002_2017_clean

May 20, 2025

[1]:
```
import pandas as pd
import numpy as np
import logging
import os

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler("002_2017_clean.log"), # Actualizado para 2017
        logging.StreamHandler()
    ]
)

logging.info("Inicio del notebook de limpieza y transformación para 2017.csv␣
 ↪(002_2017_clean.ipynb).")

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

```
2025-05-20 17:46:08,016 - INFO - Inicio del notebook de limpieza y
transformación para 2017.csv (002_2017_clean.ipynb).
```

[2]:
```
# --- Definición de la ruta al archivo ---
file_path_2017_raw = "/home/nicolas/Escritorio/workshops ETL/workshop_3/data/
 ↪raw/2017.csv" # Actualizado
logging.info(f"Ruta del archivo raw a cargar: {file_path_2017_raw}")

df_2017_raw = None
df_2017_clean = None

try:
    logging.info(f"Intentando cargar el archivo CSV: {os.path.
 ↪basename(file_path_2017_raw)}")
    df_2017_raw = pd.read_csv(file_path_2017_raw)
```

```
    logging.info(f"Archivo {os.path.basename(file_path_2017_raw)} cargado␣
 ↪exitosamente.")
    logging.info(f"El DataFrame df_2017_raw tiene {df_2017_raw.shape[0]} filas␣
 ↪y {df_2017_raw.shape[1]} columnas.")
    df_2017_clean = df_2017_raw.copy()
    logging.info("Copia del DataFrame creada como df_2017_clean para␣
 ↪transformaciones.")

except Exception as e:
    logging.error(f"Ocurrió un error al cargar {os.path.
 ↪basename(file_path_2017_raw)}: {e}")
```

```
2025-05-20 17:46:08,023 - INFO - Ruta del archivo raw a cargar:
/home/nicolas/Escritorio/workshops ETL/workshop_3/data/raw/2017.csv
2025-05-20 17:46:08,025 - INFO - Intentando cargar el archivo CSV: 2017.csv
2025-05-20 17:46:08,028 - INFO - Archivo 2017.csv cargado exitosamente.
2025-05-20 17:46:08,029 - INFO - El DataFrame df_2017_raw tiene 155 filas y 12
columnas.
2025-05-20 17:46:08,029 - INFO - Copia del DataFrame creada como df_2017_clean
para transformaciones.
2025-05-20 17:46:08,025 - INFO - Intentando cargar el archivo CSV: 2017.csv
2025-05-20 17:46:08,028 - INFO - Archivo 2017.csv cargado exitosamente.
2025-05-20 17:46:08,029 - INFO - El DataFrame df_2017_raw tiene 155 filas y 12
columnas.
2025-05-20 17:46:08,029 - INFO - Copia del DataFrame creada como df_2017_clean
para transformaciones.
```

[3]:
```
if df_2017_clean is not None:
    print("DataFrame 2017 cargado y copiado para limpieza:")
    print(df_2017_clean.head().to_markdown(index=False))
else:
    print("Error al cargar df_2017_raw. No se puede continuar con la limpieza.")
    logging.error("Deteniendo el proceso de limpieza debido a un error en la␣
 ↪carga de datos.")
```

```
DataFrame 2017 cargado y copiado para limpieza:
| Country      |   Happiness.Rank |   Happiness.Score |   Whisker.high |
Whisker.low |   Economy..GDP.per.Capita. |   Family |   Health..Life.Expectancy.
|   Freedom |   Generosity |   Trust..Government.Corruption. |
Dystopia.Residual |
|:-----------|-----------------:|------------------:|---------------:|---------
-----:|---------------------------:|---------:|---------------------------:|----
------:|------------:|--------------------------------:|--------------------:|
| Norway       |                1 |             7.537 |        7.59444 |
7.47956 |                    1.61646 |  1.53352 |                    0.796667 |
0.635423 |     0.362012 |                         0.315964 |             2.27703
|
```

