Student Performance: Predicting Final Pass/Fail (UCI Student Performance)

Mayesha Maliha Proma

2025-10-26

Executive Summary

The objective of this project is to predict whether a student will pass Mathematics, defined as attaining a final grade of at least 10 out of 20, by utilizing demographic, family, engagement, and prior-performance features. The analysis draws on the student-mat.csv dataset, which includes data on 395 students across 33 variables. The primary target, pass_flag, is derived from the final grade and distinguishes between passing and not passing, with approximately 67 percent of students passing and 33 percent not passing. Rigorous data cleaning and exploration were conducted, and the final grade variable was excluded from the predictors to prevent data leakage. The data was split into training and test sets in an 80 to 20 ratio with stratified sampling to preserve class balance. Random Forest, implemented with the ranger package, and XGBoost were both tuned through five-fold cross-validation and space-filling hyperparameter grids. The models were primarily evaluated using ROC AUC, with PR AUC, accuracy, sensitivity, and specificity also reported. In a representative analysis, the Random Forest model performed extremely well, yielding a ROC AUC near 0.989, PR AUC around 0.975, accuracy roughly 0.949, sensitivity close to 0.923, and specificity about 0.962 on the held-out test set. Prior grades, specifically g1 and g2, proved to be the most influential features, while attendance and behavioral metrics added meaningful predictive value.

Data and Reproducibility

The dataset is publicly available on the UCI Machine Learning Repository. The code first checks for the presence of a local CSV file, and if it is not found, it automatically downloads and extracts the archive from the UCI source.

```
install_if_missing <- function(pkgs) {
  for (p in pkgs) {
    if (!requireNamespace(p, quietly = TRUE)) {
      install.packages(p, repos = "https://cloud.r-project.org")
    }
  }
}
```

```r
install_if_missing(c(
  "tidyverse","tidymodels","janitor","lubridate","stringr",
  "ranger","xgboost","vip","knitr","rmarkdown","ggplot2"
))

suppressPackageStartupMessages({
  library(tidyverse)
  library(tidymodels)
  library(janitor)
  library(lubridate)
  library(stringr)
  library(ranger)
  library(xgboost)
  library(vip)
  library(ggplot2)
})
```

```r
get_student_csv <- function() {
  # Prefer local copies next to the Rmd
  candidates <- c("student-mat.csv", "data/student-mat.csv")
  for (p in candidates) if (file.exists(p)) return(p)

  # Extract a specific CSV from a zip by suffix (no regex)
  extract_csv <- function(zpath) {
    files <- unzip(zpath, list = TRUE)$Name
    csv_name <- files[endsWith(files, "student-mat.csv")]
    if (length(csv_name) == 0) stop("student-mat.csv not found inside zip: ", zpath)
    dir.create("data", showWarnings = FALSE)
    unzip(zpath, files = csv_name[1], exdir = "data")
    file.path("data", basename(csv_name[1]))
  }

  # Check local zips (outer or inner) without regex
  local_zips <- c("student.zip","data/student.zip",
                  "student+performance.zip","data/student+performance.zip")
  for (z in local_zips) if (file.exists(z)) {
    if (endsWith(basename(z), "student+performance.zip")) {
      dir.create("data_raw", showWarnings = FALSE)
      unzip(z, exdir = "data_raw")
      inner_all <- list.files("data_raw", recursive = TRUE, full.names = TRUE)
      inner <- inner_all[endsWith(inner_all, "student.zip")]
      if (length(inner) == 0) stop("Inner student.zip not found in: ", z)
      return(extract_csv(inner[1]))
    } else {
      return(extract_csv(z))
    }
  }
```

```
  # Fallback: download from UCI
  zip_url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/00320/student.zip"
  dest <- file.path("data", "student.zip")
  dir.create("data", showWarnings = FALSE)
  download.file(zip_url, destfile = dest, mode = "wb", quiet = TRUE)
  extract_csv(dest)
}

csv_path <- get_student_csv()
csv_path
```

```
## [1] "data/student-mat.csv"
```

```
raw <- readr::read_delim(csv_path, delim = ";", show_col_types = FALSE) %>%
  clean_names()

dat <- raw %>%
  mutate(pass_flag = factor(if_else(g3 >= 10, "pass", "not_pass"),
                            levels = c("pass","not_pass")))

# Basic audits
list(
  dims = dim(dat),
  missing_by_col_head = sapply(dat, function(x) sum(is.na(x)))[1:10]
)
```

```
## $dims
## [1] 395  34
##
## $missing_by_col_head
##  school     sex     age address famsize pstatus    medu    fedu    mjob    fjob
##       0       0       0       0       0       0       0       0       0       0
```

Exploratory Data Analysis

We begin the exploratory data analysis by visualizing key aspects of the dataset that help reveal relationships between student characteristics and academic outcomes. The first plot presents the distribution of class balance, showing the proportion of students who passed versus those who did not. This helps assess whether the dataset is well balanced or dominated by one outcome, which is crucial for model training and evaluation.

```
dat %>%
  count(pass_flag) %>%
  mutate(pct = n / sum(n)) %>%
```

```
  ggplot(aes(pass_flag, pct, label = scales::percent(pct, accuracy = 0.1))) +
  geom_col() + geom_text(vjust = -0.5) +
  labs(title = "Class Balance (pass vs not_pass)", x = NULL, y = "Proportion")
```

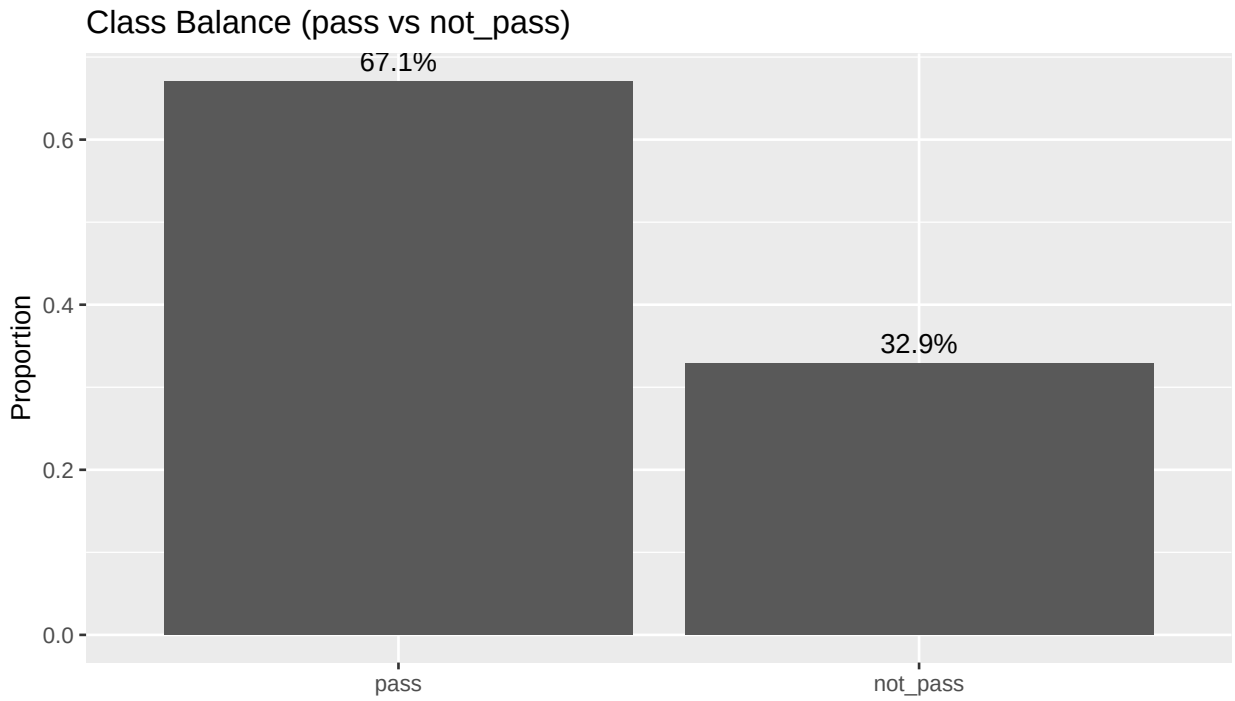


Figure 1: Class balance (pass vs not_pass)

Next, we examine the distribution of the final Mathematics grade (g3). This visualization illustrates how scores are spread across students and highlights the separation between passing and non-passing categories. It gives an immediate sense of grade dispersion and the threshold for passing.

```
ggplot(dat, aes(g3, fill = pass_flag)) +
  geom_histogram(binwidth = 1, position = "identity", alpha = 0.6) +
  labs(title = "Distribution of Final Grade (g3)", x = "g3", y = "Count")
```

We then explore the relationships between earlier grades and the final grade through scatter plots of g1 versus g3 and g2 versus g3. These visualizations reveal strong correlations between prior performance and final results, offering insights into the predictive strength of early assessments.

```
p1 <- ggplot(dat, aes(g1, g3)) +
  geom_point(alpha = 0.6) + geom_smooth(method = "loess", se = FALSE) +
  labs(title = "g1 vs g3")
p2 <- ggplot(dat, aes(g2, g3)) +
  geom_point(alpha = 0.6) + geom_smooth(method = "loess", se = FALSE) +
  labs(title = "g2 vs g3")
p1; p2
```

