Instalación y carga de datos desde la API

```
!pip install sodapy
```

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status
```

```
#!/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)
```

```
# Dimensiones de la tabla descargada
results_df.shape
```

```
import pandas as pd

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

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-2530573883.py in <cell line: 0>()
      1 import pandas as pd
      2
----> 3 df =
pd.read_csv("/kaggle/input/suelos/Resultados_de_Anlisis_de_Laboratorio_Suelos_en
      4 df.head()

                                 4 frames
/usr/local/lib/python3.12/dist-packages/pandas/io/common.py in
get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text,
errors, storage_options)
    871         if ioargs.encoding and "b" not in ioargs.mode:
    872             # Encoding
--> 873             handle = open(
    874                 handle,
    875                 ioargs.mode,
```

Próximos pasos: Explicar error

```
#Muestra cuántas filas y columnas tiene tu DataFrame df.
df.shape
```

```
#Lista todos los nombres de columnas del DataFrame
df.columns
```

```
# Filtra todas las filas que tienen al menos un valor nulo (NaN) en alguna colu
df[df.isnull().any(axis=1)]
```

```
# Cuenta cuántos valores nulos hay en cada columna.
df.isnull().sum()
```

```
#Muestra el tipo de dato de cada columna
df.dtypes
```

```
#Resume la estructura del DataFrame:

#número de filas
#número de columnas
#tipo de cada columna
#cuántos valores no nulos hay por columna.

df.info()
```

```
#Dice cuántos valores distintos tiene cada columna (sirve para ver si es categó
df.nunique()
```

```
#Recorre todas las columnas.

#Para cada una:
#imprime el nombre de la columna,
#imprime todos los valores únicos que hay en esa columna.
#Útil para revisar categorías o ver si hay valores raros como "ND", "NR", etc.

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

```
#Similar a la anterior, pero ahora imprime el conteo de cada valor distinto (fr

#Sirve para ver:
#qué valores son más frecuentes,
#si hay valores “basura” que aparecen pocas veces,
#distribución de categorías.

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

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

```
#Crea un DataFrame summary donde:

#"grupos_totales" = número de valores únicos por columna.
#"grupos" = el listado de valores únicos de cada columna.
#Guarda ese resumen en un CSV grupos_por_columna.csv.
#Muestra summary en pantalla.


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

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

summary
```

```
#Filtra filas donde la columna "Zinc disponible doble acido" (nota el espacio c

#.shape te dice cuántas filas tienen ese valor "ND".
#Esto sirve para ver cuántas entradas “no detectadas” hay.

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

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

```
break
```

```
#Imprime la lista de columnas como una lista de Python.
print(df.columns.tolist())
```

```
['Secuencial', 'Fecha de Análisis', 'Departamento', 'Municipio', 'Cultivo', 'Est
```

Definir qué columnas son numéricas (propiedades químicas)

```
# Todas las columnas químicas numéricas (desde pH hasta el último metal)

#Defino una lista llamada numeric_cols que incluye todas las columnas
#que necesito tratar como variables numéricas químicas.
#Esta lista es la base para aplicar los procesos de limpieza, detección de outl
#y demás transformaciones, asegurando que estos procedimientos se ejecuten únic
#sobre las columnas que efectivamente contienen información numérica.

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.

```
    #Esta función limpia textos, símbolos, límites de detección y errores del CSV,


    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
            except:
                return np.nan                            # si no podemos parsear, l

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

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

```
#Conversión a numérico y limpieza básica

#Recorre cada columna numérica.
#Intenta convertirla a número (float).
#Si encuentra algo no convertible (por ejemplo "ND"), lo convierte en NaN (errc
#Esto homogeneiza los tipos de datos para poder hacer estadística.

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)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-3725730721.py in <cell line: 0>()
     10 # Aplicar la limpieza a cada columna numérica
     11 for col in numeric_cols:
---> 12     df[col] = df[col].apply(limpiar_valor)
     13
     14 # Opcional: ver un resumen estadístico de estas columnas ya como
numéricas

NameError: name 'df' is not defined
```

