Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An autonomous Institution affiliated to Anna University)

Degree & Branch	M.Tech Computer Science & Engineering Integrated (5 Years)	**Semester** V
Subject Code & Name	ICS1512 & Machine Learning Algorithms Laboratory	
Academic year	2025–2026 (Odd)	**Batch:** 2023–2028

Name: I.S.Rajesh **Register No.: 3122237001042**

Experiment #1: Working with Python packages – Numpy, Scipy, Scikit-learn, Matplotlib

Aim:
To explore key operations and methods of Python libraries such as Numpy, Pandas, Scipy, Scikit-learn, and Matplotlib and apply them to perform complete machine learning workflows on real-world datasets.

Libraries used:
`numpy, pandas, scipy, sklearn, matplotlib`

Python Codes and Outputs for Exploring Libraries

Listing 1: Import libraries

```
# Import all necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats, integrate
from sklearn.datasets import load_iris
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
```

Listing 2: Numpy

```
# 1. NumPy: Array Manipulation

arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Original␣array:\n", arr)
print("Shape:", arr.shape)
print("Transpose:\n", arr.T)
print("Element-wise␣square:\n", np.square(arr))
print("Mean␣of␣elements:", np.mean(arr))
```

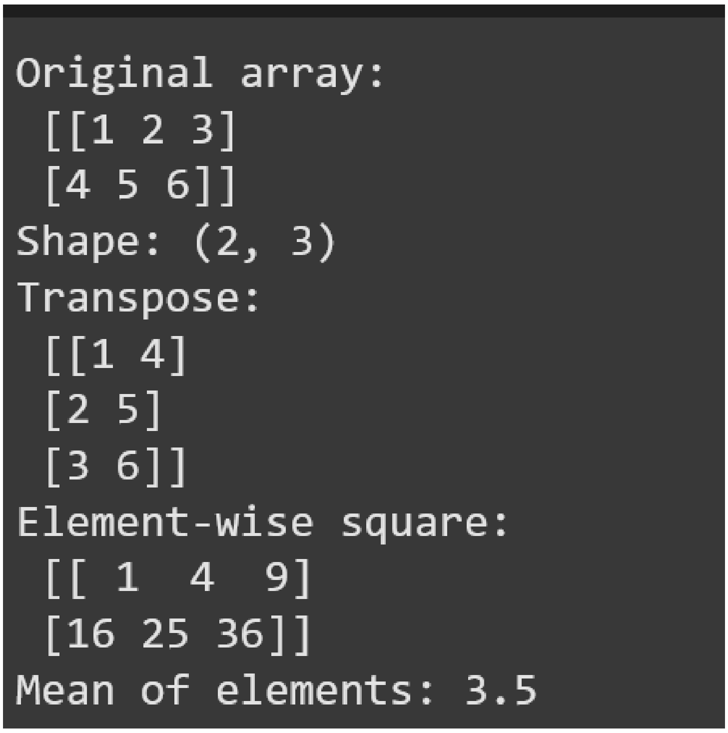

Figure 1: Numpy Output

Listing 3: Pandas

```
# 2. Pandas: Data Handling & Preprocessing

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 32, 30, np.nan],
        'Gender': ['F', 'M', 'M', 'M']}
df = pd.DataFrame(data)
print("Original␣DataFrame:\n", df)

# Handling missing values
df['Age'].fillna(df['Age'].mean(), inplace=True)
print("\nAfter␣filling␣missing␣Age:\n", df)

# Encoding categorical variables
df['Gender'] = df['Gender'].map({'F': 0, 'M': 1})
print("\nAfter␣encoding␣Gender:\n", df)
```

```
Original DataFrame:
      Name   Age Gender
0    Alice  25.0      F
1      Bob  32.0      M
2  Charlie  30.0      M
3    David   NaN      M

After filling missing Age:
      Name   Age Gender
0    Alice  25.0      F
1      Bob  32.0      M
2  Charlie  30.0      M
3    David  29.0      M

After encoding Gender:
      Name   Age  Gender
0    Alice  25.0       0
1      Bob  32.0       1
2  Charlie  30.0       1
3    David  29.0       1
```

Figure 2: Pandas Output

Listing 4: Scipy

```python
# 3. SciPy: Mathematical Functions

# Generate normal distribution and calculate statistics
sample = np.random.normal(loc=0, scale=1, size=1000)
mean = np.mean(sample)
std_dev = np.std(sample)
kurtosis = stats.kurtosis(sample)
skew = stats.skew(sample)

print(f"Mean: {mean:.2f}, Std Dev: {std_dev:.2f}")
print(f"Kurtosis: {kurtosis:.2f}, Skewness: {skew:.2f}")
```

```
Mean: -0.02, Std Dev: 1.04
Kurtosis: -0.08, Skewness: 0.01
```

Figure 3: Scipy Output

Listing 5: Scikitlearn

```python
# 4. Scikit-learn: Basic ML Workflow

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, stratify=y, random_state=42)

# Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Model training
model = LogisticRegression(max_iter=200)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9333333333333333
Confusion Matrix:
 [[10  0  0]
 [ 0  9  1]
 [ 0  1  9]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.90      0.90      0.90        10
           2       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

Figure 4: Scikitlearn Output

Listing 6: Scikitlearn

```python
# 5. Matplotlib & Seaborn: Visualizations

# Histogram
plt.figure(figsize=(6, 4))
plt.hist(sample, bins=30, color='skyblue', edgecolor='black')
```

```
plt.title("Histogram␣of␣Random␣Normal␣Distribution")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

# Scatter plot (Iris dataset)
df_iris = pd.DataFrame(X, columns=iris.feature_names)
df_iris['species'] = y
plt.figure(figsize=(6, 5))
sns.scatterplot(data=df_iris, x='sepal␣length␣(cm)', y='petal␣length␣(
    ↪ cm)', hue='species', palette='viridis')
plt.title("Iris␣-␣Sepal␣vs␣Petal␣Length")
plt.show()

# Heatmap of correlation
plt.figure(figsize=(8, 6))
sns.heatmap(df_iris.drop('species', axis=1).corr(), annot=True, cmap='
    ↪ coolwarm')
plt.title("Feature␣Correlation␣-␣Iris")
plt.show()
```

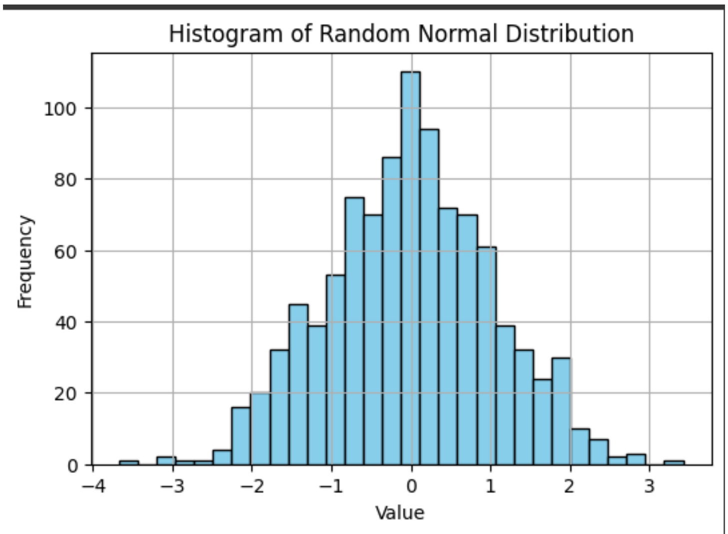

Figure 5: Matplotlib Output 1

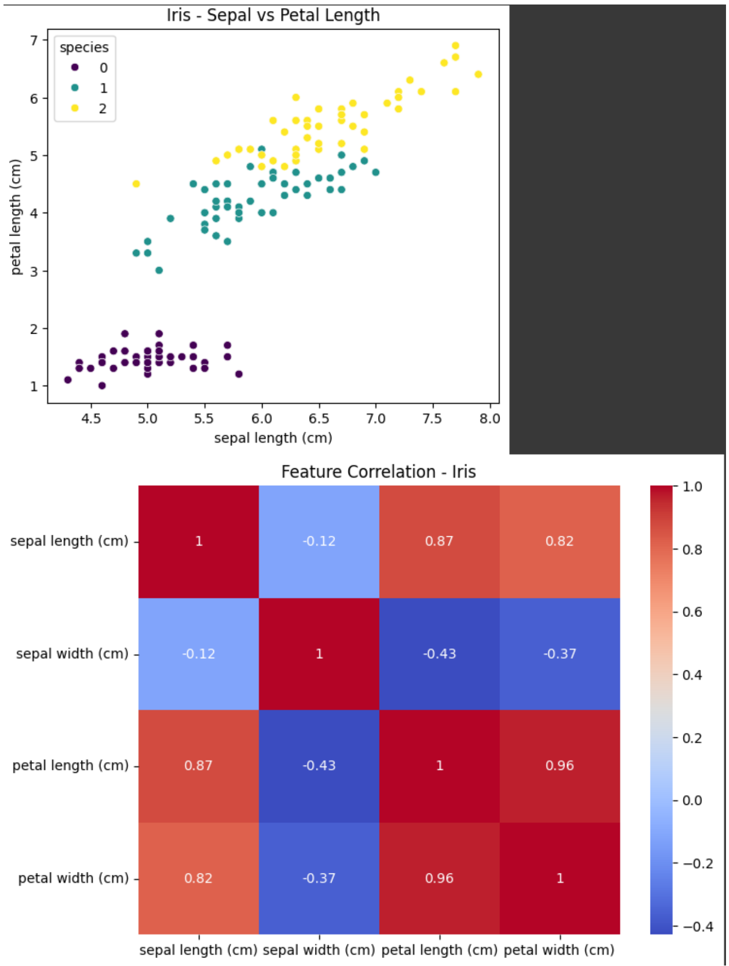

Figure 6: Matplotlib Output 2

Python Codes for Working on Real-World Datasets

Listing 7: Import libraries

```
# Machine Learning Workflows for Various Datasets

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_selection import SelectKBest, f_classif, chi2, f_regression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, mean_absolute_error, mean_squared_error, r2_score, ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.datasets import load_digits, load_iris
from sklearn.feature_extraction.text import TfidfVectorizer
```

Listing 8: Loan Prediction

```
# i) Loan Amount Prediction
print("\n---␣Loan␣Prediction␣---")
df = pd.read_csv("train_u6lujuX_CVtuZ9i.xls")

# Drop Loan_ID
df.drop('Loan_ID', axis=1, inplace=True)

# Drop rows where LoanAmount (target) is missing
df = df.dropna(subset=['LoanAmount'])

# Identify Numerical and Categorical Columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
    ↪ .tolist()
numerical_cols.remove('LoanAmount')  # Exclude target

categorical_cols = df.select_dtypes(include='object').columns.tolist()

# Handle Missing Values
for col in numerical_cols:
    df[col] = df[col].fillna(df[col].median())

for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])

# Target-Guided Ordinal Encoding for Categorical Columns
for col in categorical_cols:
    ordering = df.groupby(col)["LoanAmount"].mean().sort_values().index
    ordinal_map = {key: idx for idx, key in enumerate(ordering)}
    df[col] = df[col].map(ordinal_map)

# Limited Feature Engineering (since dataset lacks property price etc.)
df['IncomeToLoanRatio'] = (df['ApplicantIncome'] + df['
    ↪ CoapplicantIncome']) / (df['LoanAmount'] + 1)
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']

# Define Features and Target
X = df.drop('LoanAmount', axis=1)
y = df['LoanAmount']

# Standard Scaling (outliers present)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-Test Split (80-20)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
    ↪ test_size=0.2, random_state=42)

# Train Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_test_pred = model.predict(X_test)

# Performance Metrics
mse = mean_squared_error(y_test, y_test_pred)
rmse = np.sqrt(mse)
```

```
mae = mean_absolute_error(y_test, y_test_pred)
r2 = r2_score(y_test, y_test_pred)
n, p = X_test.shape
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("Mean Absolute Error:", mae)
print("R2 Score:", r2)
print("Adjusted R2 Score:", adjusted_r2)

# Actual vs Predicted Plot
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_test_pred, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.xlabel("Actual Loan Amount")
plt.ylabel("Predicted Loan Amount")
plt.title("Actual vs Predicted Loan Amount")
plt.show()

# Residual Distribution Plot
residuals = y_test - y_test_pred
plt.figure(figsize=(6, 4))
sns.histplot(residuals, kde=True, color='orange')
plt.title("Residuals Distribution")
plt.show()
```

Listing 9: Handwritten Digit Recognition

```
# ii) Handwritten Digit Recognition
print("\n--- Handwritten Digit Recognition ---")
digits = load_digits()
X, y = digits.data, digits.target
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    ↪ =0.3, stratify=y)
clf = RandomForestClassifier().fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Digit Image Visualization
plt.gray()
plt.matshow(digits.images[0])
plt.title(f"Digit Label: {digits.target[0]}")
plt.show()

ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
plt.title("Confusion Matrix - Digits")
plt.show()
```

Listing 10: Spam Email Classification

```
# iii) Spam Email Classification
print("\n--- Email Spam Classification ---")
spam_df = pd.read_csv("spam.csv", encoding="latin-1")[["v1", "v2"]]
spam_df.columns = ["label", "message"]
spam_df['label'] = spam_df['label'].map({"ham": 0, "spam": 1})
```

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(spam_df['message'])
y = spam_df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    ↪ =0.2, stratify=y)
model = MultinomialNB().fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Spam Distribution
spam_df['label'].value_counts().plot(kind='bar')
plt.title("Ham vs Spam Distribution")
plt.xticks([0,1], ['Ham', 'Spam'], rotation=0)
plt.ylabel("Count")
plt.show()

ConfusionMatrixDisplay.from_estimator(model, X_test, y_test)
plt.title("Confusion Matrix - Spam Detection")
plt.show()
```

Listing 11: Diabetes Prediction

```
# iv) Diabetes Prediction
print("\n--- Diabetes Prediction ---")
diabetes = pd.read_csv("diabetes.csv")
cols_with_zero = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin
    ↪ ', 'BMI']
diabetes[cols_with_zero] = diabetes[cols_with_zero].replace(0, np.nan)
diabetes.fillna(diabetes.median(numeric_only=True), inplace=True)

X = diabetes.drop("Outcome", axis=1)
y = diabetes["Outcome"]

# Use StandardScaler due to outliers
X = StandardScaler().fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    ↪ =0.2, stratify=y, random_state=42)
model = LogisticRegression(max_iter=200).fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

plt.hist(diabetes['Glucose'].dropna(), bins=20)
plt.title("Glucose Distribution")
plt.xlabel("Glucose")
plt.ylabel("Frequency")
plt.show()

ConfusionMatrixDisplay.from_estimator(model, X_test, y_test)
plt.title("Confusion Matrix - Diabetes")
plt.show()
```

Listing 12: Iris Classification

```
# v) Iris Classification
print("\n--- Iris Dataset Classification ---")
iris = load_iris()
X, y = iris.data, iris.target

# Apply StandardScaler due to small numerical differences
X = StandardScaler().fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
model = SVC().fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
sns.scatterplot(data=df, x='sepal length (cm)', y='petal length (cm)', hue='species')
plt.title("Iris Feature Scatter")
plt.show()

ConfusionMatrixDisplay.from_estimator(model, X_test, y_test)
plt.title("Confusion Matrix - Iris")
plt.show()
```

Output Screenshots

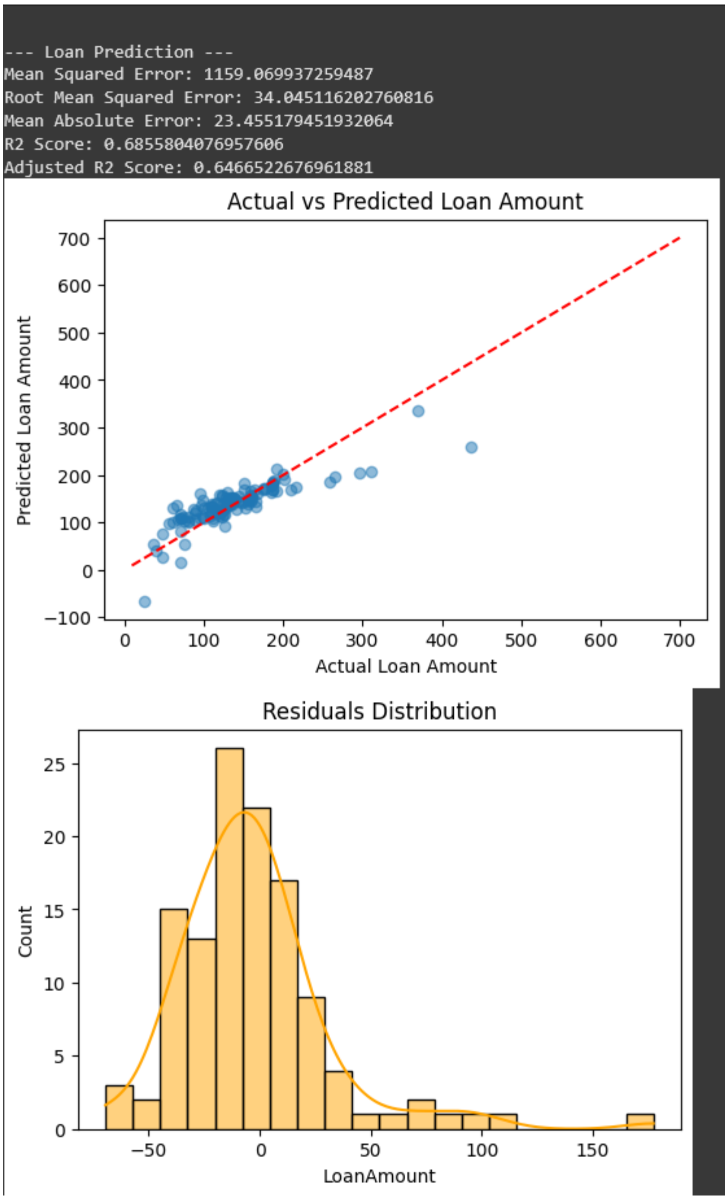

Figure 7: Loan Prediction Output and Performance

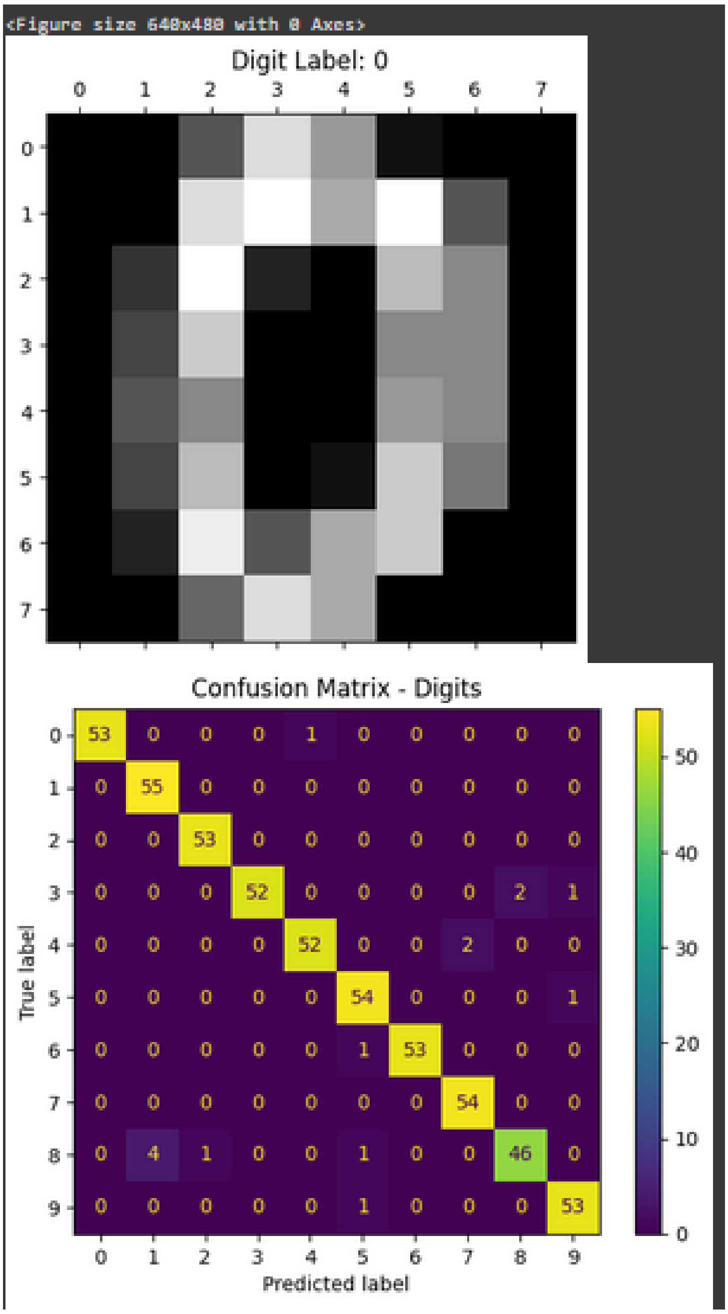

Figure 8: Handwritten Digit Classification Output

```
--- Handwritten Digit Recognition ---
Accuracy: 0.9722222222222222
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        54
           1       0.93      1.00      0.96        55
           2       0.98      1.00      0.99        53
           3       1.00      0.95      0.97        55
           4       0.98      0.96      0.97        54
           5       0.95      0.98      0.96        55
           6       1.00      0.98      0.99        54
           7       0.96      1.00      0.98        54
           8       0.96      0.88      0.92        52
           9       0.96      0.98      0.97        54

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

Figure 9: Handwritten Digit Classification Performance

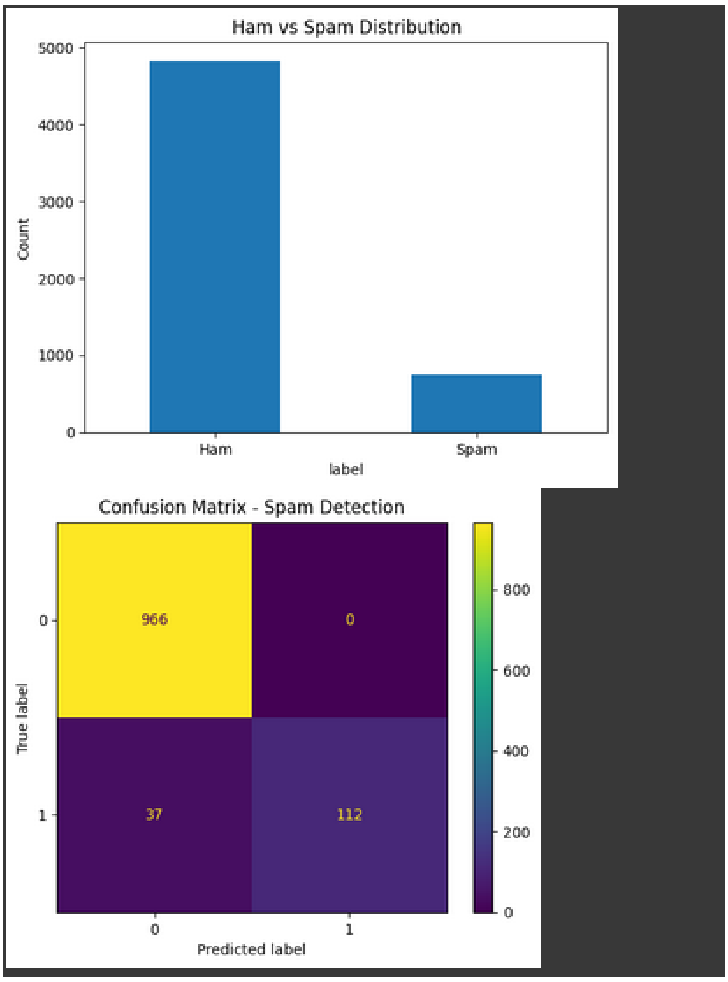

Figure 10: Spam Classification Output

```
--- Email Spam Classification ---
Accuracy: 0.9668161434977578
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       966
           1       1.00      0.75      0.86       149

    accuracy                           0.97      1115
   macro avg       0.98      0.88      0.92      1115
weighted avg       0.97      0.97      0.96      1115
```

Figure 11: Spam Classification Performance

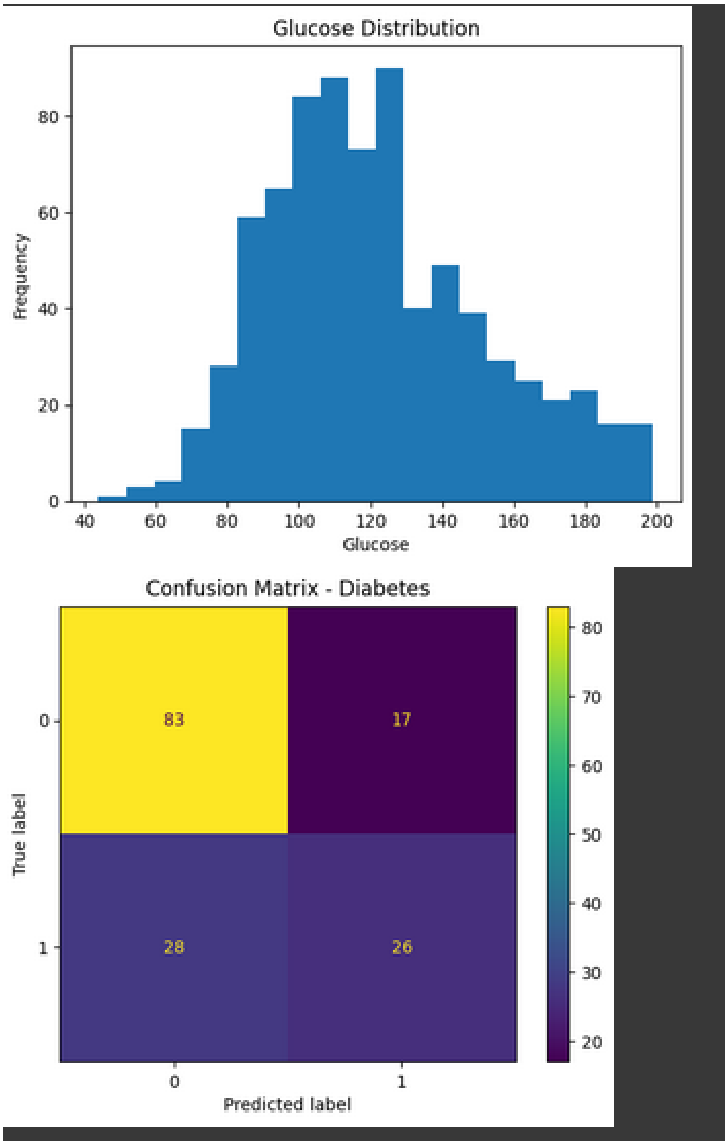

Figure 12: Diabetes Prediction Output

```
--- Diabetes Prediction ---
Accuracy: 0.7077922077922078
[[83 17]
 [28 26]]
              precision    recall  f1-score   support

           0       0.75      0.83      0.79       100
           1       0.60      0.48      0.54        54

    accuracy                           0.71       154
   macro avg       0.68      0.66      0.66       154
weighted avg       0.70      0.71      0.70       154
```

Figure 13: Diabetes Prediction Performance

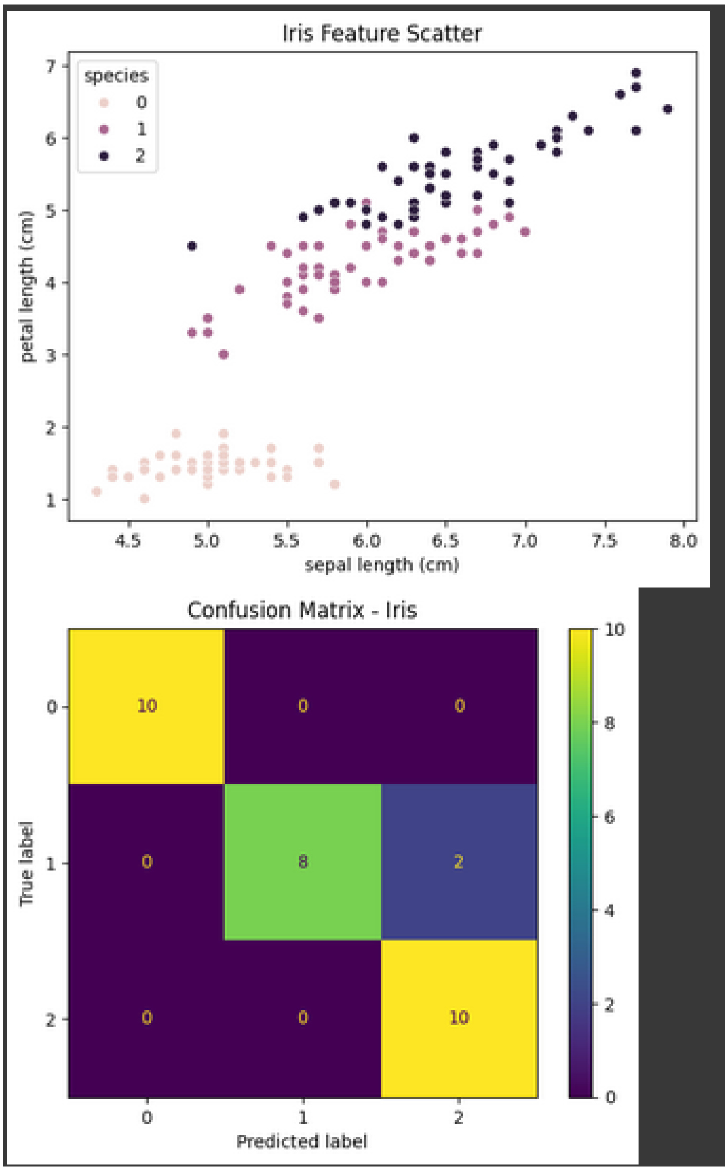

Figure 14: Iris Classification Output

```
--- Iris Dataset Classification ---
Accuracy: 0.9333333333333333
[[10  0  0]
 [ 0  8  2]
 [ 0  0 10]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.80      0.89        10
           2       0.83      1.00      0.91        10

    accuracy                           0.93        30
   macro avg       0.94      0.93      0.93        30
weighted avg       0.94      0.93      0.93        30
```

Figure 15: Iris Classification Performance

Inference Table

Dataset	**Model / ML Algorithm Used**	**Inference Summary**
Loan Amount Prediction	Linear Regression	Predicts continuous loan amount using numerical and categorical features
Handwritten Digit Recognition	Random Forest Classifier	Achieved high accuracy on multi-class digit images from the MNIST dataset
Email Spam Classification	Multinomial Naive Bayes	Efficiently classifies text messages into ham or spam
Diabetes Prediction	Logistic Regression	Binary classification model for diabetes presence, good performance post scaling
Iris Dataset	Support Vector Classifier (SVC)	Clearly distinguishes species using feature separation, high accuracy

Learning Outcomes

- Learned to preprocess datasets using feature scaling and handling missing values.
- Applied multiple ML models and understood when to use each.

- Practiced evaluating regression with MSE and R^2, and classification with accuracy/confusion matrix.
- Gained practical experience using popular Python libraries in machine learning workflows.