```
!pip install sodapy
```

Haz doble clic (o ingresa) para editar

```
#!/usr/bin/env python

# make sure to install these packages before running:
# pip install pandas
# pip install sodapy

import pandas as pd
from sodapy import Socrata

# Unauthenticated client only works with public data sets. Note 'None'
# in place of application token, and no username or password:
client = Socrata("www.datos.gov.co", None)

# Example authenticated client (needed for non-public datasets):
# client = Socrata(www.datos.gov.co,
#                  MyAppToken,
#                  username="user@example.com",
#                  password="AFakePassword")

# First 2000 results, returned as JSON from API / converted to Python list of
# dictionaries by sodapy.
results = client.get("ch4u-f3i5", limit=100000)

# Convert to pandas DataFrame
results_df = pd.DataFrame.from_records(results)
```

```
results_df.shape
```

```
import pandas as pd

df = pd.read_csv("/kaggle/input/suelos/Resultados_de_Anlisis_de_Laboratorio_Suelos_en_Colombia_20251124.csv")
df.head()
```

```
df.shape
```

```
df.columns
```

```
df[df.isnull().any(axis=1)]
```

```
df.isnull().sum()
```

```
df.dtypes
```

```
df.info()
```

```
df.nunique()
```

```
for col in df.columns:
    print(f"\nColumna: {col}")
    print(df[col].unique())
```

```
for col in df.columns:
    print(f"\n=== {col} ===")
```

```
    print(df[col].value_counts())
```

```
Comienza a programar o generar con IA.
```

```
summary = pd.DataFrame({
    "grupos_totales": df.nunique(),
    "grupos": df.apply(lambda x: x.unique())
})

summary.to_csv("grupos_por_columna.csv", index=True)

summary
```

```
df[df["Zinc disponible doble \xa0acido"] == "ND"].shape
```

```
df[df["Zinc disponible doble \xa0acido"] == "ND"].shape
```

```
Comienza a programar o generar con IA.
```

```
print(df.columns.tolist())
```

Definir qué columnas son numéricas (propiedades químicas)

```
# Todas las columnas químicas numéricas (desde pH hasta el último metal)
numeric_cols = [
    'pH agua:suelo',
    'Materia organica',
    'Fósforo Bray II',
    'Azufre Fosfato monocalcico',
    'Acidez Intercambiable',
    'Aluminio intercambiable',
    'Calcio intercambiable',
    'Magnesio intercambiable',
    'Potasio intercambiable',
    'Sodio intercambiable',
    'capacidad de intercambio cationico',
    'Conductividad electrica',
    'Hierro disponible olsen',
    'Cobre disponible',
    'Manganeso disponible Olsen',
    'Zinc disponible Olsen',
    'Boro disponible',
    'Hierro disponible doble acido',
    'Cobre disponible doble acido',
    'Manganeso disponible doble acido',
    'Zinc disponible doble \xa0acido',
]
```

Función de limpieza de valores

Objetivo:

Quitar espacios.

Convertir comas decimales a punto.

Tratar expresiones tipo "<0.09" → usamos la mitad del límite (0.09/2).

Dejar como NaN las cadenas vacías o códigos raros.

```
```

```
def limpiar_valor(x):
    """
    Limpia un valor individual proveniente del CSV
    y lo convierte en float o NaN.
    """
    if pd.isna(x):                # Si ya es NaN -> lo devolvemos igual
        return np.nan

    x = str(x).strip()            # Pasar a string y quitar espacios

    # Valores que equivalen a dato faltante
    if x == '' or x.lower() in ['na', 'nd', 'nr', 's/r', 'nan']:
        return np.nan

    # Valores tipo "<0.09" (por debajo del límite de detección)
    if x.startswith('<'):
        try:
            limite = float(x[1:].replace(',', '.'))  # tomar la parte numérica
            return limite / 2                        # decisión: usar la mitad del límite
        except:
            return np.nan                            # si no podemos parsear, lo dejamos como NaN

    # Reemplazar coma decimal por punto
    x = x.replace(',', '.')

    # Intentar convertir a float
    try:
        return float(x)
    except:
        return np.nan
```

```
import numpy as np

# Aplicar la limpieza a cada columna numérica
for col in numeric_cols:
    df[col] = df[col].apply(limpiar_valor)

# Opcional: ver un resumen estadístico de estas columnas ya como numéricas
print(df[numeric_cols].describe().T)
```

Reglas de calidad: missing, rangos físicos y outliers (IQR)

Detección de outliers univariantes con IQR

Usaremos el criterio clásico:

IQR = Q3 − Q1

Límite inferior = Q1 − 1.5·IQR

Límite superior = Q3 + 1.5·IQR

Todo lo que queda fuera de ese rango se marca como atípico.

```
def detectar_outliers_iqr(serie):
    """
    Recibe una serie numérica (columna) y devuelve:
    - máscara booleana con True en los outliers
    - tupla con (lim_inf, lim_sup)
    """
    q1 = serie.quantile(0.25)              # Primer cuartil
    q3 = serie.quantile(0.75)              # Tercer cuartil
    iqr = q3 - q1                          # Rango intercuartílico
    lim_inf = q1 - 1.5 * iqr               # Límite inferior
    lim_sup = q3 + 1.5 * iqr               # Límite superior
    mask = (serie < lim_inf) | (serie > lim_sup)  # True donde hay outlier
    return mask, (lim_inf, lim_sup)
```

```
# Inicializamos un dataframe de banderas de outliers con False
outlier_df = pd.DataFrame(False, index=df.index, columns=numeric_cols)
```

```
# Recorremos cada columna numérica
for col in numeric_cols:
    serie = df[col]                                  # tomamos la columna
    mask, limites = detectar_outliers_iqr(serie.dropna())  # aplicamos IQR sobre valores no nulos
    mask = mask.reindex(df.index, fill_value=False) # reindexamos para cubrir todas las filas
    outlier_df[col] = mask                          # guardamos la máscara en outlier_df
```

```
Comienza a programar o generar con IA.
```

Reglas de rango físico

pH debe estar entre 0 y 14.

Para el resto de variables: valores negativos se consideran físicamente imposibles.

```
# Inicializamos otro dataframe de banderas para errores físicos
fisico_df = pd.DataFrame(False, index=df.index, columns=numeric_cols)

# Regla para pH: debe estar entre 0 y 14
fisico_df['pH agua:suelo'] = (
    (df['pH agua:suelo'] < 0) |
    (df['pH agua:suelo'] > 14)
)

# Regla para el resto: no se permiten valores negativos
for col in numeric_cols:
    if col != 'pH agua:suelo':
        fisico_df[col] = df[col] < 0
```

Datos faltantes (NaN)

```
# Banderas de datos faltantes por columna
missing_df = df[numeric_cols].isna()
missing_df
```

Construcción del Índice de Calidad de Datos (ICD)

Definamos que para cada celda (fila–columna) hay un error si:

el valor es faltante (missing), o

es outlier por IQR, o

viola una regla física.

Entonces el ICD por fila será:

ICD_fila = 1 - \frac{\text{# celdas con error en esa fila}}{\text{# columnas numéricas}}

Es decir, 1 = fila perfecta, 0 = todos los datos problemáticos.

```
# Unificamos las banderas: hay error si cualquiera de las tres condiciones se cumple
errores_df = missing_df | outlier_df | fisico_df

# Número total de variables numéricas consideradas
n_vars = len(numeric_cols)

# Cálculo del ICD por fila (registro)
df['ICD_fila'] = 1 - errores_df.sum(axis=1) / n_vars

# ICD global del dataset (promedio de todas las filas)
ICD_global = df['ICD_fila'].mean()

print("Índice de Calidad de Datos (ICD) global:", ICD_global)
```

Resumen de calidad por columna

Esto te sirve para el informe del reto: cuántos missing, outliers y errores físicos tiene cada variable.

```
# Conteo de errores por columna
resumen_columnas = pd.DataFrame({
    'missing': missing_df.sum(),          # cuántos NaN por columna
    'outliers_IQR': outlier_df.sum(),     # cuántos outliers por columna
    'errores_fisicos': fisico_df.sum()    # cuántos fuera de rango físico por columna
})

# Agregar total de errores y porcentaje sobre el total de filas
resumen_columnas['total_errores'] = resumen_columnas.sum(axis=1)
resumen_columnas['porc_filas_con_error_%'] = 100 * resumen_columnas['total_errores'] / len(df)

print(resumen_columnas.sort_values('total_errores', ascending=False))
```

```
import pandas as pd
import re


# patrón de textos típicos de faltantes
pat = re.compile(r'^(nd|n\.d\.|na|n/a|null|none|nan|no indica|no aplica|sin dato|s/d|\-|--|)$')

# contar tokens sospechosos en todo el dataframe
tokens = (
    df.astype(str)
      .apply(lambda c: c.str.strip().str.lower().str.replace(r"\s+", " ", regex=True))
)

conteo = tokens.stack().value_counts()

faltantes_texto = conteo[conteo.index.map(lambda x: bool(pat.match(x)))]

print("Tokens sospechosos de faltantes (con conteo):")
print(faltantes_texto)
```

```
import pandas as pd
import numpy as np
import re

# 1. Cargar archivo
path = "/kaggle/input/suelos/Resultados_de_Anlisis_de_Laboratorio_Suelos_en_Colombia_20251124.csv"
df = pd.read_csv(path, low_memory=False)

# 2. Lista de palabras que consideramos como datos faltantes
missing_tokens = [
    "nd", "n.d.", "n.d",
    "no indica", "no_indica", "no  indica",
    "no aplica",
    "sin dato", "s/d",
    "na", "n/a", "none", "null", "nan",
    "-", "--", ""
]

# 3. Normalizar el dataframe
df_normalized = df.apply(
    lambda col: col.astype(str)
                   .str.strip()
                   .str.lower()
                   .str.replace(r"\s+", " ", regex=True)
)
```

```
# 4. Reemplazar tokens por valores NaN
df_clean = df_normalized.replace(missing_tokens, np.nan)
```

```
# 5. Conteo de datos faltantes por columna
faltantes = df_clean.isnull().sum()
```

```
# 6. Porcentaje de datos faltantes por columna
faltantes_pct = df_clean.isnull().mean() * 100
```

```
faltantes, faltantes_pct
```

grupo iser

```
# Lista de columnas que deseas seleccionar
columnas_deseadas = ['Fecha de Análisis', 'Departamento', 'Municipio', 'pH agua:suelo',
                     'Cultivo','Fósforo Bray II', 'Potasio intercambiable', 'Materia organica']

# Filtrar columnas que realmente existen en df
columnas_validas = [c for c in columnas_deseadas if c in df.columns]

# Crear el nuevo DataFrame
df2 = df[columnas_validas]

df2
```

```
Comienza a programar o generar con IA.
```