How do I pre-process text data from Yelp reviews so I can analyze it?

Introduction

Business Context. You are a business consultant for small and medium-sized businesses with a large number of customers. Examples of such businesses might include a quick-dining restaurant, clothing store, or online distributor of hobby equipment. You would like to help your small businesses understand what factors are driving positive and negative customer experiences. Customers are often unwilling to give direct feedback, but do leave large numbers of online reviews on websites such as Yelp, Amazon, and so on. You would like to develop a service that would allow businesses to quickly obtain useful summaries of their reviews across such websites. Such a service would allow your clients to answer questions like: "What are the most important factors driving negative reviews?" or "Did a recent policy change improve our reviews?"

Business Problem. Your main task is to **wrangle a dataset of text reviews and engineer relevant features in order to facilitate subsequent analysis and model building**.

Analytical Context. Text data is highly unstructured, and often requires pre-processing before we can gather any business insights from it. We will be leveraging tools from **natural language processing (NLP)** in order to help us process this data and generate new features that can be used for analytics or model building.

The case will proceed as follows: we will (1) introduce basic steps of pre-processing like word tokenization and text visualization; (2) introduce key tools in feature extraction such as n-grams, count representations, stop words, and TF-IDF; (3) leverage these tools to perform exploratory data analysis; and finally (4) look at pure text wrangling tools like regular expressions.

Context about NLP

Some of the most famous success stories involving NLP come from Google, where it is used to give very good responses to vague or misspelled internet searches, as well as fairly comprehensible automatic translations of plain text.

There is often quite a bit of misconception surrounding NLP and its place within machine learning as a whole. Many of the uninitiated think that NLP is just machine learning applied to words. However, this is far from true. NLP has two characteristics which make it intractable for naive applications of machine learning:

Challenge 1: Extraordinarily high dimensionality

Consider the book *War and Peace*. It has 3 million characters. Can we view this as a long vector of strings taking values in a 3-million-dimensional space, and then apply machine learning methods here? This is a bad idea for two reasons:

1. Basic approaches have terrible performance in such high-dimensional spaces
2. These approaches "miss out" on some important rules about language that we all know; e.g. that "don't" and "do not" mean the same thing

As a result, a huge amount of NLP involves finding ways to summarize incredibly long vectors in concise ways, so that we can tractably explore, analyze, and model build with them later.

Challenge 2: Text is context specific

For example, the word *queen* has many uses in English that are both *very different* and *common*:

1. The ruler of a country
2. A size of mattress
3. The most powerful piece in chess
4. The mother insect in certain types of insect colonies

General purpose libraries will need to deal with all of these, but reviews for mattresses will almost always be about the second. This type of mismatch can result in misleading results that can easily be fixed by a team that is familiar with the underlying NLP computations.

Pre-processing and standardization

Standardizing text involves many steps. Some of these include:

1. Simple error correction (e.g. removing ASCII coding errors) 2, Feature creation (e.g. labeling nouns and verbs)
2. Replacing words and sentences altogether (e.g. standardizing spelling by changing "yuuuuuuck!" to "yuck", or more extreme steps such as replacing words with near synonyms)

In a broad sense, standardization is similar to data wrangling with more conventional data; we are fixing errors, removing outliers, and transforming features. However, the details in NLP tend to be more complicated. We will be using Python's Natural Languages Toolkit (`nltk`) library. This library has functions that do most of the basics in NLP.

NOTE: *Many text wrangling pipelines start a little before we do, with initial "cleaning" steps that involve things like: converting all characters to lower case, expanding contractions, etc. Our main reason for skipping this initial step is that you must make some important and very context-sensitive choices in these steps. For example, auto-correcting the spelling of a document can be very helpful in reducing noise (e.g. if there are important and hard-to-spell words in the document, like "Poughkeepsie") but can also destroy the most important part of the signal (e.g. auto-correcting brand names of companies like "Rappi" to "rapping"). Similarly, the steps we outline here may need to be adapted appropriately for other contexts, such as other languages, computer programs, legal documents, etc.*

In [63]:
```
import nltk # imports the natural language toolkit
nltk.download('punkt')
nltk.download('stopwords')
import pandas as pd
import numpy  as np
```

```
import string
import plotly
from nltk.stem import PorterStemmer
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [4]:
```
# LOADING THE DATASET AND SEEING THE DETAILS
data = pd.read_csv('sdata.csv')
data.head()
```

Out[4]:

	review_id	user_id	business_id	stars	date
0	vkVSCC7xljjrAI4UGfnKEQ	bv2nCi5Qv5vroFiqKGopiw	AEx2SYEUJmTxVVB18LlCwA	5	2016-05-28
1	n6QzIUObkYshz4dz2QRJTw	bv2nCi5Qv5vroFiqKGopiw	VR6GpWIda3SfvPC-lg9H3w	5	2016-05-28
2	MV3CcKScW05u5LVfF6ok0g	bv2nCi5Qv5vroFiqKGopiw	CKC0-MOWMqoeWf6s-szl8g	5	2016-05-28
3	IXvOzsEMYtiJI0CARmj77Q	bv2nCi5Qv5vroFiqKGopiw	ACFtxLv8pGrrxMm6EgjreA	4	2016-05-28
4	L_9BTb55X0GDtThi6GlZ6w	bv2nCi5Qv5vroFiqKGopiw	s2I_Ni76bjJNK9yG60iD-Q	4	2016-05-28

In [5]:
```
AllReviews = data['text']
AllReviews.head()
```

Out[5]:
```
0    Super simple place but amazing nonetheless. It...
1    Small unassuming place that changes their menu...
2    Lester's is located in a beautiful neighborhoo...
3    Love coming here. Yes the place always needs t...
4    Had their chocolate almond croissant and it wa...
Name: text, dtype: object
```

Tokenizing sentences

Just like CSV data is composed of features, text data is composed of sentences. Thus, a natural first step is what is known as **sentence tokenization**: splitting a long document into its component sentences. At first this might seem trivial: just split whenever you see a period. Unfortunately, the same symbol is used in other ways in English (e.g. to mark an abbreviation, as part of ellipses, etc.), and so slightly more care is required. Fortunately, there are packages that will do this for us. Within `nltk`, we can use the `nltk.sent_tokenize()` function.

Exercise 1:

Give an example of a question we might be able to answer with this sort of data, and another question that we'd need additional data to answer. Assume for now that all of the reviews are coming from one business.

Sample Answer. A human reading this might be able to figure out the most appreciated features of a business by:

1. Looking for features that are consistently praised (e.g. "great customer service")
2. Looking for features that appear most often in positive reviews

On the other hand, it would be very difficult to make any conclusions about how the business is faring with respect to locals vs. tourists based on this dataset.

Here, each "document" is just a single review. Let's take a look at the first "document" review in our dataset, and tokenize it:

In [6]:
```
# Print text of first Yelp review
AllReviews[0]
```

Out[6]:
```
"Super simple place but amazing nonetheless. It's been around since the 30's and they still serve the same thing they started with: a bologna a nd salami sandwich with mustard. \n\nStaff was very helpful and friendl y."
```

In [7]:
```
# sentence tokenization
sentences = nltk.sent_tokenize(AllReviews[0])
for sentence in sentences:
    print(sentence)
    print()
```

```
Super simple place but amazing nonetheless.

It's been around since the 30's and they still serve the same thing the y started with: a bologna and salami sandwich with mustard.

Staff was very helpful and friendly.
```

Tokenizing words

Having split documents into sentences, we now split sentences into individual words. As with sentence tokenization, there is (i) a pretty good heuristic (split on spaces), (ii) a number of weird exceptions (e.g. compound words), and (iii) an existing package that does the job fairly well.

We use the `nltk.word_tokenize()` function from `nltk`:

In [8]:
```
sentences = nltk.sent_tokenize(data['text'][1])
for sentence in sentences:
    words = nltk.word_tokenize(sentence)
```

```
    print(sentence)
    print(words)
    print()
```

```
Small unassuming place that changes their menu every so often.
['Small', 'unassuming', 'place', 'that', 'changes', 'their', 'menu', 'e
very', 'so', 'often', '.']

Cool decor and vibe inside their 30 seat restaurant.
['Cool', 'decor', 'and', 'vibe', 'inside', 'their', '30', 'seat', 'rest
aurant', '.']

Call for a reservation.
['Call', 'for', 'a', 'reservation', '.']

We had their beef tartar and pork belly to start and a salmon dish and
lamb meal for mains.
['We', 'had', 'their', 'beef', 'tartar', 'and', 'pork', 'belly', 'to',
'start', 'and', 'a', 'salmon', 'dish', 'and', 'lamb', 'meal', 'for', 'm
ains', '.']

Everything was incredible!
['Everything', 'was', 'incredible', '!']

I could go on at length about how all the listed ingredients really mak
e their dishes amazing but honestly you just need to go.
['I', 'could', 'go', 'on', 'at', 'length', 'about', 'how', 'all', 'th
e', 'listed', 'ingredients', 'really', 'make', 'their', 'dishes', 'amaz
ing', 'but', 'honestly', 'you', 'just', 'need', 'to', 'go', '.']

A bit outside of downtown montreal but take the metro out and it's less
than a 10 minute walk from the station.
['A', 'bit', 'outside', 'of', 'downtown', 'montreal', 'but', 'take', 't
he', 'metro', 'out', 'and', 'it', "'s", 'less', 'than', 'a', '10', 'min
ute', 'walk', 'from', 'the', 'station', '.']
```

Exercise 2:

Conduct an exploratory analysis of the sizes of reviews: find the shortest and longest reviews, then plot a histogram showing the distribution of review lengths.

Answer. One possible solution is shown below:

In [9]:
```
review_words_lengths = AllReviews.apply(lambda x: len(nltk.word_tokenize(x)))
```

In [10]:
```
min(review_words_lengths)
```

Out[10]:
```
2
```

In [11]:
```
## This review has been written in a different language
AllReviews[review_words_lengths[review_words_lengths == 2].index]
```

Out[11]:
```
6687    在拉斯维加斯买了新房子，本来想安装布艺窗帘， 但是经过小刘的介绍，知道拉斯维加斯特别热，布艺...
Name: text, dtype: object
```

In [12]:
```
max(review_words_lengths)
```

Out[12]:
```
1148
```

In [13]:
```
AllReviews[review_words_lengths[review_words_lengths == max(review_words_lengths)].index]
print(AllReviews[9349])
```

```
Alright...this is a bit of a mixed bag review-wise...not because there is some good and bad, but because...well...read the review.

It was my birthday...or a few days later.  A friend and I were celebrating.  Smallman Galley was on my "list" to try.  I didn't know what to expect.  But I think I had in my mind a vision of what this "chef incubator" was going to be.

Things I didn't expect:  picnic table seating, 4 "store fronts", people lined up ordering from the counter.  So...go into it with that expectat
```

ion. I kind of love the idea of the place, but if you were thinking to woo your lady love (or man love, whatevs) with a candle-lit dinner in a cozy restaurant with amazing service...well, you'd miss the mark.

I had to wait. My friend was late. I was sitting at the bar, so I didn't really care. I perused the menu. First drink that popped out at me was a Primus song. "Jerry Was a Race Car Driver" reportedly had: Amara Sfumato Rabarbaro, Aperol, Genepy des Alpes, pluot syrup, and La Croix.

First impressions: I don't know what any of those things are. Not one. Hadn't heard of them. Hadn't had them. Didn't know how to pronounce them. The only word I recognized was "syrup". I vaguely recalled 'des' might have meant "of" in friench. Or Belgian. Whatever. Completely incomprehensible drink while you wait for a friend? TAKE ALL MY MONEY!

It was really good. Served in a tall glass with a lemon garnish, I sat and drank my fancy unpronouncably ingrediented drink in suave sophistication.

I texted a picture to my friend. "You're there already?" Yeah...because that's when we said we'd meet.

Whatever. I finished that drink and went two lines up on the drink menu. "Vilified" This promised brown butter-washed bourbon, sage-sweet potato syrup, and black walnut bitters. So much effort went into this drink. Who do they get to wash the bourbon in brown butter? How many natives died making the sage-sweet potato syrup? And I don't even know how you make a bitter let alone a black walnut bitter. This drink was described so pretentiously I had to lie to the bartender and tell him my name was Thurston J. Howell before he'd pour it for me. It was great. I would have had another. Wait...I did have another. I think. Things were starting to get a little...hazy. It had an orange garnish. It didn't photograph well.

I texted a picture of vilified to my friend. "What? You're on your second drink?" I was. "You're going to be hammered by the time I get there." I was.

Whatever. I finished that drink and and moved on to "Cyco Vision". Redemption rye, Yellow Chartreuse, pistachio orgeat, lemon, cardamom bitters, egg white. And listen...I'm a sucker for egg whites in my drink. I can't explain why. Rye? Awesome. Yellow Chartreuse? Confusing. Orgeat? Let's be honest. When I got to orgeat it sounded like orgy and I might have chuckled to myself and just ordered the drink because of the orgeat. I don't even know what orgeat is. But...it also reminded me of Xergiok from Adventure Time. I had to have this drink.

I had this drink. It was good. It was honestly my least favorite of the three but it was good. I took a picture...it looked like a malt. Served in a wide goblet, It had a thick foam on top and the bartender drew a little design with ...maybe with the orgeat...I don't know. Fancy!

I texted my friend the picture. "Jesus, I'm almost there, slow down!" I did.

And then my friend arrived and I recommended the Vilified and she hated it, so I drank hers and she took the "Superman" that I ordered, which was good, but I wasn't in love with it.

And then the bartender accidentally made someone another drink and it wasn't what he ordered and so she was going to dump it, but I said, "What do you do with a drink when it's the wrong one?" and she said, "Dump it, unless you want it." I did. I don't even know what it was. I think it was good. It's not completely clear.

And so now...Well now I've had five or six drinks in an hour and a half and I'm pretty much open for business, but it's time to eat, because god forbid we drink on an empty stomach. TOO LATE!! IT WAS TOO LATE!! But I walked past the previously-mentioned store fronts on my way to the bathroom and brought back a few ideas...

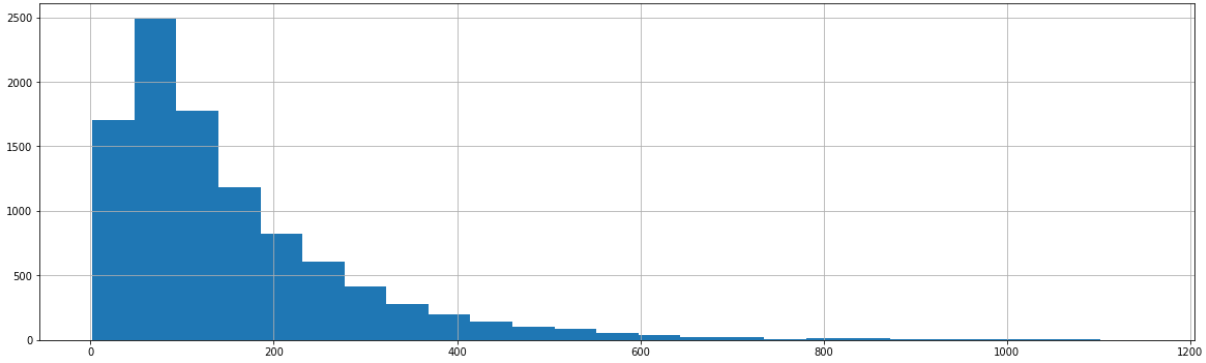
We ended up eating shrimp and grits (fairly certain this was delicious, but I have no idea which place I got it from) and poutine (again...and again).

So...I guess the mixed bag is that 1) I really don't have a clear memo

```
ry of the food.  I know it was good.  I just don't know where I got it, but 2) I have a very strong sense of the drink menu.  And it was a good drink menu.

Go for drinks!

I will be back.  What I remember of the food was that it was really good.  I know what to expect (no waitress, first come first serve seating, picnic table style).  Outstanding place to go with a group of friends and just sort of sprawl out across a picnic table and drink and try new foods.
```

In [14]:
```
## Setting the resolution for better clarity
from pylab import rcParams
rcParams['figure.figsize'] = 20, 6
review_words_lengths.hist(bins = 25)
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62320aa9b0>
```



Text visualization with word clouds

Just like visualization is crucial for standard CSV data, it is also important for text data. But text doesn't lend itself to histograms or scatterplots the way that numerical or even categorical data do. In such cases, **word clouds** are a common and useful tool:

In [15]:
```
# Importing the required parameter for plotting
import matplotlib.pyplot as plt
from wordcloud import WordCloud
word_cloud_text = ''.join(data.text)
wordcloud = WordCloud(max_font_size=100, max_words=100, background_colo
r="white",\
                      scale = 10,width=800, height=400).generate(wo
rd_cloud_text)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

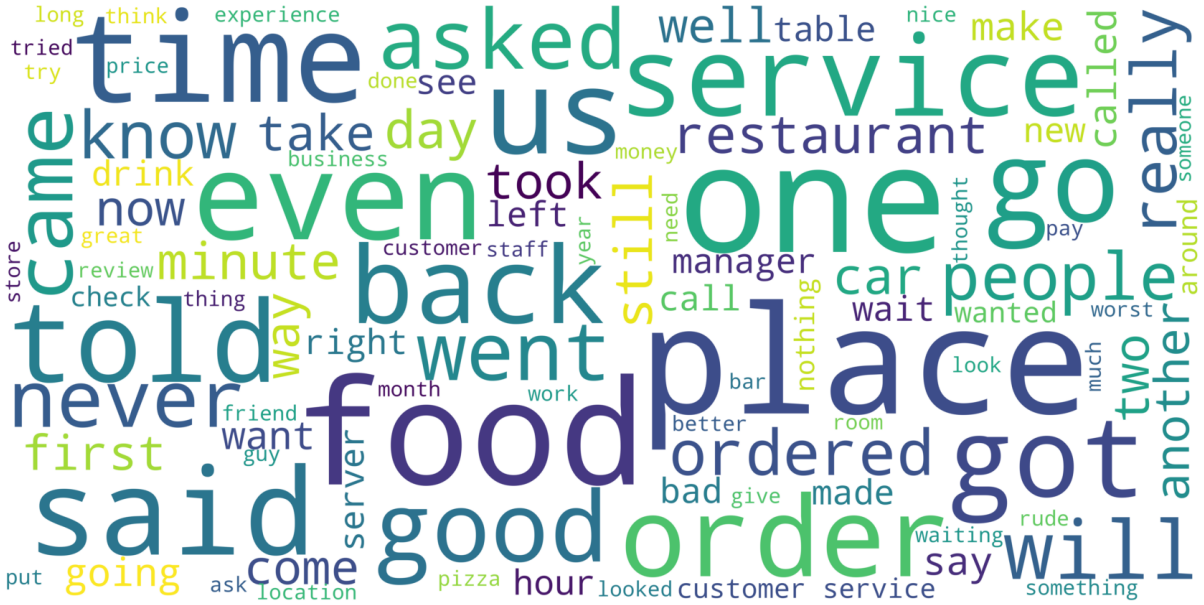
Exercise 3:

Write a function `word_cloud_rating(data, star_value)` that constructs a word cloud from the subset of `data` that exhibit a certain `star_value`. Visualize the results of this function for 1-star reviews.

Answer. One possible solution is given below:

In [16]:
```
## Setting the resolution for better clarity
from pylab import rcParams
rcParams['figure.figsize'] = 30, 60

def word_cloud_rating(data,star_value):

    data_filtered = data[data.stars == star_value] #filtering according
 to the star value
    Reviews = data_filtered.text

    Reviews_text = ' '.join(Reviews.values) #joining all the words toge
ther

    # Creating a word cloud object
    wordcloud = WordCloud(max_font_size=100, max_words=100, background_
color="white",\
                          scale = 10,width=800, height=400).generate(Re
views_text)


    # Plotting the generated word cloud
    plt.figure()
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()
```

In [17]:
```
word_cloud_rating(data, 1)
```



Exercise 4:

The word "good" seems to appear quite frequently in the negative reviews. Investigate why that is and come up with a reasonable explanation.

Answer. Let's look at the first 5 reviews or so with 1-star ratings to see if there are any discernible patterns:

In [18]:
```
[each for each in data[data.stars == 1].text if 'good' in each][:5]
```

Out[18]:
```
['after reading the reviews on yelp, my boyfriend and i decided to give
this place a try, especially since we are on a mission to try ramen in
every city we visit.\n\nwe were really excited, as when we were in NYC,
we had really good ramen. \n\nlistening to the reviews we started off w
ith the gyoza appetizer - which was not as good as everyone says it is.
It was pan fried with the thin layer of skin, but there was an aftertas
te of soap water. Which I should have known, since the glass of water t
hey served us was disgusting!\n\nWe had ordered the jumbo sumo ramen -
```

beware, it is a very big bowl of noodles, which we did not expect.\n\nT
he sumo ramen came with a half boil egg, seaweed, pork belly, and other
veggies. The noodles and pork belly was good. Soup was tasteless.\n\nDe
finitely would not go back again.',
 "Came here for a friend's birthday. She had looked it up and thought i
t sound good. First of all, when we made reservations, they only had a
7pm time slot for us (we were looking for 8pm) and when we were there,
we were seated in the lower area of the restaurant, which was pretty em
pty.\n\nThere was an extensive menu, everything on the menu sound good,
but when we got our food - there was nothing special, especially for wh
at we paid for it. \n\nThe pizza's are huge portions - so if you order
it, make sure you share it.\n\nThere are 3 risottos on the menu, the se
asonal vegetable one is the better one. Some of the pastas were portion
ed differently, some were huge, others were tiny - so be careful!\n\nTh
e food came out really fast after we ordered, so fast, that some of the
food was cold (not sure if they make it in advance and just heat up aft
erwards). The desserts took forever to come.\n\nAt the end of the night
when the bill came, there was a water service charge ($12.50) for 9 peo
ple for just getting tap water. And 18% gratuity after taxes.\n\nAtmosp
here was good, but other than that I do not recommend this place at al
l.",
 "Horrible customer service!! Warning do not purchase anything from her
e, don't waste your time!!! Purchased $2,500 worth of furniture told se
lls person numerous times to deliver furniture in original boxes I did
n't want it assembled. Well received my delivery and everything was as
sembled!!!!! $700 table was damaged!!!! Living room furniture was stuff
ed in delivery truck with people's nasty old mattress....completely unp
rofessional and not worth it!!! Sent my whole order back for refund!! C
alled customer service talked with a guy named Chris, the absolute wors
t customer service agent I've ever spoken too.....Was missing the love
seat, well guess what it's completely sold out. Please spare your time
and money! Don't waste it here, furniture might be good price but not w
orth the experience. I must save delivery guys were very nice.",
 'This whole place basically sucks and it\'s even worse now. I have had
so many bad experiences at this place, I just need to give up on it. \n
\nA few years ago, Sheppard Centre was an alright place. When I lived a
t Yonge and Sheppard, there was a grocery store across the street and a
ll the basics you would need st Sheppard Centre and it was good. \n\nSh
eppard Centre is at a major subway stop, in a hugely busy area, but som

```
ehow stores keep failing here. \n\nThat being the case, there are a few
stores that are nice and handy. I would consider this mall for errands,
now that I live away...except that the major reason for coming is gone
now.\n\nSheppard Centre used to have 1 hour of free parking, which is a
huge draw. You could go and park, run in and pick up your mail, or some
thing from Shoppers Drug Mart and be on your way. They got rid of that.
Not only that, but it\'s $3.50 for half an hour or under!\n\nI think th
is is pretty pathetic for a crappy mall. \n\n"OH, but they have to pay
for their excellent staff!" you say?\n\nYeah, let me tell you about the
staff that work there. First of all, you should also know that the tick
et machine to pay for your parking tickets likes to break down ALL THE
TIME, so you have to go to the parking garage and drive up to the ticke
t booth, park in the non-parking spots, block traffic and go in and pay
the guys sitting around in the office...an office with TV screens of th
e parking garage, where our car has been broken into before...but it\'s
not their fault. There are SIGNS that say they are not liable. Soooo...
what are the TV screens for? To WATCH peoples\' cars get broken into an
d laugh?\n\nRight, so back to the story! \n\nBefore I knew what to do w
hen the machines broke down, I went to the in-mall security\\/info boot
h to ask. The guy was behind the booth, on the phone, having a personal
conversation. He saw me, but turned away and continued for ten minutes.
Finally he got off the phone and made like he was coming over...only to
turn away at the last second and start going through some papers. Final
ly I called out and said, "Excuse me?" He told me he was busy and made
a point of moving towards me as slowly as possible.\n\nThis isn\'t the
only time a staff member has been rude to me before. Dogs are allowed i
n the subway, so I was walking through once, to the subway entrance, HO
LDING my dog, in my arms and a security guard chased me down to tell me
dogs are not allowed. I told him I was just heading to the subway, so h
e followed me there to make sure. Yeah, because I really wanted to take
my dog to the dollarstore and was clearly lying. GO STOP PEOPLE FROM BR
EAKING INTO CARS! \n\nBayview Village, down the street, or one subway s
top over, has a Shoppers, a grocery store, a post office, a Blacks phot
ography and clothing stores...like Sheppard Centre. It also has a Chapt
ers with a Starbucks, an LCBO, GOOD restaurants and the people there ar
e polite, the bathrooms nicer than mine at home AND parking is free.\n
\nSo basically Sheppard Centre can bite me. I\'m done with them.',
 'Pandora charms are great, but not at this location. \n\nI purchased a
charm from Pandora and it was put directly onto the bracelet. Overnigh
```

```
t, I reconsidered one of the charms and wanted to exchange it. I brough
t it back, with the receipt, but was informed that I couldn\'t exchange
it, because it was on the bracelet. \nI explained that it had been put
there BY an employee. The manager was brought out and she was immediate
ly snarky and rude towards me. She said she wouldn\'t accept a charm th
at was "all worn and beat up". I encouraged her to look at the charm. S
he could clearly see it was in pristine condition. She glanced at my br
acelet as a whole, but did not look at the charm, instead insisting tha
t she could tell.\n\nSeriously lady, you don\'t stand by your product a
t all? Your jewellery is so crappy it just gets beaten up and looks lik
e crap after being worn for probably an hour total?!? This wasn\'t even
my fault!!! Plus, she could clearly see from my bracelet that I had spe
nt a LOT of money on my bracelet and am a loyal customer. \n\nI know th
at if I had gotten a Pandora box and put the charm in it and brought it
back, they would have exchanged it without a problem. I can\'t now, bec
ause I\'m sure I would be recognized. Instead I\'ll go to another locat
ion...where perhaps I\'ll be treated like a valued customer and not a l
eper. \n\nI\'ve had bad experiences at this location before. The employ
ees here seem to think customers are beneath them. Each location is ind
ependently owned, so needless to say I won\'t purchase from this locati
on again. Anyone who wants to purchase from Pandora I encourage you to
check out the Scarborough or Fairview Mall locations. Always had good e
xperiences at those locations. \n\nInstead I have sent a complaining em
ail to Pandora head office. We\'ll see what they respond with.']
```

Reading each of the reviews, it is clear that good is mentioned in context like "not as good" or "sound good". This indicates that in the world of text we cannot go by single words (also called **1-grams**) alone. The context of the sentence or the surrounding words at least are very much necessary to understand the sentiment of a sentence.

n-grams

Since 1-grams are insufficient to understand the significance of certain words in our text, it is natural to consider blocks of words, or **n-grams**.

The simplest version of the n-gram model, for $n > 1$, is the **bigram** model, which looks at pairs of consecutive words. For example, the sentence "The quick brown fox jumps over the lazy dog" would have tokens "the quick", "quick brown",..., "lazy dog". The following image (courtesy: google) explains this concept:

This has obvious advantages and disadvantages over looking at words individually:

1. This retains the structure of the overall document, and
2. It paves the way for analyzing words in contex; however,
3. The dimension is vastly larger

In practice, this last challenge can be truly daunting. As an example, *War and Peace* has 3 million characters, which translates to several hundred thousand 1-grams (words). If you consider that the set of all possible bigrams can be as large as the square of the number of 1-grams, this gets us to a hundred billion possible bigrams! If classical ML techniques are not suitable for training on 3 million characters, how can they possibly deal with a hundred billion dimensions?

For this reason, it is often prudent to start by extracting as much value out of 1-grams as possible, before working our way up to more complex structures.

In this section we also start to look again at our main application: calculating some "interesting" features of our corpus of reviews.

When thinking about word analysis, the main topic of interest is finding an *efficient* and *low-dimensional* representation in order to facilitate document visualization and larger-scale analyses. We discuss two broad categories of word representations:

1. `Count-based representations`: word-word and word-document matrices.
2. `Word embeddings`: spectral embedding, UMAP, word2vec, GloVe, and many many more.

These are often used to assist with downstream tasks such as clustering, ranking and labeling, which will be briefly discussed in a different class.

Count-based representations

n-grams fall under a broader category of techniques otherwise known as **count-based representations**. These are techniques to analyze documents by indicating how frequently certain types of structures occur throughout.

Let's start with 1-grams (words). The simplest type of information would be whether a particular word occurs in particular documents. This leads to **word-document co-occurrence matrices**, where the (W, X) entry of the word-document matrix is set to 1 if word W occurs in document X, and 0 otherwise.

There are many variants of this. In lieu of the fact that we are looking for count-based representations of our documents, one natural variable is the following: the (W, X) entry of the word-document matrix equals the number of times that word W occurs in document X, rather than merely being a binary variable.

Let's create a word-document co-occurrence matrix for our set of reviews:

In [19]:
```
# The following code creates a word-document matrix.
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer()
X = vec.fit_transform(AllReviews)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
df.head()
```

Out[19]:

	00	000	0014	00429	00a	00am	00p	00pm	01	0146	...	可能說中文啊	在拉斯维加斯买了新房子	布艺隔热性不强	所以用了实木的百叶窗	推荐卓越窗帘给大家	本来想安装布艺窗帘	知道拉斯维加斯特别热	而且价格实惠
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

5 rows × 31803 columns

Exercise 5:

Find all the high-frequency (top 1%) and low-frequency (bottom 1%) words in the reviews overall. (Hint: import the `Counter()` function from the `collections` class.)

Answer. One possible solution is shown below:

In [20]:
```
from collections import Counter

all_reviews_text = ' '.join(AllReviews)
tokenized_words = nltk.word_tokenize(all_reviews_text)
word_freq = Counter(tokenized_words)
```

```
len(word_freq)
```

Out[20]: 45155

In [21]:
```
## Therefore, top 1% is ~463 words
word_freq.most_common(463)
```

Out[21]:
```
[('.', 78511),
 ('the', 53690),
 (',', 49969),
 ('and', 43873),
 ('I', 39938),
 ('a', 35396),
 ('to', 30472),
 ('was', 23572),
 ('of', 20248),
 ('it', 16975),
 ('is', 16269),
 ('!', 15181),
 ('for', 15021),
 ('in', 14282),
 ('The', 11908),
 ('that', 11672),
 ('with', 11652),
 ('you', 10436),
 ('my', 9553),
 ('but', 9425),
 ('on', 9007),
 ("n't", 9006),
 ("'s", 8741),
 ('have', 8327),
 ('this', 7690),
 ('they', 7533),
 ('not', 7109),
 ('had', 7071),
 ('were', 6743),
 ('are', 6581),
 (')', 6361),
```

```
('we', 6182),
('at', 6008),
('so', 5935),
('(', 5646),
('place', 5548),
('be', 5493),
('good', 5436),
('as', 5408),
('me', 5100),
('food', 5029),
('It', 4757),
('here', 4464),
('like', 4422),
('out', 4330),
('there', 4202),
('just', 4064),
('do', 3939),
('all', 3891),
('very', 3812),
('or', 3724),
('great', 3688),
('get', 3672),
('time', 3656),
('one', 3648),
('We', 3631),
('from', 3564),
('would', 3530),
('up', 3527),
('did', 3425),
('their', 3311),
('which', 3294),
('if', 3291),
('...', 3221),
('an', 3112),
('They', 3103),
('our', 3081),
('back', 3079),
('really', 3060),
('about', 2971),
```

```
('some', 2949),
('service', 2871),
('when', 2870),
('can', 2847),
('$', 2836),
('your', 2811),
('go', 2776),
('-', 2705),
('been', 2602),
('will', 2599),
('more', 2595),
('?', 2589),
("'ve", 2527),
('what', 2510),
('also', 2488),
('This', 2417),
("''", 2365),
('``', 2298),
(':', 2290),
("'m", 2283),
('because', 2269),
('by', 2251),
('only', 2229),
('them', 2224),
('nice', 2217),
('has', 2110),
('too', 2105),
('us', 2096),
('other', 2091),
('little', 2054),
('no', 2024),
('well', 1965),
('could', 1931),
('than', 1919),
('ordered', 1919),
('My', 1912),
('got', 1894),
('even', 1825),
('try', 1719),
```

```
('she', 1710),
('menu', 1697),
('order', 1678),
('he', 1675),
('always', 1672),
('came', 1657),
('restaurant', 1618),
('people', 1607),
('much', 1594),
('come', 1573),
('love', 1534),
('pretty', 1529),
('friendly', 1478),
('staff', 1478),
("'re", 1474),
('chicken', 1469),
('best', 1433),
('off', 1432),
('over', 1422),
('know', 1414),
('after', 1411),
('first', 1399),
('make', 1386),
('made', 1381),
('never', 1369),
('delicious', 1368),
('sauce', 1341),
('two', 1338),
('how', 1316),
('her', 1309),
('definitely', 1292),
('again', 1289),
('think', 1287),
('went', 1286),
('If', 1265),
('right', 1263),
('bar', 1256),
('better', 1247),
('experience', 1236),
```

```
('around', 1233),
('sure', 1225),
('any', 1222),
('few', 1220),
('want', 1218),
('way', 1214),
('who', 1213),
('There', 1211),
('wait', 1204),
('going', 1191),
('before', 1183),
('area', 1178),
('day', 1157),
("'ll", 1149),
('then', 1149),
('bit', 1141),
('see', 1137),
('small', 1134),
('though', 1131),
('still', 1119),
('night', 1113),
('down', 1112),
('said', 1107),
('So', 1101),
('And', 1101),
('take', 1098),
('say', 1088),
('am', 1088),
('lunch', 1081),
('&', 1060),
('2', 1057),
('You', 1048),
('fresh', 1046),
('new', 1044),
('..', 1038),
('something', 1027),
('find', 1020),
('table', 1011),
('cheese', 1010),
```

```
('But', 1001),
('since', 1000),
('side', 996),
('salad', 996),
('into', 994),
('minutes', 994),
('times', 992),
('being', 991),
('location', 986),
('most', 976),
('eat', 975),
('lot', 974),
('while', 972),
('give', 965),
('many', 946),
('de', 943),
('amazing', 940),
('recommend', 932),
('feel', 917),
('now', 915),
("'d", 902),
('thing', 894),
('another', 890),
('long', 884),
('next', 881),
('took', 880),
('does', 879),
('*', 876),
('where', 868),
('ever', 862),
('work', 854),
('meal', 851),
('looking', 849),
('tried', 844),
('bad', 843),
('room', 843),
('asked', 837),
('ca', 833),
('3', 827),
```

```
('A', 825),
('drinks', 823),
('happy', 819),
('everything', 818),
('told', 814),
('enough', 813),
('pizza', 808),
('store', 807),
('5', 803),
('price', 802),
('dinner', 798),
('sweet', 794),
('hot', 792),
('his', 783),
('need', 781),
('She', 781),
('last', 776),
('When', 771),
('different', 769),
('big', 765),
('every', 763),
('home', 758),
('super', 753),
('drink', 751),
('quite', 742),
('friend', 736),
('large', 736),
('He', 733),
('Not', 731),
('wanted', 729),
('things', 724),
('taste', 723),
('both', 718),
('dish', 717),
(';', 714),
('should', 713),
('flavor', 704),
('sandwich', 703),
('Vegas', 702),
```

```
('worth', 698),
('Great', 697),
('away', 695),
('meat', 689),
('spot', 688),
('hour', 686),
('clean', 686),
('same', 685),
('server', 679),
('inside', 677),
('those', 675),
('stars', 666),
('For', 664),
('fries', 663),
('places', 663),
('found', 661),
('et', 659),
('rice', 658),
('prices', 646),
('top', 645),
('4', 644),
('favorite', 644),
('quality', 638),
('through', 637),
('Our', 636),
('special', 634),
('car', 630),
('awesome', 629),
('its', 629),
('friends', 628),
("'", 627),
('thought', 626),
('coffee', 625),
('bread', 625),
('items', 615),
('visit', 615),
('actually', 614),
('check', 614),
('review', 614),
```

```
('nothing', 613),
('perfect', 609),
('That', 607),
('served', 605),
('full', 595),
('years', 594),
('burger', 592),
('selection', 591),
('left', 590),
('anything', 589),
('customer', 586),
('decided', 584),
('look', 577),
('10', 575),
('coming', 571),
('sushi', 569),
('dishes', 567),
('probably', 566),
('free', 564),
('couple', 563),
('fun', 561),
('getting', 561),
('kind', 559),
('beer', 557),
('him', 554),
('ask', 554),
('tasty', 553),
('without', 553),
('options', 552),
('done', 551),
('excellent', 548),
('fried', 547),
('husband', 546),
('these', 545),
('i', 540),
('far', 539),
('busy', 539),
('once', 536),
('half', 536),
```

```
('else', 535),
('enjoy', 534),
('cream', 529),
('usually', 523),
('having', 521),
('enjoyed', 519),
('As', 519),
('All', 519),
('water', 518),
('use', 516),
('soup', 516),
('breakfast', 515),
('open', 515),
('why', 514),
('chocolate', 513),
('end', 513),
('used', 512),
('outside', 511),
('quick', 510),
('hard', 507),
('each', 504),
('old', 503),
('line', 502),
('looked', 502),
('let', 500),
('....', 499),
('huge', 499),
('atmosphere', 495),
('put', 494),
('house', 494),
('fish', 490),
('front', 489),
('shop', 486),
('part', 485),
('decent', 484),
('able', 483),
('called', 483),
('cool', 481),
('la', 480),
```

```
('care', 479),
('No', 478),
('gave', 477),
('business', 477),
('tables', 476),
('pork', 474),
('several', 473),
('spicy', 470),
('high', 467),
('course', 466),
('beef', 464),
('least', 464),
('parking', 463),
('After', 460),
('wine', 460),
('What', 459),
('le', 459),
('Service', 458),
('cooked', 456),
('walk', 455),
('close', 455),
('--', 454),
('loved', 452),
('name', 450),
('everyone', 449),
('1', 448),
('almost', 447),
('tea', 447),
('maybe', 446),
('disappointed', 445),
('pay', 444),
('return', 438),
('must', 437),
('dining', 437),
('less', 436),
('ice', 436),
('during', 436),
('family', 435),
('waitress', 433),
```

```
('call', 431),
('three', 431),
('dessert', 430),
('whole', 430),
('fast', 429),
('helpful', 428),
('especially', 427),
('Their', 426),
('tasted', 426),
('such', 425),
('fantastic', 424),
('seemed', 424),
('Also', 423),
('someone', 423),
('trying', 421),
('own', 419),
('reviews', 418),
('job', 417),
('town', 417),
('sit', 417),
('tell', 417),
('size', 416),
('offer', 415),
('group', 414),
('felt', 414),
('star', 414),
('One', 413),
('restaurants', 411),
('hours', 410),
('makes', 409),
('started', 406),
('In', 406),
('un', 406),
('might', 405),
('until', 403),
('point', 403),
('steak', 403),
('waiting', 402),
('liked', 395),
```

```
 ('Overall', 392),
 ('needed', 392),
 ('chips', 391)]
```

In [22]:
```
## Similarly, bottom 1% is ~463 words
word_freq.most_common()[-463:-1]
```

Out[22]:
```
[('2xs', 1),
 ('VIbe', 1),
 ('carve', 1),
 ('profiles', 1),
 ('Ishi-yaki', 1),
 ('Sockeye', 1),
 ('flasks', 1),
 ('chillax-with-friends', 1),
 ('Fino', 1),
 ('Vaz', 1),
 ('Knowledge', 1),
 ('Conversational', 1),
 ('scripted', 1),
 ('Charla', 1),
 ('myselfShe', 1),
 ('aurora', 1),
 ('back-in-the-day', 1),
 ('Coras', 1),
 ('interrupting', 1),
 ('Mojitos', 1),
 ('Elote', 1),
 ('locos', 1),
 ('SPEED', 1),
 ('MCDONALD', 1),
 ('truest', 1),
 ('tagged', 1),
 ('Cruz', 1),
 ('Jetson', 1),
 ('Schrager', 1),
 ('Vics', 1),
 ('FWIW', 1),
 ('Kimpton', 1),
```

```
('Hogs', 1),
('communiting', 1),
('Suggested', 1),
('277', 1),
('275-near', 1),
('365-far', 1),
('201-central', 1),
('unmarked', 1),
('corridors', 1),
('RP', 1),
('Gianfranco', 1),
('Adriatic', 1),
('Montepulciano', 1),
('unloaded', 1),
('bellmen', 1),
('unusable', 1),
('bad.', 1),
('subjective', 1),
('Loyd', 1),
('Roxy', 1),
('Burlesque', 1),
('Cleve', 1),
('attracks', 1),
('Fireplace', 1),
('undoubtably', 1),
('Phoenix\\/Scottsdale', 1),
('Would\\/will', 1),
('cowboys', 1),
('Numerous', 1),
('Positive', 1),
('contend', 1),
('FS', 1),
('Tarbell', 1),
('extensively', 1),
('Tallavera', 1),
('Acacia', 1),
('Boulders', 1),
('CONS', 1),
('TT', 1),
```

```
('bordeaux', 1),
('Sancerre', 1),
('Manthers', 1),
('Cleveberg', 1),
('Puts', 1),
('infinoty', 1),
('setvice', 1),
('s7', 1),
('grubbin', 1),
('GRUBBING', 1),
('thatnk', 1),
('truss', 1),
('joann', 1),
('smidge', 1),
('disgusts', 1),
('serval', 1),
('omurice', 1),
('rice\\/pasta', 1),
('Hub', 1),
('ackee', 1),
('congees', 1),
('Gonoe', 1),
('japchae', 1),
('handroll', 1),
('RMT', 1),
('bbt', 1),
('Kum', 1),
('Skyview', 1),
('Myungdong', 1),
('Kalkuksu', 1),
('407', 1),
('bossam', 1),
('Myundong', 1),
('44.95', 1),
('36.99', 1),
('handrolls', 1),
('Gyukai', 1),
('Tsukemen', 1),
('woodoven', 1),
```

```
('29.99', 1),
('re-discovered', 1),
('Agreed', 1),
('cheesebake', 1),
('6.85', 1),
('ika', 1),
('fishball', 1),
('Torched', 1),
('yam', 1),
('905', 1),
('Kaji.I', 1),
('longan', 1),
('A___erto', 1),
('Barilla', 1),
('saaauwces', 1),
('saauwce', 1),
('Whatchyou', 1),
('Logan', 1),
('woth', 1),
('Wing', 1),
('0.35', 1),
("O'Connell", 1),
("'gwei", 1),
('Monkland', 1),
('Ninth', 1),
('Parallel', 1),
('Benu', 1),
('Quince', 1),
('Saison', 1),
('revered', 1),
('Versailles', 1),
('Bold', 1),
('ivy', 1),
('broadcasts', 1),
('best-in-class', 1),
('degustation', 1),
('Langoustine', 1),
('preperation', 1),
('Noix', 1),
```

```
('Saint-Jacques', 1),
('relates', 1),
('weeks\\/months', 1),
('Wonka-esque', 1),
('Admit', 1),
('irreverent', 1),
('stuffier', 1),
('melty-ess', 1),
('savoriest', 1),
('brine', 1),
('malai', 1),
('bought\\/transferred', 1),
('leadership', 1),
('supervisors', 1),
('Cunningham', 1),
('mid-size', 1),
('young-uns', 1),
('Steeler', 1),
('Pluses', 1),
('dismissive', 1),
('Pantages', 1),
('númerous', 1),
('Reviewed', 1),
('refunding', 1),
('imperfections', 1),
('C-', 1),
('AGH', 1),
('Beets', 1),
('aps', 1),
('220', 1),
('260.00', 1),
('720.00', 1),
('720', 1),
('BUFFALO', 1),
('DIP', 1),
('worms', 1),
('dunkaroo', 1),
('Chaison', 1),
('Nivatvongs-Tobin', 1),
```

```
('amazeballz', 1),
('atetoomuchoffhehappyhourmenu', 1),
('Alot', 1),
('Kyklos', 1),
('Flaming', 1),
('Empanadas', 1),
('Rodriguez', 1),
('wouldve', 1),
('mundo', 1),
('tequila.they', 1),
('shocked\\/put', 1),
('1:28', 1),
('Liz.She', 1),
('fabulous.She', 1),
('restaurant.I', 1),
('Colman', 1),
('unamused', 1),
('Kimmy.She', 1),
('unpersonable', 1),
('table.the', 1),
('Pearle', 1),
('Diavola', 1),
('thumble', 1),
('Weekly', 1),
('Kiki', 1),
('BUN', 1),
('Dalton', 1),
('Bryce', 1),
('restaurant.Waittttt', 1),
('Xoxo', 1),
('geoduck', 1),
('Layne', 1),
('amazeballs', 1),
('beauties', 1),
('4pm-6:00pm', 1),
('Handlers', 1),
('Archi', 1),
('strayed', 1),
('inducted', 1),
```

```
('ladie', 1),
('gem.But', 1),
('BM', 1),
('sunday.you', 1),
('strings.I', 1),
('homeade', 1),
('Vincanto', 1),
('Leche', 1),
('Lindo', 1),
('Chinita', 1),
('ramenDX', 1),
('Five50', 1),
('Degrees', 1),
('cavi-lipo', 1),
('Burquest', 1),
('Melinda', 1),
('tues', 1),
('too.a', 1),
('Humbolt', 1),
('Jamón', 1),
('restaurant.We', 1),
('Zmon', 1),
('Cegas', 1),
('nigiri.Get', 1),
('XXX2', 1),
('XXX3', 1),
('in.stage', 1),
('week.get', 1),
('stoked', 1),
('Stefanie', 1),
('desk.Thank', 1),
('Letty', 1),
('Sergio', 1),
('Shroom', 1),
('Akexandra', 1),
('burger.The', 1),
('gaucho', 1),
('wings.unfreakin', 1),
('believable', 1),
```

```
('menu.I', 1),
('monday', 1),
('Randomness', 1),
('vocals.I', 1),
('wed', 1),
('cevape', 1),
('humongo', 1),
('3:30-6:30', 1),
('mon-Friday', 1),
('gardener', 1),
('G35', 1),
('Coupe', 1),
('thermostat', 1),
('lf', 1),
('pulley', 1),
('Weins', 1),
('Fl', 1),
('needs.NEVER', 1),
('civics', 1),
('builds', 1),
('Drag', 1),
('Cars', 1),
('building\\/racing', 1),
('memorial', 1),
('downpart', 1),
('taphouse', 1),
('Resnable', 1),
('Consultation', 1),
('unrushed', 1),
('bucatini', 1),
('NINA', 1),
('pinwheels', 1),
('1LB', 1),
('Naga', 1),
('Qdobas', 1),
('madly', 1),
('F.Y', 1),
('gingivectomy', 1),
('anesthesia', 1),
```

```
('1800\\/implant', 1),
('re-experience', 1),
('goodwich', 1),
('1:20am', 1),
('slang', 1),
('Oden', 1),
('Raku-Tofu', 1),
('Agedashi-Tofu', 1),
('suzuya', 1),
('cakes\\/crepes', 1),
('Elk', 1),
('injector', 1),
('needles', 1),
('symmetrical', 1),
('bachi', 1),
('CFC', 1),
('Mentaiko-Don', 1),
('Hiyashi', 1),
('Chuka', 1),
('Managers', 1),
('annin-dofu', 1),
('hostess\\/servers', 1),
('monta\\/spring', 1),
('66', 1),
('card\\/cash', 1),
('Nakamuraya', 1),
('hiv', 1),
('std', 1),
('theyve', 1),
('nagger', 1),
('founder', 1),
('caregiver', 1),
('5-30', 1),
('CNAs', 1),
('Shrimps', 1),
('seamed', 1),
('meat\\/falafel', 1),
('alcohols', 1),
('cleanness', 1),
```

```
('consequences', 1),
('stylists\\/nail', 1),
('ichiza', 1),
('know\\/care', 1),
('up\\/cleaning', 1),
('Tori', 1),
('Tiana', 1),
('hawaii', 1),
('kiichi', 1),
('enjoed', 1),
('Firestone', 1),
('630', 1),
('not-so-savory', 1),
('stratosphere', 1),
('Fund', 1),
('Luv-It', 1),
('Prescribed', 1),
('acne-free', 1),
('persuasive', 1),
('whew', 1),
('opi', 1),
('Peso', 1),
('Fixed', 1),
('caucasian', 1),
('sahara', 1),
('Andreas', 1),
('90percent', 1),
('Tahini', 1),
('katsudon', 1),
('poke-bowl', 1),
('sushi\\/food', 1),
('puked', 1),
('Davio', 1),
('rewrite', 1),
('tea\\/milk', 1),
('Clinicians', 1),
('jargons', 1),
('Chinese\\/authentic', 1),
('panda-express', 1),
```

```
('Gow', 1),
('Food\\/beverage', 1),
('dippin', 1),
('aquafina', 1),
('SAMs', 1),
('Refer', 1),
('splitted', 1),
('spago', 1),
('Cioppino', 1),
('UNIQUE', 1),
('fragile', 1),
('flavor\\/texture', 1),
('bucks\\/a', 1),
('Moriawase', 1),
('30+minutes', 1),
('6:05pm', 1),
('TiNA', 1),
('Supersmile', 1),
('Communication', 1),
('examinations', 1),
('technique\\/machines', 1),
('grape\\/bubble', 1),
('go-pro', 1),
('X-rated', 1),
('Tsutomu-San', 1),
('locomoco', 1),
('menu.BTW', 1),
('Russel', 1),
('henderson', 1),
('Mariott', 1),
('Consiglio', 1),
('mother-in-law', 1),
('matboard', 1),
('smugly', 1),
('radiologic', 1),
('Grimy', 1),
('penicillin', 1),
('However-', 1),
('customers-', 1),
```

```
('RN', 1),
('unprofessionalism', 1),
('Selma', 1),
('=0', 1),
('volt', 1),
('subletting', 1),
('terminate', 1),
('hundred\\/month', 1),
('leg\\/foot', 1),
('pedi\\/mani-', 1),
('Ultas', 1),
('bourguignon', 1),
('real-time', 1),
('Rectory', 1),
('tenth', 1),
('ferry', 1),
('euthanize', 1),
('amend', 1),
('hypodermic', 1),
('five-minute', 1),
('9600', 1),
('catheter', 1),
('bodywork', 1),
('Xterra', 1),
('decal', 1),
('specialists', 1),
('Veterinarian', 1),
('Chicha', 1),
('papa', 1),
('saltado', 1),
('comming', 1),
('adventuredome', 1),
('delievery', 1),
('Mics', 1),
('cabling', 1),
('Ls', 1),
('Song', 1),
('geezer', 1),
```

```
 ('brained', 1),
 ('Hooples', 1)]
```

Let's do the same for bigrams. Here is the code to get the set of bigrams for the first 5 reviews:

In [23]:
```
from nltk.util import ngrams

first_5_revs = data.text[0:5]
word_tokens = nltk.word_tokenize(''.join(first_5_revs))
list(ngrams(word_tokens, 2)) #ngrams(word_tokens,n) gives the n-grams.
```

Out[23]:
```
[('Super', 'simple'),
 ('simple', 'place'),
 ('place', 'but'),
 ('but', 'amazing'),
 ('amazing', 'nonetheless'),
 ('nonetheless', '.'),
 ('.', 'It'),
 ('It', "'s"),
 ("'s", 'been'),
 ('been', 'around'),
 ('around', 'since'),
 ('since', 'the'),
 ('the', '30'),
 ('30', "'s"),
 ("'s", 'and'),
 ('and', 'they'),
 ('they', 'still'),
 ('still', 'serve'),
 ('serve', 'the'),
 ('the', 'same'),
 ('same', 'thing'),
 ('thing', 'they'),
 ('they', 'started'),
 ('started', 'with'),
 ('with', ':'),
 (':', 'a'),
 ('a', 'bologna'),
 ('bologna', 'and'),
```

```
('and', 'salami'),
('salami', 'sandwich'),
('sandwich', 'with'),
('with', 'mustard'),
('mustard', '.'),
('.', 'Staff'),
('Staff', 'was'),
('was', 'very'),
('very', 'helpful'),
('helpful', 'and'),
('and', 'friendly.Small'),
('friendly.Small', 'unassuming'),
('unassuming', 'place'),
('place', 'that'),
('that', 'changes'),
('changes', 'their'),
('their', 'menu'),
('menu', 'every'),
('every', 'so'),
('so', 'often'),
('often', '.'),
('.', 'Cool'),
('Cool', 'decor'),
('decor', 'and'),
('and', 'vibe'),
('vibe', 'inside'),
('inside', 'their'),
('their', '30'),
('30', 'seat'),
('seat', 'restaurant'),
('restaurant', '.'),
('.', 'Call'),
('Call', 'for'),
('for', 'a'),
('a', 'reservation'),
('reservation', '.'),
('.', 'We'),
('We', 'had'),

('had', 'their'),
```

```
('their', 'beef'),
('beef', 'tartar'),
('tartar', 'and'),
('and', 'pork'),
('pork', 'belly'),
('belly', 'to'),
('to', 'start'),
('start', 'and'),
('and', 'a'),
('a', 'salmon'),
('salmon', 'dish'),
('dish', 'and'),
('and', 'lamb'),
('lamb', 'meal'),
('meal', 'for'),
('for', 'mains'),
('mains', '.'),
('.', 'Everything'),
('Everything', 'was'),
('was', 'incredible'),
('incredible', '!'),
('!', 'I'),
('I', 'could'),
('could', 'go'),
('go', 'on'),
('on', 'at'),
('at', 'length'),
('length', 'about'),
('about', 'how'),
('how', 'all'),
('all', 'the'),
('the', 'listed'),
('listed', 'ingredients'),
('ingredients', 'really'),
('really', 'make'),
('make', 'their'),
('their', 'dishes'),
('dishes', 'amazing'),

('amazing', 'but'),
```

```
('but', 'honestly'),
('honestly', 'you'),
('you', 'just'),
('just', 'need'),
('need', 'to'),
('to', 'go'),
('go', '.'),
('.', 'A'),
('A', 'bit'),
('bit', 'outside'),
('outside', 'of'),
('of', 'downtown'),
('downtown', 'montreal'),
('montreal', 'but'),
('but', 'take'),
('take', 'the'),
('the', 'metro'),
('metro', 'out'),
('out', 'and'),
('and', 'it'),
('it', "'s"),
("'s", 'less'),
('less', 'than'),
('than', 'a'),
('a', '10'),
('10', 'minute'),
('minute', 'walk'),
('walk', 'from'),
('from', 'the'),
('the', 'station.Lester'),
('station.Lester', "'s"),
("'s", 'is'),
('is', 'located'),
('located', 'in'),
('in', 'a'),
('a', 'beautiful'),
('beautiful', 'neighborhood'),
('neighborhood', 'and'),

('and', 'has'),
```

```
('has', 'been'),
('been', 'there'),
('there', 'since'),
('since', '1951'),
('1951', '.'),
('.', 'They'),
('They', 'are'),
('are', 'known'),
('known', 'for'),
('for', 'smoked'),
('smoked', 'meat'),
('meat', 'which'),
('which', 'most'),
('most', 'deli'),
('deli', "'s"),
("'s", 'have'),
('have', 'but'),
('but', 'their'),
('their', 'brisket'),
('brisket', 'sandwich'),
('sandwich', 'is'),
('is', 'what'),
('what', 'I'),
('I', 'come'),
('come', 'to'),
('to', 'montreal'),
('montreal', 'for'),
('for', '.'),
('.', 'They'),
('They', "'ve"),
("'ve", 'got'),
('got', 'about'),
('about', '12'),
('12', 'seats'),
('seats', 'outside'),
('outside', 'to'),
('to', 'go'),
('go', 'along'),

('along', 'with'),
```

```
('with', 'the'),
('the', 'inside'),
('inside', '.'),
('.', 'The'),
('The', 'smoked'),
('smoked', 'meat'),
('meat', 'is'),
('is', 'up'),
('up', 'there'),
('there', 'in'),
('in', 'quality'),
('quality', 'and'),
('and', 'taste'),
('taste', 'with'),
('with', 'Schwartz'),
('Schwartz', "'s"),
("'s", 'and'),
('and', 'you'),
('you', "'ll"),
("'ll", 'find'),
('find', 'less'),
('less', 'tourists'),
('tourists', 'at'),
('at', 'Lester'),
('Lester', "'s"),
("'s", 'as'),
('as', 'well.Love'),
('well.Love', 'coming'),
('coming', 'here'),
('here', '.'),
('.', 'Yes'),
('Yes', 'the'),
('the', 'place'),
('place', 'always'),
('always', 'needs'),
('needs', 'the'),
('the', 'floor'),
('floor', 'swept'),

('swept', 'but'),
```

```
('but', 'when'),
('when', 'you'),
('you', 'give'),
('give', 'out'),
('out', 'peanuts'),
('peanuts', 'in'),
('in', 'the'),
('the', 'shell'),
('shell', 'how'),
('how', 'wo'),
('wo', "n't"),
("n't", 'it'),
('it', 'always'),
('always', 'be'),
('be', 'a'),
('a', 'bit'),
('bit', 'dirty'),
('dirty', '.'),
('.', 'The'),
('The', 'food'),
('food', 'speaks'),
('speaks', 'for'),
('for', 'itself'),
('itself', ','),
(',', 'so'),
('so', 'good'),
('good', '.'),
('.', 'Burgers'),
('Burgers', 'are'),
('are', 'made'),
('made', 'to'),
('to', 'order'),
('order', 'and'),
('and', 'the'),
('the', 'meat'),
('meat', 'is'),
('is', 'put'),
('put', 'on'),

('on', 'the'),
```

```
('the', 'grill'),
('grill', 'when'),
('when', 'you'),
('you', 'order'),
('order', 'your'),
('your', 'sandwich'),
('sandwich', '.'),
('.', 'Getting'),
('Getting', 'the'),
('the', 'small'),
('small', 'burger'),
('burger', 'just'),
('just', 'means'),
('means', '1'),
('1', 'patty'),
('patty', ','),
(',', 'the'),
('the', 'regular'),
('regular', 'is'),
('is', 'a'),
('a', '2'),
('2', 'patty'),
('patty', 'burger'),
('burger', 'which'),
('which', 'is'),
('is', 'twice'),
('twice', 'the'),
('the', 'deliciousness'),
('deliciousness', '.'),
('.', 'Getting'),
('Getting', 'the'),
('the', 'Cajun'),
('Cajun', 'fries'),
('fries', 'adds'),
('adds', 'a'),
('a', 'bit'),
('bit', 'of'),
('of', 'spice'),

('spice', 'to'),
```

```
('to', 'them'),
('them', 'and'),
('and', 'whatever'),
('whatever', 'size'),
('size', 'you'),
('you', 'order'),
('order', 'they'),
('they', 'always'),
('always', 'throw'),
('throw', 'more'),
('more', 'fries'),
('fries', '('),
('(', 'a'),
('a', 'lot'),
('lot', 'more'),
('more', 'fries'),
('fries', ')'),
(')', 'into'),
('into', 'the'),
('the', 'bag.Had'),
('bag.Had', 'their'),
('their', 'chocolate'),
('chocolate', 'almond'),
('almond', 'croissant'),
('croissant', 'and'),
('and', 'it'),
('it', 'was'),
('was', 'amazing'),
('amazing', '!'),
('!', 'So'),
('So', 'light'),
('light', 'and'),
('and', 'buttery'),
('buttery', 'and'),
('and', 'oh'),
('oh', 'my'),
('my', 'how'),
('how', 'chocolaty'),

('chocolaty', '.'),
```

```
('.', 'If'),
('If', 'you'),
('you', "'re"),
("'re", 'looking'),
('looking', 'for'),
('for', 'a'),
('a', 'light'),
('light', 'breakfast'),
('breakfast', 'then'),
('then', 'head'),
('head', 'out'),
('out', 'here'),
('here', '.'),
('.', 'Perfect'),
('Perfect', 'spot'),
('spot', 'for'),
('for', 'a'),
('a', 'coffee\\/latté'),
('coffee\\/latté', 'before'),
('before', 'heading'),
('heading', 'out'),
('out', 'to'),
('to', 'the'),
('the', 'old'),
('old', 'port')]
```

Story time: Prison Transcript

Exercise 6:

Write a function called `top_k_ngrams(word_tokens, n, k)` for printing out the top k n-grams. Use this function to get the top 10 1-grams, 2-grams, and 3-grams from the first 1000 reviews in our dataset.

Answer. One possible solution is shown below:

In [24]:
```
from nltk.util import ngrams

def top_k_ngrams(word_tokens,n,k):

    ## Getting them as n-grams
    n_gram_list = list(ngrams(word_tokens, n))

    ### Getting each n-gram as a separate string
    n_gram_strings = [' '.join(each) for each in n_gram_list]

    n_gram_counter = Counter(n_gram_strings)
    most_common_k = n_gram_counter.most_common(k)
    print(most_common_k)

   # x_pos = [k for k,v in most_common_k]
   # y_pos = [v for k,v in most_common_k]

   # plt.bar(x_pos, y_pos,align='center')
```

In [25]:
```
### Getting a single string
top_1000_reviews = data.text[0:1000]
all_reviews_text = ' '.join(top_1000_reviews)

## Splitting them into tokens
word_tokens = nltk.word_tokenize(all_reviews_text)

## Calling the function for top k
top_k_ngrams(word_tokens, 1, 10)
```

```
[('.', 10874), ('the', 6929), (',', 6198), ('I', 5920), ('and', 5325),
('a', 4746), ('to', 4376), ('was', 3058), ('of', 2865), ('it', 2553)]
```

In [26]:
```
top_k_ngrams(word_tokens, 2, 10)
```

```
[('. I', 1956), ('. The', 1512), (', but', 866), ('. It', 726), ('of th
e', 601), (', and', 601), ('I was', 531), ('in the', 518), (', I', 46
```

```
5), ('it was', 439)]
```

In [27]:
```
top_k_ngrams(word_tokens, 3, 10)
```

```
[(". It 's", 241), ('. It was', 193), (', but I', 167), (". I 've", 16
3), ('. I was', 144), ("I do n't", 131), (". I 'm", 121), ('. The foo
d', 106), (', but it', 103), ('. I had', 99)]
```

Stop words

You may have noticed a pattern in the types of words that show up in the top 10 1-grams, 2-grams, and 3-grams. In particular, these are common words that appear in every sentence of the English language: pronouns like "I", prepositions like "but", "of", "and", articles like "the", etc. These very common words are usually uninformative, and their very large occurrence values can distort the results of many NLP algorithms.

For this reason, it is common to pre-process text by removing words that you have a reason to believe are uninformative; these words are called **stop words**. Usually, it suffices to simply treat extremely common words as stop words. However, for specific types of applications it might make sense to use other stop words; e.g. the word "burger" when analyzing reviews of burger chains.

The `nltk` library has a standard list of stopwords, which you can download by writing `nltk.download("stopwords")`. We can then load the stopwords package from the nltk.corpus and use it to load the stop words:

In [28]:
```
nltk.download('stopwords')
from nltk.corpus import stopwords
print(stopwords.words("english"))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'your
selves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers',
'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'the
irs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'bee
```

```
n', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doin
g', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'unti
l', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'betw
een', 'into', 'through', 'during', 'before', 'after', 'above', 'below',
'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where',
'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'oth
er', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't",
'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ai
n', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
"doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'is
n', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn',
"needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't",
'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

You can get a list of all the Spanish stop words as well:

In [29]:
```
print(stopwords.words("spanish"))
```

```
['de', 'la', 'que', 'el', 'en', 'y', 'a', 'los', 'del', 'se', 'las', 'p
or', 'un', 'para', 'con', 'no', 'una', 'su', 'al', 'lo', 'como', 'más',
'pero', 'sus', 'le', 'ya', 'o', 'este', 'sí', 'porque', 'esta', 'entr
e', 'cuando', 'muy', 'sin', 'sobre', 'también', 'me', 'hasta', 'hay',
'donde', 'quien', 'desde', 'todo', 'nos', 'durante', 'todos', 'uno', 'l
es', 'ni', 'contra', 'otros', 'ese', 'eso', 'ante', 'ellos', 'e', 'est
o', 'mí', 'antes', 'algunos', 'qué', 'unos', 'yo', 'otro', 'otras', 'ot
ra', 'él', 'tanto', 'esa', 'estos', 'mucho', 'quienes', 'nada', 'mucho
s', 'cual', 'poco', 'ella', 'estar', 'estas', 'algunas', 'algo', 'nosot
ros', 'mi', 'mis', 'tú', 'te', 'ti', 'tu', 'tus', 'ellas', 'nosotras',
'vosotros', 'vosotras', 'os', 'mío', 'mía', 'míos', 'mías', 'tuyo', 'tu
ya', 'tuyos', 'tuyas', 'suyo', 'suya', 'suyos', 'suyas', 'nuestro', 'nu
estra', 'nuestros', 'nuestras', 'vuestro', 'vuestra', 'vuestros', 'vues
tras', 'esos', 'esas', 'estoy', 'estás', 'está', 'estamos', 'estáis',
'están', 'esté', 'estés', 'estemos', 'estéis', 'estén', 'estaré', 'esta
rás', 'estará', 'estaremos', 'estaréis', 'estarán', 'estaría', 'estaría
```

```
s', 'estaríamos', 'estaríais', 'estarían', 'estaba', 'estabas', 'estábamos', 'estabais', 'estaban', 'estuve', 'estuviste', 'estuvo', 'estuvimos', 'estuvisteis', 'estuvieron', 'estuviera', 'estuvieras', 'estuviéramos', 'estuvierais', 'estuvieran', 'estuviese', 'estuvieses', 'estuviésemos', 'estuvieseis', 'estuviesen', 'estando', 'estado', 'estada', 'estados', 'estadas', 'estad', 'he', 'has', 'ha', 'hemos', 'habéis', 'han', 'haya', 'hayas', 'hayamos', 'hayáis', 'hayan', 'habré', 'habrás', 'habrá', 'habremos', 'habréis', 'habrán', 'habría', 'habrías', 'habríamos', 'habríais', 'habrían', 'había', 'habías', 'habíamos', 'habíais', 'habían', 'hube', 'hubiste', 'hubo', 'hubimos', 'hubisteis', 'hubieron', 'hubiera', 'hubieras', 'hubiéramos', 'hubierais', 'hubieran', 'hubiese', 'hubieses', 'hubiésemos', 'hubieseis', 'hubiesen', 'habiendo', 'habido', 'habida', 'habidos', 'habidas', 'soy', 'eres', 'es', 'somos', 'sois', 'son', 'sea', 'seas', 'seamos', 'seáis', 'sean', 'seré', 'serás', 'será', 'seremos', 'seréis', 'serán', 'sería', 'serías', 'seríamos', 'seríais', 'serían', 'era', 'eras', 'éramos', 'erais', 'eran', 'fui', 'fuiste', 'fue', 'fuimos', 'fuisteis', 'fueron', 'fuera', 'fueras', 'fuéramos', 'fuerais', 'fueran', 'fuese', 'fueses', 'fuésemos', 'fueseis', 'fuesen', 'sintiendo', 'sentido', 'sentida', 'sentidos', 'sentidas', 'siente', 'sentid', 'tengo', 'tienes', 'tiene', 'tenemos', 'tenéis', 'tienen', 'tenga', 'tengas', 'tengamos', 'tengáis', 'tengan', 'tendré', 'tendrás', 'tendrá', 'tendremos', 'tendréis', 'tendrán', 'tendría', 'tendrías', 'tendríamos', 'tendríais', 'tendrían', 'tenía', 'tenías', 'teníamos', 'teníais', 'tenían', 'tuve', 'tuviste', 'tuvo', 'tuvimos', 'tuvisteis', 'tuvieron', 'tuviera', 'tuvieras', 'tuviéramos', 'tuvierais', 'tuvieran', 'tuviese', 'tuvieses', 'tuviésemos', 'tuvieseis', 'tuviesen', 'teniendo', 'tenido', 'tenida', 'tenidos', 'tenidas', 'tened']
```

Exercise 7:

7.1

Filter out all of the stop words in the first review of the Yelp review data and print out your answer. Additionally, print out (separately) the stopwords you found in this review.

Answer. One possible solution is given below:

In [30]:
```
stop_words = set(stopwords.words("english"))
without_stop_words = []
stopword = []
sentence = data.text[0]
words = nltk.word_tokenize(sentence)
for word in words:
    if word in stop_words:
        stopword.append(word)
    else:
        without_stop_words.append(word)

print(stopword)
print()
print(without_stop_words)
```

```
['but', 'been', 'the', 'and', 'they', 'the', 'same', 'they', 'with',
'a', 'and', 'with', 'was', 'very', 'and']

['Super', 'simple', 'place', 'amazing', 'nonetheless', '.', 'It', "'s",
'around', 'since', '30', "'s", 'still', 'serve', 'thing', 'started',
':', 'bologna', 'salami', 'sandwich', 'mustard', '.', 'Staff', 'helpfu
l', 'friendly', '.']
```

7.2

Modify the function `top_k_ngrams(word_tokens, n, k)` to remove stop words before determining the top n-grams.

Answer. Our recommended solution is shown below:

In [31]:
```
# Removing the most basic stop words from the ntlk corpus and including
 only those
# words with character size above 2 so as to remove punctuations
# But, this could be extended to remove further high and low frequency
 stop words

from nltk.corpus import stopwords
```

```
import string
eng_stopwords = stopwords.words('english')

### Getting a single string
all_reviews_text = ' '.join(top_1000_reviews)

## Splitting them into tokens
word_tokens = nltk.word_tokenize(all_reviews_text)

## Removing the stopwords
word_tokens_clean = [each for each in word_tokens if each.lower() not i
n eng_stopwords and len(each.lower()) > 2]

## Calling the function for top k
top_k_ngrams(word_tokens_clean, 3, 10)
```

```
[('wait come back', 11), ("n't wait come", 10), ("n't feel like", 8),
("'ll definitely back", 8), ("'ve never seen", 8), ('give place try',
7), ('staff friendly helpful', 7), ('Vegas nearly years', 7), ('Las Veg
as strip', 7), ('half dozen times', 7)]
```

In some contexts, it is common to remove both very common and very *uncommon* words. The idea is that common words like "a" are almost never informative, while uncommon words like "syzygy" occur so infrequently in a corpus that many algorithms have a hard time processing them in a meaningful way. We will not deal with uncommon words today, but you should be aware that doing so improves the performance of several NLP techniques.

Finding important words

Up to this point, we have focused on techniques for transforming our data. We are now ready to start looking for some answers, so let's take a break from discussing techniques so we can explore our dataset and various ways to summarize it.

We begin by looking at the words and n-grams that are most common in positive and negative reviews. Note that in the following code, we don't reuse many of the pre-processing steps discussed at the start of the tutorial. This is because many of them are included as options in

existing packages. In a serious project one would often customize this pre-processing to some degree, but we skip this in order to get some displayable results.

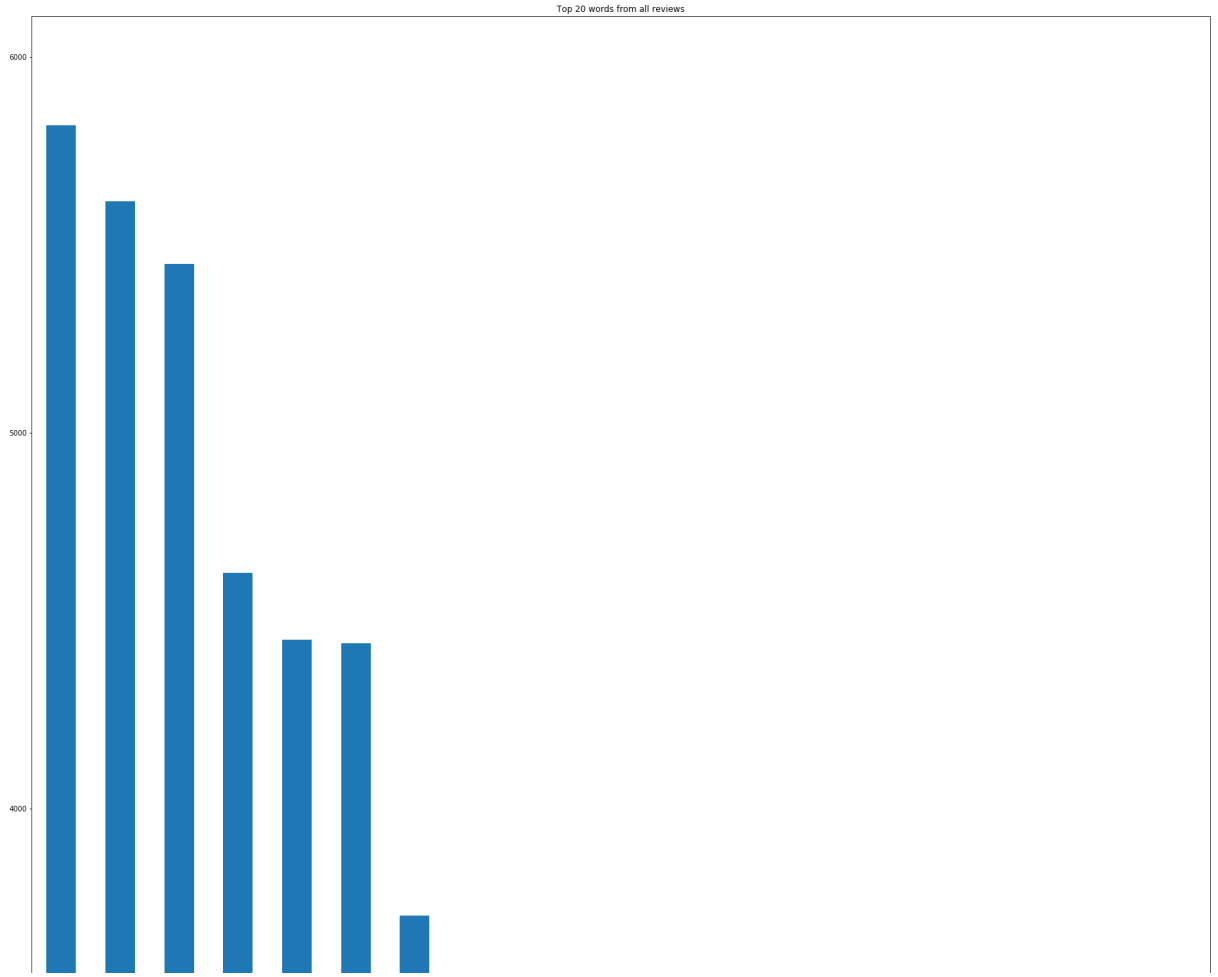
In [32]:
```
# Following code grabbed from:
# https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a
# we will use it in our context to create some visualizations.
def get_top_n_words(corpus, n=1,k=1):
    vec = CountVectorizer(ngram_range=(k,k),stop_words = 'english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
```

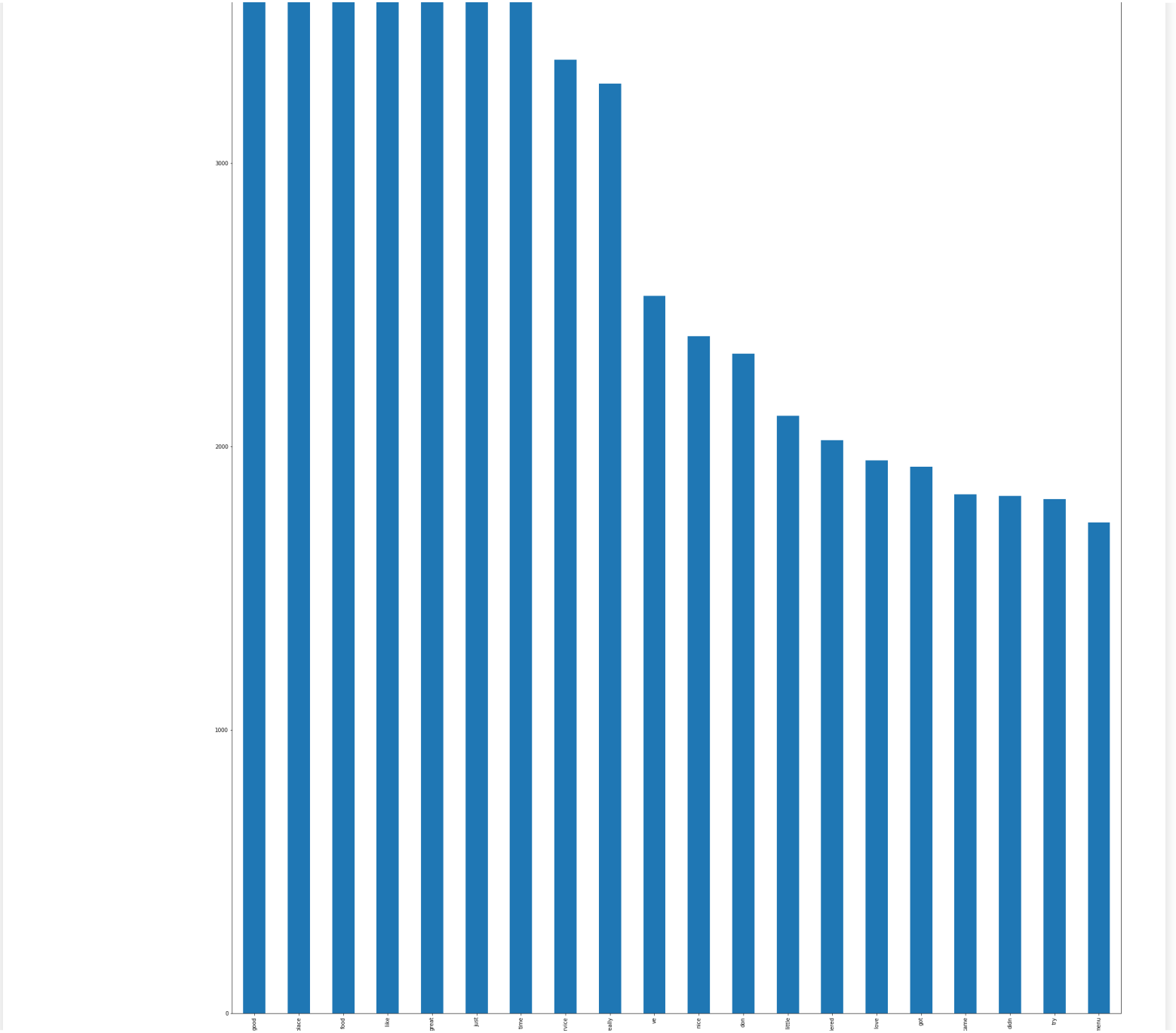
In [33]:
```
# We start by getting a list of the most common words.

common_words = get_top_n_words(data['text'], 20,1)
for word, freq in common_words:
    print(word, freq)
df = pd.DataFrame(common_words, columns = ['ReviewText' , 'count'])
df.groupby('ReviewText').sum()['count'].sort_values(ascending=False).plot(
    kind='bar',  title='Top 20 words from all reviews')
```

```
good 5818
place 5617
food 5449
like 4628
great 4449
just 4440
time 3716
service 3365
really 3280
ve 2531
nice 2390
don 2328
```

```
little 2109
ordered 2022
love 1951
got 1929
came 1830
didn 1825
try 1814
menu 1732
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7f622e1a6ba8>





Exercise 8:

8.1

Divide the data into "good reviews" (i.e. `stars` rating was greater than 3) and "bad reviews" (i.e. `stars` rating was less than 3) and make a bar plot of the top 20 words in each case. Are these results different from above?

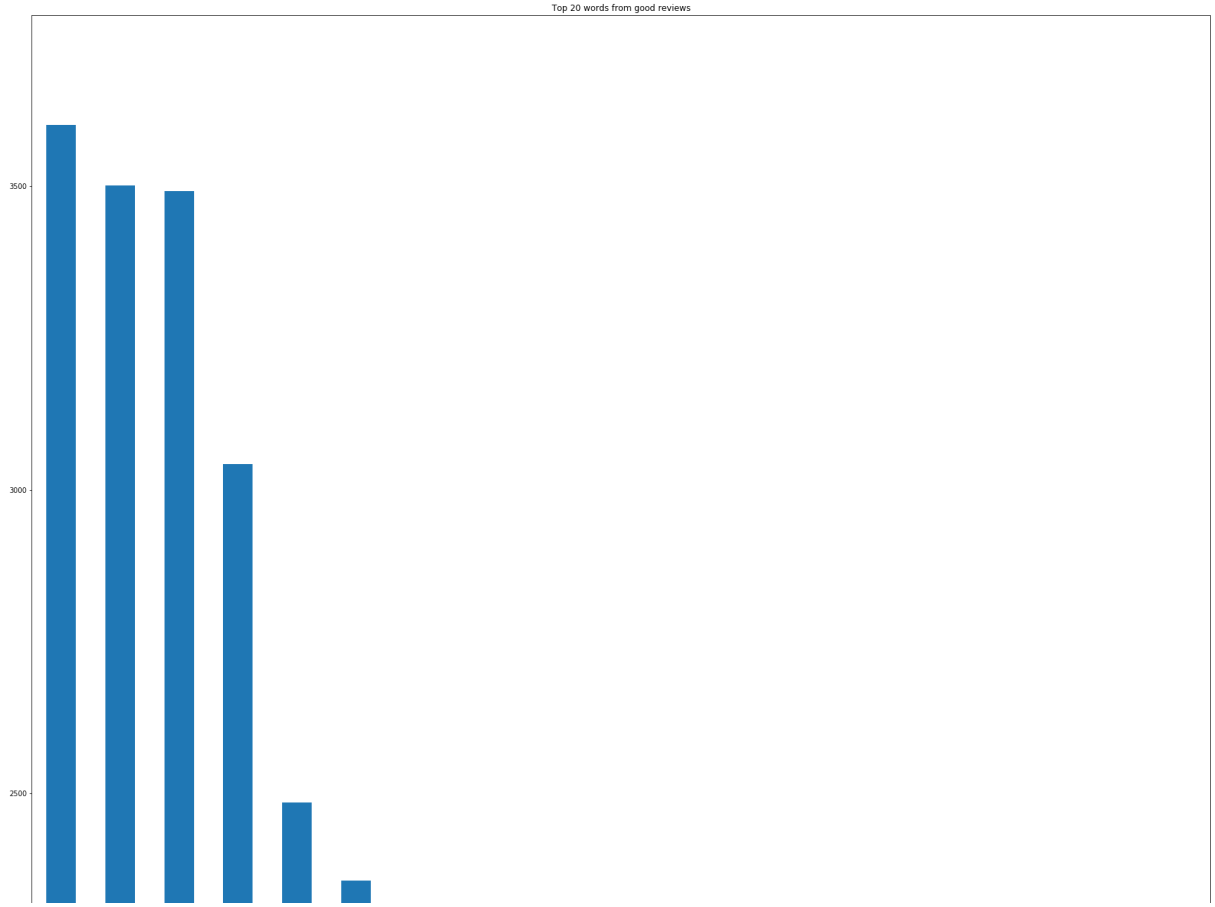
Answer. One possible solution is shown here:

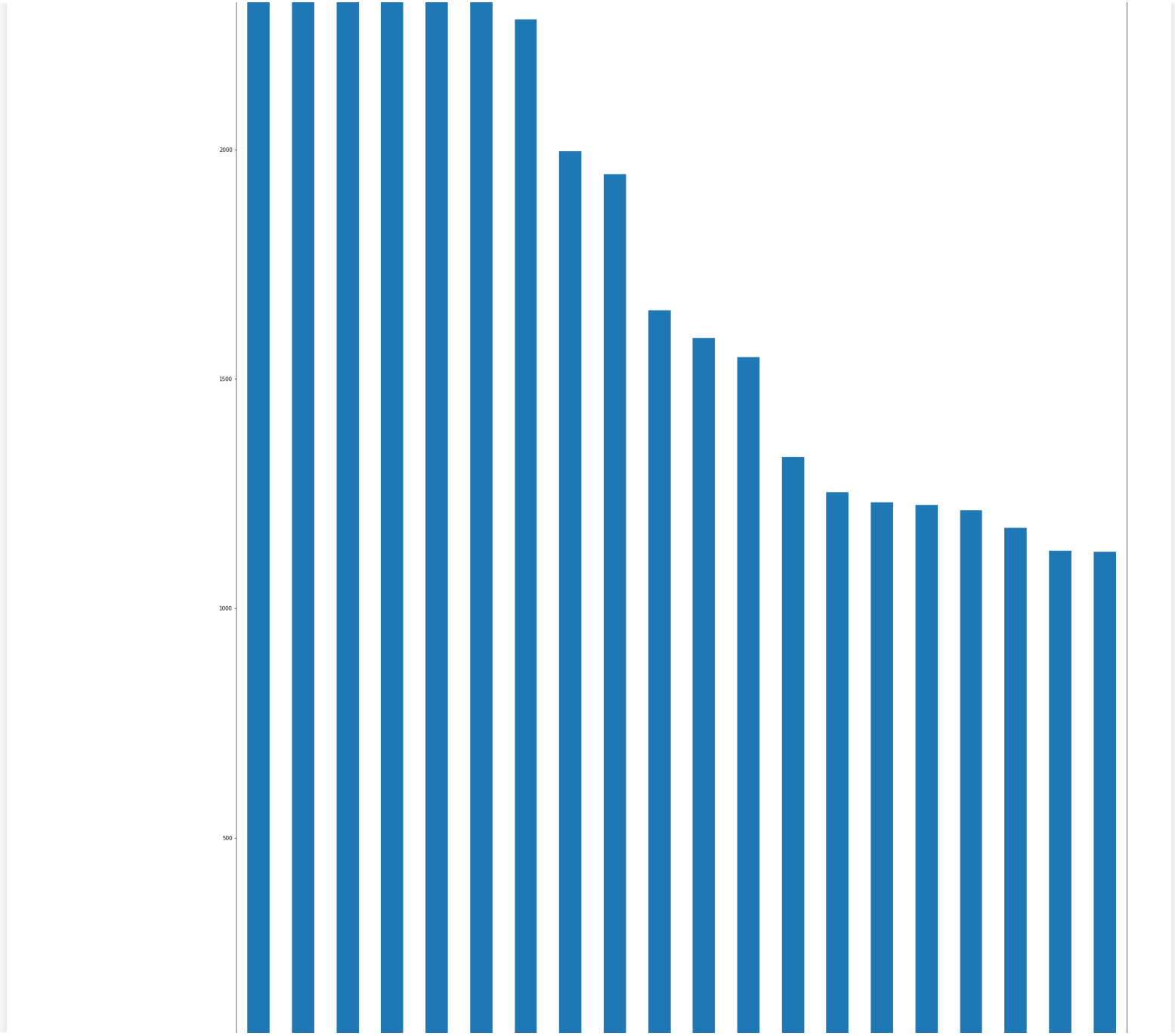
In [34]:
```
# We continue by splitting according to good/bad review scores, then gr
abbing again.

GoodInd = data['stars'] >3.1
GoodRev = data[GoodInd]
BadInd = data['stars'] <2.1
BadRev = data[BadInd]

common_words = get_top_n_words(GoodRev['text'], 20)
for word, freq in common_words:
    print(word, freq)
df = pd.DataFrame(common_words, columns = ['ReviewText' , 'count'])
df.groupby('ReviewText').sum()['count'].sort_values(ascending=False).pl
ot(
    kind='bar', title='Top 20 words from good reviews')
```

```
good 3602
place 3502
great 3492
food 3043
like 2485
just 2356
time 2284
really 1997
service 1947
```

```
ve 1650
nice 1590
love 1548
little 1330
don 1253
delicious 1231
friendly 1226
best 1214
definitely 1176
staff 1126
try 1123
```

Out[34]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f6226e97780>`

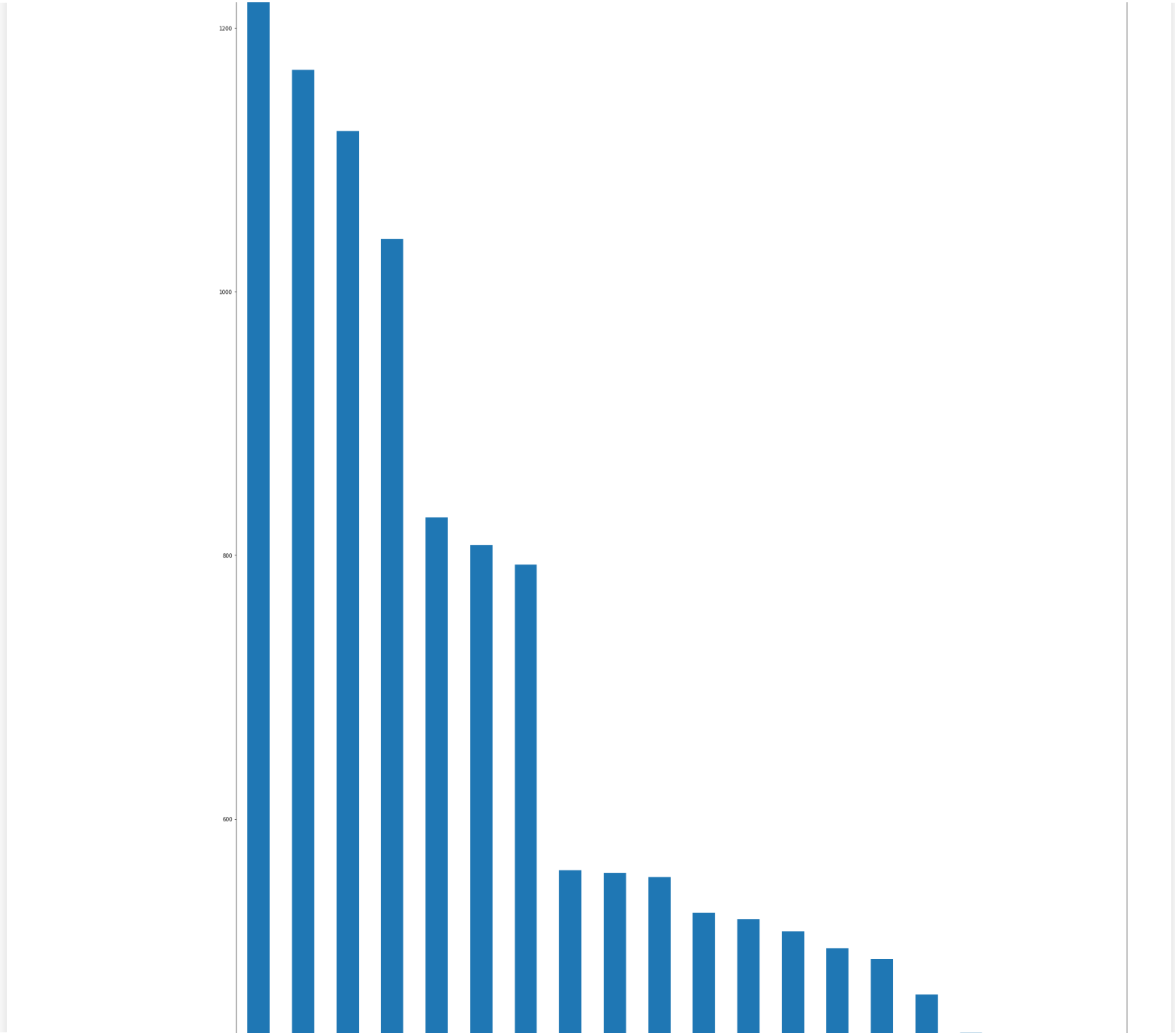


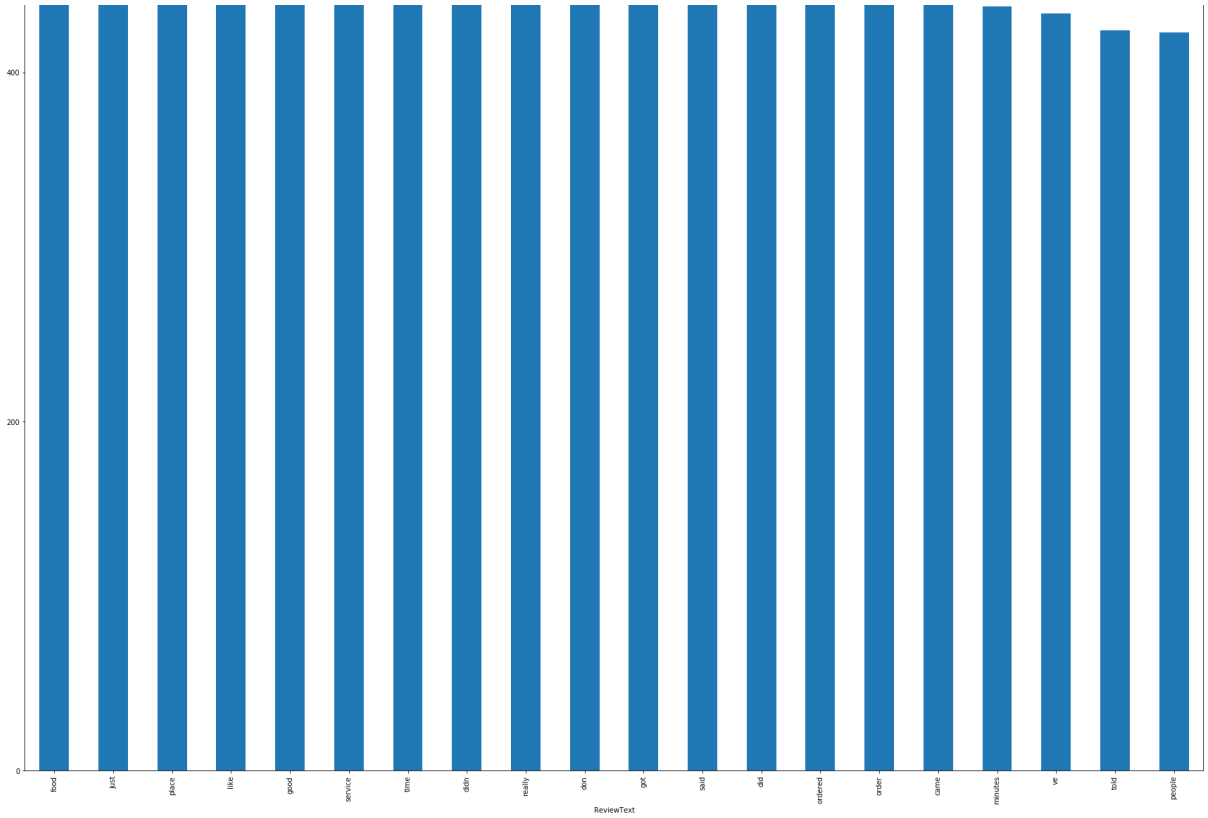


In [35]:
```
common_words = get_top_n_words(BadRev['text'], 20)
for word, freq in common_words:
    print(word, freq)
df = pd.DataFrame(common_words, columns = ['ReviewText' , 'count'])
df.groupby('ReviewText').sum()['count'].sort_values(ascending=False).pl
ot(
    kind='bar', title='Top 20 words from bad reviews')
```

```
food 1252
just 1168
place 1122
like 1040
good 829
service 808
time 793
didn 561
really 559
don 556
got 529
said 524
did 515
ordered 502
order 494
came 467
minutes 438
ve 434
told 424
people 423
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7f622dd86cc0>





Well, that was pretty useless. The "good" words are mostly a mix of generic words like "place" and overtly positive words like "good" itself.

The problem here is that we are dealing with single words, which cannot convey much information out of context. The natural solution then is to deal with n-grams, so that we can get context-aware results like "good burger" or "good service" (in the positive reviews) or, as we saw, "good 45 minutes" (in the negative reviews).

8.2

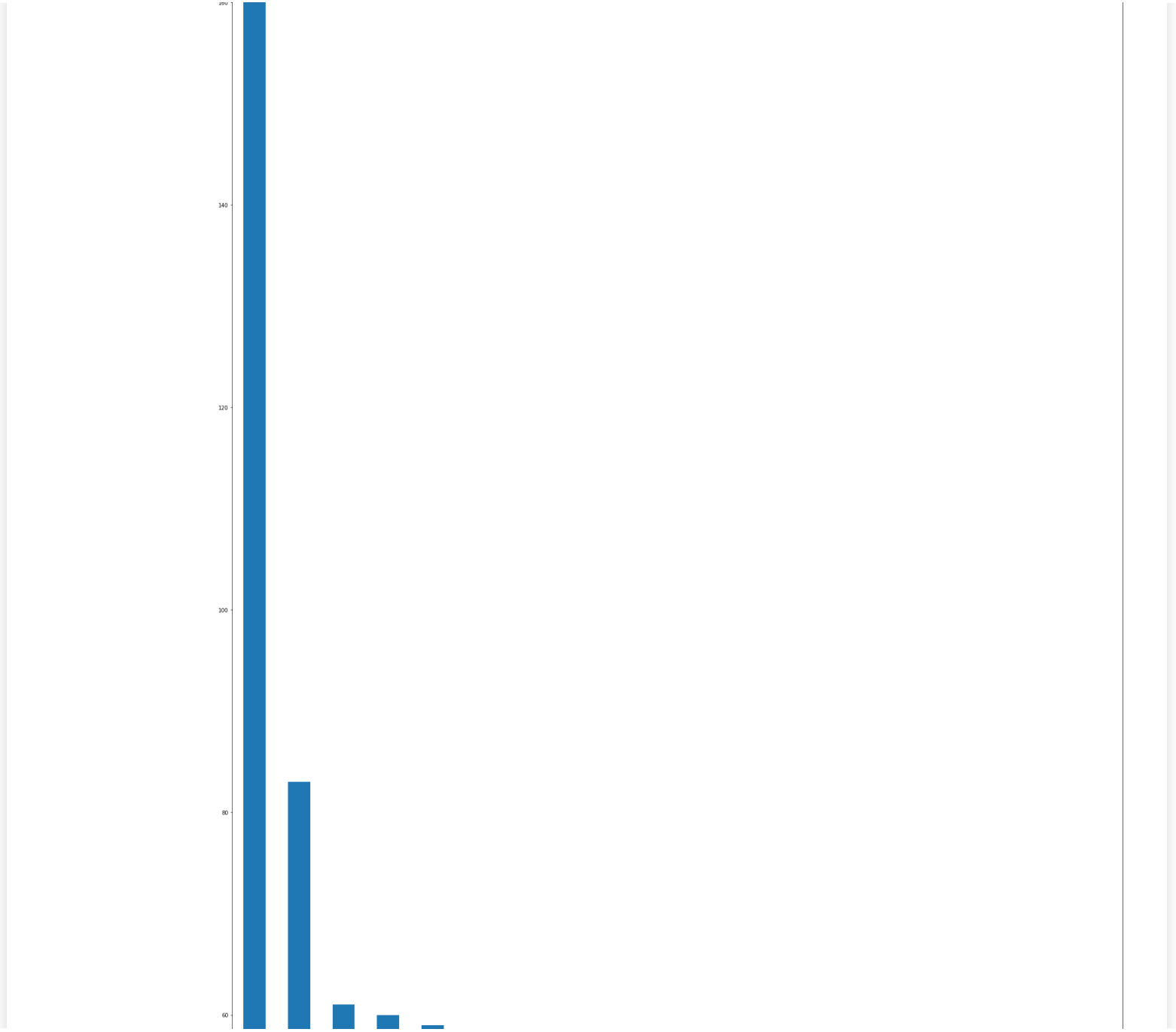
Use the `get_top_n_words()` function to find the top 20 bigrams and trigrams. Do the results seem useful?

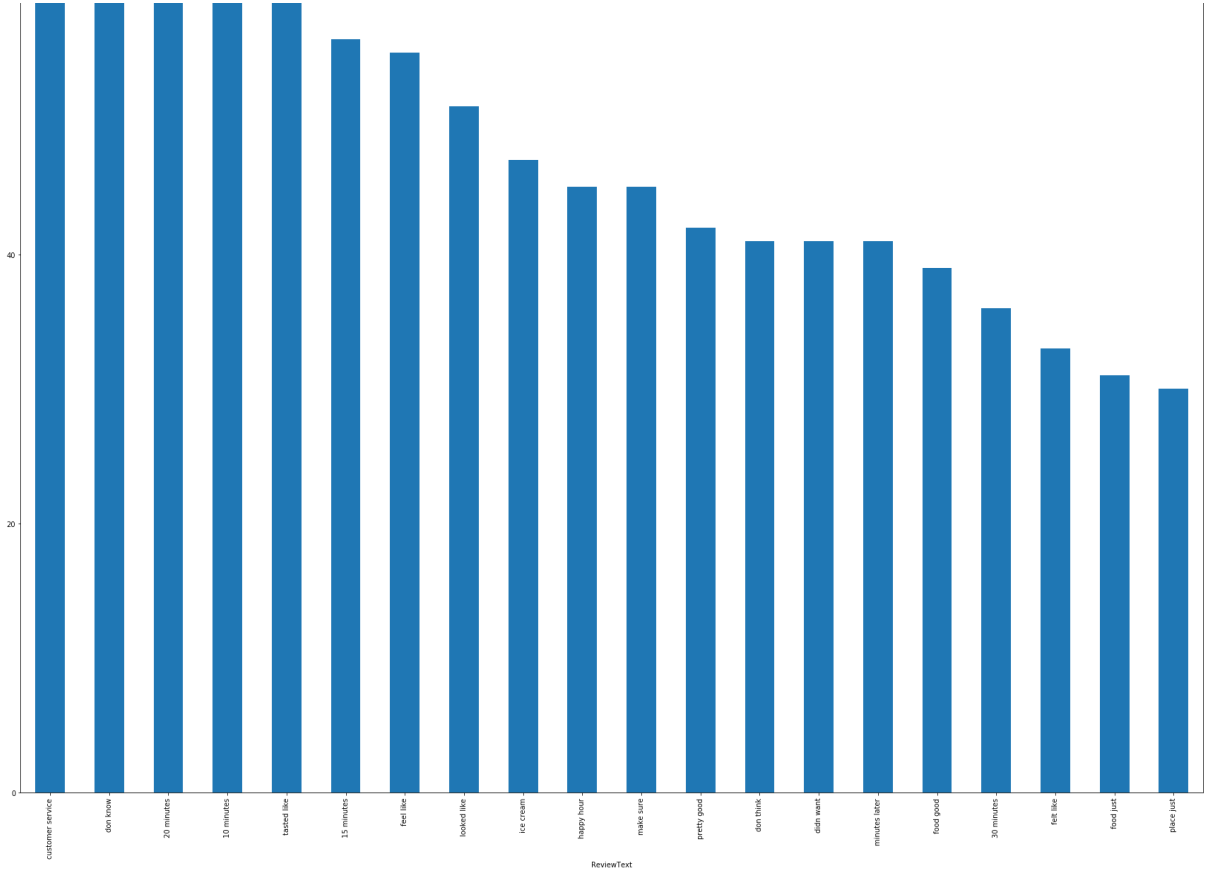
Answer. One possible solution is shown here:

In [36]:
```
# Get top bigrams and trigrams from bad reviews
common_words = get_top_n_words(BadRev['text'], 20,2)
for word, freq in common_words:
    print(word, freq)
df = pd.DataFrame(common_words, columns = ['ReviewText' , 'count'])
df.groupby('ReviewText').sum()['count'].sort_values(ascending=False).pl
ot(
    kind='bar', title='Top 20 bigrams from bad reviews')
```

```
customer service 163
don know 83
20 minutes 61
10 minutes 60
tasted like 59
15 minutes 56
feel like 55
looked like 51
ice cream 47
make sure 45
happy hour 45
pretty good 42
didn want 41
minutes later 41
don think 41
food good 39
30 minutes 36
felt like 33
food just 31
place just 30
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f622dd865c0>
```



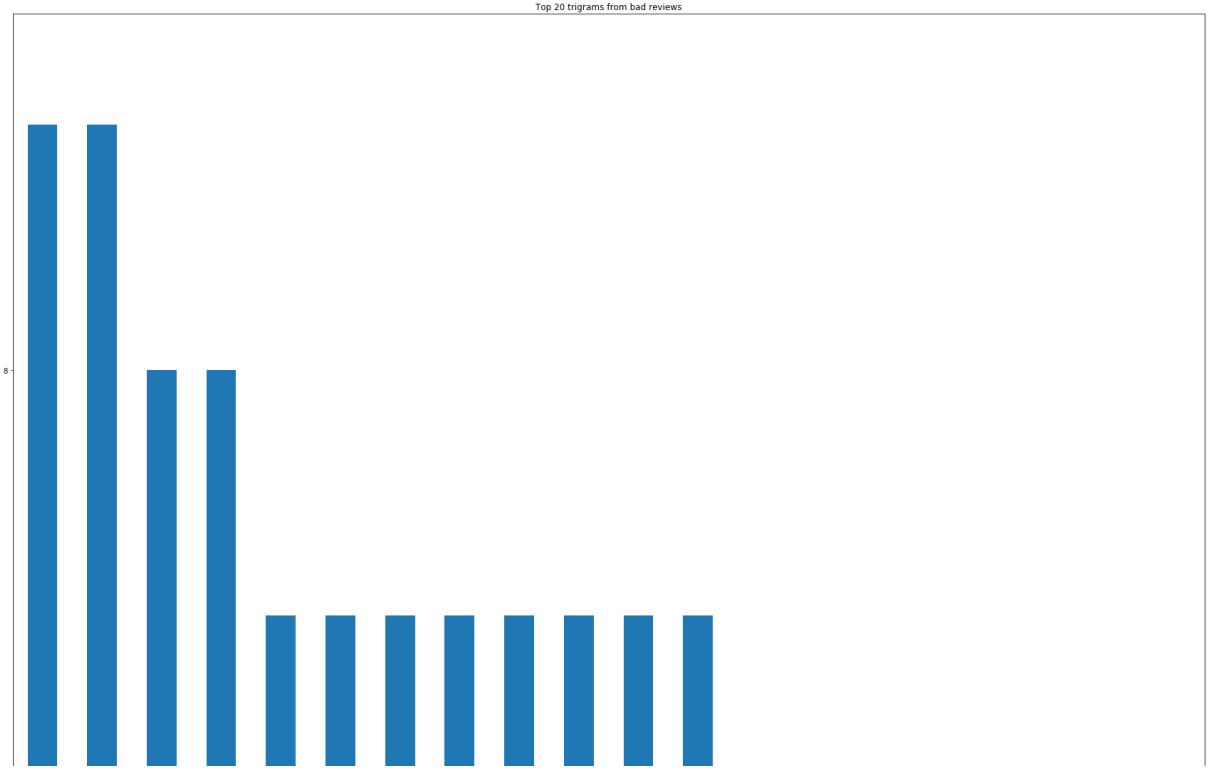


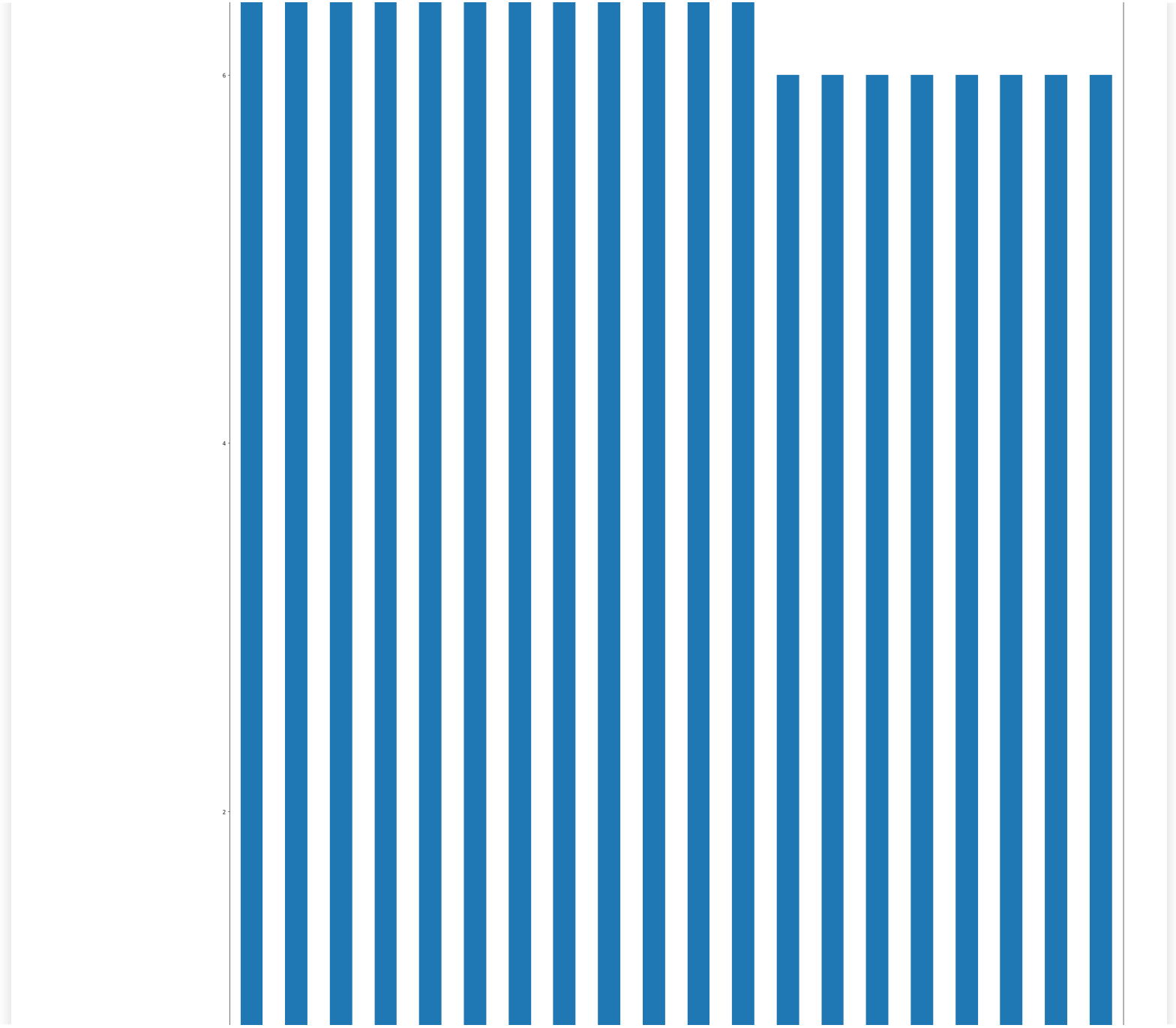
In [37]:
```
common_words = get_top_n_words(BadRev['text'], 20,3)
for word, freq in common_words:
    print(word, freq)
df = pd.DataFrame(common_words, columns = ['ReviewText' , 'count'])
df.groupby('ReviewText').sum()['count'].sort_values(ascending=False).pl
ot(
    kind='bar', title='Top 20 trigrams from bad reviews')
```

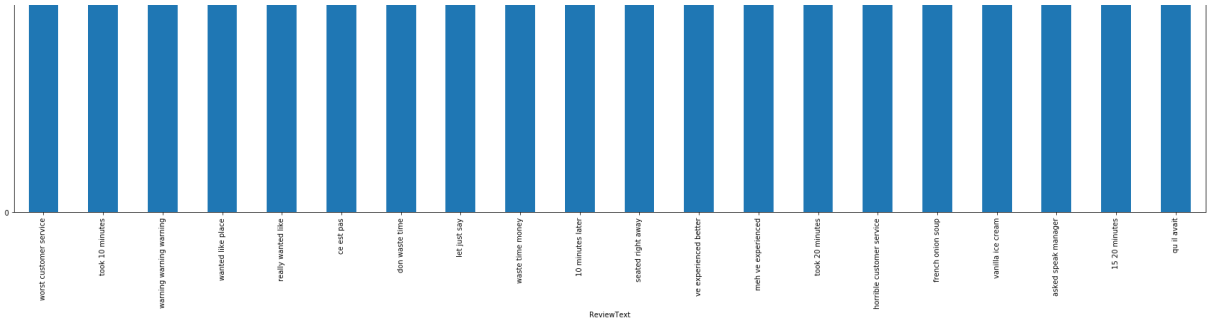
```
worst customer service 9
took 10 minutes 9
wanted like place 8
warning warning warning 8
don waste time 7
```

```
waste time money 7
let just say 7
seated right away 7
ve experienced better 7
ce est pas 7
really wanted like 7
10 minutes later 7
horrible customer service 6
vanilla ice cream 6
meh ve experienced 6
french onion soup 6
took 20 minutes 6
qu il avait 6
asked speak manager 6
15 20 minutes 6
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x7f622db6d940>



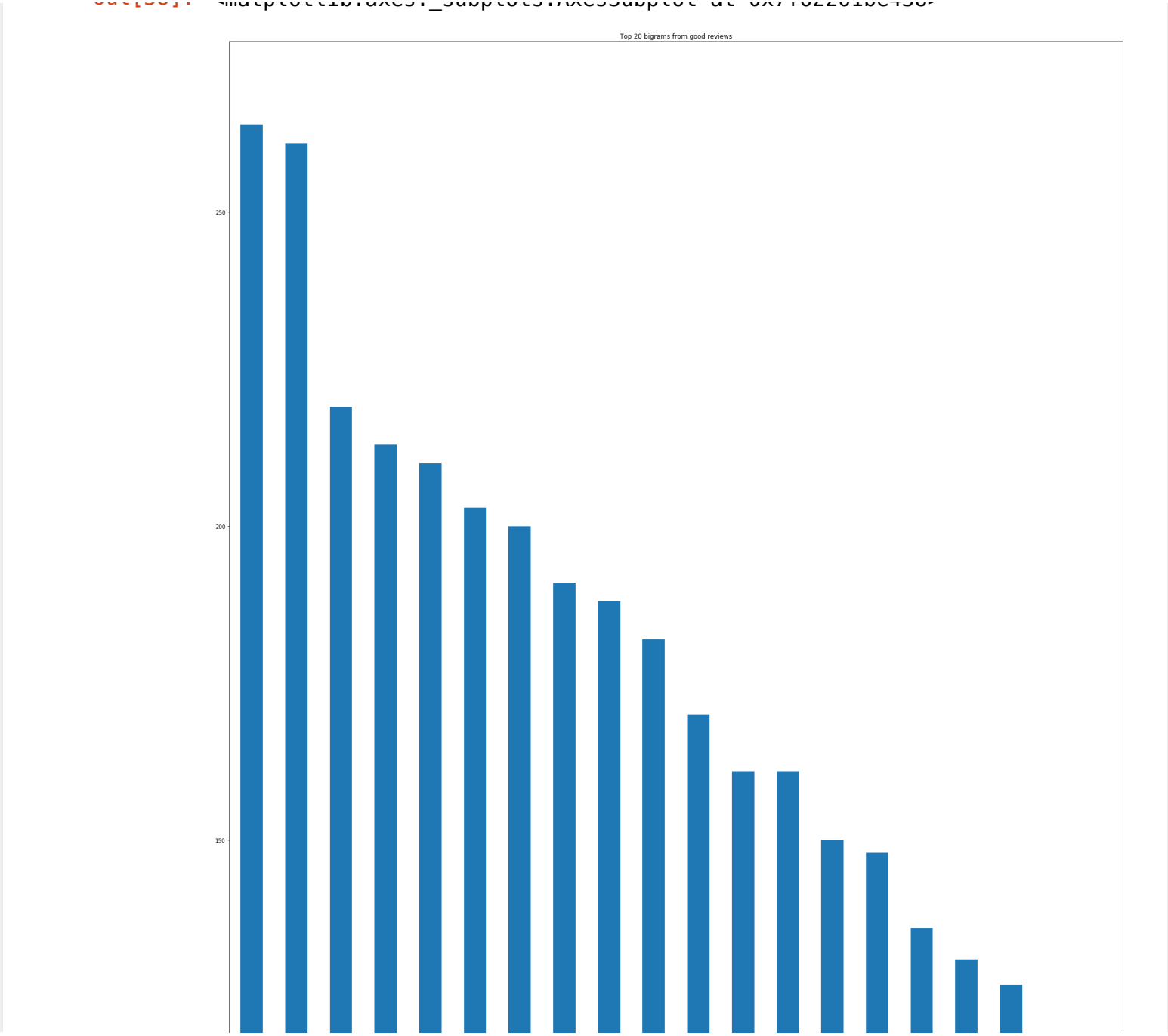


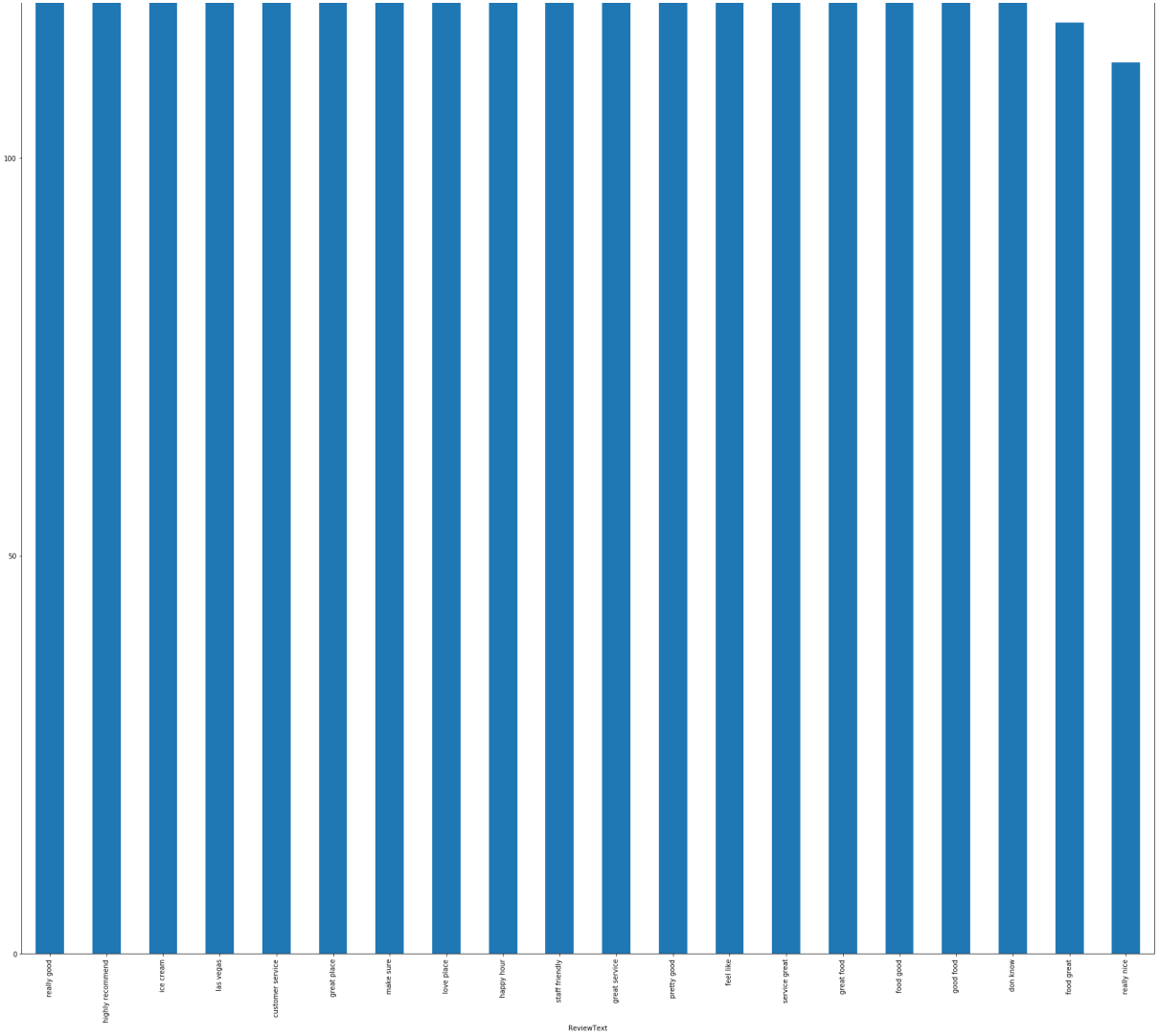


In [38]:
```
common_words = get_top_n_words(GoodRev['text'], 20,2)
for word, freq in common_words:
    print(word, freq)
df = pd.DataFrame(common_words, columns = ['ReviewText' , 'count'])
df.groupby('ReviewText').sum()['count'].sort_values(ascending=False).pl
ot(
    kind='bar', title='Top 20 bigrams from good reviews')
```

```
really good 264
highly recommend 261
ice cream 219
las vegas 213
customer service 210
great place 203
make sure 200
love place 191
happy hour 188
staff friendly 182
great service 170
pretty good 161
feel like 161
service great 150
great food 148
food good 136
good food 131
don know 127
food great 117
really nice 112
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7f62261be438>



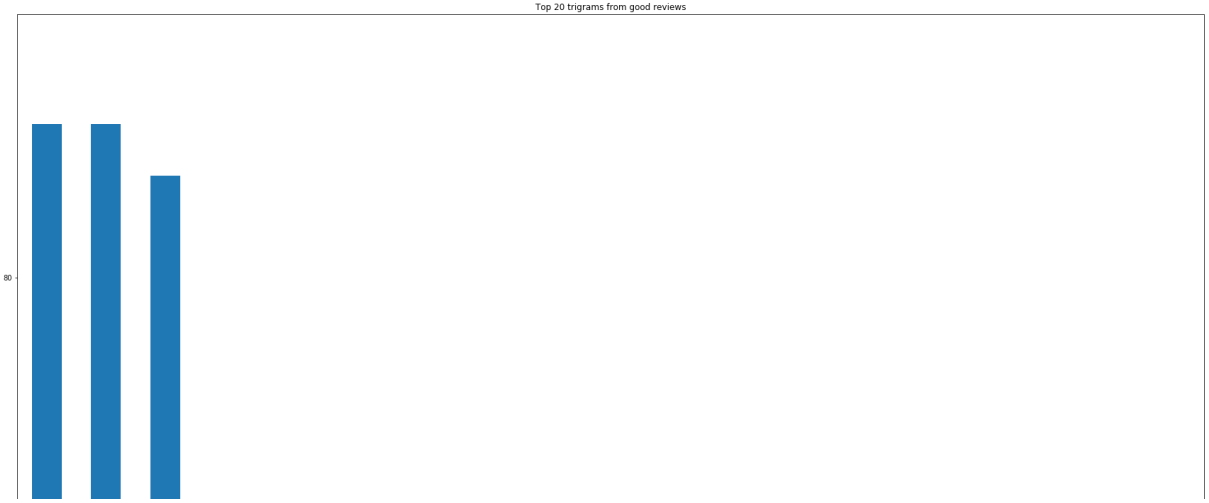


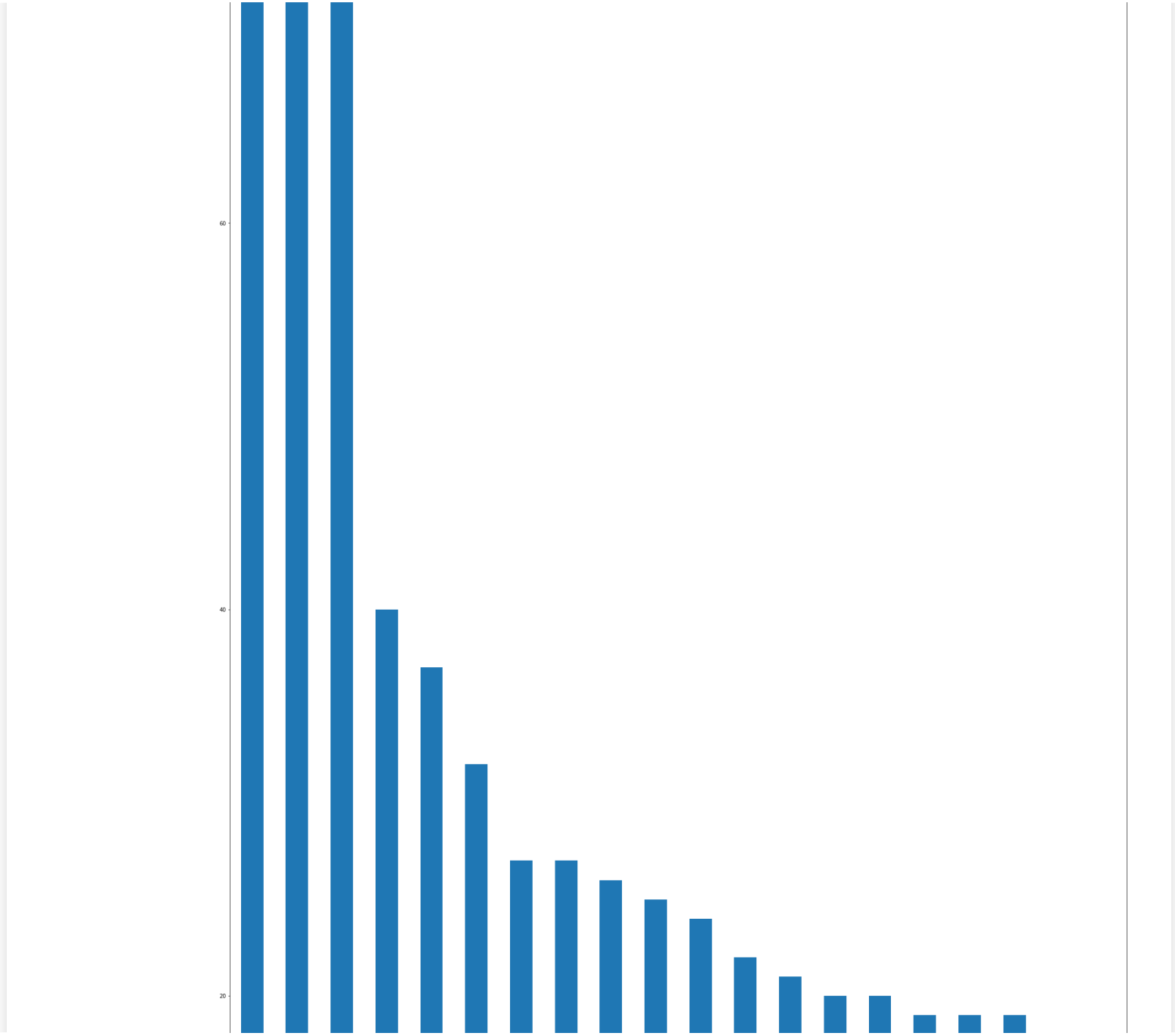
In [39]:
```
common_words = get_top_n_words(GoodRev['text'], 20,3)
for word, freq in common_words:
    print(word, freq)
df = pd.DataFrame(common_words, columns = ['ReviewText' , 'count'])
df.groupby('ReviewText').sum()['count'].sort_values(ascending=False).pl
```

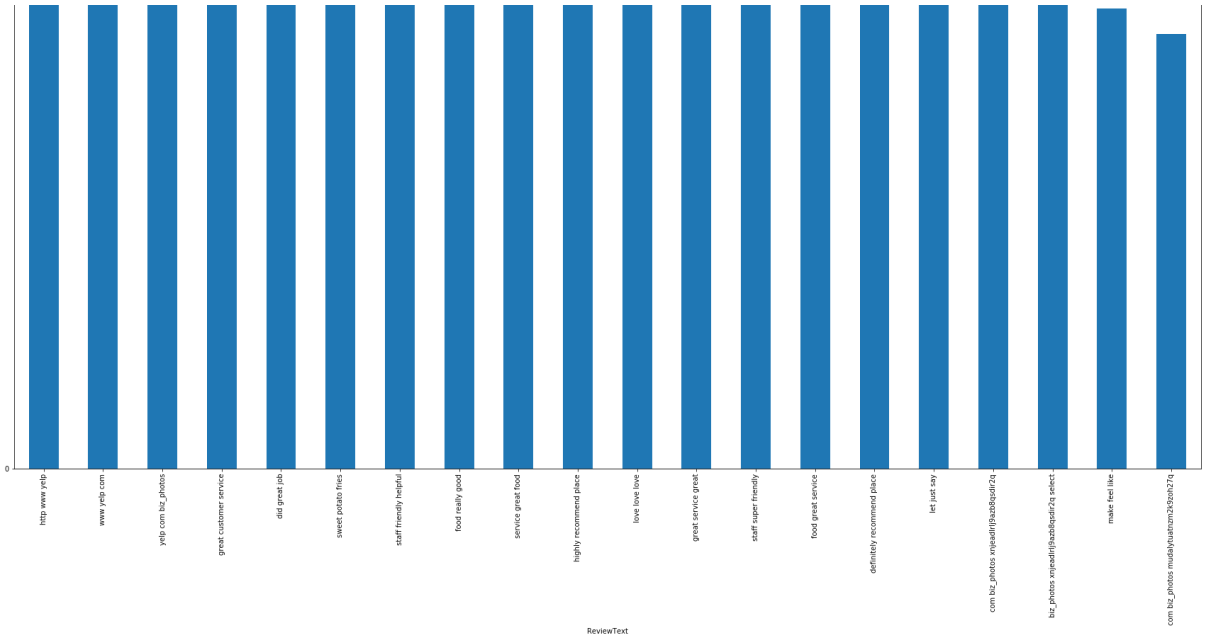
```
ot(
    kind='bar', title='Top 20 trigrams from good reviews')
```

```
http www yelp 86
www yelp com 86
yelp com biz_photos 84
great customer service 40
did great job 37
sweet potato fries 32
staff friendly helpful 27
food really good 27
service great food 26
highly recommend place 25
love love love 24
great service great 22
staff super friendly 21
definitely recommend place 20
food great service 20
let just say 19
com biz_photos xnjeadlrlj9azb8qsdir2q 19
biz_photos xnjeadlrlj9azb8qsdir2q select 19
make feel like 18
com biz_photos mudalytuatnzm2k9zoh27q 17
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7f622dfa86d8>







Although there are some nonsense entries, this is starting to be helpful. We can see a few recurring themes among good reviews (e.g. "staff friendly helpful"). Nonsense entries are typically difficult to eliminate completely in NLP with user-generated text and smaller corpora. NLP is still useful *despite* the existence of nonsense results, and we should think of the output of NLP algorithms like this as a *screening tool* for finding important phrases rather than a *careful estimate* of the most important phrases. In other words, it's an application of machine intelligence to *conduct exploratory analysis* rather than to *build predictive models*.

Question:

Look at the 5 most important bigrams for bad reviews. What *single, specific* problem seems to be the most important driver of bad reviews?

Three of the top 5 bigrams were "20 minutes", "15 minutes", and "10 minutes." These are all times, *strongly* suggesting that *waiting time for service* is a main driver for bad review scores.

Exercise 9:

9.1

You may have noticed that many of the important "bad" bigrams included the words "like" or "just" but didn't seem very informative (e.g. "felt like", "food just"). Give some ideas of how to use this sort of observation in future pre-processing of reviews, based on the pre-processing ideas we have already studied.

Answer. Two potential answers are (there are many others):

1. Having recognized that these words go together in common bigrams, you could modify your algorithm so that it "clumps" these bigrams together; i.e. treats them as one word, so that your algorithm will focus on the words following that.
2. Having recognized this as a key phrase, we could have a list of the most important words that follow these key phrases (which are presumably informative). This is more time-consuming as it requires human input.

9.2

Building on the previous question, we note that most of the most important complaints and compliments can't be *completely* observed by looking at bigrams or trigrams. This can often be fixed by small modifications. Do the following:

1. Write down a complaint that is unlikely to be (completely) picked up by bigram analysis. Hint: what might you write if your hamburger was served cold?
2. Write down a processing step that would fix this problem. Try to find a solution that would work for several similar problems without additional human input.

Answer. There are many good answers, but we focus on a simple one:

1. I would probably write "the burger was cold" or "the burger was served cold." If this were a common complaint, the most-important bigrams might include "was cold" or "served cold," which doesn't tell me *what* was served cold.
2. A simple fix, along the lines of the previous question, would be to "clump" words like "was cold" together. However, I think this is a bad fix, as it focuses too much on the word "cold" and would require a great deal of hand tuning. A better idea would be to recognize that words like "was" are always going to be a problem in this context, as they separate the important noun describing the subject ("burger") from the adjective describing the problem ("cold"). This suggests that we should take a *much* more aggressive stance towards removing stop words. This should certainly include conjugations of "to be," and likely many other common but uninformative words (like "too").

This second response is an important takeaway for NLP – this sort of problem is extremely common, and a great deal of time is often spent tweaking initial pre-processing rules. In the final part of this case, you will learn about a method that can help systematically deal with these uninformative stop words.

Regular Expressions

Having spent a lot of time on n-grams and how to featurize a document using them, we now take a break from `nltk` tools to introduce the most important text wrangling tool in Python (and many other languages): **regular expressions**.

The basic idea here is that you often want to perform some specific transformation (e.g. delete or substitute) every time that some possibly-complicated pattern (e.g. the letter 'A', the word 'hello', any word containing the letters 'a','r' in that order) occurs. Regular expressions are a compact and powerful language for expressing these sorts of patterns. This is super important whenever you are trying to clean a text dataset that contains thematically similar, but not exactly, the same errors.

In Python, the `re` module provides regular expression matching operations and common operations. Regular expressions are a deep subject, with some documentation here: https://docs.python.org/3/library/re.html?highlight=regex.

As some simple examples, we have:

1. `.` matches any character except \n (newline)
2. `\d` matches any digit (this can also be written as [0-9])
3. `\D` matches any non-digit (this can also be written as [^0-9])
4. `\w` matches any alphanumeric character ([a-zA-Z0-9_])
5. `\W` matches any non-alphanumeric character ([^a-zA-Z0-9_])

As some more complex examples, regular expressions also allow you to quantify the number of times matches can occur. For example,

1. `[a-d]+` matches any time you get $\{a, b, c, d\}$ one or more times in a row
2. `[a-d]{3}` matches any time you get them exactly 3 times in a row
3. `[a-d]*` matches any time you get them 0 or more times in a row

For now, we give a simple application based on the `re.sub()` function, which substitutes words that match a pattern:

In [40]:
```
import re
sentence = 'That was an "interesting" way to cook bread.'
pattern = r"[^\w]" # the ^ character denotes 'not',
#                  the \w character denotes a word, and []  means
#                   anything that matches anything in the brackets.
#                    Together, this refers to any character that is no
t a word.
print(re.sub(pattern, " ", sentence))
```

```
That was an  interesting  way to cook bread
```

In [41]:
```
str = "Natesh loves all the foold and loveds sdaslo"
x   = re.compile('lo')
iterator = x.finditer(str)
for item in iterator:
    print(item.span())
    print(item.group())
```

```
(7, 9)
```

```
lo
(31, 33)
lo
(42, 44)
lo
```

Exercise 10:

10.1

1. Use the `re.split()` function to split the first Yelp review into a list of its constituent words.
2. Use the `re.findall()` function to search the first 30 reviews for the number of times they contain the word "food". Print the maximum number of times the word "food" is mentioned in a single review.

Answer. One possible solution is shown below:

In [42]:
```
re.split(r'\s', AllReviews.values[0])
```

Out[42]:
```
['Super',
 'simple',
 'place',
 'but',
 'amazing',
 'nonetheless.',
 "It's",
 'been',
 'around',
 'since',
 'the',
 "30's",
 'and',
 'they',
 'still',
```

```
 'serve',
 'the',
 'same',
 'thing',
 'they',
 'started',
 'with:',
 'a',
 'bologna',
 'and',
 'salami',
 'sandwich',
 'with',
 'mustard.',
 '',
 '',
 'Staff',
 'was',
 'very',
 'helpful',
 'and',
 'friendly.']
```

In [43]:
```
food_count = []
for sentence in AllReviews.values:
    temp = len(re.findall('food', sentence))
    food_count.append(temp)
print(max(food_count[0:30]))
```

```
2
```

10.2

Using regular expressions, find the percentage of reviews in top 500 reviews that have numbers in them.

Answer. One possible solution is given below:

In [44]:
```
### Considering the top 500 reviews for this analysis
top_500_reviews = AllReviews.values[:500]
reviews_nos_regex = []

for each_review in top_500_reviews:
    number_list = re.findall('\d',each_review)

    ## number list returns all the possible digits in a review
    ## Look if the number list is empty - if so, the review has no digi
ts in them
    if(len(number_list)) > 0:
        reviews_nos_regex.append(each_review)
```

In [45]:
```
len(reviews_nos_regex)/len(top_500_reviews)
```

Out[45]:
```
0.326
```

As is clear from above, regular expressions are very useful for extracting more general properties of text. These properties are not as informative or context-aware as n-grams can be, but they are much simpler to code and therefore can often serve as the first step of an EDA on text data.

Although regular expressions usually cannot tell us much about context overall, they *can* be used to find specific instances of words in context. For example, we may be interested in finding the first word following "good" or "bad" in a review (which can help us distinguish a positive from a negative review). Let's write some code that finds the first word following "good" in the sentence "hello I want a good burger, please.":

In [46]:
```
sample = "hello I want a good burger, please"

# Find everything after "good", including "good"

post = re.findall(r'good.*', sample)[0]

print(post)
```

```
# Take just the first word after "good"

first_post = re.split(r'\s',post)[1]

print(first_post)
```

```
good burger, please
burger,
```

10.3

Using the above as a template, write a generalized function that can extract the first word following "good". Don't forget to include a default behavior for when the word doesn't appear in the sentence. Run this function for all reviews and print the first 300 results for reviews that do contain the word "good".

Answer. One possible solution is given below:

In [47]:
```
def next_word(sentence):
    post = re.findall(r'good.*', sentence)
    if (len(post) > 0):
        temp = re.split(r'\s',post[0])
        if (len(temp) > 1):
            return(temp[1])
        else:
            return('')
    else:
        return('')


print(next_word(sample))
```

```
burger,
```

In [48]:
```
post_good = []
ind = 0
for sentence in AllReviews.values:
```

```
    temp = next_word(sentence)
    post_good.append(temp)

nonempty = [i for i in post_good if i]
print(nonempty[0:300])
```

```
['Burgers', 'bully', 'The', 'burgers', 'things', 'Too', 'size', 'they',
'45', 'Soft', '-', 'and', 'and', 'for', 'Hot', 'for', 'Price', 'for',
'really', 'ramen.', 'First', 'The', 'size', 'Definitely', '-', 'I', 'a
s', 'deal!', 'with', 'in', 'price', 'as', 'food', 'What', 'to', 'Finall
y,', 'actually.', 'The', 'but', 'experiences', 'and', 'however', 'if',
'so', '(she', 'experiences', 'experiences', 'size', 'experience', 'Thi
s', 'My', 'but', 'at', 'Mexican', '(for', 'and', 'the', 'ones,', 'but',
'My', 'I', 'The', 'but', 'but', 'food', 'size', "I'm", 'reviews.',
"I'd", 'but', 'idea', 'things,', 'The', 'as', 'as', 'good,', 'as', 'Bu
t', 'opinions', 'selection', 'so', 'although', 'and', 'little', 'pulle
d', 'time', 'Crispy', 'reviews,', 'reviews', 'Mexican', 'service', 'rev
iews.', 'is', 'and', 'to.', 'directions', 'service.', 'and', 'french',
'part', 'for', 'He', 'Besides', 'and', 'and', 'experience.', 'time.',
'burger!', 'and', 'work', 'and', 'But', 'thing.', 'sign.', 'time.',
'I', 'in', 'so', 'and', 'regulars.', 'were', 'with', 'for', 'except',
'but', 'spot.', 'as', 'until', 'having', 'and', 'Loved', 'Not', 'Timel
y', 'and', 'custard', 'too,', '-', 'non-fast-food', 'care', 'and', 'dea
l.', 'but', 'view', 'drinks,', 'idea', 'time,', 'reason', 'the', 'her
e.', "it's", 'and', 'as', 'at', 'service', 'service.', 'music', 'flavo
r.', 'flavor', 'sized', 'flavors.', 'dive', 'sized,', 'seat', 'eggs,',
'sized', 'view', 'flavor', 'at', 'the', 'and', 'spot', 'burger', 'and',
'location.', 'steakhouse', 'time.', 'sized.', 'The', 'I', 'and', 'of',
'as', 'number', 'views', 'dining', 'part', 'convention', 'wine', 'assor
tment', 'views.', 'sized', 'place', 'for', 'flavor.', 'amount', 'off-st
rip', 'at', 'The', 'place', 'option', 'He', 'casual', 'move', 'ole', 'a
t', 'though.', 'wine', 'but', 'have', 'very', 'deal', 'you', 'couple',
'measure.', 'mainly', 'size', 'eats,', 'time', 'thing', 'Szechuan.', 'j
ob', 'I', 'smoked', 'taste', 'things', 'for', 'and', 'choice', 'is', 'b
ang', 'for', 'But', 'alfredo', 'dive', 'filler', 'dive', 'on', 'as', 'f
amily', 'decision', 'afloat', 'to', 'just', 'thing', 'it', 'experienc
e', 'in', 'taste.', 'to', 'to', 'kick', 'people,', 'quality', 'for', 'W
e', 'and', 'experiences', 'place', 'menu', 'food', 'flavour,', 'burge
r', 'food', 'service', 'brisket,', 'Oh', 'price', 'So', 'when', 'busine
ss', 'theres', 'visit.', 'If', 'plenty', 'Not', 'servers.', 'which', 't
```

```
he', 'to', 'place', 'as', 'as', '15', 'It', 'I', 'but', 'my', 'as', 't
o', 'The', 'at', 'it', 'weight.', 'songs,', 'success!', 'to', 'represen
tation', 'for', 'Persian', 'luck']
```

Skimming over the results of the previous exercise, a few things stood out:

1. People like to talk about good burgers – this appeared 5 times in the first 300 results.
2. A large number of the results are useless. One common problem is the occurrence of a sentence boundary; e.g. "good. The" near the start of the list. In this case, we should look *before* the word "good" rather than after. However, we can't look *immediately* before good – that word will usually be some conjugation of "to be", which is also not informative – rather, we need to look for a word before "good" that isn't too boring. Other times, there is a word following "good" that is uninformative; e.g. "good for" – we want to know *what* something was good for! In this case, we should keep on skimming *forward* until we see a word following "good" that isn't too boring.

In both of these cases, we can't use simple regular expressions by themselves, as regular expressions don't know how to ignore "boring" words. Regular expressions can only help us filter for the structure of words, not the content they convey within a context. We *can* use what we learned about stop words to remove these from the reviews before conducting the above analysis, but as we have been, we will still sometimes get not very informative phrases like "was cold" or "served cold". So we will introduce an alternative method, which can be applied to serve as an even better remover of stop words: **part-of-speech tagging**.

Part-of-speech (POS) tagging

In English, there are eight main parts of speech - `nouns`, `pronouns`, `adjectives`, `verbs`, `adverbs`, `prepositions`, `conjunctions` and `interjections`. These are `sustantivos`, `pronombres`, `adjetivos`, `verbos`, `adverbios`, `preposiciones`, `conjunciones` and `interjecciones`, respectively, in Spanish. The purpose of POS tagging is to label each word in a document with its part of speech.

Unsurprisingly, POS tagging can be very difficult to do by hand. `nltk` has a default function for this, called `nltk.pos_tag()`, which we will use. As a word of warning, this function is far from

infallible, especially on informal text (e.g. website reviews, forum posts, text messages, etc), and words in English often exhibit POS drift (e.g. the drift of "Google" from noun to verb):

In [49]:
```
import nltk
nltk.download('averaged_perceptron_tagger')
#https://www.nltk.org/book/ch05.html
text_word_token = nltk.word_tokenize("Natesh is having a good day")
#text_word_token = nltk.word_tokenize(data.text[0])
nltk.pos_tag(text_word_token)
#https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

Out[49]:
```
[('Natesh', 'NNP'),
 ('is', 'VBZ'),
 ('having', 'VBG'),
 ('a', 'DT'),
 ('good', 'JJ'),
 ('day', 'NN')]
```

In [50]:
```
text_word_token = nltk.word_tokenize("We are going to Race") # try "Race can be both a verb and a noun"
#text_word_token = nltk.word_tokenize(data.text[0])
nltk.pos_tag(text_word_token)
```

Out[50]:
```
[('We', 'PRP'), ('are', 'VBP'), ('going', 'VBG'), ('to', 'TO'), ('Race', 'VB')]
```

In [51]:
```
#https://www.nltk.org/_modules/nltk/tag/perceptron.html
nltk.pos_tag(words)
```

Out[51]:
```
[('Super', 'NNP'),
 ('simple', 'JJ'),
 ('place', 'NN'),
 ('but', 'CC'),
 ('amazing', 'JJ'),
```

```
('nonetheless', 'RB'),
('.', '.'),
('It', 'PRP'),
("'s", 'VBZ'),
('been', 'VBN'),
('around', 'IN'),
('since', 'IN'),
('the', 'DT'),
('30', 'CD'),
("'s", 'POS'),
('and', 'CC'),
('they', 'PRP'),
('still', 'RB'),
('serve', 'VBP'),
('the', 'DT'),
('same', 'JJ'),
('thing', 'NN'),
('they', 'PRP'),
('started', 'VBD'),
('with', 'IN'),
(':', ':'),
('a', 'DT'),
('bologna', 'NN'),
('and', 'CC'),
('salami', 'NN'),
('sandwich', 'NN'),
('with', 'IN'),
('mustard', 'NN'),
('.', '.'),
('Staff', 'NNP'),
('was', 'VBD'),
('very', 'RB'),
('helpful', 'JJ'),
('and', 'CC'),
('friendly', 'JJ'),
('.', '.')]
```

NLTK provides documentation for each tag, which can be queried using the tag itself; e.g. `nltk.help.upenn_tagset('RB')`. Since POS is context-sensitive, POS-taggers must

usually be trained on an existing corpus that has been tagged by professional linguists (possibly alongside unlabeled data to take advantage of semi-supervised methods). The most popular tag set is called the Penn Treebank set:

In [52]:
```
# We can get more details about any POS tag using the help function of
 nltk
nltk.download('tagsets')
nltk.help.upenn_tagset('CD$')
nltk.help.upenn_tagset('NN$')
```

```
[nltk_data] Downloading package tagsets to /root/nltk_data...
```

```
CD: numeral, cardinal
    mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-
    seven 1987 twenty '79 zero two 78-degrees eighty-four IX '60s .025
    fifteen 271,124 dozen quintillion DM2,000 ...
NN: noun, common, singular or mass
    common-carrier cabbage knuckle-duster Casino afghan shed thermostat
    investment slide humour falloff slick wind hyena override subhumani
ty
    machinist ...
```

```
[nltk_data]   Unzipping help/tagsets.zip.
```

Exercise 11:

11.1

Write code to find the percentage of reviews in the first 500 reviews of the dataset that contains a number or a cardinal using POS taggings only. (Hint: POS tag `CD` is the indicator for cardinal or number.) How does this compare to the figure we extracted from using regular expressions only?

Answer. One possible solution is given below:

In [53]:
```
cardinal_review = []
```

```
for sentence in top_500_reviews:
    words = nltk.word_tokenize(sentence)
    cd_len = [k for k,v in nltk.pos_tag(words) if 'CD' == v]

    if len(cd_len) > 0:
        cardinal_review.append(sentence)

#### Proportion of reviews with a number/cardinal in it
len(cardinal_review)/len(top_500_reviews)
## 56.6% of the reviews have a number/cardinal in the top 500 reviews.
## You could improve the accuracy of this estimate by looking at more t
han 500 reviews.
```

Out[53]: 0.566

This number is considerably higher than the one we got from using regular expressions only! The reason is because POS tagging can extract numbers in text form (e.g. "one", "two") whereas regular expressions cannot. This is one advantage of using POS tagging over regular expressions.

11.2

Extract all of the nouns from each review using POS tagging. This may be useful for later analysis. Even though words like "good" may be the most prevalent in good reviews, we think nouns like "service" or "burgers" are likely to be more informative.

Answer. One possible solution is given below:

In [54]:
```
noun_reviews = []
for sentence in AllReviews.values:
    words = nltk.word_tokenize(sentence)
    noun_pt = [k for k,v in nltk.pos_tag(words) if 'NN' == v]
    noun_reviews.append(noun_pt)
```

Exercise 12:

Use POS tagging to find the first word following "good" that has an interesting POS tag. We leave this up to your discretion, but should probably include nouns and proper nouns. Inspecting the above, we think that cardinals are also almost certainly interesting: we recognize that "good 45" is probably followed by "minutes", definitely an important (though not "good") part of a review!

Answer. One possible solution is shown below:

In [58]:
```
# First, a function that extracts only the "interesting" parts of speec
h

sentence = "This is a good burger, but I prefer pizza"
words = nltk.word_tokenize(sentence)
interesting = [k for k,v in nltk.pos_tag(words) if v in ['CD','FW','NN'
,'NNS','NNP','NNPS']]

print(interesting)
```

```
['burger', 'pizza']
```

In [59]:
```
good_pos = ['CD','FW','NN','NNS','NNP','NNPS']

def ExtractInteresting(sentence, good):
    words = nltk.word_tokenize(sentence)
    interesting = [k for k,v in nltk.pos_tag(words) if v in good]
    return(interesting)
```

In [60]:
```
# Next, a function that extracts the first "interesting" word that foll
ows "good"

sentence = "This is a good burger, but I prefer pizza"
post = re.findall(r'good.*', sentence)
print(post)
# Check that post isn't empty here before doing next line
temp = ExtractInteresting(post[0],good_pos)
```

```
print(temp)
# Check that temp isn't empty here before doing next line
print(post[0])
```

```
['good burger, but I prefer pizza']
['burger', 'pizza']
good burger, but I prefer pizza
```

In [61]:
```
def next_word2(sentence):
    post = re.findall(r'good.*', sentence)
    if (len(post) > 0):
        temp = ExtractInteresting(post[0],good_pos)
        if (len(temp) > 0):
            return(temp[0])
        else:
            return('')
    else:
        return('')


print(next_word2(sentence))
```

```
burger
```

In [62]:
```
# Finally, apply this.

post_good = []
ind = 0
for sentence in AllReviews.values:
    temp = next_word2(sentence)
    post_good.append(temp)

nonempty = [i for i in post_good if i]
print(nonempty[0:300])
```

```
['Burgers', 'sticks', 'bread', 'burgers', 'things', 'kitchen', 'size',
'smoothies', '45', 'way', '2', 'lunch', 'drink', 'Hot', 'groups', 'Pric
e', 'place', 'options', 'ramen', 'First', 'restaurant', 'size', 'item
s', 'bit', 'one', 'spot', 'deal', 'setup', 'chocolate', 'price', 'foo
```

```
d', 'finesse', 'poutinerie', 'hairdresser', 'bell', 'waiter', 'cream',
'experiences', 'combinations', 'fudge', 'nothing', 'ya', 'cappuccino',
'experiences', 'experiences', 'size', 'experience', 'sign', 'sister',
'time', 'Mexican', 'try', 'peppers', 'cauliflower', 'ones', 'goods', 't
oast', '9.98', 'perogies', 'way', 'minutes', 'food', 'size', 'choice',
'reviews', 'pass', 'principle', 'idea', 'things', 'pizza', 'one', 'serv
ice', 'fact', 'woman', 'NOTHING', 'opinions', 'selection', 'try', 'muss
els', 'atmosphere', 'pub', 'tasting', 'time', 'Crispy', 'reviews', 'rev
iews', 'restaurants', 'service', 'reviews', 'lot', 'lot', 'Everyone',
'directions', 'service', 'order', 'fry', 'part', 'food', 'plenty', 'kin
d', 'waiter', 'change', 'experience', 'time', 'burger', 'portions', 'wo
rk', 'drinks', 'end', 'thing', 'sign', 'time', 'pizza', 'fact', 'time',
'establishment', 'regulars', 'ribs', 'food', 'spot', 'girl', 'reviews',
'salsa', 'menudo', 'order', 'Anyone', 'raspberry', 'custard', 'price',
'home', 'non-fast-food', 'care', 'lot', 'food', 'staff', 'deal', 'locat
ion', 'place', 'people', 'Great', 'view', 'drinks', 'Very', 'idea', 'ti
me', 'salad', 'reason', 'glow', 'friend', 'veggie', 'experience', 'plac
e', 'spot', 'location', 'Sambalatte', 'service', 'service', 'music', 'f
lavor', 'traffic', 'flavor', 'portion', 'flavors', 'dive', 'flavor', 's
eat', 'service', 'dinner', 'salsa', 'minutes', 'portion', 'goodness',
'shape', 'butter', 'view', 'crowd', 'flavor', 'customers', 'food', 'typ
es', 'guest', 'food', 'spot', 'burger', 'drink', 'location', 'steakhous
e', 'time', 'bathroom', 'staff', 'oil', 'time', 'flavor', 'good', 'grou
p', 'meal', 'number', 'views', 'dining', 'chorizo', 'part', 'conventio
n', 'wine', 'assortment', 'views', 'portion', 'place', 'part', 'flavo
r', 'burgers', 'amount', 'off-strip', 'drinks', 'fries', 'place', 'opti
on', 'top', 'impressions', 'option', 'move', 'ole', 'goods', 'goodbye',
'wine', 'nothing', 'goods', 'day', 'deal', 'meat', 'couple', 'measure',
'spinach', 'size', 'eats', 'time', 'thing', 'Szechuan', 'job', 'Eggs',
'meat', 'taste', 'things', 'one', 'credit', 'choice', 'goodness', 'ban
g', 'bathroom', 'visit', 'alfredo', 'dive', 'filler', 'bar', 'maple',
'margaritas', 'family', 'decision', 'goodness', 'mediocrity', 'thing',
'Just', 'experience', 'commercials', 'taste', 'goodwill', 'goodness',
'kick', 'people', 'quality', 'price', 'nachos', 'beer', 'experiences',
'place', 'menu', 'food', 'flavour', 'burger', 'food', 'service', 'brisk
et', 'days', 'price', 'one', 'ranch', 'business', 'theres', 'visit', 's
omeone', 'plenty', 'McDonald', 'servers', 'goodies', 'Bleu', 'bun', 'pl
ace', 'festivals', 'selection', '15', 'goodness']
```

This seems like a much more interesting list!

Conclusions

In this case, we focused on the basic components of an NLP pipeline, virtually all of which are frequently used *before* building a model for the business question of interest. We saw that every part of the pipeline was highly customizable, and discussed how parameters might vary depending on the specific application in mind.

In addition to constructing a basic pipeline, we tried to give initial answers to a business question: "Which factors are most important for bad reviews?" In doing so, we illustrated the importance of parameter choice by applying a simple model with a very popular but non-obvious choice of parameters, known as TF-IDF. The answers we obtained with this out-of-the-box analysis were not perfect (e.g. "maps maps maps" was not a good answer), but they did seem to give some genuinely useful information. For example, 3 of the 5 most important phrases for bad reviews were "20 minutes", "10 minutes", and "15 minutes" – strong evidence that long service times were a major driver of bad reviews.

Takeaways

Text pre-processing is more complex than other forms of pre-processing you might be familiar with, as good pre-processing may rely on an enormous number of rules extracted from large corpora of English (Spanish!) text. You shouldn't try to recreate this work by yourself; instead, take advantage of large and powerful libraries such as `nltk` which have built-in corpora when possible, and use regular expressions when necessary to extend or tweak them.

Pre-processing is an extremely important and nontrivial part of NLP, and will likely take the bulk of the work for most NLP projects. Most popular parts of the pipeline come with many parameters. Some of the most popular parameters, such as in TF-IDF, are quite clever and non-obvious. They can give surprisingly useful summaries of entire corpora without much adjustment.

In [ ]: