In [1]:
```
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer

import numpy as np
import pandas as pd
import nltk
import re

from nltk.corpus import stopwords
stop_words = stopwords.words('english')
```

In [2]:
```
# Extract word vectors
word_embeddings = {}
f = open('C:\\Users\\yahoo\\Desktop\\glove.6B.100d.txt', encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    word_embeddings[word] = coefs
f.close()
```

# Method 1: frequency-driven approach

There are four steps in total:

1.Get the frequency of words

2.Get the score of sentence

3.Find the threshold according to the average score

4.Use the threshold to find summary sentence

# Method the frequency of words

In [3]:
```
#We use word tokenize to get the words in texts file.
#Then use word stemmer to normalize each words
#The dict will store appearing times of each words, which works like hash table
#The key is word, and the value is the appearing times

def frequency_table(text)->dict:
    stopword = set(stopwords.words("english"))
    words = word_tokenize(text)
    W = [word.lower() for word in words]
    result = dict()
    for word in W:
        if word in stopword:  #we ignore the stopword in our case
            continue
        if word in result:    #If the word already existing in dict, appearing time +1
            result[word] += 1
        else:
            result[word] = 1
    return result
```

# Method for find the score of sentence

In [4]:
```
def score_sentences(sentences, freqTable) -> dict:
    sentenceValue = dict()

    for sentence in sentences:
        sentence_length = (len(word_tokenize(sentence)))              #sentence_length record the length of a ser

        for wordValue in freqTable:
            if wordValue in sentence.lower():
                if sentence[:10] in sentenceValue:
                    sentenceValue[sentence[:10]] += freqTable[wordValue]
                else:
                    sentenceValue[sentence[:10]] = freqTable[wordValue]

        #we use the formular p(w) = c(w)/N
        #which p(w) is computed from input, c(w) is the occurance of words, N is the length of input
        sentenceValue[sentence[:10]] = sentenceValue[sentence[:10]] // sentence_length

    return sentenceValue
```

# Method for find the threshold

In [5]:
```
def find_average_score(sentenceValue) -> int:
    sumValues = 0
    for value in sentenceValue:
        sumValues += sentenceValue[value]

    # Average value of a sentence from original text
    average = int(sumValues / len(sentenceValue))

    return average
```

# Method for generate summary

In [6]:
```
def generate_summary(sentences, sentenceValue, threshold):
    summary = ""

    for sentence in sentences:
        if sentence[:10] in sentenceValue and sentenceValue[sentence[:10]] > (threshold):
            summary += "\n" + sentence

    return summary
```

In [7]:
```
# function to remove stopwords
def remove_stopwords(sen):
    sen_new = " ".join([i for i in sen if i not in stop_words])
    return sen_new
```

# Now we start to generate the summary

In [8]:
```
f = open("C:\\Users\\yahoo\\Desktop\\textfile.txt",encoding="utf-8")
text = (f.read())
f.close()
```

In [9]:
```
sentences = sent_tokenize(text)
sentences[:5]
```

Out[9]:
```
['Mental Models: How to Train Your Brain to Think in New Ways?',
 'You can train your brain to think better.',
 'One of the best ways to do this is to expand the set of mental models you use to think.',
 'Let me explain what I mean by sharing a story about a world-class thinker.',
 'I first discovered what a mental model was and how useful the right one could be while I was reading a story
about Richard Feynman, the famous physicist.']
```

