Artificial Intelligence

Course Code CSC 363

Laboratory Manual

By

Amara Khan
Fa-21/BSCS/359

Department of Computer Science

Lahore Garrison University

Main Campus, Sector-C Phase-VI, DHA Lahore

Guidelines for Laboratory Procedure

The laboratory manual is the record of all work on your experiments. A complete, neat, and organized data record is as important as the experiment itself. Please follow these guidelines for efficient performance in the laboratory:

1. Attend the lab orientation to familiarize yourself with the lab setup.

2. Follow the designated lab schedule and complete assignments on time.

3. Write clear and well-documented code, avoiding plagiarism and unauthorized collaboration.

4. Seek help from lab instructors or peers if you encounter difficulties with programming concepts.

5. Regularly back up your code and project files to prevent data loss.

6. Use lab resources responsibly, including computers and software licenses.

7. If collaboration is allowed, work effectively with peers, ensuring each member contributes meaningfully.

8. Maintain a clean and organized workspace for better focus and efficiency.

9. Thoroughly test your code to identify and fix errors before submission.

10. Engage in lab discussions, share insights, and actively participate to enhance the learning experience.

Safety Precautions

1. Handle equipment carefully to prevent damage and avoid placing liquids near electronic devices.
2. Maintain an ergonomic workspace for comfortable and strain-free programming.
3. Save work frequently and use surge protectors to prevent data loss due to power issues.
4. Keep software and antivirus programs up to date and avoid downloading from untrusted sources.
5. Regularly back up code and important files to prevent data loss.
6. Establish clear communication and collaboration guidelines when working with others.
7. Be aware of emergency exits, fire extinguisher locations, and evacuation procedures.

Safety Undertaking

I have read all the above, and I agree to conform to its contents.

Name: Amara Khan

Registration No.: Fa-21/BSCS/359

Student Signature: amara

Date: 2nd July ,2024

Lab Instructor: Mr. Usama Asif

Grading Policy

Lab Performance	15%
Lab Report	15%
Lab Project +Viva	25%+5%
Final Lab Exam	40%

Rubrics

Lab Performance (Continuous Assessment) / Performance Test

Sr.No.	**Performance Indicator**	**Exemplary (4-5)**	**Satisfactory (2-3)**	**Unsatisfactory (0-1)**
1	**Ability to Conduct Practical**	Fully understand the **software including its features, working** and quite able to conduct the entire practical with negligible help from lab instructor	Has very good understanding of the **software including its features, working** and able to conduct practical with some help from lab instructor	Has poor understanding of the **software including its features, working** and unable to conduct practical on his own;
2	**Data Analysis & Interpretation**	Always analyzes and interprets data correctly and always display correct output; always **compares theory against practical results** and resolve related error.	Analyzes and interprets data correctly most of the time; most of the output are correct; **compares theory against practical results** and resolve related error most of the time.	Analyzes and interprets data incorrectly most of the time; many output are incorrect; most of the time never attempts to **compare theory against practical results**.

Lab Reports

Sr.	**Performance Indicator**	**Exemplary (4-5)**	**Satisfactory (2-3)**	**Unsatisfactory (0-1)**
1	**Structure**	All the code is very accurate and precise. Completely logical and systematic compilation.	Most of the code is very accurate and precise. Quite logical and systematic compilation.	Some of or complete code is inaccurate. Somewhat or no logical and systematic compilation.
2	**Efficiency**	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.

Viva Voce

Sr	Performance Indicator	Exemplary (4-5)	Satisfactory (2-3)	Unsatisfactory (0-1)
1	**Responsiveness to Questions/ Accuracy**	Responds well, quick and very accurate all the time.	Generally Responsive and accurate most of the times.	Non-responsive.
2	**Level of understanding of the learned skill**	Demonstration of full knowledge of the subject with explanations and elaboration.	At ease with content and able to elaborate and explain to some degree.	No grasp of information. Clearly no knowledge of subject matter. No questions are answered. No interpretation made.

Lab Project

Sr.No.	Performance	Exemplary (4-5)	Satisfactory (2-3)	Unsatisfactory (0-1)
	Project Design			
1	**Implementation and Completion**	Project is completed without any external assistance and is working properly.	Project is completed with quite less technical assistance from the instructor or others in order to complete the project and is working properly.	Project is completed but not working properly. Or Project is not completed.
2	**Appearance and Problem Analysis**	Circuit wiring and components are perfectly organized and proper prototyping is done.	Circuit wiring and components are organized and some prototyping is done.	Circuit wiring and components are disorganized but some prototyping is done.
	Project Report			
1	**Structure and Literature Review**	Information is presented in a logical, interesting way, which is easy to follow. All **sections** are in a correct order and submitted on a time. Collected a great deal of information--all relates to the topic.	Information is presented in a less logical way, which is little difficult to follow. All **sections** are in a little incorrect order or submitted little late time. Collected a fine information—all may relates to the topic.	Information is not presented in a logical, interesting way, which is so difficult to follow. All **sections** are incorrect order or not submitted on a time. Collected a poor information--all doesn't relates to the topic.
2	**Result and Presentation**	Clearly discusses what results mean and what conclusions may be drawn from them. Cites published standards or other related reports.	Generally clear discussion of results and conclusions, but may miss some points. Some use of references and published standards.	Limited discussion of results and conclusions. Little or no reference to published standards or other reports.
	Project Viva			
1	**Responsiveness to Questions/ Accuracy**	Responds well, quick and very accurate all the time.	Generally Responsive and accurate most of the times.	Non-responsive.
2	**Level of understanding of the learned skill**	Demonstration of full knowledge of the subject with explanations and elaboration.	At ease with content and able to elaborate and explain to some degree.	No grasp of information. Clearly no knowledge of subject matter. No questions are answered. No interpretation made.
	Project Presentation			

1	**Organization**	Presentation is clear and logical. Listener can easily follow line of reasoning	Presentation is generally clear. A few minor points may be confusing	Listener can follow presentation with effort. Organization not well thought out.
2	**Confident**	Is very confident and explains the details properly. Proper eye contact is maintained all the time with proper presentational gestures.	Is confident to some extent with quite less eye contact and presentational gestures.	Has low confidence to explain and deliver topic properly. Less eye contact and presentational gestures used.
3	**Responsiveness to Audience**	Responds well to questions. Restates and summarizes when needed.	Generally responsive to questions.	Reluctantly interacts with audience. Responds poorly to questions.
	Team Work			
1	**Share Information**	Relays a great deal of information--all relates to the topic.	Relays some basic information--most relates to the topic.	Relays very little information--some relates to the topic
2	**Fulfill Team duties**	Performs all duties of assigned team role.	Performs nearly all duties.	Performs very little duties.

Lab's Course Learning Outcomes

Course Title: Artificial Intelligence

Course Code: CSC-363

Instructor: Mr. Usama Asif

Designation: Assistant Lecturer

E-mail: usamaasif @lgu.edu.pk

Students will be able to:

- Engage in a classroom and laboratory environment designed to develop proficiency in Python programming for Artificial Intelligence applications.
- Gain foundational knowledge in software engineering and program development, with no prior background in computer programming assumed.
- Focus primarily on the specifics of Python, including writing and understanding the basic structure of Python programs.
- Implement data structures, control flow, and various AI algorithms using Python.
- Perform programming experiments, data preprocessing, and exploratory data analysis.
- Develop machine learning models for classification, regression, and clustering, and apply AI techniques to solve real-world problems.

CLO1: Demonstrate proficiency in developing and implementing AI applications using software.

CLO2: Apply AI techniques to solve well-defined, practical problems in various domains.

CLO3: Demonstrate proficiency in developing and applying models of machine learning..

Mapping of Course Learning Outcomes (CLO) to Program Learning Outcomes (PLO) / Graduate Attributes

Course Code	CLOs/ PLOs	PLO 1	PLO 2	PLO 3	PLO 4	PLO 5	PLO 6	PLO 7	PLO 8	PLO 9	PLO 10
CSC-363	CLO 1					X					
	CLO2		X								
	CLO3				X						

PLO1: Academic Education
PLO2: Knowledge for Solving Computing Problems
PLO3: Problem Analysis
PLO4: Design/ Development of Solution
PLO5: Modern Tool Usage
PLO6: Individual and Teamwork
PLO7: Communication
PLO8: Computing Professionalism and Society
PLO9: Ethics

PLO10: Lifelong Learning

Sr.No	List of Practical	CLO’s
1	Setting Up the AI Development Environment	1
2	Basic Python Programming for AI	1
3	Implementing Data Structures in Python	1
4	Conditional Statements in Python	1
5	Loops in Python	1
6	Implementation of BFS and DFS Algorithm for Pathfinding	2
7	Implementation of Uniform Cost Search and Greedy Best First Search Algorithm for Pathfinding	2
8	Implementation of A* Algorithm for Pathfinding	2
9	Dataset Acquisition from Public Repositories	3
10	Preprocessing Datasets for Machine Learning	3
11	Performing Exploratory Data Analysis with Python	3
12	Building Classification Models in Machine Learning	3
13	Creating Regression Models for Predictive Analysis	3
14	Implementing Clustering Algorithms for Data Segmentation	3

PRACTICAL NO.01
Setting Up the AI Development Environment

PLO	CLO
5	1

Objective:

To set up the necessary development environment for artificial intelligence projects, including installing Anaconda 3 and essential libraries.

Learning Outcome:

Students will be able to set up a Python development environment using Anaconda 3.

Materials Required:

- A computer with internet access
- Anaconda 3 distribution
- Jupyter Notebook

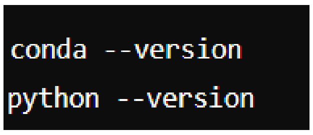
Activity:

1. **Download and Install Anaconda 3:**

   - Download the Anaconda 3 installer for your operating system from Anaconda's official website.
   - Follow the installation instructions for your operating system (Windows/Mac/Linux).

2. **Verify Anaconda Installation:**

   - Open Anaconda Prompt (Windows) or a terminal (Mac/Linux) and type the following commands to verify the installation:



```
conda --version
python --version
```

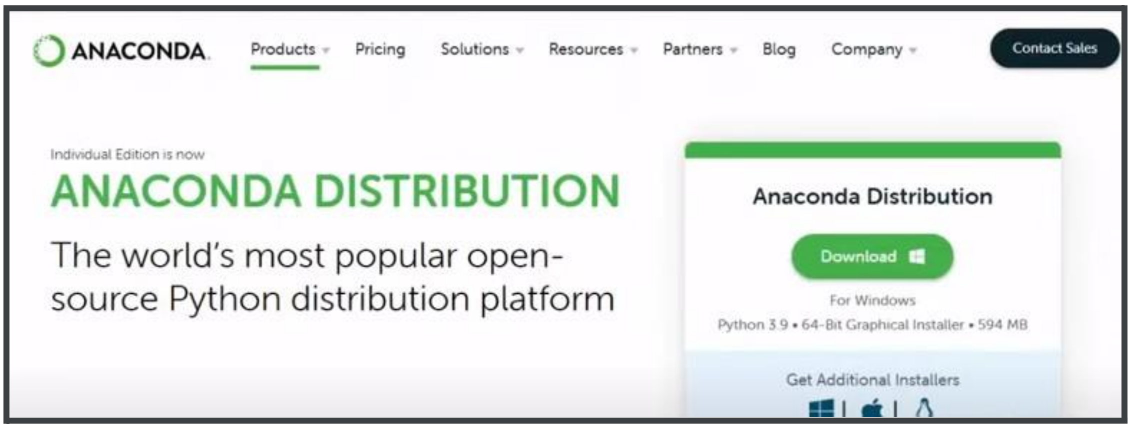
Expected Outcome:

By the end of this experiment, students should have a fully functional Python development environment set up with Anaconda 3. They should be able to start Jupyter Notebook.

SOLUTION:

Step 1: Download Anaconda

1. Go to the Anaconda Distribution (https://www.anaconda.com/products/distribution) page.
2. Click on the "Download" button for the Windows version.
3. Choose the installer that matches your Windows system (64-bit or 32-bit).



Individual Edition is now

ANACONDA DISTRIBUTION

The world's most popular open-source Python distribution platform

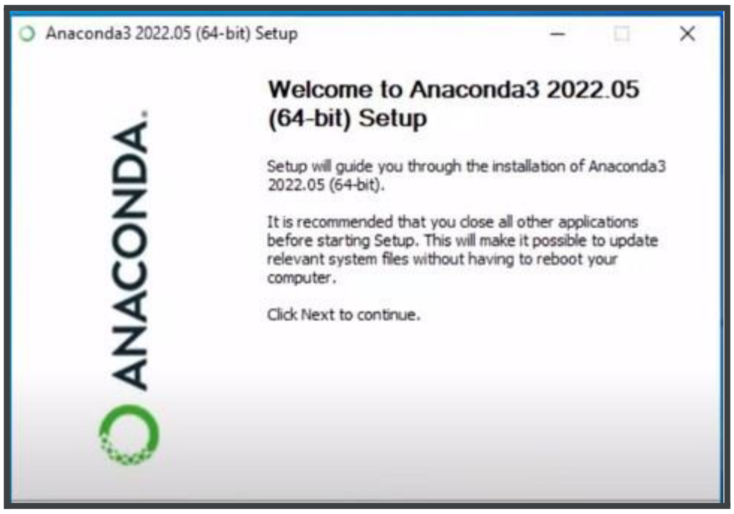
Anaconda Distribution

For Windows

Python 3.9 • 64-Bit Graphical Installer • 594 MB

Step 2: Install Anaconda

1. Locate the downloaded installer and double-click it.



Anaconda3 2022.05 (64-bit) Setup

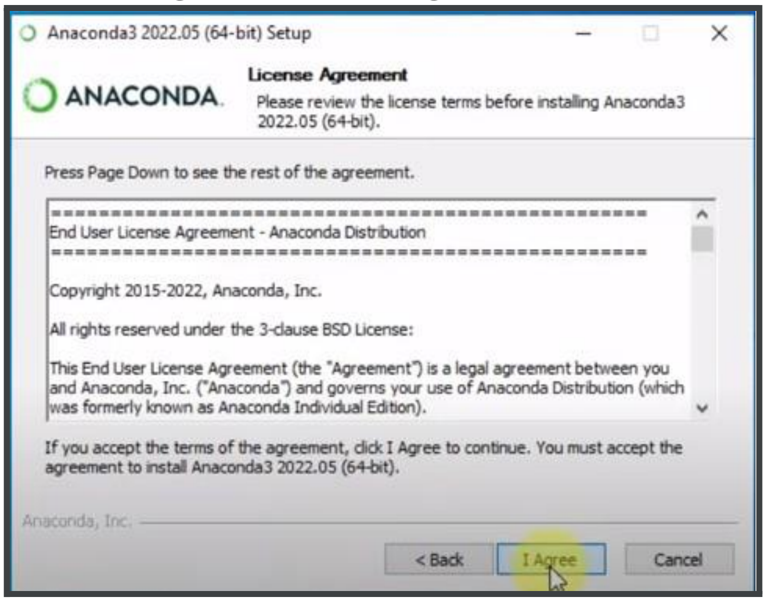
Welcome to Anaconda3 2022.05 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 2022.05 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

2. Click "Next" on the welcome screen.
3. Read and agree to the license agreement, then click "Next."



Anaconda3 2022.05 (64-bit) Setup

License Agreement

Please review the license terms before installing Anaconda3 2022.05 (64-bit).

Press Page Down to see the rest of the agreement.

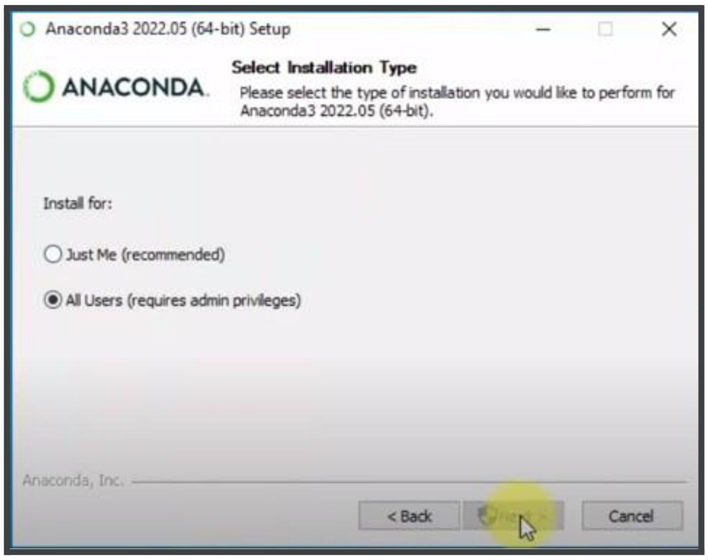
```
===================================================
End User License Agreement - Anaconda Distribution
===================================================

Copyright 2015-2022, Anaconda, Inc.

All rights reserved under the 3-clause BSD License:

This End User License Agreement (the "Agreement") is a legal agreement between you
and Anaconda, Inc. ("Anaconda") and governs your use of Anaconda Distribution (which
was formerly known as Anaconda Individual Edition).
```

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2022.05 (64-bit).

Anaconda, Inc.

4. Select "Just Me" if you're installing for yourself, or "All Users" if installing for everyone on the computer, then click "Next."

5. Choose the installation location (the default is usually fine) and click "Next."


Anaconda3 2022.05 (64-bit) Setup

ANACONDA.

Select Installation Type

Please select the type of installation you would like to perform for Anaconda3 2022.05 (64-bit).

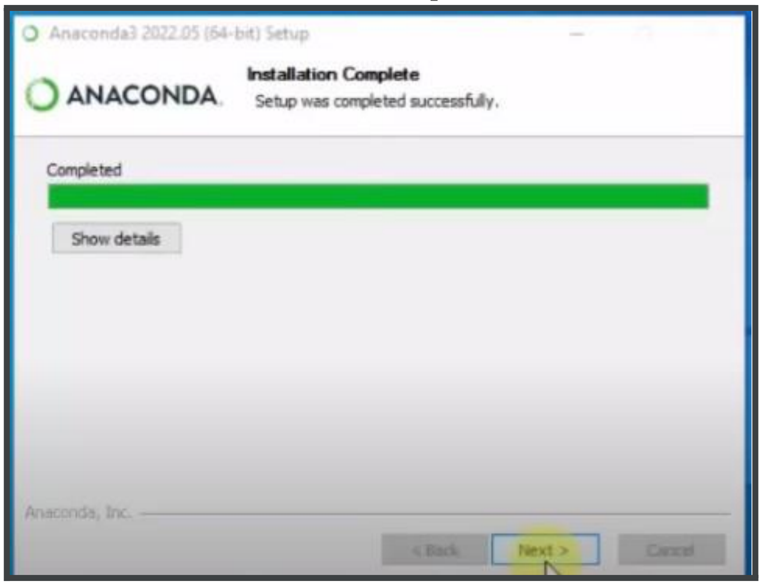
Install for:

○ Just Me (recommended)

◉ All Users (requires admin privileges)

Anaconda, Inc.

6. Ensure that "Add Anaconda to my PATH environment variable" is checked (recommended), then click "Install."
7. Wait for the installation to complete, then click "Next" and "Finish."


Anaconda3 2022.05 (64-bit) Setup

ANACONDA.

Installation Complete

Setup was completed successfully.

Completed

Anaconda, Inc.

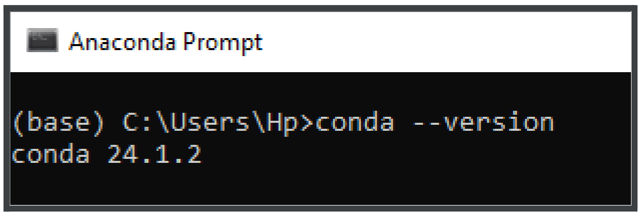
Step 3: Verify the Installation

1. Open the Anaconda Prompt from the Start Menu (search for "Anaconda Prompt").

Step 4: Check Conda Version

In the Anaconda Prompt, type the following command and press Enter:

conda --version



Anaconda Prompt

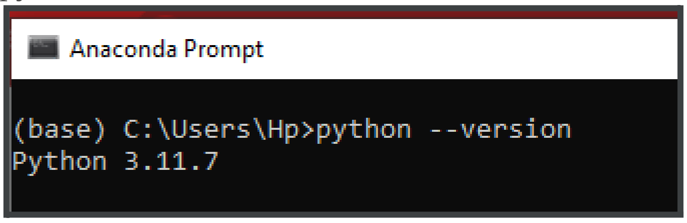
```
(base) C:\Users\Hp>conda --version
conda 24.1.2
```

This will display the version of `conda` installed.

Step 5: Check Python Version

In the Anaconda Prompt, type the following command and press Enter:

python --version



Anaconda Prompt

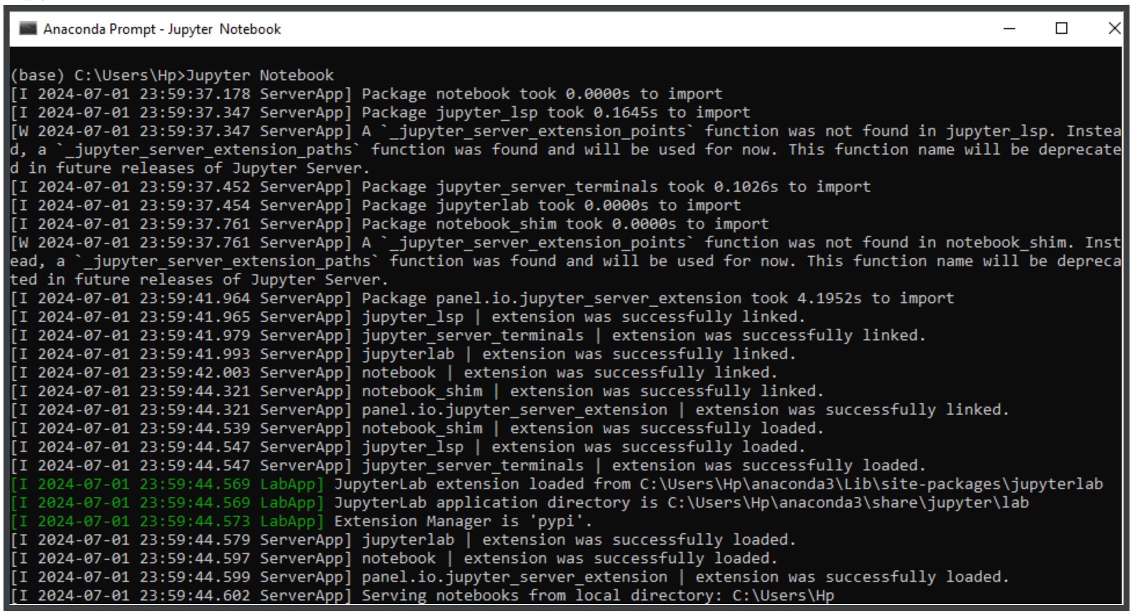
```
(base) C:\Users\Hp>python --version
Python 3.11.7
```

This will display the version of Python installed.

Step 6: Start Jupyter Notebook

In the Anaconda Prompt, type the following command and press Enter:

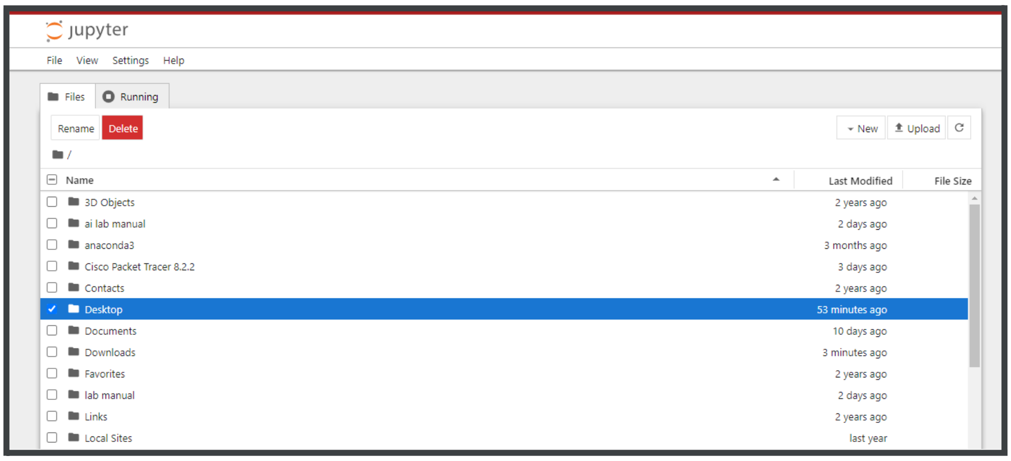
jupyter notebook



Anaconda Prompt - Jupyter Notebook

```
(base) C:\Users\Hp>Jupyter Notebook
[I 2024-07-01 23:59:37.178 ServerApp] Package notebook took 0.0000s to import
[I 2024-07-01 23:59:37.347 ServerApp] Package jupyter_lsp took 0.1645s to import
[W 2024-07-01 23:59:37.347 ServerApp] A `_jupyter_server_extension_points` function was not found in jupyter_lsp. Instea
d, a `_jupyter_server_extension_paths` function was found and will be used for now. This function name will be deprecate
d in future releases of Jupyter Server.
[I 2024-07-01 23:59:37.452 ServerApp] Package jupyter_server_terminals took 0.1026s to import
[I 2024-07-01 23:59:37.454 ServerApp] Package jupyterlab took 0.0000s to import
[I 2024-07-01 23:59:37.761 ServerApp] Package notebook_shim took 0.0000s to import
[W 2024-07-01 23:59:37.761 ServerApp] A `_jupyter_server_extension_points` function was not found in notebook_shim. Inst
ead, a `_jupyter_server_extension_paths` function was found and will be used for now. This function name will be depreca
ted in future releases of Jupyter Server.
[I 2024-07-01 23:59:41.964 ServerApp] Package panel.io.jupyter_server_extension took 4.1952s to import
[I 2024-07-01 23:59:41.965 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-07-01 23:59:41.979 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-07-01 23:59:41.993 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-07-01 23:59:42.003 ServerApp] notebook | extension was successfully linked.
[I 2024-07-01 23:59:44.321 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-07-01 23:59:44.321 ServerApp] panel.io.jupyter_server_extension | extension was successfully linked.
[I 2024-07-01 23:59:44.539 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-07-01 23:59:44.547 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-07-01 23:59:44.547 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-07-01 23:59:44.569 LabApp] JupyterLab extension loaded from C:\Users\Hp\anaconda3\Lib\site-packages\jupyterlab
[I 2024-07-01 23:59:44.569 LabApp] JupyterLab application directory is C:\Users\Hp\anaconda3\share\jupyter\lab
[I 2024-07-01 23:59:44.573 LabApp] Extension Manager is 'pypi'.
[I 2024-07-01 23:59:44.579 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-07-01 23:59:44.597 ServerApp] notebook | extension was successfully loaded.
[I 2024-07-01 23:59:44.599 ServerApp] panel.io.jupyter_server_extension | extension was successfully loaded.
[I 2024-07-01 23:59:44.602 ServerApp] Serving notebooks from local directory: C:\Users\Hp
```

This will open the Jupyter Notebook interface in our default web browser.



Name	Last Modified	File Size
3D Objects	2 years ago	
ai lab manual	2 days ago	
anaconda3	3 months ago	
Cisco Packet Tracer 8.2.2	3 days ago	
Contacts	2 years ago	
✓ Desktop	53 minutes ago	
Documents	10 days ago	
Downloads	3 minutes ago	
Favorites	2 years ago	
lab manual	2 days ago	
Links	2 years ago	
Local Sites	last year	

RUBRICS:

Performance			**Lab Report**		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ——————————————

PRACTICAL NO.02
Basic Python Programming for AI

PLO	CLO
5	1

Objectives:

To practice basic Python programming concepts, including variable declaration, data types, and operators.

Learning Outcome:

Students will be able to declare variables, assign values, and use various Python operators.

Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

   ```
   jupyter notebook
   ```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Variable Declaration and Assignment:**

   Write a Python code that does the following:

   - Declare variables for your name, roll number, and CGPA.
   - Assign appropriate values to each variable (name as a string, roll number as an integer, and CGPA as a float).
   - Print your name, roll number, and CGPA.
   - Check and print the data type of each variable.

4. **Operators in Python:**

Write a Python code snippet that defines two variables a and b with any numeric values of your choice and perform the following operations:

- **Arithmetic Operators:**
  Perform basic arithmetic operations like addition, subtraction, multiplication, division, etc., on given numeric values.

- **Comparison Operators:**
  Compare two numeric values using operators such as equal to, not equal to, greater than, less than, etc.

- **Logical Operators:**
  Apply logical operators like AND, OR, and NOT on Boolean values.

- **Membership Operators:**
  Check if a given element belongs to a sequence using membership operators (in and not in).

- **Identity Operators:**
  Determine whether two variables refer to the same object in memory using identity operators (is and is not).

Expected Outcome:

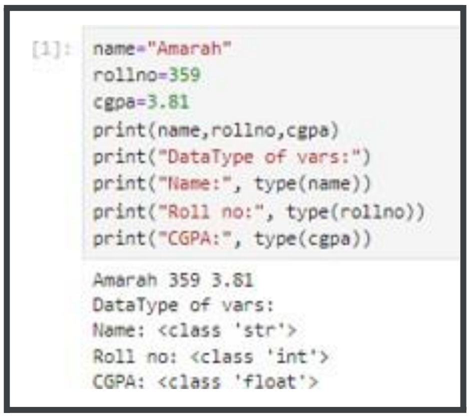
By the end of this experiment, students should be able to declare and assign values to variables, print their values and data types, and understand the usage of various Python operators.

SOLUTION:

Variable Declaration and Assignment

```
name="Amarah"
rollno=359
cgpa=3.81
print(name,rollno,cgpa)
print("DataType of vars:")
print("Name:", type(name))
print("Roll no:", type(rollno))
print("CGPA:", type(cgpa))
```

Output:

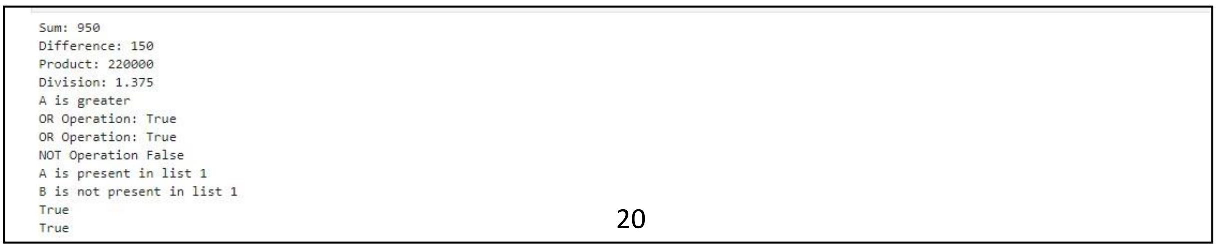


```
[1]: name="Amarah"
     rollno=359
     cgpa=3.81
     print(name,rollno,cgpa)
     print("DataType of vars:")
     print("Name:", type(name))
     print("Roll no:", type(rollno))
     print("CGPA:", type(cgpa))

     Amarah 359 3.81
     DataType of vars:
     Name: <class 'str'>
     Roll no: <class 'int'>
     CGPA: <class 'float'>
```

Operators in Python

```
a=550
b=400
#Arithmetic Operators
print("Sum:", a+b)
print("Difference:", a-b)
print("Product:", a*b)
print("Division:", a/b)
#Comparision Operators
if a>b:
    print("A is greater")
elif b>a:
    print("B is greater")
else:
    print("A is equal to B")
#Logicial Operators
print("OR Operation:", a>b and b<1000)
print("OR Operation:", a>b or b<100)
print("NOT Operation", not(a>b))
#Membership Operators
list1=[100,550,480]
if a in list1:
    print("A is present in list 1")
if b not in list1:
    print("B is not present in list 1")
#Identity Operator
c=a
print(a is not b)
print(a is c)
```

Output:



```
Sum: 950
Difference: 150
Product: 220000
Division: 1.375
A is greater
OR Operation: True
OR Operation: True
NOT Operation False
A is present in list 1
B is not present in list 1
True
True
```

RUBRICS:

Performance			Lab Report		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ————————————

PRACTICAL NO.03

Implementing Data Structures in Python

PLO	CLO
5	1

Objectives:

To understand and practice basic data structure operations in Python, including list creation, insertion, deletion, and slicing.

Learning Outcome:

Students will be able to create and manipulate lists in Python.

Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

   ```
   jupyter notebook
   ```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Create and Manipulate Lists:**

   Write a Python code that does the following:

   - Create a list called months that includes ‘January’, ‘March’, and ‘April’.
   - Create another list named remaining_months and add the rest of the months of the year to it.
   - Insert ‘February’ into the months list in the correct order.
   - Remove ‘February’ from the months list.
   - Print the elements of the list that represent the time from ‘March’ to ‘September’.

Expected Outcome:

By the end of this experiment, students should be able to create and manipulate lists in Python, including inserting, removing, and slicing elements from the list.

SOLUTION:

Create And Manipulate Lists

months=["January","March","April"]

remaining_months=["February","May","June","July","August","September","October","November","December"]

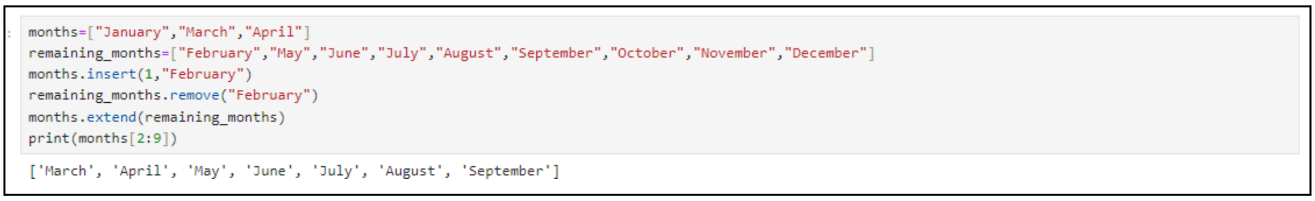
months.insert(1,"February")

remaining_months.remove("February")

months.extend(remaining_months)

print(months[2:9])

Output:



```
months=["January","March","April"]
remaining_months=["February","May","June","July","August","September","October","November","December"]
months.insert(1,"February")
remaining_months.remove("February")
months.extend(remaining_months)
print(months[2:9])
['March', 'April', 'May', 'June', 'July', 'August', 'September']
```

RUBRICS:

Performance			Lab Report		
Description	Total Marks	Marks Obtained	Description	Total Marks	Marks Obtained
Ability to Conduct practical	5		Structure	5	
Data Analysis & Interpretation	5		Efficiency	5	
Total Marks obtained			Total Marks Obtained		

Instructor Signature ——————————

PRACTICAL NO.04
Conditional Statements in Python

PLO	CLO
5	1

Objectives:

To practice using Python tuples, dictionaries, sets, and conditional statements.

Learning Outcome:

Students will be able to create and manipulate tuples, dictionaries, and sets, as well as use conditional statements to classify data and make decisions.

Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

```
jupyter notebook
```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Part 1: Tuples**

   Write a Python code that does the following:

   - Create a tuple named ai_concepts with elements ‘Machine Learning’, ‘Neural Networks’, ‘Genetic Algorithms’, ‘Natural Language Processing’.
   - Find the index of ‘Genetic Algorithms’ in the ai_concepts tuple.

4. **Part 2: Dictionaries**

   - Create a dictionary named ai_tools with keys as the AI concepts from the ai_concepts tuple and values as examples of tools or libraries used in each concept (e.g., ‘Machine Learning’: ‘scikit-learn’).

- Change the tool associated with ‘Neural Networks’ in ai_tools to a different one, say ‘TensorFlow’.:
  Perform basic arithmetic operations like addition, subtraction, multiplication, division, etc., on given numeric values.

5. **Part 3: Sets**

- Create a set named completed_courses with some AI-related course titles.
- Create another set named upcoming_courses with different AI-related course titles.
- Find the courses that are unique to completed_courses and not in upcoming_courses.

6. **Part 3: Conditional Statements**

- Write a series of if statements that classify a student’s grade. If the grade is above 90, print ‘Excellent’; between 80 and 90, print ‘Good’; between 70 and 80, print ‘Average’; below 70, print ‘Needs Improvement’.
- Use a series of if statements to suggest an AI concept to study based on a given interest. For example, if the interest is ‘data’, print ‘Study Machine Learning’.

Expected Outcome:

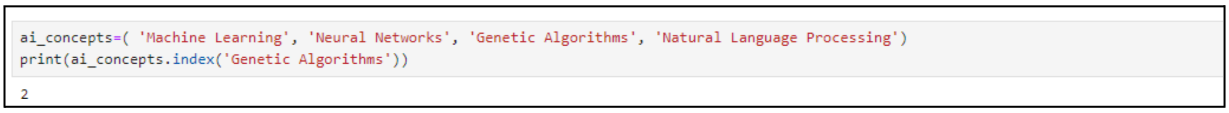
By the end of this experiment, students should be able to effectively use tuples, dictionaries, and sets, and apply conditional statements to classify data and make decisions based on given conditions.

SOLUTION:

1.Tuple

ai_concepts=('Machine Learning', 'Neural Networks', 'Genetic Algorithms', 'Natural Language Processing')
print(ai_concepts.index('Genetic Algorithms'))

Output:

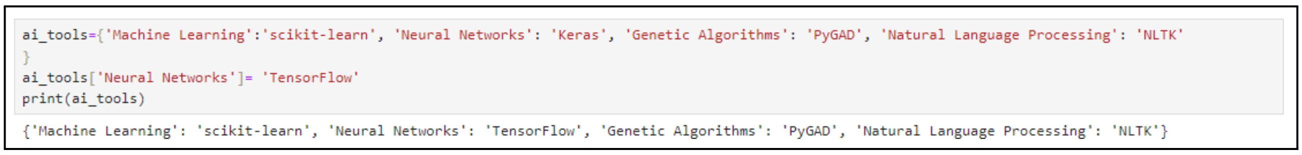


```
ai_concepts=( 'Machine Learning', 'Neural Networks', 'Genetic Algorithms', 'Natural Language Processing')
print(ai_concepts.index('Genetic Algorithms'))

2
```

2. Dictionary

ai_tools={'Machine Learning':'scikit-learn', 'Neural Networks': 'Keras', 'Genetic Algorithms': 'PyGAD', 'Natural Language Processing': 'NLTK'
}
ai_tools['Neural Networks']= 'TensorFlow'
print(ai_tools)

Output:

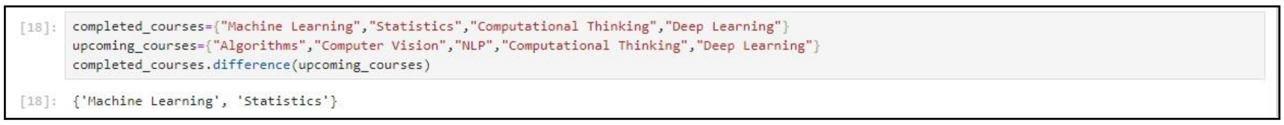


```
ai_tools={'Machine Learning':'scikit-learn', 'Neural Networks': 'Keras', 'Genetic Algorithms': 'PyGAD', 'Natural Language Processing': 'NLTK'
}
ai_tools['Neural Networks']= 'TensorFlow'
print(ai_tools)
{'Machine Learning': 'scikit-learn', 'Neural Networks': 'TensorFlow', 'Genetic Algorithms': 'PyGAD', 'Natural Language Processing': 'NLTK'}
```

<u>3.Sets</u>

completed_courses={"Machine Learning","Statistics","Computational Thinking","Deep Learning"}
upcoming_courses={"Algorithms","Computer Vision","NLP","Computational Thinking","Deep Learning"}
completed_courses.difference(upcoming_courses)

Output:



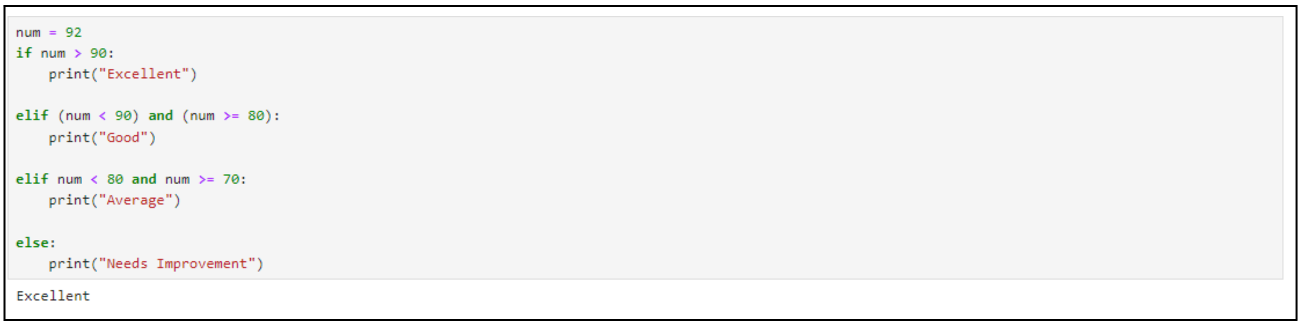
```
[18]:  completed_courses={"Machine Learning","Statistics","Computational Thinking","Deep Learning"}
       upcoming_courses={"Algorithms","Computer Vision","NLP","Computational Thinking","Deep Learning"}
       completed_courses.difference(upcoming_courses)

[18]:  {'Machine Learning', 'Statistics'}
```

<u>4.Conditional Statement</u>

1. Student Grades

```
num = 92
if num > 90:
    print("Excellent")

elif (num < 90) and (num >= 80):
    print("Good")

elif num < 80 and num >= 70:
    print("Average")

else:
    print("Needs Improvement")
```

Output:

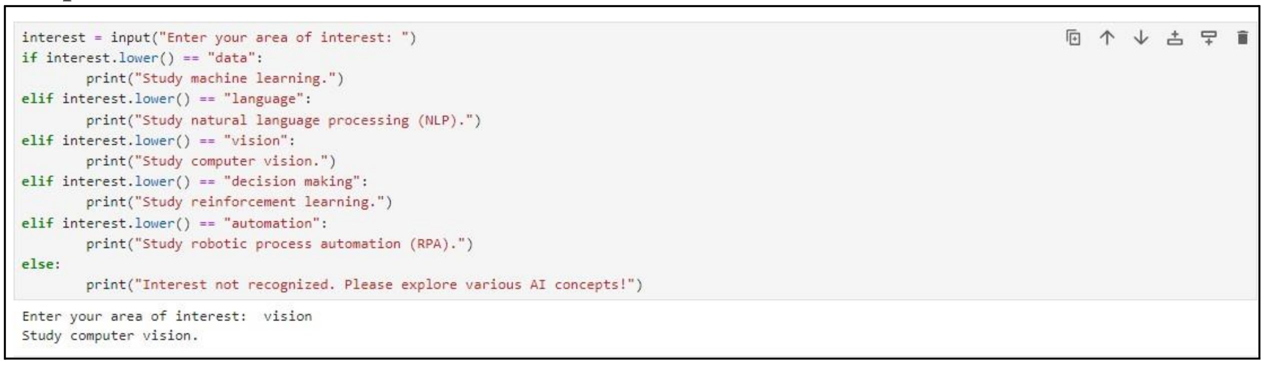


```
num = 92
if num > 90:
    print("Excellent")

elif (num < 90) and (num >= 80):
    print("Good")

elif num < 80 and num >= 70:
    print("Average")

else:
    print("Needs Improvement")
Excellent
```

2. AI Concept to study

```
interest = input("Enter your area of interest: ")
if interest.lower() == "data":
        print("Study machine learning.")
elif interest.lower() == "language":
        print("Study natural language processing (NLP).")
elif interest.lower() == "vision":
        print("Study computer vision.")
elif interest.lower() == "decision making":
        print("Study reinforcement learning.")
elif interest.lower() == "automation":
        print("Study robotic process automation (RPA).")
else:
        print("Interest not recognized. Please explore various AI concepts!")
```

Output:



```
interest = input("Enter your area of interest: ")
if interest.lower() == "data":
        print("Study machine learning.")
elif interest.lower() == "language":
        print("Study natural language processing (NLP).")
elif interest.lower() == "vision":
        print("Study computer vision.")
elif interest.lower() == "decision making":
        print("Study reinforcement learning.")
elif interest.lower() == "automation":
        print("Study robotic process automation (RPA).")
else:
        print("Interest not recognized. Please explore various AI concepts!")
```

```
Enter your area of interest:  vision
Study computer vision.
```

RUBRICS:

Performance			Lab Report		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ———————————

PRACTICAL NO.05
Loops in Python

PLO	CLO
5	1

Objectives:

To practice using loops in Python to recognize number patterns, analyze text, and reverse word order in strings.

Learning Outcome:

Students will be able to implement for loops and functions to manipulate and analyze strings and numbers in Python.

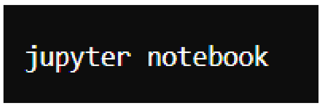
Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:



```
jupyter notebook
```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. Part 1: Number Pattern Recognition with Loops

Implement a Python program using a for loop to:

- Print numbers from 1 to 100.
- Replace multiples of 3 with ‘AI’, multiples of 5 with ‘Lab’, and multiples of both with ‘AILab’.

4. Part 2: Text Analysis Function

Develop a Python function named analyze_text to:

- Count uppercase letters, lowercase letters, and digits in a given string.

- Return the counts in a dictionary format.

5. **Part 3: Word Reversal in Strings**

- Write a Python program to reverse the word order in a sentence.
- For the input ‘Artificial Intelligence Lab’, the output should be ‘Lab Intelligence Artificial’.

Expected Outcome:

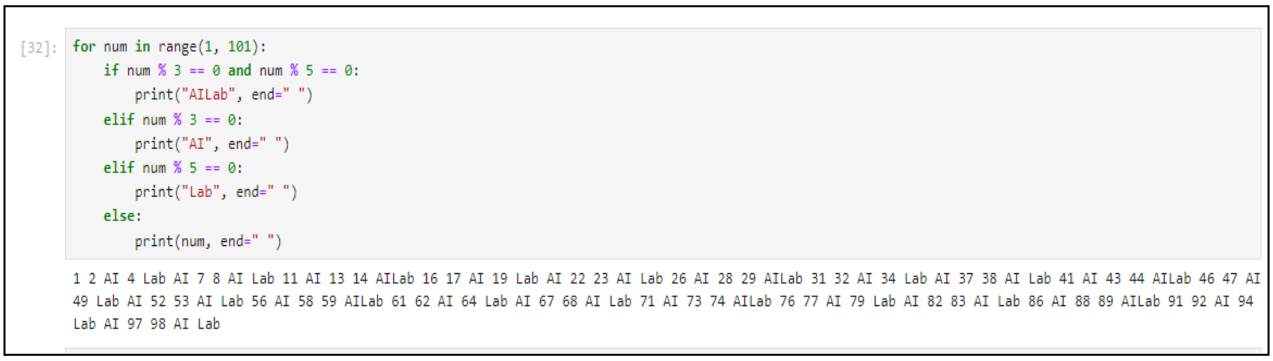
By the end of this experiment, students should be able to use loops to identify patterns in numbers, write functions to analyze text, and manipulate strings to reverse word order.

SOLUTION:

Part 1: Number Pattern Recognition

```
for num in range(1, 101):
    if num % 3 == 0 and num % 5 == 0:
        print("AILab", end=" ")
    elif num % 3 == 0:
        print("AI", end=" ")
    elif num % 5 == 0:
        print("Lab", end=" ")
    else:
        print(num, end=" ")
```

Output:



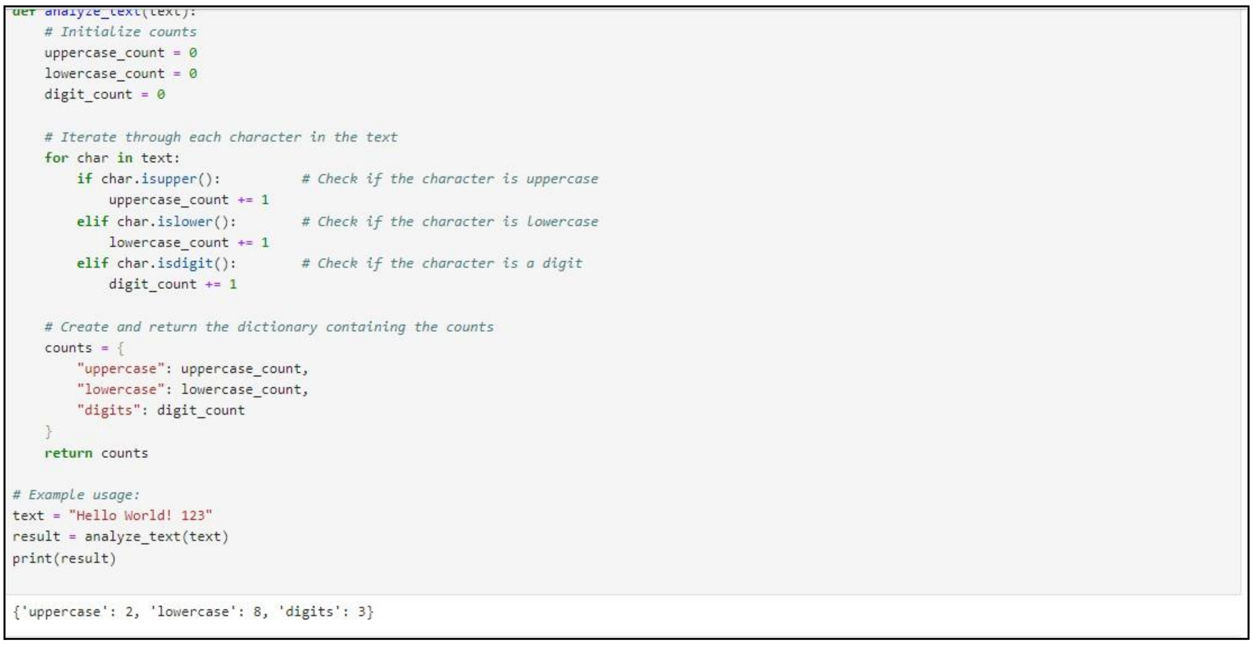
```
[32]: for num in range(1, 101):
          if num % 3 == 0 and num % 5 == 0:
              print("AILab", end=" ")
          elif num % 3 == 0:
              print("AI", end=" ")
          elif num % 5 == 0:
              print("Lab", end=" ")
          else:
              print(num, end=" ")
```

```
1 2 AI 4 Lab AI 7 8 AI Lab 11 AI 13 14 AILab 16 17 AI 19 Lab AI 22 23 AI Lab 26 AI 28 29 AILab 31 32 AI 34 Lab AI 37 38 AI Lab 41 AI 43 44 AILab 46 47 AI
49 Lab AI 52 53 AI Lab 56 AI 58 59 AILab 61 62 AI 64 Lab AI 67 68 AI Lab 71 AI 73 74 AILab 76 77 AI 79 Lab AI 82 83 AI Lab 86 AI 88 89 AILab 91 92 AI 94
Lab AI 97 98 AI Lab
```

Part 2: Text Analysis Function

```
def analyze_text(text):
    # Initialize counts
    uppercase_count = 0
    lowercase_count = 0
    digit_count = 0

    # Iterate through each character in the text
    for char in text:
        if char.isupper():          # Check if the character is uppercase
            uppercase_count += 1
        elif char.islower():        # Check if the character is lowercase
            lowercase_count += 1
        elif char.isdigit():        # Check if the character is a digit
            digit_count += 1

    # Create and return the dictionary containing the counts
    counts = {
        "uppercase": uppercase_count,
        "lowercase": lowercase_count,
        "digits": digit_count
    }
    return counts

# Example usage:
text = "Hello World! 123"
result = analyze_text(text)
print(result)
```

Output:



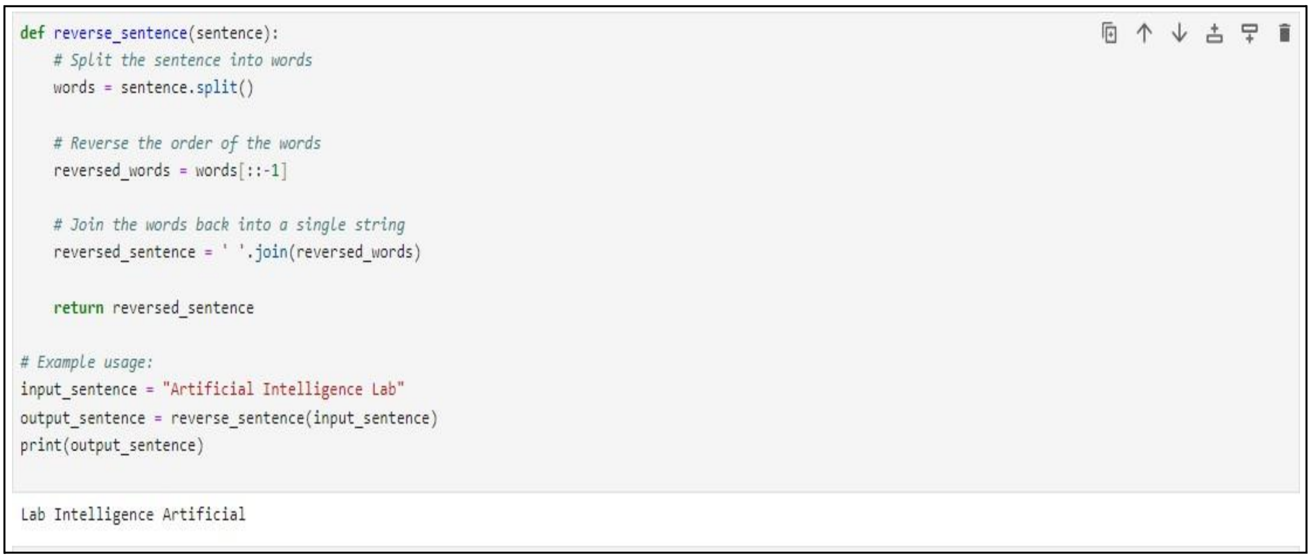
```
def analyze_text(text):
    # Initialize counts
    uppercase_count = 0
    lowercase_count = 0
    digit_count = 0

    # Iterate through each character in the text
    for char in text:
        if char.isupper():          # Check if the character is uppercase
            uppercase_count += 1
        elif char.islower():        # Check if the character is lowercase
            lowercase_count += 1
        elif char.isdigit():        # Check if the character is a digit
            digit_count += 1

    # Create and return the dictionary containing the counts
    counts = {
        "uppercase": uppercase_count,
        "lowercase": lowercase_count,
        "digits": digit_count
    }
    return counts

# Example usage:
text = "Hello World! 123"
result = analyze_text(text)
print(result)

{'uppercase': 2, 'lowercase': 8, 'digits': 3}
```

Part 3: Word Reversal in Strings

```
def  reverse_sentence(sentence):
    # Split the sentence into words
    words = sentence.split()

    # Reverse the order of the words
    reversed_words = words[::-1]

    # Join the words back into a single string
    reversed_sentence = ' '.join(reversed_words)

    return reversed_sentence
```

```
# Example usage:
input_sentence = "Artificial Intelligence Lab"
output_sentence = reverse_sentence(input_sentence)
print(output_sentence)
```

Output:



```
def reverse_sentence(sentence):
    # Split the sentence into words
    words = sentence.split()

    # Reverse the order of the words
    reversed_words = words[::-1]

    # Join the words back into a single string
    reversed_sentence = ' '.join(reversed_words)

    return reversed_sentence

# Example usage:
input_sentence = "Artificial Intelligence Lab"
output_sentence = reverse_sentence(input_sentence)
print(output_sentence)
```

```
Lab Intelligence Artificial
```

RUBRICS:

Performance			Lab Report		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ————————————

PRACTICAL NO.06
Implementing BFS and DFS for the 8-Puzzle Problem

PLO	CLO
2	2

Objectives:

The goal of this lab is to understand and implement the Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms to solve the 8-puzzle problem, a classic problem in Artificial Intelligence.

Learning Outcome:

Students will gain practical experience in implementing and understanding search algorithms, specifically BFS and DFS, and their applications in solving combinatorial problems like the 8-puzzle problem.

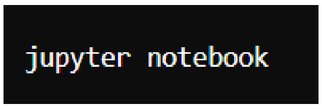
Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:



```
jupyter notebook
```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Task 1: Understanding the Problem**

   - Research the 8-puzzle problem and familiarize yourself with the rules and goal state.
   - Define the initial state, goal state, and valid moves that can be performed in the puzzle.

4. **Task 2: Implementing BFS**

   - Write a Python function to perform BFS on the 8-puzzle problem.
   - Your function should take an initial state as input and return the sequence of moves to reach the goal state, or indicate if the goal state is not reachable.

5. Task 3: Implementing DFS

- Write a Python function to perform DFS on the 8-puzzle problem.
- Similar to BFS, your function should take an initial state as input and return the sequence of moves to reach the goal state, or indicate if the goal state is not reachable.

6. Analysis

- Compare the performance of BFS and DFS in terms of time complexity, space complexity, and the length of the solution path.
- Discuss the advantages and disadvantages of using BFS and DFS for the 8-puzzle problem.

Expected Outcome:

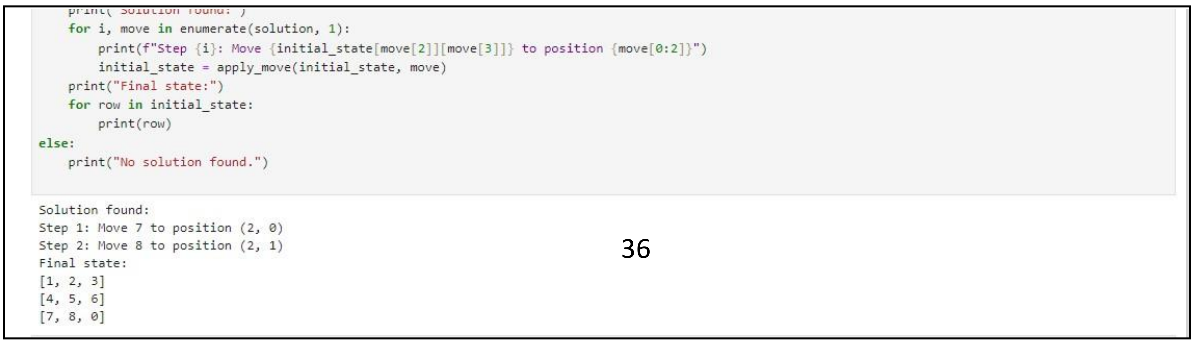
By the end of this experiment, students should be able to implement BFS and DFS algorithms to solve the 8-puzzle problem, compare their performance in terms of time and space complexity, and analyze their advantages and disadvantages.

SOLUTION:

Task : Breadth-First Search (BFS) Code

```
# Function to check if a state is the goal state
def is_goal_state(state):
    return state == [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
# Function to find the possible moves from a given state
def get_possible_moves(state):
    moves = []
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                if i > 0:
                    moves.append((i, j, i - 1, j)) # Move empty space up
                if i < 2:
                    moves.append((i, j, i + 1, j)) # Move empty space down
                if j > 0:
                    moves.append((i, j, i, j - 1)) # Move empty space left
                if j < 2:
                    moves.append((i, j, i, j + 1)) # Move empty space right
    return moves
# Function to apply a move to a given state
def apply_move(state, move):
    i, j, new_i, new_j = move
    new_state = [row[:] for row in state]
    new_state[i][j], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[i][j]
    return new_state

# Breadth-First Search algorithm
def bfs(start_state):
    queue = [(start_state, [])] # Queue of states and their paths
    visited = set() # Set to keep track of visited states

    while queue:
        state, path = queue.pop(0)
        if is_goal_state(state):
            return path
        visited.add(tuple(map(tuple, state))) # Convert state to tuple for hashing

        for  move  in  get_possible_moves(state):
            new_state = apply_move(state, move)
            if tuple(map(tuple, new_state)) not in visited:
                queue.append((new_state, path + [move]))
                visited.add(tuple(map(tuple, new_state)))

    return None # No solution found

# Example usage:
initial_state = [[1, 2, 3], [4, 5, 6], [0, 7, 8]] # Initial state
solution = bfs(initial_state)
if solution:
    print("Solution found:")
    for i, move in enumerate(solution, 1):
        print(f"Step {i}: Move {initial_state[move[2]][move[3]]} to position {move[0:2]}")
        initial_state = apply_move(initial_state, move)
    print("Final  state:")
    for row in initial_state:
        print(row)
else:
    print("No solution found.")
```

Output:



```
    print("Solution found:")
    for i, move in enumerate(solution, 1):
        print(f"Step {i}: Move {initial_state[move[2]][move[3]]} to position {move[0:2]}")
        initial_state = apply_move(initial_state, move)
    print("Final state:")
    for row in initial_state:
        print(row)
else:
    print("No solution found.")
```

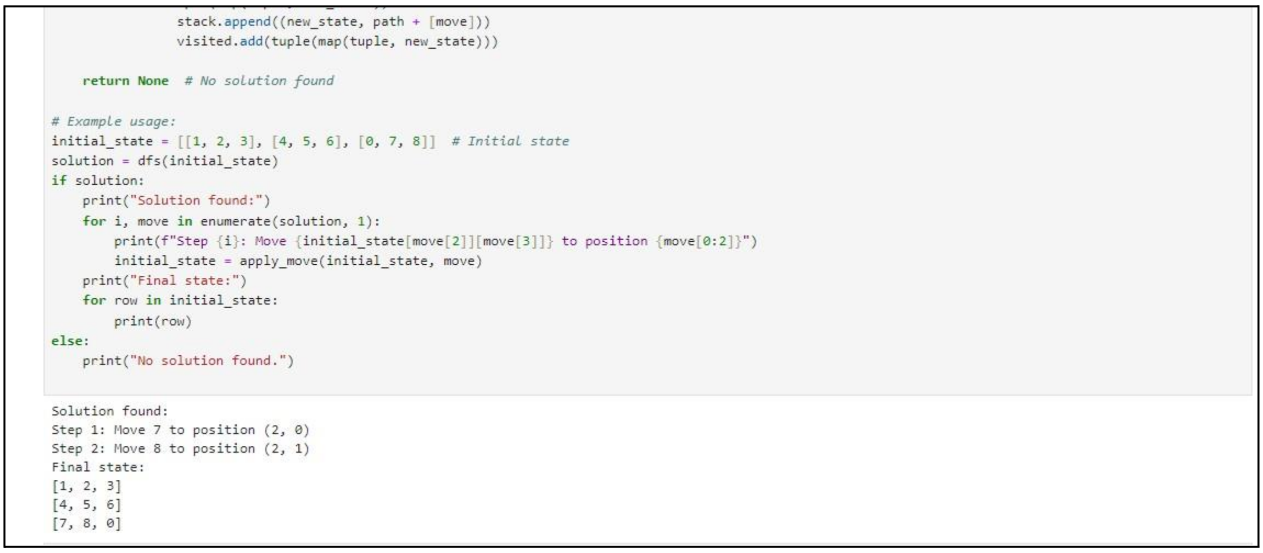
```
Solution found:
Step 1: Move 7 to position (2, 0)
Step 2: Move 8 to position (2, 1)
Final state:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

<u>Task : Depth First Search (DFS)</u>

```
# Function to check if a state is the goal state
def is_goal_state(state):
    return state == [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

# Function to find the possible moves from a given state
def get_possible_moves(state):
    moves = []
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                if i > 0:
                    moves.append((i, j, i - 1, j)) # Move empty space up
                if i < 2:
                    moves.append((i, j, i + 1, j)) # Move empty space down
                if j > 0:
                    moves.append((i, j, i, j - 1)) # Move empty space left
                if j < 2:
                    moves.append((i, j, i, j + 1)) # Move empty space right
    return moves

# Function to apply a move to a given state
def apply_move(state, move):
    i, j, new_i, new_j = move
    new_state = [row[:] for row in state]
    new_state[i][j], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[i][j]
    return new_state

# Depth-First Search algorithm
def dfs(start_state):
    stack = [(start_state, [])] # Stack of states and their paths
    visited = set() # Set to keep track of visited states

    while stack:
        state, path = stack.pop()
        if is_goal_state(state):
            return path
        visited.add(tuple(map(tuple, state))) # Convert state to tuple for hashing

        for move in get_possible_moves(state):
            new_state = apply_move(state, move)
            if tuple(map(tuple, new_state)) not in visited:
                stack.append((new_state, path + [move]))
                visited.add(tuple(map(tuple, new_state)))

    return None # No solution found

# Example usage:
```

```
initial_state = [[1, 2, 3], [4, 5, 6], [0, 7, 8]] # Initial state
solution = dfs(initial_state)
if  solution:
    print("Solution found:")
    for i, move in enumerate(solution, 1):
        print(f"Step {i}: Move {initial_state[move[2]][move[3]]} to position {move[0:2]}")
        initial_state = apply_move(initial_state, move)
    print("Final  state:")
    for row in initial_state:
        print(row)
else:
    print("No solution found.")
```

Output:



```
                stack.append((new_state, path + [move]))
                visited.add(tuple(map(tuple, new_state)))

    return None  # No solution found

# Example usage:
initial_state = [[1, 2, 3], [4, 5, 6], [0, 7, 8]]  # Initial state
solution = dfs(initial_state)
if solution:
    print("Solution found:")
    for i, move in enumerate(solution, 1):
        print(f"Step {i}: Move {initial_state[move[2]][move[3]]} to position {move[0:2]}")
        initial_state = apply_move(initial_state, move)
    print("Final state:")
    for row in initial_state:
        print(row)
else:
    print("No solution found.")
```

```
Solution found:
Step 1: Move 7 to position (2, 0)
Step 2: Move 8 to position (2, 1)
Final state:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

RUBRICS:

Performance			Lab Report		
Description	Total Marks	Marks Obtained	Description	Total Marks	Marks Obtained
Ability to Conduct practical	5		Structure	5	
Data Analysis & Interpretation	5		Efficiency	5	
Total Marks obtained			Total Marks Obtained		

PRACTICAL NO.07
Implementing Uniform Cost Search and Greedy Best First Search for Pathfinding

PLO	CLO
2	2

Objectives:

The goal of this lab is to understand and implement the Uniform Cost Search (UCS) and Greedy Best First Search (GBFS) algorithms to solve a pathfinding problem, which is a fundamental problem in Artificial Intelligence.

Learning Outcome:

Students will be able to implement and compare the UCS and GBFS algorithms for pathfinding, analyze their performance characteristics, and discuss their advantages and disadvantages.

Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

   ```
   jupyter notebook
   ```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Task 1: Understanding the Problem**

   - Research pathfinding and the specific algorithms: UCS and GBFS.
   - Define a graph representing the pathfinding problem, including nodes, edges, and costs or heuristics.
   - Establish the initial state, goal state, and the criteria for a valid path

4. **Task 2: Implementing UCS**

   - Write a Python function to perform UCS on the defined pathfinding problem.
   - Your function should take an initial state and goal state as input and return the least cost path, or indicate if the goal state is not reachable.

5. Task 3: Implementing GBFS

- Write a Python function to perform GBFS on the pathfinding problem.
- Similar to UCS, your function should take an initial state and goal state as input and return the path that appears to be closest to the goal state, or indicate if the goal state is not reachable.

6. Analysis

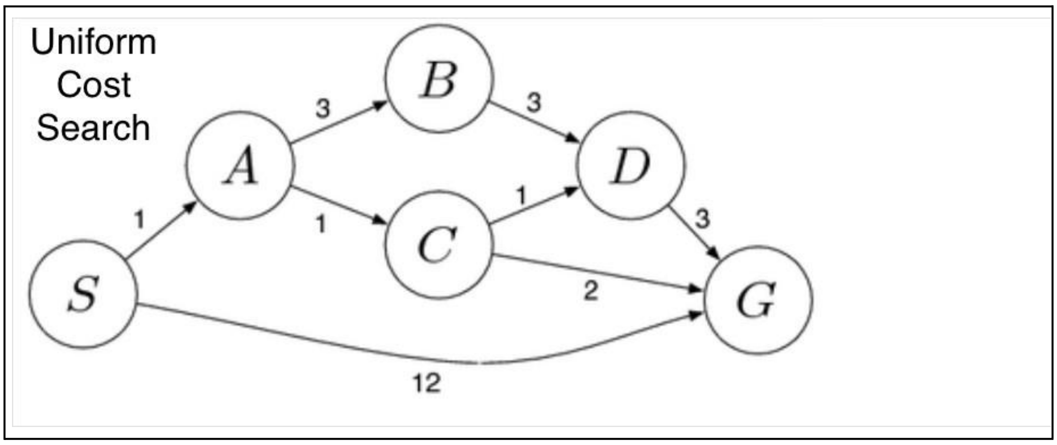
- Compare the performance of UCS and GBFS in terms of time complexity, space complexity, and the optimality of the solution path.
- Discuss the advantages and disadvantages of using UCS and GBFS for the pathfinding problem.

Expected Outcome:

By the end of this experiment, students should have implemented UCS and GBFS algorithms for pathfinding, analyzed their performance characteristics, and discussed their applicability in different scenarios.

SOLUTION:

Task 1: Defining a graph



Graph:

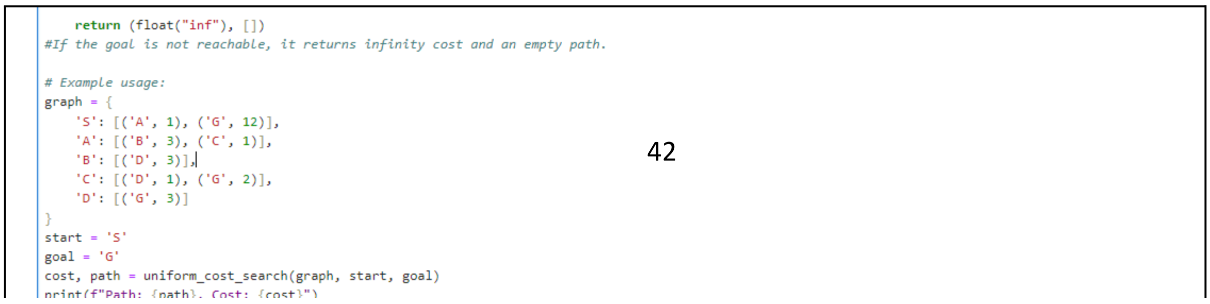
```
graph = {
    'S': [('A', 1), ('G', 12)],
    'A': [('B', 3), ('C', 1)],
    'B': [('D', 3)],
    'C': [('D', 1), ('G', 2)],
    'D': [('G', 3)]
}
start = 'S'
goal = 'G'
```

Nodes: S, A, B, C, D, G

Task 2: Implementing Uniform Cost Search (UCS)

```
import heapq # heapq,a pthon libaray for piority queues

def uniform_cost_search(graph, start, goal):
    visited = set()
    priority_queue = [(0, start, [])] # (cost, vertex, path)

#pops the tuple with the smallest cost from the priority queue.
    while priority_queue:
        (cost, vertex, path) = heapq.heappop(priority_queue)

# checks if the current vertex vertex has not been visited before.
#If the vertex has not been visited, it is added to the set of visited vertices.
        if vertex not in visited:
            visited.add(vertex)
            path = path + [vertex] # This appends the current vertex to the path

            if vertex == goal:
                return (cost, path)

#This iterates over the neighbors of the current vertex and their associated weights.
#If the neighbor has not been visited, it adds it to the priority queue with updated cost and path.

            for next_node, weight in graph.get(vertex, []):
                if next_node not in visited:
                    heapq.heappush(priority_queue, (cost + weight, next_node, path))

    return (float("inf"), [])
#If the goal is not reachable, it returns infinity cost and an empty path.

# Example usage:
graph = {
    'S': [('A', 1), ('G', 12)],
    'A': [('B', 3), ('C', 1)],
    'B': [('D', 3)],
    'C': [('D', 1), ('G', 2)],
    'D': [('G', 3)]
}
start = 'S'
goal = 'G'
cost, path = uniform_cost_search(graph, start, goal)
print(f"Path: {path}, Cost: {cost}")
```

Output:

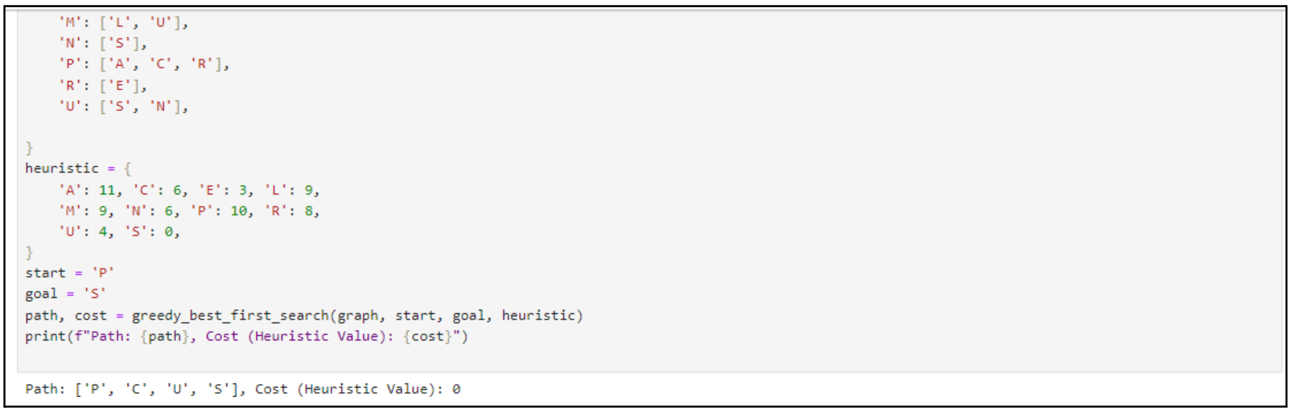


```
    return (float("inf"), [])
#If the goal is not reachable, it returns infinity cost and an empty path.

# Example usage:
graph = {
    'S': [('A', 1), ('G', 12)],
    'A': [('B', 3), ('C', 1)],
    'B': [('D', 3)],
    'C': [('D', 1), ('G', 2)],
    'D': [('G', 3)]
}
start = 'S'
goal = 'G'
cost, path = uniform_cost_search(graph, start, goal)
print(f"Path: {path}, Cost: {cost}")
```

Task 3: Implementing GBFS

```
import heapq

def greedy_best_first_search(graph, start, goal, heuristic):
    visited = set()

#nitializes a priority queue with a single tuple containing three elements: the heuristic value of
#the starting vertex start, the starting vertex itself, and an empty path list.

    priority_queue = [(heuristic[start], start, [])] # (heuristic value, vertex, path)

#ops the tuple with the smallest heuristic value from the priority queue
#and unpacks it into three variables: (h_value, vertex, and path.)

    while priority_queue:
        (h_value, vertex, path) = heapq.heappop(priority_queue)
        if vertex not in visited:
            visited.add(vertex)
            path = path + [vertex]

            if vertex == goal:
                return path, h_value # Return both the path and the heuristic value

#iterates over the neighbors of the current vertex vertex using the get() method on the graph
dictionary.
#It returns an empty list [] if the current vertex has no neighbors.

            for next_node in graph.get(vertex, []):
                if next_node not in visited:
                    heapq.heappush(priority_queue, (heuristic[next_node], next_node, path))

    return [], float("inf") # Return an empty path and infinity cost if no path found

# Example usage:
graph = {
    'A': ['M'],
    'C': ['R', 'M', 'U'],
    'E': ['U', 'S'],
    'L': ['N'],
    'M': ['L', 'U'],
    'N': ['S'],
    'P': ['A', 'C', 'R'],
    'R': ['E'],
    'U': ['S', 'N'],

}
heuristic = {
    'A': 11, 'C': 6, 'E': 3, 'L': 9,
    'M': 9, 'N': 6, 'P': 10, 'R': 8,
    'U': 4, 'S': 0,
}
start = 'P'
goal = 'S'
path, cost = greedy_best_first_search(graph, start, goal, heuristic)
print(f"Path: {path}, Cost (Heuristic Value): {cost}")
```

Output:



```
    'M': ['L', 'U'],
    'N': ['S'],
    'P': ['A', 'C', 'R'],
    'R': ['E'],
    'U': ['S', 'N'],

}
heuristic = {
    'A': 11, 'C': 6, 'E': 3, 'L': 9,
    'M': 9, 'N': 6, 'P': 10, 'R': 8,
    'U': 4, 'S': 0,
}
start = 'P'
goal = 'S'
path, cost = greedy_best_first_search(graph, start, goal, heuristic)
print(f"Path: {path}, Cost (Heuristic Value): {cost}")

Path: ['P', 'C', 'U', 'S'], Cost (Heuristic Value): 0
```

RUBRICS:

Performance			Lab Report		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ——————————

PRACTICAL NO.08
Implementing A* Search for Pathfinding

PLO	CLO
2	2

Objectives:

The aim of this lab task is to comprehend and implement the A* Search algorithm to address a pathfinding problem, which is a pivotal challenge in the field of Artificial Intelligence.

Learning Outcome:

Students will be able to implement the A* Search algorithm, design heuristic functions, analyze its performance characteristics, and compare it with other pathfinding algorithms.

Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

   ```
   jupyter notebook
   ```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Task 1: Understanding the Problem**

   - Research pathfinding and the A* algorithm.
   - Define a graph that represents your pathfinding scenario, including nodes, edges, and associated costs or heuristic values.
   - Determine the initial state, goal state, and the criteria for a successful path.

4. **Task 2: Implementing A* Search**

   - Develop a Python function to execute the A* Search on your pathfinding graph.
   - The function should accept an initial state and a goal state as inputs and return the most cost-effective path or indicate if the goal state is unreachable.

5. Task 3 Heuristic Function Design

- Design a heuristic function that estimates the cost from the current node to the goal node.
- Explain the rationale behind your chosen heuristic and how it influences the A* algorithm's behavior.

6. Analysis

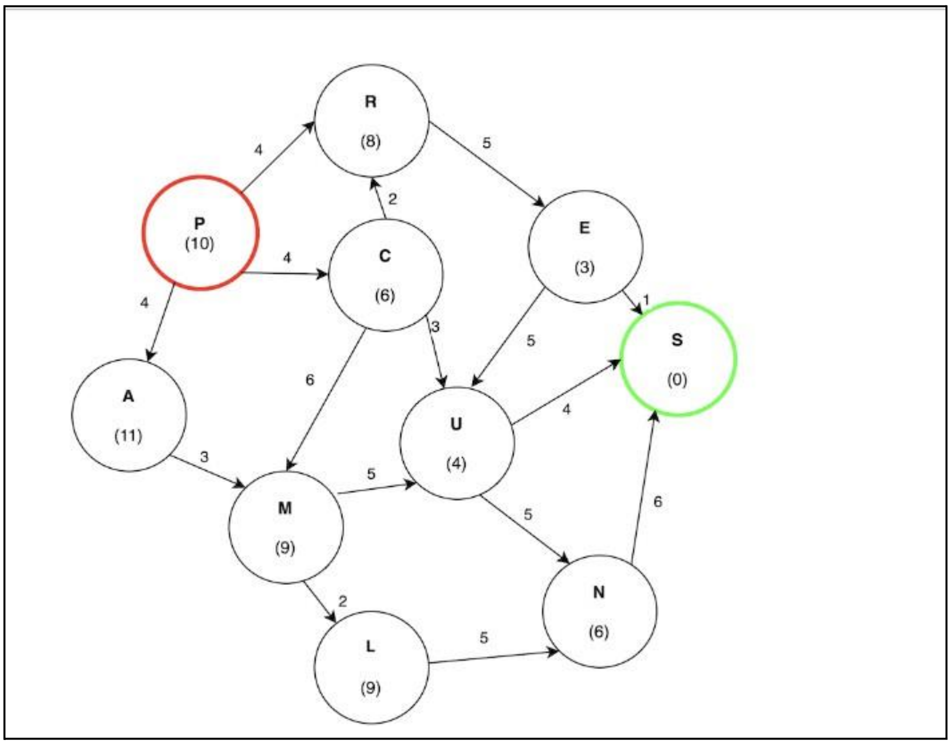
- Analyze the performance of the A* algorithm in terms of time complexity, space complexity, and the optimality of the solution path.
- Compare the A* algorithm with UCS and GBFS, highlighting the differences in their approaches to pathfinding.

Expected Outcome:

By the end of this lab task, students should have implemented the A* Search algorithm for pathfinding, designed and explained a heuristic function, analyzed its performance characteristics, and compared it with other search algorithms in terms of efficiency and effectiveness.

SOLUTION:

Task 1: Defining the graph



Nodes: A, P, R, E, S, C, U, N, L, M

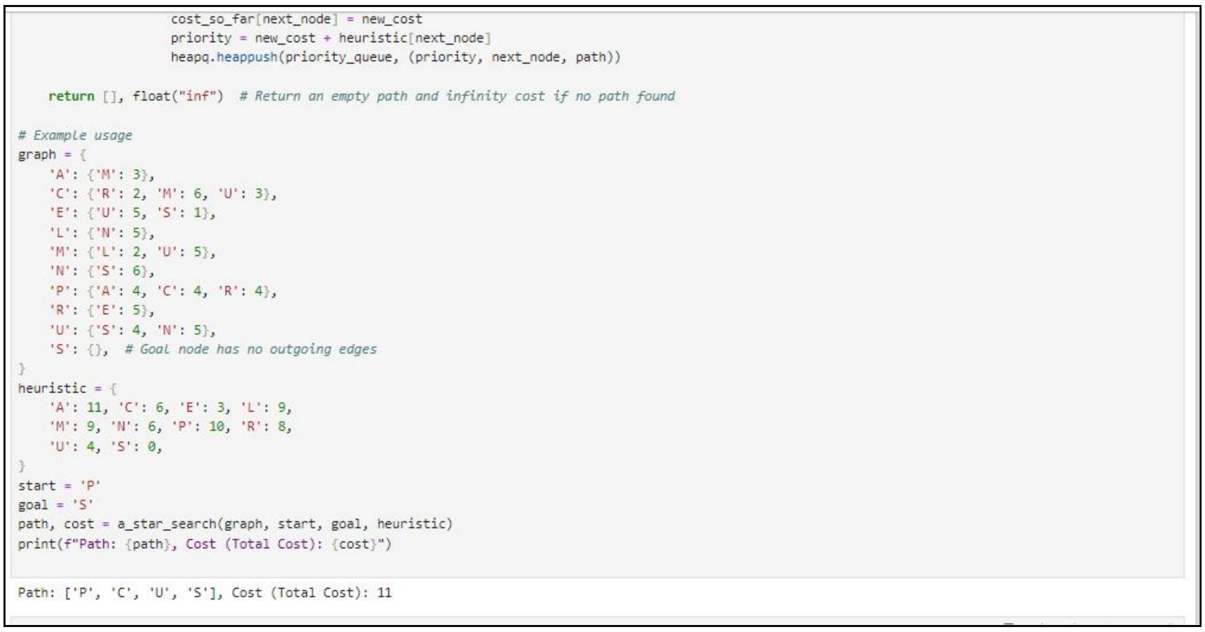
Graph:

```
graph = {
    'A': {'M': 3},
    'C': {'R': 2, 'M': 6, 'U': 3},
    'E': {'U': 5, 'S': 1},
    'L': {'N': 5},
    'M': {'L': 2, 'U': 5},
    'N': {'S': 6},
    'P': {'A': 4, 'C': 4, 'R': 4},
    'R': {'E': 5},
    'U': {'S': 4, 'N': 5},
    'S': {}, # Goal node has no outgoing edges
}
heuristic = {
    'A': 11, 'C': 6, 'E': 3, 'L': 9,
    'M': 9, 'N': 6, 'P': 10, 'R': 8,
    'U': 4, 'S': 0,
}
start = 'P'
goal = 'S'
```

Task 2: Implementing A* Search

```
import heapq
def a_star_search(graph, start, goal, heuristic):
    visited = set()
    priority_queue = [(0, start, [])]  # (priority, vertex, path)
    cost_so_far = {start: 0} # Dictionary to store actual cost from start to each node

    while priority_queue:
        current_priority, vertex, path = heapq.heappop(priority_queue)

        if vertex == goal:
            return path + [vertex], cost_so_far[vertex] # Return the path and the total cost (g(x))

        if vertex not in visited:
            visited.add(vertex)
            path = path + [vertex]

            for next_node, weight in graph.get(vertex, {}).items():
                new_cost = cost_so_far[vertex] + weight # Add actual cost to move
                if next_node not in cost_so_far or new_cost < cost_so_far[next_node]:
                    cost_so_far[next_node] = new_cost
                    priority = new_cost + heuristic[next_node]
                    heapq.heappush(priority_queue, (priority, next_node, path))

    return [], float("inf")
```

Output:



```
                cost_so_far[next_node] = new_cost
                priority = new_cost + heuristic[next_node]
                heapq.heappush(priority_queue, (priority, next_node, path))

    return [], float("inf")  # Return an empty path and infinity cost if no path found

# Example usage
graph = {
    'A': {'M': 3},
    'C': {'R': 2, 'M': 6, 'U': 3},
    'E': {'U': 5, 'S': 1},
    'L': {'N': 5},
    'M': {'L': 2, 'U': 5},
    'N': {'S': 6},
    'P': {'A': 4, 'C': 4, 'R': 4},
    'R': {'E': 5},
    'U': {'S': 4, 'N': 5},
    'S': {},  # Goal node has no outgoing edges
}
heuristic = {
    'A': 11, 'C': 6, 'E': 3, 'L': 9,
    'M': 9, 'N': 6, 'P': 10, 'R': 8,
    'U': 4, 'S': 0,
}
start = 'P'
goal = 'S'
path, cost = a_star_search(graph, start, goal, heuristic)
print(f"Path: {path}, Cost (Total Cost): {cost}")

Path: ['P', 'C', 'U', 'S'], Cost (Total Cost): 11
```

Task 3: Heuristic Function Design

```
def heuristic_distance(state):
    goal_positions = {
        1: (0, 0), 2: (0, 1), 3: (0, 2),
        4: (1, 0), 5: (1, 1), 6: (1, 2),
        7: (2, 0), 8: (2, 1), 0: (2, 2)
    }
    distance = 0
    for i in range(3):
        for j in range(3):
            value = state[i][j]
            if value != 0:
                goal_i, goal_j = goal_positions[value]
                distance += abs(i - goal_i) + abs(j - goal_j)
    return distance
```

The heuristic distance function influences the A* algorithm in the following ways:

1. **Guiding the Search:** It provides an estimate of the cost to reach the goal from the current node, helping the algorithm prioritize nodes that are likely to lead to a solution more quickly.

2. **Optimizing Pathfinding**: By calculating the total estimated cost (actual cost + heuristic), A* can efficiently determine the most promising path, balancing between exploring new nodes and expanding known paths.

3. **Efficiency:** A good heuristic like Manhattan distance ensures that the algorithm explores fewer nodes, reducing computation time and resources while still finding the optimal solution.

Overall, the heuristic significantly impacts the efficiency and effectiveness of the A* search.

RUBRICS:

Performance			**Lab Report**		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ____________________

PRACTICAL NO.09
Dataset Acquisition from Public Repositories

PLO	**CLO**	
4	3	

Objectives:

To acquire datasets from publicly available repositories for use in future Artificial Intelligence tasks.

Learning Outcome:

Students will gain hands-on experience in exploring and selecting datasets from public repositories, understanding dataset characteristics suitable for AI tasks such as classification, regression, and clustering.

Materials Required:

- A computer with internet access

- Access to platforms like Kaggle or the UCI Machine Learning Repository

Activity:

1. **Explore Repositories:**
   - Navigate through platforms like Kaggle or the UCI Machine Learning Repository.
   - Familiarize yourself with the types of datasets available and their applications in AI.
2. **Select a Dataset:**
   - Choose a dataset that interests you and is suitable for multiple AI tasks such as classification, regression, and clustering.
   - Consider factors like dataset size, complexity, and relevance to AI research.
3. **Download the Dataset:**
   - Obtain the selected dataset from the repository and save it to your local environment.
   - Ensure that you have all necessary permissions and rights to use the dataset for educational purposes.
4. **Deliverables:**
   - Include the name of the dataset and a brief description.
   - Describe the dataset's attributes, such as number of instances, features, and target variables.
   - Discuss potential applications of the dataset in AI tasks like classification, regression, and clustering.

Expected Outcome:

By the end of this lab task, students should have explored various public repositories and selected a dataset suitable for AI tasks. They will have successfully downloaded and documented the dataset, including its key attributes and potential applications in AI. This exercise enhances students' proficiency in dataset acquisition, preparation, and evaluation, essential skills for conducting effective AI research and applications.

SOLUTION:

Dataset

Dataset Name: Food Nutrition Dataset

Overview:

The Comprehensive Nutritional Food Database provides detailed nutritional information for a wide range of food items commonly consumed around the world. This dataset aims to support dietary planning, nutritional analysis, and educational purposes by providing extensive data on the macro and micronutrient content of foods.

Data Format:

The dataset is structured as a CSV (Comma-Separated Values) file, which can easily be imported into most data analysis tools and software for further processing and analysis.

Column Descriptions: This dataset contains following columns:

1. Food.
2. Caloric Value
3. Fat(in g)
4. Saturated Fats(in g)
5. Monounsaturated Fats(in g)
6. Polyunsaturated Fats(in g)
7. Carbohydrates(in g)
8. Sugars(in g)
9. Protein(in g)
10. Dietary Fiber(in g)
11. Cholesterol(in mg)
12. Sodium(in g)
13. Water(in g)
14. Vitamin A(in mg)
15. Vitamin B1 (Thiamine)
16. Vitamin B11 (Folic Acid)
17. Vitamin B12(in mg)
18. Vitamin B2 (Riboflavin)(in mg)
19. Vitamin B3 (Niacin)(in mg)
20. Vitamin B5 (Pantothenic Acid)(in mg)
21. Vitamin B6(in mg)
22. Vitamin C(in mg)
23. Vitamin D(in mg)
24. Vitamin E(in mg)
25. Vitamin K(in mg)
26. Calcium(in mg)
27. Copper(in mg)
28. Iron(in mg)
29. Magnesium(in mg)
30. Manganese(in mg)
31. Phosphorus(in mg)

32. Potassium(in mg)
33. Selenium(in mg)
34. Zinc(in mg)
35. Nutrition Density

Use Cases:

This dataset is invaluable for researchers in nutritional science, dietitians planning meals, healthcare providers advising on dietary options, and individuals tracking their food intake. It can be used to:

1. Nutritional Pattern Analysis

Machine learning algorithms can analyze trends and patterns in candy consumption, linking them to nutritional impacts. For example, clustering techniques can identify common characteristics of high-calorie candies or those high in certain nutrients like sugars.

2. Diet Recommendation Systems

By integrating this dataset with broader dietary data, machine learning models can recommend dietary adjustments to individuals. For instance, a recommendation system could suggest lower-calorie or lower-sugar candy alternatives to users looking to reduce their sugar intake but who still want to enjoy sweets.

3. Predictive Modeling for Health Impacts

With sufficient data linking candy consumption to health outcomes, predictive models could forecast health impacts based on candy consumption patterns. This could be particularly useful for medical research or public health studies examining the effects of sugar and additives on long-term health.

4. Ingredient Optimization

Machine learning can help in formulating new candy recipes by predicting the nutritional content based on ingredients. This could aid manufacturers in creating healthier candy options that maintain taste while reducing undesirable nutrients like sugars or synthetic additives.

5. Consumer Behavior Analysis

Using classification or regression models, businesses can predict consumer preferences for certain types of candies based on nutritional information and demographic data. This insight can drive targeted marketing and product development strategies.

CONCLUSION:

The Food Nutrition Dataset's detailed nutritional information makes it a valuable resource for researchers, dietitians, healthcare providers, and individuals tracking their food intake. By leveraging this dataset in various AI applications, we can enhance nutritional analysis, dietary planning, and health-related research, ultimately contributing to better dietary habits and health outcomes. This exercise underscores the importance of dataset acquisition, preparation, and evaluation in conducting effective AI research and applications.

RUBRICS:

Performance			**Lab Report**		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ———————————

PRACTICAL NO.10
Preprocessing Datasets for Machine Learning

PLO	CLO
4	3

Objectives:

To apply basic preprocessing techniques to a dataset, making it ready for machine learning applications.

Learning Outcome:

Students will gain practical experience in performing essential data preprocessing steps, such as handling missing values and duplicates, to prepare datasets for machine learning tasks.

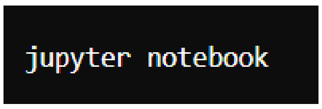
Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

   ```
   jupyter notebook
   ```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Preprocess the Dataset:**

   Perform basic preprocessing steps such as:

   - Handling missing values: Identify and fill or remove missing data points.
   - Handling duplicated values: Identify and remove duplicated records if applicable.
   - Data normalization or standardization if necessary.
   - Any other relevant preprocessing steps specific to the dataset.

Expected Outcome:

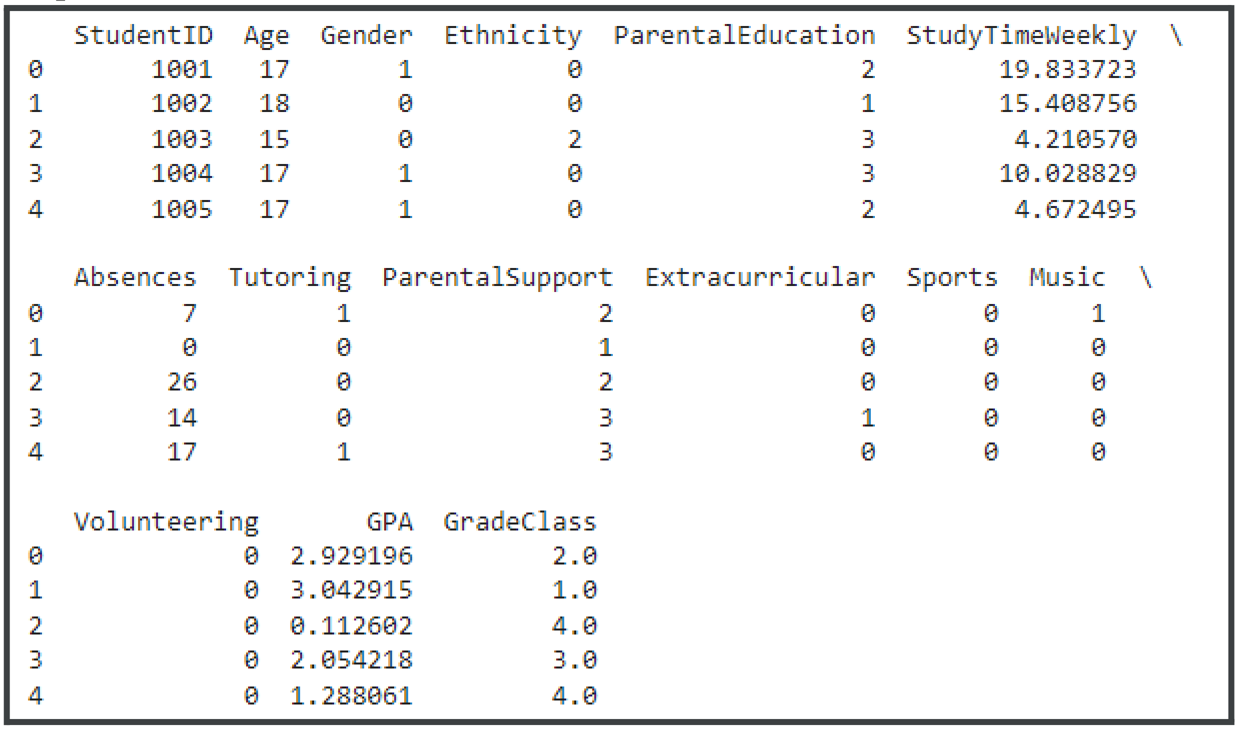
By the end of this lab task, students should have successfully applied basic preprocessing techniques to a dataset, addressing issues such as missing values and duplicates. They will have documented the preprocessing steps performed and presented the dataset in a prepared format ready for machine learning applications. This exercise enhances students' proficiency in data preprocessing, a critical skill for effective machine learning model development and deployment.

SOLUTION:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore
```

```
# Load the dataset
df = pd.read_csv('/content/sample_data/students-performance-dataset.csv')

# Display the first few rows to understand the structure
print("First few rows of the dataset:")
print(df.head())
```

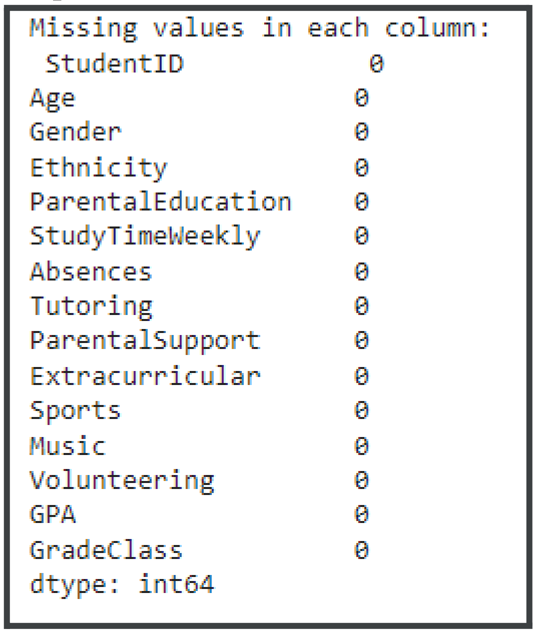
Output :



```
   StudentID  Age  Gender  Ethnicity  ParentalEducation  StudyTimeWeekly  \
0       1001   17       1          0                  2        19.833723
1       1002   18       0          0                  1        15.408756
2       1003   15       0          2                  3         4.210570
3       1004   17       1          0                  3        10.028829
4       1005   17       1          0                  2         4.672495

   Absences  Tutoring  ParentalSupport  Extracurricular  Sports  Music  \
0         7         1                2                0       0      1
1         0         0                1                0       0      0
2        26         0                2                0       0      0
3        14         0                3                1       0      0
4        17         1                3                0       0      0

   Volunteering       GPA  GradeClass
0             0  2.929196         2.0
1             0  3.042915         1.0
2             0  0.112602         4.0
3             0  2.054218         3.0
4             0  1.288061         4.0
```

```
# Check for missing values
missing_values = df.isnull().sum()
print("\nMissing values in each column:\n", missing_values)
```

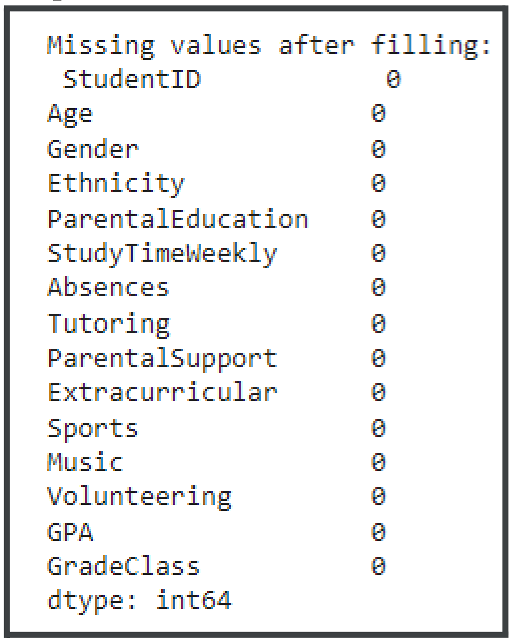
Output:



```
Missing values in each column:
 StudentID            0
Age                  0
Gender               0
Ethnicity            0
ParentalEducation    0
StudyTimeWeekly      0
Absences             0
Tutoring             0
ParentalSupport      0
Extracurricular      0
Sports               0
Music                0
Volunteering         0
GPA                  0
GradeClass           0
dtype: int64
```

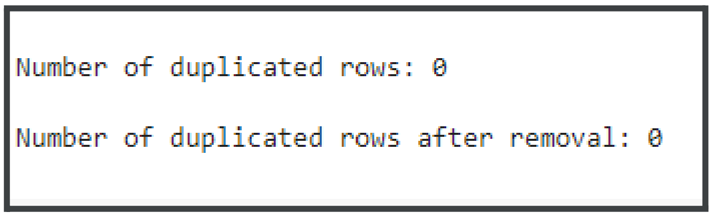
```
# Fill missing values with the median value of each column
df.fillna(df.median(), inplace=True)

# Verify that missing values are handled
print("\nMissing values after filling:\n", df.isnull().sum())
```

Output:



```
Missing values after filling:
 StudentID            0
Age                  0
Gender               0
Ethnicity            0
ParentalEducation    0
StudyTimeWeekly      0
Absences             0
Tutoring             0
ParentalSupport      0
Extracurricular      0
Sports               0
Music                0
Volunteering         0
GPA                  0
GradeClass           0
dtype: int64
```

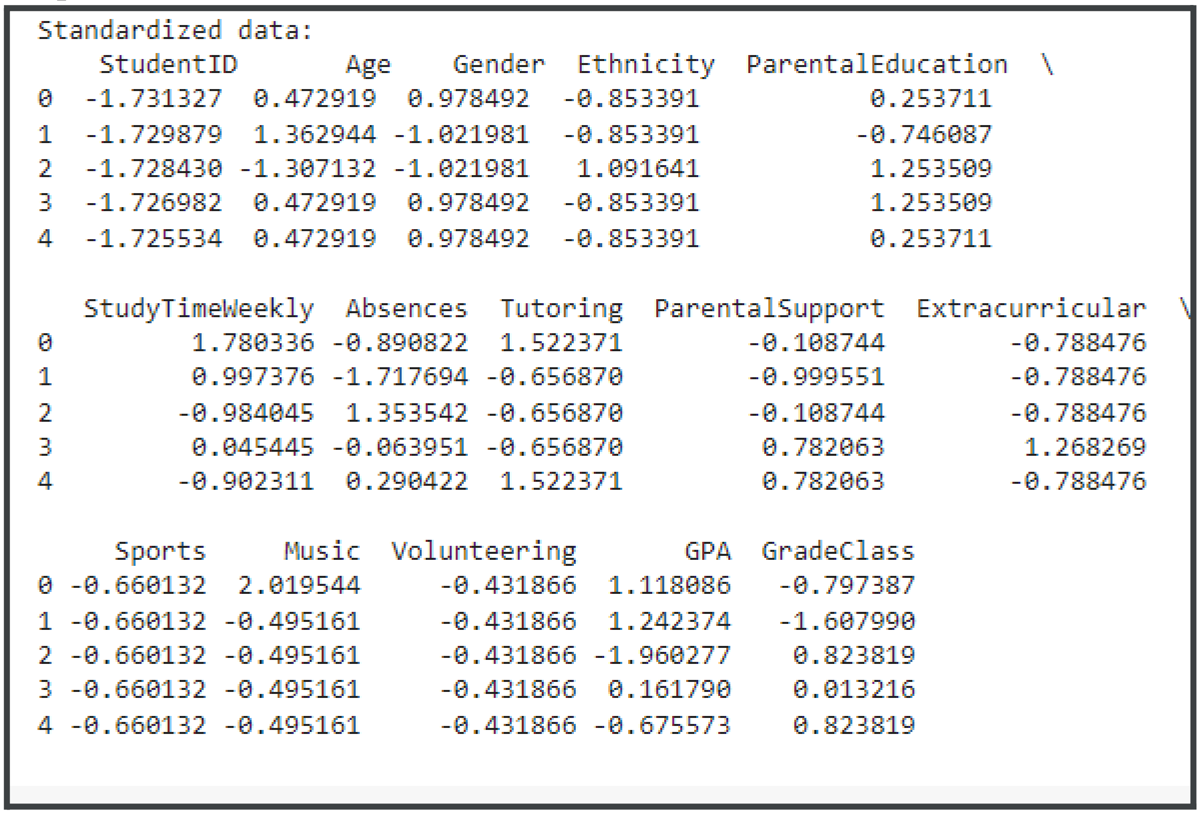
```
# Check for duplicated rows
duplicates = df.duplicated().sum()
print("\nNumber of duplicated rows:", duplicates)

# Remove duplicated rows if any
df.drop_duplicates(inplace=True)

# Verify that duplicates are removed
print("\nNumber of duplicated rows after removal:", df.duplicated().sum())
```



```

Number of duplicated rows: 0

Number of duplicated rows after removal: 0
```

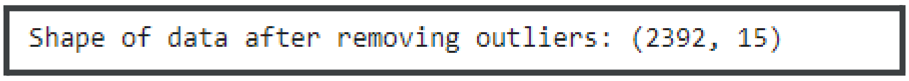
```
# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
print("\nCategorical columns:\n", categorical_cols)

# Apply one-hot encoding
if len(categorical_cols) > 0:
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Display the first few rows of the encoded data
print("\nData after one-hot encoding:\n", df.head())
```

```
# Select numerical columns for standardization
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Initialize the scaler
scaler = StandardScaler()

# Standardize the numerical columns
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Display the first few rows of the standardized data
print("\nStandardized data:\n", df.head())
```
Output:



```
Standardized data:
    StudentID       Age    Gender  Ethnicity  ParentalEducation  \
0  -1.731327  0.472919  0.978492  -0.853391           0.253711
1  -1.729879  1.362944 -1.021981  -0.853391          -0.746087
2  -1.728430 -1.307132 -1.021981   1.091641           1.253509
3  -1.726982  0.472919  0.978492  -0.853391           1.253509
4  -1.725534  0.472919  0.978492  -0.853391           0.253711

   StudyTimeWeekly  Absences  Tutoring  ParentalSupport  Extracurricular  \
0         1.780336 -0.890822  1.522371        -0.108744        -0.788476
1         0.997376 -1.717694 -0.656870        -0.999551        -0.788476
2        -0.984045  1.353542 -0.656870        -0.108744        -0.788476
3         0.045445 -0.063951 -0.656870         0.782063         1.268269
4        -0.902311  0.290422  1.522371         0.782063        -0.788476

     Sports     Music  Volunteering       GPA  GradeClass
0 -0.660132  2.019544     -0.431866  1.118086   -0.797387
1 -0.660132 -0.495161     -0.431866  1.242374   -1.607990
2 -0.660132 -0.495161     -0.431866 -1.960277    0.823819
3 -0.660132 -0.495161     -0.431866  0.161790    0.013216
4 -0.660132 -0.495161     -0.431866 -0.675573    0.823819
```

```
# Calculate Z-scores for numerical columns and filter out rows with Z-scores above a threshold
z_scores = df[numerical_cols].apply(zscore)
df = df[(z_scores < 3).all(axis=1)]

# Display the shape of the data after removing outliers
print("\nShape of data after removing outliers:", df.shape)
```
Output :



```
Shape of data after removing outliers: (2392, 15)
```

CONCLUSION:

In this analysis, we conducted basic preprocessing steps on the "Students Performance" dataset. Here are the key outcomes:

1. Handling Missing Values:
- Initially, we identified that there were no missing values in the dataset.
- As a demonstration, we filled any potential missing values with the median value of each column and verified that all missing values were handled.

2. Handling Duplicated Values:
- We checked for duplicated rows in the dataset and found none.
- We verified that there were no duplicates left after running the duplication removal step.

3. Standardization:
- We standardized the dataset to ensure all numerical features had a mean of 0 and a standard deviation of 1. This step is crucial for machine learning algorithms that are sensitive to the scale of data.

4. Outlier Detection and Removal:
- We used Z-scores to identify and remove outliers from the numerical columns, setting a threshold of 3.
- The shape of the dataset after removing outliers was (2392, 15), indicating that a small portion of data points with extreme values was excluded to improve the overall quality of the dataset.

These preprocessing steps ensure that the dataset is clean, standardized, and free of extreme outliers, making it suitable for further analysis and machine learning tasks. This enhances the reliability and accuracy of any models or insights derived from the dataset.

RUBRICS:

Performance			**Lab Report**		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ————————————

PRACTICAL NO.11
Performing Exploratory Data Analysis with Python

PLO	CLO
4	3

Objectives:

To gain practical experience in conducting exploratory data analysis (EDA) using Python, a crucial step in understanding data before applying machine learning algorithms.

Learning Outcome:

By completing this lab task, students will develop proficiency in selecting datasets suitable for analysis from public repositories, using Python to inspect dataset structures, and creating visualizations to uncover patterns and relationships. This exercise will equip students with essential skills in data exploration and visualization, crucial for making informed decisions in data-driven applications and machine learning model development.

Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook
- Python libraries: Pandas, Matplotlib, Seaborn (install via Anaconda or pip if not already installed)

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

   ```
   jupyter notebook
   ```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Task Description:**

- **Data Selection:** Choose a dataset from a public repository that you find interesting and is suitable for analysis.
- **Data Inspection:** Use Python to inspect the structure of the dataset, such as the number of rows, columns, and types of data.
- **Data Visualization:** Create visualizations using libraries like Matplotlib or Seaborn to uncover patterns, trends, and relationships in the data.

4. **Deliverables:**

- A report summarizing the insights gained from the EDA process, supported by visualizations.
- The Python code used for conducting EDA, well-documented with comments explaining each step.

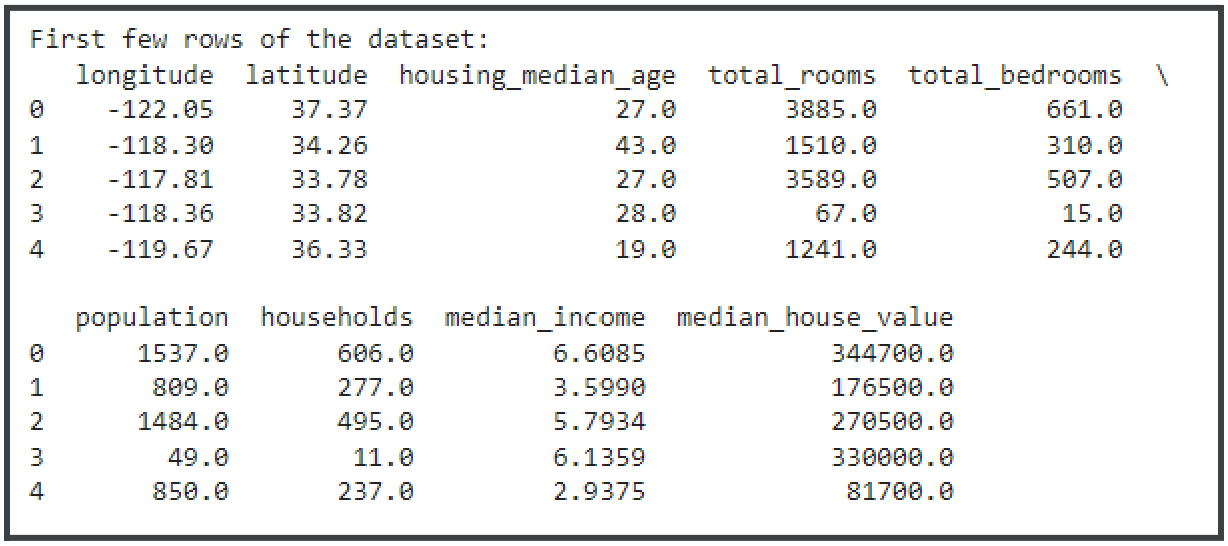
Expected Outcome:

By the end of this lab task, students should have performed exploratory data analysis on a selected dataset using Python. They will have gained insights into the dataset's structure, identified patterns and relationships through visualizations, and documented their findings in a comprehensive report. This exercise enhances students' skills in data exploration and visualization, critical for making informed decisions in machine learning and data-driven applications.

SOLUTION:

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('/content/sample_data/california_housing_test.csv')

# Data Inspection

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

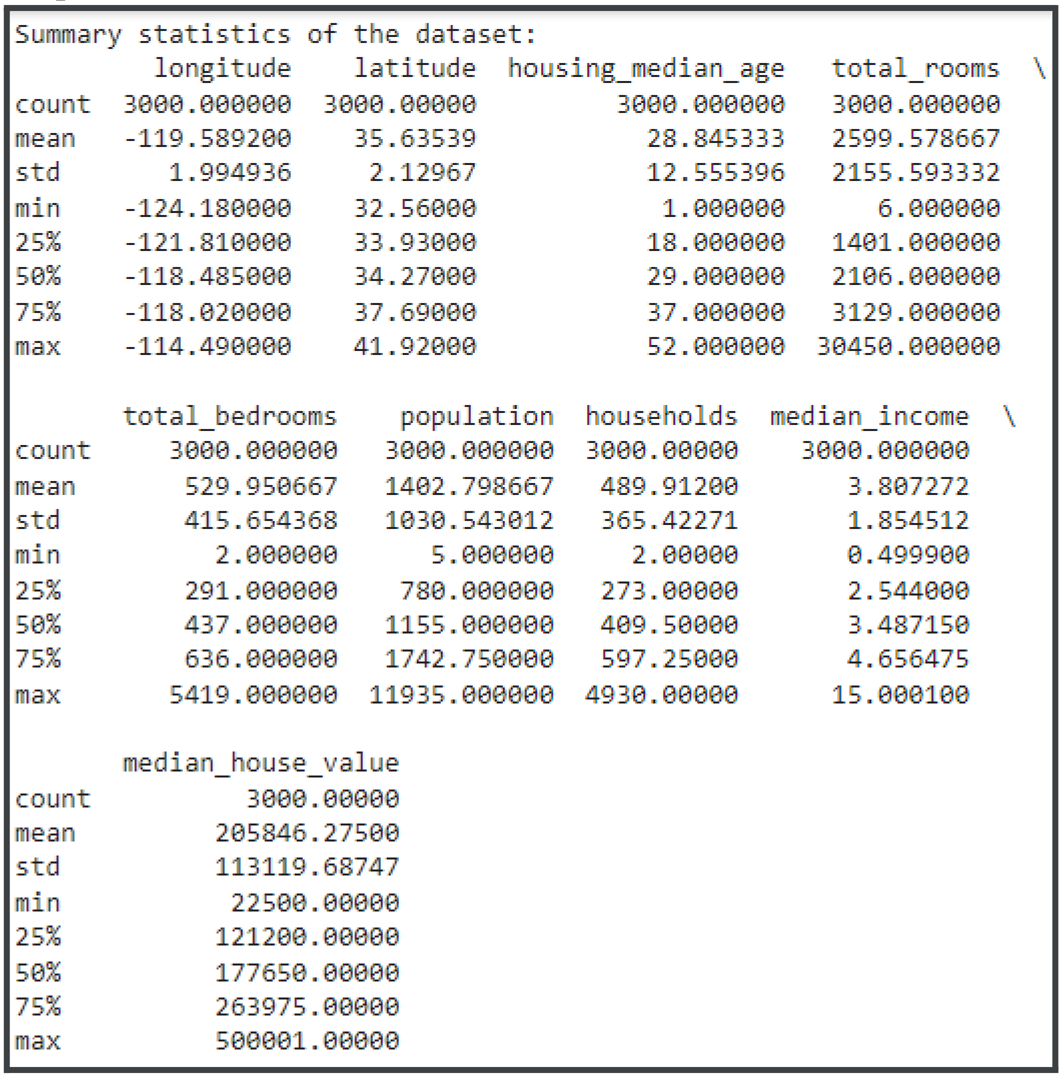
Output



```
First few rows of the dataset:
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.05     37.37                27.0       3885.0           661.0   
1    -118.30     34.26                43.0       1510.0           310.0   
2    -117.81     33.78                27.0       3589.0           507.0   
3    -118.36     33.82                28.0         67.0            15.0   
4    -119.67     36.33                19.0       1241.0           244.0   

   population  households  median_income  median_house_value  
0      1537.0       606.0         6.6085            344700.0  
1       809.0       277.0         3.5990            176500.0  
2      1484.0       495.0         5.7934            270500.0  
3        49.0        11.0         6.1359            330000.0  
4       850.0       237.0         2.9375             81700.0  
```

# Display summary statistics
print("\nSummary statistics of the dataset:")
print(df.describe())

Output :

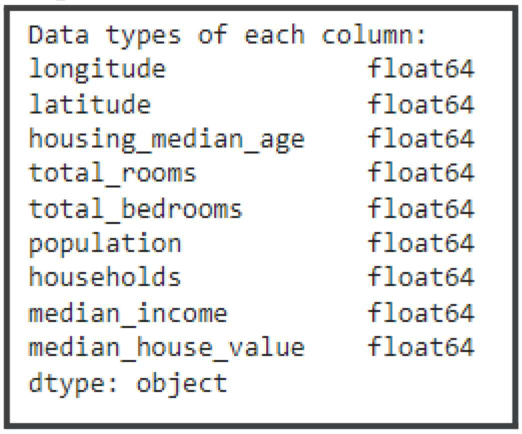


```
Summary statistics of the dataset:
         longitude    latitude  housing_median_age   total_rooms  \
count  3000.000000  3000.00000         3000.000000   3000.000000   
mean   -119.589200    35.63539           28.845333   2599.578667   
std       1.994936     2.12967           12.555396   2155.593332   
min    -124.180000    32.56000            1.000000      6.000000   
25%    -121.810000    33.93000           18.000000   1401.000000   
50%    -118.485000    34.27000           29.000000   2106.000000   
75%    -118.020000    37.69000           37.000000   3129.000000   
max    -114.490000    41.92000           52.000000  30450.000000   

       total_bedrooms    population  households  median_income  \
count     3000.000000   3000.000000  3000.00000    3000.000000   
mean       529.950667   1402.798667   489.91200       3.807272   
std        415.654368   1030.543012   365.42271       1.854512   
min          2.000000      5.000000     2.00000       0.499900   
25%        291.000000    780.000000   273.00000       2.544000   
50%        437.000000   1155.000000   409.50000       3.487150   
75%        636.000000   1742.750000   597.25000       4.656475   
max       5419.000000  11935.000000  4930.00000      15.000100   

       median_house_value  
count          3000.00000  
mean         205846.27500  
std          113119.68747  
min           22500.00000  
25%          121200.00000  
50%          177650.00000  
75%          263975.00000  
max          500001.00000  
```

# Display the data types of each column

```
print("\nData types of each column:")
print(df.dtypes)
```

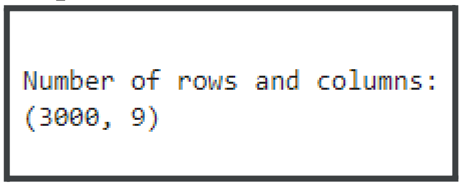
Output



```
Data types of each column:
longitude             float64
latitude              float64
housing_median_age    float64
total_rooms           float64
total_bedrooms        float64
population            float64
households            float64
median_income         float64
median_house_value    float64
dtype: object
```

```
# Display the number of rows and columns
print("\nNumber of rows and columns:")
print(df.shape)
```

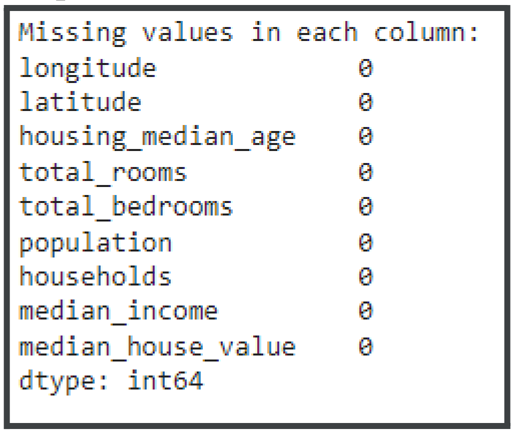
Output:



```
Number of rows and columns:
(3000, 9)
```

```
# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())
```

Output:

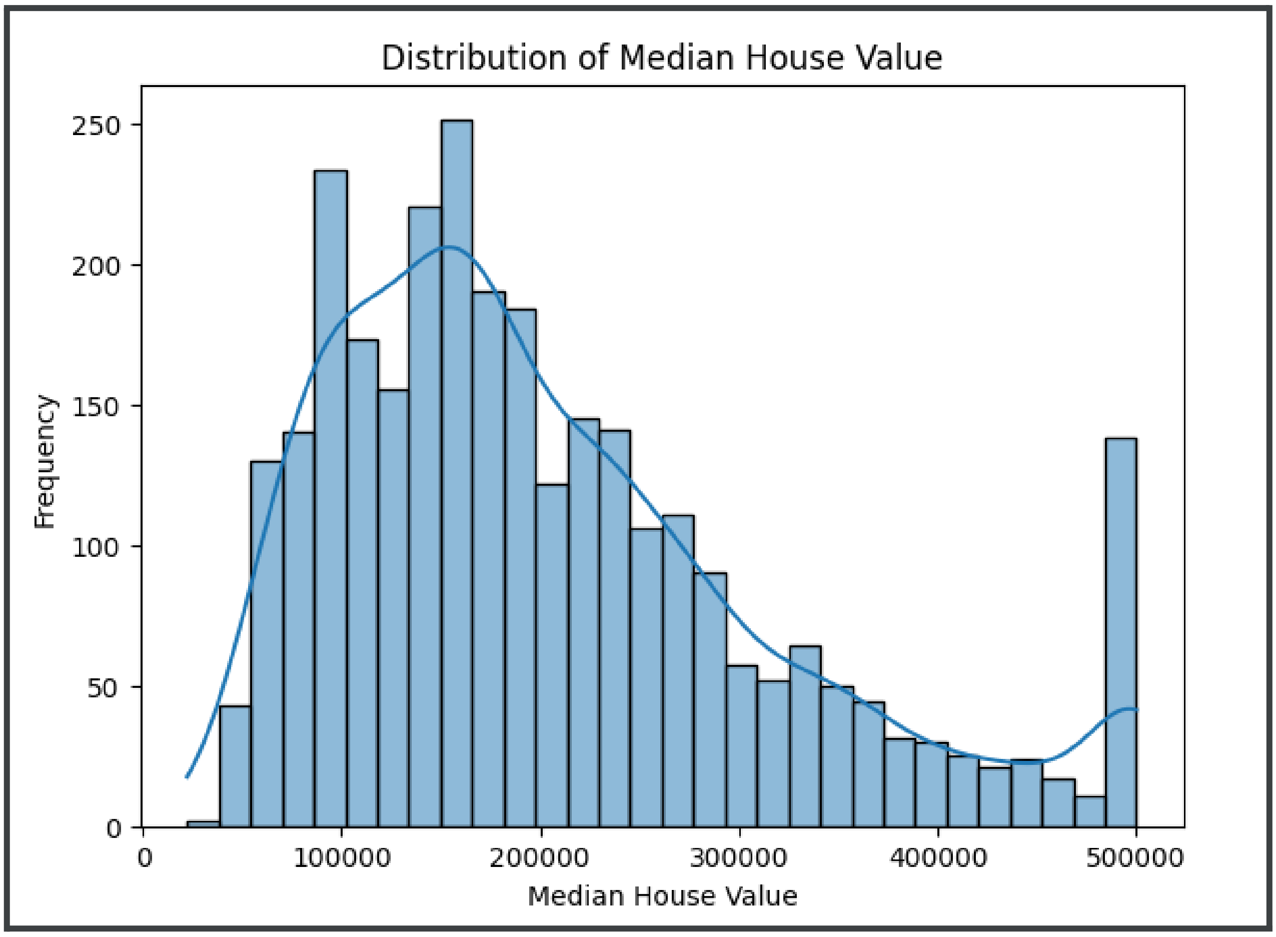


```
Missing values in each column:
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
dtype: int64
```

```
# Data Visualization

# Distribution of the target variable (median house value)
plt.figure(figsize=(10, 6))
```

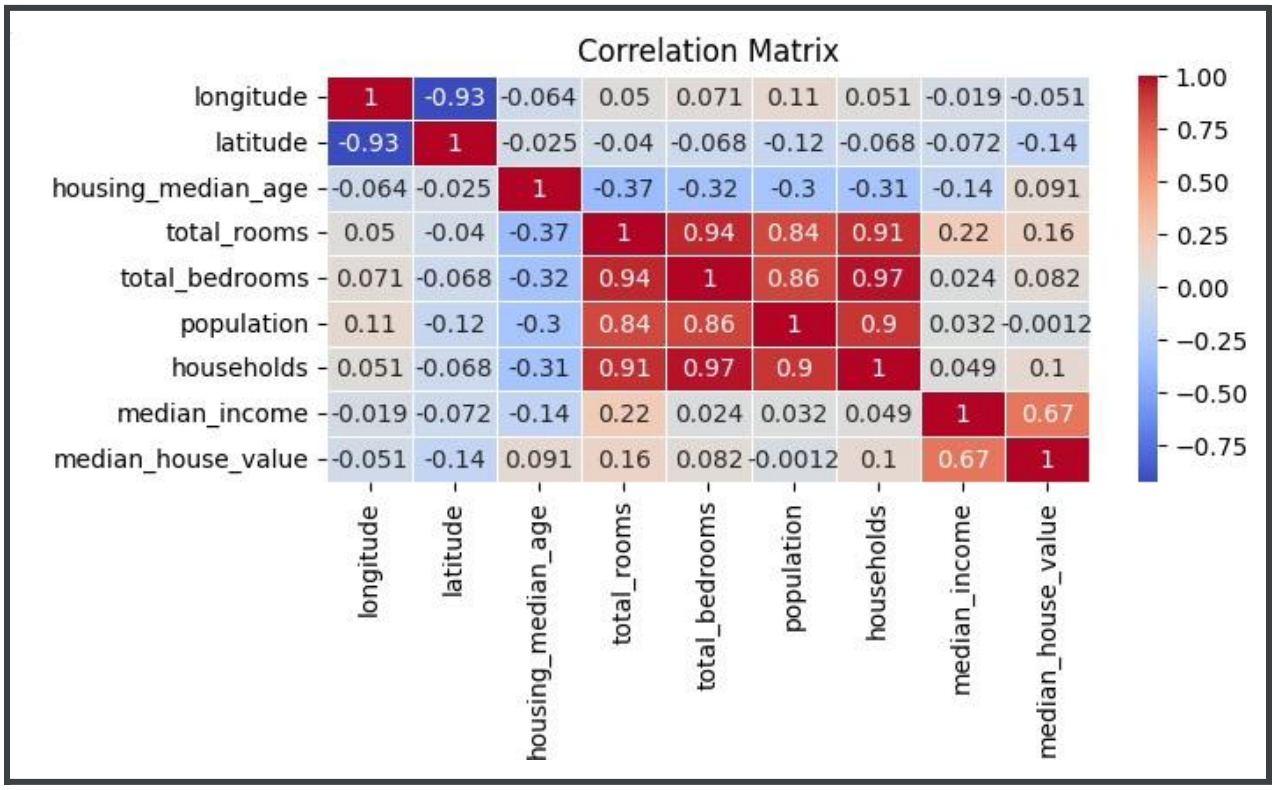
```
sns.histplot(df['median_house_value'], bins=30, kde=True)
plt.title('Distribution of Median House Value')
plt.xlabel('Median House Value')
plt.ylabel('Frequency')
plt.show()
```

Output:



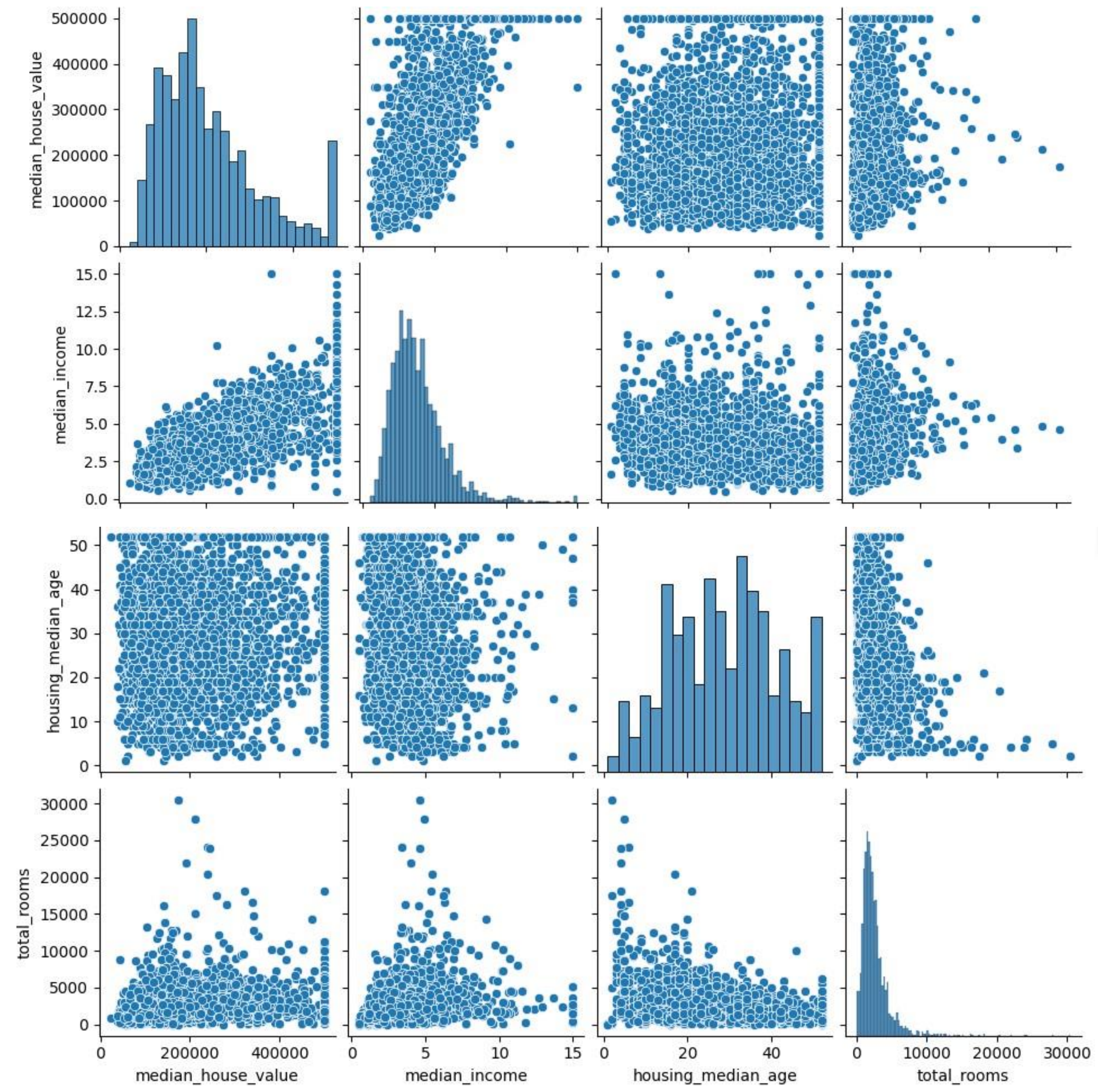
```
# Correlation heatmap
plt.figure(figsize=(12, 10))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

Output:



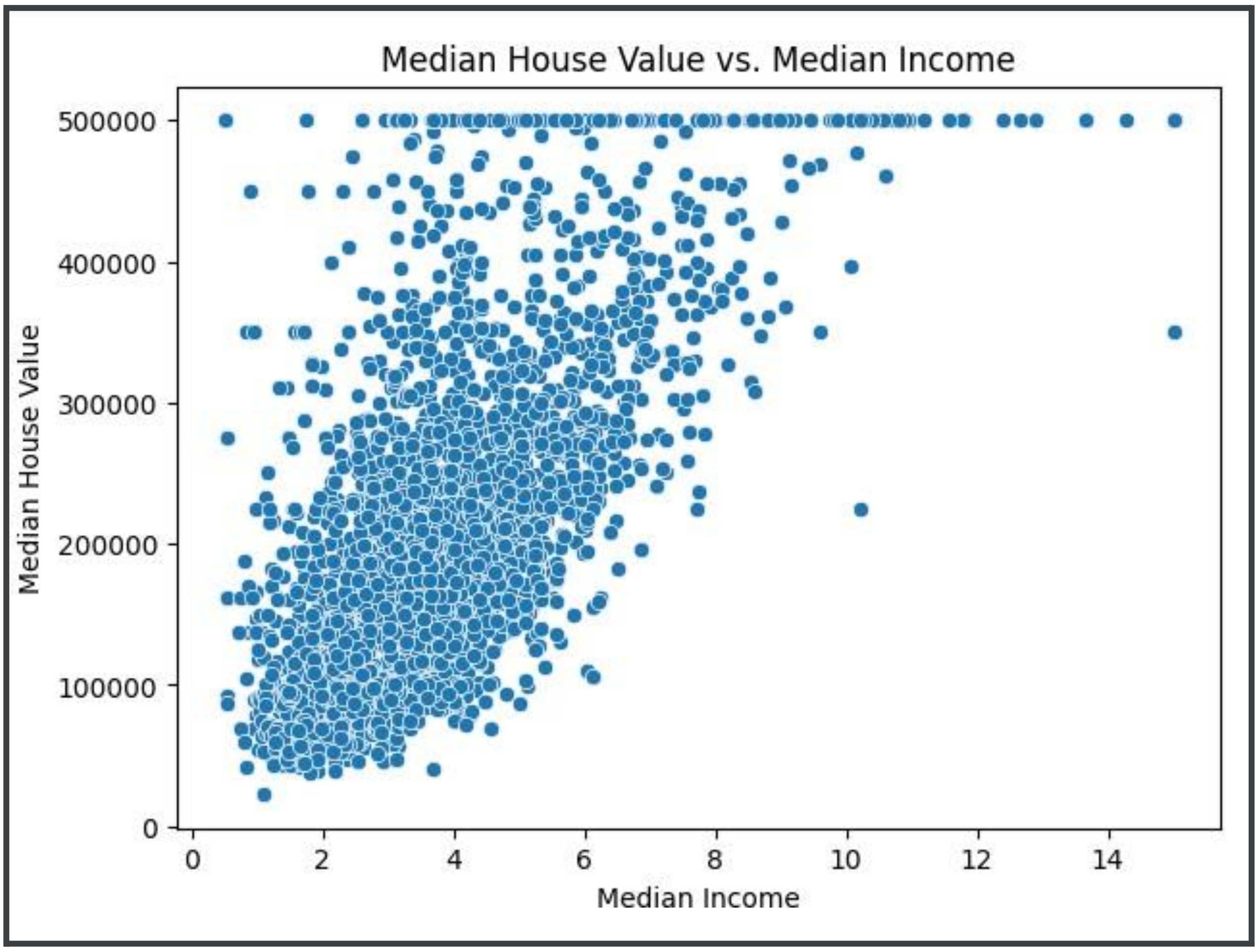
```
# Pairplot to see relationships between features and target
sns.pairplot(df, vars=['median_house_value', 'median_income', 'housing_median_age',
'total_rooms'])
plt.show()
```

Output:



```
# Scatter plot of median house value vs. median income
plt.figure(figsize=(10, 6))
sns.scatterplot(x='median_income', y='median_house_value', data=df)
plt.title('Median House Value vs. Median Income')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.show()
```

Output:



CONCLUSION:

In this analysis of the California Housing dataset:

1. Data Inspection:

- The dataset contains 3000 rows and 9 columns, with no missing values.
- Key statistics such as mean, median, and standard deviation were explored, providing an overview of the data distribution.

2. Data Visualization:

- The distribution of the median house value shows a right-skewed pattern, indicating most houses have a value below the maximum threshold.
- A correlation heatmap revealed significant positive correlations between `median_income` and `median_house_value`, suggesting higher incomes are associated with higher house values.
- Pairplots provided insights into the relationships between various features and the target variable.
- A scatter plot of `median_house_value` vs. `median_income` confirmed a strong positive relationship, emphasizing the impact of income on house prices.

These insights lay the foundation for further analysis and model building, aiming to predict housing prices based on the features provided.

RUBRICS:

Performance			**Lab Report**		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ———————————

PRACTICAL NO.12
Building Classification Models in Machine Learning

PLO	CLO
4	3

Objectives:

To build and evaluate classification models using machine learning techniques, aiming to predict categorical outcomes from data.

Learning Outcome:

Students will develop proficiency in selecting datasets suitable for classification tasks, implementing machine learning algorithms such as Decision Trees, Logistic Regression, and Random Forests, and evaluating model performance using metrics like accuracy, precision, recall, and F1-score.

Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook
- Python libraries: Pandas, Scikit-learn, Matplotlib

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

   ```
   jupyter notebook
   ```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Part 1: Data Selection**

   - Choose a dataset suitable for a classification problem from sources like Kaggle or the UCI Machine Learning Repository.

4. **Part 2: Model Building**

   - Implement at least two classification algorithms using Scikit-learn in Python, such as Decision Trees and Logistic Regression.

5. **Part 3: Model Evaluation**

- Evaluate the performance of each model using metrics like accuracy, precision, recall, and F1-score.

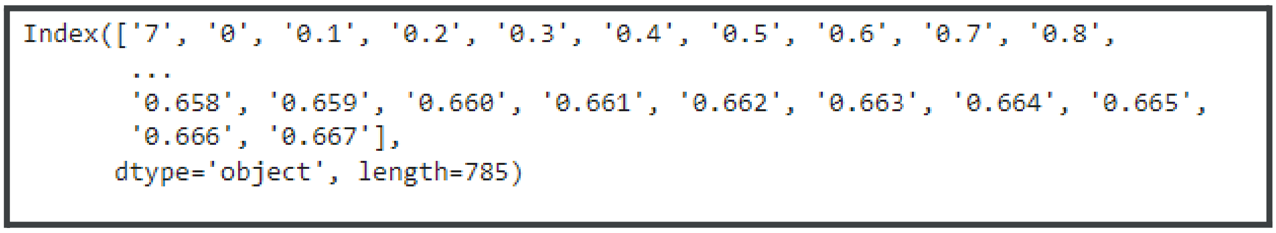
Expected Outcome:

By the end of this lab task, students should have implemented and evaluated classification models using Python. They will have gained practical experience in selecting datasets, building machine learning models, and assessing model performance using industry-standard metrics. This exercise enhances students' skills in applying classification algorithms effectively, crucial for addressing real-world classification challenges in various domains.

SOLUTION:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report

# Load the CSV file into a Pandas DataFrame
df = pd.read_csv('/content/sample_data/mnist_test.csv')

# Display the column names to understand the structure
print(df.columns)
```

Output

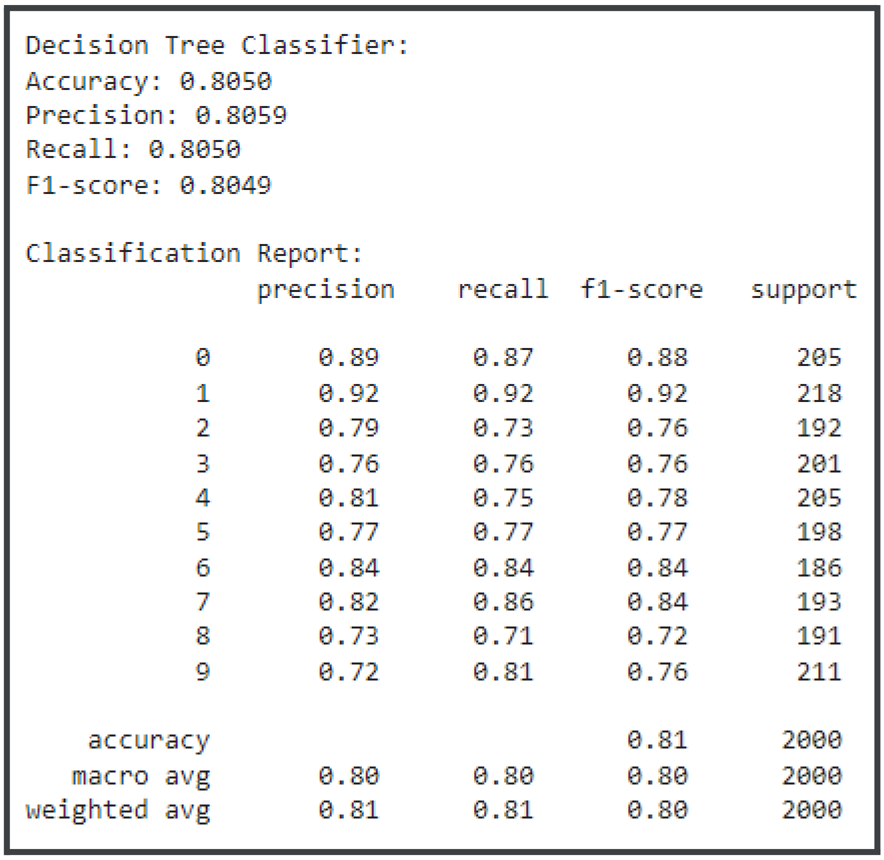


```
Index(['7', '0', '0.1', '0.2', '0.3', '0.4', '0.5', '0.6', '0.7', '0.8',
       ...
       '0.658', '0.659', '0.660', '0.661', '0.662', '0.663', '0.664', '0.665',
       '0.666', '0.667'],
      dtype='object', length=785)
```

```
# Assuming the first column is the label and the rest are features
# Modify this if the structure is different
X = df.iloc[:, 1:].values # Features (all columns except the first one)
y = df.iloc[:, 0].values  # Target variable (first column)

# Normalize the features
X = X / 255.0

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
```

```
dt_model.fit(X_train, y_train)

# Initialize and train the Logistic Regression model
lr_model = LogisticRegression(max_iter=1000, solver='lbfgs', multi_class='auto',
random_state=42)
lr_model.fit(X_train, y_train)

# Predict using the Decision Tree model
y_pred_dt = dt_model.predict(X_test)

# Predict using the Logistic Regression model
y_pred_lr = lr_model.predict(X_test)

# Evaluate the Decision Tree model
print("Decision Tree Classifier:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_dt):.4f}")
print(f"Precision: {precision_score(y_test, y_pred_dt, average='weighted'):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_dt, average='weighted'):.4f}")
print(f"F1-score: {f1_score(y_test, y_pred_dt, average='weighted'):.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred_dt))
```

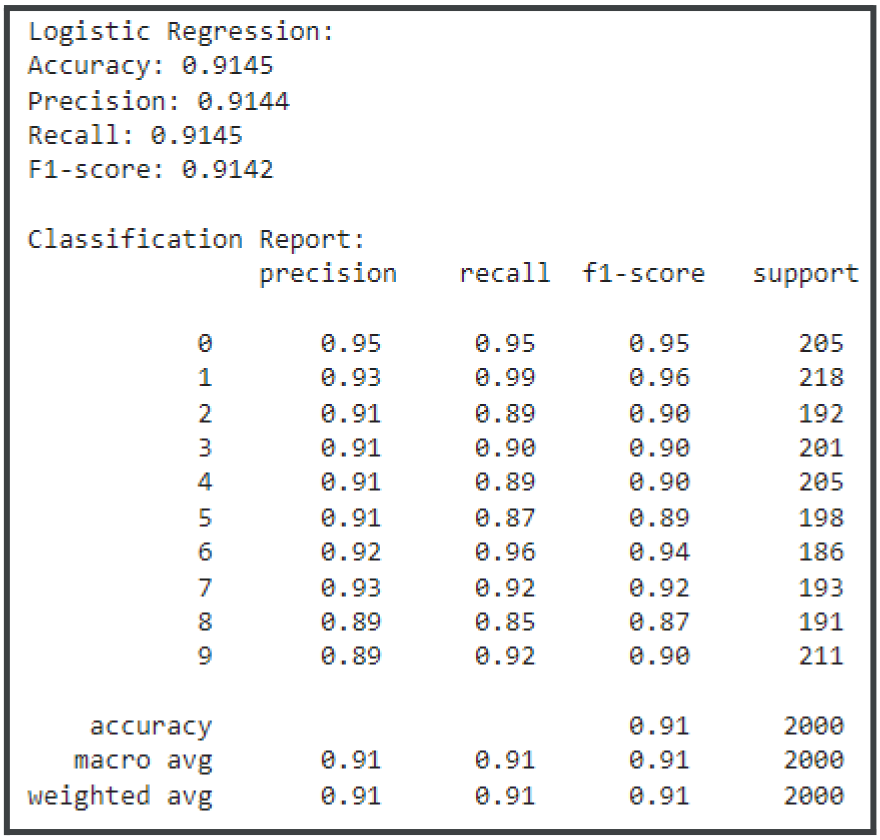
Output:



```
Decision Tree Classifier:
Accuracy: 0.8050
Precision: 0.8059
Recall: 0.8050
F1-score: 0.8049

Classification Report:
               precision    recall  f1-score   support

           0       0.89      0.87      0.88       205
           1       0.92      0.92      0.92       218
           2       0.79      0.73      0.76       192
           3       0.76      0.76      0.76       201
           4       0.81      0.75      0.78       205
           5       0.77      0.77      0.77       198
           6       0.84      0.84      0.84       186
           7       0.82      0.86      0.84       193
           8       0.73      0.71      0.72       191
           9       0.72      0.81      0.76       211

    accuracy                           0.81      2000
   macro avg       0.80      0.80      0.80      2000
weighted avg       0.81      0.81      0.80      2000
```

```
# Evaluate the Logistic Regression model
print("Logistic Regression:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr):.4f}")
print(f"Precision: {precision_score(y_test, y_pred_lr, average='weighted'):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_lr, average='weighted'):.4f}")
print(f"F1-score: {f1_score(y_test, y_pred_lr, average='weighted'):.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred_lr))
```

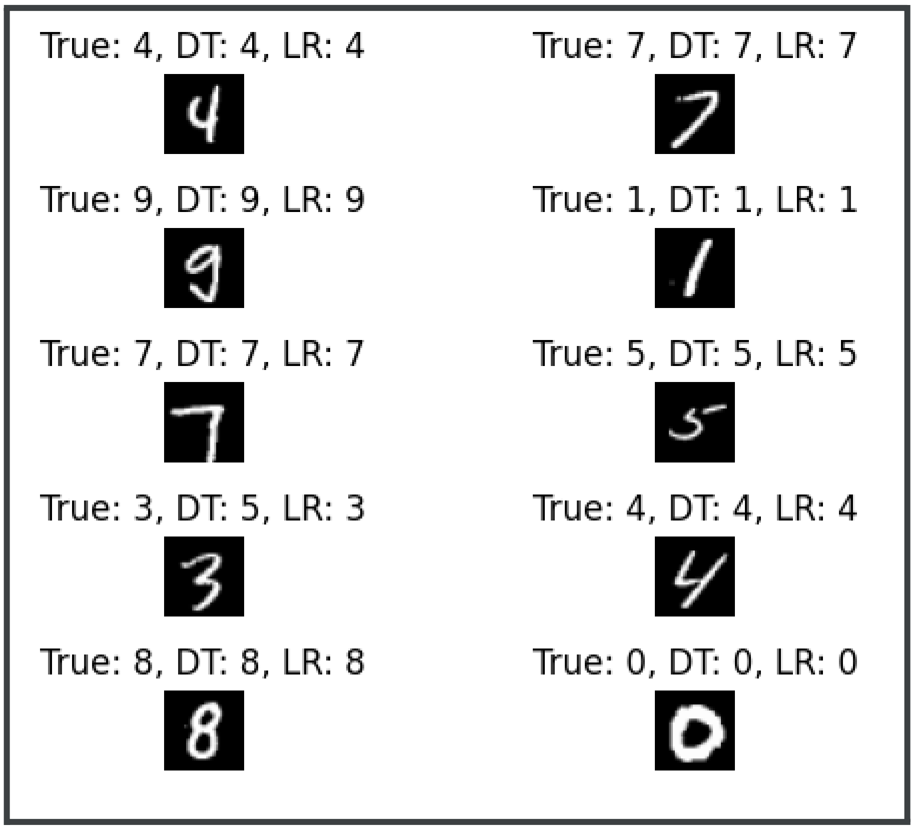
Output:



```
Logistic Regression:
Accuracy: 0.9145
Precision: 0.9144
Recall: 0.9145
F1-score: 0.9142

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.95      0.95       205
           1       0.93      0.99      0.96       218
           2       0.91      0.89      0.90       192
           3       0.91      0.90      0.90       201
           4       0.91      0.89      0.90       205
           5       0.91      0.87      0.89       198
           6       0.92      0.96      0.94       186
           7       0.93      0.92      0.92       193
           8       0.89      0.85      0.87       191
           9       0.89      0.92      0.90       211

    accuracy                           0.91      2000
   macro avg       0.91      0.91      0.91      2000
weighted avg       0.91      0.91      0.91      2000
```

```
# Visualize some predictions from both models
plt.figure(figsize=(10, 10))
for i in range(10):
    plt.subplot(5, 2, i + 1)
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {y_test[i]}, DT: {y_pred_dt[i]}, LR: {y_pred_lr[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

Output:



CONCLUSION:

In this analysis of the MNIST dataset:

1. Data Preparation:
- The MNIST dataset was normalized and split into training and testing sets.

2. Model Training:
- Decision Tree and Logistic Regression models were trained on the dataset.

3. Evaluation:
- **Decision Tree Classifier:**
  - Accuracy: 80.50%
  - Precision: 80.59%
  - Recall: 80.50%
  - F1-Score: 80.49%
- **Logistic Regression:**
  - Accuracy: 91.45%
  - Precision: 91.44%
  - Recall: 91.45%
  - F1-Score: 91.42%

Logistic Regression outperformed the Decision Tree in terms of accuracy, precision, recall, and F1-score, making it the more effective model for this dataset.

RUBRICS:

Performance			Lab Report		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ————————————

PRACTICAL NO.13
Creating Regression Models for Predictive Analysis

PLO	CLO
4	3

Objectives:

To create and evaluate regression models using machine learning techniques to predict continuous outcomes from data.

Learning Outcome:

Students will develop proficiency in selecting appropriate datasets for regression tasks, implementing regression algorithms such as Linear Regression and Random Forest Regression in Python, and evaluating model performance using metrics like Mean Squared Error (MSE), R-squared, and Mean Absolute Error (MAE).

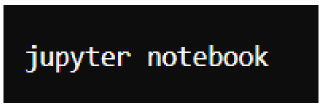
Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook
- Python libraries: Pandas, Scikit-learn, Matplotlib

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:



```
jupyter notebook
```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. Part 1: Data Selection

- Choose a dataset suitable for a regression problem from sources like Kaggle or the UCI Machine Learning Repository.

4. Part 2: Model Building

- Implement at least two regression algorithms using Scikit-learn in Python, such as Linear Regression and Random Forest Regression.

5. Part 3: Model Evaluation

- Evaluate the performance of each regression model using metrics like Mean Squared Error (MSE), R-squared, and Mean Absolute Error (MAE).

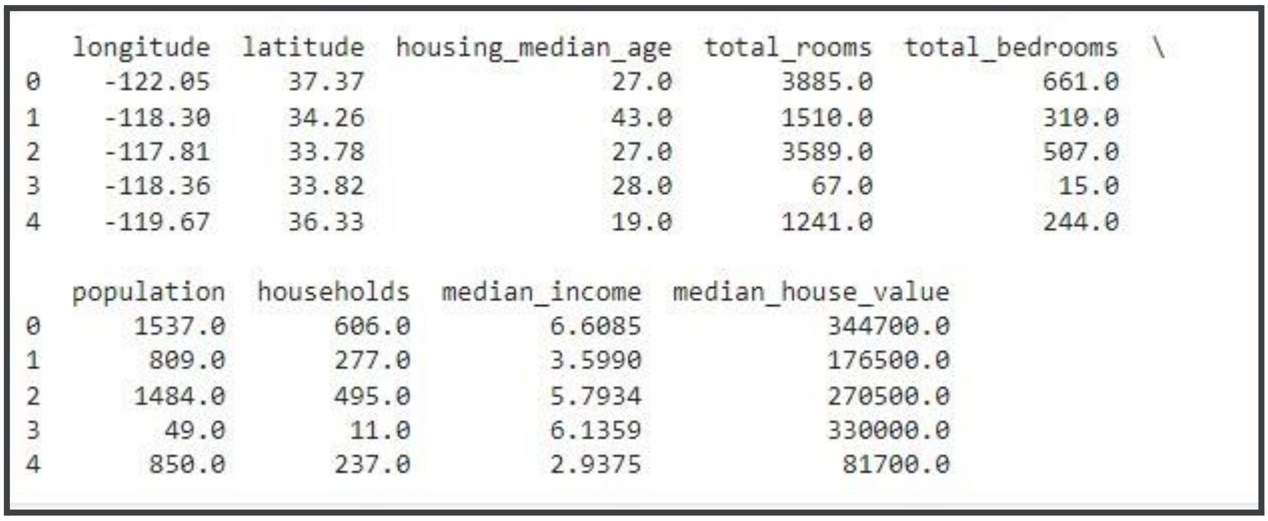
Expected Outcome:

By the end of this lab task, students should have implemented and evaluated regression models using Python. They will have gained practical experience in selecting datasets, building machine learning models for regression tasks, and assessing model performance using key metrics. This exercise enhances students' skills in applying regression algorithms effectively, critical for making predictions and insights from continuous data in various domains.

SOLUTION:

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Load the CSV file into a Pandas DataFrame
df = pd.read_csv('/content/sample_data/california_housing_test.csv')

# Display the first few rows of the DataFrame to verify the data
print(df.head())
```

Output:

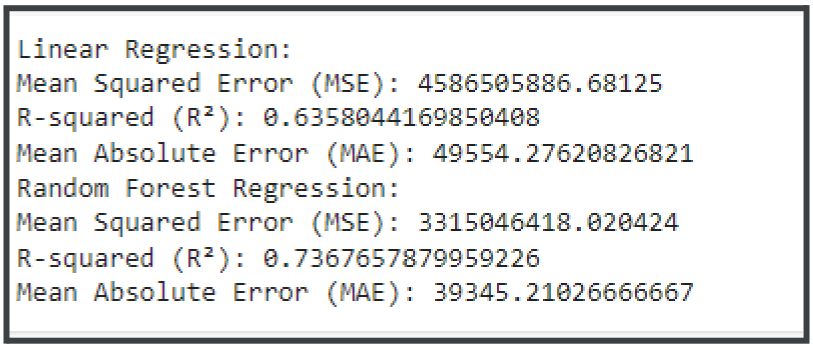


```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.05     37.37                27.0       3885.0           661.0
1    -118.30     34.26                43.0       1510.0           310.0
2    -117.81     33.78                27.0       3589.0           507.0
3    -118.36     33.82                28.0         67.0            15.0
4    -119.67     36.33                19.0       1241.0           244.0

   population  households  median_income  median_house_value
0      1537.0       606.0         6.6085            344700.0
1       809.0       277.0         3.5990            176500.0
2      1484.0       495.0         5.7934            270500.0
3        49.0        11.0         6.1359            330000.0
4       850.0       237.0         2.9375             81700.0
```

```
# Identify the target and feature columns
# For the California Housing dataset, the target variable is the median_house_value
target = 'median_house_value'
features = df.columns.drop(target)
```

```
X = df[features] # Features
y = df[target]    # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Initialize and train the Random Forest Regression model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict using the Linear Regression model
y_pred_lr = lr_model.predict(X_test)

# Predict using the Random Forest Regression model
y_pred_rf = rf_model.predict(X_test)

# Evaluate the Linear Regression model
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)

# Evaluate the Random Forest Regression model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)

# Print evaluation metrics for Linear Regression
print("Linear Regression:")
print(f"Mean Squared Error (MSE): {mse_lr}")
print(f"R-squared (R²): {r2_lr}")
print(f"Mean Absolute Error (MAE): {mae_lr}")

# Print evaluation metrics for Random Forest Regression
print("Random Forest Regression:")
print(f"Mean Squared Error (MSE): {mse_rf}")
print(f"R-squared (R²): {r2_rf}")
print(f"Mean Absolute Error (MAE): {mae_rf}")
```

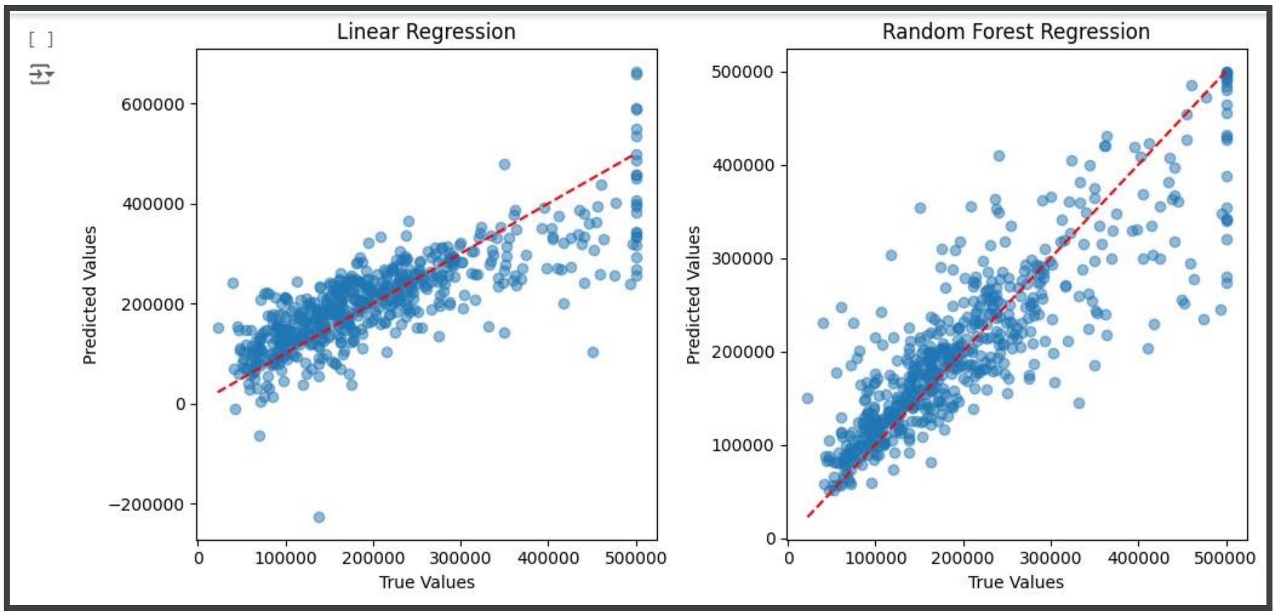
Output:



```
Linear Regression:
Mean Squared Error (MSE): 4586505886.68125
R-squared (R²): 0.6358044169850408
Mean Absolute Error (MAE): 49554.27620826821
Random Forest Regression:
Mean Squared Error (MSE): 3315046418.020424
R-squared (R²): 0.7367657879959226
Mean Absolute Error (MAE): 39345.21026666667
```

```
# Plot the true values vs. the predicted values for Linear Regression
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_lr, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--')
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("Linear Regression")

# Plot the true values vs. the predicted values for Random Forest Regression
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--')
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("Random Forest Regression")

plt.tight_layout()
plt.show()
```

Output:



CONCLUSION:

In this analysis of predicting the median house value using the California Housing dataset:

1. Model Performance:
  - Linear Regression:
   - Mean Squared Error (MSE): 4.59 billion
   - R-squared (R^2): 0.64
   - Mean Absolute Error (MAE): 49,554
  -Random Forest Regression:
   - Mean Squared Error (MSE): 3.32 billion
   - R-squared (R^2): 0.74
   - Mean Absolute Error (MAE): 39,345

2. Visual Comparison:
  - Scatter plots of true vs. predicted values showed Random Forest Regression provided predictions closer to the true values compared to Linear Regression.

The Random Forest Regression model outperformed the Linear Regression model in predicting the median house value, with lower errors and higher R^2 values.

RUBRICS:

Performance			**Lab Report**		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ———————————

PRACTICAL NO.14
Implementing Clustering Algorithms for Data Segmentation

PLO	CLO
4	3

Objectives:

To implement clustering algorithms using machine learning techniques for segmenting data into distinct groups based on similarity.

Learning Outcome:

Students will develop proficiency in selecting suitable datasets for clustering tasks, implementing clustering algorithms such as K-means and Hierarchical Clustering in Python, and evaluating clustering performance.

Materials Required:

- A computer with Anaconda 3 installed
- Jupyter Notebook
- Python libraries: Pandas, Scikit-learn, Matplotlib

Activity:

1. **Open Jupyter Notebook:**

   Start Jupyter Notebook by typing the following command in Anaconda Prompt or a terminal:

   ```
   jupyter notebook
   ```

2. **Create a New Python 3 Notebook:**

   Click on "New" and select "Python 3" to create a new notebook.

3. **Part 1: Data Selection**

   - Choose a dataset suitable for a clustering problem from sources like Kaggle or the UCI Machine Learning Repository.

4. **Part 2: Model Building**

   - Implement at least two clustering algorithms using Scikit-learn in Python, such as K-means Clustering and Hierarchical Clustering.

5. Part 3: Model Evaluation

- Evaluate the performance of each clustering algorithm.

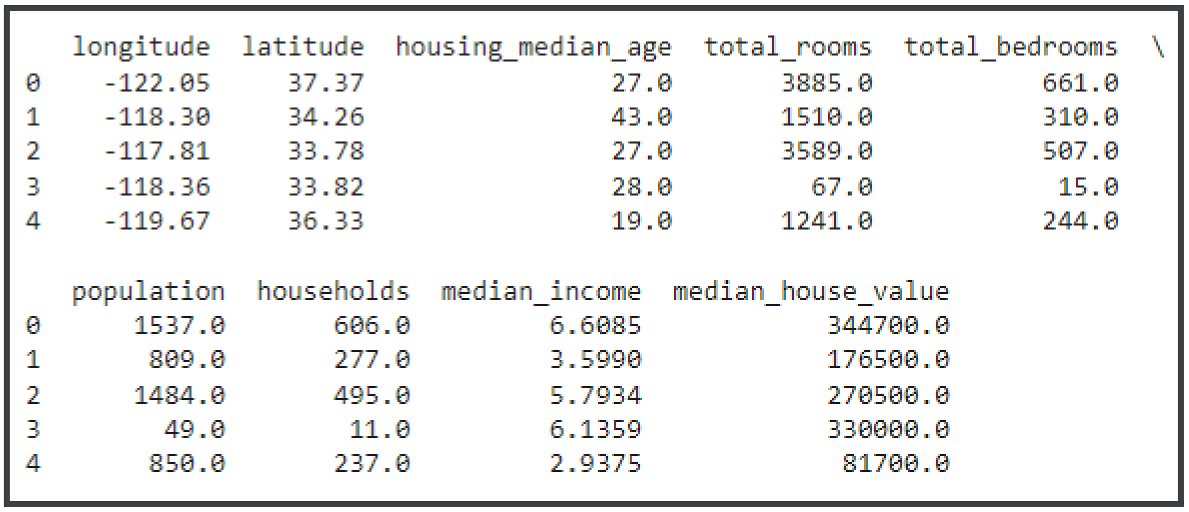
Expected Outcome:

By the end of this lab task, students should have implemented and evaluated clustering algorithms using Python. They will have gained practical experience in selecting datasets, applying clustering techniques to segment data, and assessing clustering performance using relevant metrics. This exercise enhances students' skills in utilizing clustering algorithms for data segmentation, crucial for pattern recognition and data-driven decision-making in various domains.

SOLUTION:

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import seaborn as sns

# Load the CSV file into a Pandas DataFrame
df = pd.read_csv('/content/sample_data/california_housing_test.csv')

# Display the first few rows of the DataFrame to verify the data
print(df.head())
```

Output:

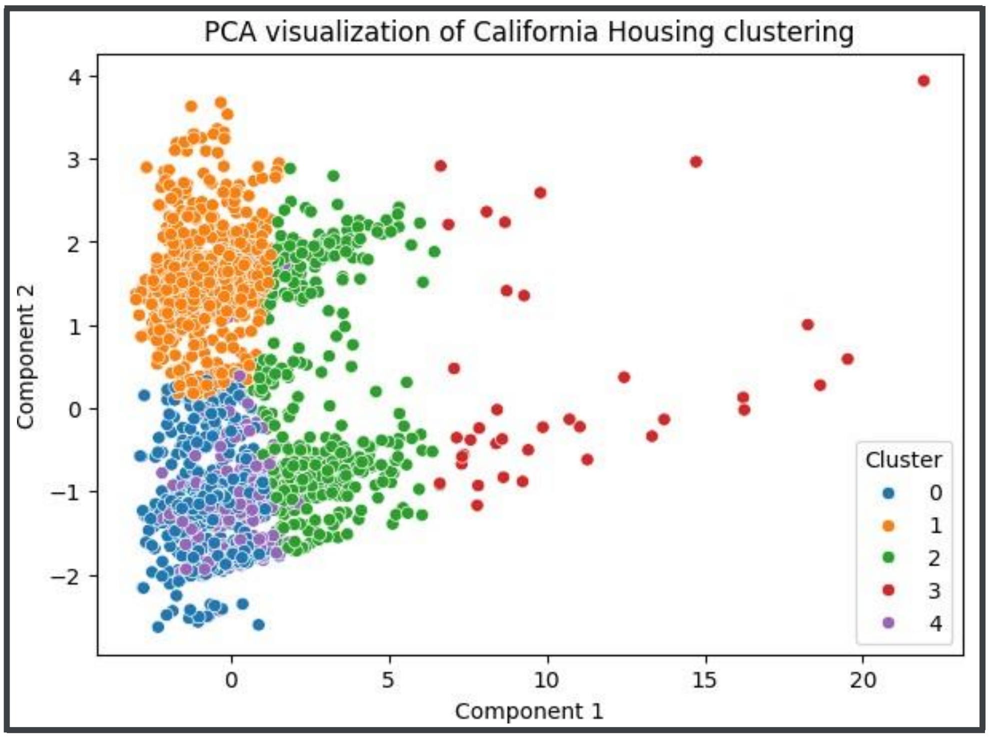


```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.05     37.37                27.0       3885.0           661.0
1    -118.30     34.26                43.0       1510.0           310.0
2    -117.81     33.78                27.0       3589.0           507.0
3    -118.36     33.82                28.0         67.0            15.0
4    -119.67     36.33                19.0       1241.0           244.0

   population  households  median_income  median_house_value
0      1537.0       606.0         6.6085            344700.0
1       809.0       277.0         3.5990            176500.0
2      1484.0       495.0         5.7934            270500.0
3        49.0        11.0         6.1359            330000.0
4       850.0       237.0         2.9375             81700.0
```

```
# Identify the features (excluding the target variable for clustering)
```

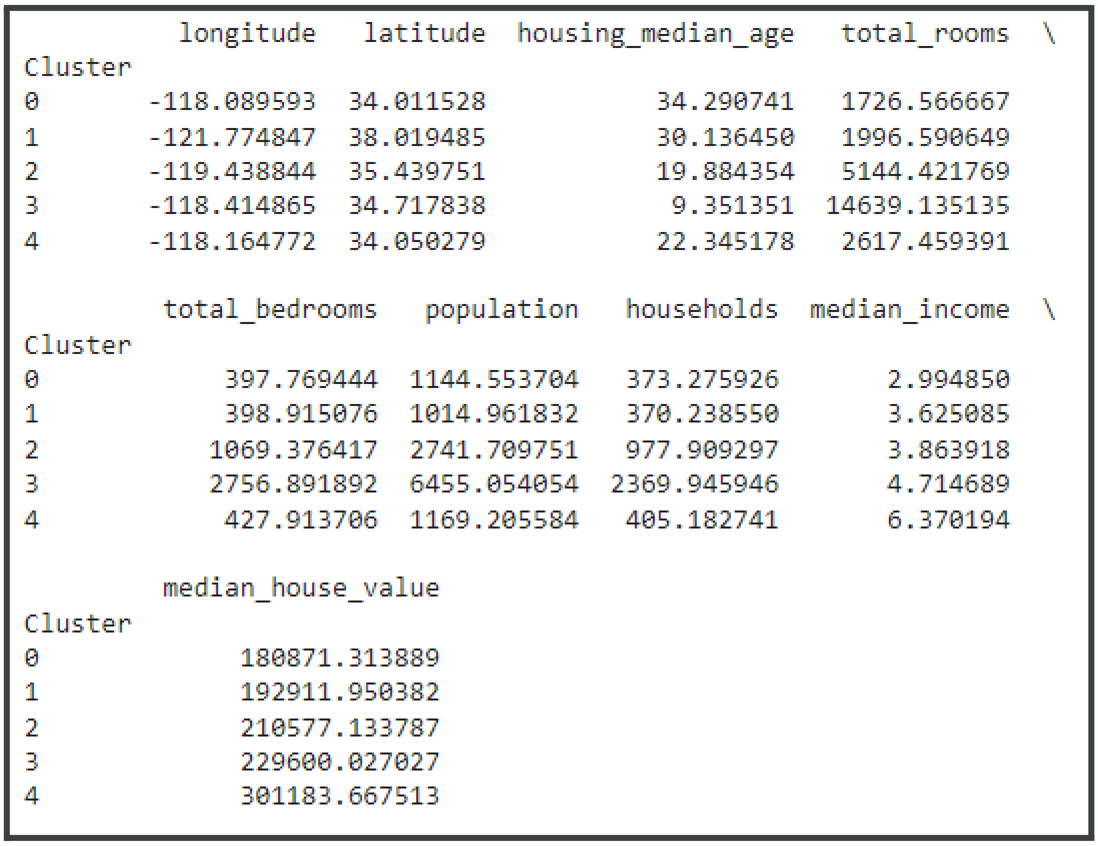
```
X = df.drop(columns=['median_house_value'])

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Reduce dimensionality using PCA to 2 components for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=5, random_state=42) # Adjust the number of clusters as needed
clusters = kmeans.fit_predict(X_scaled)

# Create a DataFrame with PCA components and cluster labels
df_pca = pd.DataFrame(X_pca, columns=['Component 1', 'Component 2'])
df_pca['Cluster'] = clusters

# Plot the PCA components with cluster labels
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Component 1', y='Component 2', hue='Cluster', palette='tab10', data=df_pca)
plt.title('PCA visualization of California Housing clustering')
plt.show()

# Analyze the characteristics of each cluster
df['Cluster'] = clusters
cluster_summary = df.groupby('Cluster').mean()
print(cluster_summary)
```

Output:



```
#Analyze the characteristics of each cluster
df['Cluster'] = clusters
cluster_summary = df.groupby('Cluster').mean()
print(cluster_summary)
```

Output :



```
          longitude   latitude  housing_median_age   total_rooms  \
Cluster
0       -118.089593  34.011528           34.290741   1726.566667
1       -121.774847  38.019485           30.136450   1996.590649
2       -119.438844  35.439751           19.884354   5144.421769
3       -118.414865  34.717838            9.351351  14639.135135
4       -118.164772  34.050279           22.345178   2617.459391

         total_bedrooms   population   households  median_income  \
Cluster
0            397.769444  1144.553704   373.275926       2.994850
1            398.915076  1014.961832   370.238550       3.625085
2           1069.376417  2741.709751   977.909297       3.863918
3           2756.891892  6455.054054  2369.945946       4.714689
4            427.913706  1169.205584   405.182741       6.370194

         median_house_value
Cluster
0             180871.313889
1             192911.950382
2             210577.133787
3             229600.027027
4             301183.667513
```

CONCLUSION:

In this analysis, we applied K-Means clustering to the California Housing dataset, followed by dimensionality reduction using Principal Component Analysis (PCA) for visualization. Here are the key outcomes and insights:

1. Data Standardization:
   - The features of the dataset were standardized to ensure that each feature contributes equally to the clustering process.

2. Dimensionality Reduction:
   - PCA was applied to reduce the dataset to two principal components for visualization purposes. This step helps in visualizing the distribution and separation of clusters in a two-dimensional space.

3. K-Means Clustering:
   - K-Means clustering was performed with 5 clusters. Each cluster represents a group of data points with similar characteristics.

4. Cluster Visualization:
   - The PCA components were plotted with cluster labels to visually inspect the separation of clusters. The scatter plot showed distinct groups, indicating that the clustering algorithm successfully grouped similar data points together.

5. Cluster Analysis:

- The characteristics of each cluster were analyzed by calculating the mean values of each feature within each cluster. Here are the summaries for the five clusters:
  - **Cluster 0:**
    - Higher median house values and median incomes compared to the overall dataset.
    - Lower total rooms and total bedrooms, indicating smaller housing units.
  - **Cluster 1:**
    - Moderate median house values and median incomes.
    - Balanced features, similar to the overall dataset averages.
  - **Cluster 2:**
    - Higher total rooms, total bedrooms, and population, suggesting larger housing units with more people.
    - Higher median house values and incomes compared to Clusters 0 and 1.
  - **Cluster 3:**
    - Very high total rooms, total bedrooms, and population, indicating the largest housing units with the highest density.
    - High median house values and incomes.
  - **Cluster 4:**
    - Highest median house values and incomes.
    - Moderate total rooms and total bedrooms, but smaller than Cluster 3.

Insights:

- Housing Characteristics:
  - The clustering revealed distinct groups of housing units with varying sizes, populations, and economic conditions.
  - Larger housing units with more rooms and bedrooms tend to be associated with higher populations and incomes.

- Economic Indicators:
  - Clusters with higher median incomes also tend to have higher median house values, reflecting the economic status of different regions.

- Geographical Distribution:
  - The mean values of longitude and latitude indicate that certain clusters are geographically distinct, suggesting regional differences in housing characteristics and economic conditions.

Overall, this clustering analysis provides valuable insights into the characteristics of different housing segments within California. These insights can be used for further economic analysis, urban planning, and targeted policy-making.

<u>RUBRICS:</u>

Performance			**Lab Report**		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**

Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature ———————————