Write a program that analyzes the sentiment of a piece of text by takeing a piece of text as input from the user, analyzes the sentiment of the text, and then displays whether the overall sentiment is positive, negative, or neutral

```
from textblob import TextBlob

def analyze_sentiment(text):
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    if polarity > 0:
        return "Positive"
    elif polarity < 0:
        return "Negative"
    else:
        return "Neutral"


text = input("Enter a piece of text: ")
sentiment = analyze_sentiment(text)
print(f"\nSentiment: {sentiment}")
```

```
Enter a piece of text: The movie was terrible, I hated it.

Sentiment: Negative
```

Part-of-Speech Tagging: Use a library like NLTK or spaCy to perform part-of-speech tagging on a sentence. Print out the words along with their corresponding parts of speech.

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

from nltk.tokenize import word_tokenize
from nltk import pos_tag

def pos_tagging(sentence):
    words = word_tokenize(sentence)
    tagged_words = pos_tag(words)
    return tagged_words


sentence = input("Enter a sentence: ")
tagged_words = pos_tagging(sentence)
print("\nPart-of-Speech Tagging:")
for word, pos in tagged_words:
  print(f"{word}: {pos}")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
Enter a sentence: The quick brown fox jumps over the lazy dog, demonstrating agility and speed as it traverses the terrain with ease

Part-of-Speech Tagging:
The: DT
quick: JJ
brown: NN
fox: NN
jumps: VBZ
over: IN
the: DT
lazy: JJ
dog: NN
,: ,
demonstrating: VBG
agility: NN
and: CC
speed: NN
as: IN
it: PRP
traverses: VBZ
the: DT
terrain: NN
with: IN
ease: NN
.: .
```

TF IDF EXAMPLE

```
from sklearn.feature_extraction.text import TfidfVectorizer

def calculate_tfidf(documents):
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_matrix = tfidf_vectorizer.fit_transform(documents)
    return tfidf_matrix, tfidf_vectorizer.get_feature_names_out()

documents = [
        "The cat sits on the mat",
        "The dog plays in the garden",
        "Birds chirp in the morning"
    ]
tfidf_matrix, feature_names = calculate_tfidf(documents)
print("\nTF-IDF Values:")
for i, doc in enumerate(documents):
  print(f"Document {i + 1}:")
  for j, word in enumerate(feature_names):
    print(f"{word}: {tfidf_matrix[i, j]}")
```

```
TF-IDF Values:
Document 1:
birds: 0.0
cat: 0.4305184979719882
chirp: 0.0
dog: 0.0
garden: 0.0
in: 0.0
mat: 0.4305184979719882
morning: 0.0
on: 0.4305184979719882
plays: 0.0
sits: 0.4305184979719882
the: 0.5085423203783267
Document 2:
birds: 0.0
cat: 0.0
chirp: 0.0
dog: 0.44839402160692654
garden: 0.44839402160692654
in: 0.3410152109911944
mat: 0.0
morning: 0.0
on: 0.0
plays: 0.44839402160692654
sits: 0.0
the: 0.5296574648148862
Document 3:
birds: 0.5046113401371842
cat: 0.0
chirp: 0.5046113401371842
dog: 0.0
garden: 0.0
in: 0.3837699307603192
mat: 0.0
morning: 0.5046113401371842
on: 0.0
plays: 0.0
sits: 0.0
the: 0.2980315863446099
```

Implement a text classification system using Support Vector Machines (SVM) and the 20 Newsgroups dataset. Preprocess the text data by tokenizing, removing stopwords, and converting words to their base forms using lemmatization. Use TF-IDF vectorization to convert text into numerical features. Train an SVM classifier with a linear kernel and evaluate its performance using accuracy and a classification report.

```
from sklearn.datasets import fetch_20newsgroups

# Load the 20 Newsgroups dataset
newsgroups_data = fetch_20newsgroups(subset='all', shuffle=True, remove=('headers', 'footers', 'quotes'))
X, y = newsgroups_data.data, newsgroups_data.target
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Download NLTK resources if not already downloaded
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

def preprocess_text(text):
    # Tokenize the text
    tokens = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
    # Lemmatize words
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
    # Join the tokens back into a single string
    preprocessed_text = ' '.join(lemmatized_tokens)
    return preprocessed_text

# Preprocess the text data
X_preprocessed = [preprocess_text(text) for text in X]

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.2, random_state=42)

# Convert text data into numerical features using TF-IDF vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Train an SVM classifier with a linear kernel
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train_tfidf, y_train)

# Make predictions
y_pred = svm_classifier.predict(X_test_tfidf)

# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=newsgroups_data.target_names))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
Accuracy: 0.6758620689655173
Classification Report:
                          precision    recall  f1-score   support

             alt.atheism       0.48      0.57      0.52       151
           comp.graphics       0.62      0.68      0.65       202
 comp.os.ms-windows.misc       0.65      0.63      0.64       195
comp.sys.ibm.pc.hardware       0.56      0.63      0.59       183
   comp.sys.mac.hardware       0.76      0.64      0.70       205
          comp.windows.x       0.80      0.71      0.75       215
            misc.forsale       0.74      0.70      0.72       193
               rec.autos       0.43      0.71      0.54       196
         rec.motorcycles       0.61      0.70      0.65       168
      rec.sport.baseball       0.82      0.77      0.79       211
        rec.sport.hockey       0.94      0.82      0.87       198
               sci.crypt       0.87      0.68      0.76       201
         sci.electronics       0.60      0.62      0.61       202
                 sci.med       0.80      0.79      0.80       194
               sci.space       0.69      0.74      0.71       189
  soc.religion.christian       0.73      0.74      0.74       202
      talk.politics.guns       0.72      0.71      0.71       188
   talk.politics.mideast       0.81      0.68      0.74       182
      talk.politics.misc       0.57      0.55      0.56       159
      talk.religion.misc       0.43      0.26      0.33       136

                accuracy                           0.68      3770
               macro avg       0.68      0.67      0.67      3770
            weighted avg       0.69      0.68      0.68      3770
```