```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns

# Load the dataset
data = pd.read_csv('sentiment reviews combined.csv')

# Step 1: Data Preprocessing
# Check for null values and remove them if any
data.dropna(inplace=True)

# Step 2: Exploratory Data Analysis
# Plot sentiment distribution
plt.figure(figsize=(6, 4))
sns.countplot(data['sentiment'], palette='viridis')
plt.title("Sentiment Distribution")
plt.xlabel("Sentiment")
plt.ylabel("Count")
plt.show()

# Generate Word Clouds for each sentiment
sentiments = data['sentiment'].unique()
for sentiment in sentiments:
    text = " ".join(data[data['sentiment'] == sentiment]['review'])
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)
    plt.figure(figsize=(8, 4))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.title(f"Word Cloud for {sentiment} Sentiment")
    plt.show()

# Step 3: Feature Engineering
# Split data into training and testing sets
X = data['review']
y = data['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert text to numerical data using TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Step 4: Hyperparameter Tuning with GridSearchCV
# Define the SVM model
svm_model = SVC(probability=True)

# Define hyperparameters grid
param_grid = {
    'C': [0.1, 1, 10],                # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'],  # Kernels to test
    'gamma': ['scale', 'auto'],       # Kernel coefficient for rbf/poly
    'degree': [2, 3, 4]               # Degree of polynomial kernel
}

# Perform GridSearchCV
```

```
grid_search = GridSearchCV(estimator=svm_model, param_grid=param_grid, scoring='accuracy',
cv=5, verbose=2)
grid_search.fit(X_train_tfidf, y_train)

# Best parameters and model
best_params = grid_search.best_params_
best_svm_model = grid_search.best_estimator_

print("Best Hyperparameters:", best_params)

# Step 5: Evaluate the Tuned Model
# Predict sentiments on test data using the tuned model
y_pred_tuned = best_svm_model.predict(X_test_tfidf)

# Evaluate the tuned model
print("Tuned SVM Accuracy:", accuracy_score(y_test, y_pred_tuned))
print("\nClassification Report:\n", classification_report(y_test, y_pred_tuned))

# Confusion Matrix for the tuned model
conf_matrix_tuned = confusion_matrix(y_test, y_pred_tuned)
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_tuned, annot=True, fmt='d', cmap='Blues', xticklabels=sentiments,
yticklabels=sentiments)
plt.title("Confusion Matrix - Tuned SVM")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Step 6: Make Predictions on New Data
def predict_sentiment(review):
    review_tfidf = vectorizer.transform([review])
    prediction = best_svm_model.predict(review_tfidf)
    return prediction[0]

# Example prediction
example_review = "This movie was absolutely fantastic!"
print(f"Sentiment Prediction: {predict_sentiment(example_review)}")
```