DS203 Project Notebook

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
from scipy.stats import entropy
from math import log, e
import statsmodels.api as sm
import pylab as py
import scipy as sp
from scipy import stats
from scipy.stats import lognorm, kstest
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, make_scorer, hinge_loss, r2_score, mean_absolute_error, mean_squared_log_error
from sklearn.model_selection import GridSearchCV
pd.options.mode.chained_assignment = None
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('/content/Data.csv')
df = df.drop(columns = ['Unnamed: 16',  'Unnamed: 17',  'Unnamed: 18'])
df['Immunization'] = df[['Immunization(DPT %)','Immunization(Measles %)']].mean(axis = 1)
```

```
countries = list(df['Country'].unique())
countries.remove('High')
countries.remove('Low')
countries.remove('Middle')

bar_params = list(df.columns)
bar_params.remove('Immunization')
bar_params.remove('Year')
bar_params.remove('Country')
print(bar_params)
```

```
print(bar_params)
```

```
['Life expectancy at birth', 'Access to Electricity(%)', 'Electric power consumption(kWh/Capita)', 'Rural Population(%)', 'Food Productio
```

Analysis of Variables

```
df.dtypes
```

```
Country                                              object
Year                                                  int64
Life expectancy at birth                            float64
Access to Electricity(%)                            float64
Electric power consumption(kWh/Capita)              float64
Rural Population(%)                                 float64
Food Production Index                               float64
GDP/Capita($)                                       float64
Immunization(DPT %)                                 float64
Immunization(Measles %)                             float64
GDP Deflator                                        float64
Road Accident Mortality(per 100,000 people)         float64
Population Density                                  float64
Total greenhouse gas emissions (kt of CO2 equivalent)   int64
Unemployed Labour(%)                                float64
Literacy Rate(%)                                    float64
Immunization                                        float64
dtype: object
```

```
print(df.nunique())
```

```
Country                                          15
Year                                             15
Life expectancy at birth                        223
Access to Electricity(%)                        150
Electric power consumption(kWh/Capita)          225
Rural Population(%)                             225
Food Production Index                           221
GDP/Capita($)                                   225
Immunization(DPT %)                              88
Immunization(Measles %)                          88
GDP Deflator                                    225
Road Accident Mortality(per 100,000 people)     186
```

```
Population Density                                    224
Total greenhouse gas emissions (kt of CO2 equivalent)    225
Unemployed Labour(%)                                  200
Literacy Rate(%)                                      151
Immunization                                          105
dtype: int64
```

```
print('Means of variables:')
for i,col in enumerate(bar_params):
  print(col,':',df[col].mean())
```

```
Means of variables:
Life expectancy at birth : 70.86710761919637
Access to Electricity(%) : 87.71144870709955
Electric power consumption(kWh/Capita) : 4344.867199263596
Rural Population(%) : 34.145733820639876
Food Production Index : 87.56255555555556
GDP/Capita($) : 16438.749858911626
Immunization(DPT %) : 86.92773003679766
Immunization(Measles %) : 86.77356210792121
GDP Deflator : 7.040877659284802
Road Accident Mortality(per 100,000 people) : 18.291796113656524
Population Density : 105.6592155922241
Total greenhouse gas emissions (kt of CO2 equivalent) : 3602396.0888888887
Unemployed Labour(%) : 6.574817024684446
Literacy Rate(%) : 87.88336783406272
```

```
print('Variance of variables:')
for i,col in enumerate(bar_params):
  print(col,':',df[col].var())
```

```
Variance of variables:
Life expectancy at birth : 106.17413646460045
Access to Electricity(%) : 417.188530530704
Electric power consumption(kWh/Capita) : 16829479.20995864
Rural Population(%) : 401.4427336954383
Food Production Index : 122.34026046006943
GDP/Capita($) : 334370791.6233327
Immunization(DPT %) : 258.7687853756203
Immunization(Measles %) : 237.84703483719062
GDP Deflator : 82.28192857652256
Road Accident Mortality(per 100,000 people) : 80.02035690258971
Population Density : 15307.164613727697
```

```
Total greenhouse gas emissions (kt of CO2 equivalent) : 26893195004059.633
Unemployed Labour(%) : 7.269864881059531
Literacy Rate(%) : 189.4014231429748
```

```
print('Skew of variables:')
for i,col in enumerate(bar_params):
  print(col,':',df[col].skew())
```

```
Skew of variables:
Life expectancy at birth : -1.2434840996695395
Access to Electricity(%) : -1.5069400016542698
Electric power consumption(kWh/Capita) : 0.8188565199336154
Rural Population(%) : 0.7478297494473113
Food Production Index : -0.15834547229094498
GDP/Capita($) : 0.9364539476108534
Immunization(DPT %) : -2.0004215060376533
Immunization(Measles %) : -1.9088142898923441
GDP Deflator : 4.928687342880331
Road Accident Mortality(per 100,000 people) : 0.4798408810601251
Population Density : 1.3842907840166243
Total greenhouse gas emissions (kt of CO2 equivalent) : 1.7242562923707925
Unemployed Labour(%) : 1.6378272308737918
Literacy Rate(%) : -1.1628444411385248
```

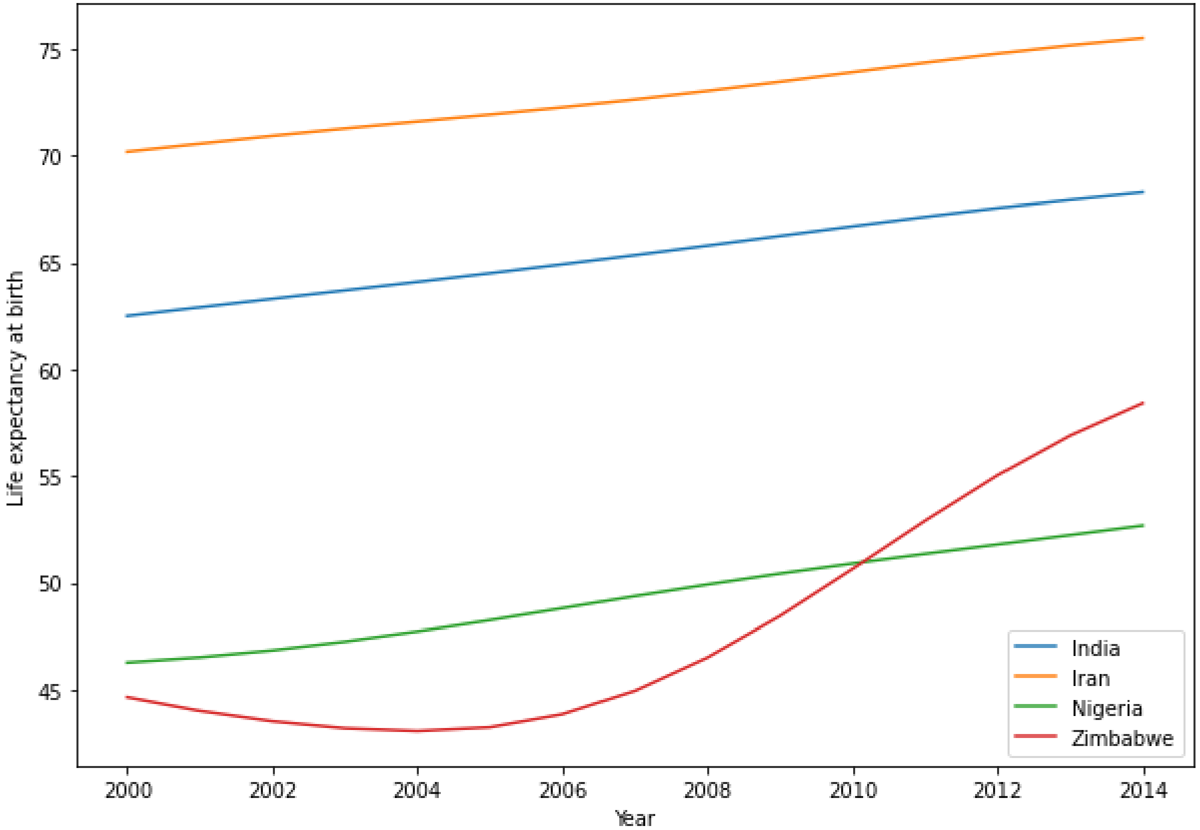
```
print('Min & Max of variables:')
for i,col in enumerate(bar_params):
  print(col,':',df[col].min(),',',df[col].max())
```

```
Min & Max of variables:
Life expectancy at birth : 43.065 , 83.5878
Access to Electricity(%) : 32.3 , 100.0
Electric power consumption(kWh/Capita) : 74.49061999999999 , 13704.58
Rural Population(%) : 8.696 , 72.333
Food Production Index : 56.62 , 121.32
GDP/Capita($) : 356.6932 , 68150.11
Immunization(DPT %) : 25.0 , 99.0
Immunization(Measles %) : 30.0 , 99.0
GDP Deflator : -2.017678703 , 95.40865975
Road Accident Mortality(per 100,000 people) : 2.9 , 41.0
Population Density : 2.493134 , 435.7612
Total greenhouse gas emissions (kt of CO2 equivalent) : 24150 , 19964160
Unemployed Labour(%) : 2.63 , 20.52
Literacy Rate(%) : 51.07765961 , 99.704997355469
```

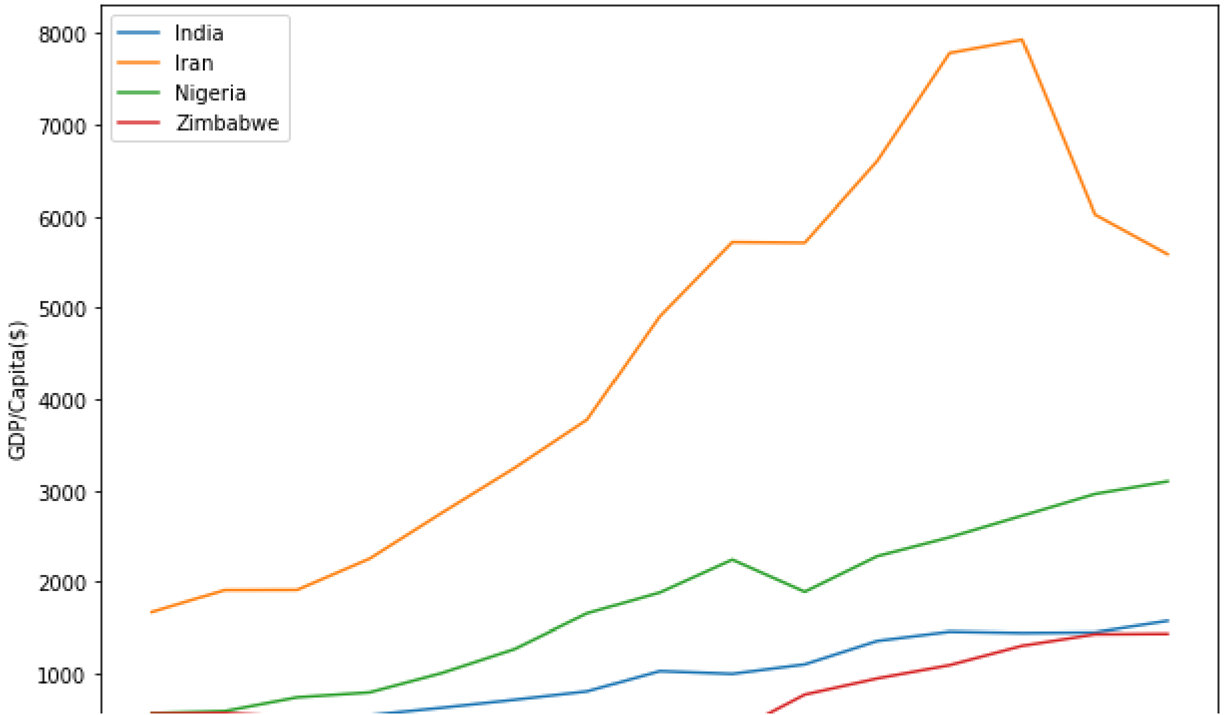
Low Income Countries

```
low_income_countries = ['India', 'Iran', 'Nigeria', 'Zimbabwe']
high_income_countries = ['UK', 'USA', 'Australia', 'Japan']
middle_income_countries = ['Russia', 'Colombia', 'Brazil', 'Mexico']
line_params          = ['Life expectancy at birth','GDP/Capita($)', 'GDP Deflator', 'Unemployed Labour(%)', 'Population Density', 'Immunizatior
years                = df['Year'].unique()
```

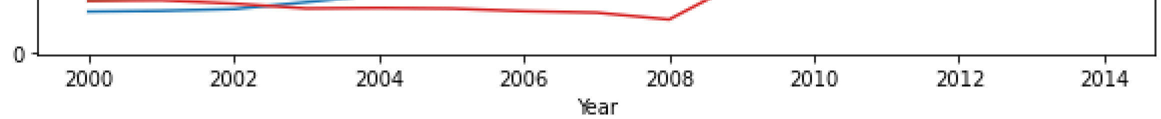
```
for param in line_params:
  plt.figure(figsize=(10,7))
  for country in low_income_countries:
    df_i = df[df['Country']==country]
    plt.plot(years,df_i[param], label = country)
    plt.xlabel('Year')
    plt.ylabel(param)
    plt.legend()
  plt.show()
from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 20000})'''))
```

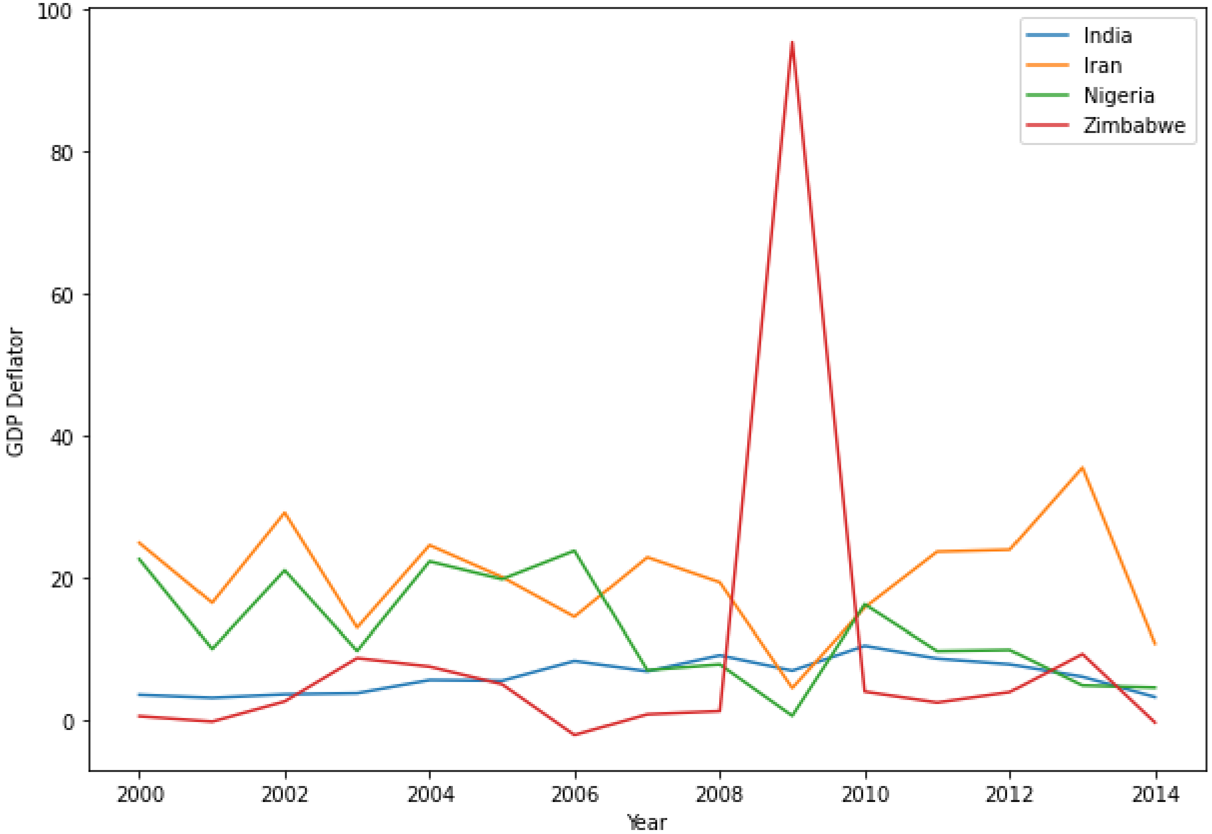


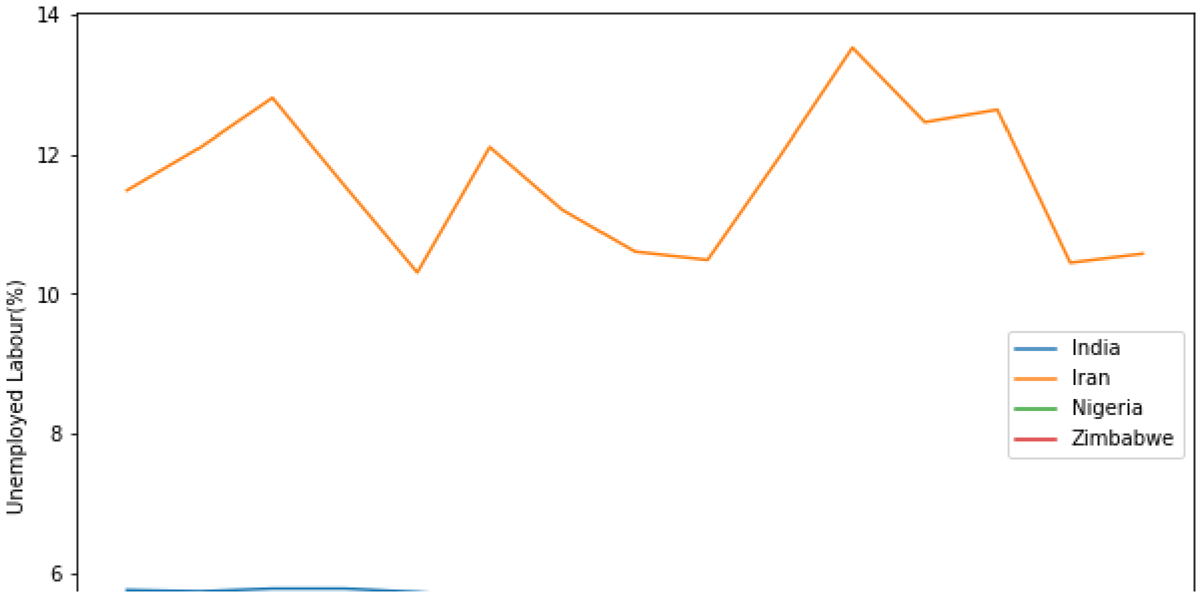
Life expectancy at birth
Year
India
Iran
Nigeria
Zimbabwe

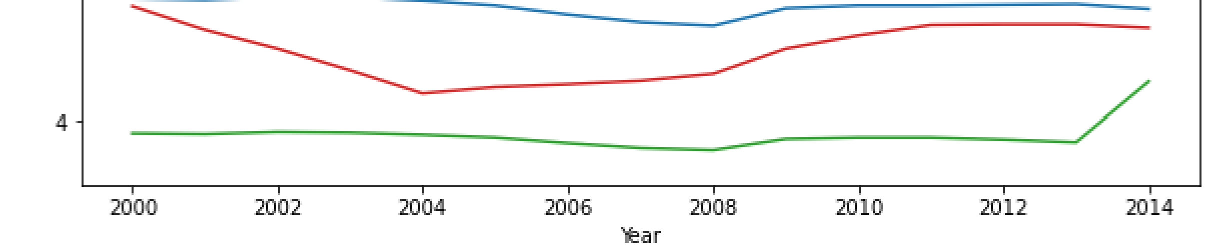


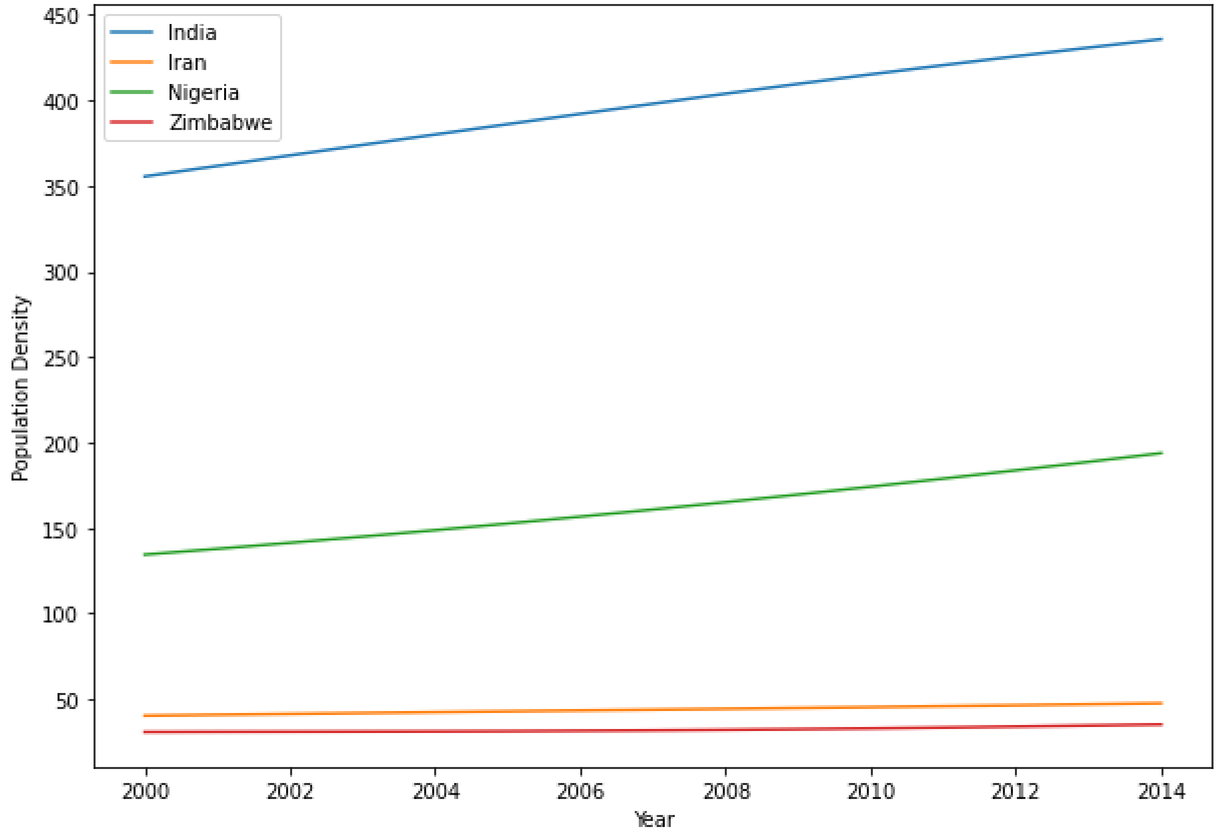
GDP/Capita($)
India
Iran
Nigeria
Zimbabwe

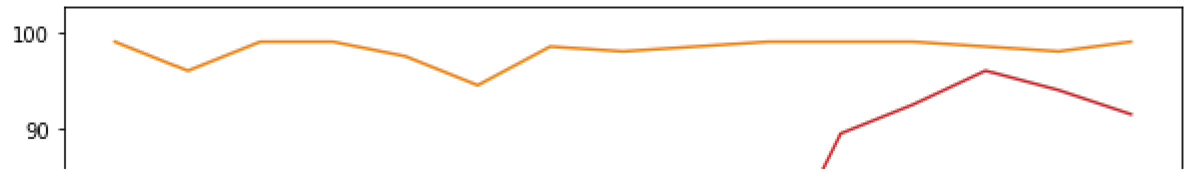

0
2000
2002
2004
2006
2008
2010
2012
2014
Year


GDP Deflator
0
20
40
60
80
100
2000
2002
2004
2006
2008
2010
2012
2014
Year
India
Iran
Nigeria
Zimbabwe


Unemployed Labour(%)
6
8
10
12
14
India
Iran
Nigeria
Zimbabwe



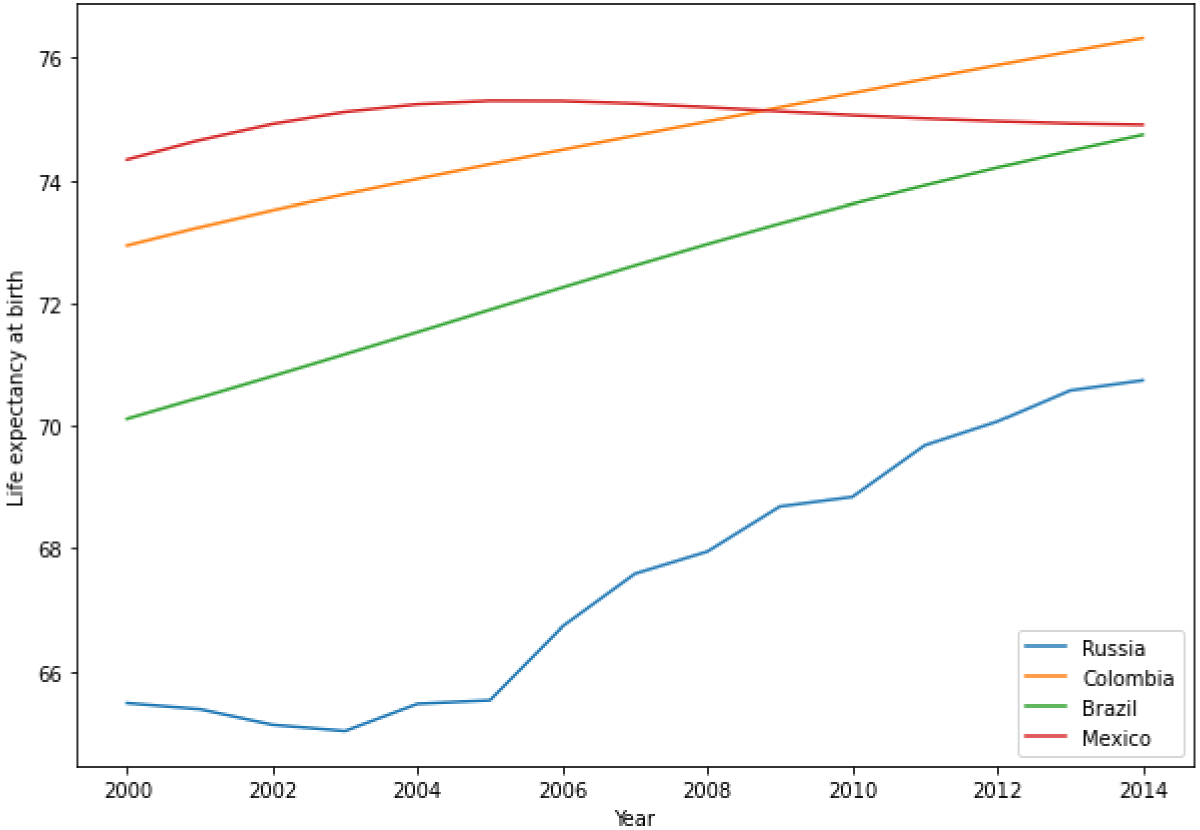




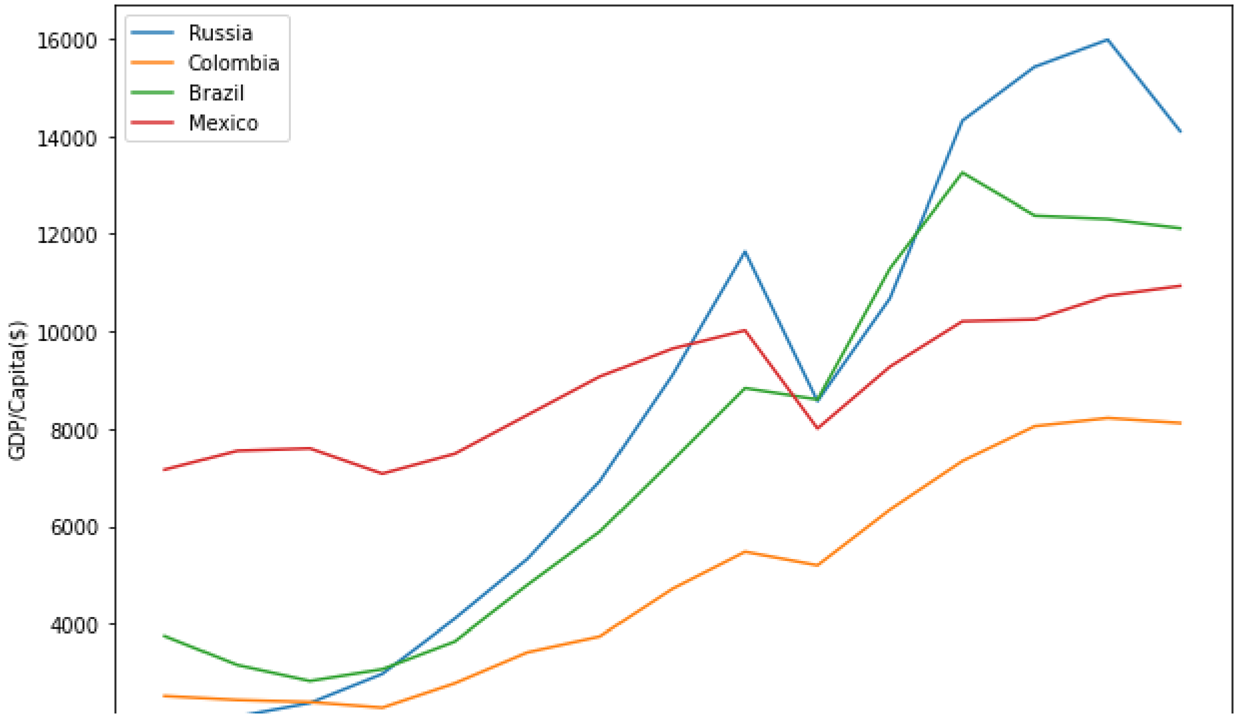
Middle Income Countries

```
for param in line_params:
  plt.figure(figsize=(10,7))
  for country in middle_income_countries:
```

```
    df_i = df[df['Country']==country]
    plt.plot(years,df_i[param], label = country)
    plt.xlabel('Year')
    plt.ylabel(param)
    plt.legend()
  plt.show()
from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 20000})'''))
```

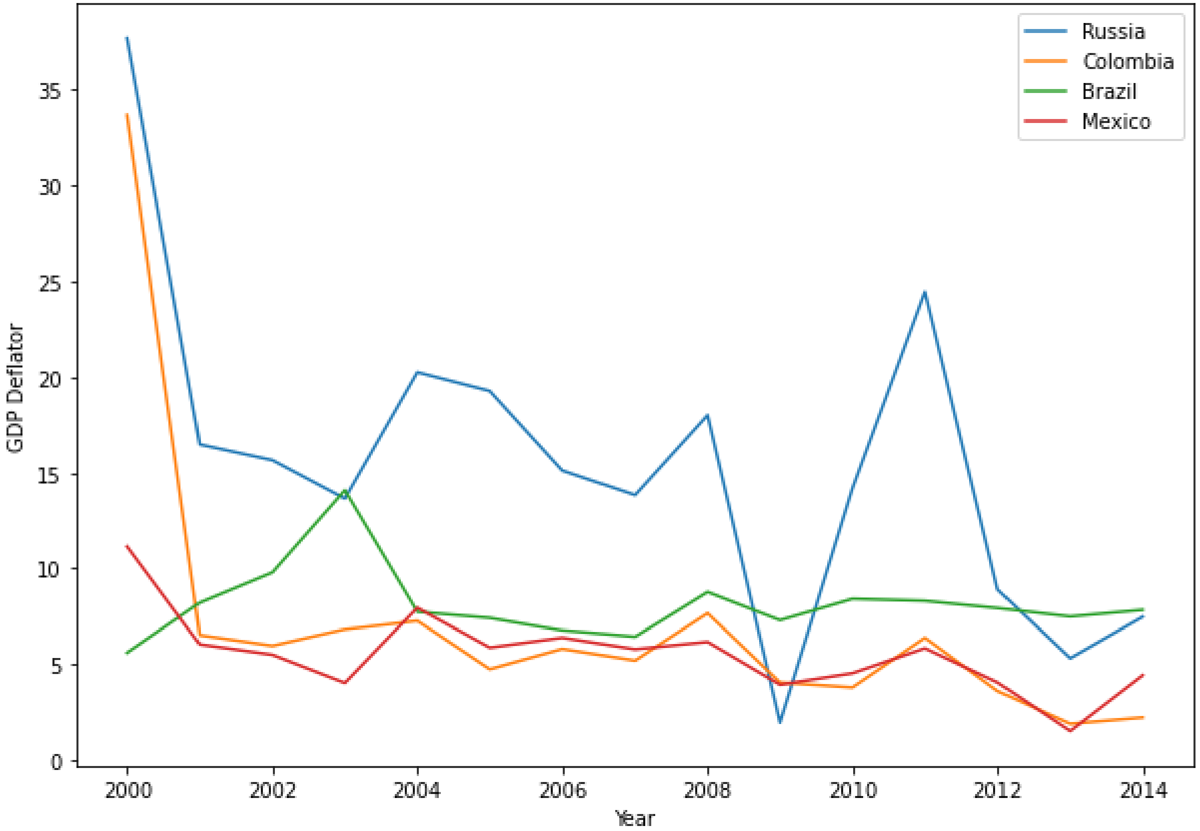


Life expectancy at birth
Year
Russia
Colombia
Brazil
Mexico

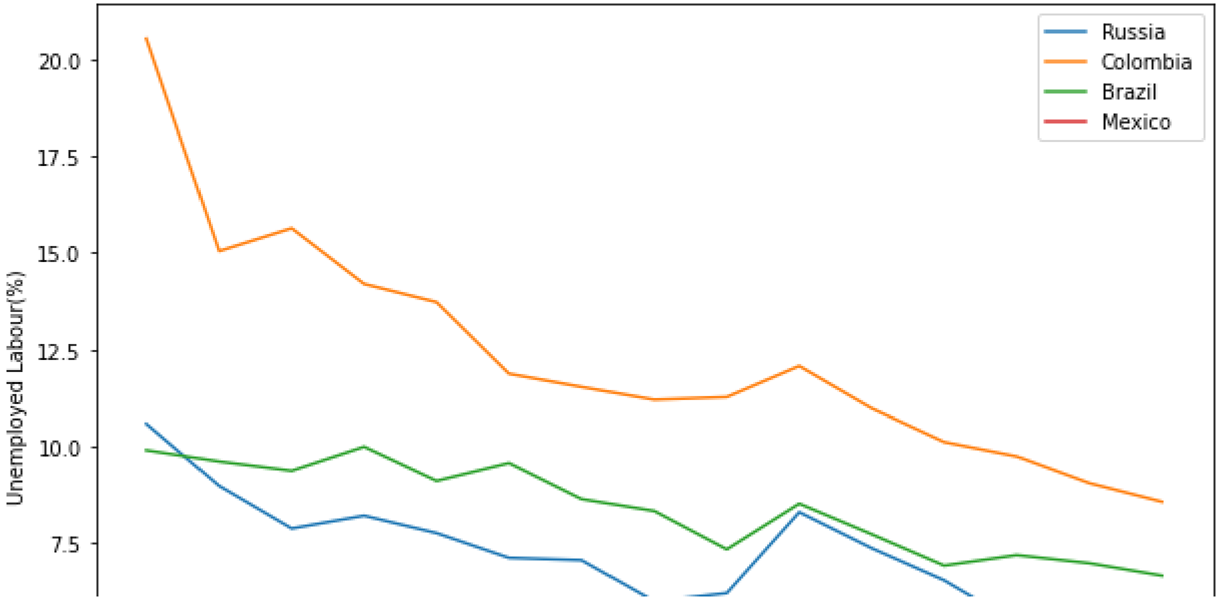


GDP/Capita($)
Russia
Colombia
Brazil
Mexico

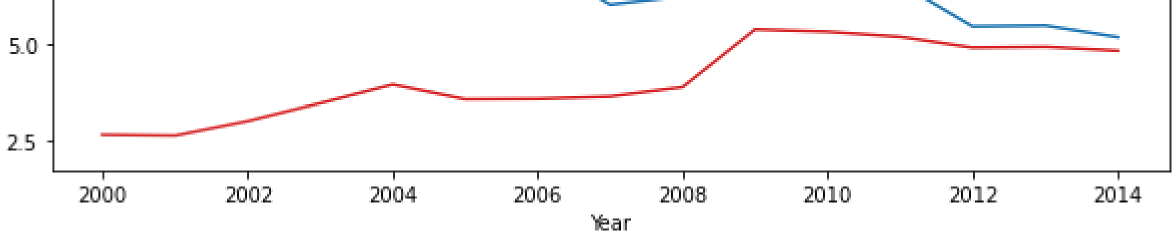
2000
2000
2002
2004
2006
2008
2010
2012
2014
Year

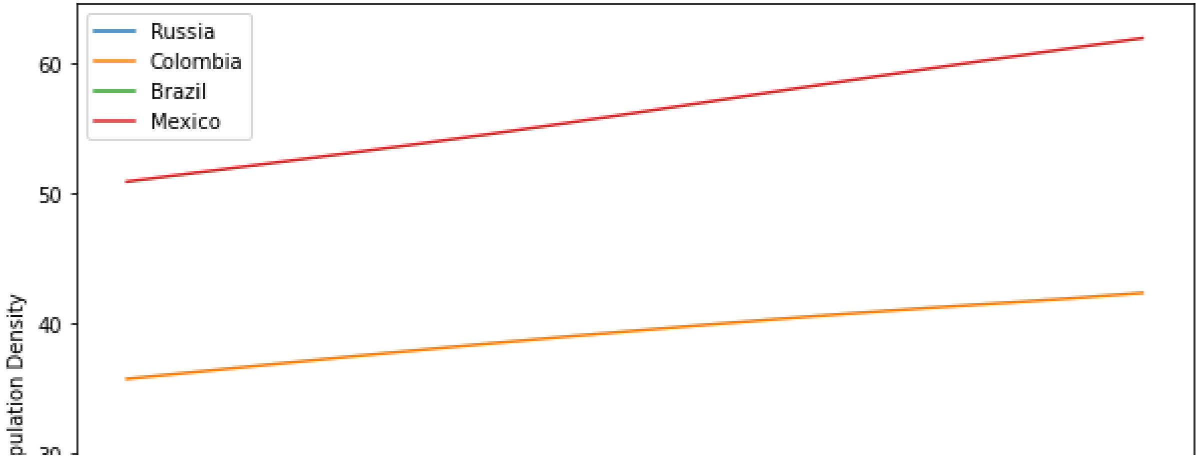


Russia
Colombia
Brazil
Mexico
GDP Deflator
0
5
10
15
20
25
30
35
2000
2002
2004
2006
2008
2010
2012
2014
Year



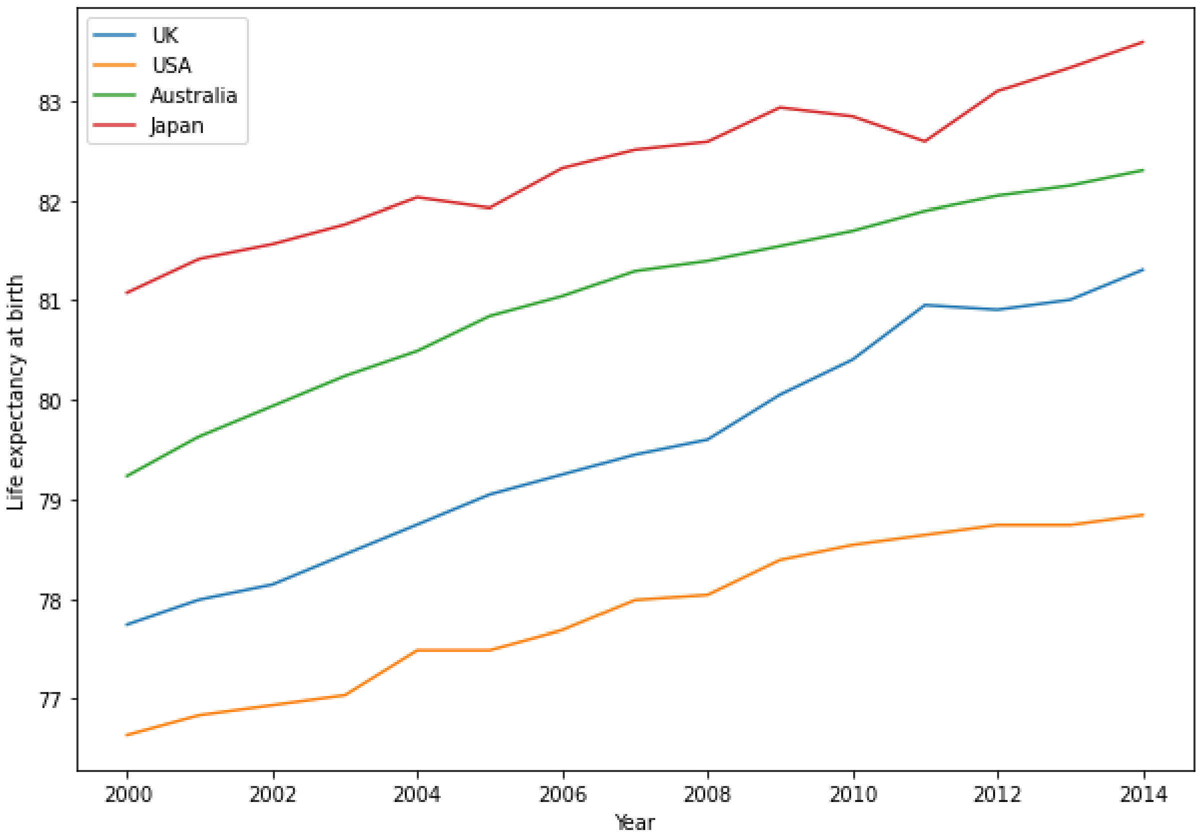
Russia
Colombia
Brazil
Mexico
Unemployed Labour(%)
7.5
10.0
12.5
15.0
17.5
20.0

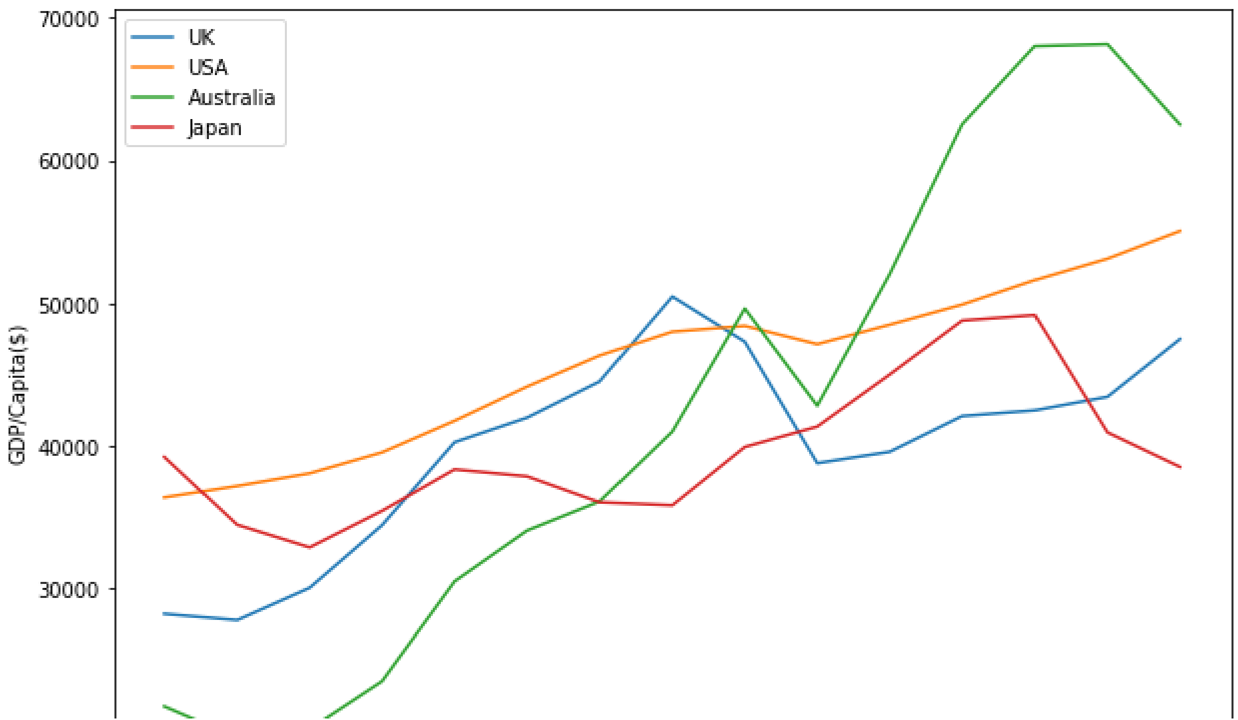




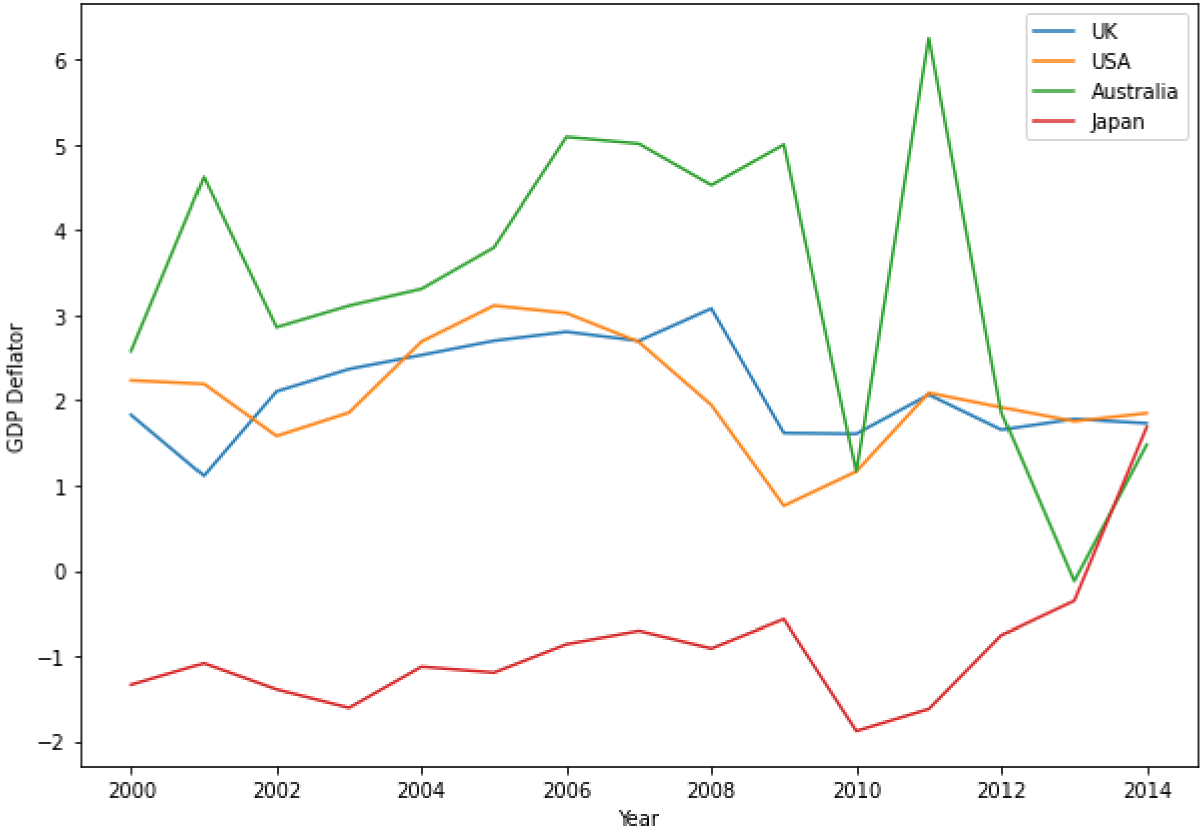
High Income Countries

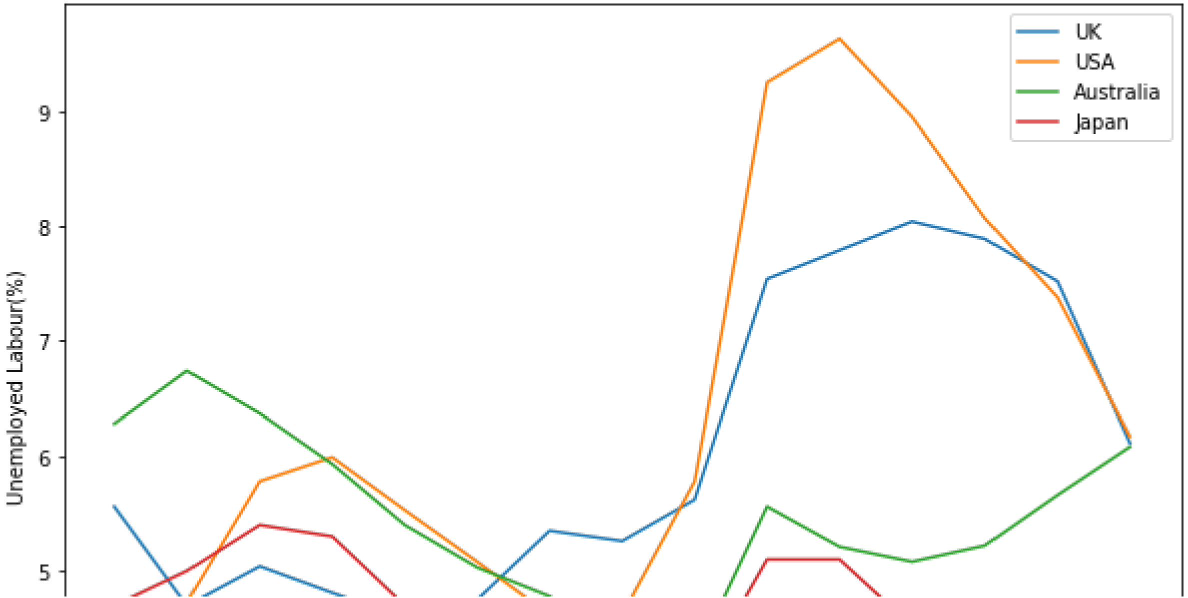
```
for param in line_params:
  plt.figure(figsize=(10,7))
  for country in high_income_countries:
    df_i = df[df['Country']==country]
    plt.plot(years,df_i[param], label = country)
    plt.xlabel('Year')
    plt.ylabel(param)
    plt.legend()
  plt.show()

from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 20000})'''))
```


UK
USA
Australia
Japan
Life expectancy at birth
Year
77
78
79
80
81
82
83
2000
2002
2004
2006
2008
2010
2012
2014

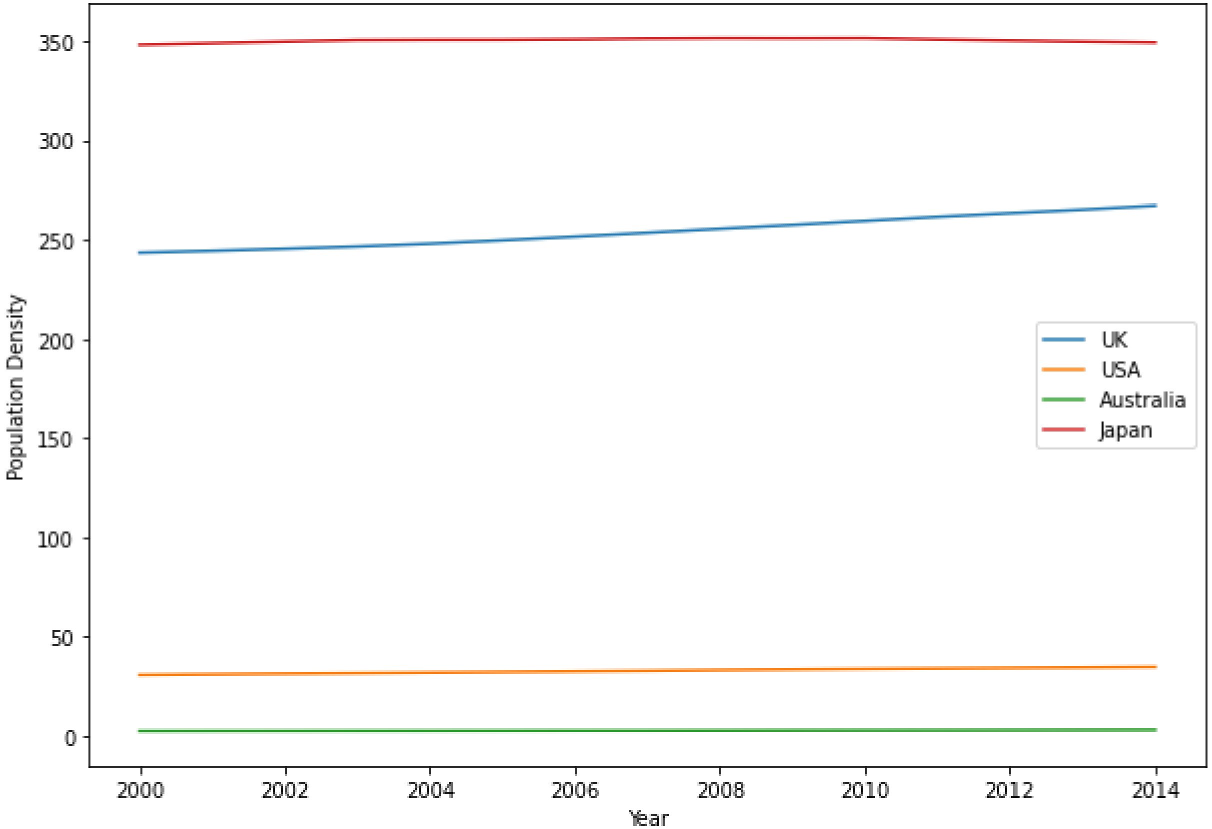

UK
USA
Australia
Japan
GDP/Capita($)
30000
40000
50000
60000
70000


20000
2000
2002
2004
2006
2008
2010
2012
2014
Year

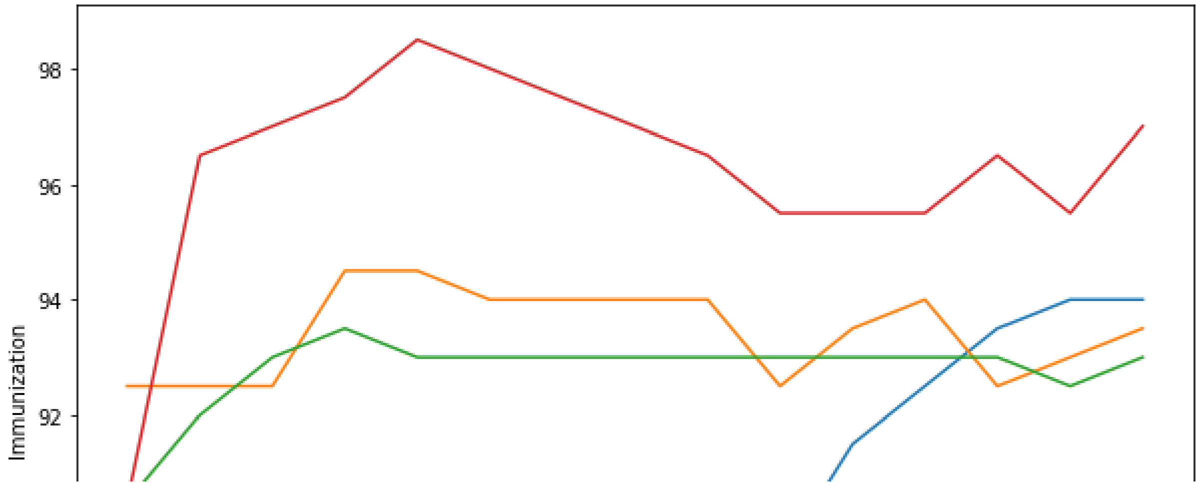

UK
USA
Australia
Japan
GDP Deflator
6
5
4
3
2
1
0
−1
−2
2000
2002
2004
2006
2008
2010
2012
2014
Year


UK
USA
Australia
Japan
Unemployed Labour(%)
9
8
7
6
5

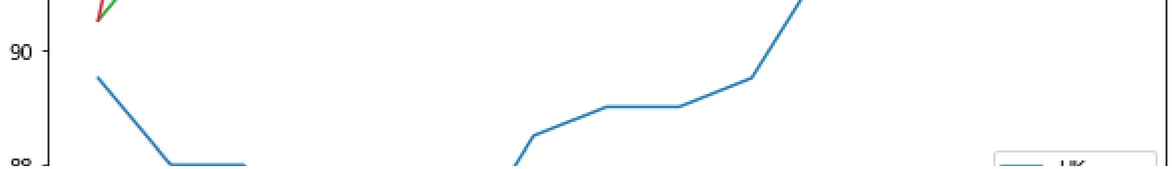
4
2000
2002
2004
2006
2008
2010
2012
2014
Year



Population Density
0
50
100
150
200
250
300
350
2000
2002
2004
2006
2008
2010
2012
2014
Year
UK
USA
Australia
Japan

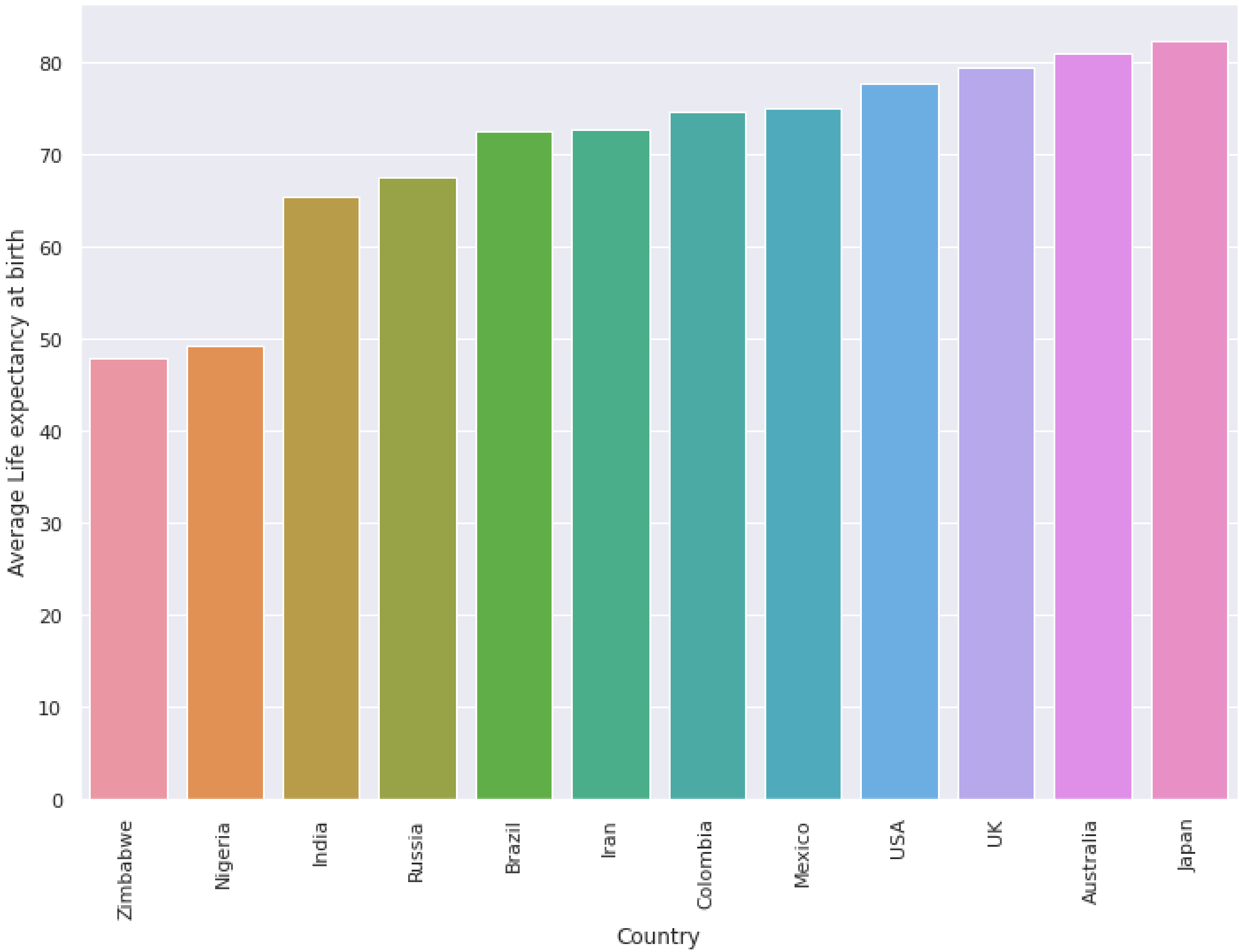


Immunization
92
94
96
98

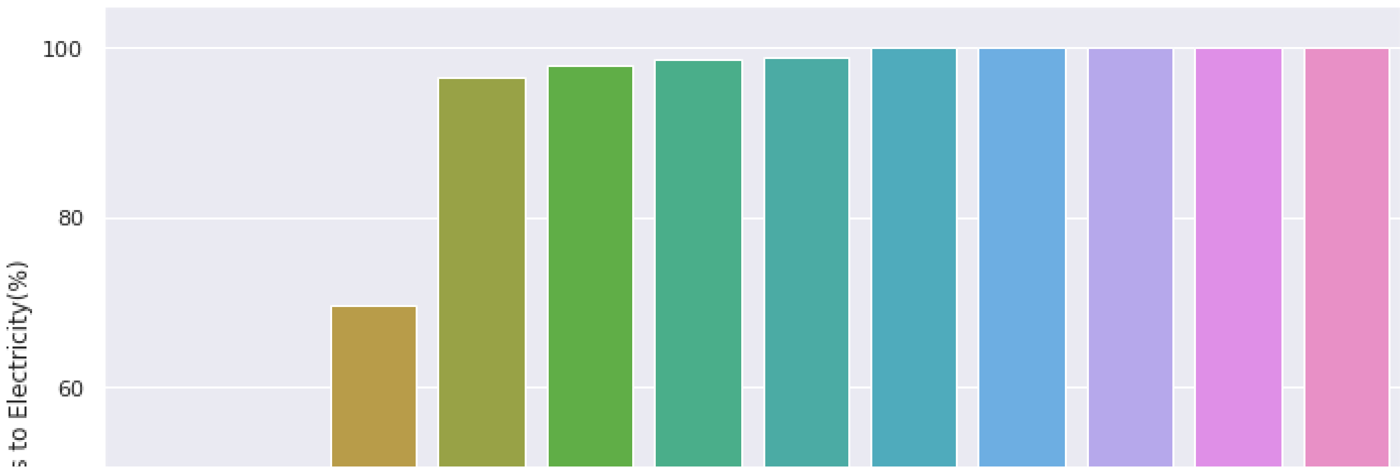


Average Value for parameters for each country from 2000-2014

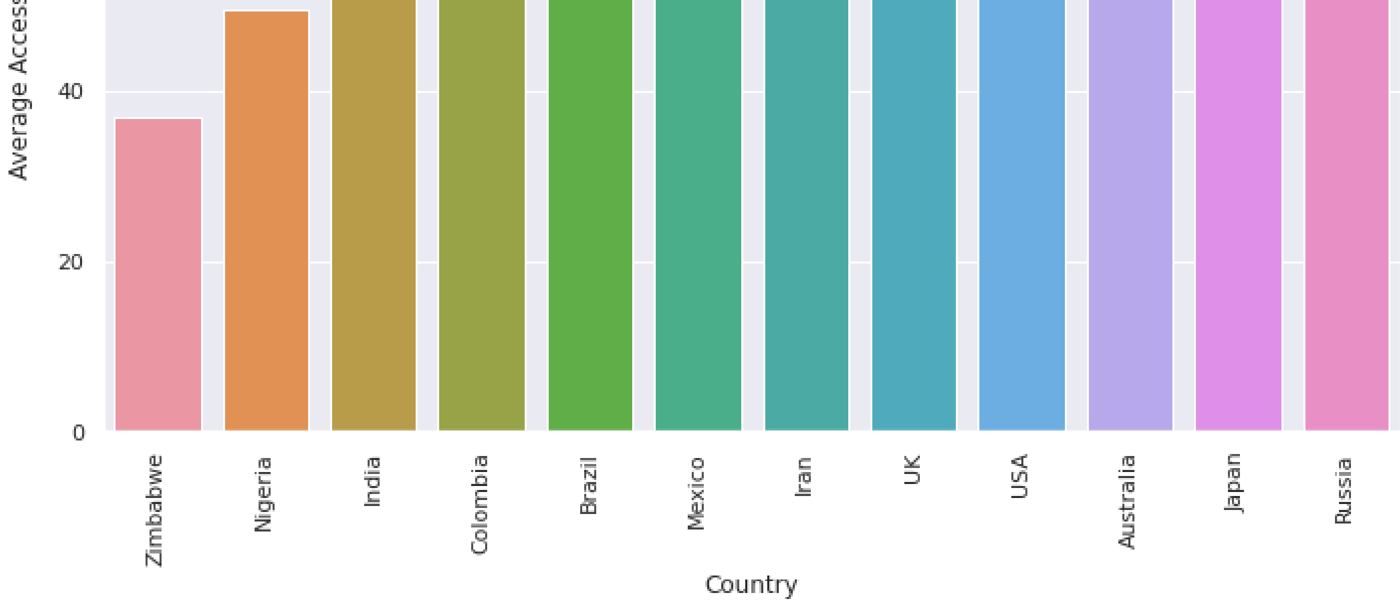
```
from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 20000})'''))

sns.set(rc={'figure.figsize':(11.7,8.27)})

for i in bar_params:
  plt.plot()
  param_value = {}
  for m in countries:
    df_country = df[df['Country']==m]
    param_value[m] = df_country[i].mean()
    param_value = dict(sorted(param_value.items(), key=lambda item: item[1]))
  sns.barplot(x=list(param_value.keys()),y=list(param_value.values()) )
  plt.xlabel('Country')
  plt.xticks(rotation = 90)
  plt.ylabel('Average '+ i)
  plt.show()
```

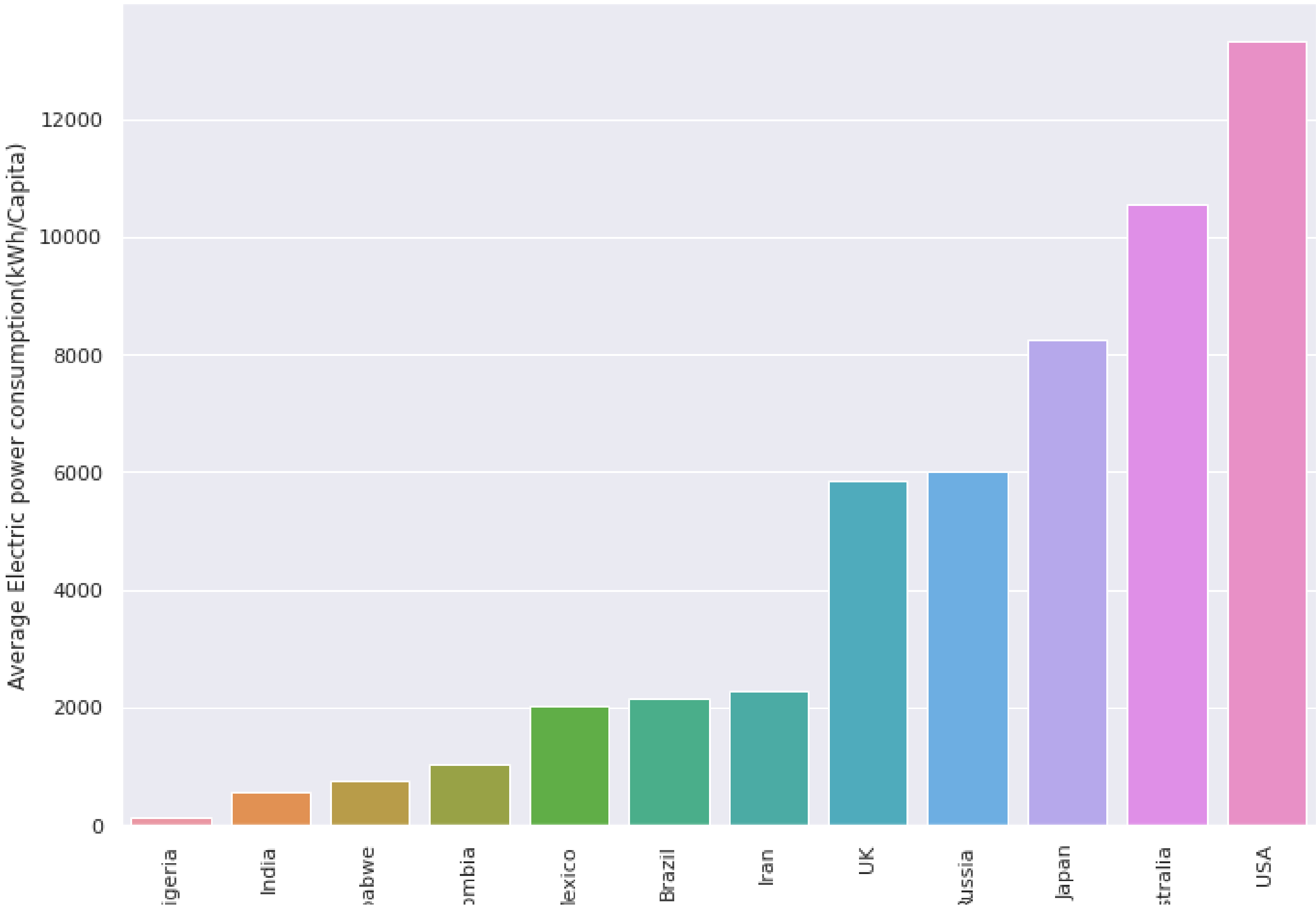


Average Life expectancy at birth
0
10
20
30
40
50
60
70
80
Zimbabwe
Nigeria
India
Russia
Brazil
Iran
Colombia
Mexico
USA
UK
Australia
Japan
Country

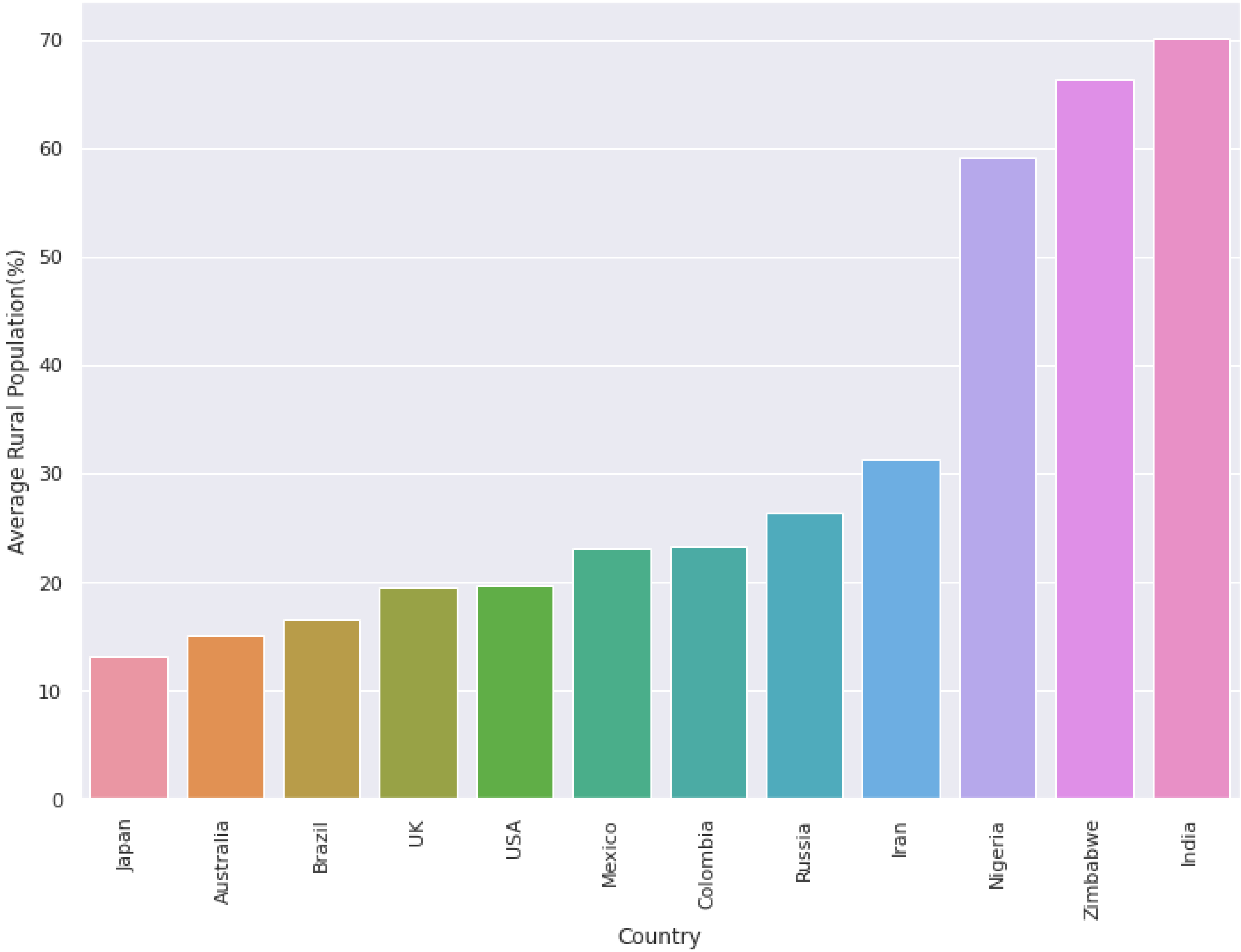


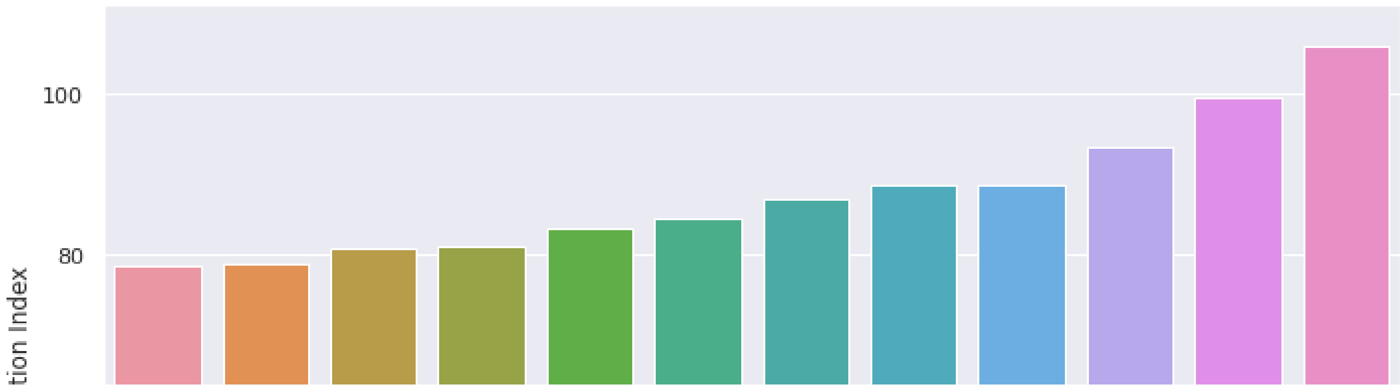
60
80
100

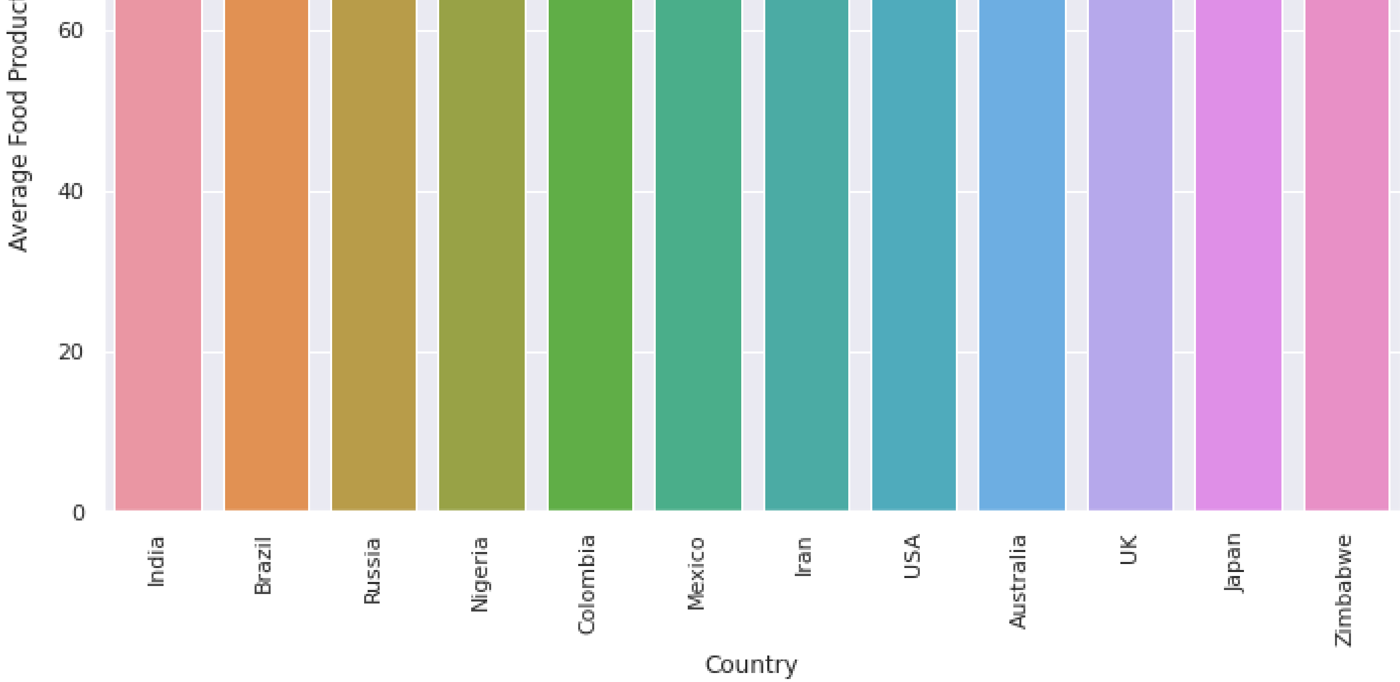

Average Access
40
20
0
Zimbabwe
Nigeria
India
Colombia
Brazil
Mexico
Iran
UK
USA
Australia
Japan
Russia
Country

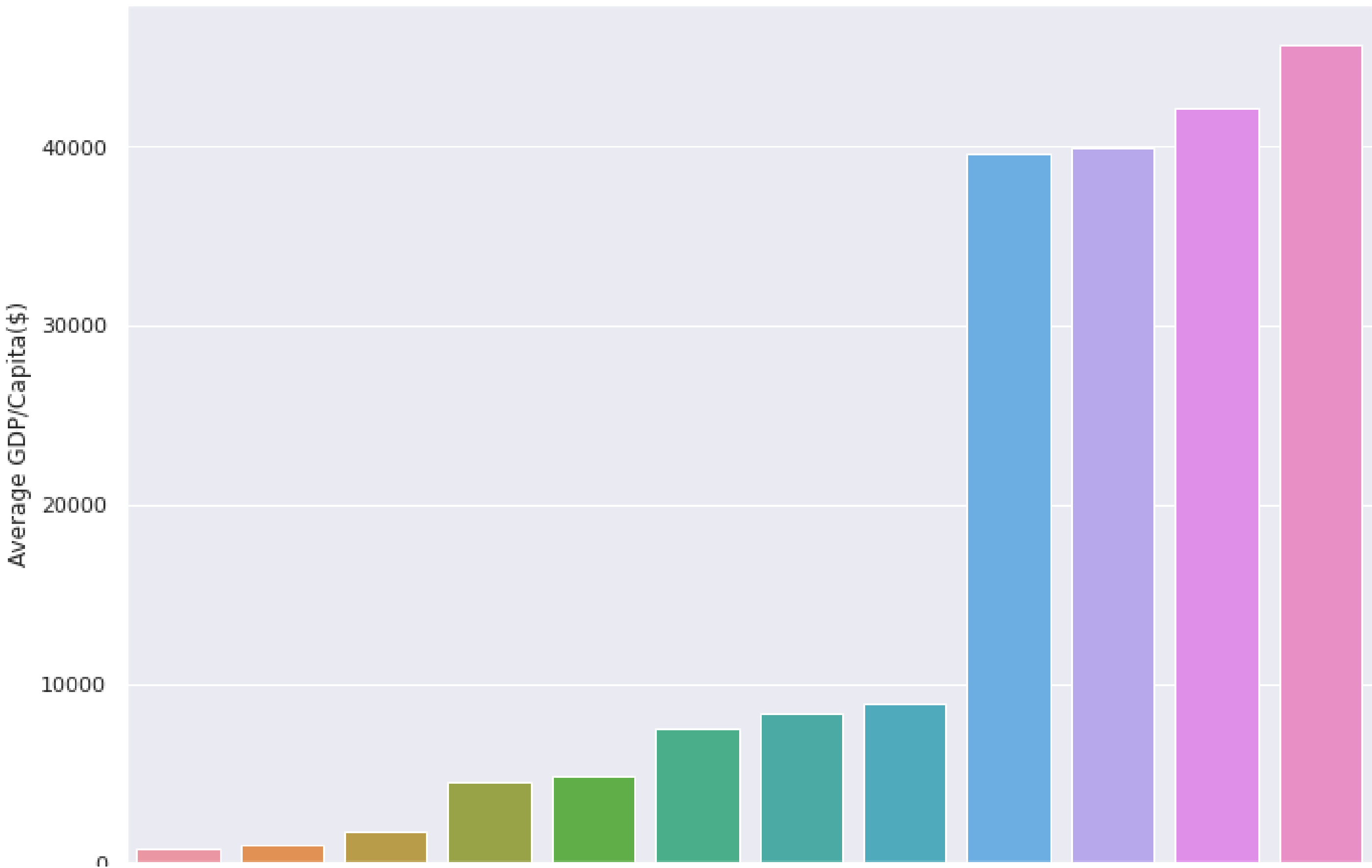

Average Electric power consumption(kWh/Capita)
12000
10000
8000
6000
4000
2000
0
India
Brazil
Iran
UK
Japan
USA


Country

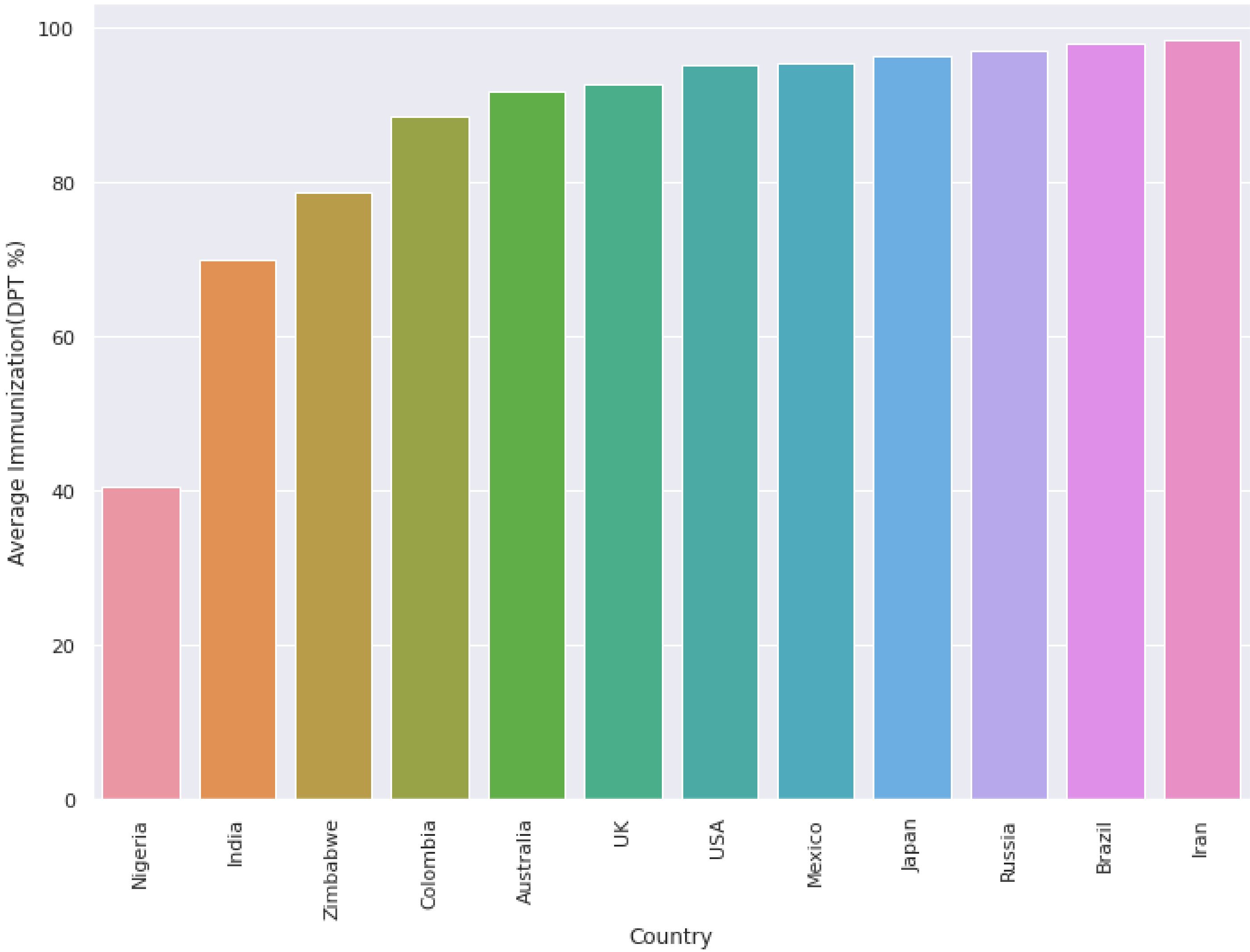

Average Rural Population(%)
70
60
50
40
30
20
10
0
Japan
Australia
Brazil
UK
USA
Mexico
Colombia
Russia
Iran
Nigeria
Zimbabwe
India
Country


100
80

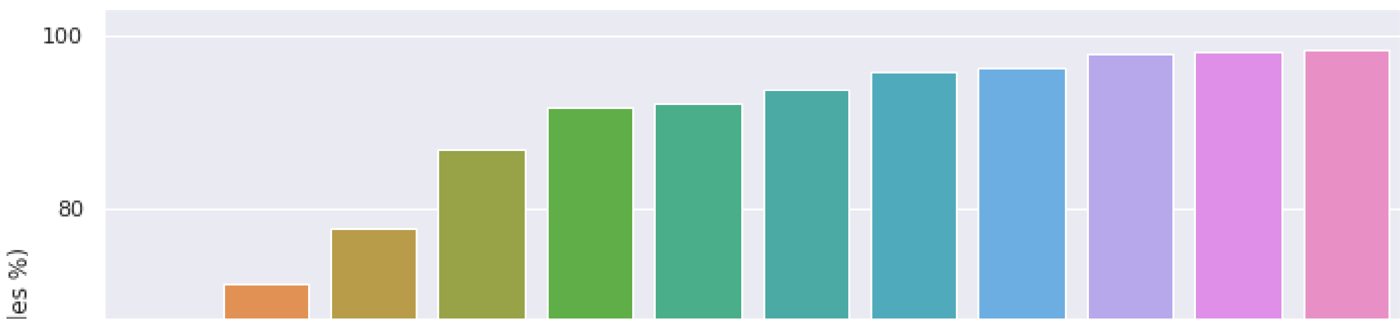

Average Food Produc
60
40
20
0
India
Brazil
Russia
Nigeria
Colombia
Mexico
Iran
USA
Australia
UK
Japan
Zimbabwe
Country


Average GDP/Capita($)
40000
30000
20000
10000
0

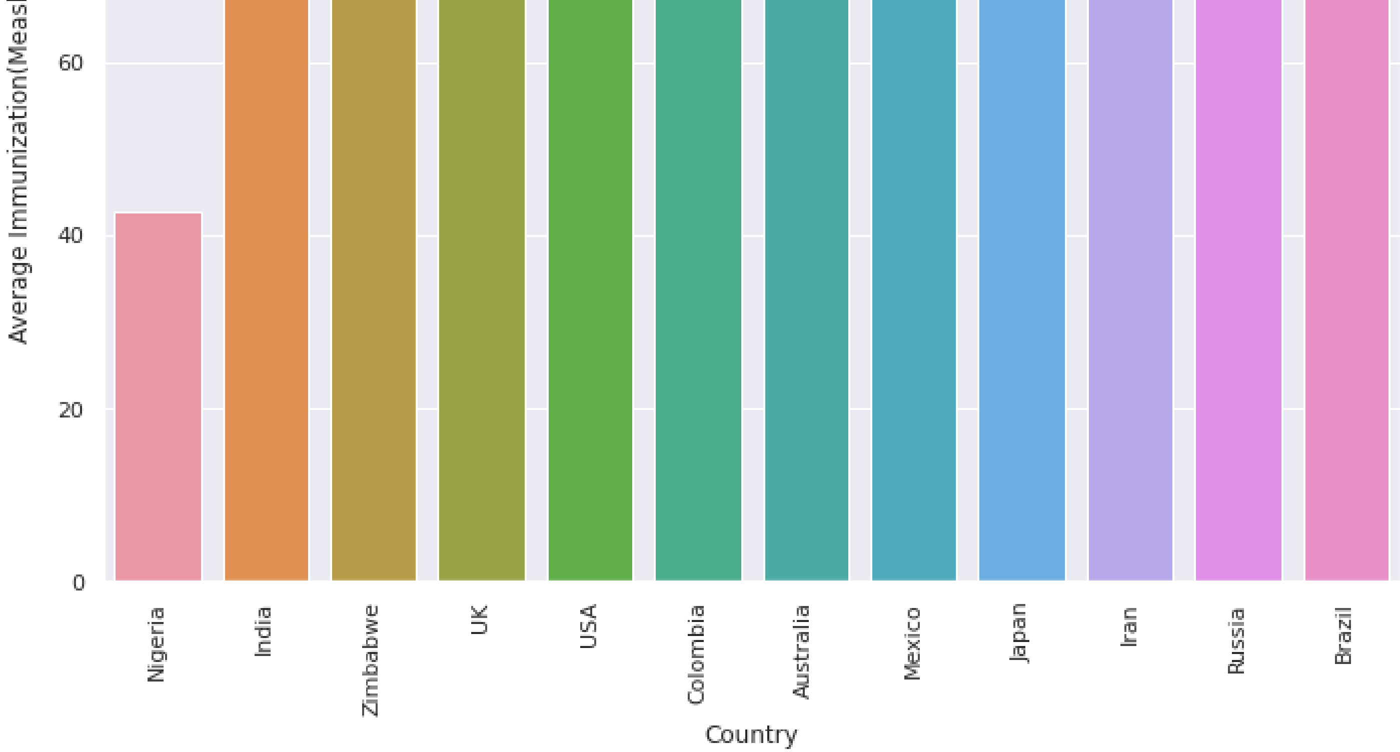
Zimbabwe
India
Nigeria
Iran
Colombia
Brazil
Russia
Mexico
Japan
UK
Australia
USA
Country

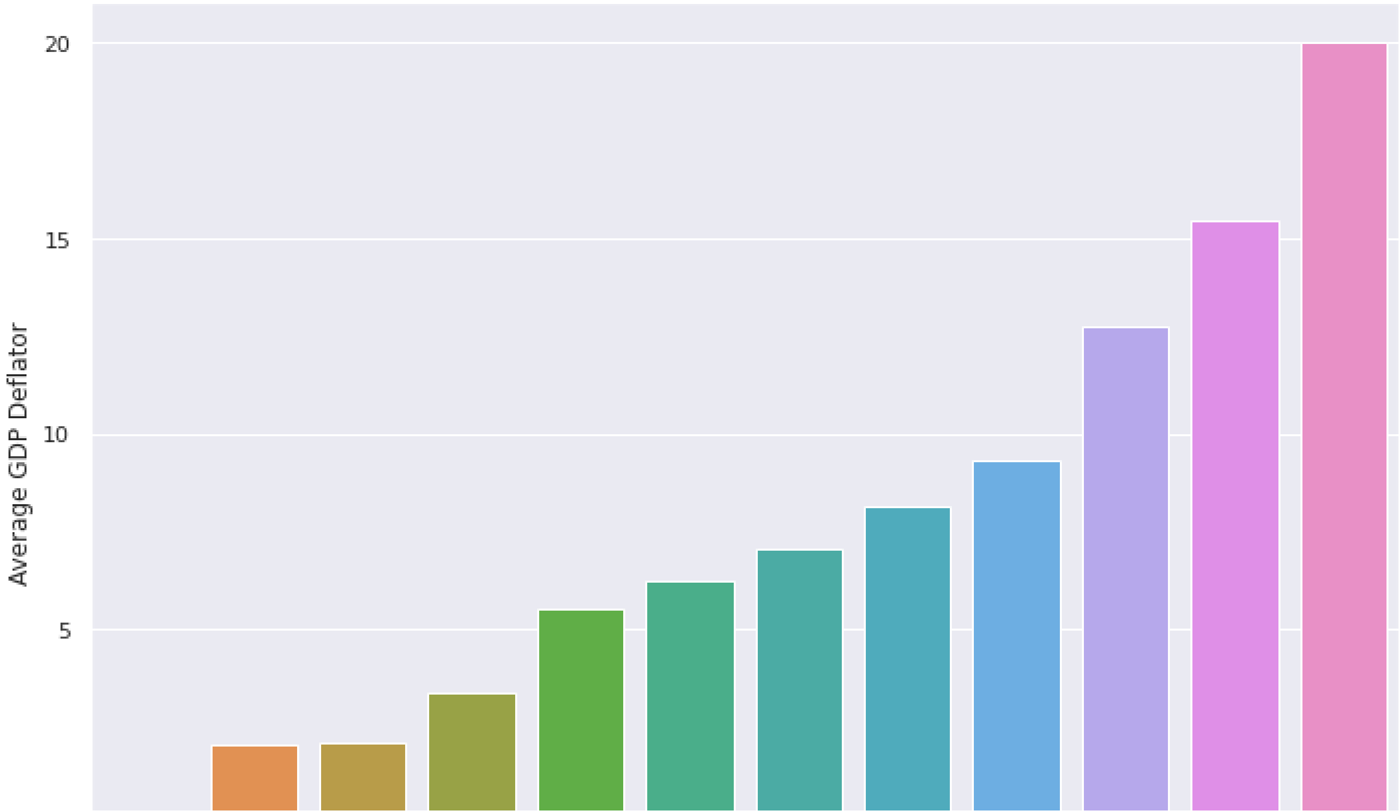


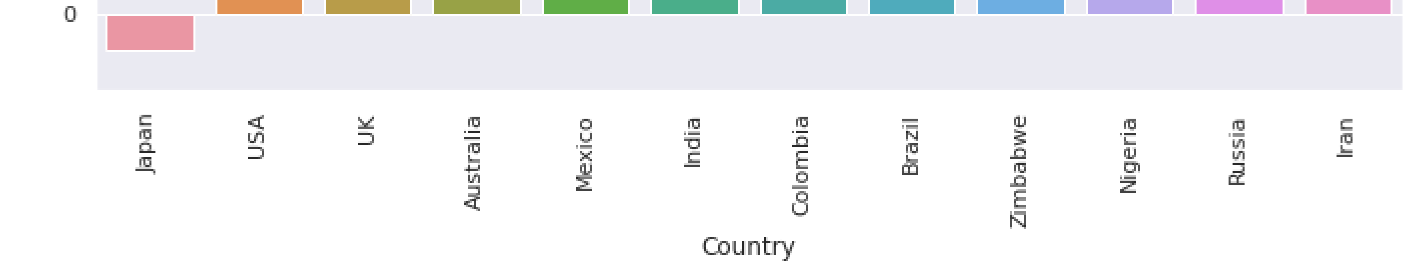
Average Immunization(DPT %)
100
80
60
40
20
0
Nigeria
India
Zimbabwe
Colombia
Australia
UK
USA
Mexico
Japan
Russia
Brazil
Iran
Country

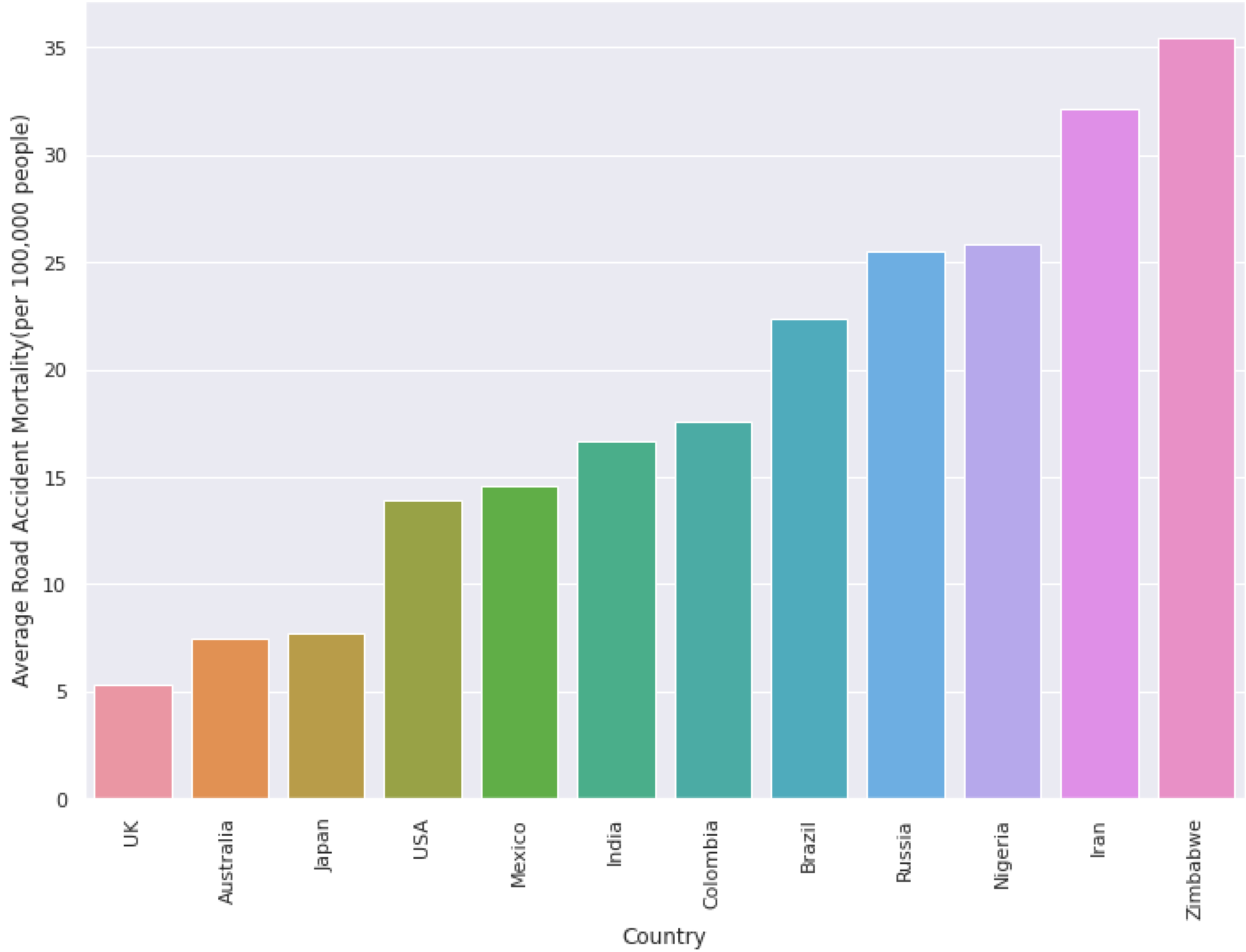


100
80

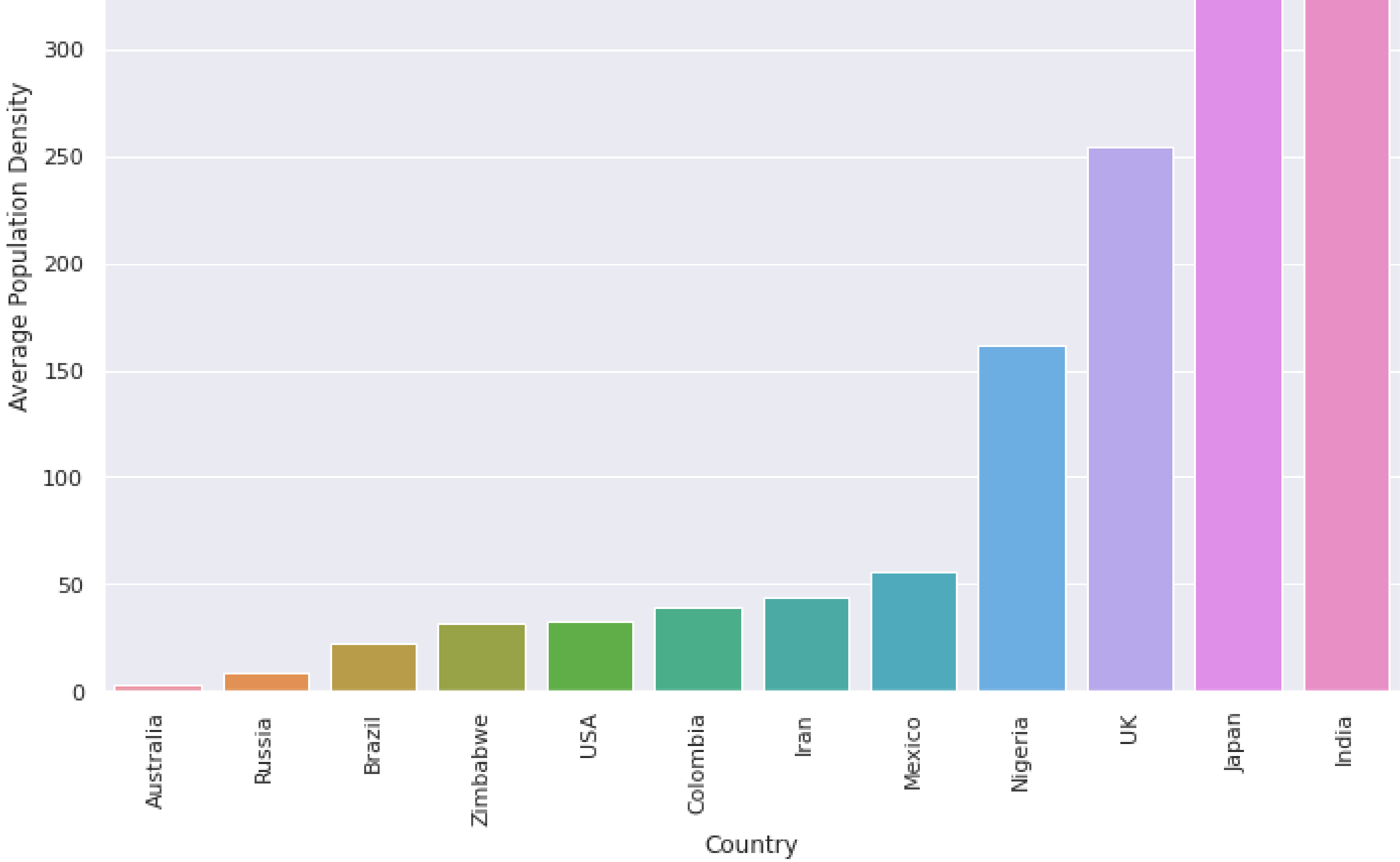

Average Immunization(Measl
0
20
40
60
Nigeria
India
Zimbabwe
UK
USA
Colombia
Australia
Mexico
Japan
Iran
Russia
Brazil
Country


Average GDP Deflator
5
10
15
20

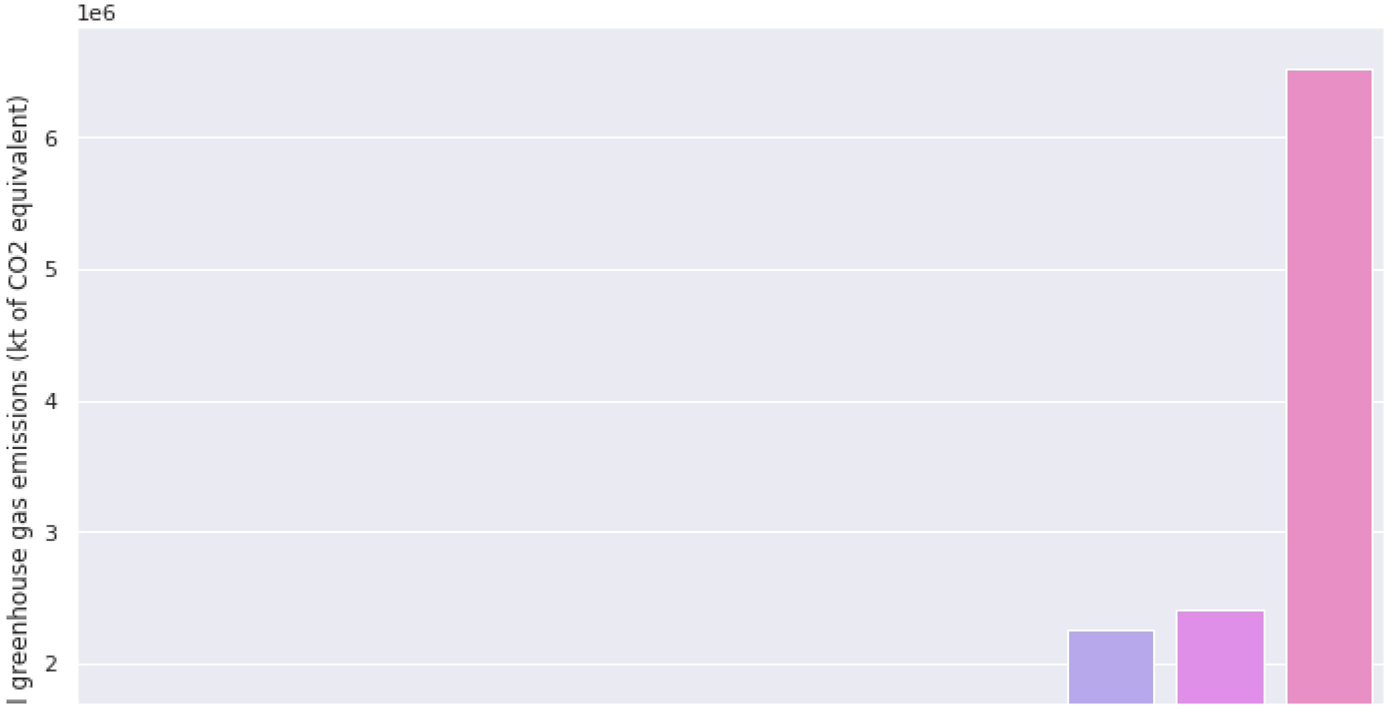

0
Japan
USA
UK
Australia
Mexico
India
Colombia
Brazil
Zimbabwe
Nigeria
Russia
Iran
Country


Average Road Accident Mortality(per 100,000 people)
35
30
25
20
15
10
5
0
UK
Australia
Japan
USA
Mexico
India
Colombia
Brazil
Russia
Nigeria
Iran
Zimbabwe
Country

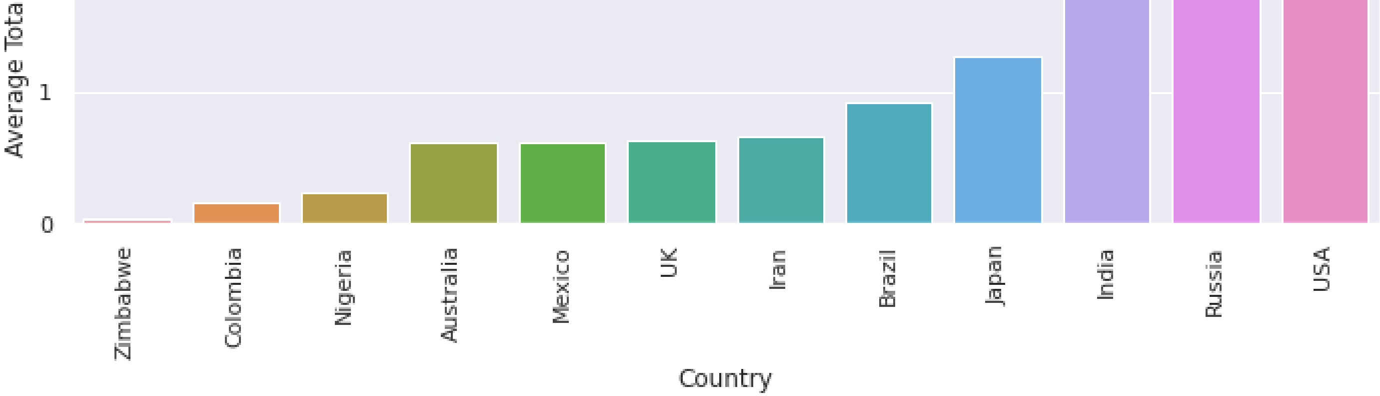

400
350

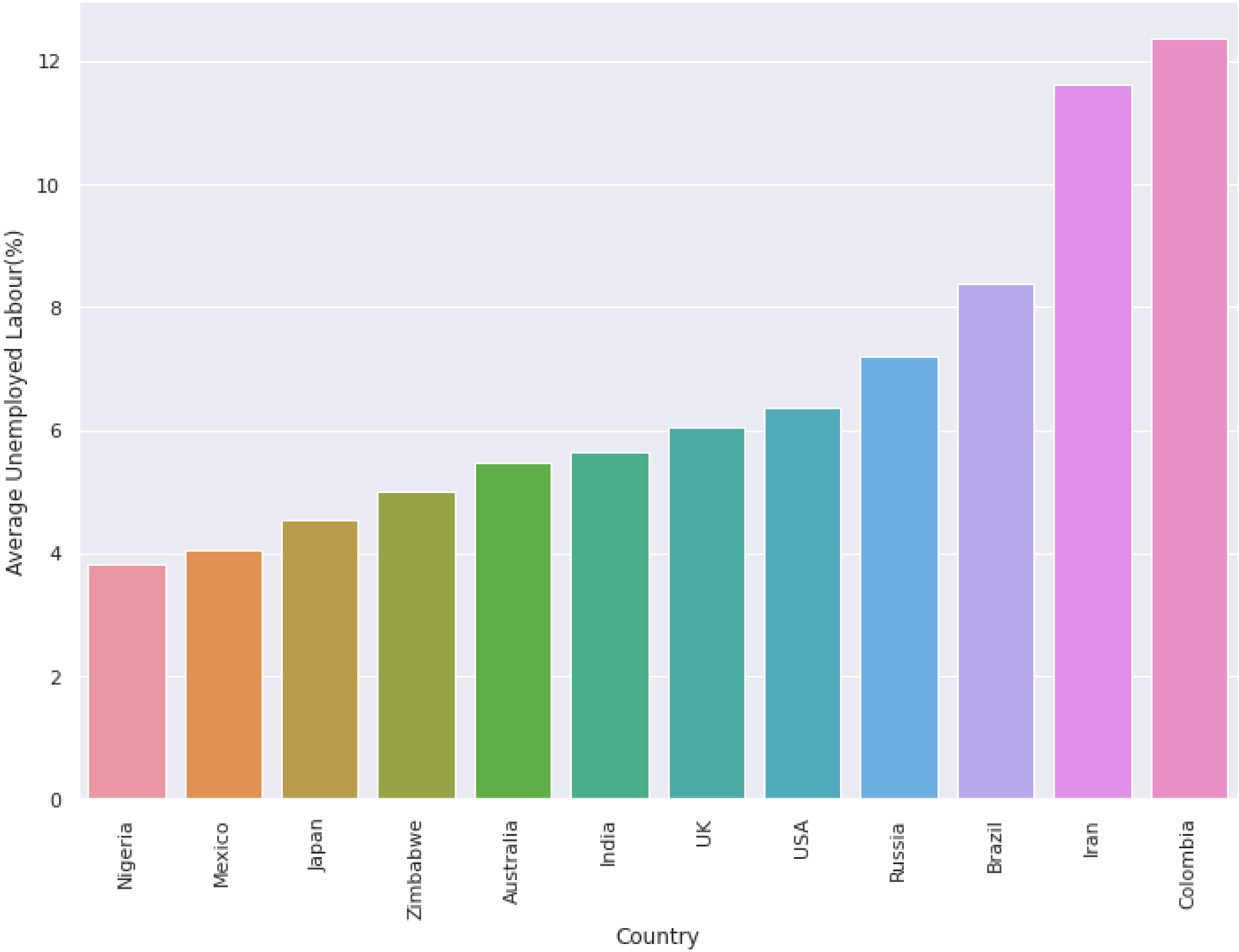


Average Population Density
300
250
200
150
100
50
0
Australia
Russia
Brazil
Zimbabwe
USA
Colombia
Iran
Mexico
Nigeria
UK
Japan
India
Country

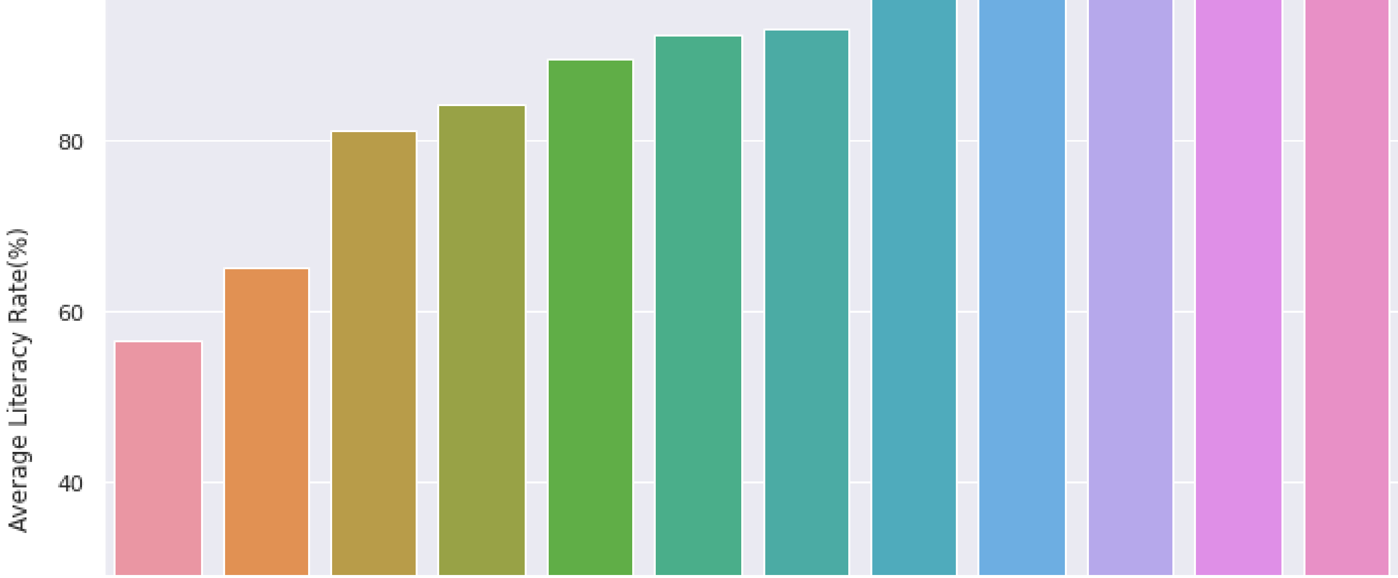


1e6
l greenhouse gas emissions (kt of CO2 equivalent)
6
5
4
3
2


Average Tota
1
0
Zimbabwe
Colombia
Nigeria
Australia
Mexico
UK
Iran
Brazil
Japan
India
Russia
USA
Country


Average Unemployed Labour(%)
12
10
8
6
4
2
0
Nigeria
Mexico
Japan
Zimbabwe
Australia
India
UK
USA
Russia
Brazil
Iran
Colombia
Country

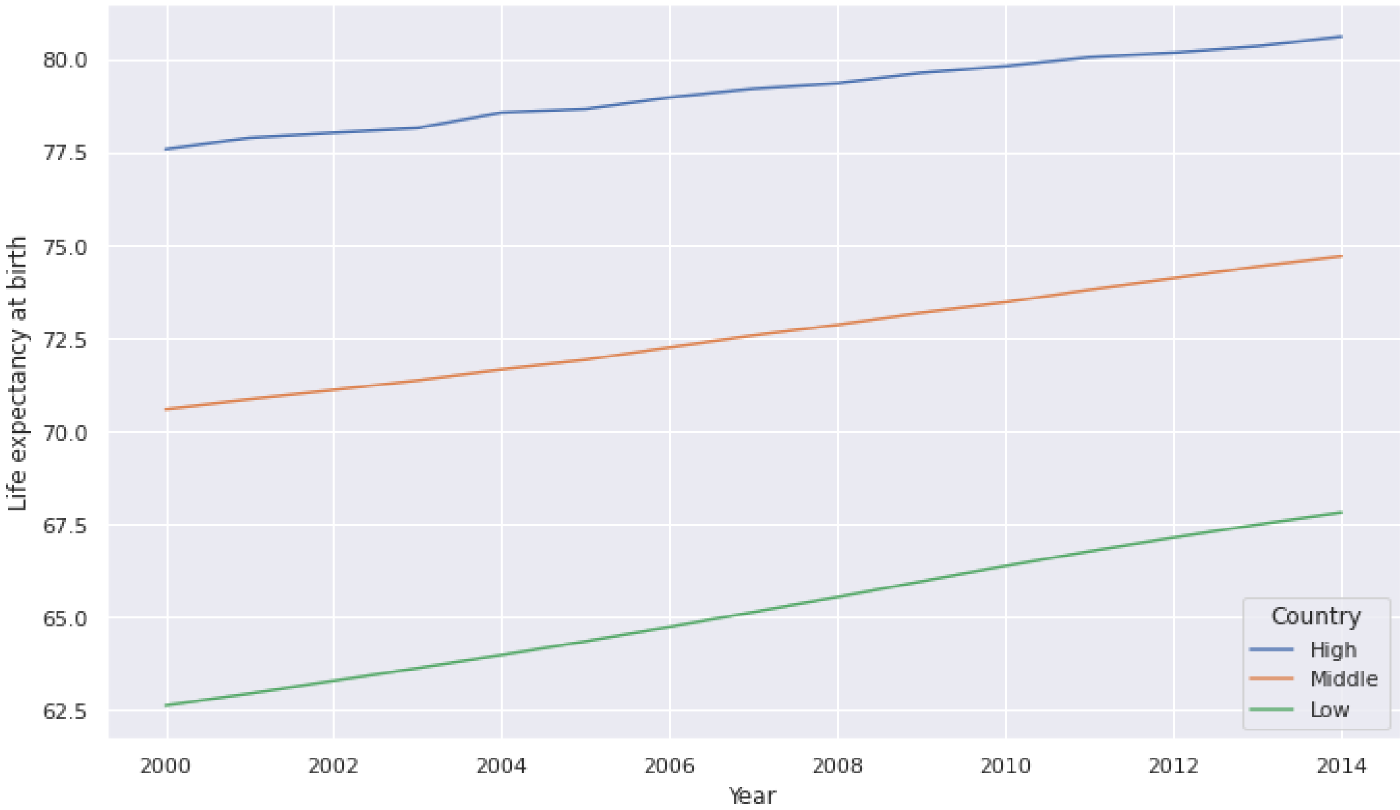

100



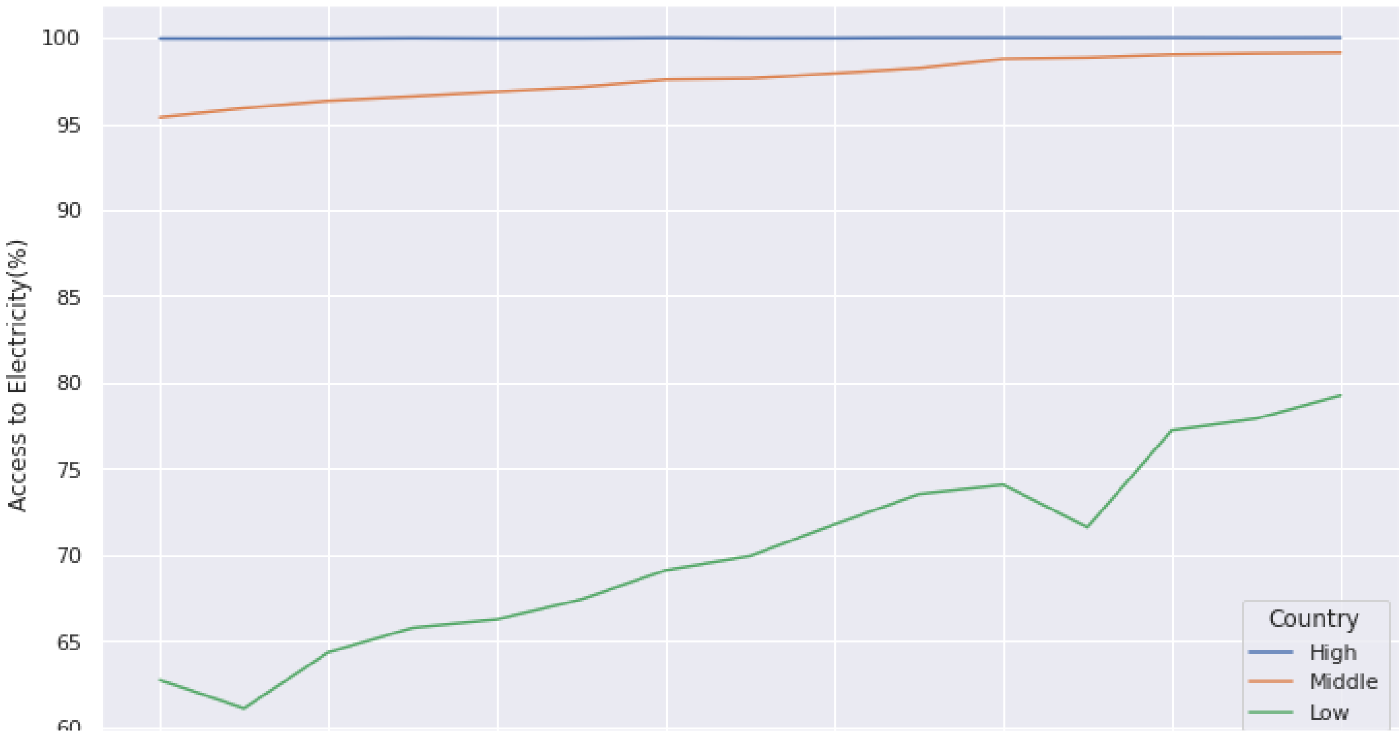
Overall Analaysis for High, Low, and Middle Income Countries

```
df_high_overall = df[df['Country']=='High'].copy()
df_middle_overall = df[df['Country']=='Middle'].copy()
df_low_overall = df[df['Country']=='Low'].copy()

df_comp=df_high_overall.append(df_middle_overall).append(df_low_overall)
df_comp['Immunization'] = df_comp[['Immunization(DPT %)', 'Immunization(Measles %)']].mean(axis=1)
```

```
from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 20000})'''))

for i,col in enumerate(bar_params):
  fig = plt.figure(figsize=(12,7))
  sns.lineplot(data=df_comp, x="Year", y=col,hue="Country")
```

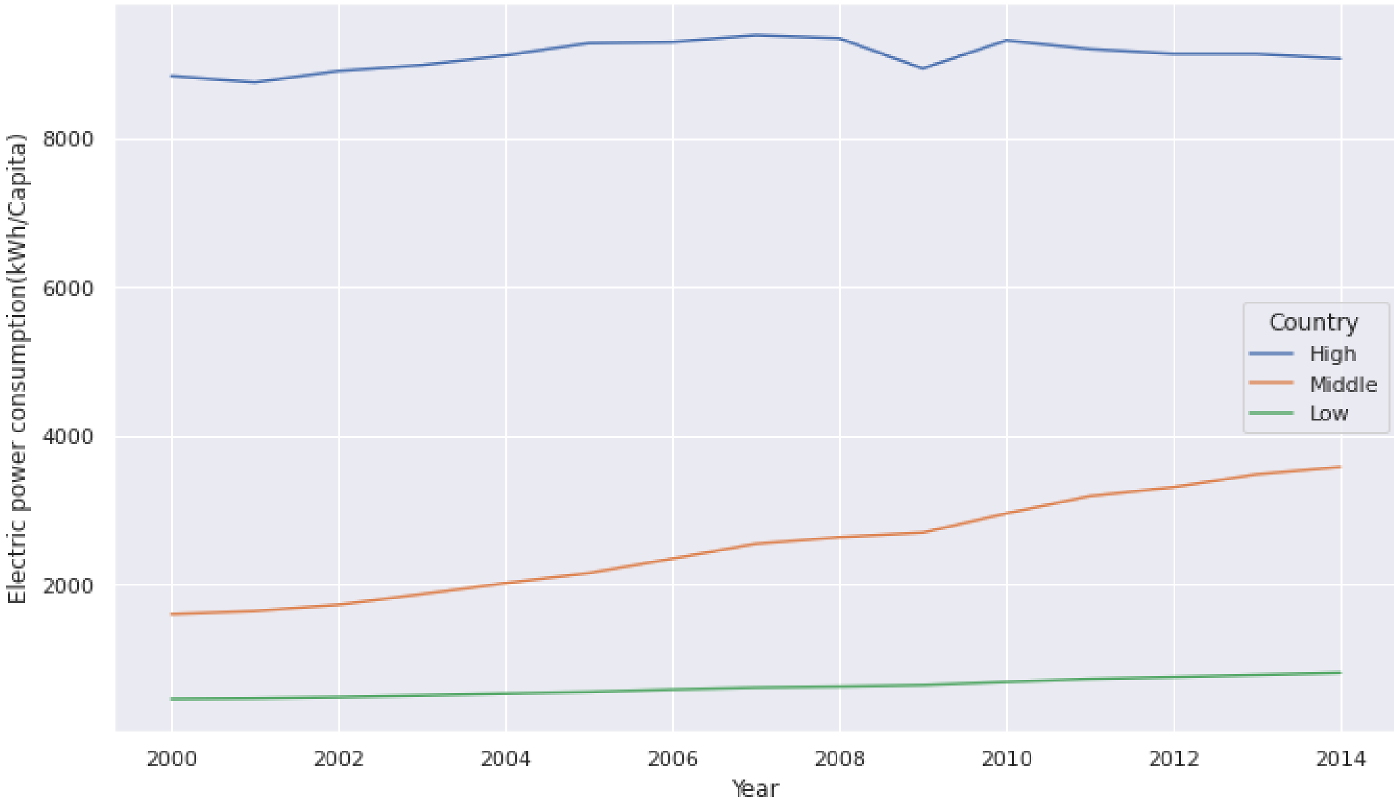


Life expectancy at birth
80.0
77.5
75.0
72.5
70.0
67.5
65.0
62.5
2000
2002
2004
2006
2008
2010
2012
2014
Year
Country
High
Middle
Low



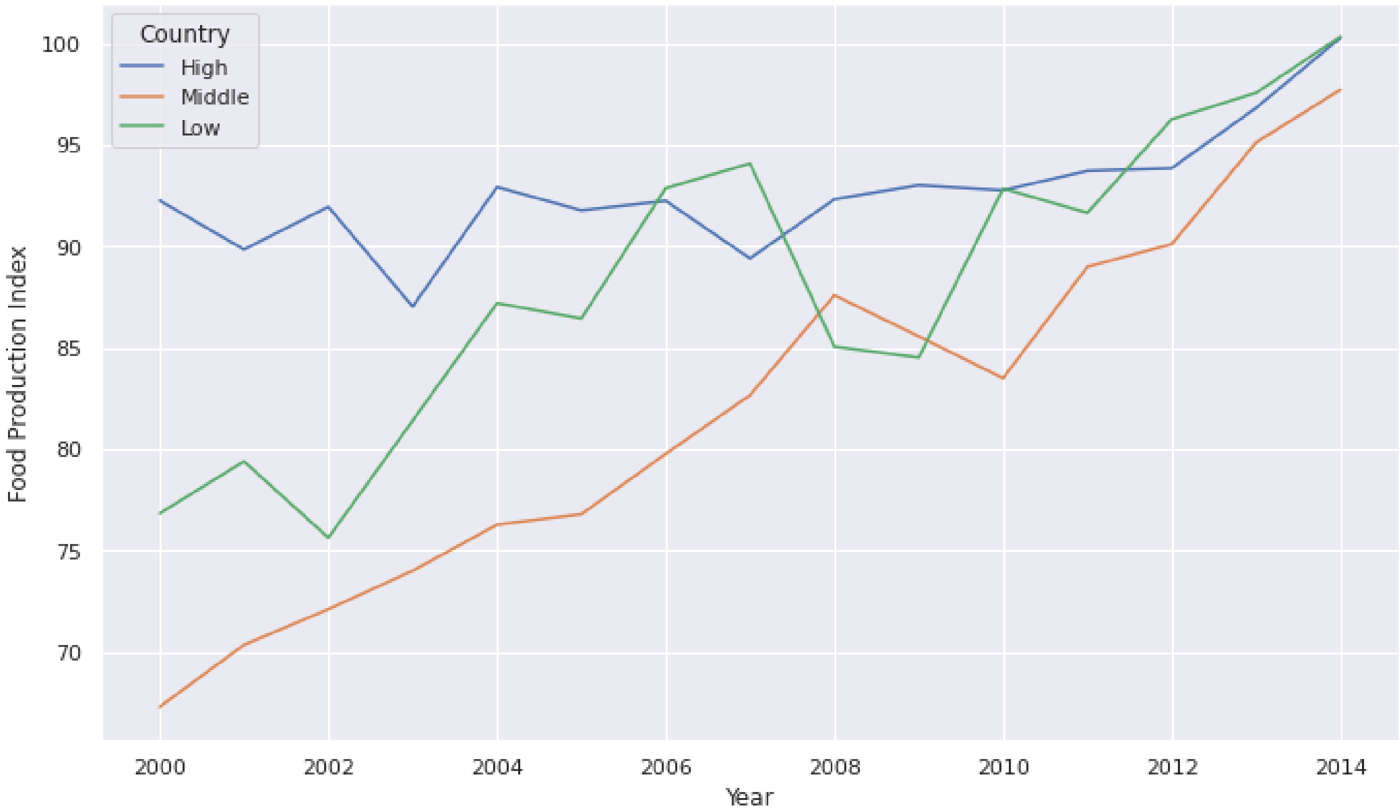
Access to Electricity(%)
100
95
90
85
80
75
70
65
60
Country
High
Middle
Low

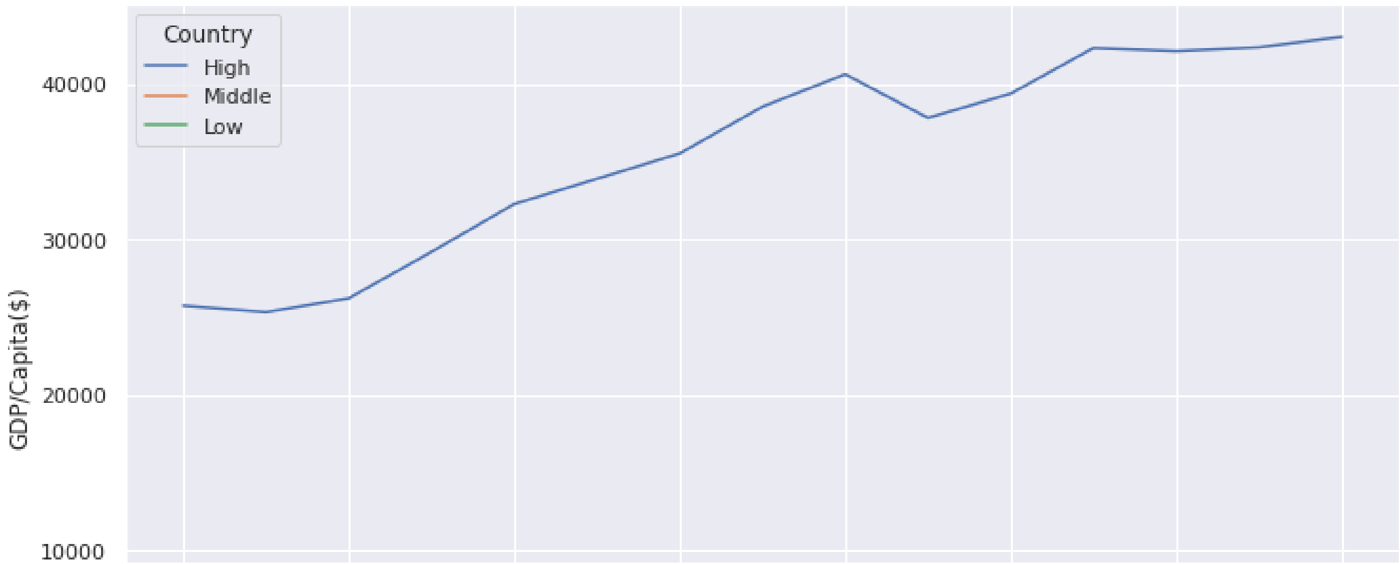

2000
2002
2004
2006
2008
2010
2012
2014
Year

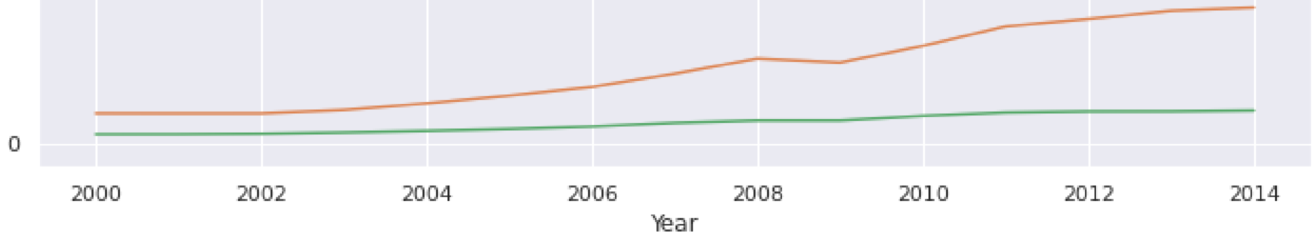

Electric power consumption(kWh/Capita)
8000
6000
4000
2000
2000
2002
2004
2006
2008
2010
2012
2014
Year
Country
High
Middle
Low

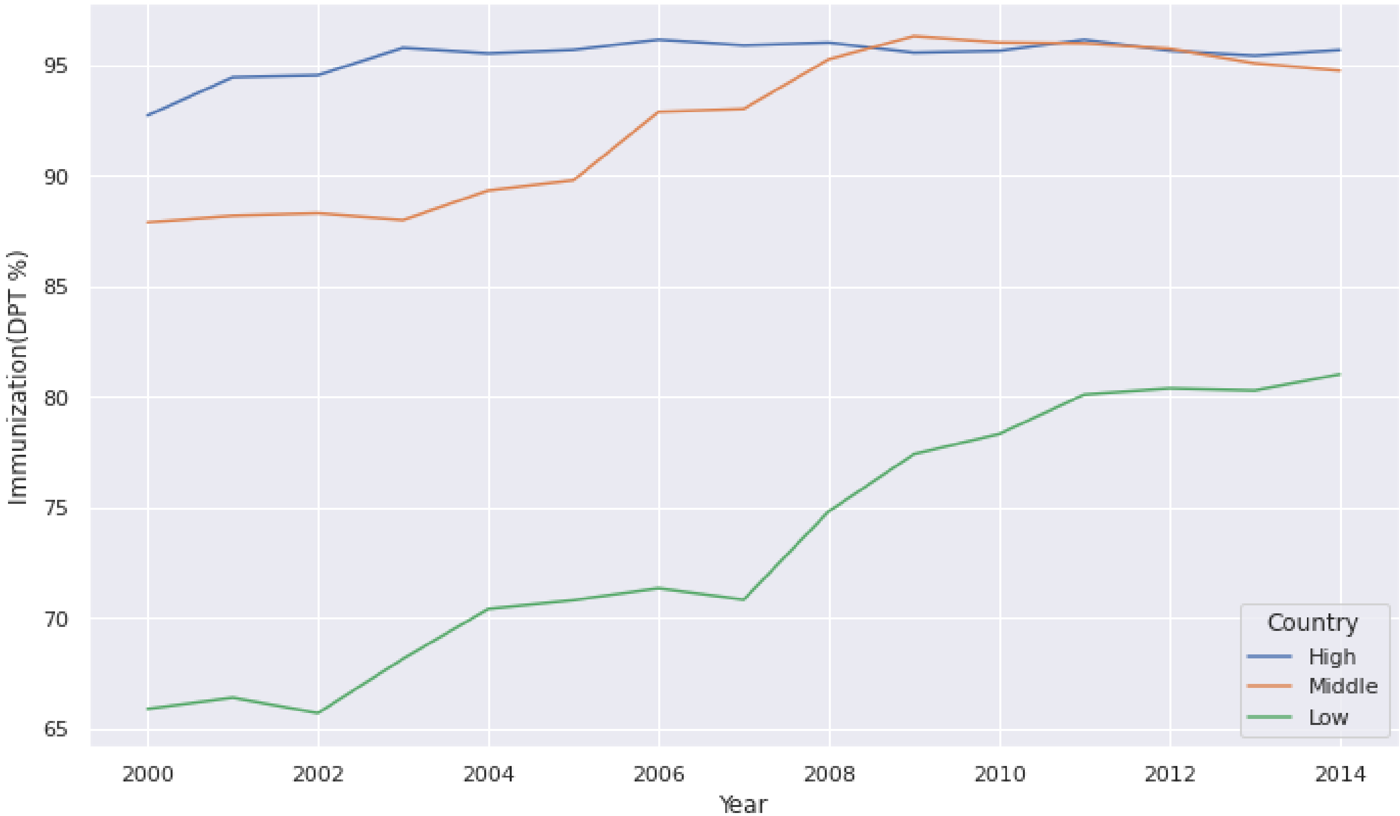

Rural Population(%)
60
50
40
30
Country
High
Middle
Low

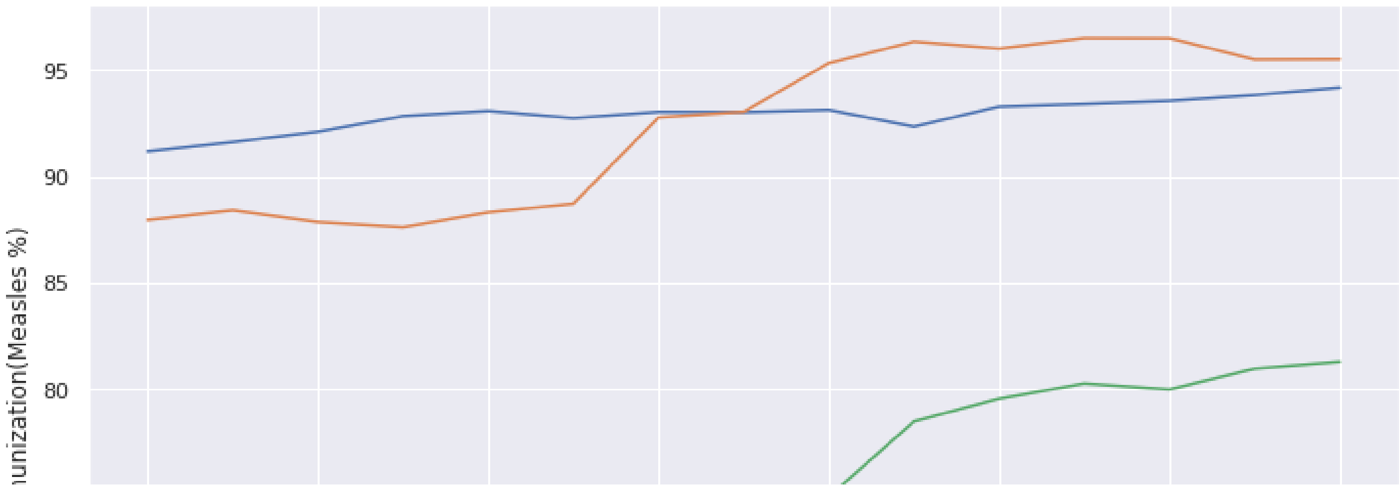

20
2000
2002
2004
2006
2008
2010
2012
2014
Year

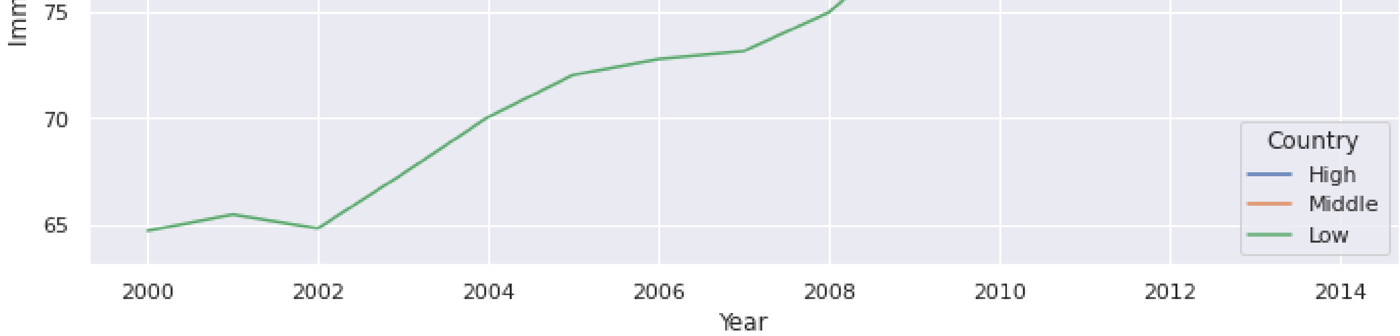

Country
High
Middle
Low
Food Production Index
100
95
90
85
80
75
70
2000
2002
2004
2006
2008
2010
2012
2014
Year


Country
High
Middle
Low
GDP/Capita($)
40000
30000
20000
10000


0
2000
2002
2004
2006
2008
2010
2012
2014
Year


Immunization(DPT %)
95
90
85
80
75
70
65
2000
2002
2004
2006
2008
2010
2012
2014
Year
Country
High
Middle
Low

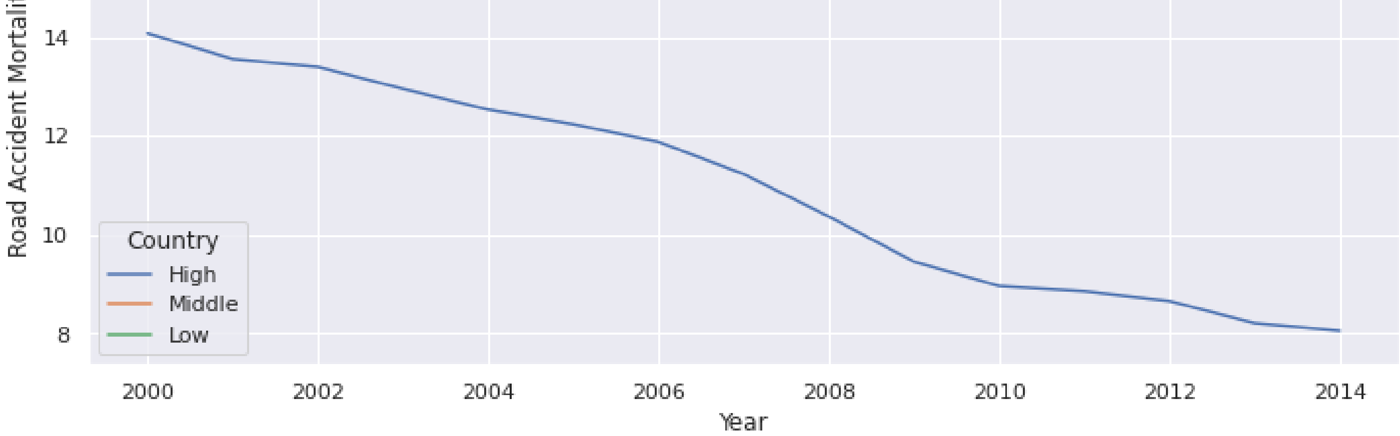

95
90
85
80
2000
2002
2004
2006
2008
2010
2012
2014

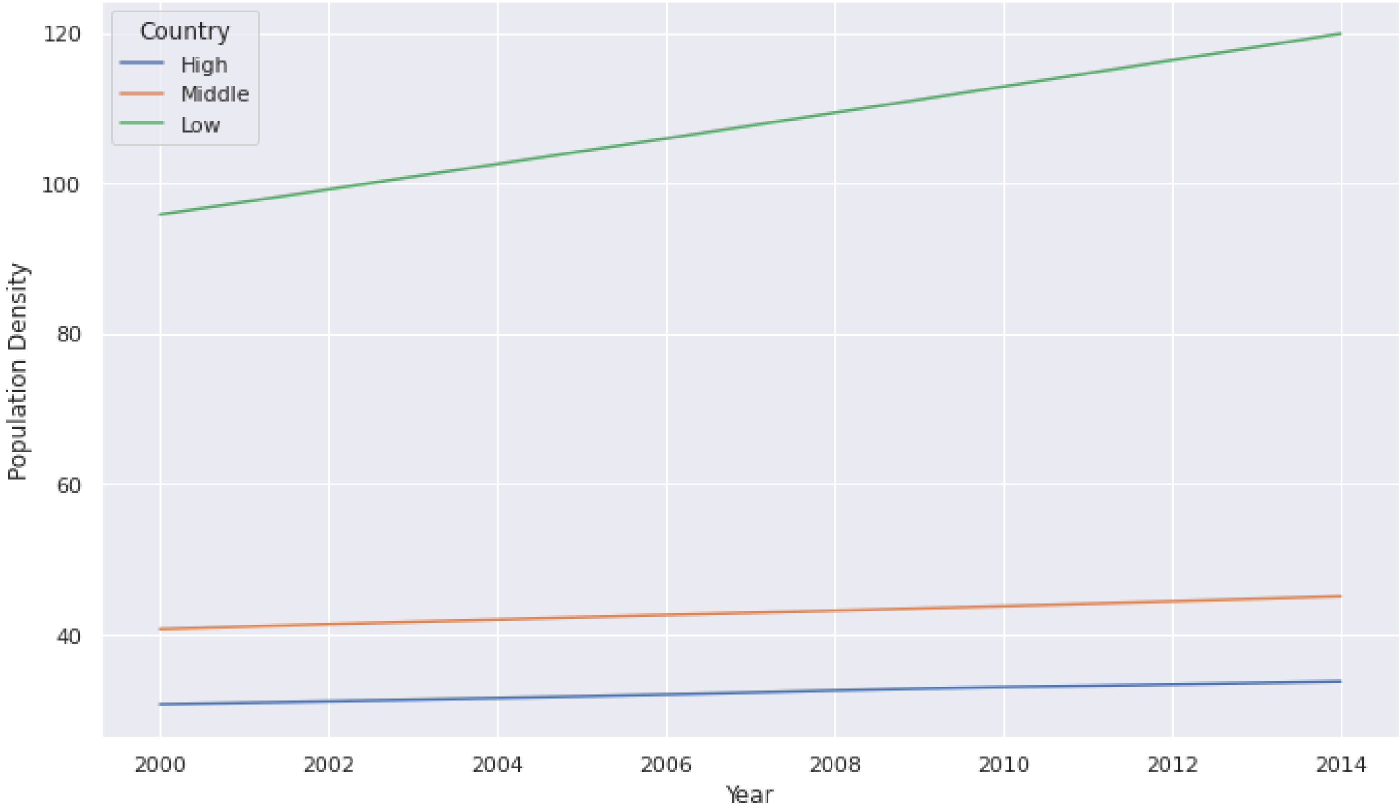


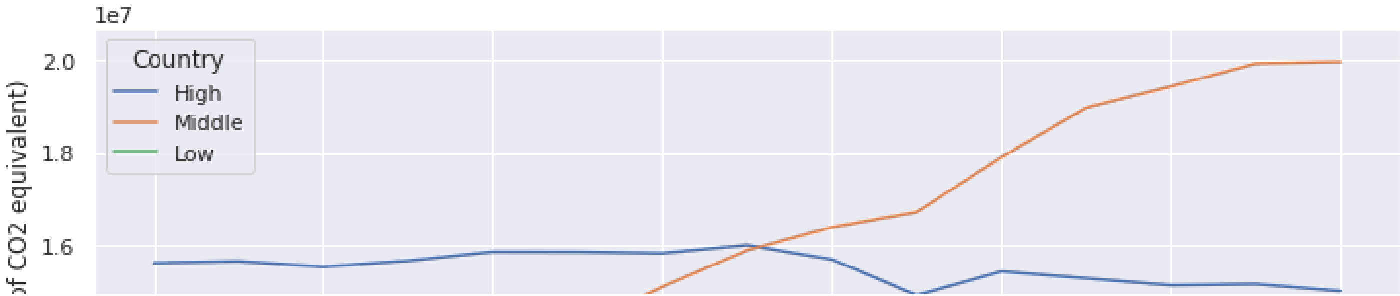
Imm
75
70
65
Country
High
Middle
Low
2000
2002
2004
2006
2008
2010
2012
2014
Year

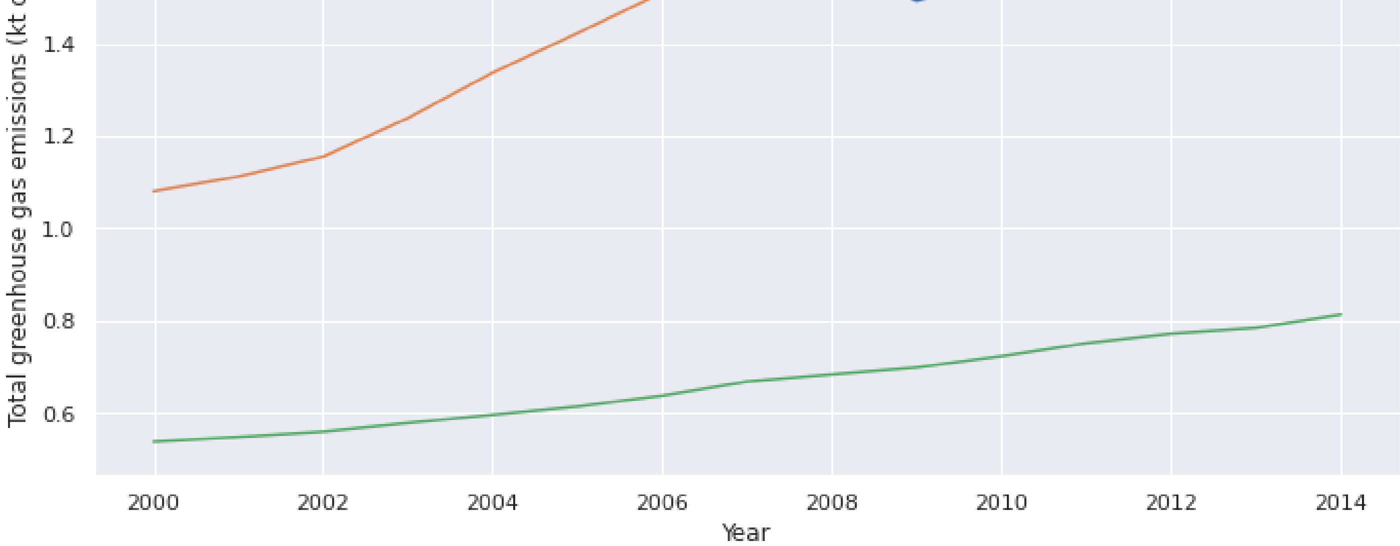
GDP Deflator
10
8
6
4
2
Country
High
Middle
Low
2000
2002
2004
2006
2008
2010
2012
2014
Year

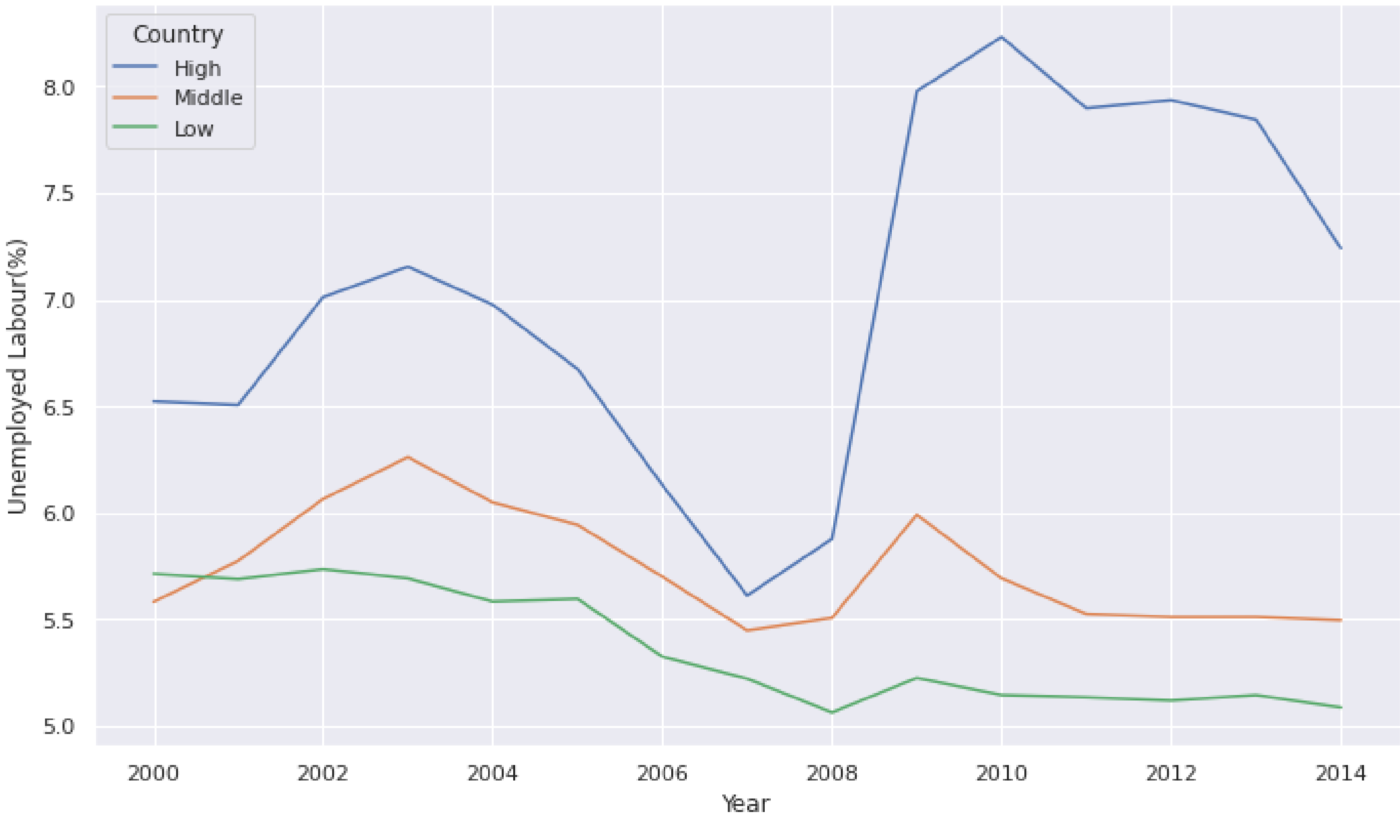
ty(per 100,000 people)
22
20
18
16

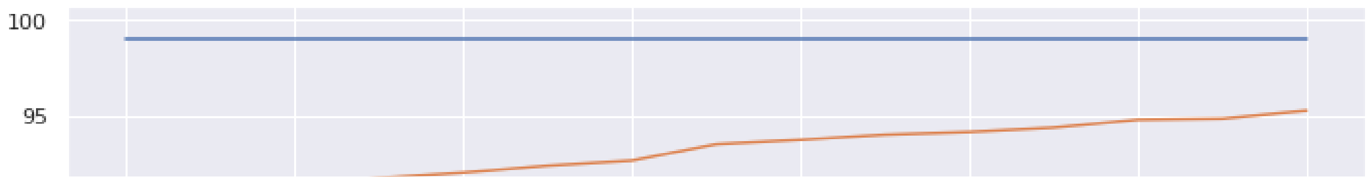

Road Accident Mortali
Country
High
Middle
Low
Year

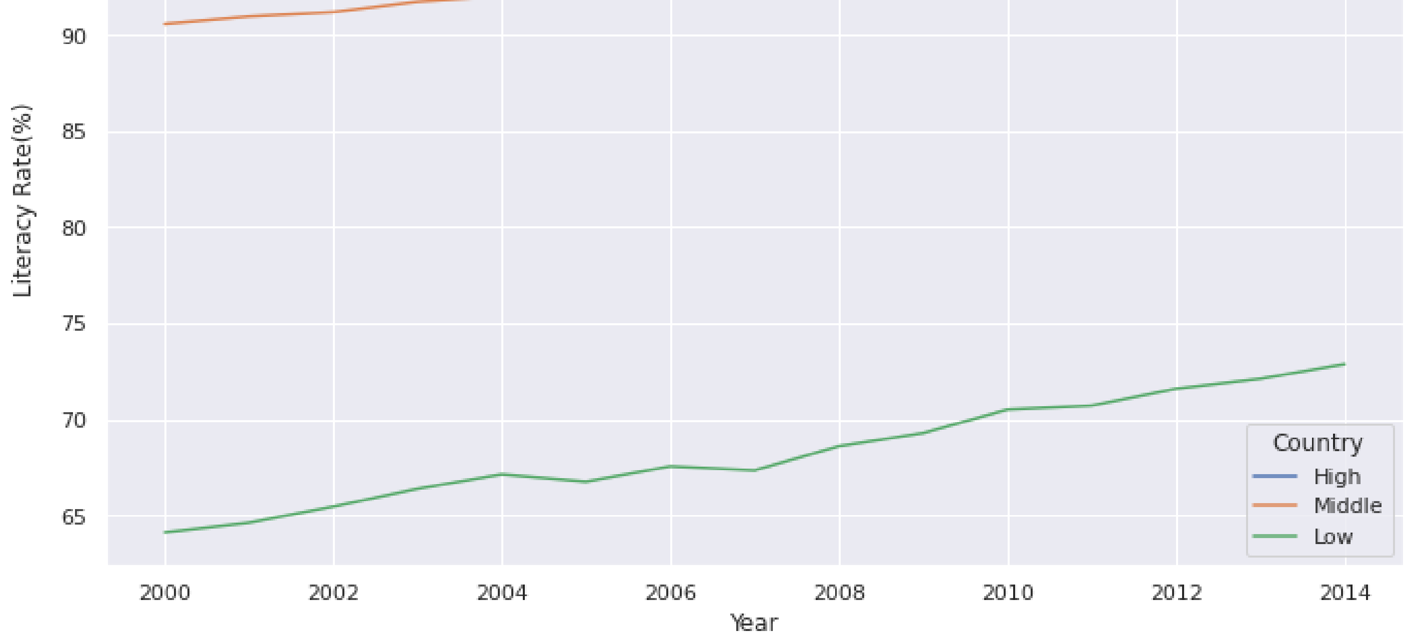

Country
High
Middle
Low
Population Density
Year


1e7
Country
High
Middle
Low
of CO2 equivalent)


Total greenhouse gas emissions (kt
1.4
1.2
1.0
0.8
0.6
2000
2002
2004
2006
2008
2010
2012
2014
Year


Country
High
Middle
Low
Unemployed Labour(%)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
2000
2002
2004
2006
2008
2010
2012
2014
Year


100
95

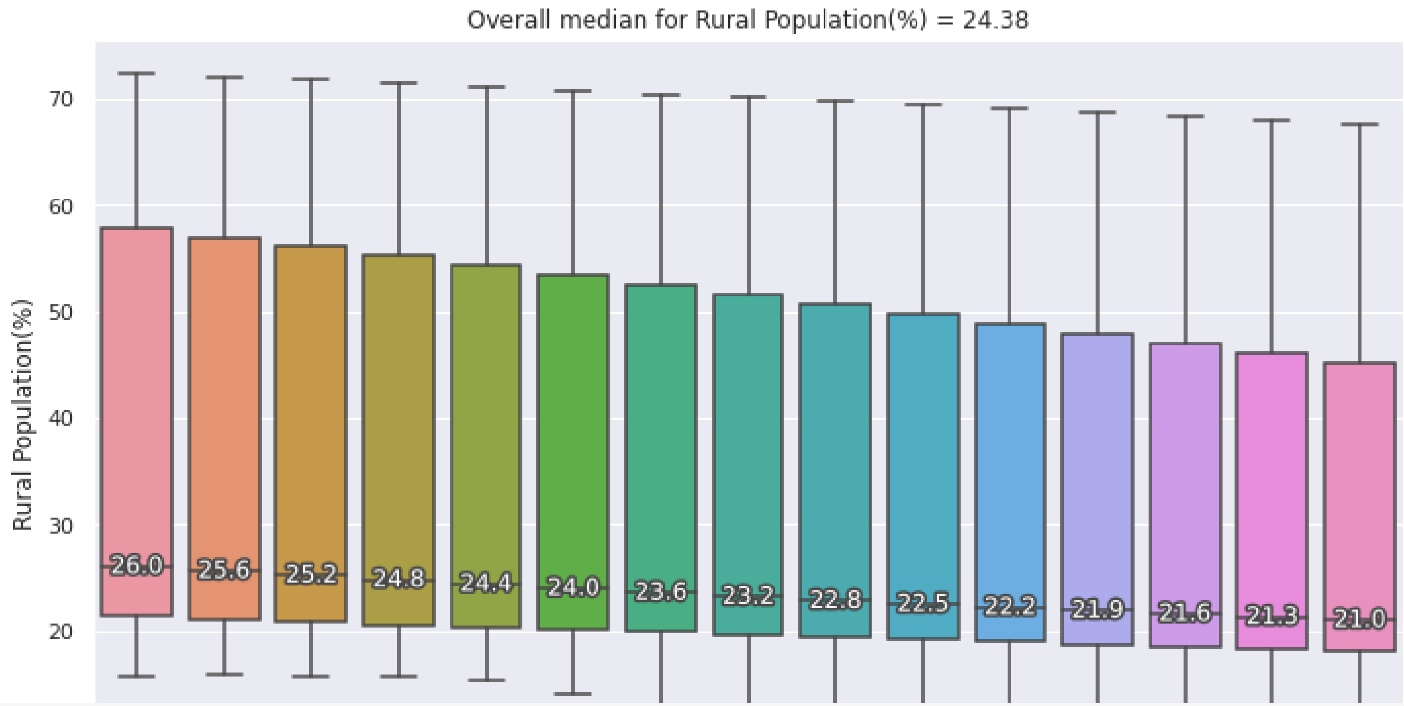



```
loop_params = ['Rural Population(%)', 'Life expectancy at birth']
```

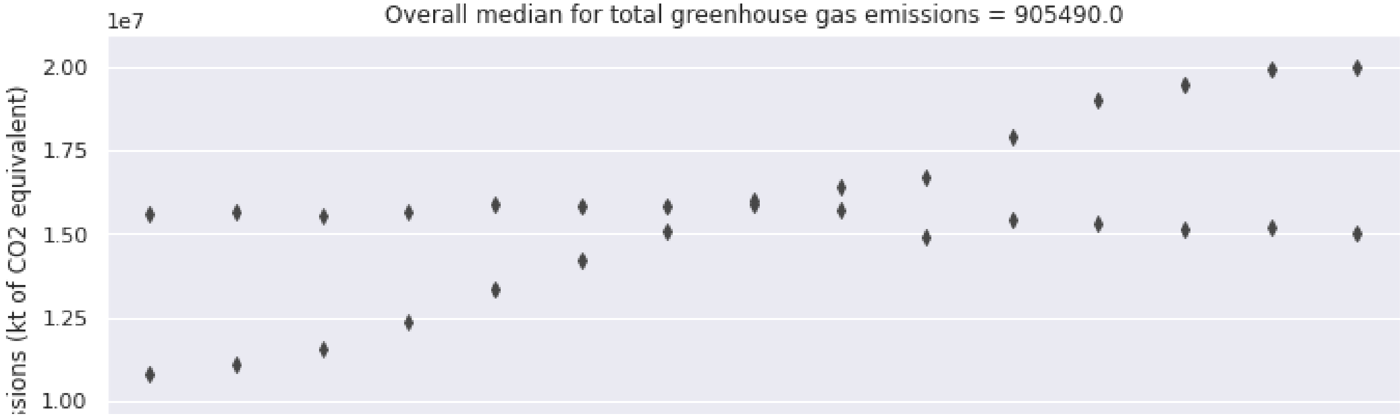
```
# Reference: https://stackoverflow.com/questions/38649501/labeling-boxplot-in-seaborn-with-median-value
import matplotlib.patheffects as path_effects
def add_median_labels(ax, precision='.1f'):
    lines = ax.get_lines()
    boxes = [c for c in ax.get_children() if type(c).__name__ == 'PathPatch']
    lines_per_box = int(len(lines) / len(boxes))
    for median in lines[4:len(lines):lines_per_box]:
        x, y = (data.mean() for data in median.get_data())
        value = x if (median.get_xdata()[1] - median.get_xdata()[0]) == 0 else y
        text = ax.text(x, y, f'{value:{precision}}', ha='center', va='center',
                       fontweight='normal', color='white')
        text.set_path_effects([
            path_effects.Stroke(linewidth=3, foreground=median.get_color()),
            path_effects.Normal(),
        ])
```

```
for param in loop_params:
```

```
  plt.figure(figsize = (12,7))
  box_plot = sns.boxplot(x = 'Year', y = param, data = df)
  plt.xticks(rotation = 90)
  plt.xlabel(xlabel = 'Year', fontsize = 15)
  plt.title('Overall median for '+ param + ' = '+ str(df[param].median()))
  add_median_labels(box_plot.axes)
  plt.show()

from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 20000})'''))
```

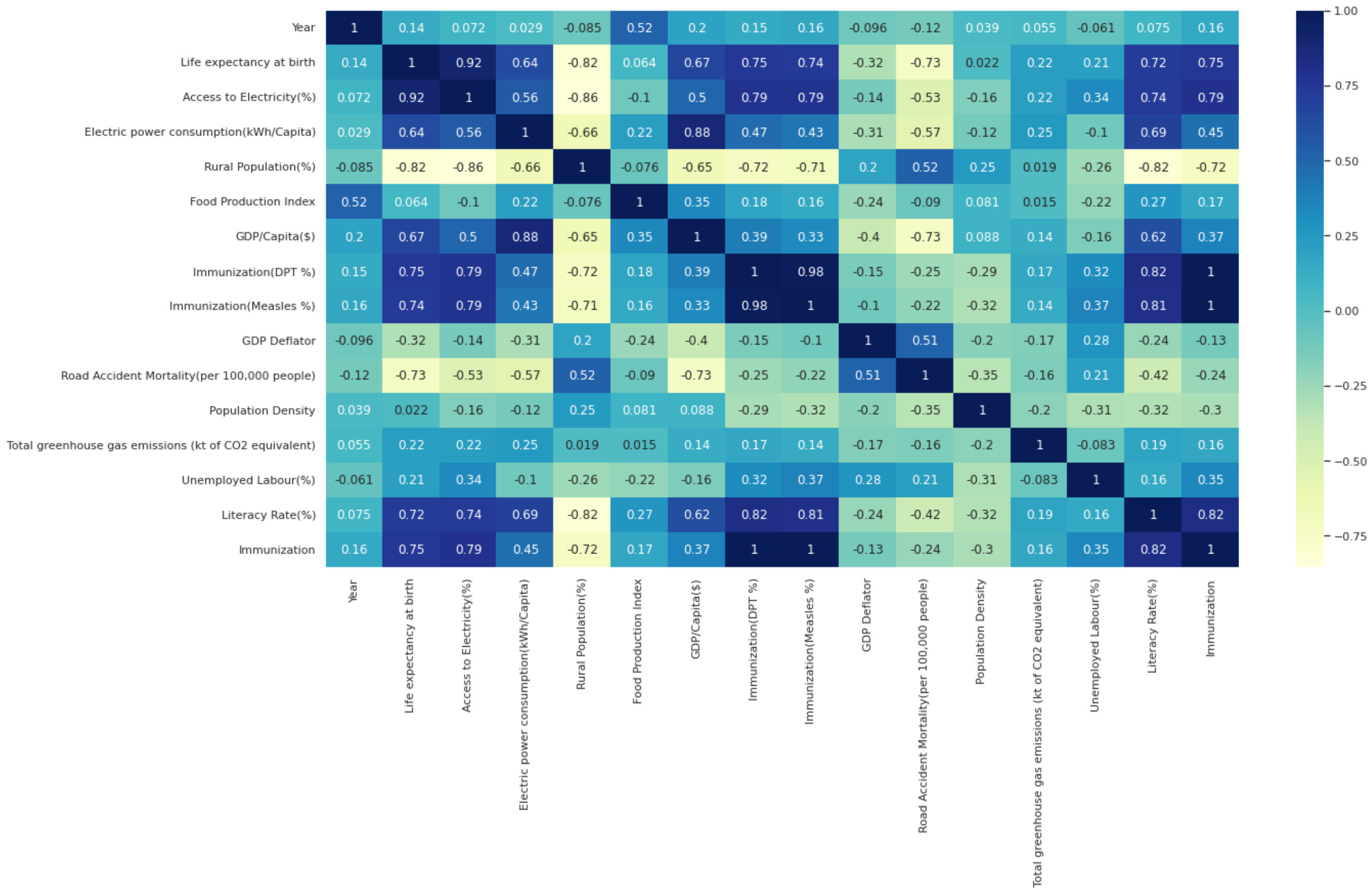



```
plt.figure(figsize = (12,7))
box_plot = sns.boxplot(x = 'Year', y = 'Total greenhouse gas emissions (kt of CO2 equivalent)', data = df)
plt.xticks(rotation = 90)
plt.xlabel(xlabel = 'Year', fontsize = 15)
plt.title('Overall median for total greenhouse gas emissions = ' + str(df['Total greenhouse gas emissions (kt of CO2 equivalent)'].median()))
#add_median_labels(box_plot.axes)
plt.show()
```



- Insights:
    - Rural population median has been decreasing over the years.
    - Greenhouse gas emissions and life expectancy medians have been increasing over the years.

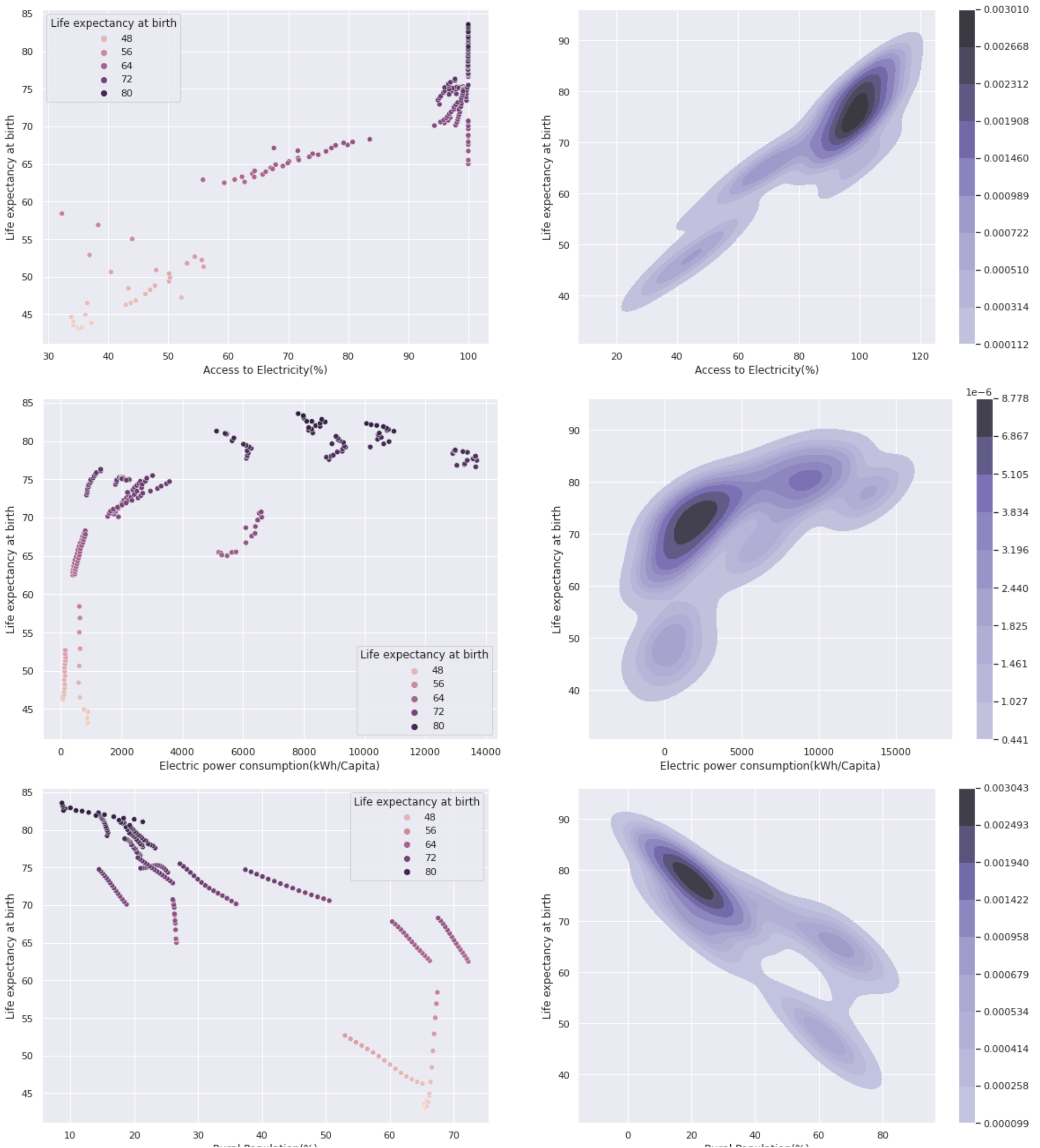
```
plt.figure(figsize = (20,10))
sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
plt.show()
```

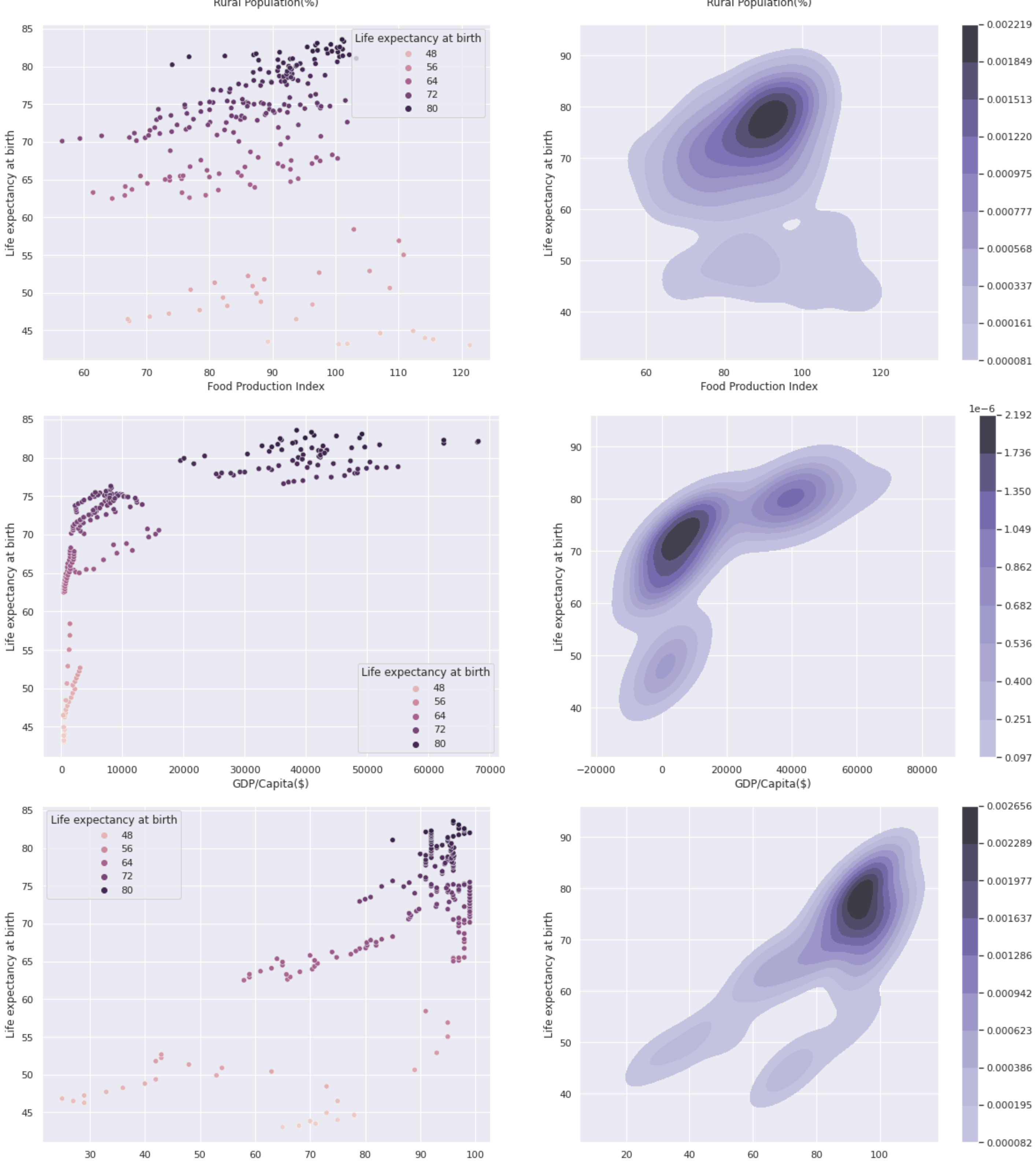


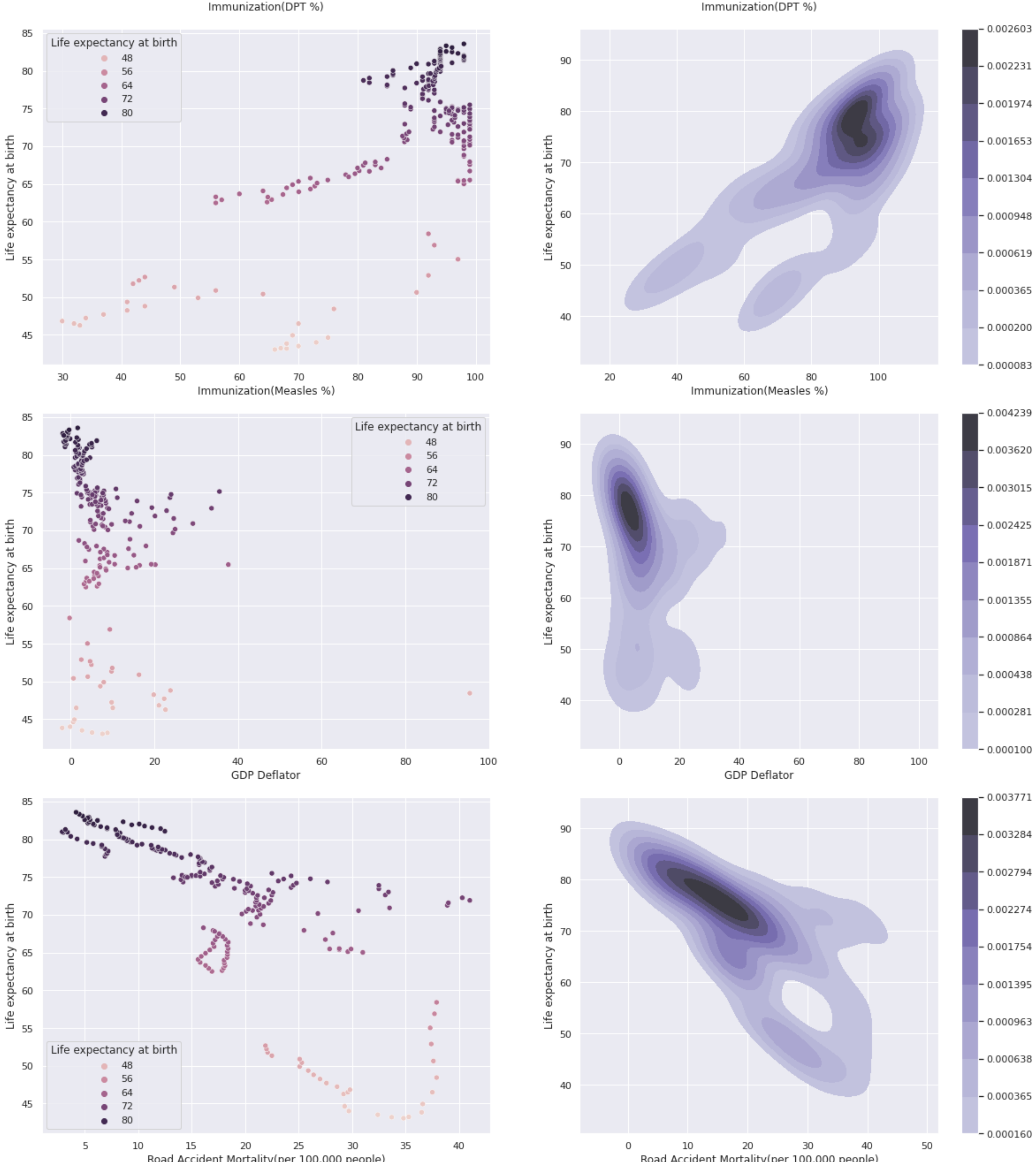

```
countries = list(df['Country'].unique())
countries.remove('High')
countries.remove('Low')
countries.remove('Middle')

scatter_params = list(df.columns)
scatter_params.remove('Immunization')
scatter_params.remove('Year')
scatter_params.remove('Country')
scatter_params.remove('Life expectancy at birth')
print(scatter_params)
```

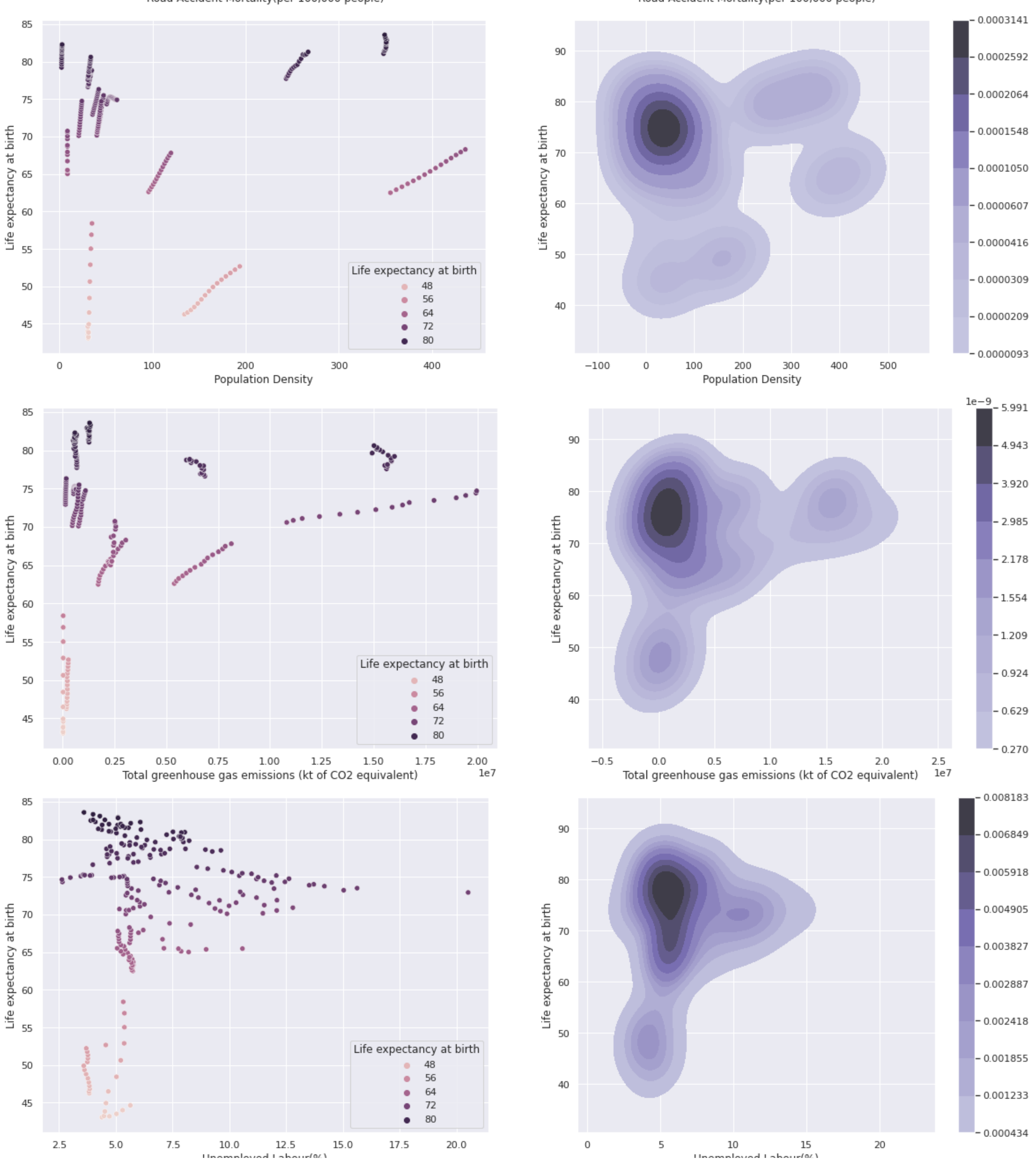
```
['Access to Electricity(%)', 'Electric power consumption(kWh/Capita)', 'Rural Population(%)', 'Food Production Index', 'GDP/Capita($)', '
```

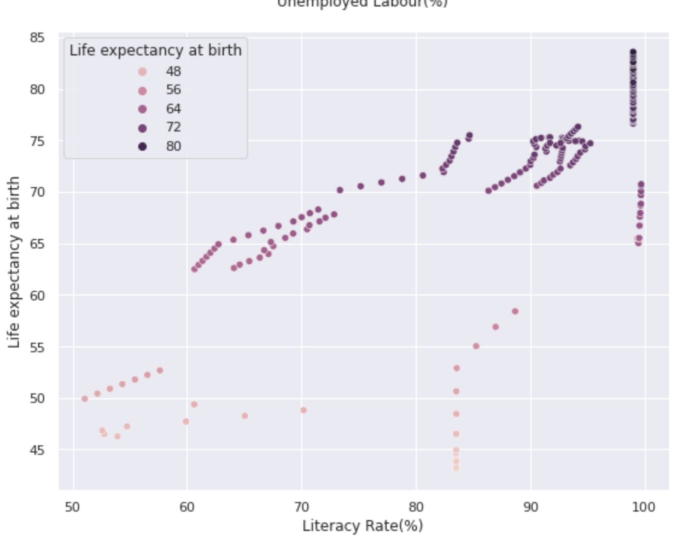
```
from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 20000})'''))

import matplotlib.pyplot as plt
import numpy as np
import pandas
for param in scatter_params :
  plt.figure(figsize=(20,7))
  plt.subplot(1,2,1)
  sns.scatterplot(x=param, y='Life expectancy at birth', data=df ,hue='Life expectancy at birth')
  plt.subplot(1,2,2)
  res=sns.kdeplot(df[param],df['Life expectancy at birth'],shade=True,cmap="Purples_d",cbar=True)
  plt.show();
```


Life expectancy at birth
48
56
64
72
80
Access to Electricity(%)
Electric power consumption(kWh/Capita)
1e−6

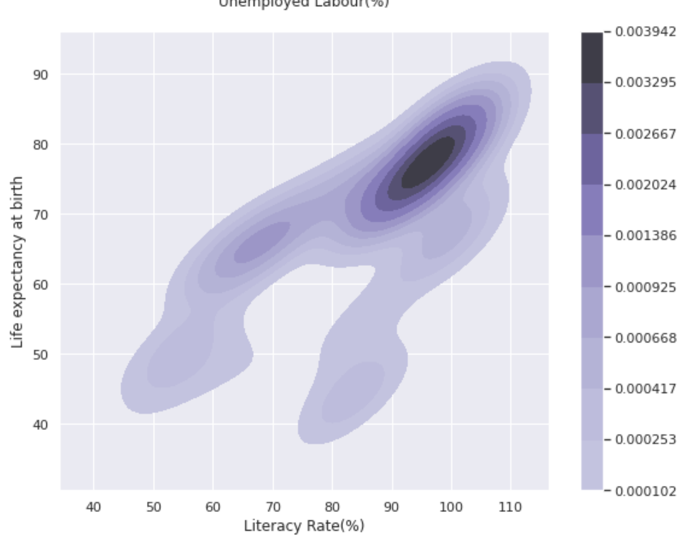

Rural Population(%)
Rural Population(%)
Life expectancy at birth
48
56
64
72
80
Life expectancy at birth
Food Production Index
Life expectancy at birth
Food Production Index
0.002219
0.001849
0.001513
0.001220
0.000975
0.000777
0.000568
0.000337
0.000161
0.000081
Life expectancy at birth
GDP/Capita($)
Life expectancy at birth
GDP/Capita($)
1e−6
2.192
1.736
1.350
1.049
0.862
0.682
0.536
0.400
0.251
0.097
Life expectancy at birth
Life expectancy at birth
0.002656
0.002289
0.001977
0.001637
0.001286
0.000942
0.000623
0.000386
0.000195
0.000082


Immunization(DPT %)
Immunization(DPT %)
Life expectancy at birth
48
56
64
72
80
Life expectancy at birth
Immunization(Measles %)
Immunization(Measles %)
0.002603
0.002231
0.001974
0.001653
0.001304
0.000948
0.000619
0.000365
0.000200
0.000083
GDP Deflator
GDP Deflator
0.004239
0.003620
0.003015
0.002425
0.001871
0.001355
0.000864
0.000438
0.000281
0.000100
Road Accident Mortality(per 100,000 people)
Road Accident Mortality(per 100,000 people)
0.003771
0.003284
0.002794
0.002274
0.001754
0.001395
0.000963
0.000638
0.000365
0.000160


Life expectancy at birth
Population Density
Life expectancy at birth
48
56
64
72
80
Total greenhouse gas emissions (kt of CO2 equivalent)
1e7
1e−9



Unemployed Labour(%)
Life expectancy at birth
48
56
64
72
80
Life expectancy at birth
Literacy Rate(%)



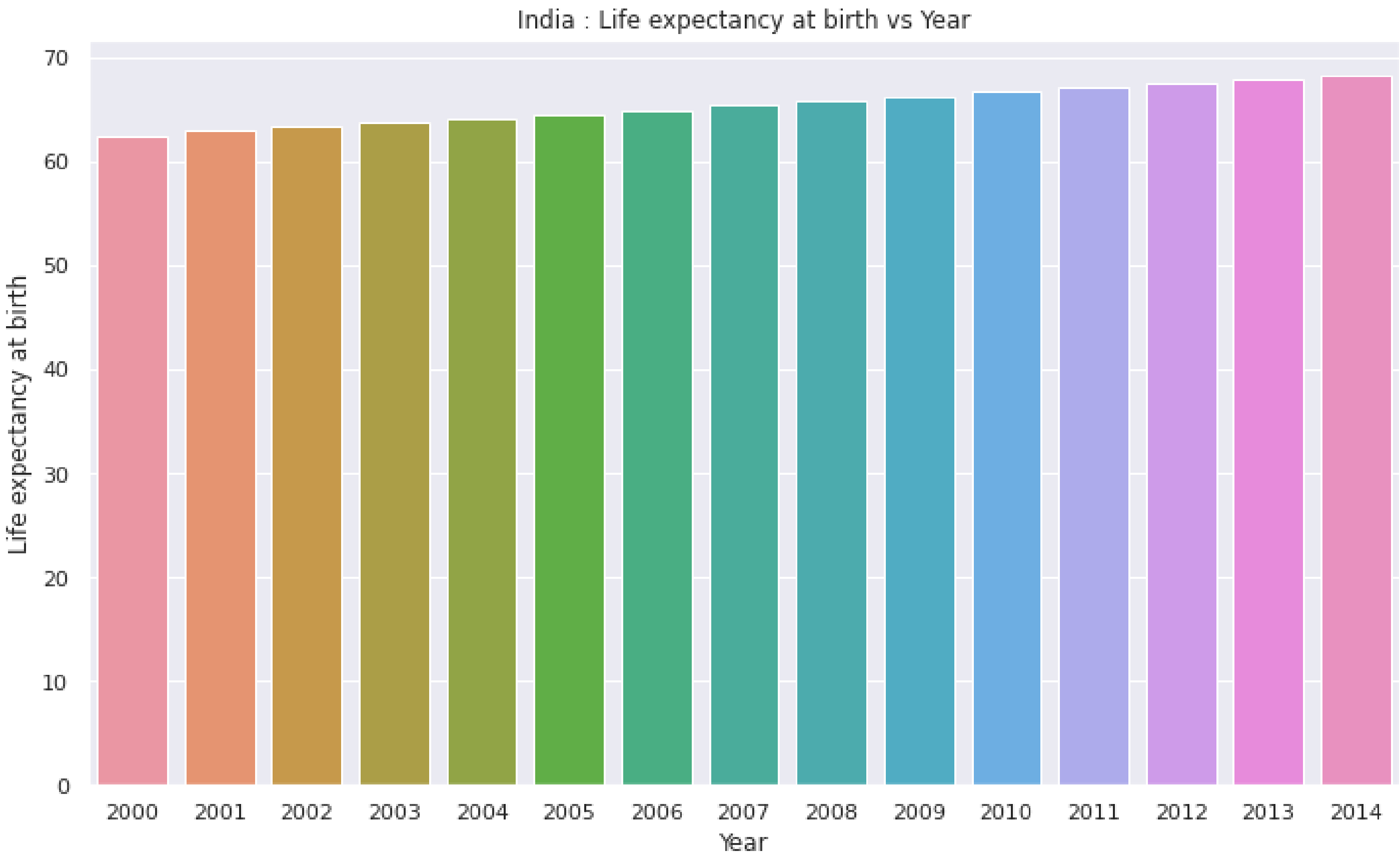
Unemployed Labour(%)
Life expectancy at birth
Literacy Rate(%)
0.003942
0.003295
0.002667
0.002024
0.001386
0.000925
0.000668
0.000417
0.000253
0.000102

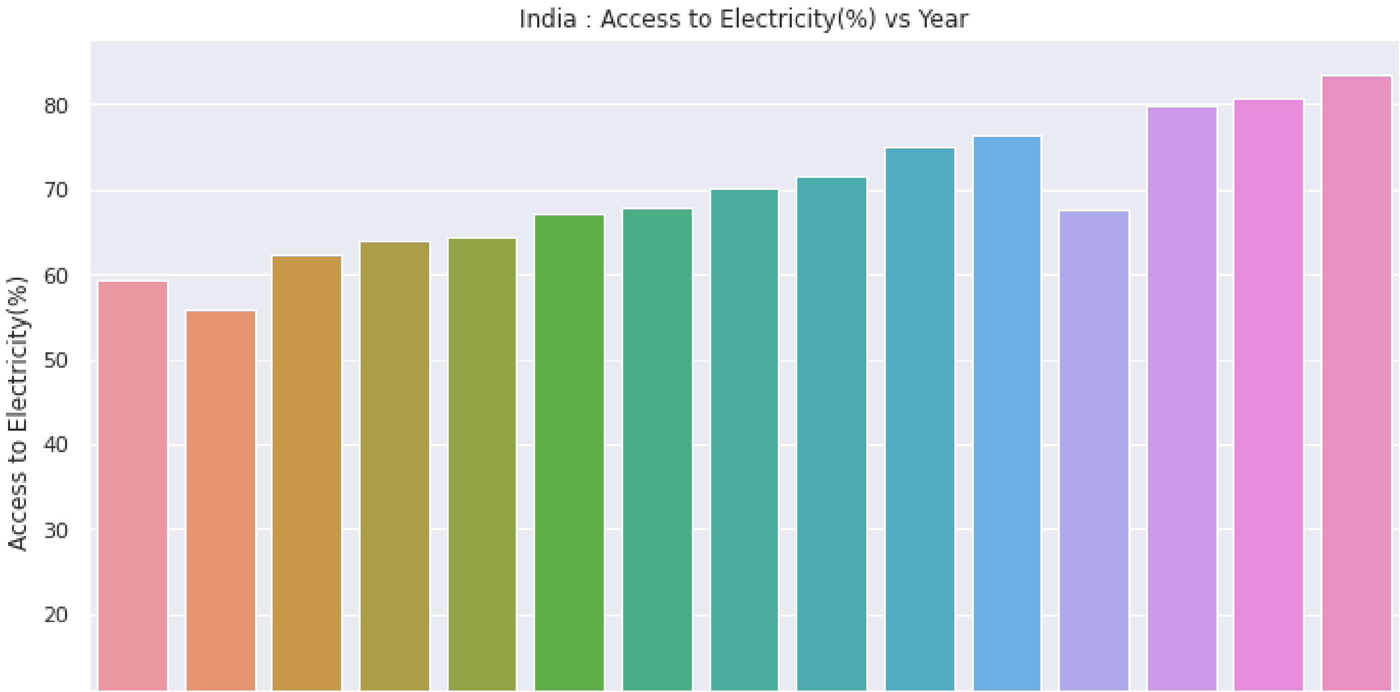
India

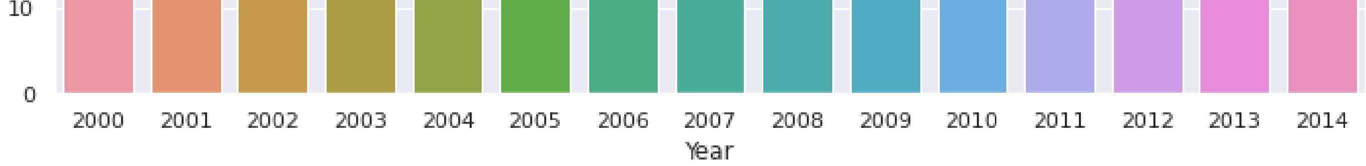
```
df_india = df.loc[df['Country'] == 'India'].copy()
df_india.drop(columns=['Country'], inplace=True)
df_india['Immunization'] = df[['Immunization(DPT %)','Immunization(Measles %)']].mean(axis=1)
df_india.reset_index(drop = True, inplace = True)
df_india
```

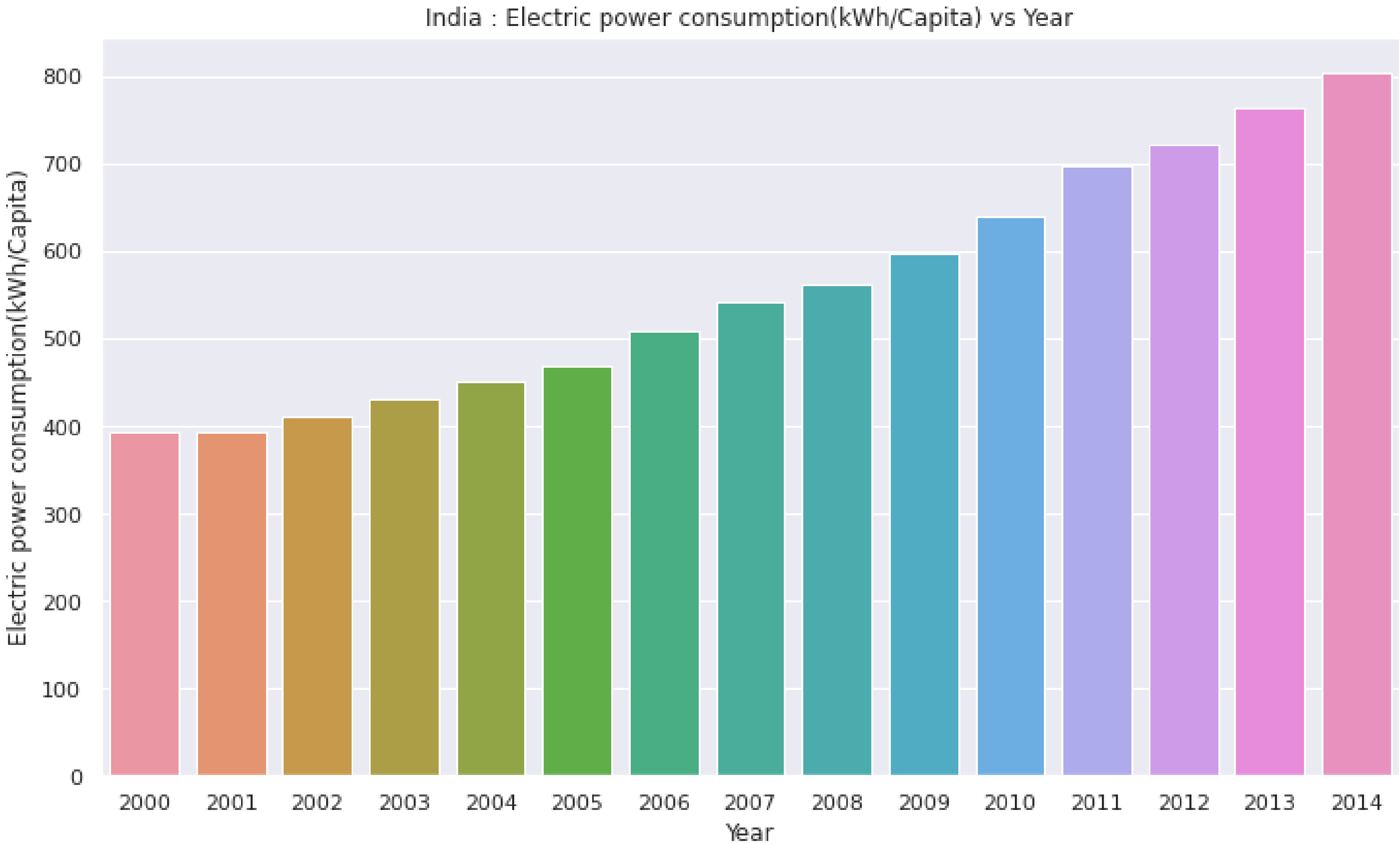
	Year	Life expectancy at birth	Access to Electricity(%)	Electric power consumption(kWh/Capita)	Rural Population(%)	Food Production Index	GDP/Capita($)	Immunization(DPT %)	Immunization(M
0	2000	62.505	59.34105	393.6462	72.333	64.56	443.3142	58.0	
1	2001	62.907	55.80000	393.8102	72.082	66.57	451.5730	59.0	
2	2002	63.304	62.30000	410.6448	71.756	61.53	470.9868	59.0	
3	2003	63.699	64.02313	430.4832	71.428	67.70	546.7266	61.0	
4	2004	64.095	64.40000	451.6115	71.097	66.63	627.7742	63.0	
5	2005	64.500	67.09344	468.0258	70.765	70.14	714.8610	65.0	
6	2006	64.918	67.90000	509.2141	70.431	73.68	806.7533	65.0	
7	2007	65.350	70.13076	541.7384	70.094	79.99	1028.3350	64.0	
8	2008	65.794	71.65108	561.2476	69.754	81.53	998.5223	70.0	
9	2009	66.244	75.00000	598.4982	69.413	79.61	1101.9610	74.0	
10	2010	66.693	76.30000	640.3946	69.070	85.63	1357.5640	79.0	

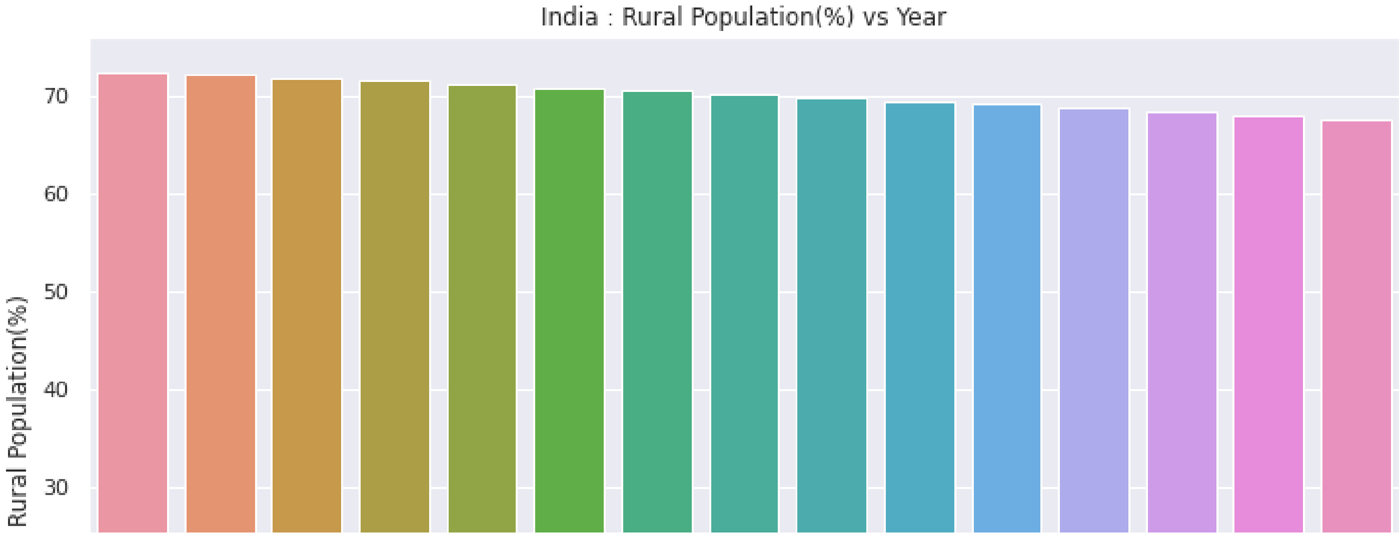
```
from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 20000})'''))

parameters = list(df_india.columns)
parameters.remove('Year')

sns.set(rc={'figure.figsize':(12,7)})
for i,param in enumerate(parameters):
  plt.figure(i)
  sns.barplot(data = df_india, x = 'Year', y = param).set_title('India : '+param+' vs Year')
```

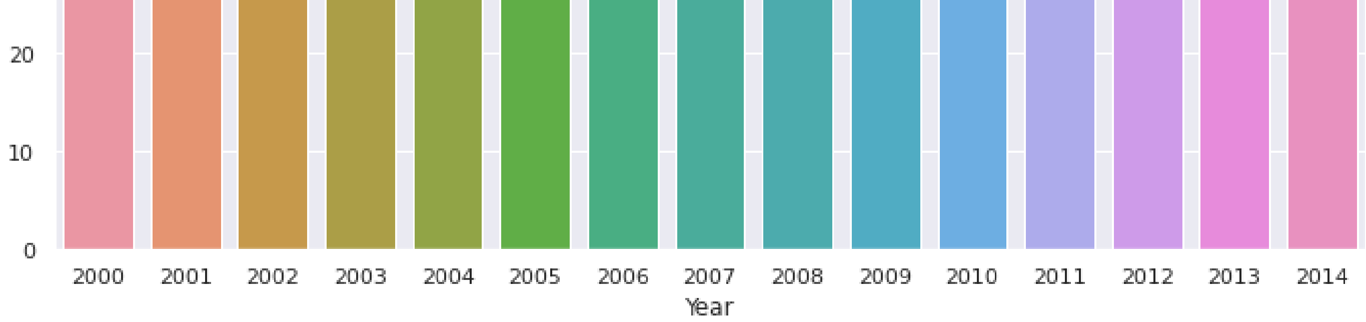

India : Life expectancy at birth vs Year
Life expectancy at birth
0
10
20
30
40
50
60
70
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Year


India : Access to Electricity(%) vs Year
Access to Electricity(%)
20
30
40
50
60
70
80

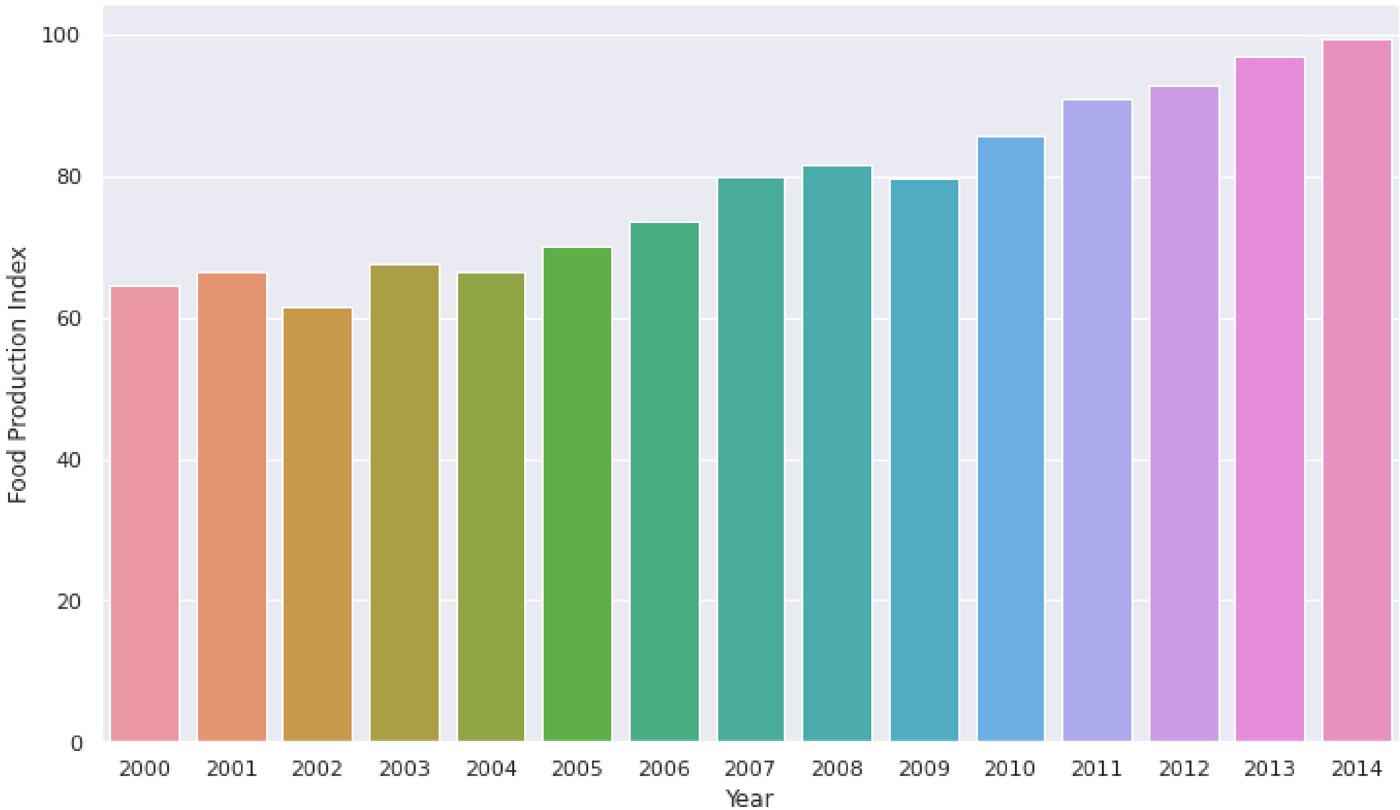

10
0
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
Year


India : Electric power consumption(kWh/Capita) vs Year
Electric power consumption(kWh/Capita)
800
700
600
500
400
300
200
100
0
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
Year

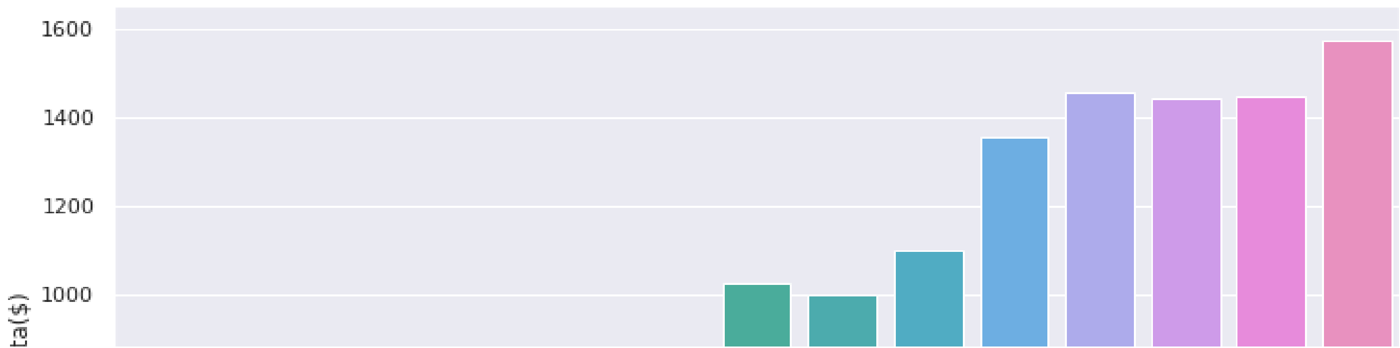

India : Rural Population(%) vs Year
Rural Population(%)
70
60
50
40
30

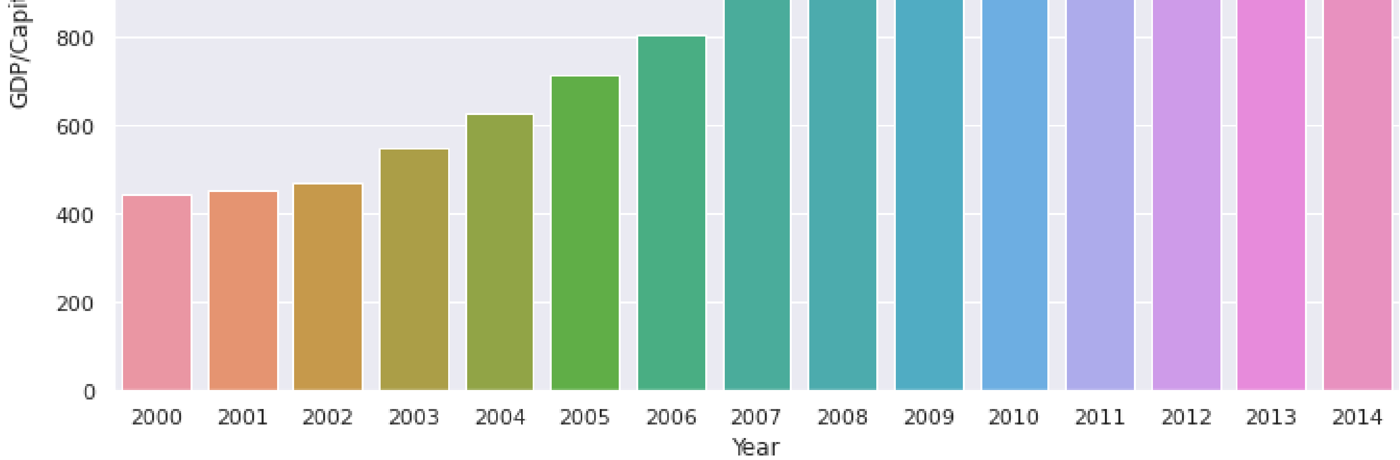


India : Food Production Index vs Year

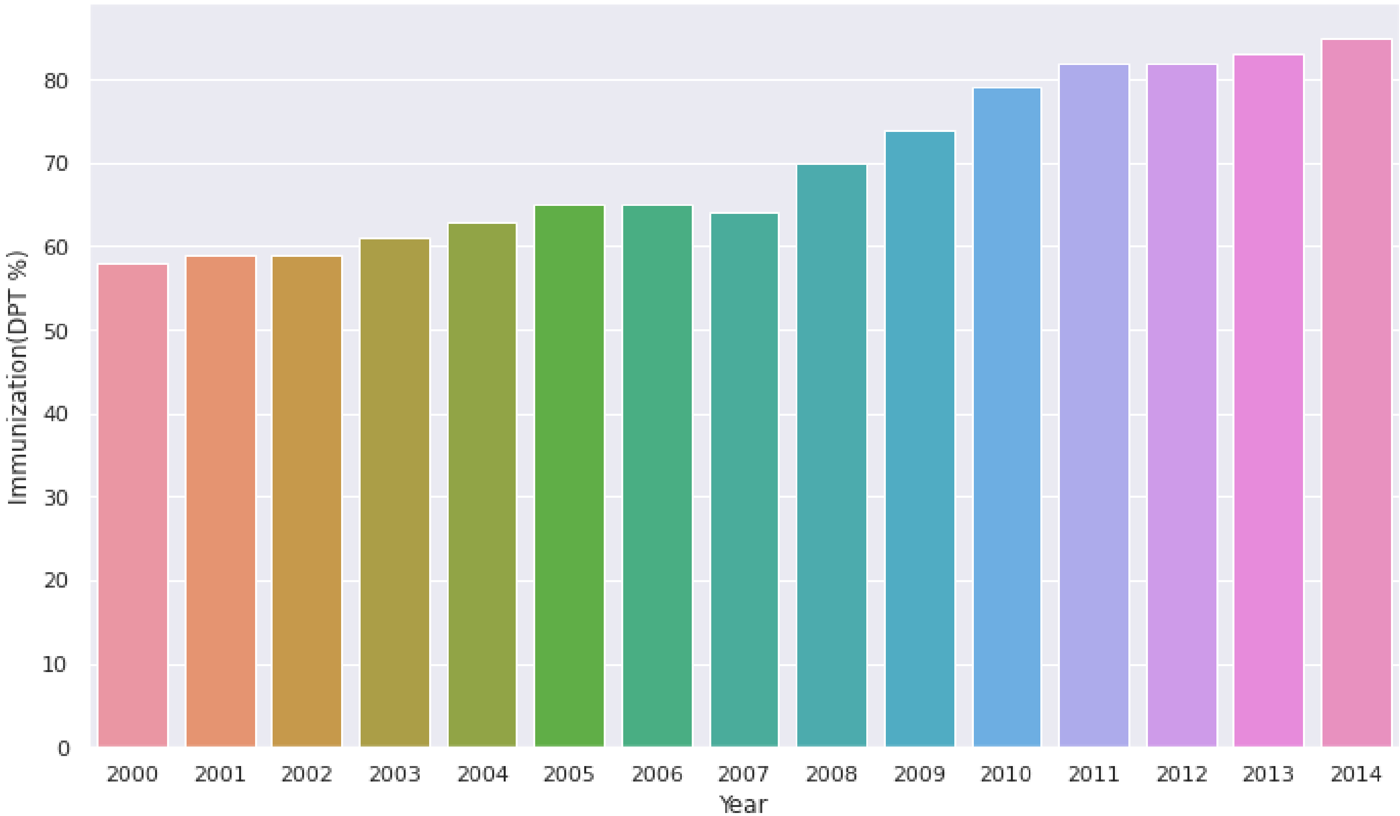


India : GDP/Capita($) vs Year

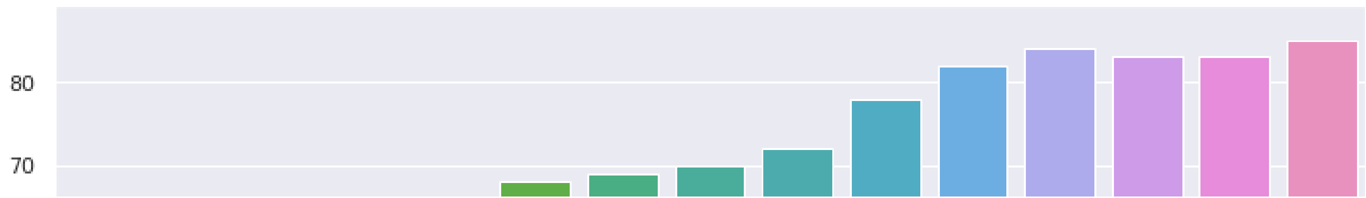


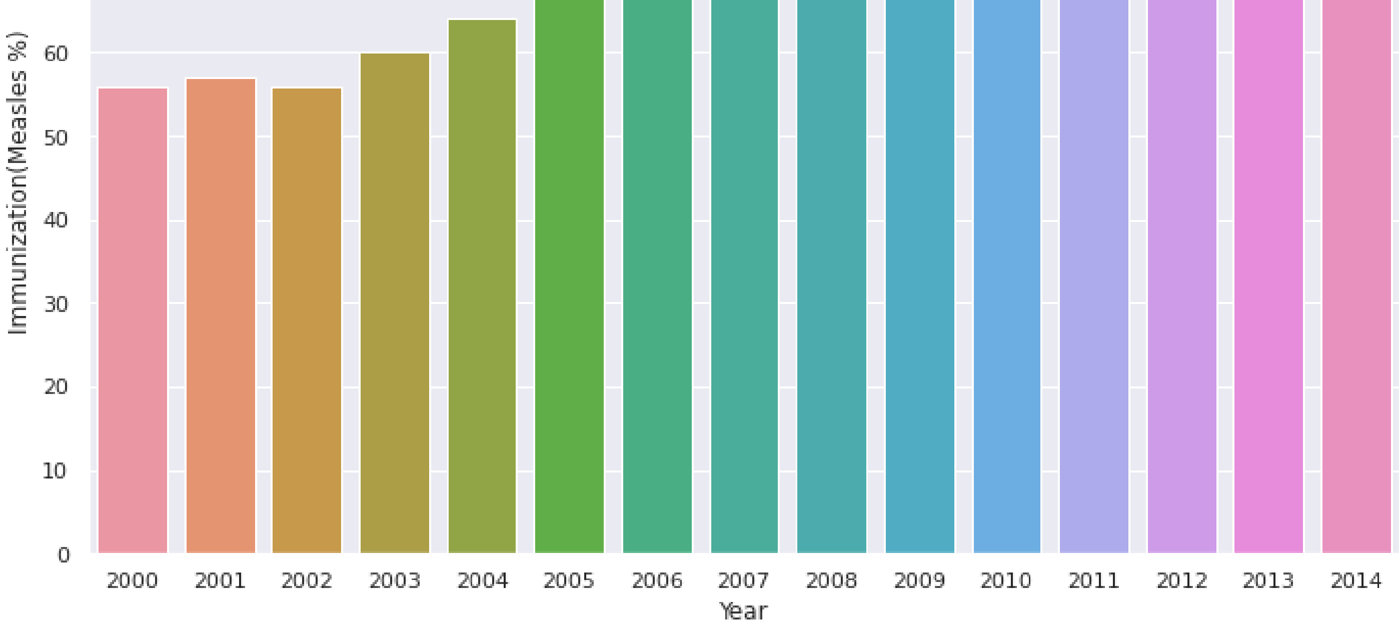


India : Immunization(DPT %) vs Year

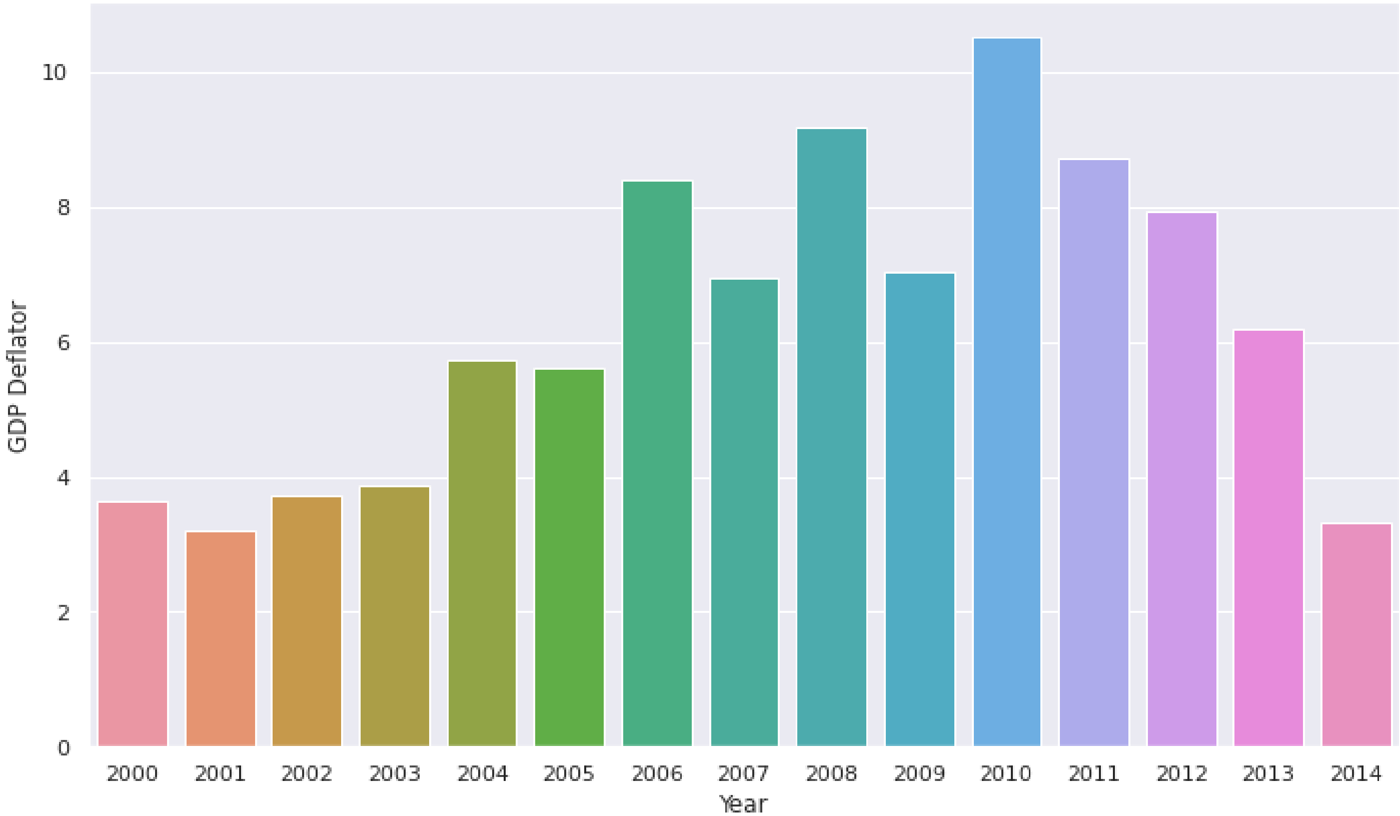


India : Immunization(Measles %) vs Year

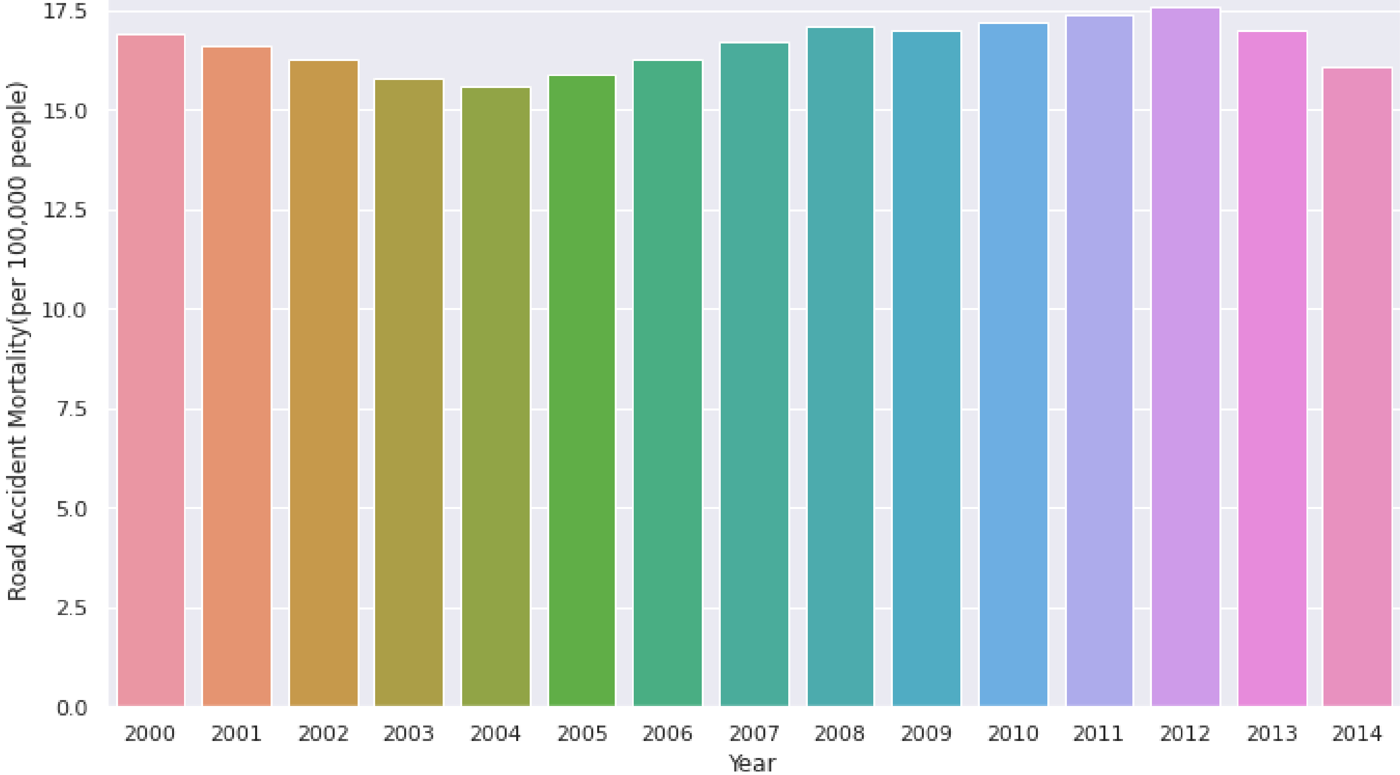




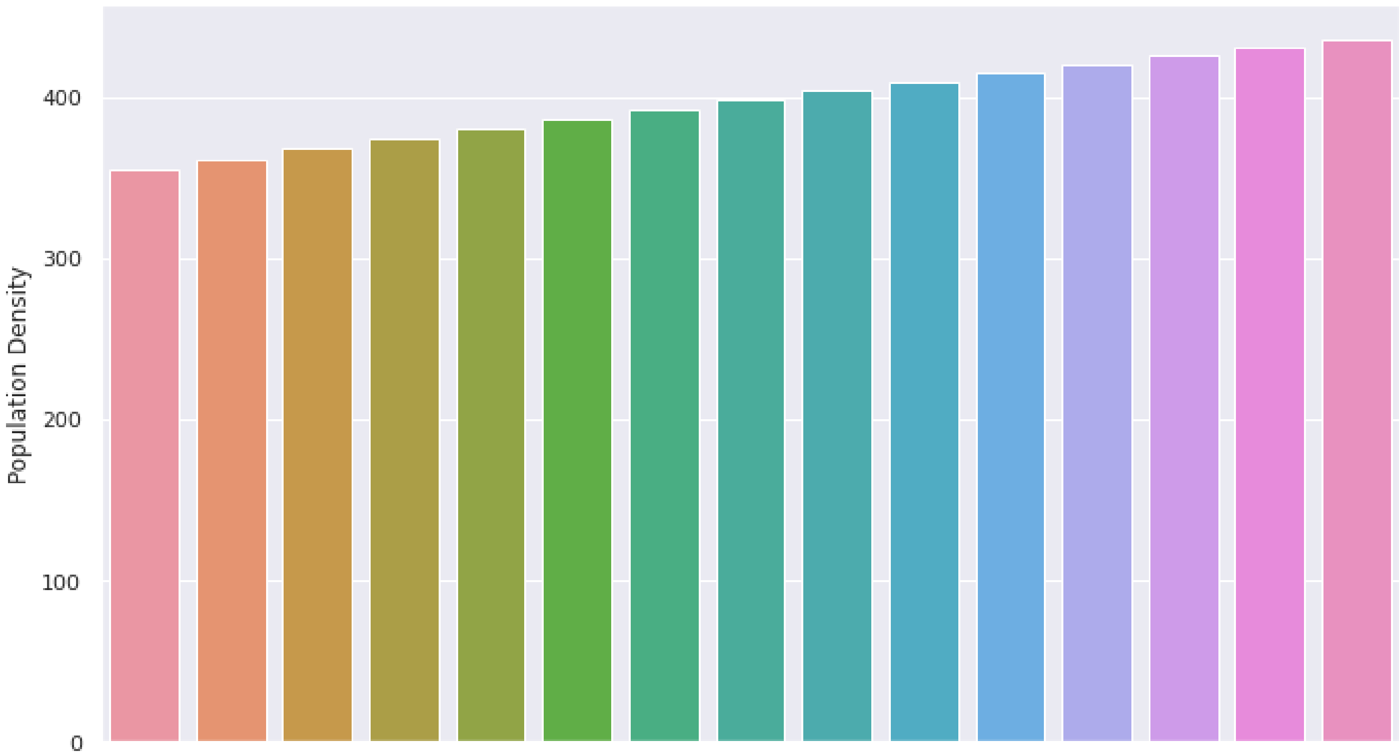
India : GDP Deflator vs Year



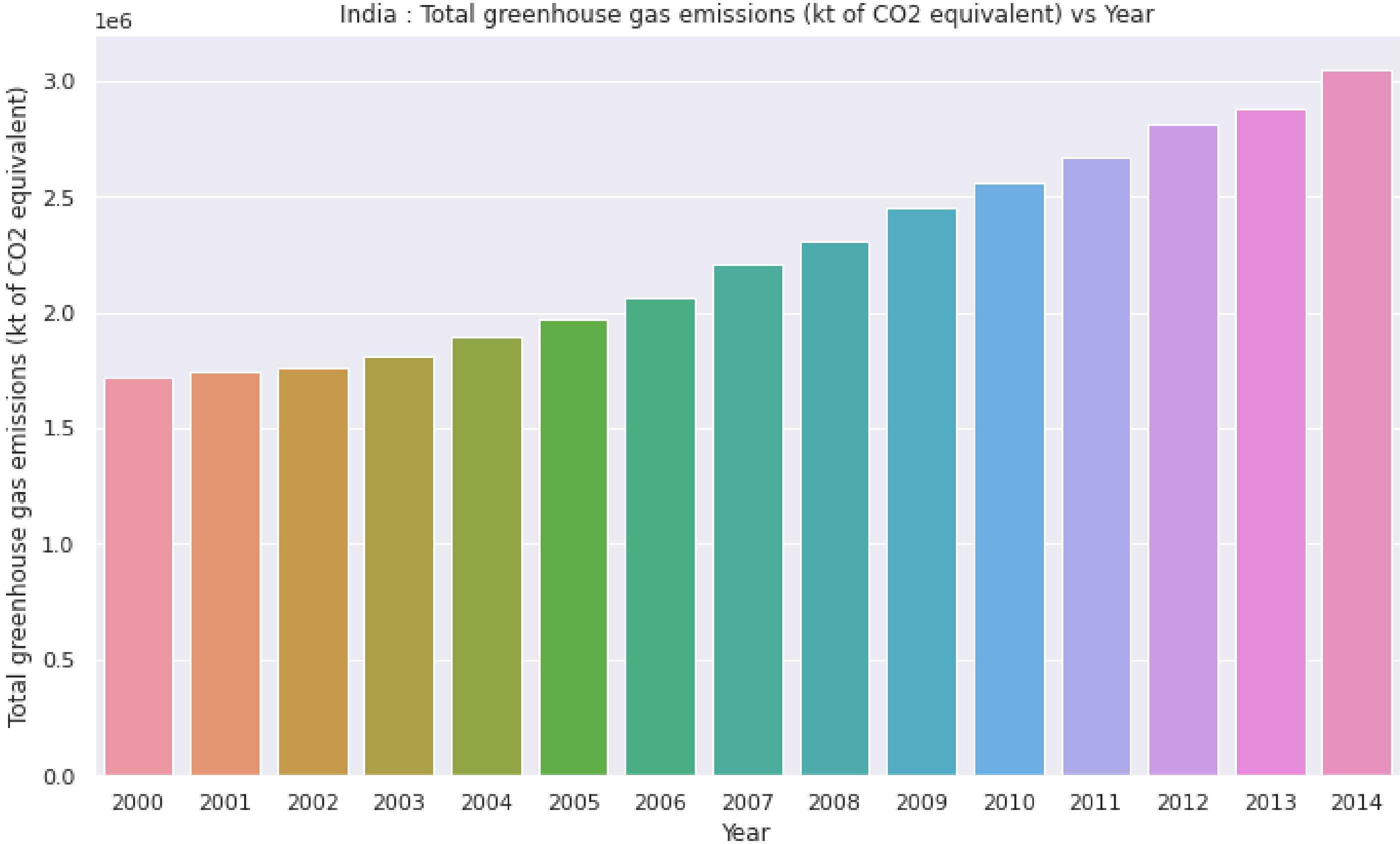
India : Road Accident Mortality(per 100,000 people) vs Year

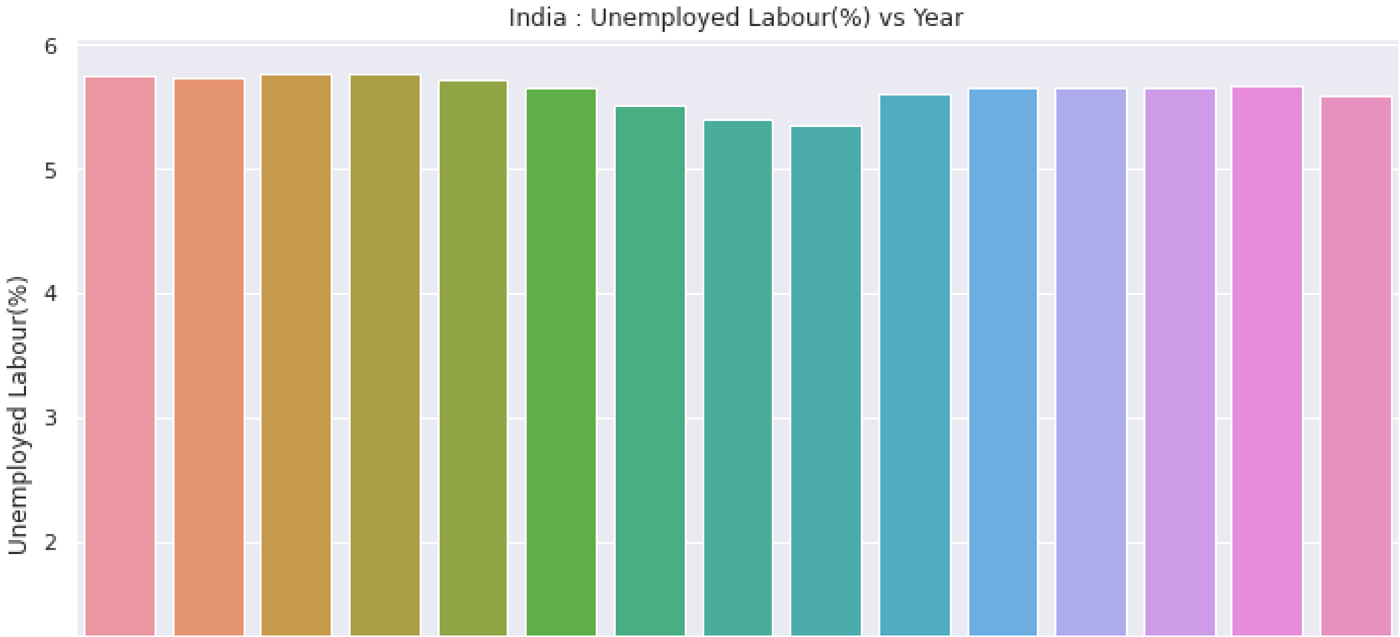


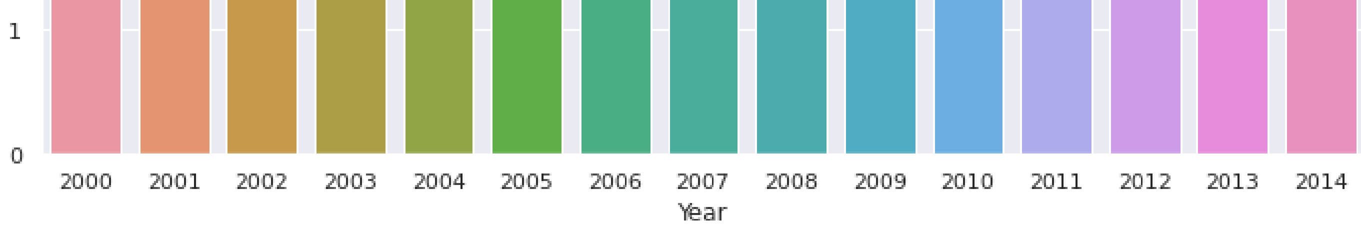
India : Population Density vs Year

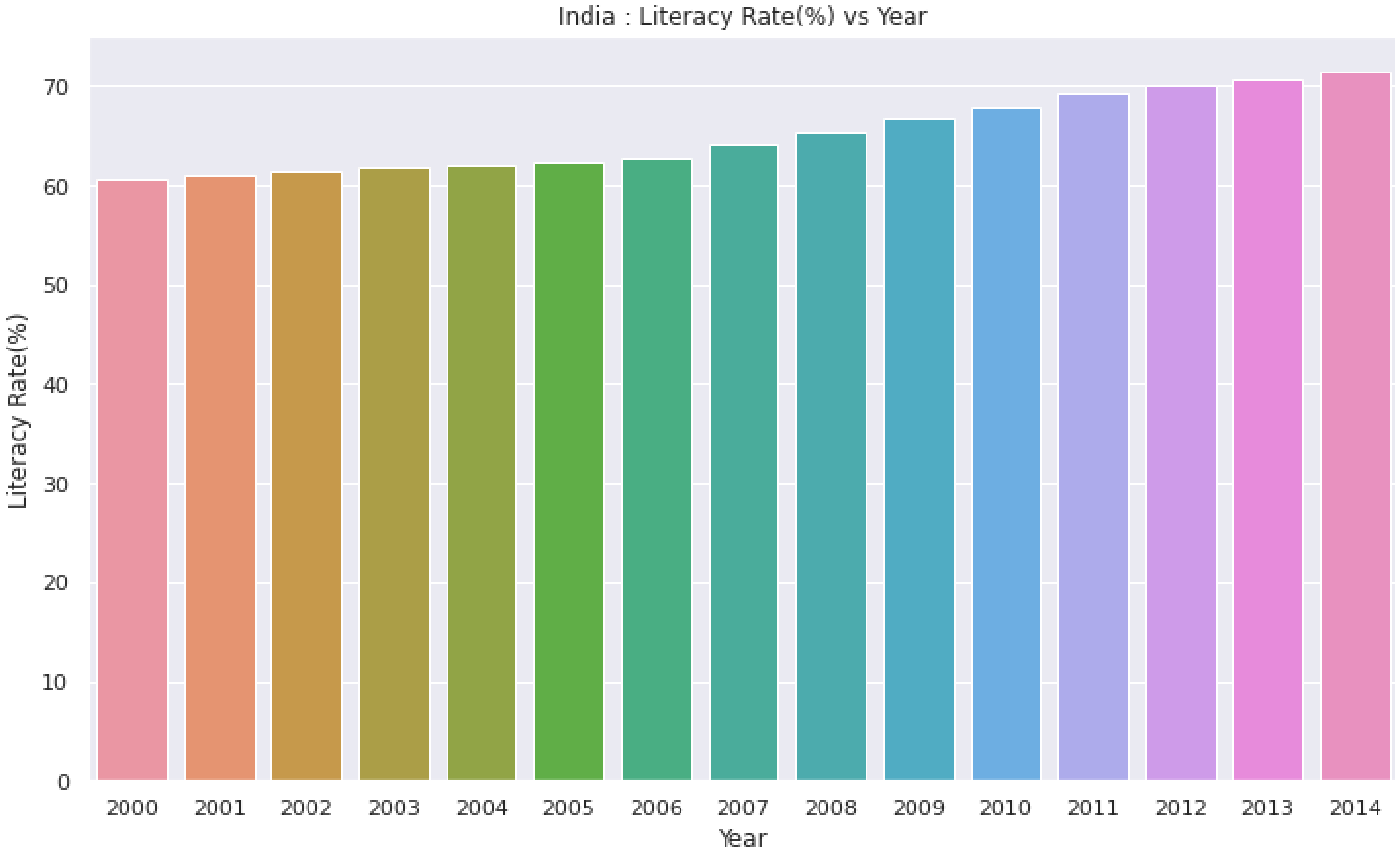


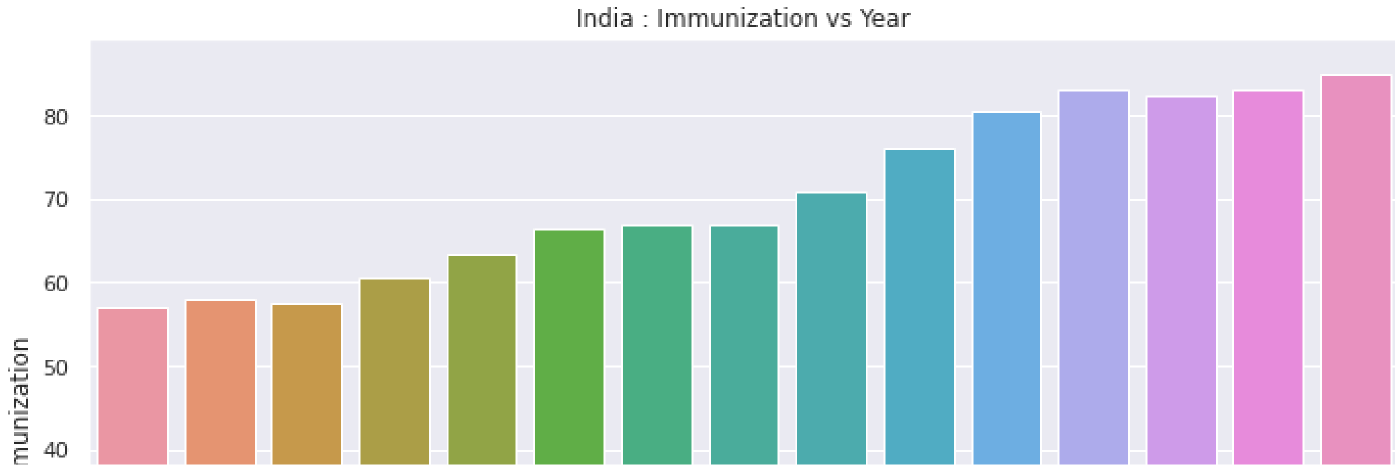

2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
Year

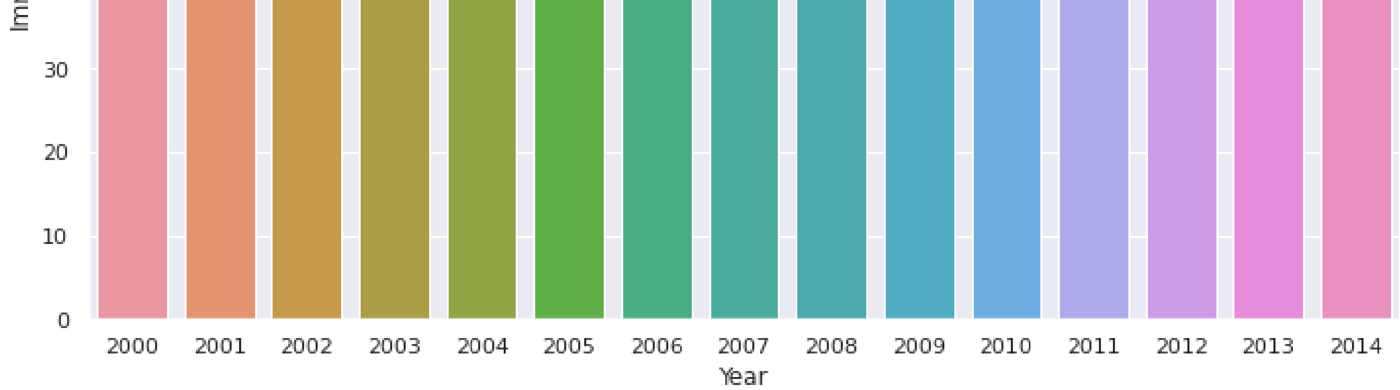

India : Total greenhouse gas emissions (kt of CO2 equivalent) vs Year
1e6
Total greenhouse gas emissions (kt of CO2 equivalent)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
Year


India : Unemployed Labour(%) vs Year
Unemployed Labour(%)
6
5
4
3
2

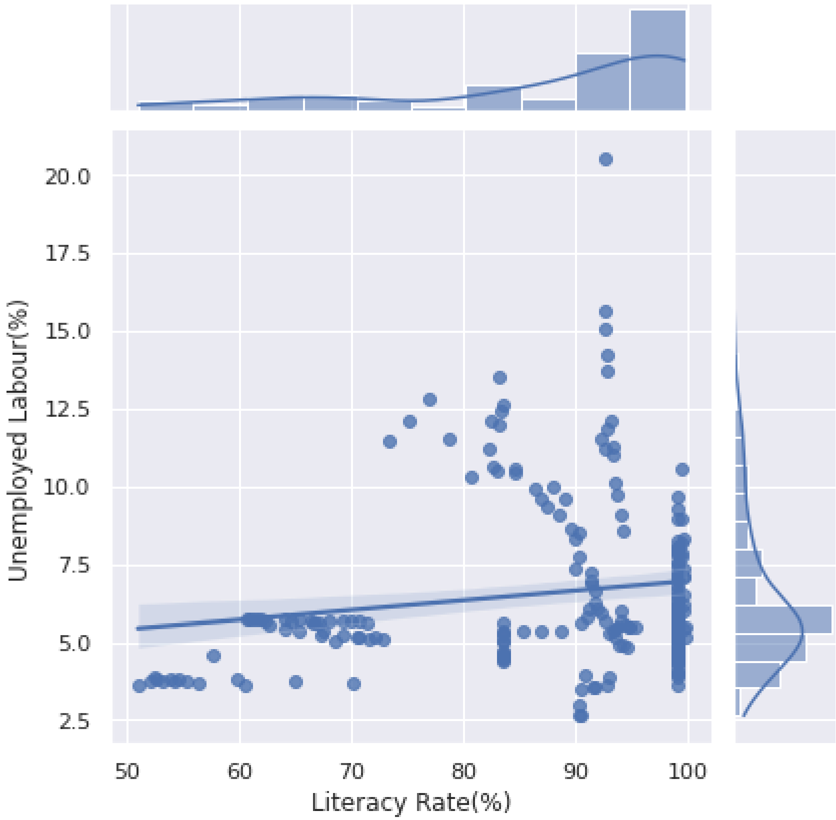

1
0
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Year


India : Literacy Rate(%) vs Year
Literacy Rate(%)
70
60
50
40
30
20
10
0
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Year


India : Immunization vs Year
munization
80
70
60
50
40

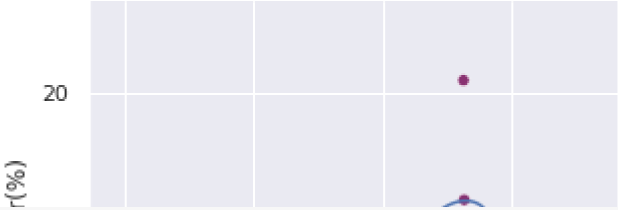



```
sns.jointplot(x='Literacy Rate(%)', y='Unemployed Labour(%)', data=df,kind='reg')
plt.show()
```

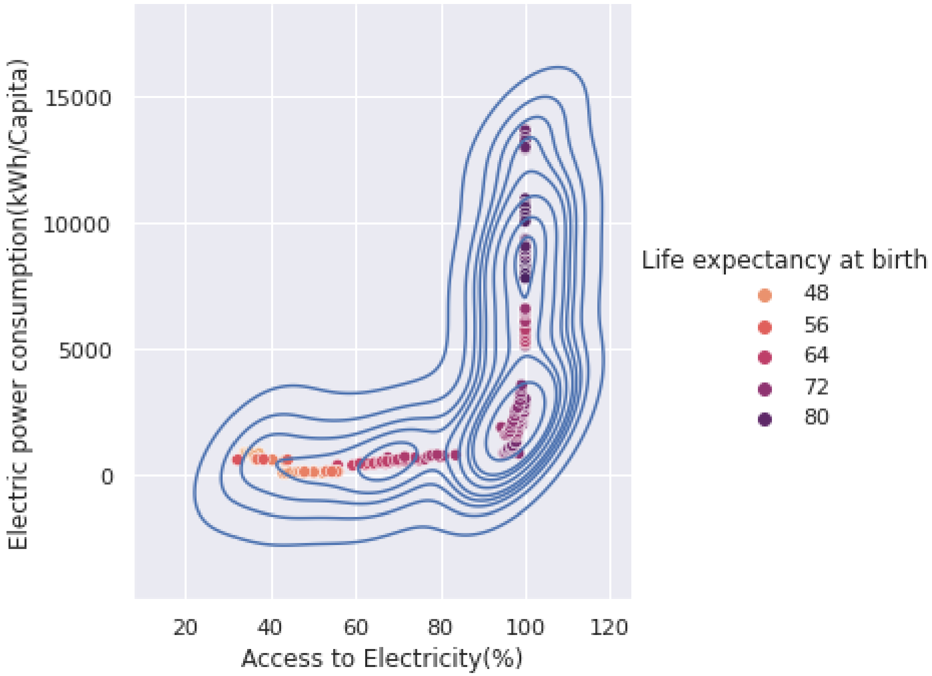


```
trivarplts=['Life expectancy at birth','Population Density']
```

```
for var in trivarplts:

  sns.relplot(x='Literacy Rate(%)', y='Unemployed Labour(%)', hue = var,palette="flare", data=df)
  sns.kdeplot(df['Literacy Rate(%)'], df['Unemployed Labour(%)'])
  plt.show();
```



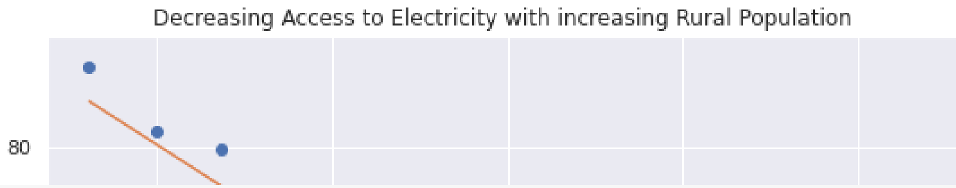
```
sns.relplot(x='Access to Electricity(%)', y='Electric power consumption(kWh/Capita)', hue='Life expectancy at birth',palette="flare", data=df)
sns.kdeplot(df['Access to Electricity(%)'], df['Electric power consumption(kWh/Capita)'])
plt.show()
```



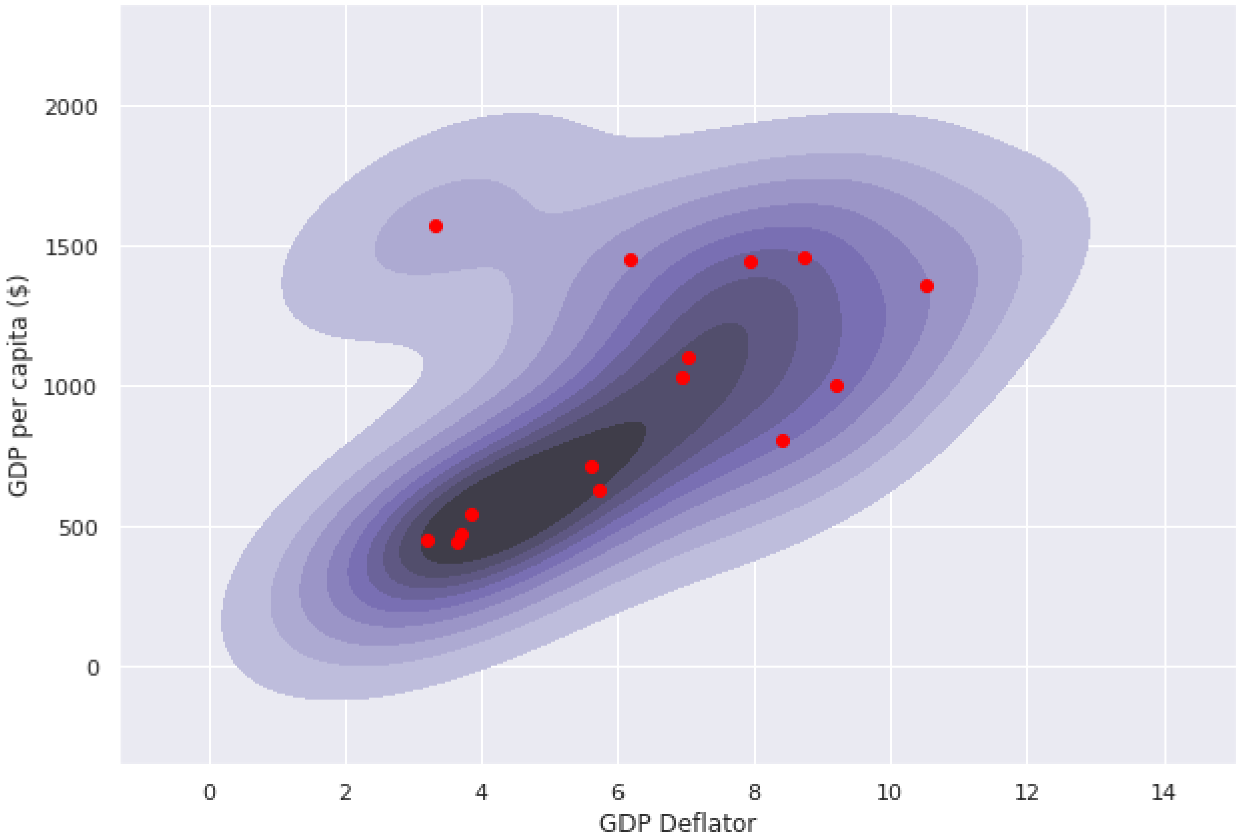
Scatter plots

```
plt.figure(figsize=(20,7))

plt.subplot(1,2,1)
plt.title('Decreasing Food Production Index with increasing Rural Population')
# FPI Vs Rural population
plt.plot(df_india['Rural Population(%)'], df_india['Food Production Index'], 'o')
#create scatter plot
```

```
m1, c1 = np.polyfit(df_india['Rural Population(%)'], df_india['Food Production Index'], 1)
#m1 = slope, c1=intercept (linear regression)

plt.plot(df_india['Rural Population(%)'], m1*df_india['Rural Population(%)'] + c1)
plt.xlabel('Rural Population(%)')
plt.ylabel('Food Production Index')

#Access to electricity vs Rural population
plt.subplot(1,2,2)
plt.title('Decreasing Access to Electricity with increasing Rural Population')
plt.plot(df_india['Rural Population(%)'], df_india['Access to Electricity(%)'], 'o')
#create scatter plot

m2, c2 = np.polyfit(df_india['Rural Population(%)'], df_india['Access to Electricity(%)'], 1)
#m2 = slope, c2=intercept (linear regression)

plt.plot(df_india['Rural Population(%)'], m2*df_india['Rural Population(%)'] + c2)
plt.xlabel('Rural Population(%)')
plt.ylabel('Access to Electricity(%)')
plt.show()
```



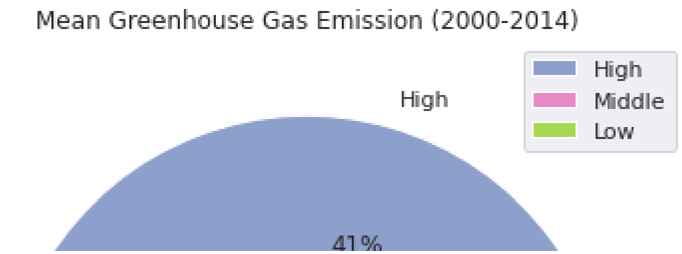

```
# GDP per capita vs GDP deflator
plt.figure(figsize=(10,7))
plt.plot(df_india['GDP Deflator'], df_india['GDP/Capita($)'], 'o', color = 'red')
#create scatter plot
sns.kdeplot(df_india['GDP Deflator'], df_india['GDP/Capita($)'], fill=True, cmap = 'Purples_d', shade = True)
plt.xlabel('GDP Deflator')
plt.ylabel('GDP per capita ($)')
plt.show()
```

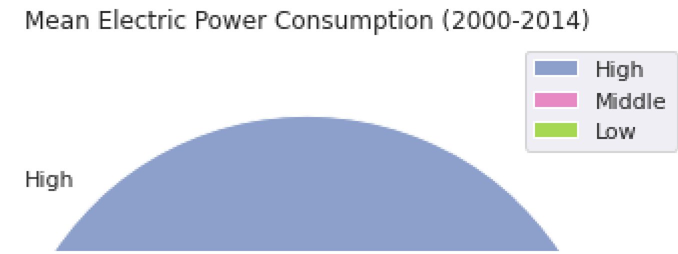


Pie charts

```
high_gh   = df.loc[df['Country']=='High']['Total greenhouse gas emissions (kt of CO2 equivalent)'].mean()
middle_gh = df.loc[df['Country']=='Middle']['Total greenhouse gas emissions (kt of CO2 equivalent)'].mean()
low_gh    = df.loc[df['Country']=='Low']['Total greenhouse gas emissions (kt of CO2 equivalent)'].mean()

high_pc   = df.loc[df['Country']=='High']['Electric power consumption(kWh/Capita)'].mean()
middle_pc   = df.loc[df['Country']=='Middle']['Electric power consumption(kWh/Capita)'].mean()
low_pc   = df.loc[df['Country']=='Low']['Electric power consumption(kWh/Capita)'].mean()
```

```
categories = ['High', 'Middle', 'Low']
data_gh    = [high_gh, middle_gh, low_gh]
data_pc    = [high_pc, middle_pc, low_pc]
colors = sns.color_palette('Set2')[2:5]

plt.figure(figsize = (20,7))
plt.subplot(1,2,1)
plt.pie(data_gh, labels = categories, colors = colors, autopct='%.0f%%')
plt.title('Mean Greenhouse Gas Emission (2000-2014)')
plt.legend(loc = 'upper right')
plt.subplot(1,2,2)
plt.pie(data_pc, labels = categories, colors = colors, autopct='%.1f%%')
plt.legend()
plt.title('Mean Electric Power Consumption (2000-2014)')
plt.show()
```





Modelling

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
```

```
df_train = pd.read_csv('/content/Data_training.csv')
df_train.head()
```

	Country	Year	Access to Electricity(%)	Food Production Index	Population Density	Unemployed Labour(%)	Road Accident Mortality(per 100,000 people)	Electric power consumption(kWh/Capita)	GDP Deflator	GDP/Capita($)	Li
0	Albania	2000	100.0	64.01	112.738212	16.58	14.3	1449.647413	5.643782	1126.683318	98.
1	Albania	2001	100.0	65.89	111.685146	16.54	14.5	1351.230796	3.813933	1281.659393	98.
2	Albania	2002	100.0	66.38	111.350730	16.61	14.5	1578.165919	3.644150	1425.124849	98.
3	Albania	2003	100.0	69.58	110.934891	16.61	14.7	1469.264539	5.197105	1846.118813	98.
4	Albania	2004	100.0	72.89	110.472226	16.52	14.7	1797.525487	3.156944	2373.579844	98.

```
df_train['Immunization'] = df_train[['Immunization(Measles %)', 'Immunization(DPT %)']].mean(axis=1)
df_train.head()
```

	Country	Year	Access to Electricity(%)	Food Production Index	Population Density	Unemployed Labour(%)	Road Accident Mortality(per 100,000 people)	Electric power consumption(kWh/Capita)	GDP Deflator	GDP/Capita($)	Li
0	Albania	2000	100.0	64.01	112.738212	16.58	14.3	1449.647413	5.643782	1126.683318	98.
1	Albania	2001	100.0	65.89	111.685146	16.54	14.5	1351.230796	3.813933	1281.659393	98.
2	Albania	2002	100.0	66.38	111.350730	16.61	14.5	1578.165919	3.644150	1425.124849	98.
3	Albania	2003	100.0	69.58	110.934891	16.61	14.7	1469.264539	5.197105	1846.118813	98.
4	Albania	2004	100.0	72.89	110.472226	16.52	14.7	1797.525487	3.156944	2373.579844	98.

Linear Regression

```
error_table = pd.DataFrame()
```

90:10

```
x = df_train.drop(['Country','Life Expectancy at Birth','Immunization(DPT %)','Immunization(Measles %)'], axis=1)
y = df_train['Life Expectancy at Birth']

xi_train, xi_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=1)

xi_train.reset_index(drop=True, inplace=True)
xi_test.reset_index(drop=True, inplace=True)

scaler = StandardScaler()
x_train = scaler.fit_transform(xi_train)
x_test = scaler.transform(xi_test)
```

```
model = LinearRegression()
model.fit(x_train, y_train)
LinearRegression()
y_pred = model.predict(x_test)

mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = np.mean(abs(y_test-y_pred))
r2 = r2_score(y_test,y_pred)
inter = model.intercept_
print('Coefficient array:',model.coef_)

Error = pd.Series({'Fraction': "90:10",'MSE': mse,'RMSE':rmse,'MAE':mae,'R2 Score':r2,'Intercept':inter})
error_table = error_table.append(Error, ignore_index=True)
error_table
```

```
Coefficient array: [ 0.18662236  4.90963572 -0.0089016   0.06666171 -0.47886285 -1.60147732
 -1.08948005  0.06983766  1.67366573 -1.02672176  0.28387706 -0.61229033
  2.11101613]
```

	Fraction	Intercept	MAE	MSE	R2 Score	RMSE
0	90:10	68.551351	2.246135	9.376017	0.836655	3.062028

```
df_test = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df_test.head()
```

	Actual	Predicted
309	74.119000	73.153400
285	76.366000	72.797789
919	53.475000	53.790576
120	55.391000	56.203757
585	67.557795	71.382511

80:20

```
x = df_train.drop(['Country','Life Expectancy at Birth','Immunization(DPT %)','Immunization(Measles %)'], axis=1)
y = df_train['Life Expectancy at Birth']

xi_train, xi_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)

xi_train.reset_index(drop=True, inplace=True)
xi_test.reset_index(drop=True, inplace=True)

scaler = StandardScaler()
x_train = scaler.fit_transform(xi_train)
x_test = scaler.transform(xi_test)

model = LinearRegression()
model.fit(x_train, y_train)
LinearRegression()
y_pred = model.predict(x_test)

mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = np.mean(abs(y_test-y_pred))
r2 = r2_score(y_test,y_pred)
inter = model.intercept_
print('Coefficient array:',model.coef_)

Error = pd.Series({'Fraction': "80:20",'MSE': mse,'RMSE':rmse,'MAE':mae,'R2 Score':r2,'Intercept':inter})
error_table = error_table.append(Error, ignore_index=True)
error_table
```

```
Coefficient array: [ 0.16522917  4.85521762  0.05358755  0.04291514 -0.45970863 -1.59112574
 -1.11857846  0.07000647  1.70261966 -1.01253029  0.31544103 -0.64540828
  2.08354396]
```

	Fraction	Intercept	MAE	MSE	R2 Score	RMSE
0	90:10	68.551351	2.246135	9.376017	0.836655	3.062028
1	80:20	68.626982	2.283324	10.048181	0.840939	3.169887

```
df_test = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df_test.head()
```

	Actual	Predicted
309	74.119000	73.192548
285	76.366000	72.915145
919	53.475000	53.890524
120	55.391000	56.325412
585	67.557795	71.329932

70:30

```
x = df_train.drop(['Country','Life Expectancy at Birth','Immunization(DPT %)','Immunization(Measles %)'], axis=1)
y = df_train['Life Expectancy at Birth']

xi_train, xi_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)

xi_train.reset_index(drop=True, inplace=True)
xi_test.reset_index(drop=True, inplace=True)

scaler = StandardScaler()
x_train = scaler.fit_transform(xi_train)
x_test = scaler.transform(xi_test)

model = LinearRegression()
model.fit(x_train, y_train)
LinearRegression()
y_pred = model.predict(x_test)

mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = np.mean(abs(y_test-y_pred))
r2 = r2_score(y_test,y_pred)
inter = model.intercept_
print('Coefficient array:',model.coef_)

Error = pd.Series({'Fraction': "70:30",'MSE': mse,'RMSE':rmse,'MAE':mae,'R2 Score':r2,'Intercept':inter})
error_table = error_table.append(Error, ignore_index=True)
error_table
```

```
Coefficient array: [ 0.06528092  5.06263763  0.09093502  0.01551206 -0.51806216 -1.5111888
 -1.10469162 -0.48499992  1.62833185 -1.04583803  0.27823032 -0.59640302
  2.09172721]
```

	Fraction	Intercept	MAE	MSE	R2 Score	RMSE
0	90:10	68.551351	2.246135	9.376017	0.836655	3.062028
1	80:20	68.626982	2.283324	10.048181	0.840939	3.169887
2	70:30	68.631378	2.472308	40.846597	0.333171	6.391134

```
df_test = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df_test.head()
```

	Actual	Predicted
309	74.119000	73.740602
285	76.366000	73.282725
919	53.475000	53.769409
120	55.391000	56.499418
585	67.557795	70.703169

60:40

```
x = df_train.drop(['Country','Life Expectancy at Birth','Immunization(DPT %)','Immunization(Measles %)'], axis=1)
y = df_train['Life Expectancy at Birth']

xi_train, xi_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=1)

xi_train.reset_index(drop=True, inplace=True)
xi_test.reset_index(drop=True, inplace=True)

scaler = StandardScaler()
x_train = scaler.fit_transform(xi_train)
x_test = scaler.transform(xi_test)
```

```
model = LinearRegression()
model.fit(x_train, y_train)
LinearRegression()
y_pred = model.predict(x_test)

mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = np.mean(abs(y_test-y_pred))
r2 = r2_score(y_test,y_pred)
inter = model.intercept_
print('Coefficient array:',model.coef_)

Error = pd.Series({'Fraction': "60:40",'MSE': mse,'RMSE':rmse,'MAE':mae,'R2 Score':r2,'Intercept':inter})
error_table = error_table.append(Error, ignore_index=True)
error_table
```

```
Coefficient array: [ 0.08574545  5.07670282  0.021653    0.06893907 -0.53266412 -1.49954248
 -1.08773543 -0.47555007  1.59659373 -1.00566461  0.25782602 -0.56204615
  2.13630037]
```

	Fraction	Intercept	MAE	MSE	R2 Score	RMSE
0	90:10	68.551351	2.246135	9.376017	0.836655	3.062028
1	80:20	68.626982	2.283324	10.048181	0.840939	3.169887
2	70:30	68.631378	2.472308	40.846597	0.333171	6.391134
3	60:40	68.525274	2.382400	29.873105	0.510407	5.465629

```
df_test = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df_test.head()
```

	Actual	Predicted
309	74.119000	73.709823
285	76.366000	73.160763
919	53.475000	53.790947
120	55.391000	56.395039
585	67.557795	70.803350

Decision Tree

```
error_table2 = pd.DataFrame()
```

90:10

```
x = df_train.drop(['Country','Life Expectancy at Birth','Immunization(DPT %)','Immunization(Measles %)'], axis=1)
y = df_train['Life Expectancy at Birth']

xi_train, xi_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=1)

xi_train.reset_index(drop=True, inplace=True)
xi_test.reset_index(drop=True, inplace=True)

scaler = StandardScaler()
x_train = scaler.fit_transform(xi_train)
x_test = scaler.transform(xi_test)

reg = DecisionTreeRegressor()
reg = reg.fit(x_train,y_train)
y_pred = reg.predict(x_test)

mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = np.mean(abs(y_test-y_pred))
r2 = r2_score(y_test,y_pred)

Error = pd.Series({'Fraction': "90:10",'MSE': mse,'RMSE':rmse,'MAE':mae,'R2 Score':r2})
error_table2 = error_table2.append(Error, ignore_index=True)
error_table2
```

	Fraction	MAE	MSE	R2 Score	RMSE
0	90:10	0.676857	1.700115	0.970381	1.303884

```
df_test = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

```
df_test.head()
```

	Actual	Predicted
309	74.119000	73.835000
285	76.366000	75.152000
919	53.475000	54.732000
120	55.391000	54.968000
585	67.557795	67.699156

80:20

```
x = df_train.drop(['Country','Life Expectancy at Birth','Immunization(DPT %)','Immunization(Measles %)'], axis=1)
y = df_train['Life Expectancy at Birth']

xi_train, xi_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)

xi_train.reset_index(drop=True, inplace=True)
xi_test.reset_index(drop=True, inplace=True)

scaler = StandardScaler()
x_train = scaler.fit_transform(xi_train)
x_test = scaler.transform(xi_test)

reg = DecisionTreeRegressor()
reg = reg.fit(x_train,y_train)
y_pred = reg.predict(x_test)

mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = np.mean(abs(y_test-y_pred))
r2 = r2_score(y_test,y_pred)

Error = pd.Series({'Fraction': "80:20",'MSE': mse,'RMSE':rmse,'MAE':mae,'R2 Score':r2})
error_table2 = error_table2.append(Error, ignore_index=True)
error_table2
```

	Fraction	MAE	MSE	R2 Score	RMSE
0	90:10	0.676857	1.700115	0.970381	1.303884
1	80:20	0.802965	3.045144	0.951796	1.745034

```
df_test = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df_test.head()
```

	Actual	Predicted
309	74.119000	73.835000
285	76.366000	76.634000
919	53.475000	54.732000
120	55.391000	50.041000
585	67.557795	67.699156

70:30

```
x = df_train.drop(['Country','Life Expectancy at Birth','Immunization(DPT %)','Immunization(Measles %)'], axis=1)
y = df_train['Life Expectancy at Birth']

xi_train, xi_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)

xi_train.reset_index(drop=True, inplace=True)
xi_test.reset_index(drop=True, inplace=True)

scaler = StandardScaler()
x_train = scaler.fit_transform(xi_train)
x_test = scaler.transform(xi_test)

reg = DecisionTreeRegressor()
reg = reg.fit(x_train,y_train)
y_pred = reg.predict(x_test)

mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

```
mae = np.mean(abs(y_test-y_pred))
r2 = r2_score(y_test,y_pred)

Error = pd.Series({'Fraction': "70:30",'MSE': mse,'RMSE':rmse,'MAE':mae,'R2 Score':r2})
error_table2 = error_table2.append(Error, ignore_index=True)
error_table2
```

	Fraction	MAE	MSE	R2 Score	RMSE
0	90:10	0.676857	1.700115	0.970381	1.303884
1	80:20	0.802965	3.045144	0.951796	1.745034
2	70:30	0.936379	3.531068	0.942355	1.879114

```
df_test = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df_test.head()
```

	Actual	Predicted
309	74.119000	74.409000
285	76.366000	75.882000
919	53.475000	56.887000
120	55.391000	48.946000
585	67.557795	68.157172

60:40

```
x = df_train.drop(['Country','Life Expectancy at Birth','Immunization(DPT %)','Immunization(Measles %)'], axis=1)
y = df_train['Life Expectancy at Birth']

xi_train, xi_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=1)

xi_train.reset_index(drop=True, inplace=True)
xi_test.reset_index(drop=True, inplace=True)
```

```
scaler = StandardScaler()
x_train = scaler.fit_transform(xi_train)
x_test = scaler.transform(xi_test)

reg = DecisionTreeRegressor()
reg = reg.fit(x_train,y_train)
y_pred = reg.predict(x_test)

mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = np.mean(abs(y_test-y_pred))
r2 = r2_score(y_test,y_pred)

Error = pd.Series({'Fraction': "60:40",'MSE': mse,'RMSE':rmse,'MAE':mae,'R2 Score':r2})
error_table2 = error_table2.append(Error, ignore_index=True)
error_table2
```

	Fraction	MAE	MSE	R2 Score	RMSE
0	90:10	0.676857	1.700115	0.970381	1.303884
1	80:20	0.802965	3.045144	0.951796	1.745034
2	70:30	0.936379	3.531068	0.942355	1.879114
3	60:40	0.967642	3.703468	0.939303	1.924440

```
df_test = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df_test.head()
```

	Actual	Predicted
309	74.119000	74.409000
285	76.366000	76.322000
919	53.475000	56.436000
120	55.391000	50.921000
585	67.557795	67.699156

Support Vector Machines for Regression

```
#Reading the data and doing one hot encoding for categorical variables 'Year' and dropping the unnecessary column 'Country'
#Reordering the columns to get expected variable as first columns
df = pd.read_csv("/content/Data_training.csv")

df['Immunization'] = (df['Immunization(DPT %)']+df['Immunization(Measles %)'])/2
df.drop(columns=['Immunization(DPT %)','Immunization(Measles %)'], inplace = True)

cols = df.columns.tolist()
cols = cols[11:12]+cols[:11]+cols[12:]
df = df[cols]
df = df.sample(frac=1, random_state = 23).reset_index(drop=True)

cols = df.columns.tolist()
cols = cols[2:]
y = df['Life Expectancy at Birth']
y = pd.DataFrame(y)
X = df[cols]
```

Initialization for grid search over the hyperparameters for model

```
param_grid = [
              {'kernel' : ['linear'], 'C':[0.001, 0.1, 10, 100] },
              {'kernel' : ['rbf','sigmoid'], 'gamma' : [0.001,0.01,0.1, 10, 100,500,1000], 'C' : [0.001, 0.1, 10, 100] },
              {'kernel' : ['poly'], 'degree' : [2,3,4], 'C' : [0.001, 0.1, 10, 100] }
]

K = 5 #5 Fold cross validation

metric_grid = {'MSE' : make_scorer(mean_squared_error, greater_is_better = False), 'R2' : make_scorer(r2_score), 'MAE' : make_scorer(mean_absol

svm_reg = SVR()
```

60:40 Split into Training and Test

```
#Split Ratio 60, 40

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.4)

train_X.reset_index(drop = True, inplace = True)
test_X.reset_index(drop = True, inplace = True)
train_y.reset_index(drop = True, inplace = True)
test_y.reset_index(drop = True, inplace = True)

#Applying Standard Scaling of test data to both test and train
scaler = StandardScaler()
train_X_scaled = scaler.fit_transform(train_X)
test_X_scaled = scaler.transform(test_X)

grid_search = GridSearchCV(svm_reg, param_grid, scoring = metric_grid, refit = 'R2', cv = K)
grid_search.fit(train_X_scaled, train_y.values.ravel())
results = pd.DataFrame(grid_search.cv_results_)

print("At split of 60 : 40 these are the  hyperparameters :\n", grid_search.best_params_, '\n')
```

```
At split of 60 : 40 these are the  hyperparameters :
 {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
```

Using the predicted best Hyperparameters from 60:40 split

```
svm_reg = SVR(kernel = 'rbf', C = 100, gamma = 0.1)
svm_reg.fit(train_X_scaled, train_y.values.ravel())
test_pred_y = svm_reg.predict(test_X_scaled)

print("Using the best hyperparameters from 60:40 split : ")
print('Mean Squared Error on Test Data :',mean_squared_error(test_y,test_pred_y))
print('Mean Absolute Error on Test Data :',mean_absolute_error(test_y,test_pred_y))
print('R2 Score on Test Data :',r2_score(test_y,test_pred_y))
```

```
Using the best hyperparameters from 60:40 split :
Mean Squared Error on Test Data : 1.068167467093339
```

```
Mean Absolute Error on Test Data : 0.6193844433164097
R2 Score on Test Data : 0.982922326682142
```

70:30 Split into Training and Test

```
#Split Ratio 70, 30

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.3)

train_X.reset_index(drop = True, inplace = True)
test_X.reset_index(drop = True, inplace = True)
train_y.reset_index(drop = True, inplace = True)
test_y.reset_index(drop = True, inplace = True)

#Applying Standard Scaling of test data to both test and train
scaler = StandardScaler()
train_X_scaled = scaler.fit_transform(train_X)
test_X_scaled = scaler.transform(test_X)

grid_search = GridSearchCV(svm_reg, param_grid, scoring = metric_grid, refit = 'R2', cv = K)
grid_search.fit(train_X_scaled, train_y.values.ravel())
results = pd.DataFrame(grid_search.cv_results_)

print("At split of 70 : 30 these are the  hyperparameters :\n", grid_search.best_params_, '\n')
```

```
At split of 70 : 30 these are the  hyperparameters :
 {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
```

Using the predicted best Hyperparameters from 70:30 split

```
svm_reg = SVR(kernel = 'rbf', C = 100, gamma = 0.1)
svm_reg.fit(train_X_scaled, train_y.values.ravel())
test_pred_y = svm_reg.predict(test_X_scaled)
```

```
print("Using the best hyperparameters from 70:30 split : ")
print('Mean Squared Error on Test Data :',mean_squared_error(test_y,test_pred_y))
print('Mean Absolute Error on Test Data :',mean_absolute_error(test_y,test_pred_y))
print('R2 Score on Test Data :',r2_score(test_y,test_pred_y))
```

```
Using the best hyperparameters from 70:30 split :
Mean Squared Error on Test Data : 0.7362402627975829
Mean Absolute Error on Test Data : 0.5190577765286374
R2 Score on Test Data : 0.9882073037099876
```

80:20 Split into Training and Test

```
#Split Ratio 80, 20

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.2)

train_X.reset_index(drop = True, inplace = True)
test_X.reset_index(drop = True, inplace = True)
train_y.reset_index(drop = True, inplace = True)
test_y.reset_index(drop = True, inplace = True)

#Applying Standard Scaling of test data to both test and train
scaler = StandardScaler()
train_X_scaled = scaler.fit_transform(train_X)
test_X_scaled = scaler.transform(test_X)

grid_search = GridSearchCV(svm_reg, param_grid, scoring = metric_grid, refit = 'R2', cv = K)
grid_search.fit(train_X_scaled, train_y.values.ravel())
results = pd.DataFrame(grid_search.cv_results_)

print("At split of 80 : 20 these are the  hyperparameters :\n", grid_search.best_params_, '\n')
```

```
At split of 80 : 20 these are the  hyperparameters :
 {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
```

Using the predicted best Hyperparameters from 80:20 split

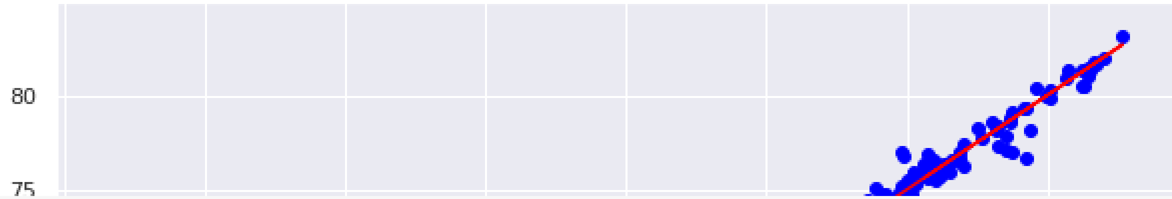
```
svm_reg = SVR(kernel = 'rbf', C = 100, gamma = 0.1)
svm_reg.fit(train_X_scaled, train_y.values.ravel())
test_pred_y = svm_reg.predict(test_X_scaled)

print("Using the best hyperparameters from 80:20 split : ")
print('Mean Squared Error on Test Data :',mean_squared_error(test_y,test_pred_y))
print('Mean Absolute Error on Test Data :',mean_absolute_error(test_y,test_pred_y))
print('R2 Score on Test Data :',r2_score(test_y,test_pred_y))
```

```
Using the best hyperparameters from 80:20 split :
Mean Squared Error on Test Data : 1.1864242620071876
Mean Absolute Error on Test Data : 0.5895846707395741
R2 Score on Test Data : 0.9799819090336667
```

Graph for Actual vs Predicted Variables with 45 degree reference line

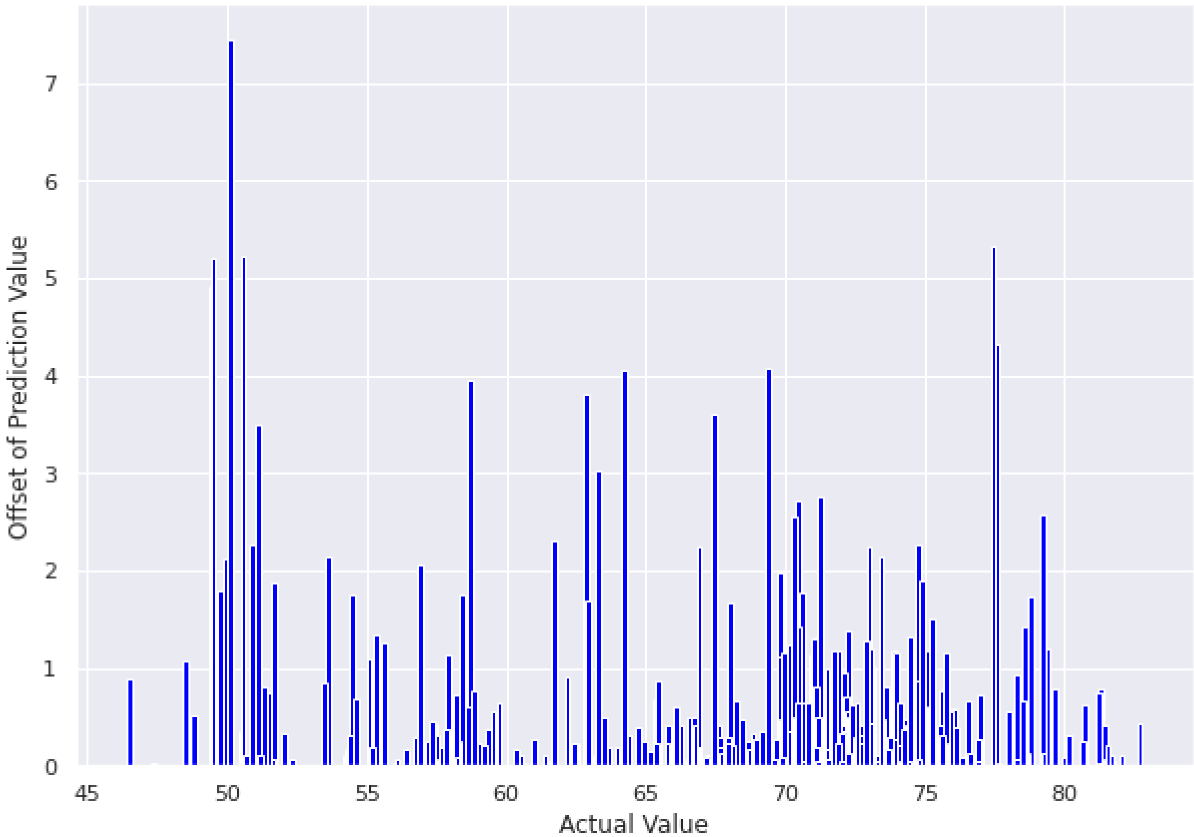
```
plt.figure(figsize = (10,7))
plt.scatter(x = test_y.values.ravel(), y = test_pred_y,color = 'blue')
plt.plot(test_y.values.ravel(), test_y.values.ravel(), color = 'red')
plt.xlabel('Actual Value')
plt.ylabel('Predicted Value')
```

Text(0, 0.5, 'Predicted Value')




```
diff = abs(test_y.values.ravel()-test_pred_y)

plt.figure(figsize = (10, 7))
plt.bar(test_y.values.ravel(), diff, color ='blue', width = 0.2)
plt.xlabel('Actual Value')
plt.ylabel('Offset of Prediction Value')
```

Text(0, 0.5, 'Offset of Prediction Value')



90:10 Split into Training and Test

```
#Split Ratio 90, 10

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.1)

train_X.reset_index(drop = True, inplace = True)
test_X.reset_index(drop = True, inplace = True)
train_y.reset_index(drop = True, inplace = True)
test_y.reset_index(drop = True, inplace = True)

#Applying Standard Scaling of test data to both test and train
scaler = StandardScaler()
train_X_scaled = scaler.fit_transform(train_X)
test_X_scaled = scaler.transform(test_X)

grid_search = GridSearchCV(svm_reg, param_grid, scoring = metric_grid, refit = 'R2', cv = K)
grid_search.fit(train_X_scaled, train_y.values.ravel())
results = pd.DataFrame(grid_search.cv_results_)

print("At split of 90 : 10 these are the  hyperparameters :\n", grid_search.best_params_, '\n')
```

```
At split of 90 : 10 these are the  hyperparameters :
 {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
```

Using the predicted best Hyperparameters from 80:20 split

```
svm_reg = SVR(kernel = 'rbf', C = 100, gamma = 0.1)
svm_reg.fit(train_X_scaled, train_y.values.ravel())
test_pred_y = svm_reg.predict(test_X_scaled)

print("Using the best hyperparameters from 90:10 split : ")
print('Mean Squared Error on Test Data :',mean_squared_error(test_y,test_pred_y))
print('Mean Absolute Error on Test Data :',mean_absolute_error(test_y,test_pred_y))
print('R2 Score on Test Data :',r2_score(test_y,test_pred_y))
```

```
Using the best hyperparameters from 90:10 split :
Mean Squared Error on Test Data : 4.019269354282607
```

```
Mean Absolute Error on Test Data : 0.7792402449800864
R2 Score on Test Data : 0.939761772236713
```

Neural Networks

```
import math
import statsmodels.api as sm
import statsmodels.formula.api as smf
import tensorflow
tensorflow.random.set_seed(1)
from tensorflow.python.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.wrappers.scikit_learn import KerasRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
df_nn = pd.read_csv('/content/Data_training.csv')
```

```
df_nn = df_nn.drop(['Country'], axis = 1)
```

```
df_nn['Immunization'] = df_nn[['Immunization(Measles %)','Immunization(DPT %)']].mean(axis = 1)
```

```
df_nn = df_nn.drop(columns = ['Immunization(Measles %)','Immunization(DPT %)'])
```

```
# Re arranging columns
cols_at_end = ['Life Expectancy at Birth']
df_nn = df_nn[[c for c in df_nn if c not in cols_at_end]
        + [c for c in cols_at_end if c in df_nn]]
```

```
x_data = df_nn.iloc[:, 0:13]
```

```
y_data = df_nn.iloc[:, 13]

from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience = 10)
```

70:30 split

```
X_train, X_val, y_train, y_val = train_test_split(x_data, y_data, random_state=1)
```

```
y_val = y_val.values.reshape(-1,1)
y_train = y_train.values.reshape(-1,1)
```

```
scaler = StandardScaler()
xtrain_scale=scaler.fit_transform(X_train)
xval_scale = scaler.fit_transform(X_val)
ytrain_scale = scaler.fit_transform(y_train)
yval_scale=scaler.fit_transform(y_val)
```

```
model = Sequential()
model.add(Dense(13, input_dim=13, kernel_initializer='normal', activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

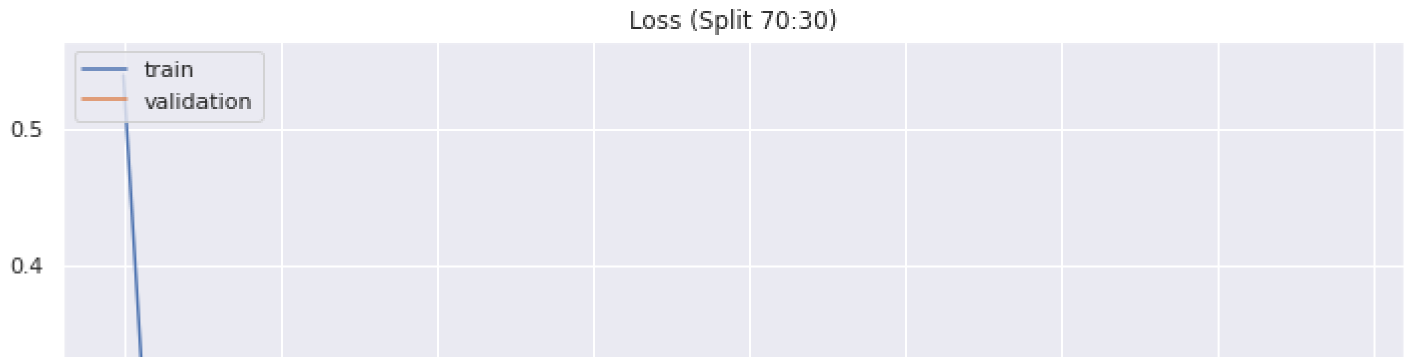
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 13)                182
_________________________________________________________________
dense_1 (Dense)              (None, 200)               2800
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 201
=================================================================
Total params: 3,183
Trainable params: 3,183
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history=model.fit(xtrain_scale, ytrain_scale, epochs=40, batch_size=20, verbose=1, validation_split=0.3, callbacks = [es])
predictions = model.predict(xval_scale)
```

```
Epoch 13/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0662 - mse: 0.0662 - mae: 0.1849 - val_loss: 0.0904 - val_mse: 0.0904 -
Epoch 14/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0630 - mse: 0.0630 - mae: 0.1804 - val_loss: 0.0951 - val_mse: 0.0951 -
Epoch 15/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0660 - mse: 0.0660 - mae: 0.1831 - val_loss: 0.0867 - val_mse: 0.0867 -
Epoch 16/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0641 - mse: 0.0641 - mae: 0.1828 - val_loss: 0.0903 - val_mse: 0.0903 -
Epoch 17/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0574 - mse: 0.0574 - mae: 0.1701 - val_loss: 0.0887 - val_mse: 0.0887 -
Epoch 18/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0561 - mse: 0.0561 - mae: 0.1691 - val_loss: 0.0910 - val_mse: 0.0910 -
Epoch 19/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0548 - mse: 0.0548 - mae: 0.1685 - val_loss: 0.0850 - val_mse: 0.0850 -
Epoch 20/40

53/53 [==============================] - 0s 5ms/step - loss: 0.0552 - mse: 0.0552 - mae: 0.1681 - val_loss: 0.0793 - val_mse: 0.0793 -
Epoch 21/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0531 - mse: 0.0531 - mae: 0.1636 - val_loss: 0.0846 - val_mse: 0.0846 -
Epoch 22/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0499 - mse: 0.0499 - mae: 0.1576 - val_loss: 0.0821 - val_mse: 0.0821 -
Epoch 23/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0475 - mse: 0.0475 - mae: 0.1551 - val_loss: 0.0819 - val_mse: 0.0819 -
Epoch 24/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0468 - mse: 0.0468 - mae: 0.1529 - val_loss: 0.0749 - val_mse: 0.0749 -
Epoch 25/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0465 - mse: 0.0465 - mae: 0.1534 - val_loss: 0.0774 - val_mse: 0.0774 -
Epoch 26/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0440 - mse: 0.0440 - mae: 0.1483 - val_loss: 0.0759 - val_mse: 0.0759 -
Epoch 27/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0417 - mse: 0.0417 - mae: 0.1450 - val_loss: 0.0756 - val_mse: 0.0756 -
Epoch 28/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0413 - mse: 0.0413 - mae: 0.1431 - val_loss: 0.0725 - val_mse: 0.0725 -
Epoch 29/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0404 - mse: 0.0404 - mae: 0.1422 - val_loss: 0.0712 - val_mse: 0.0712 -
Epoch 30/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0393 - mse: 0.0393 - mae: 0.1392 - val_loss: 0.0713 - val_mse: 0.0713 -
Epoch 31/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0389 - mse: 0.0389 - mae: 0.1393 - val_loss: 0.0689 - val_mse: 0.0689 -
Epoch 32/40
```

```
53/53 [==============================] - 0s 5ms/step - loss: 0.0353 - mse: 0.0353 - mae: 0.1336 - val_loss: 0.0692 - val_mse: 0.0692 -
Epoch 33/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0359 - mse: 0.0359 - mae: 0.1342 - val_loss: 0.0686 - val_mse: 0.0686 -
Epoch 34/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0345 - mse: 0.0345 - mae: 0.1314 - val_loss: 0.0707 - val_mse: 0.0707 -
Epoch 35/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0331 - mse: 0.0331 - mae: 0.1290 - val_loss: 0.0652 - val_mse: 0.0652 -
Epoch 36/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0337 - mse: 0.0337 - mae: 0.1309 - val_loss: 0.0640 - val_mse: 0.0640 -
Epoch 37/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0324 - mse: 0.0324 - mae: 0.1252 - val_loss: 0.0705 - val_mse: 0.0705 -
Epoch 38/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0310 - mse: 0.0310 - mae: 0.1228 - val_loss: 0.0622 - val_mse: 0.0622 -
Epoch 39/40
53/53 [==============================] - 0s 5ms/step - loss: 0.0304 - mse: 0.0304 - mae: 0.1238 - val_loss: 0.0621 - val_mse: 0.0621 -
Epoch 40/40
53/53 [==============================] - 0s 4ms/step - loss: 0.0302 - mse: 0.0302 - mae: 0.1211 - val_loss: 0.0677 - val_mse: 0.0677 -
```

```python
# Plotting vaidation and training loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss (Split 70:30)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



80:20

```
model1 = Sequential()
model1.add(Dense(13, input_dim=13, kernel_initializer='normal', activation='relu'))
model1.add(Dense(200, activation='relu'))
model1.add(Dense(1, activation='linear'))
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 13)                182
_________________________________________________________________
dense_4 (Dense)              (None, 200)               2800
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 201
=================================================================
Total params: 3,183
Trainable params: 3,183
Non-trainable params: 0
_________________________________________________________________
```

```
model1.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history1=model1.fit(xtrain_scale, ytrain_scale, epochs=50, batch_size=25, verbose=1, validation_split=0.2, callbacks = [es])
predictions = model1.predict(xval_scale)
```

```
Epoch 23/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0487 - mse: 0.0487 - mae: 0.1592 - val_loss: 0.0602 - val_mse: 0.0602 -
Epoch 24/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0470 - mse: 0.0470 - mae: 0.1571 - val_loss: 0.0627 - val_mse: 0.0627 -
```

```
Epoch 25/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0462 - mse: 0.0462 - mae: 0.1558 - val_loss: 0.0583 - val_mse: 0.0583 -
Epoch 26/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0443 - mse: 0.0443 - mae: 0.1521 - val_loss: 0.0557 - val_mse: 0.0557 -
Epoch 27/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0433 - mse: 0.0433 - mae: 0.1499 - val_loss: 0.0549 - val_mse: 0.0549 -
Epoch 28/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0423 - mse: 0.0423 - mae: 0.1502 - val_loss: 0.0638 - val_mse: 0.0638 -
Epoch 29/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0429 - mse: 0.0429 - mae: 0.1480 - val_loss: 0.0570 - val_mse: 0.0570 -
Epoch 30/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0405 - mse: 0.0405 - mae: 0.1477 - val_loss: 0.0523 - val_mse: 0.0523 -
Epoch 31/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0402 - mse: 0.0402 - mae: 0.1449 - val_loss: 0.0568 - val_mse: 0.0568 -
Epoch 32/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0395 - mse: 0.0395 - mae: 0.1445 - val_loss: 0.0531 - val_mse: 0.0531 -
Epoch 33/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0390 - mse: 0.0390 - mae: 0.1432 - val_loss: 0.0527 - val_mse: 0.0527 -
Epoch 34/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0365 - mse: 0.0365 - mae: 0.1383 - val_loss: 0.0506 - val_mse: 0.0506 -
Epoch 35/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0373 - mse: 0.0373 - mae: 0.1401 - val_loss: 0.0520 - val_mse: 0.0520 -
Epoch 36/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0375 - mse: 0.0375 - mae: 0.1412 - val_loss: 0.0527 - val_mse: 0.0527 -
Epoch 37/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0355 - mse: 0.0355 - mae: 0.1383 - val_loss: 0.0520 - val_mse: 0.0520 -
Epoch 38/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0361 - mse: 0.0361 - mae: 0.1378 - val_loss: 0.0542 - val_mse: 0.0542 -
Epoch 39/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0350 - mse: 0.0350 - mae: 0.1354 - val_loss: 0.0539 - val_mse: 0.0539 -
Epoch 40/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0330 - mse: 0.0330 - mae: 0.1326 - val_loss: 0.0478 - val_mse: 0.0478 -
Epoch 41/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0332 - mse: 0.0332 - mae: 0.1336 - val_loss: 0.0482 - val_mse: 0.0482 -
Epoch 42/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0316 - mse: 0.0316 - mae: 0.1301 - val_loss: 0.0502 - val_mse: 0.0502 -
Epoch 43/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0311 - mse: 0.0311 - mae: 0.1286 - val_loss: 0.0472 - val_mse: 0.0472 -
Epoch 44/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0312 - mse: 0.0312 - mae: 0.1287 - val_loss: 0.0481 - val_mse: 0.0481 -
Epoch 45/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0322 - mse: 0.0322 - mae: 0.1309 - val_loss: 0.0472 - val_mse: 0.0472 -
Epoch 46/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0297 - mse: 0.0297 - mae: 0.1255 - val_loss: 0.0546 - val_mse: 0.0546 -
Epoch 47/50
```

```
49/49 [==============================] - 0s 4ms/step - loss: 0.0304 - mse: 0.0304 - mae: 0.1270 - val_loss: 0.0461 - val_mse: 0.0461 -
Epoch 48/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0285 - mse: 0.0285 - mae: 0.1248 - val_loss: 0.0463 - val_mse: 0.0463 -
Epoch 49/50
49/49 [==============================] - 0s 4ms/step - loss: 0.0287 - mse: 0.0287 - mae: 0.1259 - val_loss: 0.0443 - val_mse: 0.0443 -
Epoch 50/50
49/49 [==============================] - 0s 5ms/step - loss: 0.0285 - mse: 0.0285 - mae: 0.1238 - val_loss: 0.0448 - val_mse: 0.0448 -
```

```
# Plotting vaidation and training loss
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Loss (Split 80:20)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Loss (Split 80:20)

60:40

```
model2 = Sequential()
model2.add(Dense(13, input_dim=13, kernel_initializer='normal', activation='relu'))
model2.add(Dense(200, activation='relu'))
model2.add(Dense(1, activation='linear'))
model2.summary()
```

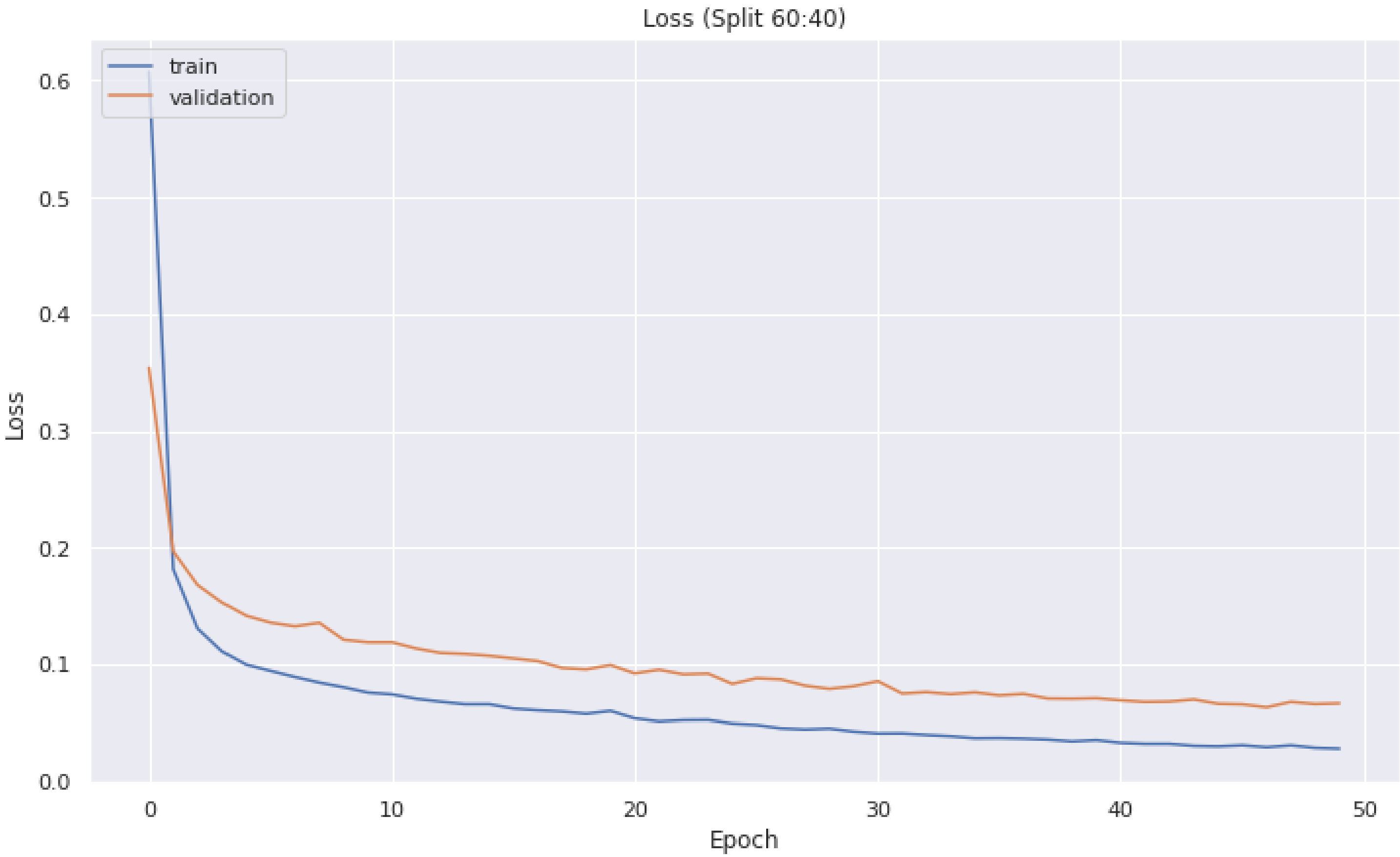
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 13)                182
_________________________________________________________________
dense_7 (Dense)              (None, 200)               2800
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 201
=================================================================
Total params: 3,183
Trainable params: 3,183
Non-trainable params: 0
_________________________________________________________________
```

```
model2.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history2=model2.fit(xtrain_scale, ytrain_scale, epochs=50, batch_size=25, verbose=1, validation_split=0.4, callbacks = [es])
predictions = model2.predict(xval_scale)
```

```
Epoch 23/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0522 - mse: 0.0522 - mae: 0.1690 - val_loss: 0.0912 - val_mse: 0.0912 -
Epoch 24/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0523 - mse: 0.0523 - mae: 0.1679 - val_loss: 0.0919 - val_mse: 0.0919 -
Epoch 25/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0491 - mse: 0.0491 - mae: 0.1622 - val_loss: 0.0830 - val_mse: 0.0830 -
Epoch 26/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0480 - mse: 0.0480 - mae: 0.1630 - val_loss: 0.0879 - val_mse: 0.0879 -
Epoch 27/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0451 - mse: 0.0451 - mae: 0.1559 - val_loss: 0.0869 - val_mse: 0.0869 -
Epoch 28/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0442 - mse: 0.0442 - mae: 0.1529 - val_loss: 0.0815 - val_mse: 0.0815 -
```

```
Epoch 29/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0449 - mse: 0.0449 - mae: 0.1562 - val_loss: 0.0788 - val_mse: 0.0788 -
Epoch 30/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0423 - mse: 0.0423 - mae: 0.1507 - val_loss: 0.0810 - val_mse: 0.0810 -
Epoch 31/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0408 - mse: 0.0408 - mae: 0.1484 - val_loss: 0.0853 - val_mse: 0.0853 -
Epoch 32/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0409 - mse: 0.0409 - mae: 0.1468 - val_loss: 0.0748 - val_mse: 0.0748 -
Epoch 33/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0395 - mse: 0.0395 - mae: 0.1442 - val_loss: 0.0760 - val_mse: 0.0760 -
Epoch 34/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0384 - mse: 0.0384 - mae: 0.1420 - val_loss: 0.0743 - val_mse: 0.0743 -
Epoch 35/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0367 - mse: 0.0367 - mae: 0.1388 - val_loss: 0.0758 - val_mse: 0.0758 -
Epoch 36/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0369 - mse: 0.0369 - mae: 0.1382 - val_loss: 0.0732 - val_mse: 0.0732 -
Epoch 37/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0364 - mse: 0.0364 - mae: 0.1401 - val_loss: 0.0745 - val_mse: 0.0745 -
Epoch 38/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0356 - mse: 0.0356 - mae: 0.1401 - val_loss: 0.0705 - val_mse: 0.0705 -
Epoch 39/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0341 - mse: 0.0341 - mae: 0.1331 - val_loss: 0.0703 - val_mse: 0.0703 -
Epoch 40/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0352 - mse: 0.0352 - mae: 0.1358 - val_loss: 0.0708 - val_mse: 0.0708 -
Epoch 41/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0328 - mse: 0.0328 - mae: 0.1318 - val_loss: 0.0690 - val_mse: 0.0690 -
Epoch 42/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0320 - mse: 0.0320 - mae: 0.1287 - val_loss: 0.0676 - val_mse: 0.0676 -
Epoch 43/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0320 - mse: 0.0320 - mae: 0.1317 - val_loss: 0.0679 - val_mse: 0.0679 -
Epoch 44/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0302 - mse: 0.0302 - mae: 0.1255 - val_loss: 0.0697 - val_mse: 0.0697 -
Epoch 45/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0298 - mse: 0.0298 - mae: 0.1255 - val_loss: 0.0660 - val_mse: 0.0660 -
Epoch 46/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0308 - mse: 0.0308 - mae: 0.1284 - val_loss: 0.0655 - val_mse: 0.0655 -
Epoch 47/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0291 - mse: 0.0291 - mae: 0.1239 - val_loss: 0.0630 - val_mse: 0.0630 -
Epoch 48/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0307 - mse: 0.0307 - mae: 0.1264 - val_loss: 0.0676 - val_mse: 0.0676 -
Epoch 49/50
37/37 [==============================] - 0s 5ms/step - loss: 0.0285 - mse: 0.0285 - mae: 0.1213 - val_loss: 0.0658 - val_mse: 0.0658 -
Epoch 50/50
37/37 [==============================] - 0s 6ms/step - loss: 0.0279 - mse: 0.0279 - mae: 0.1209 - val_loss: 0.0664 - val_mse: 0.0664 -
```

```
# Plotting vaidation and training loss
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Loss (Split 60:40)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



90:10

```
model3 = Sequential()
model3.add(Dense(13, input_dim=13, kernel_initializer='normal', activation='relu'))
```

```
model3.add(Dense(200, activation='relu'))
model3.add(Dense(1, activation='linear'))
model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 13)                182
_________________________________________________________________
dense_10 (Dense)             (None, 200)               2800
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 201
=================================================================
Total params: 3,183
Trainable params: 3,183
Non-trainable params: 0
_________________________________________________________________
```

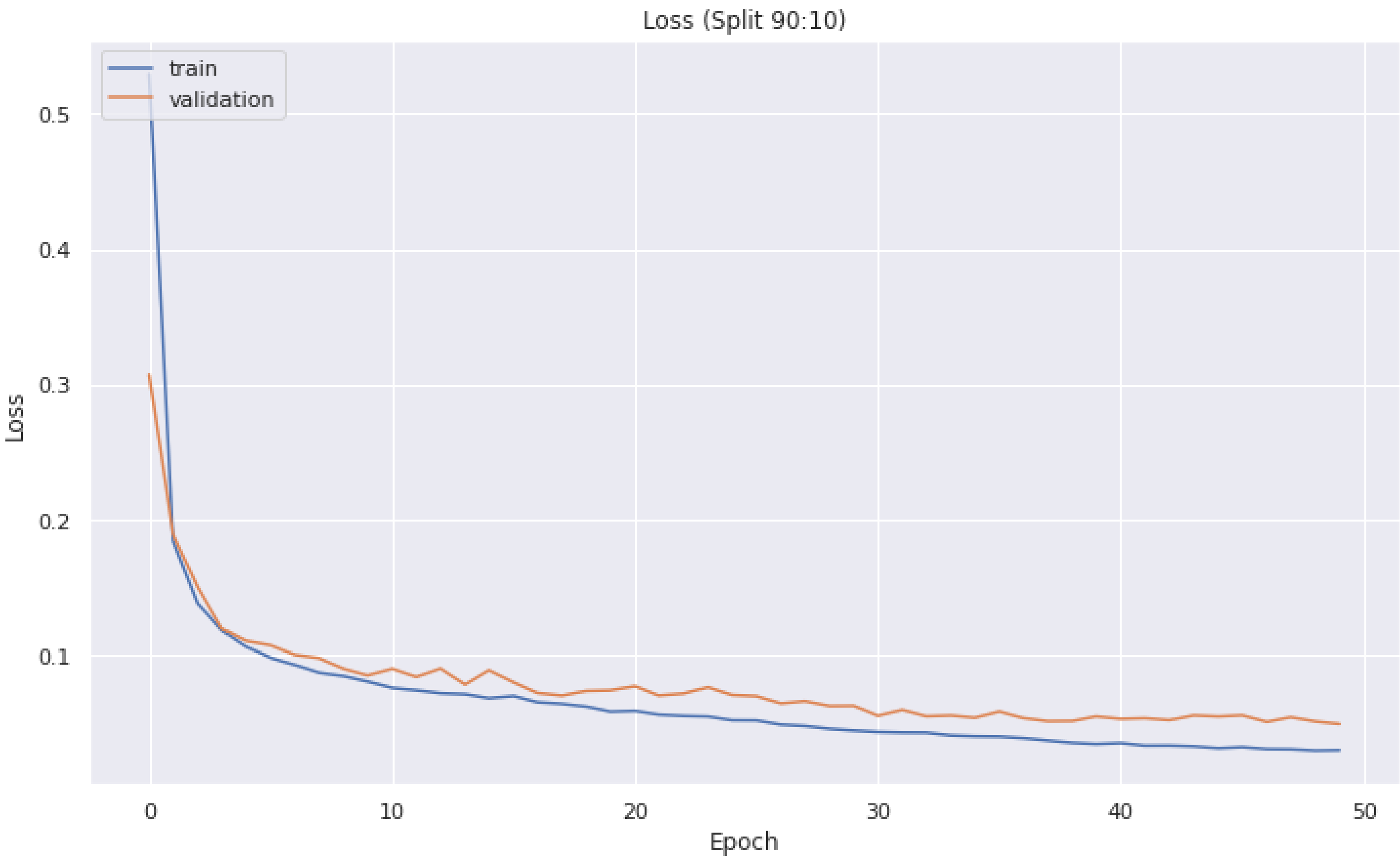
```
model3.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history3=model3.fit(xtrain_scale, ytrain_scale, epochs=50, batch_size=25, verbose=1, validation_split=0.1, callbacks = [es])
predictions = model3.predict(xval_scale)
```

```
Epoch 23/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0553 - mse: 0.0553 - mae: 0.1663 - val_loss: 0.0719 - val_mse: 0.0719 -
Epoch 24/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0549 - mse: 0.0549 - mae: 0.1643 - val_loss: 0.0765 - val_mse: 0.0765 -
Epoch 25/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0521 - mse: 0.0521 - mae: 0.1595 - val_loss: 0.0709 - val_mse: 0.0709 -
Epoch 26/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0520 - mse: 0.0520 - mae: 0.1626 - val_loss: 0.0701 - val_mse: 0.0701 -
Epoch 27/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0488 - mse: 0.0488 - mae: 0.1549 - val_loss: 0.0647 - val_mse: 0.0647 -
Epoch 28/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0478 - mse: 0.0478 - mae: 0.1510 - val_loss: 0.0663 - val_mse: 0.0663 -
Epoch 29/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0457 - mse: 0.0457 - mae: 0.1493 - val_loss: 0.0627 - val_mse: 0.0627 -
Epoch 30/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0445 - mse: 0.0445 - mae: 0.1490 - val_loss: 0.0629 - val_mse: 0.0629 -
Epoch 31/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0434 - mse: 0.0434 - mae: 0.1463 - val_loss: 0.0554 - val_mse: 0.0554 -
Epoch 32/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0431 - mse: 0.0431 - mae: 0.1452 - val_loss: 0.0598 - val_mse: 0.0598 -
Epoch 33/50
```

```
55/55 [==============================] - 0s 4ms/step - loss: 0.0430 - mse: 0.0430 - mae: 0.1440 - val_loss: 0.0551 - val_mse: 0.0551 -
Epoch 34/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0410 - mse: 0.0410 - mae: 0.1420 - val_loss: 0.0557 - val_mse: 0.0557 -
Epoch 35/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0403 - mse: 0.0403 - mae: 0.1433 - val_loss: 0.0540 - val_mse: 0.0540 -
Epoch 36/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0401 - mse: 0.0401 - mae: 0.1439 - val_loss: 0.0587 - val_mse: 0.0587 -
Epoch 37/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0390 - mse: 0.0390 - mae: 0.1417 - val_loss: 0.0537 - val_mse: 0.0537 -
Epoch 38/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0372 - mse: 0.0372 - mae: 0.1370 - val_loss: 0.0514 - val_mse: 0.0514 -
Epoch 39/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0356 - mse: 0.0356 - mae: 0.1323 - val_loss: 0.0515 - val_mse: 0.0515 -
Epoch 40/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0347 - mse: 0.0347 - mae: 0.1295 - val_loss: 0.0550 - val_mse: 0.0550 -
Epoch 41/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0354 - mse: 0.0354 - mae: 0.1349 - val_loss: 0.0531 - val_mse: 0.0531 -
Epoch 42/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0335 - mse: 0.0335 - mae: 0.1300 - val_loss: 0.0536 - val_mse: 0.0536 -
Epoch 43/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0335 - mse: 0.0335 - mae: 0.1296 - val_loss: 0.0522 - val_mse: 0.0522 -
Epoch 44/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0329 - mse: 0.0329 - mae: 0.1301 - val_loss: 0.0558 - val_mse: 0.0558 -
Epoch 45/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0315 - mse: 0.0315 - mae: 0.1262 - val_loss: 0.0549 - val_mse: 0.0549 -
Epoch 46/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0324 - mse: 0.0324 - mae: 0.1281 - val_loss: 0.0558 - val_mse: 0.0558 -
Epoch 47/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0309 - mse: 0.0309 - mae: 0.1253 - val_loss: 0.0509 - val_mse: 0.0509 -
Epoch 48/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0307 - mse: 0.0307 - mae: 0.1251 - val_loss: 0.0544 - val_mse: 0.0544 -
Epoch 49/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0297 - mse: 0.0297 - mae: 0.1223 - val_loss: 0.0512 - val_mse: 0.0512 -
Epoch 50/50
55/55 [==============================] - 0s 4ms/step - loss: 0.0300 - mse: 0.0300 - mae: 0.1247 - val_loss: 0.0493 - val_mse: 0.0493 -
```

```
# Plotting vaidation and training loss
plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('Loss (Split 90:10)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

```
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



0s completed at 5:43 AM