

In [1]:

```
1 #from google.colab import drive
2 #drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [2]: 1 !pip install ydata-profiling
```

Collecting ydata-profiling

```
  Downloading ydata_profiling-4.5.1-py2.py3-none-any.whl (357 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 357.3/357.3 kB 3.8 MB/s eta 0:0
0:00
Requirement already satisfied: scipy<1.12,>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.11.3)
Requirement already satisfied: pandas!=1.4.0,<2.1,>1.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.5.3)
Requirement already satisfied: matplotlib<4,>=3.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (3.7.1)
Requirement already satisfied: pydantic<2,>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.10.13)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (6.0.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (3.1.2)
Collecting visions[type_image_path]==0.7.5 (from ydata-profiling)
  Downloading visions-0.7.5-py3-none-any.whl (102 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━ 102.7/102.7 kB 9.1 MB/s eta 0:0
0:00
Requirement already satisfied: numpy<1.24,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.23.5)
Collecting htmlmin==0.1.12 (from ydata-profiling)
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Collecting phik<0.13,>=0.11.1 (from ydata-profiling)
  Downloading phik-0.12.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (679 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━ 679.5/679.5 kB 8.6 MB/s eta 0:0
0:00
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (2.31.0)
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.66.1)
Requirement already satisfied: seaborn<0.13,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.12.2)
Collecting multimethod<2,>=1.4 (from ydata-profiling)
  Downloading multimethod-1.10-py3-none-any.whl (9.9 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.14.0)
Collecting typeguard<3,>=2.13.2 (from ydata-profiling)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Collecting imagehash==4.3.1 (from ydata-profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 9.4 MB/s eta 0:0
0:00
Requirement already satisfied: wordcloud>=1.9.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.9.2)
Collecting dacite>=1.8 (from ydata-profiling)
  Downloading dacite-1.8.1-py3-none-any.whl (14 kB)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling) (1.4.1)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling) (9.4.0)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]==0.7.5->ydata-profiling) (23.1.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-
```

```
st-packages (from visions[type_image_path]==0.7.5->ydata-profiling) (3.1)
Collecting tangled-up-in-unicode>=0.0.4 (from visions[type_image_path]==0.7.
5->ydata-profiling)
    Downloading tangled_up_in_unicode-0.2.0-py3-none-any.whl (4.7 MB)
    ━━━━━━━━━━━━━━━━ 4.7/4.7 MB 17.4 MB/s eta 0:00:
  00
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/
dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling) (2.1.3)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib<4,>=3.2->ydata-profiling) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dis
t-packages (from matplotlib<4,>=3.2->ydata-profiling) (0.12.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib<4,>=3.2->ydata-profiling) (4.43.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib<4,>=3.2->ydata-profiling) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/
dist-packages (from matplotlib<4,>=3.2->ydata-profiling) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib<4,>=3.2->ydata-profiling) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python
3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dis
t-packages (from pandas!=1.4.0,<2.1,>1.1->ydata-profiling) (2023.3.post1)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/d
ist-packages (from phik<0.13,>=0.11.1->ydata-profiling) (1.3.2)
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/py
thon3.10/dist-packages (from pydantic<2,>=1.8.1->ydata-profiling) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from requests<3,>=2.24.0->ydata-profiling) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
10/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (2.0.6)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
10/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (2023.7.22)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dis
t-packages (from statsmodels<1,>=0.13.2->ydata-profiling) (0.5.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-package
s (from patsy>=0.5.2->statsmodels<1,>=0.13.2->ydata-profiling) (1.16.0)
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) ... done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=2
7081 sha256=60306c0a5a11f3e3fe241486e2a79953e09aa985f543cc0743e03f233a87727e
  Stored in directory: /root/.cache/pip/wheels/dd/91/29/a79cecb328d01739e640
17b6fb9a1ab9d8cb1853098ec5966d
Successfully built htmlmin
Installing collected packages: htmlmin, typeguard, tangled-up-in-unicode, mu
ltimethod, dacite, imagehash, visions, phik, ydata-profiling
Successfully installed dacite-1.8.1 htmlmin-0.1.12 imagehash-4.3.1 multimeth
od-1.10 phik-0.12.3 tangled-up-in-unicode-0.2.0 typeguard-2.13.3 visions-0.
7.5 ydata-profiling-4.5.1
```

0. Importar librerías

In [356]:

```
1 #Manejo de datos
2 import pandas as pd
3 import numpy as np
4 import scipy.stats as stats
5
6 #Analisis profundo de datos
7 from ydata_profiling import ProfileReport
8
9 # Gráficas
10 import matplotlib.pyplot as plt
11 import seaborn as sns
12
13 #Entrenamiento del modelo
14 from sklearn.model_selection import train_test_split
15 from sklearn.impute import SimpleImputer
16 from sklearn.linear_model import Lasso, Ridge, LinearRegression
17 from sklearn.pipeline import Pipeline
18 from sklearn.preprocessing import OneHotEncoder, StandardScaler, Polynomial
19 from sklearn.metrics import mean_squared_error, r2_score
20 from sklearn.compose import ColumnTransformer, make_column_selector
21 from sklearn.base import BaseEstimator, TransformerMixin
22 from sklearn.model_selection import GridSearchCV
23
24 # Encontrar palabras similares
25 from difflib import SequenceMatcher
```

1. Cargar Datos

In [233]:

```
1 #data = pd.read_csv('/content/drive/MyDrive/maestria_mine/ciencia_de_dato
2 data = pd.read_csv('data/gapminder_final.csv')
```

2. Perfilamiento y calidad de datos

In [9]: 1 data.sample(5)

Out[9]:

	country	incomeperperson	alccconsumption	armedforcesrate	breastcancerper100th	co2em
99	Peru	3180.430612	6.53	1.435633	35.1	1.1462
171	Laos	554.879840	6.99	4.301576	10.9	2.1332
78	Slovak Republic	8445.526689	13.31	0.628578	48.0	5.9067
68	Kuwait	Nan	0.10	1.609548	31.8	1.7127
140	Liberia	155.033231	5.07	0.129953	18.8	3.8991



In []: 1 ProfileReport(data)

```
Summarize dataset: 0% | 0/5 [00:00<?, ?it/s]
Generate report structure: 0% | 0/1 [00:00<?, ?it/s]
Render HTML: 0% | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	16
Number of observations	178
Missing cells	198
Missing cells (%)	7.0%
Duplicate rows	12
Duplicate rows (%)	6.7%
Total size in memory	22.4 KiB
Average record size in memory	128.7 B

Variable types

Text	1
Numeric	15

Alerts

Dataset has 12 (6.7%) duplicate rows	Duplicates
incomeperperson is highly overall correlated with breastcancerper100th and <u>6 other fields</u>	High correlation
(breastcancerper100th, co2emissions, internetuserate,	

Out[56]:

2.1 Dimensiones de los datos

In [234]: 1 data.shape

Out[234]: (178, 16)

Se ve que hay 178 registros para hacer la regresión, los cuales son relativamente pocos datos en comparación con la cantidad de columnas (10%). Por lo que se tiene que tener cuidado en temas de complejidad del modelo.

2.2 Reporte de Nulos

In [238]: 1 data.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 163 entries, 0 to 177
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   country          163 non-null    object  
 1   incomeperperson  163 non-null    float64 
 2   alcconsumption   163 non-null    float64 
 3   armedforcesrate  159 non-null    float64 
 4   breastcancerper100th  162 non-null    float64 
 5   co2emissions     161 non-null    float64 
 6   femaleemployrate 162 non-null    float64 
 7   hivrate          143 non-null    float64 
 8   internetuserate 161 non-null    float64 
 9   lifeexpectancy   163 non-null    float64 
 10  oilperperson     60 non-null    float64 
 11  polityscore      153 non-null    float64 
 12  relectricperperson 128 non-null    float64 
 13  suicideper100th  163 non-null    float64 
 14  employrate       162 non-null    float64 
 15  urbanrate        163 non-null    float64 
dtypes: float64(15), object(1)
memory usage: 21.6+ KB
```

Vemos que existen varias columnas que cuentan con nulos, la mayoría de estas un porcentaje bastante bajo de los datos. A excepción de hivrate, oilperperson y relectricperperson. Además que existen 14 variables numérica y una categórica. Directamente se eliminan los nulos con la variable dependiente, ya que no es posible imputarlos porque sesgaría el modelo.

