## ABHISHEK SAHANI

## Assignment_9_(Data Visualization)

# MATPLOTLIB ASSIGNMENT:

**1. Create a scatter plot using Matplotlib to visualize the relationship between two arrays, x and y for the giver data.**

**x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] y = [2, 4, 5, 7, 6, 8, 9, 10, 12, 13]**

**(Use Matplotlib for the visualization of the given questions)**

= Here's an example code snippet to create a scatter plot using Matplotlib:

```
import matplotlib.pyplot as plt

# Given data
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [2, 4, 5, 7, 6, 8, 9, 10, 12, 13]

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

# Set title and labels
plt.title('Scatter Plot of x vs y')
plt.xlabel('x')
plt.ylabel('y')

# Display grid
plt.grid(True)

# Show plot
plt.show()
```

This code will generate a scatter plot showing the relationship between the arrays `x` and `y`. Each point on the plot represents a pair of corresponding values from the two arrays.

To further enhance the visualization, consider adding:

* Trend line: `plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))(np.unique(x)), color='red')`
* Axis limits: `plt.xlim(min(x), max(x)); plt.ylim(min(y), max(y))`
* Annotations: `plt.annotate(f"({x[i]}, {y[i]})", (x[i], y[i]))` for specific points

These additions provide more context and insights into the relationship between `x` and `y`.

**2. Generate a line plot to visualize the trend of values for the given data.**

**data np.array ([3, 7, 9, 15, 22, 29, 35])**
**(Use Matplotlib for the visualization of the given questions)**

= Here's an example code snippet to create a line plot using Matplotlib:

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

# Given data
data = np.array([3, 7, 9, 15, 22, 29, 35])

# Create line plot
plt.plot(data)

# Set title and labels
plt.title('Line Plot of Data Trend')
plt.xlabel('Index')
plt.ylabel('Value')

# Display grid
plt.grid(True)

# Show plot
plt.show()
```

This code will generate a line plot showing the trend of values in the given data array.

To further enhance the visualization, consider adding:

*  Marker points: `plt.plot(data, marker='o')`
*  Trend line (polynomial fit):
   python
import numpy as np
x = np.arange(len(data))
plt.plot(x, np.poly1d(np.polyfit(x, data, 2))(x), color='red')
```
- Axis limits: plt.xlim(0, len(data)-1); plt.ylim(min(data), max(data))
- Annotations: plt.annotate(f"{value}", (i, value)) for specific points

These additions provide more context and insights into the trend of the data.

Additional Customization Options:

- Line styles: plt.plot(data, linestyle='--') (dashed), plt.plot(data, linestyle=':') (dotted)
- Colors: plt.plot(data, color='red')
- Labels and legends: plt.plot(data, label='Data Trend'); plt.legend()

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**3. Display a bar chart to represent the frequency of each item in the given array categories.**

**categories = ['A', 'B', 'C', 'D', 'E'] values= [25, 40, 30, 35, 20]**
**(Use Matplotlib for the visualization of the given questions)**

= Here's an example code snippet to create a bar chart using Matplotlib:

```
import matplotlib.pyplot as plt

# Given data
categories = ['A', 'B', 'C', 'D', 'E']
values = [25, 40, 30, 35, 20]

# Create bar chart
plt.bar(categories, values)

# Set title and labels
plt.title('Frequency of Each Category')
plt.xlabel('Category')
plt.ylabel('Frequency')

# Display grid
```
```

plt.grid(axis='y')

# Show plot
plt.show()
```

This code will generate a bar chart representing the frequency of each item in the given array categories.

To further enhance the visualization, consider adding:

* Horizontal bar chart: `plt.barh(categories, values)`
* Colors: `plt.bar(categories, values, color='skyblue')`
* Labels and annotations:

python
for i, value in enumerate(values):
  plt.text(i, value + 2, str(value), ha='center')

- Legends: `plt.legend(labels=categories)`
- Rotation of x-axis labels: `plt.xticks(rotation=45)`

These additions provide more context and insights into the frequency distribution.

_Additional Customization Options:_

- Bar styles: `plt.bar(categories, values, edgecolor='black')`
- Error bars: `plt.errorbar(categories, values, xerr=1, fmt='o')`
- Multiple datasets:

values2 = [20, 35, 25, 40, 30]
plt.bar(categories, values, label='Dataset 1')
plt.bar(categories, values2, label='Dataset 2', bottom=values)
```

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**4. Create a histogram to visualize the distribution of values in the array data.**

**data np.random.normal(0, 1, 1000)**
**(Use Matplotlib for the visualization of the given questions)**

= Here's an example code snippet to create a histogram using Matplotlib:

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

# Generate random data
data = np.random.normal(0, 1, 1000)

# Create histogram
plt.hist(data, bins=30, density=True, alpha=0.6, color='skyblue')

# Set title and labels
plt.title('Histogram of Normal Distribution')
plt.xlabel('Value')
plt.ylabel('Probability Density')

# Display grid
plt.grid(axis='y')

# Show plot
plt.show()
```

This code generates a histogram representing the distribution of values in the given array data.

To further enhance the visualization, consider adding:

* Curve fitting:

```
python
import scipy.stats as stats
x = np.linspace(-3, 3, 100)
plt.plot(x, stats.norm.pdf(x, 0, 1), color='red')
```

- Multiple distributions:

```
data2 = np.random.normal(1, 2, 1000)
plt.hist(data2, bins=30, density=True, alpha=0.6, color='orange')
```

- Legends:

```
plt.legend(labels=['Normal Distribution'])
```

- Interactive visualization:

```
import plotly.graph_objects as go
fig = go.Figure(data=[go.Histogram(x=data)])
fig.show()
```

These additions provide more context and insights into the distribution.

Additional Customization Options:

- Bin sizes: plt.hist(data, bins=50)
- Colors: plt.hist(data, color='blue')
- Transparency: plt.hist(data, alpha=0.8)
- Vertical/Horizontal orientation: plt.hist(data, orientation='horizontal')

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**5. Show a pie chart to represent the percentage distribution of different sections in the array 'sections'.**

**sections = ['Section A', 'Section B', 'Section C', 'Section D']**
**sizes= [25, 30, 15, 30]**

**(Use Matplotlib for the visualization of the given questions)**

= Here's an example code snippet to create a pie chart using Matplotlib:

```
import matplotlib.pyplot as plt

# Given data
sections = ['Section A', 'Section B', 'Section C', 'Section D']
sizes = [25, 30, 15, 30]

# Create pie chart
plt.pie(sizes, labels=sections, autopct='%1.1f%%')

# Set title
plt.title('Percentage Distribution of Sections')
```

```
# Equal aspect ratio ensures that pie is drawn as a circle
plt.axis('equal')

# Show plot
plt.show()
```

This code generates a pie chart representing the percentage distribution of different sections.

To further enhance the visualization, consider adding:

* Exploding a slice:

```
python
explode = [0.1, 0, 0, 0]  # Explode Section A
plt.pie(sizes, labels=sections, autopct='%1.1f%%', explode=explode)
```

- Custom colors:

```
colors = ['skyblue', 'orange', 'green', 'red']
plt.pie(sizes, labels=sections, colors=colors, autopct='%1.1f%%')
```

- Shadow:

```
plt.pie(sizes, labels=sections, autopct='%1.1f%%', shadow=True)
```

- Start angle:

```
plt.pie(sizes, labels=sections, autopct='%1.1f%%', startangle=90)
```

These additions provide more context and insights into the distribution.

Additional Customization Options:

- Label formatting: autopct='%1.2f%%'
- Font size: textprops={'fontsize': 14}
- Radius: radius=1.2

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

# SEABORN ASSIGNMENT:

**1. Create a scatter plot to visualize the relationship between two variables, by generating a synthetic dataset.**
**(Use Seaborn for the visualization of the given questions)**

= Here's an example code snippet to create a scatter plot using Seaborn:

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Generate synthetic dataset
np.random.seed(0)
x = np.random.randn(100)
y = 2 * x + np.random.randn(100)

# Create DataFrame
df = pd.DataFrame({'x': x, 'y': y})

# Create scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='x', y='y', data=df)

# Set title and labels
plt.title('Scatter Plot of x vs y')
plt.xlabel('x')
plt.ylabel('y')
```

```
# Show plot
plt.show()
```

This code generates a scatter plot visualizing the relationship between two variables.

To further enhance the visualization, consider adding:

* Regression line:
  python
sns.regplot(x='x', y='y', data=df)

- Hue:

df['category'] = np.where(df['y'] > 0, 'Positive', 'Negative')
sns.scatterplot(x='x', y='y', hue='category', data=df)

* Style:
  python
sns.scatterplot(x='x', y='y', style='category', data=df)

- Size:
```
```
sns.scatterplot(x='x', y='y', size='category', data=df)

These additions provide more context and insights into the relationship.

Additional Customization Options:

* Marker styles: `marker='o'`
* Colors: `color='red'`
* Alpha (transparency): `alpha=0.7`
* Legend location: `legend='upper right'`

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**2. Generate a dataset of random numbers. Visualize the distribution of a numerical variable.**

**(Use Seaborn for the visualization of the given questions)**

= Here's an example code snippet to generate a dataset and visualize the distribution of a numerical variable using Seaborn:

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Generate dataset of random numbers
np.random.seed(0)
data = np.random.randn(1000)

# Create DataFrame
df = pd.DataFrame({'Values': data})

# Visualize distribution
plt.figure(figsize=(8, 6))
sns.histplot(df['Values'], bins=30, kde=True)

# Set title and labels
plt.title('Distribution of Numerical Variable')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Show plot
plt.show()
``
```

This code generates a histogram with a kernel density estimate (KDE) to visualize the distribution of the numerical variable.

To further enhance the visualization, consider adding:

* Boxplot:
  python

```
sns.boxplot(df['Values'])
```

- Violin plot:

```
sns.violinplot(df['Values'])
```

* ECDF plot:

python
sns.ecdfplot(df['Values'])

- Custom colors:
```
```
sns.histplot(df['Values'], bins=30, kde=True, color='skyblue')

* Custom bin sizes:
python
sns.histplot(df['Values'], bins=50, kde=True)
```

These additions provide more context and insights into the distribution.

Additional Customization Options:

- Horizontal orientation: orient='horizontal'
- Log scale: log_scale=True
- Legend location: legend='upper right'
- Font size: fontsize=14

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**3. Create a dataset representing categories and their corresponding values. Compare different categories based on numerical values.**

**(Use Seaborn for the visualization of the given questions)**

= Here's an example code snippet to create a dataset and compare different categories based on numerical values using Seaborn:

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Create dataset
categories = ['A', 'B', 'C', 'D', 'E']
values = [25, 40, 30, 35, 20]
```
```

```
errors = [5, 3, 4, 2, 6]

# Create DataFrame
df = pd.DataFrame({
   'Category': categories,
   'Value': values,
   'Error': errors
})

# Visualize comparison
plt.figure(figsize=(8, 6))
sns.barplot(x='Category', y='Value', data=df, ci=None)
plt.errorbar(df['Category'], df['Value'], yerr=df['Error'], fmt='none', capsize=5)

# Set title and labels
plt.title('Comparison of Categories')
plt.xlabel('Category')
plt.ylabel('Value')

# Show plot
plt.show()
```

This code generates a bar plot with error bars to compare different categories based on numerical values.

To further enhance the visualization, consider adding:

- Horizontal bar plot:
```
```
```
sns.barplot(x='Value', y='Category', data=df, ci=None)
```

*  Hue (multiple variables):
   python
```
df['Subcategory'] = ['X', 'Y', 'X', 'Y', 'X']
sns.barplot(x='Category', y='Value', hue='Subcategory', data=df)
```

- Custom colors:

```
sns.barplot(x='Category', y='Value', data=df, ci=None, color='skyblue')
```

*  Custom error bar caps:
   python

```
plt.errorbar(df['Category'], df['Value'], yerr=df['Error'], fmt='none', capsize=10)
```

These additions provide more context and insights into the comparison.

Additional Customization Options:

- Log scale: plt.yscale('log')
- Legend location: plt.legend(loc='upper right')
- Font size: plt.rcParams.update({'font.size': 14})
- Rotation of x-axis labels: plt.xticks(rotation=45)

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**4. Generate a dataset with categories and numerical values. Visualize the distribution of a numerical variable across different categories.**
**(Use Seaborn for the visualization of the given questions)**

= Here's an example code snippet to generate a dataset and visualize the distribution of a numerical variable across different categories using Seaborn:

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Generate dataset
np.random.seed(0)
categories = ['A', 'B', 'C', 'D']
values = np.random.randn(100)
category_labels = np.repeat(categories, 25)

# Create DataFrame
df = pd.DataFrame({
   'Category': category_labels,
   'Value': values
})

# Visualize distribution
plt.figure(figsize=(8, 6))
sns.violinplot(x='Category', y='Value', data=df)

# Set title and labels
```

```
plt.title('Distribution of Numerical Variable Across Categories')
plt.xlabel('Category')
plt.ylabel('Value')

# Show plot
plt.show()
```

This code generates a violin plot to visualize the distribution of the numerical variable across different categories.

To further enhance the visualization, consider adding:

- Boxplot:
```
```
sns.boxplot(x='Category', y='Value', data=df)

* Swarm plot:
  python
sns.swarmplot(x='Category', y='Value', data=df)

- Bar plot with standard deviation:

sns.barplot(x='Category', y='Value', data=df, ci='sd')

* Custom colors:
  python
sns.violinplot(x='Category', y='Value', data=df, palette='Set2')

These additions provide more context and insights into the distribution.

Additional Customization Options:

- Split violin plot:
```
```
sns.violinplot(x='Category', y='Value', data=df, split=True)

* Horizontal orientation:
  python
sns.violinplot(x='Value', y='Category', data=df)

- Log scale:

```
plt.yscale('log')
```

* Legend location:
  python

```
plt.legend(loc='upper right')
```

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**5. Generate a synthetic dataset with correlated features. Visualize the correlation matrix of a dataset using a heatmap.**
**(Use Seaborn for the visualization of the given questions)**

= Here's an example code snippet to generate a synthetic dataset with correlated features and visualize the correlation matrix using a heatmap with Seaborn:

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Generate synthetic dataset with correlated features
np.random.seed(0)
n_samples = 1000
n_features = 5

# Define correlation structure
cov_matrix = np.array([
   [1.0, 0.8, 0.7, 0.4, 0.2],
   [0.8, 1.0, 0.9, 0.6, 0.3],
   [0.7, 0.9, 1.0, 0.8, 0.5],
   [0.4, 0.6, 0.8, 1.0, 0.7],
   [0.2, 0.3, 0.5, 0.7, 1.0]
])

# Generate data
data = np.random.multivariate_normal(mean=np.zeros(n_features), cov=cov_matrix, size=n_samples)
```

```
# Create DataFrame
df = pd.DataFrame(data, columns=[f'Feature {i+1}' for i in range(n_features)])

# Compute correlation matrix
corr_matrix = df.corr()

# Visualize correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True)

# Set title
plt.title('Correlation Matrix')

# Show plot
plt.show()
```

This code generates a synthetic dataset with correlated features and visualizes the correlation matrix using a heatmap.

To further enhance the visualization, consider adding:

* Masking upper triangle:
  python
```
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
```

- Custom color map:

```
sns.heatmap(corr_matrix, annot=True, cmap='viridis', square=True)
```

* Reordering features by correlation:
  python
```
corr_order = corr_matrix.columns
sns.heatmap(corr_matrix[corr_order].T[corr_order].T, annot=True, cmap='coolwarm', square=True)
```

These additions provide more context and insights into the correlation structure.

Additional Customization Options:

- Displaying only significant correlations:
```
```

```
corr_sig = corr_matrix > 0.5
sns.heatmap(corr_matrix * corr_sig, annot=True, cmap='coolwarm', square=True)
```

* Using different annotation formats:
  python

```
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', square=True)
```

- Saving the plot:

```
plt.savefig('correlation_matrix.png', dpi=300)
```

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

# PLOTLY ASSIGNMENT:

**1. Using the given dataset, to generate a 3D scatter plot to visualize the distribution of data points in a three- dimensional space.**

**np.random.seed(30) data = { '**

**'X': np.random.uniform(-10, 16, 300), 'Y': np.random.uniform(-18, 18, 306),**

**Z': np.random.uniform(-10, 10, 300)**

**df pd. DataFrame(data)**

**(Use Plotly for the visualization of the given questions)**

= Here's an example code snippet to generate a 3D scatter plot using Plotly:

```
# Import necessary libraries
import plotly.express as px
import pandas as pd
import numpy as np

# Set random seed
```

```
np.random.seed(30)

# Generate dataset
data = {
   'X': np.random.uniform(-10, 16, 300),
   'Y': np.random.uniform(-18, 18, 300),
   'Z': np.random.uniform(-10, 10, 300)
}

# Create DataFrame
df = pd.DataFrame(data)

# Generate 3D scatter plot
fig = px.scatter_3d(df, x='X', y='Y', z='Z')

# Customize plot
fig.update_layout(
   title='3D Scatter Plot',
   scene=dict(
      xaxis_title='X',
      yaxis_title='Y',
      zaxis_title='Z'
   )
)

# Show plot
fig.show()
``
```

This code generates a 3D scatter plot visualizing the distribution of data points in a three-dimensional space.

To further enhance the visualization, consider adding:

*  Color by category:

```
python
df['Category'] = np.random.choice(['A', 'B', 'C'], size=300)
fig = px.scatter_3d(df, x='X', y='Y', z='Z', color='Category')
```

- Size by value:

```
df['Value'] = np.random.uniform(0, 10, 300)
fig = px.scatter_3d(df, x='X', y='Y', z='Z', size='Value')
```

- Custom colors:

```
fig.update_layout(
   scene=dict(
      xaxis=dict(bgcolor='lightgray'),
      yaxis=dict(bgcolor='lightgray'),
      zaxis=dict(bgcolor='lightgray')
   )
)
```

- Interactive hover text:

```
fig = px.scatter_3d(df, x='X', y='Y', z='Z', hover_name='Index')
```

These additions provide more context and insights into the distribution.

Additional Customization Options:

- Different marker symbols:

```
fig = px.scatter_3d(df, x='X', y='Y', z='Z', symbol='Category')
```

- Animation:

```
fig = px.scatter_3d(df, x='X', y='Y', z='Z', animation_frame='Value')
```

- Custom camera angle:

```
fig.update_layout(
   scene_camera=dict(
      eye=dict(x=1.5, y=1.5, z=1.5)
   )
)
```

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**2. Using the Student Grades, create a violin plot to display the distribution of scores across different grade categories.**

**np.random.seed(15) data = { 'Grade': np.random.choice(['A', 'B', 'C', 'D', 'F'], 200), Score: np.random.randint(50, 189, 200) }**

**<34>**

**Q**

**df pd.DataFrame(data)**

**1. Using the sales data, generate a heatmap to visualize the variation in sales across different months and days.**

**np.random.seed (26) data**

**Month: np.random.choice(['Jan', 'Feb', 'Mar', 'Apr', 'Kay'], 100),**

**Day: np.random.choice(range(1, 31), 100),**

**'Sales': np.random.randint(1066, 5969, 108)**

**df pd.DataFrame (data)**

**(Use Plotly for the visualization of the given questions)**

= Here are the solutions to the two problems:

Problem 1: Violin Plot for Student Grades

```
import plotly.express as px
import pandas as pd
import numpy as np
```

```
np.random.seed(15)

data = {
   'Grade': np.random.choice(['A', 'B', 'C', 'D', 'F'], 200),
   'Score': np.random.randint(50, 190, 200)
}

df = pd.DataFrame(data)

fig = px.violin(df, x='Grade', y='Score', box=True, hover_data=df.columns)

fig.update_layout(
   title='Distribution of Scores Across Grades',
   xaxis_title='Grade',
   yaxis_title='Score'
)

fig.show()
```

**Problem 2: Heatmap for Sales Data**

```
python
import plotly.express as px
import pandas as pd
import numpy as np

np.random.seed(26)

data = {
   'Month': np.random.choice(['Jan', 'Feb', 'Mar', 'Apr', 'May'], 100),
   'Day': np.random.choice(range(1, 31), 100),
   'Sales': np.random.randint(1066, 5969, 100)
}

df = pd.DataFrame(data)

# Pivot data for heatmap
df_pivot = df.pivot_table(index='Month', columns='Day', values='Sales', aggfunc=np.sum)

fig = px.imshow(df_pivot, text_auto=True, aspect="auto")

fig.update_layout(
   title='Sales Variation Across Months and Days',
   xaxis_title='Day',
   yaxis_title='Month'
)
```

```
fig.show()
```

These codes generate a violin plot for student grades and a heatmap for sales data using Plotly.

To further enhance the visualizations, consider adding:

Violin Plot:

- Color by grade category
- Custom colors
- Interactive hover text

Heatmap:

- Custom color scale
- Displaying only significant sales values
- Animation

Additional Customization Options:

Violin Plot:

- Different violin plot styles
- Box plot customization

Heatmap:

- Different colormap options
- Displaying sales values as percentages

By using these customization options, you can tailor the visualizations to effectively communicate the insights and trends in the data.

**3. Using the sales data, generate a heatmap to visualize the variation in sales across different months and days.**

**np.random.seed (20)**

**data**

**{ "Month': np.random.choice(["Jan', 'Feb', 'Mar', 'Apr', 'May'], 100),**

**'Day: np.random.choice(range(1, 31), 189), 'Sales': np.random.randint(1066, 5000, 106)**

**}**

**df pd.DataFrame(data)**

**(Use Plotly for the visualization of the given questions)**

= Here's an example code snippet to generate a heatmap using Plotly:

```
import plotly.express as px
import pandas as pd
import numpy as np

np.random.seed(20)

data = {
   'Month': np.random.choice(['Jan', 'Feb', 'Mar', 'Apr', 'May'], 100),
   'Day': np.random.choice(range(1, 31), 100),
   'Sales': np.random.randint(1066, 5000, 100)
}

df = pd.DataFrame(data)

# Pivot data for heatmap
df_pivot = df.pivot_table(index='Month', columns='Day', values='Sales', aggfunc=np.sum)

fig = px.imshow(df_pivot, text_auto=True, aspect="auto")

fig.update_layout(
   title='Sales Variation Across Months and Days',
   xaxis_title='Day',
   yaxis_title='Month'
)

fig.show()
```

This code generates a heatmap visualizing the variation in sales across different months and days.

To further enhance the visualization, consider adding:

* Custom color scale: `color_continuous_scale='Viridis'`
* Displaying only significant sales values: `zmin=2000`
* Animation: `animation_frame='Month'`
* Interactive hover text: `hover_data=df.columns`

Additional Customization Options:

* Different colormap options: `color_continuous_scale='Plasma'`
* Displaying sales values as percentages: `zmax=5000`
* Customizing axis labels: `xaxis_title='Day of Month'`

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**Tips and Variations**:

* Use `px.density_heatmap` for continuous data.
* Use `px.imshow` for discrete data.
* Experiment with different aggregation functions (`aggfunc`) in `pivot_table`.
* Consider using `seaborn.heatmap` for alternative visualization options.

**4. Using the given x and y data, generate a 3D surface plot to visualize the function z = $\sin(x^2+y^2)$**

**X np.linspace (-5, 5, 100) ynp.linspace(-5, 5, 100) x, y np.meshgrid(x, y) znp.sin(np.sqrt(x*2+ y*2))**

**data = {**

**'X': x.flatten(), 'Y': y.flatten().**

**'Z': z.flatten()**

**}**

**df pd.DataFrame(data)**

**(Use Plotly for the visualization of the given questions)**

= Here's an example code snippet to generate a 3D surface plot using Plotly:

```
import plotly.express as px
import pandas as pd
import numpy as np

# Generate x and y data
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
```

```
# Create meshgrid
x, y = np.meshgrid(x, y)

# Calculate z values
z = np.sin(np.sqrt(x**2 + y**2))

# Flatten data for DataFrame
data = {
    'X': x.flatten(),
    'Y': y.flatten(),
    'Z': z.flatten()
}

# Create DataFrame
df = pd.DataFrame(data)

# Generate 3D surface plot
fig = px.scatter_3d(df, x='X', y='Y', z='Z', mode='markers')

# Update layout for surface plot
fig.update_layout(
    title='3D Surface Plot of z = sin(x²+y²)',
    scene=dict(
        xaxis_title='X',
        yaxis_title='Y',
        zaxis_title='Z'
    )
)

# Show plot
fig.show()
```

Alternatively, you can use `plotly.graph_objects` to create a surface plot:

```
python
import plotly.graph_objects as go

# Generate 3D surface plot
fig = go.Figure(data=[go.Surface(x=x, y=y, z=z)])

# Update layout
fig.update_layout(
    title='3D Surface Plot of z = sin(x²+y²)',
    scene=dict(
        xaxis_title='X',
        yaxis_title='Y',
        zaxis_title='Z'
```

```
    )
)

# Show plot
fig.show()
```

These codes generate a 3D surface plot visualizing the function $z = \sin(x^2+y^2)$.

To further enhance the visualization, consider adding:

- Custom color scale: colorscale='Viridis'
- Opacity: opacity=0.5
- Interactive hover text: hoverinfo='text'
- Customizing axis labels: xaxis_title='X Axis'

Additional Customization Options:

- Different colormap options: colorscale='Plasma'
- Displaying contour lines: contours=dict(z=dict(show=True))
- Rotating the plot: scene=dict(camera=dict(eye=dict(x=1.5, y=1.5, z=1.5)))

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**5. Using the given dataset, create a bubble chart to represent each country's population (y-axis), GDP (x- axis), and bubble size proportional to the population.**

**np.random.seed(25) data = { Country': ['USA', 'Canada', 'UK', 'Germany', 'France']. 'Population': np.random.randint(100, 1000, 50, 5) GDP: np.random.randint(500, 2000,**

**df= pd. DataFrame (data)**

**(Use Plotly for the visualization of the given questions)**

= Here's an example code snippet to generate a bubble chart using Plotly:

```
import plotly.express as px
import pandas as pd
import numpy as np

# Set random seed
np.random.seed(25)
```

```
# Generate dataset
data = {
   'Country': ['USA', 'Canada', 'UK', 'Germany', 'France'],
   'Population': np.random.randint(100000, 1000000, 5),
   'GDP': np.random.randint(500000, 2000000, 5)
}

# Create DataFrame
df = pd.DataFrame(data)

# Generate bubble chart
fig = px.scatter(df, x='GDP', y='Population', size='Population', hover_name='Country',
                 size_max=50, range_size=[100000, 1000000])

# Update layout
fig.update_layout(
   title='Country Population and GDP',
   xaxis_title='GDP',
   yaxis_title='Population'
)

# Show plot
fig.show()
```

This code generates a bubble chart representing each country's population (y-axis), GDP (x-axis), and bubble size proportional to the population.

To further enhance the visualization, consider adding:

- Custom color palette: color_discrete_sequence=['blue', 'green', 'red', 'yellow', 'purple']
- Interactive hover text: hover_data=['Country', 'Population', 'GDP']
- Customizing axis labels: xaxis_title='GDP (Billions)'
- Displaying trend line: trendline='ols'

Additional Customization Options:

- Different marker symbols: symbol='Country'
- Animation: animation_frame='Year'
- Custom font family: font_family='Arial'
- Displaying data labels: text='Country'

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

# BOKEH ASSIGNMENT:

**1. Create a Bokeh plot displaying a sine wave. Set x-values from 0 to 10 and y-values as the sine of x.**

**(Use Bokeh for the visualization of the given questions)**

= Here's an example code snippet to generate a Bokeh plot displaying a sine wave:

```
import numpy as np
from bokeh.plotting import figure, show

# Generate x-values from 0 to 10
x = np.linspace(0, 10, 100)

# Calculate y-values as sine of x
y = np.sin(x)

# Create Bokeh plot
p = figure(title="Sine Wave", x_axis_label='x', y_axis_label='sin(x)')

# Add line renderer to plot
p.line(x, y, line_width=2)

# Show plot
show(p)
```

This code generates a Bokeh plot displaying a sine wave.

To further enhance the visualization, consider adding:

* Custom line colors: `line_color='blue'`
* Axis limits: `x_range=(0, 10), y_range=(-1.1, 1.1)`
* Grid lines: `x_grid=True, y_grid=True`
* Interactive hover tool: `hover_tooltooltips=[("x", "$x"), ("y", "$y")]`

Additional Customization Options:

* Different line styles: `line_dash='dashed'`
* Plot width and height: `plot_width=800, plot_height=600`
* Custom font sizes: `title_text_font_size='20pt'`
* Displaying legend: `legend_label='Sine Wave'`

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**Tips and Variations**:

* Use `np.cos` for cosine wave.
* Experiment with different `np.linspace` parameters.
* Add multiple lines to plot using `p.line` multiple times.
* Use Bokeh's interactive tools for zooming and panning.

**2. Create a Bokeh scatter plot using randomly generated x and y values. Use different sizes and colors for the markers based on the 'sizes' and 'colors' columns.**
**(Use Bokeh for the visualization of the given questions)**
= Here's an example code snippet to generate a Bokeh scatter plot using randomly generated x and y values:

```
import numpy as np
from bokeh.plotting import figure, show
import pandas as pd
import random

# Generate random x and y values
np.random.seed(0)
x = np.random.randn(100)
y = np.random.randn(100)

# Generate random sizes and colors
sizes = np.random.randint(5, 20, 100)
colors = ["#%02x%02x%02x" % (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)) for _ in range(100)]

# Create DataFrame
df = pd.DataFrame({'x': x, 'y': y, 'sizes': sizes, 'colors': colors})

# Create Bokeh plot
p = figure(title="Scatter Plot", x_axis_label='x', y_axis_label='y')
```

```
# Add scatter renderer to plot
p.scatter(x='x', y='y', size='sizes', fill_color='colors', line_color=None, source=df)

# Show plot
show(p)
```

This code generates a Bokeh scatter plot using randomly generated x and y values, with different sizes and colors for the markers.

To further enhance the visualization, consider adding:

* Custom axis limits: `x_range=(-5, 5), y_range=(-5, 5)`
* Grid lines: `x_grid=True, y_grid=True`
* Interactive hover tool: `hover_tooltooltips=[("x", "$x"), ("y", "$y"), ("size", "$size")]`
* Legend: `legend_label='Markers'`

Additional Customization Options:

* Different marker shapes: `marker='circle_x'`
* Plot width and height: `plot_width=800, plot_height=600`
* Custom font sizes: `title_text_font_size='20pt'`
* Displaying trend line: `trend_line='ols'`

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

**Tips and Variations**:

* Use `np.random.uniform` for uniform distribution.
* Experiment with different `np.random.randn` parameters.
* Add multiple scatter renderers to plot using `p.scatter` multiple times.
* Use Bokeh's interactive tools for zooming and panning.

## 3. Generate a Bokeh bar chart representing the counts of different fruits using the following dataset.

## fruits ['Apples', 'Oranges', 'Bananas', 'Pears'] counts (20, 25, 30, 35)
## (Use Bokeh for the visualization of the given questions)

= Here's an example code snippet to generate a Bokeh bar chart representing the counts of different fruits:

```
from bokeh.plotting import figure, show
```

```
import pandas as pd

# Define dataset
fruits = ['Apples', 'Oranges', 'Bananas', 'Pears']
counts = [20, 25, 30, 35]

# Create DataFrame
df = pd.DataFrame({'Fruit': fruits, 'Count': counts})

# Create Bokeh plot
p = figure(title="Fruit Counts", x_axis_label='Fruit', y_axis_label='Count',
        x_range=fruits, plot_width=800, plot_height=600)

# Add bar renderer to plot
p.vbar(x='Fruit', top='Count', width=0.5, source=df, legend_label='Fruits')

# Show plot
show(p)
```

This code generates a Bokeh bar chart representing the counts of different fruits.

To further enhance the visualization, consider adding:

- Custom colors: fill_color='navy'
- Hover tool: hover_tooltooltips=[("Fruit", "$x"), ("Count", "$top")]
- Axis limits: y_range=(0, 40)
- Grid lines: x_grid=True, y_grid=True

Additional Customization Options:

- Different bar styles: line_color='white'
- Plot title font size: title_text_font_size='20pt'
- Customizing axis labels: x_axis_label='Fruit Type'
- Displaying data labels: text_label=['Count']

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

Tips and Variations:

- Use p.bar for horizontal bars.
- Experiment with different x_range parameters.
- Add multiple bar renderers to plot using p.vbar multiple times.
- Use Bokeh's interactive tools for zooming and panning.

**4. Create a Bokeh histogram to visualize the distribution of the given data.**

**data_hist np.random.randn(1000)**

**hist, edges np.histogram(data_hist, hins-30)**
**(Use Bokeh for the visualization of the given questions)**

= Here's an example code snippet to generate a Bokeh histogram:

```
import numpy as np
from bokeh.plotting import figure, show
import pandas as pd

# Generate random data
np.random.seed(0)
data_hist = np.random.randn(1000)

# Create histogram
hist, edges = np.histogram(data_hist, bins=30)

# Create DataFrame
df = pd.DataFrame({'hist': hist, 'left': edges[:-1], 'right': edges[1:]})

# Create Bokeh plot
p = figure(title="Histogram", x_axis_label='Value', y_axis_label='Frequency',
        x_range=(-4, 4), plot_width=800, plot_height=600)

# Add quad renderer to plot
p.quad(source=df, bottom=0, top='hist', left='left', right='right',
     fill_color='skyblue', line_color='white')

# Show plot
show(p)
```

This code generates a Bokeh histogram visualizing the distribution of the given data.

To further enhance the visualization, consider adding:

- Custom colors: fill_color='navy'
- Hover tool: hover_tooltooltips=[("Value", "$x"), ("Frequency", "$top")]
- Axis limits: y_range=(0, 100)
- Grid lines: x_grid=True, y_grid=True

Additional Customization Options:

- Different histogram styles: line_color='black'
- Plot title font size: title_text_font_size='20pt'
- Customizing axis labels: x_axis_label='Data Value'
- Displaying data labels: text_label=['Frequency']

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

Tips and Variations:

- Use p.rect for rectangular histogram bars.
- Experiment with different bins parameters.
- Add multiple histogram renderers to plot using p.quad multiple times.
- Use Bokeh's interactive tools for zooming and panning.

Bokeh Histogram with Interactive Tools

To add interactive tools, use the tools parameter:

p = figure(tools='pan,wheel_zoom,reset,hover')

This adds pan, wheel zoom, reset, and hover tools to the plot.

**5. Create Bokeh heatmap using the provided dataset.**

**data_heatmap np.random.rand(10, 10) x = np.linspace (9, 1, 18) y= np.linspace(8, 1, 18) xx, yy np.meshgrid(x, y)**

**(Use Bokeh for the visualization of the given questions)**

= Here's an example code snippet to generate a Bokeh heatmap:

```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.models import LinearColorMapper, ColorBar
from bokeh.palettes import Viridis256
```

```
# Generate random data
np.random.seed(0)
data_heatmap = np.random.rand(10, 10)

# Create meshgrid
x = np.linspace(9, 1, 10)
y = np.linspace(8, 1, 10)
xx, yy = np.meshgrid(x, y)

# Create color mapper
color_mapper = LinearColorMapper(palette=Viridis256, low=data_heatmap.min(),
high=data_heatmap.max())

# Create Bokeh plot
p = figure(title="Heatmap", x_axis_label='X', y_axis_label='Y',
        x_range=(1, 9), y_range=(1, 8), tools='hover')

# Add image renderer to plot
p.image(image=[data_heatmap], x=1, y=1, dw=8, dh=7, palette=Viridis256)

# Add color bar
color_bar = ColorBar(color_mapper=color_mapper, location=(0, 0))
p.add_layout(color_bar, 'right')

# Show plot
show(p)
```

This code generates a Bokeh heatmap using the provided dataset.

To further enhance the visualization, consider adding:

- Custom colors: palette=Magma256
- Hover tool: hover_tooltooltips=[("X", "$x"), ("Y", "$y"), ("Value", "@image")]
- Axis limits: x_range=(0, 10), y_range=(0, 10)
- Grid lines: x_grid=True, y_grid=True

Additional Customization Options:

- Different heatmap styles: interpolation='nearest'
- Plot title font size: title_text_font_size='20pt'
- Customizing axis labels: x_axis_label='Column'
- Displaying data labels: text_label=['Value']

By using these customization options, you can tailor the visualization to effectively communicate the insights and trends in the data.

Tips and Variations:

- Use p.rect for rectangular heatmap cells.
- Experiment with different palette options.
- Add multiple heatmap renderers to plot using p.image multiple times.
- Use Bokeh's interactive tools for zooming and panning.