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	1/21/20 22:00	1/22/20 12:00	1/23/20 12:00	1/2
0	Anhui	Mainland China	31.82571	117.2264	NaN	1.0	9.0	
1	Beijing	Mainland China	40.18238	116.4142	10.0	14.0	22.0	
2	Chongqing	Mainland China	30.05718	107.8740	5.0	6.0	9.0	
3	Fujian	Mainland China	26.07783	117.9895	NaN	1.0	5.0	
4	Gansu	Mainland China	36.06110	103.8343	NaN	NaN	2.0	

5 rows × 43 columns

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

Out[ ]:
```
Index(['Province/State', 'Country/Region', 'Lat', 'Long', '1/21/20 22:00',
       '1/22/20 12:00', '1/23/20 12:00', '1/24/20 0:00', '1/24/20 12:00',
       '1/25/20 0:00', '1/25/20 12:00', '1/25/20 22:00', '1/26/20 11:00',
       '1/26/20 23:00', '1/27/20 9:00', '1/27/20 19:00', '1/27/20 20:30',
       '1/28/20 13:00', '1/28/20 18:00', '1/28/20 23:00', '1/29/20 13:30',
       '1/29/20 14:30', '1/29/20 21:00', '1/30/20 11:00', '1/31/20 14:00',
       '2/1/20 10:00', '2/2/20 21:00', '2/3/20 21:00', '2/4/20 9:40',
       '2/4/20 22:00', '2/5/20 9:00', '2/5/20 23:00', '2/6/20 9:00',
       '2/6/20 14:20', '2/7/20 20:13', '2/7/20 22:50', '2/8/20 22:04',
       '2/8/20 23:04', '2/9/20 10:30', '2/9/20 23:20', '2/10/20 10:30',
       '2/10/20 19:30', '2/11/20 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   1/21/20 22:00   16 non-null     float64
 5   1/22/20 12:00   29 non-null     float64
 6   1/23/20 12:00   37 non-null     float64
 7   1/24/20 0:00    38 non-null     float64
 8   1/24/20 12:00   40 non-null     float64
 9   1/25/20 0:00    42 non-null     float64
 10  1/25/20 12:00   43 non-null     float64
 11  1/25/20 22:00   43 non-null     float64
 12  1/26/20 11:00   49 non-null     float64
 13  1/26/20 23:00   49 non-null     float64
 14  1/27/20 9:00    50 non-null     float64
 15  1/27/20 19:00   51 non-null     float64
 16  1/27/20 20:30   52 non-null     float64
 17  1/28/20 13:00   53 non-null     float64
 18  1/28/20 18:00   53 non-null     float64
 19  1/28/20 23:00   53 non-null     float64
 20  1/29/20 13:30   56 non-null     float64
 21  1/29/20 14:30   55 non-null     float64
 22  1/29/20 21:00   57 non-null     float64
 23  1/30/20 11:00   59 non-null     float64
 24  1/31/20 14:00   65 non-null     float64
 25  2/1/20 10:00    67 non-null     float64
 26  2/2/20 21:00    68 non-null     float64
 27  2/3/20 21:00    69 non-null     float64
 28  2/4/20 9:40     70 non-null     float64
 29  2/4/20 22:00    70 non-null     float64
 30  2/5/20 9:00     70 non-null     float64
 31  2/5/20 23:00    71 non-null     float64
 32  2/6/20 9:00     71 non-null     float64
 33  2/6/20 14:20    71 non-null     float64
 34  2/7/20 20:13    72 non-null     float64
 35  2/7/20 22:50    72 non-null     float64
 36  2/8/20 22:04    72 non-null     float64
 37  2/8/20 23:04    72 non-null     float64
 38  2/9/20 10:30    72 non-null     float64
 39  2/9/20 23:20    72 non-null     float64
 40  2/10/20 10:30   72 non-null     float64
 41  2/10/20 19:30   72 non-null     float64
 42  2/11/20 10:50   73 non-null     int64
dtypes: float64(40), int64(1), object(2)
memory usage: 24.6+ KB
```

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

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	Re
0	Anhui	Mainland China	31.82571	117.2264	1/21/20 22:00	NaN	NaN	
1	Beijing	Mainland China	40.18238	116.4142	1/21/20 22:00	10.0	NaN	
2	Chongqing	Mainland China	30.05718	107.8740	1/21/20 22:00	5.0	NaN	
3	Fujian	Mainland China	26.07783	117.9895	1/21/20 22:00	NaN	NaN	
4	Gansu	Mainland China	36.06110	103.8343	1/21/20 22:00	NaN	NaN	

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')
# 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

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

# 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	0.0	
1	Beijing	China	40.18238	116.4142	2020-01-21 22:00:00	10.0	0.0	
2	Chongqing	China	30.05718	107.8740	2020-01-21 22:00:00	5.0	0.0	
3	Fujian	China	26.07783	117.9895	2020-01-21 22:00:00	0.0	0.0	
4	Gansu	China	36.06110	103.8343	2020-01-21 22:00:00	0.0	0.0	

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

# full table
full_table = full_table[full_table['Province/State']!='Diamond Princess cruise s
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	0.0	
1	Beijing	China	40.18238	116.4142	2020-01-21 22:00:00	10.0	0.0	
2	Chongqing	China	30.05718	107.8740	2020-01-21 22:00:00	5.0	0.0	
3	Fujian	China	26.07783	117.9895	2020-01-21 22:00:00	0.0	0.0	
4	Gansu	China	36.06110	103.8343	2020-01-21 22:00:00	0.0	0.0	

In [ ]:
```
# full_table.info()
```

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_1088\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_1088\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_1088\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_1088\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	4.000000	0.000000	2
	Queensland	5.000000	0.000000	0
	South Australia	2.000000	0.000000	0
	Victoria	4.000000	0.000000	0
Belgium	NA	1.000000	0.000000	0
Cambodia	NA	1.000000	0.000000	0
Canada	British Columbia	4.000000	0.000000	0
	London, ON	1.000000	0.000000	0
	Toronto, ON	2.000000	0.000000	0
China	Anhui	860.000000	4.000000	105
	Beijing	342.000000	3.000000	48
	Chongqing	489.000000	2.000000	72
	Fujian	267.000000	0.000000	45
	Gansu	86.000000	2.000000	24
	Guangdong	1177.000000	1.000000	212
	Guangxi	215.000000	1.000000	33
	Guizhou	127.000000	1.000000	17
	Hainan	144.000000	3.000000	20
	Hebei	239.000000	2.000000	48
	Heilongjiang	360.000000	8.000000	28
	Henan	1105.000000	7.000000	218
	Hubei	31728.000000	974.000000	2310
	Hunan	912.000000	1.000000	247
	Inner Mongolia	58.000000	0.000000	5
	Jiangsu	515.000000	0.000000	93
	Jiangxi	804.000000	1.000000	128
	Jilin	81.000000	1.000000	18
	Liaoning	111.000000	0.000000	19
	Ningxia	53.000000	0.000000	22
	Qinghai	18.000000	0.000000	5
	Shaanxi	219.000000	0.000000	32
	Shandong	487.000000	1.000000	80

Country/Region	Province/State	Confirmed	Deaths	Recovered
	Shanghai	303.000000	1.000000	52
	Shanxi	122.000000	0.000000	30
	Sichuan	417.000000	1.000000	85
	Tianjin	105.000000	2.000000	10
	Tibet	1.000000	0.000000	0
	Xinjiang	55.000000	0.000000	3
	Yunnan	153.000000	0.000000	20
	Zhejiang	1117.000000	0.000000	270
Finland	NA	1.000000	0.000000	0
France	NA	11.000000	0.000000	0
Germany	NA	14.000000	0.000000	0
Hong Kong	Hong Kong	49.000000	0.000000	3
India	NA	3.000000	0.000000	0
Italy	NA	3.000000	0.000000	0
Japan	NA	26.000000	0.000000	1
Macau	Macau	10.000000	0.000000	10
Malaysia	NA	18.000000	0.000000	3
Nepal	NA	1.000000	0.000000	0
Philippines	NA	3.000000	1.000000	0
Russia	NA	2.000000	0.000000	0
Singapore	NA	45.000000	0.000000	7
South Korea	NA	28.000000	0.000000	1
Spain	NA	2.000000	0.000000	0
Sri Lanka	NA	1.000000	0.000000	1
Sweden	NA	1.000000	0.000000	0
Taiwan	Taiwan	18.000000	0.000000	1
Thailand	NA	32.000000	1.000000	0
UK	NA	8.000000	0.000000	0
US	Boston, MA	1.000000	0.000000	0
	Chicago, IL	2.000000	0.000000	0
	Los Angeles, CA	1.000000	0.000000	0
	Madison, WI	1.000000	0.000000	0

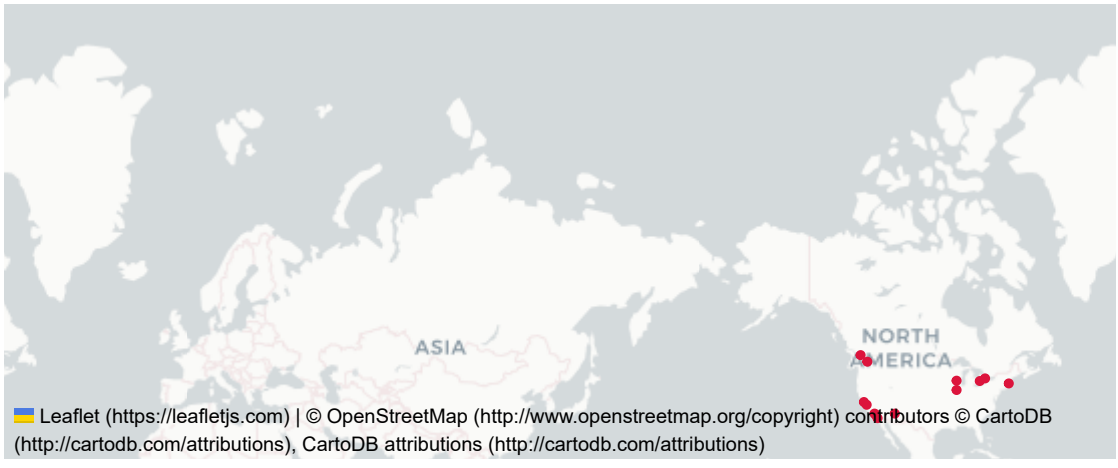
Country/Region	Province/State	Confirmed	Deaths	Recovered
	Orange, CA	1.000000	0.000000	0
	San Benito, CA	2.000000	0.000000	0
	San Diego County, CA	1.000000	0.000000	0
	Santa Clara, CA	2.000000	0.000000	0
	Seattle, WA	1.000000	0.000000	2
	Tempe, AZ	1.000000	0.000000	9
United Arab Emirates	NA	8.000000	0.000000	0
Vietnam	NA	15.000000	0.000000	1

In [ ]:
```
# Create a folium map centered at [0, 0] with specified tile style and zoom leve
m = folium.Map(location=[0, 0], tiles='cartodbpositron', min_zoom=1, max_zoom=4,

# Iterate over each row in the 'full_latest' DataFrame
for i in range(0, len(full_latest)):
    # Create a folium Circle marker for each data point
    folium.Circle(
        location=[full_latest.iloc[i]['Lat'], full_latest.iloc[i]['Long']],
        color='crimson',  # Circle color
        tooltip =    '<li><b>Country: '+str(full_latest.iloc[i]['Country/Region']
                     '<li><b>Province: '+str(full_latest.iloc[i]['Province/State'
                     '<li><b>Confirmed: '+str(full_latest.iloc[i]['Confirmed'])+
                     '<li><b>Deaths: '+str(full_latest.iloc[i]['Deaths'])+
                     '<li><b>Recovered: '+str(full_latest.iloc[i]['Recovered']),
        radius=int(full_latest.iloc[i]['Confirmed'])  # Circle radius based on c
    ).add_to(m)  # Add the circle to the map 'm'

# 'm' now contains circles representing COVID-19 data on the map
m
```

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



Top 10 Countries with most no. of reported cases

In [ ]:
```
# Create a DataFrame 'temp_f' containing columns 'Country/Region' and 'Confirmed
temp_f = full_latest_grouped[['Country/Region', 'Confirmed']]

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

# Reset the index of the sorted DataFrame to start from 0 and drop the previous
temp_f = temp_f.reset_index(drop=True)

# Select the top 10 rows from the sorted DataFrame
top_10 = temp_f.head(10)

# Apply a background gradient style to the 'top_10' DataFrame
styled_top_10 = top_10.style.background_gradient(cmap='Pastel1_r')

# The styled DataFrame 'styled_top_10' now has the background gradient applied
# It can be displayed to visualize the data with color-coded cells
styled_top_10
```

Out[ ]:

	Country/Region	Confirmed
0	China	42670.000000
1	Hong Kong	49.000000
2	Singapore	45.000000
3	Thailand	32.000000
4	South Korea	28.000000
5	Japan	26.000000
6	Taiwan	18.000000
7	Malaysia	18.000000
8	Australia	15.000000
9	Vietnam	15.000000

- 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 [ ]:
```
# Create a choropleth map using Plotly Express
fig = px.choropleth(
    full_latest_grouped,  # DataFrame containing data
    locations="Country/Region",  # Column specifying country/region names
    locationmode='country names',  # Use country names as location mode
    color="Confirmed",  # Column to determine color intensity
    hover_name="Country/Region",  # Column to display on hover
    range_color=[1, 50],  # Color scale range
    color_continuous_scale="Sunsetdark",  # Color scale for the map
    title='Countries with Confirmed Cases',  # Title for the map
    labels={'Confirmed': 'Confirmed Cases'}  # Label for the color scale legend
```

```
)

# Set the label values to be displayed on each country/region
#fig.update_traces(text=full_latest_grouped['Confirmed'])

# Hide the color scale legend to make the visualization cleaner
fig.update(layout_coloraxis_showscale=False)

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

In [ ]:
```
# Create a choropleth map using Plotly Express
fig = px.choropleth(
    full_latest_grouped,  # DataFrame containing data
    locations="Country/Region",  # Column specifying country/region names
    locationmode='country names',  # Use country names as location mode
    color="Confirmed",  # Column to determine color intensity
    hover_name="Country/Region",  # Column to display on hover
    range_color=[1, 50],  # Color scale range
    color_continuous_scale="Sunsetdark",  # Color scale for the map
    title='Countries with Confirmed Cases'  # Title for the map
)

# Hide the color scale legend to make the visualization cleaner
# fig.update(layout_coloraxis_showscale=True)
fig.update(layout_coloraxis_showscale=False)
# Set the label values to be displayed on each country/region
fig.update_traces(text=full_latest_grouped['Confirmed'])

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

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

In [ ]:
```
# Create a DataFrame 'temp_c' containing columns 'Province/State' and 'Confirmed
temp_c = china_latest_grouped[['Province/State', 'Confirmed']]

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

# Reset the index of the sorted DataFrame to start from 0 and drop the previous
temp_c = temp_c.reset_index(drop=True)

# Select the top 10 rows from the sorted DataFrame
top_10_china = temp_c.head(10)

# Apply a background gradient style to the 'top_10_china' DataFrame
styled_top_10_china = top_10_china.style.background_gradient(cmap='Pastel1_r')

# The styled DataFrame 'styled_top_10_china' now has the background gradient app
# It can be displayed to visualize the data with color-coded cells
styled_top_10_china
```

Out[ ]:

	Province/State	Confirmed
0	Hubei	31728.000000
1	Guangdong	1177.000000
2	Zhejiang	1117.000000
3	Henan	1105.000000
4	Hunan	912.000000
5	Anhui	860.000000
6	Jiangxi	804.000000
7	Jiangsu	515.000000
8	Chongqing	489.000000
9	Shandong	487.000000

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='crimson',
        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 [ ]:
```
# Create a DataFrame 'temp_flg' containing columns 'Country/Region' and 'Deaths'
temp_flg = full_latest_grouped[['Country/Region', 'Deaths']]

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

# Reset the index of the sorted DataFrame to start from 0 and drop the previous
temp_flg = temp_flg.reset_index(drop=True)

# Filter out rows where 'Deaths' is greater than 0
temp_flg = temp_flg[temp_flg['Deaths'] > 0]

# Apply a background gradient style to the 'temp_flg' DataFrame
styled_temp_flg = temp_flg.style.background_gradient(cmap='Pastel1_r')

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

Out[ ]:

	Country/Region	Deaths
0	China	1016.000000
1	Philippines	1.000000
2	Thailand	1.000000

In [ ]:
```
# Create a choropleth map using Plotly Express
fig = px.choropleth(
    full_latest_grouped[full_latest_grouped['Deaths'] > 0],  # DataFrame filtered
    locations="Country/Region",  # Column specifying country/region names
    locationmode='country names',  # Use country names as location mode
    color="Deaths",  # Column to determine color intensity
    hover_name="Country/Region",  # Column to display on hover
    range_color=[1, 50],  # Color scale range
    color_continuous_scale="Peach",  # Color scale for the map
    title='Countries with Deaths Reported'  # Title for the map
)

# Hide the color scale legend to make the visualization cleaner
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 [ ]:
```
# Countries with all the cases recovered
temp = row_latest_grouped[row_latest_grouped['Confirmed']==row_latest_grouped['R
temp = temp[['Country/Region', 'Confirmed', 'Recovered']]
temp = temp.sort_values('Confirmed', ascending=False)
temp = temp.reset_index(drop=True)
temp.style.background_gradient(cmap='Greens')
```

Out[ ]:

	Country/Region	Confirmed	Recovered
0	Macau	10.000000	10
1	Sri Lanka	1.000000	1

Most Recent Stats

In [ ]:
```
temp = full_table.groupby('Date')['Confirmed', 'Deaths', 'Recovered'].sum()
temp = temp.reset_index()
temp = temp.sort_values('Date', ascending=False)
temp.head(1).style.background_gradient(cmap='Pastel1')
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_1088\894495828.py:1: 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	43006.000000	1018.000000	4340

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

Diamond Princess Cruise ship Status

In [ ]:
```
# Cases in the Diamond Princess Cruise Ship
temp = ship.sort_values(by='Date', ascending=False).head(1)
temp = temp[['Province/State', 'Confirmed', 'Deaths', 'Recovered']].reset_index(
temp.style.background_gradient(cmap='Pastel1')
```

Out[ ]:

	Province/State	Confirmed	Deaths	Recovered

In [ ]:
```
# Filter the 'ship' DataFrame to get the latest data
temp = ship[ship['Date'] == max(ship['Date'])].reset_index()

# Create a folium map centered at [35.4437, 139.638] with specified tile style a
m = folium.Map(location=[35.4437, 139.638], tiles='cartodbpositron', min_zoom=8,

# Create a folium Circle marker for the latest ship data
folium.Circle(
    location=[temp.iloc[0]['Lat'], temp.iloc[0]['Long']],
    color='crimson',  # Circle color
    tooltip =   '<li><b>Ship: '+str(temp.iloc[0]['Province/State'])+
                '<li><b>Confirmed: '+str(temp.iloc[0]['Confirmed'])+
                '<li><b>Deaths: '+str(temp.iloc[0]['Deaths'])+
```

```
                '<li><b>Recovered: '+str(temp.iloc[0]['Recovered']),
    radius=int(temp.iloc[0]['Confirmed'])**1  # Circle radius based on confirmed
).add_to(m)  # Add the circle to the map 'm'

# Display the map with the circle marker
m
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
d:\Data science & Python  2022\0. Data Analyst_2023\Portfolio_Thach\I. Beginner level\COVID-19 - Analysis, Visualization & Comparisons\Covid-19 analysis visualization comparisons.ipynb Cell 45 in 2
      <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/COVID-19%20-%20Analysis%2C%20Visualization%20%26%20Comparisons/Covid-19%20analysis%20visualization%20comparisons.ipynb#Y145sZmlsZQ%3D%3D?line=0'>1</a> # Filter the 'ship' DataFrame to get the latest data
----> <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/COVID-19%20-%20Analysis%2C%20Visualization%20%26%20Comparisons/Covid-19%20analysis%20visualization%20comparisons.ipynb#Y145sZmlsZQ%3D%3D?line=1'>2</a> temp = ship[ship['Date'] == max(ship['Date'])].reset_index()
      <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/COVID-19%20-%20Analysis%2C%20Visualization%20%26%20Comparisons/Covid-19%20analysis%20visualization%20comparisons.ipynb#Y145sZmlsZQ%3D%3D?line=3'>4</a> # Create a folium map centered at [35.4437, 139.638] with specified tile style and zoom levels
      <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/COVID-19%20-%20Analysis%2C%20Visualization%20%26%20Comparisons/Covid-19%20analysis%20visualization%20comparisons.ipynb#Y145sZmlsZQ%3D%3D?line=4'>5</a> m = folium.Map(location=[35.4437, 139.638], tiles='cartodbpositron', min_zoom=8, max_zoom=12, zoom_start=10)

ValueError: max() arg is an empty sequence
```

- 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 [ ]:
```
formated_gdf = full_table.groupby(['Date', 'Country/Region'])['Confirmed', 'Deat
formated_gdf = formated_gdf.reset_index()
formated_gdf = formated_gdf[formated_gdf['Country/Region']!='China']
formated_gdf['Date'] = pd.to_datetime(formated_gdf['Date'])
formated_gdf['Date'] = formated_gdf['Date'].dt.strftime('%m/%d/%Y')

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",
                     title='Spread outside China over time')
fig.update(layout_coloraxis_showscale=False)
fig.show()

# ----------------------------------------------------------------------------

china_map = china.groupby(['Date', 'Province/State'])['Confirmed', 'Deaths', 'Re
                                                      'Lat', 'Long'].max()
china_map = china_map.reset_index()
china_map['size'] = china_map['Confirmed'].pow(0.5)
china_map['Date'] = pd.to_datetime(china_map['Date'])
china_map['Date'] = china_map['Date'].dt.strftime('%m/%d/%Y')
china_map.head()

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')
fig.update(layout_coloraxis_showscale=False)
fig.show()
```

Number of Places to which COVID-19 Spread

In [ ]:
```
c_spread = china[china['Confirmed']!=0].groupby('Date')['Province/State'].unique
c_spread = pd.DataFrame(c_spread).reset_index()

fig = px.line(c_spread, x='Date', y='Province/State',
              title='Number of Provinces/States/Regions of China to which COVID-
fig.show()
```

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

In [ ]:
```
spread = full_table[full_table['Confirmed']!=0].groupby('Date')['Country/Region'
spread = pd.DataFrame(spread).reset_index()

fig = px.line(spread, x='Date', y='Country/Region',
              title='Number of Countries/Regions to which COVID-19 spread over t
fig.show()
```

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

Recovery and Mortality Rate Over The Time

In [ ]:
```
temp = full_table.groupby('Date').sum().reset_index()
temp.head()

# adding two more columns
temp['No. of Deaths to 100 Confirmed Cases'] = round(temp['Deaths']/temp['Confir
temp['No. of Recovered to 100 Confirmed Cases'] = round(temp['Recovered']/temp['
temp['No. of Recovered to 1 Death Case'] = round(temp['Recovered']/temp['Deaths'

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')
fig = px.line(temp, x="Date", y="Value", color='Ratio',
              title='Recovery and Mortality Rate Over The Time')
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

In [ ]:
```
rl = row_latest.groupby('Country/Region')['Confirmed', 'Deaths', 'Recovered'].su
rl = rl.reset_index().sort_values(by='Confirmed', ascending=False).reset_index(d
rl.head().style.background_gradient(cmap='rainbow')

ncl = rl.copy()
ncl['Affected'] = ncl['Confirmed'] - ncl['Deaths'] - ncl['Recovered']
ncl = ncl.melt(id_vars="Country/Region", value_vars=['Affected', 'Recovered', 'D

fig = px.bar(ncl.sort_values(['variable', 'value']),
             x="Country/Region", y="value", color='variable', orientation='v', h
             # height=600, width=1000,
             title='Number of Cases outside China')
fig.update_layout(uniformtext_minsize=8, uniformtext_mode='hide')
fig.show()

# ----------------------------------------
```

```
cl = china_latest.groupby('Province/State')['Confirmed', 'Deaths', 'Recovered'].
cl = cl.reset_index().sort_values(by='Confirmed', ascending=False).reset_index(d
# cl.head().style.background_gradient(cmap='rainbow')

ncl = cl.copy()
ncl['Affected'] = ncl['Confirmed'] - ncl['Deaths'] - ncl['Recovered']
ncl = ncl.melt(id_vars="Province/State", value_vars=['Affected', 'Recovered', 'D

fig = px.bar(ncl.sort_values(['variable', 'value']),
             y="Province/State", x="value", color='variable', orientation='h', h
             # height=600, width=1000,
             title='Number of Cases in China')
fig.update_layout(uniformtext_minsize=8, uniformtext_mode='hide')
fig.show()
```

In [ ]:
```
gdf = gdf = full_table.groupby(['Date', 'Country/Region'])['Confirmed', 'Deaths'
gdf = gdf.reset_index()
```

In [ ]:
```
temp.melt?
```

In [ ]:
```
temp = gdf[gdf['Country/Region']=='China'].reset_index()
temp = temp.melt(id_vars='Date', value_vars=['Confirmed', 'Deaths', 'Recovered']
                 var_name='Case', value_name='Count')
fig = px.bar(temp, x="Date", y="Count", color='Case', facet_col="Case",
             title='Cases in China')
fig.show()
```

In [ ]:
```
temp = gdf[gdf['Country/Region']!='China'].groupby('Date').sum().reset_index()
temp = temp.melt(id_vars='Date', value_vars=['Confirmed', 'Deaths', 'Recovered']
                 var_name='Case', value_name='Count')
fig = px.bar(temp, x="Date", y="Count", color='Case', facet_col="Case",
             title='Cases Outside China')
fig.show()
```

In [ ]:

```
fig = px.treemap(china_latest.sort_values(by='Confirmed', ascending=False).reset
           path=["Province/State"], values="Confirmed", title='Number of Confirm
fig.show()

fig = px.treemap(china_latest.sort_values(by='Deaths', ascending=False).reset_ir
           path=["Province/State"], values="Deaths", title='Number of Deaths Rep
fig.show()

fig = px.treemap(china_latest.sort_values(by='Recovered', ascending=False).reset
           path=["Province/State"], values="Recovered", title='Number of Recover
fig.show()
```

In [ ]:

```
fig = px.treemap(row_latest, path=["Country/Region"],
                 values="Confirmed", title='Number of Confirmed Cases outside ch
fig.show()

fig = px.treemap(row_latest, path=["Country/Region"],
                 values="Deaths", title='Number of Deaths outside china')
fig.show()

fig = px.treemap(row_latest, path=["Country/Region"],
                 values="Recovered", title='Number of Recovered Cases outside ch
fig.show()
```