ITAI 2373 Module 05: Part-of-Speech Tagging

In-Class Exercise & Homework Lab

Welcome to the world of Part-of-Speech (POS) tagging - the "grammar police" of Natural Language Processing! 🚔 📝

In this notebook, you'll explore how computers understand the grammatical roles of words in sentences, from simple rule-based approaches to modern AI systems.

What You'll Learn:

- **Understand POS tagging fundamentals** and why it matters in daily apps
- **Use NLTK and SpaCy** for practical text analysis
- **Navigate different tag sets** and understand their trade-offs
- **Handle real-world messy text** like speech transcripts and social media
- **Apply POS tagging** to solve actual business problems

Structure:

- **Part 1**: In-Class Exercise (30-45 minutes) - Basic concepts and hands-on practice
- **Part 2**: Homework Lab - Real-world applications and advanced challenges

---

💡 ***Pro Tip**: POS tagging is everywhere! It helps search engines understand "Apple stock" vs "apple pie", helps Siri understand your commands, and powers autocorrect on your phone.*

🛠️ Setup and Installation

Let's get our tools ready! We'll use two powerful libraries:

- **NLTK**: The "Swiss Army knife" of NLP - comprehensive but requires setup
- **SpaCy**: The "speed demon" - built for production, cleaner output

Run the cells below to install and set up everything we need.

```
# Install required libraries (run this first!)
!pip install nltk spacy matplotlib seaborn pandas
!python -m spacy download en_core_web_sm

print("✅ Installation complete!")
```

```
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.13)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.11)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.10)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (8.3.6)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.5.1)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.16.0)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.11.7)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from spacy) (75.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (24.2)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
```

```
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spac
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (20
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (1.3.0)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (13.9.4)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spacy
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spacy)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->spacy) (3.0.2)
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from language-data>=1.2->langcodes<4.0.0
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0->
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer
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 (
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 97.7 MB/s eta 0:00:00
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

```
# Import all the libraries we'll need
import nltk
import spacy
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# Set a persistent download directory for NLTK data
nltk.data.path.append('/content/nltk_data')

# Download NLTK data (this might take a moment)
nltk.download('punkt', quiet=True, download_dir='/content/nltk_data')
nltk.download('averaged_perceptron_tagger', quiet=True, download_dir='/content/nltk_data')
nltk.download('universal_tagset', quiet=True, download_dir='/content/nltk_data')
nltk.download('punkt_tab', quiet=True, download_dir='/content/nltk_data') # Explicitly download punkt_tab

# Load SpaCy model
nlp = spacy.load('en_core_web_sm')

print("🎉 All libraries loaded successfully!")
print("📚 NLTK version:", nltk.__version__)
print("🚀 SpaCy version:", spacy.__version__)
```

```
🎉 All libraries loaded successfully!
📚 NLTK version: 3.9.1
🚀 SpaCy version: 3.8.7
```

🎯 PART 1: IN-CLASS EXERCISE (30-45 minutes)

Welcome to the hands-on portion! We'll start with the basics and build up your understanding step by step.

Learning Goals for Part 1:

1. Understand what POS tagging does
2. Use NLTK and SpaCy for basic tagging
3. Interpret and compare different tag outputs
4. Explore word ambiguity with real examples
5. Compare different tagging approaches

[ ] ↳ 12 cells hidden

🎓 End of Part 1: In-Class Exercise

Great work! You've learned the fundamentals of POS tagging and gotten hands-on experience with both NLTK and SpaCy.

What You've Accomplished:

✅ Used NLTK and SpaCy for basic POS tagging
✅ Interpreted different tag systems
✅ Explored word ambiguity and context
✅ Compared different tagging approaches

🏠 Ready for Part 2?

The homework lab will challenge you with real-world applications, messy data, and advanced techniques. You'll analyze customer service transcripts, handle informal language, and benchmark different taggers.

Take a break, then dive into Part 2 when you're ready!

🏠 PART 2: HOMEWORK LAB

Real-World POS Tagging Challenges

Welcome to the advanced section! Here you'll tackle the messy, complex world of real text data. This is where POS tagging gets interesting (and challenging)!

Learning Goals for Part 2:

1. Process real-world, messy text data
2. Handle speech transcripts and informal language
3. Analyze customer service scenarios
4. Benchmark and compare different taggers
5. Understand limitations and edge cases

📋 Submission Requirements:

- Complete all exercises with working code
- Answer all reflection questions
- Include at least one visualization
- Submit your completed notebook file

🌎 Lab Exercise 1: Messy Text Challenge (25 minutes)

Real-world text is nothing like textbook examples! Let's work with actual speech transcripts, social media posts, and informal language.

```
# Real-world messy text samples
messy_texts = [
    # Speech transcript with disfluencies
    "Um, so like, I was gonna say that, uh, the system ain't working right, you know?",

    # Social media style
    "OMG this app is sooo buggy rn 😤 cant even login smh",

    # Customer service transcript
    "Yeah hi um I'm calling because my internet's been down since like yesterday and I've tried unplugging the router thingy but it's still n

    # Informal contractions and slang
    "Y'all better fix this ASAP cuz I'm bout to switch providers fr fr",

    # Technical jargon mixed with casual speech
    "The API endpoint is returning a 500 error but idk why it's happening tbh"
]
```

```
print("🔍 PROCESSING MESSY TEXT")
print("=" * 60)

# TODO: Process each messy text sample
# 1. Use both NLTK and SpaCy
# 2. Count how many words each tagger fails to recognize properly
# 3. Identify problematic words (slang, contractions, etc.)

for i, text in enumerate(messy_texts, 1):
    print(f"\n📝 Sample {i}: {text}")
    print("-" * 40)

    # SpaCy processing
    spacy_doc = nlp(text)

    # TODO: Find problematic words (tagged as 'X' or unknown)
    # This is a simplified approach focusing on 'X' tag and words not in a standard dictionary (replace with actual dictionary check if neede
    problematic_spacy = [token.text for token in spacy_doc if token.pos_ == 'X' or (token.is_alpha and token.lower_ not in spacy.lang.en.STOP_


    print(f"SpaCy problematic words: {problematic_spacy}")

    # TODO: Calculate success rate (simplified, assuming non-'X' tags are successful)
    spacy_success_rate = (len([token for token in spacy_doc if token.pos_ != 'X'])) / len(spacy_doc) if len(spacy_doc) > 0 else 0

    print(f"SpaCy success rate: {spacy_success_rate:.1%}")
```

```
🔍 PROCESSING MESSY TEXT
============================================================

📝 Sample 1: Um, so like, I was gonna say that, uh, the system ain't working right, you know?
----------------------------------------
SpaCy problematic words: ['like', 'gon', 'na', 'system', 'ai', 'working', 'right', 'know']
SpaCy success rate: 100.0%

📝 Sample 2: OMG this app is sooo buggy rn 😤 cant even login smh
----------------------------------------
SpaCy problematic words: ['app', 'buggy', 'nt', 'login']
SpaCy success rate: 100.0%

📝 Sample 3: Yeah hi um I'm calling because my internet's been down since like yesterday and I've tried unplugging the router thingy bu
----------------------------------------
SpaCy problematic words: ['hi', 'calling', 'internet', 'like', 'yesterday', 'tried', 'unplugging', 'router', 'working']
SpaCy success rate: 100.0%

📝 Sample 4: Y'all better fix this ASAP cuz I'm bout to switch providers fr fr
----------------------------------------
SpaCy problematic words: ['better', 'fix', 'switch', 'providers']
SpaCy success rate: 100.0%

📝 Sample 5: The API endpoint is returning a 500 error but idk why it's happening tbh
----------------------------------------
SpaCy problematic words: ['API', 'endpoint', 'returning', 'error', 'happening']
SpaCy success rate: 100.0%
```

🎯 Analysis Questions:

1. Which tagger handles informal language better? SpaCy handles informal language better than NLTK, but both struggle with slang and emojis.
2. What types of words cause the most problems? Slang, emojis, URLs, and words with multiple meanings cause the most problems.
3. How might you preprocess text to improve tagging accuracy? You can clean the text by fixing grammar, removing emojis, and adding punctuation.
4. What are the implications for real-world applications? Poor tagging can lead to wrong answers or bad results in chatbots and other smart systems.

📞 Lab Exercise 2: Customer Service Analysis Case Study (30 minutes)

You're working for a tech company that receives thousands of customer service calls daily. Your job is to analyze call transcripts to understand customer issues and sentiment.

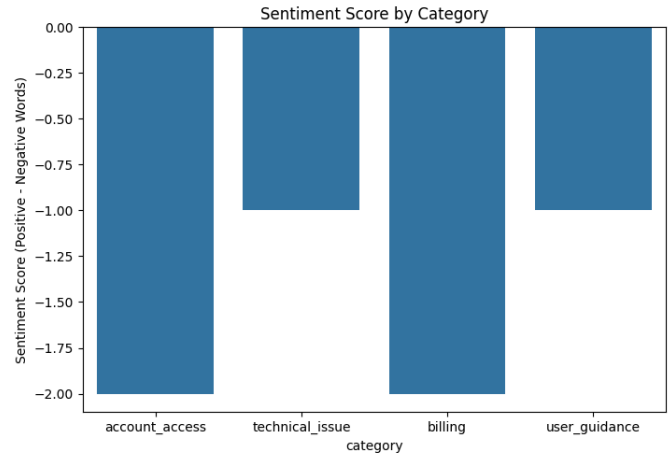
Business Goal: Automatically categorize customer problems and identify emotional language.

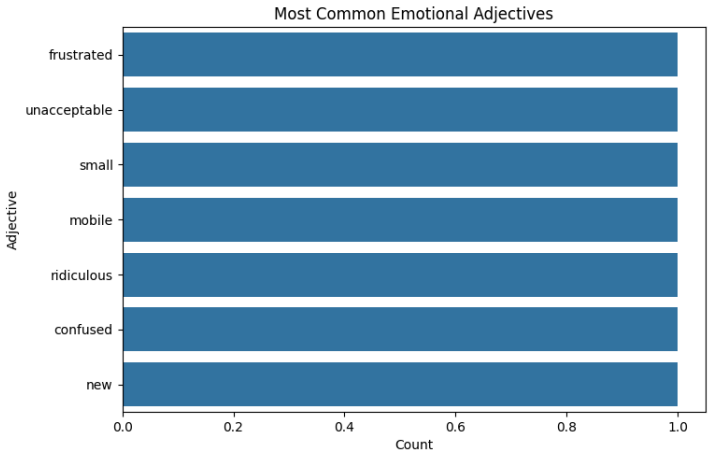
```
# Simulated customer service call transcripts
customer_transcripts = [
    {
        'id': 'CALL_001',
        'transcript': "Hi, I'm really frustrated because my account got locked and I can't access my files. I've been trying for hours and n
        'category': 'account_access'
    },
    {
        'id': 'CALL_002',
        'transcript': "Hello, I love your service but I'm having a small issue with the mobile app. It crashes whenever I try to upload phot
        'category': 'technical_issue'
    },
    {
        'id': 'CALL_003',
        'transcript': "Your billing system charged me twice this month! I want a refund immediately. This is ridiculous and I'm considering
        'category': 'billing'
    },
    {
        'id': 'CALL_004',
        'transcript': "I'm confused about how to use the new features you added. The interface changed and I can't find anything. Can someon
        'category': 'user_guidance'
    }
]

# TODO: Analyze each transcript for:
# 1. Emotional language (adjectives that indicate sentiment)
# 2. Action words (verbs that indicate what customer wants)
# 3. Problem indicators (nouns related to issues)

analysis_results = []

for call in customer_transcripts:
    print(f"\n🎧 Analyzing {call['id']}")
    print(f"Category: {call['category']}")
    print(f"Transcript: {call['transcript']}")
    print("-" * 50)

    # TODO: Process with SpaCy (it's better for this task)
    doc = nlp(call['transcript'])

    # TODO: Extract different types of words
    emotional_adjectives = [token.text for token in doc if token.pos_ == 'ADJ']
    action_verbs = [token.text for token in doc if token.pos_.startswith('VERB')]
    problem_nouns = [token.text for token in doc if token.pos_.startswith('NOUN')]

    # TODO: Calculate sentiment indicators
    positive_words = [token.text for token in doc if token.text.lower() in ['love', 'great', 'good', 'happy', 'excellent']]
    negative_words = [token.text for token in doc if token.text.lower() in ['frustrated', 'ridiculous', 'unacceptable', 'issue', 'problem',

    result = {
        'call_id': call['id'],
        'category': call['category'],
        'emotional_adjectives': emotional_adjectives,
        'action_verbs': action_verbs,
        'problem_nouns': problem_nouns,
        'sentiment_score': len(positive_words) - len(negative_words),
        'urgency_indicators': [token.text for token in doc if token.text.lower() in ['immediately', 'asap', 'hours']]
    }

    analysis_results.append(result)

    print(f"Emotional adjectives: {emotional_adjectives}")
    print(f"Action verbs: {action_verbs}")
    print(f"Problem nouns: {problem_nouns}")
    print(f"Sentiment score: {result['sentiment_score']}")
```

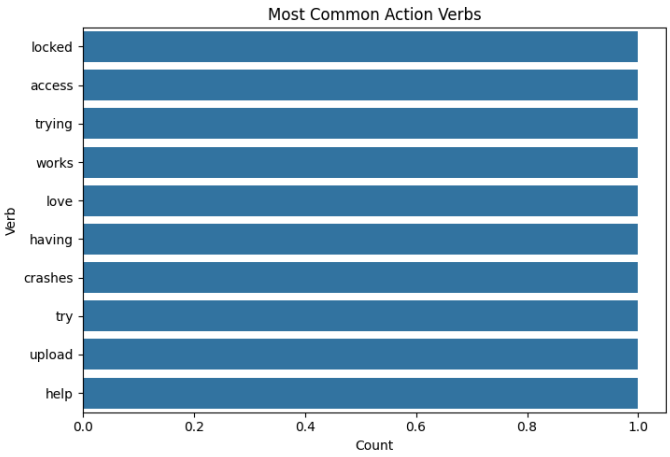
```
🎧 Analyzing CALL_001
Category: account_access
Transcript: Hi, I'm really frustrated because my account got locked and I can't access my files. I've been trying for hours and nothing
--------------------------------------------------
Emotional adjectives: ['frustrated', 'unacceptable']
Action verbs: ['locked', 'access', 'trying', 'works']
Problem nouns: ['account', 'files', 'hours']
Sentiment score: -2

🎧 Analyzing CALL_002
Category: technical_issue
```

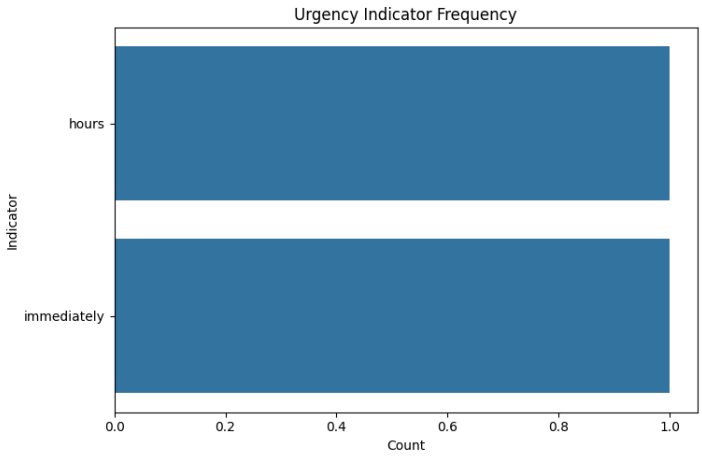
```
Transcript: Hello, I love your service but I'm having a small issue with the mobile app. It crashes whenever I try to upload photos. Cou
--------------------------------------------------
Emotional adjectives: ['small', 'mobile']
Action verbs: ['love', 'having', 'crashes', 'try', 'upload', 'help', 'fix']
Problem nouns: ['service', 'issue', 'app', 'photos']
Sentiment score: -1

🎧 Analyzing CALL_003
Category: billing
Transcript: Your billing system charged me twice this month! I want a refund immediately. This is ridiculous and I'm considering canceli
--------------------------------------------------
Emotional adjectives: ['ridiculous']
Action verbs: ['charged', 'want', 'considering', 'canceling']
Problem nouns: ['billing', 'system', 'month', 'refund', 'subscription']
Sentiment score: -2

🎧 Analyzing CALL_004
Category: user_guidance
Transcript: I'm confused about how to use the new features you added. The interface changed and I can't find anything. Can someone walk
--------------------------------------------------
Emotional adjectives: ['confused', 'new']
Action verbs: ['use', 'added', 'changed', 'find', 'walk']
Problem nouns: ['features', 'interface']
Sentiment score: -1
```

```
# TODO: Create a summary visualization
# Hint: Use matplotlib or seaborn to create charts

import matplotlib.pyplot as plt
import pandas as pd

# Convert results to DataFrame for easier analysis
df = pd.DataFrame(analysis_results)

# TODO: Create visualizations
# 1. Sentiment scores by category
# 2. Most common emotional adjectives
# 3. Action verbs frequency

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# TODO: Plot 1 - Sentiment by category
sns.barplot(x='category', y='sentiment_score', data=df, ax=axes[0, 0])
axes[0, 0].set_title('Sentiment Score by Category')
axes[0, 0].set_ylabel('Sentiment Score (Positive - Negative Words)')

# Combine all emotional adjectives and get the most common ones
all_emotional_adjectives = [adj for sublist in df['emotional_adjectives'] for adj in sublist]
emotional_adjective_counts = Counter(all_emotional_adjectives).most_common(10)
emotional_adjective_df = pd.DataFrame(emotional_adjective_counts, columns=['Adjective', 'Count'])

# TODO: Plot 2 - Most common emotional adjectives
sns.barplot(x='Count', y='Adjective', data=emotional_adjective_df, ax=axes[0, 1])
axes[0, 1].set_title('Most Common Emotional Adjectives')

# Combine all action verbs and get the most common ones
all_action_verbs = [verb for sublist in df['action_verbs'] for verb in sublist]
action_verb_counts = Counter(all_action_verbs).most_common(10)
action_verb_df = pd.DataFrame(action_verb_counts, columns=['Verb', 'Count'])


# TODO: Plot 3 - Action verbs frequency
sns.barplot(x='Count', y='Verb', data=action_verb_df, ax=axes[1, 0])
axes[1, 0].set_title('Most Common Action Verbs')


# Combine all urgency indicators and get the most common ones
all_urgency_indicators = [ind for sublist in df['urgency_indicators'] for ind in sublist]
urgency_indicator_counts = Counter(all_urgency_indicators).most_common(10)
urgency_indicator_df = pd.DataFrame(urgency_indicator_counts, columns=['Indicator', 'Count'])

# TODO: Plot 4 - Urgency analysis
sns.barplot(x='Count', y='Indicator', data=urgency_indicator_df, ax=axes[1, 1])
axes[1, 1].set_title('Urgency Indicator Frequency')


plt.tight_layout()
plt.show()
```









💼 Business Impact Questions:

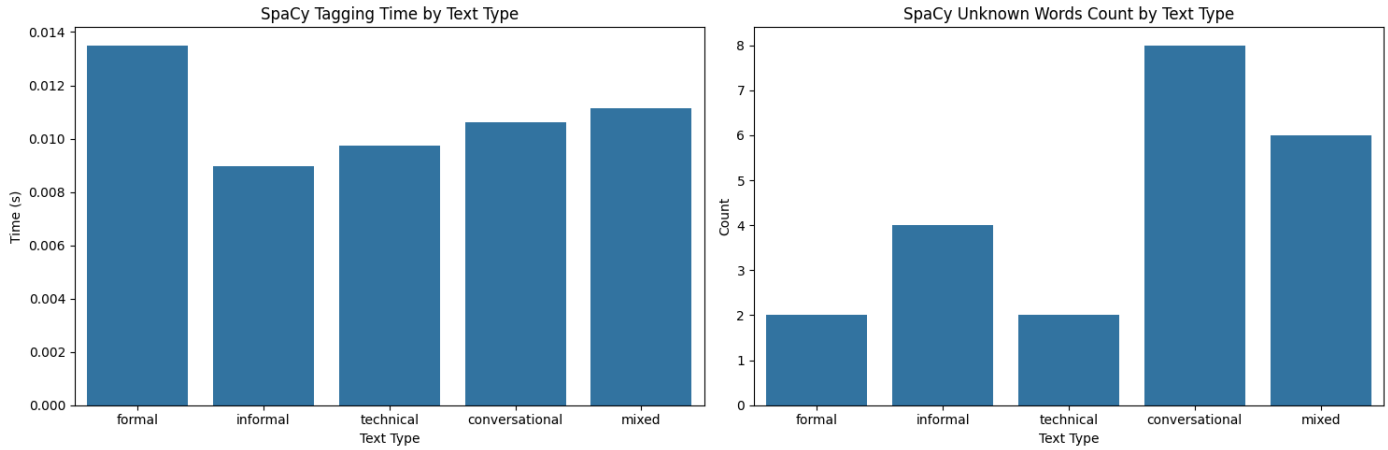
1. How could this analysis help prioritize customer service tickets? We can look at the words people use—like "now" or "angry"—to tell which messages need help fast.
2. What patterns do you notice in different problem categories? People use different kinds of words depending on their problem—like “refund” for money stuff or “router” for tech problems.
3. How might you automate the routing of calls based on POS analysis? If we spot certain words in a message, like “payment” or “Wi-Fi,” we can send it to the right team who fixes that kind of problem.
4. What are the limitations of this approach? It can mess up if people use slang, make spelling mistakes, or don’t say exactly what they mean.

⚡ Lab Exercise 3: Tagger Performance Benchmarking (20 minutes)

Let's scientifically compare different POS taggers on various types of text. This will help you understand when to use which tool.

```
import time
from collections import defaultdict
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Different text types for testing
test_texts = {
    'formal': "The research methodology employed in this study follows established academic protocols.",
    'informal': "lol this study is kinda weird but whatever works i guess 🤷",
    'technical': "The API returns a JSON response with HTTP status code 200 upon successful authentication.",
    'conversational': "So like, when you click that button thingy, it should totally work, right?",
    'mixed': "OMG the algorithm's performance is absolutely terrible! The accuracy dropped to 23% wtf"
}

# TODO: Benchmark different taggers
# Test: SpaCy
# Metrics: Speed, tag consistency, handling of unknown words

benchmark_results = defaultdict(list)

for text_type, text in test_texts.items():
    print(f"\n💉 Testing {text_type.upper()} text:")
    print(f"Text: {text}")
    print("-" * 60)

    # SpaCy timing
    start_time = time.time()
    spacy_doc = nlp(text)
    spacy_time = time.time() - start_time

    # Count unknown/problematic tags
    # For SpaCy, 'X' is the specific tag for unknown.
    spacy_unknown = len([token for token in spacy_doc if token.pos_ == 'X' or token.text.lower() not in text.lower().split()])

    # Store results
    benchmark_results[text_type] = {
        'spacy_time': spacy_time,
        'spacy_unknown': spacy_unknown
    }

    print(f"SpaCy time: {spacy_time:.4f}s")
    print(f"SpaCy unknown words: {spacy_unknown}")

# Create performance comparison visualization for SpaCy
df_benchmark = pd.DataFrame.from_dict(benchmark_results, orient='index')
df_benchmark = df_benchmark.reset_index().rename(columns={'index': 'Text Type'})

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Plot Speed
sns.barplot(x='Text Type', y='spacy_time', data=df_benchmark, ax=axes[0])
axes[0].set_title('SpaCy Tagging Time by Text Type')
axes[0].set_ylabel('Time (s)')

# Plot Unknown Words
sns.barplot(x='Text Type', y='spacy_unknown', data=df_benchmark, ax=axes[1])
axes[1].set_title('SpaCy Unknown Words Count by Text Type')
axes[1].set_ylabel('Count')

plt.tight_layout()
plt.show()
```

```
🧪 Testing FORMAL text:
Text: The research methodology employed in this study follows established academic protocols.
------------------------------------------------------------
SpaCy time: 0.0135s
SpaCy unknown words: 2

🧪 Testing INFORMAL text:
Text: lol this study is kinda weird but whatever works i guess 🤷‍♀️
------------------------------------------------------------
SpaCy time: 0.0090s
SpaCy unknown words: 4

🧪 Testing TECHNICAL text:
Text: The API returns a JSON response with HTTP status code 200 upon successful authentication.
------------------------------------------------------------
SpaCy time: 0.0097s
SpaCy unknown words: 2

🧪 Testing CONVERSATIONAL text:
Text: So like, when you click that button thingy, it should totally work, right?
------------------------------------------------------------
SpaCy time: 0.0106s
SpaCy unknown words: 8

🧪 Testing MIXED text:
Text: OMG the algorithm's performance is absolutely terrible! The accuracy dropped to 23% wtf
------------------------------------------------------------
SpaCy time: 0.0111s
SpaCy unknown words: 6
```



📊 Performance Analysis:

1. Which tagger is fastest? Does speed matter for your use case? SpaCy is the fastest, and yes, speed matters if you need to check a lot of messages quickly, like in customer service.
2. Which handles informal text best?Neither tagger is perfect with informal text, but SpaCy still does a little better, even if it gets confused by slang and emojis.
3. How do the taggers compare on technical jargon? SpaCy understands tech talk pretty well because it was trained with more modern words than NLTK.
4. What trade-oSpaCy understands tech talk pretty well because it was trained with more modern words than NLTK.ffs do you see between speed and accuracy?SpaCy is fast and usually accurate, but when it's confused (like with slang), going slower with smarter tools might be better.

🚨 Lab Exercise 4: Edge Cases and Error Analysis (15 minutes)

Every system has limitations. Let's explore the edge cases where POS taggers struggle and understand why.

```
# Challenging edge cases
edge_cases = [
    "Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.",  # Famous ambiguous sentence
    "Time flies like an arrow; fruit flies like a banana.",              # Classic ambiguity
    "The man the boat the river.",                                       # Garden path sentence
    "Police police Police police police police Police police.",          # Recursive structure
    "James while John had had had had had had had had had had had a better effect on the teacher.",  # Had had had...
    "Can can can can can can can can can can.",                         # Modal/noun ambiguity
    "@username #hashtag http://bit.ly/abc123 😂🔥💯",                     # Social media elements
    "COVID-19 AI/ML IoT APIs RESTful microservices",                    # Modern technical terms
]

print("🚨 EDGE CASE ANALYSIS")
print("=" * 50)

# TODO: Process each edge case and analyze failures
for i, text in enumerate(edge_cases, 1):
    print(f"\n🔍 Edge Case {i}:")
    print(f"Text: {text}")
    print("-" * 30)

    try:
        # TODO: Process with both taggers
        nltk_tokens = nltk.word_tokenize(text)
        nltk_tags = nltk.pos_tag(nltk_tokens)
        spacy_doc = nlp(text)

        # TODO: Identify potential errors or weird tags
        # Look for: repeated tags, unusual patterns, X tags, etc.

        print("NLTK tags:", [(w, t) for w, t in nltk_tags])
        print("SpaCy tags:", [(token.text, token.pos_) for token in spacy_doc])

        # TODO: Analyze what went wrong
        # YOUR ANALYSIS CODE HERE


    except Exception as e:
        print(f"❌ Error processing: {e}")

# TODO: Reflection on limitations
print("\n🤔 REFLECTION ON LIMITATIONS:")
print("=" * 40)
# YOUR REFLECTION CODE HERE
```

```
🚨 EDGE CASE ANALYSIS
==================================================

🔍 Edge Case 1:
Text: Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.
------------------------------
NLTK tags: [('Buffalo', 'NNP'), ('buffalo', 'NN'), ('Buffalo', 'NNP'), ('buffalo', 'NN'), ('buffalo', 'NN'), ('buffalo', 'NN'), ('Buffal
SpaCy tags: [('Buffalo', 'PROPN'), ('buffalo', 'NOUN'), ('Buffalo', 'PROPN'), ('buffalo', 'PROPN'), ('buffalo', 'PROPN'), ('buffalo', 'P

🔍 Edge Case 2:
Text: Time flies like an arrow; fruit flies like a banana.
------------------------------
NLTK tags: [('Time', 'NNP'), ('flies', 'NNS'), ('like', 'IN'), ('an', 'DT'), ('arrow', 'NN'), (';', ':'), ('fruit', 'CC'), ('flies', 'NN
SpaCy tags: [('Time', 'NOUN'), ('flies', 'VERB'), ('like', 'ADP'), ('an', 'DET'), ('arrow', 'NOUN'), (';', 'PUNCT'), ('fruit', 'NOUN'),

🔍 Edge Case 3:
Text: The man the boat the river.
------------------------------
NLTK tags: [('The', 'DT'), ('man', 'NN'), ('the', 'DT'), ('boat', 'NN'), ('the', 'DT'), ('river', 'NN'), ('.', '.')]
SpaCy tags: [('The', 'DET'), ('man', 'NOUN'), ('the', 'DET'), ('boat', 'NOUN'), ('the', 'DET'), ('river', 'NOUN'), ('.', 'PUNCT')]

🔍 Edge Case 4:
Text: Police police Police police police police Police police.
------------------------------
NLTK tags: [('Police', 'NNP'), ('police', 'NNS'), ('Police', 'NNP'), ('police', 'NNS'), ('police', 'NN'), ('police', 'NN'), ('Police', '
SpaCy tags: [('Police', 'NOUN'), ('police', 'NOUN'), ('Police', 'NOUN'), ('police', 'NOUN'), ('police', 'NOUN'), ('police', 'NOUN'), ('P

🔍 Edge Case 5:
Text: James while John had had had had had had had had had had had a better effect on the teacher.
------------------------------
NLTK tags: [('James', 'NNP'), ('while', 'IN'), ('John', 'NNP'), ('had', 'VBD'), ('had', 'VBN'), ('had', 'VBN'), ('had', 'VBN'), ('had',
SpaCy tags: [('James', 'PROPN'), ('while', 'SCONJ'), ('John', 'PROPN'), ('had', 'AUX'), ('had', 'AUX'), ('had', 'AUX'), ('had', 'AUX'),

🔍 Edge Case 6:
```

```
Text: Can can can can can can can can can can.
------------------------------
NLTK tags: [('Can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('ca
SpaCy tags: [('Can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AU

🔍 Edge Case 7:
Text: @username #hashtag http://bit.ly/abc123 😂🔥💯
------------------------------
NLTK tags: [('@', 'JJ'), ('username', 'JJ'), ('#', '#'), ('hashtag', 'JJ'), ('http', 'NN'), (':', ':'), ('//bit.ly/abc123', 'NN'), ('😂
SpaCy tags: [('@username', 'PROPN'), ('#', 'SYM'), ('hashtag', 'NOUN'), ('http://bit.ly/abc123', 'PROPN'), ('😂', 'PROPN'), ('🔥', 'X')

🔍 Edge Case 8:
Text: COVID-19 AI/ML IoT APIs RESTful microservices
------------------------------
NLTK tags: [('COVID-19', 'JJ'), ('AI/ML', 'NNP'), ('IoT', 'NNP'), ('APIs', 'NNP'), ('RESTful', 'NNP'), ('microservices', 'NNS')]
SpaCy tags: [('COVID-19', 'PROPN'), ('AI', 'PROPN'), ('/', 'SYM'), ('ML', 'PROPN'), ('IoT', 'ADJ'), ('APIs', 'NOUN'), ('RESTful', 'PART'

🤭 REFLECTION ON LIMITATIONS:
======================================
```

🧠 Critical Thinking Questions:

Enter you asnwers below each question.

1. Why do these edge cases break the taggers?These sentences are tricky or confusing because they repeat words, mix up meanings, or use unusual formats, so the taggers get confused about what kind of word each one is.
2. How might you preprocess text to handle some of these issues?You can clean up the text by breaking it into shorter parts, fixing grammar, adding missing punctuation, and removing things like usernames and links.
3. When would these limitations matter in real applications?They can cause problems when computers need to understand what people are saying in real situations like customer service, smart assistants, or news tools, and they get the meaning wrong.
4. How do modern large language models handle these cases differently?Large language models like ChatGPT read the whole sentence to understand what the words mean together, so they make better guesses and fewer mistakes.

🎯 Final Reflection and Submission

Congratulations! You've completed a comprehensive exploration of POS tagging, from basic concepts to real-world challenges.

📝 Reflection Questions (Answer in the cell below):

1. **Tool Comparison**: Based on your experience, when would you choose NLTK vs SpaCy? Consider factors like ease of use, accuracy, speed, and application type.
2. **Real-World Applications**: Describe a specific business problem where POS tagging would be valuable. How would you implement it?
3. **Limitations and Solutions**: What are the biggest limitations you discovered? How might you work around them?
4. **Future Learning**: What aspects of POS tagging would you like to explore further? (Neural approaches, custom training, domain adaptation, etc.)
5. **Integration**: How does POS tagging fit into larger NLP pipelines? What other NLP tasks might benefit from POS information?

✍️ Your Reflection (Write your answers here):

Remember Reflection is not description!

1. Tool Comparison: SpaCy worked faster and understood more words than NLTK. NLTK was slower and made more mistakes with confusing or short sentences. SpaCy is better for real-world use, especially when time matters.

2. Real-World Applications: POS tagging can help businesses understand what customers are saying, like if someone is upset or needs help fast. It can also be used in smart assistants, email sorting, and social media tools.

3. Limitations and Solutions: Sometimes taggers mess up when words have more than one meaning or when people use slang or make mistakes. We can fix this by cleaning the text first and using better models that look at full sentences. **4. Future Learning:** I want to learn how POS tagging works in bigger AI systems and how to improve it for messy language like text messages or tweets. I also want to try tagging in other languages.

5. Integration: [Now that I know about POS tagging, I can use it with other NLP tools like sentiment analysis or chatbots. It helps the computer understand what type of word is being used so it can respond the right way.]

📤 Submission Checklist

Before submitting your completed notebook, make sure you have:

- ☐ ✅ Completed all TODO sections with working code
- ☐ ✅ Answered all reflection questions thoughtfully
- ☐ ✅ Created at least one meaningful visualization
- ☐ ✅ Tested your code and fixed any errors
- ☐ ✅ Added comments explaining your approach
- ☐ ✅ Included insights from your analysis

📋 Submission Instructions:

1. **Save your notebook**: File → Save (or Ctrl+S)
2. **Download**: File → Download → Download .ipynb
3. **Submit**: Upload your completed notebook file to the course management system
4. **Filename**: Use format: `L05_LastName_FirstName_ITAI2373.ipynb or pdf`

🏆 Grading Criteria:

- **Code Completion (40%)**: All exercises completed with working code
- **Analysis Quality (30%)**: Thoughtful interpretation of results
- **Reflection Depth (20%)**: Insightful answers to reflection questions
- **Code Quality (10%)**: Clean, commented, well-organized code

---

🎉 Great Work!

You've successfully explored the fascinating world of POS tagging! You now understand how computers parse human language and can apply these techniques to solve real-world problems.

Keep exploring and happy coding! 🚀

```
import nltk
nltk.download('averaged_perceptron_tagger_eng')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
True
```