https://www.youtube.com/watch?v=jnoKa44OZv8&t=3s

In [2]:
```
!pip install nltk
```

```
Collecting nltk
  Using cached nltk-3.9.1-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: click in /opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages (from nltk) (4.67.1)
Using cached nltk-3.9.1-py3-none-any.whl (1.5 MB)
Installing collected packages: nltk
Successfully installed nltk-3.9.1

[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: pip install --upgrade pip
```

In [10]:
```
import nltk
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to
[nltk_data]     /Users/dongunyun/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

Out[10]:
```
True
```

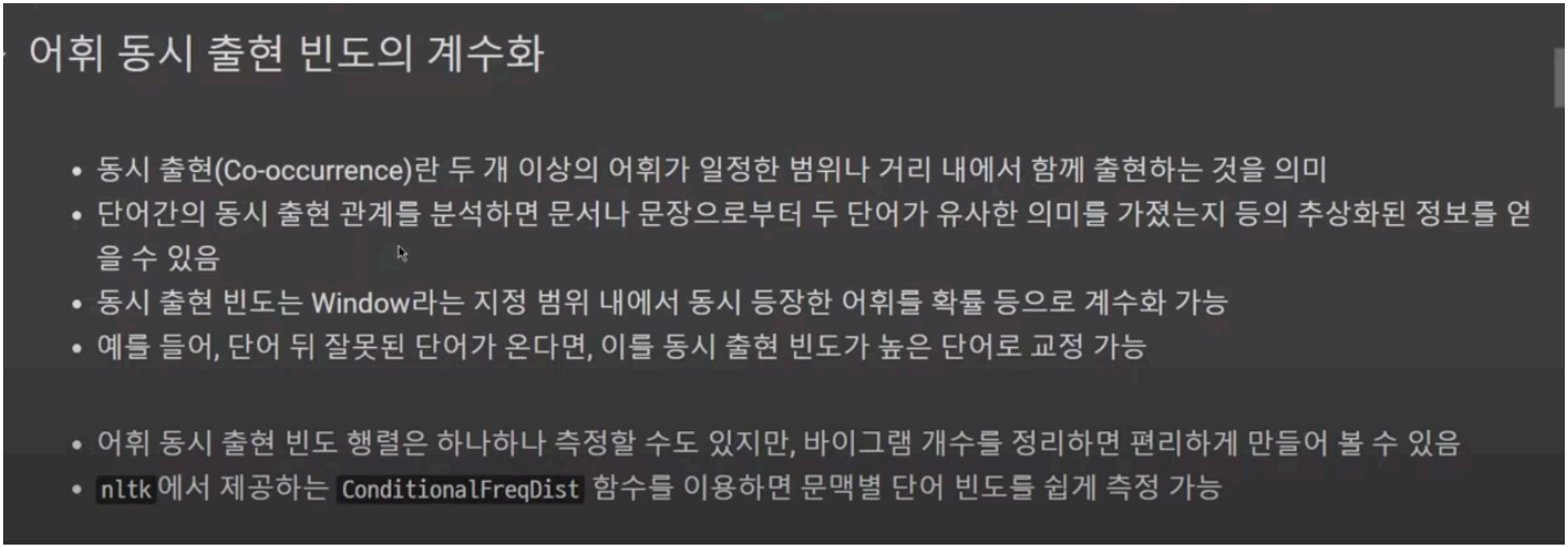
In [11]:
```
from nltk import word_tokenize, bigrams
```

In [12]:
```
sentence='I love data science and deep learning'
tokens= word_tokenize(sentence)
bgram = bigrams(tokens)
bgram_list =[x for x in bgram]
```

In [17]:
```
from nltk.util import ngrams

tgram= ngrams(tokens,3)
qgram= ngrams(tokens,4)
tgram_list= [x for x in tgram]
qgram_list= [x for x in qgram]
```

In [18]:
```
tgram_list
```

Out[18]:
```
[('I', 'love', 'data'),
 ('love', 'data', 'science'),
 ('data', 'science', 'and'),
 ('science', 'and', 'deep'),
 ('and', 'deep', 'learning')]
```



In [24]:
```
from nltk import ConditionalFreqDist

sentence=['I love data science and deep learning','I love science','I know this code']

tokens =[word_tokenize(x) for x in sentence]
bgrams = [bigrams(x) for x in tokens ]

token=[]
for i in bgrams:
    token+= ([x for x in i])
cfd = ConditionalFreqDist(token)
```

In [25]:
```
cfd.conditions()
```

Out[25]:
```
['I', 'love', 'data', 'science', 'and', 'deep', 'know', 'this']
```

In [26]:
```
print(cfd['I'])
```

```
<FreqDist with 2 samples and 3 outcomes>
```

In [29]:
```
print(cfd['I']['love'])#동시 출현 빈도
print(cfd['I'].most_common(1))
```

```
2
[('love', 2)]
```

In [31]:
```
import numpy as np

freq_matrix=[]
for i in cfd.keys():
    temp=[]
    for j in cfd.keys():
        temp.append(cfd[i][j])
    freq_matrix.append(temp)
freq_matrix=np.array(freq_matrix)
print(cfd.keys())
print(freq_matrix)
```

```
dict_keys(['I', 'love', 'data', 'science', 'and', 'deep', 'know', 'this'])
[[0 2 0 0 0 0 1 0]
 [0 0 1 1 0 0 0 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0]]
```

In [33]:
```
import pandas as pd

df= pd.DataFrame(freq_matrix, index=cfd.keys(),columns=cfd.keys())
df.style.background_gradient(cmap='coolwarm')
```

Out[33]:

	I	love	data	science	and	deep	know	this
I	0	2	0	0	0	0	1	0
love	0	0	1	1	0	0	0	0
data	0	0	0	1	0	0	0	0
science	0	0	0	0	1	0	0	0
and	0	0	0	0	0	1	0	0
deep	0	0	0	0	0	0	0	0
know	0	0	0	0	0	0	0	1
this	0	0	0	0	0	0	0	0

In [34]:
```
!pip install networkx
```

```
Requirement already satisfied: networkx in /opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages (3.2.
1)

[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: pip install --upgrade pip
```

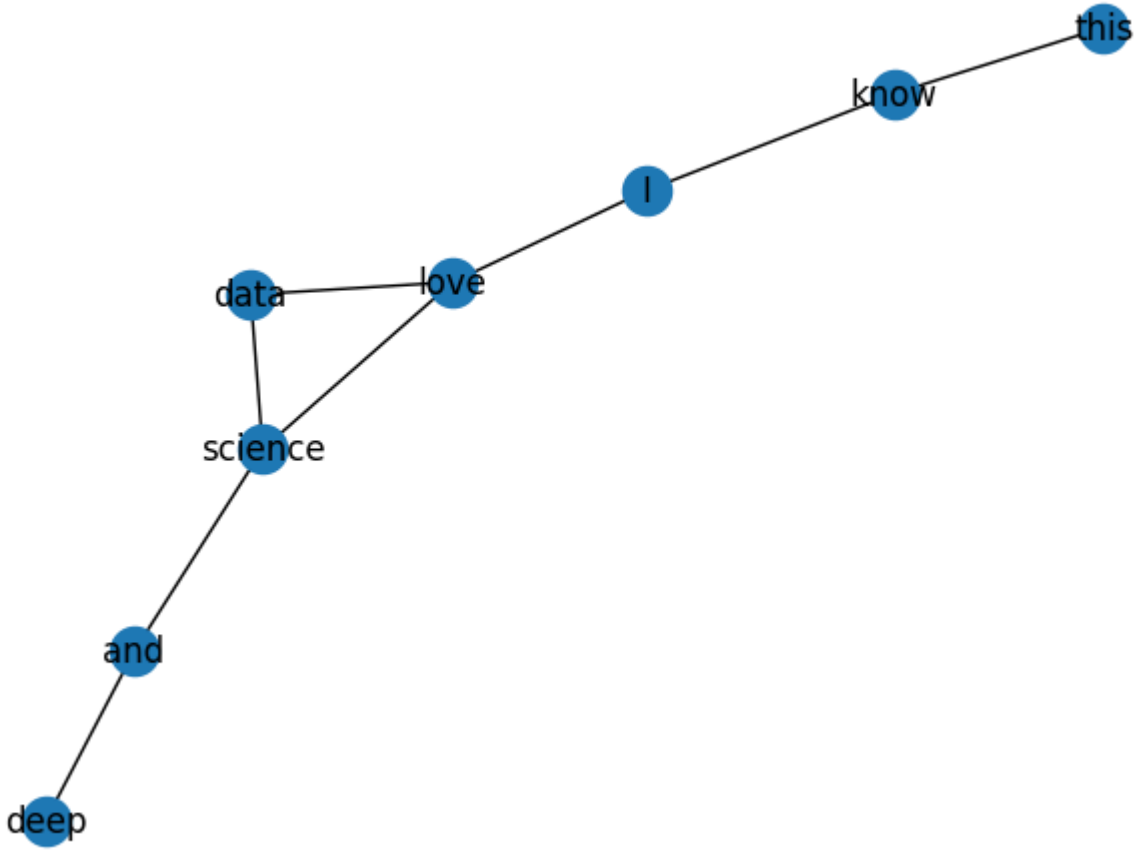
In [37]:
```
import networkx as nx
G= nx.from_pandas_adjacency(df)
print(G.nodes())
print(G.edges())
```

