Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import plotly as py
py.offline.init_notebook_mode(connected=True)                #to display plotly graph in offline mode
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

Reading Data files

In [2]:

```
data = pd.read_csv('entire-world-economic-outlook-database.csv')
df1 = pd.read_csv('coronav.csv')
df2=pd.read_csv('GDP.csv')
```

Dataset 1

In [3]:

```
data
```

Out[3]:

	WEO Country Code	ISO	WEO Subject Code	Country	Subject Descriptor	Subject Notes	Units	Scale	Country/Series-specific Notes	1980	...	2017
0	512	AFG	NGAP_NPGDP	Afghanistan	Output gap in percent of potential GDP	Output gaps for advanced economies are calcula...	Percent of potential GDP	NaN	NaN	NaN	...	NaN
1	914	ALB	PPPGDP	Albania	Gross domestic product, current prices	These data form the basis for the country weig...	Purchasing power parity; international dollars	Billions	See notes for: Gross domestic product, curren...	5.765	...	37.609
2	914	ALB	PCPI	Albania	Inflation, average consumer prices	Expressed in averages for the year, not end-of...	Index	NaN	Source: National Statistics Office Latest actu...	NaN	...	103.295
3	612	DZA	NGDP_R	Algeria	Gross domestic product, constant prices	Expressed in billions of national currency uni...	National currency	Billions	Source: National Statistics Office Latest actu...	2596.368	...	7364.675
4	612	DZA	PCPI	Algeria	Inflation, average consumer prices	Expressed in averages for the year, not end-of...	Index	NaN	Source: National Statistics Office Latest actu...	8.975	...	193.970
...	...	...	...	...	...	...	...	...	...	...	...	...
8770	582	VNM	GGXCNL_NGDP	Vietnam	General government net lending/borrowing	Net lending (+)/ borrowing (-) is calculated a...	Percent of GDP	NaN	See notes for: General government net lending...	NaN	...	-1.964
8771	582	VNM	BCA_NGDPD	Vietnam	Current account balance	Current account is all transactions other than...	Percent of GDP	NaN	See notes for: Gross domestic product, curren...	-1.599	...	-0.596
8772	474	YEM	GGX	Yemen	General government total expenditure	Total expenditure consists of total expense an...	National currency	Billions	Source: Ministry of Finance or Treasury Latest...	NaN	...	839.751
8773	474	YEM	GGXWDN_NGDP	Yemen	General government net debt	Net debt is calculated as gross debt minus fin...	Percent of GDP	NaN	See notes for: General government net debt (N...	NaN	...	76.559
8774	754	ZMB	NGDPRPPPPC	Zambia	Gross domestic product per capita, constant	GDP is expressed in constant	Purchasing power parity; 2017	Units	See notes for: Gross domestic product,	2957.214	...	3407.306

pr... international dol... international do... consta...

8775 rows × 56 columns

Data Preprocessing

In [4]:
```
data.drop(['Subject Notes','Country/Series-specific Notes','WEO Subject Code','Scale','ISO','Units'],axis=1,inpla
```

In [5]:
```
data.shape
```

Out[5]:
```
(8775, 50)
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8775 entries, 0 to 8774
Data columns (total 50 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   WEO Country Code        8775 non-null   int64  
 1   Country                 8775 non-null   object 
 2   Subject Descriptor      8775 non-null   object 
 3   1980                    3808 non-null   float64
 4   1981                    3952 non-null   float64
 5   1982                    3994 non-null   float64
 6   1983                    4041 non-null   float64
 7   1984                    4072 non-null   float64
 8   1985                    4142 non-null   float64
 9   1986                    4185 non-null   float64
 10  1987                    4204 non-null   float64
 11  1988                    4297 non-null   float64
 12  1989                    4359 non-null   float64
 13  1990                    4855 non-null   float64
 14  1991                    5016 non-null   float64
 15  1992                    5391 non-null   float64
 16  1993                    5595 non-null   float64
 17  1994                    5724 non-null   float64
 18  1995                    6086 non-null   float64
 19  1996                    6230 non-null   float64
 20  1997                    6396 non-null   float64
 21  1998                    6573 non-null   float64
 22  1999                    6651 non-null   float64
 23  2000                    7056 non-null   float64
 24  2001                    7181 non-null   float64
 25  2002                    7272 non-null   float64
 26  2003                    7312 non-null   float64
 27  2004                    7381 non-null   float64
 28  2005                    7418 non-null   float64
 29  2006                    7421 non-null   float64
 30  2007                    7431 non-null   float64
 31  2008                    7443 non-null   float64
 32  2009                    7463 non-null   float64
 33  2010                    7462 non-null   float64
 34  2011                    7469 non-null   float64
 35  2012                    7473 non-null   float64
 36  2013                    7484 non-null   float64
 37  2014                    7515 non-null   float64
 38  2015                    7534 non-null   float64
 39  2016                    7539 non-null   float64
 40  2017                    7542 non-null   float64
 41  2018                    7538 non-null   float64
 42  2019                    7531 non-null   float64
 43  2020                    7491 non-null   float64
 44  2021                    7441 non-null   float64
 45  2022                    7381 non-null   float64
 46  2023                    7364 non-null   object 
 47  2024                    7364 non-null   object 
 48  2025                    7364 non-null   object 
 49  Estimates Start After   7585 non-null   float64
dtypes: float64(44), int64(1), object(5)
memory usage: 3.3+ MB
```

In [7]:
```python
# checking and sum up all null values present
data.isnull().sum()
```

Out[7]:
```
WEO Country Code           0
Country                    0
Subject Descriptor         0
1980                    4967
1981                    4823
1982                    4781
1983                    4734
1984                    4703
1985                    4633
1986                    4590
1987                    4571
1988                    4478
1989                    4416
1990                    3920
1991                    3759
1992                    3384
1993                    3180
1994                    3051
1995                    2689
1996                    2545
1997                    2379
1998                    2202
1999                    2124
2000                    1719
2001                    1594
2002                    1503
2003                    1463
2004                    1394
2005                    1357
2006                    1354
2007                    1344
2008                    1332
2009                    1312
2010                    1313
2011                    1306
2012                    1302
2013                    1291
2014                    1260
2015                    1241
2016                    1236
2017                    1233
2018                    1237
2019                    1244
2020                    1284
2021                    1334
2022                    1394
2023                    1411
2024                    1411
2025                    1411
Estimates Start After   1190
dtype: int64
```

In [8]:
```python
data.describe()
```

Out[8]:

	WEO Country Code	1980	1981	1982	1983	1984	1985	1986	1987	
count	8775.000000	3.808000e+03	3.952000e+03	3.994000e+03	4.041000e+03	4.072000e+03	4.142000e+03	4.185000e+03	4.204000e+03	4.
mean	553.328205	4.885289e+04	4.592886e+04	4.586402e+04	4.757828e+04	4.666190e+04	4.656609e+04	4.472152e+04	4.611228e+04	4.
std	260.740915	1.215854e+06	1.080641e+06	1.141182e+06	1.226333e+06	1.094456e+06	1.047599e+06	9.408252e+05	9.557539e+05	8.
min	111.000000	-1.130530e+04	-1.057900e+04	-1.100380e+04	-1.191340e+04	-8.389300e+03	-4.667100e+03	-4.865800e+03	-1.341300e+03	-3.
25%	314.000000	7.782500e-01	7.160000e-01	5.140000e-01	5.280000e-01	1.034500e+00	7.665000e-01	8.340000e-01	1.103750e+00	1.
50%	566.000000	1.255750e+01	1.168000e+01	1.061100e+01	1.062300e+01	1.218750e+01	1.111100e+01	1.231100e+01	1.337250e+01	1.
75%	734.000000	9.947725e+01	9.570875e+01	1.000340e+02	1.016610e+02	1.059045e+02	1.074885e+02	1.050650e+02	1.217275e+02	1.
max	968.000000	6.932238e+07	6.100644e+07	6.639857e+07	7.270569e+07	6.305774e+07	5.943432e+07	5.203190e+07	5.229042e+07	4.

8 rows × 45 columns

In [9]:
```python
data['2023'] = pd.to_numeric(data['2023'], errors='coerce')
data['2024'] = pd.to_numeric(data['2024'], errors='coerce')
```

```
data['2025'] = pd.to_numeric(data['2025'], errors='coerce')
```

In [10]:
```
data.dtypes
```

Out[10]:
```
WEO Country Code           int64
Country                   object
Subject Descriptor        object
1980                     float64
1981                     float64
1982                     float64
1983                     float64
1984                     float64
1985                     float64
1986                     float64
1987                     float64
1988                     float64
1989                     float64
1990                     float64
1991                     float64
1992                     float64
1993                     float64
1994                     float64
1995                     float64
1996                     float64
1997                     float64
1998                     float64
1999                     float64
2000                     float64
2001                     float64
2002                     float64
2003                     float64
2004                     float64
2005                     float64
2006                     float64
2007                     float64
2008                     float64
2009                     float64
2010                     float64
2011                     float64
2012                     float64
2013                     float64
2014                     float64
2015                     float64
2016                     float64
2017                     float64
2018                     float64
2019                     float64
2020                     float64
2021                     float64
2022                     float64
2023                     float64
2024                     float64
2025                     float64
Estimates Start After    float64
dtype: object
```

Filling null values

In [11]:
```
X = data.iloc[:, 0:3]
Y = data.iloc[: , 3:46]
Z= data.iloc[: , 46:51]
```

In [13]:
```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan,strategy='median')
data_n = imp.fit_transform(Y)
data_new=pd.DataFrame(data_n , columns = Y.columns)
```

In [14]:
```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan,strategy='most_frequent')
data1 = imp.fit_transform(Z)
data_1 = pd.DataFrame(data1 , columns=Z.columns)
```

In [15]:
```
d1 = pd.concat([X , data_new,data_1],axis=1)
```

In [16]:
```
d1
```

```
d1
```

Out[16]:

	WEO Country Code	Country	Subject Descriptor	1980	1981	1982	1983	1984	1985	1986	...	2017	2018
0	512	Afghanistan	Output gap in percent of potential GDP	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	...	20.5095	21.0465
1	914	Albania	Gross domestic product, current prices	5.7650	6.671	7.288	7.657	8.0920	8.223	8.858	...	37.6090	40.0800
2	914	Albania	Inflation, average consumer prices	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	...	103.2950	105.3900
3	612	Algeria	Gross domestic product, constant prices	2596.3680	2674.259	2845.412	2999.064	3167.0120	3344.364	3337.676	...	7364.6750	7467.7800
4	612	Algeria	Inflation, average consumer prices	8.9750	10.286	10.964	11.823	12.5690	13.880	15.824	...	193.9700	202.2530
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8770	582	Vietnam	General government net lending/borrowing	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	...	-1.9640	-1.0230
8771	582	Vietnam	Current account balance	-1.5990	-4.197	-2.635	-1.946	-1.6020	-4.951	-3.441	...	-0.5960	1.8980
8772	474	Yemen	General government total expenditure	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	...	839.7510	1651.6000
8773	474	Yemen	General government net debt	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	...	76.5590	73.7950
8774	754	Zambia	Gross domestic product per capita, constant pr...	2957.2140	3052.876	2871.765	2752.526	2624.5240	2578.899	2546.290	...	3407.3060	3438.0800

8775 rows × 50 columns

In [17]:

```
d1.isnull().sum()
```

Out[17]:

```
WEO Country Code        0
Country                 0
Subject Descriptor      0
1980                    0
1981                    0
1982                    0
1983                    0
1984                    0
1985                    0
1986                    0
1987                    0
1988                    0
1989                    0
1990                    0
1991                    0
1992                    0
1993                    0
1994                    0
1995                    0
1996                    0
1997                    0
1998                    0
1999                    0
2000                    0
2001                    0
2002                    0
2003                    0
2004                    0
2005                    0
2006                    0
2007                    0
2008                    0
2009                    0
2010                    0
2011                    0
2012                    0
2013                    0
2014                    0
2015                    0
```

```
2016                    0
2017                    0
2018                    0
2019                    0
2020                    0
2021                    0
2022                    0
2023                    0
2024                    0
2025                    0
Estimates Start After   0
dtype: int64
```

In [18]:

```
d1.corr()
```

Out[18]:

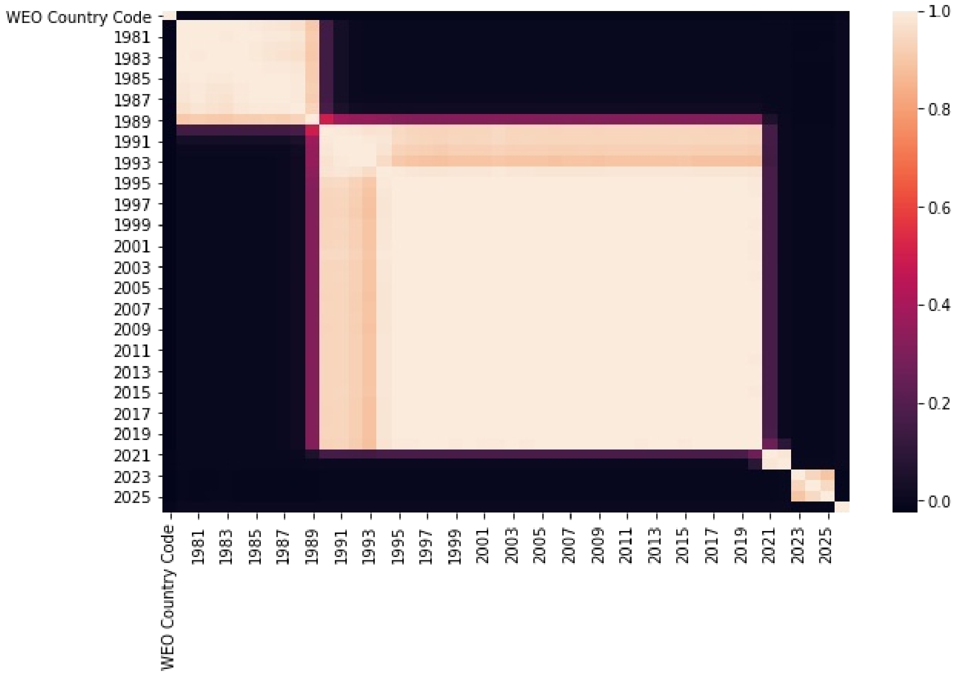
	WEO Country Code	1980	1981	1982	1983	1984	1985	1986	1987	1988	...	2017	2018
WEO Country Code	1.000000	-0.011804	-0.012722	-0.011883	-0.011408	-0.012453	-0.013095	-0.014071	-0.014592	-0.015905	...	-0.019129	-0.019129
1980	-0.011804	1.000000	0.997924	0.998231	0.998666	0.997618	0.992820	0.985471	0.980168	0.971583	...	-0.000396	-0.000396
1981	-0.012722	0.997924	1.000000	0.996925	0.995235	0.998671	0.996443	0.992465	0.988023	0.982671	...	-0.000425	-0.000425
1982	-0.011883	0.998231	0.996925	1.000000	0.999079	0.996183	0.989654	0.983473	0.976620	0.972819	...	-0.000400	-0.000400
1983	-0.011408	0.998666	0.995235	0.999079	1.000000	0.996117	0.989310	0.981610	0.975170	0.968496	...	-0.000379	-0.000379
1984	-0.012453	0.997618	0.998671	0.996183	0.996117	1.000000	0.998258	0.994361	0.990582	0.984212	...	-0.000382	-0.000382
1985	-0.013095	0.992820	0.996443	0.989654	0.989310	0.998258	1.000000	0.998233	0.996403	0.989336	...	-0.000269	-0.000269
1986	-0.014071	0.985471	0.992465	0.983473	0.981610	0.994361	0.998233	1.000000	0.999042	0.995732	...	-0.000064	-0.000064
1987	-0.014592	0.980168	0.988023	0.976620	0.975170	0.990582	0.996403	0.999042	1.000000	0.995587	...	0.001200	0.001200
1988	-0.015905	0.971583	0.982671	0.972819	0.968496	0.984212	0.989336	0.995732	0.995587	1.000000	...	0.017321	0.017323
1989	-0.020494	0.905081	0.914526	0.910968	0.905760	0.915340	0.916174	0.922246	0.919745	0.935611	...	0.314929	0.314996
1990	-0.020116	0.150646	0.151717	0.151698	0.151040	0.151955	0.151782	0.152582	0.153220	0.172029	...	0.939419	0.939538
1991	-0.018384	0.028614	0.028756	0.028766	0.028705	0.028883	0.028958	0.029239	0.030465	0.048313	...	0.941378	0.941508
1992	-0.017627	0.002090	0.002078	0.002100	0.002113	0.002131	0.002246	0.002443	0.003760	0.021495	...	0.922385	0.922535
1993	-0.016974	-0.000255	-0.000280	-0.000258	-0.000238	-0.000235	-0.000119	0.000063	0.001376	0.019085	...	0.890450	0.890627
1994	-0.018782	-0.000382	-0.000411	-0.000387	-0.000366	-0.000366	-0.000249	-0.000048	0.001270	0.018466	...	0.983155	0.983226
1995	-0.019112	-0.000392	-0.000420	-0.000396	-0.000375	-0.000377	-0.000263	-0.000059	0.001220	0.017596	...	0.999345	0.999359
1996	-0.019127	-0.000392	-0.000421	-0.000397	-0.000376	-0.000378	-0.000265	-0.000060	0.001207	0.017392	...	0.999962	0.999965
1997	-0.019129	-0.000393	-0.000422	-0.000397	-0.000376	-0.000379	-0.000266	-0.000061	0.001203	0.017329	...	1.000000	1.000000
1998	-0.019130	-0.000393	-0.000422	-0.000397	-0.000376	-0.000379	-0.000267	-0.000062	0.001200	0.017270	...	0.999977	0.999974
1999	-0.019126	-0.000393	-0.000422	-0.000398	-0.000377	-0.000379	-0.000266	-0.000061	0.001208	0.017418	...	0.999924	0.999929
2000	-0.019128	-0.000393	-0.000422	-0.000398	-0.000377	-0.000379	-0.000266	-0.000061	0.001206	0.017379	...	0.999973	0.999976
2001	-0.019126	-0.000394	-0.000423	-0.000398	-0.000377	-0.000379	-0.000266	-0.000062	0.001207	0.017409	...	0.999938	0.999942
2002	-0.019117	-0.000394	-0.000423	-0.000398	-0.000377	-0.000379	-0.000266	-0.000061	0.001215	0.017547	...	0.999562	0.999573
2003	-0.019128	-0.000394	-0.000423	-0.000399	-0.000378	-0.000380	-0.000267	-0.000062	0.001204	0.017365	...	0.999985	0.999987
2004	-0.019127	-0.000394	-0.000423	-0.000399	-0.000378	-0.000380	-0.000267	-0.000062	0.001206	0.017400	...	0.999949	0.999952
2005	-0.019128	-0.000394	-0.000423	-0.000399	-0.000378	-0.000380	-0.000267	-0.000063	0.001203	0.017358	...	0.999989	0.999991
2006	-0.019129	-0.000394	-0.000423	-0.000399	-0.000378	-0.000380	-0.000268	-0.000063	0.001201	0.017318	...	1.000000	1.000000
2007	-0.019128	-0.000394	-0.000423	-0.000399	-0.000378	-0.000380	-0.000267	-0.000063	0.001203	0.017349	...	0.999994	0.999995
2008	-0.019128	-0.000394	-0.000423	-0.000399	-0.000378	-0.000380	-0.000268	-0.000063	0.001203	0.017357	...	0.999990	0.999991
2009	-0.019129	-0.000395	-0.000424	-0.000399	-0.000378	-0.000381	-0.000268	-0.000063	0.001202	0.017336	...	0.999998	0.999999
2010	-0.019128	-0.000395	-0.000424	-0.000399	-0.000378	-0.000381	-0.000268	-0.000063	0.001204	0.017368	...	0.999982	0.999985
2011	-0.019128	-0.000395	-0.000424	-0.000399	-0.000378	-0.000381	-0.000268	-0.000063	0.001203	0.017363	...	0.999986	0.999988
2012	-0.019128	-0.000395	-0.000424	-0.000400	-0.000379	-0.000381	-0.000268	-0.000063	0.001204	0.017378	...	0.999973	0.999976
2013	-0.019128	-0.000395	-0.000424	-0.000400	-0.000379	-0.000381	-0.000268	-0.000063	0.001203	0.017367	...	0.999982	0.999984
2014	-0.019128	-0.000395	-0.000424	-0.000400	-0.000379	-0.000381	-0.000268	-0.000064	0.001203	0.017370	...	0.999980	0.999982
2015	-0.019125	-0.000396	-0.000425	-0.000400	-0.000379	-0.000381	-0.000268	-0.000063	0.001206	0.017427	...	0.999905	0.999910
2016	-0.019129	-0.000396	-0.000425	-0.000400	-0.000379	-0.000382	-0.000269	-0.000064	0.001201	0.017339	...	0.999997	0.999998
2017	-0.019129	-0.000396	-0.000425	-0.000400	-0.000379	-0.000382	-0.000269	-0.000064	0.001200	0.017321	...	1.000000	1.000000
2018	-0.019129	-0.000396	-0.000425	-0.000400	-0.000379	-0.000382	-0.000269	-0.000064	0.001200	0.017323	...	1.000000	1.000000

2019	-0.019146	-0.000397	-0.000426	-0.000401	-0.000380	-0.000382	-0.000270	-0.000065	0.001200	0.017344	...	0.999994	0.999995
2020	-0.020163	-0.000425	-0.000456	-0.000430	-0.000408	-0.000414	-0.000303	-0.000102	0.001157	0.017207	...	0.996031	0.996032
2021	-0.015572	-0.000401	-0.000432	-0.000410	-0.000397	-0.000431	-0.000431	-0.000424	-0.000214	0.002524	...	0.171721	0.171728
2022	-0.012521	-0.000339	-0.000366	-0.000348	-0.000338	-0.000372	-0.000391	-0.000420	-0.000423	-0.000412	...	0.002535	0.002542
2023	-0.023766	-0.007603	-0.008191	-0.007788	-0.007563	-0.008342	-0.008771	-0.009425	-0.009588	-0.010318	...	-0.004351	-0.004351
2024	-0.021236	-0.007635	-0.008227	-0.007821	-0.007596	-0.008378	-0.008809	-0.009466	-0.009630	-0.010363	...	-0.004369	-0.004369
2025	-0.022566	-0.007542	-0.008126	-0.007726	-0.007503	-0.008275	-0.008701	-0.009350	-0.009512	-0.010236	...	-0.004314	-0.004314
Estimates Start After	-0.010241	0.000641	0.000669	0.000696	0.000679	0.000678	0.000665	0.000715	0.000759	0.000873	...	0.000667	0.000667

48 rows × 48 columns

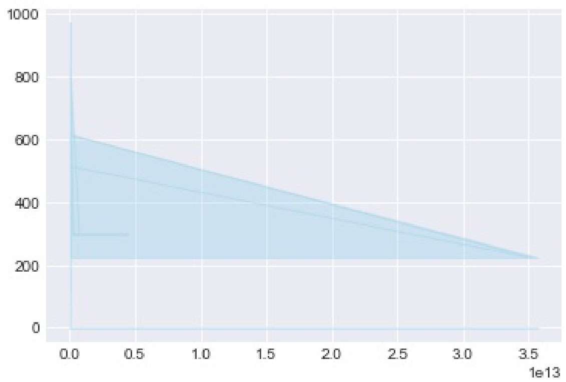
In [19]:
```
plt.figure(figsize=(10,6))
sns.heatmap(d1.corr())
```

Out[19]: <AxesSubplot:>



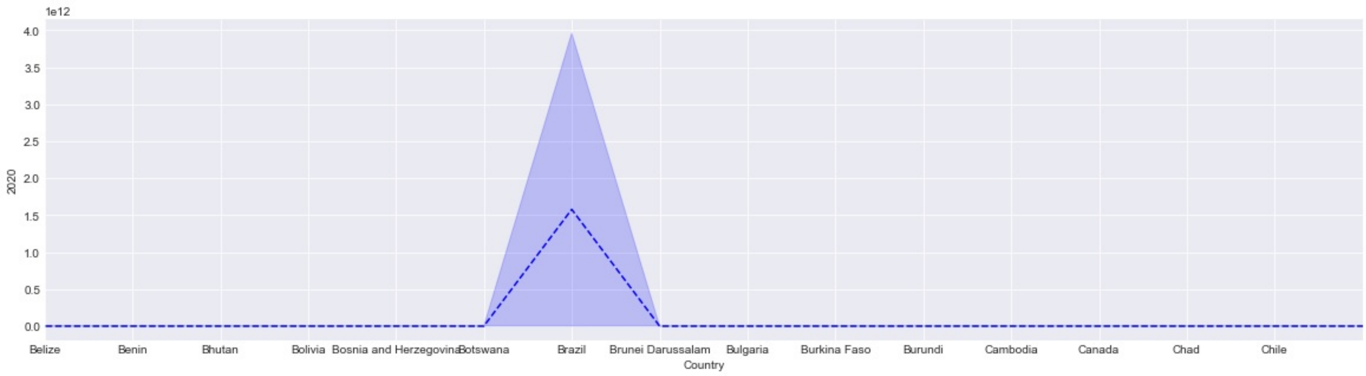
Data visualisation

In [20]:
```
#Area chart
plt.style.use('seaborn-darkgrid')
plt.fill_between(x='2020', y1='WEO Country Code',color="skyblue",alpha=0.3,data=d1)
plt.show()
```



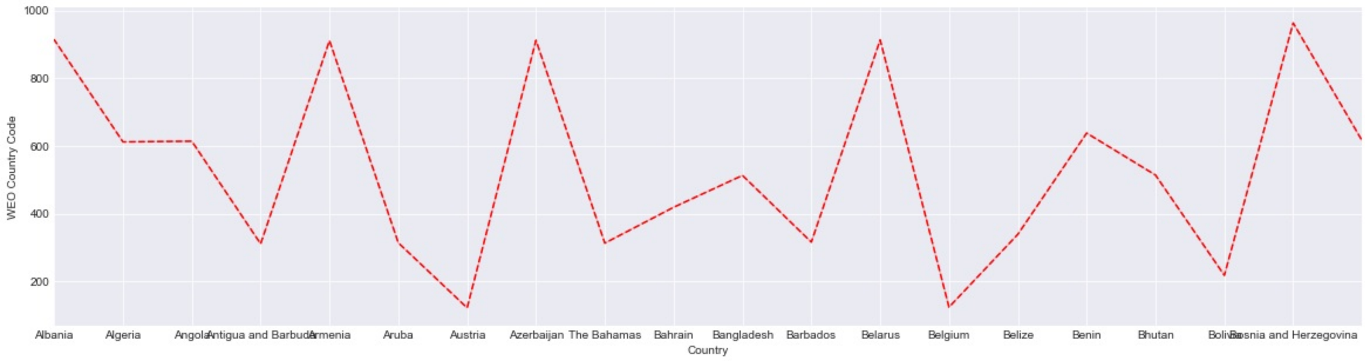
In [22]:
```
#lineplot
fig,ax=plt.subplots(figsize=(20,5))
```

```
sns.lineplot(d1['Country'], d1['2020'],ax=ax,linestyle = 'dashed',color = 'b')
ax.set_xlim(15,30)
ax.set_xticks(range(15,30))
plt.show()
```



In [21]:

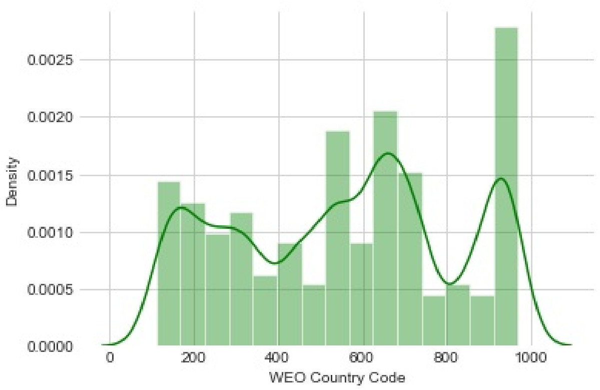
```
#lineplot
fig,ax=plt.subplots(figsize=(20,5))
sns.lineplot(d1['Country'], d1['WEO Country Code'],ax=ax,linestyle = 'dashed',color = 'r')
ax.set_xlim(1,20)
ax.set_xticks(range(1,20))
plt.show()
```



In [23]:

```
#distribution plot
sns.set_style('whitegrid')
sns.distplot(d1['WEO Country Code'], color ='green', bins = 15)
```

Out[23]: <AxesSubplot:xlabel='WEO Country Code', ylabel='Density'>



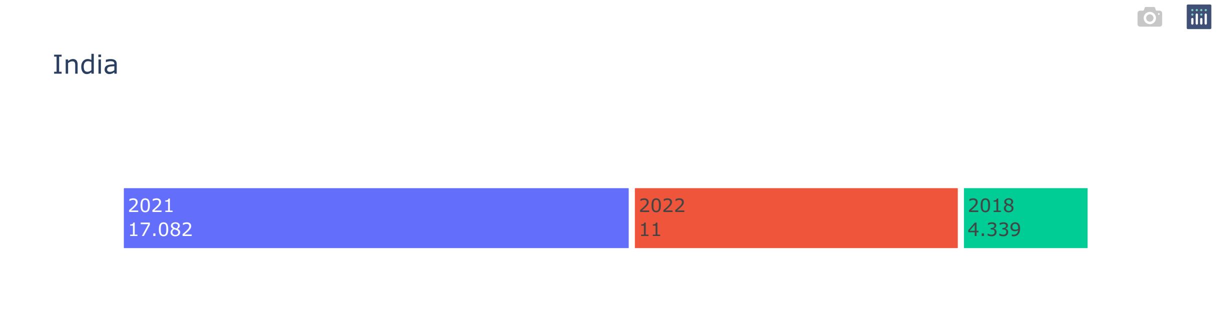
In [24]:

```
#retrieving data of country India
d_11=d1.query('Country=="India"')
```

In [25]:

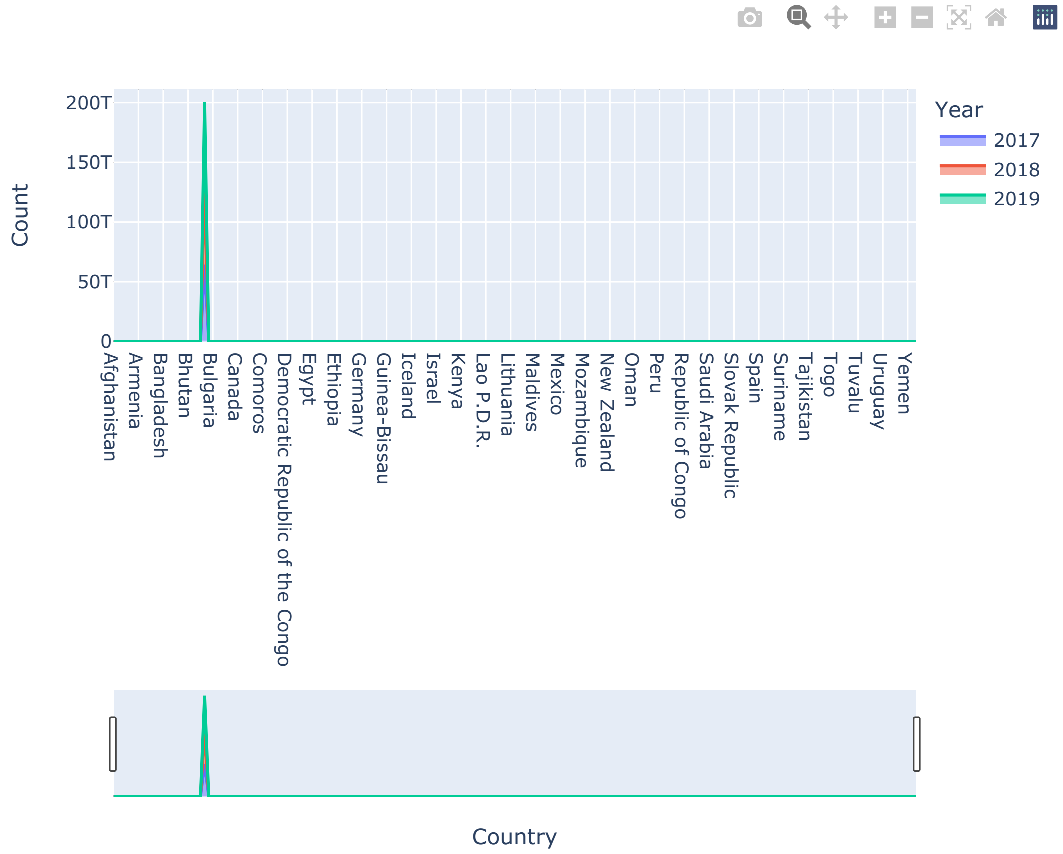
```
import plotly.express as px
temp = d_11.groupby('Subject Descriptor')['2018','2019','2021','2022'].sum().reset_index()
temp = temp[temp['Subject Descriptor']==max(temp['Subject Descriptor'])].reset_index(drop = True)
#melt plot
tm = temp.melt(id_vars = 'Subject Descriptor', value_vars = ['2018','2019','2021','2022'])
fig = px.treemap(tm, path = ['variable'], values = 'value', height = 250, width = 800,title= "India" )

fig.data[0].textinfo = 'label+text+value'
```

```
fig.show()
```



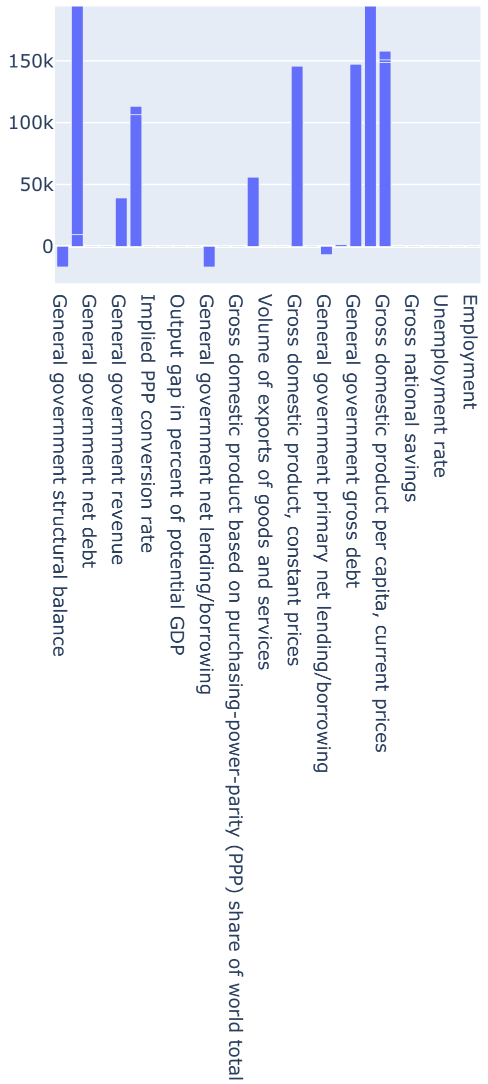
In [28]:

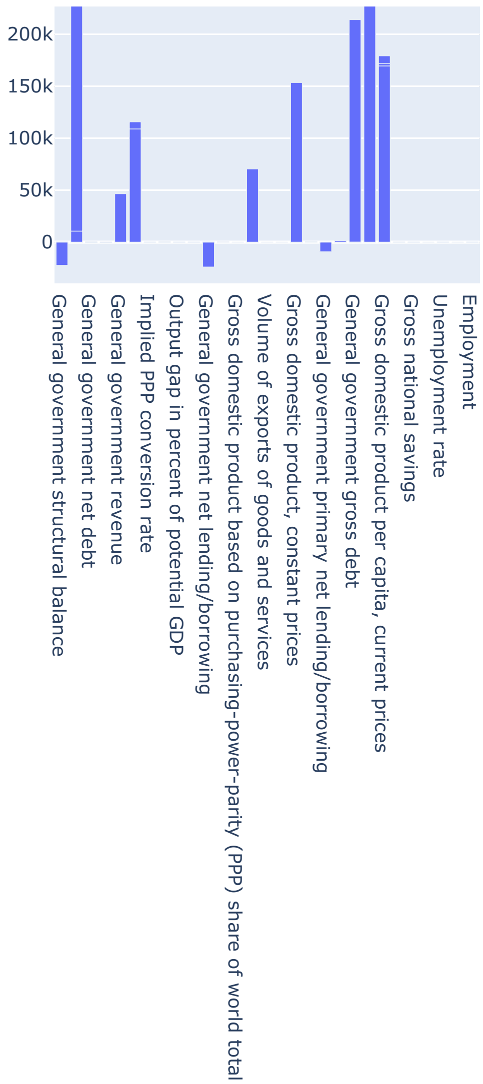
```
#area plot
temp = d1.groupby('Country')['2017','2018','2019'].sum().reset_index()
temp = temp.melt(id_vars = 'Country',value_vars = ['2017','2018','2019'], var_name = 'Year',value_name = 'Count')
fig = px.area(temp,x='Country',y='Count',color='Year',height = 600)
fig.update_layout(xaxis_rangeslider_visible=True)
fig.show()
```



In [29]:

```
from plotly.subplots import make_subplots
fig_c = px.bar(d_11,x='Subject Descriptor',y='2019')
fig_d = px.bar(d_11,x='Subject Descriptor',y='2022')

fig = make_subplots(rows=1,cols=2,shared_xaxes=False, horizontal_spacing=0.3,vertical_spacing=0.5,
subplot_titles=('2019 India','2022 India'))
fig.add_trace(fig_c['data'][0],row=1,col=1)
fig.add_trace(fig_d['data'][0],row=1,col=2)
fig.update_layout(autosize=False,width=950,height=840)
fig.show()
```

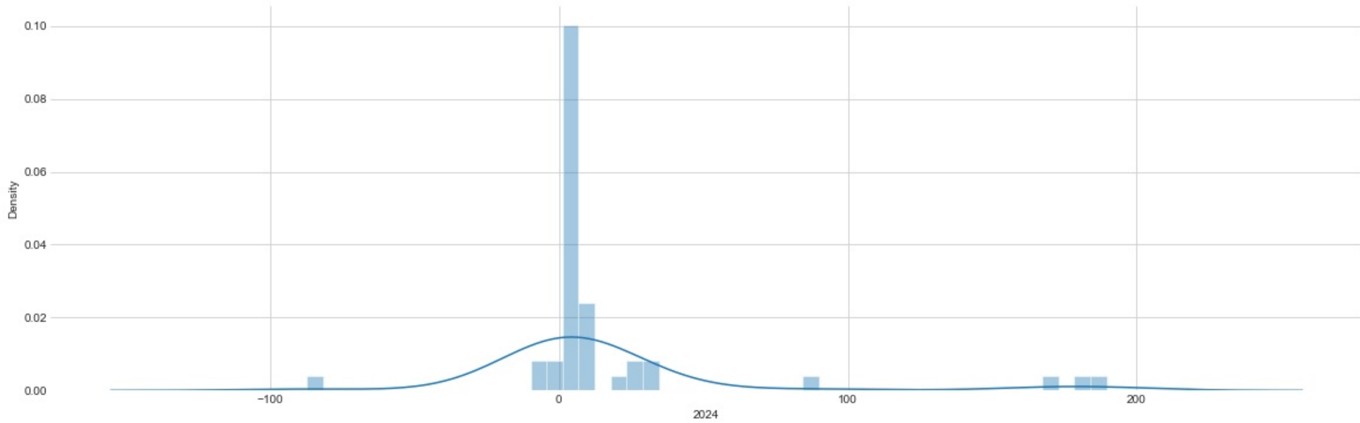




In [30]:

```
#distribution plot of country India
fig,ax=plt.subplots(figsize=(20,6))
sns.distplot(d_11["2024"])
```

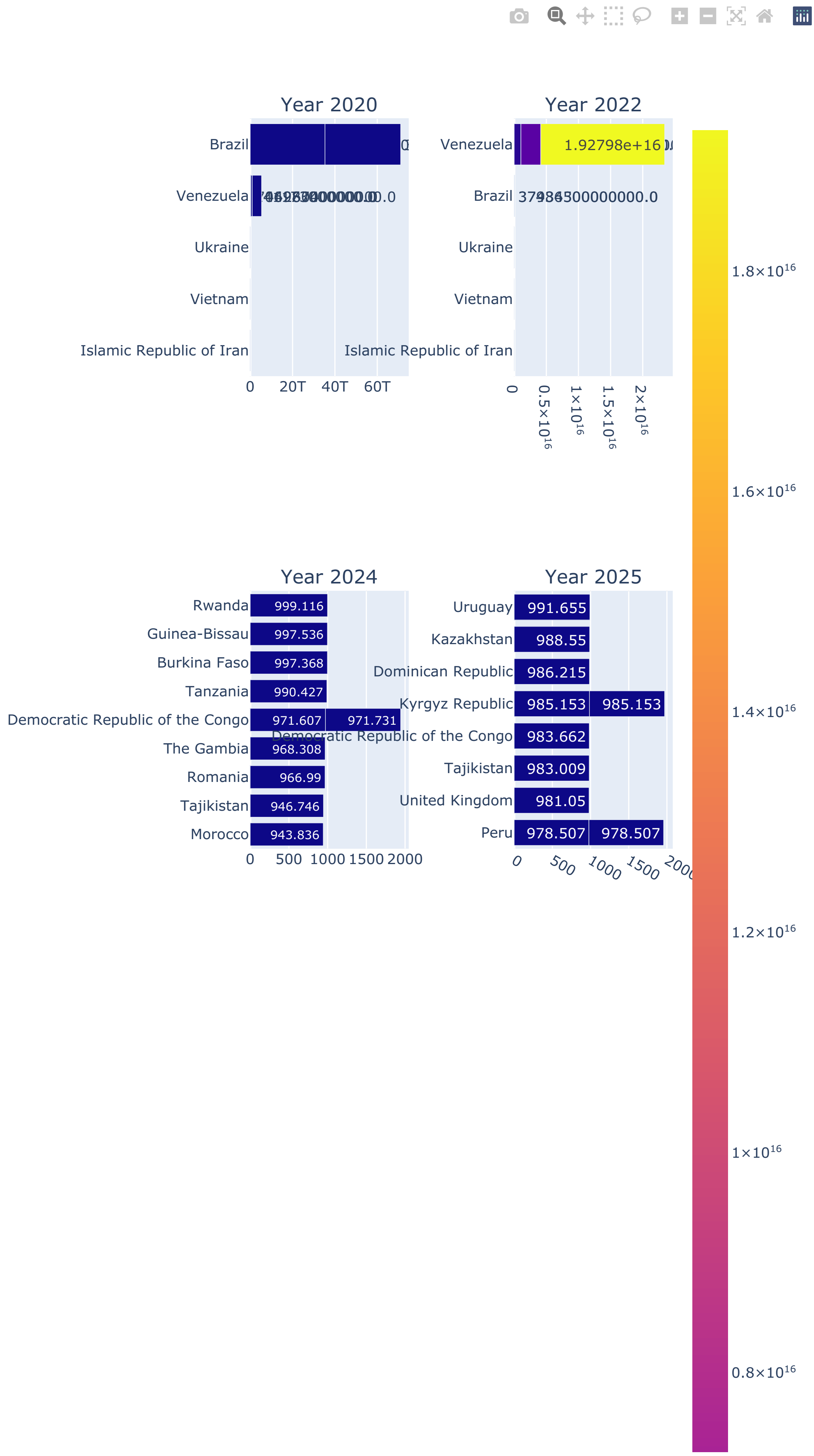
Out[30]: <AxesSubplot:xlabel='2024', ylabel='Density'>



In [31]:

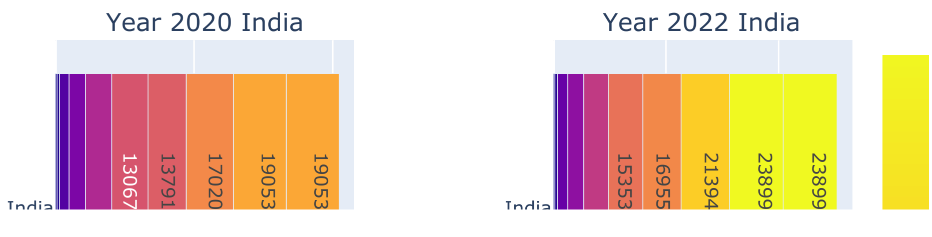
```
top=10
fig_c = px.bar(d1.sort_values('2020').tail(top),x = '2020'
               ,y='Country',text='2020',orientation='h', color='2020')

fig_a = px.bar(d1.sort_values('2022').tail(top),x = '2022'
               ,y='Country',text='2022',orientation='h', color='2022')

fig_dc = px.bar(d1.sort_values('2024').tail(top),x = '2024'
               ,y='Country',text='2024',orientation='h', color='2024')

fig_rc = px.bar(d1.sort_values('2025').tail(top),x = '2025'
,y='Country',text='2025',orientation='h', color='2025')
```

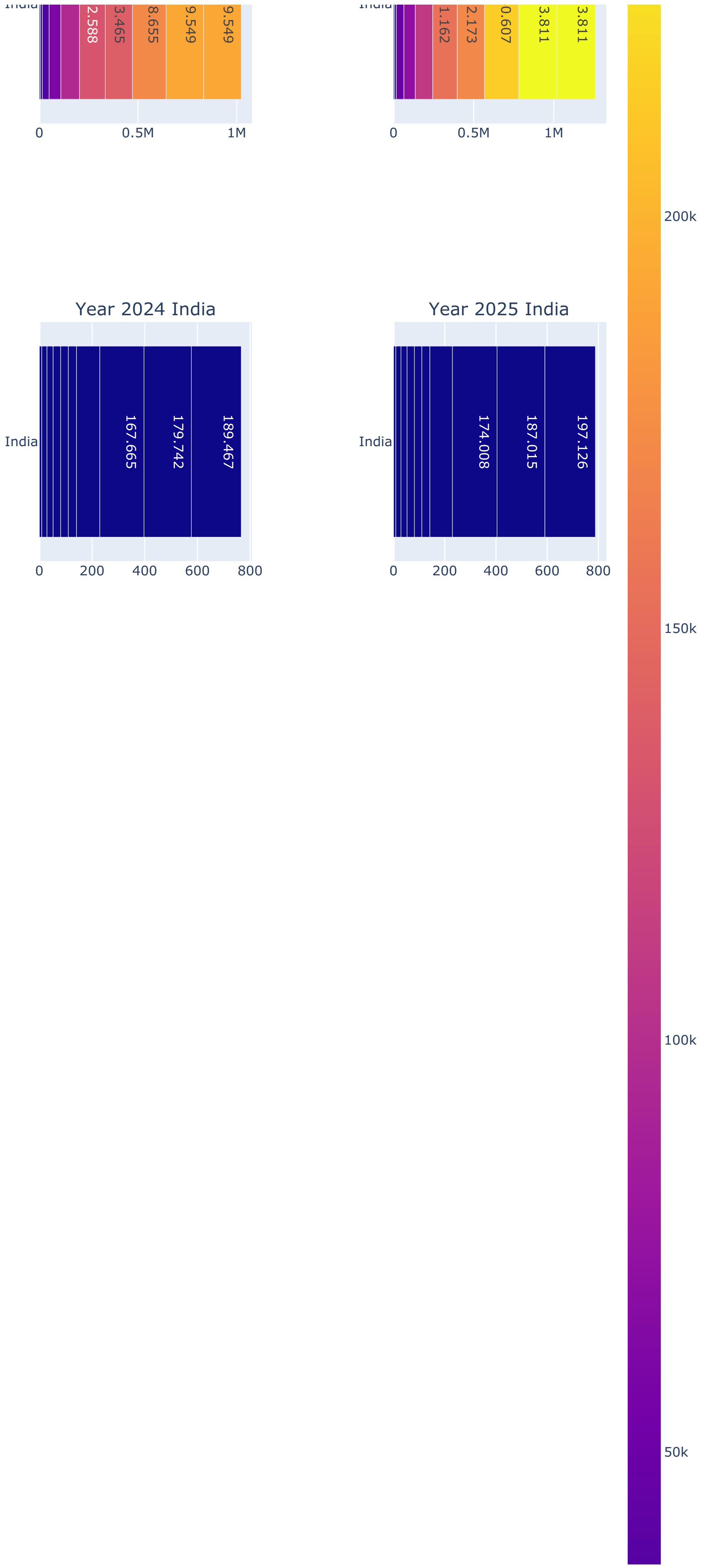
```python
fig = make_subplots(rows=5,cols=2,shared_xaxes=False,horizontal_spacing=0.25, vertical_spacing=.1,
                    subplot_titles=('Year 2020','Year 2022',
                                   'Year 2024','Year 2025'))

fig.add_trace(fig_c['data'][0],row=1,col=1)
fig.add_trace(fig_a['data'][0],row=1,col=2)
fig.add_trace(fig_dc['data'][0],row=2,col=1)
fig.add_trace(fig_rc['data'][0],row=2,col=2)
fig.update_layout(height=2000)
fig.show()
```

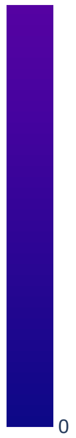


In [32]:

```
top=10
fig_c = px.bar(d_11.sort_values('2020').tail(top),x = '2020'
               ,y='Country',text='2020',orientation='h', color='2020')

fig_a = px.bar(d_11.sort_values('2022').tail(top),x = '2022'
               ,y='Country',text='2022',orientation='h', color='2022')

fig_dc = px.bar(d_11.sort_values('2024').tail(top),x = '2024'
               ,y='Country',text='2024',orientation='h', color='2024')

fig_rc = px.bar(d_11.sort_values('2025').tail(top),x = '2025'
,y='Country',text='2025',orientation='h', color='2025')

fig = make_subplots(rows=5,cols=2,shared_xaxes=False,horizontal_spacing=0.25, vertical_spacing=.1,
                    subplot_titles=('Year 2020 India','Year 2022 India',
                                    'Year 2024 India','Year 2025 India'))

fig.add_trace(fig_c['data'][0],row=1,col=1)
fig.add_trace(fig_a['data'][0],row=1,col=2)
fig.add_trace(fig_dc['data'][0],row=2,col=1)
fig.add_trace(fig_rc['data'][0],row=2,col=2)
fig.update_layout(height=2000)
fig.show()
```

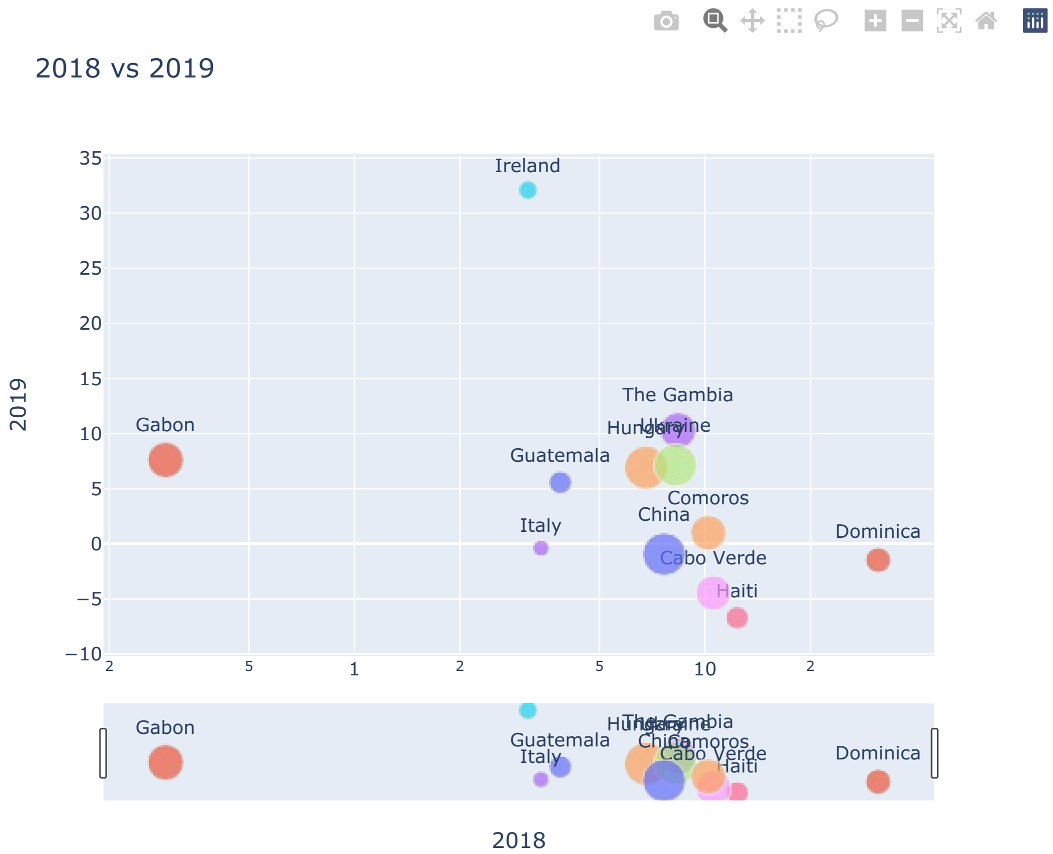



India
2.588
3.465
8.655
9.549
9.549
0
0.5M
1M
India
1.162
2.173
0.607
3.811
3.811
0
0.5M
1M
200k
Year 2024 India
India
167.665
179.742
189.467
0
200
400
600
800
Year 2025 India
India
174.008
187.015
197.126
0
200
400
600
800
150k
100k
50k



In [33]:

```
#scatter plot
top = 15
fig = px.scatter(d1.sort_values('Subject Descriptor',ascending=False).head(top), x='2018',y='2019',color='Country
height=600,text='Country',log_x=True,title='2018 vs 2019 ')
fig.update_traces(textposition='top center')
fig.update_layout(showlegend=False)
fig.update_layout(xaxis_rangeslider_visible=True)
fig.show()
```

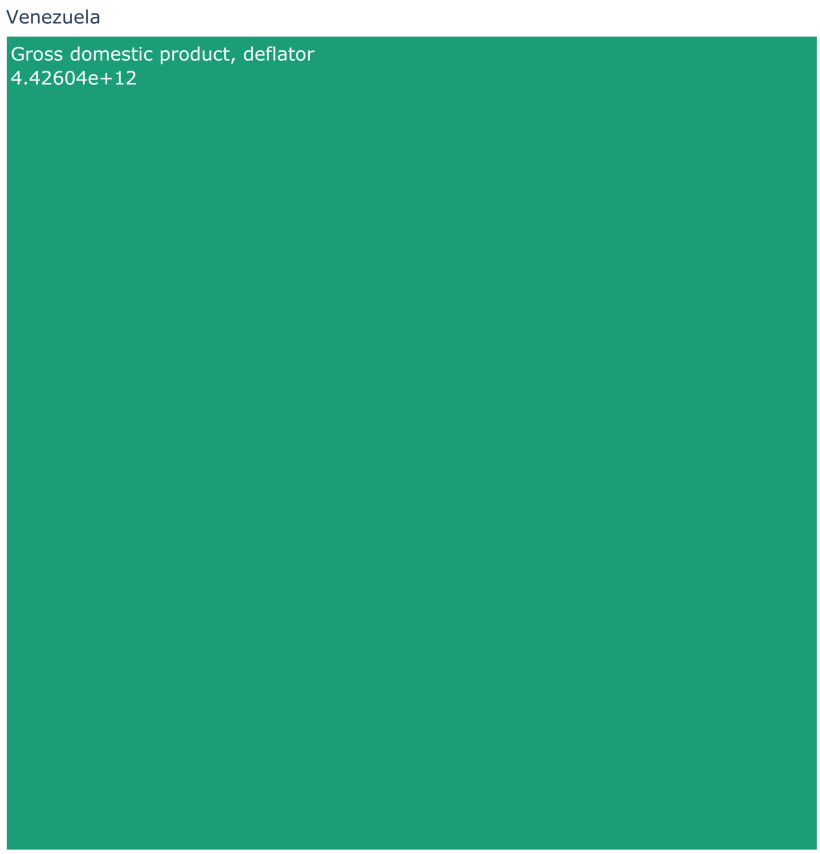


In [34]:

```
#Seaborn barplot
import plotly.express as px
fig=px.bar(d1,x='Country',y='2020',color='Subject Descriptor',orientation='h')
fig.show()
```

In [35]:

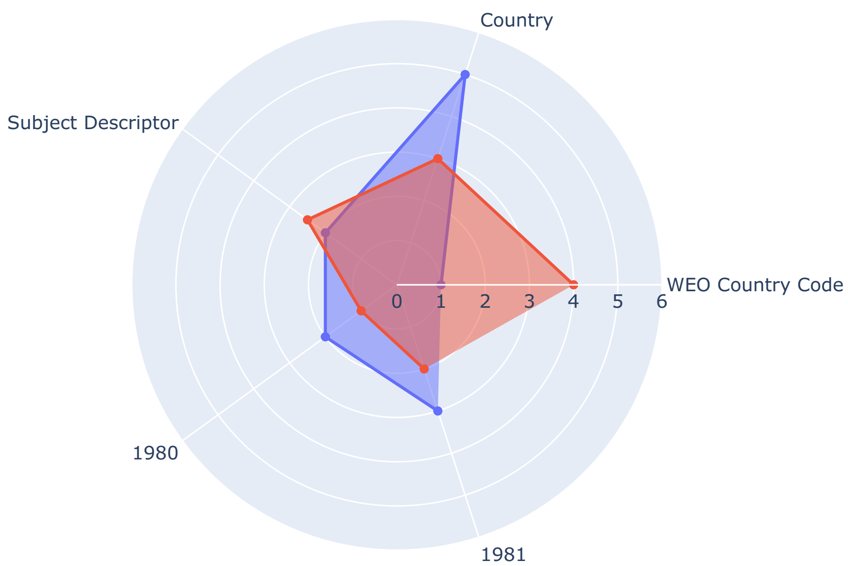
```
#treemap
full_latest = d1[d1['2021']== max(d1['2021'])]

fig = px.treemap(full_latest.sort_values(by='2020',ascending=False).reset_index (drop=True),path=['Country','Subj
height=700,
color_discrete_sequence=px.colors.qualitative.Dark2)
fig.data[0].textinfo = 'label+text+value'
fig.show()
```



In [36]:

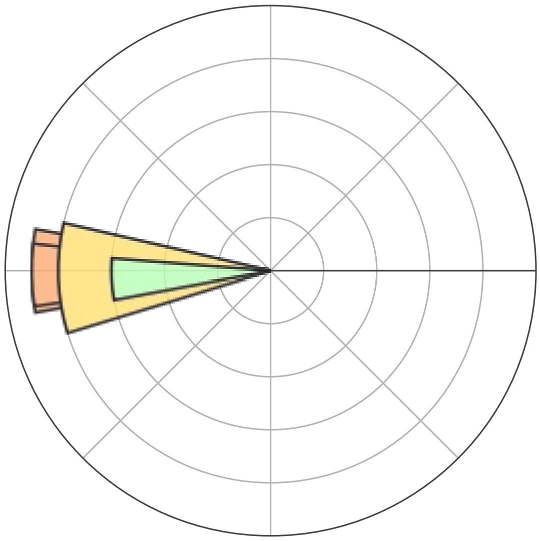
```
import plotly.graph_objects as go
py.offline.init_notebook_mode(connected=True)
fig = go.Figure()
```

```python
categories = d1.columns

fig.add_trace(go.Scatterpolar(
      r=[1, 5, 2, 2, 3],
      theta=categories,
      fill='toself'
))
fig.add_trace(go.Scatterpolar(
      r=[4, 3, 2.5, 1, 2],
      theta=categories,
      fill='toself'
))

fig.update_layout(
  polar=dict(
    radialaxis=dict(
      visible=True,
      range=[0, 6]
    )),
  showlegend=False
)

fig.show()
```



In [37]:

```python
import plotly.graph_objects as go
categories = d1.columns

fig = go.Figure(go.Barpolar(
    r=[3.5, 1.5, 2.5, 4.5, 4.5, 4, 3],
    theta=categories,
    width=[20,15,10,20,15,30,15,],
    marker_color=["#E4FF87", '#709BFF', '#709BFF', '#FFAA70', '#FFAA70', '#FFDF70', '#B6FFB4'],
    marker_line_color="black",
    marker_line_width=2,
    opacity=0.8
))

fig.update_layout(
    template=None,
    polar = dict(
        radialaxis = dict(range=[0, 5], showticklabels=False, ticks=''),
        angularaxis = dict(showticklabels=False, ticks='')
    )
)

fig.show()
```



Feature Engineering

In [38]:
```
d_1=d1.drop(['Country'],axis=1)
```

In [39]:
```
y = d_1['Subject Descriptor']
x = d_1.drop(['Subject Descriptor'], axis=1)
x
```

Out[39]:

	WEO Country Code	1980	1981	1982	1983	1984	1985	1986	1987	1988	...	2017	2018	2019
0	512	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	20.5095	21.0465	21.007
1	914	5.7650	6.671	7.288	7.657	8.0920	8.223	8.858	9.0040	9.191	...	37.6090	40.0800	41.709
2	914	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	103.2950	105.3900	106.878
3	612	2596.3680	2674.259	2845.412	2999.064	3167.0120	3344.364	3337.676	3314.3120	3251.340	...	7364.6750	7467.7800	7527.523
4	612	8.9750	10.286	10.964	11.823	12.5690	13.880	15.824	16.7510	17.746	...	193.9700	202.2530	206.200
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8770	582	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	-1.9640	-1.0230	-3.294
8771	582	-1.5990	-4.197	-2.635	-1.946	-1.6020	-4.951	-3.441	-2.6000	-2.607	...	-0.5960	1.8980	3.417
8772	474	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	839.7510	1651.6000	1745.000
8773	474	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	76.5590	73.7950	75.839
8774	754	2957.2140	3052.876	2871.765	2752.526	2624.5240	2578.899	2546.290	2509.0840	2661.393	...	3407.3060	3438.0800	3383.327

8775 rows × 48 columns

In [40]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
y
```

Out[40]:
```
array([20, 14, 18, ...,  8,  3, 11])
```

In [41]:
```
pd.DataFrame(x)
```

Out[41]:

	WEO Country Code	1980	1981	1982	1983	1984	1985	1986	1987	1988	...	2017	2018	2019
0	512	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	20.5095	21.0465	21.007

1	914	5.7650	6.671	7.288	7.657	8.0920	8.223	8.858	9.0040	9.191	...	37.6090	40.0800	41.709
2	914	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	103.2950	105.3900	106.878
3	612	2596.3680	2674.259	2845.412	2999.064	3167.0120	3344.364	3337.676	3314.3120	3251.340	...	7364.6750	7467.7800	7527.523
4	612	8.9750	10.286	10.964	11.823	12.5690	13.880	15.824	16.7510	17.746	...	193.9700	202.2530	206.200
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8770	582	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	-1.9640	-1.0230	-3.294
8771	582	-1.5990	-4.197	-2.635	-1.946	-1.6020	-4.951	-3.441	-2.6000	-2.607	...	-0.5960	1.8980	3.417
8772	474	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	839.7510	1651.6000	1745.000
8773	474	12.5575	11.680	10.611	10.623	12.1875	11.111	12.311	13.3725	13.014	...	76.5590	73.7950	75.839
8774	754	2957.2140	3052.876	2871.765	2752.526	2624.5240	2578.899	2546.290	2509.0840	2661.393	...	3407.3060	3438.0800	3383.327

8775 rows × 48 columns

In [42]:
```
x.shape
```

Out[42]: (8775, 48)

In [43]:
```
y.shape
```

Out[43]: (8775,)

Train Test split

In [44]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x , y, test_size = 0.2, random_state = 0)
```

Data Modeling

In [45]:
```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators = 20,criterion = 'entropy',max_depth = 20,random_state=2)
rf_classifier.fit(x_train,y_train)
```

Out[45]:
```
RandomForestClassifier(criterion='entropy', max_depth=20, n_estimators=20,
                       random_state=2)
```

In [46]:
```
pred_train = rf_classifier.predict(x_train)
pred_test = rf_classifier.predict(x_test)
```

In [47]:
```
from sklearn.metrics import accuracy_score
print('Training Accuracy',accuracy_score(y_train,pred_train))
print('Testing Accuracy',accuracy_score(y_test,pred_test))
```

```
Training Accuracy 0.8962962962962963
Testing Accuracy 0.392022792022792
```

In [48]:
```
#Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(criterion = 'entropy',max_depth = 10,random_state = 2)
```

In [49]:
```
dt_classifier.fit(x_train,y_train)
```

Out[49]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=2)
```

In [50]:
```
pred_train1 = dt_classifier.predict(x_train)
```

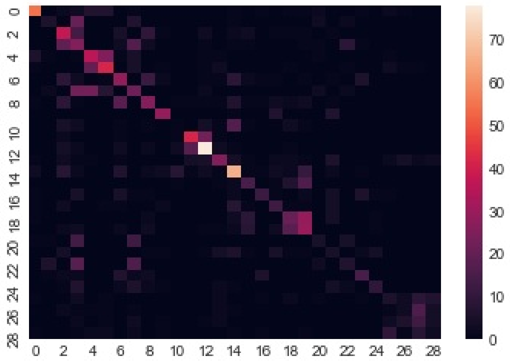
```
pred_test1 = dt_classifier.predict(x_test)
```

In [51]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
print('Training Accuracy',accuracy_score(y_train,pred_train1))
print('Testing Accuracy',accuracy_score(y_test,pred_test1))
cm = confusion_matrix(y_test,pred_test1)
```

```
Training Accuracy 0.594017094017094
Testing Accuracy 0.38746438746438744
```

In [52]:
```
sns.heatmap(cm)
```

Out[52]: <AxesSubplot:>



In [53]:
```
#XGB
import xgboost as xgb
xg_classifier = xgb.XGBClassifier(n_estimators = 30)   #there is also XGBRegressor
xg_classifier.fit(x_train, y_train)
```

```
[19:33:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[53]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=30, n_jobs=8, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [54]:
```
from sklearn.metrics import accuracy_score
pred_train2 = xg_classifier.predict(x_train)
pred_test2 = xg_classifier.predict(x_test)
print('Training Accuracy',accuracy_score(y_train,pred_train2))
print('Testing Accuracy',accuracy_score(y_test,pred_test2))
```

```
Training Accuracy 0.8505698005698006
Testing Accuracy 0.43475783475783475
```

In [55]:
```
#LGB
import lightgbm as lgb
lg_classifier = lgb.LGBMClassifier(n_estimators = 50)
lg_classifier.fit(x_train,y_train)
```

Out[55]: LGBMClassifier(n_estimators=50)

In [56]:
```
pred_train3 = lg_classifier.predict(x_train)
pred_test3 = lg_classifier.predict(x_test)
```

```
print('Training Accuracy',accuracy_score(y_train,pred_train3))
print('Testing Accuracy',accuracy_score(y_test,pred_test3))
```

```
Training Accuracy 0.8914529914529915
Testing Accuracy 0.4182336182336182
```

By Predicting dataset by differnet models we get:

Training Accuracy is approx 0.85

Testing accuracy is approx 0.43

Dataset 2

In [57]:

```
df2
```

Out[57]:

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	196
0	Aruba	ABW	GDP (current US$)	NY.GDP.MKTP.CD	NaN	NaN	NaN	NaN	NaN	Na
1	Africa Eastern and Southern	AFE	GDP (current US$)	NY.GDP.MKTP.CD	1.929193e+10	1.970186e+10	2.147035e+10	2.570500e+10	2.350165e+10	2.678117e+1
2	Afghanistan	AFG	GDP (current US$)	NY.GDP.MKTP.CD	5.377778e+08	5.488889e+08	5.466667e+08	7.511112e+08	8.000000e+08	1.006667e+0
3	Africa Western and Central	AFW	GDP (current US$)	NY.GDP.MKTP.CD	1.040732e+10	1.113130e+10	1.194684e+10	1.268022e+10	1.384262e+10	1.486682e+1
4	Angola	AGO	GDP (current US$)	NY.GDP.MKTP.CD	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	.
527	Kosovo	XKX	GDP growth (annual %)	NY.GDP.MKTP.KD.ZG	NaN	NaN	NaN	NaN	NaN	Na
528	Yemen, Rep.	YEM	GDP growth (annual %)	NY.GDP.MKTP.KD.ZG	NaN	NaN	NaN	NaN	NaN	Na
529	South Africa	ZAF	GDP growth (annual %)	NY.GDP.MKTP.KD.ZG	NaN	3.844751e+00	6.177883e+00	7.373613e+00	7.939782e+00	6.122761e+0
530	Zambia	ZMB	GDP growth (annual %)	NY.GDP.MKTP.KD.ZG	NaN	1.361382e+00	-2.490839e+00	3.272393e+00	1.221405e+01	1.664746e+0
531	Zimbabwe	ZWE	GDP growth (annual %)	NY.GDP.MKTP.KD.ZG	NaN	6.316157e+00	1.434471e+00	6.244345e+00	-1.106172e+00	4.910571e+0

532 rows × 66 columns

Data Preprocessing

In [58]:

```
#Dropping useless Columns
df2.drop(['Indicator Code','Unnamed: 65','Country Code'],axis=1,inplace=True)
```

In [59]:

```
df2.shape
```

Out[59]:

```
(532, 63)
```

In [60]:

```
df2.dtypes
```

```
Out[60]: Country Name       object
         Indicator Name     object
         1960              float64
         1961              float64
         1962              float64
                            ...
         2016              float64
         2017              float64
         2018              float64
         2019              float64
         2020              float64
         Length: 63, dtype: object
```

```
In [61]: df2.describe()
```

Out[61]:

	1960	1961	1962	1963	1964	1965	1966	1967	1968
count	1.330000e+02	2.590000e+02	2.650000e+02	2.650000e+02	2.650000e+02	2.740000e+02	2.860000e+02	2.940000e+02	2.990000e+02
mean	7.101082e+10	3.742474e+10	3.920584e+10	4.223544e+10	4.639881e+10	4.901361e+10	5.505793e+10	5.680852e+10	6.031218e+10
std	2.130243e+11	1.630170e+11	1.732999e+11	1.863664e+11	2.039176e+11	2.183232e+11	2.380692e+11	2.507789e+11	2.688567e+11
min	1.201201e+07	-2.727000e+01	-1.968504e+01	-1.227866e+01	-1.246499e+01	-1.248183e+01	-7.659066e+00	-1.574363e+01	-5.474906e+00
25%	5.083344e+08	4.424307e+00	5.216059e+00	5.397028e+00	6.589260e+00	5.451860e+00	4.666654e+00	4.573257e+00	5.905051e+00
50%	3.193200e+09	3.290234e+07	3.185692e+07	3.374941e+07	3.619383e+07	9.215942e+07	6.223350e+07	5.895363e+07	6.751429e+07
75%	2.925265e+10	3.278008e+09	3.668222e+09	3.988785e+09	4.235608e+09	4.302679e+09	4.984706e+09	5.116329e+09	5.468490e+09
max	1.384628e+12	1.440342e+12	1.545697e+12	1.665141e+12	1.824117e+12	1.986368e+12	2.155141e+12	2.294159e+12	2.476490e+12

8 rows × 61 columns

```
In [62]: df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 532 entries, 0 to 531
Data columns (total 63 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Country Name    532 non-null    object 
 1   Indicator Name  532 non-null    object 
 2   1960            133 non-null    float64
 3   1961            259 non-null    float64
 4   1962            265 non-null    float64
 5   1963            265 non-null    float64
 6   1964            265 non-null    float64
 7   1965            274 non-null    float64
 8   1966            286 non-null    float64
 9   1967            294 non-null    float64
 10  1968            299 non-null    float64
 11  1969            302 non-null    float64
 12  1970            307 non-null    float64
 13  1971            322 non-null    float64
 14  1972            322 non-null    float64
 15  1973            322 non-null    float64
 16  1974            323 non-null    float64
 17  1975            328 non-null    float64
 18  1976            333 non-null    float64
 19  1977            338 non-null    float64
 20  1978            343 non-null    float64
 21  1979            344 non-null    float64
 22  1980            357 non-null    float64
 23  1981            374 non-null    float64
 24  1982            380 non-null    float64
 25  1983            384 non-null    float64
 26  1984            385 non-null    float64
 27  1985            390 non-null    float64
 28  1986            393 non-null    float64
 29  1987            401 non-null    float64
 30  1988            409 non-null    float64
 31  1989            411 non-null    float64
 32  1990            429 non-null    float64
 33  1991            434 non-null    float64
 34  1992            439 non-null    float64
 35  1993            447 non-null    float64
 36  1994            452 non-null    float64
```

```
 37  1995           463 non-null    float64
 38  1996           473 non-null    float64
 39  1997           473 non-null    float64
 40  1998           479 non-null    float64
 41  1999           482 non-null    float64
 42  2000           488 non-null    float64
 43  2001           493 non-null    float64
 44  2002           499 non-null    float64
 45  2003           504 non-null    float64
 46  2004           504 non-null    float64
 47  2005           505 non-null    float64
 48  2006           506 non-null    float64
 49  2007           507 non-null    float64
 50  2008           504 non-null    float64
 51  2009           505 non-null    float64
 52  2010           506 non-null    float64
 53  2011           508 non-null    float64
 54  2012           507 non-null    float64
 55  2013           508 non-null    float64
 56  2014           510 non-null    float64
 57  2015           508 non-null    float64
 58  2016           506 non-null    float64
 59  2017           506 non-null    float64
 60  2018           505 non-null    float64
 61  2019           493 non-null    float64
 62  2020           449 non-null    float64
dtypes: float64(61), object(2)
memory usage: 262.0+ KB
```

Filling null values

In [63]:

```
df2.isnull().sum()
```

Out[63]:

```
Country Name        0
Indicator Name      0
1960              399
1961              273
1962              267
                 ...
2016               26
2017               26
2018               27
2019               39
2020               83
Length: 63, dtype: int64
```

In [68]:

```
X1 = df2.iloc[:, 0:2]
Y1 = df2.iloc[: , 2:65]
```

In [67]:

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan,strategy='median')
dff = imp.fit_transform(Y1)
df2_new=pd.DataFrame(dff , columns = Y1.columns)
```

In [66]:

```
df2_gdp = pd.concat([X1, df2_new],axis=1)
df2_gdp
```

Out[66]:

	Country Name	Indicator Name	1960	1961	1962	1963	1964	1965	1966	1967
0	Aruba	GDP (current US$)	3.193200e+09	3.290234e+07	3.185692e+07	3.374941e+07	3.619383e+07	9.215942e+07	6.223350e+07	5.895363e+07
1	Africa Eastern and Southern	GDP (current US$)	1.929193e+10	1.970186e+10	2.147035e+10	2.570500e+10	2.350165e+10	2.678117e+10	2.912019e+10	3.014009e+10
2	Afghanistan	GDP (current US$)	5.377778e+08	5.488889e+08	5.466667e+08	7.511112e+08	8.000000e+08	1.006667e+09	1.400000e+09	1.673333e+09
3	Africa Western and Central	GDP (current US$)	1.040732e+10	1.113130e+10	1.194684e+10	1.268022e+10	1.384262e+10	1.486682e+10	1.583747e+10	1.443065e+10
		GDP								

4	Angola	(current US$)	3.193200e+09	3.290234e+07	3.185692e+07	3.374941e+07	3.619383e+07	9.215942e+07	6.223350e+07	5.895363e+07
...	...	...	...	...	...	...	...	...	...	...
527	Kosovo	GDP growth (annual %)	3.193200e+09	3.290234e+07	3.185692e+07	3.374941e+07	3.619383e+07	9.215942e+07	6.223350e+07	5.895363e+07
528	Yemen, Rep.	GDP growth (annual %)	3.193200e+09	3.290234e+07	3.185692e+07	3.374941e+07	3.619383e+07	9.215942e+07	6.223350e+07	5.895363e+07
529	South Africa	GDP growth (annual %)	3.193200e+09	3.844751e+00	6.177883e+00	7.373613e+00	7.939782e+00	6.122761e+00	4.438308e+00	7.196576e+00
530	Zambia	GDP growth (annual %)	3.193200e+09	1.361382e+00	-2.490839e+00	3.272393e+00	1.221405e+01	1.664746e+01	-5.570310e+00	7.919697e+00
531	Zimbabwe	GDP growth (annual %)	3.193200e+09	6.316157e+00	1.434471e+00	6.244345e+00	-1.106172e+00	4.910571e+00	1.523130e+00	8.367009e+00

532 rows × 63 columns

In [69]:

```
df2_gdp.isnull().sum()
```

Out[69]:

```
Country Name      0
Indicator Name    0
1960              0
1961              0
1962              0
                 ..
2016              0
2017              0
2018              0
2019              0
2020              0
Length: 63, dtype: int64
```

In [70]:

```
df2_gdp.corr()
```

Out[70]:

	1960	1961	1962	1963	1964	1965	1966	1967	1968	1969	...	2011	2012	2013	
1960	1.000000	0.999536	0.999136	0.998990	0.999175	0.999274	0.974744	0.973547	0.974326	0.974054	...	0.945230	0.943191	0.939025	0.9
1961	0.999536	1.000000	0.999886	0.999839	0.999814	0.999801	0.975445	0.974567	0.975526	0.975175	...	0.940658	0.937903	0.933217	0.9
1962	0.999136	0.999886	1.000000	0.999948	0.999855	0.999794	0.975561	0.974786	0.975813	0.975433	...	0.938551	0.935475	0.930597	0.9
1963	0.998990	0.999839	0.999948	1.000000	0.999917	0.999865	0.975620	0.974902	0.975921	0.975577	...	0.939092	0.935930	0.931006	0.9
1964	0.999175	0.999814	0.999855	0.999917	1.000000	0.999984	0.975603	0.974822	0.975774	0.975505	...	0.941702	0.938710	0.933895	0.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2016	0.943365	0.937346	0.934467	0.934793	0.937639	0.938972	0.952692	0.949853	0.948174	0.949865	...	0.996767	0.998502	0.998884	0.9
2017	0.938306	0.931777	0.928721	0.929021	0.932046	0.933455	0.946846	0.943784	0.941940	0.943720	...	0.995633	0.997865	0.998651	0.9
2018	0.936001	0.929351	0.926213	0.926552	0.929599	0.931058	0.945002	0.941912	0.940009	0.941812	...	0.994977	0.997325	0.998267	0.9
2019	0.935797	0.928978	0.925745	0.926047	0.929109	0.930592	0.943242	0.940059	0.938166	0.939955	...	0.994026	0.996677	0.997718	0.9
2020	0.939081	0.932269	0.929379	0.929308	0.931823	0.933096	0.945912	0.942433	0.940185	0.941400	...	0.980985	0.982467	0.984386	0.9

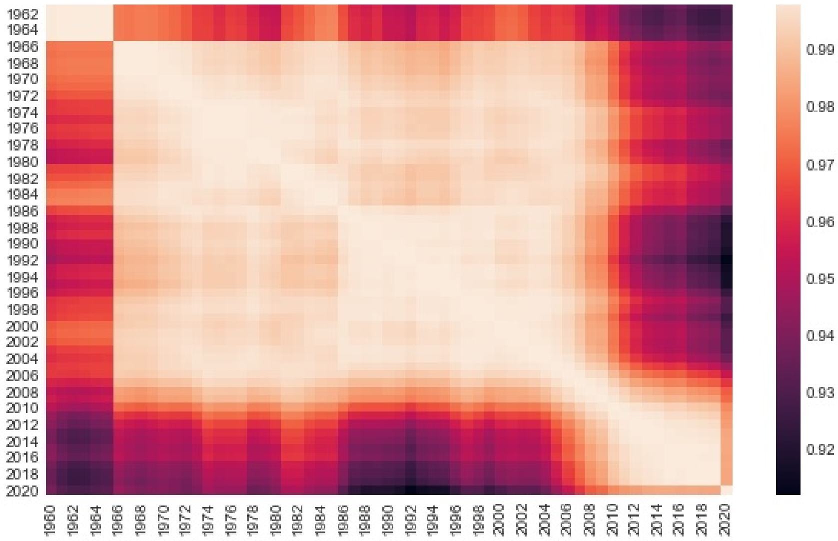
61 rows × 61 columns

Data Visualisation

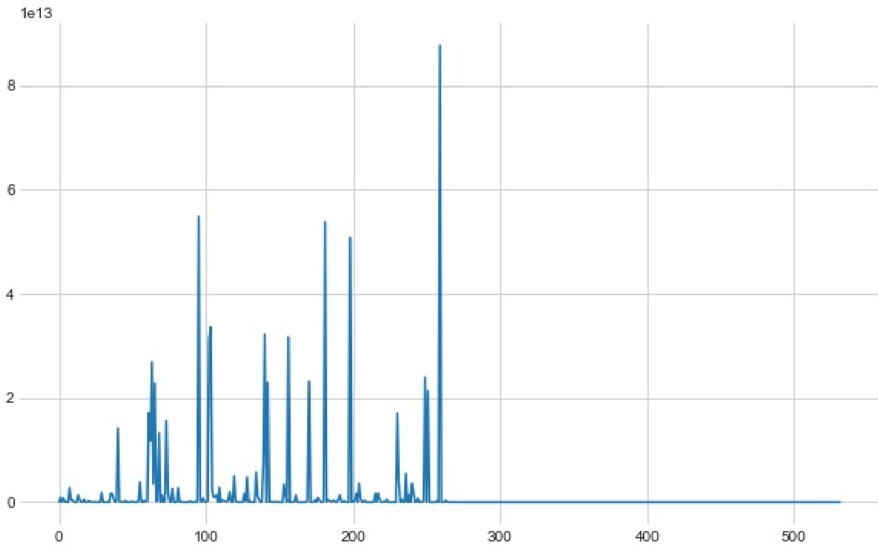
In [71]:

```
#heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df2_gdp.corr())
```

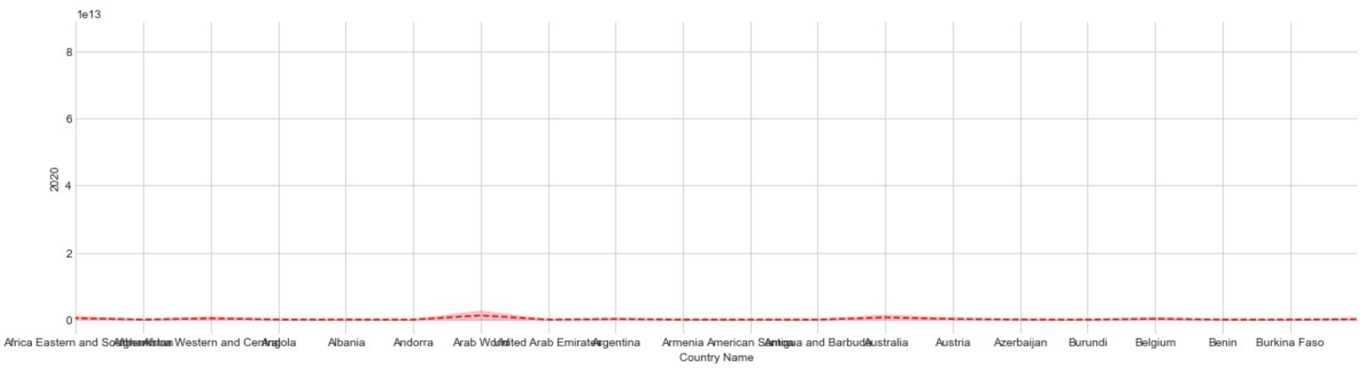
Out[71]: <AxesSubplot:>



In [72]:
```
plt.figure(figsize = (10,6))
df2_gdp['2019'].plot()
plt.show()
```

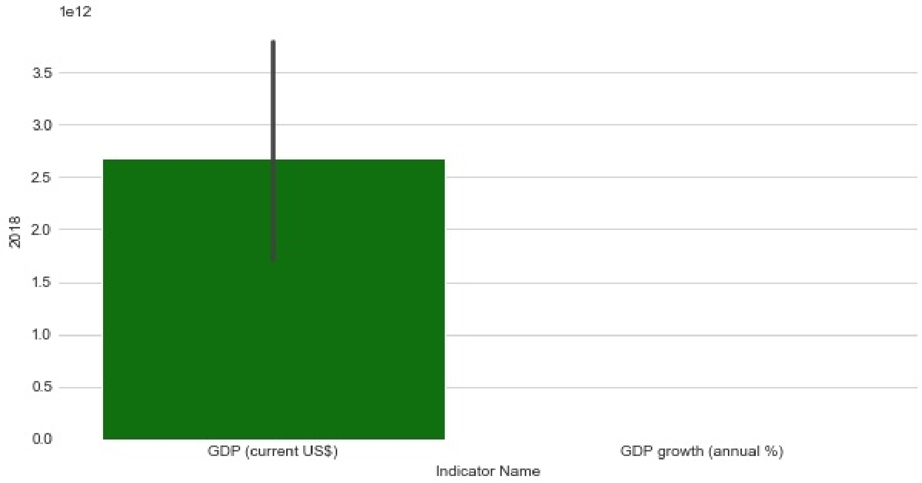


In [73]:
```
#lineplot
fig,ax=plt.subplots(figsize=(20,5))
sns.lineplot(df2_gdp['Country Name'], df2_gdp['2020'],ax=ax,linestyle = 'dashed',color = 'r')
ax.set_xlim(1,20)
ax.set_xticks(range(1,20))
plt.show()
```



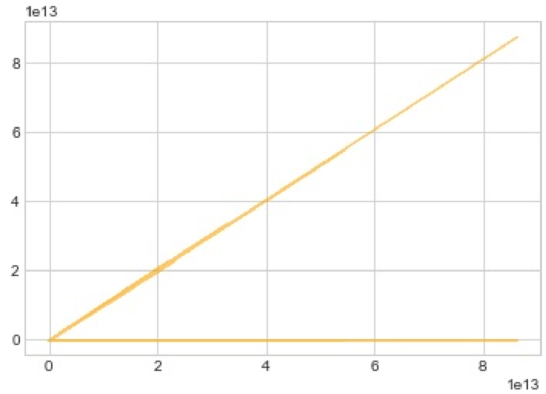
In [74]:
```
#barplot
fig = plt.figure(figsize = (10, 5))
sns.barplot(df2_gdp['Indicator Name'],df2_gdp['2018'], color ='green')
```

Out[74]: <AxesSubplot:xlabel='Indicator Name', ylabel='2018'>



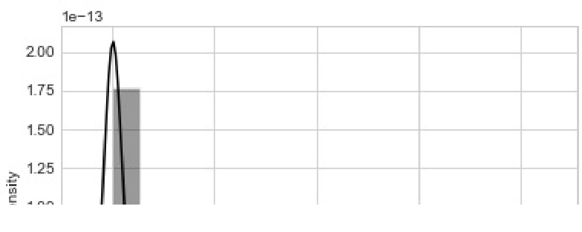
In [75]:
```
#scatterplot
plt.scatter(x=df2_gdp['Indicator Name'], y=df2_gdp['1990'])
plt.show()
```

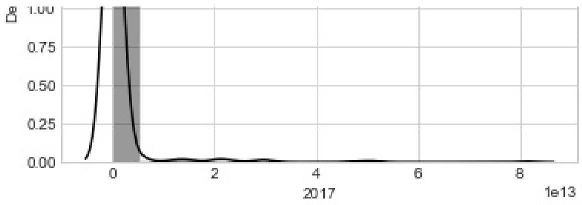
In [76]:
```
#Area chart
plt.style.use('seaborn-whitegrid')
plt.fill_between(x='2018', y1='2019',color="orange",alpha=0.6,data=df2_gdp)
plt.show()
```



In [79]:
```
#distribution plot
sns.set_style('whitegrid')
sns.distplot(df2_gdp['2017'], color ='black', bins = 15)
```

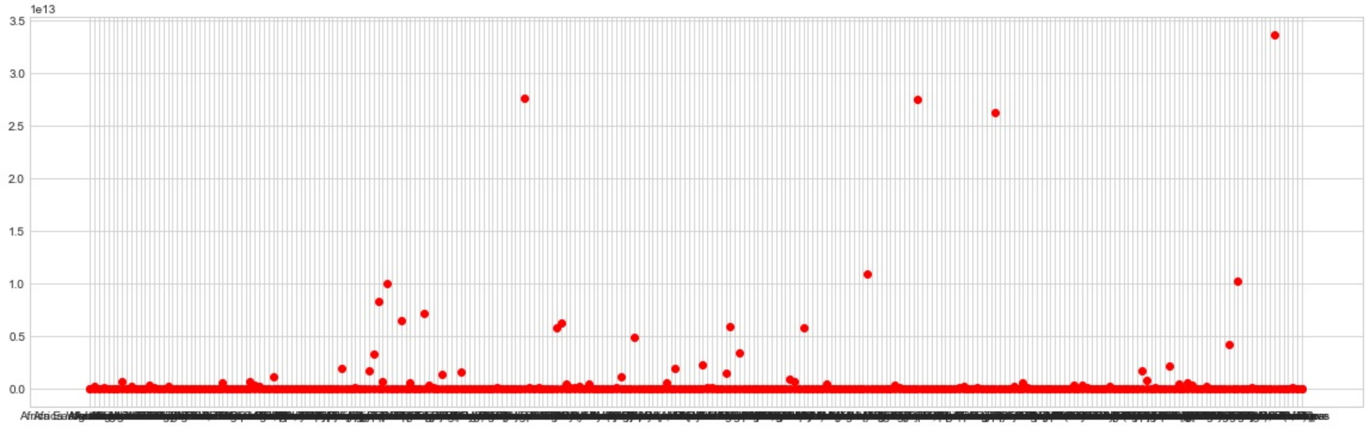
Out[79]: <AxesSubplot:xlabel='2017', ylabel='Density'>





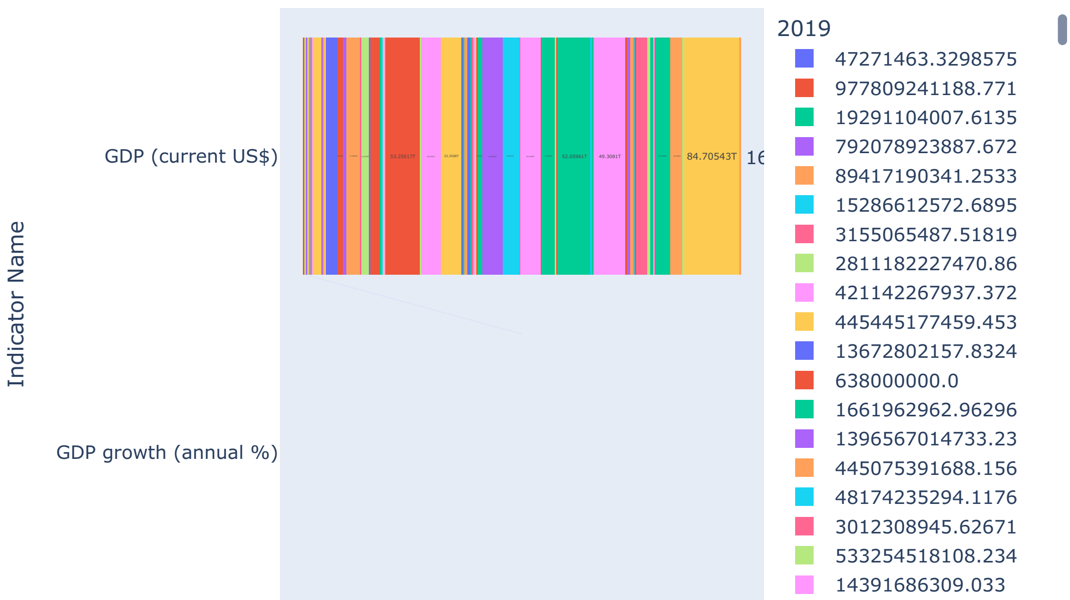
In [80]:

```
fig,ax=plt.subplots(figsize=(20,6))
plt.scatter(x=df2_gdp['Country Name'], y=df2_gdp['2000'],color="red")
plt.show()
```



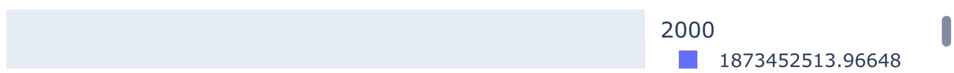
In [81]:

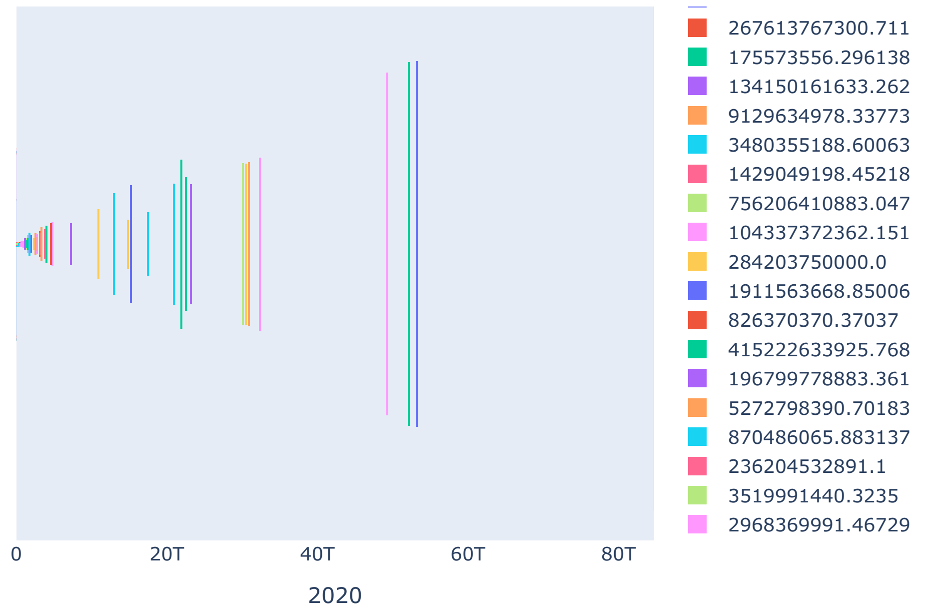
```
#funnel chart
import plotly.express as px
fig = px.funnel(df2_gdp, x = '2020',y = 'Indicator Name',color='2019')
fig.show()
```



In [82]:

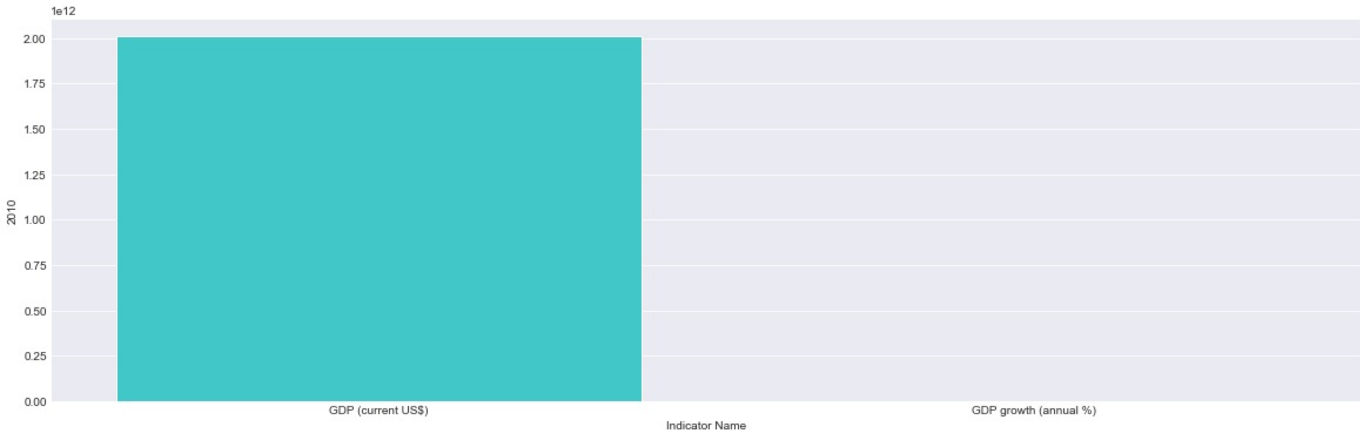
```
import plotly.express as px
fig = px.funnel(df2_gdp, x = '2020',y = '2010',color='2000')
fig.show()
```





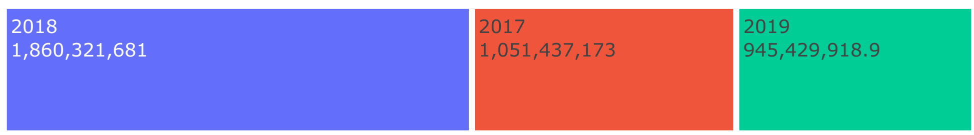
In [83]:

```python
#Seaborn barplot
sns.set_style("darkgrid")
bar,ax = plt.subplots(figsize=(20,6))
ax = sns.barplot(x='Indicator Name', y='2010', data=df2_gdp, ci=None, palette="rainbow",orient='v')
```



In [84]:

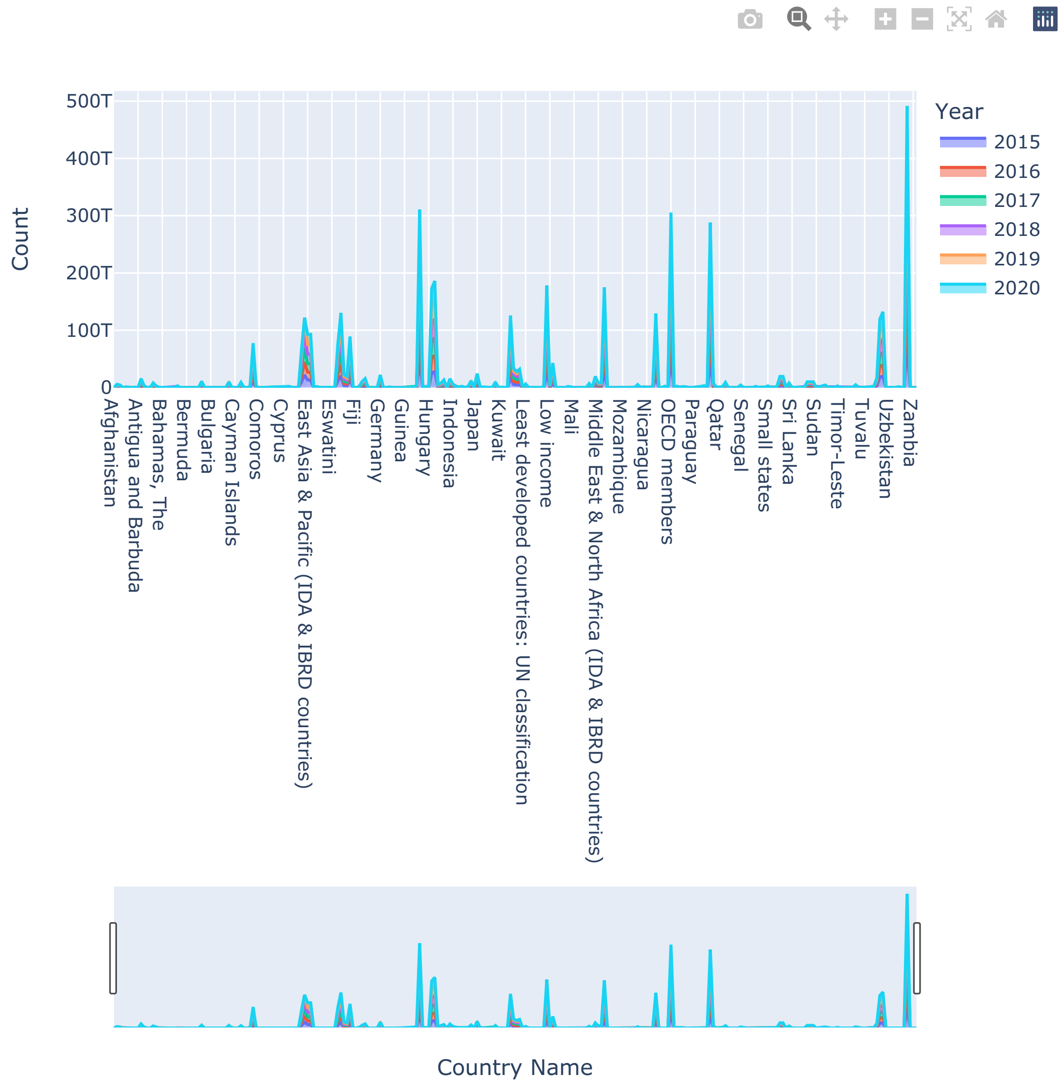
```python
import plotly.express as px
temp = df2_gdp.groupby('Indicator Name')['2017','2018','2019','2020'].sum().reset_index()
temp = temp[temp['Indicator Name']==max(temp['Indicator Name'])].reset_index(drop = True)
#melt plot
tm = temp.melt(id_vars = 'Indicator Name', value_vars = ['2017','2018','2019','2020'])
fig = px.treemap(tm, path = ['variable'], values = 'value', height = 250, width = 800)
fig.data[0].textinfo = 'label+text+value'
fig.show()
```



In [89]:

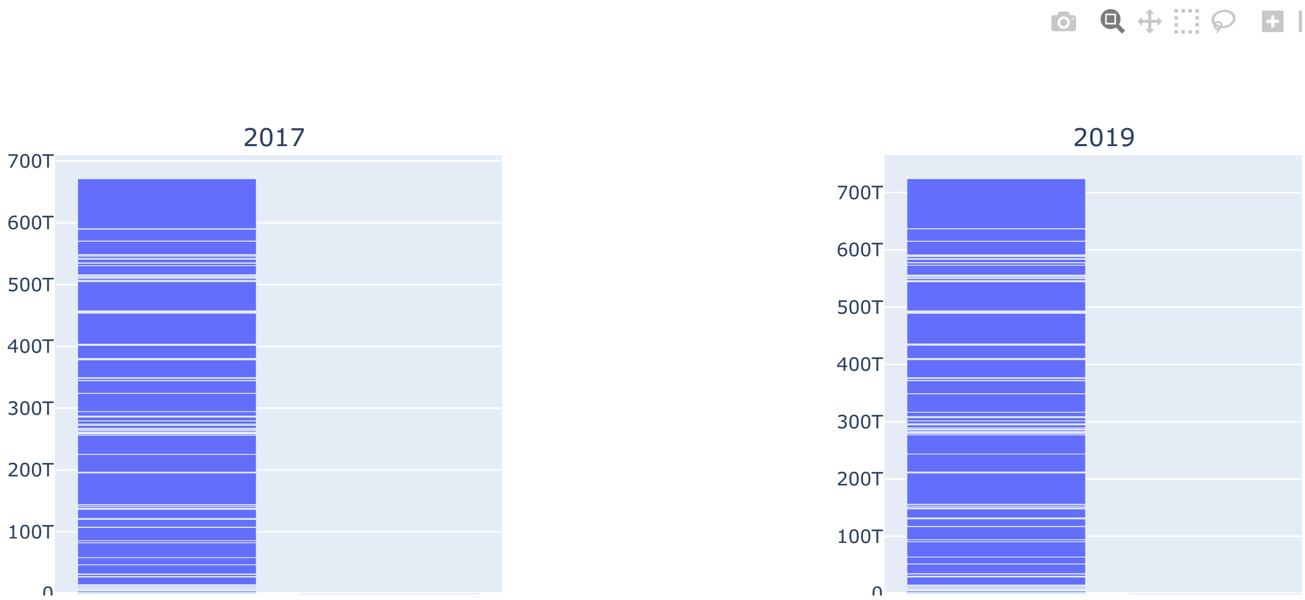
```python
temp = df2_gdp.groupby('Country Name')['2015','2016','2017','2018','2019','2020'].sum().reset_index()
temp = temp.melt(id_vars = 'Country Name',value_vars = ['2015','2016','2017','2018','2019','2020'], var_name = '
fig = px.area(temp,x='Country Name',y='Count',color='Year',height = 750)
```

```
fig.update_layout(xaxis_rangeslider_visible=True)
fig.show()
```



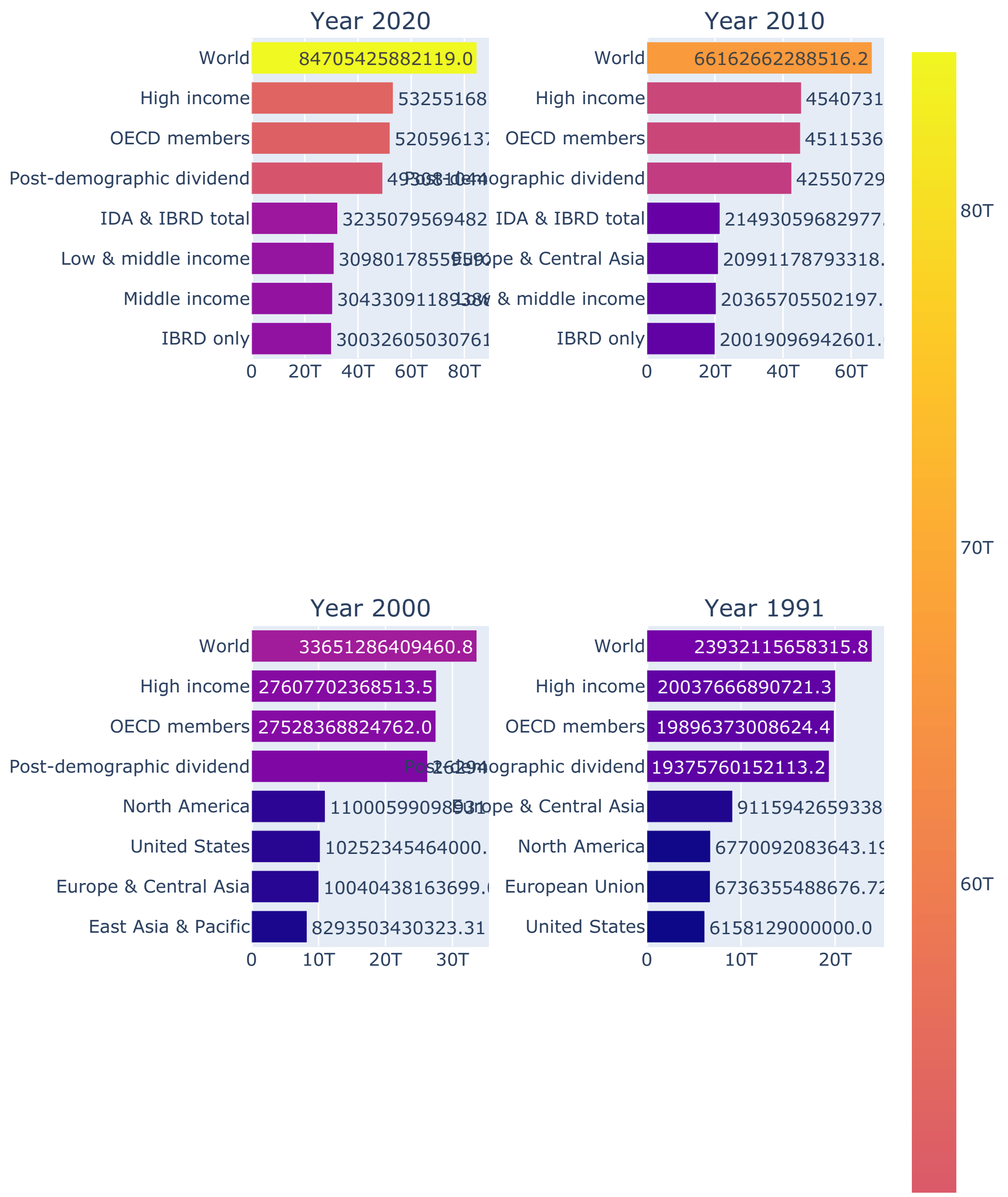
In [90]:

```
from plotly.subplots import make_subplots
fig_c = px.bar(df2_gdp,x='Indicator Name',y='2017')
fig_d = px.bar(df2_gdp,x='Indicator Name',y='2019')

fig = make_subplots(rows=1,cols=2,shared_xaxes=False, horizontal_spacing=0.3,vertical_spacing=0.5,
subplot_titles=('2017 ','2019 '))
fig.add_trace(fig_c['data'][0],row=1,col=1)
fig.add_trace(fig_d['data'][0],row=1,col=2)
fig.update_layout(autosize=False,width=990,height=480)
fig.show()
```



In [91]:

```
top=8
fig_c = px.bar(df2_gdp.sort_values('2020').tail(top),x = '2020'
               ,y='Country Name',text='2020',orientation='h', color='2020')

fig_a = px.bar(df2_gdp.sort_values('2010').tail(top),x = '2010'
               ,y='Country Name',text='2010',orientation='h', color='2010')

fig_dc = px.bar(df2_gdp.sort_values('2000').tail(top),x = '2000'
               ,y='Country Name',text='2000',orientation='h', color='2000')

fig_rc = px.bar(df2_gdp.sort_values('1991').tail(top),x = '1991'
,y='Country Name',text='1991',orientation='h', color='1991')

fig = make_subplots(rows=5,cols=2,shared_xaxes=False,horizontal_spacing=0.25, vertical_spacing=.1,
                    subplot_titles=('Year 2020','Year 2010',
                                    'Year 2000','Year 1991'))

fig.add_trace(fig_c['data'][0],row=1,col=1)
fig.add_trace(fig_a['data'][0],row=1,col=2)
fig.add_trace(fig_dc['data'][0],row=2,col=1)
fig.add_trace(fig_rc['data'][0],row=2,col=2)
fig.update_layout(height=2000)
fig.show()
```



Feature Engineering

In [92]:

```python
d_f_2= df2_gdp.drop(['Country Name'],axis=1)
```

In [93]:

```python
y1 = d_f_2['Indicator Name']
x1 = d_f_2.drop(['Indicator Name'], axis=1)
x1
y1
```

Out[93]:

```
0          GDP (current US$)
1          GDP (current US$)
2          GDP (current US$)
```

```
3          GDP (current US$)
4          GDP (current US$)
               ...
527    GDP growth (annual %)
528    GDP growth (annual %)
529    GDP growth (annual %)
530    GDP growth (annual %)
531    GDP growth (annual %)
Name: Indicator Name, Length: 532, dtype: object
```

In [94]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y1 = le.fit_transform(y1)
```

Train Test Split

In [95]:
```python
from sklearn.model_selection import train_test_split
x1_train, x1_test, y1_train, y1_test = train_test_split(x1 , y1, test_size = 0.2, random_state = 0)
```

Data Modeling

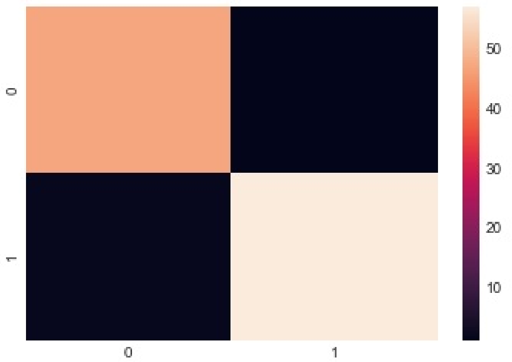
In [97]:
```python
from sklearn.ensemble import RandomForestClassifier
rf1_classifier = RandomForestClassifier(n_estimators = 20,criterion = 'entropy', max_depth = 20, random_state = 2
rf1_classifier.fit(x1_train, y1_train)

train_pred = rf1_classifier.predict(x1_train)
test_pred = rf1_classifier.predict(x1_test)

from sklearn.metrics import accuracy_score
print('Training Accuracy',accuracy_score(y1_train,train_pred))
print('Testing Accuracy',accuracy_score(y1_test,test_pred))
```

```
Training Accuracy 0.9905882352941177
Testing Accuracy 0.9719626168224299
```

In [98]:
```python
from sklearn.tree import DecisionTreeClassifier
dt1_classifier = DecisionTreeClassifier(criterion = 'entropy',max_depth = 10,random_state = 2)

dt1_classifier.fit(x1_train,y1_train)

train_pred1 = dt1_classifier.predict(x1_train)
test_pred1 = dt1_classifier.predict(x1_test)

from sklearn.metrics import accuracy_score,confusion_matrix
print('Training Accuracy',accuracy_score(y1_train,train_pred1))
print('Testing Accuracy',accuracy_score(y1_test,test_pred1))
cm1 = confusion_matrix(y1_test,test_pred1)

sns.heatmap(cm1)
```

```
Training Accuracy 0.9905882352941177
Testing Accuracy 0.9719626168224299
```

Out[98]: <AxesSubplot:>



In [99]:
```python
import xgboost as xgb
```

```
import xgboost as xgb
xg1_classifier = xgb.XGBClassifier(n_estimators = 30)   #there is also XGBRegressor
xg1_classifier.fit(x1_train, y1_train)

from sklearn.metrics import accuracy_score
train_pred2 = xg1_classifier.predict(x1_train)
test_pred2 = xg1_classifier.predict(x1_test)
print('Training Accuracy',accuracy_score(y1_train,train_pred2))
print('Testing Accuracy',accuracy_score(y1_test,test_pred2))
```

```
[19:46:31] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Training Accuracy 0.9905882352941177
Testing Accuracy 0.9719626168224299
```

In [100…

```
import lightgbm as lgb
lg1_classifier = lgb.LGBMClassifier(n_estimators = 50)
lg1_classifier.fit(x1_train,y1_train)

train_pred3 = lg1_classifier.predict(x1_train)
test_pred3 = lg1_classifier.predict(x1_test)
print('Training Accuracy',accuracy_score(y1_train,train_pred3))
print('Testing Accuracy',accuracy_score(y1_test,test_pred3))
```

```
Training Accuracy 0.9905882352941177
Testing Accuracy 0.9719626168224299
```

By Training and tesint the data using different model , result is:

Training accuracy : approx(0.99)

Testing accuracy : approx(0.97)