# Step1:get the frequency of words

In [10]:
```
freq_table = frequency_table(text)
print(freq_table)
```

{'mental': 41, 'models': 31, ':': 3, 'train': 2, 'brain': 3, 'think': 4, 'new': 6, 'ways': 2, '?': 3, 'better': 4, '.': 107, 'one': 12, 'best': 4, 'expand': 1, 'set': 7, 'use': 3, 'let': 1, 'explain': 4, 'mean': 1, 'sharing': 1, 'story': 2, 'world-class': 3, 'thinker': 2, 'first': 1, 'discovered': 1, 'model': 11, 'useful': 4, 'right': 1, 'could': 3, 'reading': 2, 'richard': 2, 'feynman': 10, ',': 91, 'famous': 1, 'physicist': 1, 'received': 1, 'undergraduate': 1, 'degree': 1, 'mit': 3, 'ph.d.': 3, 'princeton': 3, 'time': 1, 'developed': 1, 'reputation': 2, 'waltzing': 1, 'math': 2, 'department': 2, 'solving': 1, 'problems': 2, 'brilliant': 2, 'students': 1, "n't": 5, 'solve': 2, 'people': 3, 'asked': 2, 'claimed': 1, 'secret': 3, 'weapon': 1, 'intelligence': 2, 'rather': 1, 'strategy': 1, 'learned': 2, 'high': 2, 'school': 4, 'according': 1, 'physics': 4, 'teacher': 2, 'stay': 1, 'class': 2, 'day': 2, 'gave': 1, 'challenge': 1, '```': 15, "''": 15, 'said': 1, 'talk': 2, 'much': 4, 'make': 3, 'noise': 1, 'know': 2, "'re": 4, 'bored': 1, "'m": 1, 'going': 1, 'give': 1, 'book': 5, 'go': 1, 'back': 2, 'corner': 1, 'study': 2, 'everything': 2, "'s": 8, 'would': 3, 'hide': 1, 'classroom': 1, 'book—advanced': 1, 'calculus': 2, 'woods—while': 1, 'rest': 1, 'continued': 1, 'regular': 1, 'lessons': 1, 'studying': 2, 'old': 1, 'textbook': 1, 'began': 1, 'develop': 5, 'showed': 1, 'differentiate': 1, 'parameters': 1, 'integral': 3, 'sign': 2, 'wrote': 1, 'turns': 1, 'taught': 1, 'universities': 1, ';': 2, 'emphasize': 1, 'caught': 1, 'method': 1, 'used': 1, 'damn': 1, 'tool': 1, 'self-taught': 1, 'using': 1, 'peculiar': 1, 'methods': 2, 'integrals': 2, 'result': 1, 'guys': 1, 'trouble': 1, 'certain': 4, 'standard': 1, 'contour': 1, 'integration': 1, 'found': 3, 'simple': 1, 'series': 1, 'expansion': 1, 'come': 1, 'along': 1, 'try': 3, 'differentiating': 1, 'often': 4, 'worked': 1, 'got': 1, 'great': 3, 'box': 1, 'tools': 5, 'different': 5, 'everybody': 1, 'else': 1, 'tried': 1, 'giving': 1, 'problem': 7, 'every': 6, 'student': 1, 'separated': 1, 'peers': 1, 'necessarily': 1, 'raw': 1, 'way': 3, 'saw': 2, 'broader': 1, 'explanation': 3, 'something': 4, 'works': 6, 'concept': 1, 'framework': 1, 'worldview': 5, 'carry': 4, 'around': 1, 'mind': 2, 'help': 2, 'interpret': 1, 'world': 7, 'understand': 6, 'relationship': 1, 'things': 1, 'deeply': 1, 'held': 1, 'beliefs': 1, 'example': 4, 'supply': 1, 'demand': 1, 'helps': 3, 'economy': 1, 'game': 1, 'theory': 2, 'relationships': 1, 'trust': 1, 'work': 3, 'entropy': 2, 'disorder': 1, 'decay': 1, 'guide': 1, 'perception': 1, 'behavior': 1, 'thinking': 9, 'life': 8, 'decisions': 2, 'learning': 3, 'gives': 1, 'see': 3, 'world—like': 1, 'technique': 1, 'imperfect': 1, 'single': 2, 'engineering': 1, 'provides': 2, 'flawless': 1, 'entire': 2, 'universe': 1, 'disciplines': 3, 'allowed': 1, 'us': 1, 'build': 2, 'bridges': 1, 'roads': 1, 'technologies': 1, 'even': 1, 'travel': 1, 'outer': 1, 'space': 1, 'historian': 1, 'yuval': 1, 'noah': 1, 'harari': 1, 'puts': 1, 'scientists': 1, 'generally': 1, 'agree': 1, '100': 1, 'percent': 2, 'correct': 1, 'thus': 2, 'real': 2, 'test': 1, 'knowledge': 4, 'truth': 3, 'utility': 2, 'ideas': 4, 'broadly': 1, 'daily': 2, 'understanding': 1, 'concepts': 1, 'wiser': 1, 'choices': 1, 'take': 1, 'actions': 1, 'developing': 1, 'broad': 1, 'base': 1, 'critical': 1, 'anyone': 1, 'interested': 2, 'clearly': 1, 'rationally': 1, 'effectively': 1, 'decision': 1, 'making': 2, 'expanding': 3, 'experts': 3, 'need': 3, 'novices': 1, 'favorite': 1, 'ones': 1, 'naturally': 1, 'default': 1, 'happened': 1, 'grow': 1, 'older': 1, 'expertise': 2, 'area': 2, 'tend': 2, 'favor': 1, 'familiar': 1, 'dominates': 2, "'ll": 4, 'face': 3, 'pitfall': 1, 'particularly': 1, 'easy': 2, 'slip': 1, 'smart': 1, 'talented': 1, 'given': 1, 'master': 3, 'likely': 1, 'becomes': 2, 'downfall': 1, 'start': 1, 'applying': 1, 'indiscriminately': 1, 'looks': 2, 'like': 7, 'limitation': 1, 'common': 1, 'proverb': 1, 'says': 1, 'hammer': 1, 'nail': 1, 'consider': 2, 'biologist': 2, 'robert': 1, 'sapolsky': 1, 'asks': 1, 'chicken': 5, 'cross': 1, 'road': 4, 'answers': 1, 'ask': 3, 'evolutionary': 1, 'might': 3, 'say': 3, 'crossed': 3, 'potential': 3, 'mate': 1, 'side': 1, 'kinesiologist': 1, 'muscles': 1, 'leg': 2, 'contracted': 1, 'pulled': 1, 'bone': 1, 'forward': 1, 'step': 1, 'neuroscientist': 1, 'neurons': 1, 'fired': 1, 'triggered': 1, 'movement': 1, 'technically': 1, 'speaking': 1, 'none': 2, 'wrong': 1, 'nobody': 1, 'seeing': 1, 'picture': 6, 'either': 1, 'individual': 1, 'view': 1, 'reality': 1, 'challenges': 1, 'situations': 1, 'entirely': 1, 'explained': 1, 'field': 2, 'industry': 1, 'perspectives': 2, 'hold': 1, 'contain': 1, 'complete': 2, 'relying': 1, 'narrow': 1, 'wearing': 1, 'straitjacket': 1, 'cognitive': 1, 'range': 3, 'motion': 1, 'limited': 2, 'finding': 1, 'solution': 1, 'order': 1, 'unleash': 1, 'full': 2, 'collect': 1, 'decision-making': 1, 'toolbox': 1, 'learn': 4, 'employ': 1, 'variety': 2, 'process': 1, 'accumulating': 1, 'somewhat': 1, 'improving': 1, 'vision': 1, 'eye': 3, 'cover': 1, 'lose': 1, 'part': 1, 'scene': 1, 'impossible': 1, 'looking': 2, 'similarly': 1, 'provide': 1, 'internal': 1, 'continuously': 1, 'upgrade': 1, 'improve': 1, 'quality': 1, 'means': 1, 'widely': 1, 'books': 1, 'fundamentals': 2, 'seemingly': 1, 'unrelated': 1, 'fields': 1, 'wildly': 1, 'experiences': 1, 'needs': 1, 'piece': 1, 'together': 1, 'sources': 1, 'draw': 1, 'upon': 1, 'clearer': 1, 'philosopher': 1, 'alain': 1, 'de': 1, 'botton': 1, 'notes': 1, 'chief': 1, 'enemy': 1, 'good': 2, 'lack': 1, 'sufficient': 1, 'pursuit': 1, 'liquid': 2, 'separate': 1, 'silos—biology': 1, 'economics': 3, 'history': 2, 'philosophy': 2, 'information': 1, 'rarely': 1, 'divided': 1, 'neatly': 1, 'defined': 1, 'categories': 1, 'words': 1, 'charlie': 2, 'munger': 2, 'wisdom': 1, 'little': 2, 'academic': 1, 'thinkers': 2, 'silo-free': 1, 'avoid': 1, 'lens': 1, 'subject': 2, 'instead': 1, 'flows': 1, 'easily': 1, 'topic': 2, 'next': 1, 'important': 4, 'connect': 1, 'another': 1, 'creativity': 1, 'innovation': 1, 'arise': 1, 'intersection': 1, 'spotting': 1, 'links': 1, 'various': 1, 'identify': 1, 'solutions': 1, 'overlook': 1, 'news': 1, 'detail': 1, 'become': 1, '"': 1, 'humankind': 1, 'generated': 1, 'throughout': 1, 'dozen': 2, 'firm': 1, 'grasp': 1, 'many': 1, 'big': 2, 'biology': 1, 'chemistry': 1, 'mathematics': 1, 'psychology': 1, 'form': 1, 'backbone': 1, 'pillar': 1, 'include': 1, 'incentives': 1, 'scarcity': 1, 'economies': 1, 'scale': 1, 'discipline': 1, 'remarkably': 1, 'accurate': 1, 'quote': 1, '80': 1, '90': 2, 'freight': 3, 'worldly-wise': 1, 'person': 2, 'mere': 1, 'handful': 1, 'really': 1, 'heavy': 2, "'ve": 2, 'made': 1, 'personal': 1, 'mission': 1, 'uncover': 1, 'researching': 1, '1,000': 1, 'gradually': 1, 'narrowed': 1, 'matter': 1, 'written': 1, 'previously': 1, 'inversion': 1, 'covering': 1, 'future': 1, 'browse': 1, 'slowly': 1, 'list': 2, 'hope': 1, 'create': 1, 'wide': 1, 'also': 1, 'meaningful': 1, 'practical': 1, 'average': 1, 'luck': 1, 'bit': 1}

In [11]:
```
print(freq_table["mental"])
```

41

# Step2: Get the score of sentence

In [12]:
```
sentence_scores = score_sentences(sentences, freq_table)
print(sentence_scores)
```

{'Mental Mod': 7, 'You can tr': 13, 'One of the': 10, 'Let me exp': 8, 'I first di': 9, 'Feynman re': 10, 'During tha': 9, 'When peopl': 8, 'According ': 10, '"Feynman,"': 11, 'I know why': 27, "You're bor": 28, "So I'm goi": 11, 'You go up ': 6, 'So each da': 7, 'And it was': 10, '"That book': 13, '"It turns ': 6, 'But I caug': 10, 'So because': 12, '"The resul': 6, 'If it was ': 7, 'Then I com': 13, 'So I got a': 6, 'Every Ph.D': 12, 'What separ': 10, 'It was the': 13, 'He had a b': 22, 'What is a ': 9, 'A mental m': 16, 'It is a co': 8, 'Mental mod': 32, 'For exampl': 13, 'Game theor': 11, 'Entropy is': 11, 'They are t': 13, 'Learning a': 9, 'There is n': 6, 'As histori': 10, 'Thus, the ': 18, 'The best m': 16, 'They are b': 16, 'Understand': 9, 'This is wh': 10, 'The Secret': 14, 'Expanding ': 30, 'We all hav': 12, 'As you gro': 11, "Here's the": 7, 'This pitfa': 6, 'The more y': 8, 'What looks': 14, 'As the com': 10, 'When a cer': 12, 'Consider t': 14, 'He asks, "': 8, 'Then, he p': 23, 'If you ask': 8, 'Technicall': 21, 'But nobody': 13, 'Each indiv': 16, 'The challe': 7, 'All perspe': 19, 'None of th': 16, 'Relying on': 12, 'Your cogni': 14, 'When your ': 16, 'In order t': 16, 'You have t': 13, 'The proces': 15, 'Each eye c': 13, 'But if you': 14, "It's impos": 8, 'Similarly,': 21, 'We should ': 9, 'This means': 10, "The mind's": 10, 'The more s': 14, 'As the phi': 8, 'The Pursui': 19, 'In school,': 11, 'In the rea': 15, 'In the wor': 8, 'World-clas': 18, 'They avoid': 11, 'Instead, t': 12, 'Creativity': 10, 'By spottin': 16, 'Tools for ': 26, 'Of all the': 9, 'Many of th': 11, 'Each field': 14, 'If you can': 11, 'To quote C': 10, 'And, of th': 13, "I've made ": 8, 'After rese': 13, "I've writt": 8, "If you're ": 18, 'My hope is': 5, 'With any l': 12}

# Step3: Get the threshold

In [13]:
```
threshold = find_average_score(sentence_scores)
print(threshold)
```

12

# Step4: Get the summary

In [14]:
```
summary1 = generate_summary(sentences, sentence_scores, 1.2*threshold)
print(summary1)
```

I know why.
You're bored.
He had a broader set of mental models.
A mental model is an explanation of how something works.
Mental models are deeply held beliefs about how the world works.
Mental models guide your perception and behavior.
Mental models are imperfect but useful.
Thus, the real test of knowledge is not the truth, but utility."
The best mental models are the ideas with the most utility.
They are broadly useful in daily life.
Expanding your set of mental models is something experts need to work on just as much as novices.
Then, he provides answers from different experts.
Technically speaking, none of these experts are wrong.
Each individual mental model is just one view of reality.
All perspectives hold some truth.
None of them contain the complete truth.
When your set of mental models is limited, so is your potential for finding a solution.
In order to unleash your full potential, you have to collect a range of mental models.
Thus, the secret to great thinking is to learn and employ a variety of mental models.
Expanding Your Set of Mental Models.
The process of accumulating mental models is somewhat like improving your vision.
Similarly, mental models provide an internal picture of how the world works.
The Pursuit of Liquid Knowledge.
In the real world, information is rarely divided into neatly defined categories.
World-class thinkers are often silo-free thinkers.
By spotting the links between various mental models, you can identify solutions that most people overlook.
Tools for Thinking Better.
If you're interested, you can browse my slowly expanding list of mental models.

# Method 2: Latent Semantic Analysis with TextRanking algorithm

This method works different from frequcey driven method, but it also have four steps:

1.Clean the sentences,including punctuations, numbers, stopwords and special characters

2.Create vector for each sentences

3.Use consine similarity to find the similarities between each sentences. Bulid the similarity matrix

4.Apply textRank algorithm

# Step 1: Clean the sentence

In [15]:
```
# function to remove stopwords
def remove_stopwords(sen):
    sen_new = " ".join([i for i in sen if i not in stop_words])
    return sen_new
```

In [16]:
```
# remove punctuations, numbers and special characters
clean_sentences = pd.Series(sentences).str.replace("[^a-zA-Z]", " ")

# make alphabets lowercase
clean_sentences = [s.lower() for s in clean_sentences]
```

In [17]:
```
# remove stopwords from the sentences
clean_sentences = [remove_stopwords(r.split()) for r in clean_sentences]
```

# Step 2: Create vector for each sentence

In [18]:
```
#We will first take the vectors (each of size 100 elements) for the constituent words in the sentence
#And then take the average/average of these vectors to get the combined vector of the sentence.

sentence_vectors = []
for i in clean_sentences:
    if len(i) != 0:
        v = sum([word_embeddings.get(w, np.zeros((100,))) for w in i.split()])/(len(i.split())+0.001)
    else:
        v = np.zeros((100,))
    sentence_vectors.append(v)
```

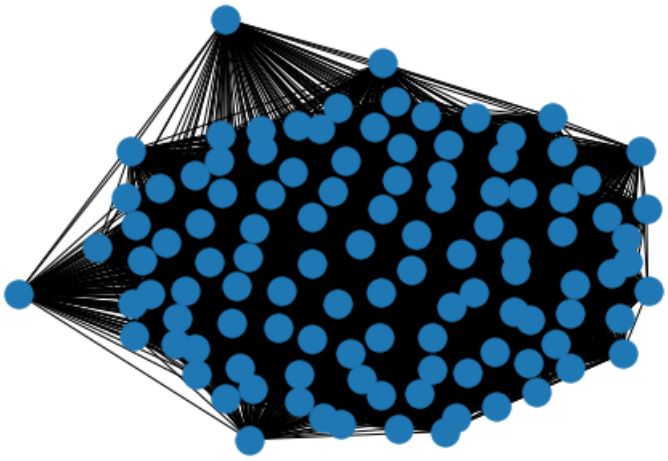
# Step 3: Use consine similarity to build similarity matrix

In [19]:
```
# similarity matrix
sim_mat = np.zeros([len(sentences), len(sentences)])
```

In [20]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

In [21]:
```
for i in range(len(sentences)):
    for j in range(len(sentences)):
        if i != j:
            sim_mat[i][j] = cosine_similarity(sentence_vectors[i].reshape(1,100), sentence_vectors[j].reshape(1,1
```

In [22]:
```
import networkx as nx

nx_graph = nx.from_numpy_array(sim_mat)
scores = nx.pagerank(nx_graph)

nx.draw(nx_graph)
```



# Step 4: Rank the sentense using textRank algorithm

In [23]:
```
ranked_sentences = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True)
```

In [24]:
```
# Extract top 10 sentences as the summary
for i in range(10):
    print(ranked_sentences[i][1])
```

```
My hope is to create a list of the most important mental models from a wide range of disciplines and explain t
hem in a way that is not only easy to understand but also meaningful and practical to the daily life of the av
erage person.
One of the best ways to do this is to expand the set of mental models you use to think.
This is why it is important to not only learn new mental models but to consider how they connect with one anot
her.
Expanding your set of mental models is something experts need to work on just as much as novices.
Thus, the secret to great thinking is to learn and employ a variety of mental models.
Of all the mental models' humankind has generated throughout history, there are just a few dozen that you need
to learn to have a firm grasp of how the world works.
The mind's eye needs a variety of mental models to piece together a complete picture of how the world works.
Learning a new mental model gives you a new way to see the world—like Richard Feynman learning a new math tec
hnique.
They are the thinking tools that you use to understand life, make decisions, and solve problems.
The challenges and situations we face in life cannot be entirely explained by one field or industry.
```

# More sample for method 1

In [25]:
```
f = open("C:\\Users\\yahoo\\Desktop\\textfile2.txt",encoding="utf-8")
text = (f.read())
f.close()

sentences = sent_tokenize(text)
#get frequecy table
freq_table = frequency_table(text)
#get sentence score
sentence_scores = score_sentences(sentences, freq_table)
#find threshold
threshold = find_average_score(sentence_scores)
#get summary
summary2 = generate_summary(sentences, sentence_scores, 1.2*threshold)
print(summary2)
```

```
First Principles: Elon Musk on the Power of Thinking for Yourself.
First principles thinking, which is sometimes called reasoning from first principles, is one of the most effec
tive strategies you can employ for breaking down complicated problems and generating original solutions.
Given the high price, he began to rethink the problem.
So I said, okay, let's look at the first principles.
SpaceX was born.
First principles thinking is the act of boiling a process down to the fundamental parts that you know are true
and building up from there.
Defining First Principles Thinking.
A military tank.
A bicycle.
Tank: metal treads, steel armor plates, and a gun.
Bicycle: handlebars, wheels, gears, and a seat.
This is the process of first principles thinking in a nutshell.
Deconstruct then reconstruct.
How First Principles Drive Innovation.
First principles thinking helps you to cobble together information from different disciplines to create new id
eas and innovations.
The Challenge of Reasoning From First Principles.
First principles thinking can be easy to describe, but quite difficult to practice.
Nylon backpacks were first sold in 1967.
How to Think for Yourself.
The human tendency for imitation is a common roadblock to first principles thinking.
They're called airplanes.
Optimize the function.
Ignore the form.
The Power of First Principles.
First principles thinking does not remove the need for continuous improvement, but it does alter the direction
of improvement.
First principles thinking sets you on a different trajectory.
```

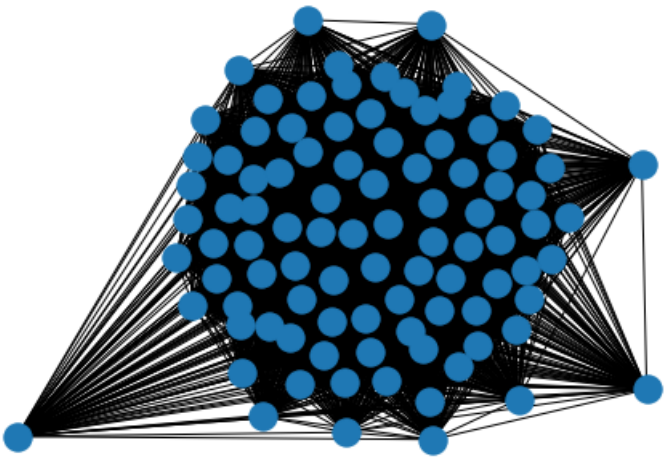
In [26]:
```
f = open("C:\\Users\\yahoo\\Desktop\\textfile3.txt",encoding="utf-8")
text = (f.read())
f.close()

sentences = sent_tokenize(text)
#get frequecy table
freq_table = frequency_table(text)
#get sentence score
sentence_scores = score_sentences(sentences, freq_table)
#find threshold
threshold = find_average_score(sentence_scores)
#get summary
summary3 = generate_summary(sentences, sentence_scores, 1.2*threshold)
print(summary3)
```

```
Entropy: Why Life Always Seems to Get More Complicated.
Murphy's Law states, "Anything that can go wrong, will go wrong."
This pithy statement references the annoying tendency of life to cause trouble and make things difficult.
Murphy's Law is just a common adage that people toss around in conversation, but it is related to one of the g
reat forces of our universe.
entropy.
But in reality, that never happens.
Quite simply, because the odds are overwhelmingly against it.
Mathematically speaking, an orderly outcome is incredibly unlikely to happen at random.
But in practice, it never happens.
Entropy is a measure of disorder.
Left to its own devices, life will always become less structured.
Sand castles get washed away.
Weeds overtake gardens.
Ancient ruins crumble.
Cars begin to rust.
People gradually age.
The pull of entropy is relentless.
Everything decays.
Disorder always increases.
Without Effort, Life Tends to Lose Order.
Before you get depressed, there is good news.
You can solve a scattered puzzle.
Successful relationships require care and attention.
Successful houses require cleaning and maintenance.
Successful teams require communication and collaboration.
Without effort, things will decay.
Order requires effort.
Entropy in Daily Life.
For example:
Why Life is Remarkable.
Consider the human body.
Mathematically speaking, the odds are overwhelmingly against your very presence.
And yet, here you are.
It is truly remarkable.
Why Art is Beautiful.
Why Marriage is Difficult.
Optimal lives are designed, not discovered.
Murphy's Law Applied to the Universe.
It is simply entropy at work.
It is simply a law of probability.
```

