Maximum Likelihood Estimation

EPSY 630 - Statistics II

Jason Bryer, Ph.D.

March 23, 2021

Agenda

1 Data Project Questions

2 Lab Questions

3 Maximum Likelihood Estimation

4 One minute papers

Data Project Questions

Data Project Proposal

Due March 30th. Select a dataset that interests you. For the proposal, you need to answer the questions below.

- Research question
- What type of statistical test do you plan to do (e.g. t-test, ANOVA, regression, logistic regression, chi-squred, etc.)
- What are the cases, and how many are there?
- Describe the method of data collection.
- What type of study is this (observational/experiment)?
- Data Source: If you collected the data, state self-collected. If not, provide a citation/link.
- Response: What is the response variable, and what type is it (numerical/categorical)?
- Explanatory: What is the explanatory variable(s), and what type is it (numerical/categorival)?
- Relevant summary statistics

More information including template and suggested datasets located here: https://epsy630.bryer.org/assignments/project/

Lab Questions

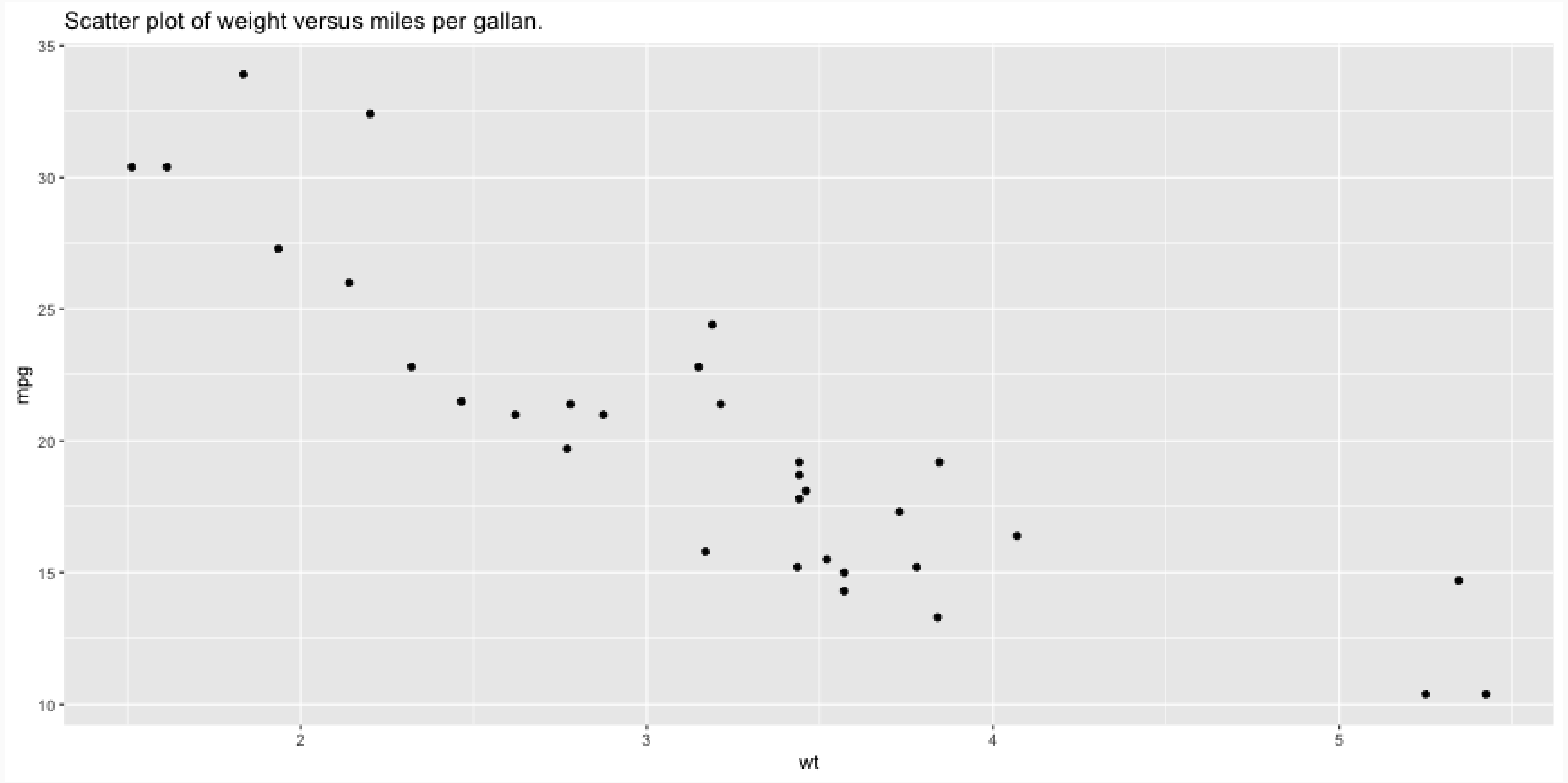
Maximum Likelihood Estimation

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.

EPSY 630
Spring 2021

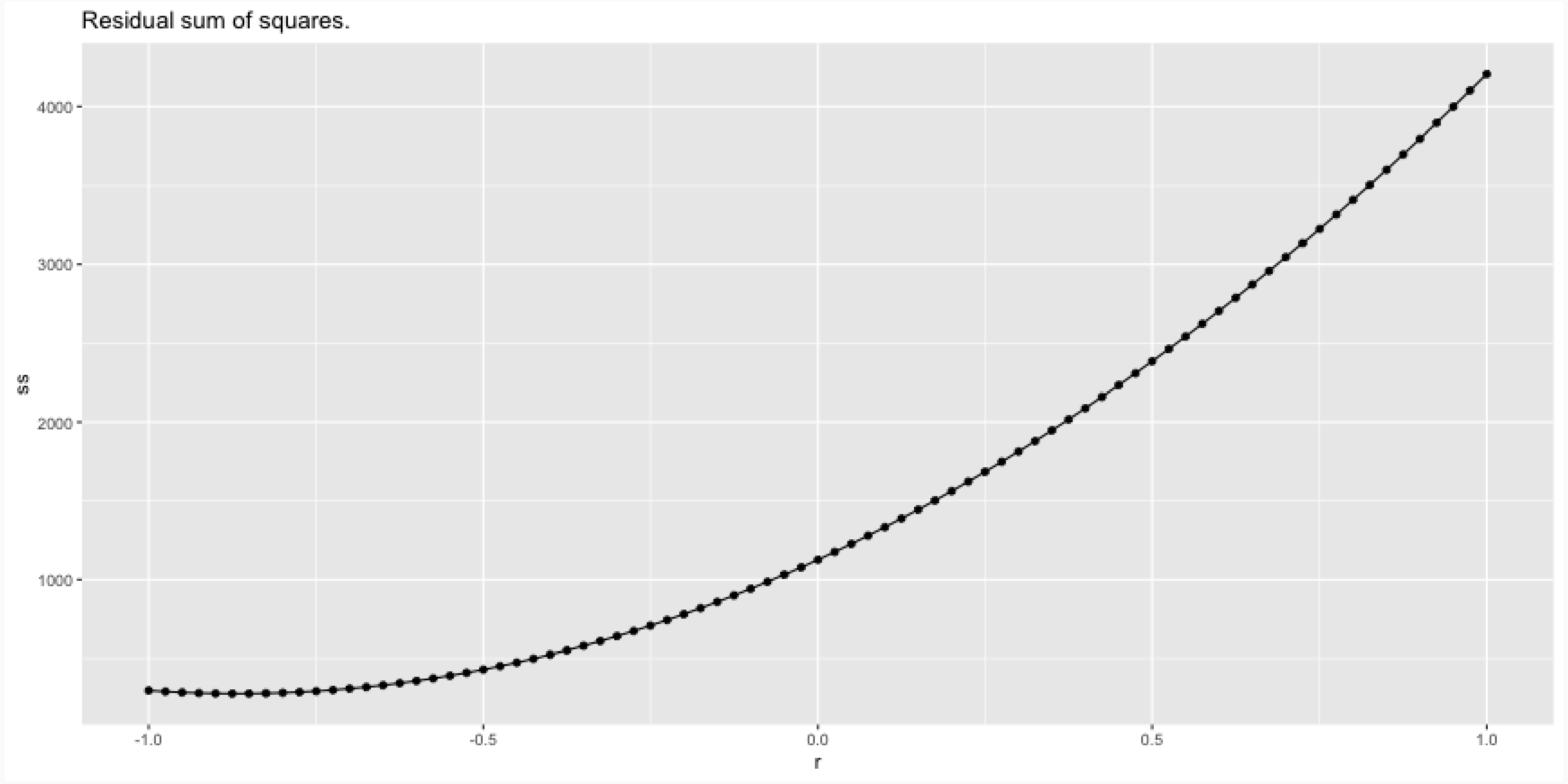
Ordinary Least Squares

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

```r
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 square
```



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.

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).

```r
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.

```r
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.

```r
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.

```r
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] 37.285125 -5.344471
```

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

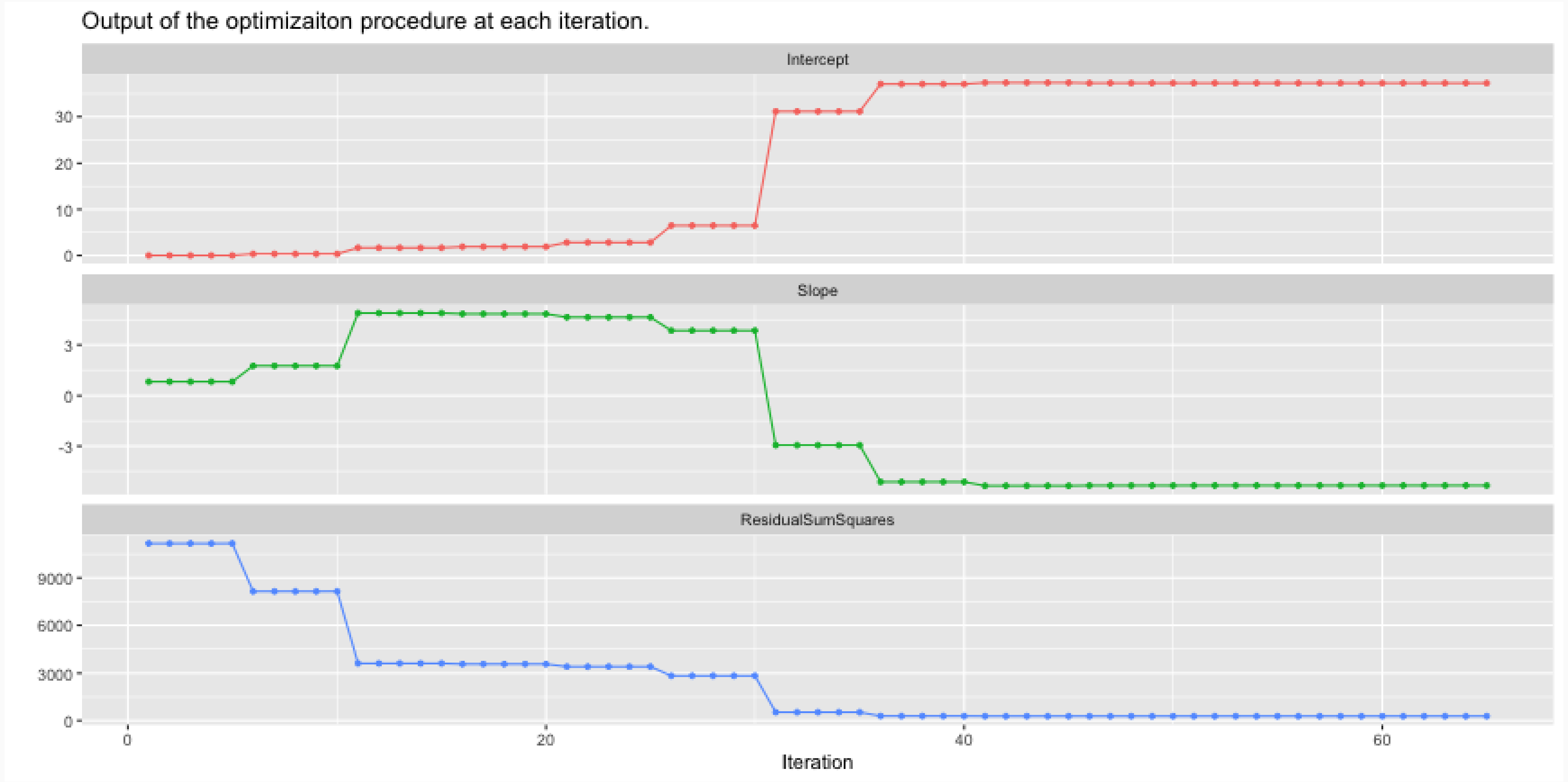
```
lm.out <- lm(mpg ~ wt, data = mtcars)
lm.out$coefficients
```

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

EPSY 630
Spring 2021

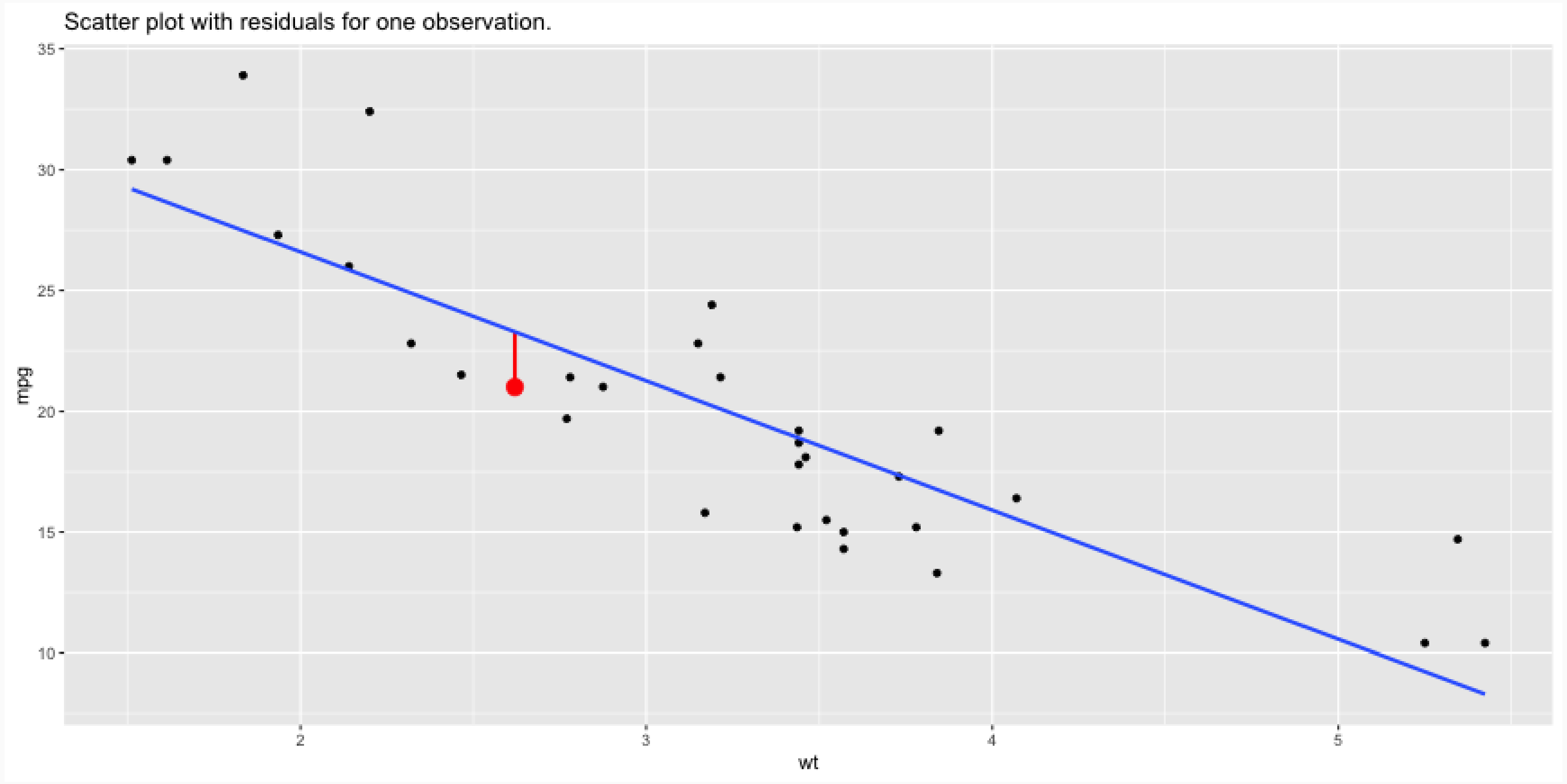
OLS with the `optim` function

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.



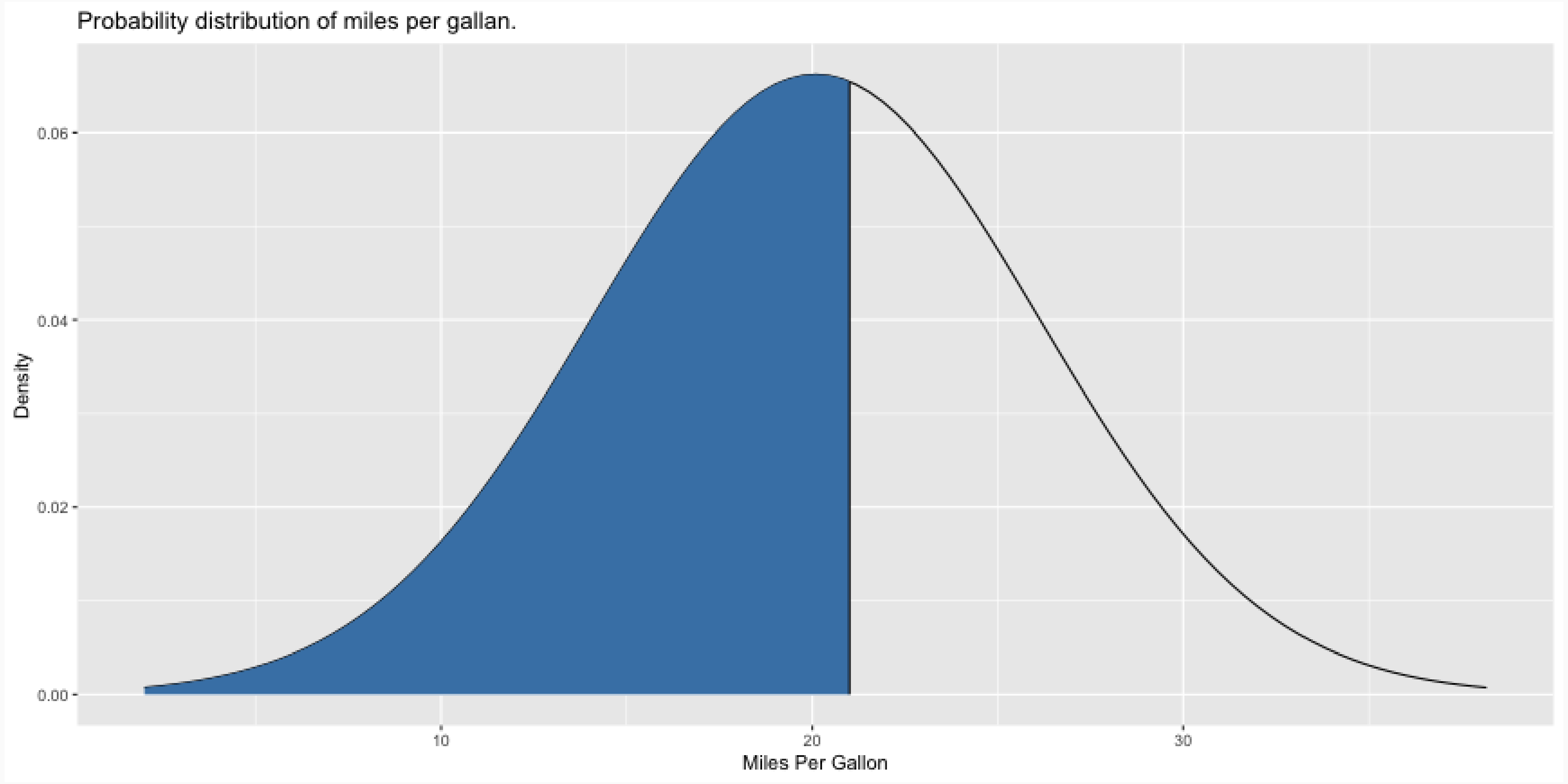
Residuals to Likelihoods

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:

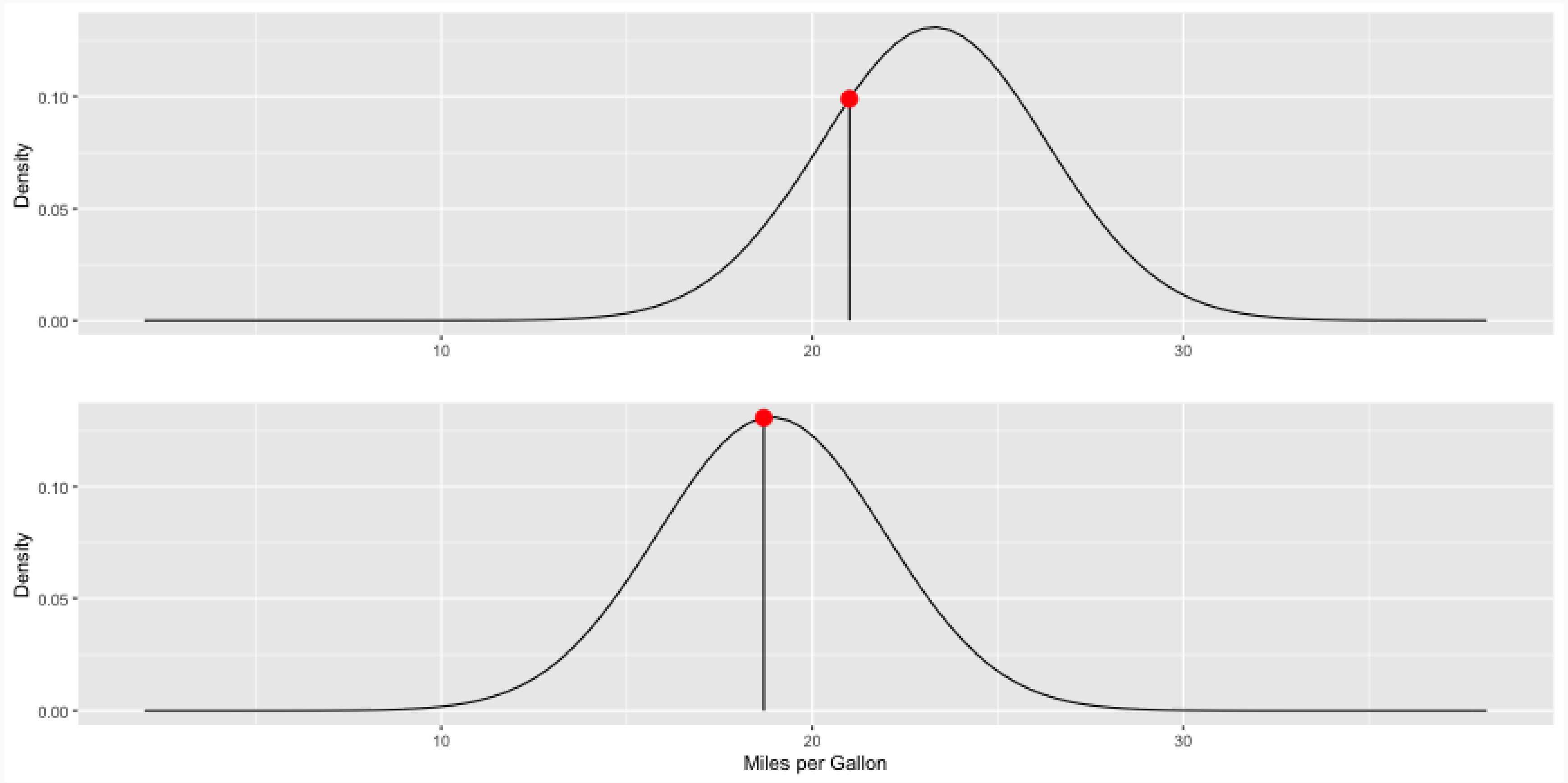
$$pr(data \mid distribution)$$

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:

```r
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.

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

```
## [1] -91.06374
```

EPSY 630 Spring 2021

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).

```r
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
```

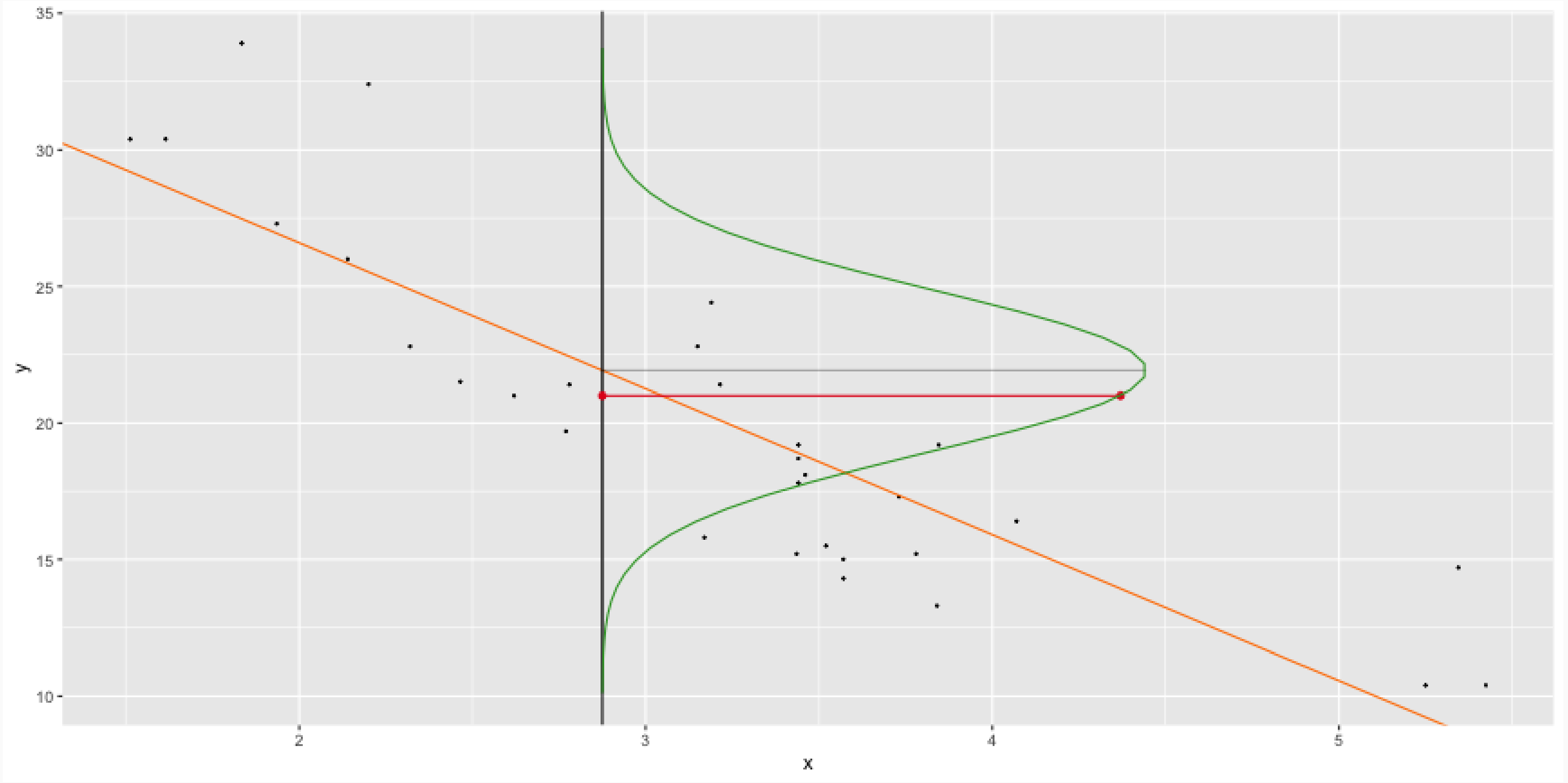
```
## (Intercept)          wt
##   37.285126   -5.344472
```

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.

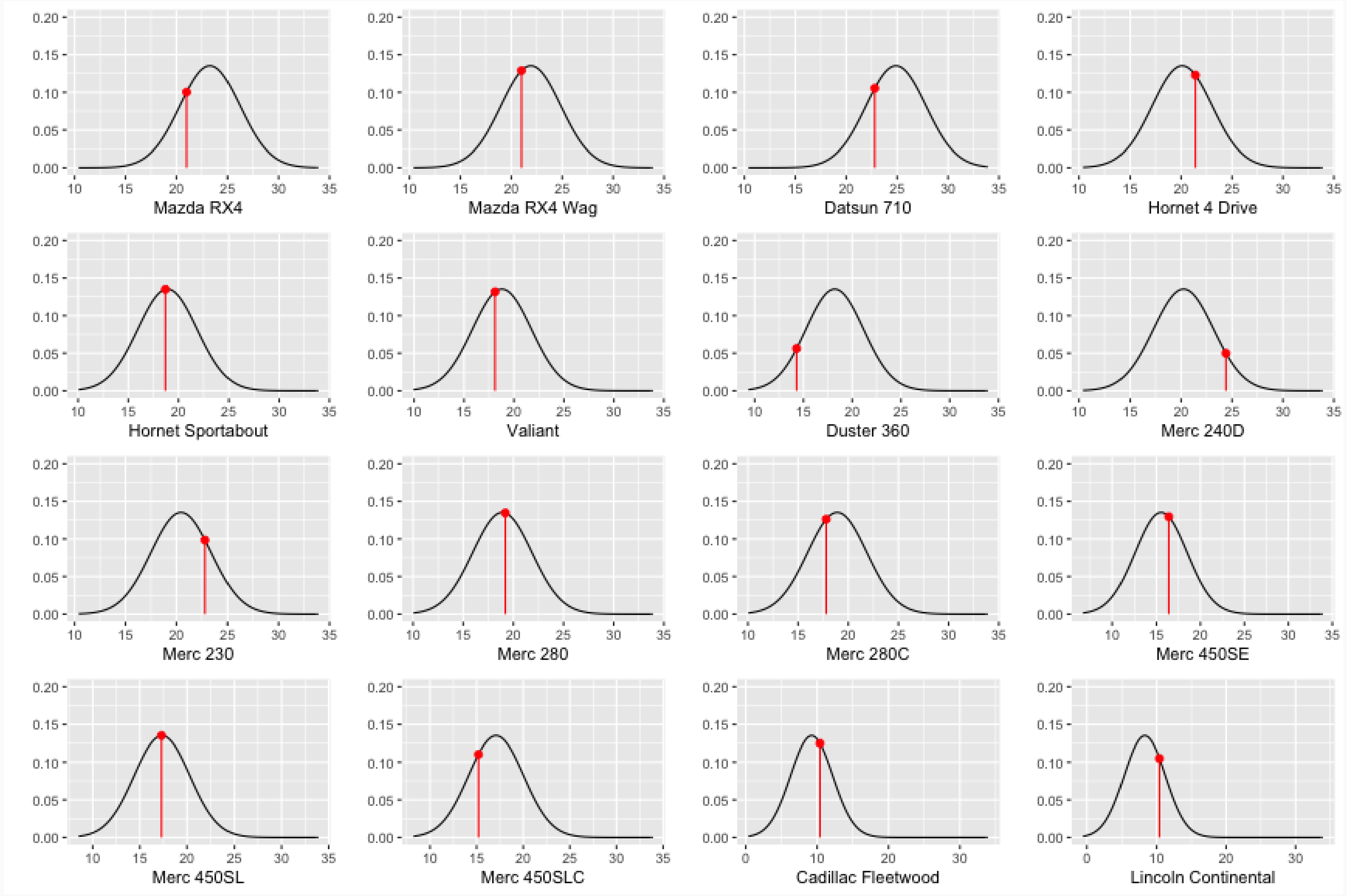
Likelihood Visualized

```r
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])
```



EPSY 630 Spring 2021

Likelihood Visualized



EPSY 630
Spring 2021

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.

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

```
## [1] 2.949162
```

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

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

One Minute Paper

Complete the one minute paper: https://forms.gle/yB3ds6MYE89Z1pURA

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