```
# Importamos el módulo TextBlob desde la librería textblob
# TextBlob es una herramienta en Python que nos permite realizar análisis de texto,
# incluyendo análisis de sentimientos, corrección ortográfica, traducción, entre otros.

from textblob import TextBlob
```

```
# Definimos una lista llamada 'frases', que contiene tres oraciones en inglés.
# Estas frases representan diferentes opiniones o comentarios, y pueden ser
# utilizadas para realizar análisis de sentimientos u otras operaciones de procesamiento de texto.

frases = [
    "The weather today is absolutely beautiful!",  # Una frase positiva sobre el clima
    "I am really disappointed with the new update.",  # Una frase negativa sobre una actualización
    "The event was well-organized but lacked excitement."  # Una frase neutral que mezcla una crítica y un elogio
]
```

```
# Definimos una lista llamada 'etiquetas_reales' que contiene valores que representan la polaridad
# esperada para cada frase: 1 es positivo, -1 es negativo, y 0 es neutral.
etiquetas_reales = [1, -1, 0]

# Creamos una lista vacía llamada 'predicciones' para almacenar las clasificaciones
# de las frases que haremos más adelante basándonos en su polaridad.
predicciones = []

# Iniciamos un ciclo for para iterar sobre cada frase en la lista 'frases'.
for frase in frases:
    # Para cada frase, creamos un objeto 'TextBlob' que permite analizar la polaridad del texto.
    blob = TextBlob(frase)

    # Mostramos la frase junto con su polaridad calculada.
    # La polaridad es un valor entre -1 (muy negativo) y 1 (muy positivo).
    print(f"Frase: '{frase}' | Sentimiento: {blob.sentiment.polarity}")

    # Clasificamos la frase según su polaridad:
    if blob.sentiment.polarity > 0:
        # Si la polaridad es mayor que 0, la frase se clasifica como positiva.
        predicciones.append(1)  # Positivo
    elif blob.sentiment.polarity < 0:
        # Si la polaridad es menor que 0, la frase se clasifica como negativa.
        predicciones.append(-1)  # Negativo
    else:
        # Si la polaridad es igual a 0, la frase se clasifica como neutral.
        predicciones.append(0)  # Neutral
```

```
Frase: 'The weather today is absolutely beautiful!' | Sentimiento: 1.0
Frase: 'I am really disappointed with the new update.' | Sentimiento: -0.3068181818181818
Frase: 'The event was well-organized but lacked excitement.' | Sentimiento: 0.0
```

```
# Importamos la función 'accuracy_score' del módulo 'metrics' de scikit-learn.
# Esta función nos permite calcular la precisión de un modelo comparando los valores
# predichos con los valores reales.

from sklearn.metrics import accuracy_score

# Calculamos la precisión del análisis de sentimientos comparando las etiquetas reales
# con las predicciones que obtuvimos usando TextBlob. La precisión es el número de predicciones correctas
# dividido por el total de predicciones, y está en un rango de 0 a 1.

precision = accuracy_score(etiquetas_reales, predicciones)

# Imprimimos el resultado de la precisión obtenida. Esto nos indica qué tan bien
# el análisis de sentimientos con TextBlob coincidió con las etiquetas reales.
print(f"Precisión del análisis de sentimientos con TextBlob: {precision}")
```

```
Precisión del análisis de sentimientos con TextBlob: 1.0
```

Actividad 2

```
# Importamos la clase 'TfidfVectorizer' del módulo 'feature extraction.text' de scikit-learn.
```

```
# Este vectorizador convierte una colección de documentos de texto en una matriz de características
# TF-IDF (Term Frequency - Inverse Document Frequency), que es una medida que refleja la importancia de una palabra
# en un documento en relación con un conjunto de documentos.

from sklearn.feature_extraction.text import TfidfVectorizer

# Importamos el clasificador 'MultinomialNB' del módulo 'naive_bayes' de scikit-learn.
# Multinomial Naive Bayes es un algoritmo de clasificación probabilística utilizado comúnmente en la clasificación
# de texto, donde las características son representadas por conteos de frecuencias de palabras o TF-IDF.

from sklearn.naive_bayes import MultinomialNB

# Importamos la función 'train_test_split' de scikit-learn, la cual nos permite dividir el conjunto de datos
# en subconjuntos de entrenamiento y prueba, para poder evaluar el rendimiento del modelo de clasificación.

from sklearn.model_selection import train_test_split

# Importamos tres métricas del módulo 'metrics' de scikit-learn:
# 1. 'accuracy_score': para medir la precisión del modelo (fracción de predicciones correctas).
# 2. 'confusion_matrix': para generar una matriz de confusión que muestra los aciertos y errores de cada clase.
# 3. 'classification_report': para obtener un reporte detallado con métricas como precisión, recall, y F1-score.

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Importamos matplotlib para crear gráficos y visualizaciones.
# En este caso, lo usaremos para visualizar la matriz de confusión.

import matplotlib.pyplot as plt

# Importamos seaborn, una librería para visualización de datos basada en matplotlib.
# Nos ayudará a crear gráficos más atractivos y fáciles de interpretar, como un mapa de calor (heatmap) para la matriz de confus

import seaborn as sns
```

```
# Definimos una lista llamada 'frases', que contiene cinco oraciones en inglés.
# Estas frases representan diferentes opiniones o comentarios que usaremos como
# conjunto de datos de ejemplo para la clasificación de texto.

frases = [
    "This app is very intuitive!",  # Una opinión positiva sobre una aplicación (clasificada como 1)
    "I didn't enjoy the customer service.",  # Una opinión negativa sobre el servicio al cliente (clasificada como 0)
    "Quite average experience.",  # Una opinión neutral o ligeramente negativa (clasificada como 0)
    "Fantastic presentation!",  # Una opinión muy positiva sobre una presentación (clasificada como 1)
    "The food was awful."  # Una opinión negativa sobre la comida (clasificada como 0)
]

# Definimos una lista llamada 'sentimientos', que contiene las etiquetas reales asociadas
# a cada frase: 1 para opiniones positivas y 0 para opiniones negativas o neutrales.

sentimientos = [1, 0, 0, 1, 0]  # Etiquetas: 1 (positivo) y 0 (negativo o neutral)
```

```
# Creamos una instancia de 'TfidfVectorizer', una herramienta que transforma texto en una representación numérica
# basada en la importancia relativa de cada palabra dentro del conjunto de datos.
# Cada palabra recibirá un peso basado en su frecuencia dentro de una frase y en comparación con las otras frases.

vectorizer = TfidfVectorizer()

# Usamos el vectorizador para ajustar y transformar las frases en una matriz TF-IDF.
# 'fit_transform' ajusta el vectorizador al conjunto de frases y luego las transforma en la matriz de características.
# El resultado, 'X', es una representación numérica de las frases, donde cada fila es una frase y cada columna es un término (p

X = vectorizer.fit_transform(frases)

# Dividimos los datos en conjuntos de entrenamiento y prueba utilizando 'train_test_split'.
# 'X_train' y 'y_train' son los datos y etiquetas de entrenamiento, mientras que 'X_test' y 'y_test' son los de prueba.
# Usamos el 80% de los datos para entrenamiento y el 20% para prueba. La semilla aleatoria (random_state=42) garantiza
# que los resultados sean reproducibles.

X_train, X_test, y_train, y_test = train_test_split(X, sentimientos, test_size=0.2, random_state=42)

# Creamos una instancia del clasificador Naive Bayes multinomial 'MultinomialNB',
# el cual es ideal para problemas de clasificación de texto donde las características son representaciones de frecuencia o TF-I
```

```
modelo = MultinomialNB()

# Entrenamos el modelo usando el conjunto de datos de entrenamiento ('X_train' y 'y_train').
# 'fit' ajusta el modelo a los datos, aprendiendo de las frases vectorizadas y sus etiquetas correspondientes.

modelo.fit(X_train, y_train)
```

```
MultinomialNB
MultinomialNB()
```

```
# Realizamos predicciones sobre el conjunto de prueba.
y_pred = modelo.predict(X_test)

# Calculamos la precisión comparando las etiquetas reales con las predicciones.
precision = accuracy_score(y_test, y_pred)

# Imprimimos la precisión del modelo.
print(f"Precisión del modelo: {precision}")
```

```
Precisión del modelo: 1.0
```

```
# Generamos un reporte de clasificación con métricas como precisión, recall y F1-score.
reporte_clasificacion = classification_report(y_test, y_pred)

# Imprimimos el reporte de clasificación.
print(f"Reporte de clasificación:\n{reporte_clasificacion}")
```

```
Reporte de clasificación:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1
```