Project: LMS Data Analytics

INFO 5200 Learning Analytics

[[Kimberly Williamson, khw44]]

Introduction

Goals: The goal of this project is to learn how to work with Learning Management System (LMS) data and apply some of the prediction skills you have learned so far. I am sharing with you an export of the class's edX data from last year between Jan 23rd and Feb 23rd. I have done some data cleaning for you and anonymized the datasets. However, I left plenty of real-world messiness for you to tackle here. As always, you should start by getting to know the datasets. In this case, you should be able to really understand what is going on because it is data from the same course and your know the platform. Moreover, you can navigate to the relevant pages on edX to see what page/action the data refers to.

Project Philosophy: Think about this project like this: in the workplace you can certainly talk to your peers about problems and work through them, but they won't write your code. This is an individual student project. Each student needs to write and submit their own work. You may talk to other students but not copy their code. Be generous in acknowledging where a conversation with another student helped you by adding: `H/T Student Name`.

Step 1: Understand the data

There are three datasets which can be connected using the hash_id column (a hashed version of the user id) and I am giving you links to the official documentation which you should read to understand the data better:

1. Clickstream data (1 row per student per action): click for documentation
2. Module States (1 row per student per accessed content): original name courseware-studentmodule (click for doumentation)
3. Assessment grades (1 row per assessment per student)

I have already converted date-time objects into a numeric `timestamp` for you.

To look up what pages URLs refer to (works for browser events, not server events), you can paste the URL into your browser and then replace the old course id `course-v1:CornellX+INFO5200+2019_Spring` with the new course id `course-v1:Cornellx+Info5200+fall2019`. This should work for most URLs.

Question 1: In the space below, explore each dataset using `head()`, `n_distinct(data$some_id)`, `summary()`, `table(data$column)`. You can also plot the distribution of variables with histograms or boxplots. Check out the data documentation linked above to understand the meaning of each column.

```
################################################
###### BEGIN INPUT: Explore each dataset ######
################################################

# Exploring Clickstreams
# add code here
head(cl)
```

```
##                            hash_id                              time name
## 1 d4b77cfe22357d389b3b56a17e001cdc 2019-01-23T18:12:57.344663+00:00 <NA>
## 2 d4b77cfe22357d389b3b56a17e001cdc 2019-01-23T18:12:52.152147+00:00 <NA>
## 3 d4b77cfe22357d389b3b56a17e001cdc 2019-01-23T18:12:47.110500+00:00 <NA>
## 4 d4b77cfe22357d389b3b56a17e001cdc 2019-01-23T18:10:20.239638+00:00 <NA>
```

```
## 5 d4b77cfe22357d389b3b56a17e001cdc 2019-01-23T18:12:46.015896+00:00 <NA>
## 6 d4b77cfe22357d389b3b56a17e001cdc 2019-01-23T18:12:14.283537+00:00 <NA>
##
## 1 /courses/course-v1:CornellX+INFO5200+2019_Spring/xblock/block-v1:CornellX+INFO5200+2019_Spring+type
## 2 /courses/course-v1:CornellX+INFO5200+2019_Spring/xblock/block-v1:CornellX+INFO5200+2019_Spring+type
## 3 /courses/course-v1:CornellX+INFO5200+2019_Spring/xblock/block-v1:CornellX+INFO5200+2019_Spring+type
## 4                  /courses/course-v1:CornellX+INFO5200+2019_Spring/xblock/block-v1:CornellX+INFO520
## 5 /courses/course-v1:CornellX+INFO5200+2019_Spring/xblock/block-v1:CornellX+INFO5200+2019_Spring+type
## 6 /courses/course-v1:CornellX+INFO5200+2019_Spring/xblock/block-v1:CornellX+INFO5200+2019_Spring+type
##
## 1 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spring/courseware/53263eda23f148b7ba
## 2 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spring/courseware/53263eda23f148b7ba
## 3 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spring/courseware/53263eda23f148b7ba
## 4 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spring/courseware/53263eda23f148b7ba
## 5 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spring/courseware/53263eda23f148b7ba
## 6 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spring/courseware/53263eda23f148b7ba
##   page event_source                                                event
## 1 <NA>       server            {"POST": {"position": ["8"]}, "GET": {}}
## 2 <NA>       server            {"POST": {"position": ["7"]}, "GET": {}}
## 3 <NA>       server            {"POST": {"position": ["6"]}, "GET": {}}
## 4 <NA>       server {"POST": {"{\\"completion\\":1}": [""]}, "GET": {}}
## 5 <NA>       server            {"POST": {"position": ["5"]}, "GET": {}}
## 6 <NA>       server            {"POST": {"position": ["5"]}, "GET": {}}
##    timestamp
## 1 1548267177
## 2 1548267172
## 3 1548267167
## 4 1548267020
## 5 1548267166
## 6 1548267134
```

```
cl %>% filter(event_source=="browser") %>%
  group_by(page) %>% summarise(n=n()) %>% arrange(desc(n))
```

```
## # A tibble: 283 x 2
##    page                                                                n
##    <chr>                                                           <int>
##  1 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~  4223
##  2 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   370
##  3 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   369
##  4 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   302
##  5 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   273
##  6 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   247
##  7 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   239
##  8 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   231
##  9 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   216
## 10 https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spr~   213
## # ... with 273 more rows
```

```
# Exploring Module States
# add code here
head(m)
```

```
##                           hash_id module_type grade             created
## 1 cbeb5a579f3e8835d69b830fdc394ef5  sequential  NULL 2019-02-04 04:46:24
## 2 cbeb5a579f3e8835d69b830fdc394ef5     chapter  NULL 2019-02-04 04:46:21
```

```
