002_2018_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_2018_clean.log"), # Actualizado para 2018
        logging.StreamHandler()
    ]
)

logging.info("Inicio del notebook de limpieza y transformación para 2018.csv␣
 ↪(002_2018_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 17:47:28,065 - INFO - Inicio del notebook de limpieza y
transformación para 2018.csv (002_2018_clean.ipynb).
```

[2]:
```
# --- Definición de la ruta al archivo ---
file_path_2018_raw = "/home/nicolas/Escritorio/workshops ETL/workshop_3/data/
 ↪raw/2018.csv" # Actualizado
logging.info(f"Ruta del archivo raw a cargar: {file_path_2018_raw}")

df_2018_raw = None
df_2018_clean = None

try:
    logging.info(f"Intentando cargar el archivo CSV: {os.path.
 ↪basename(file_path_2018_raw)}")
    # Manejar 'N/A' como NaN durante la carga
    df_2018_raw = pd.read_csv(file_path_2018_raw, na_values=['N/A'])
```

```
    logging.info(f"Archivo {os.path.basename(file_path_2018_raw)} cargado␣
 ↪exitosamente.")
    logging.info(f"El DataFrame df_2018_raw tiene {df_2018_raw.shape[0]} filas␣
 ↪y {df_2018_raw.shape[1]} columnas.")
    df_2018_clean = df_2018_raw.copy()
    logging.info("Copia del DataFrame creada como df_2018_clean para␣
 ↪transformaciones.")

except Exception as e:
    logging.error(f"Ocurrió un error al cargar {os.path.
 ↪basename(file_path_2018_raw)}: {e}")
```

```
2025-05-20 17:47:28,073 - INFO - Ruta del archivo raw a cargar:
/home/nicolas/Escritorio/workshops ETL/workshop_3/data/raw/2018.csv
2025-05-20 17:47:28,075 - INFO - Intentando cargar el archivo CSV: 2018.csv
2025-05-20 17:47:28,078 - INFO - Archivo 2018.csv cargado exitosamente.
2025-05-20 17:47:28,079 - INFO - El DataFrame df_2018_raw tiene 156 filas y 9
columnas.
2025-05-20 17:47:28,079 - INFO - Copia del DataFrame creada como df_2018_clean
para transformaciones.
2025-05-20 17:47:28,075 - INFO - Intentando cargar el archivo CSV: 2018.csv
2025-05-20 17:47:28,078 - INFO - Archivo 2018.csv cargado exitosamente.
2025-05-20 17:47:28,079 - INFO - El DataFrame df_2018_raw tiene 156 filas y 9
columnas.
2025-05-20 17:47:28,079 - INFO - Copia del DataFrame creada como df_2018_clean
para transformaciones.
```

[3]:
```
if df_2018_clean is not None:
    print("DataFrame 2018 cargado y copiado para limpieza:")
    print(df_2018_clean.head().to_markdown(index=False))
else:
    print("Error al cargar df_2018_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 2018 cargado y copiado para limpieza:
|   Overall rank | Country or region   |   Score |   GDP per capita |   Social
support |   Healthy life expectancy |   Freedom to make life choices |
Generosity |   Perceptions of corruption |
|---------------:|:--------------------|--------:|-----------------:|-----------
------:|--------------------------:|-------------------------------:|-----------
--:|----------------------------:|
|              1 | Finland             |   7.632 |            1.305 |
1.592 |                     0.874 |                          0.681 |
0.202 |                       0.393 |
|              2 | Norway              |   7.594 |            1.456 |
1.582 |                     0.861 |                          0.686 |
```

```
0.286 |                       0.34  |
|             3 | Denmark             |   7.555 |            1.351 |
1.59  |                      0.868 |                          0.683 |
0.284 |                       0.408 |
|             4 | Iceland             |   7.495 |            1.343 |
1.644 |                      0.914 |                          0.677 |
0.353 |                       0.138 |
|             5 | Switzerland         |   7.487 |            1.42  |
1.549 |                      0.927 |                          0.66  |
0.256 |                       0.357 |
```

[4]:
```python
# --- Estandarización de Nombres de Columnas para df_2018_clean ---
if df_2018_clean is not None:
    logging.info("Iniciando estandarización de nombres de columnas para df_2018.
 ↪")

