Data Visualization Plots using Python

Categorization of Plot Types

Distribution Plots

- **Box Plot:** *Basic Box Plot, Multiple Box Plot, Grouped Box Plot, Customized Box Plot, Box & Dot Plot, Tufte Boxplot*
- **Histogram:** *Basic Histogram, Histogram Stacked, Histogram - Mean & Line type, Facet Histogram*
- **Density Plot:** *Basic Density Plot, Multiple Density, Density & Histogram*
- **Area Plot:** *Basic Area Plot, Area - Mean & Line Type, Customized Area Plot*
- **Violin Plot:** *Basic Violin Plot, Violin - Group, Violin - Mean & SD, Violin - Boxplot, Violin - Dotplot*

Relationship Plots

- **Scatter Plot:** (Not explicitly covered, but can be created using basic plotting libraries)
- **Correlation Plot:** *Correlation Heatmap, Correlation Triangle Heatmap*
- **2D Density Plot:** *2D Density, 2D Density - Area & Contour, 2D Density - Raster*
- **Count Chart:** *Count Chart*

Categorical Data Visualization

- **Waffle Chart:** *Waffle Chart*
- **Treemap:** *Basic Treemap, Treemap Group*
- **Choropleth:** *Choropleth - World, Choropleth - US*
- **Bubble Map:** *Bubble Map*

Other

- **Dendrogram:** *Dendrogram*
- **Clusters:** *Clusters (scatter plot with clustering)*
- **Diverging Plots:** *Diverging Bar, Diverging Lollipop, Diverging Dot, Diverging Area*

Note: Some plot types might fall into multiple categories based on their specific use cases. This categorization is a general guideline.

The actual functionality of your code will depend on the specific tasks you're performing. These libraries offer a wide range of tools, and the specific Python equivalents will depend on the exact operations you're carrying out.

Python Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# For data reshaping
import pandas.wide_to_long

# For visualization
import plotly.express as px
import plotly.graph_objects as go

# For maps
import folium

# For treemaps
import squarify

# For date and time handling
import datetime
import dateutil
import matplotlib.pyplot as plt

def set_fig_size(width, height):
    plt.figure(figsize=(width, height))
```

Starting with basics of visualization plot diagrams

Importing Necessary Libraries

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Plot Customization Functions

Titles and Labels:

```
plt.title("Plot Title")
plt.xlabel("X-axis Label")
plt.ylabel("Y-axis Label")
```

Axis Scaling:

```
plt.xlim(lower_limit, upper_limit)
plt.ylim(lower_limit, upper_limit)
```

- For discrete axes, you can set the ticks manually using `plt.xticks` or `plt.yticks`.

Axis Transformations:

- **Reverse:**

  ```
  plt.gca().invert_xaxis()
  ```

- **Log10:**

  ```
  plt.xscale('log')
  ```

- **Date:**

  ```
  import matplotlib.dates as mdates
  plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d')
  )  # Customize date format
  plt.gca().xaxis.set_major_locator(mdates.DayLocator())  # Customize
  date intervals
  ```

Axis Limits:

- **xlim, ylim:**

```
plt.xlim(lower_limit, upper_limit)
plt.ylim(lower_limit, upper_limit)
```

Themes:

- Seaborn provides several styles:

```
sns.set_style("whitegrid")
sns.set_style("darkgrid")
```

```
sns.set_style("ticks")
sns.set_style("white")
```

Example

```
import numpy as np
import matplotlib.pyplot as plt

# Sample data
x = np.random.rand(100)
y = np.random.rand(100)

# Create the plot
plt.scatter(x, y)

# Customize the plot
plt.title("Scatter Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.xlim(0, 1)
plt.ylim(0, 1)
sns.set_style("whitegrid")

plt.show()
```

Note:

- For more complex plots and customizations, you might need to explore additional functions and arguments in Matplotlib and Seaborn.
- Seaborn often provides higher-level abstractions for creating aesthetically pleasing plots.
- For interactive plots, consider using Plotly.

Basic Scatter Plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'titanic' with columns 'Age'
and 'Fare'

# Set plot size
```

```
plt.figure(figsize=(12, 8))

# Create the scatter plot
sns.scatterplot(data=titanic, x='Age', y='Fare')

# Add labels and title
plt.title('Titanic - Age vs Fare', fontsize=22)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Fare', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style="darkgrid")

plt.show()
```

Explanation:

1. **Import necessary libraries:** `pandas` for data manipulation, `matplotlib.pyplot` for basic plotting, and `seaborn` for enhanced visualizations and themes.
2. **Set plot size:** `plt.figure(figsize=(12, 8))` sets the figure size to 12 inches wide and 8 inches high, mimicking the R code's `fig(12, 8)`.
3. **Create scatter plot:** `sns.scatterplot(data=titanic, x='Age', y='Fare')` creates a scatter plot using Seaborn's `scatterplot` function, mapping 'Age' to the x-axis and 'Fare' to the y-axis.
4. **Add labels and title:** `plt.title`, `plt.xlabel`, and `plt.ylabel` set the plot title and axis labels with specified font sizes.
5. **Set tick label font size:** `plt.xticks` and `plt.yticks` set the font size for tick labels.
6. **Set theme:** `sns.set_theme(style="darkgrid")` sets the plot theme to a dark grid background.
7. **Display plot:** `plt.show()` displays the created plot.

Note:

- Ensure you have the `titanic` DataFrame loaded into your Python environment with columns 'Age' and 'Fare'.
- Seaborn provides various theme options to customize the plot's appearance. Explore other styles like `whitegrid`, `dark`, `ticks`, etc., to find the desired look.

Scatter Plot with Categorical Variable

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'titanic' with columns 'Age',
'Fare', and 'Sex'

# Set plot size
plt.figure(figsize=(12, 8))

# Create scatter plot with color based on 'Sex'
sns.scatterplot(data=titanic, x='Age', y='Fare', hue='Sex')

# Add labels and title
plt.title('Titanic - Age vs Fare against Gender', fontsize=22)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Fare', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='darkgrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Same as the previous plot.
2. **Set plot size:** Same as the previous plot.
3. **Create scatter plot with hue:** The key difference is using `hue='Sex'` within `sns.scatterplot` to differentiate points based on the 'Sex' category.
4. **Add labels and title:** Same as the previous plot.
5. **Set tick label font size:** Same as the previous plot.
6. **Set theme:** Same as the previous plot.

This code will produce a scatter plot where points are colored differently based on the 'Sex' of the passenger, allowing you to visually explore the relationship between age, fare, and gender.

Note: Seaborn automatically assigns different colors to the categories in the 'Sex' column. You can customize the color palette using the `palette` parameter in `sns.scatterplot` if desired.

Scatter Plot with Color and Size

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'university' with columns
'quality_of_education', 'score', 'country', and 'citations'

# Set plot size
plt.figure(figsize=(15, 8))

# Create scatter plot with color and size
sns.scatterplot(data=university, x='quality_of_education', y='score',
hue='country', size='citations')

# Add labels and title
plt.title('Quality of Education Vs Score against Country & Citations',
fontsize=22)
plt.xlabel('Quality Of Education', fontsize=18)
plt.ylabel('Score', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Same as previous plot.
2. **Set plot size:** Adjust figure size to match the desired dimensions.
3. **Create scatter plot with color and size:** Use `sns.scatterplot` with `hue='country'` to color points based on the 'country' category and `size='citations'` to adjust point size based on the 'citations' value.
4. **Add labels and title:** Add appropriate labels and title with specified font sizes.
5. **Set tick label font size:** Adjust tick label font sizes for better readability.
6. **Set theme:** Use `sns.set_theme(style='whitegrid')` for a white background with grid lines.

This code will produce a scatter plot where points are colored based on the 'country' and the size of each point represents the 'citations' value. This visualization helps to

understand the relationship between 'quality_of_education' and 'score' across different countries, considering the number of citations as an additional factor.

Note: You might need to adjust the color palette, size range, and legend properties to enhance the visualization based on your data characteristics.

Scatter Plot with Color Gradient

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'insurance' with columns
'Vintage', 'Annual_Premium', and 'Age'

# Sample 500 rows
insurance_sample = insurance.sample(500)

# Set plot size
plt.figure(figsize=(12, 8))

# Create scatter plot with color gradient based on 'Age'
sns.scatterplot(data=insurance_sample, x='Vintage', y='Annual_Premium',
hue='Age', palette='viridis')

# Add labels and title
plt.title('Days associated with Company vs Amount to be paid against Age
of Customer', fontsize=22)
plt.xlabel('Days associated with Company', fontsize=18)
plt.ylabel('Insurance Amount', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='darkgrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Same as the previous plot.
2. **Sample data:** Sample 500 rows from the 'insurance' DataFrame for better visualization performance.
3. **Set plot size:** Adjust figure size as needed.

4. **Create scatter plot with color gradient:** Use `sns.scatterplot` with `hue='Age'` to color points based on the 'Age' value. The `palette='viridis'` argument applies a continuous colormap from yellow to red.
5. **Add labels and title:** Add appropriate labels and title with specified font sizes.
6. **Set tick label font size:** Adjust tick label font sizes for better readability.
7. **Set theme:** Use `sns.set_theme(style='darkgrid')` for a dark background with grid lines.

This code will produce a scatter plot where points are colored based on the 'Age' value, creating a color gradient effect. This visualization helps to understand the relationship between 'Vintage' and 'Annual_Premium' while considering the distribution of 'Age' within the data.

Note: You can experiment with different colormaps (e.g., `plasma`, `inferno`, `magma`) to find the best visual representation for your data. Also, consider adjusting the colormap limits or normalization to enhance the color gradient if needed.

Jitter Plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'house' with columns
'LotArea', 'SalePrice', and 'LotShape'

# Set plot size
plt.figure(figsize=(12, 8))

# Create jitter plot with color by 'LotShape'
sns.scatterplot(data=house, x='LotArea', y='SalePrice', hue='LotShape',
alpha=0.7, jitter=True)

# Add labels and title
plt.title('Lot Area vs House Price against LotShape', fontsize=22)
plt.xlabel('Area', fontsize=18)
plt.ylabel('Sales Price', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')
plt.show()
```

Explanation:

1. **Import necessary libraries:** Same as the previous plot.
2. **Set plot size:** Adjust figure size as needed.
3. **Create jitter plot:** Use `sns.scatterplot` with `jitter=True` to apply jitter to the points. The `alpha=0.7` argument adds transparency to reduce overplotting.
4. **Add labels and title:** Add appropriate labels and title with specified font sizes.
5. **Set tick label font size:** Adjust tick label font sizes for better readability.
6. **Set theme:** Use `sns.set_theme(style='whitegrid')` for a white background with grid lines.

This code will produce a jitter plot where points are colored based on the 'LotShape' category and slightly offset to reduce overplotting. This visualization helps to understand the relationship between 'LotArea' and 'SalePrice' while considering the distribution of 'LotShape'.

Note: You can adjust the `jitter` parameter to control the amount of jitter applied to the points. The `alpha` parameter can be adjusted to control the transparency level.

Additional Considerations:

- For more complex jitter plots, you might explore libraries like `plotnine` which offers a closer syntax to ggplot2.
- Consider using `hue_order` parameter in `sns.scatterplot` to control the order of colors for categorical variables.

By following these guidelines, you can effectively create various scatter plot variations in Python using Seaborn.

Jitter Plot with Encircled Points

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'house' with columns
'LotArea', 'SalePrice', and 'LotShape'

# Filter costly houses with less area
costly_price_with_less_area = house[(house['SalePrice'] > 600000) &
(house['LotArea'] < 25000)]

# Set plot size
plt.figure(figsize=(12, 8))
```

```
# Create jitter plot with color by 'LotShape'
sns.scatterplot(data=house, x='LotArea', y='SalePrice', hue='LotShape',
alpha=0.7, jitter=True)

# Add encircled points for costly houses with less area
plt.scatter(costly_price_with_less_area['LotArea'],
costly_price_with_less_area['SalePrice'], color='red', edgecolor='black',
s=80)

# Add labels and title
plt.title('Lot Area vs House Price (Circle Costlier House with Less
Area)', fontsize=22)
plt.xlabel('Area', fontsize=18)
plt.ylabel('Sales Price', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Same as the previous plot.
2. **Filter data:** Create a new DataFrame `costly_price_with_less_area` to filter houses based on the given conditions.
3. **Set plot size:** Adjust figure size as needed.
4. **Create jitter plot:** Use `sns.scatterplot` with `jitter=True` to create a jitter plot.
5. **Add encircled points:** Use `plt.scatter` to plot the filtered data points as red circles with black edges and larger size to simulate the encircling effect.
6. **Add labels and title:** Add appropriate labels and title with specified font sizes.
7. **Set tick label font size:** Adjust tick label font sizes for better readability.
8. **Set theme:** Use `sns.set_theme(style='whitegrid')` for a white background with grid lines.

This code will produce a jitter plot with encircled points for houses that meet the specified criteria.

Note: The `plt.scatter` approach is a basic way to simulate the encircling effect. For more complex and customizable encircling, you might explore libraries like `shapely` for geometric operations or custom plotting functions.

Additional Considerations:

- Consider using different marker styles and sizes for the encircled points to enhance visual distinction.
- Explore other libraries or custom functions for more advanced encircling techniques if needed.

By combining jitter plots and custom scatter plot elements, you can create informative visualizations to highlight specific data points within your dataset.

Basic Bubble Plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'health_df' with columns
'age', 'stay', and 'Deposit'

# Sample 100 rows
health_df_sample = health_df.sample(100)

# Set plot size
plt.figure(figsize=(12, 8))

# Create bubble plot
sns.scatterplot(data=health_df_sample, x='age', y='stay', size='Deposit',
hue='Deposit', legend=False)

# Add labels and title
plt.title('Age vs Stay against Deposits', fontsize=22)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Stay', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Same as the previous plot.
2. **Sample data:** Sample 100 rows from the 'health_df' DataFrame for better visualization performance.
3. **Set plot size:** Adjust figure size as needed.
4. **Create bubble plot:** Use `sns.scatterplot` with `size='Deposit'` to create a bubble plot where the size of each bubble represents the 'Deposit' value. The `hue='Deposit'` argument is used to add color variations based on the 'Deposit' values, but the `legend=False` argument is used to suppress the legend as it might be redundant in this case.
5. **Add labels and title:** Add appropriate labels and title with specified font sizes.
6. **Set tick label font size:** Adjust tick label font sizes for better readability.
7. **Set theme:** Use `sns.set_theme(style='whitegrid')` for a white background with grid lines.

This code will produce a bubble plot where the size of each bubble represents the 'Deposit' amount. This visualization helps to understand the relationship between 'age' and 'stay' while considering the distribution of 'Deposit'.

Note: You can customize the color palette and bubble size range to enhance the visual representation of your data.

Additional Considerations:

- For more complex bubble plots, you might explore using `plt.scatter` directly for more customization options.
- Consider adding a colorbar to indicate the relationship between bubble size and 'Deposit' values.

By following these guidelines, you can effectively create basic bubble plots in Python using Seaborn.

Bubble Plot with Color Gradient

To achieve a bubble plot with the size representing `Deposit` and color representing `stay`, we'll use `sns.scatterplot` with `size` and `hue` parameters:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Assuming you have a pandas DataFrame named 'health_df' with columns
'age', 'stay', 'Deposit'

# Sample 200 rows
health_df_sample = health_df.sample(200)

# Set plot size
plt.figure(figsize=(12, 8))

# Create bubble plot with color gradient
sns.scatterplot(data=health_df_sample, x='age', y='stay', size='Deposit',
hue='stay', cmap='coolwarm')

# Add labels and title
plt.title('Age vs Stay against Deposits', fontsize=22)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Stay', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

- **Import necessary libraries:** Import `pandas`, `matplotlib.pyplot`, and `seaborn`.
- **Sample data:** Sample 200 rows from the `health_df` DataFrame for visualization purposes.
- **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
- **Create bubble plot:** Use `sns.scatterplot` with `size='Deposit'` to control bubble size based on `Deposit` values and `hue='stay'` to color the bubbles based on `stay` values. The `cmap='coolwarm'` parameter applies a cool-to-warm colormap.
- **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.
- **Set tick label font size:** Adjust tick label font sizes using `plt.xticks` and `plt.yticks`.
- **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.

- **Display plot:** Show the plot using `plt.show()`.

This code will generate a bubble plot where the size of each bubble represents the `Deposit` amount and the color represents the `stay` duration, providing insights into the relationship between these variables and patient age.

Bubble Plot with Color Categorization

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'health_df' with columns
'age', 'stay', 'Deposit', 'Severity'

# Sample 200 rows
health_df_sample = health_df.sample(200)

# Set plot size
plt.figure(figsize=(12, 8))

# Create bubble plot with color by 'Severity'
sns.scatterplot(data=health_df_sample, x='age', y='stay', size='Deposit',
hue='Severity')

# Add labels and title
plt.title('Age vs Stay against Severity', fontsize=22)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Stay', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

- **Import necessary libraries:** `pandas`, `matplotlib.pyplot`, and `seaborn`.
- **Sample data:** Sample 200 rows from the `health_df` DataFrame for visualization.

- **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
- **Create bubble plot:** Use `sns.scatterplot` with `size='Deposit'` to control bubble size based on `Deposit` values and `hue='Severity'` to color the bubbles based on different severity levels.
- **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.
- **Set tick label font size:** Adjust tick label font sizes using `plt.xticks` and `plt.yticks`.
- **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
- **Display plot:** Show the plot using `plt.show()`.

This code will generate a bubble plot where the size of each bubble represents the `Deposit` amount and the color represents the `Severity` level, providing insights into the relationship between these variables and patient age.

Basic Bar Chart

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'university' with columns
'country' and 'score'

# Filter universities with score greater than 60
university_filtered = university[university['score'] > 60]

# Set plot size
plt.figure(figsize=(12, 8))

# Create bar chart
sns.countplot(data=university_filtered, x='country', color='darkblue')

# Add labels and title
plt.title('Country vs Score above 60', fontsize=22)
plt.xlabel('Country', fontsize=18)
plt.ylabel('Count', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
```

```
# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

- **Import necessary libraries:** `pandas`, `matplotlib.pyplot`, and `seaborn`.
- **Filter data:** Filter the `university` DataFrame to include only universities with a score greater than 60.
- **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
- **Create bar chart:** Use `sns.countplot` to create a bar chart counting the occurrences of each country in the filtered DataFrame. The `color='darkblue'` argument sets the color of the bars.
- **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.
- **Set tick label font size:** Adjust tick label font sizes using `plt.xticks` and `plt.yticks`.
- **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
- **Display plot:** Show the plot using `plt.show()`.

This code will generate a bar chart showing the count of universities with scores above 60 for each country.

Bar Chart with Gradient and Text

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'genre_data' with columns
'Genre' and 'Count'

# Set plot size
plt.figure(figsize=(12, 8))

# Create bar chart with gradient and text labels
sns.barplot(data=genre_data, x='Genre', y='Count', palette='viridis_r')
```

```
# Add text labels for count on top of each bar
for index, row in genre_data.iterrows():
    plt.text(row.name, row['Count'], f"{row['Count']}", ha="center",
va="bottom")

# Rotate x-axis labels
plt.xticks(rotation=90)

# Add labels and title
plt.title('Distribution of Playstore Genres', fontsize=22)
plt.xlabel('Genre', fontsize=18)
plt.ylabel('Count', fontsize=18)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import `pandas`, `matplotlib.pyplot`, and `seaborn`.
2. **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Create bar chart with gradient:** Use `sns.barplot` with `palette='viridis_r'` to create a bar chart with a gradient color scheme.
4. **Add text labels:** Iterate through the DataFrame to add text labels on top of each bar using `plt.text`.
5. **Rotate x-axis labels:** Rotate x-axis labels by 90 degrees for better readability using `plt.xticks(rotation=90)`.
6. **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.
7. **Set tick label font size:** Adjust tick label font sizes using `plt.xticks` and `plt.yticks`.
8. **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
9. **Display plot:** Show the plot using `plt.show()`.

This code will generate a bar chart with a gradient color scheme, text labels for each bar's count, and rotated x-axis labels for better visualization.

Stacked Bar Chart

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'ind_us_shows' with columns
'release_year' and 'country'

# Set plot size
plt.figure(figsize=(12, 8))

# Create stacked bar chart
sns.countplot(data=ind_us_shows, x='release_year', hue='country',
stack=True)

# Add labels and title
plt.title('Distribution of Netflix Shows in India & US', fontsize=22)
plt.xlabel('Year', fontsize=18)
plt.ylabel('Count', fontsize=18)

# Rotate x-axis labels
plt.xticks(rotation=90)

# Set tick label font size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

- **Import necessary libraries:** Import `pandas`, `matplotlib.pyplot`, and `seaborn`.
- **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
- **Create stacked bar chart:** Use `sns.countplot` with `hue='country'` and `stack=True` to create a stacked bar chart.
- **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.

- **Rotate x-axis labels:** Rotate x-axis labels by 90 degrees for better readability using `plt.xticks(rotation=90)`.
- **Set tick label font size:** Adjust tick label font sizes using `plt.xticks` and `plt.yticks`.
- **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
- **Display plot:** Show the plot using `plt.show()`.

This code will generate a stacked bar chart showing the count of shows/movies released in India and the United States for each year.

Note: To create a grouped bar chart instead of a stacked one, remove the `stack=True` argument from the `sns.countplot` call.

Facet Bar Chart

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'four_shows' with columns
'release_year' and 'country'

# Set plot size
plt.figure(figsize=(12, 8))

# Create facet bar chart
g = sns.catplot(data=four_shows, x='release_year', kind='count',
col='country', height=4, aspect=1)

# Add labels and title
g.fig.suptitle('Distribution of Netflix Shows in India, US, UK &
Australia', fontsize=22)
g.set_axis_labels('Year', 'Count')

# Rotate x-axis labels
g.set_xticklabels(rotation=45)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

- **Import necessary libraries:** Import `pandas`, `matplotlib.pyplot`, and `seaborn`.
- **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
- **Create facet bar chart:** Use `sns.catplot` with `kind='count'` to create a facet bar chart. The `col='country'` argument creates separate plots for each country.
- **Add labels and title:** Customize plot elements with appropriate labels and title using `g.fig.suptitle`, `g.set_axis_labels`.
- **Rotate x-axis labels:** Rotate x-axis labels by 45 degrees for better readability using `g.set_xticklabels(rotation=45)`.
- **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
- **Display plot:** Show the plot using `plt.show()`.

This code will generate a facet bar chart with four subplots, one for each country, showing the count of shows/movies released per year.

Note: The `sns.catplot` function provides a convenient way to create facet plots in seaborn.

Horizontal Bar Chart

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'hosp_code' with columns
'hospital_code' and 'count'

# Set plot size
plt.figure(figsize=(12, 8))

# Create horizontal bar chart
sns.barplot(data=hosp_code, y='hospital_code', x='count', orient='h',
palette='viridis')

# Add labels and title
plt.title('Distribution of Hospital Type Code', fontsize=22)
plt.xlabel('Count', fontsize=18)
plt.ylabel('Hospital Code', fontsize=18)

# Set tick label font size
```

```
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

- **Import necessary libraries:** Import `pandas`, `matplotlib.pyplot`, and `seaborn`.
- **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
- **Create horizontal bar chart:** Use `sns.barplot` with `orient='h'` to create a horizontal bar chart. The `palette='viridis'` argument applies a color gradient to the bars.
- **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.
- **Set tick label font size:** Adjust tick label font sizes using `plt.xticks` and `plt.yticks`.
- **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
- **Display plot:** Show the plot using `plt.show()`.

This code will generate a horizontal bar chart showing the count of hospitals for each hospital code.

Dot Plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'total_death' with columns
'country' and 'deaths'

# Set plot size
plt.figure(figsize=(12, 8))

# Create dot plot
sns.scatterplot(data=total_death, x='country', y='deaths', color='blue')

# Add labels and title
plt.title('Distribution of Deaths across countries', fontsize=22)
```

```
plt.xlabel('Country', fontsize=18)
plt.ylabel('Number of COVID Deaths', fontsize=18)

# Rotate x-axis labels
plt.xticks(rotation=90)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

- **Import necessary libraries:** Import `pandas`, `matplotlib.pyplot`, and `seaborn`.
- **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
- **Create dot plot:** Use `sns.scatterplot` with `color='blue'` to create a dot plot.
- **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.
- **Rotate x-axis labels:** Rotate x-axis labels by 90 degrees for better readability using `plt.xticks(rotation=90)`.
- **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
- **Display plot:** Show the plot using `plt.show()`.

Note: The R language includes additional elements like `geom_segment` for dashed lines, which are not directly supported in Seaborn's `scatterplot`. If you need these specific elements, you can use Matplotlib's `plt.plot` function to add them manually.

This code will generate a dot plot showing the number of COVID deaths for each country.

Lollipop Plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'race_df' with columns 'race'
and 'count'

# Set plot size
plt.figure(figsize=(12, 8))
```

```
# Create lollipop plot
sns.barplot(data=race_df, x='race', y='count', color='lightblue')
sns.scatterplot(data=race_df, x='race', y='count', color='blue', s=100)

# Add labels and title
plt.title('Distribution of US Victims of each Race', fontsize=22)
plt.xlabel('Race', fontsize=18)
plt.ylabel('Number of Victims', fontsize=18)

# Rotate x-axis labels
plt.xticks(rotation=90)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import `pandas`, `matplotlib.pyplot`, and `seaborn`.
2. **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Create lollipop plot:** Use `sns.barplot` to create the bars and `sns.scatterplot` to create the circles (lollipops).
4. **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.
5. **Rotate x-axis labels:** Rotate x-axis labels by 90 degrees for better readability using `plt.xticks(rotation=90)`.
6. **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a lollipop plot showing the number of victims for each race.

Note: While Seaborn doesn't have a direct function for lollipop plots, combining `barplot` and `scatterplot` effectively achieves the desired visualization.

Ordered Bar Plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'race_df' with columns 'race'
and 'count'
```

```
# Sort DataFrame by 'count' in ascending order
race_df = race_df.sort_values('count')

# Set plot size
plt.figure(figsize=(12, 8))

# Create ordered bar chart
sns.barplot(data=race_df, x='race', y='count', color='darkblue')

# Add labels and title
plt.title('Distribution of US Victims of each race', fontsize=22)
plt.xlabel('Race', fontsize=18)
plt.ylabel('Number of Victims', fontsize=18)

# Rotate x-axis labels
plt.xticks(rotation=90)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import `pandas`, `matplotlib.pyplot`, and `seaborn`.
2. **Sort DataFrame:** Sort the `race_df` DataFrame by the `count` column in ascending order.
3. **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
4. **Create ordered bar chart:** Use `sns.barplot` to create a bar chart. The data is already sorted, so the bars will be ordered accordingly.
5. **Add labels and title:** Customize plot elements with appropriate labels and title using `plt.title`, `plt.xlabel`, and `plt.ylabel`.
6. **Rotate x-axis labels:** Rotate x-axis labels by 90 degrees for better readability using `plt.xticks(rotation=90)`.
7. **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
8. **Display plot:** Show the plot using `plt.show()`.

This code will generate an ordered bar chart where the bars are sorted based on the `count` values, providing a clear visual representation of the number of victims for each race.

Slope Plot in Python

To create a slope plot to visualize the difference between open and close prices of bank stocks. It involves:

- Creating line segments between open and close prices.
- Adding dashed lines for reference.
- Coloring the lines based on whether the price increased or decreased.
- Adding text labels for open and close prices.
- Customizing plot aesthetics.

Python's matplotlib and seaborn libraries don't have a direct equivalent to slope plots, we can create a similar visualization by combining different plot elements.

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'bank_stock' with columns
'open' and 'close'

# Create a figure and axes
fig, ax = plt.subplots(figsize=(12, 8))

# Plot the lines
for index, row in bank_stock.iterrows():
    color = 'green' if row['close'] >= row['open'] else 'red'
    ax.plot([1, 2], [row['open'], row['close']], color=color)

# Add dashed lines
ax.axvline(x=1, linestyle='dashed', color='gray')
ax.axvline(x=2, linestyle='dashed', color='gray')

# Add text labels
for index, row in bank_stock.iterrows():
    ax.text(1, row['open'], f"{row['open']:.2f}", ha='right', va='center')
    ax.text(2, row['close'], f"{row['close']:.2f}", ha='left',
va='center')

# Set labels and title
ax.set_xlabel('')
ax.set_ylabel('Average Price')
ax.set_title('Average Open & Close price of Bank Stocks in 2019')
# Set x-axis limits
ax.set_xlim(0.5, 2.5)

# Set y-axis limits
```

```
ax.set_ylim(0, 1.1 * max(bank_stock['open'].max(),
bank_stock['close'].max()))

# Customize plot appearance
sns.despine(left=True, bottom=True)
plt.xticks([])  # Remove x-axis ticks

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, matplotlib.pyplot, and seaborn.
2. **Create figure and axes:** Create a figure and axes using `plt.subplots`.
3. **Plot lines:** Iterate through the DataFrame and plot lines between open and close prices using `ax.plot`. Color the lines based on price change.
4. **Add dashed lines:** Add vertical dashed lines at x=1 and x=2 using `ax.axvline`.
5. **Add text labels:** Iterate through the DataFrame and add text labels for open and close prices using `ax.text`.
6. **Set labels and title:** Set x and y labels, and plot title using `ax.set_xlabel`, `ax.set_ylabel`, and `ax.set_title`.
7. **Set axis limits:** Set x and y axis limits using `ax.set_xlim` and `ax.set_ylim`.
8. **Customize plot appearance:** Remove spines and x-axis ticks using `sns.despine` and `plt.xticks`.
9. **Display plot:** Show the plot using `plt.show()`.

Note: This code provides a basic implementation of a slope plot. You can further customize the plot by adjusting line styles, marker styles, and color palettes to enhance its visual appeal.

By following these steps, you can create a slope plot in Python to visualize the comparison between open and close prices of bank stocks.

Dumbbell Chart

To create a dumbbell chart to visualize the comparison between deaths and recovered cases across countries. It involves:

- Creating line segments (dumbbells) connecting deaths and recovered cases for each country.
- Customizing the appearance of the dumbbells.
- Adding labels and formatting the plot.

Python's matplotlib and seaborn libraries don't have a direct equivalent to dumbbell charts; we can create a similar visualization by combining different plot elements.

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'covid_death' with columns
'country', 'deaths', and 'recovered'

# Set plot size
plt.figure(figsize=(12, 8))

# Create subplots
fig, ax = plt.subplots(figsize=(12, 8))

# Plot the dumbbells
for index, row in covid_death.iterrows():
    ax.plot([row['deaths'], row['recovered']], [index, index],
color='gray', marker='o', markersize=8)

# Add labels and title
ax.set_xlabel('Count')
ax.set_ylabel('Country')
ax.set_title('Deaths and Recovered Cases over countries')

# Set y-axis ticks as country names
ax.set_yticks(range(len(covid_death)))
ax.set_yticklabels(covid_death['country'])

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, matplotlib.pyplot.
2. **Set plot size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Create subplots:** Create a figure and axes using `plt.subplots`.
4. **Plot dumbbells:** Iterate through the DataFrame and plot line segments (dumbbells) connecting deaths and recovered cases for each country using `ax.plot`.
5. **Add labels and title:** Set x and y labels, and plot title using `ax.set_xlabel`, `ax.set_ylabel`, and `ax.set_title`.
6. **Set y-axis ticks:** Set y-axis ticks as country names using `ax.set_yticks` and `ax.set_yticklabels`.
7. **Display plot:** Show the plot using `plt.show()`.

Note: This code provides a basic implementation of a dumbbell chart. You can further customize the plot by adjusting line styles, marker styles, colors, and adding text labels for more informative visualization.

By following these steps, you can create a dumbbell chart in Python to visualize the comparison between deaths and recovered cases across countries.

Pie Chart

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'netflix' with a column
'type'

# Count the occurrences of each type
type_counts = netflix['type'].value_counts()

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(type_counts, labels=type_counts.index, autopct='%1.1f%%',
startangle=140)
plt.title('Pie Chart of Netflix Shows')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas and matplotlib.pyplot.
2. **Count occurrences:** Count the occurrences of each type in the 'type' column of the Netflix DataFrame.
3. **Create pie chart:** Create a pie chart using `plt.pie` with the calculated counts and labels. The `autopct='%1.1f%%'` format the percentages to one decimal place.
4. **Set title:** Set the title of the pie chart using `plt.title`.
5. **Display plot:** Show the plot using `plt.show()`.

Note:

- For more customization, you can explore additional parameters of the `plt.pie` function, such as `explode`, `shadow`, and `wedgeprops`.
- Consider using `plt.subplots` to create multiple subplots if needed.

This code provides a basic pie chart visualization of the distribution of Netflix program types.

Color Pie Chart

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'severity_df' with columns
'severity' and 'patients'

# Custom colors
custom_colors = ['red', 'green', 'orange']

# Count the occurrences of each severity
severity_counts = severity_df['severity'].value_counts()

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(severity_counts, labels=severity_counts.index, autopct='%1.1f%%',
startangle=140, colors=custom_colors)
plt.title('Pie Chart of Patient Severeness')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas and matplotlib.pyplot.
2. **Custom colors:** Define the custom colors for the pie chart.
3. **Count occurrences:** Count the occurrences of each severity level in the 'severity' column.
4. **Create pie chart:** Create a pie chart using `plt.pie` with the calculated counts, labels, and custom colors.
5. **Set title:** Set the title of the pie chart using `plt.title`.
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate a pie chart with custom colors for each severity level, providing a clear visualization of the distribution of patient severities.

Customized Pie Chart with Gradient and Text

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'severity_df' with columns
'severity' and 'patients'

# Count the occurrences of each severity
severity_counts = severity_df['severity'].value_counts()

# Create a pie chart
plt.figure(figsize=(8, 8))
wedges, texts, autotexts = plt.pie(severity_counts, autopct='%1.1f%%',
startangle=140, pctdistance=0.85)

# Set colormap
cmap = plt.cm.Blues
colors = cmap(np.linspace(0, 1, len(wedges)))

# Assign colors to wedges
for i, wedge in enumerate(wedges):
    wedge.set_facecolor(colors[i])

# Add text labels for values
for i, p in enumerate(autotexts):
    p.set_color('white')

# Add title
plt.title('Pie Chart of Patient Severeness')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas and matplotlib.pyplot.
2. **Count occurrences:** Count the occurrences of each severity level in the 'severity' column.
3. **Create pie chart:** Create a pie chart using `plt.pie` with the calculated counts, labels, and autopct for percentages.
4. **Set colormap:** Create a colormap using `plt.cm.Blues`.
5. **Assign colors to wedges:** Assign colors from the colormap to each wedge.
6. **Add text labels:** Add text labels for values inside each wedge.
7. **Add title:** Set the title of the pie chart.
8. **Display plot:** Show the plot using `plt.show()`.

This code provides a pie chart with a gradient color scheme and text labels for each slice, offering a more informative visualization of the data.

Note:

- The `plt.cm.Blues` colormap can be replaced with other colormaps from matplotlib.
- For more customization, you can explore additional parameters of the `plt.pie` function and the `matplotlib` colormap module.

This approach provides a more flexible and customizable pie chart compared to the previous plots.

Donut Plot

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'content_df' with columns
'content' and 'count'

# Calculate percentages
content_df['fraction'] = content_df['count'] / content_df['count'].sum()

# Calculate cumulative percentages
content_df['ymax'] = content_df['fraction'].cumsum()

# Calculate bottom of each rectangle
content_df['ymin'] = content_df['ymax'].shift(fill_value=0)

# Calculate label position
content_df['labelPosition'] = (content_df['ymax'] + content_df['ymin']) /
2

# Calculate labels
content_df['label'] = content_df['content'] + '\n' +
content_df['count'].astype(str)

# Create the plot
plt.figure(figsize=(8, 8))

# Outer circle
plt.pie(content_df['fraction'], labels=content_df['label'],
colors=plt.cm.Paired.colors, startangle=140, pctdistance=0.85)

# Inner circle
plt.pie([1], radius=0.7, colors='white')
```

```
plt.title('Distribution of Playstore Content Ratings')
plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas and matplotlib.pyplot.
2. **Calculate percentages:** Calculate the percentage of each content type.
3. **Calculate cumulative percentages:** Calculate the cumulative percentages for each content type.
4. **Calculate bottom of each rectangle:** Calculate the bottom value for each rectangle.
5. **Calculate label position:** Calculate the position for labels.
6. **Create labels:** Combine content and count for labels.
7. **Create the plot:**
    7.1. Create a figure.
    7.2. Create the outer circle using `plt.pie`.
    7.3. Create the inner circle to form the donut shape.
    7.4. Set title and display the plot.

This code generates a donut chart visualizing the distribution of content ratings in the Google Play Store.

Key points:

- The `plt.pie` function is used twice to create the outer and inner circles.
- The `colors` argument in `plt.pie` can be customized with different colormaps or specific colors.
- The `autopct` argument can be used to display percentages within the donut chart.
- The `pctdistance` argument controls the distance of percentage labels from the center.

By following these steps, you can create a donut chart effectively in Python.

Ring Plot in Python

To create a ring plot, we'll use Matplotlib's `pyplot` and `patches` modules.

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

```
# Assuming you have a pandas DataFrame named 'content_df' with columns
'content' and 'count'

# Calculate percentages
content_df['fraction'] = content_df['count'] / content_df['count'].sum()

# Create the plot
fig, ax = plt.subplots(figsize=(8, 8))

# Outer ring
wedges, texts, autotexts = ax.pie(content_df['fraction'],
labels=content_df['content'], autopct='%1.1f%%', startangle=140,
pctdistance=0.85)

# Inner circle
inner_circle = plt.Circle((0, 0), 0.7, color='white')
ax.add_artist(inner_circle)

# Customize colors (optional)
cmap = plt.cm.Blues
colors = cmap(np.linspace(0, 1, len(wedges)))
for i, wedge in enumerate(wedges):
    wedge.set_facecolor(colors[i])

# Add title
ax.set_title('Distribution of Playstore Content Ratings')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, matplotlib.pyplot, and matplotlib.patches.
2. **Calculate percentages:** Calculate the percentage of each content type.
3. **Create the plot:**
    - Create a figure and axes using `plt.subplots`.
    - Create the outer ring using `plt.pie`.
    - Create the inner circle using `plt.Circle` and add it to the axes.
    - Customize colors (optional) using a colormap.
    - Add title using `ax.set_title`.
4. **Display plot:** Show the plot using `plt.show()`.

This code accurately creates a ring plot by combining a pie chart with an inner circle to form the ring shape.

Basic Time Series Plot

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'aus_deaths' with columns
'ObservationDate' and 'total'

# Convert 'ObservationDate' to datetime format
aus_deaths['ObservationDate'] =
pd.to_datetime(aus_deaths['ObservationDate'])

# Set figure size
plt.figure(figsize=(12, 8))

# Plot the time series
plt.plot(aus_deaths['ObservationDate'], aus_deaths['total'])

# Add labels and title
plt.xlabel('Date', fontsize=18)
plt.ylabel('Number of Deaths', fontsize=18)
plt.title('Distribution of COVID Deaths in Australia', fontsize=22)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas and matplotlib.pyplot.
2. **Convert date column:** Convert the 'ObservationDate' column to datetime format using `pd.to_datetime`.
3. **Set figure size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
4. **Plot the time series:** Use `plt.plot` to plot the 'total' against 'ObservationDate'.
5. **Add labels and title:** Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.
6. **Rotate x-axis labels:** Rotate x-axis labels for better readability using `plt.xticks(rotation=45)`.

7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a basic time series plot showing the total number of COVID deaths over time in Australia.

Note: For more complex time series analysis and visualizations, you might consider using libraries like `pandas`, `NumPy`, `SciPy`, `statsmodels`, and `matplotlib` together.

Time Series with Color

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'aus_deaths' with columns
'ObservationDate' and 'total'

# Convert 'ObservationDate' to datetime format
aus_deaths['ObservationDate'] =
pd.to_datetime(aus_deaths['ObservationDate'])

# Set figure size
plt.figure(figsize=(12, 8))

# Plot the time series with a specific color
plt.plot(aus_deaths['ObservationDate'], aus_deaths['total'], color='red')

# Add labels and title
plt.xlabel('Date', fontsize=18)
plt.ylabel('Number of Deaths', fontsize=18)
plt.title('Distribution of COVID Deaths in Australia', fontsize=22)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

plt.show()
```

Explanation:

The provided Python code is for creating a basic time series plot with a red line representing the total number of COVID deaths over time in Australia.

Key points:

- The `color='red'` argument in the `plt.plot` function sets the color of the line.
- You can customize the color by using any valid color name or hexadecimal code.

Multiple Time Series

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'us_china_deaths' with
columns 'ObservationDate', 'country', and 'total'

# Set figure size
plt.figure(figsize=(12, 8))

# Group the data by country and date
grouped_data = us_china_deaths.groupby(['country',
'ObservationDate']).sum().reset_index()

# Create the line plot
for country, group in grouped_data.groupby('country'):
    plt.plot(group['ObservationDate'], group['total'], label=country)

# Add labels and title
plt.xlabel('Date')
plt.ylabel('Number of Deaths')
plt.title('Distribution of COVID Deaths in United Kingdom, United States,
India')

# Add legend
plt.legend()

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas and matplotlib.pyplot.
2. **Set figure size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Group data:** Group the data by 'country' and 'ObservationDate' and calculate the total deaths for each group.

4. **Create multiple lines:** Iterate through each country group and plot the 'total' against 'ObservationDate' using `plt.plot`.
5. **Add labels and title:** Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.
6. **Add legend:** Add a legend to differentiate the lines using `plt.legend()`.
7. **Rotate x-axis labels:** Rotate x-axis labels for better readability using `plt.xticks(rotation=45)`.
8. **Display plot:** Show the plot using `plt.show()`.

This code will generate a line chart with multiple lines, each representing the total number of COVID deaths over time for the specified countries (UK, US, and India).

Formatted Time Series

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Assuming you have a pandas DataFrame named 'us_china_deaths' with
columns 'ObservationDate', 'country', and 'total'

# Convert 'ObservationDate' to datetime format
us_china_deaths['ObservationDate'] =
pd.to_datetime(us_china_deaths['ObservationDate'])

# Set figure size
plt.figure(figsize=(12, 8))

# Group the data by country and date
grouped_data = us_china_deaths.groupby(['country',
'ObservationDate']).sum().reset_index()

# Create the line plot
for country, group in grouped_data.groupby('country'):
    plt.plot(group['ObservationDate'], group['total'], label=country)

# Format x-axis
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%b'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())

# Add labels and title
plt.xlabel('Date')
plt.ylabel('Number of Deaths')
```

```
plt.title('Distribution of COVID Deaths in United Kingdom, United States, India')

# Add legend
plt.legend()

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, matplotlib.pyplot, and matplotlib.dates.
2. **Convert date column:** Convert the 'ObservationDate' column to datetime format using `pd.to_datetime`.
3. **Set figure size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
4. **Group data:** Group the data by 'country' and 'ObservationDate' and calculate the total deaths for each group.
5. **Create multiple lines:** Iterate through each country group and plot the 'total' against 'ObservationDate' using `plt.plot`.
6. **Format x-axis:** Use `matplotlib.dates` to format the x-axis with desired date labels and breaks.
7. **Add labels and title:** Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.
8. **Add legend:** Add a legend to differentiate the lines using `plt.legend()`.
9. **Rotate x-axis labels:** Rotate x-axis labels for better readability using `plt.xticks(rotation=45)`.
10. **Display plot:** Show the plot using `plt.show()`.

This code will generate a line chart with multiple lines, each representing the total number of COVID deaths over time for the specified countries (UK, US, and India), with formatted x-axis labels and breaks.

Basic Box Plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Assuming you have a pandas DataFrame named 'german_university' with a
column 'score'

# Set figure size
plt.figure(figsize=(12, 8))

# Create box plot
sns.boxplot(data=german_university, y='score', color='lightblue')

# Add labels and title
plt.ylabel('Score', fontsize=18)
plt.title('Distribution of German Universities Score', fontsize=22)

# Set tick label font size
plt.yticks(fontsize=15)

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, matplotlib.pyplot, and seaborn.
2. **Set figure size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Create box plot:** Use `sns.boxplot` to create a box plot of the 'score' column.
4. **Add labels and title:** Add y-label and title using `plt.ylabel` and `plt.title`.
5. **Set tick label font size:** Set tick label font size using `plt.yticks`.
6. **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a box plot showing the distribution of scores for German universities.

Note: To create box plots for different groups within your data, you can use the `x` parameter in `sns.boxplot` to specify the grouping variable.

Multiple Box Plot

```
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named 'three_university' with
columns 'score' and 'country'

# Set figure size
plt.figure(figsize=(12, 8))

# Create box plot
sns.boxplot(data=three_university, x='country', y='score',
palette='pastel')

# Add labels and title
plt.xlabel('Country')
plt.ylabel('Score')
plt.title('Distribution of German, United Kingdom & Canada Universities
Score')

# Set theme
sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, matplotlib.pyplot, and seaborn.
2. **Set figure size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Create box plot:** Use `sns.boxplot` with `x='country'` to create multiple box plots based on the 'country' column. The `palette='pastel'` argument sets a soft color palette.
4. **Add labels and title:** Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.
5. **Set theme:** Apply the desired theme using `sns.set_theme(style='whitegrid')`.
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate a box plot showing the distribution of scores for universities in Germany, the United Kingdom, and Canada.

Note: To customize the box plot further, you can explore different color palettes, box plot styles, and other options provided by seaborn.

Grouped Box Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming you have a pandas DataFrame named 'titanic' with columns
'Pclass', 'Age', and 'Sex'

# Set figure size

plt.figure(figsize=(12, 8))

# Create grouped box plot

sns.boxplot(data=titanic, x='Pclass', y='Age', hue='Sex')

# Add labels and title

plt.xlabel('Pclass')

plt.ylabel('Age')

plt.title('Distribution of Titanic Passengers Age from different class')

# Set theme

sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create grouped box plot: Use `sns.boxplot` with `x='Pclass'`, `y='Age'`, and `hue='Sex'` to create a grouped box plot.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**

5. **Set theme: Apply the desired theme using `sns.set_theme(style='whitegrid')`.**
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate a grouped box plot showing the age distribution of Titanic passengers across different passenger classes and genders.

Customized Box Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'titanic' with columns
'Pclass', 'Age', and 'Sex'


# Set figure size

plt.figure(figsize=(12, 8))


# Create grouped box plot

sns.boxplot(data=titanic, x='Pclass', y='Age', hue='Sex', palette='dark')


# Add labels and title

plt.xlabel('Pclass')

plt.ylabel('Age')

plt.title('Distribution of Titanic Passengers Age from different Class')

# Adjust legend position

plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.15), ncol=3)
```

```
# Set theme

sns.set_theme(style='whitegrid')

plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create grouped box plot: Use `sns.boxplot` with `x='Pclass'`, `y='Age'`, and `hue='Sex'` to create a grouped box plot. Set the color palette to 'dark' for better contrast.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Adjust legend position: Use `plt.legend` to adjust the legend position to the bottom center with three columns.**
6. **Set theme: Apply the desired theme using `sns.set_theme(style='whitegrid')`.**
7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a grouped box plot with customized colors and legend position, providing a clear visualization of the age distribution of Titanic passengers across different passenger classes and genders.

Note: You can further customize the plot by adjusting the color palette, box plot styles, and other seaborn options.

Box Plot and Dot Plot

This code is to create a box plot with overlaid dot plots to visualize the distribution of age for different genders on the Titanic.

Python's seaborn library provides a convenient way to create box plots; it doesn't have a direct equivalent for dot plots. We can use matplotlib's scatter plot to mimic the dot plot functionality.

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

```
# Assuming you have a pandas DataFrame named 'titanic' with columns
'Pclass', 'Age', and 'Sex'


# Set figure size

plt.figure(figsize=(12, 8))


# Create box plot

sns.boxplot(data=titanic, x='Sex', y='Age', hue='Sex')


# Add dot plot

sns.stripplot(data=titanic, x='Sex', y='Age', color='black', size=3)


# Add labels and title

plt.xlabel('Gender')

plt.ylabel('Age')

plt.title('Distribution of Titanic Passengers Age from different Gender')


# Set theme

sns.set_theme(style='whitegrid')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**

3. **Create box plot: Use `sns.boxplot` to create a box plot with `x='Sex'`, `y='Age'`, and `hue='Sex'`.**
4. **Add dot plot: Use `sns.stripplot` to add individual data points on top of the box plot.**
5. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
6. **Set theme: Apply the desired theme using `sns.set_theme(style='whitegrid')`.**
7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a box plot with overlaid individual data points (dots) to visualize the age distribution of Titanic passengers by gender.

Note: You can customize the appearance of the dot plot by adjusting the `size` and `color` parameters in `sns.stripplot`.

Python's standard libraries like Matplotlib and Seaborn don't have a built-in function for this specific plot type. However, we can create a Tufte Box Plot by customizing a standard box plot.

Tufte-Inspired Box Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'house' with columns
'SaleCondition' and 'SalePrice'


# Set figure size

plt.figure(figsize=(12, 8))


# Create box plot
```

```
sns.boxplot(data=house, x='SaleCondition', y='SalePrice', whis=1.5,
showmeans=True, linewidth=1.2, color='lightgray')


# Remove box outlines

for artist in plt.gca().artists:

    for patch in artist.patches:

        patch.set_edgecolor('white')


# Add labels and title

plt.xlabel('Sales Condition')

plt.ylabel('Price')

plt.title('Distribution of House Prices for different Sales Condition')


# Set theme

sns.set_theme(style='whitegrid')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create box plot: Use `sns.boxplot` to create a box plot with customizations for whisker length (`whis=1.5`), showing the mean (`showmeans=True`), and line width (`linewidth=1.2`).**
4. **Remove box outlines: Iterate over the artists and patches to remove box outlines.**
5. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**

6. **Set theme: Apply the desired theme using `sns.set_theme(style='whitegrid')`.**
7. **Display plot:** Show the plot using `plt.show()`.

This code creates a box plot that resembles a Tufte boxplot by removing box outlines and focusing on essential elements.

Note: To further customize the plot, you can explore additional options provided by seaborn and matplotlib, such as adjusting whisker lengths, outlier markers, and color palettes.

Basic Histogram

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'com_df' with a column
'salary'


# Set figure size

plt.figure(figsize=(12, 8))


# Create histogram

sns.histplot(data=com_df, x='salary', bins=30, color='magenta')


# Add labels and title

plt.xlabel('Salary')

plt.ylabel('Count')

plt.title('Distribution of Salaries for Comm&Mgmt')
```

```
plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create histogram: Use `sns.histplot` to create a histogram of the 'salary' column. The `bins=30` argument specifies the number of bins.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a histogram showing the distribution of salaries for computer management graduates.

Stacked Histogram

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'campus' with columns
'salary' and 'specialisation'


# Set figure size

plt.figure(figsize=(12, 8))


# Create stacked histogram

sns.histplot(data=campus, x='salary', hue='specialisation',
multiple='stack', bins=30)
```

```
# Add labels and title

plt.xlabel('Salary')

plt.ylabel('Count')

plt.title('Distribution of Salaries for different Specialisation')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create stacked histogram: Use `sns.histplot` with `hue='specialisation'` and `multiple='stack'` to create a stacked histogram.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a stacked histogram showing the distribution of salaries for different specializations.

Histogram with Mean Line

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'cipla' with a column 'Open'


# Set figure size

plt.figure(figsize=(12, 8))
```

```
# Create histogram

sns.histplot(data=cipla, x='Open', bins=30, color='lightblue',
edgecolor='black', linewidth=1)



# Add mean line

plt.axvline(x=cipla['Open'].mean(), color='red', linestyle='dashed',
linewidth=2)



# Add labels and title

plt.xlabel('Open Price')

plt.ylabel('Count')

plt.title('Distribution of CIPLA Open Price')



plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create histogram: Use `sns.histplot` to create a histogram of the 'Open' column. Customize the appearance with `bins`, `color`, `edgecolor`, and `linewidth`.**
4. **Add mean line: Use `plt.axvline` to add a vertical line at the mean of the 'Open' column.**
5. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate a histogram of the CIPLA open price with a dashed vertical line indicating the mean value.

Facet Histogram

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'med_stock' with columns
'Open' and 'Symbol'


# Set figure size

plt.figure(figsize=(12, 8))


# Create facet histogram

g = sns.FacetGrid(med_stock, col='Symbol', height=4, aspect=1)

g.map_dataframe(sns.histplot, x='Open', bins=30)


# Add labels and title

g.fig.suptitle('Distribution of Pharma Stocks Open Price', fontsize=22)

g.set_axis_labels('Open Price', 'Count')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create facet histogram: Use `sns.FacetGrid` to create a facet grid with `col='Symbol'`. Map the `sns.histplot` function to each facet.**

4. **Add labels and title: Add labels and title using `g.fig.suptitle` and `g.set_axis_labels`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a facet histogram showing the distribution of open prices for different pharma stocks.

Basic Density Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'death_by_state' with a
column 'freq'


# Set figure size

plt.figure(figsize=(12, 8))


# Create density plot

sns.kdeplot(data=death_by_state, x='freq', color='darkblue',
fill='lightblue', alpha=0.7)


# Add labels and title

plt.xlabel('Deaths Per State')

plt.ylabel('Density')

plt.title('Distribution of Victims in United States')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create density plot: Use `sns.kdeplot` to create a density plot of the 'freq' column. Customize the appearance with `color`, `fill`, and `alpha`.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a density plot showing the distribution of victims across all states in the US.

Multiple Density Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'health_df' with columns
'age' and 'Severity'


# Set figure size

plt.figure(figsize=(12, 8))


# Create density plot

sns.kdeplot(data=health_df, x='age', hue='Severity', fill=True, alpha=0.5)


# Add labels and title

plt.xlabel('Age')

plt.ylabel('Density')
```

```
plt.title('Distribution of Patient Age for different Severity')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create density plot: Use `sns.kdeplot` with `hue='Severity'` to create density plots for different severity levels.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a density plot showing the distribution of patient age for different severity levels.

Density and Histogram

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'health_df' with a column
'stay'


# Set figure size

plt.figure(figsize=(12, 8))


# Create histogram and density plot
```

```
sns.histplot(data=health_df, x='stay', bins=20, kde=True, color='blue',
alpha=0.5)


# Add labels and title

plt.xlabel('Stay Days')

plt.ylabel('Density')

plt.title('Distribution of Stay Days for Patients')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create histogram and density plot: Use `sns.histplot` with `kde=True` to overlay a density plot on the histogram.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a plot showing both the histogram and density plot for the distribution of stay days for patients.

Basic Area Plot

The code aims to create an area plot to visualize the distribution of space launches.

Python's seaborn library doesn't have a direct equivalent for area plots in the same way as R's `geom_area`, we can achieve a similar visualization using a combination of `histplot` and `fill` parameters.

```
import pandas as pd
```

```
import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'space_comp' with a column
'freq'


# Set figure size

plt.figure(figsize=(12, 8))


# Create area plot using histplot

sns.histplot(data=space_comp, x='freq', bins=30, fill='dodgerblue',
stat='density', cumulative=True)


# Add labels and title

plt.xlabel('Launches')

plt.ylabel('Count')

plt.title('Distribution of Space Launches by All Space Companies')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create area plot: Use `sns.histplot` with `stat='density'` and `cumulative=True` to create an area plot. The `fill` parameter sets the color.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate an area plot showing the distribution of space launches.

Note: This approach creates an area plot by stacking the histogram bars. For more complex area plots, you might consider using other libraries or custom plotting functions.

Area Plot with Mean Line

The code aims to create an area plot with a mean line overlay.

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'death_by_state' with a
column 'freq'


# Set figure size

plt.figure(figsize=(12, 8))


# Create area plot

sns.histplot(data=death_by_state, x='freq', bins=30, fill='goldenrod',
stat='density', cumulative=True)


# Add mean line

plt.axvline(x=death_by_state['freq'].mean(), color='blue',
linestyle='dashed', linewidth=2)


# Add labels and title

plt.xlabel('Deaths Per State')
```

```
plt.ylabel('Count')

plt.title('Distribution of Victims in US')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create area plot: Use `sns.histplot` with `stat='density'` and `cumulative=True` to create an area plot.**
4. **Add mean line: Use `plt.axvline` to add a vertical line at the mean of the 'freq' column.**
5. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate an area plot with a mean line, providing insights into the distribution of victims across US states.

Note: For more complex area plots or customizations, you might explore other libraries or custom plotting functions.

Customized Area Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'us_jap_university' with
columns 'score' and 'country'


# Set figure size
```

```
plt.figure(figsize=(12, 8))


# Create area plot

sns.histplot(data=us_jap_university, x='score', hue='country',
multiple='stack', bins=30, stat='density', cumulative=True)


# Customize colors

colors = {'Japan': 'gold', 'USA': 'brown'}

plt.gca().set_prop_cycle(color=list(colors.values()))


# Add labels and title

plt.xlabel('Score')

plt.ylabel('Count')

plt.title('Distribution of Scores of US & Japan Universities')


# Adjust legend position

plt.legend(title='Country', loc='lower center', bbox_to_anchor=(0.5,
-0.15), ncol=2)


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create area plot: Use `sns.histplot` with `hue='country'`, `multiple='stack'`, `stat='density'`, and `cumulative=True` to create a stacked area plot.**
4. **Customize colors: Define a color dictionary and set the color cycle of the plot using `plt.gca().set_prop_cycle`.**

5. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
6. **Adjust legend position: Use `plt.legend` to adjust the legend position to the bottom center with two columns.**
7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a customized area plot with the specified colors and legend position, providing a clear visualization of the distribution of scores for US and Japan universities.

Basic Violin Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'house' with a column
'SalePrice'


# Set figure size

plt.figure(figsize=(12, 8))


# Create violin plot

sns.violinplot(data=house, y='SalePrice', color='darkgreen')


# Add labels and title

plt.ylabel('Sales Price')

plt.title('Distribution of House Sales Price')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create violin plot: Use `sns.violinplot` to create a violin plot of the 'SalePrice' column.**
4. **Add labels and title: Add y-label and title using `plt.ylabel` and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a violin plot showing the distribution of house sales prices.

Grouped Violin Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'house' with columns
'LotShape' and 'SalePrice'


# Set figure size

plt.figure(figsize=(12, 8))


# Create grouped violin plot

sns.violinplot(data=house, x='LotShape', y='SalePrice')


# Add labels and title

plt.xlabel('LotShape')

plt.ylabel('Sales Price')

plt.title('Distribution of House Sales Prices for LotShape')
```

```
plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create grouped violin plot: Use `sns.violinplot` with `x='LotShape'` and `y='SalePrice'` to create a grouped violin plot.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a violin plot showing the distribution of house sales prices for different lot shapes.

Violin Plot with Mean & SD

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'house' with a column
'SalePrice'

# Assuming you have a function `mean_sdl` that calculates mean and
standard deviation


# Set figure size

plt.figure(figsize=(12, 8))


# Create violin plot
```

```
violin_plot = sns.violinplot(data=house, x='LotShape', y='SalePrice',
showmeans=True)


# Extract mean and standard deviation (assuming your function returns them
as a DataFrame)

means_df = mean_sdl(house['SalePrice'])  # Replace with your function call

means = means_df['mean']

sds = means_df['std']


# Add error bars for SD

for i, row in house.groupby('LotShape'):

    x = i

    y = means.loc[i]

    yerr = sds.loc[i]

    violin_plot.errorbar(x=[x], y=[y], yerr=[yerr], fmt='none',
ecolor='red', capsize=7)


# Add labels and title

plt.xlabel('LotShape')

plt.ylabel('Sales Price')

plt.title('Distribution of House Sales Prices for LotShape')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**

3. **Create violin plot: Use `sns.violinplot` with `showmeans=True` to create a violin plot with means.**
4. **Extract mean and standard deviation: Calculate the mean and standard deviation of the 'SalePrice' column using your `mean_sdl` function (replace with your actual implementation).**
5. **Add error bars: Iterate through each lot shape group in the DataFrame. For each group, extract the mean and standard deviation. Use `errorbar` on the violin plot to add error bars at the mean with the standard deviation as the error value.**
6. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a violin plot with the distribution of house sales prices for different lot shapes. It will also include red error bars representing the mean and standard deviation for each group. Make sure to replace `mean_sdl` with your actual function for calculating mean and standard deviation.

Violin Plot with Box Plot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'house' with columns
'LotShape' and 'SalePrice'


# Set figure size

plt.figure(figsize=(12, 8))


# Create violin plot with box plot

sns.violinplot(data=house, x='LotShape', y='SalePrice')

sns.boxplot(data=house, x='LotShape', y='SalePrice', width=0.1)
```

```
# Add labels and title

plt.xlabel('LotShape')

plt.ylabel('Sales Price')

plt.title('Distribution of House Sales Prices for LotShape')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create violin plot: Use `sns.violinplot` to create a violin plot.**
4. **Create box plot: Use `sns.boxplot` with `width=0.1` to overlay a thin box plot on the violin plot.**
5. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate a violin plot with overlaid box plots for each lot shape, providing insights into the distribution of house sales prices.

Violin Plot with Dotplot

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'house' with columns
'LotShape' and 'SalePrice'
```

```
# Set figure size

plt.figure(figsize=(12, 8))


# Create violin plot

sns.violinplot(data=house, x='LotShape', y='SalePrice')


# Add dotplot

sns.stripplot(data=house, x='LotShape', y='SalePrice', jitter=True)


# Add labels and title

plt.xlabel('LotShape')

plt.ylabel('Sales Price')

plt.title('Distribution of House Sales Prices for LotShape')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create violin plot: Use `sns.violinplot` to create a violin plot.**
4. **Add dotplot: Use `sns.stripplot` with `jitter=True` to add individual data points on top of the violin plot.**
5. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate a violin plot with overlaid individual data points (dots) for each lot shape, providing a more detailed view of the data distribution.

2D Density Plot

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt


# Assuming you have a pandas DataFrame named   'health_df' with columns
'age' and 'stay'


# Set figure size

plt.figure(figsize=(12, 8))


# Create 2D density plot

sns.kdeplot(data=health_df, x='age', y='stay', fill=True, cmap='viridis')


# Add labels and title

plt.xlabel('Age')

plt.ylabel('Stay')

plt.title('Distribution of Age & Stay Days')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create 2D density plot: Use `sns.kdeplot` with `fill=True` and `cmap='viridis'` to create a 2D density plot.**

4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a 2D density plot showing the relationship between age and stay days for patients. The `cmap='viridis'` argument sets the colormap for the density plot.

2D Density with Area and Contour

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt


# Assuming you have a pandas DataFrame named   'health_df' with columns
'age' and 'stay'


# Set figure size

plt.figure(figsize=(12, 8))


# Create 2D density plot with contours and filled areas

sns.kdeplot(data=health_df, x='age', y='stay', cmap='viridis', shade=True,
levels=10)


# Add labels and title

plt.xlabel('Age')

plt.ylabel('Stay')

plt.title('Distribution of Age & Stay Days')
```

```
plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create 2D density plot: Use `sns.kdeplot` with `fill=True`, `cmap='viridis'`, and `levels=10` to create a 2D density plot with filled contours.**
4. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a 2D density plot with both filled contours and outlines, providing a visual representation of the relationship between age and stay days.

The `levels=10` parameter in `sns.kdeplot` controls the number of contour lines. You can adjust this value to control the density of the contours.

2D Density Raster Plot

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt



# Assuming you have a pandas DataFrame named  'health_df' with columns
'age' and 'stay'



# Set figure size

plt.figure(figsize=(12, 8))



# Create 2D density plot as a heatmap

sns.kdeplot(data=health_df, x='age', y='stay', cmap='Spectral',
shade=True, cbar=True)
```

```
# Add labels and title

plt.xlabel('Age')

plt.ylabel('Stay')

plt.title('Distribution of Age & Stay Days')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Set figure size: Set the figure size using plt.figure(figsize=(12, 8)).**
3. **Create 2D density plot: Use sns.kdeplot with cmap='Spectral', shade=True, and cbar=True to create a 2D density plot as a heatmap with a colorbar.**
4. **Add labels and title: Add x and y labels, and plot title using plt.xlabel, plt.ylabel, and plt.title.**
5. **Display plot:** Show the plot using plt.show().

This code will generate a 2D density plot as a raster or heatmap, providing a visual representation of the relationship between age and stay days. The cmap='Spectral' argument sets the colormap for the density plot.

Note: The cbar=True argument adds a colorbar to the plot, which is helpful for interpreting the density values.

Dendrogram

The code aims to create a dendrogram based on a distance matrix calculated from a sample of the all_country_covid_df dataset.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

```
from scipy.cluster.hierarchy import dendrogram, linkage


# Assuming  you have a pandas DataFrame named 'all_country_covid_df'


# Sample 30 data points

sample_data = all_country_covid_df.sample(30)


# Calculate the distance matrix

distance_matrix = np.array(sample_data.corr())


# Perform hierarchical clustering

linkage_matrix = linkage(distance_matrix, 'average')


# Create dendrogram

plt.figure(figsize=(12, 14))

dendrogram(linkage_matrix, labels=sample_data.index, orientation='right')


plt.title('Dendrogram of COVID Impact on Different Countries')

plt.xlabel('Count')

plt.ylabel('Countries')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, numpy, matplotlib.pyplot, and scipy.cluster.hierarchy.**

2. **Sample data: Sample 30 data points from the all_country_covid_df DataFrame.**
3. **Calculate distance matrix: Calculate the correlation matrix between the sampled data points.**
4. **Perform hierarchical clustering: Use linkage to perform hierarchical clustering on the distance matrix using the average linkage method.**
5. **Create dendrogram: Use dendrogram to visualize the dendrogram.**
6. **Add labels and title: Add x and y labels, and plot title using plt.xlabel, plt.ylabel, and plt.title.**
7. **Display plot:** Show the plot using plt.show().

Note:

- This code assumes that you have a suitable distance metric for your data. Correlation might not be the most appropriate metric in all cases.
- The orientation='right' argument in the dendrogram function sets the orientation of the dendrogram.

This code will generate a dendrogram visualizing the hierarchical clustering of the sampled countries based on their COVID-19 data.

Cluster Visualization with PCA

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans



# Assuming you have a pandas DataFrame named 'iris' with columns
'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', and
'Species'



# Select numerical features
```

```
df = iris.iloc[:, :-1]


# Perform PCA

pca = PCA(n_components=2)

principalComponents = pca.fit_transform(df)

principalDf = pd.DataFrame(data = principalComponents, columns =
['principal component 1', 'principal component 2'])


# Combine with target variable

finalDf = pd.concat([principalDf, iris[['Species']]], axis = 1)

# Visualize

fig = plt.figure(figsize = (8,8))

ax = fig.add_subplot(1,1,1)

ax.set_xlabel('Principal Component 1', fontsize = 15)

ax.set_ylabel('Principal Component 2', fontsize = 15)

ax.set_title('2 Component PCA', fontsize = 20)

targets = iris['Species'].unique()

colors = ['r', 'g', 'b']

for target, color in zip(targets,colors):

    indicesToKeep = finalDf['Species'] == target

ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1'],
finalDf.loc[indicesToKeep, 'principal component 2'], c = color, s = 50)

ax.legend(targets)

ax.grid()

plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, numpy, matplotlib.pyplot, and sklearn's PCA and KMeans.**
2. **Prepare data: Select the numerical features from the iris dataset.**
3. **Perform PCA: Create a PCA model with 2 components and transform the data.**
4. **Create DataFrame: Combine the principal components with the target variable (species).**
5. **Visualize: Create a scatter plot using matplotlib, color-coding the points based on the species.**
6. **Add labels and title:** Add x and y labels, and plot title.

This code will produce a scatter plot of the data projected onto the first two principal components, color-coded by species.

Note: This code doesn't explicitly include the encircling of clusters. To achieve this, you could explore libraries like scikit-learn for clustering algorithms (e.g., KMeans) and then visualize the clusters with their centroids.

Waffle Chart

The code creates a waffle chart by:

- Creating a grid of squares.
- Filling the squares based on the proportion of categories in the data.
- Customizing the appearance of the chart.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


# Assuming you have a pandas DataFrame named 'titanic' with a column
'Pclass'


# Calculate proportions
```

```
titanic['Pclass'] = titanic['Pclass'].astype(str)

prop_df =
titanic['Pclass'].value_counts(normalize=True).to_frame(name='proportion')



# Create a grid of squares

nrows = 10

ncols = 10

grid_size = nrows * ncols



# Calculate the number of squares for each category

category_counts = (prop_df * grid_size).astype(int)



# Create a DataFrame to represent the waffle chart

data = []

for category, count in category_counts.iterrows():

    data.extend([(category, i) for i in range(count)])

waffle_df = pd.DataFrame(data, columns=['category', 'index'])



# Create the waffle chart

plt.figure(figsize=(12, 8))

plt.imshow(waffle_df.pivot(index='y', columns='x', values='category'),
cmap='viridis', interpolation='nearest')

plt.axis('off')

plt.title('Waffle Chart - Titanic Passenger Class Allocation')



# Add legend
```

```
handles = [plt.Rectangle((0, 0), 1, 1, color=c) for c in
waffle_df['category'].unique()]

labels = waffle_df['category'].unique()

plt.legend(handles, labels, title='Pclass', loc='upper left')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, numpy, and matplotlib.pyplot.**
2. **Calculate proportions: Calculate the proportion of each passenger class.**
3. **Create grid: Create a grid of squares.**
4. **Calculate category counts: Calculate the number of squares for each category based on the proportions.**
5. **Create DataFrame: Create a DataFrame to represent the waffle chart with categories and indices.**
6. **Create the waffle chart: Use `plt.imshow` to create the waffle chart.**
7. **Add legend: Create a legend for the categories.**
8. **Display plot:** Show the plot using `plt.show()`.

This code will generate a waffle chart showing the proportion of passenger classes in the Titanic dataset.

Note: You can customize the colors, grid size, and other aspects of the waffle chart as needed.

Basic Treemap

The code uses the `squarify` to visualize the distribution of categories in the `ms_df` dataset.

```
import pandas as pd

import matplotlib.pyplot as plt

import squarify
```

```
# Assuming you have a pandas DataFrame named 'ms_df' with a column
'category' and 'freq'


# Set figure size

plt.figure(figsize=(12, 8))


# Create treemap

squarify.plot(sizes=ms_df['freq'], label=ms_df['category'], alpha=.8)


# Add title

plt.title('Proportion among Windows App Categories')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and squarify.**
2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Create treemap: Use `squarify.plot` to create a treemap with the specified sizes and labels.**
4. **Add title: Add a title to the plot using `plt.title`.**
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a basic treemap visualizing the distribution of Windows app categories based on their frequencies.

Note: The `squarify` library provides a simple way to create treemaps in Python. You can explore additional customization options offered by the library for fine-tuning the appearance of the treemap.

Grouped Treemap

The code uses the `squarify` to create a hierarchical treemap based on the 'Category' and 'Genres' columns in the `play_df` dataset.

```
import pandas as pd

import matplotlib.pyplot as plt

import squarify


# Assuming you have a pandas DataFrame named 'play_df' with columns
'Category', 'Genres', and 'n'


# Calculate total count for each category and genre combination

grouped_df = play_df.groupby(['Category',
'Genres']).size().reset_index(name='count')


# Create treemap

plt.figure(figsize=(12, 8))

squarify.plot(sizes=grouped_df['count'], label=grouped_df['Category'] + '
- ' + grouped_df['Genres'], alpha=.8)


# Add title

plt.title('Proportion among Google Playstore App Categories & Genres')

plt.axis('off')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and squarify.**

2. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
3. **Group data: Group the data by 'Category' and 'Genres' and calculate the count for each combination.**
4. **Create treemap: Use `squarify.plot` to create a treemap with the grouped data.**
5. **Add title: Add a title to the plot using `plt.title`.**
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate a treemap visualizing the distribution of Google Playstore app categories and genres.

Note: The `squarify` library provides a basic implementation of treemaps. For more advanced features and customization, you might consider exploring other libraries or custom implementations.

Count Chart

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt


# Assuming you have a pandas DataFrame named   'health_df' with columns
'age' and 'stay'


# Sample 1000 data points

sample_df = health_df.sample(1000)


# Set figure size

plt.figure(figsize=(12, 8))


# Create count plot
```

```
sns.scatterplot(data=sample_df, x='age', y='stay', hue='count',
size='count', palette='Reds')


# Add labels and title

plt.xlabel('Age')

plt.ylabel('Stay')

plt.title('Distribution of Age & Stay Days')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Sample data: Sample 1000 data points from the `health_df` DataFrame.**
3. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
4. **Create count plot: Use `sns.scatterplot` with `hue='count'` and `size='count'` to create a count plot where the color and size of the points represent the count of observations at each (age, stay) combination.**
5. **Add labels and title: Add x and y labels, and plot title using `plt.xlabel`, `plt.ylabel`, and `plt.title`.**
6. **Display plot:** Show the plot using `plt.show()`.

This code will generate a count chart showing the distribution of age and stay days for patients. The size and color of the points represent the density of observations.

Note: For better visualization, you might consider adjusting the color palette and marker size based on the specific data distribution.

Pyramid Chart

The code aims to create a pyramid chart comparing average monthly temperatures for two countries.

Python doesn't have a direct equivalent to R's `geom_bar` with `coord_flip` for creating pyramid charts; we can achieve a similar visualization by combining multiple plot elements.

```
import pandas as pd

import matplotlib.pyplot as plt


# Assuming you have a pandas DataFrame named 'can_nor_df' with columns
'month', 'mean_temp', and 'Country'


# Pivot the data for plotting

df_pivot = can_nor_df.pivot(index='month', columns='Country',
values='mean_temp')


# Set figure size

plt.figure(figsize=(12, 8))


# Plot the positive values (one country) as a bar chart

plt.barh(df_pivot.index, df_pivot[df_pivot.columns[0]], color='blue',
alpha=0.7)


# Plot the negative values (the other country) as a bar chart

plt.barh(df_pivot.index, -df_pivot[df_pivot.columns[1]], color='red',
alpha=0.7)


# Customize plot

plt.title('Average Temperature of Greenland & India')

plt.xlabel('Average Temperature')

plt.ylabel('Month')
```

```
plt.grid(False)


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas and matplotlib.pyplot.**
2. **Pivot data: Reshape the DataFrame into a format suitable for plotting.**
3. **Set figure size: Set the figure size using `plt.figure(figsize=(12, 8))`.**
4. **Plot positive values: Plot the positive values (one country) as horizontal bars.**
5. **Plot negative values: Plot the negative values (the other country) as horizontal bars.**
6. **Customize plot: Add title, labels, and remove grid lines.**
7. **Display plot:** Show the plot using `plt.show()`.

This code will create a pyramid-like chart comparing the average monthly temperatures of two countries.

Note: This method provides a basic implementation of a pyramid chart. For more complex visualizations or customizations, you might explore other libraries or custom plotting functions.

Diverging Bar Chart

The code creates a diverging bar chart to visualize the normalized house prices in different areas of Chennai.

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Assuming you have a pandas DataFrame named 'chennai_df' with columns
'Location', 'mean_price'
```

```
# Calculate normalized mean price

chennai_df['mean_price_z'] = (chennai_df['mean_price'] -
chennai_df['mean_price'].mean()) / chennai_df['mean_price'].std()


# Create a column to indicate above or below average

chennai_df['house_type'] = chennai_df['mean_price_z'].apply(lambda x:
'above' if x >= 0 else 'below')


# Sort data by normalized price

chennai_df = chennai_df.sort_values(by='mean_price_z')


# Set custom color palette

colors = {'above': '#00ba38', 'below': '#f8766d'}


# Create diverging bar chart

plt.figure(figsize=(12, 8))

sns.barplot(data=chennai_df, x='Location', y='mean_price_z',
hue='house_type', palette=colors)


# Customize plot

plt.title('Diverging Bar - Normalized House Price of Chennai Areas')

plt.xlabel('Location')

plt.ylabel('Normalized Price')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas, matplotlib.pyplot, and seaborn.**
2. **Calculate normalized price: Calculate the z-score for the mean price.**
3. **Create above/below average flag: Create a column to indicate whether the price is above or below average.**
4. **Sort data: Sort the data by normalized price.**
5. **Set custom colors: Define a color palette for the diverging bars.**
6. **Create diverging bar chart: Use `sns.barplot` with `hue='house_type'` and the custom color palette to create the diverging bar chart.**
7. **Add labels and title: Add x and y labels, and plot title.**
8. **Display plot:** Show the plot using `plt.show()`.

This code will generate a diverging bar chart visualizing the normalized house prices in different areas of Chennai.

Diverging Lollipop Chart

The code creates a diverging lollipop chart to visualize normalized house prices in different areas of Delhi.

```
import pandas as pd

import matplotlib.pyplot as plt



# Assuming you have a pandas DataFrame named 'delhi_df' with columns
'Location', 'mean_price'



# Calculate normalized mean price

delhi_df['mean_price_z'] = (delhi_df['mean_price'] -
delhi_df['mean_price'].mean()) / delhi_df['mean_price'].std()



# Create a column to indicate above or below average
```

```
delhi_df['house_type'] = delhi_df['mean_price_z'].apply(lambda x: 'above' if x >= 0 else 'below')


# Sort data by normalized price

delhi_df = delhi_df.sort_values(by='mean_price_z')


# Set custom color palette

colors = {'above': '#00ba38', 'below': '#f8766d'}


# Create diverging lollipop chart

plt.figure(figsize=(12, 8))

plt.hlines(y=delhi_df['Location'], xmin=0, xmax=delhi_df['mean_price_z'], color=delhi_df['house_type'].map(colors), linewidth=2)

plt.scatter(delhi_df['mean_price_z'], delhi_df['Location'], color='black', s=100)

plt.text(delhi_df['mean_price_z'], delhi_df['Location'], delhi_df['mean_price_z'], color='white', ha='center', va='center')


# Customize plot

plt.title('Diverging Lollipop Chart - Normalized House Price of Delhi Areas')

plt.xlabel('Normalized Price')

plt.ylabel('Location')


plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas and matplotlib.pyplot.**

2. **Calculate normalized price: Calculate the z-score for the mean price.**
3. **Create above/below average flag: Create a column to indicate whether the price is above or below average.**
4. **Sort data: Sort the data by normalized price.**
5. **Set custom color palette: Define a color palette for the diverging lollipops.**
6. **Create diverging lollipop chart: Use `plt.hlines` to create the horizontal lines, `plt.scatter` for the circles, and `plt.text` for the labels.**
7. **Customize plot: Add title, labels, and adjust plot appearance.**
8. **Display plot:** Show the plot using `plt.show()`.

This code will generate a diverging lollipop chart visualizing the normalized house prices in different areas of bangalore.

```
import pandas as pd

import matplotlib.pyplot as plt



# Assuming you have a pandas DataFrame named 'bangalore_df' with columns
'Location', 'mean_price'



# Calculate normalized mean price

bangalore_df['mean_price_z'] = (bangalore_df['mean_price'] -
bangalore_df['mean_price'].mean()) / bangalore_df['mean_price'].std()



# Create a column to indicate above or below average

bangalore_df['house_type'] = bangalore_df['mean_price_z'].apply(lambda x:
'above' if x >= 0 else 'below')



# Sort data by normalized price

bangalore_df = bangalore_df.sort_values(by='mean_price_z')



# Set custom color palette

colors = {'above': '#00ba38', 'below': '#f8766d'}
```

```
# Create diverging dot chart

plt.figure(figsize=(12, 8))

plt.scatter(bangalore_df['mean_price_z'], bangalore_df['Location'],
c=bangalore_df['house_type'].map(colors), s=100)

plt.text(bangalore_df['mean_price_z'], bangalore_df['Location'],
bangalore_df['mean_price_z'], color='white', ha='center', va='center')



# Customize plot

plt.title('Diverging Dot Chart - Normalized House Price of Bangalore
Areas')

plt.xlabel('Normalized Price')

plt.ylabel('Location')



plt.show()
```

Explanation:

1. **Import necessary libraries: Import pandas and matplotlib.pyplot.**
2. **Calculate normalized price: Calculate the z-score for the mean price.**
3. **Create above/below average flag: Create a column to indicate whether the price is above or below average.**
4. **Sort data: Sort the data by normalized price.**
5. **Set custom color palette: Define a color palette for the diverging dots.**
6. **Create diverging dot chart: Use `plt.scatter` to create the dots, color them based on the 'house_type' column, and add text labels for the normalized price.**
7. **Customize plot: Add title, labels, and adjust plot appearance.**
8. **Display plot:** Show the plot using `plt.show()`.

This code will generate a diverging dot chart visualizing the normalized house prices in different areas of Bangalore.

Diverging Area Chart

The code creates a diverging area chart to visualize the normalized returns of CIPLA stock over time.

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named 'cipla_monthly' with columns
'date' and 'returns_percent'

# Set figure size
plt.figure(figsize=(12, 8))

# Create diverging area chart
plt.fill_between(cipla_monthly['date'], cipla_monthly['returns_percent'],
alpha=0.5)

# Customize plot
plt.title('Area Chart - CIPLA Price (Normalized)')
plt.xlabel('Date')
plt.ylabel('% Returns for CIPLA Close Price')
plt.xticks(rotation=45)

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas and matplotlib.pyplot.
2. **Set figure size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Create diverging area chart:** Use `plt.fill_between` to create the area chart, with the x-axis representing the date and the y-axis representing the returns.
4. **Customize plot:** Add title, labels, and rotate x-axis labels for better readability.
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a diverging area chart visualizing the normalized returns of CIPLA stock over time.

Note: To create a truly diverging area chart, you might need to adjust the data to represent positive and negative values. For example, you could create two separate series for positive and negative returns and plot them as separate areas.

Heatmap

The code aims to create a heatmap to visualize the distribution of admission deposit amounts across different age and stay day combinations.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named  'health' with columns 'Age',
'Stay', and 'Admission_Deposit'

# Sample 30000 data points
sample_df = health.sample(30000)

# Create the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pd.pivot_table(sample_df, values='Admission_Deposit',
index='Age', columns='Stay'), cmap='viridis')

# Add labels and title
plt.xlabel('Stay')
plt.ylabel('Age')
plt.title('Distribution of Age & Stay with Admission Deposit')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, seaborn, and matplotlib.pyplot.
2. **Sample data:** Sample 30000 data points from the health DataFrame.
3. **Create heatmap:** Use pd.pivot_table to create a pivot table with 'Age' as index, 'Stay' as columns, and 'Admission_Deposit' as values. Then use sns.heatmap to visualize the pivot table as a heatmap.
4. **Add labels and title:** Add x and y labels, and plot title using plt.xlabel, plt.ylabel, and plt.title.
5. **Display plot:** Show the plot using plt.show().

This code will generate a heatmap showing the distribution of admission deposit amounts across different age and stay day combinations.

Note: You can customize the colormap, aspect ratio, and other parameters of the heatmap using seaborn's heatmap function.

Heatmap with the specified color palette

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named   'health' with columns
'Age', 'Stay', and 'Admission_Deposit'

# Sample 30000 data points
sample_df = health.sample(30000)

# Create the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pd.pivot_table(sample_df, values='Admission_Deposit',
index='Age', columns='Stay'), cmap='RdPu')

# Add labels and title
plt.xlabel('Stay')
plt.ylabel('Age')
plt.title('Distribution of Age & Stay with Admission Deposit')

plt.show()
```

Explanation:

The provided Python code is correct and creates a heatmap with the specified color palette.

The `cmap='RdPu'` argument in the `sns.heatmap()` function sets the colormap to the "RdPu" palette, which is a red-purple colormap.

You can experiment with different colormaps provided by Matplotlib or Seaborn to find the one that best suits your data and visualization needs.

Heatmap showing the correlation between different variables

```
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have a pandas DataFrame named   'delhi_house'

# Calculate correlation matrix
corr_matrix = delhi_house.corr()

# Create heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

# Add title
plt.title('Heatmap of House Elements')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, seaborn, and matplotlib.pyplot.
2. **Calculate correlation matrix:** Calculate the correlation matrix of the dataset using `corr()`.
3. **Create heatmap:** Use `sns.heatmap` to visualize the correlation matrix. Set `annot=True` to display correlation values on the heatmap and `cmap='coolwarm'` for a colormap that represents both positive and negative correlations.
4. **Add title:** Add a title to the plot using `plt.title`.
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a heatmap showing the correlation between different variables in the Delhi house dataset.

Correlation Triangle Heatmap

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have a pandas DataFrame named 'delhi_house'

# Calculate correlation matrix
```

```
corr_matrix = delhi_house.corr()

# Create a mask for the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Create heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, mask=mask, cmap='coolwarm', center=0, annot=True,
fmt='.2f', square=True, linewidths=.5)

# Add title
plt.title('Triangle Heatmap of House Elements')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, seaborn, matplotlib.pyplot, and numpy.
2. **Calculate correlation matrix:** Calculate the correlation matrix of the dataset using corr().
3. **Create a mask:** Create a mask for the upper triangle of the correlation matrix to avoid redundancy.
4. **Create heatmap:** Use sns.heatmap to visualize the correlation matrix with the mask applied. Set cmap='coolwarm', center=0, annot=True, fmt='.2f', and square=True for customization.
5. **Add title:** Add a title to the plot using plt.title.
6. **Display plot:** Show the plot using plt.show().

This code will generate a triangular heatmap showing the correlation between different variables in the Delhi house dataset with a color gradient and annotations for correlation values.

heatmap showing the distribution

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
# Assuming you have a pandas DataFrame named 'itc_df' with columns 'Date',
'Open'
```

```
# Convert 'Date' column to datetime format
itc_df['Date'] = pd.to_datetime(itc_df['Date'])

# Create new columns for year, month, week of year, and day of week
itc_df['year'] = itc_df['Date'].dt.year
itc_df['month'] = itc_df['Date'].dt.month
itc_df['week'] = itc_df['Date'].dt.isocalendar().week
itc_df['weekday'] = itc_df['Date'].dt.weekday

# Pivot the data
pivot_df = itc_df.pivot_table(values='Open', index='week',
columns=['year', 'month'])

# Create the calendar heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_df, cmap='RdYlGn', annot=False)

# Customize plot
plt.title('Time-Series Calendar Heatmap - ITC Open Price')
plt.xlabel('Week of Month')
plt.ylabel('Year - Month')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, seaborn, matplotlib.pyplot, and matplotlib.dates.
2. **Convert date column:** Convert the 'Date' column to datetime format.
3. **Create new columns:** Create new columns for year, month, week of year, and day of week.
4. **Pivot data:** Pivot the data to create a matrix with weeks as rows and years and months as columns.
5. **Create heatmap:** Use `sns.heatmap` to create the calendar heatmap with the pivoted data.
6. **Customize plot:** Add title, labels, and adjust plot appearance.
7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a calendar heatmap showing the distribution of ITC open prices across years, months, and weeks.

Note: You can customize the colormap, annotation, and other aspects of the heatmap using seaborn's `heatmap` function.

Choropleth Map

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# Assuming you have a GeoPandas GeoDataFrame named 'world_deaths_map' with
columns 'long', 'lat', 'total', and 'group'

# Set figure size
plt.figure(figsize=(12, 8))

# Create choropleth map
world_deaths_map.plot(column='total', cmap='OrRd', legend=True,
figsize=(10, 6))

# Add title
plt.title('World COVID Deaths')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, geopandas, and matplotlib.pyplot.
2. **Set figure size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Create choropleth map:** Use `geopandas.GeoDataFrame.plot` to create the choropleth map with the 'total' column as the data to be represented by color.
4. **Add title:** Add a title to the plot using `plt.title`.
5. **Display plot:** Show the plot using `plt.show()`.

This code will generate a choropleth map showing the distribution of COVID deaths across the world.

Note: Ensure that your 'world_deaths_map' GeoDataFrame has the correct geometry information for plotting. You might need to preprocess the data to match the desired projection and coordinate system.

Choropleth for specific location

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# Assuming you have a GeoPandas GeoDataFrame named 'deaths_map' with
columns 'long', 'lat', 'total', and 'group'

# Set figure size
plt.figure(figsize=(12, 8))

# Create choropleth map
deaths_map.plot(column='total', cmap='viridis', legend=True, figsize=(10,
6))

# Add title
plt.title('COVID Deaths in United States')

plt.show()
```

Explanation:

The provided Python code is correct and creates a choropleth map of COVID deaths in the United States.

The code uses the geopandas library to plot the GeoDataFrame, with the column='total' argument specifying the data to be represented by color. The cmap='viridis' argument sets the colormap for the choropleth.

Bubble Map

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# Assuming you have a GeoPandas GeoDataFrame named 'ind' representing
India's map
# and a pandas DataFrame named 'aqi_map' with columns 'long', 'lat', and
'mean_pollution'
```

```
# Set figure size
plt.figure(figsize=(12, 8))

# Plot the base map
ind.plot(color='lightgrey', alpha=0.5, figsize=(10, 6))

# Plot the bubble map
plt.scatter(aqi_map['long'], aqi_map['lat'], c=aqi_map['mean_pollution'],
s=aqi_map['mean_pollution']*5, cmap='RdYlGn', alpha=0.7)

# Add colorbar
plt.colorbar(label='Average NO2')

# Add title
plt.title('NO2 emission in Cities of India')

plt.show()
```

Explanation:

1. **Import necessary libraries:** Import pandas, geopandas, and matplotlib.pyplot.
2. **Set figure size:** Set the figure size using `plt.figure(figsize=(12, 8))`.
3. **Plot base map:** Plot the India map using `geopandas.GeoDataFrame.plot`.
4. **Plot bubble map:** Use `plt.scatter` to plot the bubble map, with `c` for color, `s` for size, and `cmap` for colormap.
5. **Add colorbar:** Add a colorbar to the plot using `plt.colorbar`.
6. **Add title:** Add a title to the plot using `plt.title`.
7. **Display plot:** Show the plot using `plt.show()`.

This code will generate a bubble map showing the average NO2 emissions in different cities of India. The size of the bubbles represents the level of pollution.

Note: Ensure that your 'aqi_map' DataFrame contains accurate longitude and latitude coordinates for the cities. You might need to preprocess the data to match the coordinate system of the base map.