**Software Project Management**

**Software Measurement:** A measurement is a manifestation of the size, quantity, amount, or dimension of a particular attribute of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process.

It is an authority within software engineering. The software measurement process is defined and governed by ISO Standard.

**Software Measurement Principles**

The software measurement process can be characterized by five activities-

1. **Formulation:**The derivation of software measures and metrics appropriate for the representation of the software that is being considered.
2. **Collection:** The mechanism used to accumulate data required to derive the formulated metrics.
3. **Analysis:** The computation of metrics and the application of mathematical tools.
4. **Interpretation:**The evaluation of metrics results in insight into the quality of the representation.
5. **Feedback:**Recommendation derived from the interpretation of product metrics transmitted to the software team.

**Need for Software Measurement**

Software is measured to:

* Create the quality of the current product or process.
* Anticipate future qualities of the product or process.
* Enhance the quality of a product or process.
* Regulate the state of the project concerning budget and schedule.
* Enable data-driven decision-making in project planning and control.
* Identify bottlenecks and areas for improvement to drive process improvement activities.
* Ensure that industry standards and regulations are followed.
* Give software products and processes a quantitative basis for evaluation.
* Enable the ongoing improvement of software development practices.

**Classification of Software Measurement**

There are 2 types of software measurement:

1. **Direct Measurement:** In direct measurement, the product, process, or thing is measured directly using a standard scale.
2. **Indirect Measurement:** In indirect measurement, the quantity or quality to be measured is measured using related parameters i.e. by use of reference.

**Software Metrics**

A metric is a measurement of the level at which any impute belongs to a system product or process.

Software metrics are a quantifiable or countable assessment of the attributes of a software product. There are 4 functions related to software metrics:

1. **Planning**
2. **Organizing**
3. **Controlling**
4. **Improving**

**Characteristics of software Metrics**

1. **Quantitative:** Metrics must possess a quantitative nature. It means metrics can be expressed in numerical values.
2. **Understandable:** Metric computation should be easily understood, and the method of computing metrics should be clearly defined.
3. **Applicability:** Metrics should be applicable in the initial phases of the development of the software.
4. **Repeatable:** When measured repeatedly, the metric values should be the same and consistent.
5. **Economical:** The computation of metrics should be economical.
6. **Language Independent:** Metrics should not depend on any programming language.

**Types of Software Metrics**

1. **Product Metrics:** Product metrics are used to evaluate the state of the product, tracing risks and undercover prospective problem areas. The ability of the team to control quality is evaluated. Examples include lines of code, cyclomatic complexity, code coverage, defect density, and code maintainability index.
2. **Process Metrics:** Process metrics pay particular attention to enhancing the long-term process of the team or organization. These metrics are used to optimize the development process and maintenance activities of software. Examples include effort variance, schedule variance, defect injection rate, and lead time.
3. **Project Metrics:** The project metrics describes the characteristic and execution of a project. Examples include effort estimation accuracy, schedule deviation, cost variance, and productivity. Usually measures-
   * Number of software developer
   * Staffing patterns over the life cycle of software
   * Cost and schedule
   * Productivity

**Advantages of Software Metrics**

1. Reduction in cost or budget.
2. It helps to identify the particular area for improvising.
3. It helps to increase the product quality.
4. Managing the workloads and teams.
5. Reduction in overall time to produce the product,.
6. It helps to determine the complexity of the code and to test the code with resources.
7. It helps in providing effective planning, controlling and managing of the entire product.

**Disadvantages of Software Metrics**

1. It is expensive and difficult to implement the metrics in some cases.
2. Performance of the entire team or an individual from the team can’t be determined. Only the performance of the product is determined.
3. Sometimes the quality of the product is not met with the expectation.
4. It leads to measure the unwanted data which is wastage of time.
5. Measuring the incorrect data leads to make wrong decision making.

# Zipf’s Law

**Zipf’s law** is an empirical formula discovered by **George Zipf**in 1930s. Zip’s law describes the relationship between the frequency of words in language corpus and their rank in a frequency sorted list. **In this article, we will be diving into the concept of Zipf’s law and its application in natural language processing.**

## What is Zipf’s Law?

[Zipf’s law](https://www.geeksforgeeks.org/python-zipf-discrete-distribution-in-statistics/) is also known as the principle of least effort. In natural language texts, it has been observed that:

* The second most used word appears half as often as the most used word.
* The third most used word appears one-third the number of times the most used word appears, and so on.

Zipf proposed that such a distribution was observed because we tend to frequently use words that we are more comfortable with. We try to communicate as efficiently as possible by putting in the least amount of effort.

*F Auerbach, a German physicist observed the phenomenon concerning population in cities. The second most populous city had half the population of the most populous city. In 1932, Zipf observed a similar distribution of word frequencies in natural language text (English). He proposed a law based on his findings and it began to be known as Zipf’s law. The same kind of relationship was observed in corporation sizes, income of people etc.*

## Mathematical Formulation

Zipf’s Law can be understood intuitively by considering that in any language, there are a few extremely common words (e.g., “the,” “of,” “and”) that are used very frequently, while the vast majority of words are used relatively infrequently. This distribution of word frequencies follows a power-law distribution, where the frequency of a word is proportional to its rank raised to a negative power.

Mathematically, Zipf’s Law can be expressed as:

�(�)=���*f*(*r*)=*rsC*​

where f(r) is the frequency of the word at rank r, C is a constant, and s is the Zipf exponent.

### ****Key concepts and terms:****

* **Zipf exponent:**The exponent in Zipf’s Law equation determines the steepness of the frequency distribution curve. It reflects the degree of inequality in word frequencies.
* **Rank-frequency distribution:**A plot showing the relationship between the rank of words in a language and their frequency of occurrence.

## Example of Zipf’s Law

*Two friends were met by a bear. One climbed a tree, abandoning the other. The other played dead, and the bear left him unharmed.*

When we read the above story we enjoy it because we are humans and English is one of the languages we speak and understand. If I were a computer, I would be looking at a bunch of words waiting to be analyzed statistically.

Let’s do a small experiment. We’ll find out the frequency of words in the above story. It simply means we’ll count the number of times every word appears and arrange them in descending order. To the same table, let’s add a column, Rank. We’ll assign the highest rank(=1) to the word that appears the most and lowest to the one that appears the least number of times.

| **Words** | **Frequency(f)** | **Rank(r)** |
| --- | --- | --- |
| **the** | 3 | 1 |
| **a** | 2 | 2 |
| **bear** | 2 | 3 |
| **other** | 2 | 4 |
| **two** | 1 | 5 |
| **friends** | 1 | 6 |
| **were** | 1 | 7 |
| **met** | 1 | 8 |
| **by** | 1 | 9 |
| **one** | 1 | 10 |
| **climbed** | 1 | 11 |
| **tree** | 1 | 12 |
| **abandoning** | 1 | 13 |
| **played** | 1 | 14 |
| **dead** | 1 | 15 |
| **and** | 1 | 16 |
| **left** | 1 | 17 |
| **him** | 1 | 18 |
| **unharmed** | 1 | 19 |

In the table, “the” has the highest(=1) rank and “unharmed” has the lowest(=19) rank. Also, one quick glance at the table shows that the most common word appears with almost twice the frequency of the second most common word(fthe ≈ 2fa).

To understand it better, let’s write a program to plot a graph with Frequency(f) as a function of Rank(r). ***r*** is plotted along x-axis and ***f*** along y-axis.

## Python Implementation of Zipf’s Law

The code segment demonstrates Zipf’s law by plotting the frequency of words against their ranks in a given text passage. The resulting plot typically shows a curve indicating the inverse relationship between word frequency and rank, as predicted by Zipf’s law. Let’s discuss the code in detail:

1. **Importing necessary libraries:** The code starts by importing the **[matplotlib.pyplo](https://www.geeksforgeeks.org/pyplot-in-matplotlib/" \t "_blank)t** library for plotting and the re library for regular expressions, which is used later to clean the text.
2. **Defining the input text:** The input text is a string containing a passage of text.
3. **Cleaning the text:** The text is converted to lowercase and split into words using the**re.findall**method with a regular expression pattern \b\w+\b, which matches words. This ensures that only words are considered for frequency analysis.
4. **Calculating word frequencies:** The code then iterates over the list of words and creates a dictionary **textDict** to store the frequency of each word.
5. **Sorting the word frequencies:** The textDict dictionary is sorted in descending order based on the word frequencies, and the sorted dictionary is stored in **wordFrequency**.
6. **Creating rank and frequency lists:** Two lists, rank and frequency, are created to store the ranks and frequencies of words, respectively. The rank is simply the index of the word in the sorted dictionary, and the frequency is the corresponding frequency value.
7. **Plotting the Zipfian distribution:** The code uses**plt.plot** to plot the rank on the x-axis and the frequency on the y-axis. The plot is displayed using **plt.show()**.
8. **Labeling the axes and providing a title:** The x-axis is labeled as “Rank(r)”, the y-axis is labeled as “Frequency(f)”, and the plot is given the title “Zipf’s law

Python3

**import** **matplotlib.pyplot** **as** **plt**

**import** **re**

text = "Two friends were met by a bear. One climbed a tree, abandoning the other. The other played dead, and the bear left him unharmed."

*#Convert text to lower case*

text = text.lower()

*#Remove the unwanted characters*

textList = re.split(', | |\. ', text)

textDict = {}

wordFrequency={}

*#Find the frequency of words*

**for** txt **in** textList:

**if** txt **in** textDict.keys():

textDict[txt]+=1

**else**:

textDict[txt]=1

*#Sort the word frequencies in descending order*

wordFrequency = dict(

sorted(

textDict.items(),

key=**lambda** x: x[1],

reverse=**True**)

)

*#Define two lists, rank and frequency*

rank = []

frequency = []

init = 0

*#Assign ranks based on frequencies of words*

**for** freq **in** wordFrequency.values():

init+=1

rank.append(init)

frequency.append(freq)

*#Plot the rank and frequency*

plt.plot(rank,frequency)

*# Labelling the x axis*

plt.xlabel('Rank(r)')

*# Labelling the y axis*

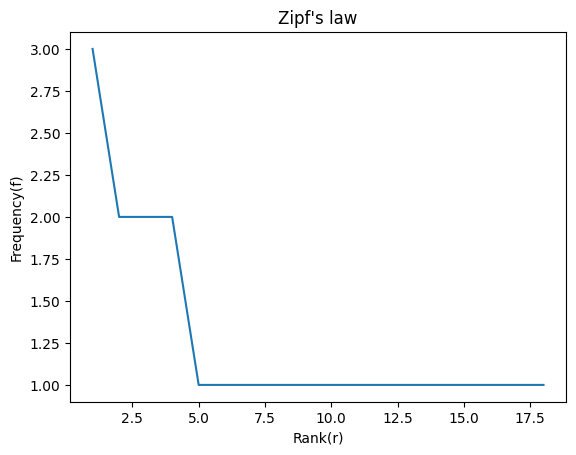
plt.ylabel('Frequency(f)')

*# Providing a title to the graph*

plt.title("Zipf's law")

plt.show()

**Output:**



*Zipf’s Law*

We notice that the plot **roughly** follows the pattern of the reciprocal function **y=1/x**. As x increases, y decreases. If the numerical value of rank is high, the frequency is low.

In this example, we have considered a very small corpus for the purpose of understanding. If the corpus is large, we will get a comparatively smoother curve which will resemble the reciprocal function **y=1/x**.

## Applications

Zipf’s Law has a wide range of applications across various fields. Some key applications include:

1. **Information Retrieval:** In information retrieval systems, Zipf’s Law is used to improve the efficiency of search algorithms by focusing on the most relevant terms.
2. [**Search Engine Optimization (SEO)**](https://www.geeksforgeeks.org/search-engine-optimization-seo-basics/)**:** Understanding the distribution of keywords in content can help optimize websites for search engines, as it allows for the prioritization of important keywords.
3. **Language Modeling:** Zipf’s Law is used in language modeling to predict word frequencies and distributions, which is crucial for tasks like speech recognition and machine translation.
4. **Economics:** Zipf’s Law has been observed in the distribution of income, city sizes, and company sizes, providing insights into economic inequalities and market structures.
5. **Genetics:** Zipf’s Law has been applied in genetics to study the distribution of gene frequencies and mutations in populations.
6. **Network Theory:** In network theory, Zipf’s Law is used to describe the distribution of links or connections in complex networks, such as social networks or the internet.
7. **Urban Planning:** Zipf’s Law has been used in urban planning to understand the distribution of population sizes in cities and to plan infrastructure and services accordingly.

## Deviation from Zipf’s Law

Indeed, deviations from Zipf’s Law are common and can be attributed to various factors. Here are some key points regarding deviations from the law:

1. **Small Percentage of Words Fit the Law:** In large corpora, it’s often observed that only a small percentage of words actually fit the Zipfian distribution. This is because Zipf’s Law describes a general trend rather than an exact rule, and there are always exceptions and variations in real-world data.
2. **Deviation in East Asian Languages:** Many languages of East Asia, such as Chinese, Japanese, and Korean, often deviate significantly from Zipf’s Law, especially at the borders of the rank-frequency distribution. This is attributed to the nature of these languages, which have a large number of homophones (words that sound the same but have different meanings) and complex morphological structures.
3. **Causes of Deviation:** Deviations from Zipf’s Law can occur due to various factors, including the specific characteristics of the language or text, the size of the corpus, and the method of analysis. Other factors such as grammatical structure, word length, and cultural influences can also contribute to deviations.
4. **Implications for Analysis:** When analyzing text data, it’s important to be aware of the potential deviations from Zipf’s Law. While the law provides a useful framework for understanding word frequencies, it’s not a strict rule, and deviations are to be expected in real-world data.