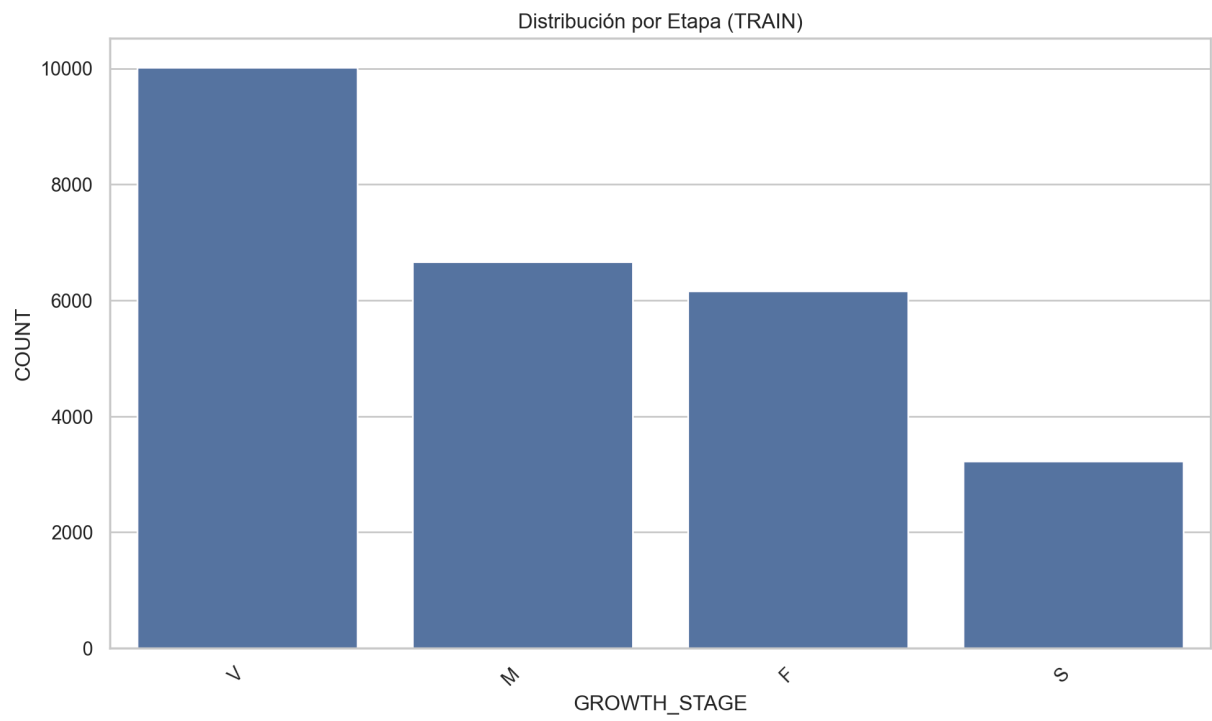
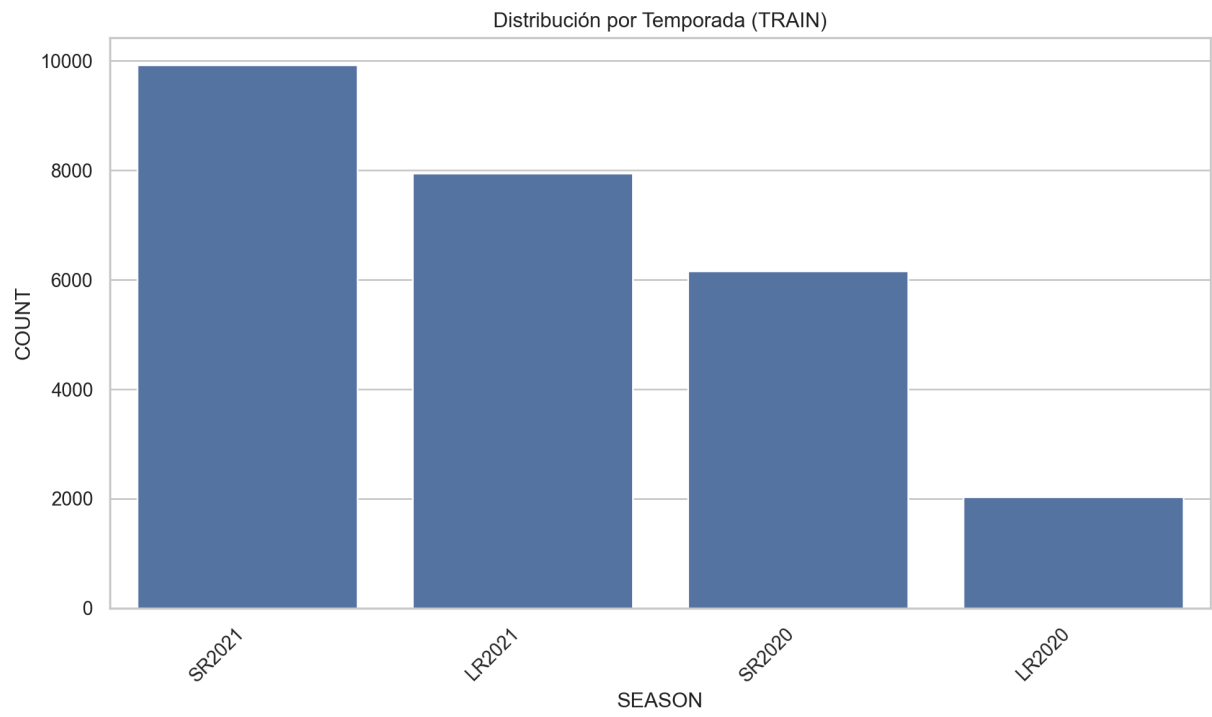
Implicaciones para modelado: La alta concentración en cero de **EXTENT** sugiere la necesidad de técnicas especializadas para datos con exceso de ceros (zero-inflated models) o transformaciones logarítmicas. La presencia de valores atípicos identificados requiere evaluación cuidadosa de su impacto en el rendimiento del modelo.

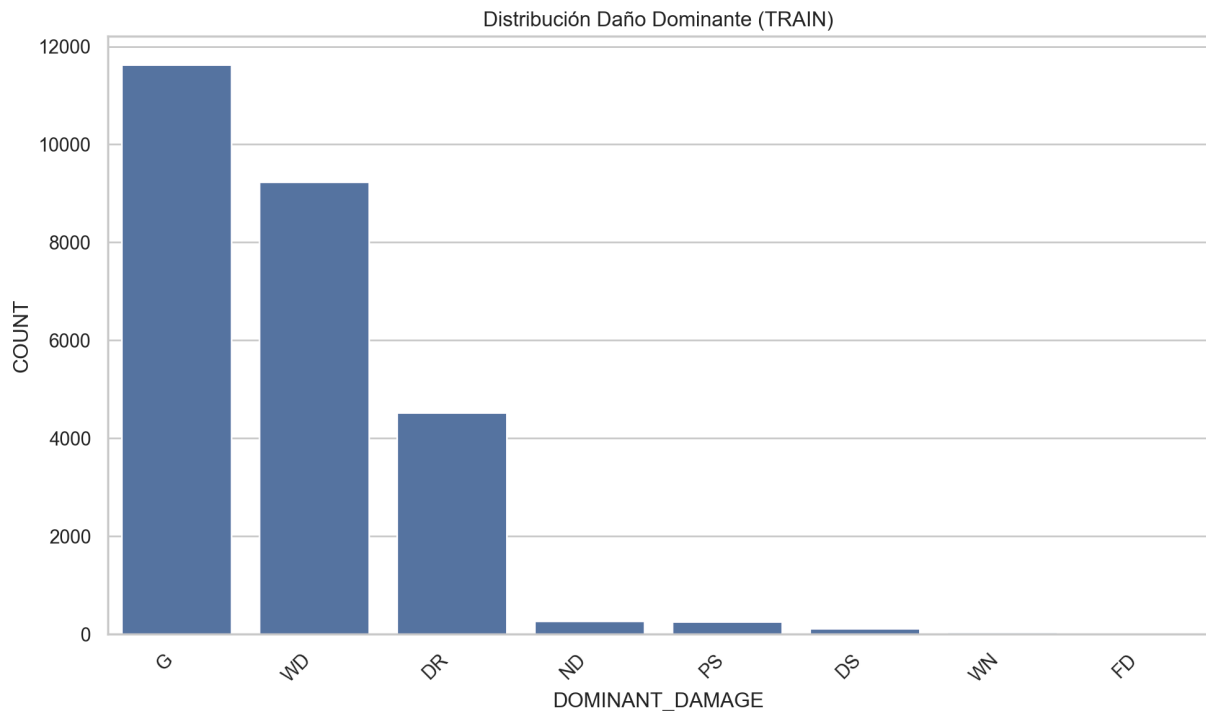
5.5 Barras categóricas

```
In [ ]: # Barras categóricas
print("\n### DISTRIBUCIÓN DE CATEGÓRICAS")
def PLOT_BAR_COUNTS(DF, COL, TITLE, FNAME):
    if COL not in DF.columns:
        print(f"[AVISO] NO EXISTE {COL} PARA GRAFICAR.")
        return
    VC = DF[COL].value_counts(dropna=False)
    plt.figure(figsize=(10,6))
    sns.barplot(x=VC.index.astype(str), y=VC.values)
    plt.title(TITLE)
    plt.xlabel(COL)
    plt.ylabel("COUNT")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    plt.savefig(OUTPUT_DIR / FNAME)
    plt.close()

PLOT_BAR_COUNTS(TRAIN_CLEAN, "SEASON", "Distribución por Temporada (TRAIN)", "BAR_S
PLOT_BAR_COUNTS(TRAIN_CLEAN, "GROWTH_STAGE", "Distribución por Etapa (TRAIN)", "BAR
PLOT_BAR_COUNTS(TRAIN_CLEAN, "DOMINANT_DAMAGE", "Distribución Daño Dominante (TRAIN

### DISTRIBUCIÓN DE CATEGÓRICAS
```





Se observa desbalance en temporadas (SR2021 es la más representada) y en etapas fenológicas (predominio vegetativo). El daño dominante se concentra en G (sin daño) y WD, con muy baja frecuencia en otras categorías.

5.6 Dispersión entre dos etiquetas

```
In [ ]: print("\n### DIAGRAMA DE DISPERSIÓN (ejemplo con 2 primeras variables numéricas)")

numeric_data_cols = [col for col in numeric_cols if col not in ['GROWTH_STAGE_F', '

if len(numeric_data_cols) >= 2:
    x_col = numeric_data_cols[0]
    y_col = numeric_data_cols[1] if len(numeric_data_cols) > 1 else 'GROWTH_STAGE_F

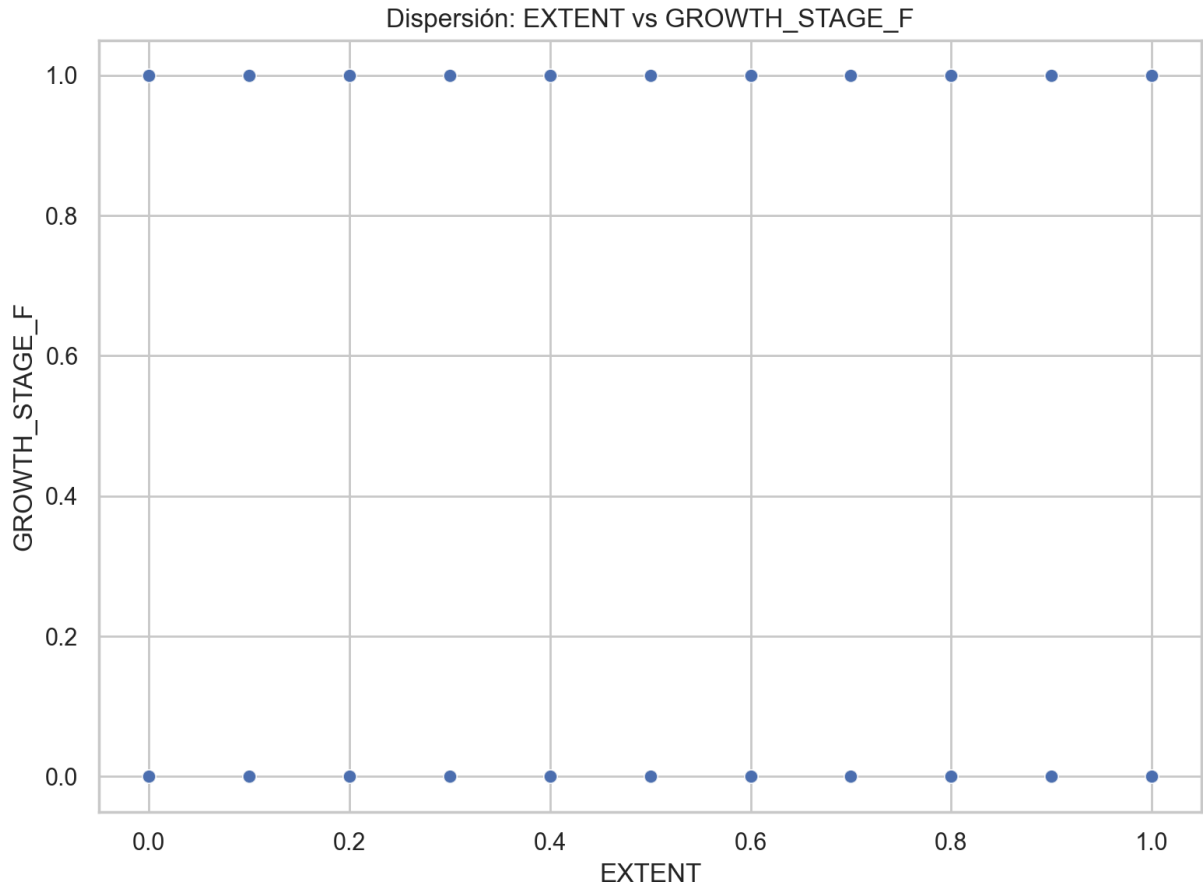
    plt.figure(figsize=(8,6))
    sns.scatterplot(x=TRAIN_CLEAN[x_col], y=TRAIN_CLEAN[y_col], alpha=0.5)
    plt.title(f"Dispersión: {x_col} vs {y_col}")
    plt.xlabel(x_col)
    plt.ylabel(y_col)
    plt.tight_layout()
    plt.savefig(OUTPUT_DIR / "SCATTER_LABELS.png", dpi=150)
    plt.show()
elif len(numeric_data_cols) >= 1:
    x_col = numeric_data_cols[0]
    y_col = 'GROWTH_STAGE_F'

    plt.figure(figsize=(8,6))
    sns.scatterplot(x=TRAIN_CLEAN[x_col], y=TRAIN_CLEAN[y_col], alpha=0.5)
    plt.title(f"Dispersión: {x_col} vs {y_col}")
    plt.xlabel(x_col)
    plt.ylabel(y_col)
    plt.tight_layout()
```

```
plt.savefig(OUTPUT_DIR / "SCATTER_LABELS.png", dpi=150)
plt.show()
else:
    print("[AVISO] No hay suficientes columnas numéricas para graficar dispersión.")

print(f"Columnas numéricas disponibles: {numeric_data_cols}")
```

DIAGRAMA DE DISPERSIÓN (ejemplo con 2 primeras variables numéricas)



Columnas numéricas disponibles: ['EXTENT']

La dispersión muestra codificación one-hot (0/1) en etapas, lo que limita su utilidad gráfica. Aun así, confirma la correcta transformación de categóricas y la fuerte presencia de EXTENT=0.

5.7 Matriz de correlación

```
In [ ]: print("\n### MATRIZ DE CORRELACIÓN")

LABEL_COLS = TRAIN_CLEAN.attrs.get("LABEL_COLS", [])

if not LABEL_COLS:
    potential_labels = ['DR', 'DS', 'FD', 'G', 'ND', 'PS', 'WD', 'WN']
    LABEL_COLS = [col for col in potential_labels if col in TRAIN_CLEAN.columns]

if not LABEL_COLS:
    print("[INFO] No se encontraron columnas de etiquetas originales.")
    print("Usando variables numéricas disponibles para correlación...")
```

```

        numeric_meaningful = ['EXTENT'] + [col for col in TRAIN_CLEAN.columns
                                             if col.startswith(('GROWTH_STAGE_', 'DOMI
                                             and pd.api.types.is_numeric_dtype(TRAIN_C

    LABEL_COLS = numeric_meaningful

if LABEL_COLS:
    print(f"Columnas para correlación: {LABEL_COLS}")

    corr = TRAIN_CLEAN[LABEL_COLS].corr()
    display(corr.round(3))

    plt.figure(figsize=(12, 8))
    sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm",
                center=0, square=True, cbar=True)
    plt.title("Matriz de Correlación")
    plt.xticks(rotation=45, ha='right')
    plt.yticks(rotation=0)
    plt.tight_layout()
    plt.savefig(OUTPUT_DIR / "CORR_MATRIX.png", dpi=150, bbox_inches='tight')
    plt.show()
else:
    print("[AVISO] NO SE ENCONTRARON COLUMNAS VÁLIDAS PARA CORRELACIÓN.")

