Resampling Methods

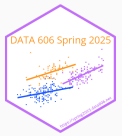
IS382 - Predictive Modeling

Jason Bryer, Ph.D.

Week 6

Bootstrapping

- First introduced by Efron (1979) in *Bootstrap Methods: Another Look at the Jackknife*.
- Estimates confidence of statistics by resampling *with* replacement.
- The *bootstrap sample* provides an estimate of the sampling distribution.
- The `boot` R package provides a framework for doing bootstrapping: https://www.statmethods.net/advstats/bootstrapping.html

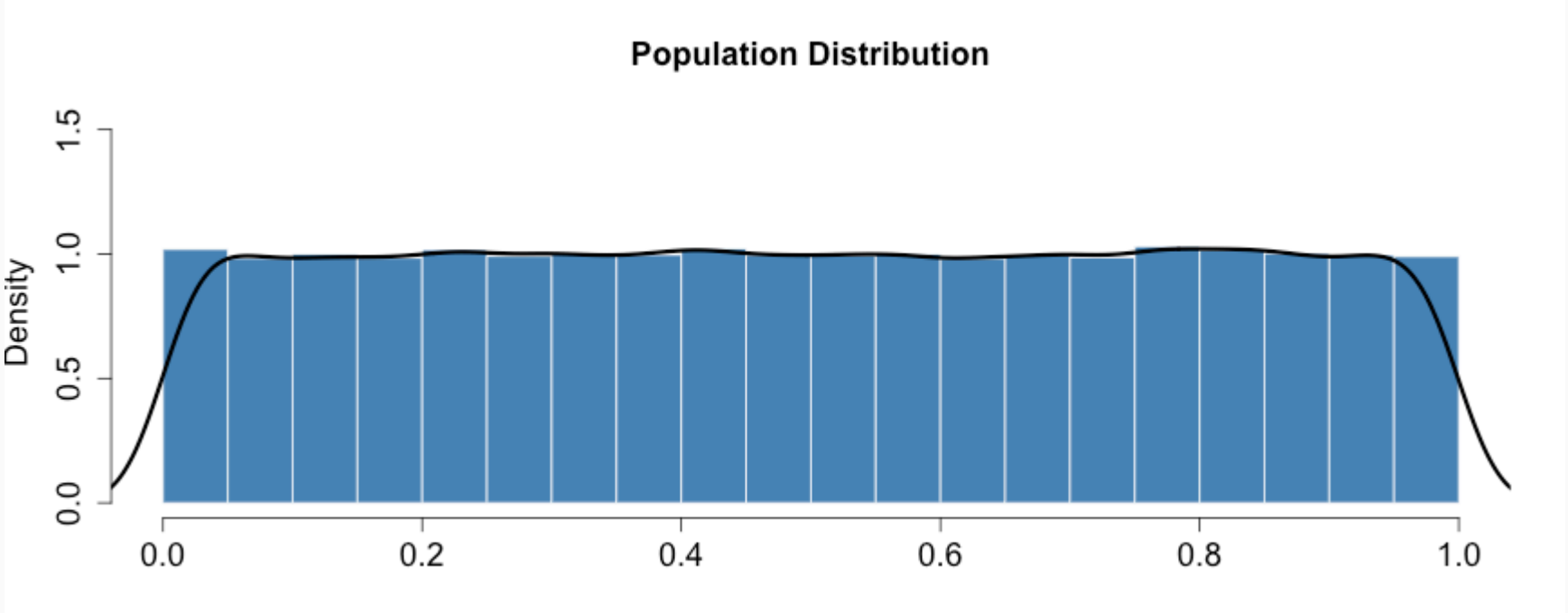


Bootstrapping Example (Population)

Define our population with a uniform distribution.

```
n <- 1e5
pop <- runif(n, 0, 1)
mean(pop)
```

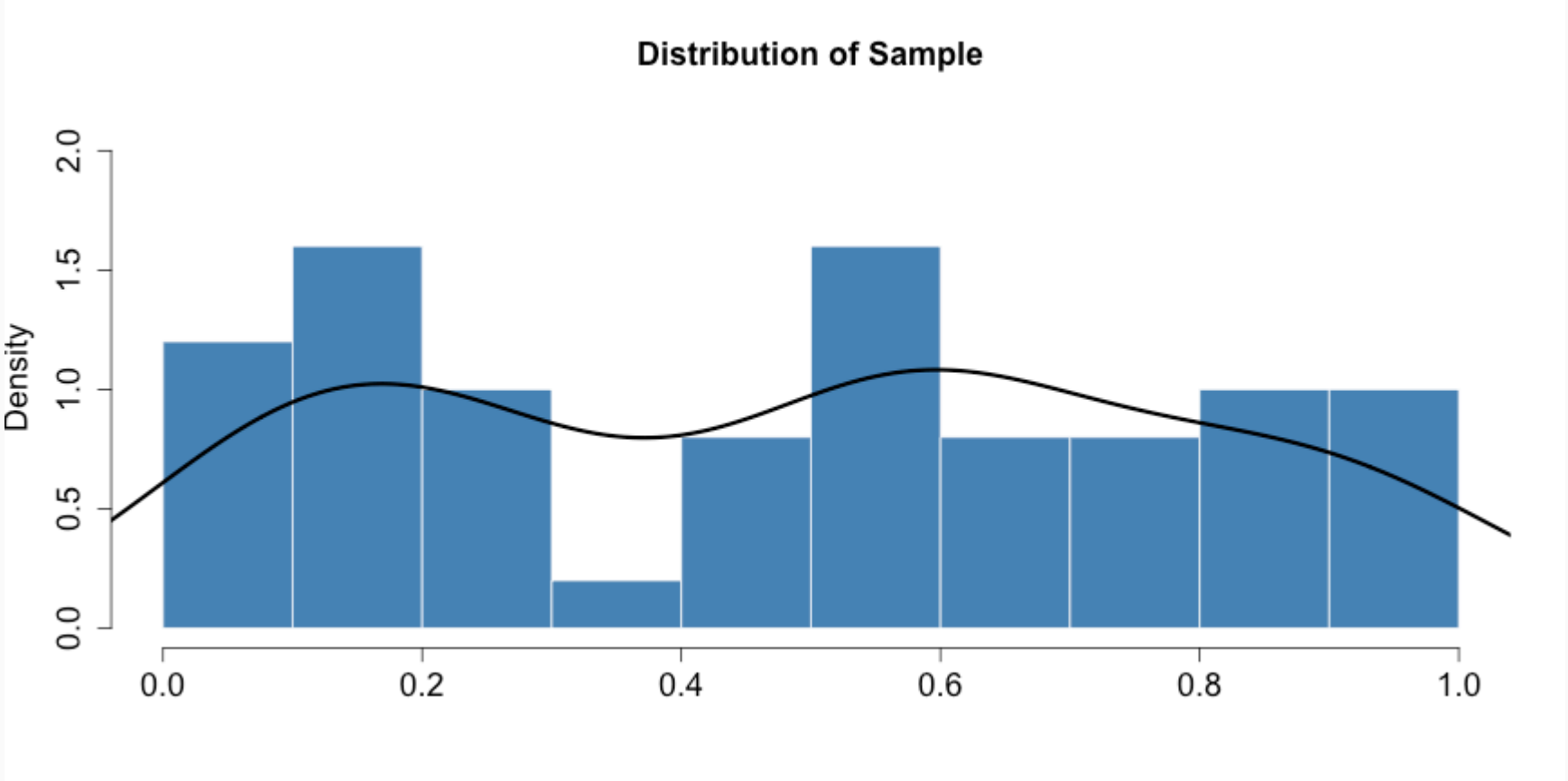
```
## [1] 0.4999781
```



Bootstrapping Example (Sample)

We observe one random sample from the population.

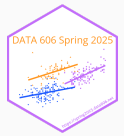
```
samp1 <- sample(pop, size = 50)
```



Bootsrapping Example (Estimate)

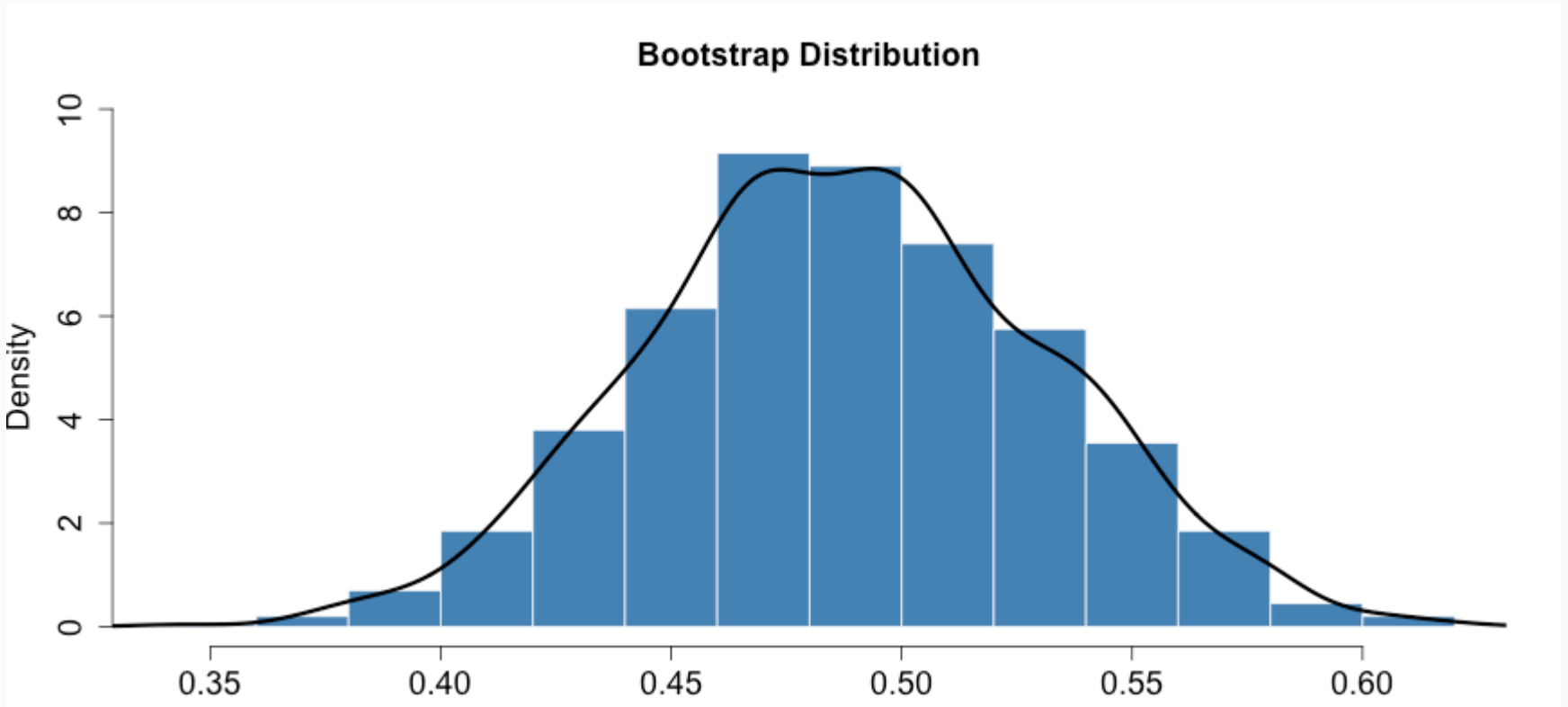
```
boot.samples <- numeric(1000) # 1,000 bootstrap samples
for(i in seq_along(boot.samples)) {
    tmp <- sample(samp1, size = length(samp1), replace = TRUE)
    boot.samples[i] <- mean(tmp)
}
head(boot.samples)
```

```
## [1] 0.4298443 0.5731988 0.4922695 0.4763085 0.5144343 0.4591576
```



Bootsrapping Example (Distribution)

```
d <- density(boot.samples)
h <- hist(boot.samples, plot=FALSE)
hist(boot.samples, main='Bootstrap Distribution', xlab="", freq=FALSE,
     ylim=c(0, max(d$y, h$density)+.5), col='steelblue', border = "white",
     cex.main = 1.5, cex.axis = 1.5, cex.lab = 1.5)
lines(d, lwd=3)
```

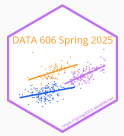


95% confidence interval

```
c(mean(boot.samples) - 1.96 * sd(boot.samples),
  mean(boot.samples) + 1.96 * sd(boot.samples))
```

```
## [1] 0.4034142 0.5727765
```

The standard deviation of the bootstrap sample is an approximation of the standard error.



Bootstrapping is not just for means!

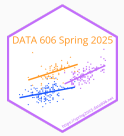
```
boot.samples.median <- numeric(1000) # 1,000 bootstrap samples
for(i in seq_along(boot.samples.median)) {
    tmp <- sample(samp1, size = length(samp1), replace = TRUE)
    boot.samples.median[i] <- median(tmp) # NOTICE WE ARE NOW USING THE median FUNCTION!
}
head(boot.samples.median)
```

```
## [1] 0.5497712 0.5825038 0.4755876 0.2879578 0.3992934 0.5662654
```

95% confidence interval for the median

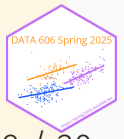
```
c(mean(boot.samples.median) - 1.96 * sd(boot.samples.median),
  mean(boot.samples.median) + 1.96 * sd(boot.samples.median))
```

```
## [1] 0.3731672 0.6617643
```



Boostrap for linear regression (custom function)

```r
lm_boot <- function(formula, data, R = 1000, ...) {
    results <- list()                                                    # Object to save our results
    results$full_model <- lm(formula = formula, data = data, ...)        # Run the full model
    results$coeficients <- data.frame(row.names = 1:R)                   # Data frame to store the coeficients
    for(i in names(results$full_model$coefficients)) {
        results$coeficients[,i] <- NA
    }
    results$fit <- data.frame(row.names = 1:R,                           # Data frame for the fit statistics
                              F_statistic = rep(NA, R),
                              df_numerator = rep(NA, R),
                              df_denominator = rep(NA, R),
                              R_squared = rep(NA, R),
                              Adjusted_R_squared = rep(NA, R))
    results$sample <- list()                                             # This will contain the vectors for
    for(i in seq_len(R)) {                                               # for the bootstrap samples.
        results$sample <- sample(nrow(data), size = nrow(data), replace = TRUE)
        boot_data <- data[results$sample,]
        boot_lm_out <- lm(formula = formula, data = boot_data, ...)      # Calling lm with the bootstrap sample
        boot_lm_out_summary <- summary(boot_lm_out)
        results$coeficients[i,] <- boot_lm_out$coefficients              # Saving all the results
        results$fit[i,] <- c(boot_lm_out_summary$fstatistic,
                             boot_lm_out_summary$r.squared,
                             boot_lm_out_summary$adj.r.squared)
    }
    class(results) <- c('lmboot', 'list')                                # Changing the class so we can write
    return(results)                                                      # custom S3 methods for plot and summary
}
```



S3 OOB and "generic" functions

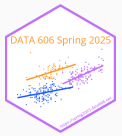
S3 is an objected oriented framework that works by defining conventions. There are many functions that are considered "generic." You can always tell "generic" functions as the implementation calls a function `UseMethod('function_name')` where `function_name` is the name of the generic (e.g. `plot`, `summary`).

```
plot
```

```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0xc31ad5e00>
## <environment: namespace:base>
```

Objects can have different classes. On the previous slide we changed the `class` to `lmboot`. The `UseMethod` will look for a function with the following format: `generic_function.class_name()`.

When we call `plot()` with our result from `lm_boot()`, R will actually call `plot.lm_boot()`.

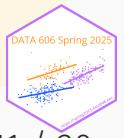


Summary and Plot functions

Here we are defining the S3 methods for `plot` and `summary`.

```
plot.lmboot <- function(x, ...) {
    coef <- reshape2::melt(x$coeficients, id.vars = NULL)
    ggplot2::ggplot(coef, ggplot2::aes(x = value, y = variable)) +
        ggdist::stat_halfeye(side = 'top') +
        ggplot2::scale_y_discrete(limits = rev) +
        ggplot2::xlab('Estimate')
}
```

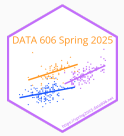
```
summary.lmboot <- function(object, ...) {
    ncols <- ncol(object$coeficients)
    summary_table <- data.frame(
        row.names = names(object$coeficients),
        boot_mean = rep(NA, ncols),
        boot_se = rep(NA, ncols),
        boot_median = rep(NA, ncols))
    for(i in row.names(summary_table)) {
        summary_table[i,] <- c(
            mean(object$coeficients[,i]),
            sd(object$coeficients[,i]),
            median(object$coeficients[,i])
        )
    }
    summary_table <- cbind(
        summary(object$full_model)$coefficients[,1:2],
        summary_table) |> as.data.frame()
    cat('Coefficients:\n')
    print(summary_table)
    cat('\nR-Squared:', mean(object$fit$R_squared),
        '(se =', sd(object$fit$R_squared), ')\n')
    cat('Adjusted R-Squared:', mean(object$fit$Adjusted_R_squared),
        '(se =', sd(object$fit$Adjusted_R_squared), ')')
    invisible(summary_table)
}
```



Using our custom functions

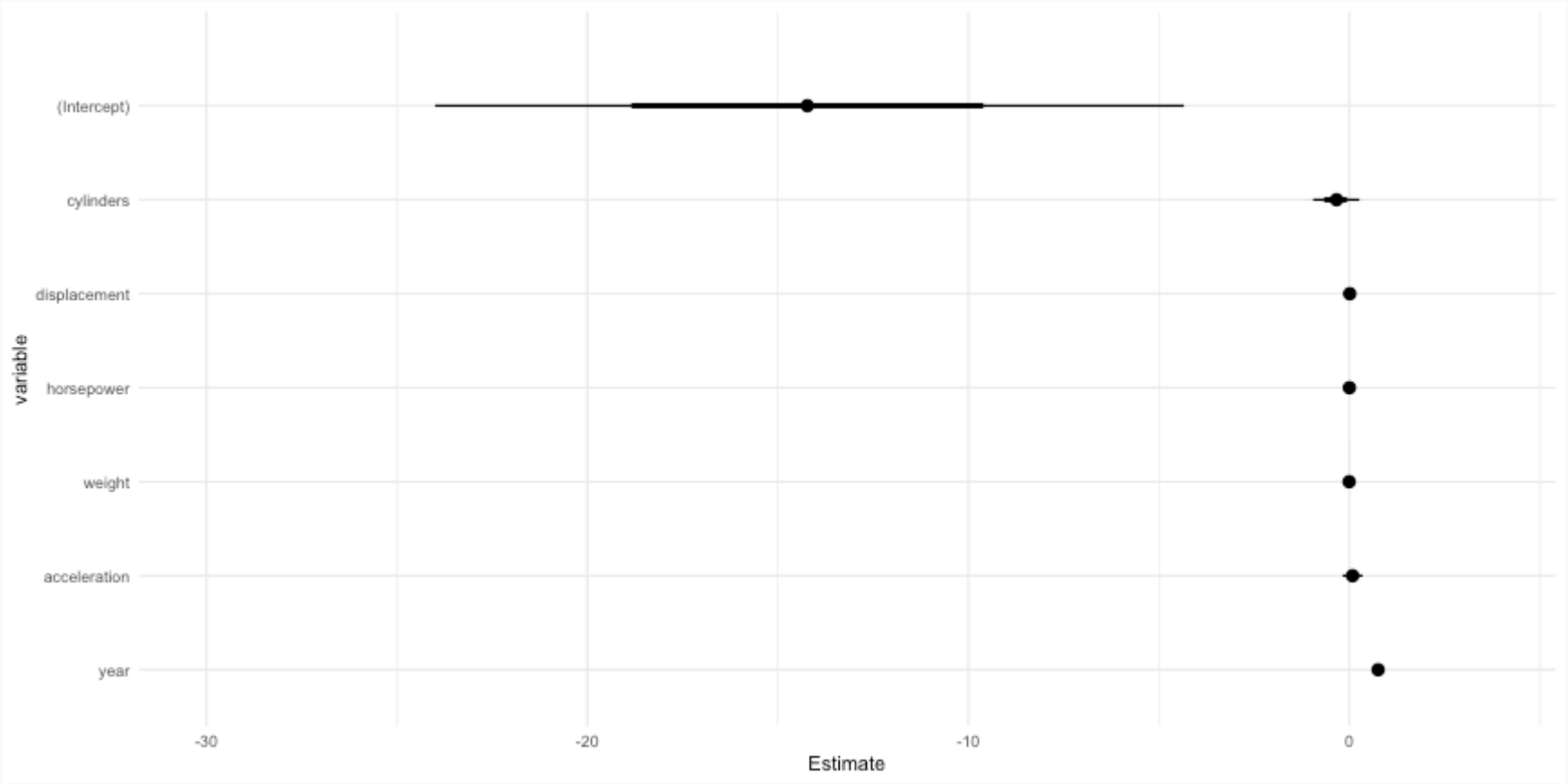
```
data('Auto', package = 'ISLR2')
boot_out <- lm_boot(mpg ~ cylinders + displacement + horsepower + weight + acceleration + year, data = Auto)
summary(boot_out)
```

```
## Coefficients:
##                  Estimate   Std. Error     boot_mean      boot_se   boot_median
## (Intercept)  -1.453525e+01 4.7638818940 -1.418375e+01 4.9545700296 -1.422827e+01
## cylinders    -3.298591e-01 0.3321041332 -3.472660e-01 0.3114523184 -3.382069e-01
## displacement  7.678430e-03 0.0073577361  7.924305e-03 0.0083969321  8.254807e-03
## horsepower   -3.913556e-04 0.0138365215 -2.700511e-04 0.0147399757  1.549855e-04
## weight       -6.794618e-03 0.0006700227 -6.817239e-03 0.0008219454 -6.769501e-03
## acceleration  8.527325e-02 0.1020355670  7.979797e-02 0.1333355924  8.257806e-02
## year          7.533672e-01 0.0526181480  7.509873e-01 0.0519976298  7.525952e-01
##
## R-Squared: 0.8126992 (se = 0.01358161 )
## Adjusted R-Squared: 0.8097803 (se = 0.01379327 )
```



Plot function

```
plot(boot_out)
```

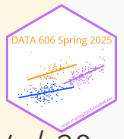


Bootstrap for linear regression (using `car::Boot`)

There are other frameworks for bootstrapping including the `boot` package and the `Boot()` function in the `car` package. First, we estimate the full model.

```
lm_out <- lm(mpg ~ cylinders + displacement + horsepower + weight + acceleration + year, data = Auto)
summary(lm_out)
```

```
##
## Call:
## lm(formula = mpg ~ cylinders + displacement + horsepower + weight +
##     acceleration + year, data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.6927 -2.3864 -0.0801  2.0291 14.3607
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.454e+01  4.764e+00  -3.051  0.00244 **
## cylinders    -3.299e-01  3.321e-01  -0.993  0.32122
## displacement  7.678e-03  7.358e-03   1.044  0.29733
## horsepower   -3.914e-04  1.384e-02  -0.028  0.97745
## weight       -6.795e-03  6.700e-04 -10.141  < 2e-16 ***
## acceleration  8.527e-02  1.020e-01   0.836  0.40383
## year          7.534e-01  5.262e-02  14.318  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```



Bootstrap for linear regression (using `car::Boot`)

Then we pass the results to the `car::Boot()` function.

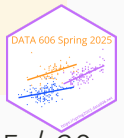
```
set.seed(2112)
lm_boot <- car::Boot(lm_out)
summary(lm_boot)
```

```
##
## Number of bootstrap replications R = 999
##                 original    bootBias     bootSE     bootMed
## (Intercept)  -1.4535e+01 -1.7005e-01 4.84505716 -14.5345349
## cylinders    -3.2986e-01  1.1551e-02 0.31583530  -0.3241365
## displacement  7.6784e-03 -3.6095e-05 0.00822394   0.0077617
## horsepower   -3.9136e-04  4.5486e-04 0.01364407  -0.0002457
## weight       -6.7946e-03 -4.4523e-05 0.00078233  -0.0068111
## acceleration  8.5273e-02  1.0281e-03 0.12454698   0.0887860
## year          7.5337e-01  2.3507e-03 0.05357928   0.7554141
```

Instead of using standard error (remember which is the standard deviation of the bootstrap distribution), we can use quantiles, here the 2.5% and 97.5% percentiles. This may be better if the bootstrap distribution does not reasonably approximate the normal distribution (although the CLT indicates it should in most situations).

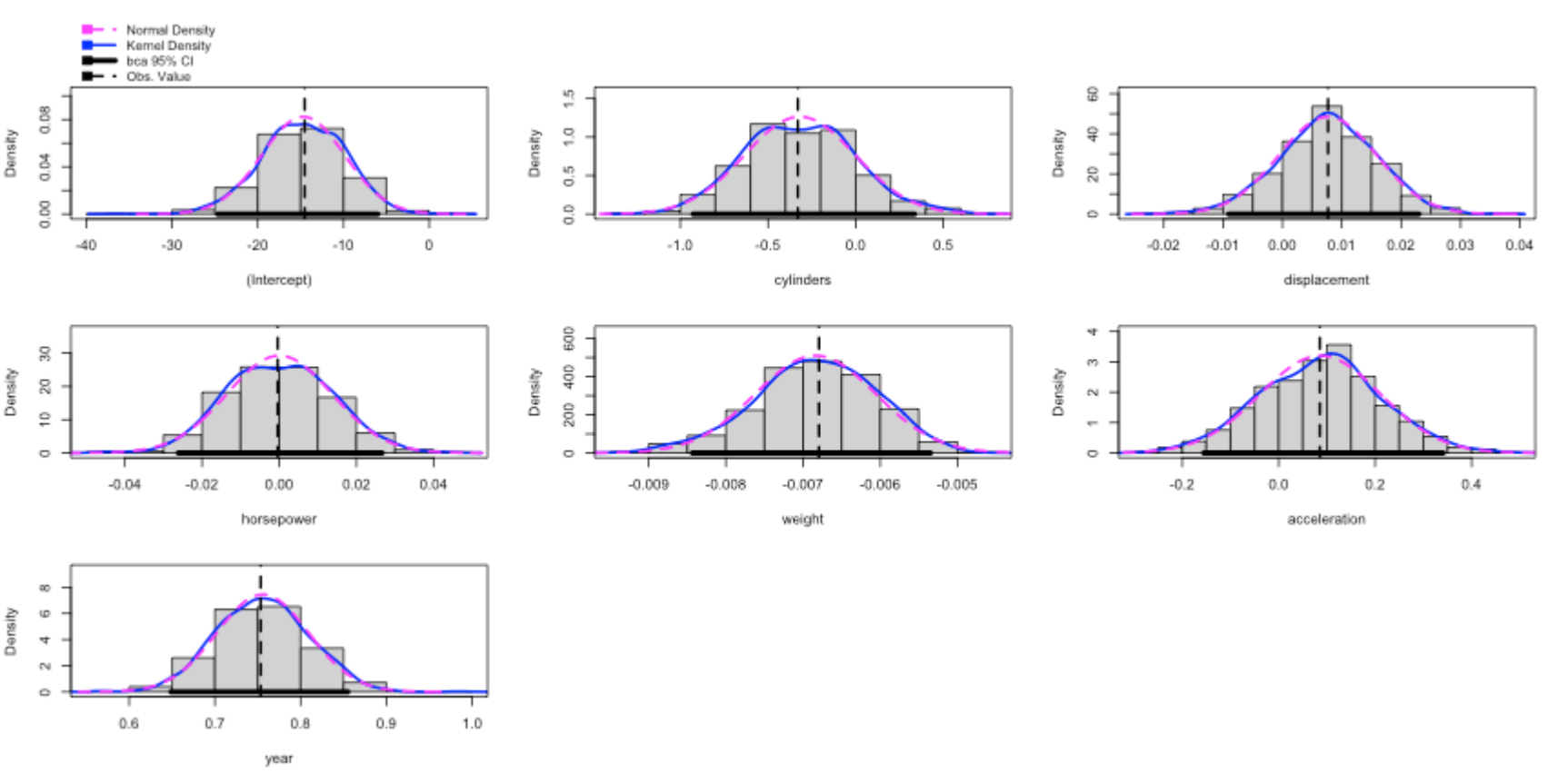
```
confint(lm_boot, level = 0.95, type = "perc")
```

```
## Bootstrap percent confidence intervals
##
##                      2.5 %       97.5 %
## (Intercept)  -24.629496905 -5.868224926
## cylinders     -0.917546064  0.339775915
## displacement  -0.009076681  0.022960679
## horsepower    -0.025587070  0.026677207
## weight        -0.008532166 -0.005420217
## acceleration  -0.158128785  0.333320304
## year           0.652184852  0.856692733
```

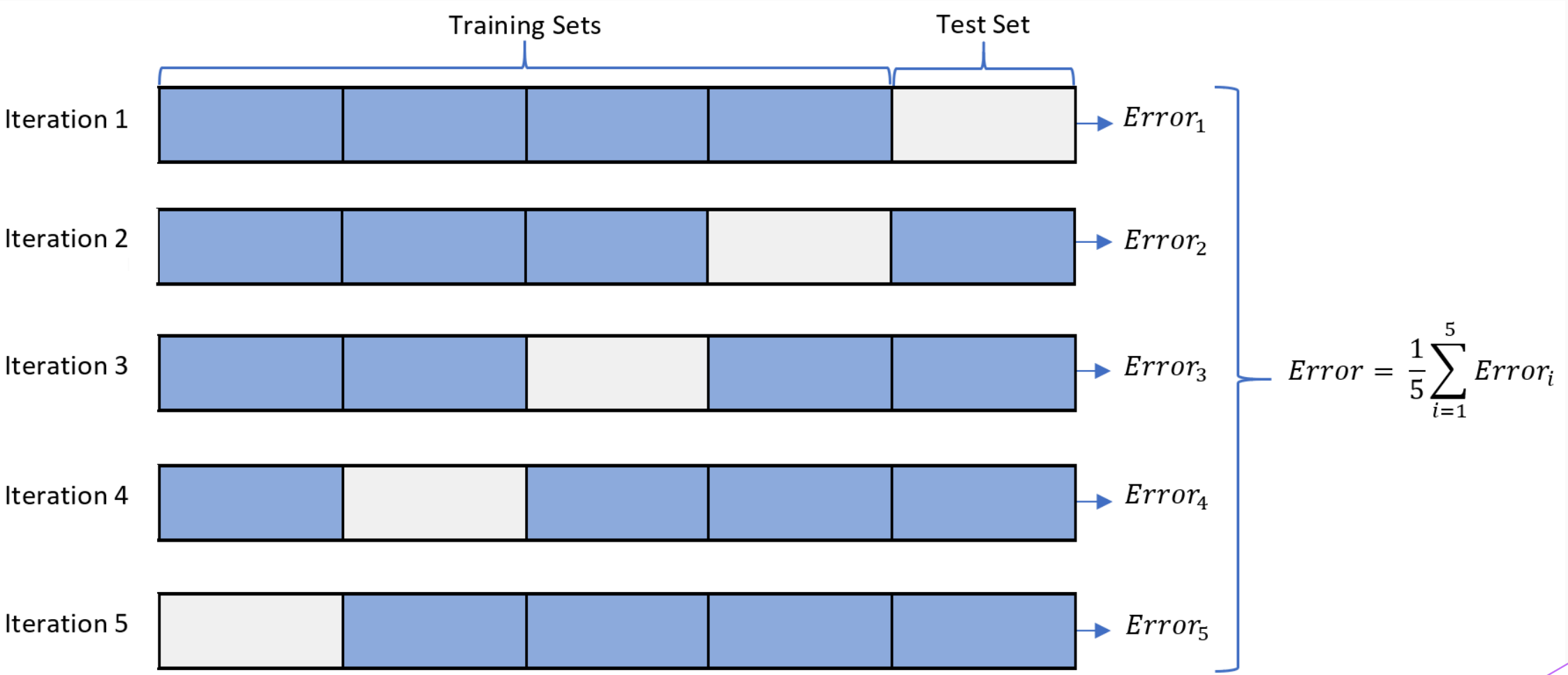


Plotting the results

```
hist(lm_boot)
```



k-fold validation



Using `tidymodels` for k-fold validation

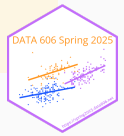
```r
set.seed(2112)
folds <- rsample::vfold_cv(Auto, v = 5)

# Define the model type (e.g., Linear Regression)
lm_spec <- parsnip::linear_reg() |> parsnip::set_engine("lm")

# Fit the model across all resamples
cv_results <- tune::fit_resamples(
  lm_spec,
  preprocessor = mpg ~ cylinders + displacement + horsepower + weight + acceleration + year,
  resamples = folds
)

# Extract evaluation metrics (RMSE, R-squared)
tune::collect_metrics(cv_results)
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   3.49      5  0.276  pre0_mod0_post0
## 2 rsq     standard   0.804     5  0.0118 pre0_mod0_post0
```

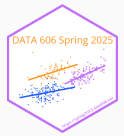


Recommendations to Prevent Overfitting

1. Split your dataset into a training and validation dataset (typically 70/30 works, but can vary depending on sample size).

2. Use k-fold validation when fitting different models (k = 10 is pretty common). This can be different modeling strategies (e.g. regression, trees, etc.) or different variable sets.

3. Once you decide on the best model in step 3, estimate your model fit statistics with the validation dataset created (and set aside) in step 1.



One Minute Paper

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

2. What important question remains unanswered for you?

Scan the QR code or click here to complete the One Minute Paper