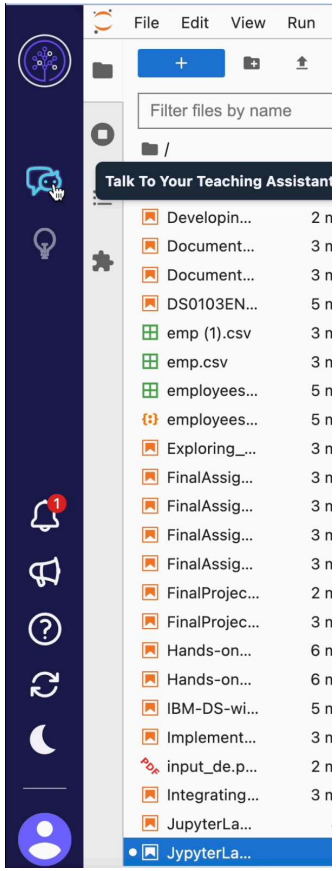
Test Environment for Generative AI classroom labs

This lab provides a test environment for the codes generated using the Generative AI classroom.

Follow the instructions below to set up this environment for further use.

Please note, at any point in time, to use the Generative AI tool, click on `TAI` the teaching assistant in the laft bar.



Setup

Install required libraries

In case of a requirement of installing certain python libraries for use in your task, you may do so as shown below.

In [1]:
```
%pip install seaborn
```

```
Requirement already satisfied: seaborn in /opt/conda/lib/python3.12/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /opt/conda/lib/python3.12/site-packages (from seaborn) (2.2.3)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.12/site-packages (from seaborn) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /opt/conda/lib/python3.12/site-packages (from seaborn) (3.1
0.1)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=
3.4->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>
=3.4->seaborn) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>
=3.4->seaborn) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=
3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->se
aborn) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=
3.4->seaborn) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.
1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-packages (from pandas>=1.2->seaborn) (2
024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-packages (from pandas>=1.2->seaborn)
(2025.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-packages (from python-dateutil>=2.7->matplo
tlib!=3.6.1,>=3.4->seaborn) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

Dataset URL from the GenAI lab

Use the URL provided in the GenAI lab in the cell below.

In [2]:
```
URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMSkillsNetwork-AI0272EN-SkillsNetwork/lab
```

Downloading the dataset

Execute the following code to download the dataset in to the interface.

In [3]:
```
import requests

# Fetch the CSV file
response = requests.get(URL)

# Check if the request was successful
if response.status_code == 200:

    # Write the response content to a local file
    with open("dataset.csv", 'wb') as f:
        f.write(response.content)
```

Test Environment

In [4]:
```
# Keep appending the code generated to this cell, or add more cells below this to execute in parts
import pandas as pd

# Function to read and display CSV file
def read_csv_file(file_path):
    try:
        # Read the CSV file
        df = pd.read_csv(file_path)

        # Print the first 5 rows
        print(df.head())
    except Exception as e:
        print(f"Error: {e}")

# Provide the file path
file_path=URL
read_csv_file(file_path)
```

```
       Country          Region  Happiness Rank  Happiness Score  \
0      Denmark  Western Europe               1            7.526
1  Switzerland  Western Europe               2            7.509
2      Iceland  Western Europe               3            7.501
3       Norway  Western Europe               4            7.498
4      Finland  Western Europe               5            7.413

   Lower Confidence Interval Upper Confidence Interval  \
0                      7.460                     7.592
1                      7.428                      7.59
2                      7.333                     7.669
3                      7.421                     7.575
4                      7.351                     7.475

  Economy (GDP per Capita)   Family Health (Life Expectancy)  Freedom  \
0                  1.44178  1.16374                  0.79504  0.57941
1                  1.52733  1.14524                  0.86303  0.58557
2                  1.42666  1.18326                  0.86733  0.56624
3                  1.57744  1.12690                  0.79579  0.59609
4                  1.40598  1.13464                  0.81091  0.57104

   Trust (Government Corruption)  Generosity  Dystopia Residual
0                        0.44453     0.36171            2.73939
1                        0.41203     0.28083            2.69463
2                        0.14975     0.47678            2.83137
3                        0.35776     0.37895            2.66465
4                        0.41004     0.25492            2.82596
```

In [28]:
```python
df1 = pd.read_csv(file_path)
# Display column data types
print("Column Data Types:\n", df1.dtypes)

# Check for potential incorrect types (e.g., numerical data stored as object)
for col in df1.columns:
    if df1[col].dtype == 'object':
        try:
            df1[col] = pd.to_numeric(df1[col])
            print(f"Column '{col}' was stored as object but contains numeric values.")
        except ValueError:
            print(f"Column '{col}' is correctly stored as object (non-numeric values detected).")
```

```
Column Data Types:
 Country                          object
Region                           object
Happiness Rank                    int64
Happiness Score                 float64
Lower Confidence Interval       float64
Upper Confidence Interval        object
Economy (GDP per Capita)         object
Family                          float64
Health (Life Expectancy)         object
Freedom                          object
Trust (Government Corruption)   float64
Generosity                      float64
Dystopia Residual               float64
dtype: object
Column 'Country' is correctly stored as object (non-numeric values detected).
Column 'Region' is correctly stored as object (non-numeric values detected).
Column 'Upper Confidence Interval' is correctly stored as object (non-numeric values detected).
Column 'Economy (GDP per Capita)' is correctly stored as object (non-numeric values detected).
Column 'Health (Life Expectancy)' is correctly stored as object (non-numeric values detected).
Column 'Freedom' is correctly stored as object (non-numeric values detected).
```

In [29]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 157 entries, 0 to 156
Data columns (total 13 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Country                        157 non-null    object
 1   Region                         157 non-null    object
 2   Happiness Rank                 157 non-null    int64
 3   Happiness Score                157 non-null    float64
 4   Lower Confidence Interval      153 non-null    float64
 5   Upper Confidence Interval      155 non-null    object
 6   Economy (GDP per Capita)       156 non-null    object
 7   Family                         157 non-null    float64
 8   Health (Life Expectancy)       155 non-null    object
 9   Freedom                        157 non-null    object
 10  Trust (Government Corruption)  157 non-null    float64
 11  Generosity                     157 non-null    float64
 12  Dystopia Residual              157 non-null    float64
dtypes: float64(6), int64(1), object(6)
memory usage: 16.1+ KB
```

In [30]:
```python
# Identify columns with missing values
missing_columns = df1.columns[df1.isnull().any()].tolist()
if missing_columns:
    print("Columns with missing values:", missing_columns)
    print("\nMissing Values Count:\n", df1[missing_columns].isnull().sum())
else:
    print("No missing values found in the dataset.")
```

```
Columns with missing values: ['Lower Confidence Interval', 'Upper Confidence Interval', 'Economy (GDP per Capita)',
'Health (Life Expectancy)']

Missing Values Count:
 Lower Confidence Interval    4
Upper Confidence Interval    2
Economy (GDP per Capita)     1
Health (Life Expectancy)     2
dtype: int64
```

In [15]:
```python
import pandas as pd
import numpy as np
```

```
# Function to clean the DataFrame
def clean_dataframe(df, column_to_strip):
    try:
        # 1. Remove leading and trailing whitespaces from values in a column
        #df[column_to_strip] = df[column_to_strip].str.strip()

        # 2. Replace empty strings with NaN values
        df.replace("", np.nan, inplace=True)

        # 3. Convert columns to appropriate data types using pandas' convert_dtypes()
        df = df.convert_dtypes()

        # Display the cleaned DataFrame
        print("Cleaned DataFrame:\n", df.head())
        print("\nUpdated Column Data Types:\n", df.dtypes)

        return df

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


# Loop over each column in the missing_columns list and call clean_dataframe with df and the column name
for col in missing_columns:
    df = clean_dataframe(df, col)  # Pass the DataFrame (df) and column name to the function
```

```
Error: 'NoneType' object has no attribute 'replace'
Error: 'NoneType' object has no attribute 'replace'
Error: 'NoneType' object has no attribute 'replace'
Error: 'NoneType' object has no attribute 'replace'
```

In [31]:
```
df1.head()
```

Out[31]:

	Country	Region	Happiness Rank	Happiness Score	Lower Confidence Interval	Upper Confidence Interval	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trus (Governmen Corruption
0	Denmark	Western Europe	1	7.526	7.460	7.592	1.44178	1.16374	0.79504	0.57941	0.4445
1	Switzerland	Western Europe	2	7.509	7.428	7.59	1.52733	1.14524	0.86303	0.58557	0.4120
2	Iceland	Western Europe	3	7.501	7.333	7.669	1.42666	1.18326	0.86733	0.56624	0.1497
3	Norway	Western Europe	4	7.498	7.421	7.575	1.57744	1.12690	0.79579	0.59609	0.3577
4	Finland	Western Europe	5	7.413	7.351	7.475	1.40598	1.13464	0.81091	0.57104	0.4100

In [37]:
```
for col in missing_columns:
    if df1[col].dtype in ['float64', 'int64']:  # Only replace in numeric columns
        df1[col].replace('NaN',df1[col].mean(),inplace=True)  # Replace missing with mean
        print(f"Missing values in column '{col}' replaced with mean value ")
```

```
Missing values in column 'Lower Confidence Interval' replaced with mean value
/tmp/ipykernel_1151/2198304054.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series th
rough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or d
f[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df1[col].replace('NaN',df1[col].mean(),inplace=True)  # Replace missing with mean
```

In [42]:
```
import pandas as pd
import plotly.graph_objects as go
```

```
# Sort by GDP per capita (if necessary) and select the top 10 countries
df1_sorted = df1.sort_values(by="Economy (GDP per Capita)", ascending=False).head(10)

# Create the bar chart using Plotly
fig1 = go.Figure()

# Add GDP per capita bars
fig1.add_trace(go.Bar(
    x=df1_sorted['Country'],
    y=df1_sorted['Economy (GDP per Capita)'],
    name='GDP per capita',
    marker=dict(color='blue')
))

# Add Healthy Life Expectancy bars
fig1.add_trace(go.Bar(
    x=df1_sorted['Country'],
    y=df1_sorted['Health (Life Expectancy)'],
    name='Healthy Life Expectancy',
    marker=dict(color='green')
))

# Update layout to make the chart more readable
fig1.update_layout(
    title="Top 10 Countries by GDP per Capita and Healthy Life Expectancy",
    xaxis_title="Countries",
    yaxis_title="Value",
    barmode='group',  # Group bars together
    xaxis_tickangle=-45,  # Rotate country names for better readability
    template='plotly_dark'  # Use a dark theme for the plot
)

# Show the figure
fig1.show()
```

In [45]:
```
import pandas as pd
import plotly.figure_factory as ff

# Create sub-dataset with specified columns
sub_dataset = df1[['Economy (GDP per Capita)', 'Family', 'Health (Life Expectancy)',
                   'Freedom', 'Trust (Government Corruption)', 'Generosity', 'Happiness Score']]

# Step 1: Clean the data by converting to numeric and handling errors (non-numeric values)
# This will coerce any non-numeric values to NaN (Not a Number)
sub_dataset = sub_dataset.apply(pd.to_numeric, errors='coerce')
```

```
# Step 2: Fill missing values (NaN) if necessary
# You can fill NaN values with the column's mean (or another strategy such as median)
sub_dataset.fillna(sub_dataset.mean(), inplace=True)

# Step 3: Calculate the correlation matrix
correlation_matrix = sub_dataset.corr()

# Step 4: Create a heatmap using Plotly
fig2 = ff.create_annotated_heatmap(
    z=correlation_matrix.values,
    x=correlation_matrix.columns.tolist(),
    y=correlation_matrix.index.tolist(),
    colorscale='Viridis',  # Color scale for the heatmap
    showscale=True
)

# Step 5: Update layout for the figure
fig2.update_layout(
    title="Correlation Heatmap of Selected Attributes",
    xaxis_title="Attributes",
    yaxis_title="Attributes",
    width=800,  # Width of the heatmap
    height=600  # Height of the heatmap
)

# Show the figure
fig2.show()
```

In [46]:
```
import pandas as pd
import plotly.express as px

# Create a scatter plot using Plotly Express
fig3 = px.scatter(
```

```
    df1,
    x='Economy (GDP per Capita)',
    y='Happiness Score',
    color='Region',  # Color the points based on Region
    title="Happiness Score vs GDP per Capita",
    labels={"GDP per Capita": "GDP per Capita", "Happiness Score": "Happiness Score"},
    template='plotly_dark'  # Use a dark theme for the plot
)

# Show the figure
fig3.show()
```

In [47]:
```
import pandas as pd
import plotly.express as px

# Aggregate Happiness Score by Region
region_happiness = df1.groupby('Region')['Happiness Score'].mean().reset_index()

# Create a pie chart using Plotly Express
fig4 = px.pie(
    region_happiness,
    names='Region',
```

```
    values='Happiness Score',
    title="Happiness Score by Region",
    template='plotly_dark'  # Use a dark theme for the plot
)

# Show the figure
fig4.show()
```

In [50]:
```
import pandas as pd
import plotly.express as px
```

```
# Ensure the 'Economy (GDP per Capita)' column is numeric
df1['Economy (GDP per Capita)'] = pd.to_numeric(df1['Economy (GDP per Capita)'], errors='coerce')
df1['Health (Life Expectancy)'] = pd.to_numeric(df1['Health (Life Expectancy)'], errors='coerce')

# Fill NaN values in 'Economy (GDP per Capita)' with 0 or any other appropriate value
df1['Economy (GDP per Capita)'].fillna(0, inplace=True)

# Create a map using Plotly Express
fig5 = px.scatter_geo(
    df1,
    locations="Country",  # Country names will be used to determine the location
    size="Economy (GDP per Capita)",  # Size of the points will be based on GDP per Capita
    hover_name="Country",  # Hover label will show the country name
    hover_data=["Health (Life Expectancy)"],  # Tooltip will display Healthy Life Expectancy
    title="GDP per Capita and Healthy Life Expectancy by Country",
    projection="natural earth",  # Map projection style
    template='plotly_dark'  # Use a dark theme for the map
)

# Show the map
fig5.show()
```

```
/tmp/ipykernel_1151/3472203184.py:9: FutureWarning:

A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or d
f[col] = df[col].method(value) instead, to perform the operation inplace on the original object.
```

In [52]:
```
import plotly.io as pio
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Create a subplot with 2 rows and 2 columns.
# Specify the subplot types: "xy" for scatter plots and "domain" for pie charts
fig_dashboard = make_subplots(
    rows=2, cols=2,  # Adjust rows and columns if necessary
    subplot_titles=["Figure 1", "Figure 2", "Figure 3", "Figure 4"],
    horizontal_spacing=0.1, vertical_spacing=0.1,
```

```
    specs=[[{"type": "xy"}, {"type": "xy"}],  # First row: scatter/xy plots
           [{"type": "xy"}, {"type": "domain"}]]  # Second row: scatter/xy and pie chart
)

# Add each figure as a subplot
fig_dashboard.add_trace(fig1.data[0], row=1, col=1)  # Add fig1 to the first subplot
fig_dashboard.add_trace(fig2.data[0], row=1, col=2)  # Add fig2 to the second subplot
fig_dashboard.add_trace(fig3.data[0], row=2, col=1)  # Add fig3 to the third subplot
fig_dashboard.add_trace(fig4.data[0], row=2, col=2)  # Add fig4 (pie chart) to the fourth subplot

# Update layout for the dashboard
fig_dashboard.update_layout(
    title_text="Dashboard with Four Plotly Figures",
    template='plotly_dark',  # Optional: Choose your template
    showlegend=False
)

# Write the dashboard to an HTML file
pio.write_html(fig_dashboard, file='dashboard.html')

# Optionally, you can view the dashboard in a browser
fig_dashboard.show()
```

In [55]:
```
Narrative for World Happiness Report Dashboard
Introduction: The World Happiness Report dashboard provides a comprehensive visualization of various aspects influenc

1. Heatmap: Correlation Analysis
Our first visualization is a Heatmap showcasing the correlation between different variables such as GDP per capita, H

Interpretation: The heatmap reveals critical patterns such as a positive correlation between GDP per capita and Happi
2. Scatter Plot: GDP per Capita vs Happiness Score by Region
The scatter plot displays the relationship between GDP per capita and Happiness Score for different countries, segmer
```

```
Interpretation: This scatter plot reveals a clear trend where higher GDP per capita is associated with higher happine
3. Pie Chart: Happiness Score Distribution by Region
The pie chart provides a breakdown of Happiness Scores across different global regions. Each segment represents the p

Interpretation: This chart emphasizes the regional disparities in happiness. For example, countries in Northern Europ
4. Map: GDP per Capita and Healthy Life Expectancy by Country
Finally, the geographical map displays the GDP per capita of each country, with Healthy Life Expectancy included as a

Interpretation: The map allows us to identify patterns at a global level. Wealthier countries, particularly those in
Conclusion:
This dashboard offers a holistic view of the factors influencing global happiness. By exploring the correlations betw

Through these insights, policymakers, researchers, and the public can better appreciate the multi-dimensional nature
```

```
  Cell In[55], line 17
    Finally, the geographical map displays the GDP per capita of each country, with Healthy Life Expectancy included
as a tooltip for each country. The size of each country’s marker corresponds to its GDP per capita, and when hovering
over a country, additional information about its Healthy Life Expectancy is revealed.

^
SyntaxError: invalid character '’' (U+2019)
```

Authors

Abhishek Gagneja