In [236]: 1 data = data.dropna(subset=['incomeperperson'])

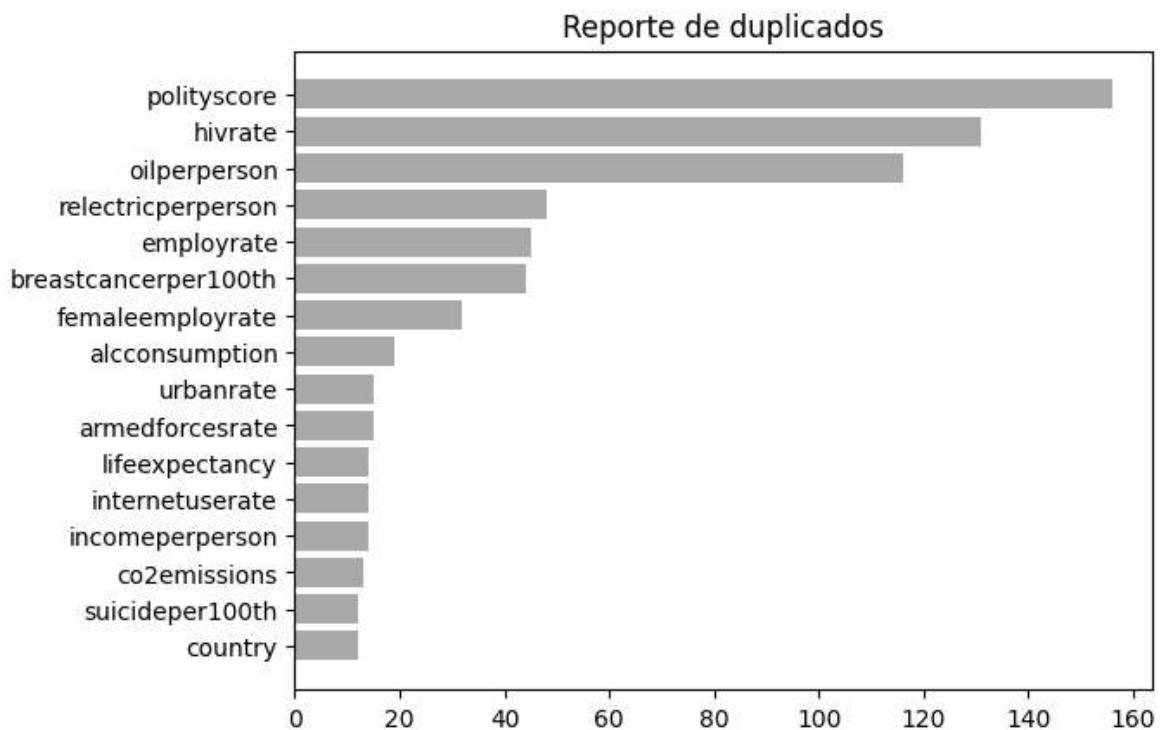
2.3 Reporte de Duplicados

In [8]:

```

1 columns = []
2 duplicates = []
3 for c in data:
4     columns.append(c)
5     duplicates.append(data[c].duplicated().sum())
6
7 df = pd.DataFrame({'columns':columns,'duplicates':duplicates}).sort_values
8 plt.title('Reporte de duplicados')
9 plt.barrh(df['columns'].to_list(), df['duplicates'].to_list(), color='darkred')
10 plt.show()

```



Encontramos 12 duplicados en la columna country que es categórica. En las demás columnas del dataset que son numéricas, los duplicados suelen ser menos importantes.

In [239]:

```
1 data = data.drop_duplicates(['country'])
```

2.4 Elección de 5 indicadores más importantes

La variable que queremos predecir es el PIB, por lo tanto hay que entender que las variables que se escogen deben tener influencia directa en la economía y productividad de un país. Por esto, escogemos las siguientes:

- employrate, la tasa de empleo de un país evidentemente tiene una influencia directa en la cantidad de ingreso que genera la clase trabajadora, entre mayor sea mayor productividad habrá en las personas de un país. Todo esto, tiene también influencia en los

problemas sanitarios y sociales, dado que un país con alto porcentaje de empleo formal es un país en el que las personas podrán pagar la salud y asegurar su retiro.

- polityscore, además de la productividad de un país es muy importante la estabilidad política y que exista un sistema democrático que permita a las personas salir adelante y que disminuya lo máximo posible la corrupción para que haya la mínima malversación de bienes posible.
- relectricperperson, es una variable que tiene en cuenta el consumo eléctrico por persona en un país lo cual es una medida del desarrollo, ya que a mayor consumo energético quiere decir que tienen más electrodomésticos y mayor desarrollo tecnológico en los hogares, lo cual evidentemente se desprende de una buena situación económica del país.
- urbanrate, Esto nos brinda una medida de si el país tiene una mayor población campesina o urbana, por lo general los países más desarrollados, suelen tener urbes con millones de personas y una minoría en los campos. Los países con escasos recursos suelen tener una población campesina dominante y sus poblados no suelen ser considerados urbes por su falta de servicios básicos.
- co2emissions, se considera que esta variable es sumamente importante dado que los países con mayores recursos suelen tener una cantidad de empresas muy grande lo cual se desprende en emisiones de co2 bastante altas.

Por último, tenemos la variable incomeperperson que es la que vamos a predecir, en este

2.5 Análisis univariado

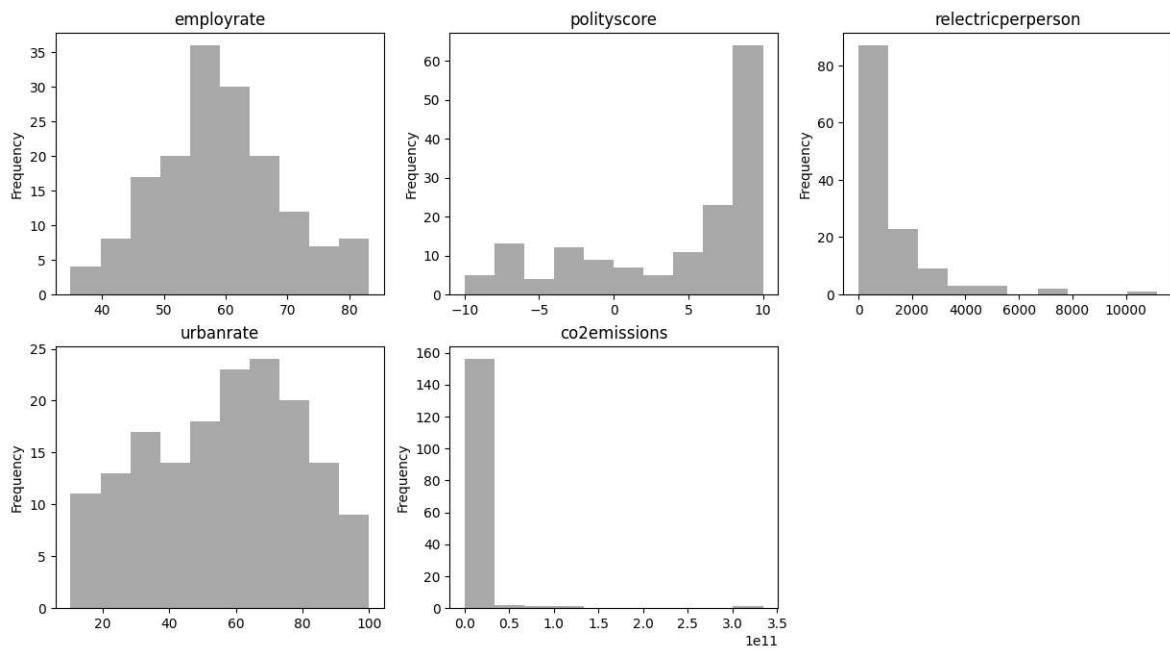
2.5.1 Histogramas de variables

In [240]:

```

1 columns = ['employrate', 'polityscore', 'relectricperperson', 'urbanrate',
2 plt.figure(figsize=(15,12))
3 for i in range(len(columns)):
4     plt.subplot(3,3,i+1)
5     plt.title(columns[i])
6     data[columns[i]].plot.hist(color='darkgrey')
7 plt.show()

```



A partir de los histogramas realizados, se puede observar que las variables employrate y urbanrate tienen una distribución aproximadamente normal, mientras que las variables co2emissions, polityscore y relectricperperson, tienen un distribución con el volumen de datos dirigido a los extremos de los valores. co2emissions y relectricperperson hacia los valores mínimos y polityscore hacia los valores máximos. Esto también podría significar que existen algunos outliers.

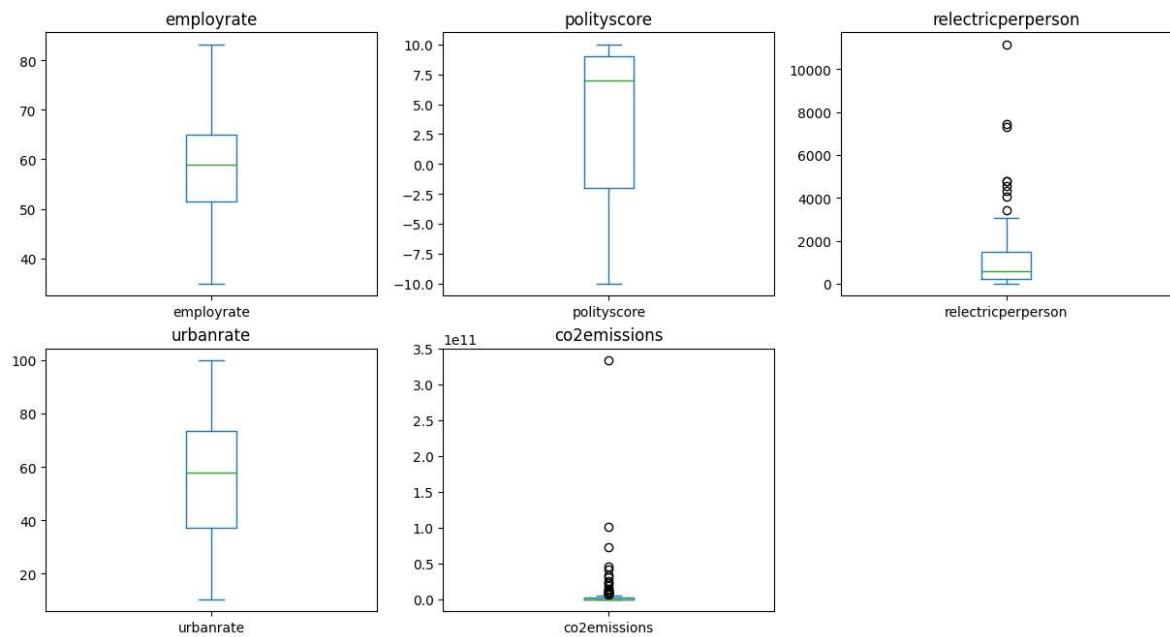
2.5.2 Boxplot

In [241]:

```

1 columns = ['employrate', 'polityscore', 'relectricperperson', 'urbanrate',
2 plt.figure(figsize=(15,12))
3 for i in range(len(columns)):
4     plt.subplot(3,3,i+1)
5     plt.title(columns[i])
6     data[columns[i]].plot.box()
7 plt.show()

```



De estos gráficos se destaca la asimetría de la distribución de los datos para las variables urbanrate y polityscore hacia los valores máximos y la aparición de outliers para las variables co2emissions y relectricperperson. Estos outliers no se consideran necesariamente descartables, dado que sí tiene mucho sentido que las potencias mundiales tengan una cantidad muy superior de emisiones por co2 y consumo eléctrico por persona. Se considera que se pueden dejar en el modelo para que este aprenda la gran variabilidad que hay en estas características para determinados países que son potencias en el mundo.

2.5.3 Estadísticas descriptivas

In [242]: 1 data[columns].describe()

Out[242]:

	employrate	polityscore	relectricperperson	urbanrate	co2emissions
count	162.000000	153.000000	128.000000	163.000000	1.610000e+02
mean	59.074691	3.849673	1144.245457	56.015215	6.156077e+09
std	10.364735	6.226821	1596.990968	22.600016	2.855923e+10
min	34.900002	-10.000000	0.000000	10.400000	8.506667e+05
25%	51.575001	-2.000000	219.736499	37.090000	7.894333e+07
50%	58.850000	7.000000	597.136436	57.940000	2.771707e+08
75%	65.000000	9.000000	1491.145249	73.470000	2.406741e+09
max	83.199997	10.000000	11154.755030	100.000000	3.340000e+11

Se deduce de estas estadísticas que los indicadores polityscore y relectricperperson presentan una gran variabilidad en los datos dado que tienen una desviación estándar considerablemente mayor a su promedio. Lo cual posiblemente es causado por los outliers encontrados anteriormente.

2.6 Análisis multivariado

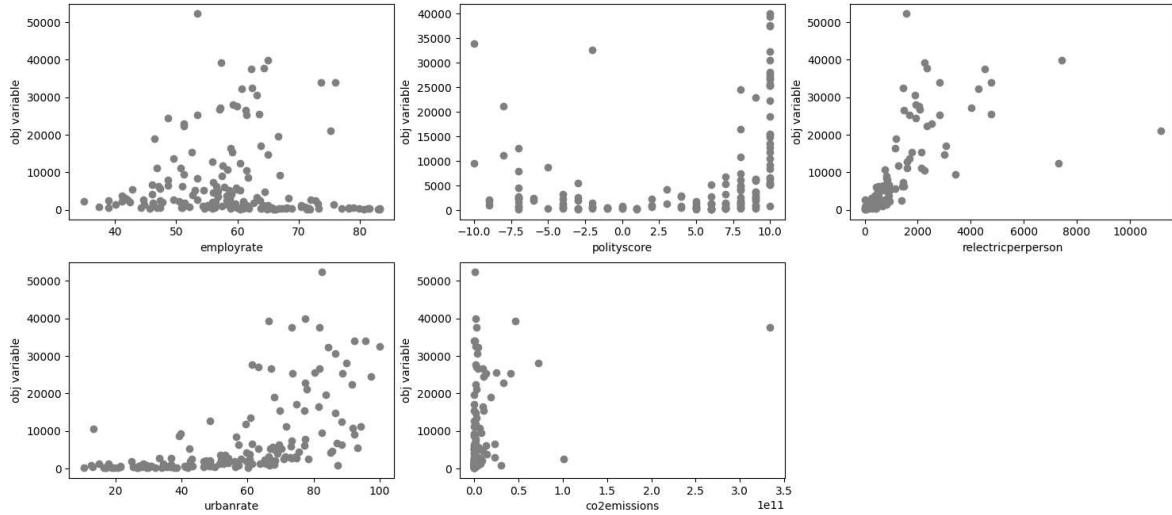
2.6.1 Indicadores elegidos contra variable objetivo

In [243]:

```

1 columns = ['employrate', 'polityscore', 'relectricperperson', 'urbanrate',
2 plt.figure(figsize=(18,12))
3 for i in range(len(columns)):
4     plt.subplot(3,3,i+1)
5     #plt.title(columns[i])
6     plt.xlabel(columns[i])
7     plt.ylabel('obj variable')
8     plt.scatter(data[columns[i]], data['incomeperperson'], color ='grey')
9 plt.show()

```



Se destaca que no parece haber mucha colinealidad entre estas variables lo cual es bueno para la regresión.

2.6.2 Relación entre indicadores y regiones

In [244]:

```

1 #continent_countries = pd.read_csv('/content/drive/MyDrive/maestria_mine/
2 continent_countries = pd.read_csv('data/continents_countries.csv')

```

Buscamos los continentes y subregiones para cada país por medio del website <https://www.gapminder.org/fw/four-regions/> (<https://www.gapminder.org/fw/four-regions/>). Además obtenemos de esta misma web la clasificación de ingresos.

In [245]:

```

1 def find_continent(x):
2     country_list = continent_countries['name'].drop_duplicates()
3     corrected_name = None
4     name = []
5     similarity = []
6     for country in country_list:
7         name.append(country)
8         similarity.append(SequenceMatcher(None, x, country).ratio())
9     name_similarity = pd.DataFrame({'name':name, 'similarity':similarity})
10    max_similarity = name_similarity['similarity'].max()
11    corrected_name = name_similarity[name_similarity['similarity'] == max_similar]
12    return continent_countries[continent_countries['name'] == corrected_name]
13
14 def find_subregion(x):
15     country_list = continent_countries['name'].drop_duplicates()
16     corrected_name = None
17     name = []
18     similarity = []
19     for country in country_list:
20         name.append(country)
21         similarity.append(SequenceMatcher(None, x, country).ratio())
22     name_similarity = pd.DataFrame({'name':name, 'similarity':similarity})
23     max_similarity = name_similarity['similarity'].max()
24     corrected_name = name_similarity[name_similarity['similarity'] == max_similar]
25     return continent_countries[continent_countries['name'] == corrected_name]
26
27 def find_income_level(x):
28     country_list = continent_countries['name'].drop_duplicates()
29     corrected_name = None
30     name = []
31     similarity = []
32     for country in country_list:
33         name.append(country)
34         similarity.append(SequenceMatcher(None, x, country).ratio())
35     name_similarity = pd.DataFrame({'name':name, 'similarity':similarity})
36     max_similarity = name_similarity['similarity'].max()
37     corrected_name = name_similarity[name_similarity['similarity'] == max_similar]
38     return continent_countries[continent_countries['name'] == corrected_name]

```

In [246]:

```

1 data_modified = data.copy()
2 data_modified['continent'] = data['country'].apply(lambda x : find_continent(x))
3 data_modified['subregion'] = data['country'].apply(lambda x : find_subregion(x))
4 data_modified['income_level'] = data['country'].apply(lambda x : find_income_level(x))

```

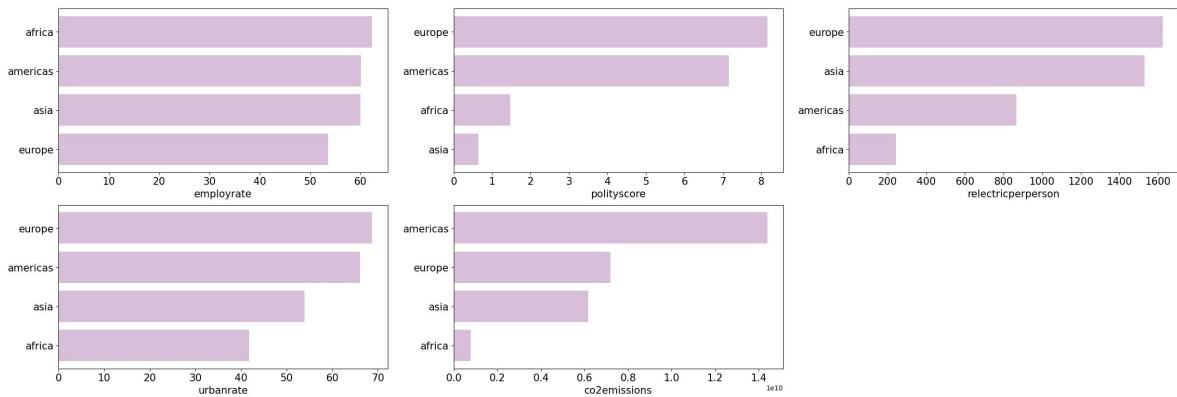
In [248]:

```

1 columns = ['employrate', 'polityscore', 'relectricperperson', 'urbanrate',
2 plt.figure(figsize=(30,15))
3 plt.suptitle('Continents',size=30)
4 for i in range(len(columns)):
5     plt.subplot(3,3,i+1)
6     #plt.title(columns[i])
7     graph_data = data_modified[['continent', columns[i]]].groupby('continent')
8     labels = graph_data.index
9     values = graph_data[columns[i]]
10    graph_data=pd.DataFrame({'values':values,'labels':labels}).sort_values('values')
11    plt.barh(graph_data['labels'],graph_data['values'],color='thistle')
12    plt.xlabel(columns[i],size=15)
13    plt.yticks(fontsize=15)
14    plt.xticks(fontsize=15)
15 plt.show()

```

Continents



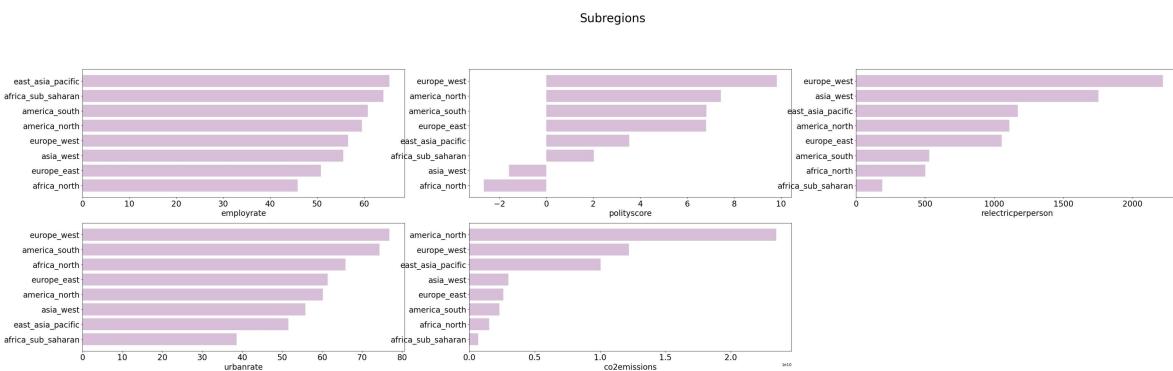
Se destaca que america es la principal productora de co2 entre los continentes, mientras que africa no produce casi nada en comparación a los demás continentes. Que los mayores consumidores eléctricos por persona son asia y europa, además de que los continentes notoriamente con menor polityscore son asia y africa. Encontramos cifras de empleo similares en todos los continentes destacando las de africa y america. Y una urbanización mucho mayor en europa y america.

In [249]:

```

1 columns = ['employrate', 'polityscore', 'relectricperperson', 'urbanrate',
2 plt.figure(figsize=(50,20))
3 plt.suptitle('Subregions',size=30)
4 for i in range(len(columns)):
5     plt.subplot(3,3,i+1)
6     #plt.title(columns[i])
7     graph_data = data_modified[['subregion', columns[i]]].groupby('subregion')
8     labels = graph_data.index
9     values = graph_data[columns[i]]
10    graph_data=pd.DataFrame({'values':values,'labels':labels}).sort_values(
11        'values', ascending=False)
12    plt.barh(graph_data['labels'],graph_data['values'],color='thistle')
13    plt.xlabel(columns[i],size=20)
14    plt.yticks(fontsize=20)
15    plt.xticks(fontsize=20)
16 plt.show()

```

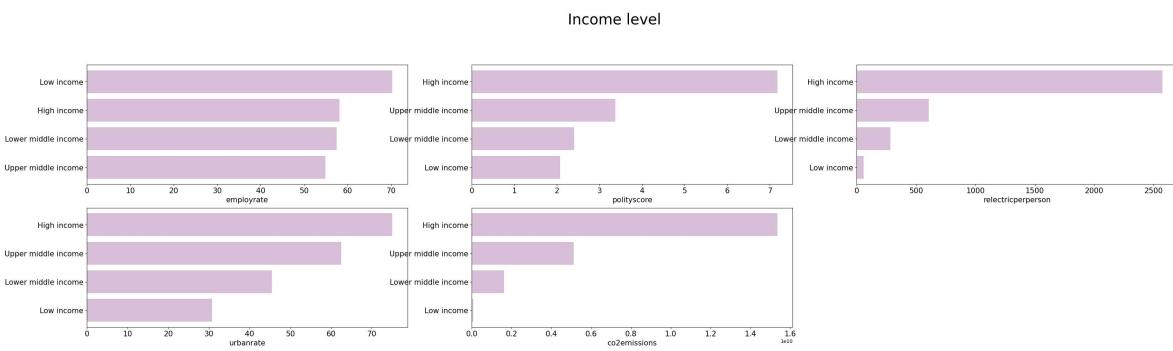


In [250]:

```

1 columns = ['employrate', 'polityscore', 'relectricperperson', 'urbanrate',
2 plt.figure(figsize=(40,15))
3 plt.suptitle('Income level',size=30)
4 for i in range(len(columns)):
5     plt.subplot(3,3,i+1)
6     #plt.title(columns[i])
7     graph_data = data_modified[['income_level', columns[i]]].groupby('income_level')
8     labels = graph_data.index
9     values = graph_data[columns[i]]
10    graph_data=pd.DataFrame({'values':values,'labels':labels}).sort_values(
11        'values', ascending=False)
12    plt.barh(graph_data['labels'],graph_data['values'],color='thistle')
13    plt.xlabel(columns[i],size=15)
14    plt.yticks(fontsize=15)
15    plt.xticks(fontsize=15)
16 plt.show()

```

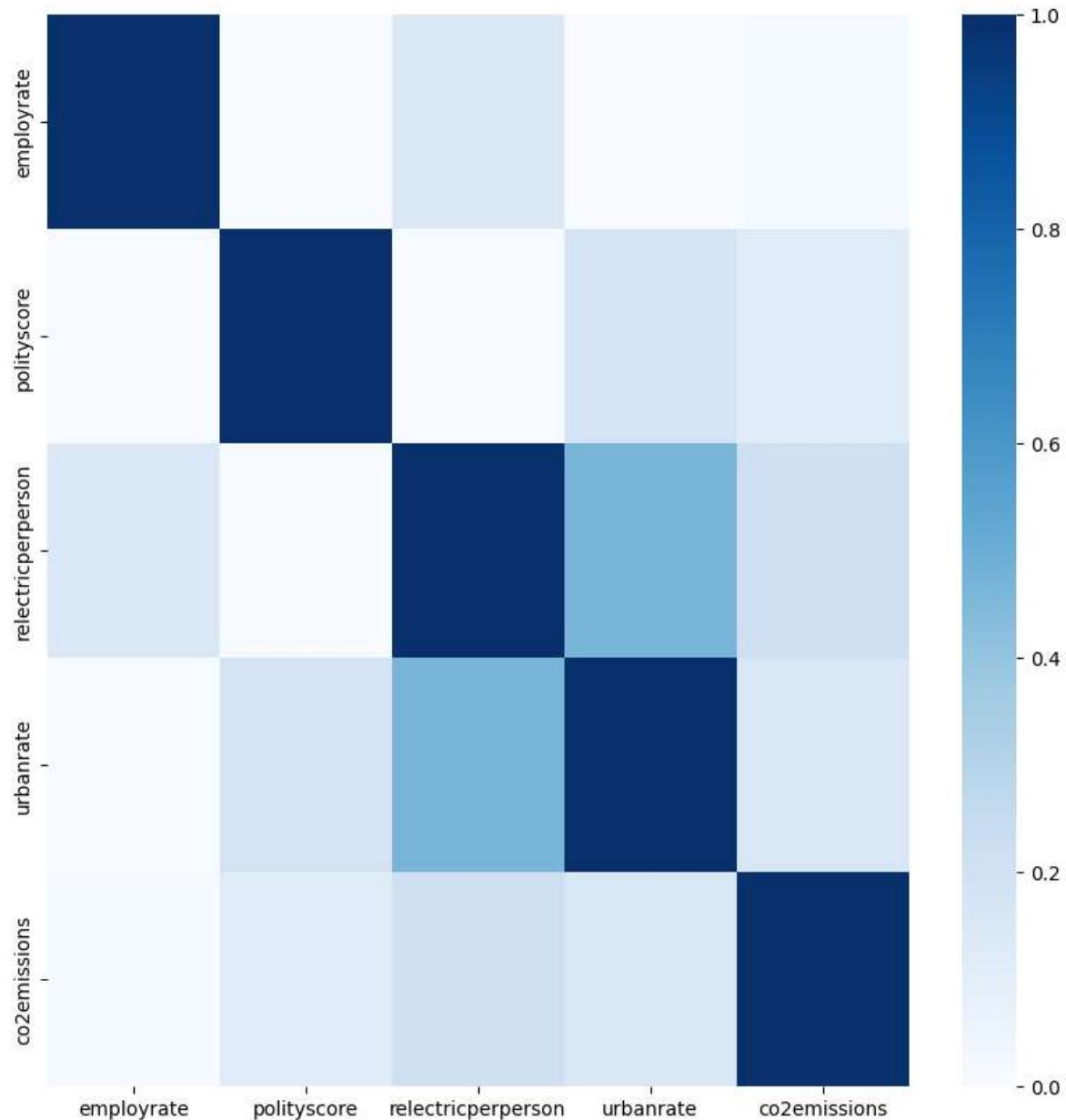


Se denota de esta gráfica que los que a mayor ingreso en los países. se tienen mayores emisiones de co2, mayor urbanización, menores tasas de empleo, mayor polityscore y mayor consumo eléctrico.

