In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from scipy.stats import randint

# Load the CSV file into a pandas DataFrame
df = pd.read_csv('emails.csv')

# Display the first few rows of the DataFrame
print(df.head())
```

```
  Email No.  the  to  ect  and  for  of    a  you  hou  ...  connevey  jay  \
0   Email 1    0   0    1    0    0   0    2    0    0  ...         0    0   
1   Email 2    8  13   24    6    6   2  102    1   27  ...         0    0   
2   Email 3    0   0    1    0    0   0    8    0    0  ...         0    0   
3   Email 4    0   5   22    0    5   1   51    2   10  ...         0    0   
4   Email 5    7   6   17    1    5   2   57    0    9  ...         0    0   

   valued  lay  infrastructure  military  allowing  ff  dry  Prediction  
0       0    0               0         0         0   0    0           0  
1       0    0               0         0         0   1    0           0  
2       0    0               0         0         0   0    0           0  
3       0    0               0         0         0   0    0           0  
4       0    0               0         0         0   1    0           0  

[5 rows x 3002 columns]
```

In [2]:

```
# Check for missing values
print(df.isnull().sum())
```

```
Email No.     0
the           0
to            0
ect           0
and           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3002, dtype: int64
```

In [3]:

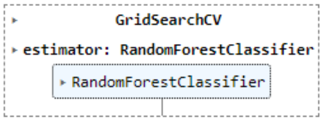
```
# Identify non-numeric columns
non_numeric_cols = df.select_dtypes(include=['object']).columns

#  Label Encoding for categorical features (for ordinal data)
label_encoders = {}
for col in non_numeric_cols:
    label_encoders[col] = LabelEncoder()
    df[col] = label_encoders[col].fit_transform(df[col])

# Assuming the last column is the target variable
X = df.iloc[:, :-1].values  # Features
y = df.iloc[:, -1].values   # Target variable

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the model
model = RandomForestClassifier(random_state=42)
```

Grid Search

In [4]:
```
# Grid Search
# Define the grid of hyperparameters
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}
```

In [5]:
```
# Perform grid search
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
```

Out[5]:



In [6]:
```
# Best parameters from grid search
print("Best Parameters from Grid Search:", grid_search.best_params_)
```

```
Best Parameters from Grid Search: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}
```

In [7]:
```
# Best model from grid search
best_grid_model = grid_search.best_estimator_
```

In [8]:
```
# Predict on test set using the best model from grid search
y_pred_grid = best_grid_model.predict(X_test)
```

Model Evaluation

In [9]:
```
# Evaluate the grid search model
accuracy_grid = accuracy_score(y_test, y_pred_grid)
precision_grid = precision_score(y_test, y_pred_grid, average='weighted')
recall_grid = recall_score(y_test, y_pred_grid, average='weighted')
f1_grid = f1_score(y_test, y_pred_grid, average='weighted')

print(f"Grid Search - Accuracy: {accuracy_grid:.4f}")
print(f"Grid Search - Precision: {precision_grid:.4f}")
print(f"Grid Search - Recall: {recall_grid:.4f}")
print(f"Grid Search - F1 Score: {f1_grid:.4f}")
```

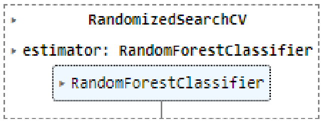
```
Grid Search - Accuracy: 0.9768
Grid Search - Precision: 0.9769
Grid Search - Recall: 0.9768
Grid Search - F1 Score: 0.9769
```

Random Search

In [10]:
```
# Random Search
# Define the distribution of hyperparameters
param_dist = {
    'n_estimators': randint(100, 200),
    'max_depth': [None, 10, 20],
    'min_samples_split': randint(2, 6)
}
```

In [11]:
```
# Perform random search
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter=50, cv=3, scoring='accuracy', random_
random_search.fit(X_train, y_train)
```

Out[11]:



In [12]:

```
# Best parameters from random search
print("Best Parameters from Random Search:", random_search.best_params_)
```

```
Best Parameters from Random Search: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 181}
```

In [13]:

```
# Best model from random search
best_random_model = random_search.best_estimator_
```

In [14]:

```
# Predict on test set using the best model from random search
y_pred_random = best_random_model.predict(X_test)
```

Model Evaluation

In [15]:

```
# Evaluate the random search model
accuracy_random = accuracy_score(y_test, y_pred_random)
precision_random = precision_score(y_test, y_pred_random, average='weighted')
recall_random = recall_score(y_test, y_pred_random, average='weighted')
f1_random = f1_score(y_test, y_pred_random, average='weighted')

print(f"Random Search - Accuracy: {accuracy_random:.4f}")
print(f"Random Search - Precision: {precision_random:.4f}")
print(f"Random Search - Recall: {recall_random:.4f}")
print(f"Random Search - F1 Score: {f1_random:.4f}")
```

```
Random Search - Accuracy: 0.9778
Random Search - Precision: 0.9779
Random Search - Recall: 0.9778
Random Search - F1 Score: 0.9778
```