002_2016_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_2016_clean.log"), # Actualizado para 2016
        logging.StreamHandler()
    ]
)

logging.info("Inicio del notebook de limpieza y transformación para 2016.csv␣
 ↪(002_2016_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 17:45:24,376 - INFO - Inicio del notebook de limpieza y
transformación para 2016.csv (002_2016_clean.ipynb).
```

[2]:
```
# --- Definición de la ruta al archivo ---
file_path_2016_raw = "/home/nicolas/Escritorio/workshops ETL/workshop_3/data/
 ↪raw/2016.csv" # Actualizado
logging.info(f"Ruta del archivo raw a cargar: {file_path_2016_raw}")

df_2016_raw = None
df_2016_clean = None

try:
    logging.info(f"Intentando cargar el archivo CSV: {os.path.
 ↪basename(file_path_2016_raw)}")
    df_2016_raw = pd.read_csv(file_path_2016_raw)
```

```
    logging.info(f"Archivo {os.path.basename(file_path_2016_raw)} cargado␣
 ↪exitosamente.")
    logging.info(f"El DataFrame df_2016_raw tiene {df_2016_raw.shape[0]} filas␣
 ↪y {df_2016_raw.shape[1]} columnas.")
    df_2016_clean = df_2016_raw.copy()
    logging.info("Copia del DataFrame creada como df_2016_clean para␣
 ↪transformaciones.")

except Exception as e: # Captura genérica, puedes refinarla como en el EDA
    logging.error(f"Ocurrió un error al cargar {os.path.
 ↪basename(file_path_2016_raw)}: {e}")
```

```
2025-05-20 17:45:24,382 - INFO - Ruta del archivo raw a cargar:
/home/nicolas/Escritorio/workshops ETL/workshop_3/data/raw/2016.csv
2025-05-20 17:45:24,383 - INFO - Intentando cargar el archivo CSV: 2016.csv
2025-05-20 17:45:24,387 - INFO - Archivo 2016.csv cargado exitosamente.
2025-05-20 17:45:24,388 - INFO - El DataFrame df_2016_raw tiene 157 filas y 13
columnas.
2025-05-20 17:45:24,388 - INFO - Copia del DataFrame creada como df_2016_clean
para transformaciones.
```

[3]:
```
# Verificar carga antes de proceder
if df_2016_clean is not None:
    print("DataFrame 2016 cargado y copiado para limpieza:")
    print(df_2016_clean.head().to_markdown(index=False))
else:
    print("Error al cargar df_2016_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 2016 cargado y copiado para limpieza:
| Country     | Region          |   Happiness Rank |   Happiness Score |   Lower
Confidence Interval |   Upper Confidence Interval |   Economy (GDP per Capita) |
Family |   Health (Life Expectancy) |   Freedom |   Trust (Government
Corruption) |   Generosity |   Dystopia Residual |
|:------------|:----------------|-----------------:|------------------:|---------
-------------------:|----------------------------:|--------------------------
:|---------:|--------------------------:|----------:|-------------------------
------:|-------------:|--------------------:|
| Denmark     | Western Europe |                1 |             7.526 |
7.46  |                       7.592 |                    1.44178 |  1.16374 |
0.79504 |   0.57941 |                          0.44453 |      0.36171 |
2.73939 |
| Switzerland | Western Europe |                2 |             7.509 |
7.428 |                       7.59  |                    1.52733 |  1.14524 |
0.86303 |   0.58557 |                          0.41203 |      0.28083 |
2.69463 |
```

```
| Iceland     | Western Europe |                3 |             7.501 |
7.333 |                        7.669 |                    1.42666 |  1.18326 |
0.86733 |   0.56624 |                          0.14975 |      0.47678 |
2.83137 |
| Norway      | Western Europe |                4 |             7.498 |
7.421 |                        7.575 |                    1.57744 |  1.1269  |
0.79579 |   0.59609 |                          0.35776 |      0.37895 |
2.66465 |
| Finland     | Western Europe |                5 |             7.413 |
7.351 |                        7.475 |                    1.40598 |  1.13464 |
0.81091 |   0.57104 |                          0.41004 |      0.25492 |
2.82596 |
```

[4]:
```python
# --- Estandarización de Nombres de Columnas para df_2016_clean ---
if df_2016_clean is not None:
    logging.info("Iniciando estandarización de nombres de columnas para df_2016.
 ↪")

    column_mapping_2016_to_standard = {
        'Country': 'country',
        'Region': 'region',
        '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'
    }

    df_2016_clean.rename(columns=column_mapping_2016_to_standard, inplace=True)

    logging.info(f"Columnas después del renombrado (df_2016_clean):␣
 ↪{df_2016_clean.columns.tolist()}")
    print("\nColumnas después del renombrado (df_2016_clean):")
    print(df_2016_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2016_clean no está cargado. No se puede renombrar␣
 ↪columnas.")
```

