AI with Python – 33 Labs Outline Manual

University of Kotli, AJK

Department of CS&IT

🔰 **Part I: Python Basics for AI (Labs 1–8)**

1. Lab 01 – Python Setup and Jupyter Notebooks
2. Lab 02 – Data Types, Variables, and Input/Output
3. Lab 03 – Control Structures (if, for, while)
4. Lab 04 – Functions and Lambda Functions
5. Lab 05 – Lists, Tuples, Dictionaries
6. Lab 06 – File Handling (CSV & TXT files)
7. Lab 07 – Exception Handling
8. Lab 08 – Working with Libraries (numpy, matplotlib, pandas)

🧪 **Part II: Data Handling & Visualization (Labs 9–13)**

9. Lab 09 – DataFrames in pandas
10. Lab 10 – Data Cleaning and Missing Value Handling
11. Lab 11 – Data Normalization and Scaling
12. Lab 12 – Data Visualization with matplotlib and seaborn
13. Lab 13 – Exploratory Data Analysis (EDA)

🤖 **Part III: Machine Learning with scikit-learn (Labs 14–25)**

14. Lab 14 – Introduction to Machine Learning
15. Lab 15 – Linear Regression with scikit-learn
16. Lab 16 – Logistic Regression
17. Lab 17 – K-Nearest Neighbors (KNN)
18. Lab 18 – Decision Trees
19. Lab 19 – Random Forest
20. Lab 20 – Naive Bayes
21. Lab 21 – Support Vector Machines (SVM)
22. Lab 22 – Model Evaluation: Accuracy, Precision, Recall, F1
23. Lab 23 – Confusion Matrix

24. Lab 24 – Cross-validation
25. Lab 25 – Model Saving and Loading with joblib or pickle

🧠 **Part IV: Deep Learning Basics with TensorFlow / Keras (Labs 26–30)**

26. Lab 26 – Introduction to Neural Networks
27. Lab 27 – Building a Simple ANN using Keras
28. Lab 28 – Activation Functions and Optimizers
29. Lab 29 – Overfitting and Regularization
30. Lab 30 – Image Classification with CNN (MNIST Dataset)

👨‍🔧 **Part V: Final Project-Based Labs (Labs 31–33)**

31. Lab 31 – Mini Project 1: Predict House Prices
32. Lab 32 – Mini Project 2: Handwritten Digit Recognition
33. Lab 33- Building an AI Agent

✅ Lab 1: Introduction to Python and IDE Setup

Objective:
Set up Python environment and run the first Python script using an IDE (like IDLE, PyCharm, VS Code, or Jupyter Notebook).

Theory Overview:
Python is a high-level, interpreted programming language known for its readability and versatility. Before writing Python code, we need to install Python and choose an IDE or code editor.

Tasks/Instructions:

1. Install the latest version of Python from python.org.
2. Install any one IDE (IDLE comes with Python, or install PyCharm/VS Code/Jupyter).
3. Open the IDE and create a new Python file.
4. Write a program to print your name and your course title.
5. Run the program and observe the output.

 # My first Python program

print("Name: Ali")

print("Course: Introduction to Programming with Python")

Output:

Name: Ali

Course: Introduction to Programming with Python

Assessment Questions:

1. What is the purpose of an IDE?
2. What are the benefits of using Python?
3. Write and run a Python program that displays your favorite quote.

✅ Lab 2: Simple Python Statements and Variables

Objective:
Learn how to declare variables, assign values, and use basic print statements in Python.

Theory Overview:
In Python, variables are used to store data. Unlike some other languages, Python doesn't require you to declare a variable type beforehand—types are inferred dynamically.

Basic syntax:

variable_name = value

Data types include:

- int – integers
- float – decimal numbers
- str – strings (text)
- bool – boolean values (True/False)

The print() function displays output to the screen.

Tasks/Instructions:

1. Declare variables for your name, age, and GPA.
2. Print each variable on a separate line using the print() function.
3. Update the age variable by adding 1 and print the new age.
4. Try printing a sentence using string concatenation or formatting.
5. Use the type() function to print the data type of each variable.

Sample Code:

```
# Variable assignment

name = "Ali"

age = 20
```

```
gpa = 3.5

is_student = True

# Printing values

print("Name:", name)

print("Age:", age)

print("GPA:", gpa)

print("Student status:", is_student)

# Updating and printing

age += 1

print("New Age:", age)

# Type checking

print("Type of age:", type(age))

print("Type of name:", type(name))
```

Assessment Questions:

1. What is the output of type(3.14) in Python?
2. Create a variable called country and assign your country name. Print a full sentence using it.
3. What happens if you assign a string to a variable and then later assign an integer to the same variable? Try it and observe the output.

🧪 Lab 02 (b) – Operators and Expressions

🎯 Objective:

To understand and use various operators in Python including arithmetic, comparison, logical, assignment, and bitwise operators.

🧠 Theory:

Python supports a rich set of operators:

- **Arithmetic Operators**: +, -, *, /, //, %, **
- **Comparison Operators**: ==, !=, >, <, >=, <=
- **Logical Operators**: and, or, not
- **Assignment Operators**: =, +=, -=, *=, /=
- **Bitwise Operators**: &, |, ^, ~, <<, >>

Expressions combine variables, constants, and operators to produce a new value.

✅ Tasks to Perform:

Task 1: Arithmetic Operations

```
a = 15

b = 4

print("Addition:", a + b)

print("Subtraction:", a - b)

print("Multiplication:", a * b)

print("Division:", a / b)

print("Floor Division:", a // b)

print("Modulus:", a % b)

print("Exponent:", a ** b)
```

Task 2: Comparison Operations

```
x = 10

y = 20

print("x == y:", x == y)

print("x != y:", x != y)

print("x > y:", x > y)

print("x <= y:", x <= y)
```

Task 3: Logical Operations

```
print(True and False)

print(True or False)

print(not True)
```

Task 4: Assignment Operators

```
z = 5

z += 3

print("After += :", z)

z *= 2

print("After *= :", z)
```

Task 5: Bitwise Operators (Optional but useful for AI logic gates)

```
m = 5    # 0101

n = 3    # 0011

print("Bitwise AND:", m & n)

print("Bitwise OR:", m | n)

print("Bitwise XOR:", m ^ n)

print("Bitwise NOT (~m):", ~m)
```

……

❓ **Assessment Questions:**

What is the difference between == and = in Python?

What is the output of 5 // 2?

What does the operator ** do?

🧪 Lab 03 (a) – Conditional Statements (if, elif, else)

🎯 Objective:

To learn how to make decisions in Python using conditional statements (if, elif, else).

🧠 Theory:

Conditional statements are used to execute code only if a certain condition is true.

Syntax:

```
if condition:
    # code block
elif another_condition:
    # another code block
else:
    # code block if all conditions fail
```

Indentation is very important in Python to define blocks of code.

✅ Tasks to Perform:

Task 1: Basic if Statement

```
age = int(input("Enter your age: "))

if age >= 18:
    print("You are eligible to vote.")
```

Task 2: if-else Statement

```
marks = int(input("Enter your marks: "))

if marks >= 50:
    print("You passed the exam.")
else:
    print("You failed the exam.")
```

Task 3: if-elif-else Ladder

```
score = int(input("Enter your score: "))

if score >= 90:
    print("Grade: A")
elif score >= 80:
```

```
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: F")
```

Task 4: Nested if Statements

```
x = int(input("Enter a number: "))

if x > 0:
    if x % 2 == 0:
        print("Positive Even Number")
    else:
        print("Positive Odd Number")
else:
    print("Non-positive Number")
```

❓ **Assessment Questions:**

1. What is the difference between if and elif?
2. What happens if indentation is not maintained in Python?
3. Write a Python program that checks whether a number is positive, negative, or zero.

🧪 Lab 03 (b) – Loops in Python (for and while)

🎯 Objective:

To understand how to use looping structures (for and while) in Python for repeating tasks.

🧠 Theory:

Loops are used to execute a block of code repeatedly.

- **for loop**: Iterates over a sequence (like a list, tuple, dictionary, set, or string).
- **while loop**: Repeats as long as a given condition is true.

✅ Tasks to Perform:

Task 1: for Loop with Range

```
print("Printing numbers from 1 to 5:")
for i in range(1, 6):
    print(i)
```

Task 2: Loop Through a List

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print("I like", fruit)
```

Task 3: while Loop Example

```
count = 1
while count <= 5:
    print("Count is:", count)
    count += 1
```

Task 4: Nested Loops

```
for i in range(1, 4):
    for j in range(1, 4):
        print(f"({i}, {j})", end=" ")
    print()  # New line after inner loop
```

Task 5: Loop with break and continue

```
print("Break at 3:")
for i in range(1, 6):
    if i == 3:
        break
    print(i)

print("Continue at 3:")
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

📤 **Expected Output:**

- Sequence of numbers printed with various conditions and controls.
- Output of nested loops and control statements demonstrated.

❓ **Assessment Questions:**

1. What is the purpose of the range() function in loops?
2. How is a while loop different from a for loop?
3. What do break and continue do in loops?

✅ **Lab 3 (c): Break, Continue, and Pass Statements**

Objective:

Understand how to control loop behavior using break, continue, and pass statements in Python.

Theory Overview:

- **break**: Immediately exits the loop.
- **continue**: Skips the rest of the current loop iteration and goes to the next iteration.
- **pass**: Does nothing. Used as a placeholder.

Tasks/Instructions:

1. Write a loop that prints numbers 1 to 10, but stops when it reaches 5 using break.
2. Write a loop that prints odd numbers between 1 and 10, using continue.
3. Use pass inside an if statement as a placeholder without any action.

Sample Code:

Using break:

```
for i in range(1, 11):
    if i == 5:
        break
    print(i)
```

Using continue:

```
for i in range(1, 11):
    if i % 2 == 0:
        continue
    print(i)
```

Using pass:

```
for i in range(5):
    if i == 3:
        pass  # Placeholder for future code
    print("Processing", i)
```

Assessment Questions:

1. What is the difference between break and continue in Python?
2. What will be the output if break is removed from the first example?
3. Write a program that reads a number from a user. If the number is negative, pass it; otherwise, print it.

🧪 Lab 04 – Functions and Lambda Expressions

🎯 Objective:

To learn how to define and use functions in Python, including anonymous (lambda) functions.

🧠 Theory:

Functions help in organizing code into reusable blocks.

- **Defining a Function:**

```
def function_name(parameters):
    # code block
    return value
```

Calling a Function:

```
function_name(arguments)
```

Lambda Function: A small anonymous function using the lambda keyword.

```
lambda arguments: expression
```

✅ Tasks to Perform:

Task 1: Define and Call a Simple Function

```
def greet():
    print("Hello, AI enthusiast!")

greet()
```

Task 2: Function with Parameters and Return

```
def add(a, b):
    return a + b

result = add(5, 3)
print("Sum:", result)
```

Task 3: Function with Default Parameters

```
def multiply(a, b=2):
    return a * b

print(multiply(4))       # Uses default b
print(multiply(4, 5))    # Uses given b
```

Task 4: Lambda Function for Square

```
square = lambda x: x * x
print("Square of 6 is:", square(6))
```

Task 5: Lambda with map()

```
numbers = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x**2, numbers))
print("Squares:", squares)
```

❓ **Assessment Questions:**

What is the benefit of using functions in Python?
What is a lambda function? How is it different from a regular function?
Write a lambda function to cube a number.

🧪 Lab 05 – Lists, Tuples, and Sets

🎯 Objective:

To understand how to create and manipulate collections like lists, tuples, and sets in Python.

🧠 Theory:

- **List**: Ordered, mutable, allows duplicates.
  Example: my_list = [1, 2, 3]
- **Tuple**: Ordered, immutable, allows duplicates.
  Example: my_tuple = (1, 2, 3)
- **Set**: Unordered, mutable, no duplicates.
  Example: my_set = {1, 2, 3}

✅ Tasks to Perform:

Task 1: List Creation and Operations

```
fruits = ['apple', 'banana', 'cherry']
fruits.append('mango')
fruits.remove('banana')
print("Fruits List:", fruits)
print("First Fruit:", fruits[0])
```

Task 2: Tuple Access and Slicing

```
colors = ('red', 'green', 'blue', 'yellow')
print("All Colors:", colors)
print("Second Color:", colors[1])
print("Slice Colors:", colors[1:3])
```

Task 3: Set Operations

```
a = {1, 2, 3}
b = {3, 4, 5}

print("Union:", a | b)
print("Intersection:", a & b)
print("Difference (a - b):", a - b)
```

Task 4: List Comprehension

```
squares = [x**2 for x in range(6)]
print("Squares using list comprehension:", squares)
```

Task 5: Tuple Packing and Unpacking

```
point = (10, 20)
x, y = point
print("x =", x, ", y =", y)
```

❓ **Assessment Questions:**

1. How is a tuple different from a list?
2. Why are sets useful when working with unique elements?
3. Write Python code to remove duplicates from a list using a set.

🧪 Lab 05 (b) – Dictionaries and JSON Basics

🎯 Objective:

To understand how to use dictionaries in Python and how to work with basic JSON data.

🧠 Theory:

- **Dictionary**: Unordered collection of key-value pairs.
  Syntax: my_dict = {'key1': 'value1', 'key2': 'value2'}
- **JSON** (JavaScript Object Notation): A standard format for data exchange. Python uses the json module to handle JSON data.

✅ Tasks to Perform:

Task 1: Create and Access a Dictionary

```
student = {'name': 'Ali', 'age': 21, 'course': 'AI'}
print("Name:", student['name'])
print("Age:", student.get('age'))
```

Task 2: Add, Update, and Delete Elements

```
student['grade'] = 'A'
student['age'] = 22
del student['course']
print("Updated Student:", student)
```

Task 3: Loop Through Dictionary

```
for key, value in student.items():
    print(key, ":", value)
```

Task 4: Convert Dictionary to JSON and Back

```
import json

data_dict = {'id': 1, 'name': 'Sara', 'score': 95}
json_data = json.dumps(data_dict)  # Convert to JSON
print("JSON:", json_data)

parsed_data = json.loads(json_data)  # Convert back to dict
```

```
print("Parsed Dictionary:", parsed_data)
```

Task 5: Nested Dictionary Example

```
records = {
   'student1': {'name': 'Ali', 'marks': 85},
   'student2': {'name': 'Sara', 'marks': 90}
}
print("Student1 Name:", records['student1']['name'])
```

❓ **Assessment Questions:**

What is a dictionary in Python? How is it different from a list?
How do you safely access dictionary values?
Convert a dictionary {'x': 10, 'y': 20} into a JSON string.

✅ Lab 6 – File Handling (CSV & TXT Files)

Objective:
Learn how to read from and write to text (.txt) and CSV (.csv) files using Python.

Theory Overview:

- **Text Files**: Regular files containing plain text.
- **CSV Files**: "Comma Separated Values" files where data is organized in rows and columns, separated by commas.

Python provides built-in functions like open(), read(), write(), and modules like csv for handling files.

Tasks/Instructions:

1. Create a .txt file and write some text into it.
2. Read the contents of a .txt file and display it.
3. Create a .csv file and write multiple rows of data (e.g., student names and marks).
4. Read the .csv file and print its contents row by row.

