# LING 388: Computers and Language

Lecture 19

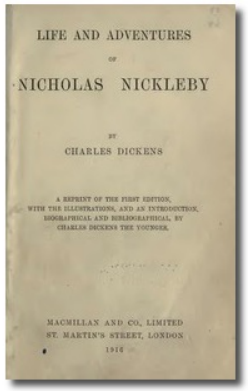
# Today's Topics

- Homework 7 Review
- New Topic:
  - `nltk.bigrams()`
  - Generating random text using `nltk.ConditionalFreqDist()`
  - Brown Corpus
    - `brown.fileids()`
    - `brown.categories()`
    - `brown.words(categories='news')`
  - using `nltk.ConditionalFreqDist()` with the Brown corpus

# Homework 7 Review

Project Gutenberg › 73,133 free eBooks › 202 by Charles Dickens

## Nicholas Nickleby by Charles Dickens

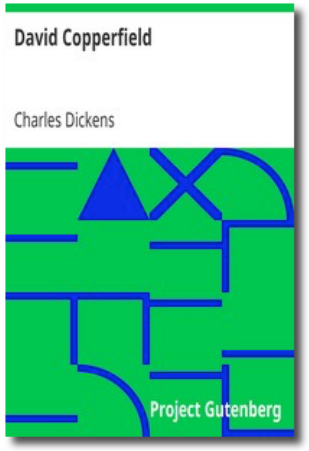



### Read now or download (free!)

| | Choose how to read this book | Size | | | |
|---|---|---|---|---|---|
| | Read online (web) | 2.2 MB | | | |
| | EPUB3 (E-readers incl. Send-to-Kindle) | 23.2 MB | | | |
| | EPUB (older E-readers) | 23.2 MB | | | |
| | EPUB (no images, older E-readers) | 871 kB | | | |
| | Kindle | 23.8 MB | | | |
| | older Kindles | 23.7 MB | | | |
| | Plain Text UTF-8 | 1.8 MB | | | |
| | Download HTML (zip) | 22.8 MB | | | |

# Homework 7 Review

## David Copperfield by Charles Dickens




### Read now or download (free!)

| | Choose how to read this book | Size | | | |
|---|---|---|---|---|---|
| | Read online (web) | 2.0 MB | | | |
| | EPUB3 (E-readers incl. Send-to-Kindle) | 8.9 MB | | | |
| | EPUB (older E-readers) | 8.8 MB | | | |
| | EPUB (no images, older E-readers) | 906 kB | | | |
| | Kindle | 9.3 MB | | | |
| | older Kindles | 9.3 MB | | | |
| | Plain Text UTF-8 | 1.9 MB | | | |

# Homework 7 Review

```
$ python
>>> raw = open('nn.txt').read()
>>> len(raw)
1848364
>>> import nltk
>>> nn = nltk.word_tokenize(raw)
>>> len(nn)
396970
>>> raw2 = open('dc.txt').read()
>>> len(raw2)
1934660
>>> dc = nltk.word_tokenize(raw2)
>>> len(dc)
443615
>>> raw3 =
open('oliver_twist.txt').read()
>>> len(raw3)
893534
>>> ot = nltk.word_tokenize(raw3)
>>> len(ot)
199836
```

# Homework 7 Review

Step 4: remove the punctuation

```
>>> def isword(x):
...     return any(c.isalpha() for c in x)
...
>>> nn2 = [word for word in nn if isword(word)]
>>> len(nn2)
328468
>>> dc2 = [word for word in dc if isword(word)]
>>> len(dc2)
361824
>>> ot2 = [word for word in ot if isword(word)]
>>> len(ot2)
160639
```

# Homework 7 Review

- Step 6: use a list comprehension to grab the word lengths

```
>>> def chunk(words, nth, n):
...     return [len(word) for word in words[(nth - 1)*n: nth*n]]
...
>>> lnn1 = chunk(nn2, 1, 5000)
>>> len(lnn1)
5000
>>> lnn2 = chunk(nn2, 2, 5000)
>>> len(lnn2)
5000
>>> lnn1[:20]
[3, 4, 3, 10, 2, 8, 8, 10, 1, 8, 7, 2, 3, 8, 11, 9, 12, 3, 8, 6]
>>> lnn2[:20]
[8, 3, 11, 4, 3, 9, 6, 4, 2, 7, 3, 4, 7, 3, 7, 2, 3, 7, 2, 12]
>>> ldc1 = chunk(dc2, 1, 5000)
>>> ldc2 = chunk(dc2, 2, 5000)
>>> len(ldc1)
5000
>>> len(ldc2)
5000
>>> lot1 = chunk(ot2, 1, 5000)
>>> lot2 = chunk(ot2, 2, 5000)
>>> len(lot1)
5000
>>> len(lot2)
5000
```

# Homework 7 Review

## Step 7: histogram plot them with overlay

```
>>> import matplotlib.pyplot as plt
>>> mx = max(max(lnn1),max(lnn2),max(ldc1),max(ldc2),max(lot1),max(lot2))
>>> mx
20
>>> fig , (axs1, axs2, axs3) = plt.subplots(3, sharex=True, sharey=True)
>>> axs1.set_title('Nicholas Nickleby')
Text(0.5, 1.0, 'Nicholas Nickleby')
>>> axs2.set_title('David Copperfield')
Text(0.5, 1.0, 'David Copperfield')
>>> axs3.set_title('Oliver Twist')
Text(0.5, 1.0, 'Oliver Twist')
```

