Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An Autonomous Institution Affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	**Semester**	V
Subject Code & Name	ICS1512 – Machine Learning Algorithms Laboratory		
Academic Year	2025–2026 (Odd)	**Batch**	2023–2028

Experiment #1: Exploring Python Libraries for Machine Learning

Aim:
To explore and understand the core functionalities of essential Python libraries — NumPy, Pandas, SciPy, Scikit-learn, and Matplotlib — for array manipulation, data preprocessing, machine learning workflows, and data visualization.

Libraries Used:

- `numpy`
- `pandas`
- `scipy`
- `scikit-learn`
- `matplotlib`
- `seaborn`

Objective:
To apply the core Python libraries on real-world datasets from UCI and Kaggle, demonstrate their features, and identify suitable machine learning models for different tasks.

Summary of Tasks:

- Performed numerical operations using NumPy: arrays, reshaping, broadcasting, statistics.
- Used Pandas for loading, cleaning, grouping, and aggregating tabular data.
- Applied SciPy functions for mathematical operations and statistical analysis.
- Built ML models with Scikit-learn: classification, regression, and feature selection.
- Visualized trends and distributions using Matplotlib and Seaborn.
- Worked with five datasets: Loan Prediction, Handwritten Digits, Spam Detection, Diabetes, and Iris.

1.Explore the various functions and methods available in the following Python libraries: Numpy, Pandas, Scipy, Scikit-learn, Matplotlib. Understand the key operations such as array manipulations, data preprocessing, mathematical computing, machine learning workflows, and data visualization

NUMPY

```
# Import Numpy
import numpy as np

# Creating basic arrays
arr = np.array([1, 2, 3, 4, 5])
print("Original 1D array:", arr)

# Array creation using various methods
a = np.array([1, 2, 3])
b = np.zeros((2, 2))
c = np.ones((3, 1))
d = np.arange(0, 10, 2)
e = np.linspace(0, 1, 5)

print("\nArray a (np.array):", a)
print("Array b (np.zeros):\n", b)
print("Array c (np.ones):\n", c)
print("Array d (np.arange):", d)
print("Array e (np.linspace):", e)

# Shape of arrays
print("\nShapes:")
print("a:", a.shape)
print("b:", b.shape)
print("c:", c.shape)
print("d:", d.shape)
print("e:", e.shape)

# Displaying data type
print("\nType of 'arr':", type(arr))         # object type
print("Data type of elements:", arr.dtype)  # data type of elements

# Creating 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print("\n2D Array:\n", arr2)

# Array dimensions
print("Dimensions of arr:", arr.ndim)
print("Dimensions of arr2:", arr2.ndim)

# Indexing (1D and 2D)
print("\n1D Indexing: arr[0] =", arr[0])
print("2D Indexing: 2nd element of 1st row =", arr2[0, 1])
print("Sum of arr[1] + arr[3] =", arr[1] + arr[3])

# Negative indexing
```

```
print("\nNegative indexing (last element):", arr[-1])

# Slicing
print("\nSlicing arr[1:3]:", arr[1:3])
print("Slicing arr[1:]:", arr[1:])
print("Slicing arr[:3]:", arr[:3])
print("Negative slicing arr[-3:-1]:", arr[-3:-1])
print("Step slicing arr[1:5:2]:", arr[1:5:2])
print("Slicing 2D arr2[1,1:3]:", arr2[1, 1:3])

# Specific data type array
arr3 = np.array([1, 2, 3, 4, 5, 6], dtype='float')
print("\nArray with float dtype:", arr3)

# Changing data type
changedarr = arr3.astype('int')
print("Changed to int:", changedarr)
print("New dtype:", changedarr.dtype)

# Copy vs View
x = arr.copy()
print("\nOriginal array:", arr)
print("Copied array:", x)

a = np.array([1, 2, 3, 4])
b = a.view()
b[1] = 100
print("\nView after modifying b:", b)
print("Original a after modifying b:", a)  # reflects in a

# Reshaping
arr4 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr4.reshape(4, 3)
print("\nReshaped array:\n", newarr)

# Joining arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print("\nConcatenated:", np.concatenate([arr1, arr2]))
print("Horizontal Stack:", np.hstack([arr1, arr2]))
print("Vertical Stack:\n", np.vstack([arr1, arr2]))
print("Depth Stack:\n", np.dstack([arr1, arr2]))

# Splitting arrays
split_arr = np.array_split(arr4, 3)
print("\nSplitting arr4 into 3 parts:")
print("Part 1:", split_arr[0])
```

```
print("Part 2:", split_arr[1])
print("Part 3:", split_arr[2])

# Sorting
arr = np.array([3, 2, 0, 1])
print("\nSorted array:", np.sort(arr))

# Random number generation
from numpy import random
print("\nRandom integer (0{100):", random.randint(100))

# Broadcasting example
print("\nBroadcasting:")
arr5 = np.array([1, 2, 3])
print("Original:", arr5)
print("After adding scalar 10:", arr5 + 10)  # scalar broadcast
arr6 = np.array([[1], [2], [3]])
print("Broadcast with compatible shapes:\n", arr6 + arr5)  # row + col

# Aggregate functions
print("\nAggregate Functions:")
print("Sum:", np.sum(a))
print("Mean:", np.mean(a))
print("Min:", np.min(a))
print("Max:", np.max(a))
print("Standard Deviation:", np.std(a))
print("Variance:", np.var(a))
print("Argmax (position of max):", np.argmax(a))
print("Argmin (position of min):", np.argmin(a))

# Mathematical operations
print("\nMath Operations:")
print("Dot product of a and a:", np.dot(a, a))
print("Exponent:", np.exp(a))
print("Square Root:", np.sqrt(a))
```

OUTPUT:

```
Original 1D array: [1 2 3 4 5]

Array a (np.array): [1 2 3]
Array b (np.zeros):
 [[0. 0.]
 [0. 0.]]
Array c (np.ones):
 [[1.]
 [1.]
```

```
 [1.]]
Array d (np.arange): [0 2 4 6 8]
Array e (np.linspace): [0.   0.25 0.5  0.75 1.  ]

Shapes:
a: (3,)
b: (2, 2)
c: (3, 1)
d: (5,)
e: (5,)

Type of 'arr': <class 'numpy.ndarray'>
Data type of elements: int64

2D Array:
 [[1 2 3]
 [4 5 6]]
Dimensions of arr: 1
Dimensions of arr2: 2

1D Indexing: arr[0] = 1
2D Indexing: 2nd element of 1st row = 2
Sum of arr[1] + arr[3] = 6

Negative indexing (last element): 5

Slicing arr[1:3]: [2 3]
Slicing arr[1:]: [2 3 4 5]
Slicing arr[:3]: [1 2 3]
Negative slicing arr[-3:-1]: [3 4]
Step slicing arr[1:5:2]: [2 4]
Slicing 2D arr2[1,1:3]: [5 6]

Array with float dtype: [1. 2. 3. 4. 5. 6.]
Changed to int: [1 2 3 4 5 6]
New dtype: int64

Original array: [1 2 3 4 5]
Copied array: [1 2 3 4 5]

View after modifying b: [  1 100   3   4]
Original a after modifying b: [  1 100   3   4]

Reshaped array:
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

```
Concatenated: [1 2 3 4 5 6]
Horizontal Stack: [1 2 3 4 5 6]
Vertical Stack:
 [[1 2 3]
 [4 5 6]]
Depth Stack:
 [[[1 4]
  [2 5]
  [3 6]]]

Splitting arr4 into 3 parts:
Part 1: [1 2 3 4]
Part 2: [5 6 7 8]
Part 3: [ 9 10 11 12]

Sorted array: [0 1 2 3]

Random integer (0{100): 31

Broadcasting:
Original: [1 2 3]
After adding scalar 10: [11 12 13]
Broadcast with compatible shapes:
 [[2 3 4]
 [3 4 5]
 [4 5 6]]

Aggregate Functions:
Sum: 108
Mean: 27.0
Min: 1
Max: 100
Standard Deviation: 42.16040796766559
Variance: 1777.5
Argmax (position of max): 1
Argmin (position of min): 0

Math Operations:
Dot product of a and a: 10026
Exponent: [2.71828183e+00 2.68811714e+43 2.00855369e+01 5.45981500e+01]
Square Root: [ 1.         10.          1.73205081  2.        ]
```

PANDAS

```
import pandas as pd

# create a simple Series (like a 1D array)
```

```
data = [10, 20, 30]
series = pd.Series(data)
print("Series:\n", series)

# Series with custom labels
series = pd.Series(data, index=['a', 'b', 'c'])
print("\nSeries with custom index:\n", series)

# create a basic DataFrame (like a table with rows and columns)
data = {
    "Name": ["Alice", "Bob", "Charlie", "David", "Eve"],
    "Age": [25, 30, 35, None, 25],
    "City": ["Delhi", "Mumbai", "Chennai", "Delhii", "Delhi"]
}
df = pd.DataFrame(data)
print("\nDataFrame:\n", df)

# read data from a CSV or JSON file (if you have those files)
# df_csv = pd.read_csv("data.csv")
# print("\nCSV Data:\n", df_csv.head())
# df_json = pd.read_json("data.json")
# print("\nJSON Data:\n", df_json.head())

# look at the first and last few rows
print("\nFirst few rows:\n", df.head())
print("\nLast few rows:\n", df.tail())

# check column names and data types
print("\nColumn names:", df.columns)
print("\nData types:\n", df.dtypes)

# get basic statistics for numeric columns
print("\nSummary statistics:\n", df.describe())

# check for missing values
print("\nMissing values:\n", df.isnull())

# drop rows with missing data
df_cleaned = df.dropna()
print("\nAfter dropping missing values:\n", df_cleaned)

# or fill missing values with a default
df_filled = df.fillna("Unknown")
print("\nAfter filling missing values:\n", df_filled)

# fix formatting issues: convert 'Age' to numeric, invalid entries become NaN
df["Age"] = pd.to_numeric(df["Age"], errors='coerce')
```

```
# fix typos in 'City' column
df["City"] = df["City"].replace("Delhii", "Delhi")

# remove any duplicate rows
df = df.drop_duplicates()

# check correlation between numeric columns
print("\nCorrelation matrix:\n", df.select_dtypes(include='number').corr())

# group by a column and find average age in each group
grouped = df.groupby("City")["Age"].mean()
print("\nAverage age by city:\n", grouped)

# count how many times each city appears
print("\nCity counts:\n", df["City"].value_counts())

# sort the data by age
df_sorted = df.sort_values(by="Age")
print("\nSorted by age:\n", df_sorted)

# rename a column
df = df.rename(columns={"Name": "FullName"})
print("\nAfter renaming column:\n", df)

# add a new column: Year of Birth
df["YearOfBirth"] = 2025 - df["Age"]
print("\nAfter adding YearOfBirth column:\n", df)

# drop a column
df = df.drop("YearOfBirth", axis=1)

# filter rows where age is greater than 25
adults = df[df["Age"] > 25]
print("\nRows where age > 25:\n", adults)
```

OUTPUT:

```
Series:
 0    10
1    20
2    30
dtype: int64

Series with custom index:
 a    10
b    20
c    30
```

```
dtype: int64

DataFrame:
      Name   Age     City
0    Alice  25.0    Delhi
1      Bob  30.0   Mumbai
2  Charlie  35.0  Chennai
3    David   NaN   Delhii
4      Eve  25.0    Delhi

First few rows:
      Name   Age     City
0    Alice  25.0    Delhi
1      Bob  30.0   Mumbai
2  Charlie  35.0  Chennai
3    David   NaN   Delhii
4      Eve  25.0    Delhi

Last few rows:
      Name   Age     City
0    Alice  25.0    Delhi
1      Bob  30.0   Mumbai
2  Charlie  35.0  Chennai
3    David   NaN   Delhii
4      Eve  25.0    Delhi

Column names: Index(['Name', 'Age', 'City'], dtype='object')

Data types:
 Name     object
Age     float64
City     object
dtype: object

Summary statistics:
             Age
count   4.000000
mean   28.750000
std     4.787136
min    25.000000
25%    25.000000
50%    27.500000
75%    31.250000
max    35.000000

Missing values:
    Name    Age   City
0  False  False  False
```

```
1  False  False  False
2  False  False  False
3  False   True  False
4  False  False  False

After dropping missing values:
      Name   Age     City
0    Alice  25.0    Delhi
1      Bob  30.0   Mumbai
2  Charlie  35.0  Chennai
4      Eve  25.0    Delhi

After filling missing values:
      Name      Age     City
0    Alice     25.0    Delhi
1      Bob     30.0   Mumbai
2  Charlie     35.0  Chennai
3    David  Unknown   Delhii
4      Eve     25.0    Delhi

Correlation matrix:
      Age
Age  1.0

Average age by city:
 City
Chennai    35.0
Delhi      25.0
Mumbai     30.0
Name: Age, dtype: float64

City counts:
 City
Delhi      3
Mumbai     1
Chennai    1
Name: count, dtype: int64

Sorted by age:
      Name   Age     City
0    Alice  25.0    Delhi
4      Eve  25.0    Delhi
1      Bob  30.0   Mumbai
2  Charlie  35.0  Chennai
3    David   NaN    Delhi

After renaming column:
   FullName   Age     City
```

```
0    Alice  25.0    Delhi
1      Bob  30.0   Mumbai
2  Charlie  35.0  Chennai
3    David   NaN    Delhi
4      Eve  25.0    Delhi

After adding YearOfBirth column:
   FullName   Age     City  YearOfBirth
0    Alice  25.0    Delhi       2000.0
1      Bob  30.0   Mumbai       1995.0
2  Charlie  35.0  Chennai       1990.0
3    David   NaN    Delhi          NaN
4      Eve  25.0    Delhi       2000.0

Rows where age > 25:
   FullName   Age     City
1      Bob  30.0   Mumbai
2  Charlie  35.0  Chennai
```

SCIPY

```
import numpy as np
from scipy import constants, stats, optimize, interpolate, linalg, signal, special, spatial

# constants
print("Speed of light (m/s):", constants.c)
print("Avogadro's number:", constants.N_A)
print()

# stats - probability distributions & tests
x = np.linspace(-3, 3, 100)
normal = stats.norm(loc=0, scale=1)
print("normal pdf(0):", normal.pdf(0))
print("normal cdf(0):", normal.cdf(0))
print("mean:", normal.mean(), "std:", normal.std())

# t-test
a = np.random.randn(20) + 0.5
b = np.random.randn(20)
t_stat, p_val = stats.ttest_ind(a, b)
print()
print("t-statistic:", t_stat)
print("p-value:", p_val)

# chi-squared test
obs = np.array([10, 20, 30])
exp = np.array([15, 15, 30])
chi2, p = stats.chisquare(obs, f_exp=exp)
```

```
print()
print("chi2 value:", chi2)
print("p-value:", p)
print()

# optimize
root = optimize.root_scalar(lambda t: t**2 - 16, bracket=[0, 5]).root
print("Root of x^2 - 16:", root)

res = optimize.minimize(lambda t: (t-2)**2, x0=0)
print("Minimize (x-2)^2 result:", res.x)

def func(x, a, b): return a * np.exp(-b * x)
xp = np.linspace(0, 4, 50)
yp = func(xp, 2.5, 1.3) + 0.2 * np.random.normal(size=xp.size)
params, cov = optimize.curve_fit(func, xp, yp)
print("Fitted parameters (a, b):", params)
print()

# interpolate
f = interpolate.interp1d(xp, yp, kind='cubic')
print("Interpolated value at x=1.5:", f(1.5))
print()

# linalg
M = np.array([[3, 1], [1, 2]])
w, v = linalg.eig(M)
print("Eigenvalues:", w)
print("Inverse of matrix M:\n", linalg.inv(M))
print()

# signal
sig = np.sin(xp)
kernel = np.ones(5) / 5
smoothed = signal.convolve(sig, kernel, mode='same')
print("Smoothed signal (first 5 values):", smoothed[:5])
print()

# special
print("Bessel function J0(1):", special.j0(1))
print("Gamma function of 5:", special.gamma(5))
print()

# spatial
pts = np.random.rand(10, 2)
kdt = spatial.KDTree(pts)
dist, index = kdt.query(pts[0], k=2)
print("Distance to nearest neighbor:", dist[1])
```

```
print("Index of nearest neighbor:", index[1])
print()
```

OUTPUT:

```
peed of light (m/s): 299792458.0
Avogadro's number: 6.02214076e+23

normal pdf(0): 0.3989422804014327
normal cdf(0): 0.5
mean: 0.0 std: 1.0

t-statistic: 1.8178532689200808
p-value: 0.07697707619624078

chi2 value: 3.3333333333333335
p-value: 0.1888756028375618

Root of x^2 - 16: 4.0
Minimize (x-2)^2 result: [1.99999998]
Fitted parameters (a, b): [2.44637921 1.24173072]

Interpolated value at x=1.5: 0.22041128436603624

Eigenvalues: [3.61803399+0.j 1.38196601+0.j]
Inverse of matrix M:
 [[ 0.4 -0.2]
 [-0.2  0.6]]

Smoothed signal (first 5 values): [0.04881659 0.09730806 0.16145984 0.24084467 0.31862543]

Bessel function J0(1): 0.7651976865579665
Gamma function of 5: 24.0

Distance to nearest neighbor: 0.3540393223995117
Index of nearest neighbor: 6
```

SCIKIT-LEARN

```
# importing all required libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, Normalizer
from sklearn.linear_model import LinearRegression
```

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, mean_squared_error, r2_score

# load iris dataset for classification
iris = load_iris()
X = iris.data
y = iris.target

# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Random Forest Classification
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print("Random Forest Classification Accuracy:", accuracy_score(y_test, y_pred))
print()

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print()

print("Classification Report:\n", classification_report(y_test, y_pred))
print()

# Standardization (Z-score normalization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("First 3 rows after standard scaling:\n", X_scaled[:3])
print()

# Normalization (scaling between 0 and 1)
normalizer = MinMaxScaler()
X_normalized = normalizer.fit_transform(X)
print("First 3 rows after normalization:\n", X_normalized[:3])
print()

# Encode target labels (though iris target is already encoded)
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)
print("Encoded class labels:", y_encoded[:10])
print()

# Cross-validation
cv_scores = cross_val_score(clf, X, y, cv=5)
print("Cross-validation scores:", cv_scores)
print()
```

```
print("Mean cross-validation accuracy:", np.mean(cv_scores))
print()

# GridSearchCV for best hyperparameter
params = {'n_estimators': [50, 100]}
grid = GridSearchCV(RandomForestClassifier(), param_grid=params, cv=3)
grid.fit(X_train, y_train)
print("Best parameters from GridSearch:", grid.best_params_)
print()

# KMeans clustering (unsupervised)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X[:, :2])  # use first 2 features for simplicity
print("KMeans cluster centers:\n", kmeans.cluster_centers_)
print()

print("KMeans Silhouette Score:", silhouette_score(X[:, :2], kmeans.labels_))
print()

# PCA dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
print("PCA reduced shape:", X_pca.shape)
print()
```

OUTPUT:

```
Random Forest Classification Accuracy: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       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


First 3 rows after standard scaling:
```

```
 [[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
 [-1.14301691 -0.13197948 -1.34022653 -1.3154443 ]
 [-1.38535265  0.32841405 -1.39706395 -1.3154443 ]]

First 3 rows after normalization:
 [[0.22222222 0.625      0.06779661 0.04166667]
 [0.16666667 0.41666667 0.06779661 0.04166667]
 [0.11111111 0.5        0.05084746 0.04166667]]

Encoded class labels: [0 0 0 0 0 0 0 0 0 0]

Cross-validation scores: [0.96666667 0.96666667 0.93333333 0.96666667 1.        ]

Mean cross-validation accuracy: 0.9666666666666668

Best parameters from GridSearch: {'n_estimators': 50}

KMeans cluster centers:
 [[6.81276596 3.07446809]
 [5.77358491 2.69245283]
 [5.006      3.428     ]]

KMeans Silhouette Score: 0.4450525692083638

PCA reduced shape: (150, 2)
```

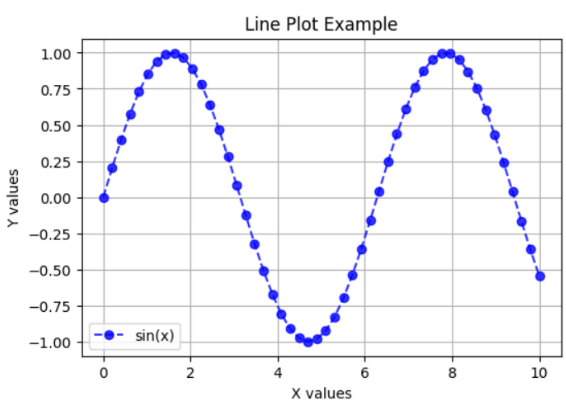
MATPLOTLIB

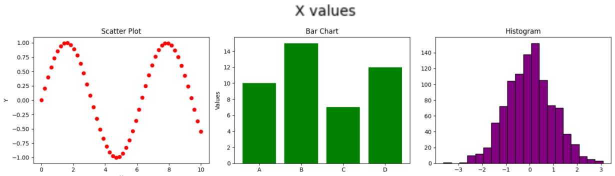
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# Sample data
x = np.linspace(0, 10, 50)
y = np.sin(x)
categories = ['A', 'B', 'C', 'D']
values = [10, 15, 7, 12]
data = np.random.randn(1000)

# Line plot with customization
plt.figure(figsize=(6, 4))
plt.plot(x, y, color='blue', linestyle='--', marker='o', label='sin(x)', alpha=0.8)
plt.title('Line Plot Example')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.grid(True)
plt.legend()
```

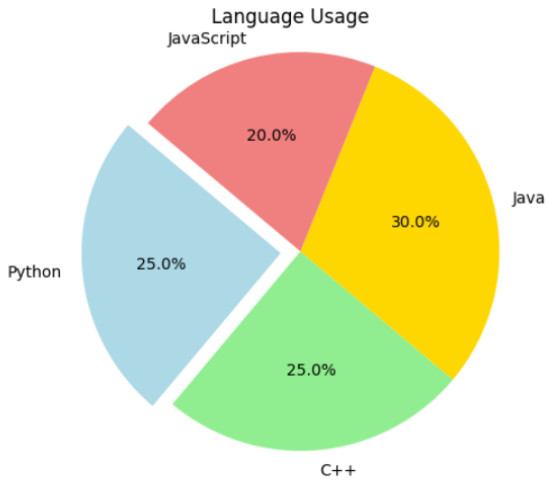
```
plt.savefig("line_plot.png")  # Save the plot
plt.show()

# Scatter, Bar, Histogram as subplots
fig, axs = plt.subplots(1, 3, figsize=(15, 4))

# Scatter
axs[0].scatter(x, y, color='red')
axs[0].set_title('Scatter Plot')
axs[0].set_xlabel('X')
axs[0].set_ylabel('Y')

# Bar chart
axs[1].bar(categories, values, color='green')
axs[1].set_title('Bar Chart')
axs[1].set_ylabel('Values')

# Histogram
axs[2].hist(data, bins=20, color='purple', edgecolor='black')
axs[2].set_title('Histogram')

plt.tight_layout()
plt.savefig("subplot_visuals.png")
plt.show()

# Pie chart
sizes = [25, 25, 30, 20]
labels = ['Python', 'C++', 'Java', 'JavaScript']
colors = ['lightblue', 'lightgreen', 'gold', 'lightcoral']
explode = [0.1, 0, 0, 0]

plt.figure(figsize=(5, 5))
plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%', startangle=14
plt.title("Language Usage")
plt.axis('equal')
plt.savefig("pie_chart.png")
plt.show()

# Box plot
data_box = np.random.normal(100, 10, 200)
plt.boxplot(data_box)
plt.title("Box Plot")
plt.ylabel("Values")
plt.savefig("box_plot.png")
plt.show()

# Heatmap using imshow()
matrix = np.random.rand(5, 5)
```

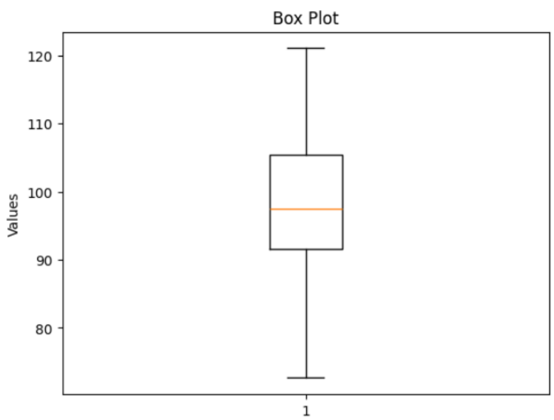
```
plt.imshow(matrix, cmap='viridis', interpolation='nearest')
plt.colorbar()
plt.title("Heatmap with imshow()")
plt.savefig("heatmap.png")
plt.show()

# Integration with Pandas
df = pd.DataFrame({
    'x': np.arange(10),
    'y': np.random.rand(10)
})
df.plot(x='x', y='y', kind='line', title="Pandas Line Plot")
plt.savefig("pandas_plot.png")
plt.show()

# Seaborn integration
iris = sns.load_dataset("iris")
sns.scatterplot(data=iris, x='sepal_length', y='sepal_width', hue='species')
plt.title("Seaborn Scatterplot - Iris Dataset")
plt.savefig("seaborn_scatter.png")
plt.show()
```

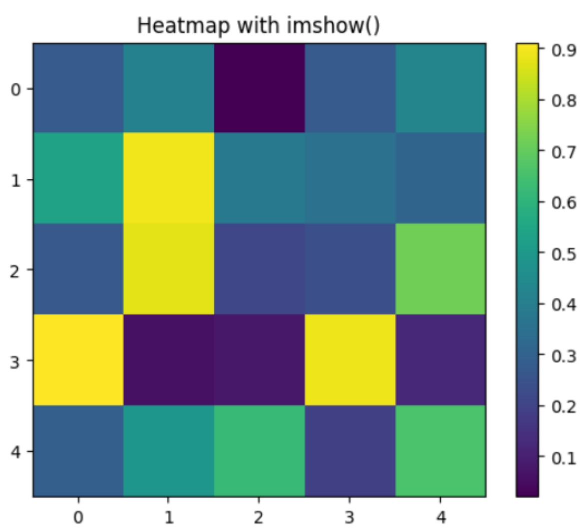
OUTPUT:

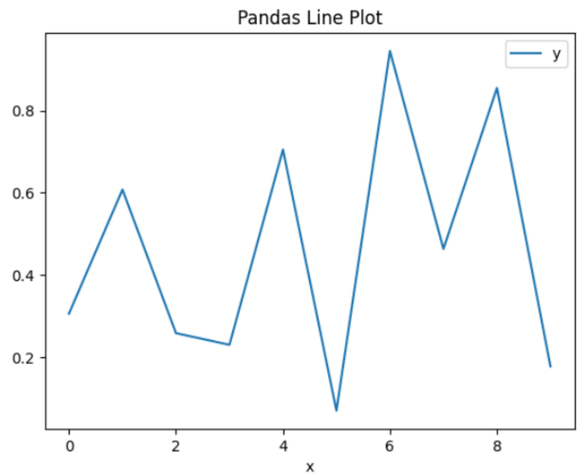


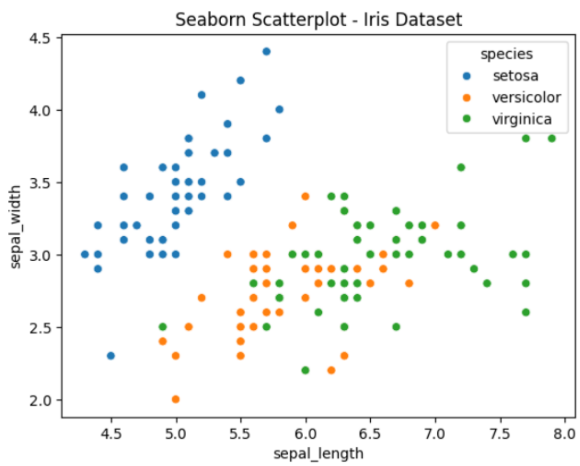



Language Usage
JavaScript
20.0%
Java
30.0%
Python
25.0%
25.0%
C++


Box Plot
120
110
100
90
80
Values
1


Heatmap with imshow()
0
1
2
3
4
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1





2.Explore public repositories such as the UCI Machine Learning Repository (UCI Repository) and Kaggle Datasets. Download the following datasets and identify the appropriate machine learning model to be used (e.g., Supervised, Unsupervised, Semi-supervised, Regression, Clas sification) [CO1, K3]. i.) Loan amount prediction ii.) Handwritten character recognition iii.) Classification of Email spam and MNIST data iv.) Predicting Diabetes v.) Iris Dataset

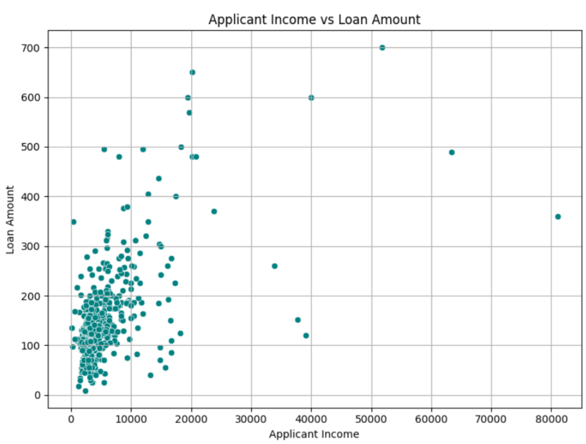
CODE:

```
1.LOAN AMOUNT PREDICTION
from google.colab import drive
drive.mount('/content/drive')

# 1. Import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab import drive

# 2. Mount Google Drive and load dataset
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/loan_prediction.csv')
```

```
# 3. Display correlation between LoanAmount and incomes
print("Correlation Matrix:")
print(df[['LoanAmount', 'ApplicantIncome', 'CoapplicantIncome']].corr())

# 4. Scatterplot of ApplicantIncome vs LoanAmount
plt.figure(figsize=(8,6))
sns.scatterplot(x='ApplicantIncome', y='LoanAmount', data=df, color='teal')
plt.title("Applicant Income vs Loan Amount")
plt.xlabel("Applicant Income")
plt.ylabel("Loan Amount")
plt.grid(True)
plt.tight_layout()
plt.show()
```

OUTPUT:

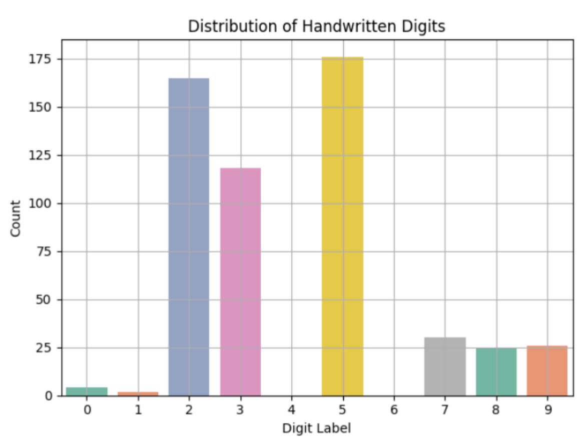
```
Correlation Matrix:
                   LoanAmount  ApplicantIncome  CoapplicantIncome
LoanAmount           1.000000         0.570909           0.188619
ApplicantIncome      0.570909         1.000000          -0.116605
CoapplicantIncome    0.188619        -0.116605           1.000000
```

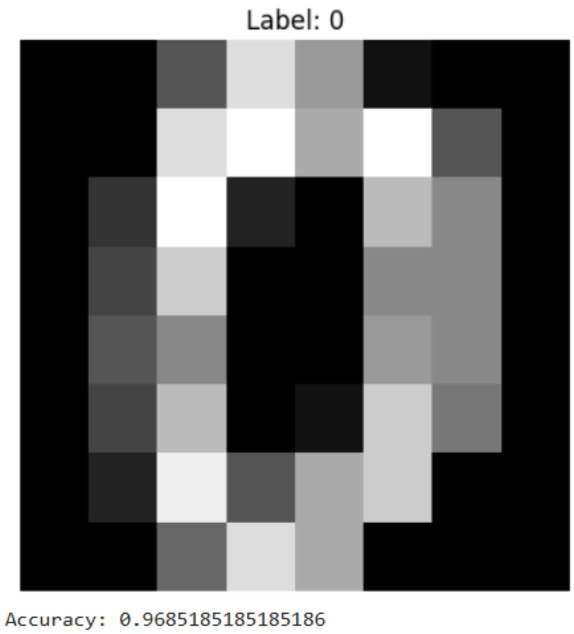



```
2.HANDWRITTEN CHARACTER RECOGNITION MODEL
# 1. Import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# 2. Load dataset
digits = load_digits()
```

```
X = digits.data
y = digits.target

# 3. Convert to DataFrame for visualization
df = pd.DataFrame(X)
df['label'] = y

# 4. Show class distribution
sns.countplot(x='label', data=df, palette='Set2')
plt.title("Distribution of Handwritten Digits")
plt.xlabel("Digit Label")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()

# 5. Display a sample digit image
plt.imshow(digits.images[0], cmap='gray')
plt.title(f'Label: {digits.target[0]}')
plt.axis('off')
plt.show()

# 6. Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 7. Train the model
model = LogisticRegression(max_iter=3000)
model.fit(X_train, y_train)

# 8. Evaluate the model
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:





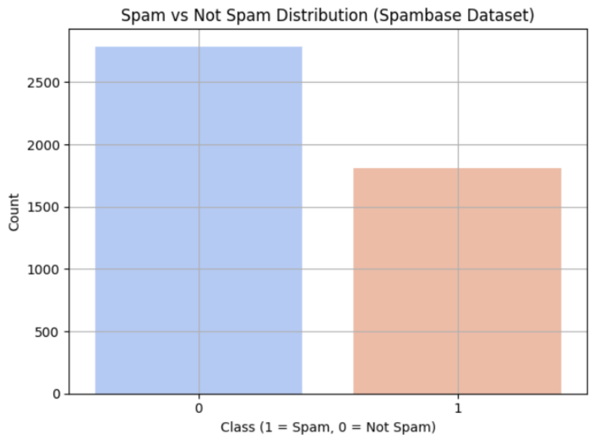

```
Accuracy: 0.9685185185185186

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       0.96      0.94      0.95        50
           2       0.96      1.00      0.98        47
           3       0.98      0.96      0.97        54
           4       1.00      0.97      0.98        60
           5       0.94      0.95      0.95        66
           6       0.96      0.98      0.97        53
           7       1.00      0.96      0.98        55
           8       0.91      0.98      0.94        43
           9       0.97      0.95      0.96        59

    accuracy                           0.97       540
   macro avg       0.97      0.97      0.97       540
weighted avg       0.97      0.97      0.97       540
```

3.CLASSIFICATION OF EMAIL SPAM

```
# 1. Install ucimlrepo
!pip install ucimlrepo

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

```
from ucimlrepo import fetch_ucirepo
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# 3. Load dataset from UCI
spambase = fetch_ucirepo(id=94)

# 4. Extract features and target
X = spambase.data.features
y = spambase.data.targets

# 5. Combine features and target for visualization
df = pd.concat([X, y], axis=1)
target_col = y.columns[0]  # Usually 'class' or 'target'

# 6. Visualize class distribution
sns.countplot(x=target_col, data=df, palette="coolwarm")
plt.title("Spam vs Not Spam Distribution (Spambase Dataset)")
plt.xlabel(f"{target_col} (1 = Spam, 0 = Not Spam)")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()

# 7. Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y[target_col], test_size=0.2, random_sta

# 8. Train Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# 9. Evaluate model
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:




```
Accuracy: 0.8067318132464713

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.72      0.81       538
           1       0.70      0.93      0.80       383

    accuracy                           0.81       921
   macro avg       0.82      0.82      0.81       921
weighted avg       0.84      0.81      0.81       921
```

4.PREDICTING DIABETES

```
# 1. Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, roc_curve

# 2. Load diabetes dataset from sklearn
data = load_diabetes(as_frame=True)
df = data.frame

# 3. EDA - Show info and description
print(df.info())
print(df.describe())

# 4. Convert continuous target into binary Outcome (0 = low, 1 = high)
df['Outcome'] = (df['target'] > df['target'].median()).astype(int)
```

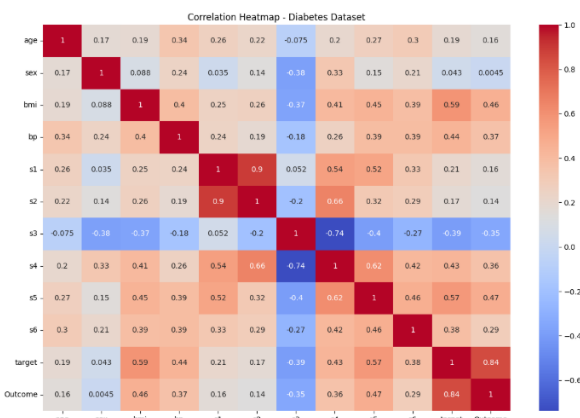
```
# 5. Correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap - Diabetes Dataset")
plt.tight_layout()
plt.show()

# 6. Class distribution
sns.countplot(x='Outcome', data=df, palette='Set2')
plt.title("Diabetes Outcome Distribution (0 = Low, 1 = High)")
plt.xlabel("Outcome")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()

# 7. Preprocessing
X = df.drop(columns=['target', 'Outcome'])
y = df['Outcome']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 8. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=

# 9. Train Random Forest model
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

# 10. Evaluation
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# 11. Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=["Low", "High"], yticklabels=["Low", "High"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()

# 12. Feature Importance
feature_names = X.columns
```

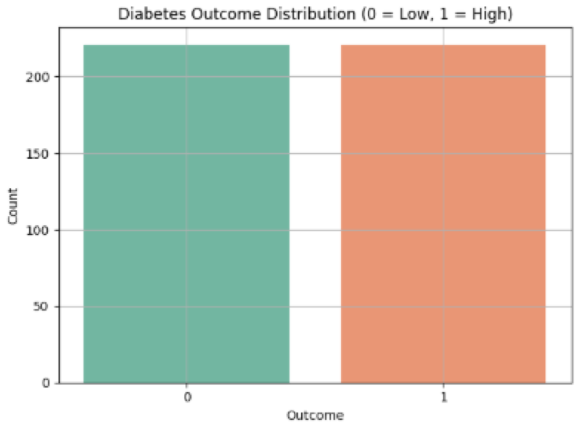
```
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices], y=feature_names[indices])
plt.title("Feature Importances (Random Forest)")
plt.xlabel("Importance")
plt.ylabel("Features")
plt.tight_layout()
plt.show()

# 13. ROC Curve
y_proba = clf.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([-0.01, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()
```

OUTPUT:

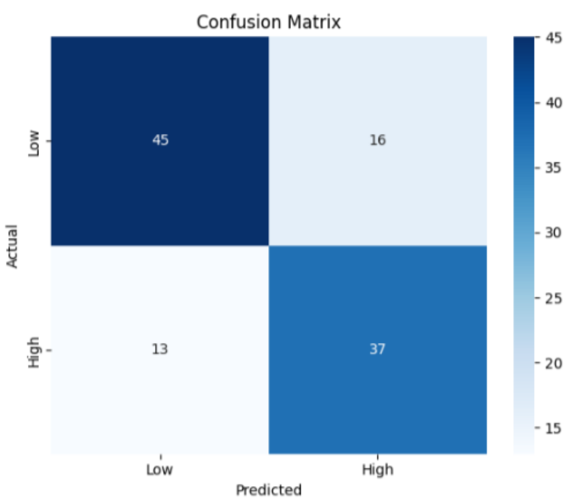
```
angeIndex: 442 entries, 0 to 441
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
 10  target  442 non-null    float64
dtypes: float64(11)
memory usage: 38.1 KB
None
```

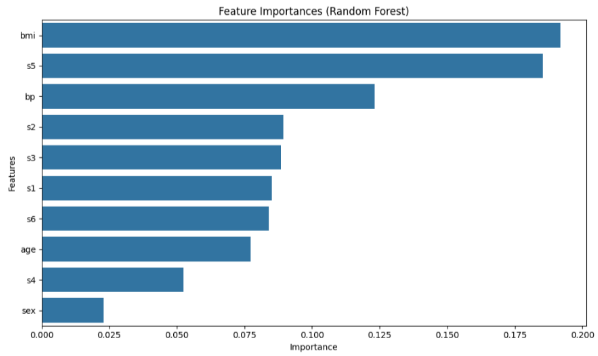
```
                age           sex           bmi            bp            s1  \
count  4.420000e+02  4.420000e+02  4.420000e+02  4.420000e+02  4.420000e+02
mean  -2.511817e-19  1.230790e-17 -2.245564e-16 -4.797570e-17 -1.381499e-17
std    4.761905e-02  4.761905e-02  4.761905e-02  4.761905e-02  4.761905e-02
min   -1.072256e-01 -4.464164e-02 -9.027530e-02 -1.123988e-01 -1.267807e-01
25%   -3.729927e-02 -4.464164e-02 -3.422907e-02 -3.665608e-02 -3.424784e-02
50%    5.383060e-03 -4.464164e-02 -7.283766e-03 -5.670422e-03 -4.320866e-03
75%    3.807591e-02  5.068012e-02  3.124802e-02  3.564379e-02  2.835801e-02
max    1.107267e-01  5.068012e-02  1.705552e-01  1.320436e-01  1.539137e-01

                 s2            s3            s4            s5            s6  \
count  4.420000e+02  4.420000e+02  4.420000e+02  4.420000e+02  4.420000e+02
mean   3.918434e-17 -5.777179e-18 -9.042540e-18  9.293722e-17  1.130318e-17
std    4.761905e-02  4.761905e-02  4.761905e-02  4.761905e-02  4.761905e-02
min   -1.156131e-01 -1.023071e-01 -7.639450e-02 -1.260971e-01 -1.377672e-01
25%   -3.035840e-02 -3.511716e-02 -3.949338e-02 -3.324559e-02 -3.317903e-02
50%   -3.819065e-03 -6.584468e-03 -2.592262e-03 -1.947171e-03 -1.077698e-03
75%    2.984439e-02  2.931150e-02  3.430886e-02  3.243232e-02  2.791705e-02
max    1.987880e-01  1.811791e-01  1.852344e-01  1.335973e-01  1.356118e-01

           target
count  442.000000
mean   152.133484
std     77.093005
min     25.000000
25%     87.000000
50%    140.500000
75%    211.500000
max    346.000000
```

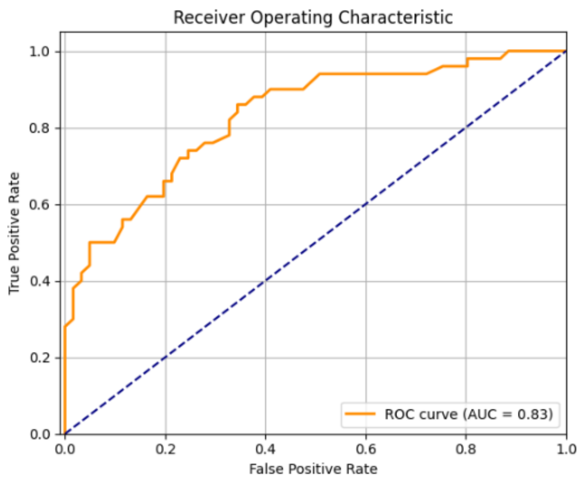





```
Accuracy: 0.7387387387387387

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.74      0.76        61
           1       0.70      0.74      0.72        50

    accuracy                           0.74       111
   macro avg       0.74      0.74      0.74       111
weighted avg       0.74      0.74      0.74       111
```

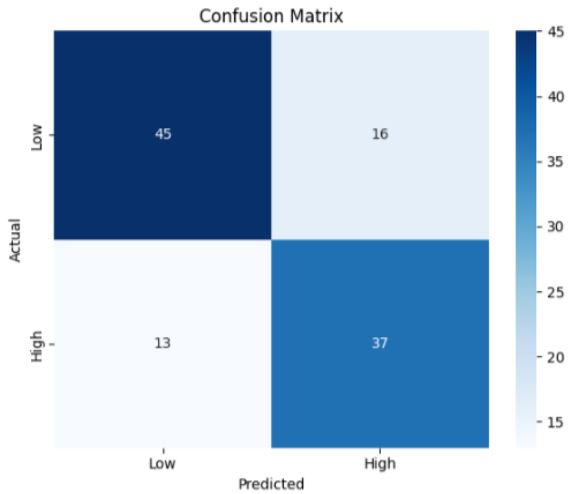







```
5.IRIS DATASET MODEL
import seaborn as sns
from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt

iris = load_iris(as_frame=True)
df = iris.frame

sns.pairplot(df, hue='target')
plt.suptitle("Iris Dataset - Pairwise Plot by Species", y=1.02)
plt.show()
```

OUTPUT:



Inference Table:

Dataset	**ML Task**	**Model / Technique**
Loan Amount Prediction	Regression (Supervised)	Linear Regression
Handwritten Digit Recognition	Classification (Supervised)	Logistic Regression
Spam Detection	Classification (Supervised)	Logistic Regression with Chi-square Feature Selection
Diabetes Prediction	Classification (Supervised)	Linear Regression with SelectKBest
Iris Dataset	Classification	Random Forest, PCA, KMeans (for clustering)

Learning Outcomes:

- Understood the usage of key ML libraries in Python.
- Learned how to clean, preprocess, and visualize datasets.
- Explored different ML models and their evaluation techniques.
- Identified suitable algorithms for various ML problems.