Obl1

September 13, 2019

0.1 Oblig 1 in NLP

0.1.1 Exercise 1

[1]:

```
import nltk
from nltk.corpus import brown
import pandas as pd
from nltk import pos_tag
```

[2]:

```
labels = ['he', 'him', 'she','her']
genres = ['news','fiction','religion','government','romance']
```

[3]:

```
cond = nltk.ConditionalFreqDist([(genre, word.lower())
                                  for genre in genres
                                  for word in brown.words(categories=genre)
                                  if word.lower() in labels])
df=pd.DataFrame(cond, columns=genres)
df.fillna(0)
freq_labels = df.sum(axis = 1, skipna = True)
df['total'] = freq_labels.to_frame()

sum_genres = df.sum(axis = 0, skipna = True)
modDfObj = df.append(sum_genres, ignore_index=True)

total = modDfObj['total'][4]
print(total)
df
```

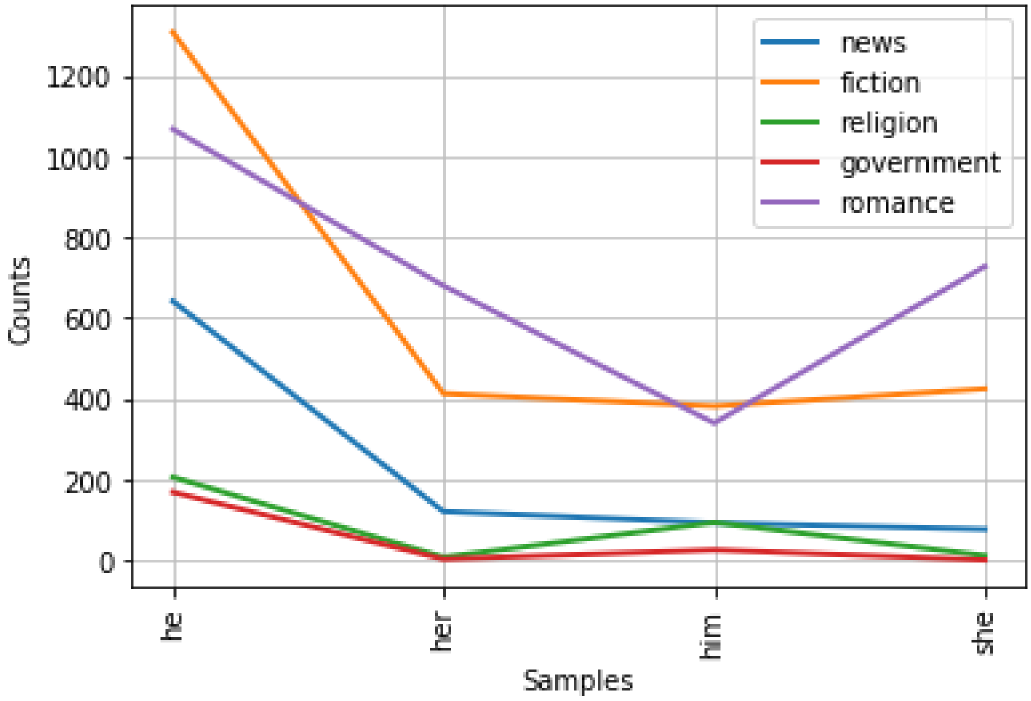
```
6796
```

[3]:

```
     news  fiction  religion  government  romance  total
he    642     1308       206         169     1068   3393
him    93      382        94          26      340    935
she    77      425        12           1      728   1243
her   121      413         8           3      680   1225
```

[4]:

```
%matplotlib inline
cond.plot()
```



[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f41aa704790>
```

From the observations we can see that the masculine forms are more frequent than the feminine forms across all genres. The objective forms has an almost equal frequency throughout the chart. We can also see that news and fiction graphs follow the same pattern and her him she has a constant number in the genres. Romance tends to use the nominative forms more frequent than the objective form. Government and religion has the lowest frequency count of the labels.

[5]:
```
tk = nltk.ConditionalFreqDist()

for genre in genres:
    for word in brown.words(categories=genre):
        w = word.lower()
        if w in labels:
            if w == 'he' or w == 'she':
                tk[genre]['nom_form'] += 1
            elif w == 'him' or w == 'her':
                tk[genre]['obj_form'] += 1

df1=pd.DataFrame(tk, columns=genres)
df1.fillna(0)
```

[5]:

	news	fiction	religion	government	romance
nom_form	719	1733	218	170	1796
obj_form	214	795	102	29	1020

[6]:

```
he = float("{0:.3f}".format(freq_labels['he'] / total))
him = float("{0:.3f}".format(freq_labels['him'] / total))
she = float("{0:.3f}".format(freq_labels['she'] / total))
her = float("{0:.3f}".format(freq_labels['her'] / total))
data = [ [(he) ,(him) ,(she), (her)]]
df = pd.DataFrame(data, columns = ['he', 'him','she','her'])
df
```

[6]:

```
      he    him    she   her
0  0.499  0.138  0.183  0.18
```

[7]:

```
brown_tagged = brown.tagged_words(categories=genres, tagset='universal')
brown_tagged_pron = [word.lower() for (word, tag) in brown_tagged if tag ==␣
 ↪'PRON']

pos_tagged = pos_tag(brown_tagged_pron)
labels = ['he', 'him', 'she','her','hers','his']
labels.sort()
tagged_words = [
            (word,tag)
                for (word,tag) in pos_tagged
                    if word.lower() in labels
                    if tag == 'PRP$'
                    or tag == 'PRP']
```

[ ]:

```

```

[8]:

```
cond = nltk.ConditionalFreqDist(tagged_words)

df2=pd.DataFrame(cond, columns=labels).fillna(0)
df2
```

[8]:

```
           he  her  hers    him   his     she
PRP    3393.0   39     0  935.0   0.0  1243.0
PRP$      0.0  397     0    0.0  17.0     0.0
```

[9]:

```
her = float("{0:.3f}".format(df2['her'].PRP / total))
he = float("{0:.3f}".format(df2['he'].PRP / total))
she = float("{0:.3f}".format(df2['she'].PRP / total))
him = float("{0:.3f}".format(df2['him'].PRP / total))
data = [ [(he) ,(him) ,(she), (her)]]
df_new = pd.DataFrame(data, columns = ['he', 'him','she','her'])
df
```

[9]:

```
      he    him    she   her
0  0.499  0.138  0.183  0.18
```

[10]:

```
df_new
```

[10]:

```
      he    him    she    her
0  0.499  0.138  0.183  0.006
```

I think that the masculine pronoun is more frequent than the feminine pronoun because I think men use more personal pronouns when writing, maybe to display power and influence over others. I also think that for historical reasons he is more used then she.

In language, words are unique but can have different meanings depending on the context in which they are being evaluated. This results in ambiguity. We can have words or even sentences with different meanings in the same sentence depending on the way we interpret these words. I think ambiguity is a problem when developing language technology. I also think that the process of linking together mentions that relates to persons is hard. For example linking she or he to the person referred to.

0.2 Exercise 2

[11]:
```
from urllib import request
from nltk import word_tokenize
import re, string
import numpy as np
```

[12]:
```
url = "https://www.gutenberg.org/files/74/74-0.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
```

[13]:
```
text = re.search("(?<=HARTFORD, 1876.)[\s\S]*CONCLUSION", raw)[0] # lookbehind␣
  ↪from conclusion to start of book
text = text[:-10] # remove word: conclusion
text = text.replace('.','') # replace punctuation with nothing
tokens = nltk.word_tokenize(text)
tokens[:5] # first five words in book
tokens = [t.lower() for t in tokens if t.isalpha()]
```

[14]:
```
# I do not want to case fold the text since some words will loose its meaning,␣
  ↪like names etc.
```

[15]:
```
tokens_freq = nltk.FreqDist(tokens)
most_common = tokens_freq.most_common(20)
token_df=pd.DataFrame(most_common, columns=['token', 'freq'])
token_df.style.hide_index()
```

[15]:
```
<pandas.io.formats.style.Styler at 0x7f41a790a890>
```

[16]:
```
def n_times(n):
    return df.freq == n

def between_11_and_50():
    x = df.freq > 11
    y = df.freq < 50
    return x & y

def between_50_and_100():
    x = df.freq > 50
    y = df.freq < 100
    return x & y

def from_51_to_100():
    x = df.freq >= 51
    y = df.freq <= 100
    return x & y
```

[17]:

```
df = pd.DataFrame(tokens_freq.items(), columns=['token', 'freq'])

select_n_times = df.loc[n_times(3)]

select_one_time = df.loc[df.freq == 1]

select_between_11_and_50 = df.loc[between_11_and_50()]

select_between_50_and_100 = df.loc[between_50_and_100()]

select_larger_than_100 = df.loc[df.freq > 100]

select_51_100 = df.loc[from_51_to_100()]
```

[18]:

```
x = df.loc[df.freq > 3000]
```

[19]:

```
a = select_one_time.sum(axis=1).count()
b = select_between_11_and_50.sum(axis=1).count()
c = select_between_50_and_100.sum(axis=1).count()
d = select_larger_than_100.sum(axis=1).count()
e = select_51_100.sum(axis=1).count()
Selectors = ['one time', 'between 11 and 50', 'between 50 and 100', 'larger␣
 ↪than 100','51-100']
```

[20]:

```
raw_data = {'Selectors': Selectors,
        'Occurrences': [a,b,c,d,e]}
df = pd.DataFrame(raw_data, columns = ['Selectors', 'Occurrences'])
df
```

[20]:

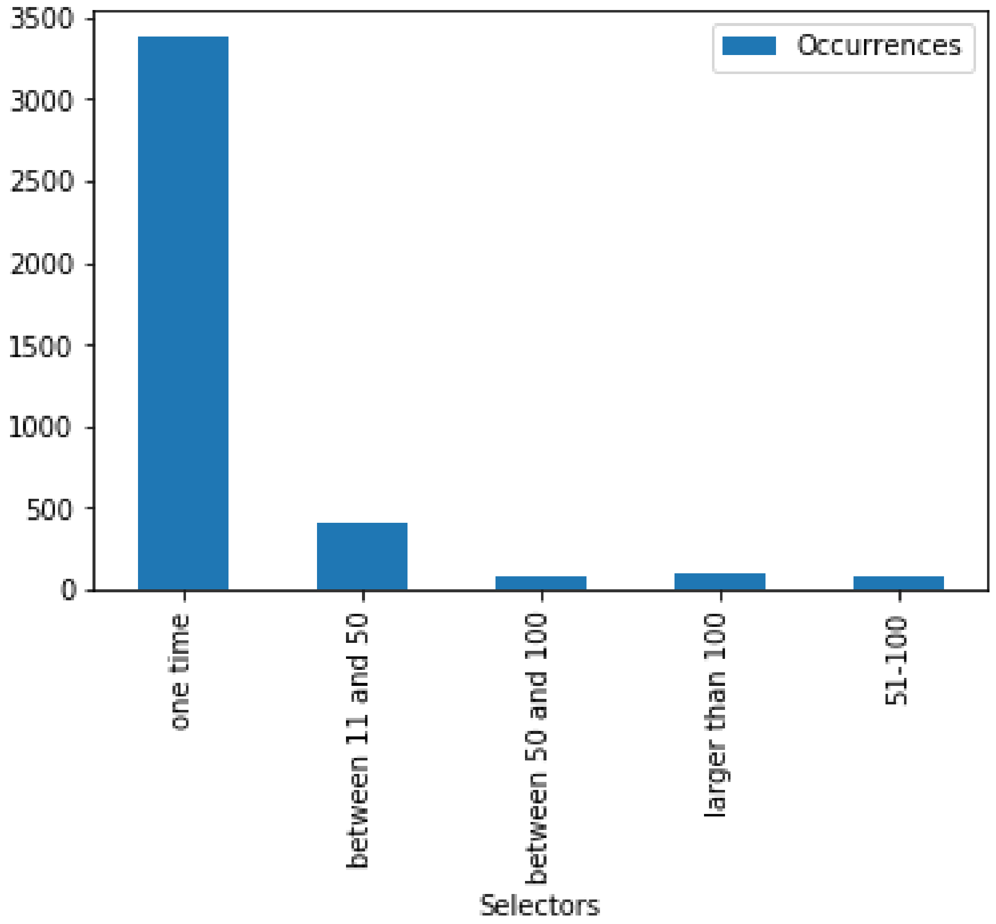
```
            Selectors  Occurrences
0            one time         3384
1   between 11 and 50          407
2  between 50 and 100           79
3     larger than 100          101
4              51-100           79
```

[21]:

```
df.plot(kind='bar',x='Selectors',y='Occurrences')
```

[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f41a710e310>
```



[22]:

```
words = [(word) for (word,freq) in tokens_freq.most_common(20)]
freq = [(freq) for (word,freq) in tokens_freq.most_common(20)]
rank = [rank for rank in range(1,21)]
```

[23]:

```
raw_data = {
        'Words': words,
        'Frequency': freq,
        'Rank': rank
}
df = pd.DataFrame(raw_data, columns = ['Words', 'Frequency', 'Rank'])

df['zip'] = df.Frequency*df.Rank
df
```

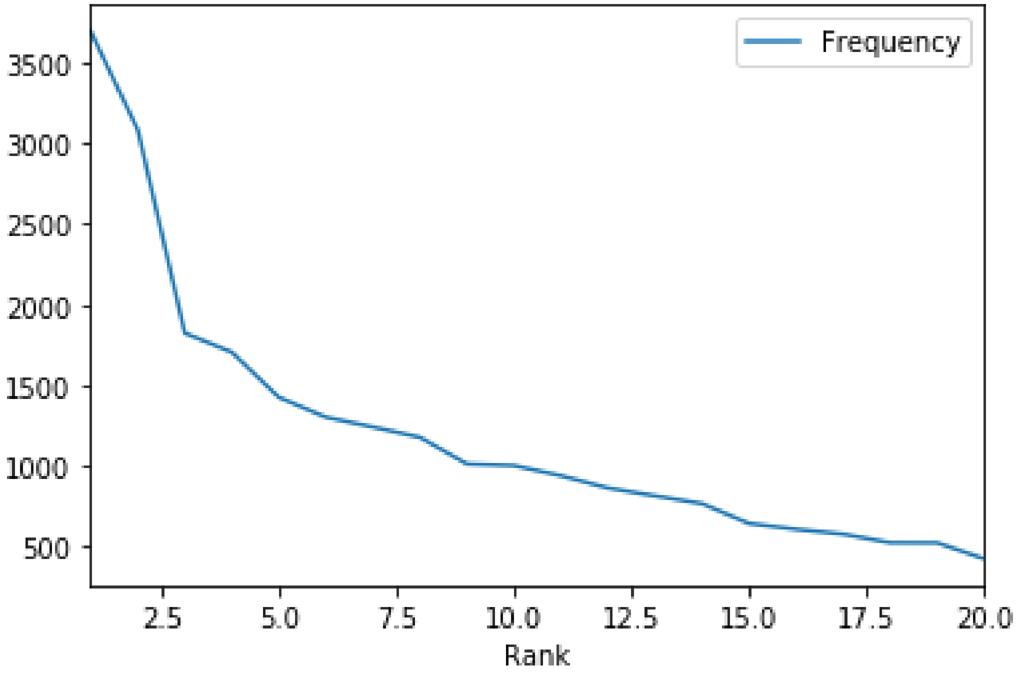
[23]:

```
  Words  Frequency  Rank   zip
0   the       3694     1  3694
1   and       3082     2  6164
2     a       1824     3  5472
3    to       1706     4  6824
```

```
4     of      1426    5   7130
5     it      1303    6   7818
6     he      1242    7   8694
7    was      1178    8   9424
8   that      1013    9   9117
9      i      1003   10  10030
10    in       940   11  10340
11   you       864   12  10368
12   his       815   13  10595
13   tom       767   14  10738
14  with       645   15   9675
15  they       609   16   9744
16   but       579   17   9843
17   had       526   18   9468
18   for       525   19   9975
19   him       426   20   8520
```

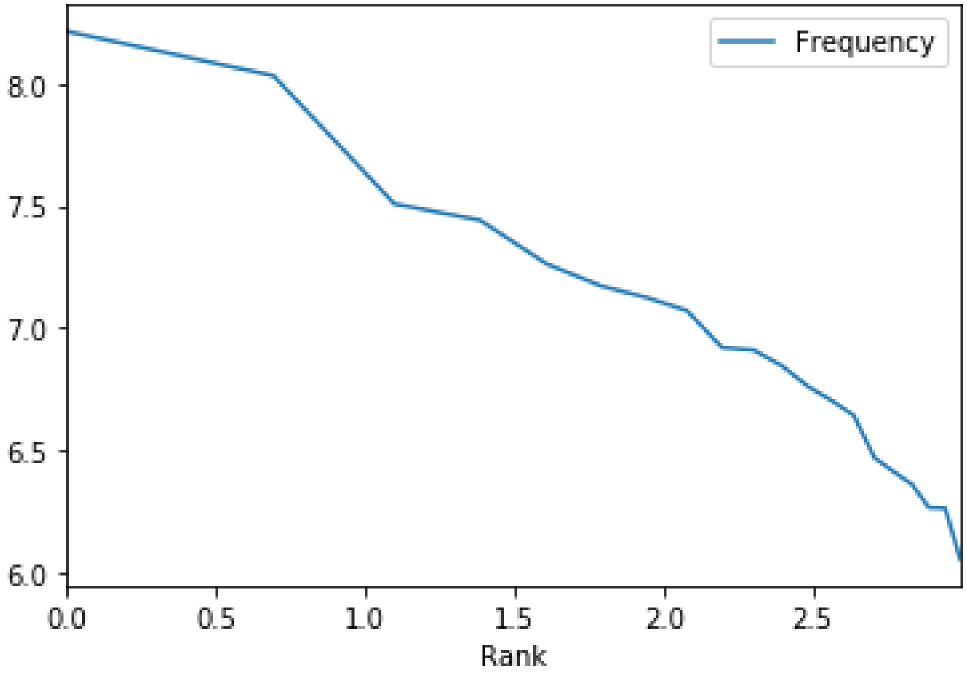
[24]:
```python
df.plot(kind='line',x='Rank',y='Frequency')
```

[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f41a6fda990>
```



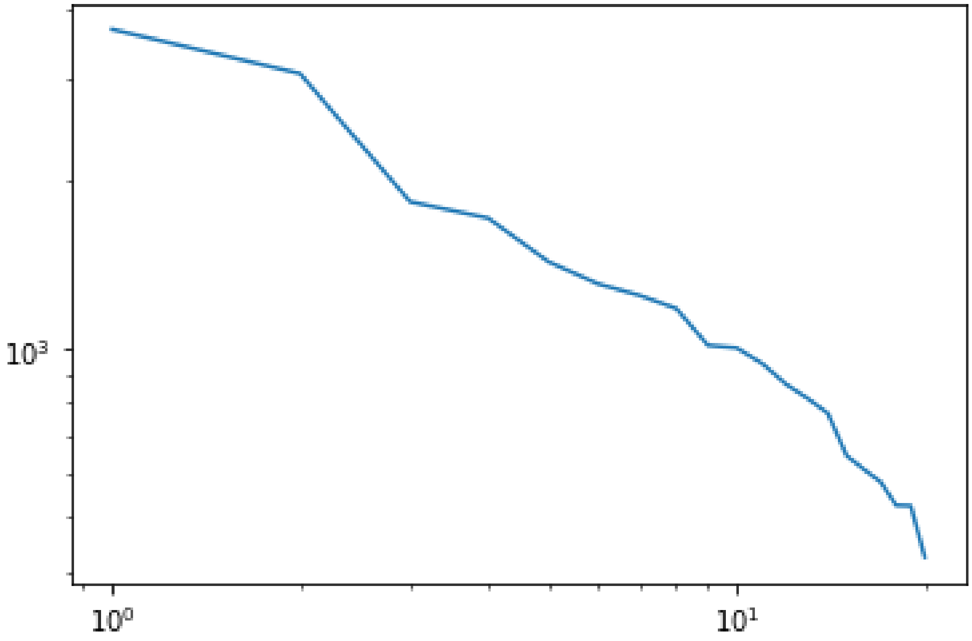
[25]:
```python
df_copy = df.copy()
df_copy['Rank'] = np.log(df.Rank.values)
df_copy['Frequency'] = np.log(df.Frequency.values)
df_copy.plot(kind='line',x='Rank',y='Frequency')
```

```
[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f41a6fcf110>
```



```
[26]: import matplotlib.pyplot as plt
      plt.loglog(df.Rank.values, df.Frequency.values)
```

```
[26]: [<matplotlib.lines.Line2D at 0x7f41a6857410>]
```



[ ]:

```
 
```

0.2.1 Exercise 3

[27]:

```
def factorial(n):
    if n == 1:
        return n
    else:
        return n * factorial(n-1)
t = factorial(5)
t
```

[27]:

```
120
```

[28]:

```
def binom(n,m):
    if n == m:
        return 1
    return factorial(n)//(factorial(m)*factorial(n-m))
print(binom(10,7))
print(binom(5,3))
print(1/binom(34,7)) # $lotto$
print(binom(1,1))
```

```
120
10
1.858868737099451e-07
1
```

[29]:

```
def binom_pmf(k, n, p):
    if k == 0:
        return
    if k>=n:
        return 0
    return binom(n,k)*p**k*(1-p)**(n-k)

print("Eksempel: Dersom man kaster en terning tre ganger, og terningen er␣
 ↪velbygd, slik at sannsynligheten for å få en sekser er 1/6, blir␣
 ↪sannsynligheten for å få sekser to ganger")
print(binom_pmf(2,3,1/6))
```

```
Eksempel: Dersom man kaster en terning tre ganger, og terningen er velbygd, slik
at sannsynligheten for å få en sekser er 1/6, blir sannsynligheten for å få
sekser to ganger
0.06944444444444445
```

[30]:
```
def binom_cdf(k,n,p):
    answer = 0
    for i in range(1,k+1):
        answer = answer + binom_pmf(i,n,p)
    return answer

binom_cdf(15,100,0.2)
```

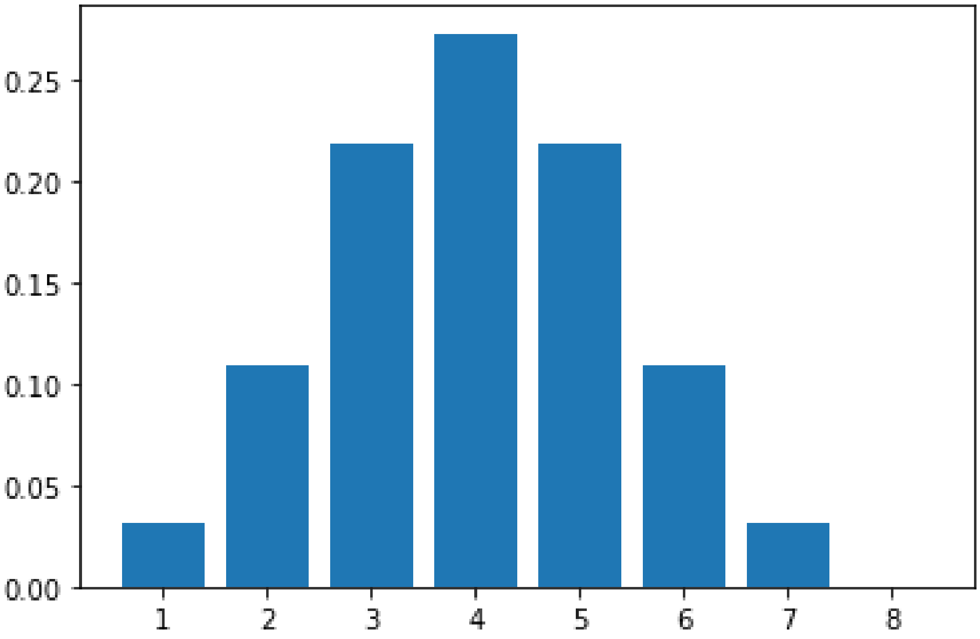
[30]:
```
0.1285055146350941
```

[31]:
```
n = 8
p = 0.5
k = [1,2,3,4,5,6,7,8]
pmf = []
cdf = []
for i in k:
    cdf.append(binom_cdf(i,n,p))

for i in k:
    pmf.append(binom_pmf(i,8,p))
print("pmf ",pmf)
print("cdf ", cdf)
raw_data = {
        'pmf': pmf,
        'cdf': cdf
}
df = pd.DataFrame(raw_data, columns = ['pmf', 'cdf'])
df
```

