```
import pandas as pd
import numpy as np
import re
from bs4 import BeautifulSoup
from sklearn.model_selection import train_test_split, KFold
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
import shap
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt

# Custom stop words (converted to list)
custom_stop_words = list(set([
    "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves",
    "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their",
    "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are",
    "was", "were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an",
    "the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about",
    "against", "between", "into", "through", "during", "before", "after", "above", "below", "to", "from", "up",
    "down", "in", "out", "on", "off", "over", "under", "again", "further", "then", "once", "here", "there", "when",
    "where", "why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some", "such", "no",
    "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will", "just", "don",
    "should", "now", "d", "ll", "m", "o", "re", "ve", "y", "ain", "aren", "couldn", "didn", "doesn", "hadn",
    "hasn", "haven", "isn", "ma", "mightn", "mustn", "needn", "shan", "shouldn", "wasn", "weren", "won",
    "wouldn", "ok", "nice", "fire", "best", "doesnt", "wont", "connected", "good", "okay", "hi"
]))

# Step 1: Load and preprocess your data
df = pd.read_csv('cleaned_dataset.csv', encoding='latin1')

# Preprocessing function
def clean_text(text):
    text = BeautifulSoup(text, "lxml").get_text()  # Use get_text() to avoid MarkupResemblesLocatorWarning
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return text

df['Base_Reviews'] = df['Base_Reviews'].astype(str).apply(clean_text)

# Encode the labels (categories) if they are not already numerical
label_encoder = LabelEncoder()
df['category'] = label_encoder.fit_transform(df['category'])

# Step 2: Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df['Base_Reviews'], df['category'], test_size=0.2,
random_state=123)

# Step 3: Vectorize the text data using TF-IDF with custom stop words
vectorizer = TfidfVectorizer(stop_words=custom_stop_words, ngram_range=(1, 1))
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Step 4: Implement K-Fold cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=1)

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train_tfidf)):
```

```
    print(f"Training fold {fold + 1}")

    # Split the training data into training and validation sets
    X_fold_train, X_fold_val = X_train_tfidf[train_idx], X_train_tfidf[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    # Train the MLP model with a higher max_iter
    mlp_model = MLPClassifier(random_state=1, max_iter=1000)
    mlp_model.fit(X_fold_train, y_fold_train)

    # Optionally evaluate on validation data (for demonstration purposes)
    score = mlp_model.score(X_fold_val, y_fold_val)
    print(f"Fold {fold + 1} accuracy: {score}")

# Step 5: Train the model on the entire training set after cross-validation
mlp_model.fit(X_train_tfidf, y_train)

# Define target labels
target_labels = ['feature', 'user_experience', 'other_information', 'issue']

# Iterate through each class and generate SHAP values and plots
for class_name in target_labels:
    class_label = label_encoder.transform([class_name])[0]  # Convert class name to label
    class_indices = np.where(y_test == class_label)[0]

    # Ensure class_indices is not empty
    if len(class_indices) == 0:
        print(f"No samples found for class {class_name} in the test set.")
        continue

    # Create the SHAP explainer with a representative background sample
    explainer = shap.KernelExplainer(mlp_model.predict_proba, X_train_tfidf[:100])  # Using a sample of 100 for
background
    shap_values = explainer.shap_values(X_test_tfidf[class_indices])

    # Get the SHAP values for the current class
    class_index = np.where(mlp_model.classes_ == class_label)[0][0]
    shap_values_class = shap_values[class_index]

    # Ensure the shap_values_class matches the feature matrix in shape by removing extra columns if necessary
    if shap_values_class.shape[1] != X_test_tfidf[class_indices].shape[1]:
        shap_values_class = shap_values_class[:, :X_test_tfidf[class_indices].shape[1]]  # Adjust column size

    feature_names = vectorizer.get_feature_names_out()

    # Get the top impacting words/features
    top_words = np.argsort(np.abs(shap_values_class).mean(0))[-10:]  # Get the top 10 words/features
    top_words_features = [feature_names[idx] for idx in top_words]

    print(f"Class: {class_name}")
    print(f"Top impacting words/features: {top_words_features}")

    # Fix: Convert X_test_tfidf[class_indices] to dense format before plotting
    plt.figure()
    shap.summary_plot(shap_values_class, X_test_tfidf[class_indices].toarray(), feature_names=feature_names)
```

```
# Add the SHAP value and impact text
text = f"SHAP value of {class_name} class | Impact on model output"
plt.text(0.5, -0.2, text, ha="center", fontsize=10, transform=plt.gca().transAxes)

# Save the plot as a PNG file
plt.savefig(f"{class_name}.png", dpi=300, bbox_inches="tight")

# Display the plot
plt.show()
```