# Homework 7 Review

## Step 7: histogram plot them with overlay

```
>>> axs1.hist(lnn1, range(1,mx+1), histtype='step', label='1st 5000')
(array([2.050e+02, 9.540e+02, 1.082e+03, 7.970e+02, 5.240e+02, 4.070e+02,
       3.220e+02, 2.460e+02, 2.060e+02, 1.160e+02, 6.100e+01, 4.700e+01,
       1.400e+01, 8.000e+00, 7.000e+00, 3.000e+00, 1.000e+00, 0.000e+00,
       0.000e+00]), array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20]), [<matplotlib.patches.Polygon object at 0x16d0be760>])
>>> axs1.hist(lnn2, range(1,mx+1), histtype='step', label='2nd 5000')
(array([1.830e+02, 8.120e+02, 1.107e+03, 8.450e+02, 5.660e+02, 4.370e+02,
       3.610e+02, 2.750e+02, 1.730e+02, 1.150e+02, 5.200e+01, 4.000e+01,
       1.500e+01, 6.000e+00, 5.000e+00, 3.000e+00, 2.000e+00, 1.000e+00,
       2.000e+00]), array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20]), [<matplotlib.patches.Polygon object at 0x16d0be940>])
```

# Homework 7 Review

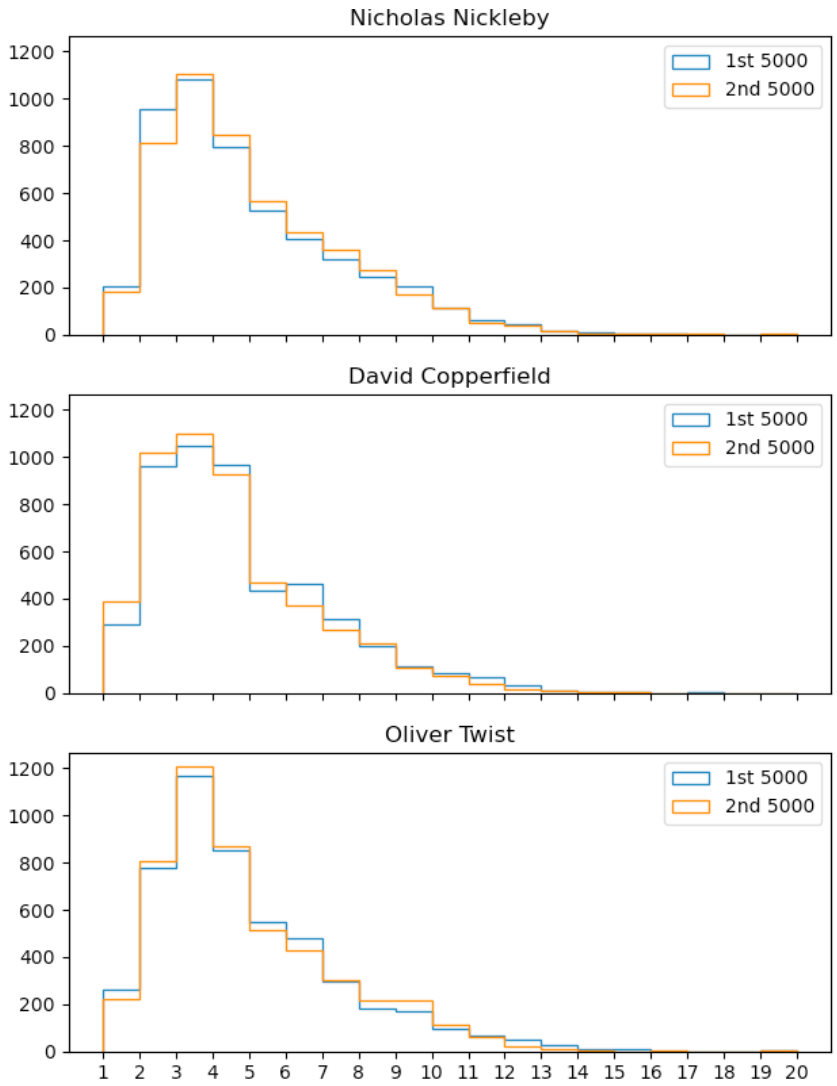
Step 7: histogram plot them with overlay

```
>>> axs2.hist(ldc1, range(1,mx+1), histtype='step', label='1st 5000')
(array([2.92e+02, 9.61e+02, 1.05e+03, 9.67e+02, 4.32e+02, 4.63e+02, 3.15e+02, 1.99e+02, 1.15e+02,
8.70e+01, 6.60e+01, 3.30e+01,  1.20e+01, 2.00e+00, 3.00e+00, 1.00e+00, 2.00e+00, 0.00e+00, 0.00e+00]),
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,18, 19, 20]),
[<matplotlib.patches.Polygon object at 0x16d0be7f0>])
>>> axs2.hist(ldc2, range(1,mx+1), histtype='step', label='2nd 5000')
(array([ 388., 1016., 1097.,  929.,  468.,  373.,  268.,  210.,  106.,   75.,   41.,   14.,   10.,    3.,    2.,    0.,    0.,    0.,  0.]),
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]), [<matplotlib.patches.Polygon object at
0x16d0beac0>])
>>> axs3.hist(lot1, range(1,mx+1), histtype='step', label='1st 5000')
(array([2.610e+02, 7.790e+02, 1.165e+03, 8.540e+02, 5.500e+02, 4.780e+02,  2.990e+02, 1.810e+02,
1.720e+02, 9.800e+01, 6.800e+01, 4.800e+01, 2.700e+01, 8.000e+00, 8.000e+00, 1.000e+00, 0.000e+00,
0.000e+00,3.000e+00]),  array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,18, 19,
20]), [<matplotlib.patches.Polygon object at 0x16d0be910>])
>>> axs3.hist(lot2, range(1,mx+1), histtype='step', label='2nd 5000')
(array([ 221.,  806.,
1205.,  872.,  514.,  429.,  302.,  218.,  215.,114.,   63.,   21.,   11.,    5.,    0.,    2.,    0.,
0.,2.]), array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,18, 19, 20]),
[<matplotlib.patches.Polygon object at 0x16d2703d0>])
```

# Homework 7 Review

## Step 7: histogram plot them with overlay

```
>>> axs1.set_xticks(range(1,mx+1))
[<matplotlib.axis.XTick object at 0x16d3e3a60>, <matplotlib.axis.XTick object at 0x16d3e3a30>,
<matplotlib.axis.XTick object at 0x16d3e3910>, <matplotlib.axis.XTick object at 0x16d270700>, <matplotlib.axis.XTick
object at 0x16d2700d0>, <matplotlib.axis.XTick object at 0x16d275310>, <matplotlib.axis.XTick object at
0x16d275a60>, <matplotlib.axis.XTick object at 0x16d27c1f0>, <matplotlib.axis.XTick object at 0x16d27c940>,
<matplotlib.axis.XTick object at 0x16d281160>, <matplotlib.axis.XTick object at 0x16d27cbb0>, <matplotlib.axis.XTick
object at 0x16d275d30>, <matplotlib.axis.XTick object at 0x16d281850>, <matplotlib.axis.XTick object at
0x16d281e80>, <matplotlib.axis.XTick object at 0x16d287610>, <matplotlib.axis.XTick object at 0x16d287d60>,
<matplotlib.axis.XTick object at 0x16d28f4f0>, <matplotlib.axis.XTick object at 0x16d2878b0>, <matplotlib.axis.XTick
object at 0x16d270f10>, <matplotlib.axis.XTick object at 0x16d28f250>]
>>> axs1.legend()
<matplotlib.legend.Legend object at 0x16b790d90>
>>> axs2.legend()
<matplotlib.legend.Legend object at 0x16d298520>
>>> axs3.legend()
<matplotlib.legend.Legend object at 0x16d27c1c0>
>>> plt.show()
```

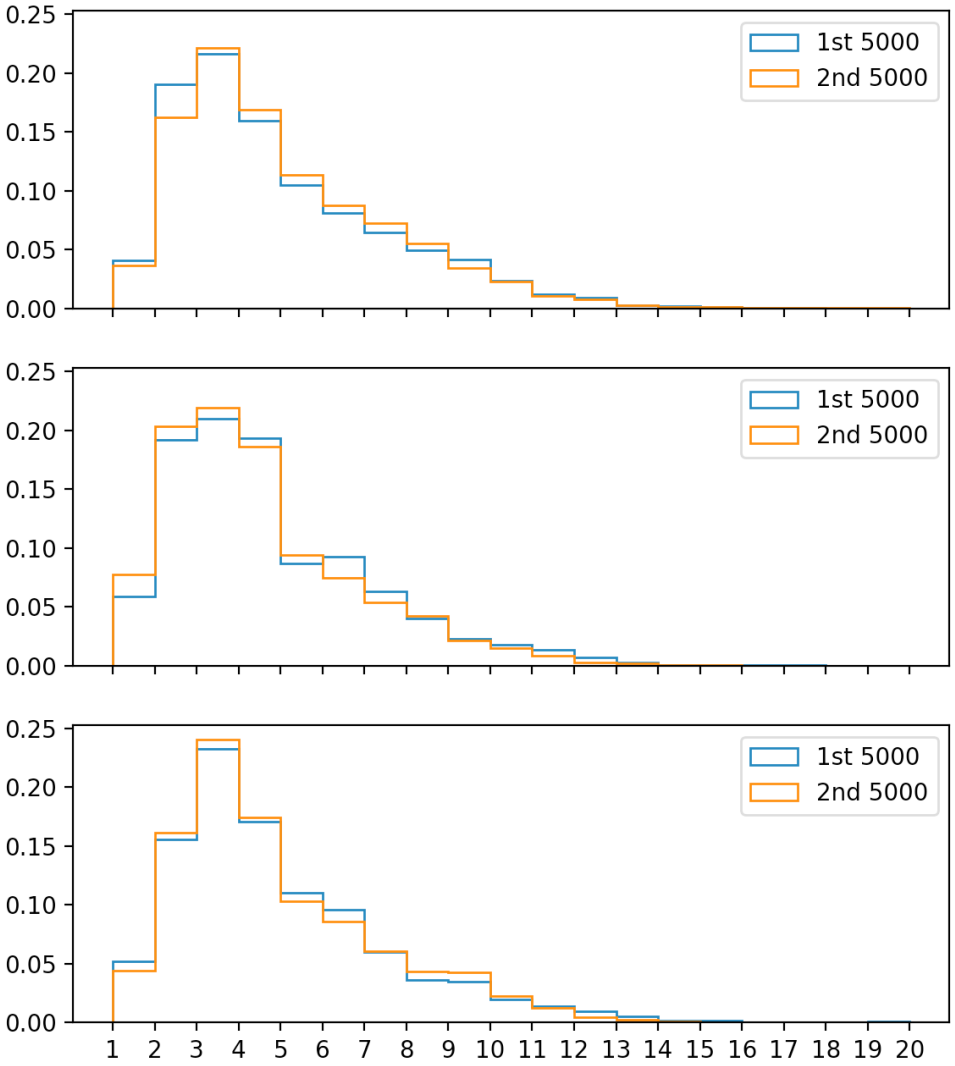

Nicholas Nickleby
1st 5000
2nd 5000
0
200
400
600
800
1000
1200
David Copperfield
1st 5000
2nd 5000
0
200
400
600
800
1000
1200
Oliver Twist
1st 5000
2nd 5000
0
200
400
600
800
1000
1200
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

# Homework 7 Review

- density=True

**density** : *bool, default: False*

If `True`, draw and return a probability density: each bin will display the bin's raw count divided by the total number of counts *and the bin width* (`density = counts / (sum(counts) * np.diff(bins))`), so that the area under the histogram integrates to 1 (`np.sum(density * np.diff(bins)) == 1`).



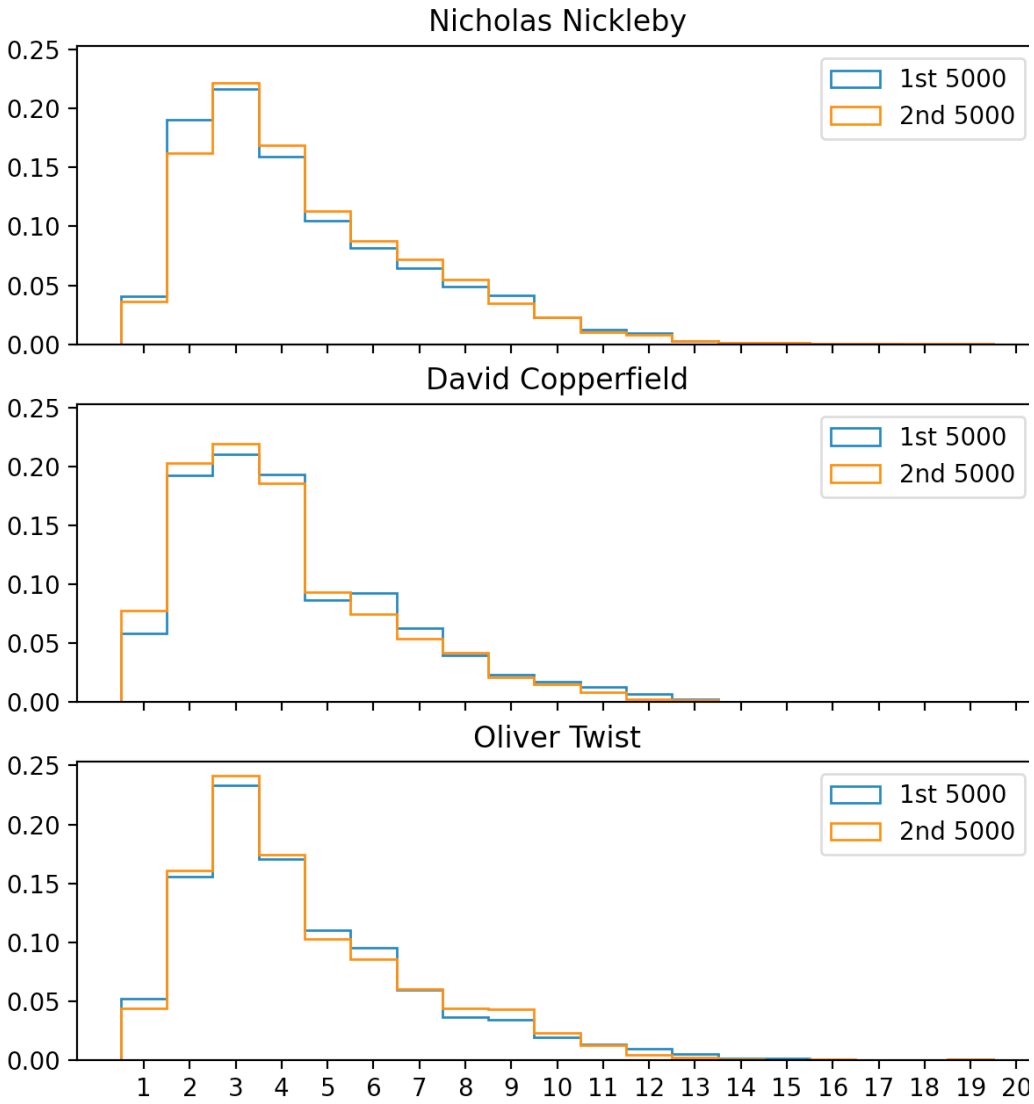
# Homework 7 Review

- align='left'

**align** : *{'left', 'mid', 'right'}, default: 'mid'*

The horizontal alignment of the histogram bars.

- 'left': bars are centered on the left bin edges.
- 'mid': bars are centered between the bin edges.
- 'right': bars are centered on the right bin edges.



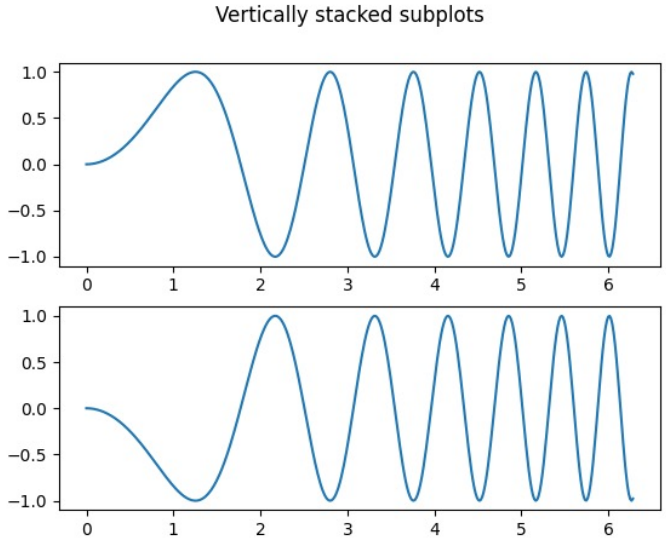
https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html

## Stacking subplots in one direction

The first two optional arguments of `pyplot.subplots` define the number of rows and columns of the subplot grid.

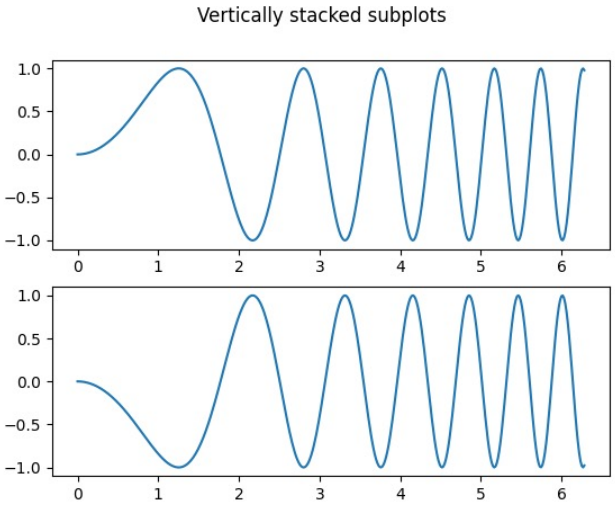
When stacking in one direction only, the returned `axs` is a 1D numpy array containing the list of created Axes.

```
fig, axs = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')
axs[0].plot(x, y)
axs[1].plot(x, -y)
```



If you are creating just a few Axes, it's handy to unpack them immediately to dedicated variables for each Axes. That way, we can use `ax1` instead of the more verbose `axs[0]`.

```
fig, (ax1, ax2) = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')
ax1.plot(x, y)
ax2.plot(x, -y)
```



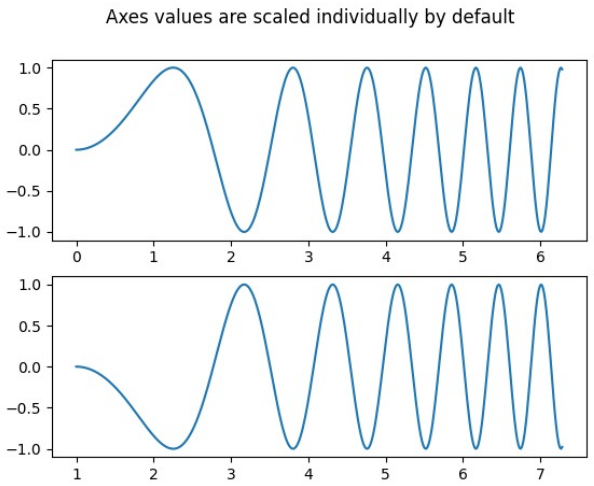

To obtain side-by-side subplots, pass parameters `1, 2` for one row and two columns.

```
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Horizontally stacked subplots')
ax1.plot(x, y)
ax2.plot(x, -y)
```

# Sharing axes

By default, each Axes is scaled individually. Thus, if the ranges are different the tick values of the subplots do not align.

```
fig, (ax1, ax2) = plt.subplots(2)
fig.suptitle('Axes values are scaled individually by default')
ax1.plot(x, y)
ax2.plot(x + 1, -y)
```




You can use *sharex* or *sharey* to align the horizontal or vertical axis.

```
fig, (ax1, ax2) = plt.subplots(2, sharex=True)
fig.suptitle('Aligning x-axis using sharex')
ax1.plot(x, y)
ax2.plot(x + 1, -y)
```

On this page

# Homework 7

- Part 2:
  - Based on your three-way comparison, what do you think about Mendenhall's claim about six-letter words for Charles Dickens? Is it justified? Explain.

When the number of words in a group is increased to five thousand, the accidental irregularities begin to disappear, the curve becomes smoother, approximating more nearly to the normal curve which, it is assumed, is characteristic of the writer. Fig. 4 exhibits two groups, each of five thousand words, from 'Oliver Twist,' and it will be seen that considerable differences still exist. One of the curves shows an excess of nine-letter words, which does not appear in the other. They agree in showing a greater number of six-letter words than a smooth curve would demand. This excess may persist, and prove to be a real characteristic of Dickens's composition.

# Homework 7

- Part 3:
  - Now take (slice) the first 100,000 words for each of *Oliver Twist* (1838) with *Nicholas Nickleby* (1839) and *David Copperfield* (1850).
  - Plot them over one another.
  - Submit your histogram and code.
  - What do you think of Mendenhall 100,000 word claim?

From the examinations thus far made, I am convinced that one hundred thousand words will be necessary and sufficient to furnish the characteristic curve of a writer, — that is to say, if a curve is constructed from one hundred thousand words of a writer, taken from any one of his productions, then a second curve constructed from another hundred thousand words would be practically identical with the first, — and that this curve would, in general, differ from that formed in the same way from the composition of another writer, to such an extent that one could always be distinguished from the other. To demonstrate the

though not probable, that two writers might show identical characteristic curves.

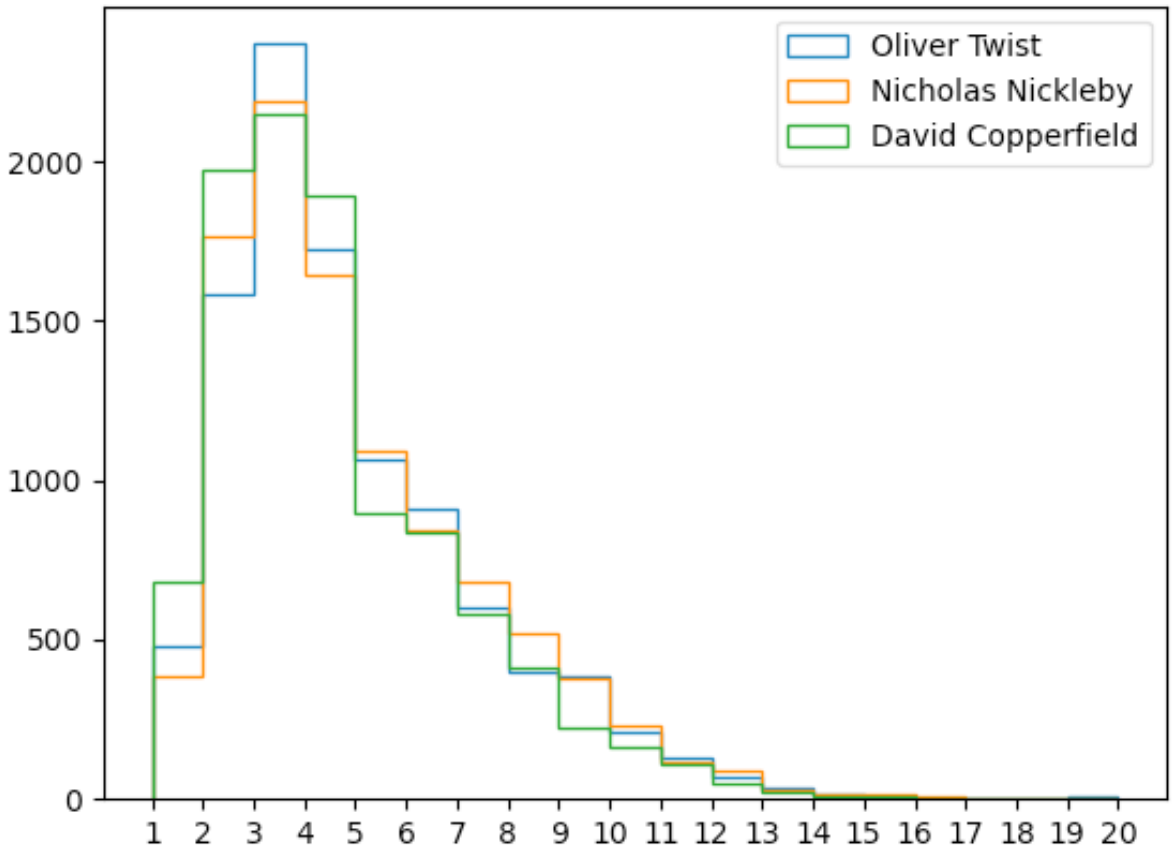
T. C. MENDENHALL.

# Homework 7 Review

```
>>> lot = chunk(ot2, 1, 10000)
>>> len(lot)
10000
>>> lnn = chunk(nn2, 1, 10000)
>>> ldc = chunk(dc2, 1, 10000)
>>> plt.hist(lot, range(1,mx+1), histtype='step', label='Oliver Twist')
(array([ 482., 1585., 2370., 1726.,
1064.,  907.,  601.,  399.,  387.,  212.,  131.,   69.,   38.,   13.,    8.,    3.,    0.,    0.,    5.])
, array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,  18, 19, 20]),
[<matplotlib.patches.Polygon object at 0x16d2d0550>])
>>> plt.hist(lnn, range(1,mx+1), histtype='step', label='Nicholas Nickleby')
(array([3.880e+02, 1.766e+03, 2.189e+03, 1.642e+03, 1.090e+03, 8.440e+02, 6.830e+02, 5.210e+02,
3.790e+02, 2.310e+02, 1.130e+02, 8.700e+01,2.900e+01, 1.400e+01, 1.200e+01, 6.000e+00, 3.000e+00,
1.000e+00,2.000e+00]), array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,18,
19, 20]), [<matplotlib.patches.Polygon object at 0x16d2d09a0>])
>>> plt.hist(ldc, range(1,mx+1), histtype='step', label='David Copperfield')
(array([6.800e+02, 1.977e+03, 2.147e+03, 1.896e+03, 9.000e+02, 8.360e+02,  5.830e+02, 4.090e+02,
2.210e+02, 1.620e+02, 1.070e+02, 4.700e+01, 2.200e+01, 5.000e+00, 5.000e+00, 1.000e+00, 2.000e+00,
0.000e+00,0.000e+00]), array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,18,
19, 20]), [<matplotlib.patches.Polygon object at 0x16d2d0af0>])
```

# Homework 7 Review

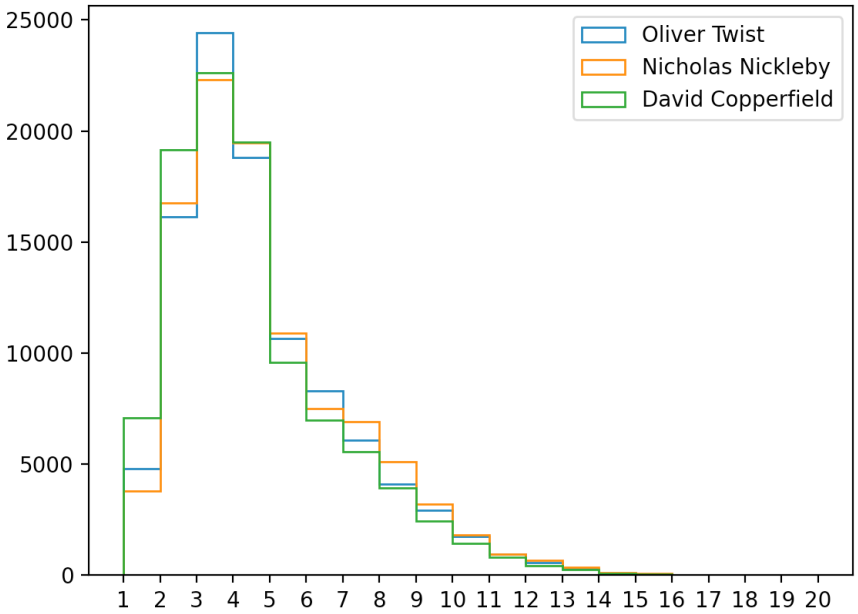
```
>>> mx = max(max(lot),max(lnn),max(ldc))
>>> mx
20
>>> plt.xticks(range(1,mx+1))
([<matplotlib.axis.XTick object at 0x16d2a3ee0>, <matplotlib.axis.XTick object at 0x16d2a3eb0>,
<matplotlib.axis.XTick object at 0x16d2a3d90>, <matplotlib.axis.XTick object at 0x16d2da040>,
<matplotlib.axis.XTick object at 0x16d2da640>, <matplotlib.axis.XTick object at 0x16d2dad90>,
<matplotlib.axis.XTick object at 0x172dfc520>, <matplotlib.axis.XTick object at 0x16d2dadf0>,
<matplotlib.axis.XTick object at 0x16d2d0a30>, <matplotlib.axis.XTick object at 0x172e05040>,
<matplotlib.axis.XTick object at 0x172e05640>, <matplotlib.axis.XTick object at 0x172e05d90>,
<matplotlib.axis.XTick object at 0x172e0a520>, <matplotlib.axis.XTick object at 0x172e0ac70>,
<matplotlib.axis.XTick object at 0x172e0aa00>, <matplotlib.axis.XTick object at 0x172e05700>,
<matplotlib.axis.XTick object at 0x172e372e0>, <matplotlib.axis.XTick object at 0x172e37910>,
<matplotlib.axis.XTick object at 0x172e350a0>, <matplotlib.axis.XTick object at 0x172e357f0>],
[Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''),
Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''),
Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, ''),
Text(0, 0, ''), Text(0, 0, '')])
>>> plt.legend()
<matplotlib.legend.Legend object at 0x16d2c1400>
>>> plt.show()
```


Oliver Twist
Nicholas Nickleby
David Copperfield
2000
1500
1000
500
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

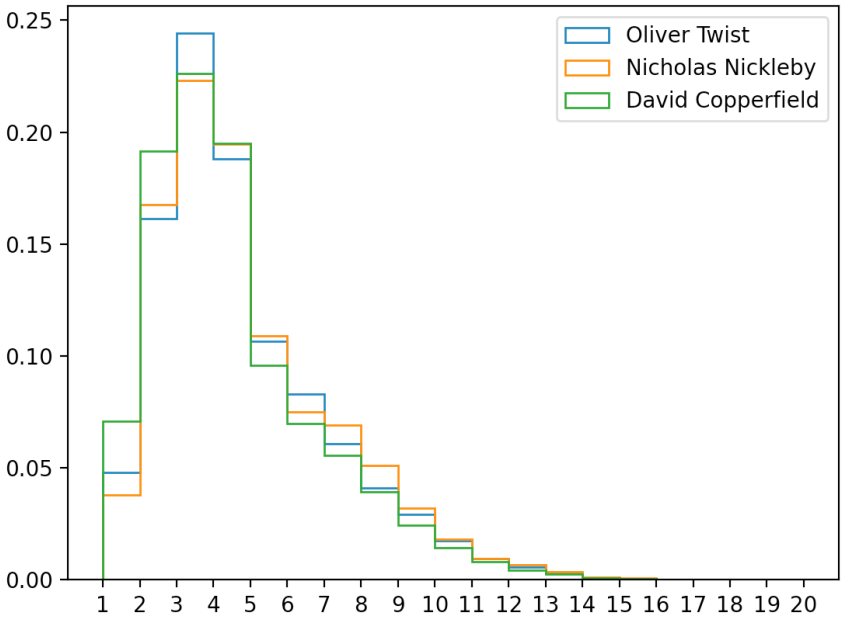
# Homework 7 Review

```
>>> lot = chunk(ot2,1, 100000)
>>> lnn = chunk(nn2,1, 100000)
>>> ldc = chunk(dc2,1, 100000)
>>> plt.hist(lot,range(1,mx+1), histtype='step', label='Oliver
Twist')
>>> plt.hist(lnn, range(1,mx+1), histtype='step', label='Nicholas
Nickleby')
>>> plt.hist(ldc, range(1,mx+1), histtype='step', label='David
Copperfield')
       18, 19, 20]), [<matplotlib.patches.Polygon object at
0x16eedb9a0>])
>>> plt.xticks(range(1,mx+1))
>>> plt.legend()
>>> plt.show()
```

# Homework 7 Review




`density=True`



# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

```
>>> import nltk
>>> from nltk.corpus import genesis
>>> genesis
<PlaintextCorpusReader in '../corpora/genesis' (not loaded yet)>
>>> g = genesis.words('english-kjv.txt')
>>> len(g)
44764
>>> g[:20]
['In', 'the', 'beginning', 'God', 'created', 'the', 'heaven',
'and', 'the', 'earth', '.', 'And', 'the', 'earth', 'was',
'without', 'form', ',', 'and', 'void']
```

# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

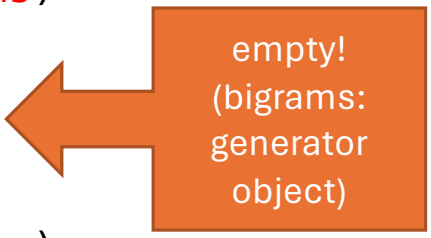
```
>>> bigrams = nltk.bigrams(g)
>>> bigrams
<generator object bigrams at 0x11d540200>
>>> l = list(bigrams)
>>> len(l)
44763
>>> l[:20]
[('In', 'the'), ('the', 'beginning'), ('beginning', 'God'), ('God',
'created'), ('created', 'the'), ('the', 'heaven'), ('heaven', 'and'),
('and', 'the'), ('the', 'earth'), ('earth', '.'), ('.', 'And'), ('And',
'the'), ('the', 'earth'), ('earth', 'was'), ('was', 'without'),
('without', 'form'), ('form', ','), (',', 'and'), ('and', 'void'),
('void', ';')]
```

# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

- `nltk.ConditionalFreqDist()` can take as input a bigram corpus
  - actually, a list of (*word1*, *word2*) tuples such that *word1* immediately precedes *word2*.

```
>>> cfd = nltk.ConditionalFreqDist(bigrams)
>>> cfd
<ConditionalFreqDist with 0 conditions>
>>> bigrams = nltk.bigrams(g)
>>> cfd = nltk.ConditionalFreqDist(bigrams)
>>> cfd
<ConditionalFreqDist with 2789 conditions>
```



# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

- Unigrams (= list of words):

```
['In', 'the', 'beginning', 'God', 'created', 'the',
'heaven', 'and', 'the', 'earth', '.', 'And', 'the',
'earth', 'was', 'without', 'form', ',', 'and', 'void']
```

- Bigrams (= list of tuples (word1,word2) *etc*.):
  - `[('In', 'the'), ('the', 'beginning'), ('beginning', 'God'), ('God', 'created'), ('created', 'the'), …]`
- `cfd[word]` gives a count dict of which words can follow `word`:

```
>>> cfd['In']
FreqDist({'the': 9, 'my': 2, 'thee': 1})
```

# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

- `cfd[word]` gives a count dict of which words can follow `word`:

```
>>> cfd['The']
FreqDist({'sons': 5, 'children': 4, 'LORD': 4, 'man': 3, 'days': 2, 'three': 2,
'name': 1, 'woman': 1, 'serpent': 1, 'earth': 1, ...})
>>> cfd['the']
FreqDist({'land': 156, 'LORD': 154, 'earth': 105, 'sons': 69, 'name': 42,
'field': 39, 'men': 36, 'man': 34, 'waters': 30, 'children': 30, ...})
```

- `cfd[word].max()` tells us what word is most likely to follow `word`:

```
>>> cfd['the'].max()
'land'
>>> cfd['The'].max()
'sons'
```

# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

- `cfd[word]` gives a count dict of which words can follow `word`:

```
>>> cfd['The']
FreqDist({'sons': 5, 'children': 4, 'LORD': 4, 'man': 3, 'days': 2, 'three': 2,
'name': 1, 'woman': 1, 'serpent': 1, 'earth': 1, ...})
>>> cfd['the']
FreqDist({'land': 156, 'LORD': 154, 'earth': 105, 'sons': 69, 'name': 42,
'field': 39, 'men': 36, 'man': 34, 'waters': 30, 'children': 30, ...})
```

- `cfd[word].N()` tells us the size (total # words in the corpus) that can follow `word`:

```
>>> cfd['the'].N()
2411
>>> cfd['The'].N()
50
```

# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

- `cfd[word]` gives a count dict (aka nltk. FreqDist) of which words can follow `word`:

```
>>> cfd['The']
FreqDist({'sons': 5, 'children': 4, 'LORD': 4, 'man': 3, 'days': 2, 'three': 2, 'name': 1,
'woman': 1, 'serpent': 1, 'earth': 1, ...})
```

- `cfd[word].keys()` gives the keys to the count dict; i.e. all the words that can follow `word`:

```
>>> cfd['The'].keys()
dict_keys(['name', 'woman', 'serpent', 'earth', 'end', 'length', 'fountains', 'sons',
'children', 'princes', 'Kenites', 'sun', 'LORD', 'thing', 'field', 'water', 'voice',
'days', 'speckled', 'ringstraked', 'God', 'soul', 'child', 'Hebrew', 'keeper', 'three',
'dream', 'seven', 'man', 'lad', 'land', 'Angel','sceptre', 'archers', 'blessings',
'purchase'])
```

- `cfd[word][word2]` gives the number of times `word2` can follow `word`:

```
>>> cfd['The']['name']
1
>>> cfd['The']['sons']
5
```

# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

- `cfd[word]` gives a count dict (aka nltk. FreqDist) of which words can follow `word`:

```
>>> cfd['The']
FreqDist({'sons': 5, 'children': 4, 'LORD': 4, 'man': 3, 'days': 2,
'three': 2, 'name': 1, 'woman': 1, 'serpent': 1, 'earth': 1, ...})
```

- list comprehension creates a (sub-)corpus for `'The'`:

```
>>> [k for k in cfd['The'].keys() for i in range(cfd['The'][k])]
['name', 'woman', 'serpent', 'earth', 'end', 'length', 'fountains', 'sons', 'sons', 'sons', 'sons',
'sons', 'children', 'children', 'children', 'children', 'princes', 'Kenites', 'sun', 'LORD', 'LORD',
'LORD', 'LORD', 'thing', 'field', 'water', 'voice', 'days', 'days', 'speckled', 'ringstraked', 'God',
'soul', 'child', 'Hebrew', 'keeper', 'three', 'three', 'dream', 'seven', 'man', 'man', 'man', 'lad',
'land', 'Angel', 'sceptre', 'archers', 'blessings', 'purchase']
```

- More generally, for word `w`:

```
[k for k in cfd[w].keys() for i in range(cfd[w][k])]
```

# NLTK Book: Chapter 2

## 2.4 Generating Random Text with Bigrams

- Algorithm for randomly generating the next word after `'The'`:
  1. Randomly pick a word from the distribution of possible following words
  2. Repeat with the selected word, i.e. chain word after word
- Example
  - Start with 'The', randomly pick from here:
  - ['name', 'woman', 'serpent', 'earth', 'end', 'length', 'fountains', 'sons', 'sons', 'sons', 'sons', 'sons', 'children', 'children', 'children', 'children', 'princes', 'Kenites', 'sun', 'LORD', 'LORD', 'LORD', 'LORD', 'thing', 'field', 'water', 'voice', 'days', 'days', 'speckled', 'ringstraked', 'God', 'soul', 'child', 'Hebrew', 'keeper', 'three', 'three', 'dream', 'seven', 'man', 'man', 'man', 'lad', 'land', 'Angel', 'sceptre', 'archers', 'blessings', 'purchase']
  - `>>> [k for k in cfd['The'].keys() for i in range(cfd['The'][k])]`
  - Next, use 'children' and randomly pick from here:
  - `>>> [k for k in cfd['children'].keys() for i in range(cfd['children'][k])]`
  - [';', ';', 'to', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'of', 'born', ':', 'by', 'by', 'and', '.', '.', 'suck', 'unto', 'unto', 'struggled', ',', ',', ',', ',', ',', ',', ',', ',', ',', ',', ',', 'now', 'are', 'are', 'which', 'which', 'foremost', 'after', 'came', 'be', '"', 'shall', 'also']
  - Next, use 'of' and so on…

# NLTK Book: Chapter 2

```
from random import choice
def next_word(w):
    return choice([k for k in cfd[w].keys() for i in range(cfd[w][k])])
def nwords(n, w):
    for i in range(n):
        print(w, end=' ')
        w = next_word(w)
    print()
>>> nwords(10,'The')
The Kenites , and it was not to the children
>>> nwords(10,'The')
The keeper of beasts I wrestled with this night ;
>>> nwords(10,'The')
The water ye rewarded evil beast of your occupation ?
```

# `nltk.ConditionalFreqDist()`

## Summary:

- `nltk.ConditionalFreqDist()` takes as input a bigram corpus (actually a list of (word1,word2) etc. tuples) produced by `nltk.bigrams(`*`text`*`)`
- `cfd = nltk.ConditionalFreqDist(nltk.bigrams(`*`text`*`))`
- `cfd[`*`word1`*`]` returns a `FreqDist` for *word2*.
- `cfd[`*`word1`*`].max()` tells us what word is most likely to follow *word1*.
- `cfd[`*`word1`*`].N()` tells us the size (total # words in the corpus) that can follow *word1*.
- `cfd[`*`word1`*`].keys()` gives the keys to the count dict; i.e. list of words that can follow *word1*.
- `cfd[word1][word2]` gives the number of times `word2` can follow `word1`.