Predictive Models in Education

INFO 5200 Learning Analytics: Week 4 Homework

[[Cole Walsh, 4399966]]

The Dataset

For this homework, you will be analyzing the Assisstments dataset from last homework. You should be familiar with the properties of the data at this point. If I gave you new dataset, you would most likely be going through some of the same steps as in the previous homework to get familiar with the dataset. Below I'm copying some of the general info about the dataset from before just in case:

The dataset provides question-level data of students practicing math problems in academic year 2004-2005 using the Assisstments platform. On this platform, students can attempt a problem many times to get it right and they can ask for more and more hints on a problem until the final hint tells them what the answer is. Based on the first few lines of data, and what we know about the dataset, we can infer the following:

- *studentID* is an identifier for students
- *itemid* is an identifier for math questions
- *correctonfirstattempt* is an indicator of whether a student answered correctly on the first attempt
- *attempts* is the number of answer attempts required
- *hints* the number of hints a student requested
- *seconds* time spent on the question in seconds
- the remaining columns provide start and end times and dates for each question

The dataset is in **long format** (1 row = 1 event) instead of wide format (1 row = 1 individual). However, as you can see from the *attempts* variable, you do not have data on each attempt, but a question-level rollup. The data is at the student-question level, which means that there is one row for each question a student attempted that summarizes interaction with the question (performance indicators and time spent).

Start by loading the dataset:

```r
library(tidyverse, quietly = T)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.1

## -- Attaching packages --------------------------------------------------------------- tidyverse 1.2.1 -

## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0

## Warning: package 'ggplot2' was built under R version 3.5.1

## Warning: package 'dplyr' was built under R version 3.5.1

## -- Conflicts ------------------------------------------------------------------ tidyverse_conflicts() -
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
# load info5200.2.assisstments.rds, use the readRDS() function.
asm = readRDS("info5200.2.assisstments.rds")
```

1. Problem Identification

In the real world, we usually start by identifying the problem and then collect data. Here we have a dataset to work with. So what problems might we solve? Here are some ideas:

- predict dropout, (how long) will students stay engaged to intervene before they disengage
- predict correctness on first attempt to start adapting content for at-risk students
- predict time spent, predict number of hints for improving the experience

For the purpose of this homework, we are going to predict dropout. It's a common problem and it is at the student-level, which simplifies methodological considerations.

We can set this up two different ways: - As a regression problem, the outcome can be the number quizzes completed i.e. how far did you get - As a classification problem, the outcome can be returning after a given point – e.g. of those students who have come in and finish 100 questions, how many are going to do at least 300 questions?

For both outcomes, you will need to assume that you are observing these students for a while (say until they finished 100 questions) and then you try to predict the future. You can use the data you observed to make predictions but nothing thereafter.

2. Data Collection

Which of the variables in the dataset will be used. First, what is the outcome? Second, what are the predictors?

Outcomes - For the regression problem we are interested in the number (i.e. numeric) of quizzes. - For the classification problem we are interested in whether (i.e. binary) they go on to complete at least 300, after completing 100 questions.

Predictors - there are no user attributes in this dataset (socio-demographic or other) - however, you have access to information about quiz-taking that can be used to engineer features

3. Feature Engineering

This is where you create the dataset that you will be using in the prediction model. **You need a student-level dataset.** Check out the previous homework to see how to use the group_by and summarise functions from the tidyverse package to achieve this.

Usually feature engineering focuses on just the predictors, but let's also create the outcomes in this section.

(a) Create a dataset (call it *asm_outcomes*) that has for each student the number of quizzes completed and and indicator of whether that below 300 (i.e. dropped out before). You are looking for a dataset with 912 rows (# of unique students) and three columns: studentID, num_quiz, quiz300. You can refer to the last HW for help.

```r
asm_outcomes = asm %>%
  group_by(studentID) %>%
  summarize(num_quiz = n(),
            quiz300 = num_quiz >= 300)

nrow(asm_outcomes)
```

```
## [1] 912
```

```r
head(asm_outcomes)
```

```
## # A tibble: 6 x 3
##   studentID num_quiz quiz300
##       <int>    <int> <lgl>
## 1       136      518 TRUE
## 2       137      687 TRUE
## 3       139      538 TRUE
## 4       140      522 TRUE
## 5       141      113 FALSE
## 6       142        5 FALSE
```

(b) Now let's engineer some features to predict dropout. I will leave this up to your creativity. You can create as many features as you an think of. You can also evaluate them by looking at their correlation with the outcome if you like. Here is just one example to get you started. I'll create a feature that is the total time spent so far working on questions.

However, there is one critical step not to forget. The features can only be computed using data up to the 100th quiz, given the prediction problem. You will need to throw out the rest. First, keep only the first 100 question records for each student. In this dataset, it takes some (cumbersome) data processing because of how the dates are formatted. Here is one way to do it.

We make a timestamp that can be rank ordered. Then we create a variable i that counts the question order for each student. Now that we know the order in which questions were answered, we can filter out all but the first 100.

```r
# We first need to go through this tedious process of
#  dealing with the dates to make them sortable

# convert to character string
asm$start_day = as.character(asm$start_day)
# split up e.g. 03-OCT-05
start_day_split = strsplit(asm$start_day, split = "-", fixed = T)
# get the day
asm$start_d = unlist(lapply(start_day_split, first))
# get the year, add 20 in front
asm$start_y = paste0(20, unlist(lapply(start_day_split, last)))
# get/convert month
asm$start_m = match(unlist(lapply(start_day_split, function(x) x[2])), toupper(month.abb))
# convert time to character string
asm$start_time = as.character(asm$start_time)
# concat it all
asm$start_timestamp = paste0(asm$start_y, asm$start_m, asm$start_d, asm$start_time)
```

```
# Compute the order in which students answered questions, keep first 100
asm_sub = asm %>%
    group_by(studentID) %>%
    mutate(i = rank(start_timestamp, ties.method = "random")) %>%
    filter(i <= 100)
```

Now that you have a dataset with only the information in it that you can use for prediction, you can start engineering features. Below, you should engineer 10-15 features. Be creative, think about what behaviors could signal that a student will/won't drop out.

```
Q_Difficulty <- asm_sub %>%
  group_by(itemid) %>%
  summarize(Q_diff = mean(correctonfirstattempt))

asm_sub <- left_join(asm_sub, Q_Difficulty, by = 'itemid')

asm_sub$start_day <- as.Date(asm_sub$start_day, format="%d-%b-%y")
asm_sub$finish_day <- as.Date(asm_sub$finish_day, format="%d-%b-%y")

# Now using the asm_sub dataset we can finally compute features like total time
asm_features_overall = asm_sub %>%
    group_by(studentID) %>%
    summarise(
        total_time = sum(seconds),
        avg_hints = mean(hints),
        avg_attempts = mean(attempts),
        avg_correct = mean(correctonfirstattempt),
        sd_attempts = sd(attempts),
        sd_time = sd(seconds),
        #seconds_per_attempt = mean(seconds/attempts),
        #hints_per_attempt = mean(hints/attempts),
        n_days = as.numeric(difftime(max(finish_day), min(start_day))),
        mean_Q_diff = mean(Q_diff))

asm_features_last10 = asm_sub %>%
  filter(i > 90) %>%
  group_by(studentID) %>%
  summarize(total_time_last10 = sum(seconds),
            avg_hints_last10 = mean(hints),
            avg_attempts_last10 = mean(attempts),
            avg_correct_last10 = mean(correctonfirstattempt)
            #seconds_per_attempt_last10 = mean(seconds/attempts),
            #hints_per_attempt_last10 = mean(hints/attempts))
  )

asm_features <- left_join(asm_features_overall, asm_features_last10, by = "studentID")

# check out your features to make sure you don't have
# missing values and the distributions look reasonable
# if there are missing values (NAs) then you should handle them before moving on
asm_features[is.na(asm_features)] <- 0

summary(asm_features)
```

```
##    studentID        total_time      avg_hints       avg_attempts
##  Min.   : 136.0   Min.   :   11   Min.   :0.0000   Min.   :0.000
##  1st Qu.: 447.8   1st Qu.: 3202   1st Qu.:0.3479   1st Qu.:1.306
##  Median : 745.5   Median : 4426   Median :0.6633   Median :1.490
##  Mean   :1088.0   Mean   : 4463   Mean   :0.7645   Mean   :1.518
##  3rd Qu.:1054.2   3rd Qu.: 5726   3rd Qu.:1.1000   3rd Qu.:1.680
##  Max.   :6802.0   Max.   :11264   Max.   :3.5000   Max.   :5.000
##   avg_correct      sd_attempts        sd_time           n_days
##  Min.   :0.0000   Min.   :0.0000   Min.   :  0.00   Min.   :  0.0
##  1st Qu.:0.2700   1st Qu.:0.9551   1st Qu.: 45.18   1st Qu.: 28.0
##  Median :0.3900   Median :1.1989   Median : 59.89   Median : 56.0
##  Mean   :0.3957   Mean   :1.2929   Mean   : 64.63   Mean   : 70.7
##  3rd Qu.:0.5100   3rd Qu.:1.4976   3rd Qu.: 78.78   3rd Qu.:103.0
##  Max.   :1.0000   Max.   :5.6569   Max.   :399.69   Max.   :266.0
##   mean_Q_diff      total_time_last10 avg_hints_last10 avg_attempts_last10
##  Min.   :0.08852   Min.   :   0.0    Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.35289   1st Qu.: 126.0    1st Qu.:0.0000   1st Qu.:0.900
##  Median :0.38547   Median : 309.5    Median :0.3000   Median :1.200
##  Mean   :0.39933   Mean   : 352.9    Mean   :0.5233   Mean   :1.137
##  3rd Qu.:0.44176   3rd Qu.: 525.2    3rd Qu.:0.9000   3rd Qu.:1.600
##  Max.   :0.85238   Max.   :1588.0    Max.   :3.2000   Max.   :3.800
##  avg_correct_last10
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.3000
##  Mean   :0.3429
##  3rd Qu.:0.6000
##  Max.   :1.0000
```

```r
nrow(asm_features)
```

```
## [1] 912
```

