ICP-Data Integration and Visulaization Summary 1) Importing libraries

2) Reading Covid Data sets

3) Data filtering

4) Data Integration using merge() ,join() and concatenate

5) Da ib, pandas and seaborn (pie chart, line chart, heatmaps etc).

Run cell (Ctrl+Enter)
cell executed since last change

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

```
impor
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import matplotlib.ticker as ticker
import seaborn as sns
```

```
df = pd.read_csv('https://raw.githubusercontent.com/M3IT/COVID-19_Data/master/Data/COVID_AU_cumulative.csv')
df1 = pd.read_csv('https://raw.githubusercontent.com/M3IT/COVID-19_Data/master/Data/COVID_AU_national.csv')
```

```
print(df.shape)
print(df1.shape)
```

```
(11592, 18)
(1288, 19)
```

```
df
```

	date	confirmed	deaths	tests	positives	recovered	hosp	icu	vent	vaccines	people_vaccinated	people_fully_vaccinated	popu
0	2020-01-25	4	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0	25
1	2020-01-26	4	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0	25
2	2020-01-27	5	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0	25
3	2020-01-28	5	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0	25
4	2020-01-29	9	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0	25
...	...	...	...	...	...	...	...	...	...	...	...	...	
11587	2023-08-04	1725730	2954	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	5
11588	2023-08-04	932940	1548	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	1
11589	2023-08-04	302852	277	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	
11590	2023-08-04	2978839	7454	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	6
11591	2023-08-04	1350729	1080	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	2

11592 rows × 18 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df1
```

	date	confirmed	confirmed_cum	deaths	deaths_cum	tests	tests_cum	positives	positives_cum	recovered	recovered_cum	hosp	ho:
0	2020-01-25	4	4	0	0	NaN	0.0	4	0.0	NaN	0.0	0	
1	2020-01-26	0	4	0	0	NaN	0.0	0	0.0	NaN	0.0	0	
Run cell (Ctrl+Enter) cell executed since last change			5	0	0	NaN	0.0	1	0.0	NaN	0.0	0	
executed by Bhagya rekha Ammisetty 11:35 PM (0 minutes ago) executed in 0.353s			5	0	0	NaN	0.0	0	0.0	NaN	0.0	0	
4	2020-01-29	4	9	0	0	NaN	0.0	4	0.0	NaN	0.0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1283	2023-07-31	0	11731441	0	19999	NaN	0.0	0	0.0	NaN	0.0	0	
1284	2023-08-01	0	11731441	0	19999	NaN	0.0	0	0.0	NaN	0.0	0	
1285	2023-08-02	0	11731441	0	19999	NaN	0.0	0	0.0	NaN	0.0	0	
1286	2023-08-03	0	11731441	0	19999	NaN	0.0	0	0.0	NaN	0.0	0	
1287	2023-08-04	5082	11736523	0	19999	NaN	NaN	5082	NaN	NaN	NaN	0	

1288 rows × 19 columns

Next steps: Generate code with df1 | View recommended plots | New interactive sheet

Lets remove all the non-cumulative columns from the df 1 data frame

```
df_filtered=df1.drop(['confirmed','deaths','tests','positives','recovered','hosp', 'icu','vent'], axis=1)
df_filtered
```

	date	confirmed_cum	deaths_cum	tests_cum	positives_cum	recovered_cum	hosp_cum	icu_cum	vent_cum	vaccines	vaccines_cum
0	2020-01-25	4	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
1	2020-01-26	4	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
2	2020-01-27	5	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
3	2020-01-28	5	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
4	2020-01-29	9	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
1283	2023-07-31	11731441	19999	0.0	0.0	0.0	0.0	0.0	0.0	0	0
1284	2023-08-01	11731441	19999	0.0	0.0	0.0	0.0	0.0	0.0	0	0

Next steps: Generate code with df_filtered | View recommended plots | New interactive sheet

** Data Integration**

Pandas merge(): Combining Data on Common Columns or Indices. The first technique you'll learn is merge(). You can use merge() any time you want to do database-like join operations. It's the most flexible joining operation (the other are join() and concat()).

How to merge()

Before getting into the details of how to use merge(), you should first understand the various forms of joins:

> inner

> outer

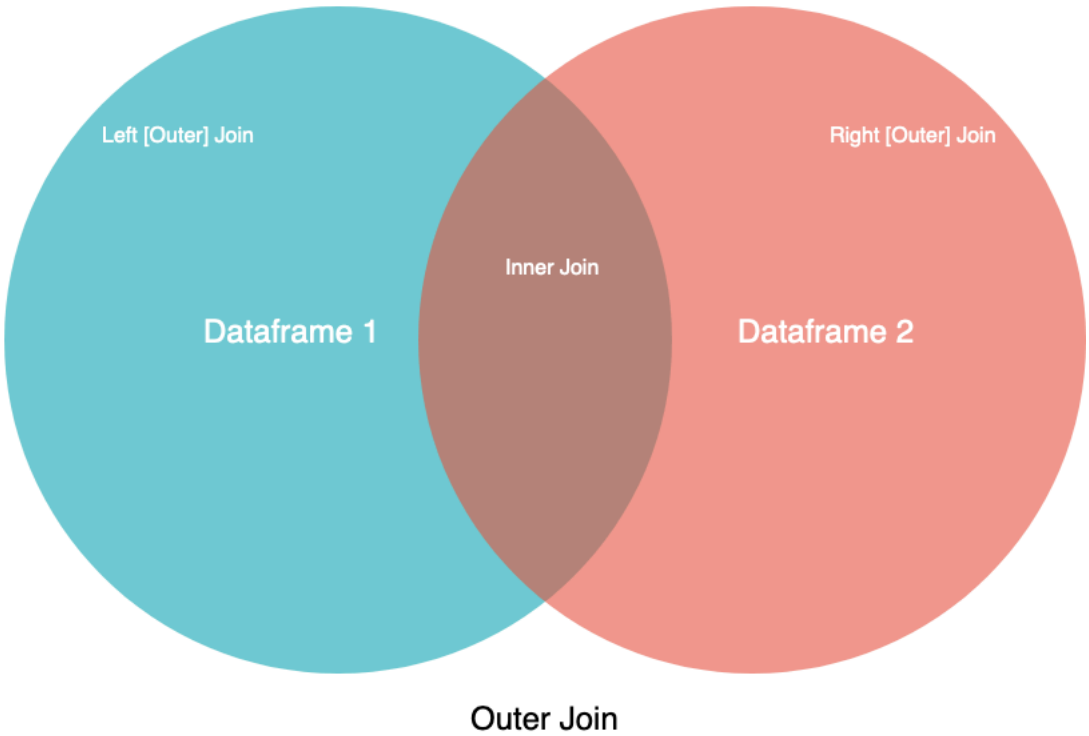
> left

> right

Outer Join

Here, Run cell (Ctrl+Enter) cell executed since last change the how parameter. Remember from the diagrams below that in an outer join (also known as a full outer join), executed by Bhagya rekha Ammisetty 11:35 PM (0 minutes ago) executed in 0.353s will be present in the new DataFrame.

If a r ot her DataFrame (based on the key column[s]), then you won't lose the row like you would with an inner join. Instead, the row will be in the merged DataFrame with NaN values filled in where appropriate.




