Line-by-Line Explanation: Spam Classifier

1■■ Importing Necessary Libraries

```
import os
import io
import numpy
import pandas as pd
from pandas import DataFrame
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
```

- os: Handles file operations and directory traversal.
- io: Used to read email files.
- numpy: Useful for numerical computations.
- pandas: Used to structure the dataset into a DataFrame.
- CountVectorizer: Converts email text into numerical data for ML models.
- MultinomialNB: A Naïve Bayes classifier used for spam detection.

2■■ Function to Read Email Files

```
def readFiles(path):
    for root, dirnames, filenames in os.walk(path):
        for filename in filenames:
            path = os.path.join(root, filename)
            inBody = False
            lines = []
            f = io.open(path, 'r', encoding='latin1')
            for line in f:
                if inBody:
                    lines.append(line)
                elif line == '\n':
                    inBody = True
            f.close()
            message = '\n'.join(lines)
            yield path, message
```

- Reads email files from a given directory.
- Uses os.walk() to traverse the file structure.
- Opens files with io.open() using latin1 encoding.
- Skips headers and extracts only the email body (after the first empty line \n).
- Uses yield to process one email at a time, saving memory.

3■■ Function to Convert Emails into a DataFrame

```
def dataFrameFromDirectory(path, classification):
    rows = []
    index = []
    for filename, message in readFiles(path):
        rows.append({'message': message, 'class': classification})
        index.append(filename)
    return DataFrame(rows, index=index)
```

- Calls readFiles() to retrieve email data.
- Stores email content and its classification (spam or ham) in a list.
- Converts the list into a Pandas DataFrame.

4■■ Creating the Final DataFrame

```
data = DataFrame({'message': [], 'class': []})
data = pd.concat([data, dataFrameFromDirectory('emails/spam', 'spam')])
data = pd.concat([data, dataFrameFromDirectory('emails/ham', 'ham')])
```

- Initializes an empty DataFrame.
- Appends spam emails from 'emails/spam' and labels them 'spam'.
- Appends ham emails from 'emails/ham' and labels them 'ham'.
- Uses pd.concat() to merge them into a single dataset.

■ Why yield Instead of return?
- yield processes one file at a time instead of storing all in memory.
- If we had millions of emails, return would load everything at once, using too much RAM.
- Using yield = more efficient and scalable!

■ Final Takeaways:
■ Extracts email text from files efficiently.
■ Stores messages in a structured DataFrame for ML processing.
■ Uses Naïve Bayes for spam classification.
■ Uses yield to handle large datasets efficiently.