Code Implementation

1. PII Masking (Masking Personal Identifiable Information)

pii_masking.py

This Python script is responsible for detecting and masking PII (Personal Identifiable Information) entities like names, email addresses, phone numbers, etc., using regular expressions (regex).

```
import re


# PII detection patterns

PII_PATTERNS = {

    "full_name": r"\b([A-Z][a-z]*\s[A-Z][a-z]*)\b",  # Simple name pattern

    "email": r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b",

    "phone_number": r"\b\d{10}\b",  # 10 digit phone number

    "dob": r"\b\d{2}/\d{2}/\d{4}\b",  # Date format: DD/MM/YYYY

    "aadhar_num": r"\b\d{12}\b",  # Aadhar number: 12 digits

    "credit_debit_no": r"\b\d{16}\b",  # Credit/Debit card number

    "cvv_no": r"\b\d{3}\b",  # CVV (3 digit)

    "expiry_no": r"\b\d{4}/\d{2}\b",  # Expiry date: MM/YY

}


def mask_pii(email_text: str):

    masked_email = email_text

    entities = []


    # Mask PII using regex patterns

    for entity, pattern in PII_PATTERNS.items():

        matches = re.finditer(pattern, email_text)

        for match in matches:

            start, end = match.span()

            original_entity = match.group(0)

            # Mask the found PII entity
            masked_email = masked_email[:start] + f"[{entity}]" + masked_email[end:]

            entities.append({

                "position": [start, end],

                "classification": entity,

                "entity": original_entity

            })


    return masked_email, entities
```

Explanation of Code:

- **PII_PATTERNS**: A dictionary where each key represents a type of PII (e.g., full name, email, phone number) and the corresponding value is the regular expression used to detect it.
- **mask_pii()**: This function takes an email body as input, runs it through the regex patterns, masks the detected PII, and stores information about the entity (position, classification, original value).

---

2. Email Classification

The email classification script uses **Random Forest Classifier** to categorize emails into predefined categories.

email_classifier.py

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics import accuracy_score, classification_report


# Load the dataset (Ensure you have a .csv file with email data)

def load_data():

    # Load your email dataset here

    df = pd.read_csv('email_data.csv')

    return df


# Preprocess the email data (convert to lowercase, remove stop words, etc.)
```

```
def preprocess_data(df):
    df['email_text'] = df['email_text'].str.lower()  # Convert to lowercase
    df['email_text'] = df['email_text'].str.replace('[^\w\s]', '')  # Remove punctuation
    return df

# Train the model
def train_model(df):
    # Vectorize the email text
    vectorizer = TfidfVectorizer(stop_words='english')
    X = vectorizer.fit_transform(df['email_text'])
    y = df['category']

    # Train-Test Split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Initialize the classifier
    classifier = RandomForestClassifier()

    # Train the model
    classifier.fit(X_train, y_train)

    # Make predictions
    y_pred = classifier.predict(X_test)

    # Evaluate the model
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
    print(classification_report(y_test, y_pred))

    return classifier, vectorizer

# Predict the category of an email
```

```
def predict_email_category(email_text, model, vectorizer):
    email_vec = vectorizer.transform([email_text])
    prediction = model.predict(email_vec)
    return prediction[0]
```

Explanation of Code:

- **load_data()**: Loads the dataset containing emails and their respective categories (e.g., technical support, billing).
- **preprocess_data()**: Preprocesses the text data by converting it to lowercase and removing punctuation.
- **train_model()**: Vectorizes the email text using TfidfVectorizer and then trains a RandomForestClassifier on the data.
- **predict_email_category()**: This function takes an email body, transforms it using the trained vectorizer, and then predicts its category using the trained model.

---

3. API Implementation

FastAPI is used to expose the email classification model and PII masking functionality through an API.

api.py

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import email_classifier
import pii_masking

# Initialize the FastAPI app
app = FastAPI()

# Load the trained model and vectorizer
model, vectorizer = email_classifier.train_model(email_classifier.load_data())

# Pydantic model for incoming email data
class Email(BaseModel):
    body: str
```

```
# Endpoint for classifying an email
@app.post("/classify_email/")
async def classify_email(email: Email):
    try:
        # Mask PII
        masked_email, entities = pii_masking.mask_pii(email.body)

        # Predict email category
        category = email_classifier.predict_email_category(masked_email, model, vectorizer)

        return {
            "input_email_body": email.body,
            "list_of_masked_entities": entities,
            "masked_email": masked_email,
            "category_of_the_email": category
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Explanation of Code:

- **FastAPI Setup**: The FastAPI app is created using FastAPI().
- **Pydantic Model**: A Pydantic model Email is used to validate and define the structure of the incoming request body.
- **/classify_email/**: This POST endpoint takes an email body, masks any PII, and predicts the email category using the trained model. The response includes the original email body, masked email body, list of masked entities, and the predicted category.

---

4. Deployment Instructions

Prerequisites:

- Python 3.x installed
- Install dependencies using the requirements.txt file

requirements.txt

```
fastapi
```

uvicorn

scikit-learn

pandas

pydantic

1. **Install dependencies**: Run the following command to install the necessary packages:
2. pip install -r requirements.txt
3. **Run the FastAPI app**: You can run the FastAPI app locally using uvicorn by running:
4. uvicorn api:app --reload
   - This will start the API server locally at http://127.0.0.1:8000.
5. **Test the API**:
   - Use Postman or cURL to send a POST request to http://127.0.0.1:8000/classify_email/ with the email body in the JSON format:
6. {
7.   "body": "My name is John Doe, and my email is johndoe@example.com. My phone number is 1234567890."
8. }

Documentation for Scripts/Modules

1. pii_masking.py

Purpose:

This script provides a clear and simple method to detect and mask Personal Identifiable Information (PII) from email content using regular expressions (regex).

Functions:

- **mask_pii(email_text: str)**:
  - **Description**: This function receives an email's content as input, identifies any PII entities (e.g., full name, email, phone number, etc.), and masks them. The positions and types of the entities found are also stored for potential demasking.
  - **Parameters**:
    - email_text (str): The body of the email in which PII needs to be masked.
  - **Returns**:
    - masked_email (str): The email content with the PII masked using tags like [full_name], [email], etc.

        - entities (list): A list of dictionaries where each dictionary contains the position, classification, and original entity for each detected PII.

Example Usage:

```
from pii_masking import mask_pii


email_text = "My name is John Doe, and my email is johndoe@example.com."

masked_email, entities = mask_pii(email_text)


print(masked_email)

# Output: "My name is [full_name], and my email is [email]."


print(entities)

# Output: [{'position': [11, 20], 'classification': 'full_name', 'entity': 'John Doe'}, {'position': [45, 67], 'classification': 'email', 'entity': 'johndoe@example.com'}]
```

---

2. email_classifier.py

Purpose:

This script contains the entire pipeline for training and using a Random Forest Classifier to categorize emails into predefined categories such as "Technical Support", "Billing", "Account Management", etc.

Functions:

- **load_data()**:
    - **Description**: This function loads the email dataset (in CSV format) containing emails and their respective categories.
    - **Returns**:
        - df (pandas.DataFrame): A DataFrame containing email data with columns for email text and categories.
- **preprocess_data(df)**:
    - **Description**: Preprocesses the email data by converting the text to lowercase and removing unnecessary punctuation.
    - **Parameters**:
        - df (pandas.DataFrame): The DataFrame containing the raw email data.
    - **Returns**:

        - df (pandas.DataFrame): The preprocessed email data with text converted to lowercase and cleaned of punctuation.

- **train_model(df)**:
    - **Description**: Trains a Random Forest Classifier using TF-IDF vectorization of email text. It also splits the data into training and test sets, then evaluates the model's performance.
    - **Parameters**:
        - df (pandas.DataFrame): The preprocessed email data with email text and categories.
    - **Returns**:
        - classifier (RandomForestClassifier): The trained Random Forest model.
        - vectorizer (TfidfVectorizer): The vectorizer used to transform email text into features.
- **predict_email_category(email_text, model, vectorizer)**:
    - **Description**: This function uses the trained model and vectorizer to predict the category of a new email based on its content.
    - **Parameters**:
        - email_text (str): The email body to be classified.
        - model (RandomForestClassifier): The trained model.
        - vectorizer (TfidfVectorizer): The vectorizer used to transform email text into feature vectors.
    - **Returns**:
        - category (str): The predicted category for the email (e.g., "Technical Support", "Billing", etc.).

Example Usage:

```
from email_classifier import load_data, preprocess_data, train_model, predict_email_category


# Load and preprocess the data

df = load_data()

df = preprocess_data(df)


# Train the model

model, vectorizer = train_model(df)
```

Predict the category of a new email

email_text = "I need help with my billing issue."

category = predict_email_category(email_text, model, vectorizer)

print(category) # Output: "Billing"

---

3. api.py

Purpose:

This script exposes the model and PII masking functionalities through a FastAPI-based API. It provides a /classify_email/ POST endpoint that allows clients to send email bodies and receive classified categories and masked PII in response.

Functions:

- **/classify_email/** (POST endpoint):
    - **Description**: This endpoint accepts an email body, applies PII masking, classifies the email, and returns both the masked email and the predicted category along with the details of the masked PII entities.
    - **Request Body** (JSON):
    - {
    -   "body": "My name is John Doe and I need technical support."
    - }
    - **Response Body** (JSON):
    - {
    -   "input_email_body": "My name is John Doe and I need technical support.",
    -   "list_of_masked_entities": [
    -     {"position": [11, 20], "classification": "full_name", "entity": "John Doe"}
    -   ],
    -   "masked_email": "My name is [full_name] and I need technical support.",
    -   "category_of_the_email": "Technical Support"
    - }

Explanation of Code:

- **FastAPI Setup**: We create a FastAPI app and set up a POST endpoint at /classify_email/ to process email data.

- **PII Masking**: The email body is passed to the mask_pii() function, which returns the masked email and details of any detected PII entities.
- **Email Classification**: The masked email text is then passed to the predict_email_category() function, which returns the predicted category.
- **Error Handling**: If any error occurs during the process, a 500 HTTP status code is returned along with the error details.

Example Usage:

To run the API:

1. Install the required dependencies:
2. pip install fastapi uvicorn scikit-learn pandas
3. Run the API server:
4. uvicorn api:app --reload
5. Test the endpoint using Postman or cURL by sending a POST request to http://127.0.0.1:8000/classify_email/ with the following JSON body:
6. {
7.   "body": "I need help with my technical support issue."
8. }
