NAME : PAVAN DEEPAK THATIKONDA

Student ID : 22032994

Course : Msc Data Science with Placement year

APPLIED DATA SCIENCE REPORT :

# Exploratory Data Analysis on the World Bank Data

The goal of this project is to gather data from the World Bank Open APIs, prepare it using Python's PyPlots, and then analyse it.

In [ ]:

**import** pandas **as** pd **import** numpy **as** np **import** requests

**from** IPython.display **import** display

**import** matplotlib.pyplot **as** plt

# Task-1: Data Identification

The following API has been chosen for this analysis: <https://datahelpdesk.worldbank.org/knowledgebase/articles/889392-api-documentation>

The World Bank's data includes statistics and demographic information about population, employment, health, GDP, energy consumption, and other topics for all nations from 1960 to 2018. Each of these categories, which are known as indicators, is described by a code. The following indicators have been chosen for analysis:

SP.POP.TOTL - Total Population

SP.POP.TOTL.FE.IN - Total Female Population SP.POP.TOTL.MA.IN - Total Male Population

SP.DYN.CBRT.IN Birth Rate

SP.DYN.CDRT.IN Death Rate

SE.COM.DURS - Compulsory Education Duration SL.IND.EMPL.ZS - Employment in Industry(%)

SL.AGR.EMPL.ZS - Employment in Agriculture(%)

SL.AGR.EMPL.FE.ZS - Female Employment in Agriculture(%) SL.IND.EMPL.FE.ZS - Female Employment in Industry(%)

SL.UEM.TOTL.ZS - Unemployment(%) NY.GDP.MKTP.CD - GDP in USD

NY.ADJ.NNTY.PC.KD.ZG - National Income per Capita NY.GSR.NFCY.CD - Net income from Abroad

NV.AGR.TOTL.CD - Agriculture value added(in USD)

EG.USE.ELEC.KH.PC - Electric Power Consumption(kWH per capita) EG.FEC.RNEW.ZS - Renewable Energy Consumption (%)

EG.USE.COMM.FO.ZS - Fossil Fuel Consumption (%)

The following countries have been chosen for analysis:

US - United States of America CN - China

JP - Japan

CA - Canada

GB - Great Britain ZA - South Africa

The default API address is <http://api.worldbank.org/v2/>. to get the data for each year, and is followed by the indicator code and country code. In order to utilise the API, no authentication is necessary.

# Task-2: Data Collection

## Defining Constants

Since the API uses codes, we have created a mapping between the codes and meaningful names to use while parsing and storing the data.

In [ ]:

*# Base URL used in all the API calls*

BASE\_URL**=**['http://api.worldbank.org/v2/'](http://api.worldbank.org/v2/%27)

*# List of indicators according to the features defined above*

INDICATOR\_CODES**=**['SP.POP.TOTL', 'SP.POP.TOTL.FE.IN', 'SP.POP.TOTL.MA.IN', 'SP.DYN.CBRT.IN', 'SP.DYN.CDRT.IN',

'SE.COM.DURS',

'SL.IND.EMPL.ZS', 'SL.AGR.EMPL.ZS', 'SL.AGR.EMPL.FE.ZS', 'SL.IND.EMPL.FE.ZS', 'SL.UEM.TOTL.ZS', 'NY.GDP.MKTP.CD',

'NY.ADJ.NNTY.PC.KD.ZG', 'NY.GSR.NFCY.CD', 'NV.AGR.TOTL.CD',

'EG.USE.ELEC.KH.PC', 'EG.FEC.RNEW.ZS', 'EG.USE.COMM.FO.ZS']

COUNTRY\_LIST**=**['USA', 'China', 'Japan', 'Canada', 'Great Britain', 'South Africa']

*# mapping of feature codes to more meaningful names*

featureMap**=**{

"SP.POP.TOTL": "Total Population", "SP.POP.TOTL.FE.IN": "Female Population", "SP.POP.TOTL.MA.IN": "Male Population", "SP.DYN.CBRT.IN": "Birth Rate",

"SP.DYN.CDRT.IN": "Death Rate",

"SE.COM.DURS": "Compulsory Education Dur.", "SL.IND.EMPL.ZS":"Employment in Industry(%)", "SL.AGR.EMPL.ZS": "Employment in Agriculture(%)", "SL.AGR.EMPL.FE.ZS": "Female Employment in Agriculture(%)", "SL.IND.EMPL.FE.ZS": "Female Employment in Industry(%)", "SL.UEM.TOTL.ZS": "Unemployment(%)",

"NY.GDP.MKTP.CD": "GDP in USD",

"NY.ADJ.NNTY.PC.KD.ZG":"National Income per Capita", "NY.GSR.NFCY.CD":"Net income from Abroad", "NV.AGR.TOTL.CD":"Agriculture value added(in USD)", "EG.USE.ELEC.KH.PC":"Electric Power Consumption(kWH per capita)", "EG.FEC.RNEW.ZS":"Renewable Energy Consumption (%)", "EG.USE.COMM.FO.ZS":"Fossil Fuel Consumption (%)"

}

*# Mapping of country codes to their actual names*

countryMap**=**{ "US": "USA",

"CN": "China",

"JP": "Japan",

"CA": "Canada",

"GB": "Great Britain", "ZA": "South Africa"

}

*# constant parameters used in sending the request.*

params **=** dict()

*# to ensure we receive a JSON response*

params['format']**=**'json'

*# The data we fetch is for 59 years.*

*# Hence we change the default page size of 50 to 100 to ensure we need only one API call per feature.*

params['per\_page']**=**'100'

*# Range of years for which the data is needed*

params['date']**=**'1960:2018'

## Fetch data through API calls

The following routines were created to call an API for each feature and retrieve the data. These functions each carry out the following duties:

**getCountrywiseDF():** Calls the loadJSONData function for the specified country and returns a dataframe containing all of the data for that nation.

**loadJSONData():** Makes the request to the endpoint after creating the proper URL using the base URL, the indicator code, and the country code. It gives back a list of values for each feature.

In [ ]:

*# Function to get JSON data from the endpoint*

**def** loadJSONData(country\_code): dataList**=**[]

*# iterate over each indicator code specified in the contant INDICATOR\_CODES defined above*

**for** indicator **in** INDICATOR\_CODES:

*# form the URL in the desired format*

*# E.g:* [*http://api.worldbank.org/v2/countries/us/indicators/SP.POP.TOTL?format=json&per\_page=200&date=1960:2018*](http://api.worldbank.org/v2/countries/us/indicators/SP.POP.TOTL?format=json&per_page=200&date=1960%3A2018)

url**=**BASE\_URL**+**'countries/'**+**country\_code**.**lower()**+**'/indicators/'**+**indicator

*# send the request using the resquests module*

response **=** requests**.**get(url, params**=**params)

*# validate the response status code*

*# The API returns a status\_code 200 even for error messages,*

*# however, the response body contains a field called "message" that includes the details of the error # check if message is not present in the response*

**if** response**.**status\_code **==** 200 **and** ("message" **not in** response**.**json()[0]**.**keys()):

*# print("Successfully got data for: " + str(featureMap[indicator]))*

*# list of values for one feature*

indicatorVals**=**[]

*# the response is an array containing two arrays - [[{page: 1, ...}], [{year: 2018, SP.POP.TOTL: 123455}, ...]] # hence we check if the length of the response is >1*

**if** len(response**.**json()) **>** 1:

*# if yes, iterate over each object in the response # each object gives one single value for each year* **for** obj **in** response**.**json()[1]:

*# check for empty values*

**if** obj['value'] **==** "" **or** obj['value'] **is None**: indicatorVals**.**append(**None**)

**else**:

*# if a value is present, add it to the list of indicator values*

indicatorVals**.**append(float(obj['value'])) dataList**.**append(indicatorVals)

**else**:

*# print an error message if the API call failed*

print("Error in Loading the data. Status Code: " **+** str(response**.**status\_code))

*# Once all the features have been obtained, add the values for the "Year"*

*# The API returns the indicator values from the most recent year. Hence, we create a list of years in reverse order*

dataList**.**append([year **for** year **in** range(2018, 1959, **-**1)])

*# return the list of lists of feature values [[val1,val2,val3...], [val1,val2,val3...], [val1,val2,val3...], ...]*

**return** dataList

*#----------------------------------------------------------------------------------------------------*

*# function to invokde the loadJSONData function and form the final DataFrame for each country*

**def** getCountrywiseDF(country\_code):

*# The resulting dataframe needs to have meaningful column names*

*# hence we create a list of column names from the map defined above*

col\_list**=**list(featureMap**.**values()) *# append the year column name* col\_list**.**append('Year')

print("------------------Loading data for: "**+**countryMap[country\_code]**+**" ")

*# for the given country call the loadJSONData function and fetch the data from the API*

dataList**=**loadJSONData(country\_code)

*# transform the list of lists of features into a DataFrame # np.column\_stack is used to add each list as a column*

df**=**pd**.**DataFrame(np**.**column\_stack(dataList), columns**=**col\_list)

*# add the country column by extracting the country name from the map using the country code*

df['Country'] **=** countryMap[country\_code]

*# return the formed dataframe for the given country*

**return** df

In [ ]:

*# Call the getCountrywiseDF function with the code of each country under consideration # We will have a seperate dataframe for each country - 7 data frames*

US\_df**=**getCountrywiseDF('US') CN\_df**=**getCountrywiseDF('CN') JP\_df**=**getCountrywiseDF('JP') CA\_df**=**getCountrywiseDF('CA') GB\_df**=**getCountrywiseDF('GB') ZA\_df**=**getCountrywiseDF('ZA')

print("Data Loading Completed")

------------------Loading data for: USA-----------------------

------------------Loading data for: China-----------------------

------------------Loading data for: Japan-----------------------

------------------Loading data for: Canada-----------------------

------------------Loading data for: Great Britain-----------------------

------------------Loading data for: South Africa-----------------------

Data Loading Completed

# Task-3: Data Pre-processing

The data mentioned above was most recently compiled from several API calls. The dataframes we constructed by obtaining the features from the API calls show that some of the features have some missing values. This indicates that before being used for analysis, the data has to be processed.

We generate a list of the dataframes created to simplify the pre-processing procedure and prevent manually giving the dataframes to each of the following functions. The original, unprocessed dataframes have not been altered; instead, the copy() technique has been utilised.

In [ ]:

*# store all the DataFrames in a list to iteratively apply pre-processing steps*

list\_df**=**[US\_df**.**copy(), CN\_df**.**copy(), JP\_df**.**copy(), CA\_df**.**copy(), GB\_df**.**copy(), ZA\_df**.**copy()]

## Fill missing values

The following function has been implemented to fill the missing values in features:

**fill\_missing\_values():** This function accepts a dataframe that has to have any missing values filled in. It fills in NaN values using the pandas dataframes *fillna()* function. The missing values are indicated by None, as can be seen from the raw dataframes that were populated above. Thus, instead of filling in None with NaN right away, we first fill in the mean value of the columns to take the place of the NaNs.

In [ ]:

*# Function to fill the remaining missing values with average values for columns*

**def** fill\_missing\_values(df):

*# validation for dataframes*

**if** df **is None**:

print("No DataFrame received")

**return**

*# create a copy*

df\_cp**=**df**.**copy()

print("Filling missing features for: " **+** df\_cp**.**iloc[0]['Country'])

*# get the list of columns in the dataframe*

cols\_list**=**list(df\_cp**.**columns)

*# exclude the last column - Country*

*# This column was added explicitly when the data was loaded, hence, it does not contain any missing values. # Also, fillna function does not work on categorical features since it performs an aggregation.* cols\_list**.**pop()

*# replace all None values with NaN, fillna only works on nans*

df\_cp**.**fillna(value**=**pd**.**np**.**nan, inplace**=True**)

*# replace all NaN values with the mean of the column values*

**for** col **in** cols\_list: df\_cp[col]**.**fillna((df\_cp[col]**.**mean()), inplace**=True**)

print("Filling missing values completed")

**return** df\_cp

In [ ]:

*# call the function on each DF for each country.*

*# The function fill\_missing\_features will be applied on each dataframe in list\_df through the map function in python.*

list\_df**=**list(map(fill\_missing\_values, list\_df))

Filling missing features for: USA Filling missing values completed Filling missing features for: China Filling missing values completed Filling missing features for: Japan Filling missing values completed Filling missing features for: Canada Filling missing values completed

Filling missing features for: Great Britain Filling missing values completed

Filling missing features for: South Africa Filling missing values completed

## Change the type of Numeric but Categorical Features

We can see that the 'Year' column contains a numeric number, but its magnitude is unimportant. Instead of representing a number, it actually stands for a period. This column can therefore be transformed into a category variable. To do this, the following function has been put in place:

**change\_year\_type():** This function transforms the year's dtype from one dataframe to another and then returns the revised dataframe.

In [ ]:

*# Function to change year type*

**def** change\_year\_type(df):

print("Changing type of Year for: " **+** df**.**loc[0]['Country']) *# validation to check if year column exists in the dataframe* **if** 'Year' **in** df**.**columns:

*# convert year to a string*

df['Year'] **=** df**.**Year**.**astype(str)

print("Completed changing type")

*# return the updated df*

**return** df

In [ ]:

*# call the function on each DF for each country.*

*# The function fill\_missing\_features will be applied on each dataframe in list\_df through the map function in python.*

list\_df**=**list(map(change\_year\_type, list\_df))

Changing type of Year for: USA Completed changing type

Changing type of Year for: China Completed changing type

Changing type of Year for: Japan Completed changing type

Changing type of Year for: Canada Completed changing type

Changing type of Year for: Great Britain Completed changing type

Changing type of Year for: South Africa Completed changing type

# Task 5: Analyse and Summarise the cleaned dataset

We may generate visualisations from the pre-processed datasets we have available to search for trends. To analyse the data, I utilised the **matplotlib** and **seaborn** libraries.

## Prepare a combined DataFrame for further analysis

Separate dataframes have been produced for each nation. In certain analyses, we might require all of the data from all the countries. Having a single consolidated dataframe available is therefore a smart idea.

In [ ]:

combined\_df**=**pd**.**concat(list\_df,sort**=False**) combined\_df**.**head(5)

Out[ ]:

**Compulsory**

**Female**

**Female**

**National**

**Agriculture**

**Electric Power Renewable Fossil Fuel**

**Total**

**Female**

**Male Birth Death Education Employment Employment in Employment in Employment in Unemployment(%) GDP in USD Income per Net income value added(in Consumption(kWH**

**Energy Consumption Year Country**

**Population Population Population Rate Rate**

**Dur. in Industry(%) Agriculture(%) Agriculture(%) Industry(%)**

**Capita from Abroad**

**USD)**

**per capita) Consumption**

**(%)**

**(%)**

**0** 326838199.0 164926348.0 161911851.0 11.6 8.678

**1** 325122128.0 164151818.0 160970309.0 11.8 8.638

**2** 323071755.0 163224028.0 159847727.0 12.2 8.493

**3** 320738994.0 162158414.0 158580581.0 12.4 8.440

12.0 19.870001

12.0 19.730000

12.0 19.780001

12.0 19.860001

1.37

1.43

1.43

1.44

0.76

0.75

0.77

0.76

8.78

8.71

8.71

8.83

3.90 2.053306e+13 1.833050 2.903070e+11 1.855984e+11

4.36 1.947734e+13 2.135014 2.929490e+11 1.844248e+11

4.87 1.869511e+13 -0.111701 2.319440e+11 1.762284e+11

5.28 1.820602e+13 3.214351 2.203830e+11 1.882382e+11

10318.041669

10318.041669

10318.041669

10318.041669

10.12 89.289662 2018 USA

9.92 89.289662 2017 USA

9.46 89.289662 2016 USA

9.03 82.427828 2015 USA

**4** 318386329.0 161084758.0 157301571.0 12.5 8.237 12.0 19.980000 1.35 0.70 8.86 6.17 1.755068e+13 2.182302 2.352260e+11 2.057054e+11 12993.965579 9.22 83.089042 2014 USA

## Analysis Using Plots

### Comparing Population of Countries in 2000 and 2018:

We can examine the population changes for various nations in 2018—the most recent year—as opposed to 2000 because we have statistics for several nations. The code that follows creates two DataFrames, one for each year being taken into account. The column Total Population is extracted. A chart using **grouped bar** data has been used to illustrate the population disparity.

In [ ]:

*# refer to the list of countries*

list\_countries **=** COUNTRY\_LIST

*# intialise two dataframes df\_00- year 2000, df\_18 - year 2018*

df\_00 **=** pd**.**DataFrame() df\_18 **=** pd**.**DataFrame()

*# for each dataframe in the list of cleaned dataframes*

**for** i,df **in** enumerate(list\_df):

*# pick the value of Total Population for year 2000 and 2018* df\_00[list\_countries[i]] **=** df[df['Year'] **==** "2000"]["Total Population"] df\_18[list\_countries[i]] **=** df[df['Year'] **==** "2018"]["Total Population"]

*# The resulting dataframes have the countries as columns and the two rows each for 2000 and 2018 # To be able to draw a grouped bar plot we need the years as columns, hence we take a transpose* df\_00 **=** df\_00**.**T

df\_18 **=** df\_18**.**T

pd**.**options**.**display**.**float\_format **=** '{:,.1f}'**.**format *# set other global format # rename the columns to the year*

df\_00 **=** df\_00**.**rename(columns**=**{18 : 2000}) df\_18 **=** df\_18**.**rename(columns**=**{0 : 2018})

*# join the dataframes for both the years*

df\_both\_years**=** df\_00**.**join(df\_18)

*# the index is the Country name, hence we add it as a column into the data frame.*

df\_both\_years['Countries'] **=** df\_both\_years**.**index

*# drop the original index*

df\_both\_years**.**reset\_index(drop**=True**)

print("Data of Total Population for 2000 and 2018 for all countries: ") display(df\_both\_years)

Data of Total Population for 2000 and 2018 for all countries:

**2000 2018 Countries**

**USA** 282,162,411.0 326,838,199.0 USA

**China** 1,262,645,000.0 1,402,760,000.0 China

**Japan** 126,843,000.0 126,811,000.0 Japan

**Canada** 30,685,730.0 37,065,084.0 Canada

**Great Britain** 58,892,514.0 66,460,344.0 Great Britain

**South Africa** 46,813,266.0 57,339,635.0 South Africa

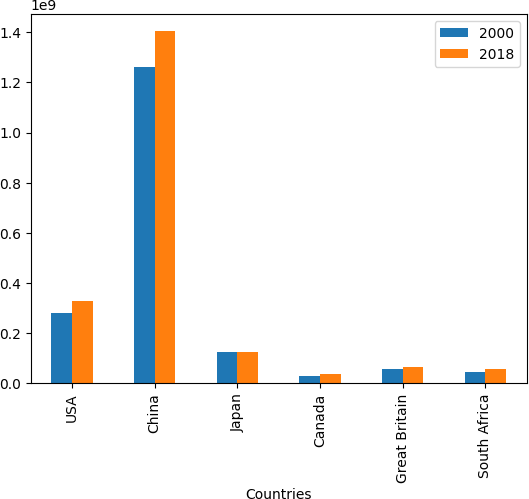
In [ ]:

plt**.**figure(figsize**=**(7, 5))

*# plot the chart using matplotlib.pyplot library*

df\_both\_years**.**plot(kind**=**'bar',x**=**'Countries',y**=**[2000, 2018])

Out[ ]: <Axes: xlabel='Countries'>

<Figure size 700x500 with 0 Axes>

China has the biggest population in both years, as is evident. Moreover, China's population has increased more in 2018 than in 2017. Canada is the country with the lowest population, albeit it is growing slightly. The population of Japan was the same in 2000 and 2018; this is a surprising discovery and suggests that the nation has effective population control systems.

### Average Birth Rate and Death Rate for countries across all the years

For numerous nations, we compared the total population in 2000 and 2018. Let's now contrast the average birth and mortality rates for various nations. We have **grouped the rows by country** and determined the mean birth rate and death rate using the combined dataframe that comprises the data for all the attributes for all the nations. I grouped the nations according to their birth and death rates using a bar chart.

The following function has been implemented:

**group\_df():** This function takes a column name as a parameter and returns a new DF with the data for countrywise average for the specified feature.

**plot\_bar():** This function takes a dataframe as an argument and plots a bar chart based on the X and Y features

In [ ]:

**def** group\_df(feature):

*# create a new dataframe*

df\_grouped**=**pd**.**DataFrame()

*# find average for each country*

df\_grouped['Avg. ' **+** feature]**=**combined\_df**.**groupby('Country')[feature]**.**mean()

*# set the index as a column - countries*

df\_grouped['Country']**=**df\_grouped**.**index

*# drop the index*

df\_grouped**.**reset\_index(drop**=True**, inplace**=True**)

*# sort the rows based of Avg Birth rate*

df\_grouped**.**sort\_values('Avg. '**+**feature, inplace**=True**, ascending**=False**)

print("Avg. " **+** feature) display(df\_grouped)

**return** df\_grouped

**def** plot\_bar(df, x\_feature, y\_feature):

*# bar plot*

plt**.**figure(figsize**=**(8, 5)) plt**.**bar(df[x\_feature], df["Avg. " **+** y\_feature],

width **=** 0.4)

plt**.**xlabel("Country") plt**.**ylabel(y\_feature) plt**.**show()

In [ ]:

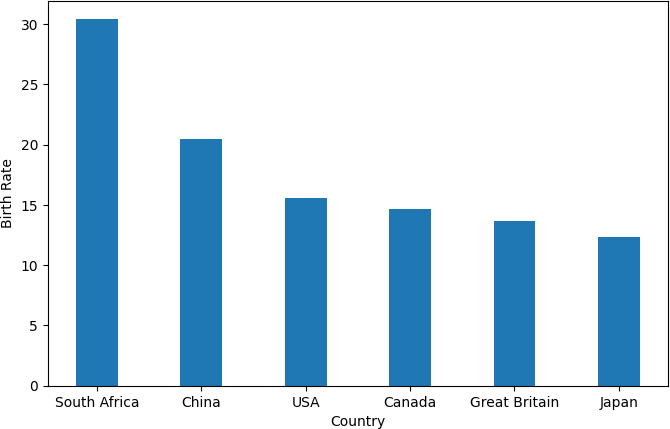
df\_birth**=**group\_df('Birth Rate') plot\_bar(df\_birth, 'Country', 'Birth Rate')

print("========================================================")

df\_death**=**group\_df('Death Rate') plot\_bar(df\_death, 'Country', 'Death Rate')

Avg. Birth Rate

**Avg. Birth Rate Country**



**4**

**1**

**5**

**0**

**2**

**3**

30.4 South Africa

20.5 China

15.6 USA

14.6 Canada

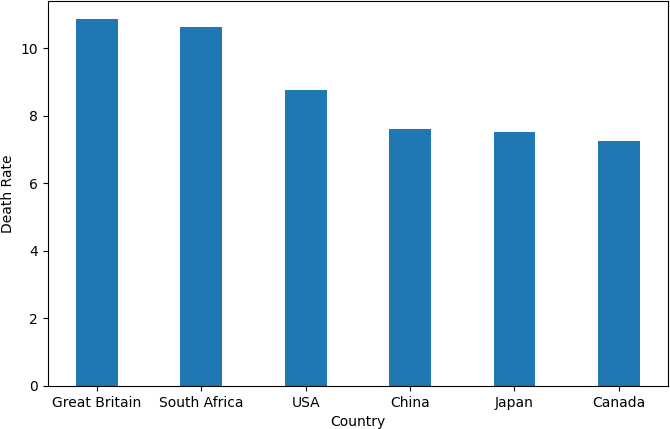
13.6 Great Britain

12.3 Japan

========================================================

Avg. Death Rate

**Avg. Death Rate Country**



**2**

10.9 Great Britain

**4**

**5**

**1**

**3**

**0**

10.6 South Africa

8.7 USA

7.6 China

7.5 Japan

7.2 Canada

The two plots above show that South Africa has the greatest average birth rate, which is noteworthy given that Great Britain has the second-highest average death rate. Another unexpected trend is that it is similar for the USA. China has the highest population in the world because its birth rate is higher than its death rate. Great Britain has a low population because the death rate is significantly greater than the average birth rate.

### GDP for all countries in the last 10 years

To illustrate how the GDP of different nations has changed over the past ten years, I utilised a line chart. From each country's dataframe, we take the columns Year, GDP in USD, and Country, and we store them in a smaller dataframe for visualisation. A **line plot** clearly shows the trends over a period of time and can also be used to compare the trends of different categories.

In [ ]:

*# function to to form a dataframe with Year, GDP and Country*

**def** extract\_columns(df\_cleaned): df**=**pd**.**DataFrame()

*# pick data for the recent 10 years, note that the data sorted in descending order of year*

df['Year']**=**df\_cleaned**.**loc[:10, 'Year']

df['GDP in USD']**=**df\_cleaned**.**loc[:10, 'GDP in USD'] df['Country']**=**df\_cleaned**.**loc[:10, 'Country'] **return** df

*# function to fetch a single dataframe with 3 features from each country*

**def** form\_gdp\_df():

*# function call to extract\_columns()* usdf**=**extract\_columns(US\_df) cndf**=**extract\_columns(CN\_df) jpdf**=**extract\_columns(JP\_df) cadf**=**extract\_columns(CA\_df) gbdf**=**extract\_columns(GB\_df) zadf**=**extract\_columns(ZA\_df)

*# combine the 7 dfs into a single df with 3 columns # we ignore the original index*

gdp\_df**=**pd**.**concat([usdf, cndf, jpdf, cadf, gbdf, zadf], ignore\_index**=True**)

**return** gdp\_df

*# get the combined DF*

gdp\_df**=**form\_gdp\_df()

print("Few records from the Dataframe containing Year, GDP and Country:") display(gdp\_df**.**head())

*# set figure size* plt**.**figure(figsize**=**(6, 5)) fig, ax **=** plt**.**subplots()

ax**.**plot(gdp\_df[gdp\_df**.**Country**==**'USA']**.**Year,gdp\_df[gdp\_df**.**Country**==**'USA']['GDP in USD'], color **=** 'green', label **=** 'USA') ax**.**plot(gdp\_df[gdp\_df**.**Country**==**'China']**.**Year,gdp\_df[gdp\_df**.**Country**==**'China']['GDP in USD'], color **=** 'orange', label **=** 'China') ax**.**plot(gdp\_df[gdp\_df**.**Country**==**'Japan']**.**Year,gdp\_df[gdp\_df**.**Country**==**'Japan']['GDP in USD'], color **=** 'blue', label **=** 'Japan') ax**.**plot(gdp\_df[gdp\_df**.**Country**==**'Canada']**.**Year,gdp\_df[gdp\_df**.**Country**==**'Canada']['GDP in USD'], color **=** 'red', label **=** 'Canada') ax**.**plot(gdp\_df[gdp\_df**.**Country**==**'Great Britain']**.**Year,gdp\_df[gdp\_df**.**Country**==**'Great Britain']['GDP in USD'], color **=** 'yellow', label **=** 'Great Britain') ax**.**plot(gdp\_df[gdp\_df**.**Country**==**'South Africa']**.**Year,gdp\_df[gdp\_df**.**Country**==**'South Africa']['GDP in USD'], color **=** 'black', label **=** 'South Africa') ax**.**legend(loc **=** 'upper left')

plt**.**show()

Few records from the Dataframe containing Year, GDP and Country:

**Year GDP in USD Country**

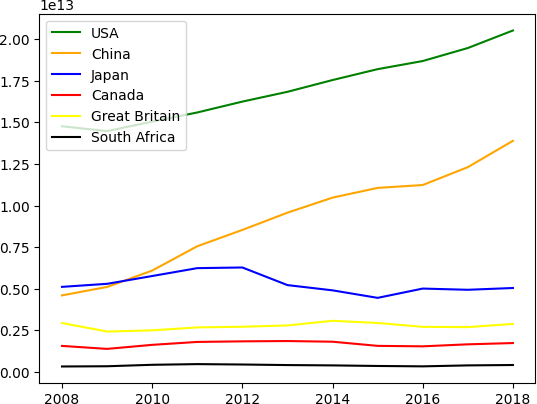
**0** 2018 20,533,057,312,000.0 USA

**1** 2017 19,477,336,549,000.0 USA

**2** 2016 18,695,110,842,000.0 USA

**3** 2015 18,206,020,741,000.0 USA

**4** 2014 17,550,680,174,000.0 USA

<Figure size 600x500 with 0 Axes>

We can see that USA has the highest GDP amongst all countries which is clear throughout the years. The GDP for China was low in the year 2008 and shows a significant rise upto 2018 which is attributed to their progress in various fields like manufacturing, however, in comparison to USA, it is still very less.

For Japan, the GDP shows a slight rise upto 2012 which may be due to Japan's growth strategy that was adopted to pull it out from deflation, but thereafter there is a dip in the GDP.

### Total Population vs Electric Power consumption for Canada upto 2015

Canada is a country with very low population. Let us analyse if the population has any effect on the Electrical Energy Consumption in this country upto the year 2015. I have used a single line plot to see the trend with increase in population.

In [ ]:

*# read the columns from the df for Canada*

df**=**CA\_df**.**loc[3:, ['Electric Power Consumption(kWH per capita)','Total Population', 'Year']]

print("First few records of the data: ") display(df**.**head())

*# line plot*

plt**.**figure(figsize**=**(6, 5))

plt**.**plot(df['Total Population'], df['Electric Power Consumption(kWH per capita)']) plt**.**show()

First few records of the data:

**Electric Power Consumption(kWH per capita) Total Population Year**

**3**

**4**

**5**

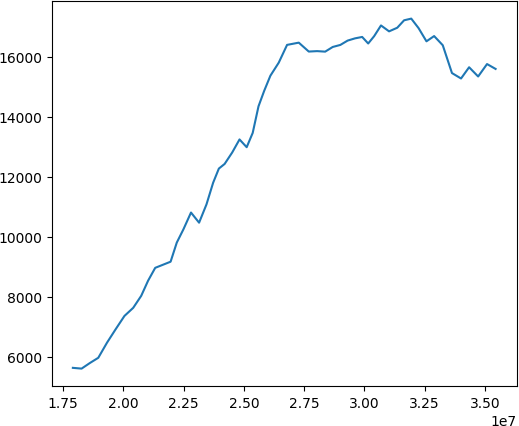
**6**

None 35,702,908.0 2015

15,588.5 35,437,435.0 2014

15,750.8 35,082,954.0 2013

15,336.6 34,714,222.0 2012

**7** 15,644.5 34,339,328.0 2011

From the above plot we can observe that for Canada the Electric power consumption has risen constantly with the rise in population. However, it remained constant after a point and thereafter reduced. Although the population in Canada is the least, it has a high electric power consumption which may be due to the extremely lower temperatures in the country.

### Variation in different Energy Consumption over the years for USA

For this analysis, I have chosen the energy consumption data for USA over the years upto 2010 and plotted a multi-line chart to observe the trend.

In [ ]:

*# Pick the columns Year, and 3 different power consumptions from the dataframe for russia*

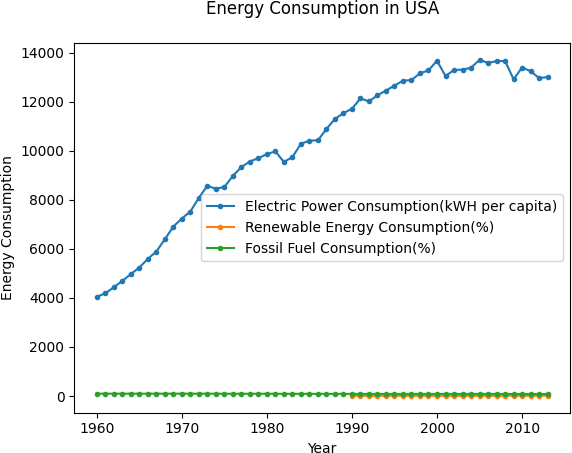
plt**.**plot(US\_df**.**loc[5:, ['Year']],US\_df**.**loc[5:, ['Electric Power Consumption(kWH per capita)']],'.-')

plt**.**plot(US\_df**.**loc[5:, ['Year']],US\_df**.**loc[5:, ['Renewable Energy Consumption (%)']],'.-')

plt**.**plot(US\_df**.**loc[5:, ['Year']],US\_df**.**loc[5:, ['Fossil Fuel Consumption (%)']],'.-')

plt**.**legend(['Electric Power Consumption(kWH per capita)', 'Renewable Energy Consumption(%)', 'Fossil Fuel Consumption(%)'], loc**=**'best') plt**.**title("Energy Consumption in USA\n")

plt**.**xlabel('Year') plt**.**ylabel('Energy Consumption') plt**.**show()



This figure makes it clear that the amount of electricity consumed has grown dramatically over time. Yet, compared to electrical energy, the usage of fossil fuels and renewable energy is modest. Use of fossil fuels appears to have grown between 2000 and 2010.

# Summary and Conclusion

A number of attributes in the World Bank dataset can be exploited to create engaging patterns in the data. I have gathered information for roughly 20 features for 7 different countries. Pre-processing was done on the data to get rid of columns with an excessive number of missing values and fill in the gaps with the average values for the features. Another option would have been to discard records with blank fields, but since the data only includes records dating back 59 years, this was not done (one record per year per country). In light of this, eliminating these values would have left a very small dataset that was unusable for analysis. To improve the analysis, more effective methods can be utilised to identify missing values.

A comparison between population counts, birth rates, and death rates was made, and the results showed some intriguing patterns on how the birth and death rates affect the population as a whole. Similar to this, it was discovered that China has the largest percentage of employment in agriculture depending on the characteristics of the countries. Additional examinations of the relationship between power consumption and overall population revealed that most nations had the highest electrical power usage when compared to their populations.

Additionally, extra analysis can be done on the employment characteristics, such as the proportion of men and women in each industry and comparisons across years and nations.

This approach can also be used to create predictive models that forecast GDP growth based on employment or population expansion based on birth and death rates. By incorporating more API parameters like arable land, education, health services, etc., the analysis can be expanded in the future and forecasts made.

GIT REPOSITORIES:

https://github.com/pt22aap/ADS-1-assignment