EDA Epubs

Name: **Merijn van der Leek**

StudentID: **12870862**

Email: [**merijn.van.der.leek@student.uva.nl**]

Submitted on: **22-3-2024**

Github: https://github.com/merijn212/Thesis-2024-Data-Science/

Data loading

First, we need to install the necessary libraries to run our script effectively:

In [1]:
```
import os
import subprocess
import pandas as pd
import json
import seaborn as sns
import ebooklib
from ebooklib import epub
import epub_meta
import matplotlib.pyplot as plt
import zipfile
from bs4 import BeautifulSoup
import warnings
warnings.filterwarnings('ignore')
```

After gathering a database of EPUB files from various sites mostly found on https://www.reddit.com/r/FreeEBOOKS/. This dataset is not final and is updated daily. Let's examine the size of this collection. The following script counts the number of EPUB files in the specified directory and selects one of them for further inspection:

In [43]:
```
# Path to the directory where your ePub files are stored
epub_folder = 'C:\\Users\\vande\\Downloads\\epubs'  # Change this to the actual path

# Function to list all ePub files in the directory
def list_epub_files(directory):
    epub_files = [file for file in os.listdir(directory) if file.endswith('.epub')]
    return epub_files

# Calling the function and printing the list of ePub files
epub_files = list_epub_files(epub_folder)
print("Found ePub files:", len(epub_files))

# Ensure the list is not empty and then select the first ePub file
if epub_files:
    selected_epub_file = epub_files[0]  # Select the first file for demonstration
    print(f"Selected ePub file for inspection: {selected_epub_file}")
else:
    print("No ePub files found in the specified directory.")
```

```
Found ePub files: 103
Selected ePub file for inspection:  - An Accidental Diplomat. My Years in the Irish Foreign Service 1987-95-ePu
b Direct_New Island Publishing_NewIsland (2013).epub
```

This code snippet is designed to give us a closer look into the structure of an EPUB file by extracting and displaying its contents. An EPUB is essentially a zip archive containing XHTML files (which form the textual content of the book), CSS (which styles the content), and usually images and other media that are part of the book's layout:

In [44]:
```
def extract_and_show_content(epub_path):
    # Using zipfile to open and extract the ePub content
    with zipfile.ZipFile(epub_path, 'r') as zip_ref:
        # Extract to a temporary directory
        extract_dir = os.path.join('temp_epub_extract', os.path.basename(epub_path).replace('.epub', ''))
        zip_ref.extractall(extract_dir)

        # Search for HTML or XHTML files without assuming specific directory names
        for root, dirs, files in os.walk(extract_dir):
            html_files = [f for f in files if f.endswith('.html') or f.endswith('.xhtml')]
            if html_files:
                # Just read the first HTML file for demonstration
                html_path = os.path.join(root, html_files[0])
                with open(html_path, 'r', encoding='utf-8') as file:
                    html_content = file.read()
                    print(html_content[:1000])  # Print the first 1000 characters of the HTML content
                break
```

```
        else:
            print("No HTML content found.")

epub_path = os.path.join(epub_folder, selected_epub_file)
extract_and_show_content(epub_path)
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Acknowledgements</title>
  <link href="../Styles/template.css" rel="stylesheet" type="text/css"/>
  <meta content="application/xhtml+xml; charset=UTF-8" http-equiv="Content-Type"/>
  <style type="text/css">
/*<![CDATA[*/

  li.sgc-1 {list-style: none; display: inline}
  /*]]>*/
  </style>
<meta content="urn:uuid:050545b7-33f4-4596-8f34-90dc3d880b72" name="Adept.expected.resource"/>
</head>

<body>



<p class="txs1"><b>Eamon Delaney </b>lives in Dublin where he is an author and freelance journalist, contributi
ng to a variety of newspapers, magazines and radio programmes. In 1995, he published a novel, <i>The Casting of
Mr O’Shaughnessy</i>. He is presently working on another but may yet come to regret the forfeit of a Civil Serv
ice pension.</p>
<p class="centh"><b>Acknowledgements</b></p>
<p class="txs">My thanks to those who read the book in its early stages, including David Murphy, Mary Rose Door
ly, John Ryan – who suggested
```

This might not align with the initial expectations surrounding electronic publications; however, the nature of EPUB files becomes more apparent when we examine the content of the first HTML file within such an ebook:

In [46]:
```
def get_html_files_content(epub_path, num_files):
    contents = []
    with zipfile.ZipFile(epub_path, 'r') as zip_ref:
        extract_dir = os.path.join('temp_epub_extract', os.path.basename(epub_path).replace('.epub', ''))
        zip_ref.extractall(extract_dir)
        # Make sure that number of files is not bigger than num_files
        html_files = []
        for root, dirs, files in os.walk(extract_dir):
            for file in files:
                if file.endswith('.html') or file.endswith('.xhtml'):
                    html_files.append(os.path.join(root, file))
                    if len(html_files) >= num_files:
                        break
            if len(html_files) >= num_files:
                break

        for html_file in html_files:
            with open(html_file, 'r', encoding='utf-8') as f:
                soup = BeautifulSoup(f.read(), 'html.parser')
                text = soup.get_text(separator='\n', strip=True)
                contents.append(text)
    return contents

contents = get_html_files_content(epub_path, 1)  # We are collecting just the first file

for i, content in enumerate(contents, 1):
    print(f"Content of file {i}:\n{content}\n\n---\n")
```

```
Content of file 1:
Acknowledgements
Eamon Delaney
lives in Dublin where he is an author and freelance journalist, contributing to a variety of newspapers, magazi
nes and radio programmes. In 1995, he published a novel,
The Casting of Mr O’Shaughnessy
. He is presently working on another but may yet come to regret the forfeit of a Civil Service pension.
Acknowledgements
My thanks to those who read the book in its early stages, including David Murphy, Mary Rose Doorly, John Ryan –
who suggested the title – and Joe Joyce. Thanks also to my sister, Catherine, and to the many people who backed
up facts and stories, especially Dermot McEvoy and Patrick Farrelly in New York, and Ray O’Hanlon of the
Irish Echo
. Special thanks must go to all at New Island Books, especially Ciara Considine and Edwin Higel for their help
and perseverance and Joseph Hoban for publicity. And to Roddy Flynn, the editor, whose forensic eye and surgica
l suggestions did much to improve the clarity of the story. I also am grateful to Barry Lyons, solicitor, for l
egal advice and to Micheal O’Higgins, S.C. and Ronan Munro, who read the text at different stages.

---
```

Running ace

To detect accessibility issues within EPUB files, the Daisy Ace Checker is commonly employed. Let's apply Ace to analyze our collection

of EPUBs:

In [47]:
```
%%capture
def check_epubs_in_directory(directory):
    # Identify EPUB files within the specified directory
    epub_files = [file for file in os.listdir(directory) if file.lower().endswith('.epub')]

    for epub in epub_files:
        process_epub_file(directory, epub)

def process_epub_file(directory, epub_file):
    epub_path = os.path.join(directory, epub_file)
    result_directory = os.path.join(directory, 'ace_checks', epub_file.split('.')[0])

    if not os.path.exists(result_directory):
        os.makedirs(result_directory)
    #run Ace on our epubs and show if the test failed or succeeded
    ace_command = f'ace -o "{result_directory}" "{epub_path}"'
    print(f'Executing: {ace_command}')

    try:
        subprocess.run(ace_command, shell=True, check=True)
        print(f'Successfully checked: {epub_file}')
    except subprocess.CalledProcessError as error:
        print(f'Failed to check {epub_file}: {error}')

if __name__ == '__main__':
    epub_folder_path = 'C:\\Users\\vande\\Downloads\\epubs'
    check_epubs_in_directory(epub_folder_path)
```

To compile and analyze accessibility error reports from EPUB files, we first locate all 'report.json' files in a specified directory. After gathering these reports, we extract their metadata and errors, organizing this information into dataframes for comprehensive analysis. This process involves assessing error types, grouping errors by impact and title, and evaluating errors per EPUB. Additionally, we merge metadata with error details for an in-depth overview and categorize errors by publisher, enabling targeted insights. The findings are summarized and saved into an Excel file for easy access and reference.

In [48]:
```
def find_ace_report_files(base_directory):
    found_reports = []
    for current_path, directories, files in os.walk(base_directory):
        for file_name in files:
            if file_name.lower() == 'report.json':  # Ensures case-insensitive match
                full_path = os.path.join(current_path, file_name)
                found_reports.append(full_path)
    return found_reports

ace_results_dir = 'C:\\Users\\vande\\Downloads\\epubs'
report_paths = find_ace_report_files(ace_results_dir)

def process_reports(report_paths):
    all_metadata, all_errors = [], []

    # Process each report and collect data
    for path in report_paths:
        with open(path, encoding='utf-8') as f:
            data = json.load(f)
        epub_id = os.path.basename(os.path.dirname(path))

        # Collect metadata
        metadata = data['earl:testSubject']['metadata']
        metadata['epubID'] = epub_id
        all_metadata.append(metadata)

        # Collect errors
        errors = [dict(result['earl:test'], epubID=epub_id)
                  for assertion in data.get('assertions', [])
                  for result in assertion.get('assertions', [])
                  if result['earl:result']['earl:outcome'] == 'fail']
        all_errors.extend(errors)

    # Convert lists to DataFrames
    metadata_df = pd.DataFrame(all_metadata).set_index('epubID')
    errors_df = pd.DataFrame(all_errors)

    return metadata_df, errors_df

def analyze_errors(errors_df):
    error_impact_counts = errors_df['earl:impact'].value_counts()
    errors_grouped = errors_df.groupby(['earl:impact', 'dct:title']).size()
    errors_per_epub = errors_df.groupby('epubID').size()

metadata_df, errors_df = process_reports(report_paths)

# Analysis and saving results
analyze_errors(errors_df)
```

```
# Merging metadata with errors for a comprehensive overview and further analysis
combined_df = pd.merge(errors_df, metadata_df, on='epubID')

# Filtering, analyzing, and summarizing errors by publisher
publisher_analysis = combined_df.dropna(subset=['dc:publisher'])
publisher_analysis['dc:publisher'] = publisher_analysis['dc:publisher'].astype(str)
errors_per_publisher = publisher_analysis.groupby('dc:publisher')['earl:impact'].value_counts().unstack(fill_va
errors_summary = errors_per_publisher.sum(axis=1).describe()

# Save to Excel and print summary
errors_per_publisher.to_excel('ErrorsPerPublisher.xlsx')
errors_df
```

Out[48]:

	earl:impact	dct:title	dct:description	help	rulesetTags	epubID
0	serious	epub-lang	Ensures the OPF XML language is provided	{'url': 'http://kb.daisy.org/publishing/docs/e...	[EPUB]	- An Accidental Diplomat
1	serious	metadata-accessmode	Ensures a 'schema:accessMode' metadata is present	{'url': 'http://kb.daisy.org/publishing/docs/m...	[EPUB]	- An Accidental Diplomat
2	moderate	metadata-accessmodesufficient	Ensures a 'schema:accessModeSufficient' metada...	{'url': 'http://kb.daisy.org/publishing/docs/m...	[EPUB]	- An Accidental Diplomat
3	serious	metadata-accessibilityfeature	Ensures a 'schema:accessibilityFeature' metada...	{'url': 'http://kb.daisy.org/publishing/docs/m...	[EPUB]	- An Accidental Diplomat
4	serious	metadata-accessibilityhazard	Ensures a 'schema:accessibilityHazard' metadat...	{'url': 'http://kb.daisy.org/publishing/docs/m...	[EPUB]	- An Accidental Diplomat
...	...	...	...	...	...	...
11949	serious	document-title	Ensures each HTML document contains a non-empt...	{'url': 'http://kb.daisy.org/publishing/docs/h...	[cat.text-alternatives, wcag2a, wcag242, ACT, ...	_onz022191301_01
11950	serious	html-has-lang	Ensures every HTML document has a lang attribute	{'url': 'http://kb.daisy.org/publishing/docs/h...	[cat.language, wcag2a, wcag311, ACT, TTv5, TT1...	_onz022191301_01
11951	serious	document-title	Ensures each HTML document contains a non-empt...	{'url': 'http://kb.daisy.org/publishing/docs/h...	[cat.text-alternatives, wcag2a, wcag242, ACT, ...	_onz022191301_01
11952	serious	html-has-lang	Ensures every HTML document has a lang attribute	{'url': 'http://kb.daisy.org/publishing/docs/h...	[cat.language, wcag2a, wcag311, ACT, TTv5, TT1...	_onz022191301_01
11953	serious	html-has-lang	Ensures every HTML document has a lang attribute	{'url': 'http://kb.daisy.org/publishing/docs/h...	[cat.language, wcag2a, wcag311, ACT, TTv5, TT1...	_onz022191301_01

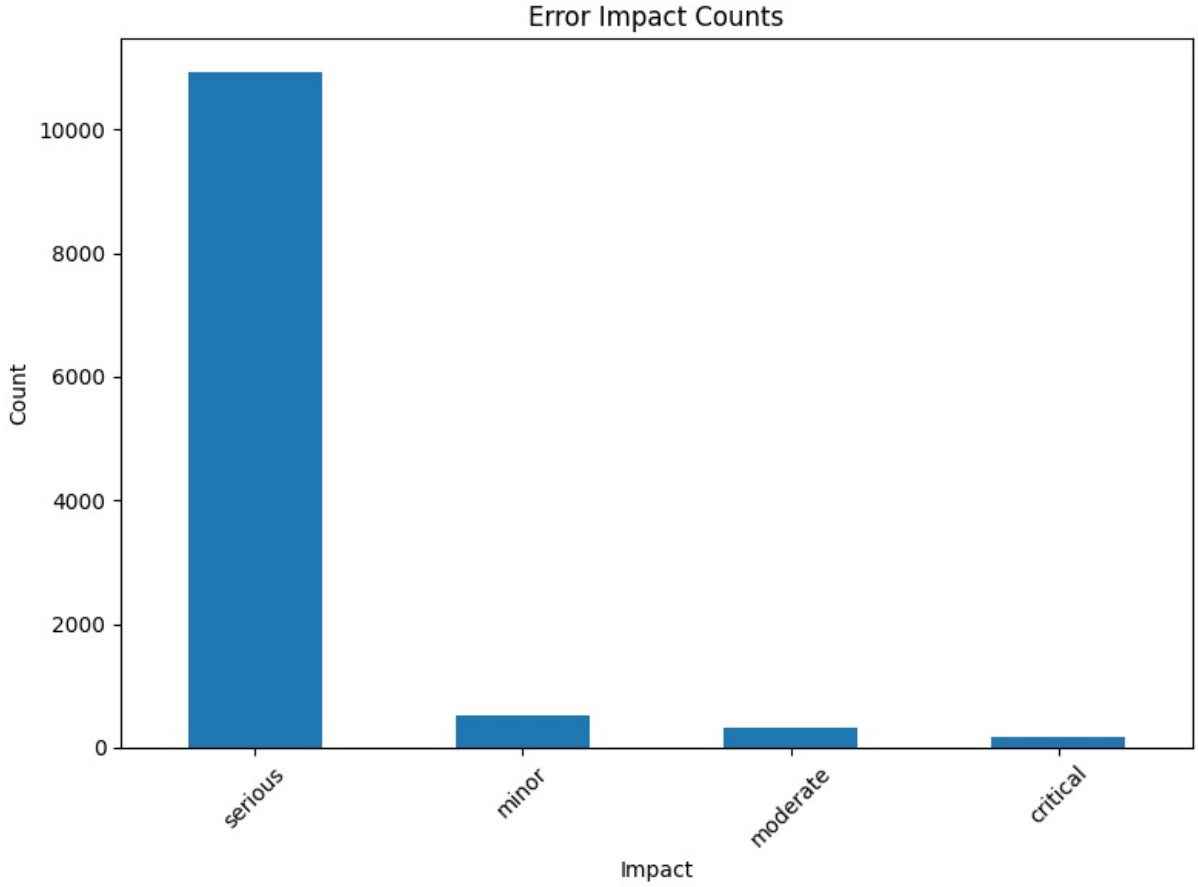
11954 rows × 6 columns

Let's visualize the data from our dataset to identify prevalent errors and pinpoint which publishers and books are most affected by them. Through these visualizations, we aim to uncover the most common types of errors, as well as highlight the books and publishers that exhibit the highest number of issues:

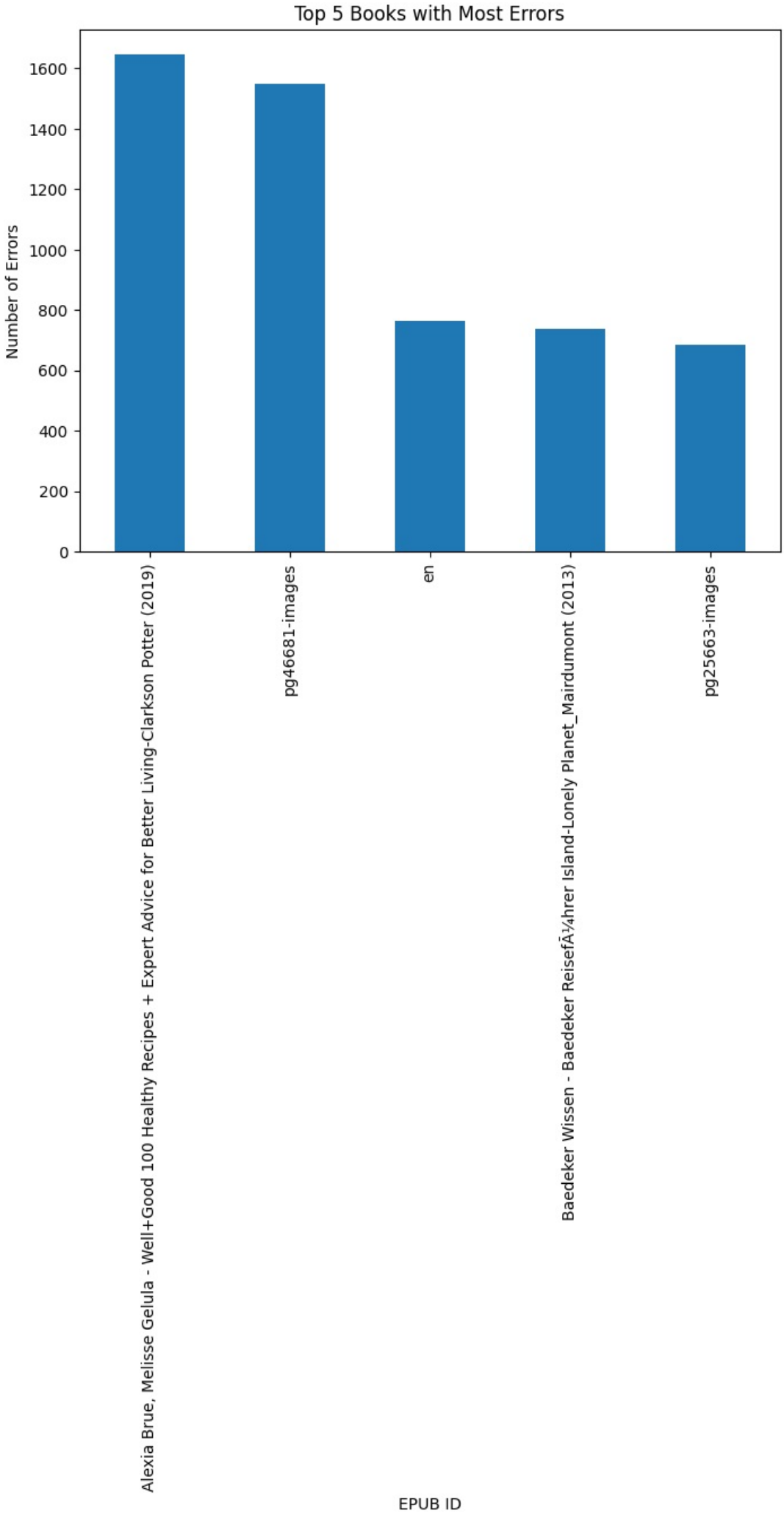
In [51]:

```
def plot_error_impact_counts(error_impact_counts):
    plt.figure(figsize=(8, 6))  # Set the figure size here to match the others
    error_impact_counts.plot(kind='bar')
    plt.title('Error Impact Counts')
    plt.xlabel('Impact')
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

def plot_top_5_errors_per_epub(errors_per_epub):
    top_errors_per_epub = errors_per_epub.sort_values(ascending=False).head(5)
    fig, ax = plt.subplots(figsize=(8, 6))
    top_errors_per_epub.plot(kind='bar', ax=ax)
    ax.set_title('Top 5 Books with Most Errors')
    ax.set_xlabel('EPUB ID')
    ax.set_ylabel('Number of Errors')
    ax.set_xticklabels(top_errors_per_epub.index, rotation=90, ha="center")
    plt.tight_layout()
    plt.show()

def plot_top_5_errors_per_publisher(errors_per_publisher):
```

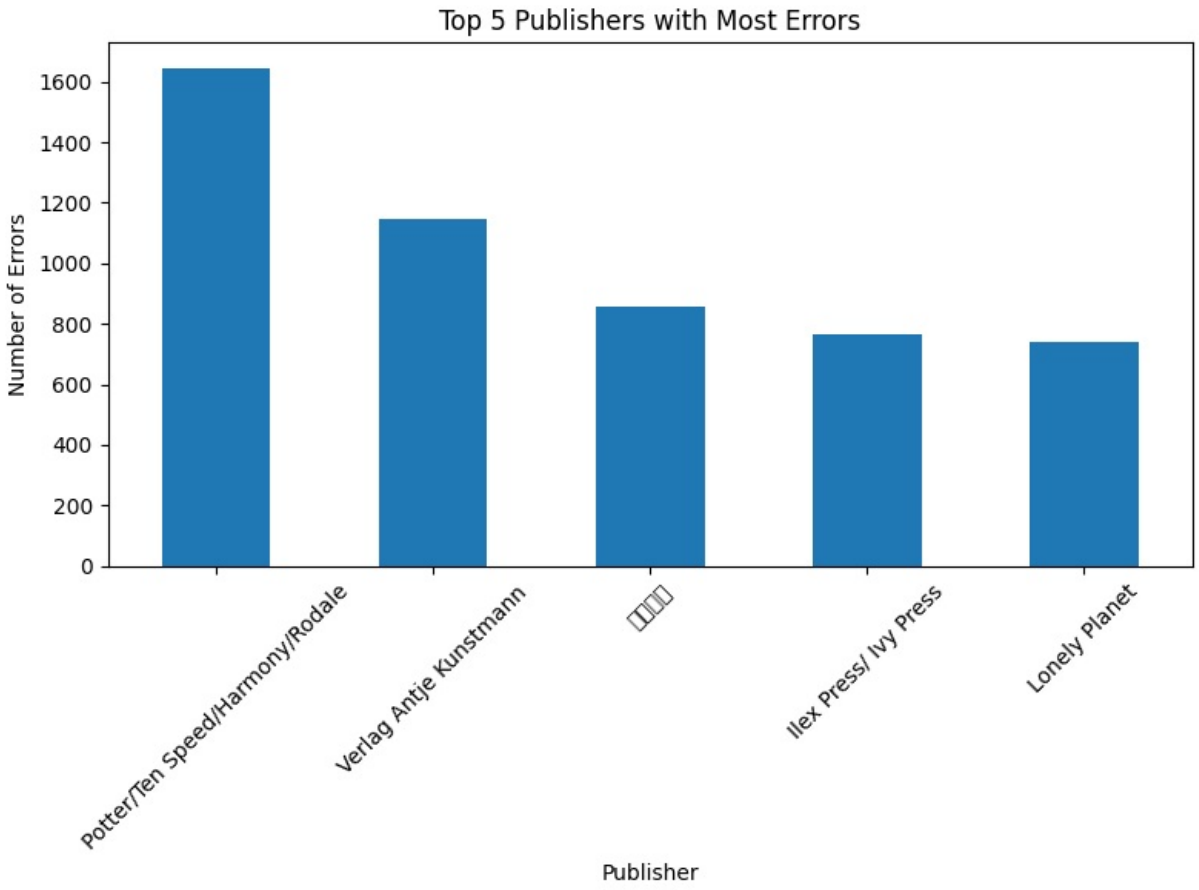
```
    top_errors_per_publisher = errors_per_publisher.sum(axis=1).sort_values(ascending=False).head(5)
    fig, ax = plt.subplots(figsize=(8, 6))
    top_errors_per_publisher.plot(kind='bar', ax=ax)
    ax.set_title('Top 5 Publishers with Most Errors')
    ax.set_xlabel('Publisher')
    ax.set_ylabel('Number of Errors')
    ax.set_xticklabels(top_errors_per_publisher.index, rotation=45)
    plt.tight_layout()
    plt.show()


error_impact_counts = errors_df['earl:impact'].value_counts()
plot_error_impact_counts(error_impact_counts)
```



In [52]:
```
errors_per_epub = errors_df.groupby('epubID').size()
plot_top_5_errors_per_epub(errors_per_epub)
```



In [53]:
```
plot_top_5_errors_per_publisher(errors_per_publisher)
```



In [54]:

```python
def analyze_and_visualize_high_priority_errors(errors_df):
    # Filter errors classified as critical or serious
    high_priority_errors = errors_df[errors_df['earl:impact'].isin(['critical', 'serious'])]

    # Analyze the most common high-priority errors
    common_errors = high_priority_errors['dct:title'].value_counts().head(10)

    plt.figure(figsize=(10, 6))
    common_errors.plot(kind='barh', color='teal')
    plt.title('Top 10 High-Priority Errors')
    plt.xlabel('Count')
    plt.ylabel('Error Title')
    plt.gca().invert_yaxis()
    plt.tight_layout()
    plt.show()


analyze_and_visualize_high_priority_errors(errors_df)
```

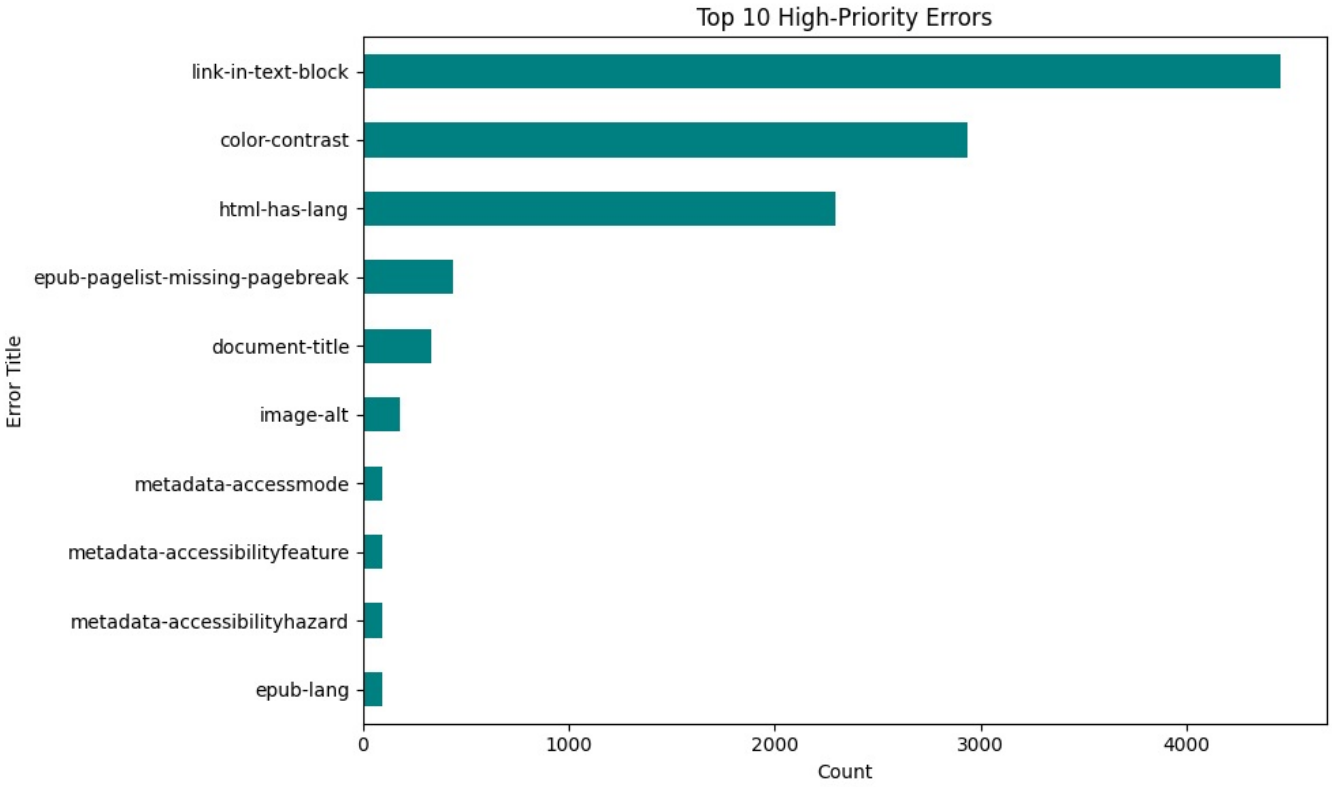


Image-alt & html-has-lang errors

In the plot above we see a large amount of errors related to 'image-alt' and 'html-has-lang'. The 'image-alt' error indicates missing alternative text for images, which is crucial for accessibility, especially for users with visual impairments. Alternative text describes the content and function of images, allowing screen readers and other assistive technologies to convey this information to users. The 'html-has-lang' error signifies missing or incorrect language declarations in HTML documents, which can also impact user experience by affecting how content is interpreted by browsers and assistive technologies. In my research I want to focus on fixing these issues, let's have a look at which epub's have these error's:

In [55]:
```
# Get the epubs where the error is 'image-alt'
image_alt_errors = errors_df[errors_df['dct:title'] == 'image-alt']['epubID'].unique()
html_has_lang_errors = errors_df[errors_df['dct:title'] == 'html-has-lang']['epubID'].unique()

# Convert the lists to DataFrames for nicer display
image_alt_errors_df = pd.DataFrame(image_alt_errors, columns=['EPUBs with Image-Alt Error'])
html_has_lang_errors_df = pd.DataFrame(html_has_lang_errors, columns=['EPUBs with HTML-Has-Lang Error'])


print("First 5 files with 'image-alt' error:")
display(image_alt_errors_df[:5])

print("\n First 5 files with 'html-has-lang' error:")
display(html_has_lang_errors_df[:5])
```

```
First 5 files with 'image-alt' error:
```

	EPUBs with Image-Alt Error
0	EdiciÃ³n digital como metodologÃ_a para una ed...
1	Gary Chapman - The 5 Love Languages_ The Secre...
2	Harry Mulisch - De elementen-De Bezige Bij (1988)
3	nton - Absolute Power
4	pg73219-images

```
 First 5 files with 'html-has-lang' error:
```

	EPUBs with HTML-Has-Lang Error
0	- An Accidental Diplomat
1	Abaco Islands of the Bahamas
2	T
3	Anna Barnes - 50 Tips to Build Your Confidence...
4	Anna Barnes - 50 Tips to Build Your Self-estee...

Next, we'll dive deeper to uncover the specific names of images lacking alternative text within these EPUB files. Additionally, we'll explore examples of alternative text already provided for some images, offering insight into how alt text is used when available:

In [56]:
```
%%capture
def find_images_with_and_without_alt(epub_path):
    images_info = {'without_alt': [], 'with_alt': []}
    try:
        with zipfile.ZipFile(epub_path, 'r') as epub:
            for file_name in epub.namelist():
                if file_name.endswith('.html') or file_name.endswith('.xhtml'):
                    html_content = epub.read(file_name)
                    soup = BeautifulSoup(html_content, 'html.parser')
                    # Finding images without alt text
                    for img in soup.find_all('img', alt=False):
                        images_info['without_alt'].append({'file': file_name, 'image': img.get('src')})
                    # Finding images with alt text (limited to a sample for demonstration)
                    for img in soup.find_all('img', alt=True):
                        if img.get('alt'):  # Ensures alt attribute is not empty
                            images_info['with_alt'].append({'file': file_name, 'image': img.get('src'), 'alt':
                            if len(images_info['with_alt']) >= 5:  # Limit to first 5 examples
                                break
    except zipfile.BadZipFile:
        print(f"Skipping invalid EPUB file: {epub_path}")
    return images_info

def epub_image_alt_sample_report(epub_dir):
    report = []
    epub_files = [f for f in os.listdir(epub_dir) if f.endswith('.epub')]
    for epub_file in epub_files:
        epub_path = os.path.join(epub_dir, epub_file)
        images_info = find_images_with_and_without_alt(epub_path)
        if images_info['without_alt'] or images_info['with_alt']:
            report.append({
                'epub': epub_file,
                'images_info': images_info
            })
    return report

# Example usage
epub_dir = 'C:\\Users\\vande\\Downloads\\epubs'
report = epub_image_alt_sample_report(epub_dir)

# Normalizing and displaying the report for images with and without alt text
df_report_with_alt = pd.json_normalize(report, record_path=['images_info', 'with_alt'], meta=['epub'], errors='
df_report_without_alt = pd.json_normalize(report, record_path=['images_info', 'without_alt'], meta=['epub'], er
```

In [58]:
```
print("Images with alt text:")
display(df_report_with_alt.head())
print("Images without alt text:")
display(df_report_without_alt.head())
```

Images with alt text:

	file	image	alt	epub
0	OEBPS/Text/titlepage.xhtml	../Images/logo.jpg	logo	- An Accidental Diplomat. My Years in the Iri...
1	OEBPS/Text/Copyright.xhtml	../Images/AC_Logo_fmt.jpeg	artscouncil2.tif	- An Accidental Diplomat. My Years in the Iri...
2	OEBPS/cher_9781593764968_oeb_c20_r1.html	cher_9781593764968_oeb_083_r1.jpg	083	T. Cohen-Greene, Lorna Garano - An intimate l...
3	OEBPS/cher_9781593764968_oeb_c20_r1.html	cher_9781593764968_oeb_084_r1.jpg	084	T. Cohen-Greene, Lorna Garano - An intimate l...
4	OEBPS/cher_9781593764968_oeb_c20_r1.html	cher_9781593764968_oeb_085_r1.jpg	085	T. Cohen-Greene, Lorna Garano - An intimate l...

Images without alt text:

	file	image	epub
0	OPS/xhtml/000-portada.xhtml	../img/portada.jpg	EdiciÃ³n digital como metodologÃ_a para una ed...
1	The_5_Love_Languages_The_Secret_split_010.html	images/00002.jpg	Gary Chapman - The 5 Love Languages_ The Secre...
2	The_5_Love_Languages_The_Secret_split_013.html	images/00002.jpg	Gary Chapman - The 5 Love Languages_ The Secre...
3	The_5_Love_Languages_The_Secret_split_016.html	images/00002.jpg	Gary Chapman - The 5 Love Languages_ The Secre...
4	The_5_Love_Languages_The_Secret_split_020.html	images/00002.jpg	Gary Chapman - The 5 Love Languages_ The Secre...

In the upcoming code segment, we will identify HTML files flagged for missing lang attributes of our selected file and read the content of one such file to understand the implications of this error:

In [59]:
```
%%capture
def scan_html_for_lang_attribute(epub_path):
    missing_lang_files = []
    try:
        with zipfile.ZipFile(epub_path) as epub:
            html_files = [f for f in epub.namelist() if f.endswith(('.html', '.xhtml'))]
            for file_name in html_files:
                with epub.open(file_name) as file:
                    soup = BeautifulSoup(file.read(), 'html.parser')
```

```
                    if not soup.html or not soup.html.has_attr('lang'):
                        missing_lang_files.append(file_name)
    except zipfile.BadZipFile:
        print(f"Cannot process {epub_path}, not a valid EPUB.")
    return missing_lang_files

def generate_lang_attribute_report(directory):

    epub_files = [os.path.join(directory, file) for file in os.listdir(directory) if file.endswith('.epub')]
    report_items = [{'epub': os.path.basename(epub), 'missing_lang_files': scan_html_for_lang_attribute(epub)}
                    for epub in epub_files if scan_html_for_lang_attribute(epub)]
    return report_items

epub_directory = 'C:\\Users\\vande\\Downloads\\epubs'

# Generate and display the report
lang_report = generate_lang_attribute_report(epub_directory)
df_lang_report = pd.json_normalize(lang_report, 'missing_lang_files', ['epub'])
```

In [60]:
```
df_lang_report.columns = ['XHTML File', 'EPUB']
display(df_lang_report.head())
df_lang_report['EPUB'][0]
```

	XHTML File	EPUB
0	OEBPS/Text/chap16.xhtml	- An Accidental Diplomat. My Years in the Iri...
1	OEBPS/Text/chap13.xhtml	- An Accidental Diplomat. My Years in the Iri...
2	OEBPS/Text/chap02.xhtml	- An Accidental Diplomat. My Years in the Iri...
3	OEBPS/Text/titlepage.xhtml	- An Accidental Diplomat. My Years in the Iri...
4	OEBPS/Text/Glossary.xhtml	- An Accidental Diplomat. My Years in the Iri...

Out[60]:
```
' - An Accidental Diplomat. My Years in the Irish Foreign Service 1987-95-ePub Direct_New Island Publishing_New Island (2013).epub'
```

In [61]:
```
def display_lang_mistake_example(epub_path, html_file):
    try:
        with zipfile.ZipFile(epub_path, 'r') as epub:
            with epub.open(html_file) as file:
                soup = BeautifulSoup(file.read(), 'html.parser')
                html_tag = soup.find('html')
                # Display the <html> tag or the beginning of the file if <html> is not found
                if html_tag:
                    print(str(html_tag)[:500])  # Print the first 500 characters of the <html> tag
                else:
                    print("No <html> tag found in the file.")
    except zipfile.BadZipFile:
        print(f"Cannot process {epub_path}, not a valid EPUB.")
    except KeyError:
        print(f"The file {html_file} does not exist in {epub_path}.")

epub_example = 'C:\\Users\\vande\\Downloads\\epubs\\ - An Accidental Diplomat. My Years in the Irish Foreign Se
html_example = 'OEBPS/Text/chap16.xhtml'

display_lang_mistake_example(epub_example, html_example)
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Chapter 16</title>
<link href="../Styles/template.css" rel="stylesheet" type="text/css"/>
<meta content="application/xhtml+xml; charset=utf-8" http-equiv="Content-Type"/>
<style type="text/css">
/*<![CDATA[*/

  li.sgc-1 {list-style: none; display: inline}
  /*]]>*/
  </style>
<meta content="urn:uuid:050545b7-33f4-4596-8f34-90dc3d880b72" name="Adept.expected.resource"/>
</head>
<body>
<p class="cn"><b>16</b></p>
<p class="ct"><b>Prot
```

The error in this case is a common one where the lang attribute is missing entirely, failing to specify the document's language with a tag like <html lang=\"en\">.