Text Mining the Hitchhiker Trilogy

Greg Johnson

Contents

Reading and Tidying Our Data

Let's read in Doug's first book in his "trilogy," The Hitchhiker's Guide to the Galaxy. We will read in the text file as a character vector in which each element is a line from the book.

```
# setwd('~/Documents/Analytics/R/DouglasAdams')
for (book in list.files("data")) {
    temp <- book %>% paste("data", ., sep = "/") %>% readLines() %>%
        tibble(text = .) %>% mutate(bookline = row_number(),
        book = substr(book, 9, nchar(book) - 4))

    assign(book %>% substr(1, 5), temp)
}

c(" ", head(Book1$text, 20), " ") %>% kable
```

Douglas Adams

The Hitch Hiker's Guide to the Galaxy

Book 1

for Jonny Brock and Clare Gorst
and all other Arlingtonians
for tea, sympathy, and a sofa

Far out in the uncharted backwaters of the unfashionable end of the
western spiral arm of the Galaxy lies a small unregarded yellow sun.
Orbiting this at a distance of roughly ninety-two million miles is an
utterly insignificant little blue green planet whose apedescended life forms are so amazingly primitive that they still think digital watches are

---

Our data are in! But they're not tidy in the sense that we want one token per line. For a first look at our data, we will look at words. The *unnest_tokens* function will tidy our data (by default) into word tokens. It will also strip punctuation and convert our words to lowercase.

```
Doug <- bind_rows(Book1, Book2, Book3, Book4, Book5)
Doug %<>% mutate(chapter = cumsum(str_detect(text, regex("Chapter \\d+$",
    ignore_case = TRUE))))

Dougwords <- Doug %>% unnest_tokens(word, text)
```

We will remove stop words from our new tidy dataset with the *anti_join* function.

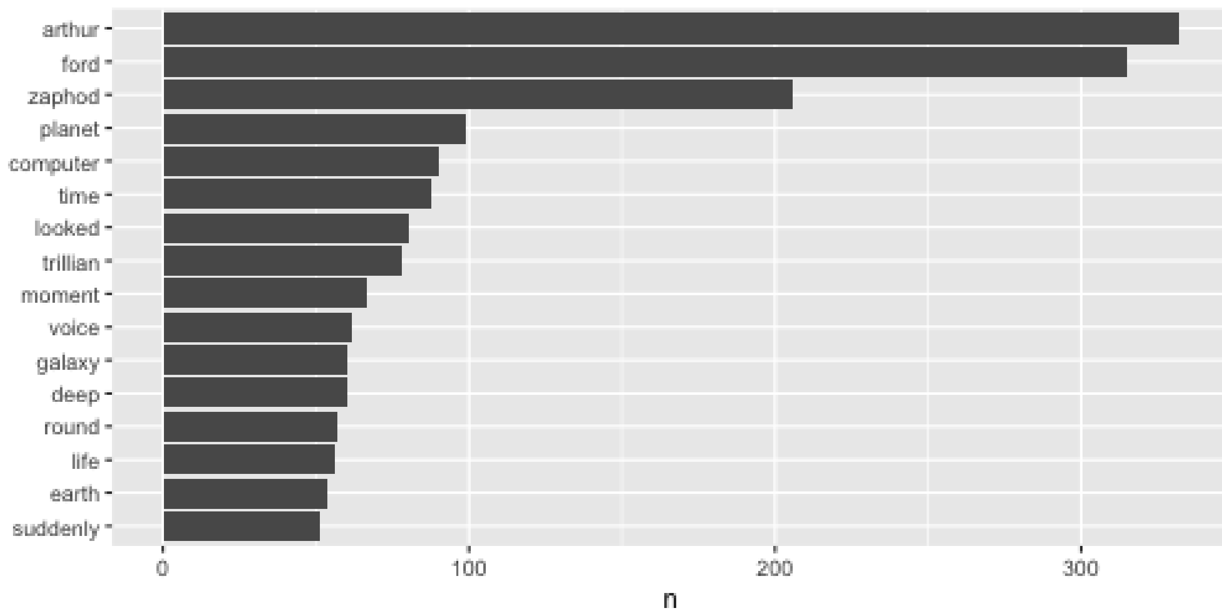
```
Dougwords %<>% anti_join(stop_words, by = "word")
```

Term Frequency

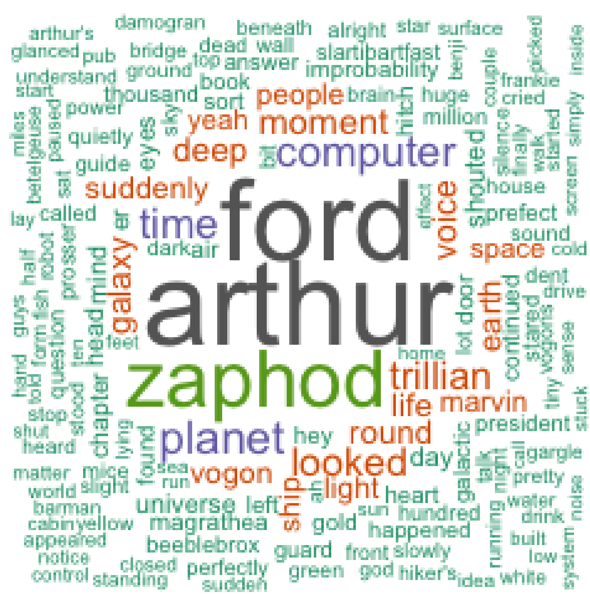
A Simple Word Count

Now we can start processing our tidy data. For now let's look at the first book, **Hitchiker's Guide to the Galaxy**. We'll start with a simple word frequency count aka **term frequency** using *dplyr* and a visualization of frequencies using a word cloud.

```
Book1count <- Dougwords %>% filter(book == "The Hitchhiker's Guide to the Galaxy") %>%
    count(word, sort = TRUE)
Book1count %>% filter(n > 50) %>% mutate(word = reorder(word,
    n)) %>% ggplot(aes(word, n)) + geom_col() + xlab(NULL) +
    coord_flip()
```




```
wordcloud(words = Book1count$word, freq = Book1count$n, max.words = 200,
    random.order = FALSE, rot.per = 0.35, colors = brewer.pal(8,
        "Dark2"))
```



Between-Book Comparisons

What if we wanted to look at word frequency comparisons between Doug's books? One method is to consider the proportions of words in the books as random variables. Each word is a statistical unit that is observed twice: proportion in book 1 and proportion in book 2. If the two books were very similar to each other, we would expect that for most words, the proportion in one book would be about the same as that in the other book; if two books were very different, many words would be common in one book but not in the other - in other words, there would be very little overlap in the set of words used.

```r
Dougfreq <- Dougwords %>% count(book, word) %>% group_by(book) %>%
    mutate(proportion = n/sum(n)) %>% select(-n) %>% spread(book,
    proportion) %>% gather(book, proportion, `The Restaurant at the End of the Universe`,
    `So Long, and Thanks for All the Fish`, `Life, the Universe and Everything`,
    `Mostly Harmless`)


# expect a warning about rows with missing values being
# removed
ggplot(Dougfreq, aes(x = proportion, y = `The Hitchhiker's Guide to the Galaxy`,
    color = abs(`The Hitchhiker's Guide to the Galaxy` - proportion))) +
    geom_abline(color = "gray40", lty = 2) + geom_jitter(alpha = 0.1,
    size = 2.5, width = 0.3, height = 0.3) + geom_text(aes(label = word),
    check_overlap = TRUE, vjust = 1.5) + scale_x_log10(labels = percent_format()) +
    scale_y_log10(labels = percent_format()) + scale_color_gradient(limits = c(0,
    0.001), low = "darkslategray4", high = "gray75") + facet_wrap(~book,
    nrow = 2, ncol = 2) + theme(legend.position = "none") + labs(y = "Hitchhiker's Guide to the Galaxy"
    x = NULL)
```


Life, the Universe and Everything
Mostly Harmless
So Long, and Thanks for All the Fish
The Restaurant at the End of the Universe
Hitchhiker's Guide to the Galaxy
1.00%
0.10%
0.01%
0.01%
0.10%
1.00%

Cool! Some observations:

1. We can tell who the main characters were for each book. Arthur is present for each book, as noted by his high frequency of mention for each book (and as a result, his presence on the upper right of the diagonal line).
2.

We can investigate claim 2 with some actual statistical evidence.

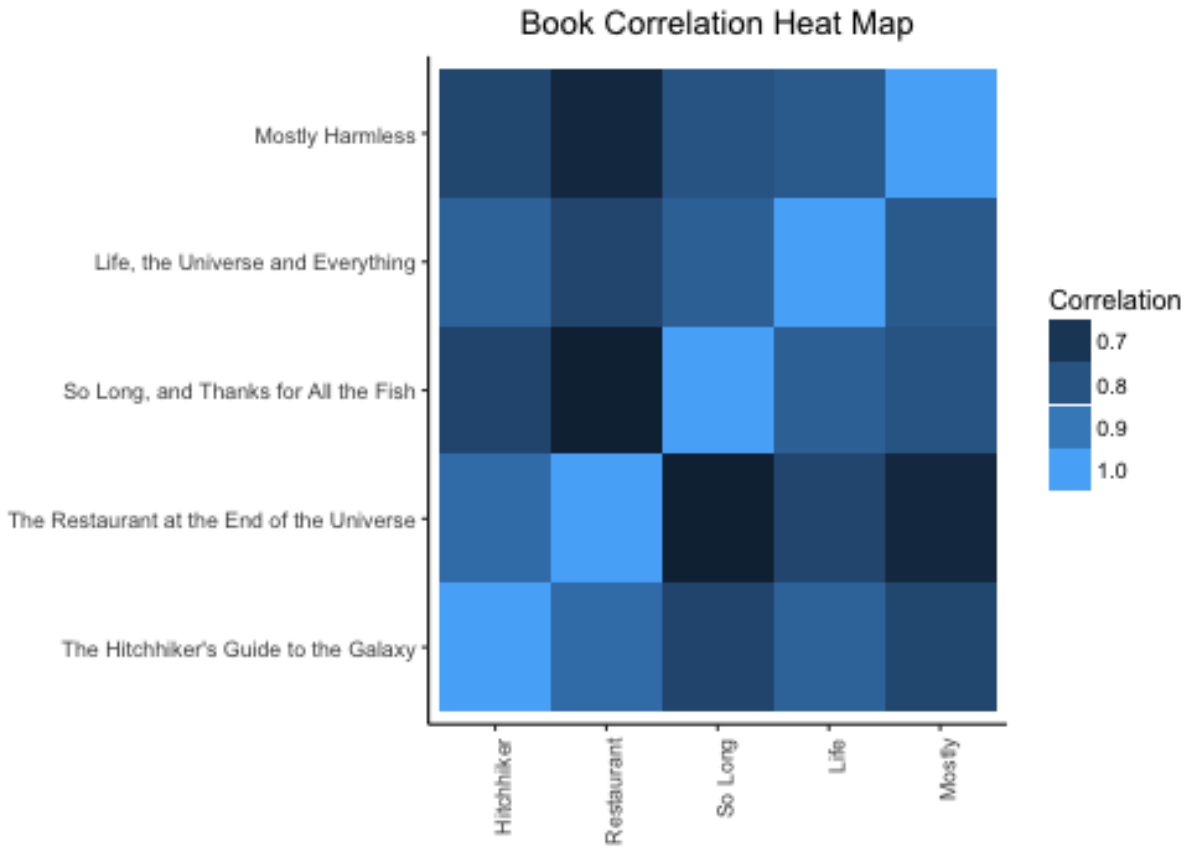
```
Dougfreq <- Dougwords %>% count(book, word) %>% group_by(book) %>%
    mutate(proportion = n/sum(n)) %>% select(-n) %>% spread(book,
    proportion) %>% gather(book, proportion, `The Hitchhiker's Guide to the Galaxy`,
    `The Restaurant at the End of the Universe`, `So Long, and Thanks for All the Fish`,
    `Life, the Universe and Everything`, `Mostly Harmless`)

getCor <- function(book1, book2) {
    cor.test(Dougfreq %>% filter(book == book1) %>% select(proportion) %>%
        pull(), Dougfreq %>% filter(book == book2) %>% select(proportion) %>%
        pull())[["estimate"]]
}


booknames <- unique(Dougfreq$book)
R <- matrix(0, 5, 5, dimnames = list(c("Hitchhiker", "Restaurant",
    "So Long", "Life", "Mostly"), booknames))

for (i in 2:5) {
    for (j in 1:(i - 1)) {
        R[i, j] <- getCor(booknames[i], booknames[j])
    }
}
R <- R + t(R)
diag(R) <- 1

meltR <- melt(R)
ggplot(data = meltR, aes(x = Var1, y = Var2, fill = value)) +
    geom_tile() + scale_colour_gradient(low = "red", high = "white") +
    labs(title = "Book Correlation Heat Map", x = "", y = "") +
    guides(fill = guide_legend(title = "Correlation")) + theme_bw() +
    theme(panel.border = element_blank(), panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(), axis.line = element_line(colour = "black"),
        plot.title = element_text(hjust = 0.5), axis.text.x = element_text(angle = 90,
            hjust = 1))
```



Our correlations are based on scatterplots of proportions of words between books - they appear to be appropriate measures of association because of linearity (although one might argue a bit inappropriate because proportions of words aren't necessarily independent of each other).

Sentiment Analysis/Opinion Mining

Let's go a step further with our text analysis and look at the emotional context. We will take a basic but very popular approach of analyzing the **sentiment content** of a book as the sum of the sentiment content of the individual words. Now how do we assess sentiment content at all? We need a **sentiment lexicon** - a dictionary that maps certain tokens to a certain sentiment e.g. "subversion" is mapped to "fear" in the nrc sentiment lexicon. Included in the *tidytext* package are four *unigram sentiment lexicons* i.e. lexicons based on single words:

1. **AFINN-111** - 2477 words mapped to a range of -5 to +5, negative sentiment to positive.
2. **bing** - 6800 words with a binary mapping to positive or negative.
3. **loughran** - 4149 words also classified as positive or negative.
4. **nrc** - classifies 6468 words into (possibly) multiple sentiments like anger, sadness, etc.

It's important to keep in mind a couple of things: 1. These lexicons were constructed and validated on more modern texts - this may limit the generalizability of these tools to texts that fall outside of modern literature. It is always a good idea to check for the existence of domain-specific sentiment lexicons that may be more appropriate for your specific text. 2. Negation is not considered e.g. "not abandoned" will be mapped to negative because "not" will be ignored. 3. The size of text that we use to sum up word sentiment will have an effect - a sentence or a paragraph will generally have the same sentiment throughout but over several paragraphs there may be fluctuations so that when an average sentiment is computed, everything cancels out.

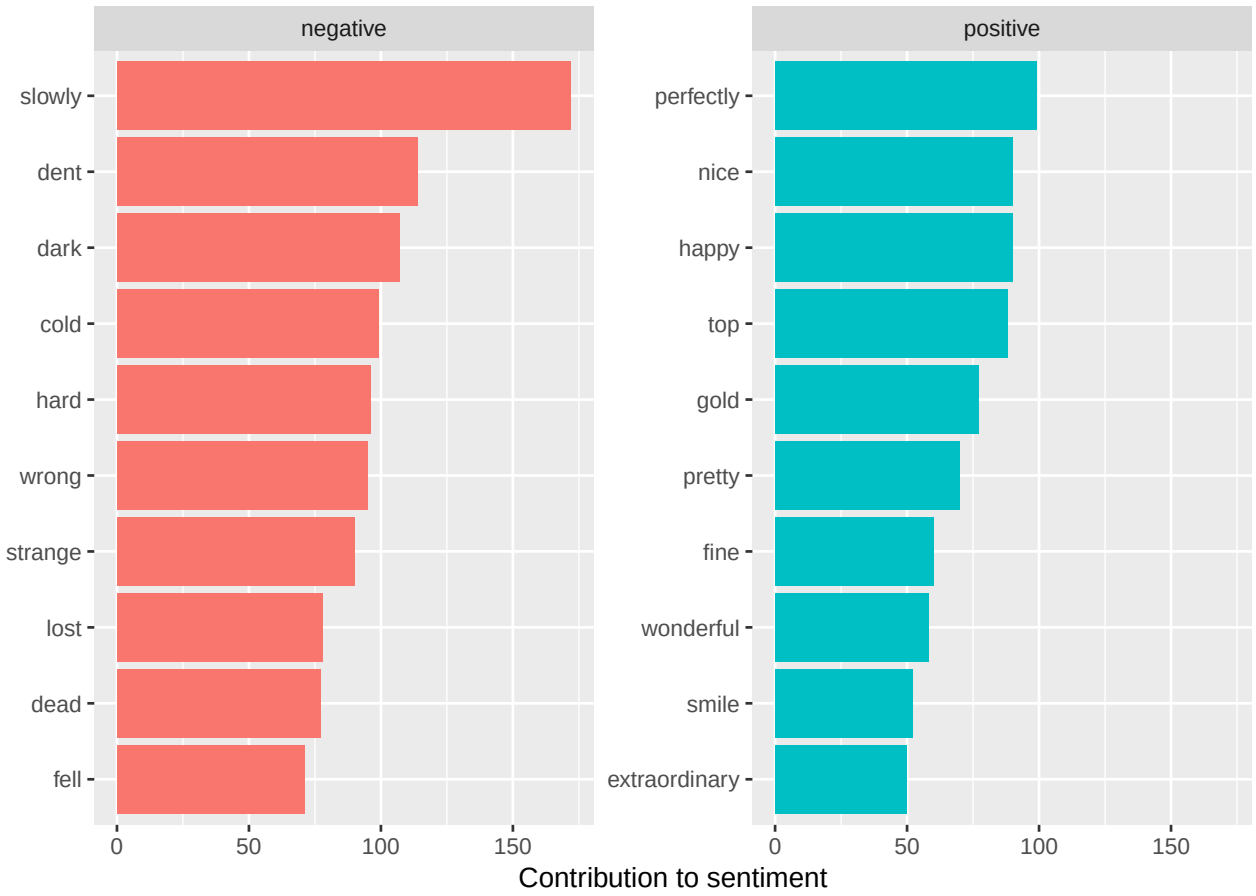
Frequent Sentiment Words

To perform sentiment analysis, we need to map our Douglas Adam words to sentiments. We can do that with the *inner_join* function. Let's take a look at the most frequent words in our trilogy that are associated with the "anger" sentiment.

```
nrcAnger <- sentiments %>% filter(lexicon == "nrc", sentiment ==
    "anger")
nrcAngerCount <- Dougwords %>% inner_join(nrcAnger) %>% count(word,
    sort = TRUE)
wordcloud(words = nrcAngerCount$word, freq = nrcAngerCount$n,
    max.words = 50, random.order = FALSE, rot.per = 0.35, colors = brewer.pal(8,
        "Dark2"))
```

How about if we look at positive vs. negative sentiments. We can create a word cloud that has a "sentiment axis" where words are placed according to frequency *and* their sentiment valence.

```
Dougwords %>% inner_join(sentiments %>% filter(lexicon == "bing")) %>%
    count(word, sentiment, sort = TRUE) %>% ungroup() %>% group_by(sentiment) %>%
    top_n(10) %>% ungroup() %>% mutate(word = reorder(word, n)) %>%
    ggplot(aes(word, n, fill = sentiment)) + geom_col(show.legend = FALSE) +
    facet_wrap(~sentiment, scales = "free_y") + labs(y = "Contribution to sentiment",
    x = NULL) + coord_flip()
```



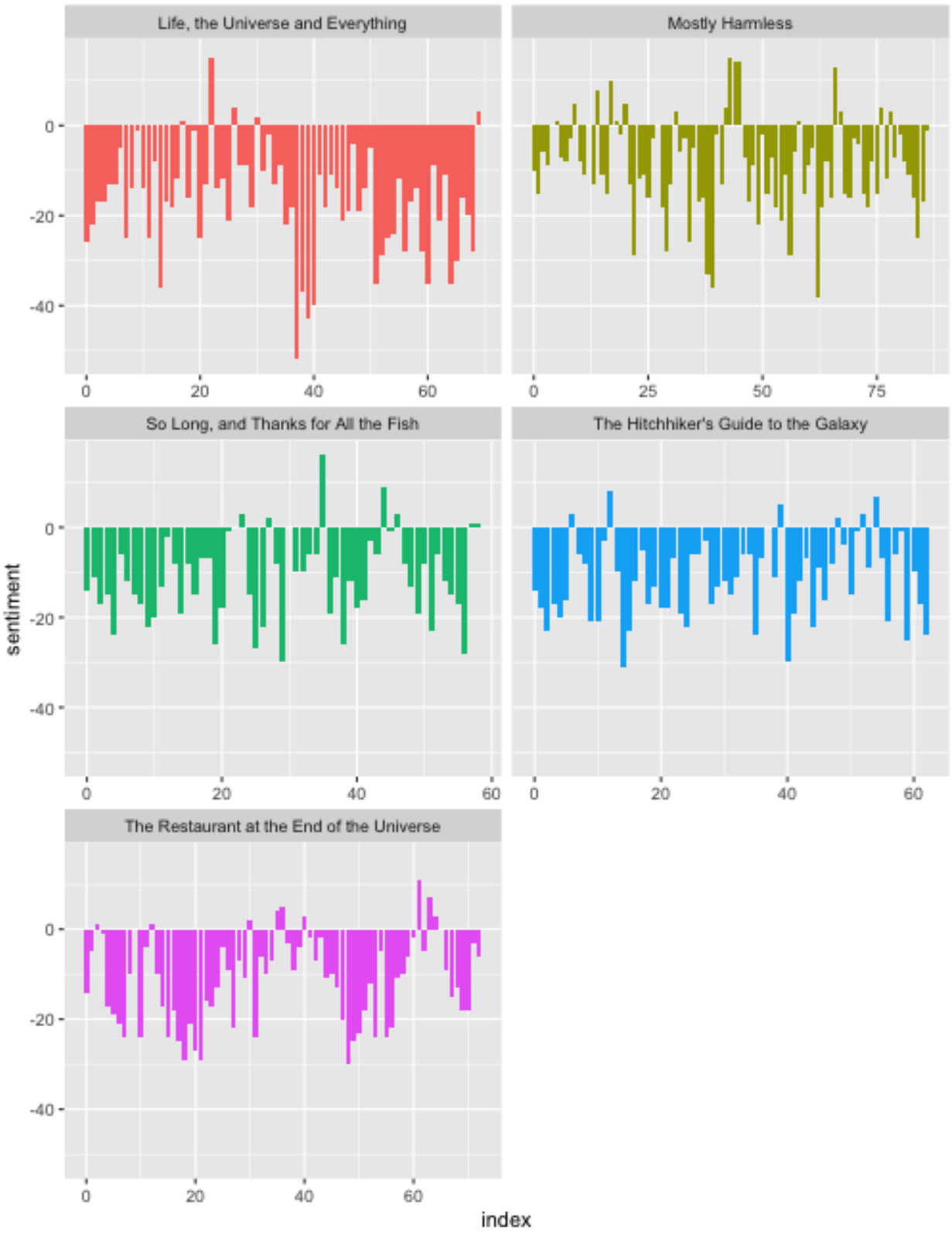

```
Dougwords %>% inner_join(sentiments %>% filter(lexicon == "bing")) %>%
    count(word, sentiment, sort = TRUE) %>% acast(word ~ sentiment,
    value.var = "n", fill = 0) %>% comparison.cloud(colors = c("#F8766D",
    "#00BFC4"), max.words = 100)
```

Sentiment over Time

Another aspect of sentiment analysis is exploring changes in sentiment over some index that keeps track of where we are in the books. We will use the **bing** lexicon to get positive and negative sentiments, then get the average sentiment for every block of 80 lines. Finally we can plot the average sentiment of the trilogy from beginning to end.

```
Dougsentiment <- Dougwords %>% inner_join(sentiments %>% filter(lexicon ==
    "bing"), by = "word") %>% count(book, index = bookline%/%80,
    sentiment) %>% spread(sentiment, n, fill = 0) %>% mutate(sentiment = positive -
    negative)

ggplot(Dougsentiment, aes(index, sentiment, fill = book)) + geom_col(show.legend = FALSE) +
    facet_wrap(~book, ncol = 2, scales = "free_x")
```



Wow, this trilogy is overwhelmingly negative according to our sentiment analysis and you know what? I have to agree. And it's not just due to Marvin either. Adams' writing style is rife with dry humour, satire and existential crises - not exactly a literary cocktail for good feelings. So this makes sense. What was

most surprising for me is that I expected **So Long, and Thanks for All the Fish** to have more positive sentiment seeing as it has this whole tangential, out-of-place love-story with Arthur and Fenchurch (despite its eventual, depressing denouement). In an absolute sentiment sense we have to keep our limitations of our sentiment analysis in mind; however in a relative between-book sense the limitations should apply more-or-less uniformly so it's still surprising to me that **So Long** is not even close to the least-negative book in the series.

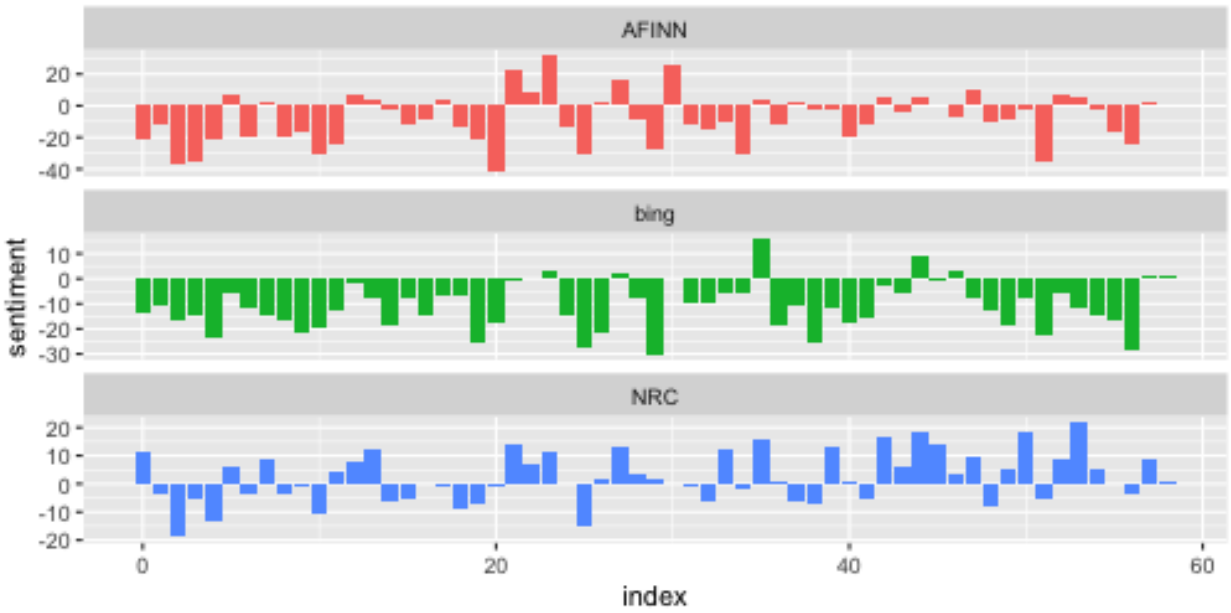
Let's follow up with a between-lexicon look at **So Long**. Perhaps it is just the **bing** lexicon we used. Let's compare to our other lexicons.

```
SoLong <- Dougwords %>% filter(book == "So Long, and Thanks for All the Fish")

afinn <- SoLong %>% inner_join(sentiments %>% filter(lexicon ==
    "AFINN")) %>% group_by(index = bookline%/%80) %>% summarise(sentiment = sum(score)) %>%
    mutate(method = "AFINN")

bingNRC <- bind_rows(SoLong %>% inner_join(sentiments %>% filter(lexicon ==
    "bing")) %>% mutate(method = "bing"), SoLong %>% inner_join(sentiments %>%
    filter(lexicon == "nrc") %>% filter(sentiment %in% c("positive",
    "negative"))) %>% mutate(method = "NRC")) %>% count(method,
    index = bookline%/%80, sentiment) %>% spread(sentiment, n,
    fill = 0) %>% mutate(sentiment = positive - negative)

bind_rows(afinn, bingNRC) %>% ggplot(aes(index, sentiment, fill = method)) +
    geom_col(show.legend = FALSE) + facet_wrap(~method, ncol = 1,
    scales = "free_y")
```



It looks like it was the bing lexicon that (arguably) introduced a downward bias in sentiment! The reason is actually simple, bing has far more negative words than the other lexicons:

```
nrcvalence <- sentiments %>% filter(lexicon == "nrc", sentiment %in%
    c("positive", "negative")) %>% count(sentiment)

afinnvalence <- sentiments %>% filter(lexicon == "AFINN") %>%
    count(score > 0)
```

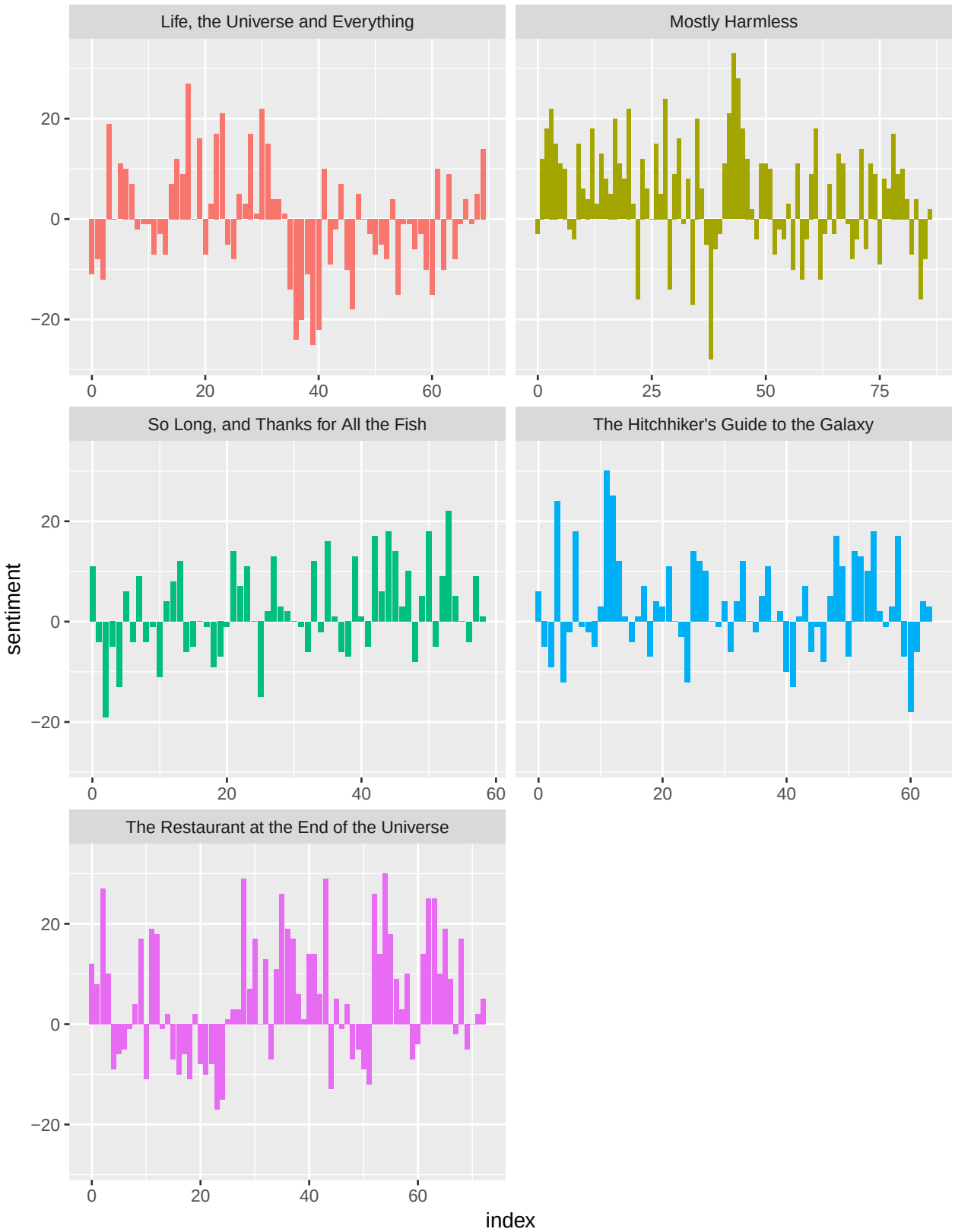
```
bingvalence <- sentiments %>% filter(lexicon == "bing") %>% count(sentiment)
to_print <- cbind(nrcvalence$n, afinnvalence$n, bingvalence$n)
dimnames(to_print) <- list(c("Count of Negative Words", "Count of Positive Words"),
    c("nrc", "AFINN", "bing"))
kable(to_print)
```

	nrc	AFINN	bing
Count of Negative Words	3324	1598	4782
Count of Positive Words	2312	878	2006

Let's redo our trilogy sentiment analysis with the nrc lexicon.

```
Dougsentiment2 <- Dougwords %>% inner_join(sentiments %>% filter(lexicon ==
    "nrc", sentiment %in% c("positive", "negative")), by = "word") %>%
    count(book, index = bookline%/%80, sentiment) %>% spread(sentiment,
    n, fill = 0) %>% mutate(sentiment = positive - negative)

ggplot(Dougsentiment2, aes(index, sentiment, fill = book)) +
    geom_col(show.legend = FALSE) + facet_wrap(~book, ncol = 2,
    scales = "free_x")
```



The books look a lot more positive now that we've used a different lexicon! But their relative positivity looks about the same. **Mostly Harmless** and **Restaurant** are still the most positive and **So Long** is still one of the least positive. Oh well - looks like cherry-picking lexicons won't save my theory that **So Long** has the

most positive sentiment. But this does highlight an important point about lexicons - the absolute scale of positive and negative sentiment may change between lexicons but the relative rankings between texts stays the same (for the most part).

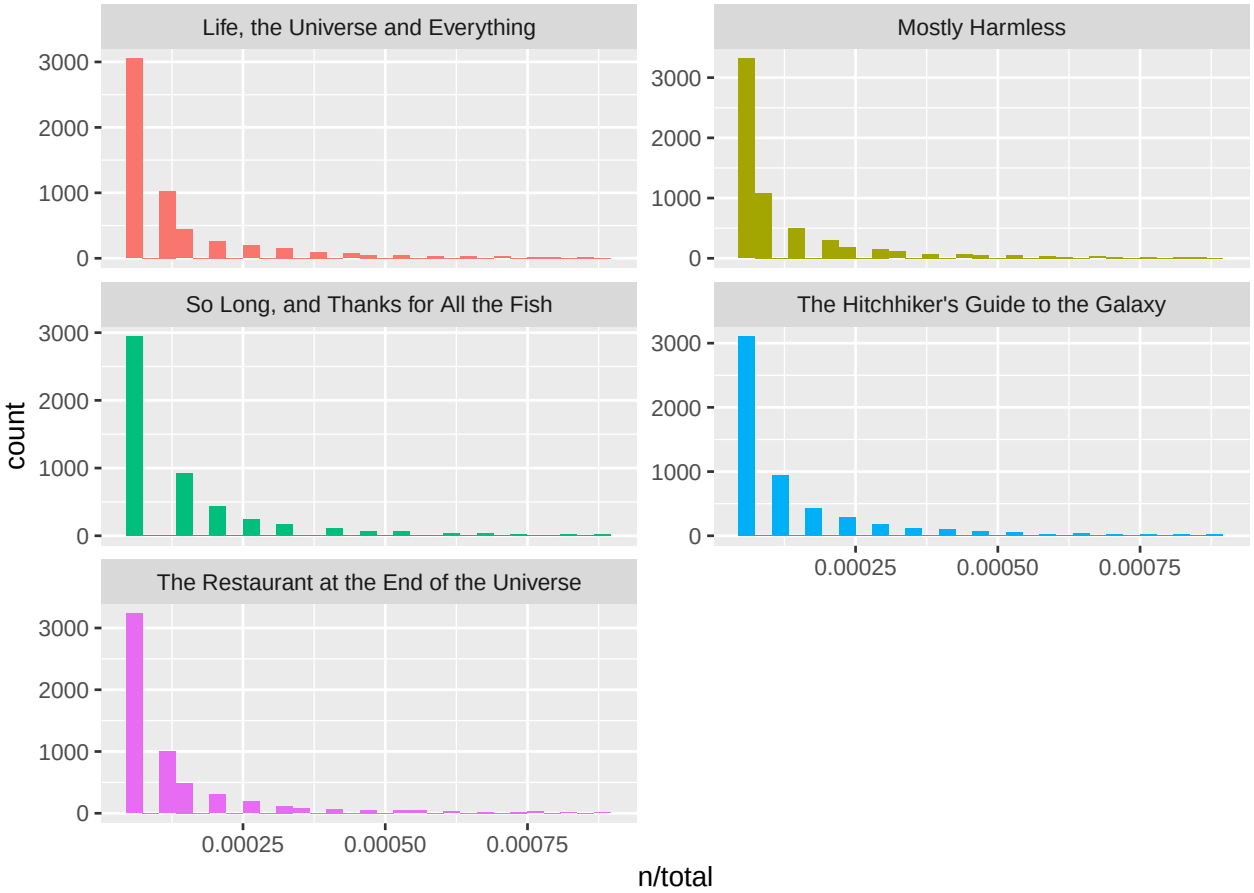
Revisiting Word Frequency

Zipf's Law

Distributions of term frequencies consistently take a certain shape - extremely long right tails.

```
# get counts for each word for each book
book_words <- Dougwords %>% count(book, word, sort = TRUE) %>%
    ungroup()
# total number of words per book
total_words <- book_words %>% group_by(book) %>% summarize(total = sum(n))
# append to tf per book dataset the total number of words per
# book
book_words <- left_join(book_words, total_words)

# plot distribution of term frequencies
ggplot(book_words, aes(n/total, fill = book)) + geom_histogram(show.legend = FALSE) +
    xlim(NA, 9e-04) + facet_wrap(~book, ncol = 2, scales = "free_y")
```



How did we form these distributions? First words are ranked according to term frequency, then their term

frequencies are plotted. We can see that as rank decreases, so does term frequency *but the decrease doesn't look linear*. It looks like an exponential decrease.

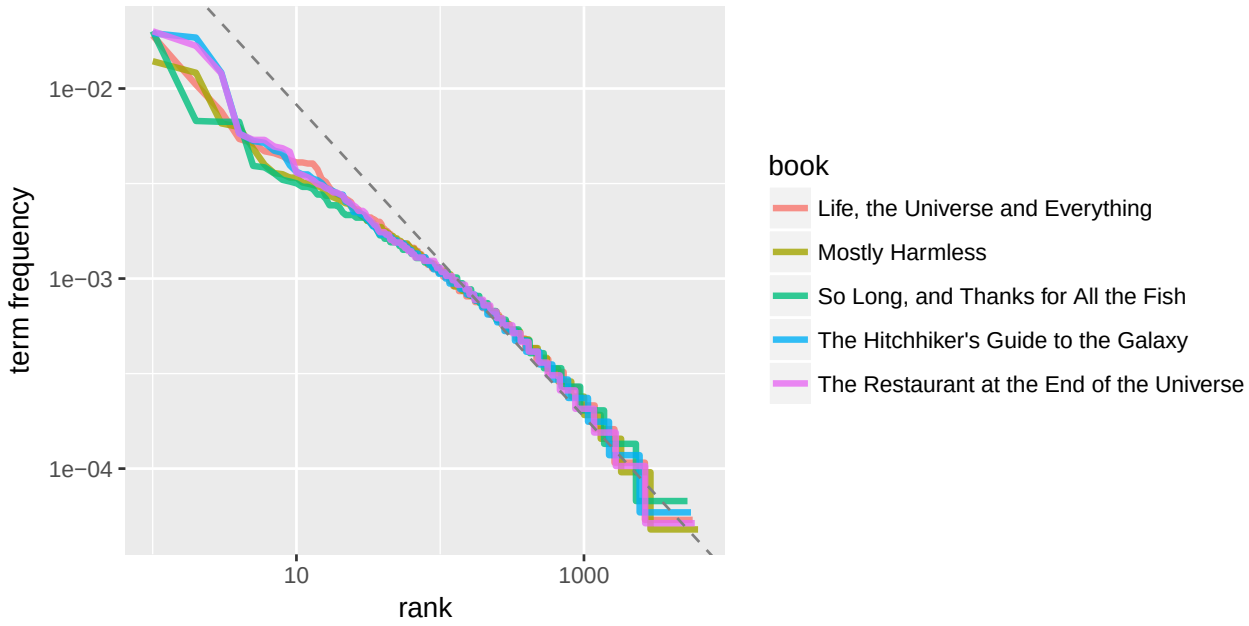
Zipf's law attemps to catch this term frequency phenomenon It states that a word's term frequency is inversely proportional to its rank. In other words, the logarithms of term frequency and of rank should have a negative linear relationship. As I mentioned above, this may be an oversimplication.

```
freq_by_rank <- book_words %>% group_by(book) %>% mutate(rank = row_number(),
    `term frequency` = n/total)

p <- freq_by_rank %>% ggplot(aes(rank, `term frequency`, color = book)) +
    geom_line(size = 1.2, alpha = 0.8) + scale_x_log10() + scale_y_log10()
p
```

Looks pretty linear! We can fit an OLS line to see just how well Zipf's Law applies.

```
OLSfit <- lm(log10(`term frequency`) ~ log10(rank), data = freq_by_rank) %>%
    coef
p + geom_abline(intercept = OLSfit[1], slope = OLSfit[2], color = "gray50",
    linetype = 2)
```



Ill fitting at the higher ranks but the rest of the ranks seem to fit Zipf's pretty well. This phenomenon is pretty common.

tf-idf Statistic

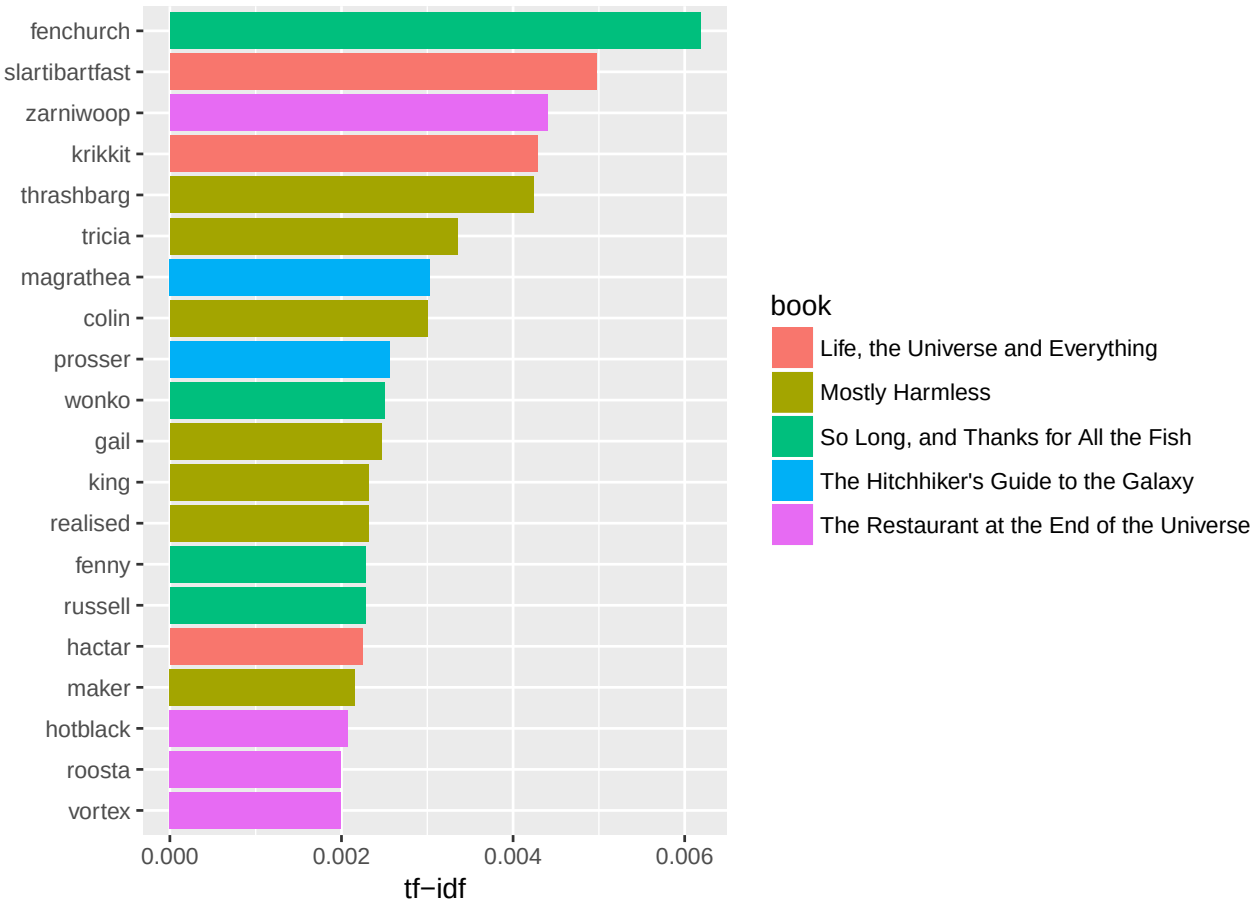
As we saw earlier, one method of analyzing a text's content is to look at raw word frequency. However we may want to adjust word frequency by weighting words based on how much they're used in a collection of texts. For example, we may want to weight *less* those words that are common between several texts and weight *more* those that are unique to a text. One statistic that applies this logic is *tf-idf* which stands for the term frequency - inverse document frequency. For some word w,

$$tf - idf(w) = tf(w) \times idf(w) = n_w \times \ln\left(\frac{n_{\text{docs}}}{n_{\text{docs with term}}}\right)$$

The idea is that for words that occur in many documents (of consideration), $\frac{n_{\text{docs}}}{n_{\text{docs with term}}}$ is close to 1 and its logarithm is close to 0, thus the raw term frequency is weighted towards zero. When the word occurs in few documents, $\frac{n_{\text{docs}}}{n_{\text{docs with term}}}$ is much greater than zero and so is its logarithm, giving a larger weight to the term frequency.

Using the *tf-idf* statistics, let's investigate which words are characteristic of which documents.

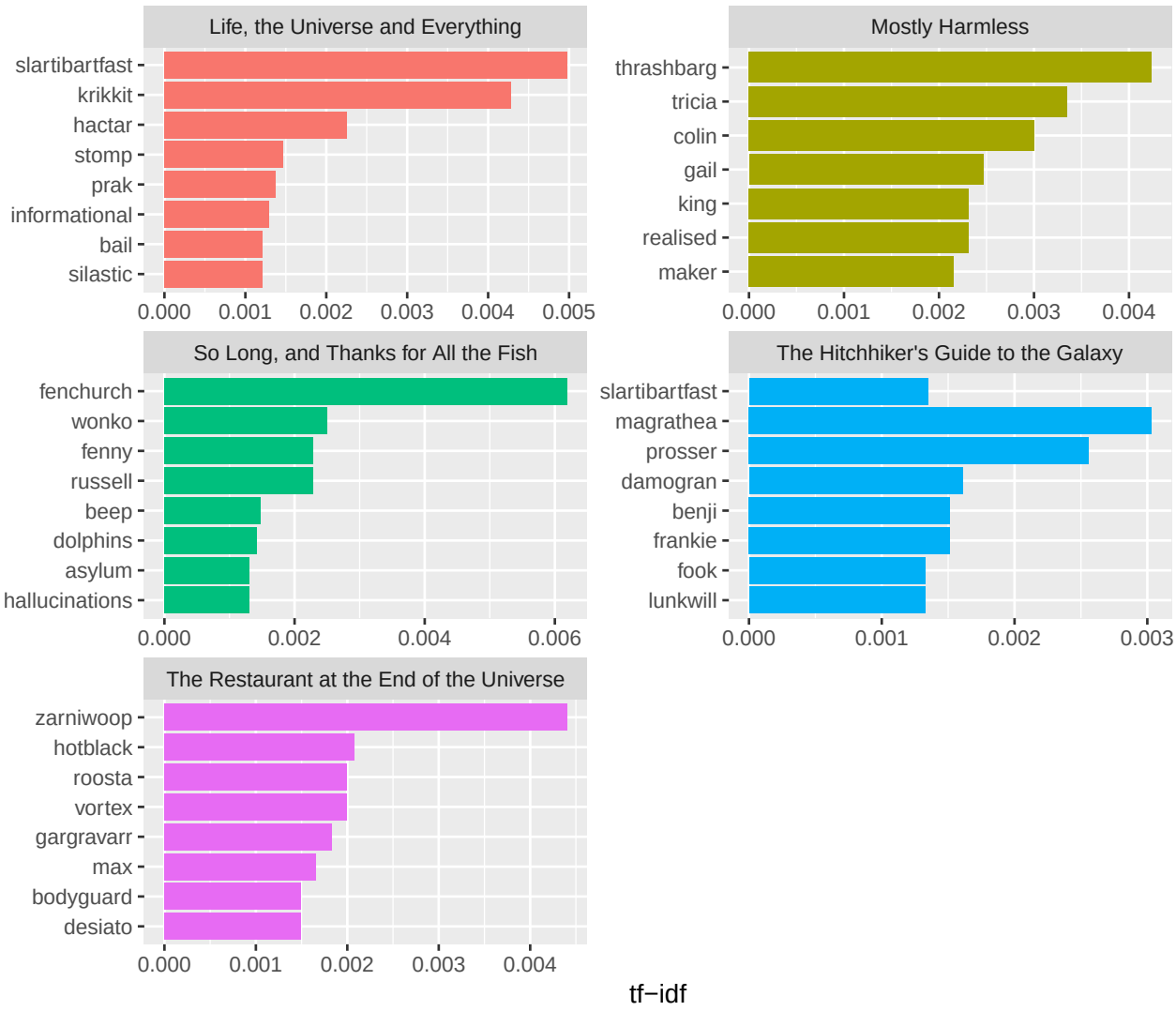
```
book_words %<>% bind_tf_idf(word, book, n)
to_kable <- book_words %>% select(-total) %>% arrange(desc(tf_idf)) %>%
    top_n(10, tf_idf)
to_kable %<>% mutate(tf = round(tf, 4), idf = round(idf, 3),
    tf_idf = round(tf_idf, 4))
to_kable %>% kable()
plot_doug <- book_words %>% arrange(desc(tf_idf)) %>% mutate(word = factor(word,
    levels = word %>% unique %>% rev))
plot_doug %>% top_n(20) %>% ggplot(aes(word, tf_idf, fill = book)) +
    geom_col() + labs(x = NULL, y = "tf-idf") + coord_flip()
```



book	word	n	tf	idf	tf_idf
So Long, and Thanks for All the Fish	fenchurch	100	0.0067	0.916	0.0062
Life, the Universe and Everything	slartibartfast	101	0.0054	0.916	0.0050
The Restaurant at the End of the Universe	zarniwoop	53	0.0027	1.609	0.0044
Life, the Universe and Everything	krikkit	87	0.0047	0.916	0.0043
Mostly Harmless	thrashbarg	55	0.0026	1.609	0.0042
Mostly Harmless	tricia	137	0.0066	0.511	0.0033
The Hitchhiker's Guide to the Galaxy	magrathea	32	0.0019	1.609	0.0030
Mostly Harmless	colin	39	0.0019	1.609	0.0030
The Hitchhiker's Guide to the Galaxy	prosser	27	0.0016	1.609	0.0026
So Long, and Thanks for All the Fish	wonko	23	0.0016	1.609	0.0025

It looks like the terms with highest *tf-idf* are mostly names of characters unique to one book. The highest ranked term? Poor, poor Fenchurch. Second? Our buddy Slartibartfast who features largely in **The Hitchhiker's Guide** and has a small cameo in **Life**. Let's reorganize this chart so that we can see the high *tf-idf* terms grouped by book.

```
plot_doug %>% group_by(book) %>% top_n(7) %>% ungroup() %>% ggplot(aes(word,
    tf_idf, fill = book)) + geom_col(show.legend = FALSE) + labs(x = NULL,
    y = "tf-idf") + facet_wrap(~book, ncol = 2, scales = "free") +
    coord_flip()
```



Between-Word Relationships

Bigram Tokenization

Text analysis of single words is a simple and easy yet powerful approach. But we have to admit that the real meat-and-potatoes is in higher units of analysis. The next natural self-contained unit that comes to mind is the sentence but what about groups of words within sentences? We can tokenize n-grams, n-size groups of consecutive words.

```
Dougbigrams <- Doug %>% unnest_tokens(bigram, text, token = "ngrams",
    n = 2)
```

Bigram Frequency

Following our single word (unigram) approach, we can again look at term frequency:

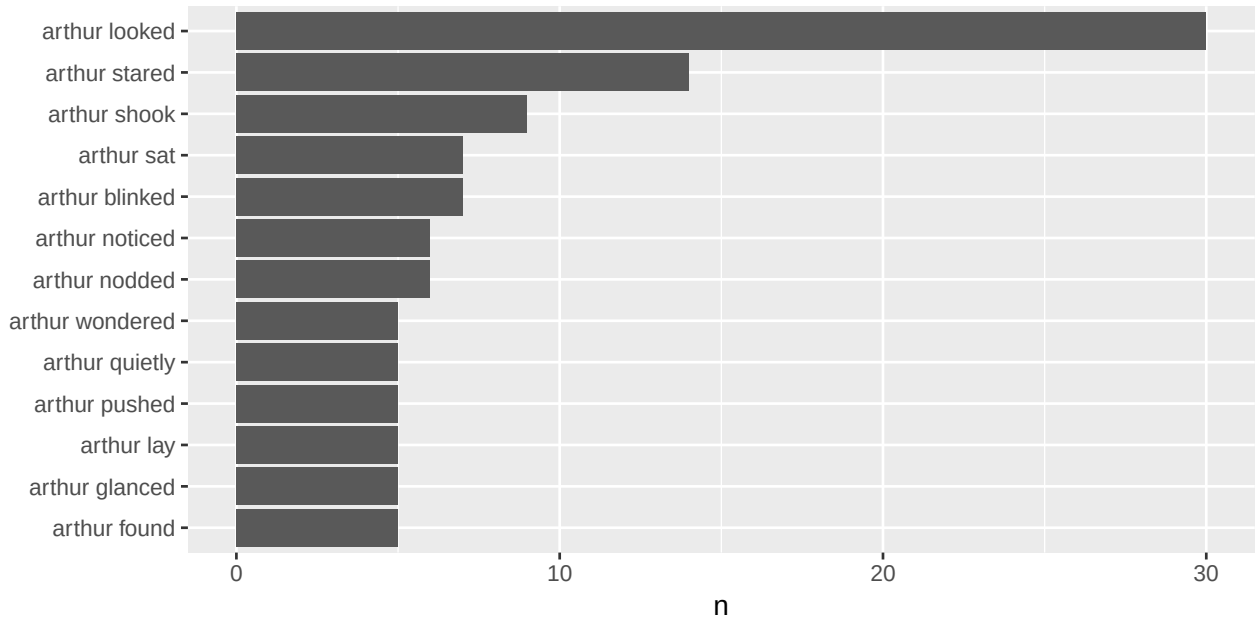
```
DougbigramFiltered <- Dougbigrams %>% separate(bigram, c("word1",
    "word2"), sep = " ") %>% filter(!word1 %in% stop_words$word) %>%
    filter(!word2 %in% stop_words$word)


DougbigramCount <- DougbigramFiltered %>% unite(bigram, word1,
    word2, sep = " ") %>% count(bigram, sort = TRUE)

wordcloud(words = DougbigramCount$bigram, freq = DougbigramCount$n,
    max.words = 50, random.order = FALSE, rot.per = 0.35, colors = brewer.pal(8,
        "Dark2"))
```

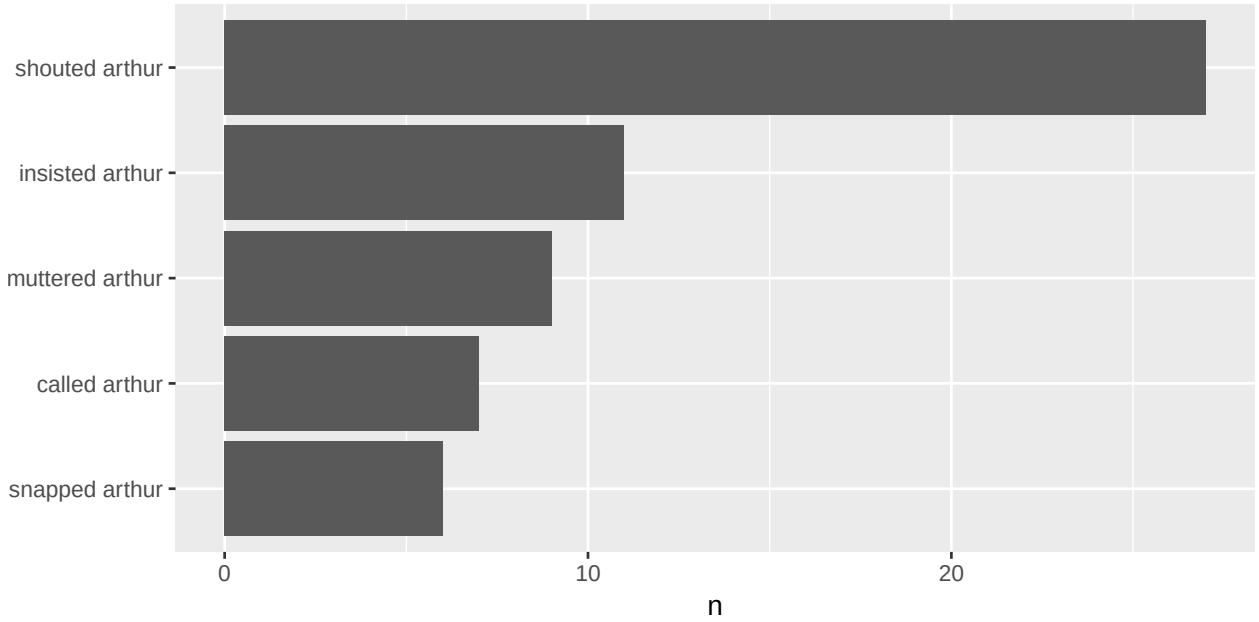
What if we were interested in Arthur's actions throughout the book? We could take a look at all the bigrams in which Arthur is the first word (and Dent isn't the second word).

```
DougbigramFiltered %>% filter(word1 == "arthur" & !word2 %in%
    c("dent", "dent's")) %>% unite(bigram, word1, word2, sep = " ") %>%
    count(bigram, sort = TRUE) %>% filter(n >= 5) %>% mutate(bigram = reorder(bigram,
    n)) %>% ggplot(aes(bigram, n)) + geom_col() + xlab(NULL) +
    coord_flip()
```



Arthur's certainly doing a lot of looking and staring! In fact a lot of the verbs following Arthur are passive, which jives quite well with my milquetoast impression of the character! What if we looked at words that preceded Arthur? Those tend to be verbs about speaking and perhaps this can also give us some insight into Arthur's character.

```
DougbigramFiltered %>% filter(word2 == "arthur") %>% unite(bigram,
    word1, word2, sep = " ") %>% count(bigram, sort = TRUE) %>%
    filter(n >= 5) %>% mutate(bigram = reorder(bigram, n)) %>%
    ggplot(aes(bigram, n)) + geom_col() + xlab(NULL) + coord_flip()
```

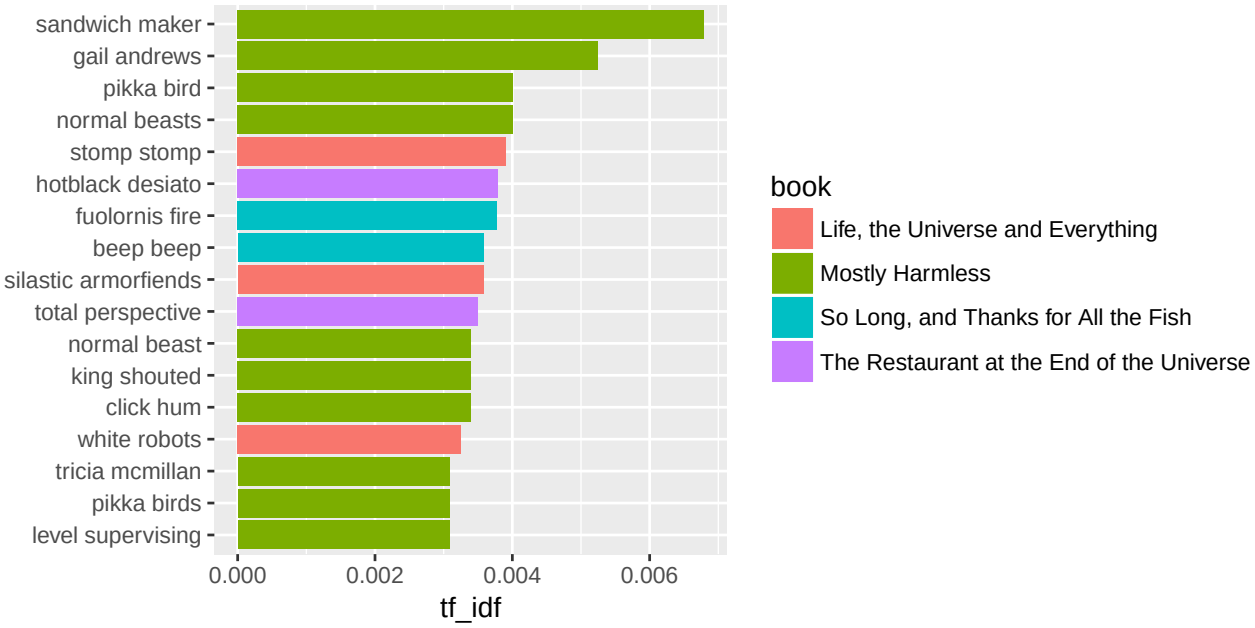


Here we get a slightly different picture of Arthur - he's saying things quite aggressively like shouting and insisting.

Just like with unigrams, we can adjust our bigram frequencies for overall frequency of bigrams in the trilogy.

This yields the following:

```
DougbigramFiltered %>% unite(bigram, word1, word2, sep = " ") %>%
    count(book, bigram) %>% bind_tf_idf(bigram, book, n) %>%
    arrange(desc(tf_idf)) %>% filter(tf_idf >= 0.003) %>% mutate(bigram = reorder(bigram,
    tf_idf)) %>% ggplot(aes(bigram, tf_idf, fill = book)) + geom_col() +
    xlab(NULL) + coord_flip()
```



Wow! Mostly Harmless really stands out with the bigrams. First is this "sandwich maker" bigram. What do we make of this? Well this bigram comes almost exclusively from Chapter 13 when it is used to describe Arthur's new persona on the utopian planet Lamuella, which is filled with "normal beasts" and "pikka birds" both of which are also bigrams with high tf-idf. Lamuella is unique to Mostly Harmless and is why these bigrams dominate according to our tf-idf statistic.

Accounting for Negations with Bigrams

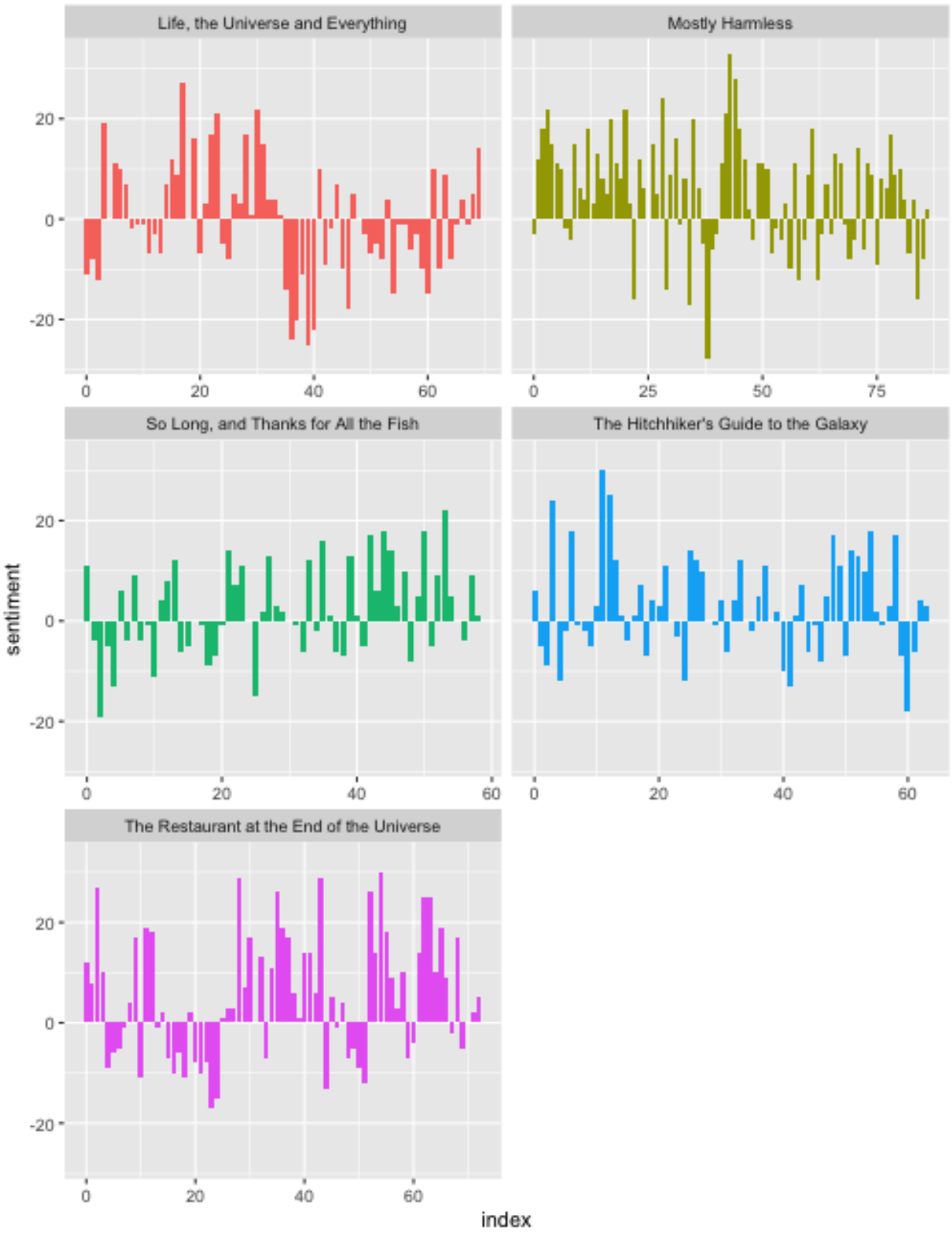
Bigram Network Graphs

Networks are a natural representation for bigrams. Individual words are the nodes and the links are determined by the frequency with which two words appear in a bigram together.

```
bigram_graph <- DougbigramCount %>% separate(bigram, c("word1",
    "word2"), sep = " ") %>% filter(n > 10) %>% graph_from_data_frame()

a <- grid::arrow(type = "closed", length = unit(0.15, "inches"))

bigram_graph %>% ggraph(layout = "fr") + geom_edge_link(aes(edge_alpha = n),
    show.legend = FALSE, arrow = a, end_cap = circle(0.07, "inches")) +
    geom_node_point(color = "lightblue", size = 5) + geom_node_text(aes(label = name),
    vjust = 1, hjust = 1) + theme_void()
```


Life, the Universe and Everything
Mostly Harmless
So Long, and Thanks for All the Fish
The Hitchhiker's Guide to the Galaxy
The Restaurant at the End of the Universe
sentiment
index
20
0
-20
0
20
40
60
25
50
75

Sentence Analysis with CoreNLP

Using single words (aka unigrams) in text analysis is a powerful but simplistic approach. As mentioned earlier, one drawback is the misclassification of negation words in sentiment analysis. What else can we do? We can use n-grams of $n > 1$ or we can use the natural unit of the sentence. Unfortunately (but justifiably) this introduces some incredible complexity into our analysis. **CoreNLP** is an open source Natural Language Processing software from Stanford. It contains remarkably sophisticated tools for processing the syntactic structure of sentences e.g. subject-object dependencies, entity recognition, etc. Perhaps Adams' prose will be a good challenge. Why? Check out this sentence:

> "Not unnaturally, many elevators imbued with intelligence and precognition became terribly frustrated with the mindless business of going up and down, up and down, experimented briefly with the notion of going sideways, as a sort of existential protest, demanded participation in the decision-making process and finally took to squatting in basements sulking."

That's a hell of a sentence to process. But regardless, let's take a look at some of CoreNLP's tools for processing sentences.