Overview: https://data.cdc.gov/api/views/9j2v-jamp/rows.json?accessType=DOWNLOAD

- Web scrape the JSON format data
- Clean the date
- Make a data frame
- Clean the data frame
- Analyze the data
- Visualize the data

1. **Importing the libraries**

```
In [1]: import numpy as np # For calculations
        import pandas as pd # For data manipulation
        import matplotlib.pyplot as plt # For visualization
        import seaborn as sns # For visualization
        import requests # For requesting access from server
        from bs4 import BeautifulSoup # For webscraping
        import re # For data cleaning
        import json # For json functions
```

2. **Getting the link and requesting access**

```
In [2]: url='https://data.cdc.gov/api/views/9j2v-jamp/rows.json?accessType=DOWNLOAD'
        req=requests.get(url)
        req
```

```
Out[2]: <Response [200]>
```

3. **Web scrapping the data**

```
In [3]: data=BeautifulSoup(req.content)
        data
```


```
      }, {
        "id" : 541989513,
        "name" : "YEAR_NUM",
        "dataTypeName" : "number",
        "description" : "Numeric code for levels of YEAR that r
e.",
        "fieldName" : "year_num",
        "position" : 9,
        "renderTypeName" : "number",
        "tableColumnId" : 137980913,
        "cachedContents" : {
          "non_null" : "6390",
          "largest" : "42",
```

4. **Getting Necessary data (<p> tag)**

```
In [5]: arr=[]
        for i in data.findAll('p'):
            arr.append(str(i))
```

5. **Accessing data**

```
In [6]: arr[0]
```

```
"rowsUpdatedBy" : "ki96-txhe",\n      "tableId" : 18340773,\n      "totalTimesRa
"viewLastModified" : 1651067439,\n      "viewType" : "tabular",\n      "approval
4,\n        "reviewedAutomatically" : true,\n        "state" : "approved",\n
ssionObject" : "public_audience_request",\n        "submissionOutcome" : "change
34,\n        "targetAudience" : "public",\n        "workflowId" : 2100,\n
onType" : "READ"\n        },\n        "submissionOutcomeApplication" : {\n
reCount" : 0,\n          "startedAt" : 1623790534,\n          "status" : "succes
"id" : "ki96-txhe",\n          "displayName" : "NCHS"\n        }\n      } ],\n
xtVariables" : [ ],\n        "inheritedVariables" : { }\n      },\n      "column
```

6. **Using re module, putting data without tags to a list**

```
In [8]: crr=[]
        for i in data.findAll('p'):
            i=re.sub('<p>|</p>','',str(i))
            crr.append(i)
```

7. **Converting JSON strings to dictionaries and appending in list**

```
In [10]: cleaned=[]
         for y in crr:
                 dicti=json.loads(y)
                 cleaned.append(dicti)
```

8. **Getting the necessary value from the key which holds necessary data**

```
In [11]: rows = []
         for z in cleaned:
             if 'data' in z:
                 rows.extend(z['data'])
```

9. **Accessing each data which will be each row**

```
In [13]: rows[3]
```

```
Out[13]: ['row-usyq~7vsy~ag46',
          '00000000-0000-0000-DAE8-3BAD17B11C1C',
          0,
          1623083971,
          None,
          1623083971,
          None,
          '{ }',
          'Death rates for suicide',
          'Deaths per 100,000 resident population, age-adjusted',
          '1',
          'Total',
          '0',
          'All persons',
          '0'
```

10. **Creating the data frame**

```
In [14]: df=pd.DataFrame(rows)
```

11. **Displaying df**

```
In [15]: df
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18
0	row-sj5k-g243-v6cb	00000000-0000-0000-ED82-9AA08DFD0A9A	0	1623083971	None	1623083971	None	{ }	Death rates for suicide	Deaths per 100,000 resident population, age-ad...	...	Total	0	All persons	0	1950	1	All ages	0
1	row-ejmi~wx6s-9cxn	00000000-0000-0000-6815-4FD0CEFCA3F6	0	1623083971	None	1623083971	None	{ }	Death rates for suicide	Deaths per 100,000 resident population, age-ad...	...	Total	0	All persons	0	1960	2	All ages	0
									Death	Deaths per									

12. **Dropping un necessary columns**

