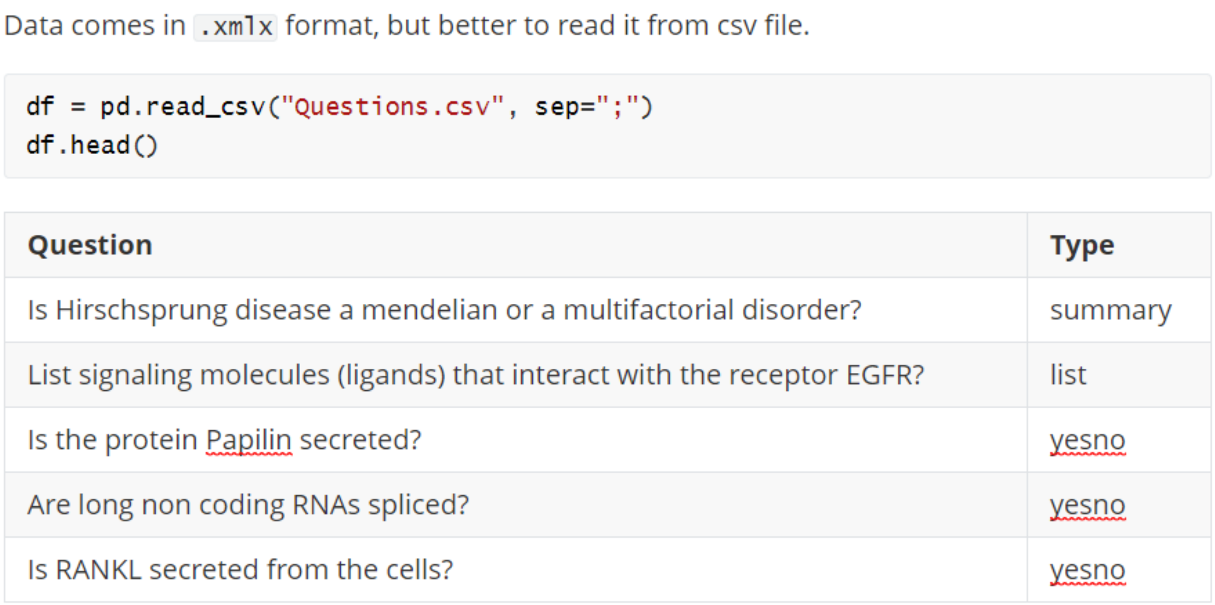
QUESTION CLASSIFIER

MATEUSZ DOROBEK

THE PROBLEM

DATA

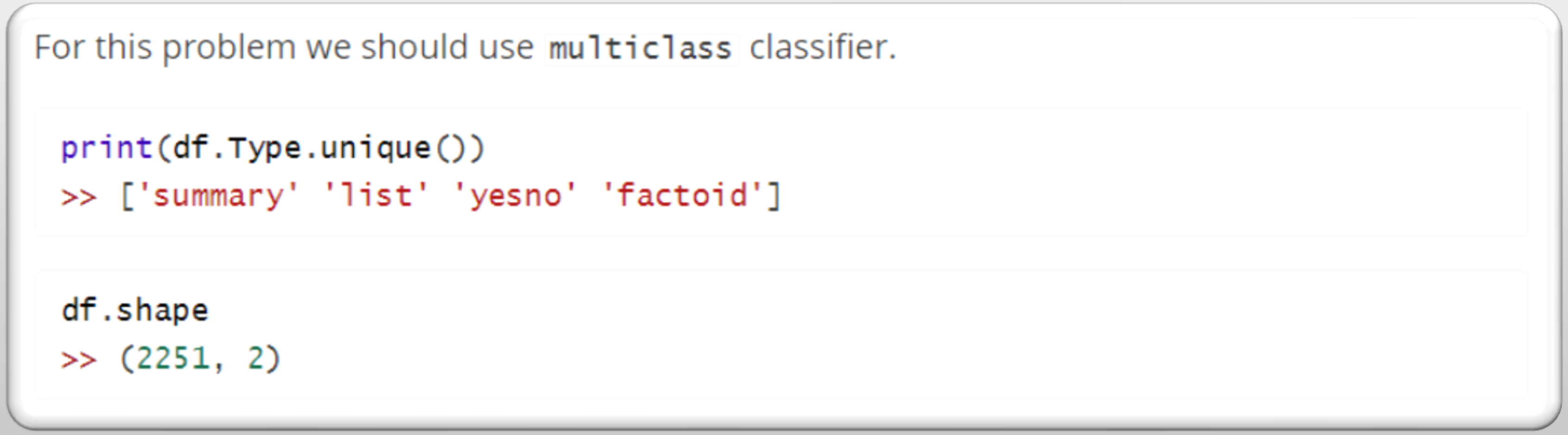


Data comes in `.xmlx` format, but better to read it from csv file.

```
df = pd.read_csv("Questions.csv", sep=";")
df.head()
```

Question	Type
Is Hirschsprung disease a mendelian or a multifactorial disorder?	summary
List signaling molecules (ligands) that interact with the receptor EGFR?	list
Is the protein Papilin secreted?	yesno
Are long non coding RNAs spliced?	yesno
Is RANKL secreted from the cells?	yesno

DATA



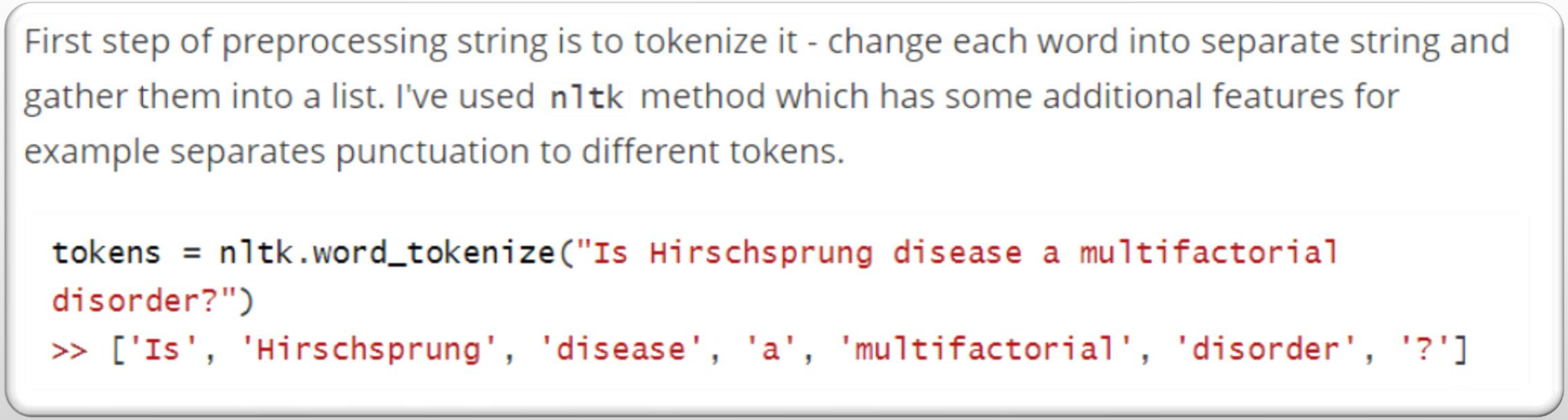
For this problem we should use `multiclass` classifier.

```
print(df.Type.unique())
>> ['summary' 'list' 'yesno' 'factoid']
```

```
df.shape
>> (2251, 2)
```

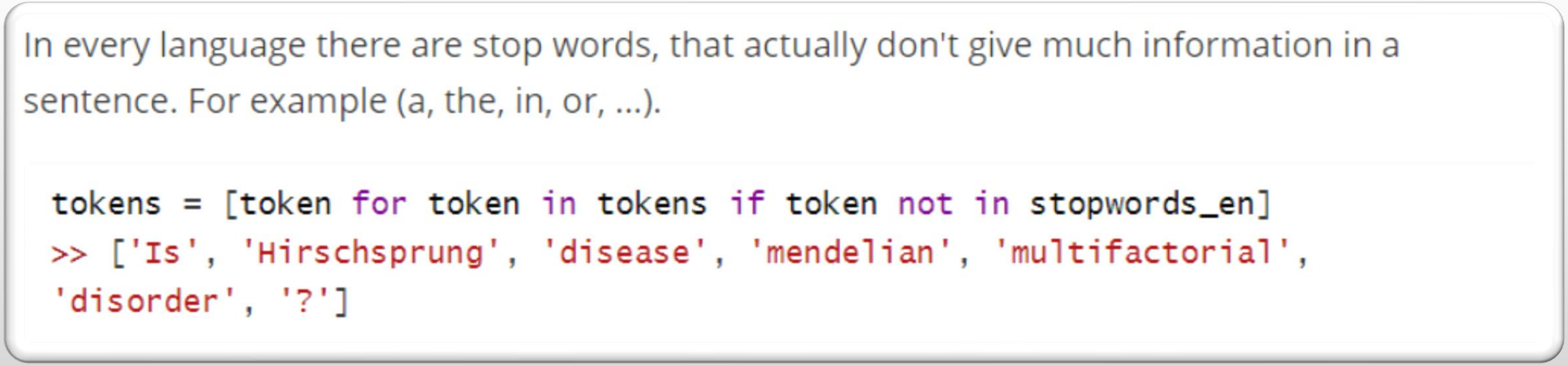
PREPROCESSING

TOKENIZING



First step of preprocessing string is to tokenize it - change each word into separate string and gather them into a list. I've used `nltk` method which has some additional features for example separates punctuation to different tokens.

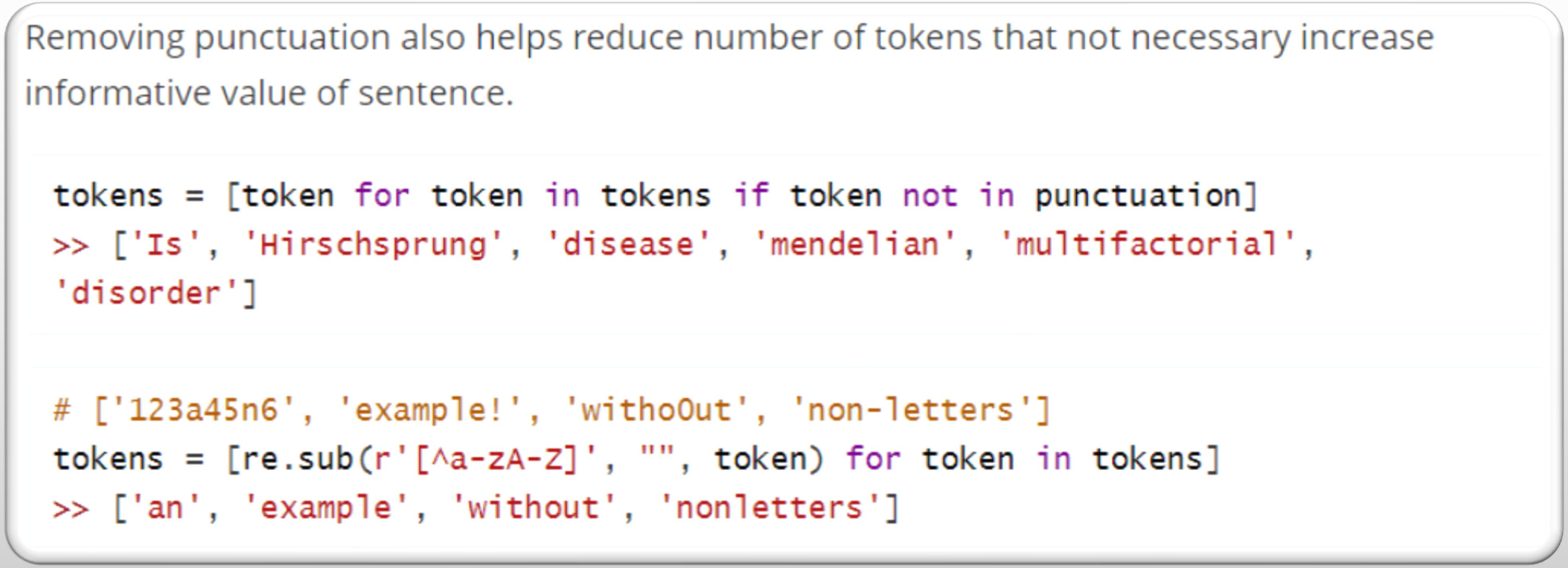
```
tokens = nltk.word_tokenize("Is Hirschsprung disease a multifactorial
disorder?")
>> ['Is', 'Hirschsprung', 'disease', 'a', 'multifactorial', 'disorder', '?']
```



In every language there are stop words, that actually don't give much information in a sentence. For example (a, the, in, or, ...).

```
tokens = [token for token in tokens if token not in stopwords_en]
>> ['Is', 'Hirschsprung', 'disease', 'mendelian', 'multifactorial',
'disorder', '?']
```

STOPWORDS

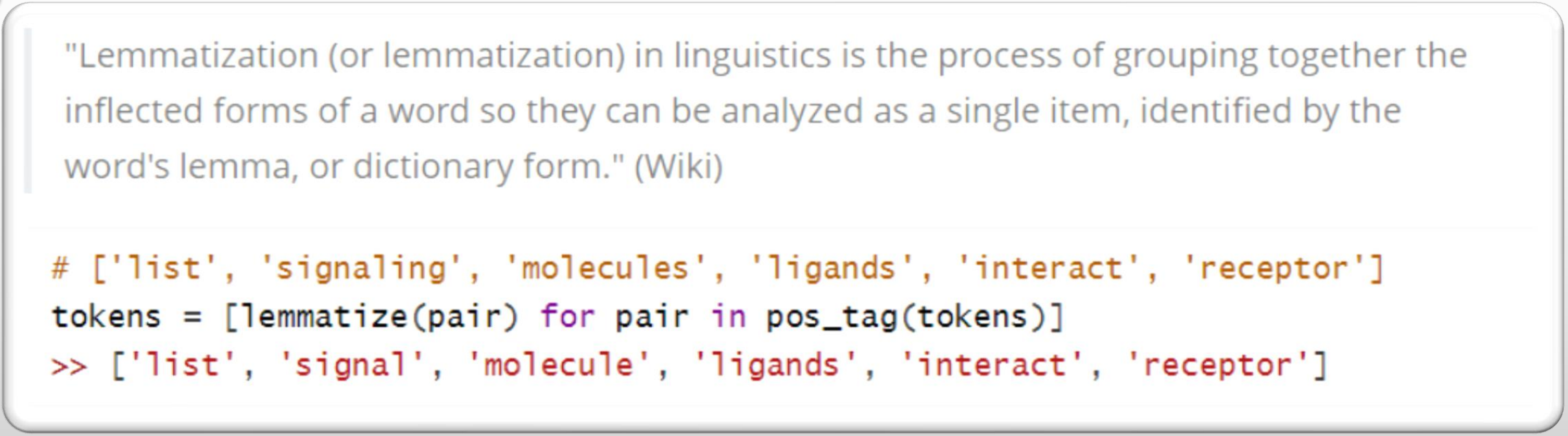


Removing punctuation also helps reduce number of tokens that not necessary increase informative value of sentence.

```
tokens = [token for token in tokens if token not in punctuation]
>> ['Is', 'Hirschsprung', 'disease', 'mendelian', 'multifactorial',
'disorder']
```

```
# ['123a45n6', 'example!', 'withoOut', 'non-letters']
tokens = [re.sub(r'[^a-zA-Z]', "", token) for token in tokens]
>> ['an', 'example', 'without', 'nonletters']
```

PUNCTUATION

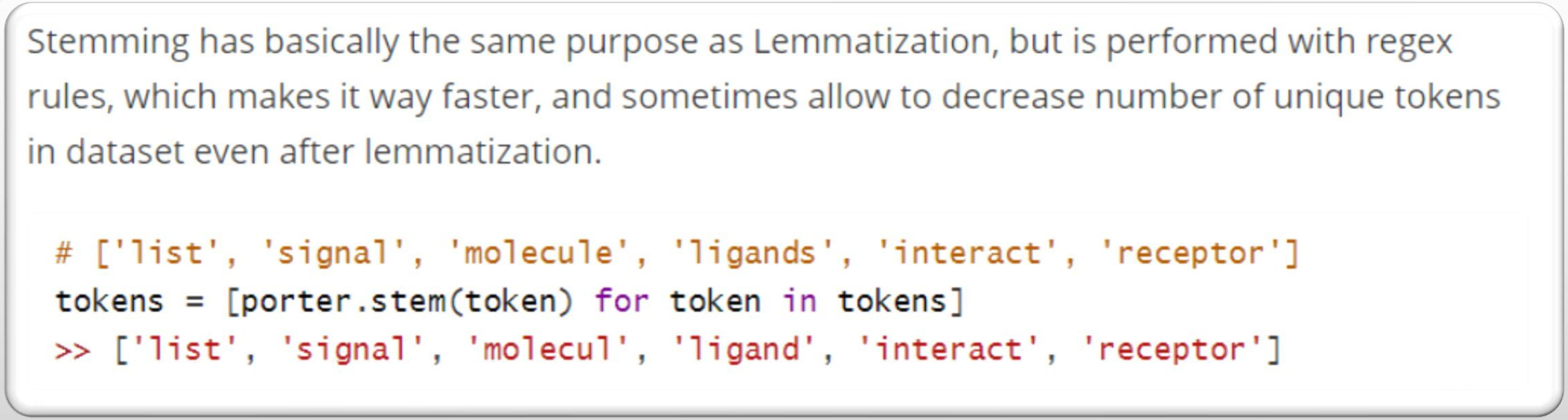


> "Lemmatization (or lemmatization) in linguistics is the process of grouping together the inflected forms of a word so they can be analyzed as a single item, identified by the word's lemma, or dictionary form." (Wiki)

```
# ['list', 'signaling', 'molecules', 'ligands', 'interact', 'receptor']
tokens = [lemmatize(pair) for pair in pos_tag(tokens)]
>> ['list', 'signal', 'molecule', 'ligands', 'interact', 'receptor']
```

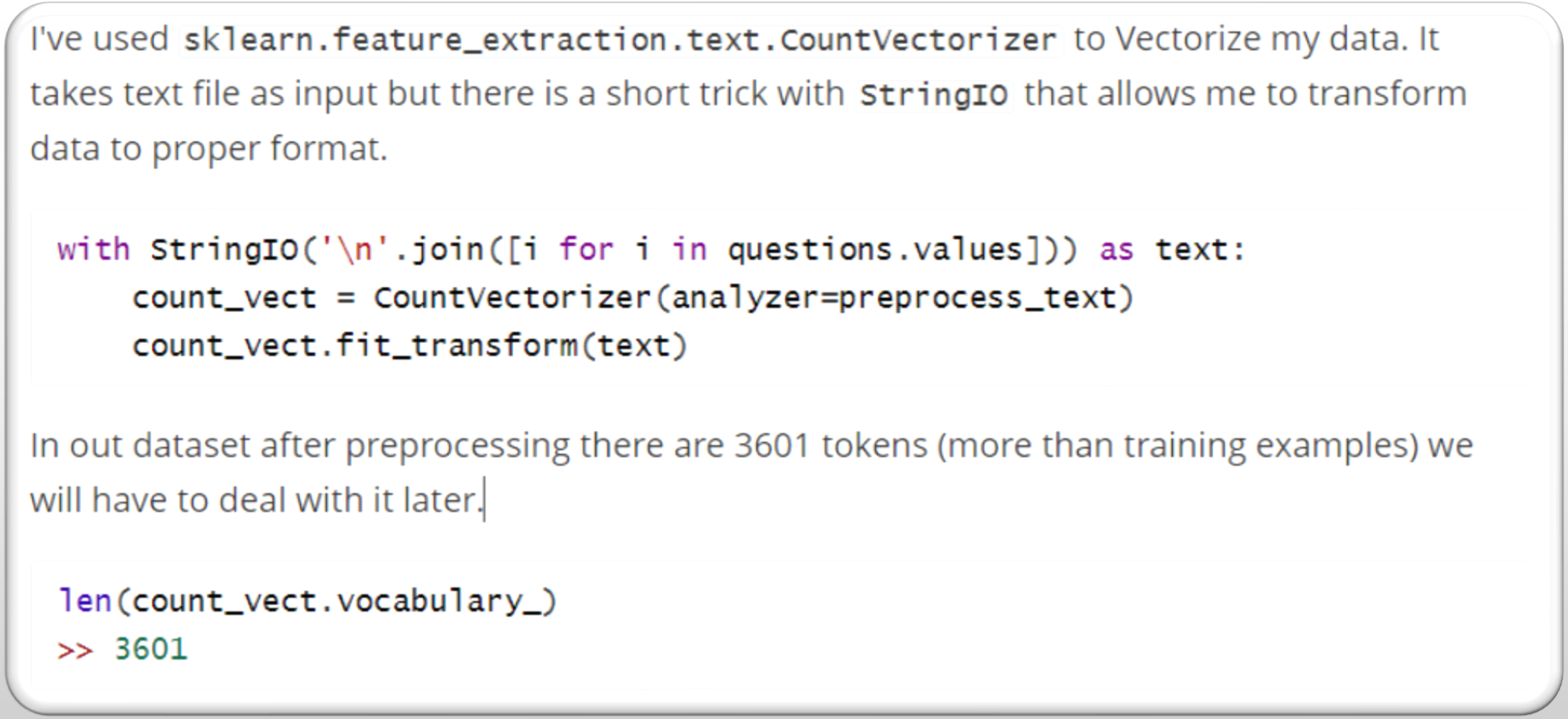
LEMMATIZATION

STEMMING



Stemming has basically the same purpose as Lemmatization, but is performed with regex rules, which makes it way faster, and sometimes allow to decrease number of unique tokens in dataset even after lemmatization.

```
# ['list', 'signal', 'molecule', 'ligands', 'interact', 'receptor']
tokens = [porter.stem(token) for token in tokens]
>> ['list', 'signal', 'molecul', 'ligand', 'interact', 'receptor']
```



I've used `sklearn.feature_extraction.text.CountVectorizer` to Vectorize my data. It takes text file as input but there is a short trick with `StringIO` that allows me to transform data to proper format.

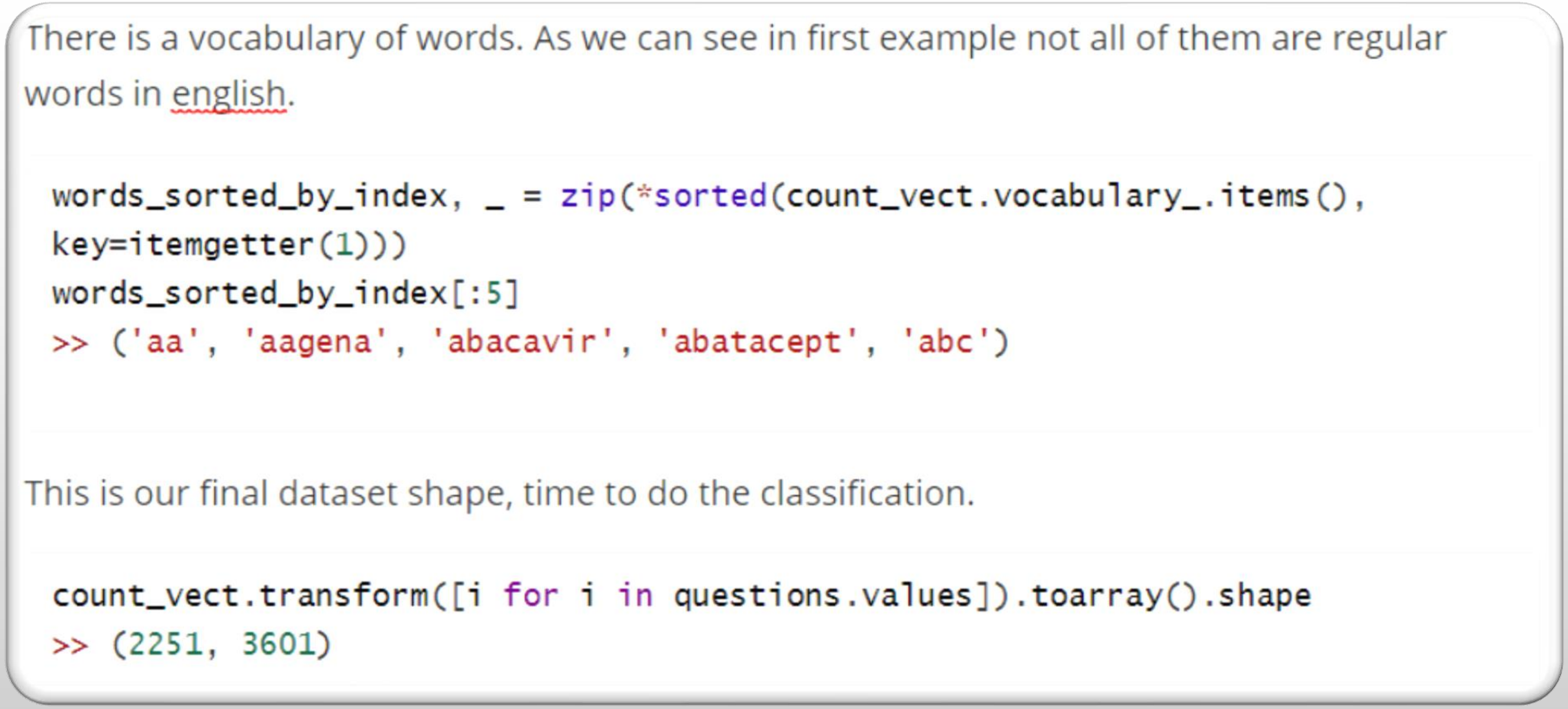
```
with StringIO('\n'.join([i for i in questions.values])) as text:
    count_vect = CountVectorizer(analyzer=preprocess_text)
    count_vect.fit_transform(text)
```

In out dataset after preprocessing there are 3601 tokens (more than training examples) we will have to deal with it later.

```
len(count_vect.vocabulary_)
>> 3601
```

VECTORIZATION

FINAL DATA SHAPE



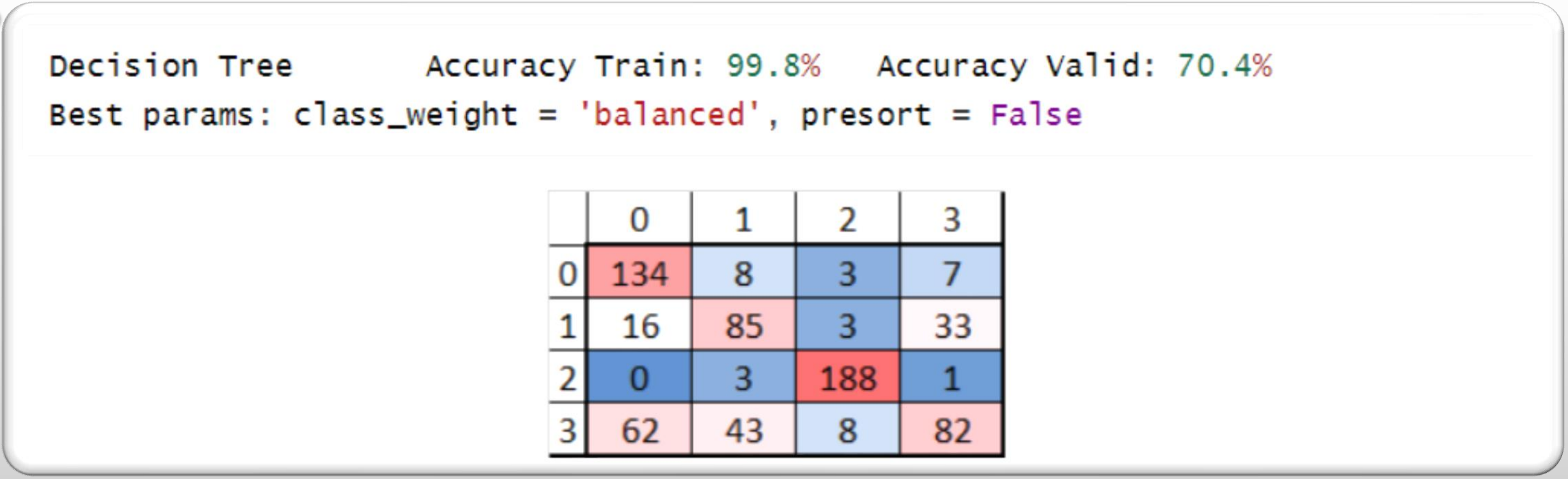
There is a vocabulary of words. As we can see in first example not all of them are regular words in english.

```
words_sorted_by_index, _ = zip(*sorted(count_vect.vocabulary_.items(),
key=itemgetter(1)))
words_sorted_by_index[:5]
>> ('aa', 'aagena', 'abacavir', 'abatacept', 'abc')
```

This is our final dataset shape, time to do the classification.

```
count_vect.transform([i for i in questions.values]).toarray().shape
>> (2251, 3601)
```

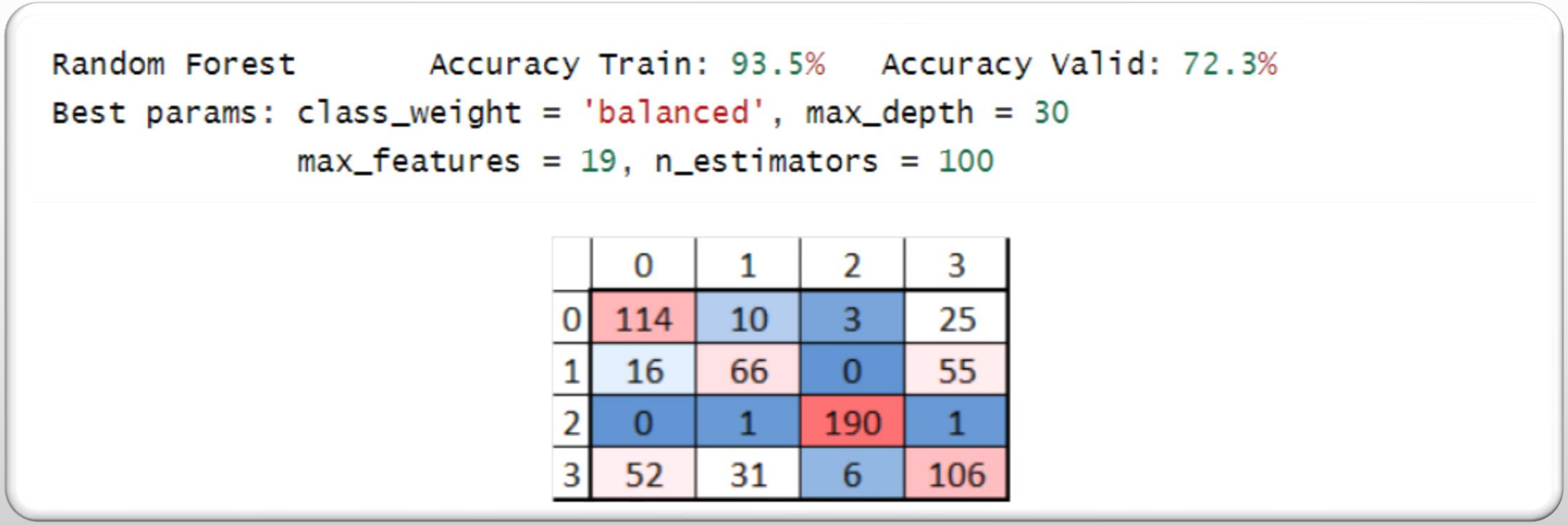
CLASSIFICATION



```
Decision Tree        Accuracy Train: 99.8%   Accuracy Valid: 70.4%
Best params: class_weight = 'balanced', presort = False
```

	0	1	2	3
0	134	8	3	7
1	16	85	3	33
2	0	3	188	1
3	62	43	8	82

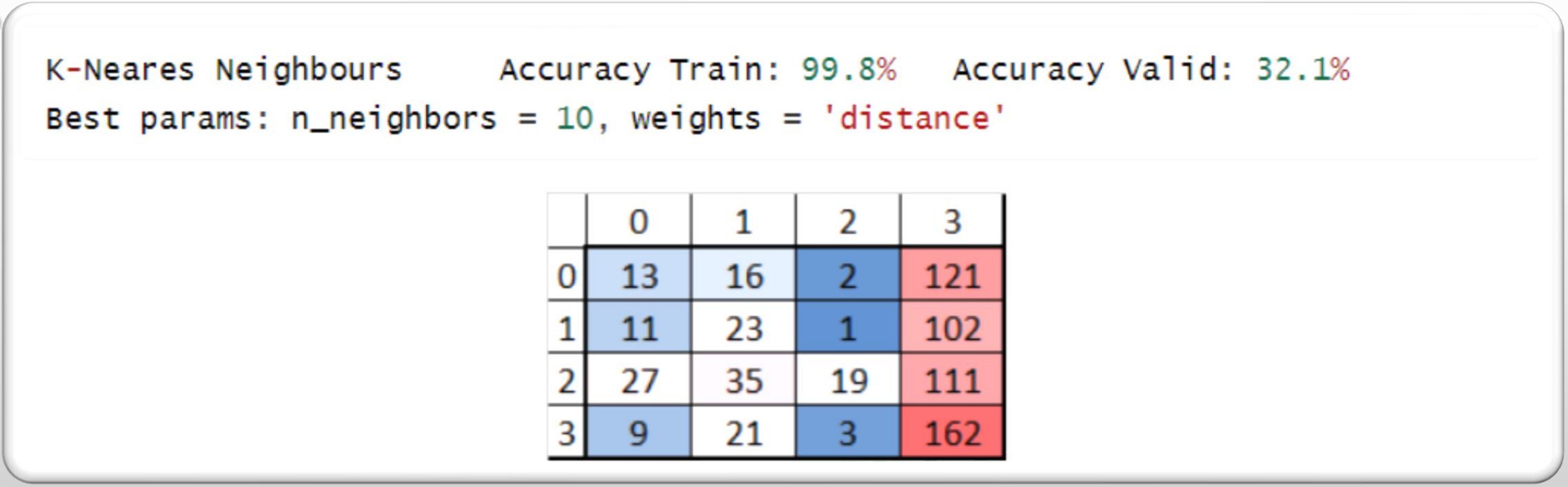
DECISION TREE



```
Random Forest       Accuracy Train: 93.5%   Accuracy Valid: 72.3%
Best params: class_weight = 'balanced', max_depth = 30
             max_features = 19, n_estimators = 100
```

	0	1	2	3
0	114	10	3	25
1	16	66	0	55
2	0	1	190	1
3	52	31	6	106

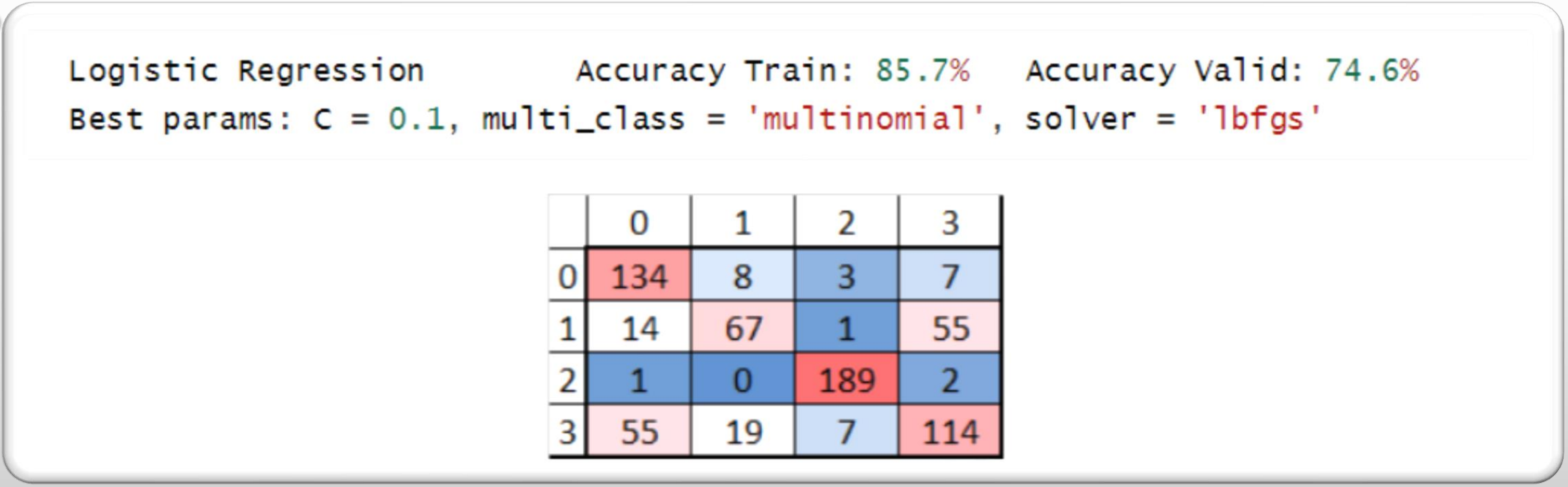
RANDOM FOREST



```
K-Neares Neighbours     Accuracy Train: 99.8%   Accuracy Valid: 32.1%
Best params: n_neighbors = 10, weights = 'distance'
```

	0	1	2	3
0	13	16	2	121
1	11	23	1	102
2	27	35	19	111
3	9	21	3	162

K-NEARES NEIGHBOURS



```
Logistic Regression          Accuracy Train: 85.7%   Accuracy Valid: 74.6%
Best params: C = 0.1, multi_class = 'multinomial', solver = 'lbfgs'
```

	0	1	2	3
0	134	8	3	7
1	14	67	1	55
2	1	0	189	2
3	55	19	7	114

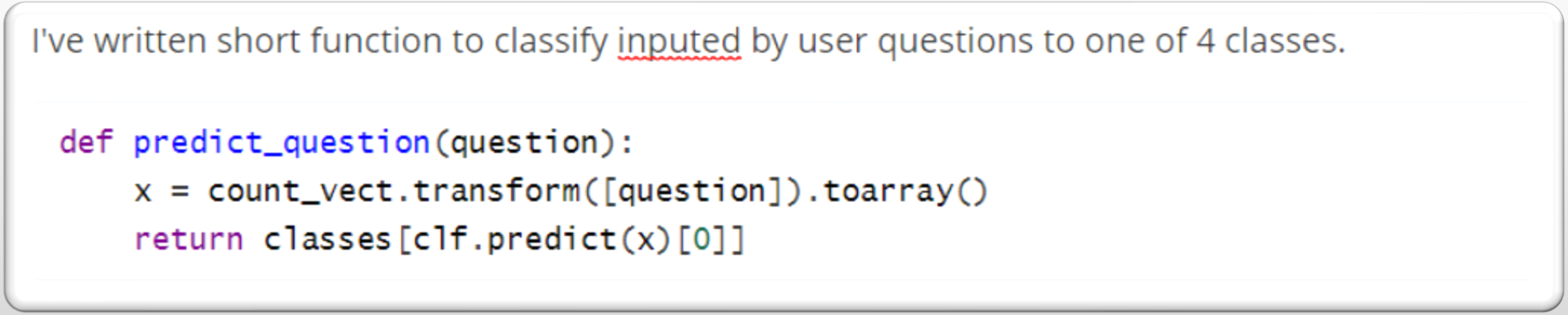
LOGISTIC REGRESSION

NOTES

- I'VE TESTED THAT ***PCA*** DOESN'T IMPROVE PERFORMANCE OF ANY OF CLASSIFIERS.

- USING ***STANDARSCALER()*** WASN'T A GOOD DUE TO BINARY CHARACTER OF DATA.

- MY VALIDATION METRIC WAS ***ACCURACY*** DUE TO EVEN DISTIBUTION IN CLASS.



I've written short function to classify inputed by user questions to one of 4 classes.

```
def predict_question(question):
    x = count_vect.transform([question]).toarray()
    return classes[clf.predict(x)[0]]
```

RESULTS

RESULTS

- *LIST TWO OF YOUR FAVOURIE FILMS.* LIST
- *WHERE ARE YOU?* FACTOID
- *HOW OLD ARE YOU?* SUMMARY

RESULTS

- *DO YOU LIKE TO STUDY?* YESNO
- *HOW DO YOU FEEL RRIGHT NOW?* SUMMARY
- *WHAT IS YOUR NAME?* SUMMARY

THANK YOU