    column_mapping_2018_to_standard = {
        'Country or region': 'country',
        'Overall rank': 'happiness_rank',
        'Score': 'happiness_score',
        'GDP per capita': 'economy_gdp_per_capita',
        'Social support': 'social_support',
        'Healthy life expectancy': 'health_life_expectancy',
        'Freedom to make life choices': 'freedom_to_make_life_choices',
        'Perceptions of corruption': 'perceptions_of_corruption',
        'Generosity': 'generosity'
    }

    df_2018_clean.rename(columns=column_mapping_2018_to_standard, inplace=True)

    logging.info(f"Columnas después del renombrado (df_2018_clean):␣
 ↪{df_2018_clean.columns.tolist()}")
    print("\nColumnas después del renombrado (df_2018_clean):")
    print(df_2018_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2018_clean no está cargado. No se puede renombrar␣
 ↪columnas.")
```

```
2025-05-20 17:47:28,102 - INFO - Iniciando estandarización de nombres de
columnas para df_2018.
2025-05-20 17:47:28,104 - INFO - Columnas después del renombrado
(df_2018_clean): ['happiness_rank', 'country', 'happiness_score',
'economy_gdp_per_capita', 'social_support', 'health_life_expectancy',
'freedom_to_make_life_choices', 'generosity', 'perceptions_of_corruption']


Columnas después del renombrado (df_2018_clean):
|   happiness_rank | country   |   happiness_score |   economy_gdp_per_capita |
```

```
social_support |   health_life_expectancy |   freedom_to_make_life_choices |
generosity |   perceptions_of_corruption |
|----------------:|:----------|-----------------:|-------------------------:|-
---------------:|--------------------------:|-------------------------------:|--
-----------:|----------------------------:|
|               1 | Finland   |            7.632 |                    1.305 |
1.592 |                   0.874 |                          0.681 |        0.202
|                     0.393 |
|               2 | Norway    |            7.594 |                    1.456 |
1.582 |                   0.861 |                          0.686 |        0.286
|                     0.34  |
```

[5]:
```python
# --- Creación de la Columna 'region' para df_2018_clean ---
if df_2018_clean is not None:
    logging.info("Iniciando creación de la columna 'region' para df_2018.")

    # Mapeo actualizado incluyendo los países que faltaban y los nuevos no␣
 ↪mapeados
    country_to_region_map_2018 = {
        "Western Europe": ['Finland', 'Denmark', 'Iceland', 'Switzerland',␣
 ↪'Netherlands', 'Austria',
                           'Ireland', 'Germany', 'Belgium', 'Luxembourg',␣
 ↪'United Kingdom', 'France',
                           'Spain', 'Portugal', 'Italy', 'Malta', 'Northern␣
 ↪Cyprus', 'Cyprus', 'Norway', 'Sweden',
                           'Greece'], # 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',␣
 ↪'Syria', 'Turkey', 'Iran', 'Iraq', 'Yemen',
                                            'Azerbaijan', 'Palestinian␣
 ↪Territories', 'Georgia', 'Armenia'], # 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',
                                         'Dominican Republic', 'Trinidad &␣
 ↪Tobago', 'Venezuela',
                                         'Haiti'], # Añadido
```

```
      "Southeastern Asia": ['Thailand', 'Singapore', 'Malaysia', 'Indonesia',␣
↪'Philippines', 'Vietnam',
                            'Cambodia', 'Myanmar', 'Laos'],
      "Central and Eastern Europe": ['Czech Republic', 'Slovakia',␣
↪'Slovenia', 'Poland', 'Hungary',
                                     'Croatia', 'Bosnia and Herzegovina',␣
↪'Serbia', 'Romania', 'Bulgaria',
                                     'Estonia', 'Latvia', 'Lithuania',␣
↪'Belarus', 'Moldova', 'Kosovo',
                                     'Macedonia', # Asumiendo que 'Macedonia'␣
↪es el nombre en el df
                                     'North Macedonia', # Incluir ambos si no␣
↪estás seguro del nombre exacto en el df
                                     'Montenegro', 'Russia', 'Ukraine',␣
↪'Albania'],
      "Eastern Asia": ['China', 'Hong Kong', 'Japan', 'South Korea',␣
↪'Taiwan', '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', 'Sudan',
                            'Mauritania', 'Lesotho', 'Togo', 'Burundi'],
      "Southern Asia": ['India', 'Pakistan', 'Bangladesh', 'Sri Lanka',␣
↪'Nepal', 'Bhutan', 'Afghanistan',
                        'Kazakhstan', 'Kyrgyzstan', 'Tajikistan',␣
↪'Turkmenistan', 'Uzbekistan']
  }

  region_lookup_2018 = {}
  for region, countries in country_to_region_map_2018.items():
      for country_name in countries:
          region_lookup_2018[country_name.strip()] = region

  # Estandarizar nombres de país en el DataFrame ANTES de mapear.
  # Debes verificar los nombres EXACTOS en df_2018_clean['country'] para␣
↪estos países no mapeados.
  country_name_replacements_2018 = {
```

```
        "Trinidad & Tobago": "Trinidad & Tobago", # Ejemplo, asegurar que␣
↪coincida con la llave del mapa
        # Si 'Macedonia' en el df debe mapear a 'North Macedonia' en tu mapa o␣
↪viceversa, ajústalo aquí o en el mapa.
        # Ej: "Macedonia": "Macedonia" (si la llave del mapa es "Macedonia")
        # "Palestinian Territories": "Palestinian Territories" (si la llave del␣
↪mapa es esa)
    }
    # Aplicar strip() primero, luego los reemplazos.
    df_2018_clean['country_standardized_for_map'] = df_2018_clean['country'].
↪str.strip().replace(country_name_replacements_2018)

    # Mapeo
    df_2018_clean['region'] = df_2018_clean['country_standardized_for_map'].
↪map(region_lookup_2018)
    df_2018_clean.drop(columns=['country_standardized_for_map'], inplace=True)

    unmapped_countries_2018 = df_2018_clean[df_2018_clean['region'].
↪isnull()]['country'].unique()
    if len(unmapped_countries_2018) > 0:
        logging.warning(f"Países NO mapeados a región en df_2018:␣
↪{list(unmapped_countries_2018)}. Revisa el mapeo␣
↪'country_to_region_map_2018' y los nombres de país en el DataFrame␣
↪(especialmente después de 'country_name_replacements_2018').")
        print(f"\nADVERTENCIA: Países no mapeados a región en df_2018:␣
↪{list(unmapped_countries_2018)}.")
    else:
        logging.info("Todos los países de df_2018 fueron mapeados a una región␣
↪exitosamente.")

    logging.info("Columna 'region' creada y poblada en df_2018_clean.")
else:
    logging.error("df_2018_clean no está cargado. No se puede crear la columna␣
↪'region'.")
```

```
2025-05-20 17:47:28,123 - INFO - Iniciando creación de la columna 'region' para
df_2018.
2025-05-20 17:47:28,128 - INFO - Todos los países de df_2018 fueron mapeados a
una región exitosamente.
2025-05-20 17:47:28,129 - INFO - Columna 'region' creada y poblada en
df_2018_clean.
2025-05-20 17:47:28,128 - INFO - Todos los países de df_2018 fueron mapeados a
una región exitosamente.
2025-05-20 17:47:28,129 - INFO - Columna 'region' creada y poblada en
df_2018_clean.
```

[6]:
```
df_2018_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156 entries, 0 to 155
Data columns (total 10 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   happiness_rank                  156 non-null    int64
 1   country                         156 non-null    object
 2   happiness_score                 156 non-null    float64
 3   economy_gdp_per_capita          156 non-null    float64
 4   social_support                  156 non-null    float64
 5   health_life_expectancy          156 non-null    float64
 6   freedom_to_make_life_choices    156 non-null    float64
 7   generosity                      156 non-null    float64
 8   perceptions_of_corruption       155 non-null    float64
 9   region                          156 non-null    object
dtypes: float64(7), int64(1), object(2)
memory usage: 12.3+ KB
```

[7]:
```
# --- Verificación de la Columna 'region' Creada (df_2018_clean) ---
if df_2018_clean is not None and 'region' in df_2018_clean.columns:
    logging.info("Verificando la columna 'region' creada en df_2018_clean.")

