Project Title-

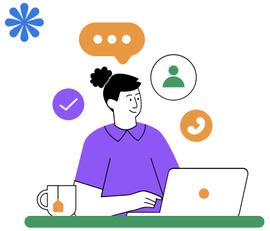
KatyPerry

Data Analysis with. Zipf's Law

This project examines Post Katy Perry's song lyrics using Zipf's Law and Exploratory Data Analysis, by analyzing word frequencies and album trends, the study aims to uncover linguistic insights with data science techniques., Using Python tools like Pandas and Matplotlib.

EDAtarians

Akanksha: Research & Documentation: theoretical insights and ppt work.
Bhavana Talkute: Coding Lead, data cleaning and Zipf's Law implementation.
Ananya Gupta: Final formatting and synthesis.
Anishka Khurana: created charts and data scripts.



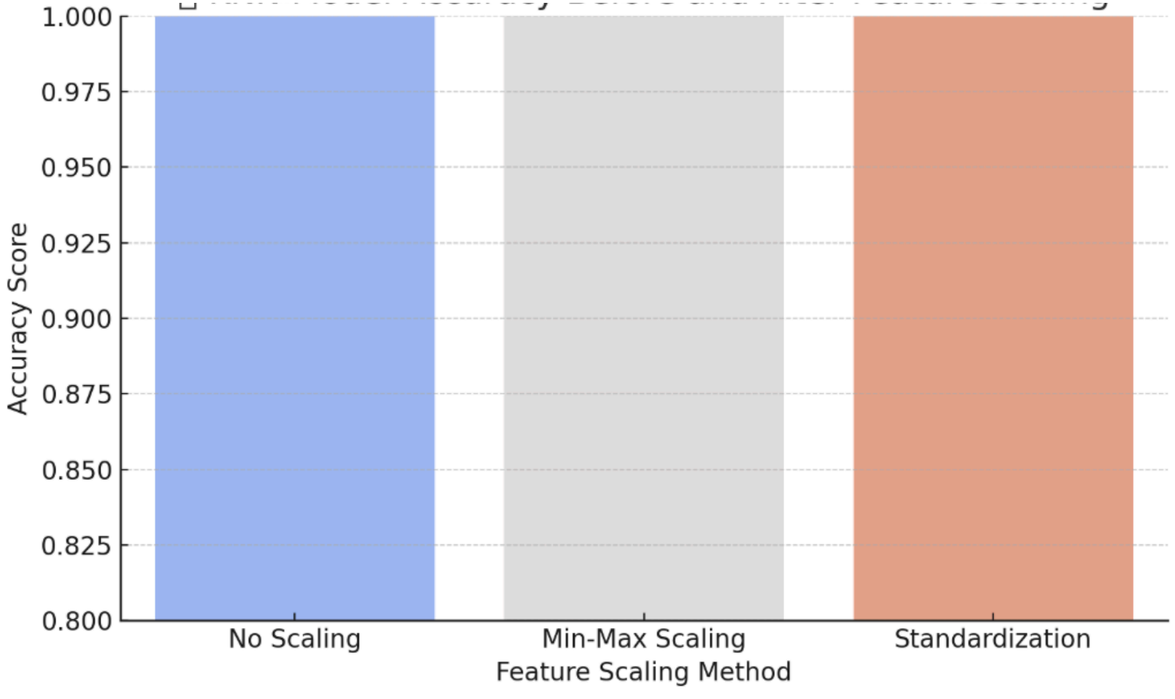
Tools & Technology

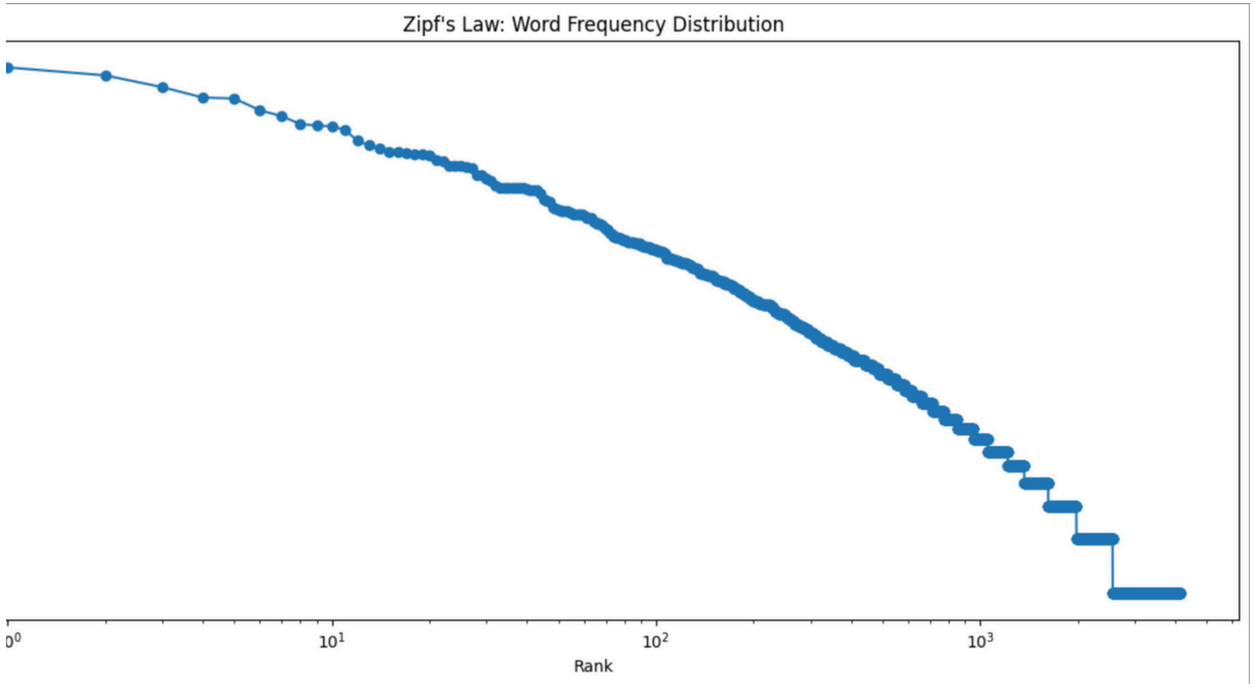
- **Python**
  Core programming language for implementation
- **Pandas**
  Used for efficient data handling and manipulation
- **Scikit-learn**
  Provides robust preprocessing methods for scaling techniques
- **Visualization Libraries**
  For comparing distributions before and after scaling
- **Dataset Source**
  PostMalone.csv with lyrics and metadata
- **Access Method**
  Mounted Google Drive for direct CSV reading in Colab

An Exploratory Data Analysis using Python and Pandas.

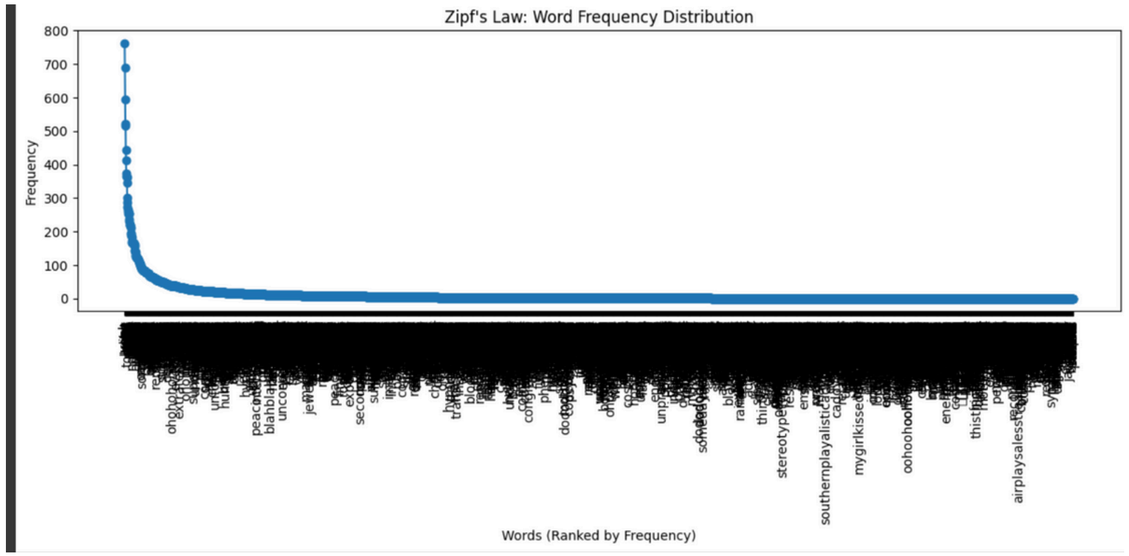
Zipf's Law in Lyrics Data

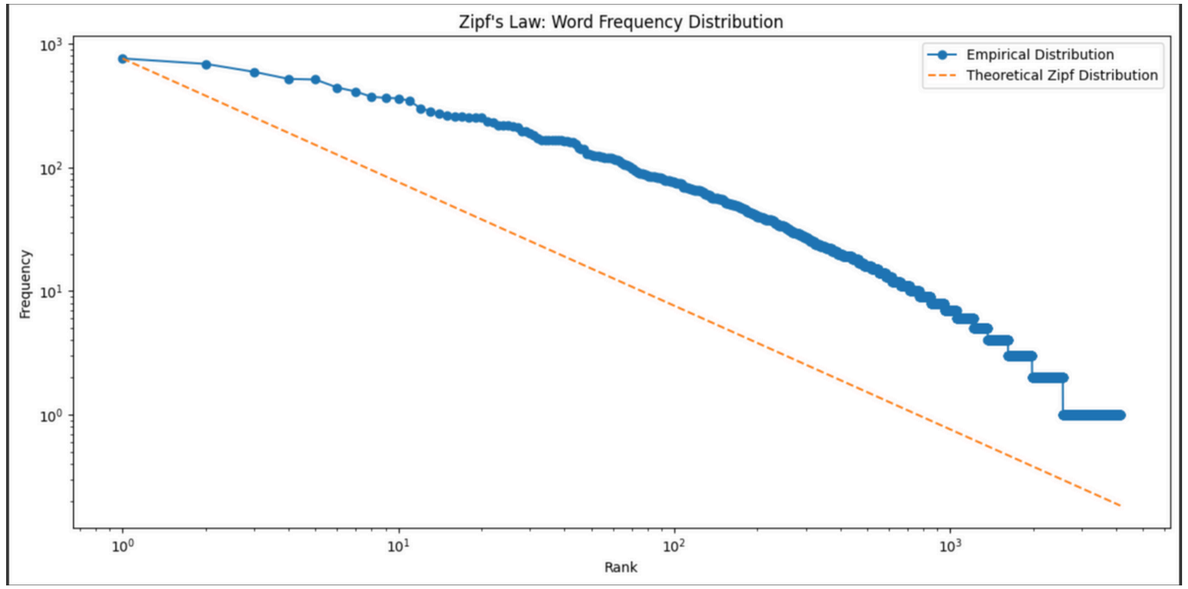
- **The graph shows a log-log plot of word frequency vs. rank in the lyrics dataset.**
- **As per Zipf's Law, a few words appear very frequently, while most words appear rarely.**
- **This pattern confirms a power-law distribution, which is typical in natural language.**
- **Understanding word frequency helps in feature selection and text vectorization steps.**
- **This insight is valuable for preprocessing in NLP models like TF-IDF or word embeddings.**





Analysis of Word Frequency in Song Lyrics:





- **Data Preparation:**

  The lyric variable consolidates all lyrics from the dataset into a single string for analysis.
- **Tokenization:**

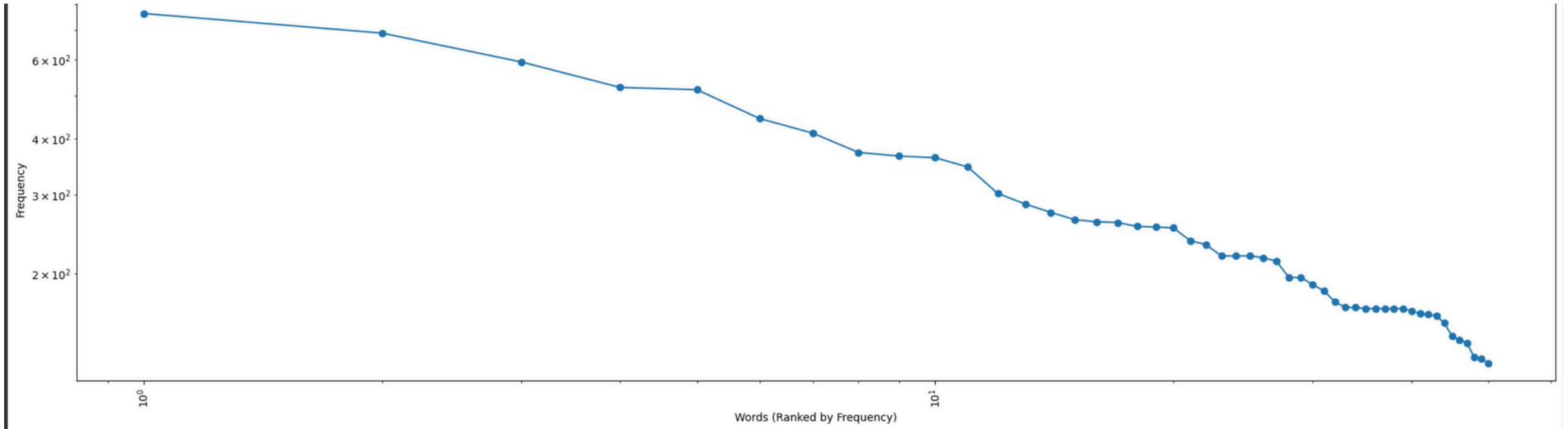
  word_tokenise splits the text into individual words, and all words are converted to lowercase to ensure uniformity.
- **Stopwords Removal:**

  Common words (stop words) like "and," "the," etc., are filtered out to focus on significant words using NLTK's stopword list.
- **Word Frequency Distribution:**

  The remaining words are used to create a frequency distribution (word_freq) using nltk.FreqDist.
- **Plotting Word Frequency:**

  A graph is plotted showing the frequency of words (ranked by their occurrence), visually representing Zipf's Law, which suggests that the frequency of words in a corpus follows a predictable pattern where a few words occur very frequently, and most words occur rarely.

  Feel free to modify the phrasing as needed for your presentation style!



Codes & Visuals

```
!pip install textblob==0.17.1
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt
import numpy as np # Import numpy

# Assuming 'df' is your DataFrame with a 'Lyrics' column
lyric = ' '.join(df['Lyric'].astype(str).tolist()) # Combine all lyrics
tokens = word_tokenize(lyric.lower())
stop_words = set(stopwords.words('english'))
filtered_tokens = [w for w in tokens if not w in stop_words and w.isalnum()]
word_freq = nltk.FreqDist(filtered_tokens)

# Sort and prepare for plotting
sorted_word_freq = sorted(word_freq.items(), key=lambda item: item[1], reverse=True)
words = [item[0] for item in sorted_word_freq]
frequencies = [item[1] for item in sorted_word_freq]

# Calculate ranks for x-axis (log scale needs numerical data)
ranks = np.arange(1, len(words) + 1)

# Plotting on a log-log scale
plt.figure(figsize=(12, 6))
plt.loglog(ranks, frequencies, marker='o', linestyle='-')  # Changed to plt.loglog
plt.title("Zipf's Law: Word Frequency Distribution")
plt.xlabel("Rank") # Changed x-axis label to Rank
plt.ylabel("Frequency")
# plt.xticks(rotation=90)  # Removed rotation for better log scale visualization
plt.tight_layout()
plt.show()
```

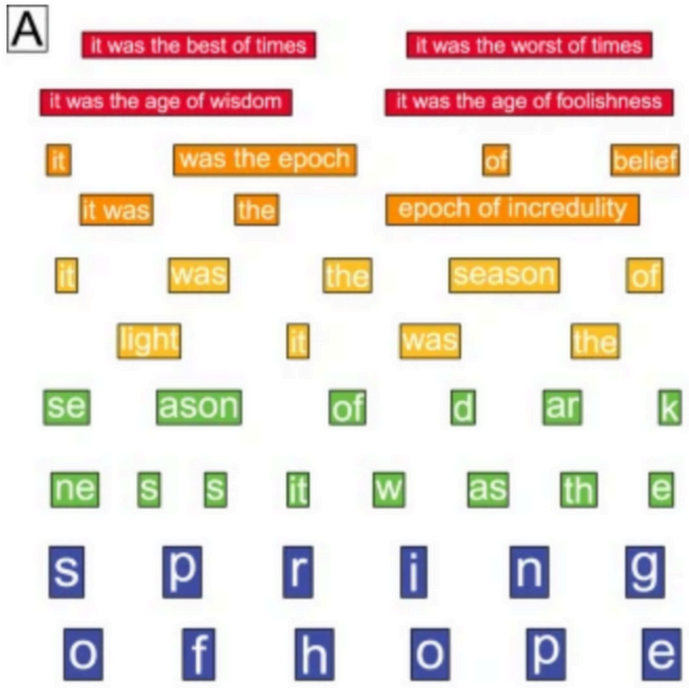
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame with a 'Lyrics' column
lyric = ' '.join(df['Lyric'].astype(str).tolist()) # Combine all lyrics
tokens = word_tokenize(lyric.lower())
stop_words = set(stopwords.words('english'))
filtered_tokens = [w for w in tokens if not w in stop_words and w.isalnum()]
word_freq = nltk.FreqDist(filtered_tokens)

# Sort and prepare for plotting
sorted_word_freq = sorted(word_freq.items(), key=lambda item: item[1], reverse=True)
words = [item[0] for item in sorted_word_freq]
frequencies = [item[1] for item in sorted_word_freq]

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(words, frequencies, marker='o', linestyle='-')
plt.title("Zipf's Law: Word Frequency Distribution")
plt.xlabel("Words (Ranked by Frequency)")
plt.ylabel("Frequency")
plt.xticks(rotation=90)
```

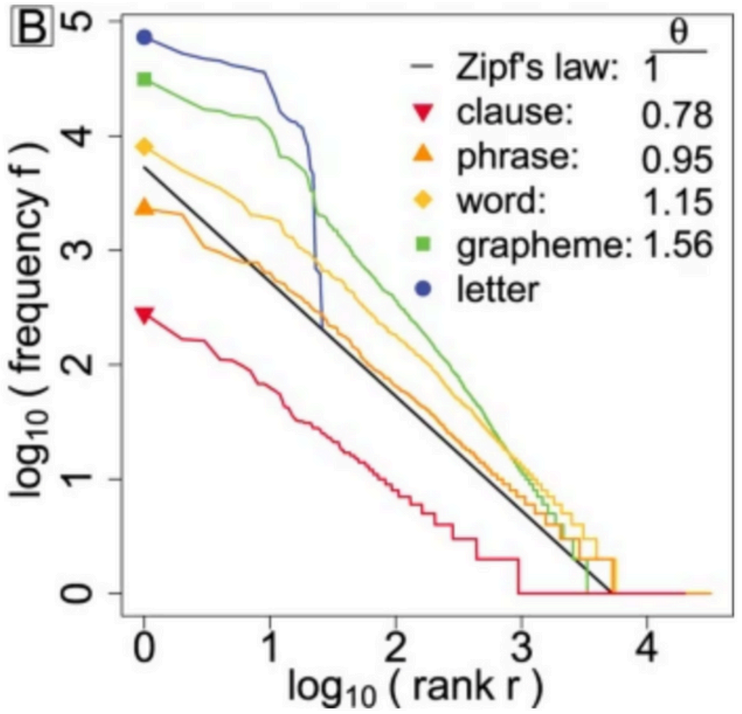
Key Insights Limitations

Zipf's Law Holds Common words rank high as expected

Deviations Explained Limited dataset, lyrical repetition, and style affect smoothness.

Conclusion : Lyrics follow Zipf's Law broadly but not perfectly.





Thank You