In [1]:

```python
import sys
print(sys.path)
```

```
['C:\\Users\\harsh\\OneDrive\\Desktop\\UNT Subjects\\NLP', 'C:\\Users\\harsh\\anaconda3\\envs\\tf-gpu\\python38.zi
p', 'C:\\Users\\harsh\\anaconda3\\envs\\tf-gpu\\DLLs', 'C:\\Users\\harsh\\anaconda3\\envs\\tf-gpu\\lib', 'C:\\Users
\\harsh\\anaconda3\\envs\\tf-gpu', '', 'C:\\Users\\harsh\\AppData\\Roaming\\Python\\Python38\\site-packages', 'C:\\U
sers\\harsh\\anaconda3\\envs\\tf-gpu\\lib\\site-packages', 'C:\\Users\\harsh\\anaconda3\\envs\\tf-gpu\\lib\\site-pac
kages\\win32', 'C:\\Users\\harsh\\anaconda3\\envs\\tf-gpu\\lib\\site-packages\\win32\\lib', 'C:\\Users\\harsh\\anaco
nda3\\envs\\tf-gpu\\lib\\site-packages\\Pythonwin']
```

Importing the required libraries

In [2]:

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
from nltk.probability import ConditionalFreqDist
import docx2txt
import docx
import random
import re
import matplotlib.pyplot as plt
```

In [3]:

```python

# Open the DOCX file
doc = docx.Document('5 papers related to blockchain.docx')
text = '\n'.join([paragraph.text for paragraph in doc.paragraphs])

# Remove unnecessary punctuations,spaces brackets
text = re.sub(r'\[\d+\]', '', text)
# Remove underscores and hyphens from the sides of words
text = re.sub(r'[-_]', '', text)
# Remove numbers used for points
text = re.sub(r'\d+\.', '', text)
# remove "
text = re.sub(r'"', '', text)
# remove '
text = re.sub(r"'", '', text)
# remove special characters
text = re.sub(r'[+-.,!@#$%^&<>?/\{}()*_=:;|]', '', text)
# remove multiple spaces
text = re.sub(r'\n\s*\n', '\n', text)
# remove numbers
text = re.sub(r'\s\d+\s', ' ', text)
# remove new line, tabs
text = re.sub(r'\n|\t', ' ', text)
# Remove bullets
text = re.sub(r'^[\s\u2022\u2023\u25E6\u2043]*', '', text, flags=re.MULTILINE)
# remove multiple spaces
text = re.sub(r' +', ' ', text)
text = text.lower()

# Write the cleaned text to a TXT file
with open('input.txt', 'w',encoding="utf-8") as file:
    file.write(text)

```

File reading starts from here

In [4]:

```python
filename = "input.txt"
with open(filename,encoding="utf-8") as f:
    content = f.readlines()
content = [ line for line in content if line != '\n' ]
content = [ line.lower() for line in content ]
```

In [5]:

```python
tokens = [t for l in content for t in l.split() ]
# tokens
```

In [6]:

```python
tl = len(tokens)
tl
```

Out[6]:

```
9703
```

Calculating Diversity score of entire data set

In [7]:

```python
# Calculate  unique words
unique_words = set(tokens)
num_unique_words = len(unique_words)

diversity_score = num_unique_words / len(tokens)
print(diversity_score)
```

```
0.20807997526538183
```

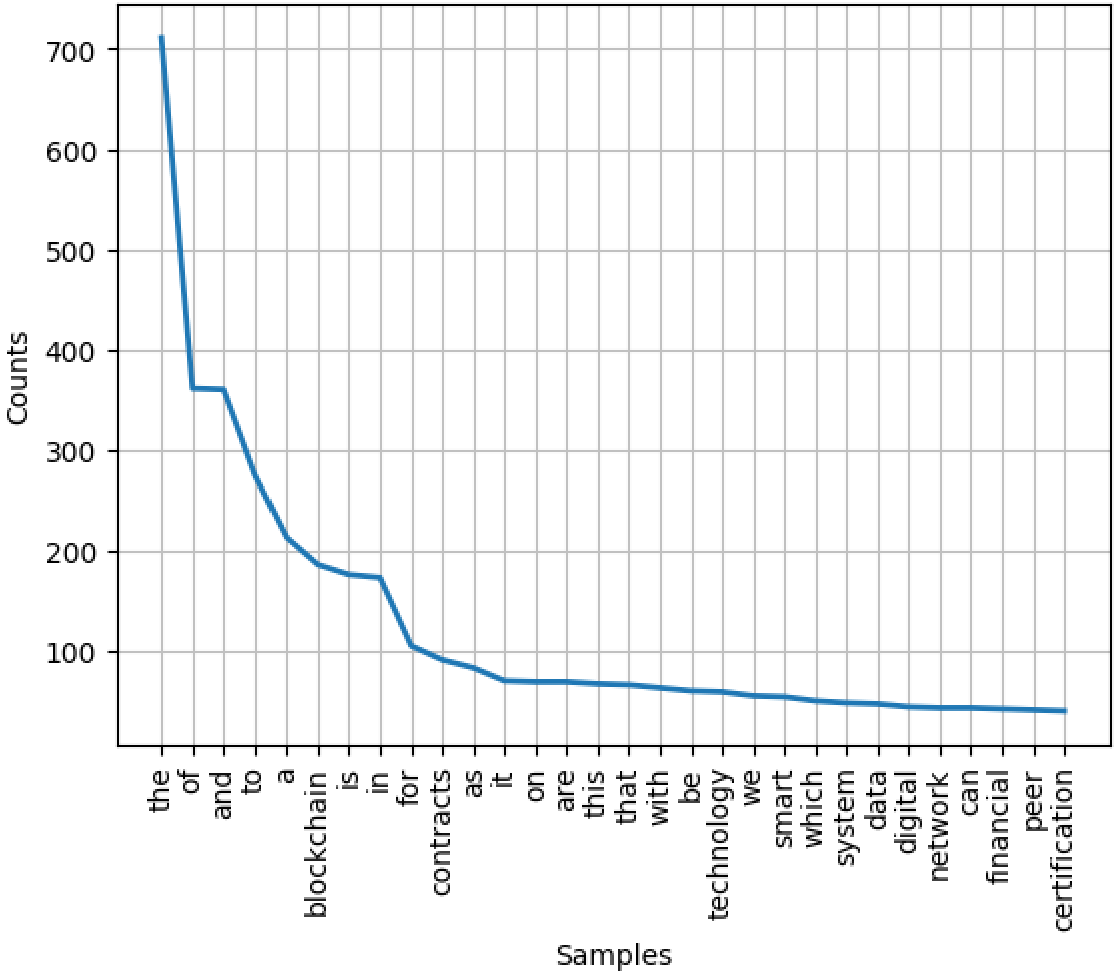
In [8]:

```python
freq = nltk.FreqDist(tokens)
probs = {k: v/tl for (k,v) in freq.items()}
pvals = list(probs.values())
cumprobs = {k: sum(pvals[0:ix+1]) for ix, (k,v) in enumerate(probs.items())}
```

Frequency plot for the generated tokens

In [9]:

```
freq.plot(30, cumulative=False)
plt.show()
```



In [10]:

```
cfd_bigram = nltk.ConditionalFreqDist(nltk.bigrams(tokens))
```

Heat map for bigrams with conditional Frequency distribution

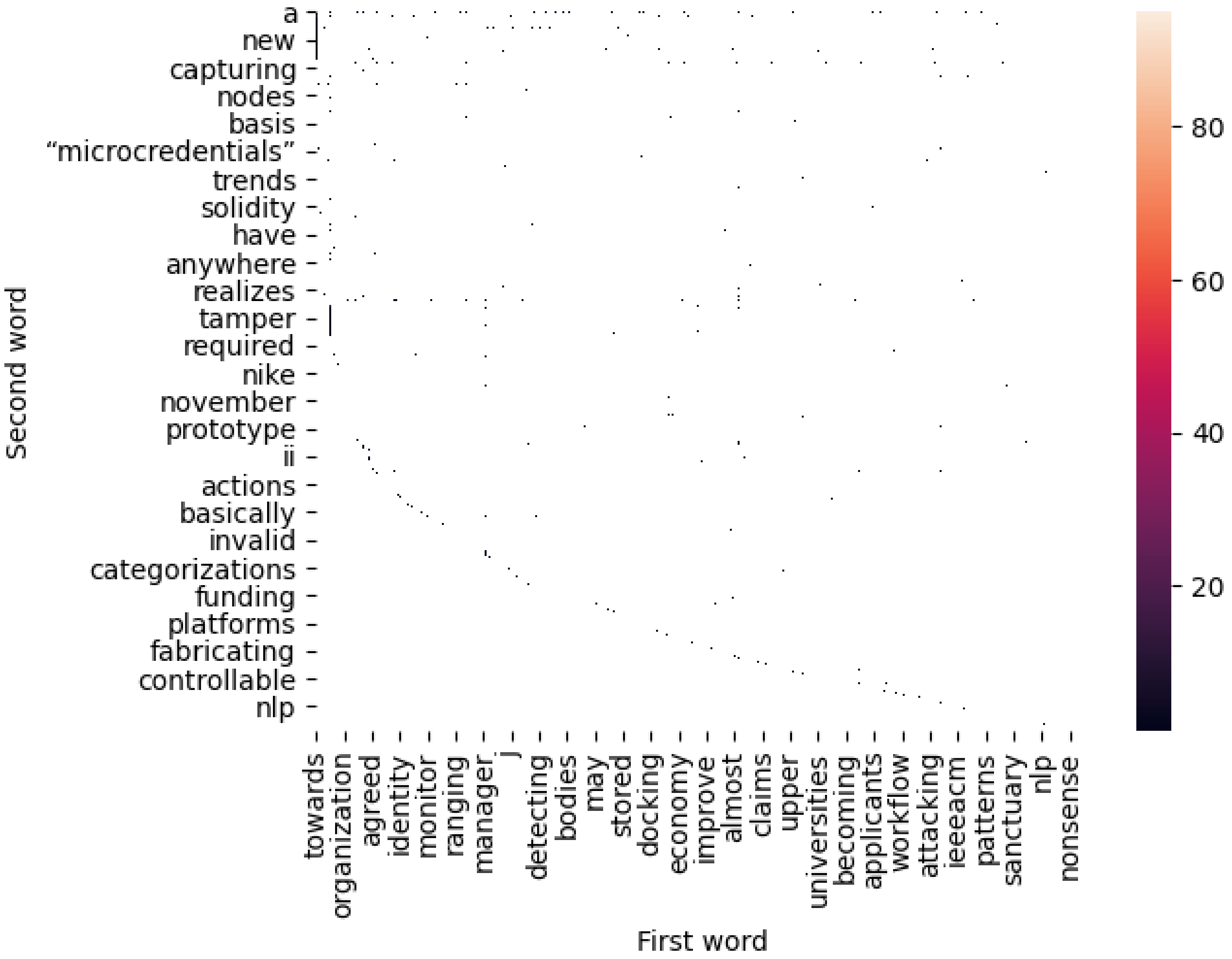
In [11]:

```python
import seaborn as sns
import pandas as pd

df = pd.DataFrame(cfd_bigram)

# Create the heatmap
sns.heatmap(df)
plt.xlabel("First word")
plt.ylabel("Second word")
plt.show()

```


a
new
capturing
nodes
basis
“microcredentials”
trends
solidity
have
anywhere
realizes
tamper
required
nike
november
prototype
ii
actions
basically
invalid
categorizations
funding
platforms
fabricating
controllable
nlp
Second word
towards
organization
agreed
identity
monitor
ranging
manager
j
detecting
bodies
may
stored
docking
economy
improve
almost
claims
upper
universities
becoming
applicants
workflow
attacking
ieeeacm
patterns
sanctuary
nlp
nonsense
First word
80
60
40
20

In [12]:

```
cfd_bigram.items()
```

```
1, 'supervision': 1, 'future': 1, 'main': 1, ...})), ('technical', FreqDist({'principles': 2, 'requirements': 2,
'limitations': 1, 'side': 1, 'characteristics': 1, 'defects': 1, 'means': 1, 'issues': 1, 'problems': 1})), ('limi
tations', FreqDist({'however': 1})), ('however', FreqDist({'the': 6, 'recognition': 1, 'as': 1, 'peer': 1, 'in':
1, 'with': 1, 'emerging': 1, 'by': 1, 'we': 1})), ('as', FreqDist({'a': 17, 'the': 9, 'well': 6, 'peer': 3, 'an':
2, 'blockchain': 2, 'follows': 2, 'chain': 2, 'security': 1, 'healthcare': 1, ...})), ('or', FreqDist({'a': 1, 'ad
ditions': 1, 'consortium': 1, 'individual': 1, 'the': 1, 'quality': 1, 'minimizing': 1, 'who': 1, 'authority': 1,
'bitcoin': 1, ...})), ('software', FreqDist({'component': 2, 'engineering': 2})), ('component', FreqDist({'require
s': 1, 'in': 1, 'of': 1})), ('requires', FreqDist({'a': 1, 'more': 1, 'regulatory': 1})), ('comprehensive', FreqDi
st({'understanding': 1, 'evaluation': 1})), ('characterization', FreqDist({'of': 1})), ('principles', FreqDist({'a
nd': 2, 'of': 1})), ('characteristics', FreqDist({'the': 3, 'of': 3, 'it': 1, 'such': 1, 'athe': 1, 'and': 1})),
('latter', FreqDist({'introduces': 2, 'is': 1})), ('introduces', FreqDist({'an': 2, 'several': 1, 'the': 1})), ('a
n', FreqDist({'important': 4, 'open': 3, 'architecture': 2, 'attempt': 2, 'uncertainty': 1, 'organization': 1, 'id
entical': 1, 'ongoing': 1, 'urgent': 1, 'isolated': 1, ...})), ('uncertainty', FreqDist({'for': 1})), ('organizati
on', FreqDist({'to': 1, 'this': 1})), ('decide', FreqDist({'which': 1})), ('which', FreqDist({'is': 13, 'blockchai
n': 4, 'are': 4, 'we': 3, 'includes': 2, 'will': 2, 'might': 2, 'aims': 1, 'type': 1, 'indexes': 1, ...})), ('prot
ocol', FreqDist({'to': 3, 'best': 1, 'is': 1, 'handles': 1, 'under': 1, 'in': 1, 'architecture': 1, 'can': 1})),
('best', FreqDist({'meets': 1})), ('meets', FreqDist({'its': 1})), ('needs', FreqDist({'to': 4, 'and': 1, 'safet
y': 1, 'the': 1, 'constant': 1, 'of': 1})), ('demands', FreqDist({'in': 1})), ('in', FreqDist({'the': 53, 'a': 13,
'this': 7, 'digital': 7, 'order': 6, 'terms': 5, 'particular': 5, 'addition': 4, 'many': 3, 'its': 3, ...})), ('ge
neral', FreqDist({'there': 1, 'inconsistency': 1, 'but': 1})), ('there', FreqDist({'are': 12, 'is': 6, 'was':
```

Calculating perplexity score of bigrams

In [13]:

```
import math

log_prob_sum = 0
num_tokens = len(tokens)
for i in range(num_tokens - 1):
    w0 = tokens[i]
    w1 = tokens[i+1]
    bigram_count = cfd_bigram[w0][w1]
    bigram_prob = (bigram_count + 1) / (cfd_bigram[w0].N() + len(cfd_bigram[w0]))
    log_prob_sum += math.log2(bigram_prob)

perplexity = math.pow(2, -log_prob_sum / num_tokens)
print(perplexity)

```

```
10.965655218617805
```

Calculating diversity score of bigrams

In [14]:

```
num_unique_bigrams = len(set(nltk.bigrams(tokens)))
diversity = num_unique_bigrams / len(list(nltk.bigrams(tokens)))
print(diversity)

```

```
0.7062461348175634
```

In [15]:

```python
def generate_bigram_sentence(data):
    # randomly select a starting word
    word = random.choice(list(data.keys()))
    sentence = [word]

    # generate the rest of the sentence using bigrams
    for i in range(9):
        if word not in data:
            break
        freq_dist = data[word]
        next_word = freq_dist.most_common(1)[0][0]
        sentence.append(next_word)
        word = next_word

    # join the sentence and return
    return " ".join(sentence)

for i in range(5):
    sentence = generate_bigram_sentence(cfd_bigram)
    print("Sentence : ",i+1)
    print(sentence)

```

```
Sentence :  1
architecture for the blockchain technology and the blockchain technology and
Sentence :  2
reads data and the blockchain technology and the blockchain technology
Sentence :  3
modules is a blockchain technology and the blockchain technology and
Sentence :  4
though the blockchain technology and the blockchain technology and the
Sentence :  5
faults in the blockchain technology and the blockchain technology and
```

In [16]:

```python
trigrams = nltk.trigrams(tokens)
```

In [17]:

```
condition_pairs = (((w0, w1), w2) for w0, w1, w2 in trigrams)
cfd_trigram = nltk.ConditionalFreqDist(condition_pairs)
```

Calculating perplexity score of trigrams

In [18]:

```
trigram_prob = 0.0
for w0_w1, freq_w2 in cfd_trigram.items():
    if '' in w0_w1:
        continue
    for w2 in freq_w2:
        trigram_count = cfd_trigram[w0_w1][w2]
        bigram_count = cfd_bigram[w0_w1[0]][w0_w1[1]]
        trigram_prob *= (trigram_count + 1) / (bigram_count + len(freq))


if trigram_prob == 0.0:
    perplexity = float('inf')
else:
    perplexity = math.pow(2, -1 * (math.log2(trigram_prob)))

print(perplexity)

```

```
inf
```

Calculating diversity score of trigrams

In [19]:

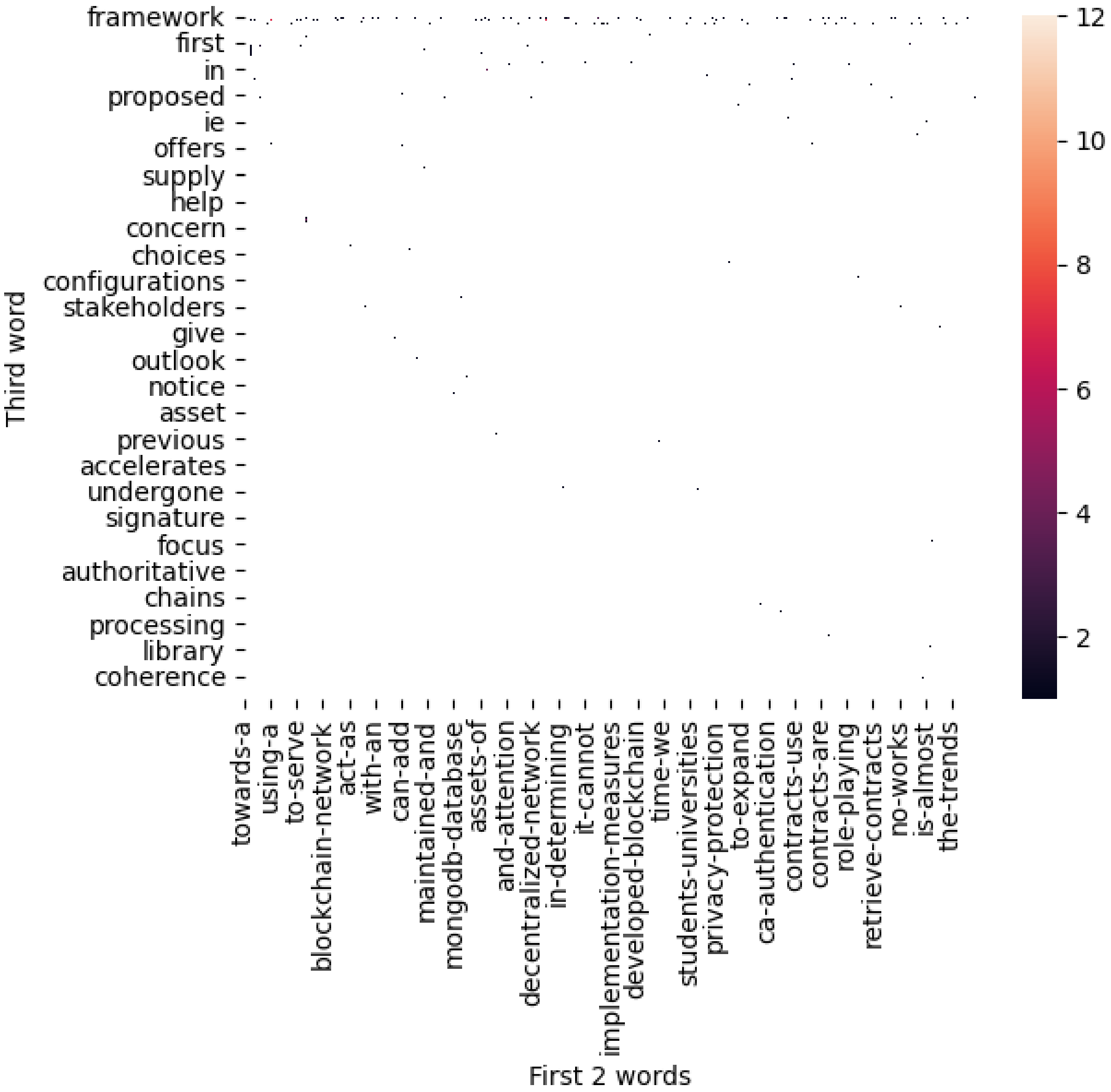
```
trigrams = list(nltk.trigrams(tokens))
diversity_score = len(set(trigrams)) / len(trigrams)
print(diversity_score)

```

```
0.920832903824348
```

In [20]:

```
df = pd.DataFrame(cfd_trigram)
sns.heatmap(df)
plt.xlabel("First 2 words")
plt.ylabel("Third word")
plt.show()
```


framework
first
in
proposed
ie
offers
supply
help
concern
choices
configurations
stakeholders
give
outlook
notice
asset
previous
accelerates
undergone
signature
focus
authoritative
chains
processing
library
coherence
Third word
towards-a
using-a
to-serve
blockchain-network
act-as
with-an
can-add
maintained-and
mongodb-database
assets-of
and-attention
decentralized-network
in-determining
it-cannot
implementation-measures
developed-blockchain
time-we
students-universities
privacy-protection
to-expand
ca-authentication
contracts-use
contracts-are
role-playing
retrieve-contracts
no-works
is-almost
the-trends
First 2 words
12
10
8
6
4
2

In [21]:

```
cfd_trigram.items()
```

```
e': 1})), (('evaluating', 'the'), FreqDist({'behaviour': 1, 'topic': 1})), (('the', 'behaviour'), FreqDist({'of':
1})), (('behaviour', 'of'), FreqDist({'blockchain': 1})), (('protocols', 'under'), FreqDist({'different': 1})),
(('under', 'different'), FreqDist({'test': 1})), (('different', 'test'), FreqDist({'scenarios': 1})), (('test', 's
cenarios'), FreqDist({'a': 1})), (('scenarios', 'a'), FreqDist({'distributed': 1})), (('a', 'distributed'), FreqDi
st({'ledger': 3, 'network': 1, 'ca': 1})), (('ledger', 'is'), FreqDist({'often': 1})), (('is', 'often'), FreqDist
({'described': 1})), (('often', 'described'), FreqDist({'as': 1})), (('described', 'as'), FreqDist({'a': 1})),
(('a', 'shared'), FreqDist({'distributed': 1, 'governance': 1})), (('shared', 'distributed'), FreqDist({'databas
e': 1})), (('distributed', 'database'), FreqDist({'which': 1, 'the': 1})), (('database', 'which'), FreqDist({'is':
1})), (('which', 'is'), FreqDist({'a': 4, 'accessed': 1, 'considered': 1, 'ideal': 1, 'an': 1, 'feasible': 1, 'joi
ntly': 1, 'impressive': 1, 'the': 1, 'also': 1})), (('is', 'accessed'), FreqDist({'and': 1})), (('accessed', 'an
d'), FreqDist({'maintained': 1})), (('and', 'maintained'), FreqDist({'by': 1})), (('maintained', 'by'), FreqDist
({'the': 2, 'a': 1})), (('by', 'a'), FreqDist({'set': 1})), (('of', 'independent'), FreqDist({'possibly': 1})),
(('independent', 'possibly'), FreqDist({'untrusted': 1})), (('possibly', 'untrusted'), FreqDist({'participants':
1})), (('untrusted', 'participants'), FreqDist({'ie': 1})), (('participants', 'ie'), FreqDist({'nodes': 1})), (('i
e', 'nodes'), FreqDist({'each': 1})), (('nodes', 'each'), FreqDist({'participant': 1, 'network': 1})), (('each',
'participant'), FreqDist({'owns': 1})), (('participant', 'owns'), FreqDist({'an': 1})), (('owns', 'an'), FreqDist
({'identical': 1})), (('an', 'identical'), FreqDist({'copy': 1})), (('identical', 'copy'), FreqDist({'of': 1})),
(('copy', 'of'), FreqDist({'the': 1})), (('of', 'the'), FreqDist({'blockchain': 12, 'application': 5, 'nodes': 4,
'financial': 4, 'network': 3, 'most': 3, 'system': 3, 'database': 2, 'state': 2, 'deployment': 2, ...})), (('the',
'database'), FreqDist({'of': 1, 'in': 1, 'and': 1, 'is': 1, 'at': 1})), (('database', 'of'), FreqDist({'transactio
```

In [22]:

```
def generate_trigram_sentence():
    sentence = []
    # Pick the first word at random
    w0 = random.choice(list(cfd_trigram.keys()))[0]
    sentence.extend([w0])

    # Pick the second word from bigram
    if w0 in cfd_trigram:
        w1 = random.choice(list(cfd_trigram[w0].keys()))
    else:
        # If w0 not present, pick random word from corpus
        w1 = random.choice(tokens)
    sentence.extend([w1])

    # Use those two words to pick the most frequent third word
    if w0 in cfd_trigram and w1 in cfd_trigram[w0]:
        w2 = cfd_trigram[w0][w1].most_common(1)[0][0]
    else:
        # If w0 or w1 not present, picking random word from corpus
        w2 = random.choice(tokens)
    sentence.extend([w2])

    # Iterate until 10 words
    for i in range(7):
        w0, w1, w2 = w1, w2, None
        if w0 in cfd_trigram and w1 in cfd_trigram[w0]:
            w2 = cfd_trigram[w0][w1].most_common(1)[0][0]
        if w2 is None:
            w2 = random.choice(tokens)
        sentence.extend([w2])

    return ' '.join(sentence)

for i in range(5):
    print('Sentence : ',i+1)
    print(generate_trigram_sentence())

```

```
Sentence :  1
final developers feasibility institutes the the regulatory digital the at
Sentence :  2
of blockchain generate with not system aspects peer security and
Sentence :  3
brains a development systemic feasible this application controlled a number
Sentence :  4
time questions protocol affect a the industrial database a structure
Sentence :  5
framework popular tremendous incorporating aspects in can to tolerance consensus
```

In [23]:

```python
quadgrams = nltk.ngrams(tokens,4)
```

In [24]:

```python
condition_pairs = (((w0, w1, w2), w3) for w0, w1, w2, w3 in quadgrams)
cfd_quadgram = nltk.ConditionalFreqDist(condition_pairs)
```

Calculating perplexity score of quadgrams

In [25]:

```python
quadgram_prob = 1.0
for w0, w1, w2, w3 in quadgrams:
    trigram_count = cfd_quadgram[(w0, w1, w2)][w3]
    bigram_count = cfd_trigram[(w0, w1)][w2]
    quadgram_prob *= (trigram_count + 1) / (bigram_count + len(freq))

perplexity = math.pow(2, -1 * (math.log2(quadgram_prob)))
print(perplexity)
```

```
1.0
```

Calculating diversity score of quadgrams

In [26]:

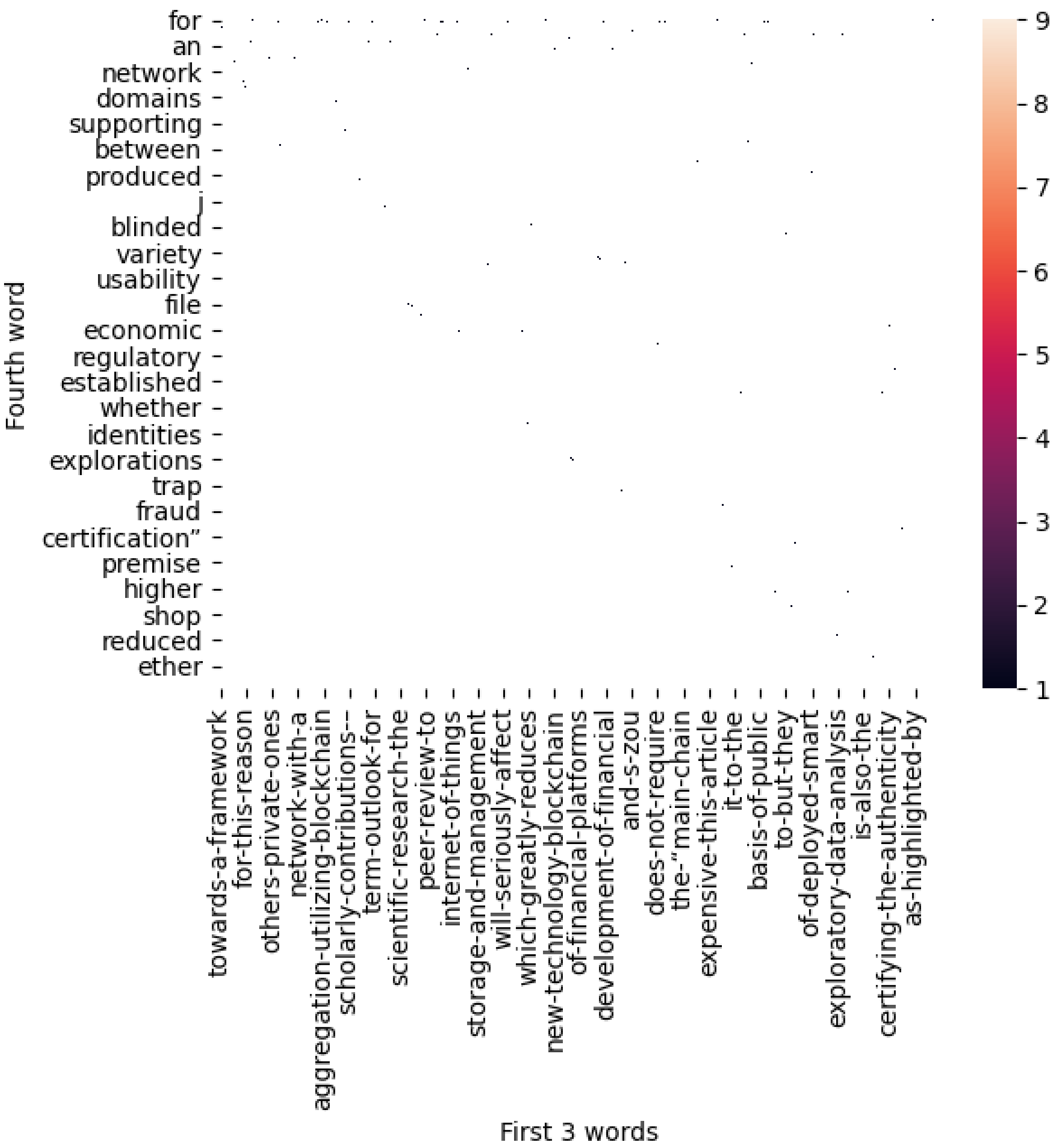
```python
diversity_score = 0.0
freq_quadgrams = nltk.FreqDist(quadgrams)
unique_quadgrams = set(quadgrams)
if len(freq_quadgrams) > 0 :
    diversity_score = len(unique_quadgrams) / len(freq_quadgrams)
print(diversity_score)
```

```
0.0
```

In [27]:

```
df = pd.DataFrame(cfd_quadgram)

# Create the heatmap
sns.heatmap(df)
plt.xlabel("First 3 words")
plt.ylabel("Fourth word")
plt.show()
```


Fourth word
for
an
network
domains
supporting
between
produced
j
blinded
variety
usability
file
economic
regulatory
established
whether
identities
explorations
trap
fraud
certification”
premise
higher
shop
reduced
ether
towards-a-framework
for-this-reason
others-private-ones
network-with-a
aggregation-utilizing-blockchain
scholarly-contributions--
term-outlook-for
scientific-research-the
peer-review-to
internet-of-things
storage-and-management
will-seriously-affect
which-greatly-reduces
new-technology-blockchain
of-financial-platforms
development-of-financial
and-s-zou
does-not-require
the-“main-chain
expensive-this-article
it-to-the
basis-of-public
to-but-they
of-deployed-smart
exploratory-data-analysis
is-also-the
certifying-the-authenticity
as-highlighted-by
First 3 words
9
8
7
6
5
4
3
2
1

```
In [28]:  cfd_quadgram.items()
```

```
'challenges': 1})), (('ledger', 'technology', 'dlt'), FreqDist({'appears': 1, 'was': 1})), (('technology', 'dlt',
'appears'), FreqDist({'to': 1})), (('dlt', 'appears', 'to'), FreqDist({'be': 1})), (('appears', 'to', 'be'), FreqD
ist({'at': 1})), (('to', 'be', 'at'), FreqDist({'a': 1})), (('be', 'at', 'a'), FreqDist({'worldwide': 1})), (('a
t', 'a', 'worldwide'), FreqDist({'threshold': 1})), (('a', 'worldwide', 'threshold'), FreqDist({'of': 1})), (('wor
ldwide', 'threshold', 'of'), FreqDist({'acceptance': 1})), (('threshold', 'of', 'acceptance'), FreqDist({'and':
1})), (('of', 'acceptance', 'and'), FreqDist({'adoption': 1})), (('acceptance', 'and', 'adoption'), FreqDist({'sin
ce': 1})), (('and', 'adoption', 'since'), FreqDist({'their': 1})), (('adoption', 'since', 'their'), FreqDist({'inc
eption': 1})), (('since', 'their', 'inception'), FreqDist({'several': 1})), (('their', 'inception', 'several'), Fr
eqDist({'innovative': 1})), (('inception', 'several', 'innovative'), FreqDist({'projects': 1})), (('several', 'inn
ovative', 'projects'), FreqDist({'have': 1})), (('innovative', 'projects', 'have'), FreqDist({'been': 1})), (('pro
jects', 'have', 'been'), FreqDist({'proposing': 1})), (('have', 'been', 'proposing'), FreqDist({'solutions': 1})),
(('been', 'proposing', 'solutions'), FreqDist({'to': 1})), (('proposing', 'solutions', 'to'), FreqDist({'the':
1})), (('solutions', 'to', 'the'), FreqDist({'blockchain': 1})), (('to', 'the', 'blockchain'), FreqDist({'trilemm
a': 1})), (('the', 'blockchain', 'trilemma'), FreqDist({'improving': 1})), (('blockchain', 'trilemma', 'improvin
g'), FreqDist({'blockchain': 1})), (('trilemma', 'improving', 'blockchain'), FreqDist({'features': 1})), (('improv
ing', 'blockchain', 'features'), FreqDist({'and': 1})), (('blockchain', 'features', 'and'), FreqDist({'its': 1})),
(('features', 'and', 'its'), FreqDist({'technical': 1})), (('and', 'its', 'technical'), FreqDist({'limitations':
1})), (('its', 'technical', 'limitations'), FreqDist({'however': 1})), (('technical', 'limitations', 'however'), F
reqDist({'the': 1})), (('limitations', 'however', 'the'), FreqDist({'adoption': 1})), (('however', 'the', 'adoptio
n'), FreqDist({'of': 1})), (('the', 'adoption', 'of'), FreqDist({'blockchain': 1})), (('adoption', 'of', 'blockcha
```

In [29]:

```python
for i in range(5):
    # select random word
    print('Sentence : ',i+1)
    w0, w1, w2 = random.choice(list(cfd_quadgram))

    sentence = [w0, w1, w2]

    # Using the conditional frequencies to generate the sentence
    for j in range(7):
        w3_freqdist = cfd_quadgram[(w0, w1, w2)]
        w3 = w3_freqdist.max()
        sentence.append(w3)
        w0, w1, w2 = w1, w2, w3

    print(" ".join(sentence))
```

```
Sentence :  1
technology and application development white paper blockchain reference framework and
Sentence :  2
produced by the network as well as a connectivity manager
Sentence :  3
with enough cryptocurrency to pay for the user gas costs
Sentence :  4
will not be conducive to the development trend of blockchain
Sentence :  5
have also demonstrated what can be achieved while pushing the
```

In [ ]: