Tester

July 5, 2023

[1]:
```
import pandas as pd
```

[2]:
```
df = pd.read_csv("/Users/samantarana/Downloads/Datasets/CovidDeaths.csv")
df.head(2)
```

[2]:

	iso_code	continent	location	date	population	total_cases	\
0	AFG	Asia	Afghanistan	2020-01-03	41128772	NaN	
1	AFG	Asia	Afghanistan	2020-01-04	41128772	NaN	

	total_deaths	total_tests	new_cases	new_cases_per_million	…	\
0	NaN	NaN	0.0	0.0	…	
1	NaN	NaN	0.0	0.0	…	

	female_smokers	male_smokers	handwashing_facilities	\
0	NaN	NaN	37.746	
1	NaN	NaN	37.746	

	hospital_beds_per_thousand	life_expectancy	human_development_index	\
0	0.5	64.83	0.511	
1	0.5	64.83	0.511	

	excess_mortality_cumulative_absolute	excess_mortality_cumulative	\
0	NaN	NaN	
1	NaN	NaN	

	excess_mortality	excess_mortality_cumulative_per_million
0	NaN	NaN
1	NaN	NaN

[2 rows x 51 columns]

[3]:
```
X = df[["location", "date", "total_cases", "new_cases", "total_deaths",␣
 ↪"population"]]
X.head(2)
```

[3]:

	location	date	total_cases	new_cases	total_deaths	population
0	Afghanistan	2020-01-03	NaN	0.0	NaN	41128772
1	Afghanistan	2020-01-04	NaN	0.0	NaN	41128772

0.1 Analyzing Total cases vs Total deaths

[4]:
```
Y = df[["location", "date", "total_cases", "total_deaths"]].copy()
Y.head(2)
```

[4]:

	location	date	total_cases	total_deaths
0	Afghanistan	2020-01-03	NaN	NaN
1	Afghanistan	2020-01-04	NaN	NaN

[5]:
```
Y["Death_Percentage"] = (Y["total_deaths"]/Y["total_cases"])*100
Y.head(2)
```

[5]:

	location	date	total_cases	total_deaths	Death_Percentage
0	Afghanistan	2020-01-03	NaN	NaN	NaN
1	Afghanistan	2020-01-04	NaN	NaN	NaN

[6]:
```
##To get rid of warnings

#import warnings

## Filter and ignore all warnings
#warnings.filterwarnings("ignore")

## Reset the warning filters (optional)
#warnings.resetwarnings()
```

[7]:
```
#Choosing a random entry to ensure the calculations are applied

data = Y.iloc[82]
data
```

[7]:
```
location            Afghanistan
date                 2020-03-25
total_cases                42.0
total_deaths                1.0
Death_Percentage       2.380952
Name: 82, dtype: object
```

[8]:
```
#sorting Y according to location and date

sorted_Y = Y.sort_values(by=['location', 'date'], ascending=False)
```

[9]:
```
Y.head()
```

[9]:

	location	date	total_cases	total_deaths	Death_Percentage
0	Afghanistan	2020-01-03	NaN	NaN	NaN
1	Afghanistan	2020-01-04	NaN	NaN	NaN
2	Afghanistan	2020-01-05	NaN	NaN	NaN

```
3  Afghanistan  2020-01-06          NaN           NaN               NaN
4  Afghanistan  2020-01-07          NaN           NaN               NaN
```

[10]:
```
data = Y.iloc[81]
data
```

[10]:
```
location            Afghanistan
date                 2020-03-24
total_cases                40.0
total_deaths                1.0
Death_Percentage            2.5
Name: 81, dtype: object
```

The percentage in the above dataframe, Y , shows the chances of you dying if you contract Covid in you country

0.2 Analyzing Total Cases vs the Population

[11]:
```
Z = df[["location", "date", "population", "total_cases"]].copy()
Z.head(3)
```

[11]:
```
       location        date  population  total_cases
0  Afghanistan  2020-01-03    41128772          NaN
1  Afghanistan  2020-01-04    41128772          NaN
2  Afghanistan  2020-01-05    41128772          NaN
```

[12]:
```
Z["Pop_Death_Rate"] = (Z["total_cases"]/Z["population"])*100
Z.head(2)
```

[12]:
```
       location        date  population  total_cases  Pop_Death_Rate
0  Afghanistan  2020-01-03    41128772          NaN             NaN
1  Afghanistan  2020-01-04    41128772          NaN             NaN
```

[13]:
```
#printing few random rows to analyze the validity of the syntax above

Z.loc[51:58,]
```

[13]:
```
        location        date  population  total_cases  Pop_Death_Rate
51  Afghanistan  2020-02-23    41128772          NaN             NaN
52  Afghanistan  2020-02-24    41128772          NaN             NaN
53  Afghanistan  2020-02-25    41128772          NaN             NaN
54  Afghanistan  2020-02-26    41128772          1.0        0.000002
55  Afghanistan  2020-02-27    41128772          1.0        0.000002
56  Afghanistan  2020-02-28    41128772          1.0        0.000002
57  Afghanistan  2020-02-29    41128772          1.0        0.000002
58  Afghanistan  2020-03-01    41128772          1.0        0.000002
```

0.3 Analyzing Countries with highest infection rates w.r.t population

[14]:
```
# Taking the maximum value out of all the total_cases as Highest Infected Count

df["HighestInfectionCount"] = df["total_cases"].max()
df.head()
```

[14]:

	iso_code	continent	location	date	population	total_cases	\
0	AFG	Asia	Afghanistan	2020-01-03	41128772	NaN	
1	AFG	Asia	Afghanistan	2020-01-04	41128772	NaN	
2	AFG	Asia	Afghanistan	2020-01-05	41128772	NaN	
3	AFG	Asia	Afghanistan	2020-01-06	41128772	NaN	
4	AFG	Asia	Afghanistan	2020-01-07	41128772	NaN	

	total_deaths	total_tests	new_cases	new_cases_per_million	…	\
0	NaN	NaN	0.0	0.0	…	
1	NaN	NaN	0.0	0.0	…	
2	NaN	NaN	0.0	0.0	…	
3	NaN	NaN	0.0	0.0	…	
4	NaN	NaN	0.0	0.0	…	

	male_smokers	handwashing_facilities	hospital_beds_per_thousand	\
0	NaN	37.746	0.5	
1	NaN	37.746	0.5	
2	NaN	37.746	0.5	
3	NaN	37.746	0.5	
4	NaN	37.746	0.5	

	life_expectancy	human_development_index	\
0	64.83	0.511	
1	64.83	0.511	
2	64.83	0.511	
3	64.83	0.511	
4	64.83	0.511	

	excess_mortality_cumulative_absolute	excess_mortality_cumulative	\
0	NaN	NaN	
1	NaN	NaN	
2	NaN	NaN	
3	NaN	NaN	
4	NaN	NaN	

	excess_mortality	excess_mortality_cumulative_per_million	\
0	NaN	NaN	
1	NaN	NaN	
2	NaN	NaN	
3	NaN	NaN	

```
4               NaN                                        NaN

   HighestInfectionCount
0           766894311.0
1           766894311.0
2           766894311.0
3           766894311.0
4           766894311.0

[5 rows x 52 columns]
```

[15]:
```
#Creating a subset of the dataframe df

W1 = df[["location","date", "population", "HighestInfectionCount"]].copy()
W = W1[df['continent'].notnull()].copy()
```

[16]:
```
#Calculating the percentage of the population who got infected country wise

W["PercentPopulationInfected"] = round((df["total_cases"].max()/
 ↪df["population"])*100,2)
```

[17]:
```
W.head(3)
```

[17]:
```
      location        date  population  HighestInfectionCount  \
0  Afghanistan  2020-01-03    41128772            766894311.0
1  Afghanistan  2020-01-04    41128772            766894311.0
2  Afghanistan  2020-01-05    41128772            766894311.0

   PercentPopulationInfected
0                    1864.62
1                    1864.62
2                    1864.62
```

[18]:
```
import numpy as np
```

[19]:
```
# Grouping data so to analyze it better

grouped_data = W.groupby(['location', 'population', 'HighestInfectionCount'])

# the groupby object is not directly printed, to access the grouped data␣
 ↪various ,methods and functions can be used,
#one of which is mean.

mean_values =np.log(grouped_data['PercentPopulationInfected'].mean())

mean_values_sorted = round(mean_values.sort_values(ascending=False),2)
```

```
print(mean_values_sorted)
```

```
location          population  HighestInfectionCount
Pitcairn          47          766894311.0              21.21
Vatican           808         766894311.0              18.37
Tokelau           1893        766894311.0              17.52
Niue              1952        766894311.0              17.49
Falkland Islands  3801        766894311.0              16.82
                                                       …
Pakistan          235824864   766894311.0               5.78
Indonesia         275501344   766894311.0               5.63
United States     338289856   766894311.0               5.42
India             1417173120  766894311.0               3.99
China             1425887360  766894311.0               3.98
Name: PercentPopulationInfected, Length: 243, dtype: float64
```

0.4 Observing countries with highest number of death counts per population

[20]:
```
V1 = df[["location", "total_deaths"]].copy()
V = V1[df['continent'].notnull()].copy()
```

[21]:
```
V["HighestDeathCount"] = V["total_deaths"].max()

# Grouping data so to analyze it better

grouped_data = V.groupby(['location'])

# the groupby object is not directly printed, to access the grouped data␣
 ↪various ,methods and functions can be used,
#one of which is mean.

max_values = round(grouped_data['total_deaths'].max(),2)

max_values_sorted = max_values.sort_values(ascending=False)


V = pd.DataFrame(max_values_sorted)

max_values_sorted = ['TotalDeathCount']

V.columns = [max_values_sorted]

V.head()
```

[21]:
```
               TotalDeathCount
location
United States        1127152.0
```

```
Brazil              702421.0
India               531843.0
Russia              398919.0
Mexico              334079.0
```

0.5 Breaking the above analyzation by continent

[22]:
```
U = df[["location", "total_deaths", "continent"]].copy()
U = U[df['continent'].notnull()]
```

[23]:
```
# Grouping data so to analyze it better

grouped_data = U.groupby(['continent'])

# the groupby object is not directly printed, to access the grouped data␣
 ↪various ,methods and functions can be used,
#one of which is mean.

max_values = round(grouped_data['total_deaths'].max(),2)

max_values_sorted = max_values.sort_values(ascending=False)


U = pd.DataFrame(max_values_sorted)

max_values_sorted = ['TotalDeathCountbyContinent']

U.columns = [max_values_sorted]

U.head()
```

[23]:

continent	TotalDeathCountbyContinent
North America	1127152.0
South America	702421.0
Asia	531843.0
Europe	398919.0
Africa	102595.0

######Here if we observe North America and United States have the same death counts which means that Canada's death count is not included in this - trying to work to be more inclusive with the numbers

0.6 Possible Answer for now :-

[24]:
```
T1 = df[["location", "total_deaths"]].copy()
T = T1[df['continent'].isnull()].copy()

T = T[~T['location'].str.contains('High income')]
T = T[~T['location'].str.contains('Upper middle income')]

T["HighestDeathCount"] = T["total_deaths"].max()

# Grouping data so to analyze it better

grouped_data = T.groupby(['location'])

# the groupby object is not directly printed, to access the grouped data␣
 ↪various ,methods and functions can be used,
#one of which is mean.

max_values = round(grouped_data['total_deaths'].max(),2)

max_values_sorted = max_values.sort_values(ascending=False)


T = pd.DataFrame(max_values_sorted)

max_values_sorted = ['TotalDeathCount']

T.columns = [max_values_sorted]

T.head()
```

[24]:

location	TotalDeathCount
World	6935876.0
Europe	2063499.0
Asia	1632439.0
North America	1601478.0
South America	1352565.0

0.7 GLOBAL NUMBERS

[25]:
```
R = df[["location", "date", "new_cases", "total_deaths"]].copy()

R["total_cases"] = R["new_cases"].sum()

R["total_deaths"] = R["total_deaths"].sum()

R = R[df['continent'].notnull()].copy()
```

```
R["Death_Percentage"] = (R["total_deaths"]/R["total_cases"])*100

# Grouping data so to analyze it better

grouped_data = R.groupby(['date'])

# the groupby object is not directly printed, to access the grouped data␣
 ↪various ,methods and functions can be used,
#one of which is mean.

sum_values = round(grouped_data['new_cases'].sum(),2)

#Aggregading per 100000of the population

sum_values = round(sum_values/100000,2)

sum_values_sorted = sum_values.sort_values(ascending=False)

R = pd.DataFrame(sum_values_sorted)

sum_values_sorted = ['Death_Percentage']

R.columns = [sum_values_sorted]

R.head()
```

[25]:

date	Death_Percentage
2022-12-22	74.61
2022-12-23	73.09
2022-12-21	69.57
2022-12-20	63.30
2022-12-24	61.35

0.7.1 Dealing with question - 1 : Can the current vaccination rates and success rates be used to predict future trends in COVID-19 deaths globally? What are the projected impacts of successful vaccination campaigns on reducing mortality rates in the coming months

[26]:
```
df=df.fillna(value=0, axis=1)
df.head()
```

[26]:

	iso_code	continent	location	date	population	total_cases	\
0	AFG	Asia	Afghanistan	2020-01-03	41128772	0	
1	AFG	Asia	Afghanistan	2020-01-04	41128772	0	
2	AFG	Asia	Afghanistan	2020-01-05	41128772	0	

```
3      AFG      Asia  Afghanistan  2020-01-06   41128772           0
4      AFG      Asia  Afghanistan  2020-01-07   41128772           0

  total_deaths total_tests new_cases new_cases_per_million  … male_smokers  \
0            0           0       0.0                   0.0  …            0
1            0           0       0.0                   0.0  …            0
2            0           0       0.0                   0.0  …            0
3            0           0       0.0                   0.0  …            0
4            0           0       0.0                   0.0  …            0

  handwashing_facilities hospital_beds_per_thousand life_expectancy  \
0                 37.746                        0.5           64.83
1                 37.746                        0.5           64.83
2                 37.746                        0.5           64.83
3                 37.746                        0.5           64.83
4                 37.746                        0.5           64.83

  human_development_index excess_mortality_cumulative_absolute  \
0                   0.511                                     0
1                   0.511                                     0
2                   0.511                                     0
3                   0.511                                     0
4                   0.511                                     0

  excess_mortality_cumulative excess_mortality  \
0                           0                0
1                           0                0
2                           0                0
3                           0                0
4                           0                0

  excess_mortality_cumulative_per_million HighestInfectionCount
0                                       0          766894311.0
1                                       0          766894311.0
2                                       0          766894311.0
3                                       0          766894311.0
4                                       0          766894311.0

[5 rows x 52 columns]
```

[27]:
```
from datetime import datetime
```

[28]:
```
df['date'].max()
```

[28]:
```
'2023-05-29'
```

[29]:
```
df['date'].min()
```

[29]:
```
'2020-01-01'
```

[30]:
```
df['date']=pd.to_datetime(df['date'], format='%Y-%m-%d')
```

[31]:
```
df['date_m']=df['date'].dt.strftime('%b-%Y')
df['date_m'].head()
```

[31]:
```
0    Jan-2020
1    Jan-2020
2    Jan-2020
3    Jan-2020
4    Jan-2020
Name: date_m, dtype: object
```

[40]:
```
import plotly.express as px
import matplotlib.pyplot as plt
```

[33]:
```
import seaborn as sns
```

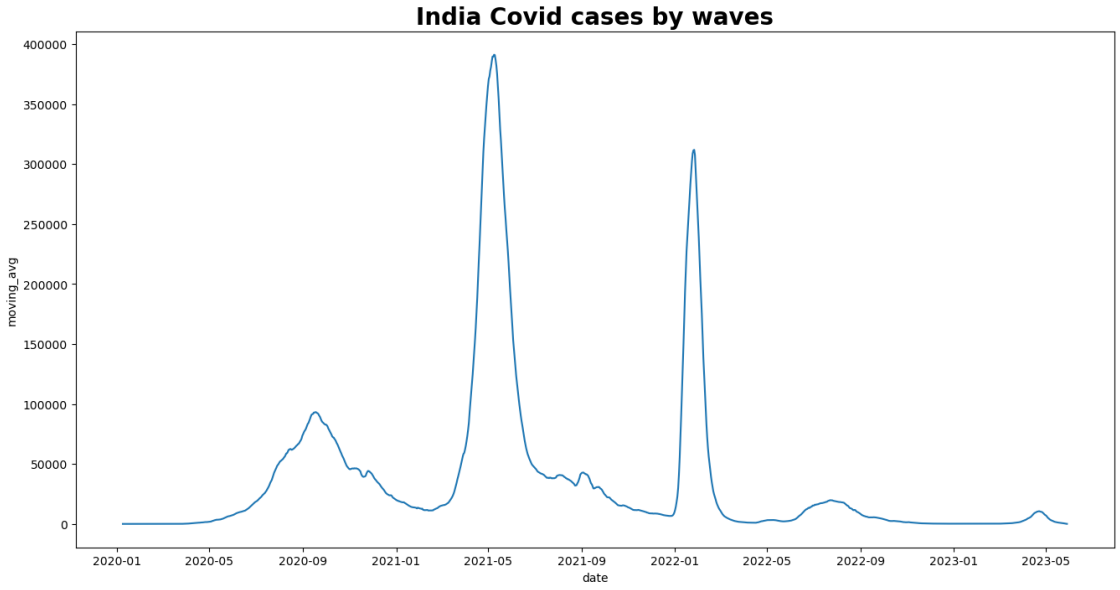
[36]:
```
df['moving_avg']=df.groupby(by=['location']).rolling(7).mean()['new_cases'].
  ↪reset_index(drop=True)
```

```
/var/folders/ss/9hl3rq7s0s31mmyq3h60qwyw0000gn/T/ipykernel_16162/764933178.py:1:
FutureWarning: Dropping of nuisance columns in rolling operations is deprecated;
in a future version this will raise TypeError. Select only valid columns before
calling the operation. Dropped columns were Index(['continent', 'date',
'date_m', 'iso_code', 'tests_units'], dtype='object')
  df['moving_avg']=df.groupby(by=['location']).rolling(7).mean()['new_cases'].re
set_index(drop=True)
```

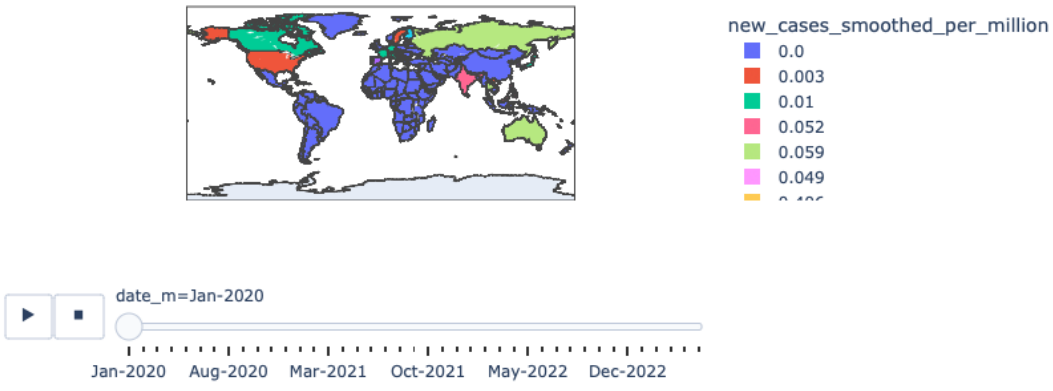
[37]:
```
# Selecting only the required columns before performing the rolling operation
columns_to_keep = ['location', 'new_cases']
df_subset = df[columns_to_keep].copy()

# Calculating the 7-day moving average of new_cases grouped by location
df_subset['moving_avg'] = df_subset.groupby('location')['new_cases'].rolling(7).
  ↪mean().reset_index(drop=True)
```

[41]:
```
fg = plt.figure(figsize=(16,8))
plt.title('India Covid cases by waves', fontsize=20, weight='bold')
ax=sns.lineplot(data=df[df['location']=='India'], y='moving_avg',x='date')
```



[43]:
```
px.choropleth(data_frame=df, locations='location',locationmode='country␣
↪names',animation_frame=df['date_m'],color =␣
↪'new_cases_smoothed_per_million',range_color=(2,df['new_cases_smoothed_per_million'].
↪max()))
```



[ ]: