Wikipedia usage by Spanish professors

Sameer Sapre

7/10/2020

Introduction

Wikipedia is a free online encyclopedia relying on crowd-sourcing and volunteers for editing. While it is a free and fast resource for those hoping for some quick information, it can also come with pitfalls.

For the same reason that Wikipedia is so fast and publicly available, it can be subject to misinformation. Anyone really can write anything for any subject and while mistakes can be corrected by fellow volunteer editors, mistakes that go unaddressed or at least unaddressed for some time can lead to researchers using incorrect information. For this reason, Wikipedia can be deemed as unreliable and inappropriate in some circumstances.

There is not a consensus on whether Wikipedia should be used in an academic setting. From personal experience, while it can be helpful as a tool to guide research or get a basic understanding of a subject, it may not be best to use it as the sole source of information. Sometimes professors want research papers used, sometimes Wikipedia articles don't contain enough detail or offer all perspectives on a topic that I am looking for. Some professors that I have had have been more enient with its use then others, but again, there is by no means a consensus.

That is what I am trying to address with this analysis. I am going to use data gathered from surveys sent to professors at two spanish universities (Universitat de Oberta de Catalunya and Universitat Fabra). This data contains information related to a professor's demographic information as well as their responses to a questionaire. Using this data, I will try to build a classifier that predicts whether the professor has a positive view of Wikipedia usage by student and faculty or whether they have a negative or neutral point of view.

Due to the large number of columns, I will be using Principal Component Analysis to build a new set of continuous predictors for the dataset. Using these principal components I will be able to reduce dimensionality drastically while still keeping a majority of the variation in the data. This is done by finding a new set of axis along which the natural variation in the data is still present, to an extent. This makes it easier for us to visualize our data and reduce the amount of noise in a potential model. There is a tradeoff, however, with how much variation you can keep. Naturally, the more PCs kept in the model the more of the initial variance is present. However, you still want to reduce the amount of dimensions in the data. The goal is find the optimal point where you can keep as mucn as the variance as possible, but reduce the amount of dimensions at the same time.

After PCA, we will use those transformed components as features to predict how a professor will feel towards Wikipedia usage. We will use a classifier, but there are many to choose from. Since this is a binary classification problem (two outcomes: 0 or 1), we will be looking at binary classification algorithms (though many can extend to multi-class). Though there are many techniques we can use, this analysis will focus on some of the generative modeling methods: logistic regression, linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), and k-nearest neighbors (kNN).

Logisitic Regression is similar to linear regression in that it uses a linear equation to arrive at its estimates and maximum likelihood to produce its coefficients. However, logistic regression is used to estimate probabilites that a sample belongs to a category or class whereas linear regression estimates a continuous output.

LDA and QDA are methods that look to estimate the distribution of predictors among different categories. In other words, these methods look to use "predictors" to estimate the distribution, as best as possible, of the categories of the response variable. In our case, the categories are just 0 and 1. We can then apply this to the rest of the data to find the most likely distribution of each test point in order to classify it.

k-NN is a method that classifies a data point based on the classes of its "neighbors" or closest points. This algorithm assumes that data points that are close to each other, according to some distance or similarity metric, are likely to belong to the same class. In other words, if an unclassified point is close to a cluster of points that all belong to a certain class, then that point is likely to be classified as that class. An example of this could be predicting whether a student passes a test based on past performance. For example, say we wanted to predict if a student would pass a test based on their performance over the past semester. We might look at how others performed on the test who had similar grades over the semester to make a prediction on the new student's outcome.

The evaluation criteria will include prediction accuracy. That is, what percentage of test samples are classified correctly. However, that is not always the best metric to evaluate classifiers as classes are often imbalanced and the accuracy for one class can be much different than others. For example, if a classifier wanted to classify whether a patient had a certain ailment, and that ailment only occured in 2% of the samples then a simple classifier that always predicted that no ailment was present would have 98% accuracy which sounds good, but may actually be a pretty bad classifier because it was unable to correctly identify positive cases.

To evaluate how well these classifiers predict each of the two classes we will look at *sensitiviy*, the proportion of times the classifier correctly predicted that the response was *1*, and *specificity*, the proportion of the times the classifier correctly predicted that the response was *0*.

We will see that by the end of the analysis, the LDA classifier does the best job with this particular dataset.

Data

Again, this data comes from surveys completed by university professors from 2 Spanish universities. The data source is the UCI Machine Learning Repository. Here are the dimensions.

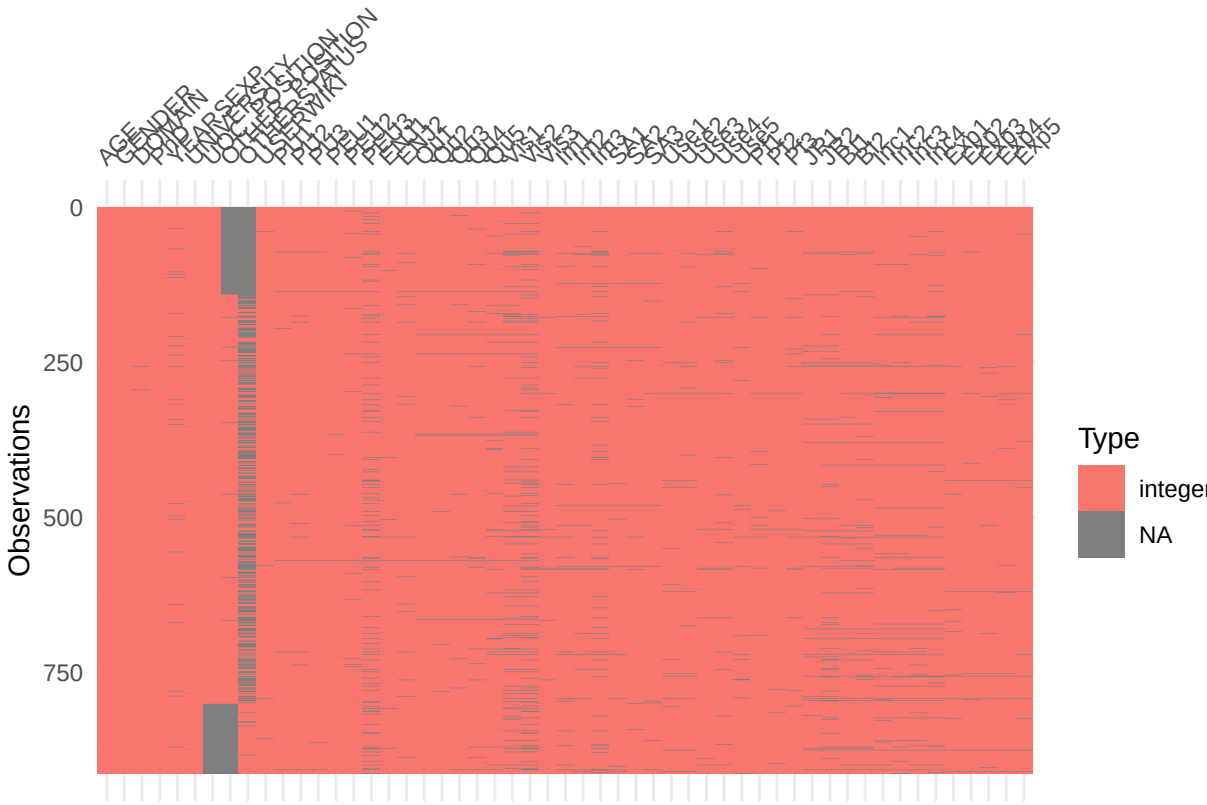
```
wiki = read.csv("wiki4HE.csv", header=T, sep=";", na.strings="?")
dim(wiki)
```

```
## [1] 913  53
```

As you can see there are 53 columns. The variables included are survey results and demographic information about each professor. The survey results are on the Likert scale meaning that the scores range from 1(strongly disagree/never) to 5(strongly agree/always).

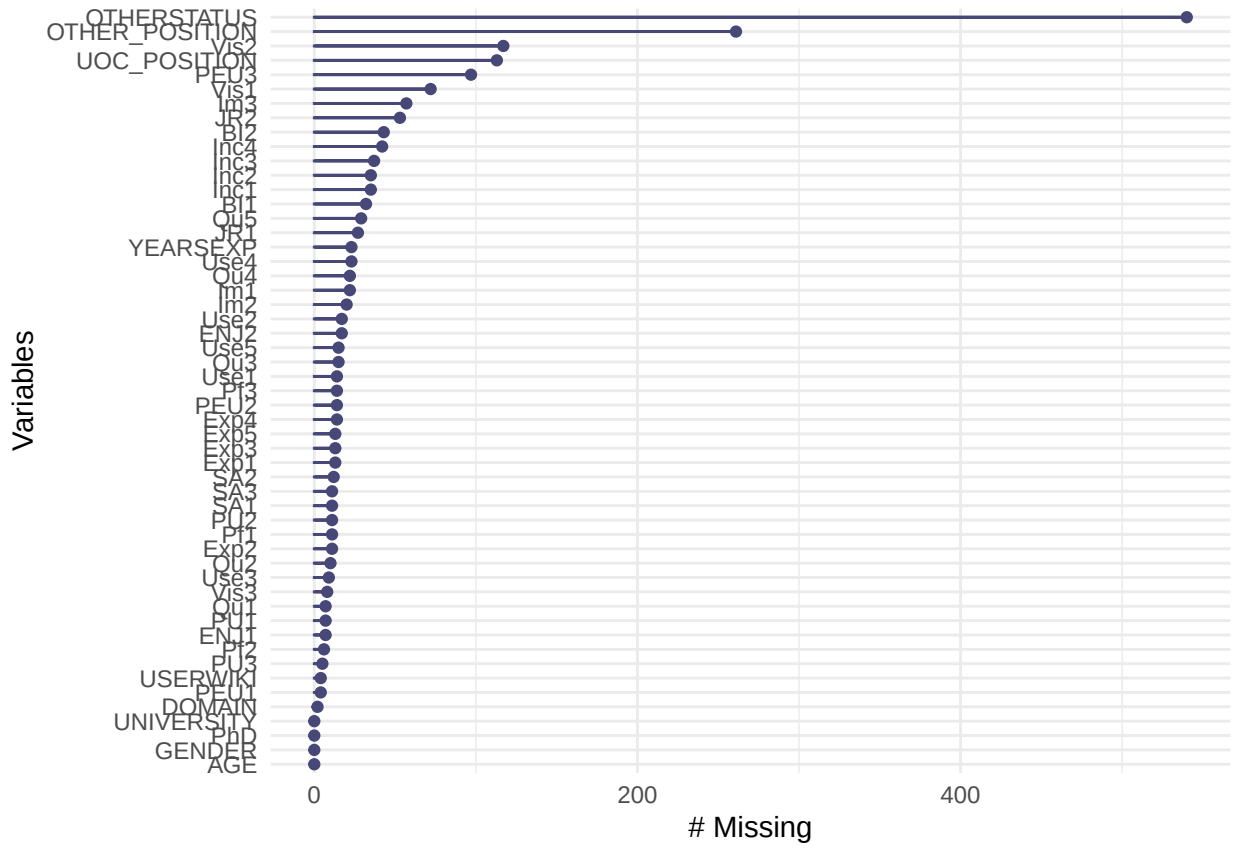
The response variable that we will be trying to predict is actually a combination of responses to the 5 *Use behavior* questions. It is the average response for all 5 questions to create a composite score, also on a scale of 1 to 5. I then take that score and create a binary variable: If the score is greater than 3, than it is considered that the professor uses Wikipedia somewhat often or has a positive view towards its usage by their colleagues and students. If it is less than 3, then they do not carry those positive views and/or do not use it very often in their own work. I came to this cutoff because I wanted the distinction to be clear. If neutral was about a 2.5-3, then all positive cases should be above a composite score of 3.

```
library(naniar)
library(caret)
library(visdat)
set.seed(1)
vis_dat(wiki)
```



Missing Values

```
gg_miss_var(wiki)
```




```
miss_var_summary(wiki)
```

```
## # A tibble: 53 x 3
##    variable       n_miss pct_miss
##    <chr>           <int>    <dbl>
##  1 OTHERSTATUS       540    59.1
##  2 OTHER_POSITION    261    28.6
##  3 Vis2              117    12.8
##  4 UOC_POSITION      113    12.4
##  5 PEU3               97    10.6
##  6 Vis1               72     7.89
##  7 Im3                57     6.24
##  8 JR2                53     5.81
##  9 BI2                43     4.71
## 10 Inc4               42     4.60
## # ... with 43 more rows
```

```
# Row-wis
miss_case_summary(wiki)
```

```
## # A tibble: 913 x 3
##     case n_miss pct_miss
##    <int>  <int>    <dbl>
##  1   532     22     41.5
##  2   792     22     41.5
```

```
##  3   907     22     41.5
##  4   137     21     39.6
##  5   301     20     37.7
##  6   257     18     34.0
##  7   721     18     34.0
##  8    75     17     32.1
##  9   570     17     32.1
## 10   875     17     32.1
## # ... with 903 more rows
```

Other Position and *Other Status* look to have a great deal of missingness which means that we could remove them since they'd only be hindering our efforts.

```
library(tidyverse)
nearZeroVar(wiki,saveMetrics = TRUE)
```

```
##                freqRatio percentUnique zeroVar   nzv
## AGE             1.113636     5.0383352   FALSE FALSE
## GENDER          1.353093     0.2190581   FALSE FALSE
## DOMAIN          1.972678     0.6571742   FALSE FALSE
## PhD             1.153302     0.2190581   FALSE FALSE
## YEARSEXP        1.184615     3.9430449   FALSE FALSE
## UNIVERSITY      7.079646     0.2190581   FALSE FALSE
## UOC_POSITION    9.691176     0.6571742   FALSE FALSE
## OTHER_POSITION  1.432836     0.2190581   FALSE FALSE
## OTHERSTATUS     1.214953     0.7667032   FALSE FALSE
## USERWIKI        6.272000     0.2190581   FALSE FALSE
## PU1             1.380753     0.5476451   FALSE FALSE
## PU2             1.389344     0.5476451   FALSE FALSE
## PU3             1.248000     0.5476451   FALSE FALSE
## PEU1            1.420732     0.5476451   FALSE FALSE
## PEU2            1.430070     0.5476451   FALSE FALSE
## PEU3            1.420000     0.5476451   FALSE FALSE
## ENJ1            1.738739     0.5476451   FALSE FALSE
## ENJ2            1.758929     0.5476451   FALSE FALSE
## Qu1             1.204545     0.5476451   FALSE FALSE
## Qu2             1.046921     0.5476451   FALSE FALSE
## Qu3             1.648305     0.5476451   FALSE FALSE
## Qu4             1.295652     0.5476451   FALSE FALSE
## Qu5             1.564103     0.5476451   FALSE FALSE
## Vis1            2.084211     0.5476451   FALSE FALSE
## Vis2            2.670886     0.5476451   FALSE FALSE
## Vis3            1.828829     0.5476451   FALSE FALSE
## Im1             1.139456     0.5476451   FALSE FALSE
## Im2             1.091575     0.5476451   FALSE FALSE
## Im3             1.568807     0.5476451   FALSE FALSE
## SA1             1.094828     0.5476451   FALSE FALSE
## SA2             1.311258     0.5476451   FALSE FALSE
## SA3             1.740351     0.5476451   FALSE FALSE
## Use1            1.209559     0.5476451   FALSE FALSE
## Use2            2.275362     0.5476451   FALSE FALSE
## Use3            1.053097     0.5476451   FALSE FALSE
## Use4            1.021552     0.5476451   FALSE FALSE
```

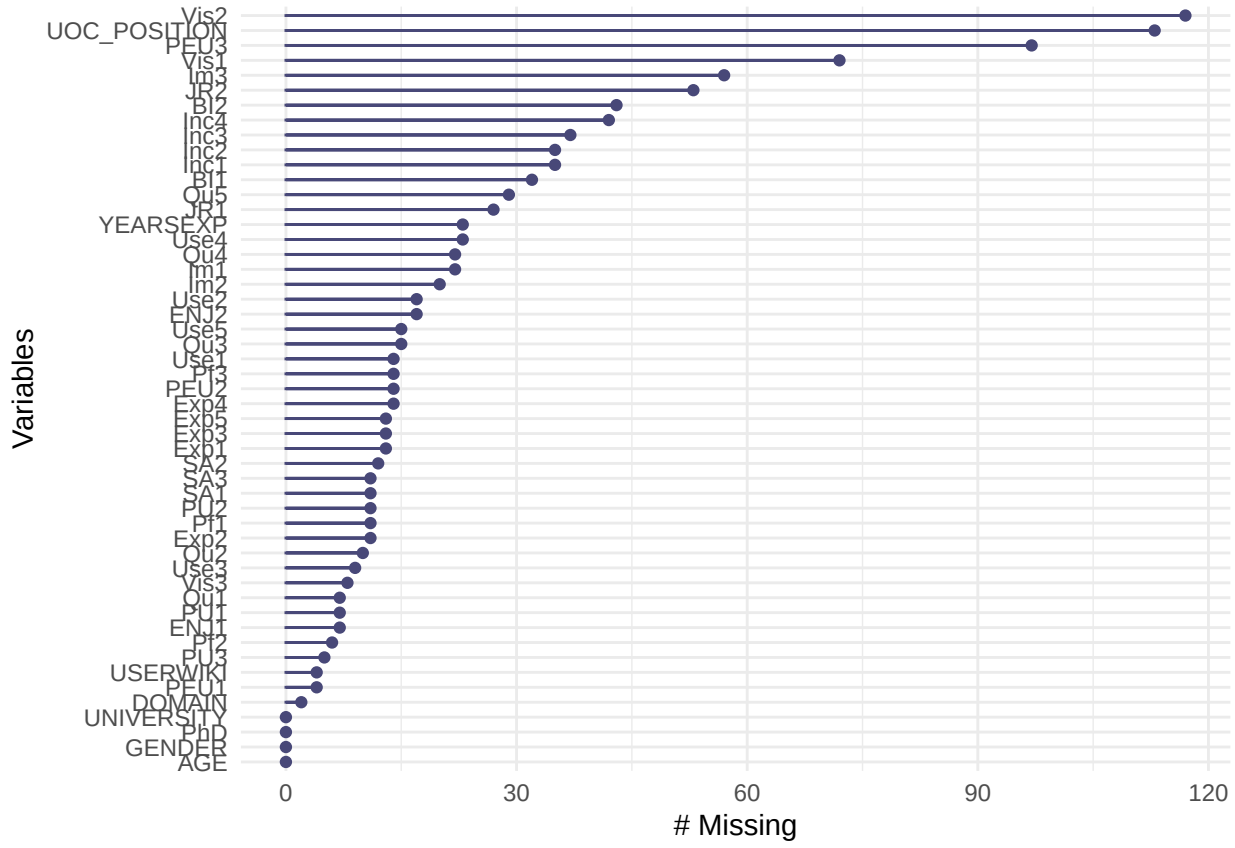
```
## Use5            1.043165     0.5476451   FALSE FALSE
## Pf1             1.559633     0.5476451   FALSE FALSE
## Pf2             1.037209     0.5476451   FALSE FALSE
## Pf3             1.212264     0.5476451   FALSE FALSE
## JR1             1.425439     0.5476451   FALSE FALSE
## JR2             1.288372     0.5476451   FALSE FALSE
## BI1             1.669903     0.5476451   FALSE FALSE
## BI2             1.439614     0.5476451   FALSE FALSE
## Inc1            1.231076     0.5476451   FALSE FALSE
## Inc2            1.046154     0.5476451   FALSE FALSE
## Inc3            1.028000     0.5476451   FALSE FALSE
## Inc4            1.024691     0.5476451   FALSE FALSE
## Exp1            1.073276     0.5476451   FALSE FALSE
## Exp2            1.472973     0.5476451   FALSE FALSE
## Exp3            1.524664     0.5476451   FALSE FALSE
## Exp4            3.295455     0.5476451   FALSE FALSE
## Exp5            1.390476     0.5476451   FALSE FALSE
```

Here we see that both `UOC_Position` and `Other_Position` have large frequency ratios compared to many other potential predictors. Add to that, the large percentage of missing values and both these variables may be candidates to drop completely.

```
wiki %>% dplyr::select(-c(OTHERSTATUS,OTHER_POSITION)) -> wiki
# Take look at missing data after removing problematic columns
miss_case_summary(wiki)
```

```
## # A tibble: 913 x 3
##     case n_miss pct_miss
##    <int>  <int>    <dbl>
##  1   532     21     41.2
##  2   792     21     41.2
##  3   907     21     41.2
##  4   301     20     39.2
##  5   137     19     37.3
##  6   721     18     35.3
##  7   257     17     33.3
##  8   570     17     33.3
##  9   875     16     31.4
## 10    75     15     29.4
## # ... with 903 more rows
```

```
gg_miss_var(wiki)
```




```
miss_var_summary(wiki)
```

```
## # A tibble: 51 x 3
##    variable     n_miss pct_miss
##    <chr>         <int>    <dbl>
##  1 Vis2            117    12.8
##  2 UOC_POSITION    113    12.4
##  3 PEU3             97    10.6
##  4 Vis1             72     7.89
##  5 Im3              57     6.24
##  6 JR2              53     5.81
##  7 BI2              43     4.71
##  8 Inc4             42     4.60
##  9 Inc3             37     4.05
## 10 Inc1             35     3.83
## # ... with 41 more rows
```

Create Score Binary Variable and Final Dataset Here is where we actually preprocess the data to create a composite "usage" score and then use that to create the binary response variable. In addition, we like to see how many complete cases in the data we can get.

```
# Remove variables with missing percentage > 10 and the use variables
wiki %>% select(-c(Vis2,UOC_POSITION,PEU3)) -> wiki
miss_case_summary(wiki)
```

```
## # A tibble: 913 x 3
##     case n_miss pct_miss
##    <int>  <int>    <dbl>
##  1   792     20     41.7
##  2   532     19     39.6
##  3   907     19     39.6
##  4   301     18     37.5
##  5   137     17     35.4
##  6   257     17     35.4
##  7   721     16     33.3
##  8   570     15     31.2
##  9   756     15     31.2
## 10   875     15     31.2
## # ... with 903 more rows
```

```
# Create a composite score based on the average of all USE scores
wiki %>% select(c(Use1,Use2,Use3,Use4,Use5)) %>% rowMeans(na.rm = T) -> score
wiki$Score = ifelse(score >= 3,yes = 1, no = 0)

wiki %>% select(c(Use1,Use2,Use3,Use4,Use5)) -> use_cases
wiki %>% select(-c(Use1,Use2,Use3,Use4,Use5)) -> wiki
miss_case_table(wiki)
```

```
## # A tibble: 19 x 3
##    n_miss_in_case n_cases pct_cases
##             <int>   <int>     <dbl>
##  1              0     649    71.1
##  2              1     136    14.9
##  3              2      42     4.60
##  4              3      22     2.41
##  5              4      12     1.31
##  6              5      11     1.20
##  7              6      13     1.42
##  8              7       3     0.329
##  9              8       6     0.657
## 10              9       1     0.110
## 11             10       3     0.329
## 12             11       3     0.329
## 13             12       2     0.219
## 14             13       2     0.219
## 15             14       1     0.110
## 16             15       2     0.219
## 17             16       2     0.219
## 18             18       2     0.219
## 19             19       1     0.110
```

```
# looks like we can retain about 69.66% of data if we we use na.omit
wiki = na.omit(wiki)
```

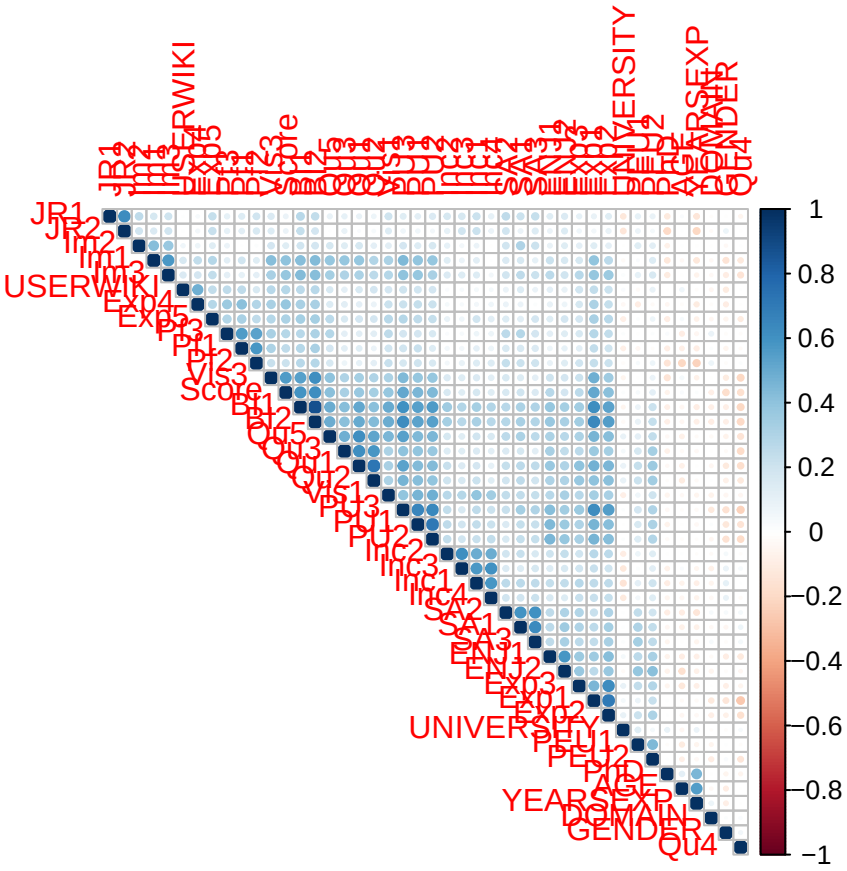
It looks like about 71 % of the rows are complete cases. At 649 obeservation, that should work for us. Of course, this is not an exact science, there are probably much better ways to go about addressing missing values, including imputation, but that will be saved for another project.

```
library(corrplot)
```

Correlation

```
## corrplot 0.84 loaded
```

```
corr_matrix = cor(wiki)
corrplot(corr_matrix,order = "hclust",type = "upper")
```



Analyses

PCA

```
# Create split based on the 'Score' variable
trainIndex = createDataPartition(wiki$Score,p = 0.7, list = FALSE, times = 1)
# Take out binary variables
binary_vars = c("GENDER","PhD","UNIVERSITY","USERWIKI","Score")
wiki_pca = wiki %>% select(-binary_vars)
```

```
## Note: Using an external vector in selections is ambiguous.
```

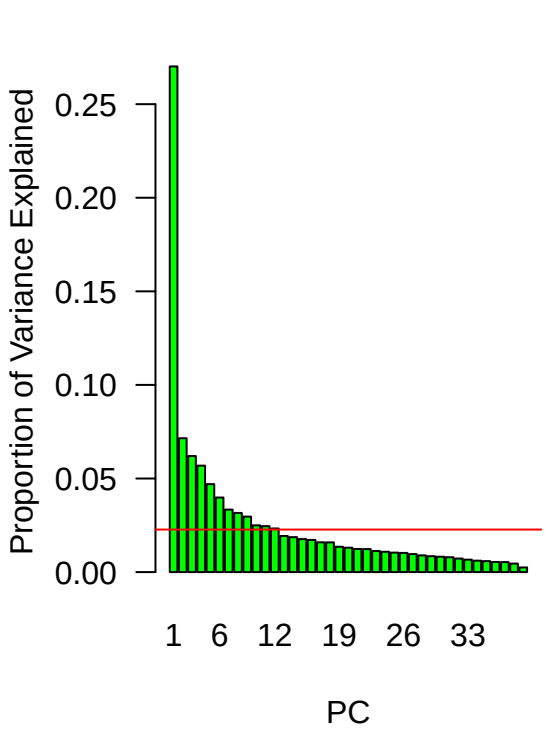
```
## i Use ‘all_of(binary_vars)‘ instead of ‘binary_vars‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

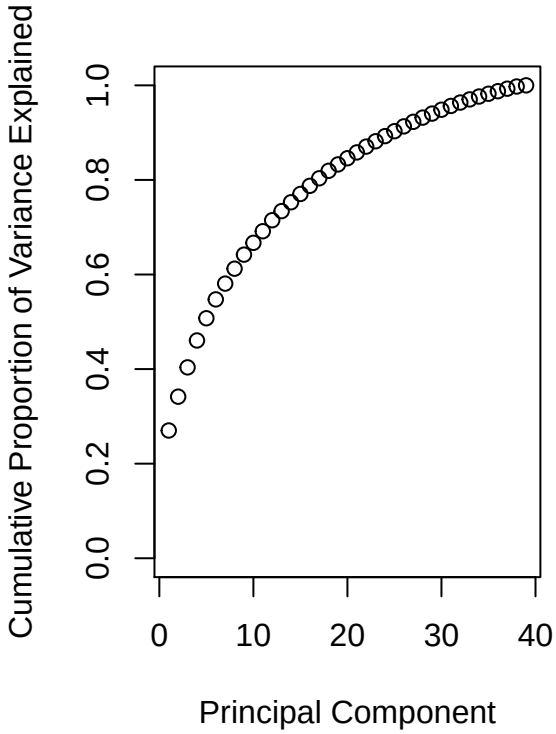
```
pca_train = wiki_pca[trainIndex,]
pca_test = wiki_pca[-trainIndex,]
pca = prcomp(pca_train, scale = T,center = T)
summary(pca)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     3.2459 1.67004 1.55475 1.48936 1.35377 1.24637 1.14102
## Proportion of Variance 0.2702 0.07151 0.06198 0.05688 0.04699 0.03983 0.03338
## Cumulative Proportion  0.2702 0.34167 0.40365 0.46052 0.50752 0.54735 0.58073
##                           PC8     PC9    PC10    PC11   PC12    PC13    PC14
## Standard deviation     1.1102 1.07511 0.98698 0.97814 0.9532 0.86702 0.85328
## Proportion of Variance 0.0316 0.02964 0.02498 0.02453 0.0233 0.01927 0.01867
## Cumulative Proportion  0.6123 0.64197 0.66695 0.69148 0.7148 0.73405 0.75272
##                           PC15    PC16    PC17    PC18    PC19    PC20    PC21
## Standard deviation     0.82900 0.81832 0.78784 0.78667 0.72638 0.71264 0.69416
## Proportion of Variance 0.01762 0.01717 0.01592 0.01587 0.01353 0.01302 0.01236
## Cumulative Proportion  0.77034 0.78751 0.80343 0.81929 0.83282 0.84585 0.85820
##                           PC22    PC23    PC24   PC25    PC26    PC27    PC28
## Standard deviation     0.69228 0.66347 0.65085 0.6368 0.63145 0.61231 0.58960
## Proportion of Variance 0.01229 0.01129 0.01086 0.0104 0.01022 0.00961 0.00891
## Cumulative Proportion  0.87049 0.88178 0.89264 0.9030 0.91326 0.92287 0.93179
##                           PC29    PC30    PC31    PC32    PC33   PC34    PC35
## Standard deviation     0.57558 0.56585 0.55646 0.53116 0.50873 0.4879 0.47775
## Proportion of Variance 0.00849 0.00821 0.00794 0.00723 0.00664 0.0061 0.00585
## Cumulative Proportion  0.94028 0.94849 0.95643 0.96366 0.97030 0.9764 0.98226
##                           PC36    PC37    PC38    PC39
## Standard deviation     0.45817 0.45623 0.42097 0.31104
## Proportion of Variance 0.00538 0.00534 0.00454 0.00248
## Cumulative Proportion  0.98764 0.99298 0.99752 1.00000
```

```
variance = (pca$sdev)^2
pve = variance/sum(variance)

loadings = pca$rotation
rownames(loadings) = colnames(pca)
scores = pca$x
```

```
par(mfrow= c(1,2))
barplot(pve, xlab = "PC", ylab = "Proportion of Variance Explained", names.arg = 1:length(pve), las = 1
abline(h = 1/ncol(wiki), col = "red")
plot(cumsum(pve), xlab = "Principal Component", ylab = "Cumulative Proportion of Variance Explained", yl
```





Scree Plot

The red line represents the cutoff of what one variable's worth of data should contribute if all variables contribute the same amount of variance. That cutoff happens our PC 12, so we will start out using 12 PCs.

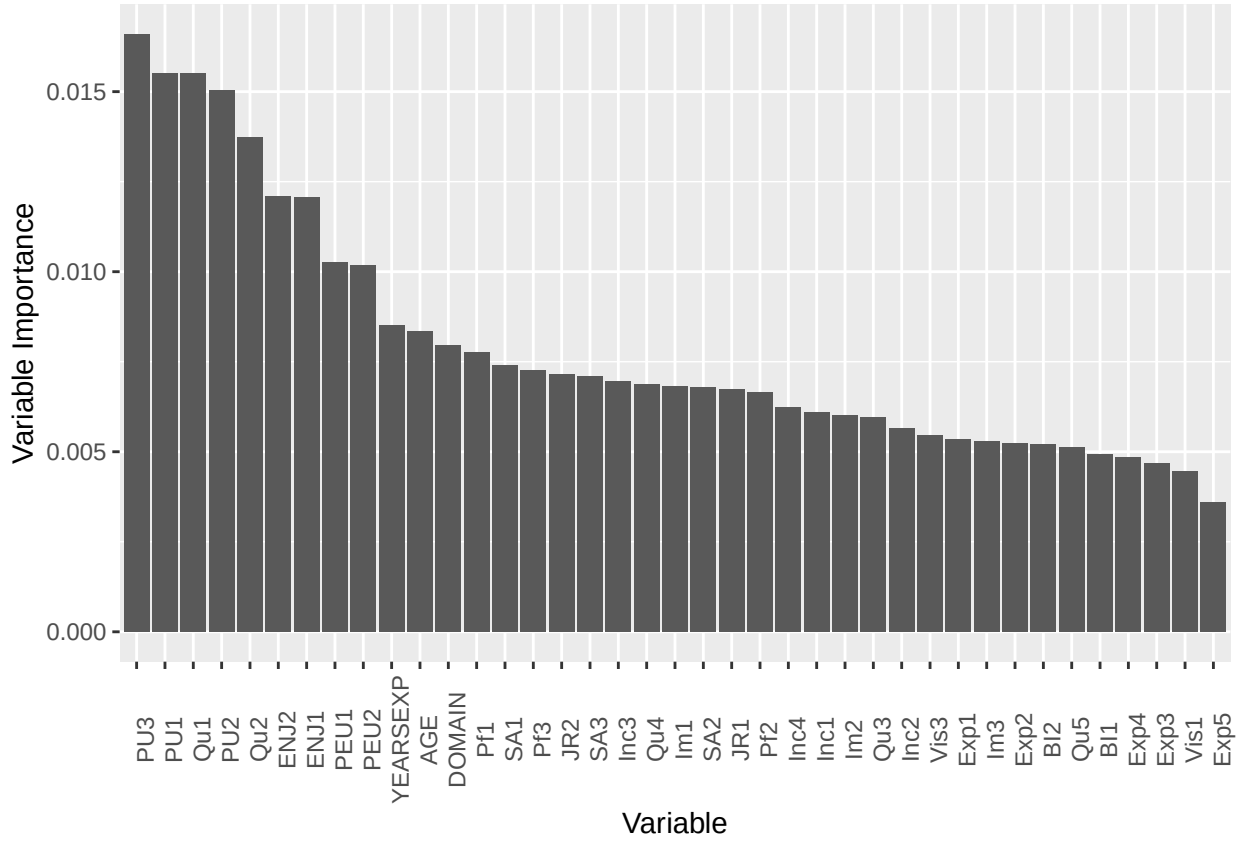
Variable Importance Let's see which variables contribute the most to the top 10 PCs.

```
library(reshape)
```

```
##
## Attaching package: 'reshape'

## The following object is masked from 'package:dplyr':
##
##     rename

## The following objects are masked from 'package:tidyr':
##
##     expand, smiths
```

```
pve.mat = matrix(rep(pve,each = 12), nrow = length(pve))
var.impact = apply(pca$rotation[,c(1:12)]^2 * pve.mat,1,sum)
melt_var = melt(var.impact)

ggplot(data = melt_var) +
  geom_col(aes(x = reorder(rownames(melt_var),-value), y = value)) +
```

```
  theme(axis.text.x = element_text(angle = 90)) +
  labs(x = "Variable", y = "Variable Importance")
```



Looks like the Perceived Usefulness (PU) scores contributed the most to the variance explained.

Train a classifier

Set up test data by projecting points onto new axes.

```
s.pca_test = scale(x = pca_test,center = pca$center)
test_scores = s.pca_test %*% pca$rotation
# Get  just important PCAs
test_scores = test_scores[,c(1:12)]
```

Logistic Regression Set up the df with PCAs and binary variables and fit initial models.

```
pca_df = scores[,1:12]
## add back column variables
wiki %>% select(binary_vars)%>% slice(trainIndex) %>% cbind(pca_df) -> train_df
wiki %>% select(binary_vars) %>% slice(-trainIndex) %>% cbind(test_scores) -> test_df
glm1 = glm(formula = Score ~ .,family = binomial,data = train_df)
summary(glm1)
```

```
##
## Call:
## glm(formula = Score ~ ., family = binomial, data = train_df)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.68529  -0.38753  -0.09581   0.27911   2.89481
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.727254   0.659562  -2.619 0.008824 **
## GENDER      -0.246848   0.345671  -0.714 0.475159
## PhD         -0.262799   0.347373  -0.757 0.449330
## UNIVERSITY   0.319989   0.533603   0.600 0.548722
## USERWIKI     0.931828   0.485455   1.919 0.054922 .
## PC1         -0.925015   0.101421  -9.121  < 2e-16 ***
## PC2         -0.882563   0.124926  -7.065 1.61e-12 ***
## PC3          0.227414   0.105115   2.163 0.030504 *
## PC4         -0.434332   0.123230  -3.525 0.000424 ***
## PC5          0.040021   0.128097   0.312 0.754713
## PC6          0.018633   0.140633   0.132 0.894591
## PC7          0.281872   0.146009   1.931 0.053543 .
## PC8          0.465197   0.150543   3.090 0.002001 **
## PC9          0.007339   0.153766   0.048 0.961935
## PC10        -0.193510   0.168444  -1.149 0.250635
## PC11         0.240723   0.162984   1.477 0.139683
## PC12         0.099788   0.184844   0.540 0.589300
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 594.84  on 454  degrees of freedom
## Residual deviance: 257.81  on 438  degrees of freedom
## AIC: 291.81
##
## Number of Fisher Scoring iterations: 7
```

Now let's try refining the model by including only the significant predictors

```
glm2 = glm(formula = Score ~ PC1 + PC2 + PC3 + PC4 + PC8,family = binomial,data = train_df)
summary(glm2)
```

```
##
## Call:
## glm(formula = Score ~ PC1 + PC2 + PC3 + PC4 + PC8, family = binomial,
##     data = train_df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6473  -0.4114  -0.1120   0.3347   2.9722
##
## Coefficients:
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.37276    0.18960  -7.240 4.48e-13 ***
## PC1         -0.89728    0.09313  -9.635  < 2e-16 ***
## PC2         -0.86781    0.11727  -7.400 1.36e-13 ***
## PC3          0.21753    0.09738   2.234  0.02549 *
## PC4         -0.37750    0.11474  -3.290  0.00100 **
## PC8          0.44458    0.14308   3.107  0.00189 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 594.84  on 454  degrees of freedom
## Residual deviance: 270.06  on 449  degrees of freedom
## AIC: 282.06
##
## Number of Fisher Scoring iterations: 6
```

```
# Get predictions for test data
glm2.probs = predict(glm2,newdata = test_df,type  = "response")
glm2.pred = rep(0,nrow(test_df))
glm2.pred[glm2.probs>.5]= 1
conf_matrix = table(glm2.pred,test_df$Score)
mean(glm2.pred==test_df$Score)
```

```
## [1] 0.8247423
```

```
conf_matrix
```

```
##
## glm2.pred   0   1
##         0 119  12
##         1  22  41
```

```
sensitivity(conf_matrix)
```

```
## [1] 0.8439716
```

```
specificity(conf_matrix)
```

```
## [1] 0.7735849
```

While accuracy was not bad. The specificity was a bit lower, but does not look like a bad classifier

```
library(MASS)
```

LDA

```
##
## Attaching package: 'MASS'

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

```
lda.fit = lda(formula = Score~., data = train_df)


# Testing
lda.pred = predict(lda.fit,test_df)$class
conf_lda = table(lda.pred,test_df$Score)
mean(lda.pred==test_df$Score)
```

```
## [1] 0.8453608
```

```
conf_lda
```

```
##
## lda.pred   0   1
##        0 120   9
##        1  21  44
```

```
sensitivity(conf_lda)
```

```
## [1] 0.8510638
```

```
specificity(conf_lda)
```

```
## [1] 0.8301887
```

Again, here accuracy is good. This looks like a better classifier than the logistic regression.

```
qda.fit = qda(Score~., data = train_df)
```

```
qda.pred = predict(qda.fit,test_df)$class

conf_qda = table(qda.pred,test_df$Score)
mean(qda.pred==test_df$Score)
```

QDA

```
## [1] 0.814433
```

```
conf_qda
```

```
##
## qda.pred   0   1
##        0 117  12
##        1  24  41
```

```
sensitivity(conf_qda)
```

```
## [1] 0.8297872
```

```
specificity(conf_qda)
```

```
## [1] 0.7735849
```

```
library(class)
```

kNN

```
##
## Attaching package: 'class'
```

```
## The following object is masked from 'package:reshape':
##
##     condense
```

```
train.X = subset(train_df, select = -c(Score)) %>% as.matrix()

test.X = subset(test_df, select = -c(Score)) %>% as.matrix()

knn.pred = knn(train = train.X, test = test.X, cl = train_df$Score,k = 1)

mean(knn.pred == test_df$Score)
```

```
## [1] 0.8092784
```

```
knn_table = table(knn.pred,test_df$Score)
knn_table
```

```
##
## knn.pred   0   1
##        0 116  12
##        1  25  41
```

```r
sensitivity(knn_table)
```

```
## [1] 0.822695
```

```r
specificity(knn_table)
```

```
## [1] 0.7735849
```

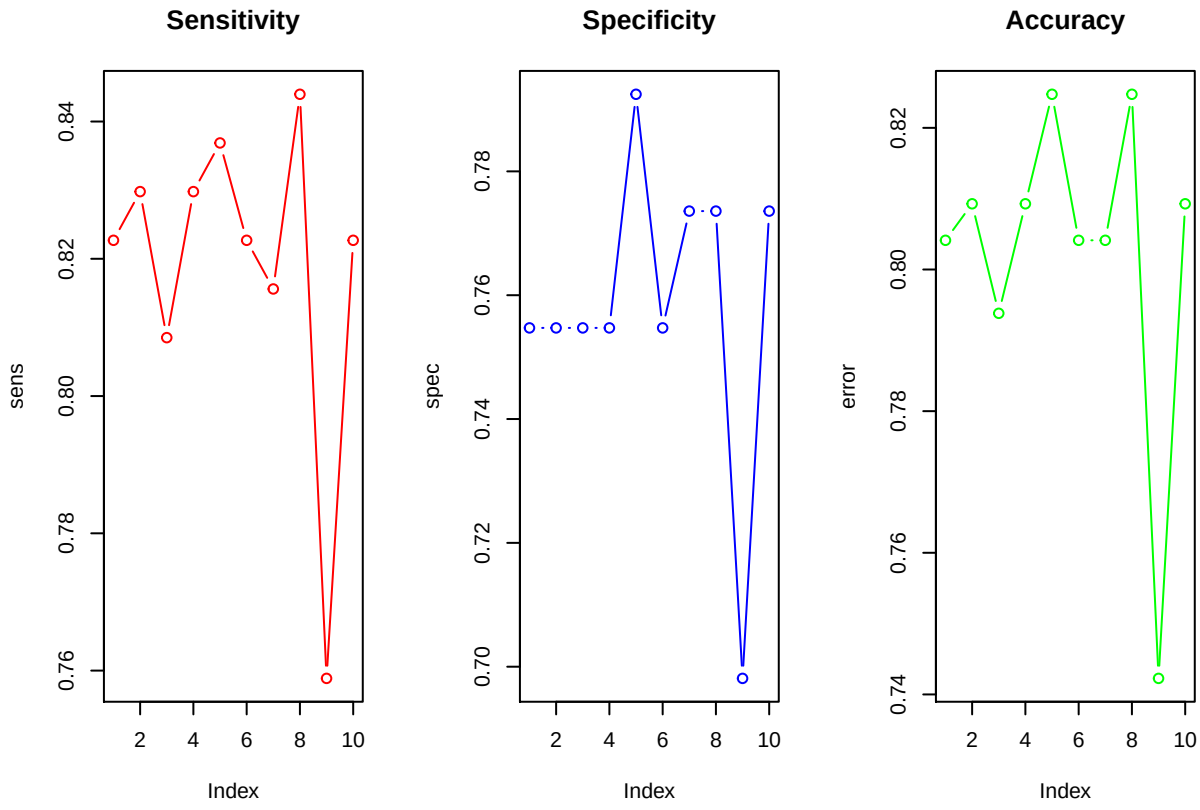
```r
sens = c()
spec = c()
error = c()
for (k in seq(1:10)) {
  train.X = subset(train_df, select = -c(Score)) %>% as.matrix()

  test.X = subset(test_df, select = -c(Score)) %>% as.matrix()

  knn.pred = knn(train = train.X, test = test.X, cl = train_df$Score,k = k)

  error = c(mean(knn.pred == test_df$Score),error)

  knn_table = table(knn.pred,test_df$Score)
  knn_table

  sens = c(sensitivity(knn_table),sens)
  spec = c(specificity(knn_table),spec)

}
par(mfrow = c(1,3))
plot(sens,type = "b",col = "red") + title("Sensitivity")
```

```
## integer(0)
```

```r
plot(spec,type = "b",col = "blue") + title("Specificity")
```

```
## integer(0)
```

```r
plot(error,type = "b", col = "green") + title("Accuracy")
```




```
## integer(0)
```

Let's try k=7. It looks to have maximum or close to maximum values for each

```
train.X = subset(train_df, select = -c(Score)) %>% as.matrix()

test.X = subset(test_df, select = -c(Score)) %>% as.matrix()

knn.pred = knn(train = train.X, test = test.X, cl = train_df$Score,k = 7)

mean(knn.pred == test_df$Score)
```

```
## [1] 0.8092784
```

```
knn_table = table(knn.pred,test_df$Score)
knn_table
```

```
##
## knn.pred   0   1
##        0 117  13
##        1  24  40
```

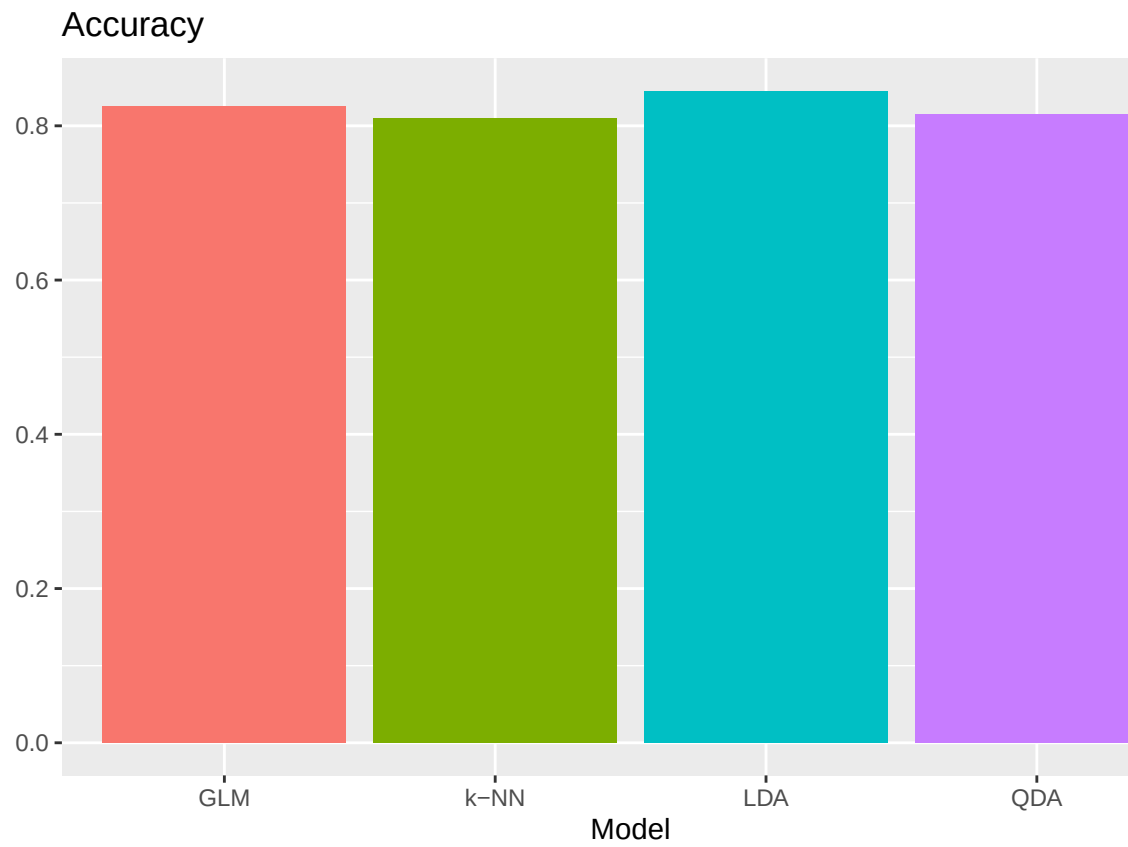
```
sensitivity(knn_table)
```

```
## [1] 0.8297872
```

```
specificity(knn_table)
```

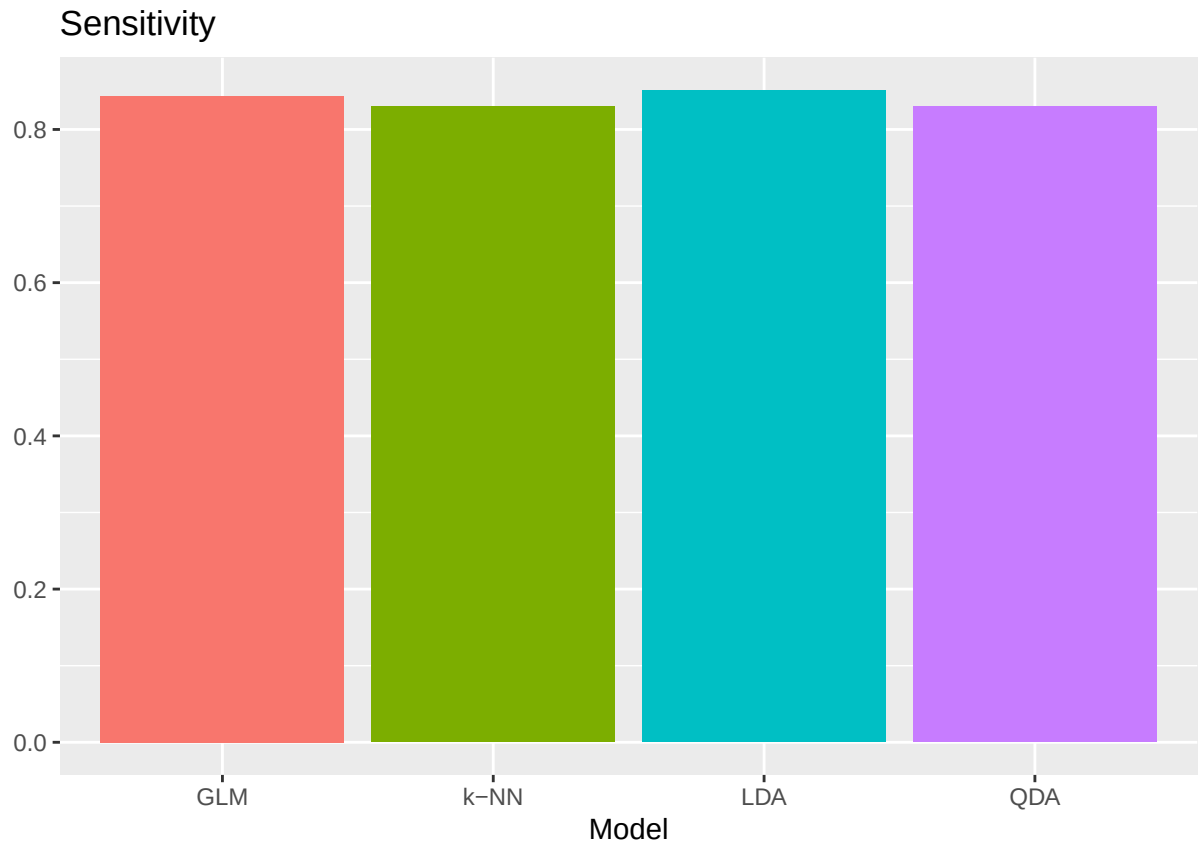
```
## [1] 0.754717
```

```
acc = c(mean(glm2.pred==test_df$Score), mean(lda.pred==test_df$Score),mean(qda.pred == test_df$Score), m
sens = sapply(list(conf_matrix,conf_lda,conf_qda,knn_table),sensitivity)
spec = sapply(list(conf_matrix,conf_lda,conf_qda,knn_table),specificity)
cbind(acc,sens,spec) %>% data.frame() %>% mutate(Model = c("GLM","LDA","QDA","k-NN")) -> model_eval

par(mfrow=c(1,3))
ggplot(model_eval,aes(x = Model,y = acc, fill = Model)) + geom_col() + theme(legend.position = "none") +
  labs(title = "Accuracy", y= "")
```

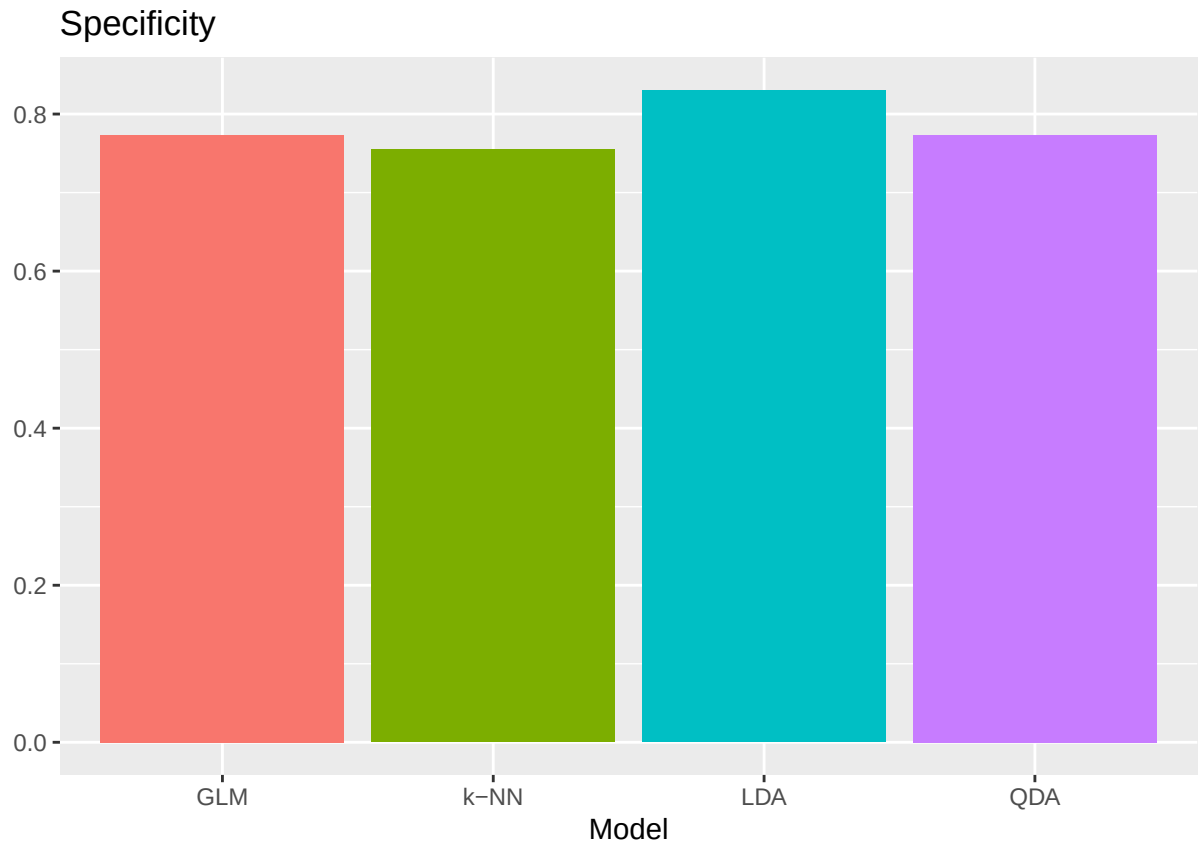


Model Comparison

```
ggplot(model_eval,aes(x = Model,y = acc, fill = Model)) + theme(legend.position = "none") +
  geom_col(aes(x = Model, y= sens)) + labs(title = "Sensitivity", y= "")
```




```
ggplot(model_eval,aes(x = Model,y = spec, fill = Model)) + theme(legend.position = "none")+ geom_col(aes
  labs(title = "Specificity",y="")
```



Conclusion

In conclusion, though the LDA model performed as one of the best when it came to accuracy, it also outperformed the other classifiers when it came to sensitivity and specificity making it the all-around better classifier. That being said, I would like to try cross-validation as a next step for this project just to make sure that this is the best classifier. In addition, as I mentioned earlier, it would be interesting to examine more advanced techniques of dealing with missing data. That includes examining patterns in the missingness to find out if the missing data is occuring at random and implementing imputation so we can keep more of our data. Finally, as I mentioned early, these classifiers are generatve. They don't look to find the differences among classes, but rather look to learn the characteristics of each class and chose the most likely one based on characteristics of the test point. Discriminative classifiers look to find differences and ways to separate classes based on their "predictors".