Problem 1

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
sns.set()
```

In [2]:
```
with open('acm (1).txt', 'r', newline = '', encoding = 'utf-8') as file:
    data = file.readlines()
```

In [3]:
```
len(data)
```

Out[3]: 25494853

In [4]:
```
data.append('\n')
data = [a.replace('  ', ' ') for a in data]
data = [a.replace(', Jr.', 'Jr.') for a in data]
data = [a.replace(', I', 'I') for a in data]
data = [a.replace(', II', 'II') for a in data]
data = [a.replace(', -', ' ') for a in data]
data = [a.replace(',  -', '') for a in data]

data = [a.replace('  ', ' ') for a in data]
```

In [5]:
```
df = []
i = 0
j = 0

while i < len(data):
  if(data[i] == '\n'):
   df.append(data[j:i])
   j = i + 1
  i = i + 1

textdata = {'Title':[], 'Author': [], 'Year': [], 'Publication Venue': [], 'Index':[], 'References': [], 'Abstract': []}

for x in df:
  title = []
  author = []
  year = []
  publication_venue = []
  index_id = []
  references = []
  abstract = []

  for y in x:
    if(y[:2] == '#*'):
      title.append(y[2:].split('\n')[0])
    if (y[:2] == '#@'):
      author.append(y[2:].split('\n')[0])
    if (y[:2] == '#t'):
      year.append(y[2:].split('\n')[0])
    if (y[:2] == '#c'):
      publication_venue.append(y[2:].split('\n')[0])
    if (y[:6] == '#index'):
      index_id.append(y[6:].split('\n')[0])
    if (y[:2] == '#%'):
      references.append(y[2:].split('\n')[0])
    if (y[:2] == '#!'):
      abstract.append(y[2:].split('\n')[0])

  textdata["Title"].append('; '.join(title))
  textdata["Author"].append('; '.join(author))
  textdata["Year"].append('; '.join(year))
  textdata["Publication Venue"].append('; '.join(publication_venue))
  textdata["Index"].append('; '.join(index_id))
  textdata["References"].append('; '.join(references))
  textdata["Abstract"].append('; '.join(abstract))

textdata = pd.DataFrame(textdata)
```

In [6]:
```
textdata.head()
```

Out[6]:

	Title	Author	Year	Publication Venue	Index	References	Abstract
0	MOSFET table look-up models for circuit simula...		1984	Integration, the VLSI Journal	1		
1	The verification of the protection mechanisms ...	Virgil D. Gligor	1984	International Journal of Parallel Programming	2		
2	Another view of functional and multivalued dep...	M. Gyssens, J. Paredaens	1984	International Journal of Parallel Programming	3		
3	Entity-relationship diagrams which are in BCNF	Sushil Jajodia, Peter A. Ng, Frederick N. Spri...	1984	International Journal of Parallel Programming	4		
4	The computer comes of age	Rene Moreau	1984	The computer comes of age	5		

In [7]:
```
textdata.tail()
```

Out[7]:

	Title	Author	Year	Publication Venue	Index	References	Abstract
2385062	Linear-time computation of prefix table for we...	-	2016	Theoretical Computer Science	2385063	2381731	The prefix table of a string is one of the mos...
2385063	A space-efficient alphabet-independent Four-Ru...	-	2016	Theoretical Computer Science	2385064	2381731	Given two strings X (| X | = m) and Y (| Y ...
2385064	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385065		
2385065	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385066		
2385066							

Part A

In [8]:
```
len(textdata['Author'].explode().unique())
```

Out[8]: 1670103

In [9]:
```
len(textdata['Publication Venue'].unique())
```

Out[9]: 273330

In [10]:
```
len(textdata['Title'].unique())
```

Out[10]: 2183552

In [11]:
```
len(textdata['References'].unique())
```

Out[11]: 884933

Part B

In [12]:
```
textdata[textdata['Publication Venue'].str.contains('Principles and Practice of Knowledge Discovery in Databases')]
```

Out[12]:

	Title	Author	Year	Publication Venue	Index	References	Abstract
799595	Summarization of dynamic content in web collec...	Adam Jatowt, Mitsuru Ishizuka	2004	PKDD '04 Proceedings of the 8th European Confe...	799596	168250; 207271; 217577; 272248; 287615; 357907...	This paper describes a new research proposal o...
799732	Proceedings of the 8th European Conference on ...	Jean-François Boulicaut, Floriana Esposito, Fo...	2004	PKDD '04 Proceedings of the 8th European Confe...	799733		
799733	Random matrices in data analysis	Dimitris Achlioptas	2004	PKDD '04 Proceedings of the 8th European Confe...	799734		We show how carefully crafted random matrices ...
799734	Data privacy	Rakesh Agrawal	2004	PKDD '04 Proceedings of the 8th European Confe...	799735		There is increasing need to build information ...
799735	Breaking through the syntax barrier: searching...	Soumen Chakrabarti	2004	PKDD '04 Proceedings of the 8th European Confe...	799736		The next wave in search technology will be dri...
...	...	...	...	...	...	...	...
1673617	Speeding up logistic model tree induction	Marc Sumner, Eibe Frank, Mark Hall	2005	PKDD'05 Proceedings of the 9th European confer...	1673618	136349; 290481; 810934; 2135000	Logistic Model Trees have been shown to be ver...
1673618	A random method for quantifying changing distr...	Haixun Wang, Jian Pei	2005	PKDD'05 Proceedings of the 9th European confer...	1673619	115607; 342599; 400846; 424996; 443615; 481459...	In applications such as fraud and intrusion de...
1673619	Deriving class association rules based on leve...	Takashi Washio, Koutarou Nakanishi, Hiroshi Mo...	2005	PKDD'05 Proceedings of the 9th European confer...	1673620	210159; 248791; 397383; 466482; 481289; 546046...	Most approaches of Class Association Rule (CAR...
1673620	An incremental algorithm for mining generators...	Lijun Xu, Kanglin Xie	2005	PKDD'05 Proceedings of the 9th European confer...	1673621	280466; 464203; 466663; 481289; 511332; 546697...	This paper presents an efficient algorithm for...
1673621	Hybrid technique for artificial neural network...	Cleber Zanchettin, Teresa Bernarda Ludermir	2005	PKDD'05 Proceedings of the 9th European confer...	1673622	11719; 36407; 369235; 386198; 388153; 465881; ...	This work presents a technique that integrates...

212 rows × 7 columns

In [13]:
```
len(textdata[textdata['Publication Venue'].str.contains('Principles and Practice of Knowledge Discovery in Databases')])
```

Out[13]: 212

Observation:
The Count numbers don't seem accurate. For the same conference, we can see that the venues have different names. While a different publication don't have the same names. This will increase our count of venues than the true count as it is not consistent.

Part C

In [14]:
```
textdata['Author'] = textdata['Author'].str.split(', ')
```

In [15]:
```
textdata = textdata.explode('Author')
```

In [16]:
```
pubs_per_author = textdata.groupby(['Author'],as_index=False)['Title'].count()
```

In [17]:
```
pubs_per_author = pubs_per_author[pubs_per_author["Author"]!= ""]
```

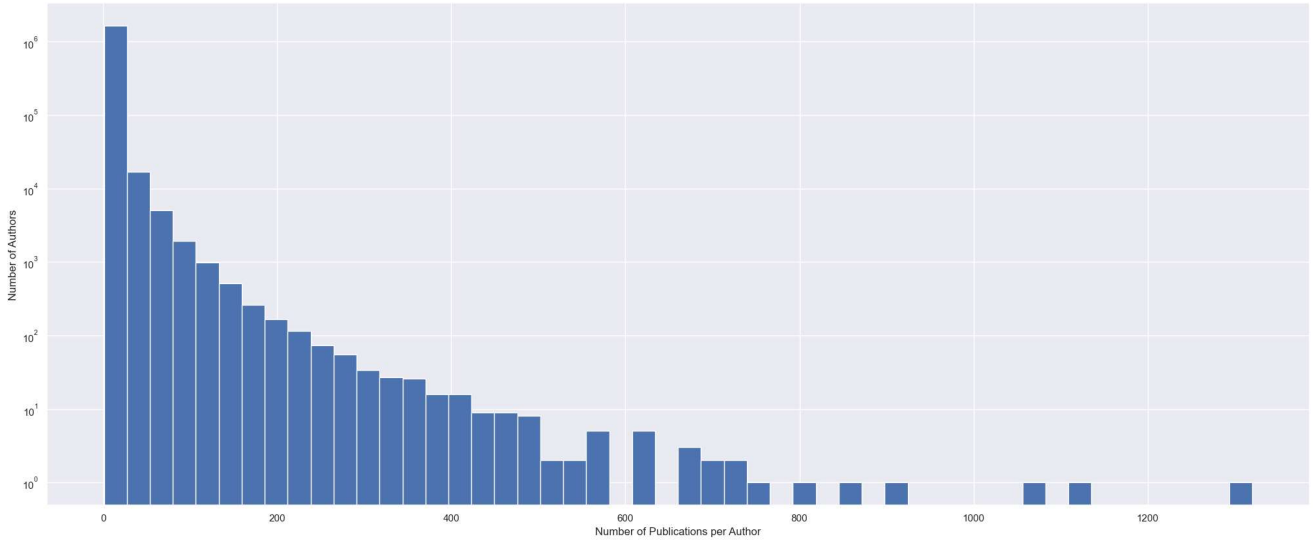
In [18]:
```
pubs_per_author.sort_values(["Title"],ascending = False)
```

Out[18]:

	Author	Title
1539675	Wei Wang	1320
275763	Computer Staff	1111
878237	Linux Journal Staff	1082
860548	Lei Zhang	906
1539831	Wei Zhang	859
...	...	...
692385	Jia-Shiuan Tsai	1
692383	Jia-Shing Sheu	1
692382	Jia-Shing Ma	1
692381	Jia-Shing Chen	1
1668882	脓bero Camilo Kreps Ben挪tez	1

1668882 rows × 2 columns

In [19]:
```
pubs_per_author['Title'].plot(kind = "hist", logy = True, bins = 50, figsize = (25,10))

plt.xlabel("Number of Publications per Author")
plt.ylabel("Number of Authors")
plt.show()
```



Part D

In [20]:
```
mean = np.mean(pubs_per_author['Title'])
std_dev = np.std(pubs_per_author['Title'])

q1 = np.percentile(pubs_per_author['Title'], 25)
q2 = np.percentile(pubs_per_author['Title'], 50)
q3 = np.percentile(pubs_per_author['Title'], 75)

print("Mean:", mean)
print("Standard Deviation:", std_dev)
print("Q1 (First Quartile):", q1)
print("Q2 (Second quartile or median):", q2)
print("Q3 (Third quartile):", q3)
```

```
Mean: 3.388953203402038
Standard Deviation: 9.826593915411662
Q1 (First Quartile): 1.0
Q2 (Second quartile or median): 1.0
Q3 (Third quartile): 2.0
```

Majority of authors have one publications as both the first quartile & median values are 1. The distribution is also skewed to the right.

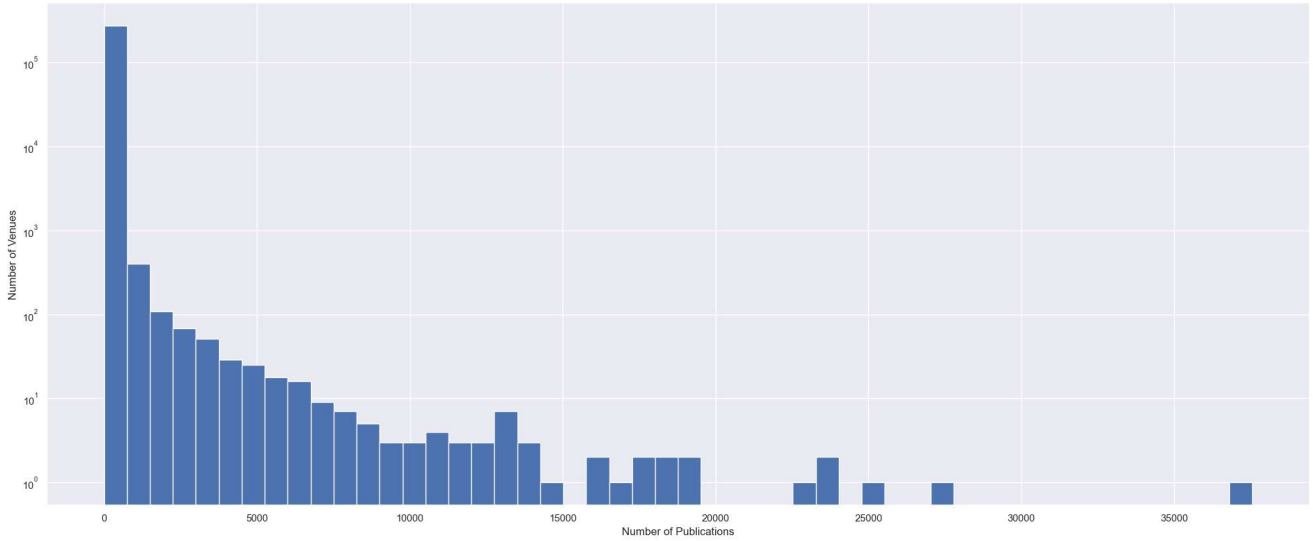
Part E

In [21]:
```
pubs_per_venue = textdata.groupby('Publication Venue')['Title'].count()
```

In [22]:
```
pubs_per_venue.sort_values(ascending = False)
```

Out[22]:
```
Publication Venue
Microelectronic Engineering                                                                    37546
Bioinformatics                                                                                 27385
IEEE Transactions on Information Theory                                                        25174
Expert Systems with Applications: An International Journal                                     23465
IEEE Transactions on Signal Processing                                                         23419
                                                                                               ...
A hop by hop architecture for multicast transport in ad hoc wireless networks                      1
A history-based semantics for algebraic methods in object-oriented software engineering           1
A history of modern computing                                                                      1
A history of general purpose computer uses in the united states 1954 to 1977 and likely future trends.    1
“Virtual fixtures”: perceptual overlays enhance operator performance in telepresence tasks         1
Name: Title, Length: 273330, dtype: int64
```

In [23]:
```
pubs_per_venue.plot(kind="hist", logy = True, bins = 50, figsize = (25,10))

plt.xlabel("Number of Publications")
plt.ylabel("Number of Venues")
plt.show()
```



In [24]:
```
most_pubs_venue = pubs_per_venue.idxmax()

print("Venue with the largest number of publications:", most_pubs_venue)
```

```
Venue with the largest number of publications: Microelectronic Engineering
```

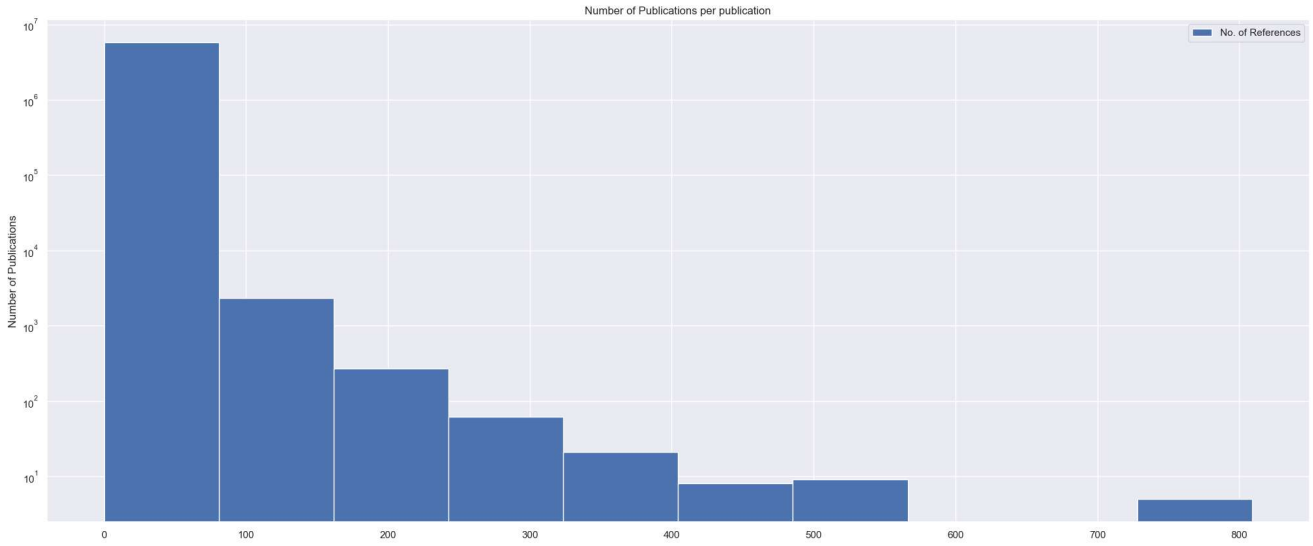
Part F

In [25]:
```
textdata_1 = textdata.copy()
textdata_1['References'] = textdata_1['References'].apply(lambda x: x.split('; '))
textdata_1['No. of References'] = textdata_1['References'].apply(lambda x: 0 if x==[''] else len(x))
textdata_1.tail()
```

Out[25]:

	Title	Author	Year	Publication Venue	Index	References	Abstract	No. of References
2385062	Linear-time computation of prefix table for we...	-	2016	Theoretical Computer Science	2385063	[2381731]	The prefix table of a string is one of the mos...	1
2385063	A space-efficient alphabet-independent Four-Ru...	-	2016	Theoretical Computer Science	2385064	[2381731]	Given two strings X (| X | = m) and Y (| Y ...	1
2385064	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385065	[]		0
2385065	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385066	[]		0
2385066						[]		0

In [26]:
```
textdata_1.plot(kind="hist", logy = True, figsize = (25,10), title = 'Number of Publications per publication')

plt.ylabel("Number of Publications")
plt.show()
```



In [27]:
```
reference_high = textdata_1.sort_values('No. of References', ascending = False)['Title'].values[0]
reference_high
```

Out[27]: 'Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles'

In [28]:
```
textdata_1.tail()
```

Out[28]:

	Title	Author	Year	Publication Venue	Index	References	Abstract	No. of References
2385062	Linear-time computation of prefix table for we...	-	2016	Theoretical Computer Science	2385063	[2381731]	The prefix table of a string is one of the mos...	1
2385063	A space-efficient alphabet-independent Four-Ru...	-	2016	Theoretical Computer Science	2385064	[2381731]	Given two strings X (| X | = m) and Y (| Y ...	1
2385064	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385065	[]		0
2385065	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385066	[]		0
2385066						[]		0

In [29]:
```
textdata_1 = textdata_1.explode('References').groupby('References',as_index=False).agg(
                                                    No_of_Citations = ('Index',np.count_nonzero))
textdata_1.head()
```

Out[29]:

	References	No_of_Citations
0		2875506
1	10	2
2	1000	6
3	10000	1
4	100000	6

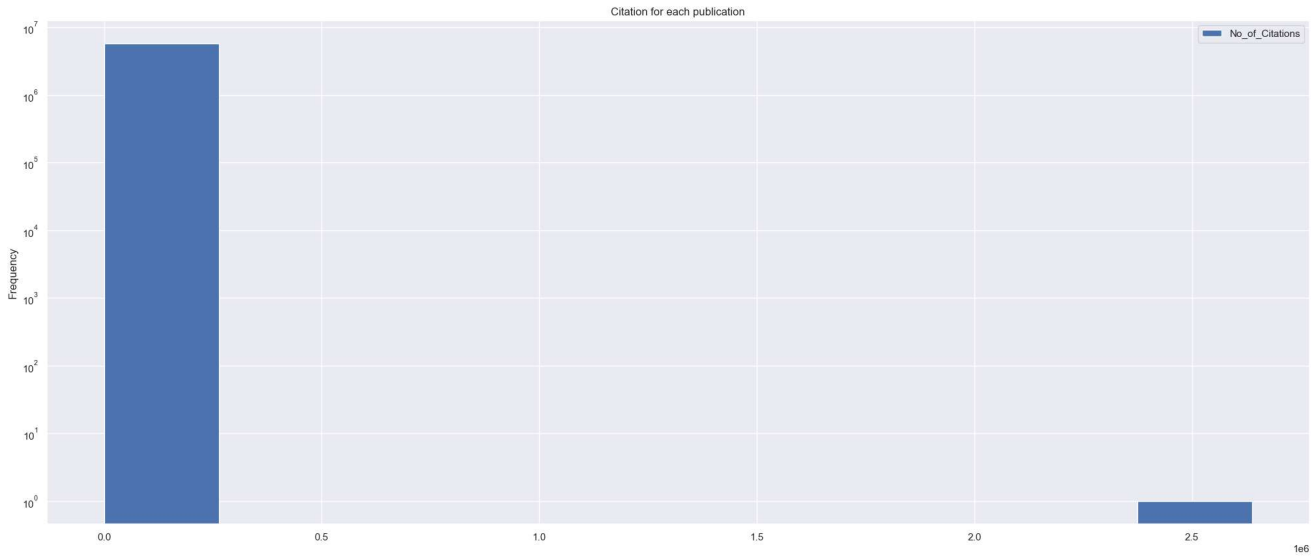
In [30]:
```
textdata = textdata.merge(textdata_1[1:],how='left',left_on='Index',right_on='References').fillna(0)
textdata.head()
```

Out[30]:

	Title	Author	Year	Publication Venue	Index	References_x	Abstract	References_y	No_of_Citations
0	MOSFET table look-up models for circuit simula...		1984	Integration, the VLSI Journal	1			0	0.0
1	The verification of the protection mechanisms ...	Virgil D. Gligor	1984	International Journal of Parallel Programming	2			0	0.0
2	Another view of functional and multivalued dep...	M. Gyssens	1984	International Journal of Parallel Programming	3			3	6.0
3	Another view of functional and multivalued dep...	J. Paredaens	1984	International Journal of Parallel Programming	3			3	6.0
4	Entity-relationship diagrams which are in BCNF	Sushil Jajodia	1984	International Journal of Parallel Programming	4			4	8.0

In [31]:
```
textdata[1:].plot(y = 'No_of_Citations', kind = 'hist', logy = True, figsize = (25,10))
plt.title('Citation for each publication')
```

Out[31]: Text(0.5, 1.0, 'Citation for each publication')



In [32]:
```
textdata[textdata['Index'] == '2135000']['Title'].values[0]
```

Out[32]: 'INFORMS Journal on Computing'

In [33]:
```
textdata_1[1:].sort_values('No_of_Citations', ascending = False)['No_of_Citations'].values[0]
```

Out[33]: 2638303

In [34]:
```
textdata
```

Out[34]:

	Title	Author	Year	Publication Venue	Index	References_x	Abstract	References_y	No_of_Citations
0	MOSFET table look-up models for circuit simula...		1984	Integration, the VLSI Journal	1			0	0.0
1	The verification of the protection mechanisms ...	Virgil D. Gligor	1984	International Journal of Parallel Programming	2			0	0.0
2	Another view of functional and multivalued dep...	M. Gyssens	1984	International Journal of Parallel Programming	3			3	6.0
3	Another view of functional and multivalued dep...	J. Paredaens	1984	International Journal of Parallel Programming	3			3	6.0
4	Entity-relationship diagrams which are in BCNF	Sushil Jajodia	1984	International Journal of Parallel Programming	4			4	8.0
...	...	...	...	...	...	...	...	...	...
5806864	Linear-time computation of prefix table for we...	-	2016	Theoretical Computer Science	2385063	2381731	The prefix table of a string is one of the mos...	0	0.0
5806865	A space-efficient alphabet-independent Four-Ru...	-	2016	Theoretical Computer Science	2385064	2381731	Given two strings X (| X | = m) and Y (| Y ...	0	0.0
5806866	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385065			0	0.0
5806867	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385066			0	0.0
5806868								0	0.0

5806869 rows × 9 columns

In [35]:
```
textdata.drop(labels=['References_y'], axis=1, inplace=True)
```

In [36]:
```
textdata.rename(columns={'References_x' :'References'},inplace=True)
```

In [37]:
```
textdata
```

Out[37]:

	Title	Author	Year	Publication Venue	Index	References	Abstract	No_of_Citations
0	MOSFET table look-up models for circuit simula...		1984	Integration, the VLSI Journal	1			0.0
1	The verification of the protection mechanisms ...	Virgil D. Gligor	1984	International Journal of Parallel Programming	2			0.0
2	Another view of functional and multivalued dep...	M. Gyssens	1984	International Journal of Parallel Programming	3			6.0
3	Another view of functional and multivalued dep...	J. Paredaens	1984	International Journal of Parallel Programming	3			6.0
4	Entity-relationship diagrams which are in BCNF	Sushil Jajodia	1984	International Journal of Parallel Programming	4			8.0
...	...	...	...	...	...	...	...	...
5806864	Linear-time computation of prefix table for we...	-	2016	Theoretical Computer Science	2385063	2381731	The prefix table of a string is one of the mos...	0.0
5806865	A space-efficient alphabet-independent Four-Ru...	-	2016	Theoretical Computer Science	2385064	2381731	Given two strings X (| X | = m) and Y (| Y ...	0.0
5806866	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385065			0.0
5806867	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385066			0.0
5806868								0.0

5806869 rows × 8 columns

Observation:
The journal 'INFORMS Journal on Computing' has the highest citation number, so I believe that this number is ture.

Part G

In [38]:
```
textdata_2 = textdata.groupby('Publication Venue').agg(Number_of_publications = ('Index',np.count_nonzero),
                                                       Total_Citations = ('No_of_Citations',np.sum))
```

In [39]:
```
textdata_2.head()
```

Out[39]:

Publication Venue	Number_of_publications	Total_Citations
	435	7224.0
!%@ (4th ed.): a directory of electronic mail addressing & networks	2	0.0
!%@:: a directory of electronic mail addressing & networks	2	0.0
!%@:: a directory of electronic mail addressing and networks: second edition	2	2.0
"... but will RISC run LISP??" (a feasibility study)	1	1.0

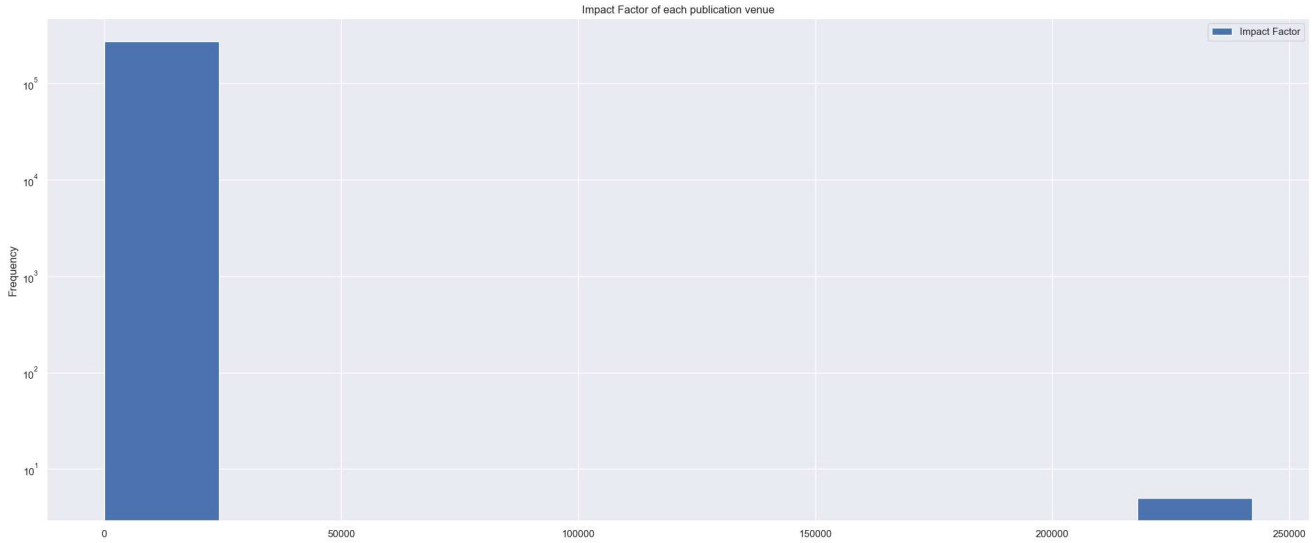
In [40]:
```
textdata_2['Impact Factor'] = textdata_2['Total_Citations'].div(textdata_2['Number_of_publications'])
textdata_2.tail()
```

Out[40]:

Publication Venue	Number_of_publications	Total_Citations	Impact Factor
“High-tech” materials: challenges and opportunities for chemical engineers	1	0.0	0.0
“Meeting the free dreamer”	1	0.0	0.0
“Post-game analysis”: a heuristic resource management framework for concurrent systems	1	10.0	10.0
“Ten trajectories of dawn and a babble of musics”, for computer-generated sounds and video	1	0.0	0.0
“Virtual fixtures”: perceptual overlays enhance operator performance in telepresence tasks	1	8.0	8.0

In [41]:
```
textdata_2.plot(y = 'Impact Factor', kind = 'hist', logy = True, figsize = (25,10))
plt.title('Impact Factor of each publication venue')
```

Out[41]: Text(0.5, 1.0, 'Impact Factor of each publication venue')



Part H

In [42]:
```
textdata_2.sort_values('Impact Factor', ascending = False).head(5)
```

Out[42]:

Publication Venue	Number_of_publications	Total_Citations	Impact Factor
IJIRR: International Journal of Information Retrieval Research.	1	242132.0	242132.0
PVLDB	5	1210660.0	242132.0
AI EDAM	3	726396.0	242132.0
Graphics Interface 1990	2	484264.0	242132.0
Graz	3	726396.0	242132.0

Observation:
We observe that the number of journals which is not equivalent to the highest impact factor. I don't believe this number as these journals only have a single publication.

Part I

In [43]:
```
textdata['Number_of_publications'] = textdata.groupby('Publication Venue')['Index'].transform('count')
textdata.tail()
```

Out[43]:

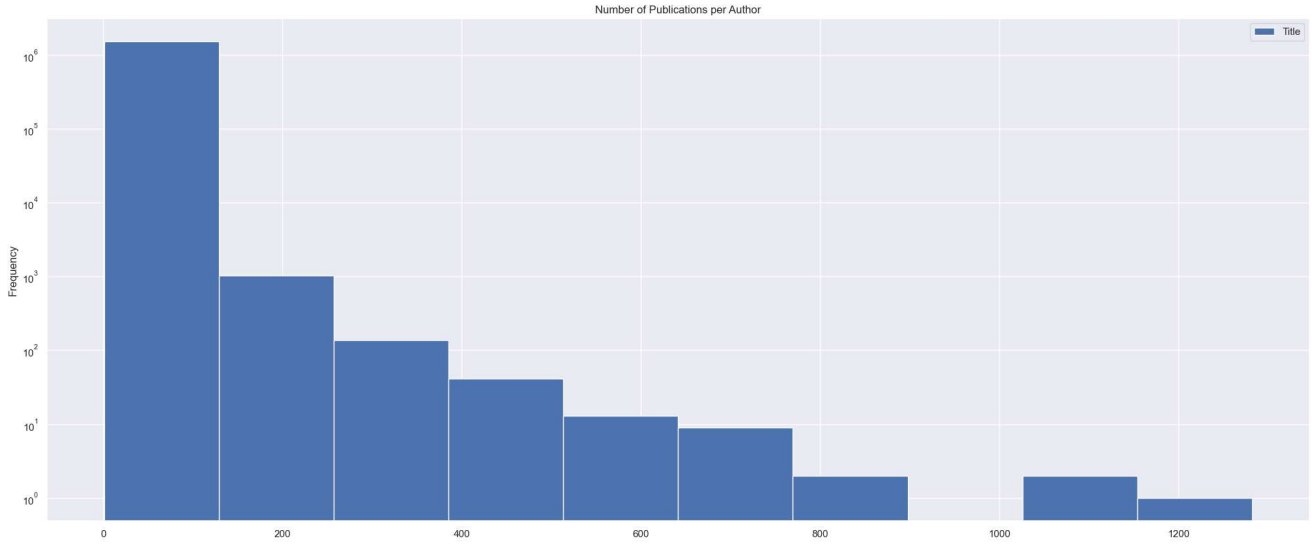
	Title	Author	Year	Publication Venue	Index	References	Abstract	No_of_Citations	Number_of_publications
5806864	Linear-time computation of prefix table for we...	-	2016	Theoretical Computer Science	2385063	2381731	The prefix table of a string is one of the mos...	0.0	17259
5806865	A space-efficient alphabet-independent Four-Ru...	-	2016	Theoretical Computer Science	2385064	2381731	Given two strings X (| X | = m) and Y (| Y ...	0.0	17259
5806866	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385065			0.0	2
5806867	Computers in Entertainment (CIE) - Special Iss...		2016	Computers in Entertainment (CIE)	2385066			0.0	2
5806868								0.0	436

In [44]:
```
textdata = textdata[textdata['Number_of_publications'] >= 10]
textdata['Author'] = textdata['Author'].apply(lambda x: x.split('; '))
```

```
C:\Users\ayush\AppData\Local\Temp\ipykernel_8996\403841256.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  textdata['Author'] = textdata['Author'].apply(lambda x: x.split('; '))
```

In [45]:
```
textdata.explode('Author').groupby('Author').count()[1:].plot(y = 'Title', kind = 'hist', logy = True, figsize = (25,10))
plt.title('Number of Publications per Author')
```

Out[45]: Text(0.5, 1.0, 'Number of Publications per Author')

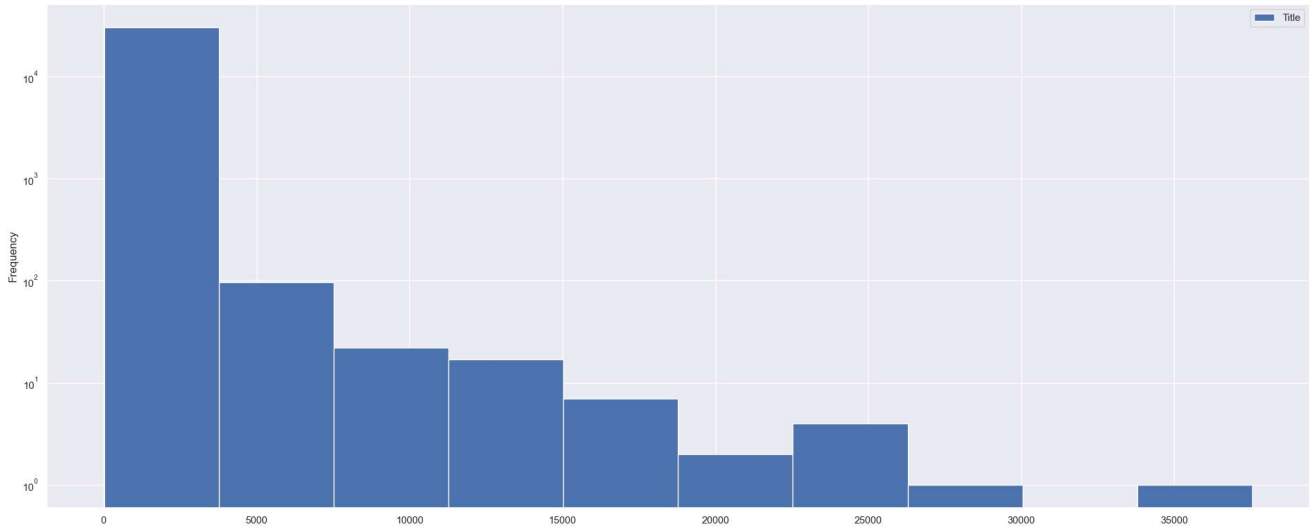


In [46]:
```
textdata_1 = textdata.explode('Author').groupby('Author').count()['Title']
mean = np.mean(textdata_1)
std_dev = np.std(textdata_1)

q1 = np.percentile(textdata_1, 25)
q2 = np.percentile(textdata_1, 50)
q3 = np.percentile(textdata_1, 75)

print("Mean:", mean)
print("Standard Deviation:", std_dev)
print("Q1 (First Quartile):", q1)
print("Q2 (Second quartile or median):", q2)
print("Q3 (Third quartile):", q3)

del textdata_1
```

```
Mean: 3.500984691486151
Standard Deviation: 118.00874824451012
Q1 (First Quartile): 1.0
Q2 (Second quartile or median): 1.0
Q3 (Third quartile): 3.0
```

In [47]:
```
textdata.groupby('Publication Venue').count().plot(y = 'Title', kind = 'hist', logy = True, figsize = (25,10))
```

Out[47]:
```
<AxesSubplot:ylabel='Frequency'>
```



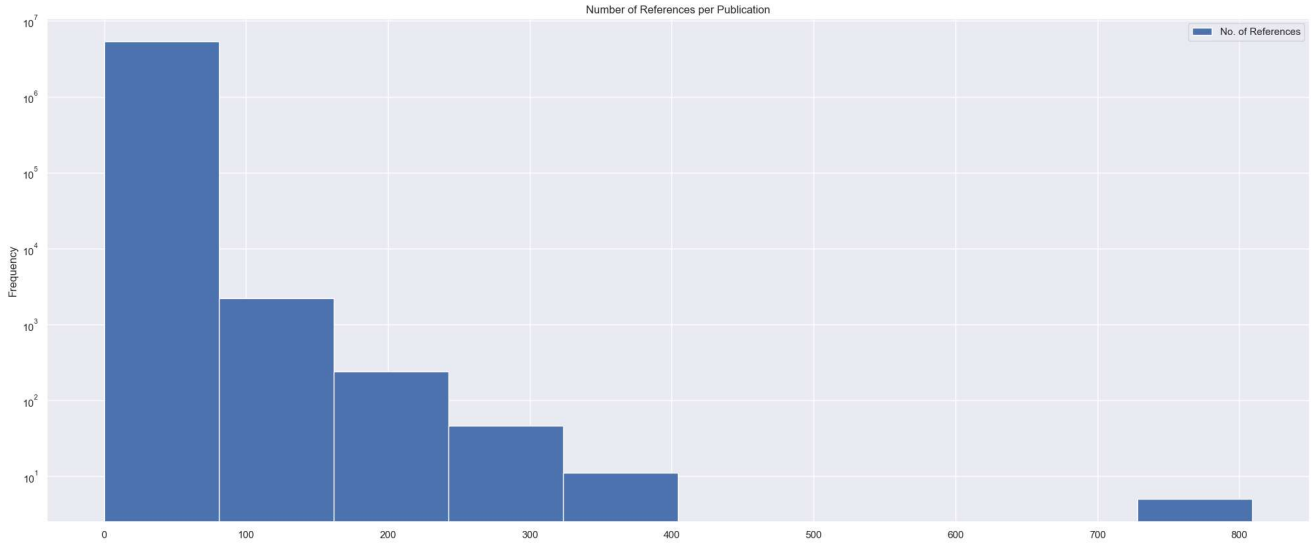
In [48]:
```
textdata_1 = textdata.groupby('Publication Venue').count()
mean = np.mean(textdata_1['Title'])
std_dev = np.std(textdata_1['Title'])

q1 = np.percentile(textdata_1['Title'], 25)
q2 = np.percentile(textdata_1['Title'], 50)
q3 = np.percentile(textdata_1['Title'], 75)

print("Mean:", mean)
print("Standard Deviation:", std_dev)
print("Q1 (First Quartile):", q1)
print("Q2 (Second quartile or median):", q2)
print("Q3 (Third quartile):", q3)
```

```
Mean: 176.2768660145736
Standard Deviation: 733.7731970404155
Q1 (First Quartile): 30.0
Q2 (Second quartile or median): 64.0
Q3 (Third quartile): 142.0
```

In [49]:
```
textdata['References'] = textdata['References'].apply(lambda x: x.split('; '))
textdata['No. of References'] = textdata['References'].apply(lambda x: 0 if x== [''] else len(x))
textdata.plot(y = 'No. of References', kind = 'hist', logy = True, figsize = (25,10))
plt.title('Number of References per Publication')
```

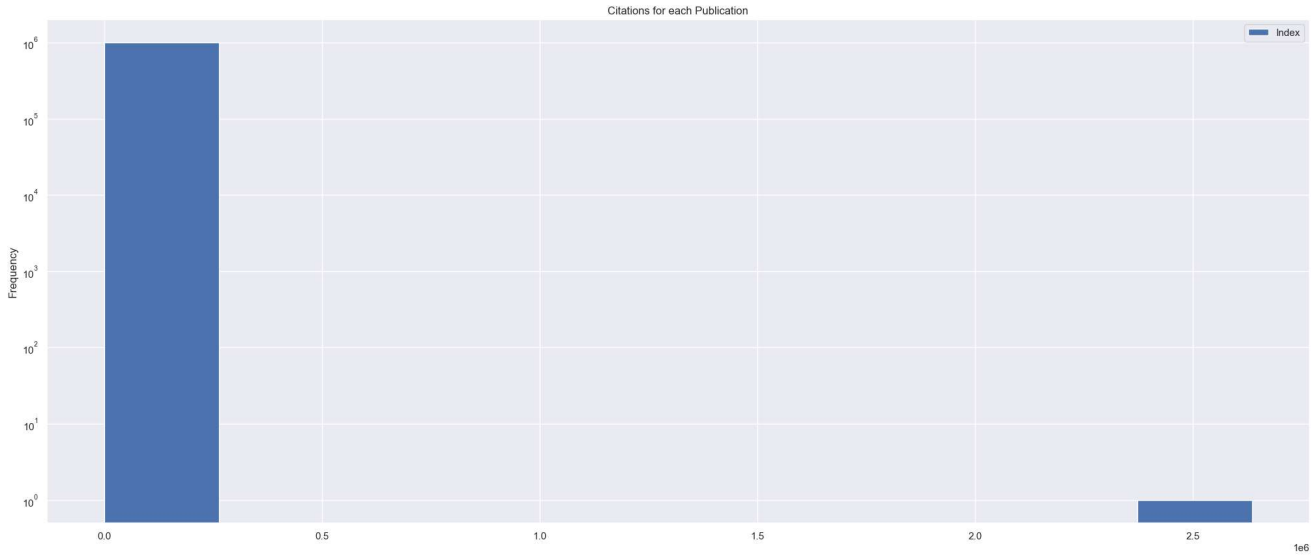
```
C:\Users\ayush\AppData\Local\Temp\ipykernel_8996\3058429958.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  textdata['References'] = textdata['References'].apply(lambda x: x.split('; '))
C:\Users\ayush\AppData\Local\Temp\ipykernel_8996\3058429958.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  textdata['No. of References'] = textdata['References'].apply(lambda x: 0 if x== [''] else len(x))
```

Out[49]: Text(0.5, 1.0, 'Number of References per Publication')



In [50]:
```
textdata.explode('References').groupby('References').count()[1:].plot(y = 'No_of_Citations', kind = 'hist',
                                                                      logy = True, figsize = (25,10))
plt.title('Citations for each Publication')
```

Out[50]: Text(0.5, 1.0, 'Citations for each Publication')



In [51]:
```
textdata
```

Out[51]:

	Title	Author	Year	Publication Venue	Index	References	Abstract	No_of_Citations	Number_of_publications	No. of References
0	MOSFET table look-up models for circuit simula...	[]	1984	Integration, the VLSI Journal	1	[]		0.0	2335	0
1	The verification of the protection mechanisms ...	[Virgil D. Gligor]	1984	International Journal of Parallel Programming	2	[]		0.0	1749	0
2	Another view of functional and multivalued dep...	[M. Gyssens]	1984	International Journal of Parallel Programming	3	[]		6.0	1749	0
3	Another view of functional and multivalued dep...	[J. Paredaens]	1984	International Journal of Parallel Programming	3	[]		6.0	1749	0
4	Entity-relationship diagrams which are in BCNF	[Sushil Jajodia]	1984	International Journal of Parallel Programming	4	[]		8.0	1749	0
...	...	...	...	...	...	...	...	...	...	...
5806862	Foreword	[-]	2016	Theoretical Computer Science	2385061	[]		0.0	17259	0
5806863	Editorial Board	[]	2016	Theoretical Computer Science	2385062	[]		0.0	17259	0
5806864	Linear-time computation of prefix table for we...	[-]	2016	Theoretical Computer Science	2385063	[2381731]	The prefix table of a string is one of the mos...	0.0	17259	1
5806865	A space-efficient alphabet-independent Four-Ru...	[-]	2016	Theoretical Computer Science	2385064	[2381731]	Given two strings X (| X | = m) and Y (| Y ...	0.0	17259	1
5806868		[]				[]		0.0	436	0

5370451 rows × 10 columns

Observations:
The histogram doesn't change. The mean also changes in this case.

Part J

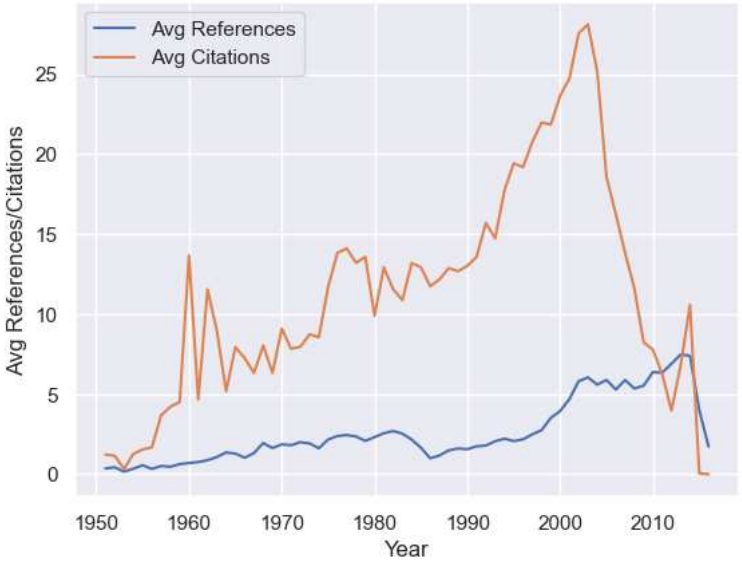
In [52]:
```
textdata['Year'] = pd.to_numeric(textdata['Year'], errors='coerce')
textdata = textdata[pd.notna(textdata['Year'])]
textdata = textdata.groupby('Year').agg(Avg_References = ('No. of References',np.mean),
                                        Avg_Citations = ('No_of_Citations', np.mean))
textdata.head()
```

```
C:\Users\ayush\AppData\Local\Temp\ipykernel_8996\2232921417.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  textdata['Year'] = pd.to_numeric(textdata['Year'], errors='coerce')
```

Out[52]:

Year	Avg_References	Avg_Citations
1951.0	0.360656	1.229508
1952.0	0.428571	1.152381
1953.0	0.162162	0.324324
1954.0	0.345133	1.265487
1955.0	0.558824	1.539216

In [53]:
```
import seaborn as sns
sns.lineplot(x = textdata.index, y = "Avg_References", data = textdata, label = 'Avg References')
sns.lineplot(x = textdata.index, y = "Avg_Citations", data = textdata, label = 'Avg Citations')
plt.xlabel('Year')
plt.ylabel('Avg References/Citations')
plt.legend()
plt.show()
```



Observation:
We can see that with increasing number of references, the number of citation also increase tremendously. There is a sudden spike and dip from the year 1990 to 2010.

In [ ]:

Section A

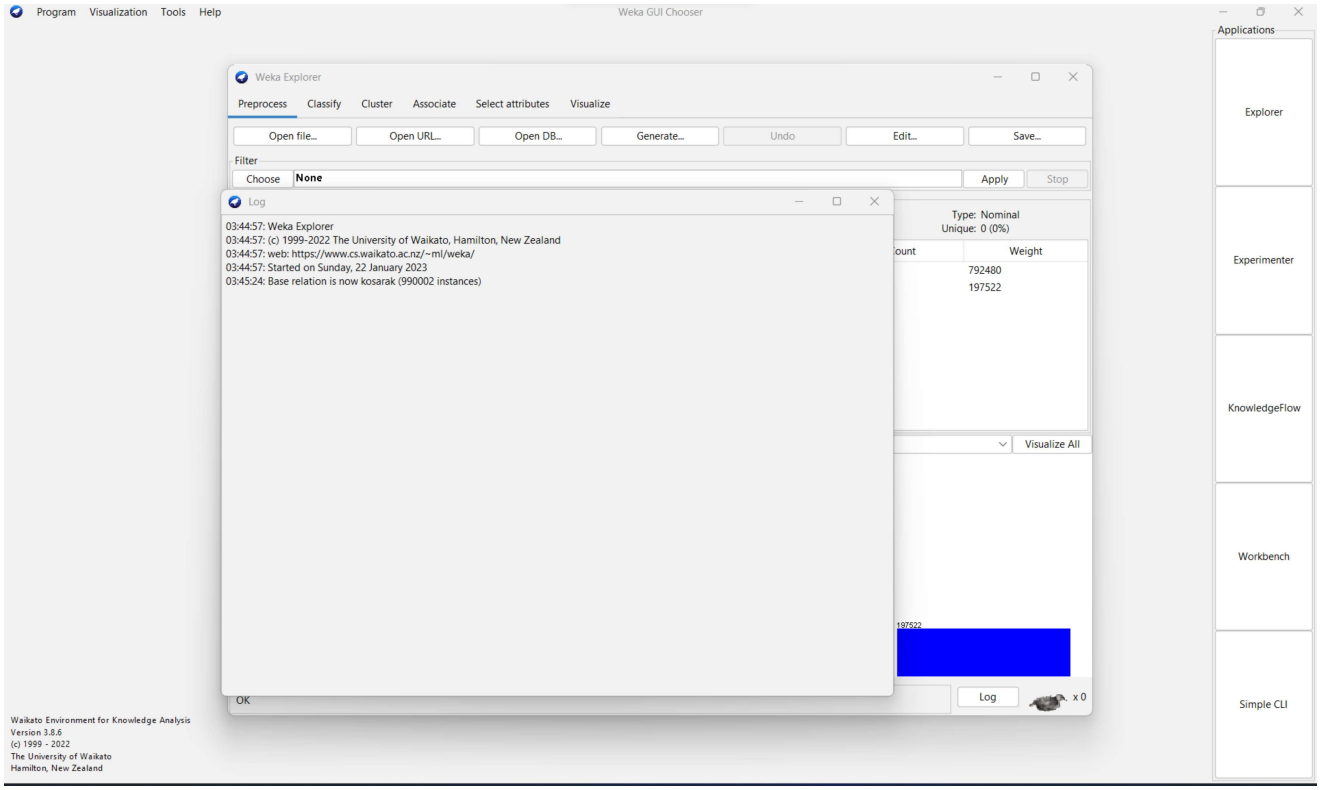
In [2]:
```
import time
start_time = time.process_time()

with open('kosarak.dat') as f:
  lines = f.read().split( '\n')

articles = []
users = []

for line in lines:
  stripped_line = line.rstrip()
  if stripped_line != "":
    line_components = stripped_line.split(' ')
    user_articles = set()

    for article in line_components:
      if article != "":
        article_num = int(article)
        if article_num not in articles:
          articles.append (article_num)
        user_articles.add(article_num)
    users.append(sorted(user_articles))

article_strings = [("@attribute news%d {0,1}" % (num - 1)) for num in articles]
user_strings = ["{%s}" % ",".join ("%d 1" % (num - 1) for num in user) for user in users]
all_lines = ["@relation kosarak"] + article_strings + ["@data"] + user_strings
with open("kosarak.arff", "w") as f:
  for line in all_lines:
    f.write(line + "\n")

print(f"Time elapsed: {time.process_time() - start_time} seconds")
```

```
Time elapsed: 237.79842923700002 seconds
```

Section B

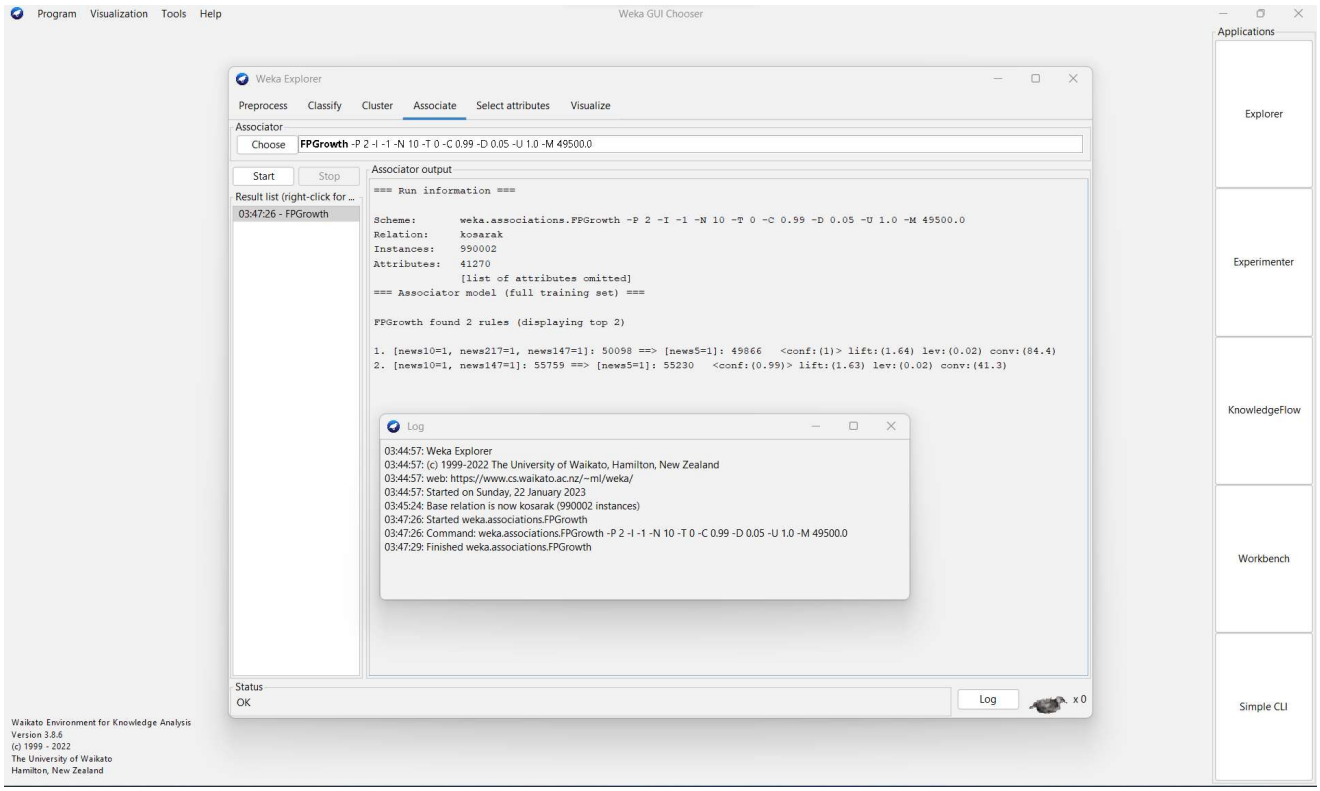
The time taken for converting kosark.dat to kosark.arff file is 237.79 seconds.

Section C



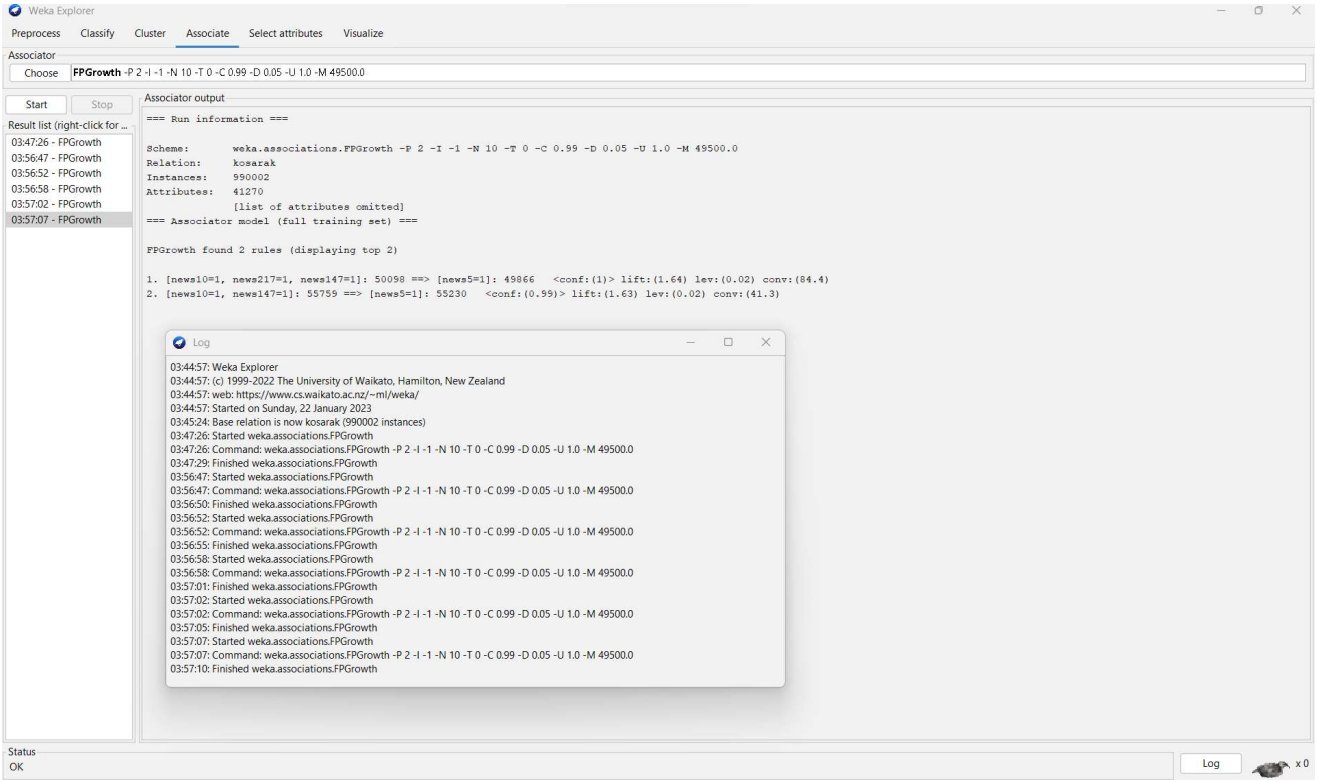
The time taken to load the file in Weka was 27 seconds.

Section D



Type *Markdown* and LaTeX: α^2

Section E



For the first time it took 3 seconds.

For the second time it took 7 seconds.

The third time it took 6 seconds.

The fourth time it took 4 seconds.

The fifth time took 5 seconds.

The Average time taken is 5 seconds which is way less than the time taken to convert the dataset and then load into Weka.

MNIST

In [1]:
```
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
import numpy as np
from statistics import mean
from sklearn.metrics.pairwise import euclidean_distances


# Load MNIST dataset
mnist = fetch_openml("mnist_784")
X = mnist.data
y = mnist.target

# Create a dataframe from the data and labels
df = pd.DataFrame(X)
df['label'] = y

# Randomize the dataframe
df = df.sample(frac=1, random_state = 42)

# Normalize the data
df.iloc[:, :-1] = df.iloc[:, :-1] / 255.0

# Split the dataframe into train, validation, and test sets
train_df, val_and_test_df = train_test_split(df, test_size = 0.2, random_state = 42)
val_df, test_df = train_test_split(val_and_test_df, test_size = 0.5, random_state = 42)
```

In [2]:
```
# Access train data
X_train = train_df.iloc[:, :-1]
y_train = train_df.iloc[:, -1]

# Access validation data
X_val = val_df.iloc[:, :-1]
y_val = val_df.iloc[:, -1]

# Access test data
X_test = test_df.iloc[:, :-1]
y_test = test_df.iloc[:, -1]
```

In [3]:
```
# Here, I have taken threshold as 784 because each data point has 784 features (28x28 image) and each feature can have a differen
def edit_distance(df, index, threshold):
    x = df.iloc[index, :-1].to_numpy()
    edit_sim = []
    for i in range(df.shape[0]):
        y = df.iloc[i, :-1].to_numpy()
        diff = np.sum(x != y)
        if diff < threshold:
            edit_sim.append(diff)
    return edit_sim
```

In [4]:
```
edit_mat = []
threshold = 784
for i in range(train_df[:100].shape[0]):
  edit_mat.append(edit_distance(train_df[:100], i, threshold=threshold))
```

In [5]:
```
# Edit distance for train set
pd.DataFrame(edit_mat)
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
0	0	250	271	294	282	226	259	265	212	240	...	291	257	298	193	221	260	295	230	223	235
1	250	0	234	283	268	222	248	278	226	217	...	255	216	295	193	219	240	315	206	214	229
2	271	234	0	258	309	264	269	281	255	244	...	259	244	296	205	233	259	318	250	239	277
3	294	283	258	0	292	271	270	257	247	255	...	258	241	283	230	266	277	293	253	239	286
4	282	268	309	292	0	221	299	264	241	205	...	256	268	292	226	271	301	282	238	252	268
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	260	240	259	277	301	256	228	284	219	251	...	259	222	294	211	235	0	325	208	223	254
96	295	315	318	293	282	267	303	242	281	285	...	272	292	270	249	277	325	0	281	279	281
97	230	206	250	253	238	190	234	237	193	186	...	243	189	268	169	195	208	281	0	152	215
98	223	214	239	239	252	197	229	227	195	193	...	243	169	257	147	186	223	279	152	0	222
99	235	229	277	286	268	230	257	265	197	226	...	260	245	290	183	197	254	281	215	222	0

100 rows × 100 columns

In [6]:
```
edit_mat = []
threshold = 784
for i in range(val_df[:100].shape[0]):
  edit_mat.append(edit_distance(val_df[:100], i, threshold=threshold))
```

In [7]:
```
# Edit distance for validation set
pd.DataFrame(edit_mat)
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
0	0	262	197	283	273	257	164	194	228	165	...	241	177	219	162	216	256	174	180	215	196
1	262	0	282	270	322	305	267	289	275	260	...	254	265	262	246	221	278	268	267	302	276
2	197	282	0	295	276	265	197	226	265	212	...	250	183	209	175	247	291	174	181	225	215
3	283	270	295	0	287	288	266	298	266	286	...	266	257	276	246	230	299	262	266	302	268
4	273	322	276	287	0	239	259	296	285	270	...	325	246	252	243	302	313	226	242	276	259
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	256	278	291	299	313	317	258	298	294	273	...	277	263	273	262	231	0	277	270	277	285
96	174	268	174	262	226	217	152	200	242	185	...	240	143	196	135	221	277	0	152	202	171
97	180	267	181	266	242	226	177	212	245	199	...	234	156	195	139	223	270	152	0	188	209
98	215	302	225	302	276	270	219	228	282	221	...	273	192	249	162	263	277	202	188	0	243
99	196	276	215	268	259	231	181	216	235	198	...	262	161	220	164	232	285	171	209	243	0

100 rows × 100 columns

In [8]:
```
edit_mat = []
threshold = 784
for i in range(test_df[:100].shape[0]):
  edit_mat.append(edit_distance(test_df[:100], i, threshold=threshold))
```

In [9]:
```
# Edit distance for test set
pd.DataFrame(edit_mat)
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
0	0	196	229	208	195	246	220	263	225	263	...	232	207	187	213	224	311	266	241	257	197
1	196	0	194	155	164	261	196	226	166	249	...	187	166	105	184	196	251	205	212	219	177
2	229	194	0	195	205	284	210	245	219	289	...	235	232	195	207	225	279	254	243	268	192
3	208	155	195	0	139	261	179	214	194	267	...	158	204	151	165	187	254	216	193	219	169
4	195	164	205	139	0	243	186	213	199	264	...	190	195	161	179	199	263	222	186	230	153
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	311	251	279	254	263	284	256	251	252	329	...	269	304	246	253	269	0	285	275	273	262
96	266	205	254	216	222	291	238	264	218	298	...	249	230	203	240	253	285	0	254	251	237
97	241	212	243	193	186	278	226	232	230	294	...	195	246	211	217	233	275	254	0	243	211
98	257	219	268	219	230	243	231	234	242	237	...	230	246	215	238	266	273	251	243	0	223
99	197	177	192	169	153	259	174	201	206	260	...	201	204	177	185	198	262	237	211	223	0

100 rows × 100 columns

In [10]:
```
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial import distance
```

In [11]:
```
cosine_similarity = cosine_similarity(X_train[0:100],X_test[0:100])
pd.DataFrame(cosine_similarity).head()
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	
0	0.551144	0.469566	0.448152	0.245708	0.356309	0.452462	0.257069	0.423434	0.356337	0.561108	...	0.256095	0.464106	0.398832	0.242606	0.174993	0.3
1	0.382656	0.370046	0.530117	0.398405	0.478537	0.425578	0.615472	0.843360	0.422943	0.485541	...	0.383439	0.424420	0.288372	0.392560	0.275894	0.4
2	0.310248	0.457490	0.339790	0.198726	0.357188	0.395585	0.459584	0.384409	0.495801	0.496352	...	0.171090	0.376262	0.348728	0.256892	0.193711	0.3
3	0.485382	0.418995	0.480378	0.305639	0.330039	0.477089	0.244072	0.287121	0.488368	0.458880	...	0.265638	0.331755	0.368072	0.371919	0.455567	0.4
4	0.122312	0.150346	0.307128	0.277546	0.294933	0.300244	0.226405	0.356087	0.198791	0.342281	...	0.444025	0.173181	0.138457	0.356161	0.399185	0.4

5 rows × 100 columns

In [12]:
```
from sklearn.metrics.pairwise import euclidean_distances
euclidean_dist = euclidean_distances(X_train[0:100],X_test[0:100])
pd.DataFrame(euclidean_dist).head()
```

Out[12]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	9
0	8.916407	9.427856	10.500139	10.435508	10.241900	10.932743	11.341094	10.575030	10.488715	10.940891	...	11.068141	9.676412	9.353736	10.83383
1	10.698294	10.526647	9.878821	9.715684	9.504654	11.383048	8.403069	5.627629	10.190262	11.900676	...	10.361854	10.273074	10.411925	10.03041
2	11.499357	9.983889	11.883041	11.292863	10.719255	11.825811	10.114415	11.325102	9.725265	11.880702	...	12.188073	10.881559	10.246906	11.25748
3	9.919100	10.276556	10.509710	10.543785	10.888997	10.972511	11.876366	12.146666	9.754720	12.273801	...	11.448965	11.214011	10.064563	10.36038
4	12.465427	11.906367	11.774503	10.246070	10.722968	12.350526	11.585536	11.187023	11.705626	13.217511	...	9.607601	12.018946	11.035029	10.03489

5 rows × 100 columns

In [13]:
```
from sklearn.metrics.pairwise import manhattan_distances
man_dist = manhattan_distances(X_train[0:100],X_test[0:100])
pd.DataFrame(man_dist).head()
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	
0	107.733333	111.380392	138.925490	137.627451	130.298039	150.694118	154.607843	143.823529	139.278431	148.180392	...	151.921569	120.231373	111.69
1	140.792157	130.023529	122.925490	118.372549	110.956863	159.509804	93.635294	53.031373	130.658824	168.039216	...	133.137255	128.835294	131.12
2	160.066667	121.541176	168.560784	153.701961	136.756863	169.600000	125.764706	156.886275	122.521569	169.164706	...	175.164706	144.556863	129.75
3	129.329412	130.364706	139.243137	139.529412	144.050980	153.043137	168.925490	179.341176	124.874510	182.058824	...	159.227451	154.196078	127.50
4	185.403922	165.129412	167.662745	133.203922	140.054902	186.286275	162.239216	155.580392	167.662745	206.172549	...	119.011765	172.890196	148.30

5 rows × 100 columns

20 NEWS GROUPS

In [14]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MaxAbsScaler
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.metrics.pairwise import manhattan_distances

# fetch the 20NG dataset
newsgroups_train = fetch_20newsgroups(subset='train')

# create a dataframe of the 20NG data
df = pd.DataFrame({'text': newsgroups_train.data, 'target': newsgroups_train.target})

# randomize the dataframe
df = df.sample(frac=1).reset_index(drop=True)

# create a tf-idf vectorizer
tfidf_vectorizer = TfidfVectorizer(stop_words = 'english')

# fit and transform the text data
X = tfidf_vectorizer.fit_transform(df['text'])

# split the data into train, validation and test sets
X_train, X_val_and_test, y_train, y_val_and_test = train_test_split(X, df['target'], test_size=0.2)
X_val, X_test, y_val, y_test = train_test_split(X_val_and_test, y_val_and_test, test_size=0.5)
```

In [16]:
```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity = cosine_similarity(X_train,X_test)
pd.DataFrame(cosine_similarity).head()
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9	...	1122	1123	1124	1125	1126	
0	0.004910	0.001426	0.010954	0.014342	0.003912	0.003751	0.020691	0.010994	0.020765	0.004400	...	0.026427	0.013804	0.027962	0.001971	0.004403	0.0
1	0.003027	0.000324	0.008767	0.009195	0.002028	0.005409	0.019902	0.005434	0.009743	0.002358	...	0.021792	0.003670	0.004343	0.006511	0.010871	0.0
2	0.008855	0.001038	0.012186	0.012945	0.035480	0.006335	0.024275	0.000969	0.003301	0.064260	...	0.004119	0.000556	0.006689	0.005618	0.004950	0.0
3	0.011085	0.000617	0.008596	0.058642	0.016485	0.004405	0.002018	0.021895	0.003439	0.010428	...	0.056521	0.002391	0.006196	0.012246	0.006896	0.0
4	0.007261	0.015803	0.003588	0.018964	0.015183	0.014708	0.002032	0.003030	0.007964	0.014515	...	0.002886	0.014186	0.000373	0.006445	0.017943	0.0

5 rows × 1132 columns

In [17]:
```
euclidean_dist = euclidean_distances(X_train,X_test)
pd.DataFrame(euclidean_dist).head()
```

Out[17]:

	0	1	2	3	4	5	6	7	8	9	...	1122	1123	1124	1125	1126	
0	1.410737	1.413205	1.406446	1.404035	1.411445	1.411559	1.399506	1.406418	1.399453	1.411099	...	1.395402	1.404419	1.394301	1.412819	1.411097	1.4
1	1.412071	1.413985	1.408001	1.407697	1.412779	1.410384	1.400070	1.410366	1.407308	1.412545	...	1.398719	1.411616	1.411139	1.409602	1.406506	1.4
2	1.407938	1.413480	1.405570	1.405030	1.388899	1.409727	1.396943	1.413529	1.411878	1.368020	...	1.411298	1.413820	1.409476	1.410236	1.410709	1.4
3	1.406353	1.413777	1.408122	1.372121	1.402508	1.411095	1.412786	1.398645	1.411780	1.406820	...	1.373666	1.412522	1.409826	1.405528	1.409329	1.4
4	1.409070	1.402994	1.411674	1.400740	1.403436	1.403775	1.412776	1.412069	1.408571	1.403912	...	1.412171	1.404147	1.413950	1.409649	1.401469	1.4

5 rows × 1132 columns

In [18]:
```
man_dist = manhattan_distances(X_train,X_test)
pd.DataFrame(man_dist).head()
```

Out[18]:

	0	1	2	3	4	5	6	7	8	9	...	1122	1123	1124	112
0	10.298847	13.339463	10.643572	10.998267	13.866226	15.830739	12.983150	17.136788	18.466079	15.139718	...	13.412690	19.170001	12.174557	9.57560
1	11.547579	14.471015	11.876978	12.327189	14.975129	16.841070	14.206836	18.321850	19.595473	16.361411	...	14.579331	20.446363	13.700847	10.62730
2	8.244593	11.334138	8.675817	8.996399	11.546662	13.756210	11.059113	15.340335	16.823196	12.644677	...	11.813737	17.500855	10.591391	7.38401
3	9.007187	12.215520	9.534740	9.248408	12.549927	14.659765	12.164648	15.991820	17.712730	13.863844	...	12.359884	18.318587	11.496762	8.11438
4	12.994280	15.662312	13.521006	13.842600	16.178259	18.156696	15.950845	19.950129	21.238189	17.594512	...	16.486512	21.739306	15.447120	12.09687

5 rows × 1132 columns

In [21]:
```
# create a Count Vectorizer
count_vectorizer = CountVectorizer(stop_words = 'english')
# fit and transform the text data
X = count_vectorizer.fit_transform(newsgroups_train.data[:100])
```

In [22]:
```
def edit_distance(df, index, threshold):
    x = df[index]
    edit_sim = []
    for i in range(df.shape[0]):
        y = df[i]
        diff = np.sum(x != y)
        if diff > threshold:
            edit_sim.append(diff)
        else:
          edit_sim.append(0)
        edit_sim.append(diff)
    return edit_sim
```

In [23]:
```
edit_mat = []
threshold = 100
for i in range(X.shape[0]):
  edit_mat.append(edit_distance(X, i, threshold=threshold))
```

In [24]:
```
# Edit distance for 20NG
pd.DataFrame(edit_mat)
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	...	190	191	192	193	194	195	196	197	198	199
0	0	0	102	102	176	176	105	105	134	134	...	0	94	244	244	129	129	0	91	103	103
1	102	102	0	0	189	189	119	119	147	147	...	102	102	248	248	136	136	0	100	119	119
2	176	176	189	189	0	0	190	190	209	209	...	182	182	325	325	214	214	177	177	192	192
3	105	105	119	119	190	190	0	0	140	140	...	109	109	251	251	136	136	105	105	112	112
4	134	134	147	147	209	209	140	140	0	0	...	136	136	279	279	165	165	127	127	141	141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	0	94	102	102	182	182	109	109	136	136	...	0	0	243	243	122	122	0	94	109	109
96	244	244	248	248	325	325	251	251	279	279	...	243	243	0	0	271	271	235	235	237	237
97	129	129	136	136	214	214	136	136	165	165	...	122	122	271	271	0	0	125	125	142	142
98	0	91	0	100	177	177	105	105	127	127	...	0	94	235	235	125	125	0	0	0	99
99	103	103	119	119	192	192	112	112	141	141	...	109	109	237	237	142	142	0	99	0	0

100 rows × 200 columns

In [1]:
```
from sklearn.datasets import fetch_openml
from sklearn.metrics import euclidean_distances
from sklearn.model_selection import train_test_split
import numpy as np

class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        y_pred = []
        distances = euclidean_distances(X_test, self.X_train)
        for i in range(X_test.shape[0]):
            indices = np.argsort(distances[i])[:self.k]
            k_nearest_labels = [self.y_train[j] for j in indices]
            y_pred.append(max(k_nearest_labels, key=k_nearest_labels.count))
        return y_pred

    def score(self,X_test, y_test):
        y_pred = self.predict(X_test)
        return (y_pred == y_test).mean()

# load mnist dataset
mnist = fetch_openml('mnist_784')
data = mnist.data
labels = mnist.target

# divide the data into train, validation, and test sets
train_data, val_and_test_data, train_labels, val_and_test_labels = train_test_split(data, labels, test_size=0.2)
val_data, test_data, val_labels, test_labels = train_test_split(val_and_test_data, val_and_test_labels, test_size=0.5)

# train KNN classifier on train_data
knn = KNN(k=5)
knn.fit(train_data, train_labels)

# evaluate training performance
train_acc = knn.score(train_data, train_labels)
print("Training Accuracy:", train_acc)

# evaluate validation performance
val_acc = knn.score(val_data, val_labels)
print("Validation Accuracy:", val_acc)

# evaluate testing performance
test_acc = knn.score(test_data, test_labels)
print("Testing Accuracy:", test_acc)
```

```
Training Accuracy: 0.9833214285714286
Validation Accuracy: 0.9718571428571429
Testing Accuracy: 0.9728571428571429
```

In [2]:
```
from sklearn.metrics import classification_report

# predict labels for test data
y_pred = knn.predict(test_data)

# generate classification report
print(classification_report(test_labels, y_pred, target_names=np.unique(labels)))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       712
           1       0.96      0.99      0.98       757
           2       0.99      0.96      0.97       688
           3       0.97      0.97      0.97       704
           4       0.98      0.96      0.97       690
           5       0.97      0.97      0.97       622
           6       0.97      0.99      0.98       685
           7       0.96      0.98      0.97       771
           8       0.99      0.93      0.96       691
           9       0.95      0.96      0.96       680

    accuracy                           0.97      7000
   macro avg       0.97      0.97      0.97      7000
weighted avg       0.97      0.97      0.97      7000
```

In [3]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer

class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        y_pred = []
        distances = euclidean_distances(X_test, self.X_train)
        for i in range(X_test.shape[0]):
            indices = np.argsort(distances[i])[:self.k]
            k_nearest_labels = [self.y_train[j] for j in indices]
            y_pred.append(max(k_nearest_labels, key=k_nearest_labels.count))
        return y_pred

    def score(self,X_test, y_test):
        y_pred = self.predict(X_test)
        return (y_pred == y_test).mean()

# load 20 newsgroups dataset
newsgroups = fetch_20newsgroups(subset='all')
data = newsgroups.data
labels = newsgroups.target

# preprocessing and vectorization of the data
vectorizer = TfidfVectorizer()
data = vectorizer.fit_transform(data)

# divide the data into train, validation, and test sets
train_data, val_and_test_data, train_labels, val_and_test_labels = train_test_split(data, labels, test_size=0.2)
val_data, test_data, val_labels, test_labels = train_test_split(val_and_test_data, val_and_test_labels, test_size=0.5)

# train KNN classifier on train_data
knn = KNN(k=5)
knn.fit(train_data, train_labels)

# evaluate training performance
train_acc = knn.score(train_data, train_labels)
print("Training Accuracy:", train_acc)

# evaluate validation performance
val_acc = knn.score(val_data, val_labels)
print("Validation Accuracy:", val_acc)

# evaluate testing performance
test_acc = knn.score(test_data, test_labels)
print("Testing Accuracy:", test_acc)
```

```
Training Accuracy: 0.9127752719554258
Validation Accuracy: 0.8238726790450929
Testing Accuracy: 0.8164456233421751
```

In [4]:
```
# predict labels for test data
y_pred = knn.predict(test_data)

# generate classification report
print(classification_report(test_labels, y_pred, target_names=newsgroups.target_names))
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.74      0.91      0.81        75
           comp.graphics       0.78      0.75      0.76       119
 comp.os.ms-windows.misc       0.82      0.70      0.76       111
comp.sys.ibm.pc.hardware       0.68      0.73      0.71        82
   comp.sys.mac.hardware       0.80      0.75      0.77       103
          comp.windows.x       0.82      0.78      0.80        91
            misc.forsale       0.68      0.60      0.64        83
               rec.autos       0.88      0.86      0.87        92
         rec.motorcycles       0.86      0.87      0.87       101
      rec.sport.baseball       0.87      0.86      0.87       109
        rec.sport.hockey       0.89      0.95      0.92        96
               sci.crypt       0.84      0.91      0.87        99
         sci.electronics       0.81      0.71      0.76        94
                 sci.med       0.93      0.83      0.88        99
               sci.space       0.89      0.86      0.88        95
  soc.religion.christian       0.84      0.84      0.84        85
      talk.politics.guns       0.86      0.87      0.87       100
   talk.politics.mideast       0.81      0.97      0.88       105
      talk.politics.misc       0.70      0.83      0.76        84
      talk.religion.misc       0.77      0.69      0.73        62

                accuracy                           0.82      1885
               macro avg       0.81      0.81      0.81      1885
            weighted avg       0.82      0.82      0.82      1885
```