

Prompt

im trying to load this csv in colab from my drive and its not working can you show me the proper way to mount drive and read the file also i need to print the head and basic info

Response

Sure, here's the standard setup for loading a CSV from Google Drive in Colab:

```
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')

df =
pd.read_csv('/content/drive/MyDrive/Data_preprocessing/population_data
.csv'
```

Prompt

i have some exploration done i need to check missing values and duplicates and also show me which columns arent numeric

Response

You can extend the initial exploration with a few extra checks.

Here's how to look for null values, duplicated rows, and any columns that aren't numeric

```
# Check for missing values
print("\nMissing values per column:")
print(df.isna().sum())

# Check for duplicates
print(f"\nNumber of duplicate rows: {df.duplicated().sum()}")

# Non numeric columns
print("\nNon-numeric columns:")
print(df.select_dtypes(exclude='number').columns.tolist())
```

Prompt

some of these column names are too long i want shorter ones show me how to rename them

Response

You can rename columns by passing a dictionary that maps the original names to the new ones.

Here's an example using the columns in your dataset — this will make the names much easier to work with when cleaning or plotting:

```
df = df.rename(columns={  
    'Country (or dependency)': 'country',  
    'Population 2025': 'population_2025',  
    'Fert. Rate': 'fertility_rate',  
    'Median Age': 'median_age',  
    'Urban Pop %': 'urban_pop_pct'  
})  
  
# Keep a clean copy for later steps  
df_pop = df.copy()  
df_pop.head()
```

After this, `df.columns` should show consistent, Python-friendly names.

Prompt

im trying to scrape the life expectancy table from
<https://www.worldometers.info/demographics/life-expectancy/> but `read_html`
keeps giving me empty list do you know how to get it

Response

Worldometers blocks some requests unless you send a User-Agent header, so `read_html` alone won't always work.

Instead, fetch the page manually with `requests` and then pass the HTML into `read_html`:

```
import requests  
  
url = "https://www.worldometers.info/demographics/life-expectancy/"  
headers = {  
    "User-Agent": (  
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "  
        "AppleWebKit/537.36 (KHTML, like Gecko) "  
        "Chrome/124.0.0.0 Safari/537.36"  
    )  
}  
  
resp = requests.get(url, headers=headers, timeout=30)  
print("Status:", resp.status_code, "| HTML length:", len(resp.text))
```

```
html = resp.text
```

If the status code is 200 and the HTML looks long, the tables will load correctly in the next step.

Prompt

that worked how do i pull the tables now and see the first few rows

Response

Now you can use `pd.read_html` to extract all tables from that HTML.

The life expectancy data is usually the first one:

```
import pandas as pd

tables = pd.read_html(html)
print("Number of tables found:", len(tables))

# Quick preview of the first few tables
for i, t in enumerate(tables[:3]):
    print(f"\n--- Table {i} preview ---")
    display(t.head(5))
```

This helps confirm that you're grabbing the correct table before cleaning it.

Prompt

ok i can see the tables now table 0 looks like the life expectancy one but the column names are a bit messy and the numbers are showing up as objects

i want to clean the column names and make the numeric cols real numbers but also not crash if the website changes a bit

can you show me a good way to do that

Response

Great, since table 0 is the correct one, you can first copy it into its own DataFrame, tidy up the column names, and then convert the numeric-looking columns to actual numeric types.

Here's a pattern that's fairly robust even if the site changes slightly:

```
# Copy the full table so we keep every column
```

```

df_life = tables[0].copy()
df_life.columns = [str(c).strip() for c in df_life.columns]

print("Columns in scraped table:")
print(df_life.columns.tolist())

# Preview the first few rows
display(df_life.head())

```

Now we'll standardise the column names (lowercase, underscores) and convert the key numeric columns. We'll also keep a raw backup in case you need the original later.

```

import re

# ① Keep original scraped table safe
df_life_raw = df_life.copy()

# ② Standardize column names
df_life.columns = (
    df_life.columns
    .str.strip()          # remove extra spaces
    .str.lower()           # lowercase
    .str.replace(' ', '_') # replace spaces with underscores
)

print("Standardized column names:")
print(df_life.columns.tolist())

```

Now define which columns should be numeric. To make this safer, we first list the ones we expect, then keep only those that actually exist in case the website layout changes.

```

# ③ Convert numeric columns (leave 'country' as text)
# define numeric columns using your actual column names
num_cols = ['rank', 'life_expectancy_(both.sexes)',
'females_life_expectancy', 'males_life_expectancy']

# only keep ones that actually exist (in case the site changes)
num_cols = [c for c in num_cols if c in df_life.columns]

```

```
# convert them
df_life[num_cols] = df_life[num_cols].apply(
    lambda s: pd.to_numeric(
        s.astype(str).str.replace(r'^\d\.\-]', '', regex=True),
        errors='coerce'
    )
)
```

Finally, do a quick sanity check on the cleaned data:

```
# ④ Quick preview
print(f"\nShape: {df_life.shape[0]} rows x {df_life.shape[1]} columns")
display(df_life.head())

# ⑤ Confirm data types
print("\nData types:")
print(df_life.dtypes)
```

Now you should have clean, Python-friendly column names and proper numeric types for the life expectancy metrics.

Prompt

now i need to scrape the education index from
<https://worldpopulationreview.com/country-rankings/education-index-by-country> but when i just use read_html im getting 403 errors sometimes can you show me the right way to pull that page and get the table

Response

That site also blocks requests unless you include a User-Agent header, similar to Worldometers. The safest approach is:

1. Fetch the page using `requests` with headers
2. Pass the returned HTML into `pandas.read_html`

3. Extract and clean the first table

Here's the full setup:

```
import requests
import pandas as pd
from io import StringIO

# --- 1) Fetch HTML (with headers to avoid 403) ---
url =
"https://worldpopulationreview.com/country-rankings/education-index-by-
-country"
headers = {
    "User-Agent": (
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
        "AppleWebKit/537.36 (KHTML, like Gecko) "
        "Chrome/124.0.0.0 Safari/537.36"
    )
}

resp = requests.get(url, headers=headers, timeout=30)
resp.raise_for_status()
html = resp.text
print("Status:", resp.status_code, "| HTML length:", len(html))
```

Once you get a 200 response and the HTML length is large enough, you can parse the tables.

Prompt

ok page is loading now how do i get table and clean the column names again like we did for the other one

Response

`read_html` should work once the HTML is fetched manually. Then we take the first table and clean it:

```
# --- 2) Parse tables (simple path) ---
tables = pd.read_html(StringIO(html))
```

```
print("Number of tables found:", len(tables))

# Use the first table (usually the main one)
df_edu = tables[0].copy()
```

Now clean the column names in the same consistent style:

```
# --- 3) Standardize column names & quick preview ---
df_edu.columns = [str(c).strip().lower().replace(' ', '_') for c in
df_edu.columns]

print("Columns in scraped table:", df_edu.columns.tolist())
display(df_edu.head())
```

And convert numeric-like columns if present:

```
# --- 4) (Optional) Convert numeric-like columns to numeric ---
num_cols = [c for c in df_edu.columns if 'index' in c or 'score' in c
or c in ('rank',)]

if num_cols:
    df_edu[num_cols] = df_edu[num_cols].apply(
        lambda s: pd.to_numeric(
            s.astype(str).str.replace(r'^\d\.\-\]', '', regex=True),
            errors='coerce'
        )
    )
```

Finally, keep a raw copy in case you need future comparisons:

```
# Keep a raw copy if you plan further cleaning
df_edu_raw = df_edu.copy()
```

Prompt

the education table is nearly ok now but theres some weird columns called unnamed something and also the column names have strange characters in them from encoding i want to drop any unnamed cols and then fully clean the column names so theres no weird arrows or symbols how do i do that

Response

That's common with scraped tables — extra "Unnamed" columns and strange characters from encoding. You can handle this in two steps:

1. Drop any columns whose name starts with "Unnamed"
 2. Re-clean the column names to remove non-ASCII or weird symbols and standardise the format

Here's how you can do it:

```
# Drop any columns that start with 'unnamed'  
df_edu = df_edu.loc[:, ~df_edu.columns.str.contains('unnamed',  
case=False)]  
  
print("Columns after dropping unnamed ones:")  
print(df_edu.columns.tolist())
```

Now we'll normalise the remaining column names more aggressively:

```
# Fix messy encoding and standardize the education column name
df_edu.columns = (
    df_edu.columns
    .str.encode('ascii', 'ignore')      # remove non-ASCII characters
like '\x86\x93
    .str.decode('utf-8')
    .str.strip()
    .str.lower()
    .str.replace(r'\s+', '_', regex=True)          # spaces ->
underscores
    .str.replace('[^0-9a-zA-Z_]', '', regex=True)  # drop any leftover
symbols
)
print("Cleaned column names:")
print(df_edu.columns.tolist())
```

After this, all your column names should be clean ASCII strings with only letters, numbers, and underscores, which makes them much easier to use later when merging or plotting

Prompt

For this prompt I include the result of `sorted(df_edu['country'].unique().tolist()), sorted(df_life['country'].unique().tolist()), sorted(df_pop['country'].unique().tolist())`

And then I asked chat gpt to take note of different names and then make a normalisation function

Response

```
def normalize_country_name(name):
    if not isinstance(name, str):
        return None

    name = name.strip().lower()

    replacements = {
        # Ivory Coast / Côte d'Ivoire encoding differences
        "côte d'ivoire": "cote_divoire",
        "cã'te d'ivoire": "cote_divoire",
        "ivory coast": "cote_divoire",

        # Curaçao encoding differences
        "curaçao": "curacao",
        "curaÃ§ao".lower(): "curacao",

        # Congo variations
        "congo": "republic_of_the_congo",
        "republic of the congo": "republic_of_the_congo",
        "dr congo": "democratic_republic_of_the_congo",
        "democratic republic of the congo":
        "democratic_republic_of_the_congo",

        # Palestine variations
        "state of palestine": "palestine",
        "palestine": "palestine",
```

```
# Czechia variations
"czech republic (czechia)": "czechia",
"czechia": "czechia",

# Cabo Verde variations
"cabo verde": "cabo_verde",
"cape verde": "cabo_verde",

# South Korea variations
"south korea": "korea_republic",
"korea, republic of": "korea_republic",

# North Korea variations
"north korea": "korea_democratic_peoples_republic",
"korea, democratic people's republic of":
"korea_democratic_peoples_republic",

# Fix macao/macau
"macao": "macao",
"macau": "macao",

# Réunion encoding differences
"réunion": "reunion",
"rÃ©union": "reunion",

# Saint Kitts & Nevis
"saint kitts & nevis": "saint_kitts_and_nevis",
"saint kitts and nevis": "saint_kitts_and_nevis",

# Saint Vincent & the Grenadines
"st. vincent & grenadines": "saint_vincent_and_the_grenadines",
"saint vincent and the grenadines":
"saint_vincent_and_the_grenadines",

# Sao Tome & Principe
"sao tome & principe": "sao_tome_and_principe",
"sao tome and principe": "sao_tome_and_principe",
```

```
    }

    return replacements.get(name, name)
```

Prompt

im trying to join the population df with the life expectancy one using the country_std but the merge keeps giving me nans how do i do the left merge properly just for these two

Response

To keep all the population rows and only bring in the life expectancy column, you want a left merge on `country_std`. Something like:

```
df_merge1 = df_pop.merge(
    df_life[['country_std', 'life_expectancy_(both.sexes)']],
    on='country_std',
    how='left'
)
```

If you want to see how many didn't match:

```
df_merge1['life_expectancy_(both.sexes)'].isna().sum()
```

That tells you which ones from the population dataset didn't have a matching entry in the life expectancy table.