**Appendix for Chart Representations**

**Performance Metrics Line Chart**

import matplotlib.pyplot as plt

# Example data (replace with actual data)

iterations = [1, 2, 3, 4, 5]

accuracy = [0.75, 0.82, 0.89, 0.91, 0.95]

efficiency = [0.80, 0.85, 0.88, 0.92, 0.94]

scalability = [0.70, 0.78, 0.82, 0.88, 0.90]

# Plotting the line chart

plt.figure(figsize=(10, 6))

plt.plot(iterations, accuracy, marker='o', label='Accuracy')

plt.plot(iterations, efficiency, marker='o', label='Efficiency')

plt.plot(iterations, scalability, marker='o', label='Scalability')

# Adding labels and title

plt.xlabel('Iterations')

plt.ylabel('Metrics Score')

plt.title('Performance Metrics Over Iterations')

plt.legend()

plt.grid(True)

plt.show()

Figure 1

*This line chart represents the hypothetical performance metrics (accuracy, efficiency, and scalability) over different iterations of your project.*

**Data-Security and Privacy Vulnerabilities Matrix**

import numpy as np

import matplotlib.pyplot as plt

# Example data (replace with actual data)

vulnerabilities = ['Encryption', 'Access Controls', 'Privacy Assessment', 'Regular Audits']

mitigation\_scores = np.array([[4, 5, 3, 4], # Tool 1

[3, 4, 5, 3], # Tool 2

[5, 3, 4, 5]]) # Tool 3

# Plotting the matrix chart

fig, ax = plt.subplots(figsize=(10, 6))

cax = ax.matshow(mitigation\_scores, cmap='viridis')

# Adding labels

plt.xticks(np.arange(len(vulnerabilities)), vulnerabilities)

plt.yticks(np.arange(len(mitigation\_scores)), ['Tool 1', 'Tool 2', 'Tool 3'])

plt.xlabel('Vulnerabilities')

plt.ylabel('Mitigation Tools')

# Adding colorbar

cbar = fig.colorbar(cax)

# Adding title

plt.title('Data-Security and Privacy Vulnerabilities Matrix')

plt.show()

Figure 2

*This matrix chart represents the hypothetical mitigation scores for different data-security and privacy vulnerabilities, with tools on one axis and vulnerabilities on the other. The color intensity indicates the effectiveness of each tool in mitigating specific vulnerabilities.*

**Appendix for Coding Report**

**Automated EEG Classification**

1. **Data Preprocessing**

# Import necessary libraries

import pandas as pd

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

from sklearn.feature\_extraction.text import TfidfVectorizer

# Load EEG data and text reports

data = pd.read\_csv('eeg\_data.csv’) # Update with the actual file name

X = data['text\_report']

y = data['abnormal\_activity\_label']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Convert text data to TF-IDF features

vectorizer = TfidfVectorizer()

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

1. **Model Training and Evaluation**

# Import machine learning models and metrics

from sklearn.svm import SVC

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

# Train a Support Vector Machine (SVM) classifier

classifier = SVC(kernel='linear')

classifier.fit(X\_train\_tfidf, y\_train)

# Make predictions on the test set

y\_pred = classifier.predict(X\_test\_tfidf)

# Evaluate the performance of the model

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print(f'Accuracy: {accuracy}')

print('Classification Report:\n', report)

1. **Hyperparameter Tuning**

# Import GridSearchCV for hyperparameter tuning

from sklearn.model\_selection import GridSearchCV

# Define parameter grid for SVM

param\_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}

# Perform GridSearchCV

grid\_search = GridSearchCV(SVC(), param\_grid, cv=5)

grid\_search.fit(X\_train\_tfidf, y\_train)

# Get the best parameters

best\_params = grid\_search.best\_params\_

print(f'Best Hyperparameters: {best\_params}')

*This coding report provides a starting point for addressing the client's objectives and considerations in automating the cataloging of EEG reports. It's essential to continue refining the model, considering ethical implications, and ensuring compliance with regulations throughout the development process.*

Recommendations

Based on the evaluation results and hyperparameter tuning, it is recommended to use a linear SVM classifier with the identified hyperparameters for the automated EEG classification.