Sample Code:

Text File Handling:

```
# Writing to a text file
with open("sample.txt", "w") as file:
    file.write("Hello, this is a sample text file.\nWelcome to Python File Handling!")

# Reading from a text file
with open("sample.txt", "r") as file:
    content = file.read()
    print(content)
```

CSV File Handling:

```
import csv

# Writing to a CSV file
with open("students.csv", "w", newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Name", "Marks"])
```

```
    writer.writerow(["Ali", 90])
    writer.writerow(["Sara", 85])

# Reading from a CSV file
with open("students.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Expected Output:

Text File Reading Output:

Hello, this is a sample text file.
Welcome to Python File Handling!

CSV File Reading Output:

['Name', 'Marks']
['Ali', '90']
['Sara', '85']

Assessment Questions:

1. What is the difference between text and CSV files?
2. How do you open a file for appending new data?
3. Write Python code to add another student's record to the existing students.csv file.

✅ Lab 7 – Exception Handling

Objective:

Learn how to handle runtime errors using try, except, else, and finally blocks in Python.

Theory Overview:

- **Exception**: An error that occurs during program execution.
- **Handling exceptions** prevents the program from crashing.

Structure:

```
try:
    # Code that may raise an error
except ExceptionType:
    # Code that runs if an error occurs
else:
    # Code that runs if no error occurs
finally:
    # Code that always runs
```

Tasks/Instructions:

1. Handle division by zero using exception handling.
2. Handle invalid user input (e.g., entering text when expecting a number).
3. Use finally to display a message that the program has ended.

Sample Code:

```
# Division example with exception handling
try:
    a = int(input("Enter numerator: "))
    b = int(input("Enter denominator: "))
    result = a / b
except ZeroDivisionError:
    print("Error: Division by zero is not allowed!")
except ValueError:
    print("Error: Please enter valid integers!")
else:
    print("Result:", result)
```

```
finally:
    print("Program finished!")
```

Expected Output (Example 1 - Division by zero):

```
Enter numerator: 10
Enter denominator: 0
Error: Division by zero is not allowed!
Program finished!
```

Expected Output (Example 2 - Valid input):

```
Enter numerator: 10
Enter denominator: 2
Result: 5.0
Program finished!
```

Assessment Questions:

1. What will happen if an exception is not handled in Python?
2. What is the role of the finally block?
3. Write a program that catches an IndexError when accessing a wrong list index.

✅ Lab 8 – Working with Libraries (NumPy, Matplotlib, Pandas)

Objective:
Understand how to use popular Python libraries: **NumPy** for arrays, **Matplotlib** for plotting, and **Pandas** for data manipulation.

Theory Overview:

- **NumPy**: Supports powerful numerical arrays and matrix operations.
- **Matplotlib**: Used for data visualization.
- **Pandas**: Great for working with structured data (DataFrames).

Installation (if needed):

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print("Array:", arr)
print("Sum:", np.sum(arr))
print("Mean:", np.mean(arr))
```

Matplotlib Example:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [2, 4, 6, 8]

plt.plot(x, y)
plt.title("Simple Line Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Pandas Example:

```
import pandas as pd

data = {
   'Name': ['Ali', 'Sara', 'Ahmed'],
```

```
    'Marks': [85, 90, 78]
}

df = pd.DataFrame(data)
print(df)
```

Expected Output:

NumPy Output:

```
Array: [1 2 3 4 5]
Sum: 15
Mean: 3.0
```

Matplotlib Output: (A simple straight line graph will be displayed.)

Pandas Output:

```
   Name  Marks
0   Ali     85
1  Sara     90
2 Ahmed     78
```

Assessment Questions:

1. What is the difference between a list and a NumPy array?
2. How can you plot a bar graph using Matplotlib?
3. Create a DataFrame for 5 students with their marks in three subjects.

🧪 Part II: Data Handling & Visualization (Labs 9–13)

✅ Lab 9 – DataFrames in pandas

Objective:

Understand how to create, modify, and access DataFrames using pandas in Python.

Theory Overview:

- **DataFrame: A 2D table of data with labeled rows and columns.**
- **Pandas provides easy ways to create and work with DataFrames.**

Tasks/Instructions:

1. **Create a DataFrame with student names, ages, and marks.**
2. **Access specific columns and rows.**
3. **Add a new column for pass/fail status based on marks.**

Sample Code:

```
import pandas as pd

# Creating a DataFrame
data = {
   'Name': ['Ali', 'Sara', 'Ahmed'],
   'Age': [20, 21, 19],
   'Marks': [85, 90, 78]
}

df = pd.DataFrame(data)

# Display the DataFrame
print(df)

# Access specific columns
print(df['Name'])

# Add new column
df['Status'] = ['Pass' if mark >= 80 else 'Fail' for mark in df['Marks']]
print(df)
```

Expected Output:

```
  Name  Age  Marks
0   Ali   20     85
1  Sara   21     90
2 Ahmed   19     78

0    Ali
1   Sara
2  Ahmed
Name: Name, dtype: object

   Name  Age  Marks Status
0   Ali   20     85   Pass
1  Sara   21     90   Pass
2 Ahmed   19     78   Fail
```

Assessment Questions:

How do you access multiple columns together in pandas?

How can you add a new row to a DataFrame?

✅ Lab 10 – Data Cleaning and Handling Missing Values

Objective:
Learn how to find and handle missing data in a pandas DataFrame.

Theory Overview:

- Missing values are often represented as NaN in pandas.
- Methods for handling missing data:
  - dropna(): Remove missing data.
  - fillna(): Replace missing data with a value.

Tasks/Instructions:

1. Create a DataFrame with missing values.
2. Detect missing values.
3. Fill missing values with a specific value.
4. Drop rows containing missing values.

Sample Code:

```
import pandas as pd
import numpy as np

data = {
   'Name': ['Ali', 'Sara', 'Ahmed', 'Mona'],
   'Marks': [85, np.nan, 78, np.nan]
}

df = pd.DataFrame(data)

# Detect missing values
print(df.isnull())

# Fill missing values
df_filled = df.fillna(0)
print(df_filled)

# Drop missing values
df_dropped = df.dropna()
```

```
print(df_dropped)
```

Expected Output:

```
   Name  Marks
0   Ali   85.0
1  Sara    NaN
2 Ahmed   78.0
3  Mona    NaN

   Name  Marks
0   Ali   85.0
1  Sara   0.0
2 Ahmed  78.0
3  Mona  0.0

   Name  Marks
0   Ali  85.0
2 Ahmed  78.0
```

Assessment Questions:

What is the difference between fillna() and dropna()?
How would you fill missing values with the column average?

✅ Lab 11 – Data Normalization and Scaling

Objective:
Understand how to normalize and scale data for machine learning and data analysis.

Theory Overview:

- **Normalization**: Rescaling values to a range (like 0 to 1).
- **Standardization**: Making data have a mean of 0 and standard deviation of 1.

Methods:

- Using MinMaxScaler for normalization.
- Using StandardScaler for standardization.

Tasks/Instructions:

1. Normalize a dataset using Min-Max scaling.
2. Standardize a dataset using StandardScaler.

Sample Code:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

data = {'Score': [100, 80, 60, 40, 20]}
df = pd.DataFrame(data)

# Normalize
scaler = MinMaxScaler()
df['Normalized'] = scaler.fit_transform(df[['Score']])

# Standardize
scaler_std = StandardScaler()
df['Standardized'] = scaler_std.fit_transform(df[['Score']])

print(df)
```

Expected Output:

```
   Score  Normalized  Standardized
0    100        1.00      1.414214
```

1	80	0.75	0.707107
2	60	0.50	0.000000
3	40	0.25	-0.707107
4	20	0.00	-1.414214

Assessment Questions:

What is the need for data normalization?

How do Min-Max Scaling and Standardization differ?

✅ Lab 12 – Data Visualization with matplotlib and seaborn

Objective:
Create various types of plots for data visualization using Matplotlib and Seaborn libraries.

Theory Overview:

- **Matplotlib**: Basic plotting (line, bar, scatter).
- **Seaborn**: Advanced visualization (heatmaps, histograms, boxplots).

Tasks/Instructions:

1. Create a line plot and a bar plot using Matplotlib.
2. Create a heatmap using Seaborn.

Sample Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Line Plot
x = [1, 2, 3, 4]
y = [10, 20, 30, 40]
plt.plot(x, y)
plt.title('Line Plot Example')
plt.show()

# Bar Plot
plt.bar(x, y)
plt.title('Bar Plot Example')
plt.show()

# Heatmap
data = [[1, 2], [3, 4]]
sns.heatmap(data, annot=True)
plt.title('Heatmap Example')
plt.show()
```

Expected Output:

Line graph showing (x, y) relation.

Bar chart for same data.

Heatmap displaying matrix values.

Assessment Questions:

What is the difference between Matplotlib and Seaborn?
What type of plot is most suitable for showing correlation between two variables?

✅ Lab 13 – Exploratory Data Analysis (EDA)

Objective:
Perform initial data investigation to discover patterns, detect outliers, and check assumptions.

Theory Overview:

- **EDA**: Summarizing the main characteristics of a dataset using statistics and visualization.
- Tools: describe(), info(), histograms, scatter plots.

Tasks/Instructions:

1. Load a dataset (use built-in datasets like Iris or Titanic from seaborn).
2. Generate summary statistics.
3. Plot histograms and scatter plots.
4. Find correlation between variables.

Sample Code:

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Load Dataset
df = sns.load_dataset('iris')

# Summary Statistics
print(df.describe())

# Info about data
print(df.info())

# Histograms
df.hist(figsize=(10,8))
plt.show()

# Scatter plot
sns.scatterplot(x='sepal_length', y='sepal_width', data=df)
plt.title('Sepal Length vs Width')
plt.show()
```

```
# Correlation matrix
print(df.corr())
sns.heatmap(df.corr(), annot=True)
plt.title('Correlation Heatmap')
plt.show()
```

Expected Output:

Printed summary and data info.

Histograms for each numeric column.

Scatter plot for two features.

Heatmap showing feature correlations.

Assessment Questions:

What steps are involved in Exploratory Data Analysis?
Why is it important to check correlations during EDA?

🤖 Part III: Machine Learning with scikit-learn (Labs 14–25)

✅ Lab 14 – Introduction to Machine Learning

Objective:
Understand the basics of Machine Learning and its categories.

Theory Overview:

- **Machine Learning (ML)**: Ability of computers to learn from data without being explicitly programmed.
- Categories:
  - **Supervised Learning** (e.g., regression, classification)
  - **Unsupervised Learning** (e.g., clustering, association)
  - **Reinforcement Learning** (learning via rewards)

Tasks/Instructions:

1. Install scikit-learn library.
2. Import basic modules.
3. Load a simple dataset (like Iris).

Sample Code:

```
# Install scikit-learn if needed:
# pip install scikit-learn

from sklearn import datasets

# Load iris dataset
iris = datasets.load_iris()
print(iris.keys())
```

Expected Output:

A dictionary-like structure containing dataset attributes (data, target, feature_names, etc.)

Assessment Questions:

What are the three main types of machine learning?
Name two examples of supervised learning problems.

✅ Lab 15 – Linear Regression with scikit-learn

Objective:
Learn to apply simple linear regression using scikit-learn.

Theory Overview:

- **Linear Regression**: Predicts a continuous dependent variable using one or more independent variables.

Tasks/Instructions:

1. Create a dataset.
2. Train a Linear Regression model.
3. Predict and plot results.

Sample Code:

```
mport numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Data
x = np.array([5, 15, 25, 35, 45, 55]).reshape(-1, 1)
y = np.array([5, 20, 14, 32, 22, 38])

# Model
model = LinearRegression()
model.fit(x, y)

# Prediction
y_pred = model.predict(x)

# Plot
plt.scatter(x, y)
plt.plot(x, y_pred, color='red')
plt.show()

print("Coefficient:", model.coef_)
print("Intercept:", model.intercept_)
```

Expected Output:

Scatter plot with regression line.

Printed slope and intercept values.

Assessment Questions:

What is the purpose of linear regression?
How do you interpret the coefficients in a linear model?

✅ Lab 16 – Logistic Regression

Objective:
Perform binary classification using Logistic Regression.

Theory Overview:

- **Logistic Regression**: Predicts a binary outcome (0 or 1).

Tasks/Instructions:

1. Use the Iris dataset.
2. Build a logistic regression model.
3. Predict and evaluate.

Sample Code:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load Data
iris = load_iris()
X = iris.data
y = (iris.target == 0).astype(int)  # binary classification: is Setosa?

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Expected Output:

Printed classification accuracy.

Assessment Questions:

What kind of output does logistic regression give?
How can you handle multi-class problems with logistic regression?

✅ Lab 17 – K-Nearest Neighbors (KNN)

Objective:
Use KNN for classification.

Theory Overview:

- **KNN**: Instance-based learning method, predicts based on the majority vote of nearest neighbors.

Tasks/Instructions:

1. Train a KNN classifier on Iris dataset.
2. Predict and visualize.

Sample Code:

```
rom sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Data
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2)

# Model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predict
print("Accuracy:", knn.score(X_test, y_test))
```

Expected Output:

Accuracy score based on test data.

Assessment Questions:

What happens if you choose too small or too large a value of k?
Is KNN sensitive to feature scaling?

✅ Lab 18 – Decision Trees

Objective:
Apply Decision Trees for classification.

Theory Overview:

- **Decision Trees**: Tree-like model of decisions.

Tasks/Instructions:

1. Train a Decision Tree.
2. Visualize the tree.

Sample Code:

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Model
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# Plot
plot_tree(tree)
plt.show()

print("Accuracy:", tree.score(X_test, y_test))
```

Expected Output:

Decision tree diagram.
Model accuracy.

Assessment Questions:

What is overfitting in decision trees?
How can you control tree depth?

✅ Lab 19 – Random Forest

Objective:
Learn ensemble learning using Random Forests.

Theory Overview:

- **Random Forest**: Collection of decision trees for better performance.

Tasks/Instructions:

1. Train a Random Forest.
2. Evaluate performance.

Sample Code:

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=100)
forest.fit(X_train, y_train)

print("Accuracy:", forest.score(X_test, y_test))
```

Expected Output:

Higher accuracy due to ensemble effect.

Assessment Questions:

Why does Random Forest reduce overfitting?
What does n_estimators mean?

✅ Lab 20 – Naive Bayes

Objective:
Apply Naive Bayes for classification tasks.

Theory Overview:

- **Naive Bayes**: Based on Bayes' Theorem with independence assumptions.

Tasks/Instructions:

1. Train Gaussian Naive Bayes model.
2. Predict and check accuracy.

Sample Code:

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(X_train, y_train)

print("Accuracy:", model.score(X_test, y_test))
```

Expected Output:

Accuracy result printed.

Assessment Questions:

What assumptions are made by Naive Bayes?
Is Naive Bayes good for small datasets?

✅ Lab 21 – Support Vector Machines (SVM)

Objective:
Classify data using SVM.

Theory Overview:

- **SVM**: Finds the best hyperplane separating classes.

Tasks/Instructions:

1. Train an SVM.
2. Predict and measure performance.

Sample Code:

```
from sklearn.svm import SVC

svm = SVC()
svm.fit(X_train, y_train)

print("Accuracy:", svm.score(X_test, y_test))
```

Expected Output:

Test set accuracy.

Assessment Questions:

What is a kernel in SVM?
When would you prefer SVM over other algorithms?

✅ **Lab 22 – Model Evaluation: Accuracy, Precision, Recall, F1**

Objective:
Learn different metrics for model evaluation.

Theory Overview:

- **Accuracy**: Overall correct predictions.
- **Precision**: Correct positive predictions.
- **Recall**: Coverage of actual positives.
- **F1 Score**: Harmonic mean of precision and recall.

Tasks/Instructions:

1. Predict labels.
2. Calculate evaluation metrics.

Sample Code:

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

Expected Output:

Full classification report with all metrics.

Assessment Questions:

Why is F1-score more important in imbalanced datasets?

✅ Lab 23 – Confusion Matrix

Objective:
Plot and interpret confusion matrices.

Theory Overview:

- Matrix showing true vs predicted labels.

Tasks/Instructions:

1. Predict and create confusion matrix.

Sample Code:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

Expected Output:

Confusion matrix plot.

✅ Lab 24 – Cross-validation

Objective:
Use cross-validation to assess models.

Theory Overview:

- Splitting data into multiple folds to validate models better.

Tasks/Instructions:

1. Perform 5-fold cross-validation.

Sample Code:

```
rom sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean())
```

Expected Output:

Cross-validation accuracies and average.

✅ Lab 25 – Model Saving and Loading with joblib or pickle

Objective:
Save trained models for reuse.

Theory Overview:

- Save using joblib or pickle.

Tasks/Instructions:

1. Save and load a model.

Sample Code:

```
import joblib

# Save model
joblib.dump(model, 'model.pkl')

# Load model
loaded_model = joblib.load('model.pkl')
print("Accuracy after loading:", loaded_model.score(X_test, y_test))
```

Expected Output:

Successfully saved and loaded model with similar accuracy.

🧠 Part IV: Deep Learning Basics with TensorFlow / Keras (Labs 26–30)

✅ Lab 26 – Introduction to Neural Networks

Objective:
Understand the basic concepts of Artificial Neural Networks (ANNs).

Theory Overview:

- **Neural Networks**: Modeled after the human brain; consist of neurons (nodes) arranged in layers.
- **Key Concepts**:
  - **Input Layer**, **Hidden Layers**, **Output Layer**
  - **Weights**, **Biases**, **Activation Functions**

Tasks/Instructions:

1. Understand basic components of a neural network.
2. Sketch a basic ANN structure.

Sample Code (basic Keras installation test):

```
# Install TensorFlow if needed:
# pip install tensorflow

import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

Expected Output:

TensorFlow version printed without errors.

Assessment Questions:

What are the basic building blocks of an ANN?
Why are activation functions used in neural networks?

✅ Lab 27 – Building a Simple ANN using Keras

Objective:

Build and train a basic ANN using Keras API.

Theory Overview:

- **Sequential Model**: Linear stack of layers.
- **Dense Layer**: Fully connected layer.

Tasks/Instructions:

1. Build a simple ANN for binary classification (e.g., Titanic dataset or random data).

Sample Code:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Dummy Data
X = np.random.random((1000, 20))
y = np.random.randint(2, size=(1000, 1))

# Build model
model = Sequential([
   Dense(64, activation='relu', input_dim=20),
   Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train
model.fit(X, y, epochs=10, batch_size=32)
```

Expected Output:

Training loss and accuracy after each epoch.

Assessment Questions:

What is the role of the loss function during training?

What is the difference between relu and sigmoid activations?

✅ Lab 28 – Activation Functions and Optimizers

Objective:

Explore various activation functions and optimizers.

Theory Overview:

- **Activation Functions**: ReLU, Sigmoid, Tanh, Softmax
- **Optimizers**: SGD, Adam, RMSprop

Tasks/Instructions:

1. Experiment with different activation functions.
2. Compare optimizers' effect on training.

Sample Code:

```
# Same as previous, but change activation or optimizer
model = Sequential([
  Dense(64, activation='tanh', input_dim=20),
  Dense(1, activation='sigmoid')
])

model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X, y, epochs=10, batch_size=32)
```

Expected Output:

Training performance using different activations and optimizers.

Assessment Questions:

Which activation function is better for binary classification?

What advantage does Adam optimizer have over SGD?

✅ Lab 29 – Overfitting and Regularization

Objective:
Learn how to detect and reduce overfitting in neural networks.

Theory Overview:

- **Overfitting**: High training accuracy but low testing accuracy.
- Techniques:
  - Dropout
  - L2 Regularization
  - Early Stopping

Tasks/Instructions:

1. Add dropout to a model.
2. Compare model performance with and without regularization.

Sample Code:

```
from tensorflow.keras.layers import Dropout

model = Sequential([
  Dense(64, activation='relu', input_dim=20),
  Dropout(0.5),
  Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X, y, epochs=10, batch_size=32, validation_split=0.2)
```

Expected Output:

Training vs validation loss curve showing less overfitting.

Assessment Questions:

How does dropout help reduce overfitting?
What does L2 regularization penalize?

✅ Lab 30 – Image Classification with CNN (MNIST Dataset)

Objective:
Classify handwritten digits using Convolutional Neural Networks (CNN).

Theory Overview:

- **Convolutional Neural Networks**: Designed for image processing.
- Layers:
  - Convolution Layer
  - MaxPooling Layer
  - Dense Layer

Tasks/Instructions:

1. Load MNIST dataset.
2. Build and train a CNN model.

Sample Code:

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Build model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
```

```
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

# Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
```

Expected Output:

- CNN achieves >97% accuracy on MNIST.

Assessment Questions:

1. What is the role of convolutional layers in CNNs?
2. Why is MNIST dataset popular in deep learning?

🚀 Part V: Deployment & Final Project (Labs 31–33)

✅ Lab 31 – Model Deployment using Flask

🎯 Objective:

Learn how to deploy a trained Machine Learning model using Flask, a lightweight web framework in Python.

📘 Theory:

- **Model Deployment**: Making an ML model accessible as a web service.
- **Flask**: Used to build REST APIs.
- **joblib/pickle**: Used to save/load ML models.

🧪 Tasks/Steps:

1. Train and save a simple model (e.g., Logistic Regression).
2. Create a Flask app to serve predictions.
3. Run and test the app via curl or browser.

🔤 Sample Code:

1. Save a trained model (model.py):

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
import joblib

iris = load_iris()
X, y = iris.data, iris.target
model = LogisticRegression(max_iter=200)
model.fit(X, y)

joblib.dump(model, 'iris_model.pkl')
```

2. Create Flask app (app.py):

```
from flask import Flask, request, jsonify
import joblib
import numpy as np
```

```
app = Flask(__name__)
model = joblib.load('iris_model.pkl')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    prediction = model.predict([np.array(data['features'])])
    return jsonify({'prediction': int(prediction[0])})

if __name__ == '__main__':
    app.run(debug=True)
```

✅ **Expected Output:**

A local server that returns predictions when you send feature data as JSON.

🧠 **Assessment Questions:**

What is the role of joblib in model deployment?
Why is Flask suitable for ML deployment?

✅ **Lab 32 – Final Mini Project**

🎯 **Objective:**
Apply end-to-end Python and ML skills in a mini-project of your choice.

💡 **Suggested Projects:**

- **Student Performance Predictor**
- **Spam Email Classifier**
- **COVID-19 Dataset Analysis**
- **Sales Forecasting**
- **Image Classifier (Fashion MNIST or CIFAR-10)**

📘 **Project Components:**

1. **Problem Definition**
2. **Data Collection**
3. **Exploratory Data Analysis (EDA)**
4. **Preprocessing**
5. **Model Building and Evaluation**
6. **Optional: Deployment via Flask**

📂 Example Template:

Project Title: Student Performance Predictor

Problem Statement:
Predict final exam scores based on input features.

Dataset Used:
student-mat.csv

Technologies:
Pandas, sklearn, matplotlib, Flask (for deployment)

Steps Performed:
- Data loading and cleaning
- EDA and visualization
- Model training (Linear Regression)

- Model evaluation
- Web app interface using Flask

Outcome:
Model accuracy: 82%, deployed on local server.

✅ **Expected Output**:

A short report + working code notebook or Flask app.

🧠 **Assessment Questions:**

What metric did you use to evaluate your model and why?
How could this project be improved with more data?

🤖 Lab 33 – Building an AI Agent

🎯 Objective:

Understand and implement the concept of a simple AI agent that perceives an environment and acts intelligently.

📘 Theory Overview:

- **AI Agent**: An entity that perceives its environment through sensors and acts upon it using actuators to achieve goals.
- **Types of AI Agents**:
  - Simple reflex agents
  - Model-based agents
  - Goal-based agents
  - Learning agents

In this lab, we'll simulate a **goal-based agent** using Python in a simple grid environment.

🧪 Task:

Build a basic AI agent that navigates a 5x5 grid to find a goal using intelligent decision-making (e.g., depth-first search or random movement with memory).

🧠 Step-by-Step Instructions:

1. Create a 5x5 grid environment.
2. Place the agent at a random start position.
3. Place a goal at another random position.
4. Let the agent explore the grid and find the shortest path to the goal using a simple search algorithm.

💻 Sample Code (AI Grid Agent with DFS):

```
import random

# Define grid size
GRID_SIZE = 5
goal = (random.randint(0, 4), random.randint(0, 4))

# Directions (Up, Down, Left, Right)
moves = [(-1,0), (1,0), (0,-1), (0,1)]
```

```
# DFS search
def is_valid(x, y, visited):
    return 0 <= x < GRID_SIZE and 0 <= y < GRID_SIZE and (x, y) not in visited

def dfs(position, path, visited):
    if position == goal:
        return path
    x, y = position
    visited.add(position)

    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if is_valid(nx, ny, visited):
            result = dfs((nx, ny), path + [(nx, ny)], visited)
            if result:
                return result
    return None

# Initialize agent at random position
start = (random.randint(0, 4), random.randint(0, 4))
while start == goal:
    start = (random.randint(0, 4), random.randint(0, 4))

# Run the agent
path_to_goal = dfs(start, [start], set())

# Output
print("Grid Size: 5x5")
print("Start Position:", start)
print("Goal Position:", goal)
print("Path Found by Agent:", path_to_goal)
```

✅ Expected Output:

Start and goal coordinates

A valid path (sequence of coordinates) showing how the agent reached the goal

🧠 **Assessment Questions:**

What type of agent is used in this lab? Why?
Can this agent work if obstacles are introduced into the grid? What would you change?
How would you convert this into a learning agent?

📌 **Optional Extensions:**

- Add obstacles (represented as blocked cells)
- Switch DFS to A* search for optimality
- Visualize the grid and path using matplotlib