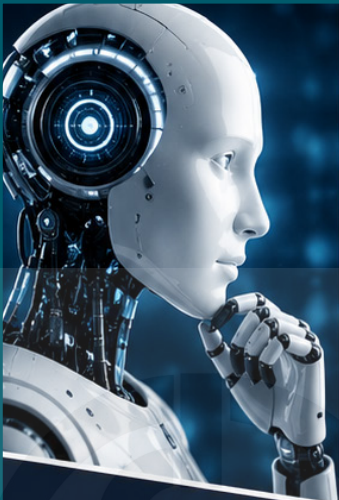
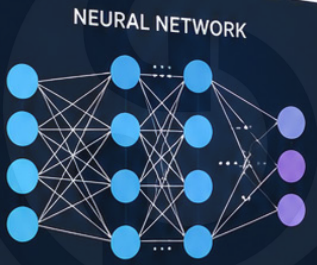




NEURAL NETWORK



INPUT LAYER HIDDEN LAYERS OUTPUT LAYER

DATA ANALYTICS



ARTIFICIAL INTELLIGENCE — AND — MACHINE LEARNING LAB



THINK



LEARN



INNOVATE

Building Intelligent Systems
for a Smarter Tomorrow





```
import numpy as np
from sklearn.model_selection
import train_test_split
from sklearn.ensemble
import RandomForestClassifier

X, y = load_data()
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)

model = RandomForestClassifier(
n_estimators=100, random_state=42)
model.fit(X_train, y_train)

pred = model.predict(X_test)

accuracy = np.mean(pred == y_test)

print(f"Accuracy: {accuracy:.2f}")
```



MACHINE LEARNING

-  SUPERVISED LEARNING
-  UNSUPERVISED LEARNING
-  REINFORCEMENT LEARNING
-  DEEP LEARNING



MCSL-069
Artificial Intelligence
and
Machine Learning Lab

Artificial Intelligence and Machine Learning Lab

Lab Sessions on Artificial Intelligence

Lab Sessions on Machine Learning

PROGRAMME DESIGN COMMITTEE

Prof. D. K. Lobiyal, JNU, New Delhi
Prof. Dinesh Kumar Vishwakarma, DTU, New Delhi

Prof. Sandeep Singh Rawat, SOCIS, IGNOU
Prof. P.V. Suresh, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Prof. Divakar Yadav, SOCIS, IGNOU
Prof. Akshay Kumar, SOCIS, IGNOU
Prof. M.P. Mishra, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assoc Professor, SOCIS

COURSE DESIGN COMMITTEE

Prof. D. K. Lobiyal, JNU, New Delhi
Prof. Dinesh Kumar Vishwakarma, DTU, New Delhi
Prof. T. V. Vijay Kumar, JNU, New Delhi
Prof. D. P. Vidyarthi, JNU, New Delhi
Prof. Vivek Kumar Singh, University of Delhi, Delhi
Prof. A. K. Mohapatra, IGDTUW, New Delhi
Mr. Kapil Pandey, HCL Technologies Limited, Delhi – NCR
Prof. Manish Trivedi, SOS (Statistics), IGNOU
Dr. Rajesh Kaliraman, Assistant Professor, SOS (Statistics), IGNOU
Dr. Prabhat Kumar Sangal, Assistant Professor, SOS (Statistics), IGNOU

Prof. Sandeep Singh Rawat, SOCIS, IGNOU
Prof. P.V. Suresh, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Prof. Divakar Yadav, SOCIS, IGNOU
Prof. Akshay Kumar, Professor, SOCIS IGNOU
Prof. M.P. Mishra, SOCIS IGNOU
Dr. Sudhansh Sharma, Assoc. Prof., SOCIS

SOCIS FACULTY

Prof. Sandeep Singh Rawat, Director, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Prof. Akshay Kumar, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assoc. Prof., SOCIS, IGNOU

Prof. P.V. Suresh, IGNOU, SOCIS, IGNOU
Prof. Divakar Yadav, SOCIS, IGNOU
Prof. M.P. Mishra, SOCIS, IGNOU

CONTENT PREPARATION TEAM

Prof. Adarsh Kumar,
DoCSE, UPES, Dehradun (Writer)

Prof. Santoshi SenGupta,
Graphics Era hill University Dehradun (Language Editor)

Dr. Arun Kumar Yadav,
DoCSE, NIT Hamirpur (Content Editor)

COURSE COORDINATOR

Prof. Divakar Yadav, SOCIS, IGNOU

PRINT PRODUCTION

Sh. Sanjay Aggarwal
Assistant Registrar, MPDD, IGNOU, New Delhi

,2025

©Indira Gandhi National Open University, 2025

ISBN – xx-xxxx-xxx-x

All rights reserved. No part of this work may be reproduced in any form, by mimeography or any other means, without permission in writing from the Indira Gandhi National Open University.

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi 110068.

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by MPDD, IGNOU.

COURSE INTRODUCTION

This course introduces you to practically work with the two most powerful concepts: Artificial Intelligence (AI) and Machine Learning (ML). After completing this course, you will be able to perform programming for various techniques of AI and ML, you learned MCS-224 (Artificial Intelligence and Machine Learning) course of this programme.

The pre-requisites of the course include the understanding of Python programming, you may refer to the concepts of Python programming given in the course MCS-201 (Programming in C & Python) and also try to attempt the lab exercises given in the section-2 of MCSL-205 (C and Python Lab).

In order to have better understanding you are advised to firstly get the conceptual clarity of various algorithms presented in the MCS-224 (Artificial Intelligence and Machine Learning) course of this programme, and then implement your understanding in Python programming to solve the problems given in the various sessions of this course.



ignou
THE PEOPLE'S
UNIVERSITY

BLOCK INTRODUCTION

This is the lab course, wherein you will have the hands-on exercise for programming various algorithms of Artificial Intelligence (AI) and Machine Learning (ML) in Python. You have studied the concepts of both Artificial Intelligence (AI) and Machine Learning (ML) in the course (MCS-224 i.e. Artificial Intelligence and Machine Learning).

In lab sessions on AI, you will implement the concepts of AI learned in MCS-224, by using Python programming, whereas in the lab sessions on Machine Learning, you will implement the concepts of Machine Learning, learned in MCS-224, by using python programming. You may use Google Colabs or ANACONDA or any other environment for Python programming.

A list of programming problems for both Artificial Intelligence (AI) and Machine Learning (ML) is provided, you need to attempt the problems given in respective sessions.

Please go through the general guidelines and the program documentation guidelines carefully.

This block consists of Lab sessions on AI and Machine learning and they are organized as follows:

- **Lab sessions on AI:** Comprises of 10 sessions where in 20 programming tasks are framed to be practiced in lab
- **Lab sessions on Machine learning:** Comprises of 10 sessions where in 20 programming tasks are framed to be practiced in lab

Wish you all the best!!

MCSL-069 ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LAB

Structure	Page No.
1. Introduction	5
2. Objectives	5
3. General Guidelines	6
4. Python Libraries for AI and Machine Learning	7
5. Working with Python	13
6. Running Python Programs	15
7. Solved Examples of Artificial Intelligence and Machine learning	20
8. List of Lab Assignments-Session wise	34
9. References	39

1. INTRODUCTION

This lab course is designed to have the hands-on experience in AI/ML domain and related algorithms. Further, the AI/ML techniques, algorithms, libraries and frameworks are used to analyse the real-time problems, dataset analysis, interpretations and visualizations.

This course focuses on the implementation aspects of various AI/ML algorithms using Python programming language. For experimentation, Google Collab or ANACONDA, Jupyter Notebook, Eclipse or any relevant platform can be used for execution. Kindly go through the general guidelines and the programming documents carefully before starting experimentations.

This lab course is important to first get the theoretical and mathematical understanding of various algorithms presents in the theory course MCS-224 (Artificial intelligence and Machine Learning) and then implement your understanding in Python programming in this lab course, to solve the problems given in the various sessions of this course.

2. OBJECTIVES

After completing this lab course you will be able to:

- To develop a strong conceptual understanding of AI/ML, including their scope, significance, and application in solving complex and hard-to-solve problems within a reasonable time by leveraging domain knowledge.
- To familiarize learners with different knowledge representation techniques such as Propositional Logic, First Order Predicate Logic, Rule-based Systems, Semantic Networks, and Frames, and understand how these techniques contribute to building intelligent systems.
- To introduce learners to the fundamental concepts, techniques, and algorithms in Machine Learning, including Supervised and Unsupervised Learning, and to explore their practical applications across various domains.

- (d) To equip learners with the foundational knowledge and practical insights required to pursue further studies or careers in the rapidly growing fields of Artificial Intelligence and Machine Learning.

3. GENERAL GUIDELINES

- It is mandatory to complete all the problems and assignments given in each session of this lab course.
- Lab instructor will be available throughout the lab hours and can support you while working on the exercises. However, you can ask technical difficulties that you encounter related to the algorithm implementation in AI/ML concepts.
- It is strongly recommended that you should add proper comments in the code (preferably before each function, including main function) to give description of function's functionality. The comments should explain the purpose, meaning of the function, and its return value.
- In case, there is need to explain inside the code for better understanding then it should be placed immediately above the relevant function or algorithm's source code.
- The explanation may include where and when the function is necessary in the programming.
- A well-documented program with real input/output data and proper interactivity will be rewarded.
- There should not be any form of plagiarism. For example, if submission from two or more students found to be the same then none of such submission will be counted. Thus, it is strongly advised that you should not copy somebody else's work.
- It is mandatory to keep the observation book and lab record.
- A separate directory for each student will ensure that no body can read or copy your work.
- In the lab manual, the list of the programs are available. These programs are session wise. These programs are algorithms-based assignments. Thus, it is expected that you must prepare the algorithms and the programs written in the observation book. Lab hours are kept for your experimentation, execution, testing and analysis of the programs.
- It is your responsibility to inform the lab instructor/in-charge after completing the exercise for evaluation. After evaluation, you should get the signature from him/her on the observation book.
- Completed lab assignments should be submitted in the form of a Lab Record in which you have to write the algorithm, program code along with comments and output for various inputs given.
- The total no. of lab sessions (3 hours each) are 20 and the list of assignments is provided session-wise. It is important to observe the deadline given for each assignment.

4. PYTHON LIBRARIES FOR AI AND MACHINE LEARNING

Python is a popular programming language for AI/ML, Neural Networks, Time-series Analysis, Mathematical derivations and many more. This programming language is well developed to solve complex problems and prototype development in real-time applications. In Python programming language, there are various operating systems calls, libraries, modules and packages are available which are extensions of other programming languages like C, C++ or Java. Popular originals use this language in their valuable projects. Python is a high-level, general purpose and interpreted programming language with automatic garbage collection, multi-threading programming paradigms, support for both object-oriented and functional programming, and comprehensive standard library support.

Python is popular in AI/ML domains as well with its great support in emerging fields in computer science. For example, it supports key libraries include NumPy for numerical computations, Pandas for data manipulation, and Matplotlib for data visualization. For building and training deep learning models, Python offers TensorFlow and PyTorch, which are widely used in both academia and industry. In Natural Language Processing (NLP), libraries like NLTK and spaCy provide tools for text processing, while scikit-learn offers extensive support for traditional ML algorithms like regression, clustering, and classification. For reasoning under uncertainty and probabilistic modelling, pgmpy (for probabilistic graphical models) and Pyro (for probabilistic programming) are highly useful. Libraries like SymPy support symbolic reasoning, and tools such as Pyke allow for rule-based inferencing.

- (a) **SymPy:** This library is a powerful python library for symbolic mathematical calculations. Further, it is useful for users to perform algebraic manipulations, calculus, equation solving and many more. This library is mainly used when you deals with formulas, expressions or mathematical objects without replacement of a specific value. For example, this library can perform differentiation, integration, simplification, matrix operations, and solving systems of equations, making it popular in scientific computing, engineering, and research. This library can be integrated with major tools like Jupyter Notebooks, Google Collab and others and it is easy to install and use. This library can develop reasoning with mathematical expressions while handling uncertainties or variable dependencies in symbolic form.
- (b) **Pyke:** This library is useful in rule-based inference engine and can support for forward chaining and backward chaining for logical reasoning development. It is very useful in knowledge-based system development with facts, rules and inferencing mechanisms to solve the problems. For example, problems with reasoning with uncertainty or incomplete information can easily be developed using this library. It uses declarative rule language and Python code for interaction with rules and develop the knowledge bases. Among real-time applications, this library is useful for expert systems, diagnostic systems and other automated reasoning-based systems where encoding the expert knowledge and applying logical

inference for conclusions are important. This library is useful to enable other Python libraries that are made for building hybrid systems combining machine learning, rule-based reasoning and probabilistic inference.

- (c) **LogPy:** This library is for logic programming that allows users to perform declarative programming and symbolic reasoning. It is very useful for establishing relations and logic rules to find solutions. The problems which can be efficiently handled using this library include constraint satisfaction, inference engines, knowledge representation and many more. Further, the matching structures unification, backtracking in finding the solutions, recursive queries, rule-based AI, natural language processing and many more issues can be efficiently handled with this library. It is similar to Prolog for logic development with an addition of seamlessly integration with Python.
- (d) **NetworkX:** This is a powerful Python library designed for initiation, creation, manipulation and analysis of complex networks and graphs. Various types of graphs which are supported using this library include: directed, undirected, multigraphs, modelling of social networks, transportation systems-based graphs, biological networks and representations and many more. This library compute centrality measures, shortest paths, clustering coefficients, community detection and many more. This library is very useful in visualization, interoperability of various data formats, research purposes, data scientists and engineering working in network-based analysis.
- (e) **rdflib:** This library is developed to work with resource description framework data in semantic web. Using this library, users can create, parse, query and serialize the resource description framework with flexible graph-based model. Here, data is represented with triples (subject, predicate, object). In this library, popular resource description framework' format like Turtle, RDF/XML, N-Triples and JSON-LD can use used. Additionally, support for SPARQL, which is standard query language for resource description framework, is available. This allows the developers to effectively search and filter large semantic datasets. This is widely used for linked data applications, knowledge graph development and ontology management.
- (f) **OpenCog:** This is an open-source framework for artificial general intelligence systems. Here, a set of tools, components and algorithms are available for knowledge representation, develop reasoning and data learning. The main core of this library is the AtomSpace which is a flexible graph-based knowledge representation systems to store and manipulate concepts, associated relationships, terminologies and facts. This library also supports natural language processing, machine learning, evolutionary programming and pattern mining. This library is a good tool for advanced AI research and other AI-related tasks which can make it a generalize and reason-based preference across different domains.
- (g) **Scikit-learn:** This is a main machine learning library for simple and efficient data mining and predictive modelling aspects using various tools and features available in this. This library uses NumPy, SciPy and

Matplotlib for wide array of algorithms including supervised and unsupervised learning algorithms. Further, it provides support for classification, regression, clustering and dimensionality reduction. Here, user have an option for model selection, preprocessing and evaluation. Thus, it is a comprehensive toolkit for end-to-end machine learning workflows. This library is very common in-use and strong community support made it a go-to library for users including data scientists, developers, students, faculty, researchers and other practitioners.

- (h) **TensorFlow:** This is an open-source library useful for AI/ML-related tasks. Using this library, various tasks like deep learning, neural networks, computer vision, natural language processing and numerical computations can be handled easily. This library provides more flexibility to architectures that allowed models to deploy different platforms such as CPUs, GPUs and TPUs for mobile and edge devices. Further, it support for high-level APIs like Keras for rapid prototype development and improved performance. Using this library lower-level optimization operations for advanced customization can be performed easily. It is also very useful for research and production environments especially research tasks related to model training, evaluation, visualization and deployment. This is a graph-based computational model that provides execution and optimization of large-scale machine learning workflows. Overall, this library is very useful in AI ecosystem.
- (i) **PyTorch:** This is an open-source machine learning library and very useful for deep learning and natural language processing tasks. This library is built using Torch library and very flexible and easy-to-use platform for developing machine learning models. This library is very useful for supports in dynamic, computational graphs, developers to modify the network behaviours, and complex model building and debugging. Using this library, it is easy to perform tasks like image classification, computer vision, natural language processing and reinforcement learning. This library can easy be integrated with NumPy, SciPy and Cython. With this integration, it is easy to perform seamless data processing and model deployment. Overall, this library, because of its features, performance and flexibility, is widely used for academic research and industrial applications.
- (j) **Keras:** This is an open-source, high-level neural network API. This library can be used for building and training deep learning models. This library is simple, easy to use, modular and helpful for sequential and functional APIs especially in field of deep learning models. This library has wide range of pre-trained models and layers. As a result, tasks like image classification, object detection and text processing can be simplified. This library is very useful in quick prototyping, model development and processing. Library has the support for backends like TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK). In GUP and TPU processing, this library is also useful for making efficient large-scale machine learning tasks.
- (k) **Pandas:** This python library is an open-source library for data manipulation and analysis. This library is very useful for handling structured data. The library provides a powerful data structures and functions in data cleaning, transformation and analysis tasks which make the machine learning much

easier for real-time applications. This library can presents the large datasets in tabular form like SQL tables or excel spreadsheets does. There are various other tasks and functionalities which this library can perform like data indexing, reshaping, merging and aggregation. All of these tasks are important for data scientists and analysts in preparing data machine learning models or visualization. Thus, in data science ecosystem, this library can handle missing data, time series data analysis and complex data operations.

- (l) **NumPy:** This library can do numerical computing for large multi-dimensional arrays and matrices. The library is high-performance multidimensional array object called 'ndarray'. Using this library, a comprehensive set of mathematical functions and operations for arrays can be conducted. As a result, this library can do extensive scientific computations, data analysis and machine learning applications. The important areas where these tasks can be conducted include linear algebra, Fourier transformations, random number generation and statistical computations. Like any other useful library, this library can be integrated with other libraries like Pandas, Matplotlib, TensorFlow, and SciPy. Overall, data processing and mathematical computations would be much easier with this library.
- (m) **Matplotlib:** This data visualization library are useful in creating static, animated, and interactive visualizations. Using this library, tasks like 2D plots with line charts, bar graphs, histograms, scatter plots and pie charts can be drawn easily. In addition to 2D plots, this library can be useful for 3D plotting as well and capabilities. Using this library, it is easy to do high-level plotting functions. For example, 'pyplot' module in this library allows users to customize every aspect of a plot. In this plot, labels, legends, colour maps and grid lines can be easily drawn. Using datasets and available libraries, it is easy to perform data analysis, machine learning and scientific computing in visualization. Using Matplotlib library, integration with other libraries like Pandas and NumPy will make this a versatile tools for data science and research activities.
- (n) **Seaborn:** This library is important visualization library built on top of Matplotlib. This library is helpful in creation of aesthetically pleasing and informative statistical graphics. For example, heatmaps, violin plots, box plots, pair plots, categorical plots can be easily drawn using this library as it provides a high-level interface for drawing. To visualize the complex relationships in datasets, this library support through various built-in themes and colour palettes. With Pandas data structure integration, plotting of DataFrames Seaborn is very easy. This library is very useful in data analysis and machine learning tasks as well. The importance of this library in these tasks mainly include the exploration and visualization of data patterns effectively.
- (o) **AIMA-Python:** This library is an open-source library for implementing algorithms and examples. The algorithms and examples which are mainly using this library include search algorithms, game playing, knowledge representation, probabilistic reasoning, machine learning and reinforcement learning. Students, researchers and practitioners can easily take advantage of this library in understanding and experimenting with AI concepts mainly

through Python codes. This library is easy to understand in terms of implementation and building AI algorithms. Here, real world applications can be easily taken up for learning AI algorithms and their analysis.

- (p) **SciPy:** This library is an open-source software for mathematics, science and engineering. This library uses NumPy and provides a collection of mathematical examples, algorithms, functions and tools for scientific and technical computations. In this library, there are modules available for optimization, linear algebra, integration, interpolation, signal processing, image processing, statistical computations and various other analysis. This library can be used for data science, machine learning and scientific research. In these tasks, this library mainly performs complex mathematical computations and handling of large datasets. This library is flexible enough to handle robust scientific and numerical tasks which researchers and engineers have to perform in solving mathematical problems and analysis.
- (q) **NLTK:** This python library is very powerful and widely used. This library can be used with human language data and provide easy to use interface for over 50 corpora and lexical resources. The major tasks such as WordNet, includes text processing for tokenization, stemming, tagging, parsing and semantic reasoning. This library is widely used for natural language processing applications which enable the important tasks like sentiment analysis, text classification, language modelling and text analysis. As it has wide use in real-time applications, the students, researchers and other IT professionals use this library at large for prototyping and experimenting with language-based tasks.
- (r) **spaCy:** This is another advanced and efficient python library for natural language processing tasks. This library was mainly developed for production use, give fast performance and easy to use. Like other natural language processing libraries, this library can also perform tokenization, entity recognition, part of speech tagging, dependency parsing and word vectors. This library is widely used for industrial natural language processing application as it provides high-speed processing capabilities and pre-trained models for multiple languages. This library support deep learning workflows and can be integrated with popular frameworks like TensorFlow and PyTorch. All of these functionalities make this library an ideal choice for building scalable natural language processing applications.
- (s) **OpenAI Gym:** This library is popularly used for OpenAI-based reinforcement learning algorithms and associated tasks. This library can be widely used for pre-built environments like classic control problems, Atari games, and simulated robotics tasks. These tasks help the researchers and developers to test and train reinforcement learning models. OpenAI Gym library simplifies the process of design, test and benchmarking reinforcement learning algorithms. Here, there is use of standard API and a large suite of challenging environments. All of these features make this library an efficient tool for researchers, academia and industry applications. With this library, intelligent agents can be build that are capable of learning from interactions with the environment.