```
In [16]: df.drop([0,1,2,3,4,5,6],axis=1,inplace=True)
```

```
In [17]: # Dropping further unnecessary columns
         df.drop([7,8,9,10,11,12,14,16,18,20],axis=1,inplace=True)
```

13. After dropping

```
In [18]: df
```

Out[18]:

	13	15	17	19
0	All persons	1950	All ages	13.2
1	All persons	1960	All ages	12.5
2	All persons	1970	All ages	13.1
3	All persons	1980	All ages	12.2
4	All persons	1981	All ages	12.3
...	...	...	...	...

14. Changing column names

```
In [19]: columns=["Gender","Year","Age_Criteria","Death_Rate(Per 100,000)"]
         df.columns=columns
```

15. After changing

```
In [20]: df
```

Out[20]:

	Gender	Year	Age_Criteria	Death_Rate(Per 100,000)
0	All persons	1950	All ages	13.2
1	All persons	1960	All ages	12.5
2	All persons	1970	All ages	13.1
3	All persons	1980	All ages	12.2
4	All persons	1981	All ages	12.3
...	...	...	...	...
6385	Female: Not Hispanic or Latino: Black or Afric	2018	65 years and over	1.3

16. Checking categories of the column Gender:

```
In [21]: df["Gender"].value_counts()
```

```
Out[21]: Gender
         Female: Black or African American                    87
         Male: Hispanic or Latino: All races                  86
         Female: Hispanic or Latino: All races                86
         Male: White                                          86
         Male: American Indian or Alaska Native               86
                                                              ..
```

We can see this column has:

- Gender Info → Needed
- Race Info → Needed, but have to create another column
- Age Info → No need as data is already there in another column

17. Replacing data

```
In [22]: df["Gender"]=df["Gender"].str.replace('\d{2}-\d+ years','',regex=True)
```

```
In [24]: df["Gender"]=df["Gender"].str.replace('\d{2} years and over','',regex=True)
```

18. Creating new column for Race

```
In [26]: df["Race"]=df["Gender"]
```

19. Removing Race Info from Gender Column

```
In [28]: df["Gender"]=df["Gender"].str.replace(':.*','',regex=True)
```

20. Checking categories of Gender column now

```
In [29]: df["Gender"].value_counts()
```

```
Out[29]: Gender
         Male              2987
         Female            2731
                            588
         All persons         84
         Name: count, dtype: int64
```

21. Checking newly created Race column

```
In [32]: df["Race"].value_counts()
```

```
Out[32]: Race
                                                                    588
         Female:                                                    588
         Male:                                                      588
         Male: Black or African American:                           305
         Male: White:                                               301
         Female: Black or African American:                         176
         Female: Not Hispanic or Latino: White:                     172
         Female: White:                                             172
```

It has

- Gender info → No need as it is already there in gender column
- Race info → Needed

22. Replacing gender info from Race Column

```
In [33]: # Replacing gender info (Female)
         df["Race"]=df["Race"].str.replace('Female:','',regex=True)

In [34]: # Replacing gender info (male)
         df["Race"]=df["Race"].str.replace('Male:','',regex=True)

In [36]: # We can still some gender data so replacing
         df["Race"]=df["Race"].str.replace('Male','',regex=True)

In [37]: df["Race"]=df["Race"].str.replace('Female','',regex=True)

In [38]: df["Race"]=df["Race"].str.replace('All persons','',regex=True)
```

23. Checking after cleaning:

```
In [39]: df["Race"].value_counts()

Out[39]: Race
                                                                    1176
                                                                     840
         Black or African American:                                  481
         White:                                                      473
         Hispanic or Latino: All races:                              344
         Not Hispanic or Latino: Black or African American:          344
         Not Hispanic or Latino: White:                              344
         Asian or Pacific Islander:                                  336
         American Indian or Alaska Native:                           336
```

24. Ignoring spaces and some special characters

```
In [40]: df["Race"]=df["Race"].replace('White ','White',regex=True)

In [41]: df["Race"]=df["Race"].replace('American Indian or Alaska Native ','American Indian or Alaska Native',regex=True)

In [42]: df["Race"]=df["Race"].replace('Black or African American ','Black or African American',regex=True)

In [43]: df["Race"]=df["Race"].replace('Hispanic or Latino: All races ','Hispanic or Latino: All races',regex=True)

In [44]: ace"]=df["Race"].replace('Not Hispanic or Latino: Black or African American ','Not Hispanic or Latino: Black or A

In [45]: df["Race"]=df["Race"].replace('Not Hispanic or Latino: White ','Not Hispanic or Latino: White',regex=True)

In [46]: df["Race"]=df["Race"].replace('Asian or Pacific Islander ','Asian or Pacific Islander',regex=True)

In [47]: ce"].replace('Not Hispanic or Latino: Asian or Pacific Islander ','Not Hispanic or Latino: Asian or Pacific Islan

In [48]: df["Race"]=df["Race"].replace('Not Hispanic or Latino: Asian ','Not Hispanic or Latino: Asian',regex=True)

In [49]: df["Race"]=df["Race"].replace('Not Hispanic or Latino: Native Hawaiian or Other Pacific Islander ','Not Hispanic
```

25. Replacing Common names into one: Standardizing

```
In [89]: df["Race"]=df["Race"].replace({
' ':'Not Given',
' Not Hispanic or Latino: Black or African American: ':'Black or African American',
' Not Hispanic or Latino: White: ':'White',
' Not Hispanic or Latino: American Indian or Alaska Native: ':'American Indian or Alaska Native',
' Not Hispanic or Latino: Asianor Pacific Islander: ':'Asian or Pacific Islander',
' Not Hispanic or Latino: Asian: ':'Asian',
' Not Hispanic or Latino: Native Hawaiian or Other Pacific Islander: ':'Native Hawaiian or Other Pacific Islander'
regex=False)
```

```
In [87]: # Further cleaning for ones separated by spaces
df["Race"]=df["Race"].replace({
' White: ':'White',
"Not Given":'',
' American Indian or Alaska Native: ':'American Indian or Alaska Native',
' Black or African American: ':'Black or African American',
' Asian or Pacific Islander: ':'Asian or Pacific Islander',
' Hispanic or Latino: All races: ':'Hispanic or Latino: All races'
},regex=False)
```

26. Checking after thorough cleaning

```
: df["Race"].unique()
```

```
: array(['', 'White', 'Black or African American',
       'American Indian or Alaska Native', 'Asian or Pacific Islander',
       'Hispanic or Latino: All races', 'Asian',
       'Native Hawaiian or Other Pacific Islander'], dtype=object)
```

27. Checking data types

```
In [54]: df.dtypes

Out[54]: Gender                      object
         Year                        object
         Age_Criteria                object
         Death_Rate(Per 100,000)     object
         Race                        object
         dtype: object
```

28. Converting relevant columns

```
In [55]: df["Death_Rate(Per 100,000)"]=pd.to_numeric(df["Death_Rate(Per 100,000)"])
```

Analyzing data

29. To check the max death rate based on race

```
In [92]: df.groupby("Race")["Death_Rate(Per 100,000)"].agg('max').sort_values(ascending=False).head(10)
```

```
Out[92]: Race
White                                          74.8
                                               69.5
American Indian or Alaska Native               60.7
Native Hawaiian or Other Pacific Islander      38.1
Hispanic or Latino: All races                  28.5
Black or African American                      21.8
Asian or Pacific Islander                      20.4
Asian                                          15.5
Name: Death_Rate(Per 100,000), dtype: float64
```

30. To display max death rate based on Gender

```
In [93]: df.groupby("Gender")["Death_Rate(Per 100,000)"].agg('max').sort_values(ascending=False)
```

```
Out[93]: Gender
Male           74.8
               31.1
Female         20.7
All persons    14.8
Name: Death Rate(Per 100,000), dtype: float64
```

31. To display max death rate based on age

```
In [60]: df.groupby("Age_Criteria")["Death_Rate(Per 100,000)"].agg('max').sort_values(ascending=False)
```

```
Out[60]: Age_Criteria
85 years and over    74.8
75-84 years          61.9
15-24 years          60.7
25-44 years          60.2
65 years and over    55.8
65-74 years          53.2
55-64 years          43.6
45-64 years          39.5
All ages             34.8
45-54 years          32.0
35-44 years          28.1
20-24 years          28.0
25-34 years          27.6
15-19 years          18.1
10-14 years           3.7
Name: Death_Rate(Per 100,000), dtype: float64
```

32. To display sum of death rates per year

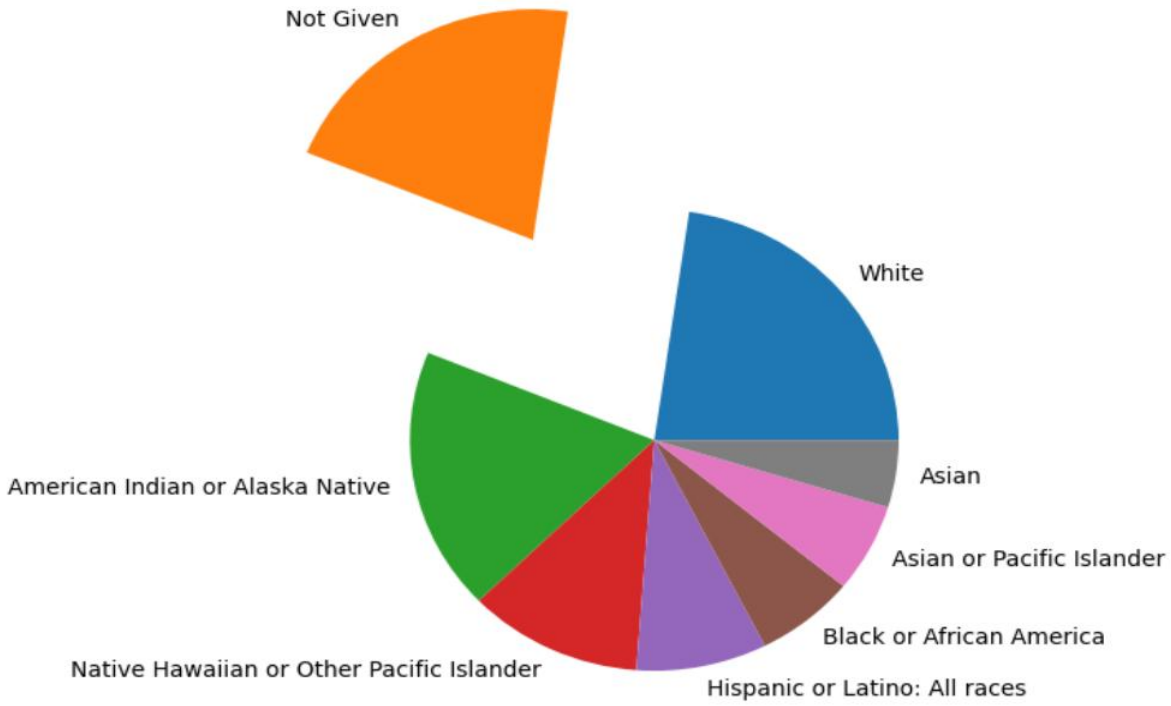
```
In [61]: df.groupby("Year")["Death_Rate(Per 100,000)"].agg('sum')
```

```
Out[61]: Year
         1950    1259.3
         1960    1180.5
         1970    1181.2
         1980    1394.0
         1981    1207.3
         1982    1224.4
         1983    1245.7
         1984    1266.9
         1985    1775.0
         1986    1653.4
         1987    1657.7
         1988    1796.1
         1989    1810.5
         1990    1816.6
         1991    1808.3
         1992    1725.8
         1993    1764.8
         1994    1746.4
         1995    1703.4
         1996    1654.7
         1997    1611.7
         1998    1592.9
         1999    1813.1
         2000    1779.4
         2001    1795.0
         2002    1815.1
```

Data Visualization:

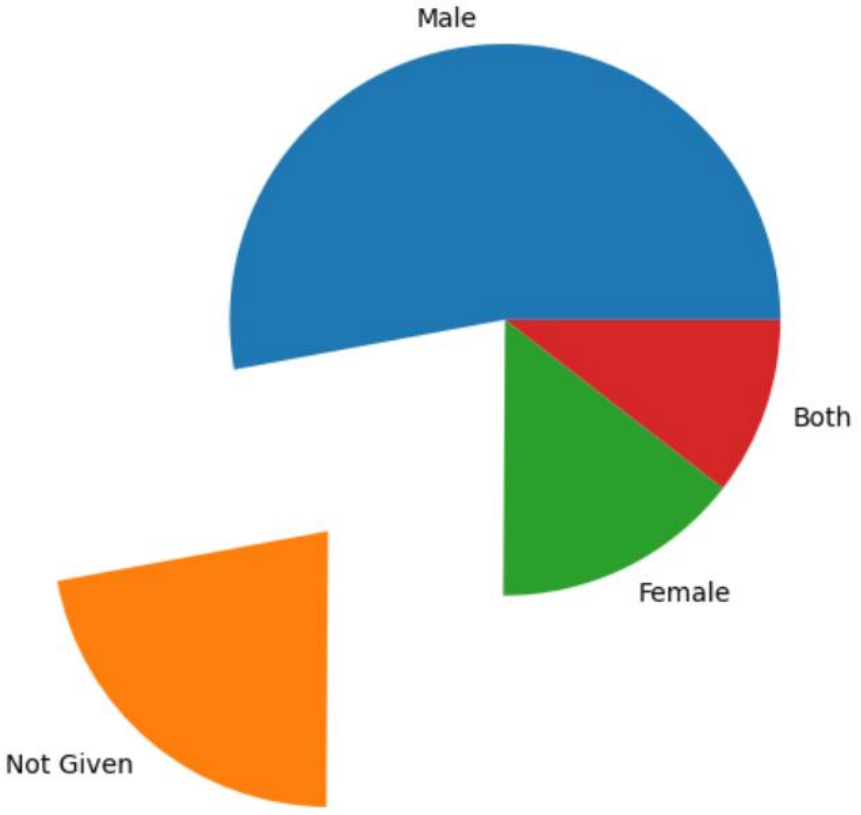
33. Pie chart for max death rate based on race

```
In [94]: plot=plt.pie(df.groupby("Race")["Death_Rate(Per 100,000)"].agg('max').sort_values(ascending=False),
                     labels=["White","Not Given","American Indian or Alaska Native",
                             "Native Hawaiian or Other Pacific Islander",
                             "Hispanic or Latino: All races","Black or African America",
                             "Asian or Pacific Islander","Asian"],explode=[0,1,0,0,0,0,0,0])
```



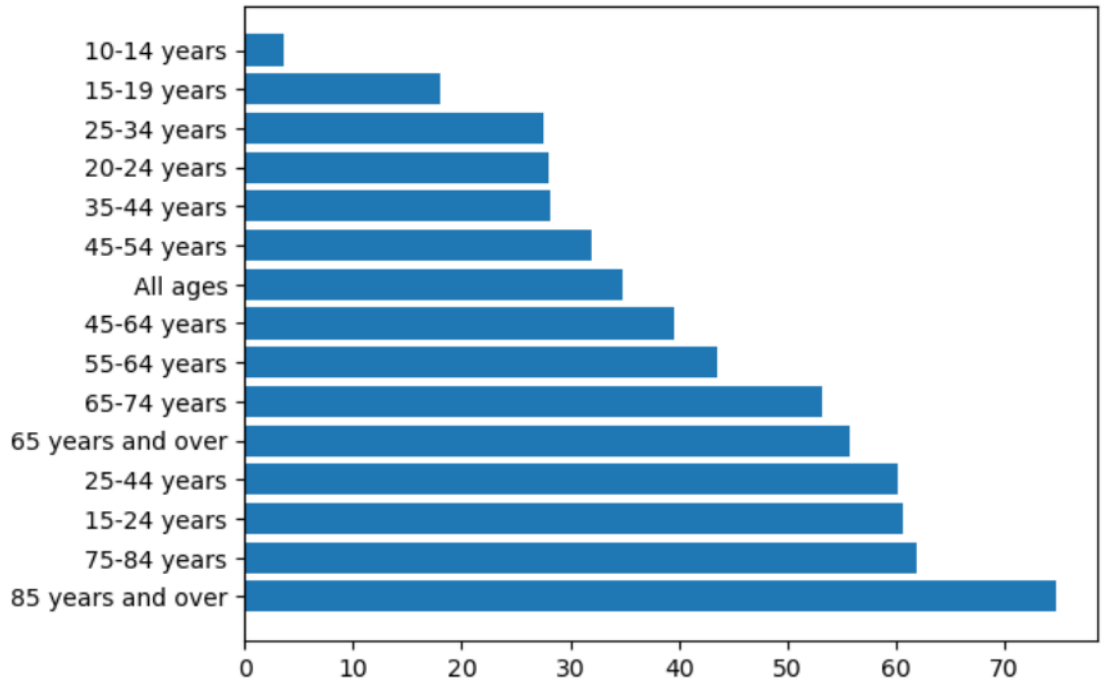
34. Pie chart for max death rate based on gender

```
In [95]: plot2=plt.pie(df.groupby("Gender")["Death_Rate(Per 100,000)"].agg('max').sort_values(ascending=False),
                       labels=["Male","Not Given","Female",'Both'],explode=[0,1,0,0])
```



35. Horizontal Bar chart for max death rate based on age criteria

```
In [96]: G=df.groupby("Age_Criteria")["Death_Rate(Per 100,000)"].agg('max').sort_values(ascending=False)
         plot3=plt.barh(G.index,G.values)
```

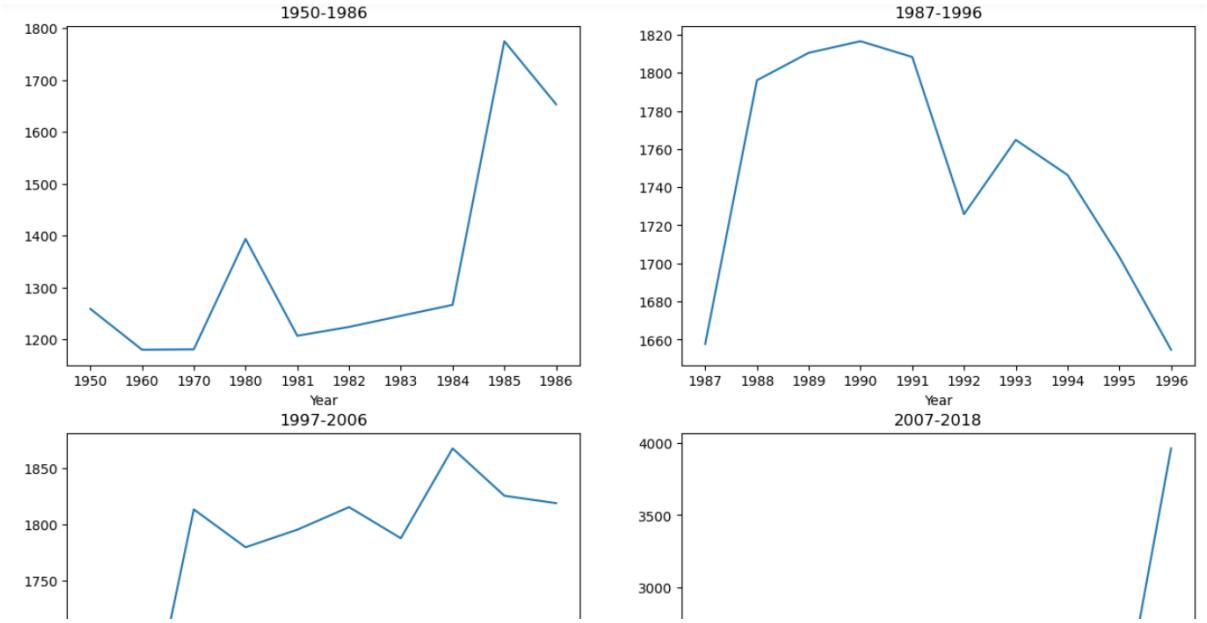


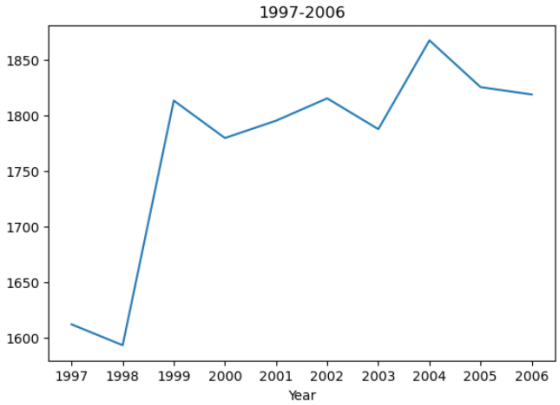
36. Slicing grouped by data due to many years

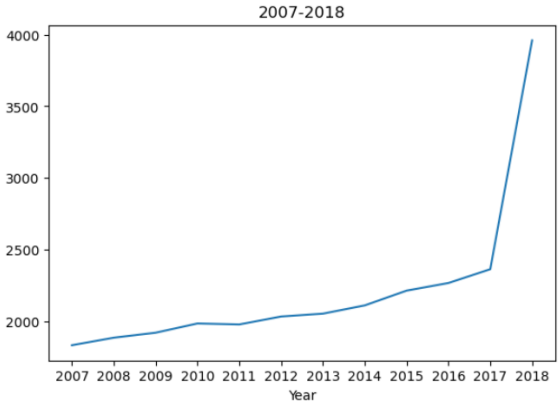
```
In [97]: D=df.groupby("Year")["Death_Rate(Per 100,000)"].agg('sum')
         E=D[0:10]
         F=D[10:20]
         G=D[20:30]
         H=D[30:42]
```

37. Using sub plots and line plots for sum death rates based on year

```
In [98]: # Increasing figure size
         plt.figure(figsize=(15,10))

         # First plot
         plt.subplot(2,2,1)
         sns.lineplot(x=E.index,y=E.values,data=E)
         plt.title("1950-1986")

         # Second Plot
         plt.subplot(2,2,2)
         sns.lineplot(x=F.index,y=F.values,data=F)
         plt.title("1987-1996")

         # Third plot
         plt.subplot(2,2,3)
         sns.lineplot(x=G.index,y=G.values,data=G)
         plt.title("1997-2006")

         # Fourth Plot
         plt.subplot(2,2,4)
         sns.lineplot(x=H.index,y=H.values,data=H)
         plt.title("2007-2018")
```




1997-2006
1850
1800
1750
1700
1650
1600
1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
Year


2007-2018
4000
3500
3000
2500
2000
2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
Year