Catch Them Before They Leave:
Using Deep Learning to Predict Academic Outcomes at a Public University

William Albert
Stockton University

Summary:

Educators at colleges and universities have a number of reasons to be concerned with student retention and graduation rates. It is important for public perception of the institution, as well as for the student on an individual level. With all of the data available to education professional, there should be more technology-assisted tools available to identify students who are at-risk of ending their college careers prematurely.

Previous research has found success in using artificial neural networks to predict different types of academic performance, including graduation and retention, with some success. Many of those studies, however, focused on pre-admission background data as input variables. Here, I primarily used student performance – grades earned – in their first year at Stockton University. My goal was to create a neural network model that would identify students who would leave the university before completing their degree.

I identified students who were invited to attend new student orientation during the summer of 2015 as my target sample. I had their names and ID numbers available (one of my job functions is to assist students with registration at orientation), and was able to obtain course records, transfer credits, and different status indicators from the Degree Works curriculum planning database, with some help from Stockton Information Technology Services. After much cleaning and rearranging of the data in R, I had a complete sample of 1,295 students.

My input variables were highest and lowest grades earned in each of the students' first two semesters – fall 2015 and spring 2016 (four variables), as well as total number of courses withdrawn in the first year, and the number of transfer credits, for six variables in all. My output variables were three binary classifications – whether or not a student: graduated on time in four years, was still continuing at Stockton, or had left the university before completing their degree.

Unfortunately, my model was mostly unsuccessful in identifying the students who "left early" in a test data set from my sample, though it did reasonable well at predicting if they would graduate. If nothing else, this study provides a possible framework and starting point for future research in this area.

Full Report:

Retention is an incredibly important concept in higher education. I know that statement is begging for a citation, and a simple Google search containing the words, “retention,” “higher,” and “education” will turn up endless results, but the simple fact is that there are plenty of reasons why it is desirable for college students to experience success in and out of the classroom and hopefully earn a degree in a reasonable amount of time. At a big-picture level, it can affect the public reputation of an institution. If a relatively low percentage of its students aren’t completing degrees, it could lead prospective students and their parents question whether it would be a worthwhile investment, lead to lower overall enrollment, and end in decreased funding. On a more individual level, educators and higher education professionals should want their students to succeed. Not only to support their own interests (via the funding), but because they should, and many do (derived anecdotally from personal experience), genuinely care for their students. According to the US Department of education (https://www.ed.gov, 2016), having a bachelor’s degree can mean earning (“typically”) 66 percent more and “approximately” a million more dollars over a lifetime, compared to those who do not have one. And, of course, it is important for the student to get done in a timely fashion because every extra semester can mean a lot more money spent (see: any college website/tuition page).

So what can educators do to ensure that their students stick around and graduate on time? Instructors can try to engage with students who do poorly in their classes, and refer them to academic advisors or offices whose function it is to assist students in a variety of ways. More proactively, a university could have a framework in place, much like Stockton University’s preceptorial advising system, to ensure that students regularly meet with a faculty member who can function as both an academic advisor and (ideally) a type of mentor. Then again, it is up to the student to actually meet with the advisor or preceptor, so that certainly adds an element of difficulty. Perhaps one possibility would be to utilize some kind of technology to identify students who might be having some trouble and reach out to them before they prematurely end their college career.

Colleges sure do have a lot of data available. Data is collected from every student from the moment they submit their application – or before, depending on how aggressive the admissions office is – until after they graduate, through organizations such as alumni networks and honor societies. One previous study, “An Artificial Neural Network for Predicting Student Graduation Outcomes,” (Karamouzis & Vrettos, 2008) used a variety of pre-admission data to predict graduation success using an artificial

neural network. Their input variables included ethnicity, gender, age at application, their high school, whether or not they needed disability or support services, and a few others. Another, "Application of an Artificial Neural Network to Predict Graduation Success at the United States Military Academy," (Lesinski, Corns & Dagli, 2016) also used pre-admission data including high school class rank, standardized (SAT/ACT) test scores and parents' levels of education, to predict graduation success. A third, "Predicting General Academic Performance and Identifying the Differential Contribution of Participating Variables using Artificial Neural Networks," (Musso, et. al., 2013) went in a different direction. They used data from cognitive performance assessments and questionnaires, along with some background information (parents' occupations and education) to predict student performance in terms of grade point average (split into three categories).

Background data is great; all three studies achieved varying levels of success. However, way too much data is generated during a student's time in school to be ignored (note: it's probably not being ignored, but my research turned up mostly studies using pre-admission data). Every class for which a student registers, the type of class, the program, the term, and most importantly, the grade earned (or lack thereof), is recorded in a database. Stockton primarily uses the Banner system to record, store, and access this data, but also uses the curriculum planning tool Degree Works. With the assistance of a colleague in Stockton Information and Technology Services (Tom DiMond), I was able to access this student course data through Degree Works.

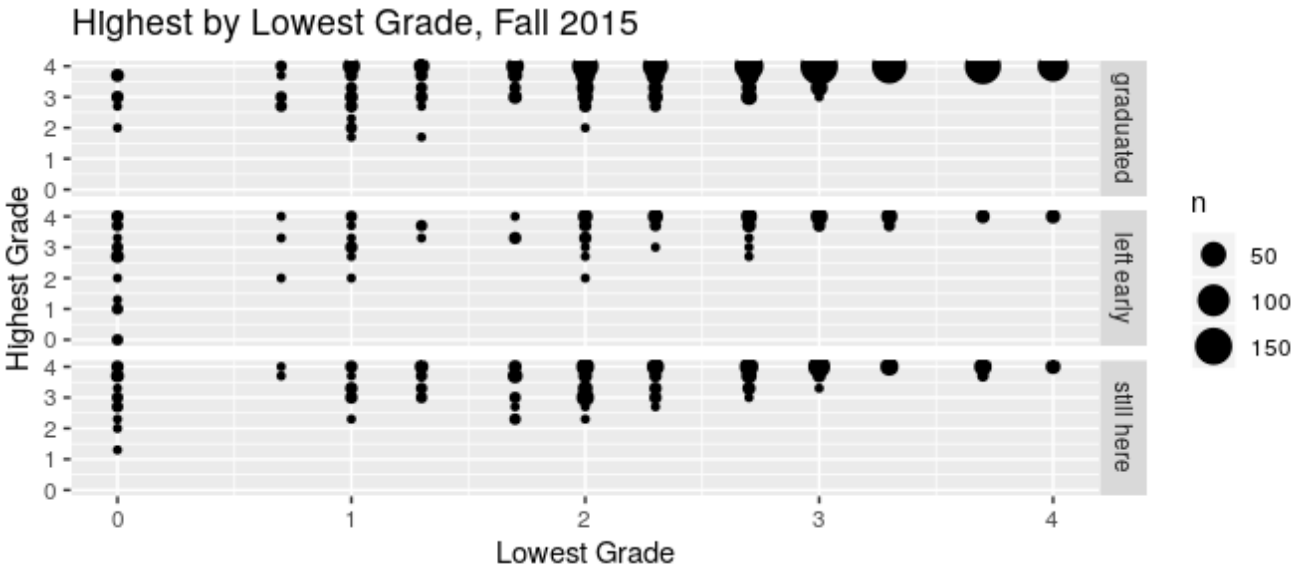
I already had access to lists of students who had been invited to new student orientations for the past several years, so I decided to look at the most recent group of students who would have had a full four years to complete their degree. I was able to get course data for 1,671 students who were invited to attend orientation during summer 2015. Of those, 1,295 completed a full school year, taking classes in both the Fall 2015 and Spring 2016 semesters. These students included both first-time freshmen and transfers, but only students who were attending Stockton for the first time (some had taken classes at Stockton in previous years). While I had data for all of their courses, it seemed that the first two semesters seemed a reasonable amount of data to use while still having time to intervene with students who might be at risk to not finish their degree. In order to control for the fact that students do not all take the same number of courses each semester, I included only two classes for each student in each of the two semesters – their highest grade and their lowest. Only students who took at least two courses in each semester were included in the analysis. I also wanted to look at the total number of withdrawals

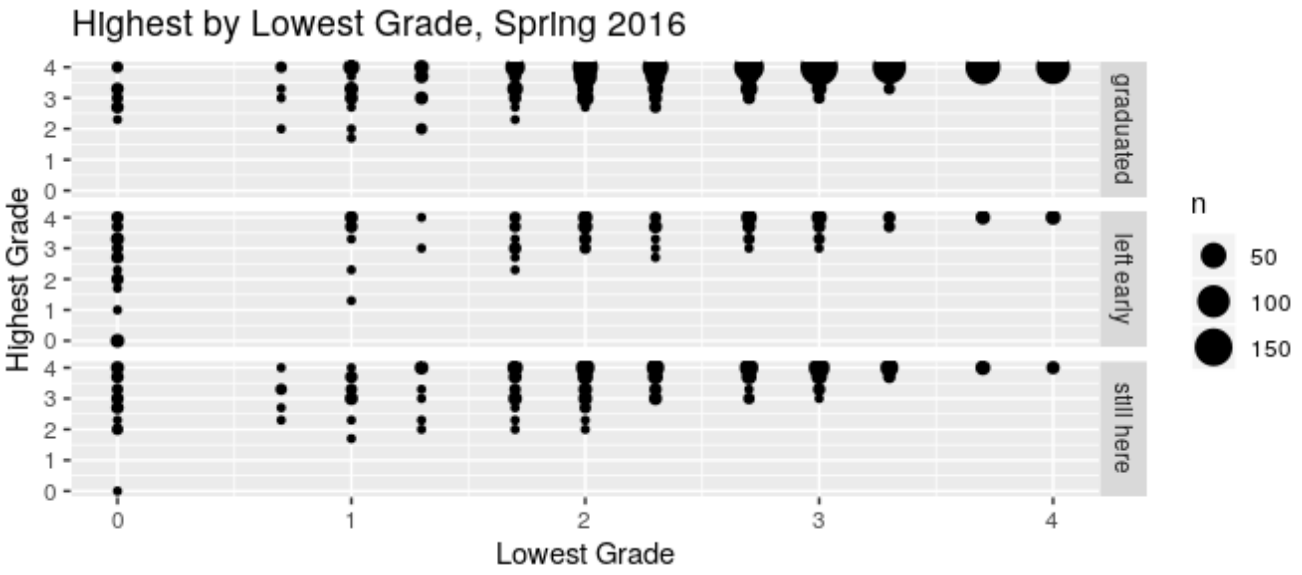
for the full academic year, as well as the number of general studies courses, and transfer credits (my only pre-admission variable).

My goal was to predict whether a student graduated in four years, left Stockton early (for whatever reason), or was still considered an active student. I used a combination of student status variables available in Degree Works to make those determinations. Of the 1,295 students included in the analysis, 946 graduated in four years (whether as freshmen or transfers) and 228 were still active students. 121 did not graduate and were no longer considered active students. It is those 121 who I would most want to be able to identify early to try to help them finish their degree.

I am very grateful for the data I was able to obtain, but it required quite a bit of reorganizing. For this, I used R and parts of the "Tidyverse" library (see Appendix A). The course data file was in "long" format. Each course was in its own row, so each student had many rows – a student beginning as a freshman with no credits would take 32 classes to earn a bachelor's degree at Stockton. I needed the data in "wide" format so that each student only had one row, so that my final dataframe could be converted to a matrix to use as input for a neural network. Basically, I took the long dataframe, chopped it up into a bunch of smaller dataframes, and joined them all back together in the desired format. Through that process, I ended up with the 1,295 cases and six variables that I ended up using for my analysis. I then had a separate file that contained transfer credits and different status variables, including current student status, graduation status, and level (undergraduate vs. graduate). Some of the students had graduated and continued in graduate programs at Stockton, so I had to make sure they were coded as graduated and filter out their graduate student records. After a little more cleaning, the students' transfer credits and student status were joined with their course data.

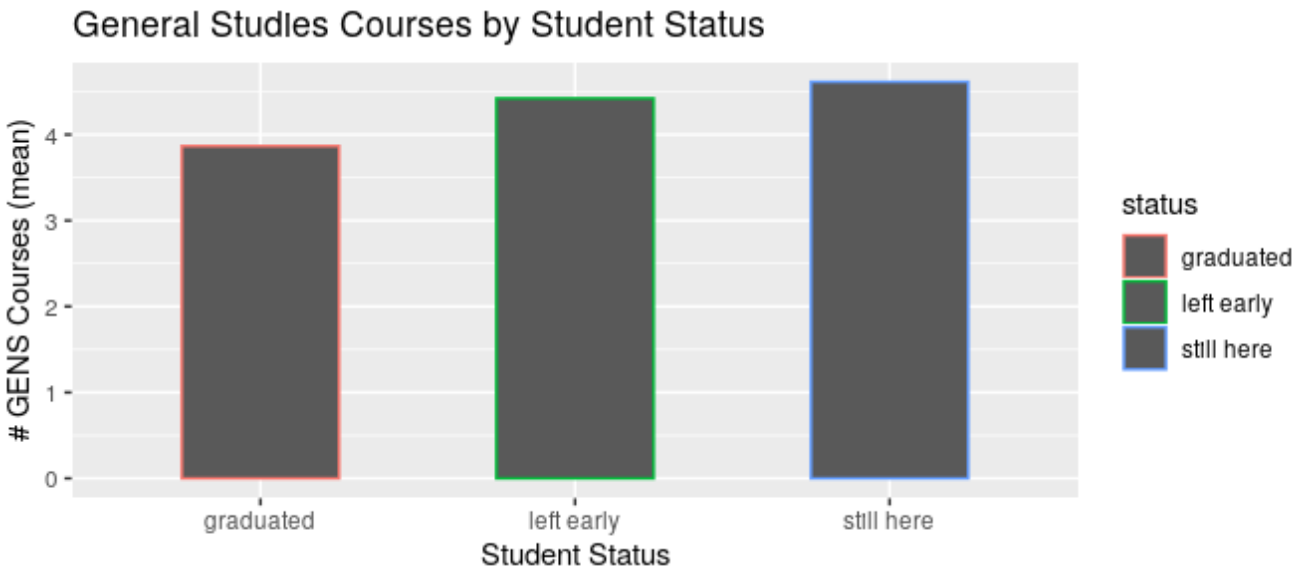
Grades were in letter format, so those were recoded as numbers - A=4, B=3, and so on - with plusses 0.3 above and minuses 0.3 below. Once everything was broken down, rebuilt, and ready to go, I first looked at the high and low grades as they related to student status (graduated, left early, or still here (classified active).





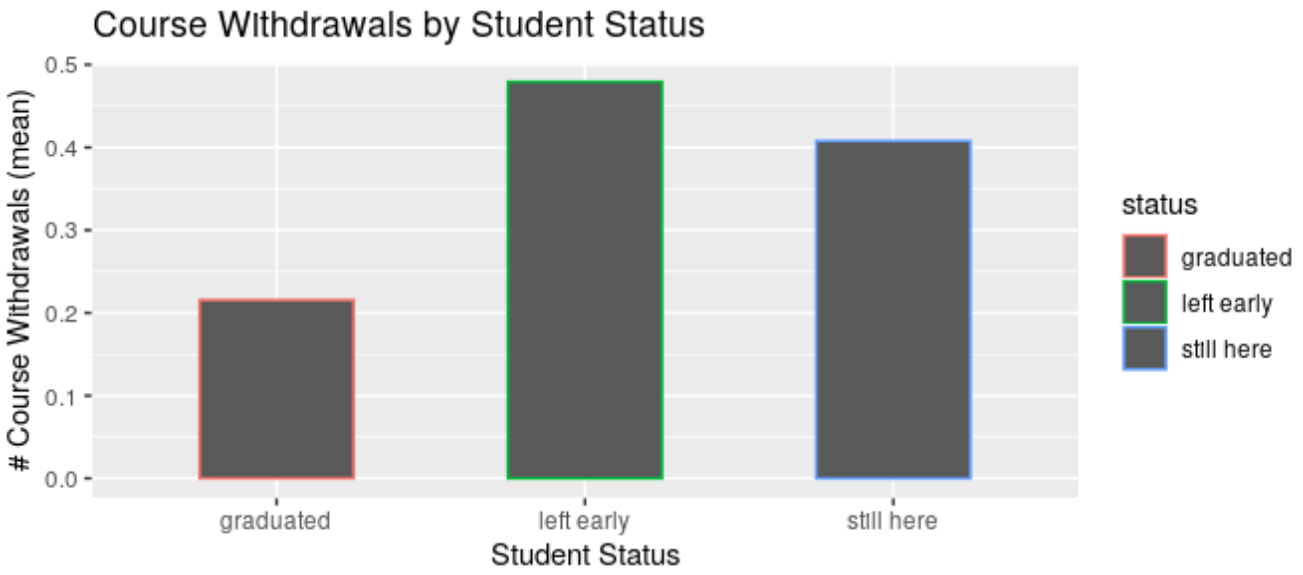
For the charts above, the marker size indicates the number of students who share the same data point, or combination of high grade for the semester, low grade, and status category. It is pretty clear that the students who graduated on time had grades on the high end (even their lowest grades) and students who left early had proportionally more (low and high) grades on the lower end.

The next variable I was interested in exploring was the number of general studies classes each student took their first year. General studies classes are one of the things that makes Stockton special (in my opinion; I know others agree), and I was curious to see if there was any relationship with retention.



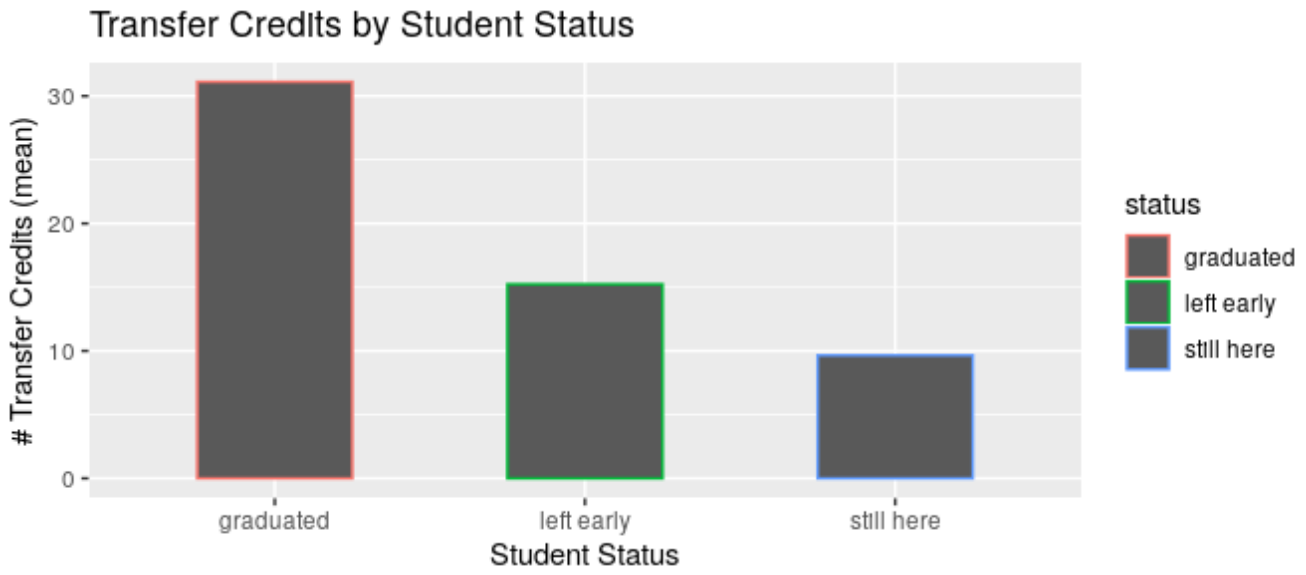
Not quite. I was surprised that the group that graduated on time actually took less (on average) in their first year than the other two groups, even if it was a small difference. I ultimately decided to exclude this variable from my model, not because it didn't show what I wanted, but because in the end I didn't really see how it would add that much information to the analysis.

The number of classes a student withdrew from in their first year definitely seemed like a variable that would indicate if a student was likely to leave early.



The scale is small when the mean is calculated, but the difference is apparent. Those who graduated on time had fewer withdrawals in their first year, on average than both students who left early and continuing students. This would join the high and low grade for each semester as the fifth variable.

For the sixth, I did want to use one pre-admission variable, so I was fortunate to have each student's transfer credits.



So, of course students with more transfer credits would be more likely to graduate in four years (or less), but it also looks like there may have been more transfer students in the "left early" group than in the "still here" group. I certainly think that given more data to work with, it would be worth looking at first-time freshmen and transfer students separately. For this project, though, it made enough sense to look at them together.

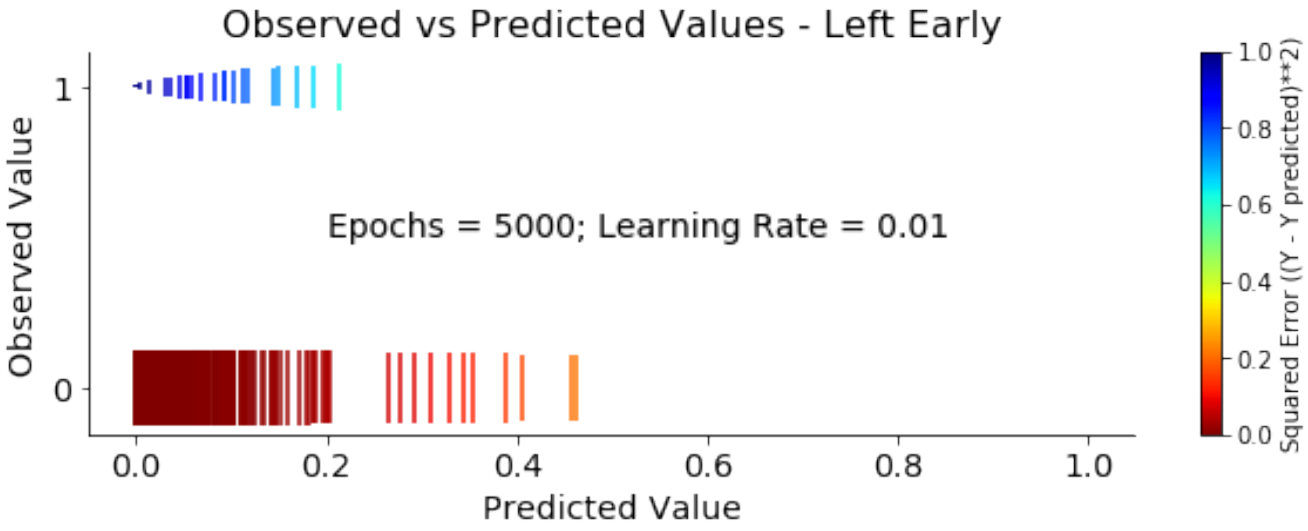
In the end, I had six input variables – high grade fall '15, low grade fall '15, high grade spring '16, low grade spring '16, withdrawals in the first year, and transfer credits. In order to get all values between 0 and 1 for neural network input, the grade variables were divided by 4 (the maximum), withdrawals were divided by the total number of classes taken in the first year, and transfer credits were divided by 96 – the maximum that would be accepted at Stockton.

Keras and Tensorflow in Python were used to build a sequential artificial neural network. After trying out a few different network configurations, I settled on one hidden layer with 18 neurons (following the 6-neuron input layer. Both the input and hidden layers used a RELU activation function. The output layer had three neurons, one for each student status classification, and used a sigmoid activation function. Binary crossentropy was used for the loss function, the learning rate was set at 0.01, and the

model was run for 5000 epochs. Initially, all of the data was used to experiment with different configurations, but ultimately, a 75/25 train/test split was used.

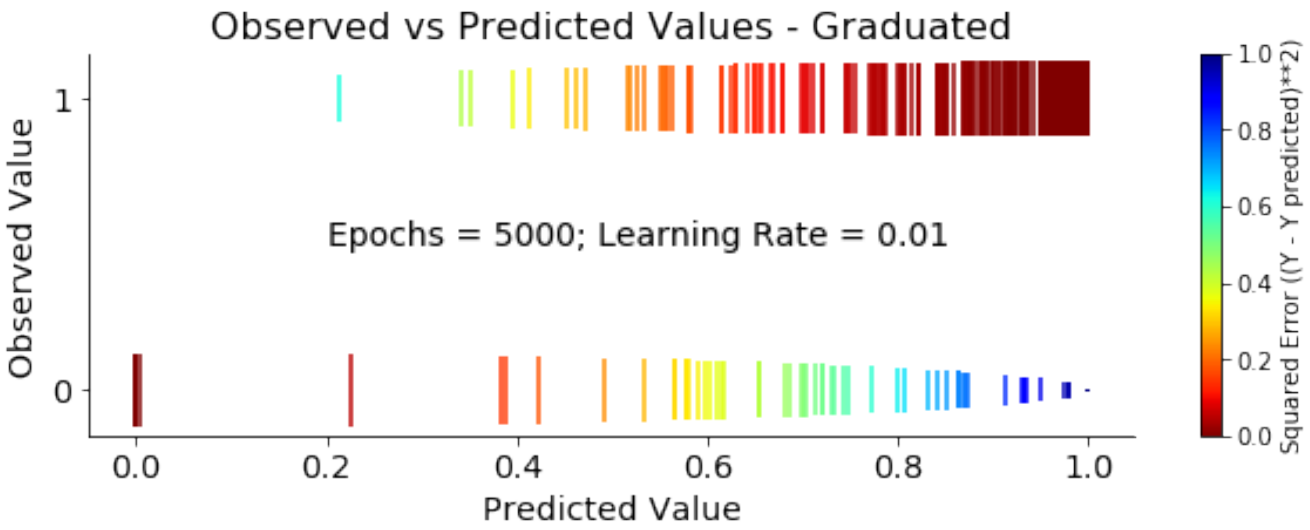
Overall, the model had an accuracy of 85.55%. Similar values (all between 80 and 85%) were seen with different configurations. Unfortunately, when predicting outcomes for the test data, it failed to accurately predict any of the students who left early, as shown by the confusion matrix and plot below.

```
Confusion Matrix
================
True negatives:  295
False negatives:  28
False positives:  1
True positives:  0
```



It did, at least, do quite a bit better predicting which students would graduate on time…

```
Confusion Matrix
================
True negatives:  10
False negatives:  8
False positives:  55
True positives:  251
```



Overall, I feel that there were mixed results here. Throughout the data exploration process, it seemed that many of my hypotheses had proved to be reasonably accurate, with how the students' high and low grades related to their graduation status, as well as their transfer credits and withdrawals. Though my accuracy score was decent, I did not succeed in correctly predicting even one student who left early in the test set. All of the other studies mentioned above had much better prediction accuracy (even though one was looking at GPA and not graduation). The best of the three studies (in my opinion) – Lesinski, Corns & Dagli – did have a much larger sample of 5,100 students to work with. However, their focus was more on predicting whether students would graduate on time rather than if they were at risk to leave prematurely.

I think the sample size in my study was definitely the biggest limitation, especially when it was split into training and testing sets to use with the neural network. The three groups were very uneven, and there just wasn't enough information on students who left early to properly train the network. I do believe this methodology has some promise, and perhaps with a larger sample and a few more input variables, the model could catch enough of the "left early" students to make it worth the time and effort that would be required to clean the data.

References

https://www.ed.gov/news/press-releases/fact-sheet-college-degree-surest-pathway-expanded-opportunity-success-american-students

Karamouzis, Stamos, and Andreas Vrettos, “An Artificial Neural Network for Predicting Student Graduation Outcomes,” *Proceedings of the World Congress on Engineering and Computer Science 2008*. San Francisco, CA, October 22-24, 2008.

Lesinski, Gene, Steven Corns & Cihan Dagli, “’Application of an Artificial Neural Network to Predict Graduation Success at the United States Military Academy.” *Proceedia Computer Science*, 95 (2016), 375-382.

Musso, Mariel, Eva Kyndt, Eduardo Cascallar & Filip Dochy, “Predicting General Academic Performance and Identifying the Differential Contribution of Participating Variables using Artificial Neural Networks,” *Frontline Learning Research*, 1 (2013), 42-71.

Appendix A:

All of the below scripts are available at: https://github.com/chillington/DSSA_Practicum

Cleaning and rearranging in R:

```
#Cleaning and building data prior to analysis

#R version 3.6.0

#Set working directory, load necessary libraries
#---------------------

#setwd("C:/Users/albertw/Desktop/practicum")

setwd("/home/will/datascience/dssa5302/practicum")

library(tidyverse)
library(readxl)


#First, the course data - all courses for all requested student ID's
#---------------------
#the 4th column needs to be numeric, so set that manually
courses_full <- read_excel("WA_Cohort_Courses.xls", col_types = c("text", "text", "text",
                                                    "numeric", "text", "text", "text"))
#take a look
glimpse(courses_full)
#how many students?
students <- courses_full$Z_NUMBER %>% unique()
#mark rows with courses before term=201580 (fall 2015; new students' first term)
courses_full$returning <- ifelse(courses_full$COURSE_TERM < 201580, 1, 0)
#mark if transfer course by looking for "T" in grade column
courses_full$transfer_course <- ifelse(grepl("T", courses_full$COURSE_GRADE), "yes", "no")
#filter to get only non-transfer courses
stk_courses <- filter(courses_full, transfer_course=="no")
#make sure everything looks good
glimpse(stk_courses)
#filter for courses prior to 201580, indicating returning students
returning_students <- filter(stk_courses, returning==1)
#filter to remove those students
new_students <- filter(stk_courses, !stk_courses$Z_NUMBER %in%
returning_students$Z_NUMBER)
#check it
glimpse(new_students)
#check terms in new dataframe
new_students %>%
  count(COURSE_TERM)
```

```
#filter to get only rows with term=201580 (fall 2015) or 201620 (spring 2016); first full academic year
new_yr1 <- filter(new_students, new_students$COURSE_TERM==201580 |
new_students$COURSE_TERM==201620)
#make sure we got the terms we wanted
new_yr1 %>%
  count(COURSE_TERM)
#list general studies course abbreviations
gens <- c("FRST", "GAH", "GEN", "GIS", "GNM", "GSS")
#mark rows if they are general studies
new_yr1$g_course <- ifelse(new_yr1$COURSE_SUBJ %in% gens, 1, 0)
#check with new variable
glimpse(new_yr1)
#check grade possibilities (and count them)
new_yr1 %>%
  count(COURSE_GRADE)
##consider listing grades to exclude
#exclude_grades <- c("NC", "P", "X")
##consider marking classes with grades to exclude
#new_yr1$exclude_grade <- ifelse(new_yr1$COURSE_GRADE %in% exclude_grades, 1, 0)
##consider recording possible grades to exclude in a new dataframe
#excluded_df <- filter(new_yr1, new_yr1$exclude_grade==1)
#create a dataframe with the total number of courses for each student
tot_courses_data <- new_yr1 %>%
  count(Z_NUMBER) %>%
  rename(tot_courses=n)
#look
glimpse(tot_courses_data)
#get a dataframe with just general studies courses
gens_df <- filter(new_yr1, new_yr1$g_course==1)
#dataframe with number of gens courses for each student
gens_data <- gens_df %>%
  count(Z_NUMBER) %>%
  rename(gens_courses=n)
#dataframe with courses where students voluntarily withdrew
withdrawals_df <- filter(new_yr1, new_yr1$COURSE_GRADE=="W")
#dataframe with number of courses from which each student withdrew
w_data <- withdrawals_df %>%
  count(Z_NUMBER) %>%
  rename(w_courses=n)
#list grades to keep
COURSE_GRADE <- c("A", "A-", "B+", "B", "B-", "C+", "C", "C-", "D+", "D", "D-", "F")
#list number values for letter grades
num_grade <- c(4.0, 3.7, 3.3, 3.0, 2.7, 2.3, 2.0, 1.7, 1.3, 1.0, 0.7, 0)
#make a tibble with letter grades and their corresponding number values
grades <- tibble(COURSE_GRADE, num_grade)
#select only courses with grades to keep
data1 <- filter(new_yr1, new_yr1$COURSE_GRADE %in% grades$COURSE_GRADE)
#add the values by joining
data2 <- right_join(data1, grades)
```

```
#make sure it worked
glimpse(data2)
#make sure we have only the grades we wanted
data2 %>%
  count(COURSE_GRADE)
#get just student id and number grade
data3 <- data2 %>%
  select(Z_NUMBER, num_grade)
#separate dataframes for each term
courses_201580 <- filter(data3, data2$COURSE_TERM==201580)
courses_201620 <- filter(data3, data2$COURSE_TERM==201620)
#get each student's lowest grade for 201580
low_201580 <- courses_201580 %>%
  group_by(Z_NUMBER) %>%        #for each student
  arrange(num_grade) %>%        #sort grades in order
  filter(row_number()==1) %>%   #take the first row
  rename(low_201580=num_grade)   #rename grade column
#and highest for 201580
high_201580 <- courses_201580 %>%
  group_by(Z_NUMBER) %>%
  arrange(num_grade) %>%
  filter(row_number()==n())%>%
  rename(high_201580=num_grade)
#low for 201620
low_201620 <- courses_201620 %>%
  group_by(Z_NUMBER) %>%
  arrange(num_grade) %>%
  filter(row_number()==1)%>%
  rename(low_201620=num_grade)
#high for 201620
high_201620 <- courses_201620 %>%
  group_by(Z_NUMBER) %>%
  arrange(num_grade) %>%
  filter(row_number()==n())%>%
  rename(high_201620=num_grade)
#join each student's highest and lowest grade for each term
highlow_201580 <- full_join(high_201580, low_201580)
highlow_201620 <- full_join(high_201620, low_201620)
#join the terms together so that each row is one student with highest and lowest grade each term
highlow_all <- full_join(highlow_201580, highlow_201620)
#see how it looks
glimpse(highlow_all)
#look at the first 20 rows
highlow_all[1:20,]
#join the number of gens courses for each student
highlow_all_gens <- full_join(highlow_all, gens_data)
#join the number of withdrawals for each student
data4 <- full_join(highlow_all_gens, w_data)
#make sure we have the same students in the total courses dataframe
```

```
tot_courses_data2 <- tot_courses_data %>%
  filter(tot_courses_data$Z_NUMBER %in% data4$Z_NUMBER)
#join the total courses for each student
data5 <- full_join(data4, tot_courses_data2)
#look at the first 20
data5[1:20,]
#replace the NA values for gens and withdrawals with 0
data5 <- data5 %>%
  replace_na(list(gens_courses = 0, w_courses = 0))
#remove the students who did not take courses in either 201580 or 201620
course_data <- data5 %>%
  filter(!is.na(high_201580) & !is.na(high_201620))
#first 20
course_data[1:20,]

#save the data we're going to use
write_csv(course_data, "course_data_to_use.csv")

#clean this place up!
rm(courses_201580, courses_201620, courses_full, data1, data2, data3, data4, data5,
   excluded_df, gens_data,gens_df, grades, high_201580, high_201620, low_201580, low_201620,
   highlow_201580, highlow_201620, highlow_all, highlow_all_gens, new_students, new_yr1,
   returning_students, stk_courses, tot_courses_data, tot_courses_data2, w_data, withdrawals_df,
   COURSE_GRADE, exclude_grades, gens, num_grade)

#in case i need to reopen the course data file:
#-----
course_data <- read.csv("course_data_to_use.csv")

#Next, the file with transfer credits and degree info.
#------------------

student_info <- read_excel("WA_Cohort_Info.xls")
#see what we have
glimpse(student_info)
#get only students in our course_data dataframe
student_data1 <- student_info %>%
  filter(student_info$Z_NUMBER %in% course_data$Z_NUMBER)
#check the first 30
student_data1[1:30,]
#glimpse is better with this many variables (40)
glimpse(student_data1)
#make dataframe with just id and number of transfer credits, one row per student
tcredits <- student_data1 %>%
  select(Z_NUMBER, TRANSFER_CREDITS) %>%       #just id and transfer credits
  replace_na(list(TRANSFER_CREDITS=0)) %>%     #make sure no transfer credits = 0 (not NA)
  distinct()    #ensuring one row for each student
#what's it look like?
glimpse(tcredits)
```

```
#great! join that to the original course_data dataframe
courses_tcredits <- full_join(course_data, tcredits)

##now I need to make 3 variables indicating if a student graduated, left early, or is continuing
#check student status variable
student_data1 %>%
  count(CUR_STU_STATUS_DESC)
#make sure we only have undergraduate records
student_data2 <- filter(student_data1, GRAD_LEVL=="U"| is.na(GRAD_LEVL))
#count grad level
student_data2 %>%
  count(GRAD_LEVL)
#count graduation status
student_data2 %>%
  count(GRAD_STAT_DESC)
#count level in degree works
student_data2 %>%
  count(DW_LEVL)
#get a crosstab for student status by grad status
student_data2 %>%
  group_by(CUR_STU_STATUS_DESC, GRAD_STAT_DESC) %>%
  summarize(n=n()) %>%
  spread(GRAD_STAT_DESC, n)
#crosstab for grad status and degree works level
student_data2 %>%
  group_by(GRAD_STAT_DESC, DW_LEVL) %>%
  summarize(n=n()) %>%
  spread(DW_LEVL, n)
#one more for grad status and degree works percentage (to verify degree completion in any uncertain
cases)
student_data2 %>%
  group_by(GRAD_STAT_DESC, DW_AUDIT_PCT) %>%
  summarize(n=n()) %>%
  spread(DW_AUDIT_PCT, n)
#mark students who graduated based on grad status, degree works level, and degree works percentage
student_data2 <- student_data2 %>%
  mutate(graduated = if_else(GRAD_STAT_DESC=="Awarded" | DW_LEVL=="G" |
DW_AUDIT_PCT=="100", 1,0))
#replace NA's with 0's
student_data2 <- student_data2 %>%
  replace_na(list(graduated = 0))
#mark students who are inactive/left early based on graduated variable and student status
student_data2 <- student_data2 %>%
  mutate(inactive = if_else(graduated==0 & CUR_STU_STATUS_DESC=="Inactive",1,
                            if_else(CUR_STU_STATUS_DESC=="Leave of Absence"|
CUR_STU_STATUS_DESC=="Official Withdrawal", 1,0)))
#replace NA's with 0's
student_data2 <- student_data2 %>%
  replace_na(list(inactive = 0))
```

```
#check crosstab for graduated and inactive to ensure no overlap so far
student_data2 %>%
  group_by(graduated, inactive) %>%
  summarize(n=n()) %>%
  spread(inactive, n)
#mark continuing students based on graduated and inactive
student_data2 <- student_data2 %>%
  mutate(continuing = if_else(graduated==0&inactive==0, 1,0))
#add all three together to make sure no student meets more than one condition
student_data3 <- student_data2 %>%
  mutate(status_check = graduated + inactive + continuing)

#if status check is all 1's, we're good to go!
student_data3 %>%
  count(status_check)
#[thumbs up]
#get a dataframe with id and our 3 new status variables, making sure one row per student
status_data <- student_data3 %>%
  select(Z_NUMBER, graduated, inactive, continuing) %>%
  distinct()

#there appears to be two extra rows... (1302 vs 1300 in course_data)
x <- status_data$Z_NUMBER #get a list of student id's
x[duplicated(x)]          #which ones occur more than once?
status_data %>% filter(Z_NUMBER=="#########")    #let's look at just them, one at a time
status_data %>% filter(Z_NUMBER=="#########")    #(id number intentionally hidden)

#two students must have had a second major but only completed one
#let's remove the rows for their incomplete degrees

status_data <- status_data %>%
  filter(!((Z_NUMBER=="#########"&continuing==1)|
           (Z_NUMBER=="#########"&continuing==1))
         )

#now to join the course data with transfer credits and status variables
full_data <- full_join(courses_tcredits, status_data)
full_data <- ungroup(full_data)
#and add an anonymous id variable
full_data <- full_data %>% mutate(id=rownames(full_data))
#how does it look?
glimpse(full_data)
#one more variable, to combine the 3 separate status variables, for analyses
full_data <- full_data %>%
  mutate(status=if_else(graduated==1, "graduated",
                        if_else(inactive==1, "left early",
                                if_else(continuing==1, "still here", "error"))))
#check for "errors"
full_data %>%
```

```
  count(status)
#make a dataframe just to match the real id's to anonymous id's (just in case something goes terribly
wrong)
id_match <- full_data %>%
  select(Z_NUMBER, id)
#remove real id from main dataframe
full_data_use <- full_data %>%
  select(-Z_NUMBER)
#lookin good?
glimpse(full_data_use)
#save id key dataframe and full data to use separately
write_csv(id_match, "id_match.csv")
write_csv(full_data_use, "full_data_use.csv")
```

Exploration and Analysis in R:

```
#Data analysis phase

#Set working directory, load necessary libraries
#---------------------

#setwd("C:/Users/albertw/Desktop/practicum")

setwd("/home/will/datascience/dssa5302/practicum")

library(tidyverse)

#read the data
#----------
data <- read_csv("full_data_use.csv")

data %>% count(status)

ggplot(data, aes(x=low_201580))+
  geom_histogram()+
  facet_grid(rows=vars(status))

ggplot(data, aes(x=high_201580))+
  geom_histogram()+
  facet_grid(rows=vars(status))

ggplot(data, aes(x=low_201580, y=high_201580)) +
  geom_count()+
  facet_grid(rows=vars(status))+
  labs(x="Lowest Grade", y="Highest Grade", title="Highest by Lowest Grade, Fall 2015")

ggplot(data, aes(x=low_201620, y=high_201620)) +
  geom_count() +
  facet_grid(rows=vars(status))+
```

```
  labs(x="Lowest Grade", y="Highest Grade", title="Highest by Lowest Grade, Spring 2016")

ggplot(data, aes(x=TRANSFER_CREDITS))+
  geom_histogram()+
  facet_grid(rows=vars(status))

ggplot(data, aes(x=gens_courses))+
  geom_histogram()+
  facet_grid(rows=vars(status))

ggplot(data, aes(x=w_courses))+
  geom_histogram()+
  facet_grid(rows=vars(status))

ggplot(data, aes(x=status, y=gens_courses, color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)+
  labs(x="Student Status", y="# GENS Courses (mean)", title="General Studies Courses by Student
Status")

ggplot(data, aes(x=status, y=TRANSFER_CREDITS, color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)+
  labs(x="Student Status", y="# Transfer Credits (mean)", title="Transfer Credits by Student Status")

ggplot(data, aes(x=status, y=w_courses, color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)+
  labs(x="Student Status", y="# Course Withdrawals (mean)", title="Course Withdrawals by Student
Status")

ggplot(data, aes(x=status, y=tot_courses, color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)

ggplot(data, aes(x=status, y=(TRANSFER_CREDITS/96), color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)

ggplot(data, aes(x=status, y=high_201580, color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)

ggplot(data, aes(x=status, y=low_201580, color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)

ggplot(data, aes(x=status, y=high_201620, color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)

ggplot(data, aes(x=status, y=low_201620, color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)

ggplot(data, aes(x=status, y=(gens_courses/tot_courses), color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)
```

```
ggplot(data, aes(x=status, y=(w_courses/tot_courses), color=status))+
  geom_bar(stat="summary", fun.y="mean", width=0.5)
```

Preparing data for neural network in R:

```
#Preparing data for neural network input

#Set working directory, load necessary libraries
#---------------------

#setwd("C:/Users/albertw/Desktop/practicum")

setwd("/home/will/datascience/dssa5302/practicum")

library(tidyverse)

#read the data
#----------
data <- read_csv("full_data_use.csv")

glimpse(data)

data_adj <- data %>%
  mutate(high_fall=(high_201580/4),
         low_fall=(low_201580/4),
         high_spring=(high_201620/4),
         low_spring=(low_201620/4),
         gens_prop=(gens_courses/tot_courses),
         w_prop=(w_courses/tot_courses),
         tcred_prop=(TRANSFER_CREDITS/96))
glimpse(data_adj)

data_for_nn <- data_adj %>%
  select(id, high_fall, low_fall, high_spring, low_spring, w_prop, tcred_prop,
         inactive, continuing, graduated)
#made the decision to get rid of gens_prop; only use 6 variables
glimpse(data_for_nn)
data_for_nn[1:20,]

write_csv(data_for_nn, "data_for_nn.csv")
```

Neural network in Python:

```
#Building the neural network

# Import necessary libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

```
from keras.constraints import maxnorm
from keras.callbacks import ModelCheckpoint
from keras.models import model_from_json
from keras import optimizers
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# set random seed for reproducibility
np.random.seed(53)

# load dataset
dataset = np.loadtxt("data_for_nn.csv", delimiter=",", skiprows=1)

# split into input (X) and output (Y) variables
X = dataset[:,1:7] #input features
Y = dataset[:,7:10] #output targets
# split again into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,shuffle=False)

print('begin...')

# create sequential model
model1 = Sequential()
#add a hidden layer with 12 neurons that takes 6 input values
#and uses a RELU activation function
model1.add(Dense(18,input_dim=6,kernel_initializer='normal',activation='relu'))
#add second hidden layer with 9 neruons using RELU function
model1.add(Dense(9,kernel_initializer='normal',activation='relu'))
#output layer with 3 neurons
model1.add(Dense(3,kernel_initializer='normal',activation='sigmoid'))

# Compile model
#set learning rate
learning_rate=0.01
#define optimizer using "adam" with set learning rate
adam = optimizers.Adam(lr=learning_rate)

print('compile...')

#compile using Binary Cross-entropy loss function and adam optimizer
#output accuracy metric
model1.compile(loss='binary_crossentropy',optimizer=adam,metrics=['accuracy'])

# Save model to disk ############
# Save model structure as json
print('==================')
print('Saving model to disk')
```

```
print('==================')
model_json1 = model1.to_json()
with open('model.json', 'w') as json_file:
    json_file.write(model_json1)
#################################

# Set up checkpointing

filepath = 'weights.best.hdh5'
checkpoint =
ModelCheckpoint(filepath,monitor='val_acc',verbose=0,save_best_only=True,mode='max')
callbacks_list = [checkpoint]

print('fit...')

history1=model1.fit(X_train,Y_train,validation_split=0.333,epochs=5000,verbose=0,callbacks=callbac
ks_list)

# Evaluate the model
scores1 = model1.evaluate(X_train, Y_train)
Y_predict1 = model1.predict(X_test)

print("\n%s: %.2f%%" % (model1.metrics_names[1], scores1[1]*100))

# leave early
# create confusion matrix details
inactive=Y_test[:,0]
inactive_pred=Y_predict1[:,0]
rounded = [round(i) for i in inactive_pred]
y_pred = np.array(rounded,dtype='int64')
print('Confusion Matrix')
print('================')
CM = confusion_matrix(inactive, y_pred)
print('True negatives: ',CM[0,0])
print('False negatives: ',CM[1,0])
print('False positives: ',CM[0,1])
print('True positives: ',CM[1,1])

#plot predictions for "left early"
#reformat predicted values to a list, subtract from
#observed Y, and calculate squared difference to
#show accuracy of each prediction
y1 = [i for i in inactive_pred]
diff=inactive-y1
sqdiff = [i**2 for i in diff]
#combine squared difference, predicted Y, and observed Y
#to a single array to plot more efficiently
sqdiff = np.asarray(sqdiff)
y1 = np.asarray(y1)
```

```
data1 = np.vstack((inactive,y1,sqdiff))

#create scatter plots for observed vs predicted values,
#with squared differences determining color and marker size

#set figure size
plt.figure(figsize=(10,3))
#scatterplot for observed vs predicted values
#with color defined by squared difference
#and size defined by squared difference subtracted from 1
#so that the more accurate markers are larger
plt.scatter(data1[1], data1[0], c = data1[2], s=((1-data1[2])*1000), marker="|")
plt.title("Observed vs Predicted Values - Left Early",fontsize=16)
plt.ylabel("Observed Value",fontsize=14)
plt.xlabel("Predicted Value",fontsize=14)
#add text to display current learning rate
plt.text(0.2,0.5,'Epochs = 5000; Learning Rate = %s' % learning_rate, fontsize=14)
#set the color palette
plt.set_cmap('jet_r') # "_r to reverse the order"
#add color bar
cbar=plt.colorbar()
#label the color bar definition of our new accuracy calculation
cbar.set_label('Squared Error ((Y - Y predicted)**2)')
#set y axis to only show 0 and 1 for observed values
plt.yticks(range(0,2,1), fontsize=14)
plt.xticks(fontsize=14)
#add axes to remove top and right
lines = plt.axes()
lines.spines["top"].set_visible(False)
lines.spines["right"].set_visible(False)
plt.sca(lines) #adds axes to current figure

plt.show()

# graduate
# create confusion matrix details
graduated=Y_test[:,2]
grad_pred=Y_predict1[:,2]
rounded = [round(i) for i in grad_pred]
y_pred = np.array(rounded,dtype='int64')
print('Confusion Matrix')
print('================')
CM = confusion_matrix(graduated, y_pred)
print('True negatives: ',CM[0,0])
print('False negatives: ',CM[1,0])
print('False positives: ',CM[0,1])
print('True positives: ',CM[1,1])

#plot predictions for "graduated"
```

```
#reformat predicted values to a list, subtract from
#observed Y, and calculate squared difference to
#show accuracy of each prediction
y1 = [i for i in grad_pred]
diff=graduated-y1
sqdiff = [i**2 for i in diff]
#combine squared difference, predicted Y, and observed Y
#to a single array to plot more efficiently
sqdiff = np.asarray(sqdiff)
y1 = np.asarray(y1)
data1 = np.vstack((graduated,y1,sqdiff))

#create scatter plots for observed vs predicted values,
#with squared differences determining color and marker size

#set figure size
plt.figure(figsize=(10,3))
#scatterplot for observed vs predicted values
#with color defined by squared difference
#and size defined by squared difference subtracted from 1
#so that the more accurate markers are larger
plt.scatter(data1[1], data1[0], c = data1[2], s=((1-data1[2])*1000), marker="|")
plt.title("Observed vs Predicted Values - Graduated",fontsize=16)
plt.ylabel("Observed Value",fontsize=14)
plt.xlabel("Predicted Value",fontsize=14)
#add text to display current learning rate
plt.text(0.2,0.5,'Epochs = 5000; Learning Rate = %s' % learning_rate, fontsize=14)
#set the color palette
plt.set_cmap('jet_r') # "_r to reverse the order"
#add color bar
cbar=plt.colorbar()
#label the color bar definition of our new accuracy calculation
cbar.set_label('Squared Error ((Y - Y predicted)**2)')
#set y axis to only show 0 and 1 for observed values
plt.yticks(range(0,2,1), fontsize=14)
plt.xticks(fontsize=14)
#add axes to remove top and right
lines = plt.axes()
lines.spines["top"].set_visible(False)
lines.spines["right"].set_visible(False)
plt.sca(lines) #adds axes to current figure

plt.show()
```

Appendix B:

This report can be accessed online at my Gihub blog: https://chillington.github.io/Practicum