TDS Archive

Sentiment Analysis on the Texts of Harry Potter

With bonus tutorial of Matplotlib advanced features!

I'm Greg Rafferty, a data scientist in the Bay Area. You can check out the code for this project on my github. Feel free to contact me with any questions!

In this series of posts, I'm looking at a few handy NLP techniques, through the lens of Harry Potter. Previous posts in this series on basic NLP looked at Topic Modeling with Latent Dirichlet Allocation, Regular Expressions, and text summarization.

What is sentiment analysis?

In a previous post I looked at topic modeling, which is an NLP technique to learn the subject of a given text. Sentiment analysis exists to learn *what was said* about that topic — was it good or bad? With the growing use of the internet in our daily lives, vast amounts of unstructured text is being published every second of every day, in blog posts, forums, social media, and review sites, to name a few. Sentiment analysis systems can take this unstructured data and automatically add structure to it, capturing the public's opinion about products, services, brands, politics, etc. This data holds immense value in the fields of marketing analysis, public relations, product reviews, net promoter scoring, product feedback, and customer service, for example.

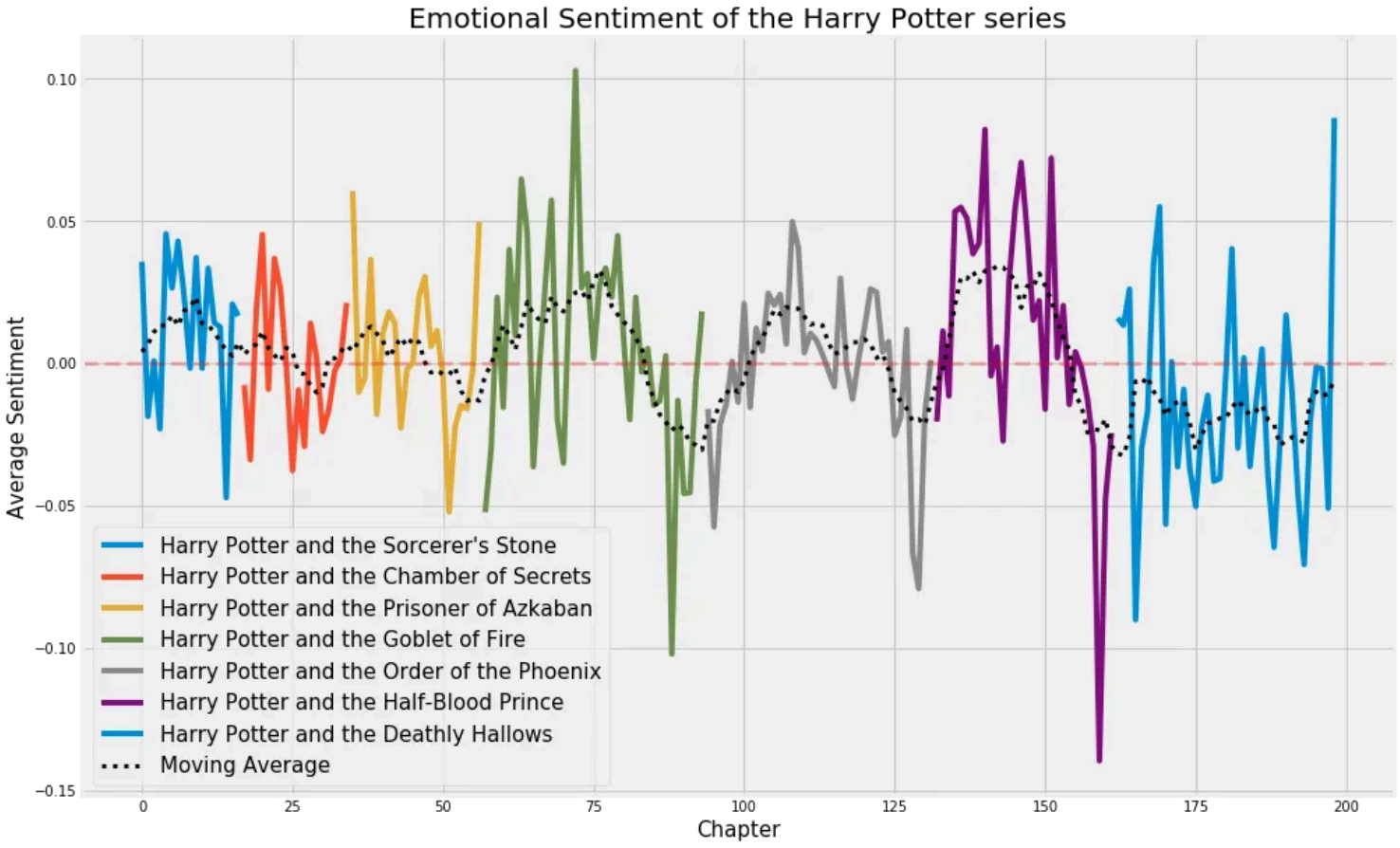
I've been demonstrating a lot of these NLP tasks using the text of Harry Potter. The books are rich in emotionally charged experiences that the reader can viscerally feel. Can a computer capture that feeling? Let's take a look.

VADER

I used C.J. Hutto's VADER package to extract the sentiment of each book. VADER, which stands for **V**alence **A**ware **D**ictionary and s**E**ntiment **R**easoning, is a lexicon and rule-based tool that is specifically tuned to social media. Given a string of text, it outputs a decimal between 0 and 1 for each of negativity, positivity, and neutrality for the text, as well as a compound score from -1 to 1 which is an aggregate measure.

A complete description of the development, validation, and evaluation of the VADER package can be read in this paper, but the gist is that the package's authors first constructed a list of lexical features correlated with sentiment and then combined the list with some rules that describe how the grammatical structure of a phrase will intensify or diminish the sentiment. When tested against human raters, VADER outperforms with accuracy scores of 96% to 84%.

VADER works best on short texts (a couple sentences at most), and applying it to an entire chapter at once resulted in extreme and largely worthless scores. Instead, I looped over each sentence individually, got the VADER scores, and then took an average of all sentences in a chapter.



Goblet of Fire, Chapter 16: Harry discovers the missionary position

By plotting at the VADER compound score for each chapter of each book, we can clearly mark events in the books. The three greatest spikes in that chart above are Harry being chosen by the Goblet of Fire around chapter 70 of the series, Cedric Diggory's death at about chapter 88, and Dumbledore's death at about chapter 160.

Here's the code to produce that chart (the full notebook is available on my Github). The data exists in a dictionary with each book's title as a key; the value for each book is another dictionary with each chapter number as a key. The value for each chapter is a tuple consisting of the chapter title and the chapter text. I defined a function to calculate the moving average of the data, which essentially smooths out the curve a bit and makes it easier to see long multi-chapter arcs throughout the stories. In order to plot each book as a different color, I created a dictionary called `book_indices` with each book's title as the key and the values being a 2-element tuple of the book's starting chapter number and ending chapter number (as if all the books were concatenated with chapters numbered sequentially throughout the entire series). I then plotted the story arc in segments based upon their chapter numbers.

```
import matplotlib.pyplot as plt

# Use FiveThirtyEight style theme
plt.style.use('fivethirtyeight')

# Moving Average function used for the dotted line
def movingaverage(interval, window_size):
    window = np.ones(int(window_size))/float(window_size)
    return np.convolve(interval, window, 'same')

length = sum([len(hp[book]) for book in hp])
x = np.linspace(0, length - 1, num=length)
```

```
y = [hp[book][chapter][2]['compound'] for book in hp for chapter in
hp[book]]

plt.figure(figsize=(15, 10))
for book in book_indices:
    plt.plot(x[book_indices[book][0]: book_indices[book][1]],
             y[book_indices[book][0]: book_indices[book][1]],
             label=book)
plt.plot(movingaverage(y, 10), color='k', linewidth=3, linestyle=':',
label = 'Moving Average')
plt.axhline(y=0, xmin=0, xmax=length, alpha=.25, color='r',
linestyle='--', linewidth=3)
plt.legend(loc='best', fontsize=15)
plt.title('Emotional Sentiment of the Harry Potter series',
fontsize=20)
plt.xlabel('Chapter', fontsize=15)
plt.ylabel('Average Sentiment', fontsize=15)
plt.show()
```

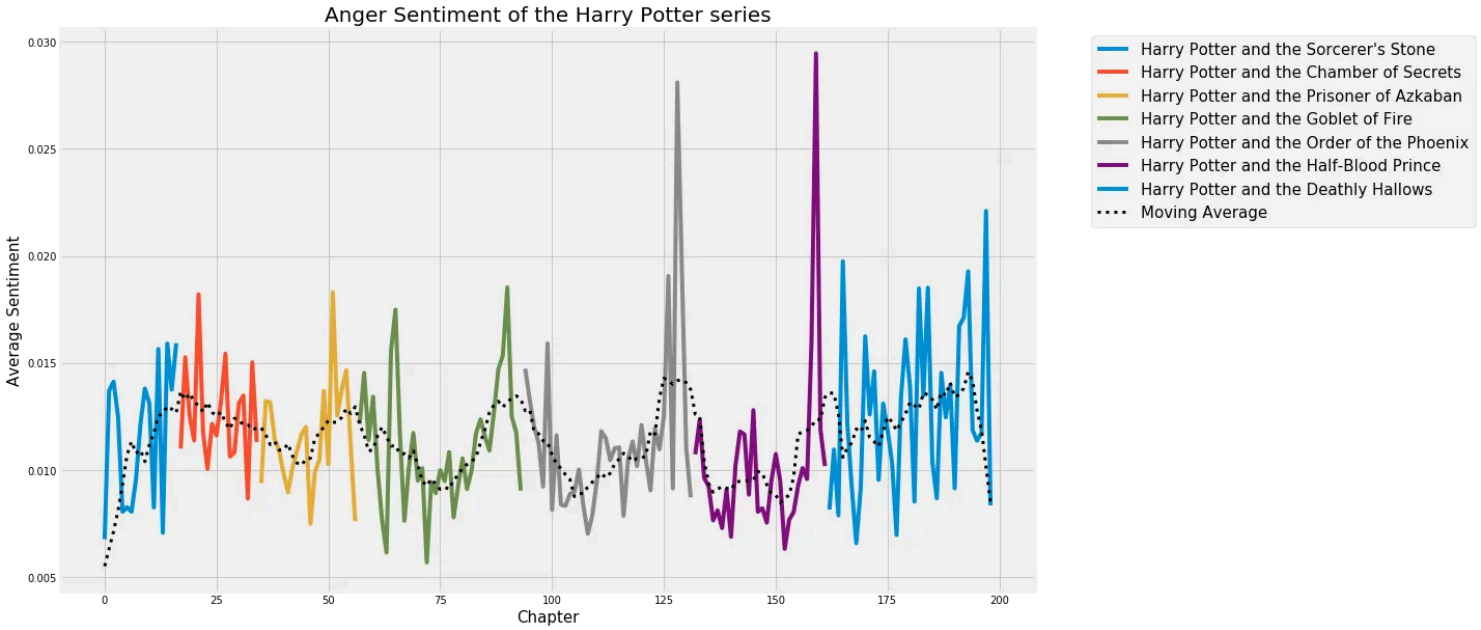
I also made this same chart using the TextBlob Naive Bayes and Pattern analyzers with worse results (see the Jupyter notebook on my Github for these charts). The Naive Bayes model was trained on movie reviews which must not translate well to the Harry Potter universe. The Pattern analyzer worked much better (almost as well as VADER); it is based on the Pattern library, a rule-based model very similar to VADER.

Emotion Lexicon

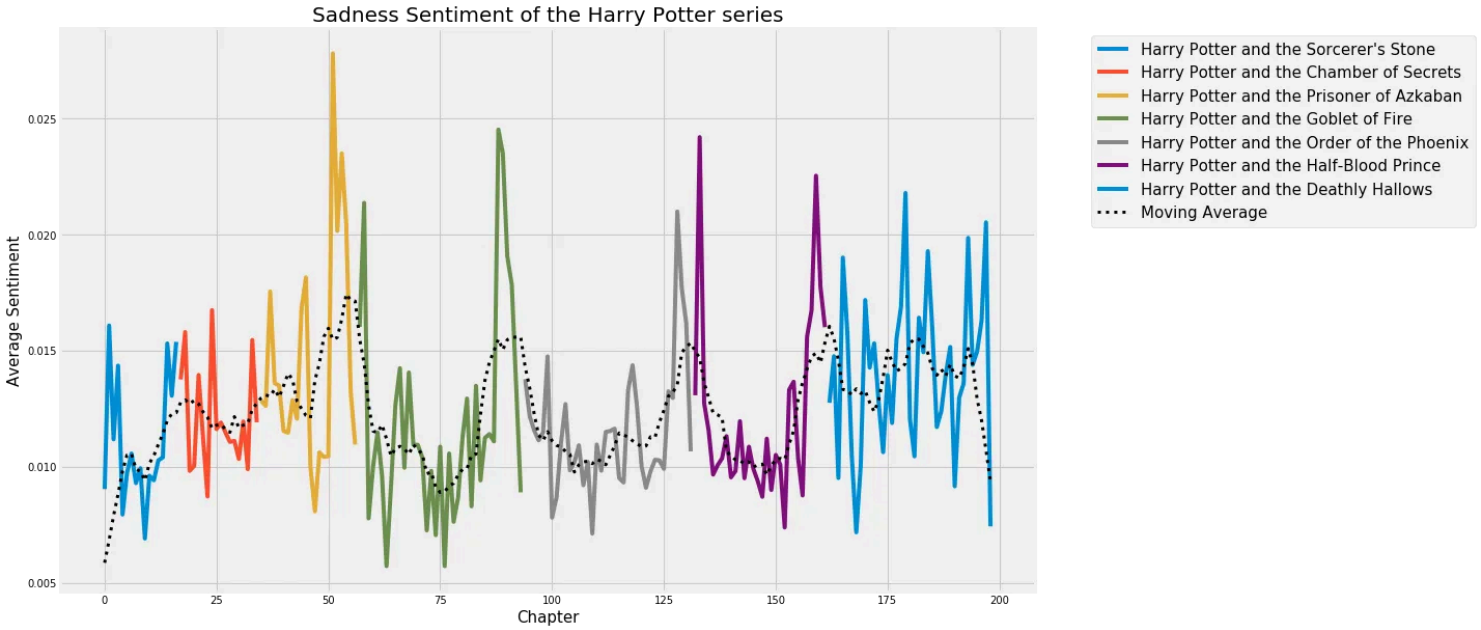
I also looked at emotions by using a lexicon created by the National Research Council of Canada of over 14,000 words, each scored as either associated or not-associated with any of two sentiments (negative, positive) or eight emotions (anger, anticipation, disgust, fear, joy, sadness, surprise, trust). They kindly provided me access to the lexicon, and I wrote up a Python script which loops over each word in a chapter, looks it up in the lexicon, and outputs whichever emotions the word was associated with. Each chapter was

then assigned a score for each emotion corresponding to a ratio of how many words associated with that emotion the chapter contains compared to the total word count in the chapter (this basically normalizes the scores).

Here are plots of the ‘Anger’ and ‘Sadness’ sentiments. I find it interesting that anger always exists with sadness, but sadness can sometimes exist without anger:

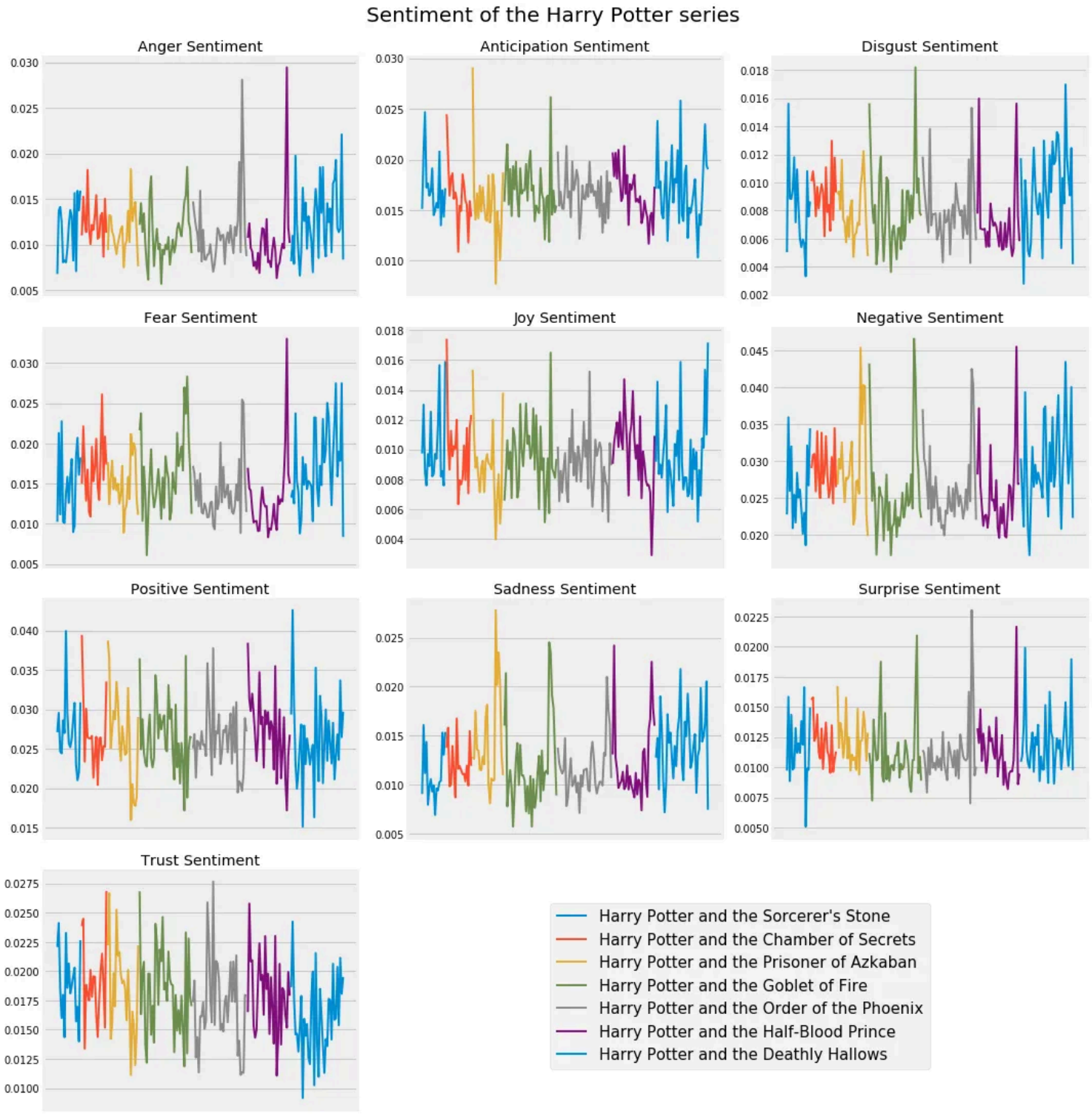


Wow, Voldemort. You really pissed off Harry when you killed the adults in his life



Those mood swings hit hard during puberty

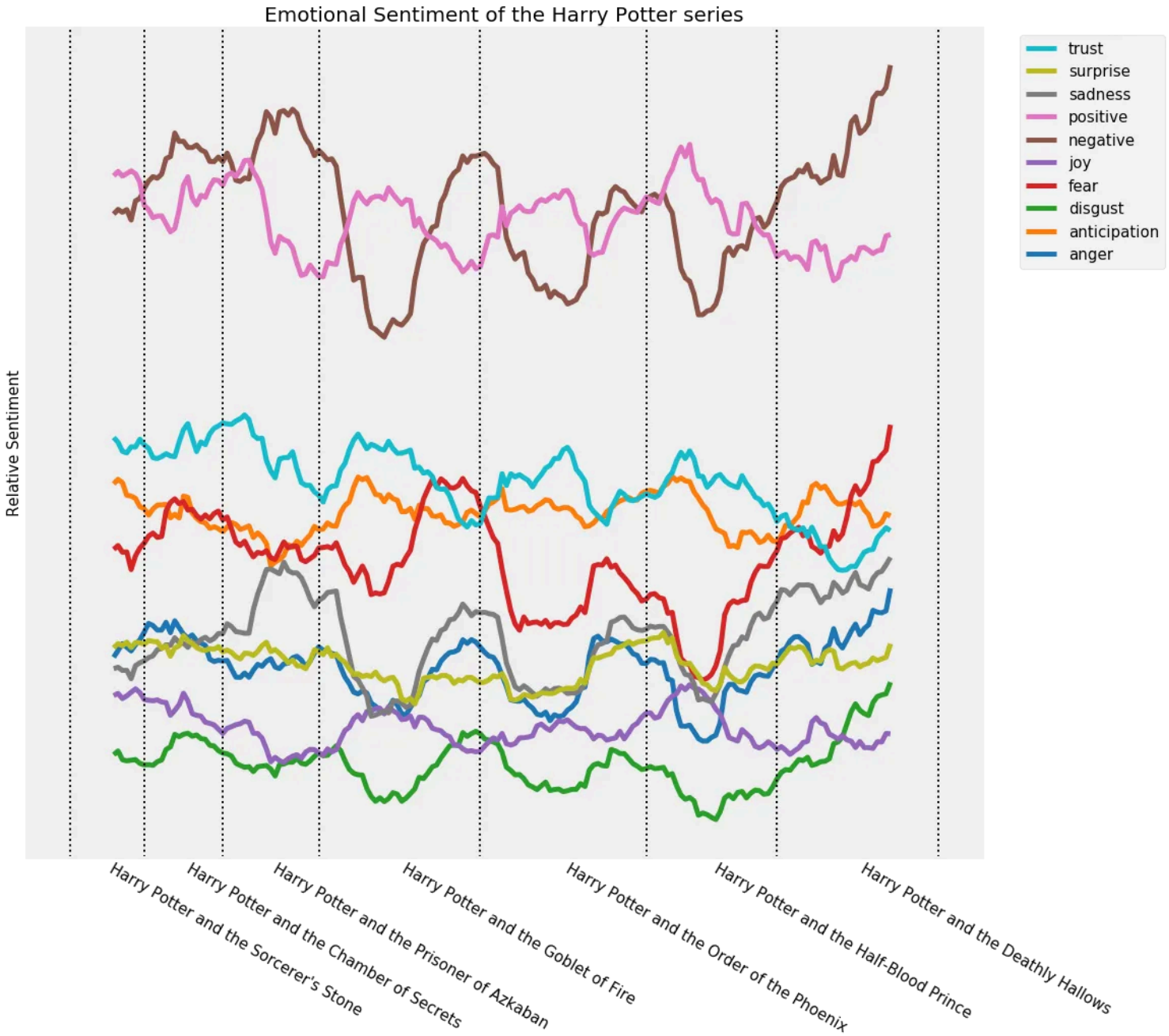
Again, take a look at the Jupyter notebook on my Github to see detailed charts for all sentiments. Here's a condensed version:



Let's see how I made all those subplots:

```
length = sum([len(hp[book]) for book in hp])
x = np.linspace(0, length - 1, num=length)

fig, ax = plt.subplots(4, 3, figsize=(15, 15), facecolor='w',
edgecolor='k')
fig.subplots_adjust(hspace = .5, wspace=.1)
fig.suptitle('Sentiment of the Harry Potter series', fontsize=20,
y=1.02)
fig.subplots_adjust(top=0.88)

ax = ax.ravel()

for i, emotion in enumerate(emotions):
    y = [hp_df.loc[book].loc[hp[book][chapter][0]][emotion] for book
in hp for chapter in hp[book]]
    for book in book_indices:
        ax[i].plot(x[book_indices[book][0]: book_indices[book][1]],
                y[book_indices[book][0]: book_indices[book][1]],
                label=book, linewidth=2)

ax[i].set_title('{} Sentiment'.format(emotion.title()))
    ax[i].set_xticks([])

fig.legend(list(hp), loc='upper right', fontsize=15, bbox_to_anchor=
(.85, .2))
fig.tight_layout()
fig.delaxes(ax[-1])
fig.delaxes(ax[-2])
plt.show()
```

But it really becomes interesting to see how all the sentiments compare to each other. Overlaying 10 lines with this much variance quickly became a mess, so I again used the moving average:



It's interesting to see contradicting emotions acting counter to each other, most obviously the pink and brown lines above for 'Positive' and 'Negative' sentiment. Note that, due to the moving average window size of 20 data points, the first 10 and last 10 chapters have been left off the plot.

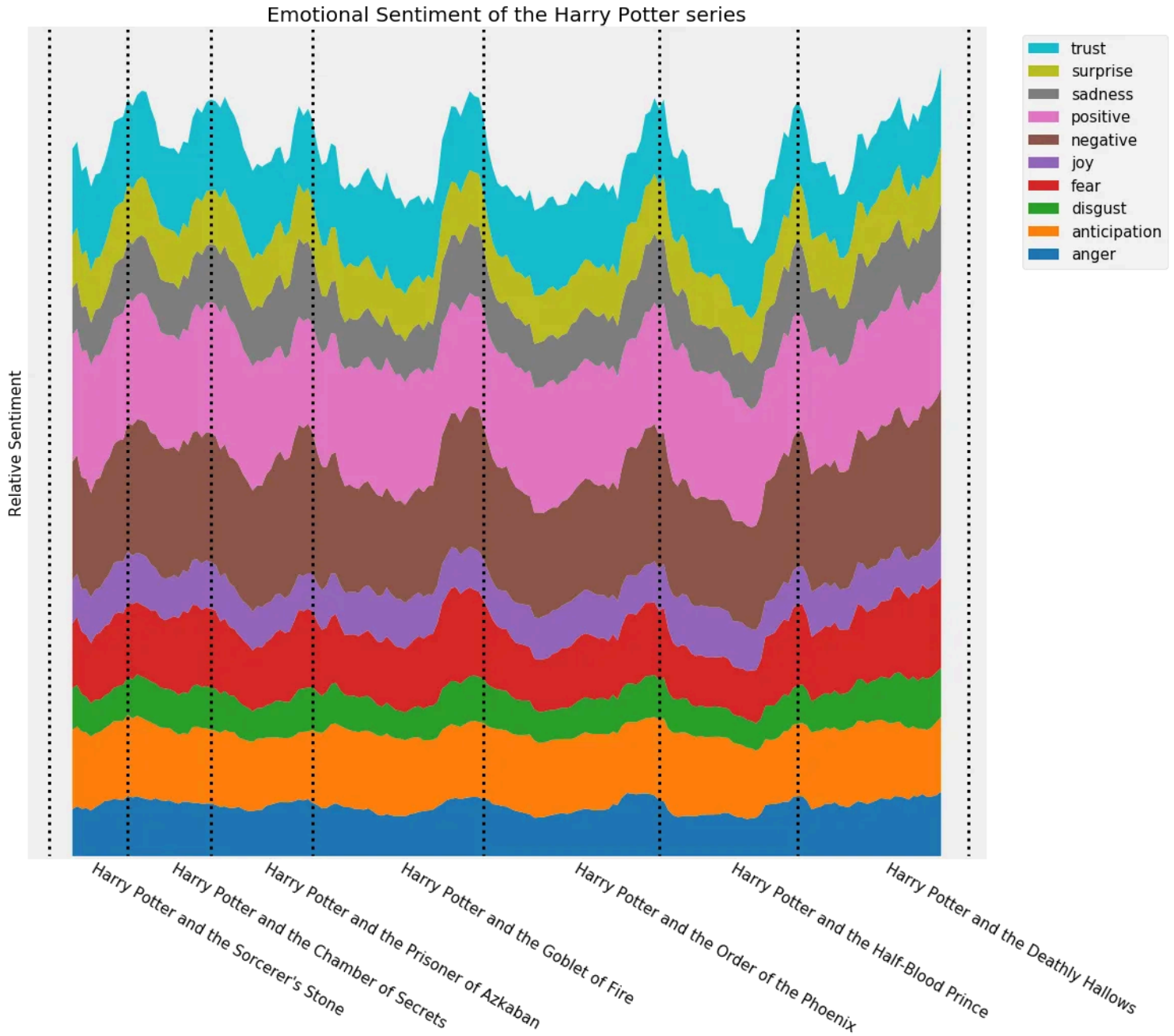
I removed the y-axis because those numbers are meaningless to us (mere decimals: the ratio of words of that emotion to total words in the chapter). I also removed the horizontal and vertical chart lines to clean up the plot. I don't particularly care to mark regular chapter numbers but I do want to mark the books; therefore, I added those vertical dotted lines. The legend has been reversed in this plot, which isn't really necessary for readability or anything but I did it for consistency with the area and column charts coming up next.

Here's how I made it:

```
# use the Tableau color scheme of 10 colors
tab10 = matplotlib.cm.get_cmap('tab10')

length = sum([len(hp[book]) for book in hp])
window = 20

# use index slicing to remove data points outside the window
x = np.linspace(0, length - 1, num=length)[int(window / 2): -
int(window / 2)]

fig = plt.figure(figsize=(15, 15))
ax =fig.add_subplot(1, 1, 1)

# Loop over the emotions with enumerate in order to track colors
for c, emotion in enumerate(emotions):
    y = movingaverage([hp_df.loc[book].loc[hp[book][chapter][0]]
[emotion] for book in hp for chapter in hp[book]], window)[int(window
/ 2): -int(window / 2)]
    plt.plot(x, y, linewidth=5, label=emotion, color=(tab10(c)))

# Plot vertical lines marking the books
for book in book_indices:
    plt.axvline(x=book_indices[book][0], color='black', linewidth=2,
linestyle=':')
plt.axvline(x=book_indices[book][1], color='black', linewidth=2,
linestyle=':')

plt.legend(loc='best', fontsize=15, bbox_to_anchor=(1.2, 1))
plt.title('Emotional Sentiment of the Harry Potter series',
fontsize=20)
plt.ylabel('Relative Sentiment', fontsize=15)
```

```
# Use the book titles for X ticks, rotate them, center the left edge
plt.xticks([(book_indices[book][0] + book_indices[book][1]) / 2 for
book in book_indices],
           list(hp),
           rotation=-30,
           fontsize=15,
           ha='left')
plt.yticks([])

# Reverse the order of the legend
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles[::-1], labels[::-1], loc='best', fontsize=15,
bbox_to_anchor=(1.2, 1))

ax.grid(False)

plt.show()
```

I also made an area plot to show the overall emotive qualities of each chapter. This is again a moving average in order to smooth out the more extreme spikes and to show the story arc better across all books:



The books seem to start with a bit of trailing emotion from the previous story but quickly calm down during the middle chapters only to pick back up again at the end.

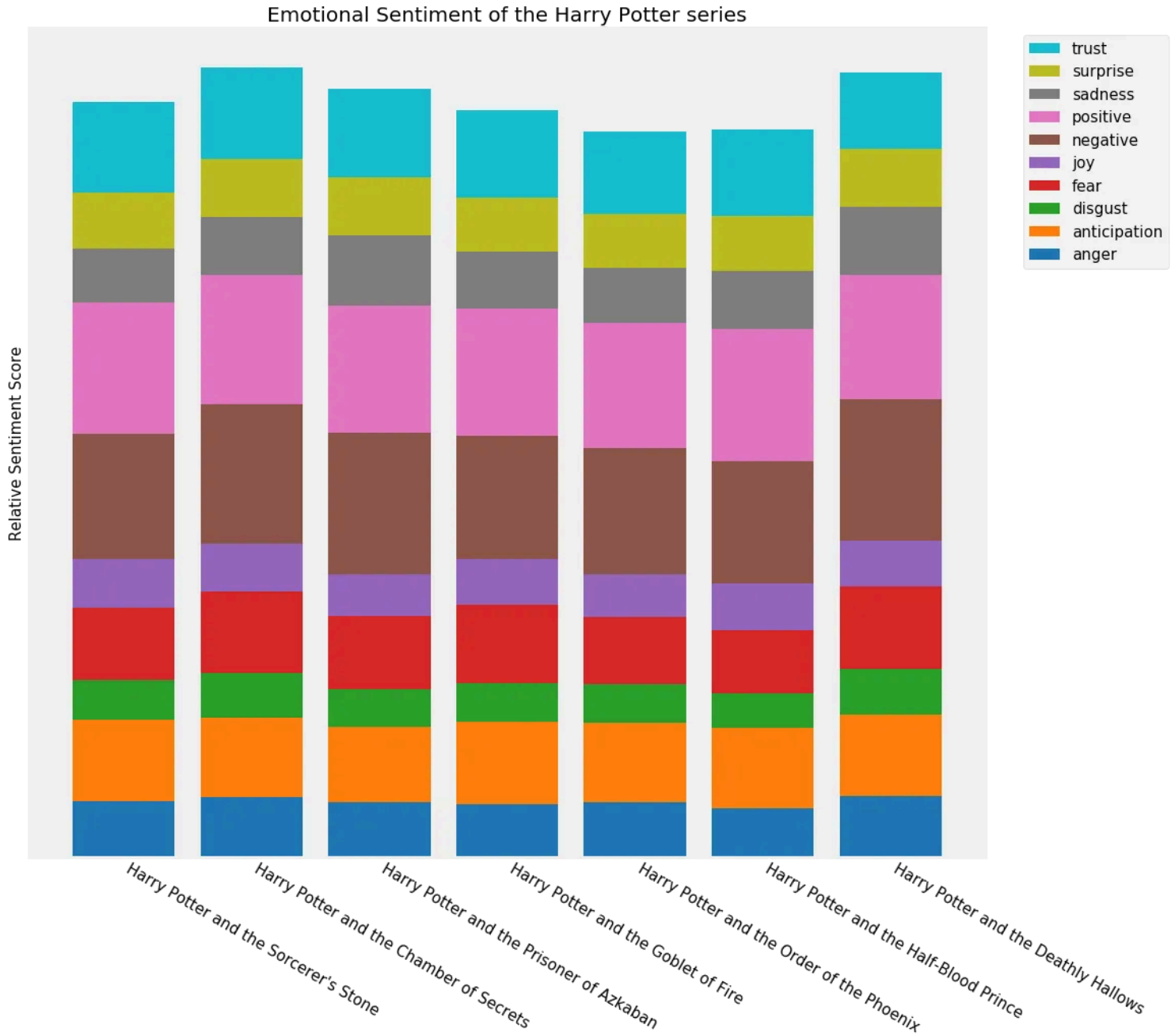
```
length = sum([len(hp[book]) for book in hp])
window = 10
```

```
x = np.linspace(0, length - 1, num=length)[int(window / 2): -
int(window / 2)]

fig = plt.figure(figsize=(15, 15))
ax = fig.add_subplot(1, 1, 1)

y = [movingaverage(hp_df[emotion].tolist(), window)[int(window / 2):
-int(window / 2)] for emotion in emotions]

plt.stackplot(x, y, colors=(tab10(0),
                            tab10(.1),
                            tab10(.2),
                            tab10(.3),
                            tab10(.4),
                            tab10(.5),
                            tab10(.6),
                            tab10(.7),
                            tab10(.8),
                            tab10(.9)), labels=emotions)

# Plot vertical lines marking the books
for book in book_indices:
    plt.axvline(x=book_indices[book][0], color='black', linewidth=3,
linestyle=':')
plt.axvline(x=book_indices[book][1], color='black', linewidth=3,
linestyle=':')

plt.title('Emotional Sentiment of the Harry Potter series',
fontsize=20)
plt.xticks([(book_indices[book][0] + book_indices[book][1]) / 2 for
book in book_indices],
           list(hp),
           rotation=-30,
           fontsize=15,
           ha='left')
plt.yticks([])
plt.ylabel('Relative Sentiment', fontsize=15)

# Reverse the legend
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles[::-1], labels[::-1], loc='best', fontsize=15,
bbox_to_anchor=(1.2, 1))

ax.grid(False)

plt.show()
```

Note how in this chart, reversing the legend became necessary for readability. By default, the legend items are added in alphabetical order going down, but the data is stacked from the bottom up. So the colors of the

legend and the area plot run in opposite direction — to my eye, quite confusing and difficult to follow. So with 'Anger' plotted at the bottom, I also wanted it to be on the bottom of the legend and likewise with 'Trust' at the top.

And lastly, a stacked bar chart to show the weights of the various sentiments across the books:



Naturally, words associated with any of the positive emotions would also be associated with the 'Positive' sentiment, and likewise for 'Negative', so it shouldn't come as a surprise that those two sentiments carry the bulk of the emotive quality of the books. I find it notable that the emotions are relatively consistent from book to book with just slight differences in magnitude but consistent weights, except for the 'Fear' emotion in red; it seems to exhibit the most variance across the series. I also would have expected the cumulative magnitude of sentiments to increase throughout the series as the stakes became higher and higher; however although the final book is indeed the highest, the other 6 books don't show this gradual increase but almost the opposite, with a constant decline starting with book 2.

```
books = list(hp)
margin_bottom = np.zeros(len(books))

fig = plt.figure(figsize=(15, 15))
ax = fig.add_subplot(1, 1, 1)

for c, emotion in enumerate(emotions):
    y = np.array(hp_df2[emotion])
    plt.bar(books, y, bottom=margin_bottom, label=emotion, color=
(tab10(c)))
    margin_bottom += y

# Reverse the legend
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles[::-1], labels[::-1], loc='best', fontsize=15,
bbox_to_anchor=(1.2, 1))

plt.title('Emotional Sentiment of the Harry Potter series',
fontsize=20)
plt.xticks(books, books, rotation=-30, ha='left', fontsize=15)
plt.ylabel('Relative Sentiment Score', fontsize=15)
plt.yticks([])
ax.grid(False)
plt.show()
```

The tricky bit in this plot is using the `margin_bottom` variable to stack each of the columns. Other than that, it just uses a couple tricks from the previous plots.

Machine Learning | Data Science | NLP | Harry Potter | Python

Responses (7)