    unique_regions_2018 = df_2018_clean['region'].unique()
    print("\nValores únicos en la columna 'region' (df_2018_clean):")
    print(sorted([str(r) for r in unique_regions_2018]))
    logging.info(f"Valores únicos en 'region' (df_2018): {sorted([str(r) for r␣
 ↪in unique_regions_2018])}")

    print("\nConteo de países por región (df_2018_clean):")
    region_value_counts_2018 = df_2018_clean['region'].
 ↪value_counts(dropna=False)
    print(region_value_counts_2018.to_markdown())
    logging.info(f"Conteo de valores por región (df_2018):␣
 ↪{region_value_counts_2018.to_dict()}")

    print("\nEjemplo de países y sus regiones asignadas (df_2018_clean):")
    print(df_2018_clean[['country', 'region']].sample(5).
 ↪to_markdown(index=False))
else:
    # ... (mismo manejo de error que antes)
    if df_2018_clean is None: logging.error("df_2018_clean no está cargado.")
    else: logging.error("La columna 'region' no fue creada o no se encuentra en␣
 ↪df_2018_clean.")
    print("\nNo se puede verificar la columna 'region'.")
```

```
2025-05-20 17:47:28,150 - INFO - Verificando la columna 'region' creada en
df_2018_clean.
2025-05-20 17:47:28,152 - INFO - Valores únicos en 'region' (df_2018):
['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 17:47:28,154 - INFO - Conteo de valores por región (df_2018): {'Sub-
Saharan Africa': 39, 'Latin America and Caribbean': 22, 'Middle East and
Northern Africa': 22, 'Western Europe': 21, 'Central and Eastern Europe': 21,
'Southern Asia': 12, 'Southeastern Asia': 9, 'Eastern Asia': 6, 'North America':
2, 'Australia and New Zealand': 2}


Valores únicos en la columna 'region' (df_2018_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_2018_clean):
| region                          |   count |
|:--------------------------------|--------:|
| Sub-Saharan Africa              |      39 |
| Latin America and Caribbean     |      22 |
| Middle East and Northern Africa |      22 |
| Western Europe                  |      21 |
| Central and Eastern Europe      |      21 |
| Southern Asia                   |      12 |
| Southeastern Asia               |       9 |
| Eastern Asia                    |       6 |
| North America                   |       2 |
| Australia and New Zealand       |       2 |

Ejemplo de países y sus regiones asignadas (df_2018_clean):
| country            | region                          |
|:-------------------|:--------------------------------|
| Thailand           | Southeastern Asia               |
| Kuwait             | Middle East and Northern Africa |
| Brazil             | Latin America and Caribbean     |
| Dominican Republic | Latin America and Caribbean     |
| Senegal            | Sub-Saharan Africa              |
```

[8]:
```python
# --- Eliminación de Columnas no Deseadas de df_2018_clean ---
if df_2018_clean is not None:
    logging.info("No se eliminan columnas del DataFrame df_2018_clean según el␣
 ↪plan.")
    print("\nNo se eliminaron columnas de df_2018_clean.")
```

```
else:
    logging.error("df_2018_clean no está cargado.")
```

```
2025-05-20 17:47:28,162 - INFO - No se eliminan columnas del DataFrame
df_2018_clean según el plan.
```

```
No se eliminaron columnas de df_2018_clean.
```

[9]:
```
# --- Manejo de Valores Nulos en df_2018_clean ---
if df_2018_clean is not None:
    column_to_impute = 'perceptions_of_corruption'
    if column_to_impute in df_2018_clean.columns:

        nan_indices_before = df_2018_clean[df_2018_clean[column_to_impute].
 ↪isnull()].index
        nulos_antes = len(nan_indices_before)

        logging.info(f"Valores nulos en '{column_to_impute}' antes de la␣
 ↪imputación: {nulos_antes}")

        if nulos_antes > 0:
            imputed_value_used = None # Variable para guardar el valor que se␣
 ↪usó

            if 'region' in df_2018_clean.columns and df_2018_clean.
 ↪loc[nan_indices_before, 'region'].notnull().all():

                first_nan_index = nan_indices_before[0]
                region_of_first_nan = df_2018_clean.loc[first_nan_index,␣
 ↪'region']

                # Calcular la mediana solo para esa región
                median_for_specific_region =␣
 ↪df_2018_clean[df_2018_clean['region'] ==␣
 ↪region_of_first_nan][column_to_impute].median()

                if pd.notnull(median_for_specific_region):
                    imputed_value_used = median_for_specific_region
                    logging.info(f"Imputando NaNs en '{column_to_impute}' con␣
 ↪la mediana de su respectiva región. Mediana de la región␣
 ↪'{region_of_first_nan}' (para el primer NaN encontrado) es:␣
 ↪{imputed_value_used:.4f}")
                    df_2018_clean[column_to_impute] = df_2018_clean.
 ↪groupby('region')[column_to_impute].transform(lambda x: x.fillna(x.median()))
                else: # Si no se puede calcular la mediana regional (ej. todos␣
 ↪los valores de la región son NaN)
```

```
                    logging.warning(f"No se pudo calcular la mediana para la␣
 ↪región '{region_of_first_nan}'. Se usará la mediana global.")
                    # Se pasará al bloque de mediana global

        else:
            logging.info(f"No se encontraron NaNs para imputar en␣
 ↪'{column_to_impute}'.")
    else:
        logging.warning(f"Columna '{column_to_impute}' no encontrada en␣
 ↪df_2018_clean para imputación.")
else:
    logging.error("df_2018_clean no está cargado. No se puede manejar nulos.")
```

```
2025-05-20 17:47:28,174 - INFO - Valores nulos en 'perceptions_of_corruption'
antes de la imputación: 1
2025-05-20 17:47:28,177 - INFO - Imputando NaNs en 'perceptions_of_corruption'
con la mediana de su respectiva región. Mediana de la región 'Middle East and
Northern Africa' (para el primer NaN encontrado) es: 0.1270
2025-05-20 17:47:28,177 - INFO - Imputando NaNs en 'perceptions_of_corruption'
con la mediana de su respectiva región. Mediana de la región 'Middle East and
Northern Africa' (para el primer NaN encontrado) es: 0.1270
```

[10]:
```
df_2018_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156 entries, 0 to 155
Data columns (total 10 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   happiness_rank                 156 non-null    int64
 1   country                        156 non-null    object
 2   happiness_score                156 non-null    float64
 3   economy_gdp_per_capita         156 non-null    float64
 4   social_support                 156 non-null    float64
 5   health_life_expectancy         156 non-null    float64
 6   freedom_to_make_life_choices   156 non-null    float64
 7   generosity                     156 non-null    float64
 8   perceptions_of_corruption      156 non-null    float64
 9   region                         156 non-null    object
dtypes: float64(7), int64(1), object(2)
memory usage: 12.3+ KB
```

[11]:
```
# --- Añadir Columna 'year' ---
if df_2018_clean is not None:
    df_2018_clean['year'] = 2018 # Actualizado
    logging.info("Columna 'year' con valor 2018 añadida a df_2018_clean.")
    # (El print se puede omitir aquí si ya se ve en la siguiente celda)
```

```
else:
    logging.error("df_2018_clean no está cargado.")
```

```
2025-05-20 17:47:28,207 - INFO - Columna 'year' con valor 2018 añadida a
df_2018_clean.
```

[12]:
```
# --- Limpieza de Valores en Columna 'country' (Ejemplo) ---
if df_2018_clean is not None:
    if 'country' in df_2018_clean.columns:
        df_2018_clean['country'] = df_2018_clean['country'].str.strip()
        logging.info("Espacios extra eliminados de la columna 'country' en␣
 ↪df_2018_clean.")
    # (El print se puede omitir aquí si ya se ve en la siguiente celda)
else:
    logging.error("df_2018_clean no está cargado.")
```

```
2025-05-20 17:47:28,217 - INFO - Espacios extra eliminados de la columna
'country' en df_2018_clean.
```

[13]:
```
# --- Reordenamiento de Columnas para df_2018_clean ---
if df_2018_clean is not None:
    logging.info("Iniciando reordenamiento de columnas para df_2018_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_2018_clean.columns.tolist()
    final_column_order_2018 = [col for col in desired_column_order if col in␣
 ↪existing_columns_in_df]

    # ... (lógica de verificación de columnas faltantes/extras y reordenamiento␣
 ↪como en la celda para 2017)
    missing_desired_cols = [col for col in desired_column_order if col not in␣
 ↪final_column_order_2018] # Copiar lógica completa
    if missing_desired_cols: logging.warning(f"Columnas deseadas no encontradas␣
 ↪en df_2018_clean: {missing_desired_cols}")
    extra_cols_in_df = [col for col in existing_columns_in_df if col not in␣
 ↪final_column_order_2018]
    if extra_cols_in_df: logging.warning(f"Columnas extra en df_2018_clean que␣
 ↪serán eliminadas: {extra_cols_in_df}")

    try:
        df_2018_clean = df_2018_clean[final_column_order_2018]
```

```
        logging.info(f"Columnas reordenadas para df_2018_clean. Nuevo orden:␣
 ↪{df_2018_clean.columns.tolist()}")
        print("\nDataFrame df_2018_clean después de reordenar columnas:")
        print(df_2018_clean.head(2).to_markdown(index=False))
    except KeyError as e:
        logging.error(f"Error al reordenar columnas en df_2018_clean: {e}")
else:
    logging.error("df_2018_clean no está cargado.")
```

```
2025-05-20 17:47:28,230 - INFO - Iniciando reordenamiento de columnas para
df_2018_clean.
2025-05-20 17:47:28,232 - INFO - Columnas reordenadas para df_2018_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_2018_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 |
|-------:|:---------------|:----------|-----------------:|------------------:|--
--------------:|-------------------------:|------------:|----------------------
----------:|-------------------------:|---------------------------:|
|   2018 | Western Europe | Finland   |                1 |             7.632 |
1.592 |                    0.874 |        0.202 |                           0.681
|                    1.305 |                       0.393 |
|   2018 | Western Europe | Norway    |                2 |             7.594 |
1.582 |                    0.861 |        0.286 |                           0.686
|                    1.456 |                       0.34  |
```

[14]:
```
# --- Verificación de Tipos de Datos Final (df_2018_clean) ---
if df_2018_clean is not None:
    logging.info("Mostrando información final de df_2018_clean (tipos de datos).
 ↪")
    print("\nInformación final del DataFrame df_2018_clean:")
    df_2018_clean.info()
else:
    logging.error("df_2018_clean no está cargado.")
```

```
2025-05-20 17:47:28,242 - INFO - Mostrando información final de df_2018_clean
(tipos de datos).


Información final del DataFrame df_2018_clean:
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 156 entries, 0 to 155
Data columns (total 11 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   year                            156 non-null    int64
 1   region                          156 non-null    object
 2   country                         156 non-null    object
 3   happiness_rank                  156 non-null    int64
 4   happiness_score                 156 non-null    float64
 5   social_support                  156 non-null    float64
 6   health_life_expectancy          156 non-null    float64
 7   generosity                      156 non-null    float64
 8   freedom_to_make_life_choices    156 non-null    float64
 9   economy_gdp_per_capita          156 non-null    float64
 10  perceptions_of_corruption       156 non-null    float64
dtypes: float64(7), int64(2), object(2)
memory usage: 13.5+ KB
```

[15]:
```python
# --- Guardar el DataFrame Limpio df_2018_clean (Opcional) ---
if df_2018_clean is not None:
    output_path_2018 = "/home/nicolas/Escritorio/workshops ETL/workshop_3/data/processed/2018_cleaned.csv" # Actualizado
    try:
        df_2018_clean.to_csv(output_path_2018, index=False)
        logging.info(f"DataFrame limpio df_2018_clean guardado en: {output_path_2018}")
        print(f"\nDataFrame limpio df_2018_clean guardado en: {output_path_2018}")
    except Exception as e:
        logging.error(f"Error al guardar el DataFrame limpio df_2018_clean: {e}")
else:
    logging.error("df_2018_clean no está cargado.")
```

```
2025-05-20 17:47:28,282 - INFO - DataFrame limpio df_2018_clean guardado en:
/home/nicolas/Escritorio/workshops
ETL/workshop_3/data/processed/2018_cleaned.csv


DataFrame limpio df_2018_clean guardado en: /home/nicolas/Escritorio/workshops
ETL/workshop_3/data/processed/2018_cleaned.csv
```