Maximum Likelihood Estimation and Logistic Regression

DATA 606 - Statistics & Probability for Data Analytics

Jason Bryer, Ph.D. and Angela Lui, Ph.D.

April 27, 2022

One Minute Paper Results

What was the most important thing you learned during this class?

```
## NULL
```

What important question remains unanswered for you?

```
## NULL
```

Maximum Likelihood Estimation

Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is an important procedure for estimating parameters in statistical models. It is often first encountered when modeling a dichotomous outcome variable vis-à-vis logistic regression. However, it is the backbone of generalized linear models (GLM) which allow for error distribution models other than the normal distribution. Most introductions to MLE rely on mathematical notation that for many students is opaque and hinders learning how this method works. The document outlines an approach to understanding MLE that relies on visualizations and mathematical notation is only used when necessary.

Bivariate Regression

We will begin with a typical bivariate regression using the `mtcars` data set where we wish to predict `mpg` (miles per gallon) from `wt` (weight in 1,000 lbs).

Linear Regression

Our goal is to estimate

$$Y_{mpg} = \beta_{wt} X + e$$

where β_{wt} is the slope and e is the intercept.

Ordinary Least Squares

With ordinary least squares (OLS) regression our goal is to minimize the residual sum of squares (RSS):

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

where y_i is the variable to be predicted, $f(x_i)$ is the predicted value of y_i, and n is the sample size.

The basic properties we know about regression are:

- The correlation measures the strength of the relationship between x and y (see this shiny app for an excellent visual overview of correlations).
- The correlation ranges between -1 and 1.
- The mean of x and y must fall on the line.
- The slope of a line is defined as the change in y over the change in x ($\frac{\Delta y}{\Delta x}$). For regression use the ration of the standard deviations such that the correlation is defined as $m = r\frac{s_y}{s_x}$ where m is the slope, r is the correlation, and s is the sample standard deviation.

Ordinary Least Squares

We can easily calculate the RSS for various correlations (r) ranging between -1 and 1.

```
y <- mtcars$mpg
x <- mtcars$wt
mean.y <- mean(y)
mean.x <- mean(x)
sd.y <- sd(y)
sd.x <- sd(x)
ols <- tibble(
    r = seq(-1, 1, by = 0.025),            # Correlation
    m = r * (sd.y / sd.x),                 # Slope
    b = mean.y - m * mean.x                # Intercept
) %>% rowwise() %>%
    mutate(ss = sum((y - (m * x + b))^2)) %>% # Sum of squares residuals
    as.data.frame()
```

Ordinary Least Squares

```
ggplot(ols, aes(x = r, y = ss)) + geom_path() + geom_point() +
    ggtitle('Residual sum of squares.') + xlab('Correlation') + ylab('Sum of Squares Residual')
```

Ordinary Least Squares

The correlation with the correlation the resulted in the smallest RSS is -0.875.

```
ols %>% dplyr::filter(ss == min(ss)) # Select the row with the smallest RSS
```

```
##        r         m       b       ss
## 1 -0.875 -5.389687 37.4306 278.3826
```

Calculating the correlation in R gives us -0.8676594 and the slope is -5.3444716 which is close to our estimate here. We could get a more accurate result if we tried smaller steps in the correlation (see the `by` parameter in the `seq` function above).

Minimizing RSS Algorithmically

This approach works well here because the correlation is bounded between -1 and 1 and we can easily calculate the RSS for a bunch of possible correlations. However, there are more efficient ways of finding the correlation that minimizes the RSS than trying correlations equally distributed across the possible range. For example, consider the following simple algorithm:

1. Calculate the RSS for $r = 0$.
2. Calculate the RSS for $r = 0.5$ If $RSS_{0.5} < RSS_0$ then calculate the RSS with $r = 0.75$, else calculate the RSS with $r = -0.5$

We can repeat this procedure, essentially halving the distance in each iteration until we find a sufficiently small RSS.

Minimizing RSS Algorithmically

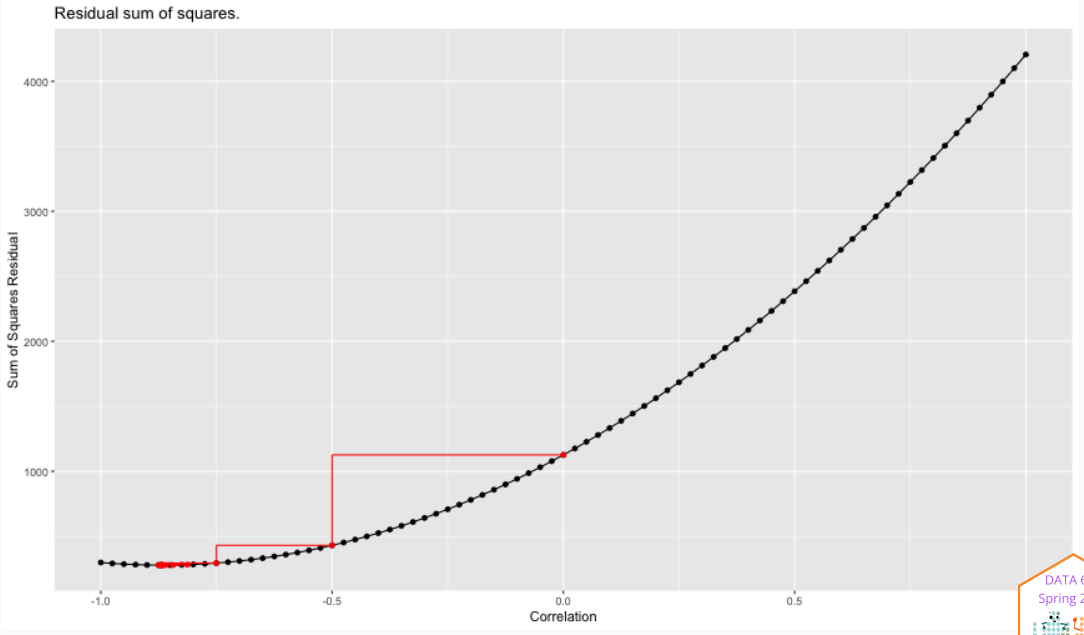
```
y <- mtcars$mpg
x <- mtcars$wt
ssr <- function(r, x, y) {
    mean.y <- mean(y); mean.x <- mean(x)
    sd.y <- sd(y); sd.x <- sd(x)
    m = r * (sd.y / sd.x)
    b = mean.y - m * mean.x
    ss = sum((y - (m * x + b))^2)
    return(ss)
}
r_left <- -1
r_right <- 1
ssr_left <- ssr(r_left, x = x, y = y)
ssr_right <- ssr(r_right, x = x, y = y)
iter <- numeric()
threshold <- 0.00001
while(abs(ssr_left - ssr_right) > threshold) {
    if(ssr_left < ssr_right) {
        r_right <- r_right - (r_right - r_left) / 2
        ssr_right <- ssr(r_right, x = x, y = y)
        iter <- c(iter, r_right)
    } else {
        r_left <- r_left + (r_right - r_left) / 2
        ssr_left <- ssr(r_left, x = x, y = y)
        iter <- c(iter, r_left)
    }
}
```

```
r_left; r_right; cor(x, y)
```

```
## [1] -0.8676758
```

```
## [1] -0.8676147
```

```
## [1] -0.8676594
```



The `optim` function

This process is, in essence, the idea of numerical optimization procedures. In R, the `optim` function implements the Nedler-Mead (Nedler & Mead, 1965) and Limited Memory BFGS (Byrd et al, 1995) methods for optimizing a set of parameters. The former is the default but we will use the latter throughout this document since it allows for specifying bounds for certain parameters (e.g. only consider positive values). The details of *how* the algorithm works is beyond the scope of this article (see this interactive tutoral by Ben Frederickson for a good introduction), instead we will focus on *what* the algorithm does.

Example

To begin, we must define a function that calculates a metric for which the optimizer is going to minimize (or maximize).

```
residual_sum_squares <- function(parameters, predictor, outcome) {
    a <- parameters[1] # Intercept
    b <- parameters[2] # beta coefficient
    predicted <- a + b * predictor
    residuals <- outcome - predicted
    ss <- sum(residuals^2)
    return(ss)
}
```

The `parameters` is a vector of the parameters the algorithm is going to minimize (or maximize). Here, these will be the slope and intercept. The `predictor` and `outcome` are parameters passed through from the `...` parameter on the `optim` function and are necessary for us to calculate the RSS. We can now get the RSS for any set of parameters.

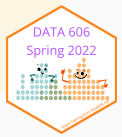
```
residual_sum_squares(c(37, -5), mtcars$wt, mtcars$mpg)
```

```
## [1] 303.5247
```

Small Digression: Saving the steps along the way...

In order to explore each step of the algorithm, we need to wrap the `optim` function to capture the parameters and output of the function. The `optim_save` function will add two elements to the returned list: `iterations` is the raw list of the parameters and output saved and `iterations_df` is a `data.frame` containing the same data.

```
optim_save <- function(par, fn, ...) {
    iterations <- list()
    wrap_fun <- function(parameters, ...) {
        n <- length(iterations)
        result <- fn(parameters, ...)
        iterations[[n + 1]] <<- c(parameters, result)
        return(result)
    }
    optim_out <- stats::optim(par, wrap_fun, ...)
    optim_out$iterations <- iterations
    optim_out$iterations_df <- as.data.frame(do.call(rbind, iterations))
    names(optim_out$iterations_df) <- c(paste0('Param', 1:length(par)), 'Result')
    optim_out$iterations_df$Iteration <- 1:nrow(optim_out$iterations_df)
    return(optim_out)
}
```



OLS with the `optim` function

We can now call the `optim_save` function with our `residual_sum_squares` function. We initialize the algorithm with two random values for the intercept and slope, respectively. Note that we are using Broyden, Fletcher, Goldfarb, and Shanno optimization method which allows for the specification of bounds on the parameter estimates which we will use later.

```
optim.rss <- optim_save(
    par = runif(2),
    fn = residual_sum_squares,
    method = "L-BFGS-B",
    predictor = mtcars$wt,
    outcome = mtcars$mpg
)
```

OLS with the `optim` function

The `par` parameter provides the final parameter estimates.

```
optim.rss$par
```

```
## [1] 37.285113 -5.344468
```

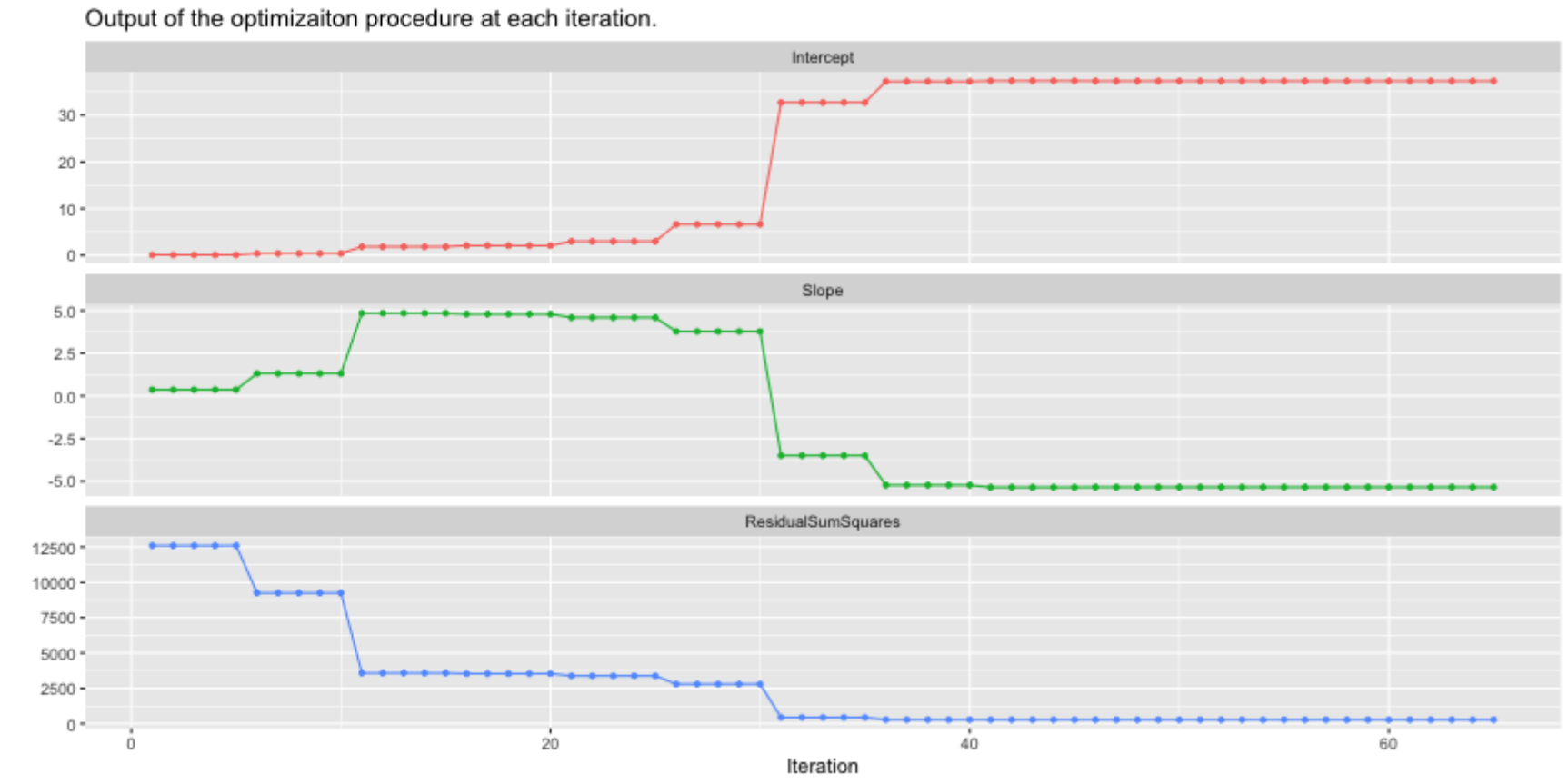
We can see that the parameters are accurate to at least four decimal places to the OLS method used by the `lm` function.

```
lm.out <- lm(mpg ~ wt, data = mtcars)
lm.out$coefficients
```

```
## (Intercept)          wt
##   37.285126   -5.344472
```

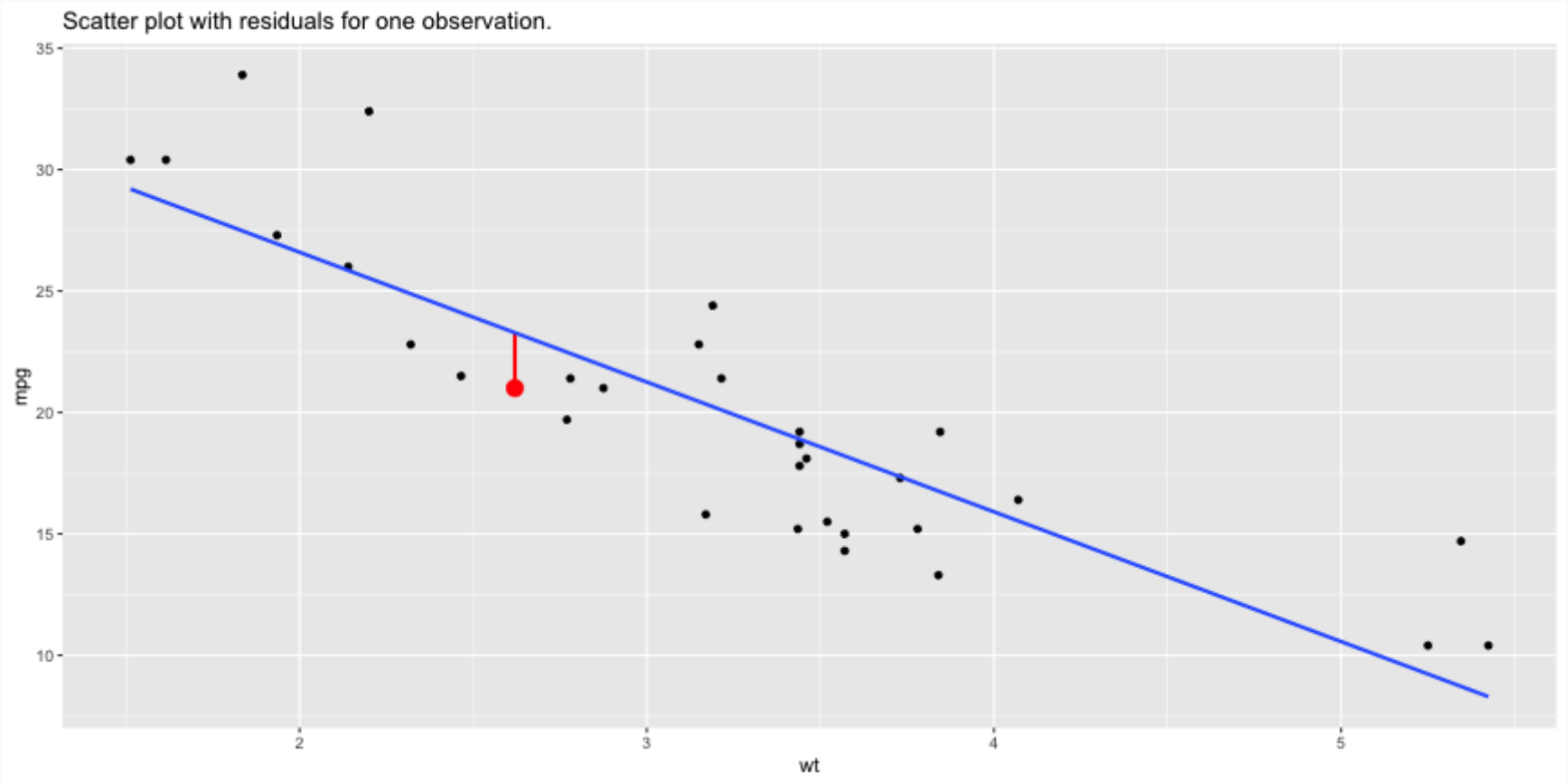
OLS with the `optim` function

It took the `optim` function 65 iterations to find the optimal set of parameters that minimized the RSS. This figure shows the value of the parameters (i.e. intercept and slope) and the RSS for each iteration.



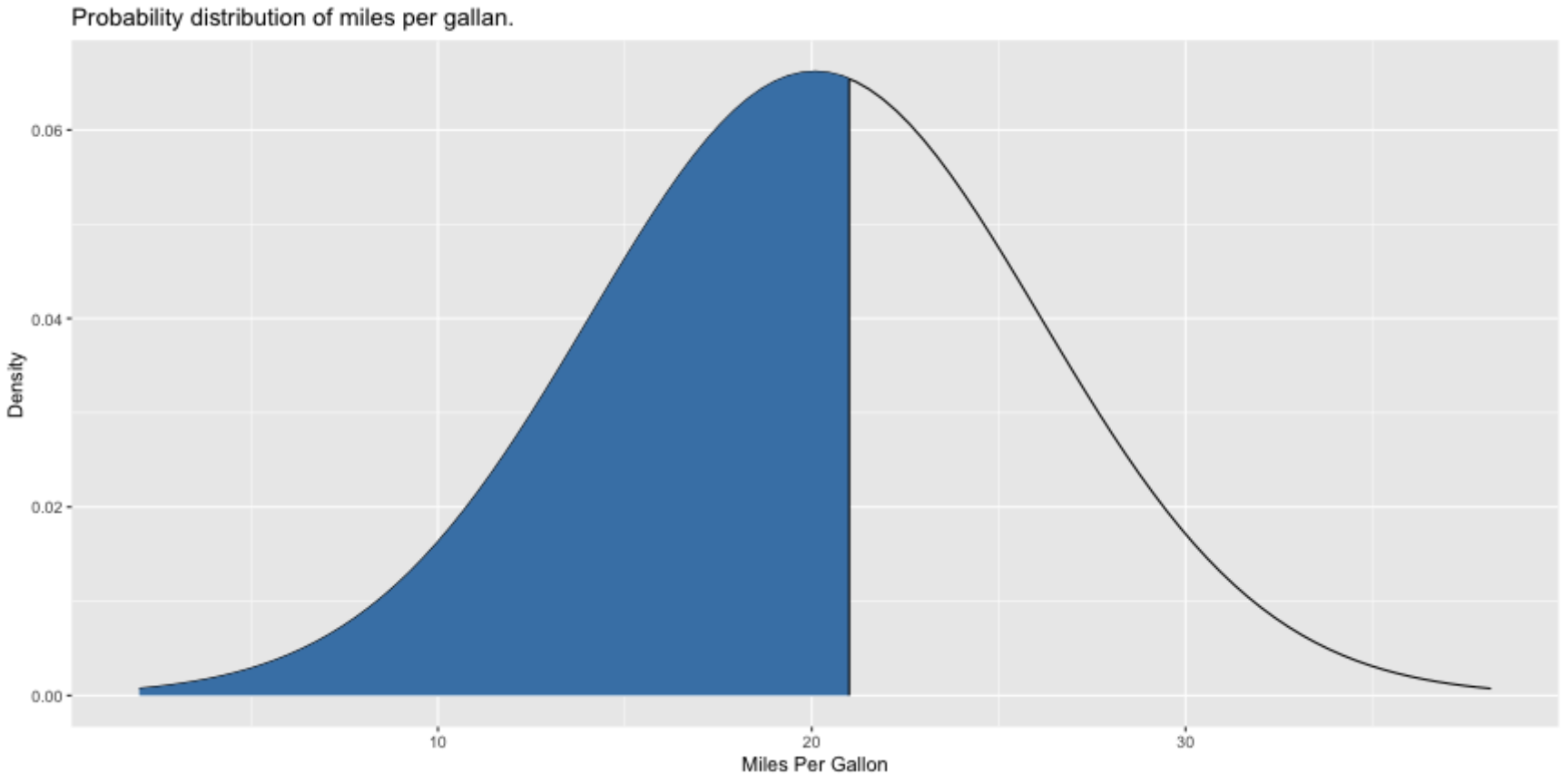
Residuals to Likelihoods

Now that we have laid the groundwork for finding parameters algorithmically, we need to introduce another way of evaluating how well parameters *fit* the data, namely the likelihood. First, let's revisit what we are doing in OLS.



Probability

We often think of probabilities as the areas under a fixed distribution. For example, the first car in `mtcars` is Mazda RX4 with an average miles per gallon of 21 and weighs 2620lbs. The probability of a car with a miles per gallon less than Mazda RX4 given the data we have in `mtcars` is 0.5599667.



Probabilities and Likelihoods

For probabilities, we are working with a fixed distribution, that is:

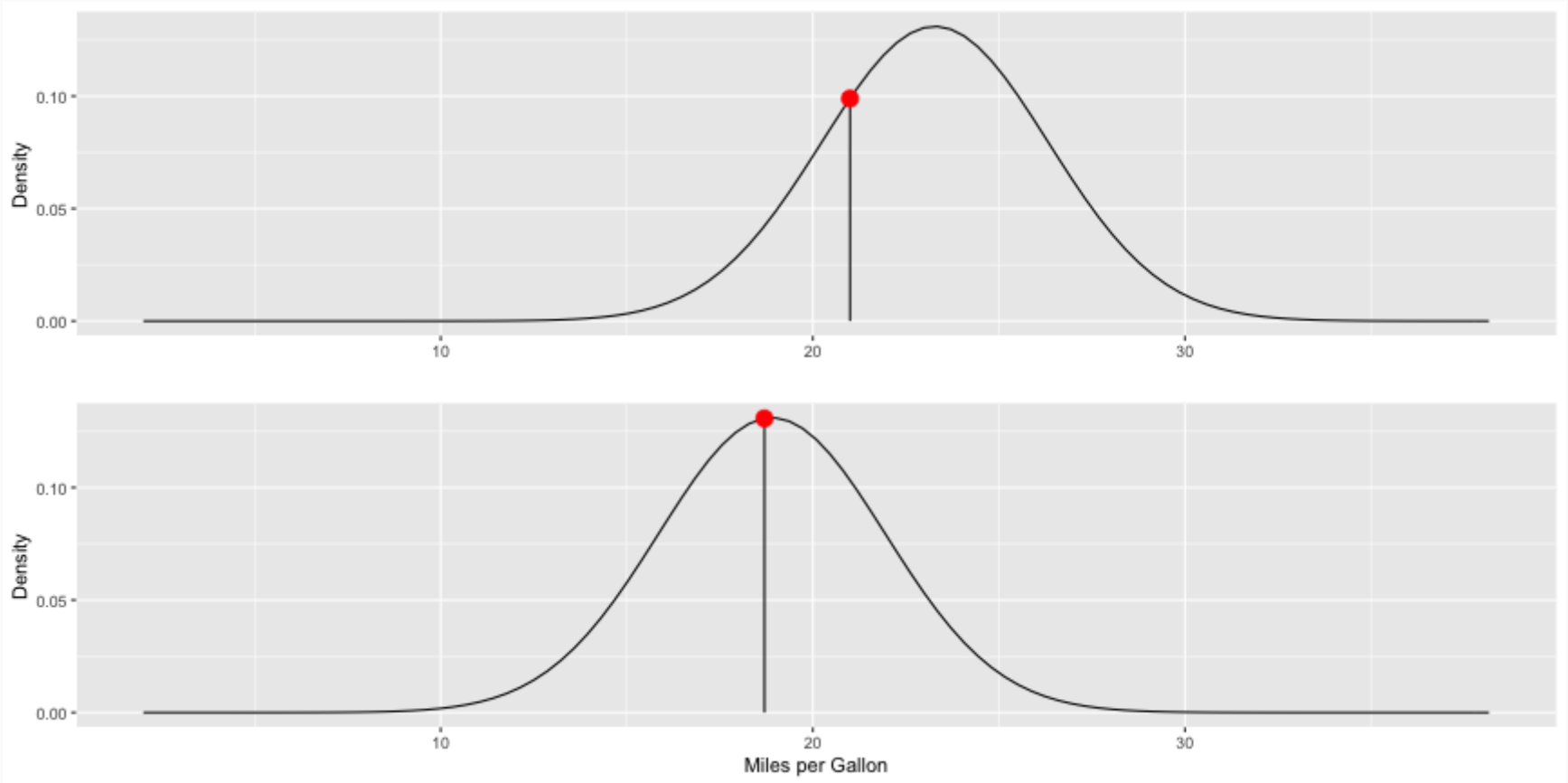
$$pr(data \mid distribution)$$

The likelihood are the y-axis values (i.e. density) for fixed data points with distributions that can move, that is:

$$L(distribution \mid data)$$

Likelihoods

The likelihood is the height of the density function. This figure depicts two likelihood for two observations. The mean of each distribution is equal to $\beta_{wt}X + e$ and the intercept (also known as the error term) defines the standard deviation of the distribution.



Log-Likelihood Function

We can then calculate the likelihood for each observation in our data. Unlike OLS, we now want to *maximize* the sum of these values. Also, we are going to use the log of the likelihood so we can add them instead of multiplying. We can now define our log likelihood function:

```
loglikelihood <- function(parameters, predictor, outcome) {
    a <- parameters[1]     # intercept
    b <- parameters[2]     # slope / beta coefficient
    sigma <- parameters[3] # error
    ll.vec <- dnorm(outcome, a + b * predictor, sigma, log = TRUE)
    return(sum(ll.vec))
}
```

Note that we have to estimate a third parameter, sigma, which is the error term and defines the standard deviation for the normal distribution for estimating the likelihood. This is connected to the distribution of the residuals as we will see later. We can now calculate the log-likelihood for any combination of parameters.

```
loglikelihood(c(37, -5, sd(mtcars$mpg)), predictor = mtcars$wt, outcome = mtcars$mpg)
```

Maximum Likelihood Estimation

We can now use the `optim_save` function to find the parameters that *maximize* the log-likelihood. Note two important parameter changes:

1. We are specifying the `lower` parameter so that the algorithm will not try negative values for sigma since the variance cannot be negative.
2. The value for the `control` parameter indicates that we wish to maximize the values instead of minimizing (which is the default).

```
optim.ll <- optim_save(
    runif(3),                     # Random initial values
    loglikelihood,                # Log-likelihood function
    lower = c(-Inf, -Inf, 1.e-5), # The lower bounds for the values, note sigma, cannot be negative
    method = "L-BFGS-B",
    control = list(fnscale = -1), # Indicates that the maximum is desired rather than the minimum
    predictor = mtcars$wt,
    outcome = mtcars$mpg
)
```

Maximum Likelihood Estimation

We can get our results and compare them to the results of the `lm` function and find that they match to at least four decimal places.

```
optim.ll$par[1:2]
```

```
## [1] 37.285127 -5.344472
```

```
lm.out$coefficients
```

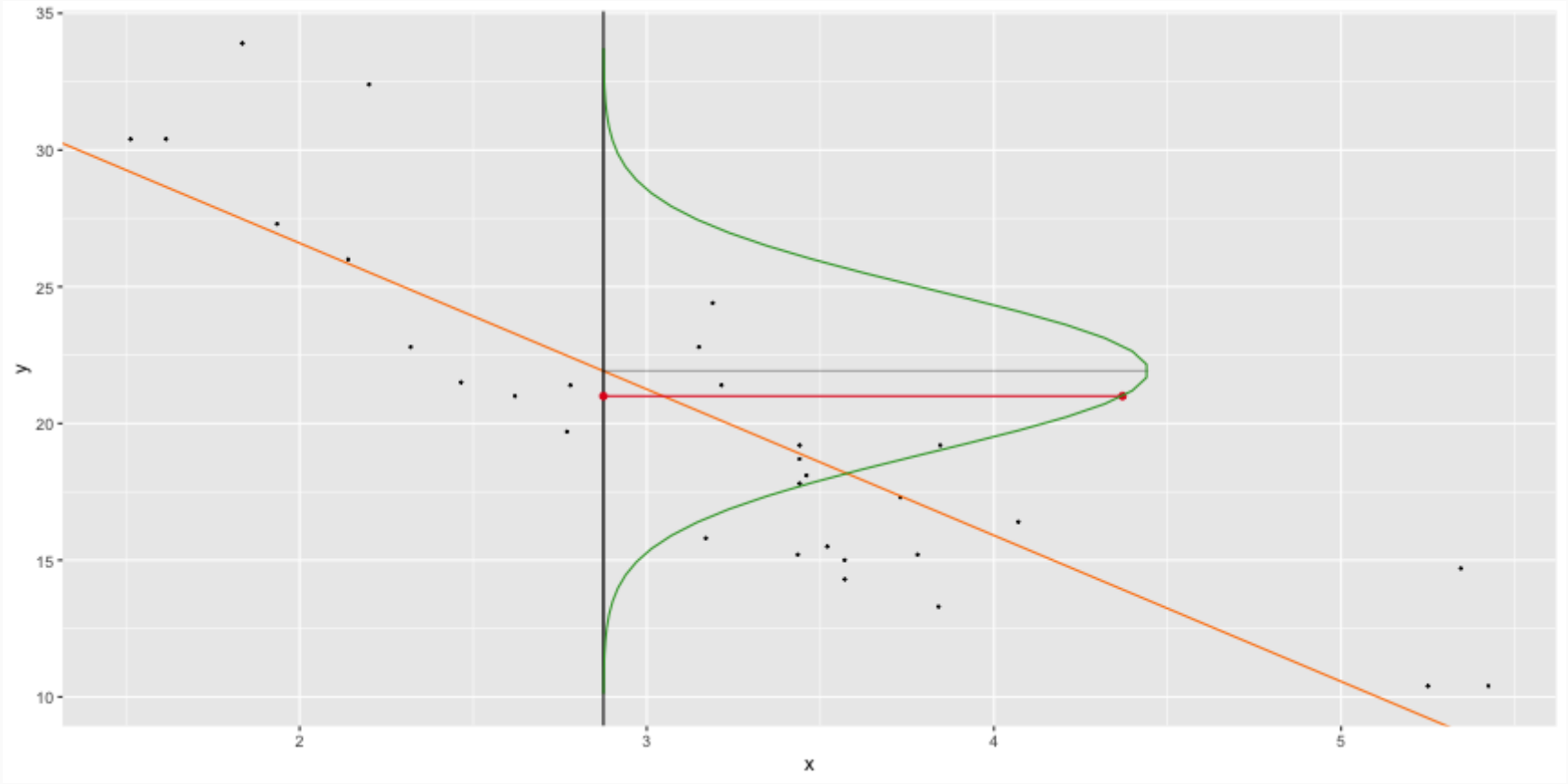
```
## (Intercept)          wt
##   37.285126   -5.344472
```

The steps of MLE

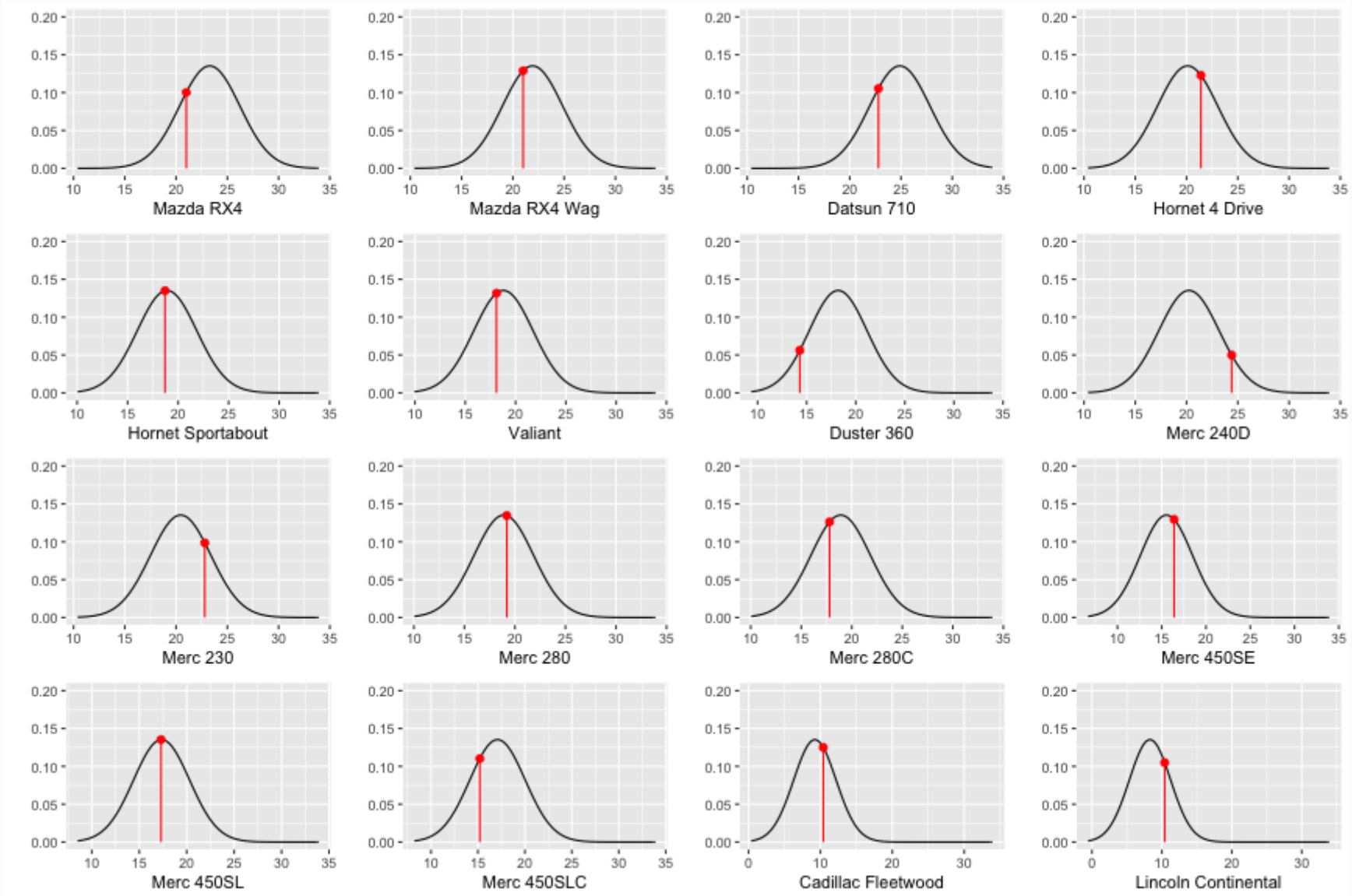
This figure shows the estimated regression line for each iteration of the optimization procedure (on the left; OLS regression line in blue; MLE regression line in black) with the estimated parameters and log-likelihood for all iterations on the left.

Likelihood Visualized

```
visualMLE::plot_likelihood(x = mtcars$wt, y = mtcars$mpg, pt = 2,
                           intercept = optim.ll$par[1],
                           slope = optim.ll$par[2],
                           sigma = optim.ll$par[3])
```



Likelihood Visualized



Root-Mean-Square Error

With MLE we need to estimate what is often referred to as the error term, or as we saw above is the standard deviation of the normal distribution from which we are estimating the likelihood from. In the previous figure notice that the normal distribution id drawn vertically. This is because the likelihood is estimated from the error, or the residuals. In OLS we often report the root-mean-square deviation (RMSD, or root-mean-square error, RMSE). The RMSD is the standard deviation of the residuals:

$$RMSD = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}}$$

Where i is the observation, x_i is the observed value, $\hat{x}_i$ is the estimated (predicted) value, and N is the sample size. Below, we see that the numerical optimizer matches the RMSD within a rounding error.

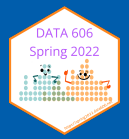
```
optim.ll$par[3]
```

```
## [1] 2.949164
```

```
sqrt(sum(resid(lm.out)^2) / nrow(mtcars))
```

```
## [1] 2.949163
```

Logistic Regression



Dichotomous (x) and continuous (y) variables

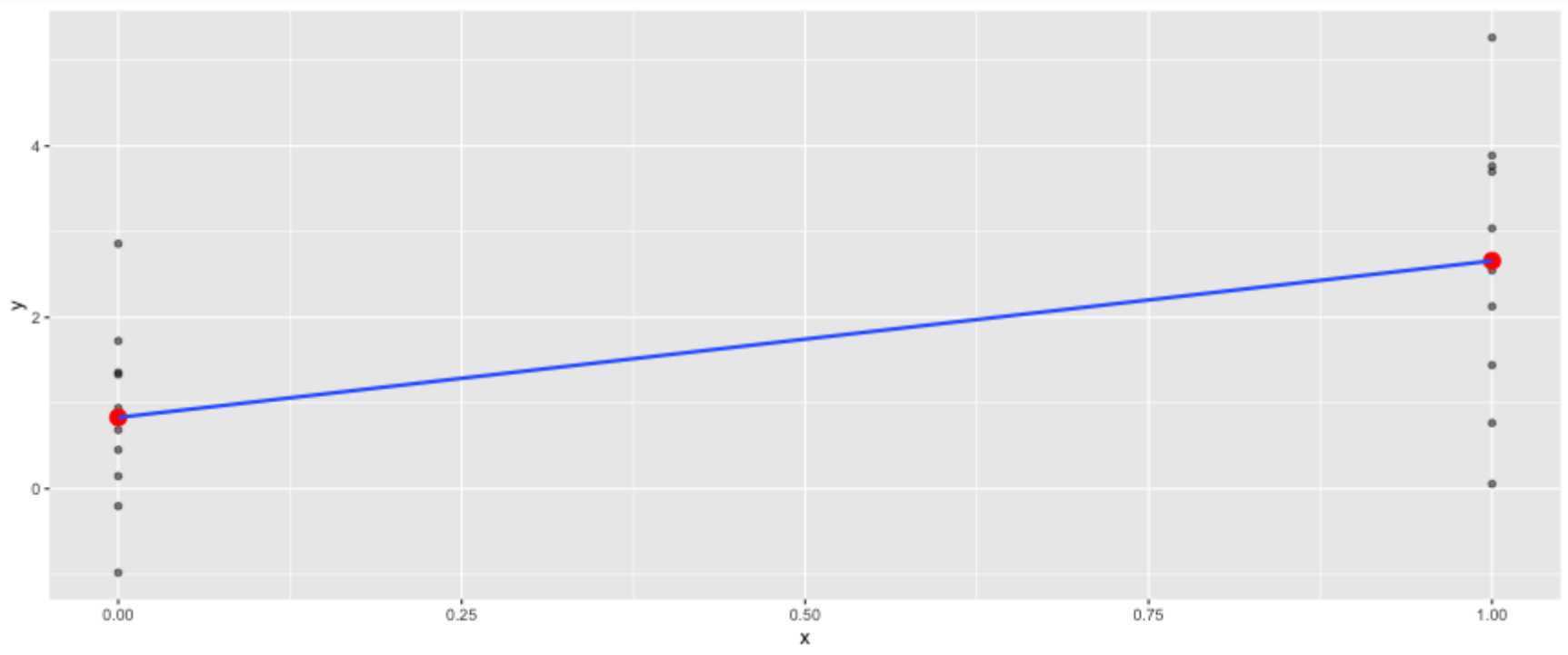
```
df <- data.frame(
    x = rep(c(0, 1), each = 10),
    y = c(rnorm(10, mean = 1, sd = 1),
          rnorm(10, mean = 2.5, sd = 1.5))
)
head(df)
```

```
##   x          y
## 1 0  1.3573080
## 2 0  2.8606923
## 3 0 -0.2019866
## 4 0  1.3343358
## 5 0  1.7258741
## 6 0  0.4548514
```

```
tab <- describeBy(df$y, group = df$x, mat = TRUE, skew = FALSE)
tab$group1 <- as.integer(as.character(tab$group1))
```

Dichotomous (x) and continuous (y) variables

```
ggplot(df, aes(x = x, y = y)) +    geom_point(alpha = 0.5) +
    geom_point(data = tab, aes(x = group1, y = mean), color = 'red', size = 4) +
    geom_smooth(method = lm, se = FALSE, formula = y ~ x)
```



Regression so far...

At this point we have covered:

- Simple linear regression
  - Relationship between numerical response and a numerical or categorical predictor
- Multiple regression
  - Relationship between numerical response and multiple numerical and/or categorical predictors
- Maximum Likelihood Estimation

All of the approaches we have used so far have a quantitative variable with normally distributed errors (i.e. residuals).

What we haven't seen is what to do when the predictors are weird (nonlinear, complicated dependence structure, etc.) or when the response is weird (categorical, count data, etc.)

Odds

Odds are another way of quantifying the probability of an event, commonly used in gambling (and logistic regression).

For some event E,

$$\text{odds}(E) = \frac{P(E)}{P(E^c)} = \frac{P(E)}{1 - P(E)}$$

Similarly, if we are told the odds of E are x to y then

$$\text{odds}(E) = \frac{x}{y} = \frac{x/(x+y)}{y/(x+y)}$$

which implies

$$P(E) = x/(x+y), \quad P(E^c) = y/(x+y)$$

Generalized Linear Models

Generalized linear models (GLM) are a generalization of OLS that allows for the response variables (i.e. dependent variables) to have an error distribution that is ***not*** distributed normally. All generalized linear models have the following three characteristics:

1. A probability distribution describing the outcome variable .
2. A linear model: $\eta = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$.
3. A link function that relates the linear model to the parameter of the outcome distribution: $g(p) = \eta$ or $p = g^{-1}(\eta)$.

We can estimate GLMs using maximum likelihood estimation (MLE). What will change is the log-likelihood function.

Logistic Regression

Logistic regression is a GLM used to model a binary categorical variable using numerical and categorical predictors.

We assume a binomial distribution produced the outcome variable and we therefore want to model p the probability of success for a given set of predictors.

To finish specifying the Logistic model we just need to establish a reasonable link function that connects η to p. There are a variety of options but the most commonly used is the logit function.

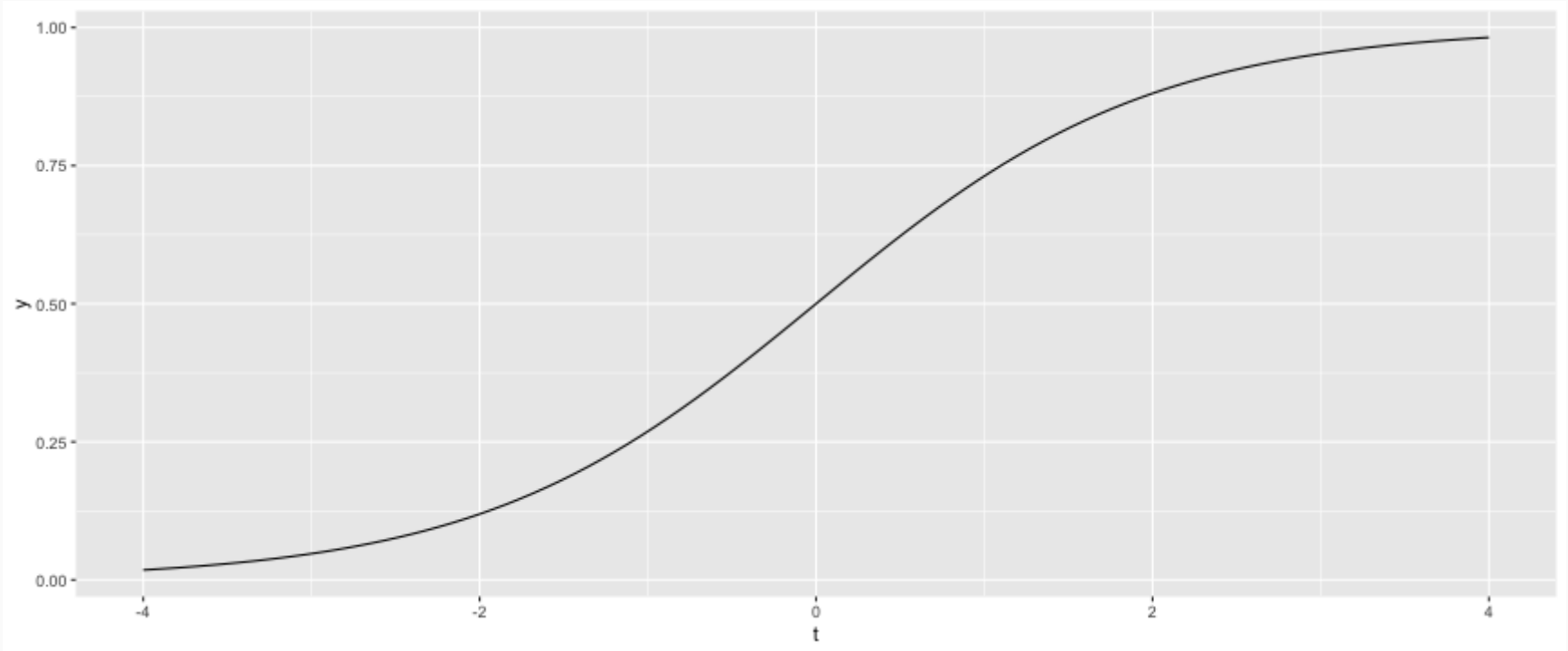
Logit function

$$logit(p) = \log\left(\frac{p}{1-p}\right), \text{ for } 0 \leq p \leq 1$$

The Logistic Function

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

```
logistic <- function(t) { return(1 / (1 + exp(-t))) }
ggplot() + stat_function(fun = logistic, n = 101) +  xlim(-4, 4) + xlab('t')
```



t as a Linear Function

$$t = \beta_0 + \beta_1 x$$

The logistic function can now be rewritten as

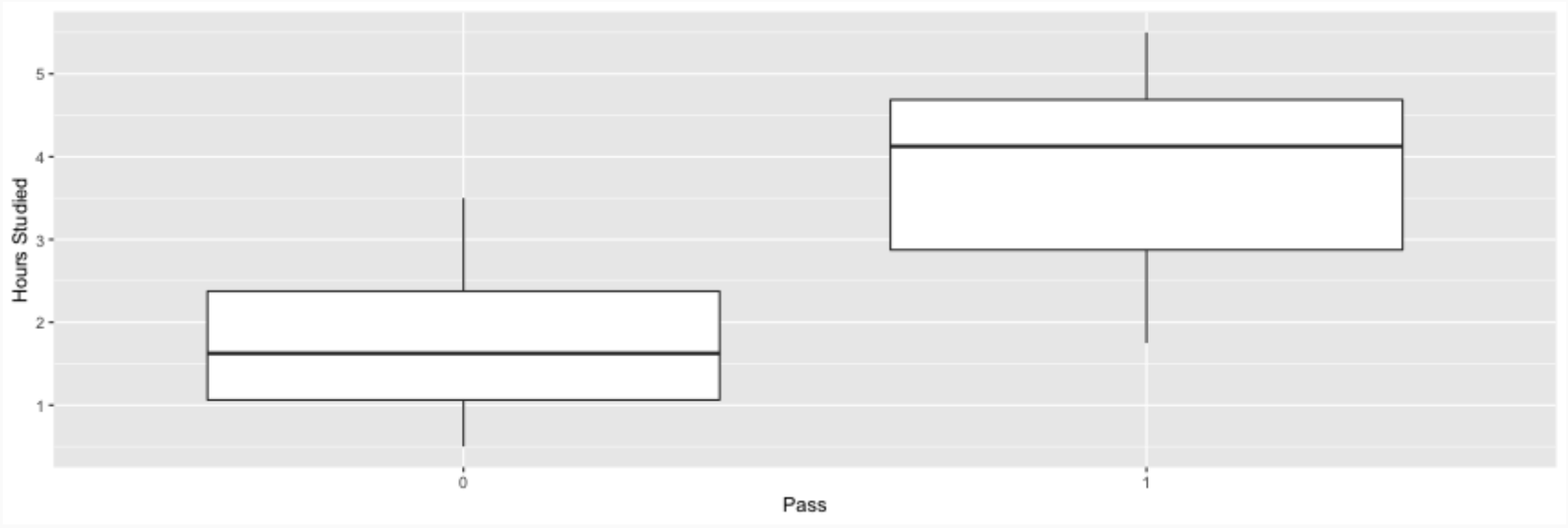
$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Similar to OLS, we wish to minimize the errors. However, instead of minimizing the least squared residuals, we will use a maximum likelihood function.

Example: Hours Studying Predicting Passing

```
study <- data.frame(
    Hours=c(0.50,0.75,1.00,1.25,1.50,1.75,1.75,2.00,2.25,2.50,2.75,3.00,
            3.25,3.50,4.00,4.25,4.50,4.75,5.00,5.50),
    Pass=c(0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1)
)
```

```
ggplot(study, aes(x=factor(Pass), y=Hours)) + geom_boxplot() + xlab('Pass') + ylab('Hours Studied')
```



Loglikelihood Function

We need to define logit function and the log-likelihood function that will be used by the optim function. Instead of using the normal distribution as above (using the dnorm function), we are using a binomial distribution and the logit to link the linear combination of predictors.

```
logit <- function(x, beta0, beta1) {
    return( 1 / (1 + exp(-beta0 - beta1 * x)) )
}

loglikelihood.binomial <- function(parameters, predictor, outcome) {
    a <- parameters[1] # Intercept
    b <- parameters[2] # beta coefficient
    p <- logit(predictor, a, b)
    ll <- sum( outcome * log(p) + (1 - outcome) * log(1 - p))
    return(ll)
}
```

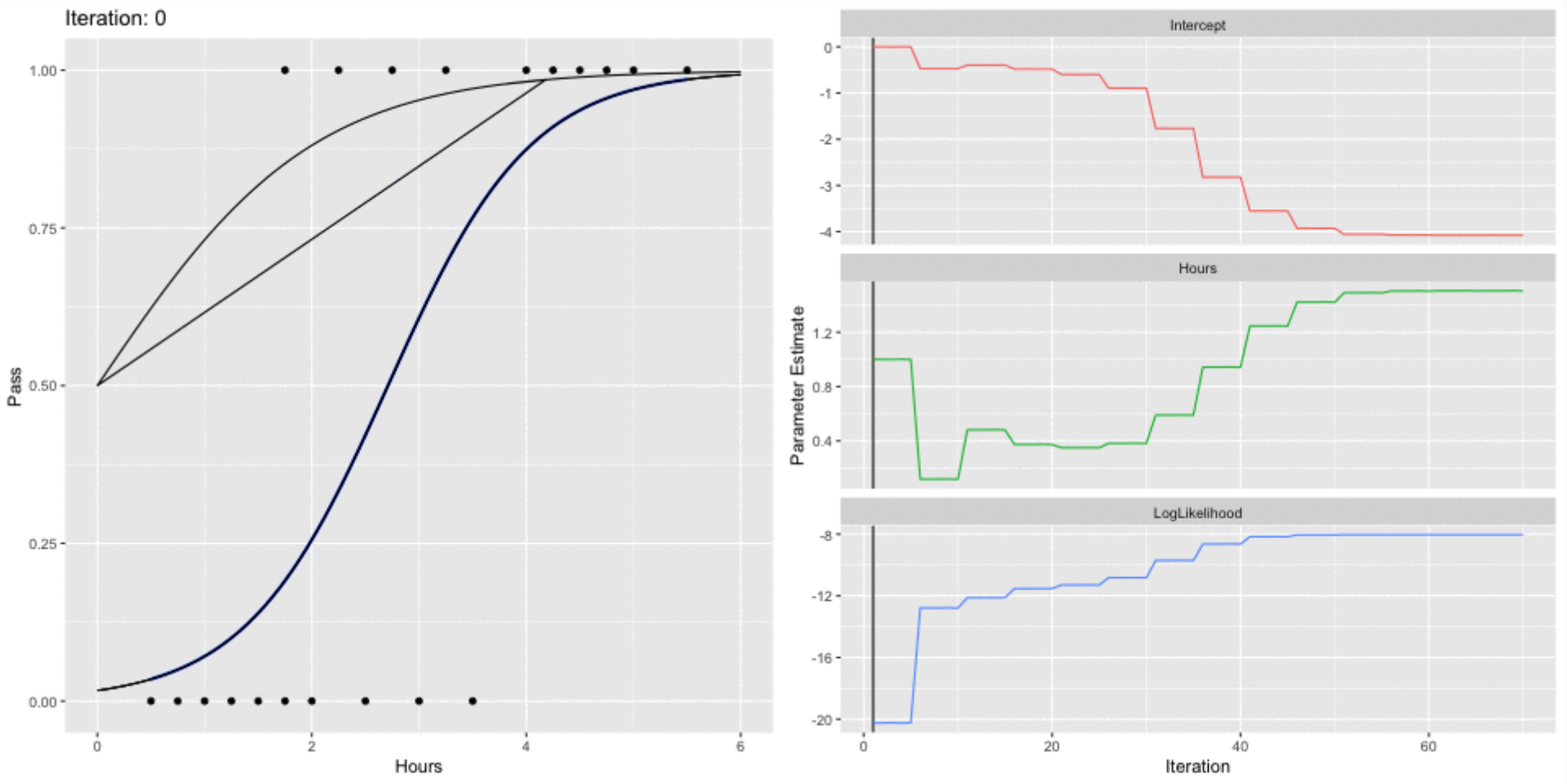
Estimating parameters using the `optim` function

```
optim.binomial <- optim_save(
    c(0, 1), # Initial values
    loglikelihood.binomial,
    method = "L-BFGS-B",
    control = list(fnscale = -1),
    predictor = study$Hours,
    outcome = study$Pass
)

optim.binomial$par
```

```
## [1] -4.077575  1.504624
```

How did the optimizer get to this result?



The `glm` function

```
( lr.out <- glm(Pass ~ Hours, data = study, family = binomial(link = 'logit')) )
```

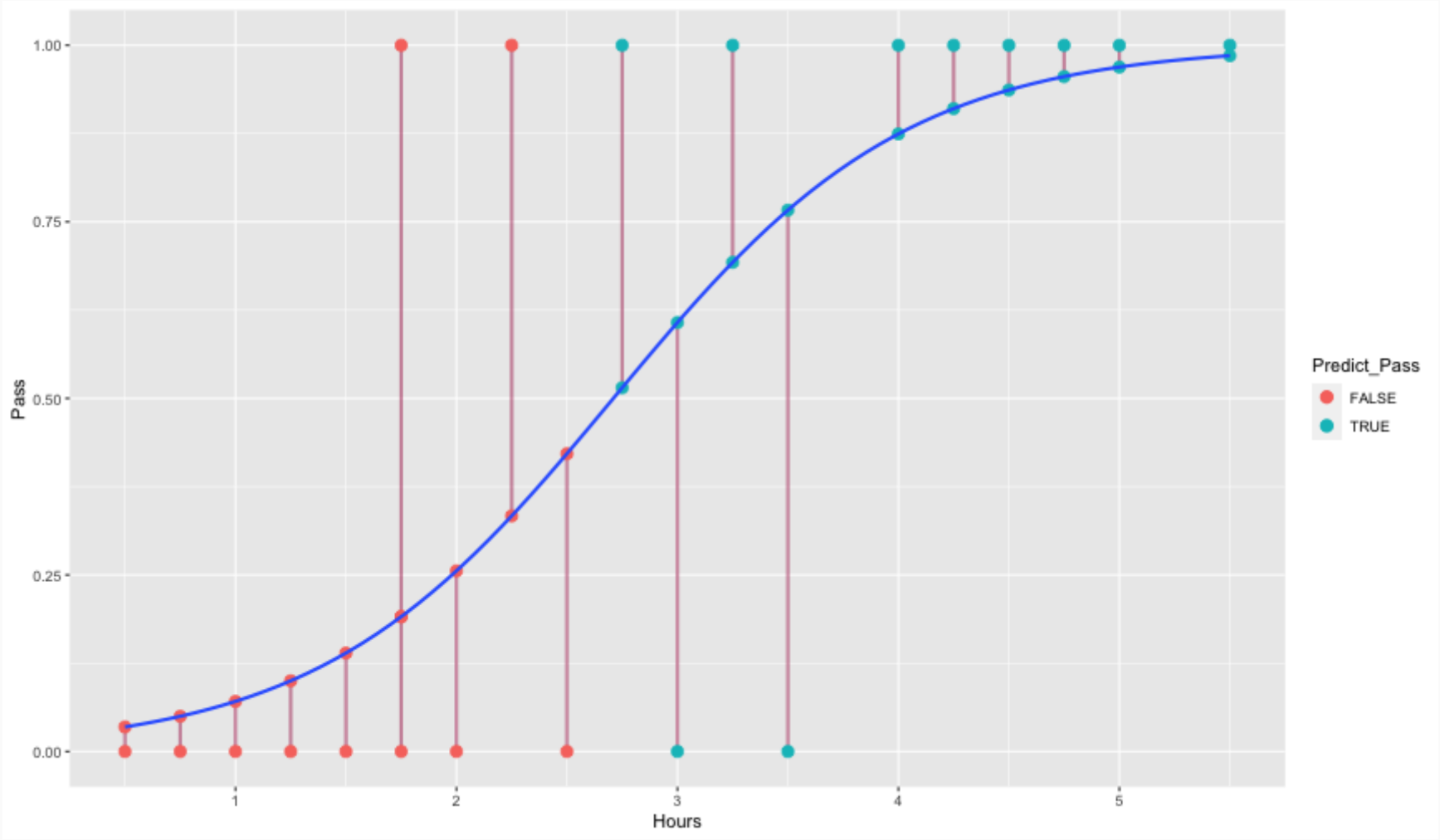
```
##
## Call:  glm(formula = Pass ~ Hours, family = binomial(link = "logit"),
##     data = study)
##
## Coefficients:
## (Intercept)        Hours
##      -4.078        1.505
##
## Degrees of Freedom: 19 Total (i.e. Null);  18 Residual
## Null Deviance:        27.73
## Residual Deviance: 16.06     AIC: 20.06
```

How does this compare to the `optim` function?

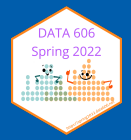
```
optim.binomial$par
```

```
## [1] -4.077575  1.504624
```

Plotting the Results



Predictive Modeling



Prediction

Odds (or probability) of passing if studied **zero** hours?

$$log(\frac{p}{1-p}) = -4.078 + 1.505 \times 0$$

$$\frac{p}{1-p} = exp(-4.078) = 0.0169$$

$$p = \frac{0.0169}{1.169} = .016$$

Odds (or probability) of passing if studied **4** hours?

$$log(\frac{p}{1-p}) = -4.078 + 1.505 \times 4$$

$$\frac{p}{1-p} = exp(1.942) = 6.97$$

Fitted Values

```
study[1,]
```

```
##   Hours Pass    Predict Predict_Pass          p
## 1   0.5    0 0.03471034        FALSE 0.03471462
```

```
logistic <- function(x, b0, b1) {
    return(1 / (1 + exp(-1 * (b0 + b1 * x)) ))
}
logistic(.5, b0=-4.078, b1=1.505)
```

```
## [1] 0.03470667
```

Model Performance

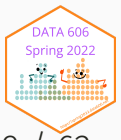
The use of statistical models to predict outcomes, typically on new data, is called predictive modeling. Logistic regression is a common statistical procedure used for prediction. We will utilize a **confusion matrix** to evaluate accuracy of the predictions.

		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

Predicting survivors of the Titanic

```
str(titanic)
```

```
## 'data.frame':    1309 obs. of  14 variables:
##  $ pclass   : Ord.factor w/ 3 levels "First"<"Second"<..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ survived : Factor w/ 2 levels "No","Yes": 2 2 1 1 1 2 2 1 2 1 ...
##  $ name     : Factor w/ 1307 levels "Abbing, Mr. Anthony",..: 22 24 25 26 27 31 46 47 51 55 ...
##  $ sex      : Factor w/ 2 levels "female","male": 1 2 1 2 1 2 1 2 1 2 ...
##  $ age      : num  29 0.92 2 30 25 48 63 39 53 71 ...
##  $ sibsp    : int  0 1 1 1 1 0 1 0 2 0 ...
##  $ parch    : int  0 2 2 2 2 0 0 0 0 0 ...
##  $ ticket   : Factor w/ 929 levels "110152","110413",..: 188 50 50 50 50 125 93 16 77 826 ...
##  $ fare     : num  211 152 152 152 152 ...
##  $ cabin    : Factor w/ 187 levels "","A10","A11",..: 45 81 81 81 81 151 147 17 63 1 ...
##  $ embarked : Factor w/ 4 levels "","C","Q","S": 4 4 4 4 4 4 4 4 4 2 ...
##  $ boat     : Factor w/ 28 levels "","1","10","11",..: 13 4 1 1 1 14 3 1 28 1 ...
##  $ body     : int  NA NA NA 135 NA NA NA NA NA 22 ...
##  $ home.dest: Factor w/ 370 levels "","?Havana, Cuba",..: 310 232 232 232 232 238 163 25 23 230 ...
```



Data Setup

We will split the data into a training set (70% of observations) and validation set (30%).

```
train.rows <- sample(nrow(titanic), nrow(titanic) * .7)
titanic_train <- titanic[train.rows,]
titanic_test <- titanic[-train.rows,]
```

This is the proportions of survivors and defines what our "guessing" rate is. That is, if we guessed no one survived, we would be correct 62% of the time.

```
(survived <- table(titanic_train$survived) %>% prop.table)
```

```
##
##        No       Yes
## 0.6135371 0.3864629
```

Model Training

```
lr.out <- glm(survived ~ pclass + sex + sibsp + parch, data=titanic_train, family=binomial(link = 'logit'))
summary(lr.out)
```

```
##
## Call:
## glm(formula = survived ~ pclass + sex + sibsp + parch, family = binomial(link = "logit"),
##     data = titanic_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1414  -0.7180  -0.4857   0.7142   2.3834
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.26831    0.15357   8.259   <2e-16 ***
## pclass.L    -1.27555    0.14574  -8.752   <2e-16 ***
## pclass.Q     0.03935    0.15941   0.247   0.8050
## sexmale     -2.46018    0.17823 -13.803   <2e-16 ***
## sibsp       -0.22839    0.09702  -2.354   0.0186 *
## parch        0.10561    0.10175   1.038   0.2993
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1222.20  on 915  degrees of freedom
## Residual deviance:  885.72  on 910  degrees of freedom
## AIC: 897.72
```

Predicted Values

```
titanic_train$prediction <- predict(lr.out, type = 'response', newdata = titanic_train)
ggplot(titanic_train, aes(x = prediction, color = survived)) + geom_density()
```

Results

```
titanic_train$prediction_class <- titanic_train$prediction > 0.5
tab <- table(titanic_train$prediction_class,
             titanic_train$survived) %>% prop.table() %>% print()
```

```
##
##                  No        Yes
##   FALSE 0.52183406 0.12882096
##   TRUE  0.09170306 0.25764192
```

For the training set, the overall accuracy is 77.95%. Recall that 38.65% of passengers survived. Therefore, the simplest model would be to predict that everyone died, which would mean we would be correct 61.35% of the time. Therefore, our prediction model is 16.59% better than guessing.

Checking with the validation dataset

```
(survived_test <- table(titanic_test$survived) %>% prop.table())
```

```
##
##        No       Yes
## 0.6284987 0.3715013
```

```
titanic_test$prediction <- predict(lr.out, newdata = titanic_test, type = 'response')
titanic_test$prediciton_class <- titanic_test$prediction > 0.5
tab_test <- table(titanic_test$prediciton_class, titanic_test$survived) %>%
    prop.table() %>% print()
```

```
##
##                  No        Yes
##    FALSE 0.56997455 0.12213740
##    TRUE  0.05852417 0.24936387
```

The overall accuracy is 81.93%, or 19.1% better than guessing.

Receiver Operating Characteristic (ROC) Curve

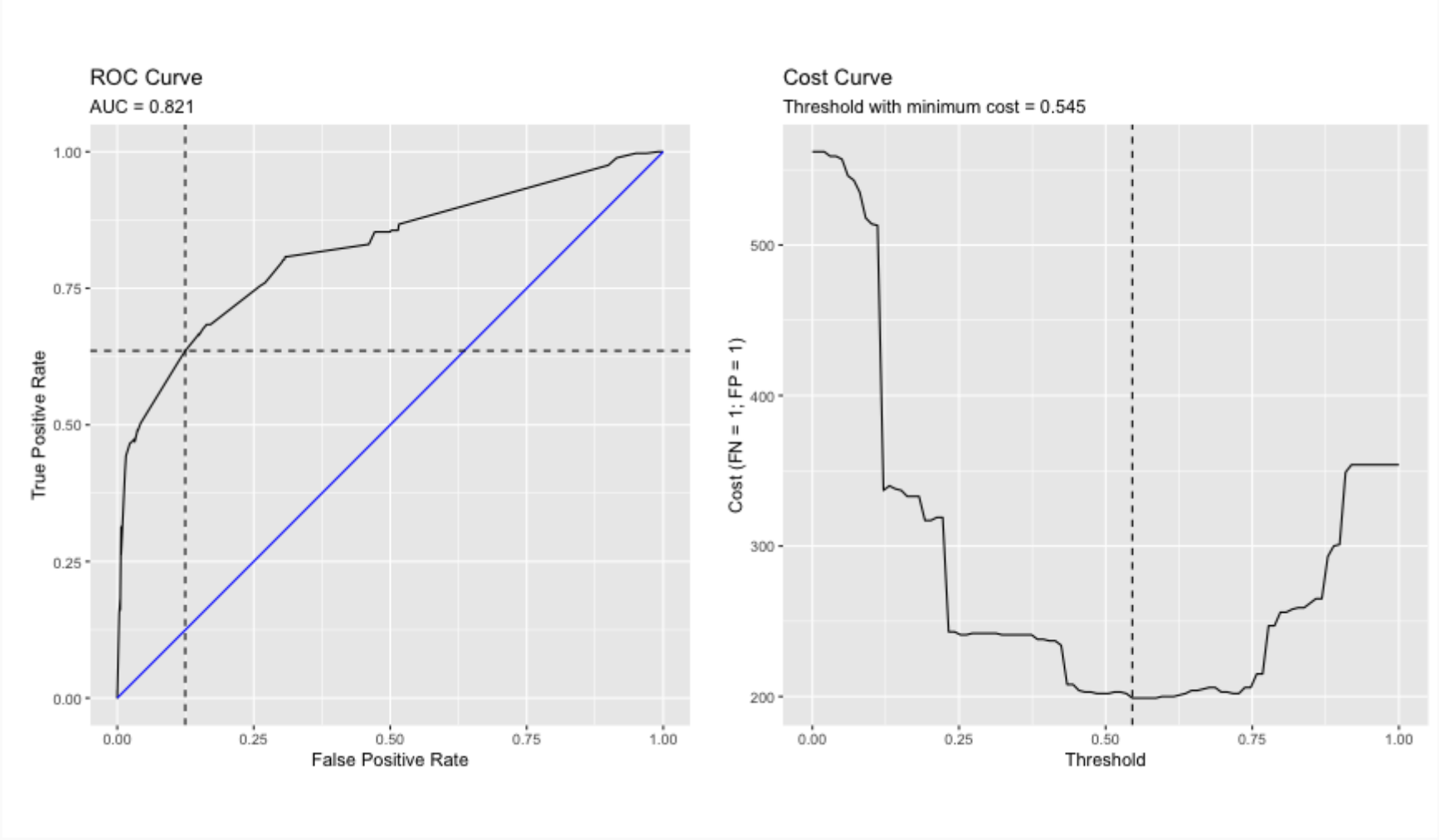
The ROC curve is created by plotting the true positive rate (TPR; AKA sensitivity) against the false positive rate (FPR; AKA probability of false alarm) at various threshold settings.

```
roc <- calculate_roc(titanic_train$prediction, titanic_train$survived == 'Yes')
summary(roc)
```

```
## AUC = 0.821
## Cost of false-positive = 1
## Cost of false-negative = 1
## Threshold with minimum cost = 0.545
```

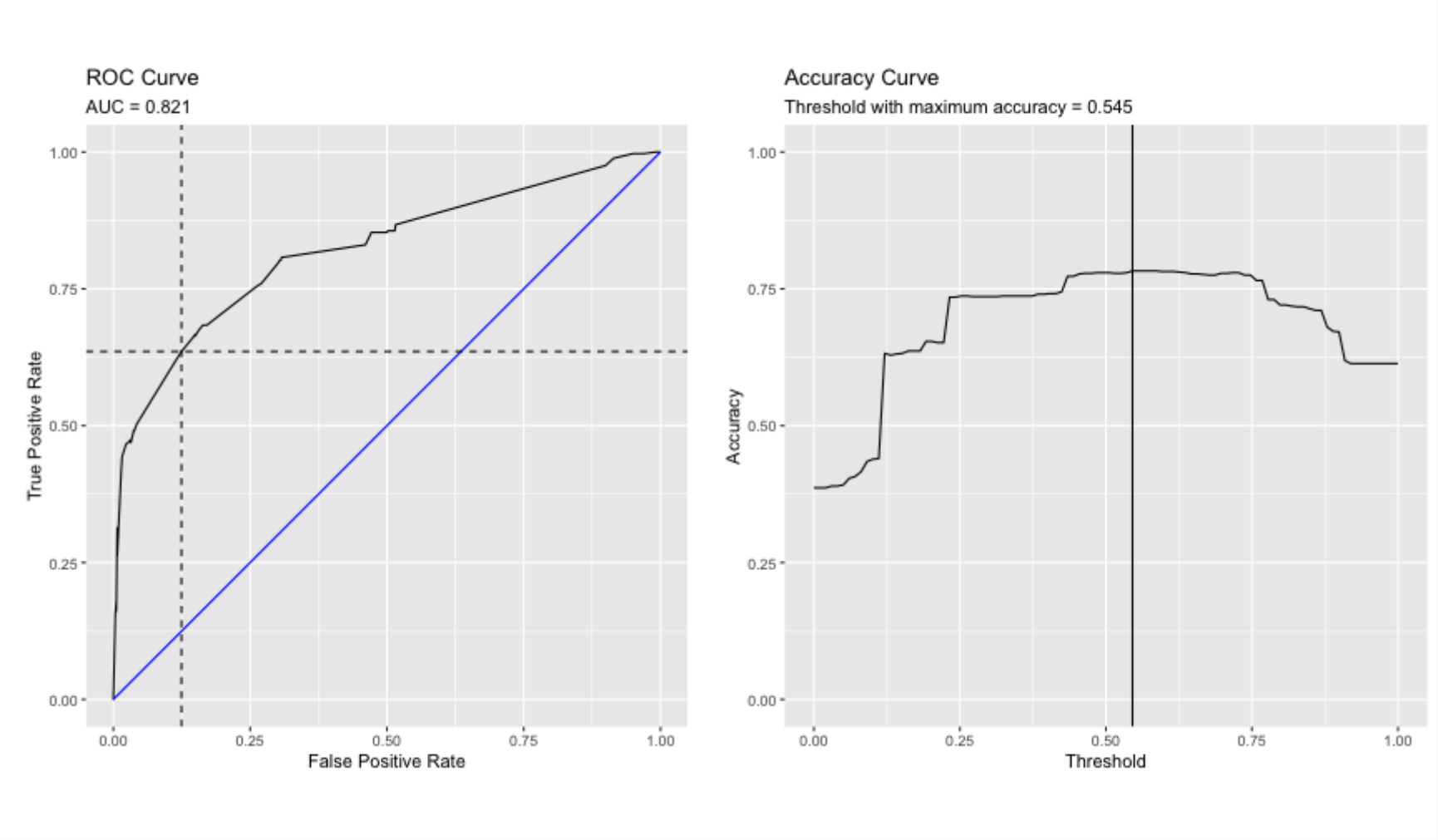
ROC Curve

```
plot(roc)
```



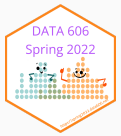
ROC Curve

```
plot(roc, curve = 'accuracy')
```



Caution on Interpreting Accuracy

- Loh, Sooo, and Zing (2016) predicted sexual orientation based on Facebook Status.
- They reported model accuracies of approximately 90% using SVM, logistic regression and/or random forest methods.
- Gallup (2018) poll estimates that 4.5% of the Americal population identifies as LGBT.
- *My proposed model:* I predict all Americans are heterosexual.
- The accuracy of my model is 95.5%, or *5.5% better than Facebook's model!*
- Predicting "rare" events (i.e. when the proportion of one of the two outcomes large) is difficult and requires independent (predictor) variables that strongly associated with the dependent (outcome) variable.



Fitted Values Revisited

What happens when the ratio of true-to-false increases (i.e. want to predict "rare" events)?

Let's simulate a dataset where the ratio of true-to-false is 10-to-1. We can also define the distribution of the dependent variable. Here, there is moderate separation in the distributions.

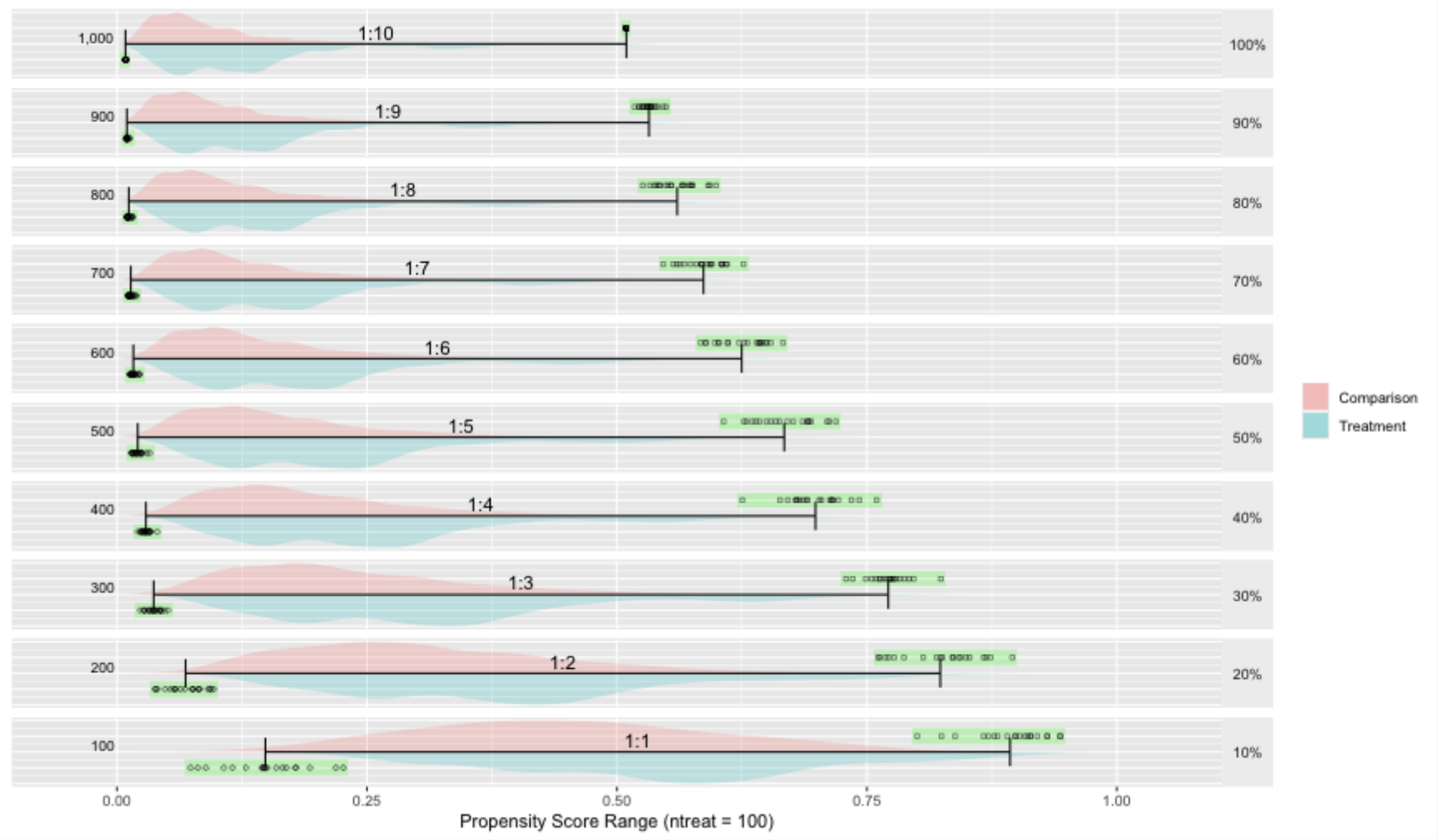
```
test.df2 <- getSimulatedData(
    treat.mean=.6, control.mean=.4)
```

The `multilevelPSA::psrange` function will sample with varying ratios from 1:10 to 1:1. It takes multiple samples and averages the ranges and distributions of the fitted values from logistic regression.

```
psranges2 <- psrange(test.df2, test.df2$treat, treat ~ .,
                     samples=seq(100,1000,by=100), nboot=20)
```

Fitted Values Revisited (cont.)

```
plot(psranges2)
```



Additional Resources

- Logistic Regression Details Pt 2: Maximum Likelihood
- StatQuest: Maximum Likelihood, clearly explained
- Probability concepts explained: Maximum likelihood estimation

One Minute Paper

Complete the one minute paper:

https://forms.gle/qxRnsCyydx1nf8sXA

1. What was the most important thing you learned during this class?
2. What important question remains unanswered for you?