```
pmf  [0.03125, 0.109375, 0.21875, 0.2734375, 0.21875, 0.109375, 0.03125, 0]
cdf  [0.03125, 0.140625, 0.359375, 0.6328125, 0.8515625, 0.9609375, 0.9921875,
0.9921875]
```

[31]:
```
        pmf       cdf
0  0.031250  0.031250
1  0.109375  0.140625
2  0.218750  0.359375
3  0.273438  0.632812
4  0.218750  0.851562
5  0.109375  0.960938
6  0.031250  0.992188
7  0.000000  0.992188
```

[32]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.bar(k,pmf)
plt.show()
```



[33]:

```
cdf = binom_cdf(5,5,1/6)
cdf
```

[33]:

```
0.5979938271604939
```

[ ]:

0.2.2 Exercise 4

[34]:

```
# the function simulates The Bernoulli distribution. The Bernoulli distribution␣
 ↪describes events having exactly two outcomes.
import random

def mean(arr):
    total=0
    for i in arr:
        total = total + i
    return total/len(arr)

def bernoulli(p):
    if random.random() < p:
        return 1
    else:
        return 0
```

[35]:

```
results = [i for i in range(10) if bernoulli(0.5) == 1] # returns if sucess
len(results)/10
```

[35]:
```
0.8
```

[36]:
```
from collections import Counter
```

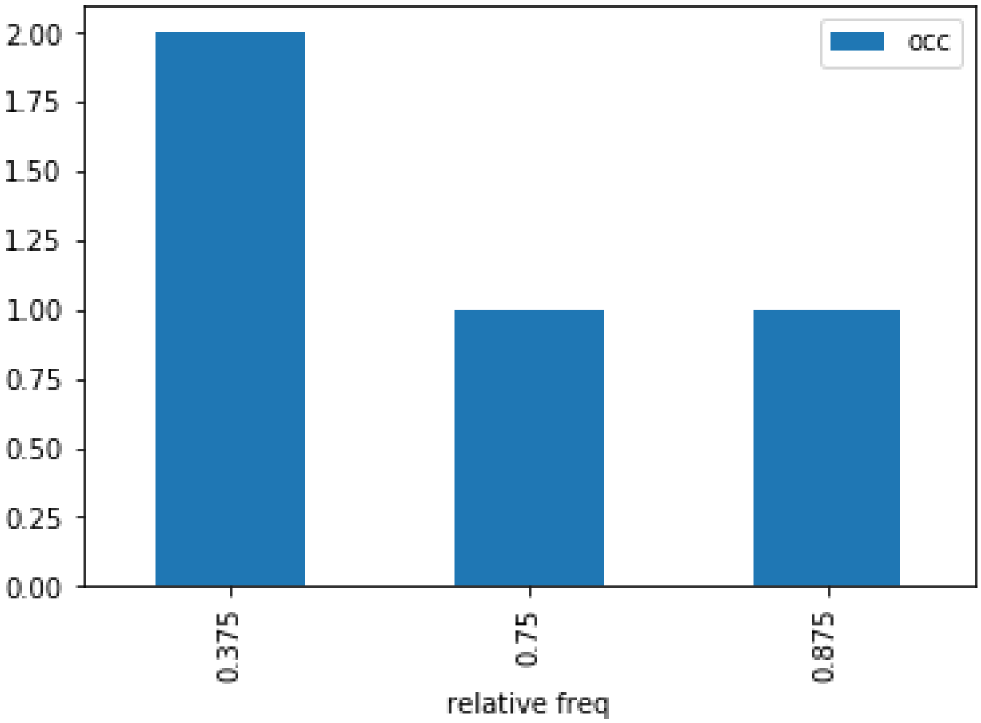
[37]:
```
def bin_exper(n, p):
    return len([i for i in range(n) if bernoulli(p) == 1]) # returns if sucess

means = []
for i in range(20):
    means.append(bin_exper(10, 0.5))

mean(means)/10
```

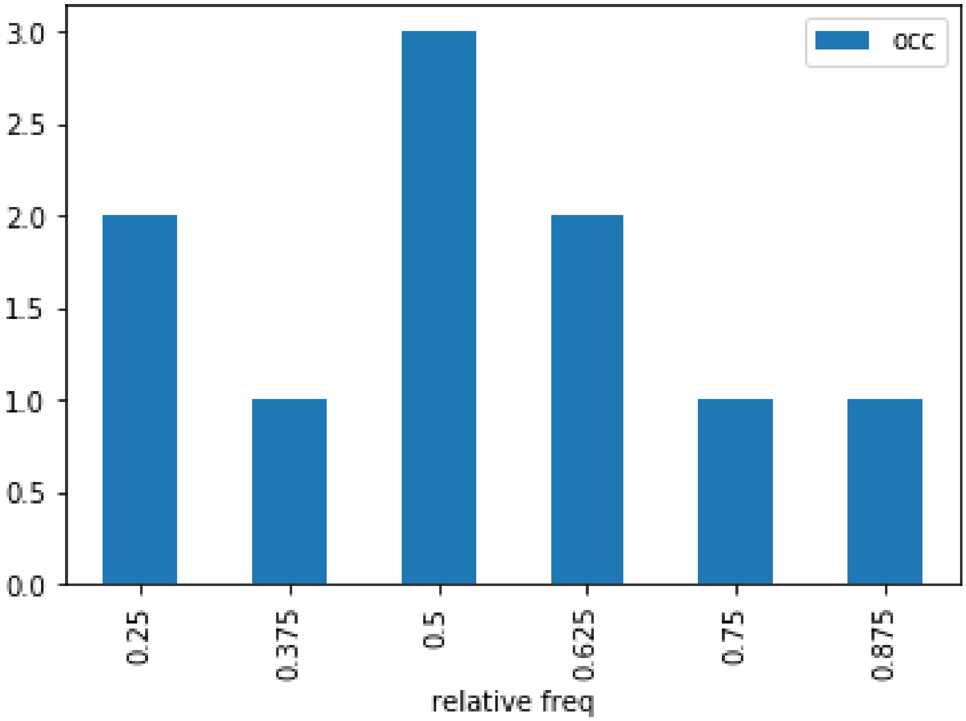
[37]:
```
0.52
```

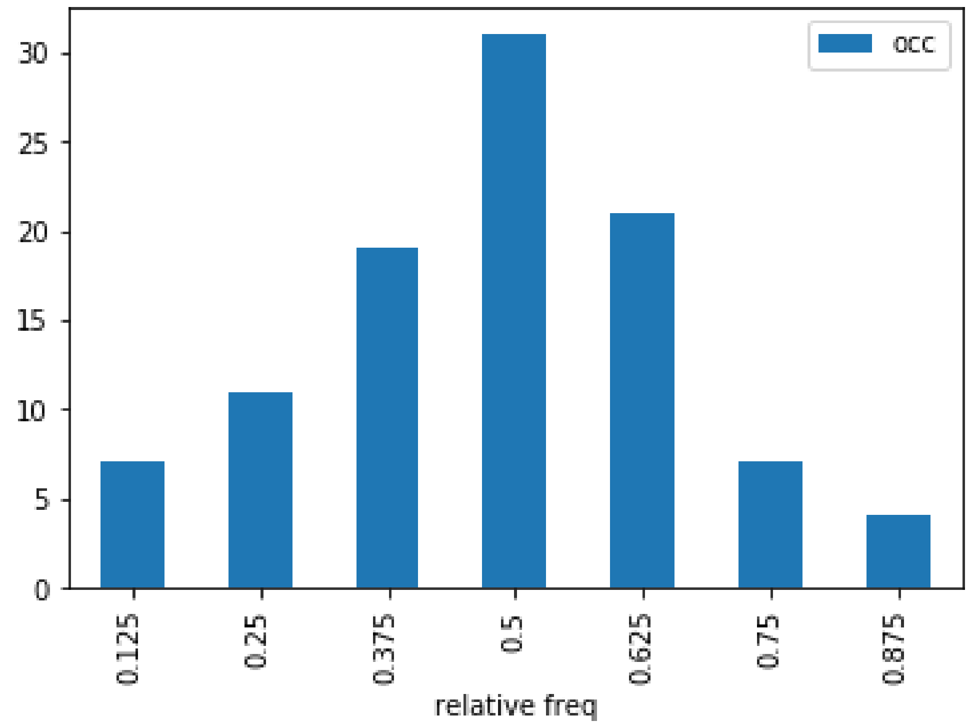
[41]:
```
def bin_freqs(m, n, p):
    freq = []
    for i in range(m):
        x = bin_exper(n,p)/n
        freq.append(x)
    return nltk.FreqDist(freq)


freqs = bin_freqs(10,8,0.5)

df=pd.DataFrame(freqs.items(),columns=['relative freq', 'occ'])
df
```

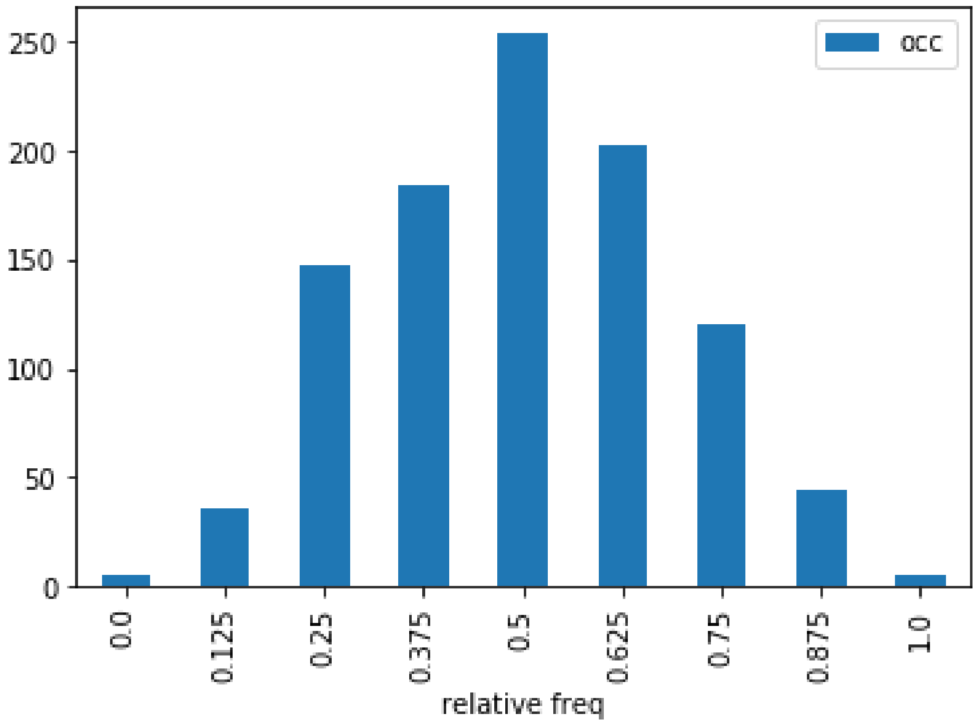
[41]:
```
   relative freq  occ
0          0.500    3
1          0.250    1
2          0.375    3
3          0.625    2
4          0.750    1
```

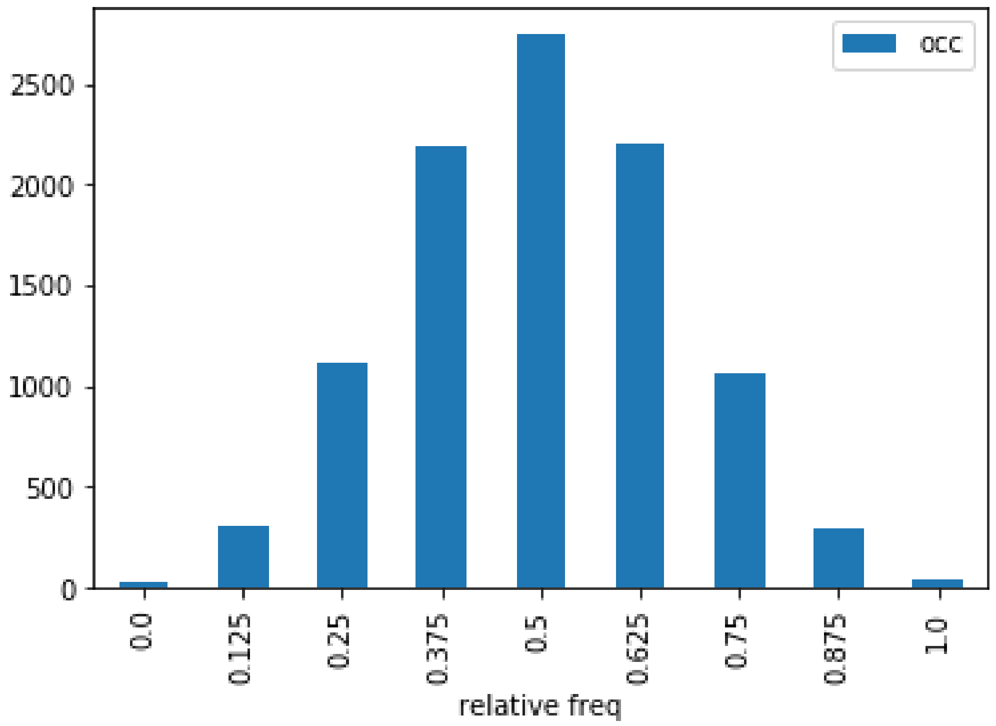
[39]:
```
n = 8
p = 0.5
m = [4,10,100,1000,10000]
items = []
for i in m:
    df=pd.DataFrame(bin_freqs(i,8,0.5).items(),columns=['relative freq', 'occ'])
    df.sort_values('relative freq',inplace=True)
    df.plot(kind='bar',x='relative freq',y='occ')
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
occ
0.375
0.75
0.875
relative freq


3.0
2.5
2.0
1.5
1.0
0.5
0.0
occ
0.25
0.375
0.5
0.625
0.75
0.875
relative freq


occ
30
25
20
15
10
5
0
0.125
0.25
0.375
0.5
0.625
0.75
0.875
relative freq


occ
250
200
150
100
50
0
0.0
0.125
0.25
0.375
0.5
0.625
0.75
0.875
1.0
relative freq


occ
2500
2000
1500
1000
500
0
0.0
0.125
0.25
0.375
0.5
0.625
0.75
0.875
1.0
relative freq

[ ]:

```
```

[ ]:

```
```