

Since 2011



Our outcomes are over 5000 trainees.



Artificial Intelligence Engineering (Level-1)

Level-1

Realistic Infotech Group

- Module 1: Introduction to AI and Machine Learning
- Module 2: Linear Algebra, Statistics and Probability for Al
- Module 3: Neural Network Architecture
- Module 4: Building Machine Learning Models
- Module 5: Deep Learning Concepts
- Module 6: Python Data Structure
- Module 7: Data Handling with Pandas and NumPy
- Module 8: Python for Al
- Module 9: Classification Al Project
- Module 10: Prediction Al Project



Artificial Intelligence Engineering (Level-1)

Module 7: Data Handling with Pandas and NumPy

Content

Realistic Infotech Group

- Numpy
- Pandas
- Data Handling and Manipulation
 - Loading Data
 - Data Exploration
 - Data Cleaning
 - Data Transformation
 - Statistical Analysis
 - Visualization

Learning Outcomes

Realistic Infotech Group

- Master NumPy Fundamentals: Learn to create and manipulate arrays for efficient numerical computations.
- Understand Pandas Basics: Use Pandas for data manipulation, including creating and modifying DataFrames and Series.
- Efficient Data Handling: Load, handle, and manipulate structured and unstructured data effectively.
- Learn Data Loading Techniques: Import data from various formats (CSV, Excel, databases) into Python for analysis.
- Explore Data: Perform exploratory data analysis to uncover patterns, trends, and insights in datasets.

Learning Outcomes



- Clean Data: Identify and handle missing, incorrect, or irrelevant data to prepare it for analysis.
- Transform Data: Apply techniques to normalize, aggregate, or restructure data for better usability.
- Perform Statistical Analysis: Apply statistical methods to summarize and infer information from data.
- Visualize Data: Create informative and visually appealing charts and graphs using libraries like Matplotlib or Seaborn.
- Prepare Data for Machine Learning: Develop skills to preprocess and transform data for use in machine learning models.

Why Data Handling is important



- Decision Making
 - Accurate data enables informed and strategic decisions.
- Problem Solving
 - Helps identify trends and patterns to address challenges effectively.
- Organized Information
 - Simplifies complex data into manageable formats.
- Enhanced Efficiency
 - Saves time and resources by optimizing data processes.
- Compliance and Security
 - Ensures data integrity and adheres to regulations.



NumPy Practices

What is NumPy?



- NumPy (Numerical Python) is a powerful open-source library in Python used for numerical and scientific computing.
- It provides support for working with large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these data structures efficiently.

Key Features:

- Supports multi-dimensional arrays
- Provides mathematical functions for array operations
- Efficient memory usage

NumPy Array Vs Python Lists



Feature	NumPy Arrays	Python Lists
Data Type Consistency	Stores elements of the same data type.	Can store mixed data types in one list.
Performance	Faster due to optimized C-based operations.	Can store mixed data types in one list.
Memory Usage	More memory-efficient.	Less efficient as it stores metadata for each element.
Mathematical Operations	Supports element-wise operations directly.	Requires manual loops or map functions.
Multidimensional Data	Supports multi-dimensional arrays (e.g., 2D, 3D).	Requires nested lists for multidimensional data.
Ease of Use	Designed for numerical computations.	General-purpose and simpler for non-mathematical tasks.

How to import NumPy



Before importing, ensure NumPy is installed in your Python environment.
 Use the following command to install it:

```
C:\Users\User>pip install numpy
Collecting numpy
Downloading numpy-2.1.3-cp313-cp313-win_amd64.whl.metadata (60 kB)
Downloading numpy-2.1.3-cp313-cp313-win_amd64.whl (12.6 MB)

12.6/12.6 MB 6.7 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-2.1.3

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\User>
```

• To use the NumPy library in Python, you need to **import** it. The standard way to import NumPy is:

```
import numpy as np
```



Creating Array

np.array(): Converts a list or tuple into a NumPy array.

np.zero(): Creates an array filled with zeros.

```
zeros = np.zeros((2, 3))
print(zeros)

$ C:/Users/User/AppData/LogeDrive/Desktop/python_oct/[[0. 0. 0.]]
[0. 0. 0.]]
```



np. one(): Creates an array filled with ones.

```
ones = np.ones((2, 3))
print(ones)

$ C:/Users/User/AppData/Lo
eDrive/Desktop/python_oct/
[[1. 1. 1.]]
```

np.arange(): Creates an array with a range of values.

```
arange_array = np.arange(0, 10, 2)
print(arange_array)

$ C:/Users/User/AppData/Local/ProgeDrive/Desktop/python_oct/test.py
[0 2 4 6 8]
```

np.linspace(): Creates an array with evenly spaced values.

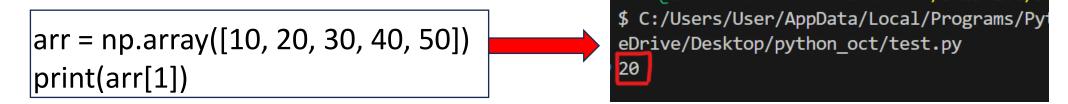
```
linspace_array = np.linspace(0, 1, 5)
print(linspace_array)

$ C:/Users/User/AppData/Local/Progra
eDrive/Desktop/python oct/test.py
[0. 0.25 0.5 0.75 1. ]
```



Accessing elements, slicing, and indexing.

Accessing elements



Slicing

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[1:4])
print(arr[:3])
print(arr[::2])

$ C:/Users/User/AppData/Local/ProgeDrive/Desktop/python_oct/test.py

[20 30 40]
[10 20 30]
[10 30 50]
```

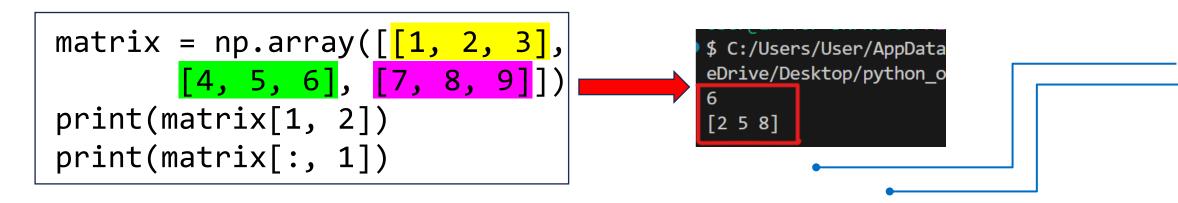


Accessing elements, slicing, and indexing

1	2	3
4	<mark>5</mark>	<mark>6</mark>
7	8	9

Indexing in Multi-dimensional Arrays

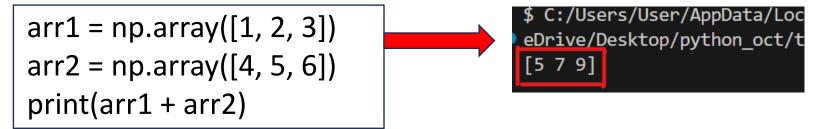
Slicing



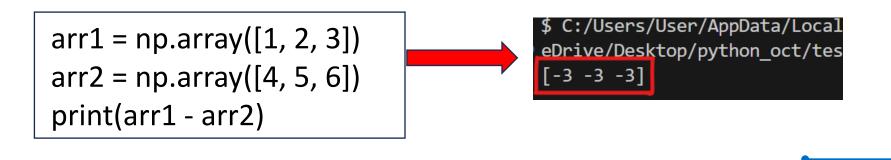
Array Arithmeic

 NumPy allows element-wise operations on arrays, making mathematical computations fast and efficient. Infotech Group

Addition:



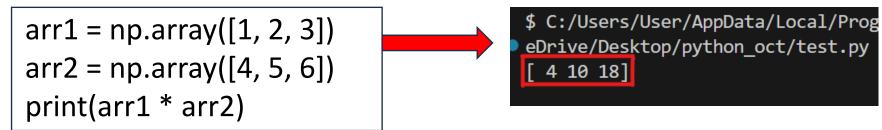
Subtraction:



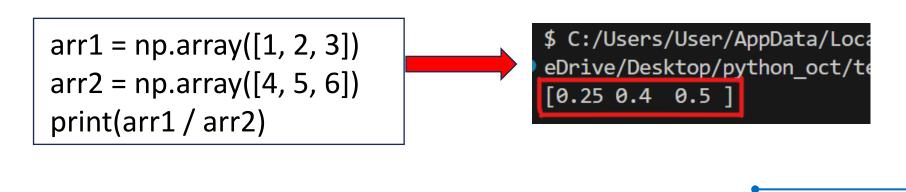


Array Arithmeic

Multiplication:



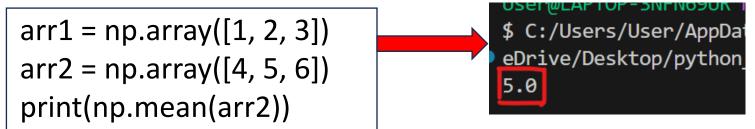
Division:



Realistic Infotech Group

- Mathematical Functions
 - NumPy provides built-in functions for common mathematical operations:

Mean: Average value.



Median: Middle value.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print(np.median(arr2))
```



Mathematical Functions

Standard Deviation: Measures the spread of data.

Sum: Total of all elements.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print(np.sum(arr2))
$ C:/Users/User/AppData/Local/Programmers
eDrive/Desktop/python_oct/test.py
15
```

Minimum and Maximum: Smallest and largest elements.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print(np.min(arr2))
print(np.max(arr2))

$ C:/Users/User/AppData/Local/PeDrive/Desktop/python_oct/test.

6

6
```

Example: Finding the Mean and Standard Deviation



```
import numpy as np
# Dataset
data = np.array([10, 20, 30, 40, 50])
# Calculating mean and standard deviation
mean = np.mean(data)
std dev = np.std(data)
print("Dataset:", data)
print("Mean:", mean) print("Standard
Deviation:", std dev)
```

\$ C:/Users/User/AppData/Local/Programs/Python
eDrive/Desktop/python_oct/test.py
Dataset: [10 20 30 40 50]
Mean: 30.0
Standard Deviation: 14.142135623730951

Standard Deviation

Formula for Standard Deviation:

For an array x with n elements:

Standard Deviation =
$$\sqrt{\frac{\sum_{i=1}^{n}(x_i - \text{mean})^2}{n}}$$

Steps for arr2:

1. Find the Mean:

$$\mathrm{Mean} = \frac{4+5+6}{3} = 5$$

2. Calculate Deviations from the Mean:

Deviations =
$$[4-5, 5-5, 6-5] = [-1, 0, 1]$$

3. Square the Deviations:

Squared Deviations =
$$[-1^2, 0^2, 1^2] = [1, 0, 1]$$

4. Find the Mean of the Squared Deviations:

Mean of Squared Deviations
$$= \frac{1+0+1}{3} = \frac{2}{3} \approx 0.6667$$

5. Take the Square Root:

Standard Deviation =
$$\sqrt{0.6667} \approx 0.8165$$



arr2 = np.array([4, 5, 6])

Example: Combining Array Creation and Operations

```
import numpy as np
# Create arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
# Perform operations
sum array = arr1 + arr2
mean value = np.mean(sum_array)
std dev = np.std(sum array)
# Print results
print("Sum Array:", sum_array)
print("Mean:", mean_value)
print("Standard Deviation:", std dev)
```



```
$ C:/Users/User/AppData/Local/Programs/Py
eDrive/Desktop/python_oct/test.py
Sum Array: [5 7 9]
Mean: 7.0
Standard Deviation: 1.632993161855452
```



Pandas Practices

What is Pandas?



Pandas is a Python library used for data manipulation and analysis. It provides high-level
data structures and tools to work efficiently with structured data, such as tabular data,
time-series data, and multidimensional data.

Key Features:

- Data structures: Series and DataFrames.
- Data cleaning, transformation, and analysis.
- Reading and writing data from different file formats (CSV, Excel, JSON, etc.).

How to import NumPy



Before importing, ensure pandas is installed in your Python environment.
 Use the following command to install it:

```
C:\Users\User>pip install pandas
Collecting pandas
Downloading pandas-2.2.3-cp313-cp313-win_amd64.whl.metadata (19 k
Requirement already satisfied: numpy>=1.26.0 in c:\users\user\appda
(from pandas) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\u
packages (from pandas) (2.9.0.post0)
Collecting pytz>=2020.1 (from pandas)
Downloading pytz-2024.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas)
Downloading tzdata-2024.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in c:\users\user\appdata\lo
python-dateutil>=2.8.2->pandas) (1.16.0)
Downloading pandas-2.2.3-cp313-cp313-win_amd64.whl (11.5 MB)
```

• To use the **pandas** library in Python, you need to **import** it. The standard way to import **pandas** is:

import pandas as pd



• A **Pandas Series** is a **one-dimensional labeled array** capable of holding data of any type (integer, float, string, etc.). It is similar to a column in a spreadsheet or a Python list, but with labels (called an **index**) that make it more powerful.

Pandas Series can be created from:

- Python lists
- NumPy arrays
- Dictionaries



Example : Creating a Series from a List

```
import pandas as pd

# Create a Series from a list
data = [10, 20, 30, 40]
series = pd.Series(data)
print(series)
$ C:/Users/User/AppData/Local/ProgreDrive/Desktop/python_oct/test.py
0 10
1 20
2 30
3 40
dtype: int64
```

Example: Adding Custom Index Labels

```
# Create a Series with custom index
data = [10, 20, 30, 40]
series = pd.Series(data, index=['a', 'b', 'c', 'd'])
print(series)
```

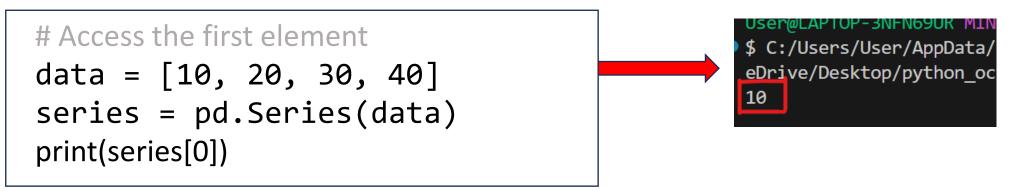
eDrive/Desktop/python_oct
a 10
b 20
c 30
d 40
dtype: int64



Indexing and Slicing

 Pandas Series allows access to elements using labels or positions, similar to Python lists.

Indexing with position



Indexing with Labels

```
# Access an element using a label
data = [10, 20, 30, 40]
series = pd.Series(data, index=['a', 'b', 'c', 'd'])
print(series['a'])
```

\$ C:/Users/User/AppData/ eDrive/Desktop/python_oc



Slicing

```
Import pandas as pd

# Access the first element
data = [10, 20, 30, 40]
series = pd.Series(data)
print(series[0])

* C:/users/user/Appua
eDrive/Desktop/pythor
b 20
c 30
dtype: int64
```

Example: Creating a Pandas Series and Accessing Elements

```
Realistic
Infotech
Group
```

```
import pandas as pd
# Create a Series
data = [100, 200, 300, 400, 500]
index labels = ['Math', 'Science',
'English','History', 'Geography']
grades = pd.Series(data,index=index labels)
# Print the Series
print("Series:")
print(grades)
# Access specific elements
print("\nGrade in Science:",grades['Science'])
print("Grade in English:", grades['English'])
```

```
# Slice a range
print("\nGrades from
Science to History:")
print(grades['Science':
'History'])
```



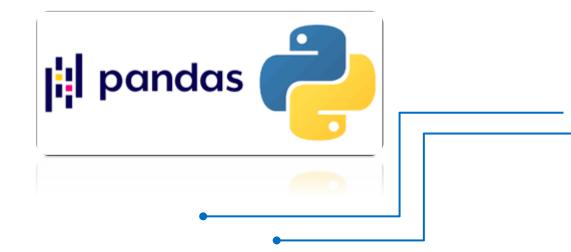
Output

```
$ C:/Users/User/AppData/Local/Programs/Pythor
eDrive/Desktop/python_oct/test.py
Series:
Math
             100
Science
             200
English
             300
History
             400
Geography
             500
dtype: int64
Grade in Science: 200
Grade in English: 300
Grades from Science to History:
Science
           200
English
           300
History
           400
dtype: int64
```

Pandas Data Frame



- A Pandas Data Frame is a two-dimensional, tabular data structure in Python that is similar
 to an Excel spreadsheet or SQL table.
- It is part of the Pandas library and is widely used for data manipulation, analysis, and visualization.





Example: Creating data frame from list

```
import pandas as pd

data = [[1, 'Alice', 22], [2, 'Bob', 25], [3, 'Charlie', 30]]

df = pd.DataFrame(data,columns=['ID', 'Name', 'Age'])

print(df)
```

```
eDrive/Desktop/python_oct/test.py

ID Name Age
0 1 Alice 22
1 2 Bob 25
2 3 Charlie 30
```



Example: Creating data frame from Dictionaries

```
eDrive/Desktop/python_oct/te
ID Name Age
0 1 Alice 22
1 2 Bob 25
2 3 Charlie 30
```

Accessing Rows and Columns



Using .loc[](label-based)

```
import pandas as pd
data = \{'ID': [1, 2, 3],
         'Name': ['Alice', 'Bob', 'Charlie'],
         'Age': [22, 25, 30]}
df = pd.DataFrame(data)
# Access a specific row
print(df.loc[1])
# Access a specific cell
print(df.loc[1, 'Name'])
```

• Ser@LAPTOP-3NFN69UR MIN
• \$ C:/Users/User/AppData/
• eDrive/Desktop/python_oc

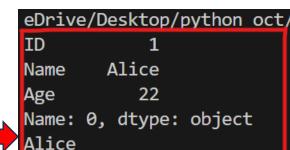
Pandas

Accessing Rows and Columns

Realistic Infotech Group

Using .iloc[](Position-based)

```
import pandas as pd
data = {'ID': [1, 2, 3],
         'Name': ['Alice', 'Bob', 'Charlie'],
         'Age': [22, 25, 30]}
df = pd.DataFrame(data)
# Access a specific row
print(df.iloc[0]) # First row
# Access a specific cell
print(df.iloc[0, 1]) # Cell at position [0, 1]
(first row, second column)
```



Loading Data into Pandas



Reading CSV, Excel, and JSON Files with Pandas		
pd.read_csv()	loads data from a CSV file.	
pd.read_excel()	loads data from an Excel file.	
pd.read_json()	loads data from a JSON file.	







Loading Data into Pandas(Loading CSV Data)



sample_data.csv

reading_data.py

Name, Age, Department, Salary Alice, 30, HR, 50000 Bob, 25, IT, 55000 Charlie, 35, Finance, 60000 David, 28, IT, 58000 Eva, 32, HR, 52000

```
import pandas as pd

# Reading CSV file

df_csv = pd.read_csv('sample_data.csv')
print("CSV Data:")
print(df_csv.head()) # Display the
first 5 rows
```

```
CSV Data:

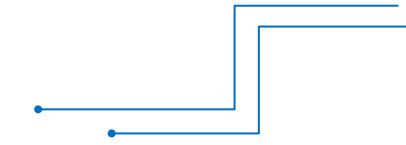
Name Age Department Salary

Bob 25 IT 55000

Charlie 35 Finance 60000

David 28 IT 58000

Eva 32 HR 52000
```



Loading Data into Pandas(Loading Excel Data)

pd.read_excel() loads data from an Excel file.

sample_data.xlxs

reading_data.py



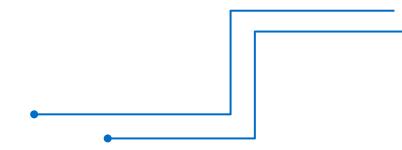
```
Department Salary
Name
         Age
Alice
                30 HR
                                   50000
Bob
                25 IT
                                   55000
Charlie
                35 Finance
                                   60000
David
                28 IT
                                   58000
Eva
                32 HR
                                   52000
              Sheet1
```

```
import pandas as pd

# Reading Excel file

df_excel = pd.read_excel('sample_data.xlsx',
    sheet_name='Sheet1')
    print("\nExcel Data:")
    print(df_excel.head())
```

```
Excel Data:
      Name
            Age Department Salary
     Alice
                              50000
       Bob
             25
                              55000
  Charlie
                   Finance
                              60000
     David
             28
                              58000
       Eva
                              52000
```



Loading Data into Pandas(Loading JSON Data)

Realistic Infotech Group

sample_data.json

```
import pandas as pd

# Reading JSON file

df_json = pd.read_json('sample_data.json')
print("\nJSON Data:")
print(df_json.head())
```

Loading Data into Pandas(Loading JSON Data)



```
JSON Data:
             Age Department
                              Salary
      Name
     Alice
0
              30
                          HR
                               50000
1
       Bob
              25
                          IT
                               55000
   Charlie
              35
                    Finance
                               60000
     David
3
              28
                          IT
                               58000
       Eva
              32
                          HR
                               52000
4
```



Explore the Data Frame	
.head(),.tail()	show the first and last 5 rows, respectively.
.info()	provides a summary of the Data Frame.
.describe()	gives a statistical summary (only for numerical columns).
.shape ()	returns the number of rows and columns.
.isnull(),.sum()	helps to identify missing values.
.fillna()	fills missing values with a specified value.
.dropna()	removes rows with missing values.



Sample Data Frame

```
# Sample DataFrame for demonstration
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 30, 35, None, 28],
    'Salary': [50000, 60000, None, 70000, 65000]
}
df = pd.DataFrame(data)
```

Display first few rows & last few rows

```
# Display first few rows
print("Head of the DataFrame:")
print(df.head())

# Display last few rows
print("\nTail of the DataFrame:")
print(df.tail())
```

```
Head of the DataFrame:
    Name Age Salary
0 Alice 25.0 50000.0
1 Bob 30.0 60000.0
2 Charlie 35.0 NaN
3 David NaN 70000.0
4 Eva 28.0 65000.0
```

```
Tail of the DataFrame:

Name Age Salary

Alice 25.0 50000.0

Bob 30.0 60000.0

Charlie 35.0 NaN

David NaN 70000.0

Eva 28.0 65000.0
```



Example

Sample Data Frame

```
# Sample DataFrame for demonstration
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 30, 35, None, 28],
    'Salary': [50000, 60000, None, 70000, 65000]
}
df = pd.DataFrame(data)
```

```
# Display first few rows
print("Head of the DataFrame:")
print(df.head())
# Display last few rows
print("\nTail of the DataFrame:")
print(df.tail())
# Display information about the
DataFrame
print("\nDataFrame Info:")
df.info()
# Display statistical summary
print("\nStatistical Summary:")
print(df.describe())
```

```
# Checking the shape of the DataFrame (rows,
columns)
print("\nShape of DataFrame:", df.shape)
# Checking for missing values
print("\nMissing Values:")
print(df.isnull().sum())
# Handling missing values
df filled = df.fillna(0)
print("\nDataFrame after filling missing values:")
print(df_filled)
df_dropped = df.dropna()
print("\nDataFrame after dropping missing
values:")
print(df dropped)
```



```
Head of the DataFrame:
                   Salary
      Name
             Age
     Alice
           25.0
                  50000.0
       Bob
           30.0
                  60000.0
   Charlie 35.0
                      NaN
     David
                  70000.0
             NaN
            28.0
                  65000.0
       Eva
Tail of the DataFrame:
      Name
             Age
                   Salary
     Alice 25.0
                  50000.0
       Bob
           30.0
                  60000.0
   Charlie 35.0
                      NaN
     David
                  70000.0
           28.0
                  65000.0
4
       Eva
```

```
Shape of DataFrame: (5, 3)
Missing Values:
Name
          0
Age
Salary
dtype: int64
DataFrame after filling missing values:
                   Salary
      Name
             Age
           25.0
     Alice
                  50000.0
           30.0
                  60000.0
       Bob
  Charlie 35.0
                      0.0
     David
                 70000.0
            28.0
                 65000.0
       Eva
DataFrame after dropping missing values:
    Name
           Age
                 Salary
   Alice 25.0
                50000.0
         30.0
                60000.0
     Bob
          28.0
                65000.0
     Eva
```

Data Cleansing



Data Cleansing		
.drop_duplicates()	removes duplicate rows from the DataFrame.	
np.percentile()	calculates a specific percentile value to identify potential outliers.	

Data Cleansing

Example: Data Cleansing

```
Realistic
Infotech
Group
```

```
import pandas as pd

data_dup = {
    'Name': ['Alice', 'Bob', 'Charlie', 'Alice', 'Bob'],
    'Age': [25, 30, 35, 25, 30]}

df_dup = pd.DataFrame(data_dup)
```

```
# Removing duplicates
df_no_duplicates =
df_dup.drop_duplicates()
print("DataFrame after removing
duplicates:")
print(df_no_duplicates)
```

```
DataFrame after removing duplicates:

Name Age

Alice 25

Bob 30

Charlie 35
```

Data Transformation

Sorting and Filtering Data

- Sorting helps to arrange data based on specific columns.
- Filtering extracts subsets of data based on conditions.

Sorting and Filtering Data		
.sort_values()	Sorts the DataFrame based on a specific column.	
.query()	filters the DataFrame based on conditions.	
.groupby().agg()	performs grouping and aggregation.	
pd.merge()	merges two DataFrames on a common key.	
pd.concat()	concatenates DataFrames along a specified axis.	



Data Transformation(Sorting and Filtering Data)

```
Realistic Infotech Group
```

```
import pandas as pd
# Sample DataFrame
data_transform = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Department': ['HR', 'Finance', 'IT', 'IT', 'HR'],
    'Salary': [50000, 60000, 55000, 70000, 65000]}
df_transform = pd.DataFrame(data_transform)
```

```
# Sorting by Salary

df_sorted = df_transform.sort_values(by='Salary', ascending=False)

print("DataFrame sorted by Salary:")

print(df_sorted)

# Filtering: IT Department with Salary > 55000

df_filtered = df_transform.query("Department == 'IT' and Salary > 55000")

print("\nFiltered DataFrame (IT Department with Salary > 55000):")

print(df_filtered)
```

Data Transformation(Sorting and Filtering Data)



```
Name Department
                       Salary
     David
                         70000
                   IT
                   HR
                         65000
       Eva
       Bob
              Finance
                         60000
   Charlie
                   IT
                         55000
     Alice
                   HR
                         50000
Filtered DataFrame (IT Department with Salary > 55000):
    Name Department Salary
   David
                 IT
                      70000
```

Data Transformation



Grouping and Aggregation

•Grouping combines rows with the same value in a column and applies aggregations.

Grouping and Aggregarion		
.groupby().agg()	performs grouping and aggregation.	

Data Transformation(Grouping and Aggregation)



```
import pandas as pd

# Sample DataFrame
data_transform = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Department': ['HR', 'Finance', 'IT', 'IT', 'HR'],
    'Salary': [50000, 60000, 55000, 70000, 65000]}
df_transform = pd.DataFrame(data_transform)
```

```
# Grouping by Department and calculating average Salary df_grouped = df_transform.groupby('Department').agg({'Salary': 'mean'}) print("\nAverage Salary by Department:") print(df_grouped)
```

Data Transformation(Grouping and Aggregation)



Data Transformation



Merging and Joining Data

Merging and Joining Data		
pd.merge()	merges two DataFrames on a common key.	
pd.concat()	concatenates DataFrames along a specified axis.	

Data Transformation(Merging and Joining Data)



```
import pandas as pd
# Sample DataFrame
data transform = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Department': ['HR', 'Finance', 'IT', 'IT', 'HR'],
    'Salary': [50000, 60000, 55000, 70000, 65000]}
df transform = pd.DataFrame(data transform)
# Additional data for merging
data additional = {
    'Name': ['Alice', 'Charlie', 'Eva'],
    'Bonus': [5000, 7000, 6000]}
df bonus = pd.DataFrame(data additional)
```

Data Transformation(Merging and Joining Data)



```
# Merging DataFrames
df_merged = pd.merge(df_transform, df_bonus, on='Name',how='left')
print("\nMerged DataFrame with Bonus:")
print(df_merged)
# Concatenating DataFrames
df_concat = pd.concat([df_transform, df_bonus], axis=0)
print("\nConcatenated DataFrame:")
print(df_concat)
```

```
Merged DataFrame with Bonus:
     Name Department Salary
                                 Bonus
     Alice
                   HR
                         50000
                                5000.0
       Bob
              Finance
                         60000
                                   NaN
   Charlie
                   IT
                         55000
                                7000.0
     David
                         70000
                                   NaN
3
                         65000
                                6000.0
       Eva
```

```
Concatenated DataFrame:
      Name Department
                        Salary
                                  Bonus
     Alice
                                    NaN
       Bob
              Finance
                                    NaN
  Charlie
                                    NaN
     David
                                    NaN
       Eva
                       65000.0
                                    NaN
     Alice
                           NaN
                                 5000.0
                  NaN
  Charlie
                  NaN
                                 7000.0
       Eva
                  NaN
```

Merging vs. Concatenating



Feature	Merging (pd.merge)	Concatenating (pd.concat)
Purpose	Combines rows based on common column values.	Stacks DataFrames (row-wise or column-wise).
Key Relationship	Requires a common column for joining.	No relationship required between DataFrames.
Handling Columns	Keeps relevant columns from both DataFrames.	Includes all columns; fills missing values with NaN .



Matplotlib Practices

What is matplotlib?



- Matplotlib is a powerful and widely-used Python library for creating visualizations and plots.
- It is especially popular for its versatility and ease of use, making it a cornerstone of the Python data visualization ecosystem.

Features of matplotlib



Wide Range of Plot Types

Line plots, scatter plots, bar plots, histograms, pie charts, box plots, heatmaps, etc.

Customization

 Control over every aspect of a plot (e.g., axis labels, ticks, colors, line styles, markers, legends).

Integration

Works seamlessly with libraries like NumPy, Pandas, and SciPy.

Publication Quality

 High-quality graphics that are suitable for research papers, presentations, and reports.

Interactivity

Supports interactive visualizations with tools like zooming and panning.

How to import matplotlib



To install the matplotlib library, which includes the pyplot module, you can
use the Python package manager pip.

```
C:\Users\User>
pip install matplotlib

Collecting matplotlib-3.9.2-cp313-cp313-win_amd64.whl.metadata (11 | B)

Collecting contourpy>=1.0.1 (from matplotlib)

Downloading contourpy-1.3.1-cp313-cp313-win_amd64.whl.metadata (5.4 | B)

Collecting cycler>=0.10 (from matplotlib)

Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)

Collecting fonttools>=4.22.0 (from matplotlib)

Downloading fonttools-4.55.0-cp313-cp313-win_amd64.whl.metadata (167 kB)

Collecting kiwisolver>=1.3.1 (from matplotlib)

Downloading kiwisolver-1.4.7-cp313-cp313-win_amd64.whl.metadata (6.4 kB)

Requirement already satisfied: numpy>=1.23 in c:\users\user\appdata\logalprograms\python\python313\lib\site-packages (from matplotlib) (2.1.3)
```

 To use the matplotlib library in Python, you need to import it. The standard way to import matplotlib is:

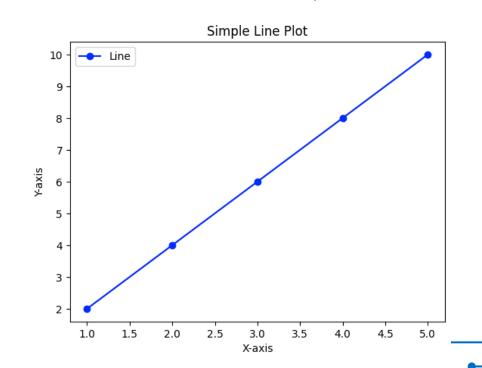
```
import matplotlib.pyplot as plt
```

Matplotlib Example

```
import matplotlib.pyplot as plt
```

```
Infotech
```

```
# Data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
# Create a plot
plt.plot(x, y, label='Line', color='blue', marker='o')
# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
plt.legend()
# Display the plot
plt.show()
```



What is Seaborn?



- Seaborn is a Python data visualization library built on top of Matplotlib.
- It provides an interface for creating attractive and informative statistical graphics with less effort and more features compared to Matplotlib.
- Seaborn is especially useful for visualizing complex relationships between variables and exploring datasets in a statistical context.

Features of Seaborn



- High-Level Interface
 - Simplifies the creation of complex visualizations compared to Matplotlib.
- Built-in Themes
 - Automatically applies aesthetically pleasing styles to plots.
- Statistical Visualizations
 - Provides specialized tools for visualizing distributions, categorical data, and relationships between variables.

Features of Seaborn



Integration with Pandas

 Works seamlessly with Pandas DataFrames, making it easier to visualize tabular data.

Data Aggregation and Summarization

 Automatically calculates means, confidence intervals, and other statistical summaries for visualization.

Customizability

 Offers customization options and works well with Matplotlib for further finetuning.

How to import Seaborn

Realistic Infotech Group

To install the Seaborn library in Python, you can use the Python package

manager **pip**.

```
C:\Users\User>pip install seaborn
Collecting seaborn
     Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\user\
pdata\local\programs\python\python313\lib\site-packages (from seabor
 (2.1.3)
Requirement already satisfied: pandas>=1.2 in c:\users\user\appdata\
al\programs\python\python313\lib\site-packages (from seaborn) (2.2.3
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\unders\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\users\user
\appdata\local\programs\python\python313\lib\site-packages (from sea
n) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\user\user\appo
a\local\programs\python\python313\lib\site-packages (from matplotlib
 .6.1,>=3.4->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\user\user\appdata
cal\programs\python\python313\lib\site-packages (from matplotlib!=3.
 >=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\user\ap
ta\local\programs\pvthon\pvthon313\lib\site-packages (from matplotli
```

• To use the **Seaborn** library in Python, you need to **import** it. The standard way to import **Seaborn** is:

import seaborn as sns

Realistic Infotech Group

Calculating Correlation Using .corr()

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Sample data
data = {
    'Age': [25, 30, 45, 50, 60],
    'Salary': [50000, 60000, 80000, 90000, 120000],
    'Experience': [2, 4, 10, 12, 15]
df = pd.DataFrame(data)
```

Calculating Correlation using .corr()

```
# Calculate correlation matrix
correlation matrix = df.corr()
print("Correlation Matrix:")
print(correlation matrix)
```

```
Output
Correlation Matrix:
                                Experience
                        Salary
Age
            1.000000
                      0.982273
                                  0.998280
Salary
                      1.000000
                                  0.969905
Experience
            0.998280
                      0.969905
                                  1.000000
```

Rigure 1

Infotech

Group

```
Correlation Heatmap
# Visualizing correlation
                                                                      Age
-
                                                                                   0.98
                                                                                                    0.995
sns.heatmap(correlation matrix,annot=True,
cmap='coolwarm')
                                                                                                    - 0.990
plt.title('Correlation Heatmap')
                                                                      salary
                                                                           0.98
                                                                                            0.97
                                                                                                    - 0.985
plt.show()
                                                                                                    - 0.980
                                                                                   0.97
                                                                                                    0.975
                                                                                   Salary
                                                                           Age
                                                                                          Experience
                                                                   ☆ ◆ → ↓ Q 至 🖺
```

Realistic Infotech Group

Using NumPy Functions with Pandas DataFrames

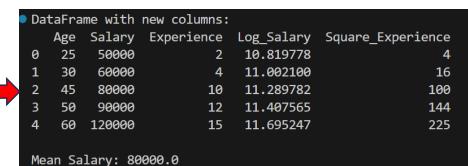
```
import pandas as pd
import numpy as np
# Sample data
data = {
    'Age': [25, 30, 45, 50, 60],
    'Salary': [50000, 60000, 80000, 90000, 120000],
    'Experience': [2, 4, 10, 12, 15]}
df = pd.DataFrame(data)
```

Using NumPy Functions with Pandas DataFrames

```
#Adding a new column with NumPy functions
df['Log Salary'] = np.log(df['Salary'])
df['Square Experience'] =
np.square(df['Experience'])
print("DataFrame with new columns:")
print(df)
# Calculate mean and standard deviation
mean salary = np.mean(df['Salary'])
std salary = np.std(df['Salary'])
print("\nMean Salary:", mean_salary)
print("Standard Deviation of Salary:",
std salary)
```



Output



Standard Deviation of Salary: 24494.89742783178

Quick Visualizations with .plot(), .hist(), .boxplot()



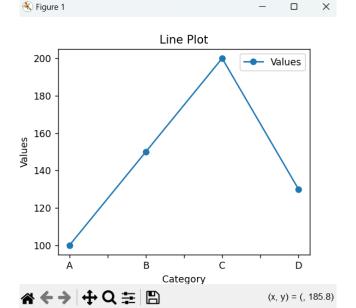
Sample data

```
import pandas as pd
import matplotlib.pyplot as plt
# Sample data
data = {
    'Age': [25, 30, 45, 50, 60],
    'Salary': [50000, 60000, 80000, 90000, 120000],
    'Experience': [2, 4, 10, 12, 15]}
df = pd.DataFrame(data)
```

Realistic Infotech Group

Quick Visualizations with Line Plot(.plot())

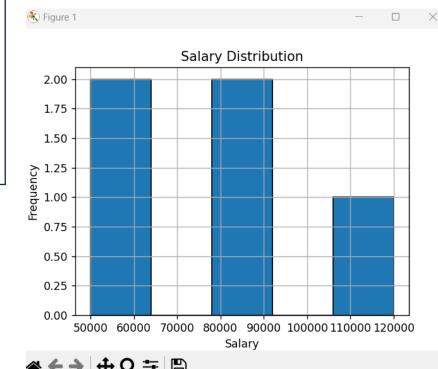
```
# Line Plot
df_viz.plot(x='Category', y='Values', kind='line',
marker='o')
plt.title('Line Plot')
plt.ylabel('Values')
plt.show()
```





Quick Visualizations with Line Plot(.hist())

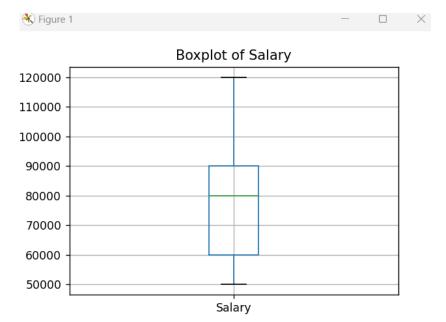
```
# Histogram
plt.figure()
df['Salary'].hist(bins=5, edgecolor='black')
plt.title('Salary Distribution')
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.show()
```



Quick Visualizations with Line Plot(.boxplot())

```
# Boxplot
plt.figure()
df.boxplot(column=['Salary'])
plt.title('Boxplot of Salary')
plt.show()
```











Realistic Infotech Group
IT Training & Services
No.79/A, First Floor
Corner of Insein Road and
Damaryon Street
Quarter (9), Hlaing Township
Near Thukha Bus Station
09256675642, 09953933826
http://www.rig-info.com