2.6.3 Matriz de Correlación

```
In [251]: 1 columns = ['employrate', 'polityscore', 'relectricperperson', 'urbanrate',
2 plt.figure(figsize=(10, 10))
3 sns.heatmap(data[columns].corr(), cmap="Blues", vmin=0, vmax=1)
```

Out[251]: <Axes: >



Se observa que no hay tanta aparente correlación entre las variables más importantes escogidas. Sobre todo entre relectricperperson y urban rate, los cuales evidentemente están correlacionadas, pero aún así no es tan alto este coeficiente.

3. Preparación de datos

3.1 Separación train-test

Antes de hacer cualquier modificación sobre los datos hacer la separación train - test para no sesgar de ninguna forma posible nuestro conjunto de validación.

```
In [252]: 1 X_train, X_test, y_train, y_test = train_test_split(data.drop(['incomeper...
```

3.2 Pipeline de preparación de datos

Agregaremos 2 nuevas columnas obtenidas del dataset obtenidas de la web gapminder:
<https://www.gapminder.org/fw/four-regions/> (<https://www.gapminder.org/fw/four-regions/>)

```
In [253]: 1 #X_train['subregion'] = X_train['country'].apply(lambda x : find_subregio...
  2 #X_train['income_Level'] = X_train['country'].apply(lambda x : find incom...
```

```
In [254]: 1 X_train.drop_duplicates(['country']).sample(5)
```

Out[254]:

		country	alccconsumption	armedforcesrate	breastcancerper100th	co2emissions	femaleen
126		Trinidad and Tobago	6.16	0.581165	51.1	NaN	5
114		Tajikistan	3.39	0.604873	13.2	1.041700e+08	5
131		Belize	5.92	0.758356	29.8	1.405800e+07	3
158		Bhutan	0.54	NaN	21.8	6.024333e+06	3
6		Mauritania	0.11	1.551262	28.1	5.681867e+07	4

Eliminar duplicados por país, dado que al querer hacer un modelo para predecir el PIB de un país, no tendría sentido ponerle dos veces la información del mismo país dado que serían redundante (no habría información nueva).

In [255]: 1 X_train.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 130 entries, 117 to 99
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   country          130 non-null    object  
 1   alcconsumption   130 non-null    float64 
 2   armedforcesrate  127 non-null    float64 
 3   breastcancerper100th  129 non-null    float64 
 4   co2emissions     128 non-null    float64 
 5   femaleemployrate 130 non-null    float64 
 6   hivrate          114 non-null    float64 
 7   internetuserate 128 non-null    float64 
 8   lifeexpectancy   130 non-null    float64 
 9   oilperperson      51 non-null    float64 
 10  polityscore      121 non-null    float64 
 11  relectricperperson 104 non-null    float64 
 12  suicideper100th  130 non-null    float64 
 13  employrate        130 non-null    float64 
 14  urbanrate         130 non-null    float64 
dtypes: float64(14), object(1)
memory usage: 16.2+ KB
```

Teniendo que tenemos relativamente pocos datos imputaremos los nulos con la media en todas las variables, excepto oilperperson, que tiene demasiados vacíos para ser imputados.

In [256]: 1 imp = SimpleImputer(missing_values=-1, strategy='mean')
2 X_train.drop(['oilperperson'],axis=1).sample(5)

Out[256]:

	country	alcconsumption	armedforcesrate	breastcancerper100th	co2emissions	female
57	Croatia	15.00	1.103279		62.1	3.100240e+08
90	Botswana	6.97	1.131910		33.4	7.894333e+07
33	South Africa	10.16	0.331863		35.0	1.460985e+10
71	Mongolia	3.41	1.211869		6.6	3.009343e+08
151	Turkmenistan	5.00	0.931418		17.9	5.313037e+08

Aplicaremos onehot encoder para variables categóricas y standard scaler para variables numéricas.

In [335]:

```

1 # Custom Transformer to add a new column
2 class CustomTransformer(BaseEstimator, TransformerMixin):
3     def fit(self, X, y=None):
4         return self
5
6     def transform(self, X, y=None):
7         X_copy = X.copy()
8         X_copy = X_copy.drop_duplicates(['country'])
9         X_copy['subregion'] = X_copy['country'].apply(lambda x : find_sub
10            X_copy['income_level'] = X_copy['country'].apply(lambda x : find_
11            X_copy = X_copy.drop(['oilperperson', 'country'], axis=1)
12            numeric_columns = list(X_copy.select_dtypes(include='number').col
13            for column in numeric_columns:
14                X_copy[column].replace(np.nan, np.mean(X_copy[column])), inplace=True)
15        return X_copy
16
17
18 def create_pipe():
19     variable_conversion = ColumnTransformer(
20         transformers=[
21             ('imputer', SimpleImputer(strategy='mean'), make_column_selector(d
22             ("num", StandardScaler(), make_column_selector(dtype_include=np.
23             ("cat", OneHotEncoder(handle_unknown='ignore')), make_column_selec
24         ]
25     )
26     pipe = Pipeline(steps=[
27         ('custom_processing', CustomTransformer()),
28         ('column_transformer', variable_conversion),
29         ('polynomial', PolynomialFeatures()),
30         ('regresion', Lasso()),
31     ])
32     return pipe

```

4. Entrenamiento de Modelo

Se entrena el modelo con un grid search usando grados polinomiales 1,2 y 3, y la regresión lasso, dado que nos brinda una mayor interpretabilidad en los resultados de los coeficientes.

In [336]:

```

1 pipeline = create_pipe()
2 parameters = {
3     'polynomial_degree':[1,2,3],
4     'regresion_alpha': [0.01, 0.1, 1, 10, 100]
5 }
6 grid_search = GridSearchCV(pipeline, parameters, verbose=2, scoring='neg_

```

In [337]:

```
1 %%time
2 grid_search.fit(X_train,y_train)
```

Fitting 5 folds for each of 15 candidates, totalling 75 fits

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_
descent.py:631: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations, check the scale of the feature
s or consider increasing regularisation. Duality gap: 3.820e+07, toleranc
e: 9.269e+05
    model = cd_fast.enet_coordinate_descent()

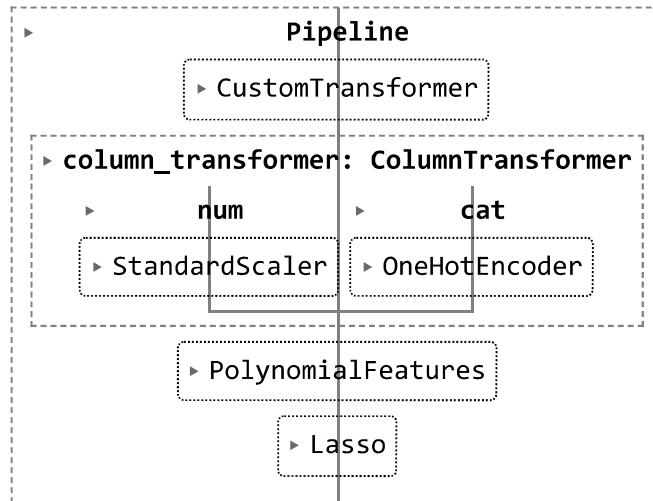
[CV] END .....polynomial_degree=1, regresion_alpha=0.01; total time=
3.4s
[CV] END .....polynomial_degree=1, regresion_alpha=0.01; total time=
1.7s
[CV] END .....polynomial_degree=1, regresion_alpha=0.01; total time=
1.7s
[CV] END .....polynomial_degree=1, regresion_alpha=0.01; total time=
1.7s
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_
descent.py:631: ConvergenceWarning: Objective did not converge. You might
```

In [338]:

```
1 best_model_lasso = grid_search.best_estimator_
2 best_model_lasso
```

Out[338]:



Mejores parámetros encontrados:

In [346]:

```
1 grid_search.best_params_
```

Out[346]:

```
{'polynomial_degree': 1, 'regresion_alpha': 100}
```

5. Evaluación del modelo

5.1 Rendimiento con dataset de train

In [347]:

```

1 y_pred_train = best_model_lasso.predict(X_train)
2 n,p = X_train.shape
3
4 print('----- Regresión Lasso con entrenamiento-----')
5 print("Residual sum of squares (MSE): %.2f" % mean_squared_error(y_train,
6 print("R2-score: %.5f" % r2_score(y_train, y_pred_train) )
7 print("Adj R2-score: %.5f" % ( 1-(1-r2_score(y_train, y_pred_train))*(n-1)

```

----- Regresión Lasso con entrenamiento-----
 Residual sum of squares (MSE): 18335332.31
 R2-score: 0.81969
 Adj R2-score: 0.79596

5.2 Rendimiento con dataset de test

In [348]:

```

1 y_pred_test = best_model_lasso.predict(X_test)
2 n,p = X_test.shape
3
4 print('----- Regresión Lasso con evaluación -----')
5 print("Residual sum of squares (MSE): %.2f" % mean_squared_error(y_test,y
6 print("R2-score: %.5f" % r2_score(y_test, y_pred_test) )
7 print("Adj R2-score: %.5f" % ( 1-(1-r2_score(y_test, y_pred_test))*(n-1)/

```

----- Regresión Lasso con evaluación -----
 Residual sum of squares (MSE): 18935617.37
 R2-score: 0.86221
 Adj R2-score: 0.74063

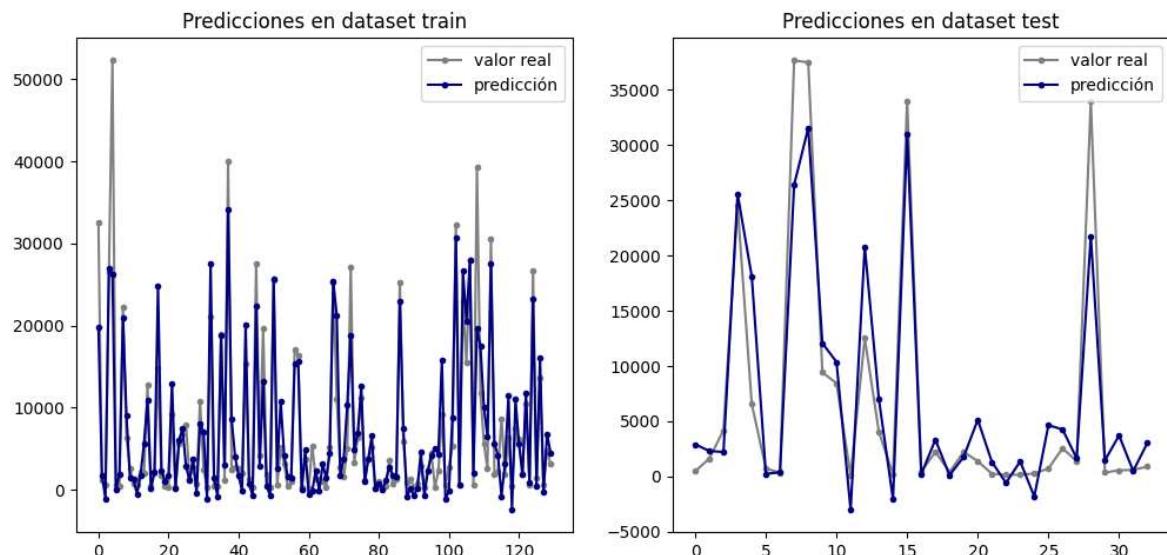
Se podría hablar de un ligero bajo ajuste dado que el r2 es menor en el dataset de entrenamiento que en el dataset de testing. Sin embargo con un r cuadrado de 0.86 para el dataset de test, podemos indicar que un gran porcentaje de la varianza de pib es explicado por el modelo.

In [349]:

```

1 plt.figure(figsize = (12,12))
2 plt.subplot(2,2,1)
3 plt.title('Predicciones en dataset train')
4 a = 0
5 b= len(y_pred_train)
6 xvals = list(range(b-a))
7 plt.plot(xvals, y_train, '.-', color='grey', label='valor real')
8 plt.plot(xvals, y_pred_train, '.-', color = 'navy',label = 'predicción')
9 plt.legend()
10 plt.subplot(2,2,2)
11 plt.title('Predicciones en dataset test')
12 a = 0
13 b= len(y_pred_test)
14 xvals = list(range(b-a))
15 plt.plot(xvals, y_test, '.-', color='grey', label='valor real')
16 plt.plot(xvals, y_pred_test, '.-', color = 'navy',label = 'predicción')
17 plt.legend()
18 plt.show()

```



5.3 Coeficientes de la regresión realizada

In [370]:

```
1 cat_names = best_model_lasso['column_transformer'].transformers_[1][1].get_feature_names_out()
2 num_names = best_model_lasso['column_transformer'].transformers_[0][1].feature_names_
3 col_names = list(num_names) + list(cat_names)
4 print('Intercepto:', best_model_lasso['regresion'].intercept_)
5 coef = list(zip(['Intercepto'] + list(col_names), [best_model_lasso['regresion'].intercept_]+best_model_lasso['regresion'].coef_))
6 coef = pd.DataFrame(coef,columns=['Variable','Parámetro']).sort_values(by='Variable')
7 coef
```

Intercepto: 7099.776439693434

Out[370]:

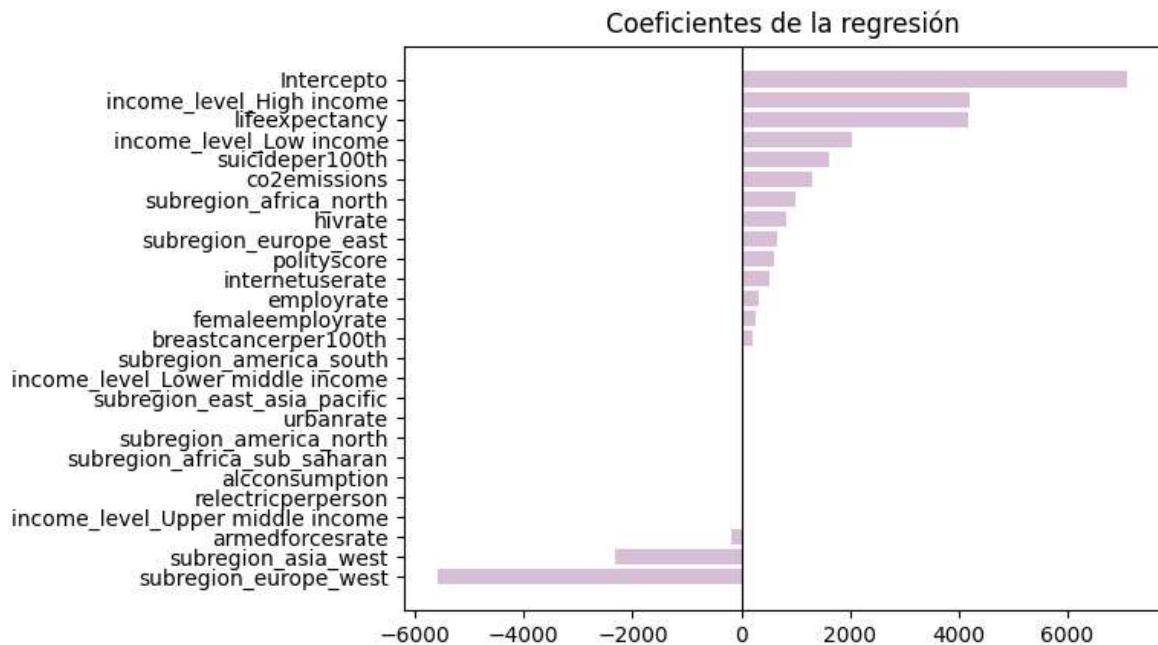
	Variable	Parámetro
0	Intercepto	7099.776440
22	income_level_High income	4211.056427
8	lifeexpectancy	4170.587768
23	income_level_Low income	2035.088740
11	suicideper100th	1621.910620
4	co2emissions	1308.132837
14	subregion_africa_north	982.356540
6	hivrate	818.058666
20	subregion_europe_east	651.255540
9	polityscore	611.328210
7	internetuserate	509.704167
12	employrate	330.091591
5	femaleemployrate	263.823965
3	breastcancerper100th	201.465759
17	subregion_americasouth	0.000000
24	income_level_Lower middle income	-0.000000
19	subregion_east_asia_pacific	-0.000000
13	urbanrate	0.000000
16	subregion_americasouth	0.000000
15	subregion_africa_sub_saharan	0.000000
1	alconsumption	0.000000
10	relectricperperson	-0.000000
25	income_level_Upper middle income	-0.000000
2	armedforcesrate	-170.700526
18	subregion_asia_west	-2321.298050
21	subregion_europe_west	-5579.523851

In [372]:

```

1 coef = coef.sort_values(by='Parámetro')
2 plt.barh(coef[ 'Variable' ],coef[ 'Parámetro' ],color='thistle')
3 plt.axvline(x=0,color='k',lw=.8)
4 plt.title('Coeficientes de la regresión')
5 plt.show()

```



Se puede observar de acuerdo a los valores obtenidos en los coeficientes que las variables que son realmente importantes para este modelo se comportan de la siguiente forma:

- La clasificación de high income por parte del banco mundial parece ser la que más indica que un país es apto para préstamos.
- Luego encontramos que una alta esperanza de vida es un buen indicador de que para un pib alto.
- las emisiones de co2, internetuserate, el polityscore y el employrate, también muestran una relación positiva con un pib alto.
- El porcentaje de personas en fuerzas armadas disminuye el pib.

5.4 Validación de los supuestos de la regresión

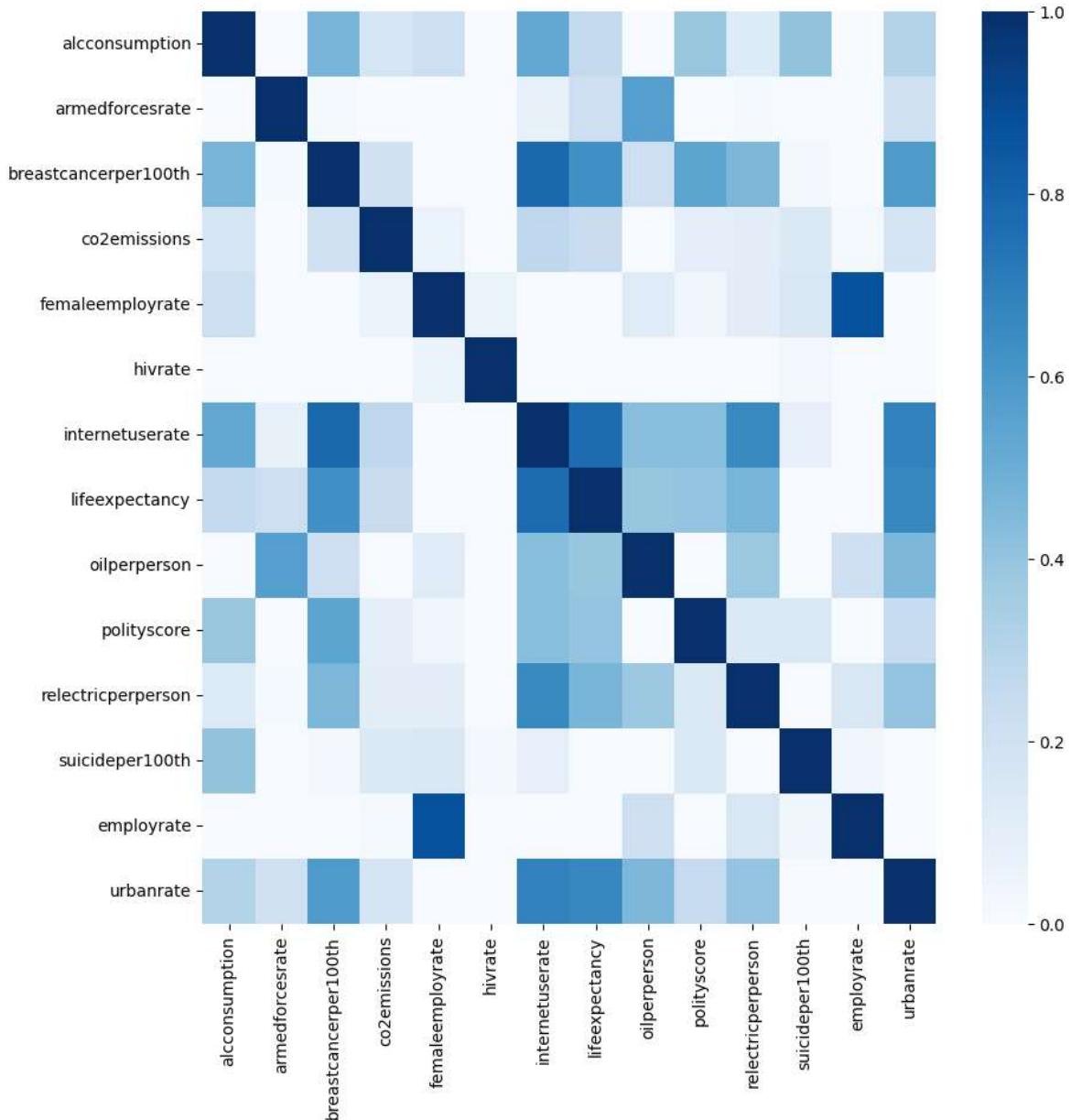
5.4.1 Colinealidad

```
In [353]: 1 plt.figure(figsize=(10, 10))
2 sns.heatmap(X_train.corr(), cmap="Blues", vmin=0, vmax=1)
```

<ipython-input-353-b9cfbf148462>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.

```
sns.heatmap(X_train.corr(), cmap="Blues", vmin=0, vmax=1)
```

Out[353]: <Axes: >



Vemos correlaciones importantes únicamente entre femaleemployrate y employrate, lo cual es bastante lógico pero no necesariamente son causales. Y adeás entre urban rate, internetuserate y lifeexpectancy, sin embargo, esta condición podría decirse que se cumple.

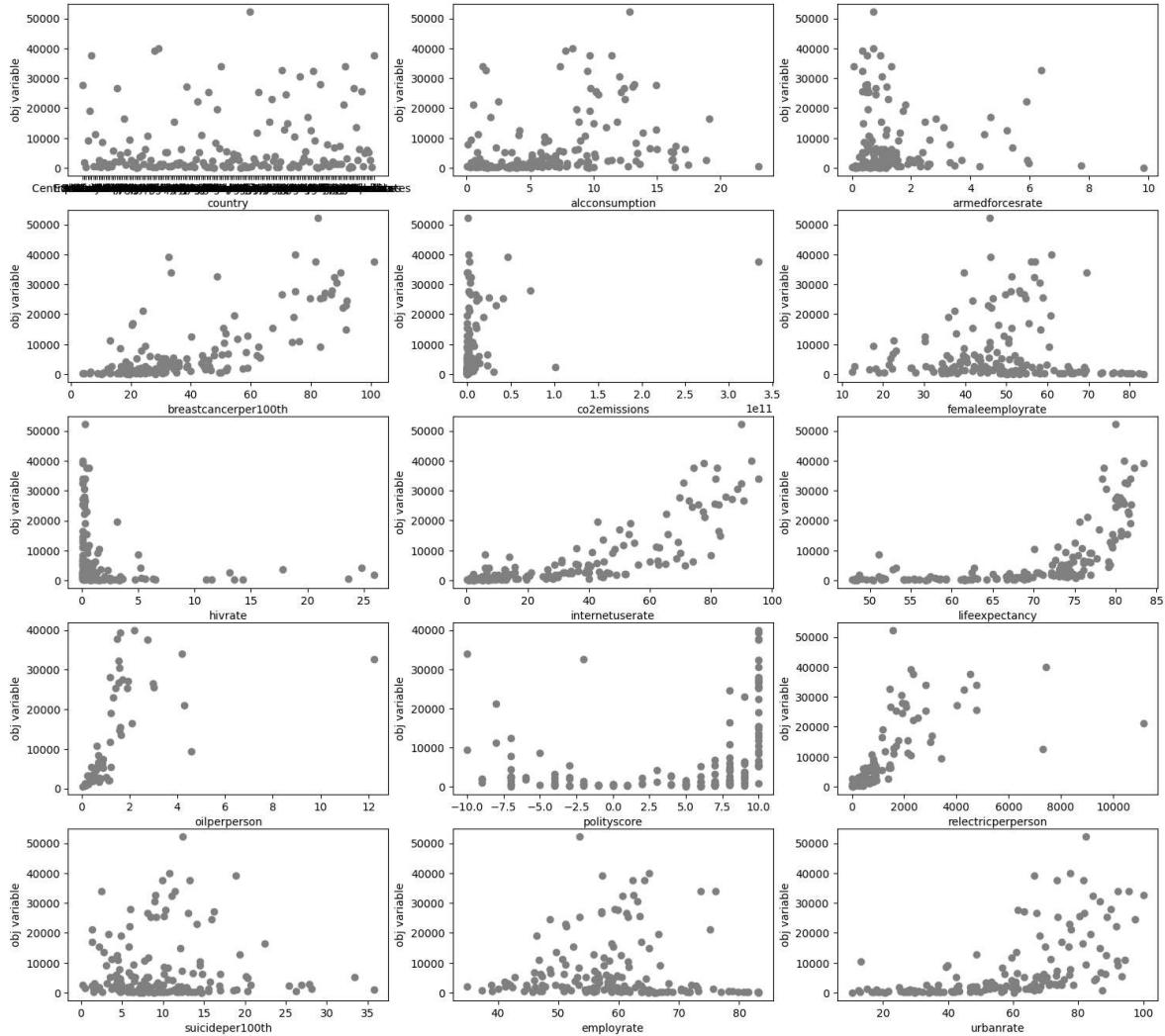
5.4.2 Linealidad

In [352]:

```

1 columns = list(data.drop(['incomeperperson'],axis=1).columns)
2 plt.figure(figsize=(18,20))
3 for i in range(len(columns)):
4     plt.subplot(int(len(columns)/3)+1,3,i+1)
# plt.title(columns[i])
5     plt.xlabel(columns[i])
6     plt.ylabel('obj variable')
7     plt.scatter(data[columns[i]], data['incomeperperson'], color ='grey')
8
9 plt.show()

```



Solo con ver la distribución de los datos, se puede observar que no necesariamente existe alguna correlación lineal entre la mayoría de los indicadores y la variable objetivo. Por lo que este supuesto no se cumple en su mayoría.

5.4.3 Normalidad de errores

In [355]:

```

1 errors = (best_model_lasso.predict(X_train)-y_train).values

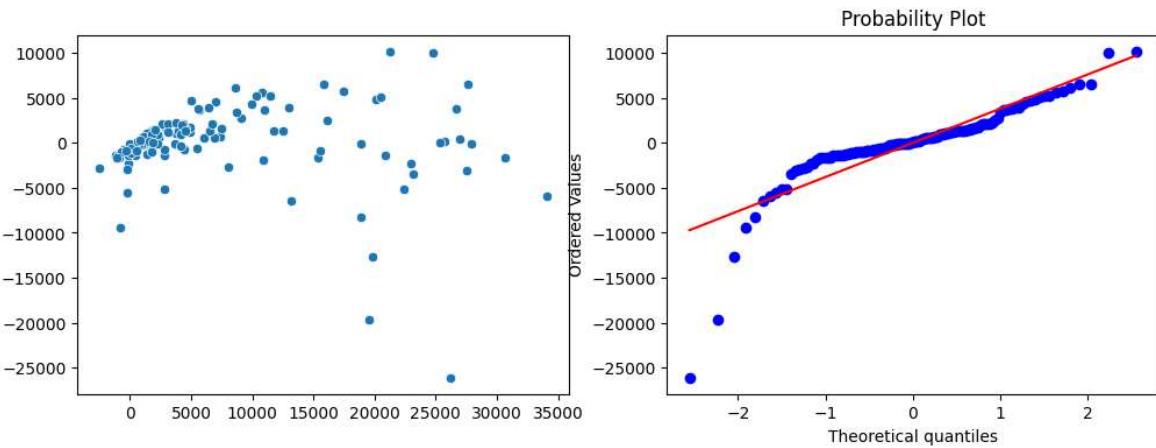
```

In [380]:

```

1 fig, axes = plt.subplots(1, 2, figsize=(12, 4))
2
3 # Dispersion
4 sns.scatterplot(x=best_model_lasso.predict(X_train), y=errors, ax=axes[0])
5
6 # q-q plot
7 _ = stats.probplot(errors, dist="norm", plot=axes[1])

```

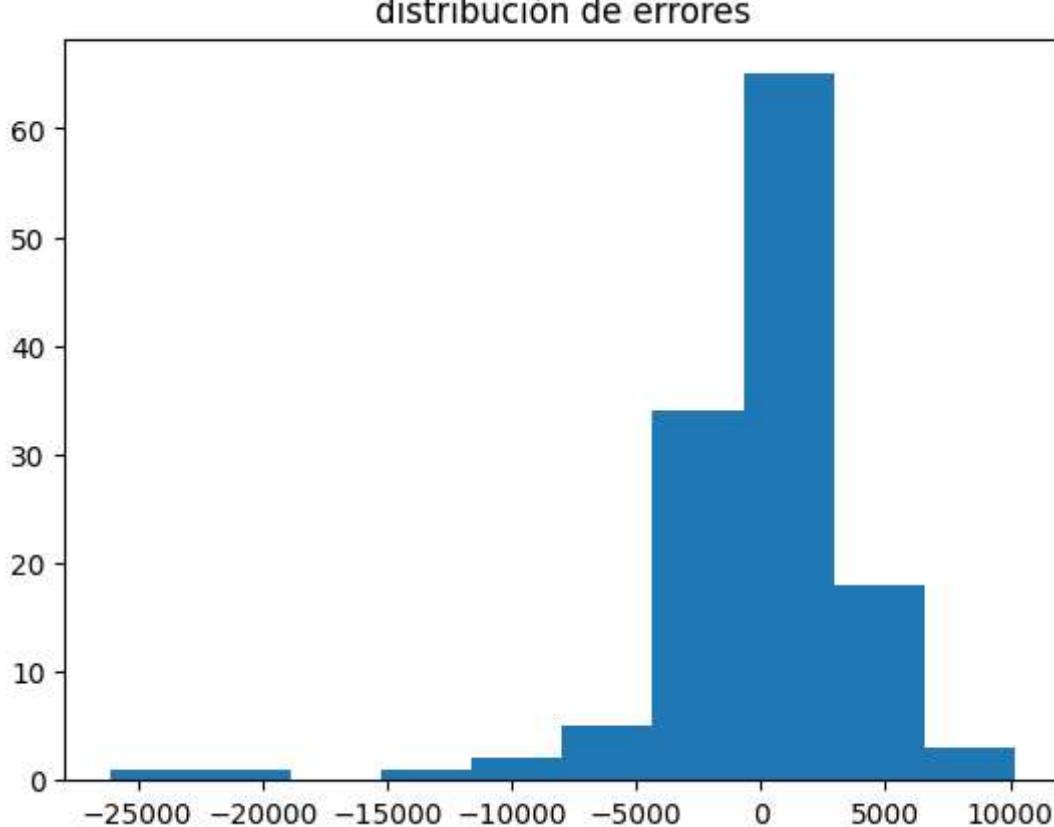


In [361]:

```

1 plt.title('distribución de errores')
2 plt.hist(errors)
3 plt.show()

```

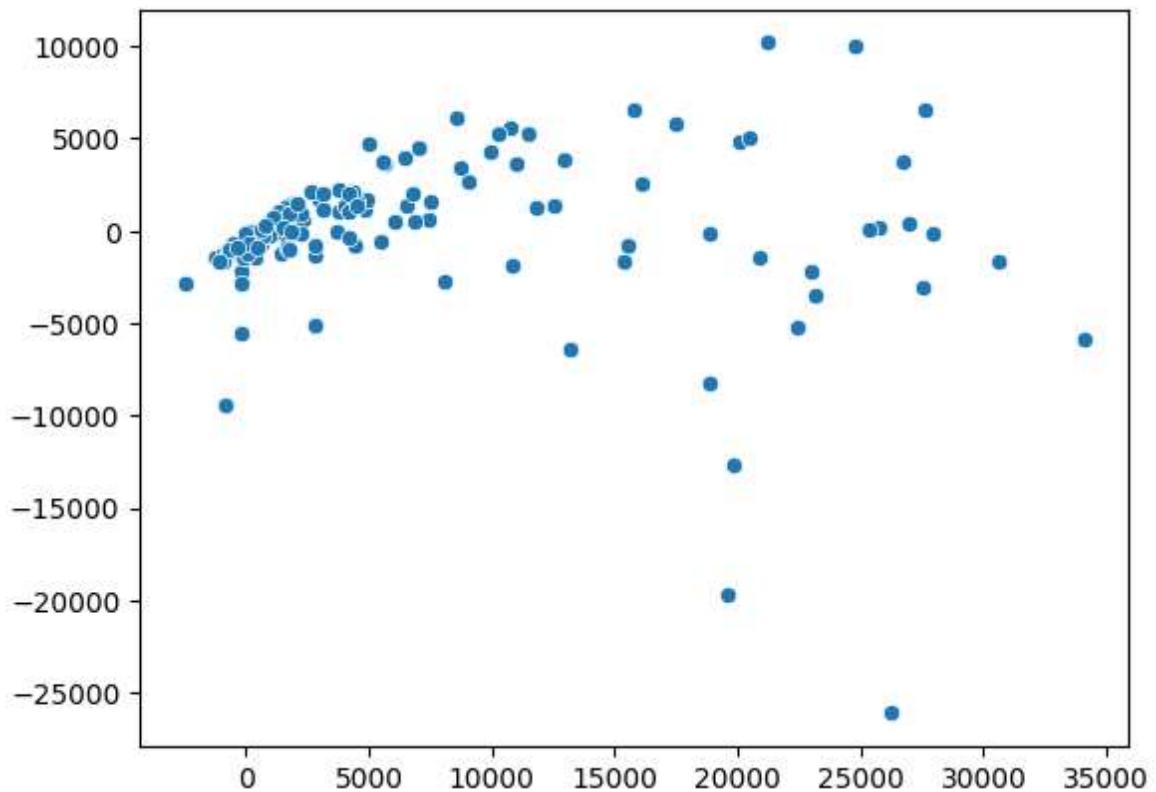


Fuera de algunos outliers, parece que los errores sí tienen una distribución normal.

5.4.4 Varianza constante en los errores (Homocedasticidad)

In [379]: 1 sns.scatterplot(x = best_model_lasso.predict(X_train), y=errors)

Out[379]: <Axes: >



No parece que se cumpla del todo una varianza constante en los errores. A pesar de todo esto el modelo tiene un buen rendimiento de rscore 0.86 para el dataset de test.

In []: 1

In []: 1