import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

\_

df = pd.read\_csv('iris.csv')

df.head()

\_

print("Feature and their types:")

df.dtypes

\_

plt.figure(figsize=(12, 6))

for i, column in enumerate(df.columns[:-1]): # Exclude 'species' column

plt.subplot(2, 2, i + 1)

ax = plt.hist(df[column], edgecolor="black")

plt.gca().bar\_label(plt.gca().containers[0], fmt='%d') # Add count labels

plt.title(f"Histogram of {column}")

plt.xlabel(column)

plt.ylabel("Frequency")

plt.tight\_layout()

plt.show()

\_

plt.figure(figsize=(12, 6))

for i, column in enumerate(df.columns[:-1]): # Exclude 'species' column

ax = plt.subplot(2, 2, i + 1)

# Create boxplot and store it in a container

box\_container = sns.boxplot(x=df[column], ax=ax, color='salmon')

plt.title(f"Boxplot of {column}")

plt.tight\_layout()

plt.show()

\_

for column in df.columns[:-1]: # Exclude 'species' column

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)][column]

print(f"\nFeature: {column}")

print(f" Mean: {df[column].mean():.2f}, Median: {df[column].median():.2f}, Std Dev: {df[column].std():.2f}")

print(f" Outliers Detected: {'Yes' if not outliers.empty else 'No'}","\n " f"Outlier Values: {outliers.tolist()}" if not outliers.empty else "")

print("-" \* 40)

\_

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

for i, column in enumerate(df.columns[:-1]): # Exclude 'species' column

plt.subplot(2, 2, i + 1)

sns.violinplot(x=df["variety"], y=df[column], palette="Set2", hue=df['variety'])

plt.title(f"Violin Plot of {column} by variety")

plt.tight\_layout()

plt.show()

It seems you've imported the necessary libraries to work with data analysis and visualization. Here's a quick rundown of each library:

1. **import pandas as pd**:
   * Pandas is a powerful library for data manipulation and analysis. It provides data structures like DataFrame and Series, which are essential for working with structured data (like CSV, Excel files, etc.).
   * pd is the alias commonly used for Pandas to make it shorter and easier to call functions.
2. **import matplotlib.pyplot as plt**:
   * Matplotlib is a plotting library used to create static, animated, and interactive visualizations in Python.
   * pyplot is a module within matplotlib that provides a collection of functions that make it easy to create a variety of plots (like line plots, histograms, bar plots, etc.).
   * plt is the alias used for pyplot to simplify the syntax.
3. **import seaborn as sns**:
   * Seaborn is a higher-level interface for creating attractive statistical graphics. It is built on top of Matplotlib and provides better default styles and easier plotting options.
   * sns is the alias used for Seaborn.

These libraries are frequently used together for data analysis and visualization, especially when you want to explore datasets and present the results visually.

### Typical Usage:

* **Pandas** would be used to load and manipulate your dataset.
* **Matplotlib** and **Seaborn** would be used to create various visualizations (like histograms, boxplots, scatter plots, etc.) to analyze the data further.

If you have a dataset you'd like to work with or need help with any specific code, feel free to share it!

Code 4 explanation:

**Code Explanation:**

1. **plt.figure(figsize=(12, 6))**:
   * This line creates a figure with a size of **12 inches wide** by **6 inches tall**. This ensures that all subplots fit nicely in the figure.
2. **for i, column in enumerate(df.columns[:-1]):**:
   * This loop iterates over each column in the dataset df. It skips the last column (df.columns[:-1]) assuming the last column is the target variable, like the 'species' column in the Iris dataset.
   * enumerate() is used to get both the index i and the column name column.
3. **plt.subplot(2, 2, i + 1)**:
   * This creates a 2x2 grid of subplots (2 rows and 2 columns). Each iteration of the loop places the plot in a different position in the grid.
   * i + 1 ensures that each plot gets placed in the correct position (1st, 2nd, 3rd, and 4th positions).
4. **ax = plt.hist(df[column], edgecolor="black")**:
   * This creates a histogram for the current column (df[column]). It represents the frequency distribution of values in that column.
   * edgecolor="black" adds a black border around the bars in the histogram.
5. **plt.gca().bar\_label(plt.gca().containers[0], fmt='%d')**:
   * This line adds labels to the bars of the histogram, showing the frequency of each bin.
   * plt.gca() gets the current axis, and .containers[0] accesses the first set of bars in the plot (the bars in the histogram).
   * fmt='%d' formats the label as an integer (no decimal places).
6. **plt.title(f"Histogram of {column}")**:
   * This sets the title for the current subplot, indicating the column name for which the histogram is plotted.
7. **plt.xlabel(column)**:
   * This sets the x-axis label to be the current column name.
8. **plt.ylabel("Frequency")**:
   * This sets the y-axis label to "Frequency", representing the number of occurrences of data points in each bin of the histogram.
9. **plt.tight\_layout()**:
   * This adjusts the spacing between subplots to ensure that they fit neatly into the figure without overlapping.
10. **plt.show()**:
    * This displays the plot.

**What This Code Does:**

* **Histograms for Each Feature**: The code generates histograms for each feature in the dataset (except for the target variable, such as 'species' in the Iris dataset).
* **Visualization of Distributions**: Each histogram shows the frequency distribution of values in that feature (like petal length, petal width, etc. for Iris).
* **Labeling Bars**: The histogram bars are labeled with the frequency count, making it easier to understand the number of occurrences in each bin.
* **Layout Management**: It arranges all the histograms in a 2x2 grid to make the figure clean and organized.

plt.gca().bar\_label(plt.gca().containers[0], fmt='%d')

What Does This Line Do?

plt.gca():

gca stands for "get current axis". It's a function that returns the current axes (the plotting area) of the plot you are working on.

It gives you access to the figure or plot you're currently modifying.

plt.gca().containers[0]:

containers is a list of all the bars in the plot (for a histogram, the bars are the containers).

[0] accesses the first set of bars (because a histogram is just one plot with one set of bars).

This is how we get the actual bars from the histogram to add labels to them.

bar\_label(...):

This function allows you to add labels to the bars of a plot.

In our case, it adds the count (or number) on top of each bar in the histogram, so you can see how many data points fall into each bin of the histogram.

fmt='%d':

This sets the format of the label.

%d tells it to display the label as an integer (whole number) without any decimal places.

In Simple Words:

plt.gca(): Get the current plot/axes.

containers[0]: Get the bars (histogram bins).

bar\_label(...): Add labels to these bars.

fmt='%d': Make sure the labels are whole numbers (integers).

Code 6 explanation:  
Let's break down and explain this code step-by-step:

**Code:**

for column in df.columns[:-1]: # Exclude 'species' column

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)][column]

print(f"\nFeature: {column}")

print(f" Mean: {df[column].mean():.2f}, Median: {df[column].median():.2f}, Std Dev: {df[column].std():.2f}")

print(f" Outliers Detected: {'Yes' if not outliers.empty else 'No'}","\n " f"Outlier Values: {outliers.tolist()}" if not outliers.empty else "")

print("-" \* 40)

**Explanation:**

1. **for column in df.columns[:-1]:**
   * This loop iterates through each column of the DataFrame (df), excluding the last column, which is the species column.
   * We want to analyze the numerical columns, so we exclude species, which is categorical.
2. **Q1 = df[column].quantile(0.25)**
   * **quantile(0.25)** calculates the **25th percentile (Q1)** of the column's values, i.e., the value below which 25% of the data points fall. This is also known as the first quartile.
3. **Q3 = df[column].quantile(0.75)**
   * **quantile(0.75)** calculates the **75th percentile (Q3)** of the column's values, i.e., the value below which 75% of the data points fall. This is also known as the third quartile.
4. **IQR = Q3 - Q1**
   * **IQR (Interquartile Range)** is the difference between the **third quartile (Q3)** and the **first quartile (Q1)**. It measures the spread of the middle 50% of the data.
   * **IQR** helps us identify the "typical range" of data.
5. **lower\_bound = Q1 - 1.5 \* IQR**
   * **Lower bound** is calculated as the value below which data points are considered "outliers" on the lower end.
   * Any data point that falls below this value is considered an outlier.
6. **upper\_bound = Q3 + 1.5 \* IQR**
   * **Upper bound** is calculated as the value above which data points are considered "outliers" on the upper end.
   * Any data point that falls above this value is considered an outlier.
7. **outliers = df[(df[column] < lower\_bound) | (df[column] > upper\_bound)][column]**
   * This line checks the DataFrame for values that are either:
     + Less than the **lower bound**, or
     + Greater than the **upper bound**.
   * These values are considered "outliers" and are stored in the variable **outliers**.
8. **print(f"\nFeature: {column}")**
   * Prints the name of the feature (or column) currently being processed.
9. **print(f" Mean: {df[column].mean():.2f}, Median: {df[column].median():.2f}, Std Dev: {df[column].std():.2f}")**
   * For each feature, it prints the **mean**, **median**, and **standard deviation** of the column, rounded to two decimal places.
   * **Mean** gives you the average value.
   * **Median** is the middle value, dividing the data into two halves.
   * **Standard deviation (Std Dev)** tells you how spread out the data is from the mean.
10. **print(f" Outliers Detected: {'Yes' if not outliers.empty else 'No'}")**
    * This prints whether any outliers were detected in the column.
    * If **outliers.empty** is False (meaning outliers exist), it will print "Yes", otherwise "No".
11. **print(f" Outlier Values: {outliers.tolist()}" if not outliers.empty else "")**
    * If there are outliers, this prints the **outlier values** found in the column as a list.
    * If there are no outliers, nothing is printed.
12. **print("-" \* 40)**
    * This prints a line of hyphens (-), creating a visual separator between the outputs for different features.

**Summary:**

This code is designed to:

* Calculate and display the **mean**, **median**, and **standard deviation** for each feature (except species) in the Iris dataset.
* Identify if there are **outliers** in each feature using the **Interquartile Range (IQR)** method.
* If outliers are found, it displays the **outlier values** for each feature.
* Finally, it prints the results in a readable format with clear labels.

**Example Output:**

For each feature (e.g., sepal\_length, sepal\_width), you will get something like this:

Feature: sepal\_length

Mean: 5.84, Median: 5.80, Std Dev: 0.83

Outliers Detected: No

----------------------------------------

Feature: sepal\_width

Mean: 3.05, Median: 3.00, Std Dev: 0.43

Outliers Detected: No

----------------------------------------

Feature: petal\_length

Mean: 3.76, Median: 4.35, Std Dev: 1.76

Outliers Detected: Yes

Outlier Values: [1.0]

----------------------------------------

Feature: petal\_width

Mean: 1.20, Median: 1.30, Std Dev: 0.76

Outliers Detected: Yes

Outlier Values: [0.2]

----------------------------------------

Code 7:  
explanation:  
Let's break down the code you've shared and explain each part:

**Code:**

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

for i, column in enumerate(df.columns[:-1]): # Exclude 'species' column

plt.subplot(2, 2, i + 1)

sns.violinplot(x=df["variety"], y=df[column], palette="Set2", hue=df['variety'])

plt.title(f"Violin Plot of {column} by variety")

plt.tight\_layout()

plt.show()

**Explanation:**

1. **plt.figure(figsize=(12, 8))**:
   * This line creates a new figure for plotting. The figsize argument specifies the dimensions of the plot in inches.
   * In this case, the figure will be **12 inches wide** and **8 inches tall**.
2. **for i, column in enumerate(df.columns[:-1]):**:
   * This loop iterates over each **column** of the DataFrame df except for the last one (species or variety column).
   * df.columns[:-1] means all columns except the last one (which is assumed to be the categorical target column, like species or variety).
   * enumerate() helps us keep track of the index (i) and the column name (column).
3. **plt.subplot(2, 2, i + 1)**:
   * plt.subplot() is used to create subplots inside the figure.
   * The parameters (2, 2, i + 1) mean that the figure will have **2 rows** and **2 columns** of subplots (making it a 2x2 grid).
   * i + 1 determines the position of each subplot within the grid (1st, 2nd, 3rd, or 4th subplot).
4. **sns.violinplot(x=df["variety"], y=df[column], palette="Set2", hue=df['variety'])**:
   * sns.violinplot() is used to create a **violin plot**, which is a combination of a box plot and a kernel density plot. It shows the distribution of the data.
   * x=df["variety"]: The x axis will represent the **variety** (assuming variety is the categorical column, like the species or group).
   * y=df[column]: The y axis represents the numerical column (one of the features from the dataset, like sepal\_length, sepal\_width, etc.).
   * palette="Set2": The palette argument defines the color scheme of the plot. "Set2" is a predefined color palette in Seaborn.
   * hue=df['variety']: The hue argument separates the data by different categories (in this case, variety). It ensures that each group in the variety column gets a different color.
5. **plt.title(f"Violin Plot of {column} by variety")**:
   * This sets the title of the current subplot.
   * The f"Violin Plot of {column} by variety" is an **f-string** that dynamically inserts the name of the column being plotted in each subplot.
6. **plt.tight\_layout()**:
   * This adjusts the spacing between subplots to ensure they fit nicely within the figure without overlapping.
7. **plt.show()**:
   * This command displays the plot on the screen after all subplots are created.

**What is a Violin Plot?**

* A **violin plot** is a combination of a **box plot** and a **kernel density plot**. It shows the distribution of numerical data across different categories. It helps visualize:
  + The **probability density** of the data at different values.
  + The **median** and **quartiles** of the data (like a box plot).
  + The **distribution shape** of the data (the width of the plot at each level indicates the density of data points).

**Purpose of This Code:**

* The code is generating **4 subplots** (2 rows, 2 columns) showing **violin plots** of each feature in the dataset (sepal\_length, sepal\_width, petal\_length, petal\_width) by **variety** (the target column).
* Each subplot will have:
  + The **distribution of the feature** for each variety (species).
  + A **comparison of distributions** for each variety (species).
  + The **violin shape** will tell you if there are multiple peaks in the distribution (multimodal), which can give insights into the data distribution.

**Example Visual:**

If the dataset is the **Iris dataset**, the violin plot for sepal\_length might show three "violins" (one for each variety of Iris: setosa, versicolor, and virginica), each with a width that shows how the data points are distributed for that variety.

Let me know if you need more clarification on this!