Lastly, you will need to merge the two datasets back together: the one with the outcome data and the one with the features. This dataset should have 912 rows.

```r
asm_combined = left_join(asm_features, asm_outcomes, by = "studentID")
nrow(asm_combined)
```

```
## [1] 912
```

4. Feature Selection

This step is usually needed when you have thousands of features, or more features than data points. One option is to remove features that are not predictive, another is to combine many weaker features into one stronger one. A common method for the latter is Principle Component Analysis (PCA).

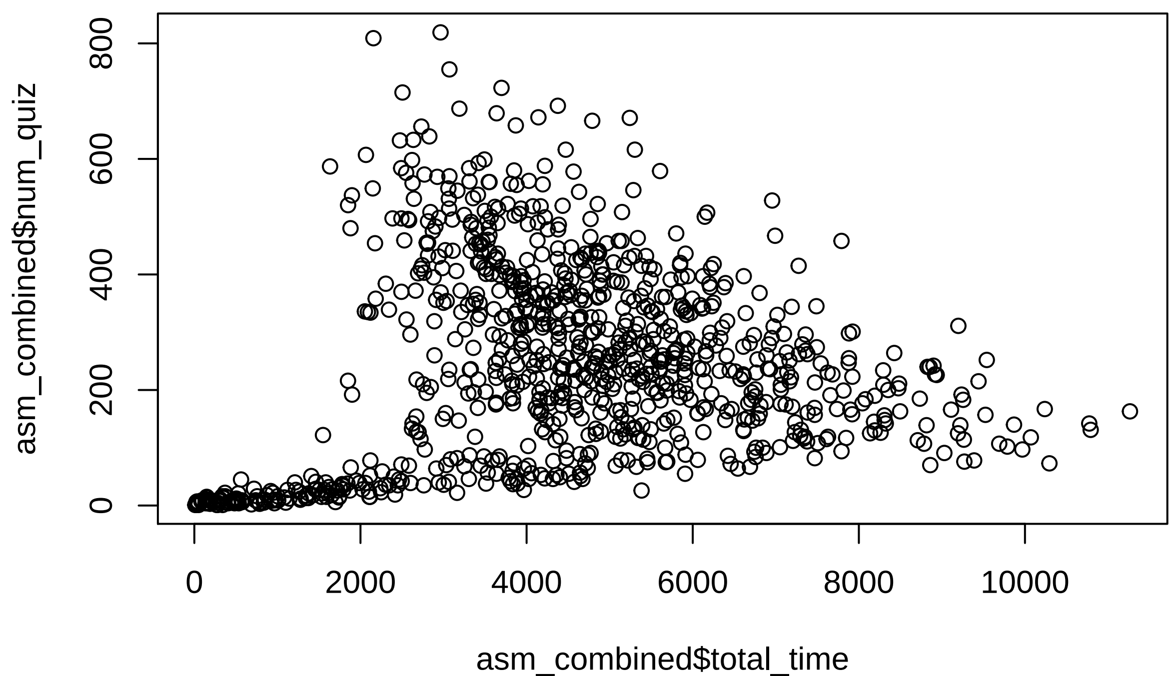
For now, I am assuming you created about 10-15 features in step 3. If you only have 5 or so, go back and come up with more.

Take the opportunity here to evaluate your various features. Check out the correlation, make plots to see if you are perhaps trying to fit a straight line when the relationship is quadratic or cubic. If so, go back and refine your features.

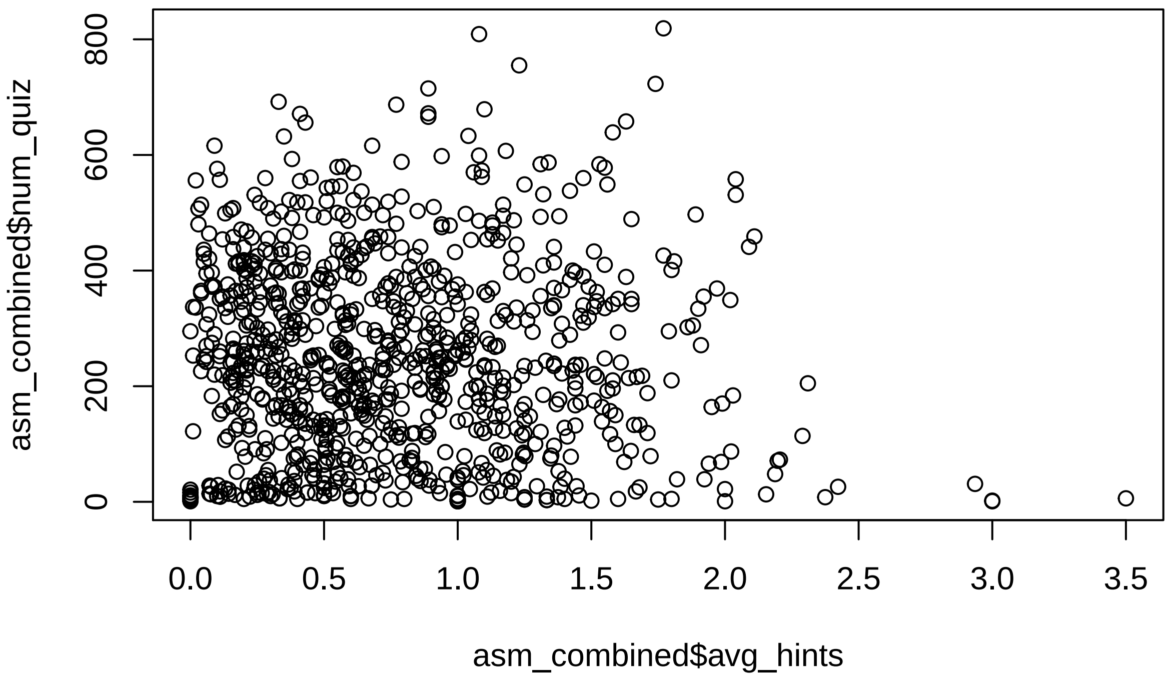
```
outcome_vars = c("num_quiz", "quiz300")
cor(asm_combined)[,outcome_vars]
```

```
##                        num_quiz     quiz300
## studentID           -0.40298119 -0.24773508
## total_time           0.13144653 -0.07465807
## avg_hints           -0.03318345 -0.02968326
## avg_attempts        -0.08553169 -0.11708874
## avg_correct          0.12727397  0.12769097
## sd_attempts         -0.01424828 -0.05653067
## sd_time             -0.28656857 -0.27106277
## n_days              -0.13532117 -0.22694866
## mean_Q_diff          0.12744415  0.10056986
## total_time_last10    0.24983690  0.06299818
## avg_hints_last10     0.25160878  0.13663082
## avg_attempts_last10  0.52317764  0.28446081
## avg_correct_last10   0.49719738  0.33577057
## num_quiz             1.00000000  0.82261259
## quiz300              0.82261259  1.00000000
```

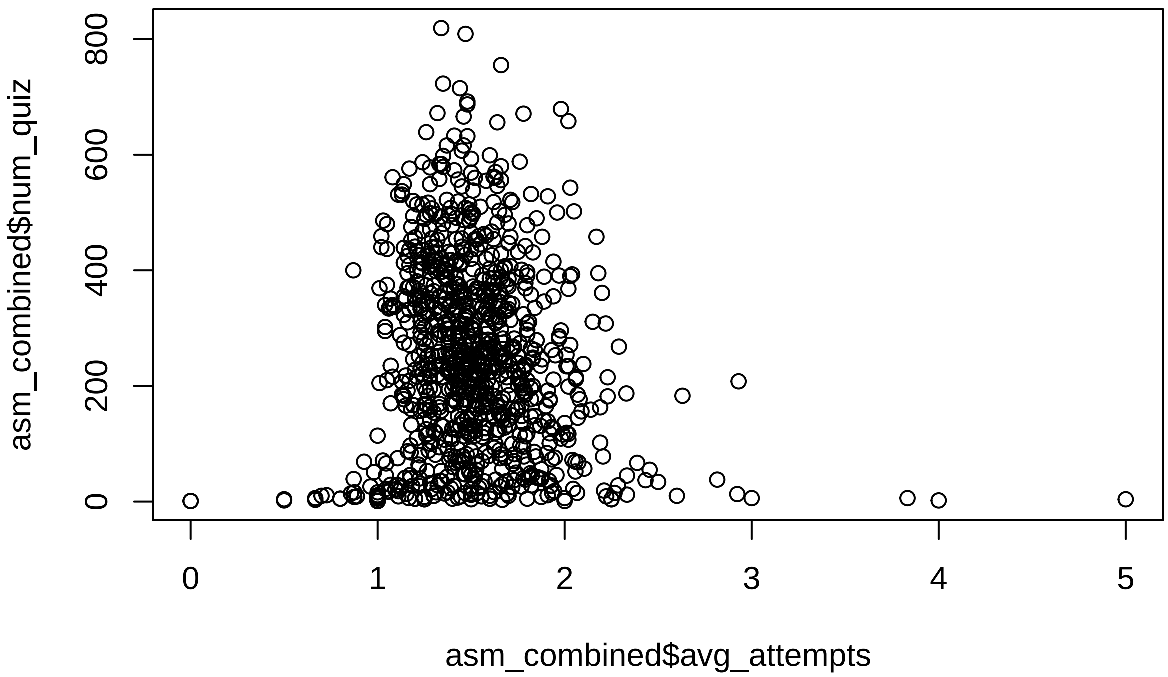
```
plot(asm_combined$total_time, asm_combined$num_quiz)
```



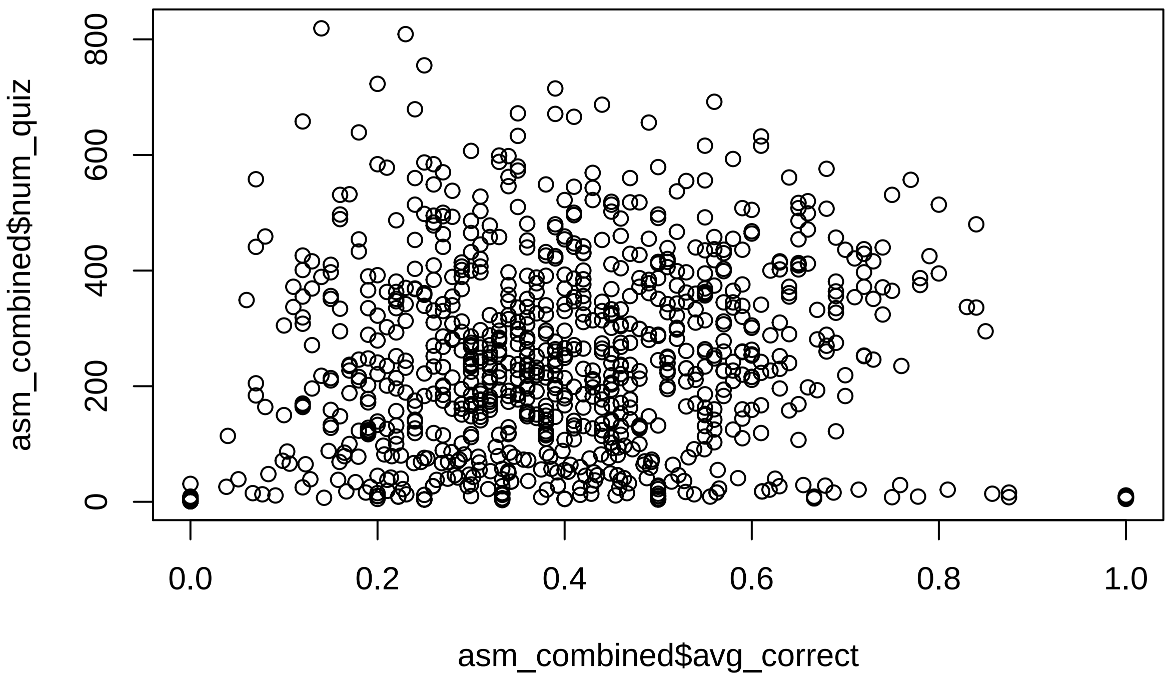

```
plot(asm_combined$avg_hints, asm_combined$num_quiz)
```



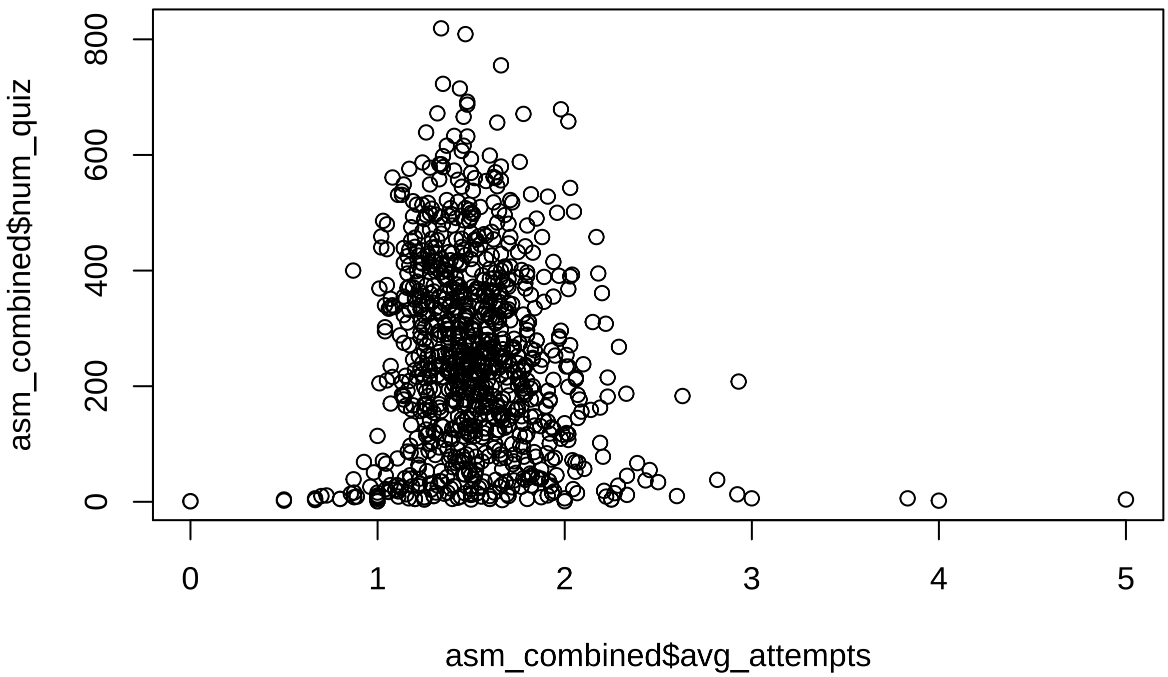

```
plot(asm_combined$avg_attempts, asm_combined$num_quiz)
```



