

## Workshop 3 ETL



Juan Carlos Quintero Esquivel (2225339)

Docente: Javier Alejandro Vergara Zorrilla

Universidad Autónoma de Occidente

Facultad de Ingeniería

Santiago de Cali

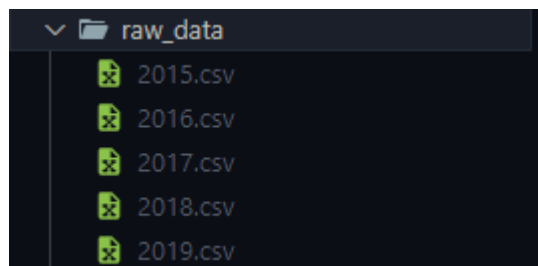
2024

## Introducción

El objetivo de este proyecto es desarrollar un modelo de machine learning capaz de predecir el nivel de felicidad de un país basándose en datos proporcionados. Para ello, se utilizarán cinco archivos CSV que contienen información relacionada con la esperanza de vida, el PIB, la confianza en el gobierno, entre otros factores. Antes de proceder con el entrenamiento del modelo, es fundamental realizar una revisión y limpieza exhaustiva de los datos para garantizar que estén libres de valores nulos o inconsistencias.

## Datos sucios

Comenzamos describiendo las columnas presentes en los diferentes datasets:



### CSV:

- **Country:** El nombre del país al que pertenecen los datos en esa fila.
- **Region:** La región geográfica a la que corresponde el país.
- **Happiness Rank:** El puesto del país en el índice de felicidad. Un rango más bajo indica mayor felicidad en comparación con otros países.
- **Happiness Score:** Representa el nivel general de felicidad de un país, utilizado para determinar su posición en el índice de felicidad.
- **Standard Error:** El margen de error asociado con el puntaje de felicidad. Un error estándar más bajo implica mayor precisión en la estimación.
- **Economy (GDP per Capita):** La contribución de la economía (PIB per cápita) al puntaje de felicidad. Valores más altos reflejan economías más prósperas.
- **Family:** Evalúa el soporte social y las relaciones familiares que reciben los ciudadanos. Valores altos indican un fuerte apoyo social y familiar.
- **Health (Life Expectancy):** Mide la esperanza de vida del país, un factor relacionado directamente con el bienestar y la felicidad.
- **Freedom:** Refleja el nivel de libertad percibido por los ciudadanos para tomar decisiones personales importantes.
- **Trust (Government Corruption):** Indica el nivel de confianza en el gobierno y la percepción de corrupción. Valores altos sugieren mayor confianza y menor corrupción.
- **Generosity:** Mide la disposición de los ciudadanos a donar tiempo o dinero a causas benéficas. Valores altos reflejan mayor generosidad.
- **Dystopia Residual:** Este valor es un ajuste hipotético que se emplea para comparar al país con una "distopía" (una sociedad extremadamente infeliz). Este indicador muestra qué tan lejos se encuentra un país de una distopía en términos de felicidad.
- **Lower Confidence Interval (Intervalo de Confianza Inferior):** Representa el límite inferior de un intervalo de confianza. Es el valor más bajo del rango en el que se

espera que esté la verdadera media o valor estimado, con un nivel de confianza específico (por ejemplo, 95%). Indica que, con este nivel de confianza, el puntaje real de felicidad será mayor o igual a este valor.

- **Upper Confidence Interval (Intervalo de Confianza Superior):** Representa el límite superior de un intervalo de confianza. Es el valor más alto dentro del rango en el que se espera que se encuentre la verdadera media o valor estimado, con un nivel de confianza dado (por ejemplo, 95%). Indica que, con este nivel de confianza, el puntaje real de felicidad será menor o igual a este valor.

### Herramientas utilizadas

- **Python:** Se empleó para crear scripts que cargan datos en la base de datos y para diseñar Dags que gestionan el flujo de trabajo en Airflow.
- **Jupyter:** Se utilizaron Notebooks de Jupyter para realizar el análisis exploratorio de datos (EDA) en ambos conjuntos de datos. Durante este proceso, la información se limpia, transforma y se generan visualizaciones que facilitan su comprensión.
- **Venv:** Herramienta utilizada para crear un entorno virtual en Python y gestionar las librerías necesarias para el proyecto.
- **Git y GitHub:** Herramientas de control de versiones empleadas para almacenar, organizar y compartir el código del proyecto.
- **SQLAlchemy:** Librería utilizada para conectarse a la base de datos, extraer los datos, procesarlos durante el EDA y posteriormente actualizarlos.
- **Pandas:** Biblioteca fundamental para analizar y manipular los datos de manera eficiente.
- **Dotenv:** Librería que permite manejar las credenciales de la base de datos de manera segura, evitando que queden expuestas en el código.
- **PostgreSQL:** Base de datos relacional utilizada para almacenar y gestionar los datos procesados.
- **Apache Kafka:** Plataforma de streaming utilizada para transferir datos de un punto a otro de manera confiable y en tiempo real.
- **Docker:** Herramienta empleada para ejecutar contenedores de Kafka y ZooKeeper, facilitando la implementación del sistema.

### Limpieza y transformación de los datos sucios

#### Importación de las librerías

Se comienza cargando las librerías necesarias para el análisis y la manipulación de los datos.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import country_converter as coco
```

## Vista general de los datos

Se imprime un resumen general del dataset para observar su contenido y estructura con el csv de cada año.

```
2015

data_2015.head

<bound method NDFrame.head of
0  Switzerland      Western Europe      1      Region Happiness Rank \
1  Iceland          Western Europe      2
2  Denmark          Western Europe      3
3  Norway           Western Europe      4
4  Canada           North America      5
..  ...
153 Rwanda          Sub-Saharan Africa  154
154 Benin           Sub-Saharan Africa  155
155 Syria           Middle East and Northern Africa  156
156 Burundi         Sub-Saharan Africa  157
157 Togo            Sub-Saharan Africa  158

Happiness Score  Standard Error  Economy (GDP per Capita)  Family \
0      7.587      0.03411      1.39651  1.34951
1      7.561      0.04884      1.38232  1.48223
2      7.527      0.03328      1.32548  1.36058
3      7.522      0.03880      1.45980  1.33095
4      7.427      0.03553      1.32629  1.32261
..  ...
153    3.465      0.03464      0.22288  0.77370
154    3.348      0.03656      0.28665  0.35386
155    3.006      0.05015      0.66320  0.47489
156    2.905      0.08658      0.01530  0.41587
157    2.839      0.06727      0.28868  0.13995
...
155    0.47179      0.32858
156    0.19727      1.83302
157    0.16681      1.56726

[158 rows x 12 columns]>
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

## Revisión del tipo de datos

Se verifica el tipo de dato asignado a cada columna para asegurar que coincidan con las necesidades del análisis con el csv de cada año.

```
data_2015.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Country                              158 non-null    object
1   Region                               158 non-null    object
2   Happiness Rank                       158 non-null    int64
3   Happiness Score                      158 non-null    float64
4   Standard Error                      158 non-null    float64
5   Economy (GDP per Capita)             158 non-null    float64
6   Family                               158 non-null    float64
7   Health (Life Expectancy)             158 non-null    float64
8   Freedom                             158 non-null    float64
9   Trust (Government Corruption)        158 non-null    float64
10  Generosity                           158 non-null    float64
11  Dystopia Residual                    158 non-null    float64
dtypes: float64(9), int64(1), object(2)
memory usage: 14.9+ KB
```

