Practical Assignment No. 1	
Title:	Data Visualization
Problem Statement:	Analyse the data for different data visualization techniques (bar-graph, histogram, boxplot etc) using python. Choose a dataset from UCI Machine Learning repository.
Objective:	To use Scikit Learn for implementing data science operations.
Outcome:	CO606.1: Make use of Scikit Learn for implementing data science operations
Software or Hardware Requirements:	Anaconda/Java/GCC
Theory:	**Data Visualization:** Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and plots, it becomes easier to understand trends, outliers, and patterns in data. It is a key step in data analysis and helps in decision-making by presenting complex data intuitively. Python provides powerful libraries for data visualization such as **Matplotlib**, **Seaborn**, and **Pandas**, which allow users to create a wide variety of plots and charts easily. **Common Data Visualization Techniques:** **a) Bar Graph** A bar graph is used to represent categorical data with rectangular bars. The length or height of each bar corresponds to the value of the category it represents. **Use case:** Comparing values of different categories. **Python functions:** plt.bar(x, height) sns.barplot(x='category', y='value', data=df) **b) Histogram** A histogram shows the frequency distribution of numerical data by dividing the data into bins and plotting the number of observations in each bin. **Use case:** Understanding the distribution of continuous data. **Python functions:** plt.hist(data, bins=10) sns.histplot(data, kde=True) **c) Boxplot**

A boxplot (or box-and-whisker plot) summarizes the distribution of data by showing the median, quartiles, and outliers.
Use case: Detecting outliers and comparing distributions.
Python functions:
```
plt.boxplot(data)
sns.boxplot(x='variable', y='value', data=df)
```

d) Scatter Plot
A scatter plot displays values for two variables as points on a Cartesian plane, used to identify correlations or relationships between variables.
Use case: Visualizing relationships or trends between two continuous variables.
Python functions:
```
plt.scatter(x, y)
sns.scatterplot(x='feature1', y='feature2', data=df)
```

e) Line Plot
A line plot connects data points with a line, typically used to visualize trends over time.
Use case: Observing patterns or trends in time series data.
Python functions:
```
plt.plot(x, y)
sns.lineplot(x='time', y='value', data=df)
```

f) Pair Plot (Scatterplot Matrix)

A **pair plot** (also known as a scatterplot matrix) visualizes pairwise relationships between multiple numerical features. It plots all combinations of two variables and also shows their univariate distributions on the diagonal.

Use case: Multivariate analysis, correlation detection.

Python function:
```
sns.pairplot(df, hue='species')
```

Insight: Helps identify relationships and separability among classes (useful in classification tasks).

g) Heatmap

A **heatmap** displays data values as a matrix with color gradients representing magnitude. It's often used for showing correlation matrices or frequency distributions.

Use case: Correlation analysis, feature selection, or cluster visualization.
Python function:
```
 sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```
Insight: Reveals strength and direction of relationships between numerical features.

h) Violin Plot

A **violin plot** combines aspects of a boxplot and a kernel density plot. It shows the distribution, probability density, and summary statistics of a dataset.
Use case: Comparing distributions across multiple categories.
Python function:
```
 sns.violinplot(x='species', y='sepal_length', data=df)
```
Insight: Displays both spread and skewness of data more effectively than boxplots.

i) Joint Plot

A **joint plot** visualizes the relationship between two variables along with their individual distributions on the axes.
Use case: Bi-variate analysis with marginal histograms or density plots.
Python function:
```
 sns.jointplot(x='sepal_length', y='petal_length', data=df, kind='reg')
```
Insight: Shows correlation and trend between two continuous features.

j) Swarm Plot

A **swarm plot** arranges data points to avoid overlapping, making each observation visible. It's useful for categorical comparison.
Use case: Displaying all data points while maintaining the overall distribution shape.
Python function:
```
 sns.swarmplot(x='species', y='petal_width', data=df)
```
Insight: Good for small to medium-sized datasets to see distribution at individual point level.

k) 3D Plot

A **3D plot** adds a third dimension to data visualization using 3D coordinates. It can be used for representing multi-variable relationships.
Use case: Visualizing high-dimensional data.
Python function:
```
 from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df['sepal_length'], df['sepal_width'], df['petal_length'])
```

	Insight: Helps in identifying clusters and outliers in three-dimensional feature spaces.
	
	l) Pairwise Correlation Cluster Map
	
	A **cluster map** arranges variables into clusters based on similarity and visualizes them as a heatmap with hierarchical clustering.
	Use case: Detecting variable groupings or feature redundancy.
	Python function:
	`sns.clustermap(df.corr(), annot=True, cmap='coolwarm')`
	Insight: Provides hierarchical relationships between correlated variables.
	
	m) Interactive Visualizations (Using Plotly)
	Plotly allows the creation of interactive, web-based visualizations where users can zoom, hover, and filter data dynamically.
	Use case: Interactive dashboards and presentations.
	Python function:
	`import plotly.express as px`
	`fig = px.scatter_3d(df, x='sepal_length', y='sepal_width', z='petal_length', color='species')`
	`fig.show()`
	Insight: Enhances user engagement and allows real-time data exploration.
	
	Steps Involved
	
	1. **Import libraries:** Import pandas, matplotlib.pyplot, and seaborn.
	2. **Load dataset:** Load the dataset from UCI repository or from local storage.
	3. **Preprocess data:** Handle missing values and check data types.
	4. **Apply visualization techniques:** Generate bar graphs, histograms, boxplots, etc.
	5. **Interpret results:** Observe patterns, trends, and outliers from the visualizations.
	
	Advantages of Visualization
	
	● Provides intuitive understanding for non-technical stakeholders.
	● Handles **multi-dimensional and complex data** effectively.
	● Helps identify **hidden patterns** and **feature correlations**.
	● Enables **interactive analysis** and **dynamic visualization**.
	● Supports **data-driven decisions** through detailed visual summaries.

Input/Datasets/Test	Dataset- from UCI Machine Learning Repository Name of the Dataset:

Cases:	Description of the Dataset: Dataset Characteristics: Subject Area: Associated Tasks: Feature Type: # Instances: # Features:
Results:	Execute code to use appropriate UCI dataset and all 13 visualizations in the output. Take a print of this code with output for submission as part of results. ● Properly mention the description and analysis of every visualization.
Analysis and conclusion:	Write your own analysis of output and conclusion(Minimum 1 statement Analysis, Minimum 1 Statement Conclusion)
References:	Reference /Links(min Any 2, include dataset ref.). Write references in IEEE format.