Experiment No: 01

Experiment Name: Introduction to Machine Learning and Python Environment.

Theory: Machine Learning is a branch of Artificial Intelligence that enables systems to learn from data and make predictions or decisions without being explicitly programmed. Python is widely used for Machine Learning due to its simplicity and availability of powerful libraries such as NumPy, Pandas, Scikit-learn, and Matplotlib. This experiment introduces the basic Machine Learning environment setup and fundamental data handling techniques.

Notebook:

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
```

[14] ✓ 0.0s

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

```
from sklearn.model_selection import train_test_split
X = df
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
print(X_train.shape, X_test.shape)
```

[15] ✓ 0.0s

```
(120, 4) (30, 4)
```

Conclusion: This experiment helped in understanding the basics of Machine Learning and setting up the Python environment. It also demonstrated how to load a dataset, perform basic exploration, and split data into training and testing sets, which are essential steps for building Machine Learning models.

Experiment No: 02

Experiment Name: Data Preprocessing and Encoding Categorical Data.

Theory: Data preprocessing is a crucial step in machine learning because real-world data is often incomplete, inconsistent, or unstructured. Proper preprocessing improves data quality and helps machine learning models perform better. Common preprocessing tasks include handling missing values and encoding categorical variables into numerical form.

Notebook:

```
import pandas as pd
import numpy as np
df = pd.read_csv(r'datasets\DataPreprocessing.csv')
df.isnull().sum()
```

[33] ✓ 0.0s

```
Country      5
Age          9
Salary       4
Purchased    0
dtype: int64
```

```
df['Age'] = df['Age'].fillna(df['Age'].mean())
df['Salary'] = df['Salary'].fillna(df['Salary'].median())
df['Country'] = df['Country'].fillna(df['Country'].mode()[0])
df.isnull().sum()
```

[32] ✓ 0.0s

```
Country      0
Age          0
Salary       0
Purchased    0
dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Country_Encoded'] = le.fit_transform(df['Country'])
encoded_df = pd.get_dummies(df['Country'])
print(df.head())
print('One Hot Encoded DataFrame:\n', encoded_df.head())
```

[48] ✓ 0.0s

```
   Country   Age   Salary Purchased  Country_Encoded
0   France  44.0  72000.0        No                0
1    Spain  27.0  48000.0       Yes                2
2  Germany  30.0  54000.0        No                1
3    Spain  38.0  61000.0        No                2
4  Germany  40.0      NaN       Yes                1
One Hot Encoded DataFrame:
    France  Germany  Spain
0    True    False  False
1   False    False   True
2   False     True  False
3   False    False   True
4   False     True  False
```

Conclusion: In this lab, data preprocessing techniques were implemented successfully. Missing values were identified and handled using different strategies such as mean, median, and mode filling. Categorical variables were converted into numerical form using label encoding and one-hot encoding. These preprocessing steps are essential to ensure clean and machine-learning-ready data.

Experiment No: 03

Experiment Name: Exploratory Data Analysis (EDA).

Theory: Exploratory Data Analysis (EDA) is an important step in the machine learning pipeline. It helps in understanding the dataset before applying any model. Through EDA, we can identify trends, relationships between variables, missing values, and outliers using statistical methods and visualizations. Proper EDA leads to better feature selection and improved model performance.

Notebook:

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
df.info()
df.describe()
df.tail()
```

[18] ✓ 0.0s

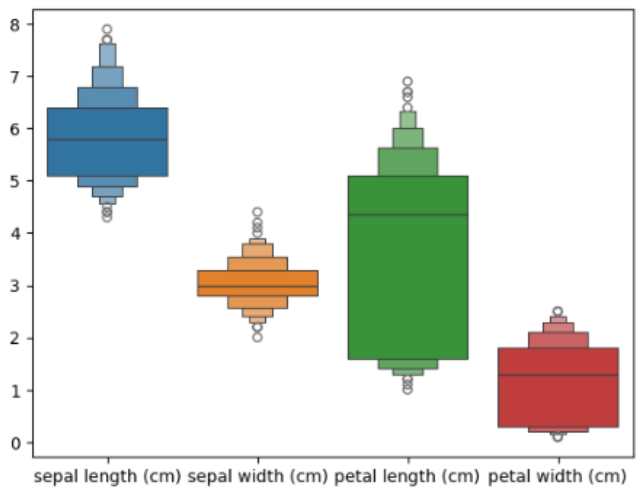
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

```
import matplotlib.pyplot as plt
import seaborn as sns
df.isnull().sum()
sns.boxenplot(data=df)
```

[26] ✓ 0.4s

```
<Axes: >
```

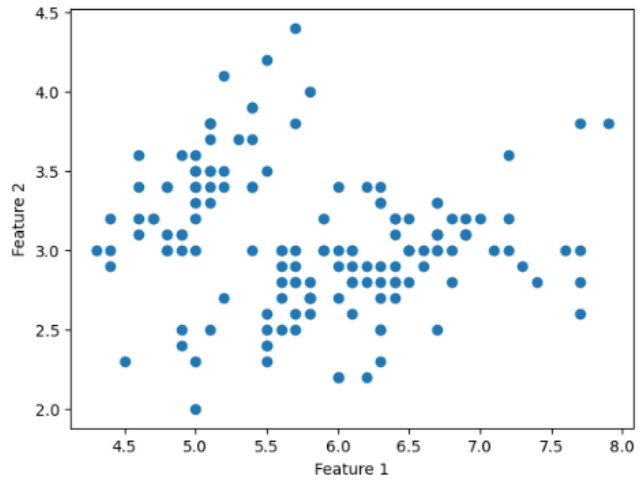


	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal length (cm)	0.871754	-0.428440	1.000000	0.962865
petal width (cm)	0.817941	-0.366126	0.962865	1.000000

```
plt.scatter(df.iloc[:,0], df.iloc[:,1])
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

[22] ✓ 0.3s

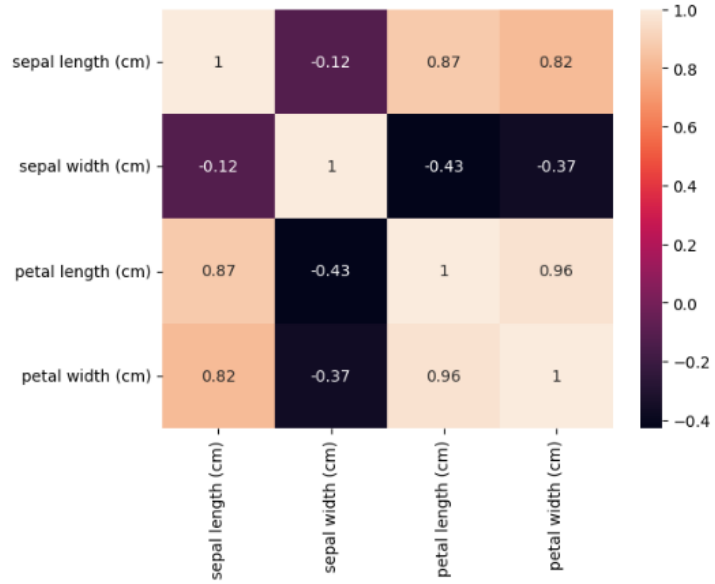
```
Text(0, 0.5, 'Feature 2')
```




```
sns.heatmap(df.corr(), annot=True)
```

[23] ✓ 0.6s

```
<Axes: >
```



Conclusion: In this lab, data preprocessing techniques were implemented successfully. Missing values were identified and handled using different strategies such as mean, median, and mode filling. Categorical variables were converted into numerical form using label encoding and one-hot encoding. These preprocessing steps are essential to ensure clean and machine-learning-ready data.

Experiment No: 04

Experiment Name: Linear Regression

Theory: Linear Regression is a supervised learning algorithm used to predict a continuous target variable based on one or more input features. In this experiment, the California Housing dataset is used to predict house prices based on features such as median income, house age, number of rooms, population, and location. Model performance is evaluated using MAE, MSE, RMSE, and R^2 score, and residual plots are used to validate model assumptions.

Notebook:

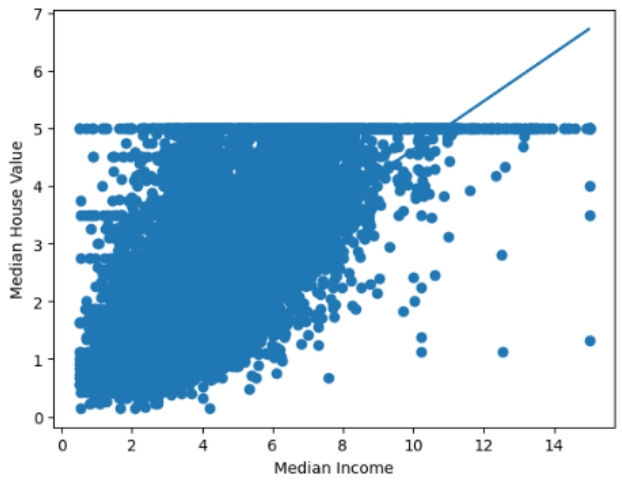
```
import pandas as pd
from sklearn.datasets import fetch_california_housing

# Load dataset
california = fetch_california_housing()
df = pd.DataFrame(california.data, columns=california.feature_names)
df['MedHouseValue'] = california.target

df.head()
```

[54] ✓ 0.0s

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseValue
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

X = df[['MedInc']]
y = df['MedHouseValue']

model = LinearRegression()
model.fit(X, y)

y_pred = model.predict(X)

# Visualization
plt.scatter(X, y)
plt.plot(X, y_pred)
plt.xlabel("Median Income")
plt.ylabel("Median House Value")
plt.show()
```

[55] ✓ 0.3s



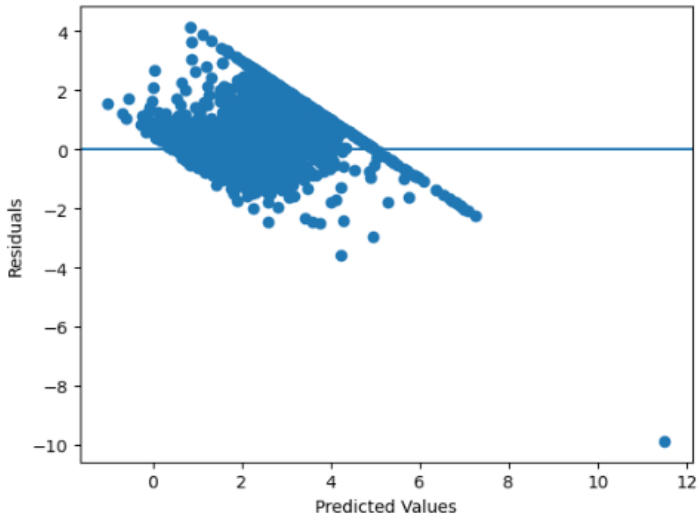
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r2)
```

[57] ✓ 0.0s

```
MAE: 0.5332001304956558
MSE: 0.555891598695244
RMSE: 0.7455813830127761
R2 Score: 0.5757877060324511
```

```
residuals = y_test - y_pred

plt.scatter(y_pred, residuals)
plt.axhline(y=0)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.show()
```

[58] ✓ 0.2s




```
# new_data must contain 8 values to match the California Housing features
# Example: [MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Lat, Long]
new_house = [[8.3252, 41, 6.984, 1.023, 322, 2.555, 37.88, -122.23]]
predicted_price = model.predict(new_house)

print("Predicted House Value:", predicted_price)
```

[59] ✓ 0.0s

```
Predicted House Value: [4.15132633]
```

Conclusion: In this experiment, linear regression was successfully applied to the California Housing dataset for house price prediction. Both simple and multiple linear regression models were implemented and evaluated using standard performance metrics. Residual analysis confirmed the validity of the regression assumptions. This experiment demonstrates the effectiveness of linear regression for real-world regression problems.

Experiment No: 05

Experiment Name: Logistic Regression

Theory: Logistic Regression is a supervised machine learning algorithm used for classification problems. It predicts the probability that a given input belongs to a particular class using a logistic (sigmoid) function. Binary Logistic Regression is used when there are two classes. Model performance is evaluated using accuracy, confusion matrix, classification report, ROC curve, and AUC score.

Notebook:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
print(X)
print(y)
```

[11] ✓ 0.1s

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
...
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
X_binary = X[y != 2]
y_binary = y[y != 2]
X_train, X_test, y_train, y_test = train_test_split(
X_binary, y_binary, test_size=0.2, random_state=42)
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

[12] ✓ 0.0s

```
Accuracy: 1.0
```

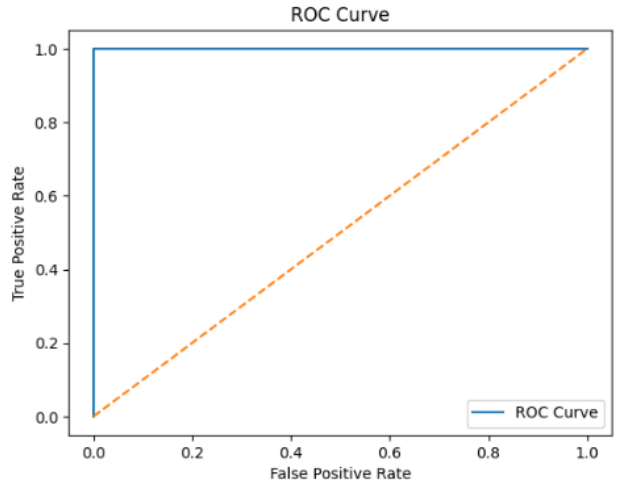
```
from sklearn.metrics import confusion_matrix, classification_report
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

[13] ✓ 0.0s

```
Confusion Matrix:
 [[12  0]
 [ 0  8]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      1.00      1.00         8

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

```
from sklearn.metrics import roc_curve, roc_auc_score
# Probability estimates
y_prob = log_reg.predict_proba(X_test)[:, 1]
# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.plot(fpr, tpr, label="ROC Curve")
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
# AUC score
print("AUC Score:", roc_auc_score(y_test, y_prob))
```

[14] ✓ 0.2s




```
AUC Score: 1.0
```

```
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=42)
multi_log = LogisticRegression(max_iter=200)
multi_log.fit(X_train, y_train)
y_multi_pred = multi_log.predict(X_test)
print("Multiclass Accuracy:", accuracy_score(y_test, y_multi_pred))
```

[15] ✓ 0.0s

```
Multiclass Accuracy: 1.0
```

Conclusion: In this lab, logistic regression was successfully implemented for both binary and multiclass classification tasks. The model's performance was evaluated using accuracy, confusion matrix, classification report, ROC curve, and AUC score. Logistic regression proved to be an effective and interpretable classification algorithm for structured datasets.

Experiment No: 06

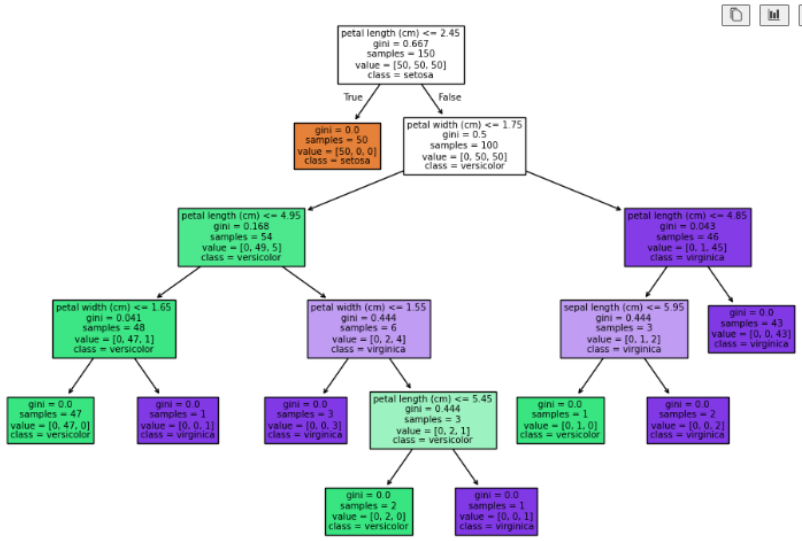
Experiment Name: Decision Trees

Theory: A Decision Tree is a supervised machine learning algorithm used for both classification and regression. It works by splitting the dataset into smaller subsets based on feature values, forming a tree-like structure of decisions. Decision Trees are easy to interpret but can easily overfit if not properly controlled. Overfitting can be reduced using pruning techniques such as limiting tree depth and minimum samples per split or leaf.

Notebook:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
iris = load_iris()
X = iris.data
y = iris.target
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X, y)
plt.figure(figsize=(12,8))
plot_tree(dt_model, feature_names=iris.feature_names,
          class_names=iris.target_names, filled=True)
plt.show()
```

[2] ✓ 0.5s




```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
dt_model.fit(X_train, y_train)
y_pred = dt_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='weighted'))
print("Recall:", recall_score(y_test, y_pred, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred, average='weighted'))
```

[3] ✓ 0.0s

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
```

Conclusion: In this lab, Decision Tree classifiers were implemented and evaluated using the Iris dataset. The effects of different splitting criteria and pruning parameters were observed. Pruning successfully reduced overfitting while maintaining good classification accuracy. Decision Trees are powerful, interpretable models suitable for both classification and regression tasks.

Experiment No: 07

Experiment Name: Random Forests and Ensemble Methods

Theory: Ensemble learning combines multiple models to improve prediction accuracy and reduce overfitting. Random Forest is an ensemble method that builds multiple decision trees using bootstrapped samples and random feature selection. The final prediction is obtained by majority voting (classification) or averaging (regression). Random Forest models are robust, accurate, and less prone to overfitting compared to single decision trees.

Notebook:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))
```

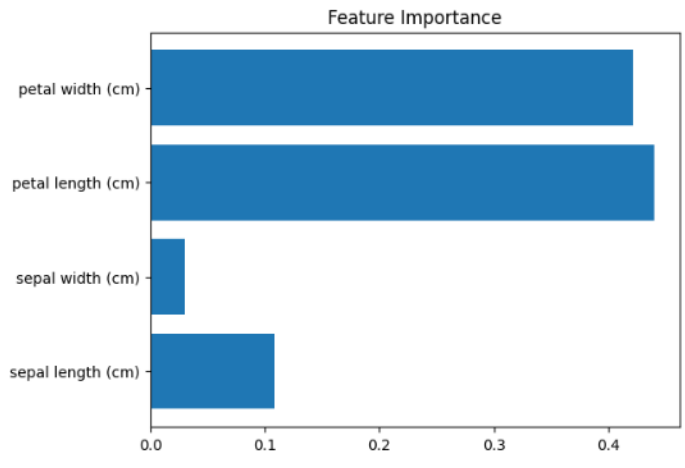
[14] ✓ 0.5s

```
Random Forest Accuracy: 1.0
```

```
importances = rf.feature_importances_
for f, i in zip(iris.feature_names, importances):
    print(f, ":", i)
plt.barh(iris.feature_names, importances)
plt.title("Feature Importance")
plt.show()
```

[15] ✓ 0.1s

```
sepal length (cm) : 0.10809762464246378
sepal width (cm) : 0.030386812473242528
petal length (cm) : 0.43999397414456937
petal width (cm) : 0.4215215887397244
```



```
rf_tuned = RandomForestClassifier(
n_estimators=200, max_depth=5,
min_samples_split=5, min_samples_leaf=3,
random_state=42)
rf_tuned.fit(X_train, y_train)
print("Tuned RF Accuracy:", accuracy_score(y_test, rf_tuned.predict(X_test)))
```

[16] ✓ 0.4s

```
Tuned RF Accuracy: 1.0
```

```
housing = fetch_california_housing()
Xh = housing.data
yh = housing.target
Xh_train, Xh_test, yh_train, yh_test = train_test_split(
Xh, yh, test_size=0.2, random_state=42)
rf_reg = RandomForestRegressor(random_state=42)
rf_reg.fit(Xh_train, yh_train)
yh_pred = rf_reg.predict(Xh_test)
print("MSE:", mean_squared_error(yh_test, yh_pred))
print("RMSE:", np.sqrt(mean_squared_error(yh_test, yh_pred)))
print("R2 Score:", r2_score(yh_test, yh_pred))
```

[17] ✓ 16.0s

```
MSE: 0.2553684927247781
RMSE: 0.5053399773665033
R2 Score: 0.8051230593157366
```

```
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
print("Decision Tree Accuracy:", accuracy_score(y_test, dt.predict(X_test)))
print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))
```

[18] ✓ 0.0s

```
Decision Tree Accuracy: 1.0
Random Forest Accuracy: 1.0
```

Conclusion: In this lab, Random Forest models were implemented for both classification and regression tasks. Feature importance analysis highlighted the most influential features. Hyperparameter tuning improved model performance and reduced overfitting. Compared to a single Decision Tree, Random Forest achieved better accuracy and generalization, demonstrating the strength of ensemble learning methods.

Experiment No: 08

Experiment Name: Support Vector Machine (SVM)

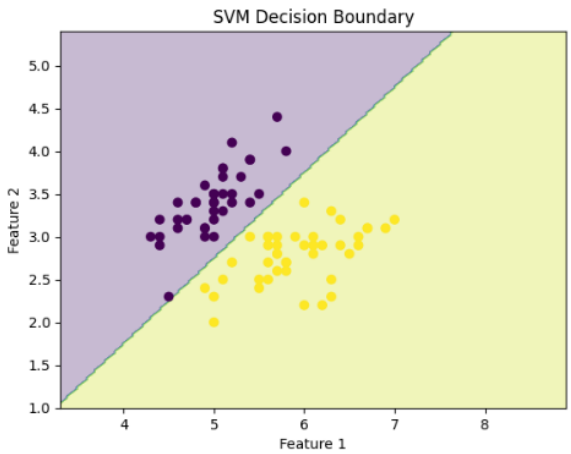
Theory: Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression. It finds an optimal hyperplane that maximizes the margin between different classes. SVM can handle linear and non-linear data using different kernel functions such as linear, polynomial, and RBF.

Notebook:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = load_iris()
X = iris.data[:, :2]
y = iris.target
mask = y < 2
X, y = X[mask], y[mask]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
svm_linear = SVC(kernel='linear', C=1)
svm_linear.fit(X_train, y_train)
print("Linear Kernel Accuracy:",
      accuracy_score(y_test, svm_linear.predict(X_test)))
svm_rbf = SVC(kernel='rbf', C=1, gamma='scale')
svm_rbf.fit(X_train, y_train)
print("RBF Kernel Accuracy:",
      accuracy_score(y_test, svm_rbf.predict(X_test)))
X = iris.data
y = iris.target
svm_multi = SVC(kernel='rbf')
svm_multi.fit(X, y)
xx, yy = np.meshgrid(
    np.linspace(X[:,0].min()-1, X[:,0].max()+1, 200),
    np.linspace(X[:,1].min()-1, X[:,1].max()+1, 200)
)
Z = svm_linear.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X_train[:,0], X_train[:,1], c=y_train)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("SVM Decision Boundary")
plt.show()
```

[1] ✓ 5.6s

```
Linear Kernel Accuracy: 1.0
RBF Kernel Accuracy: 1.0
```



Conclusion: SVM is an effective classification algorithm that performs well on both linear and non-linear datasets. Kernel selection and hyperparameter tuning significantly affect model accuracy and decision boundaries. SVM can efficiently handle binary and multi-class classification problems.

Experiment No: 09

Experiment Name: Logistic Regression

Theory: K-Nearest Neighbors (KNN) is a supervised, instance-based learning algorithm. It classifies a data point based on the majority class (or average value in regression) of its K nearest neighbors using a distance metric such as Euclidean or Manhattan. Model performance depends heavily on the choice of K and distance metric.

Notebook:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
iris = load_iris()
X = iris.data[:, :2]   # 2D for visualization
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)
print("Classification Accuracy:",
      accuracy_score(y_test, knn.predict(X_test)))

for k in [3, 5, 7]:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    print(f"K={k} Accuracy:",
          accuracy_score(y_test, knn.predict(X_test)))

cal = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(
    cal.data, cal.target, test_size=0.2, random_state=42
)
knn_reg = KNeighborsRegressor(n_neighbors=5)
knn_reg.fit(X_train, y_train)
y_pred = knn_reg.predict(X_test)

print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R2 Score:", r2_score(y_test, y_pred))
```

[2] ✓ 0.1s

```
Classification Accuracy: 0.8
K=3 Accuracy: 0.8
K=5 Accuracy: 0.8
K=7 Accuracy: 0.7666666666666667
RMSE: 1.0576778270706204
R2 Score: 0.14631049965900345
```

Conclusion: KNN is a simple yet effective algorithm for both classification and regression. The choice of K and distance metric significantly affects model performance. Proper tuning helps balance overfitting and underfitting.

Experiment No: 10

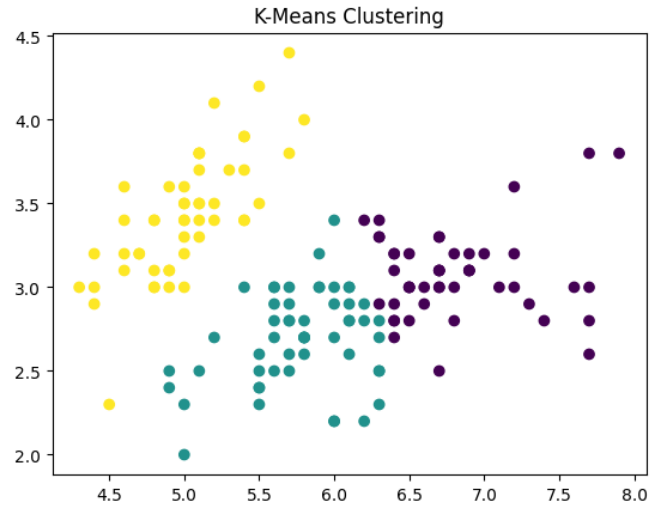
Experiment Name: Unsupervised Learning – Clustering

Theory: Clustering is an unsupervised learning technique that groups similar data points without using class labels. Popular clustering algorithms include K-Means, Hierarchical Clustering, and DBSCAN. Clustering quality can be evaluated using metrics such as Silhouette Score and Davies–Bouldin Index.

Notebook:

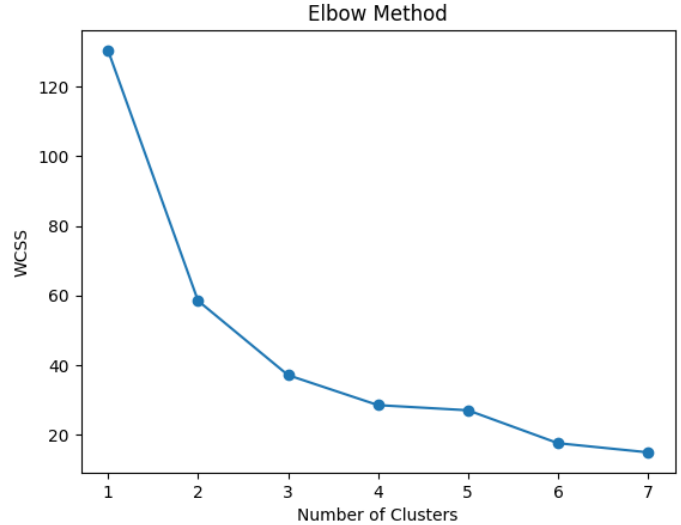
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score, davies_bouldin_score
from scipy.cluster.hierarchy import dendrogram, linkage
iris = load_iris()
X = iris.data[:, :2]
kmeans = KMeans(n_clusters=3, random_state=42)
labels_kmeans = kmeans.fit_predict(X)
plt.scatter(X[:,0], X[:,1], c=labels_kmeans)
plt.title("K-Means Clustering")
plt.show()
```

[9] ✓ 0.4s



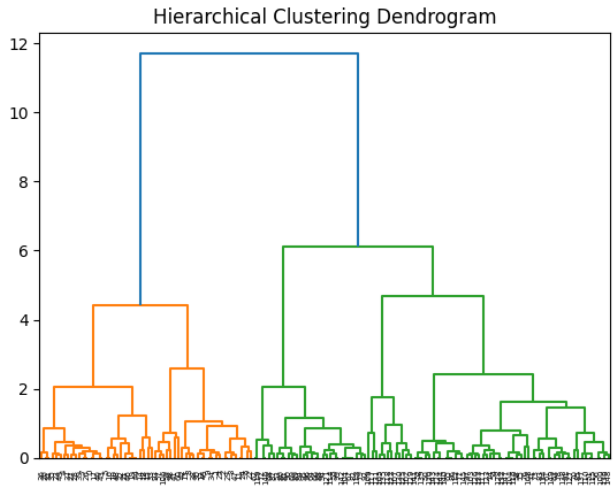

```
wcss = []
for k in range(1, 8):
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X)
    wcss.append(km.inertia_)
plt.plot(range(1, 8), wcss, marker='o')
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("Elbow Method")
plt.show()
```

[10] ✓ 0.1s



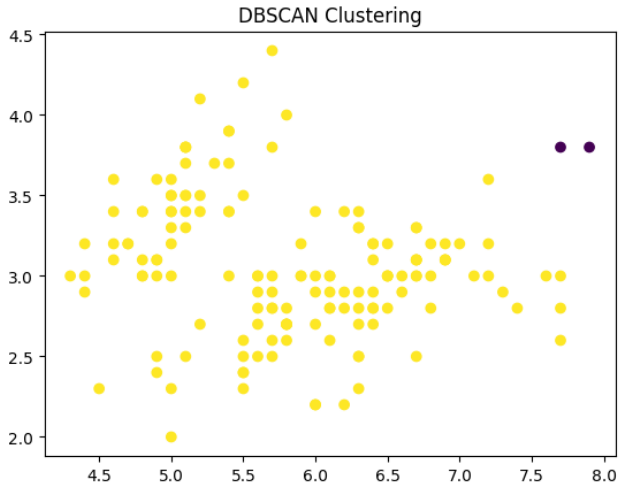
```
Z = linkage(X, method='ward')
dendrogram(Z)
plt.title("Hierarchical Clustering Dendrogram")
plt.show()
```

[11] ✓ 0.9s




```
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels_dbscan = dbscan.fit_predict(X)

plt.scatter(X[:,0], X[:,1], c=labels_dbscan)
plt.title("DBSCAN Clustering")
plt.show()
```

[12] ✓ 0.2s




```
print("K-Means Silhouette:",
      silhouette_score(X, labels_kmeans))
print("K-Means DB Index:",
      davies_bouldin_score(X, labels_kmeans))
```

[13] ✓ 0.0s

```
K-Means Silhouette: 0.4450525692083638
K-Means DB Index: 0.7675522686571644
```

Conclusion: Unsupervised clustering algorithms were successfully applied to group unlabeled data. K-Means performed well for spherical clusters, hierarchical clustering provided clear visual interpretation, and DBSCAN effectively identified noise. Clustering evaluation metrics helped compare algorithm performance.

Experiment No: 11

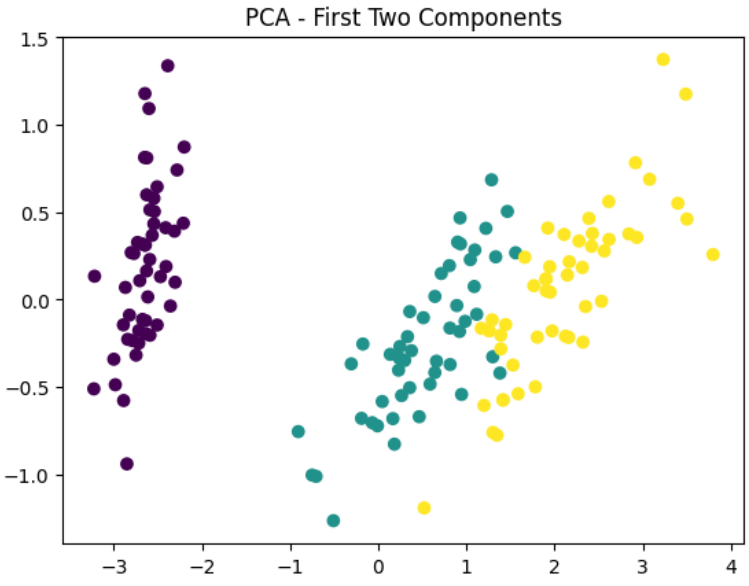
Experiment Name: Dimensionality Reduction

Theory: Dimensionality reduction transforms high-dimensional data into a lower-dimensional space while preserving significant information. PCA: Projects data along directions of maximum variance. t-SNE: Non-linear technique for visualizing high-dimensional data in 2D or 3D. Dimensionality reduction helps in visualization, noise reduction, and improving classifier performance.

Notebook:

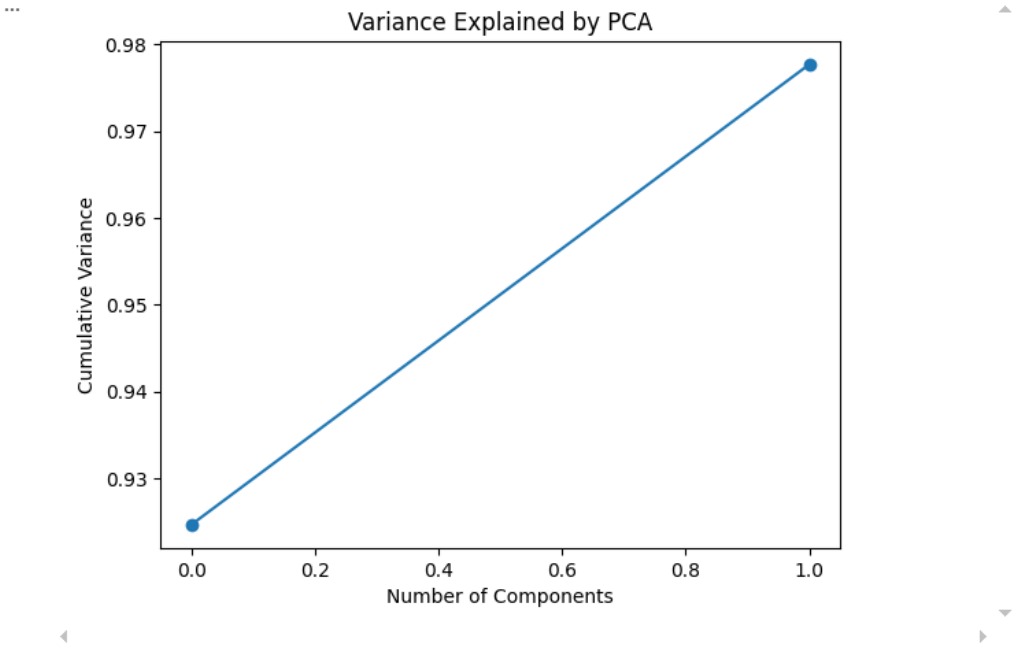
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
iris = load_iris()
X = iris.data
y = iris.target

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
plt.scatter(X_pca[:,0], X_pca[:,1], c=y)
plt.title("PCA - First Two Components")
plt.show()
```

[6] ✓ 0.1s Python



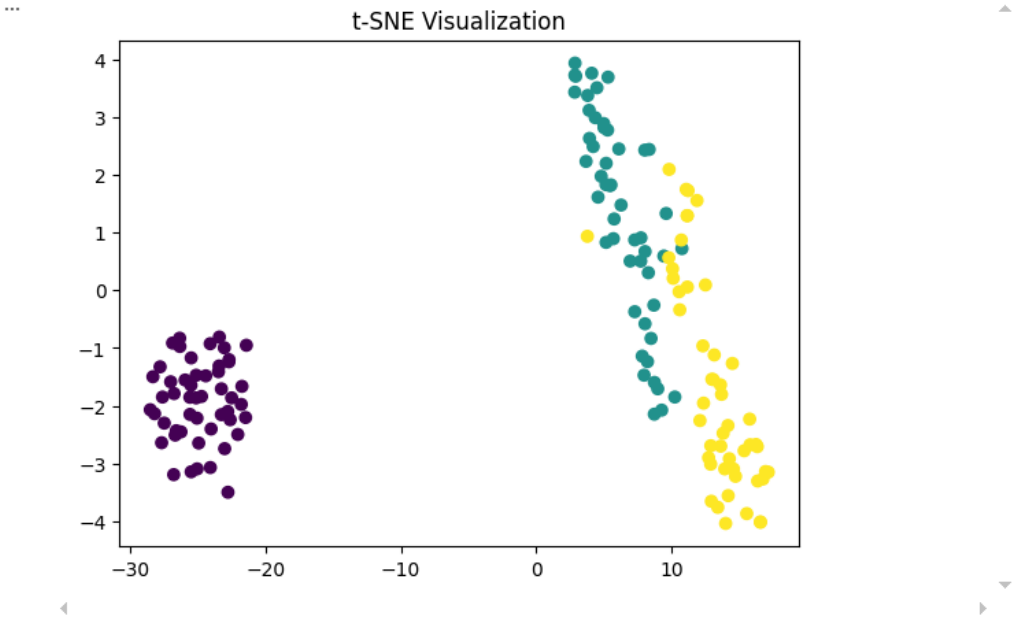

```
cum_var = np.cumsum(pca.explained_variance_ratio_)
plt.plot(cum_var, marker='o')
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Variance")
plt.title("Variance Explained by PCA")
plt.show()
```

[7] ✓ 0.2s Python




```
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)
plt.scatter(X_tsne[:,0], X_tsne[:,1], c=y)
plt.title("t-SNE Visualization")
plt.show()
```

[8] ✓ 1.1s Python



Conclusion: Dimensionality reduction techniques effectively reduce feature space while preserving essential information. PCA is useful for preprocessing and retaining variance, whereas t-SNE excels at visualizing high-dimensional data in 2D with clear cluster separation. Both methods are complementary depending on the application.

Experiment No: 12

Experiment Name: Model Evaluation and Cross-Validation

Theory: Model evaluation ensures that a machine learning model generalizes well to unseen data. Common evaluation metrics include **accuracy, precision, recall, F1-score, ROC curve, and AUC. Cross-validation** splits data into multiple folds to assess model stability. Stratified K-Fold maintains class distributions for classification tasks. **GridSearchCV** is used for hyperparameter tuning to find the best model configuration.

Notebook:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("=== Train-Test Split Metrics ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='weighted'))
print("Recall:", recall_score(y_test, y_pred, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred, average='weighted'))
```

✓ 0.0s

```
=== Train-Test Split Metrics ===
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores_kf = cross_val_score(model, X, y, cv=kf)
print("\nK-Fold Accuracy:", scores_kf.mean())
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores_skf = cross_val_score(model, X, y, cv=skf)
print("Stratified K-Fold Accuracy:", scores_skf.mean())
```

✓ 0.3s

```
K-Fold Accuracy: 0.9733333333333334
Stratified K-Fold Accuracy: 0.9666666666666668
```

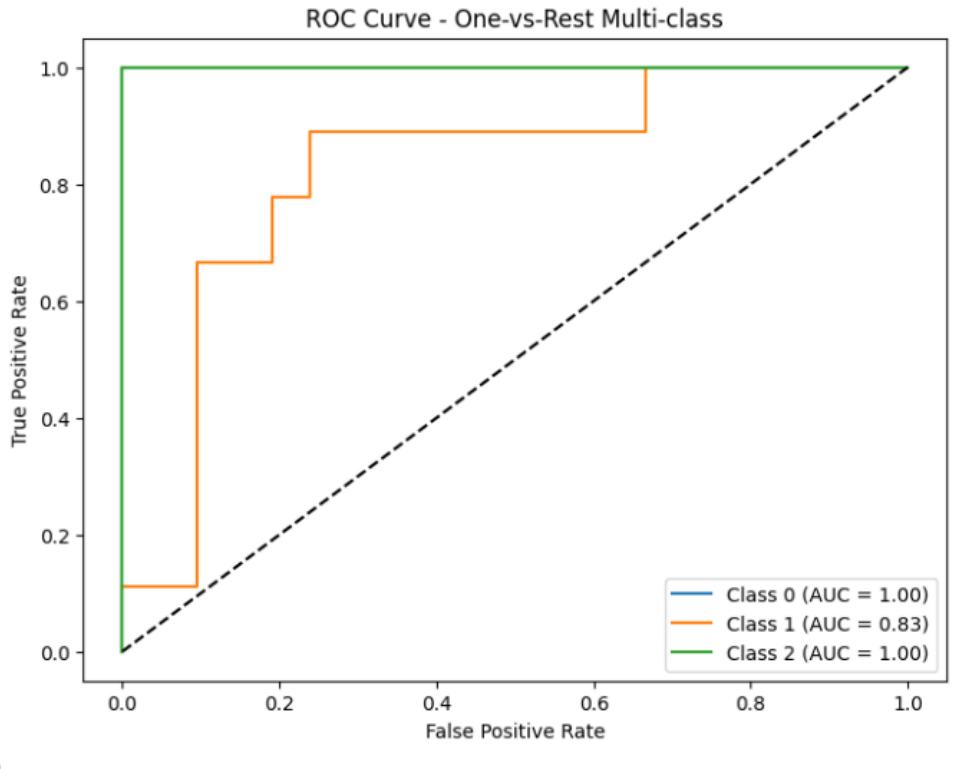
```python
param_grid = {'C': [0.1, 1, 10]}
grid = GridSearchCV(LogisticRegression(max_iter=200), param_grid, cv=5)
grid.fit(X, y)
print("\nBest Params:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)
```

[12] ✓ 0.5s Python

```
Best Params: {'C': 1}
Best CV Accuracy: 0.9733333333333334
```

```python
y_train_bin = label_binarize(y_train, classes=[0,1,2])
y_test_bin = label_binarize(y_test, classes=[0,1,2])
ovr_model = OneVsRestClassifier(LogisticRegression(max_iter=200))
ovr_model.fit(X_train, y_train_bin)
y_score = ovr_model.predict_proba(X_test)
plt.figure(figsize=(8,6))
for i in range(y_train_bin.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_bin[:,i], y_score[:,i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"Class {i} (AUC = {roc_auc:.2f})")

plt.plot([0,1], [0,1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - One-vs-Rest Multi-class")
plt.legend()
plt.show()
```

[13] ✓ 0.3s Python



Conclusion: Model evaluation and cross-validation are essential to ensure generalization. Stratified K-Fold preserves class distributions. GridSearchCV helps find optimal hyperparameters. ROC and AUC provide insights into classifier performance, especially for imbalanced datasets.