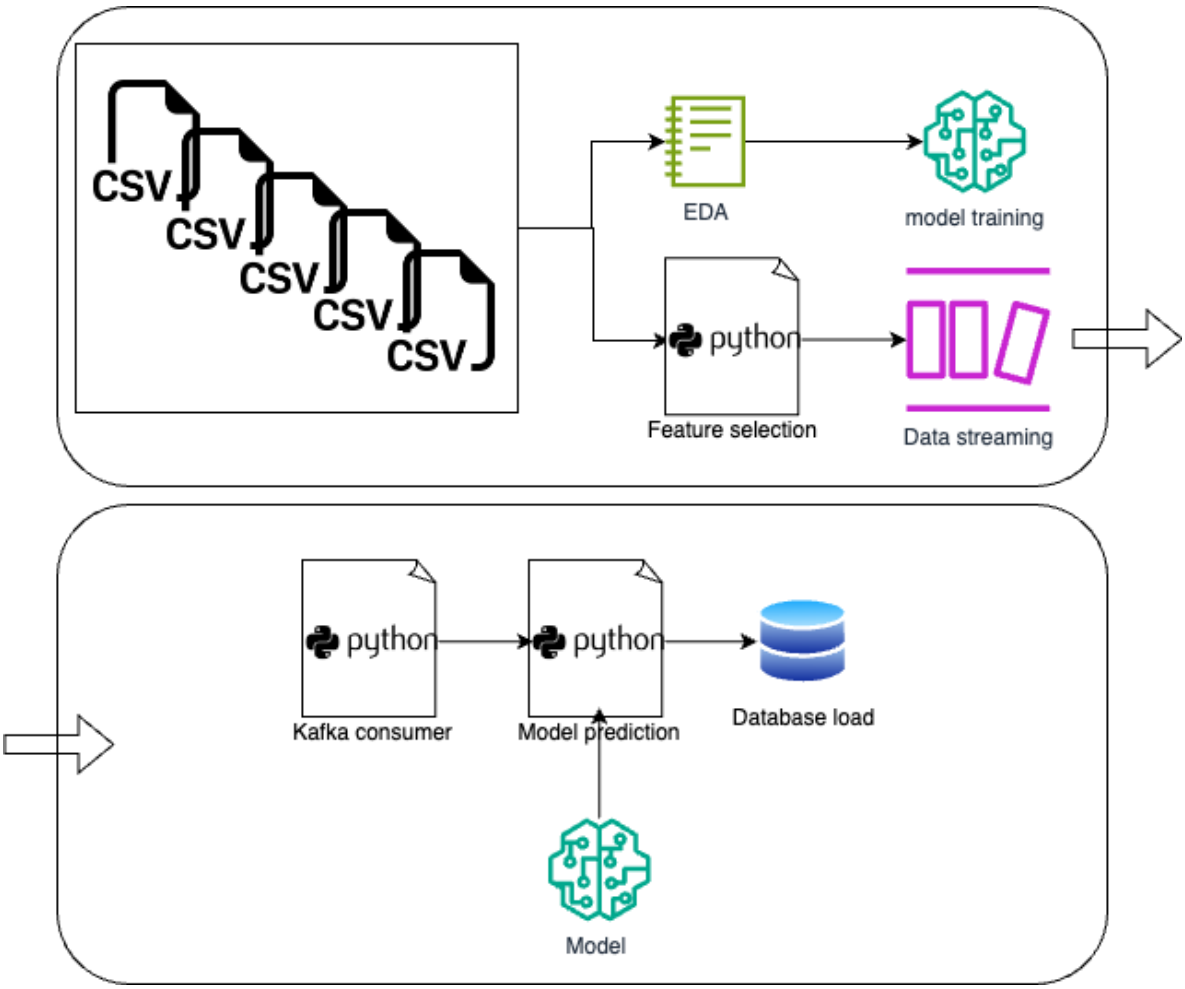
ETL Workshop 3

Machine learning and Data streaming

2221222

Joan Mateo Bermudez Collazos

El objetivo principal es desarrollar un modelo de regresión de aprendizaje automático que prediga el índice de felicidad de diferentes países basándose en los datos proporcionados en cinco archivos CSV. El proyecto implica un análisis exploratorio de datos (EDA) y procesos de extracción, transformación y carga (ETL) para extraer características relevantes de los datos.



La idea es generar un flujo de datos semejante a este

Desarrollo del Reto

Planeación

Para este reto, utilicé WSL (Windows Subsystem for Linux) junto con dos configuraciones de Docker Compose. Una configuración manejaba la base de datos PostgreSQL y la otra manejaba la aplicación de Kafka. Además, empleé la biblioteca Scikit-learn para el entrenamiento del modelo de aprendizaje automático.

Para cumplir con los requisitos del proyecto, decidí utilizar Docker para la base de datos y WSL para ejecutar el proyecto en general, dado que Kafka funciona de manera nativa en sistemas Unix. La base de datos se conectó con los notebooks de Python y Kafka stream utilizando la red local de la máquina Ubuntu (WSL).

Instalación

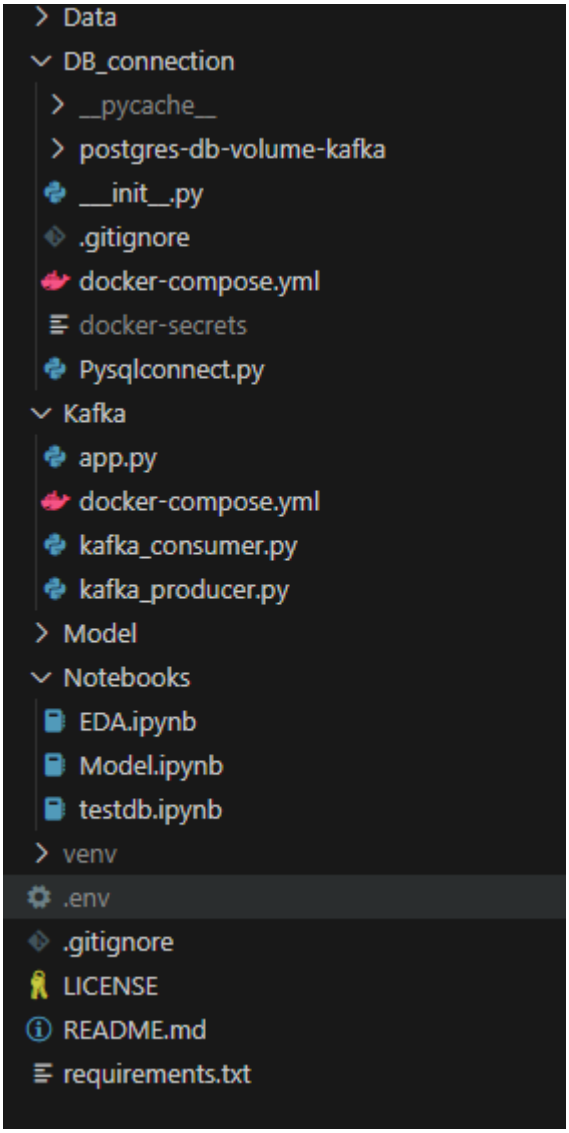
Siga estos pasos para configurar el proyecto:

1. **Instalar y Verificar WSL**:
    - Guía de Instalación de WSL: WSL Installation Guide
2. **Instalar Programas Requeridos**:
    - Instalar Python 3.
    - Instalar Docker siguiendo la guía de instalación de Docker Engine: Docker Engine Installation Guide.
3. **Crear un Entorno Virtual**:
    - Crear un entorno virtual con el comando **python3 -m venv venv**.
    - Activar el entorno virtual con el comando **source venv/bin/activate**.
    - Instalar Apache Kafka y otras dependencias usando pip con los comandos **pip install kafka-python** y **pip install -r requirements.txt**.

4. **Crear Archivos de Contraseña y el Entorno Virtual**:
    - Crear los archivos de contraseña necesarios (**./.env**, **./DB_connection/docker-secrets**) con el contenido:
        - Para el archivo .env: **DB_1 = workshop_03**, **DB_PORT = 54321**, **DB_PASSWORD = ****, **DB_HOST = localhost**, **DB_USER = postgres**.
        - Para el archivo docker-secrets: **POSTGRES_PASSWORD=****.
5. **Descargar la Imagen de Docker para la Base de Datos**:
    - En el directorio **DB_connection**, ejecutar el comando para levantar PostgreSQL con la imagen y configuraciones: **sudo docker compose up**.
    - En el directorio **Kafka**, ejecutar el archivo **docker-compose.yml** con el entorno de Kafka: **sudo docker compose up**.
6. **Ejecutar los Notebooks**:
    - Ejecutar los notebooks **EDA.ipynb** y **Model.ipynb**.
7. **Ejecutar Kafka**:
    - En una consola, ejecutar el consumidor de Kafka con el comando **python3 ./Kafka/kafka_consumer.py**.
    - En una segunda consola, ejecutar el productor de Kafka con el comando **python3 ./Kafka/kafka_producer.py**.

Estructura del Proyecto

El proyecto está organizado en las siguientes carpetas y archivos, cada uno con su propósito específico:

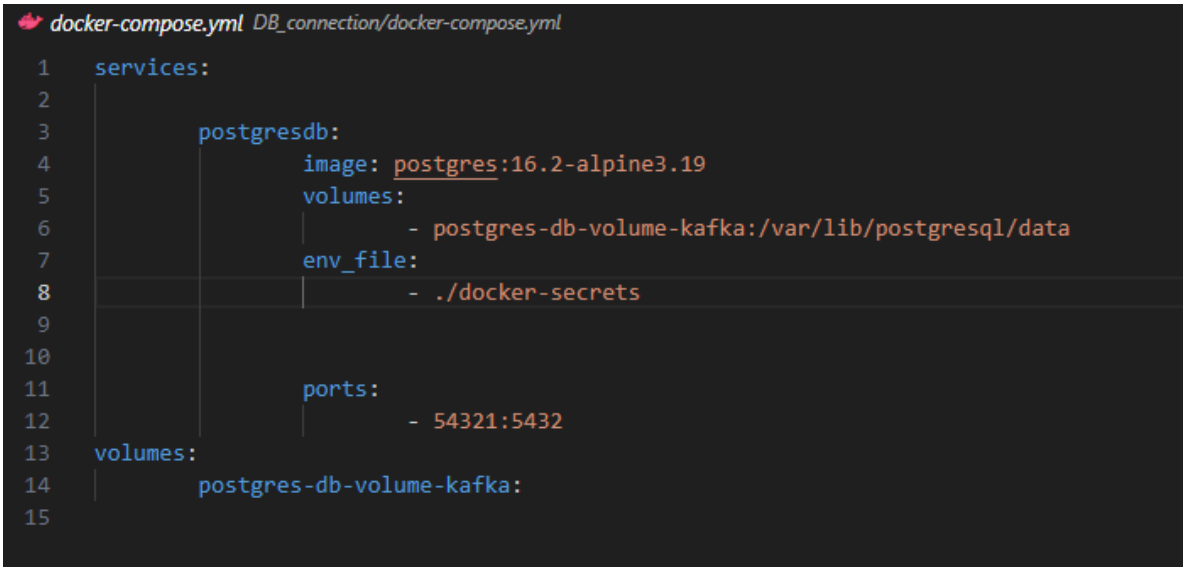


- **DB_connection**:
    - **postgres-db-volume-kafka**: Carpeta para persistir los datos de PostgreSQL y Kafka.
    - **init.py**: Archivo que indica que el directorio debe ser tratado como un módulo de Python.

    - **docker-compose.yml**: Archivo de configuración de Docker Compose para levantar la base de datos PostgreSQL.
    - **docker-secrets**: Carpeta que contiene archivos de contraseñas y otras configuraciones sensibles.
    - **Pysqlconnect.py**: Script de Python para manejar la conexión con PostgreSQL.
- **Kafka**:
    - **app.py**: Script principal de la aplicación Kafka.
    - **docker-compose.yml**: Archivo de configuración de Docker Compose para levantar el entorno de Kafka.
    - **kafka_consumer.py**: Script de Python para consumir mensajes de Kafka.
    - **kafka_producer.py**: Script de Python para producir mensajes en Kafka.
- **Model**:
    - **EDA.ipynb**: Notebook de Jupyter para el análisis exploratorio de datos.
    - **Model.ipynb**: Notebook de Jupyter para el entrenamiento del modelo.
    - **testdb.ipynb**: Notebook de Jupyter para probar la base de datos.
- **venv**:
    - Carpeta que contiene el entorno virtual de Python.
- **.env**:
    - Archivo que contiene las variables de entorno necesarias para la configuración del proyecto.
- **.gitignore**:
    - Archivo que especifica qué archivos o directorios deben ser ignorados por Git.
- **LICENSE**:
    - Archivo que contiene la licencia del proyecto.
- **README.md**:

    - Archivo de documentación que proporciona una descripción general del proyecto, instrucciones de instalación y uso.
- **requirements.txt**:
    - Archivo que contiene una lista de dependencias de Python necesarias para el proyecto.

Configuración de archivo docker Compose de base de datos



Este archivo **docker-compose.yml** define un servicio **postgresdb** usando la imagen **postgres:16.2-alpine3.19**. Se configura un volumen para persistencia de datos (**postgres-db-volume-kafka**), y se especifica un archivo de variables de entorno (**docker-secrets**) para manejar las credenciales. El puerto del contenedor PostgreSQL se expone en el puerto 54321 del host.

Contenido relevante del archivo docker-secrets:

- Define la variable **POSTGRES_PASSWORD** con la contraseña del usuario **postgres**.

Correr el contenedor con el comando **sudo docker compose up**

Ingresamos al contenedor y creamos la base de datos en este caso workshop_03

sudo docker exec -it db_connection-postgresdb-1 /bin/bash

psql -U postgres

create database workshop_03;

```
joan@ROGZEPHYRUSM:~/Desktop/etl_workshop_3/DB_connection$ sudo docker compose ps
[sudo] password for joan:
WARN[0000] The "rr" variable is not set. Defaulting to a blank string.
NAME                        IMAGE                     COMMAND                  SERVICE      CREATED       STATUS          PORTS
db_connection-postgresdb-1   postgres:16.2-alpine3.19   "docker-entrypoint.s…"   postgresdb   9 hours ago   Up 33 seconds   0.0.0.0:543
21->5432/tcp, :::54321->5432/tcp
joan@ROGZEPHYRUSM:~/Desktop/etl_workshop_3/DB_connection$ sudo docker exec -it db_connection-postgresdb-1 /bin/bash
f195c3843655:/# psql -U postgres
psql (16.2)
Type "help" for help.

postgres=# create database workshop_3
```

Vemos las tablas

```
postgres=# \l
                                                 List of databases
    Name     |  Owner   | Encoding | Locale Provider |  Collate   |   Ctype    | ICU Locale | ICU Rules |   Access privileges
-------------+----------+----------+-----------------+------------+------------+------------+-----------+-----------------------
 postgres    | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           |
 template0   | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           | =c/postgres          +
             |          |          |                 |            |            |            |           | postgres=CTc/postgres
 template1   | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           | =c/postgres          +
             |          |          |                 |            |            |            |           | postgres=CTc/postgres
 workshop_03 | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |           |
(4 rows)

postgres=#
```

EDA

EDA

In [1]:
```
#preprocess stage
#import libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```
#read all datafiles
data_2015 = pd.read_csv("../Data/2015.csv")
data_2016 = pd.read_csv("../Data/2016.csv")
data_2017 = pd.read_csv("../Data/2017.csv")
data_2018 = pd.read_csv("../Data/2018.csv")
data_2019 = pd.read_csv("../Data/2019.csv")
```

[112…
```
data_2015.name_ = "data_2015"
data_2016.name_ = "data_2016"
data_2017.name_ = "data_2017"
data_2018.name_ = "data_2018"
data_2019.name_ = "data_2019"
```

[113…
```
#create a list with the dataframes to iterate after that
datalist = [data_2015, data_2016, data_2017, data_2018, data_2019]
```

Importamos librerías y cargamos los datos

find for column amount

```
ycounter =  0
for i in datalist:
    print(f"{2015+ycounter} {i.shape}")
    ycounter +=1
```

```
2015 (158, 12)
2016 (157, 13)
2017 (155, 12)
2018 (156, 9)
2019 (156, 9)
```

We can see that the diferents files have diferent columns amounts

```
ycounter = 0
for i in datalist:
    print(f"{2015+ycounter} {i.columns}")
    ycounter +=1
```

```
2015 Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
2016 Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
```

Buscamos por inconsistencias en los datos nombres de columnas coincidencias etc

First We will check the state of the data in each dataset

[123…

```
data_2015.info()
```

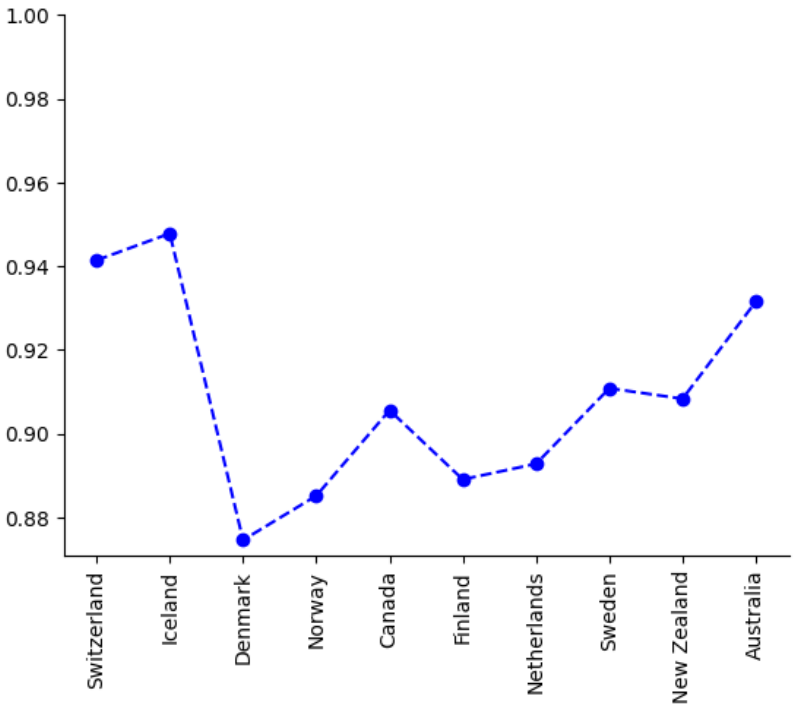
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Country                        158 non-null    object
 1   Region                         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
```

Buscamos valores nulos y vemos los tipos de datos

In [3]:

```
best_ten = data_2015[data_2015["Happiness Rank"] < 11]
fig, ax = plt.subplots()
ax.plot(best_ten.Country, best_ten['Health (Life Expectancy)'], "bo--")
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
ax.set_xticklabels(best_ten.Country, rotation=90)
ax.set_ylim(top=1)
plt.show()
```

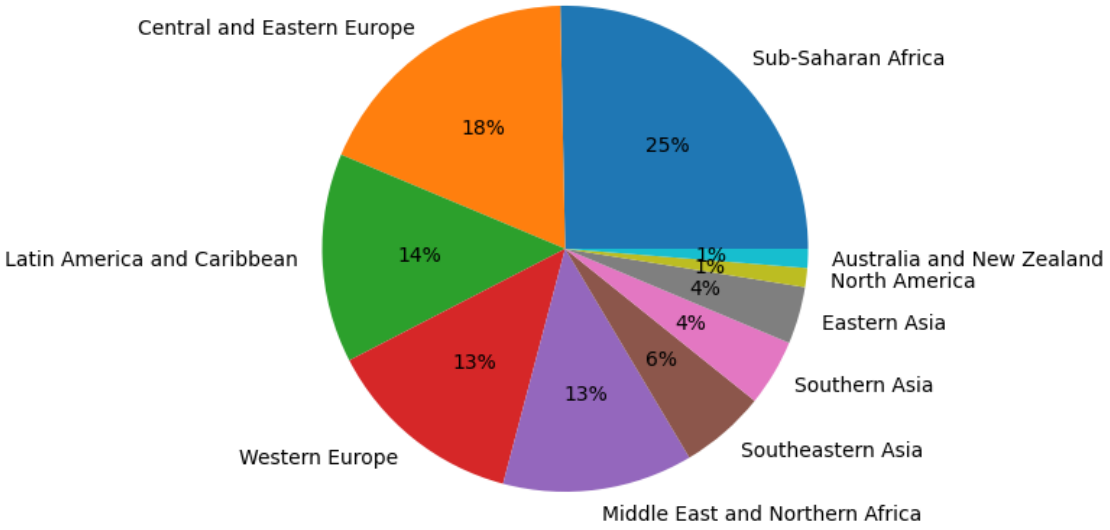
```
/tmp/ipykernel_1336/3926700399.py:6: UserWarning: FixedFormatter should only be used together with FixedLoc
tor
  ax.set_xticklabels(best_ten.Country, rotation=90)
```



Graficamos para ver los países con mayor puntuación casi todos están en Europa

[7]:

```
fig, ax = plt.subplots()
ax.pie(data_2015.Region.value_counts(),labels=data_2015.Region.value_counts().keys(),autopct=lambda p: '{:
ax.axis("equal")
#ax.legend(loc="best")
plt.show()
```



Proporción de continentes en el dataset

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	0.143422	0.237296
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.120034	0.126685
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.061675	0.150553
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.107220	0.216130
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.180255	0.309883
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.551910	0.795880

we can see that

- happiness rank have the values min and max in the correct range
- the hapiness score don't have outliers values like over 10 and lees than 1
- in GDP we have some weird values like 0
- in life ecpectancy have weird values like 0
- freedom weird min value
- trust weird min value
- Generosity weird min value

Observando la descripcion del dataset en sus valores numéricos vemos posibles datos erróneos

In [125…

```
data_2015.describe(include="object")
# we see that the most amount of registers are in the region of Sub-Saharan Africa
```

Out[125…

	Country	Region
count	158	158
unique	158	10
top	Switzerland	Sub-Saharan Africa
freq	1	40

En la descripción de datos categóricos observamos que todo se ve bien por aquí

[126…]
```
#check for the register with have the weird values
data_2015[(data_2015["Economy (GDP per Capita)"]==0) |
          (data_2015["Family"]==0) |
          (data_2015["Health (Life Expectancy)"]==0) |
          (data_2015["Freedom"]==0) |
          (data_2015["Trust (Government Corruption)"]==0) |
          (data_2017["Generosity"]==0)]
```

[126…]

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Tru (Governme Corruptio
73	Indonesia	Southeastern Asia	74	5.399	0.02596	0.82827	1.08708	0.63793	0.46611	0.000
86	Serbia	Central and Eastern Europe	87	5.123	0.04864	0.92053	1.00964	0.74836	0.20107	0.026
111	Iraq	Middle East and Northern Africa	112	4.677	0.05232	0.98549	0.81889	0.60237	0.00000	0.137
119	Congo (Kinshasa)	Sub-Saharan Africa	120	4.517	0.03680	0.00000	1.00120	0.09806	0.22605	0.076
122	Sierra Leone	Sub-Saharan Africa	123	4.507	0.07068	0.33024	0.95571	0.00000	0.40840	0.087
147	Central African Republic	Sub-Saharan Africa	148	3.678	0.06112	0.07850	0.00000	0.06699	0.48879	0.082

Chekeamos los posibles outliers vemos que csi todos esta n en africa y puede ser la dificultad de obtener los datos lo que genera esto

[128…]
```
data_2015["Happiness Rank"].duplicated().value_counts()
```

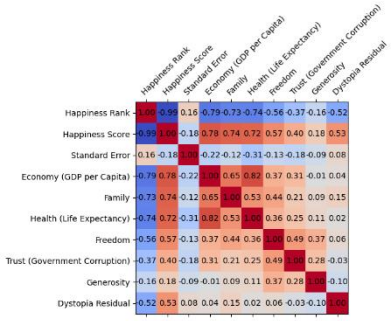
[128…]
```
Happiness Rank
False    157
True       1
Name: count, dtype: int64
```

Buscamos duplicados

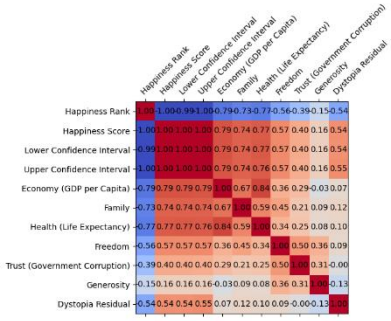
```
data_2015[data_2015["Country"].duplicated()]
```

Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Genero

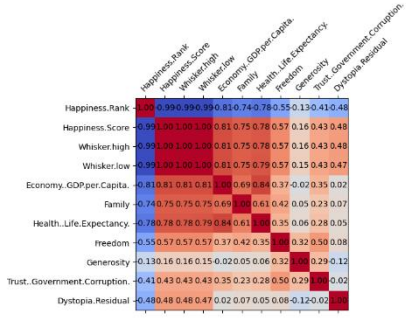
DF 2015



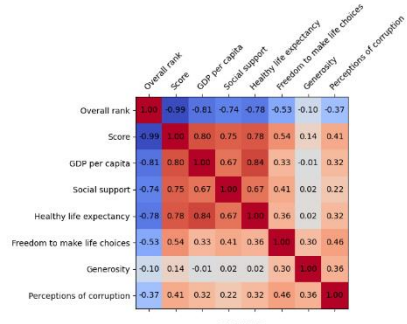
DF 2016



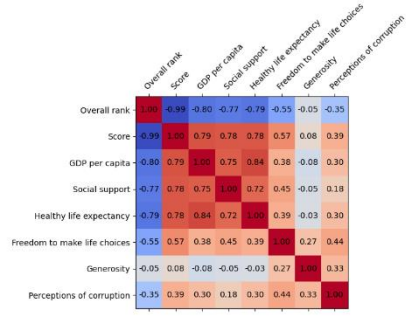
DF 2017



DF 2018



DF 2019



Buscamos por medio del mapa de correlación las variables que nos podrían funcionar para un modelo sencillo

Se terminan eligiendo las siguientes características

Looking with the correlation map we see that the columns for the model to predict the score are the next ones

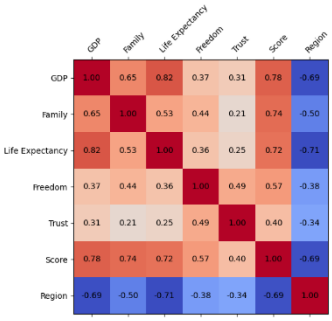
- GDP
- Family
- Life Expectancy
- Freedom
- Trust

```
#Transfotmation of the names of columns
# Found the diferents forms to name this columns

columnsdict = {"GDP":["Economy (GDP per Capita)", "Economy..GDP.per.Capita.", "GDP per capita"],
               "Family":["Family", "Social support"],
               "Life Expectancy":["Health (Life Expectancy)", "Health..Life.Expectancy.", "Healthy life e
               "Freedom":["Freedom", "Freedom to make life choices"],
               "Trust": ["Trust (Government Corruption)", "Trust..Government.Corruption.", "Perceptions o
               "Rank":["Overall rank", "Happiness.Rank", "Happiness Rank"],
               "Country":["Country", "Country or region"],
               "Score":["Happiness Score", "Happiness.Score", "Score"]}
```

```
# function to change the names

def change_names(df:pd.DataFrame) ->pd.DataFrame:
    keys = df.keys()
    gdp = ""
    family = ""
    life_expectancy = ""
    freedom = ""
    trust = ""
    rank = ""
    country = ""
    score = ""
    for i in keys:
        if i in columnsdict["GDP"]:
            gdp = i
            continue
        if i in columnsdict["Family"]:
            family = i
            continue
        if i in columnsdict["Life Expectancy"]:
            life_expectancy = i
        if i in columnsdict["Freedom"]:
            freedom = i
            continue
        if i in columnsdict["Trust"]:
            trust = i
            continue
        if i in columnsdict["Rank"]:
            rank = i
            continue
```

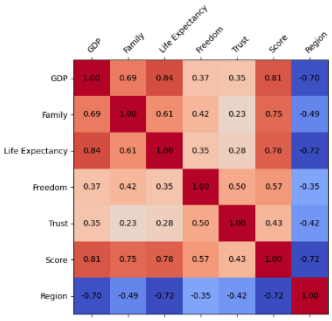
se estandarizan las columnas

DF 2015

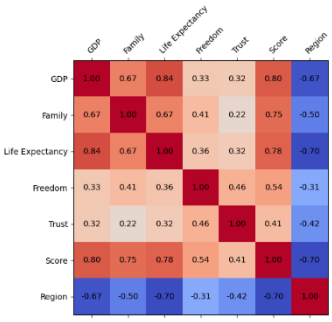


DF 2016

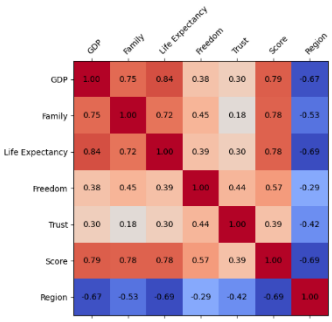
DF 2017



DF 2018



DF 2019



Se ven los nuevos mapas de correlación y todos tienen sentido

```
#save the data for the model training
data_2015.to_csv(f"../Data/Train/data_2015.csv", index=False)
data_2016.to_csv(f"../Data/Train/data_2016.csv", index=False)
data_2017.to_csv(f"../Data/Train/data_2017.csv", index=False)
data_2018.to_csv(f"../Data/Train/data_2018.csv", index=False)
data_2019.to_csv(f"../Data/Train/data_2019.csv", index=False)
```

```
df_concatenate = pd.concat([data_2015, data_2016, data_2017, data_2018, data_2019])
df_concatenate.to_csv("../Data/Train/df.csv", index=False)
```

Se guardan los datos para el entrenamiento

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import joblib
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
df = pd.read_csv("../Data/Train/df.csv")
```

```
df.head()
```

Se cargan los datos

```
X = df[['GDP', 'Family', 'Life Expectancy', 'Freedom', 'Trust', 'Region']]
y = df['Score']


print(X.isnull().sum())
print(y.isnull().sum())


X.fillna(X.mean(), inplace=True)
y.fillna(y.mean(), inplace=True)
```

```
GDP                0
Family             0
Life Expectancy    0
Freedom            0
Trust              1
Region            16
dtype: int64
0
```

```
/tmp/ipykernel_18429/2850157958.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy
  X.fillna(X.mean(), inplace=True)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
model = LinearRegression()
model.fit(X_train, y_train)
```

se crean los diferentes sub datasets para el entrenamiento

```
model = LinearRegression()
model.fit(X_train, y_train)
```

