Project: LMS Data Analytics

INFO 5200 Learning Analytics

[[Cole Walsh, 4399966]]

Introduction

Disclaimer: 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`.

Project Goal: 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. This is an export of the class's edX data from Jan 23 to Feb 23. I have done some data cleaning for you and anonymized the datasets. However, there is plenty of real-world messiness for you to tackle. 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 your data and your know the platform. Moreover, you can navigate the relevant pages on edX to see what page/action the data refers to.

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):

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)

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

In the space below, explore each dataset e.g. using `head()`, `str()`, `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
```

```
# 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
```

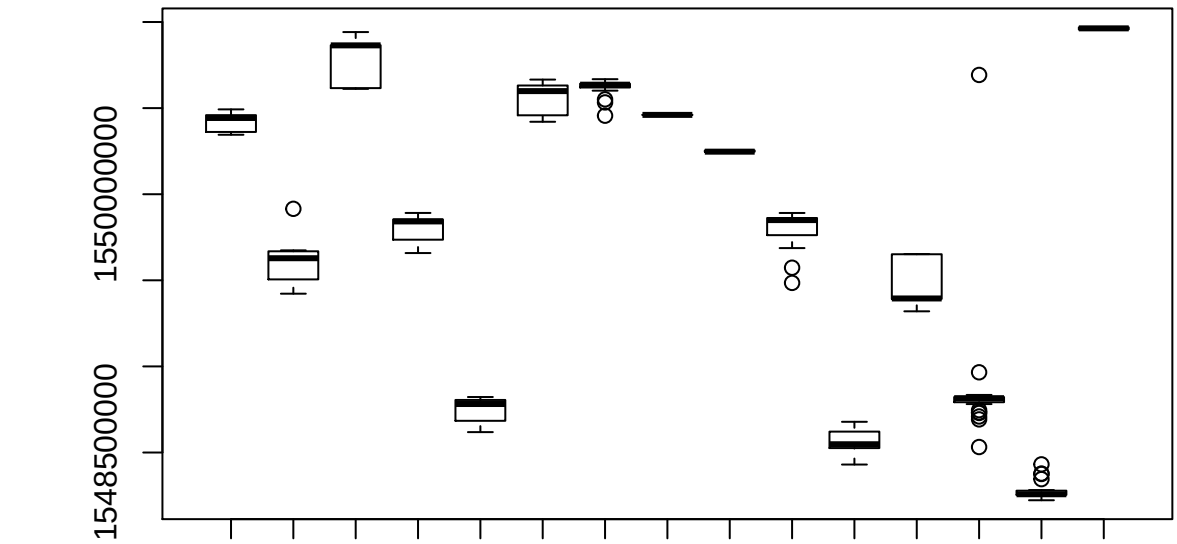
```
################################################
################################################
```

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.

```
# I left joined on a instead of m because if a student did not access any content they
#would not have an associated observation for that module in m, but every student
#received a grade on an assignment

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

colnames(ma)[colnames(ma)=="usage_key"] <- "module_id"

table(ma$module_type, ma$first_attempted_timestamp>0)
```

```
##
##              TRUE
##   sequential  356
```

```
boxplot(ma$first_attempted_timestamp ~ ma$module_id)
```



spring+type@sequential+block@105dc82edf664342a06cf8556c82e81f

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 if one day before the deadline, whether 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 = num. 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 the 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 needs 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).**

```
dat_deadline <- 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)
```

```
dat_deadline
```

```
## # A tibble: 15 x 3
##    module_id                                          deadline p_unsubmitted
##    <chr>                                                 <dbl>         <dbl>
##  1 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.387
##  2 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0645
##  3 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0323
##  4 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0968
##  5 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0968
##  6 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.226
##  7 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0645
##  8 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.194
##  9 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0323
## 10 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.968
## 11 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0645
## 12 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0968
## 13 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.161
## 14 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0.0645
## 15 block-v1:CornellX+INFO5200+2019_Spring+type@se~    1.55e9        0
```

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.

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 in based on data up to 1 day before it (i.e. `24 * 60 * 60` seconds).

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

# add code
ma_deadline <- inner_join(ma, dat_deadline, by = 'module_id') %>%
  mutate(unsubmitted = 1*is.na(first_attempted_timestamp))

# So there's this weird thing where two students have completed a certain module, but no
# one else has and possible_graded = 1 for those 2 students and 0 for everybody else...
# I take it this is a module that was still open when this data was pulled and these two
# students were overachievers who completed it early, but its not necessarily unsubmitted
# for the rest of the students...for the purposes of this project I'm going to drop this
# module since this behaviour is different than the other 14 modules that have already
# closed.

Open_Assessments <- ma_deadline %>%
  group_by(module_id) %>%
  summarize(N_Students_Open = sum(possible_graded == 0))

Open_Assessments
```

```
## # A tibble: 15 x 2
##    module_id                                          N_Students_Open
```

```
##    <chr>                                                          <int>
##  1 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
##  2 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
##  3 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
##  4 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
##  5 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
##  6 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
##  7 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
##  8 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
##  9 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
## 10 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
## 11 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
## 12 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
## 13 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
## 14 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~            0
## 15 block-v1:CornellX+INFO5200+2019_Spring+type@sequential~           29
```

```r
ma_deadline_NoOpen <- inner_join(ma_deadline, Open_Assessments, by = 'module_id') %>%
  filter(N_Students_Open == 0) %>%
  select(-c('N_Students_Open'))

# Thus, my dataframe has 31*14 = 434 rows

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

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 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).

```r
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
```

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

# add code here
# Creating features based on clickstream data, including total number of clicks before
#deadline and for various time intervals before deadline, how many times a student has
#jumped between sequences, and how many times they have saved and checked problems
#including recently.

Click_Features_df <- left_join(ma_deadline_NoOpen, cl, by = 'hash_id') %>%
  filter(timestamp < deadline - secs_day) %>%
  group_by(hash_id, module_id) %>%
  summarize(TotalNum_Events = n(),
            Num_Events_last7days = sum(timestamp >= deadline - 8 * secs_day),
            Num_Events_last3days = sum(timestamp >= deadline - 4 * secs_day),
            Num_Events_last1day = sum(timestamp >= deadline - 2 * secs_day),
            Num_Seq_Goto = sum(event_type == 'seq_goto'),
            Num_Seq_Next = sum(event_type == 'seq_next'),
            Num_Seq_Prev = sum(event_type == 'seq_prev'),
            Num_Save = sum(event_type == "problem_save"),
            Num_Save_last1day = sum((timestamp >= deadline - 2 * secs_day) &
                                      (event_type == "problem_save")),
            Num_Check = sum(event_type == "problem_check"),
            Num_Check_last1day = sum((timestamp >= deadline - 2 * secs_day) &
                                       (event_type == "problem_check")))

# Creating features based on students submissions to modules due more than 24 hours before
# the current deadline, including how many and what fraction of previous submissions (due
# at least 24h earlier) a student missed and how many and what fraction of previous
# submissions were submitted within the last 24h before the deadline.

library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.5.1
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
ma_deadline_copy = copy(ma_deadline_NoOpen)
ma_pastDeadlines <- left_join(ma_deadline_NoOpen, ma_deadline_copy, by = 'hash_id') %>%
  filter(deadline.y < deadline.x - secs_day) %>%
  group_by(hash_id, module_id.x) %>%
  summarize(N_prev_unsubmitted = sum(unsubmitted.y),
            frac_prev_unsubmitted = mean(unsubmitted.y),
            N_prev_submit_last24 = sum(first_attempted_timestamp.y > deadline.y - secs_day),
            frac_prev_submit_last24 = mean(first_attempted_timestamp.y > deadline.y - secs_day))

# I also keep the highest possible grade of the assigment as a predictor variable

Deadlines_Features_df <- left_join(ma_deadline_NoOpen[, c('hash_id', 'module_id',
                                                          'possible_graded', 'unsubmitted')],
                                   ma_pastDeadlines, by = c('hash_id', 'module_id' =
                                                            'module_id.x'))

df <- left_join(Deadlines_Features_df, Click_Features_df, by = c('hash_id', 'module_id')) %>%
  select(-c('hash_id', 'module_id'))
df[is.na(df)] <- 0
head(df)
```

```
##   possible_graded unsubmitted N_prev_unsubmitted frac_prev_unsubmitted
## 1             1.0           1                  0                     0
## 2             3.0           0                  0                     0
## 3             0.5           0                  0                     0
## 4             0.5           0                  0                     0
## 5             1.0           0                  0                     0
## 6            15.0           0                  0                     0
##   N_prev_submit_last24 frac_prev_submit_last24 TotalNum_Events
## 1                    7               0.7777778            1135
## 2                    0               0.0000000              87
## 3                    0               0.0000000              87
## 4                    1               0.5000000             176
## 5                    1               0.5000000             176
## 6                    3               0.7500000             902
##   Num_Events_last7days Num_Events_last3days Num_Events_last1day
## 1                  233                   83                  43
## 2                   87                   87                  87
## 3                   87                    0                   0
## 4                  176                   89                  87
## 5                  176                   89                  14
## 6                  432                  241                 135
##   Num_Seq_Goto Num_Seq_Next Num_Seq_Prev Num_Save Num_Save_last1day
## 1           21            7            4        0                 0
## 2            3            1            0        0                 0
## 3            3            1            0        0                 0
## 4            4            4            2        0                 0
## 5            4            4            2        0                 0
## 6           20            7            4        0                 0
##   Num_Check Num_Check_last1day
## 1         8                  0
## 2         2                  2
## 3         2                  0
```

```
## 4         4                 2
## 5         4                 0
## 6         6                 0
```

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

Step 4: Split your dataset

It is up to you how you choose to split the data but make sure you have enough to train on. You can look back at the prediction homework for how to do this.

```
################################################
######## BEGIN INPUT: Split dataset ############
################################################

# add code here
ids_train = sample(nrow(df), size = nrow(df) * 0.8)

train = df[ids_train, ]
test = df[-ids_train, ]

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

Step 5: Train your model(s)

1. Train one or more prediction models
2. Report the accuracy on the training data

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

# add code here
library(caret) #imported for confusionMatrix function so I don't need to
```

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

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
# manually compute accuracy, parameter tuning is still done manually for randomForest.
library(randomForest)
```

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

```
## 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
```

```r
library(e1071)
```

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

```r
# Check different values of mtry for highest in-sample accuracy in randomForest
print('Checking Accuracy of random forest with training data and various values for mtry...')
```

```
## [1] "Checking Accuracy of random forest with training data and various values for mtry..."
```

```r
for (mtry in 1:(ncol(train) - 1)){
  rf_model <- randomForest(as.factor(unsubmitted) ~ ., data = train, mtry = mtry)
  rf_train_pred <- predict(rf_model, train)
  print(mtry)
  print(confusionMatrix(rf_train_pred, as.factor(train$unsubmitted))$overall[1])
}
```

```
## [1] 1
##  Accuracy
## 0.8616715
## [1] 2
##  Accuracy
## 0.9855908
## [1] 3
##  Accuracy
## 0.9913545
## [1] 4
##  Accuracy
## 0.9971182
## [1] 5
##  Accuracy
```

```
## 0.9971182
## [1] 6
##  Accuracy
## 0.9971182
## [1] 7
##  Accuracy
## 0.9971182
## [1] 8
##  Accuracy
## 0.9971182
## [1] 9
##  Accuracy
## 0.9971182
## [1] 10
##  Accuracy
## 0.9971182
## [1] 11
##  Accuracy
## 0.9971182
## [1] 12
##  Accuracy
## 0.9971182
## [1] 13
##  Accuracy
## 0.9971182
## [1] 14
##  Accuracy
## 0.9971182
## [1] 15
##  Accuracy
## 0.9971182
## [1] 16
##  Accuracy
## 0.9971182
```

```
# mtry = 4 gave the highest in-sample accuracy so I use mtry = 4 in my final rf model
rf_model <- randomForest(as.factor(unsubmitted) ~ ., data = train, mtry = 4)

log_model <- glm(unsubmitted ~ ., data = train, family = 'binomial')
log_train_pred <- predict(log_model, train, type = 'response')
print('Accuracy of logistic regression with training data...')
```

```
## [1] "Accuracy of logistic regression with training data..."
```

```
confusionMatrix(as.factor(1*(log_train_pred > 0.5)), as.factor(train$unsubmitted))$overall[1]
```

```
## Accuracy
## 0.832853
```

```
nb_model <- naiveBayes(as.factor(unsubmitted) ~ ., train)
nb_train_pred <- predict(nb_model, train)
print('Accuracy of Naive Bayes with training data...')
```

```
## [1] "Accuracy of Naive Bayes with training data..."
```

```
confusionMatrix(nb_train_pred, as.factor(train$unsubmitted))$overall[1]
```

```
##  Accuracy
## 0.4178674
```

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

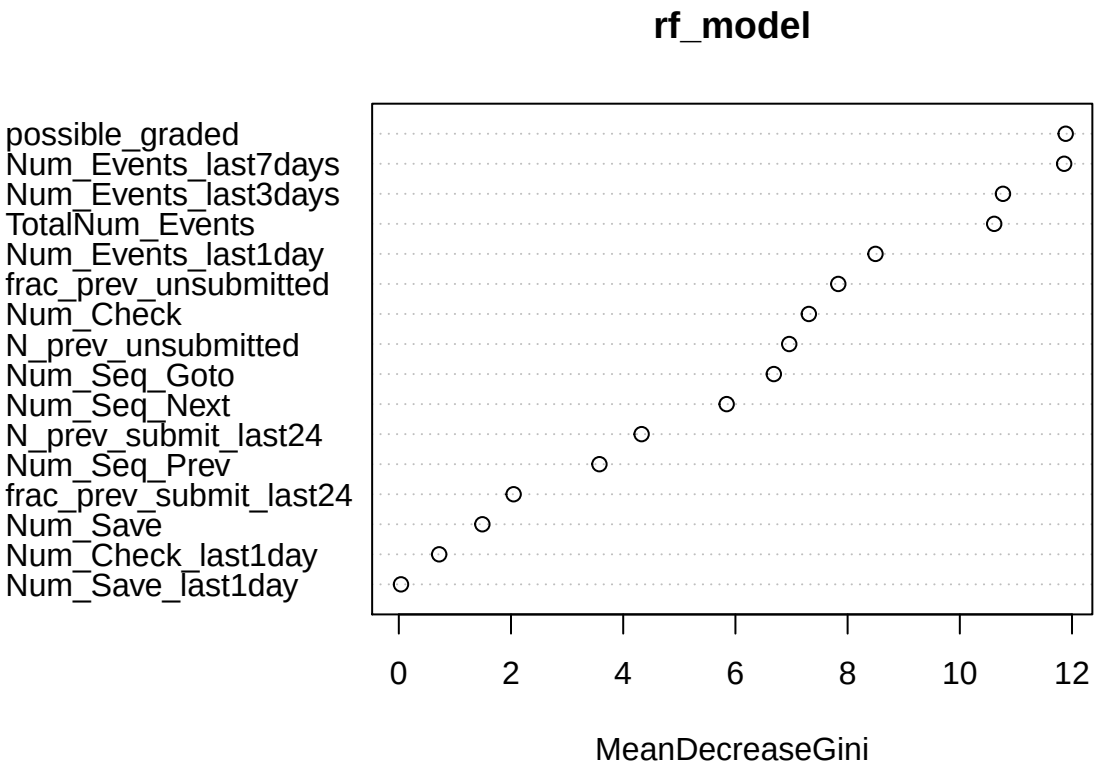
Step 6: Test your model(s)

1. Predict held-out test data with your model
2. Compute the accuracy of your model

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

# add code here
rf_pred <- predict(rf_model, test)
confusionMatrix(rf_pred, as.factor(test$unsubmitted))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 69 11
##          1  4  3
##
##                Accuracy : 0.8276
##                  95% CI : (0.7316, 0.9002)
##     No Information Rate : 0.8391
##     P-Value [Acc > NIR] : 0.6787
##
##                   Kappa : 0.1999
##  Mcnemar's Test P-Value : 0.1213
##
##             Sensitivity : 0.9452
##             Specificity : 0.2143
##          Pos Pred Value : 0.8625
##          Neg Pred Value : 0.4286
##              Prevalence : 0.8391
##          Detection Rate : 0.7931
##    Detection Prevalence : 0.9195
##       Balanced Accuracy : 0.5797
##
##        'Positive' Class : 0
##
```

```r
varImpPlot(rf_model)
```




```r
log_pred <- predict(log_model, newdata = test, type = 'response')
confusionMatrix(as.factor(1*(log_pred > 0.5)), as.factor(test$unsubmitted))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 70 13
##          1  3  1
##
##                Accuracy : 0.8161
##                  95% CI : (0.7186, 0.8911)
##     No Information Rate : 0.8391
##     P-Value [Acc > NIR] : 0.77186
##
##                   Kappa : 0.0426
##  Mcnemar's Test P-Value : 0.02445
##
##             Sensitivity : 0.95890
##             Specificity : 0.07143
##          Pos Pred Value : 0.84337
##          Neg Pred Value : 0.25000
##              Prevalence : 0.83908
```

```
##          Detection Rate : 0.80460
##    Detection Prevalence : 0.95402
##       Balanced Accuracy : 0.51517
##
##        'Positive' Class : 0
##
```

```
varImp(log_model)
```

```
##                            Overall
## possible_graded         1.35187485
## N_prev_unsubmitted      2.84053217
## frac_prev_unsubmitted   2.53347084
## N_prev_submit_last24    2.60786952
## frac_prev_submit_last24 1.83385853
## TotalNum_Events         0.74845078
## Num_Events_last7days    0.99566134
## Num_Events_last3days    1.05233566
## Num_Events_last1day     1.18805309
## Num_Seq_Goto            2.08660740
## Num_Seq_Next            0.45883013
## Num_Seq_Prev            0.70821876
## Num_Save                0.49471688
## Num_Save_last1day       0.01567679
## Num_Check               2.71820705
## Num_Check_last1day      2.37100095
```

```
nb_pred <- predict(nb_model, test)
confusionMatrix(nb_pred, as.factor(test$unsubmitted))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 15  1
##          1 58 13
##
##                Accuracy : 0.3218
##                  95% CI : (0.2256, 0.4306)
##     No Information Rate : 0.8391
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0507
##  Mcnemar's Test P-Value : 3.086e-13
##
##             Sensitivity : 0.2055
##             Specificity : 0.9286
##          Pos Pred Value : 0.9375
##          Neg Pred Value : 0.1831
##              Prevalence : 0.8391
##          Detection Rate : 0.1724
##    Detection Prevalence : 0.1839
##       Balanced Accuracy : 0.5670
```

```
##
##        'Positive' Class : 0
##
```

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

Step 7: Report

Write down a brief report. Imagine your direct supervisor asked you to investigate the possibility of an early warning system. She would like to know what model you 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 reponse answering the above questions below:

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

Add your summary here.

The Naive Bayes model is terrible, don't use this. The logistic regression and random forest models both provide reasonable predictions, though it is worth pointing out that just predicting that everyone submitted every assignment in the test set would have given an accuracy of 88.5% so accuracy itself is not necessarily the best measure to use here for an unbalanced dataset. For this reason, I would argue that the random forest model is the one that should be implemented given that out of 10 unsubmitted assignments, it predicted 4 with a tradeoff of also having 4 false positives. I think this makes this system implementable. The random forest model indicates that the number of total clicks before the deadline, indicating active users, and the number of clicks closer to the deadline, indicating recently active users, are the most important... as is the point value of the assignment (see variable importance plot above). The logistic regression model found that three variables have an associated t-statistic greater than 2, indicating that these coefficients are significant at the alpha = 0.05 significance level. These variables were the number of assignments that were previously not submitted (again, only using assignments whose deadline was at least 24h before the assignment in question) the number of times problems were checked, and the number of times problems were checked in the time period 24-48h 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.