# More sample for method 2

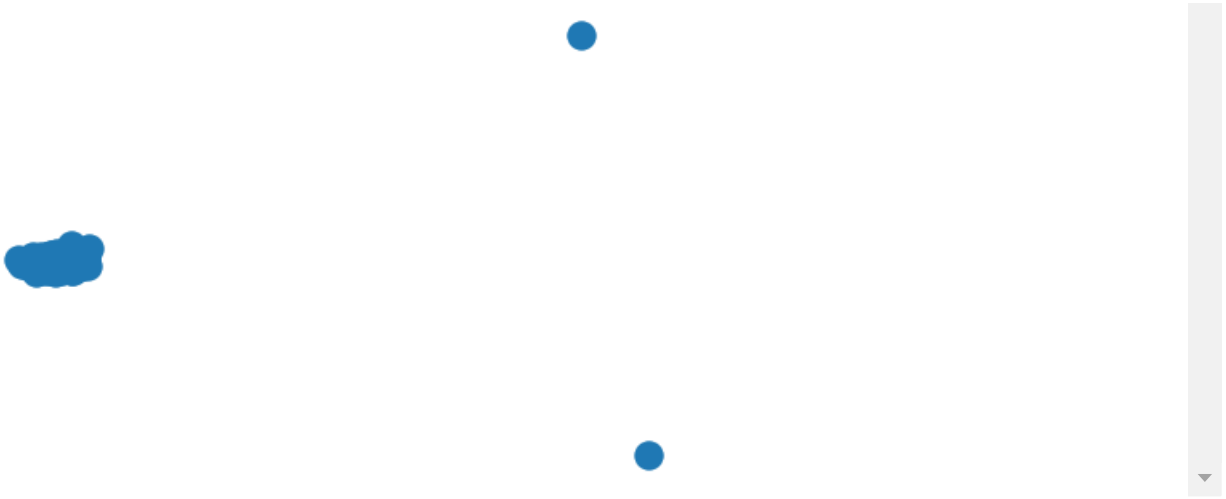
In [27]:
```
f = open("C:\\Users\\yahoo\\Desktop\\textfile2.txt",encoding="utf-8")
text = (f.read())
f.close()

sentences = sent_tokenize(text)

# remove punctuations, numbers and special characters
clean_sentences = pd.Series(sentences).str.replace("[^a-zA-Z]", " ")

# make alphabets lowercase
clean_sentences = [s.lower() for s in clean_sentences]

# remove stopwords from the sentences
clean_sentences = [remove_stopwords(r.split()) for r in clean_sentences]

sentence_vectors = []
for i in clean_sentences:
    if len(i) != 0:
        v = sum([word_embeddings.get(w, np.zeros((100,))) for w in i.split()])/(len(i.split())+0.001)
    else:
        v = np.zeros((100,))
    sentence_vectors.append(v)

# similarity matrix
sim_mat = np.zeros([len(sentences), len(sentences)])

for i in range(len(sentences)):
    for j in range(len(sentences)):
        if i != j:
            sim_mat[i][j] = cosine_similarity(sentence_vectors[i].reshape(1,100), sentence_vectors[j].reshape(1,1

nx_graph = nx.from_numpy_array(sim_mat)
scores = nx.pagerank(nx_graph)

nx.draw(nx_graph)

ranked_sentences = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True)

# Extract top 10 sentences as the summary
for i in range(10):
    print(ranked_sentences[i][1])
```

```
Over two thousand years ago, Aristotle defined a first principle as “the first basis from which a thing is kn
own.”
First principles thinking is a fancy way of saying “think like a scientist.” Scientists don’t assume anythi
ng.
Even if you aren’t trying to develop innovative ideas, understanding the first principles of your field is a s
mart use of your time.
Many of the most groundbreaking ideas in history have been a result of boiling things down to the first princi
ples and then substituting a more effective solution for one of the key parts.
Without reasoning by first principles, you spend your time making small improvements to a bicycle rather than
a snowmobile.
First principles thinking, which is sometimes called reasoning from first principles, is one of the most effec
tive strategies you can employ for breaking down complicated problems and generating original solutions.
By comparison, first principles thinking requires you to abandon your allegiance to previous forms and put the
function front and center.
Musk used first principles thinking to break the situation down to the fundamentals, bypass the high prices of
the aerospace industry, and create a more effective solution.
Rene Descartes, the French philosopher and scientist, embraced this approach with a method now called Cartesia
n Doubt in which he would “systematically doubt everything he could possibly doubt until he was left with wha
t he saw as purely indubitable truths.”
In practice, you don’t have to simplify every problem down to the atomic level to get the benefits of first pr
inciples thinking.
First principles thinking is the act of boiling a process down to the fundamental parts that you know are true
and building up from there.
It is a cycle of breaking a situation down into the core pieces and then putting them all back together in a m
ore effective way.
```



In [28]:
```
f = open("C:\\Users\\yahoo\\Desktop\\textfile3.txt",encoding="utf-8")
text = (f.read())
f.close()

sentences = sent_tokenize(text)

# remove punctuations, numbers and special characters
clean_sentences = pd.Series(sentences).str.replace("[^a-zA-Z]", " ")

# make alphabets lowercase
clean_sentences = [s.lower() for s in clean_sentences]

# remove stopwords from the sentences
clean_sentences = [remove_stopwords(r.split()) for r in clean_sentences]

sentence_vectors = []
for i in clean_sentences:
    if len(i) != 0:
        v = sum([word_embeddings.get(w, np.zeros((100,))) for w in i.split()])/(len(i.split())+0.001)
    else:
        v = np.zeros((100,))
    sentence_vectors.append(v)

# similarity matrix
sim_mat = np.zeros([len(sentences), len(sentences)])

for i in range(len(sentences)):
    for j in range(len(sentences)):
        if i != j:
            sim_mat[i][j] = cosine_similarity(sentence_vectors[i].reshape(1,100), sentence_vectors[j].reshape(1,1

nx_graph = nx.from_numpy_array(sim_mat)
scores = nx.pagerank(nx_graph)

nx.draw(nx_graph)

ranked_sentences = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True)

# Extract top 10 sentences as the summary
for i in range(10):
    print(ranked_sentences[i][1])
```

```
The collection of atoms that make up your body could be arranged in a virtually infinite number of ways and ne
arly all of them lead to no form of life whatsoever.
As one scientist put it, “Entropy is sort of like Murphy’s Law applied to the entire universe.”
It is nobody’s fault that life has problems.
At the very least, life will not be optimal—maybe you didn’t grow up in the optimal culture for your interest
s, maybe you were exposed to the wrong subject or sport, maybe you were born at the wrong time in history.
Left to its own devices, life will always become less structured.
Entropy: Why Life Always Seems to Get More Complicated.
If anything, our lives become more complicated and gradually decline into disorder rather than remaining simpl
e and structured.
Without Effort, Life Tends to Lose Order.
Murphy’s Law is just a common adage that people toss around in conversation, but it is related to one of the g
reat forces of our universe.
Given what we know about entropy, what do you think the odds are that the environment you happen to grow up in
is also the optimal environment for your talents?
Artists create a form of order and symmetry that, odds are, the universe would never generate on its own.
```



In [29]:
```
#done here
```

# Assignment4

November 28, 2020

[1]:
```
import numpy as np
import pandas as pd
import re
from nltk import download
download('stopwords')
from nltk.corpus import stopwords
import tensorflow as tf
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate,␣
 ↪TimeDistributed
import warnings
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/chenanqi/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[2]:
```
data = pd.read_csv("articles.csv")
```

[3]:
```
stop_words = set(stopwords.words('english'))

def text_cleaner(text,num):
    newString = text.lower()
    newString = re.sub(r'\([^)]*\)', '', newString)
    newString = re.sub('"','', newString)
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    newString = re.sub('[m]{2,}', 'mm', newString)
    if(num==0):
        tokens = [w for w in newString.split() if not w in stop_words]
    else:
        tokens=newString.split()
    long_words=[]
    for i in tokens:
        if len(i)>1:
            long_words.append(i)
    return (" ".join(long_words)).strip()
```

[4]:
```
cleaned_text = []
for t in data['text']:
    cleaned_text.append(text_cleaner(t,0))
```

[5]:
```
data['text'][:3]
```

[5]:
```
0    You can train your brain to think better. One of the best ways to do this
is to expand the set of mental models you use to think. Let me explain what I
mean by sharing a story about a world-class …
1    First principles thinking, which is sometimes called reasoning from first
principles, is one of the most effective strategies you can employ for breaking
down complicated problems and generating o…
2    Murphy's Law states, "Anything that can go wrong, will go wrong."\nThis
pithy statement references the annoying tendency of life to cause trouble and
make things difficult. Problems seem to arise …
Name: text, dtype: object
```

[6]:
```
cleaned_summary = []
for t in data['title']:
    cleaned_summary.append(text_cleaner(t,1))
```

[7]:
```
data['title'][:3]
```

[7]:
```
0        Mental Models: How to Train Your Brain to Think in New Ways?You can
train your brain to think better. One of the best ways to do this is to expand
the set of mental models you use to think.
1    First Principles: Elon Musk on the Power of Thinking for Yourself. First
principles thinking does not remove the need for continuous improvement, but it
does alter the direction of improvement.
2
Entropy: Why Life Always Seems to Get More Complicated. Murphy's Law states,
Anything that can go wrong, will go wrong.
Name: title, dtype: object
```

[8]:
```
data['cleaned_text']=cleaned_text
data['cleaned_summary']=cleaned_summary
```

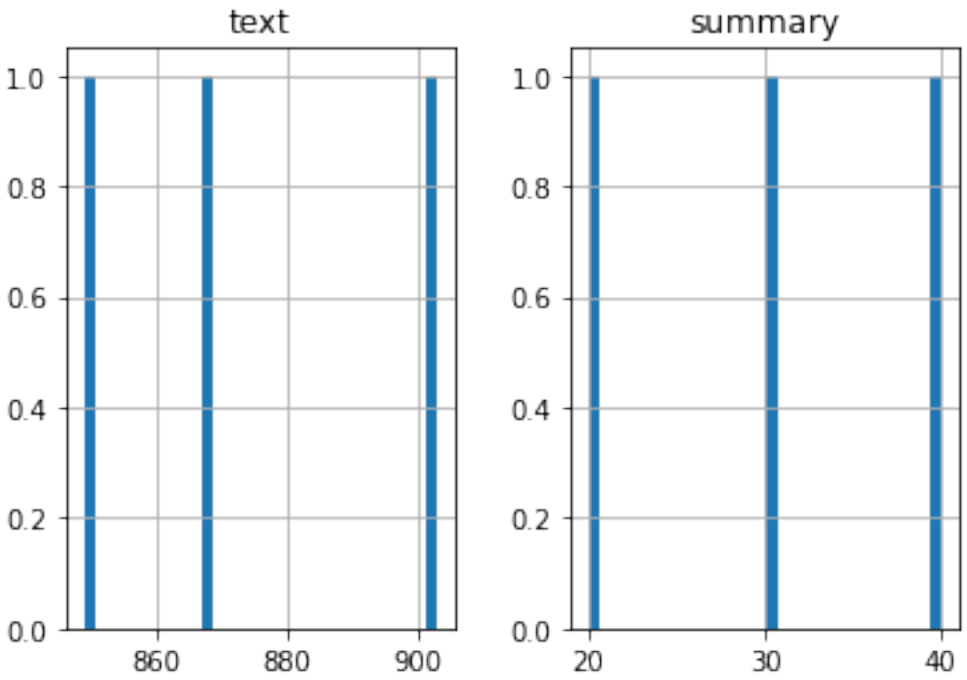
[9]:
```
for i in range(3):
    print("Title:",data['cleaned_summary'][i])
    print("\n")
```

```
Title: mental models how to train your brain to think in new ways you can train
your brain to think better one of the best ways to do this is to expand the set
of mental models you use to think


Title: first principles elon musk on the power of thinking for yourself first
```

principles thinking does not remove the need for continuous improvement but it does alter the direction of improvement

Title: entropy why life always seems to get more complicated murphy law states anything that can go wrong will go wrong

[10]:
```
import matplotlib.pyplot as plt
text_word_count = []
summary_word_count = []

# populate the lists with sentence lengths
for i in data['cleaned_text']:
      text_word_count.append(len(i.split()))

for i in data['cleaned_summary']:
      summary_word_count.append(len(i.split()))

length_df = pd.DataFrame({'text':text_word_count, 'summary':summary_word_count})
length_df.hist(bins = 30)
plt.show()
```



[11]:
```
max_text_len=910
max_summary_len=40
```

[12]:
```
count = 0
for i in data['cleaned_text']:
    if(len(i.split())<=max_text_len):
        count += 1
print(count/len(data['cleaned_text']))
```

