# Final_Project_GradingPart1

*Ang Li, ANL125@pitt.edu*

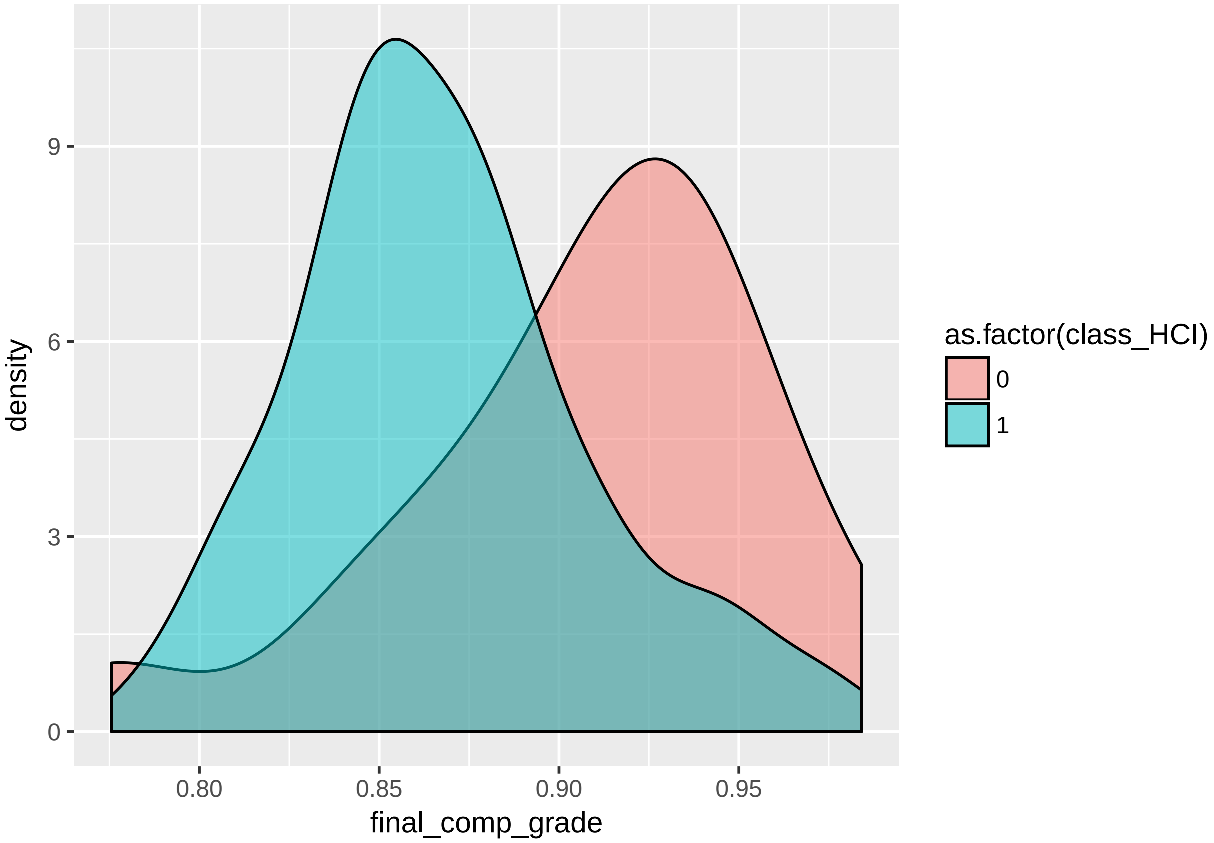
*4/14/2017*

**Explore the dataset and generate both statistical and graphical summary.**

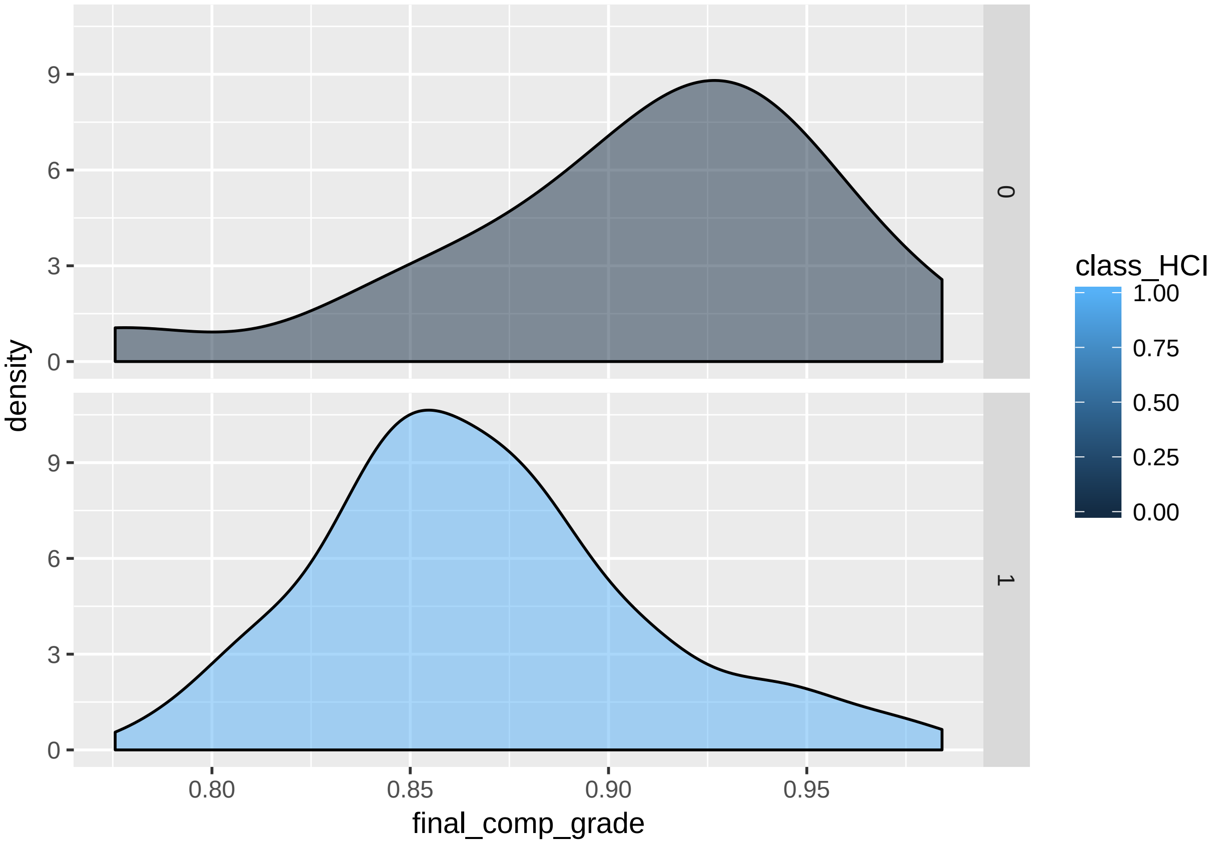
```
setwd("/Users/angli/ANG/GoogleDrive/GoogleDrive_Pitt_PhD/UPitt_PhD_O/2017_Spring/Data-Mining-Spring17/da
#setwd("/Users/ANG/GoogleDrive/GoogleDrive_Pitt_PhD/UPitt_PhD_O/2017_Spring/Data-Mining-Spring17/data/f

grading_data = read.csv("students_grade_final.csv")
```

**For numerical variables final grade, plot the density distribution.**

```
library('ggplot2')
#grading_data
ggplot(grading_data, aes(x = final_comp_grade, fill = as.factor(class_HCI))) +
  geom_density(alpha = 0.5)
```



```
#class_HCI:density plot
ggplot(grading_data, aes(x = final_comp_grade, fill = class_HCI)) +
  geom_density(alpha = 0.5) + facet_grid(class_HCI ~ .)
```



```
#class_HCI: Anova test to compare means
fit_class_HCI = lm(formula = final_comp_grade~class_HCI, data=grading_data)
anova (fit_class_HCI)
```

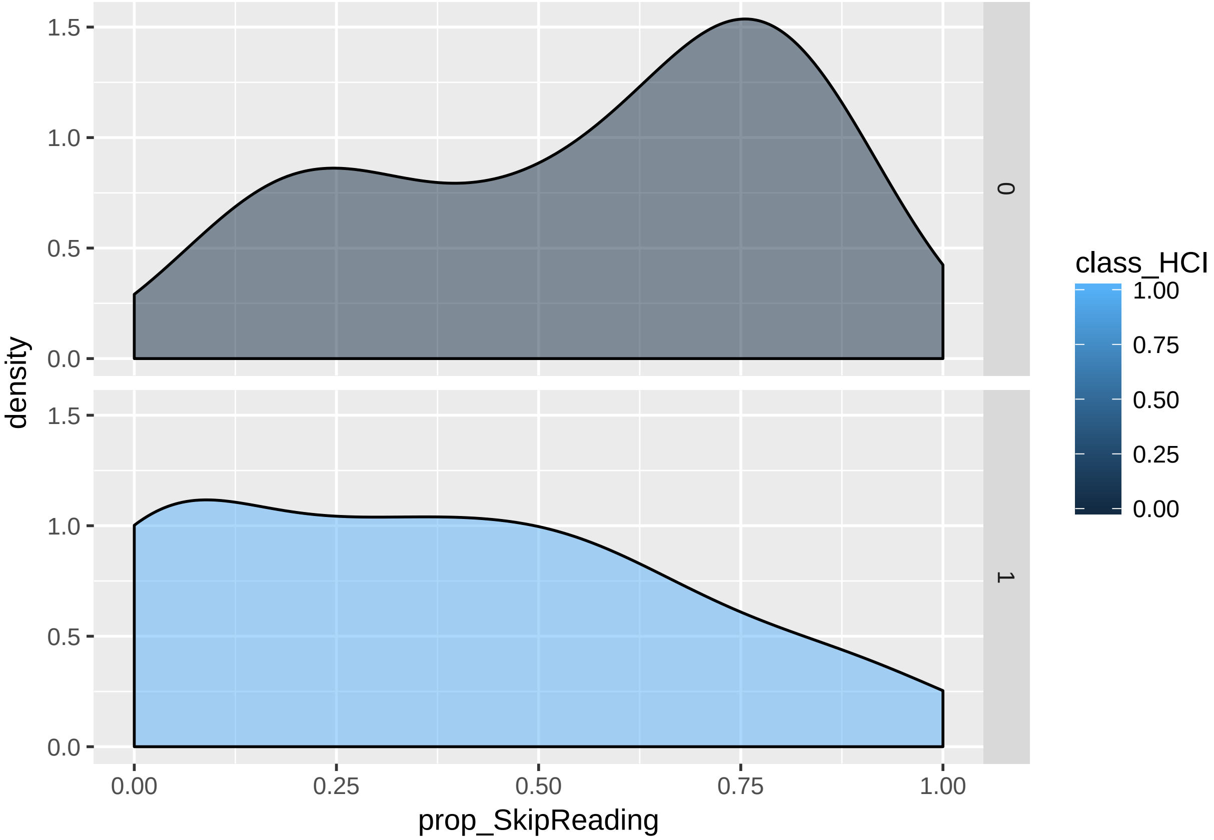
```
## Analysis of Variance Table
##
## Response: final_comp_grade
##           Df   Sum Sq   Mean Sq F value   Pr(>F)
## class_HCI  1 0.019252 0.0192524  10.066 0.002366 **
## Residuals 61 0.116672 0.0019127
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
t.test(final_comp_grade~class_HCI, data=grading_data)
```

```
##
##  Welch Two Sample t-test
##
## data:  final_comp_grade by class_HCI
## t = 2.8655, df = 24.039, p-value = 0.008514
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.01101976 0.06774636
## sample estimates:
## mean in group 0 mean in group 1
##       0.9072176       0.8678346
```

- Based on the ANOVA test above, we see that p-value $< 0.05$, we thus reject the null hypothesis H0 and accept the H1 that the 2 class groups means are statistically not equal.

