In [ ]:

```
import numpy as np
import pandas as pd

# For visualizations
import matplotlib.pyplot as plt
import nltk
import seaborn as sns
import gensim
import pyLDAvis
import textblob
import spacy

# For regular expressions
import re
# For handling string
import string
# For performing mathematical operations
import math

# Importing dataset
df=pd.read_csv('/Users/hgardner/Desktop/toxicity/data/preprocessed_data/cat_des
```

In [ ]:

```
df.shape
```

Out[ ]:

```
(721454, 3)
```

Text Stat

Dale Chall Score: https://en.wikipedia.org/wiki/Dale%E2%80%93Chall_readability_formula, 0-9.9 scale; the higher the score, the higher the level of the reader

Flesch Reading Score: https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests, 0-100 scale; the higher the score, the easier the text is to read

Gunning Fog Index: https://en.wikipedia.org/wiki/Gunning_fog_index

Instructions: https://www.analyticsvidhya.com/blog/2020/04/beginners-guide-exploratory-data-analysis-text-data/

In [ ]:

```
import textstat

df['dale_chall_score']=df['description'].apply(lambda x: textstat.dale_chall_re
df['flesch_reading_ease']=df['description'].apply(lambda x: textstat.flesch_rea
df['gunning_fog']=df['description'].apply(lambda x: textstat.gunning_fog(x))
```

In [ ]:

```
print('Dale Chall Score of Descriptions: Mean',df['dale_chall_score'].mean())
print('Dale Chall Score of Descriptions: Min',df['dale_chall_score'].min())
print('Dale Chall Score of Descriptions: Median',df['dale_chall_score'].median(
print('Dale Chall Score of Descriptions: Max',df['dale_chall_score'].max())
print('Dale Chall Score of Descriptions: Mode',df['dale_chall_score'].mode())

print('Flesch Reading Score of Descriptions: Mean',df['flesch_reading_ease'].me
print('Flesch Reading Score of Descriptions: Min',df['flesch_reading_ease'].mi
print('Flesch Reading Score of Descriptions: Median',df['flesch_reading_ease'].
print('Flesch Reading Score of Descriptions: Max',df['flesch_reading_ease'].ma
print('Flesch Reading Score of Descriptions: Mode',df['flesch_reading_ease'].m

print('Gunning Fog Index of Descriptions: Mean',df['gunning_fog'].mean())
print('Gunning Fog Index of Descriptions: Min ',df['gunning_fog'].min())
print('Gunning Fog Index of Descriptions: Median',df['gunning_fog'].median())
print('Gunning Fog Index of Descriptions: Max ',df['gunning_fog'].max())
print('Gunning Fog Index of Descriptions: Mode',df['gunning_fog'].mode())
```

```
Dale Chall Score of Descriptions: Mean 15.269600736776187
Dale Chall Score of Descriptions: Min 0.0
Dale Chall Score of Descriptions: Median 11.78
Dale Chall Score of Descriptions: Max 666.87
Dale Chall Score of Descriptions: Mode 0    35.27
Name: dale_chall_score, dtype: float64
Flesch Reading Score of Descriptions: Mean 35.63650015094354
Flesch Reading Score of Descriptions: Min -1783.3
Flesch Reading Score of Descriptions: Median 36.62
Flesch Reading Score of Descriptions: Max 206.84
Flesch Reading Score of Descriptions: Mode 0    36.62
Name: flesch_reading_ease, dtype: float64
Gunning Fog Index of Descriptions: Mean 14.071123079271572
Gunning Fog Index of Descriptions: Min  0.0
Gunning Fog Index of Descriptions: Median 15.26
Gunning Fog Index of Descriptions: Max  145.22
Gunning Fog Index of Descriptions: Mode 0    0.4
Name: gunning_fog, dtype: float64
```

In [ ]:

```
df['word_count']=df['description'].apply(lambda x: textstat.lexicon_count(x, re
```

In [ ]:

```
print('Word Count of Descriptions: Mean',df['word_count'].mean())
print('Word Count of Descriptions: Min ',df['word_count'].min())
print('Word Count of Descriptions: Median',df['word_count'].median())
print('Word Count of Descriptions: Max ',df['word_count'].max())
```

```
Word Count of Descriptions: Mean 85.8066945363114
Word Count of Descriptions: Min  0
Word Count of Descriptions: Median 66.0
Word Count of Descriptions: Max  1415
```

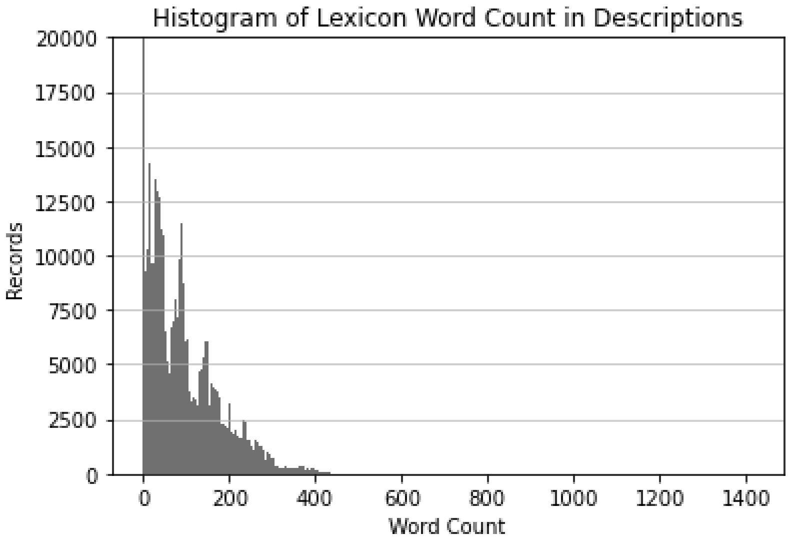
Visualizations of Lexicon Word Count

In [ ]:

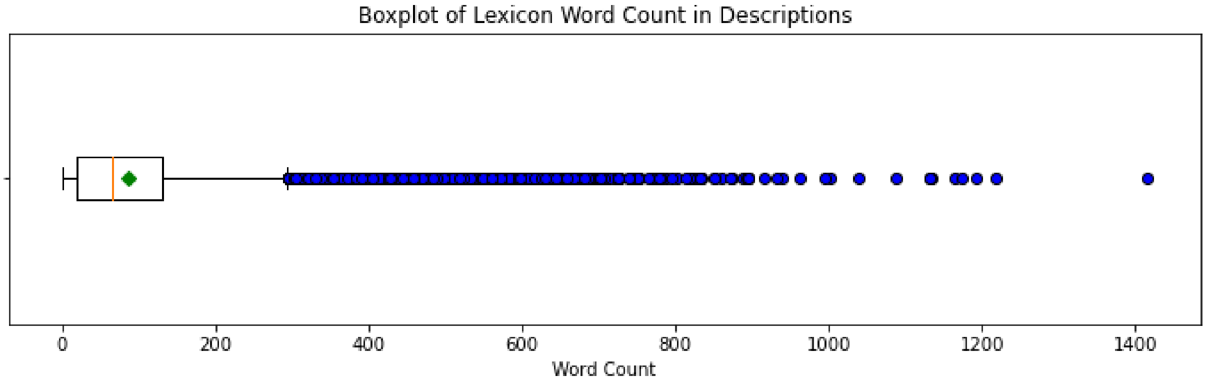
```
# An "interface" to matplotlib.axes.Axes.hist() method
n, bins, patches = plt.hist(x=df['word_count'], bins='auto', color='#333333',
                            alpha=0.7, rwidth=1)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Word Count')
plt.ylabel('Records')
plt.title('Histogram of Lexicon Word Count in Descriptions')
# Set a clean upper y-axis limit.
plt.ylim(ymax=20000)
```

Out[ ]:

```
(0.0, 20000.0)
```



In [ ]:
```
red_circle = dict(markerfacecolor='blue', marker='o', )
mean_shape = dict(markerfacecolor='green', marker='D', markeredgecolor='green')
plt.figure(figsize=(12,3))
plt.boxplot(x=df['word_count'], vert=False, flierprops=red_circle, showmeans=Tr
plt.xlabel('Word Count')
plt.title('Boxplot of Lexicon Word Count in Descriptions')
plt.show()
```



In [ ]:
```
df['reading_time']=df['description'].apply(lambda x: textstat.reading_time(x, m
print('Mean Reading Time: Mean',df['reading_time'].mean())
```

```
Mean Reading Time: Mean 7.179803854424767
```

Title EDA

In [ ]:
```
# Importing dataset
df_title=pd.read_csv('/Users/hgardner/Desktop/toxicity/data/preprocessed_data/c
```

In [ ]:
```
df_title['dale_chall_score']=df_title['title'].apply(lambda x: textstat.dale_ch
df_title['flesch_reading_ease']=df_title['title'].apply(lambda x: textstat.fles
df_title['word_count']=df_title['title'].apply(lambda x: textstat.lexicon_count
df_title['reading_time']=df_title['title'].apply(lambda x: textstat.reading_tim
```

In [ ]:
```
print('Dale Chall Score : Mean',df_title['dale_chall_score'].mean())
print('Dale Chall Score: Min',df_title['dale_chall_score'].min())
print('Dale Chall Score: Median',df_title['dale_chall_score'].median())
print('Dale Chall Score: Max',df_title['dale_chall_score'].max())

print('Flesch Reading Score: Mean',df_title['flesch_reading_ease'].mean())
print('Flesch Reading Score: Min',df_title['flesch_reading_ease'].min())
print('Flesch Reading Score: Median',df_title['flesch_reading_ease'].median())
print('Flesch Reading Score: Max',df_title['flesch_reading_ease'].max())

print('Word Count of Descriptions: Mean',df_title['word_count'].mean())
print('Word Count of Descriptions: Min ',df_title['word_count'].min())
print('Word Count of Descriptions: Median',df_title['word_count'].median())
print('Word Count of Descriptions: Max ',df_title['word_count'].max())

print('Mean Reading Time: Mean',df_title['reading_time'].mean())
```

```
Dale Chall Score : Mean 13.82108173483603
Dale Chall Score: Min 0.0
Dale Chall Score: Median 13.36
Dale Chall Score: Max 161.59
Flesch Reading Score: Mean 45.283108842110614
Flesch Reading Score: Min -1147.79
Flesch Reading Score: Median 50.5
Flesch Reading Score: Max 206.84
Word Count of Descriptions: Mean 8.581947359126282
Word Count of Descriptions: Min  0
Word Count of Descriptions: Median 6.0
Word Count of Descriptions: Max  383
Mean Reading Time: Mean 0.7278050127891331
```

In [ ]:

```
print(df_title[df_title.flesch_reading_ease < -1000])
```

```
                         bibid  \
31489       (MiAaPQ)EBC4860829
470602   (CKB)5590000000557983
470604   (CKB)5590000000557985
470607   (CKB)5590000000557988
2114819     (YBPDDA)ebc4860829
4517870     (OCoLC)ocm34477177

                                                     title  \
31489               Deterritorializing/Reterritorializing :
470602   HTML:Mason:Component:run('HTML:Mason:Component...
470604   HTML:Mason:Component:run('HTML:Mason:Component...
470607   HTML:Mason:Component:run('HTML:Mason:Component...
2114819             Deterritorializing/reterritorializing :
4517870  Modest-Witness@Second-Millennium.FemaleMan-Mee...

                                               clean_title  dale_chall_score
\
31489             ['deterritorializingreterritorializing ']             35.27
470602   ['htmlmasoncomponentrunhtmlmasoncomponentfileb...            114.22
470604   ['htmlmasoncomponentrunhtmlmasoncomponentfileb...            114.22
470607   ['htmlmasoncomponentrunhtmlmasoncomponentfileb...            114.22
2114819           ['deterritorializingreterritorializing ']             35.27
4517870  ['modest-witnesssecond-millenniumfemaleman-mee...             98.43

         flesch_reading_ease  word_count  reading_time
31489               -1063.19           1          0.56
470602              -1147.79           1          1.15
470604              -1147.79           1          1.15
470607              -1147.79           1          1.15
2114819             -1063.19           1          0.56
4517870             -1147.79           1          0.87
```

In [ ]:

```
flesch_123 = df_title[df_title.flesch_reading_ease > 122]
```

In [ ]:

```
flesch_123.head(50)
```

Out[ ]:

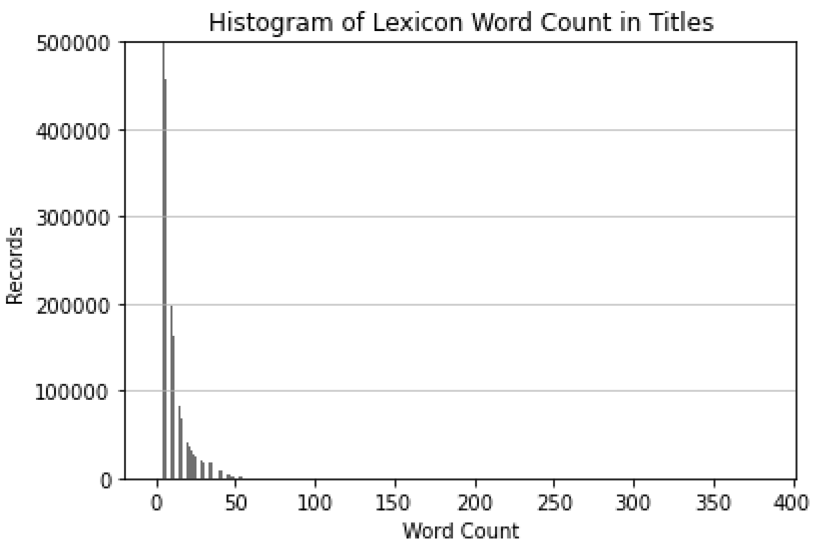
	bibid	title	clean_title	dale_chall_score	flesch_reading_ease	wo
246191	(CKB)5590000000551800	(((	['']	0.0	206.84	
3429094	(OCoLC)ocm79446449	[...] :	[' ']	0.0	206.84	
3806519	(OCoLC)ocn150473513	<>.	['']	0.0	206.84	
5122643	(OCoLC)ocm10054736	/	['']	0.0	206.84	
5251525	(OCoLC)ocm07641126	;	['']	0.0	206.84	

In [ ]:

```
#title histogram
# An "interface" to matplotlib.axes.Axes.hist() method
n, bins, patches = plt.hist(x=df_title['word_count'], bins=750, color='#333333'
                            alpha=0.7, rwidth=1)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Word Count')
plt.ylabel('Records')
plt.title('Histogram of Lexicon Word Count in Titles')
# Set a clean upper y-axis limit.
plt.ylim(ymax=500000)
```

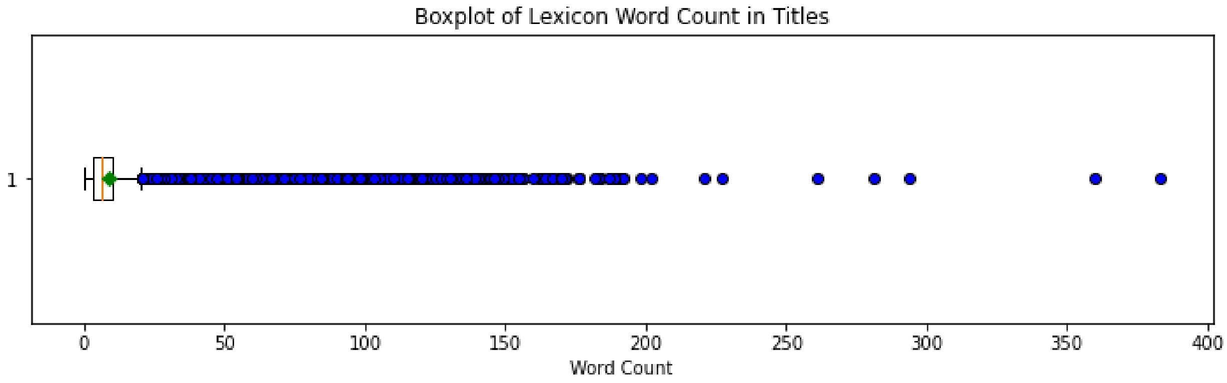
Out[ ]:

```
(0.0, 500000.0)
```



In [ ]:

```
#title box plot
red_circle = dict(markerfacecolor='blue', marker='o', )
mean_shape = dict(markerfacecolor='green', marker='D', markeredgecolor='green')
plt.figure(figsize=(12,3))
plt.boxplot(x=df_title['word_count'], vert=False, flierprops=red_circle, showme
plt.xlabel('Word Count')
plt.title('Boxplot of Lexicon Word Count in Titles')
plt.show()
```