```
outer_merged = pd.merge(df, df_filtered, how="outer", on=["date"])
outer_merged.head()
```

	date	confirmed	deaths	tests	positives	recovered	hosp	icu	vent	vaccines_x	...	confirmed_cum	deaths_cum	tests_cum	positive
0	2020-01-25	4	0	0.0	0.0	0.0	0.0	0.0	0.0	0	...	4	0	0.0	
1	2020-01-25	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	...	4	0	0.0	
2	2020-01-25	3	0	0.0	0.0	0.0	0.0	0.0	0.0	0	...	4	0	0.0	
3	2020-01-25	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	...	4	0	0.0	
4	2020-01-25	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	...	4	0	0.0	

5 rows × 28 columns

```
outer_merged.shape
```

```
(11592, 28)
```

Concatenating objects:

The concat() function (in the main pandas namespace) does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes. Note that I say “if any” because there is only a single possible axis of concatenation for Series.

Before diving into all of the details of concat and what it can do, here is a simple example:

source : link text

Run cell (Ctrl+Enter)
cell executed since last change

```
df1_a
```

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

```
'A3"],
', "B3"],
', "C3"],
        "D": ["D0", "D1", "D2", "D3"],
      },
      index=[0, 1, 2, 3],
 )

df2_a = pd.DataFrame(
    {
          "A": ["A4", "A5", "A6", "A7"],
           "B": ["B4", "B5", "B6", "B7"],
           "C": ["C4", "C5", "C6", "C7"],
           "D": ["D4", "D5", "D6", "D7"],
       },
        index=[4, 5, 6, 7],
    )


df3_a = pd.DataFrame(
       {
            "A": ["A8", "A9", "A10", "A11"],
            "B": ["B8", "B9", "B10", "B11"],
            "C": ["C8", "C9", "C10", "C11"],
            "D": ["D8", "D9", "D10", "D11"],
       },
        index=[8, 9, 10, 11],
    )


frames = [df1_a, df2_a, df3_a]
# row wise concat
result = pd.concat(frames)
```

```
df1_a
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

Next steps: Generate code with df1_a | View recommended plots | New interactive sheet

```
df2_a
```

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

Next steps: Generate code with df2_a | View recommended plots | New interactive sheet

```
df3_a
```

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10

Run cell (Ctrl+Enter)
cell executed since last change

Next executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

_a View recommended plots New interactive sheet

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

```
result
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Next steps: Generate code with `result` View recommended plots New interactive sheet

```
s1 = pd.Series(["X0", "X1", "X2", "X3"], name="X")
s2 = pd.Series(["_0", "_1", "_2", "_3"])
#column wise concat
result = pd.concat([df1_a, s1, s2], axis=1)
```

```
s1
```

	X
0	X0
1	X1
2	X2

Run cell (Ctrl+Enter)
cell executed since last change

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

```
s2
```

	0
0	_0
1	_1
2	_2
3	_3

```
result
```

	A	B	C	D	X	0
0	A0	B0	C0	D0	X0	_0
1	A1	B1	C1	D1	X1	_1
2	A2	B2	C2	D2	X2	_2
3	A3	B3	C3	D3	X3	3

Next steps: Generate code with `result` | View recommended plots | New interactive sheet

Data Visulaization

We'll be using data from Github repository that auto-updates the data daily. We'll load our data into a Pandas' dataframe based on the URL so that it'll update automatically for us every day.

```
df_global = pd.read_csv('https://raw.githubusercontent.com/datasets/covid-19/master/data/countries-aggregated.csv', parse_dates=['Date'])
df_global
```

	Date	Country	Confirmed	Recovered	Deaths
0	2020-01-22	Afghanistan	0	0	0
1	2020-01-23	Afghanistan	0	0	0
2	2020-01-24	Afghanistan	0	0	0
3	2020-01-25	Afghanistan	0	0	0
4	2020-01-26	Afghanistan	0	0	0
...	...	...	...	...	...
161563	2022-04-12	Zimbabwe	247094	0	5460
161564	2022-04-13	Zimbabwe	247160	0	5460
161565	2022-04-14	Zimbabwe	247208	0	5462
161566	2022-04-15	Zimbabwe	247237	0	5462
161567	2022-04-16	Zimbabwe	247237	0	5462

161568 rows x 5 columns

we read in the data into a dataframe df_global, and then select only the countries in our list countries. Selecting the data makes the resulting visualization a little more readable.

we create a summary column that aggregates the total number of cases across our confirmed cases, recovered cases, and any individuals who have died as a result of COVID-19.

```
countries = ['Brazil', 'Germany', 'United Kingdom', 'US', 'Italy', 'China']
df_regional = df_global[df_global['Country'].isin(countries)]

# Cre
df_re                                    l[['Confirmed', 'Recovered', 'Deaths']].sum(axis=1)
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

```
ɜf>:5: SettingWithCopyWarning:
ɔn a copy of a slice from a DataFrame.
ɔl_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df_regional['Cases'] = df_regional[['Confirmed', 'Recovered', 'Deaths']].sum(axis=1)
```

```
df_regional
```

	Date	Country	Confirmed	Recovered	Deaths	Cases
19584	2020-01-22	Brazil	0	0	0	0
19585	2020-01-23	Brazil	0	0	0	0
19586	2020-01-24	Brazil	0	0	0	0
19587	2020-01-25	Brazil	0	0	0	0
19588	2020-01-26	Brazil	0	0	0	0
...	...	...	...	...	...	...
153403	2022-04-12	United Kingdom	21846115	0	171004	22017119
153404	2022-04-13	United Kingdom	21883579	0	171662	22055241
153405	2022-04-14	United Kingdom	21916961	0	172014	22088975
153406	2022-04-15	United Kingdom	21916961	0	172014	22088975
153407	2022-04-16	United Kingdom	21916961	0	172014	22088975

4896 rows x 6 columns

Next steps: Generate code with df_regional | View recommended plots | New interactive sheet

Now that we have our data stored within a dataframe, let's prepare another dataframe that will hold our data in crosstabs, which will allow us to more easily visualize the data. we pivot our dataframe df_regional, creating columns out of countries, with the number of cases as the data fields. This new dataframe is called covid. We then set the index of the dataframe to be the date and assign the country names to column headers.

```
#  Restructuring our Data
df_regional = df_regional.pivot(index='Date', columns='Country', values='Cases')
countries = list(df_regional.columns)
print(countries)
df_regional
```

```
['Brazil', 'China', 'Germany', 'Italy', 'US', 'United Kingdom']
```

Country	Brazil	China	Germany	Italy	US	United Kingdom
Date						
2020-01-22	0	593	0	0	1	0
2020-01-23	0	691	0	0	1	0
Run cell (Ctrl+Enter) cell executed since last change		2	0	0	2	0
executed by Bhagya rekha Ammisetty 11:35 PM (0 minutes ago) executed in 0.353s		7	0	0	2	0
		0	0	0	5	0
...	...	...	...	...	...	...
2022-04-12	30846027	1669001	23149457	15565841	81464184	22017119
2022-04-13	30872838	1695023	23315135	15628582	81506332	22055241
2022-04-14	30896067	1718871	23472211	15694348	81561653	22088975
2022-04-15	30909456	1772835	23509808	15756771	81601239	22088975
2022-04-16	30912262	1773959	23549605	15821437	81613729	22088975

816 rows x 6 columns

Next steps: Generate code with df_regional | View recommended plots | New interactive sheet

```
covid = df_regional.reset_index('Date')
covid.set_index(['Date'], inplace=True)
covid.columns = countries
covid
```

	Brazil	China	Germany	Italy	US	United Kingdom
Date						
2020-01-22	0	593	0	0	1	0
2020-01-23	0	691	0	0	1	0
2020-01-24	0	982	0	0	2	0
2020-01-25	0	1487	0	0	2	0
2020-01-26	0	2180	0	0	5	0
...	...	...	...	...	...	...
2022-04-12	30846027	1669001	23149457	15565841	81464184	22017119
2022-04-13	30872838	1695023	23315135	15628582	81506332	22055241
2022-04-14	30896067	1718871	23472211	15694348	81561653	22088975
2022-04-15	30909456	1772835	23509808	15756771	81601239	22088975
2022-04-16	30912262	1773959	23549605	15821437	81613729	22088975

816 rows x 6 columns

Next steps: Generate code with covid | View recommended plots | New interactive sheet

```
covid.shape[1]
```

```
6
```

lets use some basic matplotlib for visualization

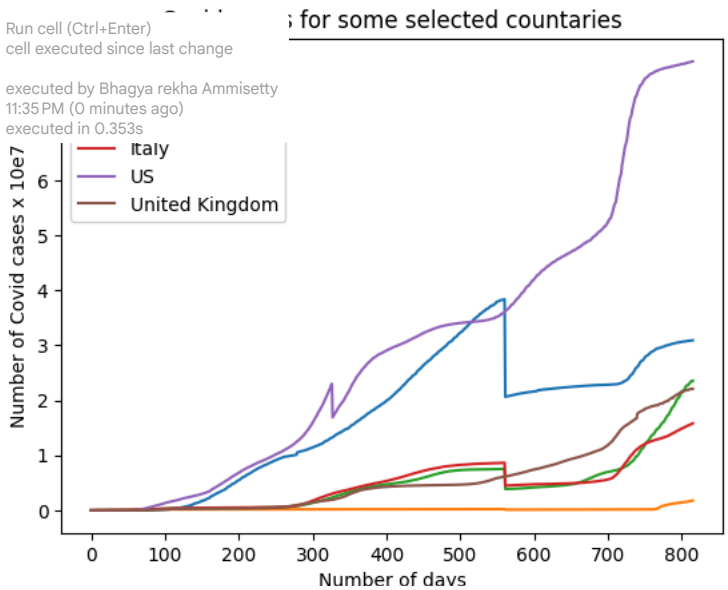
```
# get columns to plot
columns = covid.columns
# create x data
x_data = range(0, covid.shape[0])
# create figure and axis
fig, ax = plt.subplots()
# plot each column
for column in columns:
    ax.plot(x_data, covid[column], label=column)
```

```
# set title and legend
ax.set_title('Covid cases for some selected countaries')
ax.set_xlabel('Number of days')
ax.set_ylabel('Number of Covid cases x 10e7')
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7e3946237fd0>
```

Run cell (Ctrl+Enter)
cell executed since last change

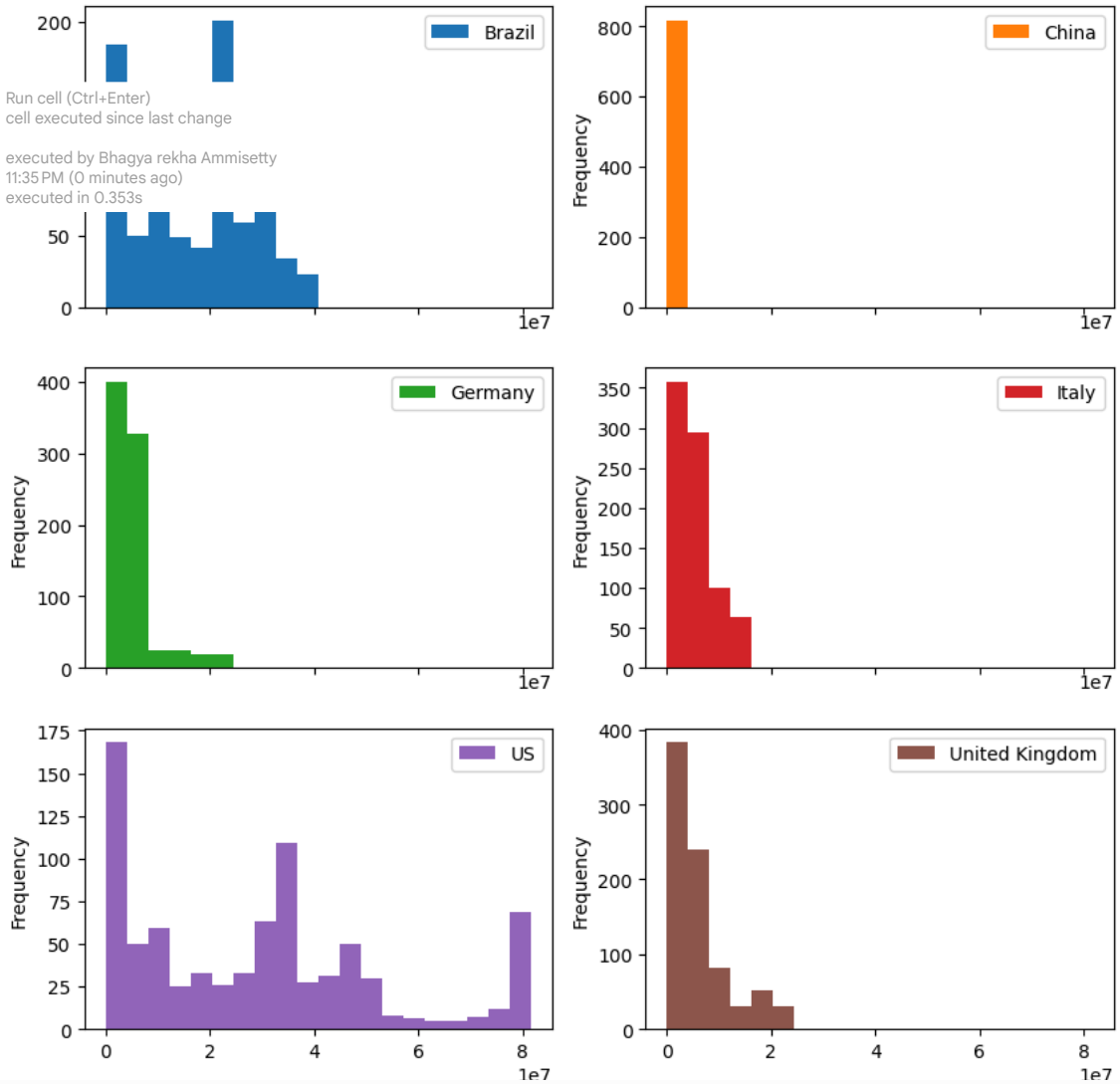
executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s



we can also use pandas plot.hist function to plot histograms

```
covid.plot.hist(subplots=True, layout=(3,2), figsize=(10, 10), bins=20)
```

```
array([[<Axes: ylabel='Frequency'>, <Axes: ylabel='Frequency'>],
       [<Axes: ylabel='Frequency'>, <Axes: ylabel='Frequency'>],
       [<Axes: ylabel='Frequency'>, <Axes: ylabel='Frequency'>]],
      dtype=object)
```



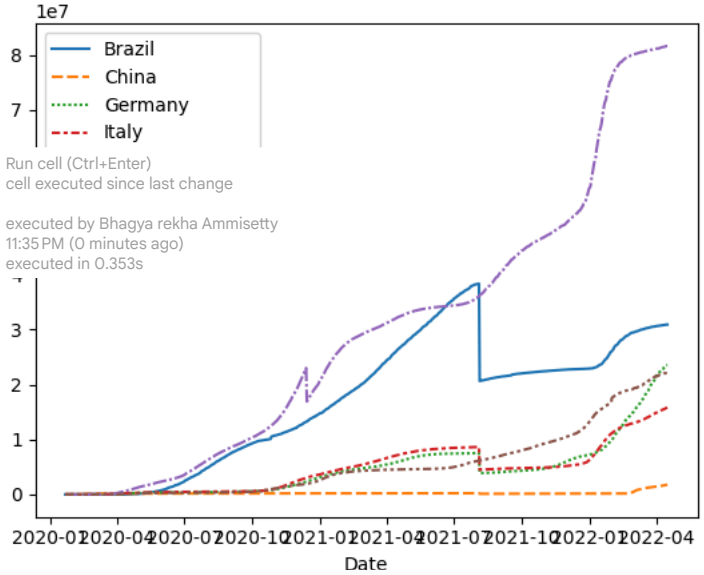
Run cell (Ctrl+Enter)
cell executed since last change

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

We can use seaborn lineplot to creating a viuslaization for this data

```
sns.lineplot(data=covid)
```

```
<Axes: xlabel='Date'>
```



Run cell (Ctrl+Enter)
cell executed since last change

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

One of the very important visualization is correlation matrix. Below is the seaborn heatmap that shows correlation matrix

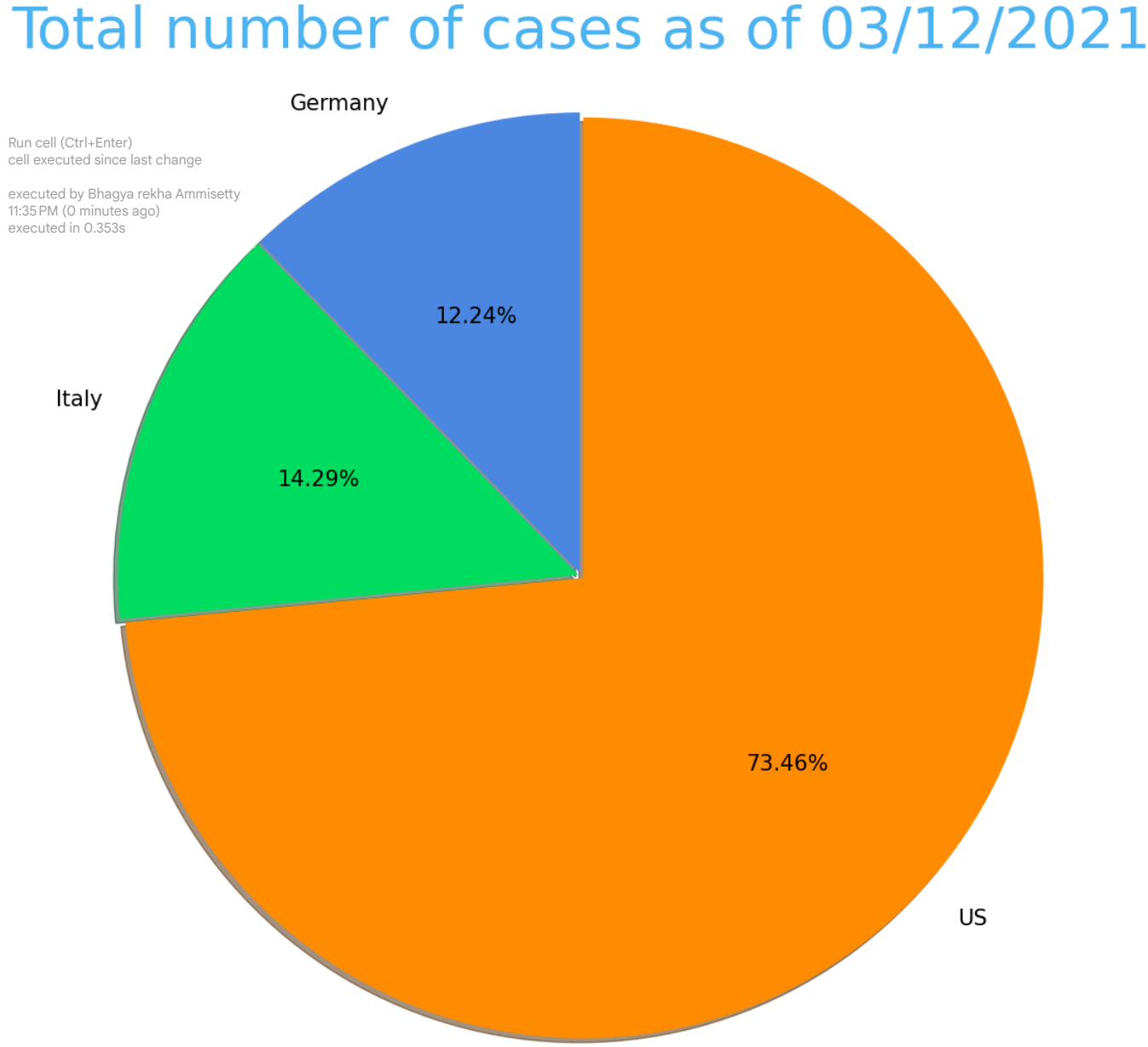
Pie Chart:

We'll be plotting the cases Pie Chart to understand the how many cases are in Germany, Italy and US as of 3/11/2021. So we have created list slices based on which our Pie Chart will be divided and the corresponding activities are it's values (in this cases countaries and the number of cases).

To plot a Pie Chart we call '.pie' function which takes x values which is 'slices' over here based on it the pie is divided followed by labels which have the corresponding string the values it represents. These string values can be altered by 'textprops'. To change the radius or size of Pie we call 'radius'. For the aesthetics we call 'shadow' as True and 'startangle '= 90. We can define colors to assign by passing a list of corresponding colors. To space out each piece of Pie we can pass on the list of corresponding values to 'explode'. The 'autopct' defines the number of positions that are allowed to be shown. In this case, autopct allows 2 positions before and after the decimal place

```
slices = [4969030, 5800684,   29815728]
activities = ['Germany', 'Italy', 'US']

cols=['#4C8BE2','#00e061','#FF8C00']
exp = [0.02,0.02,0.02]

plt.pie(slices,labels=activities,
        textprops=dict(size=15,color='black'),
        radius=3,
        colors=cols,
        autopct='%2.2f%%',
        explode=exp,
        shadow=True,
        startangle=90)

plt.title('Total number of cases as of 03/12/2021\n\n\n\n\n',color='#4fb4f2',size=40)
```

```
Text(0.5, 1.0, 'Total number of cases as of 03/12/2021\n\n\n\n')
```



In Section A, we created a dictionary that contains hex values for different countries. Storing this in a dictionary will allow us to easily call it later in a for-loop. We also assign the FiveThirtyEight style to add some general formatting, which we'll heavily build upon.

In Section B, we create our first visualization using Pandas' plot function. We use the colors parameter to assign the colors to different columns. We also use the set_major_formatter method to format values with separators for thousands.

In Section C, we create a for-loop that generates label text for the various countries. This for-loop gets each country's name from the keys in the dictionary in the form of a list and iterates over this list. It places text containing the country's name to the right of the last x-value (covid.index[-1] → the last date in the dataframe), at the current day's y-value (which will always be equal to the max value of that column).

Finally, in Section D, we add a title, subtitle, and source information about the chart. We use variables again to position the data so as the graph updates these positions are updated dynamically!

```
# Section A - Generating Colours and Style
colors = {'Brazil':'#045275', 'China':'#089099', 'Italy':'#7CCBA2', 'Germany':'#FCDE9C', 'US':'#DC3977', 'United Kingdom':'#7C1D6F'}
plt.style.use('fivethirtyeight')

# Section B - Creating the Visualization
plot = covid.plot(figsize=(12,8), color=list(colors.values()), linewidth=5, legend=False)
```

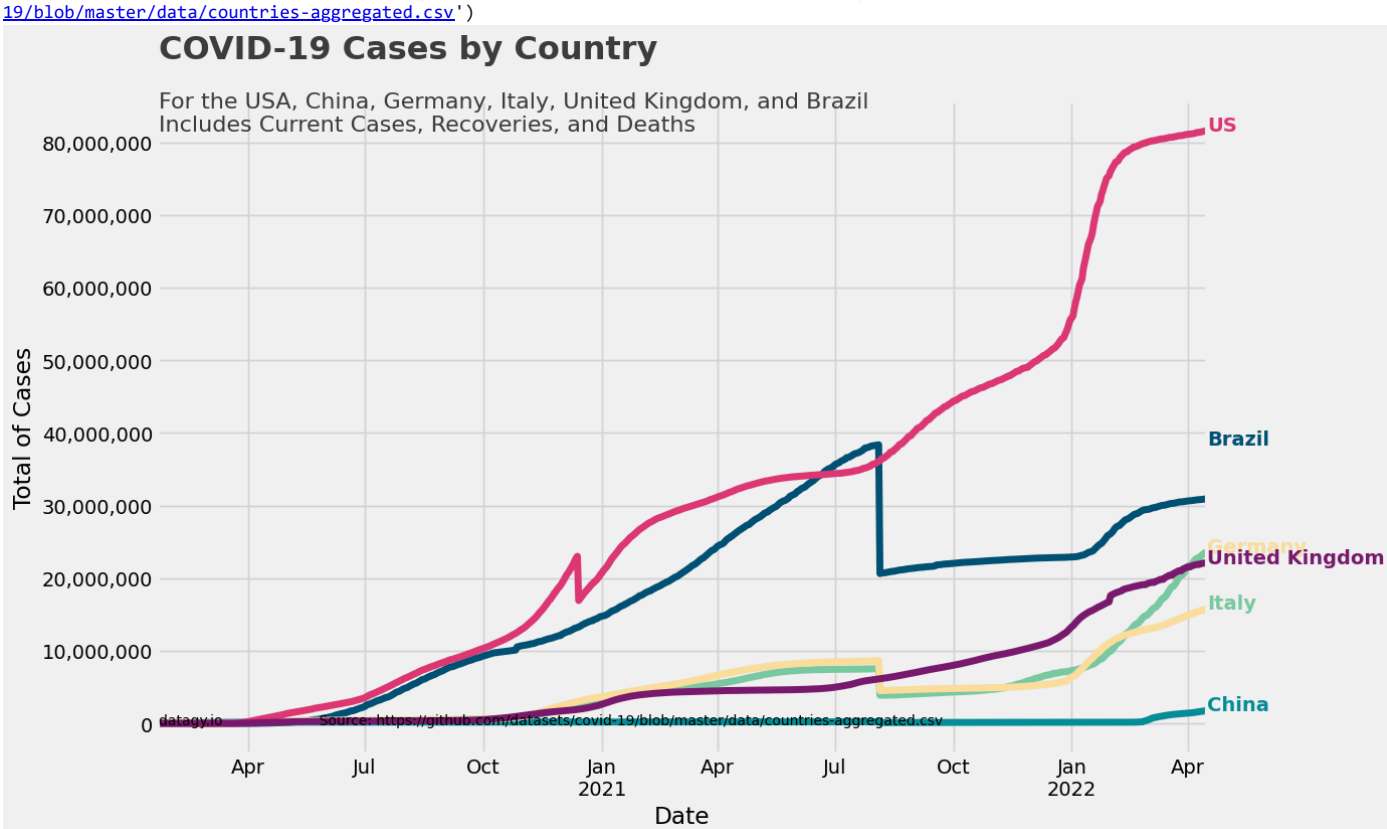
```
plot.yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))
plot.grid(color='#d4d4d4')
plot.set_xlabel('Date')
plot.set_ylabel('Total of Cases')

# Section C - Assigning Colour
for country in list(colors.keys()):
    p                                                    , y = covid[country].max(), color = colors[country], s = country, weight = 'bold')
```

Run cell (Ctrl+Enter)
cell executed since last change

```
# Sec
plot.                                                int(covid.max().max())+45000, s = "COVID-19 Cases by Country\n\n", fontsize = 23, weight = 'bold', alpha =
plot.                                                int(covid.max().max())+15000, s = "For the USA, China, Germany, Italy, United Kingdom, and Brazil\nIncludes
plot.text(x = covid.index[1], y = -100000,s = 'datagy.io                    Source: https://github.com/datasets/covid-19/blob/master/data/c
```

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

```
Text(2020-01-23 00:00:00, -100000, 'datagy.io                    Source: https://github.com/datasets/covid-19/blob/master/data/countries-aggregated.csv')
```



Here I am Making the data graphics interactive.

Creating the interactive pie chart for COVID-19 cases distribution

```
import pandas as pd
import numpy as np
import plotly.express as px

# List of countries
countries = ['Brazil', 'Germany', 'United Kingdom', 'US', 'Italy', 'China']

# Filtered the DataFrame
df_regional = df_global[df_global['Country'].isin(countries)]

# Finding the sum of 'Confirmed', 'Recovered', and 'Deaths'
df_regional['Total Cases'] = df_regional[['Confirmed', 'Recovered', 'Deaths']].sum(axis=1)

# Summarizing total cases for each country
total_cases_by_country = df_regional.groupby('Country', as_index=False)['Total Cases'].sum()
```

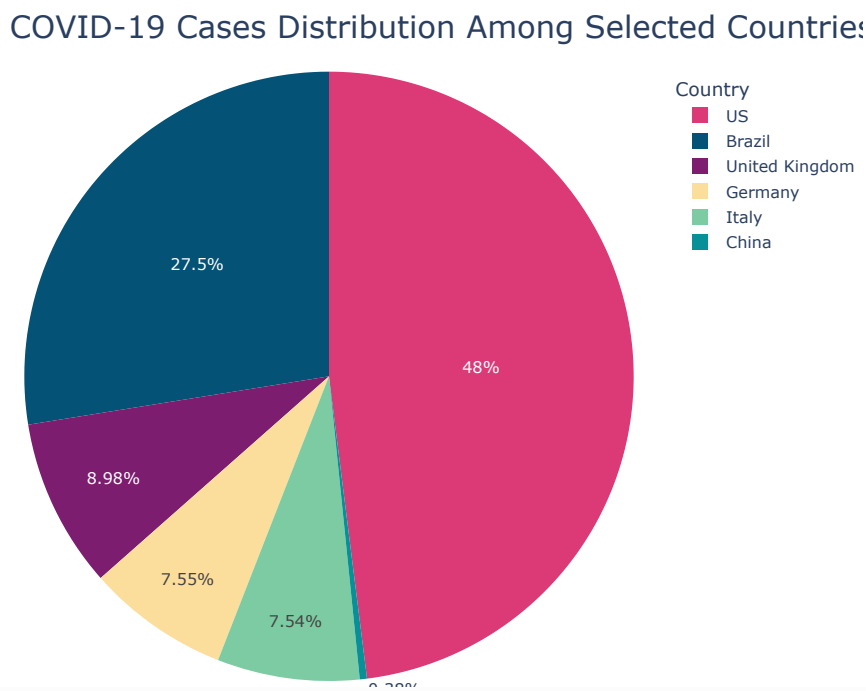
```
# Creating the interactive pie chart
fig = px.pie(
    total_cases_by_country,
    values='Total Cases',
    names='Country',
    title='COVID-19 Cases Distribution Among Selected Countries',
    color='Country'
    c
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

```
                    :',
        'Italy': '#7CCBA2',
        'China': '#089099'
    },
    labels={'Total Cases': 'Total Cases'}
)
fig.update_layout(
    title_font_size=23,
    legend_title='Country',
    margin=dict(l=0, r=0, t=50, b=0)
)
#Display the pie chart
fig.show()
```

```
<ipython-input-30-86921fab2b06>:12: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
```

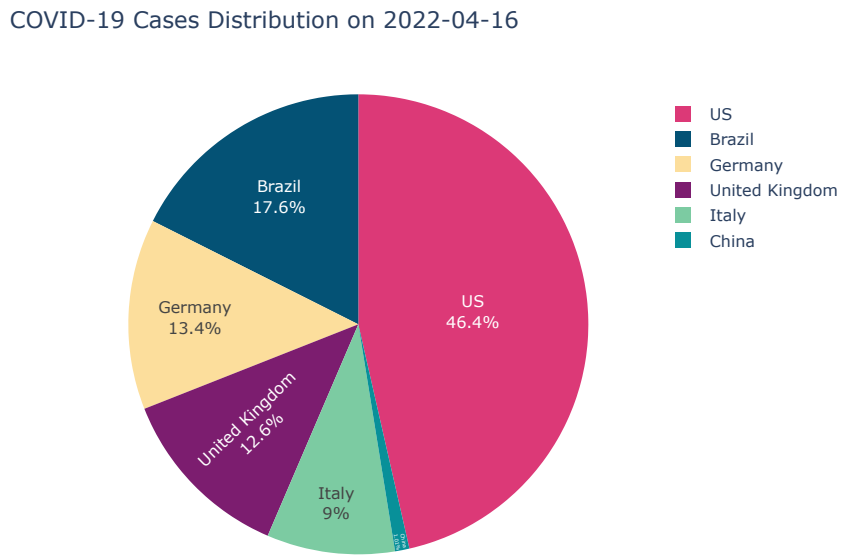



```
import plotly.express as px

#Customized my pie chart
color_map = {
    'Brazil': '#045275',
    'Germany': '#FCDE9C',
    'United Kingdom': '#7C1D6F',
    'US': '#DC3977',
    'Italy': '#7CCBA2',
    'China': '#089099'
}

latest_data = covid.iloc[-1]
```

```
# Creating the interactive pie chart with customized colors
fig = px.pie(
    names=latest_data.index,
    values=latest_data.values,
    title=f'COVID-19 Cases Distribution on {latest_data.name.date()}',  # Displaying the date of the latest data
    labels={'value': 'Cases', 'index': 'Country'},
    color=latest_data.index,  # Set color by country
    c                                  # Apply custom color map
)

# Cus                                  interactivity
fig.u
    textposition='inside',
    textinfo='percent+label'
)
#Displaying the piechart
fig.show()
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s



Creating the interactive line chart for COVID-19 cases distribution

```
import plotly.express as px

# List of countries
countries = ['Brazil', 'Germany', 'United Kingdom', 'US', 'Italy', 'China']
color_map = {
    'Brazil': '#045275',
    'Germany': '#FCDE9C',
    'United Kingdom': '#7C1D6F',
    'US': '#DC3977',
    'Italy': '#7CCBA2',
    'China': '#089099'
}

# creating the interactive bar chart by resetting the index
fig = px.bar(
    covid.reset_index(),
    x="Date",
    y=countries,  # Directly use the list of countries as y-axis data
    title="COVID-19 Cases Over Time for Selected Countries",
    labels={"value": "Number of Cases", "Date": "Date"},
    height=600,
    color_discrete_map=color_map  # Apply custom color map
)

# Adding range slider
```

```
fig.update_layout(
    xaxis_rangeslider_visible=True,
    xaxis_title='Date',
    yaxis_title='Number of Cases'
)

# Display interactive bar chart
fig.show()
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by Bhagya rekha Ammisetty
11:35 PM (0 minutes ago)
executed in 0.353s

```
st-packages/_plotly_utils/basevalidators.py:105: FutureWarning:

erties.to_pydatetime is deprecated, in a future version this will return a Series containing python datetime
```

COVID-19 Cases Over Time for Selected Countries