In [1]:

```
!pip install factor_analyzer
```

```
Requirement already satisfied: factor_analyzer in /opt/anaconda3/lib/p
ython3.9/site-packages (0.4.0)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/pyth
on3.9/site-packages (from factor_analyzer) (1.0.2)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.9/s
ite-packages (from factor_analyzer) (1.20.3)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.9/s
ite-packages (from factor_analyzer) (1.7.1)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.9/
site-packages (from factor_analyzer) (1.3.4)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/anaconda
3/lib/python3.9/site-packages (from pandas->factor_analyzer) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/pyth
on3.9/site-packages (from pandas->factor_analyzer) (2021.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.
9/site-packages (from python-dateutil>=2.7.3->pandas->factor_analyzer)
(1.16.0)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/pyth
on3.9/site-packages (from scikit-learn->factor_analyzer) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/
lib/python3.9/site-packages (from scikit-learn->factor_analyzer) (2.2.
0)
```

In [2]:

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
plt.rcParams['figure.figsize'] = (20.0, 10.0)
plt.rcParams['font.family'] = "serif"

import scipy.optimize as opt
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
idx = pd.IndexSlice
```

DATA PREPARATION

In [4]:

```
df = pd.read_csv("https://raw.githubusercontent.com/owid/co2-data/master/owid-co2-da
df
```

Out[4]:

	iso_code	country	year	co2	co2_per_capita	trade_co2	cement_co2	cement_co2
0	AFG	Afghanistan	1949	0.015	0.002	NaN	NaN	
1	AFG	Afghanistan	1950	0.084	0.011	NaN	NaN	
2	AFG	Afghanistan	1951	0.092	0.012	NaN	NaN	
3	AFG	Afghanistan	1952	0.092	0.012	NaN	NaN	
4	AFG	Afghanistan	1953	0.106	0.013	NaN	NaN	
...	...	...	...	...	...	...	...	
25984	ZWE	Zimbabwe	2016	10.738	0.765	1.415	0.639	
25985	ZWE	Zimbabwe	2017	9.582	0.673	1.666	0.678	
25986	ZWE	Zimbabwe	2018	11.854	0.821	1.308	0.697	
25987	ZWE	Zimbabwe	2019	10.949	0.748	1.473	0.697	
25988	ZWE	Zimbabwe	2020	10.531	0.709	NaN	0.697	

25989 rows × 60 columns

In [5]:

```
df.columns
```

Out[5]:

```
Index(['iso_code', 'country', 'year', 'co2', 'co2_per_capita', 'trade_
co2',
       'cement_co2', 'cement_co2_per_capita', 'coal_co2',
       'coal_co2_per_capita', 'flaring_co2', 'flaring_co2_per_capita',
       'gas_co2', 'gas_co2_per_capita', 'oil_co2', 'oil_co2_per_capit
a',
       'other_industry_co2', 'other_co2_per_capita', 'co2_growth_prc
t',
       'co2_growth_abs', 'co2_per_gdp', 'co2_per_unit_energy',
       'consumption_co2', 'consumption_co2_per_capita',
       'consumption_co2_per_gdp', 'cumulative_co2', 'cumulative_cement
_co2',
       'cumulative_coal_co2', 'cumulative_flaring_co2', 'cumulative_ga
s_co2',
       'cumulative_oil_co2', 'cumulative_other_co2', 'trade_co2_shar
e',
       'share_global_co2', 'share_global_cement_co2', 'share_global_co
al_co2',
       'share_global_flaring_co2', 'share_global_gas_co2',
       'share_global_oil_co2', 'share_global_other_co2',
       'share_global_cumulative_co2', 'share_global_cumulative_cement_
co2',
       'share_global_cumulative_coal_co2',
       'share_global_cumulative_flaring_co2',
       'share_global_cumulative_gas_co2', 'share_global_cumulative_oil
_co2',
       'share_global_cumulative_other_co2', 'total_ghg', 'ghg_per_capi
ta',
       'total_ghg_excluding_lucf', 'ghg_excluding_lucf_per_capita', 'm
ethane',
       'methane_per_capita', 'nitrous_oxide', 'nitrous_oxide_per_capit
a',
       'population', 'gdp', 'primary_energy_consumption', 'energy_per_
capita',
       'energy_per_gdp'],
      dtype='object')
```

In [6]:

```
# checking is there any NaN present in Dataframe
df.isna().any()
```

Out[6]:

```
iso_code                                 True
country                                 False
year                                    False
co2                                      True
co2_per_capita                           True
trade_co2                                True
cement_co2                               True
cement_co2_per_capita                    True
coal_co2                                 True
coal_co2_per_capita                      True
flaring_co2                              True
flaring_co2_per_capita                   True
gas_co2                                  True
gas_co2_per_capita                       True
oil_co2                                  True
oil_co2_per_capita                       True
other_industry_co2                       True
other_co2_per_capita                     True
co2_growth_prct                          True
co2_growth_abs                           True
co2_per_gdp                              True
co2_per_unit_energy                      True
consumption_co2                          True
consumption_co2_per_capita               True
consumption_co2_per_gdp                  True
cumulative_co2                           True
cumulative_cement_co2                    True
cumulative_coal_co2                      True
cumulative_flaring_co2                   True
cumulative_gas_co2                       True
cumulative_oil_co2                       True
cumulative_other_co2                     True
trade_co2_share                          True
share_global_co2                         True
share_global_cement_co2                  True
share_global_coal_co2                    True
share_global_flaring_co2                 True
share_global_gas_co2                     True
share_global_oil_co2                     True
share_global_other_co2                   True
share_global_cumulative_co2              True
share_global_cumulative_cement_co2       True
share_global_cumulative_coal_co2         True
share_global_cumulative_flaring_co2      True
share_global_cumulative_gas_co2          True
share_global_cumulative_oil_co2          True
share_global_cumulative_other_co2        True
total_ghg                                True
ghg_per_capita                           True
total_ghg_excluding_lucf                 True
ghg_excluding_lucf_per_capita            True
methane                                  True
methane_per_capita                       True
nitrous_oxide                            True
```

```
nitrous_oxide_per_capita          True
population                        True
gdp                               True
primary_energy_consumption        True
energy_per_capita                 True
energy_per_gdp                    True
dtype: bool
```

In [7]:

```
#df.isna()
```

In [8]:

```
df = df.replace(np.nan, 0)
df = df.replace(np.inf, 0)
df = df.dropna()
```

In [9]:

```
# some of the country values are NaN so filling up with mean values of the column
df.fillna(df.mean()).head()
```

Out[9]:

	iso_code	country	year	co2	co2_per_capita	trade_co2	cement_co2	cement_co2_per_c
0	AFG	Afghanistan	1949	0.015	0.002	0.0	0.0	
1	AFG	Afghanistan	1950	0.084	0.011	0.0	0.0	
2	AFG	Afghanistan	1951	0.092	0.012	0.0	0.0	
3	AFG	Afghanistan	1952	0.092	0.012	0.0	0.0	
4	AFG	Afghanistan	1953	0.106	0.013	0.0	0.0	

5 rows × 60 columns

In [10]:

```
def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)
```

DATA PREPROCESSING

In [11]:

```
# Filling NAs with 0s and creating GDP per capita column
df = df.fillna(0)
# CALCULATING THE GDP PER POPULATION FOR not NA / 0 Values
df['gdp_per_capita'] = np.where(df['population'] != 0, df['gdp'] / df['population'],
df['gdp_per_capita']
```

Out[11]:

```
0           0.000000
1        1215.332543
2        1236.236369
3        1262.264378
4        1322.255925
            ...
25984    1494.033433
25985    1541.648846
25986    1573.214825
25987       0.000000
25988       0.000000
Name: gdp_per_capita, Length: 25989, dtype: float64
```

In [12]:

```
df.head()
```

Out[12]:

	iso_code	country	year	co2	co2_per_capita	trade_co2	cement_co2	cement_co2_per_c
0	AFG	Afghanistan	1949	0.015	0.002	0.0	0.0	
1	AFG	Afghanistan	1950	0.084	0.011	0.0	0.0	
2	AFG	Afghanistan	1951	0.092	0.012	0.0	0.0	
3	AFG	Afghanistan	1952	0.092	0.012	0.0	0.0	
4	AFG	Afghanistan	1953	0.106	0.013	0.0	0.0	

5 rows × 61 columns

TRANSPOSING THE DATAFRAME

In [13]:

```
# Converting the dataframe into multi indexed dataframe for better understanding wit
#outer layer & year as Inner layer Index

data = df.set_index(['iso_code','country','year'])
data
```

Out[13]:

			co2	co2_per_capita	trade_co2	cement_co2	cement_co2_per_ca
iso_code	country	year					
AFG	Afghanistan	1949	0.015	0.002	0.000	0.000	0
		1950	0.084	0.011	0.000	0.000	0
		1951	0.092	0.012	0.000	0.000	0
		1952	0.092	0.012	0.000	0.000	0
		1953	0.106	0.013	0.000	0.000	0
...	...	...	...	...	...	...	
ZWE	Zimbabwe	2016	10.738	0.765	1.415	0.639	0
		2017	9.582	0.673	1.666	0.678	0
		2018	11.854	0.821	1.308	0.697	0
		2019	10.949	0.748	1.473	0.697	0
		2020	10.531	0.709	0.000	0.697	0

25989 rows × 58 columns

FILTERING ONLY THESE BELOW TEN COUNTRIES FROM MAIN DATAFRAME BECAUSE IN MY ANALSYIS I NEED ONLY THESE COUNTRIES

In [14]:

```
country_codes = ['AFG','BTN','CHN','GBR','IND','JPN','MYS','RUS','SAU','USA']
analysis_data = data.loc[country_codes] # USING LOC() filtering out the country

final = analysis_data[['co2', 'co2_per_capita', 'gdp', 'gdp_per_capita', 'co2_per_gd
final
```

Out[14]:

			co2	co2_per_capita	gdp	gdp_per_capita	co2_per_g
iso_code	country	year					
AFG	Afghanistan	1949	0.015	0.002	0.000000e+00	0.000000	0.(
		1950	0.084	0.011	9.421400e+09	1215.332543	0.(
		1951	0.092	0.012	9.692280e+09	1236.236369	0.(
		1952	0.092	0.012	1.001733e+10	1262.264378	0.(
		1953	0.106	0.013	1.063052e+10	1322.255925	0.(
...	...	...	...	...	...	...	
USA	United States	2016	5248.024	16.247	1.716256e+13	53132.221374	0.3
		2017	5207.751	16.020	1.759628e+13	54128.295610	0.2
		2018	5375.491	16.434	1.814065e+13	55459.654757	0.2
		2019	5255.816	15.972	0.000000e+00	0.000000	0.(
		2020	4712.771	14.238	0.000000e+00	0.000000	0.(

1460 rows × 6 columns

DATAFRAME INFO

In [15]:

```
final.head(10)
```

Out[15]:

			co2	co2_per_capita	gdp	gdp_per_capita	co2_per_gdp
iso_code	country	year					
AFG	Afghanistan	1949	0.015	0.002	0.000000e+00	0.000000	0.000
		1950	0.084	0.011	9.421400e+09	1215.332543	0.009
		1951	0.092	0.012	9.692280e+09	1236.236369	0.010
		1952	0.092	0.012	1.001733e+10	1262.264378	0.009
		1953	0.106	0.013	1.063052e+10	1322.255925	0.010
		1954	0.106	0.013	1.086636e+10	1333.080489	0.010
		1955	0.154	0.019	1.107819e+10	1339.402333	0.014
		1956	0.183	0.022	1.158124e+10	1378.903601	0.016
		1957	0.293	0.034	1.157897e+10	1356.620995	0.025
		1958	0.330	0.038	1.223884e+10	1409.989065	0.027

In [16]:

```
final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 1460 entries, ('AFG', 'Afghanistan', 1949) to ('USA', 'Uni
ted States', 2020)
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   co2                  1460 non-null   float64
 1   co2_per_capita       1460 non-null   float64
 2   gdp                  1460 non-null   float64
 3   gdp_per_capita       1460 non-null   float64
 4   co2_per_gdp          1460 non-null   float64
 5   co2_per_unit_energy  1460 non-null   float64
dtypes: float64(6)
memory usage: 107.2+ KB
```

In [17]:

```
# Total number of Countries - 10 and their data entries
final.groupby(level=[0,1]).size()
```

Out[17]:

```
iso_code  country
AFG       Afghanistan        72
BTN       Bhutan             51
CHN       China             122
GBR       United Kingdom    271
IND       India             163
JPN       Japan             153
MYS       Malaysia          131
RUS       Russia            191
SAU       Saudi Arabia       85
USA       United States     221
dtype: int64
```

STATS

In [18]:

```
final.describe()
```

Out[18]:

	co2	co2_per_capita	gdp	gdp_per_capita	co2_per_gdp	co2_per_unit_e
count	1460.000000	1460.000000	1.460000e+03	1460.000000	1460.000000	1460.0
mean	676.749615	4.598415	1.143679e+12	7473.893234	0.389464	0.0
std	1435.708663	5.767220	2.639005e+12	10933.908374	0.423208	0.1
min	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.0
25%	6.834000	0.134000	0.000000e+00	0.000000	0.000000	0.0
50%	81.794000	1.402000	1.915786e+11	2853.209046	0.285000	0.0
75%	571.521000	8.983750	9.329851e+11	9567.947769	0.631250	0.2
max	10667.887000	22.236000	1.815162e+13	55459.654757	1.648000	0.5

In [19]:

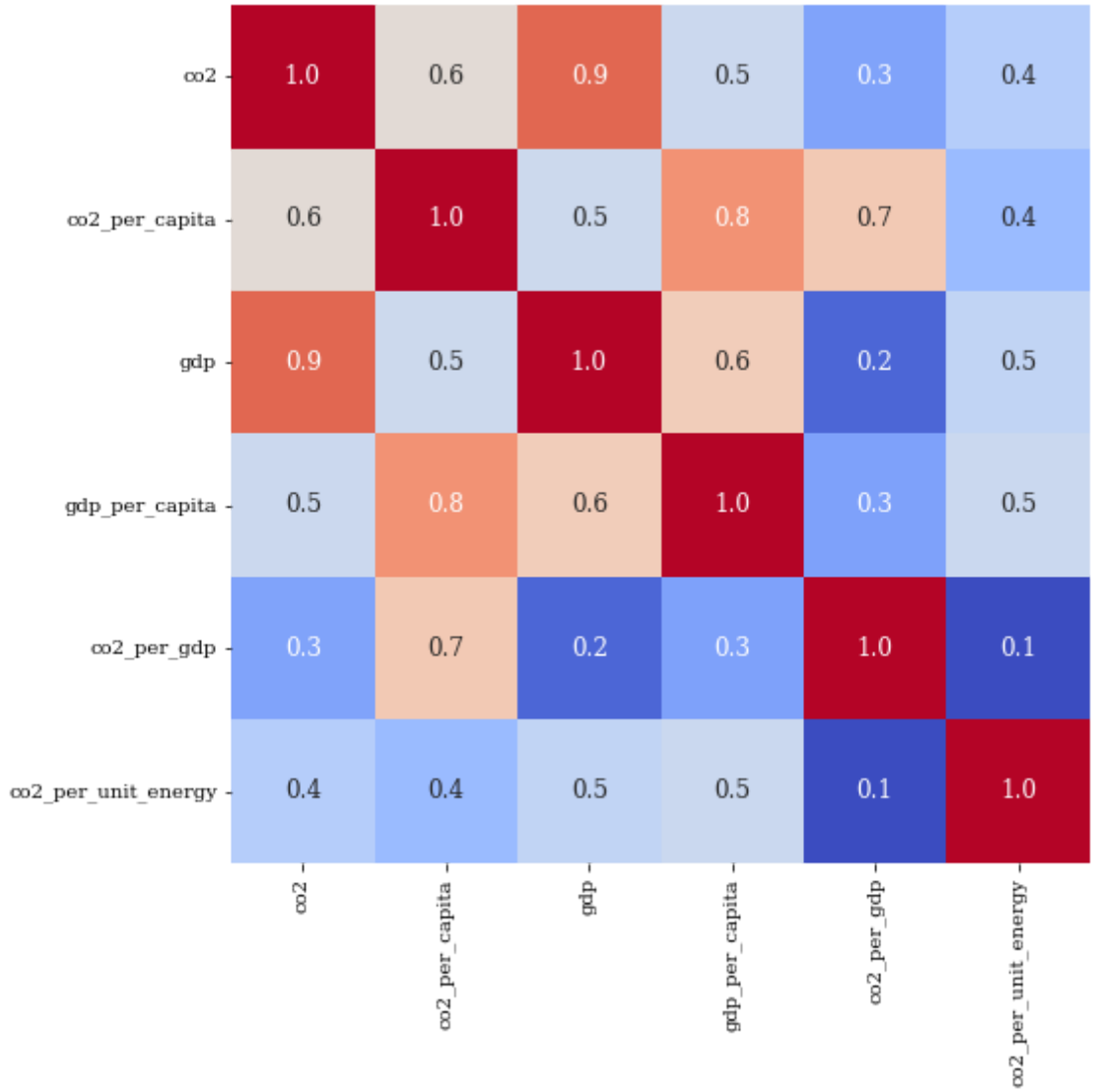
```
final_corr_df = final.corr()
final_corr_df
```

Out[19]:

	co2	co2_per_capita	gdp	gdp_per_capita	co2_per_gdp	co2_per_
co2	1.000000	0.576903	0.879032	0.503192	0.305383	
co2_per_capita	0.576903	1.000000	0.505778	0.795087	0.653995	
gdp	0.879032	0.505778	1.000000	0.641060	0.170822	
gdp_per_capita	0.503192	0.795087	0.641060	1.000000	0.303800	
co2_per_gdp	0.305383	0.653995	0.170822	0.303800	1.000000	
co2_per_unit_energy	0.438799	0.370362	0.472804	0.498188	0.117845	

In [20]:

```
#Plot correlation matrix of indicators
plt.figure(figsize=(10,8))
p = sns.heatmap(final_corr_df,
                cmap='coolwarm',
                annot=True,
                fmt=".1f",
                annot_kws={'size':12},
                cbar=False,
                square=True)
```



Type *Markdown* and LaTeX: α^2

In [ ]:

FITTING

In [21]:

```
years = [2011,2012,2013,2014,2015,2016,2017,2018,2019]
gdp = final.loc[slice(None),slice(None), years] # filtering recent Ten years only
df1 = gdp.reset_index()
gdp_df = df1.set_index(['iso_code','country'])
```

Analyszing India GDP

In [22]:

```
india_gdp_df = final.loc['IND']
india_gdp_df = india_gdp_df.reset_index()
india_df = india_gdp_df[['year', 'gdp_per_capita']]
india_df = india_df[0:161]
india_df
```

Out[22]:

	year	gdp_per_capita
0	1858	0.000000
1	1859	0.000000
2	1860	0.000000
3	1861	891.352377
4	1862	0.000000
...	...	...
156	2014	5213.597553
157	2015	5541.036749
158	2016	5864.362584
159	2017	6181.960526
160	2018	6532.220394

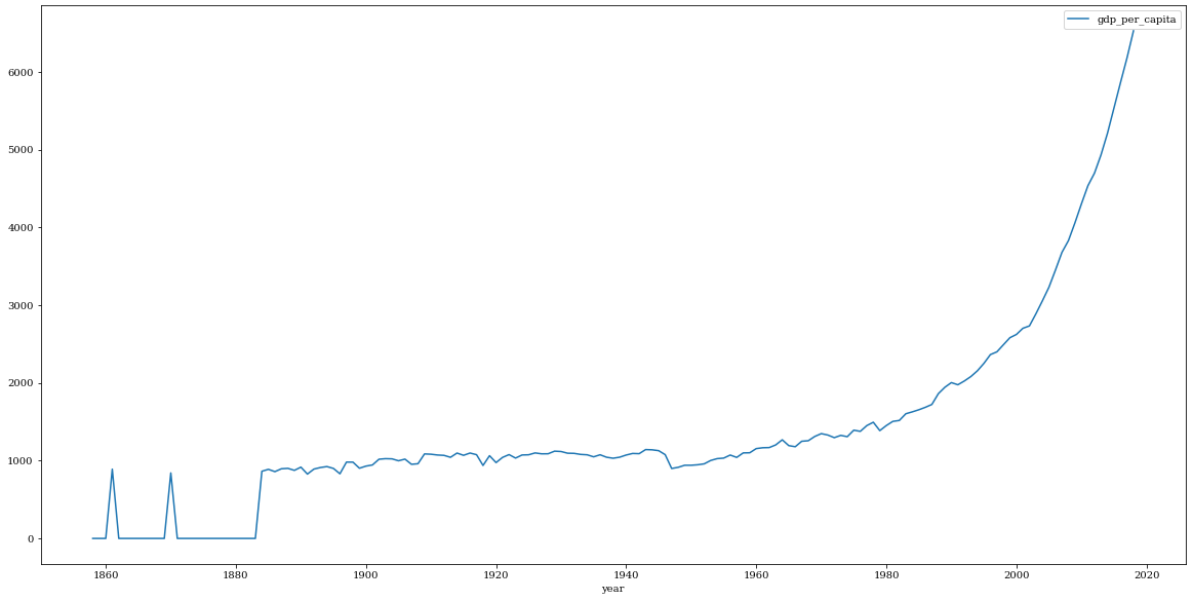
161 rows × 2 columns

In [23]:

```
india_df.plot('year','gdp_per_capita')
```

Out[23]:

```
<AxesSubplot:xlabel='year'>
```



EXPOTENTIAL

In [24]:

```
def exponential(t, n0, g):
    """Calculates exponential function with scale factor n0 and growth rate g."""
    t = t - 1960.0
    f = n0 * np.exp(g*t)
    return f
```

In [25]:

```
def linfunc(x, a, b):
    """ Function for fitting
        x: independent variable
        a, b: parameters to be fitted
    """
    y = a*x + b
    return y
```

In [26]:

```
# Fitting the data
# curvefit expects the argument list of the fit function to be (x, p1, p2, p3, p4, .
param, covar = opt.curve_fit(exponential, gdp_df["year"], gdp_df["gdp"],
                             p0=(73233967692.102798, 0.03))

# specify errors: sigma=errors, absolute_sigma=True

# list of parameters
print("parameters:", param)
# variance-covariance matrix
print()
print("covariance-matrix", covar)

# one can get the std. dev. this way
# np.diag() extracts the diagonal of a matrix
sigma1 = np.sqrt(np.diag(covar))

print()
print(f"a = {param[0]:5.3f} +/- {sigma1[0]:5.3f}")
print(f"b = {param[1]:5.3f} +/- {sigma1[1]:5.3f}")
```

```
parameters: [ 4.62229010e+13 -4.16798896e-02]

covariance-matrix [[ 1.70836329e+28 -6.78360576e+12]
 [-6.78360576e+12  2.69953722e-03]]

a = 46222901027776.000 +/- 130704371965731.125
b = -0.042 +/- 0.052
```

EXPOTENTIAL GROWTH

In [27]:

```
def exponential(t, n0, g):
    """Calculates exponential function with scale factor n0 and growth rate g."""
    t = t - 1960.0
    f = n0 * np.exp(g*t)
    return f
```

In [28]:

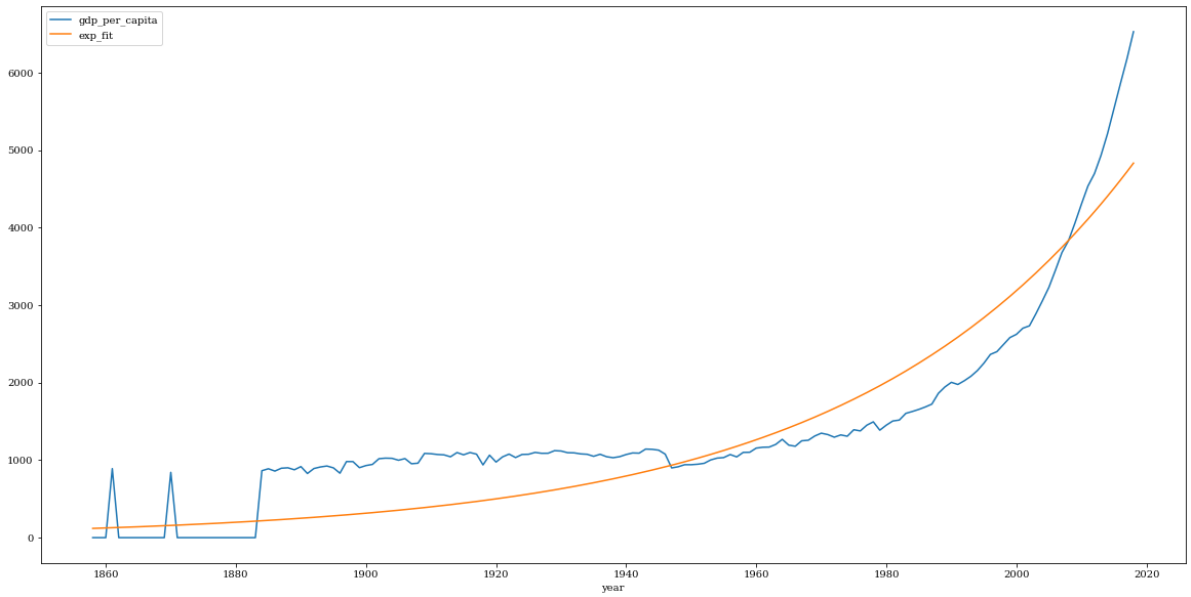
```
# Making sure year is in Int64 type
print(type(india_df["year"].iloc[1]))
india_df["year"] = pd.to_numeric(india_df["year"])
print(type(india_df["year"].iloc[1]))
param, covar1 = opt.curve_fit(exponential, india_df["year"], india_df["gdp_per_capit
                             p0=(73233967692.102798, 0.03))
```

```
<class 'numpy.int64'>
<class 'numpy.int64'>
```

In [29]:

```
india_df["exp_fit"] = exponential(india_df["year"], *param)

india_df.plot("year", ["gdp_per_capita", "exp_fit"])
plt.show()
```



LOGISTIC GROWTH

In [30]:

```
def logistic(t, n0, g, t0):
    """Calculates the logistic function with scale factor n0 and growth rate g"""
    f = n0 / (1 + np.exp(-g*(t - t0)))
    return f
```

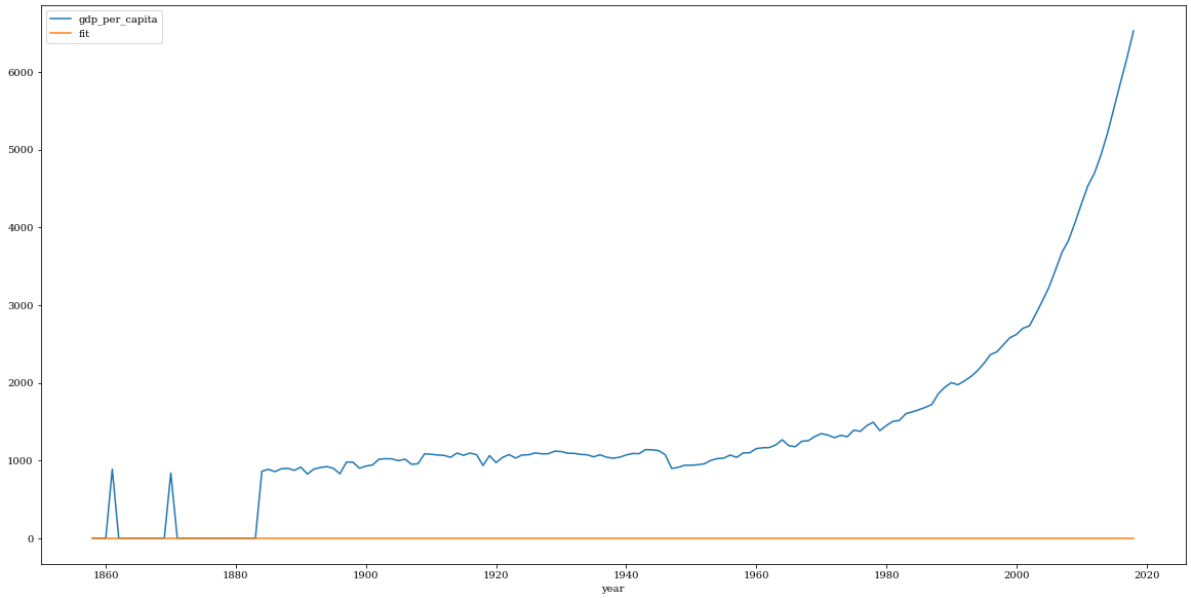
In [31]:

```
param, covar = opt.curve_fit(logistic, india_df["year"], india_df["gdp_per_capita"],
                             p0=(3e12, 0.03, 20000.0))
```

In [32]:

```
india_df["fit"] = logistic(india_df["year"], *param)

india_df.plot("year", ["gdp_per_capita", "fit"])
plt.show()
```



OUTLIERS

In [33]:

```
def err_ranges(x, func, param, sigma):
    """
    Calculates the upper and lower limits for the function, parameters and
    sigmas for single value or array x. Functions values are calculated for
    all combinations of +/- sigma and the minimum and maximum is determined.
    Can be used for all number of parameters and sigmas >=1.

    This routine can be used in assignment programs.
    """

    import itertools as iter

    # initiate arrays for lower and upper limits
    lower = func(x, *param)
    upper = lower

    uplow = []   # list to hold upper and lower limits for parameters
    for p,s in zip(param, sigma):
        pmin = p - s
        pmax = p + s
        uplow.append((pmin, pmax))

    pmix = list(iter.product(*uplow))

    for p in pmix:
        y = func(x, *p)
        lower = np.minimum(lower, y)
        upper = np.maximum(upper, y)

    return lower, upper
```

In [34]:

```
#low, up = err_ranges(india_df['year'], exponential, param, sigma1)
```

In [35]:

```
year = np.arange(1875, 2031)
print(year)
```

```
[1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888
 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902
 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916
 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930
 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944
 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958
 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972
 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986
 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000
 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028
 2029 2030]
```

In [36]:

```
df_final = india_df
```

STANDARDISATION

In [37]:

```
def norm(array):
    """ Returns array normalised to [0,1]. Array can be a numpy array
    or a column of a dataframe"""

    min_val = np.min(array)
    max_val = np.max(array)

    scaled = (array-min_val) / (max_val-min_val)

    return scaled


def norm_df(df):
    """
    Returns all columns of the dataframe normalised to [0,1] with the
    exception the first (containing the names)
    Calls function norm to do the normalisation of one column, but
    doing all in one function is also fine.
    """

    # iterate over all columns
    for col in df.columns[1:]:      # excluding the first column
        df[col] = norm(df[col])

    return df
```

In [38]:

```
#standardization along columns
df_final_std = (df_final.T-df_final.T.mean()) / df_final.T.std()
df_final_std
```

Out[38]:

	0	1	2	3	4	5	6	
year	1.497139	1.497004	1.496862	1.332257	1.496559	1.496397	1.496227	1.49
gdp_per_capita	-0.542701	-0.543673	-0.544666	0.200021	-0.546719	-0.547781	-0.548866	-0.54
exp_fit	-0.411736	-0.409658	-0.407531	-0.691486	-0.403121	-0.400835	-0.398494	-0.39
fit	-0.542701	-0.543673	-0.544666	-0.840792	-0.546719	-0.547781	-0.548866	-0.54

4 rows × 161 columns

CLUSTERING

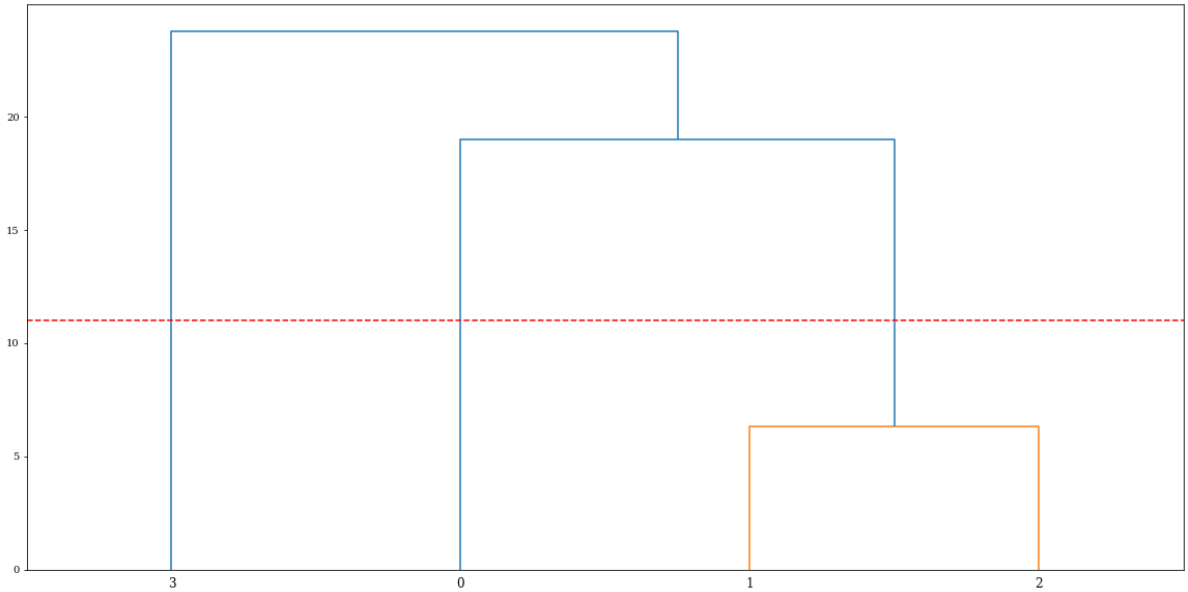
In [39]:

```
import scipy.cluster.hierarchy as sch

#Create dendrogram
dendrogram = sch.dendrogram(sch.linkage(df_final_std, method='ward'))
plt.axhline(y=11, color='r', linestyle='--')
```

Out[39]:

```
<matplotlib.lines.Line2D at 0x7fde9b73ce50>
```



K-Means Clustering

In [40]:

```
# import K-Means clustering libraries
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

from sklearn.metrics import silhouette_score
```

In [41]:

```
def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)
```

In [42]:

```
df_final_std = clean_dataset(df_final)
df_final_std
```

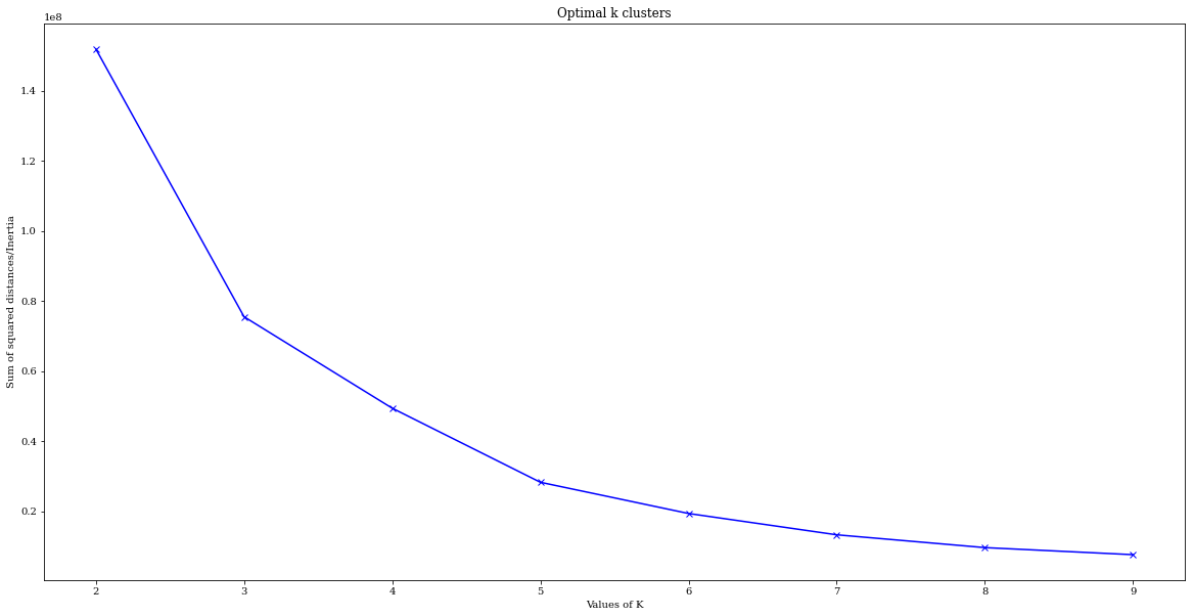
Out[42]:

	year	gdp_per_capita	exp_fit	fit
0	1858.0	0.000000	119.290809	1.282350e-224
1	1859.0	0.000000	122.083191	1.321403e-224
2	1860.0	0.000000	124.940937	1.361646e-224
3	1861.0	891.352377	127.865578	1.403114e-224
4	1862.0	0.000000	130.858679	1.445845e-224
...	...	...	...	...
156	2014.0	5213.597553	4407.915488	1.381989e-222
157	2015.0	5541.036749	4511.096793	1.424077e-222
158	2016.0	5864.362584	4616.693384	1.467447e-222
159	2017.0	6181.960526	4724.761801	1.512137e-222
160	2018.0	6532.220394	4835.359903	1.558189e-222

161 rows × 4 columns

In [43]:

```
Sum_of_squared_distances = []
K = range(2,10)
for num_clusters in K :
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(df_final_std)
    Sum_of_squared_distances.append(kmeans.inertia_)
plt.plot(K,Sum_of_squared_distances,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Sum of squared distances/Inertia')
plt.title('Optimal k clusters')
plt.show()
```



In [44]:

```
# defining the kmeans function with initialization as k-means++
kmeans = KMeans(n_clusters=2, init='k-means++')

# fitting the k means algorithm on scaled data
kmeans.fit(df_final_std)
```

Out[44]:

```
KMeans(n_clusters=2)
```

In [45]:

```
# inertia on the fitted data
kmeans.inertia_
```

Out[45]:

```
151976915.3806578
```

In [46]:

```
#Predict
pred = kmeans.predict(df_final_std)
```

In [47]:

```
frame = pd.DataFrame(df_final_std)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Out[47]:

```
0    131
1     30
Name: cluster, dtype: int64
```

In [48]:

```
frame
```

Out[48]:

	year	gdp_per_capita	exp_fit	fit	cluster
0	1858.0	0.000000	119.290809	1.282350e-224	0
1	1859.0	0.000000	122.083191	1.321403e-224	0
2	1860.0	0.000000	124.940937	1.361646e-224	0
3	1861.0	891.352377	127.865578	1.403114e-224	0
4	1862.0	0.000000	130.858679	1.445845e-224	0
...	...	...	...	...	...
156	2014.0	5213.597553	4407.915488	1.381989e-222	1
157	2015.0	5541.036749	4511.096793	1.424077e-222	1
158	2016.0	5864.362584	4616.693384	1.467447e-222	1
159	2017.0	6181.960526	4724.761801	1.512137e-222	1
160	2018.0	6532.220394	4835.359903	1.558189e-222	1

161 rows × 5 columns

In [49]:

```
final
```

Out[49]:

iso_code	country	year	co2	co2_per_capita	gdp	gdp_per_capita	co2_per_g
AFG	Afghanistan	1949	0.015	0.002	0.000000e+00	0.000000	0.(
		1950	0.084	0.011	9.421400e+09	1215.332543	0.(
		1951	0.092	0.012	9.692280e+09	1236.236369	0.(
		1952	0.092	0.012	1.001733e+10	1262.264378	0.(
		1953	0.106	0.013	1.063052e+10	1322.255925	0.(
...	...	...	...	...	...	...	
USA	United States	2016	5248.024	16.247	1.716256e+13	53132.221374	0.3
		2017	5207.751	16.020	1.759628e+13	54128.295610	0.2
		2018	5375.491	16.434	1.814065e+13	55459.654757	0.2
		2019	5255.816	15.972	0.000000e+00	0.000000	0.(
		2020	4712.771	14.238	0.000000e+00	0.000000	0.(

1460 rows × 6 columns

In [50]:

```
df_final_G = final.groupby(['iso_code']).mean()
df_final_G
```

Out[50]:

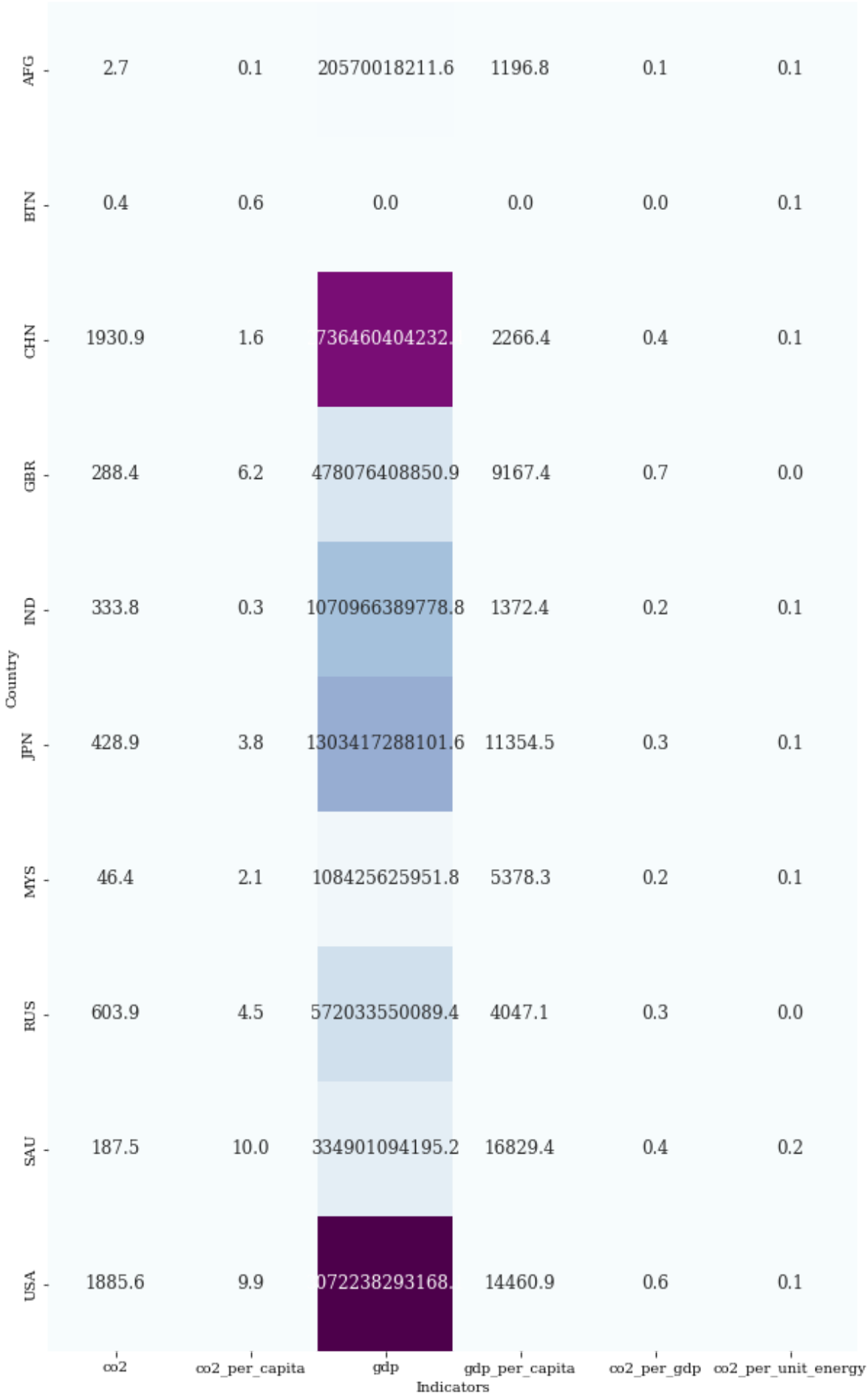
iso_code	co2	co2_per_capita	gdp	gdp_per_capita	co2_per_gdp	co2_per_unit
AFG	2.678486	0.132292	2.057002e+10	1196.844618	0.097222	(
BTN	0.376294	0.558549	0.000000e+00	0.000000	0.000000	(
CHN	1930.852590	1.552459	2.736460e+12	2266.431858	0.376025	(
GBR	288.417520	6.238565	4.780764e+11	9167.368158	0.651096	(
IND	333.843521	0.344975	1.070966e+12	1372.412631	0.153417	(
JPN	428.922647	3.769673	1.303417e+12	11354.480490	0.290111	(
MYS	46.350863	2.076252	1.084256e+11	5378.316367	0.241855	(
RUS	603.896063	4.510539	5.720336e+11	4047.051045	0.321476	(
SAU	187.486753	9.989282	3.349011e+11	16829.355161	0.421529	(
USA	1885.624833	9.864443	3.072238e+12	14460.948849	0.637950	(

In [51]:

```
plt.figure(figsize=(10,20))
sns.heatmap(df_final_G,
            cmap = 'BuPu',
            annot=True,
            fmt=".1f",
            annot_kws={'size':12},
            cbar=False,
            square=True)
plt.xlabel('Indicators')
plt.ylabel('Country')
```

Out[51]:

```
Text(70.0, 0.5, 'Country')
```


AFG 2.7 0.1 20570018211.6 1196.8 0.1 0.1
BTN 0.4 0.6 0.0 0.0 0.0 0.1
CHN 1930.9 1.6 736460404232. 2266.4 0.4 0.1
GBR 288.4 6.2 478076408850.9 9167.4 0.7 0.0
IND 333.8 0.3 1070966389778.8 1372.4 0.2 0.1
JPN 428.9 3.8 1303417288101.6 11354.5 0.3 0.1
MYS 46.4 2.1 108425625951.8 5378.3 0.2 0.1
RUS 603.9 4.5 572033550089.4 4047.1 0.3 0.0
SAU 187.5 10.0 334901094195.2 16829.4 0.4 0.2
USA 1885.6 9.9 072238293168. 14460.9 0.6 0.1
Country
co2
co2_per_capita
gdp
gdp_per_capita
co2_per_gdp
co2_per_unit_energy
Indicators

In [ ]:

In [ ]: