Project- CCPS 844 Data Mining

Salman AlMaskati

This project aims to tackle a challenging regression problem focused on predicting life expectancy. Life expectancy is a critical indicator of the overall health and well-being of a population, and accurately predicting it can have far-reaching implications for public health, social policies, and resource allocation.\ \ The key steps of the project involve data cleaing, visualizations, Clustering, feature selection and PCA, model training and evaluation. Several regression learing algorithms will be explored and compared to find the best-performing model.

Table of Contents

Data summary

In [ ]:

```python
import pandas as pd

df=pd.read_csv("world-data-2023.csv")
df
```

Out[ ]:

	Country	Density\n(P/Km2)	Abbreviation	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Calling Code	Capi
0	Afghanistan	60	AF	58.10%	652,230	323,000	32.49	93.0	
1	Albania	105	AL	43.10%	28,748	9,000	11.78	355.0	
2	Algeria	18	DZ	17.40%	2,381,741	317,000	24.28	213.0	
3	Andorra	164	AD	40.00%	468	NaN	7.20	376.0	
4	Angola	26	AO	47.50%	1,246,700	117,000	40.73	244.0	
...	...	...	...	...	...	...	...	...	
190	Venezuela	32	VE	24.50%	912,050	343,000	17.88	58.0	
191	Vietnam	314	VN	39.30%	331,210	522,000	16.75	84.0	
192	Yemen	56	YE	44.60%	527,968	40,000	30.45	967.0	
193	Zambia	25	ZM	32.10%	752,618	16,000	36.19	260.0	
194	Zimbabwe	38	ZW	41.90%	390,757	51,000	30.68	263.0	

195 rows × 35 columns

In [ ]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 35 columns):
 #   Column                                   Non-Null Count  Dtype  
---  ------                                   --------------  -----  
 0   Country                                  195 non-null    object 
 1   Density
(P/Km2)                          195 non-null    object 
 2   Abbreviation                             188 non-null    object 
 3   Agricultural Land( %)                    188 non-null    object 
 4   Land Area(Km2)                           194 non-null    object 
 5   Armed Forces size                        171 non-null    object 
 6   Birth Rate                               189 non-null    float64
 7   Calling Code                             194 non-null    float64
 8   Capital/Major City                       192 non-null    object 
 9   Co2-Emissions                            188 non-null    object 
 10  CPI                                      178 non-null    object 
 11  CPI Change (%)                           179 non-null    object 
 12  Currency-Code                            180 non-null    object 
 13  Fertility Rate                           188 non-null    float64
```

```
 14  Forested Area (%)                            188 non-null    object
 15  Gasoline Price                               175 non-null    object
 16  GDP                                          193 non-null    object
 17  Gross primary education enrollment (%)       188 non-null    object
 18  Gross tertiary education enrollment (%)      183 non-null    object
 19  Infant mortality                             189 non-null    float64
 20  Largest city                                 189 non-null    object
 21  Life expectancy                              187 non-null    float64
 22  Maternal mortality ratio                     181 non-null    float64
 23  Minimum wage                                 150 non-null    object
 24  Official language                            194 non-null    object
 25  Out of pocket health expenditure             188 non-null    object
 26  Physicians per thousand                      188 non-null    float64
 27  Population                                   194 non-null    object
 28  Population: Labor force participation (%)    176 non-null    object
 29  Tax revenue (%)                              169 non-null    object
 30  Total tax rate                               183 non-null    object
 31  Unemployment rate                            176 non-null    object
 32  Urban_population                             190 non-null    object
 33  Latitude                                     194 non-null    float64
 34  Longitude                                    194 non-null    float64
dtypes: float64(9), object(26)
memory usage: 53.4+ KB
```

In [ ]:

```
df.describe()
```

Out[ ]:

	Birth Rate	Calling Code	Fertility Rate	Infant mortality	Life expectancy	Maternal mortality ratio	Physicians per thousand	Latitude
count	189.000000	194.000000	188.000000	189.000000	187.000000	181.000000	188.000000	194.00000
mean	20.214974	360.546392	2.698138	21.332804	72.279679	160.392265	1.839840	19.092351
std	9.945774	323.236419	1.282267	19.548058	7.483661	233.502024	1.684261	23.96177
min	5.900000	1.000000	0.980000	1.400000	52.800000	2.000000	0.010000	-40.90055
25%	11.300000	82.500000	1.705000	6.000000	67.000000	13.000000	0.332500	4.54417
50%	17.950000	255.500000	2.245000	14.000000	73.200000	53.000000	1.460000	17.27384
75%	28.750000	506.750000	3.597500	32.700000	77.500000	186.000000	2.935000	40.12460
max	46.080000	1876.000000	6.910000	84.500000	85.400000	1150.000000	8.420000	64.963051

Droping columns that do not add value to the df

In [ ]:

```
cols_to_drop= ['Abbreviation','Calling Code','Capital/Major City','Currency-Code','Larg
df = df.drop(columns=cols_to_drop)
df
```

Out[ ]:

	Country	Density\n(P/Km2)	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Co2-Emissions	CPI	CF Chang (%
0	Afghanistan	60	58.10%	652,230	323,000	32.49	8,672	149.9	2.30

	Country	Density\n(P/Km2)	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Co2-Emissions	CPI	CI Chang (%
1	Albania	105	43.10%	28,748	9,000	11.78	4,536	119.05	1.40
2	Algeria	18	17.40%	2,381,741	317,000	24.28	150,006	151.36	2.00
3	Andorra	164	40.00%	468	NaN	7.20	469	NaN	Nal
4	Angola	26	47.50%	1,246,700	117,000	40.73	34,693	261.73	17.10
...	...	...	...	...	...	...	...	...	
190	Venezuela	32	24.50%	912,050	343,000	17.88	164,175	2,740.27	254.90
191	Vietnam	314	39.30%	331,210	522,000	16.75	192,668	163.52	2.80
192	Yemen	56	44.60%	527,968	40,000	30.45	10,609	157.58	8.10
193	Zambia	25	32.10%	752,618	16,000	36.19	5,141	212.31	9.20
194	Zimbabwe	38	41.90%	390,757	51,000	30.68	10,983	105.51	0.90

195 rows × 24 columns

In [ ]:

```
df.shape
```

Out[ ]:

```
(195, 24)
```

Visualizing NA's

In [ ]:

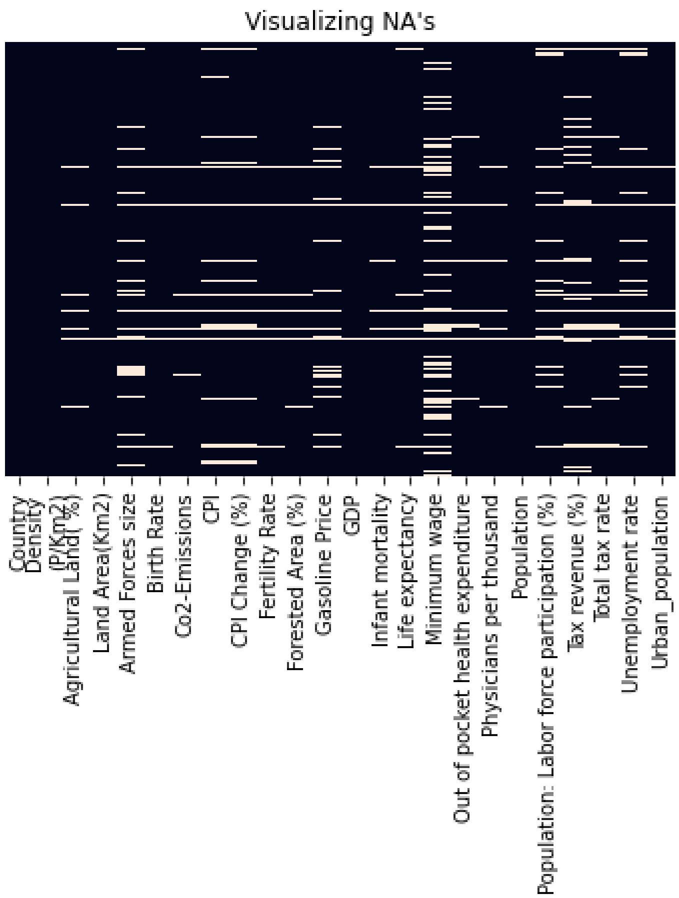
```
import seaborn as sns
import matplotlib.pyplot as plt

na_counts = df.isna().sum()
print("The Number of NA's in",na_counts)

sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
plt.title("Visualizing NA's")
```

```
The Number of NA's in Country                                    0
Density\n(P/Km2)                          0
Agricultural Land( %)                     7
Land Area(Km2)                            1
Armed Forces size                        24
Birth Rate                                6
Co2-Emissions                             7
CPI                                      17
CPI Change (%)                           16
Fertility Rate                            7
Forested Area (%)                         7
Gasoline Price                           20
GDP                                       2
Infant mortality                          6
Life expectancy                           8
Minimum wage                             45
```

```
Out of pocket health expenditure              7
Physicians per thousand                       7
Population                                    1
Population: Labor force participation (%)    19
Tax revenue (%)                              26
Total tax rate                               12
Unemployment rate                            19
Urban_population                              5
dtype: int64
```

Out[ ]:

```
Text(0.5, 1.0, "Visualizing NA's")
```



Web Scraping

Fill in Agricultural Land(%) values

In [ ]:

```
from lxml import html
from bs4 import BeautifulSoup
import requests
```

In [ ]:

```
#Find Out which countries have missing Agricultural Land( %) values
rows_with_na = df[df["Agricultural Land( %)"].isna()]

# Display the result
print(rows_with_na)

print("Eswatini, Monaco, Nauru, North Macedonia, Palestinian National Authority, South
```

```
                           Country Density\n(P/Km2) Agricultural Land( %)  \
56                        Eswatini              67                   NaN
73                    Vatican City           2,003                   NaN
113                         Monaco          26,337                   NaN
120                          Nauru             541                   NaN
128                North Macedonia              83                   NaN
133  Palestinian National Authority            847                   NaN
163                    South Sudan              18                   NaN

    Land Area(Km2) Armed Forces size  Birth Rate Co2-Emissions       CPI  \
56          17,364               NaN         NaN           NaN       NaN
73               0               NaN         NaN           NaN       NaN
113              2               NaN        5.90           NaN       NaN
120             21               NaN         NaN           NaN       NaN
128         25,713               NaN         NaN           NaN       NaN
133            NaN               NaN         NaN           NaN       NaN
163        644,329           185,000       35.01         1,727  4,583.71

    CPI Change (%)  Fertility Rate  ... Life expectancy Minimum wage  \
56             NaN             NaN  ...             NaN          NaN
73             NaN             NaN  ...             NaN          NaN
113            NaN             NaN  ...             NaN      $11.72
120            NaN             NaN  ...             NaN          NaN
128            NaN             NaN  ...             NaN          NaN
133            NaN             NaN  ...             NaN          NaN
163        187.90%             4.7  ...            57.6          NaN

    Out of pocket health expenditure  Physicians per thousand  Population  \
56                            11.30%                      NaN   1,093,238
73                               NaN                      NaN         836
113                            6.10%                     6.56      38,964
120                              NaN                      NaN      10,084
128                           35.60%                      NaN   1,836,713
133                              NaN                      NaN         NaN
163                           61.30%                      NaN  11,062,113

    Population: Labor force participation (%) Tax revenue (%)  Total tax rate  \
56                                        NaN          28.60%             NaN
73                                        NaN             NaN             NaN
113                                       NaN             NaN             NaN
120                                       NaN             NaN             NaN
128                                       NaN             NaN             NaN
133                                       NaN             NaN             NaN
163                                    72.40%             NaN          31.40%

    Unemployment rate Urban_population
56                NaN              NaN
73                NaN              NaN
113               NaN           38,964
120               NaN              NaN
128               NaN              NaN
133               NaN              NaN
163            12.24%        2,201,250

[7 rows x 24 columns]
Eswatini, Monaco, Nauru, North Macedonia, Palestinian National Authority, South Sudan
```

In [ ]:

```
#get agricultral land data
import requests
```

```
from bs4 import BeautifulSoup
```

In [ ]:

```
url = 'https://wdi.worldbank.org/table/3.2'

# end a GET request to the URL
response = requests.get(url)
html_content = response.text

# create a Beautiful Soup object
soup = BeautifulSoup(html_content, 'html.parser')

country_elements = soup.find_all(class_="country")

#Eswatini %
for country_element in country_elements:
    country_content = country_element.get_text()

    if "Eswatini" in country_content:
        row_element = country_element.find_parent("tr")
        if row_element:
            eswatini_row = row_element.get_text()
            #print("Row:", eswatini_row)

#Monaco %
for country_element in country_elements:
    country_content = country_element.get_text()

    if "Monaco" in country_content:
        row_element = country_element.find_parent("tr")
        if row_element:
            monaco_row = row_element.get_text()
            #print("Row:", monaco_row)

#Nauru %
for country_element in country_elements:
    country_content = country_element.get_text()

    if "North Macedonia" in country_content:
        row_element = country_element.find_parent("tr")
        if row_element:
            nm_row = row_element.get_text()
            #print("Row:", nm_row)

#South Sudan %
for country_element in country_elements:
    country_content = country_element.get_text()

    if "South Sudan" in country_content:
        row_element = country_element.find_parent("tr")
        if row_element:
            ss_row = row_element.get_text()
            #print("Row:", ss_row)

print("Nauru, Vatican City, and Moanco, could not be found\nPalestinian National Author

# helper function to format extaracted data
def get_first_number(words):
    for word in words:
```

```
        if word.replace('.', '', 1).isdigit():  # formating
            return float(word)


# split the rows by whitespaces
nm_words = nm_row.split()
ss_words = ss_row.split()
eswatini_words = eswatini_row.split()

# create  dictionary
data_dict = {
    'North Macedonia': get_first_number(nm_words),
    'South Sudan': get_first_number(ss_words),
    'Eswatini': get_first_number(eswatini_words)
}

print(data_dict)
```

```
Nauru, Vatican City, and Moanco, could not be found
Palestinian National Authority(will be dropped due to lack of data)
{'North Macedonia': 50.0, 'South Sudan': 45.0, 'Eswatini': 71.0}
```

In [ ]:

```
#input values to df
df.loc[df['Country'] == 'North Macedonia', 'Agricultural Land( %)'] = get_first_number(
df.loc[df['Country'] == 'South Sudan', 'Agricultural Land( %)'] = get_first_number(ss_w
df.loc[df['Country'] == 'Eswatini', 'Agricultural Land( %)'] = get_first_number(eswatin
```

Data Preprocessing

Replace '0' in all rows with NA

In [ ]:

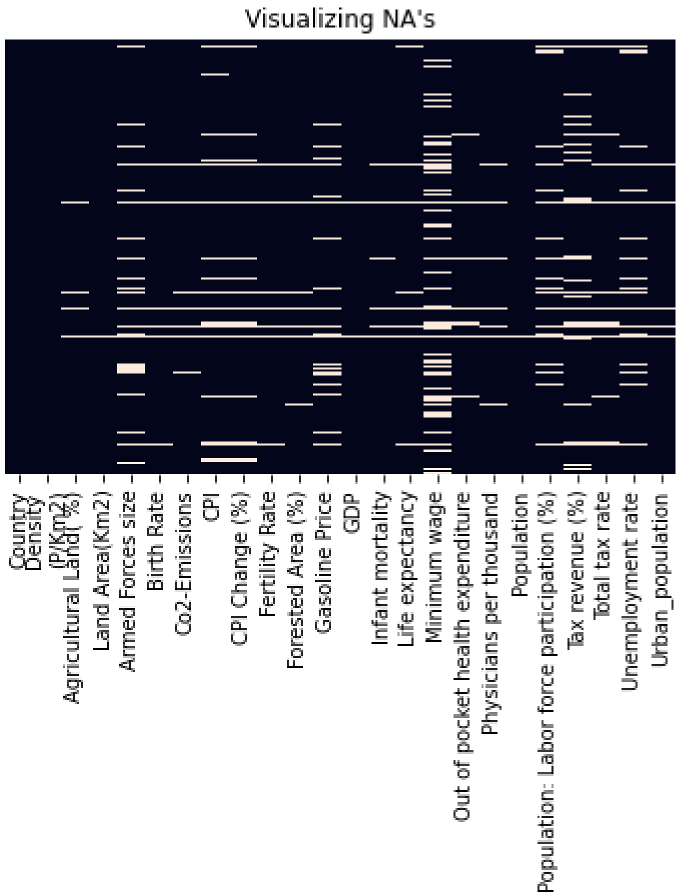
```
import numpy as np
df = df.replace(0, np.nan)
```

In [ ]:

```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
plt.title("Visualizing NA's")
na_counts = df.isna().sum()
print("The Number of NA's in",na_counts)
```

```
The Number of NA's in Country                                    0
Density\n(P/Km2)                          0
Agricultural Land( %)                     4
Land Area(Km2)                            1
Armed Forces size                        24
Birth Rate                                6
Co2-Emissions                             7
CPI                                      17
CPI Change (%)                           16
Fertility Rate                            7
Forested Area (%)                         7
Gasoline Price                           20
GDP                                       2
Infant mortality                          6
Life expectancy                           8
Minimum wage                             45
```

```
Out of pocket health expenditure             7
Physicians per thousand                      7
Population                                   1
Population: Labor force participation (%)   19
Tax revenue (%)                             26
Total tax rate                              12
Unemployment rate                           19
Urban_population                             5
dtype: int64
```



Drop Minumum Wage column due to lack of data

In [ ]:
```python
df.drop(columns='Minimum wage', inplace=True)
```

Drop ROWS that have NA values

In [ ]:
```python
df = df.dropna(axis=0)
```

In [ ]:
```python
print(df.isnull().sum())
```

```
Country                                     0
Density\n(P/Km2)                            0
Agricultural Land( %)                       0
Land Area(Km2)                              0
Armed Forces size                           0
Birth Rate                                  0
Co2-Emissions                               0
CPI                                         0
CPI Change (%)                              0
```

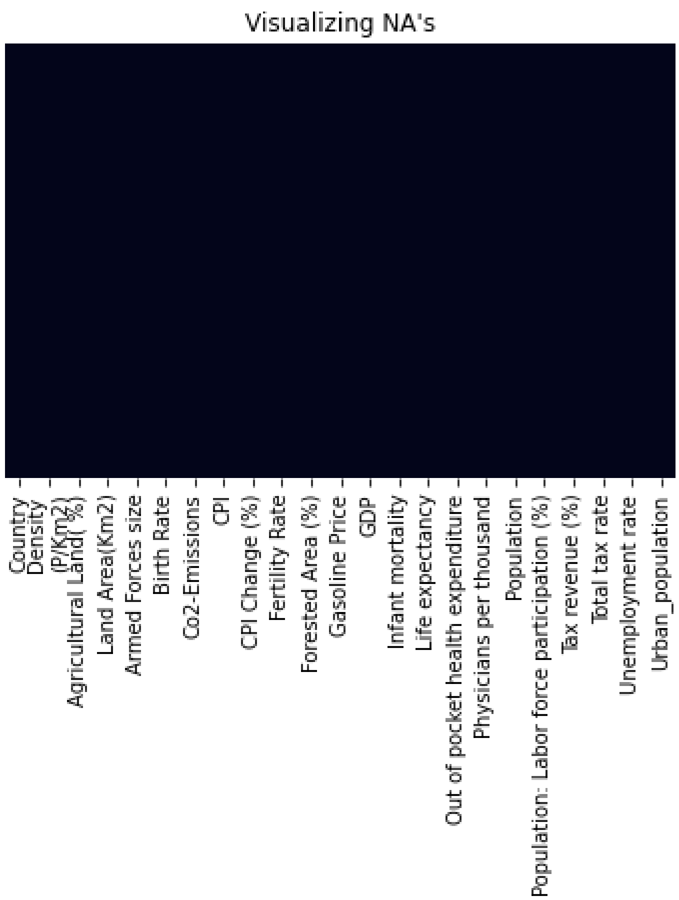
```
Fertility Rate                                0
Forested Area (%)                             0
Gasoline Price                                0
GDP                                           0
Infant mortality                              0
Life expectancy                               0
Out of pocket health expenditure              0
Physicians per thousand                       0
Population                                    0
Population: Labor force participation (%)    0
Tax revenue (%)                               0
Total tax rate                                0
Unemployment rate                             0
Urban_population                              0
dtype: int64
```

In [ ]:
```python
df.shape
```

Out[ ]: (146, 23)

In [ ]:
```python
sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
plt.title("Visualizing NA's")
```

Out[ ]: Text(0.5, 1.0, "Visualizing NA's")



change data types 'object' to 'float64' so we can fit the dataframe into machine learning algorithms

In [ ]:
```
df.dtypes
```

Out[ ]:
```
Country                                     object
Density\n(P/Km2)                            object
Agricultural Land( %)                       object
Land Area(Km2)                              object
Armed Forces size                           object
Birth Rate                                 float64
Co2-Emissions                               object
CPI                                         object
CPI Change (%)                              object
Fertility Rate                             float64
Forested Area (%)                           object
Gasoline Price                              object
GDP                                         object
Infant mortality                           float64
Life expectancy                            float64
Out of pocket health expenditure            object
Physicians per thousand                    float64
Population                                  object
Population: Labor force participation (%)   object
Tax revenue (%)                             object
Total tax rate                              object
Unemployment rate                           object
Urban_population                            object
dtype: object
```

In [ ]:
```
#Changing columns data types

#remove commas from Density column
df['Density\n(P/Km2)'] = df['Density\n(P/Km2)'].str.replace(',', '')
df['Density\n(P/Km2)'] = df['Density\n(P/Km2)'].astype(float)

#remove (%) from Agricultural Land(%) column
df['Agricultural Land( %)'] = df['Agricultural Land( %)'].str.replace('%', '')
df['Agricultural Land( %)'] = df['Agricultural Land( %)'].astype(float)


#remove commas from Land Area(Km2) column
df['Land Area(Km2)'] = df['Land Area(Km2)'].str.replace(',', '')
df['Land Area(Km2)'] = df['Land Area(Km2)'].astype(float)

#remove commas from Armed Forces size column
df['Armed Forces size'] = df['Armed Forces size'].str.replace(',', '')
df['Armed Forces size'] = df['Armed Forces size'].astype(float)

#remove commas from Co2-Emissions column
df['Co2-Emissions'] = df['Co2-Emissions'].str.replace(',', '')
df['Co2-Emissions'] = df['Co2-Emissions'].astype(float)

#remove commas from CPI column
df['CPI'] = df['CPI'].str.replace(',', '')
df['CPI'] = df['CPI'].astype(float)

#remove "%" from CPI Change column
df['CPI Change (%)'] = df['CPI Change (%)'].str.replace('%', '')
df['CPI Change (%)'] = df['CPI Change (%)'].astype(float)
```

```
#remove "%" from Forested Area (%) column
df['Forested Area (%)'] = df['Forested Area (%)'].str.replace('%', '')
df['Forested Area (%)'] = df['Forested Area (%)'].astype(float)

#remove "$" from Gasoline Price column
df['Gasoline Price'] = df['Gasoline Price'].str.replace('$', '')
df['Gasoline Price'] = df['Gasoline Price'].astype(float)

#remove "$" from GDP Change column
df['GDP'] = df['GDP'].str.replace('$', '')
#remove commas from GDP Change column
df['GDP'] = df['GDP'].str.replace(',', '')
df['GDP'] = df['GDP'].astype(float)
#Pandas formats large number to scinetifc notation, therfore this converts back
pd.set_option('display.float_format', '{:.2f}'.format)

#remove "%" from Out of pocket health expenditure column
df['Out of pocket health expenditure'] = df['Out of pocket health expenditure'].str.rep
df['Out of pocket health expenditure'] = df['Out of pocket health expenditure'].astype(

#remove commas from Population column
df['Population'] = df['Population'].str.replace(',', '')
df['Population'] = df['Population'].astype(float)

#remove "%" from Population: Labor force participation (%) column
df['Population: Labor force participation (%)'] = df['Population: Labor force participa
df['Population: Labor force participation (%)'] = df['Population: Labor force participa

#remove "%" from Tax revenue (%) column
df['Tax revenue (%)'] = df['Tax revenue (%)'].str.replace('%', '')
df['Tax revenue (%)'] = df['Tax revenue (%)'].astype(float)

#remove "%" from Total tax rate column
df['Total tax rate'] = df['Total tax rate'].str.replace('%', '')
df['Total tax rate'] = df['Total tax rate'].astype(float)

#remove "%" from Unemployment rate column
df['Unemployment rate'] = df['Unemployment rate'].str.replace('%', '')
df['Unemployment rate'] = df['Unemployment rate'].astype(float)

#remove commas from Urban_population column
df['Urban_population'] = df['Urban_population'].str.replace(',', '')
df['Urban_population'] = df['Urban_population'].astype(float)

#turn life expentancy into into so it can be used as a TARGET variable
df['Life expectancy'] = df['Life expectancy'].astype(int)
```

```
C:\Users\almas\AppData\Local\Temp/ipykernel_8696/1912219860.py:37: FutureWarning: The de
fault value of regex will change from True to False in a future version. In addition, si
ngle character regular expressions will *not* be treated as literal strings when regex=T
rue.
  df['Gasoline Price'] = df['Gasoline Price'].str.replace('$', '')
C:\Users\almas\AppData\Local\Temp/ipykernel_8696/1912219860.py:41: FutureWarning: The de
fault value of regex will change from True to False in a future version. In addition, si
ngle character regular expressions will *not* be treated as literal strings when regex=T
rue.
  df['GDP'] = df['GDP'].str.replace('$', '')
```

All NUMERIC columns that were data type 'object' are now changed to 'float64'. Target Varible 'Life Expectancy' is changed into 'int32'

In [ ]:

```
df.dtypes
```

Out[ ]:

```
Country                                      object
Density\n(P/Km2)                            float64
Agricultural Land( %)                       float64
Land Area(Km2)                              float64
Armed Forces size                           float64
Birth Rate                                  float64
Co2-Emissions                               float64
CPI                                         float64
CPI Change (%)                              float64
Fertility Rate                              float64
Forested Area (%)                           float64
Gasoline Price                              float64
GDP                                         float64
Infant mortality                            float64
Life expectancy                               int32
Out of pocket health expenditure            float64
Physicians per thousand                     float64
Population                                  float64
Population: Labor force participation (%)   float64
Tax revenue (%)                             float64
Total tax rate                              float64
Unemployment rate                           float64
Urban_population                            float64
dtype: object
```

In [ ]:

```
df
```

Out[ ]:

	Country	Density\n(P/Km2)	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Co2-Emissions	CPI	Cha
0	Afghanistan	60.00	58.10	652230.00	323000.00	32.49	8672.00	149.90	
1	Albania	105.00	43.10	28748.00	9000.00	11.78	4536.00	119.05	
2	Algeria	18.00	17.40	2381741.00	317000.00	24.28	150006.00	151.36	
4	Angola	26.00	47.50	1246700.00	117000.00	40.73	34693.00	261.73	1
6	Argentina	17.00	54.30	2780400.00	105000.00	17.02	201348.00	232.75	5
...	...	...	...	...	...	...	...	...	
186	United States	36.00	44.40	9833517.00	1359000.00	11.60	5006302.00	117.24	
187	Uruguay	20.00	82.60	176215.00	22000.00	13.86	6766.00	202.92	
191	Vietnam	314.00	39.30	331210.00	522000.00	16.75	192668.00	163.52	
193	Zambia	25.00	32.10	752618.00	16000.00	36.19	5141.00	212.31	
194	Zimbabwe	38.00	41.90	390757.00	51000.00	30.68	10983.00	105.51	

146 rows × 23 columns

Visualizations

Correlation matrix in text and heatmap (to idenitfy relationships and to help with feature selection)

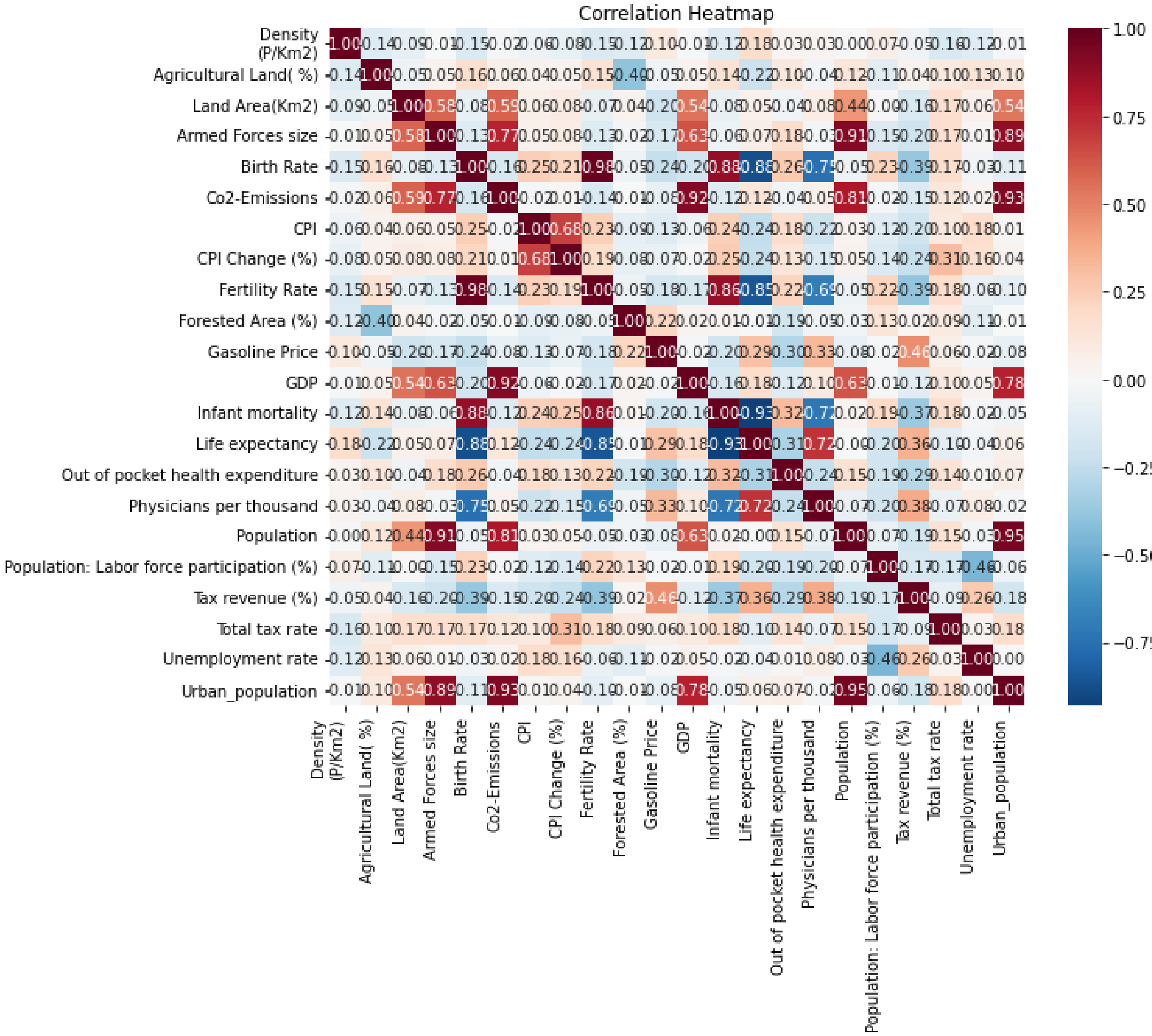
In [ ]:

```
#correlation matrix
correlation_matrix = df.corr()


plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='RdBu_r', center=0, fmt='.2f',
            xticklabels=correlation_matrix.columns, yticklabels=correlation_matrix.colu
plt.title('Correlation Heatmap')
plt.xticks(rotation=90, ha='right')
plt.yticks(rotation=0)


# Convert heatmap to text representaion
correlation_text = correlation_matrix.to_string(float_format="{:.2f}".format)

print("Correlation Heatmap (Text Representation):")
print(correlation_text)

plt.show()
```

```
Correlation Heatmap (Text Representation):
                                             Density\n(P/Km2)  Agricultural Land( %)  Land
Area(Km2)  Armed Forces size  Birth Rate  Co2-Emissions   CPI  CPI Change (%)  Fertility
Rate  Forested Area (%)  Gasoline Price   GDP  Infant mortality  Life expectancy  Out of
```

```
pocket health expenditure  Physicians per thousand  Population  Population: Labor force
participation (%)  Tax revenue (%)  Total tax rate  Unemployment rate  Urban_population
Density\n(P/Km2)                                                1.00                  -0.14
-0.09              -0.01       -0.15              -0.02 -0.06            -0.08            -0.1
5              -0.12            0.10 -0.01             -0.12            0.18
0.03                     0.03       0.00                                           0.07
-0.05          -0.16             -0.12              -0.01
Agricultural Land( %)                                          -0.14                   1.00
-0.05               0.05        0.16              0.06  0.04             0.05             0.1
5              -0.40           -0.05  0.05               0.14            -0.22
0.10                    -0.04       0.12                                          -0.11
0.04           0.10              0.13               0.10
Land Area(Km2)                                                 -0.09                  -0.05
1.00               0.58       -0.08              0.59  0.06             0.08            -0.07
0.04           -0.20  0.54              -0.08              0.05
-0.04                     0.08       0.44                                          -0.00
-0.16           0.17              0.06               0.54
Armed Forces size                                              -0.01                   0.05
0.58               1.00       -0.13              0.77  0.05             0.08            -0.13
-0.02           -0.17  0.63              -0.06              0.07
0.18                    -0.03       0.91                                          -0.15
-0.20           0.17              0.01               0.89
Birth Rate                                                     -0.15                   0.16
-0.08              -0.13        1.00              -0.16  0.25             0.21             0.9
8              -0.05           -0.24 -0.20               0.88            -0.88
0.26                    -0.75      -0.05                                           0.23
-0.39           0.17             -0.03              -0.11
Co2-Emissions                                                  -0.02                   0.06
0.59               0.77       -0.16              1.00 -0.02            0.01            -0.14
-0.01           -0.08  0.92              -0.12              0.12
-0.04                     0.05       0.81                                          -0.02
-0.15           0.12              0.02               0.93
CPI                                                            -0.06                   0.04
0.06               0.05        0.25             -0.02  1.00             0.68             0.23
-0.09           -0.13 -0.06               0.24             -0.24
0.18                    -0.22       0.03                                          -0.12
-0.20           0.10              0.18               0.01
CPI Change (%)                                                 -0.08                   0.05
0.08               0.08        0.21              0.01  0.68             1.00             0.19
-0.08           -0.07 -0.02               0.25             -0.24
0.13                    -0.15       0.05                                          -0.14
-0.24           0.31              0.16               0.04
Fertility Rate                                                 -0.15                   0.15
-0.07              -0.13        0.98              -0.14  0.23             0.19             1.0
0              -0.05           -0.18 -0.17               0.86            -0.85
0.22                    -0.69      -0.05                                           0.22
-0.39           0.18             -0.06              -0.10
Forested Area (%)                                              -0.12                  -0.40
0.04              -0.02       -0.05             -0.01 -0.09            -0.08            -0.05
1.00            0.22  0.02               0.01             -0.01
-0.19                     -0.05      -0.03                                           0.13
0.02            0.09             -0.11              -0.01
Gasoline Price                                                  0.10                  -0.05
-0.20              -0.17       -0.24              -0.08 -0.13            -0.07            -0.1
8               0.22            1.00 -0.02             -0.20            0.29
-0.30                     0.33      -0.08                                          -0.02
0.46            0.06             -0.02              -0.08
GDP                                                            -0.01                   0.05
0.54               0.63       -0.20              0.92 -0.06            -0.02            -0.17
0.02           -0.02  1.00              -0.16              0.18
```

```
-0.12                    0.10        0.63                                      -0.01
-0.12           0.10              0.05              0.78
Infant mortality                                    -0.12                   0.14
-0.08             -0.06        0.88          -0.12  0.24           0.25            0.8
6               0.01           -0.20 -0.16             1.00            -0.93
0.32                    -0.72       0.02                                        0.19
-0.37           0.18              -0.02              -0.05
Life expectancy                                      0.18                   -0.22
0.05              0.07       -0.88           0.12 -0.24           -0.24           -0.85
-0.01           0.29  0.18              -0.93              1.00
-0.31                    0.72       -0.00                                       -0.20
0.36            -0.10             -0.04              0.06
Out of pocket health expenditure                     0.03                   0.10
-0.04              0.18        0.26          -0.04  0.18           0.13            0.2
2               -0.19          -0.30 -0.12             0.32            -0.31
1.00                    -0.24       0.15                                        -0.19
-0.29           0.14              0.01              0.07
Physicians per thousand                              0.03                   -0.04
0.08             -0.03       -0.75           0.05 -0.22           -0.15           -0.69
-0.05           0.33  0.10              -0.72              0.72
-0.24                    1.00       -0.07                                       -0.20
0.38            -0.07             0.08              -0.02
Population                                           0.00                   0.12
0.44              0.91       -0.05           0.81  0.03            0.05           -0.05
-0.03           -0.08  0.63              0.02              -0.00
0.15                    -0.07       1.00                                        -0.07
-0.19           0.15              -0.03              0.95
Population: Labor force participation (%)            0.07                   -0.11
-0.00             -0.15        0.23          -0.02 -0.12           -0.14           0.2
2               0.13           -0.02 -0.01             0.19            -0.20
-0.19                    -0.20      -0.07                                        1.00
-0.17           -0.17             -0.46              -0.06
Tax revenue (%)                                     -0.05                   0.04
-0.16             -0.20        -0.39          -0.15 -0.20           -0.24          -0.3
9               0.02            0.46 -0.12             -0.37            0.36
-0.29                    0.38       -0.19                                       -0.17
1.00            -0.09             0.26              -0.18
Total tax rate                                      -0.16                   0.10
0.17              0.17        0.17           0.12  0.10            0.31            0.18
0.09            0.06  0.10              0.18              -0.10
0.14                    -0.07       0.15                                        -0.17
-0.09           1.00              0.03              0.18
Unemployment rate                                   -0.12                   0.13
0.06              0.01       -0.03           0.02  0.18            0.16           -0.06
-0.11           -0.02  0.05              -0.02              -0.04
0.01                    0.08       -0.03                                        -0.46
0.26            0.03              1.00              0.00
Urban_population                                    -0.01                   0.10
0.54              0.89       -0.11           0.93  0.01            0.04           -0.10
-0.01           -0.08  0.78              -0.05              0.06
0.07                    -0.02       0.95                                        -0.06
-0.18           0.18              0.00              1.00
```

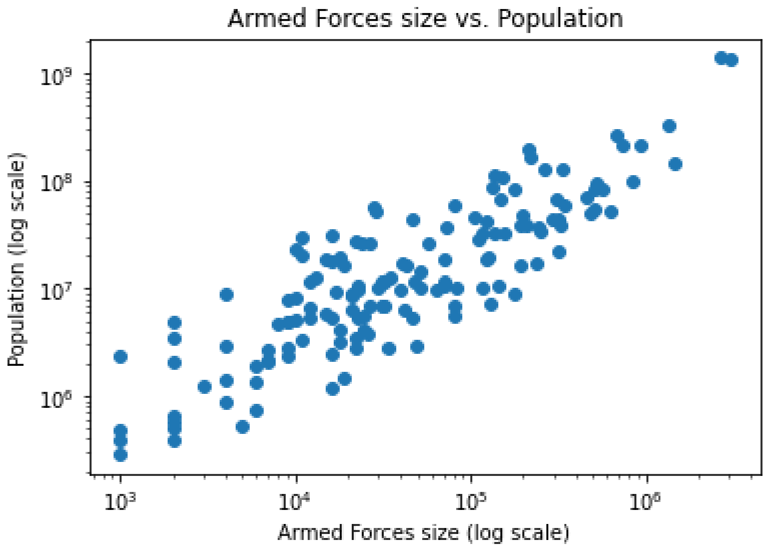


In [ ]:

```
#scatter plot with a logarithmic scale
plt.scatter(x=df['Armed Forces size'], y=df['Population'])
plt.xscale('log')
plt.yscale('log')

# labels
plt.xlabel('Armed Forces size (log scale)')
plt.ylabel('Population (log scale)')

plt.title('Armed Forces size vs. Population')
```

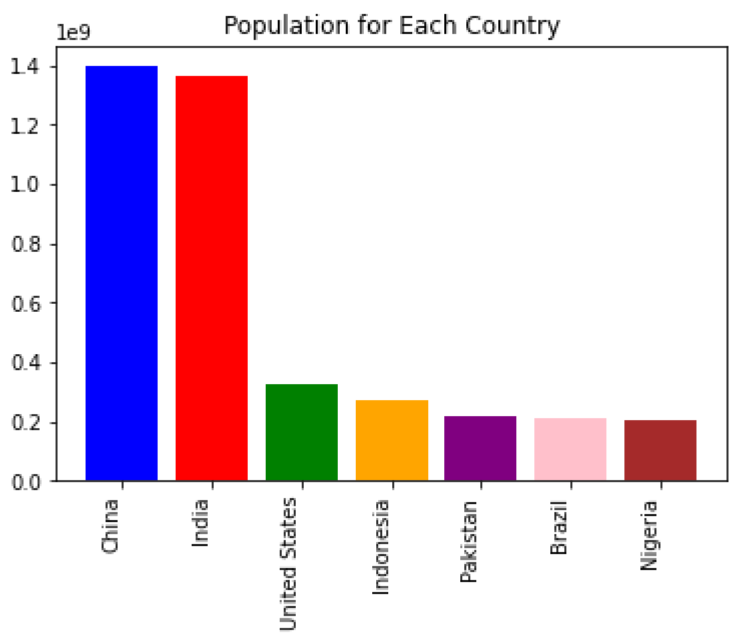
Out[ ]:

```
Text(0.5, 1.0, 'Armed Forces size vs. Population')
```

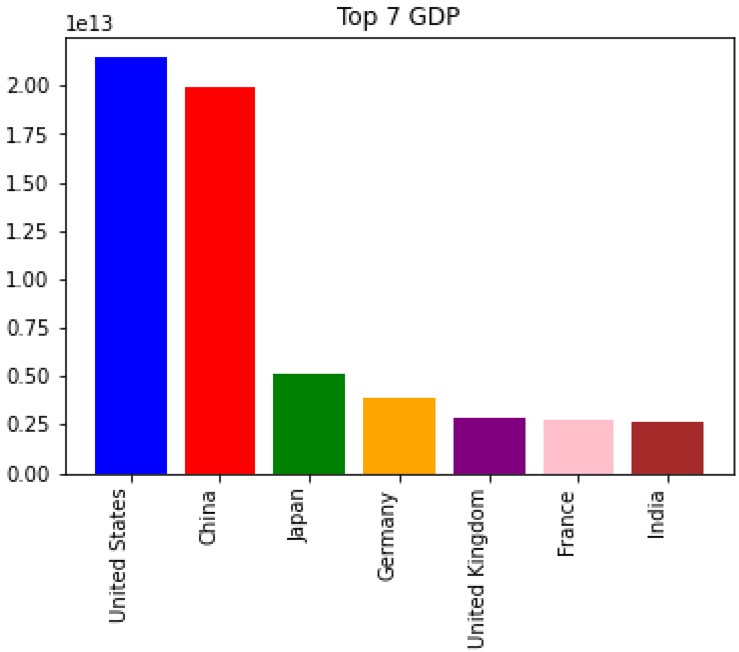


Bar graph of top 7 most populated countries

In [ ]:
```
plt.bar(df.sort_values(by="Population",ascending=False).head(7)["Country"],
        df.sort_values(by="Population",ascending=False).head(7)["Population"],
        color=['blue','red','green','orange','purple','pink','brown'])

plt.xticks(rotation=90, ha='right')
plt.title('Population for Each Country')
plt.show()
```

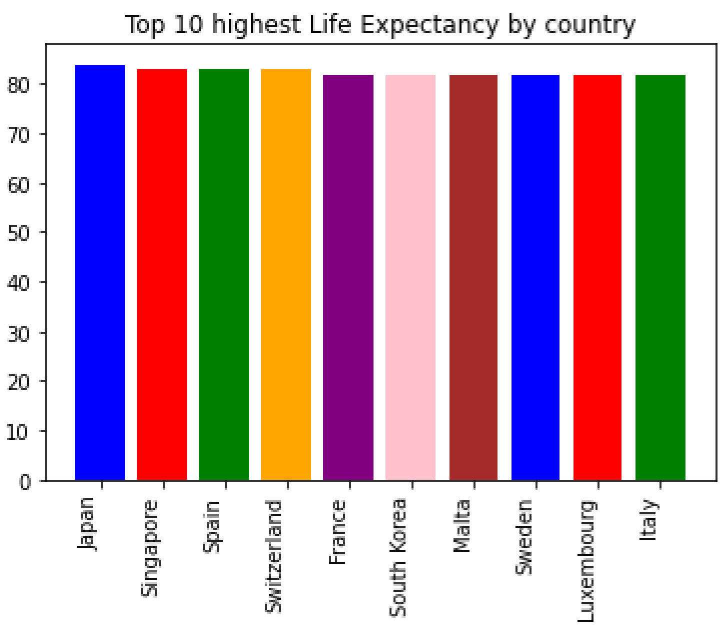


In [ ]:
```
plt.bar(df.sort_values(by="GDP",ascending=False).head(7)["Country"],
        df.sort_values(by="GDP",ascending=False).head(7)["GDP"],
        color=['blue','red','green','orange','purple','pink','brown'])

plt.xticks(rotation=90, ha='right')
plt.title('Top 7 GDP')
plt.show()
```



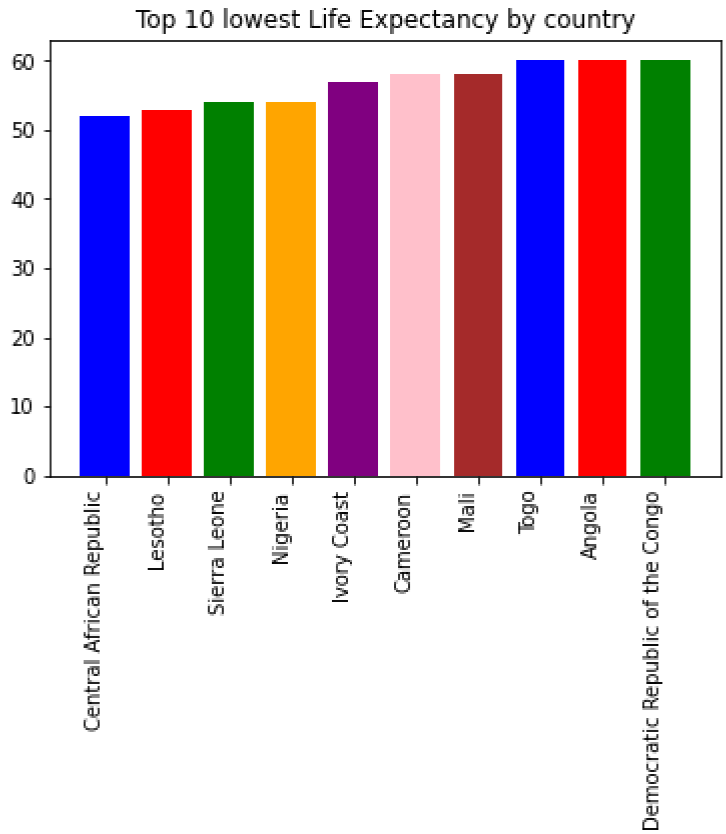
In [ ]:

```
plt.bar(df.sort_values(by="Life expectancy",ascending=False).head(10)["Country"],
        df.sort_values(by="Life expectancy",ascending=False).head(10)["Life expectancy"
        color=['blue','red','green','orange','purple','pink','brown'])

plt.xticks(rotation=90, ha='right')
plt.title('Top 10 highest Life Expectancy by country')

plt.show()
```



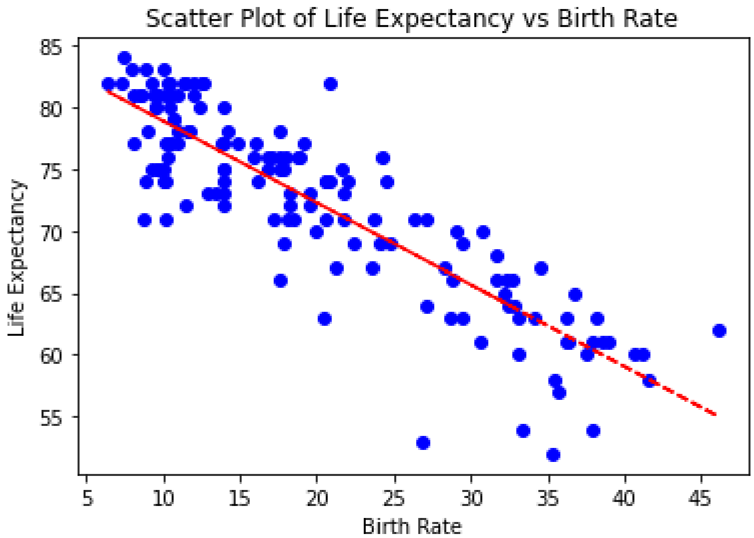
In [ ]:

```
plt.bar(df.sort_values(by="Life expectancy",ascending=True).head(10)["Country"],
        df.sort_values(by="Life expectancy",ascending=True).head(10)["Life expectancy"]
        color=['blue','red','green','orange','purple','pink','brown'])

plt.xticks(rotation=90, ha='right')
plt.title('Top 10 lowest Life Expectancy by country')
plt.show()
```



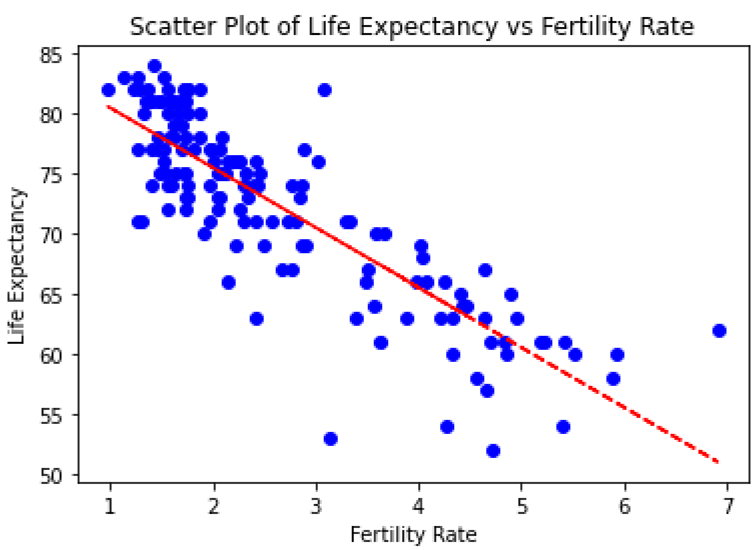
In [ ]:

```
birth_rate = df['Birth Rate']
life_expectancy = df['Life expectancy']

plt.scatter(birth_rate, life_expectancy, color='blue')
plt.xlabel('Birth Rate')
plt.ylabel('Life Expectancy')
plt.title('Scatter Plot of Life Expectancy vs Birth Rate')


# Add best fit line

#get coefficents for best fit
coefficients = np.polyfit(birth_rate, life_expectancy, 1)
#equation of best fit line
polynomial = np.poly1d(coefficients)
plt.plot(birth_rate, polynomial(birth_rate), color='red', linestyle='--', label='Line o

plt.show()
```



In [ ]:

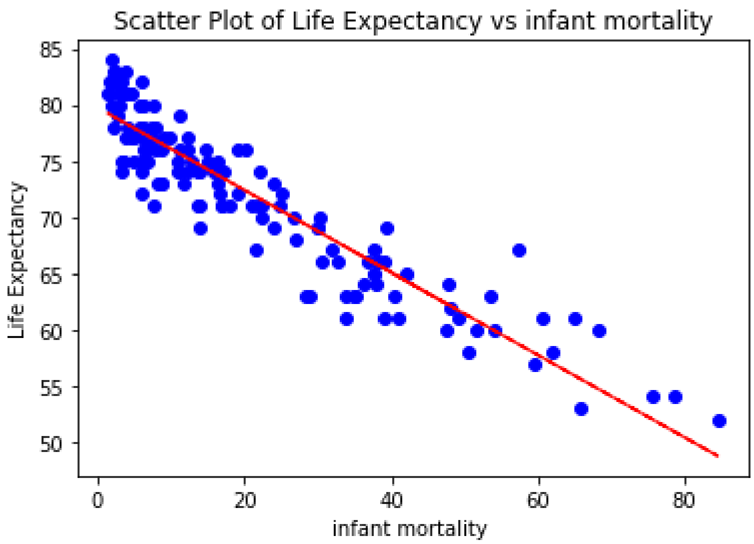
```
fertility_rate = df['Fertility Rate']
life_expectancy = df['Life expectancy']

plt.scatter(fertility_rate, life_expectancy, color='blue')
plt.xlabel('Fertility Rate')
plt.ylabel('Life Expectancy')
plt.title('Scatter Plot of Life Expectancy vs Fertility Rate')

# Add a line of best fit

coefficients = np.polyfit(fertility_rate, life_expectancy, 1)
polynomial = np.poly1d(coefficients)
plt.plot(fertility_rate, polynomial(fertility_rate), color='red', linestyle='--', label


plt.show()
```



In [ ]:

```
infant_mortality  = df['Infant mortality']
life_expectancy = df['Life expectancy']

plt.scatter(infant_mortality, life_expectancy, color='blue')
plt.xlabel('infant mortality ')
```

```
plt.ylabel('Life Expectancy')
plt.title('Scatter Plot of Life Expectancy vs infant mortality')

# Add best fit line
coefficients = np.polyfit(infant_mortality, life_expectancy, 1)
polynomial = np.poly1d(coefficients)
plt.plot(infant_mortality, polynomial(infant_mortality), color='red', linestyle='--')

plt.show()
```

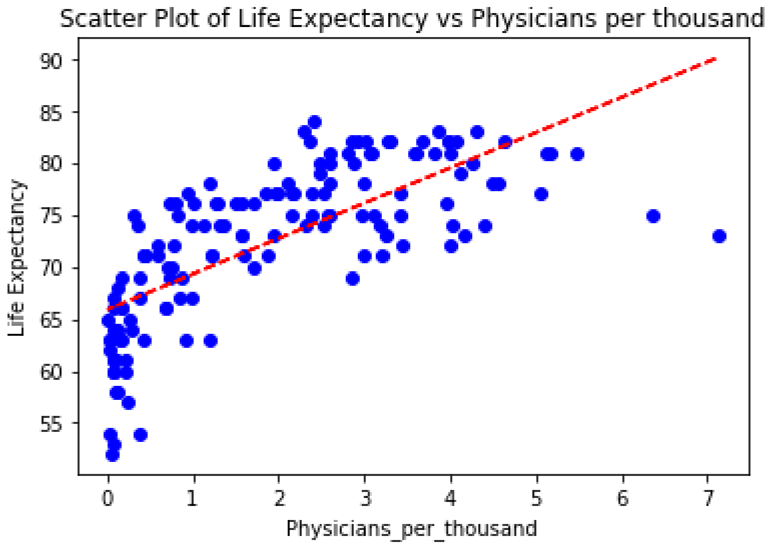


In [ ]:

```
Physicians_per_thousand  = df['Physicians per thousand']
life_expectancy = df['Life expectancy']

# Create a scatter plot
plt.scatter(Physicians_per_thousand, life_expectancy, color='blue')
plt.xlabel('Physicians_per_thousand')
plt.ylabel('Life Expectancy')
plt.title('Scatter Plot of Life Expectancy vs Physicians per thousand')


# Add a line of best fit
coefficients = np.polyfit(Physicians_per_thousand, life_expectancy, 1)
polynomial = np.poly1d(coefficients)
plt.plot(Physicians_per_thousand, polynomial(Physicians_per_thousand), color='red', lin
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x2120e4a3970>]
```



EDA, Clustering

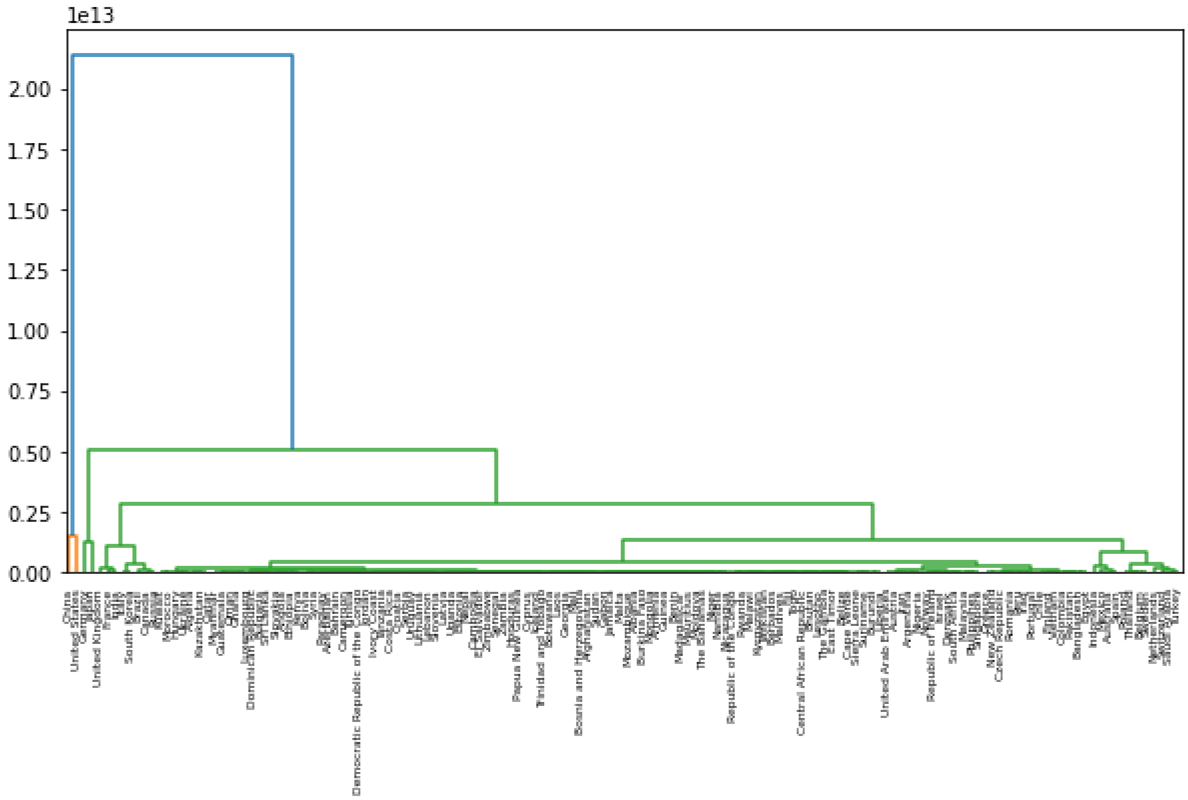
Hierarchical Clustering

In [ ]:
```
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize
```

We need to seperate Country col from the rest of the df

In [ ]:
```
# remove the 'Country' column from the DataFrame before clustering
numerical_features_df = df.drop(columns=['Country'])
country=df.Country.values
```

In [ ]:
```
mergings = linkage(numerical_features_df, method='complete')
plt.figure(figsize=(10, 5))

dendrogram(mergings,
           labels=country,
           leaf_rotation=90,
           leaf_font_size=6,
)
plt.show()
```



Feature Elimination

Recursive Feature Elimination Cross Validation to reduce features to find out the most important ones

In [ ]:

```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV
from sklearn.datasets import make_classification
%matplotlib inline
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
#max_iter to make sure algorithm reaches convergence
logit=LogisticRegression(multi_class='ovr',solver='lbfgs',max_iter=200000)
```

Set X and y to features and target

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X=df.drop(columns=['Country','Life expectancy'])
y=df['Life expectancy']

X
```

Out[ ]:

	Density\n(P/Km2)	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Co2-Emissions	CPI	CPI Change (%)	Fertility Rate
0	60.00	58.10	652230.00	323000.00	32.49	8672.00	149.90	2.30	4.47
1	105.00	43.10	28748.00	9000.00	11.78	4536.00	119.05	1.40	1.62
2	18.00	17.40	2381741.00	317000.00	24.28	150006.00	151.36	2.00	3.02
4	26.00	47.50	1246700.00	117000.00	40.73	34693.00	261.73	17.10	5.52
6	17.00	54.30	2780400.00	105000.00	17.02	201348.00	232.75	53.50	2.26
...	...	...	...	...	...	...	...	...	..
186	36.00	44.40	9833517.00	1359000.00	11.60	5006302.00	117.24	7.50	1.73
187	20.00	82.60	176215.00	22000.00	13.86	6766.00	202.92	7.90	1.97
191	314.00	39.30	331210.00	522000.00	16.75	192668.00	163.52	2.80	2.05
193	25.00	32.10	752618.00	16000.00	36.19	5141.00	212.31	9.20	4.63
194	38.00	41.90	390757.00	51000.00	30.68	10983.00	105.51	0.90	3.62

146 rows × 21 columns

RFECV without scaling X (we get a warning if we dont scale the data, beacuse the algorthim is having difficulty reaching ***convergance***)

In [ ]:

```
# Reduced number of splits and 'roc_auc' scoring
rfecv = RFECV(estimator=logit, step=1, cv=StratifiedKFold(10), scoring='accuracy')
rfecv.fit(X, y)
```

In [ ]:

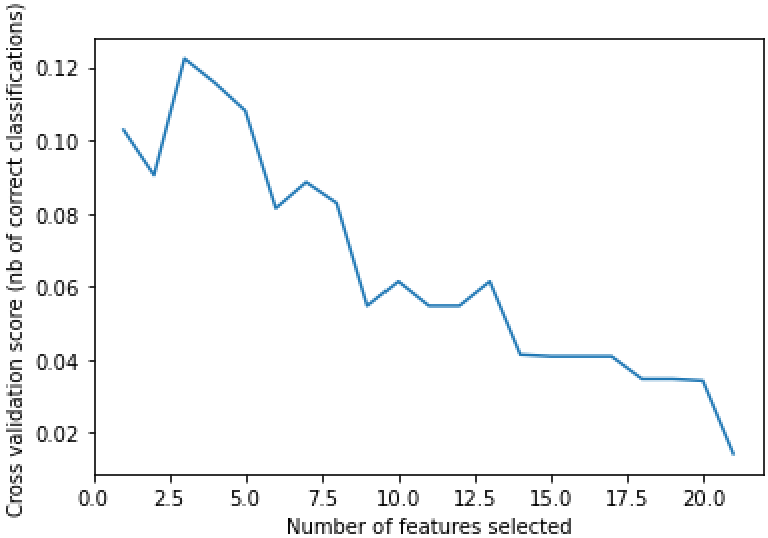
```
print(f"Optimal number of features : {rfecv.n_features_}")
```

```
Optimal number of features : 3
```

3 features yields the highest cross validation score

In [ ]:

```
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```



In [ ]:

```
print(rfecv.support_)
print("Selected Features: ",X.columns[rfecv.support_])
```

```
[False False False False  True False False False  True False False False
 False False  True False False False False False False]
Selected Features:  Index(['Birth Rate', 'Fertility Rate', 'Physicians per thousand'], d
type='object')
```

Ranking = 1 means that RFECV has chosen the feature

In [ ]:

```
rfecv.ranking_
```

Out[ ]:

```
array([12,  9, 15, 14,  1, 13, 11,  4,  1, 10, 18, 19,  2,  8,  1, 17,  5,
        6,  7,  3, 16])
```

RFECV with sacling X, to make sure the algorithm reaches convergence

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
rfecv = RFECV(estimator=logit, step=1, cv=StratifiedKFold(10), scoring='accuracy')
rfecv.fit(X_scaled, y)
```

```
c:\Users\almas\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: UserWa
rning: The least populated class in y has only 1 members, which is less than n_splits=1
0.
  warnings.warn(("The least populated class in y has only %d"
```

Out[ ]:

```
RFECV(cv=StratifiedKFold(n_splits=10, random_state=None, shuffle=False),
      estimator=LogisticRegression(max_iter=200000, multi_class='ovr'),
      scoring='accuracy')
```

In [ ]:

```
print(f"Optimal number of features : {rfecv.n_features_}")
selected_features= X.columns[rfecv.support_]
print("Selected Features: ",selected_features)

#create a df using the selected features
selected_features_df = X[selected_features]
selected_features_df
```

```
Optimal number of features : 16
Selected Features:  Index(['Agricultural Land( %)', 'Land Area(Km2)', 'Armed Forces siz
e',
       'Birth Rate', 'CPI', 'CPI Change (%)', 'Fertility Rate',
       'Forested Area (%)', 'Gasoline Price', 'Infant mortality',
       'Out of pocket health expenditure', 'Physicians per thousand',
       'Population: Labor force participation (%)', 'Tax revenue (%)',
       'Total tax rate', 'Unemployment rate'],
      dtype='object')
```

Out[ ]:

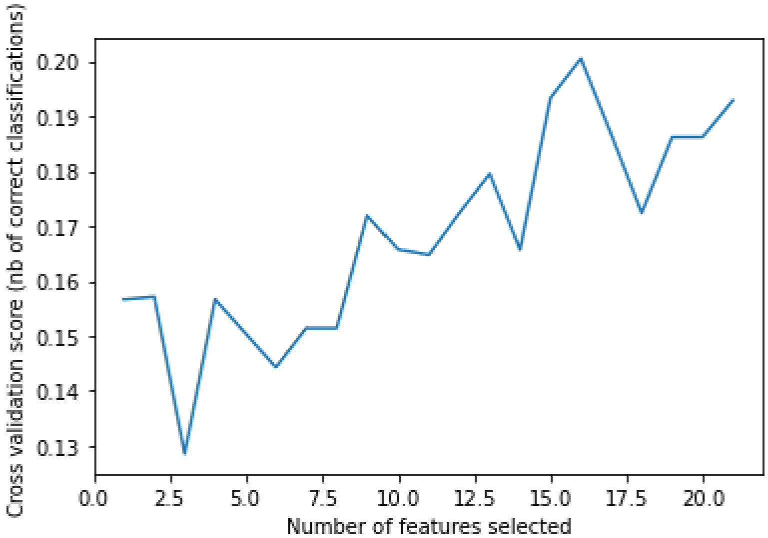
	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	CPI	CPI Change (%)	Fertility Rate	Forested Area (%)	Gasoline Price	Infant mortality
0	58.10	652230.00	323000.00	32.49	149.90	2.30	4.47	2.10	0.70	47.90
1	43.10	28748.00	9000.00	11.78	119.05	1.40	1.62	28.10	1.36	7.80
2	17.40	2381741.00	317000.00	24.28	151.36	2.00	3.02	0.80	0.28	20.10
4	47.50	1246700.00	117000.00	40.73	261.73	17.10	5.52	46.30	0.97	51.60
6	54.30	2780400.00	105000.00	17.02	232.75	53.50	2.26	9.80	1.10	8.80
...	...	...	...	...	...	...	...	...	...	...
186	44.40	9833517.00	1359000.00	11.60	117.24	7.50	1.73	33.90	0.71	5.60
187	82.60	176215.00	22000.00	13.86	202.92	7.90	1.97	10.70	1.50	6.40
191	39.30	331210.00	522000.00	16.75	163.52	2.80	2.05	48.10	0.80	16.50
193	32.10	752618.00	16000.00	36.19	212.31	9.20	4.63	65.20	1.40	40.40
194	41.90	390757.00	51000.00	30.68	105.51	0.90	3.62	35.50	1.34	33.90

146 rows × 16 columns

16 features yields the highest cross validation score

In [ ]:

```python
#plot CV score with number of features selected
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```



PCA

Dimensionality reduction algorithm on ***selcted_features_df***, to reduce the dimensions of the dataframe

In [ ]:

```
from sklearn.decomposition import PCA
pca_Object = PCA(n_components=16)
pca_Object.fit(X_scaled)
pca_X = pca_Object.transform(X)
```

In [ ]:

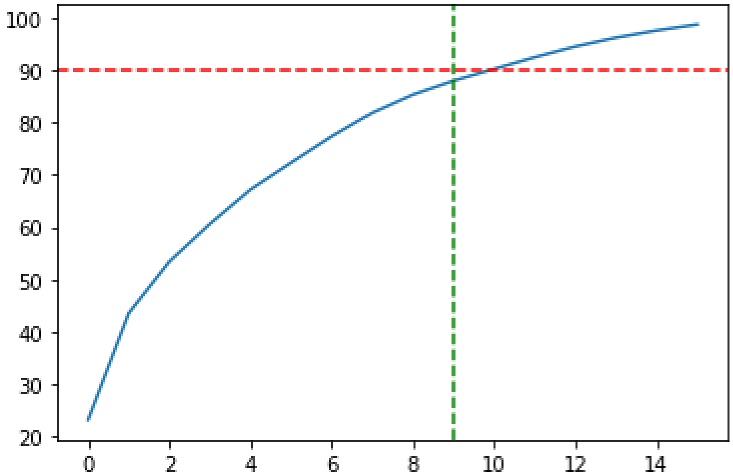
```
#The amount of variance that each PC explains
var_Data= pca_Object.explained_variance_ratio_
#Cumulative Variance explains
var1_Data=np.cumsum(np.round(pca_Object.explained_variance_ratio_, decimals=4)*100)
var1_Data
```

Out[ ]:

```
array([23.18, 43.54, 53.4 , 60.61, 67.18, 72.32, 77.35, 81.87, 85.34,
       87.98, 90.24, 92.47, 94.52, 96.21, 97.57, 98.71])
```

In [ ]:

```
plt.plot(var1_Data)
plt.axhline(y=90, color='red', linestyle='--')
plt.axvline(x=9, color='green', linestyle='--')
plt.show()
```



Selecting features that have > 90% of the variance (9 features in this case)

In [ ]:

```
from sklearn.decomposition import PCA
from sklearn import decomposition

pcaOBJ = decomposition.PCA(n_components=9)
pcaOBJ.fit(pca_X)
obj = pcaOBJ.transform(pca_X)
print("obj shape: ",obj.shape)
obj
```

```
obj shape:  (146, 9)
```

Out[ ]:

```
array([[-4.87493786e+11,  7.29694423e+06, -9.48837518e+04, ...,
        -1.97292447e+02, -1.57562916e+01, -4.40227242e+01],
       [-4.90576478e+11, -1.92718691e+07, -5.03016090e+05, ...,
        -1.34243206e+02, -2.79482985e+01, -9.88525772e+00],
       [-3.65685046e+11,  1.47602551e+07,  1.98171091e+06, ...,
        -1.47821908e+02, -2.85436096e+01, -1.51489105e+01],
       ...,
       [-2.91477272e+11,  4.78513204e+07, -6.13503980e+05, ...,
         1.94506550e+01, -8.08257122e+00,  1.49074173e+01],
       [-4.84291735e+11, -7.01658472e+06,  1.50381783e+05, ...,
        -1.82729399e+02,  5.22851105e+01,  3.78757973e+01],
       [-4.85602729e+11, -1.04848006e+07, -2.28985431e+05, ...,
        -1.91586820e+02, -5.09214783e+01,  6.35052490e+00]])
```

In [ ]:

```
import pandas as pd

# Convert the 'obj' array into a DataFrame
pca_df = pd.DataFrame(obj)

# Print the DataFrame containing the chosen principal components with updated column na
print(pca_df)
```

```
                    0           1           2          3         4  \
0    -487493786286.97    7296944.23   -94883.75  293155.97  25075.00
1    -490576478144.36  -19271869.06  -503016.09  -38965.69  -7349.40
2    -365685046327.28   14760255.08  1981710.91 -134068.24  19735.61
3    -426515000792.43    5499636.44   765716.05 -118533.82   -446.08
4    -139910852947.05    9360349.40  2375281.85 -338646.30 -10305.89
..                ...           ...         ...        ...       ...
141 16794992860626.67 -536659176.46 -1665092.48 3011280.81  40423.10
```

```
142  -457665749541.99  -19774800.46  -351530.98  -51741.36  -5544.30
143  -291477272304.45   47851320.39  -613503.98  224789.70  40564.44
144  -484291734611.70   -7016584.72   150381.78   67663.12 -13902.30
145  -485602729205.39  -10484800.58  -228985.43   83918.12  -7096.85

            5       6      7      8
0   -10631.59 -197.29 -15.76 -44.02
1    -2708.75 -134.24 -27.95  -9.89
2     -391.50 -147.82 -28.54 -15.15
3     6531.91 -170.07  89.28  11.03
4    16199.47 -105.59  57.68 -30.45
..        ...     ...    ...    ...
141  20006.63 -177.45  58.16  -8.67
142  -1615.29 -215.07  44.31 -36.88
143 -15125.34   19.45  -8.08  14.91
144   2864.06 -182.73  52.29  37.88
145  -2157.83 -191.59 -50.92   6.35

[146 rows x 9 columns]
```

Split data into train and test (for both orginal dataframe and PCA dataframe)

Split data

In [ ]:

```
#set X and Y
df.columns
df_X = df.drop(columns=['Life expectancy','Country'])
df_y=df['Life expectancy']
df_X
```

Out[ ]:

	Density\n(P/Km2)	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Co2-Emissions	CPI	CPI Change (%)	Fertility Rate
0	60.00	58.10	652230.00	323000.00	32.49	8672.00	149.90	2.30	4.47
1	105.00	43.10	28748.00	9000.00	11.78	4536.00	119.05	1.40	1.62
2	18.00	17.40	2381741.00	317000.00	24.28	150006.00	151.36	2.00	3.02
4	26.00	47.50	1246700.00	117000.00	40.73	34693.00	261.73	17.10	5.52
6	17.00	54.30	2780400.00	105000.00	17.02	201348.00	232.75	53.50	2.26
...	...	...	...	...	...	...	...	...	..
186	36.00	44.40	9833517.00	1359000.00	11.60	5006302.00	117.24	7.50	1.73
187	20.00	82.60	176215.00	22000.00	13.86	6766.00	202.92	7.90	1.97
191	314.00	39.30	331210.00	522000.00	16.75	192668.00	163.52	2.80	2.05
193	25.00	32.10	752618.00	16000.00	36.19	5141.00	212.31	9.20	4.63
194	38.00	41.90	390757.00	51000.00	30.68	10983.00	105.51	0.90	3.62

146 rows × 21 columns

In [ ]:
```
from sklearn.model_selection import train_test_split

#orginal df
df_X_train, df_X_test, df_y_train, df_y_test = train_test_split(df_X, df_y, test_size=0

#PCA df
pca_X_train, pca_X_test, pca_y_train, pca_y_test = train_test_split(pca_df, df_y, test_
```

In [ ]:
```
df_X_train
```

Out[ ]:

	Density\n(P/Km2)	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Co2-Emissions	CPI	CPI Change (%)	Fertility Rate
119	3.00	47.10	824292.00	16000.00	28.64	4228.00	157.97	3.70	3.40
82	400.00	24.60	20770.00	178000.00	20.80	65166.00	108.15	0.80	3.09
27	463.00	79.20	27830.00	31000.00	39.01	495.00	182.11	-0.70	5.41
6	17.00	54.30	2780400.00	105000.00	17.02	201348.00	232.75	53.50	2.26
69	53.00	59.00	245857.00	13000.00	36.36	2996.00	262.95	9.50	4.70
...	...	...	...	...	...	...	...	...	...
81	72.00	64.50	70273.00	9000.00	12.50	37711.00	106.58	0.90	1.75
88	94.00	48.50	580367.00	29000.00	28.75	17910.00	180.51	4.70	3.49
102	203.00	61.40	118484.00	15000.00	34.12	1298.00	418.34	9.40	4.21
13	1265.00	70.60	148460.00	221000.00	18.18	84246.00	179.68	5.60	2.04
122	508.00	53.30	41543.00	41000.00	9.70	170780.00	115.91	2.60	1.59

102 rows × 21 columns

In [ ]:
```
pca_X_train
```

Out[ ]:

	0	1	2	3	4	5	6	7	8
93	-492926879761.13	-19717685.69	256225.54	98781.60	-11653.80	-1115.61	-201.53	-0.06	-14.45
64	-183959199408.05	-27185565.61	-623347.54	-68353.35	18859.10	-6318.34	134.32	-43.21	-5.67
23	-500479450191.16	-13257115.23	-613016.44	93768.01	-8092.75	-3646.17	215.76	25.75	-31.44
4	-139910852947.05	9360349.40	2375281.85	-338646.30	-10305.89	16199.47	-105.59	57.68	-30.45
55	-491939923727.14	-11411869.72	-347960.33	30812.81	-10788.27	-372.50	-177.48	97.91	-16.82
...	...	...	...	...	...	...	...	...	...

	0	1	2	3	4	5	6	7	8
63	-189125853565.30	-32173212.42	-695049.22	66042.76	-7560.54	225.08	-172.24	-39.09	-25.86
70	-425818347608.60	15742854.47	-348130.07	326042.23	-22711.15	5840.07	-126.89	23.52	-17.46
81	-496722856174.43	-8073093.00	-583445.79	150327.81	-13331.71	-1680.22	-34.93	243.66	4.09
11	-258667874182.18	104258888.57	-1150506.61	265280.72	-14807.03	22589.15	995.19	37.40	-34.07
95	230953991129.07	-38835968.31	-872017.13	-73547.49	-3141.47	3172.27	254.38	-28.03	-19.89

102 rows × 9 columns

Regression learning algorithms

Linear Regression (Original DF)

In [ ]:

```
# import model
from sklearn.linear_model import LinearRegression

# instantiate
linreg = LinearRegression()

# fit the model to the training data (learn the coefficients)
linreg.fit(df_X_train, df_y_train)
```

Out[ ]:

```
LinearRegression()
```

In [ ]:

```
# print the intercept and coefficients
print(linreg.intercept_)
print(linreg.coef_)
```

```
85.4435090872766
[ 1.84055701e-04 -4.35098883e-02  3.47239592e-07  4.20157735e-06
  3.33298458e-02 -1.22994058e-05 -3.98249478e-03 -2.72815435e-02
 -1.26333909e+00 -4.05463409e-02  1.37548973e+00  2.60968148e-12
 -2.71481873e-01 -1.62726276e-02  3.41912389e-01  1.41881517e-08
 -5.08852319e-02 -2.35309539e-02  2.38153255e-02 -9.73249186e-02
 -2.19851545e-08]
```

In [ ]:

```
# make predictions on the testing set
df_y_pred = linreg.predict(df_X_test)
```

In [ ]:

```
from sklearn import metrics
df_lr_mae=metrics.mean_absolute_error(df_y_test, df_y_pred)
df_lr_mse=metrics.mean_squared_error(df_y_test, df_y_pred)
df_lr_rmse=np.sqrt(metrics.mean_squared_error(df_y_test, df_y_pred))

print('Mean Absoulute Error',df_lr_mae)
print('Mean Squared Error:',df_lr_mse)
print('Root Mean Squared Error: ',df_lr_rmse)
```

```
Mean Absoulute Error 3.2640263593722225
Mean Squared Error: 73.34989071379162
Root Mean Squared Error:  8.564455073954887
```

Linear Regression (PCA DF)

In [ ]:
```
# instantiate
linreg = LinearRegression()

# fit the model to the training data (learn the coefficients)
linreg.fit(pca_X_train, pca_y_train)
```

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
# print the intercept and coefficients
print(linreg.intercept_)
print(linreg.coef_)
```

```
71.60813416620944
[ 3.70777203e-13 -1.36824062e-08 -2.05564542e-07 -1.79670327e-09
  5.01305770e-05  2.22923475e-05  1.81176697e-03 -6.89317500e-02
  1.81977457e-02]
```

In [ ]:
```
# make predictions on the testing set
pca_y_pred = linreg.predict(pca_X_test)
```

In [ ]:
```
from sklearn import metrics
pca_lr_mae=metrics.mean_absolute_error(pca_y_test, pca_y_pred)
pca_lr_mse=metrics.mean_squared_error(pca_y_test, pca_y_pred)
pca_lr_rmse=np.sqrt(metrics.mean_squared_error(pca_y_test, pca_y_pred))

print('Mean Absoulute Error',pca_lr_mae)
print('Mean Squared Error:',pca_lr_mse)
print('Root Mean Squared Error: ',pca_lr_rmse)
```

```
Mean Absoulute Error 7.031591494863505
Mean Squared Error: 169.2208982835762
Root Mean Squared Error:  13.008493313354018
```

SVM (Original DF)

Without scaling

In [ ]:
```
# Fitting SVR to the dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf',gamma='auto')
regressor.fit(df_X_train, df_y_train)
```

Out[ ]:
```
SVR(gamma='auto')
```

In [ ]:
```
df_y_pred_no_scaling = regressor.predict(df_X_test)
```

In [ ]:

```
predictions = [[206,43.2,301340,347000,7.3,320411,110.62,0.6,1.29,31.8,1.61,20012443920

# Scale the prediction data
scaler_X = StandardScaler()
scaler_X.fit(predictions)
scaled_prediction = scaler_X.transform(predictions)

# Predict using the scaled input
predicted_value = regressor.predict(scaled_prediction)

# Print the predicted value
print("Predicted Life expectancy:", predicted_value)
print('Actual Life expectancy: ',)
```

```
Predicted Life expectancy: [74.04545455]
Actual Life expectancy: 82
 [74.04545455]
Actual Life expectancy: 82
```

In [ ]:

```
print('Mean Absoulute Error',metrics.mean_absolute_error(df_y_test, df_y_pred_no_scalin
print('Mean Squared Error:',metrics.mean_squared_error(df_y_test, df_y_pred_no_scaling)
print('Root Mean Squared Error: ',np.sqrt(metrics.mean_squared_error(df_y_test, df_y_pr
```

```
Mean Absoulute Error 5.890495867768595
Mean Squared Error: 55.81611570247933
Root Mean Squared Error:  7.471018384563067
```

SVM with Scaling

In [ ]:

```
#scaling
from sklearn.preprocessing import StandardScaler
scaler_X = StandardScaler()
df_X_scaled = scaler_X.fit_transform(df_X_train)
```

In [ ]:

```
#create and fit SVR using scaled values
regressor = SVR(kernel='rbf', gamma='auto')
regressor.fit(X_scaled, y)
```

Out[ ]:

```
SVR(gamma='auto')
```

In [ ]:

```
predictions = [[206,43.2,301340,347000,7.3,320411,110.62,0.6,1.29,31.8,1.61,20012443920

# Scale the prediction data
scaled_prediction = scaler_X.transform(predictions)

# Predict using scaled input
predicted_value = regressor.predict(scaled_prediction)

# Print the predicted value
print("Predicted Life expectancy:", predicted_value)
print('Actual Life expectancy: 82')
```

```
Predicted Life expectancy: [80.88365092]
Actual Life expectancy: 82
```

In [ ]:
```
print('Mean Absoulute Error',metrics.mean_absolute_error(df_y_test, df_y_pred))
print('Mean Squared Error:',metrics.mean_squared_error(df_y_test, df_y_pred))
print('Root Mean Squared Error: ',np.sqrt(metrics.mean_squared_error(df_y_test, df_y_pr
```

```
Mean Absoulute Error 3.2640263593722225
Mean Squared Error: 73.34989071379162
Root Mean Squared Error:  8.564455073954887
```

SVM (PCA df)

Without scaling

In [ ]:
```
regressor = SVR(kernel = 'rbf',gamma='auto')
regressor.fit(pca_X_train, pca_y_train)
```

Out[ ]:
```
SVR(gamma='auto')
```

In [ ]:
```
pca_y_pred_no_scaling = regressor.predict(pca_X_test)
```

In [ ]:
```
pca_predictions = [[-258667874182.18,104258888.57,-1150506.61,265280.72,-14807.03,22589

# Scale the prediction data
scaler_X = StandardScaler()
scaler_X.fit(pca_X_train)
pca_scaled_prediction = scaler_X.transform(pca_predictions)

# Predict using the scaled input
pca_predicted_value = regressor.predict(pca_scaled_prediction)

# Print the predicted value
print("Predicted Life expectancy:", pca_predicted_value)
print('Actual Life expectancy: 82')
```

```
Predicted Life expectancy: [74.04545455]
Actual Life expectancy: 82
```

In [ ]:
```
pca_svm_mae=metrics.mean_absolute_error(pca_y_test, pca_y_pred)
pca_svm_mse=metrics.mean_squared_error(pca_y_test, pca_y_pred)
pca_svm_rmse=np.sqrt(metrics.mean_squared_error(pca_y_test, pca_y_pred))
print('Mean Absoulute Error',pca_svm_mae)
print('Mean Squared Error:',pca_svm_mse)
print('Root Mean Squared Error: ',pca_svm_rmse)
```

```
Mean Absoulute Error 7.031591494863505
Mean Squared Error: 169.2208982835762
Root Mean Squared Error:  13.008493313354018
```

With scaling

In [ ]:
```
#scaling
from sklearn.preprocessing import StandardScaler
scaler_X = StandardScaler()
pca_X_scaled = scaler_X.fit_transform(pca_X_train)
```

In [ ]:
```
#create and fit SVR using scaled values
regressor = SVR(kernel='rbf', gamma='auto')
regressor.fit(pca_X_scaled, pca_y_train)
```

Out[ ]:
```
SVR(gamma='auto')
```

In [ ]:
```
pca_predictions = [[-258667874182.18,104258888.57,-1150506.61,265280.72,-14807.03,22589

# Scale the prediction data
scaler_X = StandardScaler()
scaler_X.fit(pca_X_train)
pca_scaled_prediction = scaler_X.transform(pca_predictions)

# Predict using the scaled input
pca_predicted_value = regressor.predict(pca_scaled_prediction)

# Print the predicted value
print("Predicted Life expectancy:", pca_predicted_value)
print('Actual Life expectancy: 80')
```

```
Predicted Life expectancy: [71.62540858]
Actual Life expectancy: 80
```

In [ ]:
```
pca_svmS_mae=metrics.mean_absolute_error(pca_y_test, pca_y_pred)
pca_svmS_mse=metrics.mean_squared_error(pca_y_test, pca_y_pred)
pca_svmS_rmse=np.sqrt(metrics.mean_squared_error(pca_y_test, pca_y_pred))
print('Mean Absoulute Error',pca_svmS_mae)
print('Mean Squared Error:',pca_svmS_mse)
print('Root Mean Squared Error: ',pca_svmS_rmse)
```

```
Mean Absoulute Error 7.031591494863505
Mean Squared Error: 169.2208982835762
Root Mean Squared Error:  13.008493313354018
```

KNN Regression (Original DF)

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score

knn_regressor = KNeighborsRegressor(n_neighbors=5)
#Use scaled X
knn_regressor.fit(df_X_train, df_y_train)
```

Out[ ]:
```
KNeighborsRegressor()
```

In [ ]:
```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor

k_range = list(range(1, 31))
k_scores_MAE = []
k_scores_MSE = []
k_scores_RMSE = []
min_k_MAE = k_range[0]
min_k_MSE = k_range[0]
min_k_RMSE = k_range[0]
```

```
for k in k_range:
    knn = KNeighborsRegressor(n_neighbors=k)
    # Use 'neg_mean_absolute_error' as the scoring metric
    scores_MAE = cross_val_score(knn, df_X_train, df_y_train, cv=10, scoring='neg_mean_
    k_scores_MAE.append(-scores_MAE.mean())

for k in k_range:
    knn = KNeighborsRegressor(n_neighbors=k)
    # Use 'neg_mean_squared_error' as the scoring metric
    scores_MSE = cross_val_score(knn, df_X_train, df_y_train, cv=10, scoring='neg_mean_
    k_scores_MSE.append(-scores_MSE.mean())

for k in k_range:
    knn = KNeighborsRegressor(n_neighbors=k)
    # Use 'neg_root_mean_squared_error' as the scoring metric
    scores_RMSE = cross_val_score(knn, df_X_train, df_y_train, cv=10, scoring='neg_root
    k_scores_RMSE.append(-scores_RMSE.mean())

k_scores_MAE.sort()
k_scores_MSE.sort()
k_scores_RMSE.sort()



print('Minimum MAE:', k_scores_MAE[0], 'at k =', min_k_MAE)
print('Minimum MSE:', k_scores_MSE[0], 'at k =', min_k_MSE)
print('Minimum RMSE:', k_scores_RMSE[0], 'at k =', min_k_RMSE)
```

```
Minimum MAE: 5.559870129870131 at k = 1
Minimum MSE: 46.58687899402185 at k = 1
Minimum RMSE: 6.604713443110066 at k = 1
```

KNN Regression (PCA df)

In [ ]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score

knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(pca_X_train, pca_y_train)
```

Out[ ]:

```
KNeighborsRegressor()
```

In [ ]:

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor

k_range = list(range(1, 31))
pca_k_scores_MAE = []
pca_k_scores_MSE = []
pca_k_scores_RMSE = []
min_k_MAE = k_range[0]
min_k_MSE = k_range[0]
min_k_RMSE = k_range[0]
for k in k_range:
    knn = KNeighborsRegressor(n_neighbors=k)
    # Use 'neg_mean_absolute_error' as the scoring metric
    scores_MAE = cross_val_score(knn, pca_X_train, pca_y_train, cv=10, scoring='neg_mea
    pca_k_scores_MAE.append(-scores_MAE.mean())
```

```
for k in k_range:
    knn = KNeighborsRegressor(n_neighbors=k)
    # Use 'neg_mean_squared_error' as the scoring metric
    scores_MSE = cross_val_score(knn, pca_X_train, pca_y_train, cv=10, scoring='neg_mea
    pca_k_scores_MSE.append(-scores_MSE.mean())

for k in k_range:
    knn = KNeighborsRegressor(n_neighbors=k)
    # Use 'neg_root_mean_squared_error' as the scoring metric
    scores_RMSE = cross_val_score(knn, pca_X_train, pca_y_train, cv=10, scoring='neg_ro
    pca_k_scores_RMSE.append(-scores_RMSE.mean())

pca_k_scores_MAE.sort()
pca_k_scores_MSE.sort()
pca_k_scores_RMSE.sort()


print("MAE scores:")
for k, score in zip(k_range, k_scores_MAE):
    print("k =", k, "MAE =", score)

print("\nMSE scores:")
for k, score in zip(k_range, k_scores_MSE):
    print("k =", k, "MSE =", score)

print("\nRMSE scores:")
for k, score in zip(k_range, k_scores_RMSE):
    print("k =", k, "RMSE =", score)
```

```
MAE scores:
k = 1 MAE = 5.559870129870131
k = 2 MAE = 5.577090909090909
k = 3 MAE = 5.58603305785124
k = 4 MAE = 5.5901010101010105
k = 5 MAE = 5.60267942583732
k = 6 MAE = 5.618636363636363
k = 7 MAE = 5.630121212121212
k = 8 MAE = 5.634743083003952
k = 9 MAE = 5.646420454545455
k = 10 MAE = 5.6551636363636355
k = 11 MAE = 5.65979020979021
k = 12 MAE = 5.661643356643357
k = 13 MAE = 5.661711229946524
k = 14 MAE = 5.671598746081505
k = 15 MAE = 5.685422077922078
k = 16 MAE = 5.687077922077922
k = 17 MAE = 5.688212121212121
k = 18 MAE = 5.690033670033669
k = 19 MAE = 5.7524999999999995
k = 20 MAE = 5.7841414141414145
k = 21 MAE = 5.797909090909091
k = 22 MAE = 5.828512396694215
k = 23 MAE = 5.880795454545455
k = 24 MAE = 5.945454545454545
k = 25 MAE = 5.991363636363635
k = 26 MAE = 6.083090909090909
k = 27 MAE = 6.287954545454546
k = 28 MAE = 6.824545454545455
k = 29 MAE = 7.129545454545455
k = 30 MAE = 7.422727272727272
```

```
MSE scores:
k = 1 MSE = 46.58687899402185
k = 2 MSE = 46.69948181818183
k = 3 MSE = 46.777253787878784
k = 4 MSE = 46.85133358377159
k = 5 MSE = 46.86868446654417
k = 6 MSE = 47.07009548144164
k = 7 MSE = 47.08132799999999
k = 8 MSE = 47.1026393939394
k = 9 MSE = 47.14599345253086
k = 10 MSE = 47.21397379406308
k = 11 MSE = 47.361157251527615
k = 12 MSE = 47.44026936026937
k = 13 MSE = 47.47800481182334
k = 14 MSE = 47.91210443535704
k = 15 MSE = 48.136615767045456
k = 16 MSE = 48.20710707070707
k = 17 MSE = 48.71043033889187
k = 18 MSE = 49.41036641929499
k = 19 MSE = 49.82027146464647
k = 20 MSE = 50.300681818181815
k = 21 MSE = 50.63913580246914
k = 22 MSE = 51.09359128474831
k = 23 MSE = 51.74509943181819
k = 24 MSE = 52.9969573283859
k = 25 MSE = 54.05542929292928
k = 26 MSE = 57.44425454545454
k = 27 MSE = 61.12335227272727
k = 28 MSE = 69.9249494949495
k = 29 MSE = 76.77386363636363
k = 30 MSE = 90.9009090909091

RMSE scores:
k = 1 RMSE = 6.604713443110066
k = 2 RMSE = 6.6177688685869995
k = 3 RMSE = 6.621117640262172
k = 4 RMSE = 6.626885443241572
k = 5 RMSE = 6.649646223739947
k = 6 RMSE = 6.649748701978079
k = 7 RMSE = 6.653014433461443
k = 8 RMSE = 6.6591523090502465
k = 9 RMSE = 6.668777816434617
k = 10 RMSE = 6.670497428290341
k = 11 RMSE = 6.676416612257256
k = 12 RMSE = 6.678124672699113
k = 13 RMSE = 6.685631333103286
k = 14 RMSE = 6.720406837360675
k = 15 RMSE = 6.748622374986992
k = 16 RMSE = 6.760062472606554
k = 17 RMSE = 6.787917913681817
k = 18 RMSE = 6.83219262360767
k = 19 RMSE = 6.869665695377165
k = 20 RMSE = 6.887678949578192
k = 21 RMSE = 6.888104366033588
k = 22 RMSE = 6.951764087690283
k = 23 RMSE = 6.952990742983587
k = 24 RMSE = 7.053922891738419
k = 25 RMSE = 7.111311505688965
k = 26 RMSE = 7.320246818563701
```

```
k = 27 RMSE = 7.5610146161599605
k = 28 RMSE = 8.098451858996993
k = 29 RMSE = 8.405167919718448
k = 30 RMSE = 9.098034490730683
```

Decision Tree (Original DF)

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
dtree = DecisionTreeClassifier()
```

In [ ]:
```
dtree.fit(df_X_train,df_y_train)
```

Out[ ]:
```
DecisionTreeClassifier()
```

In [ ]:
```
df_y_pred_dt = dtree.predict(df_X_test)
```

In [ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explaine

df_dt_mae = mean_absolute_error(df_y_test, df_y_pred_dt)
df_dt_mse = mean_squared_error(df_y_test, df_y_pred_dt)
df_dt_rmse = np.sqrt(df_dt_mse)
df_dt_r2 = r2_score(df_y_test, df_y_pred_dt)
df_dt_ev = explained_variance_score(df_y_test, df_y_pred_dt)
print("Mean Absolute Error:", df_dt_mae)
print("Mean Squared Error:", df_dt_mse)
print("Root Mean Squared Error:", df_dt_rmse)
print("R-squared:", df_dt_r2)
print("Explained Variance Score:", df_dt_ev)
```

```
Mean Absolute Error: 3.0
Mean Squared Error: 16.045454545454547
Root Mean Squared Error: 4.005677788521506
R-squared: 0.7042509639643928
Explained Variance Score: 0.7088589517779789
```

Decision Tree (PCA DF)

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
dtree = DecisionTreeClassifier()
```

In [ ]:
```
dtree.fit(pca_X_train,pca_y_train)
```

Out[ ]:
```
DecisionTreeClassifier()
```

In [ ]:
```
pca_y_pred_dt = dtree.predict(pca_X_test)
```

In [ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explaine

pca_dt_mae = mean_absolute_error(pca_y_test, pca_y_pred_dt)
pca_dt_mse = mean_squared_error(pca_y_test, pca_y_pred_dt)
pca_dt_rmse = np.sqrt(pca_dt_mse)
pca_dt_r2 = r2_score(pca_y_test, pca_y_pred_dt)
pca_dt_ev = explained_variance_score(pca_y_test, pca_y_pred_dt)
print("Mean Absolute Error:", pca_dt_mae)
print("Mean Squared Error:", pca_dt_mse)
print("Root Mean Squared Error:", pca_dt_rmse)
print("R-squared:", pca_dt_r2)
print("Explained Variance Score:", pca_dt_ev)
```

```
Mean Absolute Error: 4.5
Mean Squared Error: 37.09090909090909
Root Mean Squared Error: 6.0902306270706275
R-squared: 0.3163421716570667
Explained Variance Score: 0.3578140619793402
```

Random Forest (orgainal DF)

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
rfc = RandomForestClassifier(n_estimators=600)
```

In [ ]:
```
rfc.fit(df_X_train,df_y_train)
```

Out[ ]:
```
RandomForestClassifier(n_estimators=600)
```

In [ ]:
```
predictions_rf = rfc.predict(df_X_test)
```

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explaine

mae = mean_absolute_error(df_y_test, predictions_rf)
mse = mean_squared_error(df_y_test, predictions_rf)
rmse = np.sqrt(mse)
r2 = r2_score(df_y_test, predictions_rf)
ev = explained_variance_score(df_y_test, predictions_rf)
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
print("Explained Variance Score:", ev)
```

```
Mean Absolute Error: 2.0454545454545454
Mean Squared Error: 7.863636363636363
Root Mean Squared Error: 2.8042176027613057
R-squared: 0.8550578378635693
Explained Variance Score: 0.8564288094444709
```

Random Forest (PCA DF)

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
rfc = RandomForestClassifier(n_estimators=600)
```

In [ ]:
```
rfc.fit(pca_X_train,pca_y_train)
```

Out[ ]:
```
RandomForestClassifier(n_estimators=600)
```

In [ ]:
```
predictions_rf = rfc.predict(pca_X_test)
```

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explaine

mae = mean_absolute_error(pca_y_test, predictions_rf)
mse = mean_squared_error(pca_y_test, predictions_rf)
rmse = np.sqrt(mse)
r2 = r2_score(pca_y_test, predictions_rf)
ev = explained_variance_score(pca_y_test, predictions_rf)
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
print("Explained Variance Score:", ev)
```

```
Mean Absolute Error: 3.7045454545454546
Mean Squared Error: 25.113636363636363
Root Mean Squared Error: 5.011350752405619
R-squared: 0.5371066787261389
Explained Variance Score: 0.543057076212691
```

Regression learning algorithms Summary

In [ ]:
```
import pandas as pd
summary = {
    'Model': ['Linear Regression (ODF)', 'Linear Regression (PCA DF)', 'SVM (ODF NO SCA
              'SVM (PCA DF With SCALING)', 'KNN (ODF)', 'KNN (PCA DF)', 'DT (ODF)', 'DT
    'MAE': [3.26, 7.03, 5.89, 3.26, 7.03, 7.03, 5.56, 5.56, 3.00, 4.50, 2.05, 3.70],
    'MSE': [73.35, 169.22, 55.82, 73.35, 169.22, 169.22, 46.59, 46.78, 16.05, 37.09, 7.
    'RMSE': [8.56, 13.01, 7.47, 8.56, 13.01, 13.01, 6.60, 6.62, 4.01, 6.09, 2.80, 5.01]
}



df = pd.DataFrame(summary)
df
```

Out[ ]:

	Model	MAE	MSE	RMSE
0	Linear Regression (ODF)	3.26	73.35	8.56

	Model	MAE	MSE	RMSE
1	Linear Regression (PCA DF)	7.03	169.22	13.01
2	SVM (ODF NO SCALING)	5.89	55.82	7.47
3	SVM (ODF With SCALING)	3.26	73.35	8.56
4	SVM (PCA DF NO SCALING)	7.03	169.22	13.01
5	SVM (PCA DF With SCALING)	7.03	169.22	13.01
6	KNN (ODF)	5.56	46.59	6.60
7	KNN (PCA DF)	5.56	46.78	6.62
8	DT (ODF)	3.00	16.05	4.01
9	DT (PCA DF)	4.50	37.09	6.09
10	Random Forest (ODF)	2.05	7.86	2.80
11	Random Forest (PCA DF)	3.70	25.11	5.01

Report

Top 2 algorithms

Random Forest (Original DF):

The Random Forest algorithm applied to the original DataFrame demonstrates outstanding performance in predicting life expectancy. With a Mean Absolute Error (MAE) of 2.05 and a Root Mean Squared Error (RMSE) of 2.80, the model provides highly accurate predictions. The low MAE and RMSE scores indicate that the model's predictions are, on average, very close to the actual life expectancy values. The strength of Random Forest lies in its ability to handle complex relationships within the data, making it robust against overfitting. By combining multiple decision trees and averaging their outputs, Random Forest reduces the risk of individual tree biases and increases overall prediction accuracy. This makes it a powerful and reliable choice for life expectancy prediction.

Decision Tree (Original DF):

The Decision Tree algorithm, when applied to the original DataFrame, also exhibits commendable performance in predicting life expectancy. With a Mean Absolute Error (MAE) of 3.0 and a Root Mean Squared Error (RMSE) of 4.01, the model delivers relatively accurate predictions. Decision Trees are easy to understand and interpret, making them valuable for gaining insights into feature importance and the decision-making process. However, compared to Random Forest, Decision Trees might be more prone to overfitting, especially on complex datasets. Nonetheless, the model's performance is still satisfactory, and its simplicity and interpretability make it an attractive option for scenarios where model interpretability is of extreme importance or when dealing with smaller datasets.

The rest of the algorithms

The remaining algorithms, including Linear Regression and SVM, show varying levels of performance in predicting life expectancy. While SVM demonstrates good performance with proper scaling, Linear Regression performs reasonably well. However, both PCA-based models (Linear Regression and SVM) show lower accuracy compared to the original DataFrame. KNN models provide moderate performance but are sensitive to the choice of the number of

neighbors (k). In summary, SVM and Linear Regression models can provide useful insights, but **Random Forest and Decision Tree** models offer superior accuracy for life expectancy prediction.

In conclusion, both the Random Forest and Decision Tree models applied to the original DataFrame show promise in predicting life expectancy. The Random Forest stands out as the top performer, providing superior accuracy and robustness to complex data relationships. On the other hand, the Decision Tree offers simplicity and interpretability, making it suitable for scenarios where model transparency and explainability are essential. The choice between these models would depend on the specific requirements of the project and the trade-offs between accuracy and interpretability.

Correlation analysis

Birth Rate (-0.88 correlation):The strong negative correlation between life expectancy and birth rate indicates that countries with higher birth rates tend to have lower life expectancies. High birth rates can put a strain on healthcare systems and resources, impacting overall public health and access to medical care.

Fertility Rate (-0.85 correlation):The negative correlation between life expectancy and fertility rate suggests that countries with higher fertility rates also tend to have lower life expectancies. High fertility rates can lead to challenges in providing adequate healthcare and social services, which can impact population health and life expectancy.

Infant Mortality (-0.93 correlation):The significant negative correlation between life expectancy and infant mortality underscores the critical link between early-life health and overall life expectancy. Lower infant mortality rates indicate better access to healthcare and improved maternal and child health.

Physicians per Thousand (0.72 correlation):The positive correlation between life expectancy and the number of physicians per thousand reflects the importance of healthcare access and medical resources in improving life expectancy. Countries with more physicians per thousand are better equipped to provide medical care and preventive services, positively impacting life expectancy.

Salman Almaskati - 500922635