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) Data visualization using matplot lib, pandas and seaborn (pie chart, line chart, heatmaps etc).

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

```
DataFrame: df_states
Generate
View
DataFrame with shape (61942, 5)
```

```
df = pd.read_csv('https://raw.githubusercontent.cc
df1 = pd.read_csv('https://raw.githubusercontent.c

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

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

```
df
```

KeyError KeyError

Please explain the error:

```
KeyError: "The following id_vars
```

Hypothesis

The error message `KeyError: "The following id_vars or value_vars are not present in the DataFrame: ['level_0']"` arises because the DataFrame `covid` does not have a column named 'level_0' when you try to melt it using `pd.melt`. This happens because when you reset the index of a DataFrame using `reset_index()` without setting the `drop` parameter to `True`, it inserts the original index as a column named 'index' instead of 'level_0'.

Suggested Changes

```
ie to long format for easy plotti
.reset_index().melt(id_vars='ind
columns={'index': 'Days'}, inpla
 Plot
(



',
9 Cases Over Time by Country (Ar
: 'Number of Days', 'Cases': 'Nu


eadability
yout( # Changed fig to fig_area
Country',
nified'



Changed fig to fig_area for cons
```

Use code with caution

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

11592 rows × 18 columns

```
DataFrame: df_states
```

Generate

View

```
DataFrame with shape (61942, 5)
```

▶ Sources

Rate this answer

Next steps: Generate code with `df` View recommended plots New i

```
df1
```

	date	confirmed	confirmed_cum	deaths	deaths_c
0	2020-01-25	4	4	0	
1	2020-01-26	0	4	0	
2	2020-01-27	1	5	0	
3	2020-01-28	0	5	0	
4	2020-01-29	4	9	0	
...	...	...	...	...	
1283	2023-07-31	0	11731441	0	199
1284	2023-08-01	0	11731441	0	199
1285	2023-08-02	0	11731441	0	199
1286	2023-08-03	0	11731441	0	199
1287	2023-08-04	5082	11736523	0	199

```
DataFrame: df_states
```

Generate

View

```
DataFrame with shape (61942, 5)
```

1288 rows × 19 columns

Next steps: Generate code with `df1` View recommended plots New

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

```
df_filtered=df1.drop(['confirmed','deaths','tests'
df_filtered
```

	date	confirmed_cum	deaths_cum	tests_cum	posi
0	2020-01-25	4	0	0.0	
1	2020-01-26	4	0	0.0	
2	2020-01-27	5	0	0.0	
3	2020-01-28	5	0	0.0	
4	2020-01-29	9	0	0.0	
...	...	...	...	...	
1283	2023-07-31	11731441	19999	0.0	
1284	2023-08-01	11731441	19999	0.0	
1285	2023-08-02	11731441	19999	0.0	
1286	2023-08-03	11731441	19999	0.0	
1287	2023-08-04	11736523	19999	NaN	

```
DataFrame: df_states
```
Generate

View

```
DataFrame with shape (61942, 5)
```

1288 rows × 11 columns

Next steps: Generate code with `df_filtered` View recommended plot

** 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

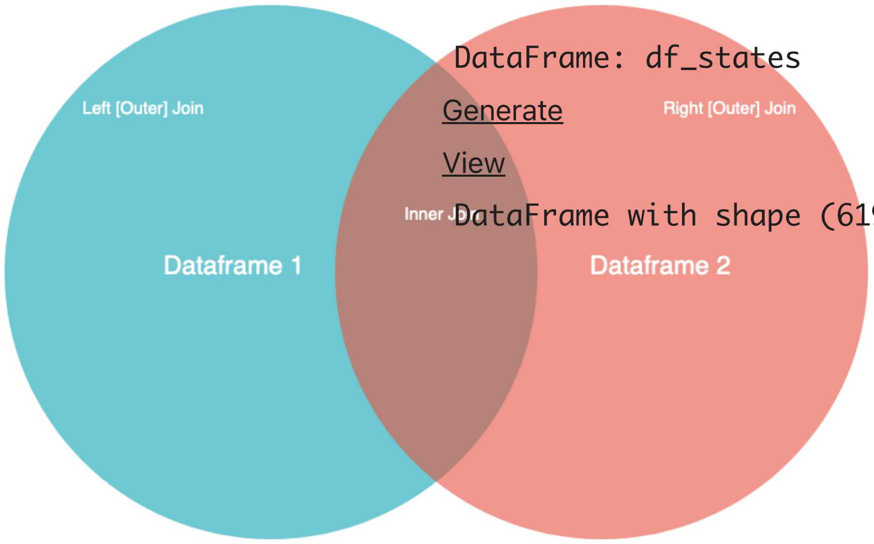
> left

> right

Outer Join

Here, you'll specify an outer join with the how parameter. Remember from the diagrams below that in an outer join (also known as a full outer join), all rows from both DataFrames will be present in the new DataFrame.

If a row doesn't have a match in the other 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.



```
DataFrame: df_states
Generate
View
DataFrame with shape (61942, 5)
```

Outer Join

```
outer_merged = pd.merge(df, df_filtered, how="outer
outer_merged.head()
```

	date	confirmed	deaths	tests	positives	recovere
0	2020-01-25	4	0	0.0	0.0	0.
1	2020-01-25	0	0	0.0	0.0	0.
2	2020-01-25	3	0	0.0	0.0	0.
3	2020-01-25	0	0	0.0	0.0	0.
4	2020-01-25	0	0	0.0	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

```
DataFrame: df_states
```

Generate

View

```
DataFrame with shape (61942, 5)
```

```
df1_a = pd.DataFrame(
       { "A": ["A0", "A1", "A2", "A3"],
            "B": ["B0", "B1", "B2", "B3"],
            "C": ["C0", "C1", "C2", "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

DataFrame: df_states

Next steps: Generate code with df1_a View recommended plots N

Generate

View

DataFrame with shape (61942, 5)

```
df2_a
```

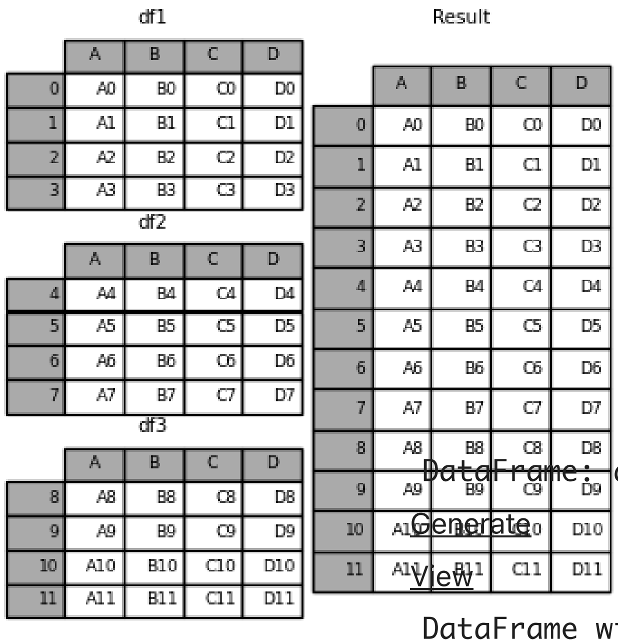
	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

Next steps: Generate code with df2_a View recommended plots N

```
df3_a
```

	A	B	C	D
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 df3_a View recommended plots N



DataFrame: df_states

Generate

View

DataFrame with shape (61942, 5)

```
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

```
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
3	X3

dtype: object

```
s2
```

	0
0	_0
1	_1
2	_2
3	_3

dtype: object

DataFrame: df_states
Generate
View
DataFrame with shape (61942, 5)

```
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

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.githubusercon
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

```
DataFrame: df_states
```

Generate View

```
DataFrame with shape (61942, 5)
```

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'
df_regional = df_global[df_global['Country'].isin(

# Creating a Summary Column
df_regional['Cases'] = df_regional[['Confirmed', '
```

```
<ipython-input-180-6b9eee60b56f>:5: SettingWithCopyWa


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

See the caveats in the documentation: https://pandas.
```

```
df_regional
```

	Date	Country	Confirmed	Recovered	Deaths	
19584	2020-01-22	Brazil	0	0	0	
19585	2020-01-23	Brazil	0	0	0	
19586	2020-01-24	Brazil	0	0	0	
19587	2020-01-25	Brazil	0	0	0	
19588	2020-01-26	Brazil	0	0	0	
...	...	...	...	...	...	
153403	2022-04-12	United Kingdom	21846115	0	171004	22
153404	2022-04-13	United Kingdom	21883579	0	171662	22

DataFrame: df_states

Generate

View

DataFrame with shape (61942, 5)

Next steps: Generate code with df_regional View recommended plot

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', colu
countries = list(df_regional.columns)
print(countries)
df_regional
```

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

Country	Brazil	China	Germany	Italy	US
Date					
2020-01-22	0	593	0	0	1
2020-01-23	0	691	0	0	1
2020-01-24	0	982	0	0	2
2020-01-25	0	1487	0	0	2
2020-01-26	0	2180	0	0	5
...	...	...	...	...	...
2022-04-12	30846027	1669001	23149457	15565841	81464184
2022-04-13	30872838	1695023	23315135	15628582	81506332

DataFrame: df_states
Generate
View
DataFrame with shape (61942, 5)

Next steps: Generate code with df_regional View recommended plot

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

Date	Brazil	China	Germany	Italy	US	U Ki
2020-01-22	0	593	0	0	1	
2020-01-23	0	691	0	0	1	
2020-01-24	0	982	0	0	2	
2020-01-25	0	1487	0	0	2	
2020-01-26	0	2180	0	0	5	
...	...	...	...	...	...	
2022-04-12	30846027	1669001	23149457	15565841	81464184	22
2022-04-13	30872838	1695023	23315135	15628582	81506332	22

DataFrame: df_states
Generate
View
DataFrame with shape (61942, 5)

Next steps: Generate code with covid | View recommended plots | N

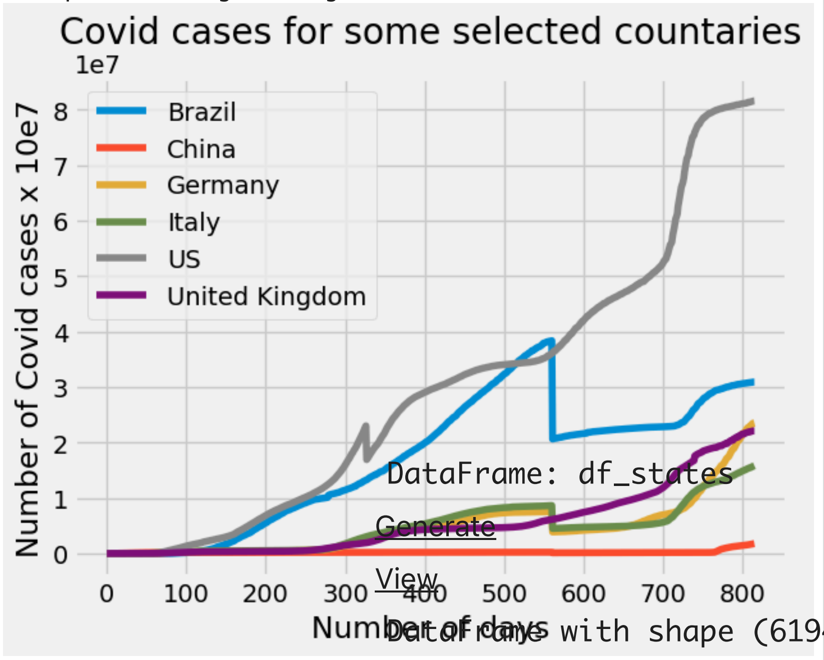
```
covid.shape[1]
```

```
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 counta
ax.set_xlabel('Number of days')
ax.set_ylabel('Number of Covid cases x 10e7')
ax.legend()
```

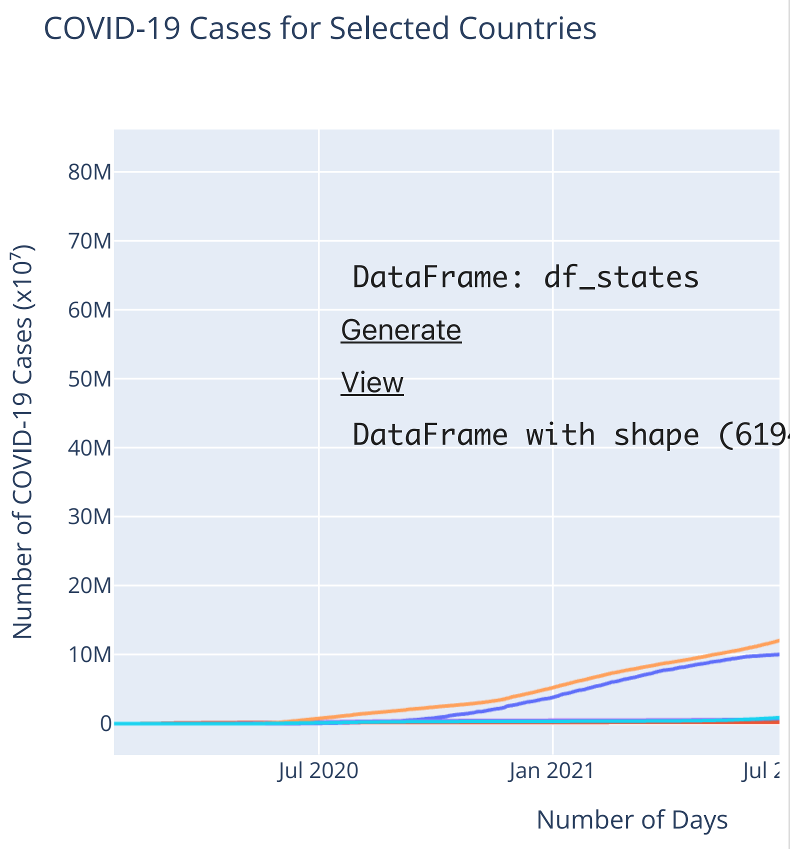
```
<matplotlib.legend.Legend at 0x786f33465310>
```



```
DataFrame with shape (61942, 5)
```

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

```
import plotly.express as px

# Directly reset and rename the index for clarity
covid_plot = covid.reset_index().rename(columns={c

# Convert to long format for easy plotting
covid_long = covid_plot.melt(id_vars='Days', var_n

# Generate interactive line plot
fig = px.line(
    covid_long,
    x='Days',
    y='Cases',
    color='Country',
    title='COVID-19 Cases for Selected Countries',
    labels={'Days': 'Number of Days', 'Cases': 'Nu
)

# Improve layout readability
fig.update_layout(
    legend_title='Country',
    hovermode='x unified'
)

# Display plot
fig.show()
```



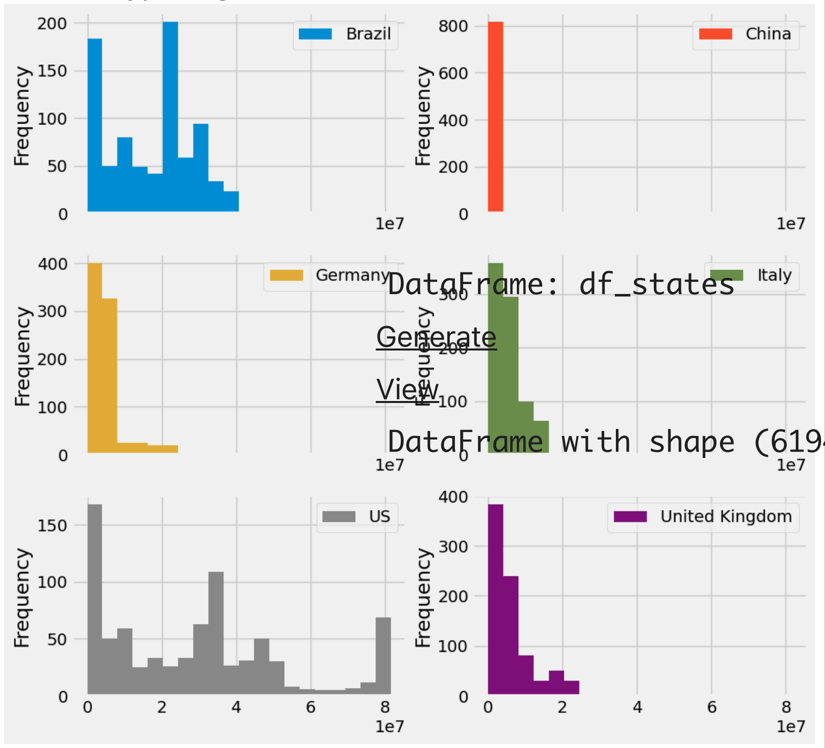
```
DataFrame: df_states
```

Generate

View

```
DataFrame with shape (61942, 5)
```

```
covid.plot.hist(subplots=True, layout=(3,2), figsi
array([[<Axes: ylabel='Frequency'>, <Axes:
ylabel='Frequency'>],
       [<Axes: ylabel='Frequency'>, <Axes:
ylabel='Frequency'>],
       [<Axes: ylabel='Frequency'>, <Axes:
ylabel='Frequency'>]],
      dtype=object)
```

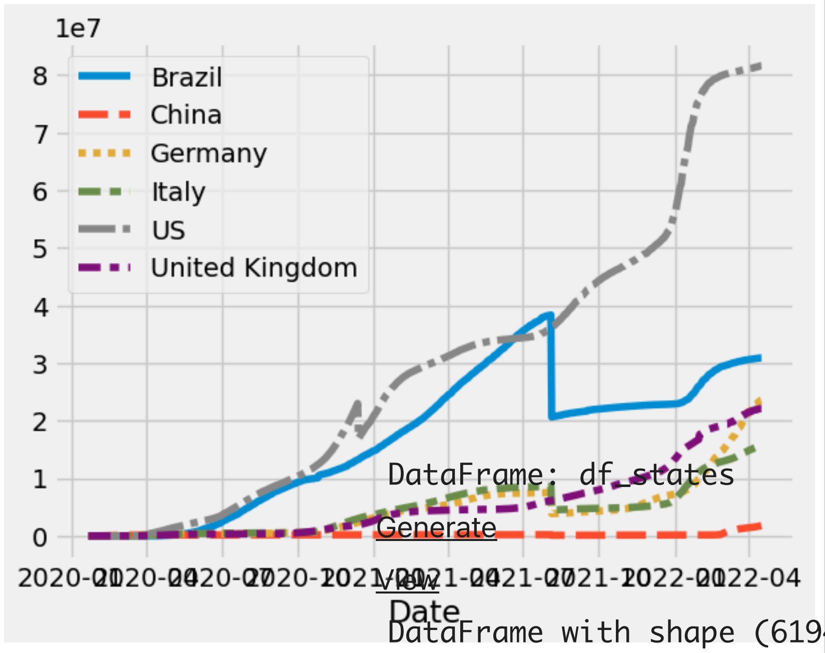


```
DataFrame: df_states
Generate
View
DataFrame with shape (61942, 5)
```

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

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

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



DataFrame: df_states

Generate

View

DataFrame with shape (61942, 5)

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\
```

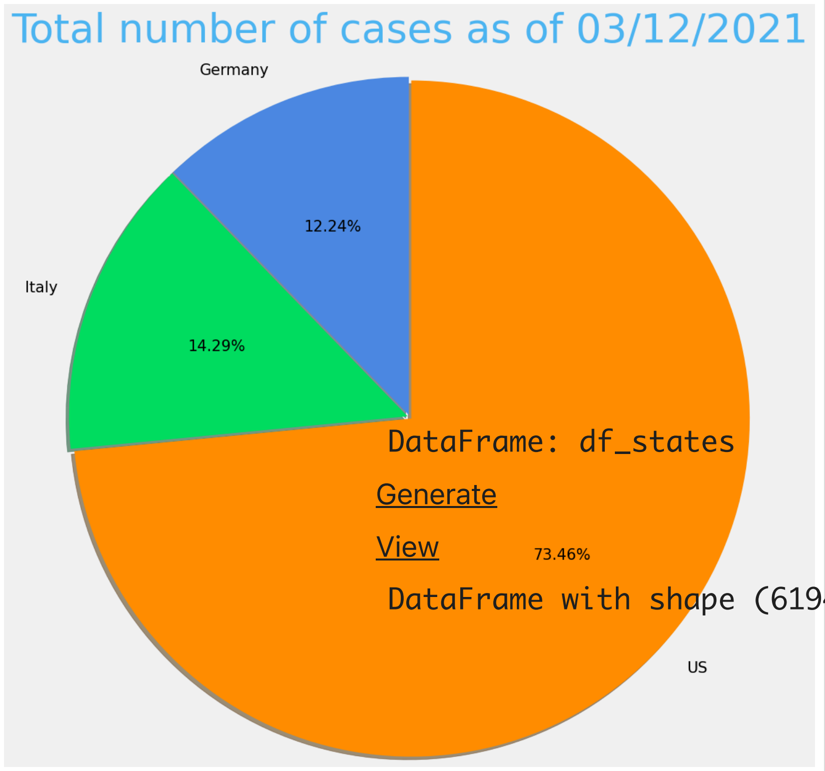
```
DataFrame: df_states
```

Generate

View

```
DataFrame with shape (61942, 5)
```

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



```
DataFrame: df_states
```

Generate

View

```
DataFrame with shape (61942, 5)
```

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!

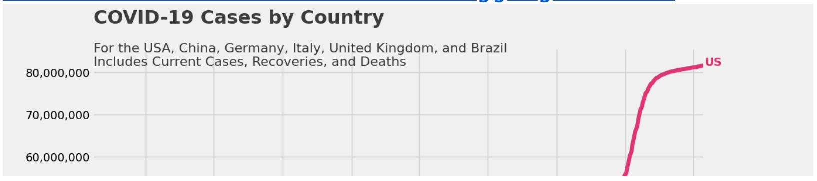
DataFrame: df_states

Generate

View

DataFrame with shape (61942, 5)

```
# Section A - Generating Colours and Style
colors = {'Brazil':'#045275', 'China':'#089099', '
plt.style.use('fivethirtyeight')

# Section B - Creating the Visualization
plot = covid.plot(figsize=(12,8), color=list(color
plot.yaxis.set_major_formatter(ticker.StrMethodFor
plot.grid(color='#d4d4d4')
plot.set_xlabel('Date')
plot.set_ylabel('Total of Cases')

# Section C - Assigning Colour
for country in list(colors.keys()):
    plot.text(x = covid.index[-1], y = covid[count

# Section D - Adding Labels
plot.text(x = covid.index[1], y = int(covid.max().
plot.text(x = covid.index[1], y = int(covid.max().
plot.text(x = covid.index[1], y = -100000,s = 'dat
Text(2020-01-23 00:00:00, -100000, 'datagy.io
Source: https://github.com/datasets/covid-
19/blob/master/data/countries-aggregated.csv')
```



```
fig, ax = plt.subplots()
for column in columns:
    ax.plot(x_data, covid[column], label=column)
ax.set_title('Covid cases for some selected countr
ax.set_xlabel('Number of days')
ax.set_ylabel('Number of Covid cases x 10e7')
ax.legend()
```

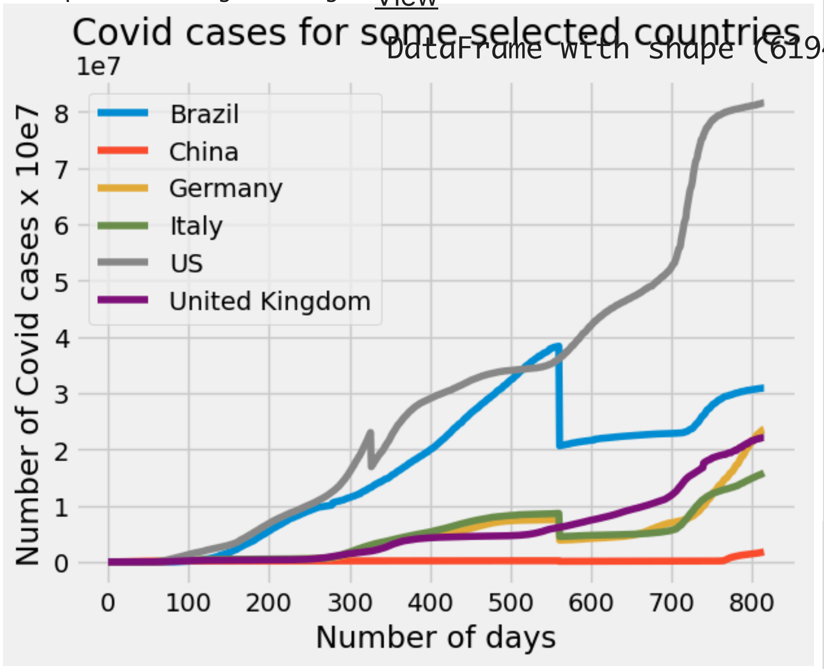
DataFrame: df_states

Generate

View

```
<matplotlib.legend.Legend at 0x786f3294d310>
```

DataFrame with shape (61942, 5)



```
slices = [4969930, 5800684, 29815728]
activities = ['Germany', 'Italy', 'US']
cols = ['#AC8BE2', '#00e061', '#FF8C00']
exp = [0.02, 0.02, 0.02]
plt.pie(slices, labels=activities, textprops=dict(
plt.title('Total number of cases as of 03/12/2021\
```

Enter a prompt here

0 / 2000

Responses may display inaccurate or offensive information that doesn't represent Google's views. Learn more