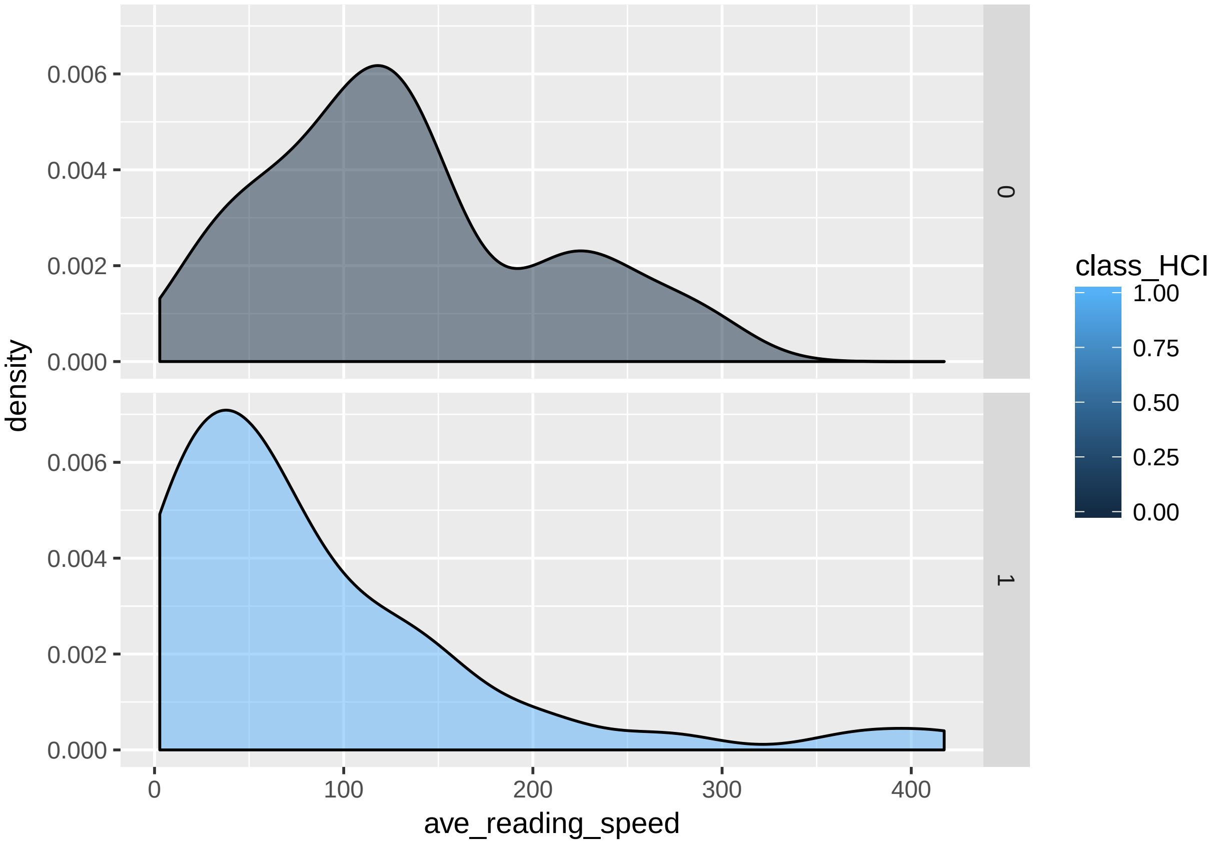
```r
#look at reading behavior
ggplot(grading_data, aes(x = prop_SkipReading, fill = class_HCI)) +
  geom_density(alpha = 0.5)  + facet_grid(class_HCI ~ .)
```



```r
t.test(prop_SkipReading~class_HCI, data=grading_data) #p=0.016
```

```
##
##  Welch Two Sample t-test
##
## data:  prop_SkipReading by class_HCI
## t = 2.5302, df = 32.466, p-value = 0.01644
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.03785247 0.34957183
## sample estimates:
## mean in group 0 mean in group 1
##       0.5590180       0.3653058
```

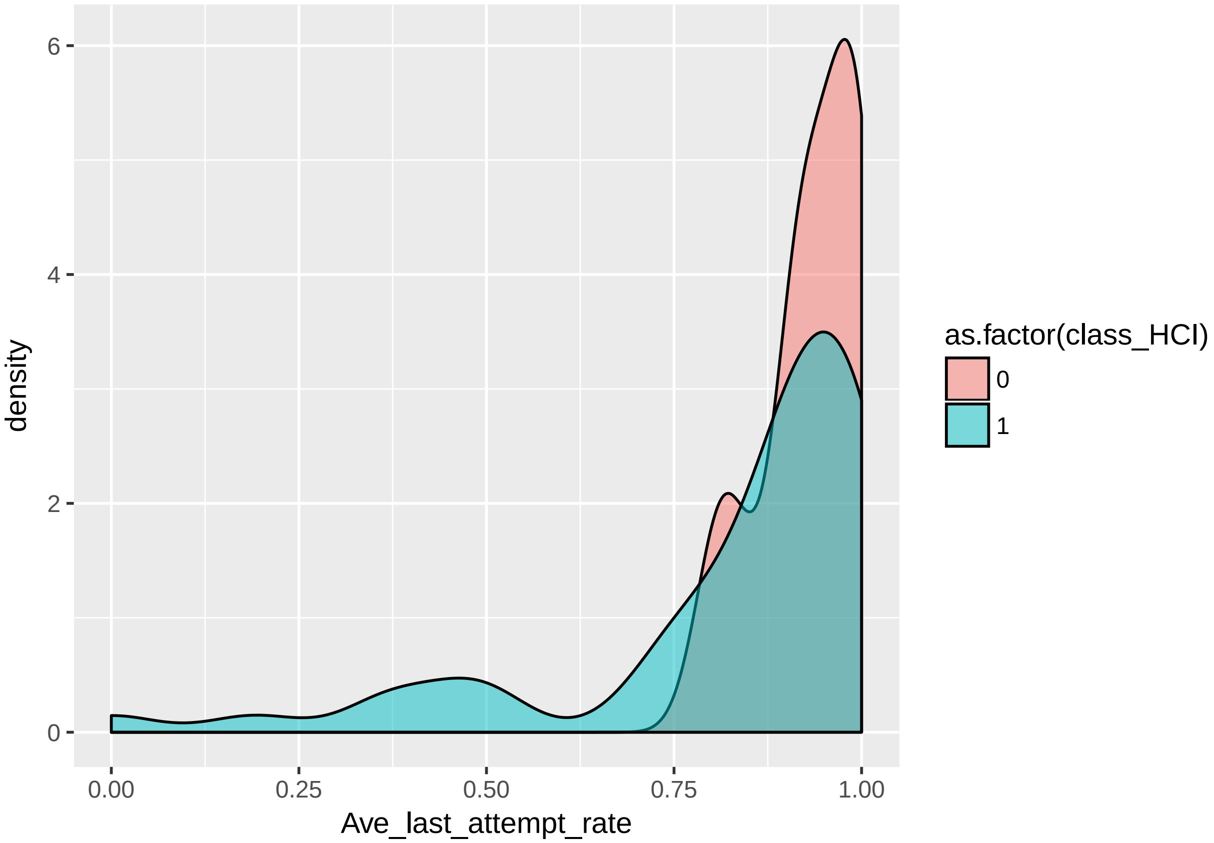
```r
#reading speed
ggplot(grading_data, aes(x = ave_reading_speed, fill = class_HCI)) +
  geom_density(alpha = 0.5)  + facet_grid(class_HCI ~ .)
```



```r
t.test(ave_reading_speed~class_HCI, data=grading_data) #p=0.04951
```

```
##
##  Welch Two Sample t-test
##
## data:  ave_reading_speed by class_HCI
## t = 2.0362, df = 34.327, p-value = 0.04951
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   0.1043502 91.5859689
## sample estimates:
## mean in group 0 mean in group 1
##       131.03287        85.18771
```

```r
ggplot(grading_data, aes(x = Ave_last_attempt_rate, fill = as.factor(class_HCI))) +
  geom_density(alpha = 0.5) # + facet_grid(class_HCI ~ .)
```



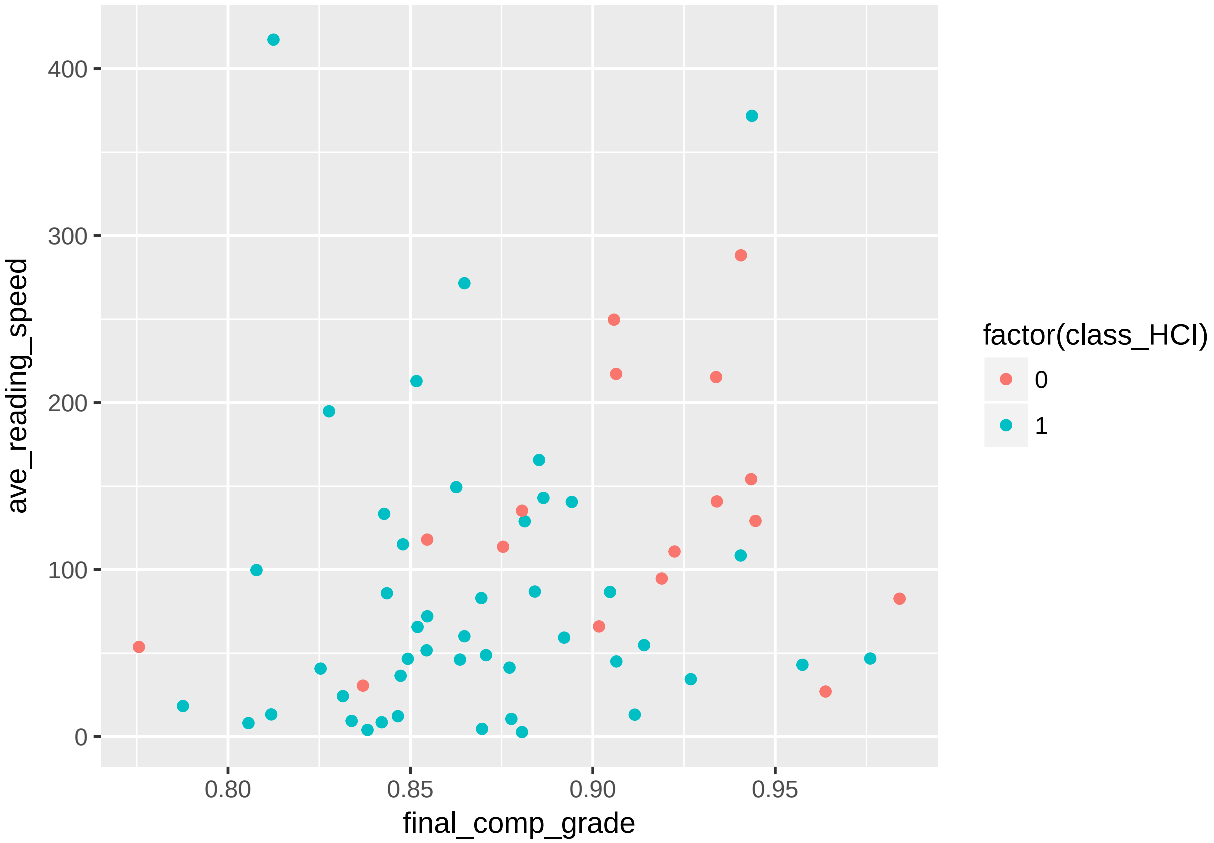
```
t.test(ave_attempt_rate~class_HCI, data=grading_data) #p=0.04951
```

```
##
##  Welch Two Sample t-test
##
## data:  ave_attempt_rate by class_HCI
## t = -1.5185, df = 60.932, p-value = 0.1341
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.14909143  0.02039191
## sample estimates:
## mean in group 0 mean in group 1
##       0.4519135       0.5162632
```

```
#reading speed Vs final performance
cor(grading_data$final_comp_grade, grading_data$ave_reading_speed)
```

```
## [1] 0.1880539
```

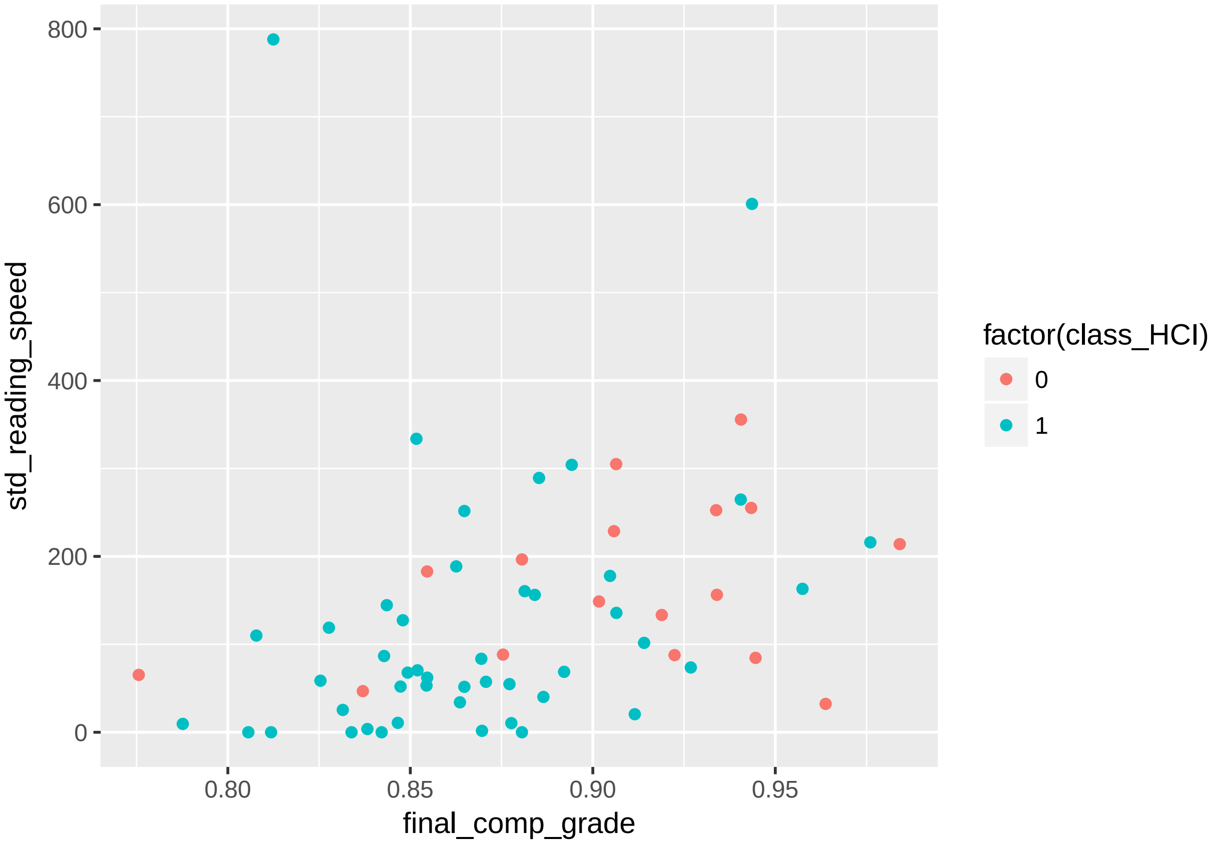
```
## [1] 0.1880

ggplot(grading_data, aes(x = final_comp_grade, y=ave_reading_speed)) + geom_point(aes(colour = factor(cl
```



```
cor(grading_data$final_comp_grade, grading_data$std_reading_speed)
```

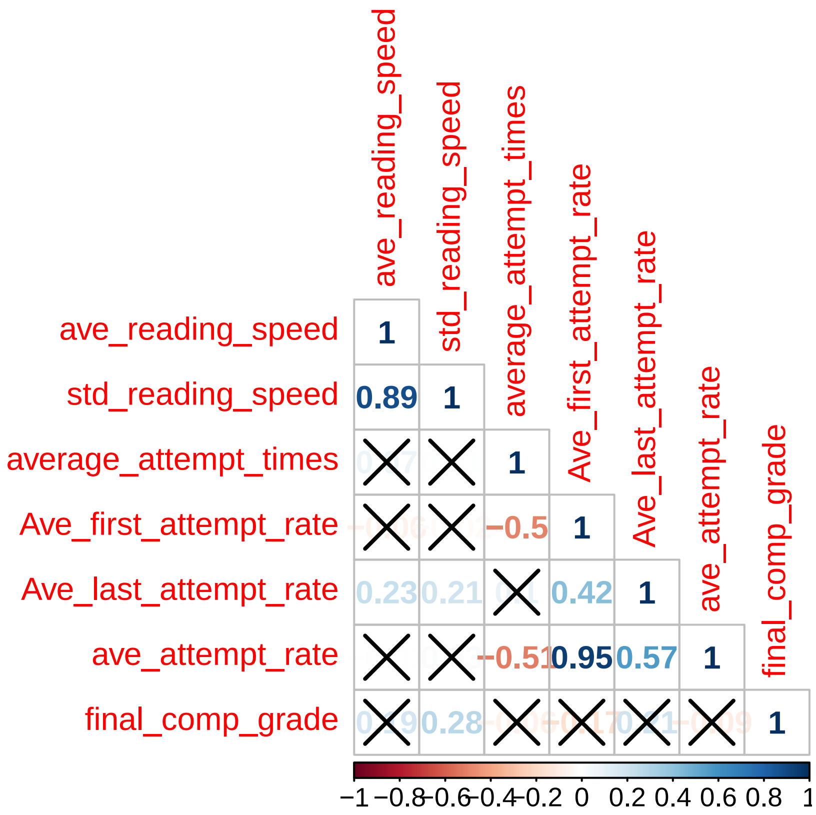
```
## [1] 0.2761949
```

```
## [1] 0.1880
ggplot(grading_data, aes(x = final_comp_grade, y=std_reading_speed)) + geom_point(aes(colour = factor(cl
```

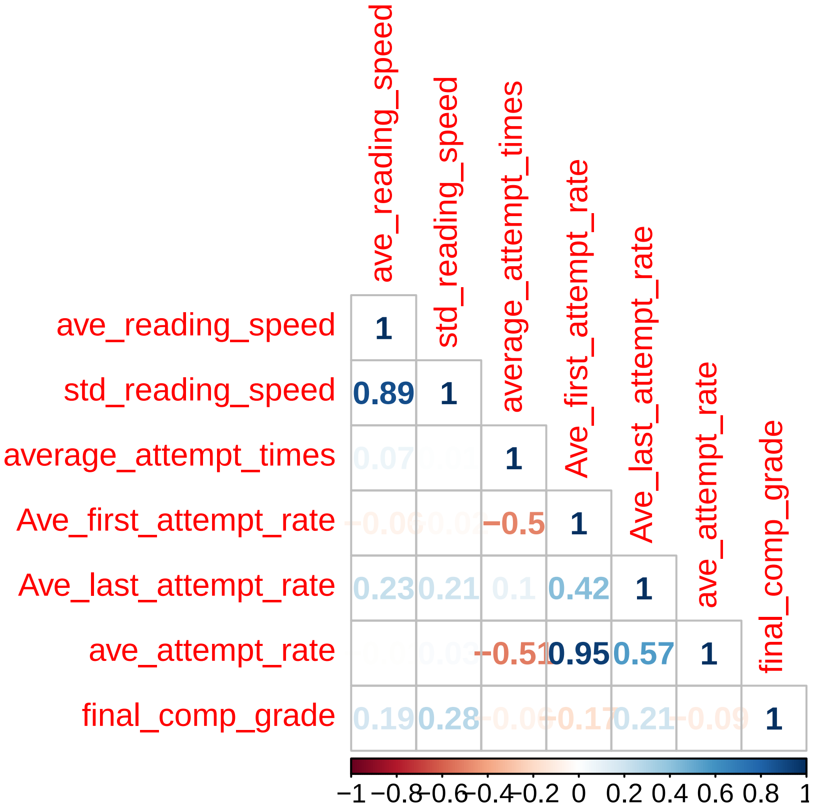


```
#reading speed and QA correction rate
#for correlations
library(corrplot)
corr_data = grading_data[c("ave_reading_speed","std_reading_speed",
                           "average_attempt_times", "Ave_first_attempt_rate",
                           "Ave_last_attempt_rate",
                           "ave_attempt_rate","final_comp_grade")]

cor.mtest <- function(mat, conf.level = 0.95){
  mat <- as.matrix(mat)
  n <- ncol(mat)
  p.mat <- lowCI.mat <- uppCI.mat <- matrix(NA, n, n)
  diag(p.mat) <- 0
  diag(lowCI.mat) <- diag(uppCI.mat) <- 1
  for(i in 1:(n-1)){
    for(j in (i+1):n){
      tmp <- cor.test(mat[,i], mat[,j], conf.level = conf.level)
      p.mat[i,j] <- p.mat[j,i] <- tmp$p.value
      lowCI.mat[i,j] <- lowCI.mat[j,i] <- tmp$conf.int[1]
      uppCI.mat[i,j] <- uppCI.mat[j,i] <- tmp$conf.int[2]
    }
  }
  return(list(p.mat, lowCI.mat, uppCI.mat))
}

res1 <- cor.mtest(na.omit(corr_data),0.99)
M <- cor(na.omit(corr_data))
```

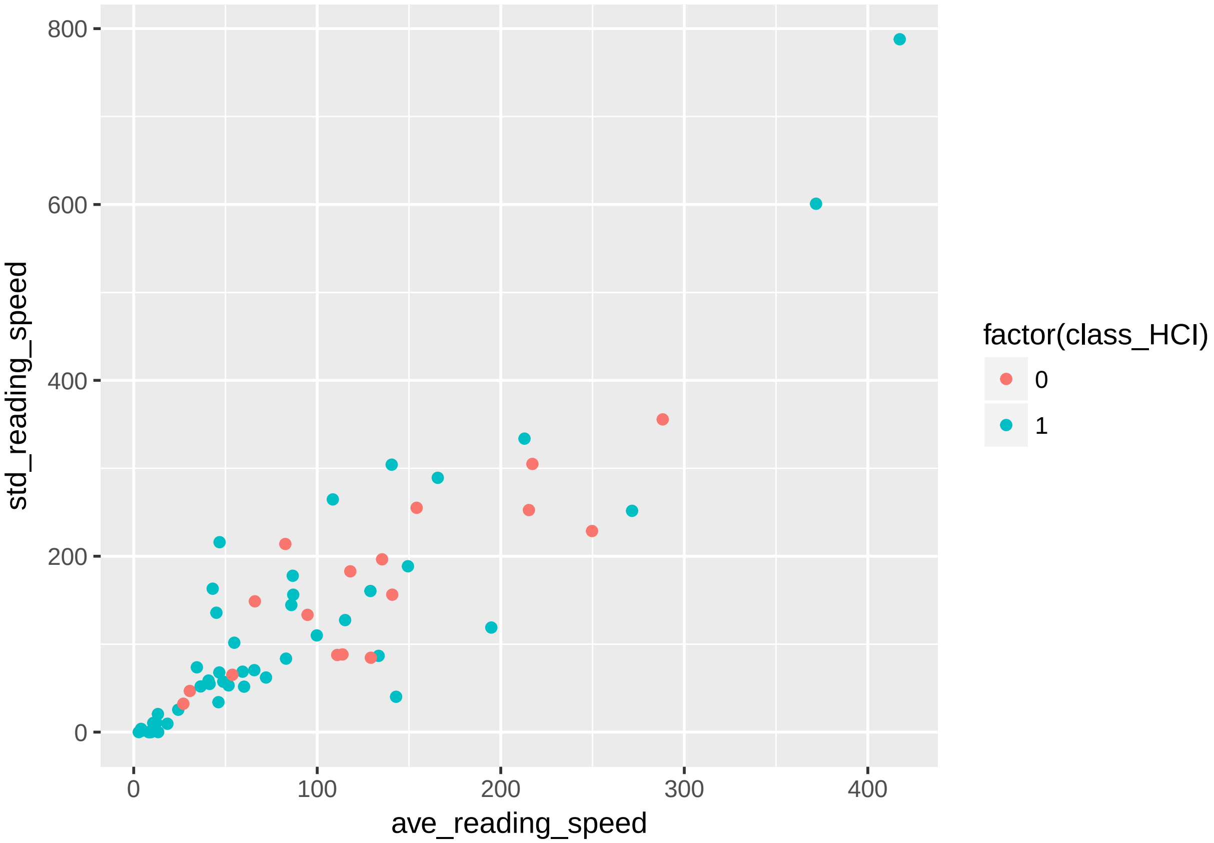
```r
corrplot(M, p.mat = res1[[1]],  type="lower",method="number",sig.level=0.1)
```



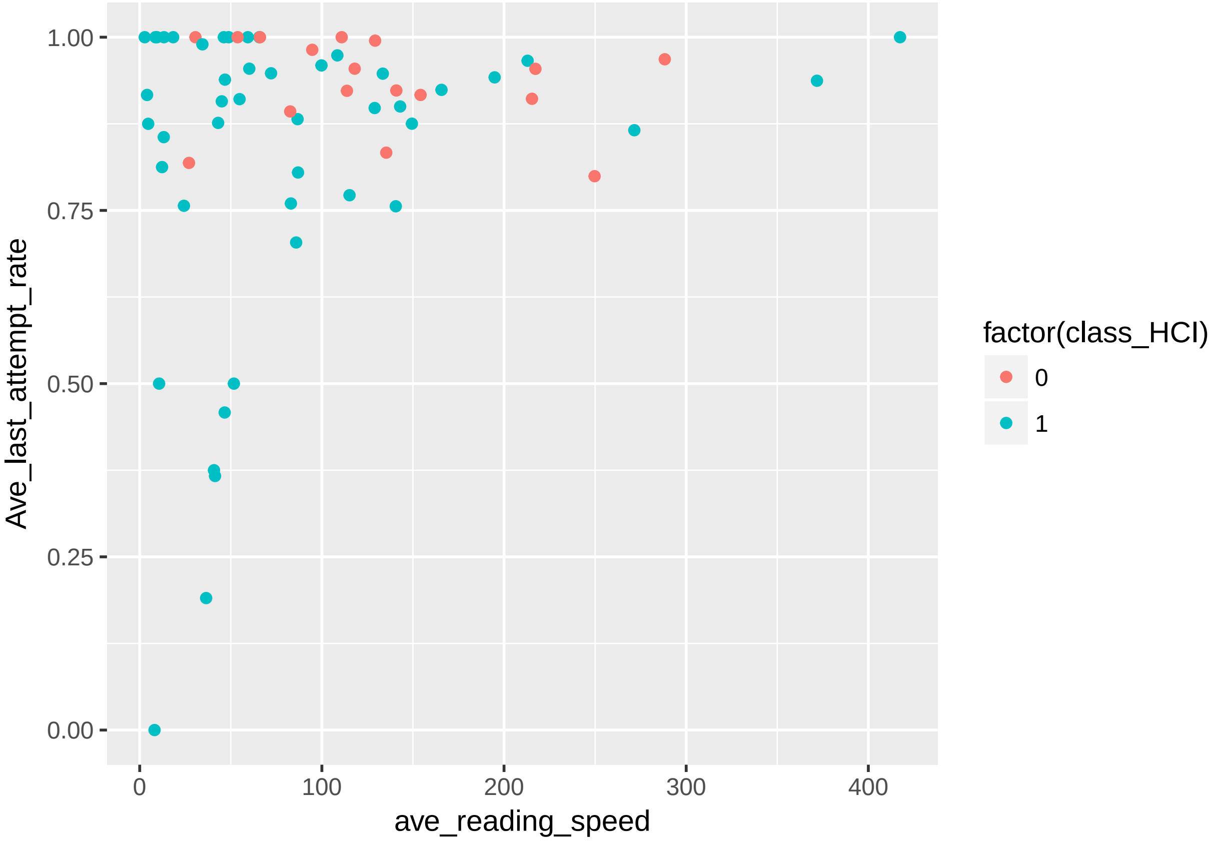
```r
corrplot(M,  type="lower",method="number",sig.level=0.1)
```



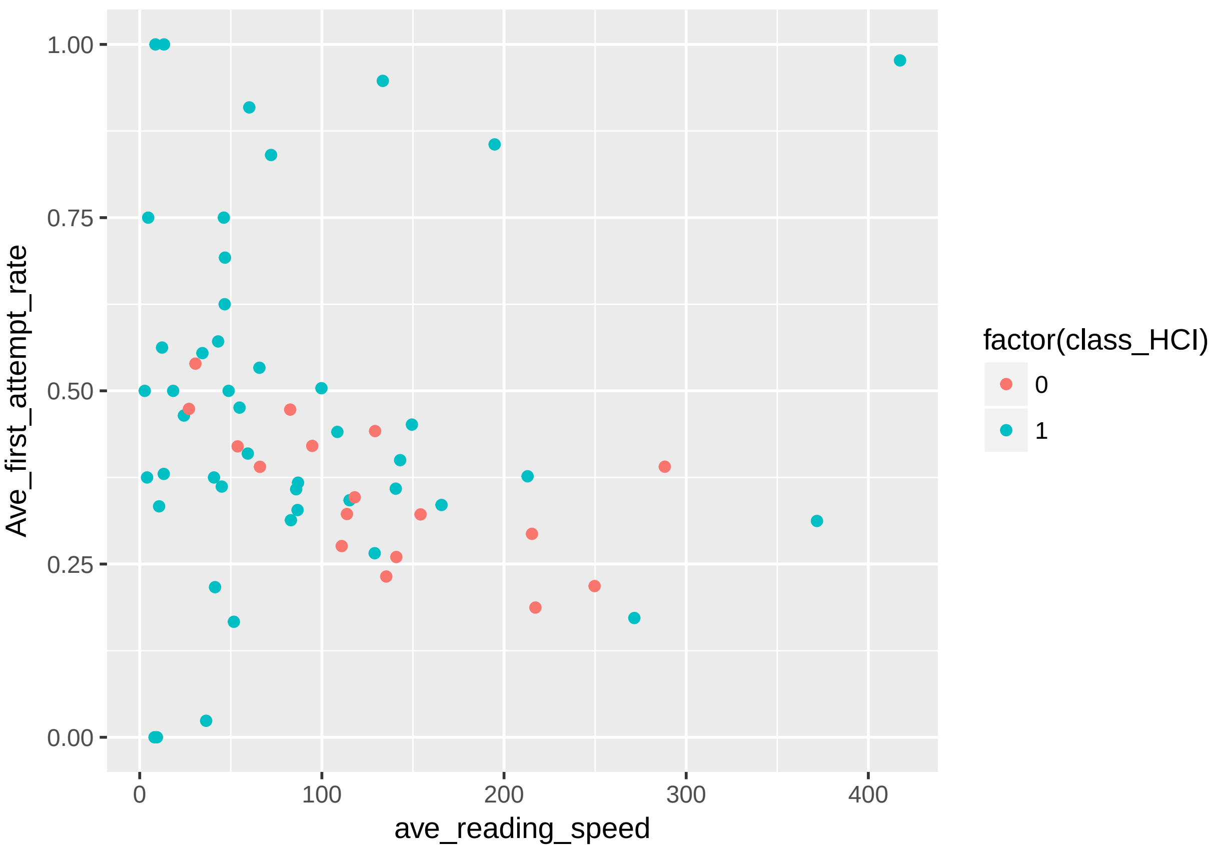
```
ggplot(grading_data, aes(x = ave_reading_speed, y=std_reading_speed)) + geom_point(aes(colour = factor(
```



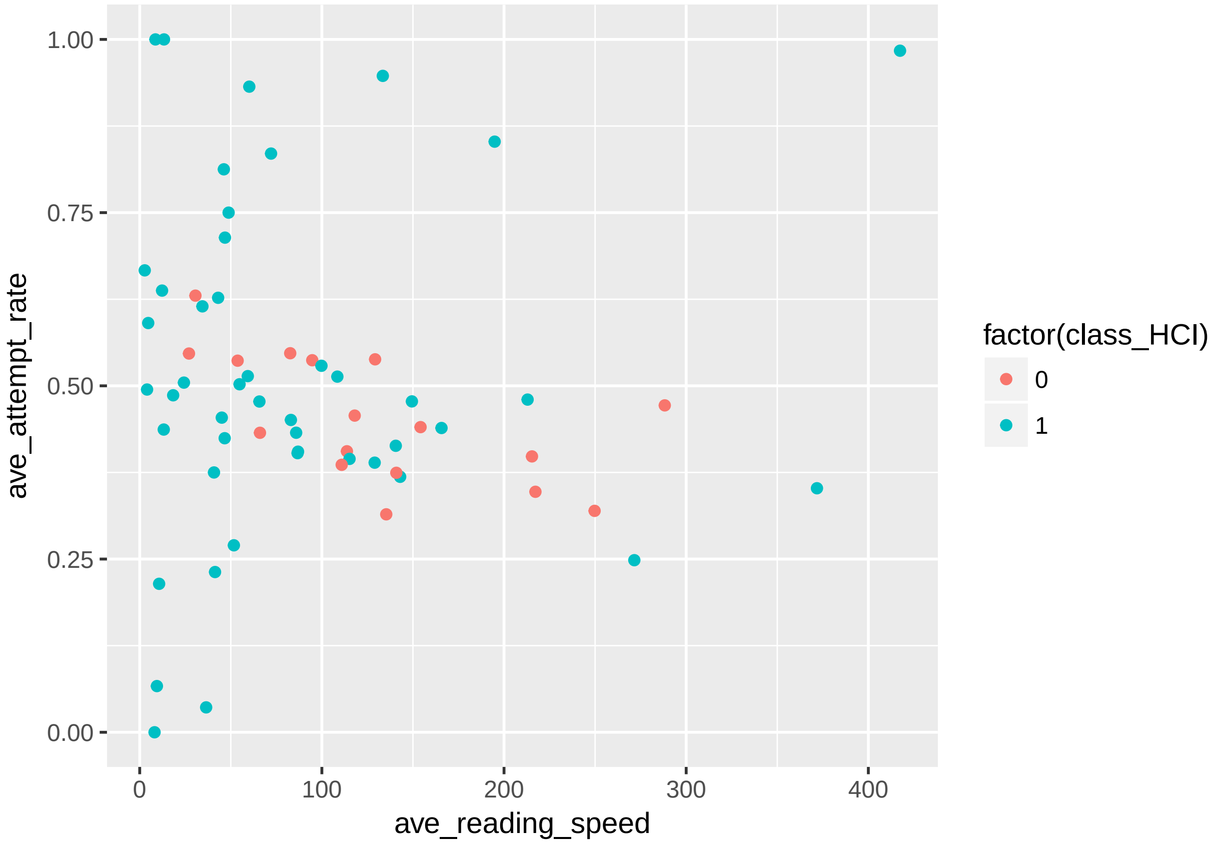
```
#ggplot(grading_data[which(grading_data$class_HCI==0),], aes(x = ave_reading_speed, y=std_reading_speed

ggplot(grading_data, aes(x = ave_reading_speed, y=Ave_last_attempt_rate)) + geom_point(aes(colour = fac
```



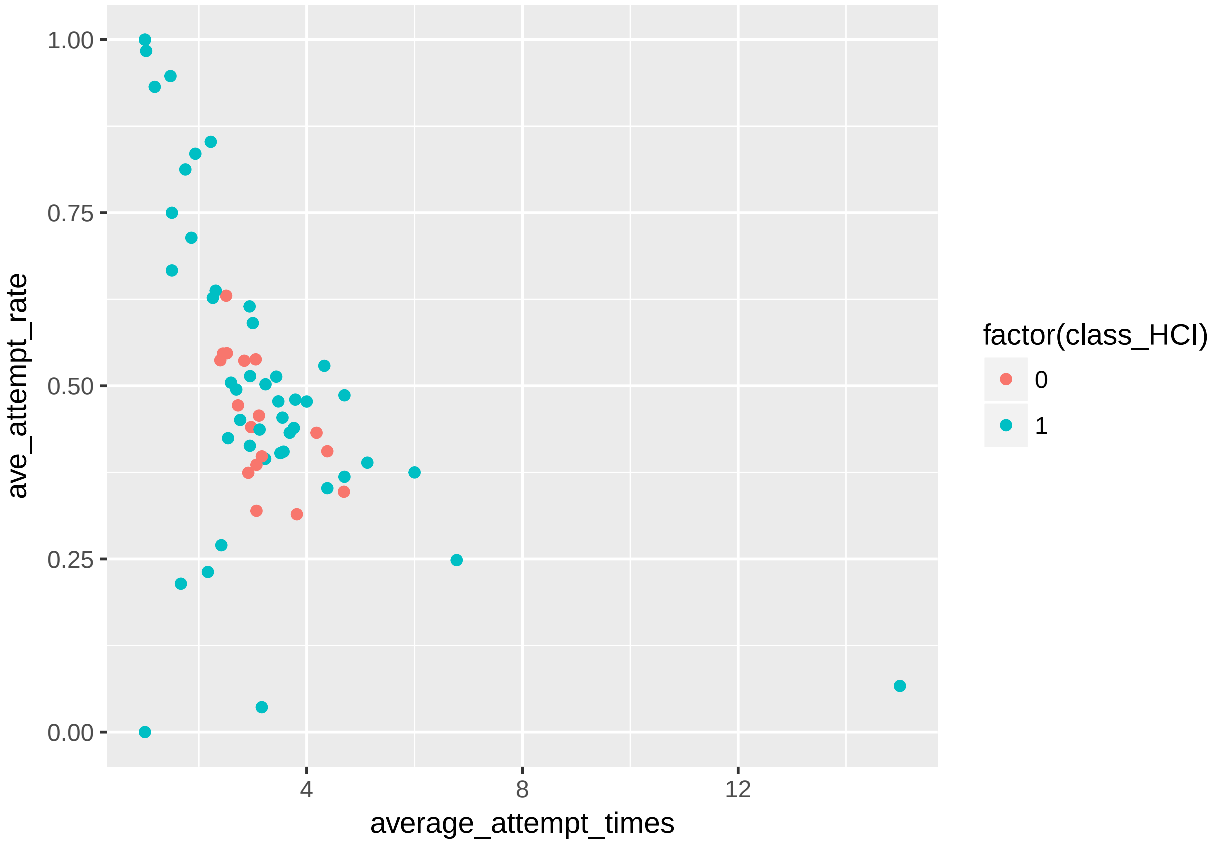
```
#ggplot(grading_data[which(grading_data$class_HCI==0),], aes(x = ave_reading_speed, y=Ave_last_attempt_

ggplot(grading_data, aes(x = ave_reading_speed, y=Ave_first_attempt_rate)) + geom_point(aes(colour = fa
```



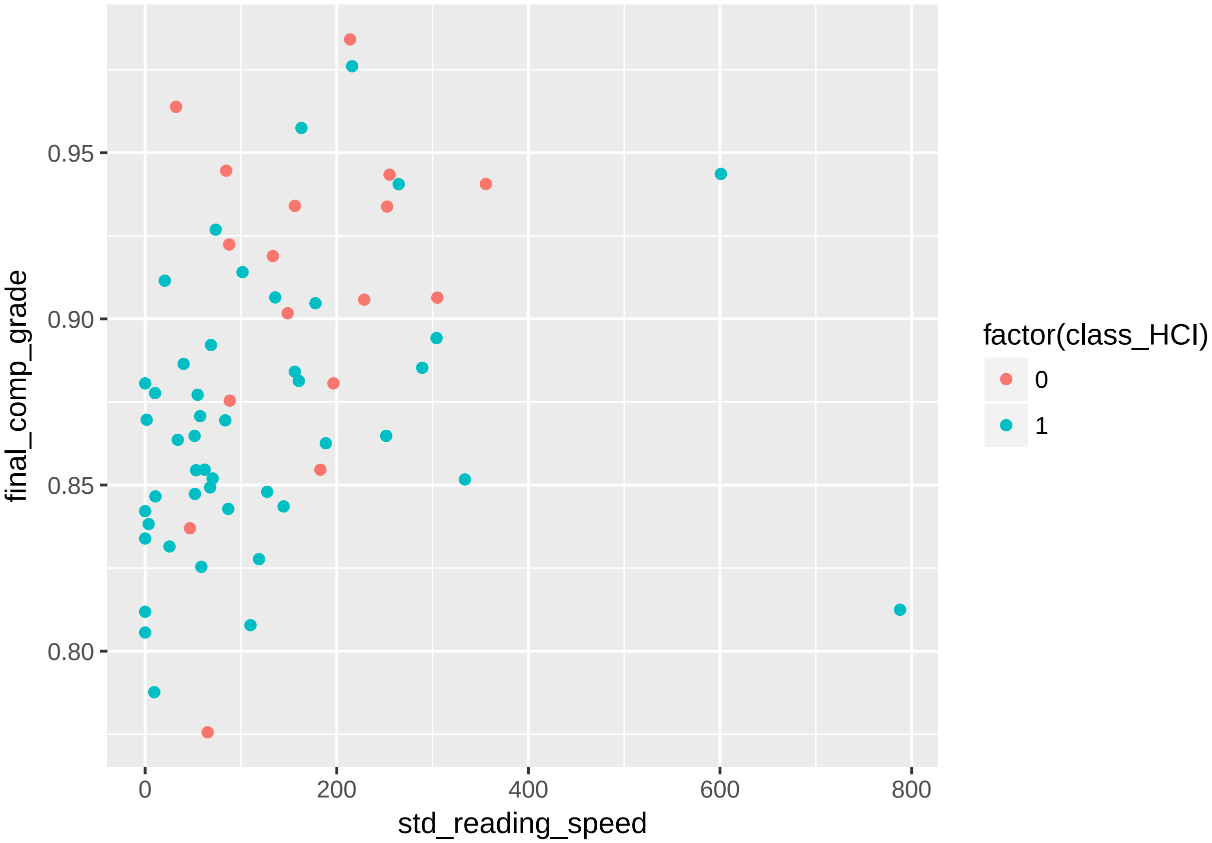
```
#ggplot(grading_data[which(grading_data$class_HCI==0),], aes(x = ave_reading_speed, y=Ave_first_attempt_

ggplot(grading_data, aes(x = ave_reading_speed, y=ave_attempt_rate)) + geom_point(aes(colour = factor(cl
```



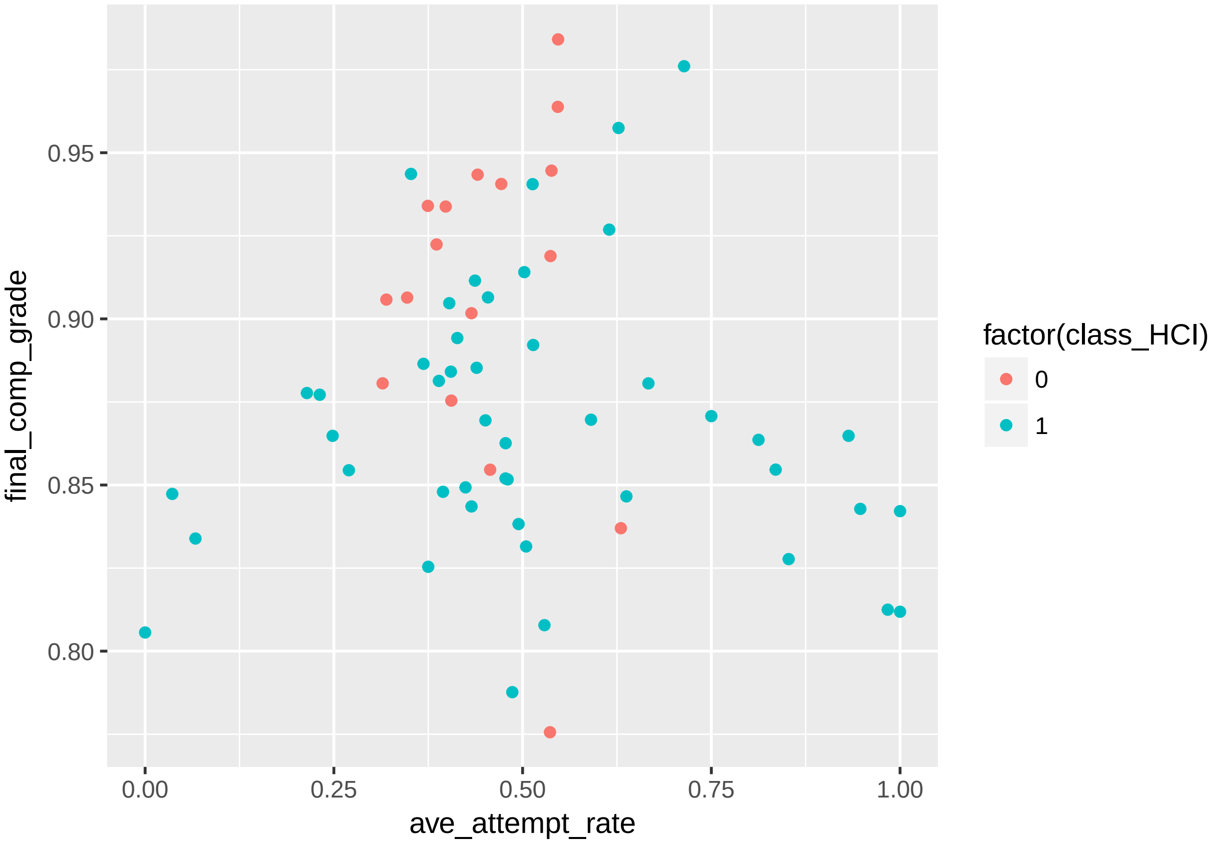
```
ggplot(grading_data, aes(x = average_attempt_times, y=ave_attempt_rate)) + geom_point(aes(colour = fact
```



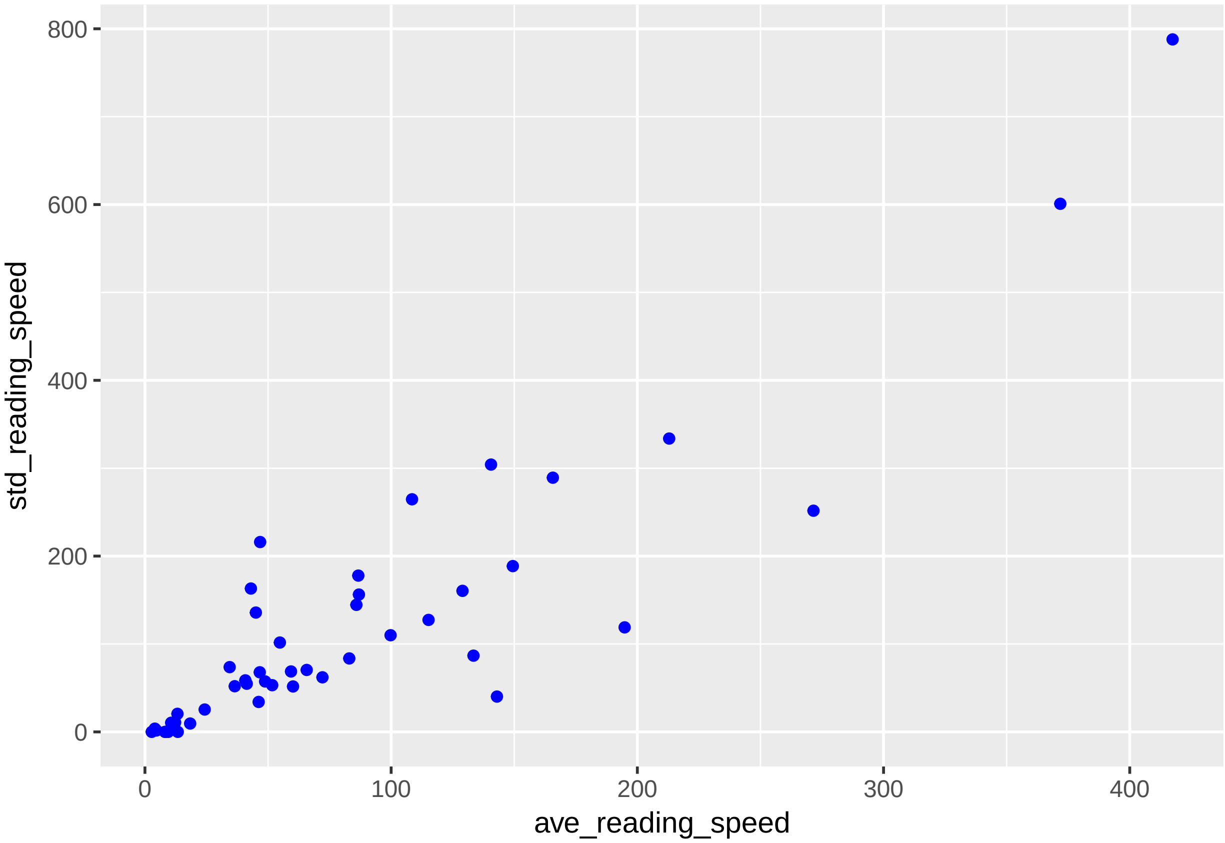
```
#correlation with final grade
ggplot(grading_data, aes(x = std_reading_speed, y=final_comp_grade)) + geom_point(aes(colour = factor(cl
```



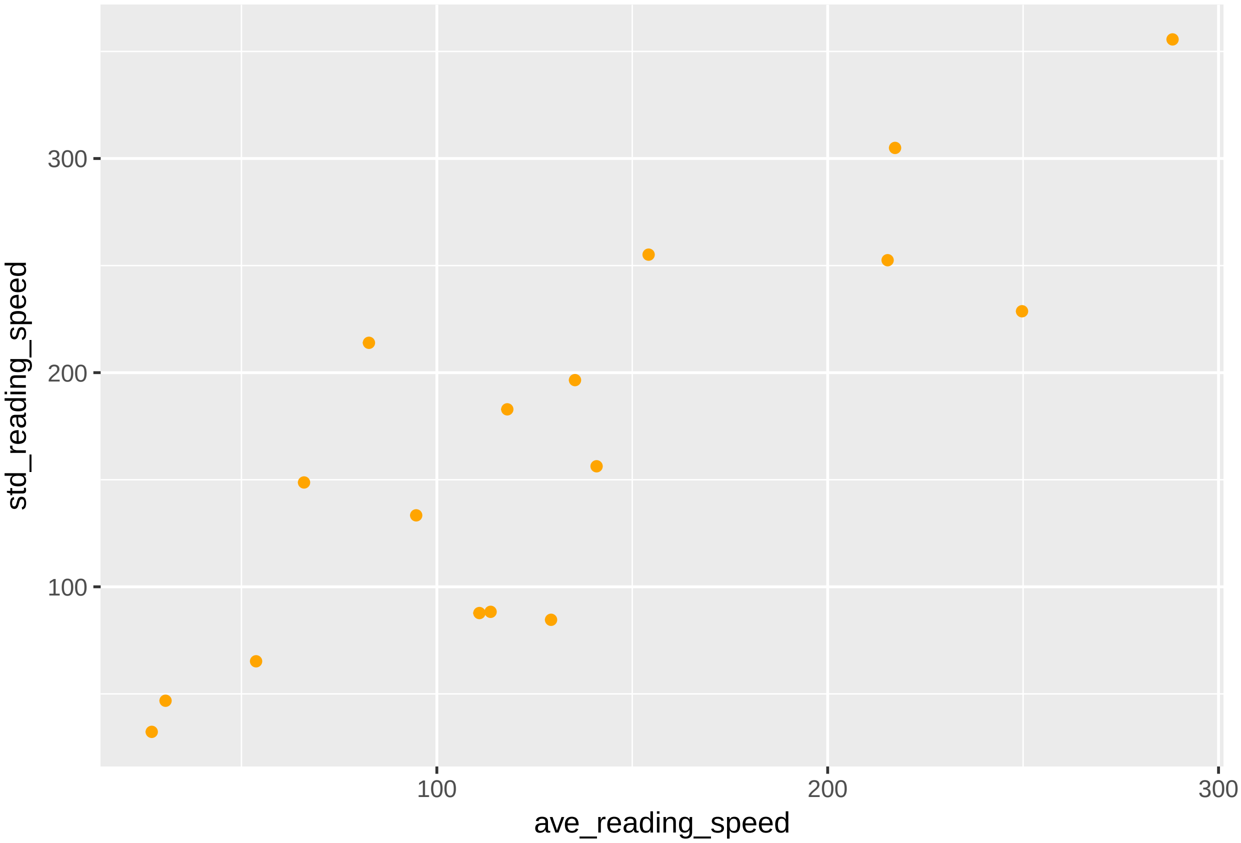
```
ggplot(grading_data, aes(x = ave_attempt_rate, y=final_comp_grade)) + geom_point(aes(colour = factor(cla
```



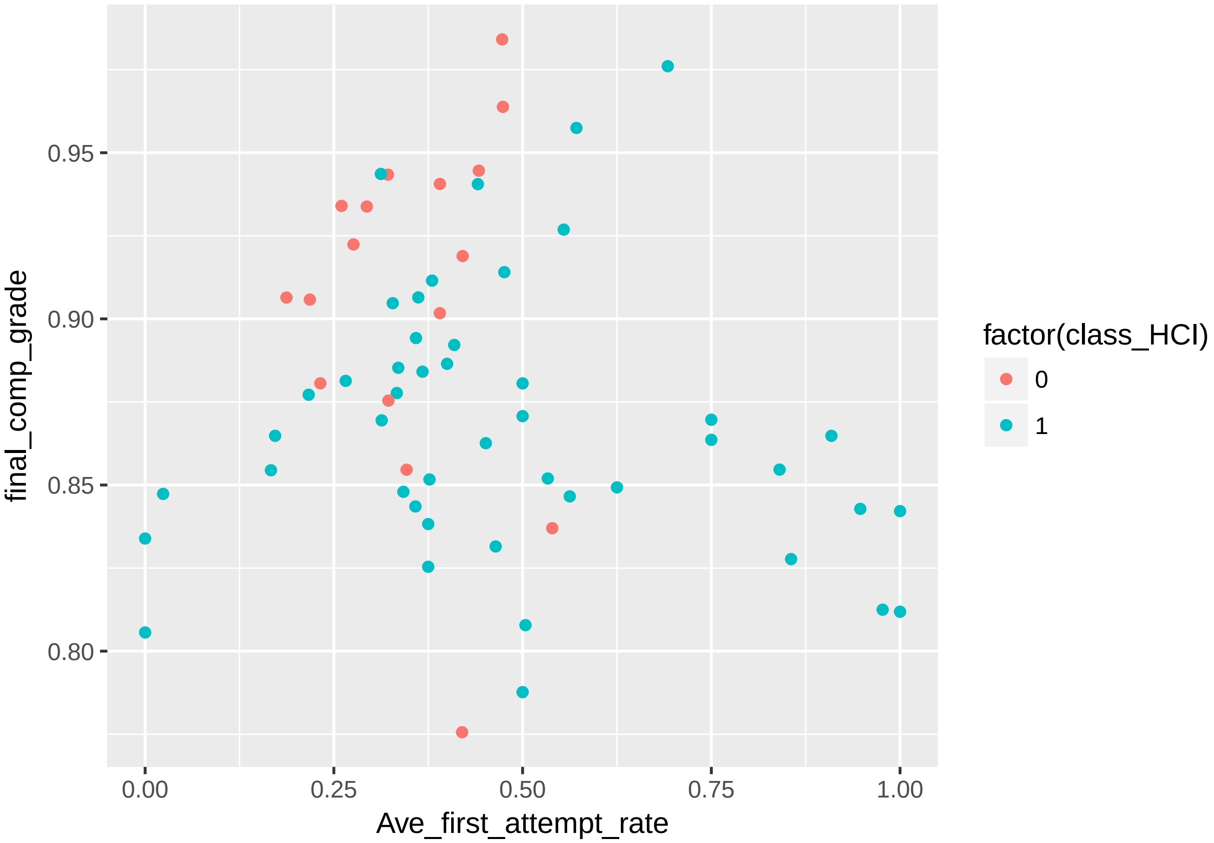
```
ggplot(grading_data[which(grading_data$class_HCI==1),], aes(x = ave_reading_speed, y=std_reading_speed)
```



```
ggplot(grading_data[which(grading_data$class_HCI==0),], aes(x = ave_reading_speed, y=std_reading_speed)
```



```
ggplot(grading_data, aes(x = Ave_first_attempt_rate, y=final_comp_grade)) + geom_point(aes(colour = fac
```



```
data = grading_data[c("ave_reading_speed","std_reading_speed","Average_nword","prop_SkipReading",
                      "prop_fullreading","num_question_answered","average_attempt_times","Std_average_at
                      "Ave_first_attempt_rate","Std_first_attempt_rate","Ave_last_attempt_rate","Std_las
                      "ave_attempt_rate","Std_ave_attempt.rate","class_HCI","Final_exam_grade", "final_c
                      )]
#data$Ave_reading_time = data$Ave_reading_time/3600
#data$Std_reading_time = data$Std_reading_time/3600

fit1 = lm(final_comp_grade ~ ave_reading_speed+std_reading_speed+Average_nword
          + prop_SkipReading + prop_fullreading
          + average_attempt_times+Std_average_attempts
          + Ave_first_attempt_rate + Std_first_attempt_rate
          +Ave_last_attempt_rate + Std_last_attempt_rate
          +ave_attempt_rate + Std_ave_attempt.rate
          + as.factor(class_HCI), data=data)
summary(fit1)
```

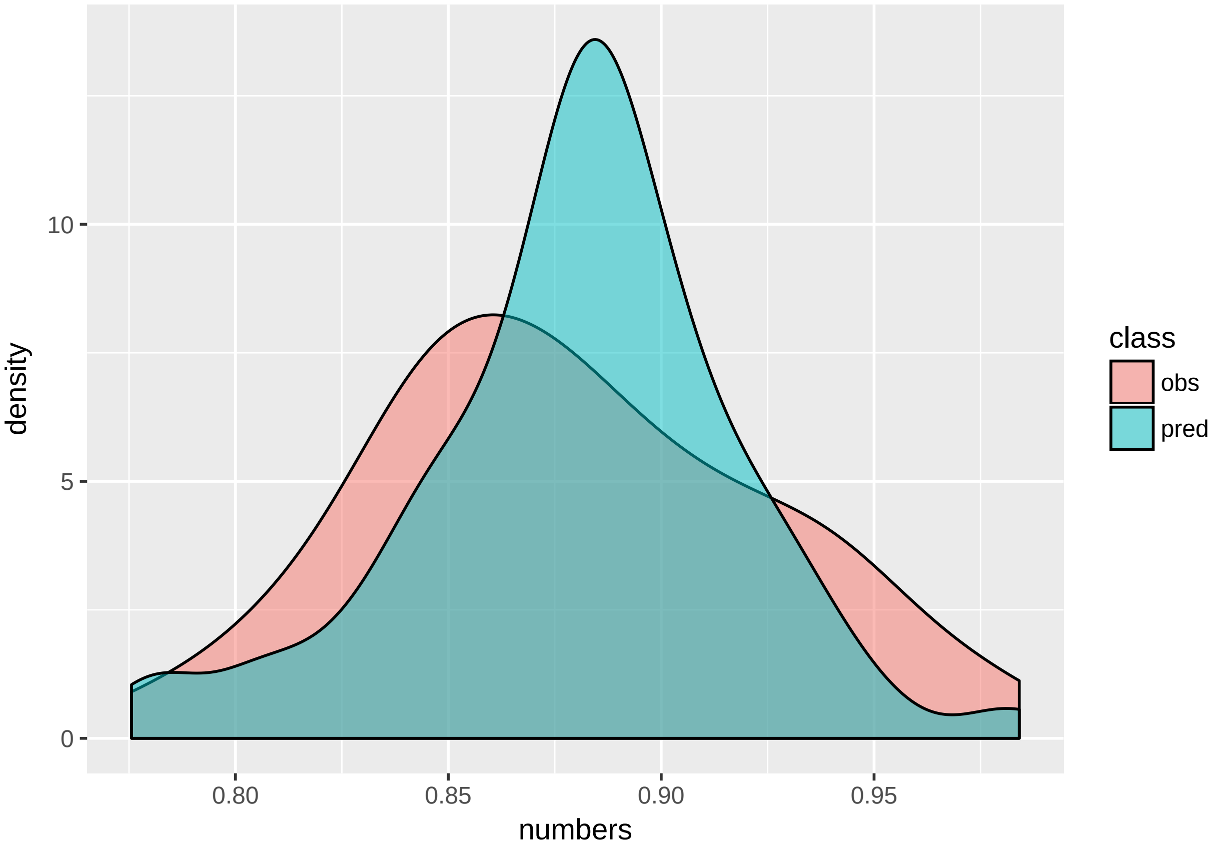
```
##
## Call:
## lm(formula = final_comp_grade ~ ave_reading_speed + std_reading_speed +
##     Average_nword + prop_SkipReading + prop_fullreading + average_attempt_times +
##     Std_average_attempts + Ave_first_attempt_rate + Std_first_attempt_rate +
##     Ave_last_attempt_rate + Std_last_attempt_rate + ave_attempt_rate +
##     Std_ave_attempt.rate + as.factor(class_HCI), data = data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.106308 -0.020367  0.003596  0.016660  0.068682
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             7.747e-01  4.646e-02  16.675  < 2e-16 ***
## ave_reading_speed      -4.246e-04  2.330e-04  -1.822  0.07467 .
## std_reading_speed       2.566e-04  1.043e-04   2.461  0.01749 *
## Average_nword           6.097e-07  1.497e-06   0.407  0.68552
## prop_SkipReading        5.963e-02  4.116e-02   1.449  0.15391
## prop_fullreading        5.603e-02  3.233e-02   1.733  0.08954 .
## average_attempt_times  -6.699e-03  7.313e-03  -0.916  0.36425
## Std_average_attempts   -2.474e-03  6.743e-03  -0.367  0.71533
## Ave_first_attempt_rate -8.430e-03  9.884e-02  -0.085  0.93238
## Std_first_attempt_rate -1.276e-02  1.100e-01  -0.116  0.90816
## Ave_last_attempt_rate   1.622e-01  5.420e-02   2.993  0.00435 **
## Std_last_attempt_rate   1.192e-01  4.879e-02   2.442  0.01833 *
## ave_attempt_rate       -9.724e-02  1.208e-01  -0.805  0.42480
## Std_ave_attempt.rate    2.851e-02  1.461e-01   0.195  0.84615
## as.factor(class_HCI)1  -3.111e-02  1.475e-02  -2.109  0.04021 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03919 on 48 degrees of freedom
## Multiple R-squared:  0.4576, Adjusted R-squared:  0.2995
## F-statistic: 2.893 on 14 and 48 DF,  p-value: 0.003117
```

```
## leave-one-out cross validation
n = length(data$final_comp_grade)
error = dim(n)
predicts = dim(n)
obss = dim(n)
for (k in 1:n) {
  train1 = c(1:n)
  train = train1[train1!=k] ## pick elements that are different from k
  m1 = lm(final_comp_grade ~ ave_reading_speed+std_reading_speed+Average_nword
          + prop_SkipReading + prop_fullreading
          + average_attempt_times+Std_average_attempts
          + Ave_first_attempt_rate + Std_first_attempt_rate
          +Ave_last_attempt_rate + Std_last_attempt_rate
          +ave_attempt_rate + Std_ave_attempt.rate
          + as.factor(class_HCI), data=data[train ,])
  pred = predict(m1, newdat=data[-train ,])
  predicts[k] = pred
  obs = data$final_comp_grade[-train]
  obss[k] = obs
  error[k] = obs-pred
}
rmse1=sqrt(mean(error^2))
rmse1 ## root mean square error 0.04695582
```

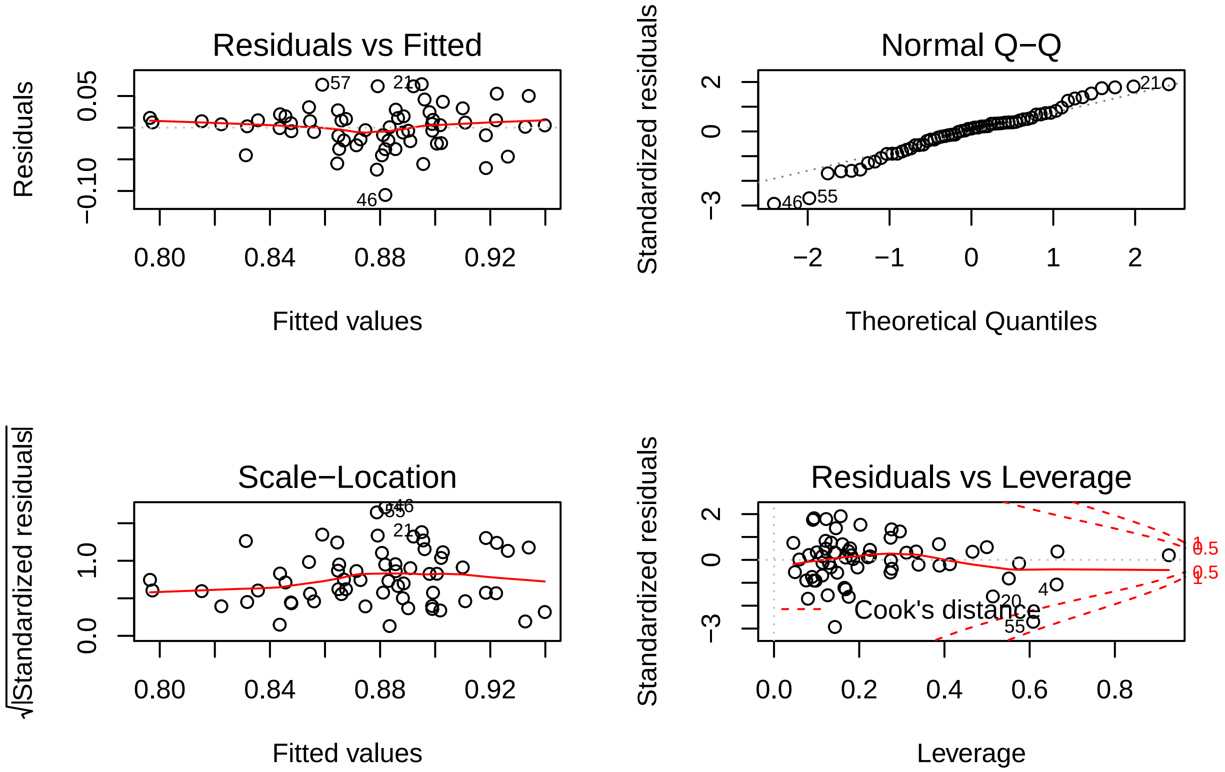
```
## [1] 0.04695582
```

```
prediction = data.frame(predicts)
colnames(prediction) = "numbers"
prediction$class = "pred"
```

```
observation = data.frame(obss)
colnames(observation) = "numbers"
observation$class="obs"

predictions_comp = rbind(prediction, observation)
ggplot(predictions_comp, aes(x = numbers, fill = class)) +
  geom_density(alpha = 0.5)
```



```
## plot regression diagnostics
par(mfrow=c(2,2))
plot(fit1)
```



- **Regression Diagnostics**
- Normality: The Normal Q-Q plot is a probability plot of the standardized residuals against the values that would be expected under normality. If we met the normality assumption, the points on this graph should fall on the straight 45-degree line. In our case, it is generally normally distributed.
- Independence: We believe that all students should be independent to each other.
- Linearity: If the dependent variable is linearly related to the independent variables, there should be no systematic relationship between the residuals and the predicted (that is, fitted) values. In other words, the model should capture all the systematic variance present in the data, leaving nothing but random noise. In the **Residuals vs Fitted graph** (upper left), we observed a little bit curved relationship, which suggests that we may want to add a quadratic term to the regression.
- Homoscedasticity: The points in the **Scale-Location graph** (bottom left) seems to be a random band around a horizontal line.

*** Here I tried 3 ways to improve the current linear regression.**

### Standard linear regression with significant terms, and count variable log transformed

```
## using improved linear model ###
fit_Linear = lm(final_comp_grade ~ ave_reading_speed+std_reading_speed
          + average_attempt_times
          + Ave_first_attempt_rate + Std_first_attempt_rate
          +Ave_last_attempt_rate + Std_last_attempt_rate
          + as.factor(class_HCI), data=data)
summary(fit_Linear)
```

```
##
## Call:
```

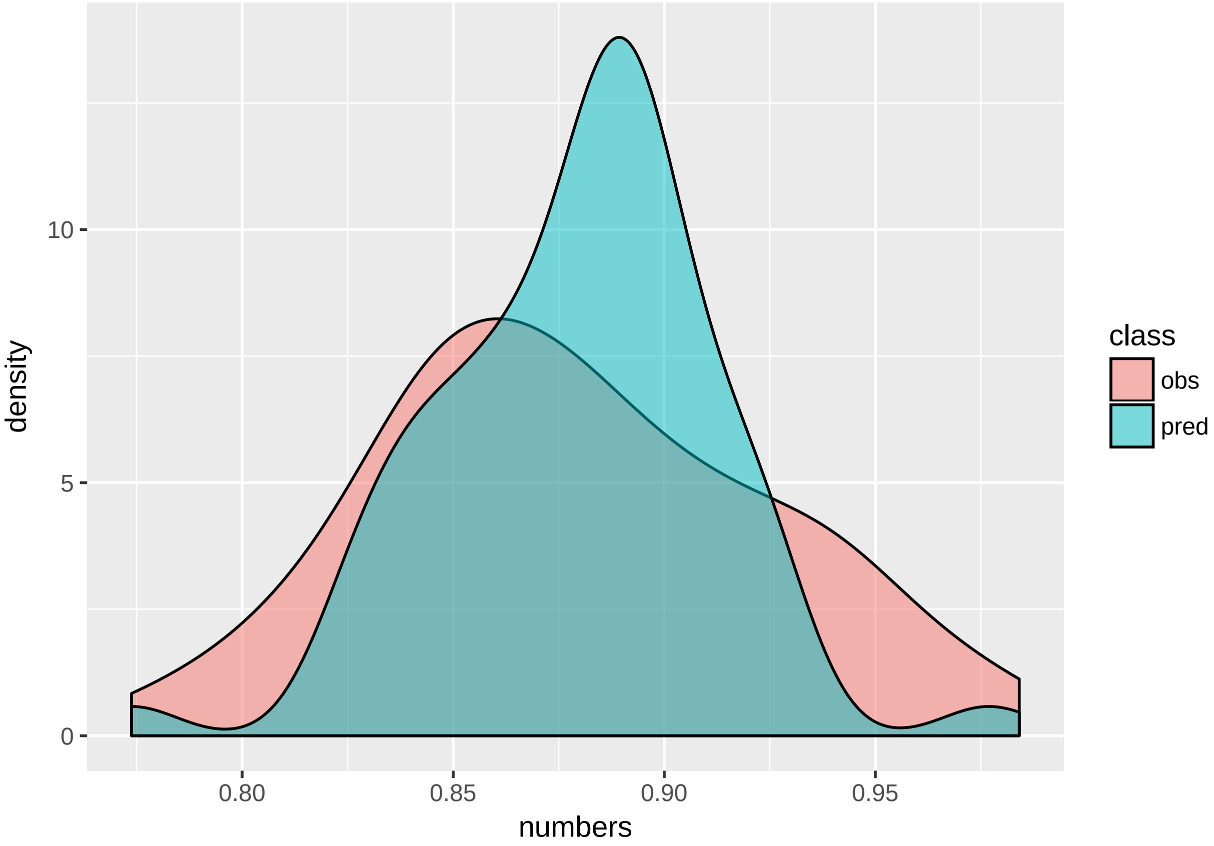
```
## lm(formula = final_comp_grade ~ ave_reading_speed + std_reading_speed +
##     average_attempt_times + Ave_first_attempt_rate + Std_first_attempt_rate +
##     Ave_last_attempt_rate + Std_last_attempt_rate + as.factor(class_HCI),
##     data = data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.117101 -0.020409  0.004222  0.018249  0.076703
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             8.266e-01  3.334e-02  24.795  < 2e-16 ***
## ave_reading_speed      -2.788e-04  1.272e-04  -2.193  0.03267 *
## std_reading_speed       1.953e-04  7.771e-05   2.513  0.01500 *
## average_attempt_times  -6.775e-03  3.622e-03  -1.871  0.06682 .
## Ave_first_attempt_rate -8.096e-02  3.583e-02  -2.260  0.02790 *
## Std_first_attempt_rate -1.652e-02  4.274e-02  -0.386  0.70065
## Ave_last_attempt_rate   1.284e-01  4.197e-02   3.059  0.00345 **
## Std_last_attempt_rate   1.120e-01  4.226e-02   2.650  0.01055 *
## as.factor(class_HCI)1  -2.515e-02  1.351e-02  -1.862  0.06801 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03886 on 54 degrees of freedom
## Multiple R-squared:  0.4002, Adjusted R-squared:  0.3113
## F-statistic: 4.504 on 8 and 54 DF,  p-value: 0.000311
```

```
## leave-one-out cross validation
n = length(data$final_comp_grade)
error = dim(n)
predicts = dim(n)
obss = dim(n)
for (k in 1:n) {
  train1 = c(1:n)
  train = train1[train1!=k] ## pick elements that are different from k
  m1 = lm(final_comp_grade ~ ave_reading_speed+std_reading_speed
          #+ prop_SkipReading + prop_fullreading
          + log(average_attempt_times)#+Std_average_attempts
          + Ave_first_attempt_rate + Std_first_attempt_rate
          +Ave_last_attempt_rate + Std_last_attempt_rate
          #+ave_attempt_rate + Std_ave_attempt.rate
          + as.factor(class_HCI), data=data[train ,])
  pred = predict(m1, newdat=data[-train ,])
  predicts[k] = pred
  obs = data$final_comp_grade[-train]
  obss[k] = obs
  error[k] = obs-pred
}
rmse_Linear=sqrt(mean(error^2))
rmse_Linear ## root mean square error 0.04475396
```

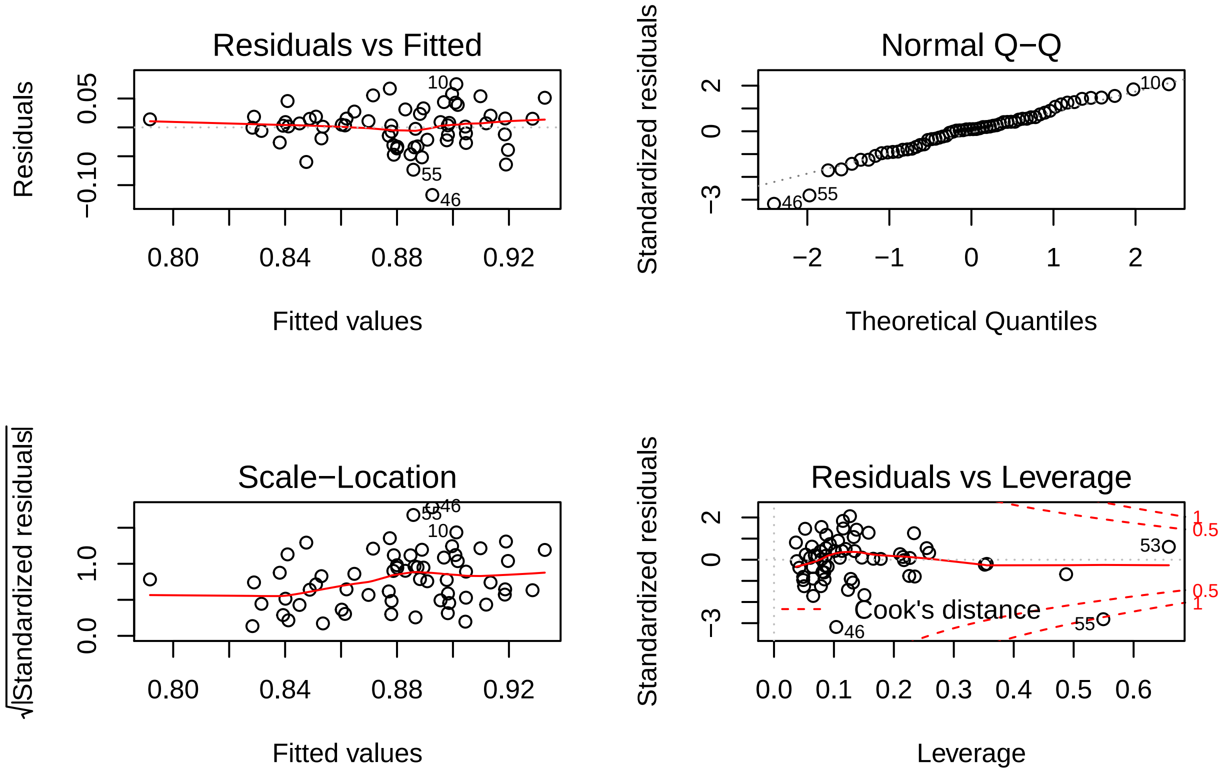
```
## [1] 0.04475396
```

```
prediction = data.frame(predicts)
colnames(prediction) = "numbers"
```

```
prediction$class = "pred"
observation = data.frame(obss)
colnames(observation) = "numbers"
observation$class="obs"

predictions_comp = rbind(prediction, observation)
ggplot(predictions_comp, aes(x = numbers, fill = class)) +
  geom_density(alpha = 0.5)
```



```
par(mfrow=c(2,2))
plot(m1)
```



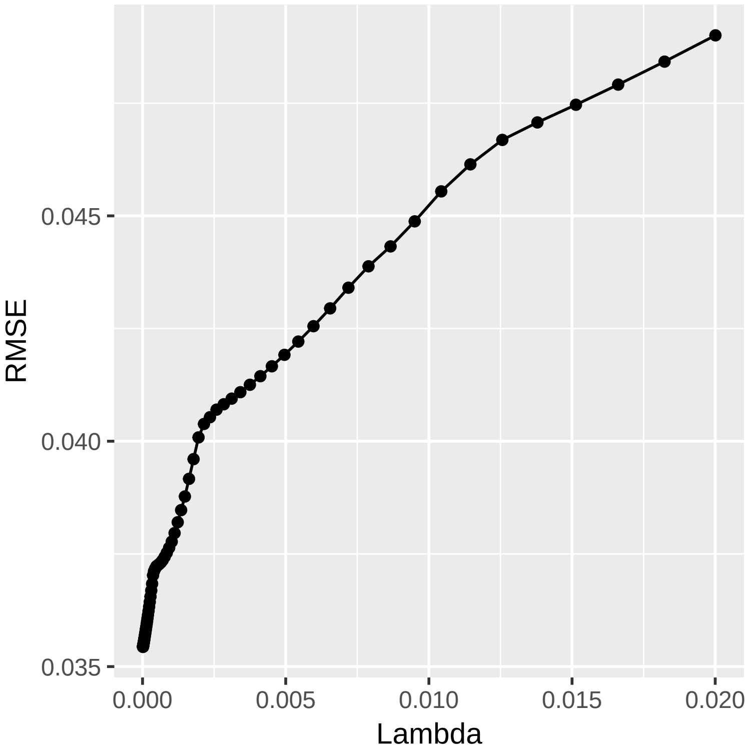
2. We then tried to improve the model using non-lieanr term with regularization. We add the quadratic term for all the numerical variables into the basic linear regression with all variables. We also did regularization on this model. After we found the best lambda on regularization term, we then able to calcuate the root mean square error: rmse= 0.042 by applying the leave-one-out cross validation.

```
##using non-linear + regularization##
library(glmnet)
```

```
## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-5
```

```
x_factor = as.factor(data$class_HCI)

#numerical variable into quadratic term
ave_reading_speed = data.frame(poly(data$ave_reading_speed, degree = 2))
colnames(ave_reading_speed) <- c("ave_reading_speed1", "ave_reading_speed2")

std_reading_speed = data.frame(poly(data$std_reading_speed, degree = 2))
colnames(std_reading_speed) <- c("std_reading_speed1", "std_reading_speed2")

Ave_first_attempt_rate = data.frame(poly(data$Ave_first_attempt_rate, degree = 2))
colnames(Ave_first_attempt_rate) <- c("Ave_first_attempt_rate1", "Ave_first_attempt_rate2")

Std_first_attempt_rate = data.frame(poly(data$Std_first_attempt_rate, degree = 2))
colnames(Std_first_attempt_rate) <- c("Std_first_attempt_rate1", "Std_first_attempt_rate2")

Ave_last_attempt_rate = data.frame(poly(data$Ave_last_attempt_rate, degree = 2))
colnames(Ave_last_attempt_rate) <- c("Ave_last_attempt_rate1", "Ave_last_attempt_rate2")
```

```r
Std_last_attempt_rate = data.frame(poly(data$Std_last_attempt_rate, degree = 2))
colnames(Std_last_attempt_rate) <- c("Std_last_attempt_rate1", "Std_last_attempt_rate2")

poly_data = cbind(ave_reading_speed, std_reading_speed, Ave_first_attempt_rate, Std_first_attempt_rate,A

#final data
x = data.matrix(data.frame(x_factor,poly_data))
y=data$final_comp_grade
df <- data.frame(X = x, Y = y)

####--- step1. Find the best lambda ----####
## using training testing to find the best lamda
## split it into a training set and a test set
n <- dim(x)[1]
indices <- sort(sample(1:n, round (0.7 * n)))
training.x <- x[indices,]
training.y <- y[indices]
test.x <- x[-indices,]
test.y <- y[-indices]

training.df <- data.frame(X = training.x, Y = training.y)
test.df <- data.frame(X = test.x, Y = test.y)

rmse <- function(y, h) {
  return(sqrt(mean((y - h) ^ 2)))
}

## loop over values of Lambda
glmnet.fit <- glmnet(training.x, training.y)
lambdas <- glmnet.fit$lambda ## regularization parameter (sequence)
performance <- data.frame()
for (lambda in lambdas) {
  performance <- rbind(performance ,
                       data.frame(Lambda = lambda ,
                                  RMSE = rmse(test.y, predict(glmnet.fit , test.x, s = lambda))))
}

## plot to see the model performance w.r.t. the range of lambdas
ggplot(performance , aes(x = Lambda , y = RMSE)) +
  geom_point() +
  geom_line()
```



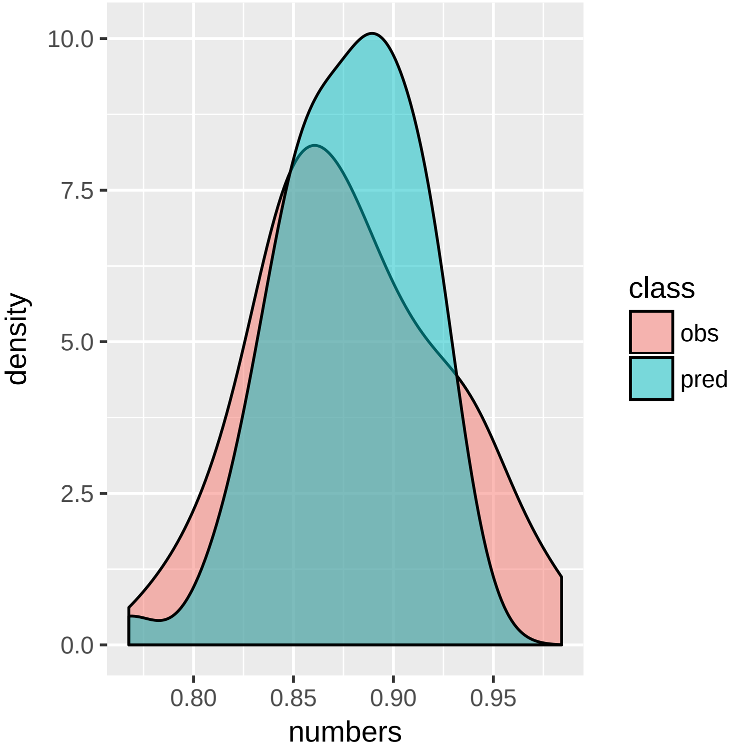
```
## we get the best possible performance with Lambda near 0.01; select that value and train our model
best.lambda <- with(performance , Lambda[which(RMSE == min(RMSE))])

##---Step2----##
##Using the best lambda to fit the model and calculate the leave one out cross validation RMSE
## leave-one-out cross validation
n = length(data$final_comp_grade)
error = dim(n)
predicts = dim(n)
obss = dim(n)
for (k in 1:n) {
  train1 = c(1:n)
  train = train1[train1!=k] ## pick elements that are different from k
  training.x2 = x[train ,]
  training.y2 = y[train]
  m1 = glmnet(training.x2, training.y2)
  pred = predict(m1, t(as.matrix(x[-train ,])), s = best.lambda[1], alpha = 1) #predict on the one left
  predicts[k] = pred
  obs = y[-train]
  obss[k] = obs
  error[k] = obs-pred
}
rmse_reg=sqrt(mean(error^2))
rmse_reg ## root mean square error
```

```
## [1] 0.04211293
```

```
#0.04211337

glmnet.fit <- with(df, glmnet(x, y))
coef(glmnet.fit , s = best.lambda[1])
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                  1
## (Intercept)             0.92689019
## x_factor               -0.02799074
## ave_reading_speed1     -0.13893252
## ave_reading_speed2      0.11683335
## std_reading_speed1      0.15647063
## std_reading_speed2     -0.17162958
## Ave_first_attempt_rate1 -0.01692977
## Ave_first_attempt_rate2 -0.10301395
## Std_first_attempt_rate1 -0.06754702
## Std_first_attempt_rate2  0.01359528
## Ave_last_attempt_rate1   0.18198461
## Ave_last_attempt_rate2   0.17293422
## Std_last_attempt_rate1   0.28837847
## Std_last_attempt_rate2  -0.03830748
```

```
prediction = data.frame(predicts)
colnames(prediction) = "numbers"
prediction$class = "pred"
observation = data.frame(obss)
colnames(observation) = "numbers"
observation$class="obs"

predictions_comp = rbind(prediction, observation)
ggplot(predictions_comp, aes(x = numbers, fill = class)) +
  geom_density(alpha = 0.5)
```



- In conclusion, we fit the baseline model by just using the linear regression with all the important variables, and also tested with non-linear regression method: using quadratic term with regularization, which decrease root mean square error based on the leave-one-out cross validation.