- (t) **Stable-Baselines3:** This library is a popular and reliable library built on top of PyTorch in developing and training reinforcement learning agents. Using this library, one can easily make a clean, efficient and modular implementation of important reinforcement learning algorithms. For example, Deep Q-Learning (DQN), Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Soft Actor-Critic (SAC) can easily be practiced using this library. This library is developed for an ease of use, rapid experimentation and reproducibility in reinforcement learning research and applications. This library also support OpenAI Gym environments which can be easily integrated with custome environments and make a preferred choice for both cacademia and industry-based real-time applications and projects.
- (u) **PySC2:** This is a Python-based reinforcement learning library and also known as Python StartCraft II Learning Environment. This library is developed for extensive research in complex strategic game environments. For example, this library provides an interface to the popular real-time strategy game StarCraft II. Thus, it is easy to train and evaluate reinforcement learning agents using this library especially for a challenging multi-agent settings. This library exposes a rich and complex environment with good observability, long-term planning and multi-agent interactions. All of these features make this library more useful and ideal for advancing the reinforcement learning algorithms especially in real-world scenarios.
- (v) **Dopamine:** This is another library useful for reinforcement learning algorithms and associated tasks. This library is a lightweight and flexible Python library for designing and evaluating important algorithms focused on deep reinforcement learning. This library is easy-to-use in terms of its implementation in standard reinforcement learning algorithms such as Deep Q-Learning or its variants. This library is very much in real-time use and allows the researchers or professional to experiment quickly and iteratively with new ideas. As a result, this library provides reproducibility, simplicity and benchmarking. This library provides support for seamlessly integration with environments like OpenAI Gym and Atari 2600 games.
- (w) **Feature-engine:** This is an open-source Python library designed for engineering jobs in machine learning pipelines. This library provides a set of tools as well which can be used to transform and engineer new features after analysing raw data, applying encoding categorical variables, handling missing data, innovating creative and interactive terms and conducting scalable numerical features. This library is designed to work seamlessly with the scikit-learn library and ensure various feature engineering steps. These steps are useful in machine learning pipelines which is the main purpose of this library. This library can be useful in improving the performance with high quality input features, creativity and managing the tasks in an organized and reproducible way.

5. WORKING WITH PYTHON

The important ways of using Python for Artificial Intelligence and Machine Learning are described as follow:

Method 1: Installing Python Locally

Using this method, Python is downloaded and installed from official Python website and then used directly via a terminal or command prompt. Further, an IDE (like Eclipse, PyCharm, VS Code or Jupyter Notebook) can also be used for experimentation.

Steps:

1. Go to website : <https://www.python.org>
2. Download and install the latest Python version for use.
3. Install libraries using command like: `pip install numpy pandas scikit-learn matplotlib seaborn tensorflow keras`
4. Run Python code in text editors like: VS Code, PyCharm, Sublime Text, etc.

There are various advantages and disadvantages of this approach. The major advantages include: (i) This installation provides full control over the environment and your configurations. (ii) Python scripts can run independently. (iii) This method is preferred approach for large and advanced Machine Learning Projects.

Method 2: Using Anaconda with Jupyter Notebook

In this method, Anaconda is installed first followed by data science platform that comes pre-installed with Jupyter Notebook for experimentation. This method can be preferably used for Machine Learning and Data Science tasks.

Steps:

1. Goto website: <https://www.anaconda.com>
2. Download and install the most suitable version for your operating system.
3. Open the Jupyter Notebook using following command: `jupyter notebook`
4. Write and run Machine Learning code:
 - a. After opening the Jupyter Notebook, libraries like pandas, sklearn, etc. can be installed and coding for problem statements can be started.

This method also has various advantages and disadvantages. The major advantages include: (i) It is an easy-to-use method with web-based interface. (ii) Pre-installed libraries and environment parameters save time. (iii) This method is preferred for prototyping, data analysis and Machine Learning model development. The major disadvantages of this method are: (i) it takes 2 to 3 GB of space for installation i.e. installation is slightly heavier. (ii) This method may arise dependency conflicts.

Method 3: Using Google Collab (an online method)

This is an online cloud-based platform for machine learning practice. This is developed by Google and a free Jupyter Notebook environment that runs entirely on Google Servers.

Steps:

1. **Goto Website:** <https://colab.research.google.com>
2. Click on New Notebook
3. Import libraries in cell like:

```
import pandas as pd
import numpy as np
import sklearn
```
4. Execute it.
5. You can select runtime GPU/TPU for leveraging the hardware acceleration.

This method also has various advantages and disadvantages. For example, (i) No installation required in this method. (ii) it provides free access to GPUs/TPUs for faster model training. (iii) It is an easy platform for collaboration and sharing notebooks. The major disadvantages include: (i) this method required active internet connection to work, and (ii) It provides limited compute time per session.

Method 4: Using Kaggle Kernels (an online method)

This method is similar to previous method (i.e. Google Colab). This method uses Kaggle for online practice. This provides an opportunity to directly integrates with datasets from Kaggle competitions and supports from Jupyter Notebook-style coding.

Steps:

1. Goto website: <https://www.kaggle.com/kernels>
2. Create a New Kernel/Notebook, and execute your code.

The major advantages of this method are: (i) This platform provides access to thousands of public datasets. (ii) It has pre-installed libraries and tools for practice. (iii) It provides support for GPU/TPU like Colab. The major disadvantages of this method are: It requires an active internet connection, and (ii) It has limited compute resources.

Method 5: Using Docker Containers

This method support the use of docker for creating isolated environments for different machine learning projects. Thus, it is very suitable for large-scale or production-level deployments.

Steps:

1. Goto website: <https://www.docker.com>
2. Download Docker and install it.
3. Pull Machine Learning Containers like:

```
docker pull jupyter/scipy-notebook
```
4. Run the container like:

```
docker run -p 8888:8888 jupyter/scipy-notebook
```
5. Access the Jupyter Notebook in the browser using URL: <http://localhost:8888>

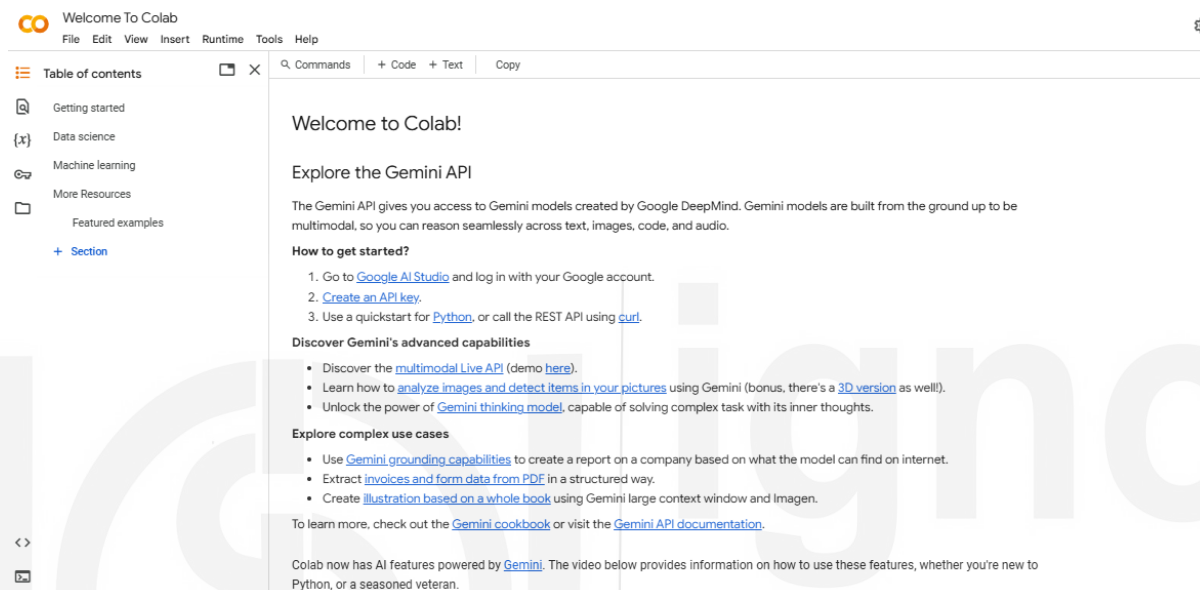
This method has various advantages like: (i) Provides complete isolation of the environment, (ii) It is comparatively use to use compare to other methods, (iii) This method is preferred method for deploying models in production. The major

disadvantages of this methods are: (i) This is a resource-intensive methods, and (ii) It give higher learning curve.

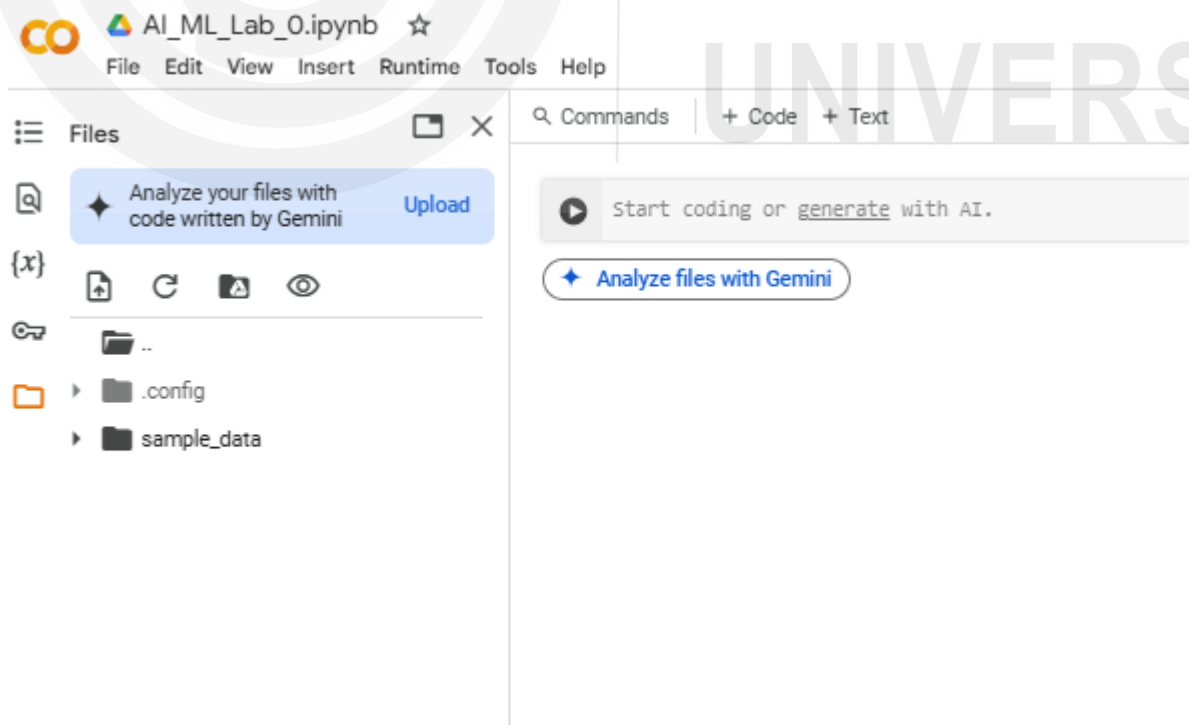
6. RUNNING PYTHON PROGRAMS

Follow are the stating steps and example for running python programs n this lab:

Step 1: In google Colab, click file option and select new workbook.



Step 2: A new Jupyter Notebook will open where you can enter your code and execute it.

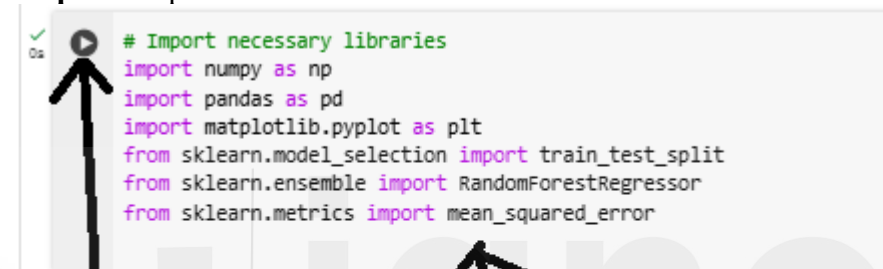


Step 3: This step demonstrates an example and its execution in Google Colab.

Problem: Consider you are employee of a petroleum industry-based organization, and your task is to predict the production rate of oil wells, optimizing extraction process and manage resources efficiently. As this sector is very sensitive, it is very hard to collect real-world data. Thus, your role is to first generate synthetic data for simulating oil well production rate based on various factors like reservoir pressure, well depth and water cut. Write python program to build AI/ML model for predicting production rates and analysing the impact of various factors.

(**Note:** This problem is in-depth AI/ML-based problem for you at this beginning state. Aim of you is to just learn the use of Google Colab, synthetic data generation, python-based code execution in Google Colab, and observe the outputs).

Step 3a: Import libraries

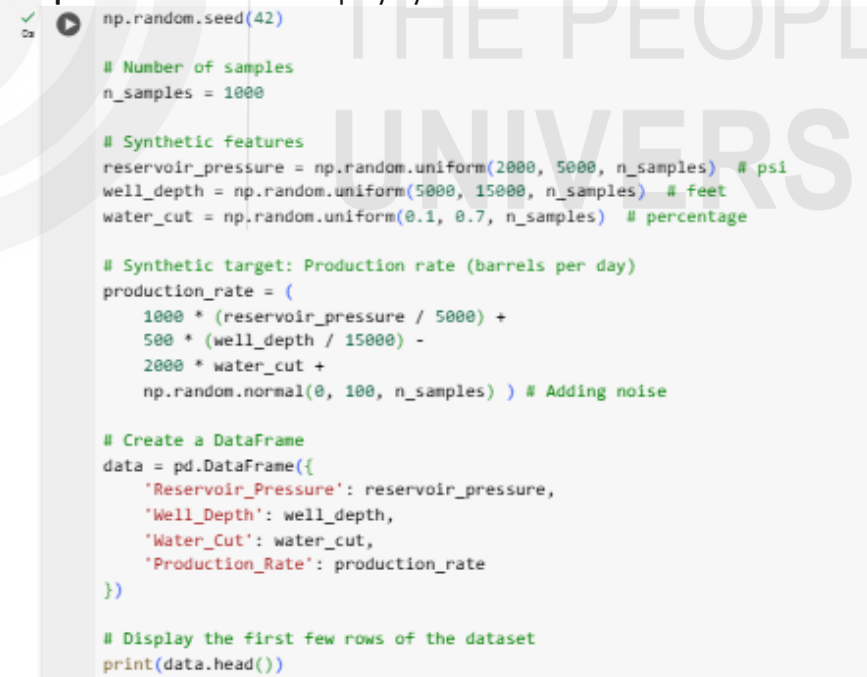


```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

Step 2: Click here to execute.

Step 1: Enter your code in cell.

Step 3b: Generate and display Synthetic data.



```
np.random.seed(42)

# Number of samples
n_samples = 1000

# Synthetic features
reservoir_pressure = np.random.uniform(2000, 5000, n_samples) # psi
well_depth = np.random.uniform(5000, 15000, n_samples) # feet
water_cut = np.random.uniform(0.1, 0.7, n_samples) # percentage

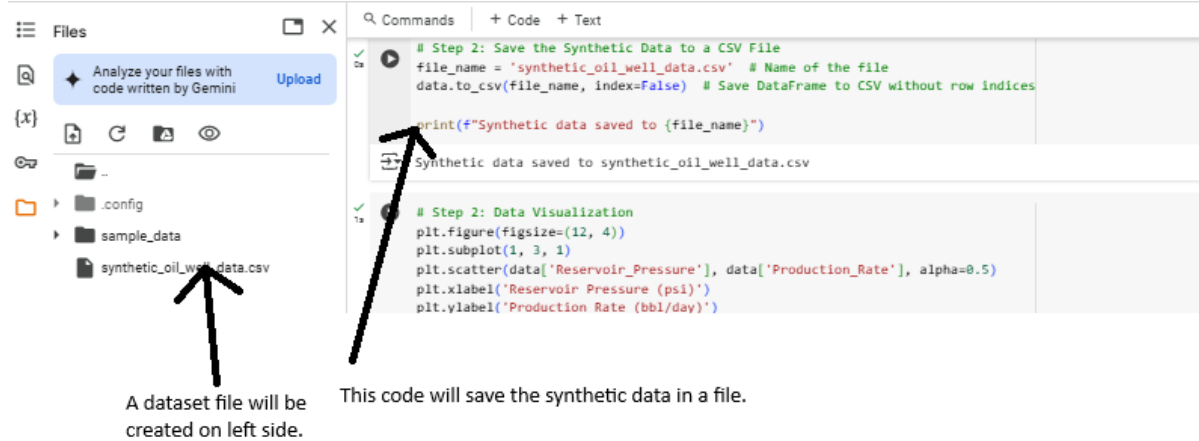
# Synthetic target: Production rate (barrels per day)
production_rate = (
    1000 * (reservoir_pressure / 5000) +
    500 * (well_depth / 15000) -
    2000 * water_cut +
    np.random.normal(0, 100, n_samples) ) # Adding noise

# Create a DataFrame
data = pd.DataFrame({
    'Reservoir_Pressure': reservoir_pressure,
    'Well_Depth': well_depth,
    'Water_Cut': water_cut,
    'Production_Rate': production_rate
})

# Display the first few rows of the dataset
print(data.head())
```

	Reservoir_Pressure	Well_Depth	Water_Cut	Production_Rate
0	3123.620357	6851.329288	0.257023	445.048532
1	4852.142919	10419.009474	0.248187	883.054936
2	4195.981825	13729.458359	0.643753	77.696413
3	3795.975453	12322.248864	0.249728	533.885708
4	2468.055921	13065.611479	0.263170	523.986294

Step 3c: Save the Synthetic data in a file.

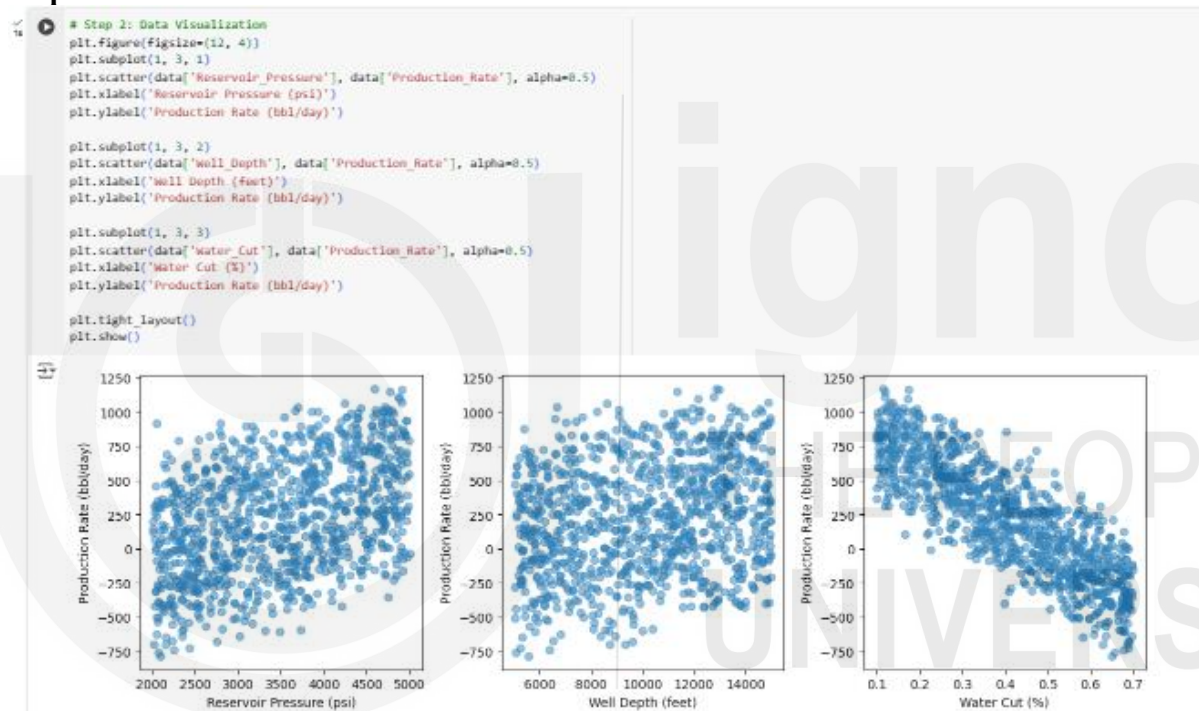


The screenshot shows a Jupyter Notebook interface. On the left, the 'Files' pane displays a directory structure with files like '.config', 'sample_data', and 'synthetic_oil_well_data.csv'. An arrow points from the text 'A dataset file will be created on left side.' to the 'synthetic_oil_well_data.csv' file. In the center, the 'Commands' pane shows two code cells. The first cell, labeled 'Step 2: Save the Synthetic Data to a CSV File', contains code to save a DataFrame to a CSV file named 'synthetic_oil_well_data.csv'. An arrow points from the text 'This code will save the synthetic data in a file.' to this code cell. The second cell, labeled 'Step 2: Data Visualization', contains code to create a scatter plot of 'Production Rate (bbl/day)' vs 'Reservoir Pressure (psi)'.

```
# Step 2: Save the Synthetic Data to a CSV File
file_name = 'synthetic_oil_well_data.csv' # Name of the file
data.to_csv(file_name, index=False) # Save DataFrame to CSV without row indices
print(f"Synthetic data saved to {file_name}")

# Step 2: Data Visualization
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.scatter(data['Reservoir_Pressure'], data['Production_Rate'], alpha=0.5)
plt.xlabel('Reservoir Pressure (psi)')
plt.ylabel('Production Rate (bbl/day)')
```

Step 3d: Visualize the data.



Step 3e: Preparing data for ML Model

```
# Step 3: Prepare Data for ML Model
X = data[['Reservoir_Pressure', 'Well_Depth', 'Water_Cut']]
y = data['Production_Rate']
```

Step 3f: Prepare the data into training and testing sets.

```
# Split 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)
```

Step 3g: Train the data using Random Forest Regressor

```
# Step 4: Train a Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

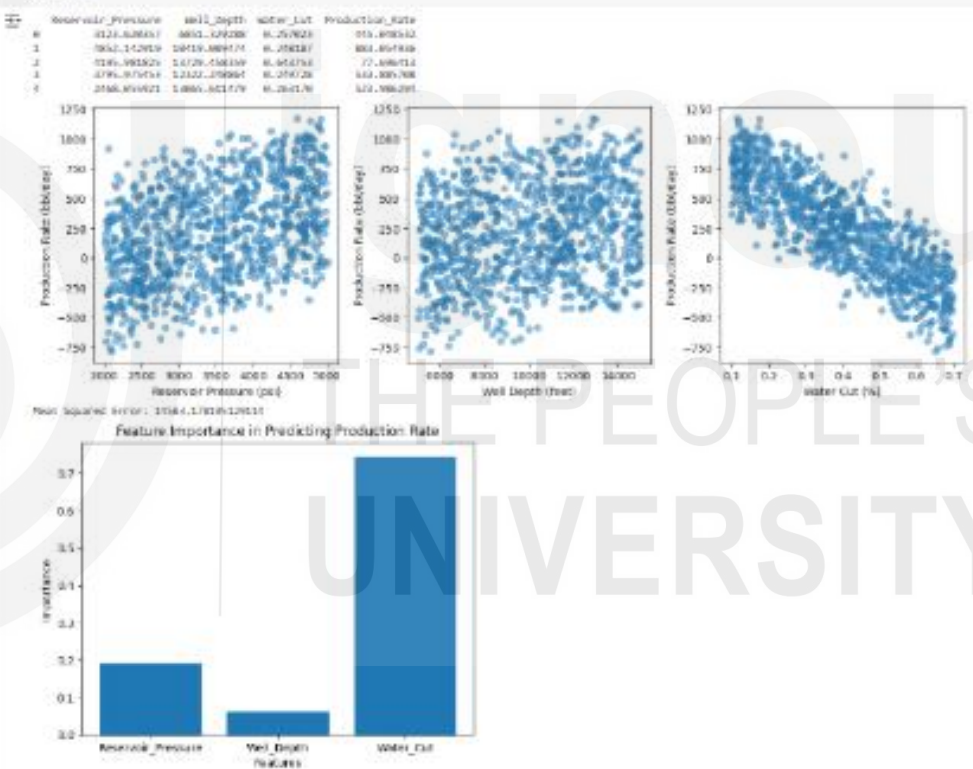
Step 3h: Evaluate the Model

```
# Step 5: Evaluate the Model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 14583.178195129114
```

Step 3i: Perform Feature Analysis

```
# Step 6: Feature Importance Analysis
feature_importances_ = model.feature_importances_
features = X.columns
plt.bar(features, feature_importances_)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance in Predicting Production Rate')
plt.show()
```



Following is the complete code:

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

Step 1: Generate Synthetic Data

```
np.random.seed(42)
```

Number of samples

```
n_samples = 1000
```

Synthetic features

```
reservoir_pressure = np.random.uniform(2000, 5000, n_samples) # psi
well_depth = np.random.uniform(5000, 15000, n_samples) # feet
water_cut = np.random.uniform(0.1, 0.7, n_samples) # percentage
```

Synthetic target: Production rate (barrels per day)

```
production_rate = (
    1000 * (reservoir_pressure / 5000) +
    500 * (well_depth / 15000) -
    2000 * water_cut +
    np.random.normal(0, 100, n_samples) # Adding noise
```

Create a DataFrame

```
data = pd.DataFrame({
    'Reservoir_Pressure': reservoir_pressure,
    'Well_Depth': well_depth,
    'Water_Cut': water_cut,
    'Production_Rate': production_rate
})
```

```
# Display the first few rows of the dataset
print(data.head())
```

Step 2: Data Visualization

```
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.scatter(data['Reservoir_Pressure'], data['Production_Rate'],
            alpha=0.5)
plt.xlabel('Reservoir Pressure (psi)')
plt.ylabel('Production Rate (bbl/day)')

plt.subplot(1, 3, 2)
plt.scatter(data['Well_Depth'], data['Production_Rate'], alpha=0.5)
plt.xlabel('Well Depth (feet)')
plt.ylabel('Production Rate (bbl/day)')

plt.subplot(1, 3, 3)
plt.scatter(data['Water_Cut'], data['Production_Rate'], alpha=0.5)
plt.xlabel('Water Cut (%)')
plt.ylabel('Production Rate (bbl/day)')

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

Step 3: Prepare Data for ML Model

```
X = data[['Reservoir_Pressure', 'Well_Depth', 'Water_Cut']]
y = data['Production_Rate']
```

Split 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)

# Step 4: Train a Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Step 5: Evaluate the Model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Step 6: Feature Importance Analysis
feature_importances = model.feature_importances_
features = X.columns
plt.bar(features, feature_importances)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance in Predicting Production Rate')
plt.show()

```

7. SOLVED EXAMPLES OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

In the following section we will study few of the solve problems related to artificial Intelligence as well as Machine learning.

7.1 Solved Examples of Artificial Intelligence

1. Write a Python Program to implement Alpha-Beta Pruning Algorithm

Solution:

```

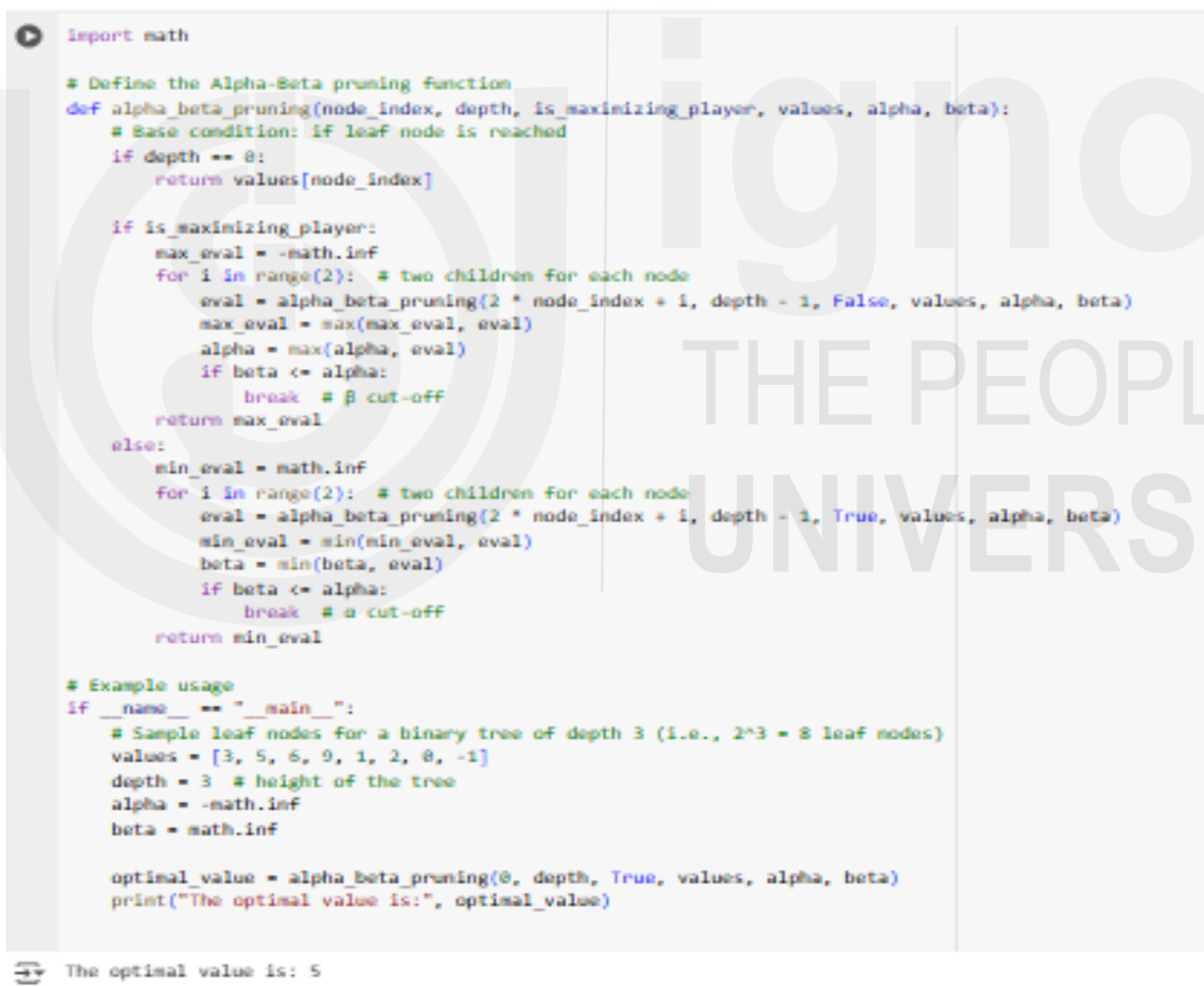
import math
def alpha_beta_pruning(node_index, depth, is_maximizing_player, values,
alpha, beta):
    # Base condition: if leaf node is reached
    if depth == 0:
        return values[node_index]
    if is_maximizing_player:
        max_eval = -math.inf
        for i in range(2): # two children for each node
            eval = alpha_beta_pruning(2 * node_index + i, depth - 1, False,
values, alpha, beta)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break #  $\beta$  cut-off
        return max_eval
    else:
        min_eval = math.inf

```

```

    for i in range(2): # two children for each node
        eval = alpha_beta_pruning(2 * node_index + i, depth - 1, True,
values, alpha, beta)
        min_eval = min(min_eval, eval)
        beta = min(beta, eval)
        if beta <= alpha:
            break #  $\alpha$  cut-off
    return min_eval
# Example usage
if __name__ == "__main__":
    # Sample leaf nodes for a binary tree of depth 3 (i.e.,  $2^3 = 8$  leaf nodes)
    values = [3, 5, 6, 9, 1, 2, 0, -1]
    depth = 3 # height of the tree
    alpha = -math.inf
    beta = math.inf
    optimal_value = alpha_beta_pruning(0, depth, True, values, alpha, beta)
    print("The optimal value is:", optimal_value)
Output:

```



```

import math

# Define the Alpha-Beta pruning function
def alpha_beta_pruning(node_index, depth, is_maximizing_player, values, alpha, beta):
    # Base condition: if leaf node is reached
    if depth == 0:
        return values[node_index]

    if is_maximizing_player:
        max_eval = -math.inf
        for i in range(2): # two children for each node
            eval = alpha_beta_pruning(2 * node_index + i, depth - 1, False, values, alpha, beta)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break #  $\beta$  cut-off
        return max_eval
    else:
        min_eval = math.inf
        for i in range(2): # two children for each node
            eval = alpha_beta_pruning(2 * node_index + i, depth - 1, True, values, alpha, beta)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break #  $\alpha$  cut-off
        return min_eval

# Example usage
if __name__ == "__main__":
    # Sample leaf nodes for a binary tree of depth 3 (i.e.,  $2^3 = 8$  leaf nodes)
    values = [3, 5, 6, 9, 1, 2, 0, -1]
    depth = 3 # height of the tree
    alpha = -math.inf
    beta = math.inf

    optimal_value = alpha_beta_pruning(0, depth, True, values, alpha, beta)
    print("The optimal value is:", optimal_value)

```

The optimal value is: 5

2. Write a Python Program to implement Iterative Deepening Depth First search (IDDFS)

Solution:

```
# Depth-Limited DFS
def DLS(graph, node, target, limit):
    print(f"Visiting Node: {node} at depth {limit}")
    if node == target:
        return True
    if limit <= 0:
        return False
    for neighbor in graph.get(node, []):
        if DLS(graph, neighbor, target, limit - 1):
            return True
    return False

# Iterative Deepening DFS
def IDDFS(graph, start, target, max_depth):
    for depth in range(max_depth + 1):
        print(f"\nSearching at depth level: {depth}")
        if DLS(graph, start, target, depth):
            print(f"Target node '{target}' found at depth {depth}")
            return True
    print(f"Target node '{target}' not found within depth {max_depth}")
    return False

# Example graph represented as an adjacency list
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['G'],
    'F': [],
    'G': []
}

# Example usage
if __name__ == "__main__":
    start_node = 'A'
    target_node = 'G'
    max_depth = 3
    IDDFS(graph, start_node, target_node, max_depth)
```

```

C # Depth-limited DFS
def dfs(graph, node, target, limit):
    print(f"visiting Node: {node} at depth {limit}")
    if node == target:
        return True
    if limit <= 0:
        return False
    for neighbor in graph.get(node, []):
        if dfs(graph, neighbor, target, limit - 1):
            return True
    return False

# Iterative Deepening DFS
def iddfs(graph, start, target, max_depth):
    for depth in range(max_depth + 1):
        print(f"Searching at depth level: {depth}")
        if dfs(graph, start, target, depth):
            print(f"target node '{target}' found at depth {depth}")
            return True
    print(f"target node '{target}' not found within depth {max_depth}")
    return False

# Example graph represented as an adjacency list
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['G'],
    'F': [],
    'G': []
}

# Example usage
if __name__ == "__main__":
    start_node = 'A'
    target_node = 'G'
    max_depth = 3
    iddfs(graph, start_node, target_node, max_depth)

```

```

III Searching at depth level: 0
visiting Node: A at depth 0

Searching at depth level: 1
visiting Node: A at depth 1
visiting Node: B at depth 0
visiting Node: C at depth 0

Searching at depth level: 2
visiting Node: A at depth 2
visiting Node: B at depth 1
visiting Node: C at depth 0
visiting Node: D at depth 0
visiting Node: E at depth 1
visiting Node: F at depth 0

Searching at depth level: 3
visiting Node: A at depth 3
visiting Node: B at depth 2
visiting Node: C at depth 1
visiting Node: D at depth 1
visiting Node: E at depth 0
target node 'G' found at depth 3

```

3. Write a Python Program to implement Best First Search algorithm

Solution:

```
import heapq

def best_first_search(graph, heuristics, start, goal):
    visited = set()
    priority_queue = []

    # Push the start node into the priority queue with its heuristic value
    heapq.heappush(priority_queue, (heuristics[start], start))

    while priority_queue:
        current_cost, current_node = heapq.heappop(priority_queue)
        print(f"Visiting Node: {current_node}")

        if current_node == goal:
            print(f"Goal node '{goal}' found!")
            return True

        if current_node not in visited:
            visited.add(current_node)

            for neighbor in graph.get(current_node, []):
                if neighbor not in visited:
                    heapq.heappush(priority_queue, (heuristics[neighbor],
neighbor))

        print(f"Goal node '{goal}' not reachable from '{start}'")
        return False

# Example graph (adjacency list)
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['G'],
    'F': [],
    'G': []
}

# Heuristic values for each node (lower is better)
heuristics = {
    'A': 5,
    'B': 3,
    'C': 4,
    'D': 6,
    'E': 2,
    'F': 6,
    'G': 1 # Goal node should have the lowest heuristic ideally
}
```

```
# Example usage
if __name__ == "__main__":
    start_node = 'A'
    goal_node = 'G'
    best_first_search(graph, heuristics, start_node, goal_node)
```

Output:

```
➡ Visiting Node: A
Visiting Node: B
Visiting Node: E
Visiting Node: G
Goal node 'G' found!
```

4. Write a Python Program to implement A* Algorithm

Solution:

```
import heapq

def a_star_search(graph, heuristics, start, goal):
    open_list = []
    heapq.heappush(open_list, (heuristics[start], 0, start, [start])) # (f = g + h,
g, node, path)
    closed_set = set()

    while open_list:
        f, g, current, path = heapq.heappop(open_list)
        print(f"Visiting Node: {current} with f = {f}, g = {g}, h =
{heuristics[current]}")

        if current == goal:
            print(f"Goal node '{goal}' found! Path: {' -> '.join(path)}")
            return path

        if current in closed_set:
            continue
        closed_set.add(current)

        for neighbor, cost in graph.get(current, []):
            if neighbor not in closed_set:
                g_new = g + cost
                f_new = g_new + heuristics[neighbor]
                heapq.heappush(open_list, (f_new, g_new, neighbor, path +
[neighbor]))

    print(f"Goal node '{goal}' not reachable from '{start}'")
    return None
```

Example weighted graph as adjacency list (node: [(neighbor, cost), ...])

```

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('D', 1), ('E', 3)],
    'C': [('F', 5)],
    'D': [],
    'E': [('G', 2)],
    'F': [],
    'G': []
}

# Heuristic values for nodes
heuristics = {
    'A': 6,
    'B': 4,
    'C': 5,
    'D': 2,
    'E': 3,
    'F': 6,
    'G': 0 # Goal node
}

# Example usage
if __name__ == "__main__":
    start_node = 'A'
    goal_node = 'G'
    a_star_search(graph, heuristics, start_node, goal_node)

```

Output:

```

➡ Visiting Node: A with f = 6, g = 0, h = 6
Visiting Node: B with f = 5, g = 1, h = 4
Visiting Node: D with f = 4, g = 2, h = 2
Visiting Node: E with f = 7, g = 4, h = 3
Visiting Node: G with f = 6, g = 6, h = 0
Goal node 'G' found! Path: A -> B -> E -> G

```

5. Write a Python Program to implement AO* Algorithm

Solution:

```

class Node:
    def __init__(self, name):
        self.name = name
        self.children = [] # List of tuples: (child_nodes, cost)
        self.heuristic = 0
        self.solved = False
        self.best_child = None

def ao_star(node, graph, solved_nodes):
    if node.solved:
        return node.heuristic

    print(f"Expanding Node: {node.name}")

```

```

min_cost = float('inf')
best_option = None

for child_group, cost in graph.get(node.name, []):
    total_cost = cost
    for child_name in child_group:
        total_cost += nodes[child_name].heuristic

    if total_cost < min_cost:
        min_cost = total_cost
        best_option = (child_group, cost)

if best_option is None:
    node.solved = True
    return node.heuristic

node.best_child = best_option[0]
node.heuristic = min_cost

all_solved = True
for child_name in best_option[0]:
    child_node = nodes[child_name]
    ao_star(child_node, graph, solved_nodes)
    if not child_node.solved:
        all_solved = False

if all_solved:
    node.solved = True
    solved_nodes.add(node.name)

return node.heuristic

def print_solution(node):
    if node.solved:
        print(f"{node.name}", end="")
        if node.best_child:
            print(" -> (", end="")
            for i, child in enumerate(node.best_child):
                print(child, end="")
                if i != len(node.best_child) - 1:
                    print(" AND ", end="")
            print(")")
            for child in node.best_child:
                print_solution(nodes[child])

# Define graph structure
# Format: node: [(child_nodes), cost), ...]
graph = {
    'A': [(['B', 'C'], 1), (['D'], 2) ],
    'B': [(['E'], 1), (['F'], 2) ],
    'C': [(['G'], 2) ],
    'D': [],

```

```

'E': [],
'F': [],
'G': []
}

# Create nodes and heuristics
nodes = {
    'A': Node('A'),
    'B': Node('B'),
    'C': Node('C'),
    'D': Node('D'),
    'E': Node('E'),
    'F': Node('F'),
    'G': Node('G')
}

# Assign heuristic values (example)
nodes['A'].heuristic = 10
nodes['B'].heuristic = 4
nodes['C'].heuristic = 2
nodes['D'].heuristic = 6
nodes['E'].heuristic = 3
nodes['F'].heuristic = 2
nodes['G'].heuristic = 0

# Solve the graph
solved_nodes = set()
ao_star(nodes['A'], graph, solved_nodes)

# Print the solution
print("\nAO* Solution Path:")
print_solution(nodes['A'])

```

Output:

```

➡ Expanding Node: A
Expanding Node: B
Expanding Node: E
Expanding Node: C
Expanding Node: G

AO* Solution Path:
A -> (B AND C)
B -> (E)
EC -> (G)
G

```

7.2 Solved Examples of Machine Learning

1. Write a Python Program to implement Polynomial Regression on a dataset of your own choice

Solution:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv("dummy_polynomial_data.csv")
X = data[['Feature']].values
y = data['Target'].values

# Create polynomial features
degree = 2 # You can change the degree to experiment
poly = PolynomialFeatures(degree=degree)
X_poly = poly.fit_transform(X)

# Fit the model
model = LinearRegression()
model.fit(X_poly, y)

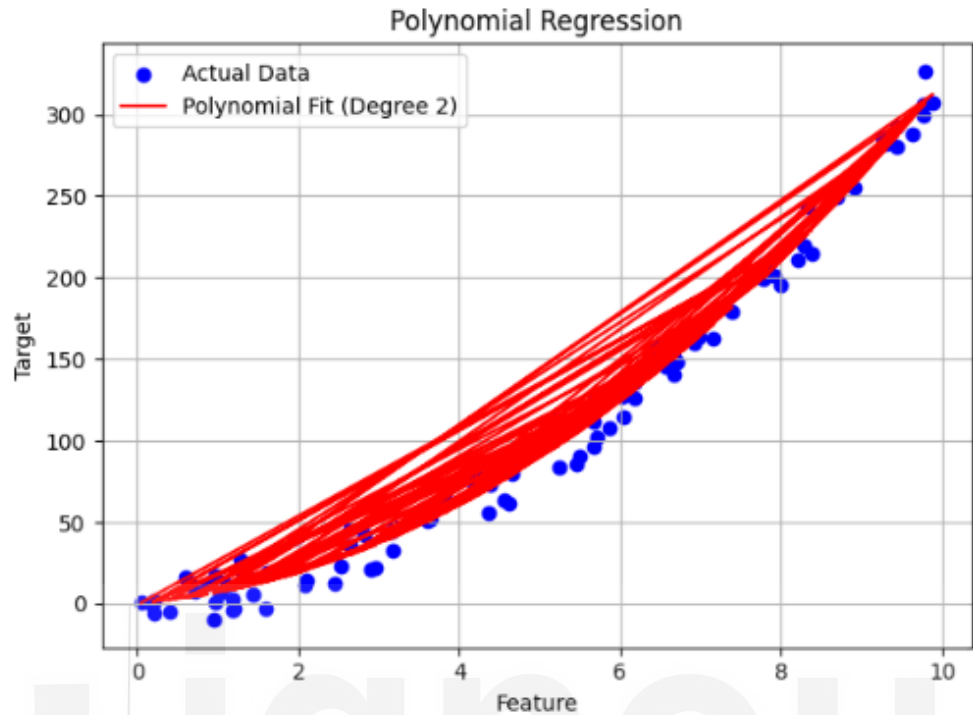
# Predict using the trained model
y_pred = model.predict(X_poly)

# Evaluation
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'R2 Score: {r2:.2f}')

# Plotting
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', label=f'Polynomial Fit (Degree {degree})')
plt.title('Polynomial Regression')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:

↔ Mean Squared Error: 97.36
R² Score: 0.99



2. Write a Python Program to implement Feed Forward Neural Network on a given dataset for data classification, choose dataset of your own choice

Solution:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# Step 1: Create dummy classification dataset
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2,
random_state=42)
```

```
# Optional: Save dataset
df = pd.DataFrame(X, columns=[f'Feature_{i}' for i in range(1, 11)])
df['Label'] = y
df.to_csv("classification_dataset.csv", index=False)
```

```
# Step 2: Preprocess
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Step 3: Build the FFNN model
model = Sequential([
    Dense(16, activation='relu', input_shape=(X.shape[1],)),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid') # Binary classification
])

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

# Step 4: Train the model
model.fit(X_train, y_train, epochs=20, batch_size=16, verbose=1)

# Step 5: Evaluate
y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, y_pred))
```

Output:

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an 'input_shape'/'input_dim' argument to a layer. When using Sequential models, prefer using an 'input(shape)' object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
50/50 ----- 1s 3ms/step - accuracy: 0.5948 - loss: 0.6645
Epoch 2/20
50/50 ----- 0s 2ms/step - accuracy: 0.7700 - loss: 0.5810
Epoch 3/20
50/50 ----- 0s 2ms/step - accuracy: 0.8592 - loss: 0.4798
Epoch 4/20
50/50 ----- 0s 2ms/step - accuracy: 0.8641 - loss: 0.4111
Epoch 5/20
50/50 ----- 0s 2ms/step - accuracy: 0.8576 - loss: 0.3756
Epoch 6/20
50/50 ----- 0s 2ms/step - accuracy: 0.8845 - loss: 0.3193
Epoch 7/20
50/50 ----- 0s 2ms/step - accuracy: 0.8854 - loss: 0.2942
Epoch 8/20
50/50 ----- 0s 2ms/step - accuracy: 0.8675 - loss: 0.3361
Epoch 9/20
50/50 ----- 0s 2ms/step - accuracy: 0.8650 - loss: 0.3199
Epoch 10/20
50/50 ----- 0s 2ms/step - accuracy: 0.8767 - loss: 0.3026
Epoch 11/20
50/50 ----- 0s 2ms/step - accuracy: 0.8942 - loss: 0.2848
Epoch 12/20
50/50 ----- 0s 2ms/step - accuracy: 0.8824 - loss: 0.2983
Epoch 13/20
50/50 ----- 0s 2ms/step - accuracy: 0.8663 - loss: 0.3288
Epoch 14/20
50/50 ----- 0s 2ms/step - accuracy: 0.8744 - loss: 0.2868
Epoch 15/20
50/50 ----- 0s 2ms/step - accuracy: 0.8782 - loss: 0.3077
Epoch 16/20
50/50 ----- 0s 3ms/step - accuracy: 0.8877 - loss: 0.2853
Epoch 17/20
50/50 ----- 0s 3ms/step - accuracy: 0.8728 - loss: 0.2872
Epoch 18/20
50/50 ----- 0s 3ms/step - accuracy: 0.8774 - loss: 0.2888
Epoch 19/20
50/50 ----- 0s 4ms/step - accuracy: 0.8973 - loss: 0.2593
Epoch 20/20
50/50 ----- 0s 3ms/step - accuracy: 0.8679 - loss: 0.3005
7/7 ----- 0s 15ms/step

              precision    recall  F1-score   support
0               0.79         0.84         0.82         89
1               0.87         0.82         0.84         111

 accuracy          0.83
 macro avg         0.83
 weighted avg       0.83
```

3. Write a Python Program to implement Principal Component Analysis on a dataset of your own choice

Solution:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
# Load the dataset
df = pd.read_csv("classification_dataset1.csv")
```

```
# Separate features and target
X = df.drop('Label', axis=1)
y = df['Label']
```

```

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

# Apply PCA to reduce dimensions (e.g., to 2 components)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Explained variance ratio
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Create a new DataFrame for visualization
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
pca_df['Label'] = y

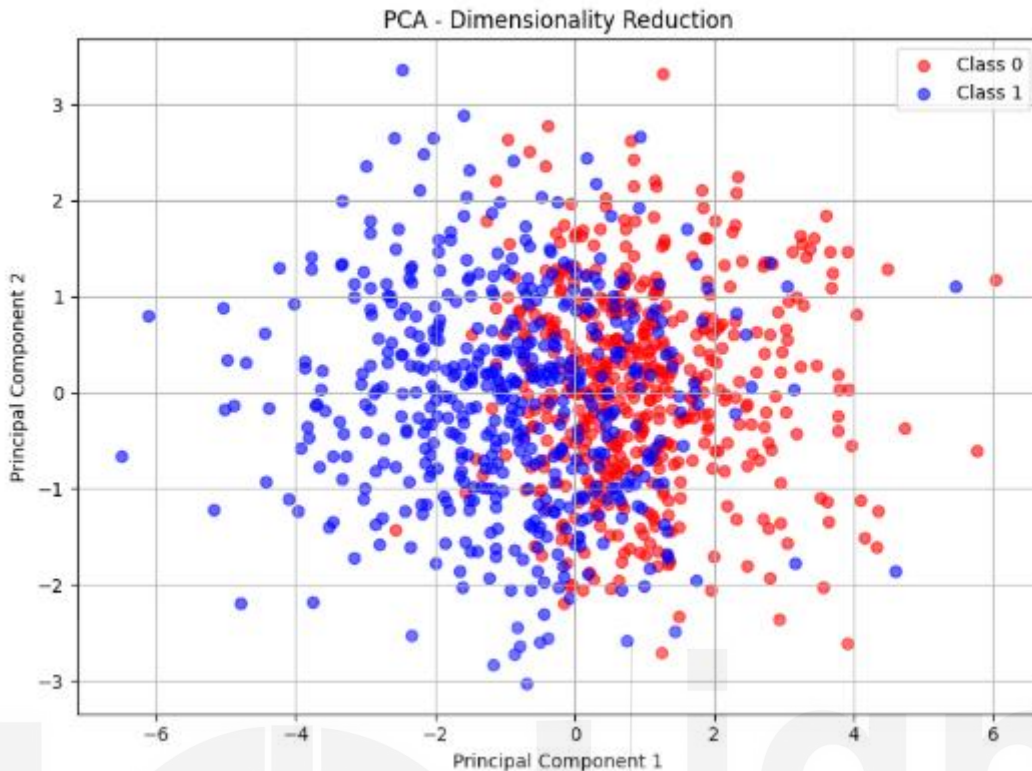
# Plot the PCA-reduced data
plt.figure(figsize=(8, 6))
colors = ['red', 'blue']
for label in [0, 1]:
    plt.scatter(pca_df[pca_df['Label'] == label]['PC1'],
                pca_df[pca_df['Label'] == label]['PC2'],
                label=f'Class {label}', alpha=0.6, color=colors[label])

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA - Dimensionality Reduction')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

```

Output:

Explained variance ratio: [0.30347958 0.11489532]



4. Write a Python Program to implement Linear Discriminant Analysis on a dataset of your own choice

Solution:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
# Load dataset
df = pd.read_csv("classification_dataset.csv")
```

```
# Separate features and label
X = df.drop('Label', axis=1)
y = df['Label']
```

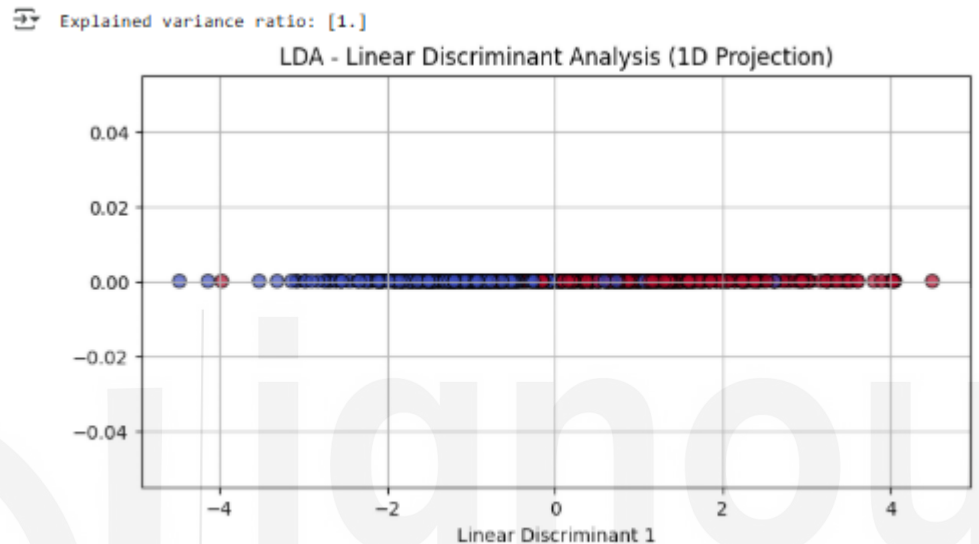
```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Apply LDA to reduce to 1 component (binary classification:
n_components = n_classes - 1 = 1)
lda = LDA(n_components=1)
X_lda = lda.fit_transform(X_scaled, y)
```

```
# Print explained variance ratio
print("Explained variance ratio:", lda.explained_variance_ratio_)
```

```
# Plot LDA-reduced data
plt.figure(figsize=(8, 4))
plt.scatter(X_lda, [0]*len(X_lda), c=y, cmap='coolwarm', edgecolor='k',
s=50, alpha=0.7)
plt.xlabel('Linear Discriminant 1')
plt.title('LDA - Linear Discriminant Analysis (1D Projection)')
plt.grid(True)
plt.show()
```

Output:



8. LIST OF LAB ASSIGNMENTS-SESSION WISE

Now, let us try solving a different kind of a AI and Machine Learning problems using Python.

8.1 LAB SESSIONS ON ARTIFICIAL INTELLIGENCE (AI)

In each of the problems given in the respective sessions, you are required to write the algorithm for the given problem, and map the steps of your algorithm with your python code, by using suitable comments.

Session 1:

- Write a Python Program to Solve N-Queen Problem without using Recursion.
- Write a Python Program to implement the Backtracking approach to solve N Queen's problem

Session 2:

- Write a Python Program to implement Min-Max Algorithm
- Write a Python Program to implement Breadth First Search
- Write a Python program to simulate a simple stimulus-response agent that reacts to input based on predefined rules.

- d) Write a Python program to Train a simple machine learning model (linear regression) to predict a value based on input data.

Session 3:

- a) Write a Python Program to implement Depth First Search
- b) Write a Python Program to implement Iterative Deepening DepthFirst search (IDDFS)

Session 4:

Write a Python Program to implement AO* Algorithm

Session 5:

Write a Python Program to implement for IDA* (Iterative Deepening A*) algorithm

Session 6:

You are given a 3x3 grid with 8 tiles and one empty space. The tiles are numbered from 1 to 8, and the goal is to rearrange the tiles from a given initial state to a goal state by sliding the tiles into the empty space. Use a search algorithm (e.g., Breadth-First Search or A*) and write a python program to find the sequence of moves required to solve the puzzle.

Session 7:

You are given a graph representing a road network with nodes as locations and edges as roads connecting them. Each edge has a uniform cost. Implement **Bidirectional Search using python** to find the shortest path between a start node and a goal node.

Session 8:

You are given a propositional logic formula, and your task is to determine whether it is **valid** (always true), **satisfiable** (true for some assignment of variables), or **unsatisfiable** (never true). For example, consider the formula $(P \vee Q) \wedge (\neg P \vee \neg Q)$. Write a Python program that takes such a formula as input, generates all possible truth assignments for the variables, and evaluates the formula to determine its validity and satisfiability. The program should output whether the formula is valid, satisfiable, or unsatisfiable.

Session 9:

You are given a set of FOL clauses, and your task is to use **resolution with variables** to determine whether the set is **satisfiable** or **unsatisfiable**. For example, consider the clauses: $\neg P(x) \vee Q(x)$, $\neg Q(y) \vee R(y)$, $P(a)$, and $\neg R(b)$. Write a Python program that applies the resolution rule iteratively, using unification to resolve clauses with variables, and checks whether the empty clause (indicating unsatisfiability) can be derived. The program should output whether the set of clauses is satisfiable or unsatisfiable.

Session 10:

You are given a knowledge base (KB) of FOL sentences and a query. Your task is to determine whether the KB **entails** the query, i.e., whether the query is true in all models where the KB is true. For example, consider the KB: $\forall x (P(x) \rightarrow Q(x))$ and $P(a)$, and the query $Q(a)$. Write a Python program

that checks whether the KB entails the query by constructing a proof or using a truth table approach. The program should output whether the KB entails the query.

8.2 LAB SESSIONS ON MACHINE LEARNING

Note: In each of the problems given in the respective sessions, you are required to write the algorithm for the given problem, and map the steps of your algorithm with your python code, by using suitable comments. Also you are required to analyze the results obtained after the implementation of your code on respective dataset, and submit your analysis as a summary to each problem given in respective sessions.

Session 1:

- a) Write a Python Program to implement K-Nearest Neighbor Algorithm for data classification , choose dataset of your own choice.
- b) Write a Python Program to implement Naïve Bayes Algorithm for data classification , choose dataset of your own choice

Session 2:

- a) Write a Python Program to implement Decision Trees for data classification , choose data set of your own choice
- b) Write a Python Program to implement Logistic Regression for data classification , choose dataset of your own choice

Session 3:

- a) Write a Python Program to implement Support Vector Machines for data classification , choose dataset of your own choice
- b) Write a Python Program to implement Linear regression on a dataset of your own choice

Session 4:

- a) Write a Python Program to implement Support Vector Regression on a dataset of your own choice
- b) Write a Python Program to implement Artificial Neural Network for data classification , choose dataset of your own choice

Session 5:

- a) Write a Python Program to implement Apriori Algorithm on a dataset of your own choice
- b) Write a Python Program to implement FP tree growth Algorithm on a dataset of your own choice

Session 6:

- a) Write a Python Program to implement K-Means Algorithm on a dataset of your own choice
- b) Write a Python Program to implement DBSCAN Algorithm on a dataset of your own choice

Session 7:

- a) Assume that a diagnostic test is used to detect a rare disease in a population. The disease affects only 2% of the population. The test is fairly accurate: it correctly identifies 95% of those who actually have the disease (true positive rate), but it also has a false positive rate of 5%, meaning it incorrectly identifies 5% of healthy individuals as having the disease. A person is selected at random from the population and tests positive for the disease. Using Bayes' Theorem, write python program to calculate the probability that the person actually has the disease. In your response, outline the role of prior probability, likelihood, and evidence in your calculation, and python program should demonstrate why the result might be lower than expected despite the test's high accuracy.
- b) A dataset (<https://www.kaggle.com/datasets?tags=14201-Binary+Classification>) is given to you with numerical features and a binary target variable (0 or 1). Your task is to train an ensemble model using Python's scikit-learn library. Use a Random Forest Classifier to predict the target variable. Write python program to (i) Load the dataset into a DataFrame using pandas. (ii) Perform any necessary preprocessing (e.g., handling missing values, encoding categorical features if needed). (iii) Split the dataset into training and test sets. (iv) Train a RandomForestClassifier using scikit-learn. (v) Evaluate the performance of the trained model using accuracy score and confusion matrix. (vi) Print the feature importances to understand which features contributed the most to the predictions.

Session 8:

- a) Using Python programming language, your task is to develop a Reinforcement Learning agent to play a simple game using Deep Q-Learning (DQN). Using TensorFlow (or PyTorch), write a Python script that: (i) Initializes a neural network to approximate the Q-function. The network should have an input layer matching the state space dimension, a few hidden layers with ReLU activation, and an output layer matching the number of actions. (ii) Implements the experience replay buffer to store experiences in the form (state, action, reward, next_state, done). (iii) Sets up the training loop where the agent interacts with the environment (you can use a simple environment like CartPole-v1 from gym). (iv) At each step, select an action using an ϵ -greedy policy, update the experience replay buffer, and train the Q-network using mini-batches sampled from the buffer. (v) Plot the total rewards per episode to monitor training progress.
- b) Using Python programming language, you are assigned a task to develop a student performance prediction system for an educational institution. Using historical data, the institute wants to predict whether a student will pass or fail. Suppose, dataset contains various features such as attendance percentage, average assignment scores, number of disciplinary actions, participation in extra curriculum activities and examination marks. Using python programming and appropriate machine learning libraries, implement a classification model with multiple algorithms, including Logistic Regression, Decision Trees, K-Nearest Neighbors (K-NN), and Support Vector Machines (SVM). Implement the following: (i) Load and

preprocess the dataset using pandas. (ii) Split the data into training and testing sets. (iii) Train models using the four mentioned classification algorithms from scikit-learn. (iv) Evaluate model performance using accuracy, precision, recall, and F1-score. (v) Finally, visualize the confusion matrices for all models to compare their classification errors. Here, the final output should include a summary table comparing the performance metrics of all the models, and you should comment on which model you would recommend for deployment and why.

Session 9:

- a) Using Python language, develop and implement a Multilayer Feedforward Neural Network (MLFNN). The purpose of this experiment is to classify handwritten digits (0-9) from the MNIST dataset. The model will utilize the Sigmoid Activation Function in the hidden layers and the Backpropagation Algorithm for training. The experiment involves downloading the MNIST Handwritten Digits Dataset (<https://www.kaggle.com/datasets/hojjatk/mnist-dataset>), preprocessing the data, and building a neural network architecture consisting of an input layer with 784 neurons (28x28 image pixels), two hidden layers with 128 and 64 neurons respectively (both using Sigmoid Activation), and an output layer with 10 neurons (using Softmax Activation Function). The model will be trained using the Cross-Entropy Loss Function and optimized using Backpropagation. The model's performance will be evaluated using metrics such as accuracy, confusion matrix, and a classification report.
- b) Use Python language to implement the Apriori Algorithm for identifying frequent itemsets from a transactional dataset and generate association rules. The algorithm will analyze the dataset to find item combinations that frequently occur together, such as products purchased in a shopping cart. The experiment should demonstrate the concept of pattern search by setting minimum support and confidence thresholds. The performance will be evaluated by extracting meaningful association rules that can assist in decision-making, such as market basket analysis. The expected outcome is to generate frequent itemsets and their association rules using the Apriori Algorithm.

Session 10:

- a) Use Python language to apply Hierarchical, and DBSCAN Clustering Algorithms to group similar data points from an unlabelled dataset. The experiment aims to compare the clustering results and identify the most suitable algorithm based on cluster compactness and separation. The expected outcome is to visualize clusters and evaluate their performance using appropriate metrics.
- b) Use Python language to implement and demonstrate Classification, Regression, Clustering, and Association Rule Mining. The experiment aims to apply these algorithms on real-world datasets to observe their performance, compare results, and understand their practical applications. Consider any dataset and evaluate: Accuracy, Precision, Recall, F1-Score,

9 REFERENCES

- [1] “Machine learning,” *W3schools.com*. [Online]. Available: https://www.w3schools.com/python/python_ml_getting_started.asp. [Accessed: 12-Mar-2025].
- [2] K. F. Improve, “AI with Python tutorial,” *GeeksforGeeks*, 24-Jan-2024. [Online]. Available: <https://www.geeksforgeeks.org/python-ai/>. [Accessed: 12-Mar-2025].
- [3] *Udemy.com*. [Online]. Available: https://www.udemy.com/course/the-introduction-of-ai-and-machine-learning-with-python/?srsltid=AfmBOopPjmU_gX9iyaBEJJ6hPmKTdZPPV3_1t1a9Jo0i-2mq19UIId2F&couponCode=NVDIN35. [Accessed: 12-Mar-2025].
- [4] H. M. Elsherif, *Python for artificial intelligence: A comprehensive guide*. Eldonusa, 2024.
- [5] T. Carter, *Artificial intelligence with python: Building intelligent applications*. Independently Published, 2024.
- [6] A. Artasanchez and P. Joshi, *Artificial Intelligence with Python: Your complete guide to building intelligent apps using Python 3.x, 2nd Edition*, 2nd ed. Birmingham, England: Packt Publishing, 2020.
- [7] P. Xiao, *Artificial intelligence programming with python: From zero to hero*. Nashville, TN: John Wiley & Sons, 2022.
- [8] P. Fagundes, *Introduction to artificial intelligence with python: A practical learning journey*. Independently Published, 2024.
- [9] T. T. Teoh and Z. Rong, *Artificial intelligence with python*, 1st ed. Singapore, Singapore: Springer, 2023.
- [10] S. Lynch, *Python for scientific computing and artificial intelligence*. Philadelphia, PA: Chapman & Hall/CRC, 2023.
- [11] I. Pjp, *Learn all about Artificial intelligence using Python*. Independently Published, 2023.
- [12] P. Gupta and A. Bagchi, *Essentials of python for artificial intelligence and machine learning*, 1st ed. Cham, Switzerland: Springer International Publishing, 2024.
- [13] S. D’Ascoli, *Artificial intelligence through machine learning With python: Every line of code explained*. Independently Published, 2023.
- [14] P. Joshi, *Artificial Intelligence with Python*. Birmingham, England: Packt Publishing, 2017.
- [15] O. Adekanmbi, *Beginners’ artificial intelligence and python programming: For grades 4 to 8*. Independently Published, 2020.
- [16] G. Blokdyk, *Artificial intelligence with python complete self-assessment guide*. 5starcooks, 2018.