University of Hertfordshire

(Affiliated with Nebula Institute of Technology, Sri Lanka)

Module Name	:	Information Theory
Module Code	:	5FTC2151
Assignment Title	:	CW1 – Decoding Real-World Data
	Submitted by:	
Student Name	:	S. A. L. M. Mihiliya Jayasiri
Student ID	:	23113610
Degree Programme	:	BSc (Hons) Data Science
Department	:	Physics, Engineering and Computer Science
Academic Year	:	Year 2 - 2025/26
	Submitted to:	
Lecturer	:	Ms. Narmada Karunarathne
Date of Submission	:	30/09/2025

Table of Contents

List of Tables

List of Figures

Task 1 - Entropy Analysis

Entropy in information theory, introduced by Claude Shannon in 1948, measures the average amount of information contained per character in a dataset. This report analyzes the entropy of a short English text dataset with 517 characters using character frequency analysis. The calculations were performed in Microsoft Excel, while Python was used for visualization.

1) Dataset Selection

The selected dataset is a short excerpt from the Wikipedia article on Entropy (Information Theory) (contributors, 2025) [2].

The concept of information entropy was introduced by Claude Shannon in his 1948 paper "A Mathematical Theory of Communication", and is also referred to as Shannon entropy. Shannon's theory defines a data communication system composed of three elements: a source of data, a communication channel, and a receiver. The "fundamental problem of communication" – as expressed by Shannon – is for the receiver to be able to identify what data was generated by the source, based on the signal it receives through the channel.

This text was chosen because it is domain relevant, sufficiently long and contains a mix of letters, digits, spaces, and punctuation.

2) Character Frequency Calculation

The frequency of each character was computed using Excel and is depicted in Table 1 below. Spaces and letters such as 'e', 'n', and 'a' dominate the dataset. The total dataset length is 517 characters.

Character	Frequency	Character	Frequency
e	52	**v**	3
n	41	**w**	3
a	40	**x**	1
o	37	**S**	4
t	31	**T**	3
i	24	**C**	2
r	23	**A**	1
s	21	**M**	1
h	20	**1**	1
c	19	**4**	1
d	16	**8**	1

m	15	**9**	1
f	11	**Space**	83
l	10	**"**	4
u	10	**,**	4
y	9	**.**	3
p	8	**–**	2
b	7	**’**	1
g	3	**:**	1

Table 1: Task 1 - Character Frequencies of Selected Text

3) Computing Shannon’s Entropy

The Shannon’s Entropy of each character was computed using Excel.

Shannon’s Entropy Formula:

$$\boldsymbol{H(X) = -\sum p(x)\log_2 p(x)}$$

Where p(x) is the probability of occurrence of each symbol.

Probabilities were computed as:

$$p(x) = \frac{frequency\ of\ character}{Total\ character\ frequency}$$

In the Table 2 below, H(x) is the individual character entropy,

$$H(x) = -\,p(x)\log_2 p(x)$$

Character	p(x)	H(x)	Character	p(x)	H(x)
e	0.1006	0.3333	**v**	0.0058	0.0431
n	0.0793	0.2900	**w**	0.0058	0.0431
a	0.0774	0.2857	**x**	0.0019	0.0174
o	0.0716	0.2723	**S**	0.0077	0.0543
t	0.0600	0.2434	**T**	0.0058	0.0431
i	0.0464	0.2056	**C**	0.0039	0.0310
r	0.0445	0.1998	**A**	0.0019	0.0174
s	0.0406	0.1877	**M**	0.0019	0.0174
h	0.0387	0.1815	**1**	0.0019	0.0174
c	0.0368	0.1752	**4**	0.0019	0.0174
d	0.0309	0.1552	**8**	0.0019	0.0174

m	0.0290	0.1482	**9**	0.0019	0.0174
f	0.0213	0.1182	**Space**	0.1605	0.4237
l	0.0193	0.1101	**"**	0.0077	0.0543
u	0.0193	0.1101	**,**	0.0077	0.0543
y	0.0174	0.1017	**.**	0.0058	0.0431
p	0.0155	0.0931	**–**	0.0039	0.0310
b	0.0135	0.0840	**’**	0.0019	0.0174
g	0.0058	0.0431	**:**	0.0019	0.0174

Table 2: Task 2 - Probability & Entropy Contribution

Total Entropy H(X) = 4.3159 bits per character

4) Results and Interpretation

Entropy Value and It’s Significance

The Shannon entropy of the selected English text dataset was calculated to be 4.3159 bits per character. This value quantifies the average amount of information contained in each character of the text. In Information Theory, entropy reflects the degree of unpredictability or uncertainty in a dataset. A value close to 0 would indicate extreme redundancy (e.g., repeated characters), while a value approaching 5-8 bits would suggest high uncertainty, typical of encrypted or unstructured data.

An entropy of 4.3159 indicates that the text is moderately predictable, consistent with structured English writing. It contains enough variation to convey meaningful content without excessive repetition, making it suitable for efficient encoding and compression.

Predictability Analysis Using Letter Frequency Comparison

To critically interpret the entropy value, a comparison was made between the standard English letter frequencies (as published on Wikipedia (contributors, 2025) [3]) and the letter frequencies from the text dataset. Prior to analysis, all uppercase letters in the dataset were converted to lowercase to ensure consistency and alignment with the standard frequency reference, which is case-insensitive. This normalization step reduces alphabet size and stabilizes frequency distributions, allowing for a more accurate entropy calculation. The comparison was visualized using a line chart (generated in python) which is illustrated below in Figure 1.

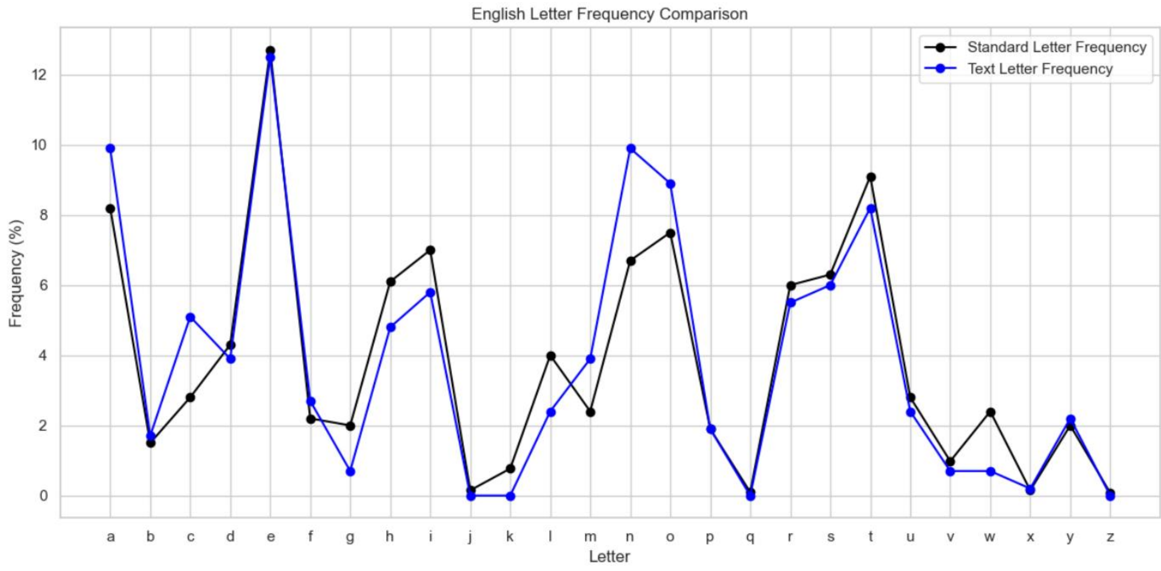

Figure 1: English Letter Frequency Comparison Line Chart

Key Observations: The letter ‘e’ has the highest frequency in both datasets, appearing slightly above 12%. This strong alignment reinforces the text’s adherence to natural English structure. Other high-frequency letters such as ‘t’, ‘a’, ‘o’, and ‘n’ also show similar proportions across both datasets, indicating that the text follows conventional linguistic patterns. Low-frequency letters like ‘q’, ‘x’, and ‘z’ remain rare in both cases, contributing minimally to the entropy.

Fluctuations and Their Impacts: Slight increases in the frequency of letters such as ‘c’, ‘m’, ‘n’, and ‘o’ suggest the presence of domain-specific vocabulary, likely due to technical nature of the text (e.g., terms like “communication”, “entropy”, and “Shannon”). These fluctuations introduce mild unpredictability, enriching the information content without disrupting the overall structure.

Interpretation: The close alignment between the two frequency curves confirms that text is linguistically typical, supporting the entropy value of 4.3159. The minor fluctuations may assume informational depth, but not uncertainty which resulting a text that is predictable enough for efficient encoding yet varied enough to avoid redundancy.

Impact of Non-Letter Characters

The inclusion of digits and punctuation marks though less frequent adds to the character diversity. These elements introduce less predictable symbols, which slightly increase the entropy.

5) Critical Reflection on Information Content

The calculated entropy value of 4.3159 bits per character suggests that the text is efficiently encoded, with a good balance between redundancy and novelty. It is neither overly repetitive nor excessively random.

Furthermore, the entropy reflects the clarity and coherence of the text. It is informative without being noisy, making it suitable for both human reading and machine processing.

By extension, this observation suggests that Wikipedia tends to exhibit moderate predictability and rich information content. Its editorial standards and structured language contribute to a character distribution that is neither overly uniform nor chaotic, aligning with the entropy characteristics observed in the text dataset. This reinforces Wikipedia’s role as a reliable and computationally tractable source for tasks involving compression, encoding, and information analysis.

Task 2: Huffman Coding

Huffman coding is a fundamental lossless data compression technique that assigns variable-length codes to characters based on their frequencies. Characters that occur more frequently receive shorter codes, while less frequent characters are assigned longer codes.

In this task, the character frequency data obtained in Task 1 was used to construct a Huffman tree, generate Huffman codes, and encode the selected text. The efficiency of Huffman coding is then evaluated by comparing the encoded text length with the theoretical entropy and fixed-length ASCII encoding.

1) Huffman Tree Construction

The Huffman tree was constructed using the character frequencies derived from Task 1, which analyzed a 517-character English text excerpt. Python was used to implement the Huffman coding algorithm, ensuring that each character was assigned a binary code based on its relative frequency.

To enhance clarity and visual interpretation, the tree was separated into two distinct subtrees:

- **Left Subtree:** Represents all codes beginning with 0.
- **Right Subtree:** Represents all codes beginning with 1.

This bifurcation was not algorithmically necessary but was intentionally applied to improve the readability of the tree structure and to clearly demonstrate the hierarchical placement of each node.

The sum of the highest- frequency nodes from both subtrees equals 517, which corresponds to the total number of characters in the dataset. This confirms that the tree accurately encapsulates the full character distribution and preserves the integrity of the original dataset.

Frequent characters such as the space and ‘e’ appear closer to the root, resulting in shorter Huffman codes, while rarer characters are placed deeper in the tree, receiving longer codes. This structure ensures optimal compression efficiency, aligning with the principles of prefix coding.

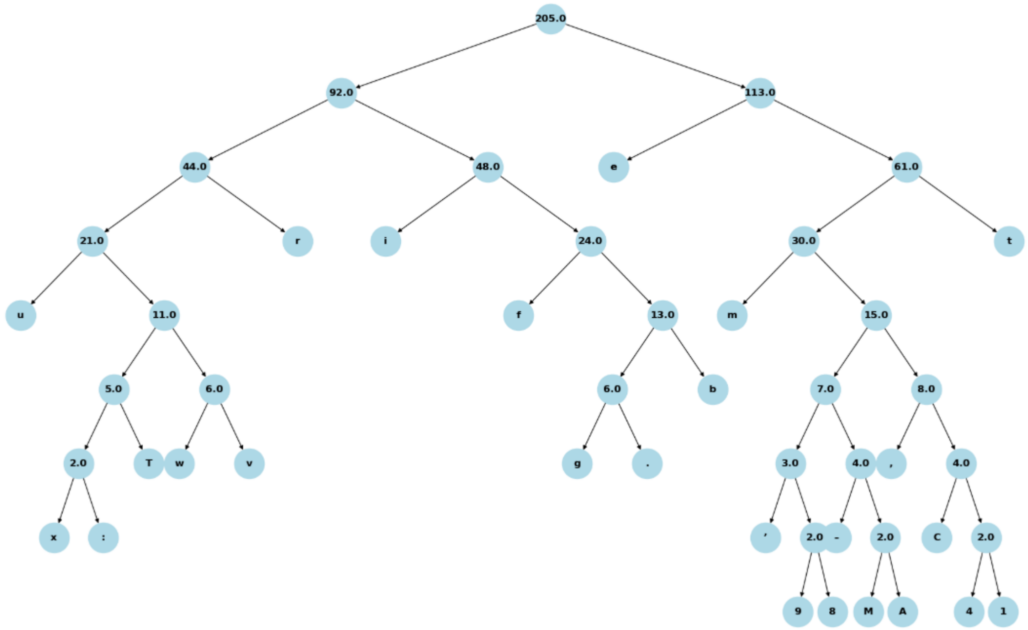

Figure 2: Task 2 - Huffman Tree - Left Subtree (0-side)

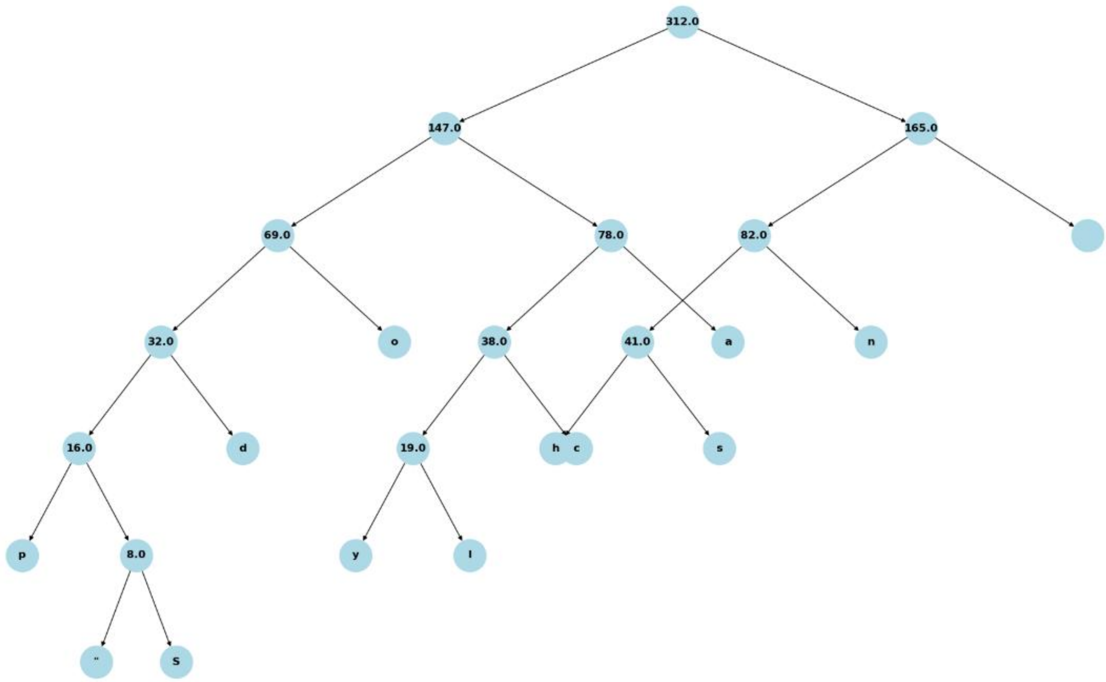

Figure 3: Task 2 - Huffman Tree - Right Subtree (1-side)

For better visualization, a clear Huffman tree was constructed using draw.io. The complete tree is provided in Annex A (Figure A1) for reference.

2) Huffman Codes

The Huffman coding algorithm generated the following codes for the dataset characters as in Table 3 using python.

Character	Frequency	Huffman Code	Character	Frequency	Huffman Code
Space	83	111	**S**	4	1000011
e	52	010	**"**	4	1000010
n	41	1101	**,**	4	0110110
a	40	1011	**g**	3	0011100
o	37	1001	**v**	3	0000111
t	31	0111	**w**	3	0000110
i	24	0010	**T**	3	0000101
r	23	0001	**.**	3	0011101
s	21	11001	**C**	2	01101110
h	20	11000	**–**	2	01101010
c	19	10101	**x**	1	0000100
d	16	10001	**A**	1	011010111
m	15	01100	**M**	1	011010110
f	11	00110	**1**	1	011011111
l	10	101001	**4**	1	011011110
u	10	00000	**8**	1	011010011
y	9	101000	**9**	1	011010010
p	8	100000	**’**	1	01101000
b	7	001111	**:**	1	00001001

Table 3: Huffman Codes from highest to lowest frequency

3) Encoding the Original Text

The original 517-character text excerpt was encoded using the Huffman codes using python.

- **Original text length (ASCII, fixed length):** 517 × 8 = 4136 bits
- **Encoded text length (Huffman):** 2252 bits

The resulting encoded text is a binary string composed of Huffman codewords for each character; fully encoded output of the original text is in Annex A (Figure A2) for reference.

4) Compression Analysis

Calculations were done using python.

- **Total bits in the encoded message:** 2252 bits
- **Average bits per character:** 4.3559 bits per character
- **Compression ratio compared to fixed-length ASCII encoding (assuming 8 bits per character):**

$$\text{Compression Ratio} = 4136 / 2252 = 1.8366$$

This means the Huffman-encoded message is almost twice as compact as ASCII encoding.

5) Reflection: Comparing Huffman-Encoded Size to Theoretical Entropy

The Shannon entropy calculated in Task 1 was 4.3159 bits per character, representing the theoretical lower bound for any lossless compression scheme based on the character distribution of the given text. After applying Huffman coding, the encoded text resulted in a total of 2252 bits, with an average of 4.3559 bits per character.

This outcome aligns closely with theoretical expectations. Huffman coding is known to produce an average code length that satisfies the inequality:

$$H \leq L \leq H + 1$$

Where H is the Shannon entropy and L is the average number of bits per character in the encoded text (Ayfer Ozgur, 2024) [1].

$$4.3159 \leq 4.3559 \leq 5.3159$$

In this case, the Huffman average is only 0.04 bits per character above the entropy, which is a minimal overhead. This slight increase is due to Huffman’s use of integer length codewords and the finite sample size of the dataset. Rare characters such as punctuations typically receive longer codewords, contributing to the marginal difference.

Overall, the Huffman encoding demonstrates near optimal performance for the given character distribution. The result confirms that Huffman coding is highly efficient for symbol-by-symbol compression and closely tracks the theoretical entropy, with only minor deviation due to structural constraints of prefix coding.

Task 3: Hamming Code (Error Detection and Correction)

Hamming codes, introduced by Richard Hamming in 1950, are a family of linear error-correcting codes that add redundant parity bits to transmitted data. Their main purpose is to detect and correct single-bit errors and, in some cases, detect two-bit errors.

In this task, the Hamming (7, 4) scheme is applied to a short binary segment derived from the Huffman-encoded text in Task 2. The report demonstrates how single-bit errors are corrected, why two-bit errors cause problems, and concludes with a comparison between Huffman coding and Run-Length Encoding (RLE).

1) Selected Segment and Setup

- **Selected 8-bit segment:** 10011011 (Random)
- **Nibble split:** 1001 and 1011
- **Encoding scheme:** Hamming (7, 4) with even parity
- **Bit layout:** positions 1, 2, 4 are parity; data bits at positions 3 = d1, 5 = d2, 6 = d3, 7= d4
- **Parity coverage:**

Bit position	7	6	5	4	3	2	1
Bit Role	**d4**	**d3**	**d2**	**p4**	**d1**	**p2**	**p1**
Bit position in binary	0111	0110	0101	0100	0011	0010	0001

  - p1: positions {1, 3, 5, 7}
  - p2: positions {2, 3, 6, 7}
  - p4: positions {4, 5, 6, 7}
- **Syndrome bits on receive (s1, s2, s4) are computed under even parity:**
  - s1 = parity (r1, r3, r5, r7)
  - s2 = parity (r2, r3, r6, r7)
  - s4 = parity (r4, r5, r6, r7)

  and error position is $(s4s2s1)_2$ with 0 meaning “no error”

2) Encoding the Two Nibbles

Nibble 1 (1001):

Bit Position	7	6	5	4	3	2	1
Bit Role	**d4**	**d3**	**d2**	**p4**	**d1**	**p2**	**p1**
Parity Bits	1	0	0	1	1	0	0
Parities in the Positions	p1, p2, p3	p2, p3	p1, p3	p3	p1, p2	p2	p1

- p1: positions 1, 3, 5, 7 → _, 1, 0, 1 → even → p1 = 0
- p2: positions 2, 3, 6, 7 → _, 1, 0, 1 → even → p2 = 0
- p4: positions 4, 5, 6, 7 → _, 0, 0, 1 → odd → p3 = 1

Codeword: 1001100

Nibble 2 (1011):

Bit Position	7	6	5	4	3	2	1
Bit Role	**d4**	**d3**	**d2**	**p4**	**d1**	**p2**	**p1**
Parity Bits	1	0	1	0	1	0	1
Parities in the Positions	p1, p2, p3	p2, p3	p1, p3	p3	p1, p2	p2	p1

- p1: positions 1, 3, 5, 7 → _, 1, 1, 1 → odd → p1 = 1
- p2: positions 2, 3, 6, 7 → _, 1, 0, 1 → even → p2 = 0
- p4: positions 4, 5, 6, 7 → _, 1, 0, 1 → even → p3 = 0

Codeword: 1010101

Encoded 14-bit segment: 10011001010101

3) Single-Bit Error: Detection and Correction

- **Error introduced:** bit flipped at position 5 in first codeword.
  - **Original:** 1001100
  - **Received:** 1011100

Syndrome Calculation

7	6	5	4	3	2	1
1	0	1	1	1	0	0

- s1: parity of positions 1, 3, 5, 7 → 0, 1, 1, 1 → odd → s1 = 1
- s2: parity of positions 2, 3, 6, 7 → 0, 1, 0, 1 → even → s2 = 0
- s3: parity of positions 4, 5, 6, 7 → 1, 1, 0, 1 → odd → s3 = 1

Syndrome: 101 → binary 5

Correction

Bit at position 5 flipped back.

Corrected codeword: 1001100

In conclusion, single-bit errors are reliably detected and corrected using the syndrome.

4) Two-Bit Error: Undetected and Miscorrected

- **Error introduced:** bit flipped at positions 2 and 6 in second codeword.
  - **Original:** 1010101
  - **Received:** 1110111

Syndrome Calculation

7	6	5	4	3	2	1
1	1	1	0	1	1	1

- s1: parity of positions 1, 3, 5, 7 → 1, 1, 1, 1 → even → s1 = 0
- s2: parity of positions 2, 3, 6, 7 → 1, 1, 1, 1 → even → s2 = 0
- s3: parity of positions 4, 5, 6, 7 → 0, 1, 1, 1 → odd → s3 = 1

Syndrome: 001 → binary 1

This incorrectly suggests an error at position 1, even though the actual errors are at positions 2 and 6.

Correction Attempt

Decoder flips bit at position 1.

Miscorrection: 1110110 ≠ 1010101

In conclusion, two-bit errors produce misleading syndromes that point to a third bit, resulting in incorrect correction, where the original data is corrupted.

5) Discussion on Best Suited Data Types for Huffman Coding and LZW

Huffman Coding and Lempel-Ziv-Welch (LZW) are both lossless compression techniques, yet they differ fundamentally in their approach to redundancy and data structure. Understanding these differences is essential when selecting the appropriate method for a given dataset.

Huffman coding is a statistical compression algorithm that assigns variable-length codes to input symbols based on their frequency. As (Sayood, 2018) [6], it is optimal when the symbol probabilities are known and data source is memoryless (no memory of past outputs). The technique constructs a binary tree where more frequent symbols receive shorter codes, minimizing the average code length.

This method is particularly effective for datasets with skewed symbol distribution, such as English text, grayscale images, or sensor logs. Its performance closely aligns Shannon entropy, making it ideal for entropy-aware encoding tasks. However, Huffman coding requires a preprocessing step to calculate character frequencies, which can be a limitation in streaming or real-time applications.

LZW, in contrast, is a dictionary-based algorithm that builds a codebook dynamically during encoding. It does not require prior knowledge of symbol frequencies and instead compresses data by referencing previously seen substrings. (Sayood, 2018) [6] highlights LZW's adaptability, noting its strength in handling data with repeated patterns such as source code, markup languages, and structured logs.

LZW is particularly well suited for medium or large datasets where patterns emerge over time. Its single pass encoding makes it efficient for streaming scenarios, and its use in formats like GIF and TIFF underscores its practical relevance. However, LZW may be less effective on short or highly random data, where dictionary growth is limited.

Feature	Huffman Coding	LZW Compression
Compression Type	Statistical	Dictionary-based
Optimization Target	Symbol frequency	Pattern repetition
Preprocessing Required	Yes	No
Adaptability	Static (unless extended)	Highly adaptive
Best Data Types	Skewed text, grayscale images	Source code, markup, structured logs
Entropy Alignment	Direct (Shannon entropy)	Indirect (pattern entropy)

Table 4: Comparison of Features is Huffman Coding & LZW

Both techniques offer distinct advantages depending on the structure and entropy profile of the input data. Huffman coding excels in symbol-level optimization, while LZW thrives in pattern-rich environments. The choice between them should be guided by the dataset’s characteristics and operational constraints such as whether preprocessing is feasible or streaming efficiency is required.

Task 4: Entropy in a Non-Latin Language

Information entropy can be applied across different languages and scripts to measure the unpredictability and information density of text. While English uses the Latin alphabet with relatively simple character composition, non-Latin scripts such as Sinhala introduce unique challenges due to complex grapheme clusters, diacritics, and joining mechanisms.

This task analyses the entropy of a Sinhala text excerpt using Shannon's entropy formula, compares the results with the English dataset from Task 1, and reflects on the challenges of working with non-Latin scripts.

1) Dataset Selection

The selected dataset is a short excerpt from the Wikipedia article "Sri Lanka" (contributors, 2025) [5].

ශ්‍රී ලංකාවේ විශාලත්වය වර්ග කි.මී 65,610 (වර්ග සැතපුම් 25,330) වේ. එය ලොව 123 වැනි විශාලතම රට වන අතර විශාලතම දූපත් අතරින් 25 වන ස්ථානයෙ සිටියි. එහි භූමි ප්‍රදේශය පළාත් නවයකට සහ දිස්ත්‍රික්ක විසිහතරකට වෙන් කර ඇත. රට පළාත් 9 කින්, දිස්ත්‍රික්ක 25 හා ප්‍රදේශිය ලේකම් කොට්ටාශය 331 හා ග්‍රාම නිලධාරි කොට්ටාශ 14,022 කින් සමන්විත වේ. අගනුවර ශ්‍රී ජයවර්ධනපුර කොට්ටෙ වන අතර විශාලතම නගරය කොළඹ වේ. කොළඹ ජාතියේ ආර්ථික, දේශපාලන කේන්ද්‍රස්ථානයයි. ශ්‍රී ලංකාව ලෝක උරුම ස්ථාන අටක්, සැතපුම් ගණනක් පුරා විහිදුනු රන්වන් වැලි සහිත වෙරළවල්, වැසි වනාන්තර සහ කඳුකර තේ වතු සහිත ඉතා සුන්දර දූපතකි. නිවර්තන වනාන්තර, වෙරළ සහ භූ දර්ශනවල ස්වභාවික සුන්දරත්වය, ජෛව විවිධත්වය මෙන්ම පොහොසත් සංස්කෘතික උරුමයන් සඳහා ප්‍රසිද්ධය, එය ලෝක ප්‍රසිද්ධ සංචාරක ගමනාන්තයක් බවට පත් කළේය.

The non-Latin dataset (Sinhala script) was encoded using UTF-8, a variable-length character encoding capable of representing all Unicode characters. UTF-8 ensures compatibility across platforms and preserves the integrity of Sinhala graphemes, including conjunct forms and diacritics. Sinhala text differs from English in that may visible characters (syllables) are formed by combining multiple Unicode code points (e.g., consonant + virama + ZWJ + another consonant). This encoding was explicitly set during file reading and verified using python's chardet and manual inspection.

It consists of letters, spaces, punctuation, digits and Zero Width Joiners (ZWJ) (contributors, 2025) [4].

2) Character Frequency Calculation

The frequency of each unique character was calculated using Excel, except for ZWJ, which was calculated using python. The total length of the Sinhala dataset is 741 characters.

Character	Frequency	Character	Frequency
ව	43	◌්	61
ර	40	◌ි	41
න	36	◌ා	27
ක	33	◌ු	13
ත	33	◌ේ	13
ස	26	◌ො	8
ය	20	◌ැ	5
ප	15	◌ෙ	5
ම	15	◌ී	4
ල	15	◌ූ	4
ද	14	◌ං	4
ට	13	◌ෝ	2
ශ	13	◌ෘ	1
හ	12	◌ෛ	1
ග	7		
ළ	7	**0**	3
අ	5	**1**	4
ඩ	5	**2**	6
ථ	4	**3**	5
එ	3	**4**	1
ජ	3	**5**	4
භ	3	**6**	2
උ	2	**9**	1
ඳ	2		
ඹ	2	**(**	1
ආ	1	**)**	1
ඉ	1	**,**	10
ඇ	1	**.**	9
ච	1		
ඤ	1	**Space**	117
බ	1	**ZWJ**	11

Table 5: Task 4 - Character Frequencies

3) Computing Shannon's Entropy

The Shannon's Entropy of each character was computed using Excel, following the same method as in Task 1.

Character	p(x)	H(x)	Character	p(x)	H(x)
ව	0.0580	0.2383	◌්	0.0823	0.2966
ර	0.0540	0.2273	◌ි	0.0553	0.2310
න	0.0486	0.2120	◌ා	0.0364	0.1741
ක	0.0445	0.1999	◌ු	0.0175	0.1023
ත	0.0445	0.1999	◌ේ	0.0175	0.1023
ස	0.0351	0.1696	◌ො	0.0108	0.0705
ය	0.0270	0.1407	◌ැ	0.0067	0.0487
ප	0.0202	0.1139	◌ෙ	0.0067	0.0487
ම	0.0202	0.1139	◌ී	0.0054	0.0407
ල	0.0202	0.1139	◌ූ	0.0054	0.0407
ද	0.0189	0.1082	◌ං	0.0054	0.0407
ට	0.0175	0.1023	◌ෝ	0.0027	0.0230
ශ	0.0175	0.1023	◌ෘ	0.0013	0.0129
හ	0.0162	0.0963	◌ෛ	0.0013	0.0129
ග	0.0094	0.0635			
ළ	0.0094	0.0635	**0**	0.0040	0.0322
අ	0.0067	0.0487	**1**	0.0054	0.0407
ඩ	0.0067	0.0487	**2**	0.0081	0.0563
ථ	0.0054	0.0407	**3**	0.0067	0.0487
එ	0.0040	0.0322	**4**	0.0013	0.0129
ජ	0.0040	0.0322	**5**	0.0054	0.0407
භ	0.0040	0.0322	**6**	0.0027	0.0230
උ	0.0027	0.0230	**9**	0.0013	0.0129
ඳ	0.0027	0.0230			
ඹ	0.0027	0.0230	**(**	0.0013	0.0129
ආ	0.0013	0.0129	**)**	0.0013	0.0129
ඉ	0.0013	0.0129	**,**	0.0135	0.0838
ඇ	0.0013	0.0129	**.**	0.0121	0.0773
ච	0.0013	0.0129			
ණ	0.0013	0.0129	**Space**	0.1579	0.4205
බ	0.0013	0.0129	**ZWJ**	0.0148	0.0902

Table 6: Task 4 - Probability & Entropy Contribution

Total Entropy H(X) = 4.8561 bits per character

4) Results and Interpretation

Entropy Value: The calculated entropy of the Sinhala dataset is 4.8561 bits per character which is 0.5402 bits higher than the English dataset analyzed in Task 1 (4.3159 bits per character). This increase reflects the greater unpredictability and diversity of symbols in Sinhala text.

Feature	Sinhala	English
Alphabet Size	60+ distinct grapheme components (base letters, vowel signs, diacritics, ZWJ)	26 base letters; 52 with case sensitivity
Character Distribution	More balanced across characters; includes rare graphemes and modifiers	Skewed toward frequent letters like ‘e’, ‘t’, ‘a’
Script Complexity	Dense encoding; syllables formed by combining multiple components	Simple encoding: characters are independent units

Table 7: Sinhala & English Script Comparison

The broader and more balanced distribution in Sinhala contributes to a higher entropy value, as each character carries more information on average. Although, the visual length of Sinhala text may appear shorter, the underlying encoding is denser and more complex.

5) Challenges Faced

Issued Identified: During initial character frequency analysis in Excel, a discrepancy was observed between the total Unicode code points and the visible character count. Specifically,

- Total string length: 741
- Characters tallied: 730

This mismatch raised concerns about the accuracy of entropy calculations.

Root Cause: The Sinhala text uses Zero Width Joiner (ZWJ, U+200D) to form request conjunct formation and half-forms (ligatures) (e.g., rakaransaya and yansaya). These joiners are:

- Invisible in rendered text
- Counted as individual code points
- Essential for accurate grapheme formation

Thus, the discrepancy stemmed from treating grapheme clusters as Sinhala characters, while the encoding counted each constituent code point.

Resolution Strategy: To resolve the issue:

- Developed a Python script to explicitly detect ZWJ (U+200D) and other special Unicode marks.
- Identified 11 ZWJ characters, reconciling the count between code points and visible graphemes.

This confirmed that grapheme clusters ≠ code points, and that entropy analysis must be based on code point-level frequency, not visual character count.

Insight Gained: This challenge highlighted a critical nuance in non-Latin text analysis:

Unicode-aware processing is essential for accurate entropy computation in languages like Sinhala, where visual characters are often composites of multiple code points.

It also reinforced the importance of distinguishing between rendered text and encoded data, especially when working with compression, entropy, or linguistic diversity.

Annex

Annex A: Huffman Tree Visualization & Huffman Encoded Text

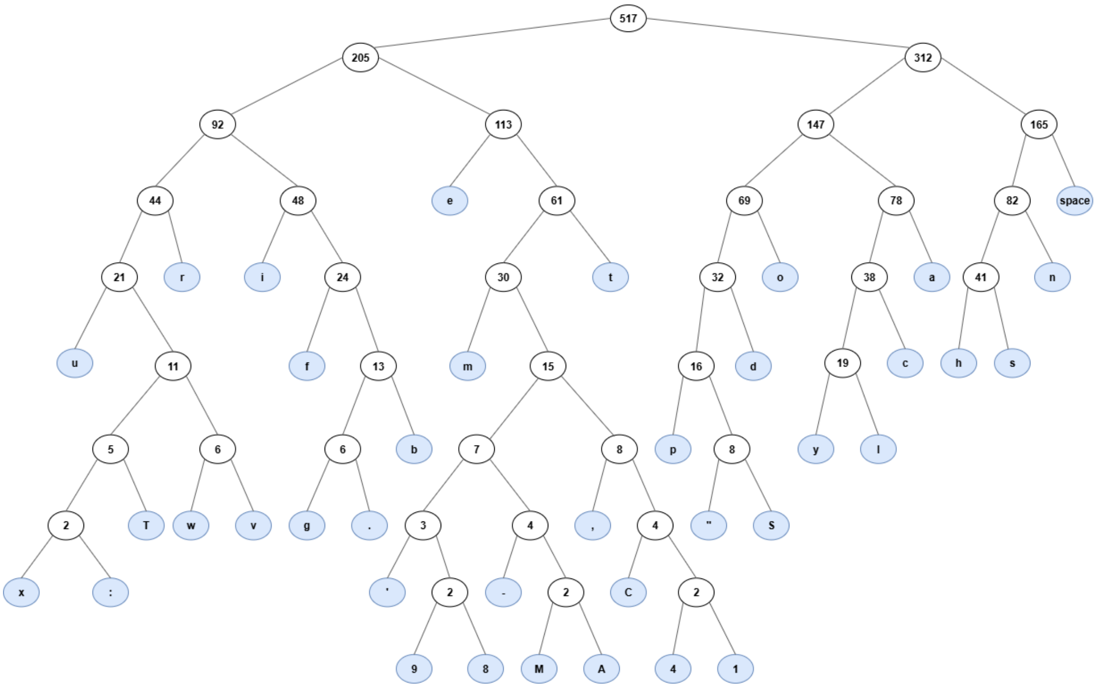

Figure A1: Full Huffman Tree generated using draw.io

```
Original text: The concept of information entropy was introduced by Claude Shannon in his 1948 p
aper "A Mathematical Theory of Communication", and is also referred to as Shannon entropy. Shann
on’s theory defines a data communication system composed of three elements: a source of data, a 
communication channel, and a receiver. The "fundamental problem of communication" – as expressed
by Shannon – is for the receiver to be able to identify what data was generated by the source, b
ased on the signal it receives through the channel.
Encoded binary string: 0000101110000101111010110011101101010101000000111111001001101110010110100
011010010001011001011011100101001110111101011010111000110011000001010001110000110101111001111001
011010111000110011000100000101010101000111100111110100011101101110101001101100000100010101111000
011110001011110111011001110111100101101111111000001011001111011011111011010010011011110011010011
1111000000101110000000100001111100001001101011111101101011010110111110000100110010110111001010101
10
111010011110000101110000101001000110100011110010011011101101110100101100011000000011010010101011
011011100101001110110000100110110111101111011000111100101100111110111010011100110011110001010001
100100001000101010001111011110011111011110011111000011110001011110111011001110111101011010111000
110011000001010000011101111100001111000101111011101100111010110100011001111011111000010100100011
010001111000101000110001011010101100111110111111100011011011110111111010110010110001100000001101
0
010101011011011100101001110111111100110100011001011101001100111101011001011001000001001110010101
0
001111100100110111011111000000101001011101010100101001100010110101111100100001001111101111111001
1001000000001101010101111001001101111000110110111101101101101110111111010110010110001100000001
1
010010101011011011100101001110111110101110001011110111010101010010110110111101111011000111110111
1100010101010101000100000111010000100111011110000101110000101111000010001100000011011000110110
11
000101101011110111010011111000000000110010011111010010100110011110010011011110101100101100011000
0
000110100101010110110111001010011101100001011101101010111101111001111010000010001000000001010110
011100101010001111001111101000111100001111000101111011101100111011110110101011100101100111100110
100100011110111110000010111000101010101010001000000111010000111101111001111001111010111101100111
11
010010101110111100111100101000101011010111001000110101000111000011011000101101111111000110110111
101111110000110101111001111001110001011010100001101101110101000111100111110100011101111110000101
11
110011001000000000110101010011011011100111110111100101010001111100110111101111100001011111001001
000111001101101110100111100100111111000101010101010001000000111010110011110111111000000110010000
00
0111001100011101111110000101111010111000101111011101010101010010011101
```

Figure A2: Huffman Encoded Binary Output

References

[1] Ayfer Ozgur, S. U., 2024. *Lecture 4: Entropy and Lossless Compression.* [Online]
Available at: https://web.stanford.edu/class/engr76/lectures/lecture4.pdf
[Accessed 30 08 2025].

[2] contributors, W., 2025. *Entropy (information theory).* [Online]
Available at:
https://en.wikipedia.org/w/index.php?title=Entropy_(information_theory)&oldid=1300594028
[Accessed 30 08 2025].

[3] contributors, W., 2025. *Letter frequency.* [Online]
Available at:
https://en.wikipedia.org/w/index.php?title=Letter_frequency&oldid=1311139883
[Accessed 30 08 2025].

[4] contributors, W., 2025. *Zero-width joiner.* [Online]
Available at: https://en.wikipedia.org/w/index.php?title=Zero-width_joiner&oldid=1267965011
[Accessed 01 09 2025].

[5] contributors, ව., 2025. *ශ්‍රී ලංකාව.* [Online]
Available at:
https://si.wikipedia.org/w/index.php?title=%E0%B7%81%E0%B7%8A%E2%80%8D%E0%B6%BB%E0%B7%93_%E0%B6%BD%E0%B6%82%E0%B6%9A%E0%B7%8F%E0%B7%80&oldid=739740
[Accessed 01 09 2025].

[6] Sayood, K., 2018. *Introduction to Data Compression.* 5th ed. Cambridge: Morgan Kaufmann Publishers.