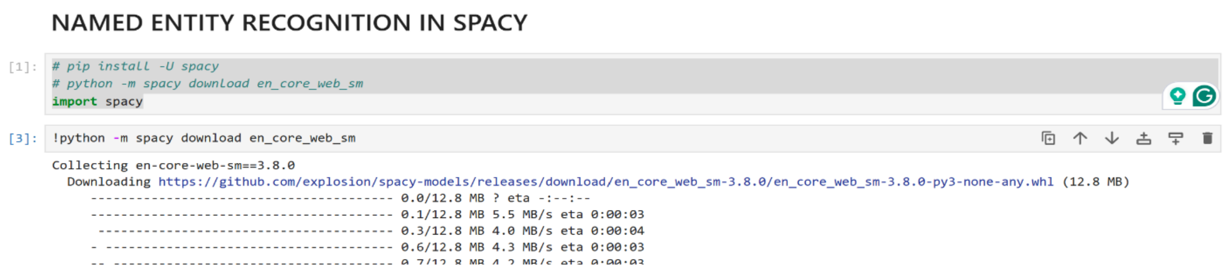
LAB03 REPORT: NATURAL LANGUAGE PROCESSING

NAME: RICHARD MENSAH

ID: 500699458

NAMED ENTITY RECOGNITION IN SPACY



NAMED ENTITY RECOGNITION IN SPACY

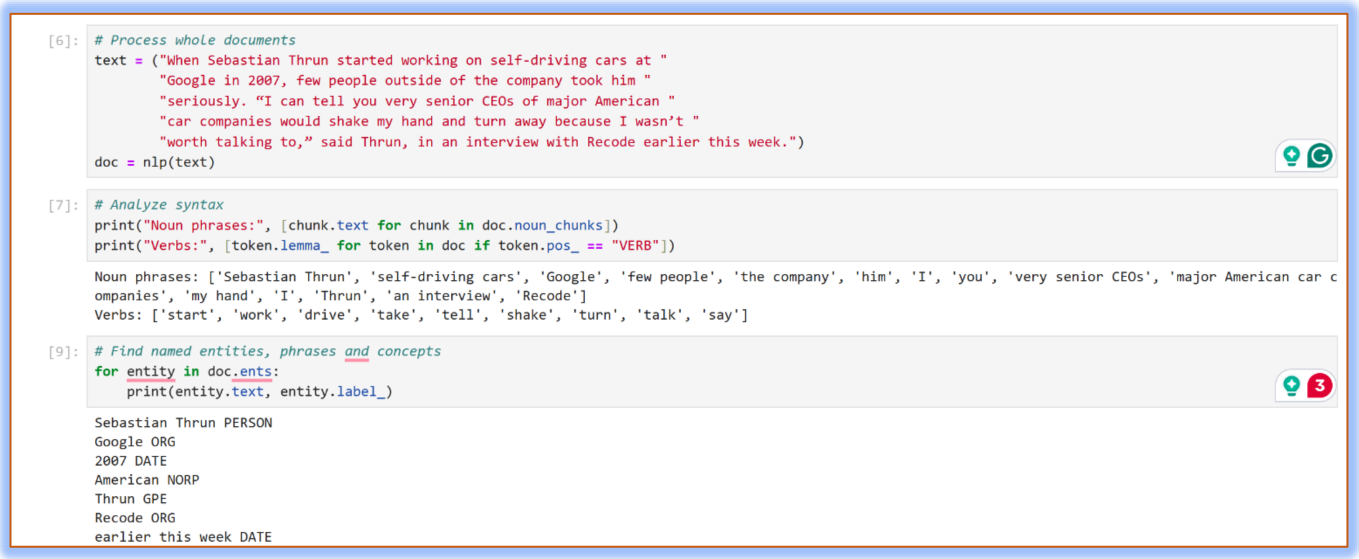
```
[1]: # pip install -U spacy
     # python -m spacy download en_core_web_sm
     import spacy
```

```
[3]: !python -m spacy download en_core_web_sm
```

```
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.whl (12.8 MB)
     ---------------------------------------- 0.0/12.8 MB ? eta -:--:--
     ---------------------------------------- 0.1/12.8 MB 5.5 MB/s eta 0:00:03
      --------------------------------------- 0.3/12.8 MB 4.0 MB/s eta 0:00:04
     - -------------------------------------- 0.6/12.8 MB 4.3 MB/s eta 0:00:03
     -- ------------------------------------- 0.7/12.8 MB 4.2 MB/s eta 0:00:03
```

The command, !python -m spacy download en_core_web_sm is used to download spaCy's small English language model, which includes the data needed for tasks like tokenization, part-of-speech tagging, and named entity recognition. Without this model, spaCy won't be able to analyze English text, as it relies on pre-trained linguistic patterns stored in these language models

# Load English tokenizer, tagger, parser, and NER
nlp = spacy.load("en_core_web_sm")



```
[6]: # Process whole documents
     text = ("When Sebastian Thrun started working on self-driving cars at "
             "Google in 2007, few people outside of the company took him "
             "seriously. "I can tell you very senior CEOs of major American "
             "car companies would shake my hand and turn away because I wasn't "
             "worth talking to," said Thrun, in an interview with Recode earlier this week.")
     doc = nlp(text)
```

```
[7]: # Analyze syntax
     print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
     print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])
```

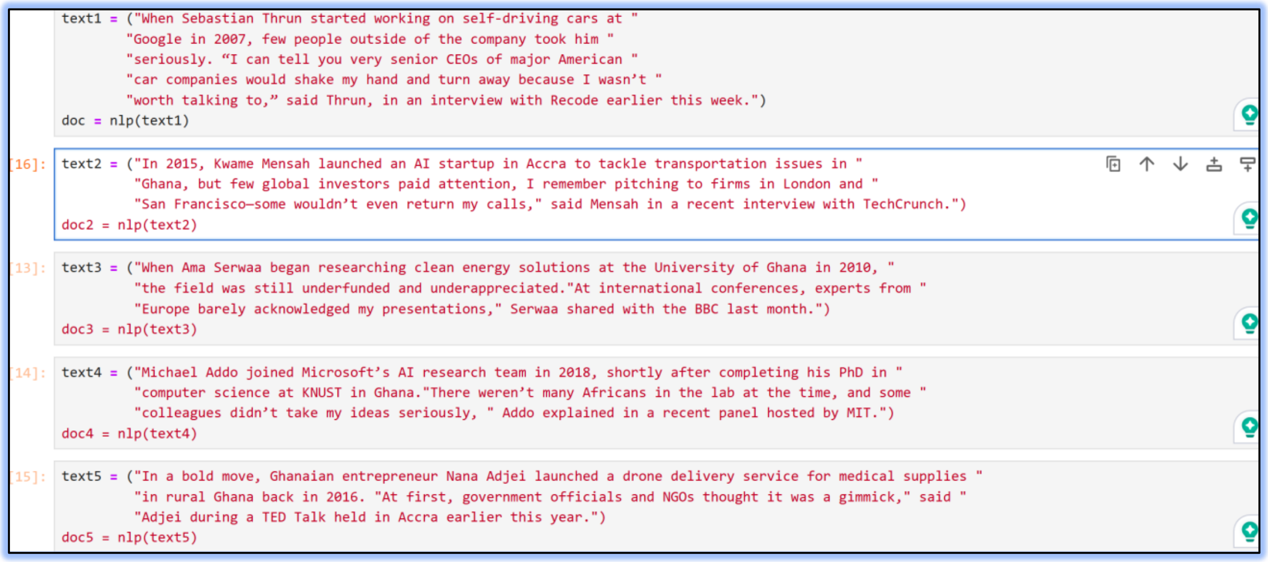
```
Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'the company', 'him', 'I', 'you', 'very senior CEOs', 'major American car c
ompanies', 'my hand', 'I', 'Thrun', 'an interview', 'Recode']
Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'talk', 'say']
```

```
[9]: # Find named entities, phrases and concepts
     for entity in doc.ents:
         print(entity.text, entity.label_)
```

```
Sebastian Thrun PERSON
Google ORG
2007 DATE
American NORP
Thrun GPE
Recode ORG
earlier this week DATE
```

spaCy is a fast and reliable tool for recognizing names, places, and dates in text. From the above code, it correctly picks out people like "Sebastian Thrun," companies like "Google," and times like "2007." It also identifies key phrases and the basic forms of verbs, giving a good overview of how the text is structured. While it's quite accurate for everyday language, it may not handle complex or specialized topics as well as newer AI models. Still, spaCy is a solid choice for many practical language tasks

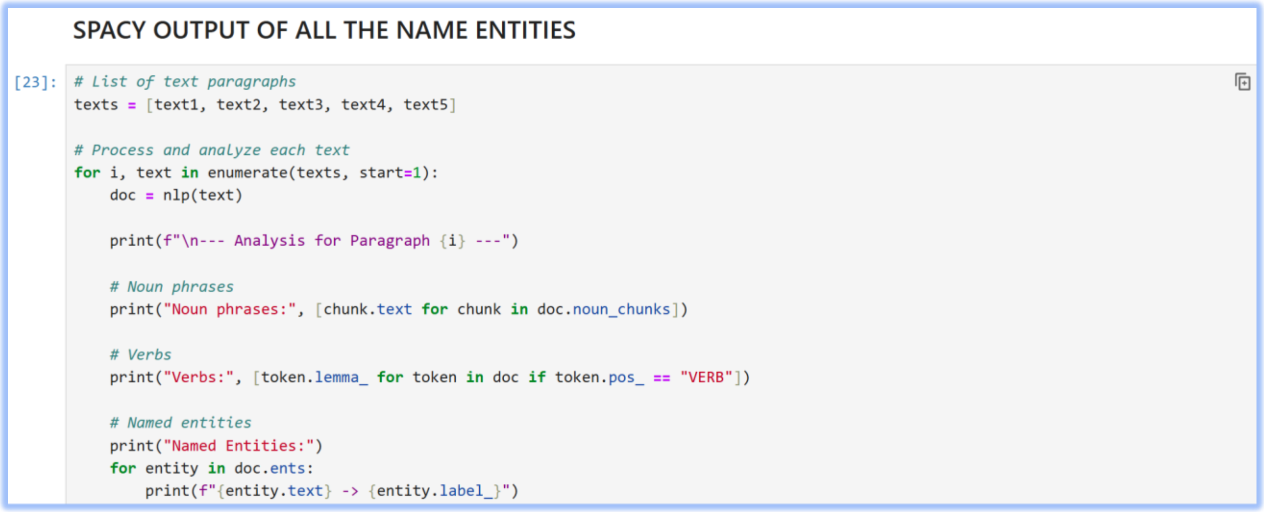
EXERCISE 3.1 NE CHUNKING IN SPACY AND NLTK



```
text1 = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. “I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasn’t "
        "worth talking to,” said Thrun, in an interview with Recode earlier this week.")
doc = nlp(text1)

[16]: text2 = ("In 2015, Kwame Mensah launched an AI startup in Accra to tackle transportation issues in "
         "Ghana, but few global investors paid attention, I remember pitching to firms in London and "
         "San Francisco—some wouldn’t even return my calls," said Mensah in a recent interview with TechCrunch.")
doc2 = nlp(text2)

[13]: text3 = ("When Ama Serwaa began researching clean energy solutions at the University of Ghana in 2010, "
         "the field was still underfunded and underappreciated."At international conferences, experts from "
         "Europe barely acknowledged my presentations," Serwaa shared with the BBC last month.")
doc3 = nlp(text3)

[14]: text4 = ("Michael Addo joined Microsoft’s AI research team in 2018, shortly after completing his PhD in "
         "computer science at KNUST in Ghana."There weren’t many Africans in the lab at the time, and some "
         "colleagues didn’t take my ideas seriously, " Addo explained in a recent panel hosted by MIT.")
doc4 = nlp(text4)

[15]: text5 = ("In a bold move, Ghanaian entrepreneur Nana Adjei launched a drone delivery service for medical supplies "
         "in rural Ghana back in 2016. "At first, government officials and NGOs thought it was a gimmick," said "
         "Adjei during a TED Talk held in Accra earlier this year.")
doc5 = nlp(text5)
```

The above picture is 5 text paragraphs that would be used to compare and contrast the effectiveness of the two Python libraries at identifying the named entities using both Spacy and NLTK.

3.1.1 Spacy Output of Name Entities



SPACY OUTPUT OF ALL THE NAME ENTITIES

```
[23]: # List of text paragraphs
texts = [text1, text2, text3, text4, text5]

# Process and analyze each text
for i, text in enumerate(texts, start=1):
    doc = nlp(text)

    print(f"\n--- Analysis for Paragraph {i} ---")

    # Noun phrases
    print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])

    # Verbs
    print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

    # Named entities
    print("Named Entities:")
    for entity in doc.ents:
        print(f"{entity.text} -> {entity.label_}")
```

3.1.1.1 Text 1

```
--- Analysis for Paragraph 1 ---
Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'the company', 'him', 'I', 'you', 'very senior CEOs', '
major American car companies', 'my hand', 'I', 'Thrun', 'an interview', 'Recode']
Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'talk', 'say']
Named Entities:
Sebastian Thrun -> PERSON
Google -> ORG
2007 -> DATE
American -> NORP
Thrun -> GPE
Recode -> ORG
earlier this week -> DATE
```

Text 2

```
--- Analysis for Paragraph 2 ---
Noun phrases: ['Kwame Mensah', 'an AI startup', 'Accra', 'transportation issues', 'Ghana', 'few global investors', 'attention', 'I', '
firms', 'London', 'San Francisco', 'some', 'my calls', 'Mensah', 'a recent interview', 'TechCrunch']
Verbs: ['launch', 'tackle', 'pay', 'remember', 'pitch', 'return', 'say']
Named Entities:
2015 -> DATE
Kwame Mensah -> PERSON
AI -> GPE
Accra -> ORG
Ghana -> GPE
London -> GPE
San Francisco -> GPE
Mensah -> PERSON
TechCrunch -> ORG
```

Text 3

```
--- Analysis for Paragraph 3 ---
Noun phrases: ['Ama Serwaa', 'clean energy solutions', 'the University', 'Ghana', 'the field', 'international conferences', 'experts'
, 'Europe', 'my presentations', 'Serwaa', 'the BBC']
Verbs: ['begin', 'research', 'acknowledge', 'share']
Named Entities:
Ama Serwaa -> PERSON
the University of Ghana -> ORG
2010 -> DATE
Europe -> LOC
BBC -> ORG
last month -> DATE
```

Text 4

```
--- Analysis for Paragraph 4 ---
Noun phrases: ['Michael Addo', 'Microsoft’s AI research team', 'his PhD', 'computer science', 'KNUST', 'Ghana', 'many Africa
ns', 'the lab', 'the time', 'some colleagues', 'my ideas', 'Addo', 'a recent panel', 'MIT']
Verbs: ['join', 'complete', 'be', 'take', 'explain', 'host']
Named Entities:
Michael Addo -> PERSON
Microsoft -> ORG
2018 -> DATE
PhD -> WORK_OF_ART
KNUST -> ORG
Ghana -> GPE
Africans -> NORP
MIT -> ORG
```

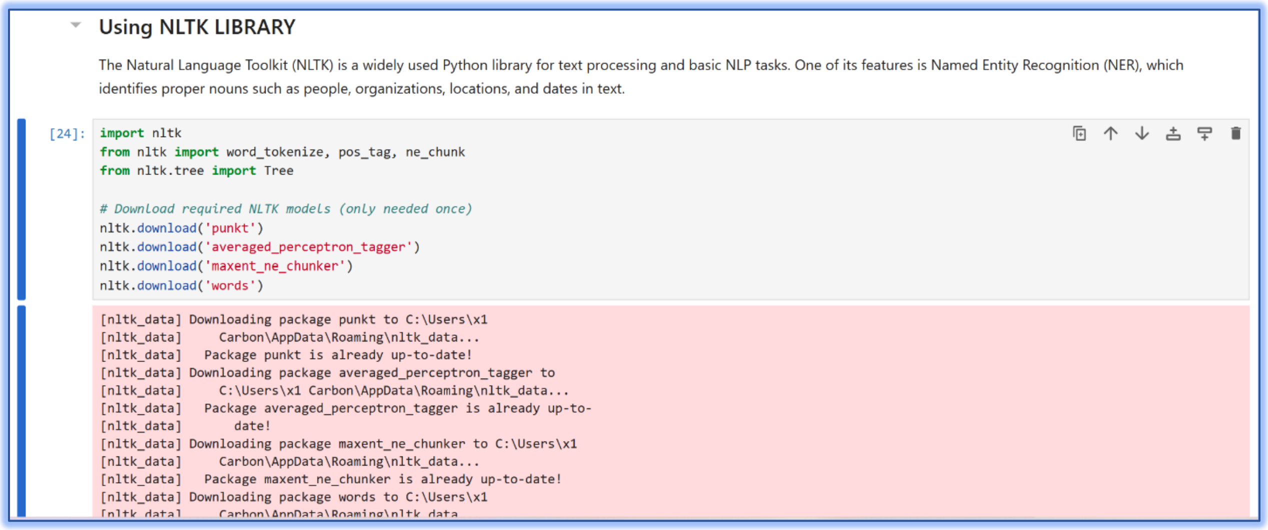
Text 5

```
--- Analysis for Paragraph 5 ---
Noun phrases: ['a bold move', 'Ghanaian entrepreneur', 'Nana Adjei', 'a drone delivery service', 'medical supplies', 'rural Ghana
', 'government officials', 'NGOs', 'it', 'a gimmick', 'Adjei', 'a TED Talk', 'Accra']
Verbs: ['launch', 'think', 'say', 'hold']
Named Entities:
Ghanaian -> NORP
Nana Adjei -> PERSON
Ghana -> GPE
2016 -> DATE
first -> ORDINAL
Adjei -> PRODUCT
Accra -> GPE
earlier this year -> DATE
```

3.1.2 Comparing spaCy and NLTK for Named Entity Recognition

3.1.2.1 Using NLTK Library

Here, I imported the nltk libraries and the required model to help extract my name entities.

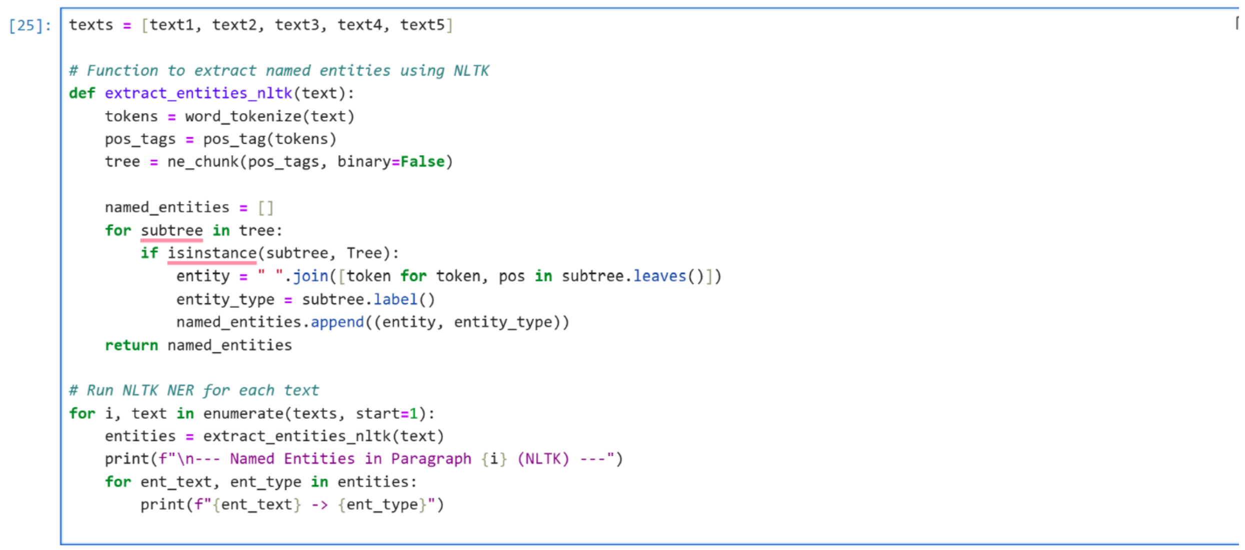


Using NLTK LIBRARY

The Natural Language Toolkit (NLTK) is a widely used Python library for text processing and basic NLP tasks. One of its features is Named Entity Recognition (NER), which identifies proper nouns such as people, organizations, locations, and dates in text.

[24]:
```python
import nltk
from nltk import word_tokenize, pos_tag, ne_chunk
from nltk.tree import Tree

# Download required NLTK models (only needed once)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package punkt to C:\Users\x1
[nltk_data]     Carbon\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\x1 Carbon\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to C:\Users\x1
[nltk_data]     Carbon\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to C:\Users\x1
[nltk_data]     Carbon\AppData\Roaming\nltk_data
```



[25]:
```python
texts = [text1, text2, text3, text4, text5]

# Function to extract named entities using NLTK
def extract_entities_nltk(text):
    tokens = word_tokenize(text)
    pos_tags = pos_tag(tokens)
    tree = ne_chunk(pos_tags, binary=False)

    named_entities = []
    for subtree in tree:
        if isinstance(subtree, Tree):
            entity = " ".join([token for token, pos in subtree.leaves()])
            entity_type = subtree.label()
            named_entities.append((entity, entity_type))
    return named_entities

# Run NLTK NER for each text
for i, text in enumerate(texts, start=1):
    entities = extract_entities_nltk(text)
    print(f"\n--- Named Entities in Paragraph {i} (NLTK) ---")
    for ent_text, ent_type in entities:
        print(f"{ent_text} -> {ent_type}")
```

NLTK Name Entity in Text1

```
--- Named Entities in Paragraph 1 (NLTK) ---
Sebastian Thrun -> PERSON
Google -> ORGANIZATION
CEOs -> ORGANIZATION
American -> GPE
Thrun -> PERSON
```

Text 2

```
--- Named Entities in Paragraph 2 (NLTK) ---
Kwame Mensah -> PERSON
Accra -> GPE
Ghana -> GPE
London -> GPE
San -> PERSON
Mensah -> PERSON
TechCrunch -> ORGANIZATION
```

Text 3

```
--- Named Entities in Paragraph 3 (NLTK) ---
Ama Serwaa -> PERSON
University -> ORGANIZATION
Ghana -> GPE
Serwaa -> PERSON
BBC -> ORGANIZATION
```

Text 4

```
--- Named Entities in Paragraph 4 (NLTK) ---
Michael -> PERSON
Addo -> PERSON
Microsoft -> PERSON
PhD -> ORGANIZATION
KNUST -> ORGANIZATION
```

Text 5

```
--- Named Entities in Paragraph 5 (NLTK) ---
Ghanaian -> GPE
Nana Adjei -> PERSON
NGOs -> ORGANIZATION
Adjei -> PERSON
TED Talk -> ORGANIZATION
```

[ ]:

Comparing sPacy and NLTK in Name Entity Extraction

Feature	spaCy	NLTK
NER Approach	Statistical model (machine learning-based)	Rule-based chunking (pattern-based with POS tags)
Entity Types	Fine-grained (e.g., PERSON, ORG, GPE, DATE, PRODUCT, etc.)	Basic types (PERSON, ORGANIZATION, GPE, etc.)

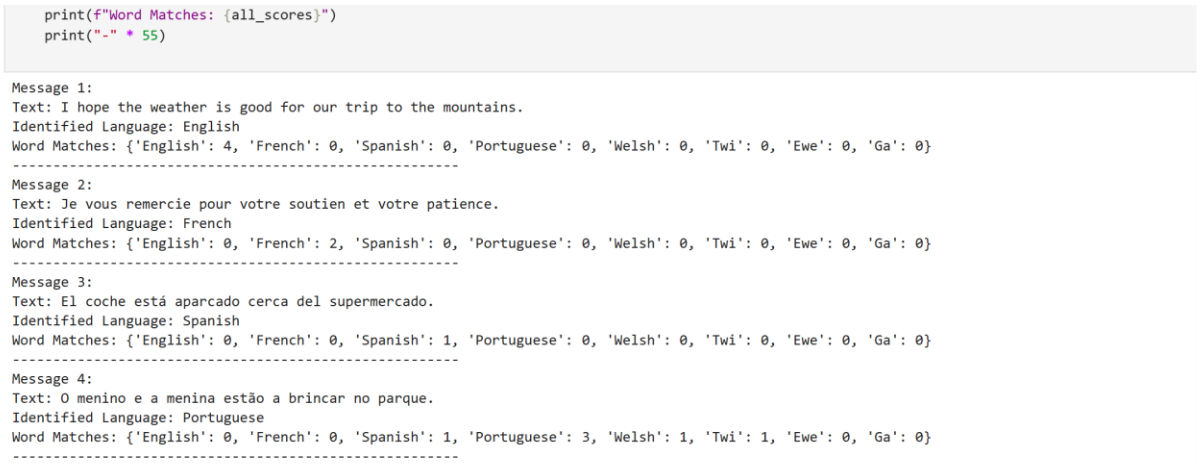
Accuracy	High, especially on modern language and named entities like startups, tech brands, etc.	Decent, but tends to miss newer or domain-specific terms
Ease of Use	Very user-friendly with intuitive API (doc.ents)	More manual – requires tokenization, POS tagging, chunking
Speed & Performance	Very fast and optimized for production use	Slower and better suited for educational/demo purposes
Language Support	Multilingual models available	English only (basic support for other languages)
NER Output in Test	Identified more entities (e.g., TechCrunch, TED Talk, KNUST, dates like 2018)	Missed or misclassified several new or uncommon names

Across all five sample paragraphs, spaCy proved to be more accurate, versatile, and easier to use than NLTK**.** While NLTK is a solid tool for learning the basics of NLP and quickly experimenting with text analysis, spaCy stands out as a more powerful and reliable option, especially when working with up-to-date language and tech-related topics, as well as a wide range of named entities. If you're looking for something suitable for real-world applications or production environments, spaCy is the better choice.

EXERCISE 3.2 – TOWER OF BABEL

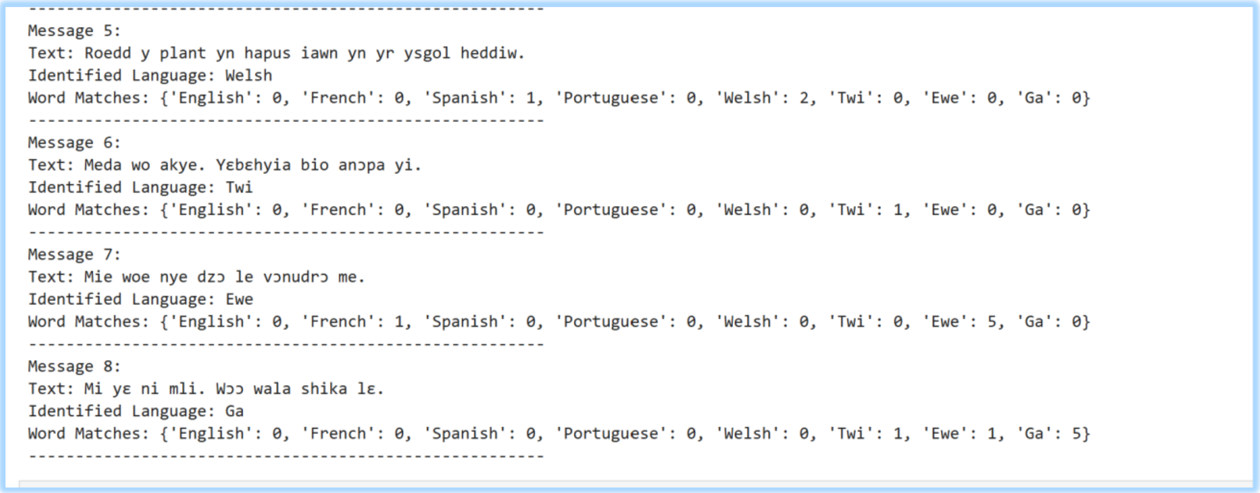
```python
# Language detection using common word matching
def detect_language(user_input):
    # Dictionary of basic vocabulary for each supported language
    language_keywords = {
        'English': ['the', 'and', 'of', 'to', 'in', 'that', 'is', 'it', 'for', 'on'],
        'French': ['le', 'la', 'et', 'de', 'un', 'vous', 'pas', 'les', 'en', 'ne'],
        'Spanish': ['el', 'la', 'que', 'y', 'de', 'no', 'es', 'por', 'un', 'en'],
        'Portuguese': ['o', 'a', 'e', 'de', 'não', 'que', 'em', 'um', 'por', 'é'],
        'Welsh': ['yn', 'y', 'a', 'oedd', 'ei', 'ar', 'am', 'fel', 'bod', 'gyda'],
        'Twi': ['wo', 'me', 'yɛ', 'de', 'ɛyɛ', 'sɛ', 'nko', 'na', 'yɛn', 'no'],
        'Ewe': ['nye', 'mie', 'woe', 'na', 'gbe', 'dzɔ', 'eɖe', 'le', 'mava', 'mi'],
        'Ga': ['mi', 'ni', 'ɔɔ', 'yɛ', 'ke', 'nɔ', 'mli', 'wala', 'shika', 'la']
    }

    # Make everything lowercase and split into words
    input_words = user_input.lower().split()

    # Count how many common words match each language
    match_counts = {}
    for lang, keywords in language_keywords.items():
        matches = sum(word in input_words for word in keywords)
        match_counts[lang] = matches

    # Choose the language with the most keyword matches
    best_guess = max(match_counts, key=match_counts.get)
    return best_guess, match_counts

# Some real messages in different languages to test detection
test_messages = [
    "I hope the weather is good for our trip to the mountains.",
    "Je vous remercie pour votre soutien et votre patience.",
    "El coche está aparcado cerca del supermercado.",
```

Output



```
    print(f"Word Matches: {all_scores}")
    print("-" * 55)
```

```
Message 1:
Text: I hope the weather is good for our trip to the mountains.
Identified Language: English
Word Matches: {'English': 4, 'French': 0, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
-------------------------------------------------------
Message 2:
Text: Je vous remercie pour votre soutien et votre patience.
Identified Language: French
Word Matches: {'English': 0, 'French': 2, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
-------------------------------------------------------
Message 3:
Text: El coche está aparcado cerca del supermercado.
Identified Language: Spanish
Word Matches: {'English': 0, 'French': 0, 'Spanish': 1, 'Portuguese': 0, 'Welsh': 0, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
-------------------------------------------------------
Message 4:
Text: O menino e a menina estão a brincar no parque.
Identified Language: Portuguese
Word Matches: {'English': 0, 'French': 0, 'Spanish': 1, 'Portuguese': 3, 'Welsh': 1, 'Twi': 1, 'Ewe': 0, 'Ga': 0}
-------------------------------------------------------
```



```
-------------------------------------------------------
Message 5:
Text: Roedd y plant yn hapus iawn yn yr ysgol heddiw.
Identified Language: Welsh
Word Matches: {'English': 0, 'French': 0, 'Spanish': 1, 'Portuguese': 0, 'Welsh': 2, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
-------------------------------------------------------
Message 6:
Text: Meda wo akye. Yɛbɛhyia bio anɔpa yi.
Identified Language: Twi
Word Matches: {'English': 0, 'French': 0, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 1, 'Ewe': 0, 'Ga': 0}
-------------------------------------------------------
Message 7:
Text: Mie woe nye dzɔ le vɔnudrɔ me.
Identified Language: Ewe
Word Matches: {'English': 0, 'French': 1, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 0, 'Ewe': 5, 'Ga': 0}
-------------------------------------------------------
Message 8:
Text: Mi yɛ ni mli. Wɔɔ wala shika lɛ.
Identified Language: Ga
Word Matches: {'English': 0, 'French': 0, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 1, 'Ewe': 1, 'Ga': 5}
-------------------------------------------------------
```

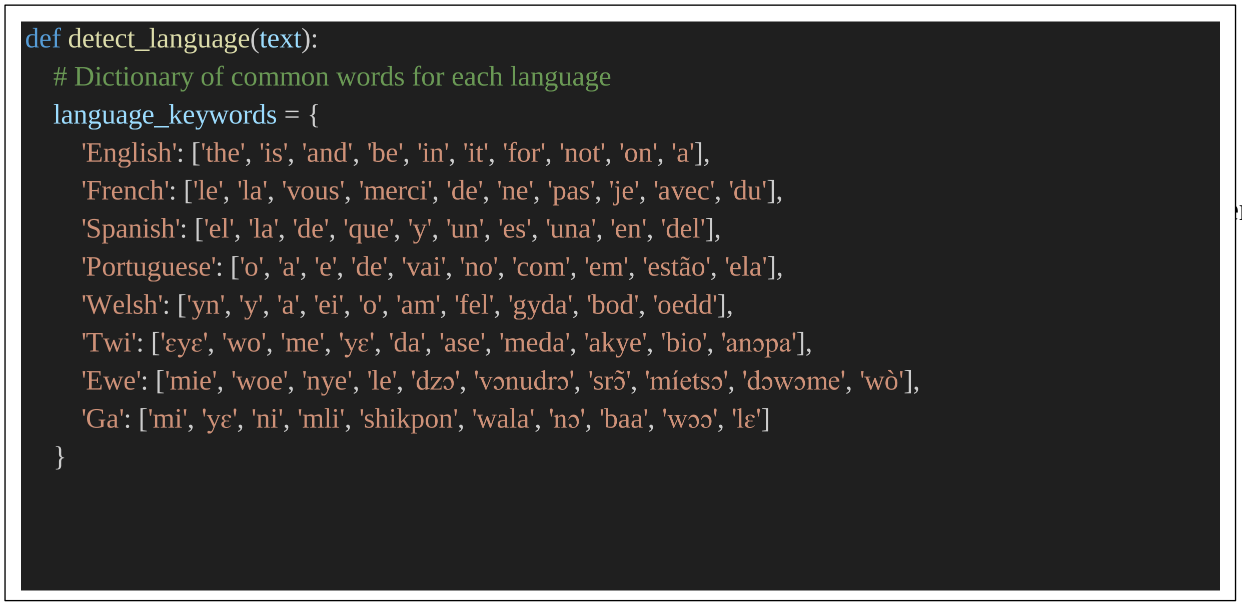
In the Tower of Babel text classification and language identification project, I developed a Python-based language identification tool using a rule-based approach. The function, named ***detect_language***, operates by comparing words in a given input message against predefined lists of frequently used words from eight different languages. These include English, French, Spanish, Portuguese, Welsh, and three Ghanaian languages: Twi, Ewe, and Ga. The script converts each message to lowercase and splits it into individual words using *user_input.lower().split().* Then, it counts how many of those words appear in each language's list of common words. The language with the most matches is chosen as the predicted language. The function also returns a dictionary showing how many words matched for each language, giving you a clearer picture of how the decision was made.

When I tested the function using eight different sentences, the English sentence was correctly identified due to the presence of four frequent English words, such as “the,” “is,” and “for.” The French message, although containing only two matching words, was correctly classified due to the uniqueness of words like ‘vous’ and ‘et’. Likewise, the Spanish sentence, which contained the word ‘el,’ was accurately identified. The Portuguese message had three matched words, which led to a confident prediction, despite some overlap with Spanish. The Welsh sentence had two key matches, which were sufficient for the script to correctly detect it as Welsh.

Also, the three Ghanaian languages were particularly well recognized. The Twi message was correctly identified with one distinctive match (‘meda’), and both Ewe and Ga sentences had five word matches each, making their classification highly accurate and confident. Withstanding, the accuracy of predictions was strongly influenced by how many distinctive and unique words were present in each message. This exercise demonstrated the usefulness of rule-based language identification, especially for underrepresented languages like Twi, Ewe, and Ga.

EXERCISE 3.3 – EVALUATING YOUR TOWER OF BABEL

To evaluate the effectiveness of my language identification program developed in the Tower of Babel exercise, I created a test set of 10 diverse sentences. Each sentence was written in one of the languages the system is designed to recognize: English, French, Spanish, Portuguese, Welsh, Twi, Ewe, and Ga. Some languages appear more than once to increase the challenge and variety.

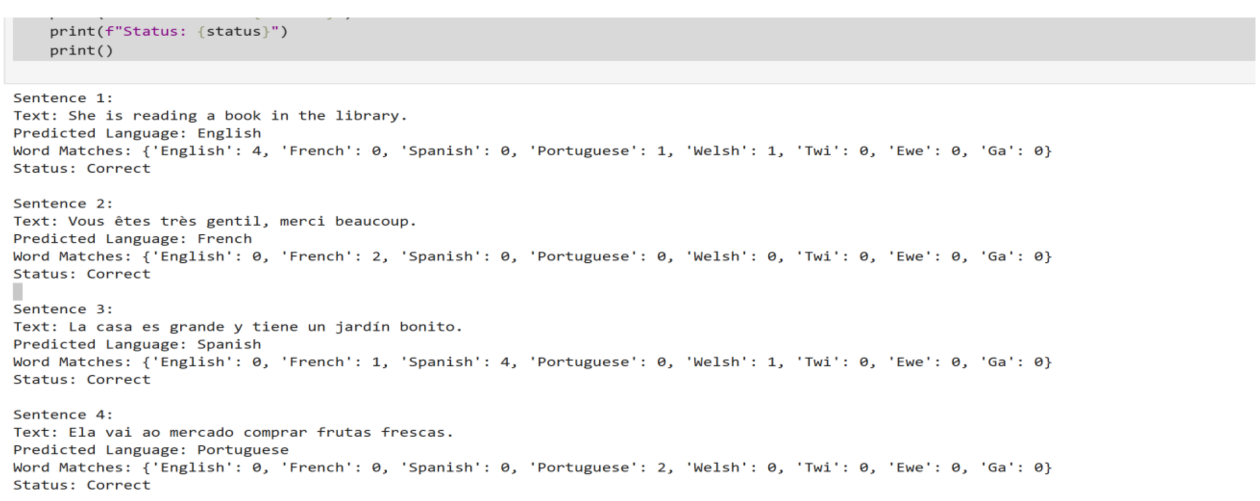


```
def detect_language(text):
    # Dictionary of common words for each language
    language_keywords = {
        'English': ['the', 'is', 'and', 'be', 'in', 'it', 'for', 'not', 'on', 'a'],
        'French': ['le', 'la', 'vous', 'merci', 'de', 'ne', 'pas', 'je', 'avec', 'du'],
        'Spanish': ['el', 'la', 'de', 'que', 'y', 'un', 'es', 'una', 'en', 'del'],
        'Portuguese': ['o', 'a', 'e', 'de', 'vai', 'no', 'com', 'em', 'estão', 'ela'],
        'Welsh': ['yn', 'y', 'a', 'ei', 'o', 'am', 'fel', 'gyda', 'bod', 'oedd'],
        'Twi': ['ɛyɛ', 'wo', 'me', 'yɛ', 'da', 'ase', 'meda', 'akye', 'bio', 'anɔpa'],
        'Ewe': ['mie', 'woe', 'nye', 'le', 'dzɔ', 'vɔnudrɔ', 'srɔ̃', 'míetsɔ', 'dɔwɔme', 'wò'],
        'Ga': ['mi', 'yɛ', 'ni', 'mli', 'shikpon', 'wala', 'nɔ', 'baa', 'wɔɔ', 'lɛ']
    }
```

m

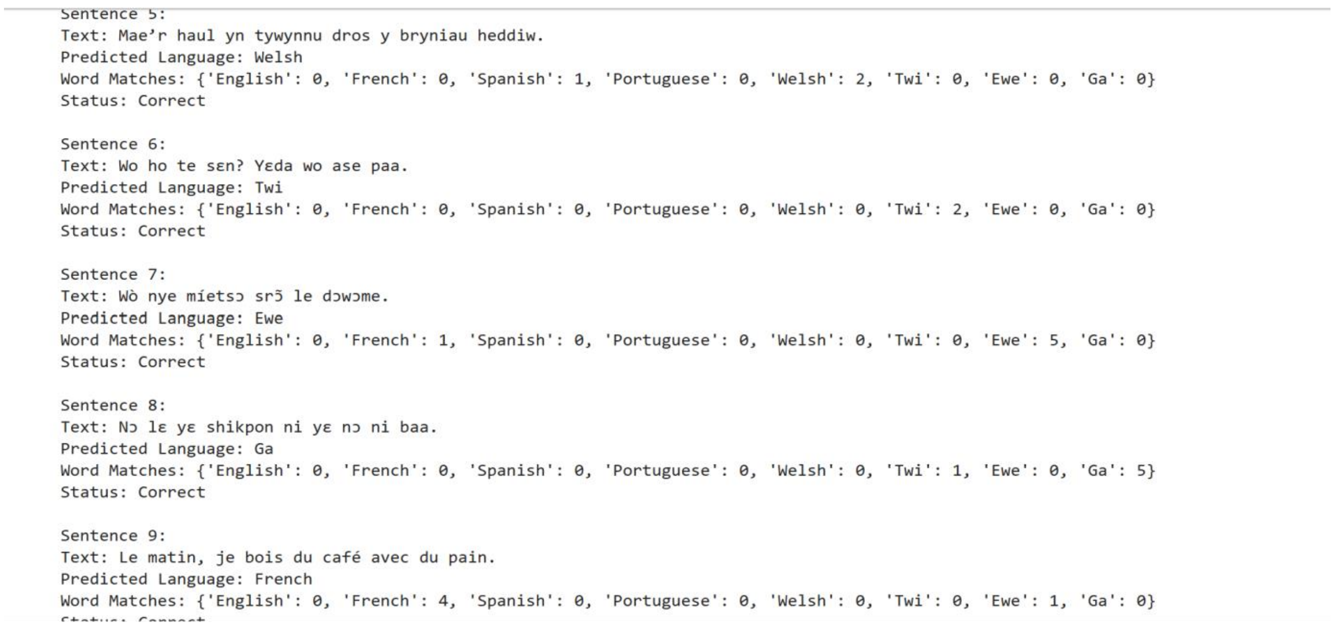
```
    words = text.lower().split()
    scores = {}

    # Count matches for each language
    for lang, keywords in language_keywords.items():
        score = sum(word in words for word in keywords)
        scores[lang] = score

    predicted_lang = max(scores, key=scores.get)
    return predicted_lang, scores

# Test sentences with expected labels
test_sentences = [
    ("She is reading a book in the library.", "English"),
    ("Vous êtes très gentil, merci beaucoup.", "French"),
    ("La casa es grande y tiene un jardín bonito.", "Spanish"),
    ("Ela vai ao mercado comprar frutas frescas.", "Portuguese"),
    ("Mae’r haul yn tywynnu dros y bryniau heddiw.", "Welsh"),
    ("Wo ho te sɛn? Yɛda wo ase paa.", "Twi"),
    ("Wò nye míetsɔ srɔ̃ le dɔwɔme.", "Ewe"),
    ("Nɔ lɛ yɛ shikpon ni yɛ nɔ ni baa.", "Ga"),
    ("Le matin, je bois du café avec du pain.", "French"),
    ("He does not like the noise in the city.", "English")
]

# Output results
for i, (text, expected_lang) in enumerate(test_sentences, 1):
    predicted, matches = detect_language(text)
    status = "Correct" if predicted == expected_lang else "Incorrect"

    print(f"Sentence {i}:")
    print(f"Text: {text}")
    print(f"Predicted Language: {predicted}")
    print(f"Word Matches: {matches}")
    print(f"Status: {status}")
    print()
```

Output



```
    print(f"Status: {status}")
    print()
```

```
Sentence 1:
Text: She is reading a book in the library.
Predicted Language: English
Word Matches: {'English': 4, 'French': 0, 'Spanish': 0, 'Portuguese': 1, 'Welsh': 1, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
Status: Correct

Sentence 2:
Text: Vous êtes très gentil, merci beaucoup.
Predicted Language: French
Word Matches: {'English': 0, 'French': 2, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
Status: Correct

Sentence 3:
Text: La casa es grande y tiene un jardín bonito.
Predicted Language: Spanish
Word Matches: {'English': 0, 'French': 1, 'Spanish': 4, 'Portuguese': 0, 'Welsh': 1, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
Status: Correct

Sentence 4:
Text: Ela vai ao mercado comprar frutas frescas.
Predicted Language: Portuguese
Word Matches: {'English': 0, 'French': 0, 'Spanish': 0, 'Portuguese': 2, 'Welsh': 0, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
Status: Correct
```



```
Sentence 5:
Text: Mae’r haul yn tywynnu dros y bryniau heddiw.
Predicted Language: Welsh
Word Matches: {'English': 0, 'French': 0, 'Spanish': 1, 'Portuguese': 0, 'Welsh': 2, 'Twi': 0, 'Ewe': 0, 'Ga': 0}
Status: Correct

Sentence 6:
Text: Wo ho te sɛn? Yɛda wo ase paa.
Predicted Language: Twi
Word Matches: {'English': 0, 'French': 0, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 2, 'Ewe': 0, 'Ga': 0}
Status: Correct

Sentence 7:
Text: Wò nye míetsɔ srɔ̃ le dɔwɔme.
Predicted Language: Ewe
Word Matches: {'English': 0, 'French': 1, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 0, 'Ewe': 5, 'Ga': 0}
Status: Correct

Sentence 8:
Text: Nɔ lɛ yɛ shikpon ni yɛ nɔ ni baa.
Predicted Language: Ga
Word Matches: {'English': 0, 'French': 0, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 1, 'Ewe': 0, 'Ga': 5}
Status: Correct

Sentence 9:
Text: Le matin, je bois du café avec du pain.
Predicted Language: French
Word Matches: {'English': 0, 'French': 4, 'Spanish': 0, 'Portuguese': 0, 'Welsh': 0, 'Twi': 0, 'Ewe': 1, 'Ga': 0}
Status: Correct
```

EXERCISE 3.4 – CONFUSION MATRIX

3.4.1 What is a Confusion Matrix?

For me, a confusion matrix is a really useful tool to understand how well a model is doing at classification tasks. It basically compares what the model predicted against the actual truth, so I can see not only where it got things right but also where it made mistakes. It breaks down the results into four categories, which helps me understand the model's strengths and weaknesses more clearly.

When I looked at the confusion matrix from my model, I saw the following:

Actual / Predicted	**Negative (0)**	**Positive (1)**
Negative (0)	24	91
Positive (1)	24	854

This is how I understood it:

- The first **24** means 24 reviews were actually negative, and my model correctly predicted them as negative. So those are true negatives.
- The **91** means that there were 91 negative reviews that my model mistakenly labeled as positive, false positives.
- The next **24** means 24 positive reviews were incorrectly predicted as negative, so those are false negatives.
- Finally, the **859** means the model correctly identified 859 positive reviews as positive, true positives.

I noticed that the second dataset gave me similar results, which tells me that my model does a good job at predicting sentiment overall, especially when the sentiment is positive. But it's still not perfect.

3.4.2 Importance and Application

From this, I can see my model is good at spotting positive reviews, as it yielded 859 of them. But it struggles a bit with negative reviews, only correctly catching 24 out of those, while confusing 91 negative reviews for positive ones.

EXERCISE 3.5 – MORE EVALUATION METRICS

After reviewing the confusion matrix earlier in Exercise 3.3, I wanted to dive deeper into how well my sentiment analysis model is performing beyond just the overall accuracy. To do this, I used the `classification_report` function from Scikit-learn, which gave me more detailed metrics including **precision**, **recall**, and **F1-score** for both the positive and negative sentiment classes.

Here's the simple line of code I used to generate the full classification report

```
from sklearn.metrics import classification_report

print(classification_report(df['Positive'], df['sentiment']))
```

3.5.1 Evaluation

The report helped me understand the model's strengths and weaknesses more clearly, especially when handling negative sentiment.

Metric	Negative (0)	Positive (1)	Meaning
Precision	0.50	0.90	Of the predicted labels, how many were correct?
Recall	0.21	0.97	Of all the actual labels, how many did the model catch?
F1-Score	0.29	0.94	The balance between precision and recall.
Accuracy	N/A	0.88	How often the model got it right.

When I looked at the precision, I saw that the model is quite good at correctly predicting positive reviews, it got 90% of them right. But for negative reviews, only 50% of the predictions were correct, meaning the model is not as confident or accurate in identifying negative feedback. The recall results made this even more obvious: the model managed to find 97% of all actual positive reviews**,** which is great. However, it only correctly identified 21% of the negative ones, which explains why I noticed earlier that a lot of negative reviews were being misclassified as positive. The F1-score, which balances both precision and recall, tells the same story. A score of 0.94 for positive sentiment shows strong performance. On the other hand, a 0.29 F1-score for negative sentiment highlights a major weakness: the model isn't doing well in detecting negative feedback.

EXERCISE 3.6 – YOUR OWN SENTIMENT ANALYSIS RESULTS

In this part of the exercise, I wanted to test the robustness of the sentiment analysis model using two new, randomly sampled subsets of the full Amazon review dataset. The goal was to see how consistent the model's performance would be across different samples, and how it compares to the larger evaluation of 20,000 reviews.

3.6 Random Sampling and Preprocessing

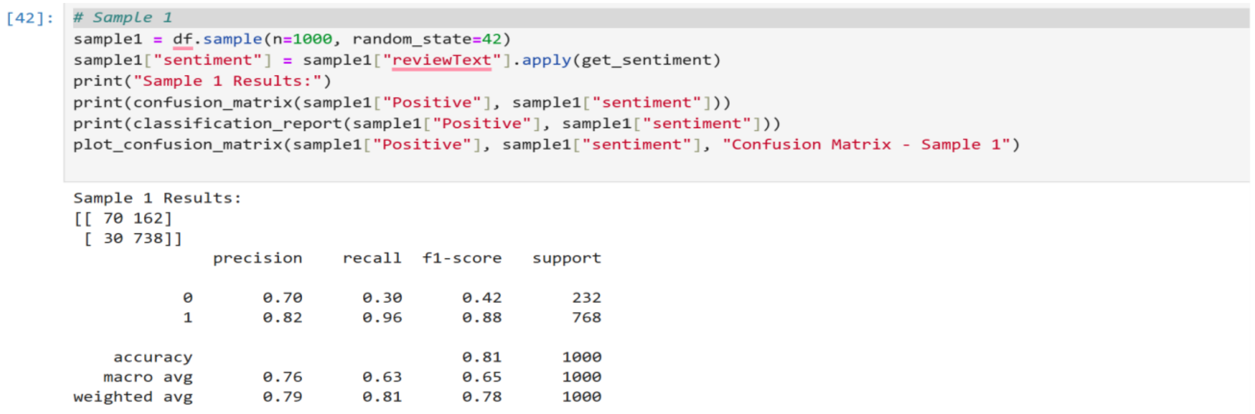
I began by taking **two random samples of 1,000 reviews each** from the complete Amazon dataset. To ensure a fair comparison, each sample was generated using a different random seed.

```python
import pandas as pd
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.metrics import confusion_matrix, classification_report

# Download required NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('vader_lexicon')

# Load full Amazon review dataset
df = pd.read_csv("https://raw.githubusercontent.com/pycaret/pycaret/master/datasets/amazon.csv")

# Preprocessing function
def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    filtered = [t for t in tokens if t not in stopwords.words("english")]
    lemmatizer = WordNetLemmatizer()
    lemmatized = [lemmatizer.lemmatize(token) for token in filtered]
    return " ".join(lemmatized)

# Apply preprocessing
df["reviewText"] = df["reviewText"].apply(preprocess_text)

# Initialize sentiment analyzer
analyzer = SentimentIntensityAnalyzer()

# Sentiment classification function
def get_sentiment(text):
    score = analyzer.polarity_scores(text)
    return 1 if score['pos'] > 0 else 0
```

Confusion Matrix for the Samples

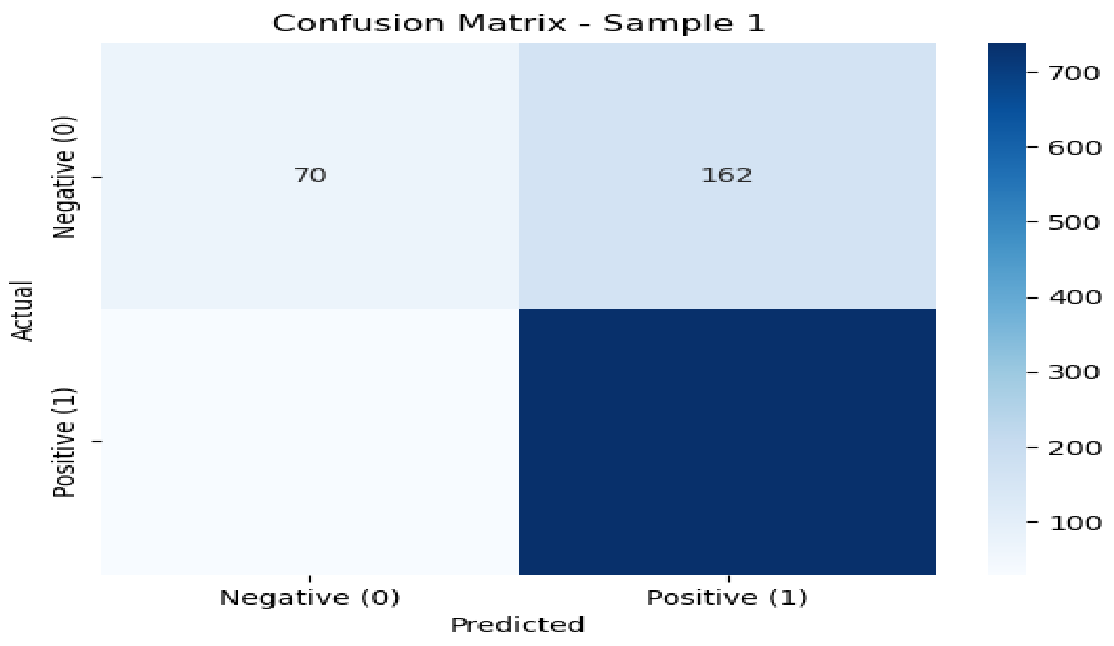
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    labels = ['Negative (0)', 'Positive (1)']
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels,
yticklabels=labels)
    plt.title(title)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

Sample One



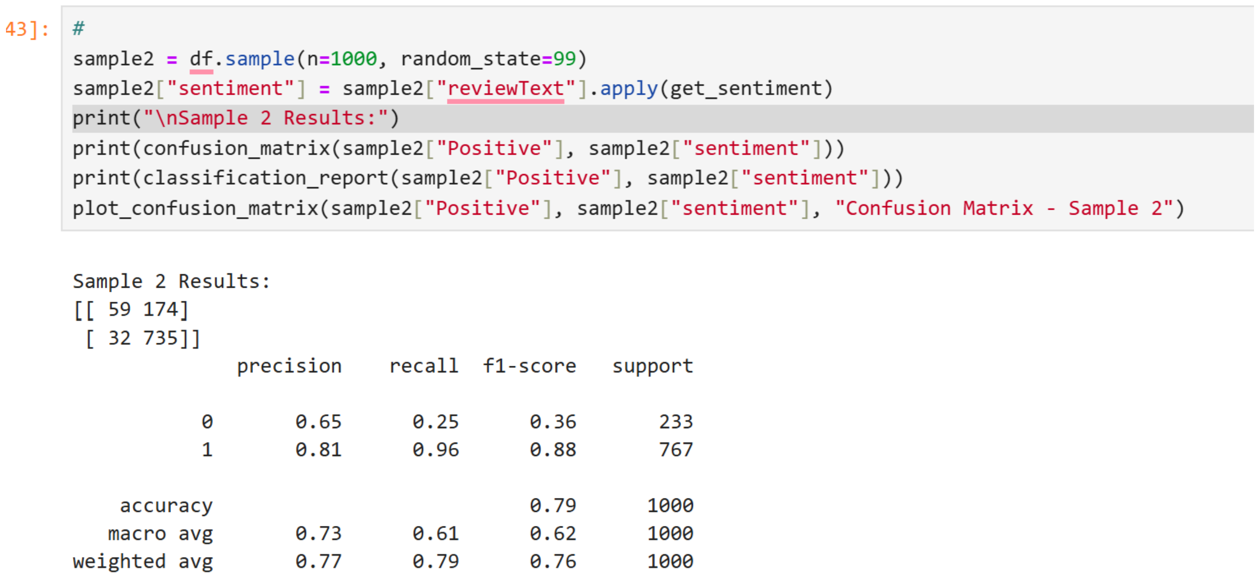
```
[42]: # Sample 1
      sample1 = df.sample(n=1000, random_state=42)
      sample1["sentiment"] = sample1["reviewText"].apply(get_sentiment)
      print("Sample 1 Results:")
      print(confusion_matrix(sample1["Positive"], sample1["sentiment"]))
      print(classification_report(sample1["Positive"], sample1["sentiment"]))
      plot_confusion_matrix(sample1["Positive"], sample1["sentiment"], "Confusion Matrix - Sample 1")
```

```
Sample 1 Results:
[[ 70 162]
 [ 30 738]]
              precision    recall  f1-score   support

           0       0.70      0.30      0.42       232
           1       0.82      0.96      0.88       768

    accuracy                           0.81      1000
   macro avg       0.76      0.63      0.65      1000
weighted avg       0.79      0.81      0.78      1000
```



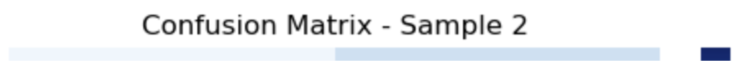
In Sample 1, the sentiment analysis model was tested on 1,000 Amazon reviews. It correctly identified 70 negative reviews but mistakenly labeled 162 negative reviews as positive. On the other hand, it successfully recognized 738 positive reviews, with only 30 positive reviews being misclassified as negative. This shows that while the model did an excellent job detecting positive reviews, it struggled to accurately identify negative ones. Out of 232 actual negative reviews, only about 30% were correctly flagged, meaning the majority were wrongly classified. Despite this weakness, the model achieved an overall accuracy of 81%, largely because most of the reviews were positive and it performed very well on those.

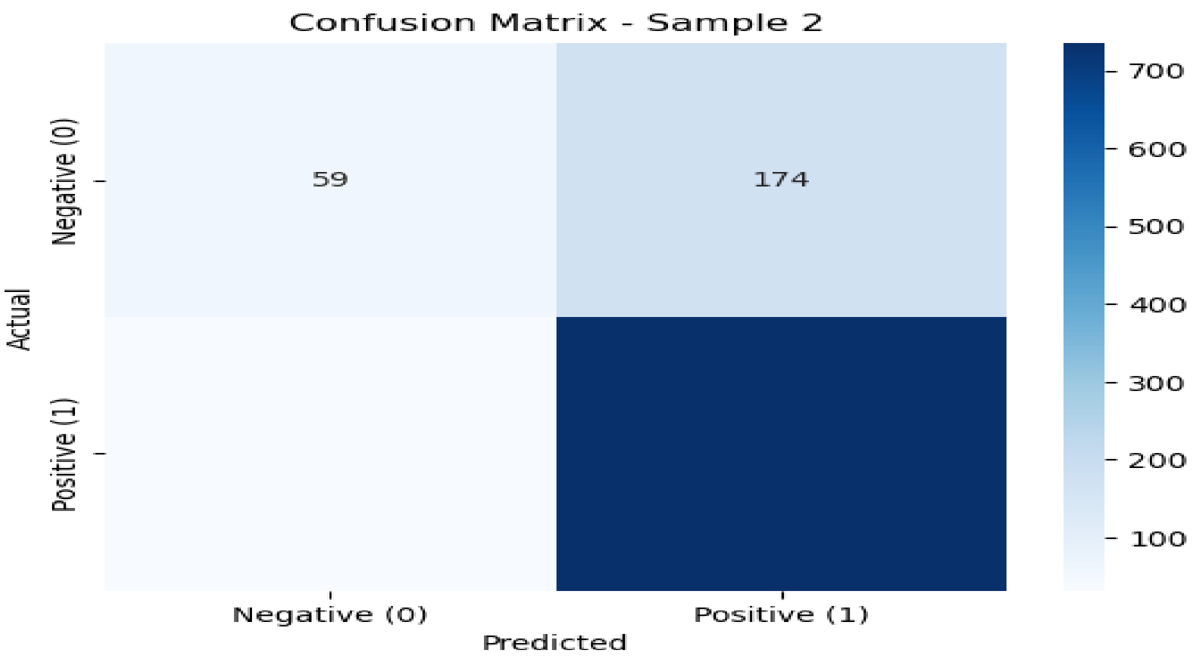
Sample 2



```
43]: #
     sample2 = df.sample(n=1000, random_state=99)
     sample2["sentiment"] = sample2["reviewText"].apply(get_sentiment)
     print("\nSample 2 Results:")
     print(confusion_matrix(sample2["Positive"], sample2["sentiment"]))
     print(classification_report(sample2["Positive"], sample2["sentiment"]))
     plot_confusion_matrix(sample2["Positive"], sample2["sentiment"], "Confusion Matrix - Sample 2")
```

```
Sample 2 Results:
[[ 59 174]
 [ 32 735]]
              precision    recall  f1-score   support

           0       0.65      0.25      0.36       233
           1       0.81      0.96      0.88       767

    accuracy                           0.79      1000
   macro avg       0.73      0.61      0.62      1000
weighted avg       0.77      0.79      0.76      1000
```





In Sample 2, the model correctly identified 59 negative reviews and 735 positive reviews. However, it mistakenly labeled 174 negative reviews as positive and missed 32 positive reviews by classifying them as negative. This shows the model is quite strong in detecting positive sentiments, achieving a high recall of 96% and a precision of 81% for positive reviews.

On the other hand, the model continues to struggle with recognizing negative reviews, capturing only 25% of them correctly. Despite this imbalance, the overall performance remains consistent, with an accuracy of 79%, similar to Sample 1. This indicates that while the model is reliable in flagging positive feedback, its ability to detect negative sentiment needs improvement.