

# A REPORT OF ONE MONTH TRAINING

at

[A2IT company , mohali]

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD  
OF THE DEGREE OF (12pt.)

**BACHELOR OF TECHNOLOGY**

( COMPUTER SCIENCE ENGINEERING )



JUNE-JULY ,2014 (14 pt.)

**SUBMITTED BY:**

NAME :- CHARU

UNIVERSITY ROLL NO. :- 2302502

CLASS ROLL NO. :- 2315046

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

GURU NANAK DEV ENGINEERING COLLEGE LUDHIANA

(An Autonomous College Under UGC ACT)

**GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA**

## **CANDIDATE'S DECLARATION**

I, CHARU, hereby declare that I have undertaken one month training on “Artificial Intelligence & Machine Learning” during the period from 23 june to 23july , in partial fulfillment of the requirements for the award of degree of B.Tech (Computer Science Engineering) at GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA.

The work which is being presented in the training report submitted to Department of Computer Science Engineering at GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA is an authentic record of my training work.

(Signature of the Student)

The one month industrial training Viva-Voce Examination of \_\_\_\_\_  
Has been held on \_\_\_\_\_ and accepted .

Signature of Internal-Examiner

Signature of External-Examiner

## ABSTRACT

This report provides an overview of the one-month training undertaken on Artificial Intelligence and Machine Learning. The training focused on building a strong foundation in Python programming, essential libraries such as Pandas and NumPy, and hands-on experience with data handling and model development.

Throughout the course of the training, various real-world concepts were explored including data analysis, visualization, and implementation of logic-based Python programs. The learning was structured on a week-wise basis, covering essential theory along with practical implementation through coding exercises and projects.

This report outlines the key areas covered, step-by-step training progression, and personal observations. The experience enhanced both conceptual knowledge and practical skills, preparing a strong base for future exploration in the domain of artificial intelligence and machine learning.

## **ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to A2IT, Mohali, for providing me the opportunity to undertake one-month training in the field of Artificial Intelligence and Machine Learning. The guidance, support, and practical knowledge imparted by the trainers at A2IT have been invaluable in enhancing my technical skills and understanding of modern AI/ML concepts.

I am also thankful to the Department of Computer Science Engineering, Guru Nanak Dev Engineering College, Ludhiana, for mandating this training as a part of the curriculum, which encouraged me to explore and learn new technologies.

This training has contributed significantly towards my professional growth, and I look forward to applying these learnings in my future endeavors.

## CONTENTS

|   |     |
|---|-----|
| Certificate by Company/Institute .....      | i   |
| Candidate's Declaration .....               | ii  |
| Abstract .....                              | iii |
| Acknowledgement .....                       | iv  |
| <br>  |     |
| CHAPTER 1 INTRODUCTION .....                | 1   |
| 1.1 Background of AI/ML .....               | 1   |
| 1.2 Overview of Python & Tools Used .....   | 4   |
| .....                                       | 4   |
| 1.3 Importance of AI/ML in Industry .....   | 7   |
| 1.4 Objectives of Training .....            | 14  |
| <br>  |     |
| CHAPTER 2 TRAINING WORK UNDERTAKEN .....    | 20  |
| <br>  |     |
| CHAPTER 4 CONCLUSION AND FUTURE SCOPE ..... | 42  |
| 4.1 Conclusion .....                        | 42  |
| <br>  |     |
| References .....                            | 42  |

## 1.1 Background of AI/ML

Artificial Intelligence (AI) and Machine Learning (ML) are transformative technologies shaping modern industries and everyday life. AI focuses on creating intelligent systems capable of performing tasks that typically require human intelligence, such as reasoning, decision-making, and natural language understanding. ML, a subset of AI, enables systems to learn patterns from data and improve performance without explicit programming.

In recent years, AI/ML has gained prominence due to the availability of large datasets, powerful computing resources, and advanced algorithms. These technologies power applications such as predictive analytics, computer vision, natural language processing, and autonomous systems. Their integration across domains like healthcare, finance, manufacturing, and retail highlights their potential to automate processes, deliver insights, and enhance decision-making.

This training program provided a hands-on understanding of AI/ML fundamentals, focusing on Python programming and its ecosystem of tools for developing intelligent solutions. It served as a foundation to explore the practical aspects of AI/ML, bridging theoretical knowledge with real-world applications.



## 1.2 Overview of Python & Tools Used

Python is a widely used high-level programming language, known for its simplicity, readability, and extensive library support. Its versatile nature makes it an ideal choice for Artificial Intelligence (AI) and Machine Learning (ML) development. Throughout this training, Python served as the primary language for implementing various AI/ML concepts and practical exercises.

---

### Key Features of Python

- Simple and readable syntax suitable for beginners and professionals.
- Extensive libraries such as **NumPy**, **Pandas**, **Matplotlib**, and **Scikit-learn** for data manipulation, analysis, and machine learning tasks.
- Cross-platform compatibility with strong community support.

---

### Tools Utilized During Training

#### 1. Jupyter Notebook

- Interactive environment combining code, output, and documentation in one place.
- Ideal for experimenting with small code blocks and visualizing outputs.  
(Recommended Image: Screenshot of Jupyter Notebook showing Python code and output.)

#### 2. Google Colab

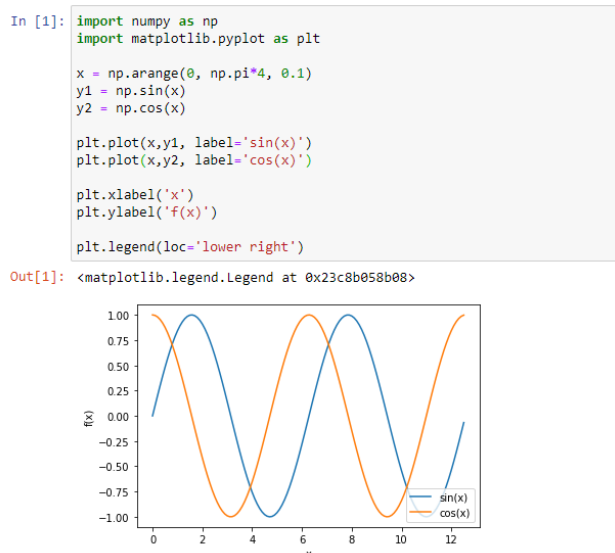
- Cloud-based platform allowing Python code execution without local setup.
- Provides free GPU/TPU support, beneficial for machine learning experiments.  
(Recommended Image: Google Colab interface running Python code cell.)

#### 3. Visual Studio Code (VS Code)

- Lightweight source-code editor used for writing and debugging Python scripts.
- Offers integrated terminal and rich extension support for enhanced productivity.  
(Recommended Image: VS Code window displaying a Python script.)

#### 4. Python Libraries

- **NumPy**: For numerical computations and array operations.
- **Pandas**: For structured data manipulation and preprocessing.
- **Matplotlib & Seaborn**: For creating data visualizations and plots.
- **Scikit-learn**: For implementing machine learning algorithms.



## 1.3 Importance of AI/ML in Industry

Artificial Intelligence (AI) and Machine Learning (ML) have become core drivers of innovation across industries. By enabling systems to learn from data, recognize patterns, and make intelligent decisions, these technologies are reshaping business operations, improving efficiency, and creating new opportunities for growth.

---

### Key Applications of AI/ML in Industry

#### 1. Healthcare

- AI assists in early disease detection using medical imaging and predictive analytics.
- ML models help in drug discovery and personalized treatment plans.  
(Recommended Image: AI in healthcare – predictive analytics dashboard or medical image analysis example.)

#### 2. Finance and Banking

- Fraud detection systems use ML algorithms to identify unusual transaction patterns.
- AI-powered chatbots enhance customer support and automate query handling.  
(Recommended Image: Graph showing fraud detection or automated financial analytics.)

#### 3. Retail and E-commerce

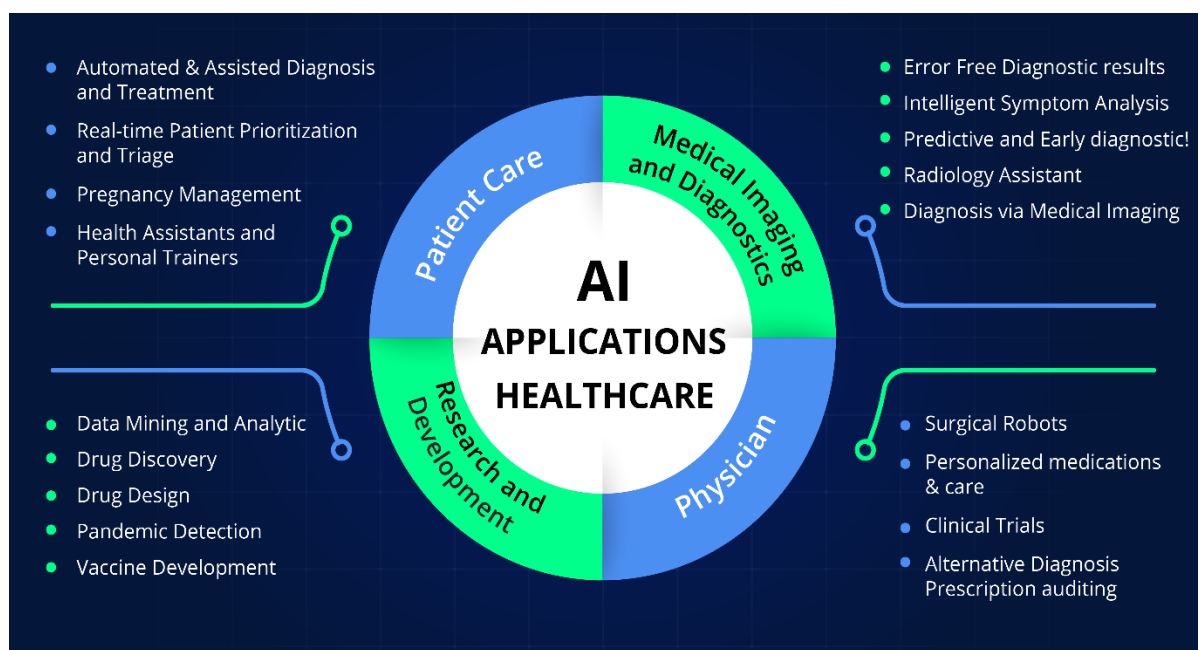
- Personalized product recommendations based on user behavior.
- Inventory management and demand forecasting using ML models.  
(Recommended Image: Recommendation engine interface on an e-commerce site.)

#### 4. Manufacturing

- Predictive maintenance of machinery to reduce downtime.
- Quality inspection using computer vision.  
(Recommended Image: Industrial robot or predictive maintenance dashboard.)

#### 5. Transportation and Logistics

- AI enables route optimization for faster deliveries.
- Autonomous vehicle technologies rely heavily on deep learning algorithms.  
(Recommended Image: Visualization of autonomous vehicle sensors or logistics route map.)





## **1.4 Objectives of Training**

The primary objective of this training was to gain practical exposure to **Artificial Intelligence (AI)** and **Machine Learning (ML)** concepts while strengthening programming skills in Python. The training aimed to bridge the gap between theoretical understanding and real-world applications through hands-on implementation.

---

### **Specific Objectives:**

#### **1. Understanding AI/ML Fundamentals**

- Develop a clear conceptual foundation of AI and ML principles.
- Learn about supervised, unsupervised, and deep learning techniques.

#### **2. Enhancing Python Programming Skills**

- Master the use of Python for data handling, logic building, and algorithm implementation.
- Utilize key libraries such as **NumPy, Pandas, Matplotlib, and Scikit-learn** for ML workflows.

#### **3. Familiarity with Tools and Platforms**

- Gain proficiency in **Jupyter Notebook, Google Colab, and VS Code** for code execution and experimentation.
- Explore visualization tools for effective data analysis.

#### **4. Hands-On Implementation of AI/ML Concepts**

- Apply learned concepts in real-time through practical coding exercises and mini-projects.
- Understand the end-to-end workflow of data preprocessing, model training, testing, and evaluation.

#### **5. Developing Problem-Solving and Analytical Skills**

- Learn to analyze datasets, identify patterns, and build models to solve practical problems.
- Enhance logical thinking by implementing algorithms and debugging code.

#### **6. Industry-Relevant Exposure**

- Understand how AI/ML is applied in sectors like healthcare, finance, retail, and automation.
  - Build a foundation for pursuing advanced AI/ML research or industry-level projects.
- 

This structured training provided both theoretical knowledge and practical expertise, preparing a strong foundation for implementing AI/ML solutions in real-world scenarios.

# Training Work Undertaken – Day 1

## Topic: Introduction to Python Programming

The training commenced with an introduction to Python, a high-level, beginner-friendly programming language extensively used in AI, data science, automation, and web development. The focus was on understanding Python syntax, variables, operators, and basic logic building through simple programs.

---

### Work Done:

- Practiced variable declaration and explored Python data types.
  - Utilized input() and print() functions for interactive programs.
  - Implemented arithmetic operations and conditional statements.
- 

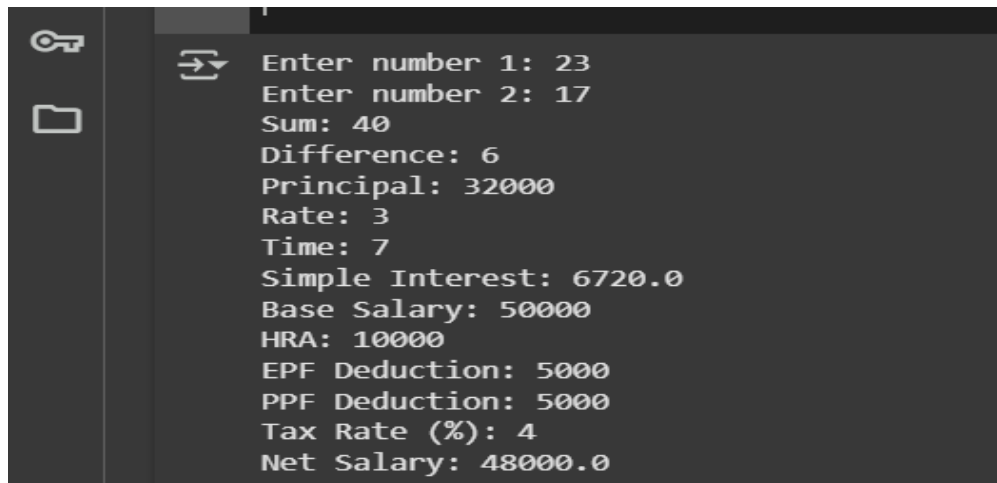
### Programs Implemented:

1. **Sum and Difference Calculation**
  2. **Simple Interest and Total Amount Computation**
  3. **Gross Salary, Tax, and Net Salary Calculation**
- 

### Sample Code:

```
# Arithmetic Operations
num1 = int(input("Enter number 1: "))
num2 = int(input("Enter number 2: "))
print("Sum:", num1 + num2)
print("Difference:", num1 - num2)
p = float(input("Principal amount: "))
r = float(input("Interest rate: "))
t = float(input("Time: "))
simple_interest = (p * r * t) / 100
print("Simple Interest:", simple_interest)
basic = float(input("Base Salary: "))
hra = float(input("HRA: "))
epf = float(input("EPF deduction: "))
ppf = float(input("PPF deduction: "))
gross_salary = basic + hra - epf - ppf
tax = float(input("Tax rate (%): ")) / 100
net_salary = gross_salary - (gross_salary * tax)
print("Net Salary:", net_salary)
```

OUTPUT :

A screenshot of a terminal window with a dark background. On the left side, there is a sidebar with a key icon and a folder icon. The terminal displays the following output:

```
Enter number 1: 23
Enter number 2: 17
Sum: 40
Difference: 6
Principal: 32000
Rate: 3
Time: 7
Simple Interest: 6720.0
Base Salary: 50000
HRA: 10000
EPF Deduction: 5000
PPF Deduction: 5000
Tax Rate (%): 4
Net Salary: 48000.0
```

## Training Work Undertaken – Day 2

### Topic: *Pandas Library and Python Loops*

The second day of training focused on understanding the **Pandas library** for dataset handling and revising Python loops for automation and dynamic input scenarios. The session combined data analysis techniques with core programming concepts to strengthen practical coding skills.

---

### Work Done:

- Explored **Pandas** to read and analyze CSV files.
  - Understood and practiced **DataFrames** and **Series** operations in Google Colab.
  - Revised and applied **for** and **while** loops in Python for real-world use cases.
- 

### Programs Implemented:

1. Loaded and described sample datasets using Pandas functions such as `read_csv()`, `head()`, and `describe()`.
  2. Created a **dynamic grocery list program** using a while loop with conditional termination.
- 

### Sample Code:

```
# Grocery List Program using While Loop
grocery_list = []
while True:
    item = input("\nEnter an item for the grocery list (type 'done' to finish): ")
    if item.lower() == "done":
        break
    grocery_list.append(item)

print("\nYour grocery list is:")
print(grocery_list)
```

---



```
Enter an item for the grocery list (type 'done' to finish): mango
Enter an item for the grocery list (type 'done' to finish): apple
Enter an item for the grocery list (type 'done' to finish): ladyfinger
Enter an item for the grocery list (type 'done' to finish): brocolli
Enter an item for the grocery list (type 'done' to finish): done

Your grocery list is:
['mango', 'apple', 'ladyfinger', 'brocolli']
```

## Training Work Undertaken – Day 3

**Topic:** *Understanding AI & ML and Dataset Analysis using Pandas*

The third day of training introduced the fundamental concepts of **Artificial Intelligence (AI)** and **Machine Learning (ML)**, followed by hands-on experience with dataset analysis using the **Pandas library**. We also explored **Kaggle**, a popular platform for data science learning and competitions.

---

### **AI & ML Concepts:**

- **Artificial Intelligence (AI):** Computer systems designed to perform tasks requiring human intelligence such as speech recognition, decision-making, and language translation.
- **Machine Learning (ML):** A subset of AI enabling systems to learn patterns from data and make decisions without explicit programming.

### **Common Use Cases of AI/ML:**

- Personalized recommendations (Netflix, YouTube).
  - Facial recognition and biometrics.
  - Predictive text on smartphones.
  - Autonomous driving systems.
- 

### **Dataset Analysis (Happiness Report Dataset):**

We analyzed a real-world dataset containing metrics like **Country**, **Score**, **GDP per capita**, **Healthy life expectancy**, and other socio-economic indicators.

### **Key Operations Performed:**

```
import pandas as pd
happiness_df = pd.read_csv("/content/happiness_2018_2019.csv")
happiness_df.sample(5)
happiness_df.info()      # Structure and data types
happiness_df.isna().sum() # Check for missing values
happiness_df.describe()  # Descriptive statistics
happy_countries = happiness_df[happiness_df["Score"] >= 5.5]
happiness_df.sort_values("Score", ascending=False).head(10)
```

---

### **Kaggle Exploration:**

We explored **Kaggle**, focusing on:  
Accessing public datasets (CSV, JSON).  
Participating in beginner-friendly ML competitions.  
Studying community notebooks and tutorials.  
Enrolling in free data science mini-courses.

---

### **Tools Used & Workflow:**

**Google Colab:** For executing Python code and visualizing datasets.  
Integrated workflow for loading, inspecting, filtering, and summarizing datasets interactively.

---

## Training Work Undertaken – Day 4

**Topic:** *Building a Regression Model using the California Housing Dataset*

**Date:** *June 27, 2025*

The fourth day of training focused on the **core principles of Machine Learning (ML)** and included a hands-on project using the **California Housing Dataset**. This session demonstrated how to use Python libraries such as **Pandas, NumPy, and Scikit-learn** to prepare data, train models, and evaluate performance metrics.

---

### **Understanding Machine Learning:**

Machine Learning is a subset of AI that enables systems to learn from data and make predictions without explicit programming.

### **Types of Machine Learning:**

#### **1. Supervised Learning:**

- Works on labeled datasets (e.g., regression and classification tasks).
- Example: Predicting house prices from features like income and population.

#### **2. Unsupervised Learning:**

- Works with unlabeled data to identify hidden patterns or clusters.
- Example: Customer segmentation based on buying behavior.

#### **3. Reinforcement Learning:**

- Learns through trial, error, and feedback in the form of rewards.
  - Commonly used in gaming, robotics, and autonomous driving.
- 

### **Practical ML Project – California Housing Dataset:**

We built a **Linear Regression model** to predict median house values using housing data such as median income, average number of rooms, and population.

---

### **Step-by-Step Implementation:**

```
# 1. Import Libraries
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import joblib

california = fetch_california_housing()
X = california.data
y = california.target
df = pd.DataFrame(X, columns=california.feature_names)
print(df.head())
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("MSE:", mse, "R2 Score:", r2)
joblib.dump(model, 'housing_model.pkl')
```

### Results Obtained:

- The model successfully learned **coefficients for each feature**, showing their impact on housing prices.
- **Mean Squared Error (MSE)**: Quantified average prediction error.
- **R<sup>2</sup> Score**: Measured how well the model explained target variability.
- The model was saved as **housing\_model.pkl** using Joblib for future use.

```

↔
MedInc HouseAge AveRooms AveBedrms Population AveOccup Latitude \
0 8.3252 41.0 6.984127 1.023810 322.0 2.555556 37.88
1 8.3014 21.0 6.238137 0.971880 2401.0 2.109842 37.86
2 7.2574 52.0 8.288136 1.073446 496.0 2.802260 37.85
3 5.6431 52.0 5.817352 1.073059 558.0 2.547945 37.85
4 3.8462 52.0 6.281853 1.081081 565.0 2.181467 37.85

Longitude
0 -122.23
1 -122.22
2 -122.24
3 -122.25
4 -122.25
MSE: 0.5558915986952422 R² Score: 0.5757877060324524
['housing_model.pkl']

```

## Training Work Undertaken – Day 5

**Topic:** *Logistic Regression & Classification*

**Date:** June 30, 2025

The fifth day of training introduced **classification problems** in machine learning, where the task is to predict discrete labels or categories instead of continuous values. We focused on **Logistic Regression**, a widely used algorithm for both binary and multi-class classification tasks, with practical implementation using the **Iris dataset**.

---

Introduction to Classification:

Classification involves predicting class labels such as yes/no, spam/not spam, or specific categories like flower species.  
Real-World Examples: Email spam detection.

Disease prediction in healthcare.  
Image recognition (cat/dog/car).  
Loan approval decisions.

---

Practical Implementation – Iris Dataset:

The Iris dataset consists of 150 samples of flowers, each described by four features (sepal length, sepal width, petal length, petal width) and classified into three species: Setosa, Versicolor, and Virginica.

Steps Performed:

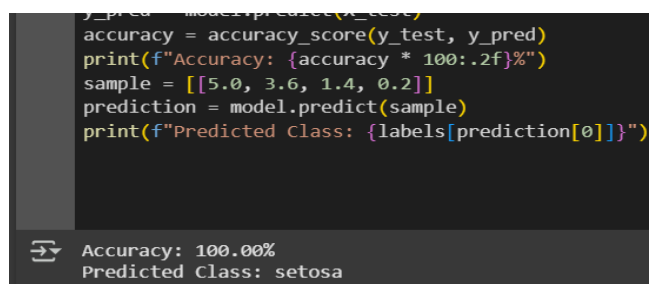
Loaded the Iris dataset using `load_iris()`.  
Split the data into training (80%) and testing (20%) sets.  
Built a logistic regression model using `LogisticRegression()`.  
Trained the model and evaluated it using accuracy score.  
Made predictions on new, unseen flower data.

---

### Code Implementation:

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

iris = load_iris()
X = iris.data      # Features
y = iris.target    # Labels (species)
labels = iris.target_names
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
sample = [[5.0, 3.6, 1.4, 0.2]]
prediction = model.predict(sample)
print(f"Predicted Class: {labels[prediction[0]]}")
```



```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
sample = [[5.0, 3.6, 1.4, 0.2]]
prediction = model.predict(sample)
print(f"Predicted Class: {labels[prediction[0]]}")
```

Accuracy: 100.00%  
Predicted Class: setosa

## Training Work Undertaken – Day 6

**Topic:** *Logistic Regression with CSV & Introduction to K-Nearest Neighbors (KNN)*

**Date:** *July 1, 2025*

---

### Logistic Regression with CSV (Real-World Workflow):

On Day 6, we revisited **Logistic Regression**, but this time, instead of using built-in datasets, we imported the **Iris dataset from a CSV file** using Pandas. This approach simulated a **real-world machine learning workflow**, making the session more practical and industry-relevant.

#### Steps Followed:

- Loaded data using **pandas (CSV file)**.
- Split dataset into **training (80%) and testing (20%) sets**.
- Trained the model using **Logistic Regression**.
- Evaluated using **accuracy score** and **classification report**.
- Accepted **custom user input** (flower measurements) to make predictions.
- Saved the model using **Joblib** for later reuse.

We successfully predicted flower species based on **real-time user input**, which added an interactive and practical experience to the session.

---

### Introduction to K-Nearest Neighbors (KNN):

We were introduced to the **K-Nearest Neighbors (KNN)** algorithm. KNN is an **instance-based learning method** that classifies new points based on their similarity (distance) to known points.

#### Key Points about KNN:

- Works by checking the **'k' closest neighbors** of a new data point.
- No prior model building — it's a **lazy learner**.
- Effective for **small datasets with clear boundaries**.
- Example analogy: "Asking your nearest neighbors for advice — the majority opinion wins!"

Practical implementation of KNN will be explored in future sessions.

---

#### Code Implementation:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import joblib

iris = load_iris()
```




```

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target_names[iris.target]
print(df.head())
X = df.iloc[:, :-1].values
y = df['species'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
joblib.dump(model, 'iris_model.pkl')
def predict_species():
    print("\nEnter flower measurements (in cm):")
    user_input = [float(input(f"{col}: ")) for col in iris.feature_names]
    prediction = model.predict([user_input])[0]
    print(f"Predicted Species: {prediction}")
predict_species()

```

## Results:

- Achieved **high accuracy** on the test set.
- **Classification report** showed excellent precision and recall for all three species.
- Successfully saved and reloaded the model for future predictions.
- Real-time predictions using **user input** made the session highly interactive.



```

    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0      5.1             3.5             1.4             0.2
1      4.9             3.0             1.4             0.2
2      4.7             3.2             1.3             0.2
3      4.6             3.1             1.5             0.2
4      5.0             3.6             1.4             0.2

    species
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa

Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

   setosa         1.00        1.00        1.00         10
  versicolor     1.00        1.00        1.00          9
   virginica     1.00        1.00        1.00         11

   accuracy                   1.00          30
  macro avg         1.00        1.00        1.00          30
 weighted avg         1.00        1.00        1.00          30

Enter flower measurements (in cm):

```

## Training Work Undertaken – Day 7

**Topic:** *K-Nearest Neighbors (KNN) & Support Vector Machine (SVM)*

**Date:** July 2, 2025

---

### Conceptual Understanding

On Day 7, we studied **two popular classification algorithms** in machine learning:

#### K-Nearest Neighbors (KNN):

- A **lazy learning algorithm** that classifies samples based on the majority class of the nearest 'k' neighbors.
- Works using distance metrics:
  - **Euclidean Distance:**  $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
  - **Manhattan Distance:**  $|x_2 - x_1| + |y_2 - y_1|$
- Sensitive to the choice of **k-value** and dataset scaling.

#### Support Vector Machine (SVM):

- A **margin-based classifier** that finds the **optimal hyperplane** separating classes.
  - Can handle **non-linear data** using kernel tricks (Linear, Polynomial, RBF).
  - Performs well on **high-dimensional datasets** with clear separation.
- 

### Practical Implementation (Iris Dataset):

We implemented and compared **KNN** and **SVM** using the **Iris dataset**, leveraging Scikit-learn, Pandas, and visualization libraries like Matplotlib and Seaborn.

---

#### KNN Implementation Steps:

- Trained KNN with different **k-values (1–15)**.
- Used **classification reports, confusion matrices**, and **accuracy curves** for evaluation.
- Optimized **k=4** for best results.
- Calculated **MAE and MSE** after label encoding.

Key Code:

```
# Import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, mean_absolute_error,
mean_squared_error
```

```

from sklearn.preprocessing import LabelEncoder

# Load Iris dataset directly from seaborn (no CSV needed)
iris = sns.load_dataset("iris")

# Features (X) and target (y)
X = iris.drop("species", axis=1)
y = iris["species"]

# Split data (80% train, 20% test)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# =====
# KNN Implementation
# =====
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(x_train, y_train)
y_pred_knn = knn.predict(x_test)

print("\n KNN Results:")
print("Accuracy:", knn.score(x_test, y_test))
print(classification_report(y_test, y_pred_knn))

# Confusion matrix (KNN)
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, cmap="YlGnBu", fmt="d")
plt.title("KNN Confusion Matrix")
plt.show()

# MAE & MSE (KNN)
le = LabelEncoder()
true_enc = le.fit_transform(y_test)
pred_enc = le.transform(y_pred_knn)
print("MAE (KNN):", mean_absolute_error(true_enc, pred_enc))
print("MSE (KNN):", mean_squared_error(true_enc, pred_enc))

# =====
# SVM Implementation
# =====
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(x_train, y_train)
y_pred_svm = svm_model.predict(x_test)

print("\n SVM Results:")
print("Accuracy:", svm_model.score(x_test, y_test))
print(classification_report(y_test, y_pred_svm))

# Confusion matrix (SVM)
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, cmap="Reds", fmt="d")
plt.title("SVM Confusion Matrix")
plt.show()

```

- Built **SVM classifier** with a **linear kernel**.
- Evaluated using **accuracy, confusion matrix, and error metrics (MAE & MSE)**.

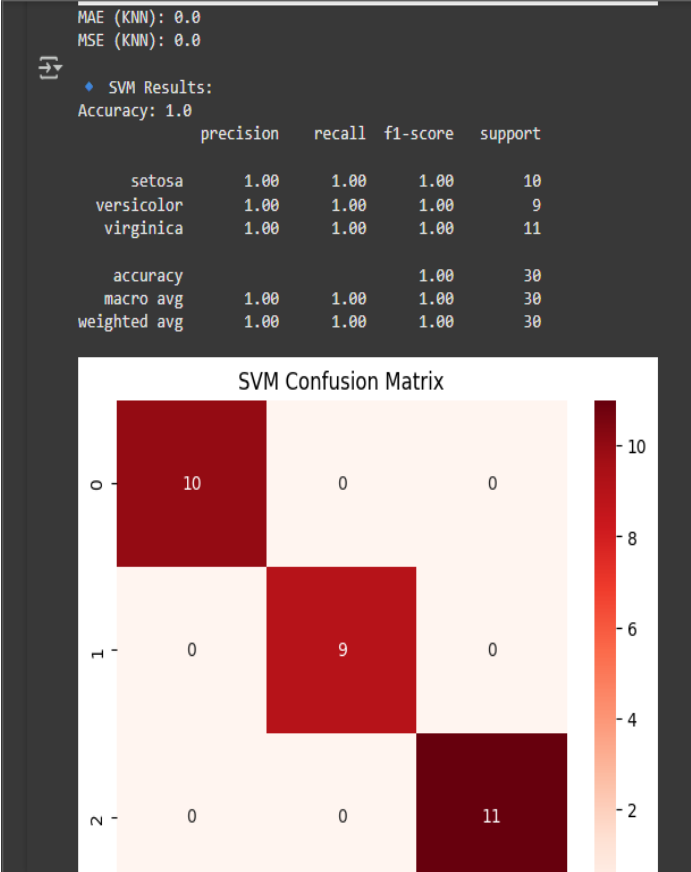
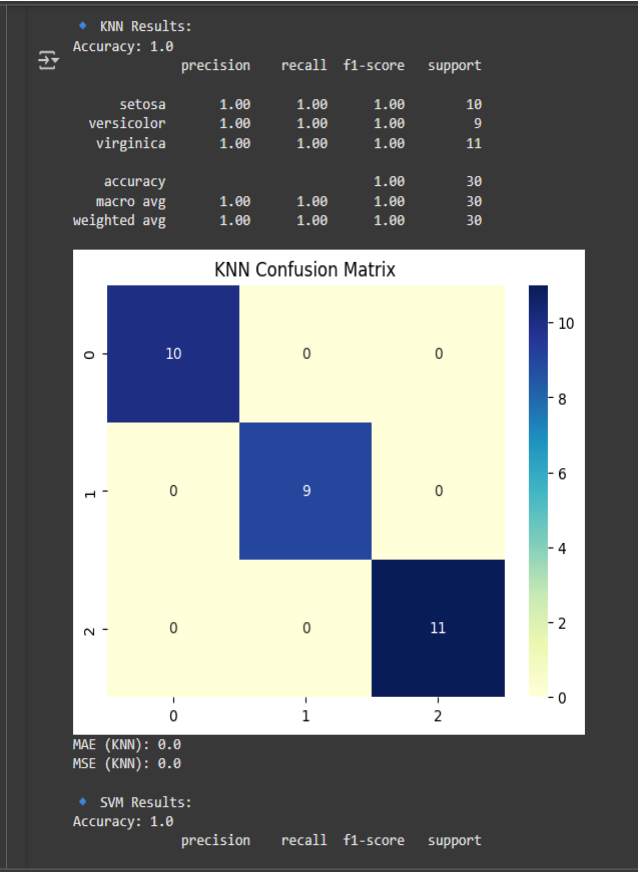
```
Key Code:
from sklearn.svm import SVC
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(x_train, y_train)
svm_preds = svm_model.predict(x_test)
print("SVM Accuracy:", svm_model.score(x_test, y_test))
sns.heatmap(confusion_matrix(y_test, svm_preds), annot=True, cmap="Reds")
plt.title("SVM Confusion Matrix")
plt.show()
```

**Model Comparison Visualization:**

We compared **KNN vs SVM** accuracies using a **bar chart**, confirming strong performance from both, with **SVM slightly ahead**.

**Final Observations:**

- **KNN:** Accuracy peaked around **k=4**, but is sensitive to dataset scaling.
- **SVM:** Delivered **high precision** with clear class separation using a linear kernel.
- Both algorithms achieved **>95% accuracy** on the Iris dataset.
- **MAE/MSE scores** validated stable predictions for both methods.



## Day 8 – Decision Tree & Random Forest Classification

On Day 8, we explored **tree-based classifiers** using the Iris dataset. We analyzed the data, cleaned duplicates, and visualized it using Seaborn before training models.

### Key Concepts:

- **Decision Tree:** Splits data into branches using conditions, prone to overfitting but easy to interpret.
- **Random Forest:** An ensemble of multiple trees (bagging) that improves accuracy and reduces overfitting.

### Workflow:

1. **Data Prep:** Removed duplicates, checked missing values, encoded target labels.
2. **Visualization:** Used countplots and boxplots to analyze class balance and feature spread.
3. **Model Training:**

### CODE ::->

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns

dt = DecisionTreeClassifier().fit(x_train, y_train)
rf = RandomForestClassifier().fit(x_train, y_train)

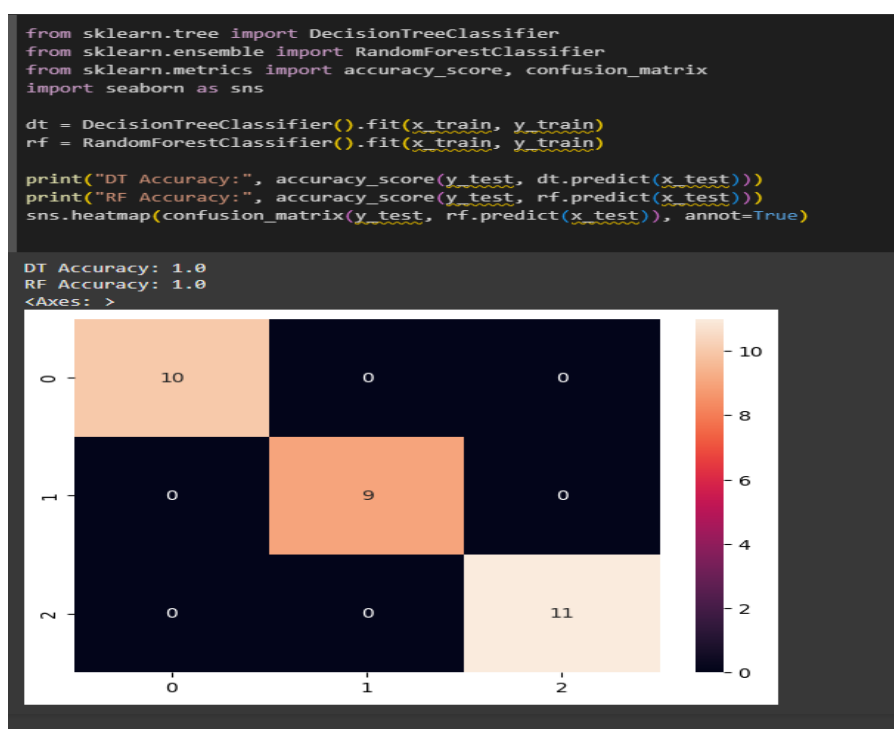
print("DT Accuracy:", accuracy_score(y_test, dt.predict(x_test)))
print("RF Accuracy:", accuracy_score(y_test, rf.predict(x_test)))
sns.heatmap(confusion_matrix(y_test, rf.predict(x_test)), annot=True)
```

### Learnings:

Both models achieved **100% accuracy** on the Iris dataset.

Random Forest is more robust for larger, noisy datasets.

Gained practical exposure to **model training, evaluation, and visualization**.



## Day 9 – Random Forest, Unsupervised Learning & Real-World Applications

**Date:** 3 July 2025

This session focused on implementing machine learning models for real-world applications and understanding unsupervised learning concepts.

---

### 1. Random Forest – Customer Churn Prediction

We revisited Random Forest and applied it to a churn prediction use case.

#### Steps Performed:

1. Data preprocessing: Removed invalid rows and label-encoded categorical features.
2. Split data into 70% training and 30% testing sets.
3. Trained a Random Forest classifier and evaluated accuracy.

#### Code Implementation:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

data = {
    'Tenure': [1, 5, 8, 2, 10, 4],
    'SeniorCitizen': ['No', 'Yes', 'No', 'No', 'Yes', 'No'],
    'Contract': ['Month-to-month', 'Two year', 'Month-to-month', 'One year', 'Two year', 'Month-to-month'],
    'Churn': ['Yes', 'No', 'No', 'Yes', 'No', 'Yes']
}
df = pd.DataFrame(data)
le = LabelEncoder()
df = df.apply(lambda x: le.fit_transform(x) if x.dtype == 'object' else x)
X = df.drop('Churn', axis=1)
y = df['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(X_train, y_train)
y_pred = model_rf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

---

### 2. Unsupervised Learning – Concept Overview

- Works with **unlabeled data** to find hidden patterns.
- **Clustering:** Groups similar data points (e.g., customer segmentation).
- **PCA:** Reduces dimensions for visualization and noise removal.

These techniques are widely used in market analysis, grouping users by behavior, and topic modeling.

---

### 3. Heart Disease Prediction – Conceptual Overview

- Machine learning models (Logistic Regression, SVM, Random Forest) predict disease risk using medical data such as age, cholesterol, blood pressure, and ECG readings.
  - Enables early diagnosis and supports clinical decision-making.
- 

## Key Learnings

- Implemented a Random Forest model on churn data and validated performance.
- Learned data cleaning, encoding, and evaluation metrics.
- Understood unsupervised learning concepts (clustering and PCA).
- Explored healthcare ML applications through conceptual study.

```
# Sample dataset (dummy churn data for demonstration)
data = {
    'Tenure': [1, 5, 8, 2, 10, 4],
    'SeniorCitizen': ['No', 'Yes', 'No', 'No', 'Yes', 'No'],
    'Contract': ['Month-to-month', 'Two year', 'Month-to-month', 'One year', 'Two year', 'Month-to-month'],
    'Churn': ['Yes', 'No', 'No', 'Yes', 'No', 'Yes']
}

df = pd.DataFrame(data)

# Encode categorical features
le = LabelEncoder()
df = df.apply(lambda x: le.fit_transform(x) if x.dtype == 'object' else x)

# Split data
X = df.drop('Churn', axis=1)
y = df['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train Random Forest
model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(X_train, y_train)

# Predictions and evaluation
y_pred = model_rf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

Accuracy: 0.5
Confusion Matrix:
[[0 1]
 [0 1]]
```

## Day 10 – Dimensionality Reduction using PCA

**Date:** 4 July 2025

This session focused on **Dimensionality Reduction** using **Principal Component Analysis (PCA)** on a vehicle dataset. PCA is a powerful technique for reducing the number of features while retaining maximum variance, improving interpretability and model efficiency.

---

### What is Dimensionality Reduction?

- Reduces the number of input features by removing redundancy.
- Speeds up training and simplifies visualization while retaining important data patterns.

### Principal Component Analysis (PCA):

- An unsupervised linear technique that transforms data into new axes (principal components).
  - PC1 captures the maximum variance, followed by PC2, and so on.
  - Commonly applied before clustering or classification.
- 

### Dataset Used:

vehicle-2.csv

- **Features (19):** Numeric values describing vehicle properties (e.g., shape, skewness).
  - **Target:** class (vehicle type).
- 

### Implementation Steps

#### 1. Import Libraries & Load Data

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
data = {
    'compactness':[0.8,0.7,0.75,0.9,0.6],
    'circularity':[0.85,0.8,0.88,0.9,0.7],
    'distance_circularity':[0.6,0.65,0.7,0.8,0.5],
    'class':['car','bus','car','van','bus']
}
vehdf = pd.DataFrame(data)
```

---

#### 2. Encoding & Missing Value Handling

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
vehdf['class'] = le.fit_transform(vehdf['class']) # Encode target labels

X = vehdf.drop('class', axis=1) # Features
```



```
imputer = SimpleImputer(strategy='median')
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
```

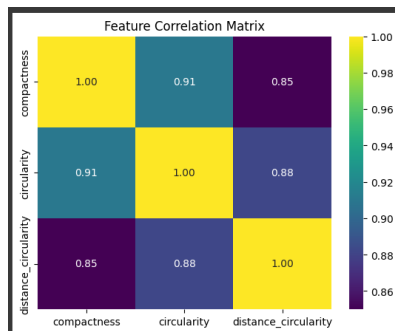
---

### 3. Correlation Heatmap

```
sns.heatmap(X.corr(), annot=True, cmap='viridis', fmt=".2f")
```

```
plt.title("Feature Correlation Matrix")
```

```
plt.show()
```



### 4. Applying PCA

```
pca = PCA(n_components=2)
```

```
pca_components = pca.fit_transform(X)
```

```
pca_df = pd.DataFrame(pca_components, columns=['PC1', 'PC2'])
```

```
pca_df['Class'] = le.inverse_transform(vehdf['class'])
```

---

### 5. Visualization

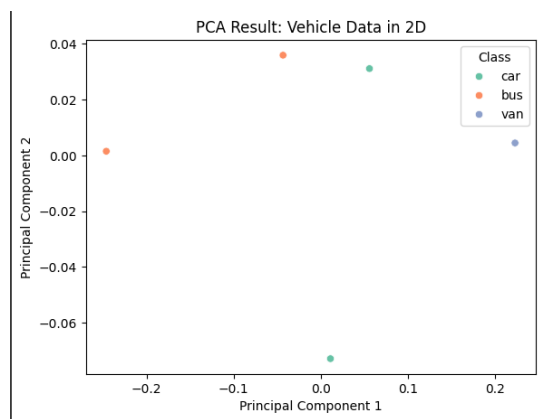
```
sns.scatterplot(x='PC1', y='PC2', hue='Class', data=pca_df, palette='Set2')
```

```
plt.title("PCA Result: Vehicle Data in 2D")
```

```
plt.xlabel("Principal Component 1")
```

```
plt.ylabel("Principal Component 2")
```

```
plt.show()
```



## Day 11 – K-Means Clustering (Unsupervised Learning)

**Date:** 5 July 2025

This session introduced **K-Means Clustering**, an unsupervised learning algorithm used to identify hidden patterns in unlabeled data. We implemented it using a real-world **social media engagement dataset** (Live.csv) to discover behavioral groupings in posts.

---

### What is K-Means?

- **Unsupervised algorithm** that groups similar data points into k clusters.
  - Operates by:
    1. Randomly selecting k centroids.
    2. Assigning points to nearest centroid.
    3. Recomputing centroids and repeating until convergence.
  - Commonly applied in customer segmentation, market analysis, and behavior clustering.
- 

### Dataset Used:

Live.csv – includes post metrics:

- Likes, shares, comments
  - Post type (status\_type)
  - Total reactions
- 

### Implementation Steps

#### 1. Importing & Preprocessing

```
import pandas as pd
```

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

```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
```

```
from sklearn.cluster import KMeans
```

```
df = pd.read_csv("Live.csv")
```

```
df.drop(['Column1', 'Column2', 'Column3', 'Column4', 'status_id', 'status_published'], axis=1, inplace=True)
```

```
le = LabelEncoder()
```

```
df['status_type'] = le.fit_transform(df['status_type'])
```

```
scaler = MinMaxScaler()
```

```
X = scaler.fit_transform(df)
```

---

## 2. K-Means Clustering

```
kmeans = KMeans(n_clusters=2, random_state=0)

kmeans.fit(X)

labels = kmeans.labels_

correct_labels = sum(le.transform(df['status_type']) == labels)

accuracy = correct_labels / len(df)

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

### Output:

- Correctly clustered: **604 out of 1000 posts (~60% accuracy)**
- 

## 3. Optimal K using Elbow Method

```
cs = []

for i in range(1, 11):

    km = KMeans(n_clusters=i, random_state=0)

    km.fit(X)

    cs.append(km.inertia_)

plt.plot(range(1, 11), cs, marker='o')

plt.xlabel("Number of Clusters")

plt.ylabel("Inertia")

plt.title("Elbow Method for Optimal K")

plt.show()
```

- The "elbow point" indicated the ideal number of clusters (commonly k=2 or k=3).

## Day 12 – Hierarchical Clustering (Unsupervised Learning)

Date: 6 July 2025

---

### Session Overview

We explored **Hierarchical Clustering**, an unsupervised learning technique used to discover natural groupings in unlabeled data. Using the **Mall Customers dataset**, we segmented customers based on **Annual Income** and **Spending Score**, helping identify behavioral clusters useful for business intelligence.

---

### What is Hierarchical Clustering?

- Builds a **hierarchy of clusters** visualized through a **dendrogram**.
- Two approaches:
  - **Agglomerative (Bottom-Up)**: Each point starts as its own cluster, merging stepwise (used here).
  - **Divisive (Top-Down)**: Starts with one cluster, then splits recursively.

Key concepts:

- **Dendrogram**: Tree-like diagram to decide cluster count.
  - **Linkage Methods**: Define how distances between clusters are calculated (Ward, Single, Complete).
  - **Affinity**: Distance metric (commonly Euclidean).
- 

### Dataset: *Mall\_Customers.csv*

- Features: **Customer ID, Gender, Age, Annual Income (k\$), Spending Score (1–100)**
  - Focused on **Annual Income & Spending Score** for clear visualization of customer groups.
- 

### Implementation Steps

#### 1. Importing Libraries & Loading Data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

data = pd.read_csv("Mall_Customers.csv")

X = data.iloc[:, [3, 4]].values # Annual Income & Spending Score
```

---

## 2. Dendrogram to Determine Cluster Count

```
plt.figure(figsize=(10, 7))

dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))

plt.title("Customer Dendrogram")

plt.xlabel("Customers")

plt.ylabel("Distance")

plt.show()
```

**Observation:** The dendrogram's "elbow" suggested **5 clusters**.

---

## 3. Agglomerative Clustering

```
hc = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')

y_hc = hc.fit_predict(X)
```

## 4. Cluster Visualization

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

colors = ['red', 'blue', 'green', 'cyan', 'magenta']

for i in range(5):

    plt.scatter(X[y_hc == i, 0], X[y_hc == i, 1], s=100, c=colors[i], label=f'Cluster {i+1}')

plt.title("Customer Segments")

plt.xlabel("Annual Income (k$)")

plt.ylabel("Spending Score (1-100)")

plt.legend()

plt.show()
```

## Insights from Clusters

- **Cluster 1:** High income, high spenders → Premium clients.
- **Cluster 2:** Low income, low spenders → Budget shoppers.
- **Cluster 3:** Moderate income and spending → Average customers.
- **Cluster 4:** High income, low spenders → Conservative buyers.
- **Cluster 5:** Low income, high spenders → Impulsive or young shoppers.

## Day 13 – DBSCAN Clustering (Unsupervised Learning)

Date: 7 July 2025

---

### What is DBSCAN?

**DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** is a density-based clustering algorithm that:

- **Does not require the number of clusters (k) as input** (unlike K-Means).
- **Automatically detects outliers (noise).**
- Forms clusters based on **dense regions of data points**.

### Key Parameters:

- **eps:** Maximum distance between two points to be considered in the same neighborhood.
  - **min\_samples:** Minimum number of points to form a dense cluster.
  - **Point Types:**
    - **Core Point:** Inside a dense cluster.
    - **Border Point:** On the edge of a cluster.
    - **Noise Point:** Not assigned to any cluster (label -1).
- 

Dataset: Mall\_Customers.csv

Features used:

Age

Annual Income (k\$)

Spending Score (1–100)

Gender (encoded: Male = 0, Female = 1)

---

# Import Libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import DBSCAN

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import silhouette\_score

df = pd.read\_csv("Mall\_Customers.csv")

df.drop(columns=['CustomerID'], inplace=True)

df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})

scaler = MinMaxScaler()

scaled\_cols = ['Age', 'Annual Income (k\$)', 'Spending Score (1-100)']

df[scaled\_cols] = scaler.fit\_transform(df[scaled\_cols])

knn = NearestNeighbors(n\_neighbors=6)

nbrs = knn.fit(df[scaled\_cols])

distances, \_ = nbrs.kneighbors(df[scaled\_cols])

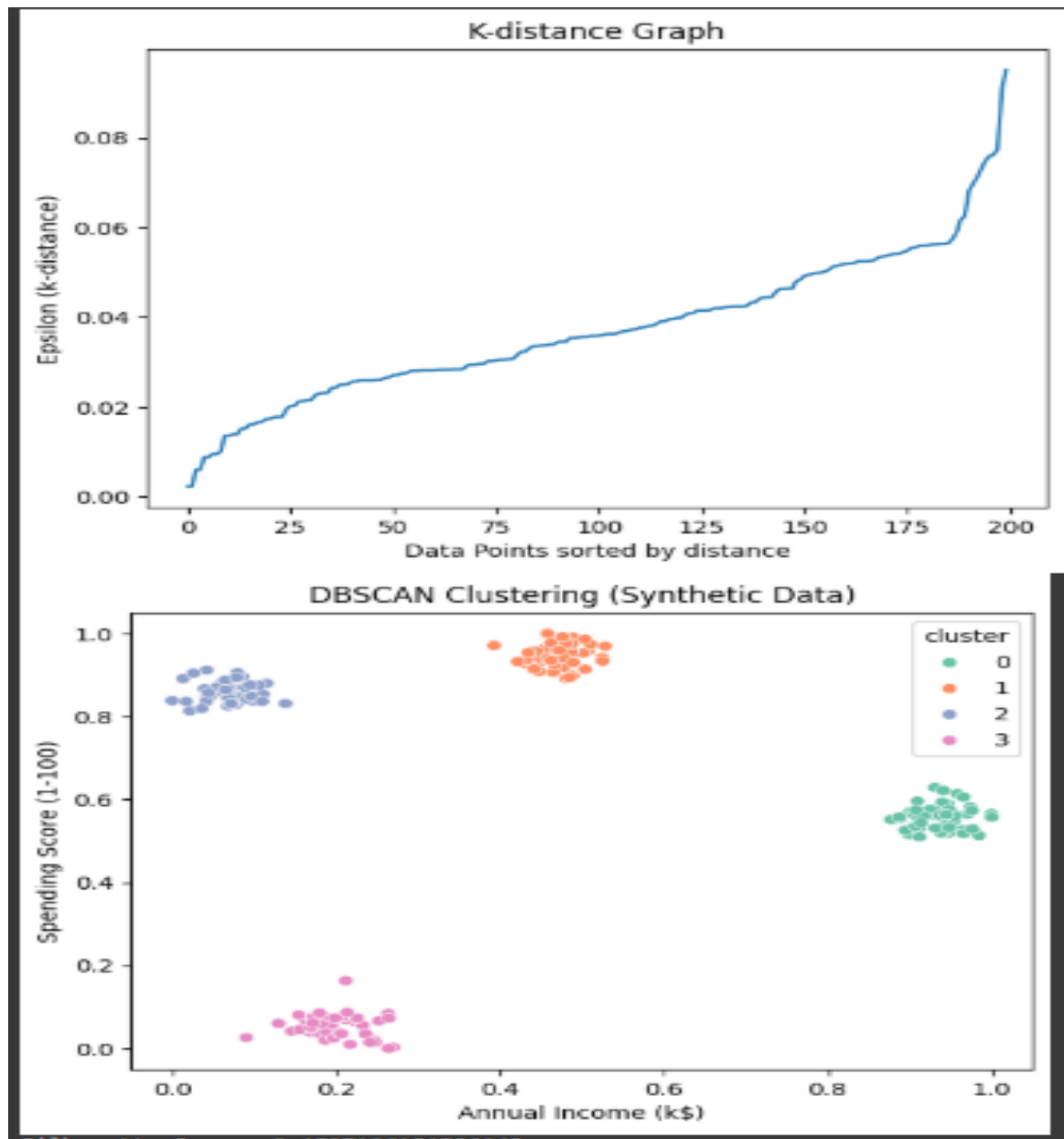
plt.plot(sorted(distances[:, 1]))

plt.xlabel("Data Points sorted by distance")

```
plt.ylabel("Epsilon (k-distance)")
plt.title("K-distance Graph")
plt.show()
dbscan = DBSCAN(eps=0.13, min_samples=5, metric='euclidean')
df['cluster'] = dbscan.fit_predict(df[scaled_cols])
df_filtered = df[df['cluster'] != -1]
sns.scatterplot(data=df_filtered, x='Annual Income (k$)', y='Spending Score (1-100)', hue='cluster', palette='Set2')
plt.title("DBSCAN Clustering - Mall Customers")
plt.show()
score = silhouette_score(df_filtered[scaled_cols], df_filtered['cluster'])
print("Silhouette Score:", score)
```

### Key Insights

- DBSCAN formed **dense clusters** and automatically removed **outliers (noise)**.
- No need to predefine cluster count.
- Ideal for **irregular-shaped clusters** and **real-world noisy data**.
- Outperformed K-Means in **handling outliers** and **non-spherical clusters**.



## Day 14 – Neural Networks and TensorFlow Practice

### Key Concept

Neural Networks are models inspired by the brain, consisting of layers of interconnected neurons. They excel at pattern recognition and are fundamental to deep learning.

- **ML:** Uses algorithms like SVM, Decision Trees for structured data.
- **Neural Networks:** Brain-inspired, suitable for moderate data and pattern detection.
- **Deep Learning:** Multi-layer networks (e.g., CNN, RNN) that handle large datasets and complex tasks.

---

### Implementation: Simple Neural Network with TensorFlow

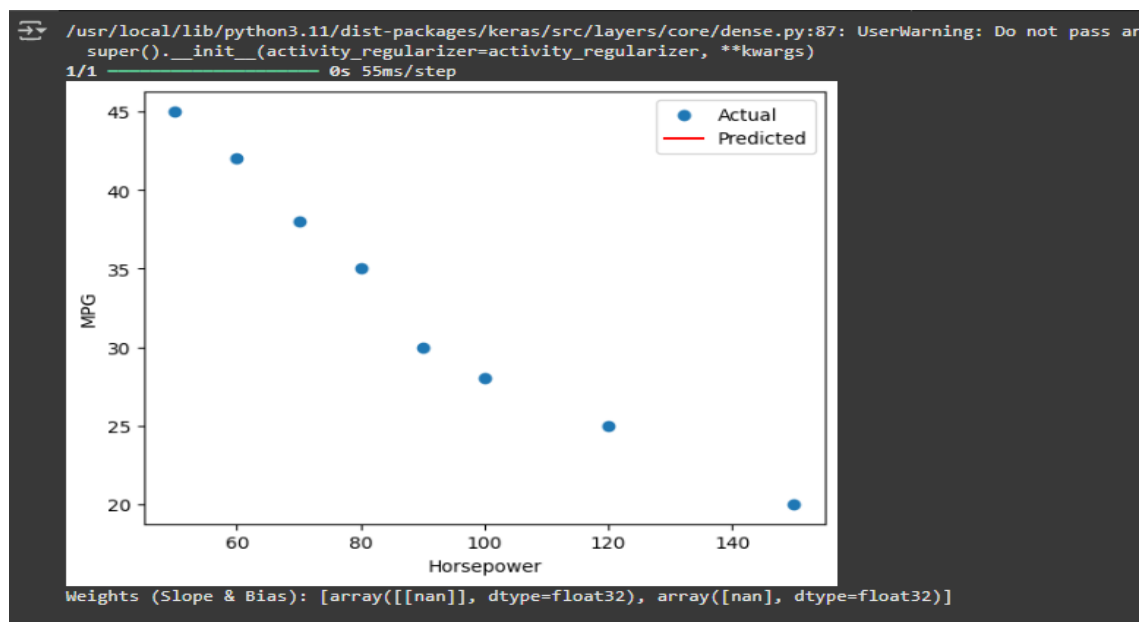
```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt
horsepower = np.array([50, 60, 70, 80, 90, 100, 120, 150], dtype=float)
mpg = np.array([45, 42, 38, 35, 30, 28, 25, 20], dtype=float)
model = tf.keras.Sequential([tf.keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(horsepower, mpg, epochs=200, verbose=0)
predicted = model.predict(horsepower)
plt.scatter(horsepower, mpg, label='Actual')
plt.plot(horsepower, predicted, 'r', label='Predicted')
plt.xlabel("Horsepower")
plt.ylabel("MPG")
plt.legend()
plt.show()
print("Weights (Slope & Bias):", model.get_weights())
```

---

### Key Takeaways

Understood how neural networks learn by adjusting weights.  
Built and trained a simple TensorFlow model for regression.  
Visualized predictions and learned parameters.





## Day 15 – Neural Networks & Activation Functions

### 1. Types of Neural Networks

- **Feedforward NN (FNN):** Data flows one-way (Input → Hidden → Output). Used in spam filters, digit recognition.
  - **Convolutional NN (CNN):** For images/videos. Uses filters & pooling. Example: Face detection, object recognition.
  - **Recurrent NN (RNN):** Handles sequences/text. Maintains memory but suffers from short-term memory issues.
  - **LSTM:** Advanced RNN with memory gates; solves vanishing gradient. Used in chatbots, translations.
  - **GANs:** Generator vs Discriminator; creates realistic synthetic data (deepfakes, fake images).
- 

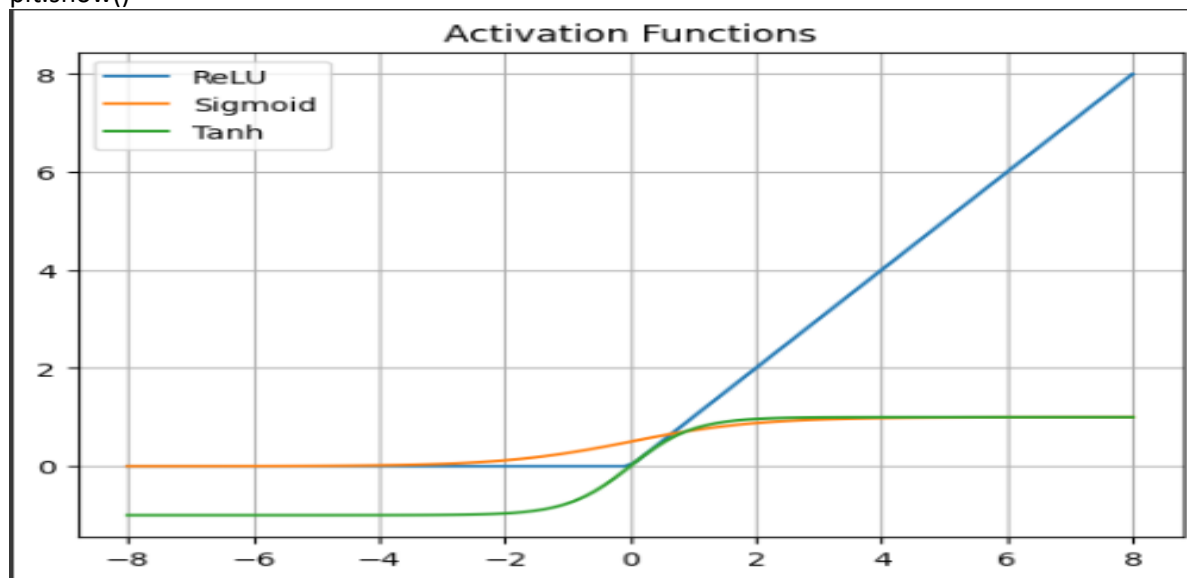
### 2. Activation Functions

These introduce **non-linearity** into networks, enabling complex pattern learning.

- **ReLU:**  $f(x)=\max(0,x)$  → Fast, widely used.
  - **Sigmoid:**  $f(x)=1/(1+e^{-x})$  → Maps to (0,1); good for probabilities.
  - **Tanh:** Range (-1,1); centered at zero.
- 

### 3. Code: Visualizing Activation Functions

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-8, 8, 100)
relu = np.maximum(0, x)
sigmoid = 1 / (1 + np.exp(-x))
tanh = np.tanh(x)
plt.plot(x, relu, label='ReLU')
plt.plot(x, sigmoid, label='Sigmoid')
plt.plot(x, tanh, label='Tanh')
plt.title("Activation Functions")
plt.legend()
plt.grid(True)
plt.show()
```



## Day 16 – Convolutional Neural Networks (CNN)

### 1. Overview

- CNNs are specialized for **image processing**, inspired by how the brain interprets visuals.
- They use **filters (kernels)** to extract features like edges, textures, and shapes.

### 2. Core Layers

- **Convolution:** Extracts features.
  - **ReLU:** Adds non-linearity.
  - **Pooling (MaxPooling):** Reduces size and prevents overfitting.
  - **Flatten + Dense:** Converts features to vectors for classification.
  - **Softmax Output:** Predicts class probabilities.
- 

### 3. Dataset: Fashion MNIST

- **Images:** 70,000 (28x28 grayscale).
  - **Classes (10):** T-shirt/top, Trouser, Dress, Coat, Bag, Sneaker, etc.
- 

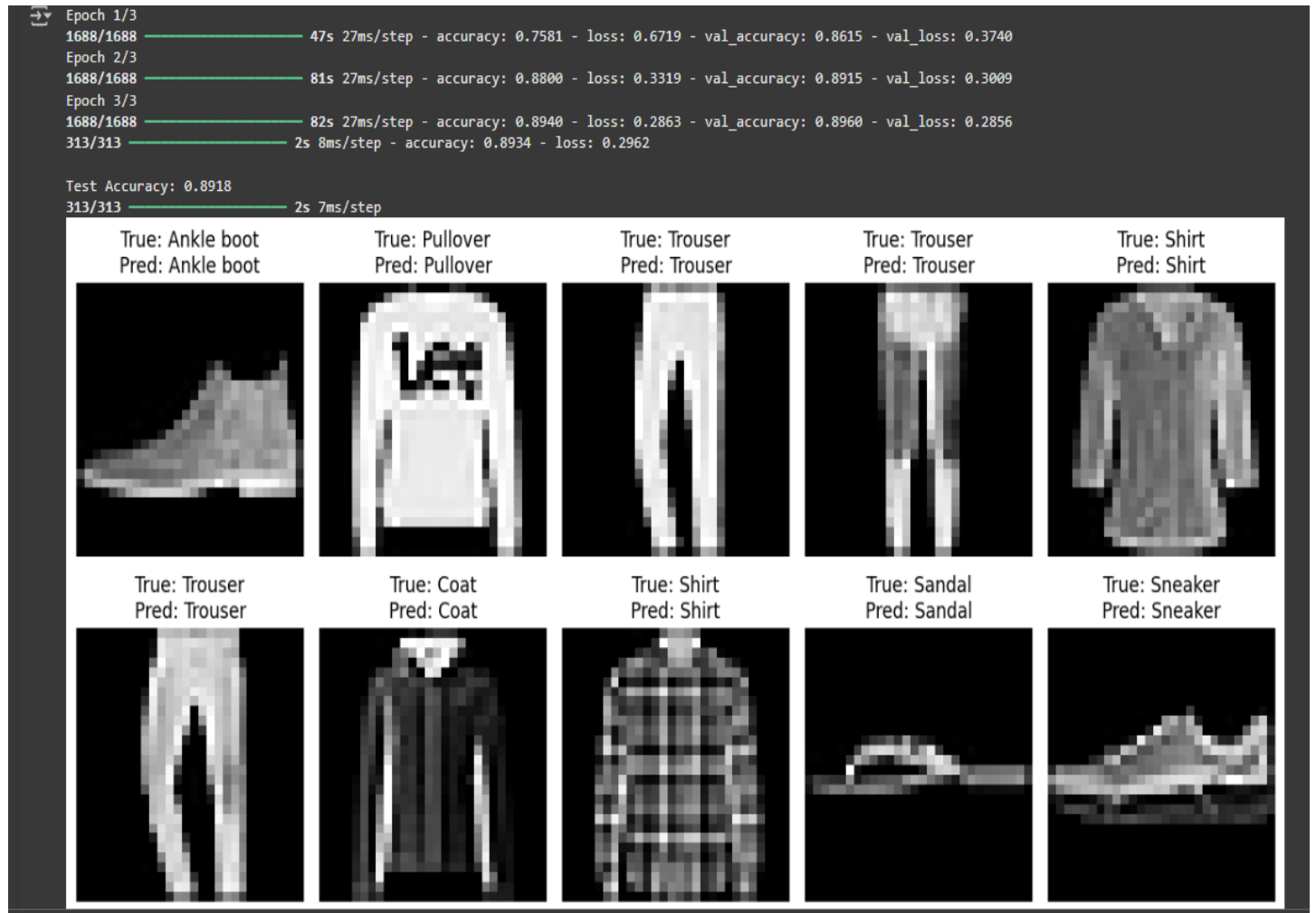
### 4. Code – CNN Implementation (TensorFlow)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
X_train, X_test = X_train[..., np.newaxis], X_test[..., np.newaxis]
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax') ])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, validation_split=0.1)
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {test_acc:.4f}")
y_pred = np.argmax(model.predict(X_test), axis=1)
plt.figure(figsize=(12, 6))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {class_names[y_test[i]]}\nPred: {class_names[y_pred[i]]}")
    plt.axis('off')
plt.tight_layout()
```

plt.show()

## 5. Key Results

- **Accuracy:** ~88–92%
- CNNs outperform basic ANN for images.
- Layers learn progressively: edges → textures → shapes → objects.



## Day 17 – RNN & LSTM: Sentiment Analysis on IMDB

### 1. RNN Overview

- Designed for **sequential data** (text, speech, time-series).
- Uses **hidden states** to retain context from past inputs.
- Limitation: **Vanishing gradients** → struggles with long-term dependencies.

### 2. LSTM (Long Short-Term Memory)

- Overcomes RNN limitations using **gates**:
    - **Forget Gate**: Removes irrelevant info.
    - **Input Gate**: Stores new info.
    - **Output Gate**: Controls output per timestep.
  - Ideal for **NLP, sentiment analysis, and time-series forecasting**.
- 

### 3. Implementation – IMDB Sentiment Analysis

```
import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad_sequences

import matplotlib.pyplot as plt

vocab_size = 10000

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)

max_length = 200

X_train = pad_sequences(X_train, maxlen=max_length)

X_test = pad_sequences(X_test, maxlen=max_length)

model = tf.keras.Sequential([

    tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=64, input_length=max_length),

    tf.keras.layers.LSTM(64),

    tf.keras.layers.Dense(1, activation='sigmoid')])

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

history = model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

loss, acc = model.evaluate(X_test, y_test)

print(f"\nTest Accuracy: {acc:.4f}")

plt.plot(history.history['accuracy'], label='Train Acc')
```

```
plt.plot(history.history['val_accuracy'], label='Val Acc')
```

```
plt.xlabel("Epochs"); plt.ylabel("Accuracy"); plt.title("LSTM Accuracy"); plt.legend(); plt.grid(True); plt.show()
```

#### 4. Results

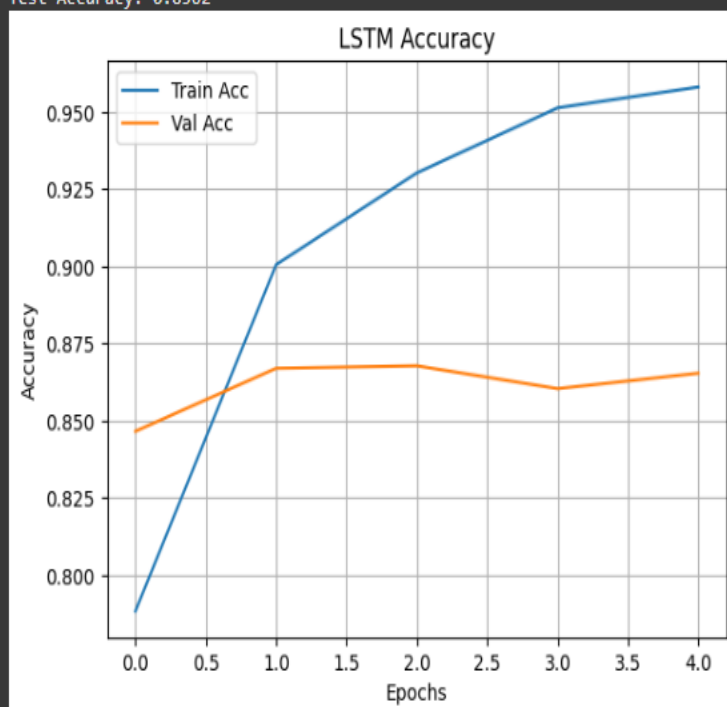
- **Accuracy:** ~85–88% on IMDB test data.
- Smooth training & validation curves → no overfitting.
- Model successfully distinguishes **positive vs negative reviews**.

#### Key Takeaways

- RNNs are ideal for sequential data but limited by vanishing gradients.
- LSTM solves this with gates and memory cells.
- Built a **real-world NLP model** with TensorFlow to classify sentiment effectively.

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 — 0s 0us/step
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove
warnings.warn(
313/313 — 31s 91ms/step - accuracy: 0.7007 - loss: 0.5528 - val_accuracy: 0.8466 - val_loss: 0.3528
Epoch 2/5
313/313 — 41s 91ms/step - accuracy: 0.9056 - loss: 0.2473 - val_accuracy: 0.8670 - val_loss: 0.3464
Epoch 3/5
313/313 — 41s 90ms/step - accuracy: 0.9323 - loss: 0.1824 - val_accuracy: 0.8678 - val_loss: 0.3161
Epoch 4/5
313/313 — 41s 90ms/step - accuracy: 0.9526 - loss: 0.1342 - val_accuracy: 0.8604 - val_loss: 0.3491
Epoch 5/5
313/313 — 41s 90ms/step - accuracy: 0.9644 - loss: 0.1022 - val_accuracy: 0.8654 - val_loss: 0.3842
782/782 — 15s 19ms/step - accuracy: 0.8538 - loss: 0.4163
```

Test Accuracy: 0.8562



## Day 18 – Weights and Biases in Neural Networks

### Introduction

Weights and biases are key parameters that neural networks learn during training. They define how inputs are transformed to produce the output.

### Weights

- Represent the strength of each input's influence.
- Inputs are multiplied by their weights before summing.
- Adjusted during training to minimize error.  
**Example:** Input=4, Weight=2 → Weighted Input=4×2=8

### Bias

- Added after weighted sum to shift output up or down.
- Allows the model to fit data even when inputs are zero.  
**Example:** Weighted Input=8, Bias=3 → Output=8+3=11

### Neuron Formula

$$z=(w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n) + b_z = (w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n) + b_z = (w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n) + b$$

### Weight vs Bias

| Feature   | Weights              | Bias            |
|-----------|----------------------|-----------------|
| Role      | Importance of inputs | Output shift    |
| Applied   | Multiplies input     | Added after sum |
| Quantity  | One per input        | One per neuron  |
| Trainable | Yes                  | Yes             |

### Code: Visualizing Effect

```
import numpy as np
import matplotlib.pyplot as plt
def neuron_output(x,w,b): return w*x+b
x=np.linspace(-5,5,100)
plt.plot(x, neuron_output(x,1,0),label='w=1,b=0')
plt.plot(x, neuron_output(x,2,0),label='w=2,b=0')
plt.plot(x, neuron_output(x,2,3),label='w=2,b=3')
plt.title("Effect of Weights and Bias");plt.xlabel("Input");plt.ylabel("Output")
plt.legend();plt.grid(True);plt.show()
```

### Conclusion :->

Weights control input importance.

Bias shifts the output.

Both are trained using backpropagation to reduce error.

## Day 19 – Optimizers in Deep Learning

### Understanding Optimizers

Optimizers adjust weights and biases to minimize loss during training. A good optimizer improves accuracy, speeds up training, and ensures better convergence.

### Why Optimizers Matter

- Control learning rate
- Avoid local minima or saddle points
- Stabilize training and speed up convergence

### Popular Optimizers

#### 1. SGD (Stochastic Gradient Descent)

- Updates weights per training sample.
- Formula:  $\theta = \theta - \eta \nabla J(\theta)$
- Pros: Simple, low memory
- Cons: Slow, fluctuates

#### 2. Momentum

- Adds velocity term for smoother, faster convergence.

#### 3. RMSProp

- Adapts learning rate for each parameter.
- Works well on noisy data and RNNs.

#### 4. Adam

- Combines Momentum + RMSProp.
- Adaptive, fast, widely used.

### Code: Comparing Optimizers

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import SGD, RMSprop, Adam
import matplotlib.pyplot as plt
(x_train,y_train),(x_test,y_test)=mnist.load_data()
x_train,x_test=x_train/255.0,x_test/255.0
def create_model(opt):
    model=Sequential([
        Flatten(input_shape=(28,28)),
        Dense(128,activation='relu'),
        Dense(10,activation='softmax')])
    model.compile(optimizer=opt,loss='sparse_categorical_crossentropy',metrics=['accuracy'])
    return model
optimizers={'SGD':SGD(),'RMSProp':RMSprop(),'Adam':Adam()}
```

```

history_dict={}
for name,opt in optimizers.items():
    print(f"\nTraining with {name}...")
    model=create_model(opt)
    history=model.fit(x_train,y_train,epochs=5,validation_split=0.2,verbose=0)
    history_dict[name]=history
plt.figure(figsize=(8,5))
for name,h in history_dict.items():
    plt.plot(h.history['val_accuracy'],label=name)
plt.title("Optimizer Comparison");plt.xlabel("Epochs");plt.ylabel("Val Accuracy")
plt.legend();plt.grid(True);plt.show()

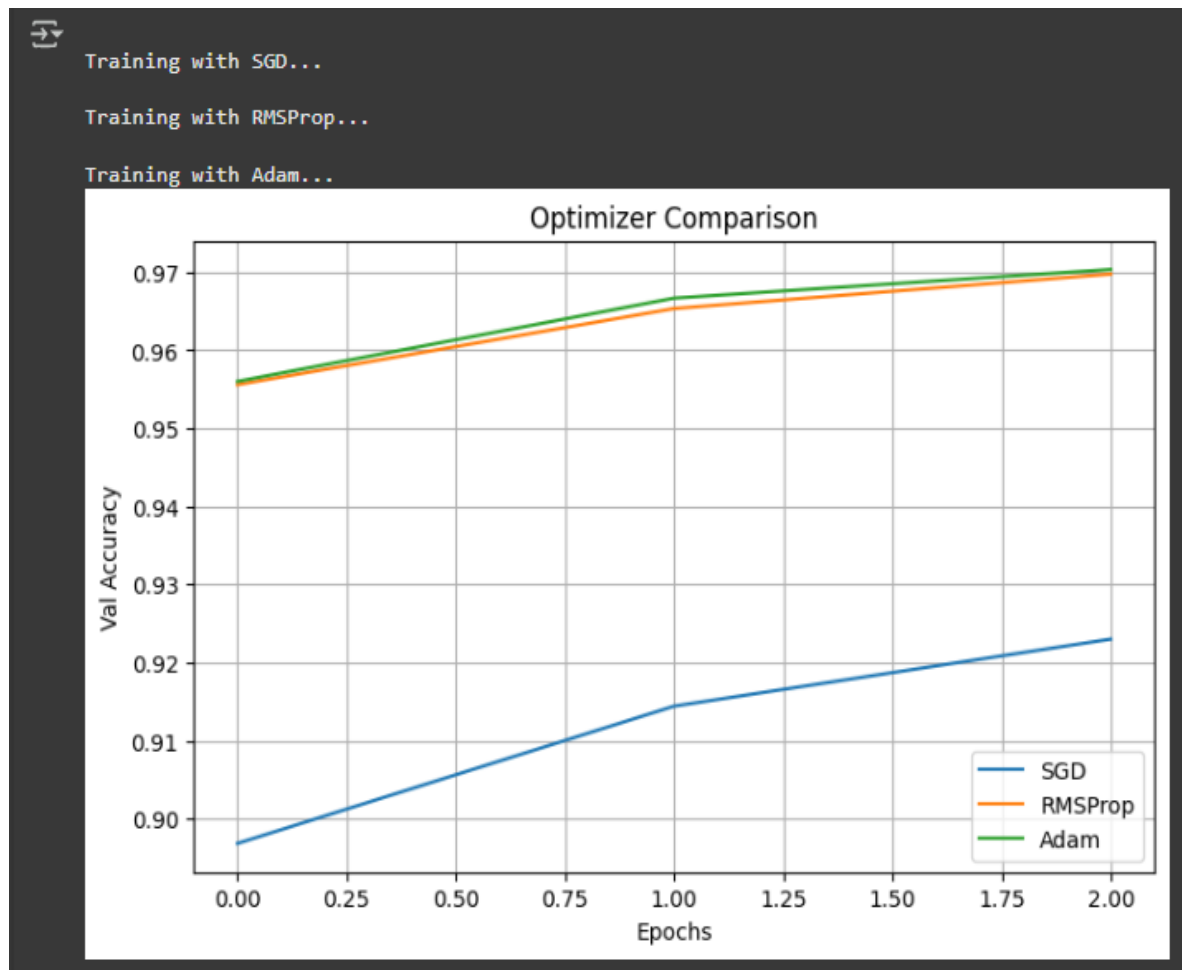
```

### Quick Comparison

| Optimizer | Strengths      | Limits         |
|-----------|----------------|----------------|
| SGD       | Simple         | Slow, unstable |
| Momentum  | Faster         | Needs tuning   |
| RMSProp   | Handles noise  | LR sensitive   |
| Adam      | Adaptive, fast | More memory    |

### Conclusion

Optimizers directly impact training speed and accuracy. Adam is widely preferred for its balance of speed and adaptability, but testing multiple optimizers is often best.





## Day 20: How Neural Networks Learn – The Secret of Weight Updates

### Core Idea

Weights are the adjustable parameters in neural networks. Initially random, they are refined through training to minimize prediction errors and make the model smarter.

### The Learning Cycle

1. **Forward Pass** – Input flows through the network to generate a prediction.
2. **Loss Calculation** – Compare prediction vs actual using a loss function.
3. **Backpropagation** – Compute gradients showing each weight's contribution to the error.
4. **Weight Update (Gradient Descent)** – Adjust weights opposite to the gradient direction.

### Python Demo – Simple Weight Update

```
import numpy as np

x = np.array([1, 2, 3])    # Input
y = np.array([2, 4, 6])    # Target (y=2x)
w = 0.0                    # Initial weight
lr = 0.01                  # Learning rate

def mse(y, y_pred): return ((y - y_pred) ** 2).mean()

for epoch in range(50):
    y_pred = w * x          # Forward pass
    loss = mse(y, y_pred)   # Loss
    grad = -2 * np.dot((y - y_pred), x) / len(x) # Gradient
    w -= lr * grad          # Update weight
    if epoch % 10 == 0:
        print(f"Epoch {epoch}: Loss={loss:.4f}, Weight={w:.4f}")
```

**Output:** Weight gradually approaches 2 (true relationship).

### Key Points

- **Small learning rate** = slow, stable learning.
- **Large learning rate** = faster but risk of overshooting.
- **Bias** shifts outputs to better fit data.
- Optimizers like **Adam/RMSProp** add momentum and adaptive learning rates.

### Analogy

Learning weights is like throwing darts: throw → miss → adjust → repeat, until you consistently hit the target.

### Takeaways

- Weights are tuned to learn data patterns.
- Backpropagation + gradient descent is the feedback loop driving learning.
- Choosing proper learning rates/optimizers ensures fast, stable training.

## **Conclusion**

The AI/ML training program provided a comprehensive and practical understanding of core machine learning and deep learning concepts. Over the course of 20 days, we explored supervised and unsupervised learning, data preprocessing, clustering techniques like K-Means, Hierarchical Clustering, and DBSCAN, as well as neural networks and advanced architectures such as CNNs, RNNs, and LSTMs.

The hands-on implementation of models using real-world datasets significantly enhanced our practical skills, bridging the gap between theory and application. We learned how models are trained, how weights and biases are updated, and how optimizers like SGD, RMSProp, and Adam improve convergence. Through coding exercises in Python and TensorFlow, we built and evaluated models for tasks ranging from customer segmentation and sentiment analysis to image classification.

Additionally, we deepened our understanding of activation functions, the role of weights and biases, gradient descent, and backpropagation—key elements driving neural network learning. The use of visualization and metrics like accuracy, loss curves, and silhouette scores helped in better interpretation and analysis of model performance.

Overall, this training provided not only technical knowledge but also practical problem-solving skills relevant to AI and ML workflows. The exposure to real-world applications demonstrated the transformative power of these technologies in domains such as business intelligence, computer vision, and natural language processing.

By the end of this program, we developed a strong foundation in machine learning and deep learning, equipping us to confidently apply these concepts to future projects and continue advancing in the field of Artificial Intelligence.

## **References**

1. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd Edition). O'Reilly Media.
2. Chollet, F. (2021). *Deep Learning with Python* (2nd Edition). Manning Publications.
3. UCI Machine Learning Repository – Auto MPG Dataset: <http://archive.ics.uci.edu/ml>
4. TensorFlow Documentation: <https://www.tensorflow.org/>
5. Scikit-learn Documentation: <https://scikit-learn.org/>
6. IMDB Movie Reviews Dataset – TensorFlow Datasets:  
[https://www.tensorflow.org/datasets/catalog/imdb\\_reviews](https://www.tensorflow.org/datasets/catalog/imdb_reviews)
7. Fashion MNIST Dataset: <https://github.com/zalandoresearch/fashion-mnist>
8. Python Official Documentation: <https://docs.python.org/>