```
2025-05-20 17:45:24,410 - INFO - Iniciando estandarización de nombres de
columnas para df_2016.
2025-05-20 17:45:24,411 - INFO - Columnas después del renombrado
(df_2016_clean): ['country', 'region', 'happiness_rank', 'happiness_score',
'Lower Confidence Interval', 'Upper Confidence Interval',
'economy_gdp_per_capita', 'social_support', 'health_life_expectancy',
'freedom_to_make_life_choices', 'perceptions_of_corruption', 'generosity',
```

```
'Dystopia Residual']


Columnas después del renombrado (df_2016_clean):
| country     | region         |   happiness_rank |   happiness_score |   Lower
Confidence Interval |   Upper Confidence Interval |   economy_gdp_per_capita |
social_support |   health_life_expectancy |   freedom_to_make_life_choices |
perceptions_of_corruption |   generosity |   Dystopia Residual |
|:------------|:---------------|-----------------:|------------------:|---------
------------------:|---------------------------:|-------------------------:|--
--------------:|-------------------------:|-------------------------------:|---
-------------------------:|------------:|--------------------:|
| Denmark     | Western Europe |                1 |             7.526 |
7.46  |                      7.592 |                  1.44178 |
1.16374 |                  0.79504 |                       0.57941 |
0.44453 |      0.36171 |             2.73939 |
| Switzerland | Western Europe |                2 |             7.509 |
7.428 |                      7.59  |                  1.52733 |
1.14524 |                  0.86303 |                       0.58557 |
0.41203 |      0.28083 |             2.69463 |
```

[5]:
```python
# --- Eliminación de Columnas no Deseadas de df_2016_clean ---
if df_2016_clean is not None:
    # Nombres originales de las columnas a eliminar en el CSV de 2016
    cols_to_drop_2016_original_names = ['Lower Confidence Interval', 'Upper␣
 ↪Confidence Interval', 'Dystopia Residual']

    # Filtramos para asegurarnos que solo intentamos eliminar columnas que␣
 ↪existen
    # (Esto es por si la celda se corre múltiples veces o si alguna ya fue␣
 ↪eliminada/renombrada)
    # Para esta primera pasada, los nombres deberían ser los originales del CSV.
    existing_cols_to_drop = [col for col in cols_to_drop_2016_original_names if␣
 ↪col in df_2016_clean.columns]

    if existing_cols_to_drop:
        df_2016_clean.drop(columns=existing_cols_to_drop, inplace=True)
        logging.info(f"Columnas eliminadas de df_2016_clean:␣
 ↪{existing_cols_to_drop}")
        logging.info(f"Columnas restantes: {df_2016_clean.columns.tolist()}")
        print(f"\nColumnas eliminadas de df_2016_clean:␣
 ↪{existing_cols_to_drop}")
        print("DataFrame df_2016_clean después de eliminar columnas:")
        print(df_2016_clean.head(2).to_markdown(index=False))
    else:
        logging.info("No se encontraron columnas para eliminar en df_2016_clean␣
 ↪o ya fueron eliminadas.")
```

```
        print("\nNo se encontraron columnas especificadas para eliminar en␣
 ↪df_2016_clean.")
else:
    logging.error("df_2016_clean no está cargado. No se pueden eliminar␣
 ↪columnas.")
```

```
2025-05-20 17:45:24,420 - INFO - Columnas eliminadas de df_2016_clean: ['Lower
Confidence Interval', 'Upper Confidence Interval', 'Dystopia Residual']
2025-05-20 17:45:24,422 - INFO - Columnas restantes: ['country', 'region',
'happiness_rank', 'happiness_score', 'economy_gdp_per_capita', 'social_support',
'health_life_expectancy', 'freedom_to_make_life_choices',
'perceptions_of_corruption', 'generosity']


Columnas eliminadas de df_2016_clean: ['Lower Confidence Interval', 'Upper
Confidence Interval', 'Dystopia Residual']
DataFrame df_2016_clean después de eliminar columnas:
| country     | region         |   happiness_rank |   happiness_score |
economy_gdp_per_capita |   social_support |   health_life_expectancy |
freedom_to_make_life_choices |   perceptions_of_corruption |   generosity |
|:------------|:---------------|-----------------:|------------------:|---------
---------------:|-----------------:|-------------------------:|----------------
---------------:|---------------------------:|-------------:|
| Denmark     | Western Europe |                1 |             7.526 |
1.44178 |          1.16374 |                  0.79504 |
0.57941 |                     0.44453 |      0.36171 |
| Switzerland | Western Europe |                2 |             7.509 |
1.52733 |          1.14524 |                  0.86303 |
0.58557 |                     0.41203 |      0.28083 |
```

[6]:
```
# --- Añadir Columna 'year' ---
if df_2016_clean is not None:
    df_2016_clean['year'] = 2016 # Actualizado
    logging.info("Columna 'year' con valor 2016 añadida a df_2016_clean.")
    print("\nDataFrame df_2016_clean con la columna 'year':")
    print(df_2016_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2016_clean no está cargado. No se puede añadir la columna␣
 ↪'year'.")
```

```
2025-05-20 17:45:24,430 - INFO - Columna 'year' con valor 2016 añadida a
df_2016_clean.


DataFrame df_2016_clean con la columna 'year':
| country     | region         |   happiness_rank |   happiness_score |
economy_gdp_per_capita |   social_support |   health_life_expectancy |
freedom_to_make_life_choices |   perceptions_of_corruption |   generosity |
```

```
year |
|:------------|:---------------|-----------------:|------------------:|---------
---------------:|-----------------:|-------------------------:|---------------
--------------:|---------------------------:|-------------:|-------:|
| Denmark     | Western Europe |                1 |             7.526 |
1.44178 |          1.16374 |                  0.79504 |
0.57941 |                    0.44453 |      0.36171 |   2016 |
| Switzerland | Western Europe |                2 |             7.509 |
1.52733 |          1.14524 |                  0.86303 |
0.58557 |                    0.41203 |      0.28083 |   2016 |
```

[7]:
```
# --- Limpieza de Valores en Columnas 'country' y 'region' (Ejemplo) ---
if df_2016_clean is not None:
    if 'country' in df_2016_clean.columns:
        df_2016_clean['country'] = df_2016_clean['country'].str.strip()
        logging.info("Espacios extra eliminados de la columna 'country' en␣
 ↪df_2016_clean.")

    if 'region' in df_2016_clean.columns:
        df_2016_clean['region'] = df_2016_clean['region'].str.strip()
        logging.info("Espacios extra eliminados de la columna 'region' en␣
 ↪df_2016_clean.")

    print("\nDataFrame df_2016_clean después de limpiar valores de 'country' y␣
 ↪'region' (si aplica):")
    print(df_2016_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2016_clean no está cargado. No se puede limpiar valores␣
 ↪de columnas.")
```

```
2025-05-20 17:45:24,438 - INFO - Espacios extra eliminados de la columna
'country' en df_2016_clean.
2025-05-20 17:45:24,441 - INFO - Espacios extra eliminados de la columna
'region' en df_2016_clean.


DataFrame df_2016_clean después de limpiar valores de 'country' y 'region' (si
aplica):
| country     | region         |   happiness_rank |   happiness_score |
economy_gdp_per_capita |   social_support |   health_life_expectancy |
freedom_to_make_life_choices |   perceptions_of_corruption |   generosity |
year |
|:------------|:---------------|-----------------:|------------------:|---------
---------------:|-----------------:|-------------------------:|---------------
--------------:|---------------------------:|-------------:|-------:|
| Denmark     | Western Europe |                1 |             7.526 |
1.44178 |          1.16374 |                  0.79504 |
0.57941 |                    0.44453 |      0.36171 |   2016 |
```

```
| Switzerland | Western Europe |                2 |             7.509 |
1.52733 |          1.14524 |                  0.86303 |
0.58557 |                      0.41203 |      0.28083 |   2016 |
```

[8]:
```
# --- Reordenamiento de Columnas para df_2016_clean ---
if df_2016_clean is not None:
    logging.info("Iniciando reordenamiento de columnas para df_2016_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_2016_clean.columns.tolist()
    final_column_order_2016 = [col for col in desired_column_order if col in␣
↪existing_columns_in_df]

    # Verificar si todas las columnas deseadas están presentes después de la␣
↪limpieza
    missing_desired_cols = [col for col in desired_column_order if col not in␣
↪final_column_order_2016]
    if missing_desired_cols:
        logging.warning(f"Las siguientes columnas deseadas no se encontraron en␣
↪df_2016_clean y no se incluirán en el reordenamiento:␣
↪{missing_desired_cols}")

    # Verificar si hay columnas extra en el df que no están en el orden deseado
    extra_cols_in_df = [col for col in existing_columns_in_df if col not in␣
↪final_column_order_2016]
    if extra_cols_in_df:
        logging.warning(f"Las siguientes columnas existen en df_2016_clean pero␣
↪no están en el orden deseado y serán eliminadas: {extra_cols_in_df}")
        # Si no quieres eliminarlas, debes añadirlas a 'final_column_order_2016'
        # Ejemplo para añadirlas al final: final_column_order_2016.
↪extend(extra_cols_in_df)
        # Por ahora, la lógica actual las eliminará si no están en␣
↪desired_column_order.

    try:
        df_2016_clean = df_2016_clean[final_column_order_2016]
        logging.info(f"Columnas reordenadas exitosamente para df_2016_clean.␣
↪Nuevo orden: {df_2016_clean.columns.tolist()}")
        print("\nDataFrame df_2016_clean después de reordenar columnas:")
        print(df_2016_clean.head(2).to_markdown(index=False))
    except KeyError as e:
```

```
        logging.error(f"Error al reordenar columnas en df_2016_clean: {e}")
        # Loguear información útil para depuración
        logging.error(f"Columnas disponibles en df_2016_clean:␣
 ↪{existing_columns_in_df}")
        logging.error(f"Columnas intentadas en el reordenamiento:␣
 ↪{final_column_order_2016}")
else:
    logging.error("df_2016_clean no está cargado. No se puede reordenar␣
 ↪columnas.")
```

```
2025-05-20 17:45:24,450 - INFO - Iniciando reordenamiento de columnas para
df_2016_clean.
2025-05-20 17:45:24,451 - INFO - Columnas reordenadas exitosamente para
df_2016_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_2016_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 |
|-------:|:---------------|:------------|-----------------:|------------------
:|-----------------:|-------------------------:|-------------:|-----------------
--------------:|-------------------------:|----------------------------:|
|   2016 | Western Europe | Denmark     |                1 |             7.526 |
1.16374 |                  0.79504 |      0.36171 |
0.57941 |                  1.44178 |                    0.44453 |
|   2016 | Western Europe | Switzerland |                2 |             7.509 |
1.14524 |                  0.86303 |      0.28083 |
0.58557 |                  1.52733 |                    0.41203 |
```

[9]:
```
# --- Verificación de Tipos de Datos Final (df_2016_clean) ---
if df_2016_clean is not None:
    logging.info("Mostrando información final de df_2016_clean (tipos de datos).
 ↪")
    print("\nInformación final del DataFrame df_2016_clean:")
    df_2016_clean.info()
else:
    logging.error("df_2016_clean no está cargado.")
```

```
2025-05-20 17:45:24,460 - INFO - Mostrando información final de df_2016_clean
(tipos de datos).


Información final del DataFrame df_2016_clean:
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 157 entries, 0 to 156
Data columns (total 11 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   year                           157 non-null    int64
 1   region                         157 non-null    object
 2   country                        157 non-null    object
 3   happiness_rank                 157 non-null    int64
 4   happiness_score                157 non-null    float64
 5   social_support                 157 non-null    float64
 6   health_life_expectancy         157 non-null    float64
 7   generosity                     157 non-null    float64
 8   freedom_to_make_life_choices   157 non-null    float64
 9   economy_gdp_per_capita         157 non-null    float64
 10  perceptions_of_corruption      157 non-null    float64
dtypes: float64(7), int64(2), object(2)
memory usage: 13.6+ KB
```

[10]:
```python
# --- Guardar el DataFrame Limpio df_2016_clean (Opcional) ---
if df_2016_clean is not None:
    output_path_2016 = "/home/nicolas/Escritorio/workshops ETL/workshop_3/data/processed/2016_cleaned.csv" # Actualizado
    try:
        df_2016_clean.to_csv(output_path_2016, index=False)
        logging.info(f"DataFrame limpio df_2016_clean guardado en: {output_path_2016}")
        print(f"\nDataFrame limpio df_2016_clean guardado en: {output_path_2016}")
    except Exception as e:
        logging.error(f"Error al guardar el DataFrame limpio df_2016_clean: {e}")
else:
    logging.error("df_2016_clean no está cargado. No se puede guardar.")
```

```
2025-05-20 17:45:24,474 - INFO - DataFrame limpio df_2016_clean guardado en:
/home/nicolas/Escritorio/workshops
ETL/workshop_3/data/processed/2016_cleaned.csv


DataFrame limpio df_2016_clean guardado en: /home/nicolas/Escritorio/workshops
ETL/workshop_3/data/processed/2016_cleaned.csv
```