```
1.0
```

[13]:
```
cnt=0
for i in data['cleaned_summary']:
    if(len(i.split())<=max_summary_len):
        cnt=cnt+1
print(cnt/len(data['cleaned_summary']))
```

```
1.0
```

[14]:
```
cleaned_text =np.array(data['cleaned_text'])
cleaned_summary=np.array(data['cleaned_summary'])

short_text=[]
short_summary=[]

for i in range(len(cleaned_text)):
    if(len(cleaned_summary[i].split())<=max_summary_len and len(cleaned_text[i].
 ↪split())<=max_text_len):
        short_text.append(cleaned_text[i])
        short_summary.append(cleaned_summary[i])

df=pd.DataFrame({'text':short_text,'summary':short_summary})
```

[15]:
```
df['summary'] = df['summary'].apply(lambda x : 'sostok '+ x + ' eostok')
```

[16]:
```
df['summary'][:3]
```

[16]:
```
0    sostok mental models how to train your brain to think in new ways you can
train your brain to think better one of the best ways to do this is to expand
the set of mental models you use to think eo…
1    sostok first principles elon musk on the power of thinking for yourself
first principles thinking does not remove the need for continuous improvement
but it does alter the direction of improvement…
2
sostok entropy why life always seems to get more complicated murphy law states
anything that can go wrong will go wrong eostok
Name: summary, dtype: object
```

[17]:
```
from sklearn.model_selection import train_test_split
x_tr,x_val,y_tr,y_val=train_test_split(np.array(df['text']),np.
 ↪array(df['summary']),
                                       test_size=0.
 ↪1,random_state=0,shuffle=True)
```

[18]:
```
x_tr=np.array(df['text'])
y_tr=np.array(df['summary'])
```

[19]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_tr))
```

[20]:
```
cnt=0
tot_cnt=0

for key,value in x_tokenizer.word_counts.items():
    tot_cnt=tot_cnt+1

print(tot_cnt)
```

```
1183
```

[21]:
```
x_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
x_tokenizer.fit_on_texts(list(x_tr))

#convert text sequences into integer sequences
x_tr_seq    =   x_tokenizer.texts_to_sequences(x_tr)
x_val_seq   =   x_tokenizer.texts_to_sequences(x_val)

#padding zero upto maximum length
x_tr    =   pad_sequences(x_tr_seq,  maxlen=max_text_len, padding='post')
x_val   =   pad_sequences(x_val_seq, maxlen=max_text_len, padding='post')

#size of vocabulary ( +1 for padding token)
x_voc   =  x_tokenizer.num_words + 1
x_voc
```

[21]:
```
1184
```

[22]:
```
#prepare a tokenizer for reviews on training data
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_tr))
```

[23]:
```
cnt=0
tot_cnt=0

for key,value in y_tokenizer.word_counts.items():
    tot_cnt=tot_cnt+1

print(tot_cnt)
```

```
61
```

[24]:
```
y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
y_tokenizer.fit_on_texts(list(y_tr))

#convert text sequences into integer sequences
y_tr_seq    =   y_tokenizer.texts_to_sequences(y_tr)
y_val_seq   =   y_tokenizer.texts_to_sequences(y_val)

#padding zero upto maximum length
y_tr    =   pad_sequences(y_tr_seq, maxlen=max_summary_len, padding='post')
y_val   =   pad_sequences(y_val_seq, maxlen=max_summary_len, padding='post')

#size of vocabulary
y_voc  =   y_tokenizer.num_words +1
y_voc
```

[24]:
```
62
```

[25]:
```
y_tokenizer.word_counts['eostok'],len(y_tr)
```

[25]:
```
(3, 3)
```

[26]:
```
from tensorflow.keras.models import Model
from keras import backend as K
from attention_keras.src.layers.attention import AttentionLayer
K.clear_session()

latent_dim = 300
embedding_dim=100

# Encoder
encoder_inputs = Input(shape=(max_text_len,))

#embedding layer
enc_emb =  Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)

#encoder lstm 1
```

```
encoder_lstm1 =␣
 ↪LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.
 ↪4,recurrent_dropout=0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
encoder_lstm2 =␣
 ↪LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.
 ↪4,recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#encoder lstm 3
encoder_lstm3=LSTM(latent_dim, return_state=True,␣
 ↪return_sequences=True,dropout=0.4,recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))

#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim, return_sequences=True,␣
 ↪return_state=True,dropout=0.4,recurrent_dropout=0.2)
decoder_outputs,decoder_fwd_state, decoder_back_state =␣
 ↪decoder_lstm(dec_emb,initial_state=[state_h, state_c])

# Attention layer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

# Concat attention input and decoder LSTM output
decoder_concat_input = Concatenate(axis=-1,␣
 ↪name='concat_layer')([decoder_outputs, attn_out])

#dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________
```

```
__________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_1 (InputLayer)            [(None, 910)]        0
----------------------------------------------------------------------------------
------------------
embedding (Embedding)           (None, 910, 100)     118400      input_1[0][0]
----------------------------------------------------------------------------------
------------------
lstm (LSTM)                     [(None, 910, 300), ( 481200      embedding[0][0]
----------------------------------------------------------------------------------
------------------
input_2 (InputLayer)            [(None, None)]       0
----------------------------------------------------------------------------------
------------------
lstm_1 (LSTM)                   [(None, 910, 300), ( 721200      lstm[0][0]
----------------------------------------------------------------------------------
------------------
embedding_1 (Embedding)         (None, None, 100)    6200        input_2[0][0]
----------------------------------------------------------------------------------
------------------
lstm_2 (LSTM)                   [(None, 910, 300), ( 721200      lstm_1[0][0]
----------------------------------------------------------------------------------
------------------
lstm_3 (LSTM)                   [(None, None, 300),  481200
embedding_1[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
----------------------------------------------------------------------------------
------------------
attention_layer (AttentionLayer ((None, None, 300),  180300      lstm_2[0][0]
                                                                 lstm_3[0][0]
----------------------------------------------------------------------------------
------------------
concat_layer (Concatenate)      (None, None, 600)    0           lstm_3[0][0]
attention_layer[0][0]
----------------------------------------------------------------------------------
------------------
time_distributed (TimeDistribut (None, None, 62)     37262
concat_layer[0][0]
==================================================================================
==================
Total params: 2,746,962
Trainable params: 2,746,962
Non-trainable params: 0
----------------------------------------------------------------------------------
------------------
```

[27]:
```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')
# model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
```

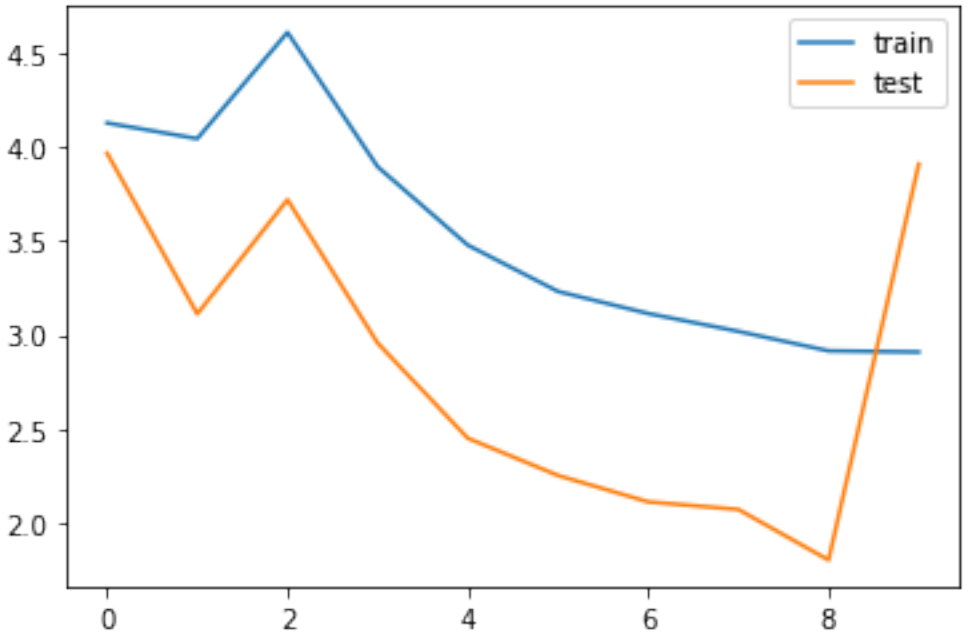
[28]:
```
from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,patience=2)
```

[29]:
```
history=model.fit([x_tr,y_tr[:,:-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1],␣
↪1)[:,1:] ,
                  ␣
↪epochs=10,callbacks=[es],batch_size=128,validation_data=([x_val,y_val[:,:
↪-1]], y_val.reshape(y_val.shape[0],y_val.shape[1], 1)[:,1:]))
```

```
Epoch 1/10
1/1 [==============================] - 3s 3s/step - loss: 4.1273 - val_loss:
3.9661
Epoch 2/10
1/1 [==============================] - 2s 2s/step - loss: 4.0438 - val_loss:
3.1106
Epoch 3/10
1/1 [==============================] - 2s 2s/step - loss: 4.6098 - val_loss:
3.7189
Epoch 4/10
1/1 [==============================] - 2s 2s/step - loss: 3.8942 - val_loss:
2.9553
Epoch 5/10
1/1 [==============================] - 2s 2s/step - loss: 3.4767 - val_loss:
2.4486
Epoch 6/10
1/1 [==============================] - 2s 2s/step - loss: 3.2302 - val_loss:
2.2504
Epoch 7/10
1/1 [==============================] - 2s 2s/step - loss: 3.1127 - val_loss:
2.1098
Epoch 8/10
1/1 [==============================] - 2s 2s/step - loss: 3.0176 - val_loss:
2.0695
Epoch 9/10
1/1 [==============================] - 2s 2s/step - loss: 2.9137 - val_loss:
1.8002
Epoch 10/10
1/1 [==============================] - 2s 2s/step - loss: 2.9078 - val_loss:
3.9088
```

[30]:
```
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
```

```
pyplot.show()
```




[50]:
```
reverse_target_word_index=y_tokenizer.index_word
reverse_source_word_index=x_tokenizer.index_word
target_word_index=y_tokenizer.word_index
```

[51]:
```
# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h,␣
 ↪state_c])

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_text_len,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the␣
 ↪states from the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2,␣
 ↪initial_state=[decoder_state_input_h, decoder_state_input_c])

#attention inference
```

```
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input,␣
↪decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2,␣
↪attn_out_inf])

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h,␣
↪decoder_state_input_c],
    [decoder_outputs2] + [state_h2, state_c2])
```

[52]:
```
def decode_sequence(input_seq):
    e_out, e_h, e_c = encoder_model.predict(input_seq)
    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Chose the 'start' word as the first word of the target sequence
    target_seq[0, 0] = target_word_index['sostok']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([[target_seq] + [e_out,␣
↪e_h, e_c]])
        output_tokens = output_tokens[:1]
        h_1 = h[:1]
        c_1 = c[:1]
        prediction_output = output_tokens[0,-1,:]
        sampled_token_index = np.random.choice(len(prediction_output),␣
↪p=prediction_output)
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='eostok'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eostok' or len(decoded_sentence.split()) >=␣
↪(max_summary_len-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index
```

```
        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

[53]:
```
def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if(i!=0 and i!=target_word_index['sostok'] and i!
 ↪=target_word_index['eostok']):
            newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=''
    for i in input_seq:
        if(i!=0):
            newString=newString+reverse_source_word_index[i]+' '
    return newString
```

[54]:
```
for i in range(len(x_tr)):
    print("Text:",seq2text(x_tr[i])[:300])
    print("Original title:",seq2summary(y_tr[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,max_text_len)))
    print("\n")
```

```
Text: train brain think better one best ways expand set mental models use think
let explain mean sharing story world class thinker first discovered mental model
useful right one could reading story richard feynman famous physicist feynman
received undergraduate degree mit ph princeton time developed reput
Original title: models how to train your brain to think in new ways you can
train your brain to think better one of the best ways to do this is to expand
the set of mental models you use to think
Predicted summary:  train always power on of go principles the


Text: first principles thinking sometimes called reasoning first principles one
effective strategies employ breaking complicated problems generating original
solutions also might single best approach learn think first principles approach
used many great thinkers including inventor johannes gutenberg milit
Original title: first principles elon musk on the power of thinking for yourself
first principles thinking does not remove the need for continuous improvement
but it does alter the direction of improvement
Predicted summary:  expand better that one yourself
```

```
Text: murphy law states anything go wrong go wrong pithy statement references
annoying tendency life cause trouble make things difficult problems seem arise
naturally solutions always require attention energy effort life never seems work
us anything lives become complicated gradually decline disorder rath
Original title: entropy why life always seems to get more complicated murphy law
states anything that can go wrong go wrong
Predicted summary:  use entropy models on sostok law this one expand will
anything on better will new models models for elon continuous wrong is direction
seems sostok will do get need best new how get set sostok ways law but better
```

[ ]: