```
import pandas as pd
df = pd.read_csv("mail_data.csv")

print(df.head())
df.info()
print(df['Category'].value_counts())

# %%
df = df.dropna(subset=['Category', 'Message'])


# %%
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.pipeline import Pipeline

# %%
df['Category'] = df['Category'].map({'ham': 0, 'spam': 1})
X_train, X_test, y_train, y_test = train_test_split(
    df['Message'], df['Category'],
    test_size=0.2, random_state=42, stratify=df['Category']
)
vectorizer = TfidfVectorizer(max_features=5000)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# %%
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier

# %%
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train_vec, y_train)

# %%
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, class_weight='balanced'),
    "Naive Bayes": MultinomialNB(),
    "SVM": SVC(kernel='linear', probability=True, class_weight='balanced'),
    "Random Forest": RandomForestClassifier(class_weight='balanced', n_estimators=100),
    "k-NN": KNeighborsClassifier(n_neighbors=5),
```

```
    "AdaBoost": AdaBoostClassifier(n_estimators=100)
}

# %%
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from time import time

results = {}

for name, model in models.items():
    print(f"Training {name}...")
    start_time = time()

    model.fit(X_train_bal, y_train_bal)

    y_pred = model.predict(X_test_vec)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    training_time = time() - start_time
    results[name] = {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1,
        'training_time': training_time
    }

    print(f"  Accuracy: {accuracy:.4f}")
    print(f"  Precision: {precision:.4f}")
    print(f"  Recall: {recall:.4f}")
    print(f"  F1 Score: {f1:.4f}")
    print(f"  Training Time: {training_time:.2f} seconds")
    print("-" * 50)



# %%
results_df = pd.DataFrame({model: metrics for model, metrics in results.items()}).T
results_df = results_df.sort_values('f1_score', ascending=False)
```

```
print("Models Performance Comparison:")
print(results_df)

plt.figure(figsize=(12, 8))
sns.heatmap(results_df[['accuracy', 'precision', 'recall', 'f1_score']], annot=True,
cmap='Blues')
plt.title('Model Performance Comparison')
plt.tight_layout()
plt.show()

best_model_name = results_df.index[0]
best_model = models[best_model_name]
y_pred_best = best_model.predict(X_test_vec)

plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred_best)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix - {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()

if best_model_name in ["Logistic Regression", "Random Forest", "AdaBoost"]:
    plt.figure(figsize=(10, 6))

    if best_model_name == "Logistic Regression":

        importance = np.abs(best_model.coef_[0])
        feature_names = np.arange(len(importance))
    elif best_model_name in ["Random Forest", "AdaBoost"]:
        importance = best_model.feature_importances_
        feature_names = np.arange(len(importance))


    indices = np.argsort(importance)[-20:]
    plt.barh(range(len(indices)), importance[indices])
    plt.yticks(range(len(indices)), feature_names[indices])
    plt.xlabel('Feature Importance')
    plt.title(f'Top 20 Important Features - {best_model_name}')
    plt.tight_layout()
    plt.show()

# %%

metrics = ['accuracy', 'precision', 'recall', 'f1_score']

plt.figure(figsize=(12, 8))
```

```
x = np.arange(len(models))
width = 0.2
multiplier = 0

for metric in metrics:
    offset = width * multiplier
    plt.bar(x + offset, results_df[metric], width, label=metric)
    multiplier += 1

plt.xlabel('Models')
plt.ylabel('Score')
plt.title('Model Performance Comparison')
plt.xticks(x + width, models, rotation=45, ha='right')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```