ARI (Automated Readability measure)

$$4.71\left(\frac{\text{characters}}{\text{words}}\right) + 0.5\left(\frac{\text{words}}{\text{sentences}}\right) - 21.43$$

ARI = 4.71*(characters/words) + 0.5*(words/sentences) - 21.43

This technique uses the average number of characters per word and the average number of words per sentence to calculate a score that corresponds to a specific grade level.

Limitations:
it does not take into account factors such as the complexity of ideas or the use of technical terminology, which can affect a text's overall readability. Therefore, it should be used in conjunction with other measures and human judgement to determine the appropriateness of a text for a particular audience.

Coleman-Liau Index:

$$= (0.0588 \times L) - (0.296 \times S) - 15.8$$

L = average number of letters per 100 words
S = average number of sentences per 100 words

CLI = 0.0588 * L - 0.296 * S - 15.8
This method is based on the average number of characters per word and per sentence to determine the grade level at which the text can be comprehended.

Limitations:

such as not taking into account the complexity of the vocabulary or the structure of the sentences, which can affect the overall readability of a text. Therefore, it should be used in conjunction with other measures and human judgement to assess the appropriateness of a text for a particular audience.

Flesh Reading Ease:

$$206.835 - 1.015 \left(\frac{\text{total words}}{\text{total sentences}} \right) - 84.6 \left(\frac{\text{total syllables}}{\text{total words}} \right)$$

This measure is little bit different from the others, that is more the score more the file is readable

To make sense of a Reading Ease score, a conversion table is needed. For example, a reading score of 60 to 70 is equivalent to a grade level of 8-9 that can be understood by 13 to 15-year-olds.

Flesch Kincaid:

FKRA = (0.39 x ASL) + (11.8 x ASW) - 15.59

$$0.39 \left(\frac{\text{total words}}{\text{total sentences}} \right) + 11.8 \left(\frac{\text{total syllables}}{\text{total words}} \right) - 15.59$$

This method calculates the average number of words per sentence and the average number of syllables per word to determine the grade level at which a reader can comprehend the text.

Advantage

The Flesch-Kincaid measure has several advantages, such as its simplicity and ease of use, as well as its reliability in predicting the readability of a text. It also takes into account both sentence length and syllable count, which are important factors in determining the complexity of a text.

Limitations:
such as not taking into account the complexity of the vocabulary or the structure of the sentences, which can affect the overall readability of a text. Therefore, it should be used in conjunction with other measures and human judgment to assess the appropriateness of a text for a particular audience.

Gunning Fog Index:

$$0.4 \times \left[\left(\frac{\text{total words}}{\text{total sentences}} \right) + 100 \left(\frac{\text{complex words}}{\text{total words}} \right) \right]$$

GFI = 0.4 * ((ASL) + (PW))
This method uses the number of complex words and the average number of words per sentence to estimate the years of education required to understand the text.
Advantage:
The Gunning-Fog Index is useful for evaluating the readability of technical writing and other materials that may contain complex vocabulary or sentence structures. It takes into account both sentence length and the complexity of the vocabulary, making it a more accurate measure of readability than methods that consider only one of these factors.

Limitations:
such as not taking into account the difficulty of understanding idiomatic expressions, cultural references, or other factors that can affect comprehension. Therefore, it should be used in conjunction with other measures and human judgment to assess the appropriateness of a text for a particular audience.

Forecast:

This is designed to analyse the technical documents like training manuals ,forms and surveys.
Grade level = 20 − (N / 10)
Where N = number of single-syllable words in a 150-word sample

Unlike other formulas, it doesn’t rely on complete sentences. It only uses a vocabulary element. It is used for technical training material. It can also be used for surveys, questionnaires or multiple choice tests.

Unlike the many other formulas, it was not designed for school material. It can’t calculate below a fifth-grade level.

Simple Measure of Gobbledygook (Smog):

$$= 3 + \sqrt{\text{polysyllabic count}}$$

SMOG estimates the years of education the average person needs to understand any piece of writing. This is known as the SMOG Grade. McLaughlin suggested calculating this by using a piece which is 30 sentences or longer and doing the following:

- Counting ten sentences near the beginning of the text, 10 in the middle and ten near the end, totalling 30 sentences
- Counting every word with three or more syllables
- Square-rooting the number and rounding it to the nearest 10
- Adding three to this figure

Advantage:
It is mostly useful in Healthcare Industry.

LEXICAL DIVERSITY MEASURE:

Lexical diversity is a measure of how varied the vocabulary is in a piece of text. There are different ways to calculate lexical diversity, but one common method is to use the Type-Token Ratio (TTR).

TTR (type-token ratio):

The formula for TTR is:

TTR = (number of unique words / number of total words) x 100

For example, let's say you have a text with 200 words, and there are 100 unique words in the text. To calculate the TTR:

TTR = (100 / 200) x 100 = 50

This means that the TTR for the text is 50%. In other words, 50% of the words in the text are unique, and the remaining 50% are repeated words.

```
# Define a function to calculate TTR
def calculate_ttr(text):
   # Split the text into words
   words = text.split()

   # Count the total number of words
   total_words = len(words)
```

```
    # Count the number of unique words
    unique_words = len(set(words))

    # Calculate TTR
    ttr = unique_words / total_words * 100

    # Return TTR
    return ttr

# Example usage
text = "The quick brown fox jumps over the lazy dog. The dog barks and the fox runs away."
ttr = calculate_ttr(text)
print("TTR:", ttr)
```

CTTR(Carroll's Corrected TTR):

Carroll's TTR (or corrected TTR) is a variation of the Type-Token Ratio (TTR) that takes into account the sample size of the text. The formula for Carroll's TTR is:

Carroll's TTR = N^V / (N(N-1))

where N is the total number of words (tokens) in the text, and V is the total number of unique words (types) in the text.

```
# Define a function to calculate Carroll's TTR
def calculate_carrolls_ttr(text):
    # Split the text into words
    words = text.split()

    # Count the total number of words
    total_words = len(words)

    # Count the number of unique words
    unique_words = len(set(words))

    # Calculate Carroll's TTR
    carrolls_ttr = (unique_words ** 2) / (total_words * (total_words - 1))

    # Return Carroll's TTR
    return carrolls_ttr

# Example usage
text = "The quick brown fox jumps over the lazy dog. The dog barks and the fox runs away."
carrolls_ttr = calculate_carrolls_ttr(text)
print("Carroll's TTR:", carrolls_ttr)
```

MSTTR (mean segmental type-token ratio) :

Once you have calculated the TTR for each segment, you can then calculate the mean segment TTR by adding up all the TTR values and dividing by the number of segments.

Here is the formula for calculating the mean segment TTR:

Mean segment TTR = (TTR1 + TTR2 + ... + TTRn) / n

Where TTR1, TTR2, ..., TTRn are the TTR values for each segment, and n is the total number of segments.

```
def calculate_msttr(text, segment_size):
    words = text.lower().split()
    ttrs = []
    for i in range(0, len(words), segment_size):
        segment = words[i:i+segment_size]
        types = len(set(segment))
        ttr = types / len(segment)
        ttrs.append(ttr)
    msttr = sum(ttrs) / len(ttrs)
    return msttr
```

MTLD (measure of textual lexical diversity)

The Measure of Textual Lexical Diversity (MTLD) is a measure of lexical diversity that takes into account both the number of unique words in a text and the length of the text
MTLD works by calculating the average length of a text that can be read before the reader encounters a set number of new words. The reader stops when they encounter a word that has not been seen before or a word that has already been seen a set number of times (known as the "criterion").

To calculate MTLD, you start by dividing the text into segments of a fixed length (typically 50 words). You then count the number of unique words in each segment and calculate the TTR (Type-Token Ratio) for that segment. If the TTR for a segment is below the criterion value, the reader moves on to the next segment. If the TTR is above the criterion value, the reader goes back to the beginning of the segment and starts over. The process continues until the entire text has been read.

MTLD is calculated as the average length of the segments that the reader was able to complete before reaching the criterion value. A higher MTLD score indicates greater lexical diversity.

MTLD has several advantages over other measures of lexical diversity. It takes into account both the number of unique words and the length of the text, and it is less affected by the presence of rare words or outliers. It is also relatively easy to use and can be applied to both spoken and written language.

MTLD = (N / S) * V

Where:

N is the total number of words in the text
S is the total number of segments the text is divided into
V is the number of words in the text divided by the average TTR of all the segments

```
def calculate_mtld(text, threshold):
    words = text.lower().split()
    TTR = 0
    TTR_thresh = 0
    word_stretches = []
    stretch_count = 0
    for word in words:
        if word not in string.punctuation:
            TTR = (TTR * len(word_stretches) + 1) / len(word_stretches + [word])
            word_stretches.append(word)
            if TTR < threshold:
                stretch_count += 1
                TTR_thresh = (TTR_thresh * (stretch_count - 1) + TTR) / stretch_count
                TTR = 0
                word_stretches = []
    if word_stretches:
        stretch_count += 1
        TTR_thresh = (TTR_thresh * (stretch_count - 1) + TTR) / stretch_count
    mtld = (len(words) / sum([len(stretch) for stretch in word_stretches])) * TTR_thresh
    return mtld
```

MATTR:

MATTR (Moving-Average Type-Token Ratio) is a measure of lexical diversity that calculates the Type-Token Ratio (TTR) for a moving window of text. It is calculated by dividing the number of unique words in the window by the total number of words in the window.

Here is a Python code that calculates the MATTR for a given text and window size:

```
def calculate_mattr(text, window_size):
    words = text.lower().split()
    ttrs = []
    for i in range(len(words) - window_size + 1):
        window = words[i:i+window_size]
        types = len(set(window))
        ttr = types / len(window)
        ttrs.append(ttr)
    mattr = sum(ttrs) / len(ttrs)
    return mattr
```

Hypergeometric distribution:

The hypergeometric distribution is a discrete probability distribution that describes the probability of k successes in a sample of size n, without replacement, from a finite population of size N that contains exactly K successes. It is often used in statistics to model situations where the sampling is done without replacement, such as in quality control or in voting systems.

The probability mass function of the hypergeometric distribution is given by:

P(X = k) = (K choose k) * (N - K choose n - k) / (N choose n)
where X is the random variable that represents the number of successes in the sample, k is a non-negative integer that represents the number of successes, N is the population size, K is the number of successes in the population, and n is the sample size.

$$P(x|N,m,n) = \frac{\left(\binom{m}{x}\binom{N-m}{n-x}\right)}{\binom{N}{n}}$$

In Python, the scipy.stats module provides a built-in function to calculate the hypergeometric distribution. Here's an example code that uses the hypergeom.pmf() function to calculate the probability mass function of the hypergeometric distribution:

```
from scipy.stats import hypergeom


# Define the parameters

N = 100 # Population size

K = 30 # Number of successes in the population

n = 10 # Sample size

k = 2 # Number of successes in the sample


# Calculate the probability mass function

pmf = hypergeom.pmf(k, N, K, n)


print(f"The probability of {k} successes in a sample of size {n} from a population of size {N} with {K} successes is {pmf:.4f}")
```

Yules's K:

Yule's K is calculated by analyzing the frequency distribution of different words in a text and determining how evenly distributed those words are.

To calculate Yule's K, you first count the total number of words in the text (N). Then, you count the number of different types of words (V). A "type" is a unique word, while a "token" is any instance of a word. For example, if the text contains the sentence "The cat sat on the mat," there are five tokens (words) but only four types (cat, sat, on, the, mat).

Yule's K is then calculated using the following formula:
K = 10^4 * (C - D) / N^2

Where C is the sum of the product of the frequency of each word type (fi) multiplied by itself (fi^2), and D is N minus the sum of the frequency of each word type (fi) squared (Σfi^2).

Yule's K ranges from 0 to infinity, with higher values indicating greater lexical diversity. A text with a low Yule's K score has a high concentration of a few frequently used words, while a text with a high Yule's K score has a more even distribution of words
Here's a Python code that calculates Yule's K for a given text:

```
        import nltk

from nltk import FreqDist

import math



def calculate_yules_k(text):

    words = nltk.word_tokenize(text.lower())

    freq_dist = FreqDist(words)

    frequency_counts = FreqDist(freq_dist.values())

    N = len(words)

    V = len(freq_dist)

    sum_of_squares = 0

    for freq in frequency_counts:

        f = frequency_counts[freq]

        r = math.log(freq)

        sum_of_squares += f * r * r

    k = 10000 * (sum_of_squares - N) / (N * N)

    return k
```

Here's an example Python code that you can use to measure readability and lexical diversity of a single TXT file:

```
import nltk
import textstat
from lexicalrichness import LexicalRichness
import os

# Read the text from a TXT file
filename = "adani_report_2022.txt"
filepath = os.path.join("adani_annual_reports", filename)
with open(filepath, "r") as file:
    text = file.read()

# Tokenize the text into words
words = nltk.word_tokenize(text)

# Calculate the Flesch Reading Ease score
flesch_score = textstat.flesch_reading_ease(text)

# Calculate the Gunning Fog Index
gunning_fog_index = textstat.gunning_fog(text)

# Calculate the lexical diversity using the TTR measure
ttr = len(set(words)) / len(words)

# Calculate the lexical diversity using the MTLD measure
mtld = LexicalRichness(words).mtld(threshold=0.72)

# Print the results
print(f"File: {filename}")
print(f"Flesch Reading Ease score: {flesch_score:.2f}")
print(f"Gunning Fog Index: {gunning_fog_index:.2f}")
print(f"TTR: {ttr:.2f}")
print(f"MTLD: {mtld:.2f}")
```