```
['I', 'love', 'data', 'science', 'and', 'deep', 'know', 'this']
[('I', 'love'), ('I', 'know'), ('love', 'data'), ('love', 'science'), ('data', 'science'), ('science', 'and'), ('and', 'deep'),
('know', 'this')]
```

In [38]:
```
print(G.edges()[('I','love')])
print(G.edges()[('I','know')])
```

```
{'weight': 2}
{'weight': 1}
```

In [40]:
```
nx.draw(G,with_labels=True)
```



In [41]:
```
from nltk.probability import ConditionalProbDist, MLEProbDist

cpd= ConditionalProbDist(cfd,MLEProbDist)
cpd.conditions()
```

Out[41]:
```
['I', 'love', 'data', 'science', 'and', 'deep', 'know', 'this']
```

In [44]:
```
prob_matrix=[]
for i in cpd.keys():
    prob_matrix.append([cpd[i].prob(j) for j  in cpd.keys()])
```

In [45]:
```
prob_matrix
```

Out[45]:
```
[[0.0, 0.6666666666666666, 0.0, 0.0, 0.0, 0.0, 0.3333333333333333, 0.0],
 [0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
```

In [47]:
```
df = pd.DataFrame(prob_matrix,index=cpd.keys(),columns=cpd.keys())
df.style.background_gradient(cmap='coolwarm')
```

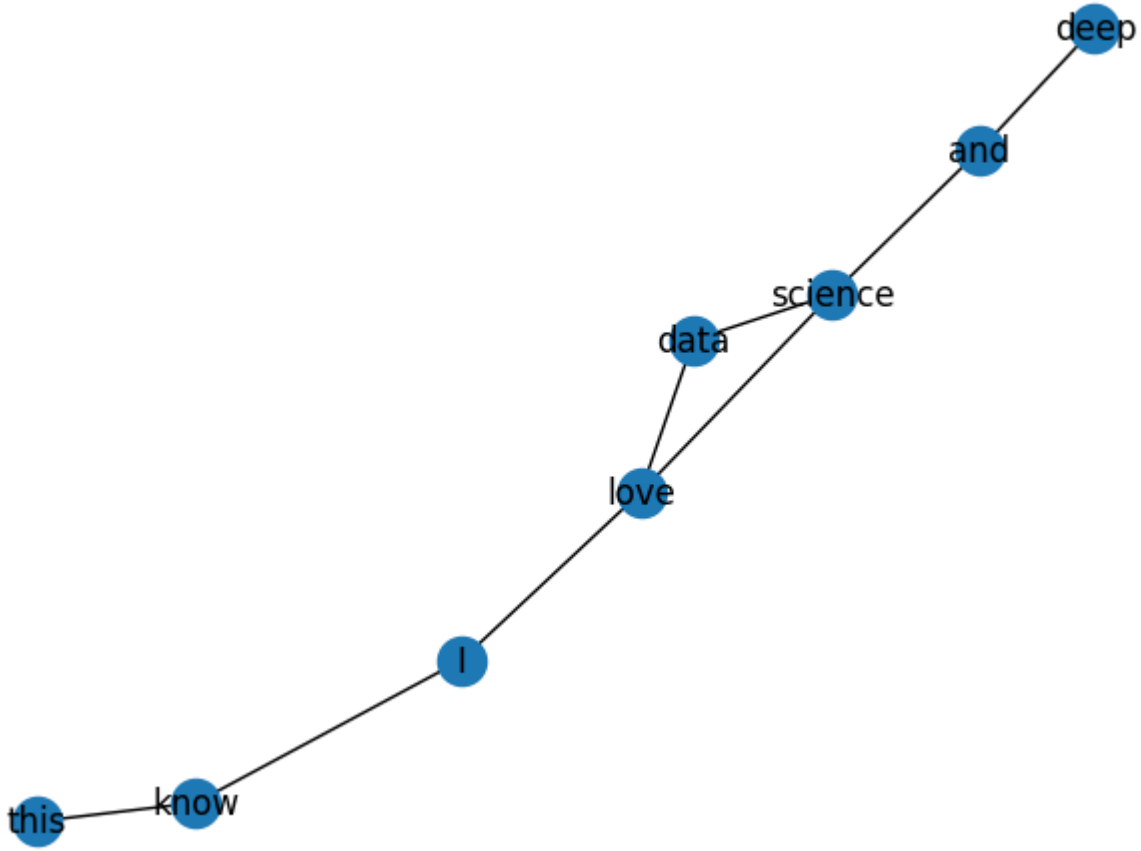
Out[47]:

	I	love	data	science	and	deep	know	this
I	0.000000	0.666667	0.000000	0.000000	0.000000	0.000000	0.333333	0.000000
love	0.000000	0.000000	0.500000	0.500000	0.000000	0.000000	0.000000	0.000000
data	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
science	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
and	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
deep	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
know	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
this	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

In [49]:
```
prob_G=nx.from_pandas_adjacency(df)

print(prob_G.edges()['I','love'])
```

```
{'weight': 0.6666666666666666}
```

In [50]:
```
nx.draw(prob_G,with_labels=True)
```



중심성(Centrality) 지수

- 연결망 분석에서 가장 많이 주목하는 속성은 바로 중심성 지수
- 중심성은 전체 연결망에서 중심에 위치하는 정도를 표현하는 지표로, 이를 분석하면 연결 정도, 중요도 등을 알 수 있음
- 중심성 지수는 나타내는 특징에 따라 연결 중심성, 매개 중심성, 근접 중심성, 위세 중심성으로 구분

연결 중심성(Degree Centrality)

- 연결 중심성은 가장 기본적이고 직관적으로 중심섬을 측정하는 지표
- 텍스트에서 다른 단어와의 동시 출현 빈도가 많은 특정 단어는 연결 중심성이 높다고 볼 수 있음
- 연결 정도로만 측정하면 연결망의 크기에 따라 달라져 비교가 어렵기 때문에 여러 방법으로 표준화
- 주로 (특정 노드 i와 직접적으로 연결된 노드 수 / 노드 i와 직간접적으로 연결된 노드 수)로 계산
- 여기서 직접적으로 연결된 노드는 서로 엣지 관계인 노드를 뜻하며, 간접적으로 연결된 노드는 서로 엣지 관계는 아니나 다른 노드와 엣지에 의해 도달할 수 있는 노드를 말함

연결 중심성 계산 수식

$$degree_{ik} = \sum_{i=1}^{N} Z_{ijk} = Z_{jk}$$

$$outdegree_{ik} = \sum_{j=1}^{N} Z_{ijk} = Z_{ik}$$

$$C_i = \sum_{j=1}^{n} (Z_{ij} + Z_{ji}) / \sum_{i=1}^{n} \sum_{j=1}^{n} (Z_{ij}) \qquad 단, 0 \leq C \leq 1$$

In [51]:
```
nx.degree_centrality(G)
```

Out[51]:
```
{'I': 0.2857142857142857,
 'love': 0.42857142857142855,
 'data': 0.2857142857142857,
 'science': 0.42857142857142855,
 'and': 0.2857142857142857,
 'deep': 0.14285714285714285,
 'know': 0.2857142857142857,
 'this': 0.14285714285714285}
```

위세 중심성(Eigenvector Centraility)

- 위세 중심성은 연결된 상대 단어의 중요성에 가중치를 둠
- 중요한 단어와 많이 연결됐다면 위세 중심성은 높아지게 됨
- 위세 중심성은 고유 벡터로써 인접해 있는 노드의 위세 점수와 관련되어 있어 직접 계산하기는 쉽지 않음

- 위세 중심성 계산 수식

$$P_i = \sum_{j=1}^{N-1} P_i Z_{ji}, \qquad 0 \le P_i \le 1$$

- 위세 중심성 계산에는 `eigenvector_centraility`를 이용해 계산
- `weight`로는 어휘 동시 출현 빈도를 이용

In [53]:
```
nx.eigenvector_centrality(G,weight='weight')
```

Out[53]:
```
{'I': 0.5055042648573065,
 'love': 0.6195557831651917,
 'data': 0.35703593885196566,
 'science': 0.39841035839294925,
 'and': 0.15933837227495717,
 'deep': 0.055886131430398216,
 'know': 0.20216573350291445,
 'this': 0.0709058113463014}
```

근접 중심성(Closeness Centrality)

- 근접 중심성은 한 단어가 다른 단어에 얼마나 가깝게 있는지를 측정하는 지표
- 직접적으로 연결된 노드만 측정하는 연결 중심성과는 다르게, 근접 중심성은 직간접적으로 연결된 모든 노드들 사이의 거리를 측정
- 근접 중심성을 측정하기 위해선 다음과 같이 계산

  (모든 노드 수 - 1 / 특정 노드 i에서 모든 노드에 이르는 최단 경로 수들 모두 더한 수)

$$C_C(A) = \frac{1}{\frac{1}{N-1}\sum_{x \neq A} l_{X,A}} = \frac{N-1}{\sum_{x \neq A} l_{X,A}}$$

- 근접 중심성을 계산하기 위해선 `closeness_centrality()` 함수를 사용

In [54]:
```
nx.closeness_centrality(G,distance='weight')
```

Out[54]:
```
{'I': 0.35,
 'love': 0.4375,
 'data': 0.3684210526315789,
 'science': 0.4117647058823529,
 'and': 0.3333333333333333,
 'deep': 0.25925925925925924,
 'know': 0.2916666666666667,
 'this': 0.23333333333333334}
```

매개 중심성(Betweeness Centrality)

- 매개 중심성은 한 단어가 단어들과의 연결망을 구축하는데 얼마나 도움을 주는지 측정하는 지표
- 매개 중심성이 높은 단어는 빈도 수가 작더라도 단어 간 의미부여 역할이 크기 때문에, 해당 단어를 제거하면 의사소통이 어려워짐
- 매개 중심성은 모든 노드 간 최단 경로에서 특정 노드가 등장하는 횟수로 측정하며,
  표준화를 위해 최댓값인 (N-1) x (N-2) / 2로 나눔

- 매개 중심성 계산 수식

$$C_B'(P_m) = \frac{\sum_i^N \sum_j^N \frac{g_{imj}}{g_{ij}}}{(\frac{N^2-3N+2}{2})}, \qquad 단, i < j, \quad i \neq j$$

- 매개 중심성을 계산하기 위해선 `current_flow_betweenness_centrality()` 함수를 이용

In [55]:
```
nx.current_flow_betweenness_centrality(G)
```

Out[55]:
```
{'I': 0.4761904761904764,
 'love': 0.6190476190476187,
 'data': 0.19047619047619005,
 'science': 0.5396825396825393,
 'and': 0.28571428571428564,
 'deep': 2.1147105230955362e-16,
 'know': 0.28571428571428614,
 'this': 8.458842092382145e-17}
```

페이지랭크(PageRank)

- 웰드 와이드 웹과 같은 하이퍼링크 구조를 가지는 문서에 상대적 중요도에 따라 가중치를 부여하는 방법
- 이 알고리즘은 서로간에 인용과 참조로 연결된 임의의 묶음에 적용 가능
- 페이지 랭크는 더 중요한 페이지는 더 많은 다른 사이트로부터 링크를 받는다는 관찰에 기초

In [56]:
```
nx.pagerank(G)
```

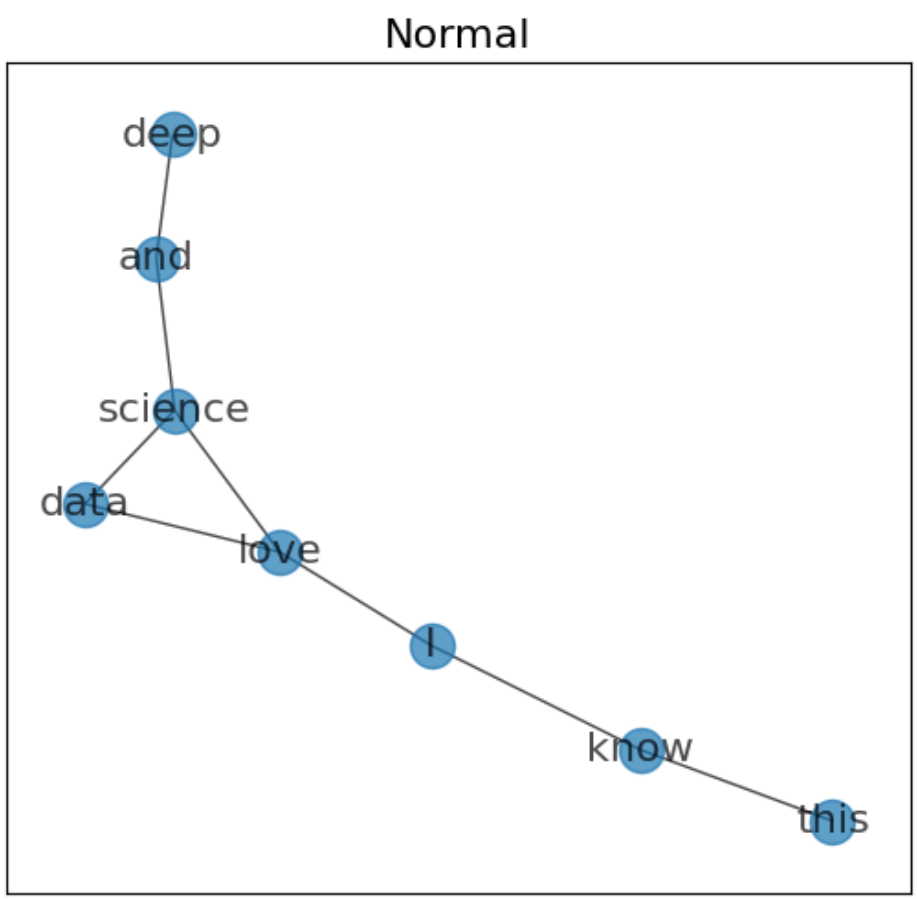
Out[56]:
```
{'I': 0.1536831077679558,
 'love': 0.19501225218917406,
 'data': 0.10481873412175656,
 'science': 0.15751225722745082,
 'and': 0.12417333539164832,
 'deep': 0.07152392879557615,
 'know': 0.1224741813421488,
 'this': 0.07080220316428934}
```

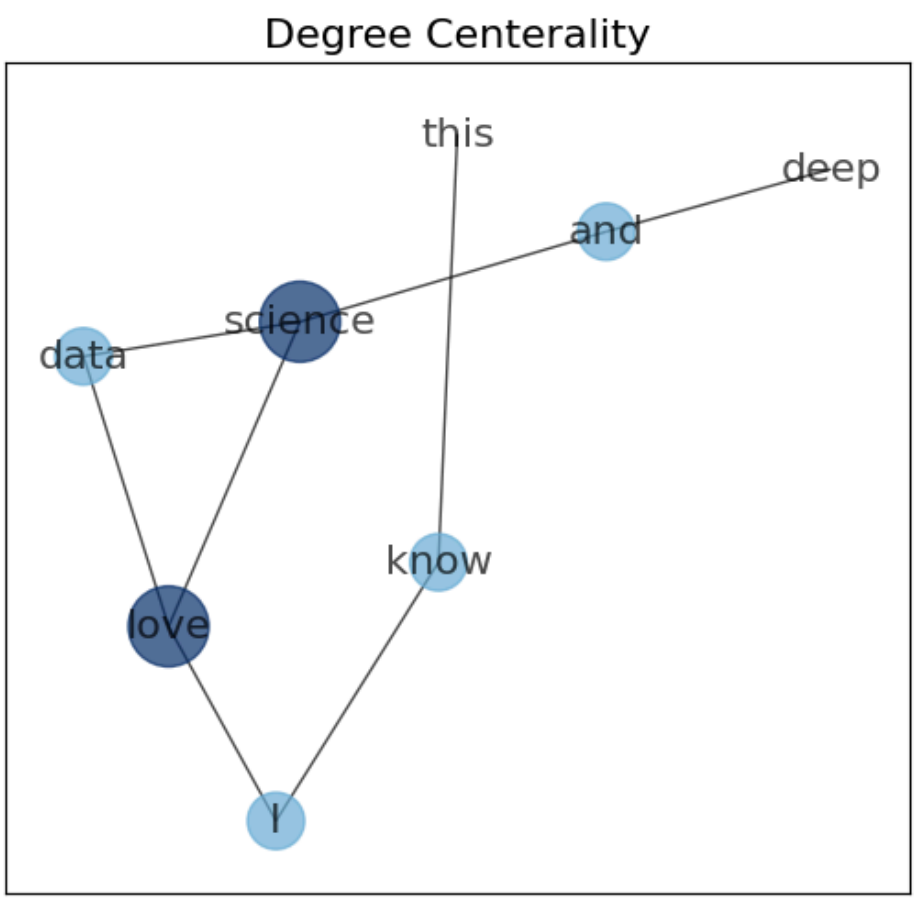
In [57]:
```
def get_node_size(node_values):
    nsize=np.array([v for v in node_values])
    nsize = 1000*(nsize-min(nsize))/ (max(nsize)-min(nsize))
    #min-max norm
    return nsize
```

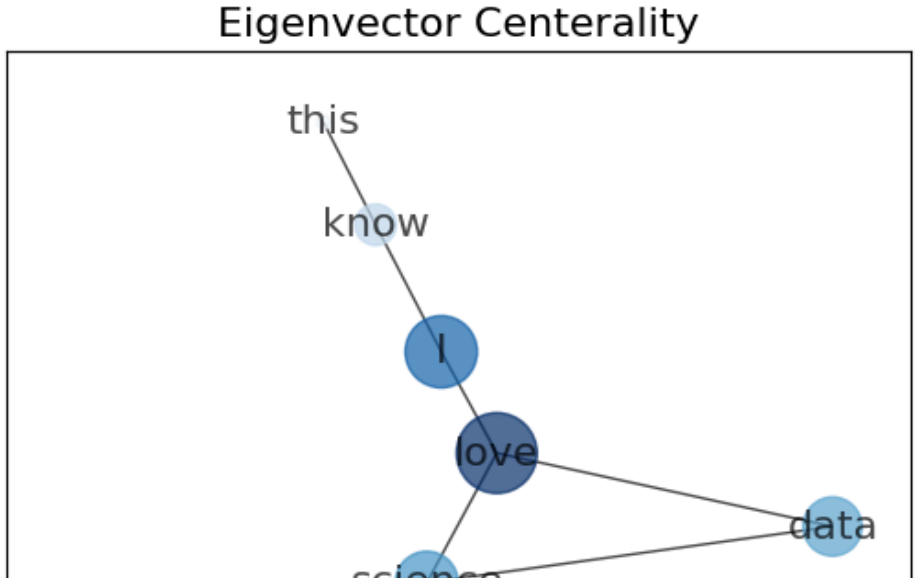
In [59]:
```
import matplotlib.pyplot as plt
```

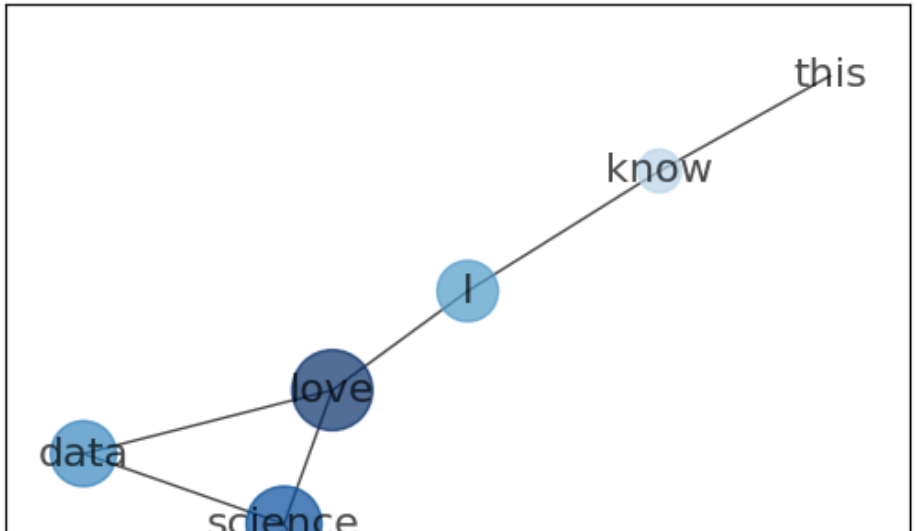
In [62]:
```
dc=nx.degree_centrality(G).values()
ec=nx.eigenvector_centrality(G,weight='weight').values()
cc=nx.closeness_centrality(G,distance='weight').values()
bc=nx.betweenness_centrality(G).values()
pr =nx.pagerank(G).values()

plt.figure(figsize=(14,20))
plt.axis('off')

plt.subplot(321)
plt.title('Normal',fontsize=16)
nx.draw_networkx(G,font_size=16, alpha=0.7,cmap=plt.cm.Blues)

plt.subplot(322)
plt.title('Degree Centerality',fontsize=16)
```

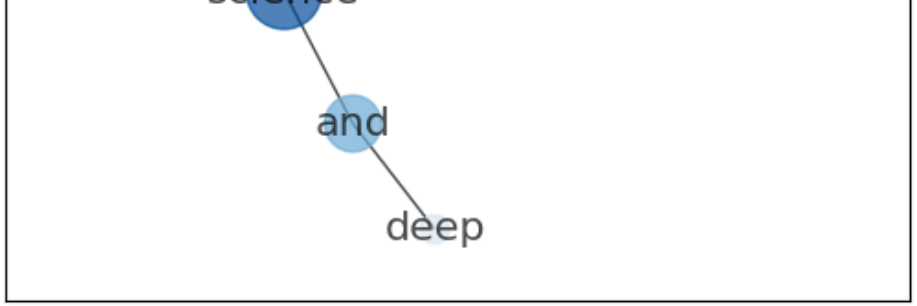
```
nx.draw_networkx(G,font_size=16,
                 node_color=list(dc), node_size=get_node_size(dc), alpha=0.7,cmap=plt.cm.Blues)
plt.subplot(323)
plt.title('Eigenvector Centerality',fontsize=16)
nx.draw_networkx(G,font_size=16,
                 node_color=list(ec), node_size=get_node_size(ec), alpha=0.7,cmap=plt.cm.Blues)
plt.subplot(324)
plt.title('Closeness Centerality',fontsize=16)
nx.draw_networkx(G,font_size=16,
                 node_color=list(cc), node_size=get_node_size(cc), alpha=0.7,cmap=plt.cm.Blues)
plt.subplot(325)
plt.title('betweenness Centerality',fontsize=16)
nx.draw_networkx(G,font_size=16,
                 node_color=list(bc), node_size=get_node_size(bc), alpha=0.7,cmap=plt.cm.Blues)
plt.subplot(326)
plt.title('pageRank Centerality',fontsize=16)
nx.draw_networkx(G,font_size=16,
                 node_color=list(pr), node_size=get_node_size(pr), alpha=0.7,cmap=plt.cm.Blues)
```


Normal
deep
and
science
data
love
I
know
this

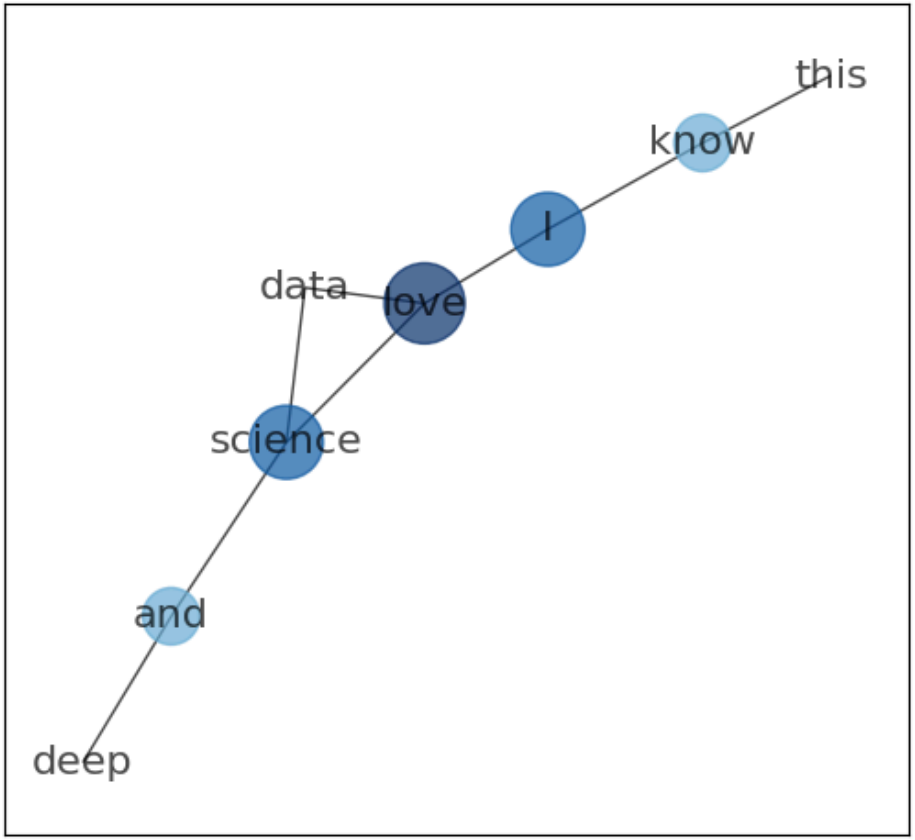

Degree Centerality
this
deep
and
science
data
know
love
I


Eigenvector Centerality
this
know
I
love
data

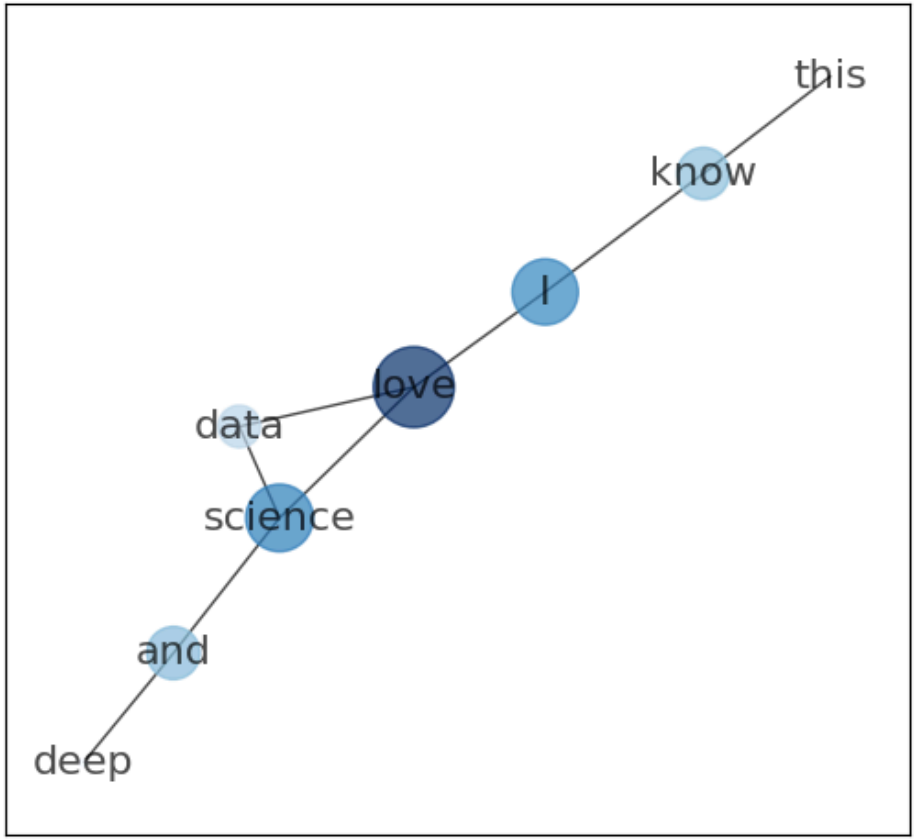

Closeness Centerality
this
know
I
love
data



betweenness Centerality



pageRank Centerality



```
pl = nx.planar_layout(G)
frl = nx.fruchterman_reingold_layout(G)
sl = nx.spectral_layout(G)
rl = nx.random_layout(G)
sl = nx.shell_layout(G)
bl = nx.bipartite_layout(G, G.nodes())
cl = nx.circular_layout(G)
sl = nx.spring_layout(G)
kkl = nx.kamada_kawai_layout(G)
```

In [ ]: