COMPLEX-SYSTEMS-1

October 18, 2024

1 Zipf's law

1.1 Author - Damian Pietroń

1.2 Introduction

Zipf's law is a phenomenon that states that, given a set of occurrences ordered in decreasing order, the value of the n-th element is inversely proportional to its frequency of occurrence.

The mathematical representation of Zipf's law can be written as:

`f(n)  1/n`

where:

- `f(n)` is the frequency of the n-th element,
- `n` is the rank of the element.

1.3 Purpose of this lab

This laboratory aims to explore and gain a deeper understanding of Zipf's Law and the phenomena it describes in large text body. By analyzing the frequency distribution of words in extensive texts, we will observe how Zipf's Law aligns with a frequency distrbution accross text

---

1.4 Code overview

Code for this lab was written both in Python and C++, Algorithmic part was done with C++, but plots were done with Python

[ ]:
```
pip install -r requirements.txt
```

In order to download all neccesarry python libraries run command above (first remember to create virtual environment)

1.5 Import libaries

[187]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
```

---

[188]:
```
# load all csv
path = Path('data')
files = path.glob('*.csv')
```

[189]:
```
# Load the data
data_frames = {}
for file in files:
    file_name = file.name.replace('.csv', '')
    print(file_name)
    data = pd.read_csv(file)
    data_frames[file_name] = data
```

```
bible
divine_comedy
faust_de
LLM_de
plato
_1984
```

1.5.1 Why do I need columns 3 and 4?

- Column 3 (Rank) is needed to calculate Theoritical ZipfLaw
- Column 4 (Frequency) is needed to calculate real frequency of a word

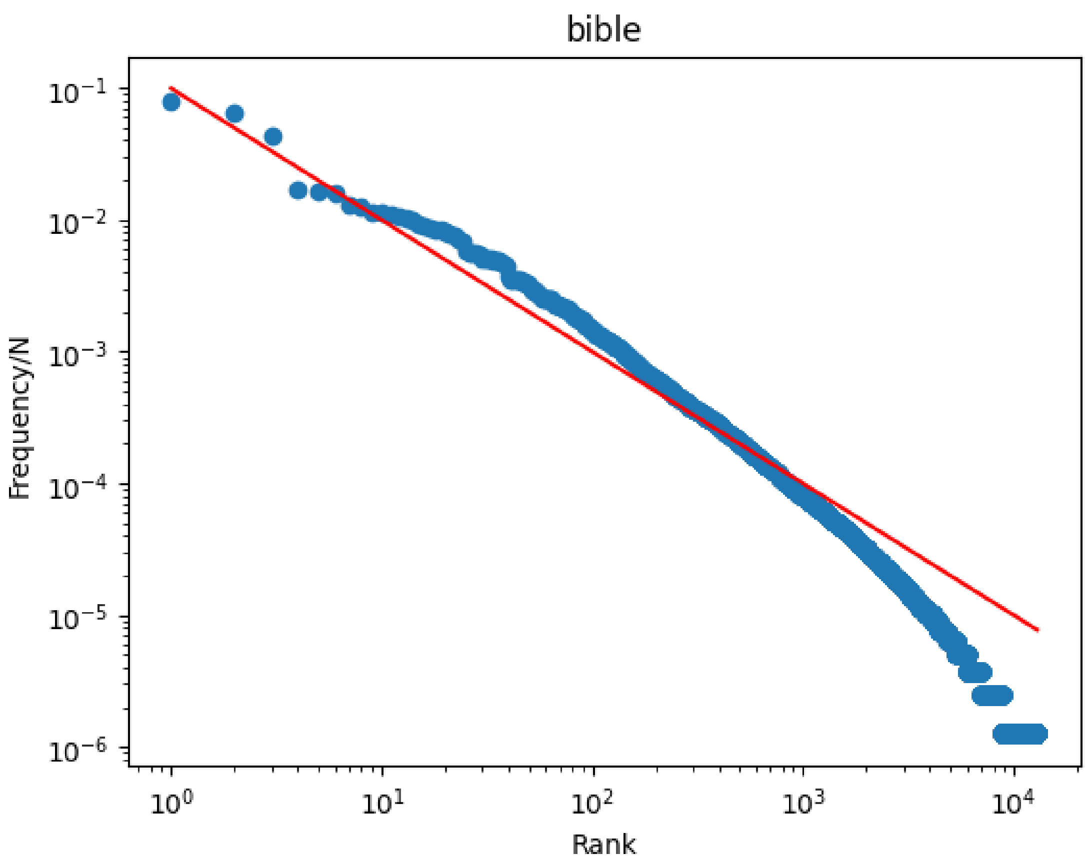
---

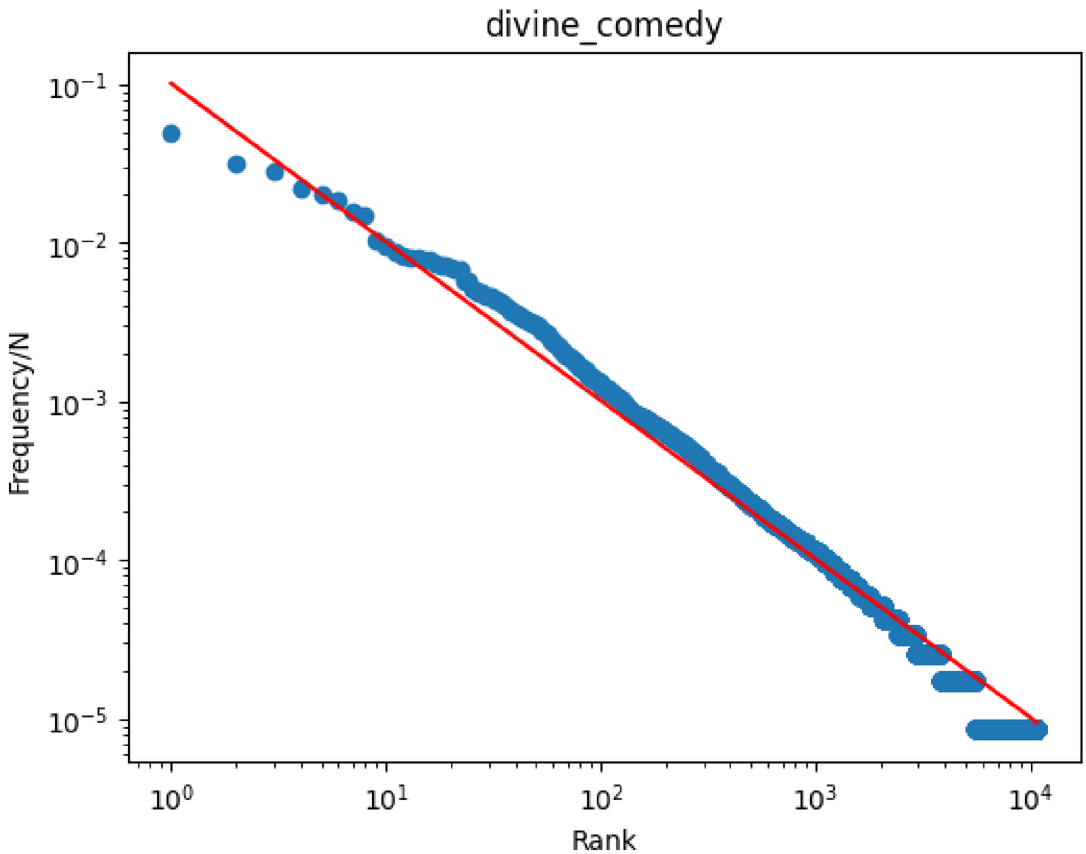
1.6 All books with log to log comparison

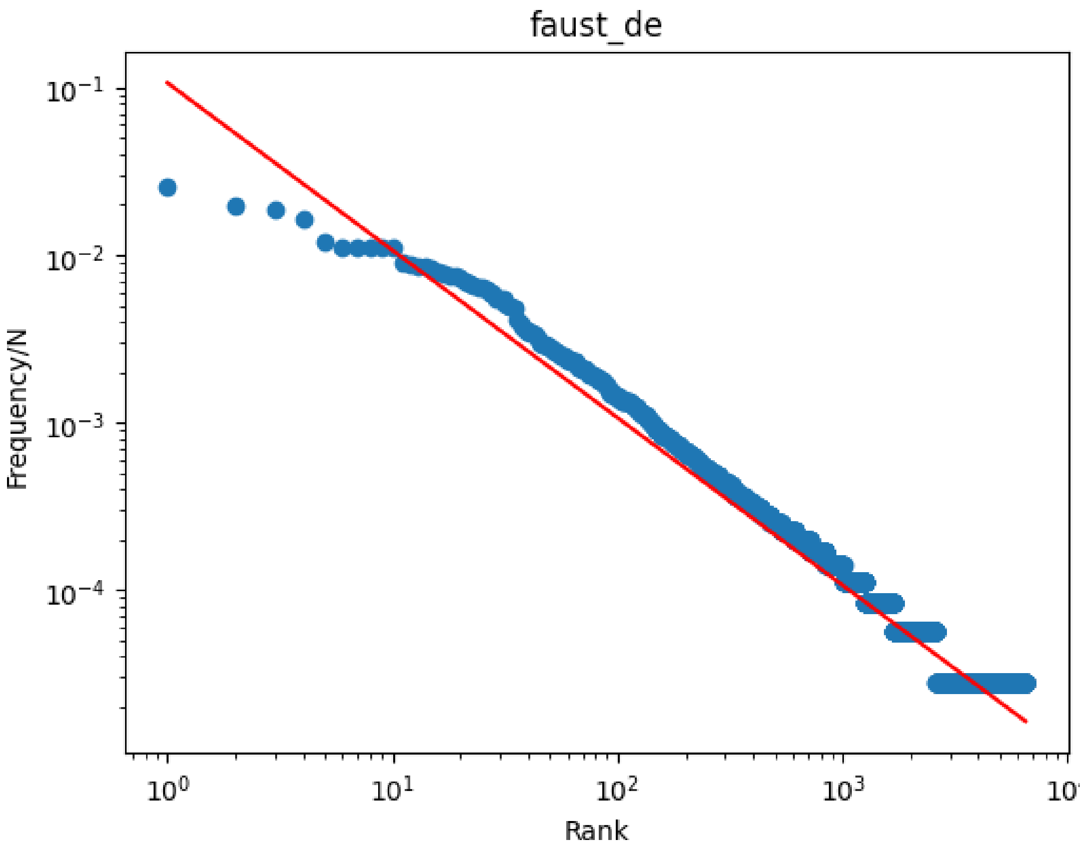
[190]:
```
fig = plt.figure(figsize=(4, 1))
for book_name, book in zip(data_frames.keys(), data_frames.values()):
    plt.figure()
    truncated_data = data_frames[book_name]
    plt.scatter(truncated_data['Rank'], truncated_data['Frequency/N'],␣
 ↪marker='o', label=book)
    plt.plot(truncated_data['Rank'], truncated_data['Zipf'], label='Zipf',␣
 ↪color='red')
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('Rank')
```

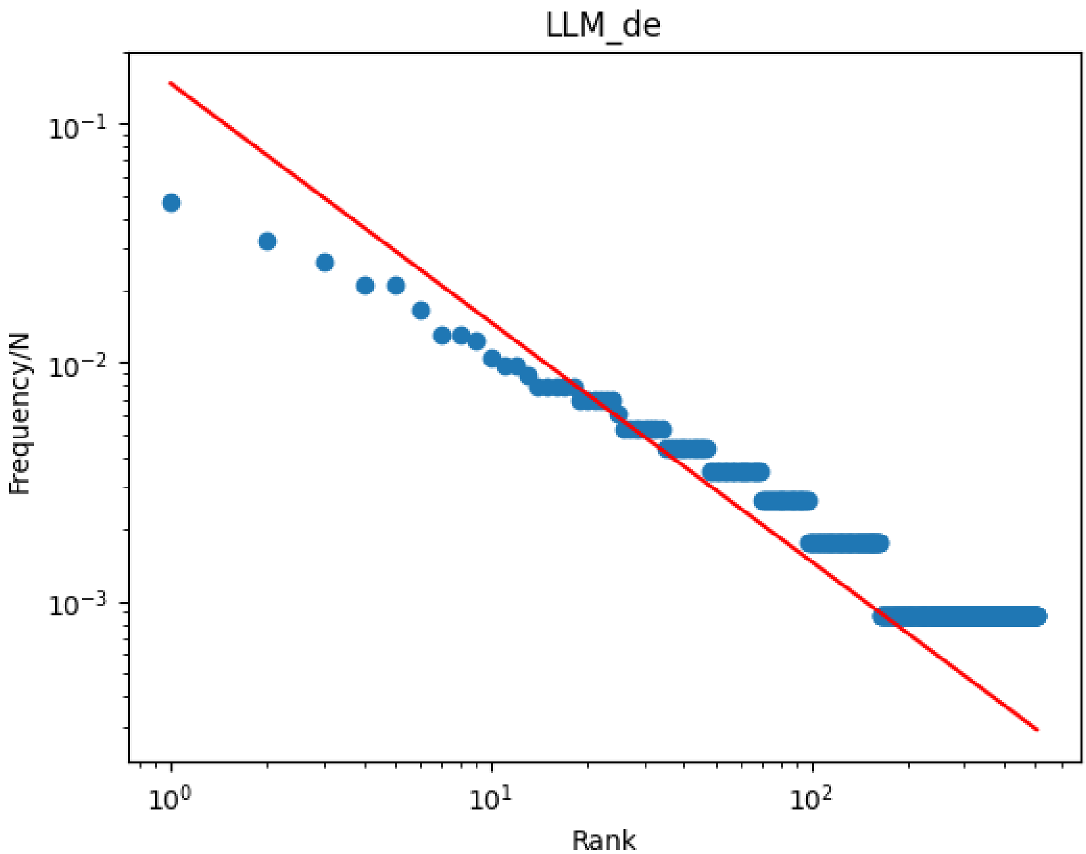
```
    plt.ylabel('Frequency/N')
    plt.title(book_name)
    plt.show()
```

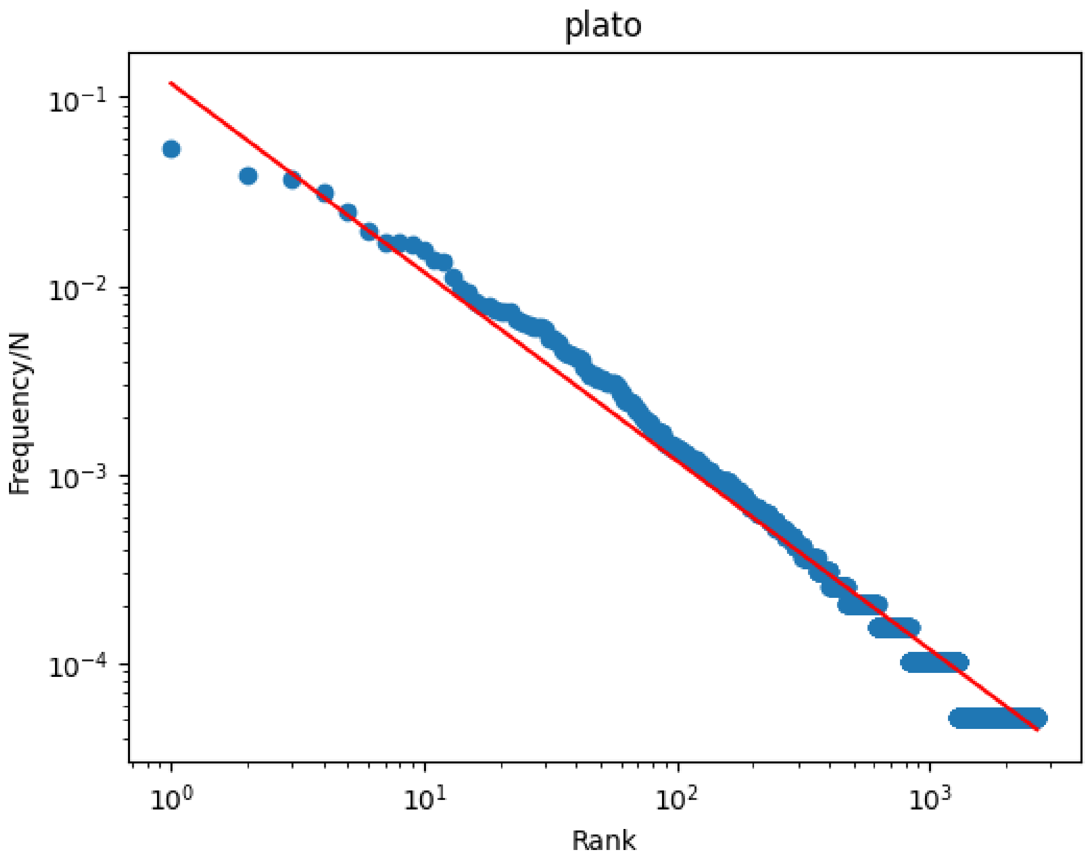
```
<Figure size 400x100 with 0 Axes>
```

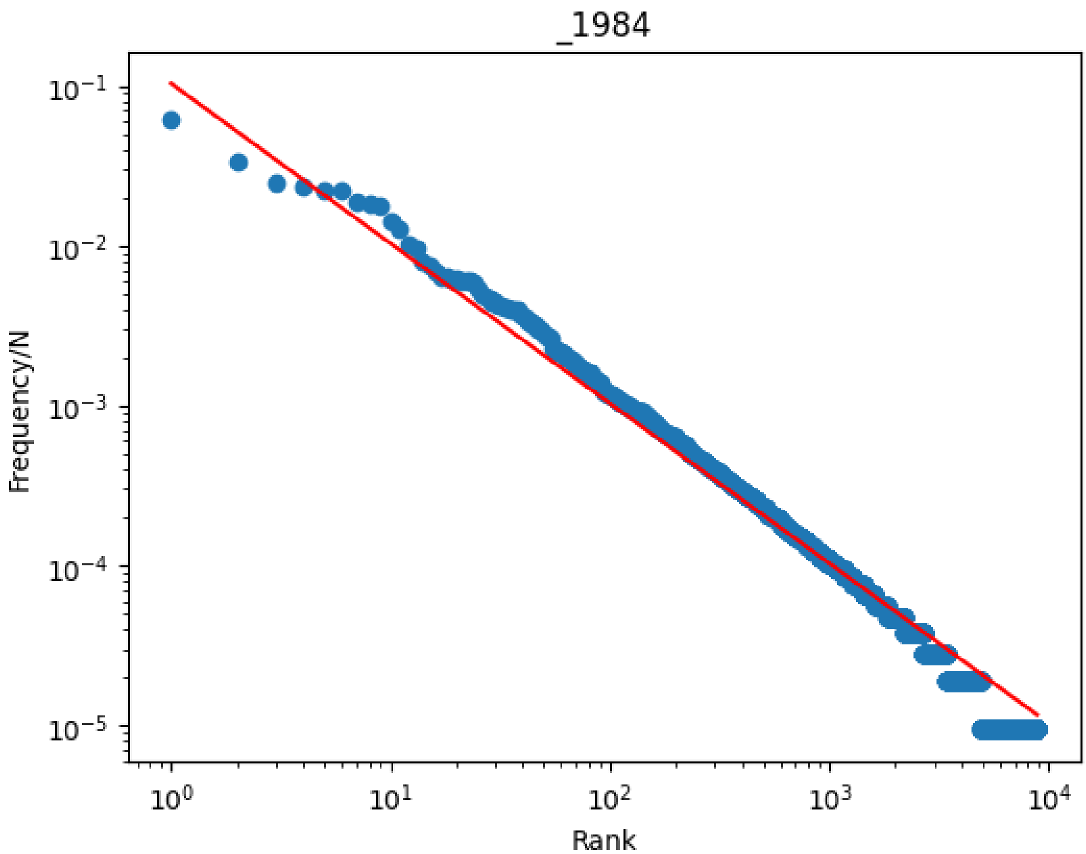



divine_comedy
Frequency/N
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
Rank


faust_de
Frequency/N
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
Rank


LLM_de
Frequency/N
Rank
10^{-1}
10^{-2}
10^{-3}
10^{0}
10^{1}
10^{2}

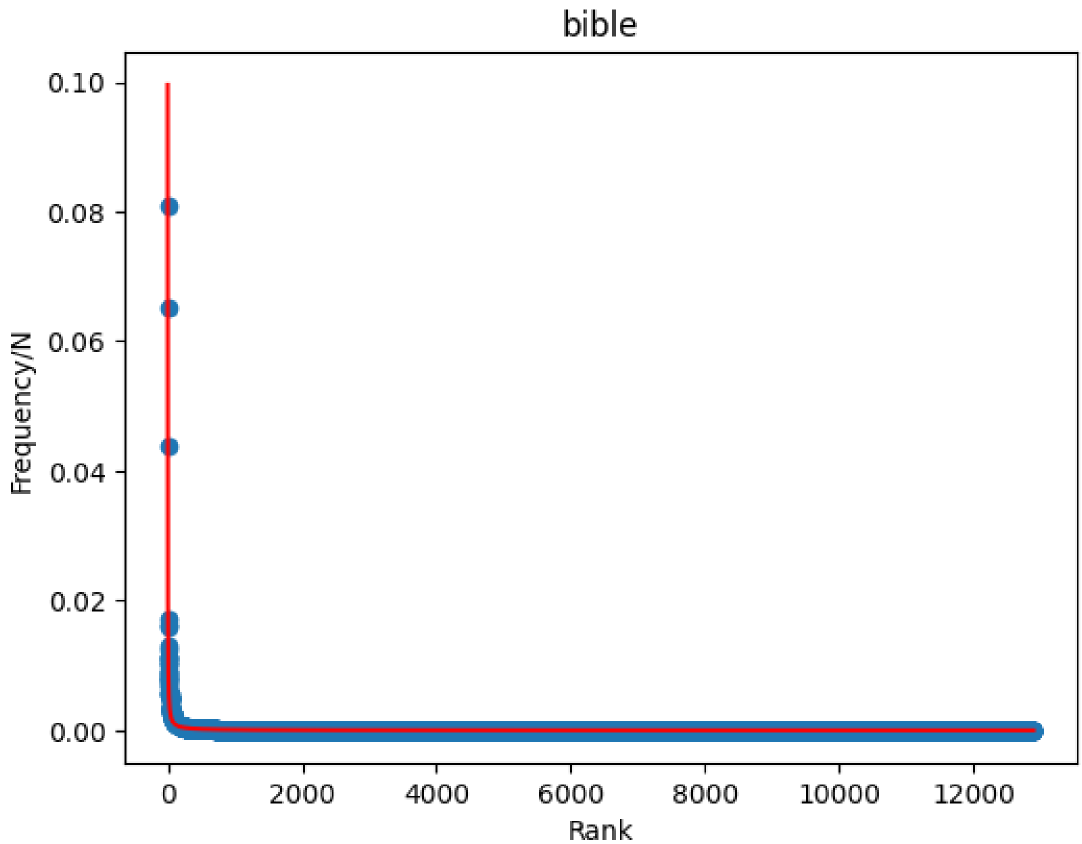

plato
Frequency/N
Rank
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}

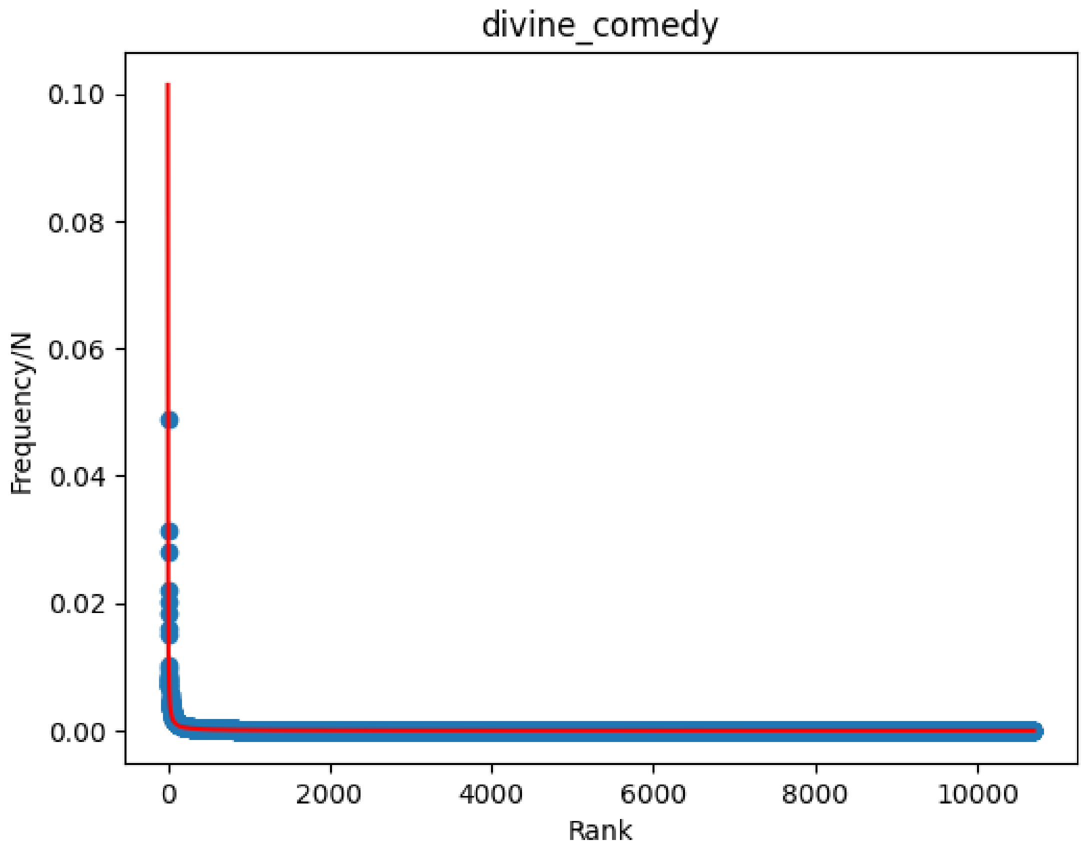


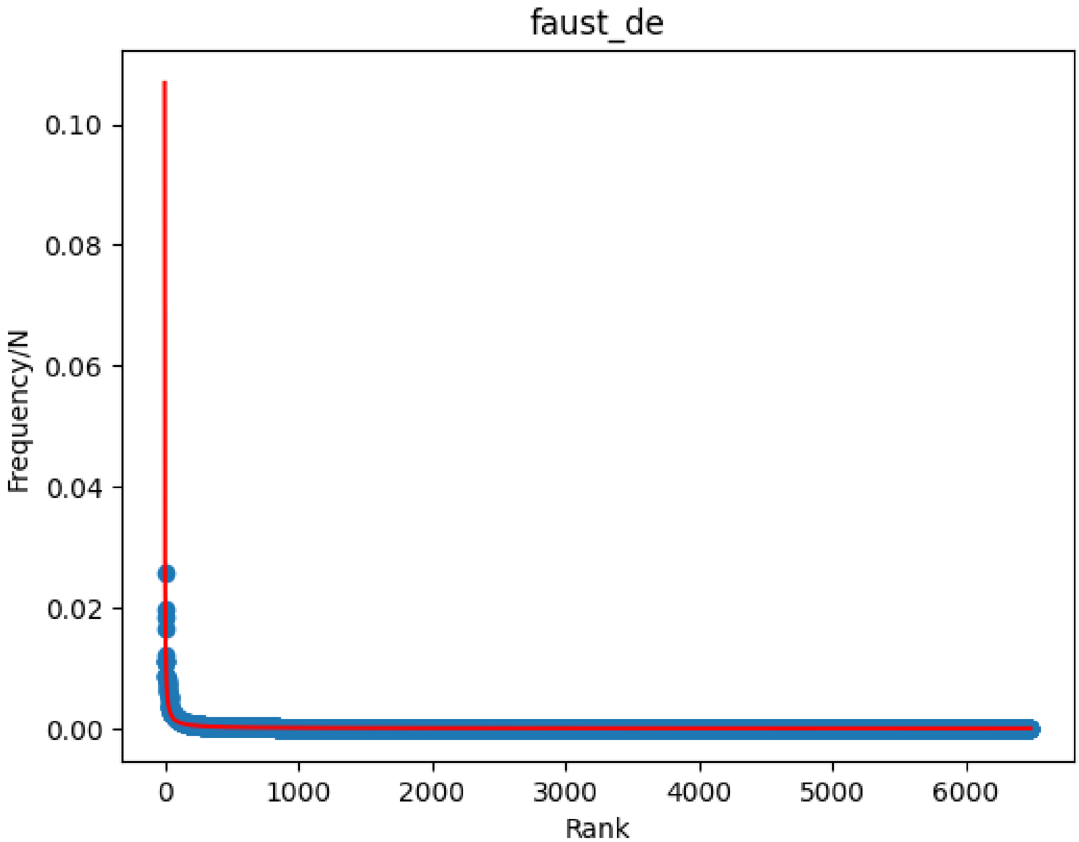
1.7 All books with linear comparison

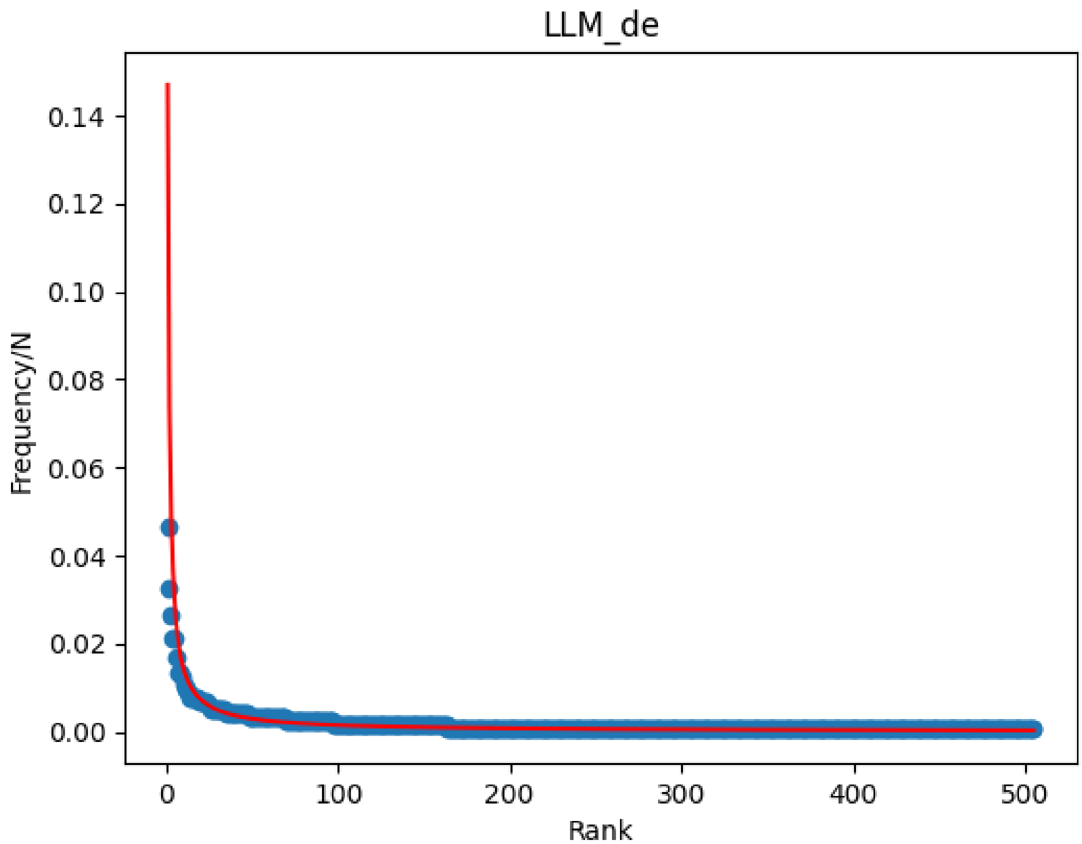
[191]:
```
fig = plt.figure(figsize=(4, 1))
for book_name, book in zip(data_frames.keys(), data_frames.values()):
    plt.figure()
    truncated_data = data_frames[book_name]
    plt.scatter(truncated_data['Rank'], truncated_data['Frequency/N'],␣
 ↪marker='o', label=book)
    plt.plot(truncated_data['Rank'], truncated_data['Zipf'], label='Zipf',␣
 ↪color='red')
    plt.xlabel('Rank')
    plt.ylabel('Frequency/N')
    plt.title(book_name)
    plt.show()
```

```
<Figure size 400x100 with 0 Axes>
```

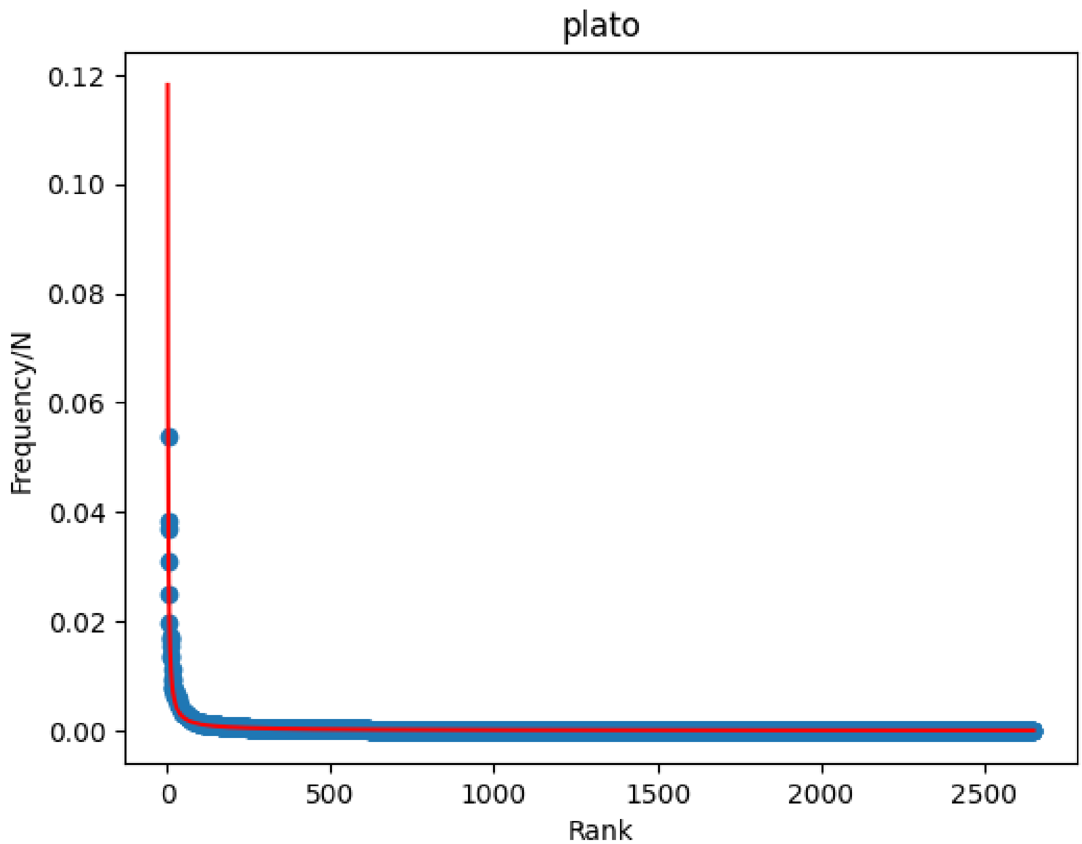

bible
Frequency/N
0.10
0.08
0.06
0.04
0.02
0.00
0
2000
4000
6000
8000
10000
12000
Rank


divine_comedy
0.10
0.08
0.06
0.04
0.02
0.00
Frequency/N
0
2000
4000
6000
8000
10000
Rank

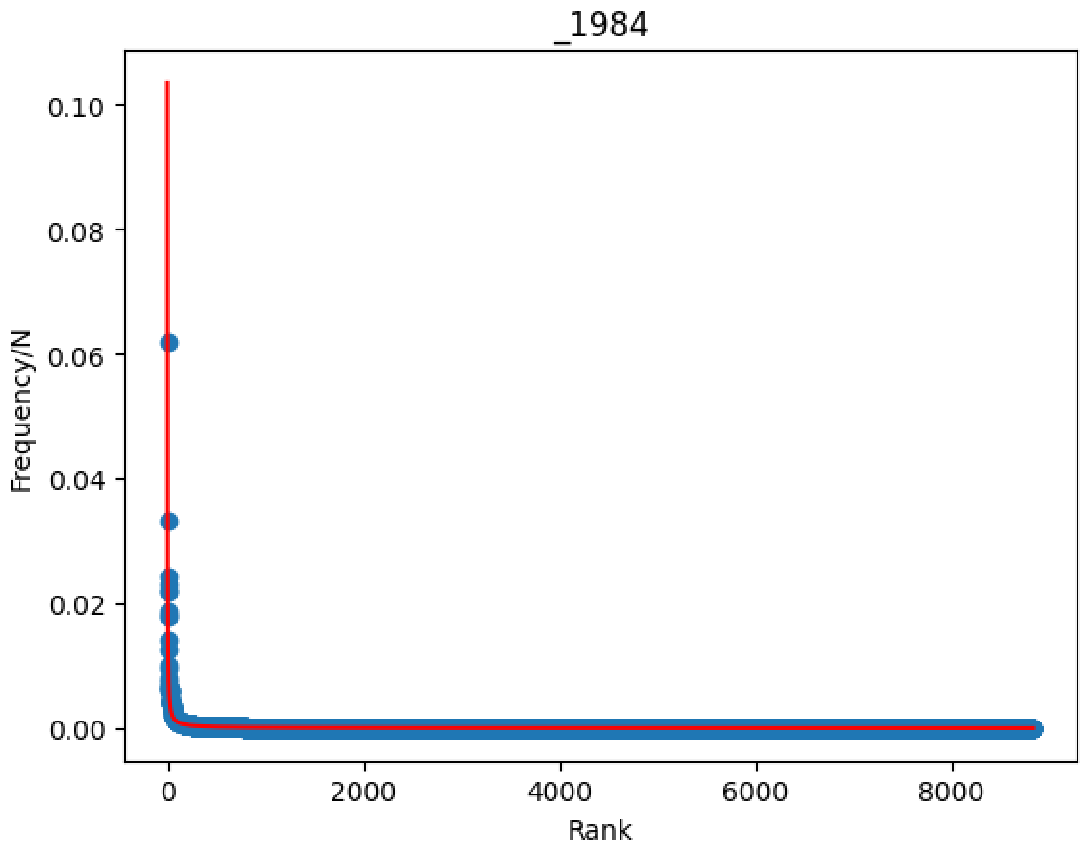

faust_de
0.10
0.08
0.06
0.04
0.02
0.00
Frequency/N
0
1000
2000
3000
4000
5000
6000
Rank



LLM_de
Frequency/N
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
100
200
300
400
500
Rank



plato
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Frequency/N
0
500
1000
1500
2000
2500
Rank



When comparing data on a linear scale, subtle changes are often difficult to observe, which become more apparent when viewed on a log-log scale. This is because **ranks increase exponentially**, typically by a factor of *10e4*. Since Zipf's Law involves the inverse of ranks, it also spans a similarly large range. In a log-log scale, this wide range is compressed and more manageable, making patterns and relationships easier to detect, whereas on a linear scale, the data can be too spread out to reveal these insights effectively.

1.8 Extra 1 & Extra 2

[192]:
```
path = Path('data_fitted')
files = path.glob('*.csv')
```

[193]:
```
data_frames = {}
for file in files:
    file_name = file.name.replace('.csv', '')
    print(file_name)
    data = pd.read_csv(file)
```

```
    data_frames[file_name] = data
    print(data.head(2))
```

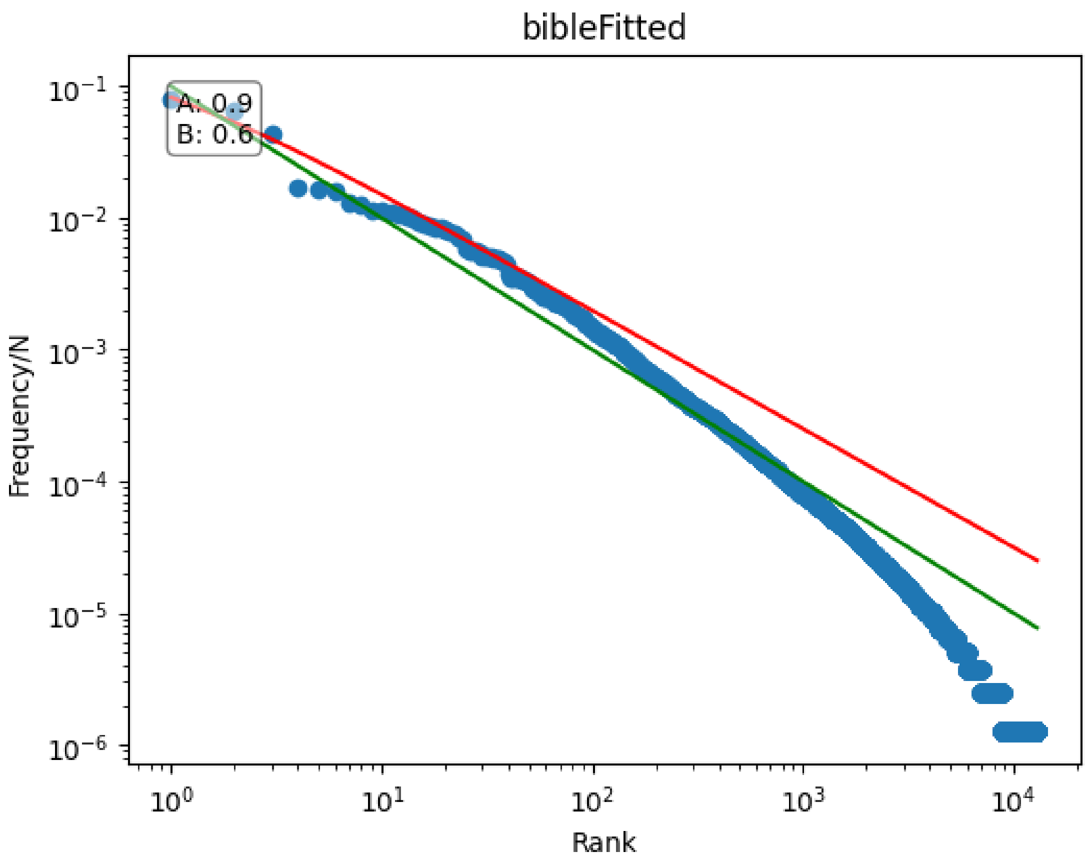
```
bibleFitted
  Token  Frequency  Rank  Frequency/N      Zipf  FittedZipf    A    B
0   the      64193     1     0.080766  0.099588    0.082163  0.9  0.6
1   and      51763     2     0.065127  0.049794    0.053077  0.9  0.6
divine_comedyFitted
  Token  Frequency  Rank  Frequency/N      Zipf  FittedZipf  A  B
0   the       5721     1     0.048968  0.101467    0.050733  1  1
1     d       3655     2     0.031284  0.050733    0.033822  1  1
faust_deFitted
  Token  Frequency  Rank  Frequency/N      Zipf  FittedZipf    A    B
0   und        917     1     0.025841  0.106898    0.022068  0.9  6.4
1   ich        700     2     0.019726  0.053449    0.019689  0.9  6.4
LLM_deFitted
  Token  Frequency  Rank  Frequency/N      Zipf  FittedZipf  A    B
0   die         53     1     0.046491  0.146999    0.044545  1  2.3
1   der         37     2     0.032456  0.073499    0.034186  1  2.3
platoFitted
  Token  Frequency  Rank  Frequency/N      Zipf  FittedZipf    A    B
0   the       1047     1     0.053994  0.118197    0.052859  0.9  2.1
1    of        744     2     0.038368  0.059098    0.041100  0.9  2.1
_1984Fitted
  Token  Frequency  Rank  Frequency/N      Zipf  FittedZipf  A    B
0   the       6527     1     0.061826  0.103473    0.060867  1  0.7
1    of       3500     2     0.033153  0.051737    0.038324  1  0.7
```

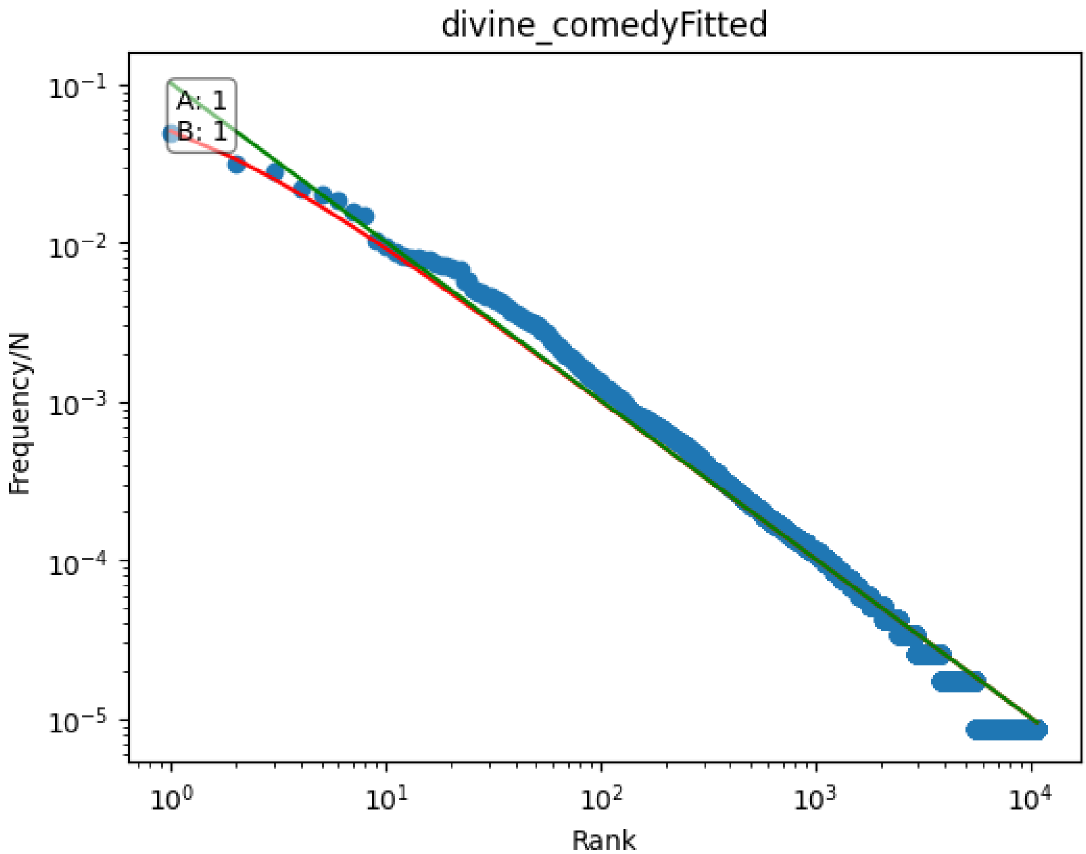
[194]:
```
fig = plt.figure(figsize=(4, 1))
for book_name, book in zip(data_frames.keys(), data_frames.values()):
    truncated_data = data_frames[book_name]
    plt.figure()
    plt.scatter(truncated_data['Rank'], truncated_data['Frequency/N'],␣
 ↪marker='o', label=book)
    plt.plot(truncated_data['Rank'], truncated_data['FittedZipf'],␣
 ↪label='FittedZipf', color='red')
    plt.plot(truncated_data['Rank'], truncated_data['Zipf'], label='Zipf',␣
 ↪color='green')
    a_value = truncated_data['A'].values[0]
    b_value = truncated_data['B'].values[0]
    plt.text(0.05, 0.95, f"A: {a_value}\nB: {b_value}", transform=plt.gca().
 ↪transAxes, fontsize=10,
             verticalalignment='top', horizontalalignment='left',␣
 ↪bbox=dict(boxstyle='round', facecolor='white', alpha=0.5))

    plt.xscale('log')
    plt.yscale('log')
```

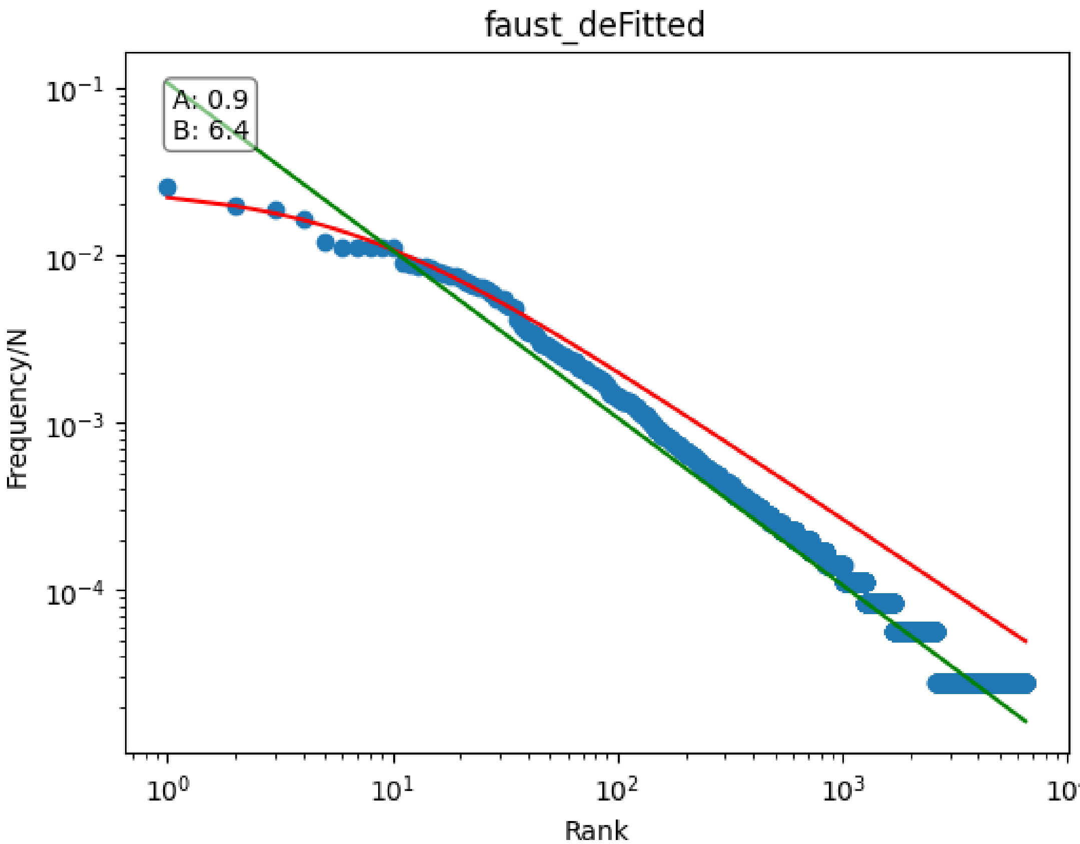
```
    plt.xlabel('Rank')
    plt.ylabel('Frequency/N')
    plt.title(book_name)
    plt.show()
```

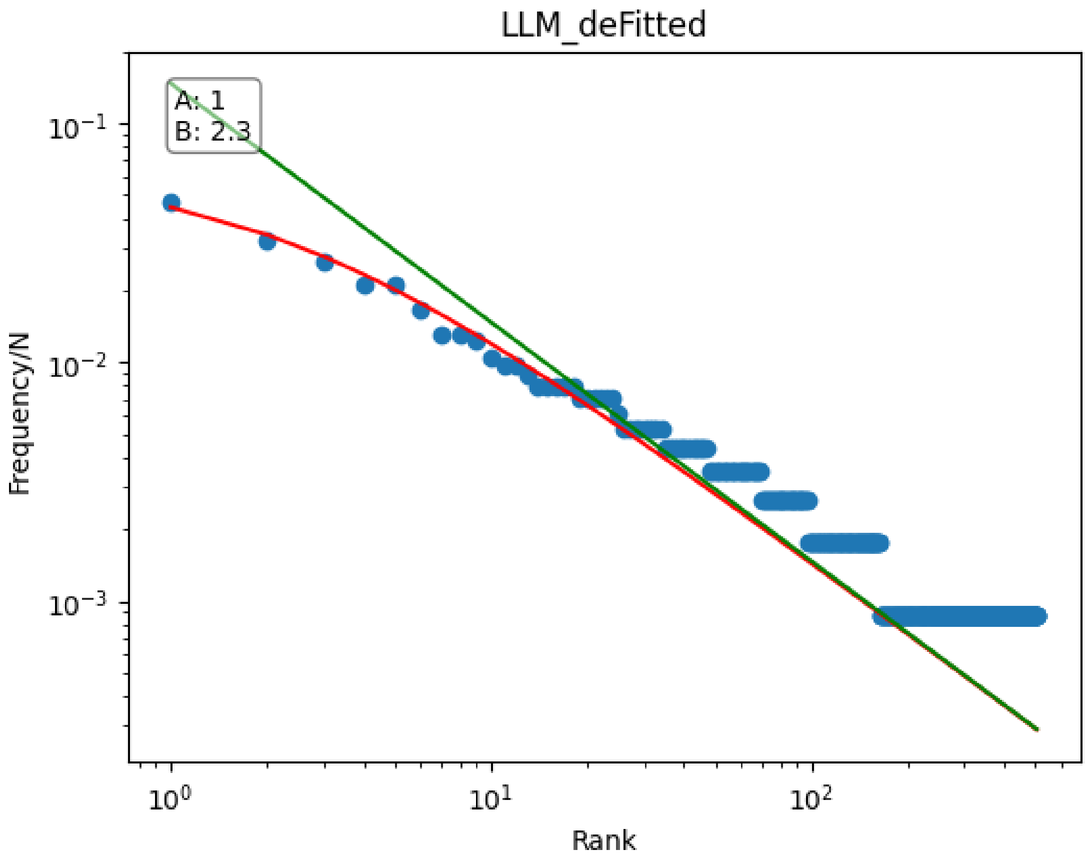
```
&lt;Figure size 400x100 with 0 Axes&gt;
```

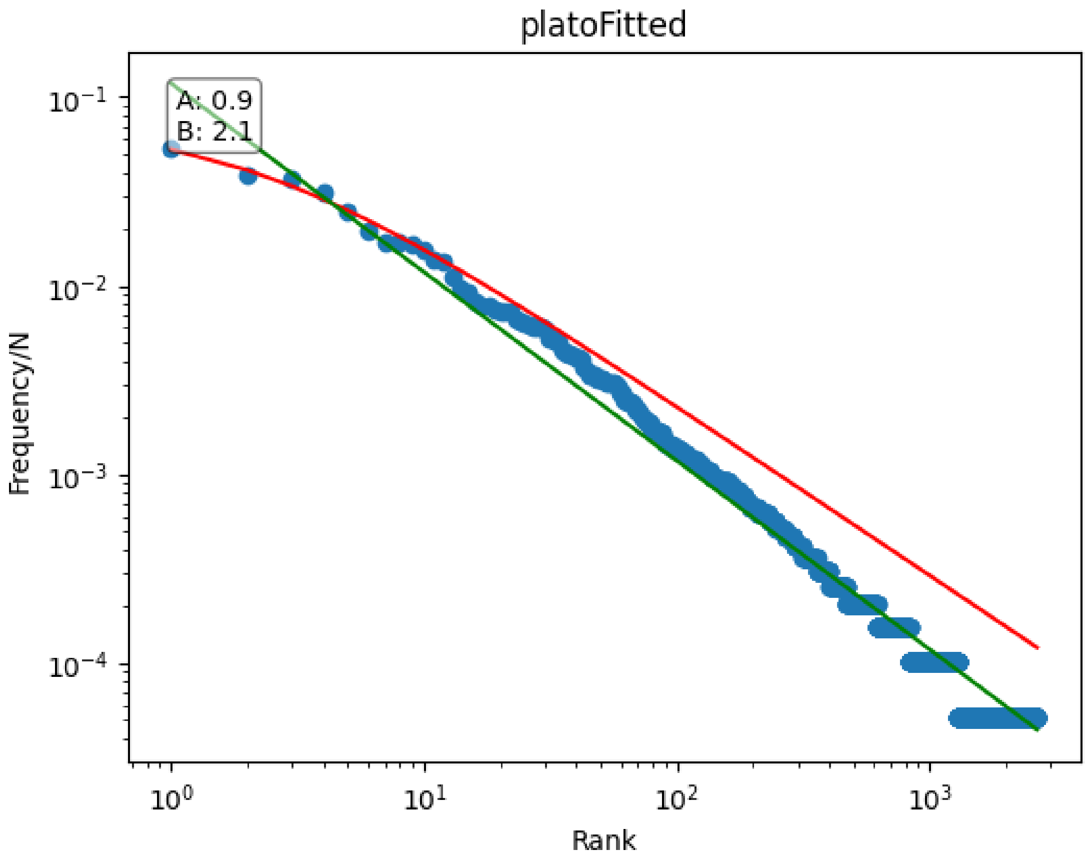


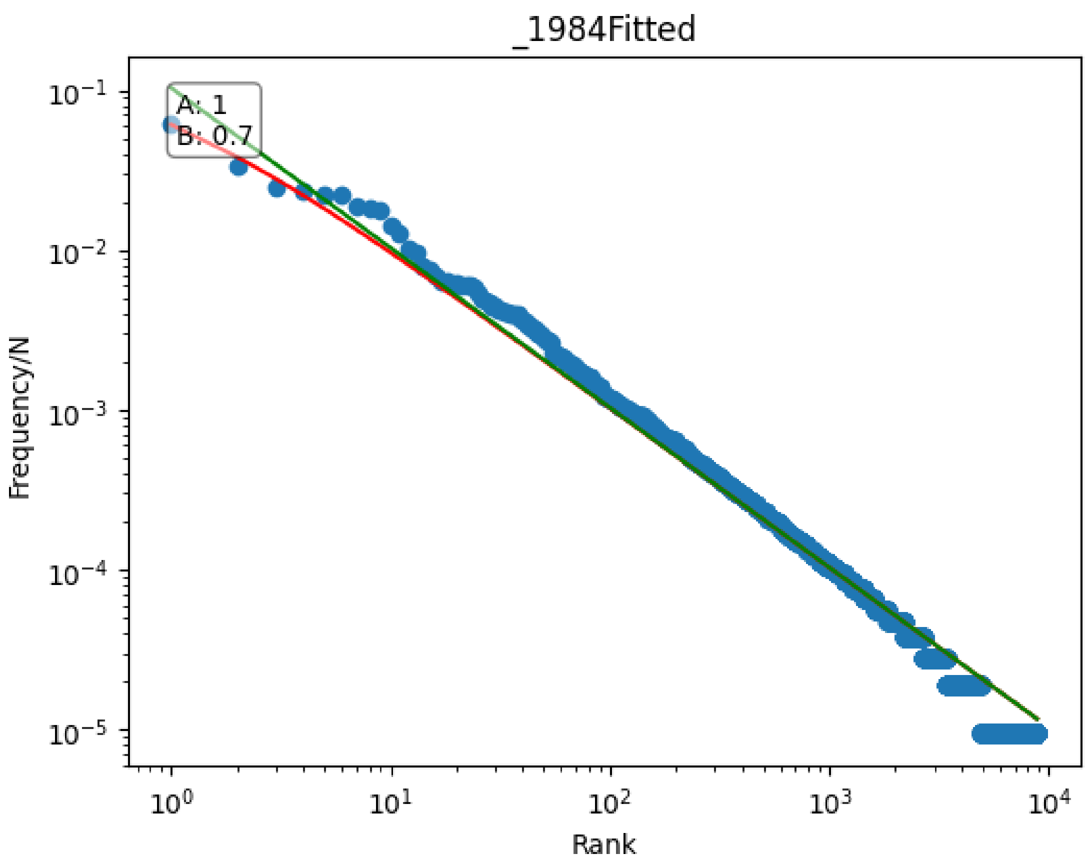


divine_comedyFitted
A: 1
B: 1
Frequency/N
Rank


faust_deFitted
A: 0.9
B: 6.4
Frequency/N
Rank
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^0
10^1
10^2
10^3
10^4


LLM_deFitted
A: 1
B: 2.3
Frequency/N
Rank
10^{-1}
10^{-2}
10^{-3}
10^{0}
10^{1}
10^{2}


platoFitted
A: 0.9
B: 2.1
Frequency/N
Rank
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}



As we can observe, the constants a and b vary across different texts. However, I believe that relying solely on these constants would not be sufficient to determine the language used in a book. It might be just a suggestion.-

The output of language models (LLMs) can be analyzed through the lens of the Zipf-Mandelbrot law. The values obtained can be fitted using linear regression, indicating that their distribution resembles that of Zipf's law. While the fit may not appear as precise as in other plots, this could be attributed to the limited sample size of the words.

Code: https://github.com/neuropython/ComplexSystems/blob/master/zipfLawAlgorithms.cpp