Data Visualization

1. Creating histograms to visualize the distribution of numerical data.

Histogram is a graph showing the number of observations within each given interval.

To visualize the distribution of numerical data using histograms in Python, you can make use of libraries such as `matplotlib` or `seaborn`, which are commonly used for data visualization in machine learning projects.

`bins`: Defines the number of bins or intervals in the histogram.
`color`: Specifies the color of the bars.
`edgecolor`: Specifies the color of the edges of the bars (useful for distinguishing individual bars).

Demo Code:

```
import matplotlib.pyplot as plt

# Example numerical data
data = [12, 15, 12, 15, 17, 22]

# Create a histogram
plt.hist(data, bins=10, edgecolor='black')  # bins define the number of bins in the histogram

# Add titles and labels
plt.title('Histogram of Data')
plt.xlabel('Value')
plt.ylabel('Frequency')

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

Demo Code:

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

# Example DataFrame
```

```
df = pd.DataFrame({
    'values': [ 29, 35, 40, 41, 42, 45, 50, 60, 62, 65]
})

# Create a histogram from the DataFrame column
df['values'].hist(bins=10, edgecolor='black')

# Add titles and labels
plt.title('Histogram of Values from DataFrame')
plt.xlabel('Value')
plt.ylabel('Frequency')

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

2. Plotting scatter plots to explore relationships between two continuous variables.

Demo Code:

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

# Example data in a DataFrame
data = pd.DataFrame({
    'x': [1, 2, 3, 4, 5, 6, 7, 8, 9],
    'y': [2, 3, 5, 7, 11, 13, 17, 19, 23]
})
# Create scatter plot directly from the DataFrame
data.plot(kind='scatter', x='x', y='y', color='green')
# Add titles and labels
plt.title('Scatter Plot of x vs. y')
plt.xlabel('X values')
plt.ylabel('Y values')
# Show the plot
plt.show()
```

A scatter plot is a diagram where each value in the data set is represented by a dot.

To explore relationships between two continuous variables in a dataset, scatter plots are an effective way to visualize how one variable might relate to another. In Python, you can use libraries like `matplotlib` and `seaborn` to create scatter plots.

- Generating bar charts to compare frequencies of different categories in a dataset.
- Constructing box plots to identify outliers and assess data variability.
- Using heatmaps to visualize correlation matrices between features.

Demo Code:

```
import matplotlib.pyplot as plt
# Example data: Two continuous variables
x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
y = [2, 3, 5, 7, 11, 13, 17, 19, 23]
# Create scatter plot
plt.scatter(x, y, color='blue')  # Customize color as needed
# Add titles and labels
plt.title('Scatter Plot of x vs. y')
plt.xlabel('X values')
plt.ylabel('Y values')
# Show the plot
plt.show()
```

Demo Code:

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

# Example data in a DataFrame
data = pd.DataFrame({
    'x': [1, 2, 3, 4, 5, 6, 7, 8, 9],
    'y': [2, 3, 5, 7, 11, 13, 17, 19, 23]
})
# Create scatter plot directly from the DataFrame
data.plot(kind='scatter', x='x', y='y', color='green')
# Add titles and labels
plt.title('Scatter Plot of x vs. y')
plt.xlabel('X values')
plt.ylabel('Y values')
# Show the plot
```

```
plt.show()
```

Color: You can change the color of the scatter plot points by specifying the `color` or `c` parameter.
Size: If you want the size of the points to vary based on another feature, you can use the `s` parameter in Matplotlib or `size` in Seaborn.
Markers: You can adjust the shape of the points by using different marker styles like `o`, `^`, or `s`.
Transparency: The `alpha` parameter controls the transparency (0 for fully transparent, 1 for fully opaque).

3. Generating bar charts to compare frequencies of different categories in a dataset.

A bar plot uses rectangular bars to represent data categories, with bar length or height proportional to their values. It compares discrete categories, with one axis for categories and the other for values.

To generate bar charts to compare frequencies of different categories in a dataset, you can use Python's data visualization libraries such as Matplotlib or Seaborn

1. Prepare Your Dataset

Ensure you have a dataset loaded into a Pandas DataFrame. Identify the categorical column whose frequencies you want to compare.

2. Count Frequencies

Use `value_counts()` to count the occurrences of each category.

3. Generate Bar Charts

Use Matplotlib or Seaborn to plot the frequencies.

Demo Code:

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

# Example dataset
data = {
    'Category': ['A', 'B', 'A', 'C', 'B', 'A', 'C', 'C', 'A', 'B', 'B', 'C', 'A']
}
df = pd.DataFrame(data)

# Count frequencies of each category
category_counts = df['Category'].value_counts()

# Matplotlib Bar Chart
plt.figure(figsize=(8, 5))
plt.bar(category_counts.index, category_counts.values, color='skyblue')
plt.title('Category Frequency (Matplotlib)', fontsize=16)
plt.xlabel('Category', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()

# Seaborn Bar Chart
plt.figure(figsize=(8, 5))
sns.barplot(x=category_counts.index, y=category_counts.values, palette='viridis')
plt.title('Category Frequency (Seaborn)', fontsize=16)
plt.xlabel('Category', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()
```

4. Constructing box plots to identify outliers and assess data variability.

Box plots are useful for visualizing the distribution of a dataset, identifying potential outliers, and assessing data variability.

Detecting Outliers using the Boxplot Method

Outliers, the data points that diverge significantly from the overall pattern, can heavily influence the results of statistical analysis and make it challenging to draw accurate conclusions. Boxplot is one of the most effective and widely used methods to detect outliers in data.

Understanding Boxplots

A boxplot, also known as a box-and-whisker plot, is a graphical representation of data distribution using five summary statistics: minimum, first quartile (Q1), median (Q2 or the second quartile), third quartile (Q3), and maximum. The box plot is divided into four sections, each representing 25% of the data points.

Steps to Detect Outliers Using a Boxplot

To detect outliers using boxplots, you can follow these steps:

1. Arrange the data in ascending order.
2. Calculate the first quartile (Q1), median (Q2), and third quartile (Q3).
3. Determine the interquartile range (IQR) by subtracting Q1 from Q3 (IQR = Q3 – Q1).
4. Calculate the lower and upper bounds for outliers. The lower bound and upper bound are included in the non-outlier zone.
   Lower Bound = Q1 – 1.5 * IQR
   Upper Bound = Q3 + 1.5 * IQR
5. Identify any data points that fall below the lower bound or above the upper bound as outliers.

Example 1

Given dataset: {22, 35, 2, 4, 20, 39, 37, 102, 101, 36}

Step 1: Arrange the data in ascending order: {2, 4, 20, 22, 35, 36, 37, 39, 101, 102}

Step 2: Calculate Q1, Q2, and Q3:

- Q2 (second quartile or median) = median({2, 4, 20, 22, 35, 36, 37, 39, 101, 102}) =(35+36)/2= 35.5
- Q1 (first quartile) = the median of the lower half (excluding Q2) = median({2, 4, 20, 22, 35}) = 20
- Q3 (third quartile) = the median of the upper half (excluding Q2) = median({36, 37, 39, 101, 102}) = 39

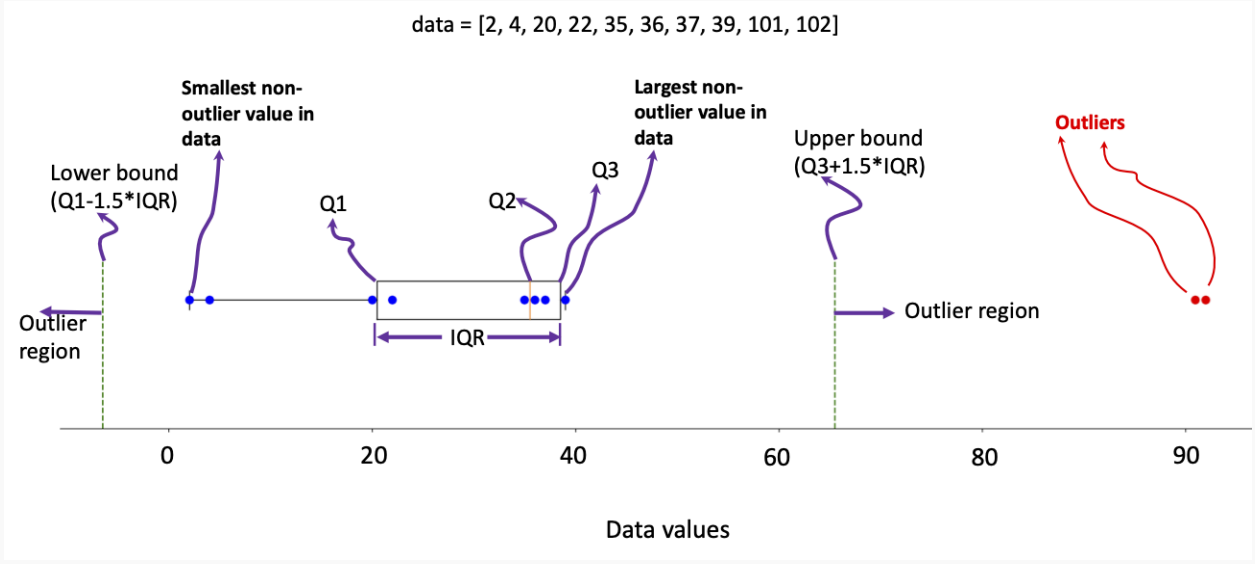
Step 3: Determine the IQR: IQR = Q3 – Q1 = 39 – 20 = 19

Step 4: Calculate the lower and upper bounds for outliers:

- Lower Bound = Q1 – 1.5 * IQR = 20 – 1.5 * 19 = -8.5
- Upper Bound = Q3 + 1.5 * IQR = 39 + 1.5 * 19 = 67.5

Step 5: Identify outliers: Any data points smaller than -8.5 or above 67.5 are considered outliers. In this dataset, we have two outliers: 101 and 102.

The following figure shows the corresponding boxplot.



Demo Code:

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

# Generate simple data
np.random.seed(42)
data = {
   'Feature_1': np.random.normal(50, 10, 100),
   'Feature_2': np.random.normal(30, 5, 100),
   'Feature_3': np.random.uniform(20, 40, 100)
}

df = pd.DataFrame(data)

# Plot simple box plot
sns.boxplot(data=df)
```

```
plt.title("Box Plots of Features")
plt.show()

# Detect outliers using IQR method
for column in df.columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    print(f"Outliers in {column}:\n{outliers}\n")
```

1. **Q1 = df[column].quantile(0.25)**:
    - This computes the first quartile (Q1) of the data in the specified column. The first quartile is the value below which 25% of the data lies. In other words, it's the 25th percentile of the data.
2. **Q3 = df[column].quantile(0.75)**:
    - This calculates the third quartile (Q3) of the data in the specified column. The third quartile is the value below which 75% of the data lies. It's the 75th percentile.
3. **IQR = Q3 - Q1**:
    - This computes the Interquartile Range (IQR), which is the difference between the third quartile (Q3) and the first quartile (Q1). The IQR measures the spread of the middle 50% of the data.
4. **lower_bound = Q1 - 1.5 * IQR**:
    - This determines the lower bound for potential outliers. Any data point below this value is considered a potential outlier. The factor of 1.5 is commonly used in box plots to define outliers.
5. **upper_bound = Q3 + 1.5 * IQR**:
    - Similarly, this calculates the upper bound for potential outliers. Any data point above this value is considered a potential outlier.

the code is used to detect outliers in a dataset by identifying values that fall below the lower bound or above the upper bound, which are calculated based on the IQR. These boundaries are often used in box plots to visualize the spread and outliers of the data.

5. Using heatmaps to visualize correlation matrices between features.

A heatmap is a data visualization technique used to display data in a matrix format where individual values are represented by colors. It is especially useful for visualizing complex data, such as correlation matrices, where it is important to quickly identify patterns, relationships, or trends between variables.

Demo Code:

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

# Generate simple data
#np.random.seed(42)
data = {
    'Feature_1': np.random.normal(50, 10, 100),
    'Feature_2': np.random.normal(30, 5, 100),
    'Feature_3': np.random.normal(10, 2, 100),
    'Feature_4': np.random.uniform(20, 40, 100)
}

df = pd.DataFrame(data)

# Compute correlation matrix
correlation_matrix = df.corr()

# Plot heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix Heatmap")
plt.show()
```

Explanation:

1. Import library
2. Generate random data:

   `np.random.seed(42)`: Sets the seed for the random number generator to ensure reproducibility of the random values.

A dictionary `data` is created with 4 features (`Feature_1`, `Feature_2`, `Feature_3`, `Feature_4`). Each feature is populated with 100 random values:

- `Feature_1`: Values drawn from a normal distribution with a mean of 50 and a standard deviation of 10.
- `Feature_2`: Values drawn from a normal distribution with a mean of 30 and a standard deviation of 5.
- `Feature_3`: Values drawn from a normal distribution with a mean of 10 and a standard deviation of 2.
- `Feature_4`: Values drawn from a uniform distribution between 20 and 40.
- `df = pd.DataFrame(data)`: Converts the dictionary into a pandas DataFrame named `df`, where each key in the dictionary becomes a column in the DataFrame.

3. **Compute the correlation matrix**:

- `correlation_matrix = df.corr()`: Calculates the correlation matrix of the numerical columns in the DataFrame (`Feature_1`, `Feature_2`, `Feature_3`, `Feature_4`). This matrix shows the pairwise correlations (Pearson correlation coefficient) between the features. The value ranges from -1 (perfect negative correlation) to 1 (perfect positive correlation).

4. **Plot the heatmap**:

- `sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')`: Uses Seaborn to create a heatmap of the correlation matrix. The parameters are:
    - `correlation_matrix`: The data to be visualized.
    - `annot=True`: Annotates each cell with the numerical value of the correlation.
    - `cmap='coolwarm'`: Specifies the color map to use for the heatmap, with blue tones for negative correlations and red tones for positive correlations.

5. **Display the plot**:

- `plt.title("Correlation Matrix Heatmap")`: Sets the title of the plot to "Correlation Matrix Heatmap".
- `plt.show()`: Displays the heatmap on the screen.