SHADOW FOX VIRTUAL INTERNSHIP

DATA SCIENCE : BEGINNER LEVEL

Report given by SAKSHI

OBJECTIVE:

Create a comprehensive documentation guide for 2 of the following Python visualisation libraries: Matplotlib,Seaborn, Plotly, Bokeh, and Pandas. Your guide should focus on the variety of graphs each library can generate and include practical examples with code snippets.

VISUALISATION LIBRARIES : 1) MATPLOTLIB

2) SEABORN

DATASET USED :

Air Quality Index (AQI) ; given in the task sheet.

GOOGLE COLAB LINK :

https://colab.research.google.com/drive/1kkRTQD-2IuHQAH7Wh_-UCLSfBAvArmeN?usp=sharing

INTRODUCTION

Data visualization in data science refers to the graphical representation of data to visually communicate insights and patterns inherent in the data. It is a critical component of data analysis and plays a crucial role in both the exploratory and explanatory phases of data science.

Importance:

Effective data visualization enhances the understanding of complex datasets, facilitates communication across interdisciplinary teams, and empowers decision-makers to derive actionable insights. In data science, it serves as a bridge between data analysis and decision-making, enabling stakeholders to leverage data-driven strategies for business, research, and policy-making purposes.

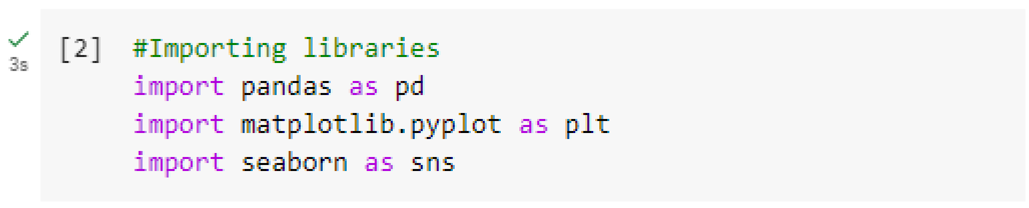
In summary, data visualization is a fundamental aspect of data science that transforms raw data into meaningful visual representations, facilitating exploration, pattern recognition, insight communication, and model evaluation. It empowers data scientists to uncover hidden insights and drive informed decisions based on data-driven evidence.

MATPLOTLIB

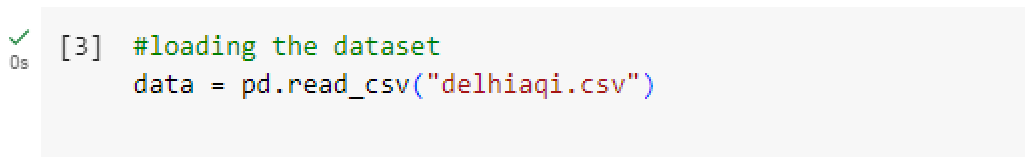
- Matplotlib is one of the most popular plotting libraries in Python and provides a MATLAB-like interface for creating static, animated, and interactive visualizations.
- It offers fine-grained control over nearly every aspect of a plot, making it suitable for a wide range of visualization needs.
- Matplotlib can be imported using the `import` statement
- *Basic Plotting*: Matplotlib allows you to create various types of plots such as line plots, scatter plots, bar plots, histograms, and more.
- *Customization:* Matplotlib provides extensive customization options for plots including labels, titles, legends, colors, markers, and more. You can customize almost every aspect of the plot to suit your requirements.
- We can save your Matplotlib plots as image files directly in Google Colab using plot.png function.
- Different plots or charts that can be created using matplotlib in google colab are given below:-
    - ☐ HISTOGRAM
    - ☐ LINE PLOT
    - ☐ BAR CHART
    - ☐ PIE CHART
    - ☐ SCATTER PLOT
    - ☐ 3D PLOT

Now, we will understand the use and implementation of above charts using code snippets and outputs.

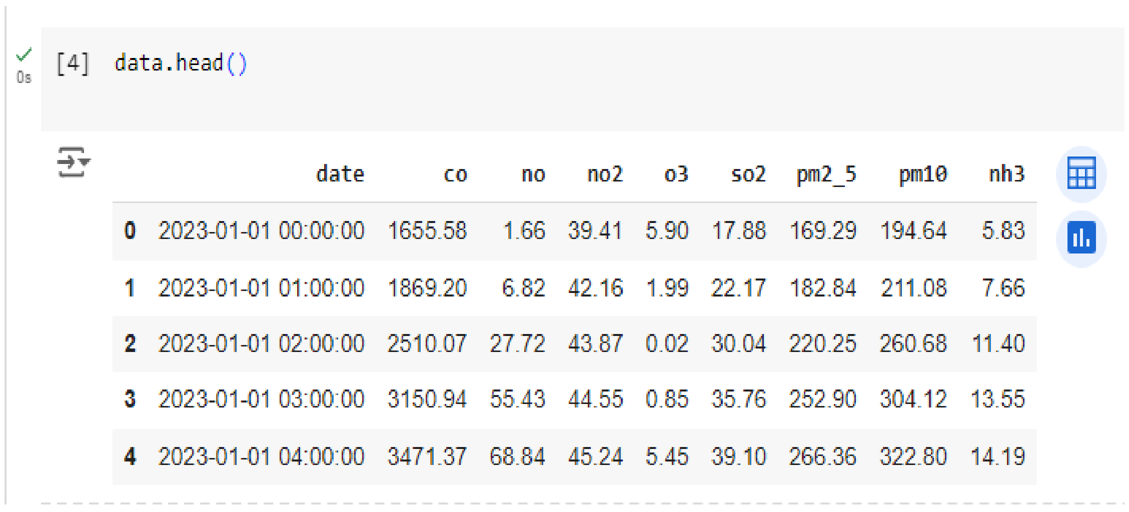
Let's begin by importing the python libraries and performing basic EDA, i.e, Exploratory Data Analysis



```
[2] #Importing libraries
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
```

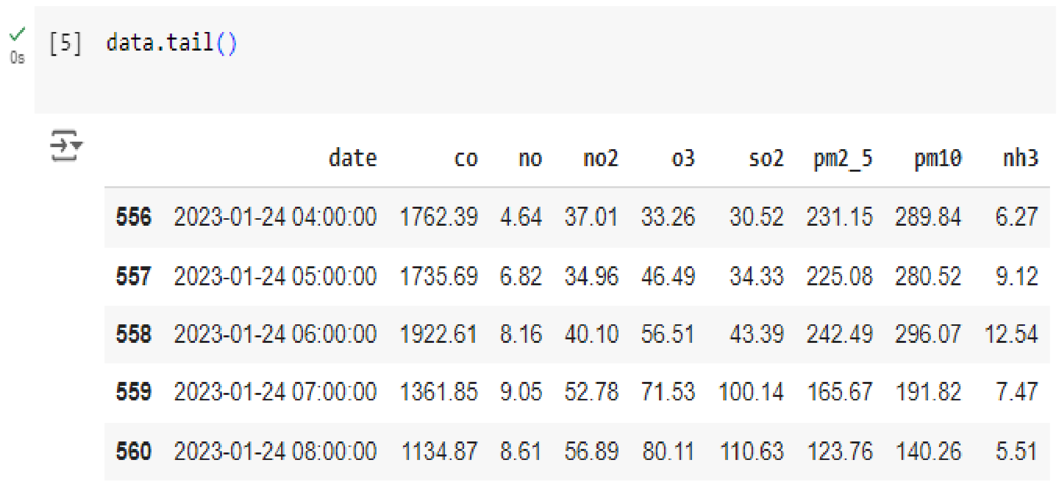


```
[3] #loading the dataset
    data = pd.read_csv("delhiaqi.csv")
```



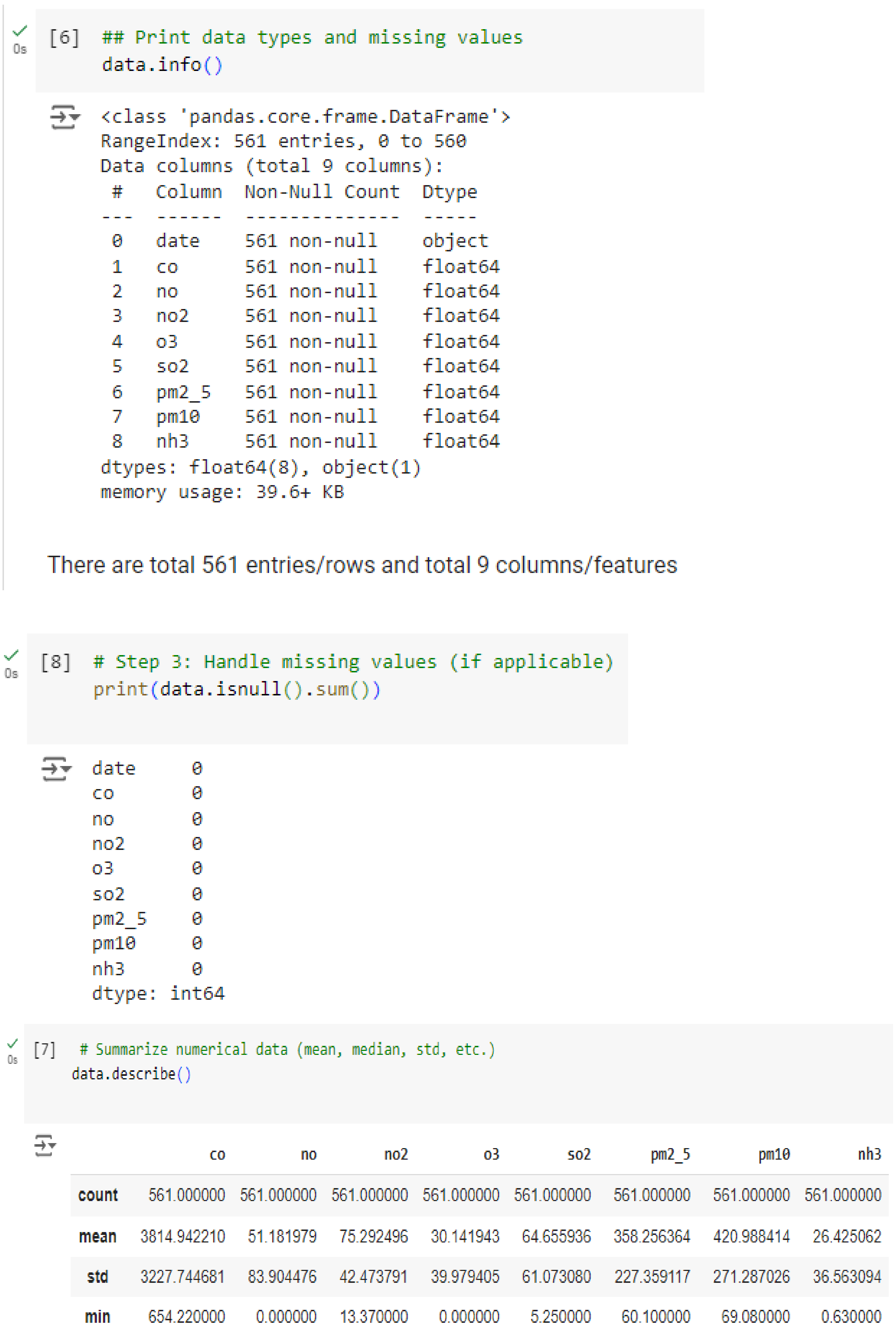
```
[4] data.head()
```

	date	co	no	no2	o3	so2	pm2_5	pm10	nh3
0	2023-01-01 00:00:00	1655.58	1.66	39.41	5.90	17.88	169.29	194.64	5.83
1	2023-01-01 01:00:00	1869.20	6.82	42.16	1.99	22.17	182.84	211.08	7.66
2	2023-01-01 02:00:00	2510.07	27.72	43.87	0.02	30.04	220.25	260.68	11.40
3	2023-01-01 03:00:00	3150.94	55.43	44.55	0.85	35.76	252.90	304.12	13.55
4	2023-01-01 04:00:00	3471.37	68.84	45.24	5.45	39.10	266.36	322.80	14.19



```
[5] data.tail()
```

	date	co	no	no2	o3	so2	pm2_5	pm10	nh3
556	2023-01-24 04:00:00	1762.39	4.64	37.01	33.26	30.52	231.15	289.84	6.27
557	2023-01-24 05:00:00	1735.69	6.82	34.96	46.49	34.33	225.08	280.52	9.12
558	2023-01-24 06:00:00	1922.61	8.16	40.10	56.51	43.39	242.49	296.07	12.54
559	2023-01-24 07:00:00	1361.85	9.05	52.78	71.53	100.14	165.67	191.82	7.47
560	2023-01-24 08:00:00	1134.87	8.61	56.89	80.11	110.63	123.76	140.26	5.51



```
[6]  ## Print data types and missing values
     data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 561 entries, 0 to 560
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    561 non-null    object
 1   co      561 non-null    float64
 2   no      561 non-null    float64
 3   no2     561 non-null    float64
 4   o3      561 non-null    float64
 5   so2     561 non-null    float64
 6   pm2_5   561 non-null    float64
 7   pm10    561 non-null    float64
 8   nh3     561 non-null    float64
dtypes: float64(8), object(1)
memory usage: 39.6+ KB
```

There are total 561 entries/rows and total 9 columns/features

```
[8]  # Step 3: Handle missing values (if applicable)
     print(data.isnull().sum())
```

```
date     0
co       0
no       0
no2      0
o3       0
so2      0
pm2_5    0
pm10     0
nh3      0
dtype: int64
```

```
[7]  # Summarize numerical data (mean, median, std, etc.)
     data.describe()
```

	co	no	no2	o3	so2	pm2_5	pm10	nh3
count	561.000000	561.000000	561.000000	561.000000	561.000000	561.000000	561.000000	561.000000
mean	3814.942210	51.181979	75.292496	30.141943	64.655936	358.256364	420.988414	26.425062
std	3227.744681	83.904476	42.473791	39.979405	61.073080	227.359117	271.287026	36.563094
min	654.220000	0.000000	13.370000	0.000000	5.250000	60.100000	69.080000	0.630000

PLOTTING OF DIFFERENT CHARTS/PLOTS IN MATPLOTLIB

HISTOGRAM

Definition:

A histogram is a graphical representation of the distribution of numerical data. It consists of bars whose heights indicate the frequency of observations within each bin.

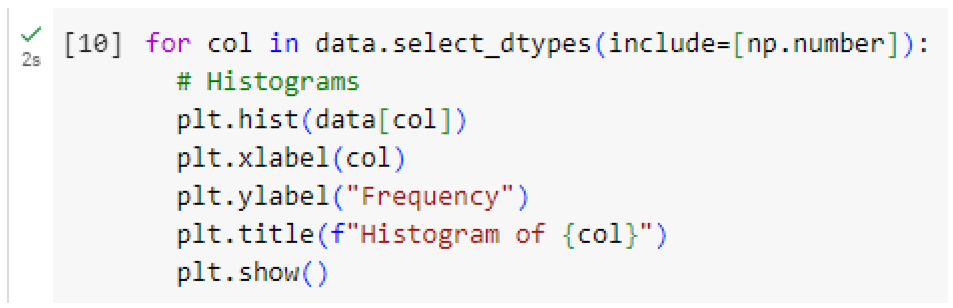
Uses:

- Visualize the distribution (spread and skewness) of continuous data.
- Identify the central tendency and spread of data.
- Detect outliers and anomalies.

Benefits:

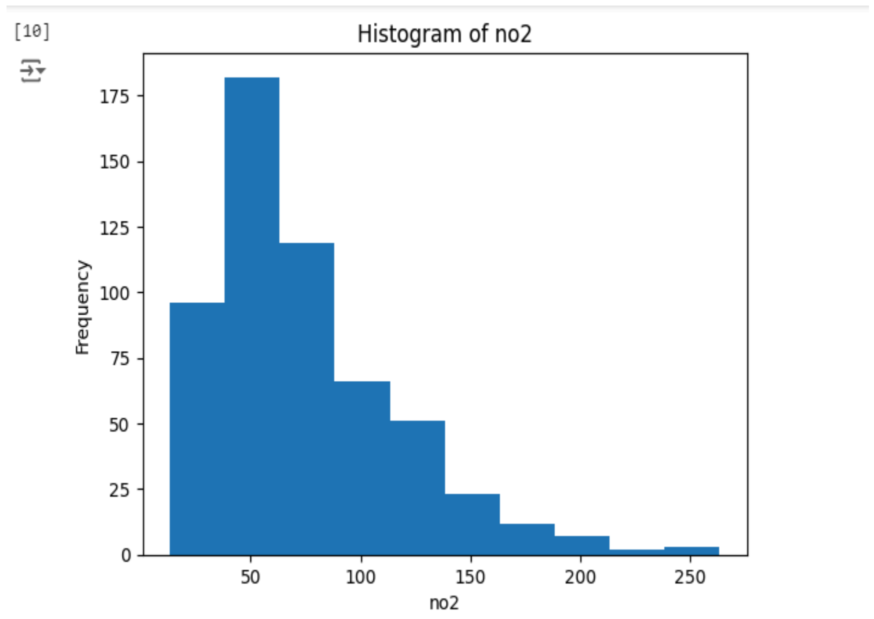
- Provides a quick summary of data distribution.
- Helps in understanding data range and variability.
- Useful for data preprocessing and outlier detection.

CODE IMPLEMENTATION:



```
[10] for col in data.select_dtypes(include=[np.number]):
        # Histograms
        plt.hist(data[col])
        plt.xlabel(col)
        plt.ylabel("Frequency")
        plt.title(f"Histogram of {col}")
        plt.show()
```

OUTPUT:



LINE PLOT

Definition:

A line plot displays data points connected by straight line segments. It is particularly useful for visualising trends over time or continuous data.

Uses:

- Show trends or patterns in data series.
- Illustrate changes in data values over time or across categories.
- Compare multiple datasets or variables with a common x-axis.

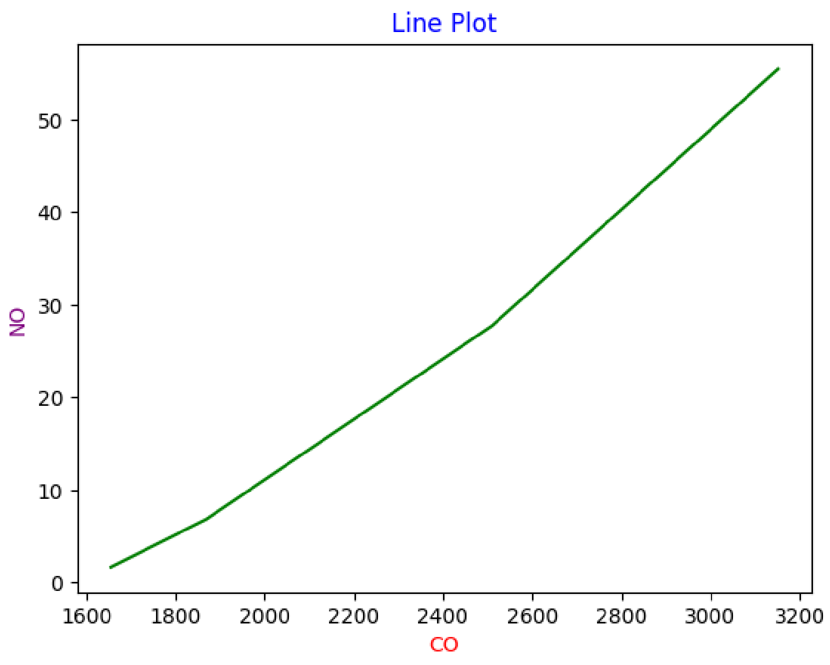
Benefits:

- Provides a clear representation of trends and patterns in data.
- Facilitates easy comparison of multiple variables or datasets.
- Useful for forecasting, time series analysis, and trend identification.

CODE IMPLEMENTATION:

```
x = data["co"].tolist()[0:4]
y = data["no"].tolist()[0:4]
#plotting after assigning the values to variable x and y
plt.plot(x,y, color = 'green')
plt.xlabel("CO", color = 'red')
plt.ylabel("NO", color = 'purple')
plt.title("Line Plot", color = 'blue')
plt.show()
```

OUTPUT:



3D PLOT

Definition:
A 3D plot is a graphical representation of three-dimensional data points. It uses three axes (x, y, z) to represent three variables.

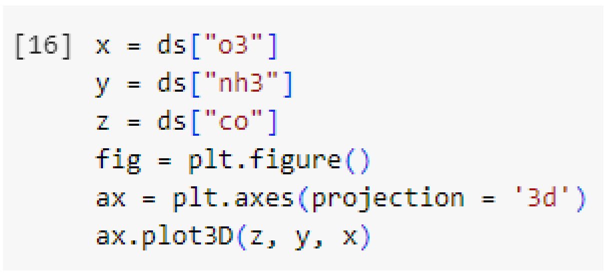
Uses:

- Visualize relationships between three variables simultaneously.
- Represent complex datasets with multiple dimensions.
- Provide depth and spatial perspective to data visualization.

Benefits:

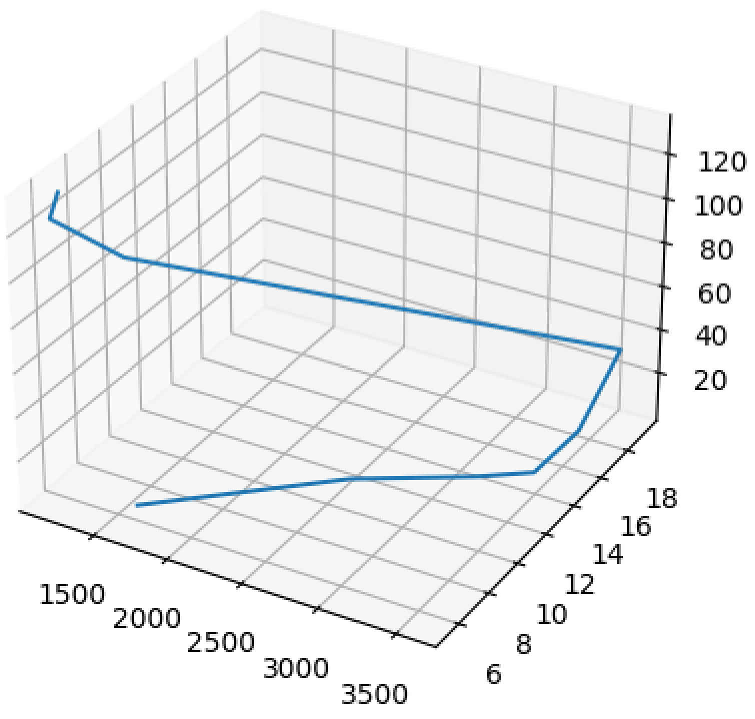
- Enhances understanding of complex spatial relationships in data.
- Allows for interactive exploration and rotation of data points.
- Useful for scientific visualization and engineering applications.

CODE IMPLEMENTATION:



```
[16] x = ds["o3"]
     y = ds["nh3"]
     z = ds["co"]
     fig = plt.figure()
     ax = plt.axes(projection = '3d')
     ax.plot3D(z, y, x)
```

OUTPUT:



SCATTER PLOT

Definition:
A scatter plot uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis corresponds to its respective values.

Uses:

- Visualize relationships or correlations between two continuous variables.
- Identify patterns, clusters, or trends in data points.
- Assess the strength and direction of relationships (positive, negative, or none).

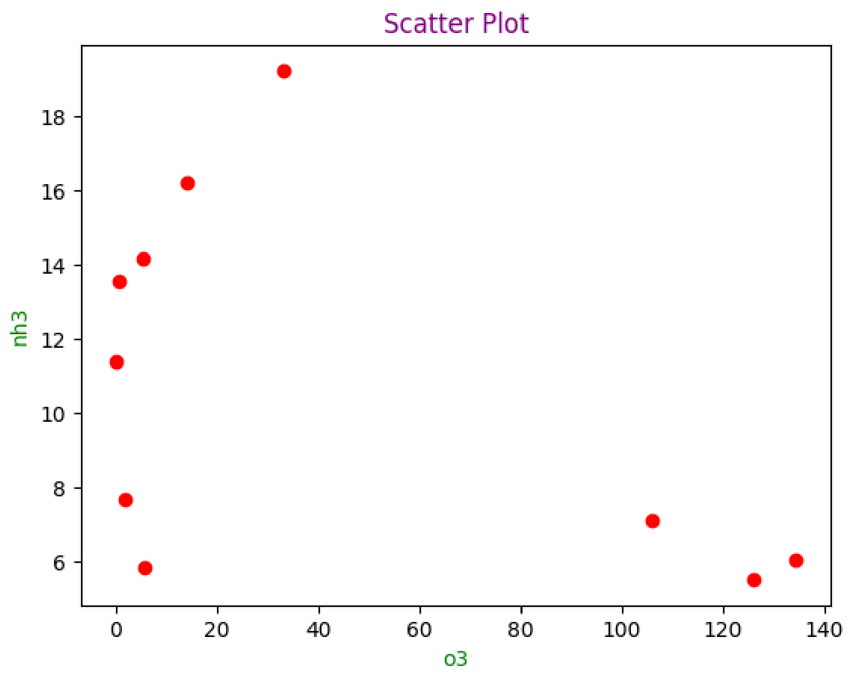
Benefits:

- Facilitates pattern recognition and outlier detection.
- Helps in identifying clusters or groups within data.
- Useful for exploring and visualizing multivariate relationships.

CODE IMPLEMENTATION:

```
x = ds["o3"]
y = ds["nh3"]
plt.scatter(x,y, color='red')
plt.xlabel("o3", color='green')
plt.ylabel("nh3", color='green')
plt.title("Scatter Plot", color = 'purple')
plt.show()
```

OUTPUT:



PIE CHART/ PLOT

Definition:
A pie chart is a circular statistical graphic divided into slices to illustrate numerical proportion. Each slice represents a category's contribution to a whole.

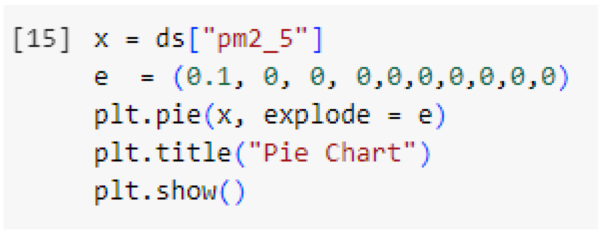
Uses:

- Show the composition of a categorical variable as parts of a whole.
- Highlight relative proportions or percentages.
- Emphasize the largest or smallest categories within a dataset.

Benefits:

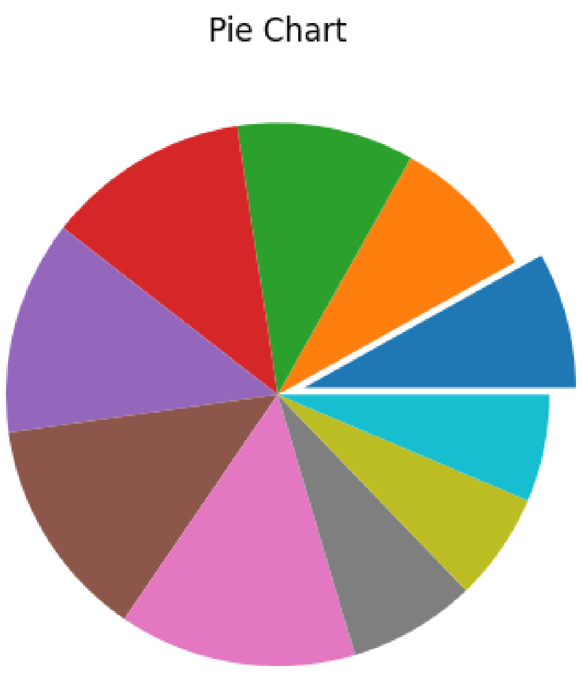
- Provides an intuitive representation of percentages or proportions.
- Easily identifies major contributors or outliers in a dataset.
- Useful for presenting data distributions in a visually appealing manner.

CODE IMPLEMENTATION:



```
[15] x = ds["pm2_5"]
     e  = (0.1, 0, 0, 0,0,0,0,0,0,0)
     plt.pie(x, explode = e)
     plt.title("Pie Chart")
     plt.show()
```

OUTPUT:



SEABORN

- Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics.

- It simplifies the creation of complex visualisations such as categorical plots, distribution plots, and matrix plots

- .**Enhanced Aesthetics**: Seaborn enhances Matplotlib plots with improved aesthetics by providing themes and colour palettes that are more visually appealing and easier to interpret.

- **Statistical Plotting**: Seaborn includes specialised functions for visualising statistical relationships such as scatter plots with regression lines, box plots, violin plots, and pair plots.

- **Flexible Categorical Plots**: Seaborn provides functions to create categorical plots like bar plots, count plots, and categorical scatter plots, which are useful for visualising distributions across categories.

- **Matrix Plots**: Seaborn allows the creation of matrix plots such as heatmaps and cluster maps, which are helpful for visualising relationships between variables in large datasets.

- ***As Seaborn compliments and enhances Matplotlib, the learning curve is quite gradual. If we are familiar with Matplotlib, we are already halfway done with Seaborn***

.

Seaborn divides its plotting functions into several categories based on the type of data and the kind of visualisation you want to create. These categories help organise the wide range of plotting capabilities offered by Seaborn:

1. **Relational Plots**:
   - These plots emphasise the relationship between variables. Examples include scatter plots, line plots, and relational heatmaps (`sns.scatterplot()`, `sns.lineplot()`, `sns.relplot()`).

2. **Categorical Plots**:
   - These plots summarize categorical data or show the relationship between numerical and categorical variables. Examples include bar plots, count plots, and box plots (`sns.barplot()`, `sns.countplot()`, `sns.boxplot()`).

   Distribution Plots:

   - These plots visualize the distribution of a dataset. Examples include histograms, kernel density estimation (KDE) plots, and rug plots (`sns.histplot()`, `sns.kdeplot()`, `sns.rugplot()`).

3. **Regression Plots**:
   - These plots fit and visualize linear regression models for exploring relationships between variables. Examples include scatter plots with regression lines and residplots (`sns.regplot()`, `sns.lmplot()`, `sns.residplot()`).

4. **Matrix Plots**:
   - These plots visualize matrices of data. Examples include heatmaps for correlation matrices and cluster maps (`sns.heatmap()`, `sns.clustermap()`).

5. **Multi-plot Grids**:
   - These plots allow you to create multiple plots that are organized in a grid-like fashion, which can be useful for comparing different aspects of the data (`sns.FacetGrid()`, `sns.PairGrid()`).

6. **Style and Color Controls**:
   - Seaborn provides functions to control the aesthetics of plots, such as colors, styles, and palettes (`sns.set_style()`, `sns.set_palette()`).

Different plots or charts that can be created using seaborn in google colab are given below:-

- ☐ COUNT PLOT
- ☐ PAIRPLOT
- ☐ STRIP PLOT
- ☐ HEATMAP
- ☐ BOX PLOT
- ☐ VIOLIN PLOT

PLOTTING OF DIFFERENT CHARTS/PLOTS USING SEABORN LIBRARY

COUNT PLOT

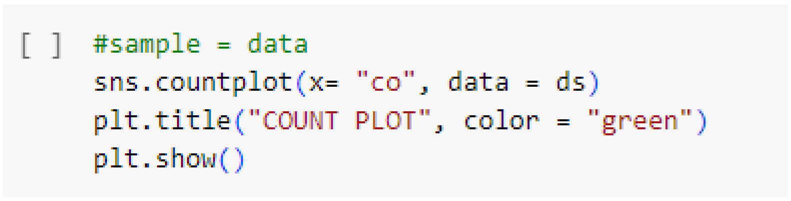
- **Definition**:

  A count plot is a categorical plot that shows the count of observations in each categorical bin using bars.

- **Uses**:
  - Visualise the distribution of categorical variables.
  - Compare the frequency of different categories.
  - Identify the most common categories within a dataset.

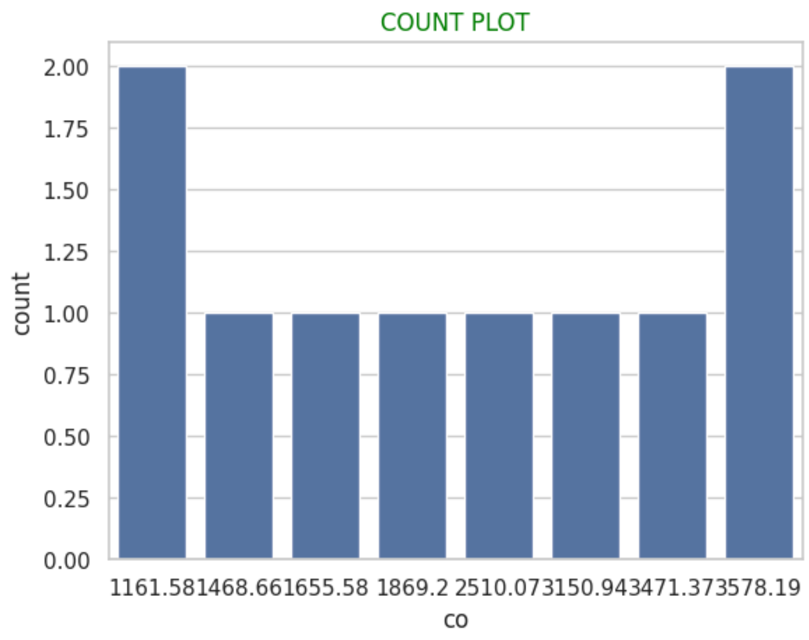
- **Benefits**:
  - Provides a quick summary of categorical data distribution.
  - Facilitates comparison between categories.
  - Useful for identifying imbalances or biases in categorical data.

CODE IMPLEMENTATION:



```
[ ]  #sample = data
     sns.countplot(x= "co", data = ds)
     plt.title("COUNT PLOT", color = "green")
     plt.show()
```

OUTPUT:



BOX PLOT

Definition:
A box plot (or box-and-whisker plot) displays the distribution of numerical data through quartiles, highlighting outliers and skewness.

Uses:

- Summarize the distribution and variability of continuous data.
- Identify outliers and extreme values.
- Compare distributions across different categories or groups.

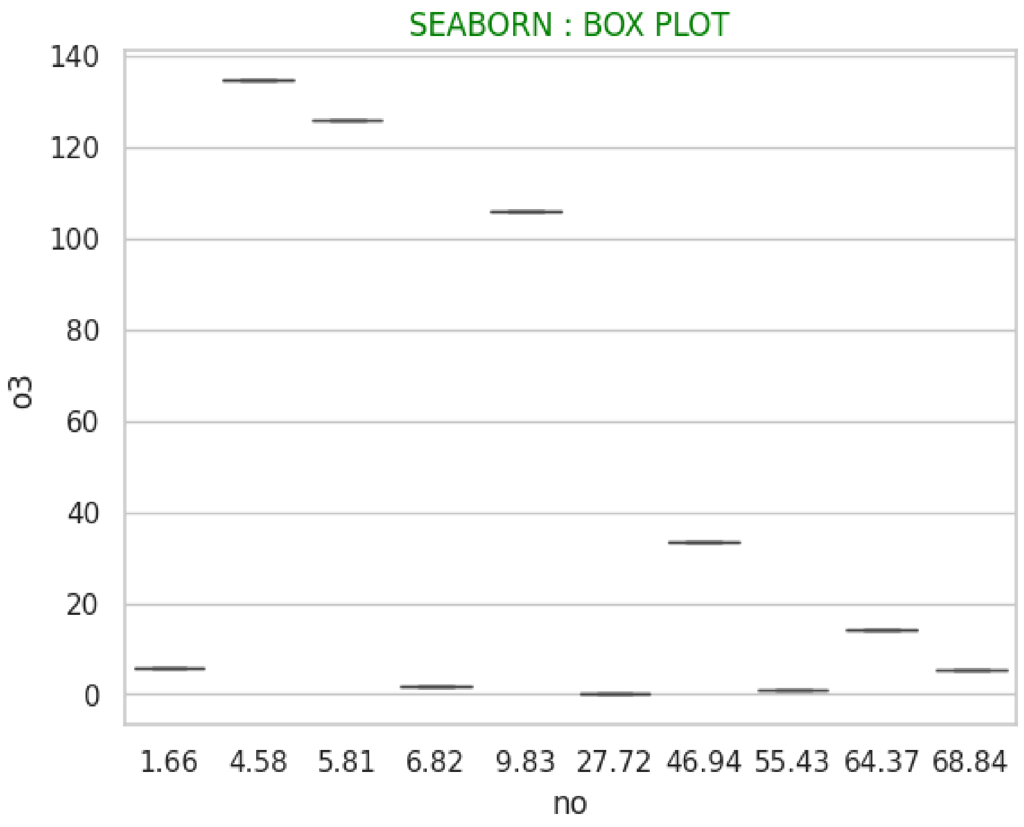
Benefits:

- Provides insights into the central tendency (median) and spread (interquartile range) of data.
- Visualizes outliers effectively.
- Facilitates easy comparison of distributions between groups.

CODE IMPLEMENTATION:

```
sns.boxplot(y=ds["o3"], x=ds["no"])
plt.title("SEABORN : BOX PLOT", color = "green")
plt.show()
```

OUTPUT:



VIOLIN PLOT

Definition:
A violin plot combines the features of a box plot and a KDE plot to show the distribution of numerical data across different levels of a categorical variable.

Uses:

- Compare the distribution of data between multiple groups.
- Display both summary statistics (like a box plot) and the entire distribution (like a KDE plot).
- Visualise the shape, spread, and multimodal nature of data distributions.

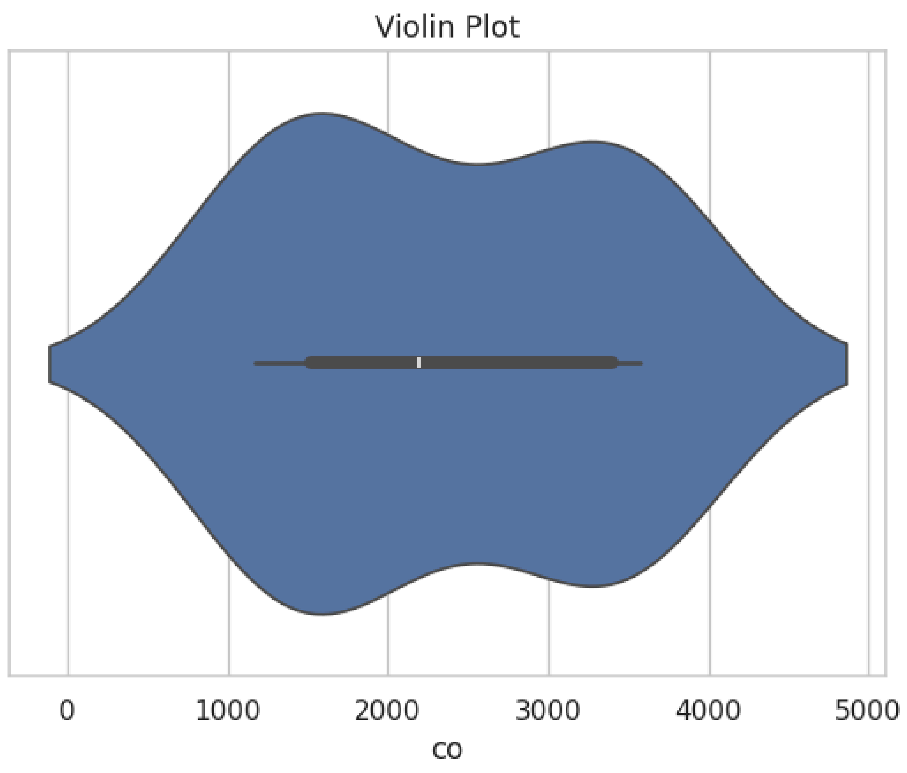
Benefits:

- Offers a richer visualisation compared to traditional box plots.
- Shows the density of data points at different values.
- Useful for understanding the distributional shape and skewness across categories.

CODE IMPLEMENTATION:

```
# Violin plot
sns.violinplot(x=ds["co"])
sns.set(style="whitegrid")
plt.title('Violin Plot')
plt.show()
```

OUTPUT:



STRIP PLOT

Definition:
A strip plot is a scatter plot where one variable is categorical. It displays individual data points along an axis.

Uses:

- Visualise the distribution of a continuous variable within each category.
- Identify individual data points and their density across categories.
- Compare distributions or patterns between groups.

Benefits:

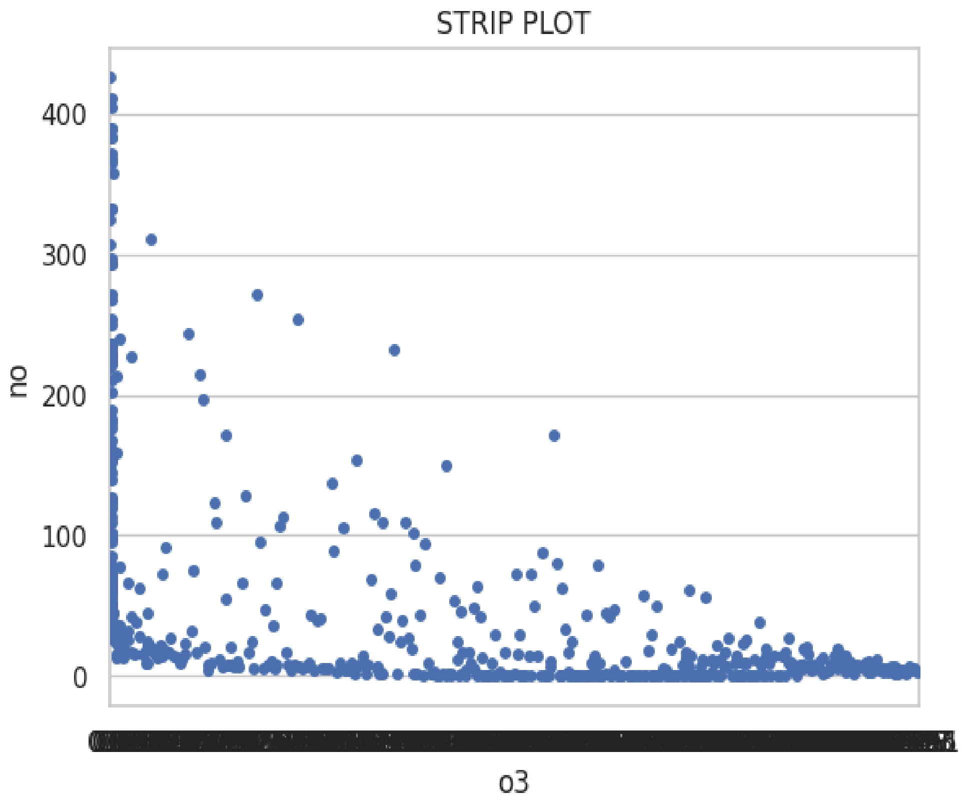
- Provides a detailed view of the distribution of data points.
- Allows for quick identification of outliers or clusters.
- Useful for exploring relationships between categorical and continuous variables.

CODE IMPLEMENTATION:

```
sns.set(style = 'whitegrid')
sns.stripplot(x="o3", y="no", data=data, jitter=0.1)
plt.title("STRIP PLOT")
```

OUTPUT:

```
Text(0.5, 1.0, 'STRIP PLOT')
```



PAIR PLOT

Definition:
A pair plot (or pairs plot) is a grid of pairwise relationships between variables in a dataset. It combines scatter plots for numerical variables and histograms or KDE plots for each variable along the diagonal.

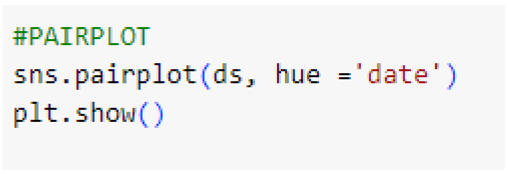
Uses:

- Explore correlations and relationships between multiple variables simultaneously.
- Visualise the distribution of each variable and its relationship with others.
- Identify patterns, trends, or clusters in high-dimensional data.

Benefits:

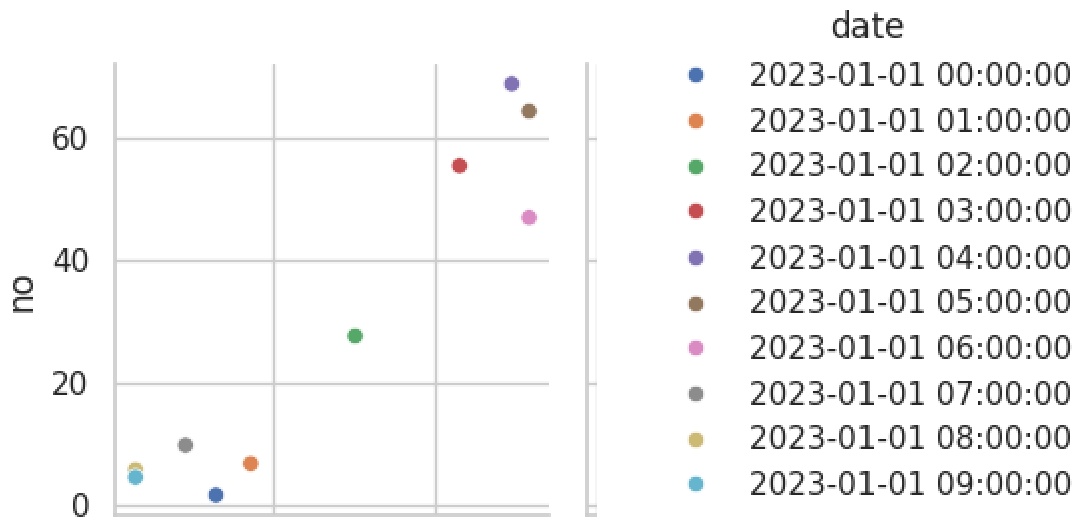
- Offers a comprehensive view of data relationships in a single plot.
- Facilitates quick identification of pairwise correlations.
- Useful for initial exploratory analysis and feature selection in data science tasks.

CODE IMPLEMENTATION:



```
#PAIRPLOT
sns.pairplot(ds, hue ='date')
plt.show()
```

OUTPUT:



COMPARISON BETWEEN MATPLOT & SEABORN

Comparison in terms of various factors is given below:

FEATURES	MATPLOTLIB	SEABORN
Ease of Use	provides fine-grained control over every aspect of a plot, allowing for highly customised visualisations	offers higher-level functions for creation of complex plots, making it easier to generate informative visualisations quickly
Interactivity	Supports basic interactivity such as zooming and panning out of the box.	Seaborn can leverage Matplotlib's interactive features, and with extensions like Plotly, it can create highly interactive plots with ease.
Customization	Highly customizable with extensive control over plot details such as axes, labels, and annotations.	Simplifies customization through intuitive APIs that automatically style plots with appealing colours and themes.
Performance with Large Datasets	Efficient for plotting large datasets due to its lower-level design, allowing direct manipulation of plot elements.	Optimised for statistical plots and handles moderate-sized datasets effectively.
Statistical Plotting	Suitable for general-purpose plotting and can be adapted for statistical visualisation with additional effort.	Specialises in statistical plotting with concise, high-level APIs for common statistical visualisations like box plots, violin plots, and pair plots.

CONCLUSION:

- **Matplotlib** offers extensive customization and control but requires more effort for complex statistical plotting and may be less intuitive for beginners.

- **Seaborn** simplifies statistical plotting with higher-level APIs, enhancing ease of use and speed of development but sacrificing some customization flexibility and performance with very large datasets.

Thus, selection/choosing between Matplotlib and Seaborn often depends on the specific requirements of the data visualisation task, including the level of customization needed, interactivity requirements, and the size and complexity of the dataset being visualised.