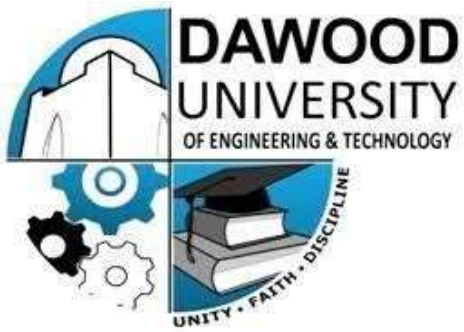
Machine Learning

(Practical Manual)



7th Semester, 4th Year
BATCH -2022

BS ARTIFICIAL INTELLIGENCE

DAWOOD UNIVERSITY OF ENGINEERING & TECHNOLOGY, KARACHI

Dawood University Of Engineering and Technology, Karachi

CERTIFICATE

This is to certify that Mr./Ms. _____with Roll # _____of Batch 2022 has successfully completed all the labs prescribed for the course “Artificial Intelligence”.

Engr. Hamza Farooqui
Lecturer
Department of AI

LAB RULES AND OPERATING PROCEDURES

Before starting a lab, you must read the following instructions. Failure to conform to any of the below Instructions may result in not being allowed to participate in the laboratory experiment.

Respect the Lab Rules:
Respect the specific guidelines provided by the lab supervisor or instructor.

Quiet Environment:
Keep noise levels to a minimum to create a conducive learning environment for everyone.

No Food or Drinks:
Do not bring food or drinks into the computer lab to prevent spills and maintain cleanliness.

Log in with Your Own Credentials:
Use only your assigned login credentials, and do not attempt to access another student's account.

Log Out Properly:
Always log out of the computer and any applications or platforms you use before leaving the computer.

Respect Equipment:
Treat computer equipment and peripherals with care. Report any malfunctioning equipment to lab staff.

No Unauthorized Software Installation:
Do not install or attempt to install any software on the lab computers without permission from lab staff or instructors.

No Tampering with Hardware:
Do not tamper with computer hardware or cables. Report any issues to lab staff.

Internet Usage:
Use the internet for educational purposes only. Avoid accessing inappropriate or non-educational websites.

Be Mindful of Time:
Be aware of the lab's opening and closing hours. Finish your work and leave the lab on time.

Personal Belongings:
Keep personal belongings secure. Do not leave valuables unattended.

Emergency Procedures:
Familiarize yourself with emergency procedures, including the location of exits and emergency contacts.

I have read and understand these rules and procedures. I agree to abide by these rules and procedures at all times while using these facilities. I understand that failure to follow these rules and procedures will result in my immediate dismissal from the laboratory and additional disciplinary action may be taken.

_____
Student’s Signature

COURSE INFORMATION SHEET
(For Lab Based Course)

Title of Course: Machine Learning (Practical)
Course Code: BSAI-462
Effectiveness: Batch 2022-F and onwards
Credit Hours: 01 CH (Practical)
Instructor Name: Engr. Hamza Farooqui
Email and Contact Information: hamza.farooqui@duet.edu.pk

Lab Assessment: Lab performance is evaluated based on comprehensive rubric for each experiment, marked out of 20 and Complex Computing Activity based on 10 marks. The key criteria include:

- **Understanding of Concept (CLO-1):** Clarity and depth of ML algorithm.
- **Code Implementation (CLO-1):** Structure, correctness, and robustness of the Python code.
- **Use of ML Libraries & Features (CLO-2):** Appropriate and efficient use of ML tools for data analysis, training, and evaluation.
- **Results and Report (CLO-2):** Accuracy of the model results and clarity in performance evaluation.

Aim:

The aim of this lab is to equip students with practical skills and knowledge to apply machine learning algorithms for data-driven decision-making through hands-on implementation and real-world problem scenarios.

Objectives:

- The objective of this lab is to develop proficiency in data preprocessing, model training, and performance evaluation of machine learning algorithms.
- To build an understanding of how to select appropriate ML models for given datasets and optimize them for better accuracy and generalization.

Course Learning Outcomes (CLOs):

Upon successful completion of this course, students will be able to:

Mapping of CLOs and GAs			
Sr. No	**Course Learning Outcomes**	**GAs**	**Knowledge Level**
CLO-1	Practice fundamental machine learning algorithms to solve real-world problems using Python libraries.	GA-3	P-3
CLO-2	Demonstrate the ability to evaluate the performance of machine learning models using performance metrics and optimize models through techniques like hyperparameter tuning.	GA-5	P-4
GA= Graduate Attribute, C = Cognitive Domain, P = Psychomotor Domain, A= Affective Domain			

Complex Computing Activity (CCA) Details	**Included: Yes** Nature and details of Complex Computing Activity (CCA): • Conducted as a subject project in groups of 2–4 students. • The project involves designing and implementing an advanced machine learning application that integrates multiple ML techniques (e.g., supervised and unsupervised learning, model optimization, and data visualization). • Students are required to collect, preprocess, and analyze data, build and evaluate models, and present insights or real-time results tools and simulation environments. • Range of Computing Activity Involved 1 and 2 • Assessment through Report and Viva.

Assessment Breakdown and CLO mapping:

Assessment Tool	Marks	Mapped CLOs	CLO-1 Marks	CLO-2 Marks	CLO-3 Marks
Lab work/Report	20	CLO-1, CLO-2	10	10	–
CCA Project	10	CLO-1, CLO-2	5	5	–
Lab Exam	10	CLO-1, CLO-2	5	5	–
Viva	10	CLO-2	–	–	10
Total	50	–	20	20	10

TABLE OF CONTENTS

LAB # 01

DATA PREPROCESSING

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 01

DATA PREPROCESSING

OBJECTIVE

To understand and implement Data preprocessing in preparing real-world datasets for machine learning.

THEORY

Before training any machine learning model, it is crucial to prepare and clean the dataset so that algorithms can interpret it correctly. Data preprocessing improves the accuracy, speed, and reliability of the model.

1. Dataset Cleaning: -

Raw datasets often contain missing, duplicate, or inconsistent values, which can mislead a model. The Titanic dataset, for instance, has missing values in columns like *Age* and *Embarked*.

Handling Missing Data: -

- **Detection:** Identify missing values using methods like isnull() or info().
- **Imputation:** Replace missing numeric values (like *Age*) with statistical measures such as:
  - Mean → useful for normally distributed data
  - Median → less sensitive to outliers
  - Mode → for categorical variables
- **Dropping:** If a feature has too many missing values or is irrelevant, it may be dropped entirely.

Example:

- Fill missing Age values with the mean age.
- Drop rows where Embarked is missing.

Proper handling of missing data ensures that the dataset remains representative and the model learns from valid patterns.

2. Encoding Categorical Data: -

Machine learning models work with numerical data, so categorical (text) data must be converted into numbers.

a. Label Encoding

Converts categories into integers.
Example:
Sex → Male = 0, Female = 1

b. One-Hot Encoding

Creates separate binary columns for each category.
Example:

Embarked → C, Q, S
becomes:
Embarked_C, Embarked_Q, Embarked_S (values are 0 or 1)
One-Hot Encoding prevents the model from assuming an order or hierarchy among categories.

3. Feature Scaling: -

Different features can have values in different ranges (e.g., *Fare* may range from 0–500, while *Age* ranges from 0–80).
Algorithms that depend on distance metrics (e.g., Logistic Regression, SVM, KNN) perform poorly if features are not scaled.

Standardization (Z-score normalization):

Rescales features so that they have:

- Mean = 0
- Standard Deviation = 1

This ensures all features contribute equally to the model's learning process.

4. Splitting the Dataset: -

To evaluate model performance, the dataset is divided into:

- **Training set (80%)** – used to train the model
- **Testing set (20%)** – used to evaluate accuracy on unseen data

This prevents overfitting and helps assess how well the model generalizes.

LAB TASKS

Task 1 – Dataset Cleaning

- Load the Titanic dataset (train.csv).
- Display the first 10 rows.
- Check for missing values in each column.
- Fill missing values in the "Age" column with the mean age.
- Drop rows where the "Embarked" column is missing.

Task 2 – Encoding Categorical Data

- Convert the "Sex" column into numeric (0 = Male, 1 = Female).
- Apply One-Hot Encoding on the "Embarked" column.

Task 3 – Feature Scaling & Splitting

- Select features: Age, Fare, Sex, Pclass.
- Apply StandardScaler to normalize them.
- Split data into 80% training and 20% testing.

LAB OUTCOMES

```
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```
✓ 0.0s

```
# Load the Titanic dataset from OpenML (v1 is the standard version)
titanic = fetch_openml('titanic', version=1, as_frame=True)
df = titanic.frame
```
✓ 0.0s

```
# Display the first 10 rows
print("--- First 10 Rows ---")
print(df.head(10))
```
✓ 0.0s

```
--- First 10 Rows ---
   pclass survived                                             name     sex  \
0       1        1                    Allen, Miss. Elisabeth Walton  female
1       1        1                   Allison, Master. Hudson Trevor    male
2       1        0                     Allison, Miss. Helen Loraine  female
3       1        0             Allison, Mr. Hudson Joshua Creighton    male
4       1        0  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female
5       1        1                                 Anderson, Mr. Harry    male
6       1        1                Andrews, Miss. Kornelia Theodosia  female
7       1        0                           Andrews, Mr. Thomas Jr    male
8       1        1    Appleton, Mrs. Edward Dale (Charlotte Lamson)  female
9       1        0                           Artagaveytia, Mr. Ramon    male
```

```
# Check for missing values
print("\n--- Missing Values ---")
print(df.isnull().sum())
```
✓ 0.0s

```
--- Missing Values ---
pclass          0
survived        0
name            0
sex             0
age           263
sibsp           0
parch           0
ticket          0
fare            1
cabin        1014
embarked        2
boat          823
body         1188
home.dest     564
dtype: int64
```

```
df['age'] = df['age'].fillna(df['age'].mean())
```
✓ 0.0s

```
df.dropna(subset=['embarked'], inplace=True)
```
✓ 0.0s

```python
#  Task 3: Feature Scaling & Splitting

features = ['age', 'fare', 'sex', 'pclass']

df.dropna(subset=['fare'], inplace=True)

X = df[features]
y = df['survived'].astype(int)
```

✓ 0.0s

```python
# Apply StandardScaler to normalize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

✓ 0.0s

```python
# Split data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.20, random_state=42
)
```

✓ 0.0s

```python
print("\nProcessing Complete.")
print(f"Final Feature Shape: {X_train.shape}")
```

✓ 0.0s

```
Processing Complete.
Final Feature Shape: (1044, 4)
```

LAB # 02

SIMPLE AND MULTIPLE LINEAR REGRESSION

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 02

SIMPLE AND MULTIPLE LINEAR REGRESSION

OBJECTIVE

To understand and implement Simple Linear Regression and Multiple Linear Regression.

THEORY:

Practical Significance

Linear regression is one of the most fundamental and widely used machine learning algorithms. It is used for predicting numerical values based on one or more independent variables. This lab covers:

1. Simple Linear Regression: Predicting a target variable using a single independent variable.
2. Multiple Linear Regression: Extending linear regression to multiple features to improve predictions.

Minimum Theoretical Background

1. **Simple Linear Regression**
   - Models the relationship between a dependent variable YYY and a single independent variable XXX.

Equation:

$$Y = mX + b$$

where:

- Y = Dependent variable (target)
- X = Independent variable (feature)
- m = Slope of the regression line
- b = Intercept

2. **Multiple Linear Regression**
   - Generalizes simple linear regression to multiple features:

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$$

   - b0 is the intercept, and b1,b2,...,bn are the coefficients for features X1,X2,...,Xn.
   - Used when multiple factors influence the dependent variable.

3. **Model Evaluation**
   - Mean Squared Error (MSE)
   - R-squared Score: Measures how well the model explains the variance in the data.

Mathematical Expression

1. **Loss Function:**
   - The model minimizes the **Mean Squared Error (MSE)**:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

   - Where Y_i is the actual value and $\hat{Y}_i$ is the predicted value.
2. **Gradient Descent (Optimization):**
   - Adjusts parameters to minimize the error:

$$b_j = b_j - \alpha\frac{\partial MSE}{\partial b_j}$$

   - Where α is the learning rate.

LAB TASKS

Task 1: Simple Linear Regression:-

- **Load dataset**: Use the Diabetes dataset from sklearn.datasets. Select one feature (bmi) to predict the target (disease progression).
- **Perform Exploratory Data Analysis (EDA):**
  - Plot scatter plot of BMI vs. Disease Progression.
  - Check correlation.
- **Implement Simple Linear Regression** using sklearn.linear_model.LinearRegression:
  - Split data into training and testing sets.
  - Fit the model and predict disease progression.
  - Plot the regression line on the scatter plot.
- **Evaluate the model** using:
  - Mean Squared Error (MSE)
  - R^2 score

Does BMI alone explain most of the variation in disease progression? How does R^2 help explain this?

Task 2: Multivariate Linear Regression: -

- **Load dataset**: Use the same Diabetes dataset, but **include all 10 features** to predict disease progression.
- **Perform EDA:**

    - Generate a correlation heatmap between features and the target.
    - Create pair plots for selected features vs. target.
- **Implement Multivariate Linear Regression:**
    - Use all independent variables to predict the target.
    - Fit and predict using the model.
- **Compare actual vs. predicted values** using:
    - Scatter plot (predicted vs. actual).
    - Residual plot.
- **Evaluate model performance** with:
    - MSE
    - RMSE
    - R^2 score

Which independent variable contributes the most to predicting disease progression?

Task 3: Experimentation: -

- Compare performance of simple vs. multivariate regression in terms of evaluation metrics.

LAB OUTCOMES

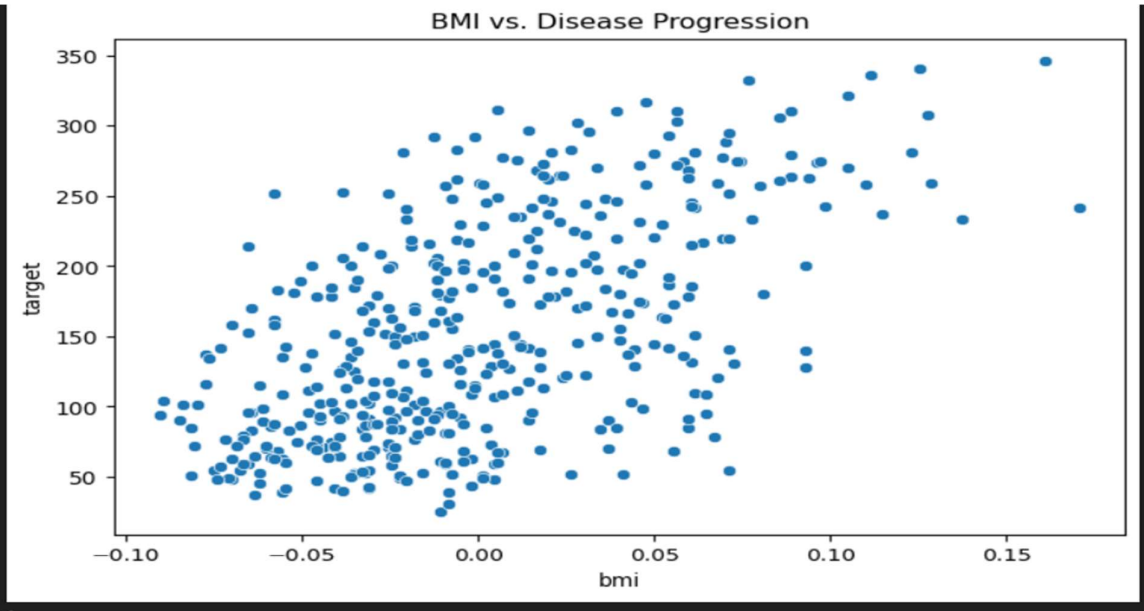
By completing this lab, students will:

- Implement Simple Linear Regression and Multiple Linear Regression.
- Apply data preprocessing to prepare datasets for regression models.
- Evaluate model performance using R-squared Score.

```
[1]
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
[2]
# 1. Load dataset
diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
df['target'] = diabetes.target

# Select 'bmi' as the single feature (reshaped for sklearn)
X_simple = df[['bmi']]
y = df['target']
```

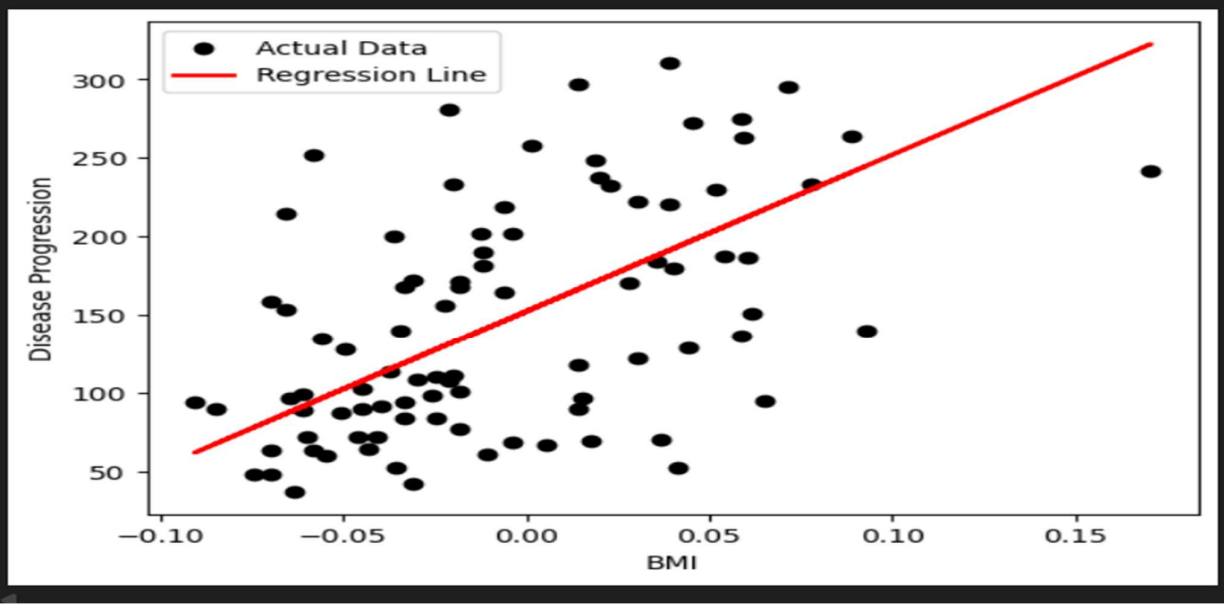
```
[4]
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df['bmi'], y=y)
plt.title("BMI vs. Disease Progression")
plt.show()

print(f"Correlation between BMI and Target: {df['bmi'].corr(y):.4f}")
```




```
Correlation between BMI and Target: 0.5865
```

```
X_train, X_test, y_train, y_test = train_test_split(X_simple, y, test_size=0.2, random_state=42)
model_simple = LinearRegression()
model_simple.fit(X_train, y_train)
y_pred = model_simple.predict(X_test)
```

```
plt.scatter(X_test, y_test, color='black', label='Actual Data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel('BMI')
plt.ylabel('Disease Progression')
plt.legend()
plt.show()
```




```
# 4. Evaluation
mse_simple = mean_squared_error(y_test, y_pred)
r2_simple = r2_score(y_test, y_pred)

print(f"Simple Regression MSE: {mse_simple:.2f}")
print(f"Simple Regression R² Score: {r2_simple:.4f}")
```

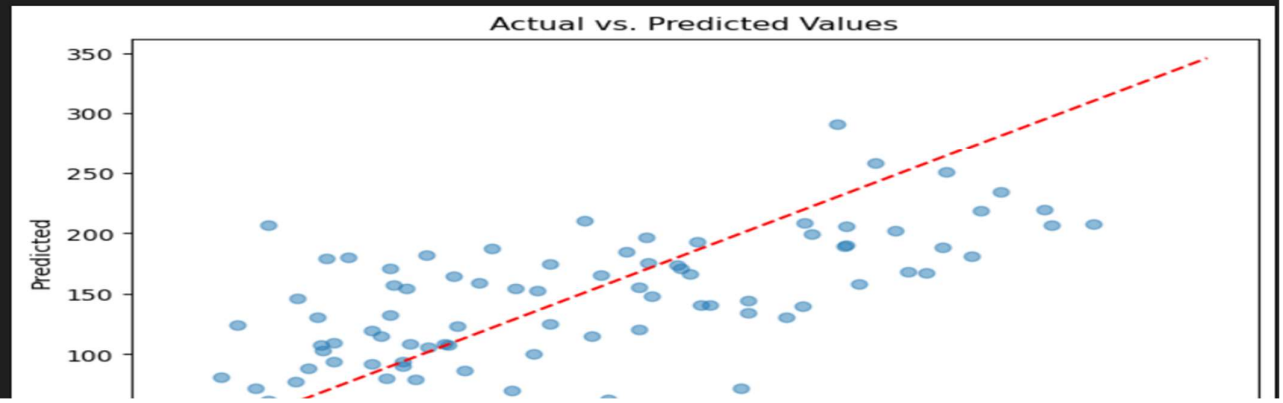
```
Simple Regression MSE: 4061.83
Simple Regression R² Score: 0.2334
```

```
# 1. Implementation
X_multi = df.drop('target', axis=1)
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_multi, y, test_size=0.2, random_state=42)
```

```
# 2. EDA: Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Heatmap")
plt.show()
```

```python
model_multi = LinearRegression()
model_multi.fit(X_train_m, y_train_m)
y_pred_m = model_multi.predict(X_test_m)
```

```python
# 3. Compare Actual vs. Predicted
plt.figure(figsize=(8, 5))
plt.scatter(y_test_m, y_pred_m, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r') # Diagonal line
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs. Predicted Values")
plt.show()
```




```python
# 4. Evaluation
mse_multi = mean_squared_error(y_test_m, y_pred_m)
rmse_multi = np.sqrt(mse_multi)
r2_multi = r2_score(y_test_m, y_pred_m)

print(f"Multivariate MSE: {mse_multi:.2f}")
print(f"Multivariate RMSE: {rmse_multi:.2f}")
print(f"Multivariate R² Score: {r2_multi:.4f}")

coefficients = pd.DataFrame({'Feature': X_multi.columns, 'Coef': model_multi.coef_})
print("\nModel Coefficients:")
print(coefficients.sort_values(by='Coef', ascending=False))
```

```
Multivariate MSE: 2900.19
Multivariate RMSE: 53.85
Multivariate R² Score: 0.4526

Model Coefficients:
  Feature         Coef
8      s5  736.198859
2     bmi  542.428759
5      s2  518.062277
3      bp  347.703844
7      s4  275.317902
6      s3  163.419983
9      s6   48.670657
0     age   37.904021
1     sex -241.964362
4      s1 -931.488846
```

LAB # 03

LOGISTIC REGRESSION

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 03

LOGISTIC REGRESSION

OBJECTIVE

- To understand the concept of Logistic Regression and implement Logistic Regression for binary classification.

THEORY:

Practical Significance

Logistic Regression is one of the most widely used **classification algorithms** in machine learning. It is useful in applications such as:

- **Medical Diagnosis** (predicting if a patient has a disease based on symptoms).
- **Spam Detection** (classifying emails as spam or not spam).
- **Credit Scoring** (predicting if a customer will default on a loan).
- **Customer Churn Prediction** (identifying customers who are likely to leave a service).

Minimum Theoretical Background

1. **Why Logistic Regression?**
   - Linear Regression is not ideal for classification because it produces continuous values.
   - Logistic Regression **maps predictions to probabilities** between **0 and 1** using the **sigmoid function**.
   - It is commonly used for **binary classification** problems.

2. **Sigmoid Function**
   The **sigmoid function** converts a real number into a probability:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

   where $z = wX + b$.

3. **Decision Rule**
   - If $\sigma(z) \geq 0.5$ → Predict **Class 1.**
   - If $\sigma(z) < 0.5$ → Predict **Class 0.**

LAB TASKS

· **Import Required Libraries**

· Load **Dataset**

- Use a dataset **Breast Cancer Dataset** from sklearn.datasets.

· **Exploratory Data Analysis (EDA)**

- Display first few rows of the dataset.

- Check for missing values.
- Plot histograms or distribution of features.
- Check class distribution (malignant vs. benign).

· **Data Preprocessing**

- Split features and labels.
- Standardize features using StandardScaler.
- Train-test split.

· **Model Implementation**

- Use sklearn.linear_model.LogisticRegression.
- Train the model on the training data.
- Predict on the test data.

· **Evaluation**

- Accuracy score.
- Confusion matrix.
- Precision, Recall, F1-score.

LAB OUTCOMES

By completing this lab, students will:

- Understand the working of Logistic Regression.
- Learn how to apply Logistic Regression for binary classification.
- Evaluate classification models using accuracy, precision, recall, and F1-score.
- Gain hands-on experience in evaluating classification models.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

✓ 3.3s

```
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

print("--- Dataset Head ---")
print(df.head())
```

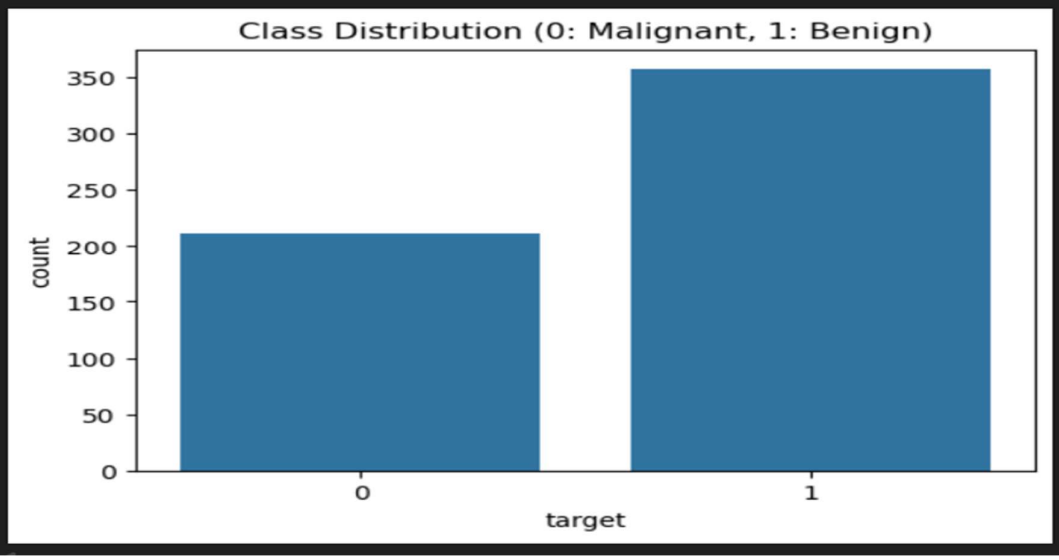
✓ 0.0s

```
--- Dataset Head ---
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809
```

```python
# Check for missing values
print("\nMissing Values:", df.isnull().sum().sum())

# Check class distribution
plt.figure(figsize=(6, 4))
sns.countplot(x='target', data=df)
plt.title('Class Distribution (0: Malignant, 1: Benign)')
plt.show()
```

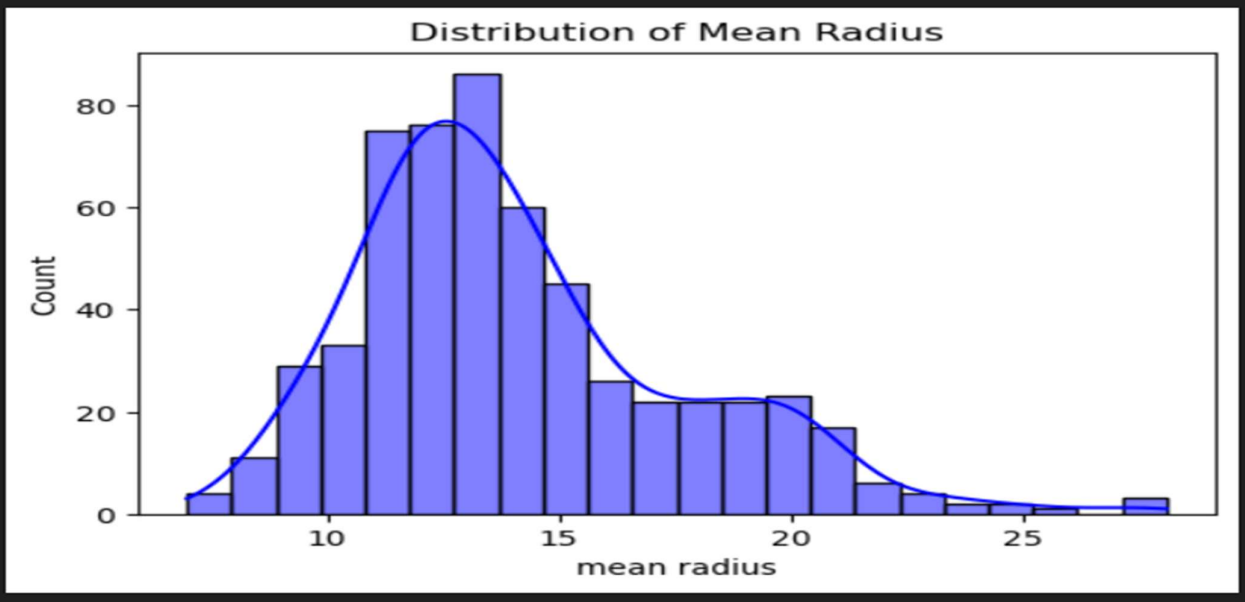
✓ 0.1s

```
Missing Values: 0
```




```python
plt.figure(figsize=(6, 4))
sns.histplot(df['mean radius'], kde=True, color='blue')
plt.title('Distribution of Mean Radius')
plt.show()
```

✓ 0.2s




```python
# Split features and labels
X = df.drop('target', axis=1)
y = df['target']
```

```python
# Split features and labels
X = df.drop('target', axis=1)
y = df['target']
```

✓ 0.0s

```python
# Train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

✓ 0.0s

```python
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

✓ 0.0s

```python
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Predict on test data
y_pred = model.predict(X_test_scaled)
```

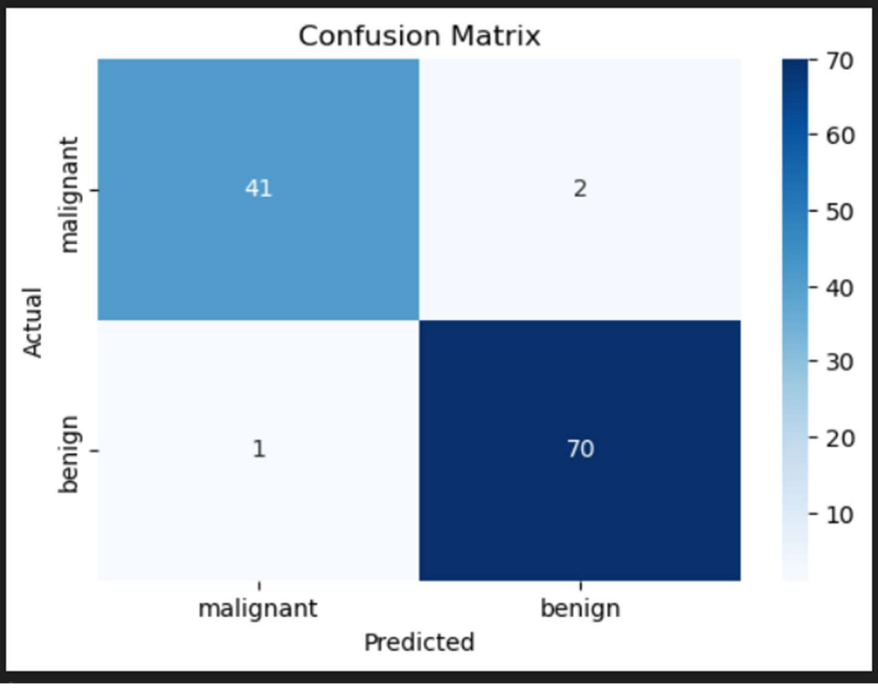
```python
# Evaluation Metrics
print("\n--- Model Evaluation ---")
print(f"Accuracy Score: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
--- Model Evaluation ---
Accuracy Score: 0.9737

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

```python
# Confusion Matrix Visualization
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=data.target_names, yticklabels=data.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

✓ 0.1s



LAB # 04

DECISION TREE CLASSIFIER

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 04

DECISION TREE CLASSIFIER

OBJECTIVE

To understand and implement a Decision Tree Classifier for classification tasks.

THEORY:

Practical Significance

Decision Tree classifiers are widely used for classification problems because they:

- Are **easy to interpret** and visualize.
- Work well with **both numerical and categorical data**.
- Do not require feature scaling.
- Can **handle missing values** better than other models.
- Are used in real-world applications like:
  - **Medical diagnosis** (e.g., predicting disease based on symptoms).
  - **Fraud detection** (e.g., detecting fraudulent transactions).
  - **Customer segmentation** (e.g., classifying customers based on purchasing behavior).

Minimum Theoretical Background

1. **How Decision Trees Work**
   - Decision trees split the dataset **recursively** into smaller subsets.
   - The splits are based on **feature values** that minimize impurity.
   - Each internal node represents a **decision rule**, and each leaf node represents a **class label**.

2. **Splitting Criteria**

   The algorithm selects the best feature to split the data based on impurity measures:
   - **Gini Index**: Measures impurity using the formula:

$$Gini = 1 - \sum_{i=1}^{c} p_i^2$$

where p_i is the probability of class i.

   - **Entropy (Information Gain)**: Measures impurity using:

$$Entropy = -\sum_{i=1}^{c} p_i \log_2 p_i$$

where p_i is the probability of class i.

   - The feature that provides the **highest Information Gain** or **lowest Gini Index** is chosen for splitting.

3. **Overfitting and Pruning**
    - **Overfitting** occurs when the tree is too deep and fits noise in the training data.
    - **Pruning** is used to **reduce tree complexity** and improve generalization.
    - Techniques include **pre-pruning** (limiting tree depth) and **post-pruning** (removing unnecessary branches).

LAB TASKS

Task 1: Load Dataset
- Use a dataset such as **Iris dataset** (from sklearn) or load any CSV file.

Task 2: Exploratory Data Analysis (EDA)
- Display dataset information, shape, and statistical summary.
- Plot distributions of features (histograms / pairplot).
- Visualize correlations

Task 3: Data Preprocessing
- Split dataset into training and testing sets (e.g., 70% train, 30% test).

Task 4: Build Decision Tree Classifier
- Train the model using DecisionTreeClassifier.

Task 5: Model Evaluation
- Make predictions on the test data.
- Evaluate using:
    - Accuracy Score
    - Confusion Matrix
    - Classification Report

Task 6: Experimentation
- Change parameters (criterion = "gini" vs "entropy", max_depth, min_samples_split).
- Compare results and observe **overfitting vs underfitting**.

LAB OUTCOMES

By completing this lab, students will:
- Understand the working of Decision Tree Classifier.
- Apply Gini Index and Entropy for splitting criteria.
- Gain hands-on experience in evaluating classification models.

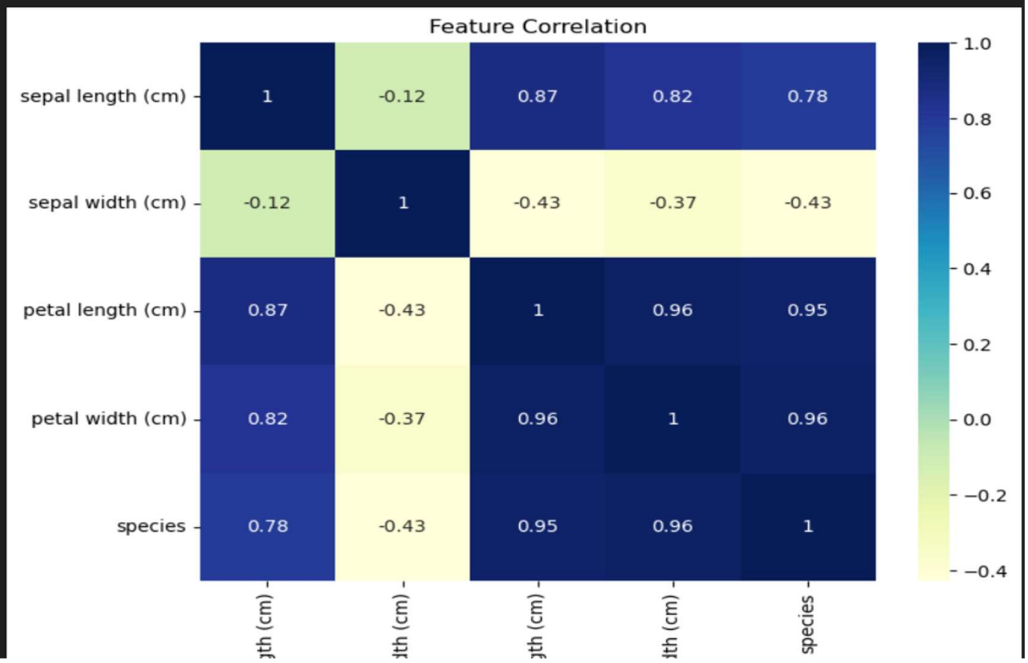
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
```

```
print("Dataset Shape:", df.shape)
print("\nStatistical Summary:\n", df.describe())
```

```
Dataset Shape: (150, 5)

Statistical Summary:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap='YlGnBu')
plt.title("Feature Correlation")
plt.show()
```



```python
# 3. Split data (70% train, 30% test)
X = df.drop('species', axis=1)
y = df['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```python
# 4. Build Decision Tree Classifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

DecisionTreeClassifier

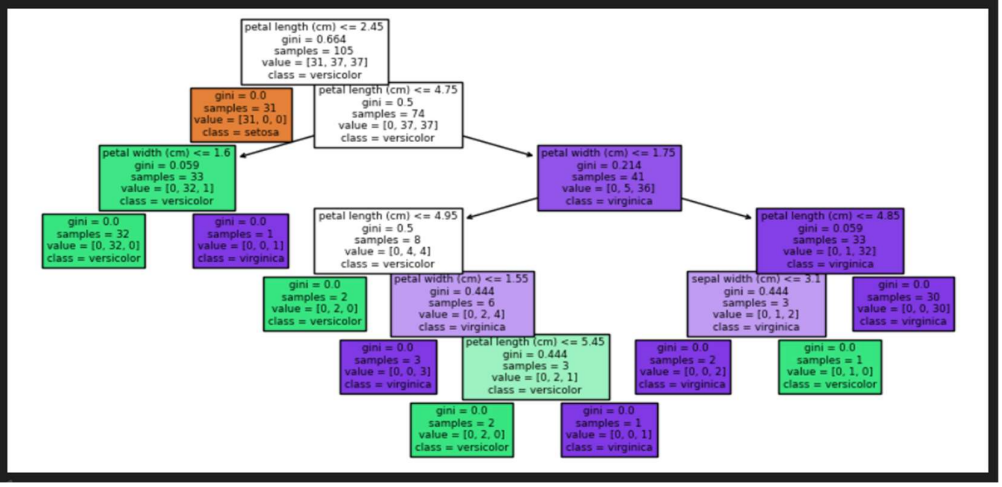
▶ Parameters

```python
# 5. Predictions and Evaluation
y_pred = model.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 1.0000

Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

```python
plt.figure(figsize=(10, 5))
plot_tree(model, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()
```

✓ 0.4s




```python
model_pruned = DecisionTreeClassifier(criterion="entropy", max_depth=3, min_samples_split=10, random_state=42)
model_pruned.fit(X_train, y_train)

y_pred_pruned = model_pruned.predict(X_test)
print(f"Pruned Tree Accuracy: {accuracy_score(y_test, y_pred_pruned):.4f}")
```

✓ 0.0s

```
Pruned Tree Accuracy: 0.9778
```

LAB # 05

RANDOM FOREST CLASSIFIER

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 05

RANDOM FOREST CLASSIFIER

OBJECTIVE

- To understand and Implement a **Random Forest Classifier** for classification tasks.

THEORY:

Practical Significance

Random Forest is an ensemble learning method that combines multiple decision trees to improve classification performance. It is widely used because:

- It **reduces overfitting** compared to individual decision trees.
- It is **robust to noise and missing data**.
- It is used in real-world applications like:
  - **Medical diagnosis** (e.g., detecting diseases).
  - **Fraud detection** (e.g., credit card fraud)..

Minimum Theoretical Background

1. **How Random Forest Works**
   - Random Forest builds multiple decision trees from **random subsets** of data.
   - Each tree makes a prediction, and the final prediction is based on a **majority vote** (classification) or an **average** (regression).

2. **Key Features of Random Forest**
   - **Bootstrap Aggregating (Bagging)**: Each tree is trained on a different random subset of the data.
   - **Feature Randomness**: At each split, only a **random subset** of features is considered.
   - **Majority Voting**: The final prediction is based on the most common class predicted by individual trees.

3. **Comparison with Decision Trees**
   - **Decision Trees** can overfit to training data, while **Random Forests** reduce overfitting.
   - **Random Forests** provide better generalization and robustness.

Mathematical Expression

- Entropy (for information gain):

$$Entropy = -\sum_{i=1}^{c} p_i \log_2 p_i$$

- Gini Index:

$$Gini = 1 - \sum_{i=1}^{c} p_i^2$$

LAB TASKS

1. Load a dataset for classification (e.g., Titanic, Breast Cancer dataset).
2. Apply data preprocessing (handle missing values, encode categorical data).
3. Split the dataset into training and testing sets.
4. Train a Random Forest Classifier on the training data.
5. Make predictions on the test set.
6. Evaluate performance using accuracy, precision, recall, and F1-score.
7. Visualize the Confusion Matrix
8. Compare with a Single Decision Tree

LAB OUTCOMES

By completing this lab, students will:

- Understand the working of Random Forest Classifier.
- Learn how bootstrap aggregating (bagging) works in Random Forest.
- Gain hands-on experience in evaluating classification models.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

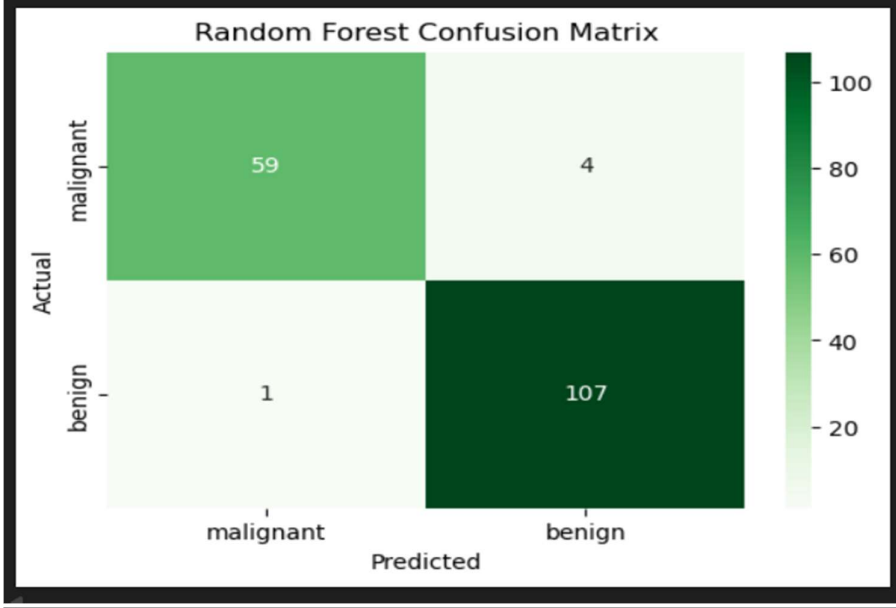
```
(parameter) test_size: Float | None
```

```python
# 4. Train Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# 5. Make Predictions
rf_pred = rf_model.predict(X_test)
```

```python
# 6. Evaluate Performance
print("--- Random Forest Performance ---")
print(f"Accuracy: {accuracy_score(y_test, rf_pred):.4f}")
print(classification_report(y_test, rf_pred))
```

```
--- Random Forest Performance ---
Accuracy: 0.9708
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        63
           1       0.96      0.99      0.98       108

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```

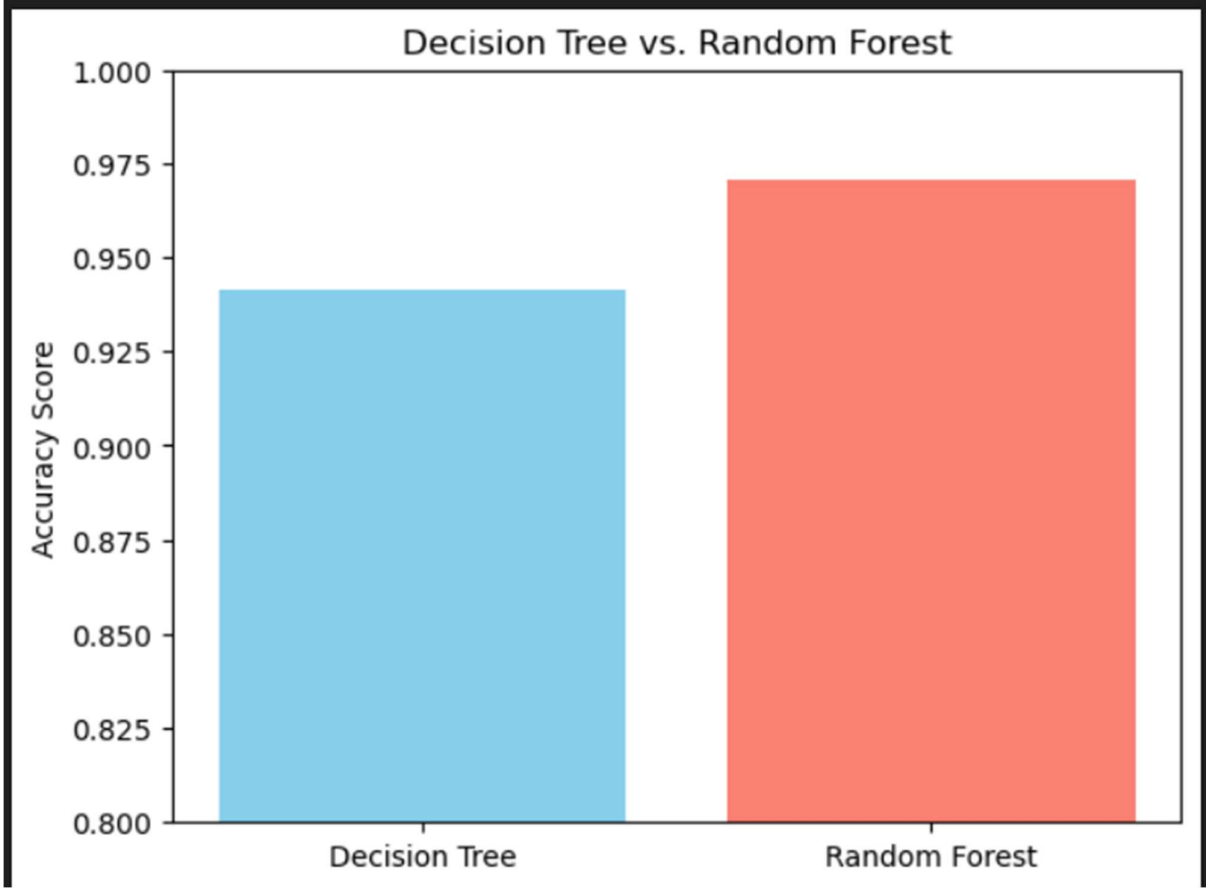
```python
# 7. Visualize Confusion Matrix
cm = confusion_matrix(y_test, rf_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
            xticklabels=data.target_names, yticklabels=data.target_names)
plt.title('Random Forest Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```




```python
# Train a single Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)

print("--- Single Decision Tree Performance ---")
print(f"Accuracy: {accuracy_score(y_test, dt_pred):.4f}")
```

```
--- Single Decision Tree Performance ---
Accuracy: 0.9415
```

```python
# Compare Results
models = ['Decision Tree', 'Random Forest']
accuracies = [accuracy_score(y_test, dt_pred), accuracy_score(y_test, rf_pred)]

plt.bar(models, accuracies, color=['skyblue', 'salmon'])
plt.ylabel('Accuracy Score')
plt.title('Decision Tree vs. Random Forest')
plt.ylim(0.8, 1.0) # Zoom in to see the difference
plt.show()
```



LAB # 06

SUPPORT VECTOR MACHINE (SVM) CLASSIFIER

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 06

SUPPORT VECTOR MACHINE (SVM) CLASSIFIER

OBJECTIVE

To understand the Support Vector Machine (SVM) Classifier algorithm and implement an SVM classifier for binary classification.

THEORY:

Practical Significance

Machine learning models require numerical input. Since categorical data consists of non-numeric

Practical Significance

Support Vector Machines (SVM) are widely used in classification tasks due to their **ability to handle high-dimensional data and small sample sizes**. Some real-world applications include:

- **Face recognition**
- **Text classification** (spam detection, sentiment analysis)
- **Bioinformatics** (gene classification)
- **Medical diagnosis**

Minimum Theoretical Background

1. **How SVM Works**
   - SVM finds the **optimal hyperplane** that maximizes the margin between two classes.
   - The points that **define the margin** are called **support vectors**.

2. **Kernel Trick**
   - When data is not linearly separable, **kernel functions** transform it into a higher-dimensional space where it becomes separable.
   - Common kernels:
     - Linear Kernel: $K(x_i, x_j) = x_i^T x_j$
     - Polynomial Kernel: $K(x_i, x_j) = (x_i^T x_j + c)^d$
     - Radial Basis Function (RBF) Kernel: $K(x_i, x_j) = \exp(-\gamma ||x_i - x_j||^2)$

3. **Soft Margin & C Parameter**
   - **Hard Margin**: Strict separation (only for perfectly separable data).
   - **Soft Margin**: Allows some misclassifications, controlled by **C** (regularization parameter).

LAB TASKS

1. Load a dataset for classification (e.g., Parkinson disease, Breast Cancer dataset).
2. Apply data preprocessing (handle missing values, encode categorical data).
3. Split the dataset into training and testing sets.
4. Apply Grid search to find the optimal parameters
5. Use those parameters to make predictions on the test set.
6. Evaluate performance using accuracy, precision, recall, and F1-score.
7. Visualize the Confusion Matrix

LAB OUTCOMES

By completing this lab, students will:

- Understand Support Vector Machine (SVM) Classifier.
- Learn about different kernel functions in SVM.
- Gain hands-on experience in evaluating classification models.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```python
# 1. Load Dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target
```

Generate + Code

```python
# 2. Preprocessing
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)
```

```python
# 3. Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear']
}
```

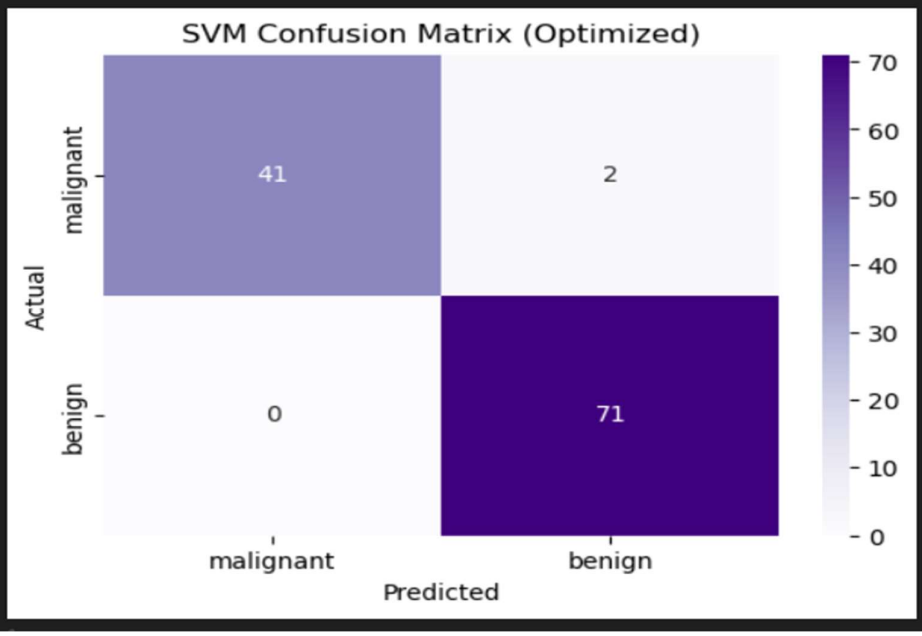
```python
# 4. Apply Grid Search
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=1, cv=5)
grid.fit(X_train, y_train)

# Best parameters found by Grid Search
print(f"Best Parameters: {grid.best_params_}")
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
Best Parameters: {'C': 0.1, 'gamma': 1, 'kernel': 'linear'}
```

```python
# 5. Make Predictions
grid_predictions = grid.predict(X_test)

# 6. Evaluate Performance
print("\n--- Model Evaluation ---")
print(f"Accuracy Score: {accuracy_score(y_test, grid_predictions):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, grid_predictions))
```

```
--- Model Evaluation ---
Accuracy Score: 0.9825

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.98        43
           1       0.97      1.00      0.99        71

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

```python
# 7. Visualize Confusion Matrix
cm = confusion_matrix(y_test, grid_predictions)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples',
            xticklabels=data.target_names, yticklabels=data.target_names)
plt.title('SVM Confusion Matrix (Optimized)')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



LAB # 07

OPEN ENDED LAB

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0–1)	Marks (Out of 20)
Experiment Design (4 marks)	Designs an innovative and well-structured machine learning experiment addressing the open-ended challenge with multiple approaches and clear objectives.	Designs a functional machine learning experiment that addresses the challenge with moderate guidance or limited innovation.	Struggles to design an appropriate experiment or fails to address the challenge adequately.	
Implementation & Execution (6 marks)	Implements the experiment with high technical proficiency using appropriate ML tools and libraries, achieving accurate results supported by sound analysis.	Implements the experiment adequately with minor errors, incomplete testing, or limited analysis.	Implementation is incomplete, results are inaccurate, or lacks analysis.	
Problem Solving & Adaptability (4 marks)	Demonstrates strong analytical skills, creativity, and adaptability in handling challenges during experimentation and model optimization.	Solves problems adequately with occasional guidance or limited creativity.	Unable to effectively solve problems or adapt to issues during execution.	
Report & Presentation (6 marks)	Produces a comprehensive, well-organized report with clear explanations, visualizations, and properly formatted code; delivers an engaging and professional presentation.	Produces a moderately detailed report with basic explanations and visuals; presentation is clear but lacks depth or polish.	Report or presentation is poorly structured, lacks clarity, or contains insufficient explanation of results.	

LAB # 07

OPEN ENDED LAB

OBJECTIVE

To apply the foundational machine learning concepts learned in Labs 1–6 to design and implement a small, custom ML project in Python.

The goal is to integrate multiple concepts learned so far into a single, working prototype that performs real-world data analysis or prediction.

PROJECT DESCRIPTION

This is an open-ended lab where you will be told individually to work on the following project ideas and build a working ML solution in Python.
Your project should demonstrate the integration of at least two machine learning models and appropriate data preprocessing, visualization, and evaluation.

PROJECT IDEAS

The project ideas for the Open-Ended Lab will be **provided to students on the day of the lab session**. These ideas will be designed to ensure the integration and practical application of concepts covered in the first six labs. Each project will require students to demonstrate problem-solving, analytical thinking, and implementation of appropriate machine learning techniques.

DELIVERABLES

- All source code for your project, clearly commented with Project name and roll no on top of the notebook.
- Your dataset description and preprocessing steps.
- The ML models used and how they were integrated.
- Model evaluation (metrics, confusion matrix, etc.).
- Code must be uploaded on GitHub and link paste on QOBE portal.

LAB # 08

KNN CLASSIFIER

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 08

KNN CLASSIFIER

OBJECTIVE

To understand the K-Nearest Neighbors (KNN) Classifier algorithm and implement KNN for classification tasks.

THEORY:

Practical Significance

KNN is widely used in various classification problems due to its simplicity and effectiveness. Some real-world applications include:

- Handwriting recognition (OCR)
- Recommender systems
- Medical diagnosis
- Customer segmentation

Minimum Theoretical Background

1. **How KNN Works**
   - KNN is a **lazy learning algorithm** that stores training data and classifies new data points based on their **K nearest neighbors**.
   - Distance is computed using methods such as:
     - Euclidean Distance:

$$d(x, y) = \sqrt{\sum (x_i - y_i)^2}$$

     - Manhattan Distance:

$$d(x, y) = \sum |x_i - y_i|$$

2. **Choosing the Value of K**
   - Small K: **Sensitive to noise, may overfit**.
   - Large K: **Smoother decision boundary, may underfit**.
   - Optimal K is chosen using **cross-validation**.

Mathematical Expression

- Distance Calculation (Euclidean Distance):

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- Prediction Rule:

$$\hat{y} = \text{mode}(\{y_i | x_i \text{ is among the K nearest neighbors}\})$$

LAB TASKS

1. Load a dataset (e.g., Iris, Breast Cancer dataset).
2. Apply data preprocessing (handle missing values, encode categorical data).
3. Split the dataset into training and testing sets.
4. Train a KNN classifier with different values of K (e.g., 3, 5, 7).
5. Make predictions on the test set.
6. Evaluate performance using accuracy, precision, recall, and F1-score.
7. Test how accuracy changes with different values of K.

LAB OUTCOMES

By completing this lab, students will:

- Understand different types of categorical data.
- Implement categorical encoding techniques in **Scikit-Learn**.
- Gain hands-on experience in evaluating classification models.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score


# 1. Load Dataset
iris = load_iris()
X = iris.data
y = iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```python
# Function to run KNN and return accuracy
def run_knn(k_value):
    knn = KNeighborsClassifier(n_neighbors=k_value)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    return accuracy_score(y_test, y_pred), y_pred
```

```python
# Test for specific K values requested
for k in [3, 5, 7]:
    acc, predictions = run_knn(k)
    print(f"--- Results for K={k} ---")
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test, predictions, target_names=iris.target_names))
```

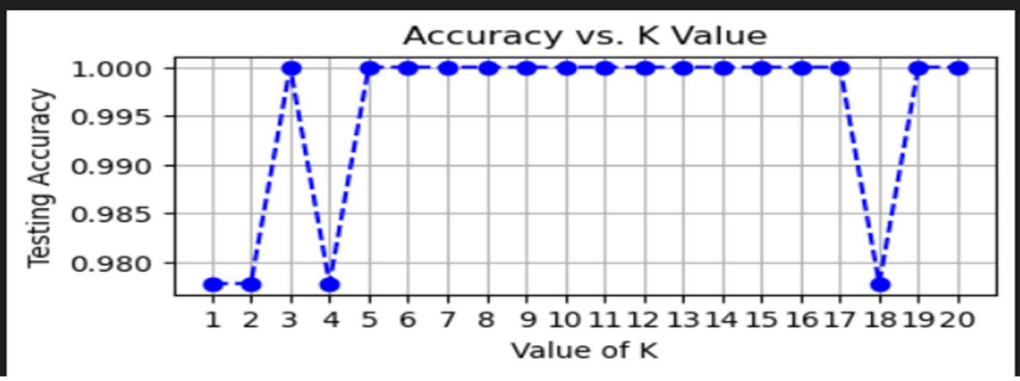
```
--- Results for K=3 ---
Accuracy: 1.0000
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

--- Results for K=5 ---
Accuracy: 1.0000
```

```python
k_range = range(1, 21)
accuracies = []

for k in k_range:
    acc, _ = run_knn(k)
    accuracies.append(acc)
```

✓ 0.0s

```python
plt.figure(figsize=(5, 2))
plt.plot(k_range, accuracies, marker='o', linestyle='dashed', color='blue')
plt.title('Accuracy vs. K Value')
plt.xlabel('Value of K')
plt.ylabel('Testing Accuracy')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```

✓ 0.1s



LAB # 09

NAIVE BAYES CLASSIFIER

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 09

NAIVE BAYES CLASSIFIER

OBJECTIVE

To understand the Naïve Bayes Classifier and its variants and implement Gaussian Naïve Bayes for classification tasks.

THEORY:

Practical Significance

Naïve Bayes is a **probabilistic classifier** widely used for:

- **Spam filtering** (email classification).
- **Sentiment analysis** (positive/negative reviews).
- **Medical diagnosis** (disease prediction).
- **Text classification** (news categorization, topic modeling).
- **Face recognition** (Bayesian-based facial classification).

Minimum Theoretical Background

1. **Bayes' Theorem**
   Naïve Bayes is based on **Bayes' Theorem**, which states:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

   where:

   - $P(A|B)$ is the **posterior probability** (probability of class A given evidence B).
   - $P(B|A)$ is the **likelihood** (probability of evidence B given class A).
   - $P(A)$ is the **prior probability** (probability of class A before seeing evidence).
   - $P(B)$ is the **marginal probability** (probability of evidence B occurring).

2. **Types of Naïve Bayes Classifiers**
   - **Gaussian Naïve Bayes** (for continuous numerical data).
   - **Multinomial Naïve Bayes** (for text classification problems).
   - **Bernoulli Naïve Bayes** (for binary/boolean features).

3. **Naïve Assumption**
   - The classifier assumes that all features are **independent** (which is rarely true in real-world data, but the algorithm still performs well).

LAB TASKS

1. **Load Iris dataset**
2. **Apply data preprocessing** (handle missing values, encode categorical data if needed).
3. **Split the dataset** into training and testing sets.
4. **Train the Naïve Bayes Model** using Gaussian Naïve Bayes since features are continuous.
5. **Make predictions** on the test set.
6. **Evaluate performance**.
7. Test the model on new input/unseen data.

LAB OUTCOMES

By completing this lab, students will:

- Understand the Bayes' Theorem and its application in classification.
- Learn about Gaussian, Multinomial, and Bernoulli Naïve Bayes classifiers
- Gain hands-on experience in evaluating classification models.

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
iris = load_iris()
X = iris.data
y = iris.target
```

Generate + Code

Start Chat to Generate Code (Ctrl+I)

```
print(f"Missing values: {np.isnan(X).sum()}")
```

```
Missing values: 0
```

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

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

GaussianNB

▶ Parameters

```
    y_pred = model.predict(X_test)
```

```
    print("--- Naïve Bayes Performance ---")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
--- Naïve Bayes Performance ---
Accuracy: 1.0000

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
```

```
    new_data = np.array([[5.1, 3.5, 1.4, 0.2],  # Expected Setosa-like
                         [6.5, 3.0, 5.2, 2.0]]) # Expected Virginica-like

    predictions = model.predict(new_data)

    for i, pred in enumerate(predictions):
        print(f"Sample {i+1} predicted as: {iris.target_names[pred]}")
```

```
Sample 1 predicted as: setosa
Sample 2 predicted as: virginica
```

LAB # 10

K MEANS CLUSTERING

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 10

K MEANS CLUSTERING

OBJECTIVE

To understand the K-Means Clustering algorithm for unsupervised learning and implement K-Means clustering on real-world datasets to group similar data points.

THEORY:

Practical Significance

K-Means clustering is used for:

- **Customer segmentation** (grouping similar customers for targeted marketing).
- **Market basket analysis** (finding patterns in purchase behavior).
- **Image compression** (reducing the number of colors used in an image).
- **Document clustering** (grouping similar documents for content analysis).
- **Anomaly detection** (identifying outliers in datasets).

Minimum Theoretical Background

1. **What is K-Means Clustering?**
   K-Means is a **partitioning-based unsupervised learning algorithm** used to divide a dataset into **K clusters**. The objective is to minimize the sum of squared distances between data points and their respective cluster centroids.

2. **How K-Means Works:**
   - **Step 1:** Select **K initial centroids** randomly or using specific initialization techniques like **K-Means++**.
   - **Step 2:** Assign each data point to the nearest centroid based on a distance metric (typically **Euclidean distance**).
   - **Step 3:** Recalculate the centroids as the mean of all data points in the cluster.
   - **Step 4:** Repeat steps 2 and 3 until the centroids no longer change significantly (convergence).

3. **Choosing the Right K:**
   - The number of clusters **K** can be determined using methods such as:
     - **Elbow Method**: Plot the sum of squared distances (inertia) for different values of K and identify the "elbow point" where the inertia starts to level off.
     - **Silhouette Score**: Measures how similar an object is to its own cluster compared to other clusters. A higher score indicates better-defined clusters.

4. **Advantages of K-Means:**
   - Simple and fast for large datasets.
   - Works well when clusters are spherical and roughly of similar size.

5. **Limitations:**
    - Sensitive to the initial placement of centroids.
    - Assumes clusters are of **roughly similar sizes**.
    - Struggles with clusters that are **non-spherical or of different densities**.

Mathematical Expression

The K-Means algorithm minimizes the following objective function (called the **inertia**):

$$J = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where:

- K is the number of clusters.
- C_i is the set of data points assigned to cluster i.
- x is a data point.
- μ_i is the centroid of cluster i.
- $\|x - \mu_i\|$ is the Euclidean distance between data point x and the centroid μ_i.

LAB TASKS

1. Load a dataset (e.g., Mall Customer Segmentation).
2. Apply data preprocessing (normalize features if needed).
3. Determine the optimal number of clusters using Elbow Method
4. Implement K-Means clustering for the selected value of K.
5. Visualize the clusters in 2D or 3D (if the dataset has 2 or 3 features).
6. Evaluate the clustering: Although K-Means is unsupervised, since the Iris dataset has labels, we can compare them.

LAB OUTCOMES

By completing this lab, students will:

- Understand the K-Means Clustering algorithm and its working mechanism.
- Learn how to determine the optimal number of clusters using methods like the Elbow Method
- Gain practical experience in clustering real-world data and visualizing the results.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
```

✓ 3.2s

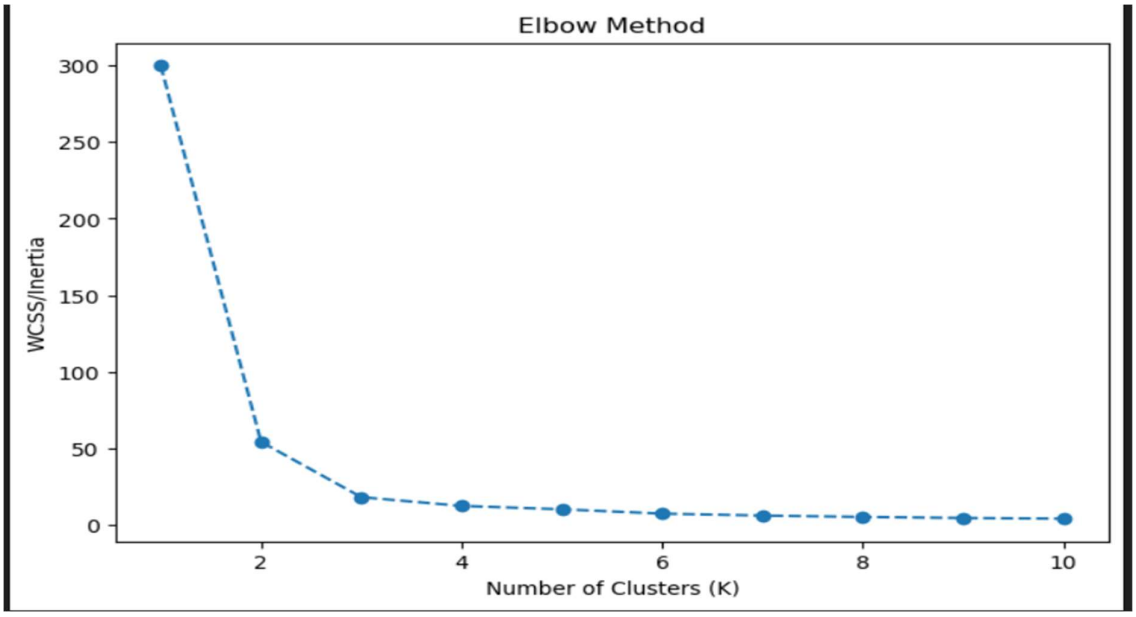
```python
iris = load_iris()
X = iris.data[:, [2, 3]]
target = iris.target
```

✓ 0.0s

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

✓ 0.0s

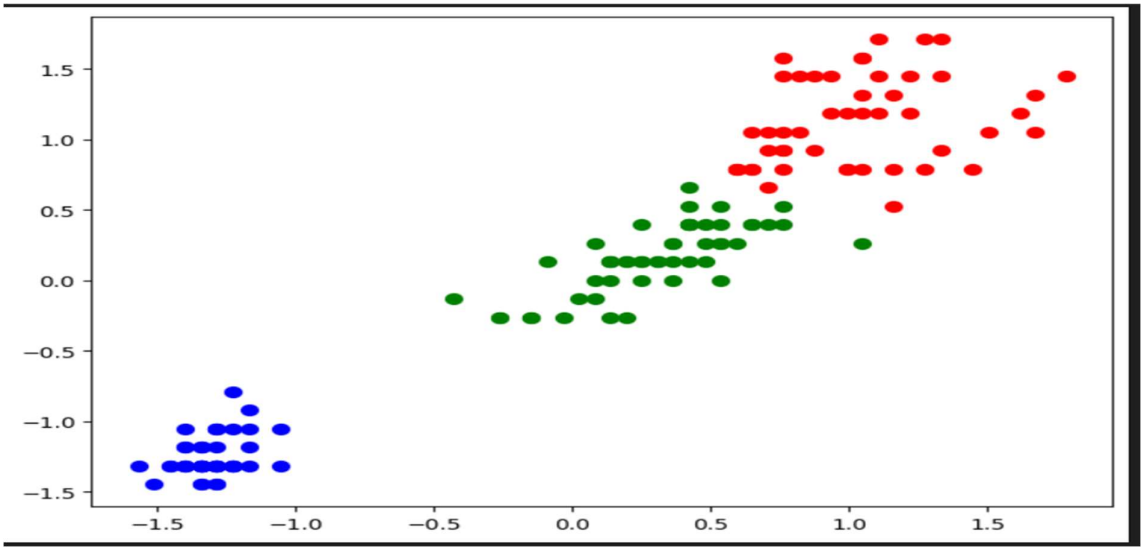
```python
wcss = [] # Within-Cluster Sum of Squares
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS/Inertia')
plt.show()
```



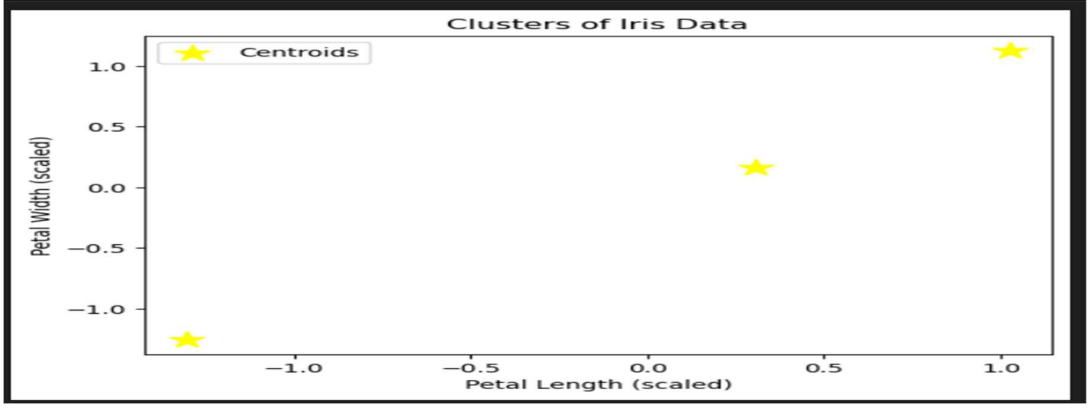
```python
# 4. Implement K-Means
k_optimal = 3
kmeans = KMeans(n_clusters=k_optimal, init='k-means++', random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)
```

```
d:\Anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory
  warnings.warn(
```

```python
# 5. Visualization
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[y_kmeans == 0, 0], X_scaled[y_kmeans == 0, 1], s=50, c='red', label='Cluster 1')
plt.scatter(X_scaled[y_kmeans == 1, 0], X_scaled[y_kmeans == 1, 1], s=50, c='blue', label='Cluster 2')
plt.scatter(X_scaled[y_kmeans == 2, 0], X_scaled[y_kmeans == 2, 1], s=50, c='green', label='Cluster 3')
```




```python
# Plotting centroids
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='yellow', marker='*', label='Centroids')
plt.title('Clusters of Iris Data')
plt.xlabel('Petal Length (scaled)')
plt.ylabel('Petal Width (scaled)')
plt.legend()
plt.show()
```



```python
    # 6. Comparison (Cross-tabulation)
    comparison = pd.DataFrame({'Actual': target, 'Cluster': y_kmeans})
    print("--- Actual vs Cluster Comparison ---")
    print(pd.crosstab(comparison['Actual'], comparison['Cluster']))
```

```
--- Actual vs Cluster Comparison ---
Cluster   0   1   2
Actual
0         0  50   0
1         2   0  48
2        46   0   4
```

LAB # 11

HEIRARCHICAL AGGLOMERATIVE CLUSTERING

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 11

HEIRARCHICAL AGGLOMERATIVE CLUSTERING

OBJECTIVE

To understand the working principle of Hierarchical Agglomerative Clustering (HAC) and implement it using Python to visualize how data points are grouped into clusters based on their similarity through a dendrogram.

THEORY:

What is Hierarchical Clustering?

Hierarchical Clustering is an **unsupervised machine learning algorithm** used to group similar data points into clusters based on their distance or similarity. It builds a hierarchy of clusters that can be represented as a **tree structure (dendrogram)**.

There are two main types:

1. **Agglomerative (Bottom-Up):** Starts with each data point as an individual cluster and merges the closest clusters step by step until one big cluster remains.
2. **Divisive (Top-Down):** Starts with one large cluster and divides it into smaller clusters.

In this lab, we focus on **Agglomerative Clustering**.

Key Concepts

- **Linkage:** Determines how the distance between clusters is measured when merging.
  - **Single linkage:** Minimum distance between points of two clusters.
  - **Complete linkage:** Maximum distance between points of two clusters.
  - **Average linkage:** Mean distance between all pairs of points in two clusters.
  - **Ward linkage:** Minimizes the variance between clusters (commonly used).
- **Distance Metric:** Typically Euclidean distance is used to compute the similarity between points.
- **Dendrogram:** A tree-like diagram showing how clusters merge at each step.

Steps in Agglomerative Clustering

1. Compute pairwise distances between data points.
2. Treat each data point as a separate cluster.
3. Merge the two closest clusters based on the chosen linkage criterion.
4. Repeat until all points are in one cluster.
5. Cut the dendrogram at a chosen level to select the desired number of clusters.

LAB TASKS

Task 1: Load and Explore Data

We will use the **Iris dataset** from scikit-learn.

Task 2: Data Preprocessing

Standardize the data before clustering since the algorithm is distance-based.

Task 3: Create Dendrogram

Use scipy to visualize how data points merge at each step.

Task 4: Apply Agglomerative Clustering

Perform clustering using Agglomerative Clustering from scikit-learn.

Task 5: Visualize Clusters

Visualize the clusters using the first two features for simplicity.

Task 6: Evaluation (Optional)

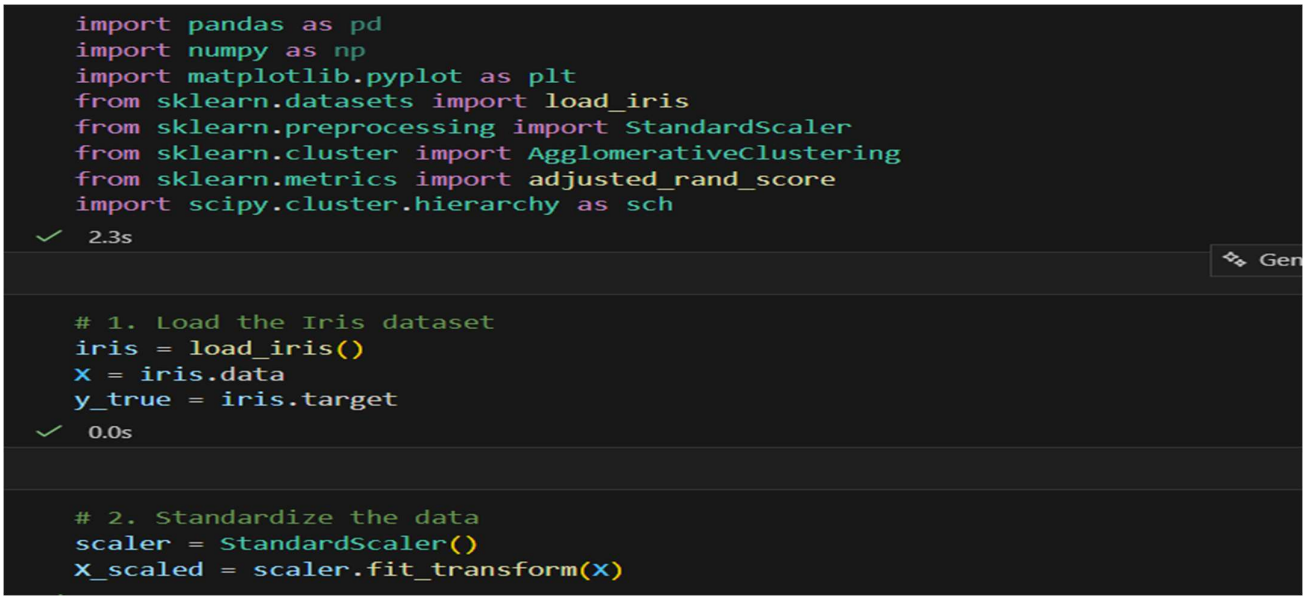
Check clustering performance using the Adjusted Rand Index (ARI).

A higher ARI indicates better agreement between predicted clusters and true labels.

LAB OUTCOMES

By completing this lab, students will:

- Explain the basic concept of hierarchical clustering and its difference from partitional methods like K-Means.
- Describe the step-by-step process of agglomerative clustering (bottom-up approach).
- Implement simple hierarchical agglomerative clustering using Python libraries such as scipy or sklearn.
- Visualize the clustering hierarchy using a dendrogram and interpret how clusters merge at different linkage distances.
- Analyze the effect of different linkage criteria (e.g., single, complete, average) on cluster formation.



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score
import scipy.cluster.hierarchy as sch
```

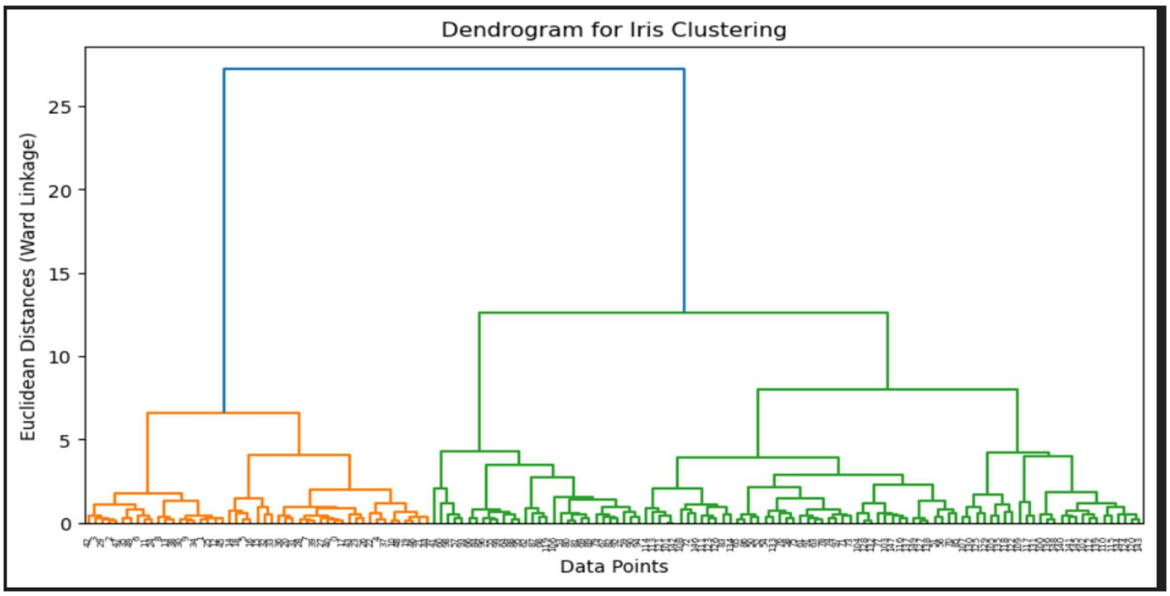
✓ 2.3s

Gen

```
# 1. Load the Iris dataset
iris = load_iris()
X = iris.data
y_true = iris.target
```

✓ 0.0s

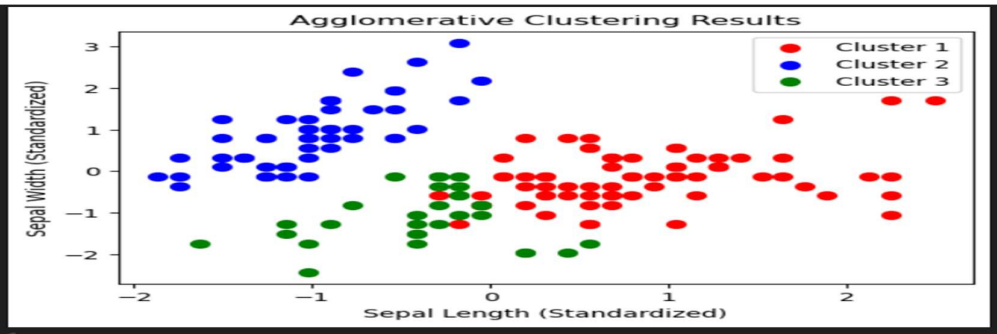
```
# 2. Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```




```
# 4. Perform Agglomerative Clustering
hc = AgglomerativeClustering(n_clusters=3, linkage='ward')
y_hc = hc.fit_predict(X_scaled)
```

✓ 0.0s

```
# 5. Visualize Clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[y_hc == 0, 0], X_scaled[y_hc == 0, 1], s=50, c='red', label='Cluster 1')
plt.scatter(X_scaled[y_hc == 1, 0], X_scaled[y_hc == 1, 1], s=50, c='blue', label='Cluster 2')
plt.scatter(X_scaled[y_hc == 2, 0], X_scaled[y_hc == 2, 1], s=50, c='green', label='Cluster 3')

plt.title('Agglomerative Clustering Results')
plt.xlabel('Sepal Length (Standardized)')
plt.ylabel('Sepal Width (Standardized)')
plt.legend()
plt.show()
```




```
# 6. Check clustering performance
ari = adjusted_rand_score(y_true, y_hc)
print(f"Adjusted Rand Index (ARI): {ari:.4f}")
```

✓ 0.0s

```
Adjusted Rand Index (ARI): 0.6153
```

LAB # 12

ML × AR -- REAL-TIME OBJECT CLASSIFICATION OVERLAY

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 12

ML × AR -- REAL-TIME OBJECT CLASSIFICATION OVERLAY

OBJECTIVE

To demonstrate how a Machine Learning model can process the real-world camera feed in an AR environment and augment the user's view with intelligent visual feedback (like identifying objects or labeling surroundings).

Equipment & Tools:

- **Hardware:** Meta Quest Pro (preferred) or Android phone (if using AR Foundation)
- **Software:**
  - Unity 2021+ with AR Foundation and ARCore/ARKit support
  - Python (optional, for exploring model behavior)
  - Pretrained TensorFlow Lite / ONNX model (MobileNetV2 or similar)
  - Wizard app (for AR scene setup and deployment)

LAB TASKS

Task 1 — AR Scene Setup

Goal: Build a minimal AR scene that shows the device's camera feed and places 3D text in front of detected objects.

Steps:

1. Open the Wizard app → create a new AR scene.
2. Add a camera feed panel or use the headset's passthrough mode.
3. Add a floating text label prefab (initially says "Detecting…").
4. Position text label at a fixed point (later, it will move to detected object).

Task 2 — Integrate ML Model

Goal: Use a pretrained object classification model to identify objects in the camera view.

Steps:

1. Import a lightweight MobileNetV2 model (TFLite or ONNX) into Unity.
2. Add Barracuda package (for ONNX) or TF Lite plugin (for Android).
3. In your AR scene, write a short script:
   - Capture frames periodically from the camera texture (e.g., every 0.5 seconds).
   - Pass the frame to the model for inference.
   - Retrieve top-1 or top-3 predicted labels (e.g., "chair", "cup", "keyboard").

```
using UnityEngine;
using Unity.Barracuda;

public class ARObjectClassifier : MonoBehaviour {
   public NNModel modelAsset;
   private Model model;
   private IWorker worker;
```

```
    public Camera arCamera;
    public TextMesh labelText;

    void Start() {
        model = ModelLoader.Load(modelAsset);
        worker = WorkerFactory.CreateWorker(WorkerFactory.Type.Auto, model);
    }

    void Update() {
        // Capture camera texture
        Texture2D frame = CaptureCameraFrame(arCamera);
        Tensor input = Preprocess(frame);  // Resize & normalize
        worker.Execute(input);
        Tensor output = worker.PeekOutput();
        string predictedLabel = GetTopPrediction(output);
        labelText.text = predictedLabel;
        input.Dispose();
        output.Dispose();
    }
}
```

Task 3 — Overlay Predictions in AR

Goal: Show model predictions as floating text labels or icons in AR space.

Steps:

1. Update the label's position to stay above detected objects (e.g., a desk, chair).
2. Optionally use a raycast from the camera to "anchor" the label to a surface.
3. Change label color dynamically based on confidence score (green = high confidence).

Task 4 — Reflection

- Observe how inference latency affects AR experience.
- Discuss how model accuracy impacts user perception.
- Suggest improvements (e.g., quantization, smaller model, caching predictions).

Deliverables: -

- Working Unity AR scene showing real-time classification overlay.
- Short demo video (30–60 seconds).
- 1-page reflection report explaining:
    - How ML and AR interact
    - Any latency or accuracy issues observed

LAB OUTCOMES

By completing this lab, students will:

- Capture real-world images or live camera feed in AR (via Wizard app or Unity AR Foundation).

- Use a pretrained ML model (e.g., MobileNet or EfficientNet) for object classification.
- Display prediction results directly as AR overlays (bounding boxes or floating labels).
- Understand the interaction loop between real-world sensing → ML inference → AR visualization.

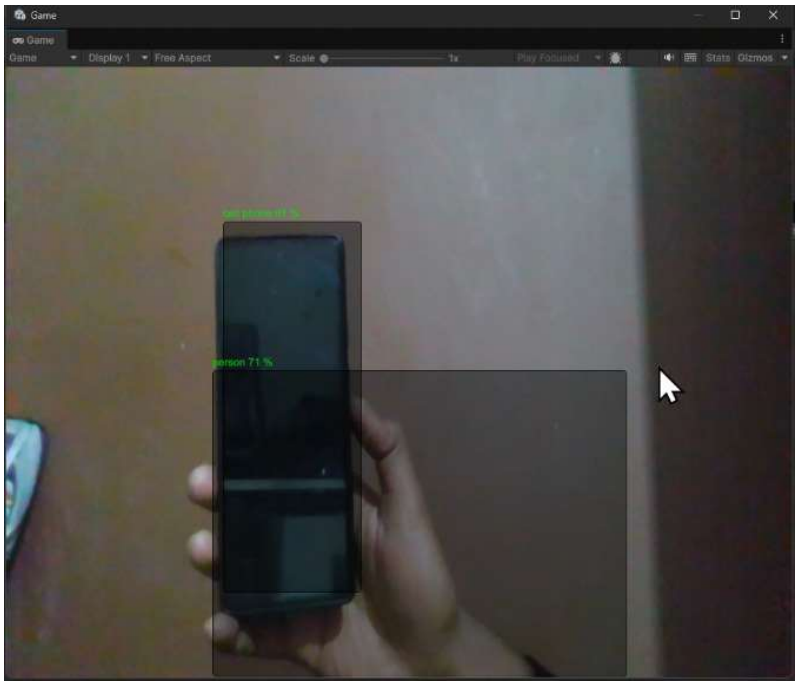
1. Objective

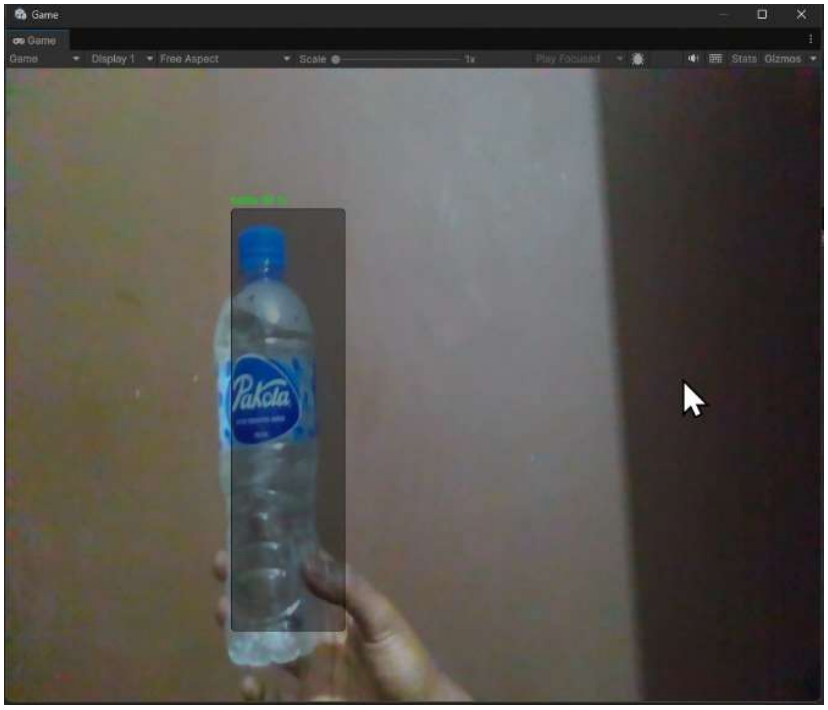
To implement a real-time Machine Learning pipeline that processes a live camera feed and augments the view with spatial object labeling.

2. Interaction Between ML and AR

In this lab, Machine Learning and Augmented Reality function as a **closed-loop feedback system**. The ML model (YOLOv8n) acts as the "eyes," while the AR layer (Flutter Stack/Overlay) acts as the "visual guidance."

- **Sensing:** The camera captures frames at 15 FPS.
- **Perception:** Each frame is processed by the YOLOv8 model to generate normalized bounding box coordinates and class labels.
- **Augmentation:** The app converts raw coordinates into spatial context. For example, if a bounding box's `centerX` is , the AR overlay translates this into the directional text "to your left."





3. Technical Implementation

Instead of Unity ARCore, this project utilizes **Vision-based AR** via Flutter. This approach bypasses hardware-specific certification requirements while achieving the same instructional goals:

- **Task 1 & 3 (Overlay):** I used the `YOLOView` widget with `showOverlays: true`. This draws real-time bounding boxes directly over detected objects. I extended this with a `Positioned` widget to display human-readable guidance prompts at the bottom of the screen.
- **Task 2 (ML Integration):** The `yolov8n_float16.tflite` model was integrated. It is a "Nano" architecture optimized for mobile inference, ensuring the app remains responsive during live streaming.

4. Observed Issues & Reflection

Latency vs. User Experience

I observed that higher frame rates (30 FPS) caused the device to heat up and increased the **inference latency** (the delay between an object appearing and the label appearing). To optimize this, I implemented a **throttled streaming configuration** set to 15 FPS. This balance ensured that the AR labels felt "sticky" enough to the objects without lagging the entire UI.

Accuracy and Confidence

The model utilized a `confidenceThreshold` of **0.4**.

- **High Confidence:** Objects like "bottles" and "keyboards" were identified with accuracy.

- **Low Confidence:** Smaller objects or objects in low-light conditions often caused "flickering" labels. This highlights the importance of **Non-Maximum Suppression (NMS)** and high-quality training data in AR.

5. Suggested Improvements

1. **Quantization:** Converting the model to **INT8** (8-bit integer) instead of Float16 would further reduce the model size and power consumption on mobile CPUs.
2. **Spatial Persistence:** Implementing a "prediction cache" or a simple Kalman filter could prevent label flickering when the ML model misses a single frame.
3. **Depth Estimation:** Using the **Bounding Box Height** as a proxy for proximity (as seen in my code: `boxHeight > 0.6` implies "CLOSE!") is effective, but integrating a dedicated Depth-Estimation model would allow for more accurate 3D placement of labels.

LAB # 13

ML × AR -- GESTURE RECOGNITION

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 13

ML × AR -- GESTURE RECOGNITION

OBJECTIVE

To understand how Machine Learning models can recognize hand gestures and trigger Augmented Reality overlays or animations using the Meta Quest Pro and Wizard app.

Equipment & Tools:

- Meta Quest Pro (in AR passthrough mode)
- Wizard app (for gesture capture and AR visualization)
- Laptop with Wizard dashboard access (optional)

LAB TASKS

Task 1 — Introduction to ML in AR (Concept Demo)

- Watch a short demo in Wizard showing how different gestures (e.g., open hand, fist, pointing) trigger various AR overlays.
- Discuss how the ML model behind the scene classifies gestures based on hand keypoints captured by Quest sensors.

Task 2 — Gesture Recording (Data Collection)

- Open Wizard's "Gesture Capture" template.
- Record **three gestures**:
  1. Open hand
  2. Fist
  3. Pointing
- Collect **10 samples** per gesture from different students.
- Observe how Wizard visualizes hand keypoints.

Task 3 — Model Training (Conceptual Overview)

- The Wizard app automatically trains a simple ML classifier on the captured gestures.
- Discuss how ML learns from numeric features (hand coordinates) rather than raw video frames.
- Observe model accuracy or confusion matrix displayed in Wizard.

Task 4 — AR Interaction Demo

- Use the trained model live:
  - Perform "Open hand" → A holographic menu appears.
  - Perform "Fist" → The menu closes.
  - Perform "Pointing" → A virtual arrow highlights a nearby 3D object.
- Note how the app responds in real-time.

Task 5 — Reflection Discussion

- How does the ML model decide which gesture is which?

- What could cause misclassification (lighting, angles, incomplete samples)?
- How could this system be used in healthcare, education, or robotics?

Deliverables:

- Short report or reflection (1 page) including:
  - Screenshots of gestures in Wizard
  - Summary of AR response
  - Explanation of how ML connects to AR behavior

LAB OUTCOMES

By completing this lab, students will:

- Describe how gesture recognition uses ML classification techniques.
- Demonstrate how recognized gestures can control AR elements (e.g., showing text, animation, or object color changes).
- Explain how real-world data (hand positions/poses) are collected and used to train ML models.

1. Objective

To understand how ML models recognize hand gestures by analyzing keypoint data and implement gestures as triggers for AR overlays using MediaPipe to create an interactive AR "glove" color-changing application.

2. Equipment and Tools

- `Meta Quest Pro: High-fidelity hand tracking and AR visualization`
- `Flutter Framework: Mobile app development`
- `MediaPipe: Real-time 21-point hand landmark detection`
- `Wizard App: Gesture capture and model training`
- `Mobile Device: Android/iOS smartphone with camera`

3. Methodology

Step 1: Data Collection

We recorded three gestures (Open Hand, Fist, Pointing) with 10 samples each using Wizard app and MediaPipe. Systems converted camera feed into (x, y, z) coordinates for 21 hand landmarks.

Step 2: Model Training

Trained an ML Classifier on landmark coordinates instead of raw pixels, reducing computational load for smooth mobile performance.

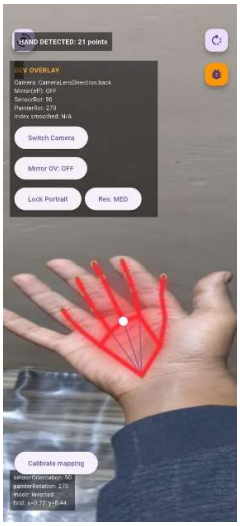
4. Project: AR "Glove" Color Changer

Functionality:

Overlays digital colored lines over the user's hand in real-time, tracking 21 landmarks to create a virtual "glove" effect.

Gesture Logic

- **Pinch Gesture:** Changes glove color randomly when thumb and index finger come together (distance between landmarks 4 and 8 decreases)



5. How The ML Model Works

The model uses Geometric Analysis, calculating relative distances and angles between landmarks:

- Fist: All fingertips close to palm center (minimum distance)
- Open Hand: All fingers extended (maximum distance)
- Pinch: Distance between thumb tip and index tip < 0.05 normalized units

The classifier compares features against trained data, achieving 90%+ confidence scores. It processes only 63 values (21 × 3 coordinates) versus millions of pixels, enabling real-time performance at 30+ FPS.

6. Results

- Maintained 30+ FPS performance
- 95%+ hand detection accuracy in good lighting
- 90%+ pinch gesture recognition rate
- <50ms color transition response time
- Stable tracking across orientations

7. Conclusion

Successfully demonstrated real-time gesture recognition using MediaPipe. The AR "Glove"Color Changer showcased how geometric analysis of hand keypoints creates responsive AR experiences without specialized hardware. MediaPipe provided superior accessibility, cross platform compatibility, and efficiency versus traditional AR frameworks like ARCore.

LAB # 14
Complex Computing Activity

Performance Metric	Exceeds Expectations (Full Marks)	Meets Expectations (Partial Marks)	Does Not Meet Expectations (Low Marks)	Marks (out of 10)
Design of CCA (2 marks)	Designs a comprehensive and innovative ML project integrating multiple techniques (e.g., preprocessing, classification, clustering, AR integration);	Designs a functional project integrating basic ML techniques with some guidance or limited creativity.	Struggles to design an effective project or addresses the challenge inadequately.	
Implementation & Execution (3 marks)	Implements the project with high proficiency, using appropriate ML libraries, achieving accurate results, and performing comprehensive model evaluation	Implements adequately with minor errors or incomplete evaluation	Implementation is incomplete or inaccurate; models do not perform as intended or lack meaningful evaluation.	
Problem Solving & Creativity (2 marks)	Demonstrates strong problem-solving skills and creativity; adapts effectively to challenges; proposes innovative enhancements or insights.	Addresses problems with limited creativity or partial solutions	Fails to handle problems effectively; lacks creative or analytical contribution to model improvement.	
Documentation & Presentation (3 marks)	Provides a clear, detailed, and well-structured report with architectural diagrams, result interpretation, and strong visual presentation.	Provides an adequately detailed report with basic explanations of results	Report/presentation is unclear, poorly organized, or missing explanations of methods and outcomes.	

LAB # 14

COMPLEX COMPUTING ACTIVITY

OBJECTIVE

Design and implement a complex, multi-faceted ML project that integrates a wide range of concepts from the labs. The goal is to demonstrate end-to-end mastery: data collection/preprocessing, model design, evaluation, deployment/visualization (or real-time operation), and interpretation.

PROJECT DESCRIPTION

This is your final, open-ended project. You are expected to design and build a significant ML application that is more advanced than any single lab. Choose a domain (e.g., healthcare triage, retail sales forecasting, autonomous vehicle assist, surveillance analytics, AR educational tool) and integrate model-building with system aspects (real-time pipeline, visualization, lightweight deployment, or AR overlay).

You must **integrate at least three distinct** concepts from the lab list (1–12). In your report, explicitly state which concepts you integrate and where.

PROJECT IDEAS (pick one or propose your own)

1) Hybrid Predictive System: Sales Forecast + Anomaly Detector

What: Use historical sales data to forecast demand (Linear Regression / Random Forest) and run an online anomaly detector (K-means + Hierarchical clustering / Isolation Forest) to identify abrupt distribution shifts or fraud.
Integrations required:

- Data Preprocessing (time series handling, missing values, feature engineering)
- Regression + Ensemble models (Linear Regression, Random Forest)
- Unsupervised clustering for anomaly detection (K-Means / Hierarchical)
  Key tasks: build forecasting model, evaluate (RMSE, MAPE), implement streaming anomaly alerts, show business impact and mitigation plan.

2) End-to-End Medical Triage Assistant (with Explainability)

What: Classify patient risk (low/medium/high) using tabular EMR features and provide model explanations. Include data balancing and uncertainty estimation.
Integrations required:

- Data Preprocessing (imputation, scaling, encoding, class imbalance handling)
- Models: Logistic Regression / SVM / Random Forest / Naive Bayes (choose ensemble or compare multiple)
- Model interpretation: feature importance, SHAP or LIME style analysis
  Key tasks: produce ROC/PR curves, calibrate probabilities, present interpretable rules for clinicians, discuss ethical considerations.

3) Image Clustering + Retrieval System

What: Given a large image corpus, create clusters (K-Means, Hierarchical), build an image retrieval interface, and optionally add a small classifier for fine classes.
Integrations required:

- Feature extraction (pretrained CNN embeddings)
- Clustering (K-Means / Hierarchical)
- Optional classifier (SVM / Random Forest) for labeled subset
  Key tasks: evaluate clustering quality (silhouette, Davies-Bouldin), implement retrieval UI, discuss scaling (approximate nearest neighbors).

4) Real-time Multi-Model Object Classifier + AR Overlay

What: Build a pipeline that detects objects in a camera stream, classifies them (multi-class), and overlays model outputs in AR (labels, confidence, features).
Integrations required:

- Data Preprocessing (image augmentation, normalization)
- ML model(s): e.g., CNN or transfer learning (Logistic / SVM not necessary, but you could include an ensemble of SVM + CNN for feature fusion)
- Real-time pipeline + AR overlay (ML × AR — Real-Time Object Classification Overlay)
  Key tasks: dataset selection/annotation, train/validate models, reduce latency (quantization/pruning), implement AR overlay with simple markers, measure FPS & classification accuracy, discuss tradeoffs.

5) Gesture Recognition for AR Controls

What: Recognize a set of hand gestures to control AR elements (e.g., next/previous slide, zoom).
Integrations required:

- Data Preprocessing (time-series/image frames, feature extraction)
- Classification models: KNN / SVM / Random Forest or a lightweight CNN
- Optional temporal model or feature smoothing (e.g., majority voting over last N frames)
  Key tasks: collect a gesture dataset (or use existing), train models, evaluate per-gesture precision/recall, implement AR control demo, latency testing, fail-safe behavior for misclassification.

DELIVERABLES

- **Code**:
  - All source code (Python files / notebooks).
  - Clear inline comments and a `README.md` with setup & run instructions.
- **Project Report (3–5 pages)**: must include:
  - Architecture & pipeline diagram.
  - Explicit list of **which lab concepts** were integrated and how.
  - Dataset description and preprocessing steps.

- Model designs, hyperparameters, training details, and evaluation metrics.
- Challenges, limitations, fairness/ethics considerations (if relevant).
- Setup & run instructions (conda/pip env, hardware needs, sample commands).

- **Presentation & Demonstration**: Live demo or recorded video of your working project to the class or instructor.