```
| Denmark     |               2 |            7.522 |       7.58173 |
7.46227 |                   1.48238 |  1.55112 |                  0.792566 |
0.626007 |     0.35528  |                       0.40077  |            2.31371
|
| Iceland     |               3 |            7.504 |       7.62203 |
7.38597 |                   1.48063 |  1.61057 |                  0.833552 |
0.627163 |     0.47554  |                       0.153527 |            2.32272
|
| Switzerland |               4 |            7.494 |       7.56177 |
7.42623 |                   1.56498 |  1.51691 |                  0.858131 |
0.620071 |     0.290549 |                       0.367007 |            2.27672
|
| Finland     |               5 |            7.469 |       7.52754 |
7.41046 |                   1.44357 |  1.54025 |                  0.809158 |
0.617951 |     0.245483 |                       0.382612 |            2.43018
|
```

[4]:
```python
# --- Estandarización de Nombres de Columnas para df_2017_clean ---
if df_2017_clean is not None:
    logging.info("Iniciando estandarización de nombres de columnas para df_2017.
 ↪")

    column_mapping_2017_to_standard = {
        'Country': 'country', # Original en 2017 CSV es "Country" con comillas,␣
 ↪Pandas lo lee sin ellas.
        'Happiness.Rank': 'happiness_rank',
        'Happiness.Score': 'happiness_score',
        'Economy..GDP.per.Capita.': 'economy_gdp_per_capita',
        'Family': 'social_support',
        'Health..Life.Expectancy.': 'health_life_expectancy',
        'Freedom': 'freedom_to_make_life_choices',
        'Trust..Government.Corruption.': 'perceptions_of_corruption',
        'Generosity': 'generosity'
        # "Whisker.high", "Whisker.low", "Dystopia.Residual" se eliminarán.
    }

    df_2017_clean.rename(columns=column_mapping_2017_to_standard, inplace=True)

    logging.info(f"Columnas después del renombrado (df_2017_clean):␣
 ↪{df_2017_clean.columns.tolist()}")
    print("\nColumnas después del renombrado (df_2017_clean):")
    print(df_2017_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2017_clean no está cargado. No se puede renombrar␣
 ↪columnas.")
```

```
2025-05-20 17:46:08,049 - INFO - Iniciando estandarización de nombres de
```

```
columnas para df_2017.
2025-05-20 17:46:08,051 - INFO - Columnas después del renombrado
(df_2017_clean): ['country', 'happiness_rank', 'happiness_score',
'Whisker.high', 'Whisker.low', 'economy_gdp_per_capita', 'social_support',
'health_life_expectancy', 'freedom_to_make_life_choices', 'generosity',
'perceptions_of_corruption', 'Dystopia.Residual']


Columnas después del renombrado (df_2017_clean):
| country    |   happiness_rank |   happiness_score |   Whisker.high |
Whisker.low |   economy_gdp_per_capita |   social_support |
health_life_expectancy |   freedom_to_make_life_choices |   generosity |
perceptions_of_corruption |   Dystopia.Residual |
|:----------|-----------------:|------------------:|---------------:|-----------
---:|-------------------------:|-----------------:|-------------------------:|--
--------------------------:|------------:|---------------------------:|----
---------------:|
| Norway     |                1 |             7.537 |        7.59444 |
7.47956 |                  1.61646 |           1.53352 |                 0.796667
|                       0.635423 |     0.362012 |                    0.315964 |
2.27703 |
| Denmark    |                2 |             7.522 |        7.58173 |
7.46227 |                  1.48238 |           1.55112 |                 0.792566
|                       0.626007 |     0.35528  |                    0.40077  |
2.31371 |
```

[5]:
```python
# --- Creación de la Columna 'region' para df_2017_clean ---
if df_2017_clean is not None:
    logging.info("Iniciando creación de la columna 'region' para df_2017.")