```

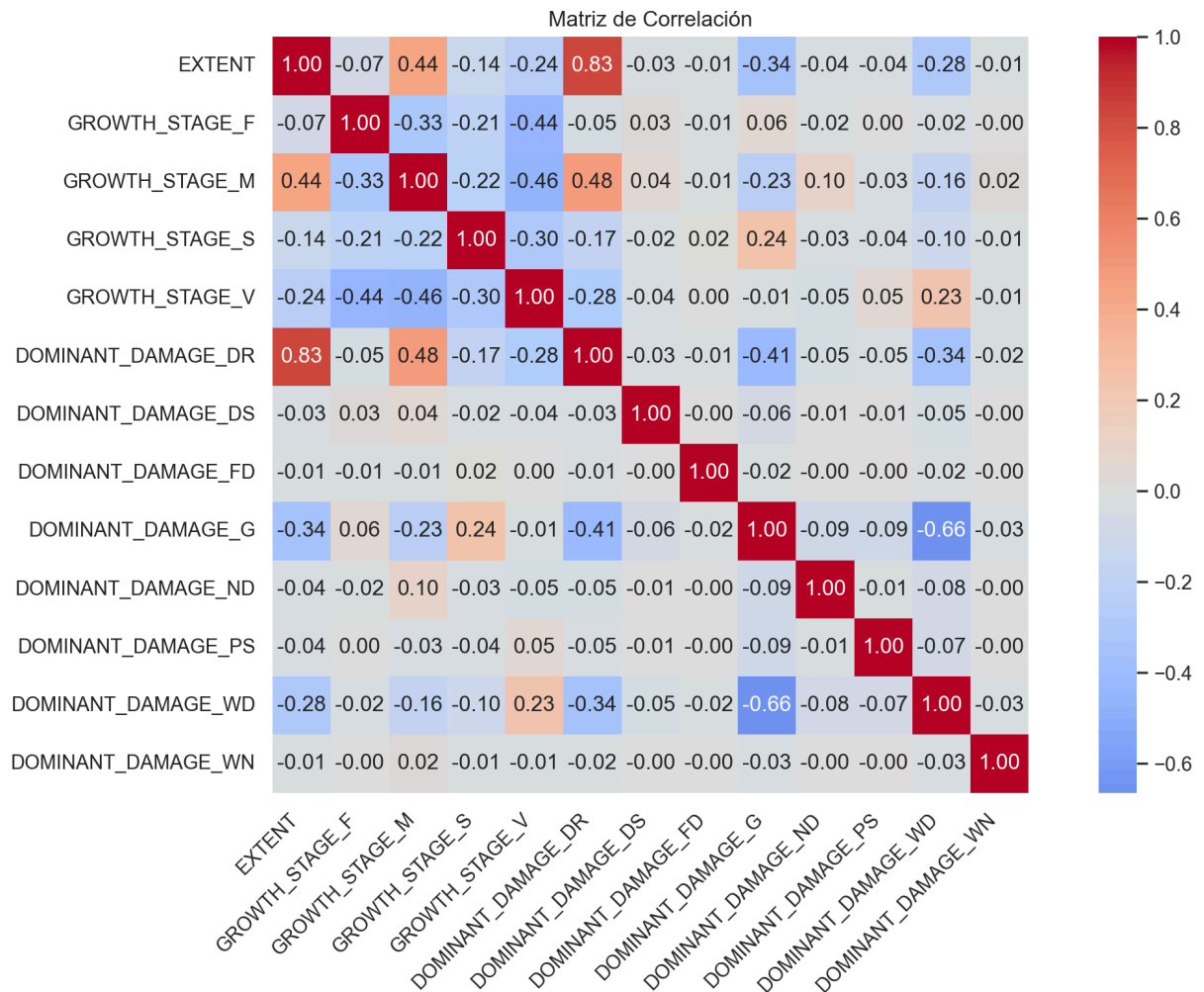
MATRIZ DE CORRELACIÓN

[INFO] No se encontraron columnas de etiquetas originales.

Usando variables numéricas disponibles para correlación...

Columnas para correlación: ['EXTENT', 'GROWTH_STAGE_F', 'GROWTH_STAGE_M', 'GROWTH_STAGE_S', 'GROWTH_STAGE_V', 'DOMINANT_DAMAGE_DR', 'DOMINANT_DAMAGE_DS', 'DOMINANT_DAMAGE_FD', 'DOMINANT_DAMAGE_G', 'DOMINANT_DAMAGE_ND', 'DOMINANT_DAMAGE_PS', 'DOMINANT_DAMAGE_WD', 'DOMINANT_DAMAGE_WN']

	EXTENT	GROWTH_STAGE_F	GROWTH_STAGE_M	GROWTH_STAGE_S
EXTENT	1.000	-0.075	0.440	-0.137
GROWTH_STAGE_F	-0.075	1.000	-0.326	-0.209
GROWTH_STAGE_M	0.440	-0.326	1.000	-0.220
GROWTH_STAGE_S	-0.137	-0.209	-0.220	1.000
GROWTH_STAGE_V	-0.236	-0.440	-0.463	-0.290
DOMINANT_DAMAGE_DR	0.833	-0.051	0.484	-0.163
DOMINANT_DAMAGE_DS	-0.025	0.028	0.038	-0.009
DOMINANT_DAMAGE_FD	-0.009	-0.008	-0.009	0.002
DOMINANT_DAMAGE_G	-0.342	0.063	-0.232	0.242
DOMINANT_DAMAGE_ND	-0.039	-0.020	0.098	-0.033
DOMINANT_DAMAGE_PS	-0.038	0.002	-0.033	-0.033
DOMINANT_DAMAGE_WD	-0.282	-0.024	-0.163	-0.163
DOMINANT_DAMAGE_WN	-0.014	-0.004	0.025	-0.004



EXTENT se correlaciona positivamente con DR ($r \approx 0.83$), lo que sugiere que esa etiqueta se asocia a valores de severidad altos. Las demás correlaciones son bajas, confirmando relativa independencia entre daños.

5.8 Outliers por daño

```
In [96]: print("\n### OUTLIERS POR DAÑO (P95)")

def OUTLIERS_PERCENTILE_BY_DAMAGE(DF: pd.DataFrame, COL: str = "EXTENT", DAMAGE_COL: str = "DAMAGE", Q: float = 0.95, NONZERO_ONLY: bool = True) -> pd.Series:
    flags = pd.Series(False, index=DF.index)
    if COL not in DF.columns or DAMAGE_COL not in DF.columns:
        return flags
    for dmg, g in DF.groupby(DAMAGE_COL, dropna=False):
        x = g[COL]
        if NONZERO_ONLY:
            x = x[x > 0]
        if x.empty:
            continue
        thr = x.quantile(Q)
        flags.loc[g.index] = g[COL] > thr
    return flags
```



```

if "EXTENT" in TRAIN_CLEAN.columns and "DAMAGE" in TRAIN_CLEAN.columns:
    TRAIN_CLEAN["OUTLIER_P95"] = OUTLIERS_PERCENTILE_BY_DAMAGE(TRAIN_CLEAN)
    OUT_SUMMARY = (
        TRAIN_CLEAN
        .groupby("DAMAGE", dropna=False)["OUTLIER_P95"]
        .agg(SUM_OUTLIERS="sum", COUNT="count")
        .assign(RATE=lambda d: d["SUM_OUTLIERS"] / d["COUNT"])
        .sort_values("RATE", ascending=False)
    )
    display(OUT_SUMMARY)
    print("TOTAL OUTLIERS (P95, NONZERO ONLY):", int(TRAIN_CLEAN["OUTLIER_P95"].sum))
else:
    print("[AVISO] NO SE PUDO CALCULAR OUTLIERS (FALTAN COLUMNAS EXTENT/DAMAGE).")

```

OUTLIERS POR DAÑO (P95)

	SUM_OUTLIERS	COUNT	RATE
DAMAGE			
DR	45	4516	0.009965
DS	0	115	0.000000
FD	0	13	0.000000
G	0	11623	0.000000
ND	0	272	0.000000
PS	0	254	0.000000
WD	0	9238	0.000000
WN	0	37	0.000000

TOTAL OUTLIERS (P95, NONZERO ONLY): 45

Solo DR presenta outliers detectables (~1%), mientras que en otras clases no se observan valores extremos. Esto refleja tanto la baja frecuencia de ciertos daños como la alta consistencia de valores discretos.

RESÚMENES PROGRAMÁTICOS Y HALLAZGOS

```

In [22]: # resumen de hallazgos
def RESUMEN_HALLAZGOS(TRAIN: pd.DataFrame):
    LBL = TRAIN.attrs.get("LABEL_COLS", [])
    LBL = [c for c in LBL if c in TRAIN.columns]

    LINEAS = []
    LINEAS.append("## 6. Conclusiones - Hallazgos principales\n")

    # faltantes
    MISS = TRAIN.isna().sum()

```

```

MISS = MISS[MISS>0].sort_values(ascending=False)
if not MISS.empty:
    TOP_MISS = MISS.head(8).to_dict()
    LINEAS.append("**Valores faltantes (top):** " + ", ".join([f"{k}: {v}" for k, v in TOP_MISS.items()]))

# distribuciones
if "SEASON" in TRAIN.columns:
    SEASON_VC = TRAIN["SEASON"].value_counts(dropna=False).to_dict()
    LINEAS.append("**Distribución por temporada:** " + ", ".join([f"{k}: {v}" for k, v in SEASON_VC.items()]))

if "GROWTH_STAGE" in TRAIN.columns:
    STAGE_VC = TRAIN["GROWTH_STAGE"].value_counts(dropna=False).to_dict()
    LINEAS.append("**Distribución por etapa:** " + ", ".join([f"{k}: {v}" for k, v in STAGE_VC.items()]))

# daño dominante
if "DOMINANT_DAMAGE" in TRAIN.columns:
    DOM_VC = TRAIN["DOMINANT_DAMAGE"].value_counts(dropna=False).to_dict()
    LINEAS.append("**Daño dominante (conteos):** " + ", ".join([f"{k}: {v}" for k, v in DOM_VC.items()]))

# estadísticas de etiquetas
if LBL:
    MEANS = TRAIN[LBL].mean().sort_values(ascending=False).to_dict()
    LINEAS.append("**Promedios de extensión por etiqueta (mayor→menor):** " + ", ".join([f"{k}: {v}" for k, v in MEANS.items()]))

TEXTO = "\n".join(LINEAS)
print(TEXTO)

RESUMEN_HALLAZGOS(TRAIN_CLEAN)

```

6. Conclusiones – Hallazgos principales

```

**Distribución por temporada:** SR2021: 9927, LR2021: 7945, SR2020: 6163, LR2020: 2033
**Promedios de extensión por etiqueta (mayor→menor):** EXTENT: 0.07%

```

6. Hallazgos y conclusiones

Calidad y estructura.

Los metadatos (`SEASON` , `GROWTH_STAGE`) están bien estandarizados y muestran distribuciones claras, con predominio de **SR2021** y de la etapa **Vegetative**.

Las etiquetas de daño se encuentran en formato *wide* y discretizadas en múltiplos de 10, con fuerte concentración en 0.

Solo se detectaron outliers en `DR` ($\approx 1\%$), lo cual indica buena consistencia en el resto de clases.

Patrones principales.

El **daño dominante general** es `G` (sin daño), seguido de `WD` , mientras que categorías como `ND` , `PS` o `FD` son casi marginales.

Hay desbalance notorio tanto por temporada como por etapa de crecimiento, lo que implica un riesgo de sesgo en modelos.

En la matriz de correlación, `EXTENT` y `DR` presentan correlación alta (≈ 0.83), lo que refleja

que la severidad medida se relaciona estrechamente con ese tipo de daño; el resto de etiquetas tiene correlaciones débiles o nulas.

Implicaciones.

Los hallazgos sugieren:

1. Incorporar **SEASON y GROWTH_STAGE** como predictores auxiliares en los modelos.
2. Manejar el problema como **multiclase altamente desbalanceado**, aplicando aumentos o técnicas de rebalanceo para daños poco frecuentes.
3. Considerar pérdidas que manejen **valores discretizados y sesgo a 0** (p. ej. focal loss o regresión ordinal).

Siguientes pasos.

Entrenar modelos base con validación estratificada por temporada y etapa para evitar *leakage*, experimentar con arquitecturas multiclase y ordinales, e incorporar métricas como **macro-F1** y **QWK** para capturar el rendimiento en clases minoritarias.

Finalmente, añadir técnicas de explicabilidad (Grad-CAM/SHAP) para evaluar la coherencia de las predicciones con criterios agronómicos.

Referencias

- Science, S. (2025, September 10). Bacterias y Enfermedades de las Plantas: Entendiendo la Amenaza. Simple Science. <https://scisimple.com/es/articles/2025-09-10-bacterias-y-enfermedades-de-las-plantas-entendiendo-la-amenaza--akg4jeg>
- Díaz, I. (2025, August 30). Efectos de la Contaminación en los Cultivos: Impacto y Soluciones. Ecología Digital. <https://ecologiadigital.bio/los-efectos-de-la-contaminacion-en-los-cultivos/>
- Gómez, P. (2025, September 14). Relación entre hongos y enfermedades en las plantas. Todo Micología. <https://todomicologia.com/fisiologia/relacion-entre-hongos-y-enfermedades-en-las-plantas/>
- Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016a). Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science*, 7. <https://doi.org/10.3389/fpls.2016.01419>
- Riley, M. B., Williamson, M. R., & Maloy, O. (2002). Plant disease diagnosis. *The Plant Health Instructor*. <https://doi.org/10.1094/phi-i-2002-1021-01>
- Shoaib, M., Shah, B., Shaker El-Sappagh, Ali, A., Ullah, A., Fayadh Alenezi, Tsanko Gechev, Hussain, T., & Ali, F. (2023). An advanced deep learning models-based plant disease detection: A review of recent research. *Frontiers in Plant Science*, 14. <https://doi.org/10.3389/fpls.2023.1158933>

- Zhou, G., Zhang, W., Chen, A., He, M., & Ma, X. (2019). Rapid Detection of Rice Disease Based on FCM-KM and Faster R-CNN Fusion. *IEEE Access*, 7, 143190–143206.
<https://doi.org/10.1109/access.2019.2943454>