## Instalación y carga de datos desde la API

```
!pip install sodapy
```

```
Requirement already satisfied: sodapy in /usr/local/lib/python3.12/dist-packages (2.2.0)
Requirement already satisfied: requests>=2.28.1 in /usr/local/lib/python3.12/dist-packages (from sodapy) (2.32.4)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.28.1->sodap
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.28.1->sodapy) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.28.1->sodapy) (2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.28.1->sodapy) (20
```

```
#!/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)
```

```
WARNING:root:Requests made without an app_token will be subject to strict throttling limits.
```

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

```
(92738, 32)
```

```
import pandas as pd

df = pd.read_csv("/content/Resultados_de_Análisis_de_Laboratorio_Suelos_en_Colombia_20251124.csv", sep=';', on_bad_lines='skip'
df.head()
```

| | Secuencial | Fecha de Análisis | Departamento | Municipio | Cultivo | Estado | Tiempo de establecimiento | Topografia | Drenaje | Riego | ... | Hie disponi ol |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 7/1/2014 | NARIÑO | SAN ANDRÉS DE TUMACO | No Indica | No indica | No indica | No indica | No indica | No indica | ... | 66 |
| **1** | 2 | 21/8/2014 | HUILA | SANTA MARÍA | Granadilla | Establecido | De 1 a 5 años | Pendiente | Buen drenaje | No Tiene | ... | 25 |
| **2** | 3 | 22/8/2014 | ANTIOQUIA | LIBORINA | Café | Por establecer | No indica | Pendiente | Mal drenaje | No Tiene | ... | |
| **3** | 4 | 22/8/2014 | ANTIOQUIA | LIBORINA | Maracuyá | Por establecer | No indica | Ondulado | Mal drenaje | No Tiene | ... | |
| **4** | 5 | 22/8/2014 | ANTIOQUIA | LIBORINA | Café | Por establecer | No indica | Ondulado | Mal drenaje | No Tiene | ... | 11 |

5 rows × 33 columns

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

```
(92772, 33)
```

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

```
Index(['Secuencial', 'Fecha de Análisis', 'Departamento', 'Municipio',
       'Cultivo', 'Estado', 'Tiempo de establecimiento', 'Topografia',
       'Drenaje', 'Riego', 'Fertilizantes aplicados', '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  acido', 'Unnamed: 32'],
      dtype='object')
```

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

| | Secuencial | Fecha de Análisis | Departamento | Municipio | Cultivo | Estado | Tiempo de establecimiento | Topografia | Drenaje | Riego |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 7/1/2014 | NARIÑO | SAN ANDRÉS DE TUMACO | No Indica | No indica | No indica | No indica | No indica | No indica |
| **1** | 2 | 21/8/2014 | HUILA | SANTA MARÍA | Granadilla | Establecido | De 1 a 5 años | Pendiente | Buen drenaje | No Tiene |
| **2** | 3 | 22/8/2014 | ANTIOQUIA | LIBORINA | Café | Por establecer | No indica | Pendiente | Mal drenaje | No Tiene |
| **3** | 4 | 22/8/2014 | ANTIOQUIA | LIBORINA | Maracuyá | Por establecer | No indica | Ondulado | Mal drenaje | No Tiene |
| **4** | 5 | 22/8/2014 | ANTIOQUIA | LIBORINA | Café | Por establecer | No indica | Ondulado | Mal drenaje | No Tiene |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **64651** | completo + Nitrogeno Total" | 4.85 | 8.43 | 21.52 | 32.99 | 1 | 0.87 | 1.1 | 0.21 | 0.13 |
| **71239** | 71 | 206 | 24/8/2022 | ANTIOQUIA | HELICONIA | Tomate | ESTABLECIDO | De 0 a 1 año | Plano | Buen drenaje |
| **73832** | 73 | 799 | 11/3/2022 | ANTIOQUIA | TURBO | Limón | ESTABLECIDO | de 1 a 5 años | Plano | No indica |
| **73833** | 73 | 800 | 11/3/2022 | ANTIOQUIA | TURBO | Limón | ESTABLECIDO | de 1 a 5 años | Pendiente moderada | No indica |
| **89343** | 89 | 310 | 27/5/2024 | NORTE DE SANTANDER | GRAMALOTE | Café | ESTABLECIDO | De 0 a 1 año | Ondulado y Pendiente | Buen drenaje |

1385 rows × 33 columns

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

| | 0 |
|---|---|
| **Secuencial** | 0 |
| **Fecha de Análisis** | 6 |
| **Departamento** | 13 |
| **Municipio** | 13 |
| **Cultivo** | 15 |
| **Estado** | 13 |
| **Tiempo de establecimiento** | 13 |
| **Topografia** | 13 |
| **Drenaje** | 13 |
| **Riego** | 13 |
| **Fertilizantes aplicados** | 13 |
| **pH agua:suelo** | 13 |
| **Materia organica** | 362 |
| **Fósforo Bray II** | 362 |
| **Azufre Fosfato monocalcico** | 362 |
| **Acidez Intercambiable** | 362 |
| **Aluminio intercambiable** | 362 |
| **Calcio intercambiable** | 362 |
| **Magnesio intercambiable** | 362 |
| **Potasio intercambiable** | 362 |
| **Sodio intercambiable** | 362 |
| **capacidad de intercambio cationico** | 362 |
| **Conductividad electrica** | 373 |
| **Hierro disponible olsen** | 381 |
| **Cobre disponible** | 381 |
| **Manganeso disponible Olsen** | 382 |
| **Zinc disponible Olsen** | 383 |
| **Boro disponible** | 383 |
| **Hierro disponible doble acido** | 383 |
| **Cobre disponible doble acido** | 384 |
| **Manganeso disponible doble acido** | 383 |
| **Zinc disponible doble  acido** | 383 |
| **Unnamed: 32** | 1382 |

**dtype:** int64

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

| | 0 |
|---|---|
| **Secuencial** | object |
| **Fecha de Análisis** | object |
| **Departamento** | object |
| **Municipio** | object |
| **Cultivo** | object |
| **Estado** | object |
| **Tiempo de establecimiento** | object |
| **Topografia** | object |
| **Drenaje** | object |
| **Riego** | object |
| **Fertilizantes aplicados** | object |
| **pH agua:suelo** | object |
| **Materia organica** | object |
| **Fósforo Bray II** | object |
| **Azufre Fosfato monocalcico** | object |
| **Acidez Intercambiable** | object |
| **Aluminio intercambiable** | object |
| **Calcio intercambiable** | object |
| **Magnesio intercambiable** | object |
| **Potasio intercambiable** | object |
| **Sodio intercambiable** | object |
| **capacidad de intercambio cationico** | object |
| **Conductividad electrica** | object |
| **Hierro disponible olsen** | object |
| **Cobre disponible** | object |
| **Manganeso disponible Olsen** | object |
| **Zinc disponible Olsen** | object |
| **Boro disponible** | object |
| **Hierro disponible doble acido** | object |
| **Cobre disponible doble acido** | object |
| **Manganeso disponible doble acido** | object |
| **Zinc disponible doble  acido** | object |
| **Unnamed: 32** | object |