    # Mapeo actualizado incluyendo los países que faltaban
    country_to_region_map_2017 = {
        "Western Europe": ['Norway', 'Denmark', 'Iceland', 'Switzerland',␣
 ↪'Finland', 'Netherlands',
                           'Austria', 'Ireland', 'Germany', 'Belgium',␣
 ↪'Luxembourg', 'United Kingdom',
                           'France', 'Spain', 'Italy', 'Portugal', 'Cyprus',␣
 ↪'Malta', 'Greece', 'North Cyprus',
                           'Sweden'], # Añadido
        "North America": ['Canada', 'United States'],
        "Australia and New Zealand": ['Australia', 'New Zealand'],
        "Middle East and Northern Africa": ['United Arab Emirates', 'Israel',␣
 ↪'Saudi Arabia', 'Qatar',
                                            'Bahrain', 'Kuwait', 'Jordan',␣
 ↪'Lebanon', 'Egypt', 'Libya',
                                            'Tunisia', 'Morocco', 'Algeria',␣
 ↪'Sudan', 'Palestinian Territories',
```

```
                                                'Syria', 'Armenia', 'Azerbaijan',␣
↪'Georgia',
                                                'Turkey', 'Iran', 'Iraq',␣
↪'Yemen'], # Añadidos
      "Latin America and Caribbean": ['Costa Rica', 'Chile', 'Argentina',␣
↪'Mexico', 'Guatemala', 'Panama',
                                      'Colombia', 'Brazil', 'Uruguay',␣
↪'Paraguay', 'Peru', 'Bolivia',
                                      'Ecuador', 'El Salvador', 'Nicaragua',␣
↪'Belize', 'Honduras', 'Jamaica',
                                      'Trinidad and Tobago', 'Dominican␣
↪Republic', 'Venezuela',
                                      'Haiti'], # Añadido
      "Southeastern Asia": ['Thailand', 'Singapore', 'Malaysia', 'Indonesia',␣
↪'Philippines', 'Cambodia',
                            'Vietnam', 'Myanmar', 'Laos'],
      "Central and Eastern Europe": ['Czech Republic', 'Slovakia',␣
↪'Slovenia', 'Poland', 'Hungary',
                                     'Croatia', 'Bosnia and Herzegovina',␣
↪'Serbia', 'Romania', 'Bulgaria',
                                     'Estonia', 'Latvia', 'Lithuania',␣
↪'Belarus', 'Moldova', 'Ukraine',
                                     'Kosovo', 'Macedonia', 'Montenegro',␣
↪'Russia', # Usamos 'Macedonia' si así está en el df_2017
                                     'Albania'], # Añadido
      "Eastern Asia": ['China', 'Hong Kong S.A.R., China', 'Japan', 'South␣
↪Korea', 'Taiwan Province of China', 'Mongolia'],
      "Sub-Saharan Africa": ['Nigeria', 'South Africa', 'Kenya', 'Ethiopia',␣
↪'Uganda', 'Tanzania',
                            'Ghana', 'Senegal', 'Cameroon', 'Congo␣
↪(Kinshasa)', 'Congo (Brazzaville)',
                            'Angola', 'Benin', 'Burkina Faso', 'Rwanda',␣
↪'Zimbabwe', 'Zambia', 'Mozambique',
                            'Namibia', 'Madagascar', 'Botswana', 'Malawi',␣
↪'Niger', 'Mali', 'Chad',
                            'Central African Republic', 'South Sudan',␣
↪'Somalia', 'Sierra Leone',
                            'Liberia', 'Guinea', 'Ivory Coast', 'Mauritius',
                            'Gabon', 'Mauritania', 'Lesotho', 'Togo',␣
↪'Burundi'], # Añadidos
      "Southern Asia": ['India', 'Pakistan', 'Bangladesh', 'Sri Lanka',␣
↪'Nepal', 'Bhutan', 'Afghanistan',
                        'Kazakhstan', 'Kyrgyzstan', 'Tajikistan',␣
↪'Turkmenistan', 'Uzbekistan']
  }
```

```
    # Invertir el mapeo para lookup: country -> region
    region_lookup = {}
    for region, countries_in_region in country_to_region_map_2017.items():
        for country_name_in_map in countries_in_region:
            # Podríamos normalizar los nombres aquí si fuera necesario
            # ej. country_name_in_map.strip().lower()
            region_lookup[country_name_in_map] = region

    # Antes de mapear, es CRUCIAL que los nombres de país en␣
↪df_2017_clean['country']
    # coincidan con las llaves en region_lookup.
    # Vamos a hacer un intento de mapeo directo y luego revisar los no mapeados.

    # Normalizar/limpiar nombres de país en el DataFrame ANTES de mapear
    # Esto es un ejemplo, necesitarás identificar las discrepancias exactas.
    country_name_replacements = {
        "Hong Kong S.A.R., China": "Hong Kong S.A.R., China", # Asegurar que␣
↪coincida con tu mapa si es diferente
        "Taiwan Province of China": "Taiwan Province of China", # Asegurar que␣
↪coincida con tu mapa
        # Añade más reemplazos si es necesario
        # "Macedonia": "North Macedonia" # Si tu mapa usa North Macedonia, pero␣
↪el df solo Macedonia
    }
    df_2017_clean['country_standardized_for_map'] = df_2017_clean['country'].
↪replace(country_name_replacements).str.strip()

    # Aplicar el mapeo para crear la nueva columna 'region' usando la columna␣
↪estandarizada temporal
    df_2017_clean['region'] = df_2017_clean['country_standardized_for_map'].
↪map(region_lookup)

    # Eliminar la columna temporal
    df_2017_clean.drop(columns=['country_standardized_for_map'], inplace=True)

    # Verificar países que no obtuvieron una región
    unmapped_countries = df_2017_clean[df_2017_clean['region'].
↪isnull()]['country'].unique()
    if len(unmapped_countries) > 0:
        logging.warning(f"Los siguientes países NO pudieron ser mapeados a una␣
↪región y tendrán NaN en 'region': {list(unmapped_countries)}")
        print(f"\nADVERTENCIA: Países no mapeados a región:␣
↪{list(unmapped_countries)}. Revisa el mapeo 'country_to_region_map_2017' y␣
↪los nombres de país en el DataFrame.")
    else:
```

```
        logging.info("Todos los países fueron mapeados a una región␣
 ↪exitosamente.")

    logging.info("Columna 'region' creada y poblada en df_2017_clean.")
    # La verificación se hará en la siguiente celda.
else:
    logging.error("df_2017_clean no está cargado. No se puede crear la columna␣
 ↪'region'.")
```

```
2025-05-20 17:46:08,066 - INFO - Iniciando creación de la columna 'region' para
df_2017.
2025-05-20 17:46:08,070 - INFO - Todos los países fueron mapeados a una región
exitosamente.
2025-05-20 17:46:08,072 - INFO - Columna 'region' creada y poblada en
df_2017_clean.
2025-05-20 17:46:08,070 - INFO - Todos los países fueron mapeados a una región
exitosamente.
2025-05-20 17:46:08,072 - INFO - Columna 'region' creada y poblada en
df_2017_clean.
```

[6]:
```
df_2017_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 155 entries, 0 to 154
Data columns (total 13 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   country                        155 non-null    object
 1   happiness_rank                 155 non-null    int64
 2   happiness_score                155 non-null    float64
 3   Whisker.high                   155 non-null    float64
 4   Whisker.low                    155 non-null    float64
 5   economy_gdp_per_capita         155 non-null    float64
 6   social_support                 155 non-null    float64
 7   health_life_expectancy         155 non-null    float64
 8   freedom_to_make_life_choices   155 non-null    float64
 9   generosity                     155 non-null    float64
 10  perceptions_of_corruption      155 non-null    float64
 11  Dystopia.Residual              155 non-null    float64
 12  region                         155 non-null    object
dtypes: float64(10), int64(1), object(2)
memory usage: 15.9+ KB
```

[7]:
```
# --- Verificación de la Columna 'region' Creada (df_2017_clean) ---
if df_2017_clean is not None and 'region' in df_2017_clean.columns:
    logging.info("Verificando la columna 'region' creada en df_2017_clean.")
```

