

# Probability Modeling

DS 6030 | Fall 2024

prob-modeling.pdf

## Contents

<b>1</b>	<b>Probability Modeling Intro</b>	<b>2</b>
1.1	Credit Card Default data (Default) . . . . .	2
1.2	Set-up . . . . .	5
1.3	Binary Probability/Risk Modeling . . . . .	5
<b>2</b>	<b>Logistic Regression</b>	<b>9</b>
2.1	Basics . . . . .	9
2.2	Estimation . . . . .	9
2.3	Logistic Regression in Action . . . . .	11
2.4	Logistic Regression Summary . . . . .	13
<b>3</b>	<b>Evaluating Binary Risk Models</b>	<b>15</b>
3.1	Common Binary Loss Functions . . . . .	15
3.2	Model Comparison . . . . .	16
3.3	Area under the ROC curve (AUC or AUROC or C-Statistic) . . . . .	16
3.4	Calibration . . . . .	18
<b>4</b>	<b>Appendix: R Code</b>	<b>20</b>

---

# 1 Probability Modeling Intro

## 1.1 Credit Card Default data (Default)

The textbook *An Introduction to Statistical Learning (ISL)* has a description of a simulated credit card default dataset. The interest is on predicting whether an individual will default on their credit card payment.

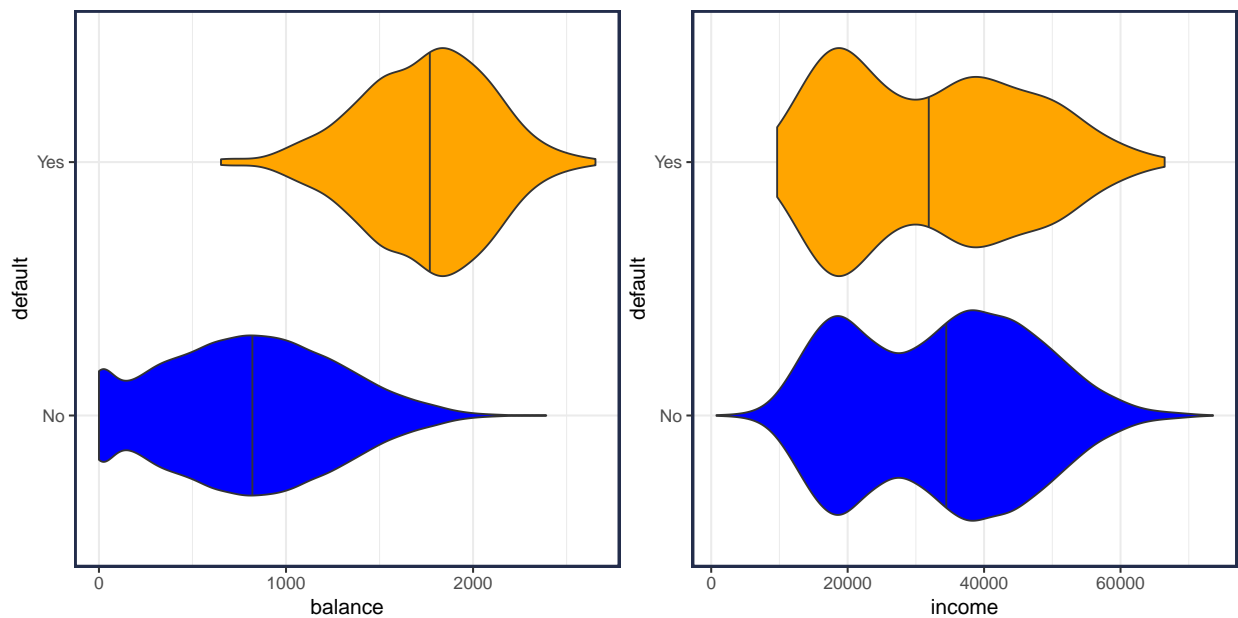
```
data(Default, package="ISLR")
```

The variables are:

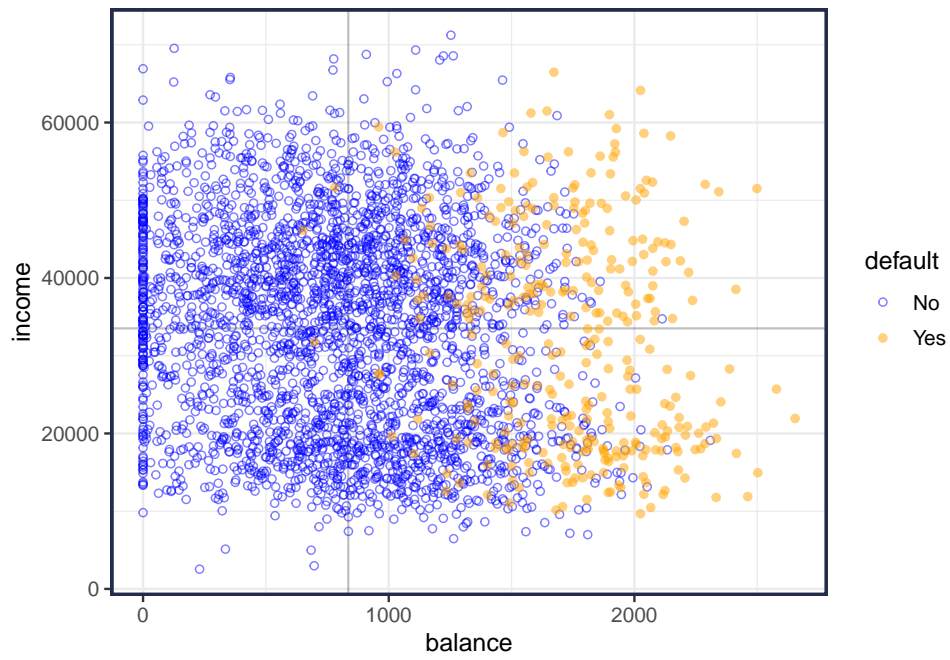
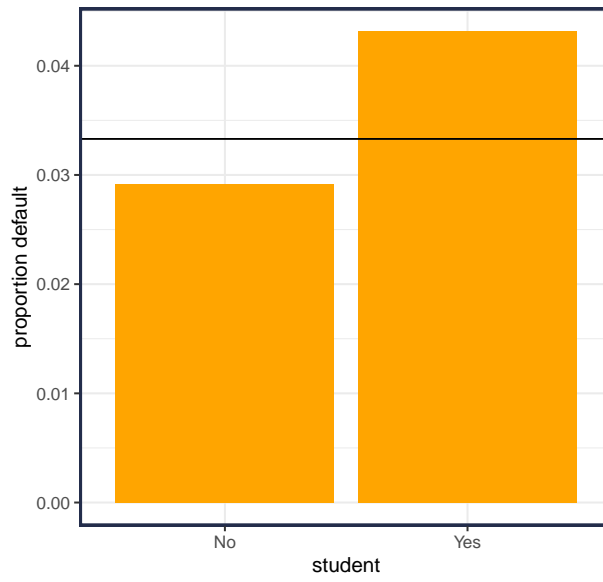
- *outcome variable* is categorical (factor) Yes and No, (default)
- the categorical (factor) variable (*student*) is either Yes or No
- the average balance a customer has after making their monthly payment (*balance*)
- the customer's income (*income*)

default	student	balance	income
No	No	396.5	41970
No	No	913.6	46907
No	Yes	561.4	21747
Yes	Yes	1889.3	22652
No	No	491.0	37836
No	Yes	282.2	19809

```
Default %>% summary
#>   default  student      balance      income
#>   No : 9667   No : 7056   Min.   :    0   Min.   :  772
#>   Yes:  333   Yes: 2944   1st Qu.:  482   1st Qu.:21340
#>                                     Median :  824   Median :34553
#>                                     Mean   :  835   Mean   :33517
#>                                     3rd Qu.:1166   3rd Qu.:43808
#>                                     Max.   :2654   Max.   :73554
```



Proportion of Defaults by Student status



**Your Turn #1 : Credit Card Default Modeling**

How would you construct a model to predict the risk of default?

## 1.2 Set-up

- The outcome variable is *categorical* and denoted  $G \in \mathcal{G}$ 
  - Default Credit Card Example:  $\mathcal{G} = \{\text{"Yes"}, \text{"No"}\}$
  - Medical Diagnosis Example:  $\mathcal{G} = \{\text{"stroke"}, \text{"heart attack"}, \text{"drug overdose"}, \text{"vertigo"}\}$
- The training data is  $D = \{(X_1, G_1), (X_2, G_2), \dots, (X_n, G_n)\}$
- The optimal decision/classification is often based on the posterior probability  $\Pr(G = g \mid \mathbf{X} = \mathbf{x})$

## 1.3 Binary Probability/Risk Modeling

- Modeling is simplified when there are only 2 classes.
  - Many multi-class problems can be addressed by solving a set of binary classification problems (e.g., [one-vs-rest](#)).
- It is often convenient to transform the outcome variable to a binary  $\{0, 1\}$  variable:

$$Y_i = \begin{cases} 1 & G_i = \mathcal{G}_1 \quad (\text{outcome of interest}) \\ 0 & G_i = \mathcal{G}_2 \end{cases}$$

- In the `Default` data, it would be natural to set `default=Yes` to 1 and `default=No` to 0.

### 1.3.1 Linear Regression

- In this set-up we can run linear regression

$$\hat{y}(\mathbf{x}) = \hat{\beta}_0 + \sum_{j=1}^p \hat{\beta}_j x_j$$

```
#: Create binary column (y)
Default = Default %>% mutate(y = if_else(default == "Yes", 1L, 0L))

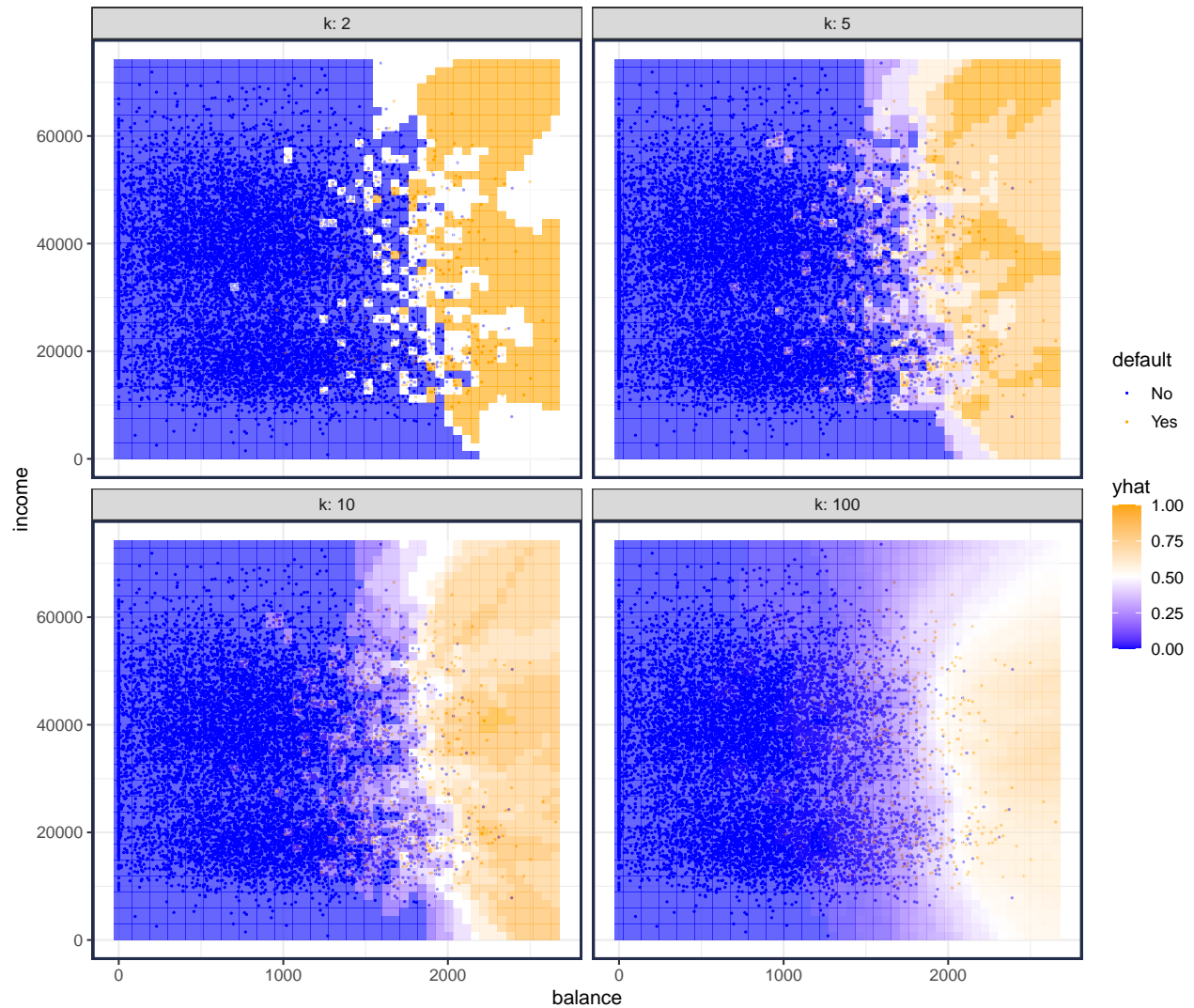
#: Fit Linear Regression Model
fit_lm = lm(y~student + balance + income, data = Default)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-0.08118	0.00838	-9.685	0.00000
studentYes	-0.01033	0.00566	-1.824	0.06817
balance	0.00013	0.00000	37.412	0.00000
income	0.00000	0.00000	1.039	0.29896

### Your Turn #2 : OLS for Binary Responses

1. For the binary  $Y$ , what is linear regression estimating?





### Your Turn #3 : Thoughts about kNN

The above plots show a kNN model using the *continuous* predictors of *balance* and *income*.

- How could you use kNN with the categorical *student* predictor?

- The  $k$ -NN model also has a more general description when the outcome variable is categorical  $G_i \in \mathcal{G}$

$$\begin{aligned}
 f_g^{\text{knn}}(x; k) &= \frac{1}{k} \sum_{i: x_i \in N_k(x)} \mathbb{1}(g_i = g) \\
 &= \widehat{\Pr}(G_i = g \mid x_i \in N_k(x))
 \end{aligned}$$

- $N_k(x)$  are the set of  $k$  nearest neighbors

- only the  $k$  closest  $y$ 's are used to generate a prediction
- it is a *simple proportion* of the  $k$  nearest observations that are of class  $g$

#### Your Turn #4 : kNN for multi-class outcomes

If using a categorical outcome, kNN models will output a *vector* of probabilities that sum to 1. For small  $k$  or if many categories, this vector may be *sparse* meaning that most entries are 0.

- How could we use concepts like Laplace smoothing to produce better predictions in these scenarios?



## 2 Logistic Regression

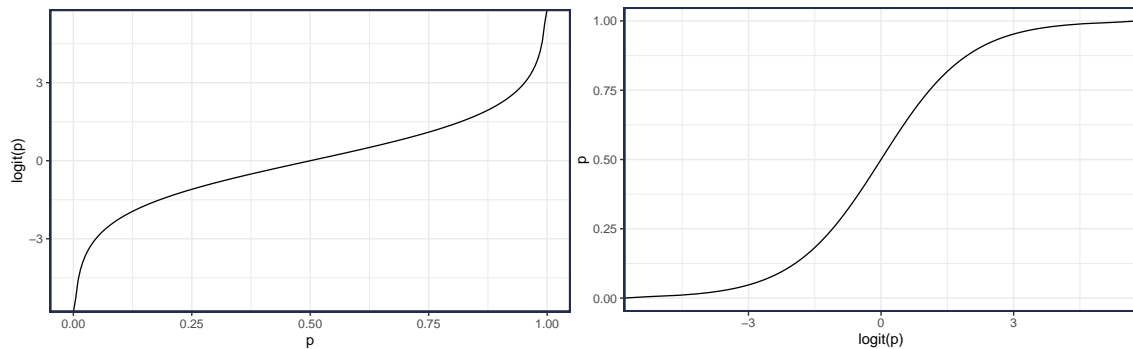
### 2.1 Basics

- Let  $0 \leq p \leq 1$  be a probability.
- The log-odds of  $p$  is called the *logit*

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

- The inverse logit is the *logistic function* (or *sigmoid function*). Let  $f = \text{logit}(p)$ , then

$$\begin{aligned} p &= \frac{e^f}{1 + e^f} \\ &= \frac{1}{1 + e^{-f}} \end{aligned}$$



Logistic Regression models have the form:

$$\text{logit} \Pr(Y = 1 | X = x) = \log\left(\frac{\Pr(Y = 1 | X = x)}{1 - \Pr(Y = 1 | X = x)}\right) = \beta^T x$$

and thus,

$$\begin{aligned} \Pr(Y = 1 | X = x) &= \frac{e^{\beta^T x}}{1 + e^{\beta^T x}} \\ &= \left(1 + e^{-\beta^T x}\right)^{-1} \end{aligned}$$

#### Note

For binary outcome variables  $Y \in \{0, 1\}$ , [Linear Regression](#) models

$$E[Y | X = x] = \Pr(Y = 1 | X = x) = \beta^T x$$

### 2.2 Estimation

- The input data for logistic regression are:  $(\mathbf{x}_i, y_i)_{i=1}^n$  where  $y_i \in \{0, 1\}$ ,  $\mathbf{x}_i = (x_{i0}, x_{i1}, \dots, x_{ip})^T$ .
- $y_i | \mathbf{x}_i \sim \text{Bern}(p_i(\beta))$

- $p_i(\beta) = \Pr(Y = 1 \mid \mathbf{X} = \mathbf{x}_i; \beta) = (1 + e^{-\beta^\top \mathbf{x}_i})^{-1}$
- where  $\beta^\top \mathbf{x}_i = \mathbf{x}_i^\top \beta = \beta_0 + \sum_{j=1}^p x_{ij} \beta_j$

- Bernoulli Likelihood Function

$$\begin{aligned} L(\beta) &= \prod_{i=1}^n p_i(\beta)^{y_i} (1 - p_i(\beta))^{1-y_i} \\ &= \sum_{i=1}^n \begin{cases} p_i(\beta) & y_i = 1 \\ 1 - p_i(\beta) & y_i = 0 \end{cases} \end{aligned}$$

$$\begin{aligned} \log L(\beta) &= \sum_{i=1}^n \{y_i \ln p_i(\beta) + (1 - y_i) \ln(1 - p_i(\beta))\} \\ &= \sum_{i=1}^n \begin{cases} \ln p_i(\beta) & y_i = 1 \\ \ln(1 - p_i(\beta)) & y_i = 0 \end{cases} \\ &= \sum_{i:y_i=1} \ln p_i(\beta) + \sum_{i:y_i=0} \ln(1 - p_i(\beta)) \end{aligned}$$

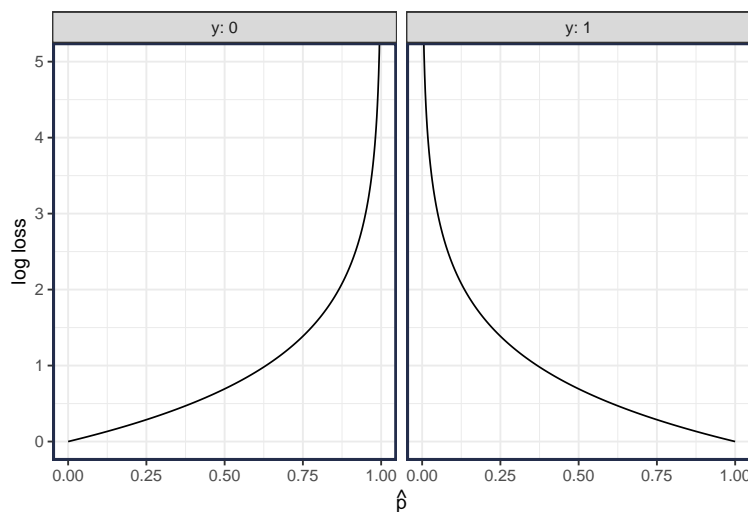
- The usual approach to estimating the Logistic Regression coefficients is *maximum likelihood*

$$\begin{aligned} \hat{\beta} &= \arg \max_{\beta} L(\beta) \\ &= \arg \max_{\beta} \log L(\beta) \end{aligned}$$

- We can also view this as the coefficients that minimize the *loss function*  $\ell(\beta)$ , where the loss function is the negative log-likelihood

$$\hat{\beta} = \arg \min_{\beta} \ell(\beta)$$

using loss  $\ell(\beta) = -C \log L(\beta)$  where  $C > 0$  is some positive constant, e.g.,  $C = 1/n$



## Log Loss

- The *log-loss* is the negative Bernoulli log-likelihood.
- The *cross-entropy* loss is also the negative Bernoulli log-likelihood.
- $-\log(p) = \log(1/p)$

- This view facilitates *penalized logistic regression*

$$\hat{\beta} = \arg \min_{\beta} \ell(\beta) + \lambda P(\beta)$$

Ridge Penalty	$P(\beta) = \ \beta\ _2^2 = \sum_{j=1}^p  \beta_j ^2 = \beta^\top \beta$
Lasso Penalty	$P(\beta) = \ \beta\ _1 = \sum_{j=1}^p  \beta_j $
Best Subsets	$P(\beta) = \ \beta\ _0 = \sum_{j=1}^p  \beta_j ^0 = \sum_{j=1}^p 1_{(\beta_j \neq 0)}$
Elastic Net	$P(\beta, \alpha) = (1 - \alpha) \ \beta\ _2^2 / 2 + \alpha \ \beta\ _1 = \sum_{j=1}^p (1 - \alpha)  \beta_j ^2 / 2 + \alpha  \beta_j $

## 2.3 Logistic Regression in Action

- In **R**, logistic regression can be implemented with the `glm()` function since it is a type of *Generalized Linear Model*.
- Because logistic regression is a special case of *Binomial* regression, use the `family=binomial()` argument

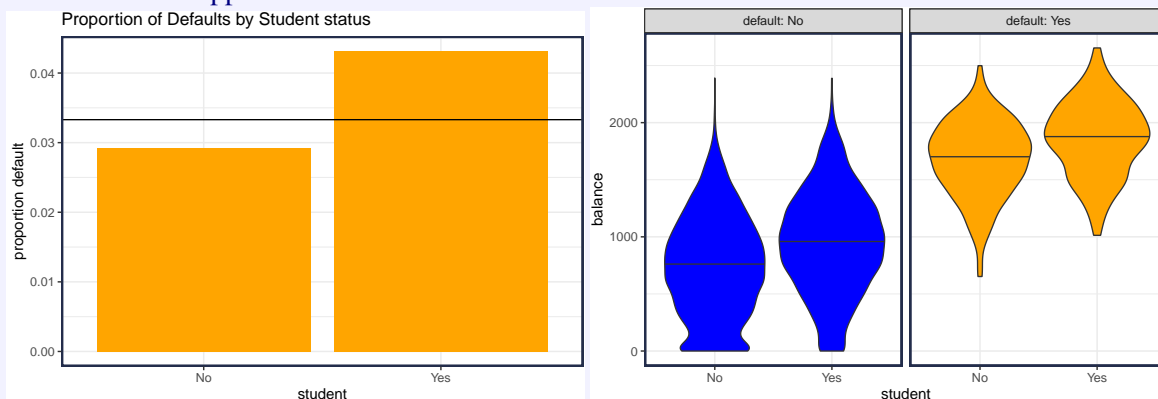
```
#: Fit logistic regression model
fit_lr = glm(y~student + balance + income,
            family = "binomial",
            data = Default)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-10.869	0.492	-22.080	0.000
studentYes	-0.647	0.236	-2.738	0.006
balance	0.006	0.000	24.738	0.000
income	0.000	0.000	0.370	0.712

## Your Turn #5 : Interpreting Logistic Regression

1. What is the estimated probability of default for a Student with a balance of \$1000?
2. What is the estimated probability of default for a *Non-Student* with a balance of \$1000?

3. Why does `student=Yes` appear to lower risk of default, when the plot of student status vs. default appears to increase risk?



### 2.3.1 Logistic vs. Linear Regression predictions

```
fit_logistic = glm(y~student + balance + income, family="binomial", data = Default)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-10.869	0.492	-22.080	0.000
studentYes	-0.647	0.236	-2.738	0.006
balance	0.006	0.000	24.738	0.000
income	0.000	0.000	0.370	0.712

```
fit_linear = lm(y~student + balance + income, data = Default)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-0.08118	0.00838	-9.685	0.00000
studentYes	-0.01033	0.00566	-1.824	0.06817
balance	0.00013	0.00000	37.412	0.00000
income	0.00000	0.00000	1.039	0.29896

Compare predictions:

student	balance	income	logistic_p	linear_p
Yes	1000	40000	0.003	0.049
No	1000	40000	0.007	0.059

### 2.3.2 Penalized Logistic Regression

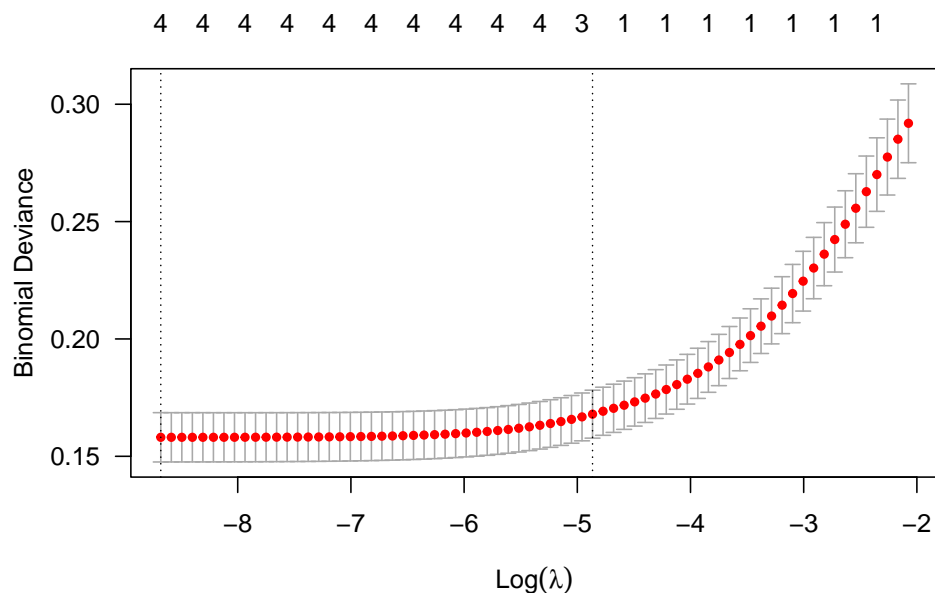
- The `glmnet()` package can estimate logistic regression using an elastic net penalty (e.g., ridge, lasso).

```

# Fit *penalized* logistic regression model
library(glmnet)
set.seed(2020)
X = makeX(Default %>% select(student, balance, income))
Y = Default$y
fit.enet = cv.glmnet(X, Y,
                    alpha=.5,
                    family="binomial")

# CV performance plot
plot(fit.enet, las=1)

```



term	unpenalized	lambda.min	lambda.1se
(Intercept)	-10.869	-11.056	-7.937
studentYes	-0.647	-0.299	-0.041
balance	0.006	0.006	0.004
income	0.000	0.000	0.000
studentNo	NA	0.325	0.044

### 2.4 Logistic Regression Summary

- Logistic Regression (both penalized and unpenalized) estimates a *posterior probability*,  $\hat{p}(x) = \widehat{\Pr}(Y = 1 | X = x)$

- This estimate is a function of the estimated coefficients

$$\begin{aligned}\hat{p}(x) &= \frac{e^{\hat{\beta}^\top x}}{1 + e^{\hat{\beta}^\top x}} \\ &= \left(1 + e^{-\hat{\beta}^\top x}\right)^{-1}\end{aligned}$$

### Your Turn #6

1. Given a person's student status, balance, and income, how could you use Logistic Regression to decide if they will default? (i.e., make a hard classification)

### 3 Evaluating Binary Risk Models

#### 3.1 Common Binary Loss Functions

- Suppose we are going to predict a binary outcome  $Y \in \{0, 1\}$  with  $0 \leq \hat{p}(x) \leq 1$ .

- Call  $\hat{p}(x)$  the *risk score*

- **Brier Score / Squared Error**

$$\begin{aligned} L(y, \hat{p}) &= (y - \hat{p})^2 \\ &= \begin{cases} (1 - \hat{p})^2 & y = 1 \\ \hat{p}^2 & y = 0 \end{cases} \end{aligned}$$

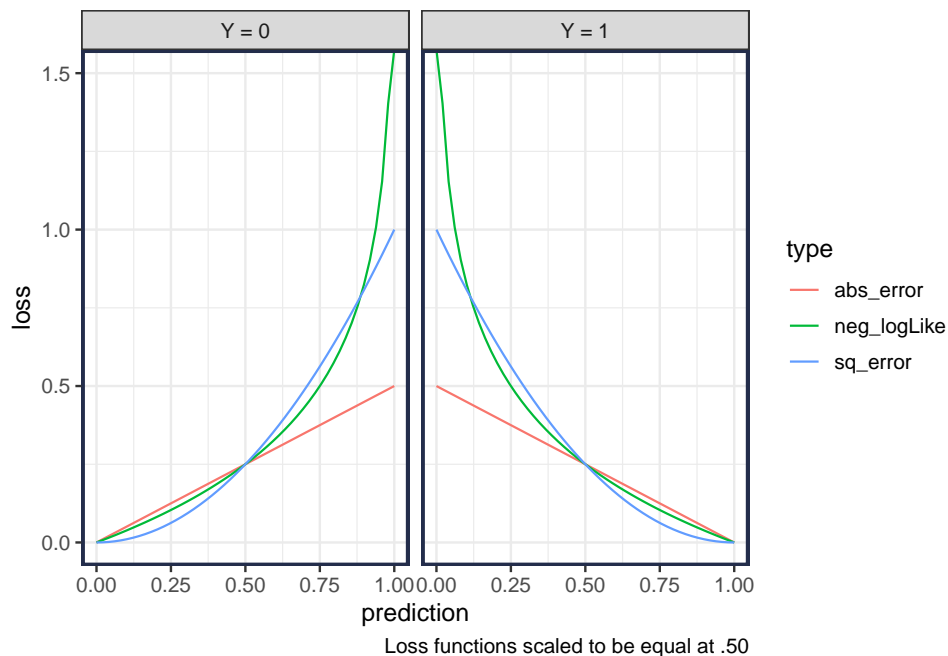
- **Absolute Error**

$$\begin{aligned} L(y, \hat{p}) &= |y - \hat{p}| \\ &= \begin{cases} 1 - \hat{p} & y = 1 \\ \hat{p} & y = 0 \end{cases} \end{aligned}$$

- **Bernoulli negative log-likelihood (Log-Loss / Cross-entropy)**

- This is the loss function used in Logistic Regression

$$\begin{aligned} L(y, \hat{p}) &= -\{y \log \hat{p} + (1 - y) \log(1 - \hat{p})\} \\ &= \begin{cases} -\log \hat{p} & y = 1 \\ -\log(1 - \hat{p}) & y = 0 \end{cases} \end{aligned}$$



### 3.2 Model Comparison

```

# : Evaluation Function
evaluate <- function(p_hat, y){
  tibble(
    mn_log_loss = -mean(dbinom(y, prob = p_hat, size=1, log=TRUE)),
    mse = mean((y-p_hat)^2),
    mae = mean(abs(y-p_hat))
  )
}

# : train/test split
set.seed(2019)
test = sample(nrow(Default), size=2000)
train = -test

# : fit logistic regression on training data
fit_lm = glm(y~student + balance + income, family='binomial',
  data=Default[train, ])
p_hat.lm = predict(fit_lm, Default[test,], type="response")
evaluate(p_hat.lm, y = Default$y[test])
#> # A tibble: 1 x 3
#>   mn_log_loss   mse   mae
#>   <dbl> <dbl> <dbl>
#> 1     0.0815 0.0228 0.0457

# : Fit lasso logistic regression (choose lambda with 10-fold cv)
X = glmnet::makeX(Default |> select(student, balance, income))
Y = Default$y
set.seed(2022)
fit_lasso = cv.glmnet(X[train,], Y[train], alpha = 1, family = "binomial")
p_hat.lasso = predict(fit_lasso, X[test,], type="response", s = "lambda.min")
evaluate(p_hat.lasso, y=Y[test])
#> # A tibble: 1 x 3
#>   mn_log_loss   mse   mae
#>   <dbl> <dbl> <dbl>
#> 1     0.0815 0.0228 0.0459

# : Fit ridge penalized linear regression (choose lambda with 10-fold cv)
set.seed(2022)
fit_linear_ridge = cv.glmnet(X[train,], Y[train],
  alpha = 0, family = "gaussian")
p_hat.linear_ridge = predict(fit_linear_ridge, X[test,], s = "lambda.min") %>%
  pmin(1) |> pmax(0) # force to [0,1]
evaluate(p_hat.linear_ridge, y=Y[test])
#> # A tibble: 1 x 3
#>   mn_log_loss   mse   mae
#>   <dbl> <dbl> <dbl>
#> 1     0.108 0.0280 0.0698

```

### 3.3 Area under the ROC curve (AUC or AUROC or C-Statistic)

The AUROC of a risk model is: the probability that the model will score a randomly chosen positive example ( $Y = 1$ ) higher than a randomly chosen negative example ( $Y = 0$ ), i.e.

$$\text{AUROC} = \Pr(\hat{p}(\tilde{X}_1) > \hat{p}(\tilde{X}_0))$$



where  $\tilde{X}_k$  is a randomly chosen example from class  $Y = k$  and  $\hat{p}(x) = \widehat{\Pr}(Y = 1 | X = x)$  is the estimated probability from a fitted model.

### 3.3.1 Naive AUC estimator

To *estimate* the AUROC you will fit a model to training data and make predictions on hold-out (test) data with known labels. Hopefully the model will assign large probabilities to the outcome of interest ( $Y = 1$ ) and low probabilities to the other class.

The *naive AUC estimator* compares the probabilities between all pairs of observations where one comes from the  $Y = 1$  set and the other from the  $Y = 0$  set. The estimated AUROC is the proportion of the pairs where the estimated probability for the outcome of interest is larger than the probability for the other outcome.

$$\widehat{\text{AUROC}} = \frac{1}{n_1 n_0} \sum_{i:y_i=1} \sum_{j:y_j=0} \mathbb{1}(\hat{p}_i > \hat{p}_j) + \frac{1}{2} \mathbb{1}(\hat{p}_i = \hat{p}_j)$$

- The extra term ( $\frac{1}{2} \mathbb{1}(\hat{p}_i = \hat{p}_j)$ ) is to handle ties in predicted probability.

- Note: see below for a more efficient way to estimate AUC using the ranked order of test predictions

### 3.3.2 AUC properties

- The AUROC assesses the *discrimination ability* of the model. It gives a different assessment on model performance from *calibration*.
- Notice that the AUROC is the same for any monotonic transformation of the estimated probabilities. E.g., we can use  $\hat{p}$  or  $\log(\hat{p})$  or  $\text{logit}(\hat{p})$  or  $\hat{p}/10$  and still get the same AUROC.
- We will discuss the Receiver Operating Curves (ROC) during Classification: Decision Theory lesson.
- Note: calibration assesses how closely the estimated probabilities match the actual probabilities as well as helping to identify the regions in feature space where the predictions are poor.

#### Calculating AUC from Ranked Sums

The AUC is related to the test statistic from a Wilcoxon rank-sum test (also known as the Mann-Whitney U test). Steps:

1. Rank all predictions  $\{\hat{p}(x_i)\}$  from *smallest to largest*.
2. Calculate  $R_1$ , the sum of the ranks for the observations correspond to the outcome of interest ( $Y = 1$ ). Hopefully, the sum of the ranks is large.
3. Estimate the AUC as

$$\widehat{\text{AUC}} = \frac{R_1 - n_1(n_1 + 1)/2}{n_1 n_0}$$

This calculation makes it clear that AUC may be better viewed as a ranking metric rather than a probability assessment.

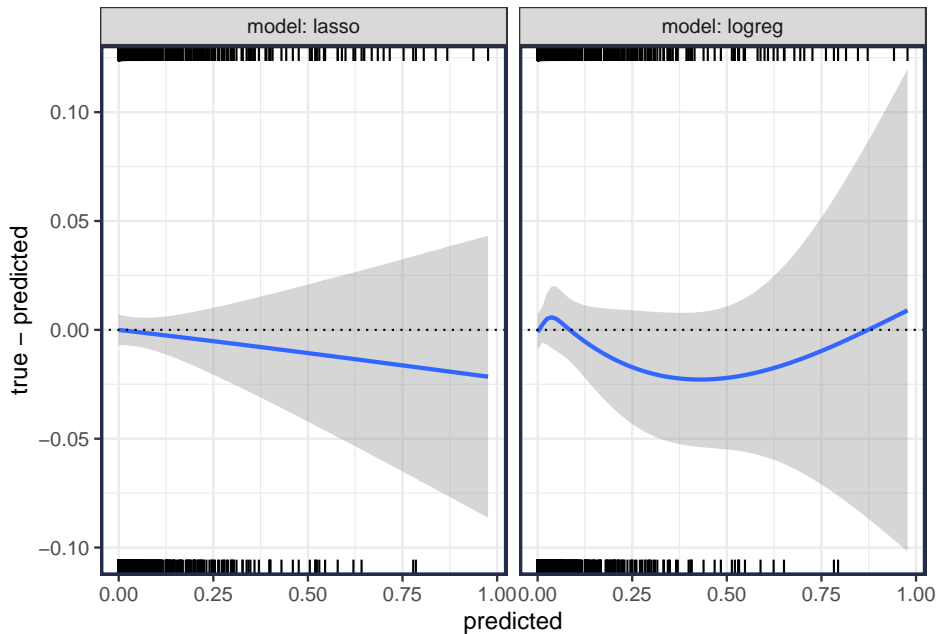
- If your problem is not fundamentally based on *ranking a set of unlabeled observations*, then AUC may not be the best metric for the problem.

A helpful discussion on AUROC (including calculation): <https://stats.stackexchange.com/questions/145566/how-to-calculate-area-under-the-curve-auc-or-the-c-statistic-by-hand>

### 3.4 Calibration

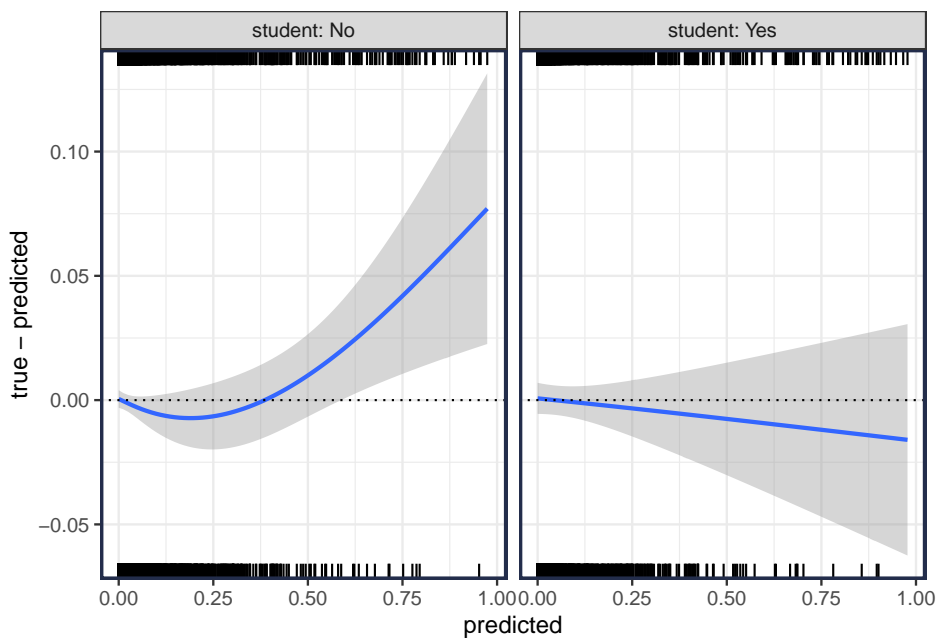
A risk model is said to be calibrated if the *predicted* probabilities are equal to the *true* probabilities.

$$\Pr(Y = 1 \mid \hat{p} = p) = p \quad \text{for all } p$$



Calibration plots can be used to measure drift, fairness, and model/algorithmic bias. Consider comparing the predictive performance of our models for Students and Non-Students.

$$\Pr(Y = 1 \mid \hat{p} = p, X = x) = p \quad \text{for all } p \text{ and } x$$



### 3.4.1 Estimating Calibration

To measure mis-calibration, we can treat the predictions as features and use the predictions as an offset. E.g., to check for linear deviation

$$\text{logit } p(x) = \beta_0 + \beta_1 \hat{p}(x) + \text{logit } \hat{p}(x)$$

fit on a hold-out set, and check how far  $\beta_0$  and  $\beta_1$  are from 0.

We will dive into calibration later in the course.

## 4 Appendix: R Code

### Set-up

```

#: Load Required Packages
library(ISLR)
library(FNN)
library(broom)
library(yardstick)
library(tidyverse)

#-----#
#: Default Data
# From the ISLR package
# The outcome variable is `default`
#-----#
library(ISLR)
data(Default, package="ISLR") # load the Default Data

#: Create binary column (y)
Default = Default %>% mutate(y = ifelse(default == "Yes", 1L, 0L))

#: Summary Stats (Notice only 333 (3.3%) have defaulted)
summary(Default)
#> default student balance income y
#> No :9667 No :7056 Min. : 0 Min. : 772 Min. :0.0000
#> Yes: 333 Yes:2944 1st Qu.: 482 1st Qu.:21340 1st Qu.:0.0000
#> Median : 824 Median :34553 Median :0.0000
#> Mean : 835 Mean :33517 Mean :0.0333
#> 3rd Qu.:1166 3rd Qu.:43808 3rd Qu.:0.0000
#> Max. :2654 Max. :73554 Max. :1.0000

#: Plots
plot_cols = c(Yes="orange", No="blue") # set colors

Default %>%
  group_by(default) %>%
  slice(1:3000) %>% # choose max of 3000 from each group
  ggplot(aes(balance, income, color=default, shape=default)) +
  geom_point(alpha=.5) +
  scale_color_manual(values=plot_cols) +
  scale_shape_manual(values=c(Yes=19, No=1))

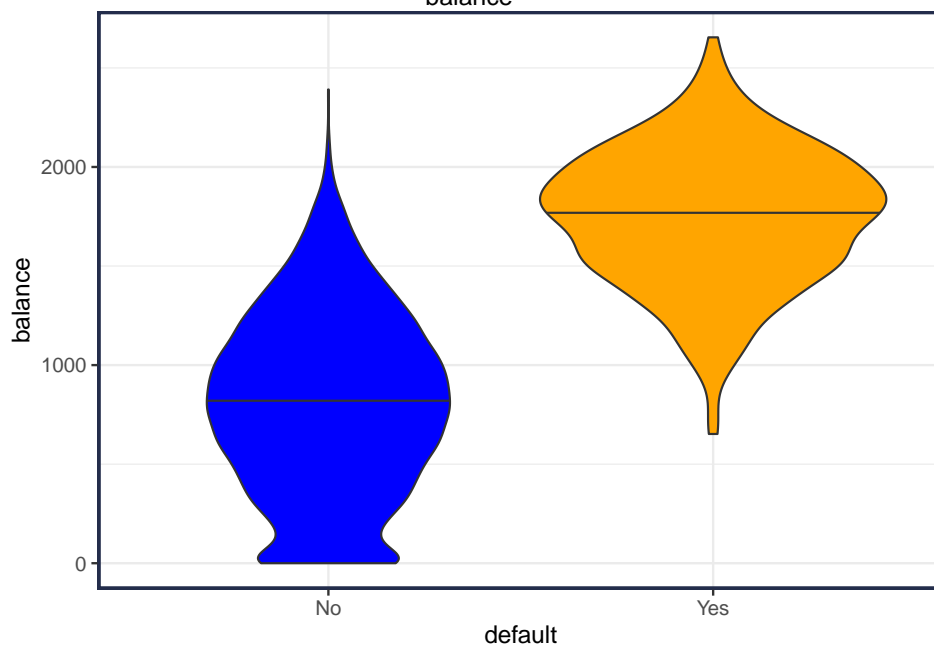
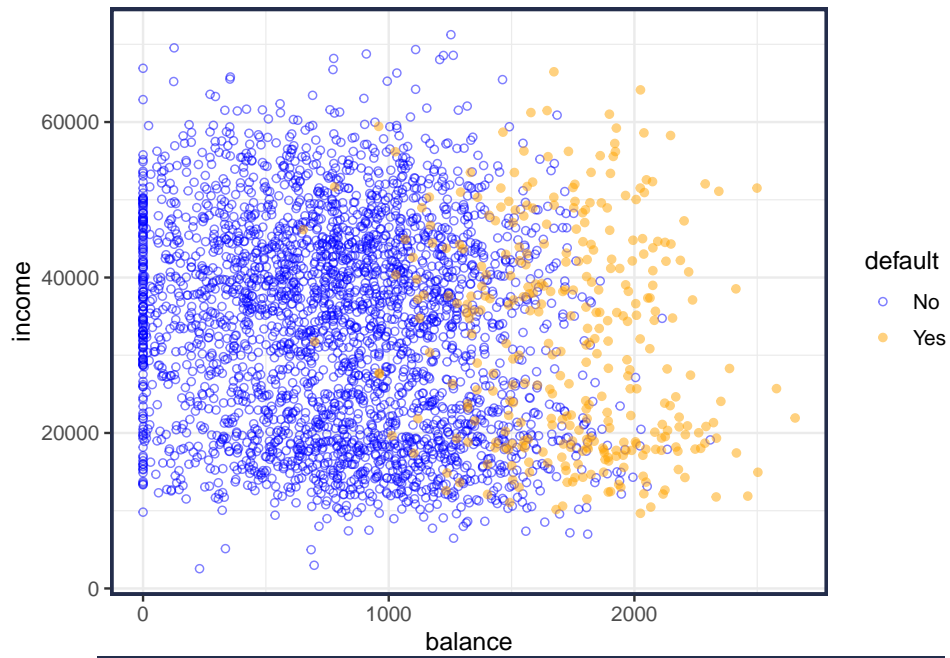
ggplot(Default, aes(default, balance, fill=default)) +
  geom_violin(draw_quantiles=.5) + #alternative: geom_boxplot() +
  scale_fill_manual(values=plot_cols, guide="none")

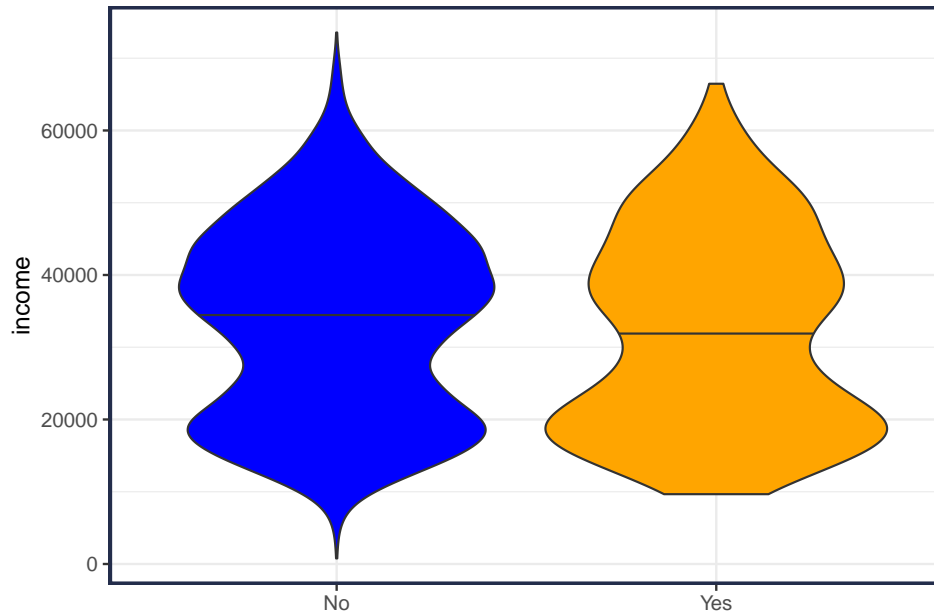
ggplot(Default, aes(default, income, fill=default)) +
  geom_violin(draw_quantiles=.5) +
  scale_fill_manual(values=plot_cols, guide="none")

count(Default, default, student) %>%
  group_by(student) %>% mutate(p=n/sum(n)) %>%
  filter(default == "Yes") %>%
  ggplot(aes(student, p, fill=default)) +
  geom_col() +
  geom_hline(yintercept=mean(Default$default == "Yes")) +
  scale_fill_manual(values=plot_cols, guide="none") +
  labs(title="Proportion of Defaults by Student status",

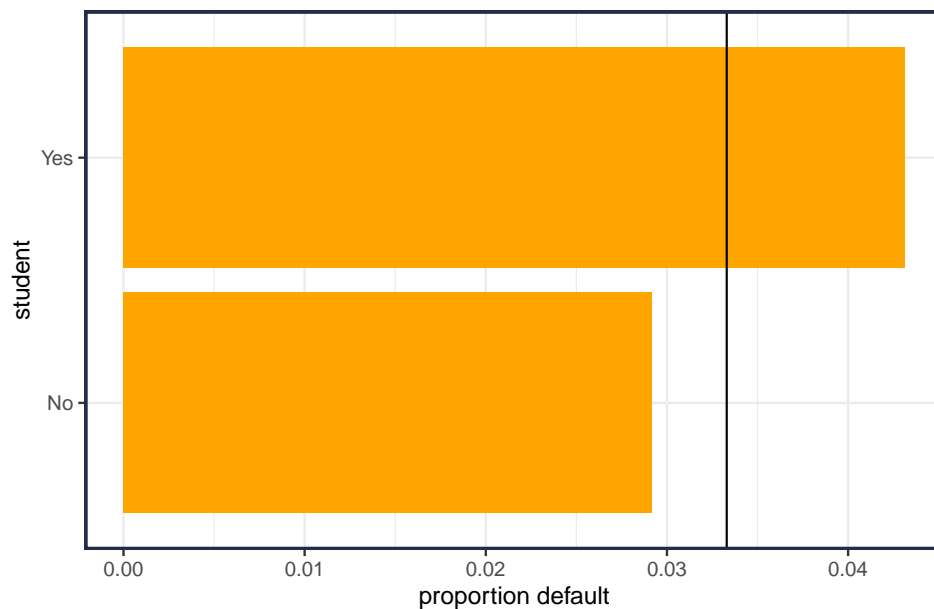
```

```
y="proportion default") +  
coord_flip()
```





Proportion of Defaults by Student status



## Linear Regression (for binary response)

```
library(broom) # to extract good stuff from models

# Fit Linear Regression Model
fit_lm = lm(y~student + balance + income, data=Default)

# Extract coefficients
coef(fit_lm) # generic coef function to get coefficients
#> (Intercept) studentYes balance income
#> -8.118e-02 -1.033e-02 1.327e-04 1.992e-07
broom::tidy(fit_lm) # tidy way to get coefficients
#> # A tibble: 4 x 5
```

```
#>   term          estimate  std.error statistic  p.value
#>   <chr>          <dbl>    <dbl>    <dbl>    <dbl>
#> 1 (Intercept) -0.0812    0.00838   -9.68 4.37e- 22
#> 2 studentYes  -0.0103    0.00566   -1.82 6.82e-  2
#> 3 balance      0.000133   0.00000355  37.4 8.04e-287
#> 4 income       0.000000199 0.000000192   1.04 2.99e-  1
```

## k nearest neighbor

```
library(FNN)           # for knn.reg() function
library(tidymodels)    # for recipe functions

# : center and scale predictors so Euclidean distance makes more sense
library(tidymodels)
pre_process =
  recipe(y ~ balance + income, data = Default) %>% # specify formula
  step_normalize(all_predictors()) %>%             # center and scale
  prep()                                           # estimate means and sds

# : apply the transformation to the predictors
X.scale = bake(pre_process, Default, all_predictors())
Y = bake(pre_process, Default, all_outcomes())$y
# Y = Default$y

# : Evaluation Points
eval.pts = expand_grid(
  balance = seq(min(Default$balance), max(Default$balance), length=50),
  income = seq(min(Default$income), max(Default$income), length=50)
)

X.eval = bake(pre_process, eval.pts) # scale eval pts too
# Note: this uses the same center and scale from the *training data*.
# This is important!
# Don't rescale the hold-out data separately!

# : fit knn model
knn5 = FNN::knn.reg(X.scale, y=Y, test=X.eval, k=5)
# Note: I'm using knn.reg() using the binary coded Y.
# we could use the knn() function (suitable for categorical outcomes),
# but to get at the probabilities isn't straightforward. They give the
# same solutions for two-class problems, so it doesn't matter.
```

## Logistic Regression

```
# More details in ISL_v2 4.7.2

# : Fit logistic regression model
fit_lr = glm(y ~ student + balance + income,
  family = "binomial", # this make it logistic regression
  data = Default)

## Alternatively, you can replace the binary y with a factor/character column
# fit_lr = glm(default~student + balance + income, data=Default,
# family="binomial")
```

```
#: Get coefficients
tidy(fit_lr)
#> # A tibble: 4 x 5
#>   term          estimate std.error statistic  p.value
#>   <chr>          <dbl>     <dbl>    <dbl>    <dbl>
#> 1 (Intercept) -10.9      0.492    -22.1  4.91e-108
#> 2 studentYes   -0.647     0.236     -2.74  6.19e- 3
#> 3 balance      0.00574   0.000232  24.7   4.22e-135
#> 4 income      0.00000303 0.00000820  0.370  7.12e- 1

#: Get predictions (for training data)
prob.lr = predict(fit_lr, type="response") # probabilities
link.lr = predict(fit_lr, type="link")    # logit (linear part)

#: Interpret
# Notice that Student=Yes has a negative coefficient, but the plot of
# defaults by student status suggests otherwise.
# Reason is because students have more balance on average than non-students,
# and they get over-estimated once balance is in the model

plot_cols = c(Yes="orange", No="blue") # set colors

ggplot(Default, aes(balance, fill=student)) +
  geom_density(alpha=.75) +
  facet_wrap(~default, labeller=label_both) +
  scale_fill_manual(values=plot_cols)

ggplot(Default, aes(student, balance, fill=student)) +
  geom_violin(draw_quantiles = .5) +
  facet_wrap(~default, labeller=label_both) +
  scale_fill_manual(values=plot_cols)

#: probability at certain values
eval.pts = tibble(
  student = c("Yes", "No"),
  balance = c(1000, 1000), # balance = 1000
  income = c(40000, 40000) # income set to 40K
)

predict(fit_lr, eval.pts, type="link")
#>   1      2
#> -5.658 -5.011
predict(fit_lr, eval.pts, type="response")
#>   1      2
#> 0.003477 0.006619

#: Simpson's Paradox

# Students have higher default rate
Default %>%
  group_by(student) %>% summarize(n=n(), p_default = mean(default == 'Yes'))
#> # A tibble: 2 x 3
#>   student      n p_default
#>   <fct>    <int>    <dbl>
#> 1 No       7056    0.0292
#> 2 Yes     2944    0.0431
```



```

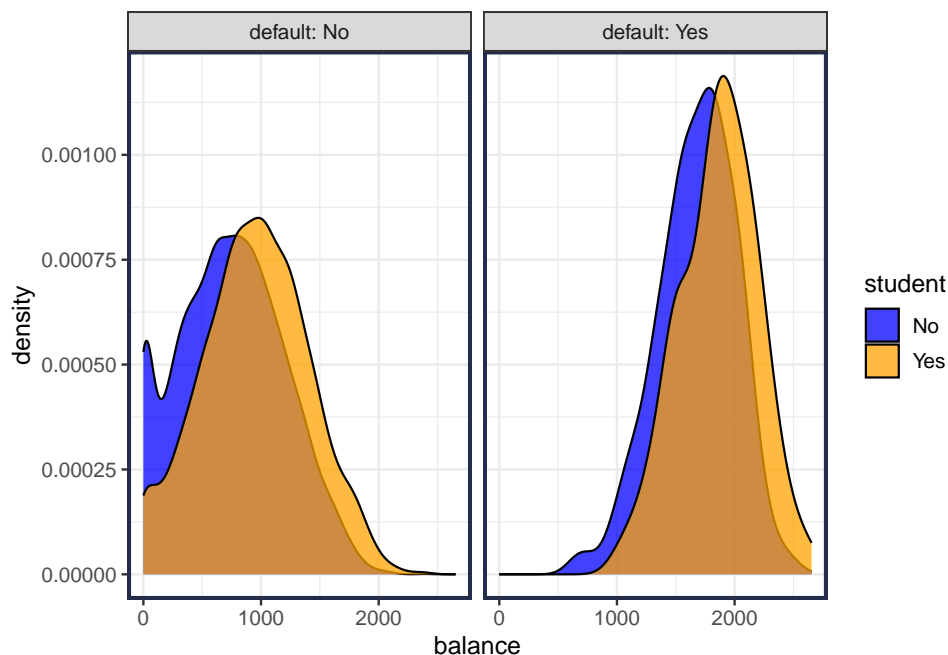
# People with higher balances have higher default rate
ggplot(Default, aes(balance, default, fill=default)) +
  geom_violin(draw_quantiles=.5) + #alternative: geom_boxplot() +
  scale_fill_manual(values=plot_cols, guide = "none")

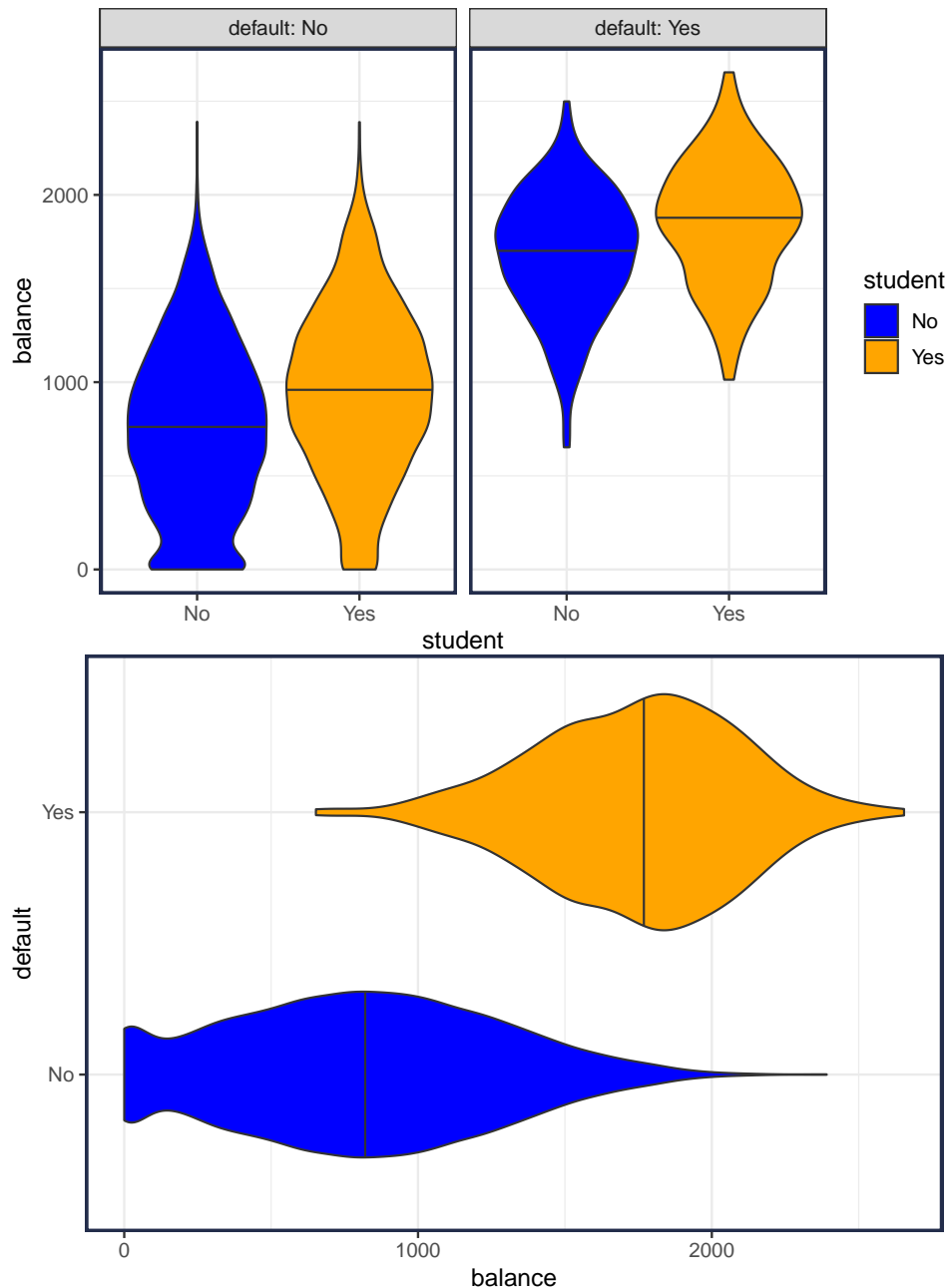
Default %>%
  group_by(default) %>% summarize(avg_balance = mean(balance))
#> # A tibble: 2 x 2
#>   default avg_balance
#>   <fct>    <dbl>
#> 1 No      804.
#> 2 Yes    1748.

# Students have higher balances on average, so they appear to be more likely
# to default if the balance is not taken into account
Default %>%
  group_by(student) %>% summarize(avg_balance = mean(balance))
#> # A tibble: 2 x 2
#>   student avg_balance
#>   <fct>    <dbl>
#> 1 No      772.
#> 2 Yes    988.

# This is why the logistic regression model correctly adjusts the student status
# negative.
Default %>%
  mutate(p_hat = prob.lr) %>%
  group_by(student) %>%
  summarize(n=n(), default_rate = mean(default == 'Yes'), avg_p = mean(p_hat))
#> # A tibble: 2 x 4
#>   student    n default_rate avg_p
#>   <fct> <int>    <dbl> <dbl>
#> 1 No     7056    0.0292 0.0292
#> 2 Yes   2944    0.0431 0.0431

```





## Convert data frame to model matrix

The `glmnet` package only handles model matrix and not data frames, so we have to convert the data into model matrix. When all predictors are numeric, this is easy (e.g., `data.matrix()` or `model.matrix()` if formula), but categorical/factor data needs to be handled separately and consistently if there are multiple data sets (e.g., train and test).

Here are a few options:

```
#: Create model formula (or vector of column names)  
fmla = as.formula(y ~ student + balance + income)  
vars = fmla %>% terms() %>% labels()
```

## tidymodels recipe package

tidymodels are a collection of useful modeling packages (like tidyverse). The main package to pre-process, transform, and prepare a data frame for passing into a modeling function is `recipe`.

- `recipe()` sets the variables: predictor, outcome, case\_weight, or ID. And optionally provides the training data.
- `prep()` performs (i.e., fits) the pre-processing/transformations using the training data.
- `bake()` applies the pre-processing to new data

```
#-----#
#: Option 1: using tidymodels()
#-----#
library(tidymodels)

rec = recipe(fmla, data = Default) %>% # sets the variable roles
  step_dummy(all_nominal(), one_hot = TRUE) %>% # one-hot encoding
  prep() # "fits" using Default data.

# to get the "matrix", set `composition = "matrix"`
X.train = bake(rec, new_data = Default, all_predictors(), composition="matrix")

# Convert the outcome variable to a vector with `pull()`
Y.train = bake(rec, new_data = Default, all_outcomes()) %>% pull()
# Now any new data, like test data, can be obtained by changing the new_data argument
# X.test = bake(rec, new_data = test, all_predictors(), composition="matrix")
```

## glmnet::makeX()

The `glmnet` package provides a basic function `makeX()` to properly create training and testing model matrices.

```
#-----#
#: Option 2: using glmnet::makeX()
#-----#
X.train = glmnet::makeX(Default |> select(any_of(vars)))

# if test data is available, pass it in at same time
# X = glmnet::makeX(
#   train = Default |> select(-y),
#   test = train |> select(-y)
# )
# X.train = X$x
# X.test = X$xtest
```

## Other approaches

```
#-----#
#: Option 3: Manually using dplyr::select() + as.matrix()
#-----#
# Note: must manually transform, dummy, interactions, etc.
X.train = select(Default, !!vars) %>%
  mutate(dummy = 1) %>%
  pivot_wider(
    names_from = student,
    values_from = dummy,
```

```

  values_fill = 0L
) %>%
as.matrix() # make X matrix
Y.train = Default$y # make Y vector
# X.test = select(test, !!vars) %>%
# mutate(dummy=1) %>% pivot_wider(names_from = student, values_from = dummy, values_fill = 0L) %>%
# as.matrix()

#-----#
#: Option 4: using model.matrix()
#-----#
X.train = model.matrix(fmla, data = Default)[,-1] # remove intercept term
Y.train = Default$y
# X.test = model.matrix(fmla, data = test)[,-1] # remove intercept term

```

## Penalized Logistic Regression

```

library(tidymodels)
rec = recipe(y~student + balance + income, data = Default) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  prep()
X.train = bake(rec, new_data = Default, all_predictors(), composition="matrix")
Y.train = bake(rec, new_data = Default, all_outcomes()) %>% pull()
X.eval = bake(rec, new_data = eval.pts, all_predictors(), composition="matrix")
#> Warning: ! There was 1 column that was a factor when the recipe was prepped:
#> * `student`
#> | This may cause errors when processing new data.

# library(glmnet)
# vars = c("student", "balance", "income")
# X = glmnet::makeX(select(Default, !!vars), select(eval.pts, !!vars))
# X.train = X$x
# Y.train = Default$y
# X.eval = X$xtest

#: Elastic net with alpha = .5. Use CV to select lambda.
library(glmnet)
set.seed(2020)
fit_enet = cv.glmnet(X.train, Y.train,
                    alpha=.5,
                    family="binomial")

#: CV performance plot
plot(fit_enet, las=1)

#: probability at certain values
predict(fit_enet, X.eval, s="lambda.min", type="response")
#>      lambda.min
#> [1,] 0.003722
#> [2,] 0.006929
predict(fit_enet, X.eval, s="lambda.1se", type="response")
#>      lambda.1se
#> [1,] 0.01444
#> [2,] 0.01569

#: Compare with intercept only model. Set large penalty (s large)

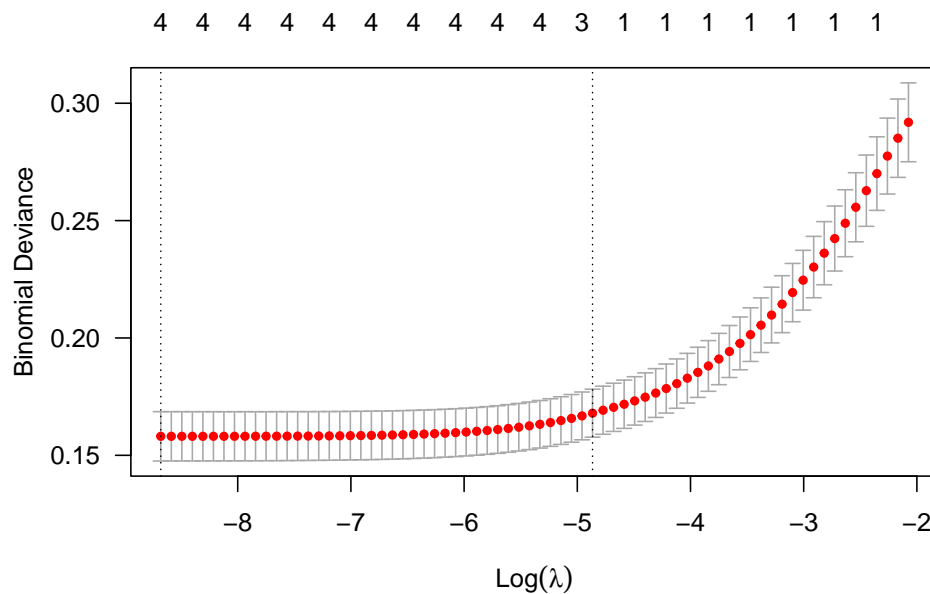
```

```

predict(fit_enet, X.eval, s=1000, type="response") # intercept only (effectively)
#>      s1
#> [1,] 0.0333
#> [2,] 0.0333
mean(Default$y) # actual intercept only
#> [1] 0.0333

#: Compare with unpenalized logistic regression. Set penalty s=0.
predict(fit_enet, X.eval, s=0, type="response") # unpenalized (effectively)
#>      s1
#> [1,] 0.003722
#> [2,] 0.006929

```



## Performance Metrics

```

#: train/test split
set.seed(2019)
test = sample(nrow(Default), size=1000)
train = -test

#: Elastic net with alpha = .5. Use CV to select lambda.
library(glmnet)
set.seed(2020)
fit_enet = cv.glmnet(X.train, Y.train,
                    alpha=.5,
                    family="binomial")
X.test = bake(rec, all_predictors(),
              new_data = Default[test,], composition="matrix")

#: Fit logistic regression model
fit_logr = glm(y ~ student + balance + income, data=Default,
              family="binomial")

```

The yardstick package is part of tidymodels and deals with predictive evaluation metrics.

```
library(yardstick)

# create evaluation dataframe
data_eval =
  tibble(
    default = Default %>% slice(test) %>% pull(default),
    p_enet = predict(fit_enet, X.test, s="lambda.min", type = "response")[,1],
    p_logr = predict(fit_logr, Default[test,], type = "response"),
  ) %>%
  pivot_longer(-default, names_to = "model", values_to = "p_hat")

# set threshold at p_hat = 0.50 and calculate metrics
data_eval |>
  mutate(G_hat = ifelse(p_hat > .50, "Yes", "No") %>% factor) %>%
  group_by(model) %>%
  metrics(default, G_hat, p_hat)

#> # A tibble: 8 x 4
#>   model .metric .estimator .estimate
#>   <chr> <chr> <chr> <dbl>
#> 1 p_enet accuracy binary 0.97
#> 2 p_logr accuracy binary 0.97
#> 3 p_enet kap binary 0.431
#> 4 p_logr kap binary 0.431
#> 5 p_enet mn_log_loss binary 6.08
#> 6 p_logr mn_log_loss binary 6.15
#> # i 2 more rows
```