Logistic Regression Models

Justin Post

Logistic Regression Model

Used when you have a **binary** response variable (a Classification task)

- Consider just a binary response
    - What is the mean of the response?

Logistic Regression Model

Suppose you have a predictor variable as well, call it x

- Given two values of x we could model separate proportions

$$E(Y|x = x_1) = P(Y = 1|x = x_1)$$

$$E(Y|x = x_2) = P(Y = 1|x = x_2)$$

Logistic Regression Model

Suppose you have a predictor variable as well, call it x

- Given two values of x we could model separate proportions

$$E(Y|x = x_1) = P(Y = 1|x = x_1)$$

$$E(Y|x = x_2) = P(Y = 1|x = x_2)$$

- For a continuous x, we could consider a SLR model

$$E(Y|x) = P(Y = 1|x) = \beta_0 + \beta_1 x$$

Linear Regression Isn't Appropriate

- Consider data about heart disease

```
library(tidyverse)
heart_data <- read_csv("https://www4.stat.ncsu.edu/online/datasets/heart.csv") |>
  filter(RestingBP > 0) #remove one value
heart_data |> select(HeartDisease, everything()) #Cholesterol has many values set to 0 so we ignore that
```

```
## # A tibble: 917 x 12
##   HeartDisease   Age Sex   ChestPainType RestingBP Cholesterol FastingBS
##          <dbl> <dbl> <chr> <chr>             <dbl>       <dbl>     <dbl>
## 1            0    40 M     ATA                 140         289         0
## 2            1    49 F     NAP                 160         180         0
## 3            0    37 M     ATA                 130         283         0
## 4            1    48 F     ASY                 138         214         0
## 5            0    54 M     NAP                 150         195         0
## # i 912 more rows
## # i 5 more variables: RestingECG <chr>, MaxHR <dbl>, ExerciseAngina <chr>,
## #   Oldpeak <dbl>, ST_Slope <chr>
```

Potability Summary

- Summarize heart disease prevalence

```
heart_data |>
  group_by(HeartDisease) |>
  summarize(count = n())
```

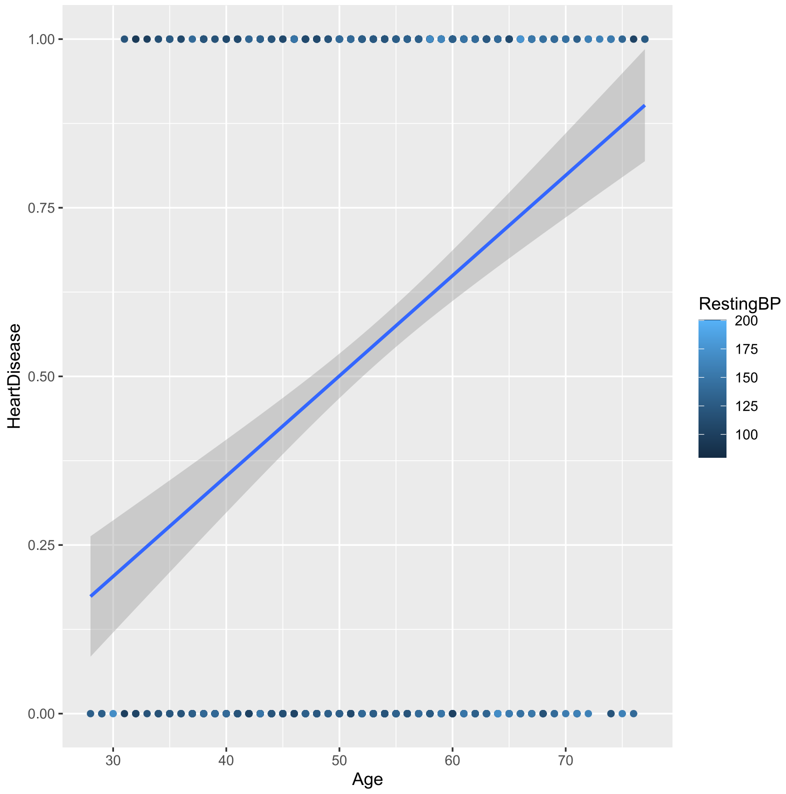
```
## # A tibble: 2 x 2
##   HeartDisease count
##          <dbl> <int>
## 1            0   410
## 2            1   507
```

```
heart_data |>
  group_by(HeartDisease) |>
  summarize(mean_Age = mean(Age),
            mean_RestingBP  = mean(RestingBP))
```

```
## # A tibble: 2 x 3
##   HeartDisease mean_Age mean_RestingBP
##          <dbl>    <dbl>          <dbl>
## 1            0     50.6           130.
## 2            1     55.9           134.
```

