**World maps**

1. \*\*Importing Libraries:\*\*

The code begins by importing two important libraries: `pandas` (often abbreviated as `pd`) and `plotly.graph\_objs` (abbreviated as `go`). These libraries provide tools to work with data and create interactive visualizations, respectively.

2. \*\*Reading the Dataset:\*\*

The `pd.read\_csv('gapminder\_internet.csv')` line reads a dataset from a CSV file named 'gapminder\_internet.csv' and stores it in a Pandas DataFrame called `df`. Think of a DataFrame as a table-like structure that holds data.

3. \*\*Defining a Function for the Map:\*\*

The `create\_map\_figure` function is defined to create a choropleth map. A function is like a set of instructions that can be reused later. It takes several inputs:

- `z\_data`: The data values for coloring the map.

- `colorscale`: The range of colors to use for the map.

- `title`: The title of the map.

- `colorbar\_title`: The title for the color bar.

- `reversescale`: An option to reverse the color scale if needed.

4. \*\*Custom Color Scale:\*\*

The `red\_scale` list defines a custom color scale with different shades of red. This scale will be used to color the map based on income values.

5. \*\*Creating the Choropleth Map:\*\*

The code creates the choropleth map using the `create\_map\_figure` function. It provides the `z\_data` as the income per person values, clips these values at a maximum of 100,000 for better visualization, and uses the `red\_scale` color scale. The map will be titled "Income per Person" and have a color bar titled "Income per Person".

6. \*\*Displaying the Map:\*\*

Finally, `fig.show()` displays the created map interactively.

\*\*Key Points:\*\*

- The code reads data from a CSV file into a DataFrame.

- A custom function is defined to create a choropleth map with specific inputs.

- The custom color scale `red\_scale` is used to represent different income ranges.

- The choropleth map is displayed using the `fig.show()` function.

\*\*Map Projection Type:\*\*

The code specifies the map's projection type as "mercator" using the `projection\_type` parameter in the `fig.update\_geos` and `fig.update\_layout` functions. A map projection is a method of representing a three-dimensional surface (the Earth) on a two-dimensional plane (the map). The "mercator" projection is a commonly used projection that preserves straight lines (rhumb lines) and is suitable for world maps.

By setting the projection type to "mercator," the code ensures that the map will have a familiar appearance where countries are accurately proportioned in terms of latitude and longitude. This choice of projection influences how the map looks and how the countries are represented visually.

```python

hovertemplate="<b>%{text}</b><br><br>" + title + ": %{z:,.2f}<extra></extra>"

```

Breaking down the components:

- `<b>%{text}</b>`: This part is used to display the country name in bold font.

- `<br><br>`: These tags add line breaks to create space between the country name and the data values.

- `title + ": %{z:,.2f}"`: This part is used to display the title (e.g., "Income per Person") along with the corresponding data value formatted as a floating-point number with two decimal places.

- `<extra></extra>`: This part adds any additional information to the tooltip. In this case, it's left empty.

Here's a brief explanation of the placeholders used:

- `%{text}`: This placeholder gets replaced by the value in the `text` column of your DataFrame, which is the country name.

- `%{z:,.2f}`: This placeholder gets replaced by the `z` data value (in this case, the income per person), formatted as a floating-point number with two decimal places.

So, when you hover over a specific country on the map, the tooltip will show the country name in bold, followed by the title ("Income per Person") and the corresponding income value formatted as a number with two decimal places.

**Bar graph**

```python

# Read the data from CSV

df = pd.read\_csv('gapminder\_internet.csv')

```

This line of code reads the data from a CSV file named 'gapminder\_internet.csv' into a pandas DataFrame called 'df'. This assumes that the CSV file contains data related to various countries and their economic indicators, including the 'incomeperperson' column.

```python

# Sort the DataFrame by 'incomeperperson' column in descending order and take the top 5 countries

sorted\_df\_high\_income = df.sort\_values(by='incomeperperson', ascending=False).head(5)

```

Here, the DataFrame 'df' is sorted in descending order based on the values in the 'incomeperperson' column. The `.sort\_values()` function with `ascending=False` ensures that the highest income countries are at the top. The `.head(5)` function call then selects the first five rows, which are the countries with the highest income per person.

```python

# Create a new figure for the first plot

fig, axes = plt.subplots(figsize=(10, 6)) # Only one plot in this figure

```

This code snippet creates a new figure using Matplotlib. The `plt.subplots()` function returns a figure object and a set of axes objects. In this case, it specifies that there will be a single plot within the figure, and the `figsize` parameter sets the dimensions of the figure.

```python

# Plot the bar graph for the highest income countries

bars\_high\_income = axes.bar(sorted\_df\_high\_income['country'], sorted\_df\_high\_income['incomeperperson'],

color='#800000', width=0.6)

```

Here, a bar graph is plotted on the axes created earlier. The `bars\_high\_income` variable holds the bar elements created using the `.bar()` function. The function takes the country names from the 'country' column and the corresponding income values from the 'incomeperperson' column of the 'sorted\_df\_high\_income' DataFrame. The bars are colored with the hex color code '#800000' (a shade of maroon) and have a width of 0.6.

```python

# Add labels and title to the plot

axes.set\_xlabel('Country')

axes.set\_ylabel('Income per Person (GDP per capita)')

axes.set\_title('Top 5 Countries with Highest Income per Person', fontweight='bold', color='#800000')

axes.set\_xticklabels(sorted\_df\_high\_income['country'], rotation=45, ha='right')

```

These lines set the labels and title for the plot. The `set\_xlabel()` and `set\_ylabel()` functions define the labels for the x-axis and y-axis, respectively. The `set\_title()` function sets the title of the plot with specified font weight and color. The `set\_xticklabels()` function sets the labels for the x-axis ticks using the country names from the 'sorted\_df\_high\_income' DataFrame, with rotation and horizontal alignment adjustments for better readability.

```python

# Add dollar sign symbol and value above each bar in the first plot

for bar in bars\_high\_income:

height = bar.get\_height()

axes.annotate(f'${height:.2f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),

xytext=(0, 3), textcoords="offset points", ha='center', va='bottom', color='#800000')

```

In this loop, the code iterates through each bar in the 'bars\_high\_income' list and adds an annotation above the bar. The annotation consists of the income value formatted as a dollar amount (with two decimal places), placed at the center of the bar's width (`xy` parameter). The `xytext` parameter controls the position of the annotation text relative to the bar. The `textcoords` parameter specifies that the annotation's position is given in offset points. The `ha` and `va` parameters define the horizontal and vertical alignment of the annotation text.

```python

# Adjust layout and display the first plot

plt.tight\_layout()

plt.show()

```

Finally, the `plt.tight\_layout()` function optimizes the spacing and layout of the plot elements to prevent overlap, and the `plt.show()` function displays the plot on the screen.

**Horizontal bar graph**

```python

# Import the necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

# Read the data from CSV

df = pd.read\_csv('gapminder\_internet.csv')

```

In the first few lines of the code, the necessary libraries are imported: `pandas` for data manipulation and `matplotlib.pyplot` for data visualization. Then, the code reads data from a CSV file named 'gapminder\_internet.csv' and stores it in a DataFrame named `df`.

```python

# Drop rows with missing data in specified columns

cols\_to\_check = ['incomeperperson', 'internetuserate', 'urbanrate']

df = df.dropna(subset=cols\_to\_check)

```

The code specifies a list of column names (`cols\_to\_check`) that are important for analysis. It then removes rows with missing data in any of these columns using the `dropna()` function. This step is essential to ensure that the subsequent calculations and analyses are performed on complete and valid data.

```python

# Calculate the overall score for each country

df['Overall\_Score'] = df[cols\_to\_check].max(axis=1)

```

A new column called 'Overall\_Score' is added to the DataFrame `df`. This column is calculated by taking the maximum value across the specified columns (`incomeperperson`, `internetuserate`, and `urbanrate`) for each row using the `max()` function with `axis=1`.

```python

# Sort the DataFrame by 'Overall\_Score' column in descending order and take the top 5 countries

sorted\_df\_high\_score = df.sort\_values(by='Overall\_Score', ascending=False).head(5)

```

The DataFrame `df` is sorted in descending order based on the 'Overall\_Score' column using the `sort\_values()` function. The resulting DataFrame, `sorted\_df\_high\_score`, contains the top 5 countries with the highest overall scores.

```python

# Plotting the Top 5 Countries with Highest Overall Score

plt.figure(figsize=(10, 6))

bars\_high\_score = plt.barh(sorted\_df\_high\_score['country'], sorted\_df\_high\_score['Overall\_Score'], color='#FFD700')

plt.xlabel('Overall Score')

plt.title('Top 5 Countries with Highest Overall Score', fontweight='bold', color='black')

plt.tight\_layout()

plt.show()

```

Here, a horizontal bar plot is created using `matplotlib.pyplot`. The figure size is set using `plt.figure(figsize=(10, 6))`. Horizontal bars are created using the `barh()` function, where the country names are plotted on the y-axis and their corresponding overall scores are plotted on the x-axis. The bars are colored using the hex color code `#FFD700`.

Labels, title, and layout adjustments are made using the `plt.xlabel()`, `plt.title()`, and `plt.tight\_layout()` functions, respectively. Finally, the plot is displayed using `plt.show()`.

The end result is a horizontal bar plot displaying the top 5 countries with the highest overall scores based on the specified criteria.

**Pie chart**

\*\*Step 1: Reading Data\*\*

```python

data = pd.read\_csv('gapminder\_internet.csv')

```

In this step, the code reads data from a CSV file named 'gapminder\_internet.csv' and stores it in a structure called a DataFrame. A DataFrame is like a table that holds data in rows and columns, making it easy to work with and analyze.

\*\*Step 2: Normalization Function\*\*

```python

def normalize\_column(col, avg\_val):

return col / avg\_val

```

Here, a function named `normalize\_column` is defined. This function takes two inputs: a column of data (`col`) and an average value (`avg\_val`). It calculates the normalized values of the column by dividing each value in the column by the average value.

\*\*Step 3: Calculating Averages\*\*

```python

overall\_avg\_income = data['incomeperperson'].mean()

overall\_avg\_internet = data['internetuserate'].mean()

overall\_avg\_urban = data['urbanrate'].mean()

```

The code calculates the average values for three specific columns: 'incomeperperson', 'internetuserate', and 'urbanrate' from the DataFrame. These averages will be used for normalizing the data.

\*\*Step 4: Filtering Data for Asia\*\*

```python

asia\_data = data[data['continent'] == 'Asia']

```

Here, the data is filtered to include only the rows where the 'continent' column has the value 'Asia'. This creates a new DataFrame (`asia\_data`) that contains only the data related to countries in Asia.

\*\*Step 5: Normalizing Data for Asia\*\*

```python

asia\_data['incomeperperson'] = normalize\_column(asia\_data['incomeperperson'], overall\_avg\_income)

asia\_data['internetuserate'] = normalize\_column(asia\_data['internetuserate'], overall\_avg\_internet)

asia\_data['urbanrate'] = normalize\_column(asia\_data['urbanrate'], overall\_avg\_urban)

```

The columns 'incomeperperson', 'internetuserate', and 'urbanrate' in the `asia\_data` DataFrame are being normalized using the `normalize\_column` function. This means that each value in these columns is divided by its corresponding average value, making the data comparable and bringing it to a common scale.

\*\*Step 6: Calculating Mean Values\*\*

```python

variables = ['incomeperperson', 'internetuserate', 'urbanrate']

asia\_mean\_values = asia\_data[variables].mean()

```

A list called `variables` is created, containing the names of the normalized columns. The code then calculates the mean values for these normalized columns in the `asia\_data` DataFrame. These mean values give us an idea of the average level of income, internet usage, and urban rate in Asian countries.

\*\*Step 7: Creating a Pie Chart\*\*

```python

fig, ax = plt.subplots()

ax.pie(asia\_mean\_values, labels=variables, autopct='%1.1f%%', startangle=90)

ax.axis('equal')

ax.set\_title("Distribution of Income, Internet Usage, and Urban Rate in Asia")

plt.show()

```

This part involves creating a pie chart using a plotting library (assumed to be `matplotlib`). The pie chart displays the distribution of the mean values for the normalized variables in Asia. Each slice of the pie chart corresponds to one of the variables ('incomeperperson', 'internetuserate', 'urbanrate') and represents its proportion in the total average values. The `autopct='%1.1f%%'` argument formats the percentage labels on the chart. The `startangle=90` rotates the chart so that the first slice starts from the top. The `ax.set\_title` line sets a title for the chart.

\*\*Step 8: Displaying the Pie Chart\*\*

```python

plt.show()

```