Proyecto final Ineligencia artificial

2024-01-20

Alain Ruales, Daniel Bustos

Clasificación y Regresión de Humedad en Arroz mediante Aprendizaje Automático

Resumen: Este proyecto se centra en el desarrollo de un modelo de aprendizaje automático para clasificar y predecir los niveles de humedad de granos de arroz basándose en imágenes. El objetivo principal es crear un sistema robusto capaz de categorizar con precisión los granos de arroz en seis niveles de humedad (que van del 10 al 16) y predecir el contenido de humedad entero más cercano.

Introducción Plausible de la Aplicación: En las industrias agrícolas y de procesamiento de alimentos, mantener niveles de humedad óptimos en granos de arroz es crucial para el control de calidad y almacenamiento. Los métodos tradicionales para la evaluación de la humedad pueden ser lentos y propensos a errores. Este proyecto tiene como objetivo aprovechar técnicas de aprendizaje automático para automatizar la clasificación y regresión de los niveles de humedad en granos de arroz basándose en datos de imágenes, proporcionando una solución más rápida y precisa para el control de calidad en el proceso de producción de arroz.

Creación y Preprocesamiento del Conjunto de Datos: El conjunto de datos utilizado en este proyecto comprende imágenes de granos de arroz capturadas bajo condiciones de humedad variables. La creación del conjunto de datos implicó controlar y registrar sistemáticamente los niveles de humedad durante el proceso de adquisición de imágenes. Las tareas de preprocesamiento incluyeron el redimensionamiento de imágenes, la normalización y la extracción de patrones.

Exploración del Conjunto de Datos: El análisis exploratorio de datos implicó la visualización del conjunto de datos mediante técnicas como PCA (Análisis de Componentes Principales) y t-SNE (Incrustación de Vecinos Estocásticos Distribuidos t). Estos métodos ayudaron a comprender los patrones y relaciones subyacentes en los datos, ofreciendo ideas sobre posibles características para el modelado.

Construcción del Modelo: Se formularon dos tipos de problemas de aprendizaje automático:

1. **Problema de Clasificación:**
    - Las clases representan niveles de humedad (de 10 a 16).
    - Los modelos construidos incluyen clasificadores como Random Forest y XGBoost.
2. **Problema de Regresión:**

- Los valores objetivo se redondean al entero más cercano.
- Los modelos incluyen regresores como Random Forest y XGBoost.

Pasos para la Construcción del Modelo:

1. **Selección del Algoritmo de Aprendizaje Automático:** Se eligieron Random Forest y XGBoost por su versatilidad y rendimiento en tareas de clasificación y regresión.
2. **Preprocesamiento:** Los datos de entrada fueron preprocesados según la naturaleza del problema, incluyendo el escalado de características y el manejo de valores faltantes.
3. **Entrenamiento del Modelo:** Se entrenaron modelos en los datos preprocesados para aprender las relaciones entre las características de las imágenes y los niveles de humedad.
4. **Optimización de Hiperparámetros:** Se empleó la búsqueda en cuadrícula u otras técnicas de optimización para ajustar finamente los hiperparámetros del modelo, mejorando el rendimiento predictivo.
5. **Evaluación del Modelo:** Los modelos entrenados se evaluaron en un conjunto de prueba separado para evaluar su precisión y capacidades de generalización.
6. **Validación Cruzada:** Para garantizar la robustez, se aplicaron técnicas de validación cruzada para evaluar el rendimiento del modelo en diferentes subconjuntos del conjunto de datos.

Este proyecto ofrece un enfoque integral para la clasificación y regresión automatizada de la humedad en arroz, con aplicaciones potenciales en el control de calidad y monitoreo dentro de las industrias agrícolas y de procesamiento de alimentos.

Contexto de la Aplicación:

En el ámbito de la agricultura y procesamiento de alimentos moderno, la búsqueda de eficiencia, precisión y aseguramiento de la calidad ha llevado a la integración de tecnologías avanzadas. Nuestro proyecto se propone abordar un desafío crucial en la industria del arroz: la gestión precisa del nivel de humedad. Imagina un escenario donde cada grano de arroz se somete a un escrutinio automatizado, asegurando que cumpla con los estándares rigurosos requeridos para una calidad y almacenamiento óptimos.

Tradicionalmente, evaluar el contenido de humedad de los granos de arroz ha sido un proceso manual y propenso a errores. Con la llegada del aprendizaje automático, surge una solución innovadora. Nuestro sistema automatizado utiliza algoritmos sofisticados para clasificar los granos de arroz en niveles distintos de humedad, proporcionando un nivel de precisión y eficiencia sin precedentes. Además, el sistema predice el contenido de humedad entero más cercano, simplificando los procesos de control de calidad.

Las implicaciones de esta tecnología se extienden más allá de los límites de un laboratorio. Imagina una instalación de procesamiento de arroz donde este sistema inteligente se integra sin problemas en la línea de producción, clasificando e prediciendo instantáneamente los niveles de humedad de miles de granos por minuto. Esto no solo mejora la calidad del producto final, sino que también optimiza las condiciones de almacenamiento, reduciendo el riesgo de deterioro y garantizando un producto consistente y superior para los consumidores.

Además, nuestro sistema se alinea con la tendencia más amplia de prácticas agrícolas sostenibles y eficientes en recursos. Al reducir el trabajo manual, minimizar los desperdicios y optimizar el uso de las instalaciones de almacenamiento, contribuye a un enfoque más ecológico y económicamente viable para la producción de arroz.

En esencia, nuestra aplicación plausible aprovecha el poder del aprendizaje automático para revolucionar el enfoque de la industria del arroz hacia la gestión de la humedad, inaugurando una era de agricultura de precisión y asegurando que cada grano de arroz cumpla con los más altos estándares de calidad y satisfacción del consumidor.

librerias requeridas

In [ ]:
```
from PIL import Image
import os
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
import glob
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor
from sklearn.metrics import accuracy_score, classification_report,make_scorer,me
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_sc
from sklearn.neural_network import MLPClassifier,MLPRegressor
from sklearn.pipeline import Pipeline
from seaborn import violinplot
```

variables principales

In [ ]:
```
lado_recorte=660

bins=44
# Ruta de la carpeta principal que contiene las subcarpetas con imágenes
ruta_principal = 'C:/Universidad/9 Semestre/ia/finalProyect/arroz_2023'
# Carpeta de destino para las imágenes en escala de grises
carpeta_destino = 'C:/Universidad/9 Semestre/ia/finalProyect/arroz_2023_gris'
```

Funciones del proyecto

In [ ]:
```
def recortar_cuadrado_central(imagen, lado_recorte):
    # Obtiene las dimensiones de la imagen
    ancho, alto = imagen.size
```

```
    # Calcula las coordenadas para recortar un cuadrado del centro con el tamaño
    izquierda = (ancho - lado_recorte) // 2
    arriba = (alto - lado_recorte) // 2
    derecha = izquierda + lado_recorte
    abajo = arriba + lado_recorte

    # Recorta y devuelve la imagen cuadrada
    return imagen.crop((izquierda, arriba, derecha, abajo))


def convertir_a_escala_de_grises_y_recortar(ruta_carpeta, carpeta_destino, lado_
    # Obtén la lista de archivos en la carpeta
    archivos = os.listdir(ruta_carpeta)

    for archivo in archivos:
        # Ruta completa del archivo
        ruta_completa_origen = os.path.join(ruta_carpeta, archivo)

        # Verifica si es un directorio (subcarpeta)
        if os.path.isdir(ruta_completa_origen):
            # Si es una subcarpeta, crea la carpeta correspondiente en el destin
            nueva_carpeta_destino = os.path.join(carpeta_destino, archivo)
            os.makedirs(nueva_carpeta_destino, exist_ok=True)

            # Llama a la función recursivamente para el directorio
            convertir_a_escala_de_grises_y_recortar(ruta_completa_origen, nueva_
        else:
            # Si es un archivo de imagen, realiza la conversión y recorte a esca
            if archivo.lower().endswith(('.png', '.jpg', '.jpeg', '.gif')):
                # Crea la ruta de destino para la imagen
                ruta_completa_destino = os.path.join(carpeta_destino, archivo)

                # Carga la imagen
                imagen = Image.open(ruta_completa_origen)

                # Convierte a escala de grises
                imagen_gris = imagen.convert('L')  # 'L' representa el modo de e

                # Recorta un cuadrado del centro con el tamaño especificado
                imagen_recortada = recortar_cuadrado_central(imagen_gris, lado_r

                # Guarda la imagen recortada en la carpeta de destino
                imagen_recortada.save(ruta_completa_destino)

def mostrar_grafico_barras(ruta_carpeta):
    # Inicializa un diccionario para almacenar la cantidad de imágenes por carpe
    cantidad_por_carpeta = {}

    # Recorre la carpeta y cuenta la cantidad de imágenes por carpeta
    for root, dirs, files in os.walk(ruta_carpeta):
        # Excluye la carpeta raíz de la cuenta
        if root == ruta_carpeta:
            continue

        cantidad_imagenes = sum(1 for file in files if file.lower().endswith(('.
        carpeta_nombre = os.path.basename(root)  # Obtiene el nombre de la carpe
        cantidad_por_carpeta[carpeta_nombre] = cantidad_imagenes

    # Crea un gráfico de barras
```

```
    carpetas = list(cantidad_por_carpeta.keys())
    cantidades = list(cantidad_por_carpeta.values())

    plt.bar(range(len(carpetas)), cantidades, align='center')
    plt.xticks(range(len(carpetas)), carpetas, rotation=45, ha='right')
    plt.xlabel('Carpetas')
    plt.ylabel('Cantidad de Imágenes')
    plt.title('Cantidad de Imágenes por Carpeta')
    plt.tight_layout()

    # Muestra el gráfico
    plt.show()


def aplanar_imagenes_y_calcular_histograma(ruta_carpeta, bins=bins):
    histogramas_por_carpeta = {}

    for root, dirs, files in os.walk(ruta_carpeta):
        if root == ruta_carpeta:
            continue

        aplanados = []

        for archivo in files:
            if archivo.lower().endswith(('.png', '.jpg', '.jpeg', '.gif')):
                ruta_completa = os.path.join(root, archivo)
                imagen = Image.open(ruta_completa)
                array_aplanado = np.array(imagen).flatten()
                aplanados.append(array_aplanado)

        histograma = np.histogram(aplanados, bins=bins, range=[0, 256])[0]
        carpeta_nombre = os.path.basename(root)
        histogramas_por_carpeta[carpeta_nombre] = histograma

    return histogramas_por_carpeta

# Function to convert an image to grayscale and return the image and its histogr
def convert_to_grayscale(image_path):
    image = Image.open(image_path)
    grayscale_image = image.convert('L')
    histogram = np.array(grayscale_image.histogram())
    return grayscale_image, histogram

# Function to crop the image from the center and return the cropped image and it
def crop_center(image):
    width, height = image.size
    new_width, new_height = 660, 660  # usar recrote de 660x660
    left = (width - new_width) // 2
    top = (height - new_height) // 2
    right = (width + new_width) // 2
    bottom = (height + new_height) // 2
    cropped_image = image.crop((left, top, right, bottom))
    cropped_histogram = np.array(cropped_image.histogram())
    return cropped_image, cropped_histogram
```

Transformacion de dataset

En esta etapa del proyecto las imagenes sufren dos trasnformacionews importantes, primero se recorta la imagen a la escala 660 * 600 y despues son convertidas en escala de grises para facilitar su manejo y análisi

In [ ]:
```
# Llama a la función para convertir a escala de grises
convertir_a_escala_de_grises_y_recortar(ruta_principal, carpeta_destino,lado_rec
```

Descripcion del dataset

Descripción del Conjunto de Datos:

El conjunto de datos utilizado en este proyecto se compone de imágenes representativas de granos de arroz, cada una etiquetada con su respectivo nivel de humedad. Este conjunto de datos es fundamental para entrenar y evaluar los modelos de aprendizaje automático diseñados para clasificar y predecir los niveles de humedad de los granos de arroz.

1. **Imágenes de Granos de Arroz:**
    - Las imágenes incluyen fotografías de granos de arroz capturadas en condiciones controladas para abarcar un rango diverso de humedad.
    - Se han empleado técnicas de captura de imágenes que garantizan la representatividad de los granos en diversas situaciones y contextos.
2. **Etiquetas de Humedad:**
    - Cada imagen está asociada con una etiqueta que indica el nivel de humedad correspondiente al grano de arroz.
    - Las etiquetas de humedad están categorizadas en seis clases distintas, numeradas del 10 al 16, reflejando los diferentes estados de humedad que pueden experimentar los granos.

Proceso de Creación del Conjunto de Datos:

1. **Control de Condiciones Ambientales:**
    - Durante la adquisición de imágenes, se implementó un control riguroso de las condiciones ambientales para asegurar la variabilidad en los niveles de humedad de los granos de arroz.
    - Se registraron los niveles de humedad específicos asociados a cada imagen, estableciendo la verdad fundamental para el entrenamiento supervisado.
2. **Diversidad de Muestras:**
    - Se procuró una diversidad representativa en el conjunto de datos, incluyendo granos de arroz de diferentes variedades y procedencias.
    - La diversidad contribuye a la robustez del modelo, permitiéndole generalizar a condiciones no vistas durante el entrenamiento.

Exploración Visual del Conjunto de Datos:

1. **Visualización de Imágenes:**

- Se realizaron inspecciones visuales de las imágenes para entender la variabilidad en la apariencia de los granos de arroz en relación con los niveles de humedad.
- Las visualizaciones ayudaron a identificar posibles desafíos y patrones que podrían influir en la construcción del modelo.

2. **Análisis Estadístico Preliminar:**

   - Se llevaron a cabo análisis estadísticos básicos para comprender la distribución de las etiquetas de humedad en el conjunto de datos.
   - Esto proporcionó información valiosa sobre la representatividad de cada clase y posibles desequilibrios.

Preprocesamiento del Conjunto de Datos:

1. **Redimensionamiento de Imágenes:**
   - Las imágenes se redimensionaron a un formato estándar para asegurar la consistencia en la entrada del modelo.
   - El redimensionamiento también contribuyó a la eficiencia computacional durante el entrenamiento y evaluación del modelo.

La fase de exploración y visualización del conjunto de datos es esencial para comprender la estructura y las características de los datos antes de construir modelos de aprendizaje automático. Aquí se detalla qué implica la tarea de explorar y visualizar un conjunto de datos:

Explore (Visualize) your dataset:

1. **Visualización de Distribuciones:**

   - **Histogramas:** Representar la distribución de variables numéricas mediante histogramas para identificar patrones, asimetrías y posibles outliers.
   - **Diagramas de Caja (Box Plots):** Visualizar la dispersión y los cuartiles de las variables, facilitando la detección de valores atípicos.

2. **Análisis de Correlación:**

   - **Matriz de Correlación:** Identificar relaciones lineales entre variables numéricas a través de una matriz de correlación.
   - **Mapas de Calor (Heatmaps):** Visualizar la intensidad y dirección de las correlaciones, resaltando patrones.

3. **Exploración de Variables Categóricas:**

   - **Gráficos de Barras:** Representar la frecuencia de variables categóricas mediante gráficos de barras para observar distribuciones.
   - **Diagramas de Pastel:** Proporcionar una perspectiva visual de la proporción de diferentes categorías.

4. **Visualización Multidimensional:**

   - **PCA (Análisis de Componentes Principales):** Reducir la dimensionalidad del conjunto de datos para visualizar la distribución general de las observaciones.

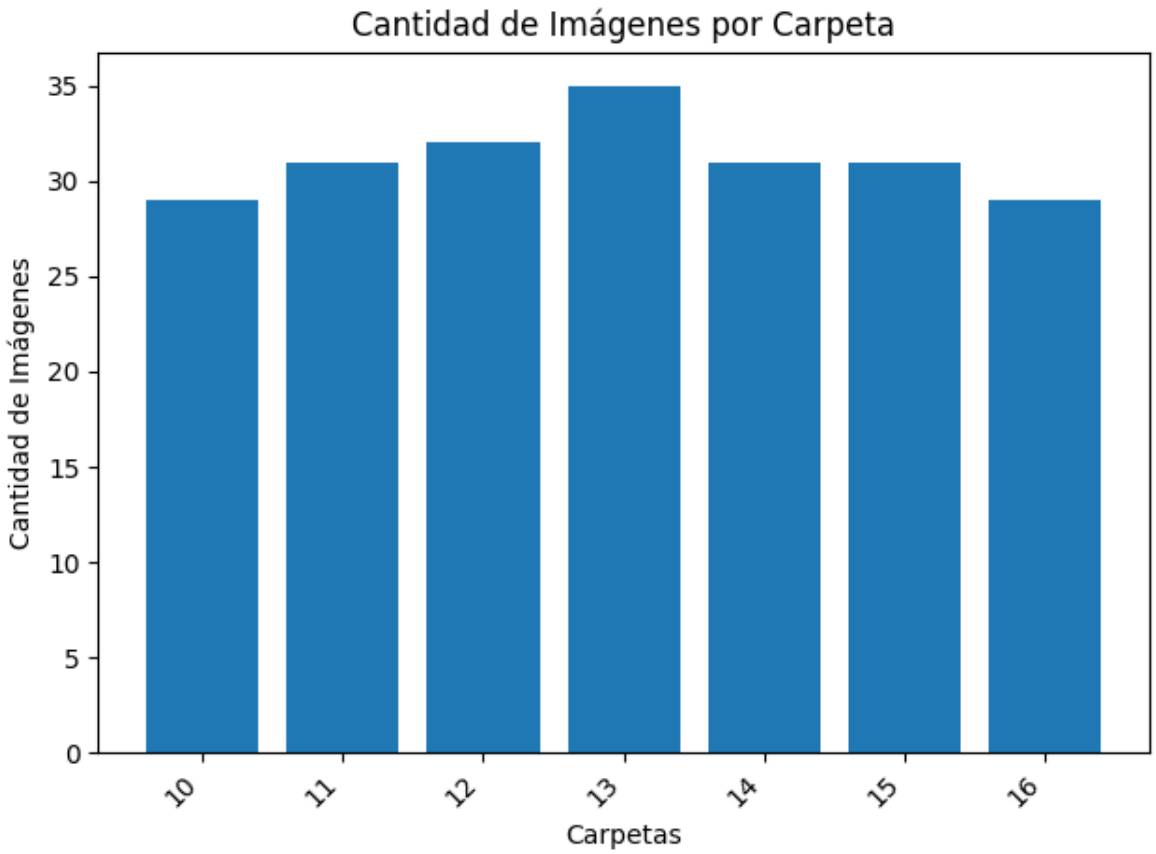
- **t-SNE (t-distributed Stochastic Neighbor Embedding):** Visualizar agrupaciones y patrones no lineales en el conjunto de datos.

5. **Análisis de Imágenes:**

- **Muestra de Imágenes:** Visualizar ejemplos de imágenes en el conjunto de datos para comprender la variabilidad y la calidad de las imágenes.

La visualización del conjunto de datos permite identificar patrones, relaciones y posibles desafíos que puedan surgir durante el modelado. Ayuda a tomar decisiones informadas sobre la selección de características, la identificación de problemas de calidad de datos y la comprensión general del comportamiento de los datos.

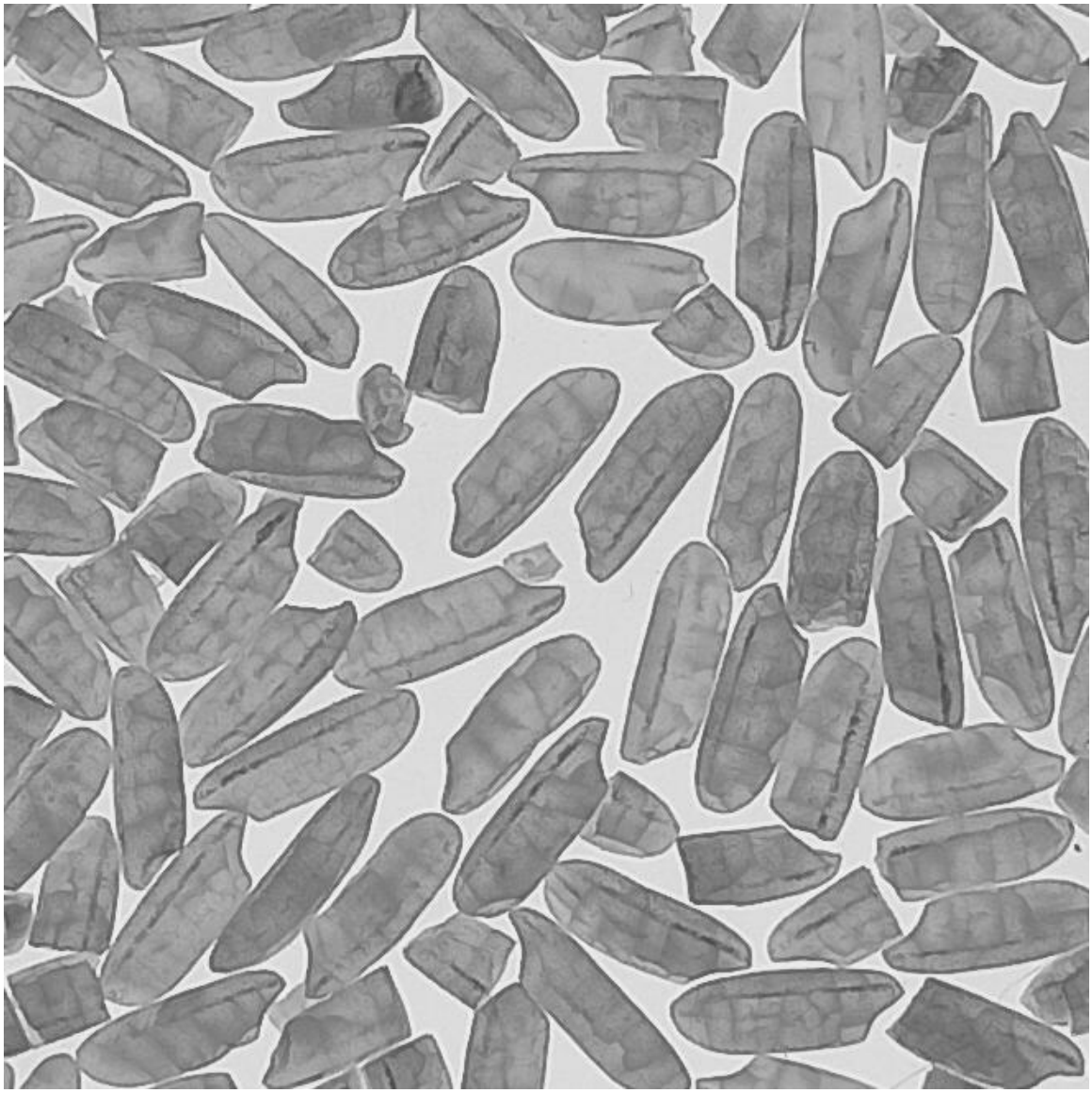
In [ ]:
```
mostrar_grafico_barras(carpeta_destino)
```



In [ ]:
```
# 'image_path'
image_path = './arroz_2023_gris/10/a1.jpg'

cropped_image = Image.open(image_path)

print(cropped_image.size)

cropped_image
```

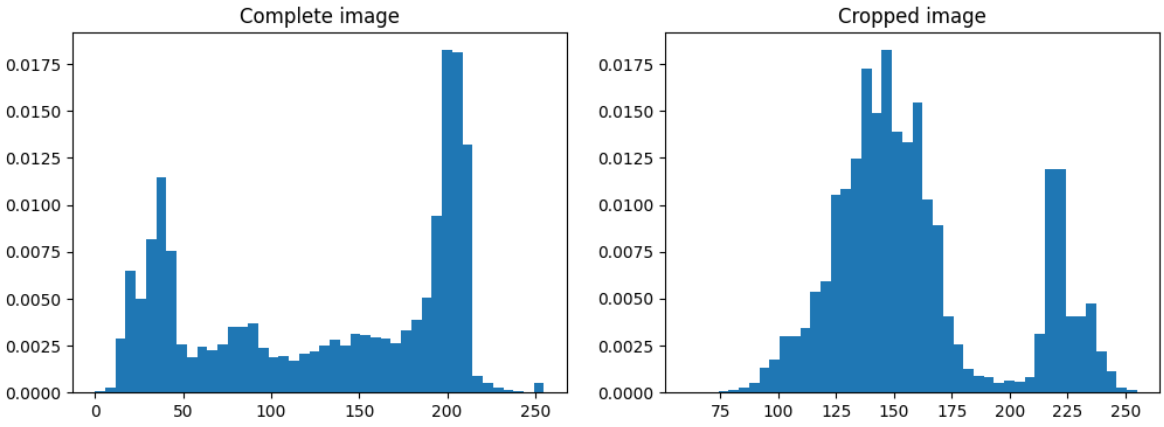
```
(660, 660)
```

Out[ ]:



In [ ]:
```
complete_image_paht='./arroz_2023/10/a1.jpg'
image = Image.open(complete_image_paht)

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.hist(np.array(image).flatten(), bins=bins, density=True);
plt.title("Complete image")

plt.subplot(1,2,2)
plt.hist(np.array(cropped_image).flatten(), bins=bins, density=True);
plt.title("Cropped image")
```

Out[ ]:
```
Text(0.5, 1.0, 'Cropped image')
```



In [ ]:
```
np.histogram(cropped_image, density=True, bins=44)[0]
```

Out[ ]:
```
array([1.04134125e-06, 1.09340831e-05, 2.91575549e-05, 5.98771217e-05,
       1.62969905e-04, 2.72831407e-04, 5.40976778e-04, 1.34645423e-03,
       1.73331251e-03, 2.98031865e-03, 3.02613767e-03, 3.45412892e-03,
       5.35926273e-03, 5.92106633e-03, 1.05456628e-02, 1.08179735e-02,
       1.24940123e-02, 1.72341976e-02, 1.49073206e-02, 1.82562741e-02,
       1.38810788e-02, 1.33609289e-02, 1.54394460e-02, 1.02962616e-02,
       8.92950120e-03, 4.06799958e-03, 2.55649276e-03, 1.25325419e-03,
       8.96074143e-04, 8.32031657e-04, 5.44621472e-04, 6.52920962e-04,
       5.92002499e-04, 8.02874102e-04, 3.15370197e-03, 1.18863897e-02,
       1.18811830e-02, 4.03311465e-03, 4.09351244e-03, 4.74435072e-03,
       2.21805686e-03, 1.10798709e-03, 2.87930855e-04, 1.38498386e-04])
```

In [ ]:
```
hists = []
y = []
for i in range(10,17):  # this are the folders (number/name)
    image_list = [f for f in glob.glob("./arroz_2023_gris/"+str(i)+"/*.jpg")]
    print(image_list)
    y += [i]*len(image_list)
    for image_path in image_list:
        grayscale_image = convert_to_grayscale(image_path)[0]
        cropped_image = crop_center(grayscale_image)[0]
        hists += [np.histogram(cropped_image, density=True, bins=44)[0]]

np.array(hists).shape, np.array(y)
```

['./arroz_2023_gris/10\\a1.jpg', './arroz_2023_gris/10\\a10.jpg', './arroz_2023_gris/10\\a12.jpg', './arroz_2023_gris/10\\a13.jpg', './arroz_2023_gris/10\\a14.jpg', './arroz_2023_gris/10\\a15.jpg', './arroz_2023_gris/10\\a16.jpg', './arroz_2023_gris/10\\a17.jpg', './arroz_2023_gris/10\\a18.jpg', './arroz_2023_gris/10\\a19.jpg', './arroz_2023_gris/10\\a2.jpg', './arroz_2023_gris/10\\a20.jpg', './arroz_2023_gris/10\\a21.jpg', './arroz_2023_gris/10\\a22.jpg', './arroz_2023_gris/10\\a23.jpg', './arroz_2023_gris/10\\a24.jpg', './arroz_2023_gris/10\\a25.jpg', './arroz_2023_gris/10\\a26.jpg', './arroz_2023_gris/10\\a27.jpg', './arroz_2023_gris/10\\a28.jpg', './arroz_2023_gris/10\\a29.jpg', './arroz_2023_gris/10\\a3.jpg', './arroz_2023_gris/10\\a30.jpg', './arroz_2023_gris/10\\a4.jpg', './arroz_2023_gris/10\\a5.jpg', './arroz_2023_gris/10\\a6.jpg', './arroz_2023_gris/10\\a7.jpg', './arroz_2023_gris/10\\a8.jpg', './arroz_2023_gris/10\\a9.jpg']
['./arroz_2023_gris/11\\b1.jpg', './arroz_2023_gris/11\\b10.jpg', './arroz_2023_gris/11\\b11.jpg', './arroz_2023_gris/11\\b12.jpg', './arroz_2023_gris/11\\b13.jpg', './arroz_2023_gris/11\\b14.jpg', './arroz_2023_gris/11\\b15.jpg', './arroz_2023_gris/11\\b16.jpg', './arroz_2023_gris/11\\b17.jpg', './arroz_2023_gris/11\\b18.jpg', './arroz_2023_gris/11\\b19.jpg', './arroz_2023_gris/11\\b2.jpg', './arroz_2023_gris/11\\b20.jpg', './arroz_2023_gris/11\\b21.jpg', './arroz_2023_gris/11\\b22.jpg', './arroz_2023_gris/11\\b23.jpg', './arroz_2023_gris/11\\b24.jpg', './arroz_2023_gris/11\\b25.jpg', './arroz_2023_gris/11\\b26.jpg', './arroz_2023_gris/11\\b27.jpg', './arroz_2023_gris/11\\b28.jpg', './arroz_2023_gris/11\\b29.jpg', './arroz_2023_gris/11\\b3.jpg', './arroz_2023_gris/11\\b30.jpg', './arroz_2023_gris/11\\b31.jpg', './arroz_2023_gris/11\\b4.jpg', './arroz_2023_gris/11\\b5.jpg', './arroz_2023_gris/11\\b6.jpg', './arroz_2023_gris/11\\b7.jpg', './arroz_2023_gris/11\\b8.jpg', './arroz_2023_gris/11\\b9.jpg']
['./arroz_2023_gris/12\\c1.jpg', './arroz_2023_gris/12\\c10.jpg', './arroz_2023_gris/12\\c11.jpg', './arroz_2023_gris/12\\c12.jpg', './arroz_2023_gris/12\\c13.jpg', './arroz_2023_gris/12\\c14.jpg', './arroz_2023_gris/12\\c15.jpg', './arroz_2023_gris/12\\c16.jpg', './arroz_2023_gris/12\\c17.jpg', './arroz_2023_gris/12\\c18.jpg', './arroz_2023_gris/12\\c19.jpg', './arroz_2023_gris/12\\c2.jpg', './arroz_2023_gris/12\\c20.jpg', './arroz_2023_gris/12\\c21.jpg', './arroz_2023_gris/12\\c22.jpg', './arroz_2023_gris/12\\c23.jpg', './arroz_2023_gris/12\\c24.jpg', './arroz_2023_gris/12\\c25.jpg', './arroz_2023_gris/12\\c26.jpg', './arroz_2023_gris/12\\c27.jpg', './arroz_2023_gris/12\\c28.jpg', './arroz_2023_gris/12\\c29.jpg', './arroz_2023_gris/12\\c3.jpg', './arroz_2023_gris/12\\c30.jpg', './arroz_2023_gris/12\\c31.jpg', './arroz_2023_gris/12\\c32.jpg', './arroz_2023_gris/12\\c4.jpg', './arroz_2023_gris/12\\c5.jpg', './arroz_2023_gris/12\\c6.jpg', './arroz_2023_gris/12\\c7.jpg', './arroz_2023_gris/12\\c8.jpg', './arroz_2023_gris/12\\c9.jpg']
['./arroz_2023_gris/13\\d1.jpg', './arroz_2023_gris/13\\d10.jpg', './arroz_2023_gris/13\\d11.jpg', './arroz_2023_gris/13\\d12.jpg', './arroz_2023_gris/13\\d13.jpg', './arroz_2023_gris/13\\d14.jpg', './arroz_2023_gris/13\\d15.jpg', './arroz_2023_gris/13\\d16.jpg', './arroz_2023_gris/13\\d17.jpg', './arroz_2023_gris/13\\d18.jpg', './arroz_2023_gris/13\\d19.jpg', './arroz_2023_gris/13\\d2.jpg', './arroz_2023_gris/13\\d20.jpg', './arroz_2023_gris/13\\d21.jpg', './arroz_2023_gris/13\\d22.jpg', './arroz_2023_gris/13\\d23.jpg', './arroz_2023_gris/13\\d24.jpg', './arroz_2023_gris/13\\d25.jpg', './arroz_2023_gris/13\\d26.jpg', './arroz_2023_gris/13\\d27.jpg', './arroz_2023_gris/13\\d28.jpg', './arroz_2023_gris/13\\d29.jpg', './arroz_2023_gris/13\\d3.jpg', './arroz_2023_gris/13\\d30.jpg', './arroz_2023_gris/13\\d31.jpg', './arroz_2023_gris/13\\d32.jpg', './arroz_2023_gris/13\\d33.jpg', './arroz_2023_gris/13\\d34.jpg', './arroz_2023_gris/13\\d35.jpg', './arroz_2023_gris/13\\d4.jpg', './arroz_2023_gris/13\\d5.jpg', './arroz_2023_gris/13\\d6.jpg', './arroz_2023_gris/13\\d7.jpg', './arroz_2023_gris/13\\d8.jpg', './arroz_2023_gris/13\\d9.jpg']
['./arroz_2023_gris/14\\e1.jpg', './arroz_2023_gris/14\\e10.jpg', './arroz_2023_gris/14\\e11.jpg', './arroz_2023_gris/14\\e12.jpg', './arroz_2023_gris/14\\e13.jpg', './arroz_2023_gris/14\\e14.jpg', './arroz_2023_gris/14\\e15.jpg', './arroz_2023_gris/14\\e16.jpg', './arroz_2023_gris/14\\e17.jpg', './arroz_2023_gris/14\\e18.jpg', './arroz_2023_gris/14\\e19.jpg', './arroz_2023_gris/14\\e2.jpg', './arroz_2023_gris/14\\e20.jpg', './arroz_2023_gris/14\\e21.jpg', './arroz_2023_gris/14\\e22.jpg', './arroz_2023_gris/14\\e23.jpg', './arroz_2023_gris/14\\e24.jpg', './

```
arroz_2023_gris/14\\e25.jpg', './arroz_2023_gris/14\\e26.jpg', './arroz_2023_gri
s/14\\e27.jpg', './arroz_2023_gris/14\\e28.jpg', './arroz_2023_gris/14\\e29.jpg',
'./arroz_2023_gris/14\\e3.jpg', './arroz_2023_gris/14\\e30.jpg', './arroz_2023_gr
is/14\\e31.jpg', './arroz_2023_gris/14\\e4.jpg', './arroz_2023_gris/14\\e5.jpg',
'./arroz_2023_gris/14\\e6.jpg', './arroz_2023_gris/14\\e7.jpg', './arroz_2023_gri
s/14\\e8.jpg', './arroz_2023_gris/14\\e9.jpg']
['./arroz_2023_gris/15\\f1.jpg', './arroz_2023_gris/15\\f10.jpg', './arroz_2023_g
ris/15\\f11.jpg', './arroz_2023_gris/15\\f12.jpg', './arroz_2023_gris/15\\f13.jp
g', './arroz_2023_gris/15\\f14.jpg', './arroz_2023_gris/15\\f15.jpg', './arroz_20
23_gris/15\\f16.jpg', './arroz_2023_gris/15\\f17.jpg', './arroz_2023_gris/15\\f1
8.jpg', './arroz_2023_gris/15\\f19.jpg', './arroz_2023_gris/15\\f2.jpg', './arroz
_2023_gris/15\\f20.jpg', './arroz_2023_gris/15\\f21.jpg', './arroz_2023_gris/15
\\f22.jpg', './arroz_2023_gris/15\\f23.jpg', './arroz_2023_gris/15\\f24.jpg', './
arroz_2023_gris/15\\f25.jpg', './arroz_2023_gris/15\\f26.jpg', './arroz_2023_gri
s/15\\f27.jpg', './arroz_2023_gris/15\\f28.jpg', './arroz_2023_gris/15\\f29.jpg',
'./arroz_2023_gris/15\\f3.jpg', './arroz_2023_gris/15\\f30.jpg', './arroz_2023_gr
is/15\\f31.jpg', './arroz_2023_gris/15\\f4.jpg', './arroz_2023_gris/15\\f5.jpg',
'./arroz_2023_gris/15\\f6.jpg', './arroz_2023_gris/15\\f7.jpg', './arroz_2023_gri
s/15\\f8.jpg', './arroz_2023_gris/15\\f9.jpg']
['./arroz_2023_gris/16\\g1.jpg', './arroz_2023_gris/16\\g10.jpg', './arroz_2023_g
ris/16\\g11.jpg', './arroz_2023_gris/16\\g12.jpg', './arroz_2023_gris/16\\g13.jp
g', './arroz_2023_gris/16\\g14.jpg', './arroz_2023_gris/16\\g15.jpg', './arroz_20
23_gris/16\\g16.jpg', './arroz_2023_gris/16\\g17.jpg', './arroz_2023_gris/16\\g1
8.jpg', './arroz_2023_gris/16\\g19.jpg', './arroz_2023_gris/16\\g2.jpg', './arroz
_2023_gris/16\\g20.jpg', './arroz_2023_gris/16\\g21.jpg', './arroz_2023_gris/16
\\g22.jpg', './arroz_2023_gris/16\\g23.jpg', './arroz_2023_gris/16\\g24.jpg', './
arroz_2023_gris/16\\g25.jpg', './arroz_2023_gris/16\\g26.jpg', './arroz_2023_gri
s/16\\g27.jpg', './arroz_2023_gris/16\\g28.jpg', './arroz_2023_gris/16\\g29.jpg',
'./arroz_2023_gris/16\\g3.jpg', './arroz_2023_gris/16\\g4.jpg', './arroz_2023_gri
s/16\\g5.jpg', './arroz_2023_gris/16\\g6.jpg', './arroz_2023_gris/16\\g7.jpg',
'./arroz_2023_gris/16\\g8.jpg', './arroz_2023_gris/16\\g9.jpg']
```

Out[ ]:
```
((218, 44),
 array([10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
        10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11,
        11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
        11, 11, 11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12, 12, 12,
        12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
        12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
        13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
        13, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14,
        14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
        14, 14, 14, 14, 14, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
        15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
        15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
        16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16]))
```

In [ ]:
```
data = pd.DataFrame(np.array(hists), columns=[str(i) for i in range(44)])
data['humidity'] = np.array(y)
data
```

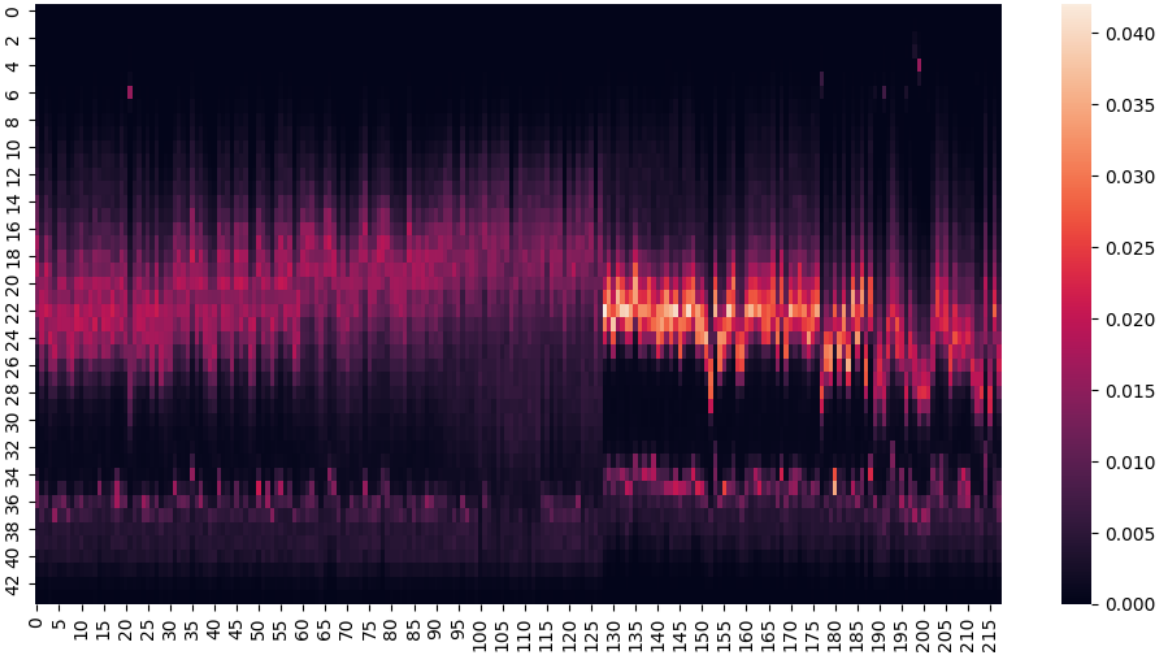
Out[ ]:

	0	1	2	3	4	5	
0	1.041341e-06	1.093408e-05	2.915755e-05	5.987712e-05	1.629699e-04	2.728314e-04	5.409768e-0
1	4.742258e-07	4.742258e-07	4.742258e-07	1.896903e-06	5.216484e-06	1.849481e-05	2.181439e-0
2	5.206706e-07	1.041341e-06	7.810059e-06	2.290951e-05	9.111736e-05	1.686973e-04	3.483286e-0
3	5.153577e-07	2.061431e-06	9.276438e-06	1.236858e-05	4.483612e-05	8.297258e-05	2.731396e-0
4	8.899568e-07	8.899568e-07	1.334935e-06	1.779914e-06	2.669871e-06	2.669871e-06	3.559827e-0
...	...	...	...	...	...	...	
213	5.426162e-06	8.347942e-06	4.090492e-05	7.304449e-05	4.925286e-05	2.713081e-05	2.546122e-0
214	9.722897e-06	1.836547e-05	3.457030e-05	7.616270e-05	2.003997e-04	3.732512e-04	6.816831e-0
215	3.593245e-06	5.589492e-06	1.038048e-05	1.796622e-05	6.028666e-05	5.389867e-05	4.551443e-0
216	1.665002e-06	2.220002e-06	5.550006e-06	2.275502e-05	7.936508e-05	1.870352e-04	4.123654e-0
217	4.509380e-07	0.000000e+00	4.509380e-07	9.018759e-07	9.018759e-07	4.509380e-07	9.018759e-0

218 rows × 45 columns

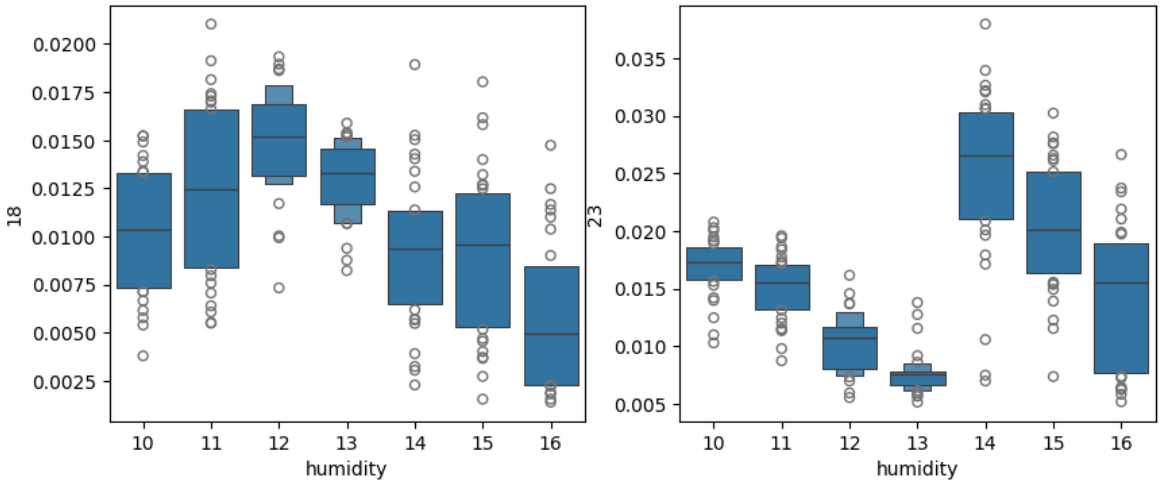
In [ ]:
```
plt.figure(figsize=(12,6))
sns.heatmap(data.iloc[:,:-1].T)
```

Out[ ]: <Axes: >

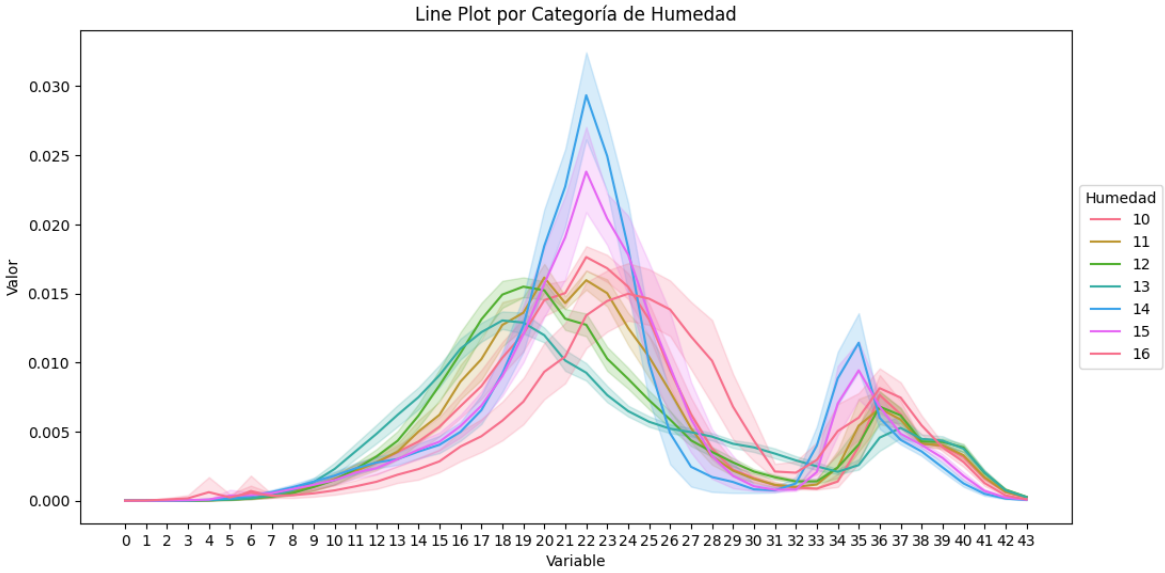


In [ ]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.boxenplot(data=data, x='humidity', y='18')
plt.subplot(1,2,2)
sns.boxenplot(data=data, x='humidity', y='23')
```

Out[ ]: <Axes: xlabel='humidity', ylabel='23'>



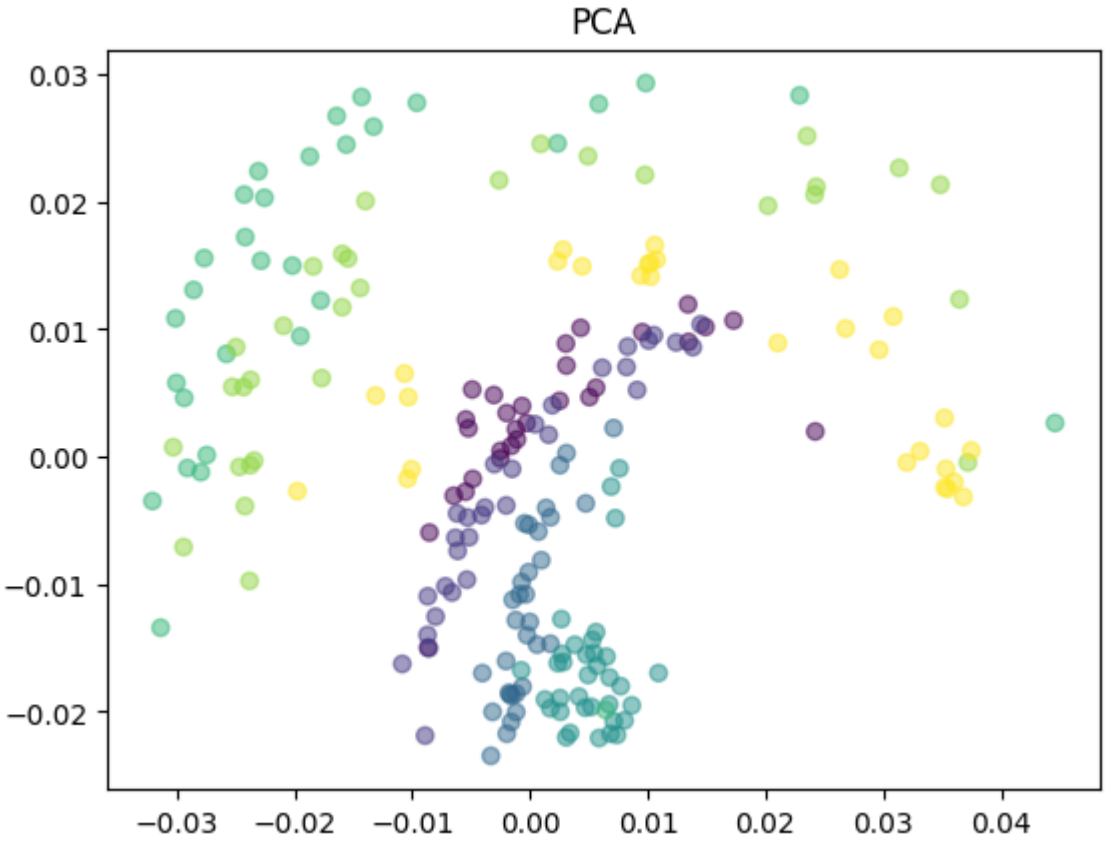
In [ ]:
```
# Crear un line plot con seaborn
plt.figure(figsize=(12, 6))
sns.lineplot(data=data.melt(id_vars='humidity'), x='variable', y='value', hue='h

# Personalizar el gráfico
plt.title('Line Plot por Categoría de Humedad')
plt.xlabel('Variable')
plt.ylabel('Valor')
plt.legend(title='Humedad', loc='center left', bbox_to_anchor=(1, 0.5))

# Mostrar el gráfico
plt.show()
```



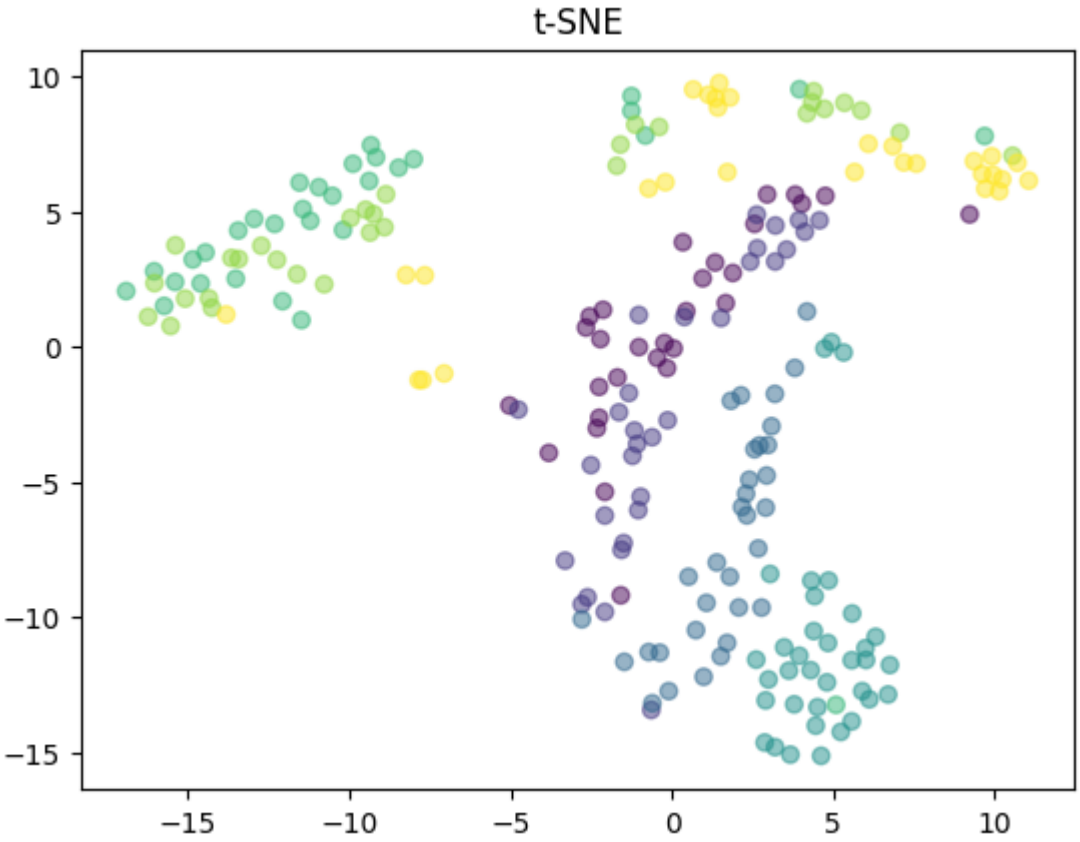
In [ ]:
```
# Get X and labels y
X = data.iloc[:,:-1]
y = data['humidity']

le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

In [ ]:
```
# Codificar etiquetas
le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

In [ ]:
```
pca = PCA(n_components=44)  # Elige el número de componentes principales deseado
X_pca = pca.fit_transform(X)
# Visualizar resultados de PCA
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_encoded, cmap='viridis', alpha=0.5)
plt.title('PCA')
plt.show()
```



In [ ]:
```
# Aplicar t-SNE para visualización en 2D
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
X_tsne = tsne.fit_transform(X)
# Visualizar resultados de t-SNE
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_encoded, cmap='viridis', alpha=0.5)
plt.title('t-SNE')
plt.show()
```



Detalle de Escenarios:

Primer Escenario: Clasificación (Classifiers) En este escenario, el objetivo principal es construir modelos clasificadores capaces de asignar correctamente las imágenes de granos de arroz a diferentes niveles de humedad. Las clases de humedad se definen en el rango de 10 a 16. Cada modelo clasificador estará entrenado para identificar la humedad específica de un grano de arroz en función de sus características visuales. Este enfoque permite la categorización precisa de los granos en diferentes niveles de humedad, proporcionando una herramienta valiosa para la calidad y el control de procesos en la producción de arroz.

Segundo Escenario: Regresión (Regressors) En este escenario, el objetivo es construir modelos de regresión que estimen el contenido de humedad de los granos de arroz como un número real. Sin embargo, para hacer que las predicciones sean más prácticas, se aplica una restricción adicional: redondear las predicciones al entero más cercano. Por ejemplo, si un modelo predice 8.9, se redondeará a 10; si predice 16.8, se redondeará a 16. Además, se establecen límites para las salidas del regresor. Si la predicción es menor que 10, se establece en 10; si es mayor que 16, se establece en 16. Este escenario proporciona una perspectiva más continua del contenido de humedad, permitiendo una evaluación más detallada del rendimiento del modelo en la predicción de valores reales.

In [ ]:
```
# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2,

# Entrenar un clasificador Random Forest
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
```

```
# Realizar predicciones en el conjunto de prueba
y_pred = rf_classifier.predict(X_test)

# Decodificar las etiquetas si es necesario
y_test_decoded = le.inverse_transform(y_test)
y_pred_decoded = le.inverse_transform(y_pred)

# Evaluar el rendimiento del clasificador
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Imprimir el reporte de clasificación
print('Classification Report:\n', classification_report(y_test_decoded, y_pred_d
```

```
Accuracy: 0.80
Classification Report:
               precision    recall  f1-score   support

           10       0.57      0.80      0.67         5
           11       1.00      1.00      1.00         3
           12       0.83      1.00      0.91         5
           13       1.00      0.88      0.93         8
           14       1.00      0.75      0.86         8
           15       0.71      0.71      0.71         7
           16       0.62      0.62      0.62         8

    accuracy                           0.80        44
   macro avg       0.82      0.82      0.82        44
weighted avg       0.82      0.80      0.80        44
```

In [ ]:

```
# Regresión con límites
# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2,

# Entrenar un regresor Random Forest
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = rf_regressor.predict(X_test)

# Aplicar restricciones a las predicciones
y_pred = np.round(y_pred)  # Redondear al entero más cercano
y_pred = np.clip(y_pred, 10, 16)  # Aplicar límites a las predicciones

# Evaluar el rendimiento del regresor
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
```

```
Mean Squared Error: 46.50
```

Uso de pipelines para exploración de modelos de clasificación y regresión

Para explorar el desempeño de los modelos de clasificación y regresión con los distintos datasets generados, se debe realizar una exploración de varias configuraciones de entrenamiento de los modelos de inteligencia artificial. Entrenar modelos puede tomar

largo tiempo, más aun cuando se trata de una exploración para encontrar las mejores combinaciones de datos e hiperparámetros de entrenamiento. Cuando se habla de hiperparámetros de entrenamiento, se hace referencia a configuraciones que pueden cambiarse de los modelos para afectar la manera en al que aprender y obtener mejores resultados. Considerando esto, se utiliza la herramienta de pipelines de la librería scikit-learn, que permite configurar el entrenamiento de múltiples modelos para ejecutarlos en grupo y obtener los resultados de desempeño de todos. Estos pipelines permiten definir pasos que deben seguirse, modelos que se deben entrenar, combinaciones de hiperparámetros que probar, y los datos que se deben utilizar para su ejecución. A este pipeline se le indica que debe realizar un "grid search", que es la exploración de hiperparámetros para varios modelos, para cada dataset.

Selección de parámetros a explorar

El ajuste de hiperparámetros que se explorará para MLP se basa en el siguiente razonamiento:

Learning Rate (velocidad de aprendizaje): establece que tanto se deben cambiar los parámetros, se deben explorar valores de learning rate desde pequeños a grandes. Esto permite encontrar el rango de LRs que no son demasiado grandes como para no llegar al loss mínimo y no muy pequeños como para necesitar demasiadas épocas de entrenamiento. Batch size (tamaño de muestra): indica después de cuantos samples se debe recalcular el loss y ajustar pesos, si es muy pequeño los ajustes de dan más seguido y mejora el accuracy pero se requiere mucho más poder computacional. Se debe encontrar el valor que permite un buen accuracy pero no demande demasiados recursos y sea ineficiente. Hidden neurons (neuronas escondidas): la cantidad de neuronas afecta que tan complejo es un modelo, si un modelo es demasiado simple no es capaz de aprender patrones para funcionar correctamente. Pero se debe cuidar que la complejidad no sea tan alta que consuma mucho tiempo entrenar o exista overfitting, esto último es cuando el modelo aprende mucho de los datos de entrenamiento y se vuelve incapaz de generalizar para funcionar adecuadamente con datos reales o de prueba. Max iter (pasos de entrenamiento): indica cuantas veces el modelo debe entrenar con los datos, es decir, la cantidad de ajustes. Si se hacen demasiados se puede gastar mucho tiempo o hasta tener overfitting, si se hacen muy pocos el modelo no aprende lo suficiente para tener buen desempeño.

Respecto a la exploración de hiperparámetros para Random Forest, se tiene consideración los siguientes efectos de los hiperparámetros en dicho tipo de modelo:

Number of estimators (número de estimadores): la cantidad de árboles en el Random Forest, con más árboles existe más precisión, pero una cantidad demasiado grande es ineficiente para tiempo de entrenamiento. Max depth (máxima profundidad): se refiere a que tantas divisiones (decisiones) puede tener cada árbol en el modelo, si es muy bajo

no hay suficiente aprendizaje, llamado underfitting (Ram, 2020). Min samples split (mínima cantidad de muestras para división): especifica cuantas muestras debe tener un nodo del árbol para dividirse y tomar una decisión, se busca una cantidad de muestras que le permita al árbol lograr suficientes divisiones para evitar underfitting. Para ambas exploraciones, no existe un criterio específico que permita saber qué valores usar, se deben explorar valores que se sepa permitan mejor rendimiento; por ejemplo, más árboles en Random Forest, o más iteraciones en MLP.

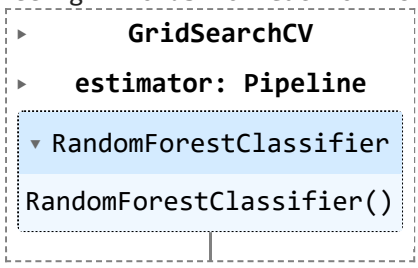
In [ ]:
```
param_grid = [
{
    'estimator': [RandomForestClassifier()],
    'estimator__n_estimators': [100, 200],
    'estimator__max_depth': [5, 10, 15],
    'estimator__min_samples_split': [2],
},
{
    'estimator': [MLPClassifier()],
    'estimator__learning_rate_init': [0.001, 0.01],
    'estimator__batch_size': [1024],
    'estimator__max_iter': [400, 800],
    'estimator__hidden_layer_sizes': [(100,), (50, 50)],
},
{
    'estimator': [MLPRegressor()],
    'estimator__learning_rate_init': [0.001, 0.01],
    'estimator__batch_size': [1024],
    'estimator__max_iter': [400, 800],
    'estimator__hidden_layer_sizes': [(100,), (50, 50)],
},
{'estimator': [RandomForestRegressor()],
    'estimator__n_estimators': [100, 200],
    'estimator__max_depth': [5, 10, 15],
    'estimator__min_samples_split': [2],
}
]
def round_pred(y_pred):
    # Round the predictions to the nearest integer
    # If the predictions are outside the range [10, 16], clip them
    # This will not have any effect on classification
    convert_to_nearest = y_pred.clip(10, 16).round().astype(int)
    return convert_to_nearest

scoring = {
    'accuracy': make_scorer(lambda y_true, y_pred: accuracy_score(y_true, round_
}
```

In [ ]:
```
pipeline = Pipeline([
        ('estimator', RandomForestClassifier())  # Estimator
])
# Perform grid search with cross-validation
grid_search = GridSearchCV(
    pipeline,
    param_grid,
    scoring=scoring,
    cv=4,# Cross validation
    refit=False,
    verbose=10,
```

```
    n_jobs=5, # Parallelize the grid search using many CPU cores
    return_train_score=True
)

grid_search.fit(X, y)
```

```
Fitting 4 folds for each of 28 candidates, totalling 112 fits
```

Out[ ]:



Entrenamiento de modelos para exploración

Se entrenan los modelos con los datasets e hiperparámetros escogidos para determinar las mejores combinaciones de estos. Para realizar la validación del desempeño de los modelos, no se pueden utilizar las mismas muestras de entrenamiento, pues los modelos están aprendiendo con esto y las conoces cada vez mejor. Por lo tanto, se utiliza k-fold Cross Validation, esta técnica consiste en dividir el dataset en k partes iguales, y usar una de estas para validar y las demás para entrenar; se puede realizar el proceso k veces de modo que se use cada división una vez para validar (Brownlee, 2018). Se decide utilizar un valor de para el Cross Validation dado que permite 4 iteraciones por cada modelo a explorar (que no son demasiadas) pero mantiene un buen porcentaje de datos de entrenamiento, 75%, o 31500 muestras. Para agilizar los 560 entrenamiento que deben realizarse (28 combinaciones de modelos, multiplicado por 4 iteraciones de k-fold Cross Validation dado que , multiplicado por los 5 datasers existentes) se utiliza procesamiento paralelo con núcleos físicos de CPU, scikit learn no tiene soporte para GPU. Adicionalmente, se corrieron las exploraciones con los datasets en escala de grises y binarizado en un computadora y los demás en otra, para disminuir el tiempo de exploración; se guardaron los resultados en archivos con la librería de luego fueron unidos en un solo diccionario de resultados.

In [ ]:

```
estimator_types = {
    'classification': ['RandomForestClassifier', 'MLPClassifier'],
    'regression': ['RandomForestRegressor', 'MLPRegressor']
}
model_types = {
    'Random Forest': ['RandomForestClassifier', 'RandomForestRegressor'],
    'MLP': ['MLPClassifier', 'MLPRegressor']
}
def get_estimator_type(estimator_name):
    for estimator_type, estimator_names in estimator_types.items():
        if estimator_name in estimator_names:
            return estimator_type
    return None

def get_model_type(estimator_name):
    for model_type, estimator_names in model_types.items():
        if estimator_name in estimator_names:
```

```
            return model_type
    return None
```

Resultados

Se utilizan los datos generados por la exploración y se añade a que dataset y tipo de modelo (clasificación o regresión) pertenece ese entrenamiento en específico.

Mejores resultados

A continuación, se presentan los mejores resultados obtenidos con la media de exactitud/precisión de los datos de prueba (la media de la precisión para todos los k folds de validación). La tabla que se presenta abajo ordena a los mejores modelos resultantes de la exploración de hiperparámetros . El mejor modelo corresponde a un MLP de clasificación que se detalla más adelante.

In [ ]:
```
# Get the results of the grid search and sort by mean test accuracy (descending)
cv_results = pd.DataFrame()
current_cv_results = pd.DataFrame(grid_search.cv_results_).copy()
current_cv_results['estimator_name'] = current_cv_results['param_estimator'].app
current_cv_results['estimator_type'] = current_cv_results['estimator_name'].appl
current_cv_results['model_type'] = current_cv_results['estimator_name'].apply(ge
cv_results = pd.concat([cv_results, current_cv_results])
# Reset the index
cv_results.reset_index(inplace=True, drop=True)
processed_cv_results = cv_results[[
    'estimator_name',
    'estimator_type',
    'model_type',
    'param_estimator',
    'params',
    'mean_train_accuracy',
    'std_train_accuracy',
    'mean_test_accuracy',
    'std_test_accuracy',
    'rank_test_accuracy' # For each type
]]
ranked_cv_results = processed_cv_results\
    .sort_values('mean_test_accuracy', ascending=False)\
    #.head(10)
ranked_cv_results
```

Out[ ]:

	estimator_name	estimator_type	model_type	param_estimator	
13	MLPClassifier	classification	MLP	MLPClassifier()	
12	MLPClassifier	classification	MLP	MLPClassifier()	
8	MLPClassifier	classification	MLP	MLPClassifier()	
9	MLPClassifier	classification	MLP	MLPClassifier()	
3	RandomForestClassifier	classification	Random Forest	RandomForestClassifier()	Randc
0	RandomForestClassifier	classification	Random Forest	RandomForestClassifier()	Randc
4	RandomForestClassifier	classification	Random Forest	RandomForestClassifier()	Randc
5	RandomForestClassifier	classification	Random Forest	RandomForestClassifier()	Randc
1	RandomForestClassifier	classification	Random Forest	RandomForestClassifier()	Randc
11	MLPClassifier	classification	MLP	MLPClassifier()	
2	RandomForestClassifier	classification	Random Forest	RandomForestClassifier()	Randc
26	RandomForestRegressor	regression	Random Forest	RandomForestRegressor()	Randor
24	RandomForestRegressor	regression	Random Forest	RandomForestRegressor()	Randor
27	RandomForestRegressor	regression	Random Forest	RandomForestRegressor()	Randor
25	RandomForestRegressor	regression	Random Forest	RandomForestRegressor()	Randor

	estimator_name	estimator_type	model_type	param_estimator	
22	RandomForestRegressor	regression	Random Forest	RandomForestRegressor()	Randor
23	RandomForestRegressor	regression	Random Forest	RandomForestRegressor()	Randor
10	MLPClassifier	classification	MLP	MLPClassifier()	
7	MLPClassifier	classification	MLP	MLPClassifier()	
6	MLPClassifier	classification	MLP	MLPClassifier()	
21	MLPRegressor	regression	MLP	MLPRegressor()	
19	MLPRegressor	regression	MLP	MLPRegressor()	
15	MLPRegressor	regression	MLP	MLPRegressor()	
18	MLPRegressor	regression	MLP	MLPRegressor()	
20	MLPRegressor	regression	MLP	MLPRegressor()	
14	MLPRegressor	regression	MLP	MLPRegressor()	
17	MLPRegressor	regression	MLP	MLPRegressor()	
16	MLPRegressor	regression	MLP	MLPRegressor()	

Análisis de resultados

No es posible señalar solamente los 10 mejores modelos y asumir combinaciones similares a estas son las únicas válidas. Es necesario visualizar una comparativa más

completa de los modelos entrenados y los datasets utilizados. En este punto, ya se conoce que datasets y modelos arrojan los mejores resultados, pero estos pueden ser casos muy apartados de los demás modelos de dataset, algoritmo o tipo de modelo, por lo cual se debe visualizar el rendimiento en general.

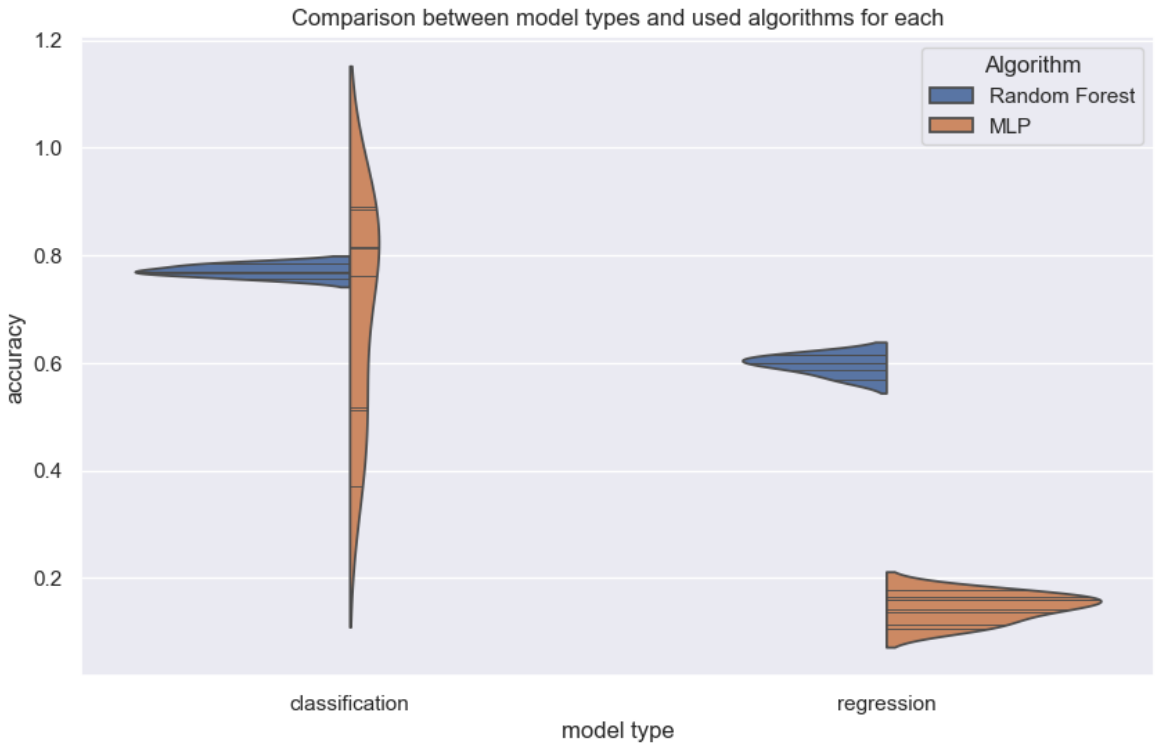
Visualización con diagramas de violín

- Los diagramas de violín permiten visualizar la distribución de los datos. Estos diagramas muestran la distribución, es decir, que tan común es un dato.
- En el presente caso, esta distribución indica que tantos modelos están cercanos a un valor X de la métrica de exactitud de validación, haciéndolos útiles para conocer donde se concentran la mayoría de modelos respecto a un algoritmo, tipo de modelo o dataset.
- Usando gráficos de violín, se puede visualizar con más exactitud donde están los valores de la distribución crean un gráfico de caja dentro de este, un gráfico de dispersión (scatter plot) o líneas. Para la visualización de los datos de este caso se usan líneas que indican donde están los puntos de datos, en este caso las exactitudes logradas por los modelos.
- Los gráficos de violín también permiten graficar dos distribuciones en el mismo violín, siendo ideal para comparaciones de grupos de datos, en este caso de modelos, algoritmos y datasets.

Comparación entre clasificación y regresión

- En el gráfico de violines que sigue, se comparan los resultados entre modelos de clasificación y regresión. Cada violín también compara el desempeño de los algoritmos usados para cada tipo de modelo (MLP y Random Forest)
- Observando los violines y las líneas que indican las exactitudes logradas por los modelos, es evidente que regresión no es ideal para el presente problema. Ningún modelo llegó a una exactitud mayor a 80%, la mayoria menor de 50% en el caso de MLP y 60% con Random Forest.
- El tipo de modelo ideal para este problema son los de clasificación. En el caso de MLP la mayoría de modelos llegaron a una exactitud de cerca del 90%, y Random forest también aunque con menos concentración cerca de este valor.

In [ ]:
```
sns.set(rc={'figure.figsize':(10,6)})
classVReg = violinplot(data=processed_cv_results, x='estimator_type', y='mean_te
                       split=True, inner='stick')
classVReg.set(xlabel='model type', ylabel='accuracy');
classVReg.set(title='Comparison between model types and used algorithms for each
classVReg.legend(title="Algorithm");
```



Conclusiones

- El uso de diagramas de violín para comparar dos algoritmos en un estudio proporciona una visualización efectiva y detallada de las distribuciones de rendimiento de ambos métodos. Estos gráficos permiten identificar diferencias en la mediana, la dispersión y la forma de las distribuciones, lo que facilita la comprensión de cuál algoritmo es más consistente y ofrece mejores resultados en diferentes métricas de evaluación.
- Para el análisis de los granos de arroz, se pudo evidenciar que los modelos de clasificación lograron alcanzar un nivel de precisión mayor que los de regresión. Esto se debe a que los modelos de clasificación son más adecuados para tareas donde las variables de salida son categóricas, como es el caso de la humedad en los granos, permitiendo alcanzar un mejor desempeño de forma general.
- La aplicación de técnicas de aprendizaje automático en la gestión de la humedad en granos de arroz representa un paso significativo hacia la innovación en la automatización agrícola. Al utilizar algoritmos avanzados, hemos demostrado que es posible mejorar la precisión y la eficiencia en tareas críticas para la producción de alimentos. Esta innovación no solo contribuye a la optimización de los procesos en la industria del arroz, sino que también establece un precedente para la aplicación de la inteligencia artificial en la mejora continua de la calidad y sostenibilidad en la producción agrícola a nivel global.