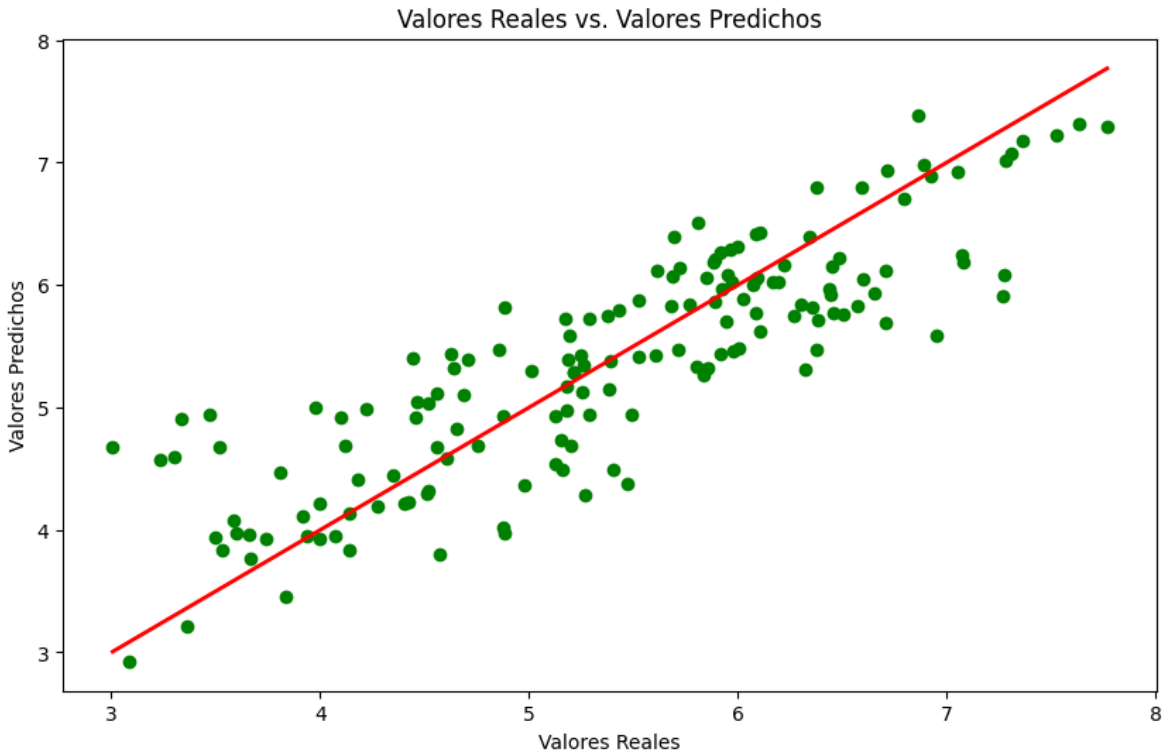
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
y_pred = model.predict(X_test)

# Calcular el error cuadrático medio (MSE) y el coeficiente de determinación (R^2)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MSE:", mse)
print("R^2:", r2)
```

```
MSE: 0.320253672962502
R^2: 0.7367153832847613
```

Se entrena y observan las métricas 73%



Con nuevas características

with Age and Region

```
data_2015 = pd.read_csv("../Data/Train/data_2015.csv")
data_2016 = pd.read_csv("../Data/Train/data_2016.csv")
data_2017 = pd.read_csv("../Data/Train/data_2017.csv")
data_2018 = pd.read_csv("../Data/Train/data_2018.csv")
data_2019 = pd.read_csv("../Data/Train/data_2019.csv")
```

```
data_2015["Age"] = 2015
data_2016["Age"] = 2016
data_2018["Age"] = 2018
data_2019["Age"] = 2019
```

```
data_2015.head()
```

🤓

```
X = df_ages[['GDP', 'Family', 'Life Expectancy', 'Freedom', 'Trust', 'Region', 'Age']]
y = df_ages['Score']

# Check for and handle any NaN values
if X.isnull().any().any() or y.isnull().any():
    X.fillna(X.mean(), inplace=True)
    y.fillna(y.mean(), inplace=True)
```

```
tmp/ipykernel_18429/3622275699.py:6: SettingWithCopyWarning:
 value is trying to be set on a copy of a slice from a DataFrame

ee the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
#returning-a-view-versus-a-copy
 X.fillna(X.mean(), inplace=True)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=45)
```

```
model = LinearRegression()
model.fit(X_train, y_train)
```

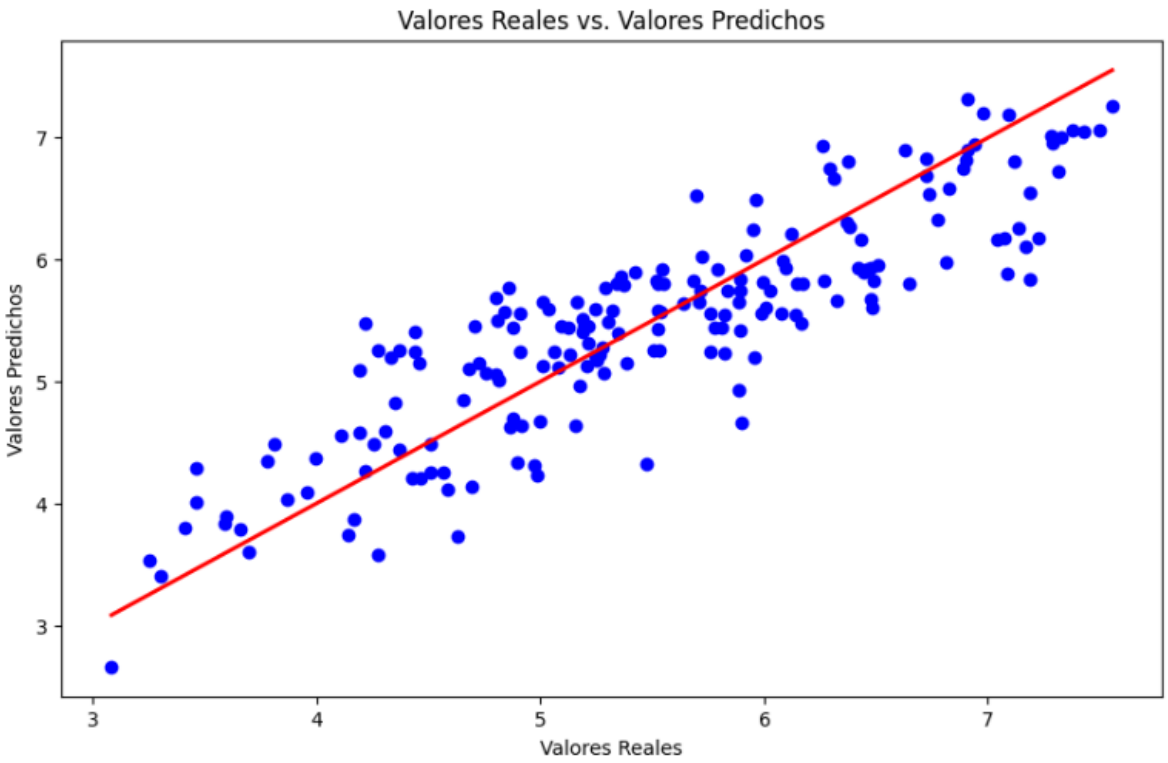
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
y_pred = model.predict(X_test)

# Calcular el Error Cuadrático Medio (MSE) y los valores de R-cuadrado (R2)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Error Cuadrático Medio:", mse)
print("R-cuadrado:", r2)
```

```
rror Cuadrático Medio: 0.25310105796505267
-cuadrado: 0.7631518188635866
```

Se observa mejoría


Valores Reales vs. Valores Predichos
Valores Predichos
Valores Reales
3
4
5
6
7

Second Age and Region T

27]:
```python
df_T = df_ages.copy()
```

41]:
```python
Region_number = {'Western Europe': 0,
 'North America': 1,
 'Australia and New Zealand': 2,
 'Middle East and Northern Africa': 3,
 'Latin America and Caribbean': 4,
 'Southeastern Asia': 5,
 'Central and Eastern Europe': 6,
 'Eastern Asia': 7,
 'Sub-Saharan Africa': 8,
 'Southern Asia': 9}
```

35]:
```python
def Create_new_years_columns(df):
    df["2015"] = (df["Age"] == 2015).astype(int)
    df["2016"] = (df["Age"] == 2016).astype(int)
    df["2017"] = (df["Age"] == 2017).astype(int)
    df["2018"] = (df["Age"] == 2018).astype(int)
    df["2019"] = (df["Age"] == 2019).astype(int)
    df.drop(columns=["Age"], inplace=True)
    return df
```

42]:
```python
def convert_region_to_columns(df):
    region_dict = Region_number
    # Invertir el diccionario para hacer el mapeo de número a nombre
    number_to_region = {v: k for k, v in region_dict.items()}

    # Crear una nueva columna 'Region_name' con los nombres de las regiones
    df['Region_name'] = df['Region'].map(number_to_region)

    # Crear columnas booleanas para cada región
    for region in region_dict.keys():
        df[region] = (df['Region_name'] == region).astype(int)

    # Eliminar las columnas 'Region' y 'Region_name'
    df.drop(columns=['Region', 'Region_name'], inplace=True)
```

con aun mas características

```
      dtype='object')
```

n [96]:

```
X = df_T[['GDP', 'Family', 'Life Expectancy', 'Freedom', 'Trust', '2015', '2016', '2017', '2018', '2019'
          'Western Europe', 'North America', 'Australia and New Zealand', 'Middle East and Northern Afri
          'Latin America and Caribbean', 'Southeastern Asia', 'Central and Eastern Europe', 'Eastern Asi
          'Sub-Saharan Africa', 'Southern Asia']]
y = df_T['Score']

# Verificar y manejar valores NaN
if X.isnull().any().any() or y.isnull().any():
    X.fillna(X.mean(), inplace=True)
    y.fillna(y.mean(), inplace=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Error Cuadrático Medio:", mse)
print("R-cuadrado:", r2)
```

```
Error Cuadrático Medio: 0.23834791323906096
R-cuadrado: 0.8087997749353245
```

```
/tmp/ipykernel_1338/1160147831.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy
  X.fillna(X.mean(), inplace=True)
```

se observa una mejora aun mayor

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linewidth=2)
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Valores Reales vs. Valores Predichos')
plt.show()
```