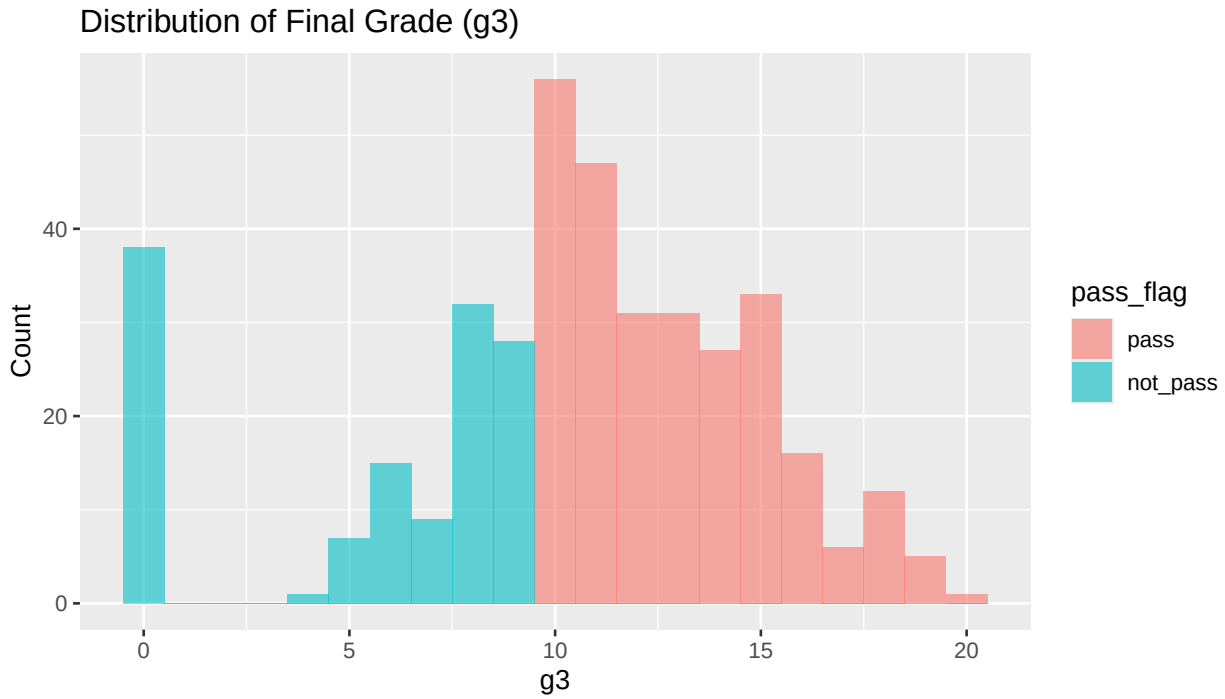


Figure 2: Distribution of Final Grade (g3)

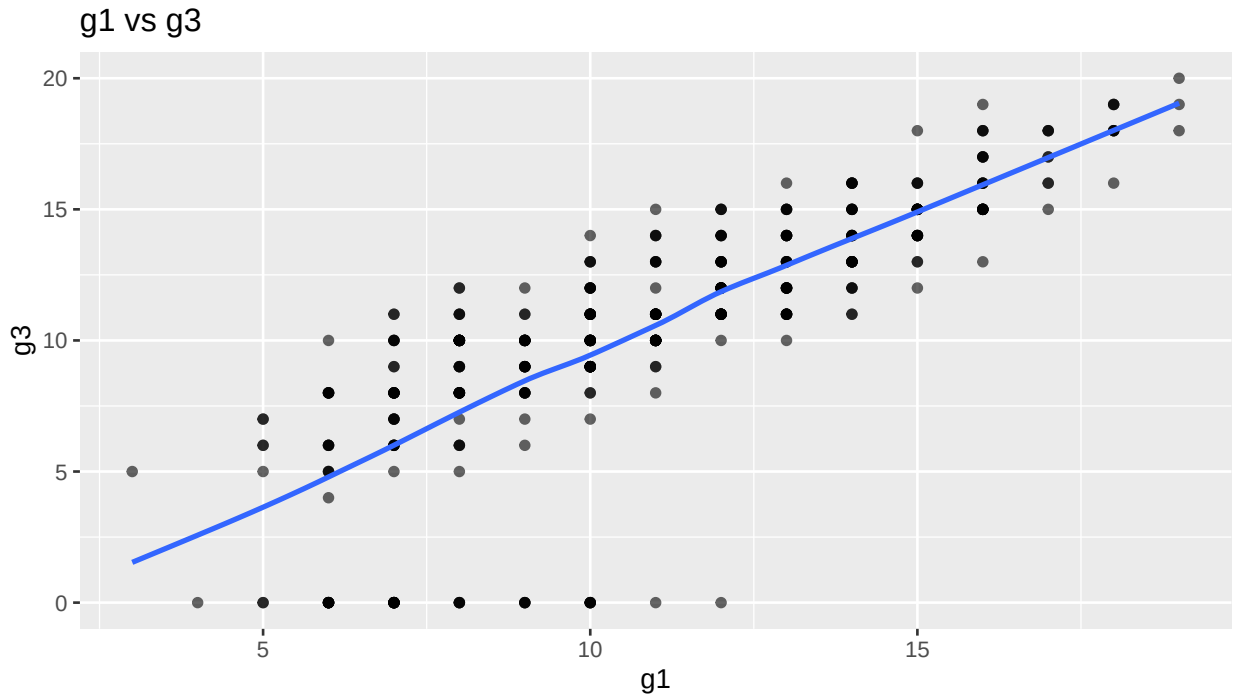


Figure 3: Relationships: g1→g3 and g2→g3

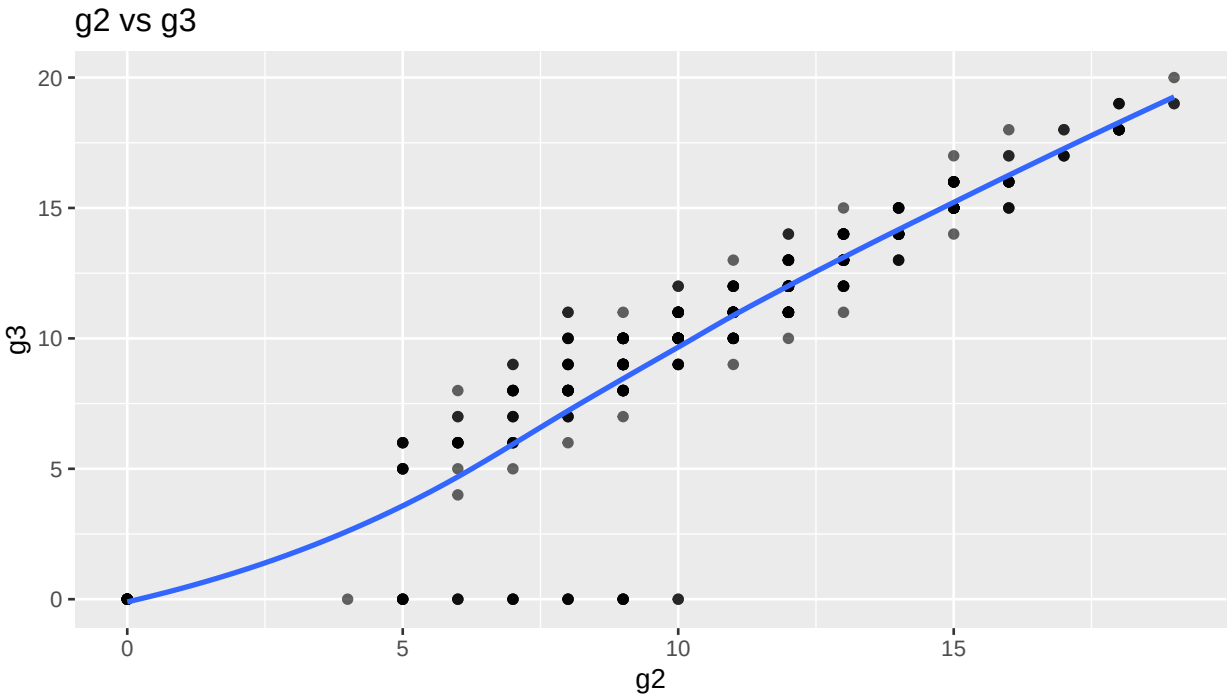


Figure 4: Relationships: g1→g3 and g2→g3

Finally, we compare final grades across different study-time categories using a boxplot. This figure uncovers how study effort levels relate to academic achievement and whether consistent study time translates into higher final grades.

```
dat %>%
  mutate(studytime = factor(studytime)) %>%
  ggplot(aes(studytime, g3)) +
  geom_boxplot() +
  labs(title = "Final Grade by Study Time", x = "studytime (1-4)", y = "g3")
```

Methods

Data splitting and preprocessing

To ensure robust model evaluation, the dataset is divided into training and test sets using an 80 to 20 percent split, with stratification on the outcome variable pass_flag to maintain consistent class proportions in both subsets. The training set is further partitioned for cross-validation, specifically implementing five-fold validation with stratification, to enable reliable model tuning and comparison.