```
    # 1. Mostrar los valores únicos de la columna 'region'
    unique_regions = df_2017_clean['region'].unique()
    print("\nValores únicos en la columna 'region' (df_2017_clean):")
    # Convertir a lista para un print más limpio si hay NaNs (que son float)
    print(sorted([str(r) for r in unique_regions]))
    logging.info(f"Valores únicos en 'region': {sorted([str(r) for r in␣
↪unique_regions])}")

    # 2. Mostrar el conteo de países por cada región
    print("\nConteo de países por región (df_2017_clean):")
    region_value_counts = df_2017_clean['region'].value_counts(dropna=False) #␣
↪dropna=False para ver conteo de NaNs si los hay
    print(region_value_counts.to_markdown())
    logging.info(f"Conteo de valores por región: {region_value_counts.
↪to_dict()}")

    # 3. Mostrar algunas filas con país y su región asignada
    print("\nEjemplo de países y sus regiones asignadas (df_2017_clean):")
    print(df_2017_clean[['country', 'region']].sample(5).
↪to_markdown(index=False)) # Muestra 5 filas aleatorias

else:
    if df_2017_clean is None:
        logging.error("df_2017_clean no está cargado.")
    else:
        logging.error("La columna 'region' no fue creada o no se encuentra en␣
↪df_2017_clean.")
    print("\nNo se puede verificar la columna 'region'. DataFrame no cargado o␣
↪columna 'region' ausente.")
```

```
2025-05-20 17:46:08,124 - INFO - Verificando la columna 'region' creada en
df_2017_clean.
2025-05-20 17:46:08,126 - INFO - Valores únicos en 'region': ['Australia and New
Zealand', 'Central and Eastern Europe', 'Eastern Asia', 'Latin America and
Caribbean', 'Middle East and Northern Africa', 'North America', 'Southeastern
Asia', 'Southern Asia', 'Sub-Saharan Africa', 'Western Europe']
2025-05-20 17:46:08,130 - INFO - Conteo de valores por región: {'Sub-Saharan
Africa': 38, 'Middle East and Northern Africa': 23, 'Latin America and
Caribbean': 22, 'Western Europe': 21, 'Central and Eastern Europe': 21,
'Southern Asia': 12, 'Southeastern Asia': 8, 'Eastern Asia': 6, 'North America':
2, 'Australia and New Zealand': 2}


Valores únicos en la columna 'region' (df_2017_clean):
['Australia and New Zealand', 'Central and Eastern Europe', 'Eastern Asia',
'Latin America and Caribbean', 'Middle East and Northern Africa', 'North
America', 'Southeastern Asia', 'Southern Asia', 'Sub-Saharan Africa', 'Western
```

```
Europe']

Conteo de países por región (df_2017_clean):
| region                          |   count |
|:--------------------------------|--------:|
| Sub-Saharan Africa              |      38 |
| Middle East and Northern Africa |      23 |
| Latin America and Caribbean     |      22 |
| Western Europe                  |      21 |
| Central and Eastern Europe      |      21 |
| Southern Asia                   |      12 |
| Southeastern Asia               |       8 |
| Eastern Asia                    |       6 |
| North America                   |       2 |
| Australia and New Zealand       |       2 |

Ejemplo de países y sus regiones asignadas (df_2017_clean):
| country    | region                    |
|:-----------|:--------------------------|
| Germany    | Western Europe            |
| Mauritius  | Sub-Saharan Africa        |
| Australia  | Australia and New Zealand |
| Cyprus     | Western Europe            |
| Tajikistan | Southern Asia             |
```

[8]:
```python
# --- Eliminación de Columnas no Deseadas de df_2017_clean ---
if df_2017_clean is not None:
    # Nombres originales de las columnas a eliminar en el CSV de 2017
    # Estos son los nombres ANTES del renombrado de la Celda 3
    cols_to_drop_2017_original_names = ['Whisker.high', 'Whisker.low',␣
 ↪'Dystopia.Residual']

    existing_cols_to_drop = [col for col in cols_to_drop_2017_original_names if␣
 ↪col in df_2017_clean.columns]

    if existing_cols_to_drop:
        df_2017_clean.drop(columns=existing_cols_to_drop, inplace=True)
        logging.info(f"Columnas eliminadas de df_2017_clean:␣
 ↪{existing_cols_to_drop}")
        print(f"\nColumnas eliminadas de df_2017_clean:␣
 ↪{existing_cols_to_drop}")
    else:
        logging.info("No se encontraron columnas para eliminar en df_2017_clean␣
 ↪o ya fueron eliminadas.")
        print("\nNo se encontraron columnas especificadas para eliminar en␣
 ↪df_2017_clean.")
```

```
    logging.info(f"Columnas restantes en df_2017_clean: {df_2017_clean.columns.
 ↪tolist()}")
    print("DataFrame df_2017_clean después de eliminar columnas:")
    print(df_2017_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2017_clean no está cargado. No se pueden eliminar␣
 ↪columnas.")
```

```
2025-05-20 17:46:08,151 - INFO - Columnas eliminadas de df_2017_clean:
['Whisker.high', 'Whisker.low', 'Dystopia.Residual']
2025-05-20 17:46:08,152 - INFO - Columnas restantes en df_2017_clean:
['country', 'happiness_rank', 'happiness_score', 'economy_gdp_per_capita',
'social_support', 'health_life_expectancy', 'freedom_to_make_life_choices',
'generosity', 'perceptions_of_corruption', 'region']


Columnas eliminadas de df_2017_clean: ['Whisker.high', 'Whisker.low',
'Dystopia.Residual']
DataFrame df_2017_clean después de eliminar columnas:
| country   |   happiness_rank |   happiness_score |   economy_gdp_per_capita |
social_support |   health_life_expectancy |   freedom_to_make_life_choices |
generosity |   perceptions_of_corruption | region         |
|:----------|-----------------:|------------------:|-------------------------:|-
---------------:|-------------------------:|-------------------------------:|--
----------:|----------------------------:|:---------------|
| Norway    |                1 |             7.537 |                  1.61646 |
1.53352 |                0.796667 |                        0.635423 |
0.362012 |                    0.315964 | Western Europe |
| Denmark   |                2 |             7.522 |                  1.48238 |
1.55112 |                0.792566 |                        0.626007 |
0.35528  |                    0.40077  | Western Europe |
```

[9]:
```
# --- Añadir Columna 'year' ---
if df_2017_clean is not None:
    df_2017_clean['year'] = 2017 # Actualizado
    logging.info("Columna 'year' con valor 2017 añadida a df_2017_clean.")
    print("\nDataFrame df_2017_clean con la columna 'year':")
    print(df_2017_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2017_clean no está cargado. No se puede añadir la columna␣
 ↪'year'.")
```

```
2025-05-20 17:46:08,160 - INFO - Columna 'year' con valor 2017 añadida a
df_2017_clean.


DataFrame df_2017_clean con la columna 'year':
| country   |   happiness_rank |   happiness_score |   economy_gdp_per_capita |
```

```
social_support |   health_life_expectancy |   freedom_to_make_life_choices |
generosity |   perceptions_of_corruption | region         |   year |
|:----------|----------------:|-----------------:|-------------------------:|-
---------------:|-------------------------:|-------------------------------:|--
-----------:|---------------------------:|:---------------|-------:|
| Norway    |                1 |             7.537 |                  1.61646 |
1.53352 |                0.796667 |                         0.635423 |
0.362012 |                    0.315964 | Western Europe |   2017 |
| Denmark   |                2 |             7.522 |                  1.48238 |
1.55112 |                0.792566 |                         0.626007 |
0.35528  |                    0.40077  | Western Europe |   2017 |
```

[10]:
```
# --- Limpieza de Valores en Columna 'country' (Ejemplo) ---
if df_2017_clean is not None:
    if 'country' in df_2017_clean.columns:
        df_2017_clean['country'] = df_2017_clean['country'].str.strip()
        logging.info("Espacios extra eliminados de la columna 'country' en␣
 ↪df_2017_clean.")

    print("\nDataFrame df_2017_clean después de limpiar valores de 'country'␣
 ↪(si aplica):")
    print(df_2017_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2017_clean no está cargado. No se puede limpiar valores␣
 ↪de columnas.")
```

```
2025-05-20 17:46:08,170 - INFO - Espacios extra eliminados de la columna
'country' en df_2017_clean.


DataFrame df_2017_clean después de limpiar valores de 'country' (si aplica):
| country   |   happiness_rank |   happiness_score |   economy_gdp_per_capita |
social_support |   health_life_expectancy |   freedom_to_make_life_choices |
generosity |   perceptions_of_corruption | region         |   year |
|:----------|----------------:|-----------------:|-------------------------:|-
---------------:|-------------------------:|-------------------------------:|--
-----------:|---------------------------:|:---------------|-------:|
| Norway    |                1 |             7.537 |                  1.61646 |
1.53352 |                0.796667 |                         0.635423 |
0.362012 |                    0.315964 | Western Europe |   2017 |
| Denmark   |                2 |             7.522 |                  1.48238 |
1.55112 |                0.792566 |                         0.626007 |
0.35528  |                    0.40077  | Western Europe |   2017 |
```

[11]:
```
# --- Reordenamiento de Columnas para df_2017_clean ---
if df_2017_clean is not None:
    logging.info("Iniciando reordenamiento de columnas para df_2017_clean.")
```

```
    desired_column_order = [
        'year', 'region', 'country', 'happiness_rank', 'happiness_score',
        'social_support', 'health_life_expectancy', 'generosity',
        'freedom_to_make_life_choices', 'economy_gdp_per_capita',
        'perceptions_of_corruption'
    ]

    existing_columns_in_df = df_2017_clean.columns.tolist()
    final_column_order_2017 = [col for col in desired_column_order if col in␣
 ↪existing_columns_in_df]

    missing_desired_cols = [col for col in desired_column_order if col not in␣
 ↪final_column_order_2017]
    if missing_desired_cols:
        logging.warning(f"Las siguientes columnas deseadas no se encontraron en␣
 ↪df_2017_clean: {missing_desired_cols}")

    extra_cols_in_df = [col for col in existing_columns_in_df if col not in␣
 ↪final_column_order_2017]
    if extra_cols_in_df:
        logging.warning(f"Las siguientes columnas existen en df_2017_clean pero␣
 ↪no están en el orden deseado y serán eliminadas: {extra_cols_in_df}")

    try:
        df_2017_clean = df_2017_clean[final_column_order_2017]
        logging.info(f"Columnas reordenadas exitosamente para df_2017_clean.␣
 ↪Nuevo orden: {df_2017_clean.columns.tolist()}")
        print("\nDataFrame df_2017_clean después de reordenar columnas:")
        print(df_2017_clean.head(2).to_markdown(index=False))
    except KeyError as e:
        logging.error(f"Error al reordenar columnas en df_2017_clean: {e}")
else:
    logging.error("df_2017_clean no está cargado. No se puede reordenar␣
 ↪columnas.")
```

```
2025-05-20 17:46:08,181 - INFO - Iniciando reordenamiento de columnas para
df_2017_clean.
2025-05-20 17:46:08,186 - INFO - Columnas reordenadas exitosamente para
df_2017_clean. Nuevo orden: ['year', 'region', 'country', 'happiness_rank',
'happiness_score', 'social_support', 'health_life_expectancy', 'generosity',
'freedom_to_make_life_choices', 'economy_gdp_per_capita',
'perceptions_of_corruption']


DataFrame df_2017_clean después de reordenar columnas:
|   year | region          | country   |   happiness_rank |   happiness_score |
social_support |   health_life_expectancy |   generosity |
```

```
freedom_to_make_life_choices |   economy_gdp_per_capita |
perceptions_of_corruption |
|-------:|:---------------|:----------|-----------------:|------------------:|--
---------------:|-------------------------:|-------------:|---------------------
----------:|-------------------------:|----------------------------:|
|   2017 | Western Europe | Norway    |                1 |             7.537 |
1.53352 |                 0.796667 |     0.362012 |
0.635423 |                   1.61646 |                    0.315964 |
|   2017 | Western Europe | Denmark   |                2 |             7.522 |
1.55112 |                 0.792566 |     0.35528  |
0.626007 |                   1.48238 |                    0.40077  |
```

[12]:
```
# --- Verificación de Tipos de Datos Final (df_2017_clean) ---
if df_2017_clean is not None:
    logging.info("Mostrando información final de df_2017_clean (tipos de datos).
 ↪")
    print("\nInformación final del DataFrame df_2017_clean:")
    df_2017_clean.info()
else:
    logging.error("df_2017_clean no está cargado.")
```

```
2025-05-20 17:46:08,198 - INFO - Mostrando información final de df_2017_clean
(tipos de datos).


Información final del DataFrame df_2017_clean:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 155 entries, 0 to 154
Data columns (total 11 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   year                           155 non-null    int64
 1   region                         155 non-null    object
 2   country                        155 non-null    object
 3   happiness_rank                 155 non-null    int64
 4   happiness_score                155 non-null    float64
 5   social_support                 155 non-null    float64
 6   health_life_expectancy         155 non-null    float64
 7   generosity                     155 non-null    float64
 8   freedom_to_make_life_choices   155 non-null    float64
 9   economy_gdp_per_capita         155 non-null    float64
 10  perceptions_of_corruption      155 non-null    float64
dtypes: float64(7), int64(2), object(2)
memory usage: 13.4+ KB
```

[13]:
```
# --- Guardar el DataFrame Limpio df_2017_clean (Opcional) ---
if df_2017_clean is not None:
```

```
    output_path_2017 = "/home/nicolas/Escritorio/workshops ETL/workshop_3/data/
↪processed/2017_cleaned.csv" # Actualizado
    try:
        df_2017_clean.to_csv(output_path_2017, index=False)
        logging.info(f"DataFrame limpio df_2017_clean guardado en:␣
↪{output_path_2017}")
        print(f"\nDataFrame limpio df_2017_clean guardado en:␣
↪{output_path_2017}")
    except Exception as e:
        logging.error(f"Error al guardar el DataFrame limpio df_2017_clean:␣
↪{e}")
else:
    logging.error("df_2017_clean no está cargado. No se puede guardar.")
```

```
2025-05-20 17:46:08,215 - INFO - DataFrame limpio df_2017_clean guardado en:
/home/nicolas/Escritorio/workshops
ETL/workshop_3/data/processed/2017_cleaned.csv
```

```
DataFrame limpio df_2017_clean guardado en: /home/nicolas/Escritorio/workshops
ETL/workshop_3/data/processed/2017_cleaned.csv
```