Lab Session #3

Python libraries

NumPy, Matplotlib, Pandas

Dr Zied M'Nasri &Dr Amr Rashad Abdullatif

Introduction

Python libraries are toolboxes filled with ready-made tools. Instead of building everything from scratch, you can use these tools to solve problems more efficiently. Python libraries cover a vast range of functionalities, from data manipulation to web development, and even artificial intelligence.

Getting Started with Libraries

Before you can use a library, you need to install it. Python comes with a package manager called pip, which you can use to install libraries. For example, to install the popular requests library for making HTTP requests, you would use:

```
pip install requests
```

Once installed, you can import the library into your Python script and start using it.

NumPy

NumPy (Numerical Python) is a fundamental library for scientific computing. It provides support for arrays, matrices, and a wide range of mathematical functions.

Example:

```
import numpy as np

# Create a 1D array

arr = np.array([1, 2, 3, 4, 5])

print("Array:", arr)

# Perform basic operations

print("Sum:", np.sum(arr))

print("Mean:", np.mean(arr))
```

Pandas

Pandas is a powerful library for data manipulation and analysis. It provides data structures like Series and DataFrame, which are perfect for handling structured data.

Example:

```
import pandas as pd

# Create a DataFrame

data = {'Name': ['Alice', 'Bob', 'Charlie'],

        'Age': [25, 30, 35]}

df = pd.DataFrame(data)


print("DataFrame:")

print(df)
```

Matplotlib

Matplotlib is a library for creating static, animated, and interactive visualizations in Python. It's especially useful for creating graphs and charts.

Example:

```
import matplotlib.pyplot as plt

# Simple line plot

x = [1, 2, 3, 4, 5]

y = [10, 20, 25, 30, 40]

plt.plot(x, y)

plt.title("Simple Line Plot")

plt.xlabel("X Axis")

plt.ylabel("Y Axis")

plt.show()
```

Requests

The requests library is used to send HTTP requests in Python. It simplifies interacting with web services and APIs.

Example:

```
import requests

# Make a GET request

response = requests.get('https://api.github.com')

# Print response content

print(response.text)
```

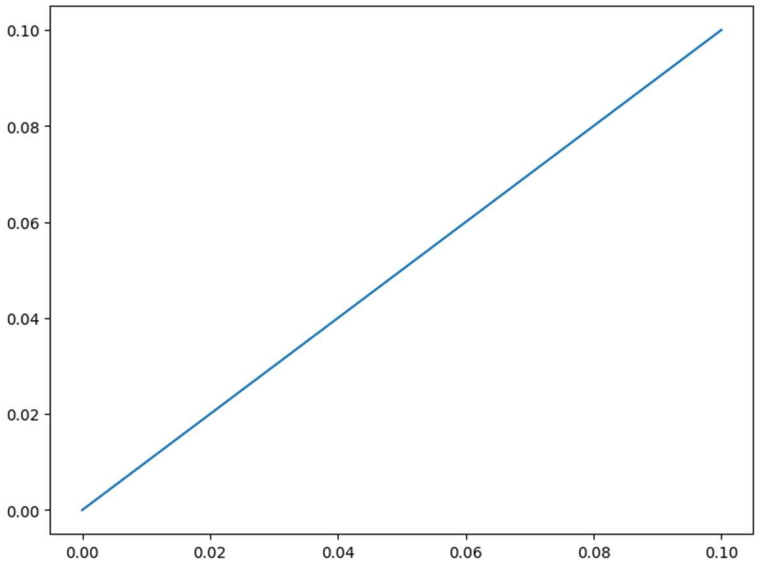
Exercise 1- Creating a Custom Axes Plot with Matplotlib

In this exercise, we will create and customize axes in a figure using matplotlib. We will manually specify the size and position of an axes object inside a figure using normalized coordinates.

Instructions

1. **Import the necessary libraries**:
    a. matplotlib.pyplot for plotting.
    b. numpy for numerical operations (if needed).
2. **Create a new figure** using plt.figure().
3. **Add a set of axes** to the figure using the add_axes() method.
4. Use the full figure area by specifying the bounding box as [0, 0, 1, 1].
5. The values represent: [left, bottom, width, height], all in normalized (0 to 1) units.
6. **Plot a simple line** with the following data points:
7. X values: [0, 0.05, 0.1]
8. Y values: [0, 0.05, 0.1]
9. **Display the plot**.

Expected result



10. Modify the axes to occupy only a part of the figure by changing the bounding box to [0.1, 0.1, 0.8, 0.8]. Observe how the plot's position and size change.

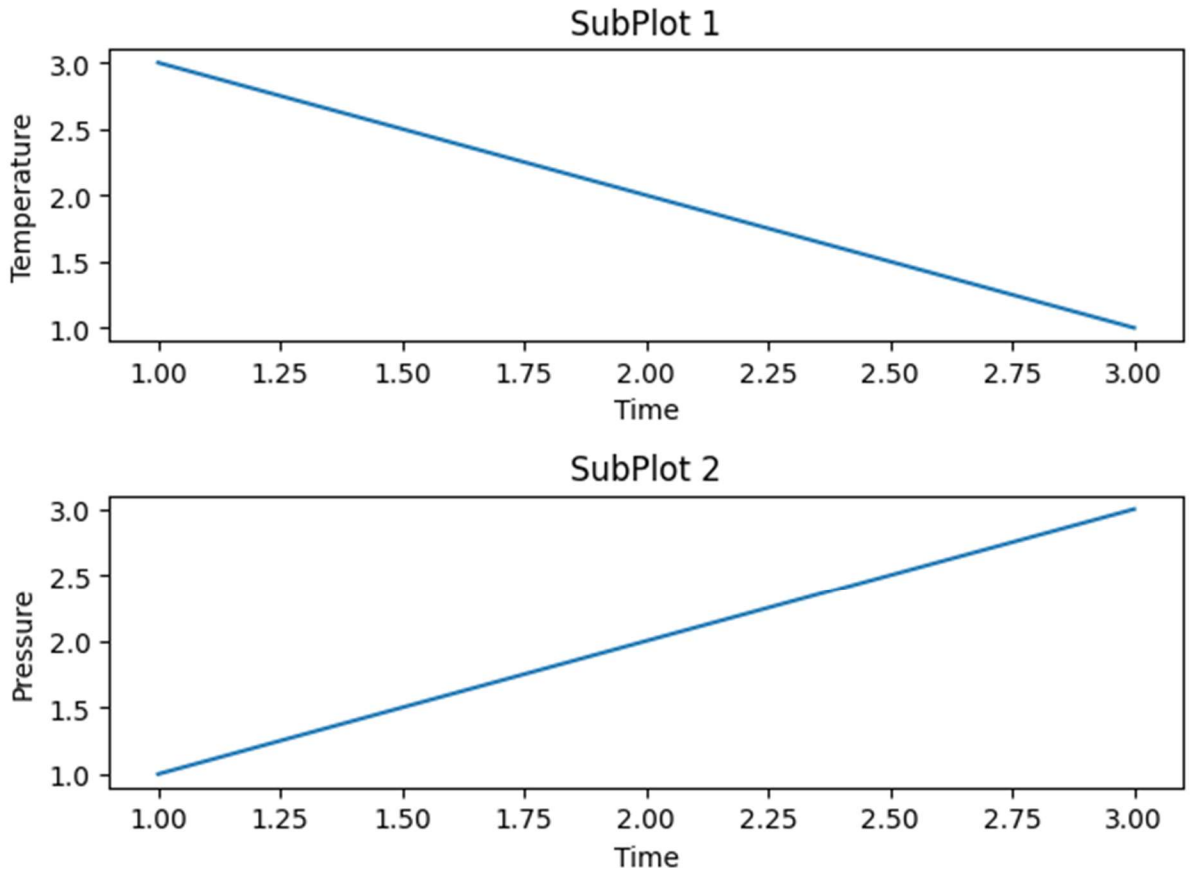
Exercise 2: Creating Subplots with Shared Layout in Matplotlib

Use **matplotlib.pyplot.subplots()** to create two vertically stacked plots, each showing a different trend in data.

1. **Import the required libraries**:
    - matplotlib.pyplot as plt.
2. **Create a figure and a set of subplots**:
    - Use **plt.subplots(rows, columns)** to create **2 rows** and **1 column** of subplots.
    - Store the result in variables fig and axs.
3. **First subplot (top)**:
    - Plot the points [1, 2, 3] on the x-axis and [3, 2, 1] on the y-axis.
    - Set the y-axis label to 'Time' using the method set_xlabel()
    - Set the y-axis label to 'Temperature' set_ylabel()
4. **Second subplot (bottom)**:
    - Plot the points [1, 2, 3] on the x-axis and [1, 2, 3] on the y-axis.
    - Set the y-axis label to 'Time'.
    - Set the y-axis label to ‘Pressure'.

5. **Adjust the layout**:
    - Use fig.tight_layout() to prevent overlapping of subplot elements.
6. **Display the figure** with plt.show().
7. Add titles to each subplot using the method set_title()

Expected result



Exercise 3: Plotting a Noisy Sine Function with Randomness

Create a plot of a mathematical function that includes a sine component, a linear term, and a random variation. This simulates noisy or real-world data.

1. Import the necessary modules:
    - math for trigonometric functions.
    - random for generating random numbers.
    - matplotlib.pyplot as plt for plotting.
2. Define a custom function myfun(x) that returns:

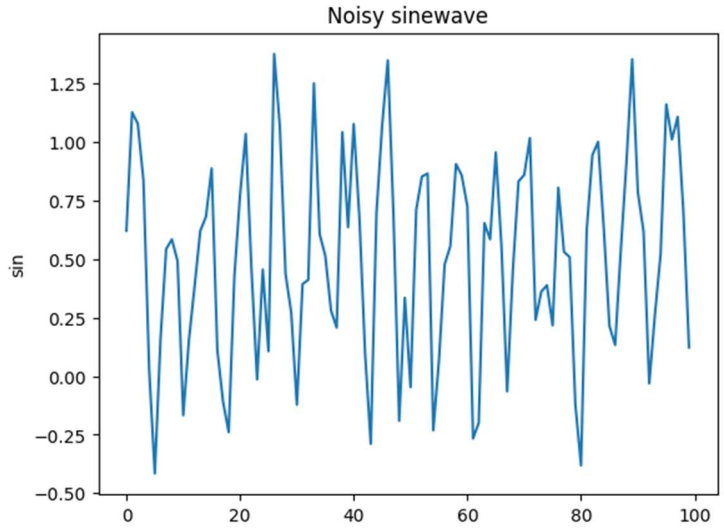
$0.5 \times \sin(x) + noise$ where noise is a random number between 0 and 1

Use:

- math.sin(x) for the sine part.
- Use random.uniform(0, 1) to add some randomness (noise).
- Combine all parts to return the final value.

3. Generate x-values:
    - Create a list xx of integers from 0 to 99.
4. Compute y-values:
    - Use a list comprehension to apply myfun(x) to each x-value in xx.
    - Store the results in a list called yy.
5. Plot the data:
    - Use the method plt.plot(xx, yy) to visualize the result.
    - Add a y-axis label 'sin' using plt.ylabel().
    - Add a title ‘Noisy Sinewave’ using the method plt.title()
    - Display the plot using plt.show().

Expected Output



6. Remove the term noise from the equation. What do you notice?

Exercise4 : Arrays and Statistics with NumPy

In this exercise, we will:

- Create a random 2x2 NumPy array.
- Compute the overall mean and variance.
- Compute the **mean across rows** and **columns** using axis.
- Compute the variance and the standard deviation for the entire array

1. **Import NumPy** as np.
2. **Create a 2x2 array** called my_array filled with **random values between 0 and 1**. You can Use np.random.random((2, 2)) for this.
3. **Print the array** using print(my_array).
4. **Calculate and print** the following statistics:
    - The **overall mean** of all elements in the array, use np.mean(my_array)
    - The **mean of each column** (i.e., mean across rows), using np.mean(my_array, axis=0)
    - The **mean of each row** (i.e., mean across columns).
        - Use np.mean(my_array, axis=1)
    - The **overall variance** of the array.
        - Use np.var(my_array)
    - The standard deviation of the array
        - Use np.std(my_array)

Expected output

```
[[0.20378908 0.31503921]
 [0.46865308 0.78288247]]
mean  0.44259096183540947
row_mean  [0.33622108 0.54896084]
column_mean  [0.25941415 0.62576778]
variance 0.04744333311437099
Standard deviation 0.21781490562946096
```

Exercise 5: Visualizing Stacked Debt Growth Over Time

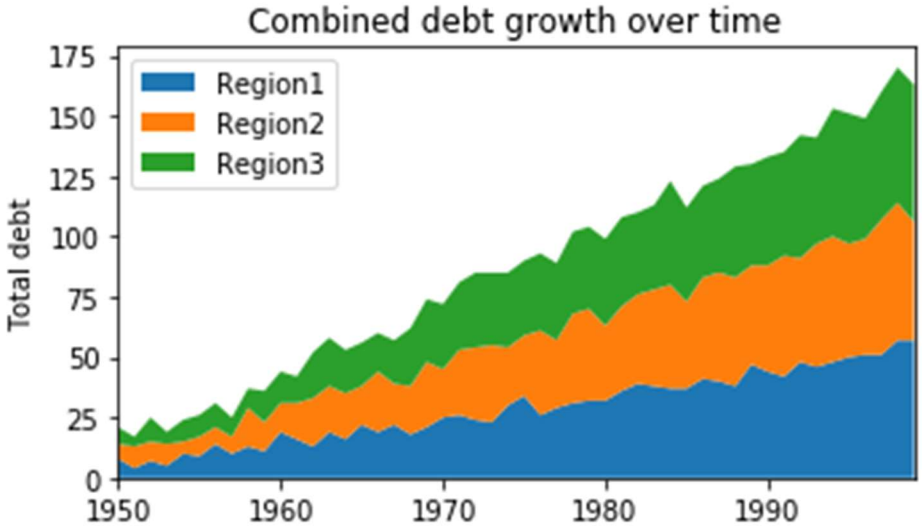
In this exercise, we aim to generate synthetic data to simulate debt growth across multiple regions over a span of 50 years. Then we will create a stacked area plot using Matplotlib to visualize how the total debt evolves over time for each region. This requires:

- Creating a time series using np.arange().
- Generating random data using np.random.randint().
- Using plt.stackplot() to visualize stacked data.
- Customizing and labeling the plot for clarity.

1. Import the required libraries:
    - numpy as np
    - matplotlib.pyplot as plt
2. Generate the data:
    - Create a range of 50 years using np.arange(50) and store it in rng.
    - Create a 3×50 matrix of random integers between 0 and 10 using np.random.randint(), and store it in rnd.
    - Create a list of years starting from 1950 by adding rng to 1950. Store this in yrs.
3. Create the plot:
    - Use plt.subplots() with a figsize of (5, 3).
    - Use ax.stackplot() to plot the cumulative data over time.
        - Stack the data as rng + rnd to simulate increasing debt over time.
        - Use labels ['Region1', 'Region2', 'Region3'].
4. Customize the plot:
    - Set the plot title to 'Combined debt growth over time'.
    - Add a legend in the upper left corner.
    - Label the y-axis as 'Total debt'.
    - Set the x-axis limits to span from the first to the last year using ax.set_xlim().
5. Use fig.tight_layout() to ensure the layout fits nicely.
6. Display the plot using plt.show().

Expected Output:

A stacked area chart showing the cumulative growth of "debt" (simulated) from 1950 to 1999 for three regions, with distinct colours, a legend, and proper axis labels.



Exercise 6 : Exploring and Visualizing a Simple DataFrame

In this exercise, we'll create a small dataset using pandas and perform some basic exploratory data:

- Create a pandas DataFrame from a dictionary.
- Explore the dataset using .info(), .mean(), and .describe().
- Plot histograms and scatter plots to understand the distribution and relationships in the data.

1. **Import the necessary libraries**:

- pandas as pd
- matplotlib.pyplot as plt

2. **Create a DataFrame** named df with the following data:

```
'c1': [1, 2, 3, 1, 2 ,10]
'c2':[1, 4, 3, 2, 4 ,22]
```

2.1. Print the dataframe

```
   c1  c2
0   1   1
1   2   4
2   3   3
3   1   2
4   2   4
5  10  22
```

3. **Explore the DataFrame:**
   - Use df.info() to display column data types and non-null counts.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 2 columns):
c1    6 non-null int64
c2    6 non-null int64
dtypes: int64(2)
memory usage: 224.0 bytes
```

   - Use df.mean() to compute the mean of each column.

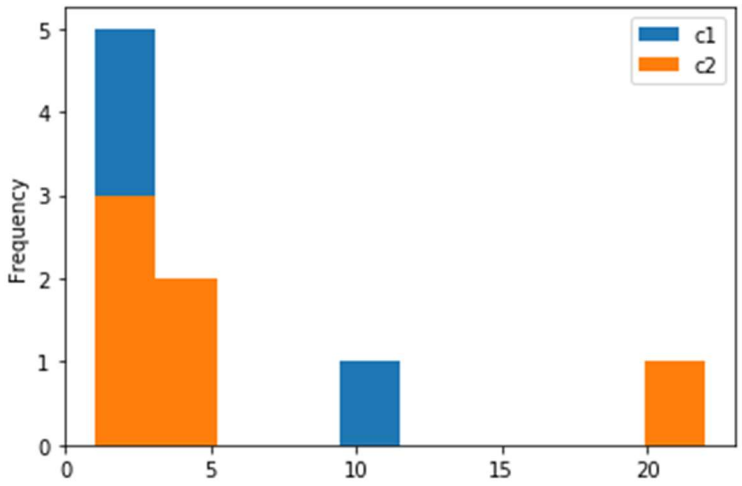
```
c1    3.1667
c2    6.0000
dtype: float64
```

   - Use df.describe() to view summary statistics (count, mean, std, min, max, etc.)

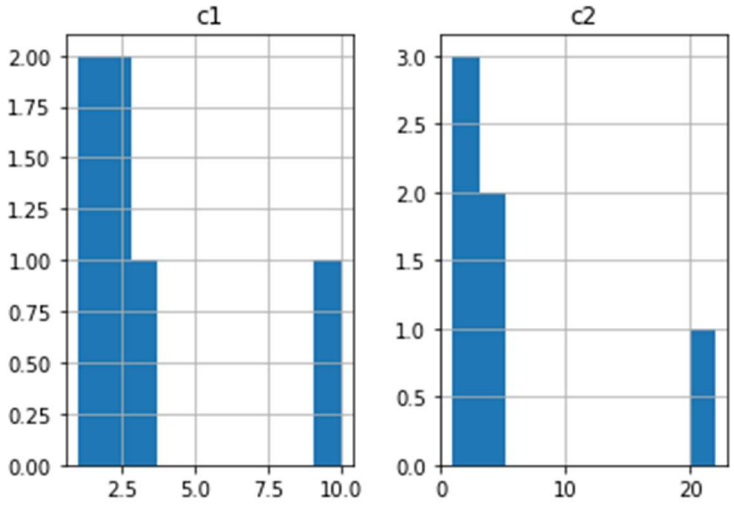
```
       c1         c2
count 6.0000     6.0000
mean  3.1667     6.0000
std   3.4303     7.9246
min   1.0000     1.0000
25%   1.2500     2.2500
50%   2.0000     3.5000
75%   2.7500     4.0000
max   10.0000    22.0000
```

4. **Visualize the data**:

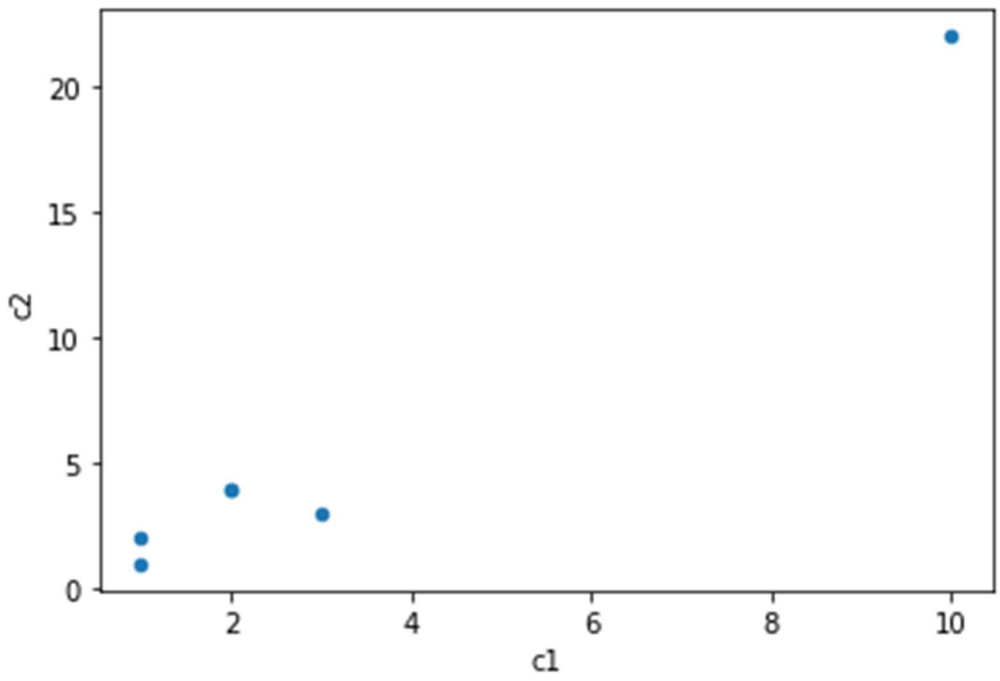
- Plot a **histogram** of all columns using df.plot.hist(bins=10).



- Use plt.show() to display the plots.



- Use df.plot.scatter(x='c1', y='c2') to produce a scatter plot to visualize the relationship between columns 'c1' and 'c2':



Exercise 7: Analyzing and Visualizing Chocolate Distribution Data

In this exercise, we will explore a real dataset (Celebrations_Data.xlsx) containing information about chocolate distributions. We will learn how to load data, check for missing values, analyze distributions, and visualize correlations using pandas, matplotlib, seaborn, and plotly.

Tasks

- Load Excel data with pandas
- Perform basic data inspection and preprocessing
- Visualize data distributions (bar chart, violin plot, density plots)
- Compute and visualize correlations with a heatmap

Steps

1. Set up the enviorment Environment
   - Import the necessary libraries:
     - os
     - re
     - seaborn as sns
     - matplotlib.pyplot as plt
     - pandas as pd
     - numpy as np
     - warnings
     - plot, iplot, init_notebook_mode from plotly.offline
     - check_output from subprocess import
     - core.display import display, HTML from IPython.

Configure display settings (pandas, warnings, plotly, and Jupyter style) using:

- pd.set_option to set:
  - 'display.max_columns' to 100
  - 'display.max_rows' to 100
  - 'precision' to 4
- warnings.simplefilter('ignore')
- init_notebook_mode()
- display(HTML("<style>.container { width:100% !important; }</style>"))
- Enable inline plotting (for Jupyter notebooks) using: %matplotlib inline

2. Load and Inspect the Dataset

   Load data from the Excel file Celebrations Data.xlsx, sheet 'RawDataTub' into a DataFrame called df_sweets using pd.read_excel()

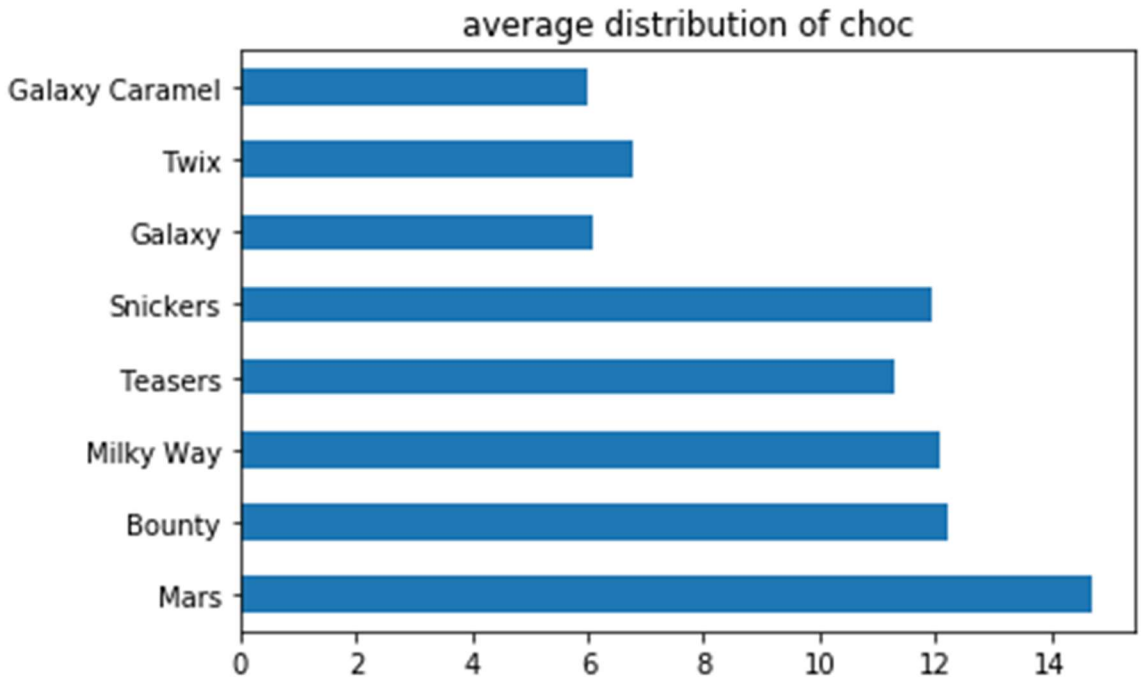
3. Print: The **shape** (or size) of the DataFrame using the method dataFrameName.shape(), then print the **first few rows and t**he **data types using the method dataFrameName.head()**
4. Print the data type using the attribute dataFrameName.dtypes
5. Print the **value counts** of the column 'Person Supplying Data' using

   dataFrameName. loc[:, 'Person Supplying Data'].value_counts()

6. Check the missing values using the method pd.isnull(dataFrameName).sum()
7. Select Only Chocolate-Related Columns using select_dtypes(include='int64') to isolate numeric columns (representing chocolate counts), excluding the last one.
8. Visualize Average Distribution of Chocolates

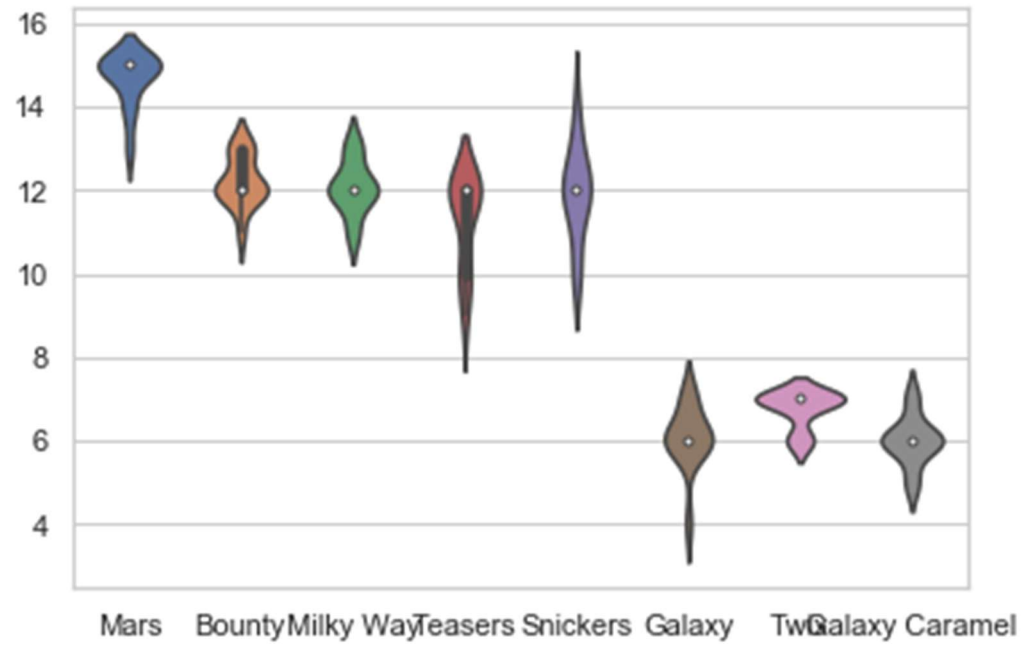
   Plot the mean count of each chocolate type as a horizontal bar chart.

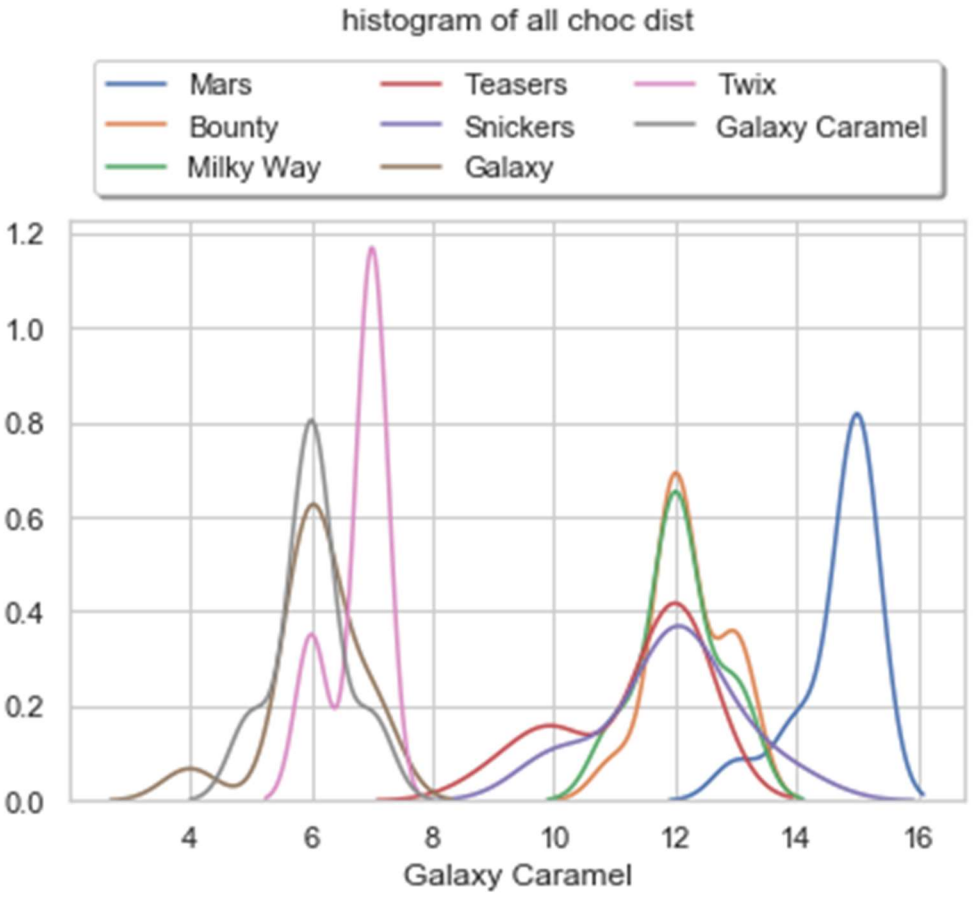
   Add a title: "average distribution of choc"



9. Visualize Chocolate Distributions

- Create a **violin plot** for the chocolates DataFrame.
- Overlay **density plots** for each chocolate type using sns.distplot().
- Customize:
  - Add a legend
  - Set the plot title: "histogram of all choc dist"
  - Position the legend above the plot


16
14
12
10
8
6
4
Mars
Bounty
Milky Way
Teasers
Snickers
Galaxy
Twix
Galaxy Caramel


histogram of all choc dist
Mars
Bounty
Milky Way
Teasers
Snickers
Galaxy
Twix
Galaxy Caramel
1.2
1.0
0.8
0.6
0.4
0.2
0.0
4
6
8
10
12
14
16
Galaxy Caramel

10. Correlation Analysis

Compute the correlation matrix for the chocolate types using .corr().

	Mars	Bounty	Milky Way	Teasers	Snickers	Galaxy	Twix	Galaxy Caramel
Mars	1.0000	0.2037	-0.1429	0.0275	0.4377	0.4016	0.3246	-0.2290
Bounty	0.2037	1.0000	-0.4844	-0.2411	0.1535	0.1409	-0.0976	0.0000
Milky Way	-0.1429	-0.4844	1.0000	0.6676	0.4756	0.3294	-0.2282	0.6760
Teasers	0.0275	-0.2411	0.6676	1.0000	0.4922	0.4640	-0.3558	0.6505
Snickers	0.4377	0.1535	0.4756	0.4922	1.0000	0.3027	-0.5505	0.3883
Galaxy	0.4016	0.1409	0.3294	0.4640	0.3027	1.0000	0.0577	0.3801
Twix	0.3246	-0.0976	-0.2282	-0.3558	-0.5505	0.0577	1.0000	-0.3291
Galaxy Caramel	-0.2290	0.0000	0.6760	0.6505	0.3883	0.3801	-0.3291	1.0000

Create a heatmap of the correlations using sns.heatmap()