Import Libraries

In [ ]:
```
# pip install folium
```

In [ ]:
```
# pip install --upgrade matplotlib
```

In [ ]:
```
# storing and anaysis
import numpy as np
import pandas as pd

# visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import folium
```

Import Dataset

In [ ]:
```
# data from Kaggle:
# https://www.kaggle.com/datasets/cptspark/novel-coronavirus-cdr-202011feb?resou
```

In [ ]:
```
# importing datasets
conf_df = pd.read_csv('time_series_2019-ncov-Confirmed.csv')
deaths_df = pd.read_csv('time_series_2019-ncov-Deaths.csv')
recv_df = pd.read_csv('time_series_2019-ncov-Recovered.csv')
```

In [ ]:
```
conf_df.head()
# deaths_df.head()
# recv_df.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/21/2020 22:00	1/22/2020 12:00	1/23/202 12:(
0	Anhui	Mainland China	31.82571	117.2264	NaN	1.0	9
1	Beijing	Mainland China	40.18238	116.4142	10.0	14.0	22
2	Chongqing	Mainland China	30.05718	107.8740	5.0	6.0	9
3	Fujian	Mainland China	26.07783	117.9895	NaN	1.0	5
4	Gansu	Mainland China	36.06110	103.8343	NaN	NaN	2

5 rows × 43 columns

In [ ]:
```
conf_df.columns
# deaths_df.columns
# recv_df.columns
```

Out[ ]:
```
Index(['Province/State', 'Country/Region', 'Lat', 'Long', '1/21/2020 22:00',
       '1/22/2020 12:00', '1/23/2020 12:00', '1/24/2020 0:00',
       '1/24/2020 12:00', '1/25/2020 0:00', '1/25/2020 12:00',
       '1/25/2020 22:00', '1/26/2020 11:00', '1/26/2020 23:00',
       '1/27/2020 9:00', '1/27/2020 19:00', '1/27/2020 20:30',
       '1/28/2020 13:00', '1/28/2020 18:00', '1/28/2020 23:00',
       '1/29/2020 13:30', '1/29/2020 14:30', '1/29/2020 21:00',
       '1/30/2020 11:00', '1/31/2020 14:00', '2/1/2020 10:00',
       '2/2/2020 21:00', '2/3/2020 21:00', '2/4/2020 9:40', '2/4/2020 22:00',
       '2/5/2020 9:00', '2/5/2020 23:00', '2/6/2020 9:00', '2/6/2020 14:20',
       '2/7/2020 20:13', '2/7/2020 22:50', '2/8/2020 22:04', '2/8/2020 23:04',
       '2/9/2020 10:30', '2/9/2020 23:20', '2/10/2020 10:30',
       '2/10/2020 19:30', '2/11/2020 10:50'],
      dtype='object')
```

In [ ]:
```
conf_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73 entries, 0 to 72
Data columns (total 43 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Province/State   52 non-null     object 
 1   Country/Region   73 non-null     object 
 2   Lat              73 non-null     float64
 3   Long             73 non-null     float64
 4   1/21/2020 22:00  16 non-null     float64
 5   1/22/2020 12:00  29 non-null     float64
 6   1/23/2020 12:00  37 non-null     float64
 7   1/24/2020 0:00   38 non-null     float64
 8   1/24/2020 12:00  40 non-null     float64
 9   1/25/2020 0:00   42 non-null     float64
 10  1/25/2020 12:00  43 non-null     float64
 11  1/25/2020 22:00  43 non-null     float64
 12  1/26/2020 11:00  49 non-null     float64
 13  1/26/2020 23:00  49 non-null     float64
 14  1/27/2020 9:00   50 non-null     float64
 15  1/27/2020 19:00  51 non-null     float64
 16  1/27/2020 20:30  52 non-null     float64
 17  1/28/2020 13:00  53 non-null     float64
 18  1/28/2020 18:00  53 non-null     float64
 19  1/28/2020 23:00  53 non-null     float64
 20  1/29/2020 13:30  56 non-null     float64
 21  1/29/2020 14:30  55 non-null     float64
 22  1/29/2020 21:00  57 non-null     float64
 23  1/30/2020 11:00  59 non-null     float64
 24  1/31/2020 14:00  65 non-null     float64
 25  2/1/2020 10:00   67 non-null     float64
 26  2/2/2020 21:00   68 non-null     float64
 27  2/3/2020 21:00   69 non-null     float64
 28  2/4/2020 9:40    70 non-null     float64
 29  2/4/2020 22:00   70 non-null     float64
 30  2/5/2020 9:00    70 non-null     float64
 31  2/5/2020 23:00   71 non-null     float64
 32  2/6/2020 9:00    71 non-null     float64
 33  2/6/2020 14:20   71 non-null     float64
 34  2/7/2020 20:13   72 non-null     float64
 35  2/7/2020 22:50   72 non-null     float64
 36  2/8/2020 22:04   72 non-null     float64
 37  2/8/2020 23:04   72 non-null     float64
 38  2/9/2020 10:30   72 non-null     float64
 39  2/9/2020 23:20   72 non-null     float64
 40  2/10/2020 10:30  72 non-null     float64
 41  2/10/2020 19:30  72 non-null     float64
 42  2/11/2020 10:50  73 non-null     int64  
dtypes: float64(40), int64(1), object(2)
memory usage: 24.6+ KB
```

In [ ]:
```
# Filter the DataFrame to select rows where the 'Province/State' column is 'Diam
ship_Confirmed = conf_df[conf_df['Province/State'] == 'Diamond Princess cruise s

# Display the resulting DataFrame containing confirmed cases on the Diamond Prin
ship_Confirmed
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/21/2020 22:00	1/22/2020 12:00	1/23/202(12:0(
71	Diamond Princess cruise ship	Others	35.4437	129.638	NaN	NaN	NaN

1 rows × 43 columns

In [ ]:
```
deaths_df.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/21/20 22:00	1/22/20 12:00	1/23/20 12:00	1/2
0	Anhui	Mainland China	31.82571	117.2264	NaN	NaN	NaN	
1	Beijing	Mainland China	40.18238	116.4142	NaN	NaN	NaN	
2	Chongqing	Mainland China	30.05718	107.8740	NaN	NaN	NaN	
3	Fujian	Mainland China	26.07783	117.9895	NaN	NaN	NaN	
4	Gansu	Mainland China	36.06110	103.8343	NaN	NaN	NaN	

5 rows × 43 columns

In [ ]:
```
recv_df.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/21/20 22:00	1/22/20 12:00	1/23/20 12:00	1/2
0	Anhui	Mainland China	31.82571	117.2264	NaN	NaN	NaN	
1	Beijing	Mainland China	40.18238	116.4142	NaN	NaN	NaN	
2	Chongqing	Mainland China	30.05718	107.8740	NaN	NaN	NaN	
3	Fujian	Mainland China	26.07783	117.9895	NaN	NaN	NaN	
4	Gansu	Mainland China	36.06110	103.8343	NaN	NaN	NaN	

5 rows × 43 columns

In [ ]:
```
```

Data Wrangling

In [ ]:
```
dates = ['1/22/20', '1/23/20', '1/24/20', '1/25/20', '1/26/20', '1/27/20', '1/28
         '1/29/20', '1/30/20', '1/31/20', '2/1/20', '2/2/20', '2/3/20', '2/4/20'
         '2/5/20', '2/6/20', '2/7/20', '2/8/20', '2/9/20', '2/10/20', '2/11/20',
         '2/13/20', '2/14/20', '2/15/20', '2/16/20', '2/17/20', '2/18/20', '2/19
dates
```

Out[ ]:
```
['1/22/20',
 '1/23/20',
 '1/24/20',
 '1/25/20',
 '1/26/20',
 '1/27/20',
 '1/28/20',
 '1/29/20',
 '1/30/20',
 '1/31/20',
 '2/1/20',
 '2/2/20',
 '2/3/20',
 '2/4/20',
 '2/5/20',
 '2/6/20',
 '2/7/20',
 '2/8/20',
 '2/9/20',
 '2/10/20',
 '2/11/20',
 '2/12/20',
 '2/13/20',
 '2/14/20',
 '2/15/20',
 '2/16/20',
 '2/17/20',
 '2/18/20',
 '2/19/20']
```

In [ ]:
```
conf_df_long = conf_df.melt(id_vars=['Province/State', 'Country/Region', 'Lat',
                            value_vars=conf_df.columns[4:], var_name='Date', val

deaths_df_long = deaths_df.melt(id_vars=['Province/State', 'Country/Region', 'La
                            value_vars=deaths_df.columns[4:], var_name='Date', v

recv_df_long = recv_df.melt(id_vars=['Province/State', 'Country/Region', 'Lat',
                            value_vars=recv_df.columns[4:], var_name='Date', val

full_table = pd.concat([conf_df_long, deaths_df_long['Deaths'], recv_df_long['Re
                        axis=1, sort=False)
full_table.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths
0	Anhui	Mainland China	31.82571	117.2264	1/21/2020 22:00	NaN	NaN
1	Beijing	Mainland China	40.18238	116.4142	1/21/2020 22:00	10.0	NaN
2	Chongqing	Mainland China	30.05718	107.8740	1/21/2020 22:00	5.0	NaN
3	Fujian	Mainland China	26.07783	117.9895	1/21/2020 22:00	NaN	NaN
4	Gansu	Mainland China	36.06110	103.8343	1/21/2020 22:00	NaN	NaN

In [ ]:

```
# cases in the Diamond Princess cruise ship
ship_1 = full_table[full_table['Province/State']=='Diamond Princess cruise ship'

ship_1.head(10)
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths
71	Diamond Princess cruise ship	Others	35.4437	129.638	1/21/2020 22:00	NaN	NaN
144	Diamond Princess cruise ship	Others	35.4437	129.638	1/22/2020 12:00	NaN	NaN
217	Diamond Princess cruise ship	Others	35.4437	129.638	1/23/2020 12:00	NaN	NaN
290	Diamond Princess cruise ship	Others	35.4437	129.638	1/24/2020 0:00	NaN	NaN
363	Diamond Princess cruise ship	Others	35.4437	129.638	1/24/2020 12:00	NaN	NaN
436	Diamond Princess cruise ship	Others	35.4437	129.638	1/25/2020 0:00	NaN	NaN
509	Diamond Princess cruise ship	Others	35.4437	129.638	1/25/2020 12:00	NaN	NaN
582	Diamond Princess cruise ship	Others	35.4437	129.638	1/25/2020 22:00	NaN	NaN
655	Diamond Princess cruise ship	Others	35.4437	129.638	1/26/2020 11:00	NaN	NaN
728	Diamond Princess cruise ship	Others	35.4437	129.638	1/26/2020 23:00	NaN	NaN

In [ ]:
```
print(ship_1)
```

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	Recovered
71	Diamond Princess cruise ship	Others	35.4437	129.638	1/21/2020 22:00	NaN	NaN	NaN
144	Diamond Princess cruise ship	Others	35.4437	129.638	1/22/2020 12:00	NaN	NaN	NaN
217	Diamond Princess cruise ship	Others	35.4437	129.638	1/23/2020 12:00	NaN	NaN	NaN
290	Diamond Princess cruise ship	Others	35.4437	129.638	1/24/2020 0:00	NaN	NaN	NaN
363	Diamond Princess cruise ship	Others	35.4437	129.638	1/24/2020 12:00	NaN	NaN	NaN
436	Diamond Princess cruise ship	Others	35.4437	129.638	1/25/2020 0:00	NaN	NaN	NaN
509	Diamond Princess cruise ship	Others	35.4437	129.638	1/25/2020 12:00	NaN	NaN	NaN
582	Diamond Princess cruise ship	Others	35.4437	129.638	1/25/2020 22:00	NaN	NaN	NaN
655	Diamond Princess cruise ship	Others	35.4437	129.638	1/26/2020 11:00	NaN	NaN	NaN
728	Diamond Princess cruise ship	Others	35.4437	129.638	1/26/2020 23:00	NaN	NaN	NaN
801	Diamond Princess cruise ship	Others	35.4437	129.638	1/27/2020 9:00	NaN	NaN	NaN
874	Diamond Princess cruise ship	Others	35.4437	129.638	1/27/2020 19:00	NaN	NaN	NaN
947	Diamond Princess cruise ship	Others	35.4437	129.638	1/27/2020 20:30	NaN	NaN	NaN
1020	Diamond Princess cruise ship	Others	35.4437	129.638	1/28/2020 13:00	NaN	NaN	NaN
1093	Diamond Princess cruise ship	Others	35.4437	129.638	1/28/2020 18:00	NaN	NaN	NaN
1166	Diamond Princess cruise ship	Others	35.4437	129.638	1/28/2020 23:00	NaN	NaN	NaN
1239	Diamond Princess cruise ship	Others	35.4437	129.638	1/29/2020 13:30	NaN	NaN	NaN
1312	Diamond Princess cruise ship	Others	35.4437	129.638	1/29/2020 14:30	NaN	NaN	NaN
1385	Diamond Princess cruise ship	Others	35.4437	129.638	1/29/2020 19:30	NaN	NaN	NaN
1458	Diamond Princess cruise ship	Others	35.4437	129.638	1/30/2020 0:00	NaN	NaN	NaN
1531	Diamond Princess cruise ship	Others	35.4437	129.638	1/30/2020 12:00	NaN	NaN	NaN
1604	Diamond Princess cruise ship	Others	35.4437	129.638	1/30/2020 23:00	NaN	NaN	NaN
1677	Diamond Princess cruise ship	Others	35.4437	129.638	1/31/2020 12:00	NaN	NaN	NaN
1750	Diamond Princess cruise ship	Others	35.4437	129.638	2/1/2020 0:00	NaN	NaN	NaN
1823	Diamond Princess cruise ship	Others	35.4437	129.638	2/1/2020 12:00	NaN	NaN	NaN
1896	Diamond Princess cruise ship	Others	35.4437	129.638	2/2/2020 0:00	NaN	NaN	NaN
1969	Diamond Princess cruise ship	Others	35.4437	129.638	2/2/2020 12:00	NaN	NaN	NaN
2042	Diamond Princess cruise ship	Others	35.4437	129.638	2/3/2020 0:00	NaN	NaN	NaN
2115	Diamond Princess cruise ship	Others	35.4437	129.638	2/3/2020 12:00	NaN	NaN	NaN
2188	Diamond Princess cruise ship	Others	35.4437	129.638	2/4/2020 0:00	NaN	NaN	NaN
2261	Diamond Princess cruise ship	Others	35.4437	129.638	2/4/2020 12:00	NaN	NaN	NaN
2334	Diamond Princess cruise ship	Others	35.4437	129.638	2/5/2020 0:00	NaN	NaN	NaN
2407	Diamond Princess cruise ship	Others	35.4437	129.638	2/5/2020 12:00	NaN	NaN	NaN
2480	Diamond Princess cruise ship	Others	35.4437	129.638	2/6/2020 0:00	NaN	NaN	NaN
2553	Diamond Princess cruise ship	Others	35.4437	129.638	2/6/2020 12:00	NaN	NaN	NaN
2626	Diamond Princess cruise ship	Others	35.4437	129.638	2/7/2020 0:00	NaN	NaN	NaN
2699	Diamond Princess cruise ship	Others	35.4437	129.638	2/7/2020 12:00	NaN	NaN	NaN
2772	Diamond Princess cruise ship	Others	35.4437	129.638	2/8/2020 0:00	NaN	NaN	NaN
2845	Diamond Princess cruise ship	Others	35.4437	129.638	2/8/2020 12:00	NaN	NaN	NaN

```
1385  1/29/2020 21:00        NaN     NaN        NaN
1458  1/30/2020 11:00        NaN     NaN        NaN
1531  1/31/2020 14:00        NaN     NaN        NaN
1604   2/1/2020 10:00        NaN     NaN        NaN
1677   2/2/2020 21:00        NaN     NaN        NaN
1750   2/3/2020 21:00        NaN     NaN        NaN
1823    2/4/2020 9:40        NaN     NaN        NaN
1896   2/4/2020 22:00        NaN     NaN        NaN
1969    2/5/2020 9:00        NaN     NaN        NaN
2042   2/5/2020 23:00        NaN     NaN        NaN
2115    2/6/2020 9:00        NaN     NaN        NaN
2188   2/6/2020 14:20        NaN     NaN        NaN
2261   2/7/2020 20:13       61.0     NaN        NaN
2334   2/7/2020 22:50       61.0     NaN        NaN
2407   2/8/2020 22:04       61.0     0.0        0.0
2480   2/8/2020 23:04       61.0     0.0        0.0
2553   2/9/2020 10:30       64.0     0.0        0.0
2626   2/9/2020 23:20       64.0     0.0        0.0
2699  2/10/2020 10:30       64.0     0.0        0.0
2772  2/10/2020 19:30      135.0     0.0        0.0
2845  2/11/2020 10:50      135.0     0.0        0.0
```

In [ ]:
```
ship_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39 entries, 71 to 2845
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Province/State  39 non-null     object 
 1   Country/Region  39 non-null     object 
 2   Lat             39 non-null     float64
 3   Long            39 non-null     float64
 4   Date            39 non-null     object 
 5   Confirmed       9 non-null      float64
 6   Deaths          7 non-null      float64
 7   Recovered       7 non-null      float64
dtypes: float64(5), object(3)
memory usage: 2.7+ KB
```

In [ ]:
```
ship_1.shape
```

Out[ ]:
```
(39, 8)
```

In [ ]:
```
confirm_var= ship_1.value_counts('Confirmed')
confirm_var
```

Out[ ]:
```
Confirmed
61.0     4
64.0     3
135.0    2
dtype: int64
```

Data Cleaning and Preprocessing

In [ ]:
```
# Step 1: Convert 'Date' Column to DateTime Format
full_table['Date'] = pd.to_datetime(full_table['Date'])
# This line converts the 'Date' column to a proper datetime format. This is impo
```

```
# Step 2: Replace 'Mainland China' with 'China' in 'Country/Region' Column
full_table['Country/Region'] = full_table['Country/Region'].replace('Mainland Ch
# This line replaces occurrences of 'Mainland China' with 'China' in the 'Countr

# Step 3: Fill Missing Values in 'Confirmed', 'Deaths', and 'Recovered' Columns
full_table[['Confirmed', 'Deaths', 'Recovered']] = full_table[['Confirmed', 'Dea
# Missing values in the 'Confirmed', 'Deaths', and 'Recovered' columns are fille

# Step 4: Convert 'Recovered' Column to Integer Data Type
full_table['Recovered'] = full_table['Recovered'].astype('int') #if not, you wil
full_table['Confirmed'] = full_table['Confirmed'].astype('int')
full_table['Deaths'] = full_table['Deaths'].astype('int')
# The 'Recovered' column is converted to integer data type. This ensures that th

# Step 5: Fill Missing Values in 'Province/State' Column
full_table[['Province/State']] = full_table[['Province/State']].fillna('NA')
# Missing values in the 'Province/State' column are filled with 'NA' to indicate
```

In [ ]:

```
# Step 6: Extract Data Related to Diamond Princess Cruise Ship
ship = full_table[full_table['Province/State'] == 'Diamond Princess cruise ship'
ship
# A new DataFrame 'ship' is created, containing data related to the Diamond Prin
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	I
71	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-21 22:00:00	0	0	
144	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-22 12:00:00	0	0	
217	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-23 12:00:00	0	0	
290	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-24 00:00:00	0	0	
363	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-24 12:00:00	0	0	
436	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-25 00:00:00	0	0	
509	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-25 12:00:00	0	0	
582	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-25 22:00:00	0	0	
655	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-26 11:00:00	0	0	
728	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-26 23:00:00	0	0	
801	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-27 09:00:00	0	0	
874	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-27 19:00:00	0	0	
947	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-27 20:30:00	0	0	
1020	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-28 13:00:00	0	0	
1093	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-28 18:00:00	0	0	

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	I
1166	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-28 23:00:00	0	0	
1239	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-29 13:30:00	0	0	
1312	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-29 14:30:00	0	0	
1385	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-29 21:00:00	0	0	
1458	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-30 11:00:00	0	0	
1531	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-31 14:00:00	0	0	
1604	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-01 10:00:00	0	0	
1677	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-02 21:00:00	0	0	
1750	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-03 21:00:00	0	0	
1823	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-04 09:40:00	0	0	
1896	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-04 22:00:00	0	0	
1969	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-05 09:00:00	0	0	
2042	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-05 23:00:00	0	0	
2115	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-06 09:00:00	0	0	
2188	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-06 14:20:00	0	0	

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	I
2261	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-07 20:13:00	61	0	
2334	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-07 22:50:00	61	0	
2407	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-08 22:04:00	61	0	
2480	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-08 23:04:00	61	0	
2553	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-09 10:30:00	64	0	
2626	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-09 23:20:00	64	0	
2699	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-10 10:30:00	64	0	
2772	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-10 19:30:00	135	0	
2845	Diamond Princess cruise ship	Others	35.4437	129.638	2020-02-11 10:50:00	135	0	

In [ ]:
```
ship.shape
```

Out[ ]:
```
(39, 8)
```

In [ ]:
```
ship.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39 entries, 71 to 2845
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype         
---  ------          --------------  -----         
 0   Province/State  39 non-null     object        
 1   Country/Region  39 non-null     object        
 2   Lat             39 non-null     float64       
 3   Long            39 non-null     float64       
 4   Date            39 non-null     datetime64[ns]
 5   Confirmed       39 non-null     int32         
 6   Deaths          39 non-null     int32         
 7   Recovered       39 non-null     int32         
dtypes: datetime64[ns](1), float64(2), int32(3), object(2)
memory usage: 2.3+ KB
```

In [ ]:
```
# Step 7: Remove Diamond Princess Data from 'full_table'
full_table = full_table[full_table['Province/State'] != 'Diamond Princess cruise
# Data related to the Diamond Princess cruise ship is removed from the 'full_tab

# Step 8: Display the First Few Rows of the Cleaned DataFrame
full_table.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	Re
0	Anhui	China	31.82571	117.2264	2020-01-21 22:00:00	0	0	
1	Beijing	China	40.18238	116.4142	2020-01-21 22:00:00	10	0	
2	Chongqing	China	30.05718	107.8740	2020-01-21 22:00:00	5	0	
3	Fujian	China	26.07783	117.9895	2020-01-21 22:00:00	0	0	
4	Gansu	China	36.06110	103.8343	2020-01-21 22:00:00	0	0	

In [ ]:
```
full_table
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths
0	Anhui	China	31.82571	117.2264	2020-01-21 22:00:00	0	0
1	Beijing	China	40.18238	116.4142	2020-01-21 22:00:00	10	0
2	Chongqing	China	30.05718	107.8740	2020-01-21 22:00:00	5	0
3	Fujian	China	26.07783	117.9895	2020-01-21 22:00:00	0	0
4	Gansu	China	36.06110	103.8343	2020-01-21 22:00:00	0	0
...	...	...	...	...	...	...	...
2841	Boston, MA	US	42.36010	-71.0589	2020-02-11 10:50:00	1	0
2842	San Benito, CA	US	36.57610	-120.9876	2020-02-11 10:50:00	2	0
2843	NA	Belgium	50.50390	4.4699	2020-02-11 10:50:00	1	0
2844	Madison, WI	US	43.07310	-89.4012	2020-02-11 10:50:00	1	0
2846	San Diego County, CA	US	32.71570	-117.1611	2020-02-11 10:50:00	1	0

2808 rows × 8 columns

In [ ]:

```
# Create a DataFrame 'china' containing data only for the 'China' country
china = full_table[full_table['Country/Region']=='China']

# Create a DataFrame 'row' containing data for countries/regions other than 'Chi
row = full_table[full_table['Country/Region']!='China']

# Create a DataFrame 'full_latest' with data for the latest date in the dataset
full_latest = full_table[full_table['Date'] == max(full_table['Date'])].reset_ir

# Create a DataFrame 'china_latest' with data for the latest date only for 'Chin
china_latest = full_latest[full_latest['Country/Region']=='China']

# Create a DataFrame 'row_latest' with data for the latest date for countries/re
row_latest = full_latest[full_latest['Country/Region']!='China']
```

```
# Group the 'full_latest' DataFrame by 'Country/Region' and calculate the sum of
full_latest_grouped = full_latest.groupby('Country/Region')['Confirmed', 'Deaths

# Group the 'china_latest' DataFrame by 'Province/State' and calculate the sum o
china_latest_grouped = china_latest.groupby('Province/State')['Confirmed', 'Deat

# Group the 'row_latest' DataFrame by 'Country/Region' and calculate the sum of
row_latest_grouped = row_latest.groupby('Country/Region')['Confirmed', 'Deaths',
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\1272672933.py:17: FutureWarning:

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

C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\1272672933.py:20: FutureWarning:

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

C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\1272672933.py:23: FutureWarning:

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

EDA

Current Situation

In [ ]:

```
# Group the 'full_latest' DataFrame by both 'Country/Region' and 'Province/State
# Calculate the maximum values of 'Confirmed', 'Deaths', and 'Recovered' for eac
temp = full_latest.groupby(['Country/Region', 'Province/State'])['Confirmed', 'D

# Apply a background gradient style to the 'temp' DataFrame
# The 'background_gradient' function applies a color gradient to cells based on
# Here, 'cmap' specifies the color map used for the gradient, 'Pastel1_r' in thi
styled_temp = temp.style.background_gradient(cmap='Pastel1_r')

# The styled DataFrame 'styled_temp' now has the background gradient applied
# It can be displayed to visualize the data with color-coded cells
styled_temp
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\487413045.py:3: FutureWarning:

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

Out[ ]:

Country/Region	Province/State	Confirmed	Deaths	Recovered
Australia	New South Wales	4	0	2
	Queensland	5	0	0
	South Australia	2	0	0
	Victoria	4	0	0
Belgium	NA	1	0	0
Cambodia	NA	1	0	0
Canada	British Columbia	4	0	0
	London, ON	1	0	0
	Toronto, ON	2	0	0
China	Anhui	860	4	105
	Beijing	342	3	48
	Chongqing	489	2	72
	Fujian	267	0	45
	Gansu	86	2	24
	Guangdong	1177	1	212
	Guangxi	215	1	33
	Guizhou	127	1	17
	Hainan	144	3	20
	Hebei	239	2	48
	Heilongjiang	360	8	28
	Henan	1105	7	218
	Hubei	31728	974	2310
	Hunan	912	1	247
	Inner Mongolia	58	0	5
	Jiangsu	515	0	93
	Jiangxi	804	1	128
	Jilin	81	1	18
	Liaoning	111	0	19
	Ningxia	53	0	22
	Qinghai	18	0	5
	Shaanxi	219	0	32
	Shandong	487	1	80

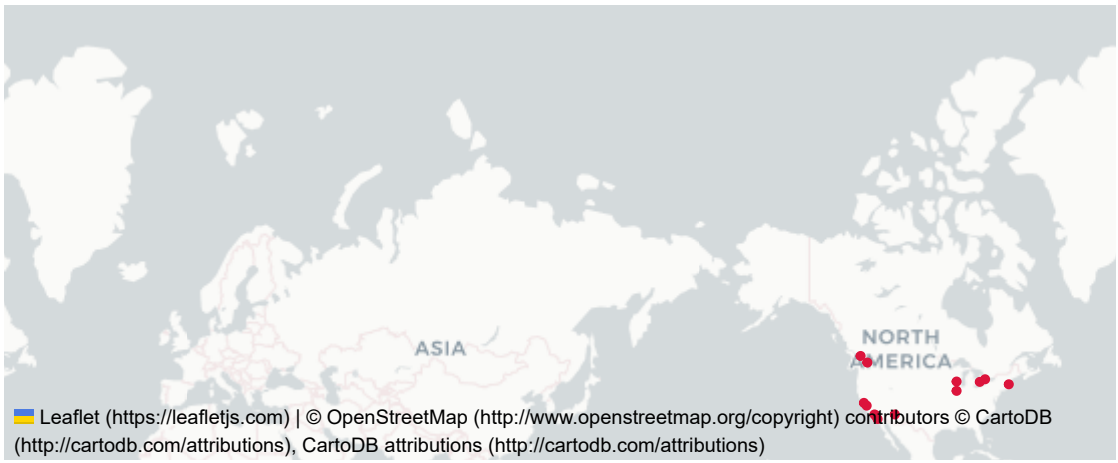
Country/Region	Province/State	Confirmed	Deaths	Recovered
	Shanghai	303	1	52
	Shanxi	122	0	30
	Sichuan	417	1	85
	Tianjin	105	2	10
	Tibet	1	0	0
	Xinjiang	55	0	3
	Yunnan	153	0	20
	Zhejiang	1117	0	270
Finland	NA	1	0	0
France	NA	11	0	0
Germany	NA	14	0	0
Hong Kong	Hong Kong	49	0	3
India	NA	3	0	0
Italy	NA	3	0	0
Japan	NA	26	0	1
Macau	Macau	10	0	10
Malaysia	NA	18	0	3
Nepal	NA	1	0	0
Philippines	NA	3	1	0
Russia	NA	2	0	0
Singapore	NA	45	0	7
South Korea	NA	28	0	1
Spain	NA	2	0	0
Sri Lanka	NA	1	0	1
Sweden	NA	1	0	0
Taiwan	Taiwan	18	0	1
Thailand	NA	32	1	0
UK	NA	8	0	0
US	Boston, MA	1	0	0
	Chicago, IL	2	0	0
	Los Angeles, CA	1	0	0
	Madison, WI	1	0	0

Country/Region	Province/State	Confirmed	Deaths	Recovered
	Orange, CA	1	0	0
	San Benito, CA	2	0	0
	San Diego County, CA	1	0	0
	Santa Clara, CA	2	0	0
	Seattle, WA	1	0	2
	Tempe, AZ	1	0	9
United Arab Emirates	NA	8	0	0
Vietnam	NA	15	0	1

In [ ]:

```
# World wide
#
# Create a base map with specified parameters
m = folium.Map(location=[0, 0], tiles='cartodbpositron',
               min_zoom=1, max_zoom=4, zoom_start=1)

# Iterate through each row in the 'full_latest' DataFrame
for i in range(0, len(full_latest)):
    # Extract the latitude, longitude, and other COVID-19 data for the current r
    lat = full_latest.iloc[i]['Lat']
    long = full_latest.iloc[i]['Long']
    country = full_latest.iloc[i]['Country/Region']
    province = full_latest.iloc[i]['Province/State']
    confirmed = full_latest.iloc[i]['Confirmed']
    deaths = full_latest.iloc[i]['Deaths']
    recovered = full_latest.iloc[i]['Recovered']

    # Create a circle marker for the current country/province
    circle = folium.Circle(
        location=[lat, long],  # Set the circle's location
        color='crimson',       # Set the circle's color
        tooltip=f'''<li><bold>Country: {country}
                    <li><bold>Province: {province}
                    <li><bold>Confirmed: {confirmed}
                    <li><bold>Deaths: {deaths}
                    <li><bold>Recovered: {recovered}''',  # Set the tooltip with
        radius=int(confirmed)  # Set the circle's radius based on the number of
    )

    # Add the circle marker to the map
    circle.add_to(m)

# The map with circle markers representing COVID-19 data has been created
# and each circle has a tooltip with detailed information about the country/prov
m
```

Out[ ]: Make this Notebook Trusted to load map: File -> Trust Notebook



Top 10 Countries with most no. of reported cases

In [ ]:
```
# Extract specific columns ('Country/Region' and 'Confirmed') from the 'full_lat
temp_f = full_latest_grouped[['Country/Region', 'Confirmed']]

# Sort the DataFrame by the 'Confirmed' column in descending order (highest firs
temp_f = temp_f.sort_values(by='Confirmed', ascending=False)

# Reset the index of the sorted DataFrame to start from 0, dropping the old inde
temp_f = temp_f.reset_index(drop=True)

# Display the top 10 rows of the sorted DataFrame with a background gradient usi
temp_f.head(10).style.background_gradient(cmap='Pastel1_r')
```

Out[ ]:

	Country/Region	Confirmed
0	China	42670
1	Hong Kong	49
2	Singapore	45
3	Thailand	32
4	South Korea	28
5	Japan	26
6	Taiwan	18
7	Malaysia	18
8	Australia	15
9	Vietnam	15

- Massive number of cases are reported in Mainland China Compared to reset of the world

- The next few countries are infact are the neighbours of China

In [ ]:
```
# Import the necessary libraries for interactive visualization
#import plotly.express as px

# Create a choropleth map using the 'full_latest_grouped' DataFrame
# The map will represent the number of confirmed cases in each country/region.
fig = px.choropleth(full_latest_grouped,
                    locations="Country/Region",  # Use the 'Country/Region' colu
                    locationmode='country names',  # Specify that the location m
                    color="Confirmed",  # The color scale will be based on the '
                    hover_name="Country/Region",  # Display the country/region n
                    range_color=[1, 50],  # Set the range of colors on the color
                    color_continuous_scale="Sunsetdark",  # Use the 'Sunsetdark'
                    title='Countries with Confirmed Cases'  # Set the title of t
)

# Hide the color scale legend, as the color scale is defined by 'color_continuou
fig.update(layout_coloraxis_showscale=False)
# fig.update(layout_coloraxis_showscale=True)

# Display the choropleth map
fig.show()
```

Top 10 Provinces in China with most no. of reported cases

In [ ]:
```
# Extract specific columns ('Province/State' and 'Confirmed') from the 'china_la
temp_c = china_latest_grouped[['Province/State', 'Confirmed']]

# Sort the DataFrame by the 'Confirmed' column in descending order (highest firs
temp_c = temp_c.sort_values(by='Confirmed', ascending=False)

# Reset the index of the sorted DataFrame to start from 0, dropping the old inde
temp_c = temp_c.reset_index(drop=True)

# Display the top 10 rows of the sorted DataFrame with a styled background gradi
temp_c.head(10).style.background_gradient(cmap='Pastel1_r')
```

Out[ ]:

	Province/State	Confirmed
0	Hubei	31728
1	Guangdong	1177
2	Zhejiang	1117
3	Henan	1105
4	Hunan	912
5	Anhui	860
6	Jiangxi	804
7	Jiangsu	515
8	Chongqing	489
9	Shandong	487

In [ ]:

```
# China
m = folium.Map(location=[30, 116], tiles='cartodbpositron',
               min_zoom=2, max_zoom=5, zoom_start=3)

for i in range(0, len(china_latest)):
    folium.Circle(
        location=[china_latest.iloc[i]['Lat'], china_latest.iloc[i]['Long']],
        color='blue',
        tooltip =   '<li><bold>Country : '+str(china_latest.iloc[i]['Country/Reg
                    '<li><bold>Province : '+str(china_latest.iloc[i]['Province/S
                    '<li><bold>Confirmed : '+str(china_latest.iloc[i]['Confirmed
                    '<li><bold>Deaths : '+str(china_latest.iloc[i]['Deaths'])+
                    '<li><bold>Recovered : '+str(china_latest.iloc[i]['Recovered
        radius=int(china_latest.iloc[i]['Confirmed'])**1).add_to(m)
m
```

Out[ ]:

- Even in China most of the cases reported are from a particular Province Hubei.

- It is no surprise, because Hubei's capital is **Wuhan**, where the the first cases are reported

Countries with deaths reported

In [ ]:
```
# Extract specific columns ('Country/Region' and 'Deaths') from the 'full_latest
temp_flg = full_latest_grouped[['Country/Region', 'Deaths']]

# Sort the DataFrame by the 'Deaths' column in descending order (highest first)
temp_flg = temp_flg.sort_values(by='Deaths', ascending=False)

# Reset the index of the sorted DataFrame to start from 0, dropping the old inde
temp_flg = temp_flg.reset_index(drop=True)

# Filter the DataFrame to include only rows where the number of deaths is greate
temp_flg = temp_flg[temp_flg['Deaths'] > 0]

# Display the filtered and styled DataFrame using a background gradient with the
temp_flg.style.background_gradient(cmap='Pastel1_r')
```

Out[ ]:

	Country/Region	Deaths
0	China	1016
1	Philippines	1
2	Thailand	1

In [ ]:
```
# Import the necessary libraries for interactive visualization
# import plotly.express as px

# Filter the 'full_latest_grouped' DataFrame to include only rows where the numb
filtered_data = full_latest_grouped[full_latest_grouped['Deaths'] > 0]

# Create a choropleth map using the filtered DataFrame
# The map will represent the number of reported deaths in each country/region wi
fig = px.choropleth(filtered_data,
                    locations="Country/Region",  # Use the 'Country/Region' colu
                    locationmode='country names',  # Specify that the location m
                    color="Deaths",  # The color scale will be based on the 'Dea
                    hover_name="Country/Region",  # Display the country/region n
                    range_color=[1, 50],  # Set the range of colors on the color
                    color_continuous_scale="Peach",  # Use the 'Peach' color sca
                    title='Countries with Deaths Reported'  # Set the title of t
)

# Update the layout to center the title
fig.update_layout(
    title_text="Countries with Deaths Reported",
    title_x=0.5  # Set title_x to 0.5 to center the title
    #  title_x=0: the left
    # title_x=1: the right
)

# Hide the color scale legend, as the color scale is defined by 'color_continuou
fig.update(layout_coloraxis_showscale=False)
```

```
# Display the choropleth map
fig.show()
```

- Outside China, there hasn't been a lot of deaths due to COVID-19 has reported

Countries with all the cases recovered

In [ ]:
```
# Filter the 'row_latest_grouped' DataFrame to include rows where the number of
# This identifies countries where all reported cases have been recovered.
temp = row_latest_grouped[row_latest_grouped['Confirmed'] == row_latest_grouped[

# Extract specific columns ('Country/Region', 'Confirmed', and 'Recovered') from
temp = temp[['Country/Region', 'Confirmed', 'Recovered']]

# Sort the DataFrame by the 'Confirmed' column in descending order (highest firs
temp = temp.sort_values('Confirmed', ascending=False)

# Reset the index of the sorted DataFrame to start from 0, dropping the old inde
temp = temp.reset_index(drop=True)

# Display the sorted DataFrame with a styled background gradient using the 'Gree
# This highlights countries where all confirmed cases have been recovered.
temp.style.background_gradient(cmap='Greens')
```

Out[ ]:

	Country/Region	Confirmed	Recovered
0	Macau	10	10
1	Sri Lanka	1	1

In [ ]:
```
temp
```

Out[ ]:

	Country/Region	Confirmed	Recovered
0	Macau	10	10
1	Sri Lanka	1	1

Most Recent Stats

In [ ]:
```
# Group the 'full_table' DataFrame by the 'Date' column and sum the 'Confirmed',
temp = full_table.groupby('Date')['Confirmed', 'Deaths', 'Recovered'].sum()

# Reset the index of the aggregated DataFrame to make 'Date' a regular column
temp = temp.reset_index()

# Sort the aggregated DataFrame by the 'Date' column in descending order (most r
temp = temp.sort_values('Date', ascending=False)

# Display the first row of the sorted DataFrame with a styled background gradien
# This highlights the latest aggregated COVID-19 statistics.
temp.head(1).style.background_gradient(cmap='Pastel1')
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\4072283970.py:2: FutureWarning:

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

Out[ ]:

	Date	Confirmed	Deaths	Recovered
38	2020-02-11 10:50:00	43006	1018	4340

- There are more recovered cases than deaths at this point of time

Diamond Princess Cruise ship Status

In [ ]:
```
ship.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	R
71	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-21 22:00:00	0	0	
144	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-22 12:00:00	0	0	
217	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-23 12:00:00	0	0	
290	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-24 00:00:00	0	0	
363	Diamond Princess cruise ship	Others	35.4437	129.638	2020-01-24 12:00:00	0	0	

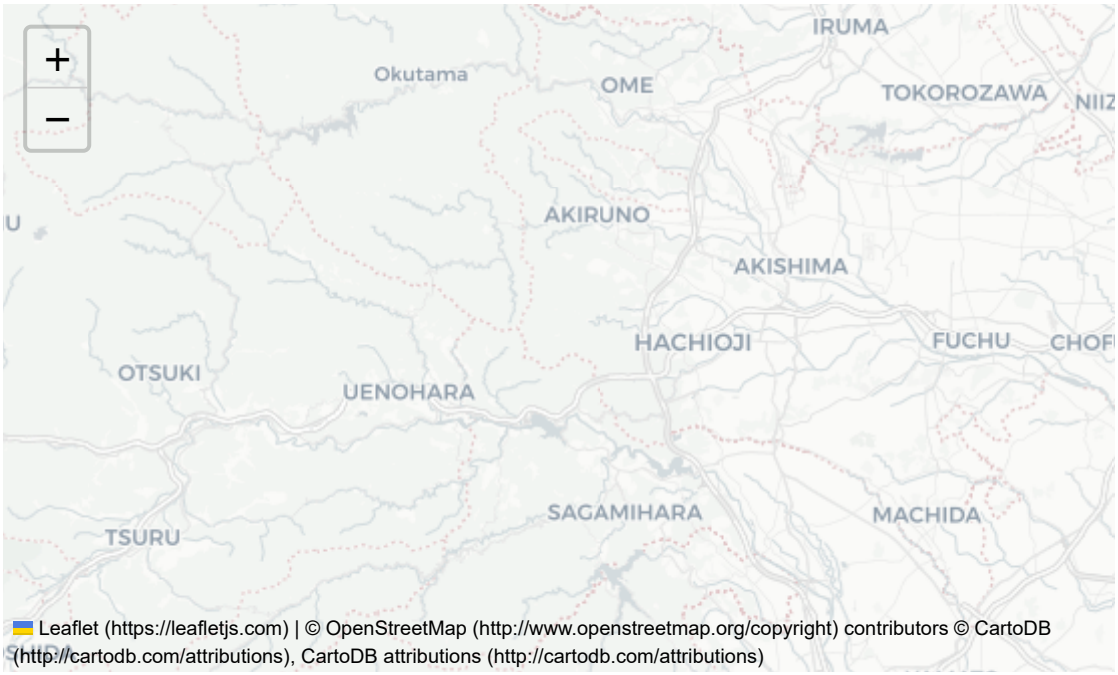
In [ ]:
```
# Cases in the Diamond Princess Cruise Ship

# Filter and display the latest COVID-19 statistics for the Diamond Princess Cru
# This code focuses on the most recent data available for the cruise ship.
# Note: The DataFrame 'ship' likely contains COVID-19 data specific to the Diamo

# Sort the 'ship' DataFrame by the 'Date' column in descending order to get the
temp = ship.sort_values(by='Date', ascending=False).head(1)

# Extract specific columns ('Province/State', 'Confirmed', 'Deaths', 'Recovered'
temp = temp[['Province/State', 'Confirmed', 'Deaths', 'Recovered']]

# Reset the index of the extracted data to start from 0, dropping the old index
temp = temp.reset_index(drop=True)

# Display the extracted data with a styled background gradient using the 'Pastel
# This highlights the latest COVID-19 statistics for the Diamond Princess Cruise
temp.style.background_gradient(cmap='Pastel1')
```

Out[ ]:

	Province/State	Confirmed	Deaths	Recovered
0	Diamond Princess cruise ship	135	0	0

In [ ]:
```
# China

temp = ship[ship['Date'] == max(ship['Date'])].reset_index()

m = folium.Map(location=[35.4437, 139.638], tiles='cartodbpositron',
               min_zoom=8, max_zoom=12, zoom_start=10)

folium.Circle(location=[temp.iloc[0]['Lat'], temp.iloc[0]['Long']],
        color='crimson',
        tooltip =   '<li><bold>Ship : '+str(temp.iloc[0]['Province/State'])+
                    '<li><bold>Confirmed : '+str(temp.iloc[0]['Confirmed'])+
                    '<li><bold>Deaths : '+str(temp.iloc[0]['Deaths'])+
                    '<li><bold>Recovered : '+str(temp.iloc[0]['Recovered']),
        radius=int(temp.iloc[0]['Confirmed'])**1).add_to(m)
m
```

Out[ ]:



- The ship was carrying 3,700 people in total
- https://www.princess.com/news/notices_and_advisories/notices/diamond-princess-update.html

In [ ]:
```
# Number of Countries/Regions to which COVID-19 spread
print(len(temp_f))
```

```
28
```

In [ ]:
```
# Number of Province/State in Mainland China to which COVID-19 spread
len(temp_c)
```

Out[ ]:
```
31
```

In [ ]:
```
# Number of countries with deaths reported
len(temp_flg)
```

Out[ ]: 3

Visual EDA

Spread Across the Globe

In [ ]:
```
# Format the COVID-19 data for visualization
# The code prepares data for a scatter geo-plot showing the spread of COVID-19 o

# Group the 'full_table' DataFrame by date and country/region, and extract maxim
formated_gdf = full_table.groupby(['Date', 'Country/Region'])['Confirmed', 'Deat

# Reset the index of the formatted DataFrame to make 'Date' and 'Country/Region'
formated_gdf = formated_gdf.reset_index()

# Exclude rows with the 'Country/Region' as 'China' from the formatted DataFrame
formated_gdf = formated_gdf[formated_gdf['Country/Region'] != 'China']

# Convert the 'Date' column to a datetime format
formated_gdf['Date'] = pd.to_datetime(formated_gdf['Date'])

# Format the 'Date' column as a string in the format 'MM/DD/YYYY'
formated_gdf['Date'] = formated_gdf['Date'].dt.strftime('%m/%d/%Y')

# Create a scatter geo-plot using Plotly Express to show the spread of COVID-19
fig = px.scatter_geo(formated_gdf[formated_gdf['Country/Region'] != 'China'],
                     locations="Country/Region", locationmode='country names',
                     color="Confirmed", size='Confirmed', hover_name="Country/Re
                     range_color=[0, max(formated_gdf['Confirmed']) + 2],
                     projection="natural earth", animation_frame="Date", # proje
                     title='Spread outside China over time')

# Hide the color scale legend in the plot, as the color scale is defined by the
fig.update(layout_coloraxis_showscale=False)

# Set the title_x parameter to 0.5 to center the title horizontally -->move the
fig.update_layout(title_x=0.5)


# Display the scatter geo-plot
fig.show()
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\1265295678.py:5: FutureWarning:

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

In [ ]:
```
# Create a scatter geo-plot to visualize the spread of COVID-19 in China over ti

# The code prepares data and creates an animated scatter geo-plot to show the sp
# in different provinces/states of China over multiple dates.

# Group the 'china' DataFrame by date and province/state, and extract maximum va
china_map = china.groupby(['Date', 'Province/State'])['Confirmed', 'Deaths', 'Re
```

```
# Reset the index of the formatted DataFrame to make 'Date' and 'Province/State'
china_map = china_map.reset_index()

# Calculate the size for markers based on the square root of confirmed cases
china_map['size'] = china_map['Confirmed'].pow(0.5)

# Convert the 'Date' column to a datetime format
china_map['Date'] = pd.to_datetime(china_map['Date'])

# Format the 'Date' column as a string in the format 'MM/DD/YYYY'
china_map['Date'] = china_map['Date'].dt.strftime('%m/%d/%Y')

# Display the first few rows of the prepared 'china_map' DataFrame
china_map.head()

# Create a scatter geo-plot using Plotly Express to show the spread of COVID-19
fig = px.scatter_geo(china_map, lat='Lat', lon='Long', scope='asia',           #
                     color="size", size='size', hover_name='Province/State',
                     hover_data=['Confirmed', 'Deaths', 'Recovered'],
                     projection="natural earth", animation_frame="Date",       #
                     title='Spread in China over time')

# Set the title_x parameter to 0.5 to center the title horizontally
fig.update_layout(title_x=0.5)

# Hide the color scale legend in the plot, as the color scale is defined by the
fig.update(layout_coloraxis_showscale=False)

# Display the scatter geo-plot
fig.show()
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\2858866747.py:7: FutureWarning:

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

Number of Places to which COVID-19 Spread

Number of Places to which COVID-19 Spread in China over time

In [ ]:
```
# Analyze the spread of COVID-19 across provinces/states/regions in China over t

# The code counts the number of unique provinces/states/regions in China
# where COVID-19 spread has been reported for each date.

# Filter the 'china' DataFrame to exclude rows with zero confirmed cases,
# then group by date and count the unique provinces/states/regions
c_spread = china[china['Confirmed'] != 0].groupby('Date')['Province/State'].unic
c_spread
```

Out[ ]:
```
Date
2020-01-21 22:00:00    12
2020-01-22 12:00:00    23
2020-01-23 12:00:00    28
2020-01-24 00:00:00    29
2020-01-24 12:00:00    29
2020-01-25 00:00:00    29
2020-01-25 12:00:00    30
2020-01-25 22:00:00    30
2020-01-26 11:00:00    30
2020-01-26 23:00:00    30
2020-01-27 09:00:00    30
2020-01-27 19:00:00    30
2020-01-27 20:30:00    30
2020-01-28 13:00:00    30
2020-01-28 18:00:00    30
2020-01-28 23:00:00    30
2020-01-29 13:30:00    30
2020-01-29 14:30:00    30
2020-01-29 21:00:00    31
2020-01-30 11:00:00    31
2020-01-31 14:00:00    31
2020-02-01 10:00:00    31
2020-02-02 21:00:00    31
2020-02-03 21:00:00    31
2020-02-04 09:40:00    31
2020-02-04 22:00:00    31
2020-02-05 09:00:00    31
2020-02-05 23:00:00    31
2020-02-06 09:00:00    31
2020-02-06 14:20:00    31
2020-02-07 20:13:00    31
2020-02-07 22:50:00    31
2020-02-08 22:04:00    31
2020-02-08 23:04:00    31
2020-02-09 10:30:00    31
2020-02-09 23:20:00    31
2020-02-10 10:30:00    31
2020-02-10 19:30:00    31
2020-02-11 10:50:00    31
Name: Province/State, dtype: int64
```

In [ ]:
```
# Convert the series to a DataFrame and reset the index to have date as a regula
c_spread = pd.DataFrame(c_spread).reset_index()
c_spread
```

Out[ ]:

	Date	Province/State
0	2020-01-21 22:00:00	12
1	2020-01-22 12:00:00	23
2	2020-01-23 12:00:00	28
3	2020-01-24 00:00:00	29
4	2020-01-24 12:00:00	29
5	2020-01-25 00:00:00	29
6	2020-01-25 12:00:00	30
7	2020-01-25 22:00:00	30
8	2020-01-26 11:00:00	30
9	2020-01-26 23:00:00	30
10	2020-01-27 09:00:00	30
11	2020-01-27 19:00:00	30
12	2020-01-27 20:30:00	30
13	2020-01-28 13:00:00	30
14	2020-01-28 18:00:00	30
15	2020-01-28 23:00:00	30
16	2020-01-29 13:30:00	30
17	2020-01-29 14:30:00	30
18	2020-01-29 21:00:00	31
19	2020-01-30 11:00:00	31
20	2020-01-31 14:00:00	31
21	2020-02-01 10:00:00	31
22	2020-02-02 21:00:00	31
23	2020-02-03 21:00:00	31
24	2020-02-04 09:40:00	31
25	2020-02-04 22:00:00	31
26	2020-02-05 09:00:00	31
27	2020-02-05 23:00:00	31
28	2020-02-06 09:00:00	31
29	2020-02-06 14:20:00	31
30	2020-02-07 20:13:00	31
31	2020-02-07 22:50:00	31
32	2020-02-08 22:04:00	31

	Date	Province/State
33	2020-02-08 23:04:00	31
34	2020-02-09 10:30:00	31
35	2020-02-09 23:20:00	31
36	2020-02-10 10:30:00	31
37	2020-02-10 19:30:00	31
38	2020-02-11 10:50:00	31

In [ ]:
```
# Create a line plot using Plotly Express to show the number of provinces/states
# to which COVID-19 spread over time
fig = px.line(c_spread, x='Date', y='Province/State',
              title='Number of Provinces/States/Regions of China to which COVID-

# Display the line plot
fig.show()
```

- COVID-19 spread to all the provinces of the China really fast and early

Number of Countries/Regions to which COVID-19 spread over the time

In [ ]:
```
# Analyze the global spread of COVID-19 across countries/regions over time

# The code counts the number of unique countries/regions where COVID-19 spread h

# Filter the 'full_table' DataFrame to exclude rows with zero confirmed cases,
# then group by date and count the unique countries/regions
spread = full_table[full_table['Confirmed'] != 0].groupby('Date')['Country/Regic
spread
```

Out[ ]:
```
Date
2020-01-21 22:00:00     5
2020-01-22 12:00:00     7
2020-01-23 12:00:00    10
2020-01-24 00:00:00    10
2020-01-24 12:00:00    11
2020-01-25 00:00:00    13
2020-01-25 12:00:00    13
2020-01-25 22:00:00    13
2020-01-26 11:00:00    15
2020-01-26 23:00:00    15
2020-01-27 09:00:00    16
2020-01-27 19:00:00    17
2020-01-27 20:30:00    18
2020-01-28 13:00:00    18
2020-01-28 18:00:00    18
2020-01-28 23:00:00    18
2020-01-29 13:30:00    21
2020-01-29 14:30:00    20
2020-01-29 21:00:00    20
2020-01-30 11:00:00    22
2020-01-31 14:00:00    26
2020-02-01 10:00:00    27
2020-02-02 21:00:00    27
2020-02-03 21:00:00    27
2020-02-04 09:40:00    28
2020-02-04 22:00:00    28
2020-02-05 09:00:00    28
2020-02-05 23:00:00    28
2020-02-06 09:00:00    28
2020-02-06 14:20:00    28
2020-02-07 20:13:00    28
2020-02-07 22:50:00    28
2020-02-08 22:04:00    28
2020-02-08 23:04:00    28
2020-02-09 10:30:00    28
2020-02-09 23:20:00    28
2020-02-10 10:30:00    28
2020-02-10 19:30:00    28
2020-02-11 10:50:00    28
Name: Country/Region, dtype: int64
```

In [ ]:
```
# Convert the series to a DataFrame and reset the index to have date as a regula
spread = pd.DataFrame(spread).reset_index()
spread
```

Out[ ]:

	Date	Country/Region
0	2020-01-21 22:00:00	5
1	2020-01-22 12:00:00	7
2	2020-01-23 12:00:00	10
3	2020-01-24 00:00:00	10
4	2020-01-24 12:00:00	11
5	2020-01-25 00:00:00	13
6	2020-01-25 12:00:00	13
7	2020-01-25 22:00:00	13
8	2020-01-26 11:00:00	15
9	2020-01-26 23:00:00	15
10	2020-01-27 09:00:00	16
11	2020-01-27 19:00:00	17
12	2020-01-27 20:30:00	18
13	2020-01-28 13:00:00	18
14	2020-01-28 18:00:00	18
15	2020-01-28 23:00:00	18
16	2020-01-29 13:30:00	21
17	2020-01-29 14:30:00	20
18	2020-01-29 21:00:00	20
19	2020-01-30 11:00:00	22
20	2020-01-31 14:00:00	26
21	2020-02-01 10:00:00	27
22	2020-02-02 21:00:00	27
23	2020-02-03 21:00:00	27
24	2020-02-04 09:40:00	28
25	2020-02-04 22:00:00	28
26	2020-02-05 09:00:00	28
27	2020-02-05 23:00:00	28
28	2020-02-06 09:00:00	28
29	2020-02-06 14:20:00	28
30	2020-02-07 20:13:00	28
31	2020-02-07 22:50:00	28
32	2020-02-08 22:04:00	28

	Date	Country/Region
33	2020-02-08 23:04:00	28
34	2020-02-09 10:30:00	28
35	2020-02-09 23:20:00	28
36	2020-02-10 10:30:00	28
37	2020-02-10 19:30:00	28
38	2020-02-11 10:50:00	28

In [ ]:
```
# Create a line plot using Plotly Express to show the number of countries/region
fig = px.line(spread, x='Date', y='Country/Region',
              title='Number of Countries/Regions to which COVID-19 spread over t

# Add label values (data values) to the plot, showing the numeric count for each
fig.update_traces(text=spread['Country/Region'], textposition='top center')


# Display the line plot
fig.show()
```

- Number of countries to which COVID-19 spread hasn't increased that much after first 2 weeks

Recovery and Mortality Rate Over The Time

In [ ]:
```
# Analyze recovery and mortality rates over time

# The code calculates various ratios related to COVID-19 recovery and mortality
# It then creates a line plot to visualize the trends of these ratios over time.

# Group the 'full_table' DataFrame by date and calculate the sum of each numeric
temp = full_table.groupby('Date').sum().reset_index()

# Display the first few rows of the resulting DataFrame
temp.head()
```

Out[ ]:

	Date	Lat	Long	Confirmed	Deaths	Recovered
0	2020-01-21 22:00:00	2174.89647	4408.81626	330	0	25
1	2020-01-22 12:00:00	2174.89647	4408.81626	555	0	28
2	2020-01-23 12:00:00	2174.89647	4408.81626	654	0	30
3	2020-01-24 00:00:00	2174.89647	4408.81626	881	26	34
4	2020-01-24 12:00:00	2174.89647	4408.81626	941	26	36

In [ ]:
```
# Calculate additional columns representing specific ratios
# 1. 'No. of Deaths to 100 Confirmed Cases': The percentage of deaths among conf
# 2. 'No. of Recovered to 100 Confirmed Cases': The percentage of recoveries amo
# 3. 'No. of Recovered to 1 Death Case': The ratio of recoveries to deaths (roun
```

```
temp['No. of Deaths to 100 Confirmed Cases'] = round(temp['Deaths'] / temp['Conf
temp['No. of Recovered to 100 Confirmed Cases'] = round(temp['Recovered'] / temp
temp['No. of Recovered to 1 Death Case'] = round(temp['Recovered'] / temp['Death

# Reshape the DataFrame using the 'melt' function to make it suitable for plotti
temp = temp.melt(id_vars='Date',
                 value_vars=['No. of Deaths to 100 Confirmed Cases',
                             'No. of Recovered to 100 Confirmed Cases',
                             'No. of Recovered to 1 Death Case'],
                 var_name='Ratio',
                 value_name='Value')

temp
```

Out[ ]:

	Date	Ratio	Value
0	2020-01-21 22:00:00	No. of Deaths to 100 Confirmed Cases	0.000
1	2020-01-22 12:00:00	No. of Deaths to 100 Confirmed Cases	0.000
2	2020-01-23 12:00:00	No. of Deaths to 100 Confirmed Cases	0.000
3	2020-01-24 00:00:00	No. of Deaths to 100 Confirmed Cases	3.000
4	2020-01-24 12:00:00	No. of Deaths to 100 Confirmed Cases	2.800
...	...	...	...
112	2020-02-09 10:30:00	No. of Recovered to 1 Death Case	3.590
113	2020-02-09 23:20:00	No. of Recovered to 1 Death Case	3.640
114	2020-02-10 10:30:00	No. of Recovered to 1 Death Case	3.932
115	2020-02-10 19:30:00	No. of Recovered to 1 Death Case	3.899
116	2020-02-11 10:50:00	No. of Recovered to 1 Death Case	4.263

117 rows × 3 columns

In [ ]:
```
# Create a line plot using Plotly Express to show the recovery and mortality rat
fig = px.line(temp, x="Date", y="Value", color='Ratio',
              title='Recovery and Mortality Rate Over Time')

# Display the line plot
fig.show()
```

- During the first few weeks the there were more Deaths reported per day than Recoverd cases
- Over the time that has changed drastically
- Although the death rate hasn't come down, the number of recovered cases has defenitly increased

Proportion of Cases

Number of Cases outside China

In [ ]:
```
# Analyze the number of cases outside China

# The code calculates and visualizes the number of confirmed, recovered, and dec

# Group the 'row_latest' DataFrame by 'Country/Region' and sum the columns 'Conf
rl = row_latest.groupby('Country/Region')['Confirmed', 'Deaths', 'Recovered'].su
rl
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\1403253331.py:6: FutureWarning:

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

Out[ ]:

Country/Region	Confirmed	Deaths	Recovered
Australia	15	0	2
Belgium	1	0	0
Cambodia	1	0	0
Canada	7	0	0
Finland	1	0	0
France	11	0	0
Germany	14	0	0
Hong Kong	49	0	3
India	3	0	0
Italy	3	0	0
Japan	26	0	1
Macau	10	0	10
Malaysia	18	0	3
Nepal	1	0	0
Philippines	3	1	0
Russia	2	0	0
Singapore	45	0	7
South Korea	28	0	1
Spain	2	0	0
Sri Lanka	1	0	1
Sweden	1	0	0
Taiwan	18	0	1
Thailand	32	1	0
UK	8	0	0
US	13	0	11
United Arab Emirates	8	0	0
Vietnam	15	0	1

In [ ]:
```
# Reset the index of the resulting DataFrame, sort it in descending order based
rl = rl.reset_index().sort_values(by='Confirmed', ascending=False).reset_index(d
rl
```

Out[ ]:

	Country/Region	Confirmed	Deaths	Recovered
0	Hong Kong	49	0	3
1	Singapore	45	0	7
2	Thailand	32	1	0
3	South Korea	28	0	1
4	Japan	26	0	1
5	Taiwan	18	0	1
6	Malaysia	18	0	3
7	Australia	15	0	2
8	Vietnam	15	0	1
9	Germany	14	0	0
10	US	13	0	11
11	France	11	0	0
12	Macau	10	0	10
13	United Arab Emirates	8	0	0
14	UK	8	0	0
15	Canada	7	0	0
16	Italy	3	0	0
17	Philippines	3	1	0
18	India	3	0	0
19	Spain	2	0	0
20	Russia	2	0	0
21	Sweden	1	0	0
22	Sri Lanka	1	0	1
23	Cambodia	1	0	0
24	Finland	1	0	0
25	Belgium	1	0	0
26	Nepal	1	0	0

In [ ]:
```
# Apply a background gradient to the top rows of the DataFrame for better visual
rl.head().style.background_gradient(cmap='rainbow')
```

Out[ ]:

	Country/Region	Confirmed	Deaths	Recovered
0	Hong Kong	49	0	3
1	Singapore	45	0	7
2	Thailand	32	1	0
3	South Korea	28	0	1
4	Japan	26	0	1

In [ ]:

```
# Create a copy of the DataFrame to calculate the number of affected cases (Conf
ncl = rl.copy()
ncl['Affected'] = ncl['Confirmed'] - ncl['Deaths'] - ncl['Recovered'] # 27 rows
ncl
```

Out[ ]:

	Country/Region	Confirmed	Deaths	Recovered	Affected
0	Hong Kong	49	0	3	46
1	Singapore	45	0	7	38
2	Thailand	32	1	0	31
3	South Korea	28	0	1	27
4	Japan	26	0	1	25
5	Taiwan	18	0	1	17
6	Malaysia	18	0	3	15
7	Australia	15	0	2	13
8	Vietnam	15	0	1	14
9	Germany	14	0	0	14
10	US	13	0	11	2
11	France	11	0	0	11
12	Macau	10	0	10	0
13	United Arab Emirates	8	0	0	8
14	UK	8	0	0	8
15	Canada	7	0	0	7
16	Italy	3	0	0	3
17	Philippines	3	1	0	2
18	India	3	0	0	3
19	Spain	2	0	0	2
20	Russia	2	0	0	2
21	Sweden	1	0	0	1
22	Sri Lanka	1	0	1	0
23	Cambodia	1	0	0	1
24	Finland	1	0	0	1
25	Belgium	1	0	0	1
26	Nepal	1	0	0	1

In [ ]:
```
# Reshape the DataFrame using the 'melt' function to make it suitable for bar pl
ncl = ncl.melt(id_vars="Country/Region", value_vars=['Affected', 'Recovered', 'D
ncl
```

Out[ ]:

	Country/Region	variable	value
0	Hong Kong	Affected	46
1	Singapore	Affected	38
2	Thailand	Affected	31
3	South Korea	Affected	27
4	Japan	Affected	25
...	...	...	...
76	Sri Lanka	Deaths	0
77	Cambodia	Deaths	0
78	Finland	Deaths	0
79	Belgium	Deaths	0
80	Nepal	Deaths	0

81 rows × 3 columns

In [ ]:
```
# Create a bar plot using Plotly Express to show the number of cases outside Chi
fig = px.bar(ncl.sort_values(['variable', 'value']),
             x="Country/Region", y="value", color='variable', orientation='v', h
             title='Number of Cases outside China', text='value')  # Add the tex

# Adjust the text size for labels to ensure readability
fig.update_layout(uniformtext_minsize=8, uniformtext_mode='hide')

# Move the title to the center
fig.update_layout(title_x=0.5)


# Display the bar plot
fig.show()
```

Number of Cases in China

In [ ]:
```
# Analyze the number of cases within China, specifically by province/state

# The code calculates and visualizes the number of confirmed, recovered, and dec
# focusing on the breakdown by province/state.

# Group the 'china_latest' DataFrame by 'Province/State' and sum the columns 'Co
cl = china_latest.groupby('Province/State')['Confirmed', 'Deaths', 'Recovered'].
cl
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\3335115484.py:7: FutureWarning:

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

Out[ ]:

Province/State	Confirmed	Deaths	Recovered
Anhui	860	4	105
Beijing	342	3	48
Chongqing	489	2	72
Fujian	267	0	45
Gansu	86	2	24
Guangdong	1177	1	212
Guangxi	215	1	33
Guizhou	127	1	17
Hainan	144	3	20
Hebei	239	2	48
Heilongjiang	360	8	28
Henan	1105	7	218
Hubei	31728	974	2310
Hunan	912	1	247
Inner Mongolia	58	0	5
Jiangsu	515	0	93
Jiangxi	804	1	128
Jilin	81	1	18
Liaoning	111	0	19
Ningxia	53	0	22
Qinghai	18	0	5
Shaanxi	219	0	32
Shandong	487	1	80
Shanghai	303	1	52
Shanxi	122	0	30
Sichuan	417	1	85
Tianjin	105	2	10
Tibet	1	0	0
Xinjiang	55	0	3
Yunnan	153	0	20
Zhejiang	1117	0	270

In [ ]:

```
# Reset the index of the resulting DataFrame, sort it in descending order based
cl = cl.reset_index().sort_values(by='Confirmed', ascending=False).reset_index(d

# Apply a background gradient to the top rows of the DataFrame for better visual
# cl.head().style.background_gradient(cmap='rainbow')
cl
```

Out[ ]:

	Province/State	Confirmed	Deaths	Recovered
0	Hubei	31728	974	2310
1	Guangdong	1177	1	212
2	Zhejiang	1117	0	270
3	Henan	1105	7	218
4	Hunan	912	1	247
5	Anhui	860	4	105
6	Jiangxi	804	1	128
7	Jiangsu	515	0	93
8	Chongqing	489	2	72
9	Shandong	487	1	80
10	Sichuan	417	1	85
11	Heilongjiang	360	8	28
12	Beijing	342	3	48
13	Shanghai	303	1	52
14	Fujian	267	0	45
15	Hebei	239	2	48
16	Shaanxi	219	0	32
17	Guangxi	215	1	33
18	Yunnan	153	0	20
19	Hainan	144	3	20
20	Guizhou	127	1	17
21	Shanxi	122	0	30
22	Liaoning	111	0	19
23	Tianjin	105	2	10
24	Gansu	86	2	24
25	Jilin	81	1	18
26	Inner Mongolia	58	0	5
27	Xinjiang	55	0	3
28	Ningxia	53	0	22
29	Qinghai	18	0	5
30	Tibet	1	0	0

In [ ]:
```
# Create a copy of the DataFrame to calculate the number of affected cases (Conf
ncl = cl.copy()
```

```
ncl['Affected'] = ncl['Confirmed'] - ncl['Deaths'] - ncl['Recovered']  # 31 rows
ncl
```

Out[ ]:

	Province/State	Confirmed	Deaths	Recovered	Affected
0	Hubei	31728	974	2310	28444
1	Guangdong	1177	1	212	964
2	Zhejiang	1117	0	270	847
3	Henan	1105	7	218	880
4	Hunan	912	1	247	664
5	Anhui	860	4	105	751
6	Jiangxi	804	1	128	675
7	Jiangsu	515	0	93	422
8	Chongqing	489	2	72	415
9	Shandong	487	1	80	406
10	Sichuan	417	1	85	331
11	Heilongjiang	360	8	28	324
12	Beijing	342	3	48	291
13	Shanghai	303	1	52	250
14	Fujian	267	0	45	222
15	Hebei	239	2	48	189
16	Shaanxi	219	0	32	187
17	Guangxi	215	1	33	181
18	Yunnan	153	0	20	133
19	Hainan	144	3	20	121
20	Guizhou	127	1	17	109
21	Shanxi	122	0	30	92
22	Liaoning	111	0	19	92
23	Tianjin	105	2	10	93
24	Gansu	86	2	24	60
25	Jilin	81	1	18	62
26	Inner Mongolia	58	0	5	53
27	Xinjiang	55	0	3	52
28	Ningxia	53	0	22	31
29	Qinghai	18	0	5	13
30	Tibet	1	0	0	1

In [ ]:
```
# Reshape the DataFrame using the 'melt' function to make it suitable for bar pl
ncl = ncl.melt(id_vars="Province/State", value_vars=['Affected', 'Recovered', 'D
ncl
```

Out[ ]:

	Province/State	variable	value
0	Hubei	Affected	28444
1	Guangdong	Affected	964
2	Zhejiang	Affected	847
3	Henan	Affected	880
4	Hunan	Affected	664
...	...	...	...
88	Inner Mongolia	Deaths	0
89	Xinjiang	Deaths	0
90	Ningxia	Deaths	0
91	Qinghai	Deaths	0
92	Tibet	Deaths	0

93 rows × 3 columns

In [ ]:
```
# Create a horizontal bar plot using Plotly Express to show the number of cases
fig = px.bar(ncl.sort_values(['variable', 'value']),
             y="Province/State", x="value", color='variable', orientation='h', h
             title='Number of Cases in China')

# Adjust the text size for labels to ensure readability
fig.update_layout(uniformtext_minsize=8, uniformtext_mode='hide')

# Display the bar plot
fig.show()
```

In [ ]:
```
# Prepare a summarized dataset for COVID-19 cases

# The code aggregates the 'full_table' dataset to create a summarized version wi
# deaths, and recovered cases for each combination of date and country/region.

# Group the 'full_table' DataFrame by 'Date' and 'Country/Region' and select the
gdf = full_table.groupby(['Date', 'Country/Region'])['Confirmed', 'Deaths', 'Rec

# Reset the index of the resulting DataFrame to have 'Date' and 'Country/Region'
gdf = gdf.reset_index()
gdf
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_7868\888619722.py:7: FutureWarning:

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

Out[ ]:

	Date	Country/Region	Confirmed	Deaths	Recovered
0	2020-01-21 22:00:00	Australia	0	0	0
1	2020-01-21 22:00:00	Belgium	0	0	0
2	2020-01-21 22:00:00	Cambodia	0	0	0
3	2020-01-21 22:00:00	Canada	0	0	0
4	2020-01-21 22:00:00	China	270	0	25
...	...	...	...	...	...
1087	2020-02-11 10:50:00	Thailand	32	1	0
1088	2020-02-11 10:50:00	UK	8	0	0
1089	2020-02-11 10:50:00	US	2	0	9
1090	2020-02-11 10:50:00	United Arab Emirates	8	0	0
1091	2020-02-11 10:50:00	Vietnam	15	0	1

1092 rows × 5 columns

In [ ]:
```
# Reshape the DataFrame using the melt function to make it suitable for further

# The melt function is used to reshape the 'temp' DataFrame from a wide format t
# This transformation is useful when we want to work with the data in a way that

# The function 'melt' gathers columns (in this case, the columns 'No. of Deaths
# 'No. of Recovered to 100 Confirmed Cases', 'No. of Recovered to 1 Death Case')
# and their corresponding values into another column ('Value').

# This reshaping makes it easier to compare and analyze the different ratios ove

temp.melt
```

Out[ ]:
```
<bound method DataFrame.melt of                    Date
Ratio  Value
0   2020-01-21 22:00:00  No. of Deaths to 100 Confirmed Cases  0.000
1   2020-01-22 12:00:00  No. of Deaths to 100 Confirmed Cases  0.000
2   2020-01-23 12:00:00  No. of Deaths to 100 Confirmed Cases  0.000
3   2020-01-24 00:00:00  No. of Deaths to 100 Confirmed Cases  3.000
4   2020-01-24 12:00:00  No. of Deaths to 100 Confirmed Cases  2.800
..                  ...                                   ...    ...
112 2020-02-09 10:30:00      No. of Recovered to 1 Death Case  3.590
113 2020-02-09 23:20:00      No. of Recovered to 1 Death Case  3.640
114 2020-02-10 10:30:00      No. of Recovered to 1 Death Case  3.932
115 2020-02-10 19:30:00      No. of Recovered to 1 Death Case  3.899
116 2020-02-11 10:50:00      No. of Recovered to 1 Death Case  4.263

[117 rows x 3 columns]>
```

In [ ]:
```
# Analyze COVID-19 cases in China

# The code focuses on analyzing COVID-19 cases specifically in China.
# It retrieves the data for China from the 'gdf' DataFrame, reshapes it, and cre

# Filter the 'gdf' DataFrame to select rows where the 'Country/Region' is 'China
```

```
temp = gdf[gdf['Country/Region'] == 'China'].reset_index()

# Reshape the data using the 'melt' function to transform the columns 'Confirmed
temp = temp.melt(id_vars='Date', value_vars=['Confirmed', 'Deaths', 'Recovered']
                 var_name='Case', value_name='Count')

# Create a bar plot using Plotly Express to show the counts of confirmed, deaths
# The facet_col parameter splits the plot into facets (subplots) based on the 'C
fig = px.bar(temp, x="Date", y="Count", color='Case', facet_col="Case",
             title='Cases in China')

# Display the bar plot
fig.show()
```

In [ ]:
```
# Analyze COVID-19 cases outside of China

# The code focuses on analyzing COVID-19 cases outside of China.
# It aggregates and summarizes the data for countries/regions other than China f
# reshapes it, and creates a bar plot to visualize the cases.

# Filter the 'gdf' DataFrame to select rows where the 'Country/Region' is not 'C
temp = gdf[gdf['Country/Region'] != 'China'].groupby('Date').sum().reset_index()

# Reshape the data using the 'melt' function to transform the columns 'Confirmed
temp = temp.melt(id_vars='Date', value_vars=['Confirmed', 'Deaths', 'Recovered']
                 var_name='Case', value_name='Count')

# Create a bar plot using Plotly Express to show the counts of confirmed, deaths
# The facet_col parameter splits the plot into facets (subplots) based on the 'C
fig = px.bar(temp, x="Date", y="Count", color='Case', facet_col="Case",
             title='Cases Outside China')

# Display the bar plot
fig.show()
```

In [ ]:
```
# Visualize COVID-19 cases in Chinese provinces using treemaps

# The code creates three separate treemaps to visualize the number of confirmed
# in different Chinese provinces. Each treemap provides insight into a specific

# Create a treemap to show the number of confirmed cases in Chinese provinces
fig = px.treemap(china_latest.sort_values(by='Confirmed', ascending=False).reset
                 path=["Province/State"], values="Confirmed", title='Number of C

# Move the title to the center
fig.update_layout(title_x=0.5)

# Display the treemap for confirmed cases
fig.show()
#---------------------

# Create a treemap to show the number of deaths reported in Chinese provinces
fig = px.treemap(china_latest.sort_values(by='Deaths', ascending=False).reset_ir
                 path=["Province/State"], values="Deaths", title='Number of Deat

# Move the title to the center
fig.update_layout(title_x=0.5)

# Display the treemap for deaths
```

```
fig.show()
#----------------------

# Create a treemap to show the number of recovered cases in Chinese provinces
fig = px.treemap(china_latest.sort_values(by='Recovered', ascending=False).reset
                 path=["Province/State"], values="Recovered", title='Number of R

# Move the title to the center
fig.update_layout(title_x=0.5)

# Display the treemap for recovered cases
fig.show()
```

Outside China

In [ ]:

```
#import plotly.express as px

# Visualize COVID-19 cases outside of China using treemaps

# The code creates three separate treemaps to visualize the number of confirmed
# in countries/regions outside of China. Each treemap provides insight into a sp

# Create a treemap to show the number of confirmed cases outside of China
fig = px.treemap(row_latest, path=["Country/Region"],
                 values="Confirmed", title='Number of Confirmed Cases outside Ch
                 labels={"Confirmed": "Number of Confirmed"}) # Adding label for

# Move the title to the center
fig.update_layout(title_x=0.5)

# Display the treemap for confirmed cases
fig.show()


# Create a treemap to show the number of deaths reported outside of China
fig = px.treemap(row_latest, path=["Country/Region"],
                 values="Deaths", title='Number of Deaths outside China',
                 labels={"Deaths": "Number of Deaths"})  # Adding label for the

# Move the title to the center
fig.update_layout(title_x=0.5)

# Display the treemap for deaths
fig.show()

# Create a treemap to show the number of recovered cases outside of China
fig = px.treemap(row_latest, path=["Country/Region"],
                 values="Recovered", title='Number of Recovered Cases outside Ch
                 labels={"Recovered": "Number of Recovered"})  # Adding label fo


# Move the title to the center
fig.update_layout(title_x=0.5)

# Display the treemap for recovered cases
fig.show()
```

Many thanks to Devakumar and Sash