

# Effort Titration

Colin Dassow

## Contents

<b>1</b>	<b>WDNR Creel Titration</b>	<b>1</b>
1.1	Current Creel Data as Baseline . . . . .	1

## 1 WDNR Creel Titration

### 1.1 Current Creel Data as Baseline

DNR has amassed a large amount of inland creel information over many decades (1984, 2023) across many waterbodies (n=317). Using this data and functionality of `wdnr.fmdb` it is possible to calculate effort, catch, harvest, and harvest rate for multiple species for each unique creel survey conducted. This information can be further grouped by month, season, day type, etc. to provide insight into important temporal patterns in the fisheries metrics of interest. These calculations are accomplished fairly easily using the following code:

```
# this code is how one would normally pull creel data from
# the WDNR database using the wdnr.fmdb package, this would
# provide the user for the most up to date creel data
# available. For this study we did an initial pull and then
# save the result to keep it constant throughout the study.

cserv = get_creel_surveys()
cvis = get_creel_visits()
ccou = get_creel_counts()
cint = get_creel_int_party()
cfish = get_creel_fish_data()

# these functions from wdnr.fmdb calculate useful metrics
# from creel data

ceff = calc_creel_effort(creel_count_data = ccou, creel_int_data = cint)

charv = calc_creel_harvest(creel_count_data = ccou, creel_int_data = cint,
                           creel_fish_data = cfish)

charvR = calc_creel_harvest_rates(creel_fish_data = cfish)
```

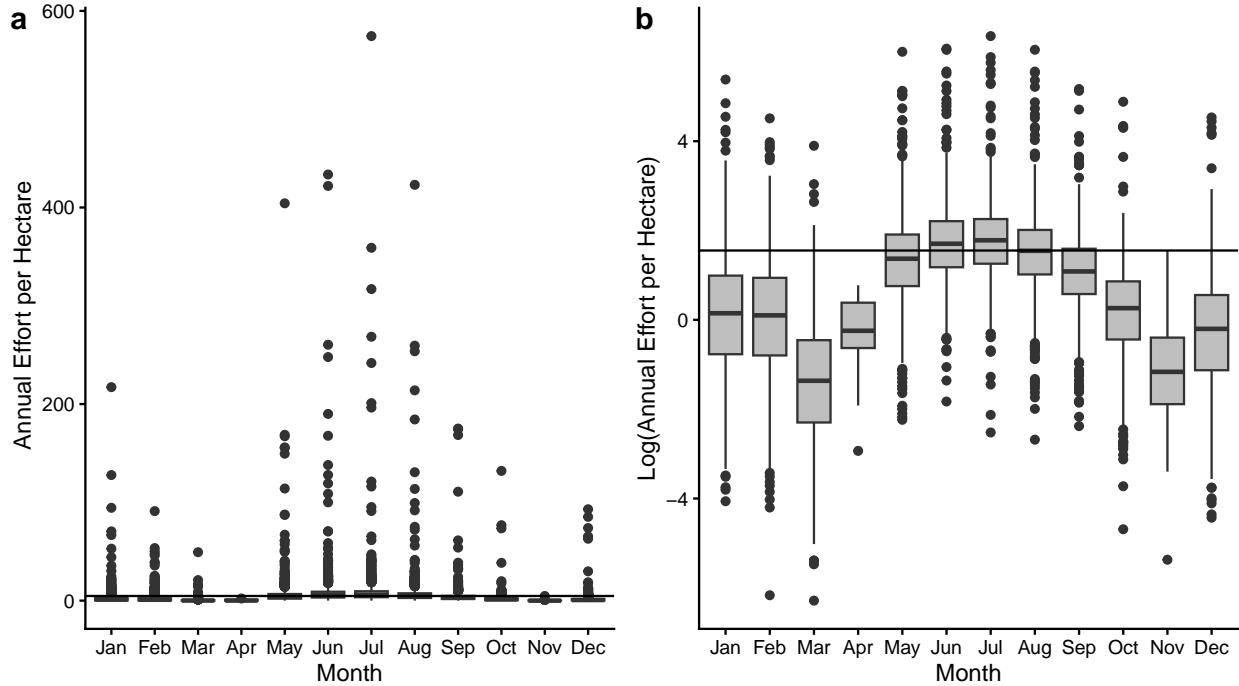


Figure 1: Distribution of effort estimates by month (panel a) for all years of CTWI creel data. Horizontal line is overall mean effort. Black points are outliers. Panel B is the same information presented on a log-scale.

### 1.1.1 Bayesian Model Fitting

Individual likelihoods were constructed for each reduced data set and fit using the functions in the `BayesianTools` package in R. These model fits were checked for convergence using visual inspection of trace plots, posterior distributions, and Gelman-Rubin Diagnostic tests to ensure that models had converged. All models were fit using Differential Evolution Markov-Chain-Monte-Carlo algorithms run for 10,000 iterations with the first 5,000 discarded as burn-in.

```
# one likelihood to estimate parms for using the 6
# different treatments creating data frame to model with
# effort estimates

# removing 0s since they can't be logged and if I were to
# make them a small number they would throw off the data
# and make it bimodal which would probably mean switching
# to a gamma distribution to model the data. There are 8
# 0's accounting for 1% of the data. I'm going to operate
# under the assumption that if a survey gives a 0 effort
# estimate with the full survey, then efs collecting less
# data isn't going to change that number.
zeros.actual = sum(ttrExp$total.eff[ttrExp$treat == "actual"] ==
  0)
zeros.nw = sum(ttrExp$total.eff[ttrExp$treat == "noWinter"] ==
  0)
zeros.ma = sum(ttrExp$total.eff[ttrExp$treat == "mayAug"] ==
  0)
zeros.wd25 = sum(ttrExp$total.eff[ttrExp$treat == "wd25"] ==
```

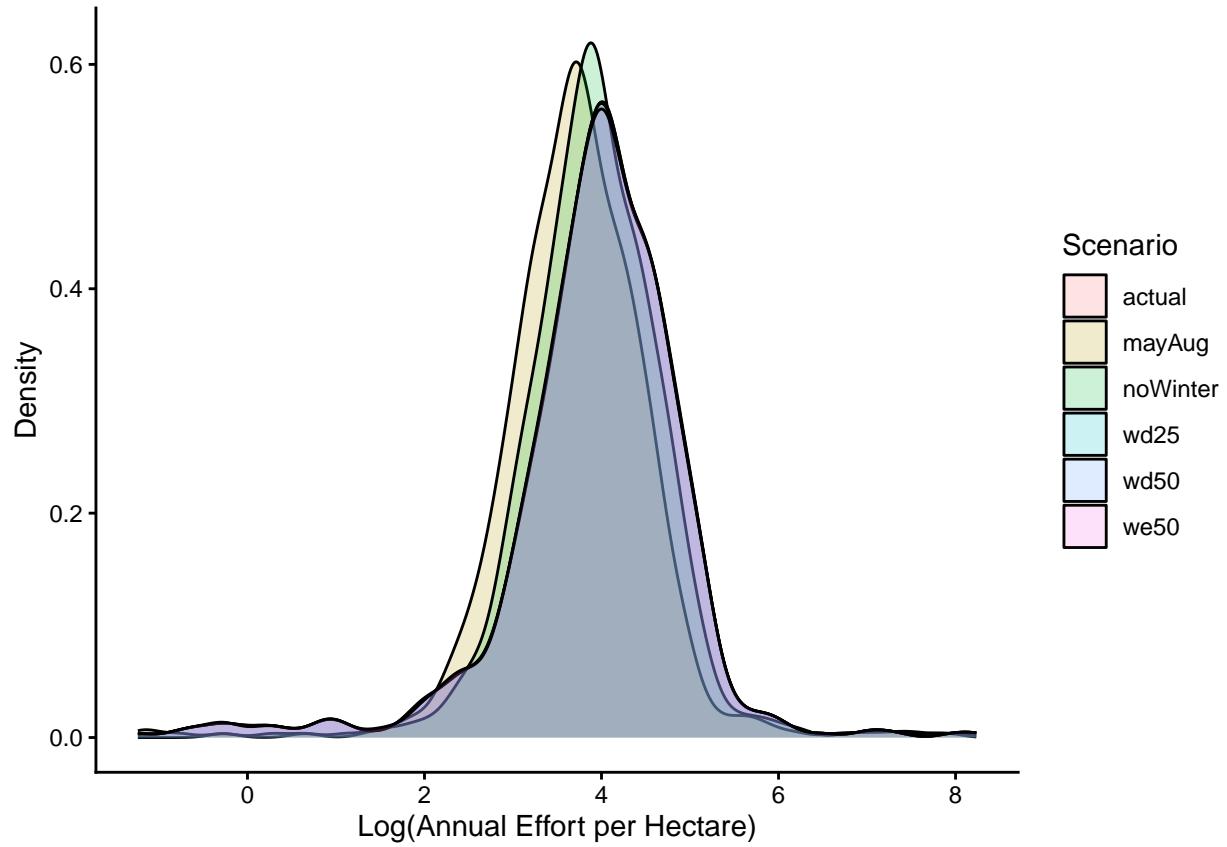


Figure 2: Distribution, on a log scale, of effort for the full data set plus 5 scenarios with reduced data. Actual = full data set, mayAug = May to August creel data only, noWinter = winter creel data removed, wd25 = 25% of weekday creel visits per month removed, wd50 = 50% of weekday creel visits per month removed, we50 = 50% of weekend creel visits per month removed.

Table 1: Table of the number of surveys with effort estimates equal to 0 for each data scenario.

Scenario	Zeros
Actual	8
No Winter	16
May-August	6
25% Weekday Reduction	8
50% Weekday Reduction	8
50% Weekend Reduction	8

```

 0)
zeros.wd50 = sum(ttrExp$total.eff[ttrExp$treat == "wd50"] ==
 0)
zeros.we50 = sum(ttrExp$total.eff[ttrExp$treat == "we50"] ==
 0)

Ztab = data.frame(Scenario = c("Actual", "No Winter", "May-August",
"25% Weekday Reduction", "50% Weekday Reduction", "50% Weekend Reduction"),
Zeros = c(zeros.actual, zeros.nw, zeros.ma, zeros.wd25, zeros.wd50,
zeros.we50))
kable(Ztab, format = "latex", caption = "Table of the number of surveys with effort estimates equal to 0")

```

```

# removing 0s here as described in the text.
modDat = ttrExp[ttrExp$total.eff != 0, ]

##### ACTUAL #####
effLL.a = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(modDat[modDat$treat == "actual", ]), meanlog = alpha,
    sdlog = beta)
  ll = dlnorm(modDat$total.eff[modDat$treat == "actual"], meanlog = mean(log(efs)),
    sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

prior = createTruncatedNormalPrior(mean = c(mean(log(modDat$total.eff[modDat$treat ==
"actual"]))), sd(log(modDat$total.eff[modDat$treat == "actual"])),
sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.actual = createBayesianSetup(effLL.a, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
burnin = 5000)
set.seed(10)
eff.actual = runMCMC(bayesianSetup = setup.actual, sampler = "DEzs",
settings = settings) # takes about 10 seconds

##### NW #####
effLL.nw = function(param) {

```

```

alpha = param[1]
beta = param[2]

efs = rlnorm(nrow(modDat[modDat$treat == "noWinter", ]),
  meanlog = alpha, sdlog = beta)
ll = dlnorm(modDat$total.eff[modDat$treat == "noWinter"],
  meanlog = mean(log(efs)), sdlog = sd(log(efs)), log = T)
return(sum(ll))
}

prior = createTruncatedNormalPrior(mean = c(mean(log(modDat$total.eff[modDat$treat ==
  "noWinter"])), sd(log(modDat$total.eff[modDat$treat == "noWinter"]))),
  sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.nw = createBayesianSetup(effLL.nw, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
  burnin = 5000)
set.seed(10)
eff.nw = runMCMC(bayesianSetup = setup.nw, sampler = "DEzs",
  settings = settings)

##### MA #####
effLL.ma = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(modDat[modDat$treat == "mayAug", ]),
    meanlog = alpha,
    sdlog = beta)
  ll = dlnorm(modDat$total.eff[modDat$treat == "mayAug"], meanlog = mean(log(efs)),
    sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

prior = createTruncatedNormalPrior(mean = c(mean(log(modDat$total.eff[modDat$treat ==
  "mayAug"])), sd(log(modDat$total.eff[modDat$treat == "mayAug"]))),
  sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.ma = createBayesianSetup(effLL.ma, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
  burnin = 5000)
set.seed(10)
eff.ma = runMCMC(bayesianSetup = setup.ma, sampler = "DEzs",
  settings = settings)

##### WD25 #####
effLL.wd25 = function(param) {
  alpha = param[1]
  beta = param[2]