Subject EDA

In [ ]:

```
df_sub = pd.read_csv('/Users/hgardner/Desktop/toxicity/data/parsed data files/r
```

In [ ]:

```
df_sub.shape
```

Out[ ]:

```
(3022612, 2)
```

In [ ]:

```
df_sub.head()
```

Out[ ]:

	bibid	subject
0	(OCoLC)557588801	Astronomy.
1	(OCoLC)1256541466	Black people in art
2	(OCoLC)48418774	Cotton manufacture
3	(OCoLC)39033407	Cooking, American
4	(OCoLC)32911699	Rock musicians

In [ ]:

```
df_sub.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3022612 entries, 0 to 3022611
Data columns (total 2 columns):
 #   Column   Dtype 
---  ------   ----- 
 0   bibid    object
 1   subject  object
dtypes: object(2)
memory usage: 46.1+ MB
```

In [ ]:

```
100*3022612/5481440
```

Out[ ]:

```
55.142663241775885
```

In [ ]:

```
df_sub.select_dtypes([object]).nunique()
```

Out[ ]:

```
bibid      2948473
subject     151392
dtype: int64
```

In [ ]:

```
freq_table = pd.crosstab(index=df_sub["subject"], columns="count")
```

In [ ]:

```
freq_table
```

Out[ ]:

col_0	count
subject	
\tPsychoanalysis and literature.	1
Diplomatic and consular service, Spanish.	1
Monuments	1
Nilpotent Lie groups.	1
!Cu-cut!	1
...	...
‘Tsv (The Hebrew word)	1
ʼBrug-pa (Sect)	1
קריירה	1
Ḥoshen mishpaṭ.	1
中国 -- 歴史 -- 明時代	1

151392 rows × 1 columns

In [ ]:

```
freq_table = freq_table.sort_values(by="count", ascending=False)
```

In [ ]:

```
#top 50 subject terms in the first term position
freq_table.head(50)
```

Out[ ]:

col_0	count
subject	
Law	17379
Indians of North America	15386
Women	12405
Geology	11621
Railroads	9856
Sermons, English	9580
Piano music.	8664
Education	7935
World War, 1939-1945	7863
Music	7824
English language	7802
African Americans	7796
Jews	7690
Taxation	7217
Operas	6661
Individual and Groups Rights	6586
Budget	6177
Wartime Conditions and Military Tactics	6001
Ballads, English	5901
English literature	5885
Missions	5616
Staging and Design	5521
Society of Friends	5513
Slavery	5487
American literature	5355
Symphonies.	5326
Archaeological surveying	5312
Presidents	5269
Science	5204
Operas.	5116
Agriculture	5003
Social and Cultural Life	4806
Christian life	4798
Organ music.	4684

col_0	count
subject	
Veterans	4512
Older people	4496
Art	4480
Christianity	4330
Small business	4322
Corrections	4287
World War, 1914-1918	4286
Medicine	4252
Church and state	4180
Groundwater	4025
Theology	3855
Motion pictures	3789
Church history	3724
Constitutional law	3592
Excavations (Archaeology)	3559
Banks and banking	3521

In [ ]:

```
freq_table.reset_index(inplace=True)
freq_table = freq_table.rename(columns = {'index':'new column name'})
```

