**Problem 1: Regex for Information Extraction**

**Question:**  
You are given the following text:

Contact us at support@example.com or call +91-9876543210.

Visit our website https://www.mywebsite.org for details.

Follow us on Twitter @TechGuru and use the hashtag #AI2025.

Meeting scheduled on 28/07/2025. Beware of badword1 and badword2.

**Tasks:**

1. Extract all **email addresses**.
2. Extract all **phone numbers** (Indian format).
3. Extract all **URLs**.
4. Extract all **hashtags**.
5. Extract all **mentions (@usernames)**.
6. Identify offensive words from the list ["badword1", "badword2", "spamword"].

**Answer (Python Code):**

import re

text = """Contact us at support@example.com or call +91-9876543210.

Visit our website https://www.mywebsite.org for details.

Follow us on Twitter @TechGuru and use the hashtag #AI2025.

Meeting scheduled on 28/07/2025. Beware of badword1 and badword2."""

emails = re.findall(r"[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}", text)

phones = re.findall(r"\+91-\d{10}", text)

urls = re.findall(r"https?://\S+", text)

hashtags = re.findall(r"#\w+", text)

mentions = re.findall(r"@\w+", text)

offensive\_words = [word for word in ["badword1", "badword2", "spamword"] if word in text]

print("Emails:", emails)

print("Phones:", phones)

print("URLs:", urls)

print("Hashtags:", hashtags)

print("Mentions:", mentions)

print("Offensive Words:", offensive\_words)

**Expected Output:**

Emails: ['support@example.com']

Phones: ['+91-9876543210']

URLs: ['https://www.mywebsite.org']

Hashtags: ['#AI2025']

Mentions: ['@TechGuru']

Offensive Words: ['badword1', 'badword2']

**Problem 2: N-Gram Model**

**Question:**  
Given the text:

"The future of AI is bright and full of opportunities."

**Tasks:**

1. Generate **unigrams**, **bigrams**, and **trigrams**.
2. Comment on how the context improves from unigram → trigram.

**Answer (Python Code):**

import nltk

from nltk.util import ngrams

nltk.download('punkt')

text = "The future of AI is bright and full of opportunities."

tokens = nltk.word\_tokenize(text)

unigrams = list(ngrams(tokens, 1))

bigrams = list(ngrams(tokens, 2))

trigrams = list(ngrams(tokens, 3))

print("Unigrams:", unigrams)

print("Bigrams:", bigrams)

print("Trigrams:", trigrams)

**Expected Output:**

Unigrams: [('The',), ('future',), ('of',), ('AI',), ('is',), ('bright',), ('and',), ('full',), ('of',), ('opportunities',), ('.',)]

Bigrams: [('The', 'future'), ('future', 'of'), ('of', 'AI'), ...]

Trigrams: [('The', 'future', 'of'), ('future', 'of', 'AI'), ...]

**Analysis:**

* **Unigrams**: Treats each word independently.
* **Bigrams**: Captures simple relationships (e.g., *"future of"*, *"of AI"*).
* **Trigrams**: Provides more context (e.g., *"The future of"*, *"full of opportunities"*) — better for predictive models.

**Problem 3: Edit Distance (Levenshtein)**

**Question:**  
Compute the **edit distance** between the words:

"kitten" → "sitting"

**Answer (Python Code):**

import nltk

nltk.download('edit\_distance')

from nltk.metrics import edit\_distance

word1 = "kitten"

word2 = "sitting"

distance = edit\_distance(word1, word2)

print(f"Edit Distance between '{word1}' and '{word2}':", distance)

**Expected Output:**

Edit Distance between 'kitten' and 'sitting': 3

**Explanation:**

* kitten → sitten (substitution)
* sitten → sittin (substitution)
* sittin → sitting (insertion)  
  **Total edits = 3**

**Problem 4: Chunking**

**Question:**  
Using the sentence:

" Donald Trump was born in Queens, New York."

Perform **POS tagging** and **chunking** to identify **noun phrases**.

**Answer (Python Code):**

import nltk  
# Download required NLTK resources  
nltk.download('punkt')  
nltk.download('averaged\_perceptron\_tagger')

# Input sentence  
sentence = "Donald Trump was born in Queens, New York."

# Tokenize and perform POS tagging  
tokens = nltk.word\_tokenize(sentence)  
tags = nltk.pos\_tag(tokens)

# Define a chunk grammar for Noun Phrases  
grammar = "NP: {<DT>?<JJ>\*<NNP>+}"

# Create the chunk parser and parse the tagged sentencechunk\_parser = nltk.RegexpParser(grammar)  
tree = chunk\_parser.parse(tags)

# Visualize the chunk tree (optional)  
tree.draw()

# Print output  
print("POS Tags:", tags)  
print("Chunks:", tree)

**Expected Output (POS Tags):**

[('Donald', 'NNP'), ('Trump', 'NNP'), ('was', 'VBD'),('born', 'VBN'), ('in', 'IN'), ('Queens', 'NNP'), (',', ','), ('New', 'NNP'), ('York', 'NNP'), ('.', '.')]

**Identified Noun Phrases:**

 Donald Trump

 Queens

 New York

**Problem 5: Named Entity Recognition (NER)**

**Question:**  
Perform NER on the following text:

"Apple Inc. is planning to open a new office in Mumbai by 2026."

**Answer (Python Code using SpaCy):**

import spacy

nlp = spacy.load("en\_core\_web\_sm")

text = "Apple Inc. is planning to open a new office in Mumbai by 2026."

doc = nlp(text)

for ent in doc.ents:

print(ent.text, ent.label\_)

**Expected Output:**

Apple Inc. ORG

Mumbai GPE

2026 DATE

**Explanation:**

* **Apple Inc. → ORG** (Organization)
* **Mumbai → GPE** (Geopolitical Entity / Location)
* **2026 → DATE**

**Problem Set: Text Preprocessing**

**Problem 1: Tokenization Methods (Word, Subword, Character)**

**Question:**  
You are given the text:

"Artificial Intelligence is revolutionizing the world."

**Tasks:**

1. Perform **word-level tokenization** using **NLTK** and **SpaCy**.
2. Perform **character-level tokenization**.
3. Perform **subword tokenization** using **WordPiece** or **BPE**.
4. Compare the outputs.

**Answer (Python Code):**

import nltk, spacy

from transformers import BertTokenizer

nltk.download('punkt')

nlp = spacy.load("en\_core\_web\_sm")

text = "Artificial Intelligence is revolutionizing the world."

# Word-level tokenization

nltk\_tokens = nltk.word\_tokenize(text)

spacy\_tokens = [token.text for token in nlp(text)]

# Character-level tokenization

char\_tokens = list(text)

# Subword tokenization (WordPiece - BERT)

tokenizer = BertTokenizer.from\_pretrained("bert-base-uncased")

subword\_tokens = tokenizer.tokenize(text)

print("NLTK Word Tokens:", nltk\_tokens)

print("SpaCy Word Tokens:", spacy\_tokens)

print("Character Tokens:", char\_tokens[:20]) # first 20 chars

print("Subword Tokens (WordPiece):", subword\_tokens)

**Expected Output:**

NLTK Word Tokens: ['Artificial', 'Intelligence', 'is', 'revolutionizing', 'the', 'world', '.']

SpaCy Word Tokens: ['Artificial', 'Intelligence', 'is', 'revolutionizing', 'the', 'world', '.']

Character Tokens: ['A', 'r', 't', 'i', 'f', 'i', 'c', 'i', 'a', 'l', ' ', 'I', 'n', 't', 'e', 'l', 'l', 'i', 'g', 'e']

Subword Tokens: ['artificial', 'intelligence', 'is', 'revolutionizing', 'the', 'world', '.']

**Comparison:**

* **Word Tokenization:** Splits text into words.
* **Character Tokenization:** Useful for handling unknown/misspelled words.
* **Subword Tokenization (WordPiece):** Breaks unknown words into meaningful sub-parts.

**Problem 2: Stopword Removal**

**Question:**  
Given the text:

"This is an example sentence, showing the effect of stopword removal."

**Tasks:**

1. Remove stopwords using **NLTK**.
2. Remove stopwords using **SpaCy**.
3. Compare word counts before and after removal.

**Answer (Python Code):**

import nltk, spacy

from nltk.corpus import stopwords

nltk.download('stopwords')

nltk.download('punkt')

text = "This is an example sentence, showing the effect of stopword removal."

tokens = nltk.word\_tokenize(text)

# NLTK Stopwords

nltk\_stopwords = set(stopwords.words('english'))

nltk\_filtered = [word for word in tokens if word.lower() not in nltk\_stopwords]

# SpaCy Stopwords

nlp = spacy.load("en\_core\_web\_sm")

spacy\_filtered = [token.text for token in nlp(text) if not token.is\_stop]

print("Original Word Count:", len(tokens))

print("After NLTK Stopword Removal:", nltk\_filtered)

print("After SpaCy Stopword Removal:", spacy\_filtered)

**Expected Output:**

Original Word Count: 11

After NLTK Stopword Removal: ['example', 'sentence', ',', 'showing', 'effect', 'stopword', 'removal', '.']

After SpaCy Stopword Removal: ['example', 'sentence', ',', 'showing', 'effect', 'stopword', 'removal', '.']

**Observation:**  
Removing stopwords reduces word count and highlights content-bearing words.

**Problem 3: Stemming vs Lemmatization**

**Question:**  
Using the list of words:

["running", "flies", "better", "studies", "wolves", "cities"]

**Tasks:**

1. Apply **Porter Stemmer** and **Lancaster Stemmer**.
2. Apply **WordNet Lemmatizer**.
3. Compare vocabulary size reduction.

**Answer (Python Code):**

from nltk.stem import PorterStemmer, LancasterStemmer, WordNetLemmatizer

import nltk

nltk.download('wordnet')

nltk.download('omw-1.4')

words = ["running", "flies", "better", "studies", "wolves", "cities"]

porter = PorterStemmer()

lancaster = LancasterStemmer()

lemmatizer = WordNetLemmatizer()

print("Original Words:", words)

print("Porter:", [porter.stem(w) for w in words])

print("Lancaster:", [lancaster.stem(w) for w in words])

print("Lemmatizer:", [lemmatizer.lemmatize(w) for w in words])

**Expected Output:**

Original Words: ['running', 'flies', 'better', 'studies', 'wolves', 'cities']

Porter: ['run', 'fli', 'better', 'studi', 'wolv', 'citi']

Lancaster: ['run', 'fli', 'bet', 'study', 'wolv', 'city']

Lemmatizer: ['running', 'fly', 'better', 'study', 'wolf', 'city']

**Comparison:**

* **Stemming**: Aggressive cutting (may produce non-words).
* **Lemmatization**: More accurate (uses dictionary + POS).
* Lemmatization keeps **semantically correct forms**.

**Problem 4: Noise Removal & Text Normalization**

**Question:**  
Given the text:

"RT @user123!!! The PRICE of Bitcoin hit $30,000 today!!! #Crypto 🚀🚀"

**Tasks:**

1. Convert text to lowercase.
2. Remove punctuation, special symbols, hashtags, and mentions.
3. Remove numbers.
4. Display cleaned text.

**Answer (Python Code):**

import re

import string

text = "RT @user123!!! The PRICE of Bitcoin hit $30,000 today!!! #Crypto 🚀🚀"

# Lowercase

text\_lower = text.lower()

# Remove mentions, hashtags, and special chars

cleaned = re.sub(r"@\w+|#\w+|http\S+|[^a-zA-Z\s]", "", text\_lower)

# Remove extra spaces

cleaned = re.sub(r"\s+", " ", cleaned).strip()

print("Original:", text)

print("Cleaned:", cleaned)

**Expected Output:**

Original: RT @user123!!! The PRICE of Bitcoin hit $30,000 today!!! #Crypto 🚀🚀

Cleaned: rt the price of bitcoin hit today

**Problem Set: Part-of-Speech (POS) Tagging**

**Problem 1: Understanding POS Taggers (3.1)**

**Question:**  
Given the text:

"John loves eating pizza while Mary reads books in the library."

1. Perform **POS tagging** using **NLTK**.
2. Perform **POS tagging** using **SpaCy**.
3. Compare the outputs.

**Answer (Python Code):**

import nltk, spacy

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

text = "John loves eating pizza while Mary reads books in the library."

# NLTK POS Tagging

tokens = nltk.word\_tokenize(text)

nltk\_tags = nltk.pos\_tag(tokens)

# SpaCy POS Tagging

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp(text)

spacy\_tags = [(token.text, token.pos\_) for token in doc]

print("NLTK Tags:", nltk\_tags)

print("SpaCy Tags:", spacy\_tags)

**Expected Output (simplified):**

NLTK Tags: [('John', 'NNP'), ('loves', 'VBZ'), ('eating', 'VBG'), ('pizza', 'NN'), ...]

SpaCy Tags: [('John', 'PROPN'), ('loves', 'VERB'), ('eating', 'VERB'), ('pizza', 'NOUN'), ...]

**Observation:**

* NLTK uses **Penn Treebank tags** (NN, VBZ, VBG).
* SpaCy uses **Universal POS tags** (NOUN, VERB, PROPN).

**Problem 2: Performance Evaluation (3.2)**

**Question:**  
You are given the following **gold standard POS tags** for a short text:

Sentence: "The dog chased the cat."

Gold Tags: [('The', 'DT'), ('dog', 'NN'), ('chased', 'VBD'), ('the', 'DT'), ('cat', 'NN')]

Compare **NLTK** and **SpaCy** taggers against the gold standard and compute **accuracy**.

**Answer (Python Code):**

gold = [('The', 'DT'), ('dog', 'NN'), ('chased', 'VBD'), ('the', 'DT'), ('cat', 'NN')]

text = "The dog chased the cat."

# NLTK

tokens = nltk.word\_tokenize(text)

nltk\_tags = nltk.pos\_tag(tokens)

# SpaCy

doc = nlp(text)

spacy\_tags = [(token.text, token.tag\_) for token in doc]

# Accuracy function

def compute\_accuracy(pred, gold):

correct = sum(1 for p, g in zip(pred, gold) if p[1] == g[1])

return correct / len(gold)

print("NLTK Accuracy:", compute\_accuracy(nltk\_tags, gold))

print("SpaCy Accuracy:", compute\_accuracy(spacy\_tags, gold))

**Expected Output (example):**

NLTK Accuracy: 1.0

SpaCy Accuracy: 0.8

**Observation:**  
Different models may have varying accuracy depending on training corpora.

**Problem 3: Rule-Based POS Tagging (3.3)**

**Question:**  
Create a **rule-based POS tagger** using **regular expressions** to tag determiners, nouns, and verbs in the sentence:

"Ravi plays cricket and watches TV daily."

**Answer (Python Code):**

from nltk import RegexpTagger

patterns = [

(r'.\*ing$', 'VBG'), # gerunds

(r'.\*ed$', 'VBD'), # past tense verbs

(r'.\*es$', 'VBZ'), # 3rd person singular verbs

(r'^Ravi$', 'NNP'), # proper noun

(r'cricket|TV', 'NN'), # nouns

(r'the|and|a|daily', 'DT'),# determiners/conjunctions

(r'.\*', 'NN') # default noun

]

tagger = RegexpTagger(patterns)

sentence = ['Ravi', 'plays', 'cricket', 'and', 'watches', 'TV', 'daily']

print("Rule-Based Tags:", tagger.tag(sentence))

**Expected Output:**

Rule-Based Tags: [('Ravi', 'NNP'), ('plays', 'VBZ'), ('cricket', 'NN'),

('and', 'DT'), ('watches', 'VBZ'), ('TV', 'NN'), ('daily', 'DT')]

**Observation:**  
Rule-based taggers are interpretable but may fail for unseen words.

**Problem 4: Statistical POS Tagging (3.4)**

**Question:**  
Train a **Hidden Markov Model (HMM)** POS tagger on a small dataset using NLTK’s Brown corpus and test it on:

"The quick brown fox jumps over the lazy dog."

**Answer (Python Code):**

from nltk.corpus import brown

from nltk.tag import hmm

nltk.download('brown')

nltk.download('universal\_tagset')

# Prepare training data

train\_data = brown.tagged\_sents(categories='news', tagset='universal')[:500]

# Train HMM Tagger

trainer = hmm.HiddenMarkovModelTrainer()

hmm\_tagger = trainer.train\_supervised(train\_data)

sentence = "The quick brown fox jumps over the lazy dog".split()

print("HMM Tags:", hmm\_tagger.tag(sentence))

**Expected Output (example):**

HMM Tags: [('The', 'DET'), ('quick', 'ADJ'), ('brown', 'ADJ'),

('fox', 'NOUN'), ('jumps', 'VERB'), ('over', 'ADP'),

('the', 'DET'), ('lazy', 'ADJ'), ('dog', 'NOUN')]

**Observation:**  
HMM taggers use probabilistic models based on training data.

**Problem 5: Deep Learning for POS Tagging (Optional, 3.5)**

**Question:**  
Use a **pretrained Transformer model (BERT)** for POS tagging on:

"Elon Musk founded SpaceX in 2002."

**Answer (Python Code using Hugging Face):**

from transformers import pipeline

nlp\_pipeline = pipeline("token-classification", model="dslim/bert-base-NER")

text = "Elon Musk founded SpaceX in 2002."

output = nlp\_pipeline(text)

for item in output:

print(item)

**Expected Output (example):**

{'word': 'Elon', 'entity': 'B-PER', 'score': 0.99}

{'word': 'Musk', 'entity': 'I-PER', 'score': 0.99}

{'word': 'SpaceX', 'entity': 'ORG', 'score': 0.99}

{'word': '2002', 'entity': 'DATE', 'score': 0.98}