Supervised Learning with Nearest Neighbors

CS 780/880 Natural Language Processing Lecture 4

Samuel Carton, University of New Hampshire

Last lecture

Key idea: Vectorizing text, computing similarity

Concepts

- Preprocessing
  - Stemming
  - Tokenization
- Vectorizing
  - Bag-of-words
  - TF-IDF
- Text similarity
  - Jaccard distance
  - Cosine distance/similarity
  - Others

Toolkits

- Numpy for vectors
- NLTK for preprocessing
- Scikit-Learn for vectorizing & similarity

Supervised learning for classification

Classification: given input text x, classify x by predicting label y

- "You are an ass!" → toxic
- "The movie was great." → positive
- "SALE! SALE! SALE!" → spam
- "You are a mensch!" → nontoxic
- "The movie was awful." → negative
- "I'm breaking up with you."→ not spam

Supervised learning: given a training set X_{train} with labels Y_{train}, learn how to predict y for an unseen input x

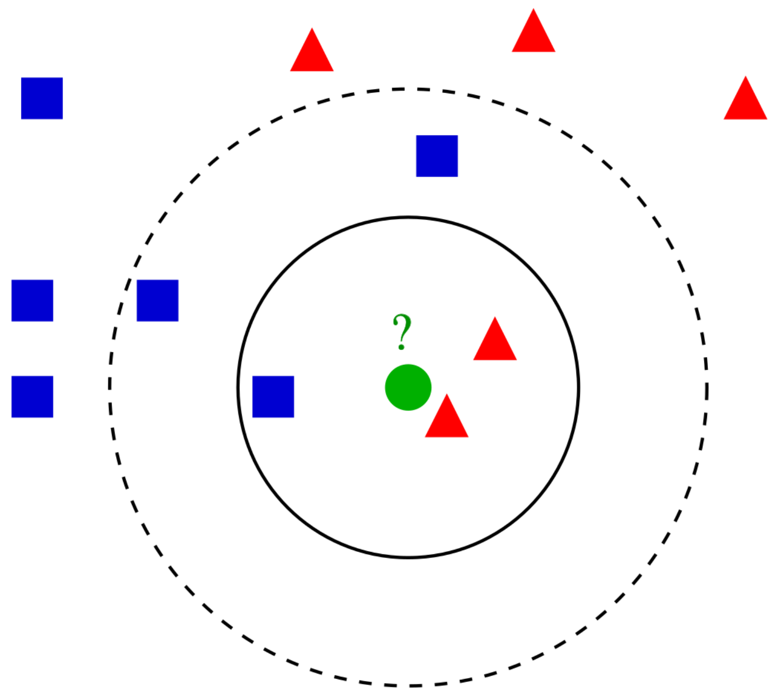
All we know how to do right now is text similarity. How to do supervised classification with just this tool?

K-nearest neighbors

Basic idea: when trying to classify x, find the K nearest neighbors of x within X_{train} and let $\hat{y}$ be the majority-vote true label y_{train} among those K neighbors

Why does it have to be K? Why not always K = 1?

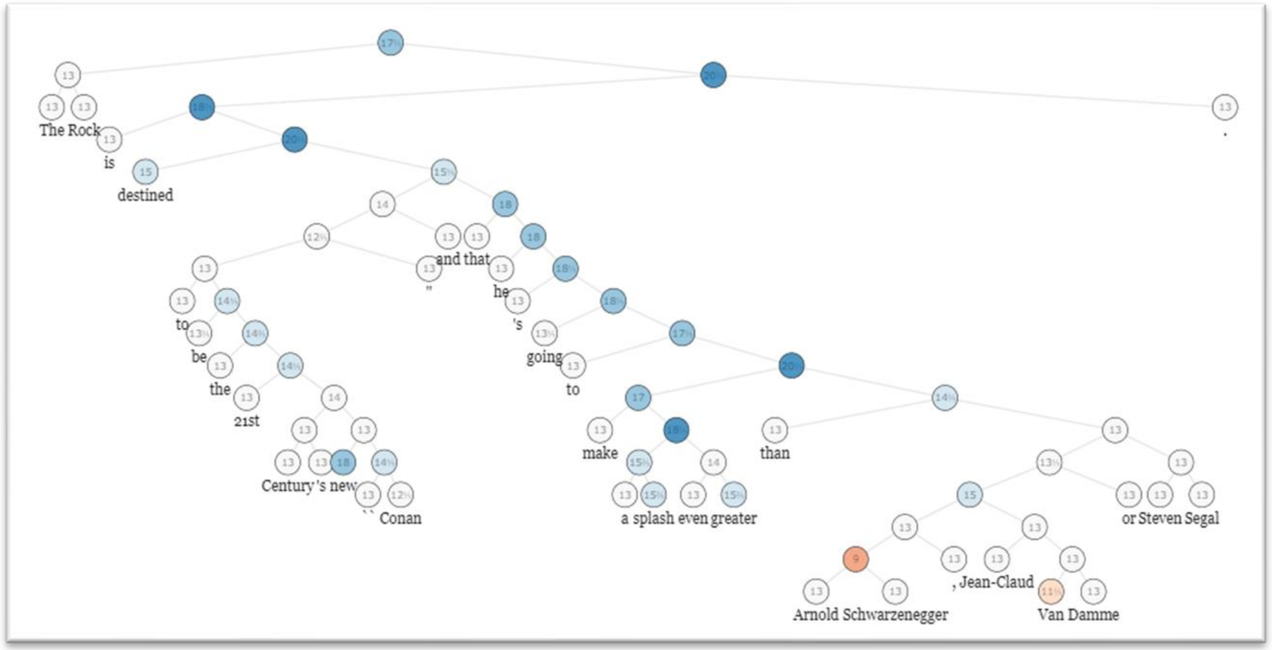
How would you implement this given what you already know?



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Case study: SST-2

- Stanford Sentiment Treebank (2-class version)
- Short movie reviews, tagged as positive or negative in sentiment
- Created by Socher et al. (2013)
- https://nlp.stanford.edu/sentiment/treebank.html
- Included as part of GLUE benchmark
  - https://gluebenchmark.com/tasks
- Size:
  - 67,349 training examples
  - 872 dev examples
  - 1821 test examples



```
the rock is destined to be the 21st century 's new `` conan ''
and that he 's going to make a splash even greater than arnold
schwarzenegger , jean-claud van damme or steven segal .
```

⬇

positive (1)

Pandas: read and manipulate data

Pandas is a useful Python library for reading and manipulating datasets of various kinds (text included)

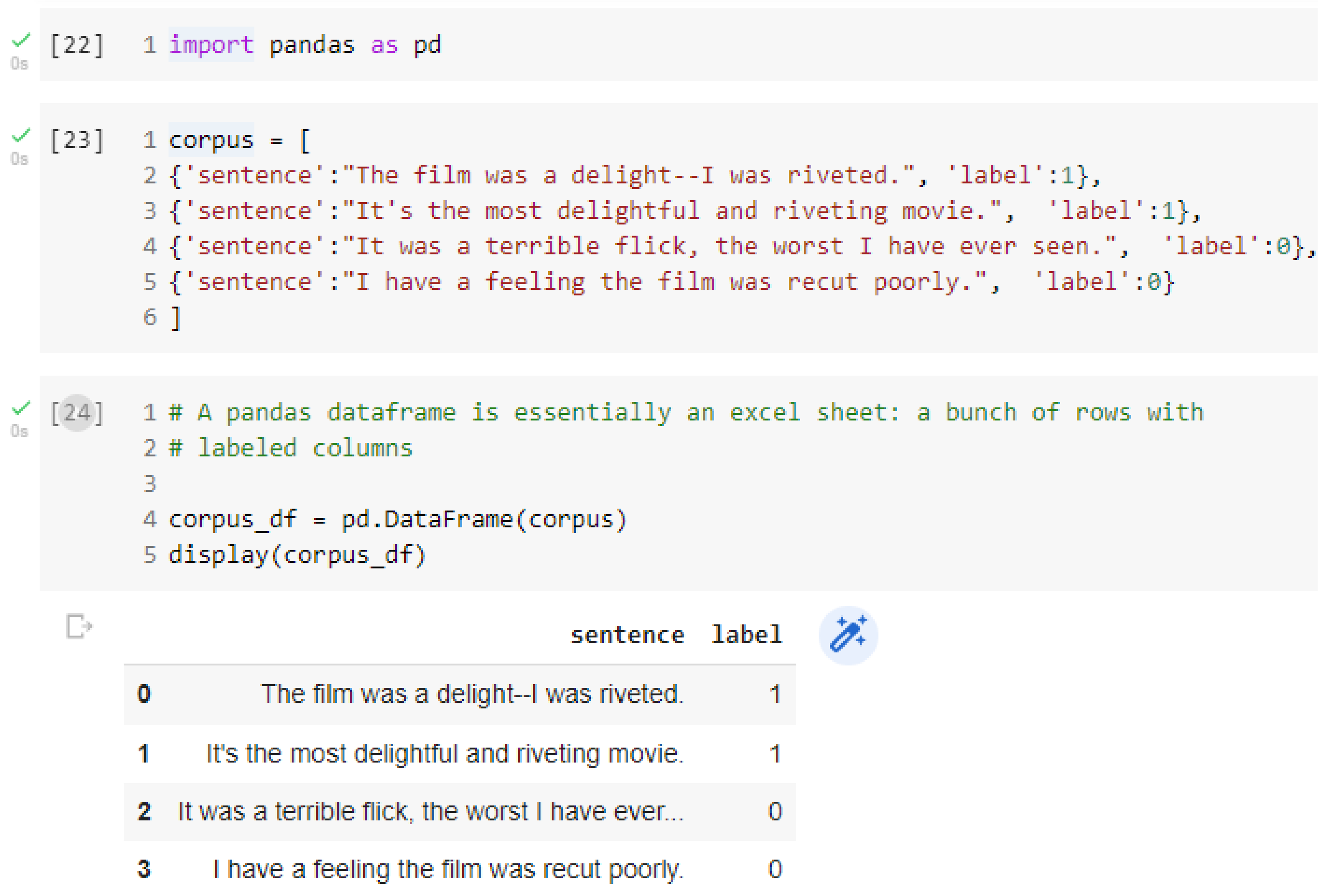
https://pandas.pydata.org/

Largely consists of an implementation of “DataFrame” from the R statistical analysis language

- Swiss army knife data structure

Likely to be covered more thoroughly in a “data science” course.

Pandas basics



[22]

```
import pandas as pd
```

[23]

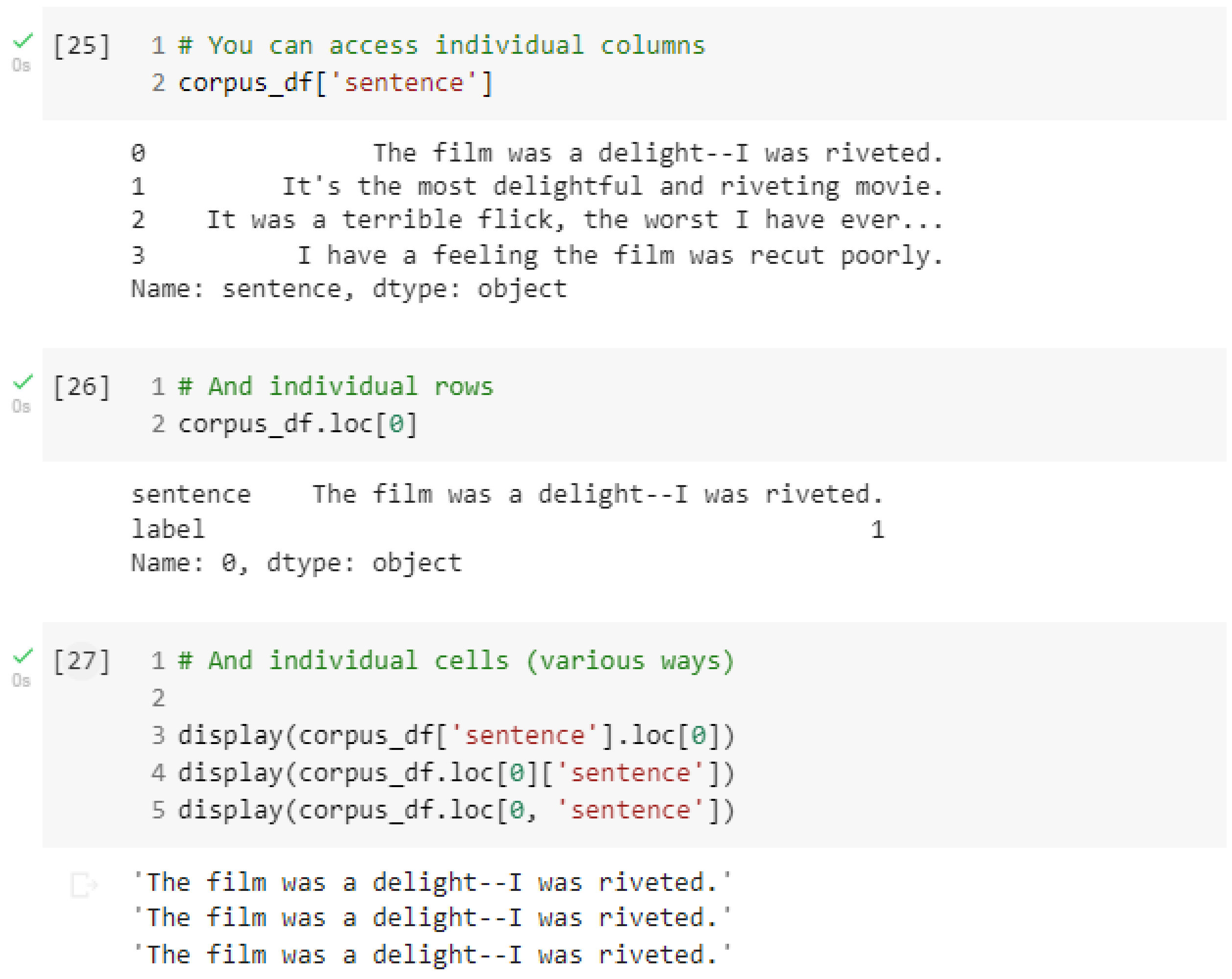
```
corpus = [
{'sentence':"The film was a delight--I was riveted.", 'label':1},
{'sentence':"It's the most delightful and riveting movie.",  'label':1},
{'sentence':"It was a terrible flick, the worst I have ever seen.",  'label':0},
{'sentence':"I have a feeling the film was recut poorly.",  'label':0}
]
```

[24]

```
# A pandas dataframe is essentially an excel sheet: a bunch of rows with
# labeled columns

corpus_df = pd.DataFrame(corpus)
display(corpus_df)
```

	sentence	label
0	The film was a delight--I was riveted.	1
1	It's the most delightful and riveting movie.	1
2	It was a terrible flick, the worst I have ever...	0
3	I have a feeling the film was recut poorly.	0

Pandas basics



[25]
```python
# You can access individual columns
corpus_df['sentence']
```

```
0              The film was a delight--I was riveted.
1          It's the most delightful and riveting movie.
2     It was a terrible flick, the worst I have ever...
3           I have a feeling the film was recut poorly.
Name: sentence, dtype: object
```

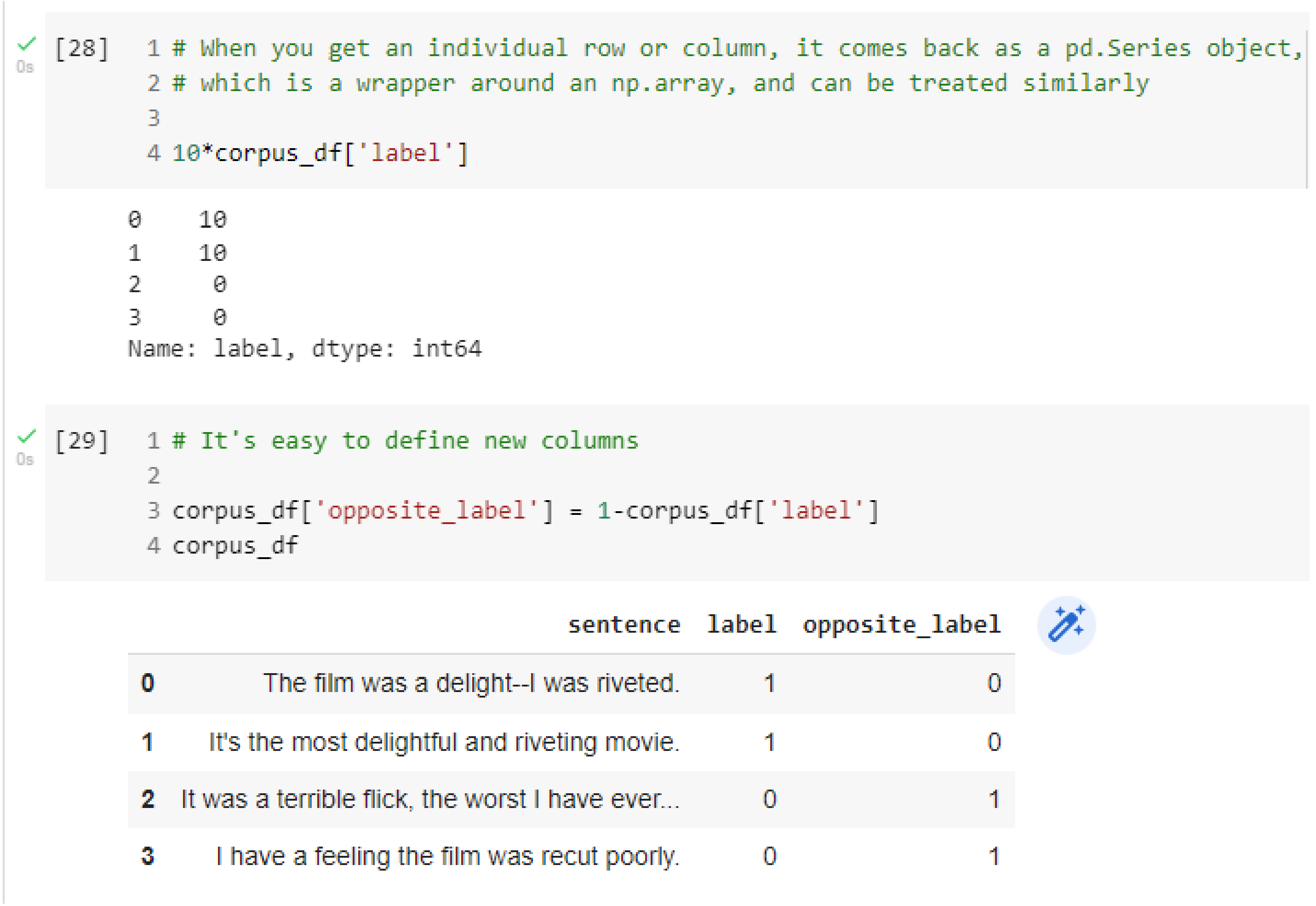
[26]
```python
# And individual rows
corpus_df.loc[0]
```

```
sentence    The film was a delight--I was riveted.
label                                            1
Name: 0, dtype: object
```

[27]
```python
# And individual cells (various ways)

display(corpus_df['sentence'].loc[0])
display(corpus_df.loc[0]['sentence'])
display(corpus_df.loc[0, 'sentence'])
```

```
'The film was a delight--I was riveted.'
'The film was a delight--I was riveted.'
'The film was a delight--I was riveted.'
```

Pandas basics



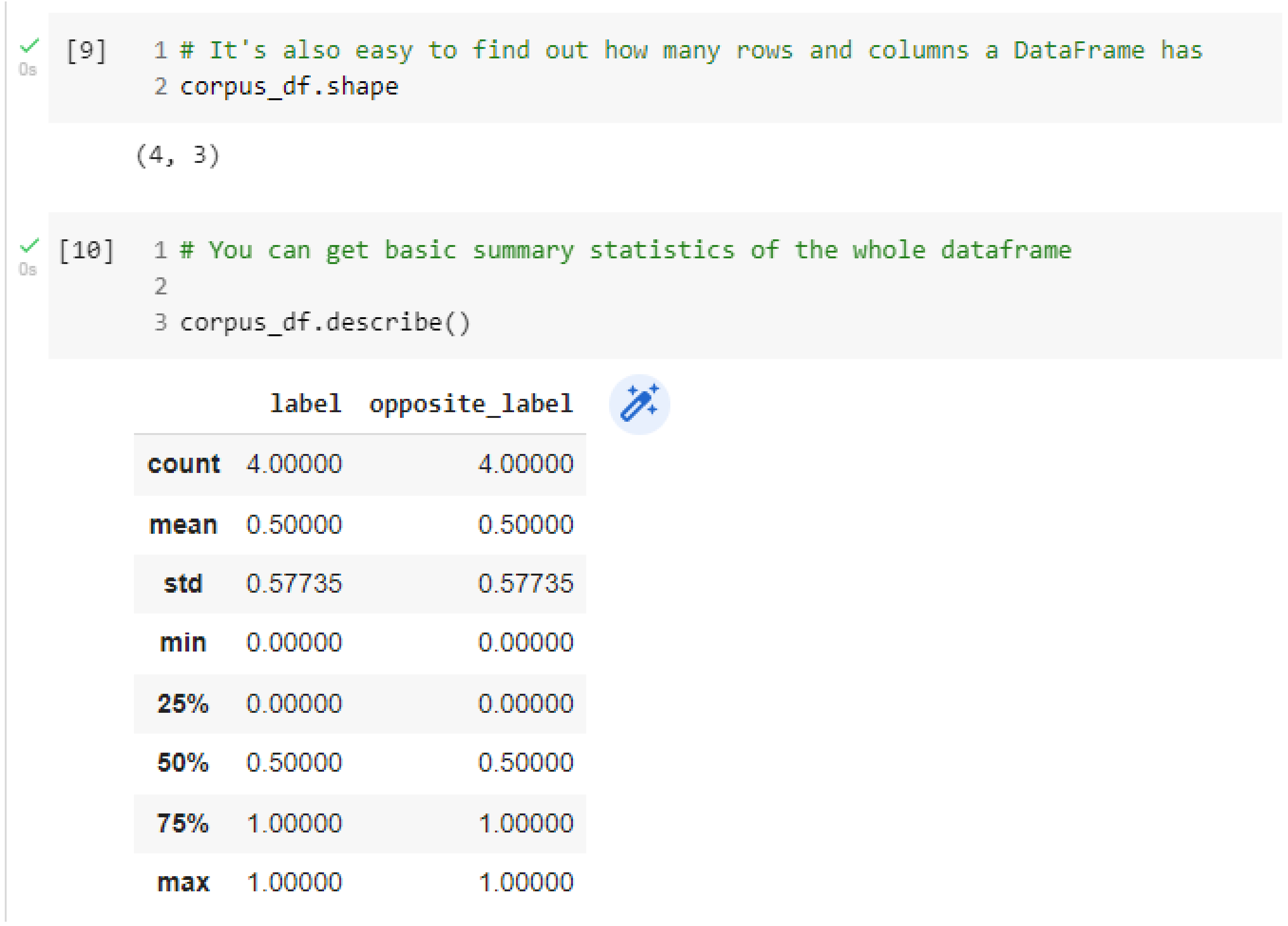
```
[28]
# When you get an individual row or column, it comes back as a pd.Series object,
# which is a wrapper around an np.array, and can be treated similarly

10*corpus_df['label']
```

```
0    10
1    10
2     0
3     0
Name: label, dtype: int64
```

```
[29]
# It's easy to define new columns

corpus_df['opposite_label'] = 1-corpus_df['label']
corpus_df
```

	sentence	label	opposite_label
0	The film was a delight--I was riveted.	1	0
1	It's the most delightful and riveting movie.	1	0
2	It was a terrible flick, the worst I have ever...	0	1
3	I have a feeling the film was recut poorly.	0	1

Pandas basics



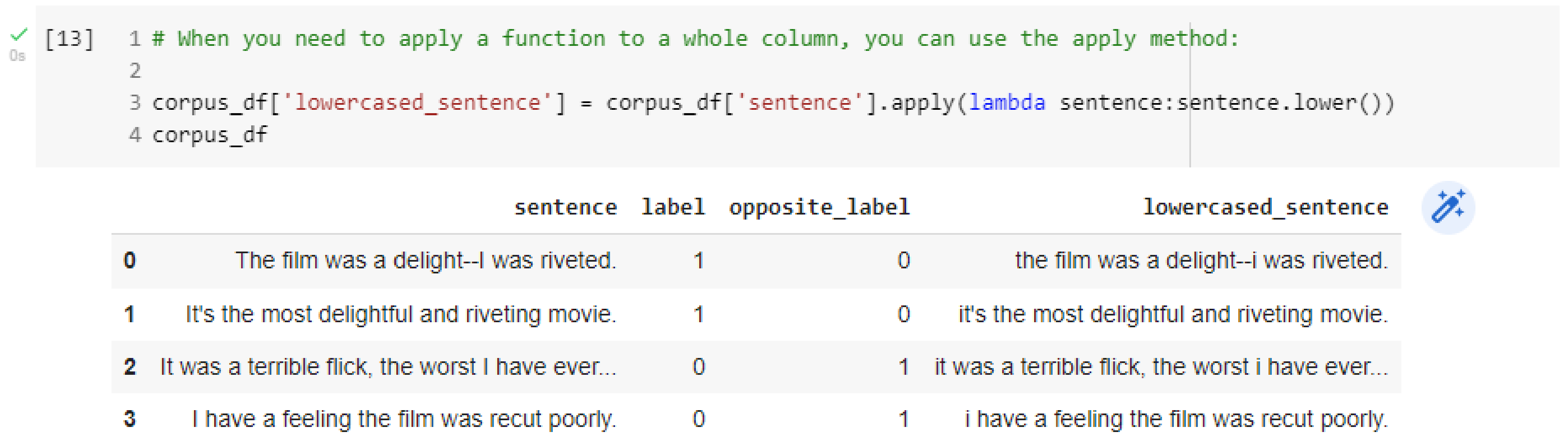
```
[9]
# It's also easy to find out how many rows and columns a DataFrame has
corpus_df.shape
```

```
(4, 3)
```

```
[10]
# You can get basic summary statistics of the whole dataframe

corpus_df.describe()
```

	label	opposite_label
count	4.00000	4.00000
mean	0.50000	0.50000
std	0.57735	0.57735
min	0.00000	0.00000
25%	0.00000	0.00000
50%	0.50000	0.50000
75%	1.00000	1.00000
max	1.00000	1.00000

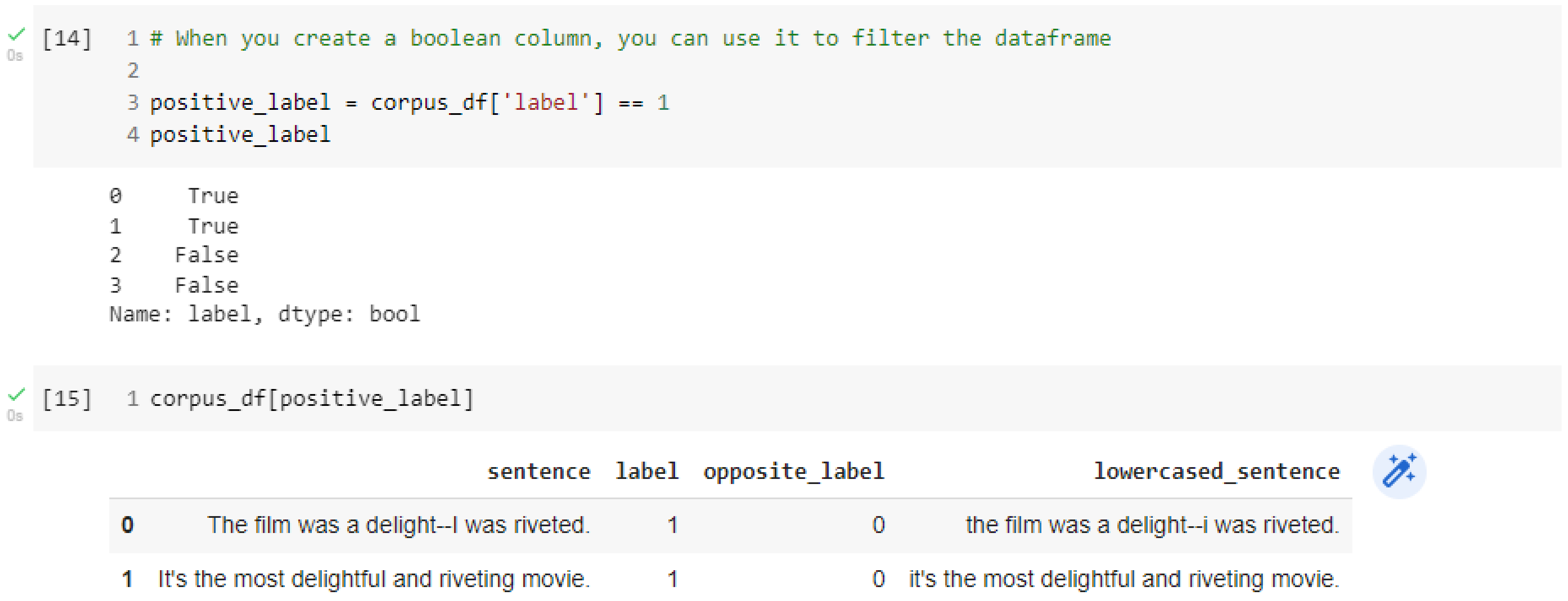
.apply() method



```
# When you need to apply a function to a whole column, you can use the apply method:

corpus_df['lowercased_sentence'] = corpus_df['sentence'].apply(lambda sentence:sentence.lower())
corpus_df
```

	sentence	label	opposite_label	lowercased_sentence
0	The film was a delight--I was riveted.	1	0	the film was a delight--i was riveted.
1	It's the most delightful and riveting movie.	1	0	it's the most delightful and riveting movie.
2	It was a terrible flick, the worst I have ever...	0	1	it was a terrible flick, the worst i have ever...
3	I have a feeling the film was recut poorly.	0	1	i have a feeling the film was recut poorly.

DataFrame filtering



[14]

```python
# When you create a boolean column, you can use it to filter the dataframe

positive_label = corpus_df['label'] == 1
positive_label
```

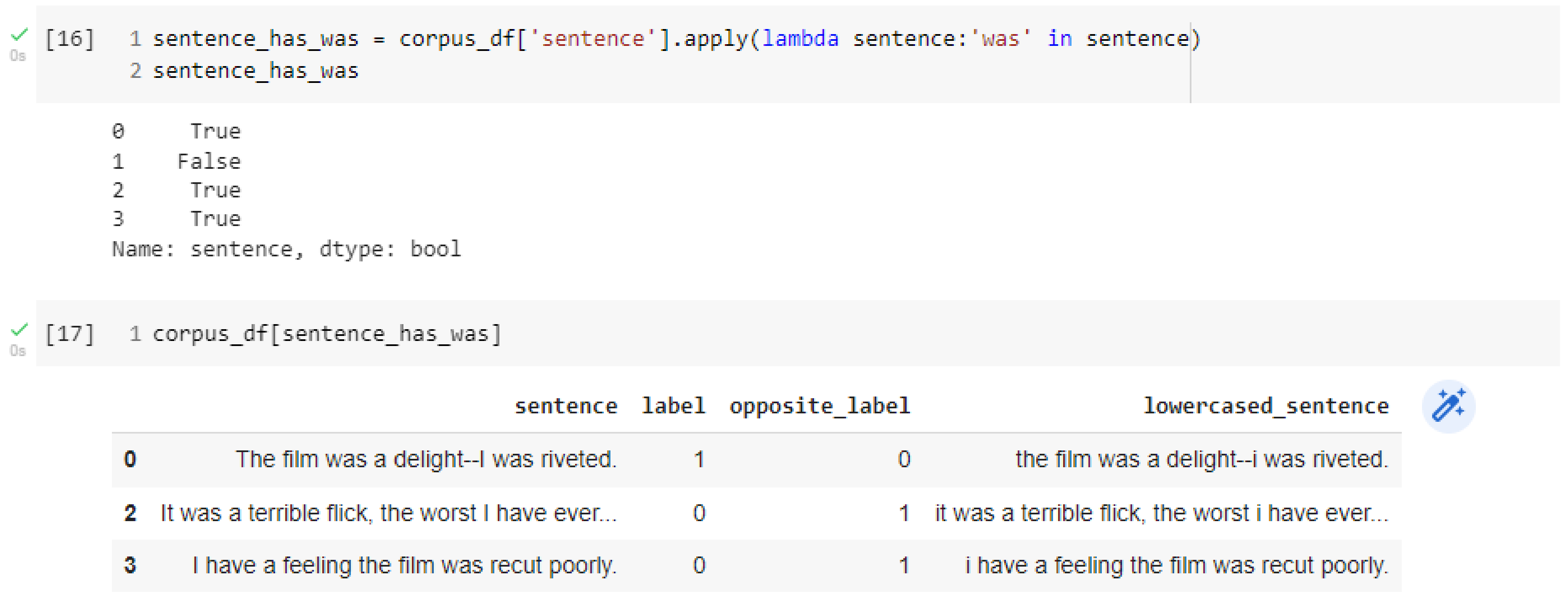
```
0     True
1     True
2    False
3    False
Name: label, dtype: bool
```

[15]

```python
corpus_df[positive_label]
```

	sentence	label	opposite_label	lowercased_sentence
0	The film was a delight--I was riveted.	1	0	the film was a delight--i was riveted.
1	It's the most delightful and riveting movie.	1	0	it's the most delightful and riveting movie.

DataFrame filtering



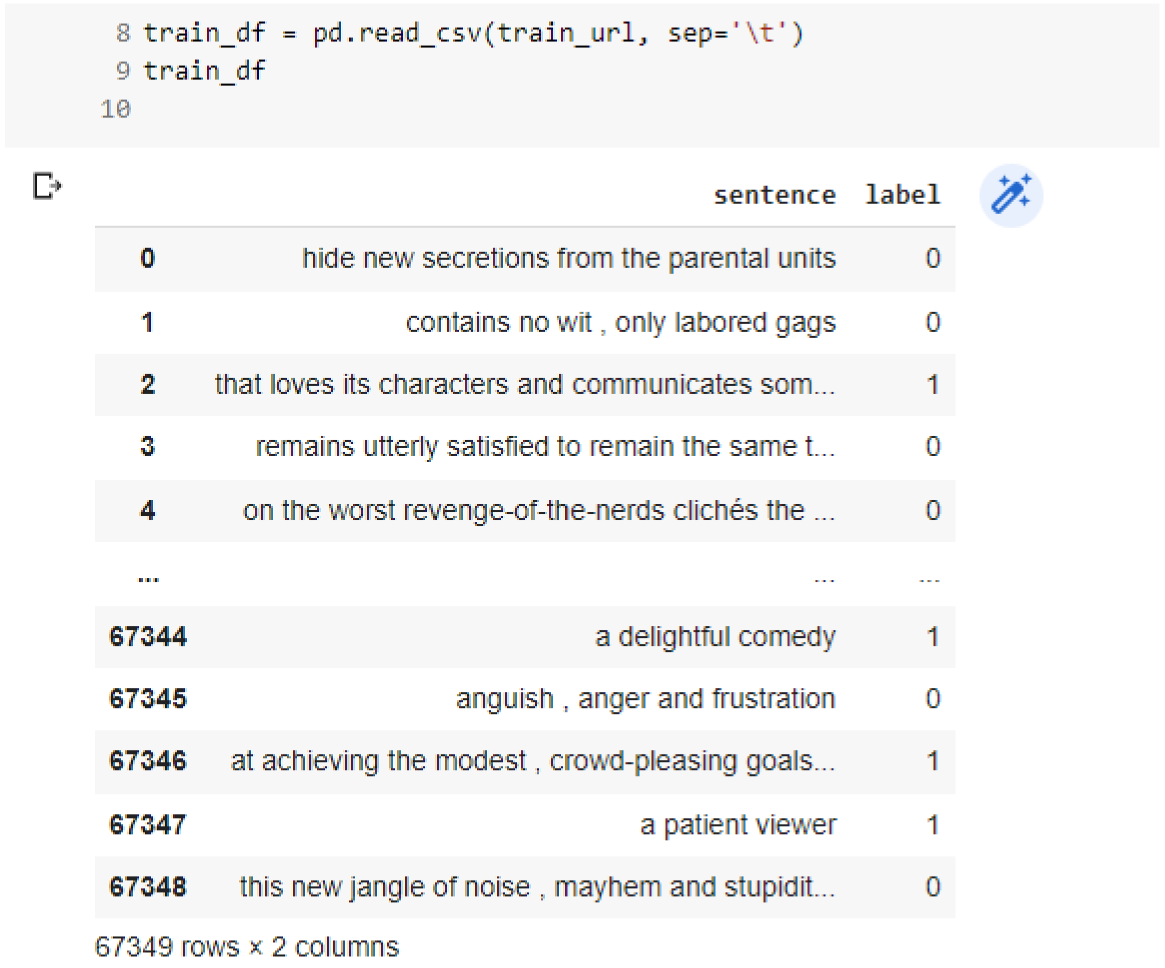
[16]
```python
sentence_has_was = corpus_df['sentence'].apply(lambda sentence:'was' in sentence)
sentence_has_was
```

```
0     True
1    False
2     True
3     True
Name: sentence, dtype: bool
```

[17]
```python
corpus_df[sentence_has_was]
```

	sentence	label	opposite_label	lowercased_sentence
0	The film was a delight--I was riveted.	1	0	the film was a delight--i was riveted.
2	It was a terrible flick, the worst I have ever...	0	1	it was a terrible flick, the worst i have ever...
3	I have a feeling the film was recut poorly.	0	1	i have a feeling the film was recut poorly.

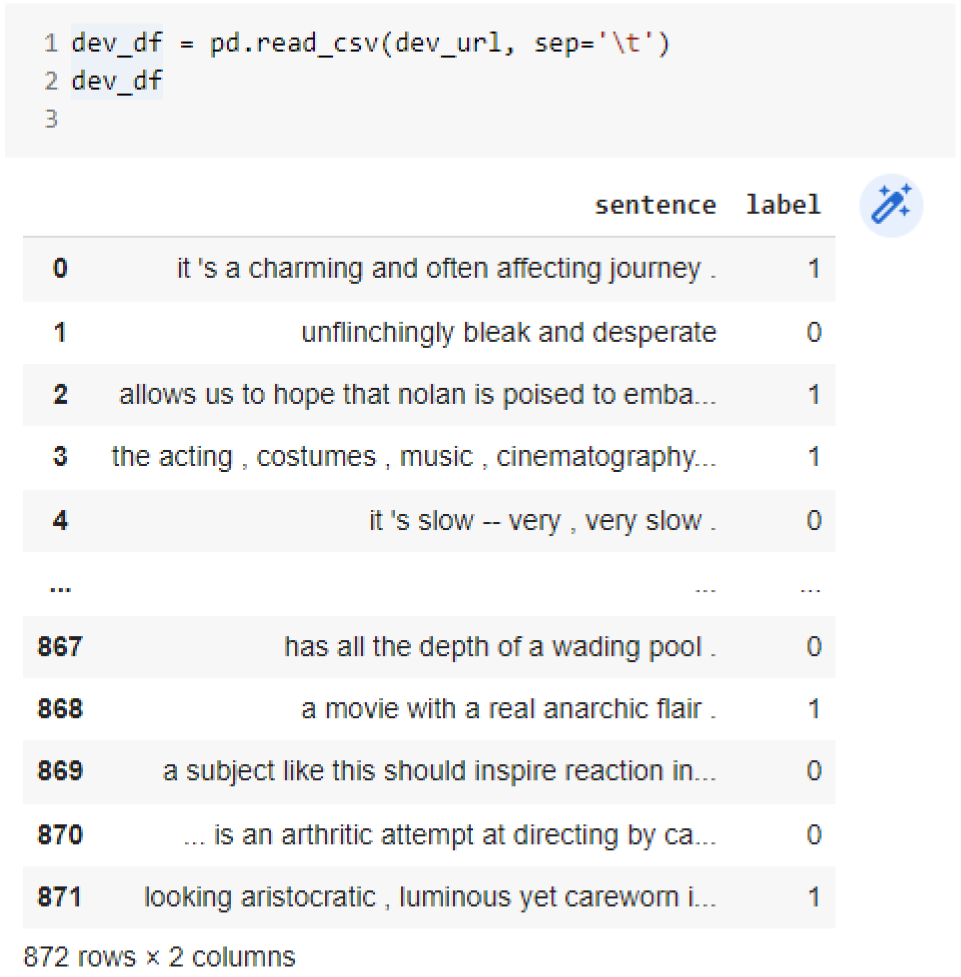
Reading the dataset



```python
train_df = pd.read_csv(train_url, sep='\t')
train_df
```

	sentence	label
0	hide new secretions from the parental units	0
1	contains no wit , only labored gags	0
2	that loves its characters and communicates som...	1
3	remains utterly satisfied to remain the same t...	0
4	on the worst revenge-of-the-nerds clichés the ...	0
...	...	...
67344	a delightful comedy	1
67345	anguish , anger and frustration	0
67346	at achieving the modest , crowd-pleasing goals...	1
67347	a patient viewer	1
67348	this new jangle of noise , mayhem and stupidit...	0

67349 rows × 2 columns

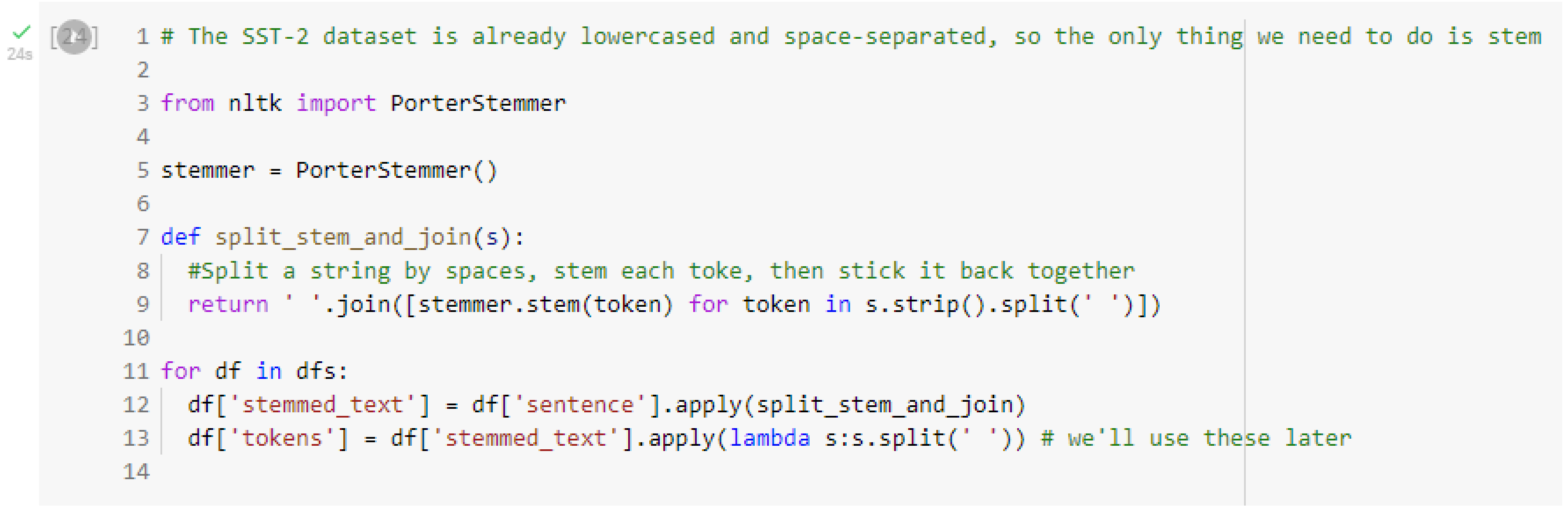


```python
dev_df = pd.read_csv(dev_url, sep='\t')
dev_df
```

	sentence	label
0	it 's a charming and often affecting journey .	1
1	unflinchingly bleak and desperate	0
2	allows us to hope that nolan is poised to emba...	1
3	the acting , costumes , music , cinematography...	1
4	it 's slow -- very , very slow .	0
...	...	...
867	has all the depth of a wading pool .	0
868	a movie with a real anarchic flair .	1
869	a subject like this should inspire reaction in...	0
870	... is an arthritic attempt at directing by ca...	0
871	looking aristocratic , luminous yet careworn i...	1

872 rows × 2 columns

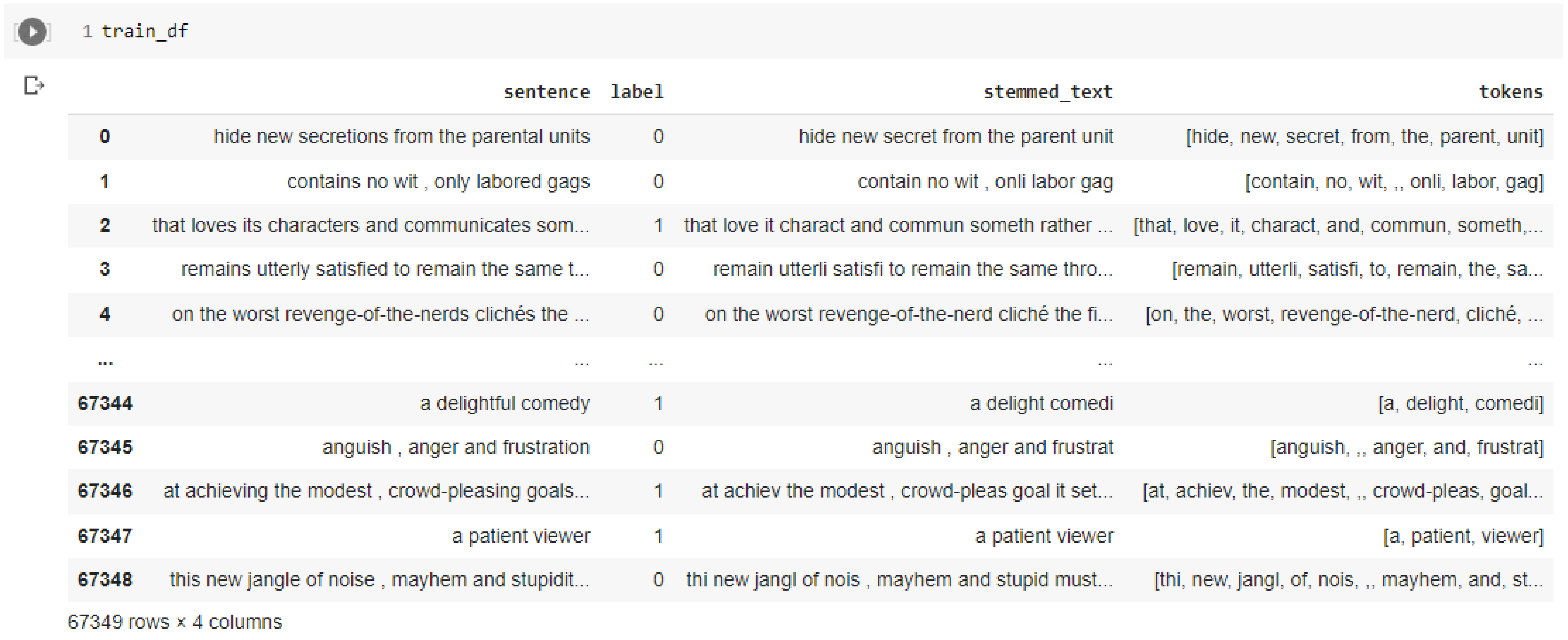
Preprocessing the dataset



[24]
24s

```python
# The SST-2 dataset is already lowercased and space-separated, so the only thing we need to do is stem

from nltk import PorterStemmer

stemmer = PorterStemmer()

def split_stem_and_join(s):
  #Split a string by spaces, stem each toke, then stick it back together
  return ' '.join([stemmer.stem(token) for token in s.strip().split(' ')])

for df in dfs:
  df['stemmed_text'] = df['sentence'].apply(split_stem_and_join)
  df['tokens'] = df['stemmed_text'].apply(lambda s:s.split(' ')) # we'll use these later

```

Preprocessing the dataset



```
train_df
```

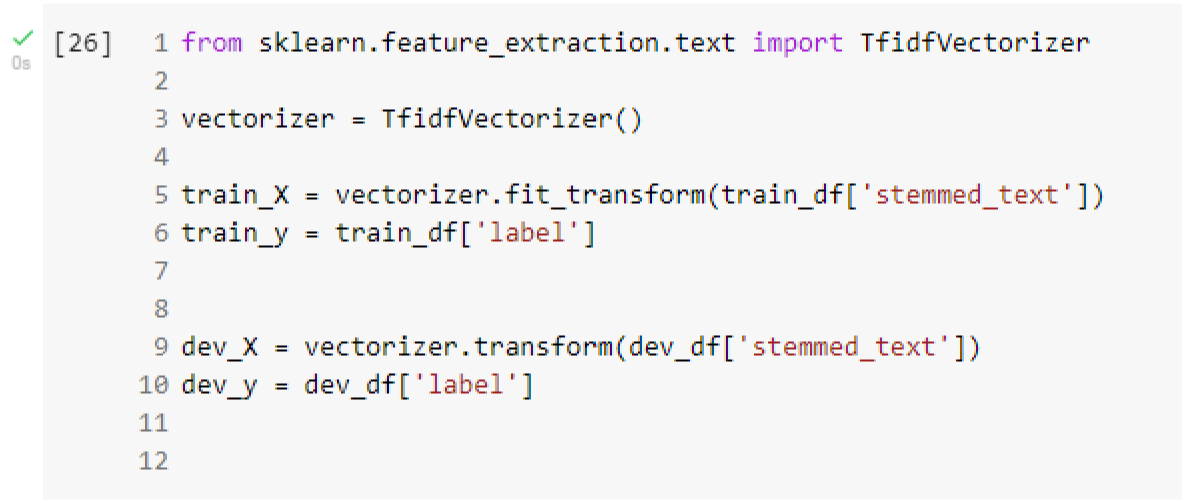
	sentence	label	stemmed_text	tokens
0	hide new secretions from the parental units	0	hide new secret from the parent unit	[hide, new, secret, from, the, parent, unit]
1	contains no wit , only labored gags	0	contain no wit , onli labor gag	[contain, no, wit, ,, onli, labor, gag]
2	that loves its characters and communicates som...	1	that love it charact and commun someth rather ...	[that, love, it, charact, and, commun, someth,...
3	remains utterly satisfied to remain the same t...	0	remain utterli satisfi to remain the same thro...	[remain, utterli, satisfi, to, remain, the, sa...
4	on the worst revenge-of-the-nerds clichés the ...	0	on the worst revenge-of-the-nerd cliché the fi...	[on, the, worst, revenge-of-the-nerd, cliché, ...
...	...	...	...	...
67344	a delightful comedy	1	a delight comedi	[a, delight, comedi]
67345	anguish , anger and frustration	0	anguish , anger and frustrat	[anguish, ,, anger, and, frustrat]
67346	at achieving the modest , crowd-pleasing goals...	1	at achiev the modest , crowd-pleas goal it set...	[at, achiev, the, modest, ,, crowd-pleas, goal...
67347	a patient viewer	1	a patient viewer	[a, patient, viewer]
67348	this new jangle of noise , mayhem and stupidit...	0	thi new jangl of nois , mayhem and stupid must...	[thi, new, jangl, of, nois, ,, mayhem, and, st...

67349 rows × 4 columns

Vectorizing the text

Note that we are using .fit_transform() on the training data, but only .transform() on the development set.

Why?

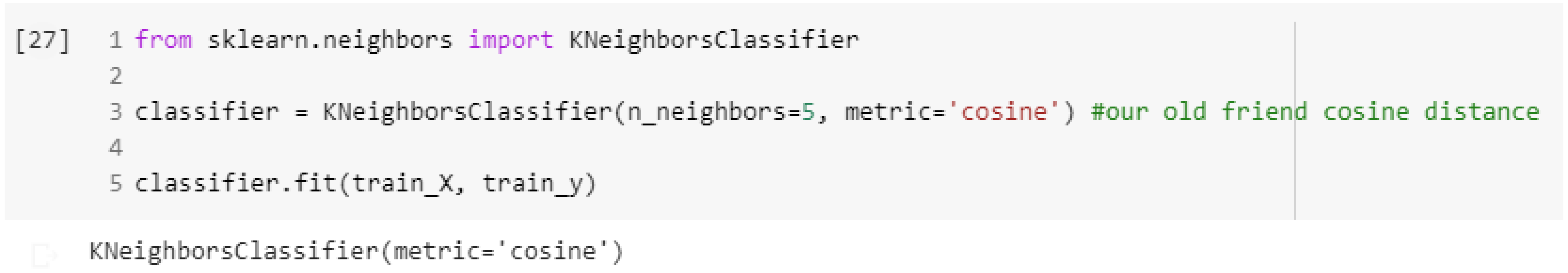


[26]

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

train_X = vectorizer.fit_transform(train_df['stemmed_text'])
train_y = train_df['label']


dev_X = vectorizer.transform(dev_df['stemmed_text'])
dev_y = dev_df['label']


```

Training the model

Very simple. Just pick a number of neighbors to consider, and a distance metric, and we're off to the races.



```
[27]
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5, metric='cosine') #our old friend cosine distance

classifier.fit(train_X, train_y)
```

```
KNeighborsClassifier(metric='cosine')
```

Evaluating classifiers

Given a set of predictions $\hat{Y}$ and the true labels Y, there are a few different ways to evaluate how well we did.

One way to divide up predictions is into errors ($\hat{y} \neq y$) and non-errors ($\hat{y} = y$)

In a binary classification setting (like SST-2), we can also think about different kinds of errors and non-errors:

- True positives (TPs): $\hat{y} = 1;\ y = 1$
- True negatives (TNs): $\hat{y} = 0;\ y = 0$
- False positives (FPs): $\hat{y} = 1;\ y = 0$
- False negatives (FNs): $\hat{y} = 0;\ y = 1$

https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers

Things get more complicated with 3+ classes, but don't worry about it for now

Accuracy

What percentage of my guesses were correct?

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} = \frac{COR}{COR + ERR}$$

Problematic when the true labels are highly **unbalanced** (e.g. 90% positive, 10% negative)

- 91% accuracy looks good by itself, but not so great if you could get 90% by just guessing the most common class.

Recall

Of all the positives examples, what percentage of them did I correctly guess were positive?

AKA sensitivity, true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

Particularly important when we *really* don't want to miss any positives

- I.e. we want to avoid false negatives
- What are tasks for which this is this the case?

Precision

When I guessed positive, how likely was I to be correct?

AKA positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$

Particularly important when we really don’t want to falsely predict any example as positive

- What are tasks where this is the case?

F1 score

Defined as the harmonic mean of precision and recall

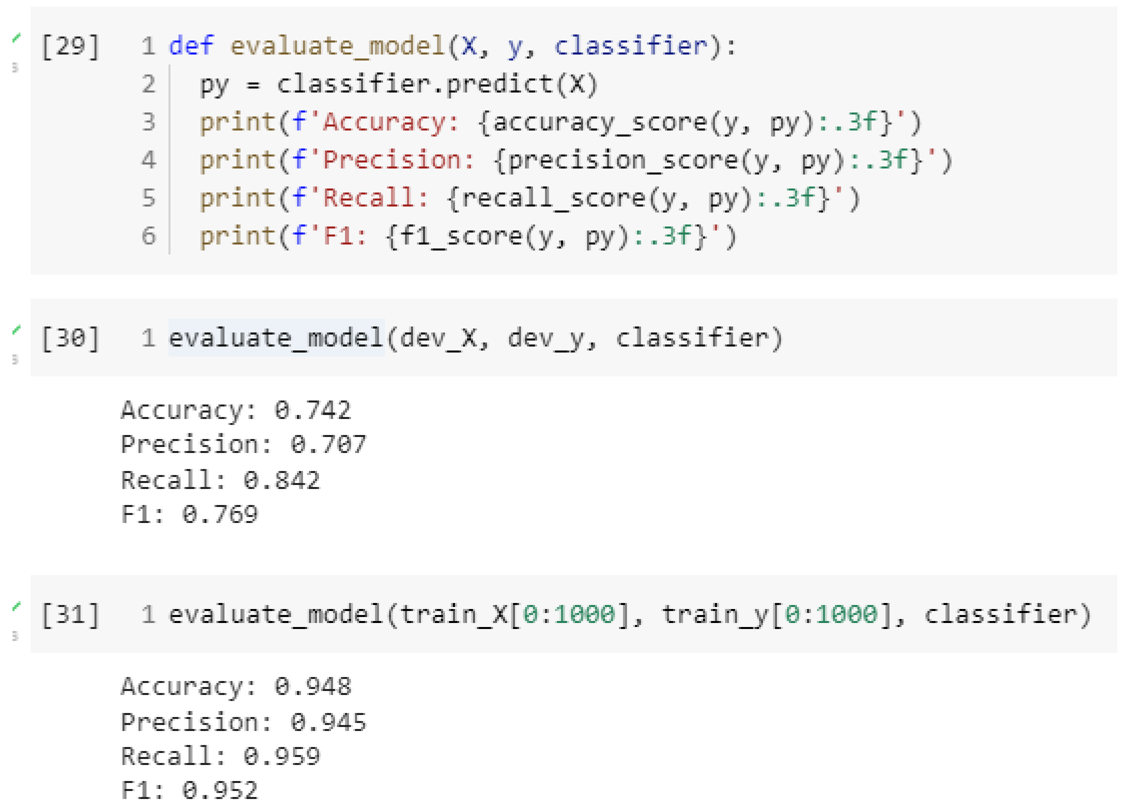
Balances precision and recall.

$$F_1 = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Does a better job of handling unbalanced data

- Although you still probably want to calculate F1 for both possible definitions of "positive", then take the mean of *that* value

Evaluating the predictions



```
[29]
def evaluate_model(X, y, classifier):
  py = classifier.predict(X)
  print(f'Accuracy: {accuracy_score(y, py):.3f}')
  print(f'Precision: {precision_score(y, py):.3f}')
  print(f'Recall: {recall_score(y, py):.3f}')
  print(f'F1: {f1_score(y, py):.3f}')
```

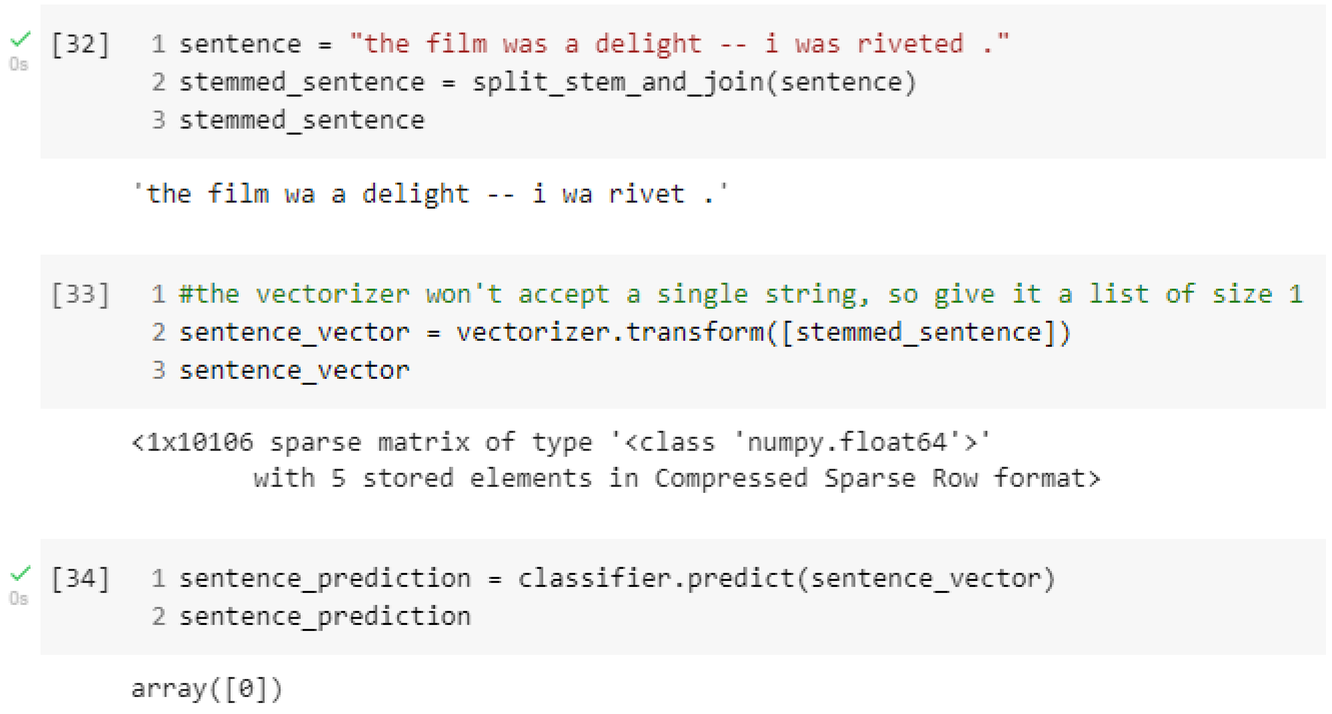
```
[30]
evaluate_model(dev_X, dev_y, classifier)
```

```
Accuracy: 0.742
Precision: 0.707
Recall: 0.842
F1: 0.769
```

```
[31]
evaluate_model(train_X[0:1000], train_y[0:1000], classifier)
```

```
Accuracy: 0.948
Precision: 0.945
Recall: 0.959
F1: 0.952
```

Understanding individual predictions



```
[32]
sentence = "the film was a delight -- i was riveted ."
stemmed_sentence = split_stem_and_join(sentence)
stemmed_sentence
```

```
'the film wa a delight -- i wa rivet .'
```

```
[33]
#the vectorizer won't accept a single string, so give it a list of size 1
sentence_vector = vectorizer.transform([stemmed_sentence])
sentence_vector
```

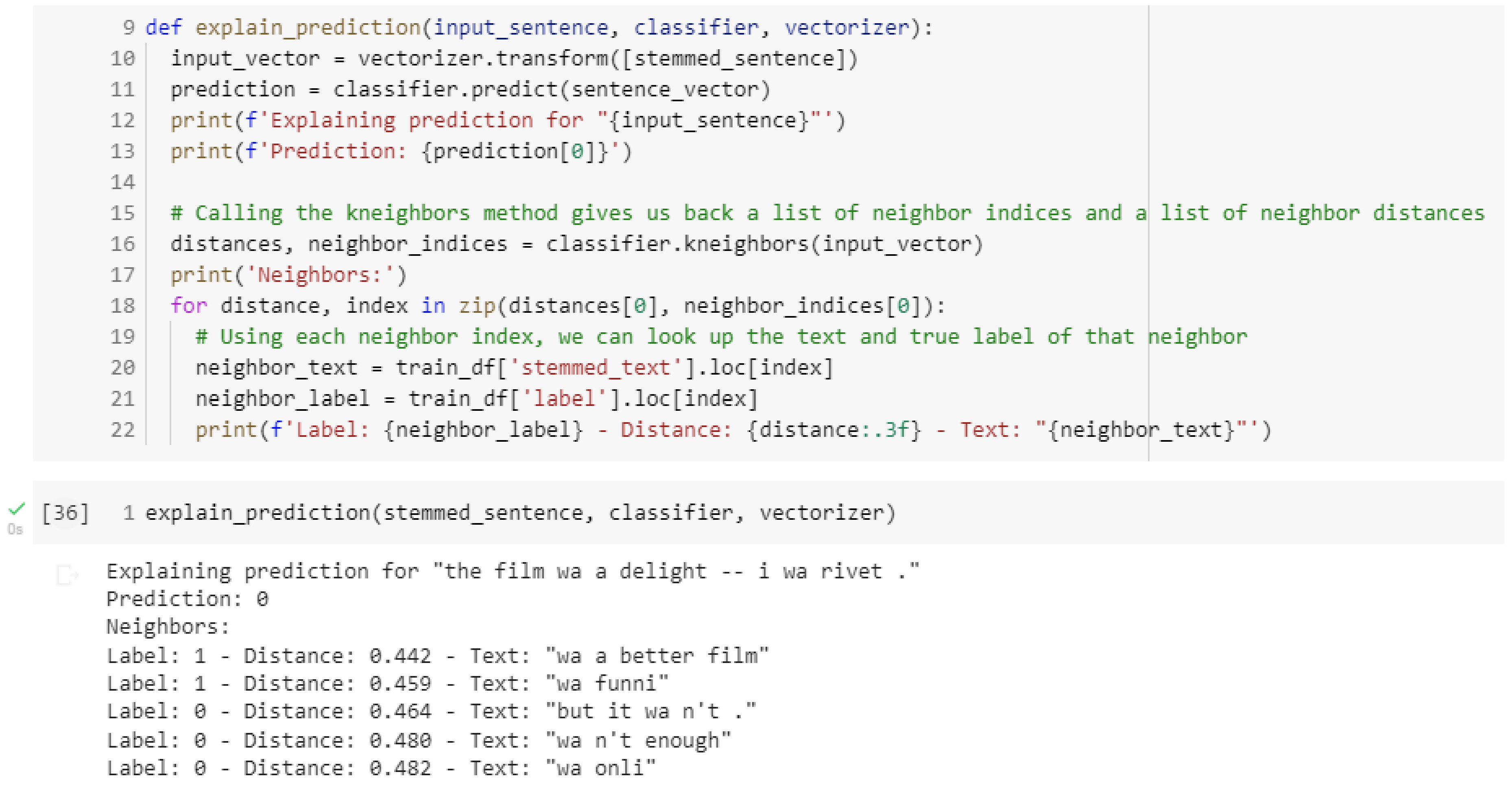
```
<1x10106 sparse matrix of type '<class 'numpy.float64'>'
	with 5 stored elements in Compressed Sparse Row format>
```

```
[34]
sentence_prediction = classifier.predict(sentence_vector)
sentence_prediction
```

```
array([0])
```

This is wrong! But why?

Explaining individual predictions



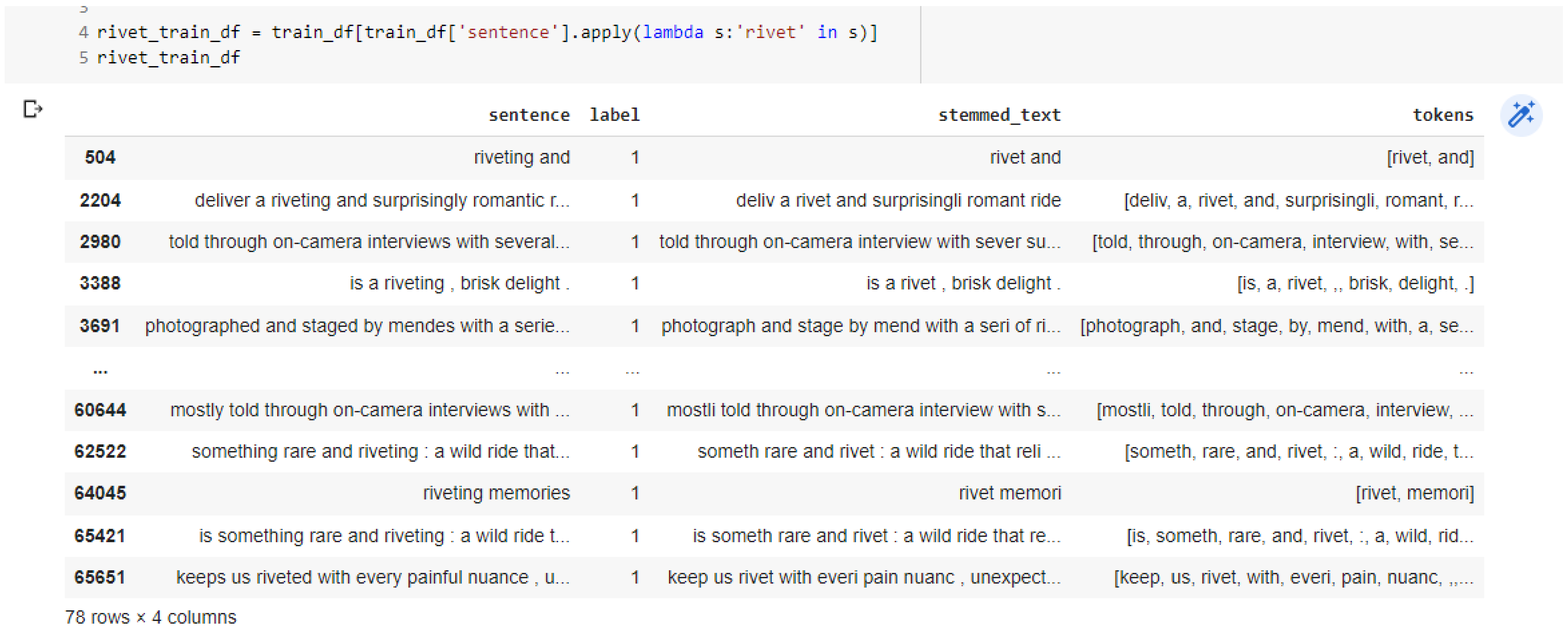
```python
def explain_prediction(input_sentence, classifier, vectorizer):
  input_vector = vectorizer.transform([stemmed_sentence])
  prediction = classifier.predict(sentence_vector)
  print(f'Explaining prediction for "{input_sentence}"')
  print(f'Prediction: {prediction[0]}')

  # Calling the kneighbors method gives us back a list of neighbor indices and a list of neighbor distances
  distances, neighbor_indices = classifier.kneighbors(input_vector)
  print('Neighbors:')
  for distance, index in zip(distances[0], neighbor_indices[0]):
    # Using each neighbor index, we can look up the text and true label of that neighbor
    neighbor_text = train_df['stemmed_text'].loc[index]
    neighbor_label = train_df['label'].loc[index]
    print(f'Label: {neighbor_label} - Distance: {distance:.3f} - Text: "{neighbor_text}"')
```

✓ 0s [36]

```python
explain_prediction(stemmed_sentence, classifier, vectorizer)
```

```
Explaining prediction for "the film wa a delight -- i wa rivet ."
Prediction: 0
Neighbors:
Label: 1 - Distance: 0.442 - Text: "wa a better film"
Label: 1 - Distance: 0.459 - Text: "wa funni"
Label: 0 - Distance: 0.464 - Text: "but it wa n't ."
Label: 0 - Distance: 0.480 - Text: "wa n't enough"
Label: 0 - Distance: 0.482 - Text: "wa onli"
```

Finding what we think should be the nearest neighbor(s)



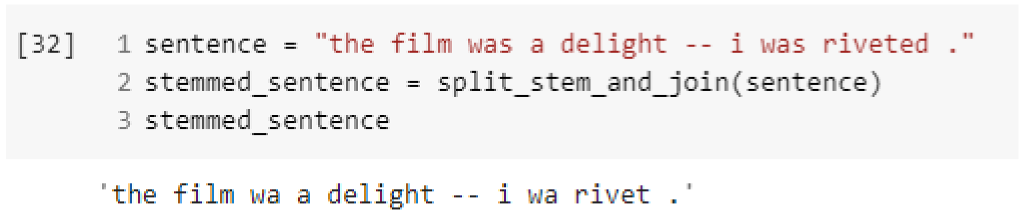
```python
rivet_train_df = train_df[train_df['sentence'].apply(lambda s:'rivet' in s)]
rivet_train_df
```

	sentence	label	stemmed_text	tokens
504	riveting and	1	rivet and	[rivet, and]
2204	deliver a riveting and surprisingly romantic r...	1	deliv a rivet and surprisingli romant ride	[deliv, a, rivet, and, surprisingli, romant, r...
2980	told through on-camera interviews with several...	1	told through on-camera interview with sever su...	[told, through, on-camera, interview, with, se...
3388	is a riveting , brisk delight .	1	is a rivet , brisk delight .	[is, a, rivet, ,, brisk, delight, .]
3691	photographed and staged by mendes with a serie...	1	photograph and stage by mend with a seri of ri...	[photograph, and, stage, by, mend, with, a, se...
...	...	...	...	...
60644	mostly told through on-camera interviews with ...	1	mostli told through on-camera interview with s...	[mostli, told, through, on-camera, interview, ...
62522	something rare and riveting : a wild ride that...	1	someth rare and rivet : a wild ride that reli ...	[someth, rare, and, rivet, :, a, wild, ride, t...
64045	riveting memories	1	rivet memori	[rivet, memori]
65421	is something rare and riveting : a wild ride t...	1	is someth rare and rivet : a wild ride that re...	[is, someth, rare, and, rivet, :, a, wild, rid...
65651	keeps us riveted with every painful nuance , u...	1	keep us rivet with everi pain nuanc , unexpect...	[keep, us, rivet, with, everi, pain, nuanc, ,,...

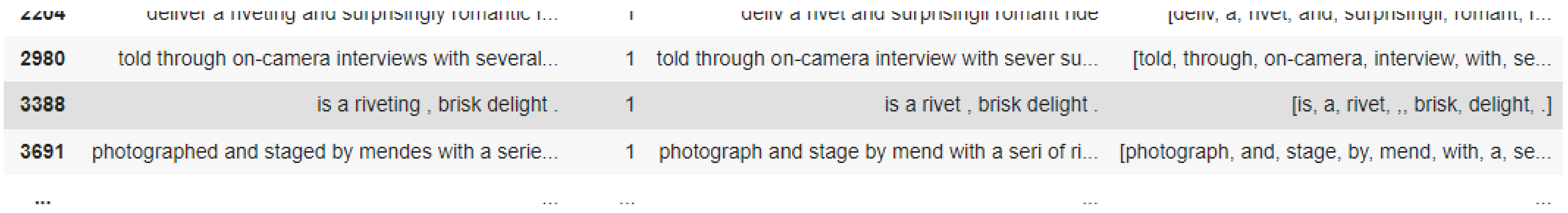
78 rows × 4 columns

Finding what we think should be the nearest neighbor(s)

Found a pretty good one…

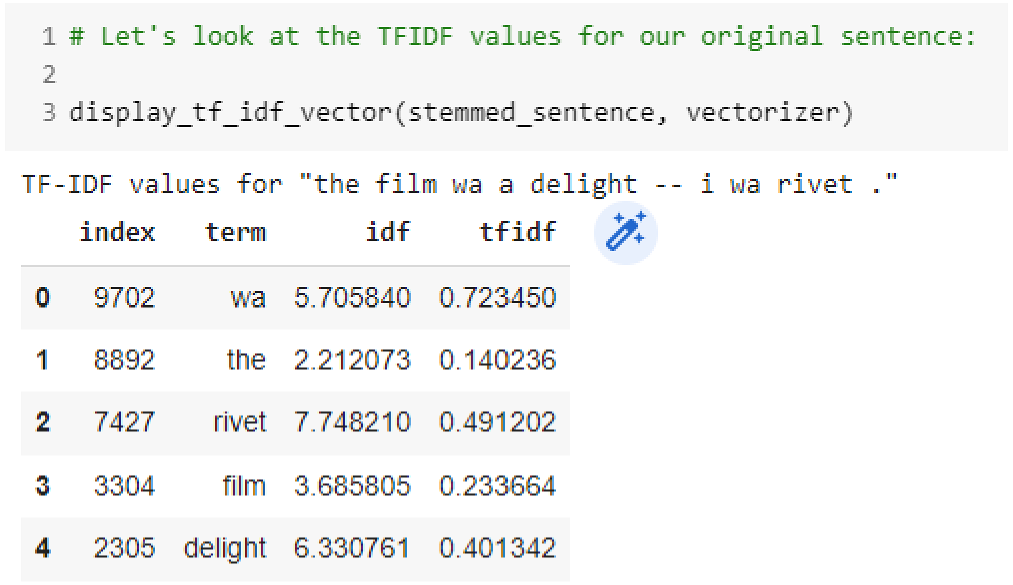


```
[32] sentence = "the film was a delight -- i was riveted ."
     stemmed_sentence = split_stem_and_join(sentence)
     stemmed_sentence

'the film wa a delight -- i wa rivet .'
```



2204	deliver a riveting and surprisingly romantic r...	1	deliv a rivet and surprisingli romant ride	[deliv, a, rivet, and, surprisingli, romant, r...
2980	told through on-camera interviews with several...	1	told through on-camera interview with sever su...	[told, through, on-camera, interview, with, se...
3388	is a riveting , brisk delight .	1	is a rivet , brisk delight .	[is, a, rivet, ,, brisk, delight, .]
3691	photographed and staged by mendes with a serie...	1	photograph and stage by mend with a seri of ri...	[photograph, and, stage, by, mend, with, a, se...
...	...	...	...	...

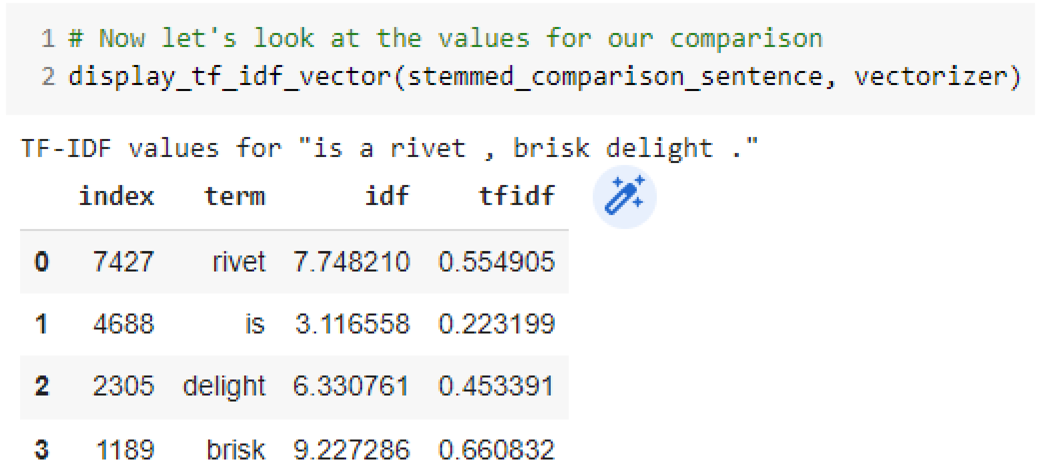
Understanding why it isn’t



```
# Let's look at the TFIDF values for our original sentence:

display_tf_idf_vector(stemmed_sentence, vectorizer)
```

```
TF-IDF values for "the film wa a delight -- i wa rivet ."
```

	index	term	idf	tfidf
0	9702	wa	5.705840	0.723450
1	8892	the	2.212073	0.140236
2	7427	rivet	7.748210	0.491202
3	3304	film	3.685805	0.233664
4	2305	delight	6.330761	0.401342

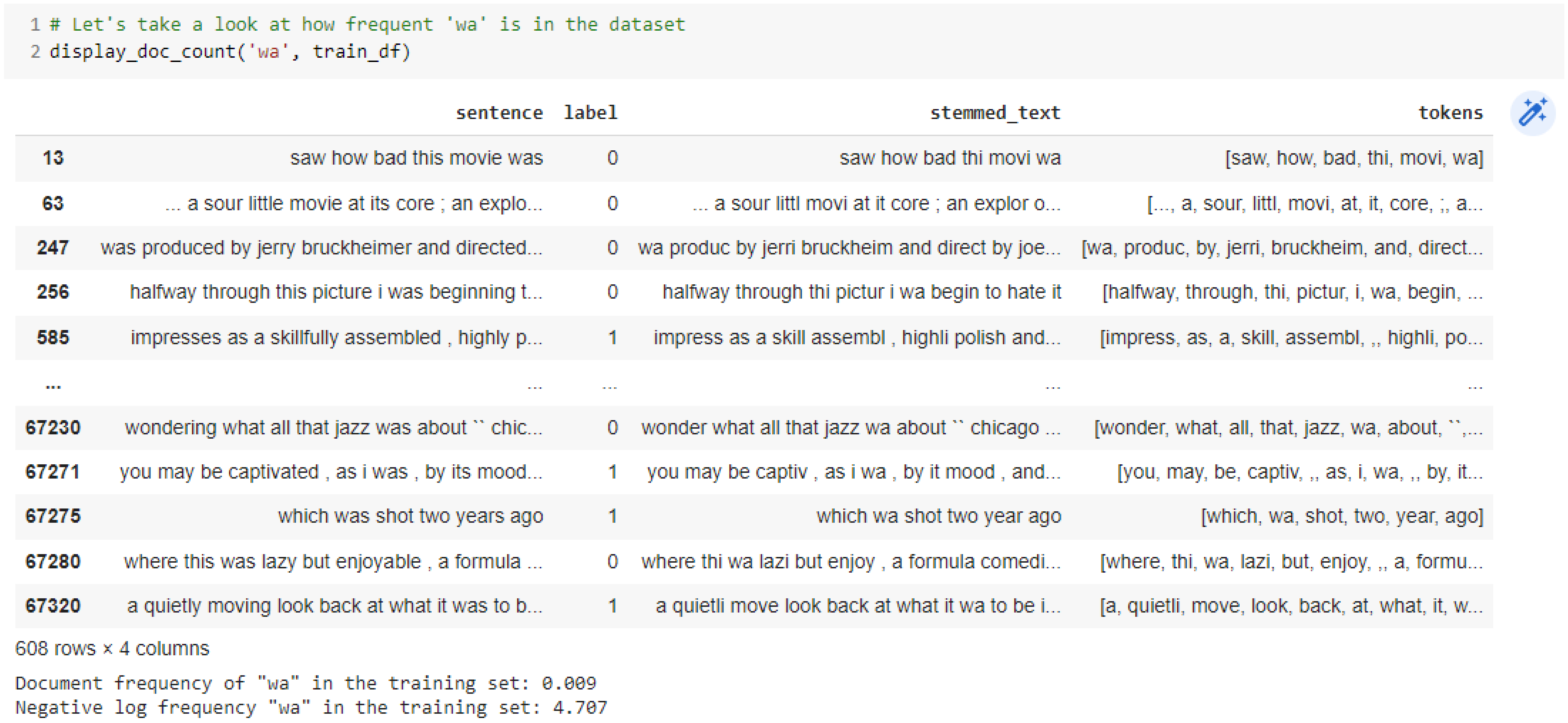


```
# Now let's look at the values for our comparison
display_tf_idf_vector(stemmed_comparison_sentence, vectorizer)
```

```
TF-IDF values for "is a rivet , brisk delight ."
```

	index	term	idf	tfidf
0	7427	rivet	7.748210	0.554905
1	4688	is	3.116558	0.223199
2	2305	delight	6.330761	0.453391
3	1189	brisk	9.227286	0.660832

Understanding “was” vs. “delight”



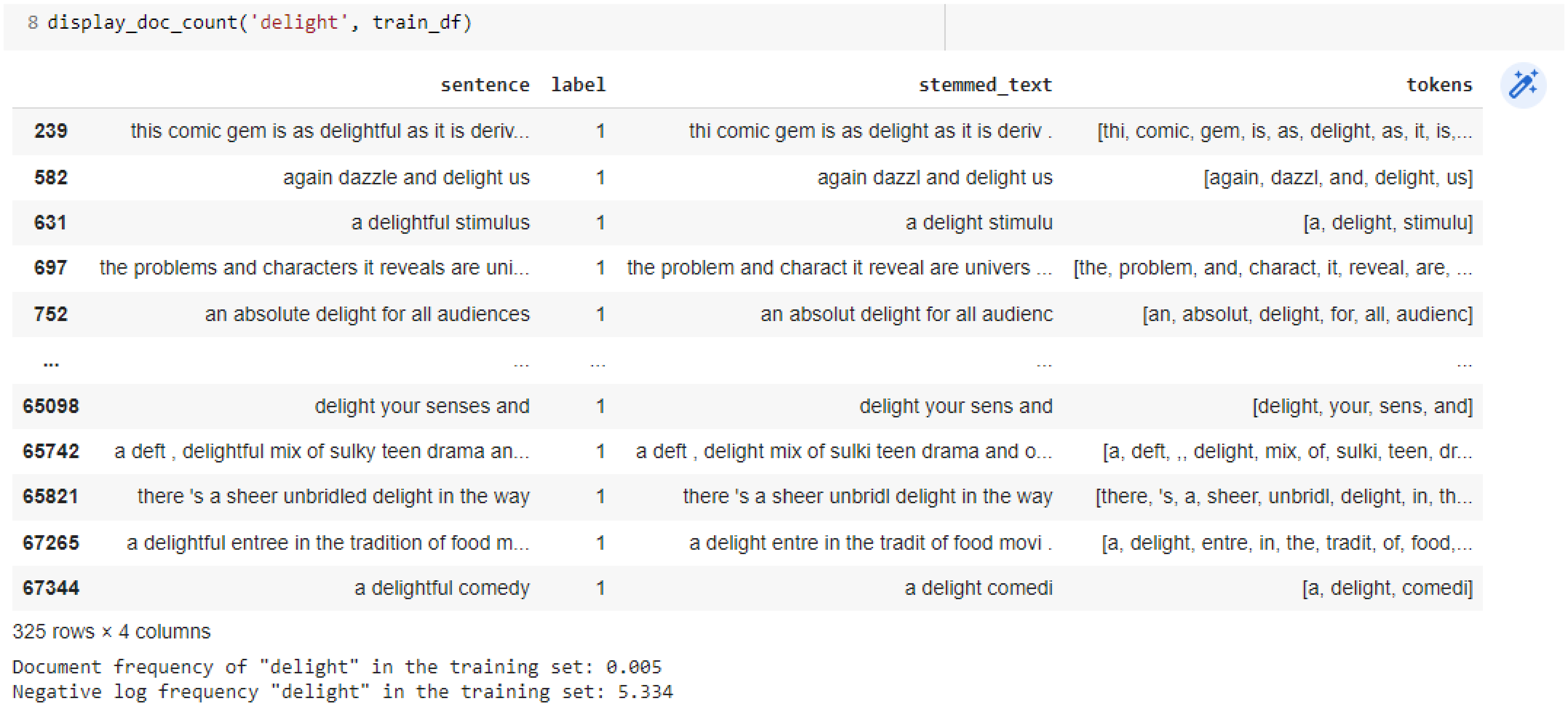
```python
# Let's take a look at how frequent 'wa' is in the dataset
display_doc_count('wa', train_df)
```

	sentence	label	stemmed_text	tokens
13	saw how bad this movie was	0	saw how bad thi movi wa	[saw, how, bad, thi, movi, wa]
63	... a sour little movie at its core ; an explo...	0	... a sour littl movi at it core ; an explor o...	[..., a, sour, littl, movi, at, it, core, ;, a...
247	was produced by jerry bruckheimer and directed...	0	wa produc by jerri bruckheim and direct by joe...	[wa, produc, by, jerri, bruckheim, and, direct...
256	halfway through this picture i was beginning t...	0	halfway through thi pictur i wa begin to hate it	[halfway, through, thi, pictur, i, wa, begin, ...
585	impresses as a skillfully assembled , highly p...	1	impress as a skill assembl , highli polish and...	[impress, as, a, skill, assembl, ,, highli, po...
...	...	...	...	...
67230	wondering what all that jazz was about `` chic...	0	wonder what all that jazz wa about `` chicago ...	[wonder, what, all, that, jazz, wa, about, ``,...
67271	you may be captivated , as i was , by its mood...	1	you may be captiv , as i wa , by it mood , and...	[you, may, be, captiv, ,, as, i, wa, ,, by, it...
67275	which was shot two years ago	1	which wa shot two year ago	[which, wa, shot, two, year, ago]
67280	where this was lazy but enjoyable , a formula ...	0	where thi wa lazi but enjoy , a formula comedi...	[where, thi, wa, lazi, but, enjoy, ,, a, formu...
67320	a quietly moving look back at what it was to b...	1	a quietli move look back at what it wa to be i...	[a, quietli, move, look, back, at, what, it, w...

608 rows × 4 columns

```
Document frequency of "wa" in the training set: 0.009
Negative log frequency "wa" in the training set: 4.707
```

Understanding “was” vs. “delight”



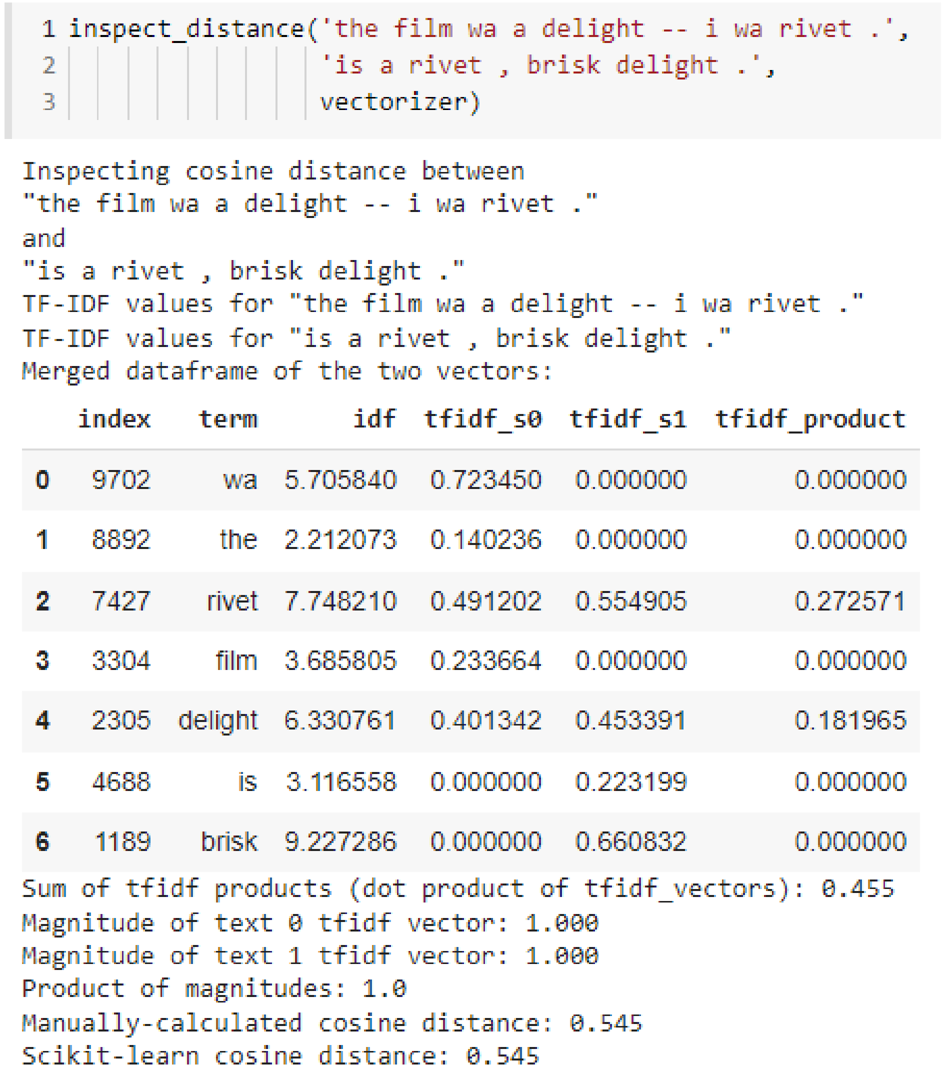
```python
display_doc_count('delight', train_df)
```

	sentence	label	stemmed_text	tokens
239	this comic gem is as delightful as it is deriv...	1	thi comic gem is as delight as it is deriv .	[thi, comic, gem, is, as, delight, as, it, is,...
582	again dazzle and delight us	1	again dazzl and delight us	[again, dazzl, and, delight, us]
631	a delightful stimulus	1	a delight stimulu	[a, delight, stimulu]
697	the problems and characters it reveals are uni...	1	the problem and charact it reveal are univers ...	[the, problem, and, charact, it, reveal, are, ...
752	an absolute delight for all audiences	1	an absolut delight for all audienc	[an, absolut, delight, for, all, audienc]
...	...	...	...	...
65098	delight your senses and	1	delight your sens and	[delight, your, sens, and]
65742	a deft , delightful mix of sulky teen drama an...	1	a deft , delight mix of sulki teen drama and o...	[a, deft, ,, delight, mix, of, sulki, teen, dr...
65821	there 's a sheer unbridled delight in the way	1	there 's a sheer unbridl delight in the way	[there, 's, a, sheer, unbridl, delight, in, th...
67265	a delightful entree in the tradition of food m...	1	a delight entre in the tradit of food movi .	[a, delight, entre, in, the, tradit, of, food,...
67344	a delightful comedy	1	a delight comedi	[a, delight, comedi]

325 rows × 4 columns

```
Document frequency of "delight" in the training set: 0.005
Negative log frequency "delight" in the training set: 5.334
```

Examining the two distances

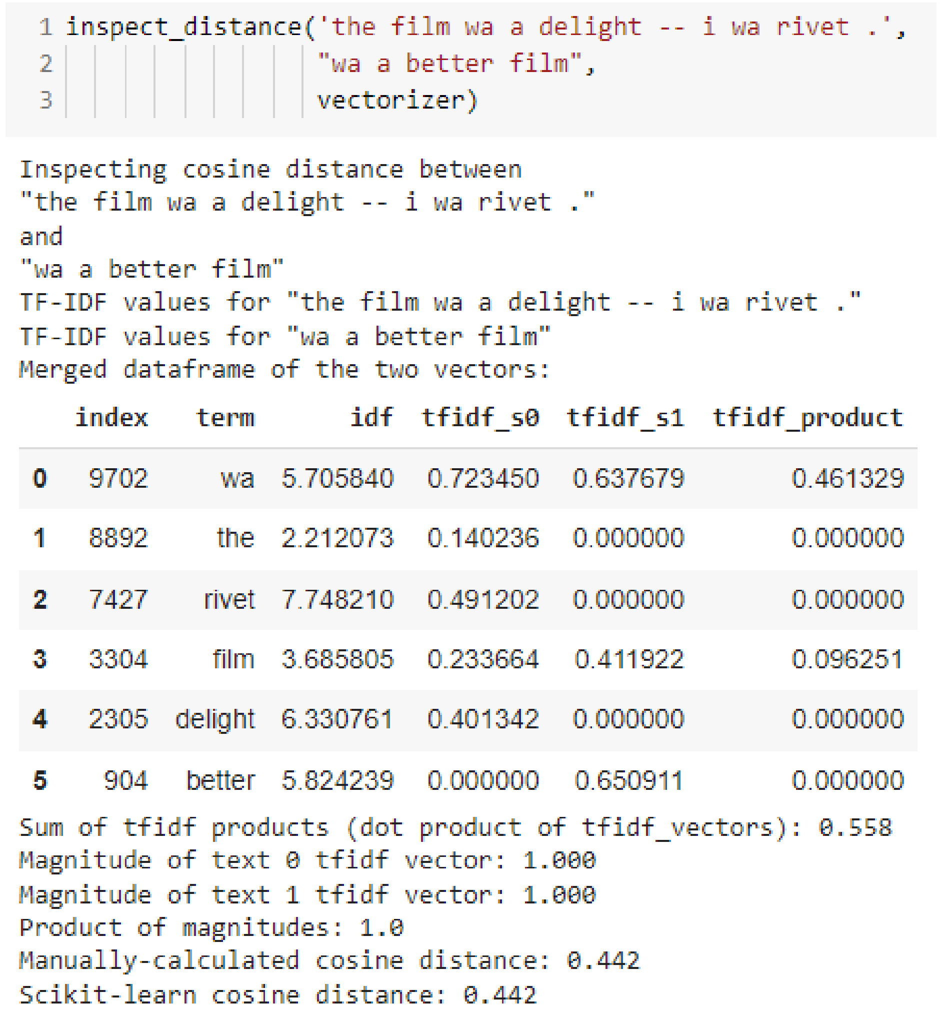


```
inspect_distance('the film wa a delight -- i wa rivet .',
                 'is a rivet , brisk delight .',
                 vectorizer)
```

```
Inspecting cosine distance between
"the film wa a delight -- i wa rivet ."
and
"is a rivet , brisk delight ."
TF-IDF values for "the film wa a delight -- i wa rivet ."
TF-IDF values for "is a rivet , brisk delight ."
Merged dataframe of the two vectors:
```

	index	term	idf	tfidf_s0	tfidf_s1	tfidf_product
0	9702	wa	5.705840	0.723450	0.000000	0.000000
1	8892	the	2.212073	0.140236	0.000000	0.000000
2	7427	rivet	7.748210	0.491202	0.554905	0.272571
3	3304	film	3.685805	0.233664	0.000000	0.000000
4	2305	delight	6.330761	0.401342	0.453391	0.181965
5	4688	is	3.116558	0.000000	0.223199	0.000000
6	1189	brisk	9.227286	0.000000	0.660832	0.000000

```
Sum of tfidf products (dot product of tfidf_vectors): 0.455
Magnitude of text 0 tfidf vector: 1.000
Magnitude of text 1 tfidf vector: 1.000
Product of magnitudes: 1.0
Manually-calculated cosine distance: 0.545
Scikit-learn cosine distance: 0.545
```



```
inspect_distance('the film wa a delight -- i wa rivet .',
                 "wa a better film",
                 vectorizer)
```

```
Inspecting cosine distance between
"the film wa a delight -- i wa rivet ."
and
"wa a better film"
TF-IDF values for "the film wa a delight -- i wa rivet ."
TF-IDF values for "wa a better film"
Merged dataframe of the two vectors:
```

	index	term	idf	tfidf_s0	tfidf_s1	tfidf_product
0	9702	wa	5.705840	0.723450	0.637679	0.461329
1	8892	the	2.212073	0.140236	0.000000	0.000000
2	7427	rivet	7.748210	0.491202	0.000000	0.000000
3	3304	film	3.685805	0.233664	0.411922	0.096251
4	2305	delight	6.330761	0.401342	0.000000	0.000000
5	904	better	5.824239	0.000000	0.650911	0.000000

```
Sum of tfidf products (dot product of tfidf_vectors): 0.558
Magnitude of text 0 tfidf vector: 1.000
Magnitude of text 1 tfidf vector: 1.000
Product of magnitudes: 1.0
Manually-calculated cosine distance: 0.442
Scikit-learn cosine distance: 0.442
```

The problem

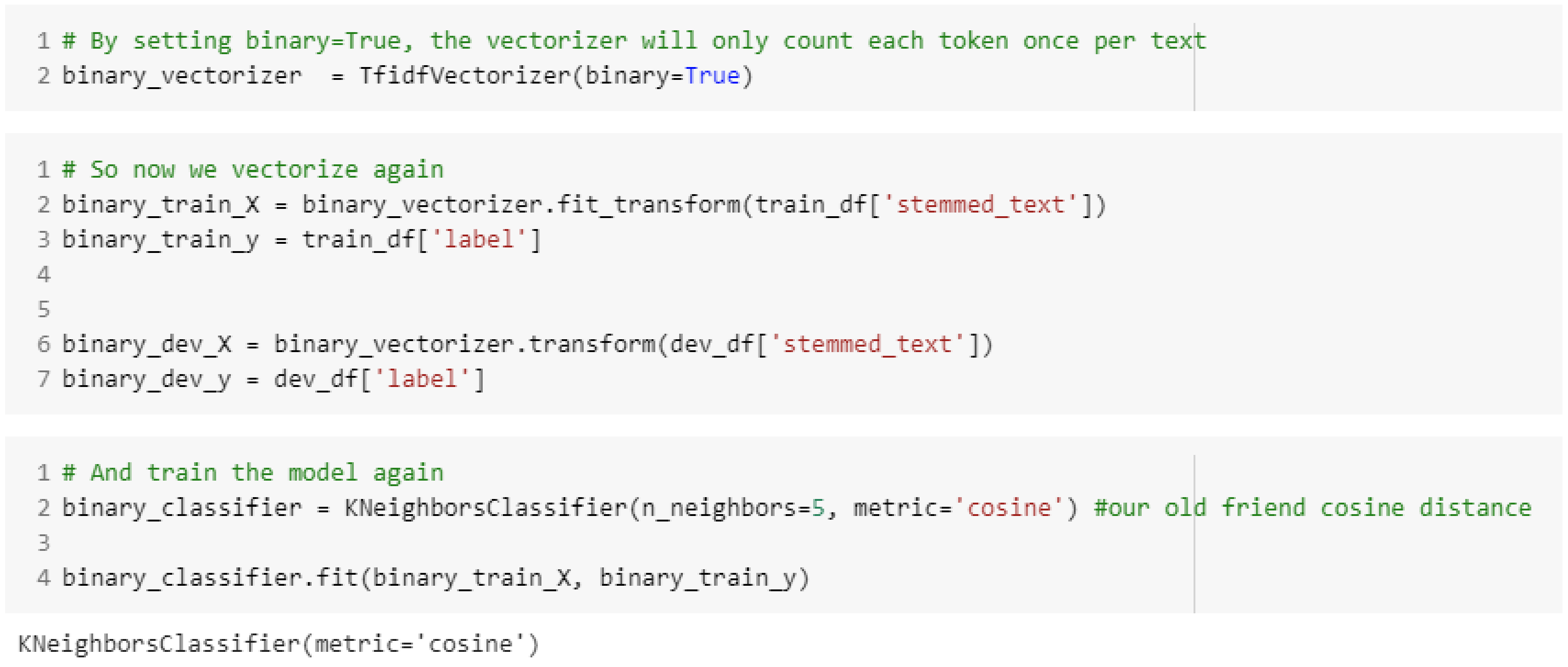
3 main things going on here:

1. It turns out that “was” is less common in the corpus (only 608 instances) than we might expect compared to delight (325 instances)
2. “was” occurs twice in "the film was a delight -- i was riveted .”, so it gets a higher tf-idf weight for that vector
3. Because cosine distance is normalized by vector magnitude, the tf-idf values in shorter texts get a higher value than the same ones in longer texts

We can’t do anything about 1 without changing the corpus, or about 3 without using a different distance metric

But what about 2?

Training a new model



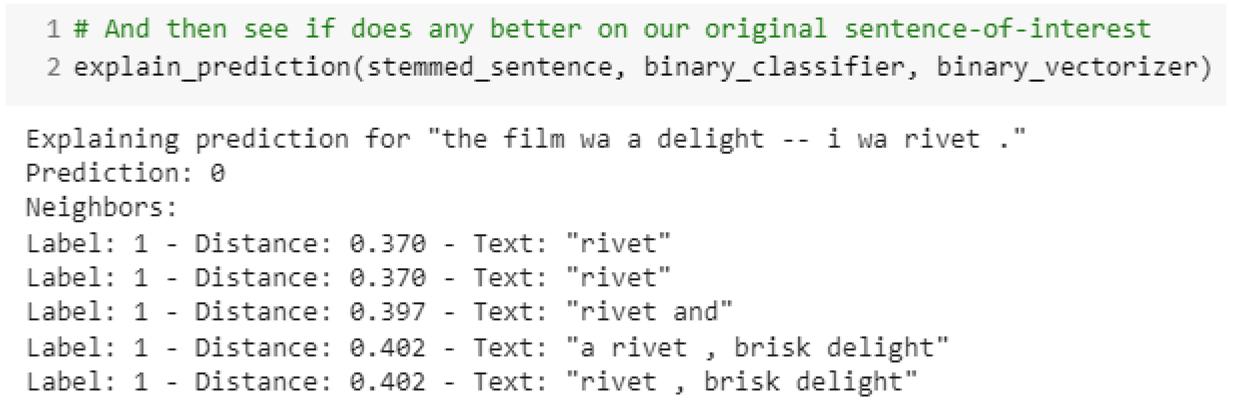
```python
# By setting binary=True, the vectorizer will only count each token once per text
binary_vectorizer  = TfidfVectorizer(binary=True)
```

```python
# So now we vectorize again
binary_train_X = binary_vectorizer.fit_transform(train_df['stemmed_text'])
binary_train_y = train_df['label']


binary_dev_X = binary_vectorizer.transform(dev_df['stemmed_text'])
binary_dev_y = dev_df['label']
```

```python
# And train the model again
binary_classifier = KNeighborsClassifier(n_neighbors=5, metric='cosine') #our old friend cosine distance

binary_classifier.fit(binary_train_X, binary_train_y)
```

```
KNeighborsClassifier(metric='cosine')
```

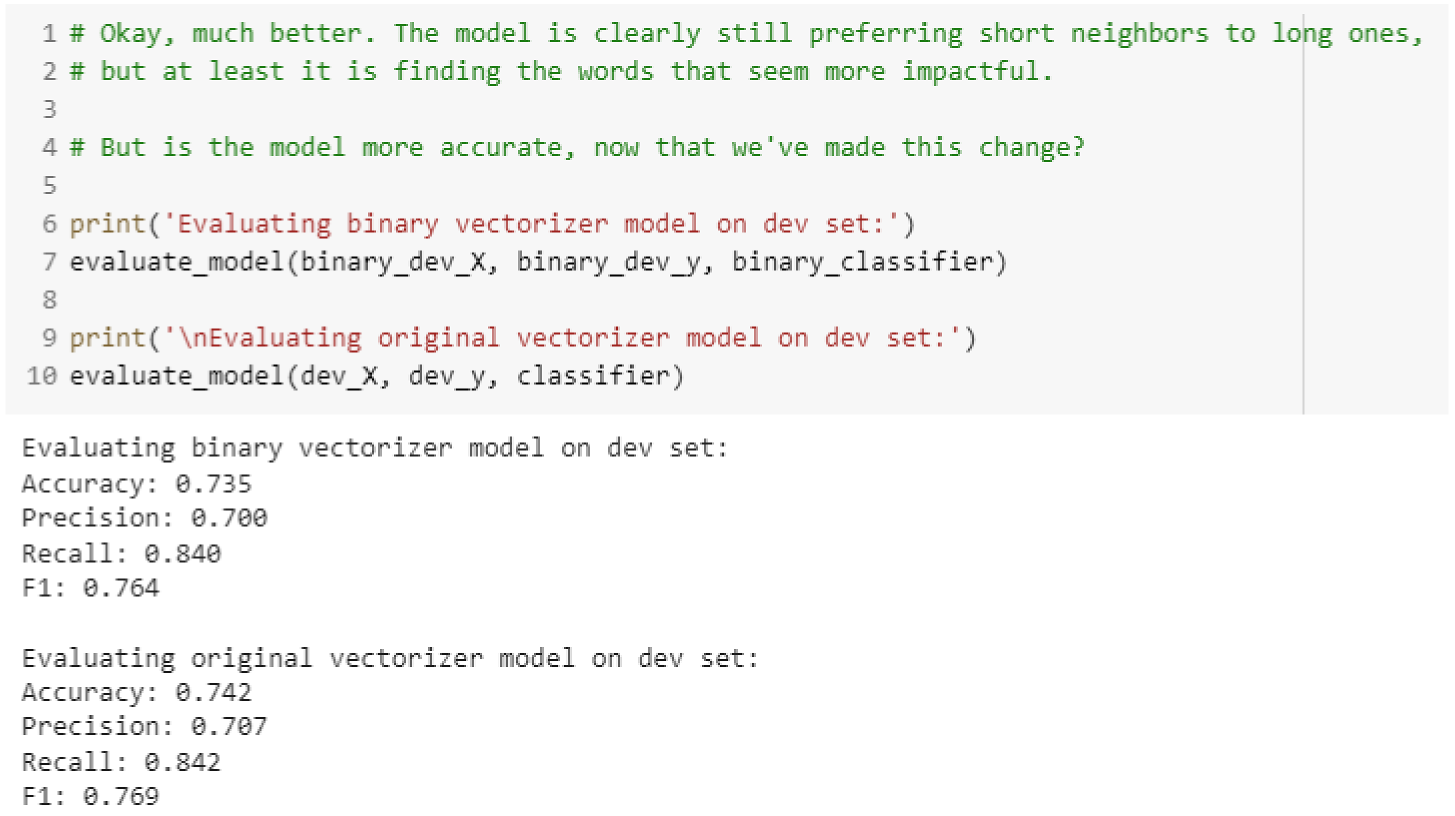
Training a new model



```python
# And then see if does any better on our original sentence-of-interest
explain_prediction(stemmed_sentence, binary_classifier, binary_vectorizer)
```

```
Explaining prediction for "the film wa a delight -- i wa rivet ."
Prediction: 0
Neighbors:
Label: 1 - Distance: 0.370 - Text: "rivet"
Label: 1 - Distance: 0.370 - Text: "rivet"
Label: 1 - Distance: 0.397 - Text: "rivet and"
Label: 1 - Distance: 0.402 - Text: "a rivet , brisk delight"
Label: 1 - Distance: 0.402 - Text: "rivet , brisk delight"
```

Training a new model



```
# Okay, much better. The model is clearly still preferring short neighbors to long ones,
# but at least it is finding the words that seem more impactful.

# But is the model more accurate, now that we've made this change?

print('Evaluating binary vectorizer model on dev set:')
evaluate_model(binary_dev_X, binary_dev_y, binary_classifier)

print('\nEvaluating original vectorizer model on dev set:')
evaluate_model(dev_X, dev_y, classifier)
```

```
Evaluating binary vectorizer model on dev set:
Accuracy: 0.735
Precision: 0.700
Recall: 0.840
F1: 0.764

Evaluating original vectorizer model on dev set:
Accuracy: 0.742
Precision: 0.707
Recall: 0.842
F1: 0.769
```

Why did I go through that with you?

1. I had to deal with it when I was writing the code, so now you get to deal with it too.

2. These kinds of issues come up all the time. Model debugging is part of the life of an NLP or data science practitioner.

Model confidence

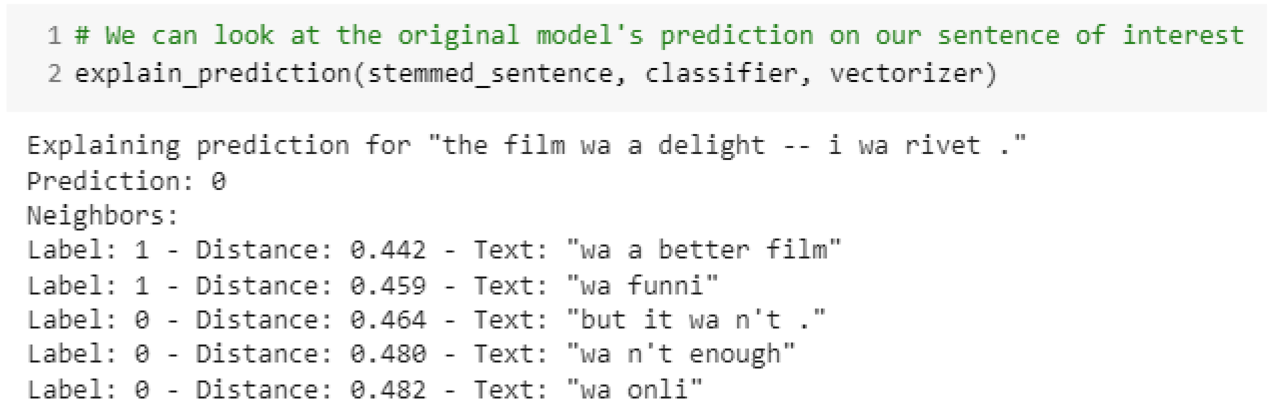
Sometimes you want not just a prediction, but a **confidence estimate** of how certain the classifier is in its prediction.

What are some cases where you might want this?

How to calculate confidence varies from model to model, and doing it robustly is a whole research topic in and of itself.

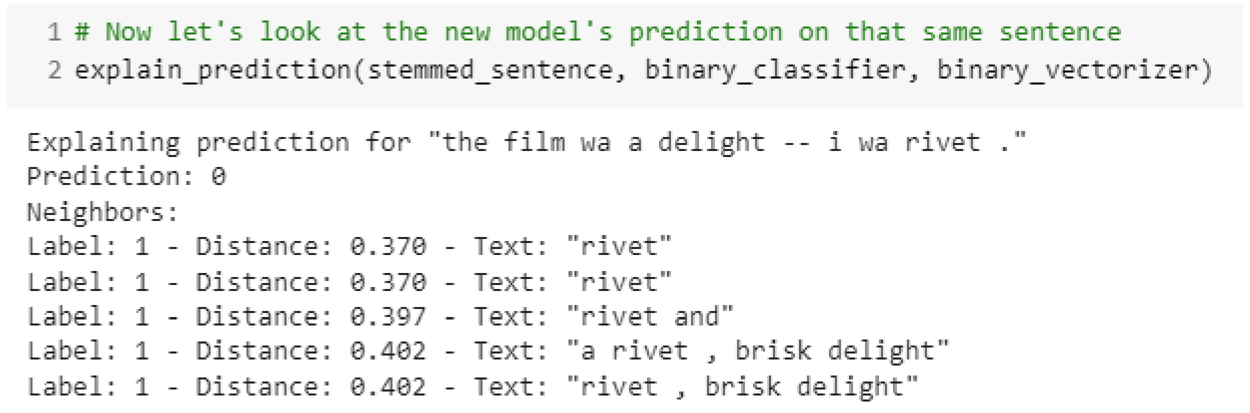
For K-nearest-neighbors, you can just look at the votes of the K neighbors.

Model confidence



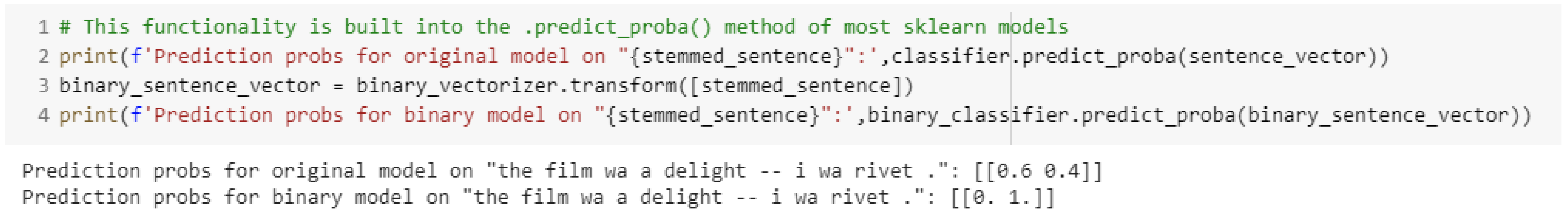
```python
# We can look at the original model's prediction on our sentence of interest
explain_prediction(stemmed_sentence, classifier, vectorizer)
```

```
Explaining prediction for "the film wa a delight -- i wa rivet ."
Prediction: 0
Neighbors:
Label: 1 - Distance: 0.442 - Text: "wa a better film"
Label: 1 - Distance: 0.459 - Text: "wa funni"
Label: 0 - Distance: 0.464 - Text: "but it wa n't ."
Label: 0 - Distance: 0.480 - Text: "wa n't enough"
Label: 0 - Distance: 0.482 - Text: "wa onli"
```



```python
# Now let's look at the new model's prediction on that same sentence
explain_prediction(stemmed_sentence, binary_classifier, binary_vectorizer)
```

```
Explaining prediction for "the film wa a delight -- i wa rivet ."
Prediction: 0
Neighbors:
Label: 1 - Distance: 0.370 - Text: "rivet"
Label: 1 - Distance: 0.370 - Text: "rivet"
Label: 1 - Distance: 0.397 - Text: "rivet and"
Label: 1 - Distance: 0.402 - Text: "a rivet , brisk delight"
Label: 1 - Distance: 0.402 - Text: "rivet , brisk delight"
```



```python
# This functionality is built into the .predict_proba() method of most sklearn models
print(f'Prediction probs for original model on "{stemmed_sentence}":',classifier.predict_proba(sentence_vector))
binary_sentence_vector = binary_vectorizer.transform([stemmed_sentence])
print(f'Prediction probs for binary model on "{stemmed_sentence}":',binary_classifier.predict_proba(binary_sentence_vector))
```

```
Prediction probs for original model on "the film wa a delight -- i wa rivet .": [[0.6 0.4]]
Prediction probs for binary model on "the film wa a delight -- i wa rivet .": [[0. 1.]]
```

Hyperparameters

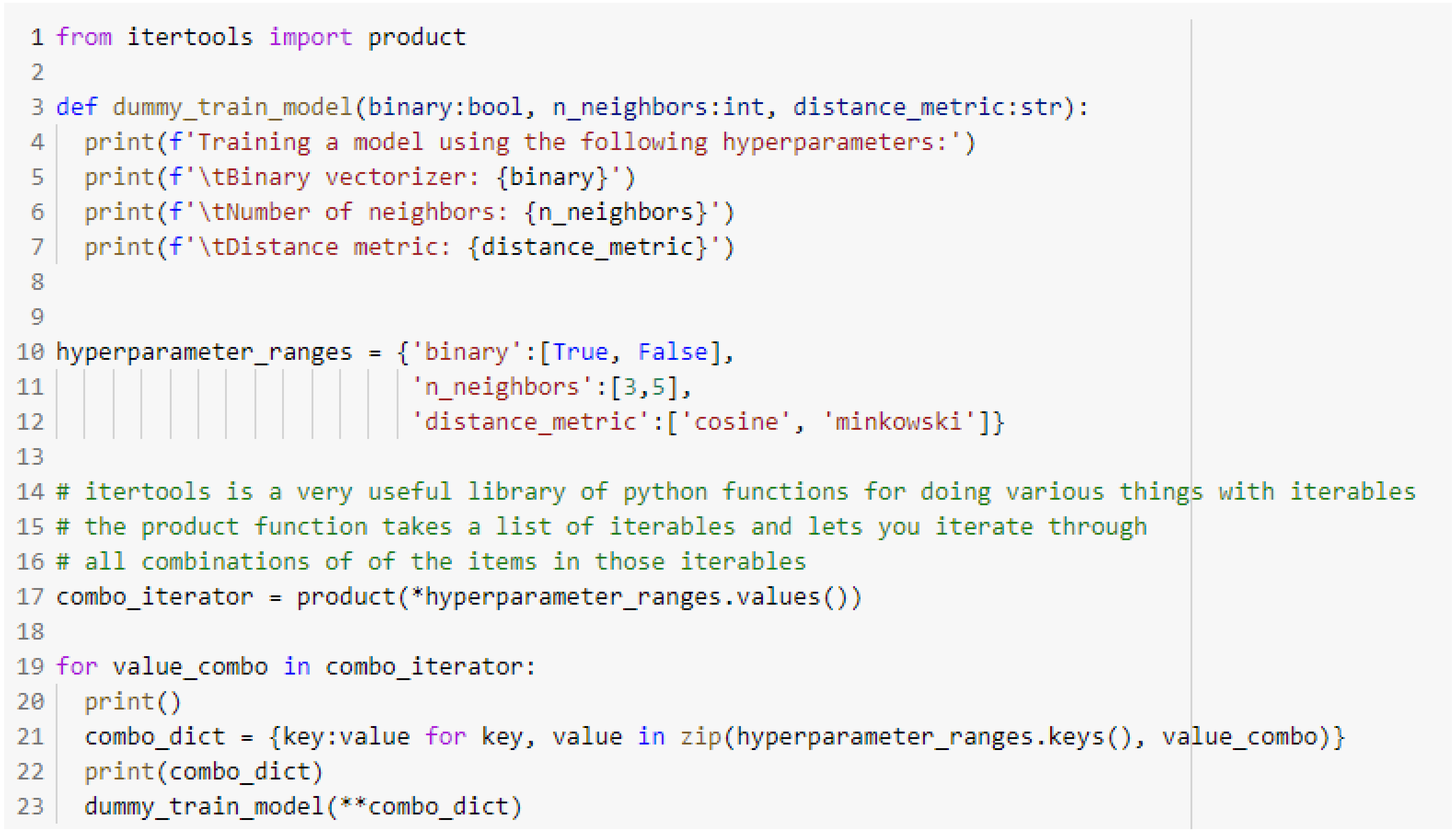
All these different choices are called “hyperparameters”

- How many neighbors to use
- What distance metric to use
- Whether to set binary=True or False in the vectorizer

A big part of model building is finding the best (or adequately okay) set of hyperparameters

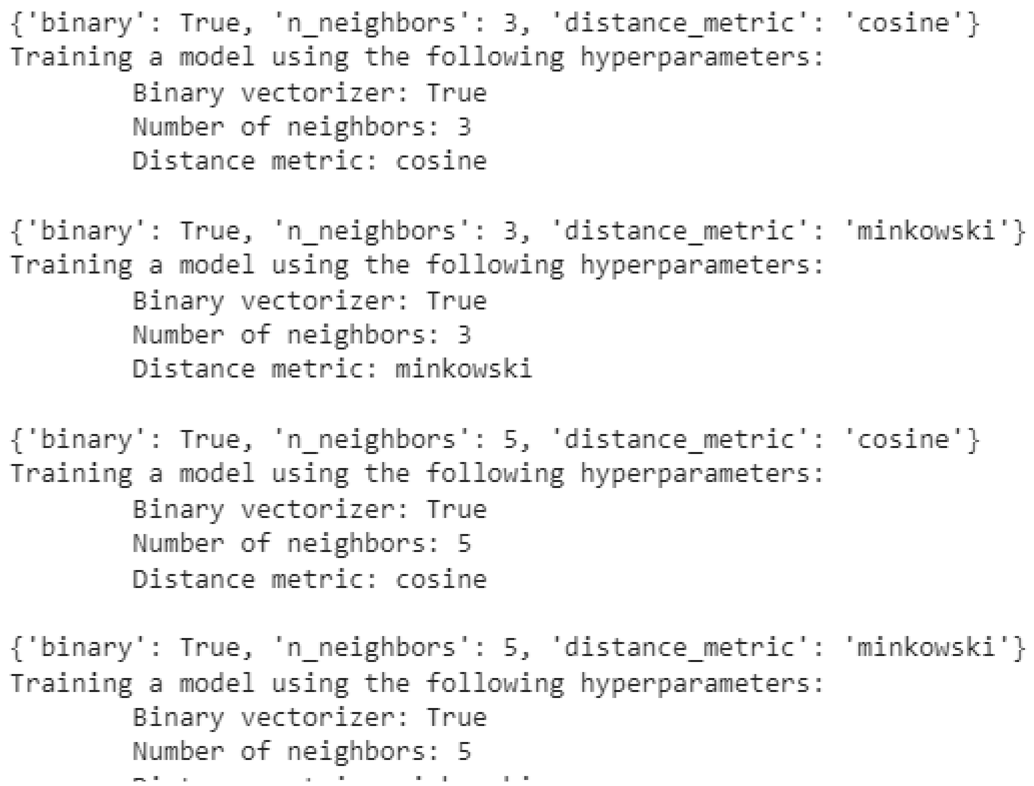
Simplest and most common approach is to just search exhaustively over space of possible values—called **grid search**

Hyperparameters



```python
from itertools import product

def dummy_train_model(binary:bool, n_neighbors:int, distance_metric:str):
  print(f'Training a model using the following hyperparameters:')
  print(f'\tBinary vectorizer: {binary}')
  print(f'\tNumber of neighbors: {n_neighbors}')
  print(f'\tDistance metric: {distance_metric}')


hyperparameter_ranges = {'binary':[True, False],
                         'n_neighbors':[3,5],
                         'distance_metric':['cosine', 'minkowski']}

# itertools is a very useful library of python functions for doing various things with iterables
# the product function takes a list of iterables and lets you iterate through
# all combinations of of the items in those iterables
combo_iterator = product(*hyperparameter_ranges.values())

for value_combo in combo_iterator:
  print()
  combo_dict = {key:value for key, value in zip(hyperparameter_ranges.keys(), value_combo)}
  print(combo_dict)
  dummy_train_model(**combo_dict)
```

Hyperparameters



```
{'binary': True, 'n_neighbors': 3, 'distance_metric': 'cosine'}
Training a model using the following hyperparameters:
        Binary vectorizer: True
        Number of neighbors: 3
        Distance metric: cosine

{'binary': True, 'n_neighbors': 3, 'distance_metric': 'minkowski'}
Training a model using the following hyperparameters:
        Binary vectorizer: True
        Number of neighbors: 3
        Distance metric: minkowski

{'binary': True, 'n_neighbors': 5, 'distance_metric': 'cosine'}
Training a model using the following hyperparameters:
        Binary vectorizer: True
        Number of neighbors: 5
        Distance metric: cosine

{'binary': True, 'n_neighbors': 5, 'distance_metric': 'minkowski'}
Training a model using the following hyperparameters:
        Binary vectorizer: True
        Number of neighbors: 5
```

Other things to know

How to use each set:

- Train on the training set
- Experiment on the dev set
- Leave the test set alone until the very end (notice we didn’t even use it)

When dealing with temporal data (which SST-2 is not, really)

- Never, ever, train on future data and test on past data
- Super common mistake in the wild

Concluding thoughts

New toolkit: Pandas

Pretty cool that we can already build models with what little we’ve learned so far. Non-parametric models so far, but we’re getting there.

When doing nearest-neighbor classification (and classification generally for 1 and 2):

1. How you choose to vectorize your text matters a lot
2. The distance metric you use matters a lot
3. Sometimes more sensible individual predictions don't translate to better performance