Simple Sentiment Analysis for Survey Feedback

Using feedback given to Indiana University's University Information Technology Services, we will create plots to analyze how various UITS users, faculty, staff, graduate students, and undergraduate students feel about the processes UITS completes.

Global parameters

Set working directory by pointing to the location on your computer where you have stored the files. Below, we have chosen to Save the folder "RAnalysis" on the Desktop on a Mac. It contains all the other R scripts, texts, notebooks, and results. If you have branched the Github, simply note where you have save the folder. If you are on a PC, you will need to use an absolute path such as "C:Users:XXX."

Hint: Can't figure out your syntax? Click Session > Set Working Directory > Choose Directory, then select the Text_Analysis directory in which you are working. This will set your working directory in the console below while you are working here, but make sure to copy the path into the "setwd" command below to keep the directory constant if you close this script and reopen later. ***

```
setwd("~/Desktop/R/Text_Analysis/RNotebooks")
```

Include necessary packages for notebook

```
library(knitr)
opts_chunk$set(tidy.opts=list(width.cutoff=50), tidy=TRUE)
library(formatR)
library(markdown)
library(rmarkdown)
library(devtools)
library(twitteR)
library(plyr)
```

```
##
## Attaching package: 'plyr'

## The following object is masked from 'package:twitteR':
##
##     id
```

```
library(ggplot2)
```

Load data

The text files were gathered from the public UITS survey results on the UITS website. Here, we scan in the text files and separate on the characters to create character vector of words.

Hint 1: The names of the variables can be anything (happiness, rainbows, cats), but usually, you want to name them using terms to indicate what the variable holds. Here we use faculty.text to represent the feedback faculty gave on the survey. The period (.) between "faculty" and "text" does not serve any purpose other than to separate the two terms for ease of reading.

Hint 2: Another popular naming convention is to list the type of object with the name, for example faculty.text.l since this object is a list. This convention helps when manipulating the data later.

```
faculty.text <- scan("~/Desktop/R/Text_Analysis/data/survey/2015UITSFaculty.txt",
    what = "character", sep = "\n")
staff.text <- scan("~/Desktop/R/Text_Analysis/data/survey/2015UITSStaff.txt",
    what = "character", sep = "\n")
grad.text <- scan("~/Desktop/R/Text_Analysis/data/survey/2015UITSGrad.txt",
    what = "character", sep = "\n")
undergrad.text <- scan("~/Desktop/R/Text_Analysis/data/survey/2015UITSUndergrad.txt",
    what = "character", sep = "\n")
```

Inspect Data

Before preparing the data for analysis, let us first see what we are working with. For tutorial purposes, we will look at the first: faculty.text, but feel free to use the same commands to inspect any of the variables we have created. This step is a good idea to ensure you know what you are ultimately analyzing. For example, if you find that the length is zero, you may have to go back and reload or re-grab the data.

Class describes the type of object.

```
class(faculty.text)
```

```
## [1] "character"
```

Length shows how many elements the object has. In this case, it will show how many comments were made by faculty.

```
length(faculty.text)
```

```
## [1] 82
```

We can also look at individual elements - Lets see what the first comment is…

```
faculty.text[1]
```

```
## [1] "1. Knowledge base can be improved. There is \"stale\" information there that makes finding up t
```

Or perhaps you would like to see the first few elements:

```
head(faculty.text)
```

```
## [1] "1. Knowledge base can be improved. There is \"stale\" information there that makes finding up t
## [2] "2. It would help to have detailed guidance for routine tasks on knowledge base. Sometimes I fee
## [3] "3. With respect to security--I get it. However the security policy revolves around information t
## [4] "4. Wireless access is awful. I would say that I am unable to access iusecure wireless in the B-s
## [5] "5. The printing situation is arcane. For class I must print paper copies on my local printer, g
## [6] "6. I appreciate that the construction and moves have created a great deal of additional work an
```

There are many more ways to inspect parts of data (check out the CRAN), but these quick checks are helpful while manipulating and trimming the data pre-analysis.

Now we are all set! Let's load in some postivity (and negativity, I guess…)!

Loading the Opinion Lexicons to Determine Sentiment

This is an essential step for sentiment analysis. These text documents from Hu and Liu, 2004* are filled with positive and negative words, respectively. The algorithm we will write next will check these documents to score each word in the survey feedback. If the algorithm runs across the word "love" in the text, it will check the positive-words.txt file, find "love" is included, and score the word with a +1. More on that in a second…

```
lex.pos = scan("~/Desktop/R/Text_Analysis/data/opinionLexicon/positive-words.txt",
    what = "character", comment.char = ";")
lex.neg = scan("~/Desktop/R/Text_Analysis/data/opinionLexicon/negative-words.txt",
    what = "character", comment.char = ";")
```

Add words relevant to our corpus using the combine c() function:

```
pos.words = c(lex.pos, "upgrade")
neg.words = c(lex.neg, "wait", "waiting", "mechanical",
    "delay", "arcane", "difficult")
```

Implement the sentiment scoring algorithm

Here is where the magic happens - let's create the algorithm! To begin a function which will iterate over all of the elements in the specified object, we need to give it a name - score.sentiment sounds good. First, we set the name of our function equal to function(). Then, we fill the parenthesis with our arguments. Here, we want an argument for the feedback (our data we gathered), our positive words, and our negative words.

We will then require the plyr and stringr packages. The plyr package splits, combines, and applies data (the laply function used below comes with this package) and the stringr package which eases string operations (we will use str_split below, which is included with the stringr package).

This function takes in an argument (feedback for us), normalizes all of the text, splits the comments into separate words to analyze, compares the feedback words to the positive and negative dictionaries, returns a TRUE if the word is in the dictionary and FALSE if it is not, the trues and falses are summed up for each comment (this is the score of each comment in the feedback), and finally, the comment's score and its text are returned as a data frame for ease of plotting.

```
score.sentiment = function(text, pos.words, neg.words,
    .progress = "none") {
    require(plyr)
    require(stringr)

    # figure out the score for each comment
    # specifically
    scores = laply(text, function(comment, pos.words,
        neg.words) {
        # normalize comment text
        comment = gsub("[[:punct:]]", "", comment)
        comment = gsub("[[:cntrl:]]", "", comment)
        comment = gsub("\\d+", "", comment)

        # REMOVE EMOJIS!
        comment = iconv(comment, "ASCII", "UTF-8",
            sub = "")

        comment.lower = tolower(comment)

        # split list into separate words
        word.list = str_split(comment.lower, "\\s+")
        words = unlist(word.list)

        # compare our words to the dictionaries of positive
        # and negative terms using match function
        pos.matches = match(words, pos.words)
        neg.matches = match(words, neg.words)

        # match returns a position of the matched term or
        # NA, but we just want the TRUE/FALSE, not NA
        pos.matches = !is.na(pos.matches)
        neg.matches = !is.na(neg.matches)

        # the score of each comment is the sum of the
        # positive matches minus the sum of the negative
        # matches
        score = sum(pos.matches) - sum(neg.matches)

        return(score)
    }, pos.words, neg.words, .progress = .progress)

    # compile the scores and text of text into a data
    # frame for plotting
    scores.df = data.frame(score = scores, text = text)
    return(scores.df)
}
```

Algorithm Testing

Whenever you create a function (or algorithm), it is best to test it on some sample data to ensure it behaves as you expect. Let's create a sample list of emotional comments and save it as "sample".

```
sample = c("This ice cream is the best! I love this flavor!",
    "I am so angry at the terrible weather today. Frustrated.",
    "Wow, spectacular, I wish I could be as perfect as you.")
```

We already did the hard part by building our score.sentiment function earlier. Here, we just need to tell the algorithm what to use as arguments! We need to put our sample data in as the "comments" argument, pos.words as the "pos.words"" argument, and neg.words as the "neg.words" argument!

Lets also save the result of running the algorithm as an object called sample.result.

```
sample.result = score.sentiment(sample, pos.words,
    neg.words)
```

```
## Loading required package: stringr
```

Did it work?

```
sample.result
```

```
##   score                                                     text
## 1     2          This ice cream is the best! I love this flavor!
## 2    -3 I am so angry at the terrible weather today. Frustrated.
## 3     3   Wow, spectacular, I wish I could be as perfect as you.
```

It worked! Now we can be sure our algorithm is behaving as we would expect. Lets score our UITS feedback data since we are confident in our function-creating abilities…

Scoring Twitter Data

Again, we have already created our function, we just need to tell it what to analyze. We will use our [respondentgroup].text objects as the "comments" argument and save each as an object called [respondentgroup].result to plot a little later.

```
faculty.result = score.sentiment(faculty.text, pos.words,
    neg.words)
staff.result = score.sentiment(staff.text, pos.words,
    neg.words)
grad.result = score.sentiment(grad.text, pos.words,
    neg.words)
undergrad.result = score.sentiment(undergrad.text,
    pos.words, neg.words)
```

Lets peek at the results to see if it is still working…

```
head(faculty.result)
```

```
##   score
## 1     0
## 2     4
## 3     3
## 4    -2
## 5     3
## 6     4
##
## 1
## 2
## 3                                            3. With respect to security--I get it. However the secu
## 4
## 5 5. The printing situation is arcane. For class I must print paper copies on my local printer, give
## 6
```

```r
head(staff.result)
```

```
##   score
## 1     2
## 2     2
## 3     1
## 4     3
## 5     0
## 6     1
##
## 1                                                                                       1- Glad to s
## 2
## 3
## 4 4- Support more mobile apps! Shame about the "official" Outlook app, which looked so promising, bu
## 5
## 6
```

```r
head(grad.result)
```

```
##   score
## 1     1
## 2     1
## 3     0
## 4    -2
## 5     2
## 6     0
##
## 1 Add a new User interface or atleast more instructions, at new ID swipe printer stations, regarding
## 2
## 3
## 4
## 5
## 6
```

```r
head(undergrad.result)
```

```
##   score
## 1     1
## 2     5
## 3     0
## 4     1
## 5     1
## 6     4
##
## 1
## 2
## 3
## 4
## 5 I would like IUTS to offer students the ability to access audio manuals. I have issues reading but
## 6
```

Plotting Twitter Data

To visualize the sentiment of these particular comments, we will use the plotting system ggplot2 and its quick plot (qplot) functionality. The qplot function takes an object, which is the score from our comments, then you can specify other parts of the plot. For example, we have titled our plots “Sentiment of [Indiana University representative group] for UITS Survey,” labeled the x-axis “Valence of Sentiment (Comment Score),” and labeled the y-axis “Count (Comments).” These were all saved as [respondentgroup].plot objects to use later.

Below, we impose the “Black and White” theme on the plots

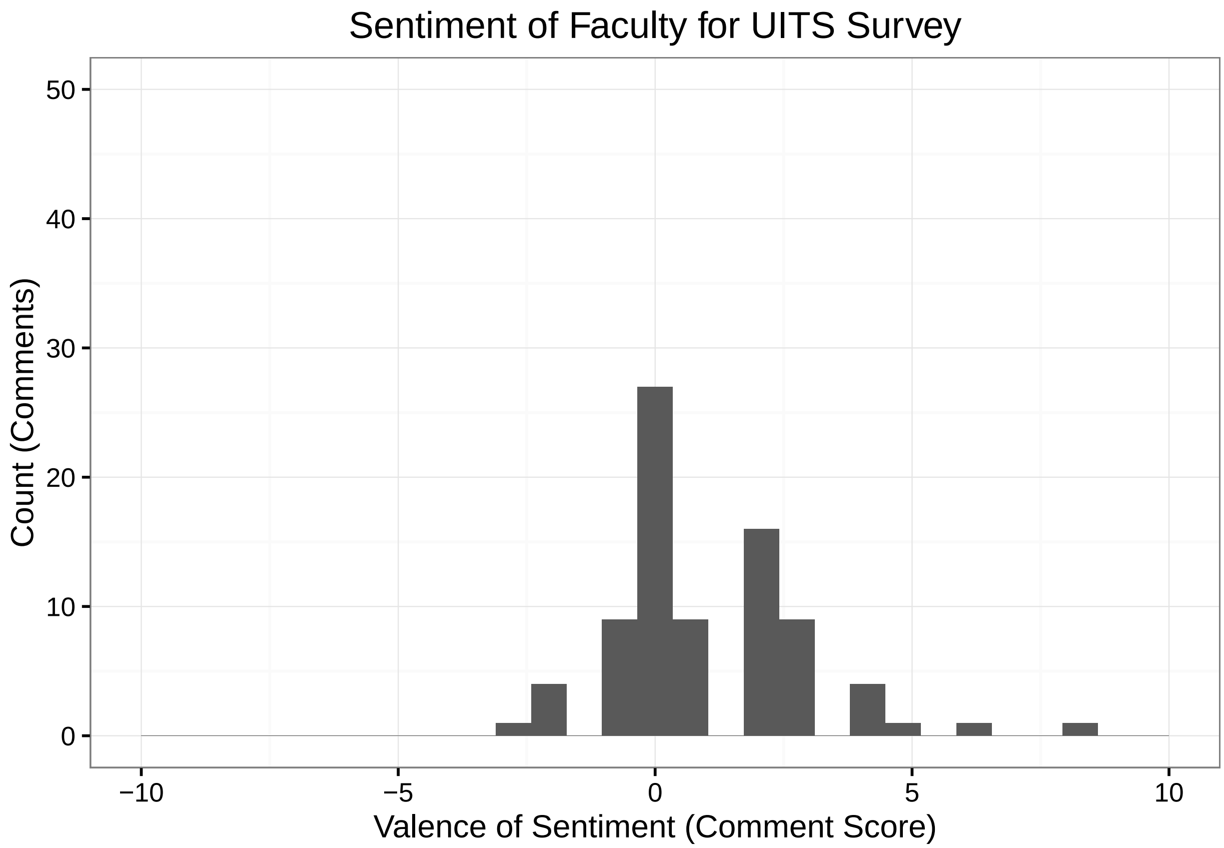
```
faculty.plot = qplot(faculty.result$score,
    main = "Sentiment of Faculty for UITS Survey",
    xlab = "Valence of Sentiment (Comment Score)",
    ylab = "Count (Comments)", xlim = c(-10,
        10), ylim = c(0, 50))
staff.plot = qplot(staff.result$score,
    main = "Sentiment of Staff for UITS Survey",
    xlab = "Valence of Sentiment (Comment Score)",
    ylab = "Count (Comments)", xlim = c(-10,
        10), ylim = c(0, 50))
grad.plot = qplot(grad.result$score,
    main = "Sentiment of Graduate Students for UITS Survey",
    xlab = "Valence of Sentiment (Comment Score)",
    ylab = "Count (Comments)", xlim = c(-10,
        10), ylim = c(0, 50))
undergrad.plot = qplot(undergrad.result$score,
    main = "Sentiment of Undergraduate Students for UITS Survey",
    xlab = "Valence of Sentiment (Comment Score)",
    ylab = "Count (Comments)", xlim = c(-10,
        10), ylim = c(0, 50))

faculty.plot = faculty.plot + theme_bw()
staff.plot = staff.plot + theme_bw()
grad.plot = grad.plot + theme_bw()
undergrad.plot = undergrad.plot + theme_bw()
```

Output each plot!

```
faculty.plot
```

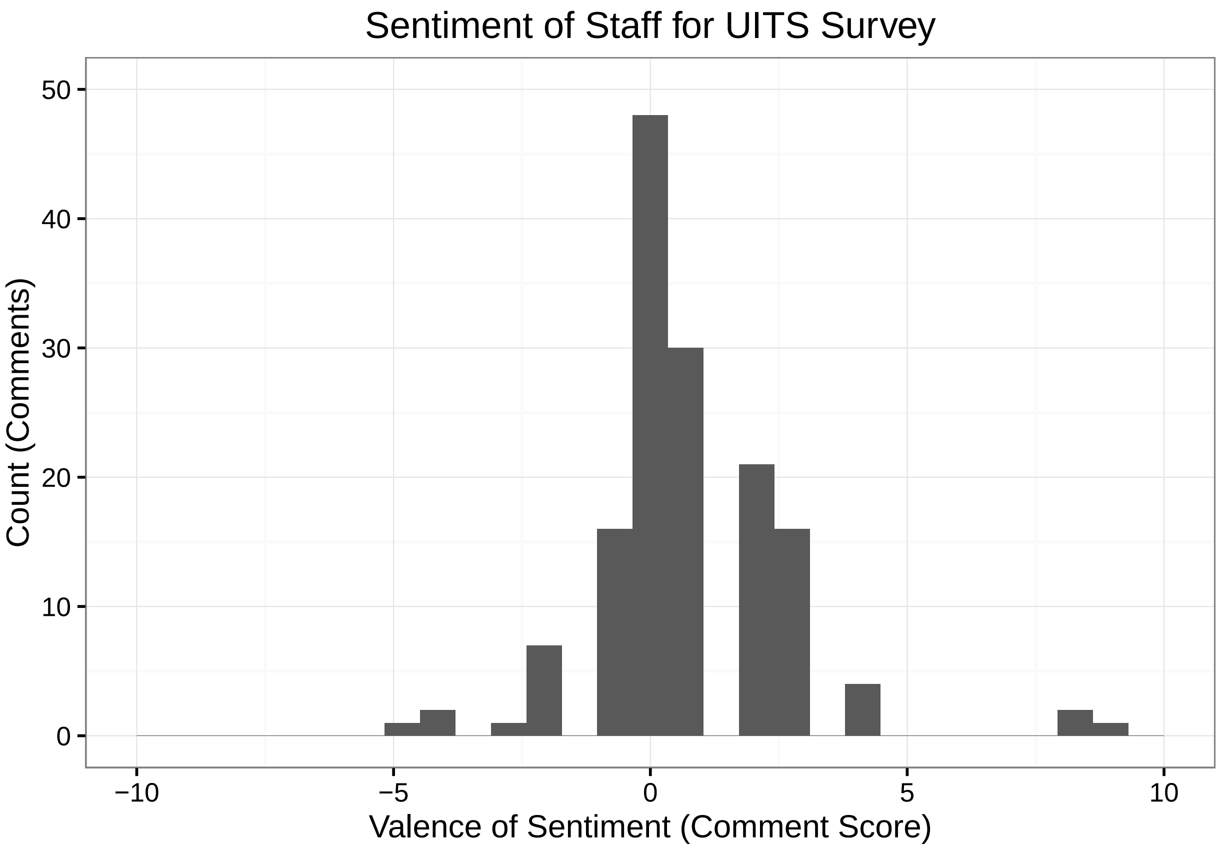
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
staff.plot
```

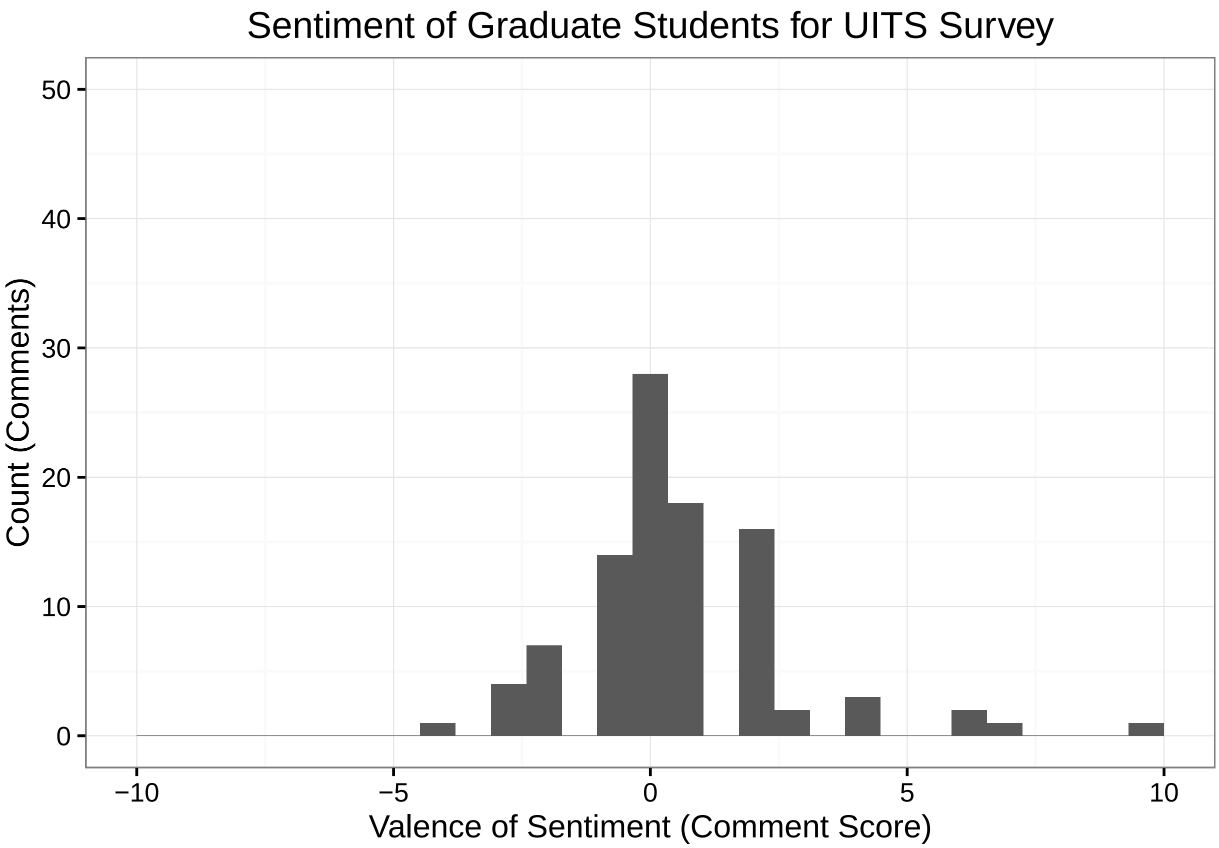
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```



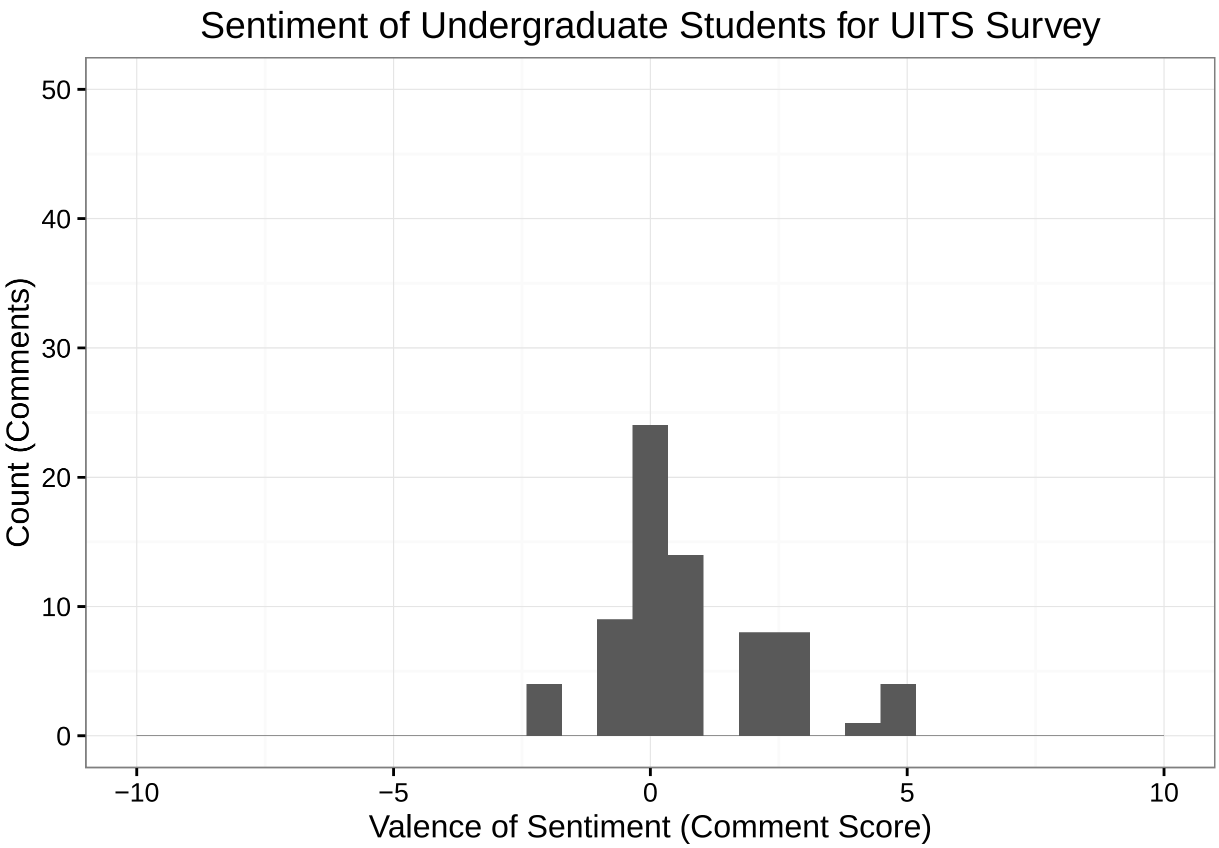
```
grad.plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
undergrad.plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Voila!

Acknowledgements: This algorithm was adapted from Jeffrey Breen's Mining Twitter for Airline Consumer Sentiment article. You can find it here: http://www.inside-r.org/howto/mining-twitter-airline-consumer-sentiment.

- Liu, Minqing Hu and Junsheng Cheng. "Opinion Observer: Analyzing and Comparing Opinions on the Web." Proceedings of the 14th International World Wide Web conference (WWW-2005), May 10-14, 2005, Chiba, Japan.