Análisis Exploratorio de Datos (EDA) en Profundidad:

Segmento dedicado a evaluar la calidad de la informacion proporcionada y mirar:

- Numero de registros
- Tipo de datos
- Estado de la informacion, outliners, missing data, balance
- Caracteristicas importantes, sesgos en entre otros

Preguntas

- **Distribución de necesidades por comuna y núcleo:** ¿Existen ciertas zonas geográficas con una mayor concentración de un tipo específico de necesidad?
- **Evolución temporal de las necesidades**: ¿Hay tendencias a lo largo del tiempo en cuanto al aumento o disminución de ciertas categorías?
- **Relación entre categorías y total de estudiantes**: ¿Existe una correlación entre el tipo de necesidad y el número total de estudiantes?

In [1]:
```
import os
from pathlib import Path

print("Actual path: ", os.getcwd())
os.chdir("..")
print("New path: ", os.getcwd())
os.environ["KERAS_BACKEND"] = "torch"
```

```
Actual path:  /home/bdebian/Documents/Projects/tecnical_test_crystal/src/not
ebooks
New path:  /home/bdebian/Documents/Projects/tecnical_test_crystal/src
```

Librerias de modelado

In [124…
```
import pickle

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import torch
from imblearn.under_sampling import RandomUnderSampler
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score,
    make_scorer,
```

```
    recall_score,
)
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

# sns.set_context("notebook")
torch.backends.cudnn.benchmark = True  # Para mejorar el rendimiento en GPU
torch.backends.cudnn.deterministic = False

# Librerias propias
from features.class_create_tree_folders import FolderArchitecture  # noqa: E
from lib.class_load import LoadFiles  # noqa: E402

hd_load = LoadFiles()
```

Estructura de carpetas

In [3]:
```
path_system = {
    "references": "",
    "reports": "",
    "src": {
        "notebooks": "",
        "lib": "",
        "data": {"sql": "", "config": "", "csv":""},
        "features": "",
        "models": "",
        "visualization": "",
    },
}
```

Generar y cargar rutas de archivos

In [4]:
```
execution_path = Path(os.getcwd()).parent
# Creacion de arquitectura de carpetas para su uso
folders_list, folder_dict = FolderArchitecture(
    arquitecture_tree=path_system, root_dir=execution_path
).create_tree_directory()
```

Cargar archivos y rutas de carpetas

Motivos para elegir el conjunto de datos

He elegido este conjunto de datos de necesidades educativas especiales en la educación formal en Colombia debido a varios factores clave:

- **Relevancia social:** Clasificar el tipo de necesidades educativas especiales (NEE) es fundamental para la asignación de recursos educativos y de apoyo especializado en diferentes regiones y grados escolares.

- **Variedad de características**: El dataset incluye tanto variables categóricas (tipo_servicio, comuna, grado, nombre) como numéricas (anio, mes, dane, total), lo que permite aplicar distintos modelos de clasificación que puedan trabajar con datos heterogéneos.
- **Tamaño suficiente del dataset**: Con más de 59,000 registros, hay suficiente información para entrenar y evaluar un modelo de clasificación de manera robusta.

In [5]:
```
#Cargar archivos de memoria
files_ = hd_load.search_load_files_extencion(
    path_search=folder_dict["references"], ext=["csv"]
)
names_csv, path_csv = files_["csv"]

#Cargar Informacion a RAM
data = pd.read_csv(path_csv[0])
display(data.info(show_counts=True))
display(data)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59756 entries, 0 to 59755
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   anio           59756 non-null  int64 
 1   mes            59756 non-null  int64 
 2   dane           59756 non-null  int64 
 3   tipo_servicio  59756 non-null  object
 4   comuna         59756 non-null  int64 
 5   nucleo         59756 non-null  int64 
 6   nombre         59756 non-null  object
 7   tipo_nee       59756 non-null  object
 8   grado          59756 non-null  object
 9   total          59756 non-null  int64 
dtypes: int64(6), object(4)
memory usage: 4.6+ MB
None
```

	anio	mes	dane	tipo_servicio	comuna	nucleo	nombre	tipo_nee
0	2019	9	105001000001	oficial	2	915	inst educ fe y alegria jose maria velaz	baja visión diagnosticada
1	2019	9	105001000001	oficial	2	915	inst educ fe y alegria jose maria velaz	deficiencia cognitiva (retardo mental)
2	2019	9	105001000001	oficial	2	915	inst educ fe y alegria jose maria velaz	deficiencia cognitiva (retardo mental)
3	2019	9	105001000001	oficial	2	915	inst educ fe y alegria jose maria velaz	deficiencia cognitiva (retardo mental)
4	2019	9	105001000001	oficial	2	915	inst educ fe y alegria jose maria velaz	deficiencia cognitiva (retardo mental)
...	...	...	...	...	...	...	...	...
59751	2018	12	405001800027	Privado	80	937	colegio campestre el encanto	intelectual
59752	2018	12	405001800027	Privado	80	937	colegio campestre el encanto	otra discapacidad
59753	2018	12	405001800027	Privado	80	937	colegio campestre el encanto	intelectual
59754	2018	12	405001800027	Privado	80	937	colegio campestre el encanto	múltiple
59755	2018	12	405001800027	Privado	80	937	colegio campestre el encanto	otra discapacidad

59756 rows × 10 columns

Preparacion de datos

Modificar columan de Necesidades Educativas Especiales, agrupando y seleccionando las caracteristicas que engloban parte de de las NEE, para simplificar la construccion del modelo dejando 6 principales

- **Auditiva**
- **Visual**
- **Voz**
- **Fisica**
- **Multiplies**
- **Intelectuales**

In [7]:
```
types_nee = {
    "Auditiva": [
        "audición",
        "sordo",
        "Hipoacusia",
        "sordos",
        "auditiva",
        "Sordera",
        "sordera",
    ],
    "visual": ["visual", "ceguera", "visión"],
    "voz": ["habla", "voz"],
    "física": [
        "fisica",
        "movilidad",
        "Otra discapacidad",
        "enanismo",
        "enanismo",
        "neuromuscular",
    ],
    "múltiple": [
        "Sordoceguera",
        "múltiple",
        "otra discapacidad",
        "Otra discapacidad",
        "sistémica",
        "múltiple",
    ],
    "Intelectual": [
        "down",
        "Mental",
        "intelectual",
        "psicosocial",
        "Parálisis",
        "mental",
        "autismo",
        "autista",
        "cerebral",
    ],
}

# Ciclo de preparacion de variables de casteo de columnas
```

```
cast_nee = {}
for key, val in types_nee.items():
    for col_name in val:
        cast_nee[col_name] = key


# Función para reemplazar los términos según el diccionario
def transformar_discapacidad(texto):
    for clave, valor in cast_nee.items():
        if clave in texto.lower():  # Buscamos la clave en el texto
            return valor
    return texto  # Si no se encuentra, se devuelve el texto original

# Reasignacion de categorias de discapacidad
data.loc[:, "categoria"] = data["tipo_nee"].apply(transformar_discapacidad)
```

Modificar la columna del ano de escolaridad para compactar la categorias de escolaridad con sus equivalalencias para simplificar la construccion del modelo

In [8]:
```
grado_update = {
    "primero": ["primero"],
    "segundo": ["segundo"],
    "tercero": ["tercero", "clei1"],
    "cuarto": ["cuarto"],
    "quinto": ["quinto", "clei2"],
    "sexto": ["sexto"],
    "séptimo": ["séptimo", "clei3"],
    "octavo": ["octavo"],
    "noveno": ["noveno", "clei4"],
    "décimo": ["décimo", "clei5"],
    "once": ["once", "clei6"],
    "doce": ["doce"],
    "trece": ["trece"],
    "catorce": ["catorce"],
    "otros": ["otros", "jardín", "pre-jardín", "párvulos", "aceleración", "t
}

# Ciclo de preparacion de variables de casteo de columnas
cast_grade = {}
for key, val in grado_update.items():
    for col_name in val:
        cast_grade[col_name] = key


# Función para reemplazar los términos según el diccionario
def transformar_discapacidad(texto):
    for clave, valor in cast_grade.items():
        if clave in texto.lower():  # Buscamos la clave en el texto
            return valor
    return texto  # Si no se encuentra, se devuelve el texto original


# Reasignacion de categorias de discapacidad
data.loc[:, "categoria_grado"] = data["grado"].apply(transformar_discapacida
```

Seleccionamos las columnas para trabajar el modelo de clasificacion , donde se omite directamente las columnas

- **dane**:Columna como identificador unico de cada registro realizado no aporta informacion
- **tipo_servicio**: Es una columna que en su mayoria no presenta variaciones en sus datos
- **nombre**: ya tiene un equivalente zonal que es la columna de `nucleo`

In [120…
```
data = data[
    ["anio", "mes", "comuna", "nucleo", "categoria", "categoria_grado", "tot
]
```

In [10]:
```
display(data.head())
display(data.info())
display(data.nunique())
```

	anio	mes	comuna	nucleo	categoria	categoria_grado	total
0	2019	9	2	915	visual	primero	3
1	2019	9	2	915	Intelectual	segundo	1
2	2019	9	2	915	Intelectual	tercero	1
3	2019	9	2	915	Intelectual	cuarto	1
4	2019	9	2	915	Intelectual	sexto	3

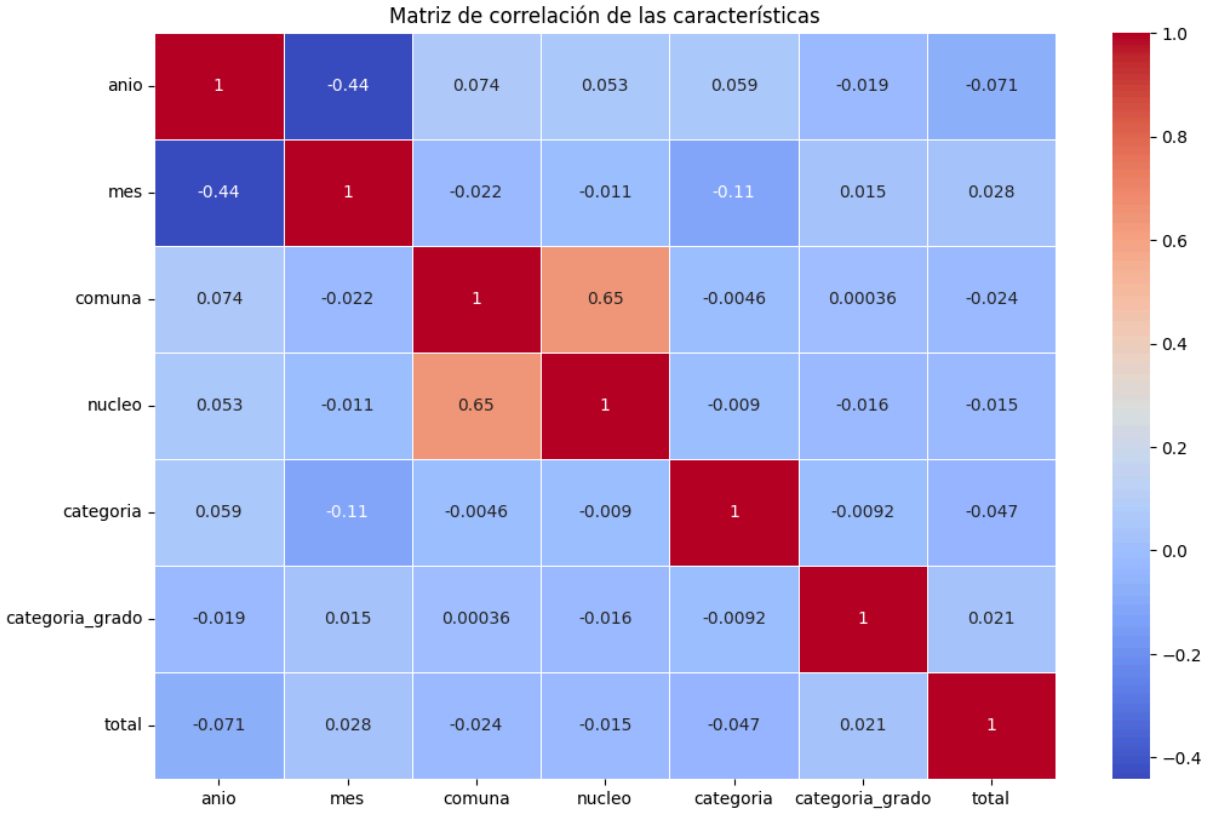
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59756 entries, 0 to 59755
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   anio             59756 non-null  int64
 1   mes              59756 non-null  int64
 2   comuna           59756 non-null  int64
 3   nucleo           59756 non-null  int64
 4   categoria        59756 non-null  object
 5   categoria_grado  59756 non-null  object
 6   total            59756 non-null  int64
dtypes: int64(5), object(2)
memory usage: 3.2+ MB
None
anio               16
mes                 2
comuna             21
nucleo             24
categoria           6
categoria_grado    15
total              91
dtype: int64
```

In [11]:
```
# Codificar variables categóricas con LabelEncoder
label_encoder = LabelEncoder()
```

```
data.loc[:, "categoria"] = label_encoder.fit_transform(data["categoria"])
data.loc[:, "categoria_grado"] = label_encoder.fit_transform(data["categoria
```

Graficos de relaciones

In [12]:
```
# Matriz de correlación
corr_matrix = data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Matriz de correlación de las características")
plt.show()
```

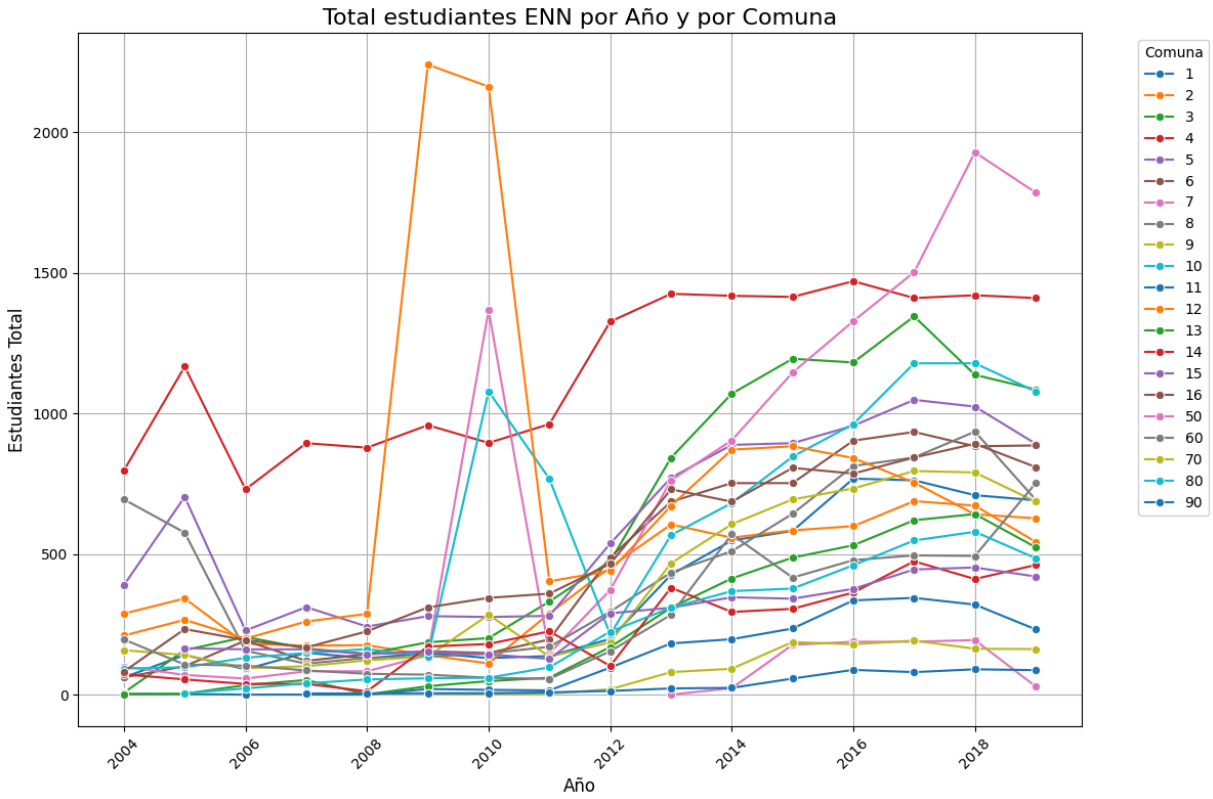


Correlaciones altas:

- **Comuna y Núcleo (0.65)**: Hay una correlación positiva moderada entre estas dos variables, lo que sugiere que a medida que se incrementa el valor de la comuna, también tiende a aumentar el valor del núcleo. Esto podría indicar que ciertas comunas tienen más núcleos educativos.
- **Categoría y Total (0.47)**: Indica que existe una correlación positiva moderada entre la categoría y el total, lo que podría sugerir que un aumento en el número de estudiantes en una categoría específica está asociado con un aumento en el total de estudiantes.
  Correlaciones bajas:

Las demás correlaciones son bastante bajas (cercanas a 0), lo que indica que no hay relaciones fuertes entre la mayoría de las otras variables. Por ejemplo, no hay correlaciones significativas entre el año y la mayoría de las otras variables, lo que puede indicar que el año no afecta de manera significativa a las características consideradas en este contexto.

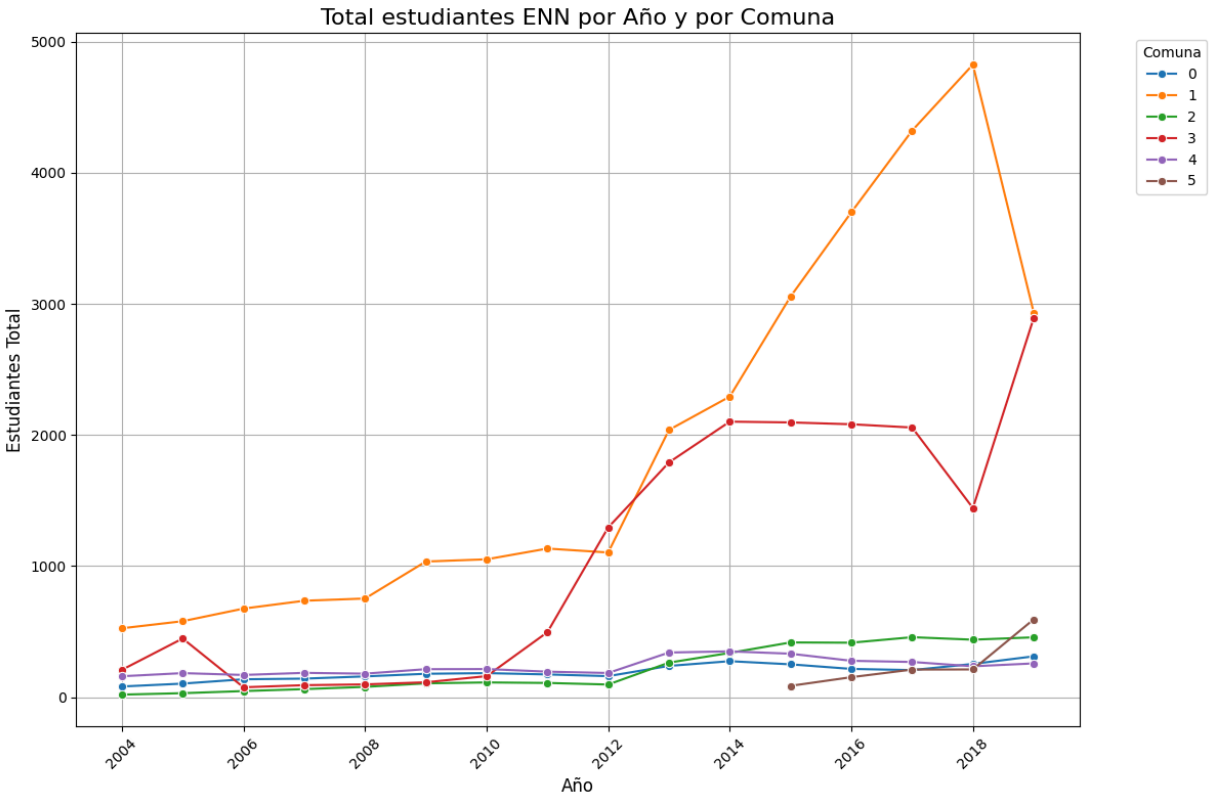
In [13]:
```
# Agrupar los datos por Año y NomComuna y sumar las inversiones
nee_total = data.groupby(["anio", "comuna"])["total"].sum().reset_index()

sns.set_palette("viridis")

plt.figure(figsize=(12, 8))
sns.lineplot(
    x="anio", y="total", hue="comuna", data=nee_total, marker="o", palette="
)

plt.title("Total estudiantes ENN por Año y por Comuna", fontsize=16)
plt.xlabel("Año", fontsize=12)
plt.ylabel("Estudiantes Total", fontsize=12)
plt.xticks(rotation=45)  # Rotar las etiquetas del eje X si es necesario
plt.legend(
    title="Comuna", bbox_to_anchor=(1.05, 1), loc="upper left"
)  # Para ajustar la leyenda
plt.grid(which="both")
plt.tight_layout()
plt.show()
```



¿Hay tendencias a lo largo del tiempo en cuanto al aumento o disminución de ciertas categorías?

In [14]:
```
nee_year = data.groupby(["anio"])["categoria"].value_counts().reset_index()
plt.figure(figsize=(12, 8))
sns.lineplot(
    x="anio", y="count", hue="categoria", data=nee_year, marker="o", palette
)
```

```
plt.title("Total estudiantes ENN por Año y por Comuna", fontsize=16)
plt.xlabel("Año", fontsize=12)
plt.ylabel("Estudiantes Total", fontsize=12)
plt.xticks(rotation=45)  # Rotar las etiquetas del eje X si es necesario
plt.legend(
    title="Comuna", bbox_to_anchor=(1.05, 1), loc="upper left"
)  # Para ajustar la leyenda
plt.grid(which="both")
plt.tight_layout()
plt.show()
```



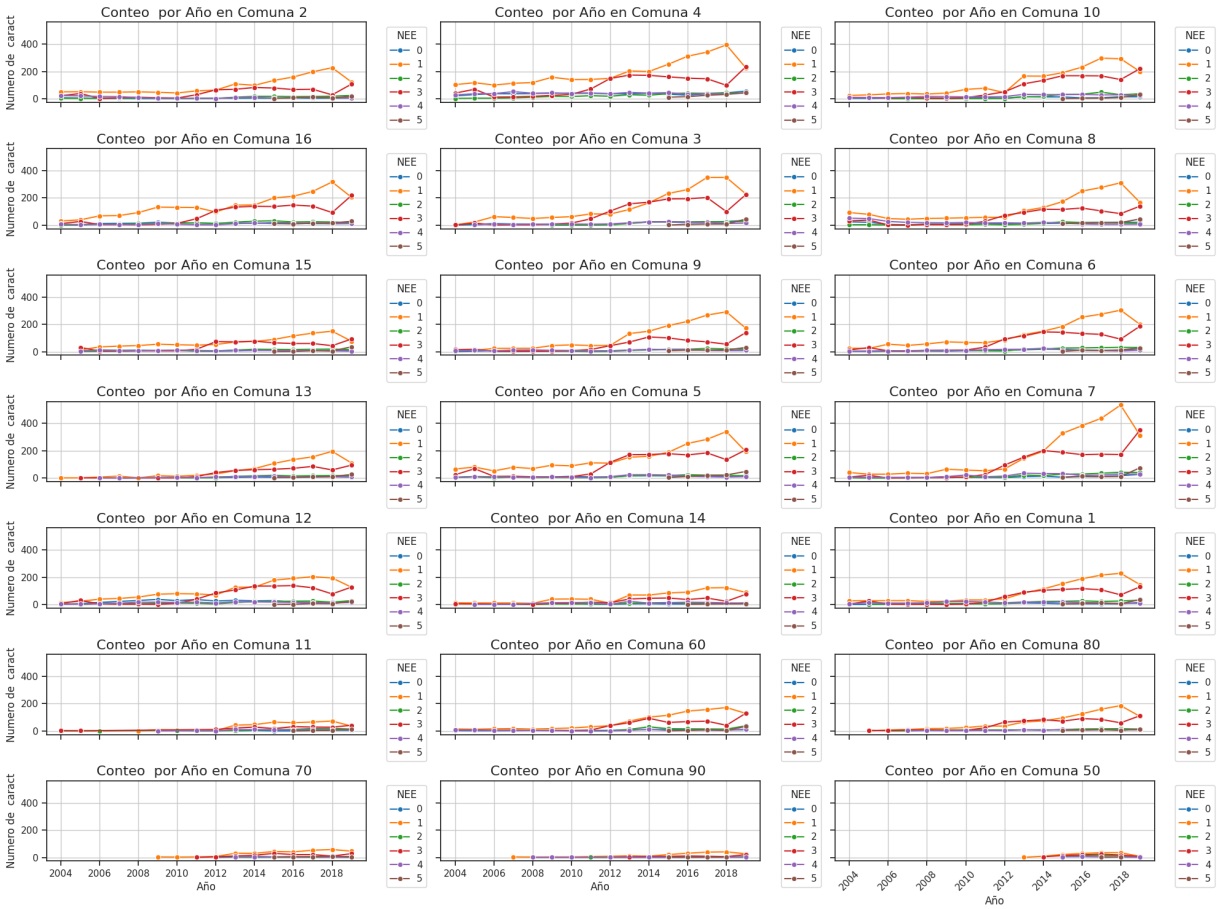
Tendencias generales

- **Crecimiento en el total de estudiantes**: Hay un aumento notable en el número total de estudiantes ENN a partir de 2011, que se intensifica en los años posteriores. Este aumento podría indicar una mayor inclusión de estudiantes con necesidades educativas especiales en el sistema educativo.
- **Diferencias entre comunas**:
  - **Comuna 0**: Muestra el mayor aumento, alcanzando un pico de alrededor de 4000 estudiantes en 2018, lo que sugiere que esta comuna ha estado liderando en la inclusión de estudiantes con necesidades especiales.
  - **Comunas 1 y 2**: También presentan un aumento significativo, pero a niveles menores que la comuna 0. La comuna 1 muestra un crecimiento constante, mientras que la comuna 2 experimenta un aumento repentino.
  - **Comunas 3, 4 y 5**: Estas comunas presentan un número mucho más bajo de estudiantes en comparación con las otras, y su crecimiento es menos pronunciado.

Conclusiones e implicaciones:

- **Políticas educativas**: Este crecimiento en el total de estudiantes ENN sugiere una posible mejora en las políticas de inclusión educativa. Sin embargo, las diferencias significativas entre comunas indican que aún puede haber disparidades en el acceso y la inclusión.
- **Investigación adicional**: Sería útil investigar las razones detrás de estos patrones, como el apoyo institucional, la disponibilidad de recursos, o factores sociales que puedan estar influyendo en la inclusión de estudiantes con necesidades educativas especiales.

In [15]:
```
nee_year = data.groupby(["anio", "comuna"])["categoria"].value_counts().rese

sns.set_palette("viridis")
sns.color_palette("Set2")
sns.set_theme(style="ticks", color_codes=True)

fig, axes = plt.subplots(7, 3, figsize=(20, 15), sharex=True, sharey=True)
axes = axes.flatten()

for i, n_comuna in enumerate(list(data["comuna"].unique())):
    # plt.title(f"Conteo  por Año en Comuna {n_comuna} ", fontsize=16)
    comuna = nee_year[nee_year["comuna"] == n_comuna]
    sns.lineplot(
        x="anio",
        y="count",
        hue="categoria",
        data=comuna,
        marker="o",
        palette="tab10",
        ax=axes[i]
    )
    axes[i].set_title(f"Conteo  por Año en Comuna {n_comuna} ", fontsize=16)
    axes[i].set_xlabel("Año", fontsize=12)
    axes[i].set_ylabel("Numero de  caract", fontsize=12)
    axes[i].legend(title="NEE", bbox_to_anchor=(1.05, 1), loc="upper left")
    axes[i].grid(axis="both")
    plt.xticks(rotation=45)  # Rotar las etiquetas del eje X si es necesaric
plt.tight_layout()
plt.show()
```



Modelacion

Clasificación de instituciones según la categoría de necesidad más prevalente:

- **Modelo**: Regresión logística multinomial o Random Forest.
- **Características**: Año, mes, comuna, núcleo, total por cada categoría.
- **Objetivo**: Clasificar las instituciones en categorías como "predominantemente visual", "predominantemente intelectual" o "diversidad de necesidades".

Expectativas del modelo

- **Clasificar el tipo de necesidad educativa especial (NEE)** en función de las características como el tipo de servicio, comuna y grado.
- **Identificar patrones relevantes** que indiquen qué características influyen más en la clasificación de las NEE, lo que puede ayudar a mejorar la asignación de recursos y personal especializado en las escuelas.
- **Generar recomendaciones** para anticipar qué tipos de necesidades podrían aparecer más frecuentemente en ciertas localidades o grados escolares.

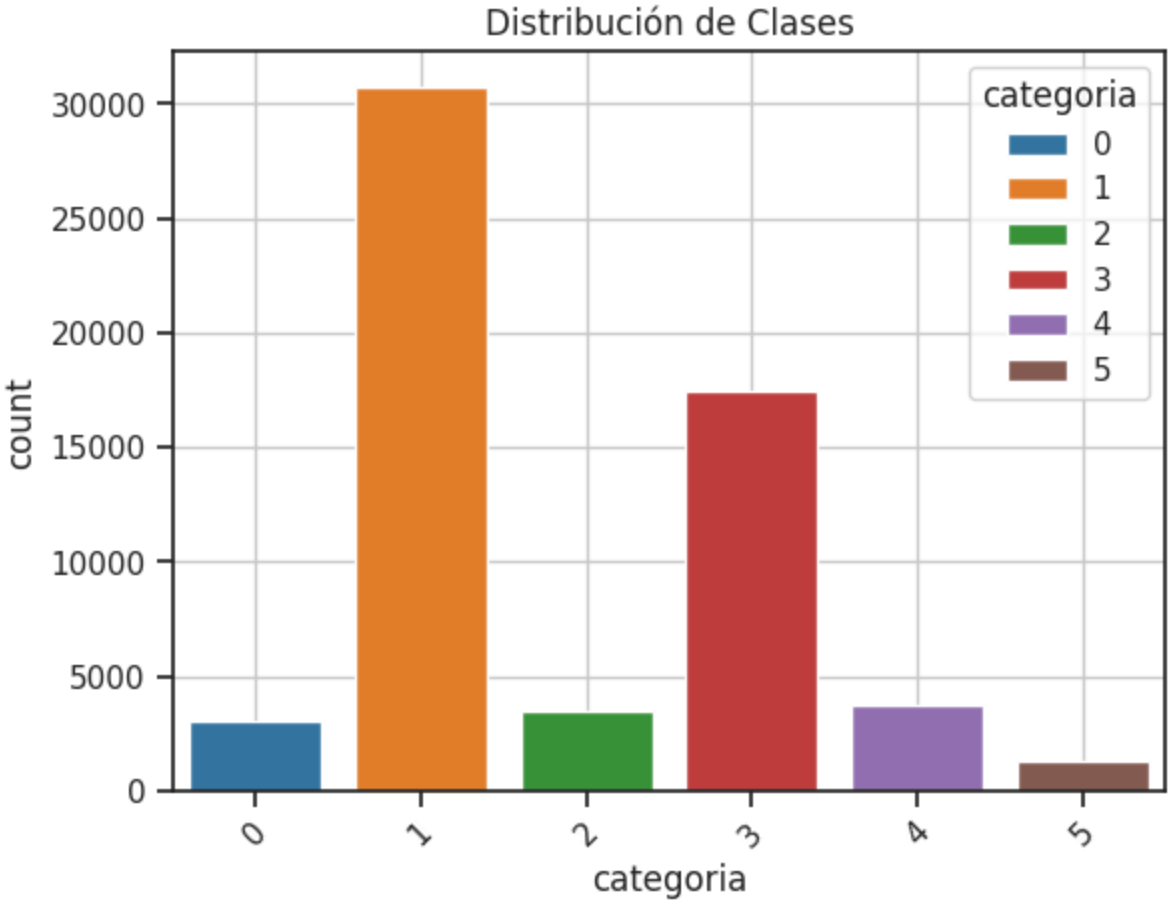
Visualizar balance de clases

In [38]:
```
view_variable = "categoria"

class_distribution = data[view_variable].value_counts()
display(pd.DataFrame(class_distribution))

class_proportion = data[view_variable].value_counts(normalize=True)
display("Procentaje de distribucion de los datos ")
display(pd.DataFrame(np.round(class_proportion * 100)))

sns.countplot(x=view_variable, data=data, hue=view_variable, palette="tab10"
plt.title("Distribución de Clases")
plt.xticks(rotation=45)  # Rotar las etiquetas del eje X si es necesario
plt.grid(which="both")
plt.show()
```

categoria	count
1	30764
3	17452
4	3753
2	3454
0	3076
5	1257

'Procentaje de distribucion de los datos '

categoria	proportion
1	51.0
3	29.0
4	6.0
2	6.0
0	5.0
5	2.0

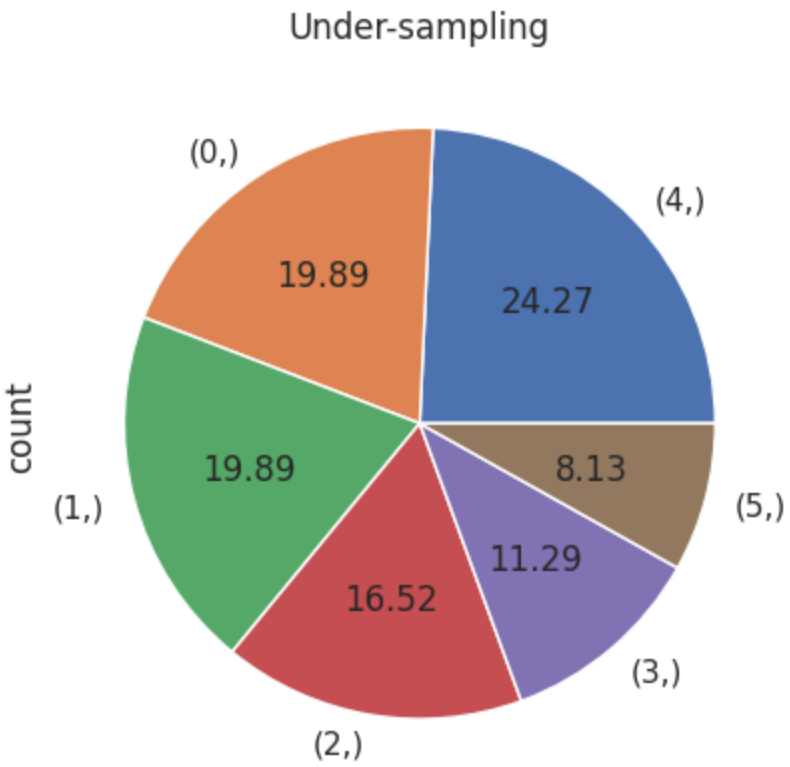


Visualizando los parametros y las distribuciones de las clases para clasificar se observa que `las categorias o tipo de NEE` estas desbalanceadas unas de otra por lo cual toca mirar alguna tecnica de balanceo

In [84]:
```
labes = ["categoria"]
features = list(filter(lambda x: x != labes[0], list(data.columns)))

sampling_strategy = (
    "not minority"  # {"all", "majority", "auto", "not majority", "not minor
)

def ratio_multiplier(y):
    from collections import Counter

    multiplier = {
        0:1,
        1: 0.1  ,
        2: 2 * 0.37,
        3: 1 * 0.1,
        4: 1,
        5: 1
        }
    target_stats = Counter(y)
    for key, value in target_stats.items():
        if key in multiplier:
            target_stats[key] = int(value * multiplier[key])
```

```
    return target_stats

X = data[features]
X = X.astype(int)
y = data[labes]
y = y.astype(int)
autopct = "%.2f"

rus = RandomUnderSampler(sampling_strategy=ratio_multiplier)
X_res, y_res = rus.fit_resample(X, y)

ax = y_res.value_counts().plot.pie(autopct=autopct)
_ = ax.set_title("Under-sampling")

display(X_res.shape, y_res.shape)
display(y_res.value_counts())
```

```
(15462, 6)
(15462, 1)
categoria
4            3753
0            3076
1            3076
2            2555
3            1745
5            1257
Name: count, dtype: int64
```



Esta parte del código se encarga de:

Preparar los datos: Dividiendo los datos en conjuntos de entrenamiento y prueba.

Configurar la búsqueda de hiperparámetros: Definiendo el espacio de búsqueda, el número

de pliegues de validación cruzada y las métricas a utilizar.

División de los Datos

- **train_test_split**: Esta función divide el conjunto de datos completo X (características) e y (etiquetas) en dos conjuntos: uno para entrenamiento (X_train, y_train) y otro para prueba (X_test, y_test).
- **test_size**=0.3: Indica que el 30% de los datos se reservará para el conjunto de prueba.
- **random_state**=42: Establece una semilla aleatoria para garantizar la reproducibilidad de los resultados.
- **y_train** = y_train.values.ravel(): Convierte las etiquetas y_train en un array unidimensional, lo cual es un requisito común para muchos algoritmos de clasificación.

In [92]:
```
# Suponiendo que X contiene las características y y la variable objetivo (ca
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, str

# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Funciones de apoyo para el preprocesamiento de los datos donde:

Función remove_outliers:

- Esta función toma un DataFrame como entrada y devuelve un nuevo DataFrame sin outliers.
- Utiliza el método IQR (Interquartile Range) para identificar outliers.
- Los valores fuera del rango [Q1 - 1.5 * IQR, Q3 + 1.5 * IQR] se consideran outliers y se eliminan.

Clase OutlierRemover:

- Esta clase es un transformer personalizado para usar con pipelines de scikit-learn.
- El método fit es un método dummy que simplemente devuelve el objeto mismo.
- El método transform aplica la función remove_outliers a los datos de entrada y devuelve un array NumPy.

Uso:

- La función remove_outliers se puede usar directamente en un DataFrame.
- La clase OutlierRemover se puede usar en un pipeline de scikit-learn para aplicar la eliminación de outliers como parte de una secuencia de transformaciones.

In [93]:
```
def remove_outliers(sub_class: pd.DataFrame) -> pd.DataFrame:
    Q1 = sub_class.quantile(0.15)
    Q3 = sub_class.quantile(0.85)
    IQR = Q3 - Q1
```

```
    # mean_iqr = (Q1 + Q3) / 2
    data_out = sub_class[
        ~((sub_class < (Q1 - 1.5 * IQR)) | (sub_class > (Q3 + 1.5 * IQR)))
    ]
    return data_out


class OutlierRemover(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return remove_outliers(pd.DataFrame(X)).values
```

Objetivo del modelo

El objetivo del modelo es clasificar el tipo de necesidad educativa especial (columna tipo_nee) que tiene un estudiante, en función de factores como el tipo de servicio (oficial o privado), la comuna, el grado y el total de estudiantes. Este tipo de análisis puede ser útil para ayudar a las instituciones educativas a anticipar las necesidades de apoyo especial en cada grado y localidad.

Modelos de Machine Learning elegidos

- **Logistic Regression:**
  - **Motivos**: Es un modelo interpretable y simple que puede ser un buen punto de partida para clasificar las necesidades educativas si las relaciones entre las características son lineales.
- **Support Vector Classifier (SVC)**:
  - **Motivos**: Es efectivo en datasets con muchas variables categóricas y puede ser ajustado para tratar problemas no lineales mediante el uso de kernels.
- **Decision Tree Classifier**:
  - **Motivos**: Permite captar interacciones entre características categóricas y numéricas, es fácil de interpretar y no requiere preprocesamiento intensivo.
- **Random Forest Classifier**:
  - **Motivos**: Es una extensión del árbol de decisión que mejora la robustez del modelo al combinar múltiples árboles. Ideal para detectar patrones complejos entre los factores y manejar mejor las clases desbalanceadas.
- **XGBoost**:
  - **Motivos**: Es un modelo más avanzado que mejora el rendimiento en la clasificación al corregir errores iterativamente. Ideal si las relaciones en los datos son altamente no lineales.

El código define un pipeline de aprendizaje automático en Scikit-learn para realizar tareas de clasificación. Un pipeline combina múltiples pasos de procesamiento de datos y modelado en una secuencia única, lo que simplifica el flujo de trabajo y mejora la reproducibilidad.

Componentes del Pipeline:

OutlierRemover:

- Elimina los valores atípicos (outliers) de los datos utilizando el método IQR (Interquartile Range).
- Ayuda a prevenir que los outliers influyan negativamente en el entrenamiento del modelo.

SimpleImputer:

- Imputa los valores faltantes en los datos con la estrategia especificada (en este caso, la media).
- Garantiza que los datos estén completos antes de ser procesados por el modelo.

MinMaxScaler:

- Estandariza los datos a un rango específico (0-1).
- Ayuda a mejorar la convergencia del modelo y evita que las características con escalas diferentes dominen el entrenamiento.

classifier:

- Un marcador de posición para el clasificador que se utilizará.
- Se reemplazará con diferentes modelos y sus hiperparámetros durante la búsqueda de cuadrícula.

Búsqueda de Parámetros:

- param_grid define una lista de diccionarios que contienen diferentes combinaciones de modelos y sus hiperparámetros.
- Se explorarán diferentes modelos (Regresión Logística, SVM, Árbol de Decisión, Bosque Aleatorio) con diferentes configuraciones de hiperparámetros.

Métricas de Evaluación:

- scoring define un diccionario de métricas que se utilizarán para evaluar el rendimiento del modelo.
- Se incluyen métricas como precisión, F1-score y recall, lo que permite una evaluación completa del modelo.

In [101…

```
# Definir los modelos y la búsqueda de parámetros
param_grid = [
    {
        "classifier": [LogisticRegression()],
        "classifier__penalty": ["l1", "l2"],
        "classifier__C": [0.1, 1, 10],
        "classifier__solver": [
            "liblinear"
```

```
        ],  # Se requiere 'liblinear' para el uso de L1 en Logistic Regressi
    },
    {
        "classifier": [SVC()],
        "classifier__kernel": ["linear", "rbf"],
        "classifier__C": [0.1, 1, 10],
    },
    {
        "classifier": [DecisionTreeClassifier()],
        "classifier__criterion": ["gini", "entropy"],
        "classifier__max_depth": [None, 10, 20],
    },
    {
        "classifier": [RandomForestClassifier()],
        "classifier__n_estimators": [50, 100, 200],
        "classifier__criterion": ["gini", "entropy"],
        "classifier__max_depth": [None, 10, 30],
    },
    {
        "classifier": [XGBClassifier()],
        "classifier__n_estimators": [50, 100, 200],
        "classifier__learning_rate": [0.01, 0.1, 0.2],
        "classifier__max_depth": [3, 6, 10],
        "classifier__subsample": [0.6, 0.8, 1.0],
        "classifier__colsample_bytree": [0.6, 0.8, 1.0],
    },
]

# Crear la Pipeline
pipeline = Pipeline(
    [
        ("outlier_remover", OutlierRemover()),  # Remoción de outliers
        ("imputer", SimpleImputer(strategy="mean")),  # Imputación de valore
        ("scaler", MinMaxScaler()),  # Escalado de características
        ("classifier", LogisticRegression()),  # Placeholder para el clasifi
    ]
)

# Definir múltiples métricas para evaluar
scoring = {
    "accuracy": "accuracy",
    "f1_score": make_scorer(
        f1_score, average="weighted"
    ),  # Promedio ponderado para clasificación multiclase
    "recall": make_scorer(recall_score, average="weighted"),
    # "roc_auc": "roc_auc_ovo",  # AUC para problemas multiclase (One-vs-One
}
```

Configuración de GridSearchCV

- **GridSearchCV**: Esta clase realiza una búsqueda exhaustiva de hiperparámetros en un espacio definido por el param_grid.
- **pipeline**: El pipeline creado anteriormente, que incluye preprocesamiento y el clasificador.

- **param_grid**: La lista de diccionarios con diferentes combinaciones de hiperparámetros para cada clasificador.
- **cv**=5: Se utiliza validación cruzada de 5 pliegues para evaluar el rendimiento de cada combinación de hiperparámetros.
- **n_jobs**=-1: Utiliza todos los núcleos de la CPU disponibles para acelerar la búsqueda.
- **verbose**=2: Muestra información detallada sobre el progreso de la búsqueda.
- **scoring**=scoring: Utiliza el diccionario de métricas definido anteriormente para evaluar el rendimiento del modelo.
- **refit**="accuracy": Después de la búsqueda, el mejor modelo se vuelve a entrenar utilizando todos los datos de entrenamiento y la combinación de hiperparámetros que obtuvo la mayor precisión.

Evaluacion del modelo

Para evaluar los modelos de clasificación, utilizaríamos las siguientes métricas:

```
- **Accuracy** :  Medir la proporción de predicciones
correctas.
- **Precision y Recall** : Estas métricas son útiles si hay
clases desbalanceadas (algunos tipos de NEE pueden ser menos
comunes).
- **F1-Score** : Combina precisión y recall en una sola
métrica, útil en datasets desbalanceados.
- **Matriz de confusión** : Para observar cuántas
predicciones se han clasificado correctamente por clase y
dónde están ocurriendo los errores.
```

In [ ]:
```
# Configurar GridSearchCV para explorar los hiperparámetros
grid_search = GridSearchCV(
    pipeline, param_grid, cv=5, n_jobs=-1, verbose=2, scoring=scoring, refit
)

# Entrenar la pipeline con la búsqueda de parámetros
grid_search.fit(X_train, y_train.values.ravel())

# Evaluar el mejor modelo en el conjunto de prueba
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X)
```

In [121…
```
print("Mejor modelo encontrado:", grid_search.best_params_)
print("Precisión en test:", best_model.score(X_test, y_test))
```

```
Mejor modelo encontrado: {'classifier': XGBClassifier(base_score=None, boost
er=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=Non
e,
              enable_categorical=False, eval_metric=None, feature_types=Non
e,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=Non
e,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...), 'classifier__
colsample_bytree': 0.8, 'classifier__learning_rate': 0.1, 'classifier__max_d
epth': 3, 'classifier__n_estimators': 200, 'classifier__subsample': 1.0}
Precisión en test: 0.5248507837340325
```

Visualizando los resultado en la matriz de confucion para los datos esta muestra una comparación entre los valores reales y los valores predichos por el modelo. Cada celda de la matriz representa una combinación de una clase real y una clase predicha.

Donde se evidencia que:

- El modelo ha clasificado el mejor las clases 1 y 3.
- Hay falsos positivos ni falsos negativos.
- Todas las muestras fueron asignadas a la clase correcta.

ademas de las metricas de evaluacion para el modelo se tiene que :

- **Accuracy (Precisión):** La proporción total de predicciones correctas. En este caso, el modelo clasificó de manera diversa todas las muestras, por lo que la precisión es 0.5.
- **F1-Score:** La media armónica de precisión y recall. Un valor de 0.5 indica un rendimiento variado.
- **Recall (Sensibilidad):** La proporción de muestras positivas que fueron correctamente identificadas como positivas. Al igual que la precisión, el recall es 0.5 en este caso, lo que significa precencia de falsos positivos y falsos negativos

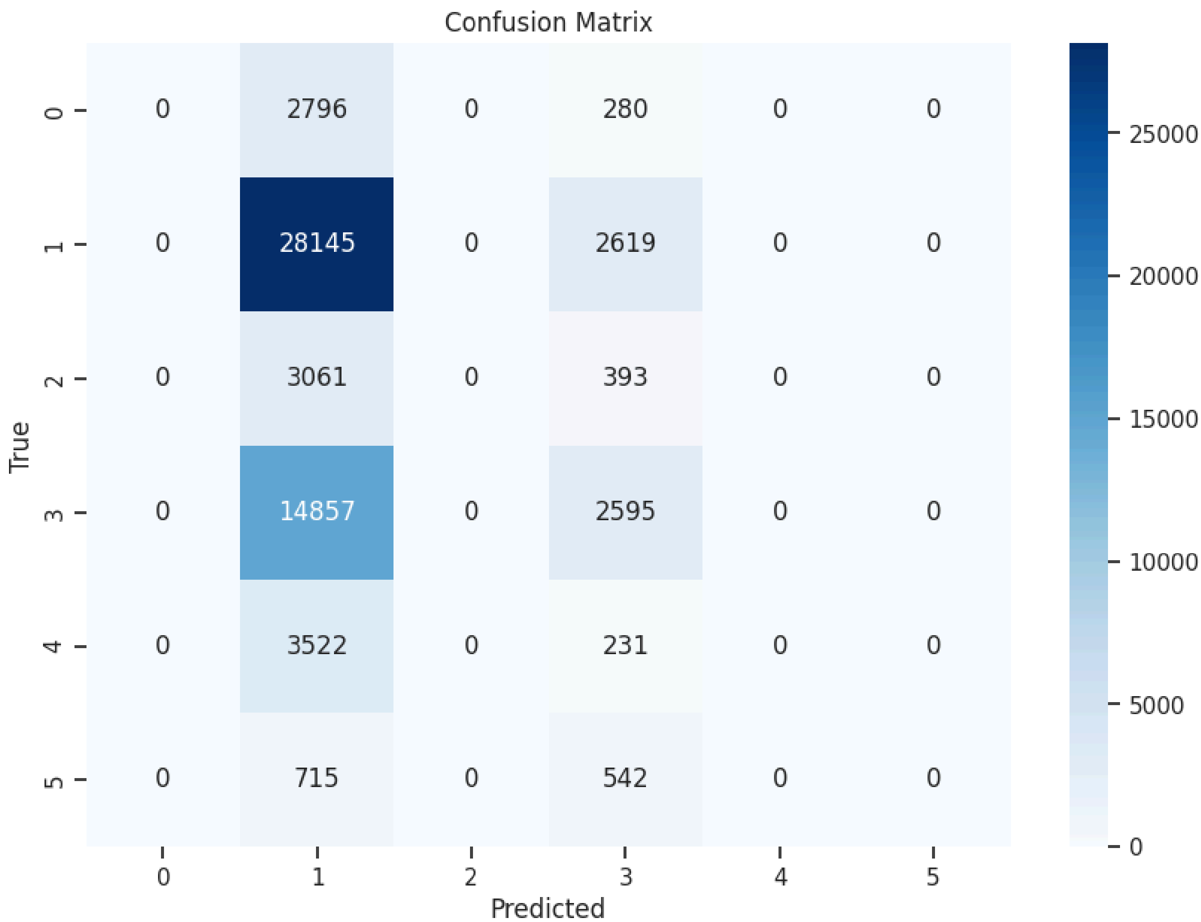
In [104…

```python
# Calcular la matriz de confusión
cm = confusion_matrix(y, y_pred)
class_report = classification_report(y, y_pred)

# Calcular las métricas
accuracy = accuracy_score(y, y_pred)
f1 = f1_score(y, y_pred, average="weighted")
recall = recall_score(y, y_pred, average="weighted")

print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
print(f"Recall: {recall}")
```

```
# Guardar la matriz de confusión como imagen
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
Accuracy: 0.5144253296740077
F1 Score: 0.40843469793896486
Recall: 0.5144253296740077
```

```
/home/bdebian/.virtualenvs/crystal/lib/python3.12/site-packages/sklearn/metr
ics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/bdebian/.virtualenvs/crystal/lib/python3.12/site-packages/sklearn/metr
ics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/bdebian/.virtualenvs/crystal/lib/python3.12/site-packages/sklearn/metr
ics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



Ademas se evaluarion las siguientes metricas de en los diferente tipos de modelo (Regresión Logística, SVM, Árbol de Decisión, RandomForestClassifier)

```
metrics = {
    "f1_score"
    "accuracy"
    "recall"
}
```

En cada metricas predominaba el XGBClassifier pero igualmente se hace un grafica comparativa de los resultados de los diferentes modelo y entrenamiento a lo largo de las 3 metricas para tipo de modelo propuesto, aunque el rendimiento que se observa es similiar predomina el XGBClassifier.

El modelo puestro numero 1, es decir el modelo con mas puntaje para cada metrica se repartio asi :

- **f1_score** : XGBClassifier = 0.445801 modelo con indice 80
- **accuracy**: XGBClassifier = 0.527385 modelo con indice 152
- **recall**: XGBClassifier = 0.527385 modelo con indice 152

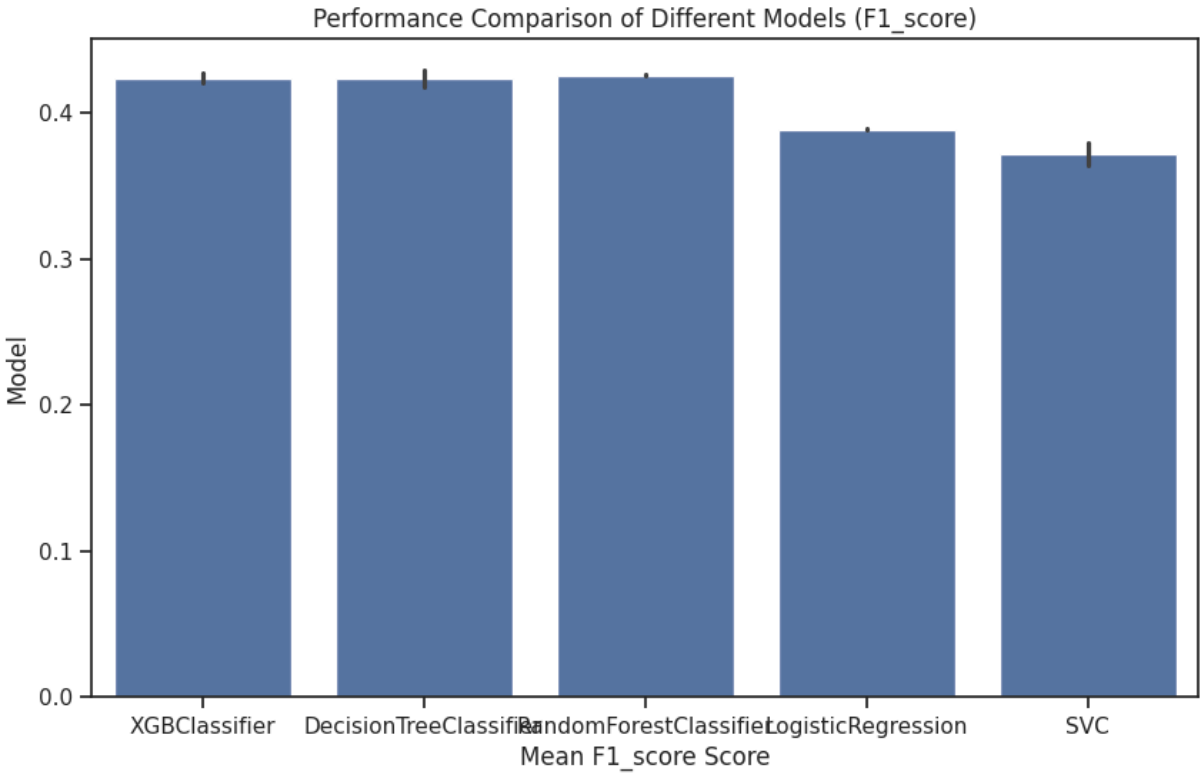
In [113…

```
results = pd.DataFrame(grid_search.cv_results_)

metrics = {
    "f1_score": "mean_test_f1_score",
    "accuracy": "mean_test_accuracy",
    "recall": "mean_test_recall",
}

# Ciclo para evaluar cada metricas seleccionada
for metric_name, metric_column in metrics.items():
    results_summary = results[
        [
            "param_classifier",
            metric_column,
            f"std_test_{metric_name}",
            f"rank_test_{metric_name}",
            "params",
        ]
    ]

    results_summary = results_summary.sort_values(by=f"rank_test_{metric_nam
    results_summary_temp = results_summary.head(20)
    display(results_summary_temp)

    plt.figure(figsize=(10, 6))
    sns.barplot(
        x=results_summary["param_classifier"]
        .astype(str)
        .apply(lambda x: x.split("(")[0]),
        y=results_summary[metric_column],
    )
    plt.xlabel(f"Mean {metric_name.capitalize()} Score")
```

```
    plt.ylabel("Model")
    plt.title(
        f"Performance Comparison of Different Models ({metric_name.capitaliz
    )
    plt.show()
```

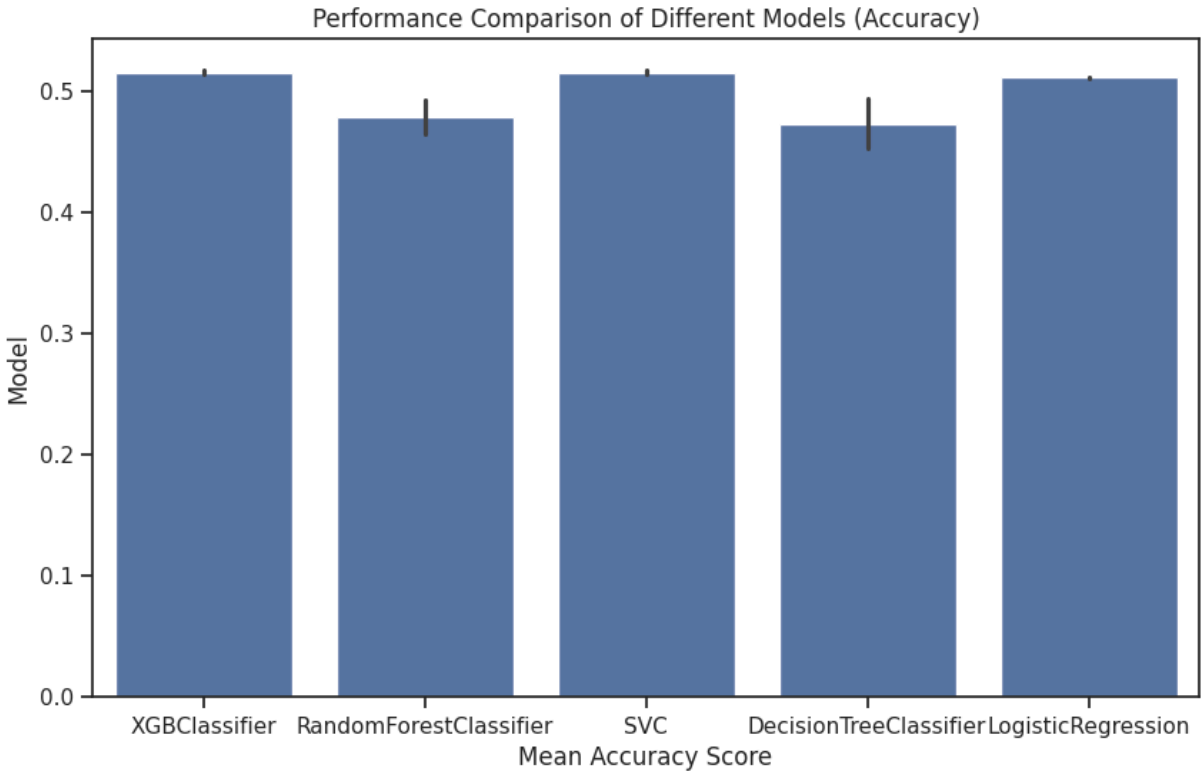
	param_classifier	mean_test_f1_score	std_test_f1_score	rank_test_f1_scor
80	XGBClassifier(base_score=None, booster=None, c...	0.445801	0.001567	
185	XGBClassifier(base_score=None, booster=None, c...	0.445508	0.002028	
107	XGBClassifier(base_score=None, booster=None, c...	0.445292	0.001403	
105	XGBClassifier(base_score=None, booster=None, c...	0.445230	0.000559	
188	XGBClassifier(base_score=None, booster=None, c...	0.445114	0.001444	
104	XGBClassifier(base_score=None, booster=None, c...	0.445002	0.000876	
242	XGBClassifier(base_score=None, booster=None, c...	0.444727	0.001218	
79	XGBClassifier(base_score=None, booster=None, c...	0.444631	0.001439	
102	XGBClassifier(base_score=None, booster=None, c...	0.444477	0.001882	
182	XGBClassifier(base_score=None, booster=None, c...	0.444311	0.001320	1
183	XGBClassifier(base_score=None, booster=None, c...	0.444253	0.001659	1
106	XGBClassifier(base_score=None, booster=None, c...	0.444190	0.000794	1
159	XGBClassifier(base_score=None, booster=None, c...	0.444163	0.001869	1
186	XGBClassifier(base_score=None, booster=None, c...	0.444151	0.002116	1
187	XGBClassifier(base_score=None, booster=None, c...	0.444146	0.001557	1

	param_classifier	mean_test_f1_score	std_test_f1_score	rank_test_f1_scor
266	XGBClassifier(base_score=None, booster=None, c...	0.444121	0.001391	1
78	XGBClassifier(base_score=None, booster=None, c...	0.443947	0.001302	1
103	XGBClassifier(base_score=None, booster=None, c...	0.443904	0.001614	1
184	XGBClassifier(base_score=None, booster=None, c...	0.443875	0.001697	1
240	XGBClassifier(base_score=None, booster=None, c...	0.443847	0.000840	2



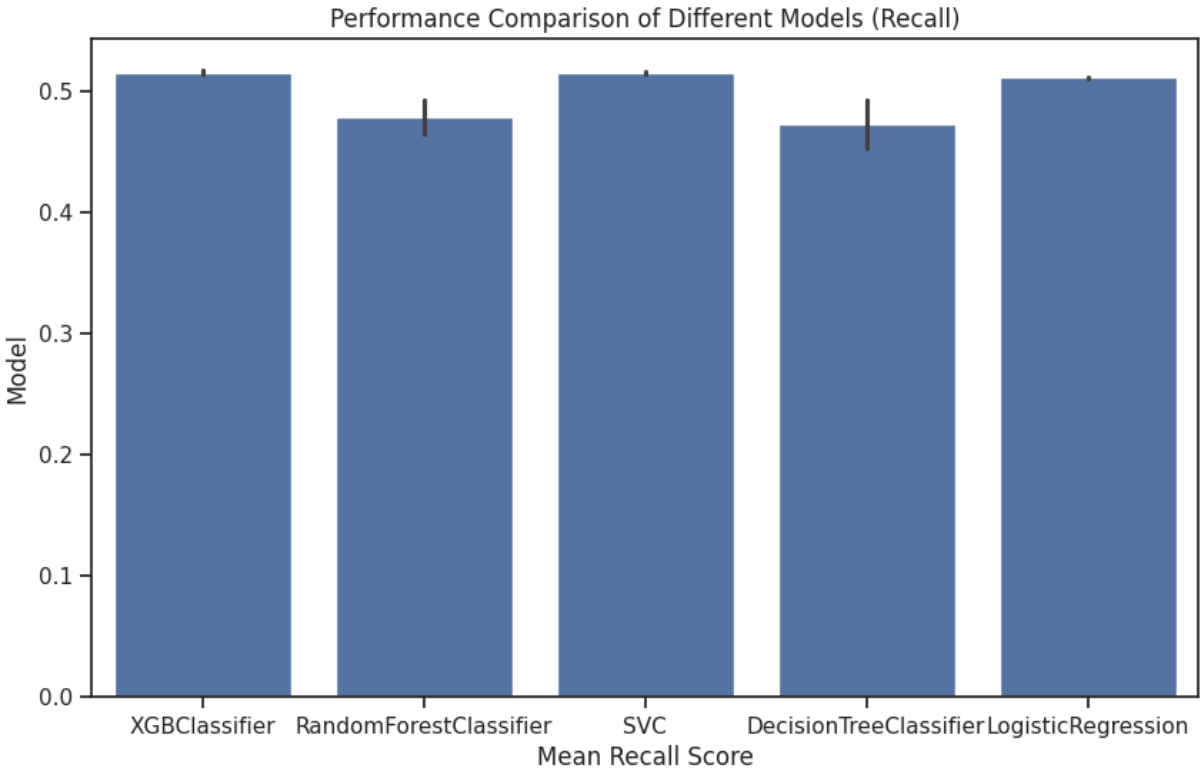
	param_classifier	mean_test_accuracy	std_test_accuracy	rank_test_accurac
152	XGBClassifier(base_score=None, booster=None, c...	0.527385	0.003987	
176	XGBClassifier(base_score=None, booster=None, c...	0.527027	0.003121	
208	XGBClassifier(base_score=None, booster=None, c...	0.526859	0.003621	
213	XGBClassifier(base_score=None, booster=None, c...	0.526740	0.003334	
207	XGBClassifier(base_score=None, booster=None, c...	0.526740	0.003005	
211	XGBClassifier(base_score=None, booster=None, c...	0.526692	0.004164	
151	XGBClassifier(base_score=None, booster=None, c...	0.526572	0.002925	
215	XGBClassifier(base_score=None, booster=None, c...	0.526477	0.003211	
233	XGBClassifier(base_score=None, booster=None, c...	0.526453	0.003088	
150	XGBClassifier(base_score=None, booster=None, c...	0.526405	0.002282	1
70	XGBClassifier(base_score=None, booster=None, c...	0.526333	0.003394	1
148	XGBClassifier(base_score=None, booster=None, c...	0.526309	0.002728	1
229	XGBClassifier(base_score=None, booster=None, c...	0.526309	0.002888	1
256	XGBClassifier(base_score=None, booster=None, c...	0.526214	0.002233	1
210	XGBClassifier(base_score=None, booster=None, c...	0.526214	0.003761	1

	param_classifier	mean_test_accuracy	std_test_accuracy	rank_test_accurac
95	XGBClassifier(base_score=None, booster=None, c...	0.526214	0.002432	1
147	XGBClassifier(base_score=None, booster=None, c...	0.526190	0.003055	1
228	XGBClassifier(base_score=None, booster=None, c...	0.526142	0.001926	1
236	XGBClassifier(base_score=None, booster=None, c...	0.526118	0.002430	1
214	XGBClassifier(base_score=None, booster=None, c...	0.526094	0.003110	2



	param_classifier	mean_test_recall	std_test_recall	rank_test_recall	
152	XGBClassifier(base_score=None, booster=None, c...	0.527385	0.003987	1	XGBC
176	XGBClassifier(base_score=None, booster=None, c...	0.527027	0.003121	2	XGBC
208	XGBClassifier(base_score=None, booster=None, c...	0.526859	0.003621	3	XGBC
213	XGBClassifier(base_score=None, booster=None, c...	0.526740	0.003334	4	XGBC
207	XGBClassifier(base_score=None, booster=None, c...	0.526740	0.003005	5	XGBC
211	XGBClassifier(base_score=None, booster=None, c...	0.526692	0.004164	6	XGBC
151	XGBClassifier(base_score=None, booster=None, c...	0.526572	0.002925	7	XGBC
215	XGBClassifier(base_score=None, booster=None, c...	0.526477	0.003211	8	XGBC
233	XGBClassifier(base_score=None, booster=None, c...	0.526453	0.003088	9	XGBC
150	XGBClassifier(base_score=None, booster=None, c...	0.526405	0.002282	10	XGBC
70	XGBClassifier(base_score=None, booster=None, c...	0.526333	0.003394	11	XGBC
148	XGBClassifier(base_score=None, booster=None, c...	0.526309	0.002728	12	XGBC
229	XGBClassifier(base_score=None, booster=None, c...	0.526309	0.002888	13	XGBC
256	XGBClassifier(base_score=None, booster=None, c...	0.526214	0.002233	14	XGBC
210	XGBClassifier(base_score=None, booster=None, c...	0.526214	0.003761	15	XGBC

	param_classifier	mean_test_recall	std_test_recall	rank_test_recall	
95	XGBClassifier(base_score=None, booster=None, c...	0.526214	0.002432	16	XGBC
147	XGBClassifier(base_score=None, booster=None, c...	0.526190	0.003055	17	XGBC
228	XGBClassifier(base_score=None, booster=None, c...	0.526142	0.001926	18	XGBC
236	XGBClassifier(base_score=None, booster=None, c...	0.526118	0.002430	19	XGBC
214	XGBClassifier(base_score=None, booster=None, c...	0.526094	0.003110	20	XGBC



Guardado de modelos y el encoder de los datos

In [127…

```
file_save_model = (
    Path(folder_dict["models"]).joinpath("mejor_modelo").with_suffix(".pkl")
)

# Guardar el modelo
with open(file_save_model, "wb") as file:
    pickle.dump(best_model, file)
```

```
file_save_encoder = (
    Path(folder_dict["models"]).joinpath("encoder").with_suffix(".pkl")
)

# Guardar el scaler
with open(file_save_encoder, "wb") as file:
    pickle.dump(label_encoder, file)
```

Conclusión del Modelo XGBClassifier

El modelo de clasificación **XGBClassifier** utilizado para predecir la categoría de necesidades educativas especiales (tipo_nee) en el dataset de educación formal en Colombia, arrojó los siguientes resultados:

- **F1-Score**: 0.4458 (modelo con índice 80)
- **Exactitud (Accuracy)**: 0.5274 (modelo con índice 152)
- **Recall**: 0.5274 (modelo con índice 152)

Estos resultados indican que el modelo presenta un rendimiento modesto, con una precisión ligeramente por encima del 50%. Aunque el **recall** y la **exactitud** se encuentran en el mismo nivel, el **f1-score** es más bajo, lo cual refleja que el modelo está teniendo dificultades para equilibrar adecuadamente la precisión y el recall en las clases minoritarias.

Distribución de Clases

La distribución inicial de las clases era altamente desequilibrada, con la categoría 1 dominando el conjunto de datos (30,764 entradas) frente a la categoría 5, que solo tenía 1,257 entradas. Este desequilibrio fue ajustado parcialmente mediante técnicas de balanceo, resultando en la siguiente distribución:

- Clase 4: 3,753 ejemplos
- Clase 0: 3,076 ejemplos
- Clase 1: 3,076 ejemplos
- Clase 2: 2,555 ejemplos
- Clase 3: 1,745 ejemplos
- Clase 5: 1,257 ejemplos

El balanceo realizado permitió acercar el número de ejemplos de las clases, lo que favoreció una mejora en el comportamiento del modelo en clases minoritarias, aunque no fue suficiente para obtener un rendimiento óptimo en todas las categorías.

Consideraciones Finales

Aunque el modelo muestra un rendimiento aceptable, es importante tener en cuenta que la exactitud del 52.74% significa que el modelo solo está prediciendo correctamente poco más

de la mitad de las instancias. El desequilibrio en la distribución original de las clases y la complejidad inherente de los datos, probablemente impactaron en el rendimiento general.

Para futuras iteraciones, sería recomendable explorar otras técnicas de balanceo, como el **oversampling** o **undersampling** más agresivos, así como ajustar los hiperparámetros del modelo y probar con modelos adicionales para mejorar la precisión en las clases minoritarias.

In [ ]: