002_2015_clean

May 20, 2025

[1]:
```
import pandas as pd
import numpy as np
import logging
import os # Para basename en logs si es necesario

# Configuración de logging (puedes ajustar el nombre del archivo de log)
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler("002_2015_clean.log"), # Log específico para␣
 ↪limpieza de 2015
        logging.StreamHandler()
    ]
)

logging.info("Inicio del notebook de limpieza y transformación para 2015.csv␣
 ↪(002_2015_clean.ipynb).")

# Opciones de Pandas (pueden ser útiles para inspeccionar)
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 17:44:44,319 - INFO - Inicio del notebook de limpieza y
transformación para 2015.csv (002_2015_clean.ipynb).
```

[2]:
```
# --- Definición de la ruta al archivo ---
file_path_2015_raw = "/home/nicolas/Escritorio/workshops ETL/workshop_3/data/
 ↪raw/2015.csv"
logging.info(f"Ruta del archivo raw a cargar: {file_path_2015_raw}")

df_2015_raw = None # Usamos un sufijo _raw para indicar que es el dato original

try:
    logging.info(f"Intentando cargar el archivo CSV: {os.path.
 ↪basename(file_path_2015_raw)}")
```

```
    df_2015_raw = pd.read_csv(file_path_2015_raw)
    logging.info(f"Archivo {os.path.basename(file_path_2015_raw)} cargado␣
 ↪exitosamente.")
    logging.info(f"El DataFrame df_2015_raw tiene {df_2015_raw.shape[0]} filas␣
 ↪y {df_2015_raw.shape[1]} columnas.")
    # Crear una copia para trabajar y preservar el original cargado
    df_2015_clean = df_2015_raw.copy()
    logging.info("Copia del DataFrame creada como df_2015_clean para␣
 ↪transformaciones.")

except FileNotFoundError:
    logging.error(f"Error: El archivo no fue encontrado en la ruta especificada:
 ↪ {file_path_2015_raw}")
    df_2015_clean = None # Asegurarse que df_2015_clean sea None si falla la␣
 ↪carga
except pd.errors.EmptyDataError:
    logging.error(f"Error: El archivo {os.path.basename(file_path_2015_raw)}␣
 ↪está vacío.")
    df_2015_clean = None
except pd.errors.ParserError:
    logging.error(f"Error: No se pudo parsear el archivo {os.path.
 ↪basename(file_path_2015_raw)}. Verifica el formato del CSV.")
    df_2015_clean = None
except Exception as e:
    logging.error(f"Ocurrió un error inesperado al cargar {os.path.
 ↪basename(file_path_2015_raw)}: {e}")
    df_2015_clean = None
```

```
2025-05-20 17:44:44,327 - INFO - Ruta del archivo raw a cargar:
/home/nicolas/Escritorio/workshops ETL/workshop_3/data/raw/2015.csv
2025-05-20 17:44:44,329 - INFO - Intentando cargar el archivo CSV: 2015.csv
2025-05-20 17:44:44,331 - INFO - Archivo 2015.csv cargado exitosamente.
2025-05-20 17:44:44,331 - INFO - El DataFrame df_2015_raw tiene 158 filas y 12
columnas.
2025-05-20 17:44:44,332 - INFO - Copia del DataFrame creada como df_2015_clean
para transformaciones.
2025-05-20 17:44:44,329 - INFO - Intentando cargar el archivo CSV: 2015.csv
2025-05-20 17:44:44,331 - INFO - Archivo 2015.csv cargado exitosamente.
2025-05-20 17:44:44,331 - INFO - El DataFrame df_2015_raw tiene 158 filas y 12
columnas.
2025-05-20 17:44:44,332 - INFO - Copia del DataFrame creada como df_2015_clean
para transformaciones.
```

[3]:
```
# Verificar carga antes de proceder
if df_2015_clean is not None:
    print("DataFrame 2015 cargado y copiado para limpieza:")
```

```
    print(df_2015_clean.head().to_markdown(index=False))
else:
    print("Error al cargar df_2015_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 2015 cargado y copiado para limpieza:
| Country     | Region         |   Happiness Rank |   Happiness Score |
Standard Error |   Economy (GDP per Capita) |   Family |   Health (Life
Expectancy) |   Freedom |   Trust (Government Corruption) |   Generosity |
Dystopia Residual |
|:------------|:---------------|-----------------:|------------------:|---------
--------:|---------------------------:|---------:|---------------------------:|-
---------:|-------------------------------:|------------:|--------------------
:|
| Switzerland | Western Europe |                1 |             7.587 |
0.03411 |                    1.39651 |  1.34951 |                    0.94143 |
0.66557 |                        0.41978 |      0.29678 |             2.51738 |
| Iceland     | Western Europe |                2 |             7.561 |
0.04884 |                    1.30232 |  1.40223 |                    0.94784 |
0.62877 |                        0.14145 |      0.4363  |             2.70201 |
| Denmark     | Western Europe |                3 |             7.527 |
0.03328 |                    1.32548 |  1.36058 |                    0.87464 |
0.64938 |                        0.48357 |      0.34139 |             2.49204 |
| Norway      | Western Europe |                4 |             7.522 |
0.0388  |                    1.459   |  1.33095 |                    0.88521 |
0.66973 |                        0.36503 |      0.34699 |             2.46531 |
| Canada      | North America  |                5 |             7.427 |
0.03553 |                    1.32629 |  1.32261 |                    0.90563 |
0.63297 |                        0.32957 |      0.45811 |             2.45176 |
```

[4]:
```
# --- Estandarización de Nombres de Columnas para df_2015_clean (usando nombres finales estándar) ---
if df_2015_clean is not None:
    logging.info("Iniciando estandarización de nombres de columnas para df_2015.")

    column_mapping_2015_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', # Estándar global
        'Health (Life Expectancy)': 'health_life_expectancy',
        'Freedom': 'freedom_to_make_life_choices', # Estándar global
        'Trust (Government Corruption)': 'perceptions_of_corruption',
```

```
        'Generosity': 'generosity'
    }

    original_columns_2015 = df_2015_clean.columns.tolist()
    df_2015_clean.rename(columns=column_mapping_2015_to_standard, inplace=True)

    logging.info(f"Columnas después del renombrado: {df_2015_clean.columns.
 ↪tolist()}")
    print("\nColumnas después del renombrado (df_2015_clean):")
    print(df_2015_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2015_clean no está cargado. No se puede renombrar␣
 ↪columnas.")
```

```
2025-05-20 17:44:44,350 - INFO - Iniciando estandarización de nombres de
columnas para df_2015.
2025-05-20 17:44:44,352 - INFO - Columnas después del renombrado: ['country',
'region', 'happiness_rank', 'happiness_score', 'Standard Error',
'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_2015_clean):
| country     | region         |   happiness_rank |   happiness_score |
Standard Error |   economy_gdp_per_capita |   social_support |
health_life_expectancy |   freedom_to_make_life_choices |
perceptions_of_corruption |   generosity |   Dystopia Residual |
|:------------|:---------------|-----------------:|------------------:|---------
--------:|-------------------------:|-----------------:|-----------------------
-:|-------------------------------:|----------------------------:|------------
:|--------------------:|
| Switzerland | Western Europe |                1 |             7.587 |
0.03411 |                  1.39651 |          1.34951 |                   0.94143
|                      0.66557 |                    0.41978 |      0.29678 |
2.51738 |
| Iceland     | Western Europe |                2 |             7.561 |
0.04884 |                  1.30232 |          1.40223 |                   0.94784
|                      0.62877 |                    0.14145 |      0.4363  |
2.70201 |
```

[5]:
```
# --- Eliminación de Columnas no Deseadas de df_2015_clean ---
if df_2015_clean is not None:
    cols_to_drop_2015 = ['Standard Error', 'Dystopia Residual']

    # Verificar si las columnas a eliminar existen (con sus nombres originales␣
 ↪antes del mapeo)
```

```
    # o con los nombres mapeados si ya se renombraron y el nombre original ya no existe.
    # Es más seguro operar sobre los nombres originales si esta celda se corre independientemente,
    # o sobre los nuevos nombres si se asume que la celda de renombrado ya se ejecutó.
    # Por ahora, asumimos que los nombres son los originales del CSV 2015.

    existing_cols_to_drop = [col for col in cols_to_drop_2015 if col in df_2015_clean.columns]

    if existing_cols_to_drop:
        df_2015_clean.drop(columns=existing_cols_to_drop, inplace=True)
        logging.info(f"Columnas eliminadas de df_2015_clean: {existing_cols_to_drop}")
        logging.info(f"Columnas restantes: {df_2015_clean.columns.tolist()}")
        print("\nDataFrame df_2015_clean después de eliminar columnas:")
        print(df_2015_clean.head(2).to_markdown(index=False))
    else:
        logging.info("No se encontraron columnas para eliminar o ya fueron eliminadas.")
        print("\nNo se encontraron columnas especificadas para eliminar en df_2015_clean.")
else:
    logging.error("df_2015_clean no está cargado. No se pueden eliminar columnas.")
```

```
2025-05-20 17:44:44,363 - INFO - Columnas eliminadas de df_2015_clean:
['Standard Error', 'Dystopia Residual']
2025-05-20 17:44:44,364 - 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']


DataFrame df_2015_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 |
|:------------|:---------------|-----------------:|------------------:|---------
---------------:|-----------------:|-------------------------:|---------------
---------------:|-------------------------:|------------:|
| Switzerland | Western Europe |                1 |             7.587 |
1.39651 |          1.34951 |                  0.94143 |
0.66557 |                      0.41978 |      0.29678 |
| Iceland     | Western Europe |                2 |             7.561 |
1.30232 |          1.40223 |                  0.94784 |
```

```
0.62877 |                    0.14145 |      0.4363  |
```

[6]:
```
# --- Añadir Columna 'year' ---
if df_2015_clean is not None:
    df_2015_clean['year'] = 2015
    logging.info("Columna 'year' con valor 2015 añadida a df_2015_clean.")
    print("\nDataFrame df_2015_clean con la columna 'year':")
    print(df_2015_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2015_clean no está cargado. No se puede añadir la columna␣
 ↪'year'.")
```

```
2025-05-20 17:44:44,383 - INFO - Columna 'year' con valor 2015 añadida a
df_2015_clean.


DataFrame df_2015_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 |
|:------------|:---------------|-----------------:|------------------:|---------
---------------:|-----------------:|-------------------------:|---------------
---------------:|-------------------------:|------------:|------:|
| Switzerland | Western Europe |                1 |             7.587 |
1.39651 |          1.34951 |                  0.94143 |
0.66557 |                    0.41978 |      0.29678 |   2015 |
| Iceland     | Western Europe |                2 |             7.561 |
1.30232 |          1.40223 |                  0.94784 |
0.62877 |                    0.14145 |      0.4363  |   2015 |
```

[7]:
```
# --- Limpieza de Valores en Columnas 'country' y 'region' (Ejemplo) ---
if df_2015_clean is not None:
    # Para 'country' (ejemplo: quitar espacios extra al inicio/final)
    if 'country' in df_2015_clean.columns:
        df_2015_clean['country'] = df_2015_clean['country'].str.strip()
        logging.info("Espacios extra eliminados de la columna 'country'.")

    # Para 'region' (ejemplo: quitar espacios extra al inicio/final)
    if 'region' in df_2015_clean.columns:
        df_2015_clean['region'] = df_2015_clean['region'].str.strip()
        logging.info("Espacios extra eliminados de la columna 'region'.")

    # Si necesitas convertir a un caso específico (ej. Title Case, aunque los␣
 ↪datos ya parecen estar así)
    # df_2015_clean['country'] = df_2015_clean['country'].str.title()
```

```
    print("\nDataFrame df_2015_clean después de limpiar valores de 'country' y 'region' (si aplica):")
    print(df_2015_clean.head(2).to_markdown(index=False))
else:
    logging.error("df_2015_clean no está cargado. No se puede limpiar valores de columnas.")
```

```
2025-05-20 17:44:44,419 - INFO - Espacios extra eliminados de la columna 'country'.
2025-05-20 17:44:44,421 - INFO - Espacios extra eliminados de la columna 'region'.


DataFrame df_2015_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 |
|:------------|:---------------|-----------------:|------------------:|-----------------------:|-----------------:|-------------------------:|-----------------------------:|----------------------------:|-------------:|-----:|
| Switzerland | Western Europe |                1 |             7.587 |                1.39651 |          1.34951 |                  0.94143 |                      0.66557 |                     0.41978 |      0.29678 | 2015 |
| Iceland     | Western Europe |                2 |             7.561 |                1.30232 |          1.40223 |                  0.94784 |                      0.62877 |                     0.14145 |      0.4363  | 2015 |
```

[8]:
```
# --- Reordenamiento de Columnas ---
if df_2015_clean is not None:
    logging.info("Iniciando reordenamiento de columnas para df_2015_clean.")

    # Definir el orden deseado de las columnas
    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'
    ]
```

```
    # Verificar que todas las columnas deseadas existan en el DataFrame
    # Esto es importante por si alguna columna esperada no se generó o se␣
↪eliminó incorrectamente.
    # También, si hay columnas en df_2015_clean que no están en␣
↪desired_column_order,
    # estas se eliminarán al reindexar solo con desired_column_order.
    # Si quieres mantenerlas, deberías añadirlas a desired_column_order o␣
↪manejarlo diferente.

    # Por ahora, asumimos que desired_column_order contiene todas las columnas␣
↪que queremos y en el orden correcto.
    # Si alguna columna de desired_column_order no está en df_2015_clean,␣
↪causará un KeyError.
    # Vamos a asegurarnos de que solo usamos columnas que existen en el df.

    # Primero, obtenemos las columnas que realmente existen en df_2015_clean
    existing_columns_in_df = df_2015_clean.columns.tolist()

    # Filtramos desired_column_order para incluir solo las columnas que existen␣
↪en el DataFrame
    # y mantenemos el orden de desired_column_order
    final_column_order = [col for col in desired_column_order if col in␣
↪existing_columns_in_df]

    # Adicionalmente, podríamos querer incluir columnas que están en el df pero␣
↪no en el desired_order
    # al final, para no perderlas, aunque tu lista parece ser exhaustiva para␣
↪las columnas finales.
    # Si alguna columna de desired_column_order no está en df_2015_clean, no␣
↪aparecerá.

    if set(final_column_order) != set(existing_columns_in_df):
        logging.warning(f"El conjunto de columnas deseadas␣
↪({len(final_column_order)}) no coincide exactamente con las columnas␣
↪existentes en el DataFrame ({len(existing_columns_in_df)}).")
        logging.warning(f"Columnas deseadas y existentes: {final_column_order}")
        logging.warning(f"Columnas actualmente en el DataFrame:␣
↪{existing_columns_in_df}")
        # Decide cómo proceder: ¿error, o continuar con las que coinciden?
        # Por ahora, continuaremos con las que coinciden y están en el orden␣
↪deseado.

    try:
        df_2015_clean = df_2015_clean[final_column_order]
        logging.info(f"Columnas reordenadas exitosamente. Nuevo orden:␣
↪{df_2015_clean.columns.tolist()}")
```

```
        print("\nDataFrame df_2015_clean después de reordenar columnas:")
        print(df_2015_clean.head(2).to_markdown(index=False))
    except KeyError as e:
        logging.error(f"Error al reordenar columnas: Una o más columnas␣
 ↪deseadas no existen en el DataFrame. {e}")
        print(f"\nError al reordenar columnas: {e}")
        print(f"Columnas disponibles: {existing_columns_in_df}")
        print(f"Columnas deseadas (filtradas por existencia):␣
 ↪{final_column_order}")

else:
    logging.error("df_2015_clean no está cargado. No se puede reordenar␣
 ↪columnas.")
```

```
2025-05-20 17:44:44,443 - INFO - Iniciando reordenamiento de columnas para
df_2015_clean.
2025-05-20 17:44:44,445 - INFO - Columnas reordenadas exitosamente. 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_2015_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 |
|-------:|:---------------|:------------|-----------------:|------------------
:|-----------------:|-------------------------:|-------------:|-----------------
--------------:|-------------------------:|----------------------------:|
|   2015 | Western Europe | Switzerland |                1 |             7.587 |
1.34951 |                  0.94143 |      0.29678 |
0.66557 |                  1.39651 |                    0.41978 |
|   2015 | Western Europe | Iceland     |                2 |             7.561 |
1.40223 |                  0.94784 |      0.4363  |
0.62877 |                  1.30232 |                    0.14145 |
```

[9]:
```
# --- Verificación de Tipos de Datos Final ---
if df_2015_clean is not None:
    logging.info("Mostrando información final de df_2015_clean (tipos de datos).
 ↪")
    print("\nInformación final del DataFrame df_2015_clean:")
    df_2015_clean.info()
else:
    logging.error("df_2015_clean no está cargado.")
```

```
2025-05-20 17:44:44,452 - INFO - Mostrando información final de df_2015_clean
```

```
(tipos de datos).


Información final del DataFrame df_2015_clean:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 11 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   year                           158 non-null    int64
 1   region                         158 non-null    object
 2   country                        158 non-null    object
 3   happiness_rank                 158 non-null    int64
 4   happiness_score                158 non-null    float64
 5   social_support                 158 non-null    float64
 6   health_life_expectancy         158 non-null    float64
 7   generosity                     158 non-null    float64
 8   freedom_to_make_life_choices   158 non-null    float64
 9   economy_gdp_per_capita         158 non-null    float64
 10  perceptions_of_corruption      158 non-null    float64
dtypes: float64(7), int64(2), object(2)
memory usage: 13.7+ KB
```

[10]:
```python
# --- Guardar el DataFrame Limpio (Opcional) ---
if df_2015_clean is not None:
    output_path = "/home/nicolas/Escritorio/workshops ETL/workshop_3/data/
 ↪processed/2015_cleaned.csv"
    try:
        df_2015_clean.to_csv(output_path, index=False)
        logging.info(f"DataFrame limpio df_2015_clean guardado en:␣
 ↪{output_path}")
        print(f"\nDataFrame limpio df_2015_clean guardado en: {output_path}")
    except Exception as e:
        logging.error(f"Error al guardar el DataFrame limpio: {e}")
        print(f"\nError al guardar el DataFrame limpio: {e}")
else:
    logging.error("df_2015_clean no está cargado. No se puede guardar.")
```

```
2025-05-20 17:44:44,467 - INFO - DataFrame limpio df_2015_clean guardado en:
/home/nicolas/Escritorio/workshops
ETL/workshop_3/data/processed/2015_cleaned.csv


DataFrame limpio df_2015_clean guardado en: /home/nicolas/Escritorio/workshops
ETL/workshop_3/data/processed/2015_cleaned.csv
```