An Overview of the Linear Regression

Applied Machine Learning for Educational Data Science

true

10/12/2021

Contents

[Updated: Fri, Oct 22, 2021 - 16:56:38]

In the machine learning literature, the prediction algorithms are classified into two main categories: *supervised* and *unsupervised*. Supervised algorithms are being used when the dataset has an actual outcome of interest to predict (labels), and the goal is to build the "best" model predicting the outcome of interest that exists in the data. On the other side, unsupervised algorithms are being used when the dataset doesn't have an outcome of interest, and the goal is typically to identify similar groups of observations (rows of data) or similar groups of variables (columns of data) in data. In this course, we plan to cover a number of *supervised* algorithms. Linear regression is one of the simplest approach among supervised algorithms, and also one of the easiest to interpret.

Model Description

In most general terms, the linear regression model with P predictors (X_1,X_2,X_3,...,X_p) to predict an outcome (Y) can be written as the following:

$$Y = \beta_0 + \sum_{p=1}^{P} \beta_p X_p + \epsilon.$$

In this model, Y represents the observed value for the outcome for an observation, X_p represents the observed value of the p^{th} variable for the same observation, and β_p is the associated model parameter for the p^{th} variable. ϵ is the model error (residual) for the observation.

This model includes only the main effects of each predictor and can be easily extended by including a quadratic or higher-order polynomial terms for all (or a specific subset of) predictors. For instance, the model below includes all first-order, second-order, and third-order polynomial terms for all predictors.

$$Y = \beta_0 + \sum_{p=1}^{P} \beta_p X_p + \sum_{k=1}^{P} \beta_{k+P} X_k^2 + \sum_{m=1}^{P} \beta_{m+2P} X_m^3 + \epsilon.$$

The simple first-order, second-order, and third-order polynomial terms can also be replaced by corresponding terms obtained from B-splines or natural splines.

Sometimes, the effect of predictor variables on the outcome variable are not additive, and the effect of one predictor on the response variable can depend on the levels of another predictor. These non-additive effects are also called interaction effects. The interaction effects can also be a first-order interaction (interaction between two variables, e.g., $X_1 * X_2$), second-order interaction ($X_1 * X_2 * X_3$), or higher orders. It is also possible to add the interaction effects to the model. For instance, the model below also adds the first-order interactions.

$$Y = \beta_0 + \sum_{p=1}^{P} \beta_p X_p + \sum_{k=1}^{P} \beta_{k+P} X_k^2 + \sum_{m=1}^{P} \beta_{m+2P} X_m^3 + \sum_{i=1}^{P} \sum_{j=i+1}^{P} \beta_{i,j} X_i X_j + \epsilon.$$

If you are not comfortable or confused with notational representation, below is an example for different models you can write with 5 predictors (X_1, X_2, X_3).

A model with only main-effects:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon.$$

A model with polynomial terms up to the 3rd degree added:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_1^2 + \beta_5 X_2^2 + \beta_6 X_3^2 + \beta_7 X_1^3 + \beta_8 X_2^3 + \beta_9 X_3^3$$

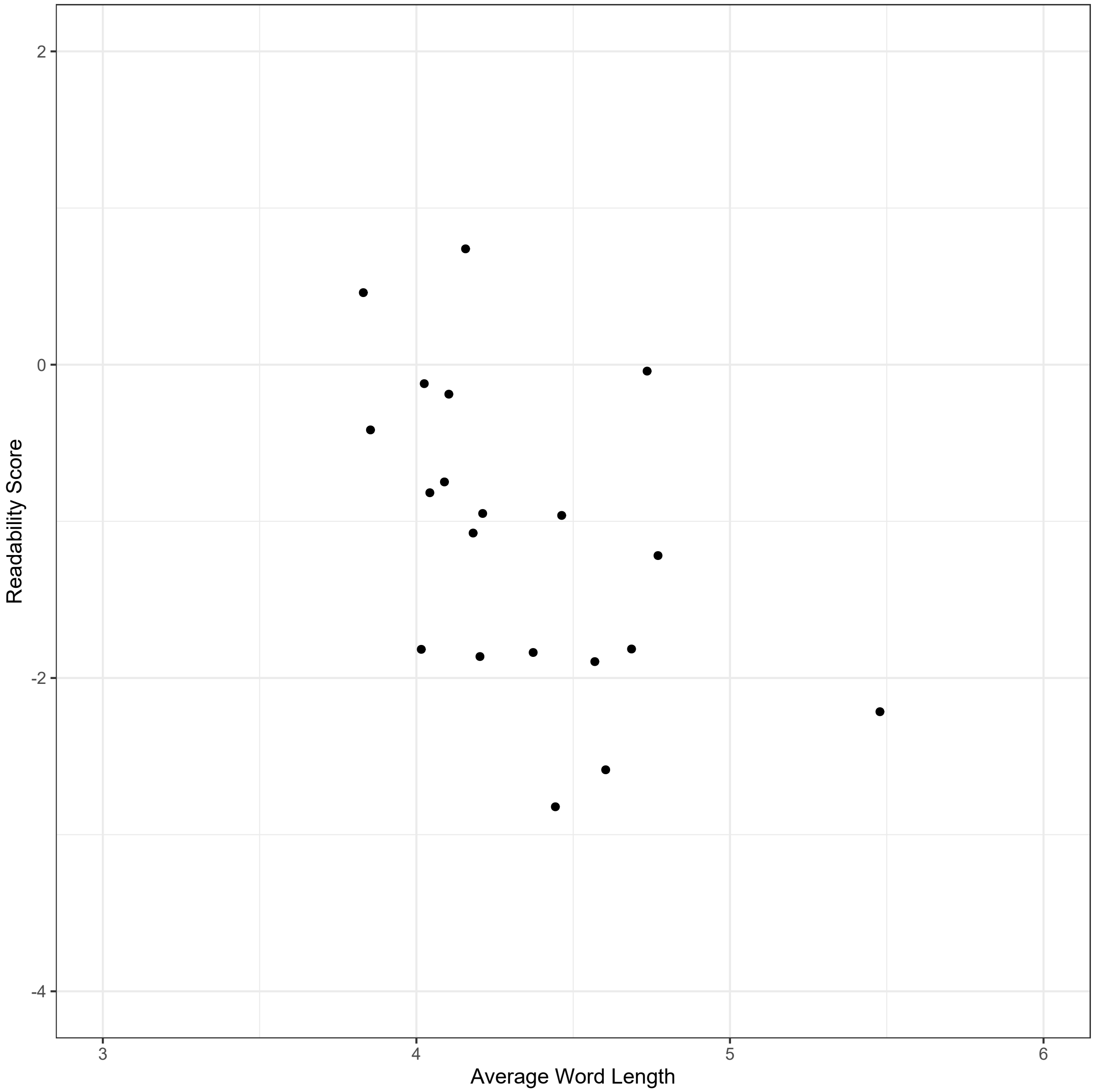
A model with both interaction terms and polynomial terms up to the 3rd degree added:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_1^2 + \beta_5 X_2^2 + \beta_6 X_3^2 + \beta_7 X_1^3 + \beta_8 X_2^3 + \beta_9 X_3^3 + \beta_{1,2} X_1 X_2 + \beta_{1,3} X_1 X_3 + \beta_{2,3} X_2 X_3 + \epsilon$$

Model Estimation

Suppose that we would like to predict the target readability score for a given text from average word length in the text. Below is a scatterplot to show the relationship between these two variables for a random sample of 20 observations. There seems to be a moderate negative correlation. So, we can tell that the higher the average word length is in a given text, the lower the readability score (more difficult to read).

```r
readability_sub <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-202
```

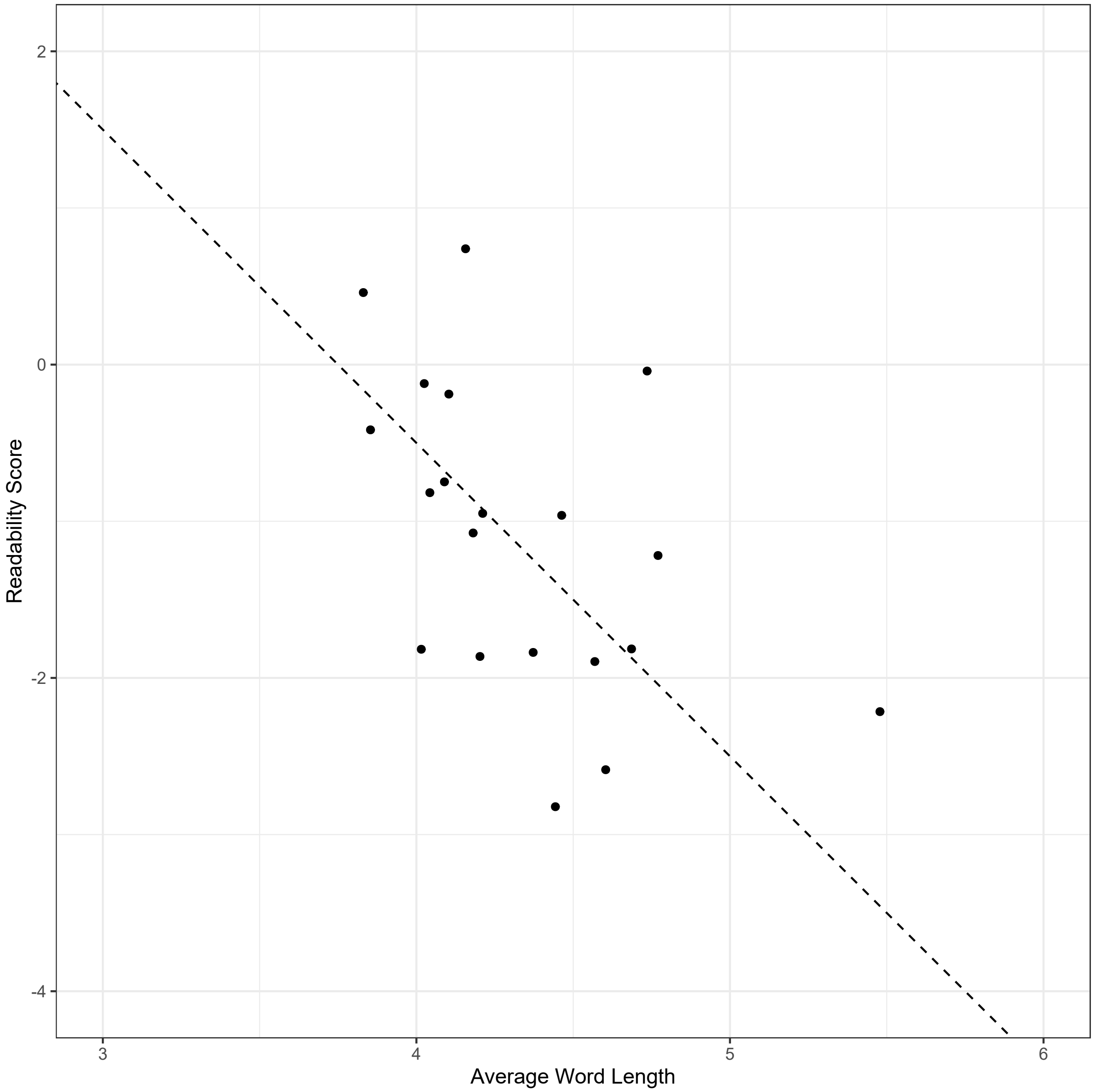


Let's consider a simple linear regression model such that the readability score is the outcome (Y) and average word length is the predictor(X). Our regression model would be

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

In this case, the set of coefficients, $\{\beta_0, \beta_1\}$, represents a linear line. We can come up with any set of $\{\beta_0, \beta_1\}$ coefficients and use it as our model. For instance, suppose I guesstimate that these coefficients are $\{\beta_0, \beta_1\}$ = {7.5,-2}. Then, my model would be

$$Y = 7.5 - 2X + \epsilon.$$



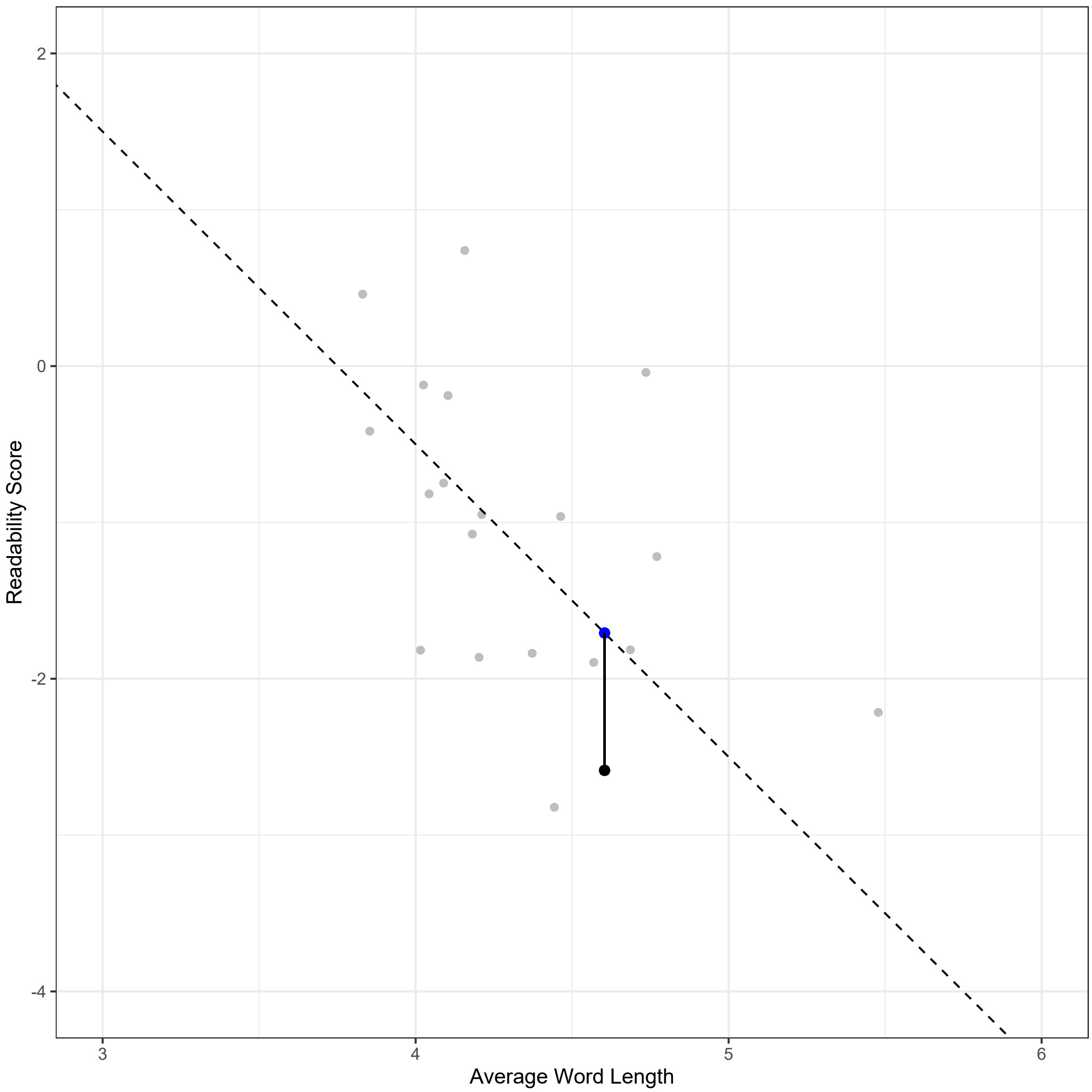
Using this model, I can predict the target readability score for any observation in my dataset. For instance, the average word length is 4.604 for the first reading passage. Then, my prediction of readability score based on this model would be -1.708. On the other side, the observed value of the readbility score for this observation is -2.586. This discrepancy between the observed value and the model prediction is the model error (residual) for the first observation and captured in the ϵ term in the model.

$$Y_{(1)} = 7.5 - 2X_{(1)} + \epsilon_{(1)}.$$

$$\hat{Y}_{(1)} = 7.5 - 2 * 4.604 = -1.708$$

$$\hat{\epsilon}_{(1)} = -2.586 - (-1.708) = -0.878$$

We can visualize this in the plot. The black dot represents the observed data point, and the blue dot on the line represents the model prediction for a given X value. The vertical distance between these two data points is the model error for this particular observation.

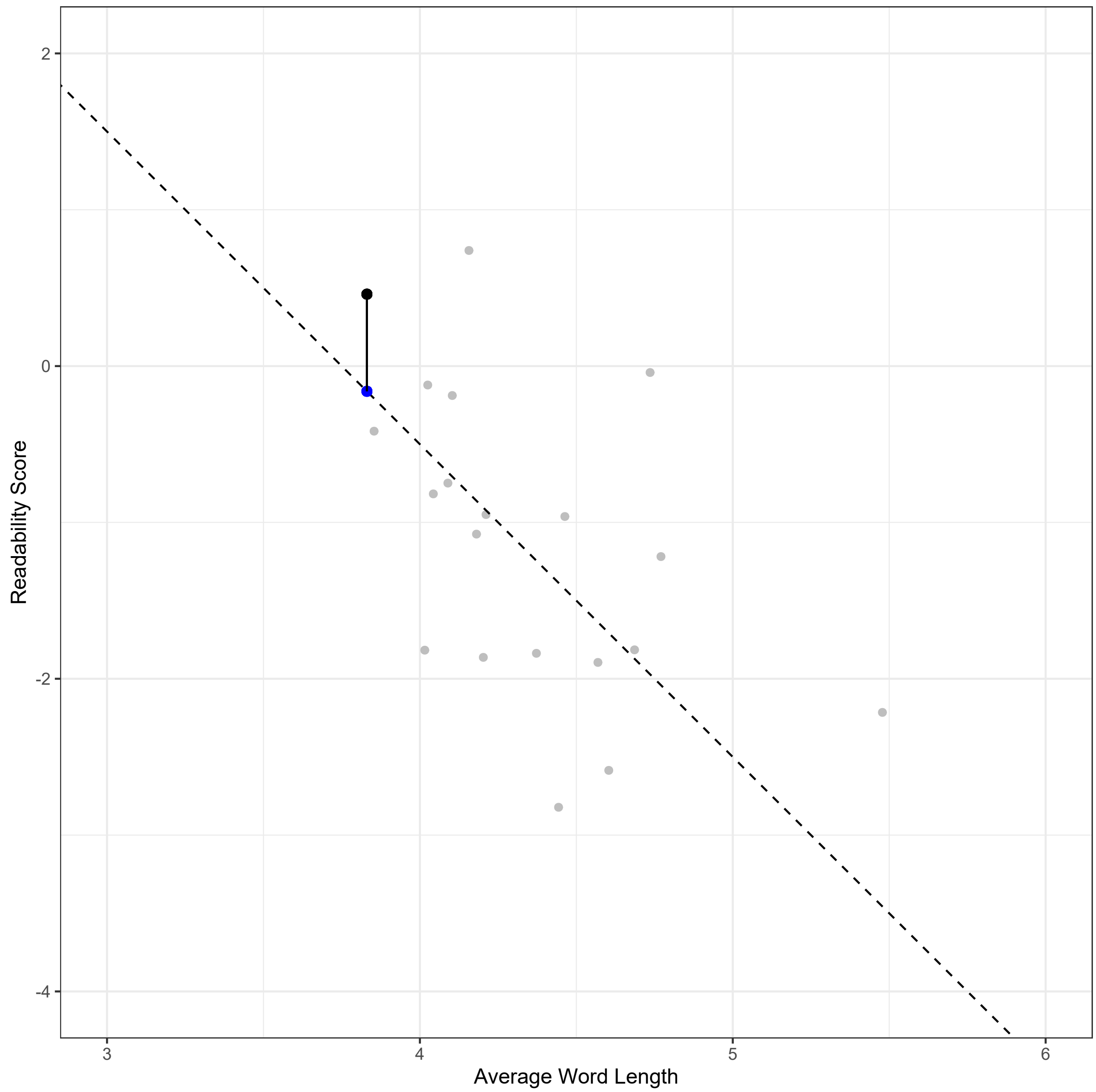


We can do the same thing for the second observation. The average word length is 3.830 for the second reading passage. The model predicts a readability score of be -0.161. Observed value of the readability score for this observation is 0.459. Therefore the model error for the second observation would be 0.62.

$$Y_{(2)} = 7.5 - 2X_{(2)} + \epsilon_{(2)}.$$

$$\hat{Y}_{(2)} = 7.5 - 2 * 3.830 = -0.161$$

$$\hat{\epsilon}_{(2)} = 0.459 - (-0.161) = 0.62$$



Using a similar approach, we can calculate the model error for every single observation.

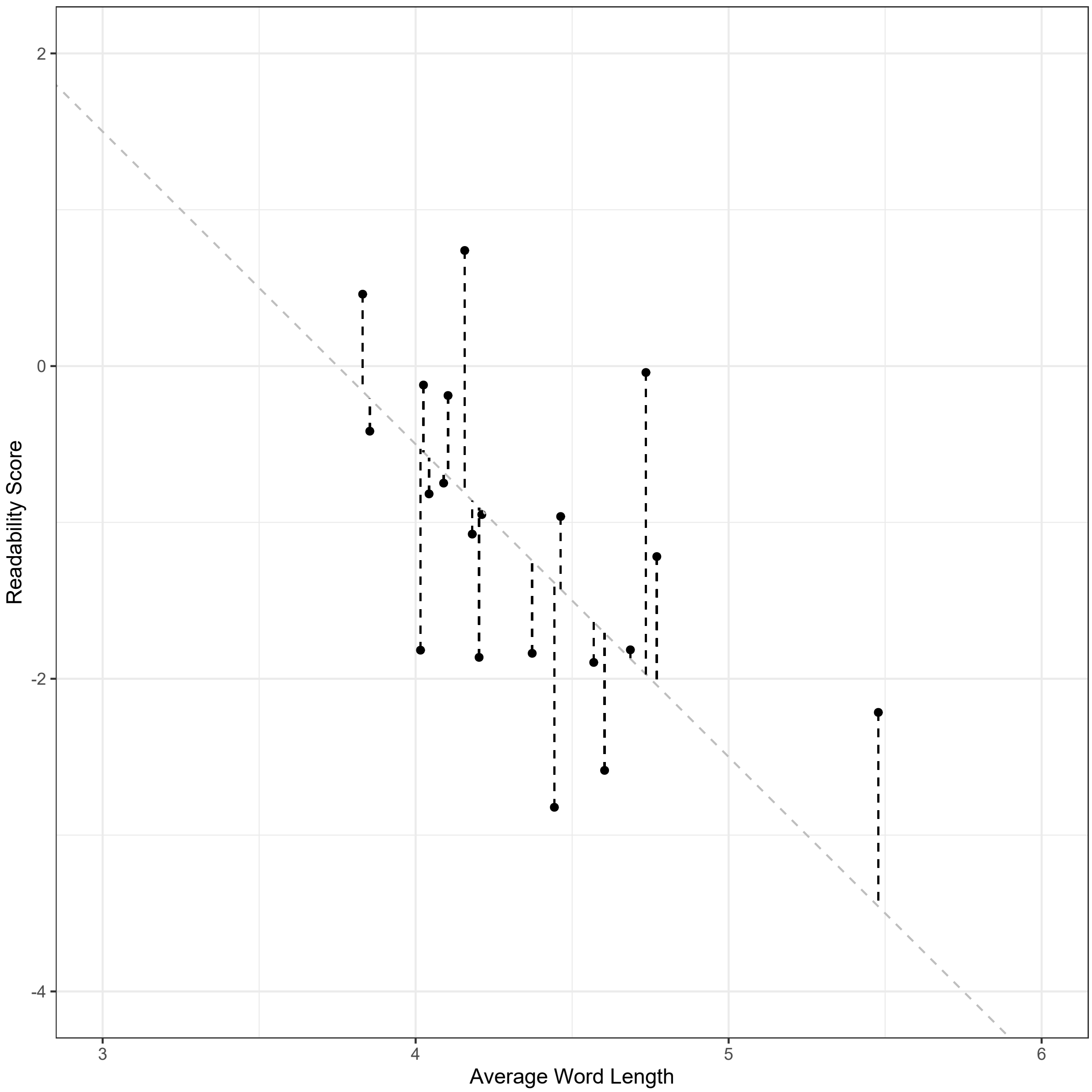
```
d <-  readability_sub[,c('mean.wl','target')]

d$predicted <- d$mean.wl*-2 + 7.5
d$error     <- d$target - d$predicted

d
```

```
    mean.wl       target  predicted       error
1  4.603659 -2.58590836 -1.7073171 -0.87859129
2  3.830688  0.45993224 -0.1613757  0.62130790
3  4.180851 -1.07470758 -0.8617021 -0.21300545
```

4	4.015544	-1.81700402	-0.5310881	-1.28591594
5	4.686047	-1.81491744	-1.8720930	0.05717559
6	4.211340	-0.94968236	-0.9226804	-0.02700194
7	4.025000	-0.12103065	-0.5500000	0.42896935
8	4.443182	-2.82200582	-1.3863636	-1.43564218
9	4.089385	-0.74845172	-0.6787709	-0.06968077
10	4.156757	0.73948755	-0.8135135	1.55300107
11	4.463277	-0.96218937	-1.4265537	0.46436430
12	5.478261	-2.21514888	-3.4565217	1.24137286
13	4.770492	-1.21845136	-2.0409836	0.82253224
14	4.568966	-1.89544351	-1.6379310	-0.25751247
15	4.735751	-0.04101056	-1.9715026	1.93049203
16	4.372340	-1.83716516	-1.2446809	-0.59248431
17	4.103448	-0.18818586	-0.7068966	0.51871069
18	4.042857	-0.81739314	-0.5857143	-0.23167886
19	4.202703	-1.86307557	-0.9054054	-0.95767016
20	3.853535	-0.41630158	-0.2070707	-0.20923088



While it is helpful to see the model error for every single observation, we will need to aggregate them in some way to form an overall measure of the total amount of error for this model. Some alternatives for aggregating these individual errors could be using

a. the sum of the residuals (SR),
b. the sum of absolute value of residuals (SAR), or
c. the sum of squared residuals (SSR)

Among these alternatives, (a) is not a useful aggregation as the positive residuals and negative residuals will cancel each other and (a) may misrepresent the total amount of error for all observations. Both (b) and (c) are plausible alternatives and can be used. On the other hand, (b) is less desirable because the absolute values are mathematically more difficult to deal with (ask a calculus professor!). So, (c) seems to be a good way of aggregating the total amount of error, and it is mathematically easier to work with. We can show (c) in a mathematical notation as the following.

$$SSR = \sum_{i=1}^{N}(Y_{(i)} - (\beta_0 + \beta_1 X_{(i)}))^2$$

$$SSR = \sum_{i=1}^{N}(Y_{(i)} - \hat{Y_{(i)}})^2$$

$$SSR = \sum_{i=1}^{N}\epsilon_{(i)}^2$$

For our model, the sum of squared residuals would be 15.384.

```
sum(d$error^2)
```

```
[1] 15.38364
```

Now, how do we know that the set of coefficients we guesstimate ,$\{\beta_0, \beta_1\} = \{7.5,\text{-}2\}$, is a good model? Is there any other set of coefficients that would provide less error than this model? The only way of knowing this is to try a bunch of different models and see if we can find a better one that gives us better predictions (smaller residuals). But, there is literally infinite pairs of $\{\beta_0, \beta_1\}$ coefficients, so which ones we should try?

Below, I will do a quick exploration. For instance, suppose the potential range for my intercept (β_0) is from -10 to 10 and I will consider every single possible value from -10 t 10 with increments of .1. Also, suppose the potential range for my slope (β_1) is from -2 to 2 and I will consider every single possible value from -2 to 2 with increments of .01. Given that every single combination of β_0 and β_1 indicates a different model, these settings suggest a total of 80,601 models to explore. If you are crazy enough, you can try every single model and compute the SSR. Then, we can plot them in a 3D by putting β_0 on the X-axis, β_1 on the Y-axis, and SSR on the Z-axis. Check the plot below and tell me if you can explore and find the minimum of this surface.

Finding the best set of $\{\beta_0, \beta_1\}$ coefficients that minimizes the sum of squared residuals is an optimization problem. For any optimization problem, there is a **loss function** we either try to minimize or maximize. In this case, our loss function is the sum of squared residuals.

$$Loss = \sum_{i=1}^{N} (Y_{(i)} - (\beta_0 + \beta_1 X_{(i)}))^2$$

In this loss function, X and Y values are observed data, and $\{\beta_0, \beta_1\}$ are unknown parameters. The goal of optimization is to find the set $\{\beta_0, \beta_1\}$ coefficients that provides the minimum value of this function. Once this minima of this function is found, we can argue that the corresponding coefficients are our best solution for the regression model.

In this case, this is a good-looking surface with a single global minima, and it is not difficult to find the minimum of this loss function. We also have an analytical solution to find its minima because of its simplicity. Most of the time, the optimization problems are more difficult, and we solve them using numerical techniques such as steepest ascent (or descent), newton-raphson, quasi-newton, genetic algorithm and many more.

Matrix Solution

For most regression problems, we can find the best set of coefficients with a simple matrix operation. Let's first see how we can represent the regression problem in matrix form. Suppose that I wrote the regression model presented in the earlier section for every single observation in a dataset with a sample size of N.

$$Y_{(1)} = \beta_0 + \beta_1 X_{(1)} + \epsilon_{(1)}.$$

$$Y_{(2)} = \beta_0 + \beta_1 X_{(2)} + \epsilon_{(2)}.$$
$$Y_{(3)} = \beta_0 + \beta_1 X_{(3)} + \epsilon_{(3)}.$$
$$...$$
$$...$$
$$...$$
$$Y_{(20)} = \beta_0 + \beta_1 X_{(20)} + \epsilon_{(20)}.$$

We can write all of these equations in a much simpler format as

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

such that $\mathbf{Y}$ is an N x 1 column vector of observed values for the outcome variable, $\mathbf{X}$ is an N x (P+1) **design matrix* for the set of predictor variables including an intercept term, and $\boldsymbol{\beta}$ is an (P+1) x 1 column vector of regression coefficients, and $\boldsymbol{\epsilon}$ is an N x 1 column vector of residuals. For the problem above with our small dataset, these matrix elements would look like the following.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \\ Y_9 \\ Y_{10} \\ Y_{11} \\ Y_{12} \\ Y_{13} \\ Y_{14} \\ Y_{15} \\ Y_{16} \\ Y_{17} \\ Y_{18} \\ Y_{19} \\ Y_{20} \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ 1 & X_3 \\ 1 & X_4 \\ 1 & X_5 \\ 1 & X_6 \\ 1 & X_7 \\ 1 & X_8 \\ 1 & X_9 \\ 1 & X_{10} \\ 1 & X_{11} \\ 1 & X_{12} \\ 1 & X_{13} \\ 1 & X_{14} \\ 1 & X_{15} \\ 1 & X_{16} \\ 1 & X_{17} \\ 1 & X_{18} \\ 1 & X_{19} \\ 1 & X_{20} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \\ \epsilon_8 \\ \epsilon_9 \\ \epsilon_{10} \\ \epsilon_{11} \\ \epsilon_{12} \\ \epsilon_{13} \\ \epsilon_{14} \\ \epsilon_{15} \\ \epsilon_{16} \\ \epsilon_{17} \\ \epsilon_{18} \\ \epsilon_{19} \\ \epsilon_{20} \end{bmatrix}$$

Or, more specifically, we can replace the observed values of $\mathbf{X}$ and $\mathbf{Y}$ with the corresponding elements.

$$
\begin{bmatrix} -2.59 \\ 0.46 \\ -1.07 \\ -1.82 \\ -1.81 \\ -0.95 \\ -0.12 \\ -2.82 \\ -0.75 \\ 0.74 \\ -0.96 \\ -2.22 \\ -1.22 \\ -1.90 \\ -0.04 \\ -1.84 \\ -0.19 \\ -0.82 \\ -1.86 \\ -0.42 \end{bmatrix} = \begin{bmatrix} 1 & 4.60 \\ 1 & 3.83 \\ 1 & 4.18 \\ 1 & 4.02 \\ 1 & 4.69 \\ 1 & 4.21 \\ 1 & 4.03 \\ 1 & 4.44 \\ 1 & 4.09 \\ 1 & 4.16 \\ 1 & 4.46 \\ 1 & 5.48 \\ 1 & 4.77 \\ 1 & 4.57 \\ 1 & 4.74 \\ 1 & 4.37 \\ 1 & 4.10 \\ 1 & 4.04 \\ 1 & 4.20 \\ 1 & 3.85 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \\ \epsilon_8 \\ \epsilon_9 \\ \epsilon_{10} \\ \epsilon_{11} \\ \epsilon_{12} \\ \epsilon_{13} \\ \epsilon_{14} \\ \epsilon_{15} \\ \epsilon_{16} \\ \epsilon_{17} \\ \epsilon_{18} \\ \epsilon_{19} \\ \epsilon_{20} \end{bmatrix}
$$

It can be shown that the set of $\{\beta_0, \beta_1\}$ coefficients that yields the minimum sum of squared residuals for this model can be analytically found using the following matrix operation.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X^T X})^{-1}\mathbf{X^T Y}$$

If we apply this matrix operation to our small datasets, we will find that the best set of $\{\beta_0, \beta_1\}$ coefficients to predict the readability score with the least amount of error using the average word length as a predictor is $\{\beta_0, \beta_1\}$ = {4.494,-1.290}. These estimates are also known as the **least square estimates**, and the best linear unbiased estimators (BLUE) for the given regression model.

```
Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(1,readability_sub$mean.wl))

Y
```

```
             [,1]
 [1,] -2.58590836
 [2,]  0.45993224
 [3,] -1.07470758
 [4,] -1.81700402
 [5,] -1.81491744
 [6,] -0.94968236
 [7,] -0.12103065
 [8,] -2.82200582
```

```
 [9,] -0.74845172
[10,]  0.73948755
[11,] -0.96218937
[12,] -2.21514888
[13,] -1.21845136
[14,] -1.89544351
[15,] -0.04101056
[16,] -1.83716516
[17,] -0.18818586
[18,] -0.81739314
[19,] -1.86307557
[20,] -0.41630158
```

```r
X
```

```
      [,1]     [,2]
 [1,]    1 4.603659
 [2,]    1 3.830688
 [3,]    1 4.180851
 [4,]    1 4.015544
 [5,]    1 4.686047
 [6,]    1 4.211340
 [7,]    1 4.025000
 [8,]    1 4.443182
 [9,]    1 4.089385
[10,]    1 4.156757
[11,]    1 4.463277
[12,]    1 5.478261
[13,]    1 4.770492
[14,]    1 4.568966
[15,]    1 4.735751
[16,]    1 4.372340
[17,]    1 4.103448
[18,]    1 4.042857
[19,]    1 4.202703
[20,]    1 3.853535
```

```r
beta <- solve(t(X)%*%X)%*%t(X)%*%Y

beta
```

```
          [,1]
[1,]  4.493847
[2,] -1.290571
```

Once we find the best estimates for the model coefficients, we can also calculate the model predicted values and residual sum of squares for the given model and dataset.

$$\hat{\boldsymbol{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$

$$\hat{\boldsymbol{\epsilon}} = \boldsymbol{Y} - \hat{\boldsymbol{Y}}$$
$$RSS = \hat{\boldsymbol{\epsilon}}^{\boldsymbol{T}}\hat{\boldsymbol{\epsilon}}$$

```r
Y_hat <-  X%*%beta

Y_hat
```

```
            [,1]
 [1,] -1.4475035
 [2,] -0.4499296
 [3,] -0.9018403
 [4,] -0.6884998
 [5,] -1.5538311
 [6,] -0.9411887
 [7,] -0.7007034
 [8,] -1.2403969
 [9,] -0.7837974
[10,] -0.8707449
[11,] -1.2663309
[12,] -2.5762403
[13,] -1.6628138
[14,] -1.4027297
[15,] -1.6179787
[16,] -1.1489710
[17,] -0.8019465
[18,] -0.7237493
[19,] -0.9300414
[20,] -0.4794160
```

```r
E <- Y - Y_hat
E
```

```
              [,1]
 [1,] -1.138404820
 [2,]  0.909861867
 [3,] -0.172867283
 [4,] -1.128504242
 [5,] -0.261086332
 [6,] -0.008493645
 [7,]  0.579672713
 [8,] -1.581608945
 [9,]  0.035345700
[10,]  1.610232426
[11,]  0.304141555
[12,]  0.361091438
[13,]  0.444362421
[14,] -0.492713788
[15,]  1.576968115
[16,] -0.688194163
[17,]  0.613760605
[18,] -0.093643860
[19,] -0.933034170
[20,]  0.063114409
```

```
RSS <- t(E)%*%E

RSS
```

```
         [,1]
[1,] 13.81062
```

Note that the matrix formulation is generalized to a regression model for more than one predictor. When there are more predictors in the model, the dimensions of the design matrix ($\mathbf{X}$) and regression coefficient matrix ($\boldsymbol{\beta}$) will be different, but the matrix calculations will be identical. It is difficult to visualize the surface we are trying to minimize beyond two coefficients, but we know that the matrix solution will always provide us the set of coefficients that yields the least amount of error in our predictions.

Let's assume that we would like to expand our model by adding the number of sentences as the second predictor. Our new model will be

$$Y_{(i)} = \beta_0 + \beta_1 X_{1(i)} + \beta_2 X_{2(i)} + \epsilon_{(i)}.$$

Note that I added a subscript for X to differentiate different predictors. Let's say X_1 represents the mean word length and X_2 represents the total number of sentence length. Now, we are looking for the best set of three coefficients, $\{\beta_0, \beta_1, \beta_2\}$ that would yield the least amount of error in predicting the readability. Now, our matrix elements will look like the following:

```
Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(1,readability_sub[,c('mean.wl','sents')]))

Y
```

```
             [,1]
 [1,] -2.58590836
 [2,]  0.45993224
 [3,] -1.07470758
 [4,] -1.81700402
 [5,] -1.81491744
 [6,] -0.94968236
 [7,] -0.12103065
 [8,] -2.82200582
 [9,] -0.74845172
[10,]  0.73948755
[11,] -0.96218937
[12,] -2.21514888
[13,] -1.21845136
[14,] -1.89544351
[15,] -0.04101056
[16,] -1.83716516
[17,] -0.18818586
[18,] -0.81739314
[19,] -1.86307557
[20,] -0.41630158
```

```
X
```

```
      1  mean.wl sents
 [1,] 1 4.603659     7
 [2,] 1 3.830688    23
 [3,] 1 4.180851    17
 [4,] 1 4.015544     7
 [5,] 1 4.686047     6
 [6,] 1 4.211340    18
 [7,] 1 4.025000    10
 [8,] 1 4.443182     4
 [9,] 1 4.089385     9
[10,] 1 4.156757    28
[11,] 1 4.463277    15
[12,] 1 5.478261    10
[13,] 1 4.770492    10
[14,] 1 4.568966     8
[15,] 1 4.735751    19
[16,] 1 4.372340    15
[17,] 1 4.103448     6
[18,] 1 4.042857     6
[19,] 1 4.202703     7
[20,] 1 3.853535    19
```

We will get the following estimates for $\{\beta_0, \beta_1, \beta_2\}$ = {1.821, -.929, .090} yielding a value of 7.365 for the residual sum of squares.

```
beta <- solve(t(X)%*%X)%*%t(X)%*%Y

beta
```

```
               [,1]
1        1.82055156
mean.wl -0.92858249
sents    0.09029887
```

```
Y_hat <-  X%*%beta

E <- Y - Y_hat

RSS <- t(E)%*%E

RSS
```

```
         [,1]
[1,] 7.365244
```

`lm()` function

While it is always exciting to learn the inner mechanics of how numbers work behind the scene, it is handy to use already existing packages and tools to do all these computations. A simple go-to function for fitting linear regression to predict a continuous outcome is the `lm()` function.

Let's fit the models we talked about in earlier section using the `lm()` function and see if we get the same regression coefficients.

Model 1: Predicting readability scores from average word length

```
mod <- lm(target ~ 1 + mean.wl,data=readability_sub)

summary(mod)
```

```
Call:
lm(formula = target ~ 1 + mean.wl, data = readability_sub)

Residuals:
     Min       1Q   Median       3Q      Max
-1.58161 -0.54158  0.01343  0.47819  1.61023

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.4938     2.2387   2.007   0.0600 .
mean.wl      -1.2906     0.5137  -2.513   0.0217 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8759 on 18 degrees of freedom
Multiple R-squared:  0.2596,    Adjusted R-squared:  0.2185
F-statistic: 6.313 on 1 and 18 DF,  p-value: 0.02173
```

In the **Coefficients** table, the numbers under the **Estimate** column are the estimated regression coefficients, and they are identical to the numbers we obtained before using matrix calculations. We ignore the other numbers in this table since our focus in this class is not significance testing. Another number in this table is **Residual Standard Error (RSE)**, and this number is directly related to the Residual Sum of Squares (RSS) for this model. Note that we obtained a value of 13.811 for RSS when we fitted the model. The relationship between RSS and RSE is

$$RSE = \sqrt{\frac{RSS}{df_{regression}}} = \sqrt{\frac{RSS}{N-k}},$$

where the degrees of freedom for the regression model in this case is equal to the difference between the number of observations (N) and the number of coefficients in the model (k).

$$RSE = \sqrt{\frac{13.811}{20-2}} = 0.8759$$

RSE is a measure that summarizes the amount of uncertainty for individual predictions. Another relavant number reported is the R-squared (0.2596) which is simply the square of the correlation between predicted values observed values.

Model 2: Predicting readability scores from average word length and number of sentences

```
mod <- lm(target ~ 1 + mean.wl + sents,data=readability_sub)

summary(mod)
```

```
Call:
lm(formula = target ~ 1 + mean.wl + sents, data = readability_sub)
```

```
Residuals:
     Min       1Q   Median       3Q      Max 
-0.95212 -0.49900  0.06346  0.43368  1.25986 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)   
(Intercept)  1.82055    1.81947   1.001  0.33105   
mean.wl     -0.92858    0.39723  -2.338  0.03189 * 
sents        0.09030    0.02341   3.857  0.00126 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6582 on 17 degrees of freedom
Multiple R-squared:  0.6052,	Adjusted R-squared:  0.5587 
F-statistic: 13.03 on 2 and 17 DF,  p-value: 0.0003711
```

Building a Prediction Model for Readability Scores

In earlier weeks, we discussed how to process text data and constructed more than 1000 features for a given text. All these features were numeric features. These features are saved as a separate dataset, and can be downloaded from the website.

```
readability <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/ma
```

This dataset has 2834 rows and 1072 columns. Each row represents a reading passage. The last column is the readability score, the outcome variable to predict (`target`), and the first 1071 columns are numerical features we can potentially use as predictors.

Initial Data Preparation

We will first do some initial exploration of the variables. First, we will look at the percentage of missing values. Particularly, I will look for any feature with more than 80% of values are missing. Then, I will remove those features from the data.

```
require(finalfit)

missing_ <- ff_glimpse(readability)$Continuous

head(missing_)
```

```
          label var_type    n missing_n missing_percent  mean    sd   min
chars     chars    <int> 2834         0             0.0 972.6 117.4 669.0
sents     sents    <int> 2834         0             0.0   9.5   4.6   2.0
tokens   tokens    <int> 2834         0             0.0 172.8  17.1 113.0
types     types    <int> 2834         0             0.0 104.8  13.1  37.0
puncts   puncts    <int> 2834         0             0.0   0.0   0.0   0.0
numbers numbers    <int> 2834         0             0.0   0.0   0.0   0.0
        quartile_25 median quartile_75    max
chars         886.0  972.0      1059.0 1343.0
sents           7.0    8.0        11.0   41.0
```

```
tokens        159.0  174.0       187.0  208.0
types          96.0  105.0       114.0  143.0
puncts          0.0    0.0         0.0    0.0
numbers         0.0    0.0         0.0    0.0
```

```
# Because there is more than 1000 variables, it is not practical to print them all
# I filter the ones with missing data, and then print

flag_na <- which(as.numeric(missing_$missing_percent) > 80)
flag_na
```

```
 [1]  155  178  959  964  970  972  984  993  994  995  998  999 1001 1003 1004
[16] 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019
[31] 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034
[46] 1035 1036 1037 1038 1039 1040 1041 1042 1044 1045 1046 1047 1048 1049 1050
[61] 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065
[76] 1066 1067 1068 1069 1070 1071
```

```
# Remove the flagged variables with high missing data percentages

readability <- readability[,-flag_na]
```

Then, I will use the `recipes` package to create a recipe for this dataset. Note that all my features are numeric, and the last column is outcome variable while every other column is a predictor variable. This recipe

- assigns the last column (`target`) as outcome and everything else as predictors,
- removes any variable with zero variance or near-zero variance,
- impute the missing values using the mean,
- standardize all variables,
- and removes variables highly correlated with one another (>.9).

```
require(recipes)

blueprint <- recipe(x     = readability,
                    vars  = colnames(readability),
                    roles = c(rep('predictor',990),'outcome')) %>%
  step_zv(all_numeric()) %>%
  step_nzv(all_numeric()) %>%
  step_impute_mean(all_numeric()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_corr(all_numeric(),threshold=0.9)
```

Train/Test Split

In order to obtain a realistic measure of model performance, we will split the data into two subsamples: training and test datasets. Due to the relatively small sample size, I will use a 90-10 split (typically a 80-20 or 70-30 split is used).

```r
set.seed(10152021)  # for reproducibility

loc      <- sample(1:nrow(readability), round(nrow(readability) * 0.9))
read_tr  <- readability[loc, ]
read_te  <- readability[-loc, ]
```

We will first train the blueprint using the training dataset, and then bake it for both training and test datasets.

```r
prepare <- prep(blueprint,
                training = read_tr)
prepare
```

```
Recipe

Inputs:

      role #variables
   outcome          1
 predictor        990

Training data contained 2551 data points and 2551 incomplete rows.

Operations:

Zero variance filter removed puncts, numbers, symbols, urls, tags, e... [trained]
Sparse, unbalanced variable filter removed wl.16, wl.17, wl.18, wl.19, wl.20, wl.2... [trained]
Mean Imputation for chars, sents, tokens, types, wl.1, wl.2, wl.3, ... [trained]
Centering and scaling for chars, sents, tokens, types, wl.1, wl.2, wl.3, ... [trained]
Correlation filter removed TTR, C, R, CTTR, U, S, Vm, Maas, lgV0, lg... [trained]
```

```r
baked_tr <- bake(prepare, new_data = read_tr)

baked_te <- bake(prepare, new_data = read_te)
```

The smaller test dataset will be used as a final hold-out set, and training dataset will be used to build the model.

Model Fitting without Cross-validation

First, I will fit the model to the training dataset using all predictors in the dataset without any cross validation. Note that we will very likely overfit with more than 800 predictors and relatively small sample size.

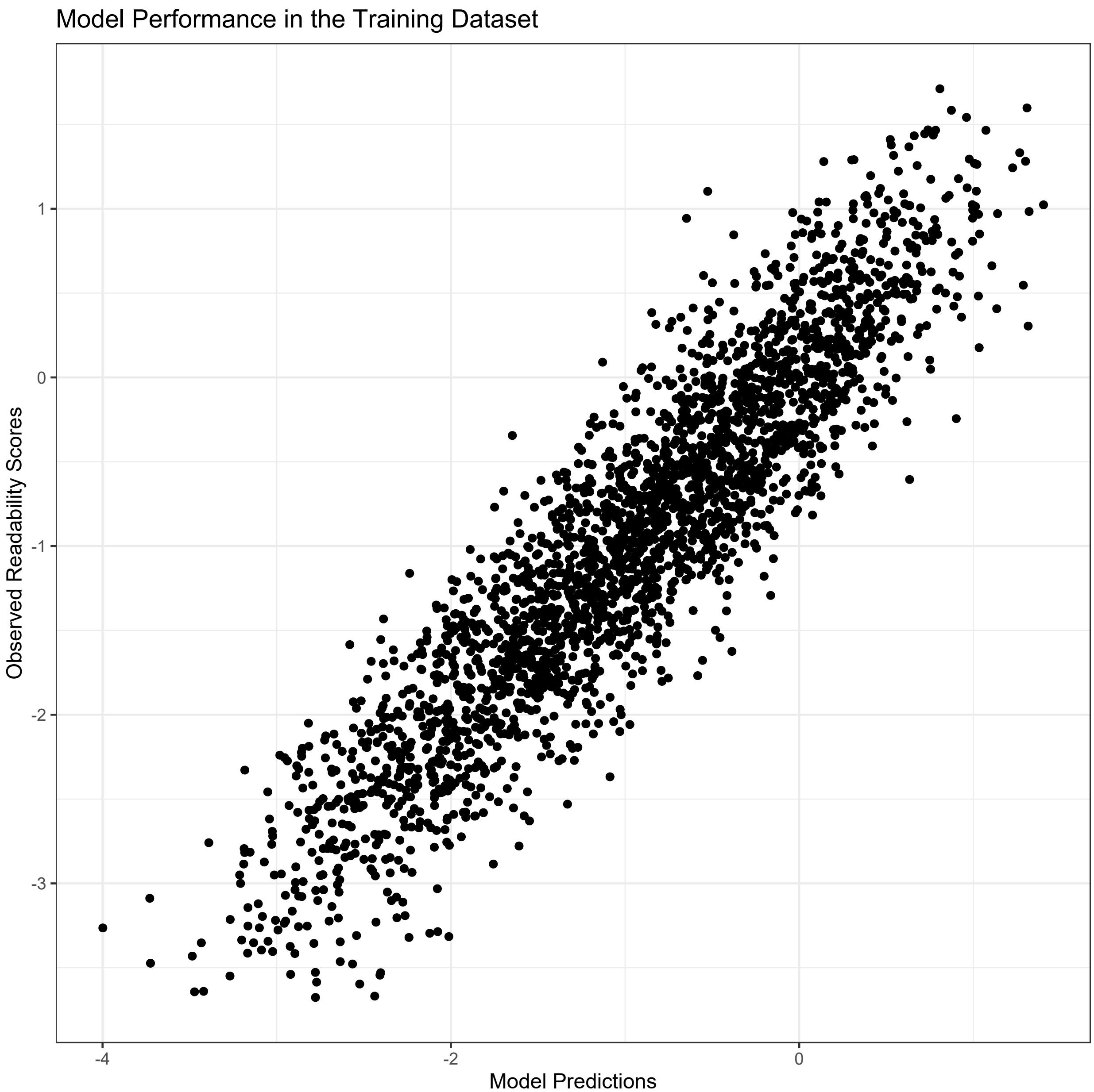
```r
mod <- lm(formula(baked_tr[,c(888,1:887)]),data=baked_tr)

summary(mod)$r.squared
```

```
[1] 0.8403438
```

```r
predicted_tr <- predict(mod)

ggplot()+
  geom_point(aes(y=baked_tr$target,x=predicted_tr))+
  xlab('Model Predictions')+
  ylab('Observed Readability Scores')+
  theme_bw()+
  ggtitle('Model Performance in the Training Dataset')
```



In the training dataset, the model explains about 84% of the total variance in the outcome variable (WOW!). We can also calculate the MAE, MSE, and RMSE for the model predictions in the training dataset.

```r
rsq_tr <- cor(baked_tr$target,predicted_tr)^2
rsq_tr
```

```
[1] 0.8403438
```

```r
mae_tr <- mean(abs(baked_tr$target - predicted_tr))
mae_tr
```

```
[1] 0.3275843
```

```r
mse_tr <- mean((baked_tr$target - predicted_tr)^2)
mse_tr
```

```
[1] 0.1708515
```

```r
rmse_tr <- sqrt(mean((baked_tr$target - predicted_tr)^2))
rmse_tr
```

```
[1] 0.4133418
```

Something is too good to be true! As we suspected, the model predictions are unusually good in the training data because we are fitting a super complex model, and we are overfitting. This is why you should never judge how well a model is by looking at the performance of the model on the dataset it is trained. Let's check how well this model does on the test data which we didn't use in the estimation.

```r
# first obtain the predictions according to the model for the observations
# in the test dataset

predicted_te <- predict(mod,newdata=baked_te)

# Calculate the outcome metrics

rsq_te <- cor(baked_te$target,predicted_te)^2
rsq_te
```

```
[1] 0.6445438
```

```r
mae_te <- mean(abs(baked_te$target - predicted_te))
mae_te
```

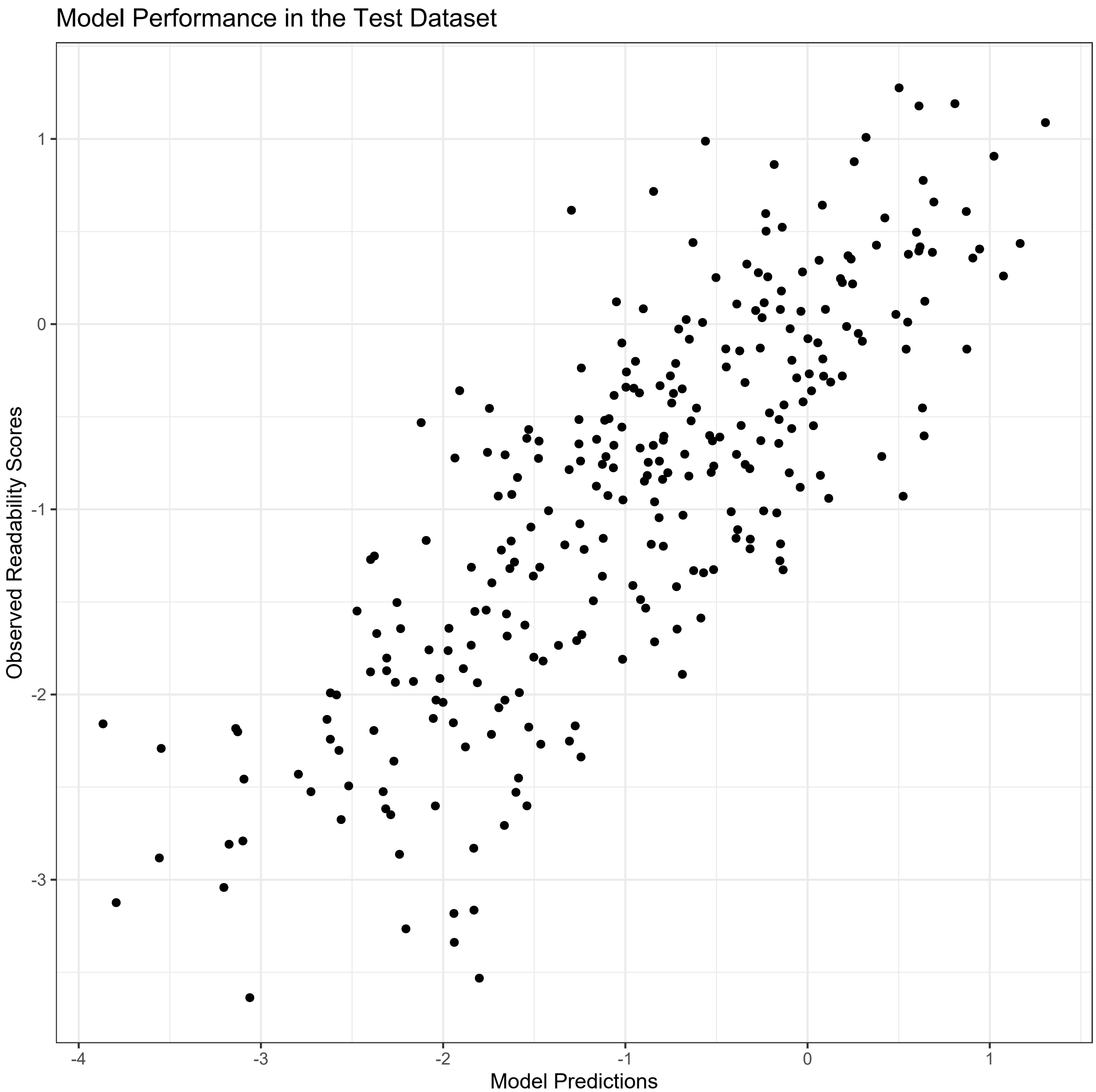
```
[1] 0.5217534
```

```r
mse_te <- mean((baked_te$target - predicted_te)^2)
mse_te
```

```
[1] 0.4152313
```

```r
rmse_te <- sqrt(mean((baked_te$target - predicted_te)^2))
rmse_te
```

```
[1] 0.6443844
```



The model performance significantly dropped in the testing dataset. This is a classic example of model variance (overfitting). We have a very complex model that does a great job in the training dataset but does not perform at the same level in a different dataset. If we are planning to use this model for any future prediction, it is much better to consider the performance on the test data as it will be a more realistic picture of model performance.

Model Fitting with 10-fold Cross-validation

One way of obtaining realistic performance values while we train the dataset is to use k-fold cross validation. The code below first creates 10 folds for the training dataset. Then, it fits the model using the nine folds while it evaluates the performance on the tenth fold.

```
set.seed(10152021)  # for reproducibility

# Randomly shuffle the data

baked_tr = baked_tr[sample(nrow(baked_tr)),]

# Create 10 folds with equal size

folds = cut(seq(1,nrow(baked_tr)),breaks=10,labels=FALSE)

# Create empty vectors for performance measures

rsq  <- c()
mae  <- c()
mse  <- c()
rmse <- c()

# Fit the model by excluding one of the folds, and then evaluate the performance
# on the excluded fold

for(i in 1:10){

  data_tr <- baked_tr[which(folds!=i),] # observation for the 9 folds
  data_te <- baked_tr[which(folds==i),] # observation for the 10th fold

  mod  <- lm(formula(data_tr[,c(888,1:887)]),data=data_tr)

  pred <- predict(mod,newdata=data_te)

  rsq[i]  <- cor(data_te$target,pred)^2
  mse[i]  <- mean(abs(data_te$target - pred))
  mse[i]  <- mean((data_te$target - pred)^2)
  rmse[i] <- sqrt(mean((data_te$target - pred)^2))

  #cat(paste0('Fold ',i,' is completed.'),'\n')
}

rsq
```

```
 [1] 0.6127930 0.6391534 0.5992858 0.6413778 0.6558642 0.6545124 0.6889783
 [8] 0.5365948 0.5753779 0.6299013
```

```
mean(rsq)
```

```
[1] 0.6233839
```

```
rmse
```

```
 [1] 0.6609228 0.6510225 0.6470804 0.6304163 0.6762909 0.6617385 0.6165307
 [8] 0.6930926 0.6900921 0.6552426
```

```
mean(rmse)
```

```
[1] 0.6582429
```

The performance evaluations we obtain from k-fold cross validation is more similar to the one we get from the test data, so they provide a more realistic picture of model performance. We will frequently use k-fold cross-validation for tuning the hyperparameters for several models in later classes.

Model Fitting Using the `caret` package

It is not always the most pleasant experience to write your own code to conduct a k-fold cross validation. Packages like `caret` provides built-in functions for conducting cross-validation and also brings a number of user-friendly experiences in modeling. `caret` provides a standardized user experience for fitting a lot of different models beyond linear regression. So, one doesn't have to learn the nuances of all different types of functions to fit different types of models. Packages like `caret` provides a more consistent workflow while working with different types of models. On the other hand, this also brings less flexibility. During this class, I will try to demonstrate both how to work with direct functions and how to work with `caret` for fitting different types of models.

Below is how one could implement the whole process using the `caret` package.

```
require(caret)
require(recipes)

set.seed(10152021)  # for reproducibility

# Train/Test Split

loc      <- sample(1:nrow(readability), round(nrow(readability) * 0.9))
read_tr  <- readability[loc, ]
read_te  <- readability[-loc, ]

# Blueprint

blueprint <- recipe(x     = readability,
                    vars  = colnames(readability),
                    roles = c(rep('predictor',990),'outcome')) %>%
  step_zv(all_numeric()) %>%
  step_nzv(all_numeric()) %>%
  step_impute_mean(all_numeric()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_corr(all_numeric(),threshold=0.9)

# For available methods in the train function

  # ?names(getModelInfo())
```

```r
  # ?getModelInfo()$lm

# Cross validation settings

cv <- trainControl(method = "cv",
                   p      = 10)

# Train the model

  # note that I provide the blueprint and original unprocessed training dataset
  # as input

caret_mod <- caret::train(blueprint,
                          data      = read_tr,
                          method    = "lm",
                          trControl = cv)

caret_mod
```

```
Linear Regression

2551 samples
 990 predictor

Recipe steps: zv, nzv, impute_mean, normalize, corr
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2296, 2297, 2295, 2296, 2295, 2296, ...
Resampling results:

  RMSE       Rsquared   MAE
  0.6634752  0.6210505  0.5276424

Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
# Once you train the model, then you apply the same blueprint to the test dataset,
# and then can predict the outcome for the observations in the test dataset using
# the model

predicted_te <- predict(caret_mod, read_te)

rsq_te <- cor(read_te$target,predicted_te)^2
rsq_te
```

```
[1] 0.6445438
```

```r
mae_te <- mean(abs(read_te$target - predicted_te))
mae_te
```

```
[1] 0.5217534
```

```r
mse_te <- mean((read_te$target - predicted_te)^2)
mse_te
```

```
[1] 0.4152313
```

```r
rmse_te <- sqrt(mean((read_te$target - predicted_te)^2))
rmse_te
```

```
[1] 0.6443844
```

Using the Prediction Model for a New Text

We now have a model to predict the readability scores using 887 features. We also have a rough idea how well it works. It is not a great model (wouldn't win any prize in the Kaggle competition), but good enough to satisfy your advisor or boss. Now, how do we use this model to predict a readability score for a new text.

Suppose, I have the following passage:

> Mittens sits in the grass. He is all alone. He is looking for some fun. Mittens hits his old ball. Smack! He smells a worm. Sniff! Mittens flips his tail back and forth, back and forth. Then he hears, Scratch! Scratch! What's that, Mittens? What's scratching behind the fence? Mittens runs to the fence. He scratches in the dirt. Scratch! Scratch! Ruff! Ruff! What's that, Mittens? What's barking behind the fence? Mittens meows by the fence. Meow! Meow!

What would be the predicted readability score for this reading passage?

Moving forward, all you need is the R object (`caret_mod`) you created to save all the information from the fitted model using the `caret::train()` function.

First, let's do a cleanup. I will remove everything but the model object from my environment.

```r
# This is pretty old school, but works!

rm(list= ls()[!(ls() %in% c('caret_mod'))])
```

Now, we have to remember how we processed the text data and constructed all the features before for the data we used to build the model. We should apply the exact same procedure to a new text and generate the same features for the new text.

```r
################################################################################

  require(quanteda)
  require(quanteda.textstats)
  require(udpipe)
  require(reticulate)
  require(text)

  ud_eng <- udpipe_load_model(here('english-ewt-ud-2.5-191206.udpipe'))

  reticulate::import('torch')
```

```
Module(torch)
```

```r
reticulate::import('numpy')
```

```
Module(numpy)
```

```r
reticulate::import('transformers')
```

```
Module(transformers)
```

```r
reticulate::import('nltk')
```

```
Module(nltk)
```

```r
reticulate::import('tokenizers')
```

```
Module(tokenizers)
```

```r
################################################################################

new.text <- "Mittens sits in the grass. He is all alone. He is looking for some fun. Mittens hits his o


    # Tokenization and document-feature matrix

      tokenized <- tokens(new.text,
                          remove_punct = TRUE,
                          remove_numbers = TRUE,
                          remove_symbols = TRUE,
                          remove_separators = TRUE)

      dm <- dfm(tokenized)

    # basic text stats

      text_sm <- textstat_summary(dm)
      text_sm$sents <- nsentence(new.text)
      text_sm$chars <- nchar(new.text)

        # text_sm[2:length(text_sm)]

    # Word-length features

      wl <- nchar(tokenized[[1]])

      wl.tab <- table(wl)

      wl.features <- data.frame(matrix(0,nrow=1,nco=30))
      colnames(wl.features) <- paste0('wl.',1:30)

      ind <- colnames(wl.features)%in%paste0('wl.',names(wl.tab))
```

```
    wl.features[,ind] <- wl.tab

    wl.features$mean.wl  <-   mean(wl)
    wl.features$sd.wl    <-   sd(wl)
    wl.features$min.wl   <-   min(wl)
    wl.features$max.wl   <-   max(wl)

    # wl.features

  # Text entropy/Max entropy ratio

    t.ent <- textstat_entropy(dm)
    n     <-  sum(featfreq(dm))
    p     <- rep(1/n,n)
    m.ent <- -sum(p*log(p,base=2))

    ent <- t.ent$entropy/m.ent

    # ent

  # Lexical diversity

    text_lexdiv <- textstat_lexdiv(tokenized,
                                   remove_numbers = TRUE,
                                   remove_punct   = TRUE,
                                   remove_symbols = TRUE,
                                   measure        = 'all')

    # text_lexdiv[,2:ncol(text_lexdiv)]

  # Measures of readability

    text_readability <- textstat_readability(new.text,measure='all')

  # POS tag frequency

    annotated <- udpipe_annotate(ud_eng, x = new.text)
    annotated <- as.data.frame(annotated)
    annotated <- cbind_morphological(annotated)

    pos_tags <- c(table(annotated$upos),table(annotated$xpos))

  # Syntactic relations

    dep_rel <- table(annotated$dep_rel)

  # morphological features

    feat_names <- c('morph_abbr','morph_animacy','morph_aspect','morph_case',
                    'morph_clusivity','morph_definite','morph_degree',
                    'morph_evident','morph_foreign','morph_gender','morph_mood',
                    'morph_nounclass','morph_number','morph_numtype',
                    'morph_person','morph_polarity','morph_polite','morph_poss',
```

```r
                      'morph_prontype','morph_reflex','morph_tense','morph_typo',
                      'morph_verbform','morph_voice')

      feat_vec <- c()

      for(j in 1:length(feat_names)){

        if(feat_names[j]%in%colnames(annotated)){
          morph_tmp   <- table(annotated[,feat_names[j]])
          names_tmp   <- paste0(feat_names[j],'_',names(morph_tmp))
          morph_tmp   <- as.vector(morph_tmp)
          names(morph_tmp) <- names_tmp
          feat_vec  <- c(feat_vec,morph_tmp)
        }
      }

    # Sentence Embeddings

      embeds <- textEmbed(x     = new.text,
                          model = 'roberta-base',
                          layers = 12,
                          context_aggregation_layers = 'concatenate')


    # combine them all into one vector and store in the list object

      input <- cbind(text_sm[2:length(text_sm)],
                          wl.features,
                          as.data.frame(ent),
                          text_lexdiv[,2:ncol(text_lexdiv)],
                          text_readability[,2:ncol(text_readability)],
                          t(as.data.frame(pos_tags)),
                          t(as.data.frame(c(dep_rel))),
                          t(as.data.frame(feat_vec)),
                          as.data.frame(embeds$x)
                          )

      input
```

```
  chars sents tokens types puncts numbers symbols urls tags emojis wl.1 wl.2
1   454    23     78    44      0       0       0    0    0      0    1   11
  wl.3 wl.4 wl.5 wl.6 wl.7 wl.8 wl.9 wl.10 wl.11 wl.12 wl.13 wl.14 wl.15 wl.16
1   14   17   13    7   13    0    1     1     0     0     0     0     0     0
  wl.17 wl.18 wl.19 wl.20 wl.21 wl.22 wl.23 wl.24 wl.25 wl.26 wl.27 wl.28 wl.29
1     0     0     0     0     0     0     0     0     0     0     0     0     0
  wl.30  mean.wl    sd.wl min.wl max.wl      ent       TTR         C        R
1     0 4.487179 1.863829      1     10 0.814852 0.5641026 0.8685891 4.982019
      CTTR       U         S        K        I          D        Vm      Maas
1 3.522819 14.3983 0.7790676 371.4661 10.63736 0.02464202 0.1200806 0.2635387
      MATTR     MSTTR     lgV0     lgeV0      ARI ARI.simple   ARI.NRI
1 0.5641026 0.5641026 3.316535 7.636603 1.400268   43.77592 0.8795987
  Bormuth.MC Bormuth.GP  Coleman Coleman.C2 Coleman.Liau.ECP Coleman.Liau.grade
1 -0.2080375  -469366.8 64.08846   97.89615         77.39033           1.858691
```

```
  Coleman.Liau.short Dale.Chall Dale.Chall.old Dale.Chall.PSK Danielson.Bryan
1            1.85641   58.00615      0.7755164       3.913553        4.400226
  Danielson.Bryan.2 Dickes.Steiwer      DRP       ELF Farr.Jenkins.Paterson
1          84.24513      -162.5317 120.8037 0.6956522             -33.68817
   Flesch Flesch.PSK Flesch.Kincaid      FOG   FOG.PSK    FOG.NRI  FORCAST
1 101.439   3.544277    -0.04687848 1.356522 0.9124766 -0.1773913 8.076923
  FORCAST.RGL    Fucks Linsear.Write      LIW       nWS    nWS.2      nWS.3
1    7.314615 15.21739             0 22.62207 0.7510004 1.663981 -0.4683565
      nWS.4       RIX Scrabble   SMOG   SMOG.C SMOG.simple SMOG.de   Spache
1 -0.7922696 0.6521739 1.809249 3.1291 5.112437           3      -2 3.171912
  Spache.old   Strain Traenkle.Bailer Traenkle.Bailer.2 Wheeler.Smith
1   3.522302 1.226087       -188.3027         -222.1848      6.956522
  meanSentenceLength meanWordSyllables ADJ ADP ADV AUX CCONJ DET INTJ NOUN PRON
1           3.391304          1.205128   4   7   6   6     2   8    2   12   13
  PROPN PUNCT VERB ,  . CC DT IN JJ NN NNP NNPS NNS PRP PRP$ RB UH VBG VBZ WP
1    12    27   10 4 23  2 10  7  4 11  10    2   1   5    2  6  2   3  13  4
  advmod amod appos aux case cc conj cop det nmod nmod:poss nsubj obj obl punct
1      4    2     1   3    7  2    4   3   8    1         2    13   3   6    27
  root morph_case_Nom morph_definite_Def morph_definite_Ind morph_degree_Pos
1   23              5                  6                  1                4
  morph_gender_Masc morph_mood_Ind morph_number_Plur morph_number_Sing
1                 7             13                 3                43
  morph_person_3 morph_poss_Yes morph_prontype_Art morph_prontype_Dem
1             20              2                  7                  3
  morph_prontype_Int morph_prontype_Prs morph_tense_Pres morph_verbform_Fin
1                  4                  7               16                 13
  morph_verbform_Part        Dim1       Dim2      Dim3         Dim4      Dim5
1                   3 -0.02706356 0.07567356 0.0537662 -0.007016965 0.2224346
        Dim6        Dim7      Dim8       Dim9       Dim10       Dim11
1 0.04939023 -0.02959983 0.0514404 0.06070077 -0.05404988 -0.09747072
       Dim12      Dim13       Dim14    Dim15      Dim16       Dim17       Dim18
1 0.01958121 0.03001023 -0.05471453 0.116444 0.03110751 0.008461111 -0.03273323
        Dim19       Dim20       Dim21      Dim22       Dim23       Dim24
1 -0.00477997 -0.02568062 -0.07733209 0.03547738 -0.04466225 -0.02642173
        Dim25      Dim26        Dim27      Dim28      Dim29       Dim30
1 -0.01528819 0.01880905 -0.004117077 0.05099677 0.03213615 -0.03850309
        Dim31       Dim32      Dim33       Dim34        Dim35      Dim36
1 -0.01875267 -0.07710975 0.03183111 -0.03924842 -0.004964702 0.03409737
       Dim37      Dim38     Dim39      Dim40      Dim41      Dim42       Dim43
1 0.04975318 0.01937219 0.0306441 0.03343704 -0.1781216 -0.1211262 -0.06195378
        Dim44      Dim45       Dim46       Dim47       Dim48      Dim49
1 -0.03983452 0.04327423 0.004913424 -0.06031298 -0.02003438 0.02600088
        Dim50      Dim51      Dim52      Dim53      Dim54      Dim55      Dim56
1 -0.03458053 0.04176696 0.06956099 0.02811206 0.01539411 -0.0369029 0.03900474
       Dim57     Dim58      Dim59      Dim60       Dim61     Dim62       Dim63
1 0.01336424 0.1464066 0.04301799 0.01624574 -0.03928176 0.1635819 -0.07011154
      Dim64      Dim65       Dim66      Dim67      Dim68      Dim69       Dim70
1 -0.020654 0.01870028 -0.04506872 0.03214189 0.03903029 0.02988067 0.007619148
       Dim71       Dim72      Dim73       Dim74      Dim75      Dim76
1 0.05666397 -0.09702756 0.04471388 -0.01197373 0.01947797 0.01181384
       Dim77     Dim78       Dim79      Dim80      Dim81       Dim82     Dim83
1 0.04100754 -3.755487 -0.04728175 0.06669956 0.06208374 -0.08338602 0.7477216
       Dim84       Dim85      Dim86      Dim87      Dim88        Dim89
1 0.03297196 -0.02081768 -0.1150639 0.02672025 0.00498704 -0.004569397
```

```
       Dim90      Dim91        Dim92      Dim93     Dim94      Dim95      Dim96
1 0.03231616 0.01925556 -0.001559907 -0.0210035 0.1013899 0.02498922 0.01998869
        Dim97     Dim98      Dim99      Dim100     Dim101      Dim102    Dim103
1 -0.01513632 0.4909967 -0.0013363 -0.04504698 0.01951685 -0.01925687 0.0945139
      Dim104      Dim105     Dim106     Dim107      Dim108     Dim109
1 0.08075197 -0.07083284 0.07294671 0.01257747 -0.04483311 0.05056348
        Dim110      Dim111     Dim112     Dim113      Dim114      Dim115
1 -0.008657346 -0.02187181 -0.0347401 0.06893411 -0.02378071 -0.01619653
    Dim116      Dim117     Dim118     Dim119    Dim120      Dim121     Dim122
1 0.017487 -0.01827984 0.03730945 0.04513853 0.1384863 -0.03498841 0.05956505
      Dim123     Dim124     Dim125     Dim126       Dim127      Dim128
1 0.02513112 0.01840405 -0.0540553 0.02714446 -0.005429095 -0.04092931
       Dim129     Dim130     Dim131     Dim132     Dim133     Dim134     Dim135
1 -0.02057803 0.01787887 -0.0277407 -0.3037597 0.01324851 0.02754409 0.04661173
       Dim136     Dim137    Dim138      Dim139      Dim140     Dim141
1 -0.02268834 0.05471192 0.0194234 0.001750701 -0.01453351 0.02751146
      Dim142      Dim143     Dim144   Dim145     Dim146      Dim147      Dim148
1 0.06153563 -0.02328374 0.01968463 0.162699 0.07096836 -0.03563375 -0.03947923
      Dim149     Dim150     Dim151      Dim152      Dim153        Dim154
1 0.03794007 0.06643526 0.06518798 -0.06979158 -0.01212464 0.00008351834
      Dim155    Dim156     Dim157    Dim158      Dim159    Dim160    Dim161
1 0.09422596 0.1242531 0.01644855 0.1783978 -0.02061414 0.1537144 0.1332055
       Dim162      Dim163      Dim164      Dim165     Dim166     Dim167
1 -0.01335572 0.005136131 -0.03188151 -0.02107196 0.02390574 0.01111268
       Dim168       Dim169      Dim170     Dim171   Dim172       Dim173
1 0.005455089 -0.001085651 -0.05858055 -0.0309496 0.064103 -0.007506766
      Dim174      Dim175      Dim176      Dim177      Dim178      Dim179
1 0.01675711 -0.01900868 -0.02708992 0.007471044 -0.01638369 0.003488106
      Dim180     Dim181     Dim182      Dim183      Dim184     Dim185
1 0.01583095 0.02703102 0.01505022 -0.05409031 -0.06259391 0.02703082
      Dim186     Dim187     Dim188     Dim189    Dim190     Dim191      Dim192
1 0.06376115 0.04706535 -0.0533913 0.03212536 0.1225724 0.09776939 0.006072238
      Dim193     Dim194     Dim195    Dim196      Dim197      Dim198    Dim199
1 0.04005886 0.03815003 0.03690949 0.1071005 -0.03470944 -0.02419928 0.0993732
      Dim200     Dim201     Dim202     Dim203      Dim204      Dim205
1 -0.0349644 0.03788871 -0.0896448 0.09789985 -0.01650451 0.008594197
      Dim206       Dim207     Dim208      Dim209      Dim210      Dim211
1 0.03339729 -0.001833718 0.03931317 -0.02594167 -0.05752807 -0.06062524
      Dim212      Dim213    Dim214     Dim215      Dim216      Dim217
1 0.09902632 -0.01489959 0.1059247 0.08952547 -0.03465532 -0.02865703
      Dim218     Dim219    Dim220     Dim221      Dim222     Dim223     Dim224
1 -0.4604221 0.03485171 0.0448519 0.04513314 -0.01278944 0.03543267 0.06323306
       Dim225    Dim226     Dim227      Dim228      Dim229     Dim230
1 0.003524951 0.0430416 0.08765961 -0.01575762 0.004051357 0.00966449
       Dim231      Dim232      Dim233      Dim234     Dim235     Dim236
1 -0.01163762 -0.02366546 0.009014995 -0.06540983 0.05872543 -0.1206237
       Dim237       Dim238       Dim239     Dim240     Dim241    Dim242
1 -0.03210227 -0.001160354 -0.003590591 0.01495555 -0.1546848 0.1052845
     Dim243     Dim244     Dim245      Dim246       Dim247    Dim248     Dim249
1 0.0657018 0.05770402 0.06140059 -0.07159876 -0.009812683 0.1166845 0.05277905
      Dim250     Dim251      Dim252        Dim253       Dim254     Dim255
1 0.01015359 0.01368594 -0.06787519 -0.0005879147 0.0008820305 0.05449436
       Dim256     Dim257     Dim258      Dim259     Dim260      Dim261
1 -0.06892669 -0.1041239 0.02042176 -0.06112328 -0.0495022 0.007336825
```

```
     Dim262     Dim263      Dim264     Dim265     Dim266     Dim267
1 -0.08096835 0.05881691 -0.07769445 -0.0469258 0.03575953 0.03662355
     Dim268       Dim269      Dim270     Dim271       Dim272     Dim273
1 0.07139029 -0.002513315 -0.05604061 0.01551373 -0.002937071 0.03161258
     Dim274     Dim275     Dim276       Dim277     Dim278      Dim279
1 0.07215895 0.00929201 0.02886985 -0.006060768 0.03395056 0.001992457
      Dim280     Dim281      Dim282      Dim283      Dim284      Dim285
1 -0.02386326 0.06078118 -0.02561207 -0.02657414 0.009780701 -0.04107339
     Dim286     Dim287    Dim288      Dim289      Dim290     Dim291     Dim292
1 0.05771509 0.08124785 0.0312119 -0.01189761 0.005442077 0.02118172 0.06831161
      Dim293      Dim294      Dim295     Dim296    Dim297      Dim298
1 0.005624752 -0.01724207 -0.05921534 -0.0163984 0.1066697 -0.01965118
     Dim299     Dim300    Dim301      Dim302      Dim303      Dim304
1 0.04835701 0.01711501 0.1276233 -0.01874138 -0.02966389 0.007723091
       Dim305      Dim306     Dim307      Dim308      Dim309    Dim310
1 -0.008173925 -0.01010179 0.09693575 -0.01644602 -0.03305898 0.0924959
       Dim311      Dim312      Dim313    Dim314     Dim315      Dim316
1 -0.005221166 -0.07871117 -0.02659134 0.1126804 0.02638598 -0.01897332
     Dim317     Dim318     Dim319     Dim320      Dim321      Dim322
1 0.07200745 0.01983377 0.07517022 0.03761548 -0.01294152 -0.06255002
     Dim323     Dim324     Dim325      Dim326     Dim327       Dim328
1 0.08980877 0.02221829 0.01777704 -0.01418209 -0.1208573 -0.001556365
       Dim329    Dim330    Dim331    Dim332     Dim333    Dim334     Dim335
1 -0.008202043 0.1182082 -0.504334 0.5523129 0.07487423 0.1484527 0.07207321
     Dim336     Dim337     Dim338    Dim339    Dim340      Dim341     Dim342
1 0.07115493 0.05266665 0.03712041 0.1231284 0.1140854 -0.09936048 0.01078093
     Dim343     Dim344    Dim345     Dim346      Dim347      Dim348    Dim349
1 -0.1328449 0.04264652 0.0278524 0.08925101 -0.01208034 -0.01879104 0.1194623
       Dim350     Dim351      Dim352     Dim353      Dim354      Dim355
1 -0.007769841 0.01397669 -0.02011757 0.03728889 0.008268313 -0.02807958
     Dim356     Dim357    Dim358      Dim359     Dim360    Dim361   Dim362
1 0.01463395 0.06802522 0.0190933 -0.02336735 -0.1009301 0.1280767 0.194199
     Dim363      Dim364       Dim365      Dim366      Dim367    Dim368
1 0.07359837 -0.02260096 -0.008612545 -0.09422011 -0.02818615 0.0786788
     Dim369    Dim370     Dim371      Dim372      Dim373      Dim374
1 0.02272207 0.1002589 0.09570162 -0.04916206 0.004951689 0.005069547
      Dim375      Dim376    Dim377     Dim378     Dim379     Dim380
1 -0.01894144 -0.01217675 0.1045737 0.02512248 0.07700328 0.05863359
      Dim381     Dim382      Dim383     Dim384    Dim385      Dim386
1 -0.03486566 0.03618479 -0.03967949 0.03187413 0.1228099 0.004328088
      Dim387      Dim388    Dim389      Dim390       Dim391     Dim392
1 -0.05571836 -0.01382447 0.1014563 0.005540595 -0.002099469 0.04699913
      Dim393      Dim394     Dim395     Dim396      Dim397      Dim398
1 -0.02092642 -0.03988863 0.00658244 0.02015599 -0.01267892 -0.04250392
     Dim399      Dim400     Dim401     Dim402       Dim403     Dim404
1 0.03640548 -0.05950782 0.01229393 0.03474214 -0.003756263 0.07556196
      Dim405     Dim406    Dim407      Dim408    Dim409     Dim410      Dim411
1 -0.01581631 0.08982348 -0.119573 -0.01489228 0.0321744 0.01545453 -0.01062769
     Dim412     Dim413     Dim414       Dim415      Dim416     Dim417
1 0.07060594 0.05067884 0.02162697 -0.007810039 -0.02777156 0.03783101
      Dim418       Dim419      Dim420      Dim421     Dim422      Dim423
1 -0.02935591 -0.008409697 -0.05037494 -0.08629285 0.04084516 -0.01418345
      Dim424     Dim425     Dim426    Dim427      Dim428     Dim429
1 -0.06365343 0.09158956 0.01061955 0.1100217 0.005894495 0.02905461
```

```
       Dim430      Dim431      Dim432     Dim433      Dim434     Dim435
1 -0.05902156 -0.03096861 0.03136865 -0.1127055 0.003991385 0.01402383
        Dim436     Dim437     Dim438      Dim439     Dim440     Dim441
1 -0.002818544 0.03812404 0.01630672 -0.01887866 0.02094255 0.03740776
       Dim442     Dim443     Dim444      Dim445      Dim446      Dim447
1 -0.02140117 0.04006733 0.03938072 -0.02091419 0.002862708 -0.04301161
      Dim448      Dim449      Dim450      Dim451     Dim452     Dim453
1 0.06656485 0.002899278 -0.07097059 0.006268004 0.07627844 -0.1656777
     Dim454     Dim455      Dim456     Dim457      Dim458      Dim459
1 -1.303071 0.03292015 0.001216001 0.01929163 0.006321201 -0.06444547
      Dim460     Dim461      Dim462     Dim463      Dim464     Dim465
1 0.03222056 0.03787213 -0.04272534 0.01621736 -0.03500902 0.05988663
       Dim466      Dim467     Dim468      Dim469     Dim470     Dim471
1 -0.05526236 -0.02389414 0.01373466 -0.08350613 0.04199342 0.04698859
       Dim472      Dim473     Dim474      Dim475      Dim476     Dim477
1 0.001606023 -0.05817793 -0.1181426 -0.01995927 -0.03852597 0.09707201
     Dim478     Dim479     Dim480     Dim481     Dim482     Dim483    Dim484
1 0.1278216 -0.0297847 0.01828551 0.01670286 0.03277786 0.09394352 0.0245835
       Dim485      Dim486       Dim487     Dim488    Dim489      Dim490
1 -0.06042151 -0.06419709 -0.001968642 0.03546384 0.1266303 -0.03766909
      Dim491      Dim492     Dim493     Dim494    Dim495     Dim496     Dim497
1 0.01635127 -0.05516074 0.07026093 0.01981271 0.1932934 0.06916854 -0.2436134
      Dim498    Dim499      Dim500       Dim501      Dim502      Dim503
1 0.01315579 0.1548217 -0.05302637 -0.005513271 -0.01395114 -0.01270218
      Dim504     Dim505    Dim506      Dim507      Dim508      Dim509
1 0.07853533 0.01600896 0.1132618 -0.02287428 -0.04421847 -0.01170723
      Dim510      Dim511    Dim512    Dim513     Dim514     Dim515      Dim516
1 -0.0404612 -0.01502825 0.1068065 0.0599024 -0.0431066 0.03054362 -0.01285851
      Dim517      Dim518      Dim519     Dim520     Dim521     Dim522
1 -0.0191697 -0.03153732 0.004567779 0.07818143 -0.0116552 0.04245283
       Dim523      Dim524      Dim525        Dim526     Dim527     Dim528
1 -0.04132793 -0.02695129 -0.03815959 -0.0002546331 0.05014114 0.05791585
      Dim529      Dim530      Dim531     Dim532     Dim533        Dim534
1 0.07237219 -0.03172074 -0.03670841 -0.1283303 0.09187798 0.00001823028
      Dim535     Dim536     Dim537     Dim538      Dim539       Dim540
1 0.01670865 0.03138058 0.05869025 0.02617778 -0.04369681 0.0009510192
       Dim541     Dim542     Dim543    Dim544    Dim545     Dim546     Dim547
1 -0.09070309 0.08426145 0.01746497 0.0143929 0.1264933 0.05054398 0.04296028
       Dim548      Dim549      Dim550     Dim551     Dim552     Dim553
1 -0.04752717 -0.01427257 0.002091692 0.01990349 -0.3774972 0.01090082
      Dim554     Dim555      Dim556     Dim557     Dim558      Dim559
1 0.04931559 0.01802916 -0.01944041 0.08597497 0.01587856 -0.05492282
       Dim560      Dim561     Dim562     Dim563     Dim564       Dim565
1 -0.07542327 0.006113836 0.02933785 0.01319544 0.01346067 -0.003360703
      Dim566    Dim567     Dim568       Dim569     Dim570     Dim571
1 0.01182955 0.0711575 0.01069196 -0.004726301 0.08904852 -0.1245938
       Dim572      Dim573     Dim574      Dim575     Dim576     Dim577
1 -0.02400672 -0.08335184 0.03032804 -0.06834591 0.03303309 0.08116671
      Dim578      Dim579    Dim580      Dim581       Dim582     Dim583
1 0.08852118 -0.04060681 0.1080719 -0.02857596 -0.002249636 0.07120091
      Dim584     Dim585     Dim586     Dim587      Dim588   Dim589      Dim590
1 0.01929063 0.05036947 0.08373432 0.03119808 -0.02472931 10.40989 -0.03335479
         Dim591    Dim592     Dim593     Dim594     Dim595      Dim596
1 0.00001590491 0.1020112 0.02520605 0.01922344 0.03705714 -0.03027939
```

```
       Dim597     Dim598      Dim599     Dim600      Dim601      Dim602
1 0.02878415 0.04377271 -0.02517537 0.06638837 -0.03684153 -0.07039945
       Dim603      Dim604     Dim605      Dim606    Dim607     Dim608     Dim609
1 0.02370637 -0.02010491 -0.1700441 -0.04967063 0.1437444 0.03364382 0.02854263
       Dim610      Dim611    Dim612       Dim613     Dim614     Dim615
1 0.07468463 -0.01405938 0.3746885 -0.003721654 0.05870501 0.08688462
       Dim616      Dim617      Dim618     Dim619     Dim620     Dim621
1 0.09220773 -0.02893779 -0.02598942 0.01744768 0.07500677 0.02593593
       Dim622     Dim623     Dim624     Dim625     Dim626    Dim627     Dim628
1 -0.0115939 0.07953761 0.05151355 -0.0124666 0.07785031 0.0837042 0.07169254
       Dim629     Dim630     Dim631     Dim632       Dim633    Dim634
1 0.01535143 0.03094547 0.02479821 0.00341702 -0.006340763 0.0334576
        Dim635     Dim636    Dim637     Dim638     Dim639       Dim640
1 0.005409091 0.07502079 0.0159725 0.02834773 0.03345724 -0.002944542
       Dim641     Dim642     Dim643      Dim644       Dim645      Dim646
1 0.05821856 0.06789106 -0.0207611 0.007156217 -0.009964033 -0.02734437
       Dim647      Dim648     Dim649     Dim650      Dim651   Dim652  Dim653
1 0.02922359 -0.03934267 0.04159756 0.05500067 -0.02733942 0.146259 0.01465
      Dim654     Dim655     Dim656     Dim657     Dim658       Dim659
1 0.0749637 0.01639464 0.07828537 0.03919239 0.02469708 -0.008865423
        Dim660      Dim661      Dim662     Dim663      Dim664      Dim665
1 -0.02033695 -0.04864632 0.002052276 0.08007035 -0.07448118 -0.08672906
        Dim666     Dim667     Dim668    Dim669    Dim670     Dim671     Dim672
1 -0.04434929 -0.0139097 -0.0387153 0.1097544 0.0047587 0.01440835 0.05295185
       Dim673     Dim674    Dim675    Dim676       Dim677       Dim678
1 0.05340496 0.05685382 0.1365726 0.0125056 0.0004585826 -0.007205268
       Dim679     Dim680     Dim681      Dim682     Dim683      Dim684
1 0.02306118 0.08900418 0.03653014 -0.02864028 0.07297028 -0.02717314
        Dim685      Dim686    Dim687      Dim688      Dim689     Dim690
1 -0.05350105 -0.09152712 0.1127953 0.003779681 -0.07994445 0.09819646
         Dim691     Dim692     Dim693       Dim694      Dim695      Dim696
1 -0.009219781 0.02686084 0.03176724 -0.004313332 -0.05570009 -0.04668238
       Dim697      Dim698     Dim699      Dim700     Dim701      Dim702
1 0.03821168 -0.01559122 0.00415158 0.002860376 -0.0194193 -0.02450354
       Dim703       Dim704     Dim705     Dim706     Dim707     Dim708
1 0.04198655 -0.007422571 0.04795915 0.04825007 0.01933656 0.06055188
        Dim709     Dim710      Dim711      Dim712     Dim713      Dim714
1 -0.07695175 0.05314383 -0.05001539 -0.01423458 0.01145514 0.007925465
        Dim715     Dim716      Dim717     Dim718     Dim719     Dim720    Dim721
1 -0.00940827 0.04849743 -0.01017194 0.02336836 0.01915652 0.07499151 0.2166015
        Dim722     Dim723      Dim724       Dim725   Dim726      Dim727
1 -0.03781433 0.02676561 -0.03970993 -0.002401707 0.109841 0.001826969
        Dim728      Dim729      Dim730        Dim731     Dim732     Dim733
1 0.005976692 -0.01576971 0.005927811 -0.0003839743 -0.2864483 0.07543286
        Dim734     Dim735      Dim736     Dim737    Dim738     Dim739
1 -0.03299744 0.06050834 -0.08300675 0.06494612 0.0186879 0.09525949
        Dim740      Dim741      Dim742       Dim743     Dim744     Dim745
1 -0.02274143 -0.04441616 -0.06460027 0.0005368666 0.01415091 0.01319657
       Dim746     Dim747      Dim748      Dim749     Dim750    Dim751     Dim752
1 0.07915809 0.02910089 -0.01064941 -0.01160094 -0.4062178 0.1150987 0.07628248
       Dim753      Dim754       Dim755     Dim756     Dim757       Dim758
1 0.09187859 0.008078155 -0.005122755 0.02323424 0.06230292 -0.002234648
        Dim759     Dim760      Dim761      Dim762     Dim763    Dim764    Dim765
1 -0.01456664 0.03246233 -0.09130879 -0.06135602 0.02928847 0.0785887 0.1399621
```

```
      Dim766      Dim767     Dim768
1 0.03987939 -0.03054548 0.02145188
```

Here, we have a small issue to deal with. Our new input vector has 938 variables. On the other hand, the original data we used to develop the model had 991 variables. We can access to this information using the model object.

```
caret_mod$recipe$var_info
```

```
# A tibble: 991 x 4
   variable type    role      source
   <chr>    <chr>   <chr>     <chr>
 1 chars    numeric predictor original
 2 sents    numeric predictor original
 3 tokens   numeric predictor original
 4 types    numeric predictor original
 5 puncts   numeric predictor original
 6 numbers  numeric predictor original
 7 symbols  numeric predictor original
 8 urls     numeric predictor original
 9 tags     numeric predictor original
10 emojis   numeric predictor original
# ... with 981 more rows
```

This happended because some of the features don't exist for our new text. They exist but the value for these features are zero, and they just don't appear when we create features from the new text. So, we have to append these missing features to the new text, and make their values to zero. Without these features, the model will look for them to apply the formula and return an error message when it can't find any information about these features in the new dataset. In addition, there were some extra features in the new text that doesn't exist in our model. However, we don't have to worry about them because our recipe is going to ignore any extra column in the new dataset that doesn't have a defined role in the recipe.

Try the following code and it should give an error message

```
  predict(caret_mod, input)
```

So, we have to do a little bit of work to make sure the new dataset have all the features the model expects.

```
# feature names from the model

  my_feats <- caret_mod$recipe$var_info$variable

# column names from the new text

  #colnames(input)

# Find the features missing from the new text

  missing_feats <- ! my_feats %in% colnames(input)

  my_feats[missing_feats]
```

```
 [1] "NUM"                "PART"               "SCONJ"
 [4] "X."                 "CD"                 "HYPH"
 [7] "MD"                 "TO"                 "VB"
[10] "VBD"                "VBN"                "WDT"
[13] "WRB"                "acl.relcl"          "advcl"
[16] "aux.pass"           "compound"           "mark"
[19] "nsubj.pass"         "nummod"             "obl.npmod"
[22] "xcomp"              "morph_case_Acc"     "morph_gender_Neut"
[25] "morph_numtype_Card" "morph_prontype_Rel" "morph_tense_Past"
[28] "morph_verbform_Ger" "morph_verbform_Inf" "morph_voice_Pass"
[31] "X.."                "X...1"              "PRP."
[34] "RP"                 "VBP"                "acl"
[37] "ccomp"              "compound.prt"       "flat"
[40] "nmod.poss"          "parataxis"          "morph_gender_Fem"
[43] "morph_person_1"     "morph_person_2"     "morph_reflex_Yes"
[46] "PDT"                "det.predet"         "morph_mood_Imp"
[49] "obl.tmod"           "EX"                 "expl"
[52] "POS"                "fixed"              "RBR"
[55] "morph_degree_Cmp"   "JJS"                "morph_degree_Sup"
[58] "JJR"                "target"
```

```
# Add the missing features (with assigned values of zeros)

  temp           <- data.frame(matrix(0,1,sum(missing_feats)))
  colnames(temp) <- my_feats[missing_feats]

  input <- cbind(input,temp)

  #input
```

Now, we are ready to apply our model to the new input data and predict the readability score.

```
  predict(caret_mod, input)
```

```
[1] 0.2738756
```

In order to make things a little easier, I will compile the code we are using to generate input features as a function. This function will require two inputs, a model object and a new text. The function will then return a a matrix of input features.

```
generate_feats <- function(my.model,new.text){

    # Tokenization and document-feature matrix

      tokenized <- tokens(new.text,
                          remove_punct = TRUE,
                          remove_numbers = TRUE,
                          remove_symbols = TRUE,
                          remove_separators = TRUE)

      dm <- dfm(tokenized)
```

```r
# basic text stats

  text_sm <- textstat_summary(dm)
  text_sm$sents <- nsentence(new.text)
  text_sm$chars <- nchar(new.text)

# Word-length features

  wl <- nchar(tokenized[[1]])

  wl.tab <- table(wl)

  wl.features <- data.frame(matrix(0,nrow=1,nco=30))
  colnames(wl.features) <- paste0('wl.',1:30)

  ind <- colnames(wl.features)%in%paste0('wl.',names(wl.tab))

  wl.features[,ind] <- wl.tab

  wl.features$mean.wl  <-   mean(wl)
  wl.features$sd.wl    <-   sd(wl)
  wl.features$min.wl   <-   min(wl)
  wl.features$max.wl   <-   max(wl)

# Text entropy/Max entropy ratio

  t.ent <- textstat_entropy(dm)
  n     <-  sum(featfreq(dm))
  p     <- rep(1/n,n)
  m.ent <- -sum(p*log(p,base=2))

  ent <- t.ent$entropy/m.ent

# Lexical diversity

  text_lexdiv <- textstat_lexdiv(tokenized,
                                 remove_numbers = TRUE,
                                 remove_punct   = TRUE,
                                 remove_symbols = TRUE,
                                 measure        = 'all')

# Measures of readability

  text_readability <- textstat_readability(new.text,measure='all')

# POS tag frequency

  annotated <- udpipe_annotate(ud_eng, x = new.text)
  annotated <- as.data.frame(annotated)
  annotated <- cbind_morphological(annotated)

  pos_tags <- c(table(annotated$upos),table(annotated$xpos))
```

```r
# Syntactic relations

  dep_rel <- table(annotated$dep_rel)

# morphological features

  feat_names <- c('morph_abbr','morph_animacy','morph_aspect','morph_case',
                  'morph_clusivity','morph_definite','morph_degree',
                  'morph_evident','morph_foreign','morph_gender','morph_mood',
                  'morph_nounclass','morph_number','morph_numtype',
                  'morph_person','morph_polarity','morph_polite','morph_poss',
                  'morph_prontype','morph_reflex','morph_tense','morph_typo',
                  'morph_verbform','morph_voice')

  feat_vec <- c()

  for(j in 1:length(feat_names)){

    if(feat_names[j]%in%colnames(annotated)){
      morph_tmp   <- table(annotated[,feat_names[j]])
      names_tmp   <- paste0(feat_names[j],'_',names(morph_tmp))
      morph_tmp   <- as.vector(morph_tmp)
      names(morph_tmp) <- names_tmp
      feat_vec  <- c(feat_vec,morph_tmp)
    }
  }

# Sentence Embeddings

  embeds <- textEmbed(x     = new.text,
                      model = 'roberta-base',
                      layers = 12,
                      context_aggregation_layers = 'concatenate')


# combine them all into one vector and store in the list object

  input <- cbind(text_sm[2:length(text_sm)],
                           wl.features,
                           as.data.frame(ent),
                           text_lexdiv[,2:ncol(text_lexdiv)],
                           text_readability[,2:ncol(text_readability)],
                           t(as.data.frame(pos_tags)),
                           t(as.data.frame(c(dep_rel))),
                           t(as.data.frame(feat_vec)),
                           as.data.frame(embeds$x)
                           )

# feature names from the model

  my_feats <- my.model$recipe$var_info$variable

# Find the features missing from the new text
```

```r
      missing_feats <- ! my_feats %in% colnames(input)

    # Add the missing features (with assigned values of zeros)

      temp           <- data.frame(matrix(0,1,sum(missing_feats)))
      colnames(temp) <- my_feats[missing_feats]

      input <- cbind(input,temp)

      return(list(input=input))
}
```

Now, we can get the features for any text using this function and then predict the scores, all in a few lines of code.

```r
# For the next few lines of codes to work, you will need the following
# in your R environment
  # 1. R Libraries (quanteda, quanteda.text, text, udpipe, reticulate)
  # 2. Supplemental Python libraries (torch, tokenizers, nltk, transformers,numpy)
  # 3. Model object (caret_mod)
  # 4. A function to generate the features in the model (generate_feats)


# Sample text 1

my.text1   <- 'Sora has a new kite. The kite is red. It is a good kite to fly. '

my.inputs1 <- generate_feats(my.model = caret_mod,
                             new.text = my.text1)

predict(caret_mod, my.inputs1$input)
```

```
[1] 0.7821635
```

```r
# Sample text 2

my.text2  <- 'Saguaros have roots underground that grow outward rather than downward.'

my.inputs2 <- generate_feats(my.model = caret_mod,
                             new.text = my.text2)

predict(caret_mod, my.inputs2$input)
```

```
[1] -0.1224886
```

Feature Redundancy, Multicollinearity, and Variable Selection

There are a number of things to consider when we wit a standard multiple regression mode with many predictors. In our example above, we have a model with 887 predictors. The large number of predictors unnecessarily increases the complexity of model, and potentially increase model variance. So, it is a typical case of overfitting. This reduces the usefulness of the model as it is less likely for the model to provide good predictions for another dataset. When there are so many predictors in the regression model, it is important to check whether or not there are redundant features and quantify the degree of redundancy. Too many redundant features may also create computational issues due to singular or near-singular design matrix. In this section, we will first try to understand what feature redundancy is, then we will try to quantify it. At the end, we will present some potential solutions and remedial strategies to deal with highly complex models with so many predictors.

First, let's do a small example. Suppose we have a model with four predictors to predict the readability score. Our predictors are **number of sentences** (sents, X1), **average word length** (mean_wl, X2), **number of finite verbs** (morph_verbform_Fin, X3), and **78th dimension of word embeddings** (Dim78). First, let's do a quick check on the correlation matrix of these four predictors.

```
cor(readability[,c('sents','mean.wl','morph_verbform_Fin','Dim78')])
```

```
                        sents    mean.wl morph_verbform_Fin      Dim78
sents               1.0000000 -0.2304859          0.6559804  0.8248184
mean.wl            -0.2304859  1.0000000         -0.5387486 -0.3425791
morph_verbform_Fin  0.6559804 -0.5387486          1.0000000  0.5610935
Dim78               0.8248184 -0.3425791          0.5610935  1.0000000
```

You should notice there is relatively higher correlations among three predictors: number of sentences, number of finite verbs, and Dim78. It is possible that some of the information in any one of these variables is redundant because the same amount of information also exist in other two variables. In order to measure this, we will define a term called **tolerance**.

Tolerance: the amount of variance that is unique to a the predictor that can not be explained by the rest of the predictors.

In other words, if we fit a model such that the **number of sentences** is the outcome and other three variables are predictors and find the value of R^2, and then substract the R^2 from 1, that would give us a measure of unique variance in the **number of sentences** that couldn't be explained by other three predictors. Let's find the tolerance value for the **number of sentences**.

```
tol_sents <- lm(sents ~ 1 + mean.wl + morph_verbform_Fin + Dim78,
                data=readability)

summary(tol_sents)
```

```

Call:
lm(formula = sents ~ 1 + mean.wl + morph_verbform_Fin + Dim78,
    data = readability)

Residuals:
     Min       1Q   Median       3Q      Max
-12.8402  -1.3640  -0.0144   1.3324  18.4418

Coefficients:
                    Estimate Std. Error t value            Pr(>|t|)
(Intercept)        33.872841   0.922890   36.70 <0.0000000000000002 ***
mean.wl             2.176754   0.111025   19.61 <0.0000000000000002 ***
morph_verbform_Fin  0.306746   0.009665   31.74 <0.0000000000000002 ***
Dim78               6.428289   0.104052   61.78 <0.0000000000000002 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.204 on 2830 degrees of freedom
Multiple R-squared:  0.7665,    Adjusted R-squared:  0.7663
F-statistic:  3097 on 3 and 2830 DF,  p-value: < 0.00000000000000022
```

```
summary(tol_sents)$r.squared
```

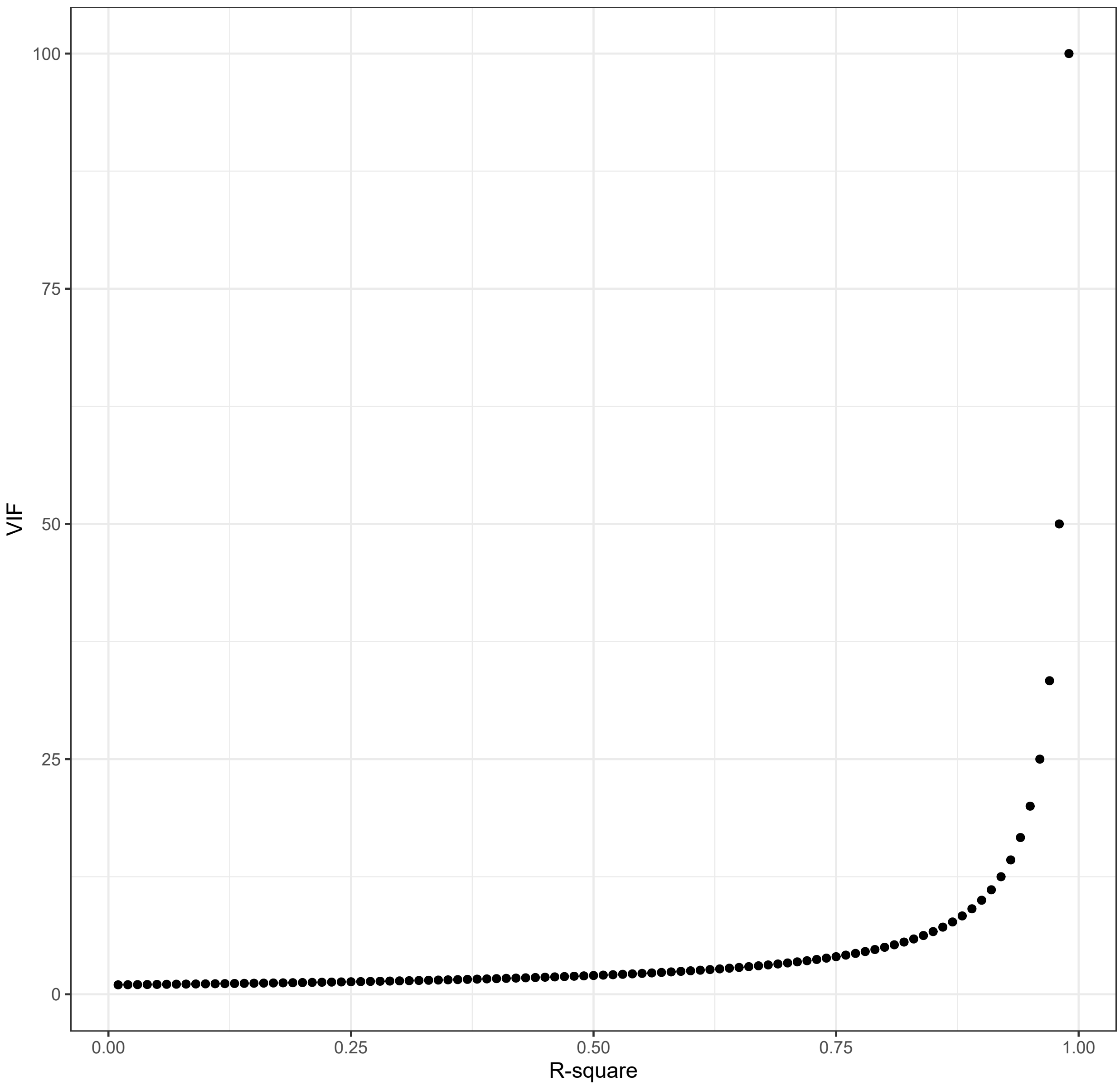
```
[1] 0.7665061
```

This indicates that about 76.65% of the variance in the **number of sentences** can be explained by the other three predictors. Therefore, only 23.35% of the variance in the **number of sentences** is unique. In other words, whatever the information is stored in the **number of sentences**, 76.65% of that information is also shared by other three predictors, or redundant. So, the tolerance value for the **number of sentences** is 0.2335 (1-.7665).

When tolerance is 0 or close to zero (when almost all of the variance in one predictor can be explained by other predictors), this is also known as **singularity**. In those situations, the least square solution is not unique, and most software will give you some sort of an error message about that.

The inverse of tolerance is known as something called **Variance Inflation Factor (VIF)**. For instance, VIF for the **number of sentences** would be 4.283 (1/0.2335). VIF can be considered as a measure of redundancy for a predictor in a model. Below is a plot showing VIF for a predictor as a function of variance in the predictor explained by remaining predictors in the model.

Variance Inflation Factor



If we go back to our example, suppose we have a model with four predictors as mentioned before. The `vif()` function from the `car` package provides a simple and quick way of calculating VIF values for all the predictors in the model.

```
mod_ex <- lm(target ~ 1 + sents + mean.wl + morph_verbform_Fin + Dim78,
                data=readability)


require(car)

vif(mod_ex)
```

```
           sents            mean.wl morph_verbform_Fin              Dim78
        4.282767           1.605680           2.469260           3.439327
```

Or, the VIF values for a given set of predictors can be found using the following matrix operation. Let $r_{\mathbf{XX}}$ is an P x P correlation matrix for P predictors in a model. Then, the corresponding VIF values for each predictor are the diagonal elements of output matrix obtained from the following formula,

$$r_{\mathbf{XX}}^{-1} r_{\mathbf{XX}} r_{\mathbf{XX}}^{-1}.$$

```
rxx <- cor(readability[,c('sents','mean.wl','morph_verbform_Fin','Dim78')])
rxx
```

```
                        sents    mean.wl morph_verbform_Fin      Dim78
sents               1.0000000 -0.2304859          0.6559804  0.8248184
mean.wl            -0.2304859  1.0000000         -0.5387486 -0.3425791
morph_verbform_Fin  0.6559804 -0.5387486          1.0000000  0.5610935
Dim78               0.8248184 -0.3425791          0.5610935  1.0000000
```

```
solve(rxx) %*% rxx %*% solve(rxx)
```

```
                        sents    mean.wl morph_verbform_Fin      Dim78
sents               4.2827669 -0.9068369         -1.6661336 -2.9083117
mean.wl            -0.9068369  1.6056804          1.0677561  0.6989374
morph_verbform_Fin -1.6661336  1.0677561          2.4692602  0.3545629
Dim78              -2.9083117  0.6989374          0.3545629  3.4393274
```

```
diag(solve(rxx) %*% rxx %*% solve(rxx))
```

```
             sents            mean.wl morph_verbform_Fin              Dim78
          4.282767           1.605680           2.469260           3.439327
```

A VIF value indicates the degree of instability (sampling variance) for any regression coefficient. For instance, a VIF value of 4.283 for variable **sents** indicates that the standard error of the regression coefficient associated by this variable is 2.07 ($\sqrt(4.283)$) times larger than what it would be if this variable were uncorrelated with other three predictors in the model. This is important as the larger sampling variance for regression coefficients yield larger sampling variance of model predicted values. So, we don't like including variables with large VIF values in our models as they contribute to the model variance. There are arbitrary cut-off values for VIF depending on what textbook you read (VIF < 4 or VIF < 10).

Let's see the range of VIF values in our model with 887 predictors.

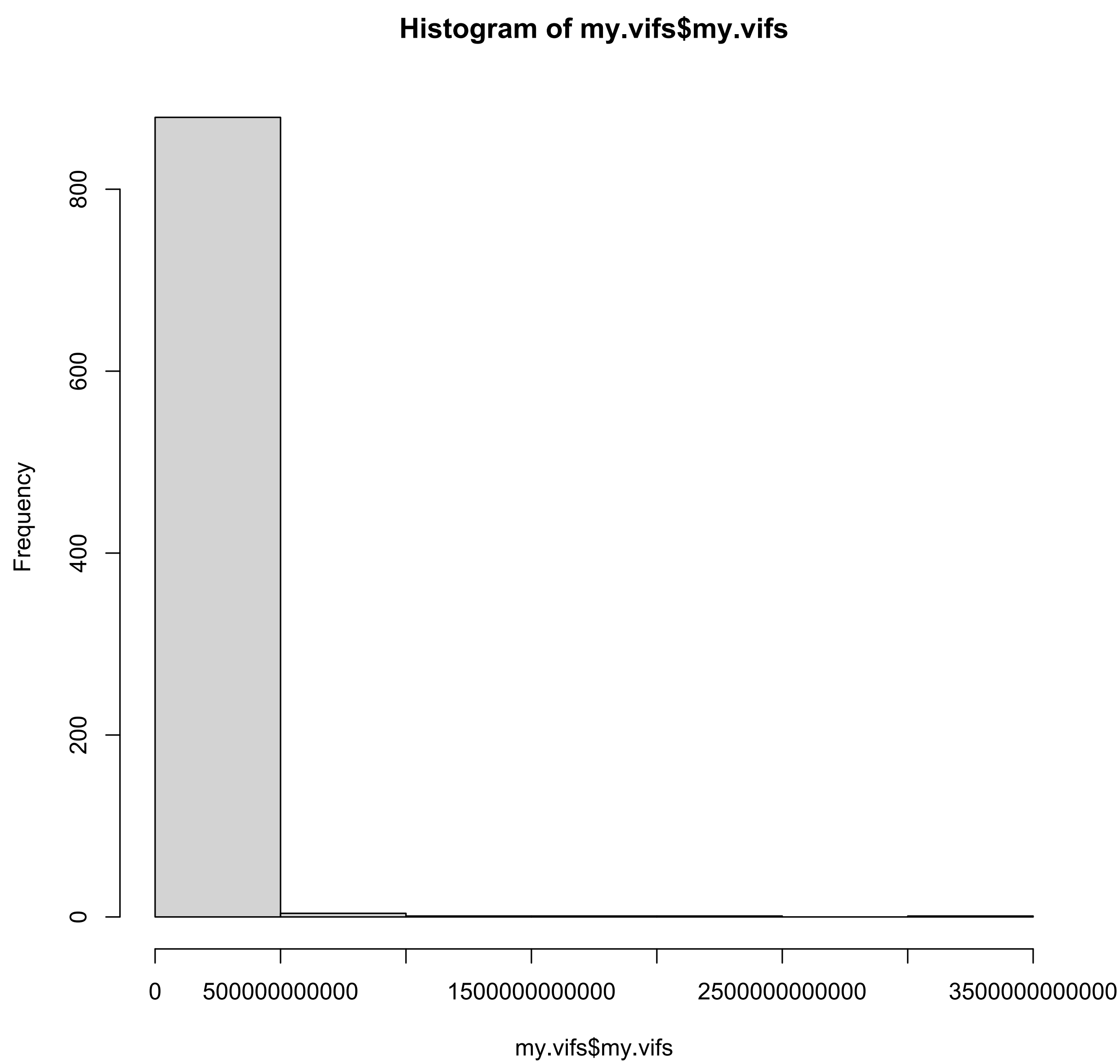
```
my.vifs <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/main/da

require(psych)

describe(my.vifs$my.vifs)
```

```
   vars   n        mean           sd      median     trimmed         mad  min
X1    1 887 44372035413 159833144573 24738324617 26982945999 13415325381 1.63
             max         range  skew kurtosis          se
X1 3342188962798 3342188962797 14.89   260.73 5366671766
```

```
hist(my.vifs$my.vifs)
```




```
# The variables with the 10 smallest variables

head(my.vifs[order(my.vifs$my.vifs),],10)
```

```
                  X  my.vifs
867       parataxis 1.632100
882             POS 1.665525
877   morph_mood_Imp 1.725523
878        obl.tmod 1.805388
871 morph_reflex_Yes 1.894897
```

```
68          obl.npmod 1.965650
886               JJS 1.970530
880                WP 1.983349
884               RBR 2.010719
883             fixed 2.037255
```

```
# The variables with the 10 highest variables

head(my.vifs[order(my.vifs$my.vifs,decreasing=TRUE),],10)
```

```
         X       my.vifs
419 Dim331 3342188962798
166  Dim78 2297168899449
542 Dim454 1609578137016
186  Dim98 1206914009747
583 Dim495  886448110889
306 Dim218  599361142849
838 Dim750  564573101087
640 Dim552  556343375833
677 Dim589  458731818423
820 Dim732  412682088094
```

We clearly have problems in our model. It seems that there are so many redundant variables in our model that doesn't bring unique information when other variables are accounted in the model. That's also the main reason why our model didn't perform in the test dataset as well as it performed in the training dataset. All these redundant variables in the model contributed to the model variance.

There are a few approached to address this issue:

- **Data reduction**: Data reduction techniques such as Principal Component Analysis (PCA) is an approach to find highly correlated variables and combine the information in these variables in new composite variables. For instance, in its most naive form, suppose Variable 1, Variable 2, Variable 3, and Variable 4 are highly correlated. We can create a new composite variable by taking the sum or mean of these four variables, and use the new composite variable in our model as predictor instead of using all four variables. PCA is a little more detailed version of this process where we first estimate a weight for each variable, and create a weighted sum of these variables as a composite variable. We can decide the number of composites needed to represent the information in all variables, and reduce the number of variables in the model by finding clusters of highly correlated variables and creating a single composite variable for them. Since PCA is a technique on its own and probably requires a few lectures, we will not get into the details of that.

- **Variable selection**: Variable selection algorithms such as forward selection, backward elimination, or stepwise regression, or best subset are well known and taught in traditional statistics courses. These algorithms use certain model fit criteria (e.g., Mallows' C_p statistic, AIC, BIC) to eliminate variables with the least information and come up with a simpler model. With very large number of variables, these algorithms can be computationally exhaustive and an efficient search for the best simplest model with adequate predictive power may not even be possible. In other words, there is no guarantee for an optimal solution with these approached when you have hundreds of potential predictors in the model.

- **Regularization**: Regularization is adding penalty terms to avoid large coefficients and a trick to trade bias with variance when fitting a model. Some specific types of regularization (e.g., lasso) may indeed behave like a natural feature selection algorithm. They may produce simpler and more interpretable model.

In the following lecture, we will talk about different types of regularizations to apply while fitting a regression model and we will try to improve the performance of our unnecessarily complex linear regression model discussed above.