In [ ]:

```
freq_table.info()
```

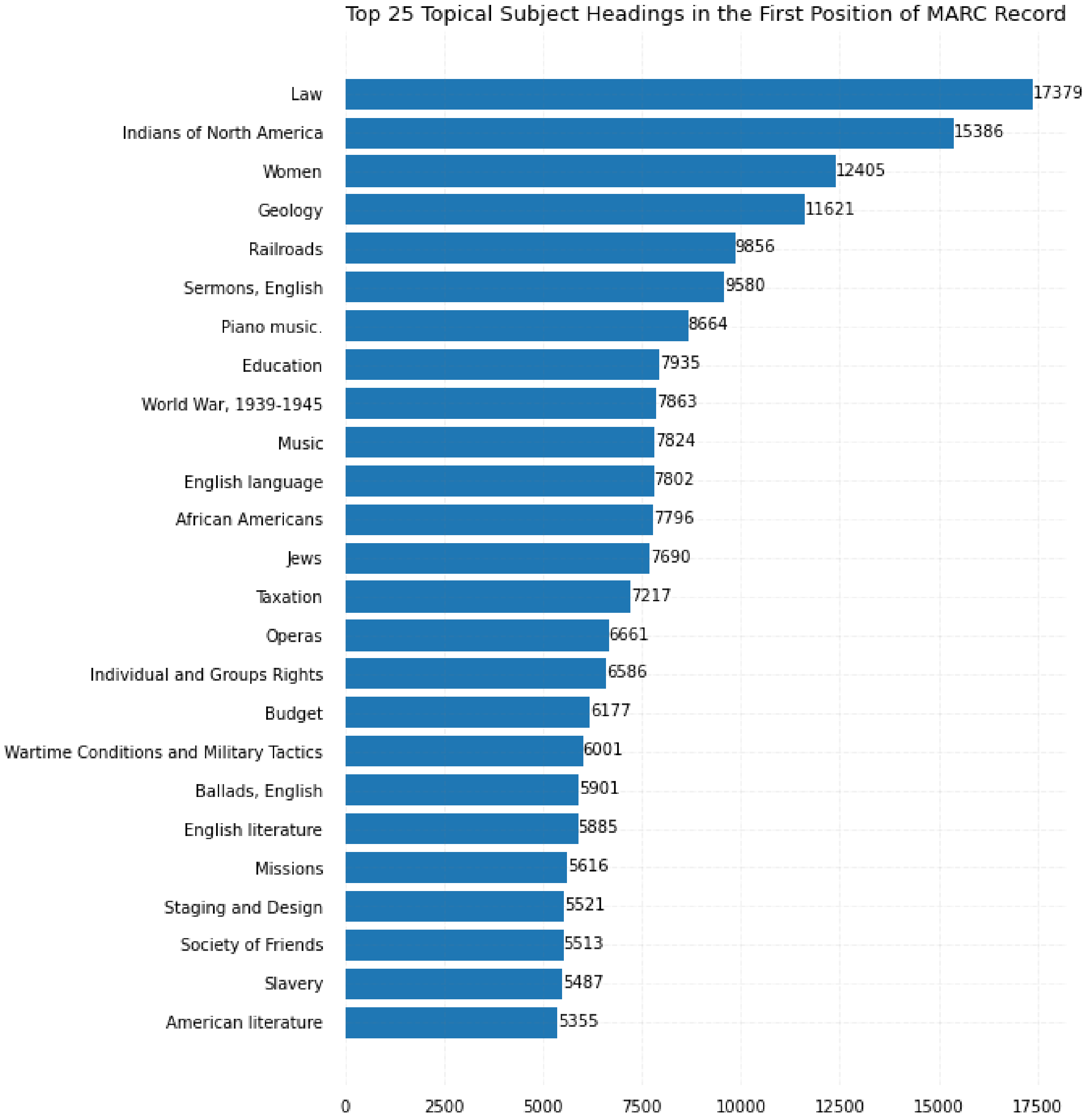
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 151392 entries, 0 to 151391
Data columns (total 2 columns):
 #   Column   Non-Null Count   Dtype 
---  ------   --------------   ----- 
 0   subject  151392 non-null  object
 1   count    151392 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 2.3+ MB
```

In [ ]:

```
subject = freq_table['subject'].head(25)
freq_ct = freq_table['count'].head(25)

# Figure Size
fig, ax = plt.subplots(figsize =(8, 12))

# Horizontal Bar Plot
ax.barh(subject, freq_ct)

# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    ax.spines[s].set_visible(False)
```

```
# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.2)

# Show top values
ax.invert_yaxis()

# Add annotation to bars
for i in ax.patches:
    plt.text(i.get_width()+0.2, i.get_y()+0.5,
             str(round((i.get_width()), 2)),
             fontsize = 10,
             color ='black')

# Add Plot Title
ax.set_title('Top 25 Topical Subject Headings in the First Position of MARC Rec
             loc ='left', fontsize = 13 )

# Show Plot
plt.show()
```

```
/var/folders/2g/1_zwfkdj3mvcdhj4kjs8zt3h0000gn/T/ipykernel_82048/4258922747.p
y:23: MatplotlibDeprecationWarning: The 'b' parameter of grid() has been renam
ed 'visible' since Matplotlib 3.5; support for the old name will be dropped tw
o minor releases later.
  ax.grid(b = True, color ='grey',
```



In [ ]:

```
print(freq_ct.sum())
(freq_ct.sum()/5481440)*100
```

```
203721
```

Out[ ]:

```
3.716559882074783
```

Publication Date

In [ ]:

```
df_pub = pd.read_csv('/Users/hgardner/Desktop/toxicity/data/parsed data files/r

#removing non-numeric characters
df_pub['pubdate'] = df_pub['pubdate'].str.extract('(\d+)', expand=False)

#removing month and day where present
df_pub['year'] = df_pub['pubdate'].str[:4]
```

```
<>:4: DeprecationWarning: invalid escape sequence \d
<>:4: DeprecationWarning: invalid escape sequence \d
/var/folders/2g/1_zwfkdj3mvcdhj4kjs8zt3h0000gn/T/ipykernel_82048/132339678.py:
4: DeprecationWarning: invalid escape sequence \d
  df_pub['pubdate'] = df_pub['pubdate'].str.extract('(\d+)', expand=False)
```

In [ ]:

```
df_pub['pubdate'].isnull().sum()
```

Out[ ]:

```
5601
```

In [ ]:

```
df_pub.dropna(inplace=True)
```

In [ ]:

```
df_pub.head()
```

Out[ ]:

	bibid	pubdate	year
0	(OCoLC)557588801	1828	1828
1	(OCoLC)13243571	1917	1917
2	(OCoLC)39033407	1998	1998
3	(OCoLC)32911699	1995	1995
4	(OCoLC)46476184	1881	1881

In [ ]:

```
df_pub['year'] = df_pub['year'].astype(int)
```

In [ ]:

```
#sanity check
df_pub.head(50)
```

Out[ ]:

	bibid	pubdate	year
0	(OCoLC)557588801	1828	1828
1	(OCoLC)13243571	1917	1917
2	(OCoLC)39033407	1998	1998
3	(OCoLC)32911699	1995	1995
4	(OCoLC)46476184	1881	1881
5	(OCoLC)14013404	1901	1901
6	(OCoLC)49300690	2002	2002
7	(OCoLC)48557504	2002	2002
8	(OCoLC)47192114	2001	2001
9	(OCoLC)1039917548	2012	2012
10	(OCoLC)1330435012	2010	2010
11	(OCoLC)841171518	2013	2013
12	(OCoLC)43475601	1986	1986
13	(OCoLC)44961579	1988	1988
14	(OCoLC)45843586	1999	1999
15	(OCoLC)64549389	2006	2006
16	(OCoLC)654658286	2006	2006
17	(OCoLC)759907747	2011	2011
18	(OCoLC)769344367	2011	2011
19	(DE-599)ZDB1473050 9	1933	1933
20	(CKB)2670000000271952	2012	2012
21	(YBPDDA)ebc1992442	2012	2012
22	(YBPDDA)ebc29289866	2022	2022
23	(YBPDDA)ebc6933471	2022	2022
24	(YBPDDA)ebs3287713	2022	2022
25	(YBPDDA)ebs3294959	2022	2022
26	(YBPDDA)ebs3296398	2022	2022
27	(YBPDDA)ebs3292448	2022	2022
28	(YBPDDA)ebs3292450	2022	2022
29	(YBPDDA)ebs3292453	2022	2022
30	(OCoLC)1183834397	2020	2020
31	(OCoLC)957655930	2016	2016
32	(OCoLC)1139151595	2019	2019
33	(OCoLC)956520869	2016	2016
34	(OCoLC)1152281636	2019	2019

	bibid	pubdate	year
35	(OCoLC)1140013425	2020	2020
36	(OCoLC)958455783	2016	2016
37	(OCoLC)1140423938	2020	2020
38	(OCoLC)958455585	2016	2016
39	(OCoLC)859687676	2013	2013
40	(OCoLC)821725631	2012	2012
41	(OCoLC)809317651	2012	2012
42	(OCoLC)863824777	2013	2013
43	(OCoLC)1225550756	2020	2020
44	(OCoLC)1236261256	2021	2021
45	(OCoLC)805418933	2012	2012
46	(OCoLC)1225551609	2020	2020
47	(OCoLC)1237863945	2021	2021
48	(OCoLC)1239991805	2021	2021
49	(OCoLC)872695599	2013	2013

In [ ]:

```
year_table = pd.crosstab(index=df_pub["year"], columns="count")
year_table.reset_index(inplace=True)
```

In [ ]:

```
year_table = year_table.sort_values(by="count", ascending=False)
```

In [ ]:

```
year_table.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1094 entries, 676 to 1093
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   year    1094 non-null   object
 1   count   1094 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 25.6+ KB
```

In [ ]:

```
year_table.head()
```

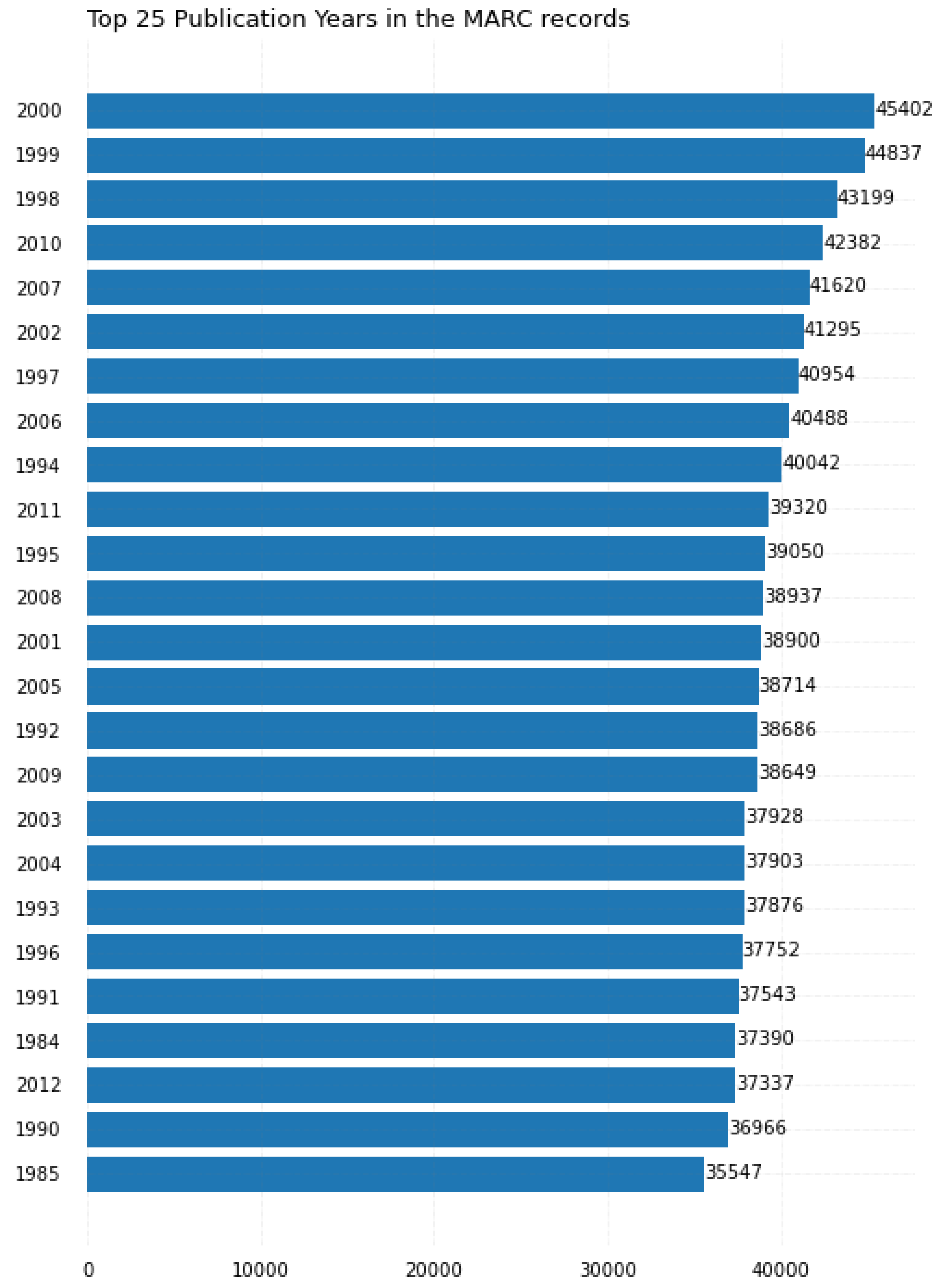
Out[ ]:

col_0	year	count
676	2000	45402
672	1999	44837
671	1998	43199
687	2010	42382
683	2007	41620

In [ ]:

```
year = year_table['year'].head(25)
yr_freq_ct = year_table['count'].head(25)

# Figure Size
fig, ax = plt.subplots(figsize =(8, 12))

# Horizontal Bar Plot
ax.barh(year, yr_freq_ct)

# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    ax.spines[s].set_visible(False)

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.2)

# Show top values
ax.invert_yaxis()

# Add annotation to bars
for i in ax.patches:
    plt.text(i.get_width()+0.2, i.get_y()+0.5,
             str(round((i.get_width()), 2)),
             fontsize = 10,
             color ='black')

# Add Plot Title
ax.set_title('Top 25 Publication Years in the MARC records',
             loc ='left', fontsize = 13 )

# Show Plot
plt.show()
```

```
/var/folders/2g/1_zwfkdj3mvcdhj4kjs8zt3h0000gn/T/ipykernel_82048/4057432388.p
y:23: MatplotlibDeprecationWarning: The 'b' parameter of grid() has been renam
ed 'visible' since Matplotlib 3.5; support for the old name will be dropped tw
o minor releases later.
  ax.grid(b = True, color ='grey',
```



In [ ]:

```
year_table['year'] = year_table['year'].astype(int)
```

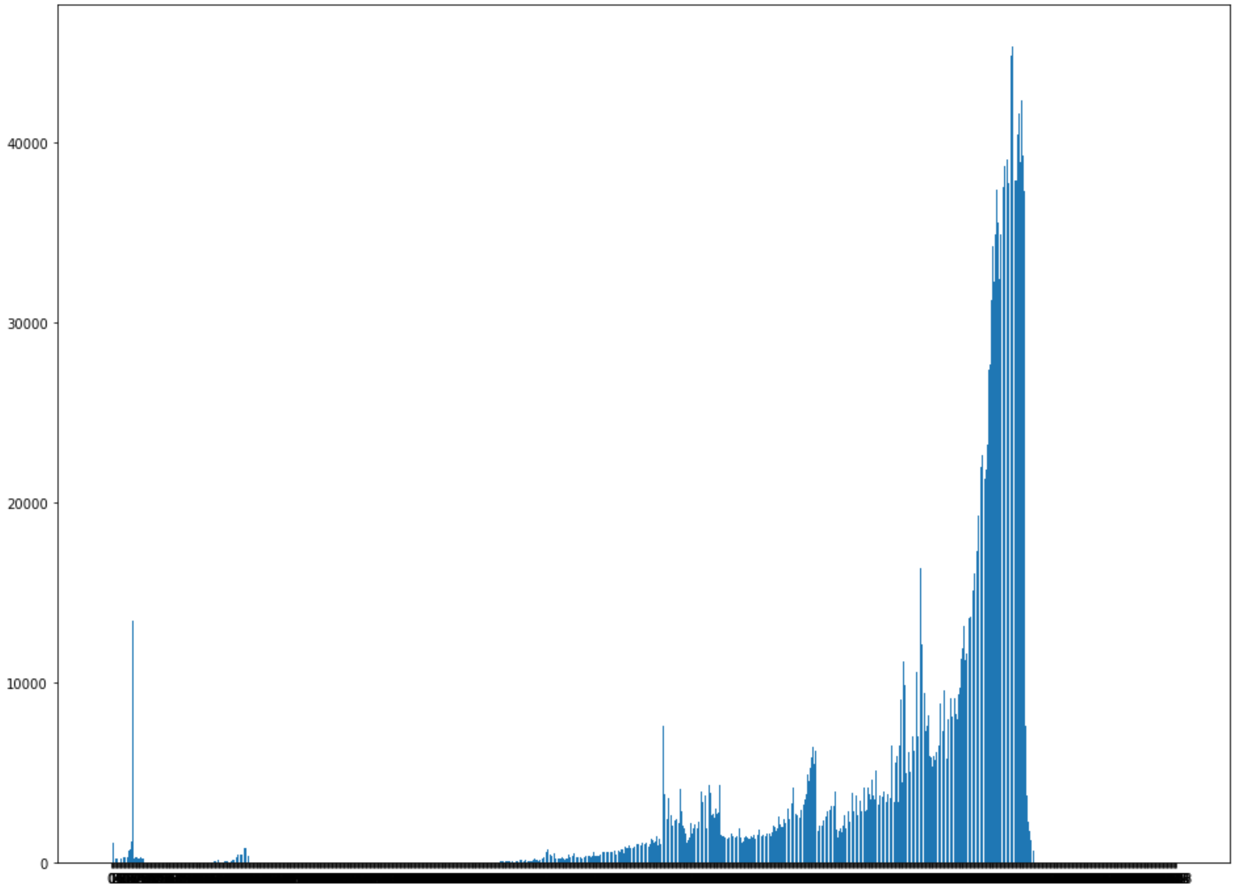
In [ ]:

```
year_table = year_table.sort_values(by="year", ascending=True)
```

In [ ]:

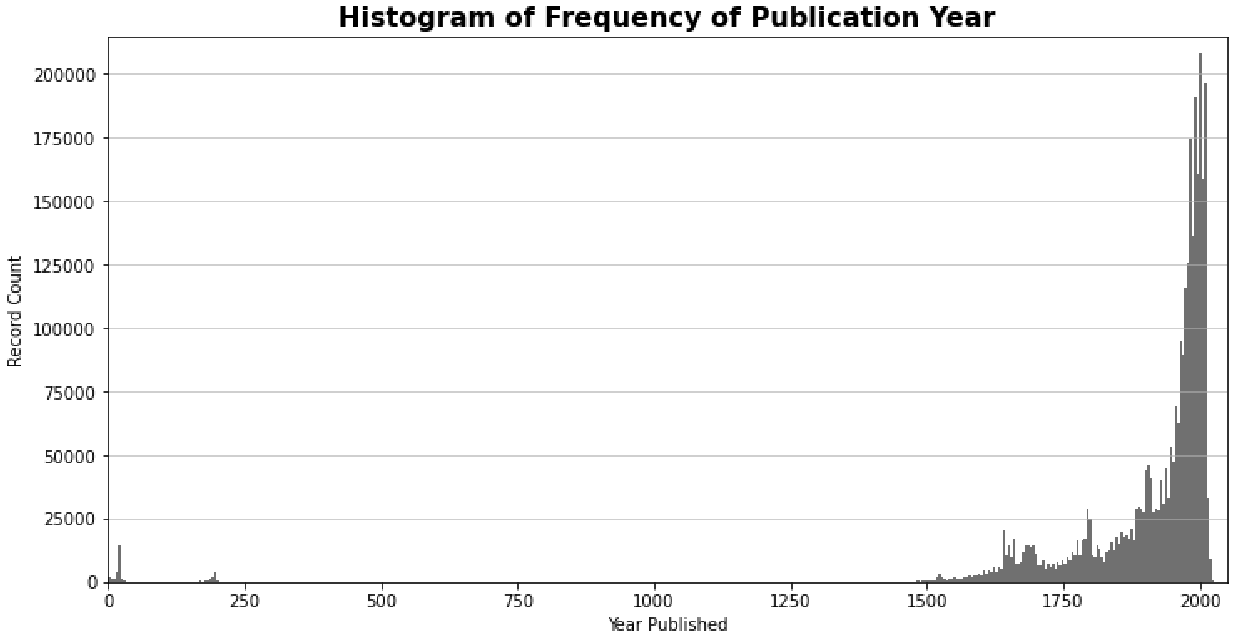
```
# Make a random dataset:
height = year_table['count']
bars = year_table['year']
y_pos = np.arange(len(bars))
# Figure Size
fig, ax = plt.subplots(figsize =(16, 12))

# Create bars
plt.bar(y_pos, height)

# Create names on the x-axis
```

```
plt.xticks(y_pos, bars)

# Show graphic
plt.show()
```



In [ ]:
```
df_pub['year'] = df_pub['year'].astype(int)
df_pub = df_pub.sort_values(by='year', ascending=True)
```

In [ ]:
```
# An "interface" to matplotlib.axes.Axes.hist() method
# Figure Size
fig, ax = plt.subplots(figsize =(12, 6))
n, bins, patches = plt.hist(x=df_pub['year'], bins=2000, color='#333333',
                            alpha=0.7, rwidth=1)

plt.grid(axis='y', alpha=0.75)
plt.xlabel('Year Published')
plt.ylabel('Record Count')
# Add Plot Title
ax.set_title('Histogram of Frequency of Publication Year',
             loc ='center', fontsize = 16, fontweight='bold')
# Set a clean upper y-axis limit.
plt.ylim(ymax=215000)
plt.xlim(0,2050)

every_nth = 50
for n, label in enumerate(ax.xaxis.get_ticklabels()):
    if n % every_nth != 0:
        label.set_visible(True)
```


Histogram of Frequency of Publication Year
Record Count
Year Published
0
25000
50000
75000
100000
125000
150000
175000
200000
0
250
500
750
1000
1250
1500
1750
2000