Próximos pasos: Explicar error

```
# Función para detectar outliers con IQR

#Esta función implementa la detección de outliers por método IQR:
#Calcula Q1, Q3 e IQR = Q3 − Q1.
#Define límites: lim_inf y lim_sup.
#Crea una máscara booleana (True = outlier).

#Devuelve:

#mask: serie booleana,

#(lim_inf, lim_sup): los límites numéricos.

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)
```

```
#Creo un DataFrame llamado outlier_df, paralelo a df, que indica para cada fila
#para cada columna numérica si el valor corresponde a un outlier (True) o no (F

#Para cada columna numérica:

#Aplico la función detectar_outliers_iqr únicamente sobre los valores no nulos.
#Reindexo la máscara resultante para que coincida con todas las filas del DataF
#Al final, guardo esa máscara booleana en outlier_df bajo el nombre de la colum


# 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 sobr
    mask = mask.reindex(df.index, fill_value=False) # reindexamos para cubrir t
    outlier_df[col] = mask                          # guardamos la máscara en c
```

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
```

```
/usr/local/lib/python3.11/dist-packages/pandas/core/computation/expressions.py:7
  return op(a, b)
/usr/local/lib/python3.11/dist-packages/pandas/core/computation/expressions.py:7
  return op(a, b)
```

Datos faltantes (NaN)

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

	pH agua:suelo	Materia organica	Fósforo Bray II	Azufre Fosfato monocalcico	Acidez Intercambiable	Alumini intercambiabl
0	False	False	False	False	True	Tru
1	False	False	False	False	False	Fals
2	False	False	False	False	False	Fals
3	False	False	False	False	False	Fals
4	False	False	False	False	False	Fals
...	...	...	...	...	...	.
92733	False	False	False	False	False	Fals
92734	False	False	False	False	False	Fals
92735	False	False	False	False	False	Fals
92736	False	False	False	False	False	Fals
92737	False	False	False	False	False	Fals

92738 rows × 21 columns

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 c
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)
```

```
Índice de Calidad de Datos (ICD) global: 0.6979447475684185
```

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 c
})

# 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_errc

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

```
                                missing  outliers_IQR  errores_fisicos  \
Zinc disponible doble  acido      87952           550                0
Manganeso disponible doble acido  87952           418                0
Hierro disponible doble acido     87952           285                0
Cobre disponible doble acido      87952           132                0
Acidez Intercambiable             45130          2735                0
Aluminio intercambiable           45130          2677                0
Zinc disponible Olsen              5872          8630                0
Manganeso disponible Olsen         5872          6975                0
Hierro disponible olsen            5872          6462                0
Fósforo Bray II                    1469          9706                0
Sodio intercambiable                  0         10912                1
Azufre Fosfato monocalcico         1499          8858                1
Cobre disponible                   5873          4415                0
Conductividad electrica            1028          8298                0
Materia organica                      4          9282                0
Magnesio intercambiable               0          8757                0
Potasio intercambiable                0          7460                0
Boro disponible                    1312          5699                0
capacidad de intercambio cationico   67          6679                0
Calcio intercambiable                 0          6017                0
pH agua:suelo                         5          2362                0

                                total_errores  porc_filas_con_error_%
Zinc disponible doble  acido            88502               95.432293
Manganeso disponible doble acido        88370               95.289957
Hierro disponible doble acido           88237               95.146542
Cobre disponible doble acido            88084               94.981561
Acidez Intercambiable                   47865               51.613147
Aluminio intercambiable                 47807               51.550605
Zinc disponible Olsen                   14502               15.637603
Manganeso disponible Olsen              12847               13.853005
Hierro disponible olsen                 12334               13.299834
Fósforo Bray II                         11175               12.050077
Sodio intercambiable                    10913               11.767560
Azufre Fosfato monocalcico              10358               11.169100
Cobre disponible                        10288               11.093619
Conductividad electrica                  9326               10.056288
Materia organica                         9286               10.013155
Magnesio intercambiable                  8757                9.442731
Potasio intercambiable                   7460                8.044167
```

```
Boro disponible                          7011                7.560008
capacidad de intercambio cationico       6746                7.274257
Calcio intercambiable                    6017                6.488171
pH agua:suelo                            2367                2.552352
```

```
# Lista de columnas que deseas seleccionar
columnas_deseadas = ['Fecha de Análisis', 'Departamento', 'Municipio', 'pH agua:
'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
```