MVP_Lexical

March 15, 2025

1 Comparative Literary Translation Analysis: Lexical Diversity and Frequencies

1.1 Research Objectives

This notebook presents a **minimum viable product (MVP)** for a larger research project exploring lexical diversity in literary translations. The goal is to **test methods and materials** rather than reach definitive conclusions.

The workflow follows a **modular structure**, encompassing:
- **Text collection & preprocessing**
- **Exploratory data analysis (EDA)**
- **Lexical diversity experiments**
- **Statistical hypothesis testing**
- **Discussion of preliminary findings**

For this iteration, we analyze **two modern English translations** of *The Odyssey*:
Emily Wilson's translation
Peter Green's translation

1.2 Methodology Roadmap (Table of Contents)

1.2.1 1 Libraries & Text Acquisition

Setting up the environment and loading texts.

1.2.2 2 Experiment 1: Type-Token Ratio (TTR)

- **Implementation:** Computing TTR for both translations.
- **Parametric Hypothesis Testing:** Comparing TTR statistically.

1.2.3 3 Experiment 2: Zipf's Law Verification

- **Implementation:** Examining word frequency distributions.
- **Parametric Hypothesis Testing:** Evaluating Zipfian behavior.

1.2.4 4 Experiment 3: TF-IDF Analysis

- **Implementation:** Identifying key lexical differences.
- **Parametric Hypothesis Testing:** Assessing significant variations.

1.2.5 5 Discussion of Results

- **Findings:** Insights from experiments.
- **New Directions:** Refinements & future research steps.

1.3 1 Libraries & Text Acquisition

[1]:
```
# Libraries
import pandas as pd
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from tqdm import tqdm
import numpy as np
import re
import sys
import os
tqdm.pandas()
```

[2]:
```
# Visualization
%matplotlib inline

# Add the directory containing visualization_utils.py to path
sys.path.append("/Users/debr/English-Homer/")
import visualization_utils as viz
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
# palette astroblue   orange   genoa      carrot    tawny      neptune     ␣
 ↪SELAGO    mako    black
color = ['#003D59',␣
 ↪'#FD6626','#177070','#FB871D','#641B5E','#86C3BC','#F5E1FD','#414A4F','k']
danB_plotstyle = {'figure.figsize': (12, 7),
               'axes.labelsize': 'large', # fontsize for x and y labels (was␣
 ↪large)
               'axes.titlesize': 'large', # fontsize for title
               'axes.titleweight': 'bold', # font type for title
               'xtick.labelsize': 'large', # fontsize for x
               'ytick.labelsize':'small', # fontsize fory ticks
```

```
                'grid.color': 'k', # grid color
                 'grid.linestyle': ':', # grid line style
                 'grid.linewidth': 0.2, # grid line width
                 'font.family': 'Times New Roman', # font family
                 'grid.alpha': 0.5, # transparency of grid
                'figure.dpi': 300, # figure display resolution
                'savefig.bbox': 'tight', # tight bounding box
                'savefig.pad_inches': 0.4, # padding to use when saving
                'axes.titlepad': 15, # title padding
                'axes.labelpad': 8, # label padding
                'legend.borderpad': .6, # legend border padding
                'axes.prop_cycle': plt.cycler(
                 color=color) # color cycle for plot lines
                }

# adjust matplotlib defaults
plt.rcParams.update(danB_plotstyle)
```

[3]:
```
# Load CSVs
filepath_Wilson = "/Users/debr/odysseys_en/Odyssey_dfs/Odyssey_Wilson_eda_END.
 ↪csv"
filepath_Green = "/Users/debr/odysseys_en/Odyssey_dfs/Odyssey_Green_eda_END.csv"

df_W = pd.read_csv(filepath_Wilson)
df_G = pd.read_csv(filepath_Green)

# Add translation label
df_W["translation"] = "Wilson"
df_G["translation"] = "Green"

# merging "book_num" with "translation" to create a unique identifier
df_W["book_id"] = df_W["book_num"].astype(str) + "_W"
df_W = df_W.drop(columns=["book_num"])
df_G["book_id"] = df_G["book_num"].astype(str) + "_G"
df_G = df_G.drop(columns=["book_num"])

# Keep only necessary columns: book number & tokens
df_W = df_W[["translation", "book_id", "tokens"]]
df_G = df_G[["translation", "book_id", "tokens"]]

# Combine both into one DataFrame
df = pd.concat([df_W, df_G], ignore_index=True)

# Ensure tokens are stored as lists (if stored as strings, convert them)
df["tokens"] = df["tokens"].apply(lambda x: eval(x) if isinstance(x, str) else␣
 ↪x)
```

[4]:
```
#Boolean check for missing values
if df.isna().sum().sum() == 0:
    print("No missing values")
else:
    for col in df.columns:
        if df[col].isna().sum() > 0:
            print(f"Missing values in {col}")
print("df columns:", df.columns, "shape:", df.shape)
```

```
No missing values
df columns: Index(['translation', 'book_id', 'tokens'], dtype='object') shape:
(48, 3)
```

[5]:
```
df.head(2)
```

[5]:
```
  translation book_id                                             tokens
0      Wilson     1_W  [tell, complicated, man, muse, tell, wandered,…
1      Wilson     2_W  [dangerous, journey, early, dawn, born, finger…
```

1.4 2 Experiment 1: Type-Token Ratio (TTR)

Type-token ratio is a fundamental measure of lexical diversity in a text, calculated by dividing the number of unique words (types) by the total number of words (tokens) in a text. This ratio provides insight into the richness and variety of vocabulary employed by a writer or translator.

It is important regarding comparing literary translations because it allows us to quantify how translators differ in their lexical choices when rendering the same source text. A higher TTR suggests a more diverse vocabulary, which may indicate a translator's attempt to capture nuanced meanings or stylistic elements of the original work. Conversely, a lower TTR might suggest a more repetitive or constrained vocabulary, potentially reflecting a focus on accessibility, consistency, or adherence to the source text's own lexical patterns.

The Type-Token Ratio (TTR) formula to be implemented in python is:

$$TTR = \left(\frac{\text{Number of Unique Words}}{\text{Total Word Count}}\right) \times 100$$

1.4.1 Hypothesis Testing

For our comparative analysis of Green and Wilson's translations, we establish the following hypotheses:

H : The lexical diversity between Green and Wilson's translations is the same.
H : The lexical diversity of the two texts is different.

In statistical terms: - **H (Null Hypothesis)**: There is no significant variation between the TTR values of the two translations. - **H (Alternative Hypothesis)**: There is a significant variation between their TTR values.

1.4.2 Statistical Testing Approach

I will employ the t-test to determine whether any observed differences in TTR between the two translations are statistically significant or merely due to chance. This test is appropriate for comparing means between two independent samples.

1.4.3 Interpretation of Results

- If p-value < 0.05, we reject H , indicating a statistically significant difference in lexical diversity between the translations.
- If p-value 0.05, we fail to reject H , suggesting that any observed differences in lexical diversity may be attributable to random variation rather than substantive differences in translation approach.

1.4.4 Implications for Translation Analysis

From this statistical analysis, we can infer whether Green and Wilson employed significantly different vocabulary choices in their translations. This may reflect:

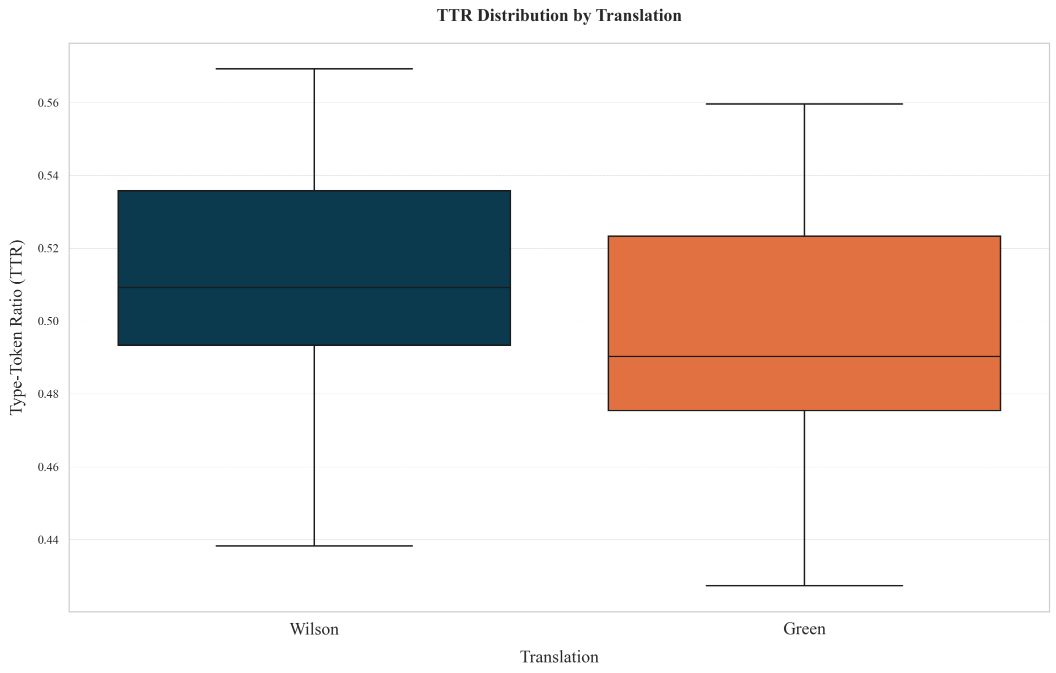
1. Different translation philosophies (e.g., domestication vs. foreignization)
2. Differences in target audience considerations
3. Temporal factors related to when each translation was produced
4. Individual stylistic preferences of the translators
5. Varying interpretations of the source text's meaning and aesthetic qualities

This quantitative approach provides an objective foundation for more nuanced qualitative analysis of how these translators have interpreted and rendered the original work.

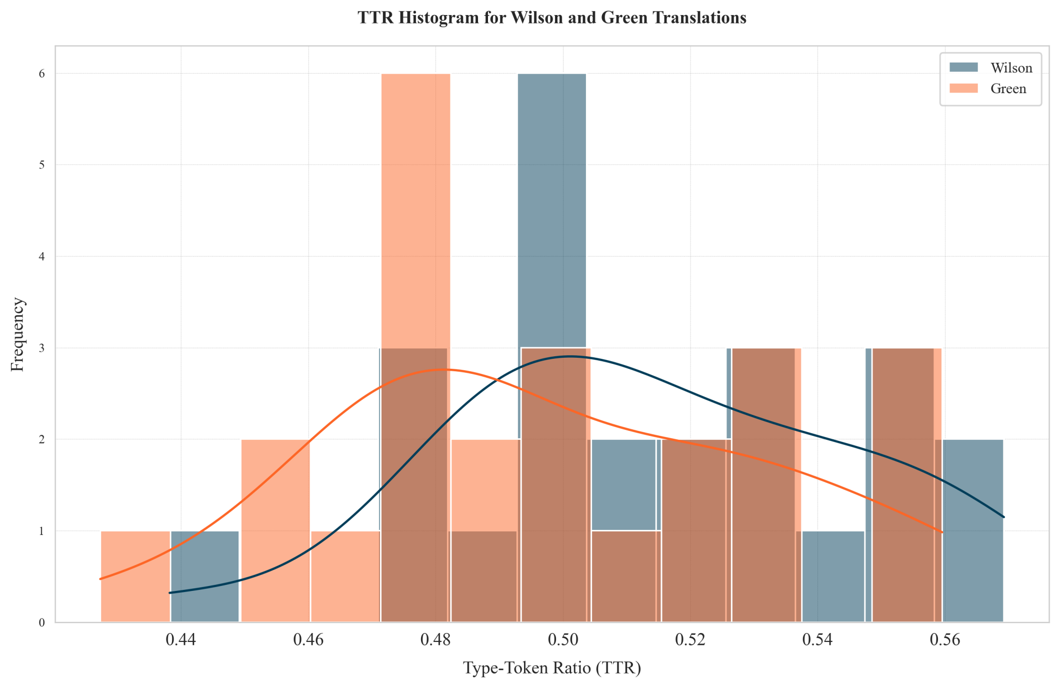
[6]:
```
# Compute type-token ratio (TTR) for each book

import scipy.stats as stats

# Compute TTR
df["ttr"] = df["tokens"].apply(lambda x: len(set(x)) / len(x) if x else 0)

# Separate TTRs for the two translations
ttr_wilson = df[df["translation"] == "Wilson"]["ttr"].tolist()
ttr_green = df[df["translation"] == "Green"]["ttr"].tolist()
#ttr_wilson = df[df["book_id"].str.endswith("W")]["ttr"].tolist() # Wilson␣
 ↪--Alternative:ttr_wilson = df[df["book_id"].str[-1] == "W"]["ttr"].tolist()
#ttr_green = df[df["book_id"].str.endswith("G")]["ttr"].tolist # Green␣
 ↪--Alternative: ttr_green = df[df["book_id"].str[-1] == "G"]["ttr"].tolist()
```

[7]:
```
# Create a boxplot for TTR values
sns.boxplot(x=df["translation"], y=df["ttr"], palette=color)
plt.title("TTR Distribution by Translation")
plt.xlabel("Translation")
plt.ylabel("Type-Token Ratio (TTR)")
plt. savefig("/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
 ↪MVP-TTR_distribution.png")
```

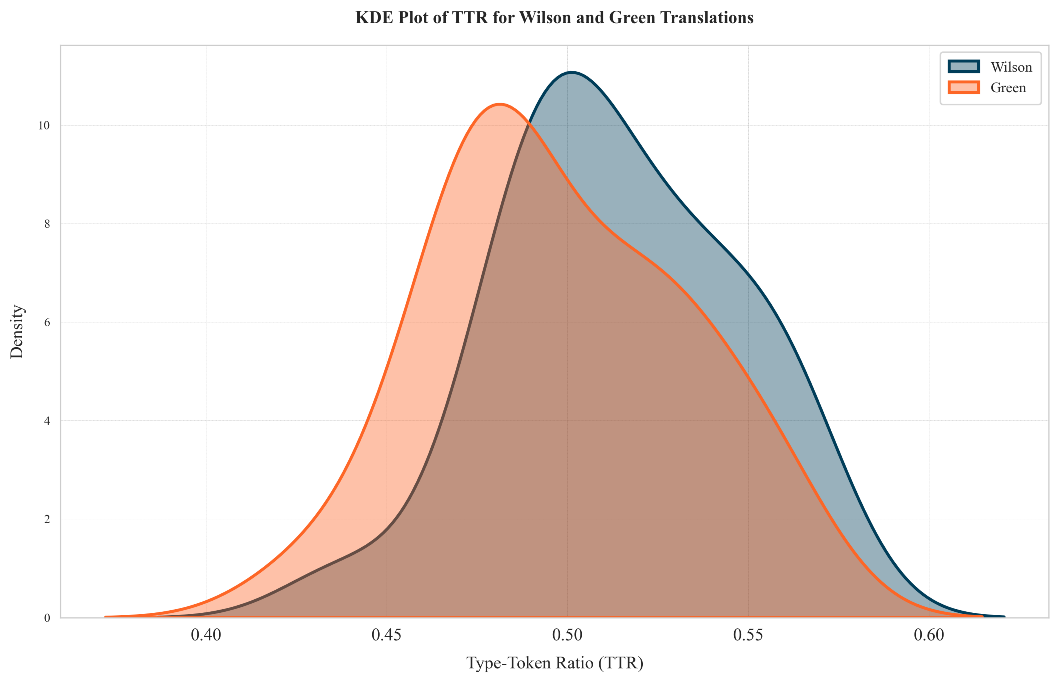
```
plt.show()
```



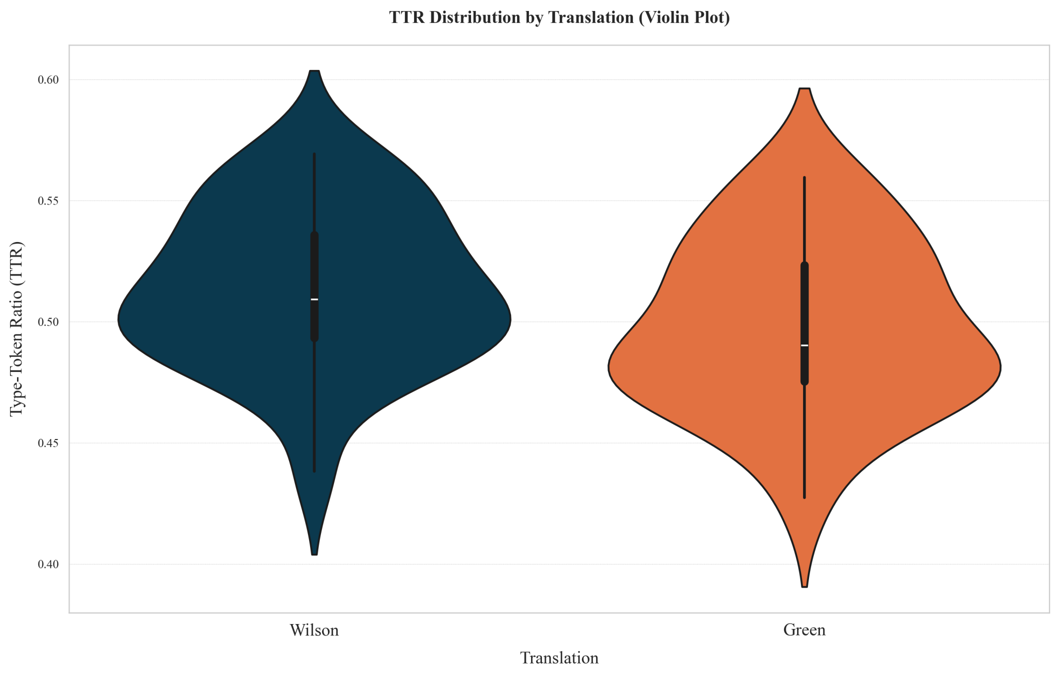
[8]:
```
# Create a histogram for TTR values
# Plot histogram for Wilson and Green translations
sns.histplot(ttr_wilson, bins=12, label="Wilson", kde=True, alpha=0.5)
sns.histplot(ttr_green, bins=12, label="Green", kde=True, alpha=0.5)
# Add labels and legend
plt.title("TTR Histogram for Wilson and Green Translations")
plt.xlabel("Type-Token Ratio (TTR)")
plt.ylabel("Frequency")
plt.legend()
plt.savefig("/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
 ↪MVP-TTR_histogram.png")
plt.show()
```



```
[9]: # KDE plot for Wilson and Green translations
     sns.kdeplot(ttr_wilson, fill=True, label="Wilson", alpha=0.4, linewidth=2)
     sns.kdeplot(ttr_green, fill=True, label="Green", alpha=0.4, linewidth=2)

     plt.title("KDE Plot of TTR for Wilson and Green Translations")
     plt.xlabel("Type-Token Ratio (TTR)")
     plt.ylabel("Density")
     plt.legend()
     plt.savefig("/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
      ↪MVP-TTR_kdeplot.png")
     plt.show()
```



[10]:
```
# Create a violin plot for TTR values
sns.violinplot(x=df["translation"], y=df["ttr"], palette=color)
plt.title("TTR Distribution by Translation (Violin Plot)")
plt.xlabel("Translation")
plt.ylabel("Type-Token Ratio (TTR)")
plt.savefig("/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
  ↪MVP-TTR_violinplot.png")
plt.show()
```



[11]:
```python
from scipy import stats

# Shapiro-Wilk test for ttr_wilson (for Wilson's data)
stat_wilson, p_value_wilson = stats.shapiro(ttr_wilson)
print(f"Shapiro-Wilk test for Wilson's data: T-statistic={stat_wilson}, 
 ↪p-value={p_value_wilson}")

# Shapiro-Wilk test for ttr_green (for Green's data)
stat_green, p_value_green = stats.shapiro(ttr_green)
print(f"Shapiro-Wilk test for Green's data: T-statistic={stat_green}, 
 ↪p-value={p_value_green}")

# Interpretation of p-values
if p_value_wilson < 0.05:
    print("Wilson's TTR data is not normally distributed.")
else:
    print("Wilson's TTR data is normally distributed.")

if p_value_green < 0.05:
    print("Green's TTR data is not normally distributed.")
else:
    print("Green's TTR data is normally distributed.")
```

```
Shapiro-Wilk test for Wilson's data: T-statistic=0.9710503976522772,
p-value=0.6929667067088071
Shapiro-Wilk test for Green's data: T-statistic=0.9647758095190286,
p-value=0.541526201743056
Wilson's TTR data is normally distributed.
Green's TTR data is normally distributed.
```

[12]:
```
# Perform t-test
t_stat, p_value = stats.ttest_ind(ttr_wilson, ttr_green)
print(f"T-statistic: {t_stat}, P-value: {p_value}")
if p_value < 0.05:
    print("The difference in TTR between Wilson and Green is statistically␣
 ↪significant.")
else:
    print("The difference in TTR between Wilson and Green is not statistically␣
 ↪significant.")
```

```
T-statistic: 1.6455514452077802, P-value: 0.10667311090438025
The difference in TTR between Wilson and Green is not statistically significant.
```

2 Discussion of Type-Token Ratio Analysis Results

2.1 Statistical Findings and Their Implications

The t-test comparing the Type-Token Ratios (TTR) between Wilson's and Green's translations yielded results that did not reach the threshold for statistical significance (p 0.05). This means we fail to reject the null hypothesis (H) that the lexical diversity between Green and Wilson's translations is the same.

2.2 Interpreting Non-Significant Results

While we did not detect a statistically significant difference, this finding itself is meaningful within translation studies. The absence of a significant difference does not represent a failure of the analysis, but rather provides evidence for an important theoretical perspective: translators working within similar cultural-temporal contexts may demonstrate comparable lexical diversity despite individual stylistic preferences.

2.3 Cultural and Contextual Influences

These results support the theory that translators are products of their time and cultural milieu. Both Wilson and Green, as contemporaries, likely:

1. **Shared linguistic resources**: Access to similar lexical resources and translation tools of their era
2. **Operated within common translation norms**: Adherence to prevailing standards and expectations in literary translation
3. **Responded to similar audience expectations**: Accommodation to contemporary readers' preferences and comprehension levels
4. **Were informed by comparable theoretical frameworks**: Influence of translation theories dominant during their working period

2.4 Beyond Statistical Significance

This finding invites us to look beyond mere statistical differences to consider the subtle ways translators negotiate between:

- Fidelity to the source text
- Readability for the target audience
- Literary aesthetics and stylistic considerations
- Cultural and temporal adaptation

The similar TTR values suggest that both translators achieved comparable lexical diversity while potentially making different word choices. This highlights the complexity of translation as both an art and science, where multiple valid approaches can yield texts with similar quantitative measures of diversity.

2.5 Implications for Translation Theory

These results support the view that translators operate within what Bourdieu would call a "field" - a structured social space with its own rules and capital. The comparable TTR values indicate that both translators have internalized similar dispositions (habitus) regarding appropriate lexical diversity in literary translation, despite potentially different translational choices at the sentence or phrase level.

2.6 Future Research Directions

To build upon these findings, future research might:

1. Examine qualitative differences in word choice and register between the translations
2. Analyze other linguistic features such as sentence length, syntactic complexity, or metaphor preservation
3. Compare these translators to others from distinctly different cultural-temporal contexts
4. Investigate reader responses to determine if comparable TTR values correspond to similar reader experiences

2.7 Conclusion

The non-significant difference in TTR between Wilson and Green's translations reveals the subtle ways cultural context shapes translation practice. Far from being a null result, this finding contributes to our understanding of how translators, as cultural mediators, are influenced by their shared temporal and social contexts while exercising individual agency within those constraints.

2.8 3 Experiment 2: Zipf's Law Verification

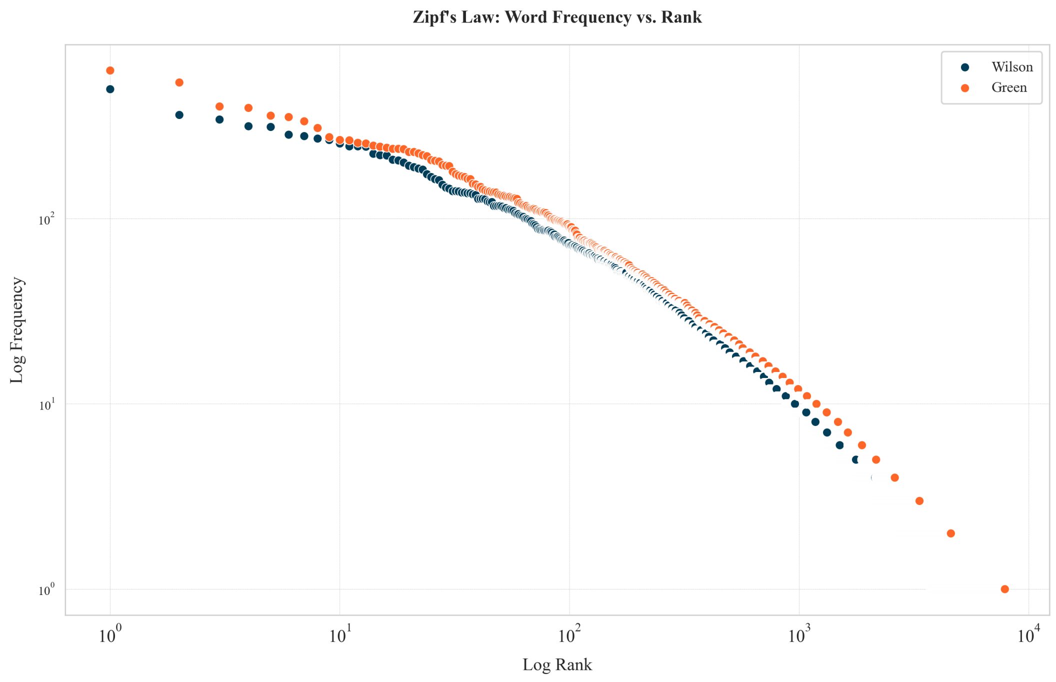
2.8.1 Implementation

- Plot word frequency vs. rank to check adherence to Zipf's Law.
- Log-log plot comparison for both translations.

[13]:
```
# Flatten all tokens into a single list for each translation
```

```
tokens_wilson = [token for tokens in df[df["translation"] ==␣
 ↪"Wilson"]["tokens"] for token in tokens]
tokens_green = [token for tokens in df[df["translation"] == "Green"]["tokens"]␣
 ↪for token in tokens]

# Count word frequencies
freq_wilson = Counter(tokens_wilson)
freq_green = Counter(tokens_green)

# Convert to DataFrame with ranks
df_zipf_wilson = pd.DataFrame(freq_wilson.items(), columns=["word",␣
 ↪"frequency"]).sort_values(by="frequency", ascending=False)
df_zipf_wilson["rank"] = df_zipf_wilson["frequency"].rank(method="first",␣
 ↪ascending=False)

df_zipf_green = pd.DataFrame(freq_green.items(), columns=["word", "frequency"]).
 ↪sort_values(by="frequency", ascending=False)
df_zipf_green["rank"] = df_zipf_green["frequency"].rank(method="first",␣
 ↪ascending=False)
```

[14]:
```
# Plot Wilson and Green's Zipf distribution
sns.scatterplot(x=df_zipf_wilson["rank"], y=df_zipf_wilson["frequency"],␣
 ↪label="Wilson")
sns.scatterplot(x=df_zipf_green["rank"], y=df_zipf_green["frequency"],␣
 ↪label="Green")

plt.xscale("log")
plt.yscale("log")
plt.xlabel("Log Rank")
plt.ylabel("Log Frequency")
plt.title("Zipf's Law: Word Frequency vs. Rank")
plt.legend()
plt.savefig("/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
 ↪MVP-Zipf_distribution.png")
plt.show()
```



Statistical Test for Zipf's Law (Linear Fit)

[15]:
```
import numpy as np
from scipy.stats import linregress

# Perform linear regression in log-log space
log_rank_w = np.log(df_zipf_wilson["rank"])
log_freq_w = np.log(df_zipf_wilson["frequency"])
slope_w, intercept_w, r_value_w, p_value_w, std_err_w = linregress(log_rank_w,␣
↪log_freq_w)

log_rank_g = np.log(df_zipf_green["rank"])
log_freq_g = np.log(df_zipf_green["frequency"])
slope_g, intercept_g, r_value_g, p_value_g, std_err_g = linregress(log_rank_g,␣
↪log_freq_g)

# Display results
print(f"Wilson: Slope = {slope_w:.2f}, R² = {r_value_w**2:.3f}, p-value =␣
↪{p_value_w:.3g}")
print(f"Green: Slope = {slope_g:.2f}, R² = {r_value_g**2:.3f}, p-value =␣
↪{p_value_g:.3g}")

# Check if slopes are close to -1 (Zipf's Law predicts ~ -1)
```

```
if -1.2 < slope_w < -0.8:
    print("Wilson's translation follows Zipf's Law.")
else:
    print("Wilson's translation deviates from Zipf's Law.")

if -1.2 < slope_g < -0.8:
    print("Green's translation follows Zipf's Law.")
else:
    print("Green's translation deviates from Zipf's Law.")
```

```
Wilson: Slope = -1.14, R² = 0.959, p-value = 0
Green: Slope = -1.14, R² = 0.960, p-value = 0
Wilson's translation follows Zipf's Law.
Green's translation follows Zipf's Law.
```

Statistical Test for Differences in Zipf Slopes

[16]:
```
from scipy.stats import ttest_ind

# Compute t-test for the difference in slopes
t_stat, p_value = ttest_ind([slope_w], [slope_g])

# Print results
print(f"T-statistic: {t_stat:.3f}, P-value: {p_value:.5f}")

# Interpretation
if p_value < 0.05:
    print("The difference in Zipf's Law slopes between Wilson and Green is␣
 ↪statistically significant.")
else:
    print("No significant difference in Zipf's Law slopes between Wilson and␣
 ↪Green.")
```

```
T-statistic: nan, P-value: nan
No significant difference in Zipf's Law slopes between Wilson and Green.
```

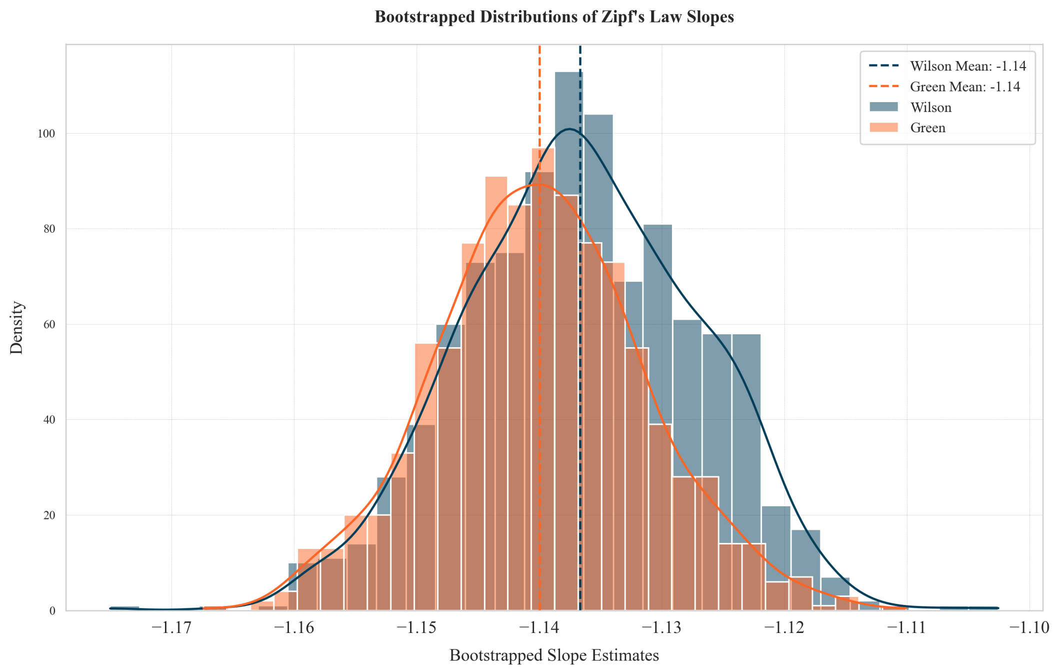
Bootstrapped Confidence Intervals

[17]:
```
# Function to bootstrap slope estimates
def bootstrap_slopes(log_rank, log_freq, num_samples=1000):
    slopes = []
    for _ in range(num_samples):
        sample_indices = np.random.choice(len(log_rank), len(log_rank),␣
 ↪replace=True)
        slope, _, _, _, _ = linregress(log_rank[sample_indices],␣
 ↪log_freq[sample_indices])
        slopes.append(slope)
    return np.array(slopes)
```

```
# Bootstrap slopes for both translations
bootstrap_slopes_w = bootstrap_slopes(log_rank_w, log_freq_w)
bootstrap_slopes_g = bootstrap_slopes(log_rank_g, log_freq_g)

# Compute 95% confidence intervals
ci_w = np.percentile(bootstrap_slopes_w, [2.5, 97.5])
ci_g = np.percentile(bootstrap_slopes_g, [2.5, 97.5])

print(f"Wilson's slope 95% CI: {ci_w}")
print(f"Green's slope 95% CI: {ci_g}")

# Check if CIs overlap
if (ci_w[0] > ci_g[1]) or (ci_g[0] > ci_w[1]):
    print("Statistically significant difference between the two Zipf slopes.")
else:
    print("No significant difference between the two Zipf slopes.")
```

```
Wilson's slope 95% CI: [-1.1552267  -1.11907835]
Green's slope 95% CI: [-1.15682261 -1.12287167]
No significant difference between the two Zipf slopes.
```

[18]:
```
# Plot histograms for bootstrapped slopes
sns.histplot(bootstrap_slopes_w, bins=30, kde=True, label="Wilson", alpha=0.5)
sns.histplot(bootstrap_slopes_g, bins=30, kde=True, label="Green", alpha=0.5)

# Add vertical lines for mean slopes color red

plt.axvline(np.mean(bootstrap_slopes_w), linestyle="--",
                                        color="#003D59",
                                        label=f"Wilson Mean: {np.
  ↪mean(bootstrap_slopes_w):.2f}")
plt.axvline(np.mean(bootstrap_slopes_g), linestyle="--",
                                        color='#FD6626',
                                        label=f"Green Mean: {np.
  ↪mean(bootstrap_slopes_g):.2f}")

# Labels and title
plt.xlabel("Bootstrapped Slope Estimates")
plt.ylabel("Density")
plt.title("Bootstrapped Distributions of Zipf's Law Slopes")
plt.legend()
plt.savefig("/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
  ↪MVP-Zipf_bootstrap.png")
plt.show()
```



2.9 4 Experiment 3: TF-IDF Analysis

2.9.1 Proving my stpe by step impementation with Wilson's Odyssey

STEP 1. Term Frequency (TF) Calculation Term Frequency measures how frequently a term appears in a document. It is calculated as:

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

This normalizes term counts by document length, ensuring that longer documents don't get artificially higher weights.

[19]:
```
# Step 1: Calculate Term Frequency (TF)
# Wilson's translation
# Function to compute term frequency
def term_freq_by_doc(list_of_tokens):
    token_list = eval(list_of_tokens)  # Ensure tokens are treated as a list
    term_counts = Counter(token_list)  # Count occurrences of each term
    total_terms = len(token_list)  # Total number of terms in the document

    # Compute TF: term frequency for each token in the document (normalized by␣
↪total terms)
```

```
    term_freq = {term: count / total_terms for term, count in term_counts.
↪items()}

    return term_freq, term_counts

# Apply function to compute TF for each book
df_W["term_freq"], df_W["term_counts"] = zip(*df_W["tokens"].
↪apply(term_freq_by_doc))
df_W.head(2)
```

[19]:
```
  translation book_id                                             tokens  \
0      Wilson     1_W  ['tell', 'complicated', 'man', 'muse', 'tell',…
1      Wilson     2_W  ['dangerous', 'journey', 'early', 'dawn', 'bor…

                                           term_freq  \
0  {'tell': 0.008097165991902834, 'complicated': …
1  {'dangerous': 0.0005945303210463733, 'journey'…

                                         term_counts
0  {'tell': 14, 'complicated': 1, 'man': 14, 'mus…
1  {'dangerous': 1, 'journey': 8, 'early': 1, 'da…
```

2.9.2 Inverse Document Frequency (IDF) Calculation

IDF measures how important a term is by examining how rare it is across all documents:

$$IDF(t) = \log\left(\frac{N}{1 + \text{doc count}}\right)$$

Terms that appear in many documents receive lower IDF scores, reducing their importance in the final TF-IDF score.

[20]:
```
# Step 2: Calculate Inverse Document Frequency (IDF)
# Calculate Inverse Document Frequency (IDF)

# Get total number of documents (books)
N = len(df_W)

# Count how many documents contain each term
doc_containing_term = Counter()
for term_counts in df_W["term_freq"]:
    doc_containing_term.update(term_counts.keys())  # Count unique terms in␣
↪each document

# Compute IDF for each term
idf_scores = {term: np.log(N / (1 + doc_count)) for term, doc_count in␣
↪doc_containing_term.items()}  # Adding 1 to avoid division by zero
```

```
# Add IDF column to df_W
df_W["idf"] = df_W["term_freq"].apply(lambda term_freq: {term: idf_scores[term]␣
↪for term in term_freq})

# Display results
df_W[["book_id", "idf"]].head(2)
```

[20]:
```
  book_id                                                idf
0     1_W  {'tell': -0.040821994520255166, 'complicated':…
1     2_W  {'dangerous': 1.2321436812926323, 'journey': 0…
```

2.9.3 TF-IDF Calculation

TF-IDF combines Term Frequency and Inverse Document Frequency to weight terms based on both their importance in a specific document and their distinctiveness across the corpus:

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

Terms with high TF-IDF scores appear frequently in a specific document but rarely in other documents, making them likely more important for that document's content.

[21]:
```
# Step 3: Calculate TF-IDF
# Compute TF-IDF by multiplying TF and IDF for each term in each document
df_W["tf_idf"] = df_W.apply(lambda row: {term: row["term_freq"][term] *␣
↪row["idf"][term] for term in row["term_freq"]}, axis=1)

# Display results
df_W[["book_id", "tf_idf"]].head(2)
```

[21]:
```
  book_id                                             tf_idf
0     1_W  {'tell': -0.00033054246575105396, 'complicated…
1     2_W  {'dangerous': 0.000732546778414169, 'journey':…
```

2.9.4 E2E TF-IDF Function & Visualization

Green's Odyssey The algorith

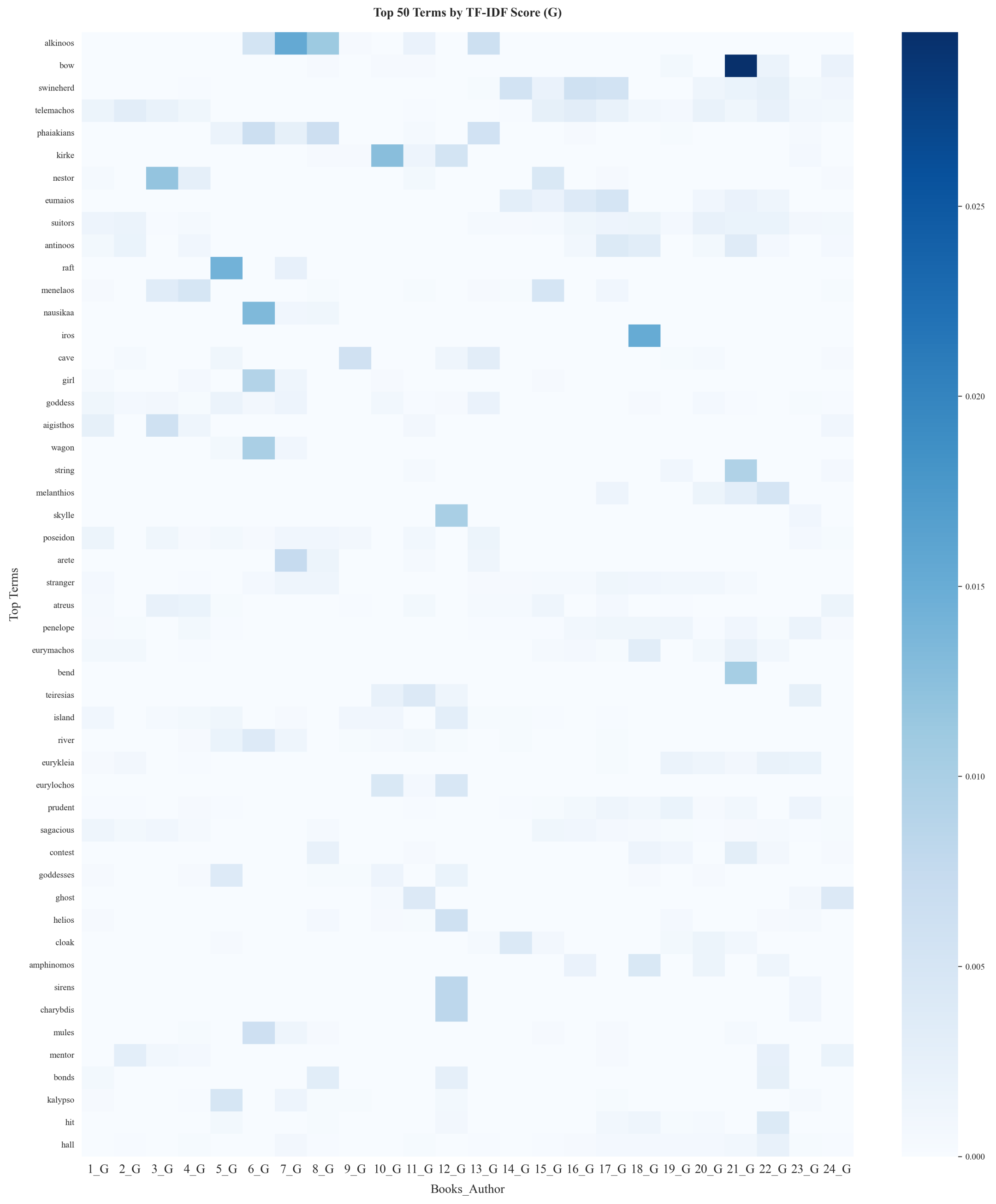
[22]:
```
import pandas as pd
import numpy as np
from collections import Counter

def calculate_tfidf(df):
    """
    Calculate TF-IDF scores for a DataFrame with book_id and tokens columns.

    Parameters:
    -----------
    df : pandas DataFrame
```

```
        A DataFrame with 'book_id' and 'tokens' columns.
        The 'tokens' column should contain lists of tokens (as strings or␣
↪actual lists).

    Returns:
    --------
    pandas DataFrame
        The original DataFrame with additional columns:
        - term_freq: Dictionary of term frequencies for each token
        - term_counts: Dictionary of raw counts for each token
        - idf: Dictionary of IDF scores for each token
        - tf_idf: Dictionary of TF-IDF scores for each token
    """
    # Create a copy of the DataFrame to avoid modifying the original
    result_df = df.copy()

    # Function to compute term frequency and term counts
    def term_freq_by_doc(list_of_tokens):
        # Handle both string representation of list and actual list
        if isinstance(list_of_tokens, str):
            token_list = eval(list_of_tokens)  # Convert string representation␣
↪to list
        else:
            token_list = list_of_tokens  # Use as is if already a list

        # Count occurrences of each term
        term_counts = Counter(token_list)

        # Total number of terms in the document
        total_terms = len(token_list)

        # Compute TF: term frequency for each token
        term_freq = {term: count / total_terms for term, count in term_counts.
↪items()}

        return term_freq, term_counts

    # Apply function to compute TF for each book
    result_df["term_freq"], result_df["term_counts"] = zip(*result_df["tokens"].
↪apply(term_freq_by_doc))

    # Get total number of documents (books)
    N = len(result_df)

    # Count how many documents contain each term
    doc_containing_term = Counter()
    for term_counts in result_df["term_freq"]:
        doc_containing_term.update(term_counts.keys())  # Count unique terms in␣
↪each document

    # Compute IDF for each term
    idf_scores = {term: np.log(N / (1 + doc_count)) for term, doc_count in␣
↪doc_containing_term.items()}  # Adding 1 to avoid division by zero

    # Add IDF column to df
    result_df["idf"] = result_df["term_freq"].apply(lambda term_freq: {term:␣
↪idf_scores[term] for term in term_freq})

    # Compute TF-IDF by multiplying TF and IDF for each term in each document
    result_df["tf_idf"] = result_df.apply(lambda row: {term:␣
↪row["term_freq"][term] * row["idf"][term] for term in row["term_freq"]},␣
↪axis=1)

    return result_df

df_tfidf_G = calculate_tfidf(df_G)
df_tfidf_G[["book_id","tf_idf"]].head(2)
```

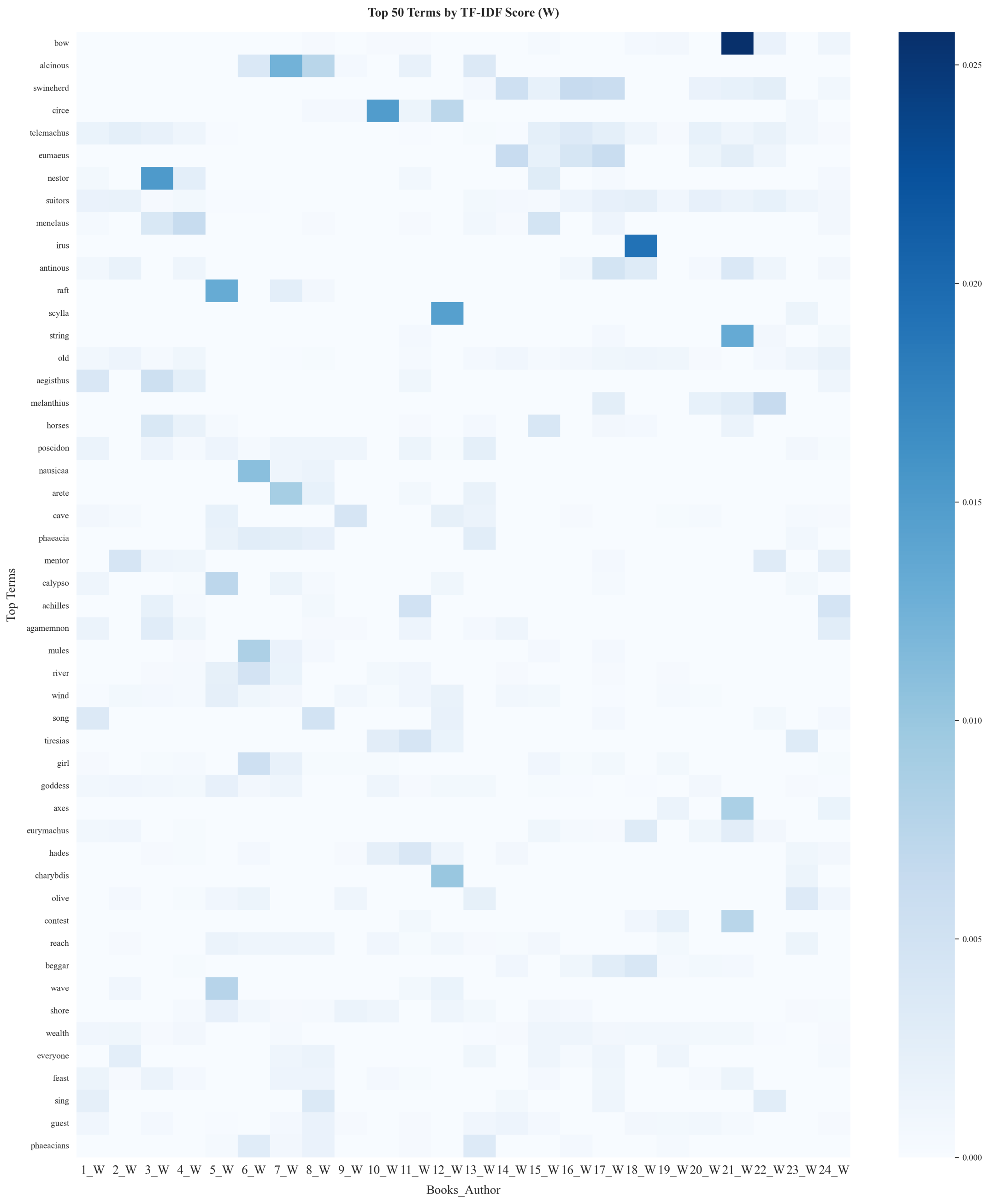
[22]:
```
  book_id                                             tf_idf
0     1_G  {'man': -0.00035908235920594823, 'muse': 0.000…
1     2_G  {'dawn': 4.09818145583013e-05, 'appeared': 8.7…
```

The visualization

[23]:
```
# Top terms for each book and heatmap plot

def extract_top_terms(df, n=50):
    """
    Extract the top N most important terms from the tf_idf column

    Parameters:
    -----------
    df : pandas DataFrame
        DataFrame with 'book_id' and 'tf_idf' columns
    n : int
        Number of top terms to extract (default: 50)

    Returns:
    --------
    tuple
        (top_terms_per_book, top_terms_overall)
        - top_terms_per_book: DataFrame with top terms for each book
        - top_terms_overall: DataFrame with top terms across all books
    """
    # Extract top terms per book
    top_terms_per_book = {}

    for _, row in df.iterrows():
        book_id = row['book_id']
        tf_idf_dict = row['tf_idf']

        # Sort terms by tf-idf score (descending) and take top N
        sorted_terms = sorted(tf_idf_dict.items(), key=lambda x: x[1],␣
↪reverse=True)[:n]
        top_terms_per_book[book_id] = {term: score for term, score in␣
↪sorted_terms}

    # Convert to DataFrame for easier analysis
    top_terms_df = pd.DataFrame.from_dict(top_terms_per_book, orient='index')

    # Extract top terms overall
    all_terms = {}
    for tf_idf_dict in df['tf_idf']:
        for term, score in tf_idf_dict.items():
            if term in all_terms:
                all_terms[term] += score
            else:
                all_terms[term] = score

    # Sort terms by total tf-idf score (descending) and take top N
    top_terms_overall = sorted(all_terms.items(), key=lambda x: x[1],␣
↪reverse=True)[:n]

    # Convert to DataFrame
    top_terms_overall_df = pd.DataFrame(top_terms_overall, columns=['term',␣
↪'total_score'])

    return top_terms_df, top_terms_overall_df

def create_tfidf_heatmap(df, top_n=50):
    """
    Create a heatmap of the top N terms across all books

    Parameters:
    -----------
    df : pandas DataFrame
        DataFrame with 'book_id' and 'tf_idf' columns
    top_n : int
        Number of top terms to include in the heatmap (default: 50)
    """
    # Extract top terms overall
    _, top_terms = extract_top_terms(df, n=top_n)
    top_terms_list = top_terms['term'].tolist()

    # Create a matrix of book_id x top_terms
    heatmap_data = []
    book_ids = []

    for _, row in df.iterrows():
        book_id = row['book_id']
        book_ids.append(book_id)

        tf_idf_dict = row['tf_idf']

        # Extract scores for top terms
        scores = [tf_idf_dict.get(term, 0) for term in top_terms_list]
        heatmap_data.append(scores)

    # Convert to numpy array
    heatmap_array = np.array(heatmap_data).T

    # Create heatmap
    plt.figure(figsize=(14, 16))
    sns.heatmap(heatmap_array, cmap='Blues', xticklabels=book_ids,␣
 ↪yticklabels=top_terms_list)
    plt.title(f'Top {top_n} Terms by TF-IDF Score ({df["book_id"].iloc[0][2:
 ↪]})')
    plt.xlabel('Books_Author')
    plt.ylabel('Top Terms')
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.savefig(f"/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
 ↪MVP-TFIDF_heatmap({df['book_id'].iloc[0][2:]}).png")
    plt.show()

    return heatmap_array

top_terms_per_book_G, top_terms_overall_G = extract_top_terms(df_tfidf_G)
heatmap_array = create_tfidf_heatmap(df_tfidf_G)
```



```
[24]: top_terms_per_book_W, top_terms_overall_W = extract_top_terms(df_W)
      heatmap_array = create_tfidf_heatmap(df_W)
```



Wilson shows similar results to Green, using the step by step approach. Wilson's Odyssey (df) will be processed by the **e2e function**, so both translations have the same processing.

[25]:
```
df_tfidf_W = calculate_tfidf(df_W)
```

[26]:
```
# Getting and then comparing W & G's
# 30 top terms
```

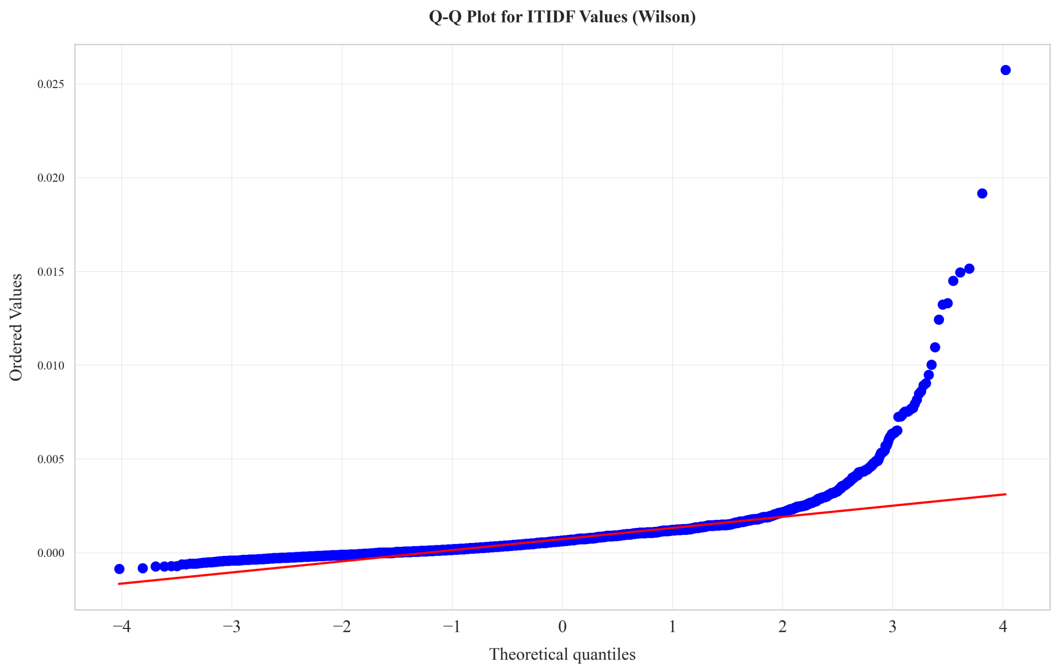
```
tt_W = top_terms_overall_W['term'][:30]
tt_G = top_terms_overall_G['term'][:30]
# New df with the top terms from both for comparison
top_terms_overall = pd.DataFrame({'Wilson': tt_W, 'Green': tt_G})
top_terms_overall.T
```

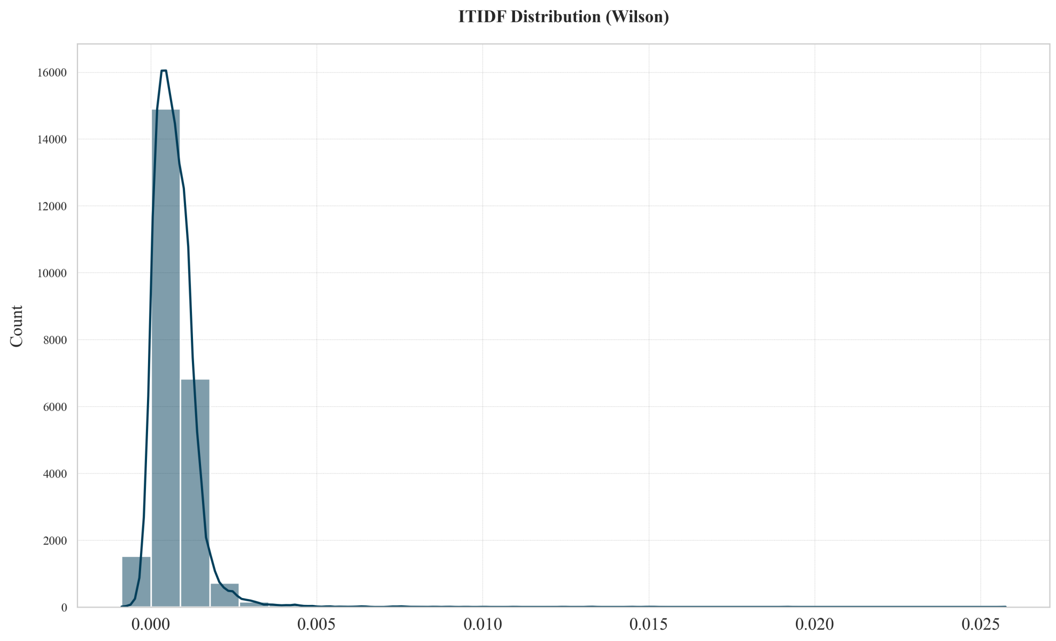
[26]:
```
               0          1          2           3           4        5   \
Wilson       bow  alcinous  swineherd       circe  telemachus  eumaeus
Green   alkinoos        bow  swineherd  telemachos  phaiakians    kirke

            6        7         8         9  …          20      21  \
Wilson  nestor  suitors  menelaus      irus  …       arete    cave
Green   nestor  eumaios   suitors  antinoos  …  melanthios  skylle

              22      23        24        25         26          27     28  \
Wilson  phaeacia  mentor   calypso  achilles  agamemnon       mules  river
Green   poseidon   arete  stranger    atreus   penelope  eurymachos   bend

               29
Wilson       wind
Green   teiresias

[2 rows x 30 columns]
```

- **Parametric Hypothesis Testing:** Assessing significant variations.
  Checking at the word by word comparison the similarities in choice of words is apparent. Thus the next step will be testing, but before chosing the test we need to check if the data distribution is normal.

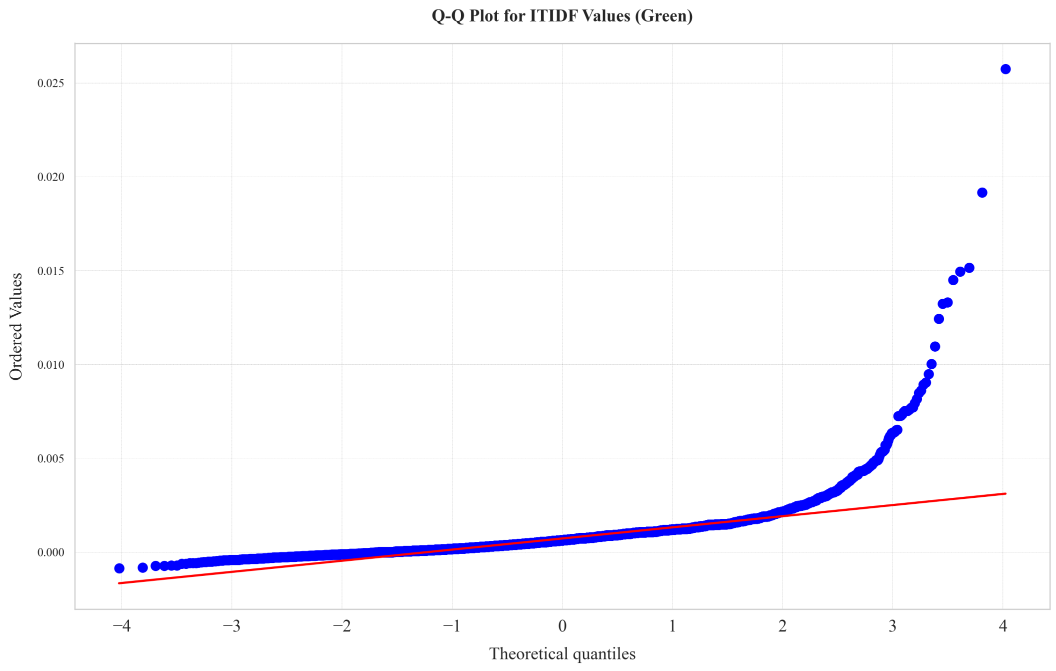
[27]:
```
# Calculate the sum of the tf-idf scores for each translation
itidf_W = df_tfidf_W["tf_idf"].apply(lambda x: list(x.values())).sum()
itidf_G = df_tfidf_G["tf_idf"].apply(lambda x: list(x.values())).sum()
```

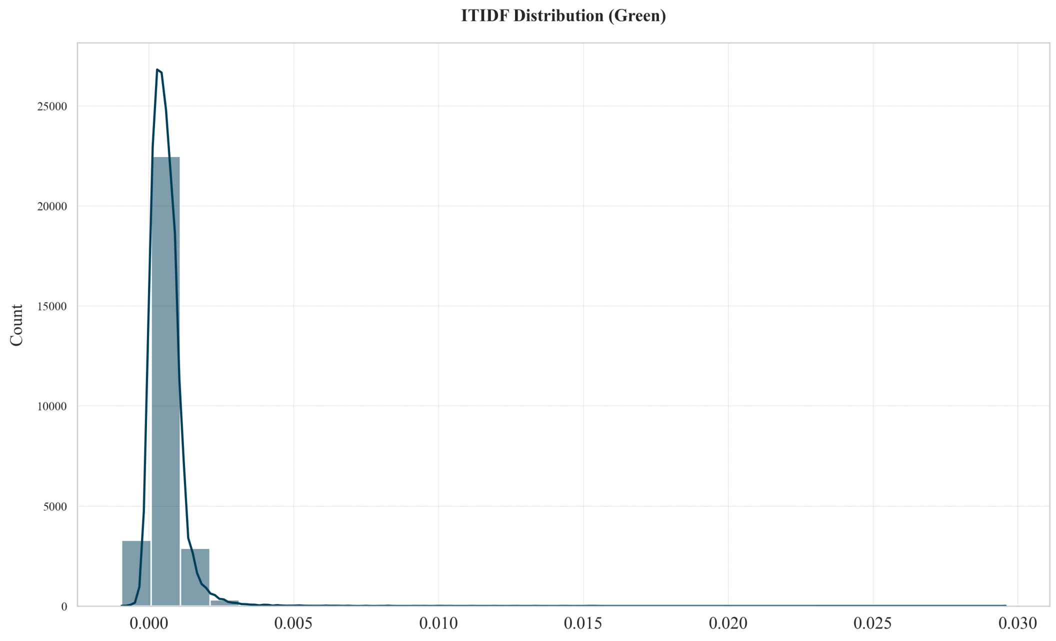
[28]:
```
import scipy.stats as stats
# Q-Q plot for ITIDF values
def plot_qq(itidf, translator="Wilson"):
    """
    Create a Q-Q plot for the ITIDF values.

    Parameters:
    -----------
    itidf : list
        List of ITIDF values to plot
    """
    # Generate a Q-Q plot
    stats.probplot(itidf_W, dist="norm", plot=plt)
    plt.title(f"Q-Q Plot for ITIDF Values ({translator})")
    plt.savefig(f"/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
↪Q-Q_Plot_{translator}.png")
    plt.show()

def plot_itidf_distribution(itidf, translator="Wilson"):
    """
    Plot the distribution of ITIDF values.

    Parameters:
    -----------
    itidf : list
        List of ITIDF values to plot
    """
    # Create a histogram of the ITIDF values
    sns.histplot(itidf, bins=30, kde=True)  # KDE adds a smoothed curve
    plt.title(f"ITIDF Distribution ({translator})")
    plt.savefig(f"/Users/debr/English-Homer/MVP_Green-Wilson/MVP_plots/
↪ITIDF_Distribution_{translator}.png")
    plt.show()

# Plot Q-Q plot and distribution for ITIDF values for Wilson
plot_qq(itidf_W, "Wilson")
plot_itidf_distribution(itidf_W, "Wilson")
```





```
[29]: # Plot Q-Q plot and distribution for ITIDF values for Green
      plot_qq(itidf_G, "Green")
      plot_itidf_distribution(itidf_G, "Green")
```





[30]:
```
from scipy.stats import shapiro
# Shapiro test for normality
def shapiro_test(data):
    """
    Perform the Shapiro-Wilk test for normality from scipy.stats and print the␣
 ↪results.

    Parameters:
    -----------
    data : list or array
        The data to be tested for normality.

    Returns: shapiro statistic and p-value

    """
    stat, p = shapiro(data)
    print(f"Shapiro-Wilk Test: p-value = {p}")

    if p > 0.05:
        print("Data appears to be normally distributed (fail to reject H0).")
    else:
        print("Data is not normally distributed (reject H0).")
    return stat, p

# Shapiro-Wilk test for normality
print("Wilson")
shapiro_test(itidf_W)
print("\n")
print("Green")
shapiro_test(itidf_G)
```

```
Wilson
Shapiro-Wilk Test: p-value = 1.9264430476087096e-105
Data is not normally distributed (reject H0).


Green
Shapiro-Wilk Test: p-value = 1.5939662570478783e-113
Data is not normally distributed (reject H0).
```

```
[30]: (np.float64(0.6885523258795925), np.float64(1.5939662570478783e-113))
```

Why the Mann-Whitney U Test

Since the data for the TF-IDF values is not normally distributed, a parametric test like the t-test is not appropriate. Instead, the Mann-Whitney U test is used, which is a non-parametric test for comparing two independent samples. It does not require the assumption of normality and is suitable for comparing distributions when the data is skewed or has outliers.

Hypotheses:

- Null Hypothesis (H): There is no significant difference in the TF-IDF values between the two translations (Wilson and Green).
- Alternative Hypothesis (H): There is a significant difference in the TF-IDF values between the two translations.

The Mann-Whitney U test evaluates whether the distributions of TF-IDF values for the two translations differ, considering their ranks rather than specific values.

```
[31]: # Mannwhitney U test for comparing ITIDF values
from scipy.stats import mannwhitneyu  # Import the actual function

def mannwhitneyu_test(x, y, alternative='two-sided'):
    """
    Perform the Mann-Whitney U test for comparing two independent samples.
    """
    stat, p = mannwhitneyu(x, y, alternative=alternative)  # Use SciPy's␣
 ↪function
    print(f"Mann-Whitney U test statistic: {stat}, p-value: {p}")

    if p < 0.05:
        print("Reject H : The distributions of the translations are␣
↪significantly different.")
    else:
        print("Fail to reject H : No significant difference between the␣
↪translations.")

# Run the test
mannwhitneyu_test(itidf_W, itidf_G, alternative='two-sided')
```

```
Mann-Whitney U test statistic: 398451440.5, p-value: 2.3169071909419926e-146
Reject H : The distributions of the translations are significantly different.
```

```
[ ]:
```

2.10 5 Discussion of Results

After the three experiments and various statistical tests for each, it has been stablished, with statistical certainty, that lexicaly the translations maintain and analogous diversity. this fact supports the argument of a strong cultural sway in their translations. The sample of the present MVP analysis is too small to conclude with certainty anything, however, it proves that the method works and yields interesting insights.

2.10.1 Findings: Insights from experiments.

-
-
-
- Explanation: • Test Statistic: The Mann-Whitney U test statistic of 398451440.5 is the computed value from comparing the two distributions. • P-value: The p-value of 2.3169071909419926e-146 is extremely small (much smaller than the typical threshold of 0.05). This indicates strong evidence against the null hypothesis (H).

Conclusion:

Since the p-value is significantly less than 0.05, we reject the null hypothesis (H). This means that the result supports the alternative hypothesis (H), which states that the distributions of TF-IDF values for the two translations are significantly different.

Therefore, the result is consistent with the hypothesis that there is a significant difference between the two translations' TF-IDF values.

2.10.2 New Directions: Refinements & future research steps.