Laterite Demonstration

mutwiri_ian@yahoo.com

16 March,2022

This is a demonstrative project I have worked to demonstrate my data analysis, machine learning and R skills. I use data obtained from the UCL machine learning archive(https://archive-beta.ics.uci.edu/ml/datasets/predict+students+dropout+and+academic+success) on student dropout rates and academic success in Portugal. The target variable in this case is whether the student finally enrolled, dropped out or graduated.

I extensively leverage the `Tidyverse` ecosystem for data importing, wrangling and visualization and the `Tidymodels` set of package to develop a machine learning pipeline. Lets get into it by first loading the `Tidyverse` package and reading in the data.

```r
library(tidyverse)
#Data is in working directory and semi-colon delimited
students <- read_delim(file = 'students.csv',delim = ";")
```

Now we can begin analyzing the data by doing some basic exploratory visualisations that can help us 'see' the data and draw insights. since the target variable is discrete, we can examine how the data is distributed and check for imbalance in some categories.

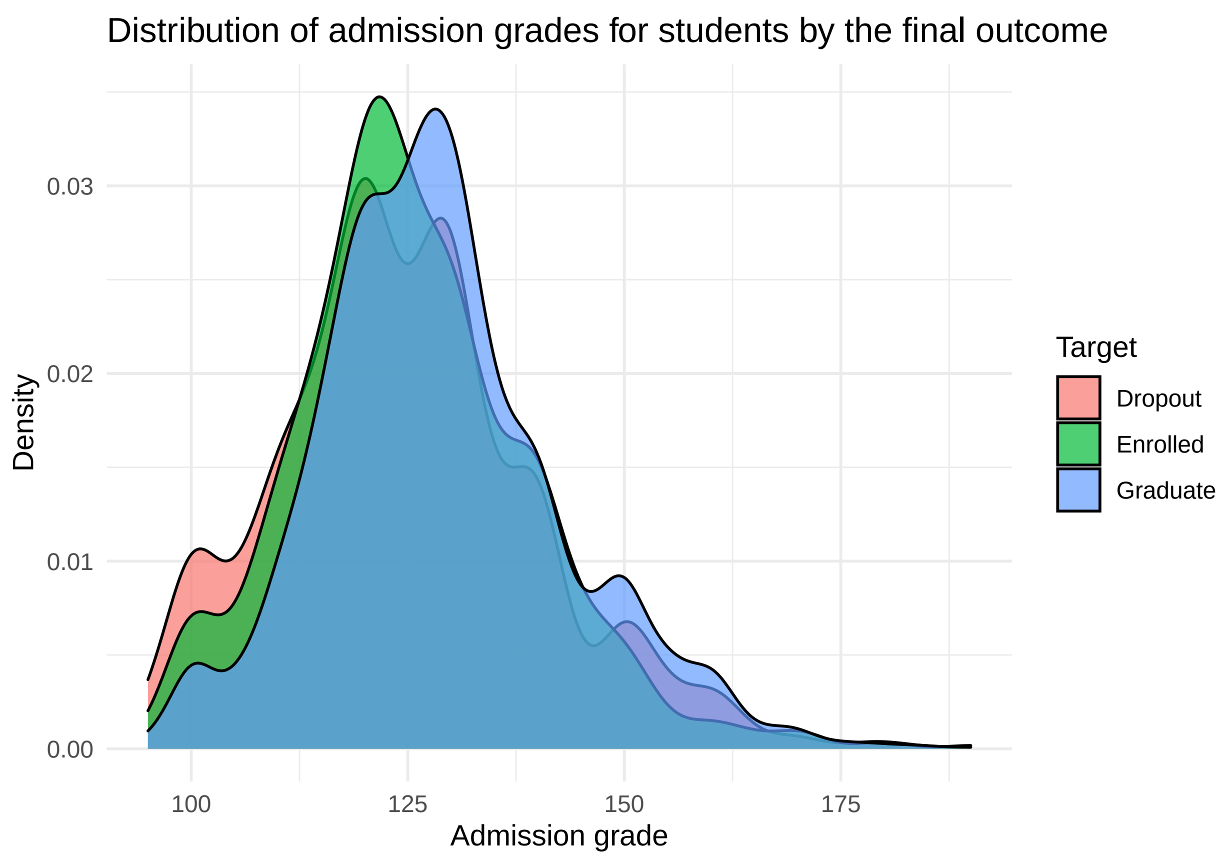
There seems to be some severe imbalance in the data particularly towards the `Graduate` category.

```r
students %>%
  count(Target) %>%
  kableExtra::kable()
```

Target	n
Dropout	1421
Enrolled	794
Graduate	2209

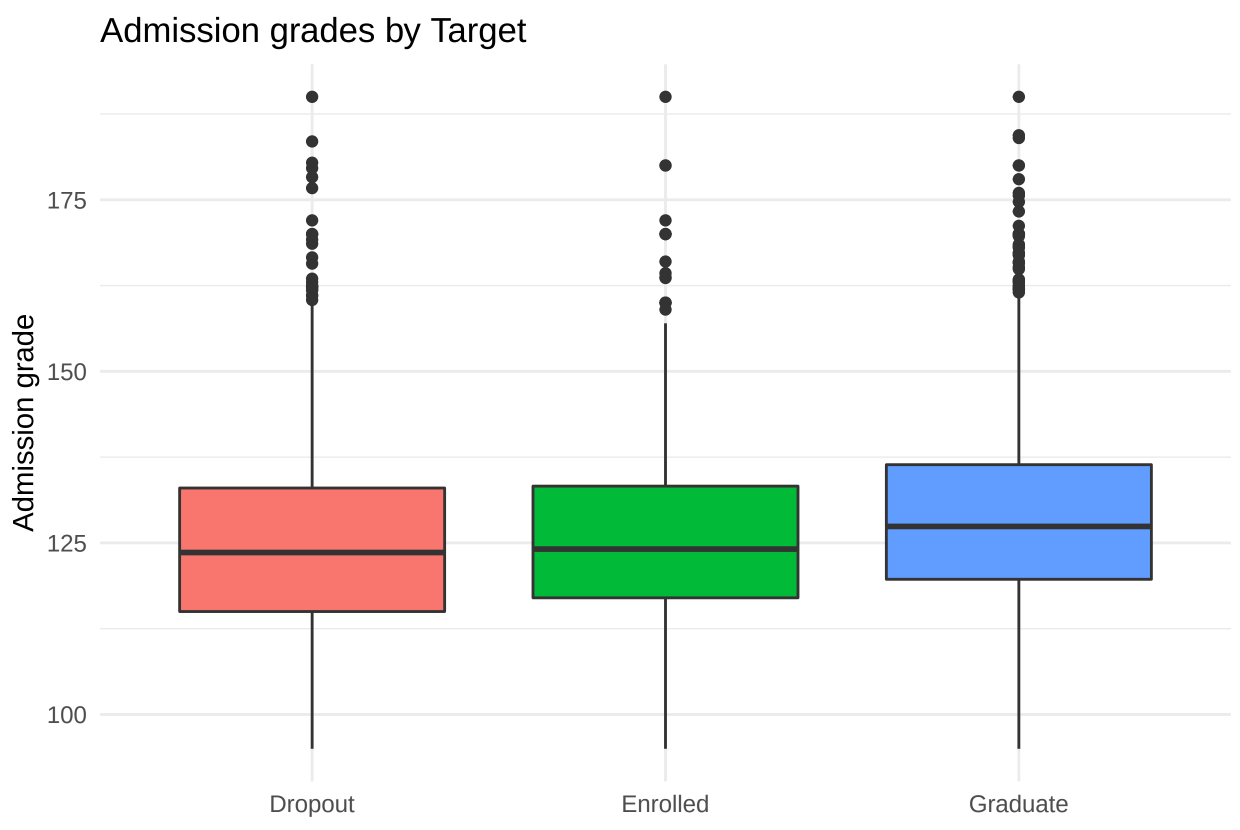
By inspection of the density plots, it appears that the data is fairly normally distributed across the `Target` levels which is good for modelling.

```r
students %>%
  ggplot(aes(`Admission grade`))+
  geom_density(aes(fill=Target),alpha=.7)+
  labs(
    title = 'Distribution of admission grades for students by the final outcome',
    y='Density'

  )
```



Visual inspection implies there is difference betweeen the mean and the variance of admission grades across all `Target` categories which is confirmed by performing a formal satistical tests.

```
students %>%
  ggplot(aes(Target,`Admission grade`))+
  geom_boxplot(aes(fill=Target),show.legend = F)+
  labs(
    title = 'Admission grades by Target'
  )+
  xlab('')
```



```
summary(aov(`Admission grade`~Target,data = students))
```

```
##                Df Sum Sq Mean Sq F value   Pr(>F)
## Target          2  14722    7361   35.65 4.38e-16 ***
## Residuals    4421 912906     206
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
bartlett.test(`Admission grade`~Target,data = students)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  Admission grade by Target
## Bartlett's K-squared = 12.222, df = 2, p-value = 0.002218
```

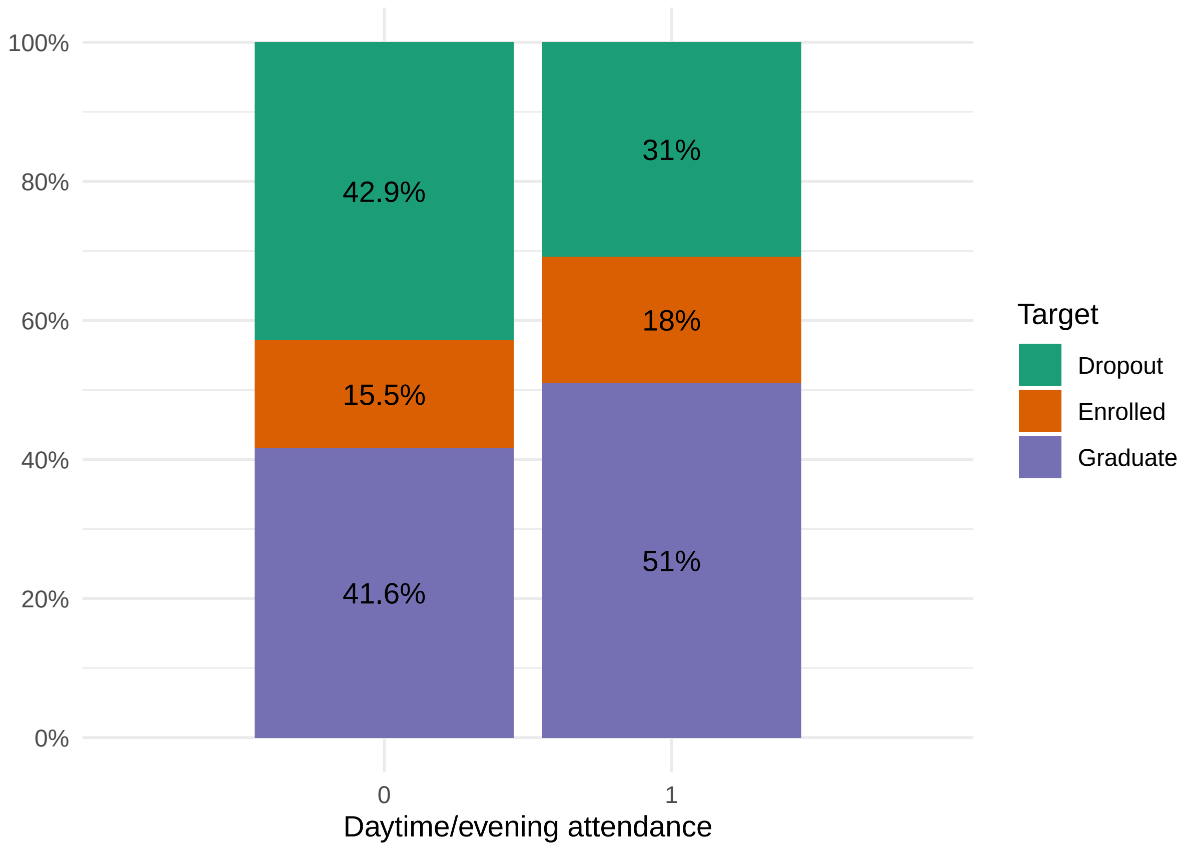
Fathers' qualification are arguably higher for all `Target` categories compared to mother's qualification.

```
students %>%
  select(Target,`Mother's qualification`,`Father's qualification`) %>%
  rename(mom_qual=`Mother's qualification`,
         dad_qual=`Father's qualification`) %>%
  group_by(Target) %>%
  summarise(Mean_Mothers_qual=mean(mom_qual),
            Mean_Fathers_qual=mean(dad_qual)) %>%
  kableExtra::kable()
```

Target	Mean_Mothers_qual	Mean_Fathers_qual
Dropout	21.03519	22.68332
Enrolled	17.62217	20.92065
Graduate	19.31145	22.49977

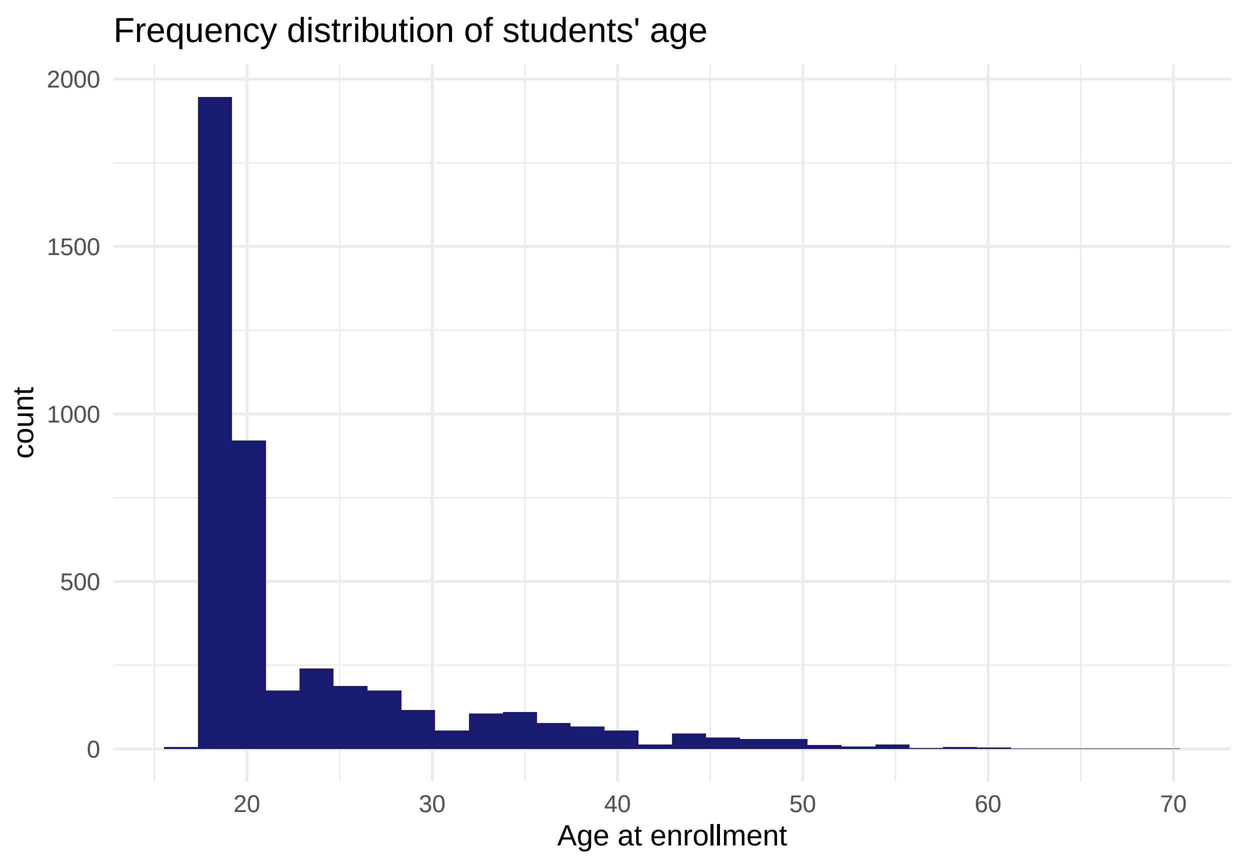
Exploring the difference between students who attend regular and evening classes shows that most students taking evening classes end up graduating as compared to to students attending day-time classes. Moreover,students attending daytime classes are more likely to dropout.

```
plotdata<- students %>%
    mutate(day_even_att=factor(`Daytime/evening attendance  `),
           Target=factor(Target)) %>%
  select(`Daytime/evening attendance    `,Target)%>%
  group_by(`Daytime/evening attendance  `,Target) %>%
  summarise(n=n()) %>%
  mutate(pct=n/sum(n),lbl=scales::percent(pct))

plotdata %>%
  ggplot(aes(`Daytime/evening attendance    `,pct,fill=Target))+
  geom_bar(stat = 'identity',position = 'fill')+
    scale_x_discrete(limits=c(0,1))+
  scale_y_continuous(breaks  = seq(0,1,.2),labels = scales::percent)+
  geom_text(aes(label=lbl),
            position = position_stack(vjust = .5))+
  scale_fill_brewer(palette = 'Dark2')+
  ylab('')
```



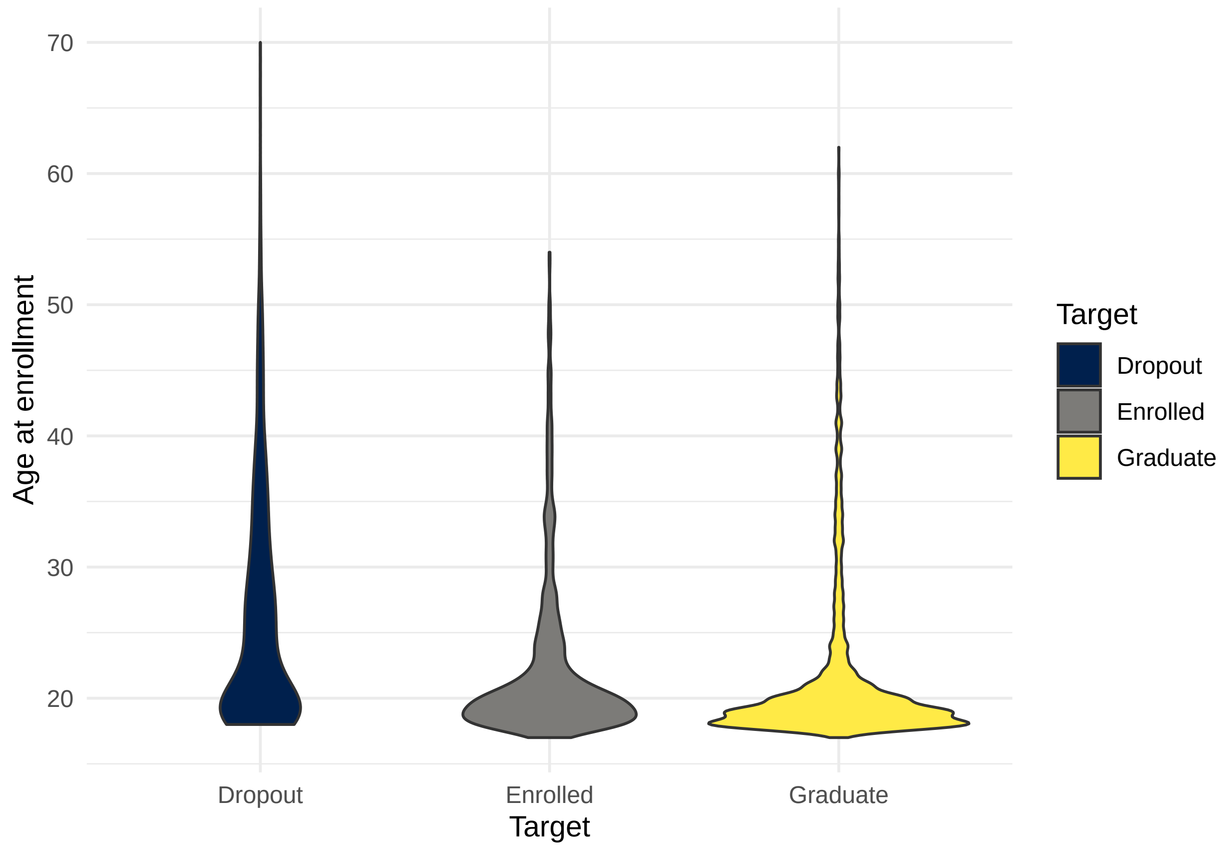
Most students enrolled are in their early 20's followed by those in their late 20's.

```r
students %>%
ggplot(aes(`Age at enrollment`))+
geom_histogram(fill='Midnightblue')+
  labs(
    title = "Frequency distribution of students' age"
  )
```



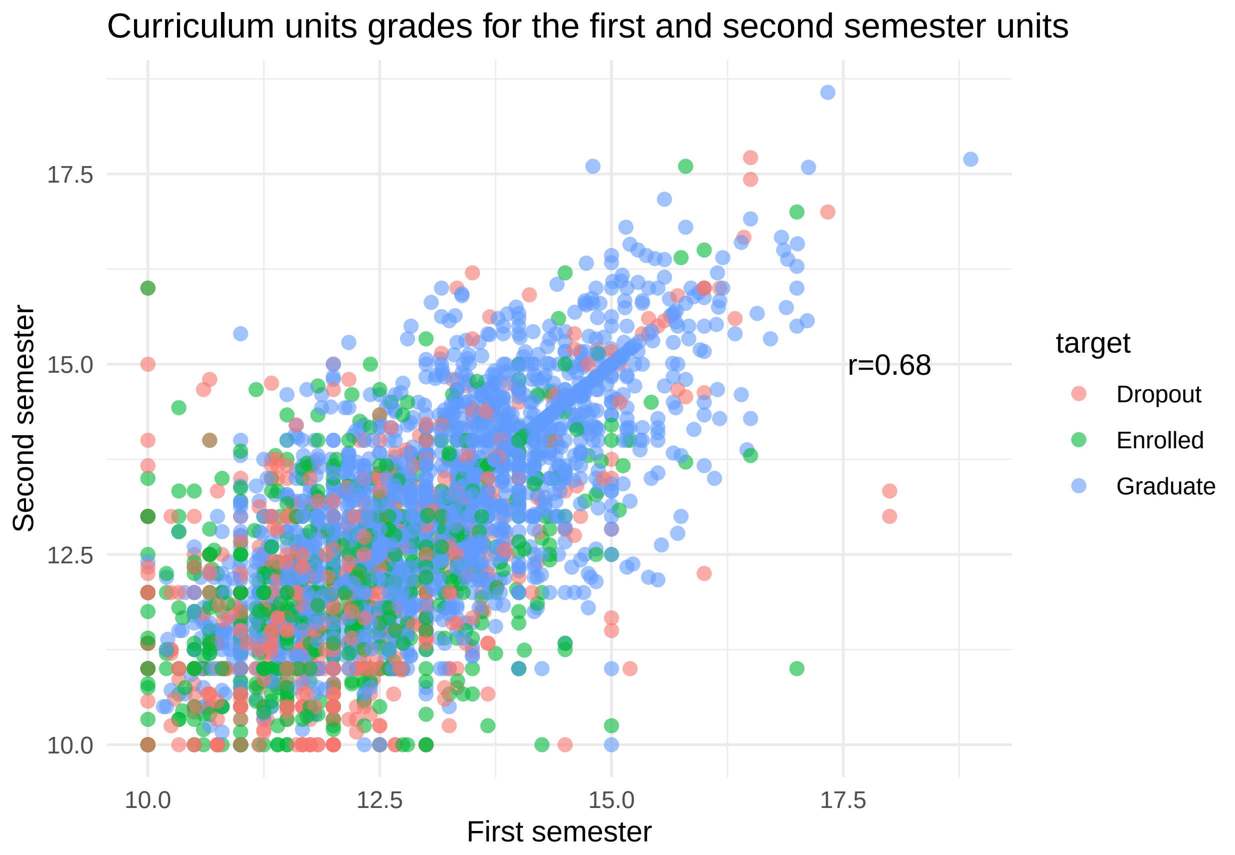
This is confirmed across the `Target` variable with most dispersion in the those that end up graduating.

```
students %>%
  ggplot() +
  geom_violin(aes(Target,`Age at enrollment`,fill=Target))+
  scale_fill_viridis_d(option = 'cividis')
```



The first semester and second semester correlation is relatively high and statistically significant. Of course correlation does not imply causation.

```
grades<- students %>%
  select(starts_with('curricular')&ends_with('(grade)'),Target) %>%
  janitor::clean_names() %>%
  rename(first_sem=curricular_units_1st_sem_grade,
         second_sem=curricular_units_2nd_sem_grade) %>%
  filter(first_sem>0 & second_sem>0)
r <- cor(grades$first_sem,grades$second_sem)
grades %>%
  ggplot(aes(first_sem,second_sem))+
  geom_point(aes(col=target),size=2,alpha=.6)+
  annotate('text',x=18,y=15,label=paste0('r=',round(r,2)))+
  labs(title = 'Curriculum units grades for the first and second semester units',
       x='First semester',y='Second semester')
```



We could also explore day time and evening class attendance across both genders. In both genders, the average age is higher for students who attend evening classes compared to those who attend daytime classes.

```
students %>%
  select(Gender,`Daytime/evening attendance `,`Age at enrollment`) %>%
  mutate_at(vars(1,2),factor) %>%
  janitor::clean_names() %>%
  group_by(gender,daytime_evening_attendance)%>%
  summarise(avg_age=mean(age_at_enrollment)) %>%
  kableExtra::kable()
```

gender	daytime_evening_attendance	avg_age
0	0	32.46557
0	1	21.23956
1	0	34.68539
1	1	23.52105

Now we can get on building a machine learning model(s) that we will use to predict the `Target` class. Both `caret` and `Tidymodels` work just fine but I will use the later for now. A machine learning model is meant to predict as accurately as possible new data after learning from the training data. To this effect, we split the full data into a training set, which we will use to train our model(s), and a testing set, on which we evaluate the performance of our model.

I choose to split the data on a 80/20 basis and stratifying on the `Target` variable so as to ensure that the distribution of categories is similar in both the training and testing data.

```r
library(tidymodels)
students <- students %>%
  mutate_if(is.character,factor) %>%
  mutate_at(vars(c(1,3,5,8,14:19,21)),factor) %>%
  janitor::clean_names()

set.seed(7577)
students_split <- initial_split(students,prop = .8,strata = target)
students_train <- training(students_split)
students_test <- testing(students_split)
```

Because the performance of the model can only be evaluated with the testing data,until then we need to figure out a way of assessing the performance of our model in the training process. We can do this by cross-validation;we randomly sample observations in the training data that we will use to build the model and use the remaining observations to assess the performance of the model. We can do this repeatedly,5 or 10 times, and average these results to get an estimate how our model would perform in production.

```r
students_folds <- vfold_cv(students_train,v = 10,strata = target)
students_folds
```

```
## #  10-fold cross-validation using stratification
## # A tibble: 10 x 2
##    splits             id
##    <list>             <chr>
##  1 <split [3183/355]> Fold01
##  2 <split [3183/355]> Fold02
##  3 <split [3183/355]> Fold03
##  4 <split [3183/355]> Fold04
##  5 <split [3183/355]> Fold05
##  6 <split [3184/354]> Fold06
##  7 <split [3185/353]> Fold07
##  8 <split [3186/352]> Fold08
##  9 <split [3186/352]> Fold09
## 10 <split [3186/352]> Fold10
```

To prepare the data for modelling, the `recipes` package comes in handy by providing some pre-processing steps like dummy-encoding of categorical variables and removal of redundant variables with the `step_zv()` function.

```r
students_rec<- recipe(target~.,data = students_train) %>%
  themis::step_downsample(target) %>%
  step_dummy(all_nominal(),-all_outcomes()) %>%
  step_zv(all_numeric()) %>%
  step_normalize(all_numeric()) %>%
  prep()
```

We then specify the models that we are will attempt to fit to the data and select the best among these. I choose to work with a multinomial regression model, a random forest and a support vector classifier.

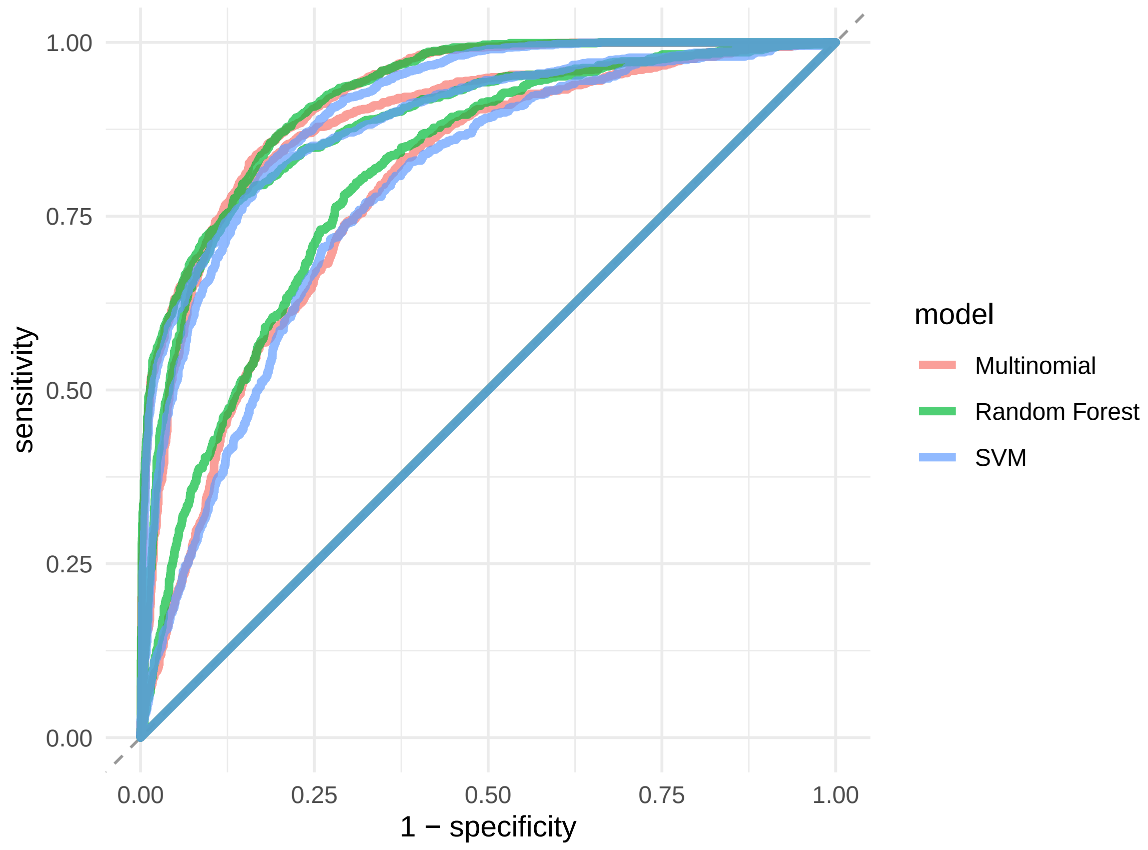
```r
multi_spec <- multinom_reg() %>%
  set_engine('nnet') %>%
  set_mode('classification')
```

```
rf_spec <- rand_forest(trees = 1000) %>%
  set_engine('ranger',importance = "impurity") %>%
    set_mode('classification')

svm_spec <- svm_rbf() %>%
  set_engine('kernlab') %>%
    set_mode('classification')
```

Using the recipe and the cross-validation folds that were previously created, we can then fit all these model specifications to the folds/resamples and then evaluate their performance.

```
multi_rs<- fit_resamples(
  multi_spec,
  students_rec,
  students_folds,
  control=control_resamples(save_pred = T)
)

rf_rs<- fit_resamples(
  rf_spec,
  students_rec,
  students_folds,
  control=control_resamples(save_pred = T)
)

svm_rs<- fit_resamples(
  svm_spec,
  students_rec,
  students_folds,
  control=control_resamples(save_pred = T)
)
```

```
multi_rs %>%
  mutate(model='Multinomial') %>%
  bind_rows(rf_rs %>%
              mutate(model='Random Forest')) %>%
  bind_rows(svm_rs %>% mutate(model="SVM")) %>%
  unnest(.predictions) %>%
  group_by(model) %>%
  roc_curve(target,.pred_Dropout:.pred_Graduate) %>%
  ggplot(aes(1-specificity,sensitivity,color=model))+
  geom_abline(slope = 1,lty=2,color='gray50',alpha=.8)+
  geom_path(size=1.5,alpha=.7)+
  labs(model=NULL)+
  coord_fixed()
```



The performance of the three models is not so different(84% overall accuracy) and I would choose to work with the logistic regression model as it is less complex and easier to interpret compared to the random forest and the support vector classifier which are like black boxes.

```
multi_rs %>%
  collect_metrics() %>%
  mutate(model='logistic') %>%
  bind_rows(
    rf_rs %>%
      collect_metrics() %>%
      mutate(model='rf')
  ) %>%
  bind_rows(
    svm_rs %>%
      collect_metrics() %>%
      mutate(model='svm')
  ) %>%
  select(model,.metric,mean)
```

```
## # A tibble: 6 x 3
##   model    .metric   mean
##   <chr>    <chr>    <dbl>
## 1 logistic accuracy 0.711
## 2 logistic roc_auc  0.850
## 3 rf       accuracy 0.706
## 4 rf       roc_auc  0.856
## 5 svm      accuracy 0.692
## 6 svm      roc_auc  0.843
```

However,these models have model-specific parameters(hyperparameters) that can be tweaked so as to match the structure of the data and improve accuracy. With random forests we can tune the number of trees to construct, the number of variables to split on and the minimum number of elements that each node should contain while the support vector classifier with a radial basis kernel function has a cost-complexity parameter which controls for the amount of overlap allowed by the classifier. We will update the model specifications so that we are able to tune these parameters.

We can combine the the recipe and the model specificatons into a workflow and the tune these hyperparameters. But first we need to update these model specifications so that these hyperparameters are tunable.

```
multi_spec <- multi_spec %>%
  update(mixture=tune())

rf_spec <- rf_spec %>%
  update(mtry=tune(),min_n=tune())

svm_spec <- svm_spec %>%
  update(cost=tune())

students_set<- workflow_set(
  list(students_rec),
  list(Multinomial=multi_spec,Random_forest=rf_spec,SVM=svm_spec),
  cross = F
)
```

These models are computationally intensive so we register a parallel processing back-end to speed up the process

```
cl <- parallel::makePSOCKcluster(2)
doParallel::registerDoParallel(cl)
```

I set the number of parameters to tune across as 5 which are selected by a space-fill design. The models with these combinations of hyperparameters are trained across the 10 cross-validation folds.

```
#Use a space-fill design to select parameter values
students_rs<- workflow_map(
  students_set,
  "tune_grid",
  resamples = students_folds,
  grid=5,
  seed = 7578,verbose = T,
  control = control_resamples(save_pred = T)
)
```

We can also visualize the performance profile of the models across the tuning parameters.

```
students_rs %>%
  collect_metrics()
```

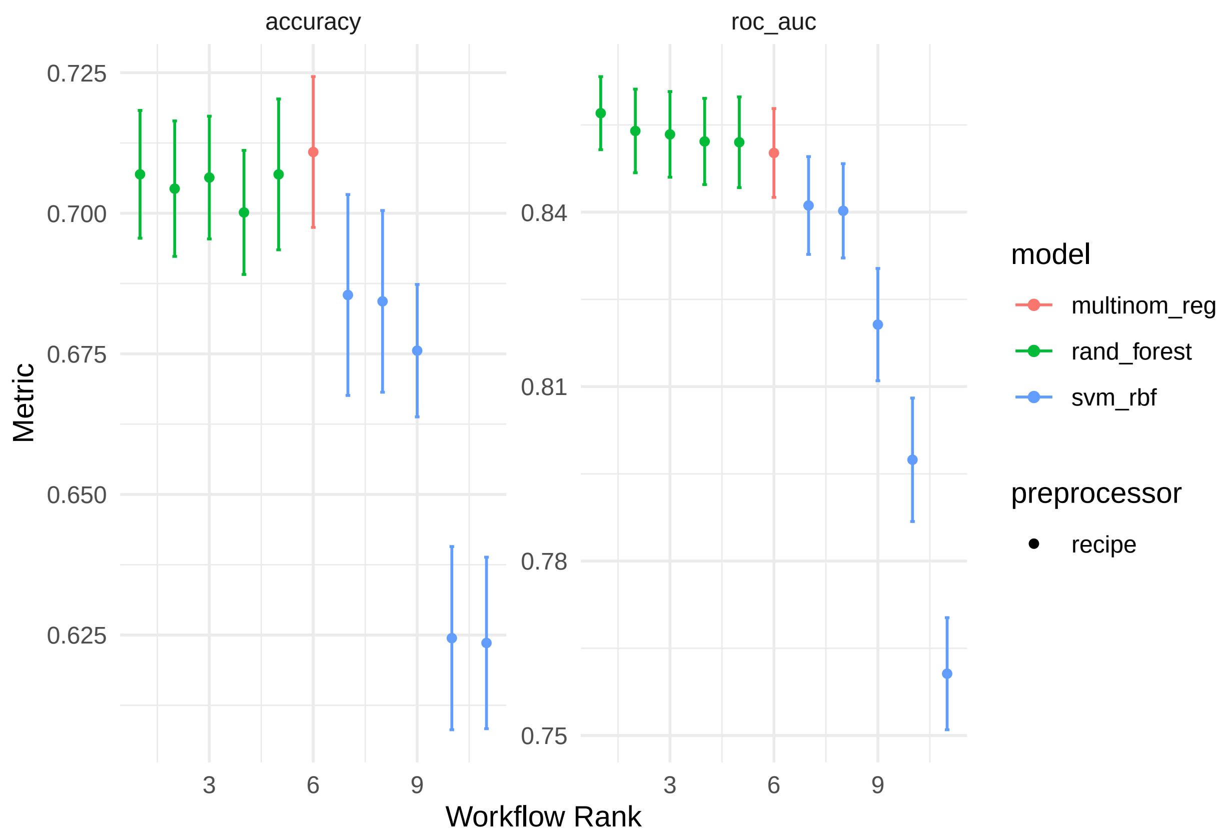
```
## # A tibble: 22 x 9
##    wflow_id          .config preproc model .metric .estimator  mean     n std_err
##    <chr>             <chr>   <chr>   <chr> <chr>   <chr>      <dbl> <int>   <dbl>
##  1 recipe_Multinom~ Prepro~ recipe  mult~ accura~ multiclass 0.711    10 0.00815
##  2 recipe_Multinom~ Prepro~ recipe  mult~ roc_auc hand_till  0.850    10 0.00464
##  3 recipe_Random_f~ Prepro~ recipe  rand~ accura~ multiclass 0.706    10 0.00663
##  4 recipe_Random_f~ Prepro~ recipe  rand~ roc_auc hand_till  0.853    10 0.00448
##  5 recipe_Random_f~ Prepro~ recipe  rand~ accura~ multiclass 0.704    10 0.00732
##  6 recipe_Random_f~ Prepro~ recipe  rand~ roc_auc hand_till  0.854    10 0.00437
##  7 recipe_Random_f~ Prepro~ recipe  rand~ accura~ multiclass 0.700    10 0.00671
##  8 recipe_Random_f~ Prepro~ recipe  rand~ roc_auc hand_till  0.852    10 0.00451
##  9 recipe_Random_f~ Prepro~ recipe  rand~ accura~ multiclass 0.707    10 0.00691
## 10 recipe_Random_f~ Prepro~ recipe  rand~ roc_auc hand_till  0.857    10 0.00382
## # ... with 12 more rows
```

We can also check for the best model by ranking the models by the overall accuracy rate, by the area under the ROC curve or by a quick visualization.

```
students_rs %>%
  rank_results(rank_metric = 'roc_auc')
```

```
## # A tibble: 22 x 9
##    wflow_id          .config .metric  mean std_err     n preprocessor model  rank
##    <chr>             <chr>   <chr>   <dbl>   <dbl> <int> <chr>        <chr> <int>
##  1 recipe_Random_f~ Prepro~ accura~ 0.707 0.00691    10 recipe       rand~     1
##  2 recipe_Random_f~ Prepro~ roc_auc 0.857 0.00382    10 recipe       rand~     1
##  3 recipe_Random_f~ Prepro~ accura~ 0.704 0.00732    10 recipe       rand~     2
##  4 recipe_Random_f~ Prepro~ roc_auc 0.854 0.00437    10 recipe       rand~     2
##  5 recipe_Random_f~ Prepro~ accura~ 0.706 0.00663    10 recipe       rand~     3
##  6 recipe_Random_f~ Prepro~ roc_auc 0.853 0.00448    10 recipe       rand~     3
##  7 recipe_Random_f~ Prepro~ accura~ 0.700 0.00671    10 recipe       rand~     4
##  8 recipe_Random_f~ Prepro~ roc_auc 0.852 0.00451    10 recipe       rand~     4
##  9 recipe_Random_f~ Prepro~ accura~ 0.707 0.00815    10 recipe       rand~     5
## 10 recipe_Random_f~ Prepro~ roc_auc 0.852 0.00474    10 recipe       rand~     5
## # ... with 12 more rows
```

```
students_rs %>%
  autoplot()
```



We see that the model in the workflow corresponding to a random forest is the best among all other models evaluated. We can extract this workflow together with the best model and use it to make predictions for new data. To do this we fit the best model to the training data and then evaluate its performance on the testing data. We can then get an idea of how accurately this model predicts on new data.

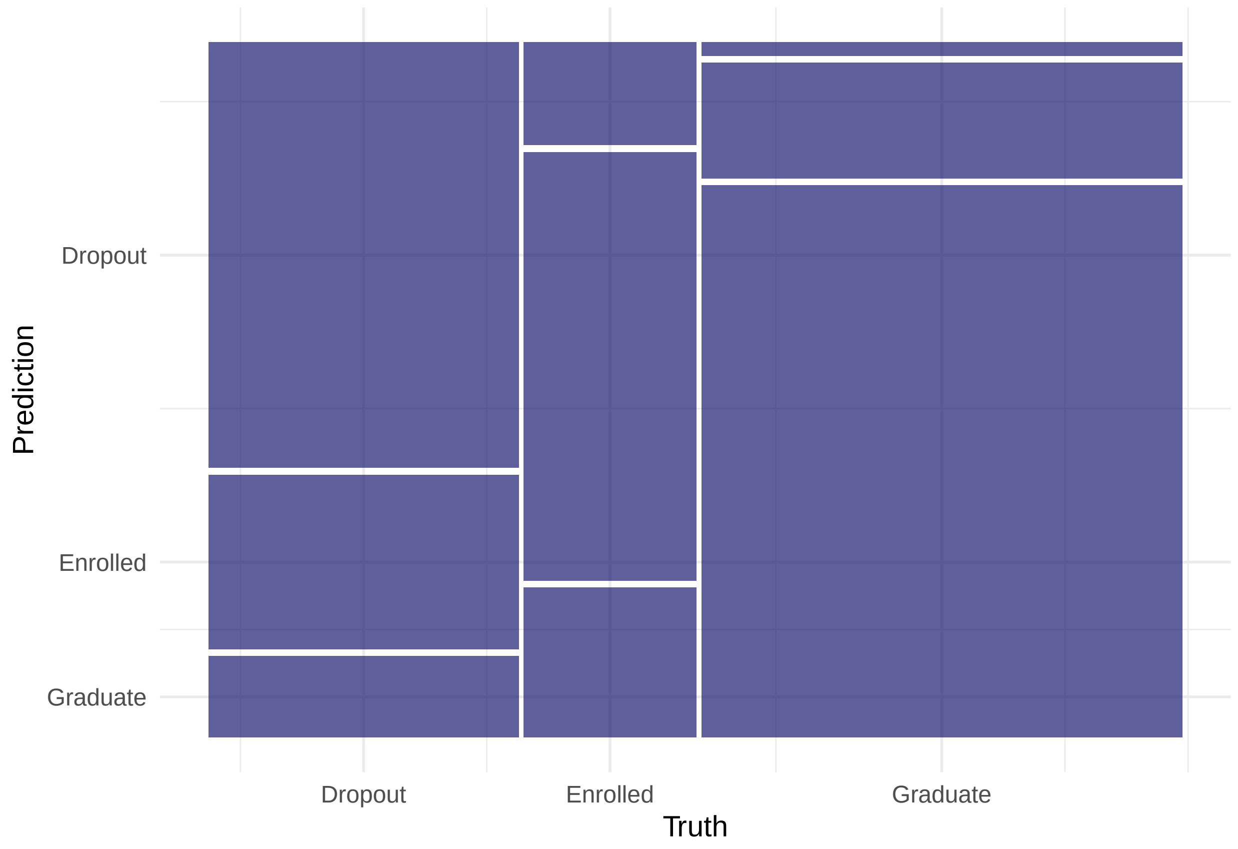
```
best_model<- students_rs %>%
  extract_workflow_set_result("recipe_Random_forest") %>%
  select_best(metric = 'roc_auc')

best_fit<- students_rs %>%
  extract_workflow("recipe_Random_forest") %>%
  finalize_workflow(best_model) %>%
  last_fit(students_split)

best_fit %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.718 Preprocessor1_Model1
## 2 roc_auc  hand_till      0.834 Preprocessor1_Model1
```

It performs worse but does not severely deviate from the expected performance. There is some over-fitting but the can be remedied with more computational power and expert domain knowledge.

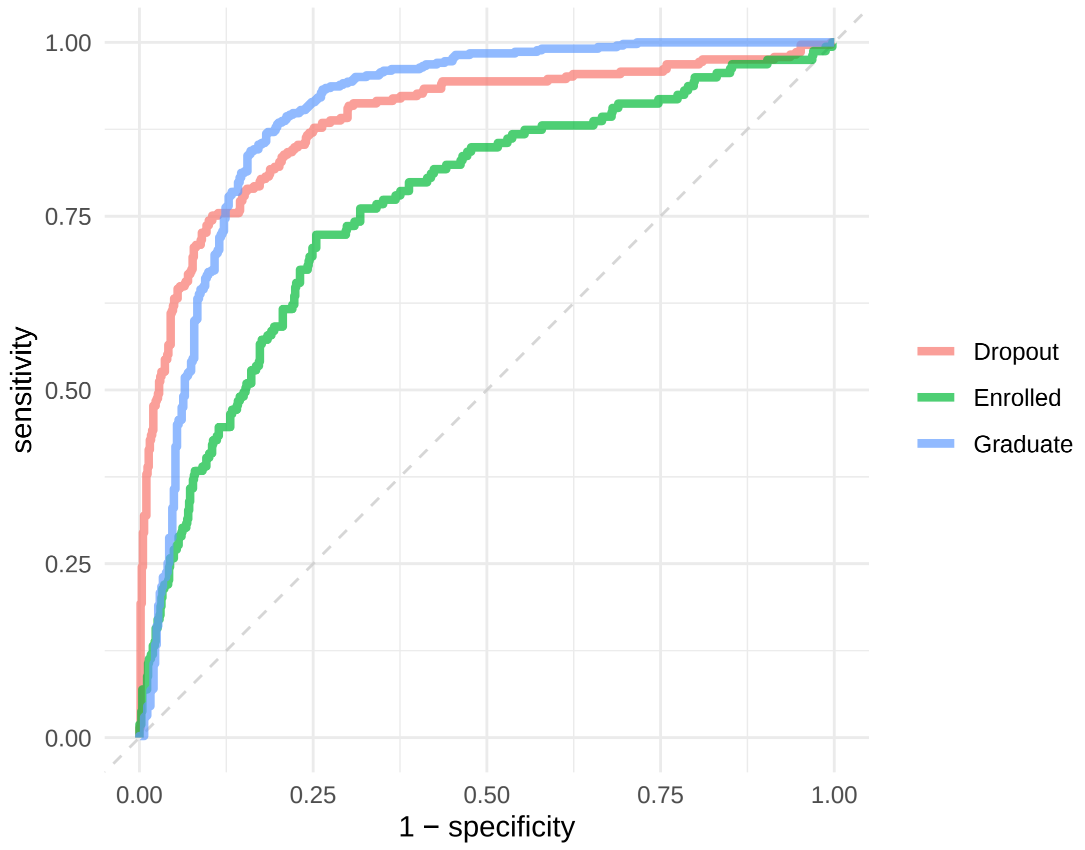
We can also visualize a confusion matrix for the class prediction

```
best_fit %>%
  collect_predictions() %>%
  conf_mat(target,.pred_class) %>%
  autoplot()
```



and ROC curves for each class.

```
best_fit %>%
  collect_predictions() %>%
  roc_curve(truth = target,.pred_Dropout:.pred_Graduate) %>%
  ggplot(aes(1-specificity,sensitivity,color=.level))+
  geom_abline(slope = 1,color='gray80',lty=2,alpha=.8)+
  geom_path(size=1.5,alpha=.7)+
  labs(color=NULL)+
  coord_fixed()
```



Our model predicts the Graduate class better than any other class.

We can randomly select an observation from our data and use our model to make a prediction for the class as follows

```
best_fit %>%
  extract_workflow() %>%
  predict(new_data = students %>% slice_sample())
```

```
## # A tibble: 1 x 1
##   .pred_class
##   <fct>
## 1 Enrolled
```