**dtype:** object

```
#Resume la estructura del DataFrame:

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

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 92772 entries, 0 to 92771
Data columns (total 33 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Secuencial                        92772 non-null  object
 1   Fecha de Análisis                 92766 non-null  object
 2   Departamento                      92759 non-null  object
 3   Municipio                         92759 non-null  object
```

```
 4   Cultivo                            92757 non-null  object
 5   Estado                             92759 non-null  object
 6   Tiempo de establecimiento          92759 non-null  object
 7   Topografia                         92759 non-null  object
 8   Drenaje                            92759 non-null  object
 9   Riego                              92759 non-null  object
 10  Fertilizantes aplicados            92759 non-null  object
 11  pH agua:suelo                      92759 non-null  object
 12  Materia organica                   92410 non-null  object
 13  Fósforo Bray II                    92410 non-null  object
 14  Azufre Fosfato monocalcico         92410 non-null  object
 15  Acidez Intercambiable              92410 non-null  object
 16  Aluminio intercambiable            92410 non-null  object
 17  Calcio intercambiable              92410 non-null  object
 18  Magnesio intercambiable            92410 non-null  object
 19  Potasio intercambiable             92410 non-null  object
 20  Sodio intercambiable               92410 non-null  object
 21  capacidad de intercambio cationico 92410 non-null  object
 22  Conductividad electrica            92399 non-null  object
 23  Hierro disponible olsen            92391 non-null  object
 24  Cobre disponible                   92391 non-null  object
 25  Manganeso disponible Olsen         92390 non-null  object
 26  Zinc disponible Olsen              92389 non-null  object
 27  Boro disponible                    92389 non-null  object
 28  Hierro disponible doble acido      92389 non-null  object
 29  Cobre disponible doble acido       92388 non-null  object
 30  Manganeso disponible doble acido   92389 non-null  object
 31  Zinc disponible doble  acido       92389 non-null  object
 32  Unnamed: 32                        91390 non-null  object
dtypes: object(33)
memory usage: 23.4+ MB
```

```
#Dice cuántos valores distintos tiene cada columna (sirve para ver si es categórica, numérica con pocas categorías, etc.).
df.nunique()
```

| | 0 |
|---|---|
| **Secuencial** | 1018 |
| **Fecha de Análisis** | 1054 |
| **Departamento** | 2401 |
| **Municipio** | 176 |
| **Cultivo** | 1086 |
| **Estado** | 303 |
| **Tiempo de establecimiento** | 27 |
| **Topografia** | 34 |
| **Drenaje** | 37 |
| **Riego** | 32 |
| **Fertilizantes aplicados** | 131 |
| **pH agua:suelo** | 5704 |
| **Materia organica** | 1230 |
| **Fósforo Bray II** | 10501 |
| **Azufre Fosfato monocalcico** | 18090 |
| **Acidez Intercambiable** | 12927 |
| **Aluminio intercambiable** | 6408 |
| **Calcio intercambiable** | 5803 |
| **Magnesio intercambiable** | 12554 |
| **Potasio intercambiable** | 7540 |
| **Sodio intercambiable** | 2512 |
| **capacidad de intercambio cationico** | 3142 |
| **Conductividad electrica** | 13548 |
| **Hierro disponible olsen** | 3841 |
| **Cobre disponible** | 33305 |
| **Manganeso disponible Olsen** | 6513 |
| **Zinc disponible Olsen** | 10597 |
| **Boro disponible** | 8265 |
| **Hierro disponible doble acido** | 1876 |
| **Cobre disponible doble acido** | 3168 |
| **Manganeso disponible doble acido** | 1036 |
| **Zinc disponible doble acido** | 2501 |
| **Unnamed: 32** | 1495 |

**dtype:** int64

```
#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())
```

```
Columna: Secuencial
['1' '2' '3' ... 'UTILIZA RENOVADOR' 'PATENKALI"'
 'completo + Nitrogeno Total"']
```

```
Columna: Fecha de Análisis
['7/1/2014' '21/8/2014' '22/8/2014' ... '5.36' '5.17' '4.85']

Columna: Departamento
['NARIÑO' 'HUILA' 'ANTIOQUIA' ... '31/3/2025' '3/4/2025' '4/4/2025']

Columna: Municipio
['SAN ANDRÉS DE TUMACO' 'SANTA MARÍA' 'LIBORINA' 'SILVANIA' 'PASCA'
 'VILLETA' 'AGUSTÍN CODAZZI' 'SAN BERNARDO' 'CARACOLÍ' 'EL ROSAL'
 'PUERTO LÓPEZ' 'SAN JUAN DE ARAMA' 'GUASCA' 'FALAN' 'DAGUA' 'NIMAIMA'
 'VENTAQUEMADA' 'GRANADA' 'CIÉNEGA' 'NEMOCÓN' 'SAN MARTÍN' 'POTOSÍ' 'ILES'
 'IPIALES' 'RIONEGRO' 'CIMITARRA' 'SIBATÉ' 'VILLAVICENCIO' 'VISTAHERMOSA'
 'NOCAIMA' 'UNE' 'FUNZA' 'LA UNIÓN' 'PALMIRA' 'CALI' 'ACACÍAS'
 'ZONA BANANERA' 'ESPINAL' 'FLORIDABLANCA' 'CERETÉ' 'EL CARMEN DE BOLÍVAR'
 'ALBÁN' 'CHIQUINQUIRÁ' 'PUERTO GAITÁN' 'VENADILLO' 'PEREIRA' 'GÜEPSA'
 'SAN PEDRO DE LOS MILAGROS' 'GUAYABAL DE SÍQUIMA' 'APULO' 'CÁQUEZA'
 'EL CERRITO' 'GINEBRA' 'SAN LORENZO' 'ARCABUCO' 'FUSAGASUGÁ' 'MOSQUERA'
 'TULUÁ' 'TRUJILLO' 'CANDELARIA' 'MIRANDA' 'SANTANDER DE QUILICHAO'
 'CAJAMARCA' 'LA CALERA' 'JESÚS MARÍA' 'SAN PELAYO' 'ALVARADO' 'LA VEGA'
 'EL CAIRO' 'VILLARRICA' 'PASTO' 'CALAMAR' 'CHÍA' 'Pandi' 'BUGALAGRANDE'
 'SAN JUAN DE RIOSECO' 'JAMUNDÍ' 'VILLA DE LEYVA' 'FLORIDA' 'DISTRACCIÓN'
 'OLAYA HERRERA' 'SAN CARLOS DE GUAROA' 'GUADUAS' 'BUESACO' 'LA MESA'
 'SAN FRANCISCO' 'AGUACHICA' 'CAMPOALEGRE' 'VILLAVIEJA' 'BARAYA'
 'EL SANTUARIO' 'TELLO' 'RIVERA' 'HOBO' 'QUÍPAMA' 'YUMBO' 'GUACARÍ'
 'GUATAVITA' 'TRINIDAD' 'PORE' 'PISBA' 'SUTAMARCHÁN' 'FONSECA' 'QUIPILE'
 'BARRANCAS' 'GACHANTIVÁ' 'CHAGUANÍ' 'ÚMBITA' 'BOGOTÁ,  D.C.' 'CHAPARRAL'
 'ZIPAQUIRÁ' 'PUERTO LLERAS' 'GUAMO' 'ARACATACA' 'CIÉNAGA' 'ORTEGA'
 'ATACO' 'NATAGAIMA' 'BITUIMA' 'TURMEQUÉ' 'CAJIBÍO' 'SAN LUIS' 'CÓRDOBA'
 'BOYACÁ' 'SANTANDER' 'ANTIOQUIA' 'CUNDINAMARCA' 'CESAR' 'NARIÑO'
 'VALLE DEL CAUCA' 'MAGDALENA' 'HUILA' 'ATLÁNTICO' 'BOLÍVAR' 'CASANARE'
 'TOLIMA' 'BOGOTÁ, D.C.' 'RISARALDA' 'CAUCA' 'GUAVIARE' 'META'
 'LA GUAJIRA' 'NORTE DE SANTANDER' 'CALDAS' 'VICHADA' 'CHOCÓ' 'PUTUMAYO'
 'QUINDÍO' 'SUCRE' 'ARAUCA' 'CAQUETÁ' 'AMAZONAS' nan '4.539' '4.981'
 '4.506' '4.682' '11.84' '29.8' ' CADA PASTOREO' 'GUAINÍA' '29.76' '9.492'
 '198.4' '211.7' '18.95' '13.57' 'VAUPÉS' ' CADA 180 DÍAS"' '8.25' '13.87'
 '7.15' '16.99' '9.28' '13.61' '21.52'
 'ARCHIPIÉLAGO DE SAN ANDRÉS, PROVIDENCIA Y SANTA CATALINA']

Columna: Cultivo
['No Indica' 'Granadilla' 'Café' ... 'MIRITÍ - PARANÁ' 'CHALÁN'
 'LOS PALMITOS']

Columna: Estado
['No indica' 'Establecido' 'Por establecer' 'Tabaco' 'Pastos' 'Caucho'
 'Brócoli' 'Mora' 'Caña Panelera' 'Batata' 'No Indica' 'Palma de Aceite'
 'Gulupa' 'Cebolla de Rama' 'Melón' 'Papa Criolla' 'Cacao' 'Balú' 'Ajo'
 'Pino' 'Eucalipto' 'Cebada' 'Papa de año' 'Maracuyá' 'Uva' 'Café'
 'Lechuga' 'Flores' 'Aguacate' 'Acelga' 'Arracacha' 'Forestales' 'Lulo'
 'Pasto Estrella' 'Chía' 'Higuerilla' 'Plátano' 'Pasto Cuba 22' 'Maíz'
 'Frijol' 'Fresa' 'Cebolla Puerro' 'Papaya' 'Silvopastoril' 'Cítricos'
 'Piña' 'Mango' 'Pasto Kikuyo' 'Pasto Ryegrass' 'Uchuva' 'Pasto Angleton'
 'Pasto Mombaza' 'Tomate' 'Limón' 'Yuca' 'Ruscus' 'Helecho' 'Sábila'
```

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

#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())
```

```
4.98     537
5.04     527
4.96     521
4.94     520
5.08     520
        ...
9.37       1
9.89       1
12.26      1
11.1       1
12.9       1
Name: count, Length: 1230, dtype: int64

=== Fósforo Bray II ===
Fósforo Bray II
1.9      257
1.62     255
1.5      250
1.84     246
1.66     243
        ...
64.06      1
107.3      1
442        1
66.48      1
74.66      1
Name: count, Length: 10501, dtype: int64

=== Azufre Fosfato monocalcico ===
Azufre Fosfato monocalcico
<3.87     11328
<1.29      2097
ND         1469
<1.42       101
1.5          51
          ...
184.46        1
172.07        1
76.08         1
88.7          1
```

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
```

| | grupos_totales | grupos |
|---|---|---|
| **Secuencial** | 1018 | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... |
| **Fecha de Análisis** | 1054 | [7/1/2014, 21/8/2014, 22/8/2014, 26/8/2014, 15... |
| **Departamento** | 2401 | [NARIÑO, HUILA, ANTIOQUIA, CUNDINAMARCA, CESAR... |
| **Municipio** | 176 | [SAN ANDRÉS DE TUMACO, SANTA MARÍA, LIBORINA, ... |
| **Cultivo** | 1086 | [No Indica, Granadilla, Café, Maracuyá, Mora, ... |
| **Estado** | 303 | [No indica, Establecido, Por establecer, Tabac... |
| **Tiempo de establecimiento** | 27 | [No indica, De 1 a 5 años, De 5 a 10 años, De ... |
| **Topografia** | 34 | [No indica, Pendiente, Ondulado, Plano, Ligera... |
| **Drenaje** | 37 | [No indica, Buen drenaje, Mal drenaje, Regular... |
| **Riego** | 32 | [No indica, No Tiene, Goteo, Aspersión, Mangue... |
| **Fertilizantes aplicados** | 131 | [No indica, PRODUCCION, NITRON-10-20-20-CRECER... |
| **pH agua:suelo** | 5704 | [5.64, 4.24, 5.22, 5.45, 5.43, 6.92, 4.59, 4.8... |
| **Materia organica** | 1230 | [3.889, 2.287, 2.518, 2.841, 2.703, 2.564, 2.7... |
| **Fósforo Bray II** | 10501 | [6.482, 520.1, 58.11, 4.583, 5.512, 100.8, 11.... |
| **Azufre Fosfato monocalcico** | 18090 | [8.384, 90.3, 2.686, 1.782, 3.138, 4.947, 8.71... |
| **Acidez Intercambiable** | 12927 | [ND, 2.021, 0.955, 0, 0.15, 3.48, 2.052, 0.925... |
| **Aluminio intercambiable** | 6408 | [ND, 1.483, 0.796, 0, 0.109, 2.897, 1.862, 0.6... |
| **Calcio intercambiable** | 5803 | [1.541, 2.932, 3.818, 4.754, 7.338, 9.023, 4.5... |
| **Magnesio intercambiable** | 12554 | [0.38, 0.925, 1.11, 3.577, 3.022, 2.96, 1.727,... |
| **Potasio intercambiable** | 7540 | [0.118, 2.131, <0.09, 0.096, 0.243, 0.272, 0.1... |
| **Sodio intercambiable** | 2512 | [0.054, 0.043, 0.01, 0.032, 0.038, 0.005, 0.02... |
| **capacidad de intercambio cationico** | 3142 | [2.094, 8.054, 5.985, 8.461, 10.64, 12.26, 9.8... |
| **Conductividad electrica** | 13548 | [0.133, 2.749, 0.328, 0.171, 0.323, 0.621, 0.1... |

```
#Filtra filas donde la columna "Zinc disponible doble acido" (nota el espacio o carácter extraño \xa0) es igual a "ND".

#.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
```

```
(87603, 33)
```

| | | |
|---|---|---|
| **Cobre disponible doble acido** | 3168 | [ND, 0.88, 1.68, 1.44, 1.24, 1.159, 2.84, 1.08... |

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

| | | |
|---|---|---|
| **Unnamed: 32** | 1495 | [nan, ND, 0,48, 0,24, 0,4, 0,44, 0,12, 0,2, 0,... |

Próximos pasos: Generar código con summary | New interactive sheet

| | Secuencial | Fecha de Análisis | Departamento | Municipio | Cultivo | Estado | Tiempo de establecimiento | Topografia | Drenaje | Riego |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 7/1/2014 | NARIÑO | SAN ANDRÉS DE TUMACO | No Indica | No indica | No indica | No indica | No indica | No indica |
| **1** | 2 | 21/8/2014 | HUILA | SANTA MARÍA | Granadilla | Establecido | De 1 a 5 años | Pendiente | Buen drenaje | No Tiene |
| **2** | 3 | 22/8/2014 | ANTIOQUIA | LIBORINA | Café | Por establecer | No indica | Pendiente | Mal drenaje | No Tiene |
| **3** | 4 | 22/8/2014 | ANTIOQUIA | LIBORINA | Maracuyá | Por establecer | No indica | Ondulado | Mal drenaje | No Tiene |
| **4** | 5 | 22/8/2014 | ANTIOQUIA | LIBORINA | Café | Por establecer | No indica | Ondulado | Mal drenaje | No Tiene |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **92767** | 92 | 734 | 7/4/2025 | SANTANDER | SOCORRO | Cacao | ESTABLECIDO | De 0 a 1 año | Ligeramente ondulado | Regular drenaje |
| **92768** | 92 | 735 | 4/4/2025 | SANTANDER | PALMAS DEL SOCORRO | Cacao | ESTABLECIDO | de 5 a 10 años | Ligeramente ondulado | Buen drenaje |
| **92769** | 92 | 736 | 3/4/2025 | SANTANDER | PALMAS DEL SOCORRO | Cacao | ESTABLECIDO | mas de 10 años | Plano | Regular drenaje |
| **92770** | 92 | 737 | 4/4/2025 | SANTANDER | PALMAS DEL SOCORRO | Cacao | ESTABLECIDO | de 1 a 5 años | Pendiente moderada | Buen drenaje |
| **92771** | 92 | 738 | 4/4/2025 | SANTANDER | PALMAS DEL SOCORRO | Cacao | ESTABLECIDO | de 5 a 10 años | Plano | Buen drenaje |

87603 rows × 33 columns

Comienza a programar o generar con IA.

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

```
['Secuencial', 'Fecha de Análisis', 'Departamento', 'Municipio', 'Cultivo', 'Estado', 'Tiempo de establecimiento', 'Topografia',
```

## 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 outliers
#y demás transformaciones, asegurando que estos procedimientos se ejecuten únicamente
#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, y devuelve siempre un número usable o NaN


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
```

```
#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 (errors="coerce").
#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)
```

```
                                   count         mean           std    min  \
pH agua:suelo                     1091.0  3267.728642  16316.966217  0.050
Materia organica                 92404.0     5.704342      1.138338  0.230
Fósforo Bray II                  92406.0     4.993972     16.793955  0.000
Azufre Fosfato monocalcico       90941.0    28.283201     74.200318  0.088
Acidez Intercambiable            90528.0    14.166891     56.175217 -1.010
Aluminio intercambiable          47455.0     2.304869      2.461865  0.000
Calcio intercambiable            47839.0     1.924856      2.498751  0.000
Magnesio intercambiable          92399.0     7.364686      9.424392  0.028
Potasio intercambiable           92391.0     2.025360      2.739217  0.006
Sodio intercambiable             92391.0     0.364194      0.704634  0.005
capacidad de intercambio cationico  92390.0     0.303673      1.403762 -0.040
Conductividad electrica          92322.0    11.182433     10.964558  0.012
Hierro disponible olsen          91191.0     3.122883     41.297946  0.004
Cobre disponible                 86517.0   237.525414    337.583085  0.100
Manganeso disponible Olsen       86516.0     2.745656      3.729914  0.000
Zinc disponible Olsen            86517.0     7.752690     14.533133  0.112
Boro disponible                  86689.0     3.721404     13.679149  0.000
Hierro disponible doble acido    90250.0     0.375034      2.930295  0.000
Cobre disponible doble acido      4786.0    60.049266     65.193253  0.240
Manganeso disponible doble acido  4786.0     1.142405      1.860582  0.160
Zinc disponible doble  acido      4786.0    10.583307     17.442350  0.040

                                      25%       50%        75%        max
pH agua:suelo                     4.78000    5.3300    6.18000  181818.00
Materia organica                  4.95000    5.4600    6.26000      31.95
Fósforo Bray II                   1.68100    2.7630    5.19175    2094.00
Azufre Fosfato monocalcico        3.03100    8.3010   25.78000    3015.00
Acidez Intercambiable             3.70700    6.1690   11.24000    4862.00
Aluminio intercambiable           0.82700    1.6900    2.94500     125.10
Calcio intercambiable             0.57300    1.3370    2.48000     277.20
Magnesio intercambiable           1.38750    4.0900    9.58000     171.80
Potasio intercambiable            0.39000    1.0060    2.46700      68.52
Sodio intercambiable              0.11400    0.2170    0.42000      64.06
capacidad de intercambio cationico  0.07000    0.0700    0.17000      99.11
Conductividad electrica           4.23100    7.4480   14.06000     207.04
Hierro disponible olsen           0.14800    0.2260    0.40900    1780.00
Cobre disponible                 51.90000  110.3600  301.77000    7208.00
Manganeso disponible Olsen        0.50000    1.8640    3.40000     219.50
Zinc disponible Olsen             2.05800    4.2750    8.51600     538.10
Boro disponible                   0.50000    1.3200    3.31300    1874.00
Hierro disponible doble acido     0.13000    0.2100    0.33500     267.60
Cobre disponible doble acido     22.33000   35.8950   79.80000    1263.00
Manganeso disponible doble acido  0.36000    0.8000    1.52000      44.08
Zinc disponible doble  acido      0.88925    3.3215   13.56000     426.98
```

```
# 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 y
#para cada columna numérica si el valor corresponde a un outlier (True) o no (False).

#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 DataFrame original.
#Al final, guardo esa máscara booleana en outlier_df bajo el nombre de la columna correspondiente.


# 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
```

## Reglas de rango físico

pH debe estar entre 0 y 14.

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