## Normalización y unificación de las columnas:

Se unificaron y normalizaron los nombres de columnas bajo categorías generales para facilitar el análisis.

```
columns_map = {  
    # país  
    'Country': 'country',  
    'Country or region': 'country',  
  
    # Happiness Score  
    'Happiness Score': 'happiness_score',  
    'Happiness.Score': 'happiness_score',  
    'Score': 'happiness_score',  
  
    # rango de felicidad  
    'Happiness Rank': 'happiness_rank',  
    'Happiness.Rank': 'happiness_rank',  
    'Overall rank': 'happiness_rank',  
  
    # economía (GDP per Capita)  
    'Economy (GDP per Capita)': 'economy',  
    'Economy..GDP.per.Capita.': 'economy',  
    'GDP per capita': 'economy',  
  
    # salud (expectativa de vida)  
    'Health (Life Expectancy)': 'health',  
    'Health..Life.Expectancy.': 'health',  
    'Healthy life expectancy': 'health',  
  
    # Soporte social  
    'Family': 'social_support',  
    'Social support': 'social_support',  
  
    # Libertad  
    'Freedom': 'freedom',  
    'Freedom to make life choices': 'freedom',  
  
    # percepción de corrupción  
    'Trust (Government Corruption)': 'corruption_perception',  
    'Trust..Government.Corruption.': 'corruption_perception',  
    'Perceptions of corruption': 'corruption_perception',  
  
    # Generosidad  
    'Generosity': 'generosity',  
  
    # Dystopia Residual  
    'Dystopia Residual': 'dystopia_residual',  
    'Dystopia.Residual': 'dystopia_residual',  
}
```

```
normalized_df = {}  
  
for year, df in happiness_df.items():  
    df = df.rename(columns=column_mapping)  
    normalized_df[year] = df
```

## Agregar columna de año:

Se crea una columna “year” para saber qué información corresponde a cada dataset antes de hacer el merge.

## Add year column

```
year_happiness_df = {}  
  
for year, df in normalized_df.items():  
    df["year"] = year  
    year_happiness_df[year] = df
```

## Merge de los dataframes:

Se combinaron los DataFrames de diferentes años, manteniendo únicamente las columnas que son comunes a todos ellos:

- **Identificación de columnas comunes:** Se calculó la intersección de los nombres de las columnas.
- **Filtrado de DataFrames:** Cada DataFrame fue reducido para contener sólo las columnas comunes.
- **Concatenación final:** Los DataFrames filtrados fueron combinados en un único DataFrame.

```
# Find the common columns across all DataFrames
shared_columns = list(set.intersection(*(set(data.columns) for data in year_happiness_df.values())))

# Filter each DataFrame to keep only the common columns
filtered_df = [data[shared_columns] for data in year_happiness_df.values()]

# Concatenate the filtered DataFrames into one, ignoring the original index
merged_df = pd.concat(filtered_df, ignore_index=True)

# Display the first few rows of the resulting DataFrame
merged_df.head()
```

## Se añade la columna de continente:

Usando la librería country converter se añade una columna de continente.

```
# Initialize the CountryConverter
cc = coco.CountryConverter()

# Convert the 'country' column to continents and store the result in a new 'continent' column
merged_df["continent"] = cc.convert(names=merged_df["country"].tolist(), to="continent")

# Display the first 10 rows of the 'continent' and 'country' columns
merged_df[["continent", "country"]].head(10)
```

```
# Define a mapping for specific countries to continents
continent_mapping = {
    "Canada": "North America",
    "Costa Rica": "Central America",
    "Mexico": "North America",
    "United States": "North America",
    "Brazil": "South America",
    "Venezuela": "South America",
    "Panama": "Central America",
    "Chile": "South America",
    "Argentina": "South America",
    "Uruguay": "South America",
    "Colombia": "South America",
    "Suriname": "South America",
    "Trinidad and Tobago": "South America",
    "El Salvador": "Central America",
    "Guatemala": "Central America",
    "Ecuador": "South America",
    "Bolivia": "South America",
    "Paraguay": "South America",
    "Nicaragua": "Central America",
    "Peru": "South America",
    "Jamaica": "Central America",
    "Dominican Republic": "Central America",
    "Honduras": "Central America",
    "Haiti": "Central America",
    "Puerto Rico": "Central America",
    "Belize": "Central America",
    "Trinidad & Tobago": "South America"
}

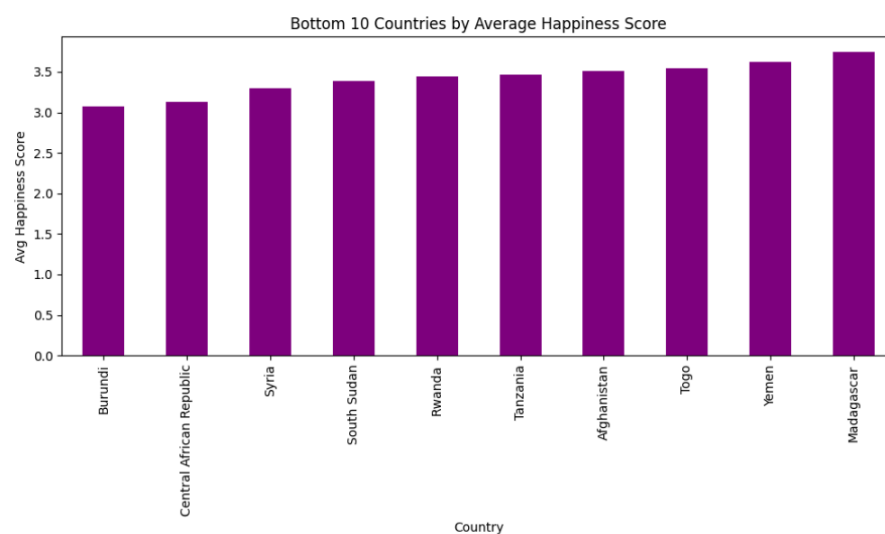
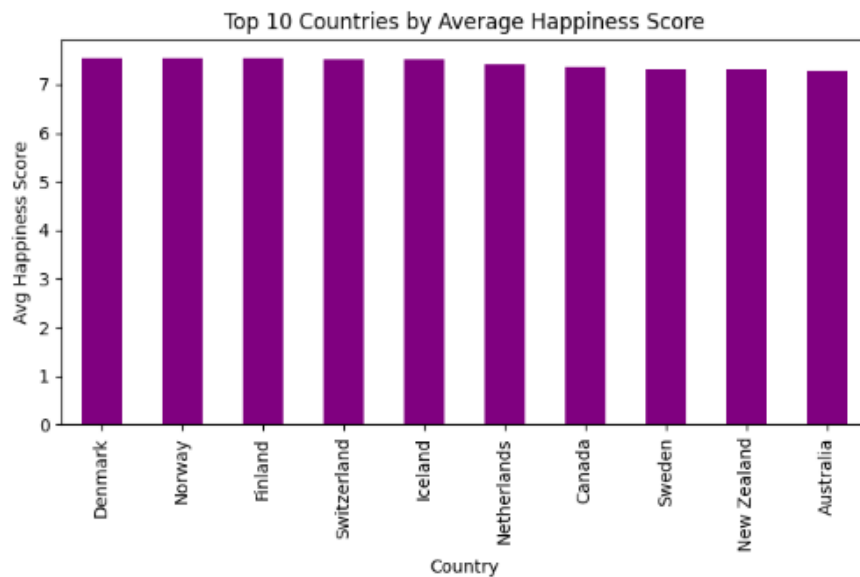
# Map the continent information based on the country and fill missing values with existing continent data
merged_df["continent"] = merged_df["country"].map(continent_mapping).fillna(merged_df["continent"])

# Display the first 10 rows of the 'continent' and 'country' columns
merged_df[["continent", "country"]].head(10)
```

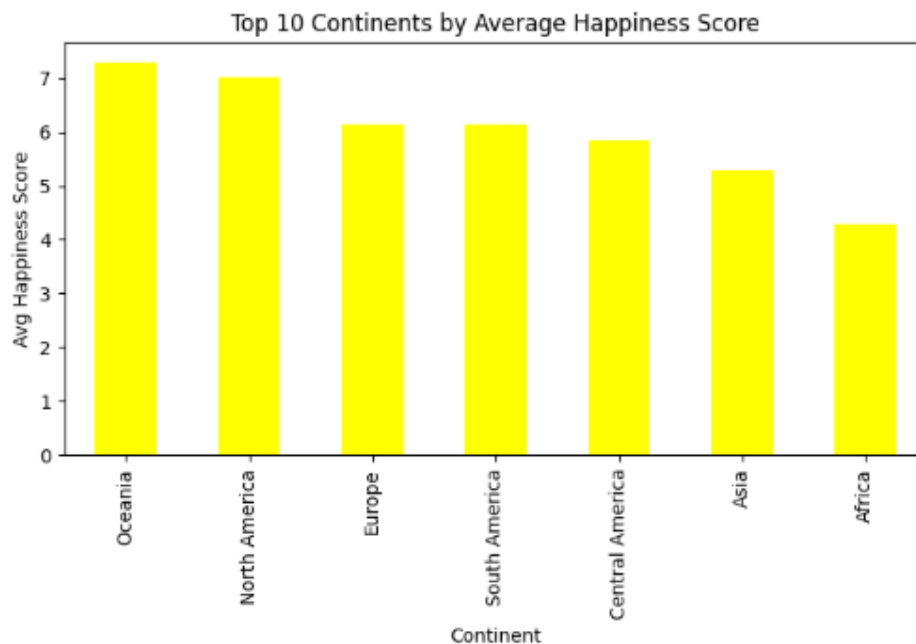
## Se hace un análisis de happines\_score por país y continente:

Despues graficamos los 10 paises con mejor happines\_score, los 10 países con menor happines score y los continentes con mejor happines\_score.

Se puede ver que dinamarca, noruega y finlandia son los que mejor promedio tienen, mientras que Burundi, la república de áfrica central y siria son las de menor promedio.



En los continentes que se puede ver que los mejores son oceanía y norteamérica, mientras los de menor promedio son áfrica y asia.



### Análisis de correlación:

Se crea una matriz de correlación que muestra las relaciones entre varios factores y métricas de felicidad. La intensidad del color indica la fuerza y la dirección de la correlación, donde los valores cercanos a 1 o -1 significan fuertes relaciones positivas o negativas, respectivamente.

**Puntuación de felicidad y economía (0,79):** Existe una fuerte correlación positiva entre la puntuación de felicidad y la economía, lo que sugiere que los países con mayor PIB per cápita tienden a tener niveles de felicidad más altos.

**Puntuación de felicidad y salud (0,74):** la salud, probablemente medida por la esperanza de vida o indicadores relacionados, también está altamente correlacionada con la felicidad, lo que indica que mejores resultados de salud están asociados con una mayor felicidad.

**Puntaje de apoyo social y felicidad (0,65):** El apoyo social muestra una correlación positiva moderada con la felicidad, lo que sugiere que las comunidades con vínculos sociales más fuertes tienden a reportar una mayor felicidad.

**Puntuación de libertad y felicidad (0,55):** La libertad muestra una correlación positiva moderada con la felicidad, lo que implica que las libertades personales contribuyen a la felicidad general.

**Rango de felicidad y puntaje de felicidad (-0,99):** como se esperaba, existe una correlación negativa muy fuerte entre el rango de felicidad y el puntaje de felicidad, dado que un puntaje más alto corresponde a un rango más bajo (mejor ranking).



**Correlaciones débiles:** la generosidad y la percepción de corrupción tienen correlaciones más débiles con la felicidad, lo que indica que estos factores pueden tener una influencia menos directa en la felicidad general en comparación con los factores económicos y de salud.

## Modelo

Después de terminar con la limpieza y unión de los datos, se empieza con la creación del modelo predictivo de felicidad. Se eligen las columnas que consideramos necesarias para la creación de este modelo.

```
features = df.drop(["happiness_score", "happiness_rank", "country"], axis=1)
target = df["happiness_score"]
```

Se importa la librería de Sklearn donde están los modelos y métricas que se van a usar para la creación del modelo y elección del mismo.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```

Se evalúan varios modelos de regresión para predecir las puntuaciones de felicidad de los países basándonos en indicadores socioeconómicos y continentales. Cada modelo presenta ventajas y metodologías únicas para abordar nuestra tarea de predicción.

**Linear Regression:** Se empieza con un modelo de regresión lineal que supone una relación rectilínea entre las características independientes y la variable objetivo.

El modelo de regresión lineal explica aproximadamente el 83% de la varianza en las puntuaciones de felicidad, lo que sugiere un ajuste razonable, aunque hay potencial de mejora con modelos más complejos.

**Random Forest Regressor:** A continuación, aplicamos un regresor de bosque aleatorio, una técnica de conjunto que agrega múltiples árboles de decisión para producir predicciones más fiables y precisas.

El modelo Random Forest logra un MSE menor y una puntuación  $R^2$  mayor que la regresión lineal, explicando aproximadamente el 86% de la varianza. Esto sugiere que capta relaciones no lineales más complejas en los datos.

**Gradient Boosting Regressor:** A continuación, probamos un regresor de aumento gradual, otro modelo de conjunto que construye árboles secuencialmente, en el que cada árbol intenta corregir los errores cometidos por el anterior.

El modelo Gradient Boosting tiene un rendimiento comparable al de Random Forest, con un MSE ligeramente inferior y una puntuación  $R^2$  similar, lo que indica que ambos métodos de conjunto son eficaces para estos datos.

**Decision Tree Regressor:** También examinamos un único regresor de árbol de decisión, un modelo sencillo e interpretable.

El modelo de árbol de decisión muestra un error mayor y una  $R^2$  menor que los modelos de conjunto, ya que es más propenso al sobreajuste y carece de la solidez que proporciona el promedio de varios árboles.

**K-Nearest Neighbors Regressor:** El regresor K-Nearest Neighbors (KNN) predice basándose en la media de los vecinos más próximos en el espacio de características.

El modelo KNN muestra un rendimiento moderado, con un error mayor que los modelos de conjunto. Puede ser útil para captar patrones localizados, pero puede tener problemas con las relaciones globales en los datos.

**Support Vector Regressor:** El regresor de vectores de soporte (SVR) intenta encontrar un hiperplano que se ajuste lo mejor posible a los datos dentro de un margen de tolerancia.

En este caso, el SVR tiene un rendimiento deficiente, con un MSE elevado y una puntuación  $R^2$  negativa, lo que indica que no capta bien las relaciones de este conjunto de datos.

**XGBoost Regressor:** Por último, utilizamos un regresor XGBoost, un modelo de refuerzo de gradiente optimizado conocido por su alto rendimiento.

El modelo XGBoost proporciona resultados competitivos, con un rendimiento cercano al de los otros modelos ensemble, capturando eficazmente los patrones no lineales.

	Model	Mean Squared Error	R2 Score
0	Linear Regression	0.210874	0.833289
1	Random Forest Regressor	0.171708	0.864253
2	Gradient Boosting Regressor	0.171200	0.864654
3	Decision Tree Regressor	0.347161	0.725545
4	K-Nearest Neighbors Regressor	0.308126	0.756405
5	Support Vector Regressor	1.265592	-0.000540
6	XGBoost Regressor	0.175235	0.861465

Al final se escogió el gradiente boosting regressor ya que es el que mejores resultados obtuvo con un  $R^2$  de 0.86, en otras palabras con un acierto del 86%.

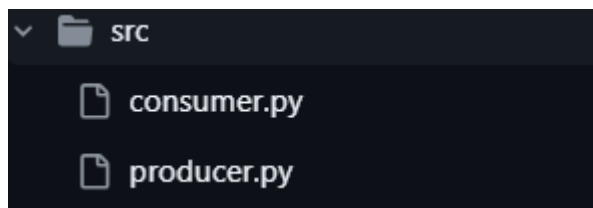
Despues se guarda el modelo en formato PKL

```
joblib.dump(gb_model, '../model/gb_model.pkl')

['../model/gb_model.pkl']
```

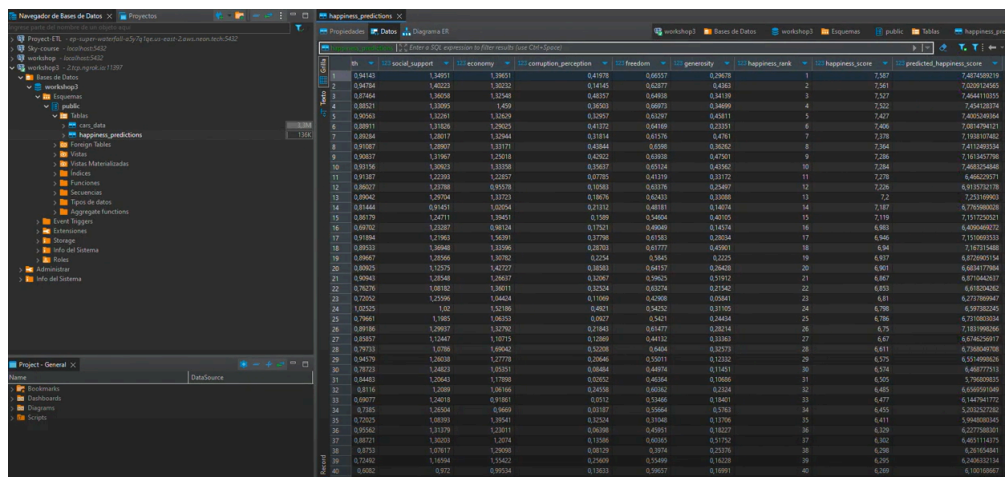
## kafka

En la carpeta src se encuentran los siguientes scripts, los cuales contienen toda la lógica necesaria para operar como productor y consumidor en la transmisión de datos. Además, el consumer se encarga de realizar la predicción del nivel de felicidad y posteriormente subirla a la base de datos. Esto permite almacenar tanto la predicción como las columnas utilizadas. A continuación, se describe brevemente la funcionalidad de cada script:



**producer.py:** Este script toma el archivo CSV limpio para su transmisión hacia el consumidor. La transmisión se realiza a través del topic llamado *"happinessPredictions"*.

**consumer.py:** Este script recibe los datos transmitidos en el topic *"happinessPredictions"*. Los datos llegan en forma de filas, y utilizando un archivo PKL, realiza la predicción añadiendo la columna *"predicted\_happiness\_score"*.



	bh	social_support	life_satisfaction	corruption_perception	freedom	generosity	happiness_rank	happiness_score	predicted_happiness_score
1	0.94143	1.34051	1.08611	0.41978	0.88557	0.28678	1	5.987	7.4874088718
2	0.94756	1.40223	1.05312	0.14145	0.82377	0.42681	2	5.981	7.5009134885
3	0.94764	1.38069	1.07546	0.40357	0.84858	0.34739	3	5.957	7.4641110315
4	0.93521	1.33085	1.4149	0.36503	0.68973	0.38888	4	5.922	7.454128274
5	0.95964	1.52061	1.02629	0.49957	0.68973	0.48811	5	5.927	7.460424896
6	0.93811	1.07526	1.29621	0.41722	0.64919	0.25333	6	5.906	7.2894796121
7	0.93384	1.38017	1.02844	0.31814	0.65178	0.4781	7	5.978	7.188107842
8	0.91887	1.28427	1.01711	0.48446	0.6986	0.24362	8	5.964	7.411492354
9	0.93837	1.37967	1.26018	0.42623	0.69338	0.47901	9	5.986	7.1613457786
10	0.93156	1.38023	1.03358	0.35537	0.65124	0.45362	10	5.984	7.4683254588
11	0.91287	1.03389	1.02857	0.07785	0.45339	0.23772	11	5.978	6.46622671
12	0.96027	1.35788	0.95578	0.16583	0.63376	0.25487	12	5.926	6.9187572178
13	0.89942	1.29754	1.07321	0.18676	0.42433	0.33888	13	5.92	7.251189863
14	0.91444	0.91451	1.05054	0.31321	0.48181	0.34874	14	5.987	6.9791888828
15	0.93179	1.24711	1.39451	0.13389	0.54804	0.40103	15	5.919	7.1517220521
16	0.89742	1.25237	0.98124	0.17421	0.46466	0.34719	16	6.068	6.4604880032
17	0.91884	1.07965	1.06391	0.37788	0.61935	0.28034	17	6.046	7.1510885533
18	0.89533	1.36848	1.03386	0.28703	0.61777	0.45981	18	6.04	7.147315488
19	0.89487	1.29488	1.04762	0.32524	0.3495	0.2322	19	6.077	6.872489154
20	0.88055	1.02575	1.42727	0.36583	0.64157	0.26428	20	6.061	6.688477884
21	0.90943	1.28548	1.06877	0.33067	0.59625	0.51912	21	6.087	6.871042837
22	0.76276	1.05182	1.08811	0.35251	0.61274	0.21542	22	6.053	6.413202882
23	0.72052	1.25588	1.04424	0.11069	0.42968	0.05841	23	6.01	6.2737868647
24	0.92525	1.02	1.02386	0.48211	0.54252	0.31105	24	6.088	6.913825245
25	0.79681	1.0885	1.06533	0.0627	0.5421	0.34854	25	6.086	6.7510888884
26	0.89186	1.29877	1.0792	0.21843	0.61477	0.28214	26	6.075	7.1811888286
27	0.85857	1.04487	1.07154	0.18489	0.44122	0.28283	27	6.07	6.8742828817
28	0.79553	1.0786	1.08842	0.52058	0.46484	0.35735	28	6.011	6.988048788
29	0.94579	1.28238	1.27778	0.26449	0.53081	0.12332	29	6.075	6.5514998828
30	0.78233	1.04023	1.05855	0.06484	0.44814	0.11401	30	6.054	6.46877313
31	0.94483	1.05443	1.17888	0.05052	0.48354	0.18888	31	6.005	5.796088815
32	0.8116	1.0889	1.06186	0.24558	0.68862	0.2324	32	6.085	6.894991588
33	0.90277	1.06713	0.91881	0.0121	0.53488	0.34871	33	6.077	6.144781772
34	0.7185	1.28508	0.9888	0.0137	0.53664	0.3783	34	6.055	5.915257252
35	0.75245	1.08887	1.0945	0.32521	0.53664	0.41786	35	6.011	6.948888888
36	0.95582	1.07379	1.06811	0.06388	0.45951	0.18227	36	6.059	6.5277888881
37	0.88221	1.08233	1.078	0.15588	0.60383	0.51752	37	6.082	6.483114375
38	0.8233	1.07877	1.08888	0.01328	0.3974	0.28278	38	6.086	6.261488881
39	0.72482	1.08844	1.05422	0.25689	0.55488	0.18228	39	6.095	6.268615114
40	0.8882	0.872	0.99534	0.18333	0.59657	0.18881	40	6.089	6.100188887

Link al video de drive con la demostración del funcionamiento:

[https://drive.google.com/file/d/12KPouRoPIdd\\_Nxhg-5y7rqMHzm4Mg6n4/view?usp=s\\_haring](https://drive.google.com/file/d/12KPouRoPIdd_Nxhg-5y7rqMHzm4Mg6n4/view?usp=s_haring)