Basic Data Science Projects using Python, NumPy, Pandas, Matplotlib, Regular Expressions, and SQL

By: Prof. James Abello, Haoyang Zhang

Computer Science Department

Rutgers University

Nov. 21, 2024.

Topic 7: Leet Speak (regular expressions)

Objective: Translate English text to Leet Speak and vice versa using regular expressions.

Estimated Completion Time: 6 hours

Leet (or "1337") speak is a language that uses various combinations of characters to replace Latin letters. For example, the word "leet" is written as "1337" in leet speak.

In this project, you will write a Python program to:

- Encode a given string into leet speak by replacing certain letters with their corresponding leet speak characters.
- Decode a leet speak string back to the original string by reversing the substitutions.

A good reference is:

> L33t sp34k ch34t sh33t by Roald Craenen
>
> https://www.gamehouse.com/blog/leet-speak-cheat-sheet/

Level 1

In this level, we will focus on a basic one-to-one mapping of characters without ambiguity. The mapping is as follows:

Latin	Leet
A	4
B	8
C	(

Latin	Leet
D	)
E	3
F	ƒ
G	6
H	#
I	!
J	]
K	|
L	1
M	м
N	и
O	Ø
P	9
Q	2
R	Я
S	5
T	7
U	µ
V	√
W	ω
X	Ж
Y	¥
Z	%

Note: this version does not map any characters to latin letters.

Task 1.1

Write a Python function `encode_all_1()` that takes a string as input and encodes all Latin letters to leet speak using the mapping above.

You can assume that the input text contains only uppercase Latin letters, lowercase Latin letters, and spaces.

```
def encode_all_1(text):
    """
    Encode all Latin letters to leet speak
    IN: text, str, input text
    OUT: str, leet speak text
```

```
    """
    pass
```

Task 1.2

Write a Python function `decode_all_1()` to reverse the operation of `encode_all_1()`.

You may assume that the final text contains only uppercase Latin letters, lowercase Latin letters, and spaces.

```
def decode_all_1(text):
    """
    Decode all leet speak to Latin letters
    IN: text, str, leet speak text
    OUT: str, Latin letters text
    """
    pass
```

Task 1.3

Write a Python function `encode_partially_1()` that takes a string and a number `p` between 0 and 1 as input, and encodes each Latin letter to leet speak with probability `p`.

For example, if `p = 0.5`, then each Latin letter has a 50% chance of being encoded to leet speak.

```
def encode_partially_1(text, p):
    """
    Encode each Latin letter to leet speak with probability p
    IN: text, str, input text
        p, float, probability of encoding
    OUT: str, partially encoded text
    """
    pass
```

Task 1.4

Write a Python function `decode_partially_1()` to reverse the operation of `encode_partially_1()`.

Observation: Any Latin letter can be "decoded" as it is, since this version of leet speak does not map any character back to a Latin letter.

```
def decode_partially_1(text):
    """
    Decode each Latin letter from leet speak
    IN: text, str, partially encoded text
    OUT: str, partially decoded text
    """
    pass
```

Level 2

In this level, users can define their own mapping of characters to leet speak as a JSON dictionary, ensuring no ambiguity (using a prefix-free code).

A prefix-free code is a type of coding system in which **no code is the prefix of another code**. For example, if `C` is encoded as `(`, then no other character can be encoded as `(` or as a sequence of characters starting with `(`.

There is a mathematical proof that **if a code is prefix-free, there is a unique way to decode the encoded text**. In other words, there is no ambiguity when decoding the encoded text.

For example, the JSON dictionary for the basic mapping in Level 1 is:

```
{
    "A": ["4"],
    "B": ["8"],
    "C": ["("],
    "D": [")"],
    "E": ["3"],
    "F": ["ƒ"],
    "G": ["6"],
    "H": ["#"],
    "I": ["!"],
    "J": ["]"],
    "K": ["|"],
    "L": ["1"],
    "M": ["м"],
    "N": ["и"],
    "O": ["Ø"],
    "P": ["9"],
    "Q": ["2"],
    "R": ["Я"],
    "S": ["5"],
    "T": ["7"],
    "U": ["µ"],
    "V": ["√"],
    "W": ["ω"],
    "X": ["Ж"],
    "Y": ["¥"],
    "Z": ["%"]
}
```

A more complex version that allows one Latin letter mapping to multiple leet speak characters or sequences of characters is:

```
{
    "A": ["4", "@", "Д"],
    "B": ["8", "ß"],
    "C": ["(", "<", "©", "¢"],
    "D": [")", ">"],
    "E": ["3", "£"],
    "F": ["ƒ"],
    "G": ["6", "&"],
```

```
    "H": ["#", "|-|"],
    "I": ["!"],
    "J": ["]", "_|"],
    "K": ["|<"],
    "L": ["1", "|_"],
    "M": ["м", "|\/|"],
    "N": ["и", "|\\|"],
    "O": ["Ø"],
    "P": ["9", "|°"],
    "Q": ["2"],
    "R": ["Я", "|~"],
    "S": ["5", "$", "§"],
    "T": ["7", "-|-"],
    "U": ["µ"],
    "V": ["√"],
    "W": ["ω", "\^/"],
    "X": ["Ж", "×"],
    "Y": ["¥", "γ"],
    "Z": ["%"]
}
```

Notice The character `K` has been changed to `|<` instead of `|` to allow other characters to be mapped to a sequence starting with `|`.

Task 2.1

Write a Python function `check_prefix_free_2()` that takes a JSON file name as input and checks if the JSON file specifies a prefix-free code.

```
def check_prefix_free_2(json_file):
    """
    Check if the JSON file specifies a prefix-free code
    IN: json_file, str, JSON file name
    OUT: dict or None, dictionary of characters mapping to leet speak
or None if not prefix-free
    """
    pass
```

Task 2.2

Write a Python function `encode_partially_2()` that takes a string and a number `p` between `0` and `1` as input and encodes each Latin letter to leet speak with probability `p` using the user-defined mapping. When a multiple mapping is possible, choose one randomly.

```
def encode_partially_2(text, p, mapping):
    """
    Encode each Latin letter to leet speak with probability p using the
user-defined mapping
    IN: text, str, input text
        p, float, probability of encoding
        mapping, dict, user-defined mapping
    OUT: str, partially encoded text
    """
    pass
```

Or, you can create a constructor for `encode_partially_2()` that takes the mapping as an argument:

```
def encode_partially_2(mapping):
    """
    Constructor for encoding each Latin letter to leet speak with
probability p using the user-defined mapping
    IN: mapping, dict, user-defined mapping
    OUT: function, (text: str, p: float) -> str, encode latin letters
to leet speak with probability p
    """
    # define the encoding function using the mapping
    def encode_partially_2(text, p):
        """
        Encode each Latin letter to leet speak with probability p using
the user-defined mapping
        IN: text, str, input text
            p, float, probability of encoding
        OUT: str, partially encoded text
        """
        # specify the encoding logic using the mapping
        pass

    # return the encoding function
    return encode_partially_2
```

Task 2.3

Write a Python function `decode_partially_2()` to reverse the operation of `encode_partially_2()` using the user-defined mapping.

```
def decode_partially_2(text, mapping):
    """
    Decode each Latin letter from leet speak using the user-defined
mapping
    IN: text, str, partially encoded text
        mapping, dict, user-defined mapping
    OUT: str, partially decoded text
    """
    pass
```

Or, you can create a constructor for `decode_partially_2()` that takes the mapping as an argument:

```
def decode_partially_2(mapping):
    """
    Constructor for decoding each Latin letter from leet speak using
the user-defined mapping
    IN: mapping, dict, user-defined mapping
    OUT: function, (text: str) -> str, decode latin letters from leet
speak
    """
    # define the decoding function using the mapping
    def decode_partially_2(text):
        """
        Decode each Latin letter from leet speak using the user-defined
```

```
mapping
        IN: text, str, partially encoded text
        OUT: str, partially decoded text
        """
        # specify the decoding logic using the mapping
        pass

    # return the decoding function
    return decode_partially_2
```

Level 3

In this level:

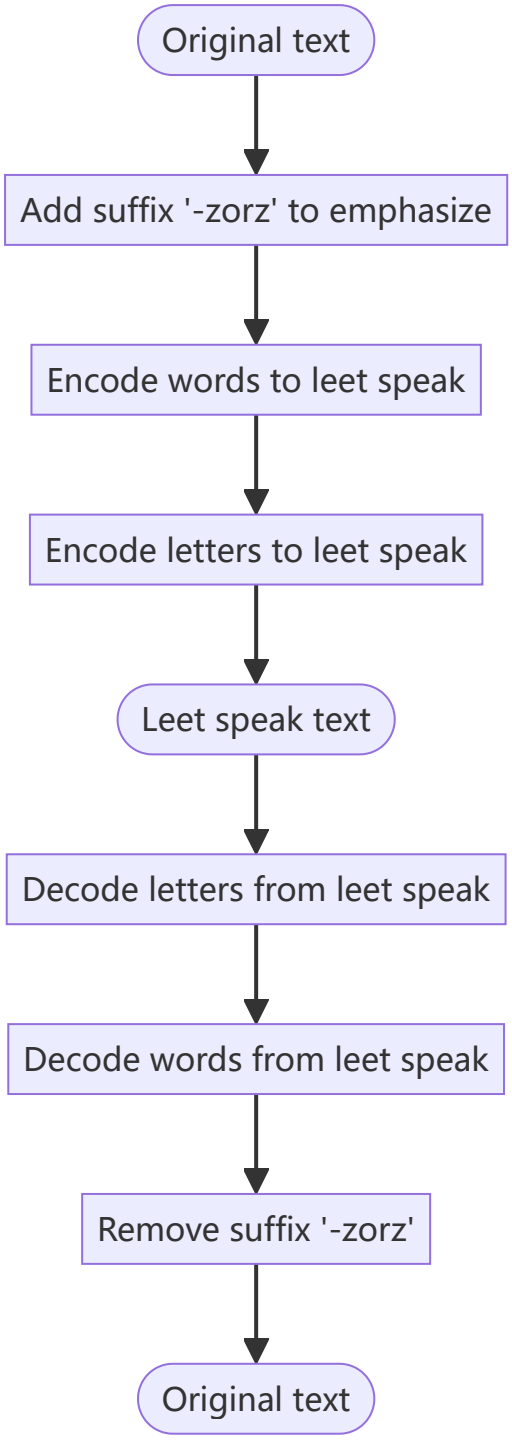
- Words can be emphasized by adding the suffix "-zorz". For example, "leet" can be emphasized as "leetzorz" and further encoded to "1337%or|~z".
- Users can define their own mapping, not only for single Latin letters but also for words, to leet speak without ambiguity (using a prefix-free code).

For example:

```
{
    "words": {
        "real": ["٢٤٨١"],
        "eye": ["٤٢٤"],
        "age": ["٨٩٤"],
        "euro": ["٤٧٢٥"],
        "total": ["٢٥٢٨١"]
    },
    "letters": {
        "A": ["4", "@", "Д"],
        "B": ["8", "ß"],
        "C": ["(", "<", "©", "¢"],
        "D": [")", ">"],
        "E": ["3", "£"],
        "F": ["ƒ"],
        "G": ["6", "&"],
        "H": ["#", "|-|"],
        "I": ["!"],
        "J": ["]", "_|"],
        "K": ["|<"],
        "L": ["1", "|_"],
        "M": ["м", "|\/|"],
        "N": ["и", "|\\|"],
        "O": ["Ø"],
        "P": ["9", "|°"],
        "Q": ["2"],
        "R": ["Я", "|~"],
        "S": ["5", "$", "§"],
        "T": ["7", "-|-"],
        "U": ["µ"],
        "V": ["√"],
        "W": ["ω", "\^/"],
        "X": ["Ж", "×"],
        "Y": ["¥", "γ"],
```

```
        "Z": ["%"]
    }
}
```

The encoding and decoding sequence is as follows:



Task 3.1

Write a Python function `check_prefix_free_3()` similar to Task 2.1 that checks if the JSON file specifies a prefix-free code for both words and letters.

```
def check_prefix_free_3(json_file):
    """
    Check if the JSON file specifies a prefix-free code for both words
```

```
and letters
    IN: json_file, str, JSON file name
    OUT: dict or None, dictionary of characters mapping to leet speak
or None if not prefix-free
    """
    pass
```

Task 3.2

Write a Python function `add_emphasis_3()` that takes a string and a list of important words as input and emphasizes the important words in the string by adding the suffix "-zorz".

```
def add_emphasis_3(text, important_words):
    """
    Add emphasis to important words by adding the suffix '-zorz'
    IN: text, str, input text
        important_words, list[str], list of important words
    OUT: str, text with emphasized words
    """
    pass
```

Task 3.3

Write a Python function `encode_partially_words_3()` that takes a string and a number `p` between `0` and `1` as input and encodes words to leet speak with probability `p` using the user-defined mapping.

```
def encode_partially_words_3(text, p, mapping):
    """
    Encode words to leet speak with probability p using the user-
defined mapping
    IN: text, str, input text
        p, float, probability of encoding
        mapping, dict, user-defined mapping
    OUT: str, partially encoded text
    """
    pass
```

Or, you can create a constructor for `encode_partially_words_3()` that takes the mapping as an argument:

```
def encode_partially_words_3(mapping):
    """
    Constructor for encoding words to leet speak with probability p
using the user-defined mapping
    IN: mapping, dict, user-defined mapping
    OUT: function, (text: str, p: float) -> str, encode words to leet
speak with probability p
    """
    # define the encoding function using the mapping
    def encode_partially_words_3(text, p):
        """
        Encode words to leet speak with probability p using the user-
defined mapping
```

```
        IN: text, str, input text
            p, float, probability of encoding
        OUT: str, partially encoded text
        """
        # specify the encoding logic using the mapping
        pass

    # return the encoding function
    return encode_partially_words_3
```

Task 3.4

Write Python function `encode_partially_letters_3()` similar to Task 2.2 that encodes each Latin letter to leet speak with probability `p` using the user-defined mapping.

```
def encode_partially_letters_3(text, p, mapping):
    """
    Encode each Latin letter to leet speak with probability p using the
user-defined mapping
    IN: text, str, input text
        p, float, probability of encoding
        mapping, dict, user-defined mapping
    OUT: str, partially encoded text
    """
    pass
```

Or, you can create a constructor for `encode_partially_letters_3()` that takes the mapping as an argument:

```
def encode_partially_letters_3(mapping):
    """
    Constructor for encoding each Latin letter to leet speak with
probability p using the user-defined mapping
    IN: mapping, dict, user-defined mapping
    OUT: function, (text: str, p: float) -> str, encode latin letters
to leet speak with probability p
    """
    # define the encoding function using the mapping
    def encode_partially_letters_3(text, p):
        """
        Encode each Latin letter to leet speak with probability p using
the user-defined mapping
        IN: text, str, input text
            p, float, probability of encoding
        OUT: str, partially encoded text
        """
        # specify the encoding logic using the mapping
        pass

    # return the encoding function
    return encode_partially_letters_3
```

Task 3.5

Write a Python function `decode_partially_words_3()` to reverse the operation of `encode_partially_words_3()` using the user-defined mapping.

```
def decode_partially_words_3(text, mapping):
    """
    Decode words from leet speak using the user-defined mapping
    IN: text, str, partially encoded text
        mapping, dict, user-defined mapping
    OUT: str, partially decoded text
    """
    pass
```

Or, you can create a constructor for `decode_partially_words_3()` that takes the mapping as an argument:

```
def decode_partially_words_3(mapping):
    """
    Constructor for decoding words from leet speak using the user-
defined mapping
    IN: mapping, dict, user-defined mapping
    OUT: function, (text: str) -> str, decode words from leet speak
    """
    # define the decoding function using the mapping
    def decode_partially_words_3(text):
        """
        Decode words from leet speak using the user-defined mapping
        IN: text, str, partially encoded text
        OUT: str, partially decoded text
        """
        # specify the decoding logic using the mapping
        pass

    # return the decoding function
    return decode_partially_words_3
```

Task 3.6

Write a Python function `decode_partially_letters_3()` to reverse the operation of `encode_partially_letters_3()` using the user-defined mapping.

```
def decode_partially_letters_3(text, mapping):
    """
    Decode each Latin letter from leet speak using the user-defined
mapping
    IN: text, str, partially encoded text
        mapping, dict, user-defined mapping
    OUT: str, partially decoded text
    """
    pass
```

Or, you can create a constructor for `decode_partially_letters_3()` that takes the mapping as an argument:

```
def decode_partially_letters_3(mapping):
    """
    Constructor for decoding each Latin letter from leet speak using
```

```
the user-defined mapping
    IN: mapping, dict, user-defined mapping
    OUT: function, (text: str) -> str, decode latin letters from leet
speak
    """
    # define the decoding function using the mapping
    def decode_partially_letters_3(text):
        """
        Decode each Latin letter from leet speak using the user-defined
mapping
        IN: text, str, partially encoded text
        OUT: str, partially decoded text
        """
        # specify the decoding logic using the mapping
        pass

    # return the decoding function
    return decode_partially_letters_3
```

Task 3.7

Write a Python function `remove_emphasis_3()` that takes a string and removes the suffix "-zorz" from words.

```
def remove_emphasis_3(text):
    """
    Remove the suffix '-zorz' from words
    IN: text, str, input text
    OUT: str, text with emphasized suffix `-zorz` removed
    """
    pass
```

References

> - Blashki, Katherine; Nichol, Sophie (2005). "Game Geek's Goss: Linguistic Creativity In Young Males Within An Online University Forum" (PDF). Australian Journal of Emerging Technologies and Society. 3 (2): 77–86.
> - LeBlanc, Tracy Rene (May 2005). "Is There A Translator in Teh House?": Cultural and Discourse Analysis of a Virtual Speech Community on an Internet Message Board (MA thesis). Louisiana State University. doi:10.31390/gradschool_theses.4112
> - Perea, M.; Duñabeitia, J. A.; Carreiras, M. (2008). "R34D1Ng W0Rd5 W1Th Numb3R5" (PDF). Journal of Experimental Psychology: Human Perception and Performance. 34 (1): 237–241. doi:10.1037/0096-1523.34.1.237. ISSN 0096-1523. PMID 18248151. S2CID 6054151
> - Raymond, Eric R.; Steele, Guy L. (1996). The New Hacker's Dictionary. MIT Press. ISBN 978-0-262-68092-9.