AbhijitMandal_DSC540_Milestone5

May 30, 2021

0.0.1 DSC 540 Week 11-12 Milestone 5

Abhijit Mandal

0.0.2 Milestone 5

- Now that you have cleaned and transformed your 3 datasets, you need to load them into a database.
- You can choose what kind of database (SQLLite or MySQL, Postgre SQL are all free options).
- You will want to load each dataset into SQL Lite as an individual table and then you must join the datasets together in Python into 1 dataset.
- Once all the data is merged together in your database, create 5 visualizations that demonstrate the data you have cleansed.
- You should have at least 2 visualizations that have data from more than one source (meaning, if you have 3 tables, you must have visualizations that span across 2 of the tables – you are also welcome to use your consolidated dataset that you created in the previous step, if you do that, you have met this requirement).
- For the visualization portion of the project, you are welcome to use a python library like Matplotlib, Seaborn, or an R package ggPlot2, Plotly, or Tableau/PowerBI.

0.0.3 CSV Dataset

CSV – The Covid 19 data is scrapped from John Hopkins University github repo : https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series , this has Daily time series summary tables, including confirmed, deaths and recovered. All data is read in from the daily case report. The time series tables are subject to be updated if inaccuracies are identified in our historical data. Two time series tables are for the US confirmed cases and deaths, reported at the county level. Three time series tables are for the global confirmed cases, recovered cases and deaths. Australia, Canada and China are reported at the province/state level. Data is updated at a daily basis.

[1]:
```python
# import libraries
# for date and time opeations
from datetime import datetime, timedelta
# for file and folder operations
import os
# for regular expression operations
import re
```

```
# for listing files in a folder
import glob
# for getting web contents
import requests
# storing and analysing data
import pandas as pd
# for scraping web contents
from bs4 import BeautifulSoup
# numerical analysis
import numpy as np
```

[2]:

```
# Read dataset
conf_df = pd.read_csv('time_series_covid19_confirmed_global.csv')
deaths_df = pd.read_csv('time_series_covid19_deaths_global.csv')
recv_df = pd.read_csv('time_series_covid19_recovered_global.csv')
uscountydata = pd.read_csv("usa_county_wise.csv")
uscountydatagrouped = uscountydata.groupby(['Admin2'])['Confirmed','Deaths'].
 ↪max()
```

```
<ipython-input-2-d007be90c1a8>:6: FutureWarning: Indexing with multiple keys
(implicitly converted to a tuple of keys) will be deprecated, use a list
instead.
  uscountydatagrouped =
uscountydata.groupby(['Admin2'])['Confirmed','Deaths'].max()
```

[3]:

```
# Merge the datasets
# extract dates
dates = conf_df.columns[4:]

# melt dataframes into longer format
conf_df_long = conf_df.melt(id_vars=['Province/State', 'Country/Region', 'Lat',␣
 ↪'Long'],
                            value_vars=dates, var_name='Date',␣
 ↪value_name='Confirmed')

deaths_df_long = deaths_df.melt(id_vars=['Province/State', 'Country/Region',␣
 ↪'Lat', 'Long'],
                            value_vars=dates, var_name='Date',␣
 ↪value_name='Deaths')

recv_df_long = recv_df.melt(id_vars=['Province/State', 'Country/Region', 'Lat',␣
 ↪'Long'],
                            value_vars=dates, var_name='Date',␣
 ↪value_name='Recovered')
```

```
print(conf_df_long.shape)
print(deaths_df_long.shape)
print(recv_df_long.shape)
```

```
(125675, 6)
(125675, 6)
(118820, 6)
```

[4]:

```
# merge dataframes to get a full dataframe, we will then perform a cleanup on␣
 ↪the final dataset

full_table = pd.merge(left=conf_df_long, right=deaths_df_long, how='left',
                      on=['Province/State', 'Country/Region', 'Date', 'Lat',␣
 ↪'Long'])
full_table = pd.merge(left=full_table, right=recv_df_long, how='left',
                      on=['Province/State', 'Country/Region', 'Date', 'Lat',␣
 ↪'Long'])

full_table.head()
```

[4]:

```
  Province/State Country/Region       Lat       Long     Date  Confirmed  \
0            NaN    Afghanistan  33.93911  67.709953  1/22/20          0
1            NaN        Albania  41.15330  20.168300  1/22/20          0
2            NaN        Algeria  28.03390   1.659600  1/22/20          0
3            NaN        Andorra  42.50630   1.521800  1/22/20          0
4            NaN         Angola -11.20270  17.873900  1/22/20          0

   Deaths  Recovered
0       0        0.0
1       0        0.0
2       0        0.0
3       0        0.0
4       0        0.0
```

[5]:

```
# 1. Convert to proper date format
full_table['Date'] = pd.to_datetime(full_table['Date'])

# 2. fill na with 0
full_table['Recovered'] = full_table['Recovered'].fillna(0)

# 3. convert to int datatype
full_table['Recovered'] = full_table['Recovered'].astype('int')

# 4. fixing Country names

# 4.1 renaming countries, regions, provinces
```

```
full_table['Country/Region'] = full_table['Country/Region'].replace('Korea,␣
 ↪South', 'South Korea')

# 4.2 Greenland
full_table.loc[full_table['Province/State']=='Greenland', 'Country/Region'] =␣
 ↪'Greenland'

# 4.3 Mainland china to China
full_table['Country/Region'] = full_table['Country/Region'].replace('Mainland␣
 ↪China', 'China')



# 5. Removing county wise data to avoid double counting
full_table = full_table[full_table['Province/State'].str.contains(',')!=True]
```

[6]:

```
# Active Case = confirmed - deaths - recovered
full_table['Active'] = full_table['Confirmed'] - full_table['Deaths'] -␣
 ↪full_table['Recovered']

# filling missing values

# fill missing province/state value with ''
full_table[['Province/State']] = full_table[['Province/State']].fillna('')

# fill missing numerical values with 0
cols = ['Confirmed', 'Deaths', 'Recovered', 'Active']
full_table[cols] = full_table[cols].fillna(0)

# fixing datatypes
full_table['Recovered'] = full_table['Recovered'].astype(int)

# Viewing sample rows
full_table.sample(6)
```

[6]:

```
       Province/State Country/Region      Lat      Long        Date  Confirmed  \
122990        Guizhou          China  26.8154  106.8748  2021-04-13        147
55650                         Cyprus  35.1264   33.4299  2020-08-11       1277
109070                     Lithuania  55.1694   23.8813  2021-02-21     194051
85380                         France  46.2276    2.2137  2020-11-27    2197283
46561         Tianjin          China  39.3054  117.3230  2020-07-09        199
102271                          Togo   8.6195    0.8248  2021-01-27       4870

        Deaths  Recovered   Active
122990       2        145        0
55650       20        870      387
109070    3171     179509    11371
85380    51567     137956  2007760
```

```
46561        3        195        1
102271      76       4092      702
```

[7]:

```
# function to change value of a column in dataframe
def change_val(date, ref_col, val_col, dtnry):
    for key, val in dtnry.items():
        full_table.loc[(full_table['Date']==date) & (full_table[ref_col]==key),␣
 ↪val_col] = val
```

[8]:

```
# we found that hubei province in China has incorrect data,
# lets see what it is and will update it with correct one
# checking values
full_table[(full_table['Date']=='2/12/20') & (full_table['Province/
 ↪State']=='Hubei')]
```

[8]:

```
     Province/State Country/Region      Lat      Long       Date  Confirmed  \
5846          Hubei          China  30.9756  112.2707 2020-02-12      33366

      Deaths  Recovered  Active
5846    1068       2686   29612
```

[9]:

```
# The confirmed deaths need to be updated to 34874 as per the latest info, we␣
 ↪will do that update
feb_12_conf = {'Hubei' : 34874}
change_val('2/12/20', 'Province/State', 'Confirmed', feb_12_conf)
```

[10]:

```
# there is ship rows info which contains ships with Covid-19 reported cases
# this is an outlier for our analysis so we will remove that info from our␣
 ↪dataframe

# ship rows containing ships with COVID-19 reported cases
ship_rows = full_table['Province/State'].str.contains('Grand Princess') | \
            full_table['Province/State'].str.contains('Diamond Princess') | \
            full_table['Country/Region'].str.contains('Diamond Princess') | \
            full_table['Country/Region'].str.contains('MS Zaandam')

# ship
ship = full_table[ship_rows]

# Latest cases from the ships
ship_latest = ship[ship['Date']==max(ship['Date'])]
# ship_latest.style.background_gradient(cmap='Pastel1_r')

# skipping rows with ships info
csv_datafame = full_table[~(ship_rows)]
csv_datafame
```

[10]:

```
       Province/State      Country/Region        Lat        Long        Date  \
0                             Afghanistan  33.939110   67.709953 2020-01-22
1                                 Albania  41.153300   20.168300 2020-01-22
2                                 Algeria  28.033900    1.659600 2020-01-22
3                                 Andorra  42.506300    1.521800 2020-01-22
4                                  Angola -11.202700   17.873900 2020-01-22
…               …                 …        …         …        …
125670                            Vietnam  14.058324  108.277199 2021-04-22
125671                 West Bank and Gaza  31.952200   35.233200 2021-04-22
125672                              Yemen  15.552727   48.516388 2021-04-22
125673                             Zambia -13.133897   27.849332 2021-04-22
125674                           Zimbabwe -19.015438   29.154857 2021-04-22

        Confirmed  Deaths  Recovered  Active
0               0       0          0       0
1               0       0          0       0
2               0       0          0       0
3               0       0          0       0
4               0       0          0       0
…           …     …        …     …
125670       2824      35       2490     299
125671     287680    3115     256559   28006
125672       6020    1157       2393    2470
125673      91189    1240      89117     832
125674      38018    1555      35073    1390

[122933 rows x 9 columns]
```

0.0.4 HTML Dataset

HTML - I will be scrapping data from https://www.worldometers.info/coronavirus/#countries to get the covid details for all countries

[11]:

```
#import libraries
import os, sys
import json
import pandas as pd
import numpy as np
from numpy import int64
import requests, io
import urllib.request
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt

# Basic plotting packages
import matplotlib.pyplot as plt
# advanced ploting
import seaborn as sns
```

```
# interactive visualization
import plotly.express as px
import plotly.graph_objs as go
# import plotly.figure_factory as ff
from plotly.subplots import make_subplots
```

[12]:

```
url = "https://www.worldometers.info/coronavirus/#countries"
response = requests.get(url)
class HTMLTableParser:

    def parse_url(self, url):
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'lxml')
        return [(table['id'],self.parse_html_table(table))\
                for table in soup.find_all('table')]

# HTML parser method to clean messy data
    def parse_html_table(self, table):
        n_columns = 0
        n_rows=0
        column_names = []

        # Find number of rows and columns
        # we also find the column titles if we can
        for row in table.find_all('tr'):

            # Determine the number of rows in the table
            td_tags = row.find_all('td')
            if len(td_tags) > 0:
                n_rows+=1
                if n_columns == 0:
                    # Set the number of columns for our table
                    n_columns = len(td_tags)

            # Handle column names if we find them
            th_tags = row.find_all('th')
            if len(th_tags) > 0 and len(column_names) == 0:
                for th in th_tags:
                    colData = th.get_text()
                    colData = colData.replace('/','').replace(' ','').
 ↪replace(',','').replace('\n','').replace(' ','').replace('&nbsp;','')
                    column_names.append(colData)

        # Safeguard on Column Titles
        if len(column_names) > 0 and len(column_names) != n_columns:
            raise Exception("Column titles do not match the number of columns")
        columns = column_names if len(column_names) > 0 else range(0,n_columns)
        df = pd.DataFrame(columns = columns,
                          index= range(0,n_rows))
        row_marker = 0
        for row in table.find_all('tr'):
            column_marker = 0
            columns = row.find_all('td')
            for column in columns:
                df.iat[row_marker,column_marker] = column.get_text()
                column_marker += 1
            if len(columns) > 0:
                row_marker += 1

        # Convert to float if possible
        for col in df:
            try:
                df[col] = df[col].astype(float)
            except ValueError:
                pass

        return df
```

[13]:
```
# Parsing Html data

hp = HTMLTableParser()
table = hp.parse_url(url)[0][1] # Grabbing the table from the tuple
table.head(10)
```

[13]:
```
   #       CountryOther   TotalCases  NewCases TotalDeaths NewDeaths  \
0     \nNorth America\n   39,713,369   +12,018     893,341      +534
1            \nAsia\n   51,070,750  +225,081     682,350    +4,362
2     \nSouth America\n   28,612,928    +9,895     775,851      +179
3           \nEurope\n   46,573,598   +40,722   1,070,780      +797
4           \nAfrica\n    4,868,075    +1,965     130,525       +60
5          \nOceania\n       68,487      +119       1,252
6                  \n\n          721                    15
7                 World  170,907,928  +289,800   3,554,114    +5,932
8  1               USA   34,041,578    +6,260    609,525       +105
9  2             India   28,046,957  +153,485    329,127     +3,129

  TotalRecovered NewRecovered ActiveCases SeriousCritical  …   TotalTests  \
0     32,571,525      +25,040   6,248,503          13,897  …
1     46,963,548     +321,087   3,424,852          30,174  …
2     25,764,548       +8,388   2,072,529          29,858  …
3     43,409,761      +58,191   2,093,057          15,391  …
4      4,383,328       +2,770     354,222           2,436  …
```

```
5         66,031          +90       1,204               4  …
6            706                        0               0  …
7    153,159,447     +415,566  14,194,367          91,760  …
8     27,838,704      +19,743   5,593,349           6,141  …  479,453,309
9     25,684,529     +237,709   2,033,301           8,944  …  343,183,748

  Tests1Mpop      Population          Continent 1CaseeveryXppl  \
0                                 North America             \n
1                                          Asia             \n
2                                 South America             \n
3                                        Europe             \n
4                                        Africa             \n
5                             Australia/Oceania             \n
6                                                           \n
7                                           All             \n
8  1,440,817    332,764,957       North America             10
9    246,485  1,392,308,927                Asia             50

  1DeatheveryXppl 1TesteveryXppl NewCases1Mpop NewDeaths1Mpop ActiveCases1Mpop
0
1
2
3
4
5
6
7
8             546              1            19            0.3           16,809
9           4,230              4           110              2            1,460

[10 rows x 22 columns]
```

[14]:

```
# There are some extra special characters (\n..\n) in the dataframe.
# We need to remove the extra characters. We only need country data for mapping in this tutorial.
# So we can drop the extra top and bottom rows that we do not need for data processing.

#Drop top buttom unwanted rows
df= table.drop(table.index[[0,1,2,3,4,5,6,7]]).reset_index(drop=True)
#drop tail unwanted rows
df.drop(df.tail(8).index,inplace=True)
#drop new line '\n' charachter
df.replace(['\n'], '', regex=True, inplace=True)
df.replace([','], '', regex=True, inplace=True)
```

[15]:
```
#We need to format the table before starting mapping.
# The special characters in the dataframe can be removed using a loop as below:
# drop unwanted drop unwanted special characters using a loop
for col in df.columns[0:20]:
    df[col]=df[col].str.replace('+', '').str.replace(',', '').str.replace('N/
 ↪A', '').str.replace(' ', '').str.replace('  ', '').str.replace('&nbsp;', '')
```

[16]:
```
# All the extracted data is in text format and some column names are improper␣
 ↪for data processing.
# We need to rename some column names.

html_dataframe = df.rename(columns={'CountryOther': 'Country_Name',␣
 ↪'SeriousCritical': 'Serious_Critical', 'Tot Cases1Mpop': 'Tot_Cases_1M_pop',␣
 ↪'Deaths1Mpop': 'Deaths_1M_pop', 'Tests1Mpop': 'Tests_1M_pop'})

html_dataframe.head()
```

[16]:
```
   # Country_Name TotalCases NewCases TotalDeaths NewDeaths TotalRecovered  \
0  1          USA   34041578     6260      609525       105       27838704
1  2        India   28046957   153485      329127      3129       25684529
2  3       Brazil   16471600               461142                 14869696
3  4       France    5666113     8541      109402        44        5315194
4  5       Turkey    5242911     6933       47405       134        5105042

  NewRecovered ActiveCases Serious_Critical  … TotalTests Tests_1M_pop  \
0        19743     5593349             6141  …  479453309      1440817
1       237709     2033301             8944  …  343183748       246485
2                  1140762             8318  …   49076549       229404
3         3021      241517             2993  …   84787739      1296354
4        10763       90464             1390  …   53919848       633145

   Population     Continent 1CaseeveryXppl 1DeatheveryXppl 1TesteveryXppl  \
0   332764957  NorthAmerica             10             546              1
1  1392308927          Asia             50            4230              4
2   213930804  SouthAmerica             13             464              4
3    65404780        Europe             12             598              1
4    85161879          Asia             16            1796              2

  NewCases1Mpop NewDeaths1Mpop ActiveCases1Mpop
0            19            0.3            16809
1           110              2             1460
2                                          5332
3           131            0.7             3693
4            81              2             1062

[5 rows x 22 columns]
```

0.0.5 API Dataset

API - I will be fetchng data from https://api.census.gov/data/2019/acs/acs1/profile?get=NAME,gr which will provide the demographic info (Age, employment, sex, ethnicity etc) for US at a County level for all the corona virus cases, this information is vital to understand the rate of spread across communities in United States. I will use this data to deep dive into corona cases in US and generate some interesting facts.

0.0.6 Load the necessary libraries.

[17]:

```
#import libraries
import urllib.request, urllib.parse, urllib.error
import json
import requests
import numpy as np
import pandas as pd
#pandasql package allows us to write SQL querry on Pandas DataFrame
import pandasql as psql
import seaborn as sns
import matplotlib.pyplot as plt
```

0.0.7 2. Reading the API data

[18]:

```

apiURl = "https://api.census.gov/data/2019/acs/acs1/profile?
 ↪get=NAME,group(DP02)&for=county:*"

filename = "acs2019_county_data.csv"
chunk_size = 100

response = requests.get(apiURl)

# calling this API and saving it as CSV
with open(filename, 'wb') as fd:
    for chunk in response.iter_content(chunk_size):
        fd.write(chunk)
```

[19]:

```
county_2019 = pd.read_csv('acs2019_county_data.csv', encoding='latin-1')
county_2019.head()
```

[19]:

```
   CountyId    State          County  TotalPop    Men   Women  Hispanic  \
0      1001  Alabama  Autauga County     55036  26899   28137       2.7
1      1003  Alabama  Baldwin County    203360  99527  103833       4.4
2      1005  Alabama  Barbour County     26201  13976   12225       4.2
3      1007  Alabama     Bibb County     22580  12251   10329       2.4
4      1009  Alabama   Blount County     57667  28490   29177       9.0
```

```
   White  Black  Native  …  Walk  OtherTransp  WorkAtHome  MeanCommute  \
0   75.4   18.9     0.3  …   0.6          1.3         2.5         25.8
1   83.1    9.5     0.8  …   0.8          1.1         5.6         27.0
2   45.7   47.8     0.2  …   2.2          1.7         1.3         23.4
3   74.6   22.0     0.4  …   0.3          1.7         1.5         30.0
4   87.4    1.5     0.3  …   0.4          0.4         2.1         35.0

   Employed  PrivateWork  PublicWork  SelfEmployed  FamilyWork  Unemployment
0     24112         74.1        20.2           5.6         0.1           5.2
1     89527         80.7        12.9           6.3         0.1           5.5
2      8878         74.1        19.1           6.5         0.3          12.4
3      8171         76.0        17.4           6.3         0.3           8.2
4     21380         83.9        11.9           4.0         0.1           4.9

[5 rows x 37 columns]
```

```
[20]: county_2019.describe(include = 'all')
```

```
[20]:             CountyId  State             County      TotalPop           Men  \
count     3220.000000   3220               3220  3.220000e+03  3.220000e+03
unique            NaN     52               1955           NaN           NaN
top               NaN  Texas  Washington County           NaN           NaN
freq              NaN    254                 30           NaN           NaN
mean     31393.605280    NaN                NaN  1.007681e+05  4.958781e+04
std      16292.078954    NaN                NaN  3.244996e+05  1.593212e+05
min       1001.000000    NaN                NaN  7.400000e+01  3.900000e+01
25%      19032.500000    NaN                NaN  1.121350e+04  5.645500e+03
50%      30024.000000    NaN                NaN  2.584750e+04  1.287900e+04
75%      46105.500000    NaN                NaN  6.660825e+04  3.301725e+04
max      72153.000000    NaN                NaN  1.010572e+07  4.979641e+06

               Women     Hispanic        White        Black       Native  …  \
count   3.220000e+03  3220.000000  3220.000000  3220.000000  3220.000000  …
unique           NaN          NaN          NaN          NaN          NaN  …
top              NaN          NaN          NaN          NaN          NaN  …
freq             NaN          NaN          NaN          NaN          NaN  …
mean    5.118032e+04    11.296584    74.920186     8.681957     1.768416  …
std     1.652164e+05    19.342522    23.056700    14.333571     7.422946  …
min     3.500000e+01     0.000000     0.000000     0.000000     0.000000  …
25%     5.553500e+03     2.100000    63.500000     0.600000     0.100000  …
50%     1.299350e+04     4.100000    83.600000     2.000000     0.300000  …
75%     3.359375e+04    10.000000    92.800000     9.500000     0.600000  …
max     5.126081e+06   100.000000   100.000000    86.900000    90.300000  …

               Walk  OtherTransp   WorkAtHome  MeanCommute      Employed  \
count   3220.000000  3220.000000  3220.000000  3220.000000  3.220000e+03
unique          NaN          NaN          NaN          NaN           NaN
```

```
top             NaN         NaN         NaN         NaN           NaN
freq            NaN         NaN         NaN         NaN           NaN
mean       3.244472    1.598696    4.736894   23.474534  4.709295e+04
std        3.891510    1.678232    3.073484    5.687241  1.558159e+05
min        0.000000    0.000000    0.000000    5.100000  3.900000e+01
25%        1.400000    0.800000    2.900000   19.600000  4.573000e+03
50%        2.300000    1.300000    4.100000   23.200000  1.061150e+04
75%        3.825000    1.900000    5.800000   27.000000  2.874725e+04
max       59.200000   43.200000   33.000000   45.100000  4.805817e+06

        PrivateWork  PublicWork  SelfEmployed  FamilyWork  Unemployment
count   3220.000000 3220.000000   3220.000000 3220.000000   3220.000000
unique          NaN         NaN           NaN         NaN           NaN
top             NaN         NaN           NaN         NaN           NaN
freq            NaN         NaN           NaN         NaN           NaN
mean      74.863323   17.086118      7.772733    0.278820      6.665590
std        7.647916    6.390868      3.855454    0.448073      3.772612
min       31.100000    4.400000      0.000000    0.000000      0.000000
25%       71.200000   12.700000      5.200000    0.100000      4.475000
50%       76.100000   15.900000      6.800000    0.200000      6.100000
75%       80.200000   19.900000      9.200000    0.300000      8.000000
max       88.800000   64.800000     38.000000    8.000000     40.900000

[11 rows x 37 columns]
```

```
[21]: # Handling Missing Values and Formatting
      # We have one missing value in the child poverty column. We fill this with 0.

      #Checking missing Data
      null_2019 = psql.sqldf("SELECT State, County, TotalPop, Income, IncomeErr,␣
       ↪Poverty, ChildPoverty\
                              FROM county_2019\
                              WHERE ChildPoverty IS NULL")
      null_2019
```

```
[21]:     State          County  TotalPop  Income  IncomeErr  Poverty ChildPoverty
      0  Hawaii  Kalawao County        86   61750      11280     12.7         None
```

```
[22]: # Fill missing value in ChildPoverty with zero
      county_2019.ChildPoverty.fillna(0)
```

```
[22]: 0       20.1
      1       16.1
      2       44.9
      3       26.6
      4       25.4
              …
```

```
3215    49.4
3216    68.2
3217    67.9
3218    62.1
3219    58.2
Name: ChildPoverty, Length: 3220, dtype: float64
```

[23]:

```
#subsetting dataset toget relevant columns
County2019 = county_2019[['CountyId', 'State', 'County', 'Men',␣
 ↪'Women','White', 'Black','Native','Hispanic', 'Asian','Pacific','TotalPop',␣
 ↪'IncomePerCap', 'Poverty', 'ChildPoverty', 'Employed', 'SelfEmployed',␣
 ↪'Unemployment']]
County2019.head()
```

[23]:

```
   CountyId    State           County    Men   Women  White  Black  Native  \
0      1001  Alabama   Autauga County  26899   28137   75.4   18.9     0.3
1      1003  Alabama   Baldwin County  99527  103833   83.1    9.5     0.8
2      1005  Alabama   Barbour County  13976   12225   45.7   47.8     0.2
3      1007  Alabama      Bibb County  12251   10329   74.6   22.0     0.4
4      1009  Alabama    Blount County  28490   29177   87.4    1.5     0.3

   Hispanic  Asian  Pacific  TotalPop  IncomePerCap  Poverty  ChildPoverty  \
0       2.7    0.9      0.0     55036         27824     13.7          20.1
1       4.4    0.7      0.0    203360         29364     11.8          16.1
2       4.2    0.6      0.0     26201         17561     27.2          44.9
3       2.4    0.0      0.0     22580         20911     15.2          26.6
4       9.0    0.1      0.0     57667         22021     15.6          25.4

   Employed  SelfEmployed  Unemployment
0     24112           5.6           5.2
1     89527           6.3           5.5
2      8878           6.5          12.4
3      8171           6.3           8.2
4     21380           4.0           4.9
```

[24]:

```
# Adding Calculated column for Men and Women in percentage
pd.options.mode.chained_assignment = None  # default='warn'
County2019['MenPercentage'] = (County2019.Men / County2019.TotalPop)*100
County2019['WomenPercentage'] = (County2019.Women / County2019.TotalPop)*100

api_dataframe = County2019
api_dataframe.head()
```

[24]:

```
   CountyId    State           County    Men   Women  White  Black  Native  \
0      1001  Alabama   Autauga County  26899   28137   75.4   18.9     0.3
1      1003  Alabama   Baldwin County  99527  103833   83.1    9.5     0.8
2      1005  Alabama   Barbour County  13976   12225   45.7   47.8     0.2
```

```
3      1007  Alabama     Bibb County  12251   10329   74.6   22.0     0.4
4      1009  Alabama   Blount County  28490   29177   87.4    1.5     0.3

   Hispanic  Asian  Pacific  TotalPop  IncomePerCap  Poverty  ChildPoverty  \
0       2.7    0.9      0.0     55036         27824     13.7          20.1
1       4.4    0.7      0.0    203360         29364     11.8          16.1
2       4.2    0.6      0.0     26201         17561     27.2          44.9
3       2.4    0.0      0.0     22580         20911     15.2          26.6
4       9.0    0.1      0.0     57667         22021     15.6          25.4

   Employed  SelfEmployed  Unemployment  MenPercentage  WomenPercentage
0     24112           5.6           5.2      48.875282        51.124718
1     89527           6.3           5.5      48.941286        51.058714
2      8878           6.5          12.4      53.341476        46.658524
3      8171           6.3           8.2      54.255979        45.744021
4     21380           4.0           4.9      49.404339        50.595661
```

[25]:

```
# Aggregating csv dataset based on
cols = ['Country/Region','Date','Confirmed','Deaths', 'Recovered', 'Active']
csv_selected_datafame = csv_datafame[cols]
csv_agg = csv_selected_datafame.groupby(['Country/Region','Confirmed',␣
 ↪'Deaths', 'Recovered', 'Active'])['Date'].max()
csv_agg = csv_selected_datafame.groupby(['Country/Region'])['Confirmed',␣
 ↪'Deaths', 'Recovered', 'Active'].sum()
csv_agg['Confirmed'] = csv_agg['Confirmed']/100
csv_agg['Deaths'] = csv_agg['Deaths']/100
csv_agg['Recovered'] = csv_agg['Recovered']/100
csv_agg['Active'] = csv_agg['Active']/100
```

```
<ipython-input-25-1468e15b545d>:5: FutureWarning: Indexing with multiple keys
(implicitly converted to a tuple of keys) will be deprecated, use a list
instead.
  csv_agg = csv_selected_datafame.groupby(['Country/Region'])['Confirmed',
'Deaths', 'Recovered', 'Active'].sum()
```

[26]:

```
# Adding Country column in API dataset
api_dataframe['Country']='USA'
api_dataframe['CountryShort']='US'
```

### Inserting all the 3 datasets into MySQL table

[27]:

```
# importing sqlite3 library
import sqlalchemy
from sqlalchemy import create_engine , select, MetaData, Table, and_
import pandas as pd
import sqlite3
```

```
engine = sqlalchemy.create_engine('sqlite:///covidWorldDb.db', echo=False)
```

[28]:
```
csv_agg.to_sql('CSVData', con=engine, if_exists='append')
```

[29]:
```
html_dataframe.to_sql('HTMLData', con=engine, if_exists='append')
```

[30]:
```
api_dataframe.to_sql('APIData', con=engine, if_exists='append')
```

[31]:
```
    uscountydata.to_sql('CSVUSCounty', con=engine, if_exists='append')
```

[32]:
```
uscountydatagrouped.to_sql('CSVUSCountyGrouped', con=engine, if_exists='append')
```

[33]:
```
# Getting the US Country Covid and Demographic data by joining CSV and API␣
 ↪Dataset
conn = sqlite3.connect('covidWorldDb.db')

# Creating CSV US County dataset
query = conn.execute('''SELECT a.* , b.* FROM CSVUSCountyGrouped as a inner␣
 ↪join APIData as b on a.[Admin2] = b.County;''')

cols = [column[0] for column in query.description]
uscountyAPIDF = pd.DataFrame.from_records(data = query.fetchall(), columns =␣
 ↪cols)

#commit the changes to db
conn.commit()
#close the connection
conn.close()
```

[34]:
```
# Getting the US Country Covid Dataset
conn = sqlite3.connect('covidWorldDb.db')

# Creating CSV US County dataset
query = conn.execute('''SELECT a.* FROM CSVUSCounty as a;''')

cols = [column[0] for column in query.description]
uscountyDF = pd.DataFrame.from_records(data = query.fetchall(), columns = cols)

#commit the changes to db
conn.commit()
#close the connection
conn.close()
```

[35]:
```
conn = sqlite3.connect('covidWorldDb.db')
#c = conn.cursor()

# Joining from CSV and HTML dataset
```

```
query = conn.execute('''SELECT a.*, b.* FROM CSVData as a inner join HTMLData␣
 ↪as b on a.[Country/Region] = b.Country_Name;''')

#csvhtmlrows = c.fetchall()

cols = [column[0] for column in query.description]
csvhtmlDF = pd.DataFrame.from_records(data = query.fetchall(), columns = cols)

#for row in csvhtmlrows:
#    print(row)

#commit the changes to db
conn.commit()
#close the connection
conn.close()
```

[36]:

```
conn = sqlite3.connect('covidWorldDb.db')
#c = conn.cursor()

# Joining from CSV and HTML dataset
query = conn.execute('''SELECT a.* FROM CSVData as a;''')

#csvhtmlrows = c.fetchall()

cols = [column[0] for column in query.description]
csvDF = pd.DataFrame.from_records(data = query.fetchall(), columns = cols)

#for row in csvhtmlrows:
#    print(row)

#commit the changes to db
conn.commit()
#close the connection
conn.close()
```

[37]:

```
conn = sqlite3.connect('covidWorldDb.db')
#c = conn.cursor()

# Joining from CSV and HTML dataset
queryApi = conn.execute('''SELECT a.*, b.* FROM APIData as a inner join␣
 ↪HTMLData as b on a.[Country] = b.Country_Name;''')

#apihtmlrows = c.fetchall()
cols = [column[0] for column in queryApi.description]
apihtmlDF = pd.DataFrame.from_records(data = queryApi.fetchall(), columns =␣
 ↪cols)
```

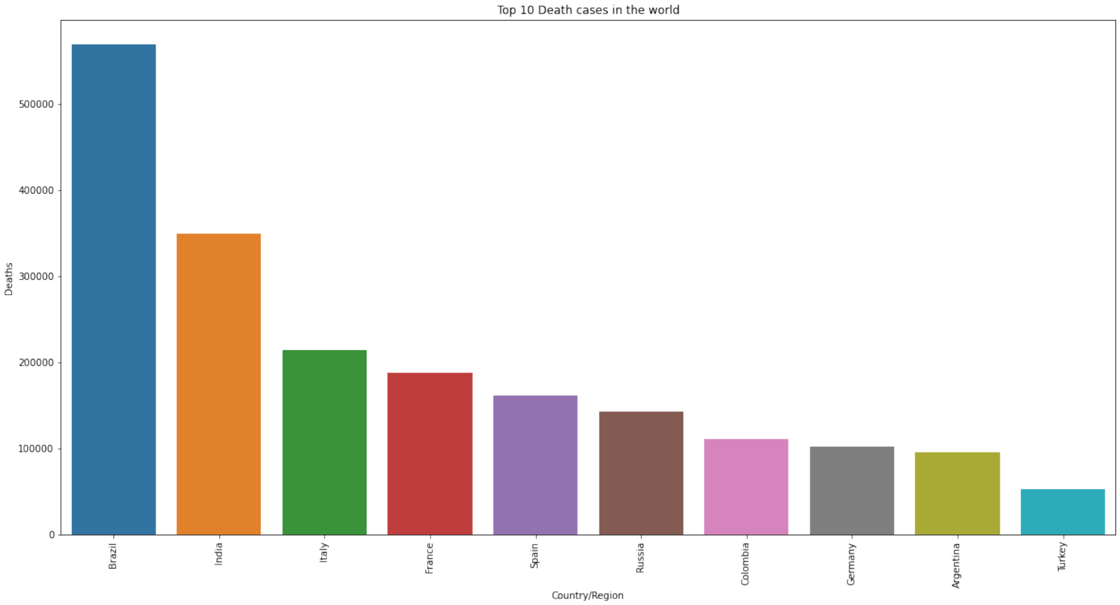
```
#for row in apihtmlrows:
#    print(row)

#commit the changes to db
conn.commit()
#close the connection
conn.close()
```

[38]:
```
conn = sqlite3.connect('covidWorldDb.db')
#c = conn.cursor()

# Joining from CSV and HTML dataset
queryApiCsv = conn.execute('''SELECT a.*, b.* FROM APIData as a inner join␣
 ↪CSVData as b on a.[CountryShort] = b.[Country/Region];''')

#apihtmlrows = c.fetchall()
cols = [column[0] for column in queryApiCsv.description]
apicsvDF = pd.DataFrame.from_records(data = queryApiCsv.fetchall(), columns =␣
 ↪cols)

#for row in apihtmlrows:
#    print(row)

#commit the changes to db
conn.commit()
#close the connection
conn.close()
```

[39]:
```
# Removing unwanted columns
del csvhtmlDF['Country_Name']
del apicsvDF['Country']
del apihtmlDF['Country_Name']
del csvhtmlDF['index']
del apicsvDF['index']
del apihtmlDF['index']
```

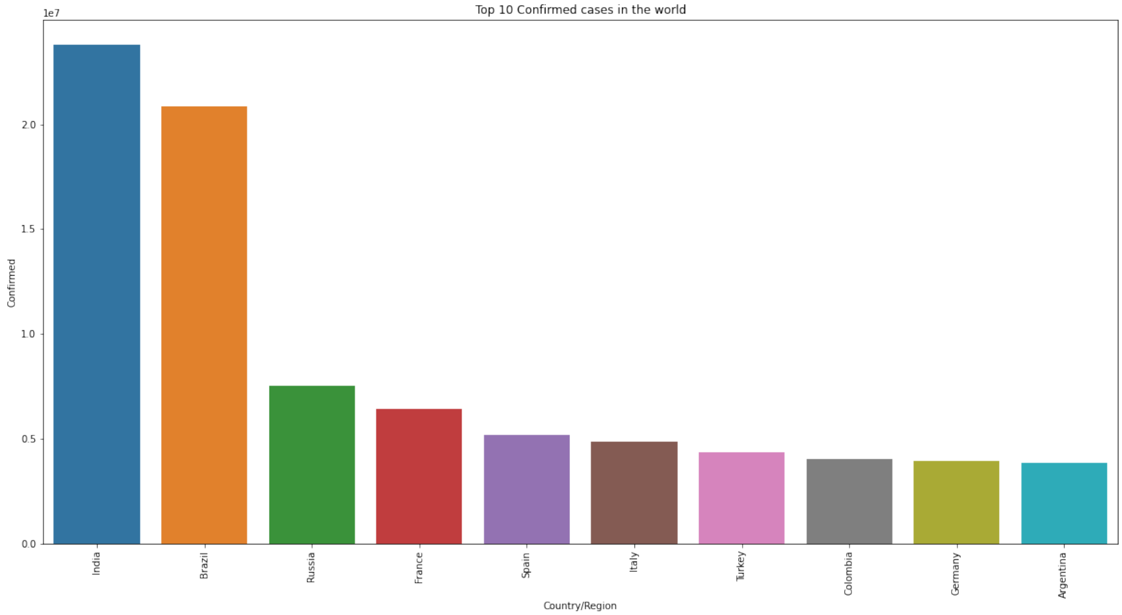
[60]:
```
# Display Top 10 Deaths in the world
sorted_data = csvhtmlDF.head(10).sort_values("Deaths", ascending = False)
plt.figure(figsize=(20,10))
sns.barplot(x=sorted_data['Country/Region'], y=sorted_data['Deaths'])
plt.xticks(rotation= 90)
plt.xlabel('Country/Region')
plt.ylabel('Deaths')
plt.title('Top 10 Death cases in the world')
```

[60]:
```
Text(0.5, 1.0, 'Top 10 Death cases in the world')
```



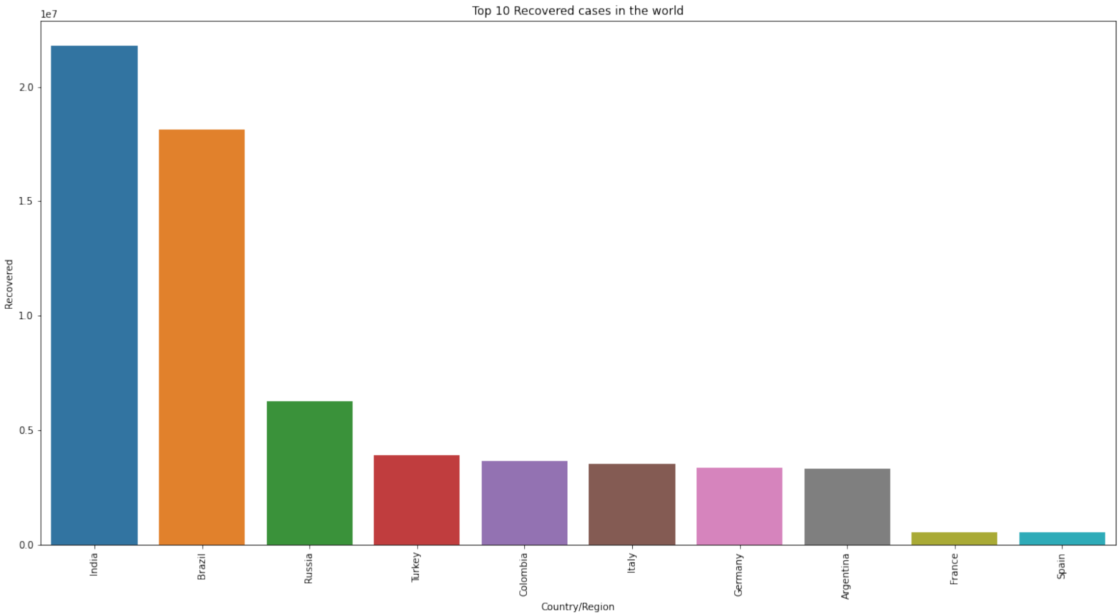
[61]:
```
# dislay confirmed in first 10 countries



sorted_data = csvhtmlDF.head(10).sort_values("Confirmed", ascending = False)
plt.figure(figsize=(20,10))
sns.barplot(x=sorted_data['Country/Region'], y=sorted_data['Confirmed'])
plt.xticks(rotation= 90)
plt.xlabel('Country/Region')
plt.ylabel('Confirmed')
plt.title('Top 10 Confirmed cases in the world')
```

[61]:
```
Text(0.5, 1.0, 'Top 10 Confirmed cases in the world')
```



[63]:
```
# Top 10 recovered countries in the world
#dislay Recovered in first 10 countries

sorted_data = csvhtmlDF.head(10).sort_values("Recovered", ascending = False)
plt.figure(figsize=(20,10))
sns.barplot(x=sorted_data['Country/Region'], y=sorted_data['Recovered'])
plt.xticks(rotation= 90)
plt.xlabel('Country/Region')
plt.ylabel('Recovered')
plt.title('Top 10 Recovered cases in the world')
```

[63]:
```
Text(0.5, 1.0, 'Top 10 Recovered cases in the world')
```

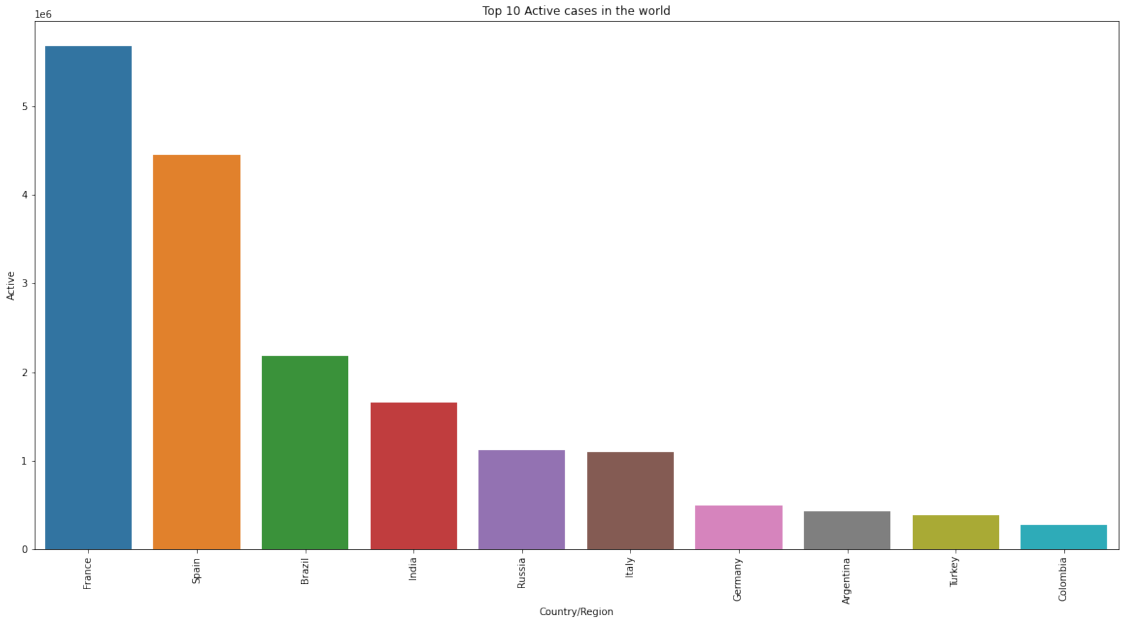


[64]:

```
## Visualizing Data
# Lets examine the top 5 countries with max confirmed numbers.




sorted_data = csvhtmlDF.head(10).sort_values("Active", ascending = False)
plt.figure(figsize=(20,10))
sns.barplot(x=sorted_data['Country/Region'], y=sorted_data['Active'])
plt.xticks(rotation= 90)
plt.xlabel('Country/Region')
plt.ylabel('Active')
plt.title('Top 10 Active cases in the world')
```

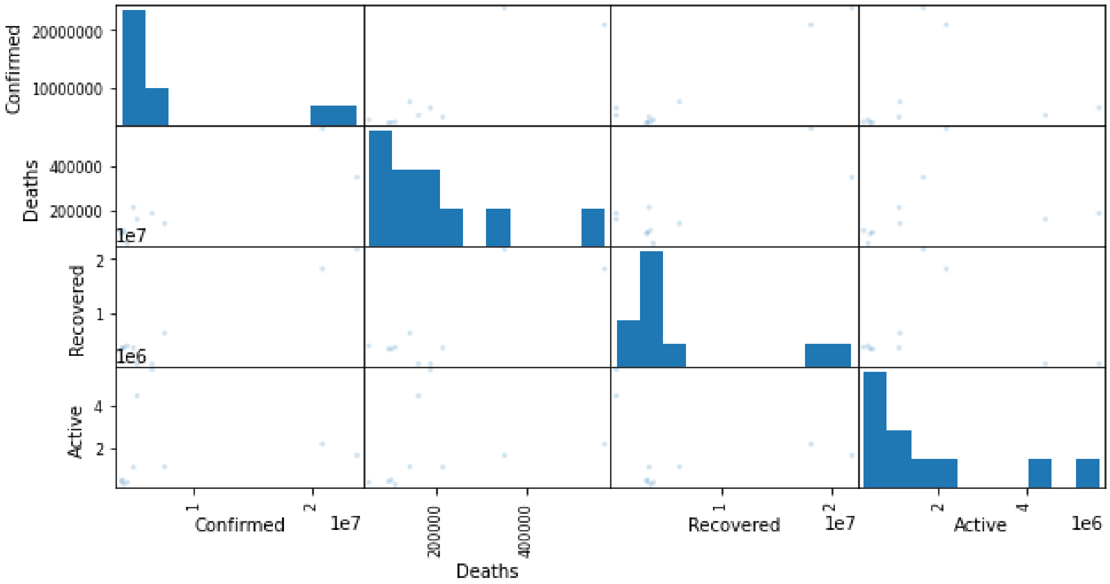
[64]:

```
Text(0.5, 1.0, 'Top 10 Active cases in the world')
```



```
[66]: #Scatter matrix plot between the confirmed, deaths and recovery category.
      #fig = px.scatter_matrix(csvhtmlDF.head(10), dimensions=['Confirmed', 'Deaths',␣
       ↪'Recovered', 'Active'], color = "Country/Region")
      #fig

      df = pd.DataFrame(csvhtmlDF.head(10), columns=['Confirmed', 'Deaths',␣
       ↪'Recovered', 'Active'])
      pd.plotting.scatter_matrix(df, alpha=0.2)
```

```
[66]: array([[<AxesSubplot:xlabel='Confirmed', ylabel='Confirmed'>,
              <AxesSubplot:xlabel='Deaths', ylabel='Confirmed'>,
              <AxesSubplot:xlabel='Recovered', ylabel='Confirmed'>,
              <AxesSubplot:xlabel='Active', ylabel='Confirmed'>],
             [<AxesSubplot:xlabel='Confirmed', ylabel='Deaths'>,
              <AxesSubplot:xlabel='Deaths', ylabel='Deaths'>,
              <AxesSubplot:xlabel='Recovered', ylabel='Deaths'>,
              <AxesSubplot:xlabel='Active', ylabel='Deaths'>],
             [<AxesSubplot:xlabel='Confirmed', ylabel='Recovered'>,
              <AxesSubplot:xlabel='Deaths', ylabel='Recovered'>,
              <AxesSubplot:xlabel='Recovered', ylabel='Recovered'>,
              <AxesSubplot:xlabel='Active', ylabel='Recovered'>],
             [<AxesSubplot:xlabel='Confirmed', ylabel='Active'>,
              <AxesSubplot:xlabel='Deaths', ylabel='Active'>,
              <AxesSubplot:xlabel='Recovered', ylabel='Active'>,
              <AxesSubplot:xlabel='Active', ylabel='Active'>]], dtype=object)
```



```
[45]: # Defining COVID-19 cases as per classifications
      cases = ['Confirmed', 'Deaths', 'Recovered', 'Active']
      full_table = csvhtmlDF
      # Defining Active Case: Active Case = confirmed - deaths - recovered
      #full_table['Active'] = full_table['TotalCases'] - full_table['TotalDeaths'] -␣
       ↪full_table['TotalRecovered'

      # cases in the ships
      #ship = full_table[full_table['Province/State'].str.contains('Grand␣
       ↪Princess')|full_table['Country/Region'].str.contains('Cruise Ship')]

      # china and the row
      china = full_table[full_table['Country/Region']=='China']
      row = full_table[full_table['Country/Region']!='China']

      # latest
      full_latest = full_table
      #full_table = [full_table['Date'] == max(full_table['Date'])].reset_index()
      #china_latest = full_table[full_latest['Country/Region']=='China']
      row_latest = full_table[full_latest['Country/Region']!='China']

      # latest condensed
      full_latest_grouped = full_latest.groupby('Country/Region')['Confirmed',␣
       ↪'Deaths', 'Recovered', 'Active'].sum().reset_index()
      #china_latest_grouped = china_latest.groupby('Province/State')['Confirmed',␣
       ↪'Deaths', 'Recovered', 'Active'].sum().reset_index()
```

```
row_latest_grouped = row_latest.groupby('Country/Region')['Confirmed',␣
 ↪'Deaths', 'Recovered', 'Active'].sum().reset_index()
```

```
<ipython-input-45-5072e2cda85b>:21: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be
deprecated, use a list instead.

<ipython-input-45-5072e2cda85b>:23: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be
deprecated, use a list instead.
```

[46]:
```
temp = full_table.groupby(['Country/Region'])['Confirmed', 'Deaths',␣
 ↪'Recovered', 'Active'].max()
#temp = full_table.groupby('Date')['Confirmed', 'Deaths', 'Recovered',␣
 ↪'Active'].sum().reset_index()
#temp = temp[temp['Date']==max(temp['Date'])].reset_index(drop=True)
temp.style.background_gradient(cmap='Pastel1')
```

```
<ipython-input-46-bb8c44304190>:1: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be
deprecated, use a list instead.
```

[46]:
```
<pandas.io.formats.style.Styler at 0x7fb069ed9af0>
```

[47]:
```
temp_f = full_latest_grouped.sort_values(by='Confirmed', ascending=False)
temp_f = temp_f.reset_index(drop=True)
temp_f.head(11).style.background_gradient(cmap='Reds')
```

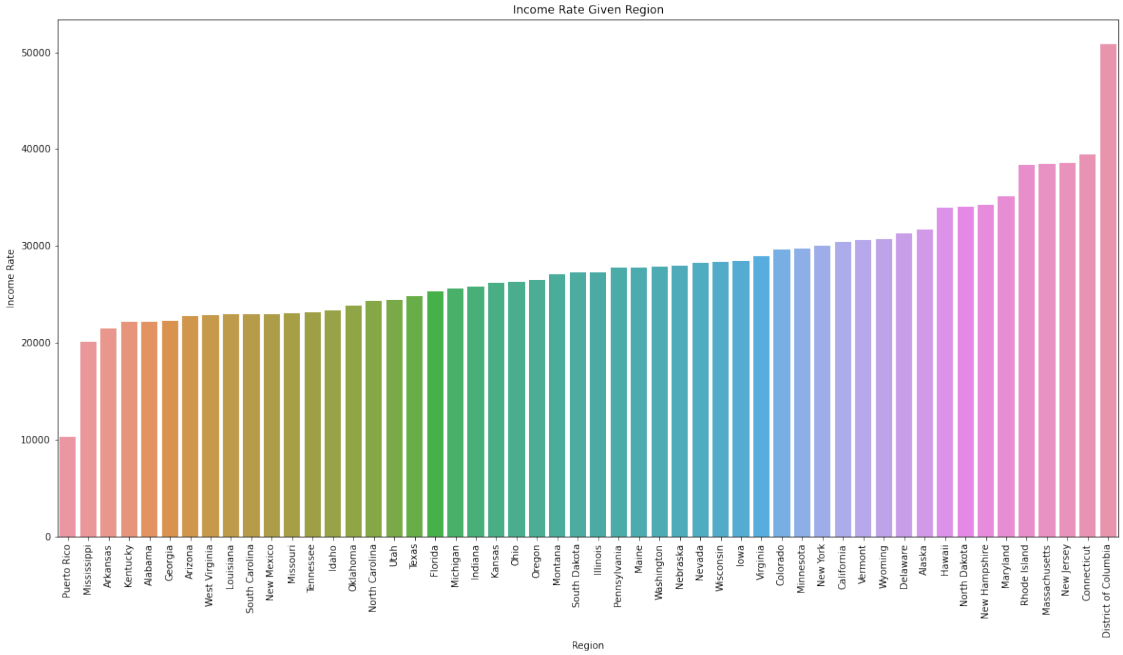
[47]:
```
<pandas.io.formats.style.Styler at 0x7fb053806910>
```

[48]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
%matplotlib inline
```

[49]:
```
region_list = list(apicsvDF['State'].unique())
region_income_ratio = []
for i in region_list:
    x = apicsvDF[apicsvDF['State']==i]                      #Find the state␣
 ↪have how many county
    region_income_rate = sum(x.IncomePerCap)/len(x)                #Then␣
 ↪calculate sum of income ratio and divided to found above
```

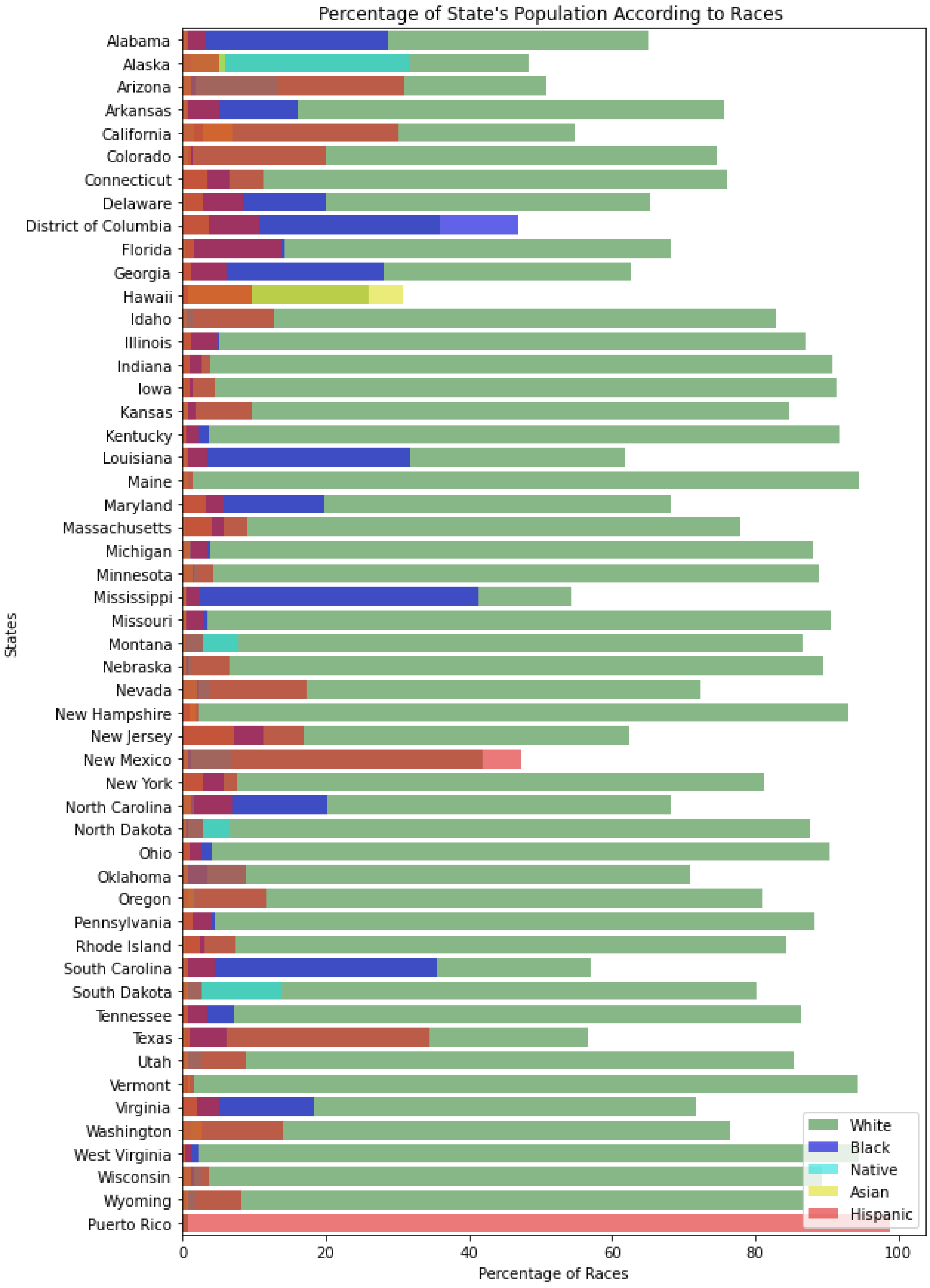
```
    region_income_ratio.append(region_income_rate)            #You append to␣
 ↪list the state


#sorting
#Sort the income ratio as from low to high
#If you change the ascending state as False,Sorting will change as from high to␣
 ↪low
data = pd.DataFrame({'region_list': region_list,'region_income_ratio':
 ↪region_income_ratio})
new_index = (data['region_income_ratio'].sort_values(ascending=True)).index.
 ↪values
sorted_data = data.reindex(new_index)

# visualization
plt.figure(figsize=(20,10))
sns.barplot(x=sorted_data['region_list'], y=sorted_data['region_income_ratio'])
plt.xticks(rotation= 90)
plt.xlabel('Region')
plt.ylabel('Income Rate')
plt.title('Income Rate Given Region')
```

```
[49]: Text(0.5, 1.0, 'Income Rate Given Region')
```



```
[50]: #Horizontal Bar Plot
      area_list = list(apicsvDF['State'].unique())

      #We create 5 empty list to keep each races
      share_white = []
      share_black = []
      share_native_american = []
      share_asian = []
      share_hispanic = []

      #Find the number of each races in the States
      for i in area_list:
          x = apicsvDF[apicsvDF['State']==i]
          share_white.append(sum(x.White)/len(x))
          share_black.append(sum(x.Black) / len(x))
          share_native_american.append(sum(x.Native) / len(x))
          share_asian.append(sum(x.Asian) / len(x))
          share_hispanic.append(sum(x.Hispanic) / len(x))

      # visualization
      f,ax = plt.subplots(figsize = (9,15))
      sns.barplot(x=share_white,y=area_list,color='green',alpha = 0.5,label='White' )
      sns.barplot(x=share_black,y=area_list,color='blue',alpha = 0.7,label='Black')
      sns.barplot(x=share_native_american,y=area_list,color='cyan',alpha = 0.
       ↪6,label='Native')
      sns.barplot(x=share_asian,y=area_list,color='yellow',alpha = 0.6,label='Asian')
      sns.barplot(x=share_hispanic,y=area_list,color='red',alpha = 0.
       ↪6,label='Hispanic')

      ax.legend(loc='lower right',frameon = True)     # legendlarin gorunurlugu
      ax.set(xlabel='Percentage of Races', ylabel='States',title = "Percentage of␣
       ↪State's Population According to Races ")
```

```
[50]: [Text(0.5, 0, 'Percentage of Races'),
       Text(0, 0.5, 'States'),
       Text(0.5, 1.0, "Percentage of State's Population According to Races ")]
```


Percentage of State's Population According to Races
States
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming
Puerto Rico
0
20
40
60
80
100
Percentage of Races
White
Black
Native
Asian
Hispanic

[51]:

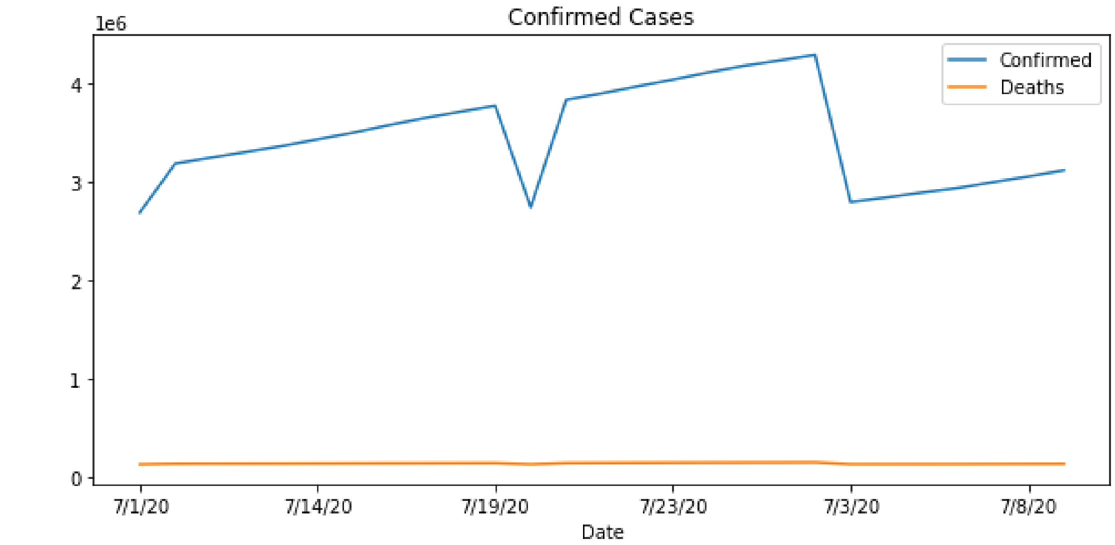
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
import time
from datetime import datetime
from scipy import integrate, optimize
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn import linear_model
from sklearn.metrics import mean_squared_error


plt.rcParams["figure.figsize"] = (10,5)
pd.options.display.max_columns = 1000
```

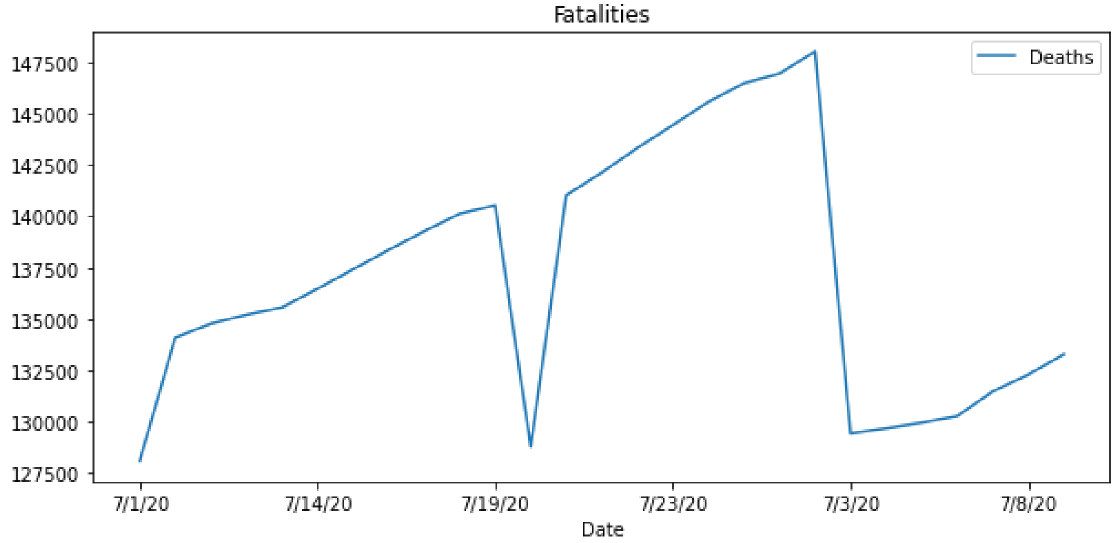
[52]:

```
fig, axes = plt.subplots(2, 1, figsize = (10, 10))
uscountyDF.groupby("Date").agg({"Confirmed": "sum","Deaths": "sum"}).plot(ax =␣
 ↪fig.axes[0])
uscountyDF.groupby("Date").agg({"Deaths": "sum"}).plot(ax = fig.axes[1])
fig.axes[0].set_title("Confirmed Cases")
fig.axes[1].set_title("Fatalities")
```

[52]:

```
Text(0.5, 1.0, 'Fatalities')
```


Confirmed Cases
1e6
4
3
2
1
0
Confirmed
Deaths
7/1/20
7/14/20
7/19/20
7/23/20
7/3/20
7/8/20
Date


Fatalities
147500
145000
142500
140000
137500
135000
132500
130000
127500
Deaths
7/1/20
7/14/20
7/19/20
7/23/20
7/3/20
7/8/20
Date