## Lab1: Text Processing and Regular Expression

17/06/2025

## Executive Summary

This report analyzes a Python script designed for text processing and regular expression pattern matching as part of an NLP laboratory exercise. The code demonstrates fundamental text analysis techniques including tokenization, frequency analysis, and advanced pattern matching using regular expressions.

## Code Structure Overview

The script is organized into three main sections: 1. **Text Analysis of Paragraph 1** - Linguistic content analysis 2. **Text Analysis of Paragraph 2** - Lorem Ipsum analysis for comparison 3. **Regular Expression Pattern Matching** - Two sets of regex exercises

## Section 1: Linguistic Text Analysis

### Methodology

The code analyzes a paragraph about parsing in computational linguistics using NLTK’s tokenization capabilities. The analysis includes: - Word tokenization using word\_tokenize() - Text normalization (lowercasing) - Alphabetic filtering to remove punctuation - Statistical analysis of word frequencies

### Key Findings for Paragraph 1

**Text Sample**: Technical content about parsing in NLP - **Total Words**: 72 words after filtering - **Unique Words**: 54 distinct terms - **Vocabulary Richness**: 75% (54/72) indicating diverse vocabulary - **Most Frequent Word**: “and” (appears 4 times) - **Longest Word**: “congratulating” (13 characters)

### Technical Terms Identified

The paragraph contains specialized NLP terminology including: - parsing, morphological, syntactic, semantic - tokenization, stemming, morphemes - linguistic structures and affixes

## Section 2: Lorem Ipsum Analysis

### Comparative Analysis

**Text Sample**: Standard Lorem Ipsum placeholder text - **Total Words**: 61 words - **Unique Words**: 47 distinct terms - **Vocabulary Richness**: 77% (47/61) - **Most Frequent Word**: “et” (appears 4 times) - **Longest Word**: “perferendis” (11 characters)

### Observations

The Lorem Ipsum text shows similar vocabulary richness to the technical text, demonstrating that placeholder text maintains linguistic diversity patterns found in natural language.

## Section 3: Regular Expression Pattern Matching

### Exercise Set 2.1 - Basic Pattern Matching

The code implements seven different regex patterns:

1. **Alphabetic Strings** (\b[a-zA-Z]+\b)
   * Successfully extracts pure alphabetic words
   * Filters out alphanumeric combinations
2. **Lowercase Words Ending in ‘b’** (\b[a-z]\*b\b)
   * Identifies words like “cab”, “grab”, “slab”
   * Correctly excludes capitalized words
3. **Consecutive Repeated Words** (\b(\w+)\s+\1\b)
   * Detects word repetitions like “the the”
   * Uses backreference for pattern matching
4. **‘a’ Preceded and Followed by ‘b’** (^(b|bab)\*$)
   * Validates strings where every ‘a’ is surrounded by ‘b’
   * Uses alternation and repetition quantifiers
5. **Lines Starting with Integer, Ending with Word** (^\d+\b.\*\b[a-zA-Z]+$)
   * Matches lines with numeric start and alphabetic end
   * Demonstrates anchoring and word boundaries
6. **Strings Containing Both ‘grotto’ and ‘raven’** ((?=.\*\bgrotto\b)(?=.\*\braven\b))
   * Uses positive lookahead for multiple word detection
   * Advanced regex technique for complex conditions
7. **First Word Capture** (^[\"\'(]\*([A-Z][a-z]\*))
   * Extracts sentence-initial words
   * Handles common punctuation prefixes

### Exercise Set 2.2 - Word-Based Pattern Analysis

Applied to a multi-line test corpus, the patterns successfully identify:

* **Single Alphabetic Words**: Complete word extraction
* **Lowercase Words**: Case-sensitive filtering
* **Capitalized Words**: Proper noun identification
* **4-Letter Words**: Length-specific matching
* **Repeated Words**: Redundancy detection
* **Words Ending in ‘ing’**: Morphological pattern matching
* **Words with Double Letters**: Orthographic pattern recognition

## Technical Implementation Analysis

### Strengths

1. **Proper Library Usage**: Effective use of NLTK for tokenization
2. **Clean Data Processing**: Appropriate filtering and normalization
3. **Comprehensive Analysis**: Multiple statistical measures
4. **Progressive Complexity**: Regex patterns increase in sophistication
5. **Practical Applications**: Patterns address real-world text processing needs

### Code Quality Observations

* Well-structured with clear section divisions
* Appropriate use of Python data structures (Counter, sets)
* Good variable naming conventions
* Comprehensive test cases for regex validation

## Applications and Use Cases

This code demonstrates techniques applicable to: - **Text Preprocessing**: Tokenization and normalization - **Content Analysis**: Frequency analysis and vocabulary assessment - **Data Validation**: Pattern matching for input validation - **Information Extraction**: Targeted text pattern identification - **Linguistic Research**: Morphological and syntactic analysis

## Recommendations for Enhancement

1. **Error Handling**: Add try-catch blocks for robust execution
2. **Parameterization**: Make text inputs configurable
3. **Visualization**: Add charts for frequency distributions
4. **Performance Metrics**: Include execution time measurements
5. **Extended Analysis**: Add readability scores and complexity metrics

## Conclusion

The code successfully demonstrates fundamental NLP techniques combining statistical text analysis with advanced pattern matching. The implementation shows progression from basic tokenization to complex regex operations, making it suitable for educational purposes and practical text processing applications. The variety of patterns covered provides a solid foundation for more advanced natural language processing tasks.