****

**Assessment Report**

on

**Spam Email Detection Using Structured Metadata**

**Name**: Abhilash Raj  
**Roll Number**: 202401100400007  
**Course**: B.Tech (1st Year)

**Branch** : CSE(AIML) - A  
**Subject**: AI  
**Project Type**: Mini Project  
**Submission Date**: 22/04/2025

**Under the supervision of**

“BIKKI KUMAR GUPTA”

**Introduction-:**

Email is one of the most widely used forms of communication, but it is also highly vulnerable to spam. Spam emails not only waste time but also pose a serious risk of phishing and malware attacks. Therefore, it is essential to build a system that can automatically classify emails as spam or not spam using available metadata like number of links, attachments, and sender reputation.

This project implements a machine learning-based solution that uses structured metadata from emails to classify them as either Spam (1) or Not Spam (0). We use basic and intermediate Python techniques with popular libraries such as pandas, seaborn, matplotlib, and sklearn.

**Methodology-:**

**Dataset**: A .csv file containing structured metadata for each email, such as:

* num\_links: Number of links in the email
* num\_attachments: Number of attachments
* sender\_reputation: Numeric score indicating sender’s trustworthiness
* is\_spam: Target label (yes or no)

**Steps**:

* Load and inspect the dataset.
* Convert the target variable is\_spam into numerical format.
* Visualize data using **pie charts** and **bar plots**.
* Prepare features and labels for model training.
* Split data into training and testing sets.
* Train a **Decision Tree Classifier**.
* Evaluate using **accuracy**, **classification report**, and **confusion matrix**.
* Make a sample prediction.

**Code -:**

# Step 1: Import libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Step 2: Load the dataset

df = pd.read\_csv("/content/your\_dataset.csv") # Update path if needed

# Step 3: Display basic info

print("🔹 Dataset Info:")

print(df.info())

print("\n🔹 First 5 Rows:")

print(df.head())

# Step 4: Handle missing data

print("\n🔹 Missing Values:")

print(df.isnull().sum())

# Step 5: Convert target variable

df['is\_spam'] = df['is\_spam'].map({'yes': 1, 'no': 0})

# Step 6: Plot pie chart of spam vs not spam

spam\_counts = df['is\_spam'].value\_counts()

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

plt.pie(spam\_counts, labels=['Not Spam', 'Spam'], autopct='%1.1f%%', colors=['lightgreen', 'lightcoral'])

plt.title('📧 Spam vs Not Spam Emails')

plt.show()

# Step 7: Bar plots for features

plt.figure(figsize=(8, 4))

sns.barplot(data=df, x='is\_spam', y='num\_links', estimator=np.mean)

plt.title('📊 Avg Number of Links by Email Type')

plt.xlabel('Email Type (0=Not Spam, 1=Spam)')

plt.ylabel('Average Links')

plt.show()

plt.figure(figsize=(8, 4))

sns.barplot(data=df, x='is\_spam', y='num\_attachments', estimator=np.mean)

plt.title('📎 Avg Attachments by Email Type')

plt.xlabel('Email Type (0=Not Spam, 1=Spam)')

plt.ylabel('Average Attachments')

plt.show()

plt.figure(figsize=(8, 4))

sns.barplot(data=df, x='is\_spam', y='sender\_reputation', estimator=np.mean)

plt.title('💼 Avg Sender Reputation by Email Type')

plt.xlabel('Email Type (0=Not Spam, 1=Spam)')

plt.ylabel('Average Reputation')

plt.show()

# Step 8: Prepare features and target

X = df[['num\_links', 'num\_attachments', 'sender\_reputation']]

y = df['is\_spam']

# Step 9: Train/Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Step 10: Train the model

model = DecisionTreeClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Step 11: Make predictions

y\_pred = model.predict(X\_test)

# Step 12: Evaluate model

print("\n✅ Model Accuracy:", accuracy\_score(y\_test, y\_pred))

print("\n📃 Classification Report:\n", classification\_report(y\_test, y\_pred))

# Step 13: Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])

plt.title('🔍 Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

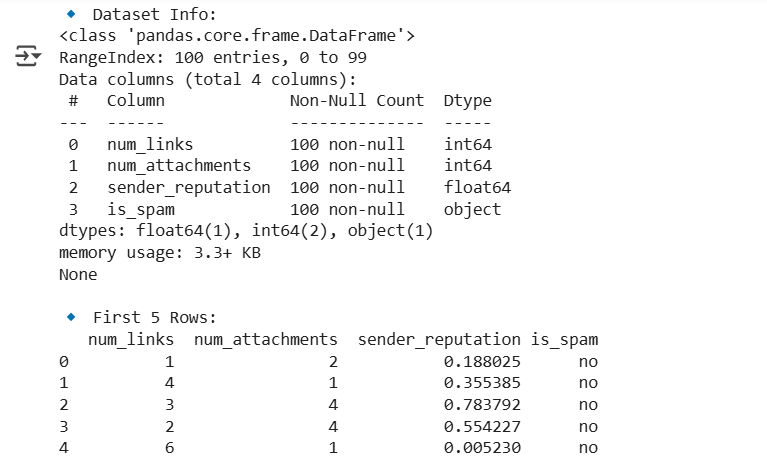
# Step 14: Predict custom input

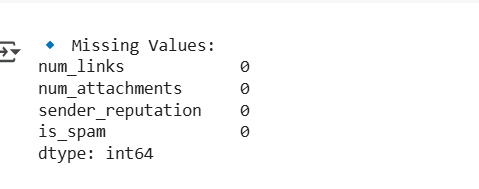
sample = [[2, 3, 0.5]]

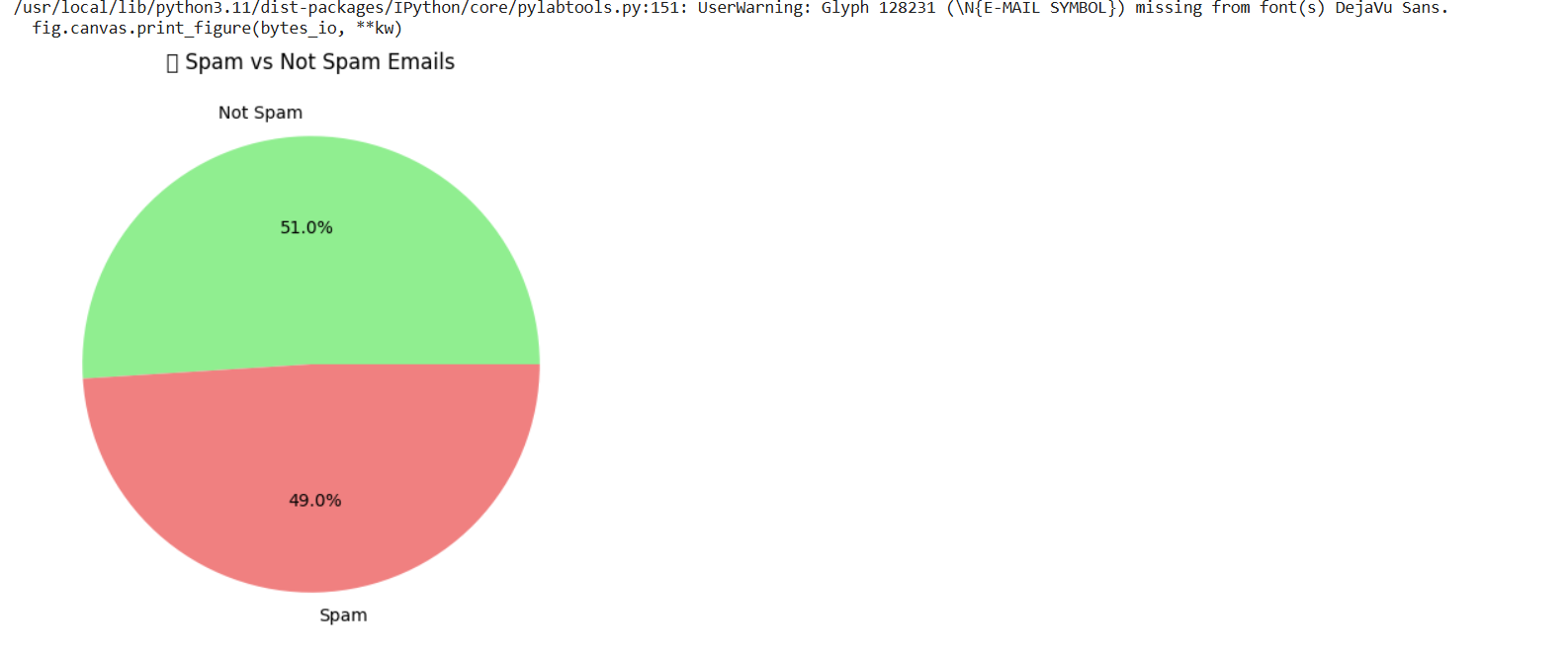
pred = model.predict(sample)[0]

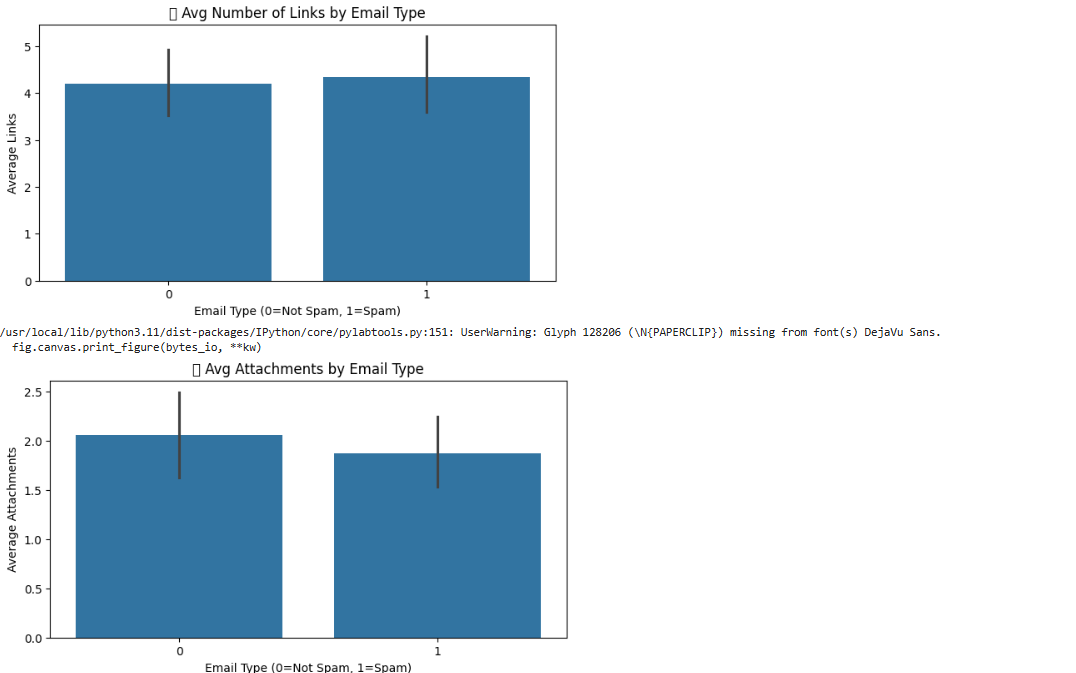
print("\n🔮 Prediction for sample [2 links, 3 attachments, 0.5 reputation]:", "Spam" if pred == 1 else "Not Spam")

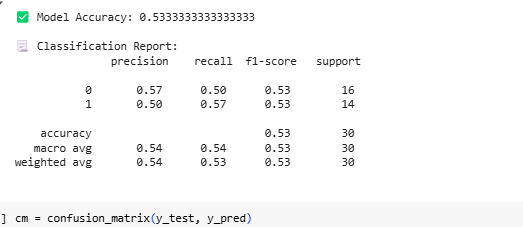
**Screenshots of Project Output-:**

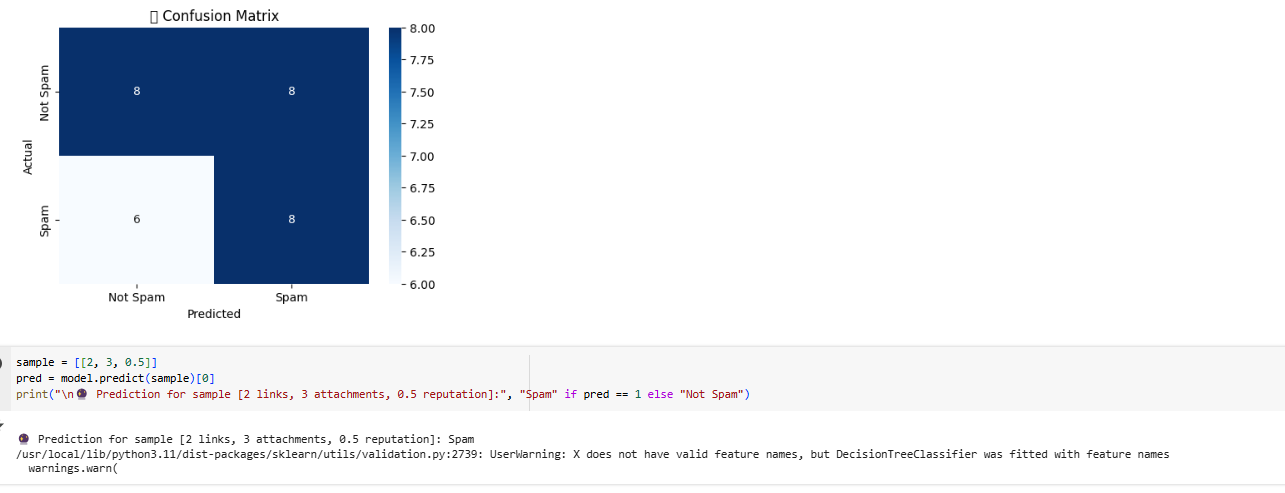
****

****

****

****

****

****

**References / Credits**

* Dataset provided by instructor / course resources.
* Python Libraries Used:
  + pandas – data loading & preprocessing
  + matplotlib, seaborn – visualizations
  + sklearn – machine learning (model, split, evaluation)
* Project support and code guidance from ChatGPT (OpenAI).