In [ ]:

```
##import all the packages that needed
import matplotlib.pyplot as plt
import pandas_bokeh
from bokeh.plotting import figure, output_file,show

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from functools import wraps
import plotly.express as px
```

In [ ]:

```
#Read in the Data
data = pd.read_csv('owid-covid-data.csv')
pd.set_option('display.max_columns', None)
data
```

In [ ]:

```
# Check the total number of missing values
data.isnull().sum().sum()
```

In [ ]:

```
#preview of the dataset
#check the column name to have a breif understand of the features
data.info()
```

In [ ]:

```
#Convert the format of 'date' (from object to datetime)
#https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html
data['date'] = pd.to_datetime(data['date'])
```

In [ ]:

```
#Check the total number of countries in the data and the number of times they ar
data['location'].value_counts()
```

In [ ]:

```
data.describe()
```

Data Cleaning Process

While doing data inspection, we found out that in "location" coloumn, there are some countries named "Asia", "Africa", etc., these are not country names but continent names, and others named as "high income", "low income", etc. With country names like these, they don't have corresponding continnent names in the "continent" column. **Since the research question is country based, not continent based or income based, we should drop rows that don't have appropriate location names.**

In [ ]:

```
des = data.describe()
des.loc['skew', : ] = data.skew()
des.loc['kurt',: ] = data.kurt()
des

#If skewness value is greater than 1 or less than -1 indicates a highly skewed d
#A value between 0.5 and 1 or -0.5 and -1 is moderately skewed.
#A value between -0.5 and 0.5 indicates that the distribution is fairly symmetri
#for the extreme skewness, log transformation is needed.
```

```
#If kurtosis value is greater than +1, the distribution is too peaked.
#Likewise, a kurtosis of less than -1 indicates a distribution that is too flat.
#data need to be transformed into its power of 1/2 or 1/4.
```

In [ ]:

```
data1_countryname = data[data["continent"].notna()]
```

Trend of the number of confirmed cases from 2020 to October 2022

In [ ]:

```
data_trend = pd.DataFrame()

data_trend['Time'] =  data['date']
data_trend['Confirmedcase'] = data['total_cases']
data_trend['Country'] = data['location']
data_trend['Continent'] = data['continent']

data_trend
```

In [ ]:

```
#Make a figure that shows the overall trend of the number of confirmed cases
plt.figure(figsize = (15, 10))
sns.set(style='darkgrid')
sns.lineplot( x = 'Time', y = 'Confirmedcase', data = data_trend, color='red')
plt.xlabel('Time', fontsize = 15)
plt.ylabel('Numbers of Confirmed Cases', fontsize = 15)
plt.title('Trend of the number of confirmed cases from Jan.2020 to Oct. 2022', f
plt.show()
```

In [ ]:

```
#to see the differences of confirmed case between country
data1_countryname = data[data["continent"].notna()]
fig = px.line(data_frame = data1_countryname, x = 'date', y = 'total_cases', col
        title = 'Trends in the total number of confirmed cases by country From J
fig.update_layout( xaxis_title = "Time",yaxis_title = "Number of confirmed cases

fig.show()
```

Question 1

In [ ]:

```
data_q1 = data.query('location == ["Japan", "United Kingdom","India","Australia"
```

In [ ]:

```
#for comparison use
plt.figure(figsize = (10,8))
sns.lineplot(data = data_q1, x = 'date', y ='total_deaths_per_million', hue = 'l
plt.ylabel('Total numbers of death per million', fontsize = 13)
plt.xlabel('Time', fontsize = 13)
plt.title('Total numbers of death in 5 countries from Jan.2020 to Oct. 2022', fo
plt.show()
```

In [ ]:

```
#for comparison use
fig,axs = plt.subplots (nrows = 2, ncols = 2, figsize = (8,5))
plt.subplots_adjust(top = 2 , right = 2)
```

```
sns.barplot(data = data_q1, x = 'location', y ='gdp_per_capita', alpha = 0.8,
            ax = axs [0][0])
sns.barplot(data = data_q1, x = 'location', y ='human_development_index',
            alpha = 0.8, ax=axs[0][1])
sns.barplot(data = data_q1, x = 'location', y ='hospital_beds_per_thousand',
            alpha = 0.8, ax=axs[1][0])
sns.barplot(data = data_q1, x = 'location', y ='population_density',
            alpha = 0.8, ax=axs[1][1])
axs[0][0].set_title('GDP per capital', size = 15)
axs[0][1].set_title('Human Development Index', size = 15)
axs[1][0].set_title('Hospital beds per thousand', size = 15)
axs[1][1].set_title('Population density', size = 15)
```

In [ ]:

```
#Add a new feature - death rate
data1_countryname['death_rate'] = ((data1_countryname['total_deaths']/data1_coun
data1_countryname['death_rate']
```

In [ ]:

```
#Make a barplot that can indicates the diffrences of death rate betwen country
data_q1 = data1_countryname.query('location == ["Japan", "United Kingdom","India

fig = sns.barplot(data = data_q1, x = 'location', y ='death_rate', alpha = 0.8)

plt.xlabel ('Country')
plt.ylabel('Death Rate (%)')
plt.title('Death Rate', fontsize = 15)
```

In [ ]:

```
#Check the correlation between these two variables
data_q1 = data1_countryname.query('location == ["Japan", "United Kingdom","India

df = px.data.tips()
fig = px.scatter(data_q1, x='icu_patients_per_million', y="new_deaths", color ='
                title = "Number of COVID - 19 patients in ICU Vs Number of new d
fig.update_layout( xaxis_title = "Number of COVID-19 patients in ICU",
                  yaxis_title = "Number of new deaths")
fig.show()
```

In [ ]:

```
#for comparison use
data_q1 = data1_countryname.query('location == ["Japan", "United Kingdom","India
fig = sns.barplot(data = data_q1, x = 'location', y ='death_rate', alpha = 0.8)

plt.xlabel ('Country')
plt.xticks(rotation = 40)
plt.ylabel('Death Rate (%)')
plt.title('Death Rate', fontsize = 15)
```

Question 2

US

In [ ]:

```
#Trend of total cases and total deaths, in US
data_US = data.query('location == ["United States"]')

sns.set(style = 'darkgrid')

Time = data_US['date']
```

```
Cases = data_US['total_cases_per_million']
Death = data_US['total_deaths_per_million']

#Initialize figure and axis
fig, ax = plt.subplots(figsize = (8,8))

#Plot lines
ax.plot(Time, Cases, color = 'green')
ax.plot(Time, Death, color = 'red')

#Fill area when confirmed case > death with green
ax.fill_between(Time, Cases, Death, where = (Cases > Death),
               interpolate = True, color = 'green', alpha = 0.2,
               label = 'Death < Cases')

#Fill area when death > confirmed case with red
ax.fill_between(Time, Cases, Death, where = (Cases <= Death),
               interpolate = True, color = 'red',alpha = 0.2,
               label = 'Death > Cases')

#Add labels, title
ax.set_xlabel ('Time')
ax.set_ylabel('Total Cases and Total Deaths per million')
ax.set_title('US Total Cases Vs Total Deaths: 2020-2022 ', fontsize = 15)
ax.legend()
```

In [ ]:

```
data_US['new_cases']
```

In [ ]:

```
data_US['new_deaths']
```

In [ ]:

```
#for comparison use
data_US = data.query('location == ["United States"]')

sns.set(style = 'darkgrid')

Time = data_US['date']
newCases = data_US['new_cases_per_million']
newDeath = data_US['new_deaths_per_million']

#Initialize figure and axis
fig, ax = plt.subplots(figsize = (8,8))

#Plot lines
ax.plot(Time, newCases, color = 'green')
ax.plot(Time, newDeath, color = 'red')

#Fill area when confirmed case > death with green
ax.fill_between(Time, newCases, newDeath, where = (Cases > Death),
               interpolate = True, color = 'green', alpha = 0.2,
               label = 'Death < Cases')

#Fill area when death > confirmed case with red
ax.fill_between(Time, newCases, newDeath, where = (Cases <= Death),
               interpolate = True, color = 'red',alpha = 0.2,
               label = 'Death > Cases')

#Add labels, title
ax.set_xlabel ('Time')
ax.set_ylabel('New Cases and New Deaths')
ax.set_title('US New Cases Vs New Deaths: 2020-2022 ', fontsize = 15)
ax.legend()
```

China

In [ ]:

```
#for comparison use
#Trend of total cases and total deaths, in China
data_CN = data.query('location == ["China"]')

sns.set(style = 'darkgrid')

Time = data_CN['date']
CasesCN = data_CN['total_cases_per_million']
DeathCN = data_CN['total_deaths_per_million']

#Initialize figure and axis
fig, ax = plt.subplots(figsize = (8,8))

#Plot lines
ax.plot(Time, CasesCN, color = 'green')
ax.plot(Time, DeathCN, color = 'red')

#Fill area when confirmed case > death with green
ax.fill_between(Time, CasesCN, DeathCN, where = (Cases > Death),
               interpolate = True, color = 'green', alpha = 0.2,
               label = 'Death < Cases')

#Fill area when death > confirmed case with red
ax.fill_between(Time, CasesCN, DeathCN, where = (Cases <= Death),
               interpolate = True, color = 'red',alpha = 0.2,
               label = 'Death > Cases')

#Add labels, title
ax.set_xlabel ('Time')
ax.set_ylabel('Total Cases and Total Deaths per million')
ax.set_title('China Total Cases Vs Total Deaths: 2020-2022 ', fontsize = 15)
ax.legend()
```

In [ ]:

```
#for comparison use
data_CN = data.query('location == ["China"]')

sns.set(style = 'darkgrid')

Time = data_CN['date']
newCasesCN = data_CN['new_cases_per_million']
newDeathCN = data_CN['new_deaths_per_million']

#Initialize figure and axis
fig, ax = plt.subplots(figsize = (8,8))

#Plot lines
ax.plot(Time, newCasesCN, color = 'green')
ax.plot(Time, newDeathCN, color = 'red')

#Fill area when confirmed case > death with green
ax.fill_between(Time, newCasesCN, newDeathCN, where = (Cases > Death),
               interpolate = True, color = 'green', alpha = 0.2,
               label = 'Death < Cases')
```

```
#Fill area when death > confirmed case with red
ax.fill_between(Time, newCasesCN, newDeathCN, where = (Cases <= Death),
               interpolate = True, color = 'red',alpha = 0.2,
               label = 'Death > Cases')

#Add labels, title
ax.set_xlabel ('Time')
ax.set_ylabel('New Cases and New Deaths')
ax.set_title('China New Cases Vs New Deaths: 2020-2022 ', fontsize = 15)
ax.legend()
```

Correlation between variables

In [ ]:

```
#Make a new DataFrame that ready to be use in this section
data_corr =  data[['total_cases','new_cases','total_deaths','new_deaths',
                   'total_cases_per_million','new_cases_per_million','total_deat
                  'reproduction_rate','icu_patients','icu_patients_per_million',
                  'hosp_patients_per_million','weekly_icu_admissions','weekly_ic
                  'weekly_hosp_admissions','weekly_hosp_admissions_per_million',
                  'new_tests','total_tests_per_thousand','new_tests_per_thousand
                  'tests_per_case','tests_units','total_vaccinations','people_va
                  'people_fully_vaccinated','total_boosters','new_vaccinations',
                  'stringency_index','population','population_density','median_a
                  'aged_65_older','aged_70_older','gdp_per_capita','extreme_pove
                   'cardiovasc_death_rate','diabetes_prevalence','female_smokers
                  'male_smokers','handwashing_facilities','hospital_beds_per_tho
                  'life_expectancy','human_development_index','excess_mortality'
```

In [ ]:

```
#Heatmap use to check the correlations between variables
corr = data_corr.corr()
n_var = len(corr)

plt.figure(figsize = (25,20))
plt.imshow(corr, cmap = 'winter')

plt.xticks(range(n_var), corr.columns, rotation = 90)
plt.yticks(range(n_var), corr.columns)

for i in range (n_var):
    for j in range(n_var):
        plt.text(i, j, '{:.2f}'.format(corr.iloc[i,j]), ha = "center", va="cente


plt.colorbar()
plt.title("Correlations", fontsize = 20)
plt.show()
```

In [ ]:

```
#Check the correlation between total cases and other
Case_n_other = data_corr.corr()['total_cases'].sort_values(ascending = False).ro
Case_n_other
```

In [ ]:

```
#Add a new feature - People fully vaccinated rate
data1_countryname['fully_vaccinated_rate'] = ((data1_countryname['people_fully_v
data1_countryname['infection_rate'] = ((data1_countryname['total_cases']/data1_c
```

In [ ]:

```
#Plot that shows relationship between Government response stringency index and t
# All countries
fig = px.scatter (data1_countryname, x = 'total_cases_per_million', y ='stringen
                 color = 'location', title = "Total Cases Vs Government Response
fig.update_layout( xaxis_title = "Total Confirmed Cases per million",
                  yaxis_title = "Government Response Stringency Index ")
fig.show()
```

In [ ]:

```
#Plot that shows relationship between Government response stringency index and t
#Selected Countries
fig = px.scatter (data_q1, x = 'total_cases_per_million', y ='stringency_index',
                 color = 'location', title = "Total Cases Vs Government Response
fig.update_layout( xaxis_title = "Total Confirmed Cases per million",
                  yaxis_title = "Government Response Stringency Index ")
fig.show()
```

In [ ]:

```
#check the correlation between people get vaccinated and total cases of that cou
fig = px.scatter(data1_countryname, x = 'people_vaccinated', y ='total_cases',
                 color = 'location',title = "People vaccinated Vs Confirmed Case
fig.update_layout( xaxis_title = "Total number of people who received at least o
                  yaxis_title = "Total number of confirmed cases ")
fig.show()
```

Dashboard

In [ ]:

```
##import all the packages that needed
import plotly.express as px
import pandas as pd
import numpy as np
import math
import pandas as pd

import plotly.express as px
import plotly.io as pio
from plotly import graph_objects as go
pio.renderers.default = 'browser'

df= pd.read_csv('owid-covid-data.csv')
```

In [ ]:

```
#Preview of the data
df.head()
```

In [ ]:

```
#Get the name of each column
df.columns
```

In [ ]:

```
#Dealing with missing values
df['total_cases'] = df['total_cases'].fillna(0)

df = df.sort_values('date')
```

In [ ]:

```
#Get readt for draw the first figure
df_fig1 = df[['date','iso_code','continent','total_cases',
        'location','total_deaths','new_cases','new_deaths']]
```

In [ ]:
```
#get a pandas pivot tables to check the data
data= data1_countryname.pivot_table(index='location',columns='date',values='peop
data
```

In [ ]:
```
head_20_index = data.loc[:,'2022-10-18'].sort_values(ascending = False).head(20)
head_20 = data.loc[head_20_index]
head_20
```

In [ ]:
```
#The second figure
fig2 = go.Figure()

x= head_20.columns
for idx in head_20.index:

    fig2.add_trace(go.Scatter(x =x, y = head_20.loc[idx].values.tolist(),
                              text = head_20.loc[idx].values.tolist(),
                              mode = 'lines+markers',name= idx,
                              hovertext = 'people_fully_vaccinated',
                              hoverinfo = 'all',opacity=0.5))
```

In [ ]:
```
##import all the packages that needed
import dash
import flask
import dash_core_components as dcc
from dash import html
server = flask.Flask(__name__)
```

In [ ]:
```
World_df = df[df['location']=="World"][['date','total_deaths','total_cases']].ta
World_df
```

In [ ]:
```
#Third figure
fig3 = px.bar(World_df,  x='date',  y=['total_deaths','total_cases'],
              title="Total Number of Confirmed Cases and Deaths Worldwide ",
              barmode='group' )
```

In [ ]:
```
#make a dataframe that contains all the data needed to make figure 3
icu_df = df[['date','location','weekly_icu_admissions','weekly_hosp_admissions_p
icu_df
```

In [ ]:
```
icu_df['year'] = icu_df['date'].map(lambda x:x[:4])
icu_df
```

In [ ]:
```
icu2022 = icu_df.groupby(['location','year']).sum().reset_index().query('year=="
icu2022
```

In [ ]:
```
fig4 = px.bar(icu2022,  x='location',
              y=['weekly_icu_admissions','weekly_hosp_admissions_per_million'],
              title="Proportion of ICU admissions to total hopistal cases ", bar
```

In [ ]:
```
#!pip install dash_bootstrap_components
```

In [ ]:
```
from dash.dependencies import Input, Output
import dash
from dash.dependencies import Input, Output
import dash_core_components as dcc
```

```
from dash import html
import dash_bootstrap_components as dbc
```

In [ ]:

```
fig = px.scatter_geo(df_fig1, locations="iso_code", color="total_cases",
                     color_continuous_scale=px.colors.sequential.OrRd,
                     hover_name="location",
                     hover_data={"total_cases":True,"new_cases": True,"total_dea
                                 "new_deaths":True,
                                 'iso_code':False},projection="natural earth",
                     animation_frame='date',animation_group='location')
def graph1():
    return dcc.Graph(id='graph1',figure=fig)

external_stylesheets = [dbc.themes.BOOTSTRAP]

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

server = app.server
app.layout = html.Div(children=[html.Div(children='''Global impact of COVID-19 '
    dcc.Graph(
        id='graph1',
        figure=fig), html.Div(children=''''''),dcc.Graph(
        id='graph2',
        figure=fig2),
    dcc.Graph(
        id='graph3',
        figure=fig3),
    dcc.Graph(
        id='graph4',
        figure=fig4)])
if __name__ == '__main__':
    app.run_server(debug=False,port=8832, use_reloader=False)
```

In [ ]:

```
#------------The End of the Assignment
```

In [ ]:

In [ ]: