spam email classification

Data Loading

```
import pandas as pd

# Load CSV file
df = pd.read_csv('emails.csv.zip')  # Use the correct path if
different

# Show first few rows and column info
print(df.head())
print(df.info())
```

```
  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]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5172 entries, 0 to 5171
Columns: 3002 entries, Email No. to Prediction
dtypes: int64(3001), object(1)
memory usage: 118.5+ MB
None
```

Data Preparation

```
# Drop identifier column
df = df.drop(columns=['Email No.'])  # Remove if not useful

# Separate features and label
X = df.drop('Prediction', axis=1)
y = df['Prediction']

# Split into training and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Optional: Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Selection and Training

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Initialize models
lr_model = LogisticRegression(max_iter=1000)
dt_model = DecisionTreeClassifier()
svm_model = SVC()

# Train models
lr_model.fit(X_train_scaled, y_train)
dt_model.fit(X_train, y_train)  # Tree doesn't need scaling
svm_model.fit(X_train_scaled, y_train)
```

```
SVC()
```

Model Evaluation

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    return {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred)
    }
```

```
# Evaluate all three models
results = {
    'Logistic Regression': evaluate_model(lr_model, X_test_scaled,
y_test),
    'Decision Tree': evaluate_model(dt_model, X_test, y_test),
    'SVM': evaluate_model(svm_model, X_test_scaled, y_test)
}
```

Pro Tips: Ensemble and Hyperparameter Tuning

```
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_results = evaluate_model(rf_model, X_test, y_test)

# Gradient Boosting
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)
gb_results = evaluate_model(gb_model, X_test, y_test)

# Optional: Grid Search for Logistic Regression
param_grid = {'C': [0.1, 1, 10]}
grid_search = GridSearchCV(LogisticRegression(max_iter=1000),
param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)
best_lr = grid_search.best_estimator_
grid_search_results = evaluate_model(best_lr, X_test_scaled, y_test)
```

** Predict a few emails and show results**

```
# Predict a few emails and show results

sample_emails = X_test.iloc[:5]
sample_labels = y_test.iloc[:5]
# Use the 'best_lr' model for prediction, as it was found using
GridSearchCV
# and trained on scaled data, which matches the scaler.transform
operation.
sample_predictions = best_lr.predict(scaler.transform(sample_emails))
sample_emails = X_test.iloc[:5]
sample_labels = y_test.iloc[:5]
# Remove the redundant line, as sample_emails and sample_labels are
already defined
# sample_predictions = model.predict(scaler.transform(sample_emails))
```

```python
print("\nEmail Classification Results:")
for i in range(len(sample_emails)):
    print(f"Email {i+1}: Actual = {sample_labels.iloc[i]}, Predicted =
{sample_predictions[i]}")
```

```
Email Classification Results:
Email 1: Actual = 0, Predicted = 0
Email 2: Actual = 0, Predicted = 0
Email 3: Actual = 1, Predicted = 1
Email 4: Actual = 0, Predicted = 0
Email 5: Actual = 0, Predicted = 0
```