```
import pandas as pd
import json
import nltk
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

```
# Load dataset
def load_jsonl(filename):
    data = []
    with open(filename, 'r', encoding='utf-8') as file:
        for line in file:
            data.append(json.loads(line))
    return pd.DataFrame(data)

# Load training dataset
df = load_jsonl("train.jsonl")
```

```
# Extract relevant features
df = df[['postText', 'spoiler', 'tags']]

# Convert lists to strings
df['postText'] = df['postText'].apply(lambda x: x[0] if isinstance(x, list) else x)
df['spoiler'] = df['spoiler'].apply(lambda x: x[0] if isinstance(x, list) else x)

# Extract first tag if it's a list
df['tags'] = df['tags'].apply(lambda x: x[0] if isinstance(x, list) and len(x) > 0 else 'unknown')
```

```
# Encode target labels (spoiler type)
label_encoder = LabelEncoder()
df['tags'] = label_encoder.fit_transform(df['tags'])

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df['postText'], df['tags'], test_size=0.2, random_state=42)
```

```
# TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=5000, stop_words='english', ngram_range=(1,2))
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

```
# Train Logistic Regression
log_reg = LogisticRegression(max_iter=500)
log_reg.fit(X_train_tfidf, y_train)
y_pred_log_reg = log_reg.predict(X_test_tfidf)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_log_reg))
print(classification_report(y_test, y_pred_log_reg))
```

```
Logistic Regression Accuracy: 0.534375
              precision    recall  f1-score   support

           0       0.58      0.13      0.21       108
           1       0.54      0.58      0.56       259
           2       0.52      0.65      0.58       273

    accuracy                           0.53       640
   macro avg       0.55      0.45      0.45       640
weighted avg       0.54      0.53      0.51       640
```

```
# Train Support Vector Machine (SVM)
svm_model = SVC(kernel='linear')
svm_model.fit(X_train_tfidf, y_train)
y_pred_svm = svm_model.predict(X_test_tfidf)
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

```
SVM Accuracy: 0.5296875
                precision    recall  f1-score   support

             0       0.57      0.19      0.29       108
             1       0.53      0.57      0.55       259
             2       0.52      0.62      0.57       273

      accuracy                           0.53       640
     macro avg       0.54      0.46      0.47       640
  weighted avg       0.53      0.53      0.51       640
```

```
# Train Random Forest
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train_tfidf, y_train)
y_pred_rf = rf_model.predict(X_test_tfidf)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.534375
                precision    recall  f1-score   support

             0       0.49      0.19      0.28       108
             1       0.51      0.69      0.59       259
             2       0.57      0.52      0.55       273

      accuracy                           0.53       640
     macro avg       0.52      0.47      0.47       640
  weighted avg       0.53      0.53      0.52       640
```

```
# Define hyperparameter configurations
log_reg_configs = [0.1, 1, 10]  # Regularization strength C
svm_configs = ['linear', 'rbf']  # Kernel types
rf_configs = [100, 200]  # Number of estimators
```

```
# Store results
results = []

# Experiment with Logistic Regression
for C in log_reg_configs:
    log_reg = LogisticRegression(C=C, max_iter=500)
    log_reg.fit(X_train_tfidf, y_train)
    y_pred = log_reg.predict(X_test_tfidf)

    results.append(["Logistic Regression", f"C={C}",
                    accuracy_score(y_test, y_pred),
                    precision_score(y_test, y_pred, average='weighted'),
                    recall_score(y_test, y_pred, average='weighted'),
                    f1_score(y_test, y_pred, average='weighted')])
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# Experiment with SVM
for kernel in svm_configs:
    svm = SVC(kernel=kernel)
    svm.fit(X_train_tfidf, y_train)
    y_pred = svm.predict(X_test_tfidf)

    results.append(["SVM", f"Kernel={kernel}",
                    accuracy_score(y_test, y_pred),
                    precision_score(y_test, y_pred, average='weighted'),
                    recall_score(y_test, y_pred, average='weighted'),
                    f1_score(y_test, y_pred, average='weighted')])
```

```
# Experiment with Random Forest
for n_estimators in rf_configs:
    rf = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
    rf.fit(X_train_tfidf, y_train)
    y_pred = rf.predict(X_test_tfidf)

    results.append(["Random Forest", f"Estimators={n_estimators}",
```

```
                    accuracy_score(y_test, y_pred),
                    precision_score(y_test, y_pred, average='weighted'),
                    recall_score(y_test, y_pred, average='weighted'),
                    f1_score(y_test, y_pred, average='weighted')])
```

```
# Create a DataFrame for results
results_df = pd.DataFrame(results, columns=["Model", "Configuration", "Accuracy", "Precision", "Recall", "F1-Score"])
print("\nModel Performance Comparison:")
print(results_df)
```

```
Model Performance Comparison:
                 Model   Configuration  Accuracy  Precision    Recall  \
0  Logistic Regression           C=0.1  0.517188   0.438943  0.517188
1  Logistic Regression             C=1  0.534375   0.541627  0.534375
2  Logistic Regression            C=10  0.546875   0.543560  0.546875
3                  SVM   Kernel=linear  0.529687   0.534441  0.529687
4                  SVM      Kernel=rbf  0.548438   0.604497  0.548438
5        Random Forest  Estimators=100  0.540625   0.532537  0.540625
6        Random Forest  Estimators=200  0.550000   0.547028  0.550000

   F1-Score
0  0.461740
1  0.510526
2  0.540879
3  0.514486
4  0.516508
5  0.521039
6  0.530579
```

```
# Visualization: Bar Chart
plt.figure(figsize=(12, 6))
metrics = ["Accuracy", "Precision", "Recall", "F1-Score"]
colors = ["blue", "green", "red", "purple"]

for i, metric in enumerate(metrics):
    plt.barh(results_df["Model"] + " " + results_df["Configuration"], results_df[metric], color=colors[i], alpha=0.6, label=metric)

plt.xlabel("Score")
plt.title("Model Performance Comparison")
plt.legend()
plt.show()
```