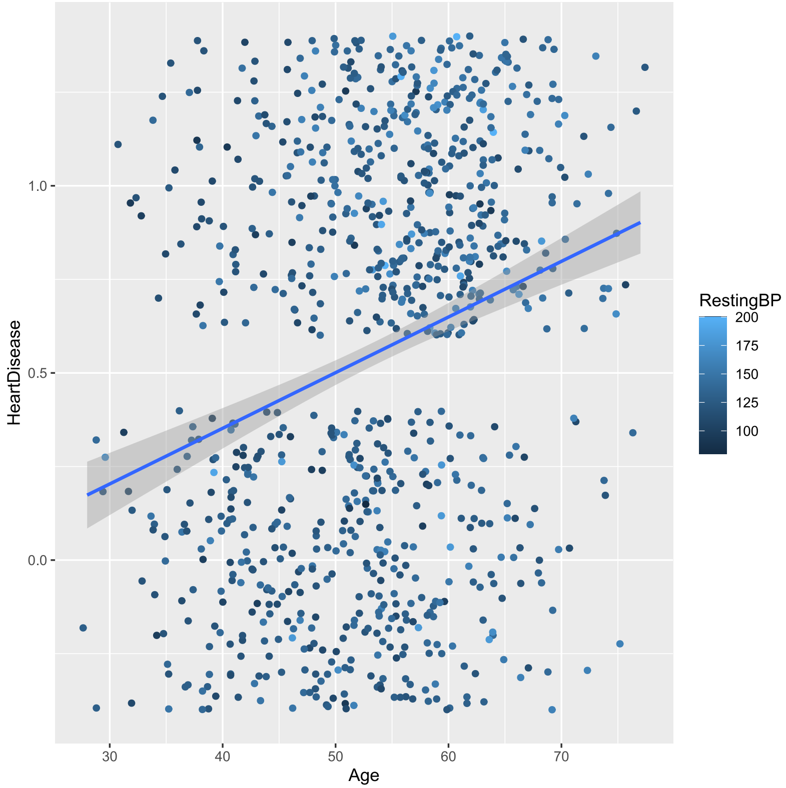
Linear Regression Isn't Appropriate

```
ggplot(heart_data, aes(x = Age, y = HeartDisease, color = RestingBP)) +
          geom_point() +
  geom_smooth(method = "lm")
```



Linear Regression Isn't Appropriate

```
ggplot(heart_data, aes(x = Age, y = HeartDisease, color = RestingBP)) +
          geom_jitter() +
  geom_smooth(method = "lm")
```



Linear Regression Isn't Appropriate

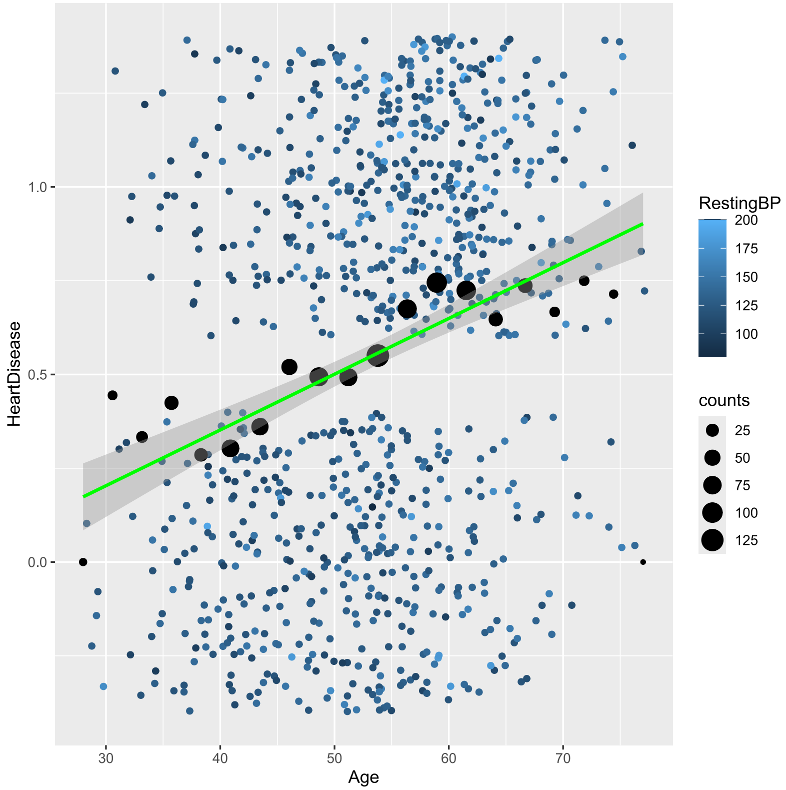
Obtain proportion with heart disease for different age groups

```
Age_x <- seq(from = min(heart_data$Age), to = max(heart_data$Age), length = 20)
heart_data_grouped <- heart_data |>
  mutate(Age_groups = cut(Age, breaks = Age_x)) |>
  group_by(Age_groups) |>
  summarize(HeartDisease_mean = mean(HeartDisease), counts = n())
heart_data_grouped
```

```
## # A tibble: 20 x 3
##    Age_groups  HeartDisease_mean counts
##    <fct>                   <dbl>  <int>
##  1 (28,30.6]               0          4
##  2 (30.6,33.2]             0.444      9
##  3 (33.2,35.7]             0.333     18
##  4 (35.7,38.3]             0.424     33
##  5 (38.3,40.9]             0.286     28
##  6 (40.9,43.5]             0.303     66
##  7 (43.5,46.1]             0.361     61
##  8 (46.1,48.6]             0.52      50
##  9 (48.6,51.2]             0.494     81
## 10 (51.2,53.8]             0.493     69
## 11 (53.8,56.4]             0.550    129
## 12 (56.4,58.9]             0.675     80
## 13 (58.9,61.5]             0.745     98
## 14 (61.5,64.1]             0.724     87
## 15 (64.1,66.7]             0.647     34
```

Linear Regression Isn't Appropriate

```r
ggplot(data = heart_data, aes(x = Age, y = HeartDisease)) +
  geom_jitter(aes(color = RestingBP)) +
  geom_point(data = heart_data_grouped, aes(x = Age_x, y = HeartDisease_mean, size = counts)) +
  geom_smooth(method = "lm", color = "Green")
```



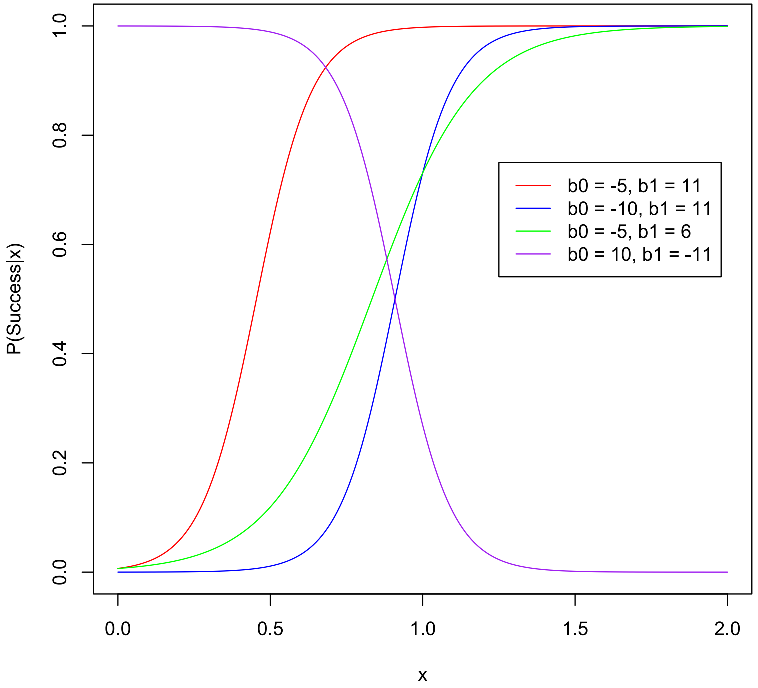
Logistic Regression

- Response = success/failure, then modeling average number of successes for a given x is a probability!
    - predictions should never go below 0
    - predictions should never go above 1
- Basic Logistic Regression models success probability using the *logistic function*

$$P(Y=1|x) = P(success|x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

Logistic Regression

$$P(Y=1|x) = P(success|x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$



Logistic Regression

$$P(Y=1|x) = P(success|x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

- The logistic regression model doesn't have a closed form solution (maximum likelihood often used to fit parameters)

Logistic Regression

$$P(Y = 1|x) = P(success|x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

- The logistic regression model doesn't have a closed form solution (maximum likelihood often used to fit parameters)
- Back-solving shows the *logit* or *log-odds* of success is linear in the parameters

$$log\left(\frac{P(success|x)}{1 - P(success|x)}\right) = \beta_0 + \beta_1 x$$

Logistic Regression

$$P(Y=1|x) = P(success|x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

- The logistic regression model doesn't have a closed form solution (maximum likelihood often used to fit parameters)

- Back-solving shows the *logit* or *log-odds* of success is linear in the parameters

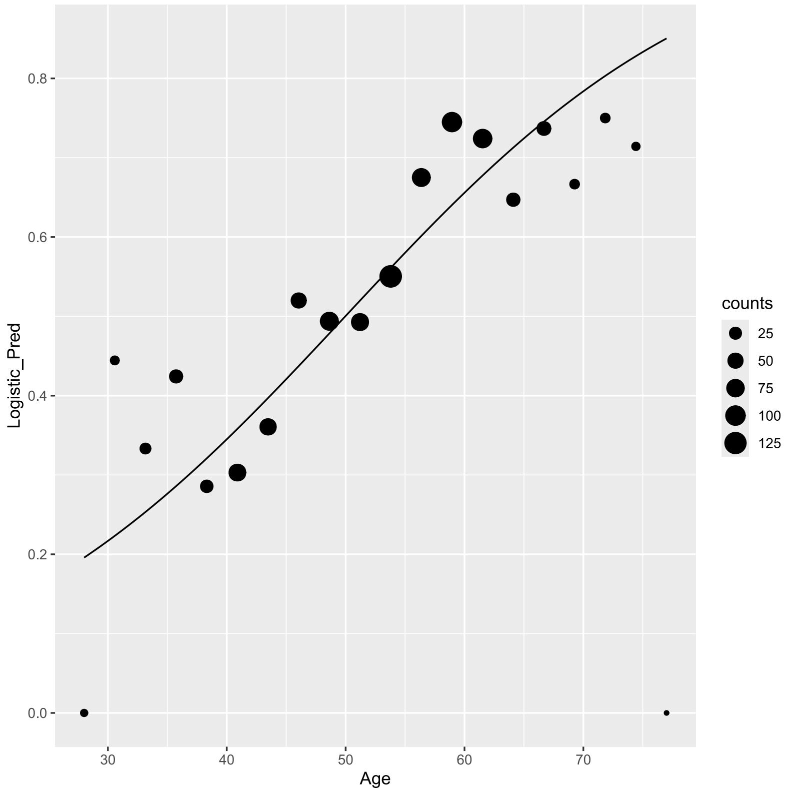
$$log\left(\frac{P(success|x)}{1-P(success|x)}\right) = \beta_0 + \beta_1 x$$

- Coefficient interpretation changes greatly from linear regression model!

- β_1 represents a change in the log-odds of success

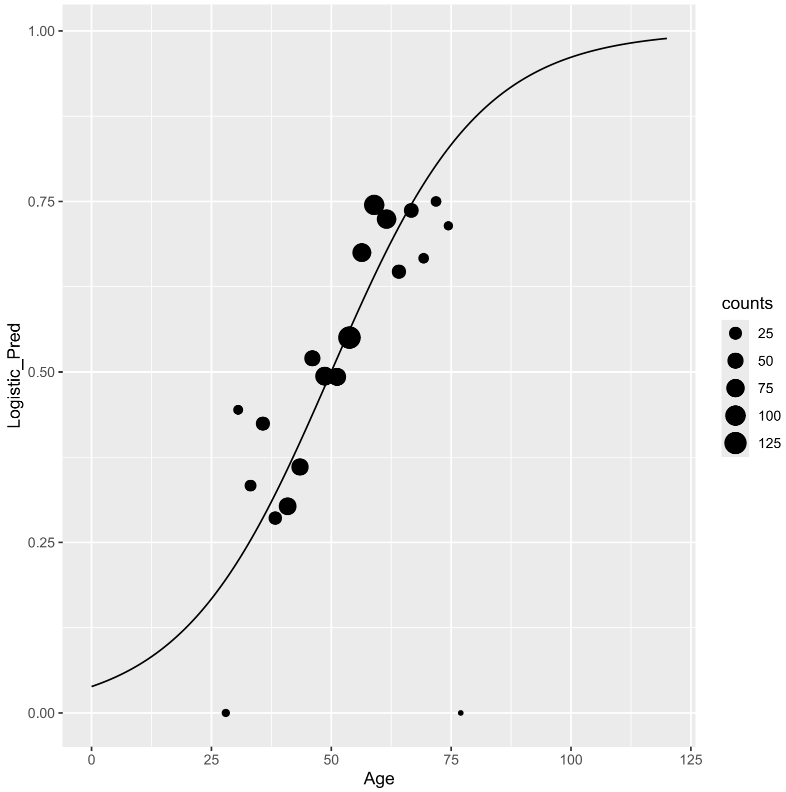
Logistic Regression Fit

Using `Age` to predict `HeartDisease` via a logistic regression model:



Logistic Regression Fit

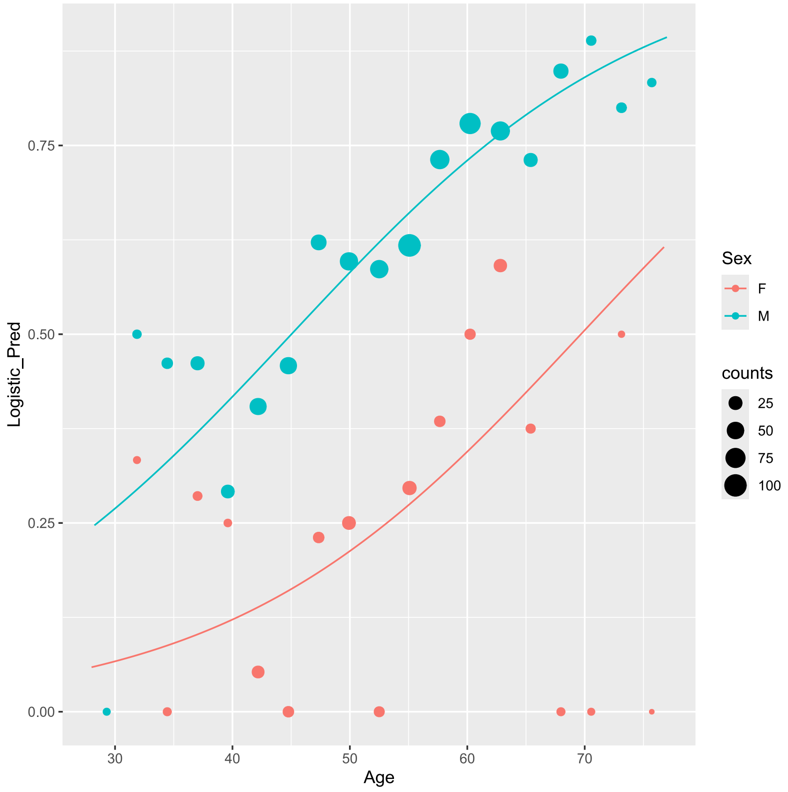
A sigmoid function that looks linear close up!



Logistic Regression

As with linear regression, we can include multiple predictors and interaction terms!

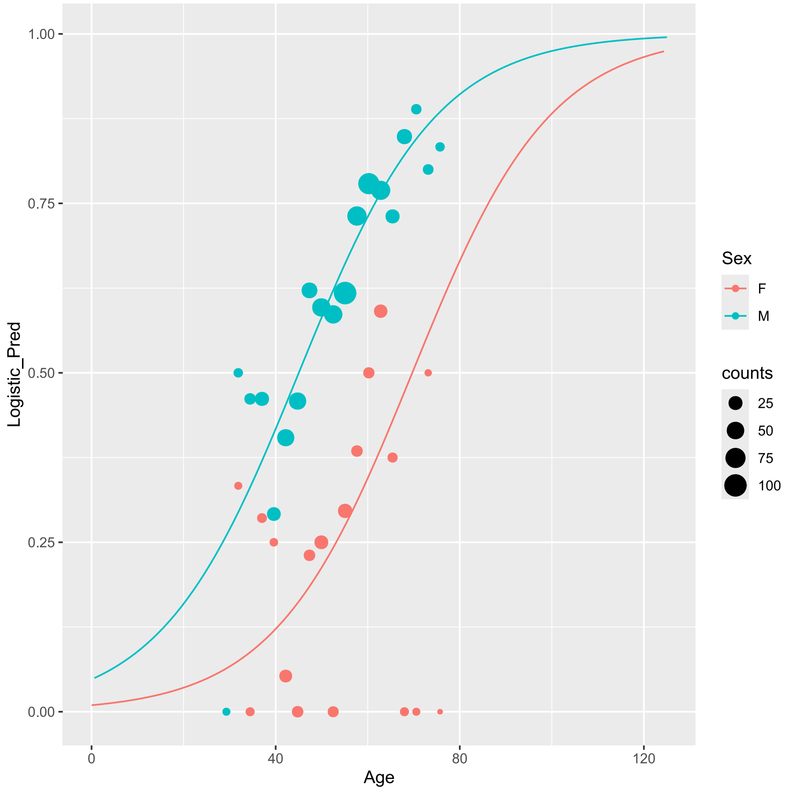
- Adding a dummy variable corresponding to a binary variable just changes the 'intercept'



Logistic Regression

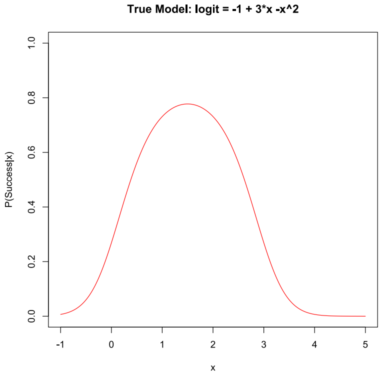
As with linear regression, we can include multiple predictors and interaction terms!

- Not a constant shift



Interaction Terms Can Be Included

- If we fit an interaction term with our dummy variable, we essentially fit two separate logistic regression models
- Can also include more than one numeric predictor
    - Difficult to visualize!
- Adding in polynomial terms increases flexibility as well!



Selecting a Model

- Recall we can use k-fold CV as a proxy for **test set** error if we don't want to split the data
- Metric to quantify prediction quality? Basic measures:
    - Accuracy:

$$\frac{\text{\# of correct classifications}}{\text{Total \# of classifications}}$$

    - Misclassification Rate:

$$\frac{\text{\# of incorrect classifications}}{\text{Total \# of classifications}}$$

Selecting a Model

- Recall we can use k-fold CV as a proxy for **test set** error if we don't want to split the data
- Metric to quantify prediction quality? Basic measures:
  - Accuracy:

$$\frac{\#\text{ of correct classifications}}{\text{Total }\#\text{ of classifications}}$$

  - Misclassification Rate:

$$\frac{\#\text{ of incorrect classifications}}{\text{Total }\#\text{ of classifications}}$$

  - Log-loss: For each observation (y = 0 or 1), $-(ylog(\hat{p}) + (1-y)log(1-\hat{p}))$

Using `tidymodels` to Fit a Logistic Regression Model

- First, we'll do a training/test split via `initial_split()`
- Let's also create our CV splits on the training data

```
library(tidymodels)
set.seed(3557)
heart_data <- heart_data |> mutate(HeartDisease = factor(HeartDisease))
heart_split <- initial_split(heart_data, prop = 0.8)
heart_train <- training(heart_split)
heart_test <- testing(heart_split)
heart_CV_folds <- vfold_cv(heart_train, 10)
```

Using `tidymodels` to Fit a Logistic Regression Model

- Next, we'll set up our recipes for the data, standardizing these numeric variables
    - Model 1: `Age` and `Sex` as predictors
    - Model 2: `Age`, `Sex`, `ChestPainType`, `RestingBP` and `RestingECG` as predictors
    - Model 3: `Age`, `Sex`, `ChestPainType`, `RestingBP`, `RestingECG`, `MaxHR`, and `ExerciseAngina`

```
LR1_rec <- recipe(HeartDisease ~ Age + Sex,
                  data = heart_train) |>
  step_normalize(Age) |>
  step_dummy(Sex)
LR2_rec <- recipe(HeartDisease ~ Age + Sex + ChestPainType + RestingBP + RestingECG,
                  data = heart_train) |>
  step_normalize(all_numeric(), -HeartDisease) |>
  step_dummy(Sex, ChestPainType, RestingECG)
LR3_rec <- recipe(HeartDisease ~ Age + Sex + ChestPainType + RestingBP + RestingECG + MaxHR + ExerciseAngina,
                  data = heart_train) |>
  step_normalize(all_numeric(), -HeartDisease) |>
  step_dummy(Sex, ChestPainType, RestingECG, ExerciseAngina)
LR3_rec |> prep(heart_train) |> bake(heart_train) |> colnames()
```

```
##  [1] "Age"              "RestingBP"        "MaxHR"
##  [4] "HeartDisease"     "Sex_M"            "ChestPainType_ATA"
##  [7] "ChestPainType_NAP" "ChestPainType_TA"  "RestingECG_Normal"
## [10] "RestingECG_ST"    "ExerciseAngina_Y"
```

Using `tidymodels` to Fit a Logistic Regression Model

- Now set up our model type and engine

```
LR_spec <- logistic_reg() |>
  set_engine("glm")
```

Using `tidymodels` to Fit a Logistic Regression Model

- Create our workflows

```r
LR1_wkf <- workflow() |>
  add_recipe(LR1_rec) |>
  add_model(LR_spec)
LR2_wkf <- workflow() |>
  add_recipe(LR2_rec) |>
  add_model(LR_spec)
LR3_wkf <- workflow() |>
  add_recipe(LR3_rec) |>
  add_model(LR_spec)
```

Using `tidymodels` to Fit a Logistic Regression Model

- Fit to our CV folds!

```
LR1_fit <- LR1_wkf |>
  fit_resamples(heart_CV_folds, metrics = metric_set(accuracy, mn_log_loss))
LR2_fit <- LR2_wkf |>
  fit_resamples(heart_CV_folds, metrics = metric_set(accuracy, mn_log_loss))
LR3_fit <- LR3_wkf |>
  fit_resamples(heart_CV_folds, metrics = metric_set(accuracy, mn_log_loss))
```

Using `tidymodels` to Fit a Logistic Regression Model

- Collect our metrics and see which model did the best!

```
rbind(LR1_fit |> collect_metrics(),
      LR2_fit |> collect_metrics(),
      LR3_fit |> collect_metrics()) |>
  mutate(Model = c("Model1", "Model1", "Model2", "Model2", "Model3", "Model3")) |>
  select(Model, everything())
```

```
## # A tibble: 6 x 7
##   Model  .metric     .estimator  mean     n std_err .config
##   <chr>  <chr>       <chr>      <dbl> <int>   <dbl> <chr>
## 1 Model1 accuracy    binary     0.689    10  0.0235 Preprocessor1_Model1
## 2 Model1 mn_log_loss binary     0.606    10  0.0246 Preprocessor1_Model1
## 3 Model2 accuracy    binary     0.768    10  0.0178 Preprocessor1_Model1
## 4 Model2 mn_log_loss binary     0.499    10  0.0268 Preprocessor1_Model1
## 5 Model3 accuracy    binary     0.783    10  0.0144 Preprocessor1_Model1
## 6 Model3 mn_log_loss binary     0.456    10  0.0204 Preprocessor1_Model1
```

```
#compare to proportion of 1's in training data
mean(heart_train$HeartDisease == "1")
```

```
## [1] 0.5607094
```

Using `tidymodels` to Fit a Logistic Regression Model

- Find the confusion matrix for our best model on the training set

```
LR_train_fit <- LR3_wkf |>
  fit(heart_train)
conf_mat(heart_train |> mutate(estimate = LR_train_fit |> predict(heart_train) |> pull()), #data
         HeartDisease, #truth
         estimate) #estimate from model
```

```
##             Truth
## Prediction   0   1
##          0 242  69
##          1  80 342
```

Using `tidymodels` to Fit a Logistic Regression Model

- Grab our 'best' model and test it on the test set

```
LR3_wkf |>
  last_fit(heart_split, metrics = metric_set(accuracy, mn_log_loss)) |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric     .estimator .estimate .config
##   <chr>       <chr>          <dbl> <chr>
## 1 accuracy    binary         0.810 Preprocessor1_Model1
## 2 mn_log_loss binary         0.409 Preprocessor1_Model1
```

```
conf_mat(heart_test |> mutate(estimate = LR_train_fit |> predict(heart_test) |> pull()), HeartDisease, estimate)
```

```
##           Truth
## Prediction  0  1
##          0 63 10
##          1 25 86
```

Using `tidymodels` to Fit a Logistic Regression Model

- Suppose we like this model the best *overall*, we'd fit it to the entire data set

```
final_model <- LR3_wkf |>
  fit(heart_data)
tidy(final_model)
```

```
## # A tibble: 11 x 5
##    term              estimate std.error statistic  p.value
##    <chr>                <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)        -0.468     0.281     -1.67  9.56e- 2
##  2 Age                 0.324     0.103      3.13  1.74e- 3
##  3 RestingBP           0.0877    0.0931     0.942 3.46e- 1
##  4 MaxHR              -0.363     0.105     -3.48  5.09e- 4
##  5 Sex_M               1.34      0.230      5.84  5.27e- 9
##  6 ChestPainType_ATA  -2.31      0.274     -8.43  3.33e-17
##  7 ChestPainType_NAP  -1.51      0.215     -7.02  2.17e-12
##  8 ChestPainType_TA   -0.937     0.360     -2.60  9.24e- 3
##  9 RestingECG_Normal  -0.113     0.233     -0.486 6.27e- 1
## 10 RestingECG_ST      -0.0737    0.294     -0.250 8.02e- 1
## 11 ExerciseAngina_Y    1.51      0.201      7.50  6.37e-14
```

Recap

- Logistic regression often a reasonable model for a binary response
- Uses a sigmoid function to ensure valid predictions
- Can predict success or failure using estimated probabilities
    - Usually predict success if probability > 0.5
    - Common metrics for classification are accuracy and log-loss