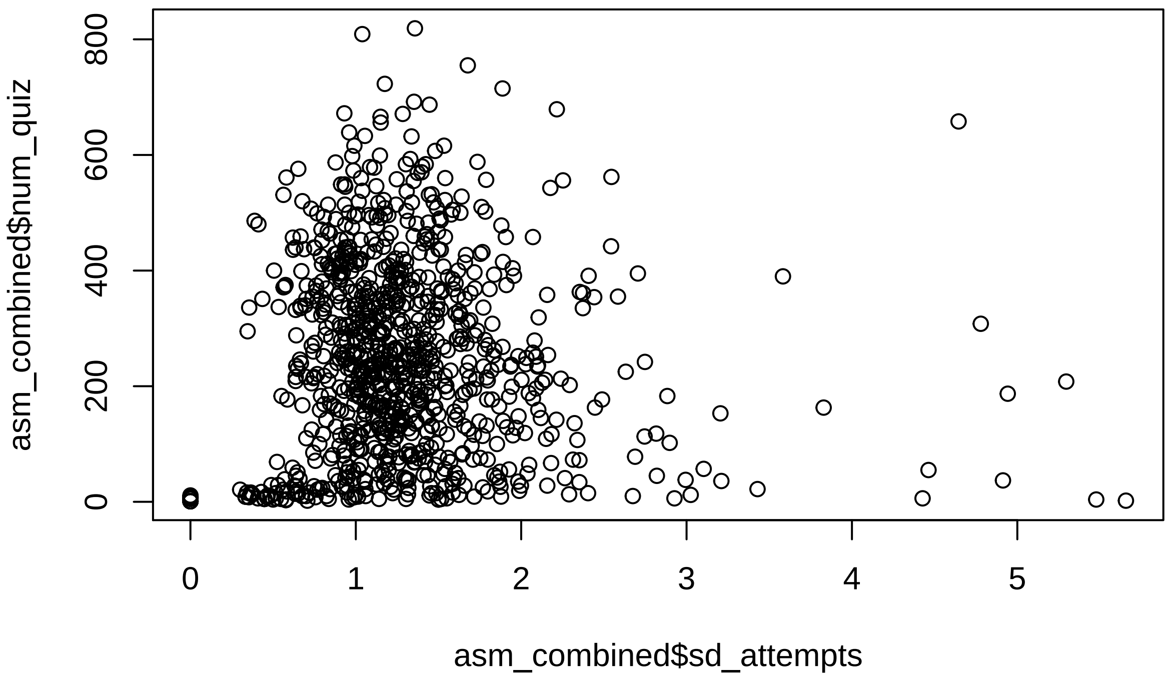

```r
plot(asm_combined$avg_correct, asm_combined$num_quiz)
```



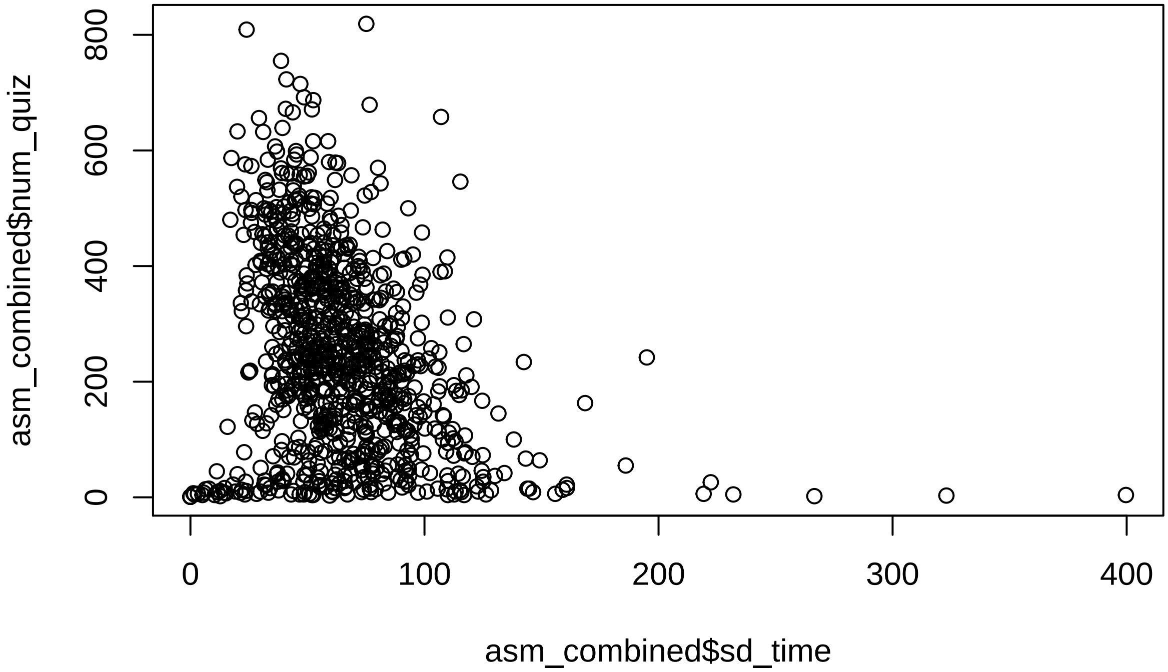

```r
plot(asm_combined$avg_attempts, asm_combined$num_quiz)
```



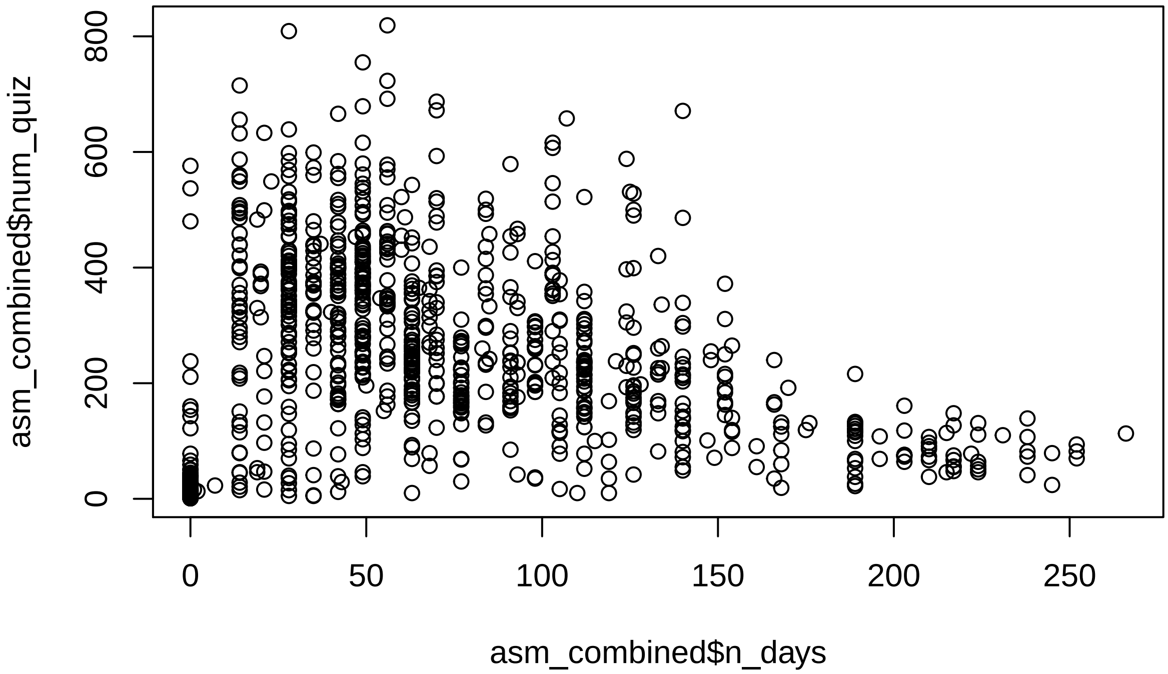

```r
plot(asm_combined$sd_attempts, asm_combined$num_quiz)
```



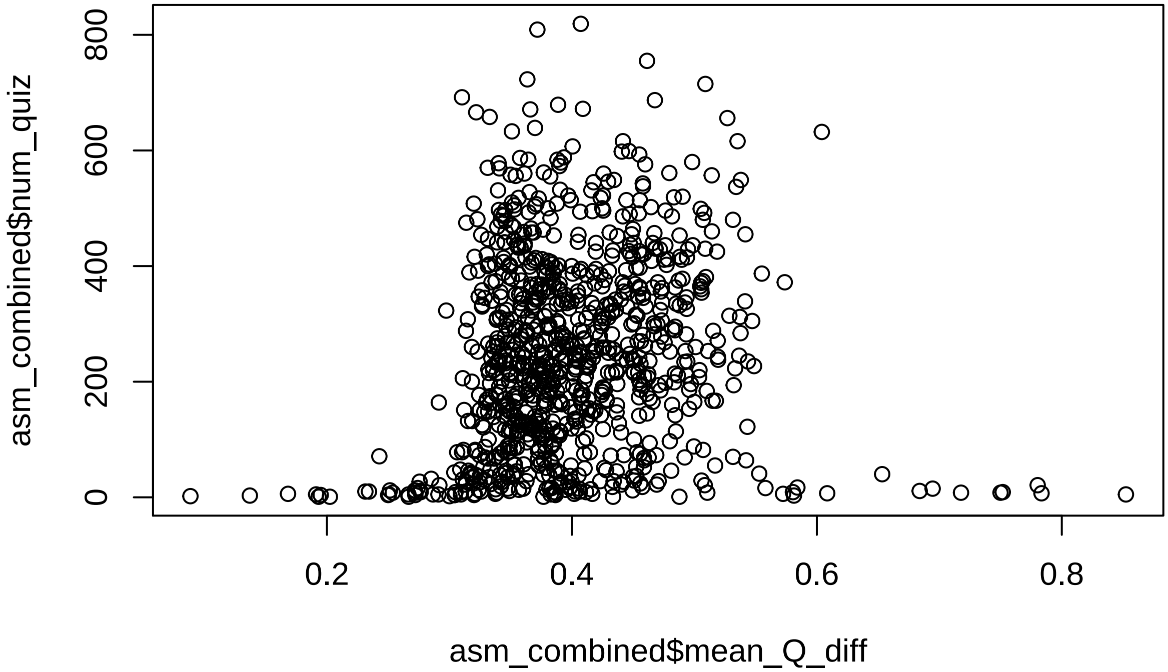

```
plot(asm_combined$sd_time, asm_combined$num_quiz)
```



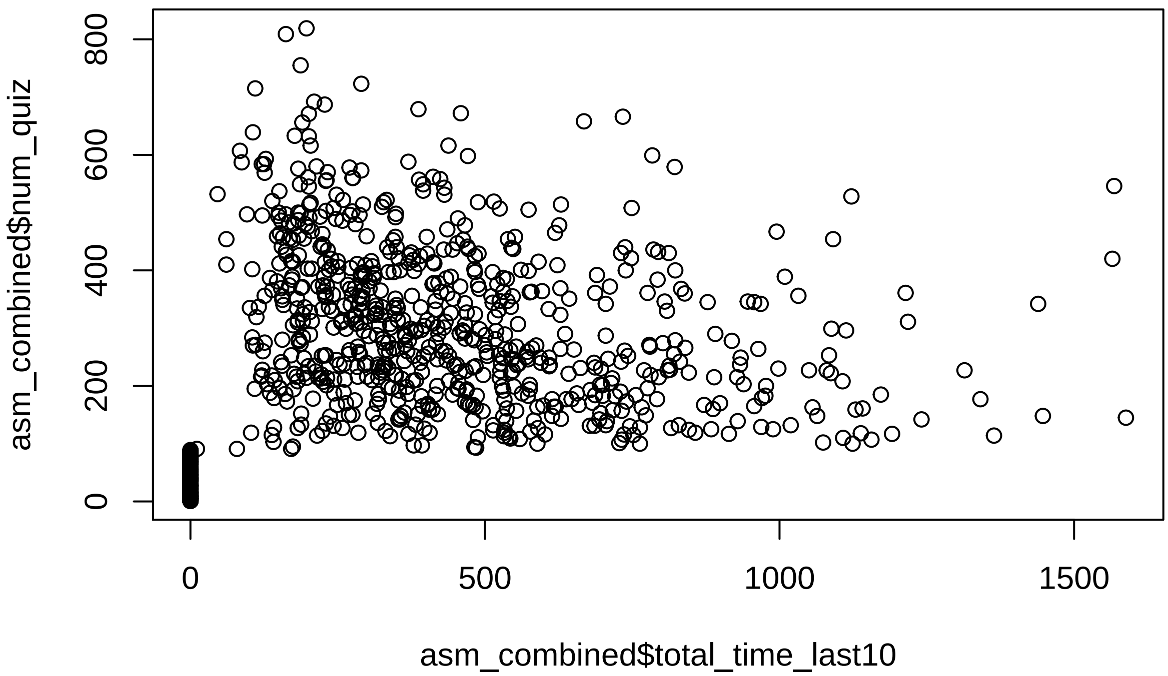

```r
plot(asm_combined$n_days, asm_combined$num_quiz)
```



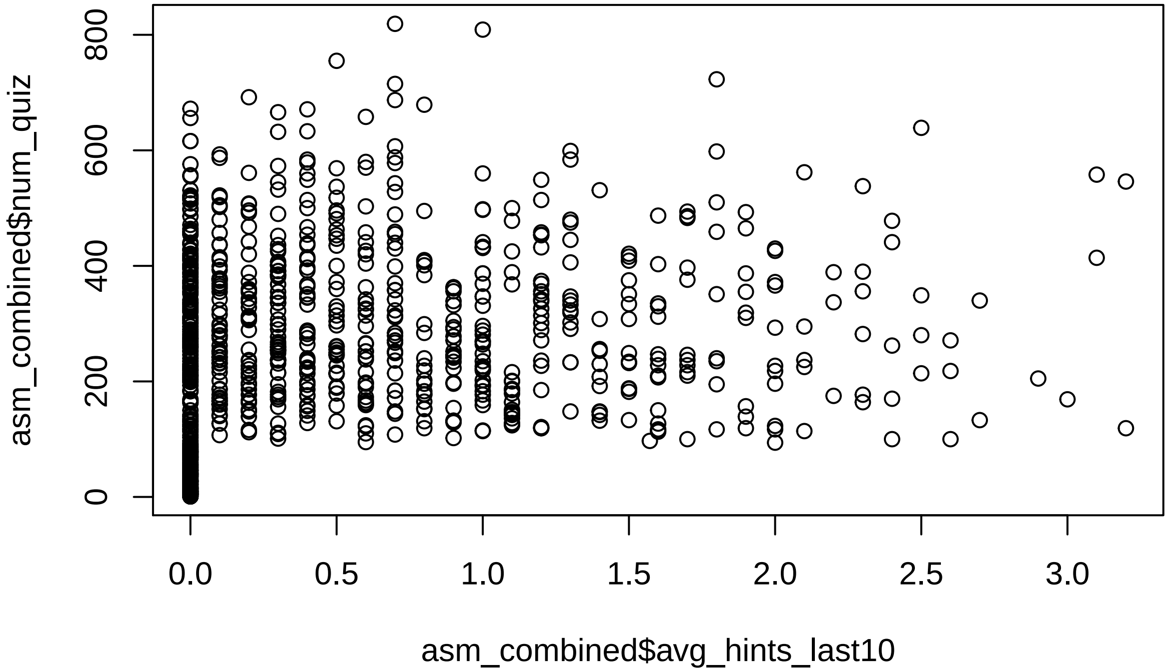

```
plot(asm_combined$mean_Q_diff, asm_combined$num_quiz)
```



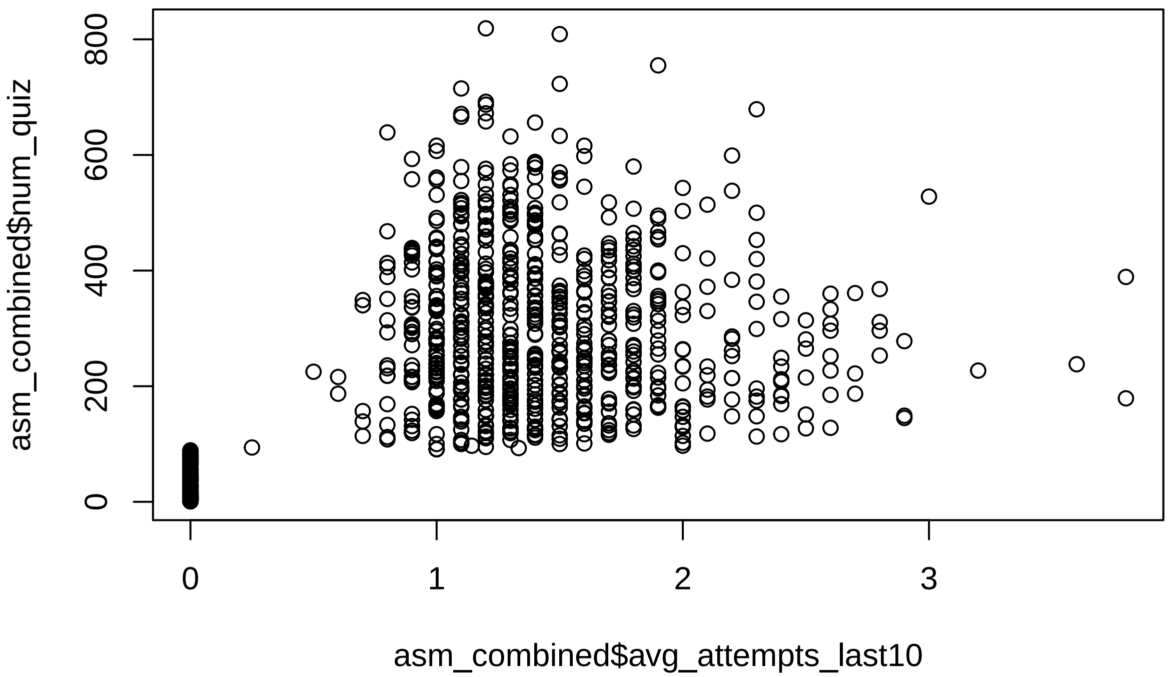

```
plot(asm_combined$total_time_last10, asm_combined$num_quiz)
```



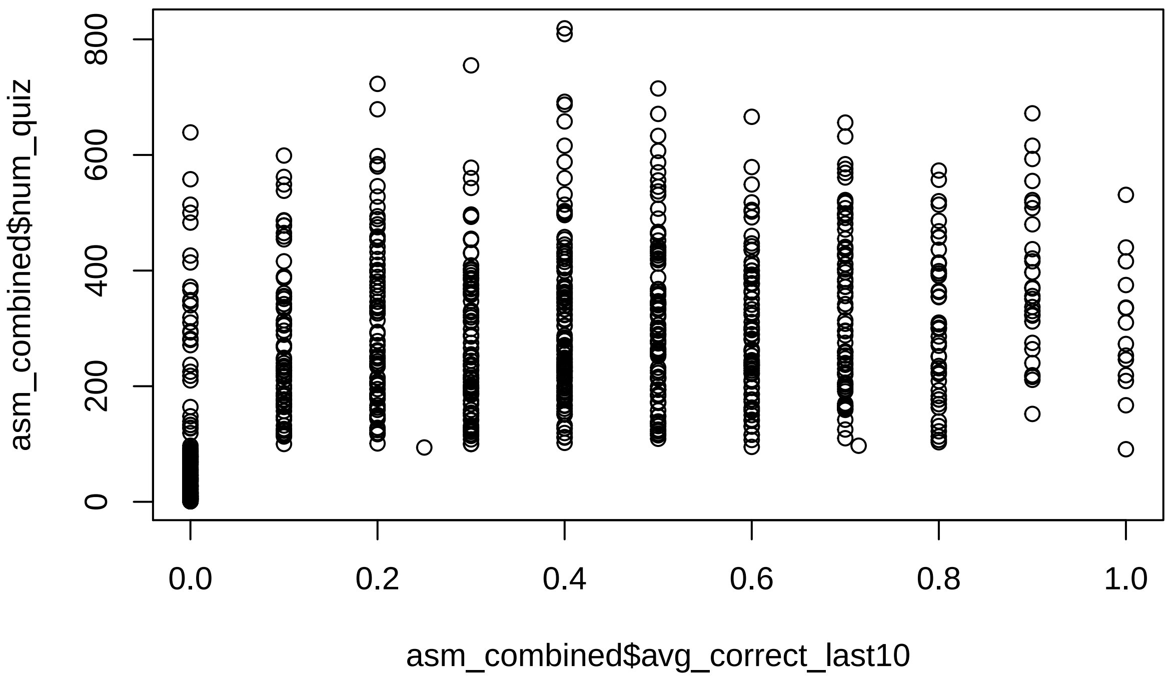

```
plot(asm_combined$avg_hints_last10, asm_combined$num_quiz)
```




```r
plot(asm_combined$avg_attempts_last10, asm_combined$num_quiz)
```




```r
plot(asm_combined$avg_correct_last10, asm_combined$num_quiz)
```



5. Model Selection / Building

Before we can start building models, we need to split our dataset into a training and a test set. (Note that it we should usually do this before feature engineering so that we are not influenced in our choices by data that we shouldn't be seeing. But then we would have to do the engineering twice. So let's just do it here.)

The dataset is now quite small: 912 students. We do want enough data to train our model, so let's do a 80/20 split: 80% training, 20% test. It is important that the split is **random**. Why? Because we want it to be a representative sample.

```
# Sample 80% of studentIDs for training and the rest is for testing,
#    you want a vector of studentIDs
ids_train = sample(asm_combined$studentID, size = 912 * 0.8)

# Split the dataset into two; use filter() and %in% to select rows
train = asm_combined %>% filter(studentID %in% ids_train)
test = asm_combined %>% filter(!studentID %in% ids_train)
```

Need a just-in-time R tutorial?

https://www.datacamp.com/community/tutorials/machine-learning-in-r

Linear regression

To fit a linear regression model, use the lm() function like this: - lm(outcome ~ predictor1 + predictor2 + predictor3, data = train)

```r
m_linreg = lm(num_quiz ~ . - studentID - quiz300, data = train)

# the output are the coefficients:
m_linreg
```

```
##
## Call:
## lm(formula = num_quiz ~ . - studentID - quiz300, data = train)
##
## Coefficients:
##         (Intercept)           total_time            avg_hints
##           66.719418            -0.005108            18.322157
##        avg_attempts          avg_correct          sd_attempts
##          -13.072837            62.063293            31.106579
##             sd_time               n_days          mean_Q_diff
##           -0.643844            -0.216848             5.448689
##    total_time_last10      avg_hints_last10  avg_attempts_last10
##           -0.119152            75.335554            90.841504
##   avg_correct_last10
##          283.160416
```

Logistic regression

To fit a logistic regression model, use the glm() function like this: - glm(outcome ~ predictor1 + predictor2 + predictor3, data = train, family = "binomial")

```r
m_logreg = glm(quiz300 ~ . - studentID - num_quiz, data = train, family = 'binomial')

# the output are the coefficients:
m_logreg
```

```
##
## Call:  glm(formula = quiz300 ~ . - studentID - num_quiz, family = "binomial",
##     data = train)
##
## Coefficients:
##         (Intercept)           total_time            avg_hints
##          -2.1776801           -0.0002668            0.3588389
##        avg_attempts          avg_correct          sd_attempts
##          -0.3834116            2.0037829            0.6328764
##             sd_time               n_days          mean_Q_diff
##          -0.0143018           -0.0099489           -1.3140988
##    total_time_last10      avg_hints_last10  avg_attempts_last10
##          -0.0011460            1.1043047            1.3464806
##   avg_correct_last10
##           4.0997204
##
```

```
## Degrees of Freedom: 728 Total (i.e. Null);  716 Residual
## Null Deviance:	    949.8
## Residual Deviance: 676.7 	AIC: 702.7
```

k Nearest Neighbor

To fit a kNN model, use the knn() function from the {class} package. However, note that the syntax starts to get different here, and you would usually do some tuning, e.g. choosing the right value of *k*. For this case, just choose a number between 1 and 5. The function takes the predictor matrix for training and testing, and a vector of outcomes (binary) for training. - knn(train = training_predictors, test = testing_predictors, cl = training_outcome, k = k)

Important: Do not forget to remove the studentID! It does not generalize well.

```r
# install.packages("class") # you may need to install this first
library(class)
m_knn = knn(train = train[, -c(1, 14, 15)], test = test[, -c(1, 14, 15)],
            cl = train$quiz300, k = 4)

# the output are the predictions:
m_knn
```

```
##   [1] FALSE FALSE TRUE  TRUE  TRUE  FALSE FALSE TRUE  TRUE  TRUE  FALSE
##  [12] TRUE  FALSE FALSE FALSE TRUE  FALSE TRUE  TRUE  FALSE FALSE FALSE
##  [23] FALSE TRUE  TRUE  FALSE TRUE  FALSE FALSE TRUE  FALSE FALSE TRUE
##  [34] FALSE FALSE FALSE TRUE  TRUE  FALSE FALSE FALSE FALSE TRUE  FALSE
##  [45] FALSE TRUE  TRUE  FALSE FALSE FALSE FALSE TRUE  FALSE TRUE  TRUE
##  [56] TRUE  FALSE FALSE FALSE TRUE  FALSE FALSE TRUE  TRUE  FALSE FALSE
##  [67] FALSE TRUE  FALSE FALSE FALSE FALSE TRUE  FALSE FALSE TRUE  FALSE
##  [78] TRUE  FALSE FALSE TRUE  TRUE  FALSE FALSE TRUE  FALSE TRUE  TRUE
##  [89] FALSE FALSE TRUE  FALSE FALSE TRUE  FALSE FALSE TRUE  TRUE  FALSE
## [100] FALSE FALSE TRUE  TRUE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE TRUE  FALSE FALSE TRUE  FALSE FALSE TRUE  TRUE
## [122] FALSE FALSE TRUE  FALSE FALSE FALSE TRUE  TRUE  TRUE  FALSE FALSE
## [133] FALSE TRUE  FALSE FALSE FALSE TRUE  FALSE FALSE TRUE  FALSE FALSE
## [144] TRUE  TRUE  TRUE  FALSE FALSE FALSE TRUE  FALSE FALSE TRUE  TRUE
## [155] TRUE  FALSE TRUE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [166] FALSE FALSE FALSE FALSE FALSE TRUE  FALSE FALSE FALSE FALSE FALSE
## [177] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## Levels: FALSE TRUE
```

Classification and Regression Trees

To fit a CART model, use the rpart() function from the {rpart} package. The syntax is pretty similar to the linear/logistic regression models. To build a classification tree you specify method as 'class', for a regression tree you specify it as 'anova'. - rpart(binary_outcome ~ predictor1 + predictor2 + predictor3, data = train, method = "class") - rpart(numeric_outcome ~ predictor1 + predictor2 + predictor3, data = train, method = "anova")

Here's an R tutorial for CART.

```r
# install.packages("rpart") # you may need to install this first
library(rpart)
m_class_tree = rpart(quiz300 ~ . - studentID - num_quiz, data = train, method = 'class')
m_reg_tree = rpart(num_quiz ~ . - studentID - quiz300, data = train, method = 'anova')

# the output are the decision trees
m_class_tree
```

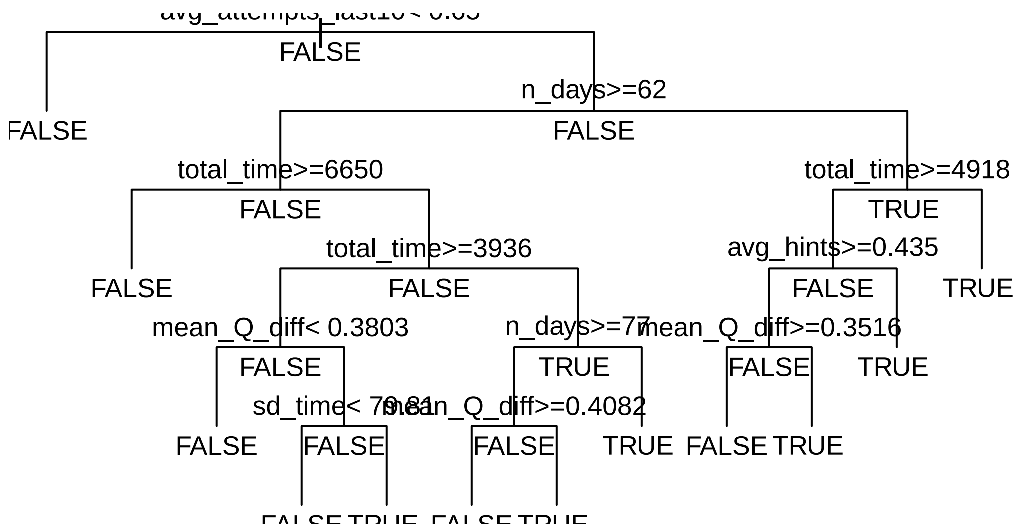
```
## n= 729
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 729 260 FALSE (0.64334705 0.35665295)
##     2) avg_attempts_last10< 0.65 163   0 FALSE (1.00000000 0.00000000) *
##     3) avg_attempts_last10>=0.65 566 260 FALSE (0.54063604 0.45936396)
##       6) n_days>=62 301  83 FALSE (0.72425249 0.27574751)
##        12) total_time>=6650 94   5 FALSE (0.94680851 0.05319149) *
##        13) total_time< 6650 207  78 FALSE (0.62318841 0.37681159)
##          26) total_time>=3935.5 173  57 FALSE (0.67052023 0.32947977)
##            52) mean_Q_diff< 0.3802796 59  10 FALSE (0.83050847 0.16949153) *
##            53) mean_Q_diff>=0.3802796 114  47 FALSE (0.58771930 0.41228070)
##             106) sd_time< 79.80671 92  32 FALSE (0.65217391 0.34782609) *
##             107) sd_time>=79.80671 22   7 TRUE (0.31818182 0.68181818) *
##          27) total_time< 3935.5 34  13 TRUE (0.38235294 0.61764706)
##            54) n_days>=77 23  11 FALSE (0.52173913 0.47826087)
##             108) mean_Q_diff>=0.4081863 9   1 FALSE (0.88888889 0.11111111) *
##             109) mean_Q_diff< 0.4081863 14   4 TRUE (0.28571429 0.71428571) *
##            55) n_days< 77 11   1 TRUE (0.09090909 0.90909091) *
##       7) n_days< 62 265  88 TRUE (0.33207547 0.66792453)
##        14) total_time>=4917.5 59  26 FALSE (0.55932203 0.44067797)
##          28) avg_hints>=0.435 36  10 FALSE (0.72222222 0.27777778)
##            56) mean_Q_diff>=0.3516395 27   4 FALSE (0.85185185 0.14814815) *
##            57) mean_Q_diff< 0.3516395 9   3 TRUE (0.33333333 0.66666667) *
##          29) avg_hints< 0.435 23   7 TRUE (0.30434783 0.69565217) *
##        15) total_time< 4917.5 206  55 TRUE (0.26699029 0.73300971) *
```

```r
m_reg_tree
```

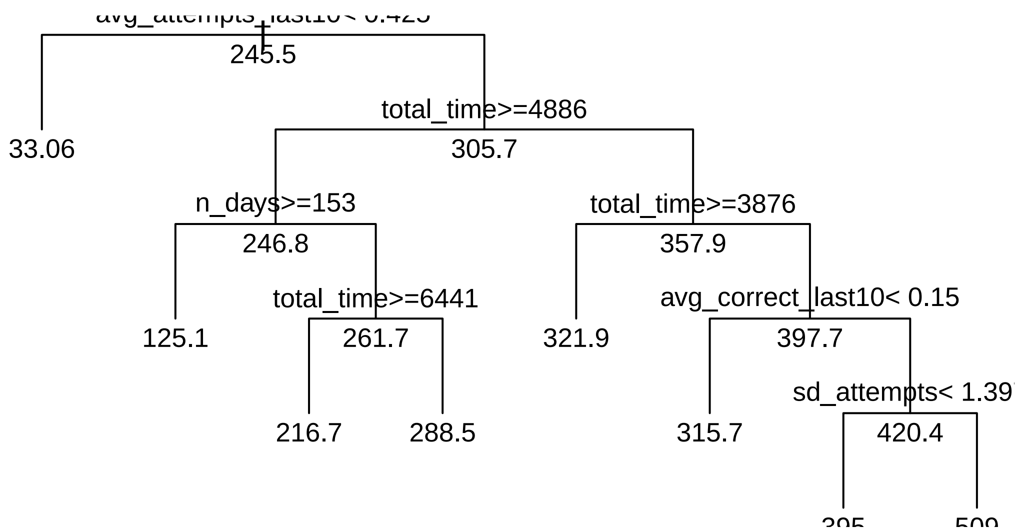
```
## n= 729
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 729 19974310.00 245.4815
##    2) avg_attempts_last10< 0.425 161   115714.50  33.0559 *
##    3) avg_attempts_last10>=0.425 568 10534250.00 305.6937
##      6) total_time>=4885.5 267  2807243.00 246.8240
##       12) n_days>=153 29    22023.86 125.0690 *
##       13) n_days< 153 238  2302931.00 261.6597
##         26) total_time>=6441 89   417251.20 216.6854 *
##         27) total_time< 6441 149  1598133.00 288.5235 *
##      7) total_time< 4885.5 301  5980874.00 357.9136
```

```
##       14) total_time>=3876.5 158  2442999.00 321.8797 *
##       15) total_time< 3876.5 143  3106048.00 397.7273
##         30) avg_correct_last10< 0.15 31   620223.90 315.7419 *
##         31) avg_correct_last10>=0.15 112  2219781.00 420.4196
##           62) sd_attempts< 1.396514 87  1443418.00 394.9540 *
##           63) sd_attempts>=1.396514 25   523605.00 509.0400 *
```

```r
# you can even plot it!
plot(m_class_tree, uniform = T)
text(m_class_tree, use.n = F, all = TRUE, cex = .8)
```




```r
plot(m_reg_tree, uniform = T)
text(m_reg_tree, use.n = F, all = TRUE, cex = .8)
```




```r
# prune the trees to avoid overfitting by limiting tree complexity
cp_class_tree = m_class_tree$cptable[which.min(m_class_tree$cptable[,"xerror"]),"CP"]
m_class_tree_pruned = prune(m_class_tree, cp = cp_class_tree)

cp_reg_tree = m_reg_tree$cptable[which.min(m_reg_tree$cptable[,"xerror"]),"CP"]
m_reg_tree_pruned = prune(m_reg_tree, cp = cp_reg_tree)
```

Naive Bayes Classifier

To fit an NB model, use the naiveBayes() function from the {e1071} package. The syntax is pretty similar to the linear/logistic regression models again. - naiveBayes(binary_outcome ~ predictor1 + predictor2 + predictor3, data = train)

Here's an R tutorial for naive bayes.

```r
# install.packages("e1071") # you may need to install this first
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.5.2
```

```r
m_nb = naiveBayes(quiz300 ~ total_time + avg_hints + avg_attempts + avg_correct + sd_attempts +
                    sd_time + n_days + mean_Q_diff + total_time_last10 + avg_hints_last10 +
                    avg_attempts_last10 + avg_correct_last10, data = train)

head(train)
```

```
## # A tibble: 6 x 15
##   studentID total_time avg_hints avg_attempts avg_correct sd_attempts
##       <int>      <int>     <dbl>        <dbl>       <dbl>       <dbl>
## 1       136       4169      0.4          1.62        0.48       1.34
## 2       137       3191      0.77         1.48        0.44       1.45
## 3       139       3414      1.42         1.51        0.28       1.04
## 4       140       4856      0.37         1.71        0.4        1.54
## 5       141       8709      0.14         1.79        0.55       2.75
## 6       142        440      0.6          1.6         0          0.548
## # ... with 9 more variables: sd_time <dbl>, n_days <dbl>,
## #   mean_Q_diff <dbl>, total_time_last10 <dbl>, avg_hints_last10 <dbl>,
## #   avg_attempts_last10 <dbl>, avg_correct_last10 <dbl>, num_quiz <int>,
## #   quiz300 <lgl>
```

```
# the output are a-prior and conditional probabilities
m_nb
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     FALSE      TRUE
## 0.6433471 0.3566529
##
## Conditional probabilities:
##        total_time
## Y            [,1]     [,2]
##   FALSE 4566.211 2511.986
##   TRUE  4247.165 1195.520
##
##        avg_hints
## Y             [,1]      [,2]
##   FALSE 0.7862105 0.5438158
##   TRUE  0.7618077 0.5253506
##
##        avg_attempts
## Y            [,1]      [,2]
##   FALSE 1.545166 0.4175394
##   TRUE  1.479615 0.2429640
##
##        avg_correct
## Y             [,1]      [,2]
##   FALSE 0.3762234 0.1780612
##   TRUE  0.4184231 0.1764783
##
##        sd_attempts
## Y            [,1]      [,2]
##   FALSE 1.316398 0.6919559
##   TRUE  1.280622 0.5321420
##
```

```
##        sd_time
## Y           [,1]     [,2]
##   FALSE 71.88597 38.54036
##   TRUE  53.34135 19.34237
##
##        n_days
## Y           [,1]     [,2]
##   FALSE 78.95309 65.94625
##   TRUE  54.62308 31.71355
##
##        mean_Q_diff
## Y            [,1]       [,2]
##   FALSE 0.3938269 0.08028323
##   TRUE  0.4089552 0.05867112
##
##        total_time_last10
## Y           [,1]     [,2]
##   FALSE 345.9339 340.4371
##   TRUE  370.9731 229.5673
##
##        avg_hints_last10
## Y            [,1]      [,2]
##   FALSE 0.4539446 0.6494988
##   TRUE  0.6550000 0.6967755
##
##        avg_attempts_last10
## Y            [,1]      [,2]
##   FALSE 0.9825008 0.8170189
##   TRUE  1.4126923 0.3968414
##
##        avg_correct_last10
## Y            [,1]      [,2]
##   FALSE 0.2762565 0.2807184
##   TRUE  0.4611538 0.2561796
```

6. Evaluation

You just trained a number of models and now you want to know which model performs the best on the test set (holdout data). For simplicity, let us just focus on the classification models here.

Get the predictions for each model using the predict() function where the type is 'response' for the logistic model and 'class' for the other models: - predict(model, newdata = test, type = …)

```
# logreg: this returns the probability of dropout, so you can set Prob > 0.5 to mean Dropout
p_logreg = predict(m_logreg, newdata = test) > 0.5
# knn: this already has the prediction
p_knn = m_knn
# class tree
p_class_tree = predict(m_class_tree_pruned, newdata = test)[, c('TRUE')] > 0.5
# naive bayes
p_nb = predict(m_nb, newdata = test)
```

Now you can create a contingency matrix for each model and compute the accuracy, recall, and precision: -

Accuracy: (TruePos + TrueNeg) / total - Recall: TruePos / (TruePos + FalseNeg) - Precision: TruePos / (TruePos + FalsePos)

```
# here is the confusion matrix for the logreg model:
cm_logreg = table(true = test$quiz300, predicted = p_logreg)

# Get the other ones and then compute the three metrics for each model
cm_knn = table(true = test$quiz300, predicted = p_knn)
cm_class_tree = table(true = test$quiz300, predicted = p_class_tree)
cm_nb = table(true = test$quiz300, predicted = p_nb)

'Logistic Regression'
```

```
## [1] "Logistic Regression"
```

```
cm_logreg
```

```
##        predicted
## true    FALSE TRUE
##   FALSE   100    9
##   TRUE     37   37
```

```
'Accuracy'
```

```
## [1] "Accuracy"
```

```
(cm_logreg[1, 1] + cm_logreg[2, 2])/(cm_logreg[1, 1] + cm_logreg[1, 2] + cm_logreg[2, 1] +
                                       cm_logreg[2, 2])
```

```
## [1] 0.7486339
```

```
'Recall'
```

```
## [1] "Recall"
```

```
cm_logreg[2, 2]/(cm_logreg[2, 2] + cm_logreg[2, 1])
```

```
## [1] 0.5
```

```
'Precision'
```

```
## [1] "Precision"
```

```
cm_logreg[2, 2]/(cm_logreg[2, 2] + cm_logreg[1, 2])
```

```
## [1] 0.8043478
```

```r
'k-nearest neighbours (k = 4)'
```

```
## [1] "k-nearest neighbours (k = 4)"
```

```r
cm_knn
```

```
##        predicted
## true     FALSE TRUE
##   FALSE     86   23
##   TRUE      35   39
```

```r
'Accuracy'
```

```
## [1] "Accuracy"
```

```r
(cm_knn[1, 1] + cm_knn[2, 2])/(cm_knn[1, 1] + cm_knn[1, 2] + cm_knn[2, 1] + cm_knn[2, 2])
```

```
## [1] 0.6830601
```

```r
'Recall'
```

```
## [1] "Recall"
```

```r
cm_knn[2, 2]/(cm_knn[2, 2] + cm_knn[2, 1])
```

```
## [1] 0.527027
```

```r
'Precision'
```

```
## [1] "Precision"
```

```r
cm_knn[2, 2]/(cm_knn[2, 2] + cm_knn[1, 2])
```

```
## [1] 0.6290323
```

```r
'Classification Tree'
```

```
## [1] "Classification Tree"
```

```r
cm_class_tree
```

```
##        predicted
## true     FALSE TRUE
##   FALSE     98   11
##   TRUE      26   48
```

```r
'Accuracy'
```

```
## [1] "Accuracy"
```

```r
(cm_class_tree[1, 1] + cm_class_tree[2, 2])/(cm_class_tree[1, 1] + cm_class_tree[1, 2] +
                                                cm_class_tree[2, 1] + cm_class_tree[2, 2])
```

```
## [1] 0.7978142
```

```r
'Recall'
```

```
## [1] "Recall"
```

```r
cm_class_tree[2, 2]/(cm_class_tree[2, 2] + cm_class_tree[2, 1])
```

```
## [1] 0.6486486
```

```r
'Precision'
```

```
## [1] "Precision"
```

```r
cm_class_tree[2, 2]/(cm_class_tree[2, 2] + cm_class_tree[1, 2])
```

```
## [1] 0.8135593
```

```r
'Naive Bayes'
```

```
## [1] "Naive Bayes"
```

```r
cm_nb
```

```
##        predicted
## true     FALSE TRUE
##   FALSE    73   36
##   TRUE      7   67
```

```r
'Accuracy'
```

```
## [1] "Accuracy"
```

```r
(cm_nb[1, 1] + cm_nb[2, 2])/(cm_nb[1, 1] + cm_nb[1, 2] + cm_nb[2, 1] + cm_nb[2, 2])
```

```
## [1] 0.7650273
```

```
'Recall'
```

```
## [1] "Recall"
```

```
cm_nb[2, 2]/(cm_nb[2, 2] + cm_nb[2, 1])
```

```
## [1] 0.9054054
```

```
'Precision'
```

```
## [1] "Precision"
```

```
cm_nb[2, 2]/(cm_nb[2, 2] + cm_nb[1, 2])
```

```
## [1] 0.6504854
```

Briefly summarize your findings

Which model has the highest/lowest accuracy, recall, precision?

My classification tree preformed best in terms of accuracy, my Naive Bayes classifier performed the best in terms recall, and my logistic regression classifier performed the best in terms of precision. Overall, the classification tree gave the best balance of precision and recall. The logistic regression classifier was too conservative, however, and performed the worst in recall. My knn classifier had the lowest accuracy and precision, however, which may have been related to my choice of k = 4.

Overall, I'm unsure of the generalizability of any of these models given the presence of students who did not complete 100 quizzes in the dataset. I engineered features that examined student behaviour on their 90th-100th quizzes because I thought a change in overall behaviour might be indicative of dropout. However, for students that did not complete 90 quizzes, these features were set to 0. Students that did not complete 100 quizzes obviously did not complete 300 quizzes, so including these students in my training and test datasets artificially enhanced the performance of my models in predicting whether students who completed 100 quizzes would complete 300 quizzes.

Submit Homework

This is the end of the homework. Please **Knit a PDF report** that shows both the R code and R output and upload it on the EdX platform. Alternatively, you can Knit it as a "doc", open it in Word, and save that as a PDF.

Important: Be sure that all your code is visible. If the line is too long, it gets cut off. If that happens, organize your code on several lines.