**The Bombay Salesian Society’s**

**Don Bosco Institute of Technology Mumbai 400070.**

****

2024/ DBIT/ IT/ Sem VI/ ITL605 - Data Science using Python lab Submitted by

53- Asad Sayyad

Department of Information Technology Don Bosco Institute of Technology Academic year 2024-2025

INDEX

|  |  |  |
| --- | --- | --- |
| Sr. No. | Experiment Name | Date |
| 1 | Experiment A – Self Study | 12/01/2025 |
| 2 | Exp-1 :Basic Commands of Numpy and Pandas | 09/02/2025 |
| 3 | Exp-2: Plotting Graphs using Matplotlib | 23/02/2025 |
| 4 | Exp-3: Data Modeling | 01/04/2025 |

**Experiment A: Self - Study**

**Prerequisites**: Database, Python

**Aim**: To perform a self-study on Artificial Intelligence (AI) & Data Science (DS)

**Procedure**: Is to study the following questions and write a summary using available resources.

1. Define Artificial Intelligence.

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems.

1. What is a Agent ?

An agent is an independent program or entity that interacts with its environment by perceiving its surroundings.

1. What are the types of Agents?

There are five types of agents namely, Simple Reflex Agent, Model based Agent, Goal based Agent, Utility Agent and Learning Agent.

1. What are the five applications and case studies on AI?

The five applications are Healthcare, Marketing, Finance, Robotics and Entertainment and case studies are speech recognition, visual perception, decision-making, scale customer support and translation between languages.

1. What are the domains/areas of AI?

The domains of AI are Expert Systems, Machine Learning, Robotics, Fuzzy Logic and Natural Language Processing.

1. What are the components/building blocks of AI?

The components of AI are Learning, Perception, Reasoning, Language Understanding and Problem Solving.

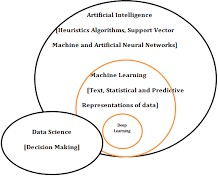
1. What is Data Science ?

Data Science is study to extract knowledge and insights from potentially noisy, structured, or unstructured data.

1. How is Data Science connected to AI or vice-versa?

Data science and artificial Intelligence are closely related as data science often incorporates AI techniques to analyze data and make predictions. AI, in turn, relies on data science to gather, pre-process, and interpret data for training its models.

1. Insert an image that gives the logical connect AI , ML , Data Science and Statistics?



1. Compare and contrast between data science and exploratory data analysis (EDA)? EDA (Sub-set) is a technique for ananlysing data sets and summarizing their main characteristics whereas Data Science (Super-set) is the study to extract meaningful insights for business from pre-processing to analysis.

**Conclusion:** Thus, for the above questions answers were found and thoroughly understood by me.

Exp 1

**Title:** Data preparation using NumPy and Pandas

**Problem Statement:**

1. Derive an index field and add it to the data set.
2. Find out the missing values.
3. Obtain a listing of all records that are outliers according to the any field. Print out a listing of the 10 largest values for that field.
4. Do the following for the any field. i. Standardize the variable. ii. Identify how many outliers there are and identify the most extreme outlier.

**Platform Used:**

Google Colab

**Name of Dataset:**

drugs(.csv file)

**Theory:**

# **What is DS, NumPy, Pandas, DataFrame, and EDA?**

* + **Data Science (DS)** involves extracting insights from data using algorithms, statistics, and programming.
  + **NumPy** is a library for numerical computing and efficient manipulation of arrays.
  + **Pandas** is a powerful library for data manipulation using DataFrames (tabular data structures).
  + **DataFrame** is a 2D structure (rows and columns) used for storing and manipulating data.
  + **EDA (Exploratory Data Analysis)** is the initial step to analyze data for insights, patterns, and cleaning.

# **How to read a CSV file (for Google Colab)?**

python

import pandas as pd

df = pd.read\_csv('filename.csv') # Reads CSV into a DataFrame In **Google Colab**, you can upload a file using:

python

:-

from google.colab import files

uploaded = files.upload() # Then use pd.read\_csv()

# **How to create a DataFrame?**

You can create a DataFrame from dictionaries, lists, or arrays. Example: python

import pandas as pd

data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]} df = pd.DataFrame(data) # Creates a DataFrame

# **How to select one/multiple columns from a dataset?**

To select one column:

python

:-

df['Column'] # Single column To select multiple columns:

python

:-

df[['Column1', 'Column2']] # Multiple columns

# **What is Index, and how to assign it?**

**Index** is the label for rows in a DataFrame. By default, pandas assigns numeric indices starting from 0. To assign a custom index:

python

:-

df.index = ['Row1', 'Row2', 'Row3']

# **set\_index() for single and multiple index**

* + **Single index**: python

:-

df.set\_index('Column1', inplace=True)

* + **Multiple index**: python

:-

df.set\_index(['Column1', 'Column2'], inplace=True)

# **Use of df.columns, head(), tail(), df.dtypes, etc.**

* + **df.columns**: Displays column names.
  + **df.head()**: Shows first 5 rows.
  + **df.tail()**: Shows last 5 rows.
  + **df.dtypes**: Displays data types of columns.
  + **df.isnull()**: Checks for missing values.
  + **df.sum()**: Sums the numeric columns.
  + **df.shape**: Returns the number of rows and columns.
  + **df.describe()**: Summary statistics of numeric columns.

# **What is Quantile, Percentile, and df.count(axis=1).head()?**

* + **Quantile**: Divides data into equal intervals (e.g., 25%, 50%).
  + **Percentile**: Represents a specific position in data (e.g., 90th percentile).
  + **df.count(axis=1).head()**: Counts non-null values for each row.

# **Syntax for replacing values in a column?**

You can replace values using:

python

:-

df['Column'].replace({old\_value: new\_value}, inplace=True)

# **What is np.nan and NaN, and iloc()?**

* + **np.nan**: Represents a missing value in NumPy.
  + **NaN**: Not a number, used to denote missing data in pandas.
  + **iloc()**: Allows index-based selection: python

:-

df.iloc[0] # First row

df.iloc[1, 2] # Row 1, Column 2

# **fillna(), bfill, ffill**

* + **fillna(value)**: Replaces missing values with a specific value.
  + **method='ffill'**: Forward fill, replaces NaN with the previous value.
  + **method='bfill'**: Backward fill, replaces NaN with the next value.

# **What is an outlier?**

An **outlier** is a data point that significantly deviates from other values. It can distort statistical analysis and may require special treatment (e.g., removal or transformation).

# **nlargest(), nsmallest(), and finding the most frequent value in a column**

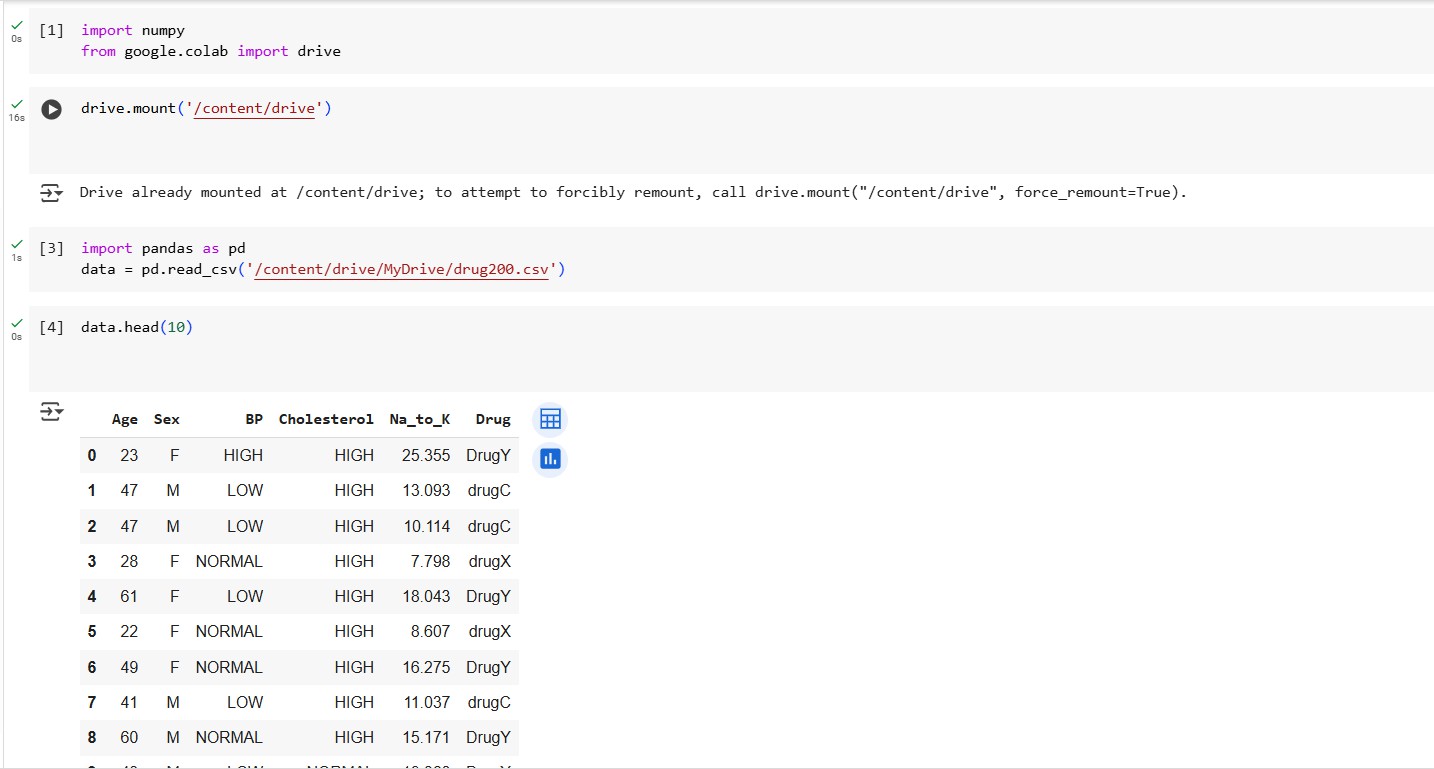
* + **nlargest(n)**: Returns the top n largest values in a column.
  + **nsmallest(n)**: Returns the top n smallest values.
  + **Most frequent value**: Use **mode()** to get the most frequent value(s) in a column.

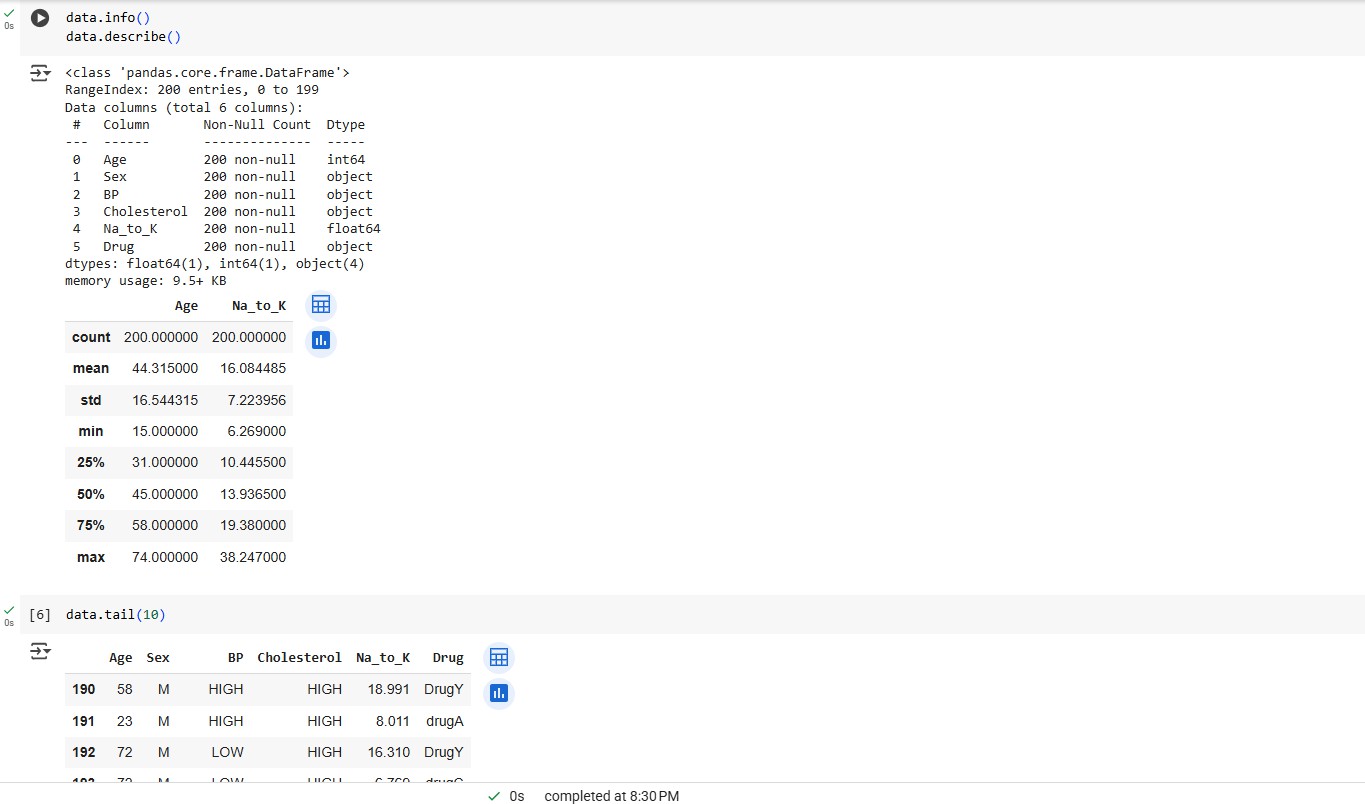
# **Ways to standardize data: Z-score, scipy.stats, StandardScaler, MinMaxScaler**

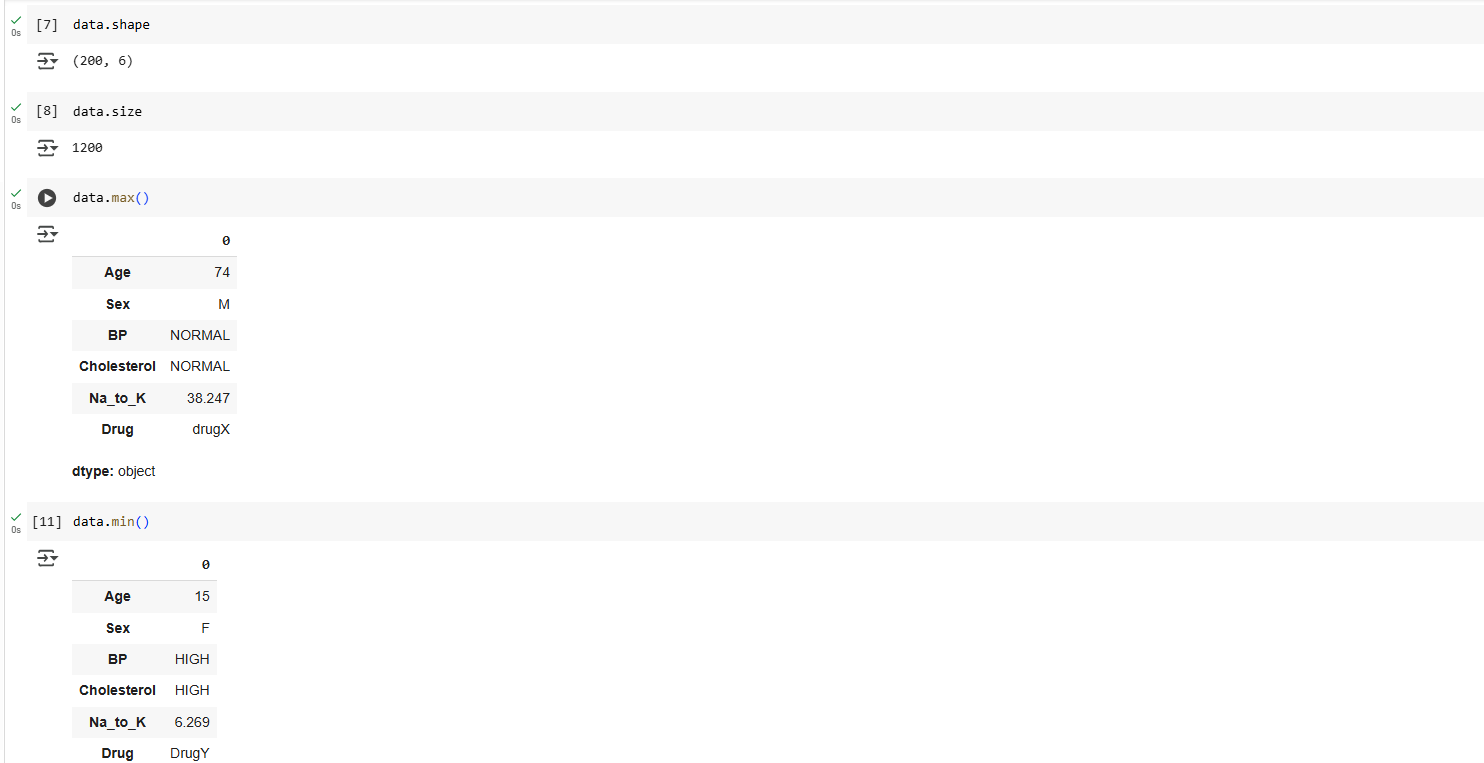
* + **Z-score**: Standardizes by subtracting the mean and dividing by the standard deviation.
  + **scipy.stats.zscore()**: Function to compute the Z-score.
  + **StandardScaler**: Scales data to have zero mean and unit variance.
  + **MinMaxScaler**: Scales data to a range (typically 0 to 1).

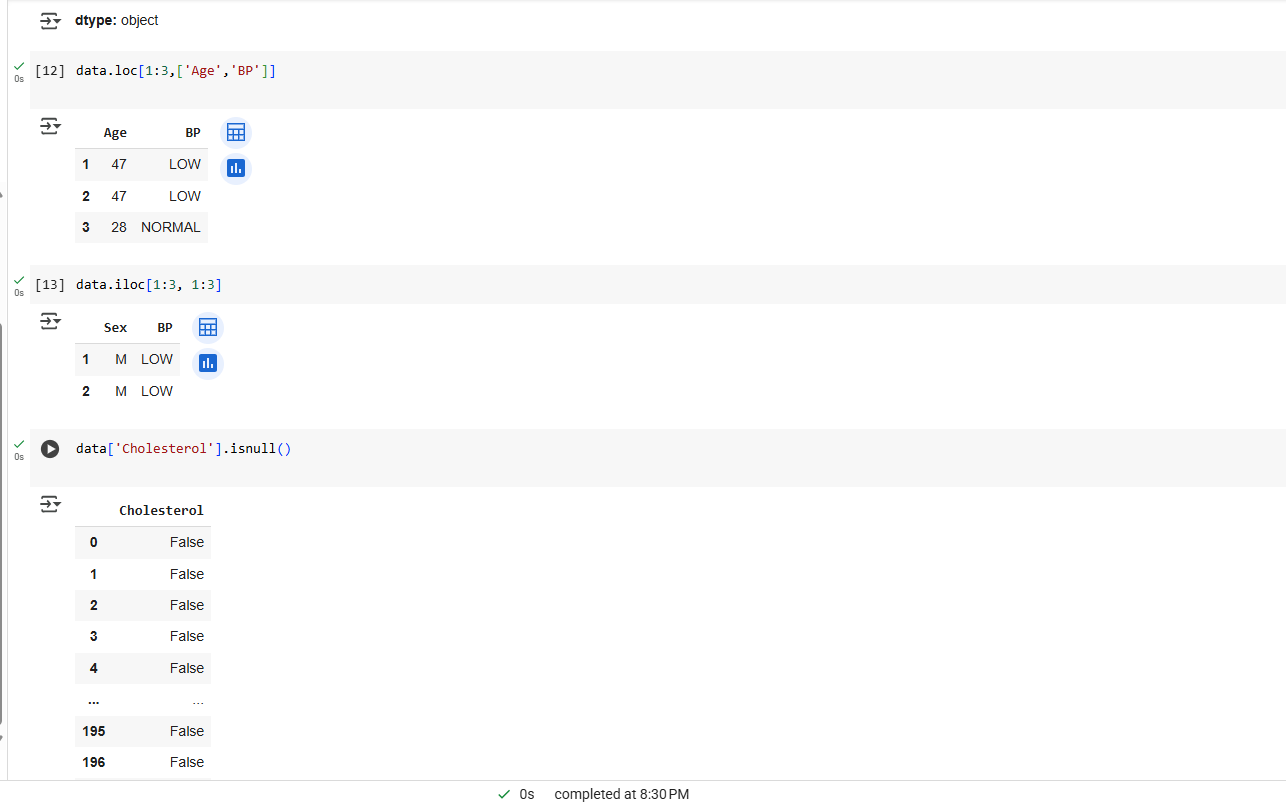
**Link\_of\_Execution:** https://colab.research.google.com/drive/1Wgop9PaZNmIzijshpSBOgNFUYzHgxa1P?usp=shari ng

**Screenshots of Code with Output:**

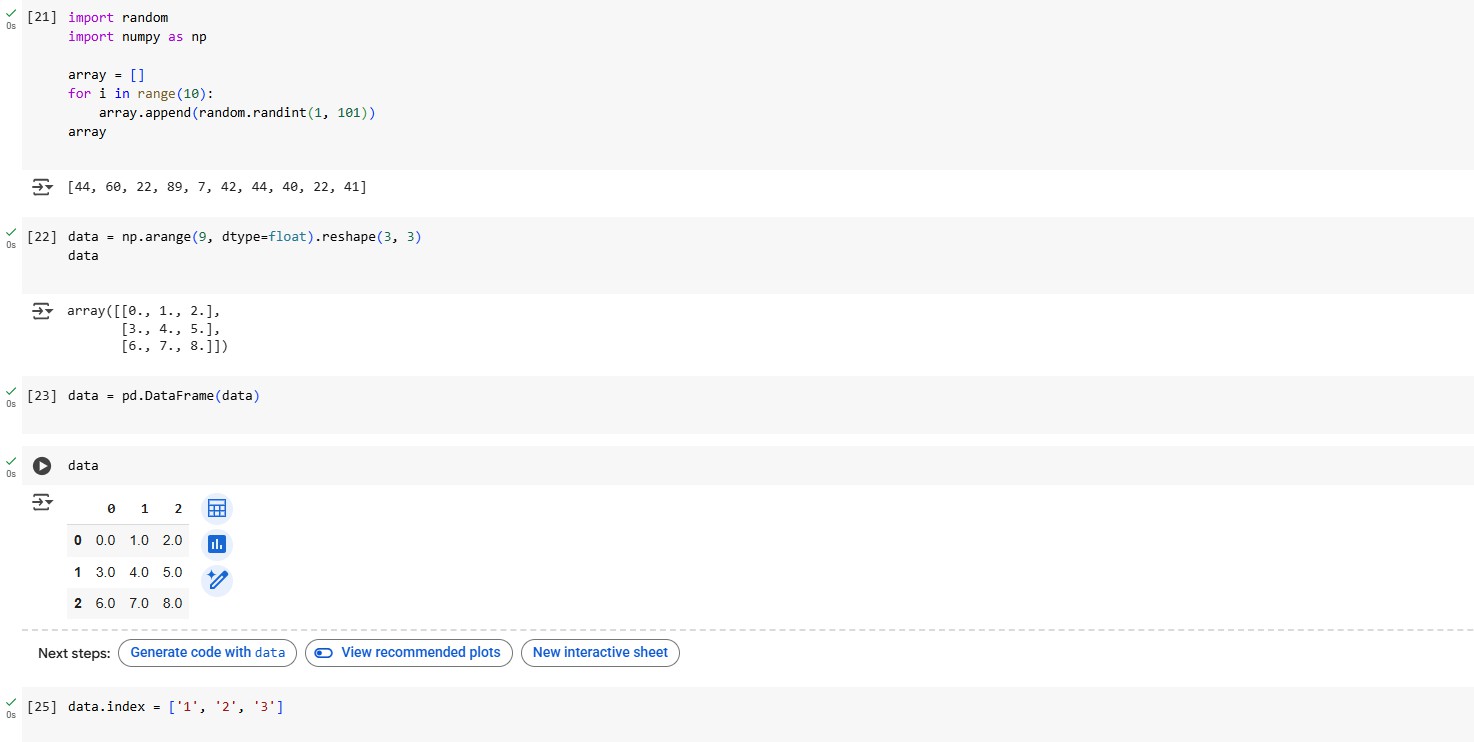
****

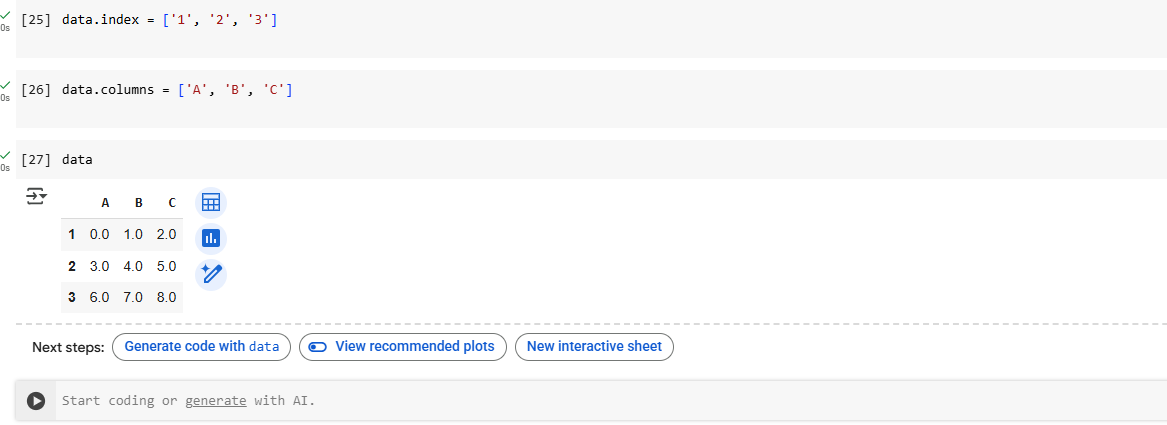


****



****



****

**Conclusion:**

Data Science uses tools like NumPy and Pandas for data analysis. NumPy handles numerical operations efficiently, while Pandas simplifies data manipulation through DataFrames. EDA helps understand the data by exploring structure, missing values, and patterns. Mastering core functions like reading files, indexing, handling nulls, statistical summaries, and standardization techniques is essential for real-world data analysis.

Exp 2

**Title:** Data Visualization / Exploratory Data Analysis for the selected data set using Matplotlib and Seaborn

**Problem Statement:**

a. Create a bar graph, contingency table using any 2 variables.

b. Create normalized histogram.

c. Describe what this graphs and tables indicates?

**Platform Used:** Google Colab

**Name of Dataset: drug200.csv**

**Theory:**

1. What is visualization  
     
   **Data Visualization** is the graphical representation of data using charts, plots, maps, and graphs.  
   It helps:  
   Understand patterns and trends  
   Communicate results clearly  
   Detect outliers or anomalies  
    Example: A scatter plot can reveal correlation between Age and Na\_to\_K.
2. Explain what is and use of seaborn and matplotlib   
     
   **Matplotlib**  
   A low-level plotting library in Python.  
   Gives complete control to design every aspect of the plot (axes, color, font, labels).  
   Good for basic visualizations like line plots, scatter plots, bar charts, etc.  
   **Seaborn**  
   A high-level data visualization library built on top of Matplotlib.  
   Provides a more concise and elegant syntax for visualizations.  
   Works well with pandas DataFrames.  
   Automatically handles styling and color mapping.
3. Screenshots 2-4 of practice of using seaborn and matplotlib

List down Data Visualization Plots  
Certainly! Here’s a more detailed explanation of each data visualization plot, including its use, purpose, and when to choose it:

### **1. Scatter Plot**

A **scatter plot** is a graph used to display the relationship between two continuous variables. Each point on the graph represents a data pair, plotted on the X and Y axes. It is particularly useful for identifying correlations or patterns, such as whether an increase in one variable corresponds with an increase or decrease in another. Scatter plots can also help identify clusters or outliers in the data.

**When to Use**:

When you want to explore the relationship between two numerical variables (e.g., height vs. weight).

To check if there is a linear or non-linear correlation.

### **2. Line Plot**

A **line plot** shows data points connected by straight lines, often used for tracking changes over periods. It is particularly effective in time-series analysis, where the X-axis represents time and the Y-axis represents some measured value. The line plot helps visualize trends, patterns, and anomalies over time.

**When to Use**:

When you want to visualize trends over time (e.g., stock prices, temperature changes, website traffic).  
  
To emphasize continuity and the progression of data.

### **3. Bar Plot**

A **bar plot** (or bar chart) uses rectangular bars to compare different categories of data. Each bar represents a category, and its length corresponds to the value or frequency of that category. Bar plots are versatile and can be oriented vertically or horizontally.

**When to Use**:

To compare discrete categories (e.g., sales in different regions, number of students in different courses).  
When categories are independent of each other.

### **4. Histogram**

A **histogram** is a type of bar plot used to show the distribution of a single continuous variable. Instead of individual categories, it divides the range of values into bins and displays how many data points fall into each bin. It is useful for visualizing the frequency distribution of data.

**When to Use**:

When you want to see the distribution of a dataset (e.g., age distribution in a population, exam scores).  
To identify the shape of the data distribution (e.g., normal, skewed).

### **5. Box Plot (Box-and-Whisker Plot)**

A **box plot** is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. The box represents the interquartile range (IQR), with lines (whiskers) extending to the minimum and maximum data points. Outliers are plotted as individual points.

**When to Use**:

To compare distributions of data across different categories.  
To identify outliers, median, and data spread.

### **6. Heatmap**

A **heatmap** is a graphical representation of data where individual values are represented by color gradients. It’s often used in the context of matrices, where the color intensity shows how different values correlate with each other.

**When to Use**:  
For visualizing correlation matrices, geographical data, or any data that benefits from a color scale (e.g., gene expression data, website traffic by time of day).

### **7. Pair Plot**

A **pair plot** visualizes the relationships between multiple variables by creating a grid of scatter plots for each pair of variables. It can also show distributions of each individual variable along the diagonal.

**When to Use**:

When you want to compare multiple numerical variables and understand how each one correlates with the others.I  
deal for exploratory data analysis (EDA).

### **8. Violin Plot**

A **violin plot** combines aspects of box plots and density plots. It shows the distribution of the data and is especially useful for comparing the distribution of multiple categories. The "violin" shape shows the data's density at different values, giving a more detailed view than a box plot.

**When to Use**:

To compare the distribution of data across categories.  
When you need a more detailed understanding of the density of the data.

### **9. Pie Chart**

A **pie chart** represents categorical data in a circular format, where each slice corresponds to a category’s proportion of the total. The area of each slice is proportional to the value it represents.

**When to Use**:

To show the composition of a whole and the proportion of each category.  
Best used when the categories are few (around 4–6).

### **10. Area Plot**

An **area plot** is similar to a line plot but with the area under the line filled. It is used to represent cumulative data and trends over time. It is effective for visualizing quantities that accumulate, such as total sales over time.

**When to Use**:

When you want to visualize cumulative data.  
To show trends and highlight changes in magnitude over time.

### **11. Contour Plot**

A **contour plot** displays three-dimensional data in two dimensions, where contour lines represent constant values of the third variable. It's often used to display geographic or scientific data.

**When to Use**:

For visualizing topographical data or any data with a spatial component (e.g., temperature or elevation).

When you want to represent data that has a continuous nature.

### **12. Bubble Chart**

A **bubble chart** is an extension of the scatter plot where an additional dimension is represented by the size of the bubbles. Each bubble represents a data point, with its size corresponding to an additional variable, and the position on the X and Y axes representing two other variables.

When you want to represent three variables simultaneously (e.g., sales, product rating, and price).  
For visualizing relationships between multiple variables.

1. Explain what is bar graph, histogram and contingency table and heatmap, mention when to use these plots also uni/bi/multi variant. Add screenshot.

**Link of Execution:**

**Screenshots of Code with Output:**

from google.colab import drive

drive.mount('/content/drive')

# importing the modules

import matplotlib

import seaborn

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# printing the versions

print("Matplotlib :", matplotlib.\_\_version\_\_)

print("Seaborn :", seaborn.\_\_version\_\_)

# loading the dataset

dataset = pd.read\_csv('/content/drive/MyDrive/Mangesh/drug200.csv')

# show top 5 rows

print(dataset.head())

  Age Sex      BP Cholesterol  Na\_to\_K   Drug

0   23   F    HIGH        HIGH   25.355  DrugY

1   47   M     LOW        HIGH   13.093  drugC

2   47   M     LOW        HIGH   10.114  drugC

3   28   F  NORMAL        HIGH    7.798  drugX

4   61   F     LOW        HIGH   18.043  DrugY

### **Scatter Plots**

A scatter plot is a graph in which the values of two variables are plotted along two axes, the pattern of the resulting points revealing any correlation present.

Before plotting the scatter plot using Maplotlib and Seaborn, let us first load the dataset. In this tutorial, we will be using the iris dataset. We’ll use seaborn’s built-in dataset to do this.

# scatter plot - Na\_to\_K vs Age

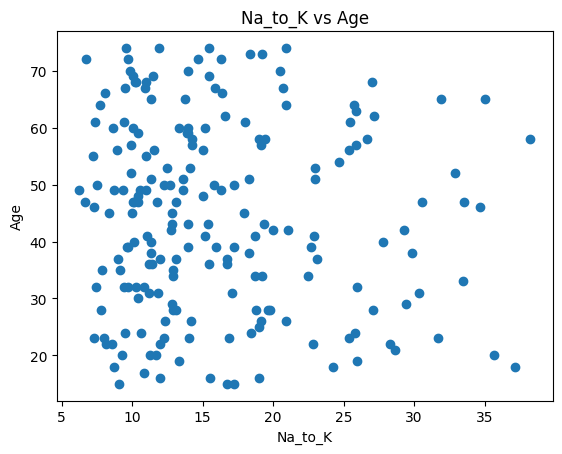
plt.scatter(dataset['Na\_to\_K'], dataset['Age'])

plt.xlabel("Na\_to\_K")

plt.ylabel("Age")

plt.title("Na\_to\_K vs Age")

plt.show()



# scatter plot with color mapping by Drug type

colors = {'DrugY': 'b', 'drugX': 'g', 'drugA': 'r', 'drugB': 'm', 'drugC': 'c'}

fig, ax = plt.subplots()

for i in range(len(dataset)):

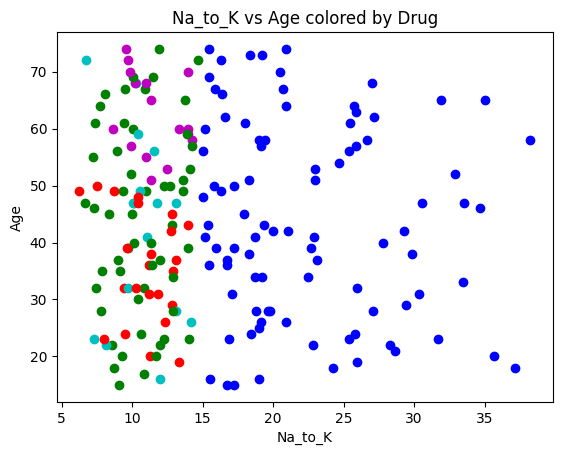
    ax.scatter(dataset['Na\_to\_K'][i], dataset['Age'][i], color=colors[dataset['Drug'][i]])

plt.xlabel("Na\_to\_K")

plt.ylabel("Age")

plt.title("Na\_to\_K vs Age colored by Drug")

plt.show()

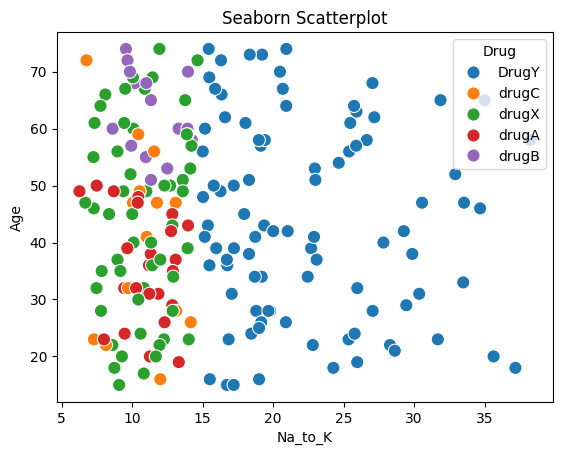


# seaborn scatterplot

sns.scatterplot(x=dataset['Na\_to\_K'], y=dataset['Age'], hue=dataset['Drug'], s=90)

plt.title("Seaborn Scatterplot")

plt.show()



### **Line Graphs A line chart displays the evolution of one or several numeric variables and is one of the most common chart types for time series and regression datasets. Plotting line charts in the Maplotlib model is very easy. The plot() method is used to plot line charts. Let us plot a line chart of petal length(cm) from the dataset:**

# line plot - Na\_to\_K

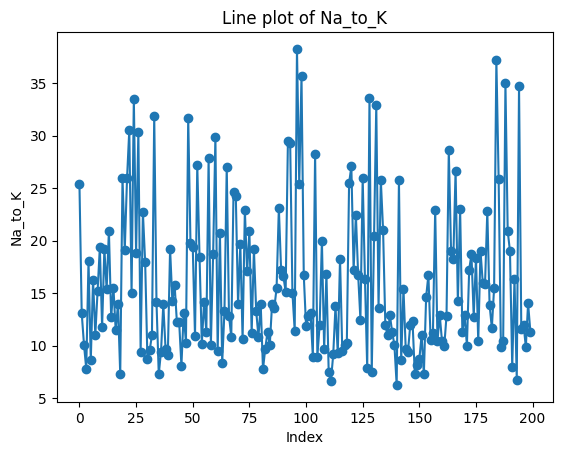
plt.plot(range(len(dataset)), dataset['Na\_to\_K'], marker='o')

plt.xlabel("Index")

plt.ylabel("Na\_to\_K")

plt.title("Line plot of Na\_to\_K")

plt.show()

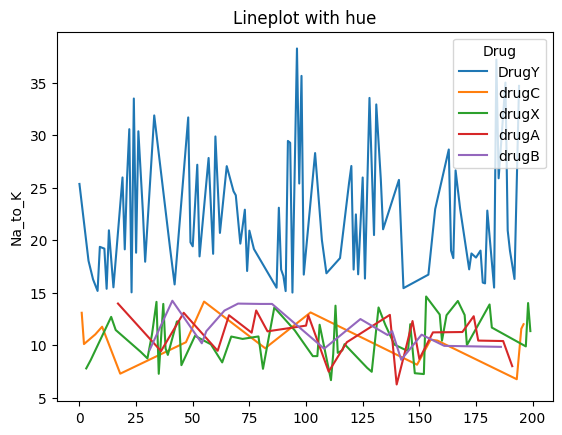


# line plot - Na\_to\_K for each Drug class

sns.lineplot(data=dataset, x=range(len(dataset)), y='Na\_to\_K', hue='Drug')

plt.title("Lineplot with hue")

plt.show()



### **Histogram Plots A histogram is a plot that lets us discover and show the underlying frequency distribution (shape) of a set of continuous data. This allows the inspection of the data for its underlying distribution (e.g., normal distribution), outliers, skewness, etc. The hist() method in the Matplotlib module is used to plot a histogram of the dataset.**

# histogram - Na\_to\_K

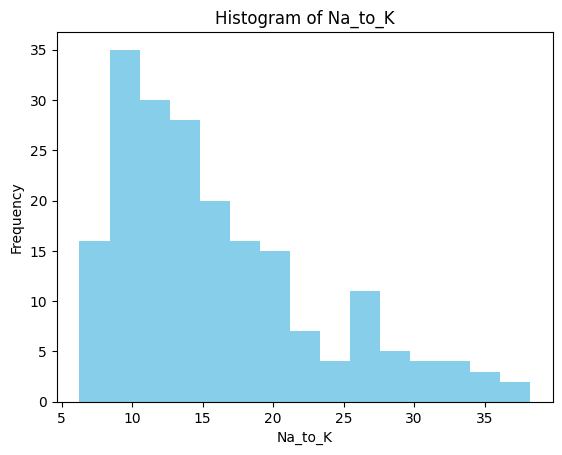
plt.hist(dataset['Na\_to\_K'], bins=15, color='skyblue')

plt.title("Histogram of Na\_to\_K")

plt.xlabel("Na\_to\_K")

plt.ylabel("Frequency")

plt.show()

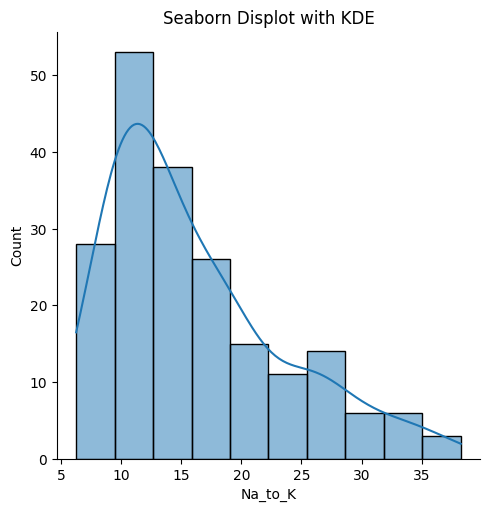


# seaborn histogram with KDE

sns.displot(data=dataset, x='Na\_to\_K', bins=10, kde=True)

plt.title("Seaborn Displot with KDE")

plt.show()



### **Box Plots A box plot is a simple way of representing statistical data on a plot in which a rectangle is drawn to represent the second and third quartiles, usually with a vertical line inside to indicate the median value. The lower and upper quartiles are shown as horizontal lines on either side of the rectangle. The boxplot() method in Matplotlib is used to plot box plots. Let us first plot the box plot for the length of petals:**

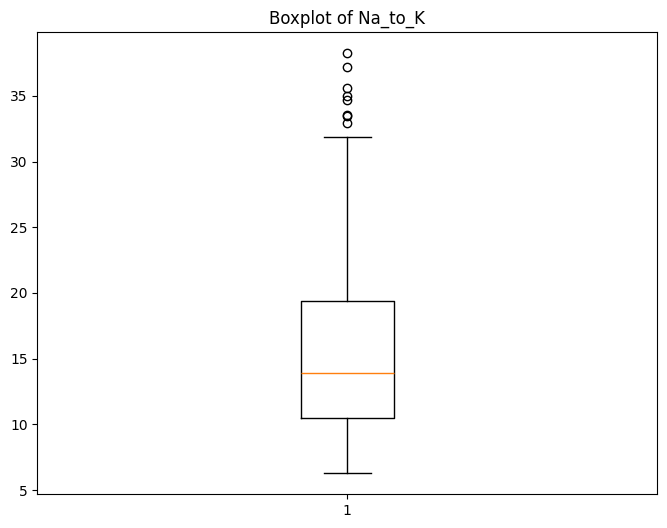
# boxplot - Na\_to\_K

plt.figure(figsize=(8,6))

plt.boxplot(dataset['Na\_to\_K'])

plt.title("Boxplot of Na\_to\_K")

plt.show()



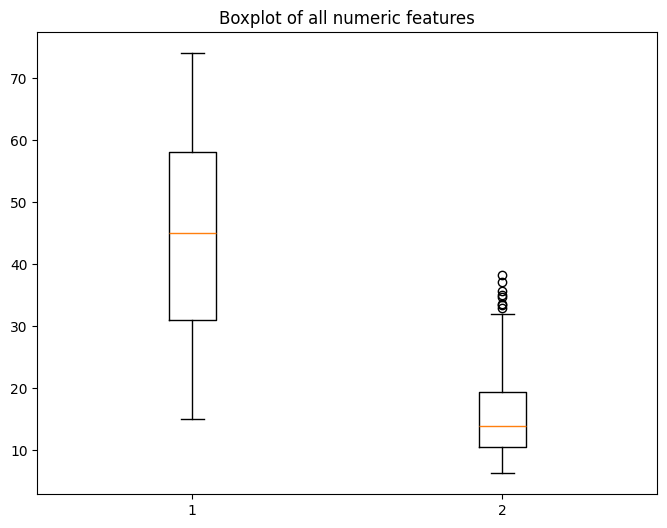
# boxplot for all numerical columns

plt.figure(figsize=(8,6))

plt.boxplot(dataset[['Age', 'Na\_to\_K']].values)

plt.title("Boxplot of all numeric features")

plt.show()

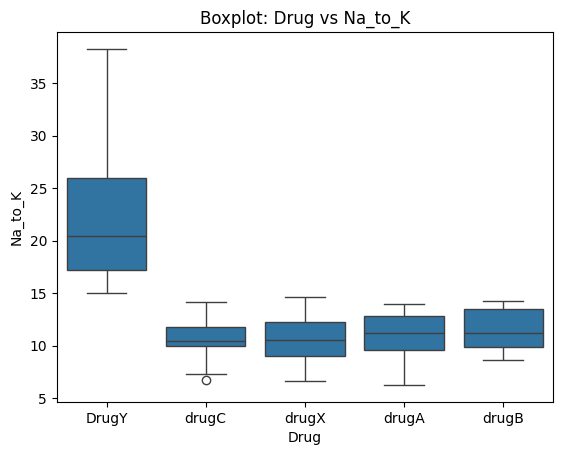


# seaborn boxplot - Drug vs Na\_to\_K

sns.boxplot(data=dataset, x='Drug', y='Na\_to\_K')

plt.title("Boxplot: Drug vs Na\_to\_K")

plt.show()



### **Violin Plots Violin plots are similar to box plots, except that they also show the probability density of the data at different values. These plots include a marker for the median of the data and a box indicating the interquartile range, as in the standard box plots. Overlaid on this box plot is a kernel density estimation. The violinplot() function in Matplotlib is used to plot violin plots.**

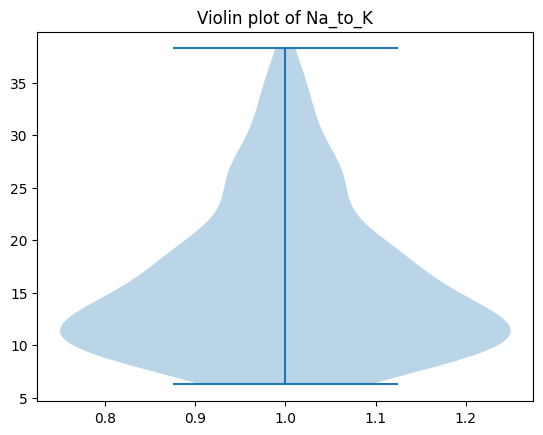
# violin plot - Na\_to\_K

plt.figure()

plt.violinplot(dataset['Na\_to\_K'])

plt.title("Violin plot of Na\_to\_K")

plt.show()



# violin plot with multiple features

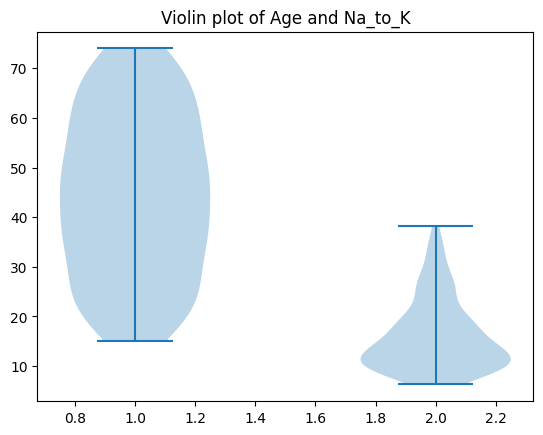
plt.figure()

data = [dataset['Age'], dataset['Na\_to\_K']]

plt.violinplot(data)

plt.title("Violin plot of Age and Na\_to\_K")

plt.show()

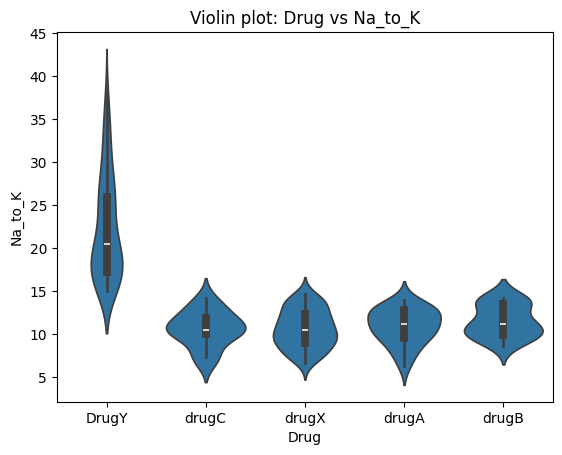


# violin plot - Drug vs Na\_to\_K

sns.violinplot(data=dataset, x='Drug', y='Na\_to\_K')

plt.title("Violin plot: Drug vs Na\_to\_K")

plt.show()



### **Bar Plots**

A bar plot is a plot that presents categorical data with rectangular bars with lengths proportional to the values that they represent. A bar plot shows comparisons among discrete categories. One axis of the plot shows the specific categories being compared, and the other axis represents a measured value. The difference between a bar plot and a histogram plot is that a bar graph is the graphical representation of categorical data while a histogram is the visual representation of grouped data continuously.

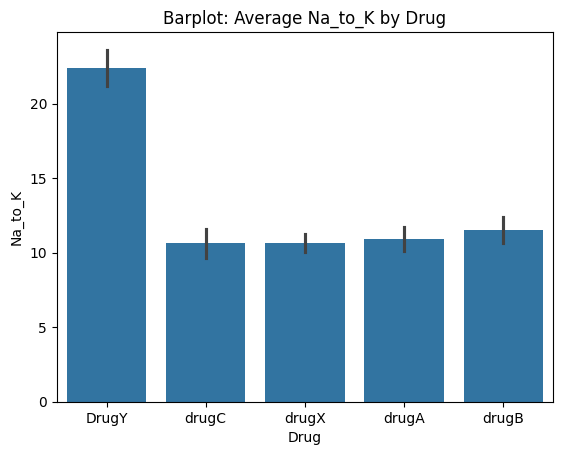
The bar( ) method in Matplotlib creates a bar plot.

# bar plot of average Na\_to\_K per drug

sns.barplot(data=dataset, x="Drug", y="Na\_to\_K")

plt.title("Barplot: Average Na\_to\_K by Drug")

plt.show()



## **Creating Legends Graph legends give meaning to the visualization of our data and provide title names for the different elements of the graph. As we’ve seen in the preceding examples, because our data is grouped by species, we were often able to get a reasonable legend with little work, especially when we used Seaborn. This is not always the case, however, so this section will explore how to create various legends. The simplest legend can be created with the plt.legend() command, which automatically creates a legend for all labeled graph elements:**

# simple multiple line plot (like sin/cos)

x = np.linspace(0, 10, 100)

fig, ax = plt.subplots()

ax.plot(x, np.sin(x), '--m', label='Sine')

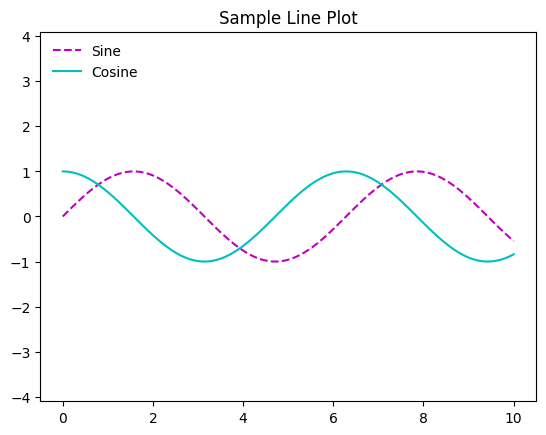
ax.plot(x, np.cos(x), '-c', label='Cosine')

ax.axis('equal')

ax.legend(loc='upper left', frameon=False)

plt.title("Sample Line Plot")

plt.show()



### **Conclusion**

Each type of plot—bar graph, histogram, contingency table, and heatmap—serves a unique purpose in data visualization and analysis. Bar graphs are excellent for comparing categories, while histograms are ideal for understanding the distribution of continuous variables. Contingency tables help reveal relationships between categorical variables, and heatmaps are powerful tools for visualizing complex patterns in multivariate data, especially correlations.

The choice of plot depends on the nature of the data and the analysis goal. Bar graphs are typically univariate but can be extended to multivariate cases, while histograms focus on the distribution of a single continuous variable. Contingency tables are primarily used for bivariate analysis of categorical data, and heatmaps offer a versatile solution for displaying relationships between two or more variables in a visually intuitive way. By understanding when and how to use these plots, you can gain deeper insights into your data and effectively communicate findings.

**Don Bosco Institute of Technology, Kurla(W),Mumbai**

**Department of Information Technology**

**Data Science using Python lab TE\_IT 2019(C scheme)**

**A.Y. 2024-25 SEM VI**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Exp 3

**Title:** Data Modeling

**Problem Statement:**

a. Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set. Use a bar graph to confirm your proportions.

b. Identify the total number of records in the training data set.

c. Validate your partition by performing a two‐sample Z‐test.

**Platform Used:** Google Collab

**Name of Dataset:drug200.csv**

**Theory:**

**Train-Test Split**

The **train-test split** is a method used in machine learning to evaluate a model's performance on unseen data. The idea is to split the dataset into two parts:

1. **Training Set**: This is the portion of the dataset used to train the model. It is where the model learns from the data and optimizes its parameters.
2. **Test Set**: This is a separate portion of the dataset used to evaluate the performance of the model. The test set is not used during training, so it provides a measure of how well the model generalizes to new, unseen data.

**Common Splitting Ratios:**

* **70% - 30%**: 70% of the data is used for training, and 30% is used for testing.
* **80% - 20%**: 80% for training, 20% for testing.
* **90% - 10%**: 90% for training, 10% for testing.

A good split ensures the model has enough data to learn from, while still leaving enough unseen data to evaluate the model's performance accurately.

**Why It's Important:**

* **Model Evaluation**: Helps in assessing the model’s performance on unseen data, preventing overfitting (where a model performs well on the training data but poorly on new data).
* **Generalization**: A good model should perform well on both training and test sets, indicating it generalizes well.

**Z-Score Method**

The **Z-score method** is a technique used in statistics to standardize data, especially when the features (attributes) have different units or scales. It transforms the data such that it has a mean of 0 and a standard deviation of 1. This is particularly useful when working with machine learning algorithms that are sensitive to the scale of the input features, such as **K-nearest neighbors (KNN)**, **support vector machines (SVM)**, and **principal component analysis (PCA)**.

The Z-score of a data point is calculated using the formula:

Z=X−μσZ = \frac{X - \mu}{\sigma}

Where:

* **X** is the value of the data point,
* **μ (mu)** is the mean of the data,
* **σ (sigma)** is the standard deviation of the data.

**Steps:**

1. **Calculate the Mean** (μ\mu): The average of all the data points in the feature.
2. **Calculate the Standard Deviation** (σ\sigma): This measures the spread of the data points around the mean.
3. **Standardize**: For each data point, subtract the mean and divide by the standard deviation to get the Z-score.

**Why Use Z-Score?**

* **Handling Outliers**: Z-scores can help in detecting outliers. Any data point with a Z-score greater than 3 or less than -3 is typically considered an outlier.
* **Normalization**: In many algorithms, especially those involving distance metrics (e.g., KNN), normalizing the data helps in reducing the bias caused by differing scales between features.
* **Improved Convergence**: For gradient-based algorithms (like neural networks), Z-score normalization can lead to faster convergence during training.

**Example:**

Suppose we have the data points [10,12,14,16,18][10, 12, 14, 16, 18]. To standardize this:

1. Mean μ=10+12+14+16+185=14\mu = \frac{10 + 12 + 14 + 16 + 18}{5} = 14
2. Standard Deviation σ=(10−14)2+(12−14)2+(14−14)2+(16−14)2+(18−14)25=2.83\sigma = \sqrt{\frac{(10-14)^2 + (12-14)^2 + (14-14)^2 + (16-14)^2 + (18-14)^2}{5}} = 2.83
3. For each value of X, apply the formula:
   * Z for 10: Z=10−142.83=−1.41Z = \frac{10 - 14}{2.83} = -1.41
   * Z for 12: Z=12−142.83=−0.71Z = \frac{12 - 14}{2.83} = -0.71
   * Z for 14: Z=14−142.83=0Z = \frac{14 - 14}{2.83} = 0
   * Z for 16: Z=16−142.83=0.71Z = \frac{16 - 14}{2.83} = 0.71
   * Z for 18: Z=18−142.83=1.41Z = \frac{18 - 14}{2.83} = 1.41

The transformed data would be [−1.41,−0.71,0,0.71,1.41][-1.41, -0.71, 0, 0.71, 1.41].

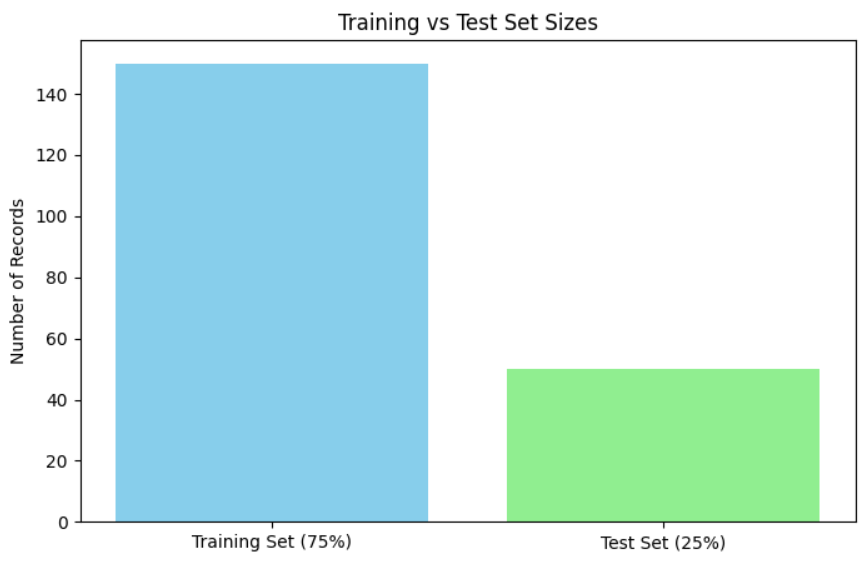
**Summary:**

* **Train-Test Split**: Divides the data into a training set for learning and a test set for evaluation. It helps to prevent overfitting and assess model performance.
* **Z-Score Method**: Standardizes features by converting them to a scale with mean 0 and standard deviation 1, helping algorithms that are sensitive to feature scaling.

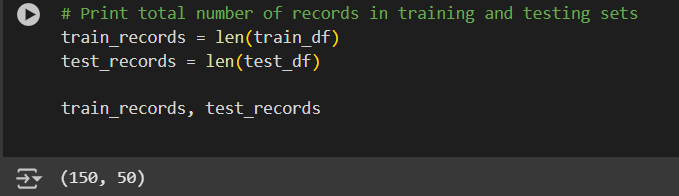
**LinkofExecution: https://colab.research.google.com/drive/13kSYlgbpp95EAZ2YsPew2\_oQkFM7jn0b?usp=sharing**

**Screenshots of Code with Output:**

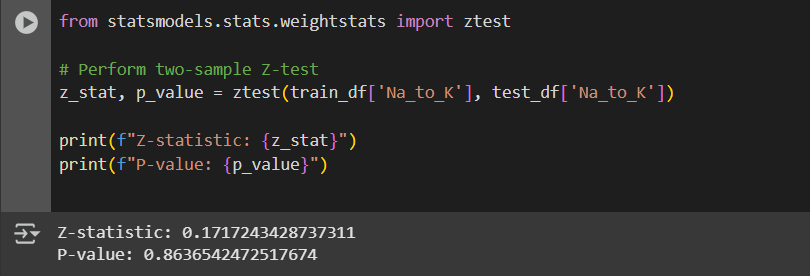
* 1. **Bar graph for train and test split**

****

**2.Number of record for train and test splits**

****

**3.Z-score statistics**

****

**Conclusion:**

**Conclusion:**

The task involves partitioning a dataset into training and test sets, followed by validating the partition through statistical analysis.

1. **Data Partitioning**: By splitting the dataset into 75% training and 25% test data, we ensure that the model has enough data to learn from while leaving a portion for evaluation. This is a standard practice in machine learning to assess model generalization and avoid overfitting. The bar graph visually confirms that the data is correctly divided, showing the proportion of records in the training and test sets.
2. **Training Data Set Size**: The total number of records in the training dataset is determined by multiplying the total dataset size by 75%. This value helps to verify the exact number of records allocated for training, providing clarity on the distribution of data.
3. **Z-Test Validation**: The two-sample Z-test validates the statistical significance of the partition. By comparing the means of the two datasets (training and test), we ensure that the partitioning process did not introduce any bias. A successful Z-test confirms that the data splitting is random and that both the training and test sets represent the overall population well.

In summary, partitioning the dataset ensures an appropriate balance for training and testing the model. The bar graph serves as a visual confirmation, and the Z-test statistically validates that the partitioning process was carried out properly without any systematic bias.