```

```

efs = rlnorm(nrow(modDat[modDat$treat == "wd25", ]), meanlog = alpha,
             sdlog = beta)
ll = dlnorm(modDat$total.eff[modDat$treat == "wd25"], meanlog = mean(log(efs)),
            sdlog = sd(log(efs)), log = T)
return(sum(ll))
}

prior = createTruncatedNormalPrior(mean = c(mean(log(modDat$total.eff[modDat$treat ==
"wd25"])), sd(log(modDat$total.eff[modDat$treat == "wd25"]))),
sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.wd25 = createBayesianSetup(effLL.wd25, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
                burnin = 5000)
set.seed(10)
eff.25wd = runMCMC(bayesianSetup = setup.wd25, sampler = "DEzs",
                    settings = settings)

##### WD50 #####
effLL.wd50 = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(modDat[modDat$treat == "wd50", ]), meanlog = alpha,
               sdlog = beta)
  ll = dlnorm(modDat$total.eff[modDat$treat == "wd50"], meanlog = mean(log(efs)),
              sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

prior = createTruncatedNormalPrior(mean = c(mean(log(modDat$total.eff[modDat$treat ==
"wd50"])), sd(log(modDat$total.eff[modDat$treat == "wd50"]))),
sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.wd50 = createBayesianSetup(effLL.wd50, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
                burnin = 5000)
set.seed(10)
eff.50wd = runMCMC(bayesianSetup = setup.wd50, sampler = "DEzs",
                    settings = settings)

##### WE50 #####
effLL.we50 = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(modDat[modDat$treat == "we50", ]), meanlog = alpha,
               sdlog = beta)
  ll = dlnorm(modDat$total.eff[modDat$treat == "we50"], meanlog = mean(log(efs)),
              sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

```

```

}

prior = createTruncatedNormalPrior(mean = c(mean(log(modDat$total.eff[modDat$treat ==
  "we50"])), sd(log(modDat$total.eff[modDat$treat == "we50"]))),
  sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.we50 = createBayesianSetup(effLL.we50, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
  burnin = 5000)
set.seed(10)
eff.50we = runMCMC(bayesianSetup = setup.we50, sampler = "DEzs",
  settings = settings)

```

### 1.1.2 Model Checking

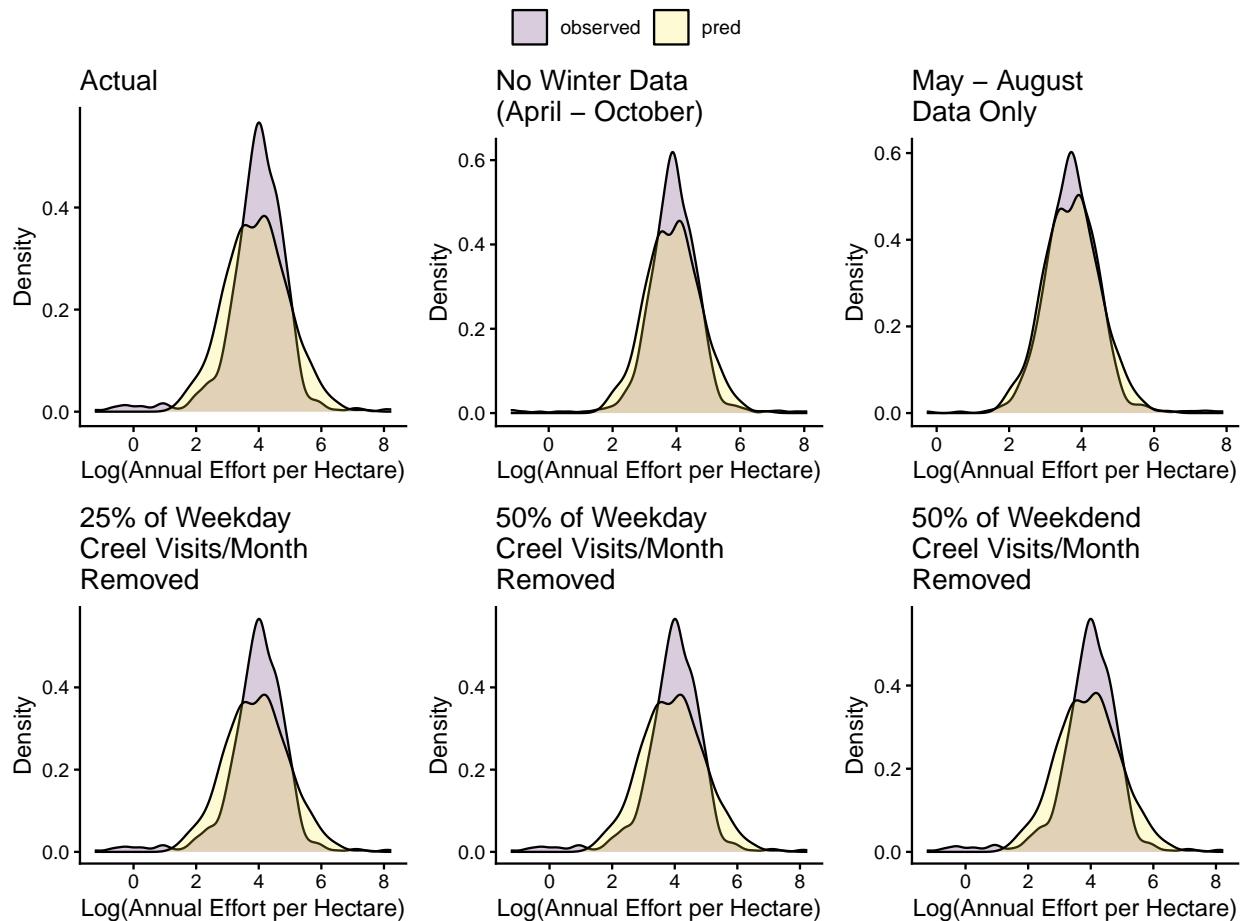


Figure 3: Distributions of the observed and model predicted data for each scenario. Model predicted data comes from lognormal distribution parameterized using the median parameter estimate of the model posterior.

```

gelmanDiagnostics(eff.actual) # converged
gelmanDiagnostics(eff.nw) # converged

```

Table 2: Bayesian p-values for a lognormal distributed model. Each p-value describes whether or not there is a significant difference between data generated by the fitted model and the actual data for that scenario. Two metrics is standard deviation (SD).

Scenario	SD p-value
Actual	0.519
No Winter	0.516
May-August	0.512
25% weeday removal	0.546
50% weekday removal	0.534
50% weekend removal	0.532

Table 3: Bayesian p-values for the comparison between the actual data and the scenario-specific model fit to a lognormal distribution. This test is asking whether the scenario-specific model can approximate the actual data. When true this signals no effect of the data reduction for that scenario on the resulting annual effort per hectare estimate. Standard deviation (SD) was used as the variance metric here.

Scenario	SD p-value
Actual	0.519
No Winter	0.001
May-August	0.000
25% weeday removal	0.544
50% weekday removal	0.537
50% weekend removal	0.554

```
gelmanDiagnostics(eff.ma) # converged
gelmanDiagnostics(eff.25wd) # converged
gelmanDiagnostics(eff.50wd) # converged
gelmanDiagnostics(eff.50we) # converged
```

### 1.1.3 Inference

Having established that the models are fitting the data well, some inference can be gained as to whether or not the reductions in creel effort proposed here would result in a meaningful change in the total annual effort per acre estimated from the reduced data.

A Bayesian p-value can be employed here too. A test between the sd of the actual data and the sd of the reduced data can describe whether the model of the reduced data is able to approximate the actual data or not. Successful approximations are p-values close to 0.5 with reduced fit as values increase or decrease beyond 0.5. Generally, the rule of thumb is that p-values  $< 0.10$  or  $> 0.90$  signal unacceptable fits. However, these cutoffs can be set based on the objectives of the study at hand and the values of the decision makers.

### 1.1.4 Year Specific Analyses

Instead of modeling the full population of annual effort estimates as was done above, here I have done the model fitting on individual creel-years so that I can see how the removal of data for a creel-year (which is a comparatively large amount of missing data since only 16-20 lakes are creeled each year) impacts the annual effort per acre estimated that year. This analysis will follow the same routine as the whole-population analyses above but only consider one creel-year at a time.

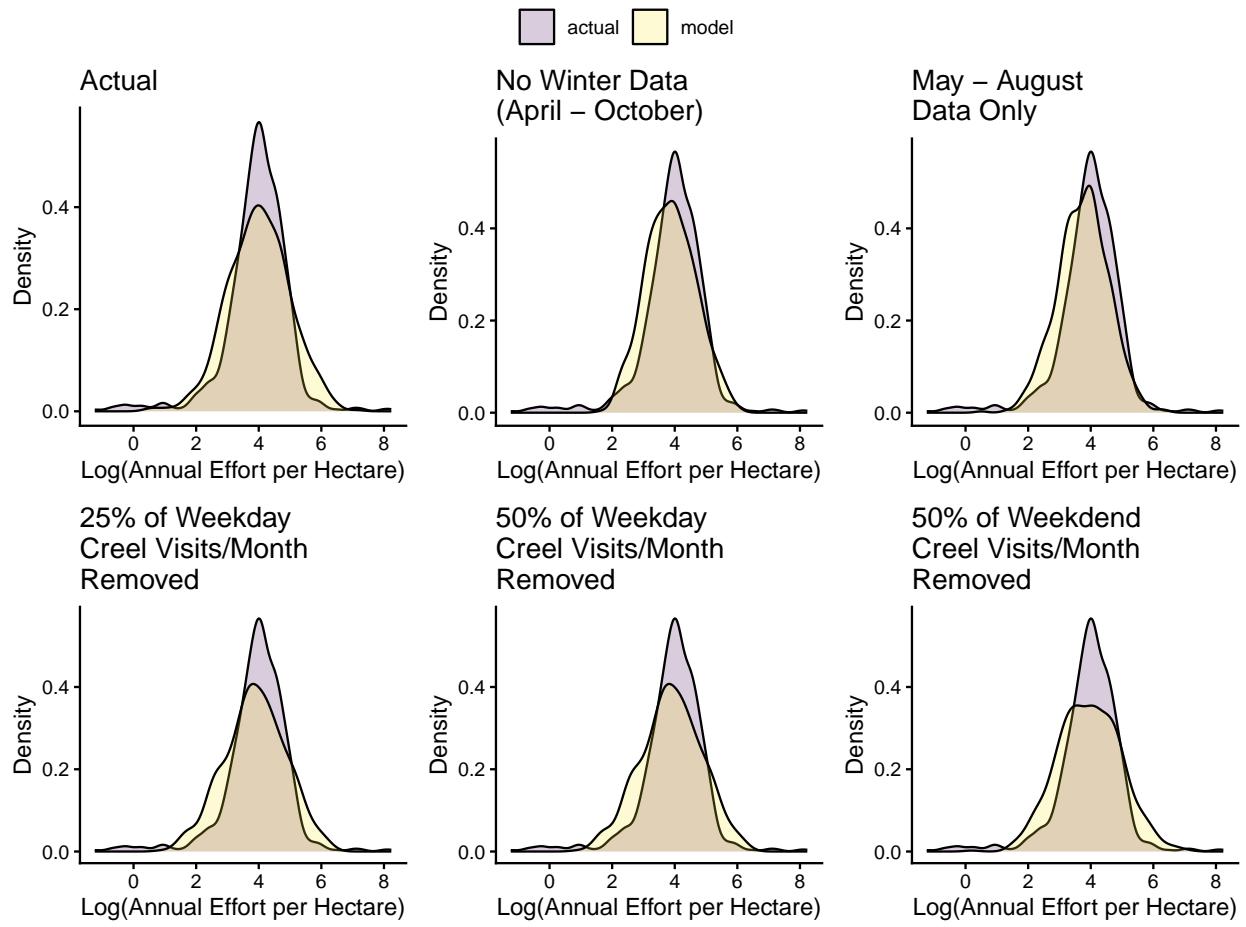


Figure 4: Comparison of the distribution of actual annual effort per hectare estimates calculated from creel data and annual effort per hectare estimated calculated from simulated creel data for each data reduction scenario and the resulting median parameter values for their respective model fits.

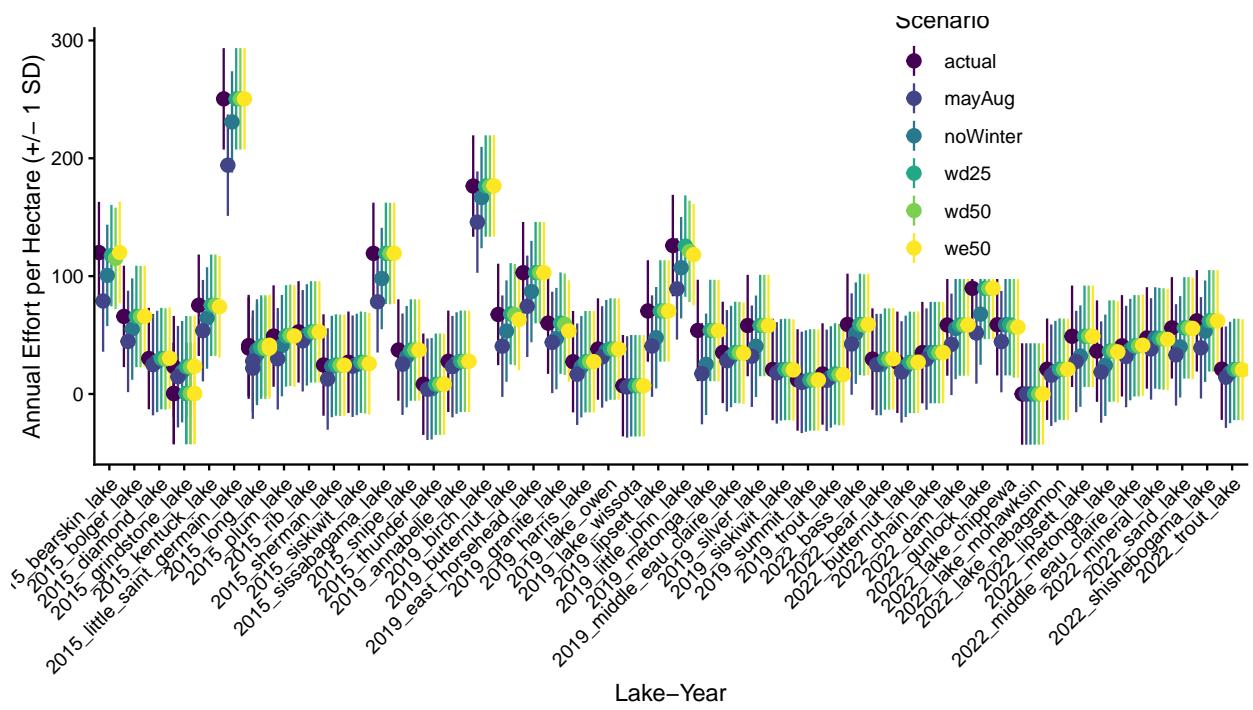


Figure 5: Comparison of the annual effort per hectare estimated from the reduced data for a handful of creel surveys. Most data reductions do not result in annual effort per hectare estimates that are much different from the actual data (most are within 1 SD of the effort estimate for that treatment). Colors represent data reduction scenarios, points are mean annual effort per hectare estimates across all months of that creel survey and vertical lines represent 1 standard deviation.

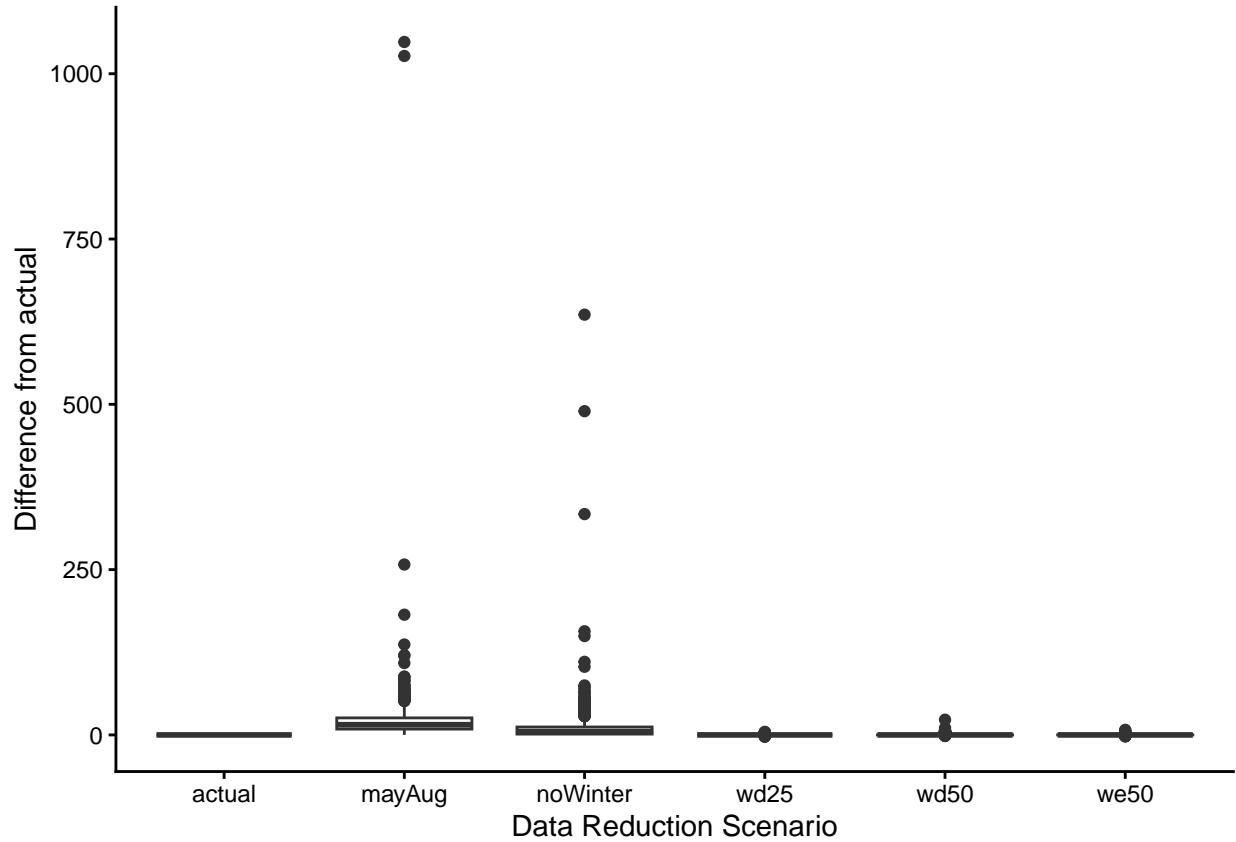


Figure 6: Boxplot of the difference between the actual annual effort per hectare estimate and the data-reduced annual effort per hectare estimate for each creel survey in the creel dataset. Most of the data exhibits differences very near 0 except the seasonal reductions, dots represent outliers ( $x < |> x$ 's percentile- $1.5 \times$ interquartile range)

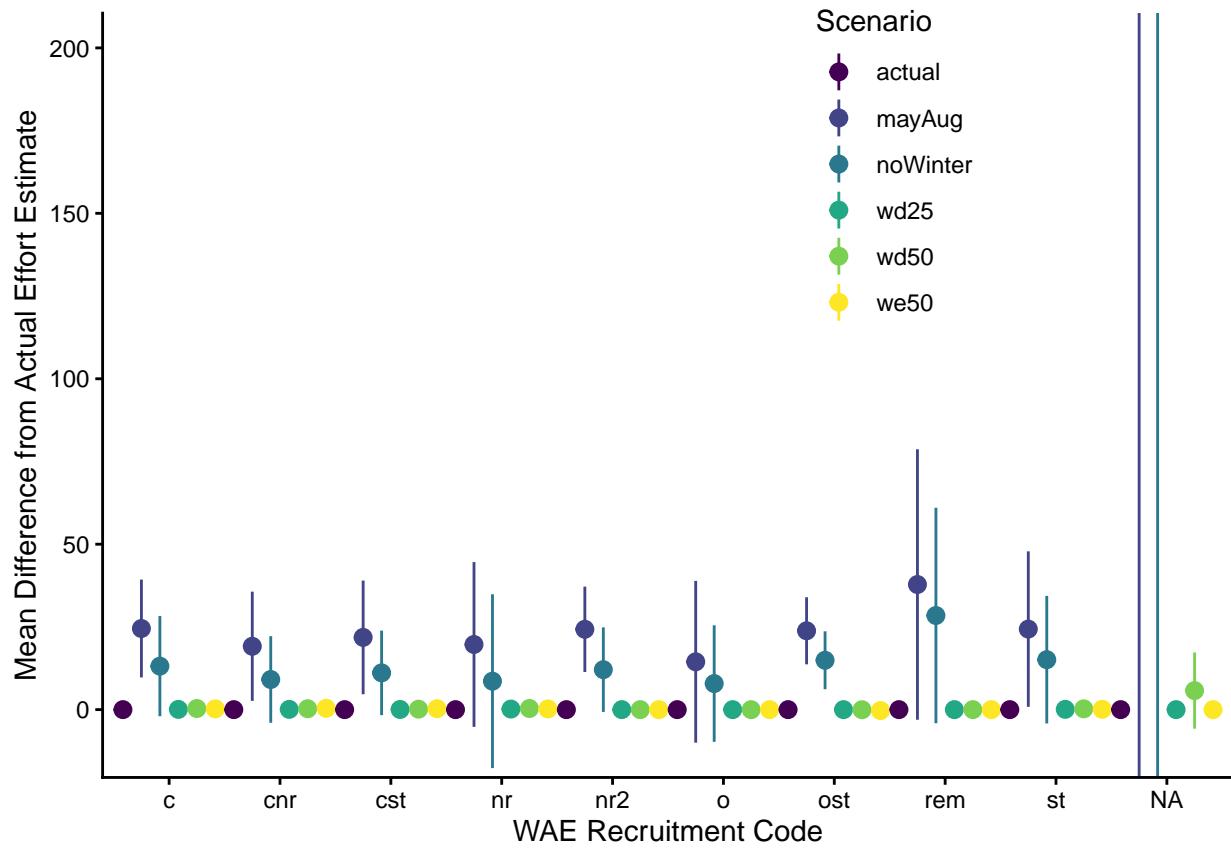


Figure 7: Comparison of the magnitude of the difference between the mean actual annual effort per hectare estimate and the estimate derived from each reduced data set. Vertical lines represent 1 standard deviation. Walleye recruitment codes are along the x axis. Note the y-axis scale, many of the differences are small. Walleye recruitment codes describe the source of young-of-year walleye in the system and are defined as follows: c=combined equal shares stocked and natural source, cnr=combined mainly from natural, cst=combined mainly from stocking, nr=natural reproduction only, ost=stocking only with few/no adult walleye present,, st=stocked only, NA=not evaluated yet.

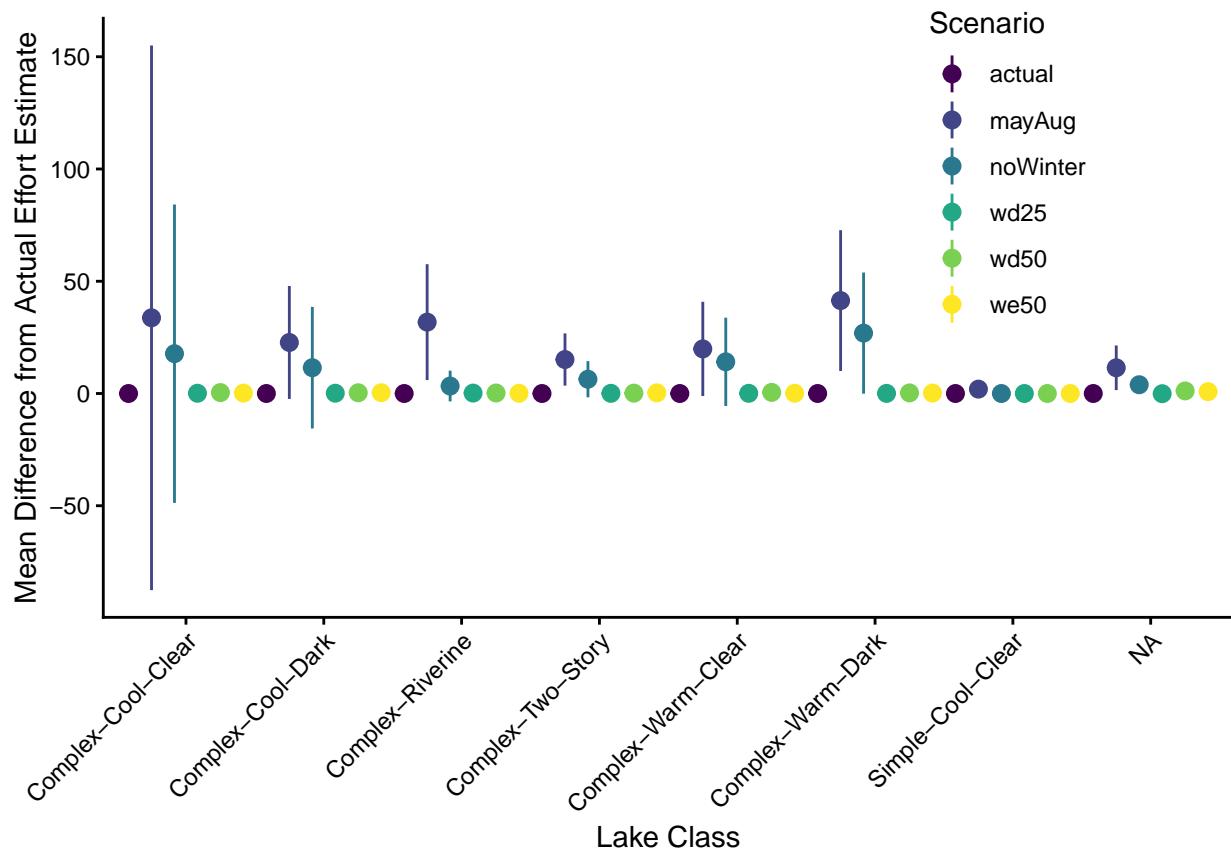


Figure 8: Comparison of the magnitude of the difference between the mean actual annual effort per acre estimate and the estimate derived from each reduced data set. Vertical lines represent 1 standard deviation. Note the y-axis scale, many of the differences are very small. Lake classes are according to Rypel et al. 2019, <https://doi.org/10.1002/fsh.10228>.

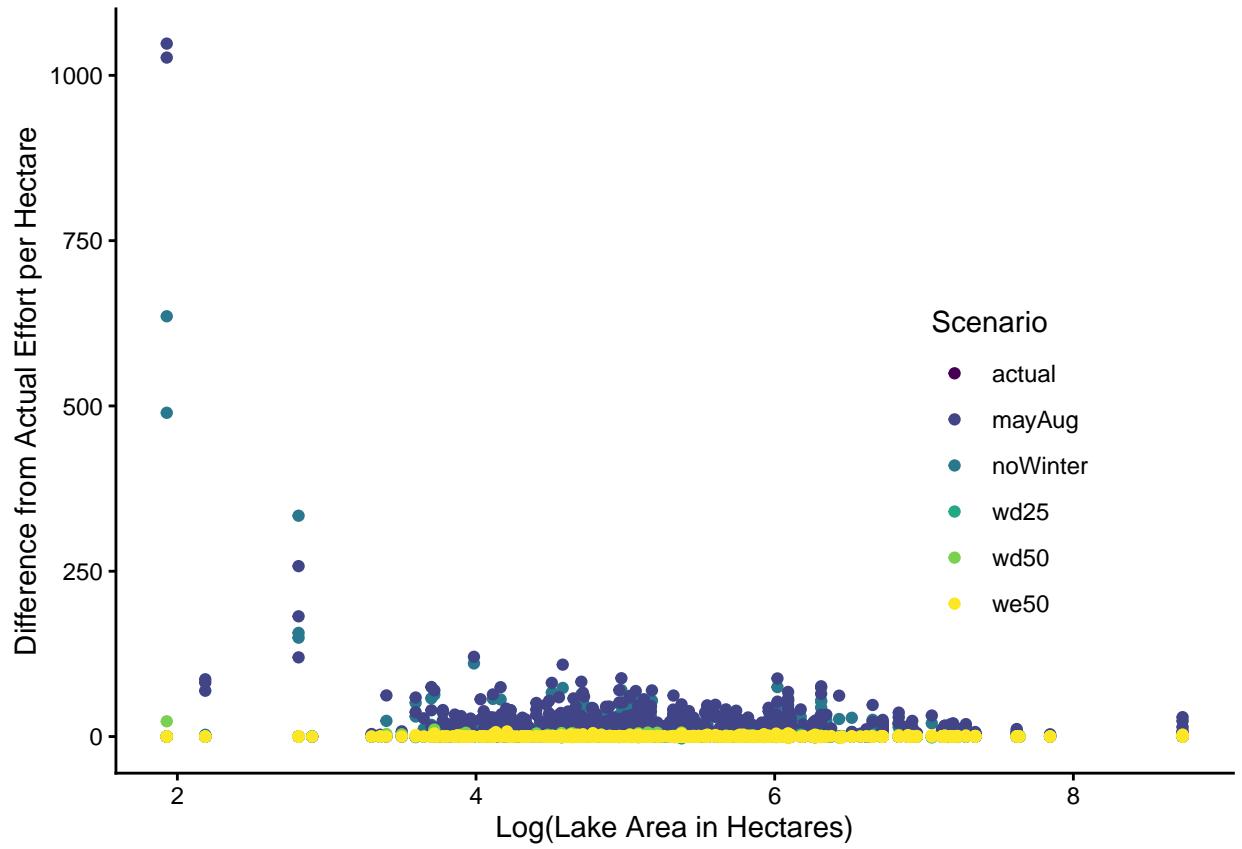


Figure 9: Comparison of the magnitude of the difference between the mean actual effort per hectare and the estimate derived from each reduced data set. Note the y-axis scale, many of the differences are small.

```

# MODELING EFFECTS OF DATA REDUCTION ON INDIVIDUAL YEARS

# likelihoods to fit
effLL.we50 = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(tdat[tdat$treat == "we50", ]), meanlog = alpha,
    sdlog = beta)
  ll = dlnorm(tdat$total.eff[tdat$treat == "we50"], meanlog = mean(log(efs)),
    sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

effLL.wd50 = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(tdat[tdat$treat == "wd50", ]), meanlog = alpha,
    sdlog = beta)
  ll = dlnorm(tdat$total.eff[tdat$treat == "wd50"], meanlog = mean(log(efs)),
    sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

effLL.wd25 = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(tdat[tdat$treat == "wd25", ]), meanlog = alpha,
    sdlog = beta)
  ll = dlnorm(tdat$total.eff[tdat$treat == "wd25"], meanlog = mean(log(efs)),
    sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

effLL.ma = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(tdat[tdat$treat == "mayAug", ]), meanlog = alpha,
    sdlog = beta)
  ll = dlnorm(tdat$total.eff[tdat$treat == "mayAug"], meanlog = mean(log(efs)),
    sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

effLL.nw = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(tdat[tdat$treat == "noWinter", ]), meanlog = alpha,
    sdlog = beta)
  ll = dlnorm(tdat$total.eff[tdat$treat == "noWinter"], meanlog = mean(log(efs)),
    sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

```

```

effLL.a = function(param) {
  alpha = param[1]
  beta = param[2]

  efs = rlnorm(nrow(tdat[tdat$treat == "actual", ]), meanlog = alpha,
    sdlog = beta)
  ll = dlnorm(tdat$total.eff[tdat$treat == "actual"], meanlog = mean(log(efs)),
    sdlog = sd(log(efs)), log = T)
  return(sum(ll))
}

# removing 0s
ttrExp = ttrExp[ttrExp$total.eff != 0, ]
loopY = sort(unique(ttrExp$year))

bpval.comp.y = data.frame(year = NA, scenario = NA, sd.pval = NA)

bpval.self.y = data.frame(year = NA, scenario = NA, sd.pval = NA)
grMetrics.y = data.frame(year = NA, scenario = NA, gr.prst = NA,
  gr.par1 = NA, gr.par2 = NA)

for (y in 1:length(loopY)) {
  # first get to year-specific data
  tdat = ttrExp[ttrExp$year == loopY[y], ]
  # Bayesian Model Fitting ACTUAL

  prior = createTruncatedNormalPrior(mean = c(mean(log(ttrExp$total.eff[ttrExp$treat ==
    "actual"])), sd(log(ttrExp$total.eff[ttrExp$treat ==
    "actual"]))), sd = c(1, 1), lower = c(-15, 0), upper = c(15,
    5))

  setup.a = createBayesianSetup(effLL.a, prior = prior)

  settings = list(iterations = 10000, nrChains = 3, message = F,
    burnin = 5000)
  set.seed(10)
  t.a = runMCMC(bayesianSetup = setup.a, sampler = "DEzs",
    settings = settings)
  # NW

  prior = createTruncatedNormalPrior(mean = c(mean(log(ttrExp$total.eff[ttrExp$treat ==
    "noWinter"])), sd(log(ttrExp$total.eff[ttrExp$treat ==
    "noWinter"]))), sd = c(1, 1), lower = c(-15, 0), upper = c(15,
    5))

  setup.nw = createBayesianSetup(effLL.nw, prior = prior)

  settings = list(iterations = 10000, nrChains = 3, message = F,
    burnin = 5000)
  set.seed(10)
  t.nw = runMCMC(bayesianSetup = setup.nw, sampler = "DEzs",
    settings = settings)
  # MA
}

```

```

prior = createTruncatedNormalPrior(mean = c(mean(log(ttrExp$total.eff[ttrExp$treat == "mayAug"])), sd(log(ttrExp$total.eff[ttrExp$treat == "mayAug"]))), sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.ma = createBayesianSetup(effLL.ma, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
    burnin = 5000)
set.seed(10)
t.ma = runMCMC(bayesianSetup = setup.ma, sampler = "DEzs",
    settings = settings)
# WD.25

prior = createTruncatedNormalPrior(mean = c(mean(log(ttrExp$total.eff[ttrExp$treat == "wd25"])), sd(log(ttrExp$total.eff[ttrExp$treat == "wd25"]))),
    sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.wd25 = createBayesianSetup(effLL.wd25, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
    burnin = 5000)
set.seed(10)
t.25wd = runMCMC(bayesianSetup = setup.wd25, sampler = "DEzs",
    settings = settings)
# WD.50

prior = createTruncatedNormalPrior(mean = c(mean(log(ttrExp$total.eff[ttrExp$treat == "wd50"])), sd(log(ttrExp$total.eff[ttrExp$treat == "wd50"]))),
    sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.wd50 = createBayesianSetup(effLL.wd50, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
    burnin = 5000)
set.seed(10)
t.50wd = runMCMC(bayesianSetup = setup.wd50, sampler = "DEzs",
    settings = settings)

# WE.50

prior = createTruncatedNormalPrior(mean = c(mean(log(ttrExp$total.eff[ttrExp$treat == "we50"])), sd(log(ttrExp$total.eff[ttrExp$treat == "we50"]))),
    sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.we50 = createBayesianSetup(effLL.we50, prior = prior)

settings = list(iterations = 10000, nrChains = 3, message = F,
    burnin = 5000)
set.seed(10)
t.50we = runMCMC(bayesianSetup = setup.we50, sampler = "DEzs",
    settings = settings)

```

```

## GR Diagnostics to record for each year

gr.a = gelmanDiagnostics(t.a)
gr.ma = gelmanDiagnostics(t.ma)
gr.nw = gelmanDiagnostics(t.nw)
gr.wd25 = gelmanDiagnostics(t.25wd)
gr.wd50 = gelmanDiagnostics(t.50wd)
gr.we50 = gelmanDiagnostics(t.50we)

## BAYESIAN P-VALUE CALCS to record for each year

pars.a = getSample(t.a)
pars.nw = getSample(t.nw)
pars.ma = getSample(t.ma)
pars.wd25 = getSample(t.25wd)
pars.wd50 = getSample(t.50wd)
pars.we50 = getSample(t.50we)

##### Pb ACTUAL #####
pval.actual = data.frame(alpha = rep(NA, nrow(pars.a)), beta = NA,
    sd = NA)
set.seed(10)
for (i in 1:nrow(pars.a)) {
    tempdat = rlnorm(n = length(tdat$total.eff[tdat$treat ==
        "actual"]), meanlog = pars.a[i, 1], sdlog = pars.a[i,
        2])
    pval.actual$alpha[i] = pars.a[i, 1]
    pval.actual$beta[i] = pars.a[i, 2]
    pval.actual$sd[i] = sd(log(tempdat))
}

# now calculate the number of times the sd exceeds that
# of the real data

pval.actual$sdExceed = 0

actual.sd = sd(log(tdat$total.eff[tdat$treat == "actual"]))

pval.actual$sdExceed[pval.actual$sd > actual.sd] = 1

##### Pb NO WINTER #####
pval.noWinter = data.frame(alpha = rep(NA, nrow(pars.nw)),
    beta = NA, sd = NA)
set.seed(10)
for (i in 1:nrow(pars.nw)) {
    tempdat = rlnorm(n = length(tdat$total.eff[tdat$treat ==
        "noWinter"]), meanlog = pars.nw[i, 1], sdlog = pars.nw[i,
        2])
    pval.noWinter$alpha[i] = pars.nw[i, 1]
    pval.noWinter$beta[i] = pars.nw[i, 2]
    pval.noWinter$sd[i] = sd(log(tempdat))
}

```

```

# now calculate the number of times the sd exceeds that
# of the real data

pval.noWinter$sdExceed = 0

noWinter.sd = sd(log(tdat$total.eff[tdat$treat == "noWinter"]))

pval.noWinter$sdExceed[pval.noWinter$sd > noWinter.sd] = 1

##### Pb MAYAUGUST #####
pval.mayAug = data.frame(alpha = rep(NA, nrow(pars.ma)),
    beta = NA, sd = NA)
set.seed(10)
for (i in 1:nrow(pars.ma)) {
    tempdat = rlnorm(n = length(tdat$total.eff[tdat$treat ==
        "mayAug"])), meanlog = pars.ma[i, 1], sdlog = pars.ma[i,
        2])
    pval.mayAug$alpha[i] = pars.ma[i, 1]
    pval.mayAug$beta[i] = pars.ma[i, 2]
    pval.mayAug$sd[i] = sd(log(tempdat))
}

# now calculate the number of times the sd exceeds that
# of the real data

pval.mayAug$sdExceed = 0

mayAug.sd = sd(log(tdat$total.eff[tdat$treat == "mayAug"]))

pval.mayAug$sdExceed[pval.mayAug$sd > mayAug.sd] = 1

##### Pb WD25 #####
pval.wd25 = data.frame(alpha = rep(NA, nrow(pars.wd25)),
    beta = NA, sd = NA)
set.seed(10)
for (i in 1:nrow(pars.wd25)) {
    tempdat = rlnorm(n = length(tdat$total.eff[tdat$treat ==
        "wd25"])), meanlog = pars.wd25[i, 1], sdlog = pars.wd25[i,
        2])
    pval.wd25$alpha[i] = pars.wd25[i, 1]
    pval.wd25$beta[i] = pars.wd25[i, 2]
    pval.wd25$sd[i] = sd(log(tempdat))
}

# now calculate the number of times the sd exceeds that
# of the real data

pval.wd25$sdExceed = 0

wd25.sd = sd(log(tdat$total.eff[tdat$treat == "wd25"]))

pval.wd25$sdExceed[pval.wd25$sd > wd25.sd] = 1

```

```

##### Pb WD50 #####
pval.wd50 = data.frame(alpha = rep(NA, nrow(pars.wd50)),
  beta = NA, sd = NA)
set.seed(10)
for (i in 1:nrow(pars.wd50)) {
  tempdat = rlnorm(n = length(tdat$total.eff[tdat$treat ==
    "wd50"]), meanlog = pars.wd50[i, 1], sdlog = pars.wd50[i,
    2])
  pval.wd50$alpha[i] = pars.wd50[i, 1]
  pval.wd50$beta[i] = pars.wd50[i, 2]
  pval.wd50$sd[i] = sd(log(tempdat))
}

# now calculate the number of times the sd exceeds that
# of the real data

pval.wd50$sdExceed = 0

wd50.sd = sd(log(tdat$total.eff[tdat$treat == "wd50"]))

pval.wd50$sdExceed[pval.wd50$sd > wd50.sd] = 1

##### Pb WE50 #####
pval.we50 = data.frame(alpha = rep(NA, nrow(pars.we50)),
  beta = NA, sd = NA)
set.seed(10)
for (i in 1:nrow(pars.we50)) {
  tempdat = rlnorm(n = length(tdat$total.eff[tdat$treat ==
    "we50"]), meanlog = pars.we50[i, 1], sdlog = pars.we50[i,
    2])
  pval.we50$alpha[i] = pars.we50[i, 1]
  pval.we50$beta[i] = pars.we50[i, 2]
  pval.we50$sd[i] = sd(log(tempdat))
}

# now calculate the number of times the sd exceeds that
# of the real data

pval.we50$sdExceed = 0

we50.sd = sd(log(tdat$total.eff[tdat$treat == "we50"]))

pval.we50$sdExceed[pval.we50$sd > we50.sd] = 1

# df to hold pvals for model comparison to self, a way
# of knowing the model fit the data well
t.pself = data.frame(year = rep(loopY[y], 6), scenario = c("Actual",
  "No Winter", "May-August", "25% weeday removal", "50% weekday removal",
  "50% weekend removal"), sd.pval = NA)
# adding self comparison pvals
t.pself$sd.pval = c(sum(pval.actual$sdExceed)/nrow(pval.actual),
  sum(pval.noWinter$sdExceed)/nrow(pval.noWinter), sum(pval.mayAug$sdExceed)/nrow(pval.mayAug),
  sum(pval.wd25$sdExceed)/nrow(pval.wd25), sum(pval.wd50$sdExceed)/nrow(pval.wd50),

```

```

    sum(pval.we50$sdExceed)/nrow(pval.we50))
bpval.self.y = rbind(bpval.self.y, t.pself)
## p-value tests comparing the scenarios to actual to
## show that some scenarios approximate the actual
## quite well.

t.pcomp = data.frame(year = rep(loopY[y], 6), scenario = c("Actual",
    "No Winter", "May-August", "25% weeday removal", "50% weekday removal",
    "50% weekend removal"), sd.pval = NA)

pval.actual$sdComp = 0
pval.actual$sdComp[pval.actual$sd > actual.sd] = 1

pval.noWinter$sdComp = 0
pval.noWinter$sdComp[pval.noWinter$sd > actual.sd] = 1

pval.mayAug$sdComp = 0
pval.mayAug$sdComp[pval.mayAug$sd > actual.sd] = 1

pval.wd25$sdComp = 0
pval.wd25$sdComp[pval.wd25$sd > actual.sd] = 1

pval.wd50$sdComp = 0
pval.wd50$sdComp[pval.wd50$sd > actual.sd] = 1

pval.we50$sdComp = 0
pval.we50$sdComp[pval.we50$sd > actual.sd] = 1

t.pcomp$sd.pval = c(sum(pval.actual$sdComp)/nrow(pval.actual),
    sum(pval.noWinter$sdComp)/nrow(pval.noWinter), sum(pval.mayAug$sdComp)/nrow(pval.mayAug),
    sum(pval.wd25$sdComp)/nrow(pval.wd25), sum(pval.wd50$sdComp)/nrow(pval.wd50),
    sum(pval.we50$sdComp)/nrow(pval.we50))
bpval.comp.y = rbind(bpval.comp.y, t.pcomp)

# adding GR results to the output dataframe
t.gr = data.frame(year = rep(loopY[y], 6), scenario = c("Actual",
    "No Winter", "May-August", "25% weeday removal", "50% weekday removal",
    "50% weekend removal"), gr.prse = c(gr.a[[2]], gr.nw[[2]],
    gr.ma[[2]], gr.wd25[[2]], gr.wd50[[2]], gr.we50[[2]]),
    gr.par1 = c(gr.a[[1]][1, 1], gr.nw[[1]][1, 1], gr.ma[[1]][1,
        1], gr.wd25[[1]][1, 1], gr.wd50[[1]][1, 1], gr.we50[[1]][1,
        1]), gr.par2 = c(gr.a[[1]][2, 1], gr.nw[[1]][2, 1],
    gr.ma[[1]][2, 1], gr.wd25[[1]][2, 1], gr.wd50[[1]][2,
        1], gr.we50[[1]][2, 1]))
grMetrics.y = rbind(grMetrics.y, t.gr)
}

# saving big loop's output since it takes a while to run
yearLoopOutput = list(bpval.self.y, bpval.comp.y, grMetrics.y)
# saveRDS(yearLoopOutput, file =
# 'yearLoopOutput_effort_9.23.25.RData')

```

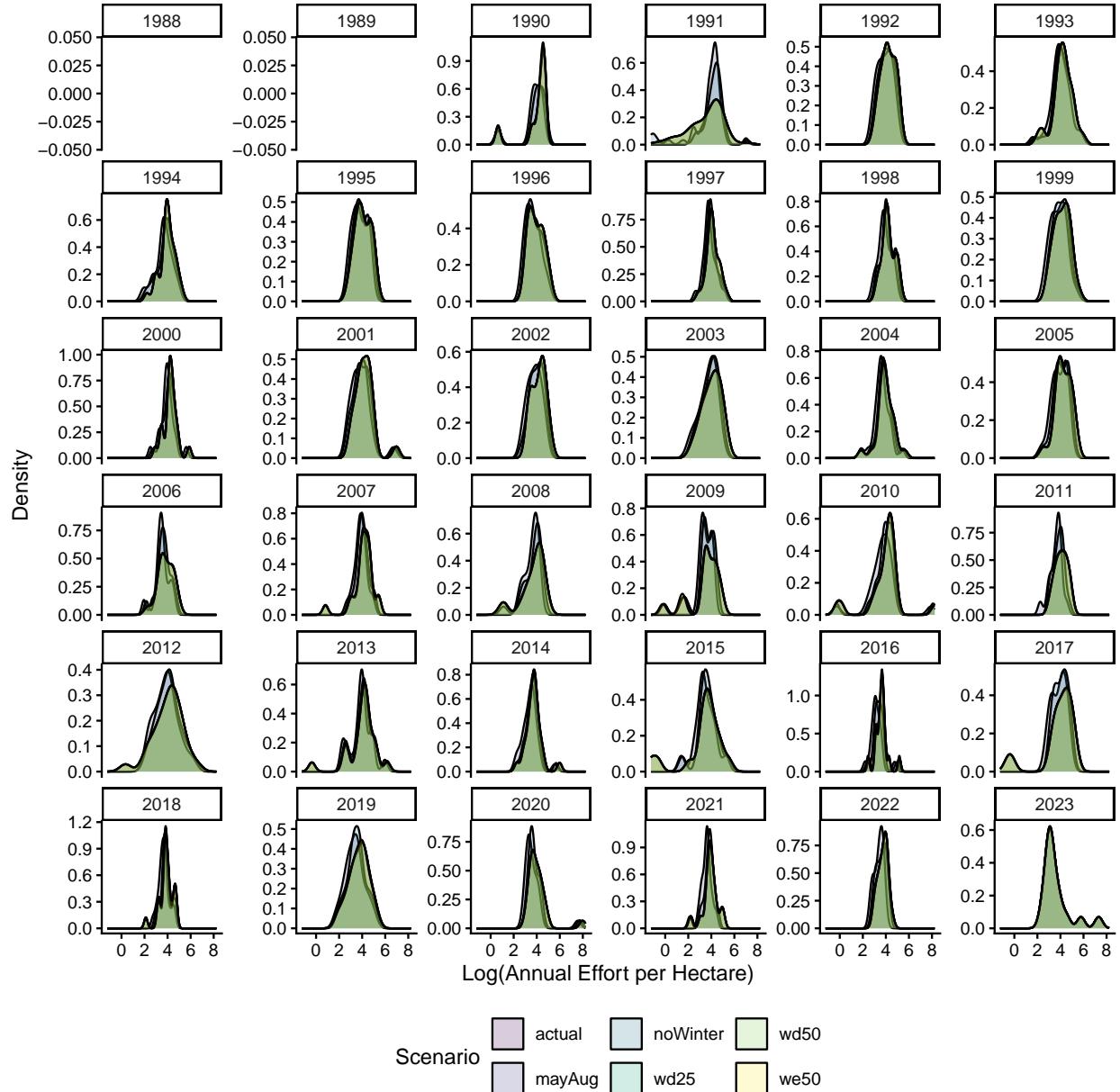


Figure 10: Distribution of effort estimates across years in the creel data set. Different data reduction scenarios are noted with varying colors.

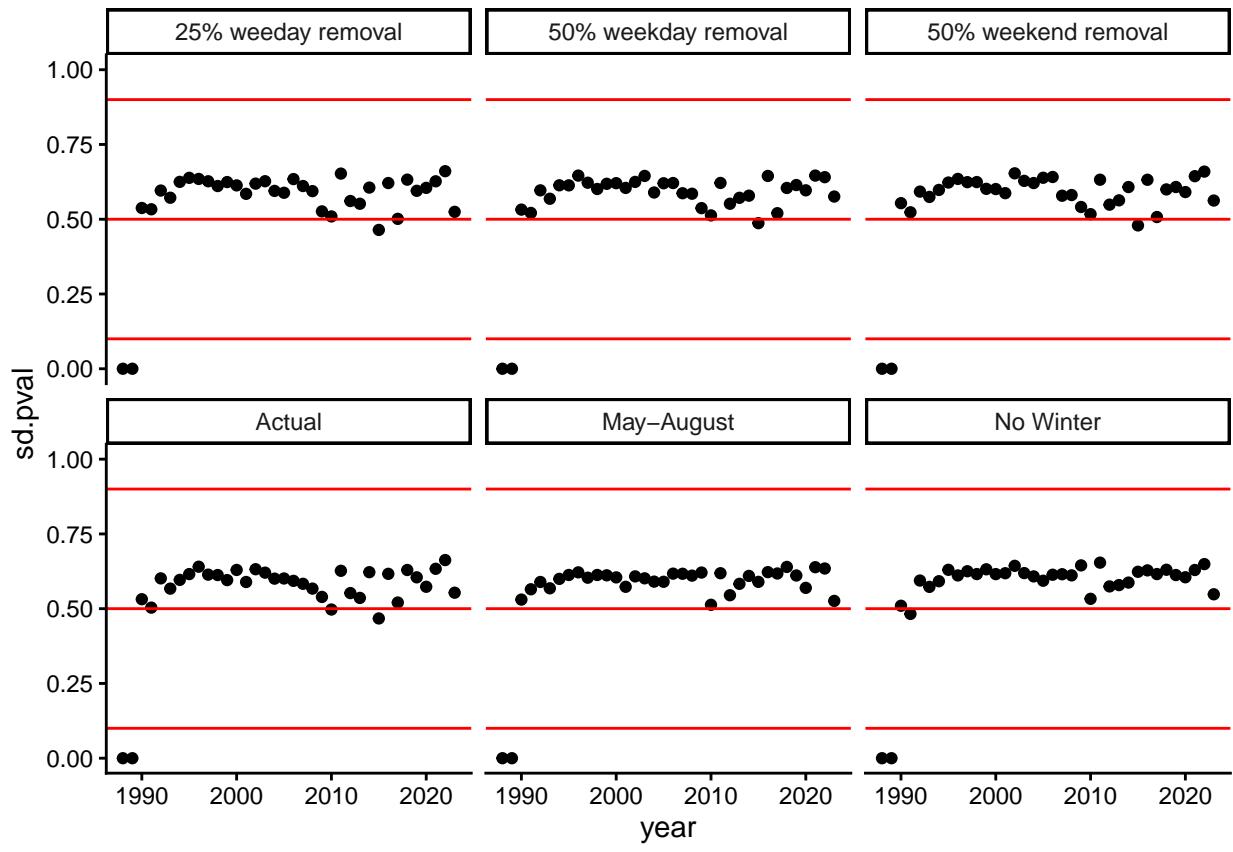


Figure 11: Comparison of coefficient of variations for bayesian p-values comparing the actual data to the simulated data for the creel year and data reduction scenario.

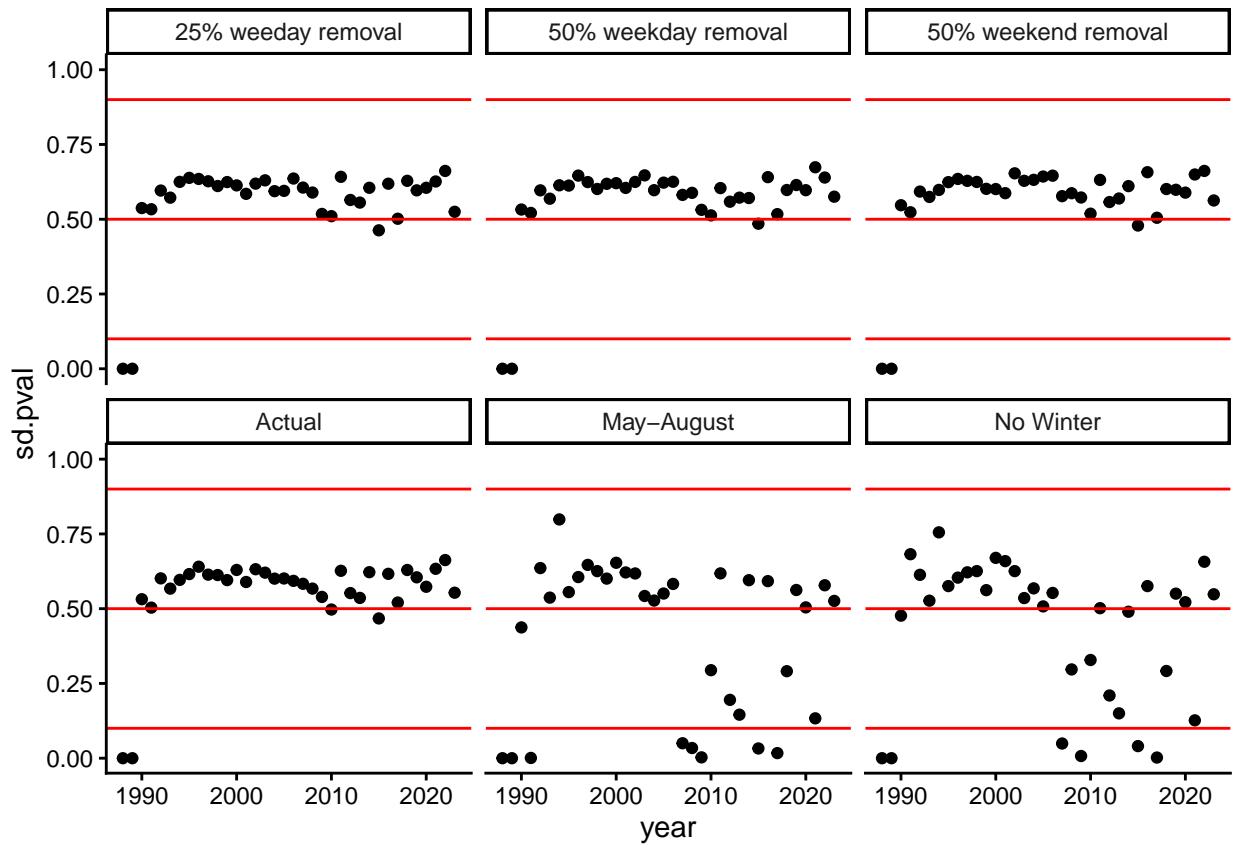


Figure 12: Comparison of the coefficient of variations for bayesian p-values comparing the simulated data from each creel year and data reduction scenario to the actual data for that same creel year.

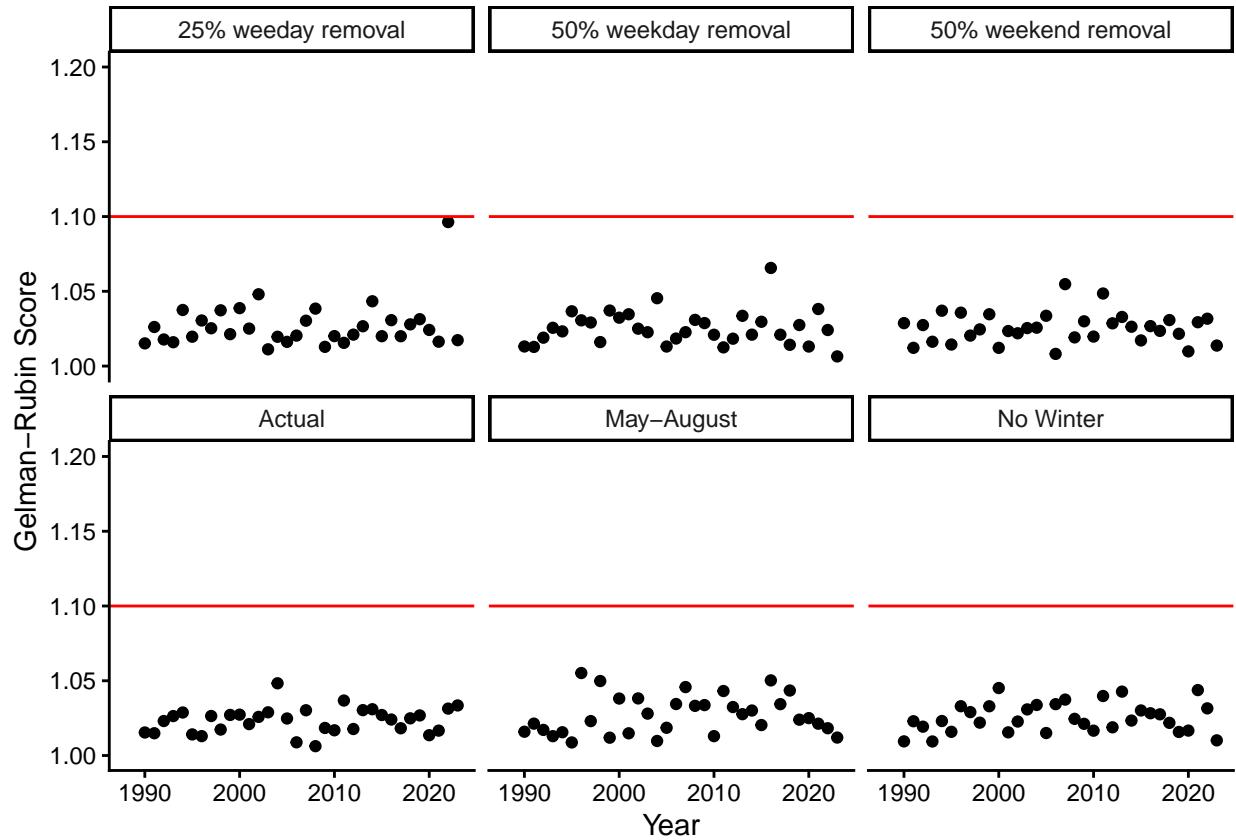


Figure 13: Gelman-Rubin diagnostic results for the year-by-year model fitting. All years with available data had models successfully converge with a Gelman-Rubin score below the 1.1 threshold required for convergence (red horizontal line).

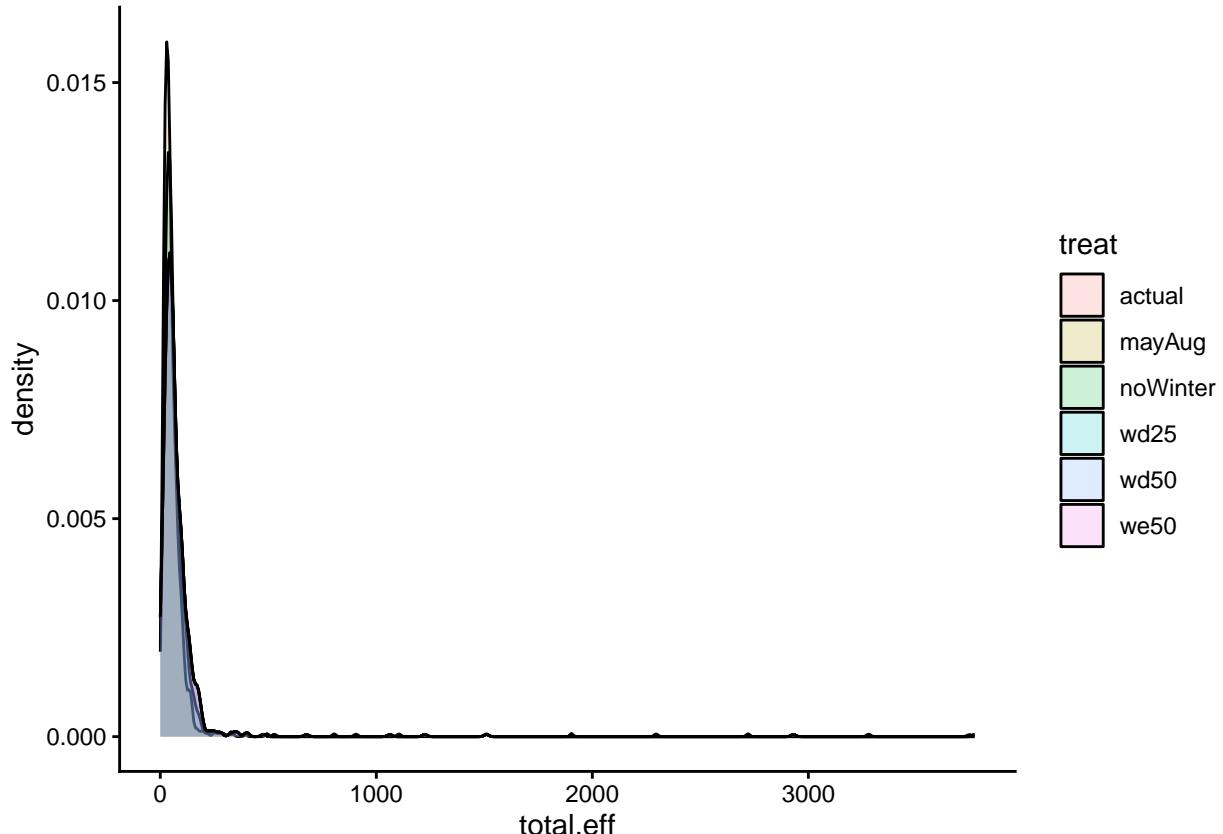
### 1.1.5 Alternative modeling distributions

Instead of the lognormal distribution there is one other probability distributions that could be used to model this data based on the characteristics of the data and the data characteristics assumed by each of the distributions.

Table of data distributions and their definitions:

Distribution	Definition
Lognormal	Continuously distributed quantities with nonnegative values. Random variables with the property that their logs are normally distributed.
Gamma	Any continuous quantity that is nonnegative. Continuous version of a Poisson distribution.

The following analyses will compare the ability of models using the lognormal and gamma distributions to fit the full data and each of the reduced data set. A cursory look at the ability of gamma distributed models showed no real difference from beta and lognormal and the definition of the beta better represents the data we're dealing with than the gamma definition so further analyses with this family of models was not pursued (Figure 14). The purpose of the comparison between lognormal and beta family models is not to see which model provided the most convenient answer but to compare their ability to fit the data. All comparisons presented in this analysis are comparing a model's ability to fit the data **within** each dataset and **not** across data sets which is what is necessary to gain inference on the effect of any data reductions.



Gelman-Rubin convergence diagnostics indicate all models have converged. Seems like the gamma models could be a potential alternative, the bayesian p values should help confirm this.

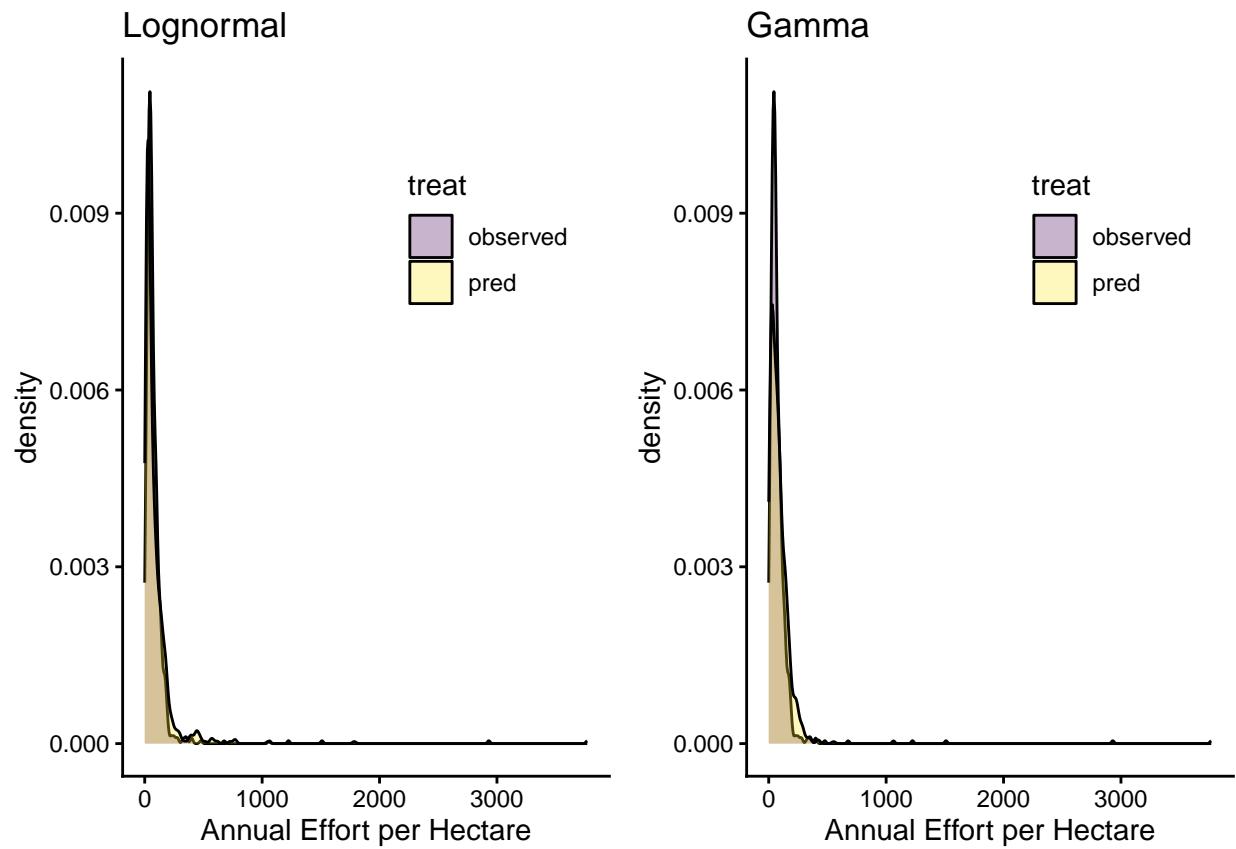


Figure 14: Comparison of model fits for lognormal and gamma, family of models. This rough look at the models suggests there are no obvious differences between the choice of modeling distribution. This will be evaluated statistically using bayesian p values later on.

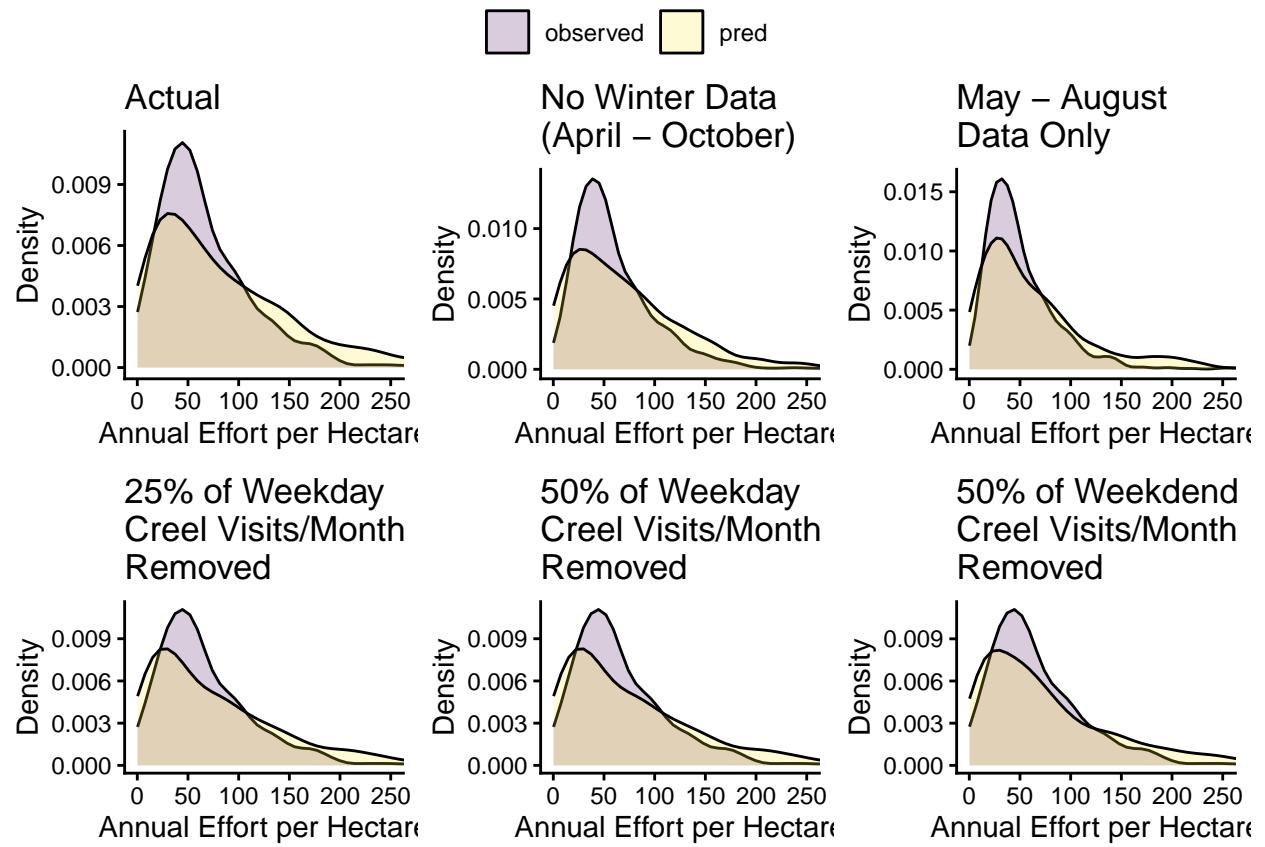


Figure 15: Visualization of the model fit to the data for each scenario using a gamma distribution instead of a lognormal distribution.

Table 5: Bayesian p-values for a gamma distributed model. Each p-value describes whether or not there is a significant difference between data generated by the fitted model and the actual data for that scenario. Standard deviation (SD) was the metric assessed here.

Scenario	SD p-value
Actual	0
No Winter	0
May-August	0
25% weeday removal	0
50% weekday removal	0
50% weekend removal	0

#### ### MODEL CHECKNG #####

```
gelmanDiagnostics(eff.actual.gamma) # converged
gelmanDiagnostics(eff.nw.gamma) # converged
gelmanDiagnostics(eff.ma.gamma) # converged
gelmanDiagnostics(eff.25wd.gamma) # converged
gelmanDiagnostics(eff.50wd.gamma) # converged
gelmanDiagnostics(eff.50we.gamma) # converged
```

The Bayesian p-values in the table presented here suggests the gamma models do a poor job of reproducing the data they were fit to (Table 5). This inference comes from the lack of p-values between 0.1 - 0.9 for the SD metric. Because this model is unable to reproduce the data it was fit to, we know that this model is not a useful one for fitting these data. This means that the gamma distribution not likely to be appropriate for making comparisons between the actual data and the reduced data to understand whether or not creel effort reductions result in significantly different annual angler effort per hectare estimates or not.

### 1.1.6 P-value sensitivity analysis

**1.1.6.1 Interpreting each statistic** Some more detail on what I'm looking for in each statistic and what the values mean that are used to calculate each Bayesian p-value. It's important to remember that the way the sampling algorithm works in this Bayesian framework the frequency of values in the MCMC output is relative to their support in the data. In other words, parameter values that produce better models fits show up in the chain more often. This is critical to understanding the Bayesian p-values here because each test statistic is calculated for simulated data based on the MCMC output so each test statistic value should appear with a frequency equal to the support provided by the data too. For example, the medians calculated from data sets produced by each parameter set in the MCMC output should produce medians that have frequencies equal to how well those parameters fit the data. Thus medians well below the median of the observed data should be pretty rare (and likewise for medians well above) because the parameter set from the MCMC chain that produces data with said median should be relatively infrequent in the MCMC output. Medians close to the median of the observed data should be more common, if the model is fitting the data well, because the parameters generating those means are more common in the MCMC output due to the better fit to the data they provide. This results in medians that are often close to the median of the observed data and by random chance should be just as likely to be above as below the median of the observed data. This is why Bayesian p-values close to 0.5 signify a good model fit, because about 50% of the time the test statistic is more extreme than the value for the observed data. If this values is extremely high or low then is signals that our model is not adequately representing the observed data.

**1.1.6.1.1 Standard Deviation** Another straightforward calculation. SD is calculated for a data set drawn from a random lognormal distribution parameterized using the values in the MCMC chain. This is

repeated for each set of parameter values in the MCMC chain to create as many simulated data sets as there are iterations in the MCMC chain. The SD of each of these simulated data sets is compared to the SD for the observed data for that scenario (Actual, No Winter, May - August, etc.). Whether the SD of the simulated data is greater than or less than the SD of the observed data is recorded. **If the model is able to reproduce the observed data well then the SD of the simulated data should be greater than the SD of the observed data about 50% of the time and less than the SD of the observed data about 50% of the time too.**

**1.1.6.1.2 Median** Simple descriptive statistic here. The median is calculated for each simulated data set produced by drawing values from a random lognormal distribution parameterized using the values in the MCMC output. The number of times the median of the simulated data set exceeds the median of the observed data is calculated and that proportion of times the test statistic exceeds the value for the observed data is the Bayesian p-value. **If the model is adequately representing the data then the medians generated from the simulated data sets should exceed the median of the observed data roughly 50% of the time.**

**1.1.6.1.3 Kurtosis** This metric essentially measures the shape of a distribution, specifically the tails. Kurtosis is calculated for the simulated data for each set of parameter values in the MCMC output and compared to the kurtosis of the observed data for the given scenario (Actual, No Winter, May - August, etc.). When the kurtosis value of the simulated data is greater than the kurtosis of the observed data this signals more frequent extreme values in the data. In other words the tails of the distribution of the simulated data are thicker than the tails of the distribution of the observed data. The opposite is true when the kurtosis of the simulated data is less than the kurtosis of the observed data. **A well-fitting model will produce data with kurtosis values similar to the the observed data's kurtosis. The kurtosis of the simulated data will exceed that of the observed data roughly 50% of the time. If the kurtosis of the simulated data is more frequently less than the observed data this would signal that the model is not doing well as representing the rare values in the tails of the data distribution. The opposite is true if kurtosis values for the simulated data tend to be larger than the observed data.**

**1.1.6.1.4 Chi Square Test ( $\chi^2$ )** This test is performed outside the Bayesian p-value framework. Instead, each simulated data set arising from a set of parameters in the MCMC output is compared to the probability distribution of the observed data to ask whether the simulated data arises from the same distribution as the observed data. Again, because of the way the parameter space is sampled in MCMC algorithms the values that produce better model fits will be more common which means that in the  $\chi^2$  test the null hypothesis that the data comes from the supplied probability distribution should be accepted more often than it's rejected. **If the null hypothesis is rejected the majority of the time that would indicate that the model is not adequately representing the data if it can't produce simulated data that would appear to be from the probability distribution of the observed data used to fit the model in the first place.**

**1.1.6.1.5 Fisher 'F' Statistic** The F statistic is the ratio of variances between the simulated and observed data sets. For this test I'm looking to see if the F statistic for the simulated:observed comparison is greater than or less than 1 (the ratio of the variance for the observed:observed comparison). **The closer the F statistic is to 1 for the simulated:observed comparison the better job the model is doing at adequately representing the data. So the number of times the F statistic exceeds 1 should, if the model is fitting the data well, be about 50% of the time. If the Bayesian p-value is much higher than 0.5 that would indicate that the variance of the simulated data is often greater than the variance of the observed data. The opposite then is true when the p-value is smaller than 0.5**

Table 6: Candidate variance metrics to use for calculation of Bayesian p-values.

Statistic	Description
SD	Square root of the variance.
Median	Middle of the data when all values are ordered from smallest to largest. Similar to a mean but less sensitive to outliers.
Kurtosis	Measure of the width, or 'tailedness' of a distribution. In other words, do the tails of the distribution contain most of the data?
X2	Chi square test statistic for a 'goodness of fit' test is calculated. Here I want to know if frequency of values in the model simulated data is similar to the observed data.
F	The ratio of the variance between the model simulated data and the observed data.
Coverage	The probability that the 95 percent confidence interval for a parameter will contain the true value of the data.

**1.1.6.1.6 Coverage** The probability that the confidence interval for the parameter of interest will include the true value of the data. Here this means comparing confidence intervals on the estimated fishery metric calculated from the reduced data set back to the calculation of the fishery metric for the full creel data set. This is repeated for each parameter value in the MCMC chain resulting from the model fits. **The percentage of parameter values that produce fishery metric data that 'covers' what was observed in the full data set is what is reported. The higher the coverage, the better the reduced creel data set is able to approximate the full data set.**

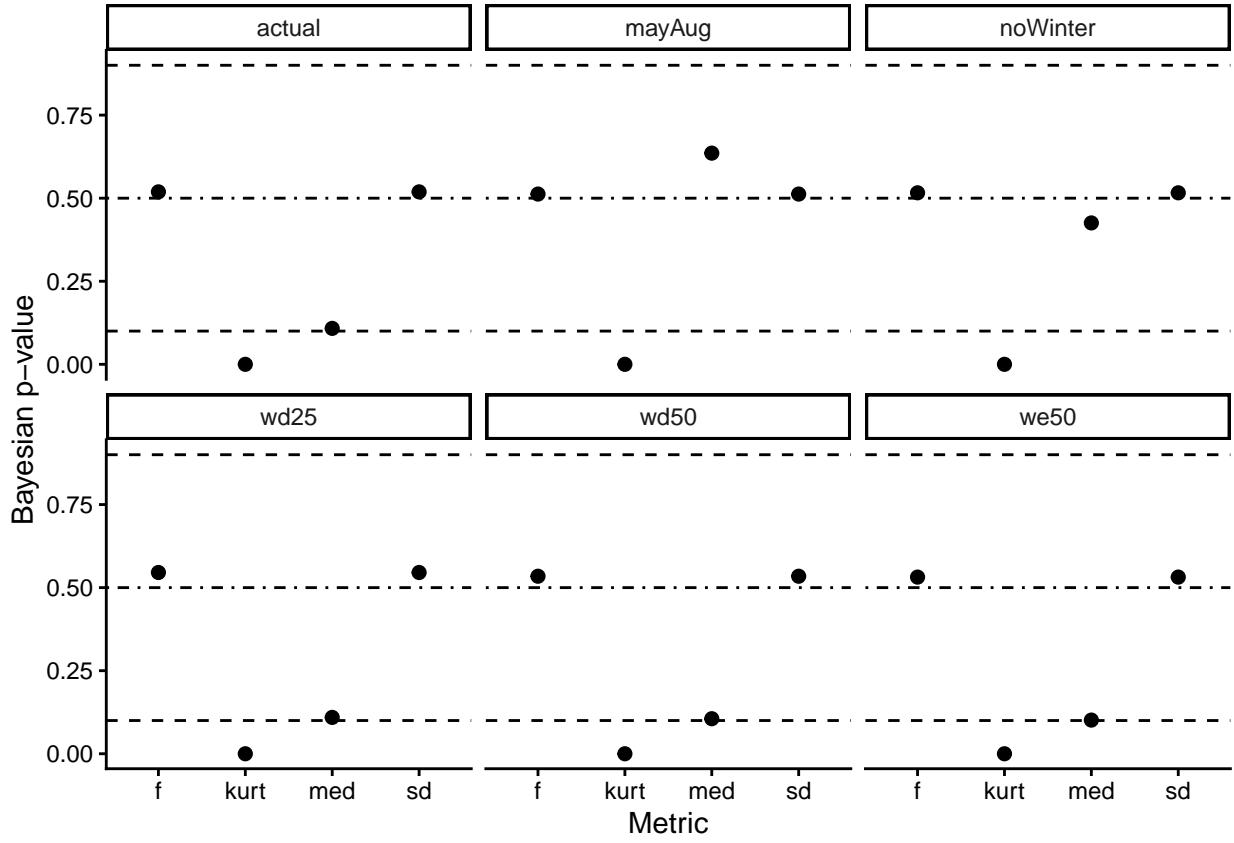


Figure 16: Comparison of multiple metrics for calculating Bayesian p-values. Each panel is a different data scenario. The comparisons in each panel are comparing the model simulated data to the observed data for that same scenario, NOT comparing model simulated data from a reduction scenario to the actual data. This comparison to self is to understand whether the model is adequately representing the data and whether the choice of statistic influences the answer to that question. Values close to 0.5 are ideal, values more extreme than 0.1 or 0.9 are considered evidence for a poor model fit (i.e. significant difference between model simulated data and the observed data).