Moving on to data visualization using Python, let's take a look at the given boilerplate code on the left of the screen:

```python
# Import required libraries
import sys
import psycopg2
import plotly.express as px
import plotly.io as pio
import pandas as pd
```

Lines 1 to 6 imports all the required libraries to our code. Here are their usages:

1. **sys**: Configuration purposes
2. **psycopg2**: Connects to the database which has our data from the previous section
3. **plotly**: Used to create data visualisation
4. **pandas**: Used to store our data after retrieving from the database

```python
# Outer try
try:

except Exception as error:
  print(f"Error: {error}")
```

Lines 9 and 74-75 is a try-except block that will catch any thrown errors inside the try block from lines 10 to 72 (excluding the inner try block) and print them out for debugging purposes.

```python
  # Database connection parameters
  db_params = {
    'dbname': 'db',
    'user': 'postgres',
    'password': 'password',
    'host': 'db',
    'port': '5432'
  }

  # Initialize an empty DataFrame
  df = pd.DataFrame()

  # Inner try
  # Any errors from this try will be caught and system exits right away
  try:
    # Establish a connection to the PostgreSQL database
    connection = psycopg2.connect(**db_params)
    cursor = connection.cursor()

    # Get all the data that you have loaded in the ETL step
    cursor.execute("SELECT * FROM healthcare;")

    # Fetch the result of the query
    data = cursor.fetchall()

    # Get column names
    colnames = [desc[0] for desc in cursor.description]

    # Store data in a DataFrame
    df = pd.DataFrame(data, columns=colnames)

    # Close the cursor and connection
    cursor.close()
    connection.close()
  except psycopg2.DatabaseError as error:
    print(f"PostgreSQL database errors from the inner try: {error}")
    sys.exit(0)
  except Exception as error:
    print(f"Generic errors from the inner try: {error}")
    sys.exit(0)
```

Lines 13 to 45 establish the connection to the database containing the data that we applied the ETL process to in the previous section.

After establishing a connection, we will get all the rows from the table with our data and store them into a pandas DataFrame.

```
53  # TODO1: Get only the singapore data
54  # (Technically, all the data in this df is for Singapore)
55  df_sg =
56
57  # TODO2: Group by 'disease' and sum the 'no_of_cases'
58  df_grouped =
59
60  # TODO3: Sort the aggregated data by 'no_of_cases' in descending order
61  df_sorted =
62
63  # TODO4: Get the top 10 diseases
64  df_top10 =
65
66  # TODO5: Create the bar plot
67  bar_fig =
68
69  # Print the html for this bar plot to display it on the right side of this page
70  bar_plot_html = pio.to_html(bar_fig, full_html=True, include_plotlyjs=True)
71  print(bar_plot_html)
```

We will focus on lines 53 to 67 for this section of the course, where we will create a bar plot to show the top infectious diseases in Singapore.

The following is the metadata of the columns in the **healthcare** table:

	Column name	Datatype of value in column
1	disease	String
2	_year	Integer
3	no_of_cases	Integer
4	country	String

Now that we understand what the boilerplate code does and the metadata of the columns in the **healthcare** table, let's start coding!

Note: Do not copy the code from the document to the code editor.

TODO1:

At line 55, we will have all our data stored in **df**. However, this data might not only be for 'Singapore' but also for other countries (depending on what country's data was fed into the pipeline previously). Since we are interested in Singapore dataset, lets extract out using this code:

df_sg = df.query('country=="singapore"')

DataFrame.query will query the column (in this case **"country"**) and check if the value is **"singapore"**. It will then return all the rows whereby its column value is **"singapore"**.

TODO2:

Since we want to know the total number of cases for each disease, we need to group all the data based on the column **"disease"** and then for each disease, sum their **"no_of_cases"**.

df_grouped = df_sg.groupby('disease', as_index=False)['no_of_cases'].sum()

DataFrame.groupby will group the data by the column **"disease"** and keeping this column as a regular column instead of index.

TODO3:

Since we want to plot the graph such that we can see which are the top infectious disease, we need to sort it in descending order, so that the diseases with more cases will be on the left of the graph.

df_sorted = df_grouped.sort_values(by='no_of_cases', ascending=False)

DataFrame.sort_values will sort the data in the DataFrame according to the values in the column **"no_of_cases"** in descending order.

TODO4:

Since there are many different types of infectious diseases and we don't want our graph to be cluttered, we will just get the top 10 infectious diseases and plot them onto our graph later on.

df_top10 = df_sorted.head(10)

DataFrame.head returns the first 10 rows for **df_sorted** based on position.

TODO5:

Now that our DataFrame (**df_top10**) is ready, let's create a bar plot for it.

bar_fig = px.bar(df_top10, x="disease", y="no_of_cases", title="Top 10 Diseases in Singapore")

px.bar returns a bar figure for DataFrame **df_top10**, where the x-axis of the bar graph is column **"disease"** and y-axis is column **"no_of_cases"**. To make the graph more intuitive, we will add a title too.

Now that you have completed your code, go ahead and execute using the button that says **'EXECUTE'** right above the code editor and after a few seconds, you should see the output (bar graph) at the bottom right of this page.

Good job on completing this section! You may proceed to the next section.