From Objects to Data Project

Daniël Schene, Tim van Dort, Tiffany Ng

20-12-2018

This Rmarkdown file is made to give some insight into how we processed the data aquired with Python here in R.

We make use of the ggplot2 package for visualisation:

```
library(ggplot2)
data_raw = read.csv('/Users/danielschene/Desktop/DataFinal.csv')
attach(data_raw)
```

Here is a summary of our dataframe:

```
summary(data_raw)
```

```
##                         Work       NoOfSen        NoOfWords
##  1984                     : 1   Min.   :  797   Min.   : 22385
##  AliceInWonderland        : 1   1st Qu.: 3530   1st Qu.: 75587
##  AnnaKarenina             : 1   Median : 5568   Median :110676
##  AroundTheWorldIn80Days: 1   Mean   : 6489   Mean   :141003
##  BraveNewWorld            : 1   3rd Qu.: 8107   3rd Qu.:167696
##  BrothersKaramazov        : 1   Max.   :26431   Max.   :595904
##  (Other)                  :48
##     AvgWperS        UniqueW        UniqRatio        LongstSen
##  Min.   :10.83   Min.   : 2724   Min.   : 3.739   Min.   :  76.0
##  1st Qu.:17.48   1st Qu.: 7078   1st Qu.: 6.004   1st Qu.: 135.5
##  Median :21.64   Median : 9297   Median : 8.769   Median : 182.0
##  Mean   :22.82   Mean   : 9891   Mean   : 8.726   Mean   : 229.2
##  3rd Qu.:26.68   3rd Qu.:12001   3rd Qu.:10.618   3rd Qu.: 272.5
##  Max.   :55.26   Max.   :22282   Max.   :16.075   Max.   :1360.0
##
##     AvgWLen      FirstPublished  NoOfRatings        GRrating
##  Min.   :3.940   Min.   :1605   Min.   :  16157   Min.   :3.420
##  1st Qu.:4.368   1st Qu.:1851   1st Qu.: 172961   1st Qu.:3.772
##  Median :4.608   Median :1876   Median : 280888   Median :3.850
##  Mean   :4.584   Mean   :1870   Mean   : 615040   Mean   :3.867
##  3rd Qu.:4.781   3rd Qu.:1906   3rd Qu.: 709025   3rd Qu.:4.000
##  Max.   :5.155   Max.   :1960   Max.   :3755275   Max.   :4.320
##
##     LTrating      LTpopularity
##  Min.   :3.450   Min.   :   9.0
##  1st Qu.:3.803   1st Qu.:  59.5
##  Median :3.890   Median : 143.0
##  Mean   :3.916   Mean   : 242.9
##  3rd Qu.:4.048   3rd Qu.: 306.2
##  Max.   :4.420   Max.   :1345.0
##
```

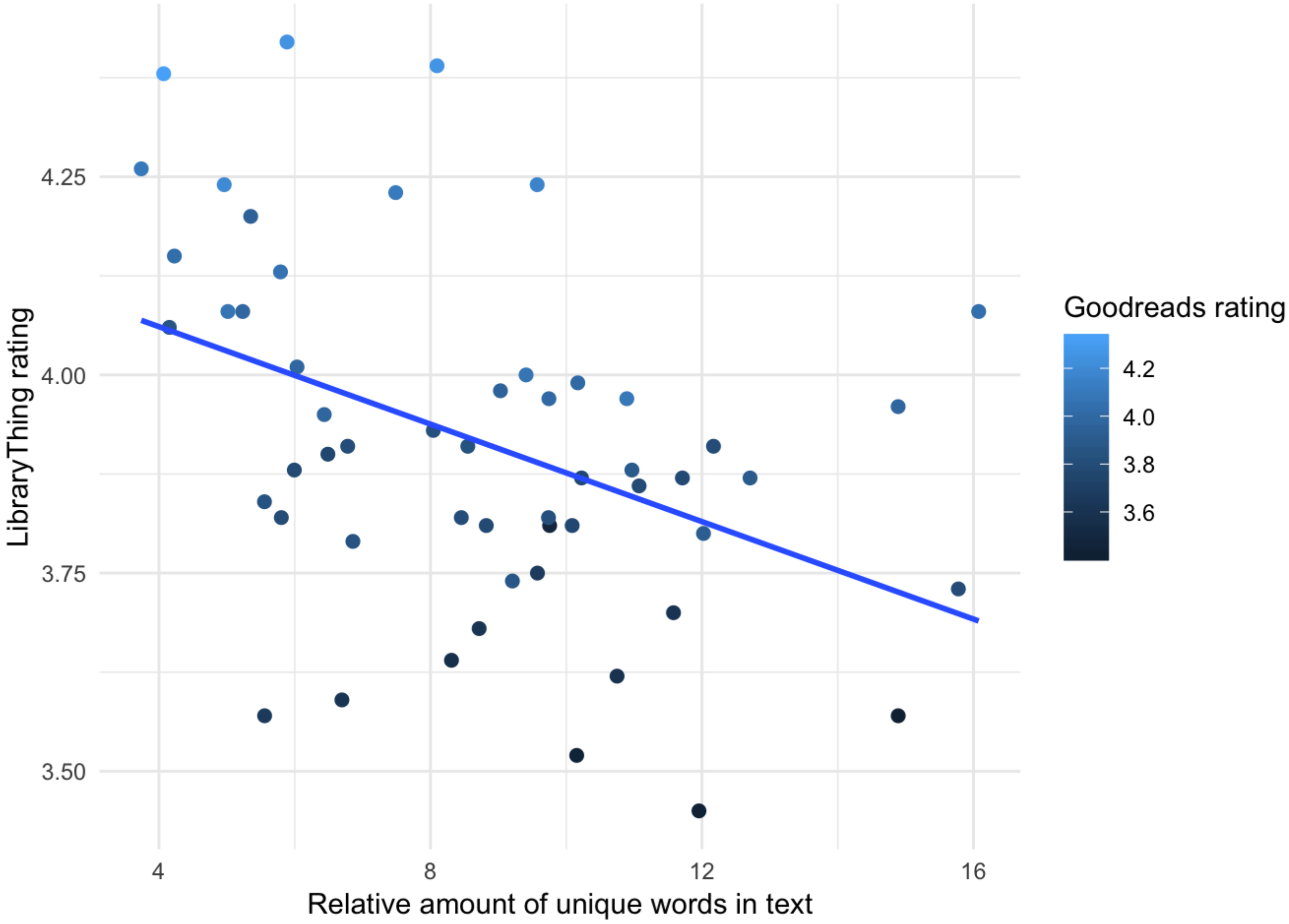
```
str(data_raw)
```

```
## 'data.frame':    54 obs. of  13 variables:
##  $ Work          : Factor w/ 54 levels "1984","AliceInWonderland",..: 23 18 28 5 3
1 27 44 48 24 20 ...
##  $ NoOfSen       : int  954 2417 14801 5233 2736 797 6335 1709 8101 9317 ...
##  $ NoOfWords     : int  26964 40120 323894 66229 80341 22385 115180 45495 140568 1
12907 ...
##  $ AvgWperS      : num  28.3 16.6 21.9 12.7 29.4 ...
##  $ UniqueW       : int  4254 5974 17332 9861 6792 2724 9325 5439 12253 10196 ...
##  $ UniqRatio     : num  15.78 14.89 5.35 14.89 8.45 ...
##  $ LongstSen     : int  129 98 174 258 235 144 178 125 175 118 ...
##  $ AvgWLen       : num  4.49 4.61 4.84 4.97 4.74 ...
##  $ FirstPublished: int  1886 1899 1871 1932 1880 1915 1960 1898 1900 1910 ...
##  $ NoOfRatings   : int  302590 348090 120371 1209004 259185 491629 3755275 76725 2
4647 66445 ...
##  $ GRrating      : num  3.8 3.42 3.95 3.98 3.81 3.8 4.26 3.44 3.62 3.96 ...
##  $ LTrating      : num  3.73 3.57 4.2 3.96 3.82 3.91 4.39 3.45 3.68 3.98 ...
##  $ LTpopularity  : int  314 147 283 20 185 745 12 1190 835 822 ...
```

```
Plot_1 = ggplot(data_raw,
          aes(x = UniqRatio, y = LTrating, colour = GRrating)) +
          geom_point(size = 2) + geom_smooth(method = "lm", se=F) + labs(colour = "Go
odreads rating", x = "Relative amount of unique words in text", y = "LibraryThing rat
ing") + theme_minimal()
```

As a first step in looking for interesting patterns in our dataframe, we plot some variables against one another. Here is the rating on LibraryThing as a function of the unique words ratio (which is not a good measure of vocabulary, we have to note).

```
Plot_1
```



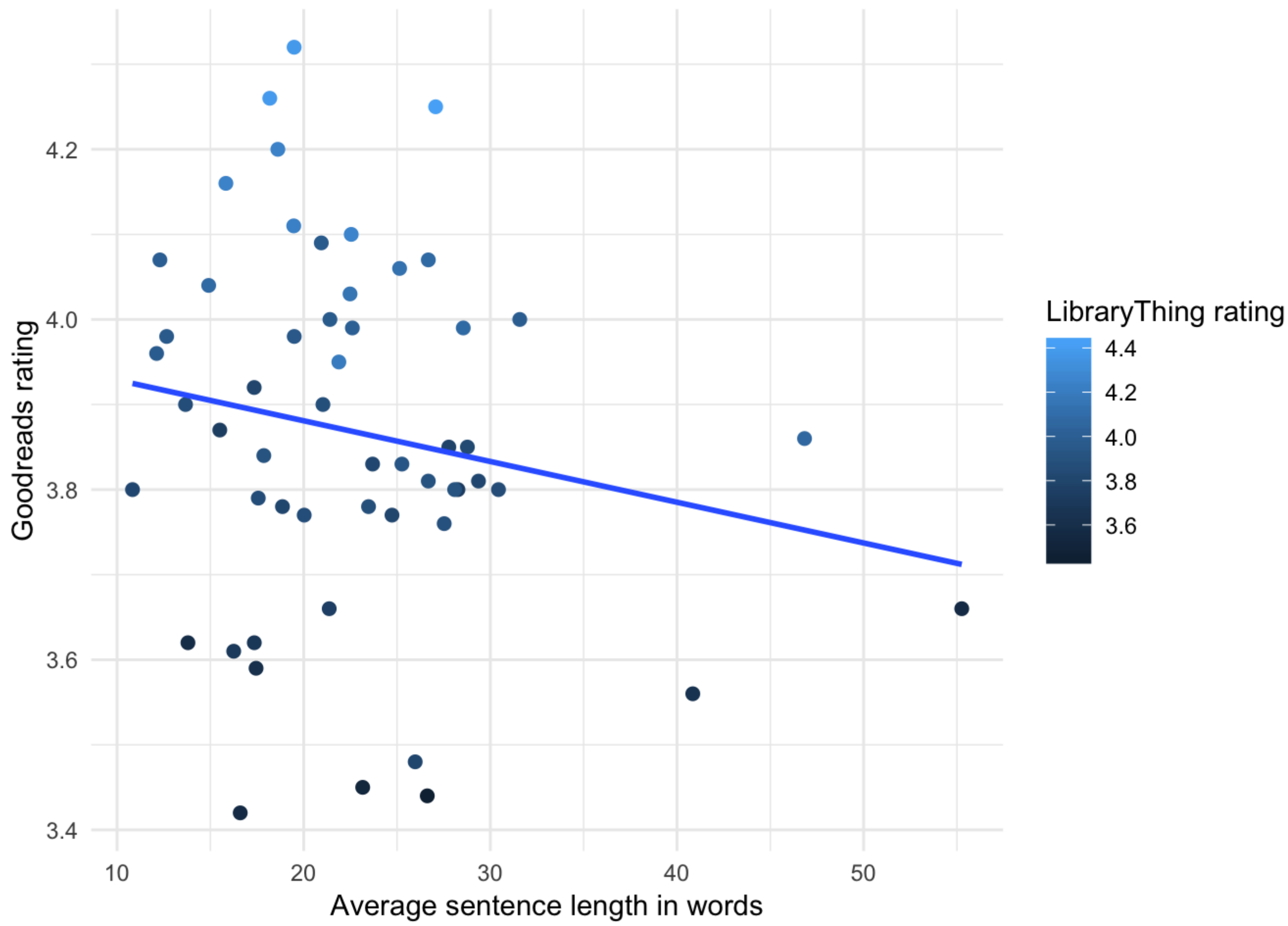

```
t.test(UniqRatio, LTrating)
```

```
##
##  Welch Two Sample t-test
##
## data:  UniqRatio and LTrating
## t = 11.454, df = 53.554, p-value = 5.088e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  3.968595 5.653094
## sample estimates:
## mean of x mean of y
##  8.726400  3.915556
```

```
summary(glm(LTrating~UniqRatio))
```

```
##
## Call:
## glm(formula = LTrating ~ UniqRatio)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.44302  -0.12575   0.00217   0.10412   0.45506
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.183852   0.083892  49.872  < 2e-16 ***
## UniqRatio   -0.030745   0.009075  -3.388  0.00135 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.0413681)
##
##     Null deviance: 2.6259  on 53  degrees of freedom
## Residual deviance: 2.1511  on 52  degrees of freedom
## AIC: -14.796
##
## Number of Fisher Scoring iterations: 2
```

Let's see the interaction between average sentence length and score on Goodreads:

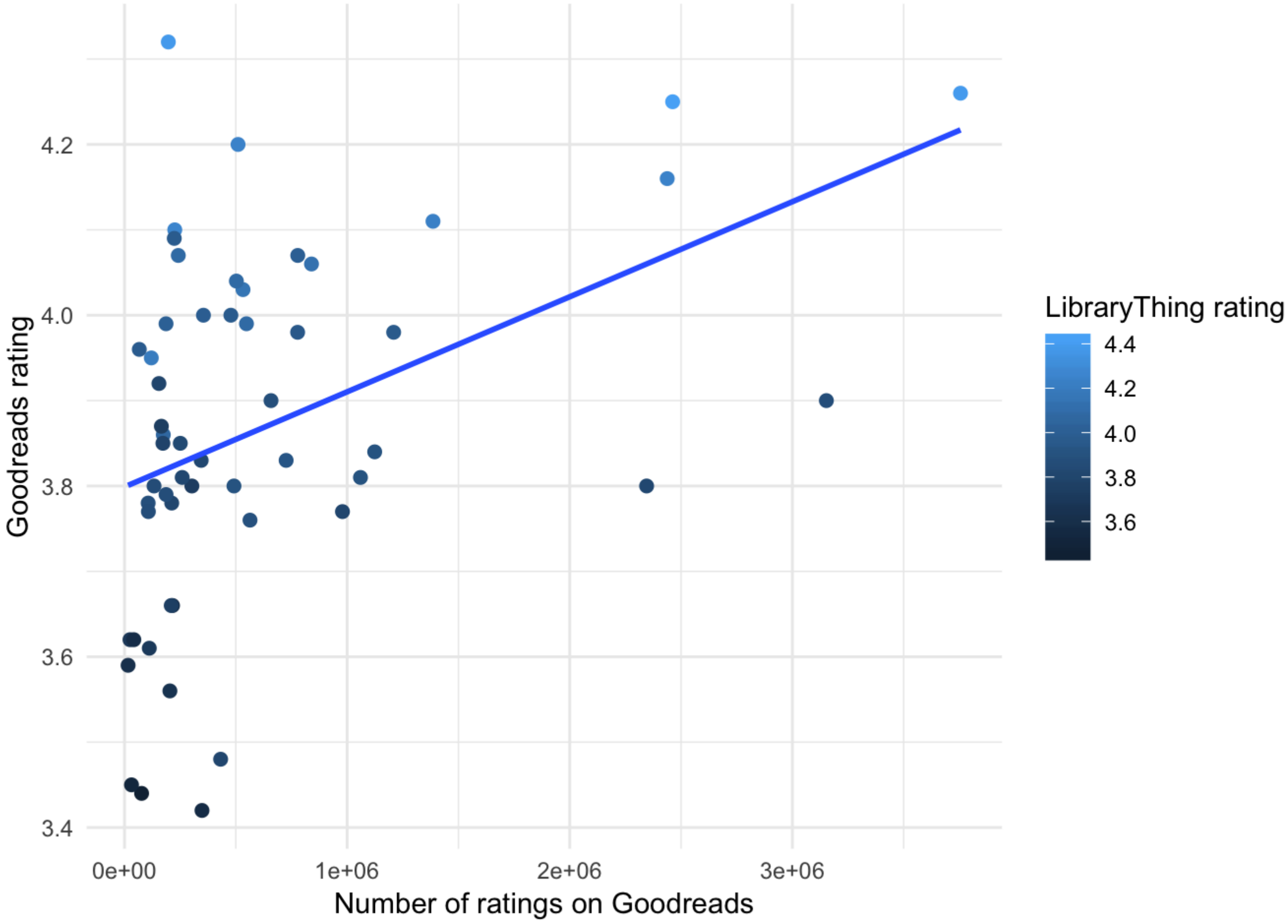


```
##
## Call:
## glm(formula = GRrating ~ AvgWperS)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.47718  -0.09898  -0.01846   0.14621   0.43664
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.976654   0.085806  46.345   <2e-16 ***
## AvgWperS    -0.004788   0.003545  -1.351    0.183
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.04430041)
##
##     Null deviance: 2.3844  on 53  degrees of freedom
## Residual deviance: 2.3036  on 52  degrees of freedom
## AIC: -11.098
##
## Number of Fisher Scoring iterations: 2
```

Perhaps it would be better to first have a look at how the number of ratings may affect the ratings themselves, and also to see if the length of a text is indicative of the amount of ratings, i.e. on the amount of people who actually finish the book:

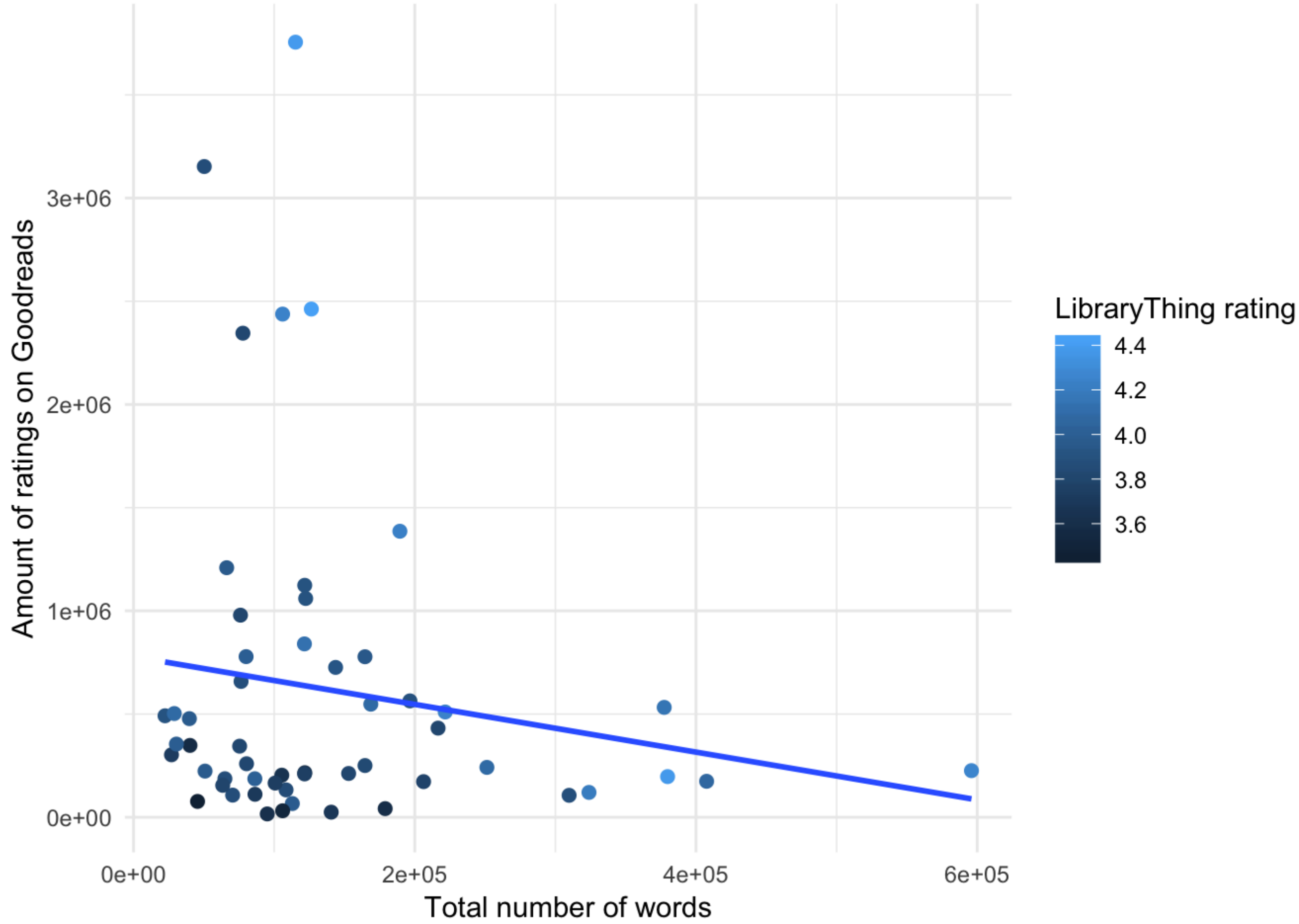
It seems that the more people give a rating, the higher the rating gets. This might be explained in that people perhaps are sensitive to the rating the book already has when they are about to rate it, and are more likely to give a rating, especially a good one, if the book is already rated highly.

```
Plot_3
```



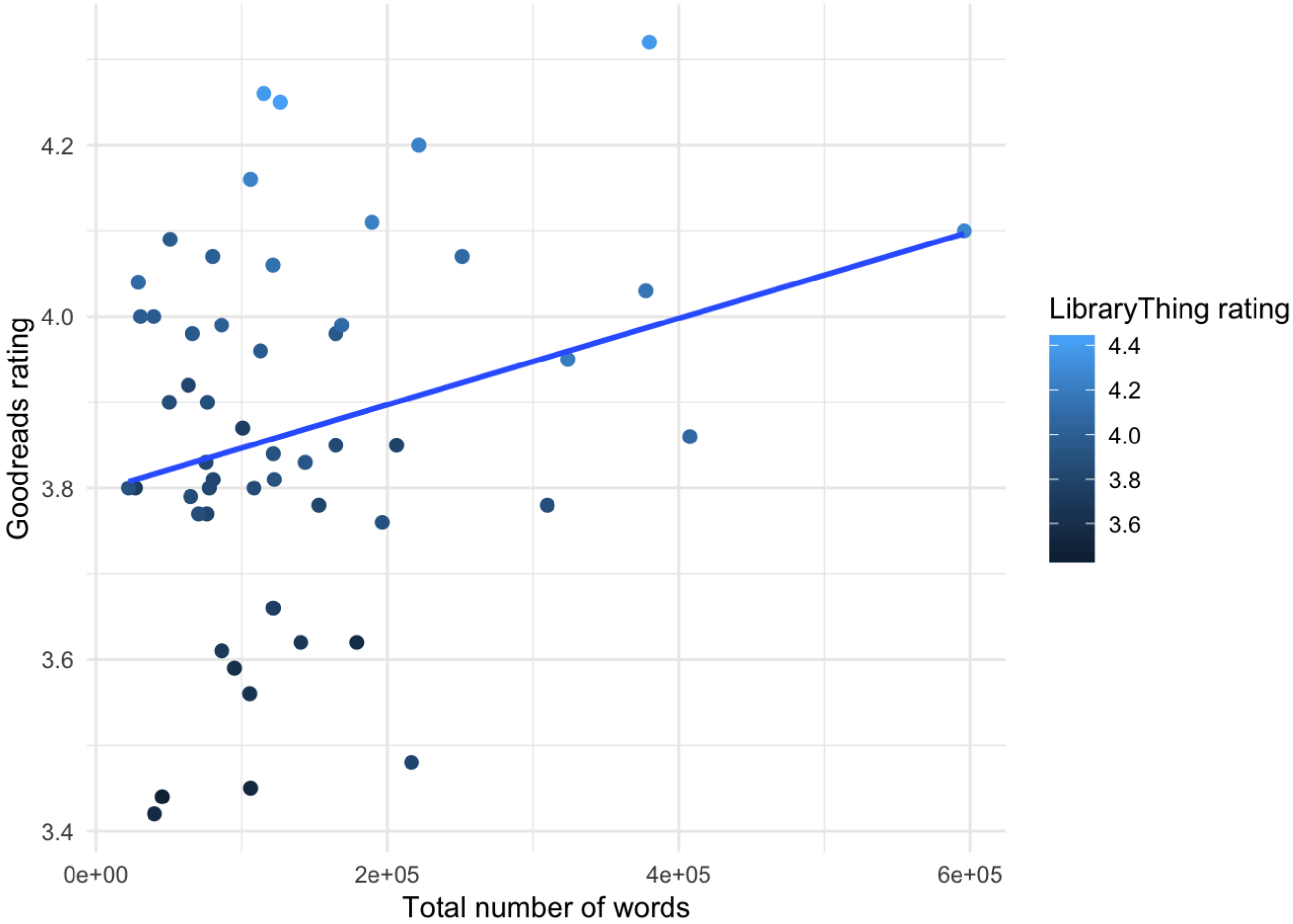
There also seems to be some effect of length on the amount of ratings, in that the longer a work is, the less ratings it receives. The scales are different, I think because of the varying nature of the values between works.

```
Plot_4
```



Yet, here we see a trend of longer works generally receiving a higher rating.

```
Plot_5
```



So long works are rated less, but higher. Short works are rated more, but lower. However, more ratings also indicates a higher score, and this would mean that short works, because they receive more ratings, should have a higher rating than long works, but the opposite is the case. Perhaps this small dataset is not useful for making such claims, but it nevertheless is apparent, be it that the sample is small.

These visualisations may be interesting, but we wanted to take it a step further and see whether we could predict user ratings based on the text-internal values we've extracted with python. R's neuralnet package provides a means of training a simple neural network for making predictions. This can then be compared with the predictions of a generalized linear model. Let me stress that I am not at all an expert when it comes to neural networks; I've come across them in another course, and was eager to apply the little knowledge I have of them to this experimental project.

We make use of the dplyr package, and the afore mentioned neuralnet package.

A pre-analysis of the variables gave us an indication of which ones tend to have the strongest predictive power, these were:

Average words per sentence Relative amount (%) of unique words in the text Total amount of unique words

Using dplyr, we make a new dataframe:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
neural_data = select(data_raw, AvgWperS, UniqRatio, UniqueW, GRrating)
```

Now we set a sample size, which will be 0.75. So 75% of the data we will use to train the network, the rest will be used for testing:

```
samplesize = 0.75*nrow(neural_data)

set.seed(1)
neural_index = sample(seq_len(nrow(neural_data)), size = samplesize)
training_data = neural_data[neural_index,]
testing_data = neural_data[-neural_index,]
```

Next we scale the data in order to be compatible between variables, and then we select the training and test set:

```
max = apply(neural_data, 2, max)
min = apply(neural_data, 2, min)

neural_df_scaled = as.data.frame(scale(neural_data, center = min, scale = max - min))



train_nn = neural_df_scaled[neural_index,]
test_nn = neural_df_scaled[-neural_index,]
```

A quick look at the sets:

```
head(train_nn)
```

```
##      AvgWperS  UniqRatio    UniqueW  GRrating
## 15 0.03302681 0.45953658 0.24573065 0.7222222
## 20 0.17526625 0.09905814 0.42294713 0.8666667
## 30 0.00000000 0.16725318 0.09142039 0.4222222
## 47 0.28463428 0.18278338 0.81005215 0.4000000
## 11 0.19489556 0.02676351 0.65093568 1.0000000
## 45 0.32488979 0.34875836 0.45152879 0.4555556
```

```
head(test_nn)
```

```
##     AvgWperS UniqRatio   UniqueW  GRrating
## 5  0.4172195 0.3821886 0.2079967 0.4333333
## 6  0.3884595 0.6833255 0.0000000 0.4222222
## 18 0.2296493 0.5859178 0.2893956 0.5333333
## 19 0.3814434 0.2527365 0.5838532 0.4777778
## 22 0.1583433 0.3899677 0.3928827 0.4666667
## 25 0.3567060 0.2465112 0.2851007 0.4333333
```

Now we can make our neural network:

```
set.seed(2)
attach(train_nn)
```

```
## The following objects are masked from data_raw:
##
##     AvgWperS, GRrating, UniqRatio, UniqueW
```

```
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

```
my_NN = neuralnet(GRrating ~ AvgWperS + UniqRatio + UniqueW, train_nn, hidden = 3, li
near.output = T, rep = 800)
```

Next we let the network predict the ratings of the test set, and we scale it back up for the numbers to make sense again:

```
predict_test_nn = neuralnet::compute(my_NN, test_nn[,c("AvgWperS", "UniqRatio", "Uniq
ueW")])

predict_test_nn = (predict_test_nn$net.result*(max(neural_data$GRrating) - min(neural
_data$GRrating))) + min(neural_data$GRrating)

predict_test_nn
```

```
##           [,1]
## 5  3.805103417
## 6  3.807429676
## 18 3.784023567
## 19 3.789162713
## 22 3.866330297
## 25 3.913455651
## 26 4.086885915
## 33 3.776953034
## 34 4.009441904
## 35 3.791983640
## 39 3.834674537
## 41 4.055562209
## 44 3.811807843
## 49 3.521556794
```

Is what we get. Now we can check back with our testing set we created earlier:

```
testing_data
```

```
##       AvgWperS    UniqRatio UniqueW GRrating
## 5  29.36440058  8.453964974    6792     3.81
## 6  28.08657465 12.168863078    2724     3.80
## 18 21.03053645 10.967218690    8384     3.90
## 19 27.77484514  6.857012644   14143     3.85
## 22 17.86236244  8.549929353   10408     3.84
## 25 26.67574635  6.780214843    8300     3.81
## 26 22.54564716  3.739192890   22282     4.10
## 33 24.73040650 10.087315572    7671     3.77
## 34 26.67696391  5.015181240   12603     4.07
## 35 17.44778859 10.748582668   10219     3.59
## 39 17.56517036 11.711004372    7607     3.79
## 41 22.60707733  6.033903022    5204     3.99
## 44 23.69158291  9.738314288    7346     3.83
## 49 30.43614931 11.072996201   12008     3.80
```

We can look at this and try to compare, but it is better to get some kind of measurement of how accurate it was. A good and easy way to operationalize accuracy is the mean squared error (MSE). This basically takes the error margin between the predicted value and the actual value, squares it (to prevent that positives and negatives cancel each other out), and then gives us the mean:

```
MSE_neuralnet <- sum((predict_test_nn - testing_data$GRrating)^2)/nrow(testing_data)

MSE_neuralnet
```

```
## [1] 0.01124771644
```

This is relatively low, so our model has done a good job, we may say. In order to asess whether it is really better than a linear model, we will compare the two:

```
lm_fit <- glm(GRrating ~ AvgWperS + UniqRatio + UniqueW, data=training_data)
summary(lm_fit)
```

```
##
## Call:
## glm(formula = GRrating ~ AvgWperS + UniqRatio + UniqueW, data = training_data)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -0.3948860  -0.1155299   0.0107101   0.1581974   0.3248531
##
## Coefficients:
##                    Estimate      Std. Error  t value
## (Intercept)  4.610251180827  0.219695499282 20.98473
## AvgWperS    -0.009692233041  0.003964092112 -2.44501
## UniqRatio   -0.040305846054  0.012729482936 -3.16634
## UniqueW     -0.000016966595  0.000009123007 -1.85976
##                           Pr(>|t|)
## (Intercept) < 0.000000000000000222 ***
## AvgWperS                0.0195088 *
## UniqRatio               0.0031387 **
## UniqueW                 0.0711065 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.04475949933)
##
##     Null deviance: 2.1616975  on 39  degrees of freedom
## Residual deviance: 1.6113420  on 36  degrees of freedom
## AIC: -4.9574012
##
## Number of Fisher Scoring iterations: 2
```

```
pr_lm <- predict(lm_fit, testing_data)
MSE_lm <- sum((pr_lm - testing_data$GRrating)^2)/nrow(testing_data)

MSE_lm
```

```
## [1] 0.014066016
```

```
MSE_neuralnet
```

```
## [1] 0.01124771644
```

It turns out that the linear model is very accurate too, up to the point where the two hardly differ in predictive “power”. The NN is still a bit better, but for a neural network the difference with a GLM is very low. This probably has to do with the fact that our dataset is very small, and for it to be accurate a NN needs a very large set of data to train on, which is not the case here. Still, the MSEs of the two models are low in general, which means that reader scores can actually be predicted with some accuracy, which is a cool finding for this project.