SPAM EMAIL CLASSIFIER

SOURCE CODE

```
"""
preprocessor.py - Module for text preprocessing and cleaning
Functional Requirement: Data input & processing
"""

import re
import string
import logging
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
import nltk

# Download required NLTK data
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')

try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)


class TextPreprocessor:
    """Preprocess and clean email text data."""

    def __init__(self):
        """Initialize TextPreprocessor with stopwords."""
        self.stop_words = set(stopwords.words('english'))
        self.punctuation = set(string.punctuation)

    def clean_text(self, text):
        """
        Clean and normalize text.
```

```
        Args:
            text (str): Raw text to clean

        Returns:
            str: Cleaned text
        """
        if not isinstance(text, str):
            return ""

        # Convert to lowercase
        text = text.lower()

        # Remove URLs
        text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

        # Remove email addresses
        text = re.sub(r'\S+@\S+', '', text)

        # Remove numbers
        text = re.sub(r'\d+', '', text)

        # Remove extra whitespace
        text = re.sub(r'\s+', ' ', text).strip()

        return text

    def tokenize(self, text):
        """
        Tokenize text into words.

        Args:
            text (str): Text to tokenize

        Returns:
            list: List of tokens
        """
        tokens = word_tokenize(text)
        return tokens

    def remove_stopwords(self, tokens):
```

```python
    def remove_stopwords(self, tokens):
        """
        Remove common stopwords and punctuation.

        Args:
            tokens (list): List of tokens

        Returns:
            list: Filtered tokens
        """
        filtered = [
            token for token in tokens
            if token not in self.stop_words and token not in self.punctuation
            and len(token) > 2
        ]
        return filtered

    def preprocess(self, text):
        """
        Full preprocessing pipeline.

        Args:
            text (str): Raw text to preprocess

        Returns:
            str: Preprocessed text (space-separated tokens)
        """
        # Clean
        text = self.clean_text(text)

        # Tokenize
        tokens = self.tokenize(text)

        # Remove stopwords
        tokens = self.remove_stopwords(tokens)

        # Join back to string
        return ' '.join(tokens)

    def preprocess_batch(self, texts):
        """
        Preprocess multiple texts.

        Args:
```

```
        Args:
            texts (list): List of text strings

        Returns:
            list: List of preprocessed texts
        """
        return [self.preprocess(text) for text in texts]


def validate_preprocessing(original, preprocessed):
    """
    Validate preprocessing output.

    Args:
        original (str): Original text
        preprocessed (str): Preprocessed text

    Returns:
        dict: Validation statistics
    """
    return {
        'original_length': len(original),
        'preprocessed_length': len(preprocessed),
        'reduction_ratio': (1 - len(preprocessed)/len(original)) * 100 if len(original) > 0 else 0,
        'is_valid': len(preprocessed) > 0
    }


if __name__ == '__main__':
    # Example usage
    preprocessor = TextPreprocessor()

    sample_emails = [
        "You have WON $1000! Click here NOW to claim your PRIZE!!!",
        "Hi John, let's meet tomorrow at 2pm to discuss the project",
        "URGENT: Your account has been compromised. Update password NOW!"
    ]

    logger.info("\n===== TEXT PREPROCESSING DEMO =====")
    for email in sample_emails:
        preprocessed = preprocessor.preprocess(email)
        stats = validate_preprocessing(email, preprocessed)
        logger.info(f"Original: {email[:50]}...")
        logger.info(f"Processed: {preprocessed[:50]}...")
        logger.info(f"Stats: {stats}\n")
```

Source code -2 (DATA LOADER)

```python
"""
data_loader.py - Module for loading and exploring email dataset
Functional Requirement: Data input & processing
"""

import pandas as pd
import numpy as np
from pathlib import Path
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)


class DataLoader:
    """Load and explore email dataset from CSV files."""

    def __init__(self, data_path='data/emails.csv'):
        """
        Initialize DataLoader.

        Args:
            data_path (str): Path to CSV file containing email data
        """
        self.data_path = Path(data_path)
        self.data = None
        self.stats = {}

    def load_data(self):
        """
        Load email dataset from CSV.

        Returns:
            pd.DataFrame: Loaded dataset

        Raises:
            FileNotFoundError: If data file doesn't exist
            pd.errors.EmptyDataError: If CSV is empty
        """
        try:
            if not self.data_path.exists():
                logger.warning(f"Data file not found at {self.data_path}")
                logger.info("Generating sample dataset...")
                self.data = self._generate_sample_data()
                return self.data
```

```
        except pd.errors.EmptyDataError:
            logger.error("CSV file is empty")
            raise

    def _generate_sample_data(self):
        """Generate sample dataset for demonstration."""
        sample_emails = {
            'id': range(1, 101),
            'email_subject': [
                'Great Offer!', 'Meeting Tomorrow', 'URGENT: Claim Prize',
                'Hi there', 'FREE MONEY NOW', 'Project Update',
                'Click here NOW', 'Team Sync', 'You Won!',
                'Budget Report'
            ] * 10,
            'email_body': [
                'Buy now and save 50%! Limited time offer',
                'Hi, let\'s meet at 2pm tomorrow to discuss the project',
                'You have won $1000! Click here to claim your prize',
                'How are you doing? Just checking in',
                'Get free money instantly! No catch!',
                'Here is the Q3 budget update for review',
                'Act now! This offer expires in 1 hour!',
                'Can we sync up at 3pm? Agenda: project timeline',
                'CONGRATULATIONS! You are selected for $500000 prize',
                'Please review attached budget for next quarter'
            ] * 10,
            'label': ['spam', 'ham', 'spam', 'ham', 'spam', 'ham',
                      'spam', 'ham', 'spam', 'ham'] * 10
        }
        return pd.DataFrame(sample_emails)

    def explore_data(self):
        """
        Explore dataset statistics and characteristics.

        Returns:
            dict: Statistics about the dataset
        """
        if self.data is None:
            self.load_data()

        self.stats = {
            'total_emails': len(self.data),
            'spam_count': (self.data['label'] == 'spam').sum(),
            'ham_count': (self.data['label'] == 'ham').sum(),
            'spam_percentage': (self.data['label'] == 'spam').sum() / len(self.data) * 100,
```

```
    logger.info(f"\n===== DATASET EXPLORATION =====")
    logger.info(f"Total emails: {self.stats['total_emails']}")
    logger.info(f"Spam emails: {self.stats['spam_count']} ({self.stats['spam_percentage']:.1f}%)")
    logger.info(f"Ham emails: {self.stats['ham_count']} ({100-self.stats['spam_percentage']:.1f}%)")
    logger.info(f"Missing values: {self.stats['missing_values']}")

    return self.stats

def get_data(self):
    """
    Get loaded dataset.

    Returns:
        pd.DataFrame: Email dataset
    """
    if self.data is None:
        self.load_data()
    return self.data

def get_split(self, test_size=0.2, random_state=42):
    """
    Split dataset into train and test sets.

    Args:
        test_size (float): Proportion of test set (0-1)
        random_state (int): Random seed for reproducibility

    Returns:
        tuple: (X_train, X_test, y_train, y_test)
    """
    from sklearn.model_selection import train_test_split

    if self.data is None:
        self.load_data()

    # Combine subject and body for text features
    X = self.data['email_subject'] + ' ' + self.data['email_body']
    y = self.data['label']

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state, stratify=y
```

```
        logger.info(f"\n===== DATA SPLIT =====")
        logger.info(f"Training set: {len(X_train)} emails")
        logger.info(f"Test set: {len(X_test)} emails")

        return X_train, X_test, y_train, y_test


if __name__ == '__main__':
    # Example usage
    loader = DataLoader()
    loader.load_data()
    loader.explore_data()
    X_train, X_test, y_train, y_test = loader.get_split()
```

OUTPUT

```
WARNING:__main__:Data file not found at data\emails.csv
INFO:__main__:Generating sample dataset...
INFO:__main__:
===== DATASET EXPLORATION =====
INFO:__main__:Total emails: 100
INFO:__main__:Spam emails: 50 (50.0%)
INFO:__main__:Ham emails: 50 (50.0%)
INFO:__main__:Missing values: {'id': 0, 'email_subject': 0, 'email_body': 0, 'label': 0}
INFO:__main__:
===== DATA SPLIT =====
INFO:__main__:Training set: 80 emails
INFO:__main__:Test set: 20 emails
```