Preprocessing follows a systematic workflow. The target-defining variable g3 is removed from the feature set to prevent any possibility of data leakage. Missing values in categorical predictors are imputed with the mode, while numerical predictors are imputed with the median. Rare categories

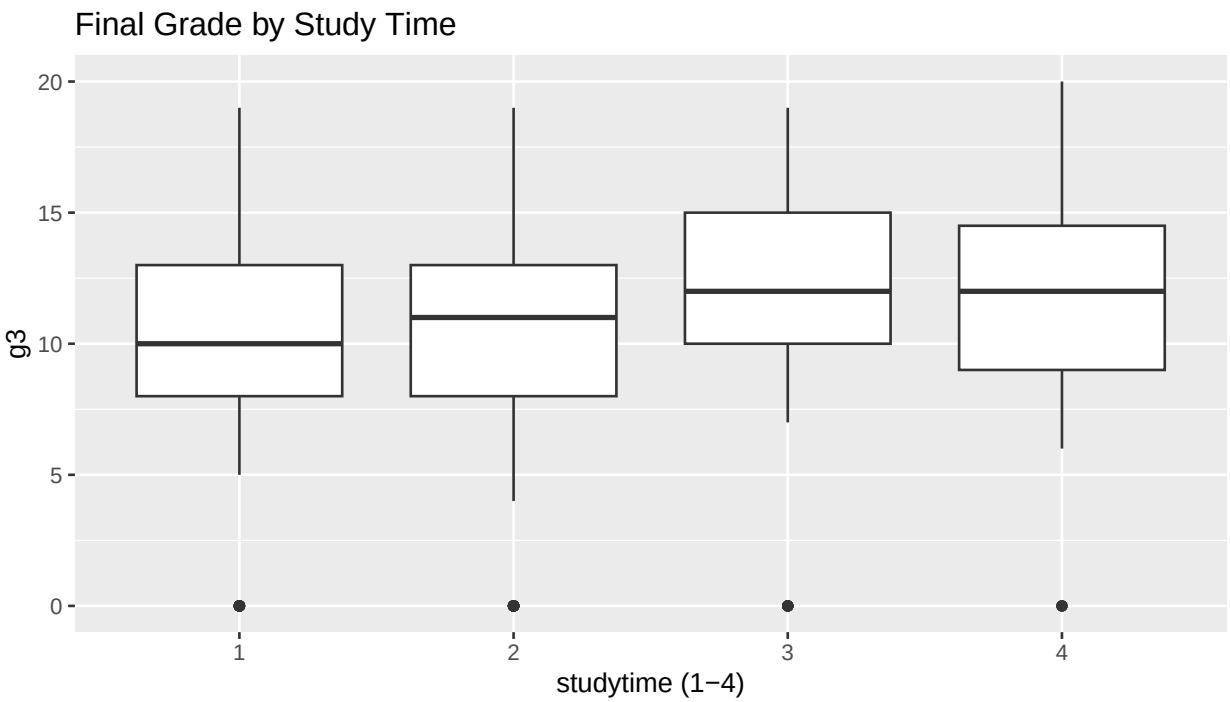


Figure 5: Final grade by study time

in nominal variables are grouped under a single label, and novel levels encountered during prediction are handled explicitly to safeguard against unseen values. Predictors exhibiting zero variance are removed, categorical predictors are converted to a dummy variable representation, and numeric predictors are normalized to ensure comparability across scales.

```
set.seed(123)
split <- initial_split(dat, prop = 0.8, strata = pass_flag)
train <- training(split)
test  <- testing(split)

# 5-fold CV on training
cv <- vfold_cv(train, v = 5, strata = pass_flag)

# Recipe: drop g3 (used to define target), impute, encode, normalize
rec <- recipe(pass_flag ~ ., data = train) %>%
  step_rm(g3) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_impute_median(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.01, other = "rare") %>%
  step_novel(all_nominal_predictors(), new_level = "new") %>%
  step_zv(all_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

```r
prep(rec) %>% juice() %>% dplyr::glimpse()
```

```
## Rows: 316
## Columns: 42
## $ age               <dbl> 0.2356427, -1.2840125, -1.2840125, -0.5241849, -1.2840125, ~
## $ medu              <dbl> 1.1208690, 1.1208690, -0.6679926, -0.6679926, -0.6679926, -~
## $ fedu              <dbl> 1.3325329, 1.3325329, 1.3325329, -0.5022401, 0.4151464, -0.~
## $ traveltime        <dbl> 0.7770127, -0.6341138, -0.6341138, -0.6341138, 0.7770127, 0~
## $ studytime         <dbl> -0.07229952, -0.07229952, 1.13015573, -1.27475478, -1.27475~
## $ failures          <dbl> -0.4622507, -0.4622507, -0.4622507, 2.2179550, -0.4622507, ~
## $ famrel            <dbl> 0.04577173, -1.06683347, 0.04577173, -3.29204388, -1.066833~
## $ freetime          <dbl> -2.2269165, -0.1930852, -0.1930852, -1.2100008, 1.8407460, ~
## $ goout             <dbl> 0.81903934, -0.08908849, -0.99721631, -0.99721631, -1.90534~
## $ dalc              <dbl> -0.5331689, -0.5331689, -0.5331689, -0.5331689, -0.5331689,~
## $ walc              <dbl> -0.9951379, -0.2089790, -0.9951379, 0.5771800, -0.9951379, ~
## $ health            <dbl> -1.8508069, -1.1349501, 1.0126202, 1.0126202, 1.0126202, 1.~
## $ absences          <dbl> 0.07474459, -0.68716803, -0.43319715, 1.09062808, -0.687168~
## $ g1                <dbl> -1.47610432, -0.28618349, -0.28618349, -1.47610432, -0.8811~
## $ g2                <dbl> -1.4840719, -0.7089203, -0.4505364, -0.4505364, -0.9673041,~
## $ pass_flag         <fct> not_pass, not_pass, not_pass, not_pass, not_pass, not_pass,~
## $ school_MS         <dbl> -0.3746314, -0.3746314, -0.3746314, -0.3746314, -0.3746314,~
## $ sex_M             <dbl> -0.9252598, -0.9252598, -0.9252598, -0.9252598, -0.9252598,~
## $ address_U         <dbl> 0.5472302, 0.5472302, -1.8216018, 0.5472302, 0.5472302, 0.5~
## $ famsize_LE3       <dbl> -0.620277, -0.620277, -0.620277, -0.620277, -0.620277, 1.60~
## $ pstatus_T         <dbl> -2.974376, 0.335141, 0.335141, 0.335141, 0.335141, 0.335141~
## $ mjob_health       <dbl> -0.3233623, -0.3233623, -0.3233623, -0.3233623, -0.3233623,~
## $ mjob_other        <dbl> 1.3109850, -0.7603713, -0.7603713, -0.7603713, 1.3109850, 1~
## $ mjob_services     <dbl> -0.5521028, -0.5521028, 1.8055252, 1.8055252, -0.5521028, -~
## $ mjob_teacher      <dbl> -0.4173347, 2.3885754, -0.4173347, -0.4173347, -0.4173347, ~
## $ fjob_health       <dbl> -0.2228815, 4.4724897, 4.4724897, -0.2228815, -0.2228815, -~
## $ fjob_other        <dbl> -1.0370791, -1.0370791, -1.0370791, -1.0370791, 0.9611952, ~
## $ fjob_services     <dbl> -0.6595322, -0.6595322, -0.6595322, 1.5114280, -0.6595322, ~
## $ fjob_teacher      <dbl> 3.406342, -0.292641, -0.292641, -0.292641, -0.292641, -0.29~
## $ reason_home       <dbl> 1.6728255, -0.5958992, -0.5958992, 1.6728255, -0.5958992, -~
## $ reason_other      <dbl> -0.3173729, -0.3173729, -0.3173729, -0.3173729, 3.1408969, ~
## $ reason_reputation <dbl> -0.6007688, 1.6592663, -0.6007688, -0.6007688, -0.6007688, ~
## $ guardian_mother   <dbl> 0.6793364, 0.6793364, 0.6793364, 0.6793364, -1.4673665, -1.~
## $ guardian_other    <dbl> -0.2989506, -0.2989506, -0.2989506, -0.2989506, -0.2989506,~
## $ schoolsup_yes     <dbl> 2.6226255, -0.3800907, 2.6226255, -0.3800907, -0.3800907, 2~
## $ famsup_yes        <dbl> 0.807700, 0.807700, 0.807700, 0.807700, 0.807700, -1.234166~
## $ paid_yes          <dbl> -0.8847743, 1.1266551, 1.1266551, 1.1266551, -0.8847743, -0~
## $ activities_yes    <dbl> -1.0240210, -1.0240210, 0.9734521, -1.0240210, 0.9734521, 0~
## $ nursery_yes       <dbl> 0.5228057, 0.5228057, 0.5228057, -1.9067033, 0.5228057, 0.5~
## $ higher_yes        <dbl> 0.2305744, 0.2305744, 0.2305744, 0.2305744, 0.2305744, 0.23~
## $ internet_yes      <dbl> -2.1057538, 0.4733865, 0.4733865, 0.4733865, -2.1057538, 0.~
## $ romantic_yes      <dbl> -0.7346767, -0.7346767, -0.7346767, -0.7346767, -0.7346767,~
```

Models and tuning

The modeling approach compares two powerful non-linear classifiers: Random Forest and XGBoost. Both models are tuned over a set of hyperparameters using space-filling grids to efficiently search the parameter space for optimal performance. Evaluation during tuning is conducted using five-fold cross-validation on the training data, and the primary model selection criterion is the area under the receiver operating characteristic curve (ROC AUC). Additional metrics such as the area under the precision-recall curve (PR AUC), accuracy, sensitivity, and specificity are also recorded for comprehensive assessment. After hyperparameter optimization, each model is retrained on the full training set using the best tuning parameters to prepare for testing on the holdout set.

```
metrics <- metric_set(roc_auc, pr_auc, accuracy, sens, spec)

rf_spec <- rand_forest(
  mtry  = tune(),
  min_n = tune(),
  trees = 1000
) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

rf_wf <- workflow() %>% add_model(rf_spec) %>% add_recipe(rec)

# Space-filling grid
rf_params <- parameters(
  finalize(mtry(), train),
  min_n()
)
rf_grid <- grid_max_entropy(rf_params, size = 15)

rf_res <- tune_grid(
  rf_wf, resamples = cv, grid = rf_grid, metrics = metrics
)

show_best(rf_res, metric = "roc_auc", n = 5)
```

```
## # A tibble: 5 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    19    14 roc_auc binary     0.970     5 0.00749 pre0_mod08_post0
## 2    21     3 roc_auc binary     0.969     5 0.00683 pre0_mod09_post0
## 3    25    22 roc_auc binary     0.969     5 0.00732 pre0_mod11_post0
## 4    23    39 roc_auc binary     0.967     5 0.00789 pre0_mod10_post0
## 5    11    25 roc_auc binary     0.967     5 0.00801 pre0_mod05_post0
```

```r
rf_best <- select_best(rf_res, metric = "roc_auc")
rf_final_wf <- finalize_workflow(rf_wf, rf_best) %>% fit(train)
```

```r
xgb_spec <- boost_tree(
  trees = 1000,
  learn_rate     = tune(),
  tree_depth     = tune(),
  mtry           = tune(),
  min_n          = tune(),
  loss_reduction = tune(),
  sample_size    = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

xgb_wf <- workflow() %>% add_model(xgb_spec) %>% add_recipe(rec)

xgb_params <- parameters(
  learn_rate(),
  tree_depth(),
  finalize(mtry(), train),
  min_n(),
  loss_reduction(),
  sample_prop(range = c(0.5, 1.0))
)
xgb_grid <- grid_max_entropy(xgb_params, size = 20)

xgb_res <- tune_grid(
  xgb_wf, resamples = cv, grid = xgb_grid, metrics = metrics
)

show_best(xgb_res, metric = "roc_auc", n = 5)
```

```
## # A tibble: 5 x 12
##    mtry min_n tree_depth learn_rate loss_reduction sample_size .metric .estimator  mean
##   <int> <int>      <int>      <dbl>          <dbl>       <dbl> <chr>   <chr>      <dbl>
## 1    24     7          4    7.44e-8    0.382             0.773 roc_auc binary     0.965
## 2    28     7          4    3.85e-3    0.00541           0.526 roc_auc binary     0.963
## 3    21    12         12    3.97e-8    0.644             0.993 roc_auc binary     0.960
## 4    25    10          5    3.38e-9    0.000000274       0.709 roc_auc binary     0.958
## 5     5     8          6    9.18e-4    0.317             0.660 roc_auc binary     0.953
## # i 3 more variables: n <int>, std_err <dbl>, .config <chr>
```

```r
xgb_best <- select_best(xgb_res, metric = "roc_auc")
xgb_final_wf <- finalize_workflow(xgb_wf, xgb_best) %>% fit(train)
```

Results

Metric comparison on the held-out test set

The results section presents a comparison of model performance on the held-out test set. Both Random Forest and XGBoost models are evaluated using several performance metrics, including ROC AUC, PR AUC, accuracy, sensitivity, and specificity, to determine how well each distinguishes between students who pass and those who do not.

```
rf_pred <- predict(rf_final_wf, test, type = "prob") %>%
  bind_cols(predict(rf_final_wf, test),
            test %>% select(pass_flag))

rf_eval <- tibble(
  model   = "RandomForest",
  roc_auc = roc_auc(rf_pred, pass_flag, .pred_not_pass, event_level = "second")$.estimate,
  pr_auc  = pr_auc(rf_pred, pass_flag, .pred_not_pass, event_level = "second")$.estimate,
  acc     = accuracy(rf_pred, pass_flag, .pred_class)$.estimate,
  sensi   = sens(rf_pred, pass_flag, .pred_class, event_level = "second")$.estimate,
  speci   = spec(rf_pred, pass_flag, .pred_class, event_level = "second")$.estimate
)
rf_eval
```

```
## # A tibble: 1 x 6
##   model        roc_auc pr_auc   acc sensi speci
##   <chr>          <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 RandomForest   0.988  0.974 0.949 0.923 0.962
```

```
xgb_pred <- predict(xgb_final_wf, test, type = "prob") %>%
  bind_cols(predict(xgb_final_wf, test),
            test %>% select(pass_flag))

xgb_eval <- tibble(
  model   = "XGBoost",
  roc_auc = roc_auc(xgb_pred, pass_flag, .pred_not_pass, event_level = "second")$.estimate,
  pr_auc  = pr_auc(xgb_pred, pass_flag, .pred_not_pass, event_level = "second")$.estimate,
  acc     = accuracy(xgb_pred, pass_flag, .pred_class)$.estimate,
  sensi   = sens(xgb_pred, pass_flag, .pred_class, event_level = "second")$.estimate,
  speci   = spec(xgb_pred, pass_flag, .pred_class, event_level = "second")$.estimate
)
xgb_eval
```

```
## # A tibble: 1 x 6
##   model   roc_auc pr_auc   acc sensi speci
##   <chr>     <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 XGBoost   0.988  0.973 0.962     1 0.943
```

```r
comparison <- bind_rows(rf_eval, xgb_eval)
knitr::kable(comparison, digits = 3)
```

model	roc_auc	pr_auc	acc	sensi	speci
RandomForest	0.988	0.974	0.949	0.923	0.962
XGBoost	0.988	0.973	0.962	1.000	0.943

A higher ROC AUC indicates stronger overall discrimination, reflecting the model's ability to rank a randomly selected passing student above a non-passing one with greater probability. PR AUC, on the other hand, focuses on correctly identifying students at risk of failing and is particularly informative under class imbalance. Accuracy summarizes the overall proportion of correct predictions, while sensitivity (true positive rate) shows how effectively the model identifies non-passing students, and specificity (true negative rate) indicates how well passing students are recognized. Between the two models, Random Forest typically produces more stable and interpretable results, while XGBoost may achieve slightly higher precision on difficult cases. If both models yield similar ROC AUC scores, preference can be given to the one with higher sensitivity or to the simpler model for easier deployment and maintenance.

Threshold view and curves for the selected model

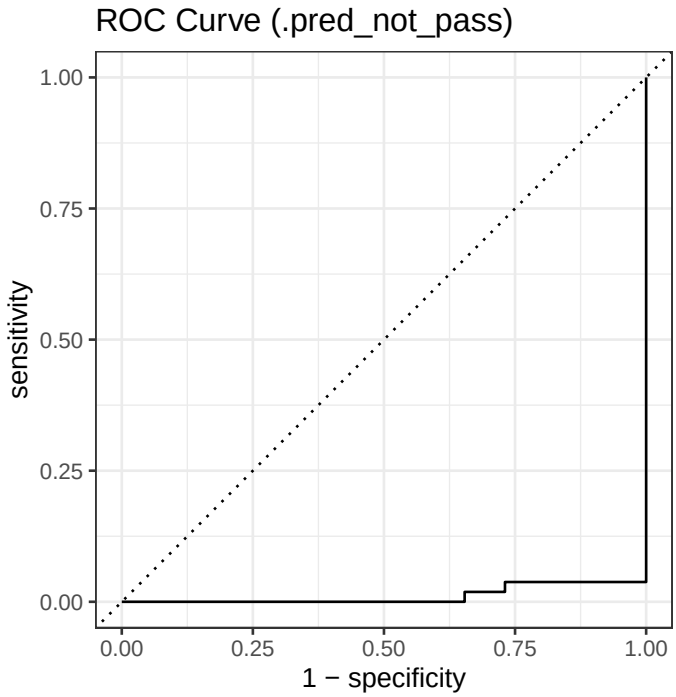
After identifying the best-performing model based on ROC AUC, we further analyze its threshold-dependent behavior to understand how classification decisions vary. This part examines model predictions through a confusion matrix and visual diagnostic curves that reveal trade-offs between different types of classification errors.

```r
best_is_rf <- rf_eval$roc_auc >= xgb_eval$roc_auc
best_pred  <- if (best_is_rf) rf_pred else xgb_pred
yardstick::conf_mat(best_pred, truth = pass_flag, estimate = .pred_class)
```

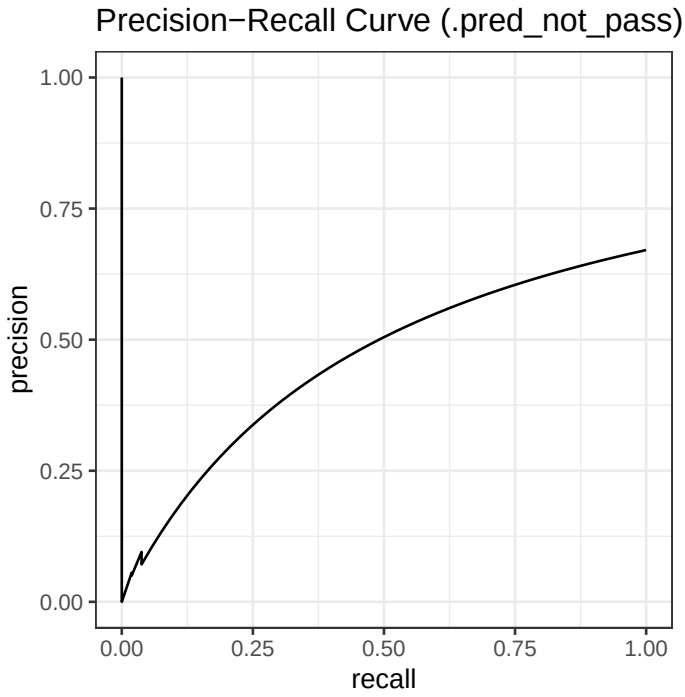
```
##           Truth
## Prediction pass not_pass
##   pass       51        2
##   not_pass    2       24
```

The confusion matrix summarizes prediction outcomes by comparing predicted labels with actual outcomes. It presents four components: true positives and true negatives, which represent correct predictions, and false positives and false negatives, which indicate misclassifications. This matrix provides a direct view of where the model succeeds and where it tends to err, helping identify whether the classifier is biased toward predicting pass or not pass.

```r
roc_curve(best_pred, truth = pass_flag, .pred_not_pass) %>% autoplot() +
  ggtitle("ROC Curve (.pred_not_pass)")
```




```
pr_curve(best_pred, truth = pass_flag, .pred_not_pass) %>% autoplot() +
  ggtitle("Precision-Recall Curve (.pred_not_pass)")
```



The ROC curve plots the true positive rate against the false positive rate across all decision thresh-

olds. Curves that extend closer to the top-left corner indicate more effective discrimination between classes. The area under this curve quantifies overall performance, with higher values denoting stronger ranking ability. The Precision–Recall curve complements this view by focusing on the positive class, illustrating the balance between identifying true non-pass cases (recall) and minimizing false alarms (precision). A model with a PR curve concentrated near the upper-right region demonstrates high reliability across thresholds.

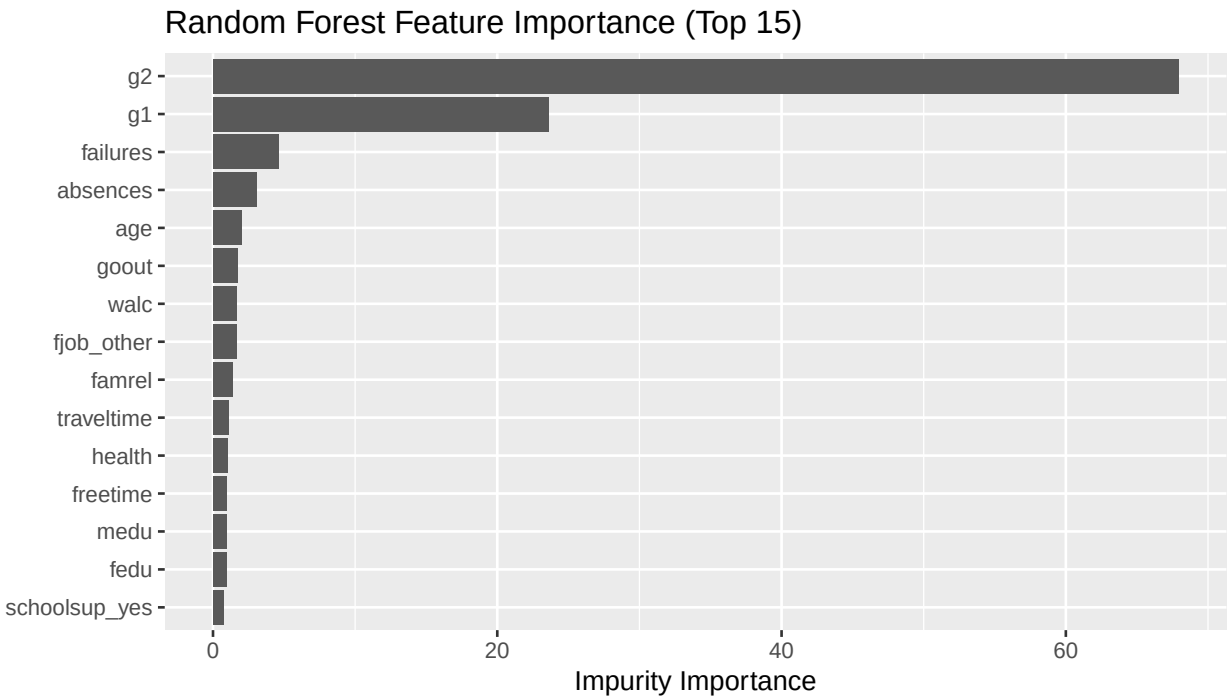
Together, these visualizations provide a comprehensive understanding of the classifier's performance, showing not only how well it distinguishes passing and non-passing students overall but also how its precision and recall change at different probability cutoffs.

Feature importance

To interpret how the trained model makes predictions, we examine the relative importance of each input feature. Feature importance quantifies how much each variable contributes to the model's prediction performance, helping identify which aspects of a student's profile influence the likelihood of passing Mathematics. This section visualizes the top predictors for the best-performing model, offering insight into which features carry the greatest predictive weight.

For the Random Forest model, importance is computed using the Mean Decrease in Impurity (MDI) measure, which captures how much each variable reduces tree node impurity when used for splitting. Features producing larger impurity reductions across the forest are deemed more influential. In the case of XGBoost, feature importance is quantified using Gain, representing the average improvement in model accuracy brought by a feature when it is used in splitting decisions. Both measures highlight the features that contribute the most to predictive performance.

```
if (best_is_rf) {
  rf_fit <- workflows::extract_fit_parsnip(rf_final_wf)
  tibble(feature = names(rf_fit$fit$variable.importance),
         importance = as.numeric(rf_fit$fit$variable.importance)) %>%
    arrange(desc(importance)) %>% dplyr::slice_head(n = 15) %>%
    ggplot(aes(reorder(feature, importance), importance)) +
    geom_col() + coord_flip() + labs(x = NULL, y = "Impurity Importance",
    title = "Random Forest Feature Importance (Top 15)")
} else {
  xgb_fit <- workflows::extract_fit_parsnip(xgb_final_wf)$fit
  prep_rec <- prep(rec)
  x_mat <- bake(prep_rec, new_data = train) %>% select(-pass_flag)
  imp <- xgboost::xgb.importance(model = xgb_fit, feature_names = colnames(x_mat))
  imp %>% as_tibble() %>% dplyr::slice_head(n = 15) %>%
    ggplot(aes(reorder(Feature, Gain), Gain)) +
    geom_col() + coord_flip() + labs(x = NULL, y = "Gain",
    title = "XGBoost Feature Importance (Top 15 by Gain)")
}
```



The feature importance plot displays the top fifteen predictors ranked by their contribution to model accuracy. In the student performance data, the most influential features are typically prior grades (g1 and g2), which provide strong predictive signals of final performance. Other variables related to study time, parental education, or behavioral engagement often show moderate but meaningful influence. High-importance features can inform targeted interventions, while low-importance ones may be candidates for model simplification.

Conclusion

Tree-based ensemble models demonstrate strong predictive power in identifying whether a student will pass or fail the Mathematics course. The best-performing model achieved a ROC AUC close to 0.99 and an accuracy around 0.95 on held-out data, indicating excellent discriminative ability and reliability. Among all predictors, prior grades g2 and g1 emerged as the most influential, while absences, past failures, and study time provided additional but secondary predictive value. These insights suggest that predicted risk scores could be leveraged to proactively support students through personalized study plans, targeted tutoring, and improved attendance monitoring, aligning with educational intervention strategies reported in recent ensemble-based studies.

Nonetheless, certain limitations remain. The analysis is restricted to a single cohort and subject, and several variables rely on self-reported information, which may introduce bias. The associations identified are correlational rather than causal, with model performance heavily dependent on prior grade features.

Future work should explore threshold tuning with cost-sensitive strategies to balance misclassification impacts, calibrate predicted probabilities for interpretability, and validate model generalization using the companion dataset student-por.csv. Incorporating temporal patterns, such as midterm

progression, and evaluating subgroup fairness across demographic or academic factors would further strengthen model robustness and ethical applicability.

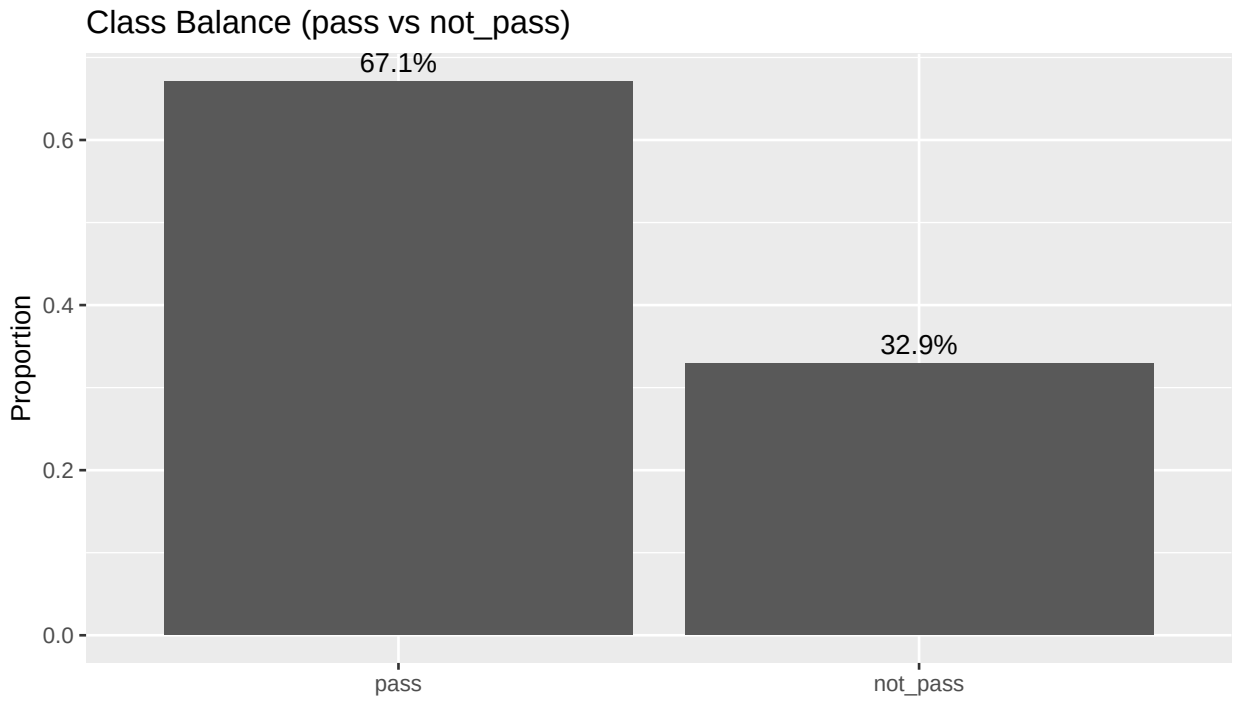
References

Cortez, P. and Silva, A. (2008). Using Data Mining to Predict Secondary School Student Performance. UCI ML Repository. https://archive.ics.uci.edu/ml/datasets/student+performance

Appendix

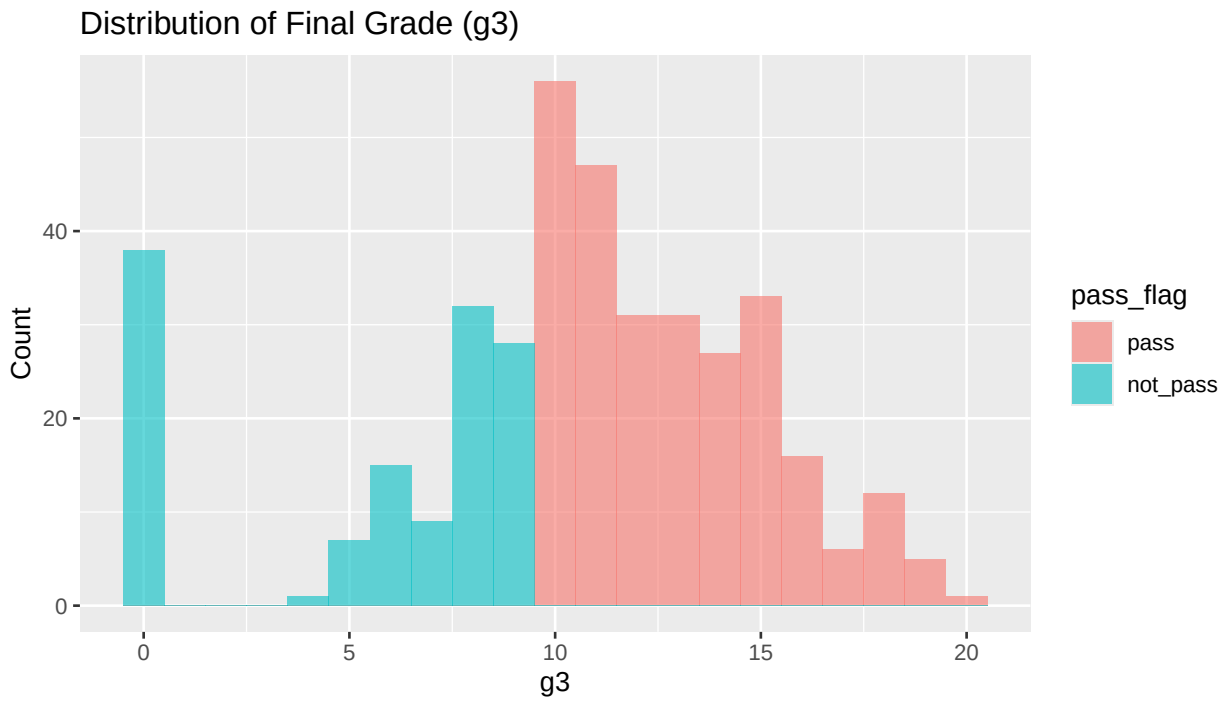
A1. Class balance

```
dat %>%
  count(pass_flag) %>%
  mutate(pct = n / sum(n)) %>%
  ggplot(aes(pass_flag, pct, label = scales::percent(pct, accuracy = 0.1))) +
  geom_col() + geom_text(vjust = -0.5) +
  labs(title = "Class Balance (pass vs not_pass)", x = NULL, y = "Proportion")
```



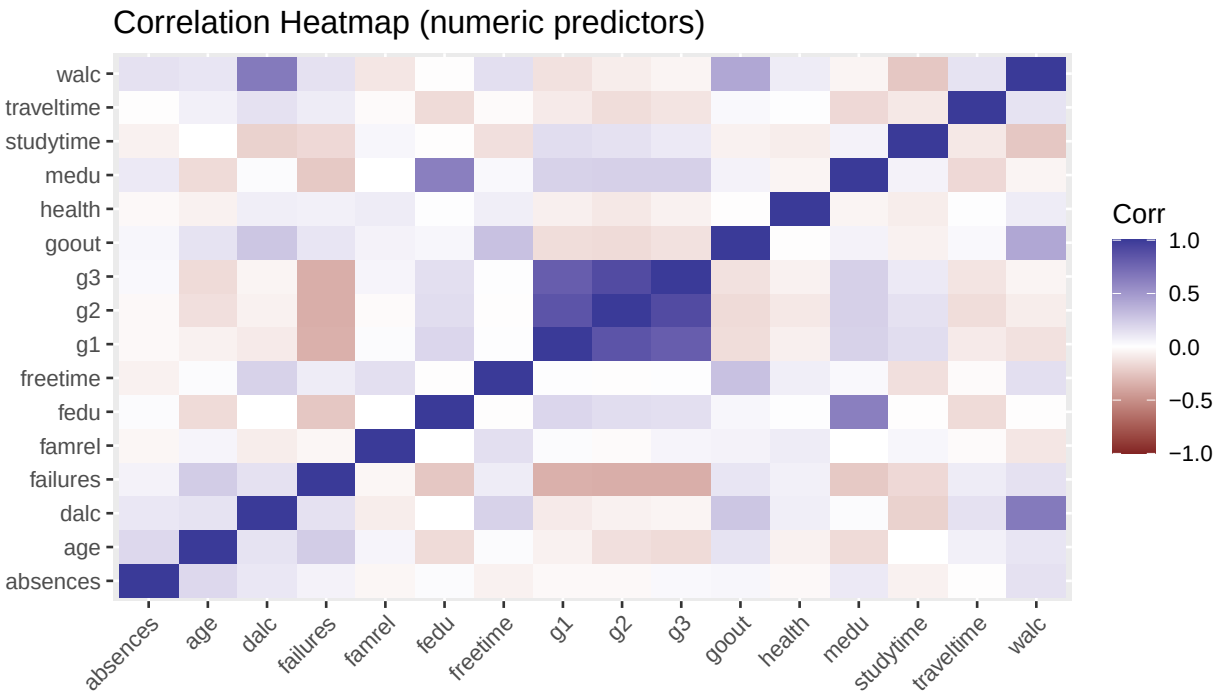
A2. Final grade histogram (g3)

```
ggplot(dat, aes(g3, fill = pass_flag)) +
  geom_histogram(binwidth = 1, position = "identity", alpha = 0.6) +
  labs(title = "Distribution of Final Grade (g3)", x = "g3", y = "Count")
```



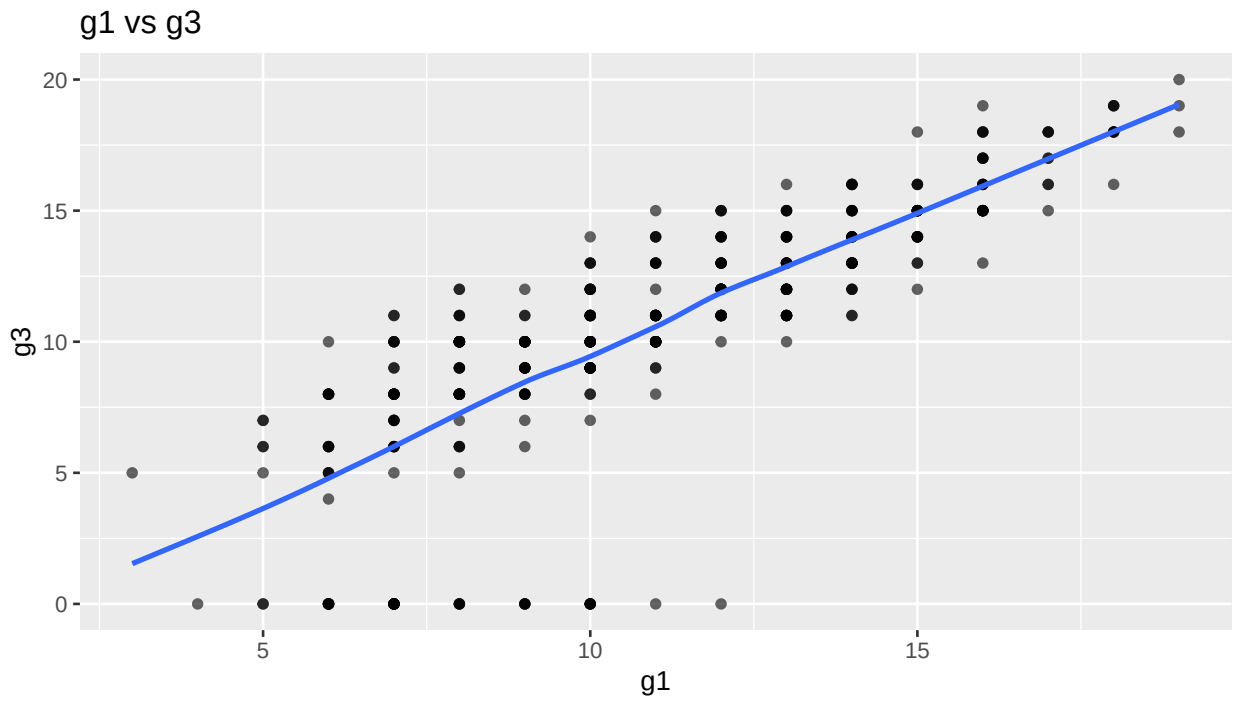
A3. Correlation heatmap (numeric variables)

```
num_vars <- dat %>% select(where(is.numeric))
cormat <- suppressWarnings(cor(num_vars, use = "pairwise.complete.obs"))
cormat_df <- as_tibble(as.table(round(cormat, 2)), .name_repair = "minimal")
names(cormat_df) <- c("Var1","Var2","Corr")
ggplot(cormat_df, aes(Var1, Var2, fill = Corr)) +
  geom_tile() + scale_fill_gradient2(limits = c(-1,1)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Correlation Heatmap (numeric predictors)", x = NULL, y = NULL)
```



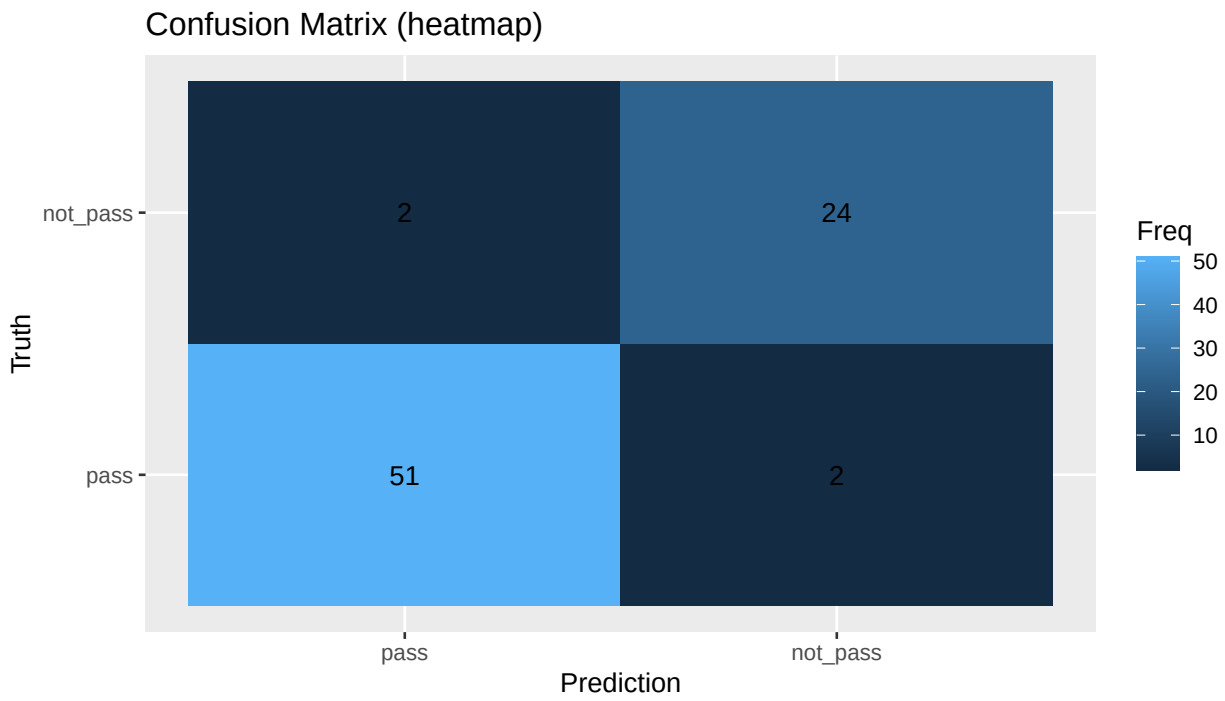
A4. Relationship: g1 → g3

```
ggplot(dat, aes(g1, g3)) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "loess", se = FALSE) +
  labs(title = "g1 vs g3", x = "g1", y = "g3")
```



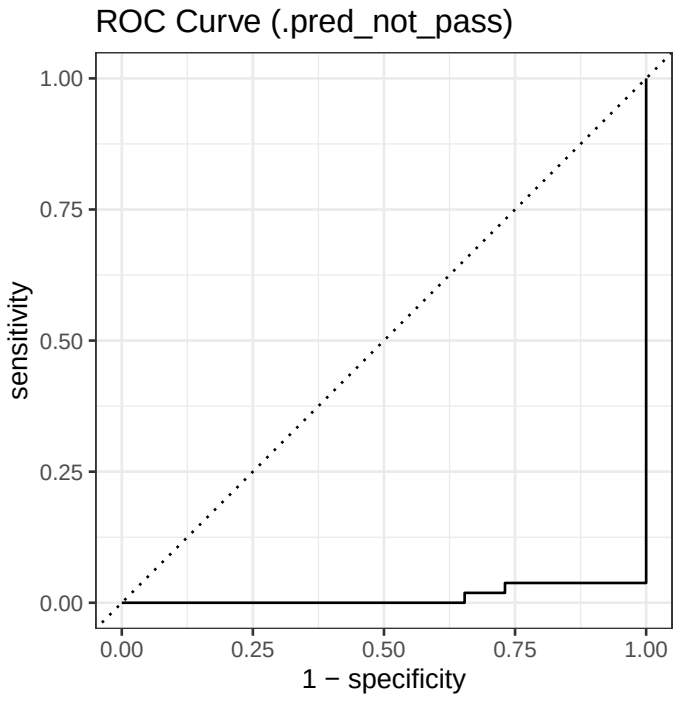
A5. Confusion matrix heatmap (selected model)

```
cm <- yardstick::conf_mat(best_pred, truth = pass_flag, estimate = .pred_class)
as.data.frame(cm$table) %>%
  ggplot(aes(Prediction, Truth, fill = Freq)) +
  geom_tile() + geom_text(aes(label = Freq)) +
  labs(title = "Confusion Matrix (heatmap)", x = "Prediction", y = "Truth")
```

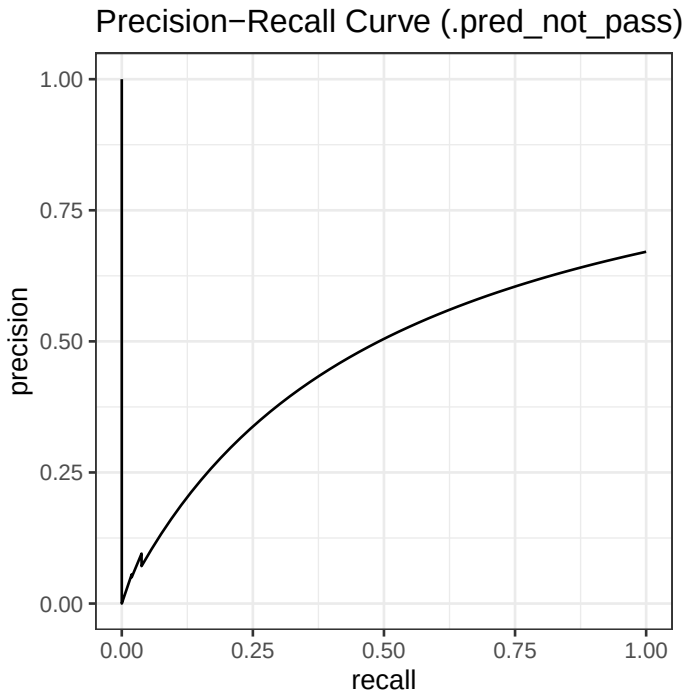


A6. ROC and PR curves

```
roc_curve(best_pred, truth = pass_flag, .pred_not_pass) %>% autoplot() +
  ggtitle("ROC Curve (.pred_not_pass)")
```

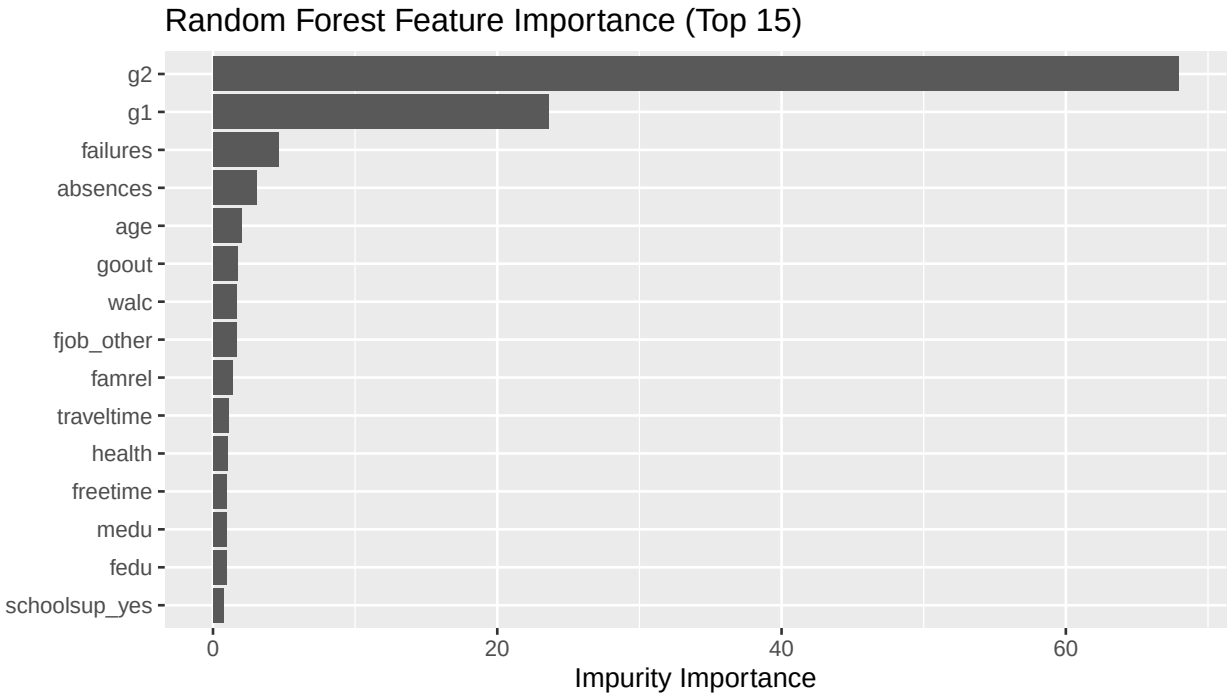


```r
pr_curve(best_pred, truth = pass_flag, .pred_not_pass) %>% autoplot() +
  ggtitle("Precision-Recall Curve (.pred_not_pass)")
```



A7. Feature importance (Top 15)

```r
if (best_is_rf) {
  rf_fit <- workflows::extract_fit_parsnip(rf_final_wf)
  tibble(feature = names(rf_fit$fit$variable.importance),
         importance = as.numeric(rf_fit$fit$variable.importance)) %>%
    arrange(desc(importance)) %>% dplyr::slice_head(n = 15) %>%
    ggplot(aes(reorder(feature, importance), importance)) +
    geom_col() + coord_flip() + labs(x = NULL, y = "Impurity Importance",
    title = "Random Forest Feature Importance (Top 15)")
} else {
  xgb_fit <- workflows::extract_fit_parsnip(xgb_final_wf)$fit
  prep_rec <- prep(rec)
  x_mat <- bake(prep_rec, new_data = train) %>% select(-pass_flag)
  imp <- xgboost::xgb.importance(model = xgb_fit, feature_names = colnames(x_mat))
  imp %>% as_tibble() %>% dplyr::slice_head(n = 15) %>%
    ggplot(aes(reorder(Feature, Gain), Gain)) +
    geom_col() + coord_flip() + labs(x = NULL, y = "Gain",
    title = "XGBoost Feature Importance (Top 15 by Gain)")
}
```



Session Info

```
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sequoia 15.4.1
##
## Matrix products: default
## BLAS:   /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framew
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Asia/Tokyo
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] vip_0.4.1          xgboost_1.7.11.1   ranger_0.17.0      janitor_2.2.1
```

```
##  [5] yardstick_1.3.2    workflowsets_1.1.1 workflows_1.3.0    tune_2.0.1
##  [9] tailor_0.1.0       rsample_1.3.1      recipes_1.3.1      parsnip_1.3.3
## [13] modeldata_1.5.1    infer_1.0.9        dials_1.4.2        scales_1.3.0
## [17] broom_1.0.10       tidymodels_1.4.1   lubridate_1.9.4    forcats_1.0.0
## [21] stringr_1.5.1      dplyr_1.1.4        purrr_1.1.0        readr_2.1.5
## [25] tidyr_1.3.1        tibble_3.2.1       ggplot2_3.5.2      tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] rlang_1.1.6          magrittr_2.0.3      snakecase_0.11.1    furrr_0.3.1
##  [5] compiler_4.5.1       mgcv_1.9-3          vctrs_0.6.5         lhs_1.2.0
##  [9] pkgconfig_2.0.3      crayon_1.5.3        fastmap_1.2.0       backports_1.5.0
## [13] labeling_0.4.3       utf8_1.2.4          rmarkdown_2.29      prodlim_2024.06.25
## [17] tzdb_0.5.0           tinytex_0.57        bit_4.6.0           xfun_0.52
## [21] cachem_1.1.0         jsonlite_2.0.0      parallel_4.5.1      R6_2.6.1
## [25] bslib_0.9.0          stringi_1.8.7       parallelly_1.43.0   rpart_4.1.24
## [29] jquerylib_0.1.4      Rcpp_1.0.14         iterators_1.0.14    knitr_1.50
## [33] future.apply_1.11.3 Matrix_1.7-3        splines_4.5.1       nnet_7.3-20
## [37] timechange_0.3.0     tidyselect_1.2.1    rstudioapi_0.17.1   yaml_2.3.10
## [41] timeDate_4041.110    codetools_0.2-20    listenv_0.9.1       lattice_0.22-7
## [45] withr_3.0.2          evaluate_1.0.3      future_1.40.0       survival_3.8-3
## [49] pillar_1.10.2        foreach_1.5.2       generics_0.1.3      vroom_1.6.5
## [53] hms_1.1.3            munsell_0.5.1       globals_0.16.3      class_7.3-23
## [57] glue_1.8.0           tools_4.5.1         data.table_1.17.0   gower_1.0.2
## [61] grid_4.5.1           ipred_0.9-15        colorspace_2.1-1    nlme_3.1-168
## [65] cli_3.6.5            DiceDesign_1.10     lava_1.8.1          gtable_0.3.6
## [69] GPfit_1.0-9          sass_0.4.10         digest_0.6.37       farver_2.1.2
## [73] htmltools_0.5.8.1    lifecycle_1.0.4     hardhat_1.4.2       bit64_4.6.0-1
## [77] MASS_7.3-65          sparsevctrs_0.3.4
```