## 3 e6a3a74a176fa00d4384efac5ecc092d  sequential  NULL 2019-02-03 19:23:55
## 4 53b7586de5f4ec9da240b7d776400d55     chapter  NULL 2019-02-03 18:55:02
## 5 16bc6774902e03a2dd667deb09b829f0  sequential  NULL 2019-02-02 19:35:21
## 6 16bc6774902e03a2dd667deb09b829f0  sequential  NULL 2019-02-02 19:35:36
##              modified max_grade
## 1 2019-02-18 14:49:01      NULL
## 2 2019-02-19 00:22:38      NULL
## 3 2019-02-07 02:14:35      NULL
## 4 2019-02-07 16:44:59      NULL
## 5 2019-02-06 14:34:51      NULL
## 6 2019-02-18 13:55:48      NULL
##                                                                                                  module_id
## 1 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@11ccf1767ecf47c68e2ed98563fc2ecc
## 2    block-v1:CornellX+INFO5200+2019_Spring+type@chapter+block@3ff66fc50f6a4bc3ab372a9e70541f80
## 3 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@11ccf1767ecf47c68e2ed98563fc2ecc
## 4    block-v1:CornellX+INFO5200+2019_Spring+type@chapter+block@3ff66fc50f6a4bc3ab372a9e70541f80
## 5 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@9c13103d27894d52b90c77082a714d04
## 6 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@7302b6c0c82c4c008716d73597dffb0f
##   created_timestamp modified_timestamp
## 1        1549255584         1550501341
## 2        1549255581         1550535758
## 3        1549221835         1549505675
## 4        1549220102         1549557899
## 5        1549136121         1549463691
## 6        1549136136         1550498148
```

```
# Exploring Assessment grades
# add code here
head(a)
```

```
##                           hash_id
## 1 f569ac40fe7e41d91e6d9c6bf48f42c7
## 2 f569ac40fe7e41d91e6d9c6bf48f42c7
## 3 f569ac40fe7e41d91e6d9c6bf48f42c7
## 4 f569ac40fe7e41d91e6d9c6bf48f42c7
## 5 f569ac40fe7e41d91e6d9c6bf48f42c7
## 6 f569ac40fe7e41d91e6d9c6bf48f42c7
##                                                                                                  usage_key
## 1 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@556b2a247fe1409bbd91458085949051
## 2 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@0de90422009d4d92a2de261e22f30e9f
## 3 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@42f84ee548b5441eaff8d78c2c597c72
## 4 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@a3a1c488161e46f98657485ffbcb1f81
## 5 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@ffd86a061274412f9f2c405696dc2b3f
## 6 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@d0dceb6594cf4a0d9d58a75f826293b6
##   earned_graded possible_graded first_attempted                  created
## 1             0               0            NULL 2019-02-04 23:01:32.325062
## 2             0               0            NULL 2019-02-11 17:56:43.002992
## 3             0               0            NULL 2019-02-04 23:01:32.335445
## 4             0               0            NULL 2019-02-04 23:01:32.345191
## 5             0               0            NULL 2019-02-04 23:01:32.354557
## 6             0               0            NULL 2019-02-04 23:01:32.364025
##                     modified created_timestamp modified_timestamp
## 1 2019-02-22 20:09:30.662380        1549321292         1550866171
## 2 2019-02-22 20:09:30.608407        1549907803         1550866171
## 3 2019-02-22 20:09:30.680054        1549321292         1550866171
```

```
## 4 2019-02-22 20:09:30.717677        1549321292         1550866171
## 5 2019-02-22 20:09:30.747969        1549321292         1550866171
## 6 2019-02-22 20:09:30.767243        1549321292         1550866171
##   first_attempted_timestamp
## 1                        NA
## 2                        NA
## 3                        NA
## 4                        NA
## 5                        NA
## 6                        NA
```

```
head(a %>% filter(earned_graded!=possible_graded))
```

```
##                            hash_id
## 1 f569ac40fe7e41d91e6d9c6bf48f42c7
## 2 f569ac40fe7e41d91e6d9c6bf48f42c7
## 3 f569ac40fe7e41d91e6d9c6bf48f42c7
## 4 f569ac40fe7e41d91e6d9c6bf48f42c7
## 5 f569ac40fe7e41d91e6d9c6bf48f42c7
## 6 f569ac40fe7e41d91e6d9c6bf48f42c7
##                                                                                                usage_key
## 1 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@68b0f0bfd9fc41eb816f118ad8d1e10d
## 2 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@a993690d1db749b6b7a0fc88e514e031
## 3 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@5a227a68dd224453b1a39dfe90172b7a
## 4 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@369866bc2a9248d9828d89a332d8ceb3
## 5 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@83dc9f838cd646a7961398f9157d33fe
## 6 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@1c39a3b139d842cd8593b6d18bf62b91
##   earned_graded possible_graded first_attempted                   created
## 1             0               1            NULL 2019-02-12 21:34:46.695462
## 2             0               1            NULL 2019-02-12 21:34:46.719318
## 3             0               1            NULL 2019-02-12 21:34:46.740133
## 4             0               1            NULL 2019-02-12 21:34:46.762613
## 5             0               1            NULL 2019-02-12 21:34:46.783571
## 6             0               1            NULL 2019-02-12 21:34:46.804271
##                    modified created_timestamp modified_timestamp
## 1 2019-02-22 20:09:30.727569        1550007287         1550866171
## 2 2019-02-22 20:09:30.757958        1550007287         1550866171
## 3 2019-02-22 20:09:30.784577        1550007287         1550866171
## 4 2019-02-22 20:09:30.803992        1550007287         1550866171
## 5 2019-02-22 20:09:30.823691        1550007287         1550866171
## 6 2019-02-22 20:09:30.852668        1550007287         1550866171
##   first_attempted_timestamp
## 1                        NA
## 2                        NA
## 3                        NA
## 4                        NA
## 5                        NA
## 6                        NA
```

```
head(a %>% filter(earned_graded==possible_graded) %>% filter(possible_graded>0))
```

```
##                            hash_id
## 1 f569ac40fe7e41d91e6d9c6bf48f42c7
## 2 f569ac40fe7e41d91e6d9c6bf48f42c7
## 3 f569ac40fe7e41d91e6d9c6bf48f42c7
```

```
## 4 f569ac40fe7e41d91e6d9c6bf48f42c7
## 5 f569ac40fe7e41d91e6d9c6bf48f42c7
## 6 f569ac40fe7e41d91e6d9c6bf48f42c7
##                                                                              usage_key
## 1 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@bc80acbecf2d43f7b1d24704ff03fdf3
## 2 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@8d65ee1703864c07881fb0420c8a4bf6
## 3 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@2eeda4077996476eb3c5df5df8e511fc
## 4 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@b0f083d8e9b64bbbb92ef99f63883c80
## 5 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@11ccf1767ecf47c68e2ed98563fc2ecc
## 6 block-v1:CornellX+INFO5200+2019_Spring+type@sequential+block@af7025c78a4a46a88efe118e496259f9
##   earned_graded possible_graded     first_attempted
## 1           3.0             3.0 2019-01-23 05:53:34
## 2           0.5             0.5 2019-01-28 04:56:51
## 3           0.5             0.5 2019-01-29 17:09:39
## 4           1.0             1.0 2019-01-30 04:33:57
## 5          15.0            15.0 2019-02-07 05:41:33
## 6          10.0            10.0 2019-02-08 18:58:58
##                      created                   modified created_timestamp
## 1 2019-01-23 19:37:42.853293 2019-02-22 20:09:30.492373        1548272263
## 2 2019-01-25 00:32:05.116035 2019-02-22 20:09:30.518594        1548376325
## 3 2019-01-25 02:11:32.468011 2019-02-22 20:09:30.537703        1548382292
## 4 2019-01-25 02:11:32.480909 2019-02-22 20:09:30.546178        1548382292
## 5 2019-02-02 21:03:35.269470 2019-02-22 20:09:30.599872        1549141415
## 6 2019-01-25 02:11:32.494539 2019-02-22 20:09:30.555071        1548382292
##   modified_timestamp first_attempted_timestamp
## 1         1550866170                1548222814
## 2         1550866171                1548651411
## 3         1550866171                1548781779
## 4         1550866171                1548822837
## 5         1550866171                1549518093
## 6         1550866171                1549652338
```

```
a %>% group_by(hash_id) %>% filter(possible_graded>0) %>% summarise(n=n())
```

```
## # A tibble: 31 x 2
##    hash_id                              n
##    <chr>                            <int>
##  1 16bc6774902e03a2dd667deb09b829f0    22
##  2 23f864f6d16b0c3ca6180faaf8be9952    22
##  3 25b8af686835d57b529244a445767112    22
##  4 485bcdf787dcdb3a39d221a97d7ac12d    22
##  5 501a9941bb5be36a1ccdadd953dd89fe    22
##  6 53a177851bef2579e3fc5e33e7ed9e6d    22
##  7 53b7586de5f4ec9da240b7d776400d55    22
##  8 5f6d08c1ee78d29c860bdf1afa82100f    22
##  9 6e242e38a1d428477c93ec15f3b7f847    23
## 10 803a5223632ed650e8ade059e8fd3bc7    22
## # ... with 21 more rows
```

```
################################################
################################################
```

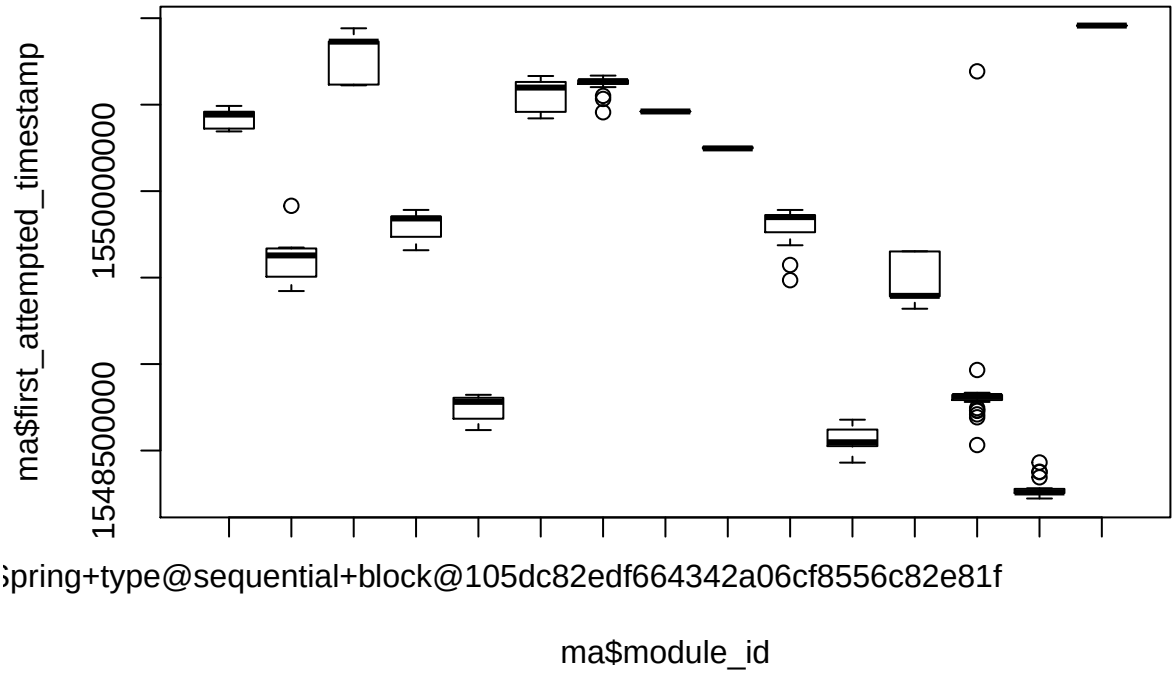
You may notice that it would be helpful to combine the information about grades and time of first attempt with the module state data. Below I make this join for you. See that only 'sequential' modules have grade data associated with them. The boxplot shows when the different sequentials (containing problems) were attempted. This gives you an idea of the order of problems in the course.

```r
ma = m %>% left_join(
    a %>% select(hash_id:possible_graded, first_attempted_timestamp),
    by = c("hash_id"="hash_id", "module_id"="usage_key")
)

# Only sequential modules have a grade associated with them
table(ma$module_type, ma$first_attempted_timestamp>0)
```

```
##
##                   TRUE
##   chapter            0
##   course             0
##   openassessment     0
##   problem            0
##   sequential       356
##   video              0
```

```r
# We see that assignments were due (submitted) at different times
boxplot(ma$first_attempted_timestamp ~ ma$module_id)
```



Step 2: Define a prediction task

Recall the guidelines for defining a good prediction problem covered in the Handbook chapter on prediction. You are looking for something actionable (an opportunity to intervene) and a situation that repeats (so the prediction can be useful in the future). The tradeoff with the dataset you have here is that on the one hand it is very relevant to you but on the other hand it is relatively small. Still, the data is fine-grained and sufficiently messy to give you a taste of LMS data analysis.

The prediction problem for this project is to build a one-day early warning system for missing a graded submission. Specifically, **your goal is to predict one day before the submission deadline, if a student will forget to submit an assignment**, so that the system can send a reminder. As you may have noticed during the data exploration phase above (if not, you should go back and examine this), there are

several graded submissions and some students missed one or more of them. We define **missing a submission** as having an NA for `first_attempted_timestamp` but of course only for those that are past due.

Instructions

1. Treat each graded assignment as a prediction task (thus there are x*n prediction opportunities where x = number of graded assignments and n = 31 students).
2. Create a dataset that has 1 row per student per graded assessment with the binary outcome (did they MISS it? yes/no) and several predictors (see next tip)
3. Predictors (i.e. features) need to be engineered with data from **24hrs before each assignment is due**, which of course varies across assignments; that means you have much more information to predict later assignments than earlier ones
4. Once your dataset is ready, split it into a training and a test set
5. Train a prediction model on the training data; you can try out any of the ones we have covered in the prediction homework and Random Forest
6. Keep tuning your model choice, model parameters (if any), and feature engineering
7. Finally, test your prediction accuracy on the test set

Step 3: Getting you started

Create the outcome variable

Identify the graded assessments and whether a student did NOT submit. Recall we want to have a *warning* system, so the outcome should be the negative action.

Get the outcome for each graded assignment. Figure out the deadline for each and compute the timestamp for 24hrs prior to the deadline. You probably want to use the `ma` dataset I created for you above.

The following table helps you see the various graded assignments to consider. We keep only those where possible_graded > 0. **I define the deadline as the 90th percentile of submissions (you may use this simplification).**

```
adl = ma %>%
    filter(possible_graded > 0) %>%
    group_by(module_id) %>%
    summarise(
        deadline = quantile(first_attempted_timestamp, probs = .9, na.rm=T),
        p_unsubmitted = mean(is.na(first_attempted_timestamp))
    ) %>%
    arrange(deadline) %>%
  filter(p_unsubmitted < 1)
```

Now you know which assessments (module_ids) to target. **Be sure to kick out the one with p_unsubmitted = 1**; it gives you no information.

Question 2: Now build a dataset with an indicator for each person and each of these module_ids with 1=unsubmitted, 0=submitted. Keep track of the deadline: you only want to use features based on data up to 24hrs before it (i.e. `24 * 60 * 60` seconds).

```
##############################################
####### BEGIN INPUT: Define outcome ###########
##############################################

# add code here
outcomeA = inner_join(ma,adl) %>% select(-c("modified","modified_timestamp"))
```

```
## Joining, by = "module_id"
```

```
outcomeA$outcome = ifelse(is.na(outcomeA$first_attempted),1,0)

################################################
################################################
```

Feature Engineering

For each graded assessment, identify what data is appropriate for feature engineering

Before you start feature engineering, you need to constrain the data for **each** assessment.

Remember that the dataset we are aiming for has 1 row per person and assessment with several feature variables and one outcome variable. You created the outcome above. Now you need to create the appropriate features to join. I'm giving you an example for using `deadline = 1550655231` and creating 2 basic features from the clickstream. You should try to create a lot more features, incuding complex ones, that can use the clistream or other datasets (but remember the timing constraint).

```
secs_day = 60 * 60 * 24
example_deadline = 1550655231

example_features = cl %>%
    filter(timestamp < example_deadline - secs_day) %>%
    group_by(hash_id) %>%
    summarise(
        num_events = n(),
        num_seq_goto = sum(event_type=="seq_goto")
    )

head(example_features)
```

```
## # A tibble: 6 x 3
##   hash_id                          num_events num_seq_goto
##   <chr>                                 <int>        <int>
## 1 16bc6774902e03a2dd667deb09b829f0        819            2
## 2 23f864f6d16b0c3ca6180faaf8be9952        430           19
## 3 25b8af686835d57b529244a445767112       1247           35
## 4 485bcdf787dcdb3a39d221a97d7ac12d       1140            6
## 5 501a9941bb5be36a1ccdadd953dd89fe       1024           35
## 6 53a177851bef2579e3fc5e33e7ed9e6d       1080           18
```

Question 3: Engineer features for each student and assessment subject to the timing constraint.

```
################################################
###### BEGIN INPUT: Engineer features #########
################################################

# add code here
for(row in 1:nrow(outcomeA)){
 outcomeA[row,"browserClicks"] = cl %>%
   filter(timestamp<outcomeA[row,"deadline"] - secs_day & hash_id==outcomeA[row,"hash_id"] &
            event_source=="browser") %>% count()

  outcomeA[row,"courseClicks"] = cl %>%
   filter(timestamp<outcomeA[row,"deadline"] - secs_day & hash_id==outcomeA[row,"hash_id"] &
            page ==
            "https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spring/course/") %>%
```

```
    count()

 outcomeA[row,"syllabusClicks"] = cl %>%
   filter(timestamp<outcomeA[row,"deadline"] - secs_day & hash_id==outcomeA[row,"hash_id"] &
            page =="https://edge.edx.org/courses/course-v1:CornellX+INFO5200+2019_Spring/courseware/2d8

  outcomeA[row,"percentBroswerClicksForClass"] = (cl %>%
filter(timestamp<outcomeA[row,"deadline"] - secs_day & hash_id==outcomeA[row,"hash_id"] &
         event_source=="browser") %>%
  count())/(cl %>% filter(timestamp<outcomeA[row,"deadline"] - secs_day &
                            event_source=="browser") %>%
   count())

  temp = outcomeA %>%
    filter(hash_id==outcomeA[row,"hash_id"] &
             first_attempted_timestamp <outcomeA[row,"deadline"] - secs_day)

  outcomeA[row,"priorGradeInClass"] =  ifelse(sum(temp[,"possible_graded"]) == 0, 1,
                                      sum(temp[,"earned_graded"])/sum(temp[,"possible_graded"]))

  modulePageID = as.list(strsplit(outcomeA[row,"module_id"],"@")[[1]])[3]
  outcomeA[row,"assigmentClicks"] = cl %>% filter(timestamp<outcomeA[row,"deadline"] - secs_day &
                                                    hash_id==outcomeA[row,"hash_id"] &
                                                    grepl(modulePageID,page)) %>% count()
}
outcomeA$priorGradeInClass = as.numeric(outcomeA$priorGradeInClass)
##############################################
##############################################
```

Step 4: Split your dataset

Question 4: It is up to you how you choose to split the data but make sure you have enough to train on (i.e. don't make the training set smaller than 70 percent of the data). You can look back at the prediction homework for how to do this.

```
##############################################
######## BEGIN INPUT: Split dataset ############
##############################################
library(splitstackshape)
library(zeallot)
# add code here
c(train,test)%<-% stratified(outcomeA, c("module_id"),.8, bothSets = TRUE)

##############################################
##############################################
```

Step 5: Train your models

Question 5: (a) Train two different prediction models; (b) report the accuracy on the training data for each one. Note: don't forget to do part b, many students lost a point for that last year.

```
##############################################
####### BEGIN INPUT: Train and report #########
```

```
################################################
library(class)
library(rpart)
library(e1071)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

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

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
# add code here
cm_eval = function(cm) {
    list(
        accur = sum(diag(cm)) / sum(cm),
        recall = cm[2,2] / sum(cm[2,]),
        precision = cm[2,2] / sum(cm[,2])
    )
}

m_knn = knn(train[,14:19], train[,14:19], train$outcome, k=2)
m_knn
```

```
##   [1] 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [36] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1
##  [71] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [106] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [141] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
## [176] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [211] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [246] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [281] 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
## [316] 0 1 0
## Levels: 0 1
```

```
m_rf = randomForest(outcome ~ courseClicks + syllabusClicks + browserClicks +
                      percentBroswerClicksForClass + priorGradeInClass + assigmentClicks,
                    data=train)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
m_rf
```

```
##
## Call:
##  randomForest(formula = outcome ~ courseClicks + syllabusClicks +      browserClicks + percentBroswe
##                Type of random forest: regression
```

```
##                       Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 0.07971082
##                     % Var explained: 3.82
```

```
m_class_tree = rpart(outcome ~ courseClicks + syllabusClicks + browserClicks +
                       percentBroswerClicksForClass + priorGradeInClass + assigmentClicks,
                     data=train, method = "class")
m_class_tree
```

```
## n= 318
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 318 29 0 (0.90880503 0.09119497)
##   2) percentBroswerClicksForClass>=0.01655405 285 17 0 (0.94035088 0.05964912) *
##   3) percentBroswerClicksForClass< 0.01655405 33 12 0 (0.63636364 0.36363636)
##     6) percentBroswerClicksForClass< 0.008947596 20  5 0 (0.75000000 0.25000000) *
##     7) percentBroswerClicksForClass>=0.008947596 13  6 1 (0.46153846 0.53846154) *
```

```
ptra_knn = m_knn
cmtra_knn = table(true=train$outcome, predicted=ptra_knn)

ptra_rf = predict(m_rf, train) > .5
cmtra_rf = table(true=train$outcome, predicted=ptra_rf)

ptra_class_tree = predict(m_class_tree, newdata=train, type = "class")
cmtra_class_tree = table(true=train$outcome, predicted=ptra_class_tree)

cm_eval(cmtra_rf)
```

```
## $accur
## [1] 0.9591195
##
## $recall
## [1] 0.5517241
##
## $precision
## [1] 1
```

```
cm_eval(cmtra_knn)
```

```
## $accur
## [1] 0.9402516
##
## $recall
## [1] 0.6551724
##
## $precision
## [1] 0.6785714
```

```
cm_eval(cmtra_class_tree)
```

```
## $accur
## [1] 0.9119497
```

```
##
## $recall
## [1] 0.2413793
##
## $precision
## [1] 0.5384615
```

```
################################################
################################################
```

Step 6: Test your model(s)

Question 6: (a) Predict held-out test data with your models; (b) report the accuracy of your models on the test data. Note: don't forget to do part b, many students lost a point for that last year.

```
################################################
####### BEGIN INPUT: Test and report ##########
################################################

# add code here
ptes_rf = predict(m_rf, test) > .5
cmtes_rf = table(true=test$outcome, predicted=ptes_rf)

m_knn = knn(train[,14:19], test[,14:19], train$outcome, k=5)
ptes_knn = m_knn
cmtes_knn = table(true=test$outcome, predicted=ptes_knn)

ptes_class_tree = predict(m_class_tree, newdata = test, type="class")
cmtes_class_tree = table(true=test$outcome, predicted=ptes_class_tree)

cm_eval(cmtes_rf)
```

```
## $accur
## [1] 0.835443
##
## $recall
## [1] 0.08333333
##
## $precision
## [1] 0.3333333
```

```
cm_eval(cmtes_knn)
```

```
## $accur
## [1] 0.8101266
##
## $recall
## [1] 0
##
## $precision
## [1] 0
```

```
cm_eval(cmtes_class_tree)
```

```
## $accur
## [1] 0.8227848
```

```
##
## $recall
## [1] 0.08333333
##
## $precision
## [1] 0.25
```

```
###############################################
###############################################
```

Step 7: Report

Question 7: Write a brief report. Imagine your supervisor asked you to investigate the possibility of an early warning system. She would like to know what model to use, what features are important, and most importantly how well it would work. Given what you've learned, would you recommend implementing the system? Write your report answering the above questions here:

%######## BEGIN INPUT: Summarize findings ###########

Edx class data was analyzed to determine if an early alert system would benefical for student success. There were 15 assignments from 31 students that we were able to use for the training and testing data sets. The features for east student assesment was calculated from activities that happen on the Edx platform prior to one day before the assignment deadline. Random Forest tree showed to be the best model to use based on predictions from both the training and the test model. None of the features seem to have that strong of an importance in regards to predicting, but total browser clicks on the platform did predict better than other features. The accuracy for the Random Forest was around the .86 mark. After running multiple models and exploring the data, I would recoomment we do not implement this system at this time. I think more unsubmitted assignment data would be need to create a more accurate prediction. In the meantime, I would recommend an alert system to be sent to all students who have not submitted the assignment a day before the deadline.

%##

Submit Project

This is the end of the project. 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.