Exploitation Rate Creel Titration

Colin Dassow

Contents

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)
```

1.2   Walleye Exploitation Rate u

Exploitation rate (u) was calculated as:

$$u = \frac{R}{M}$$

$$R = (\frac{C}{T})H$$

Where R is the estimated total number of marked fish harvested for a given creel strata and M is the total number of walleye marked that spring during fyke net surveys. The estimate of the number of marked fish harvested per strata (R) is obtained by applying the proportion of marked fish observed in the creel to the total harvest estimate for that strata (H). C is the actual number of marked fish observed in a given creel strata and T is the total number of fish examined.

When calculating u for each data set I first calculated u by each month and day type then calculated an average u for that creel survey based on `survey.seq.no` to produce one exploitation rate for that survey. This was done to match the exploitation rate database which provides exploitation rates on a survey-level and not the strata-level.

Another important note, all exploitation rates $= 0$ were dropped before modeling for two reasons, 1) because the data is lognormally distributed (Figures 1 and 2) taking the log of 0 isn't possible and 2) If a creel survey produces a 0 now, chances are that as more data is removed from that survey it won't suddenly become non-zero, so the 0s are not likely to change under future creel designs. If anything more lakes may produce 0 exploitation rates if fewer samples are taken and the scenarios here show just that.
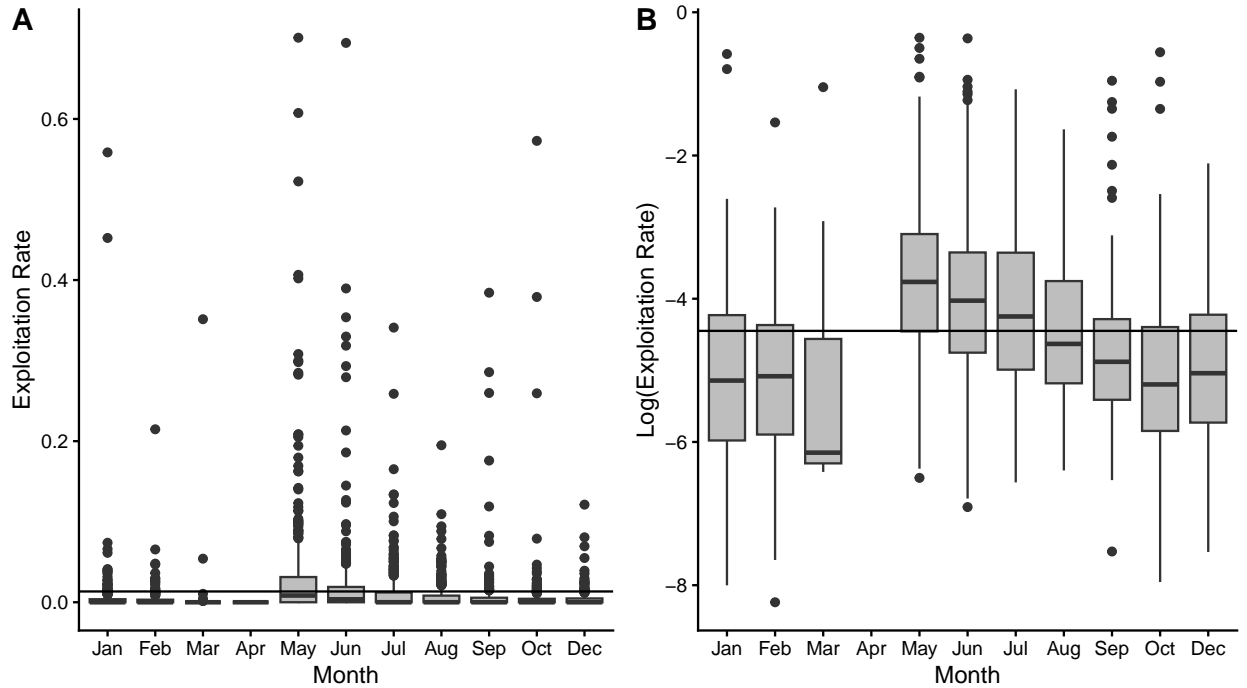

Figure 1: Distribution of exploitation rates by month (panel a) for all years of CTWI creel data. Horizontal line is overall mean exploitation rate. Black points are outliers. Panel B is the same information presented on a log-scale. In both plots exploitation rates of 0 are omitted.
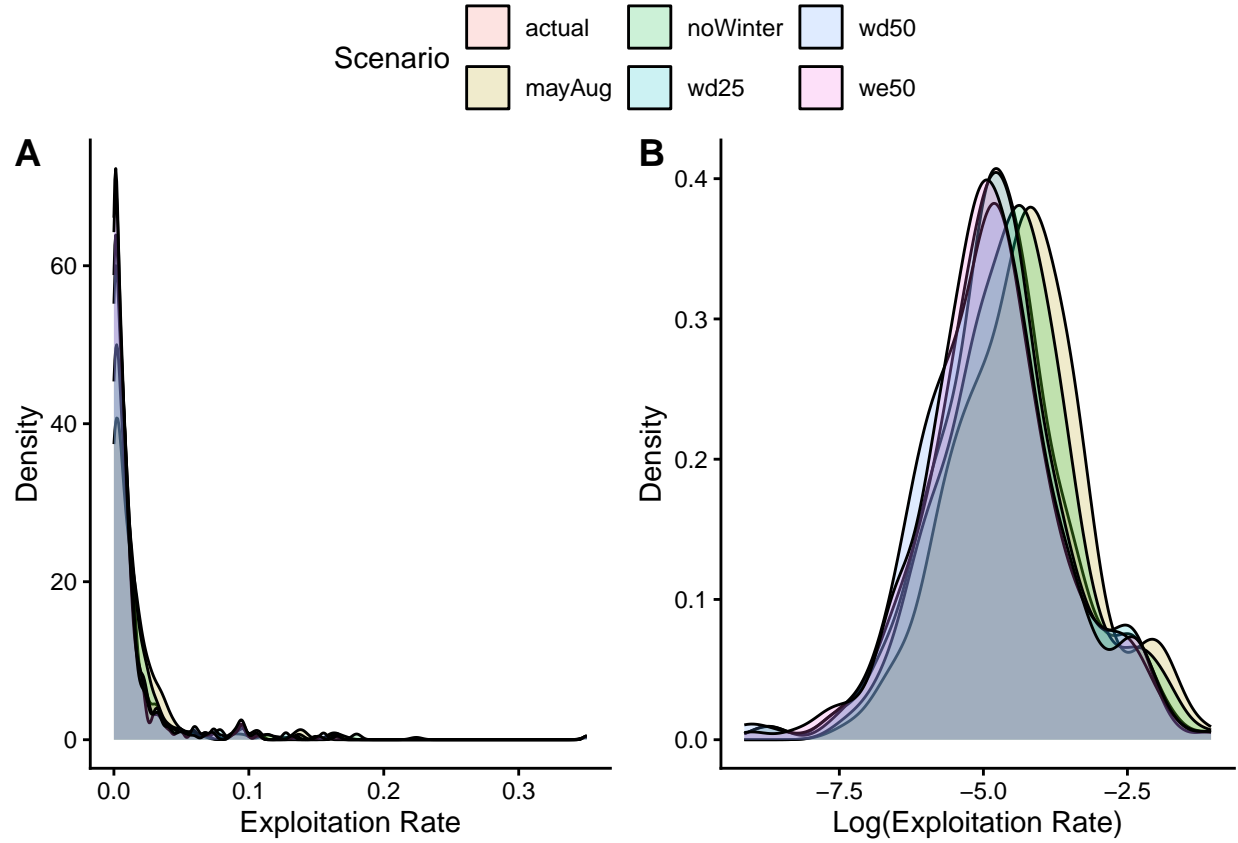
Figure 2: Distribution, on a log scale, of exploitation rates for the full data set plus 5 scenarios with reduced data. Actual = full data set, mayAug = May to August creel data only, noWinter = winter creel data removed, wd25 = 25% of weekday creel visits per month removed, wd50 = 50% of weekday creel visits per month removed, we50 = 50% of weekend creel visits per month removed.

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

Scenario	Zeros
Actual	82
No Winter	77
May-August	80
25% Weekday Reduction	83
50% Weekday Reduction	85
50% Weekend Reduction	91

1.2.1 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.

```r
# one likelihood to estimate parms for using the 6
# different treatments creating data frame to model with u
# rates that are NA removed, these are 6% of the data and
# are from years where creel happened but no fish were FN
# marked concurrently.
modDat = ttrExp[!is.na(ttrExp$exp.rate), ]

# removing 0s since they can't be logged and I'm going to
# operate under the assumption that if a survey give a 0
# exploitation rate with the full survey, then us
# collecting less data isn't going to change that number.
zeros.actual = sum(modDat$exp.rate[modDat$treat == "actual"] ==
    0)
zeros.nw = sum(modDat$exp.rate[modDat$treat == "noWinter"] ==
    0)
zeros.ma = sum(modDat$exp.rate[modDat$treat == "mayAug"] == 0)
zeros.wd25 = sum(modDat$exp.rate[modDat$treat == "wd25"] == 0)
zeros.wd50 = sum(modDat$exp.rate[modDat$treat == "wd50"] == 0)
zeros.we50 = sum(modDat$exp.rate[modDat$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 exploitation rate estimates
```

```r
# removing 0s here as described in the text.
modDat = modDat[modDat$exp.rate != 0, ]

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

```r
    us = rlnorm(nrow(modDat[modDat$treat == "actual", ]), meanlog = alpha,
        sdlog = beta)
    ll = dlnorm(modDat$exp.rate[modDat$treat == "actual"], meanlog = mean(log(us)),
        sdlog = sd(log(us)), log = T)
    return(sum(ll))
}

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

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

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


#### NW ####
uLL.nw = function(param) {
    alpha = param[1]
    beta = param[2]

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

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

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

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

#### MA ####

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

    us = rlnorm(nrow(modDat[modDat$treat == "mayAug", ]), meanlog = alpha,
        sdlog = beta)
    ll = dlnorm(modDat$exp.rate[modDat$treat == "mayAug"], meanlog = mean(log(us)),
```

```r
        sdlog = sd(log(us)), log = T)
    return(sum(ll))
}

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

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

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

#### WD25 ####

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

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

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

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

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

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

    us = rlnorm(nrow(modDat[modDat$treat == "wd50", ]), meanlog = alpha,
        sdlog = beta)
    ll = dlnorm(modDat$exp.rate[modDat$treat == "wd50"], meanlog = mean(log(us)),
        sdlog = sd(log(us)), log = T)
    return(sum(ll))
}
```

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

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

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

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

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

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

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

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

1.2.2   Model Checking

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

1.2.3   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 exploitation rates estimated from the reduced data.
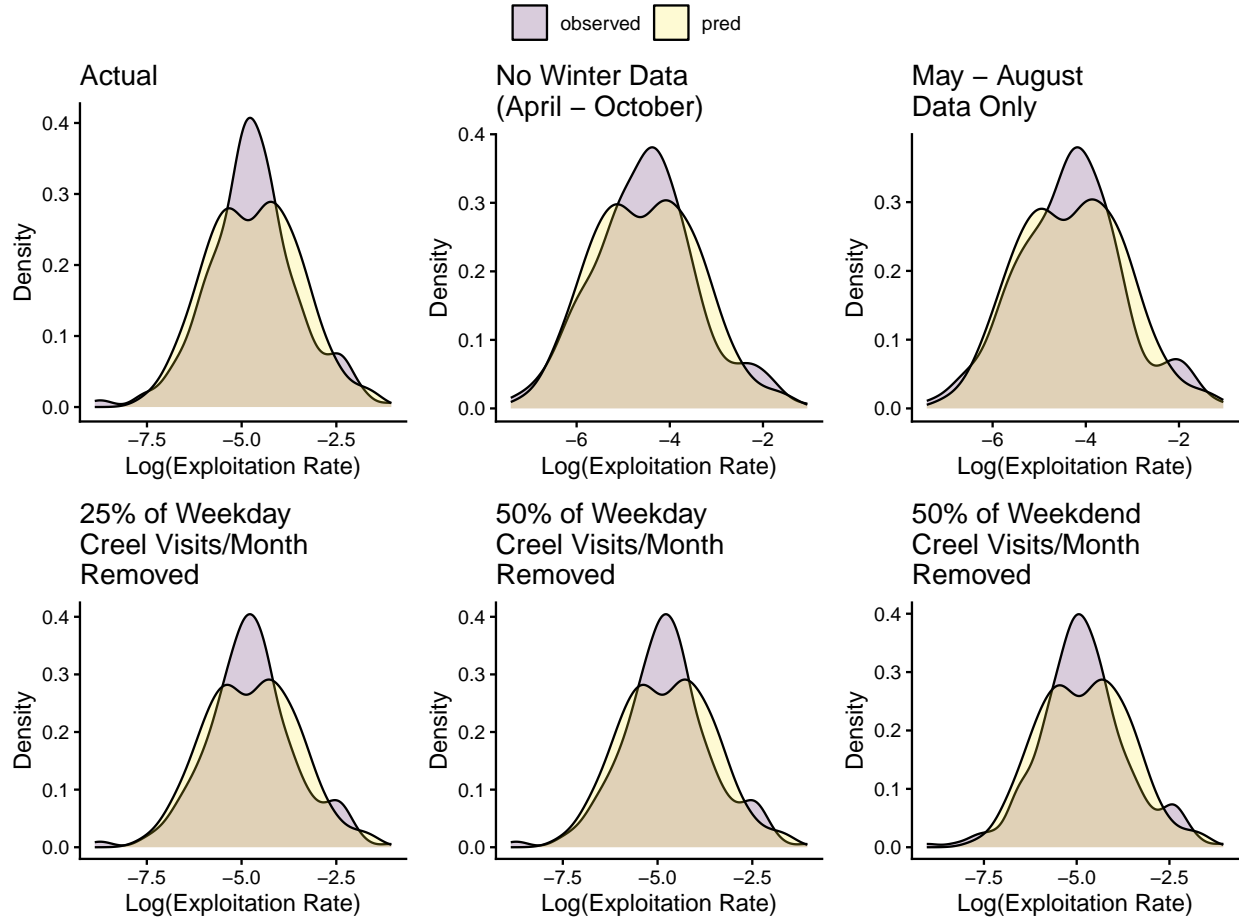
Figure 3: Distributions of the observed and model predicted data for each scenario. Model predicted data comes from lognormal distribution paramterized using the median parameter estimate of the model posterior.

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. Standard devidation (SD) was the variance metrics assessed here.

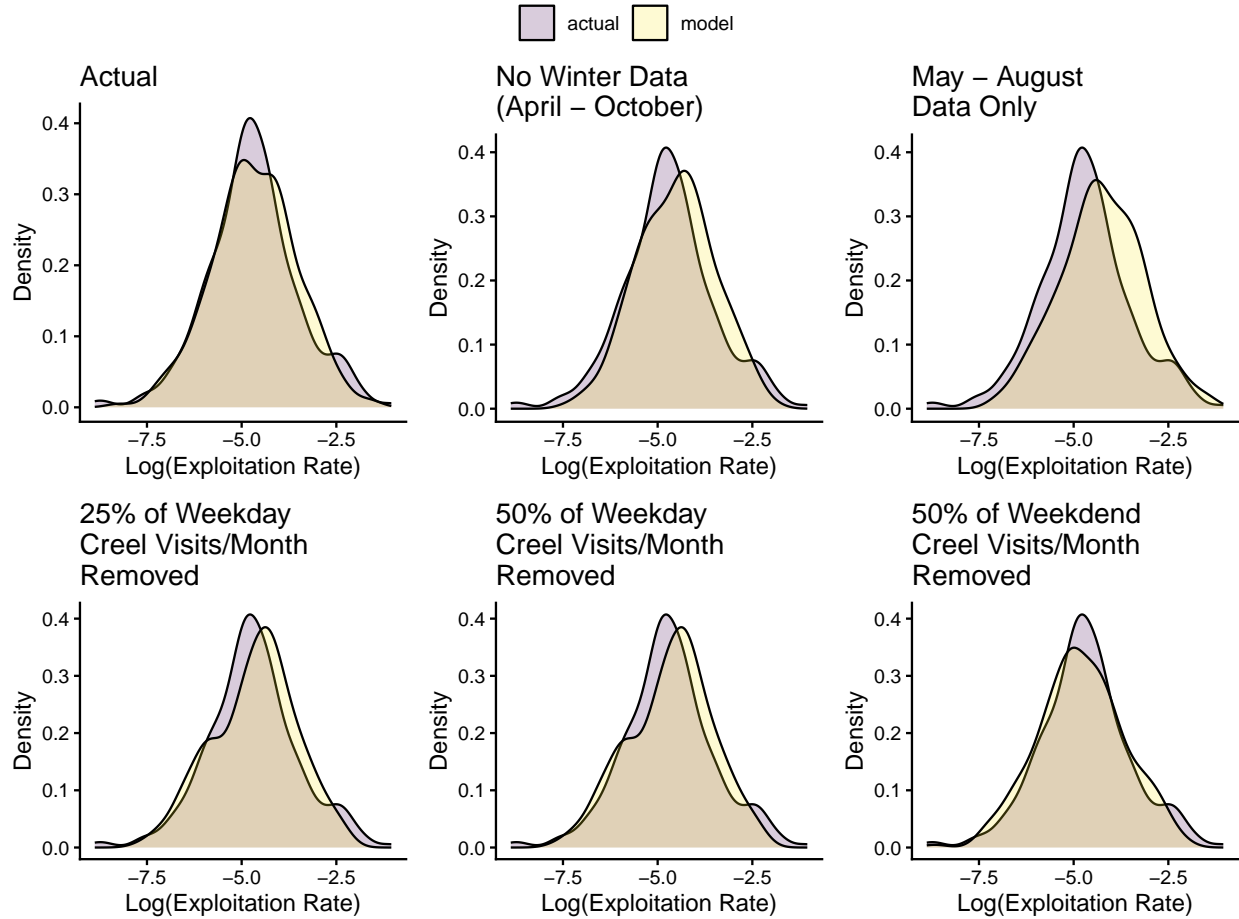| Scenario	SD p-value
Actual	0.506
No Winter	0.514
May-August	0.531
25% weeday removal	0.541
50% weekday removal	0.536
50% weekend removal	0.550

Figure 4: Comparison of the distribution of actual exploitation rates calculated from creel data and exploitation rates calculated from simulated creel data for each data reduction scenario and the resulting median parameter values for their respective model fits.

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 exploitation rates. Standard devidation (SD) was the variance metric assessed here.

Scenario	SD p-value
Actual	0.506
No Winter	0.262
May-August	0.291
25% weeday removal	0.489
50% weekday removal	0.699
50% weekend removal	0.584

Table 4: Counts of the number of times the exploitation rate estimate for the reduced data scenario differs from the actual exploitation rate by more than 1 standard deviation. NAs ocurred where there was no variance in the exploitation rate estimate and thus no standard deviation could be calculated for the actual data.

Data Reduction Scenario	u's exceeding actual	u's not exceeding actual	NAs
actual	0	298	45
mayAug	1	284	28
noWinter	1	288	31
wd25	0	298	45
wd50	0	298	45
we50	0	298	45

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.

1.2.4 Year Specific Analyses

Instead of modeling the full population of exploitation rates 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 exploitation rates estimated that year and used to set safe harvest limits. This analysis will follow the same routine as the whole-population analyses above but only consider one creel-year at a time.

```
# MODELING EFFECTS OF DATA REDUCTION ON INDIVIDUAL YEARS

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

    us = rlnorm(nrow(tdat[tdat$treat == "we50", ]), meanlog = alpha,
        sdlog = beta)
    ll = dlnorm(tdat$exp.rate[tdat$treat == "we50"], meanlog = mean(log(us)),
        sdlog = sd(log(us)), log = T)
```
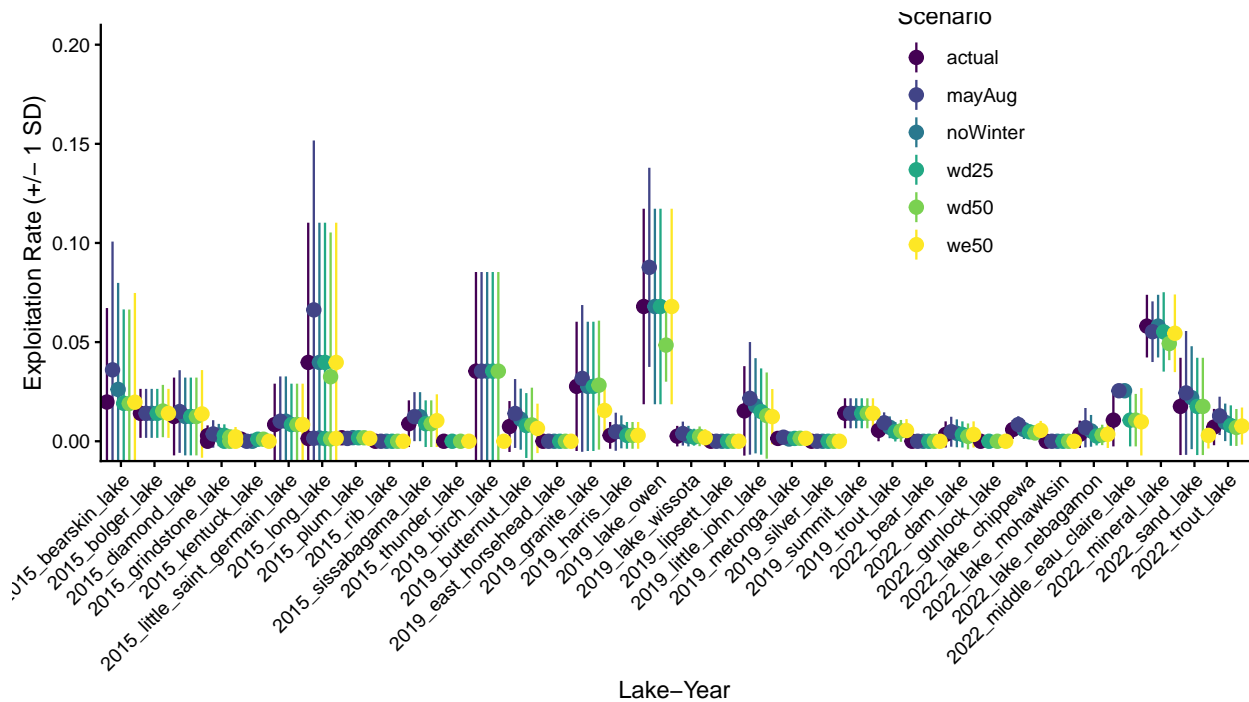
Figure 5: Comparison of the exploitation rates estimated from the reduced data for a handful of creel surveys. Most data reductions do not result in exploitaion rates that are much different from the actual data (most are within 1 SD of the actual). Colors represent data reduction scenarios, points are mean exploitaiton rate estimates across all months of that creel survey and vertical lines represent 1 standard deviation.

```
    return(sum(ll))
}
uLL.wd50 = function(param) {
    alpha = param[1]
    beta = param[2]

    us = rlnorm(nrow(tdat[tdat$treat == "wd50", ]), meanlog = alpha,
        sdlog = beta)
    ll = dlnorm(tdat$exp.rate[tdat$treat == "wd50"], meanlog = mean(log(us)),
        sdlog = sd(log(us)), log = T)
    return(sum(ll))
}
uLL.wd25 = function(param) {
    alpha = param[1]
    beta = param[2]

    us = rlnorm(nrow(tdat[tdat$treat == "wd25", ]), meanlog = alpha,
        sdlog = beta)
    ll = dlnorm(tdat$exp.rate[tdat$treat == "wd25"], meanlog = mean(log(us)),
        sdlog = sd(log(us)), log = T)
    return(sum(ll))
}
uLL.ma = function(param) {
    alpha = param[1]
    beta = param[2]
```
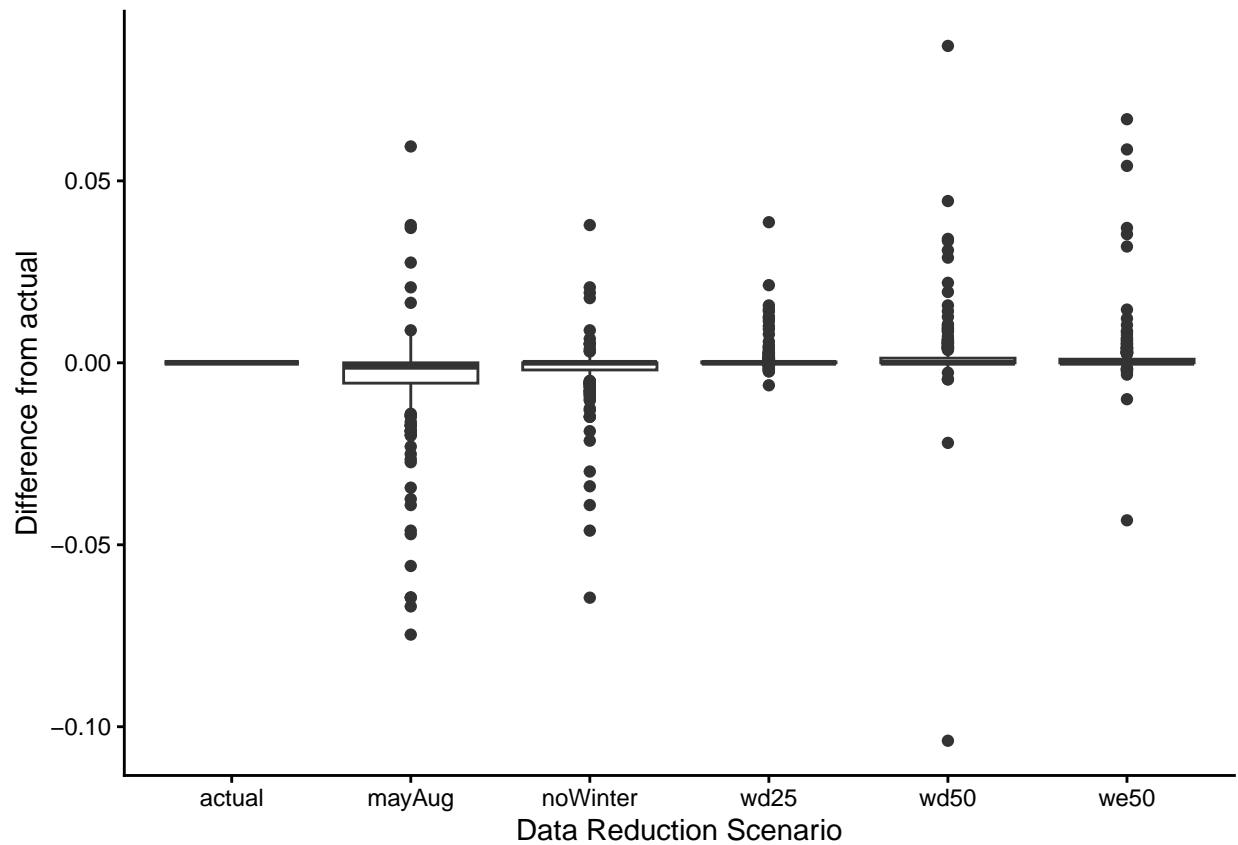
Figure 6: Boxplot of the difference between the actual exploitation rate and the data-reduced exploitation rate for each creel survey in the creel dataset. Most of the data exhibits differences very near 0, dots represent outliers (x<|> x's percentile-1.5*interquartile range)
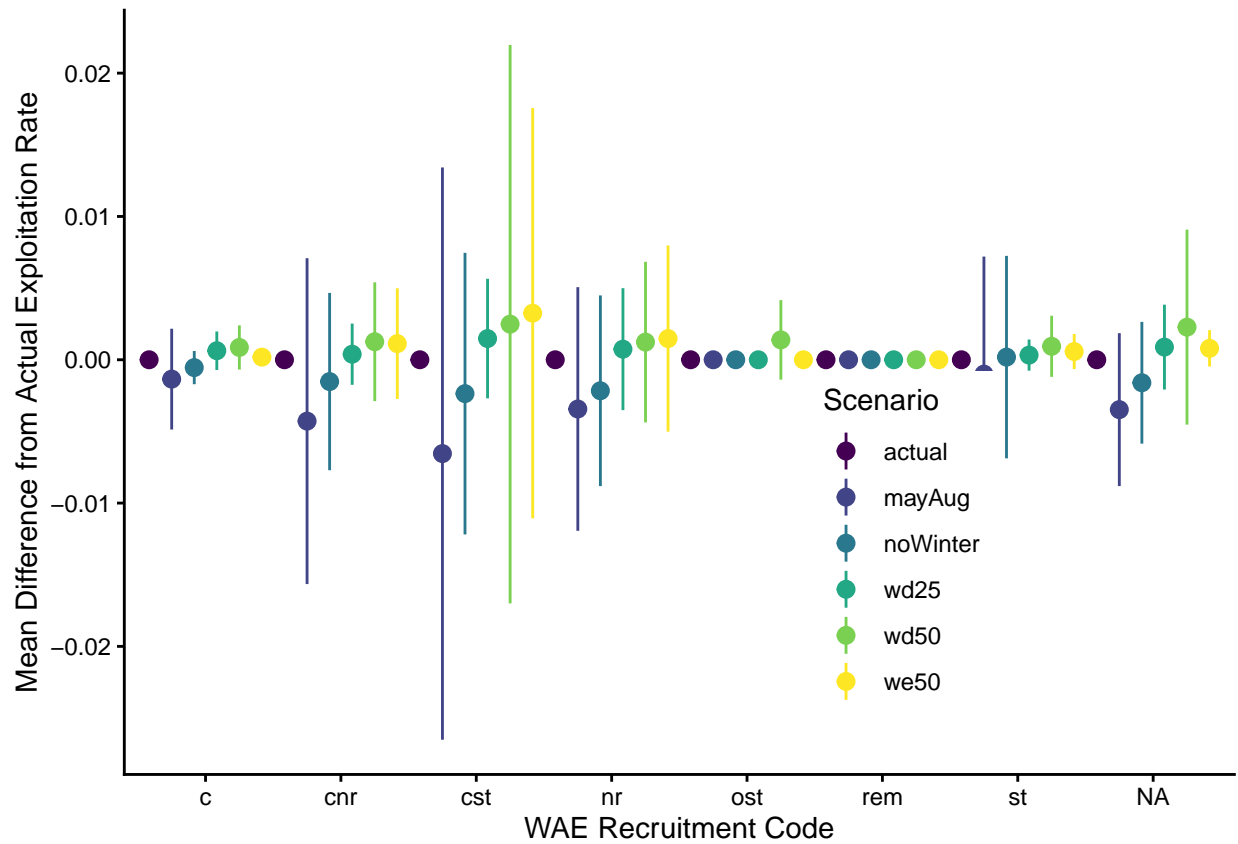
Figure 7: Comparison of the magnitutde of the difference between the mean actual exploitation rate and the estimate derived from each reduced data set. Vertical lines represent 1 standard deviation. Walleye recruitment codes are along the x axis. Note the y-axis scale, many of the differences are small. 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.
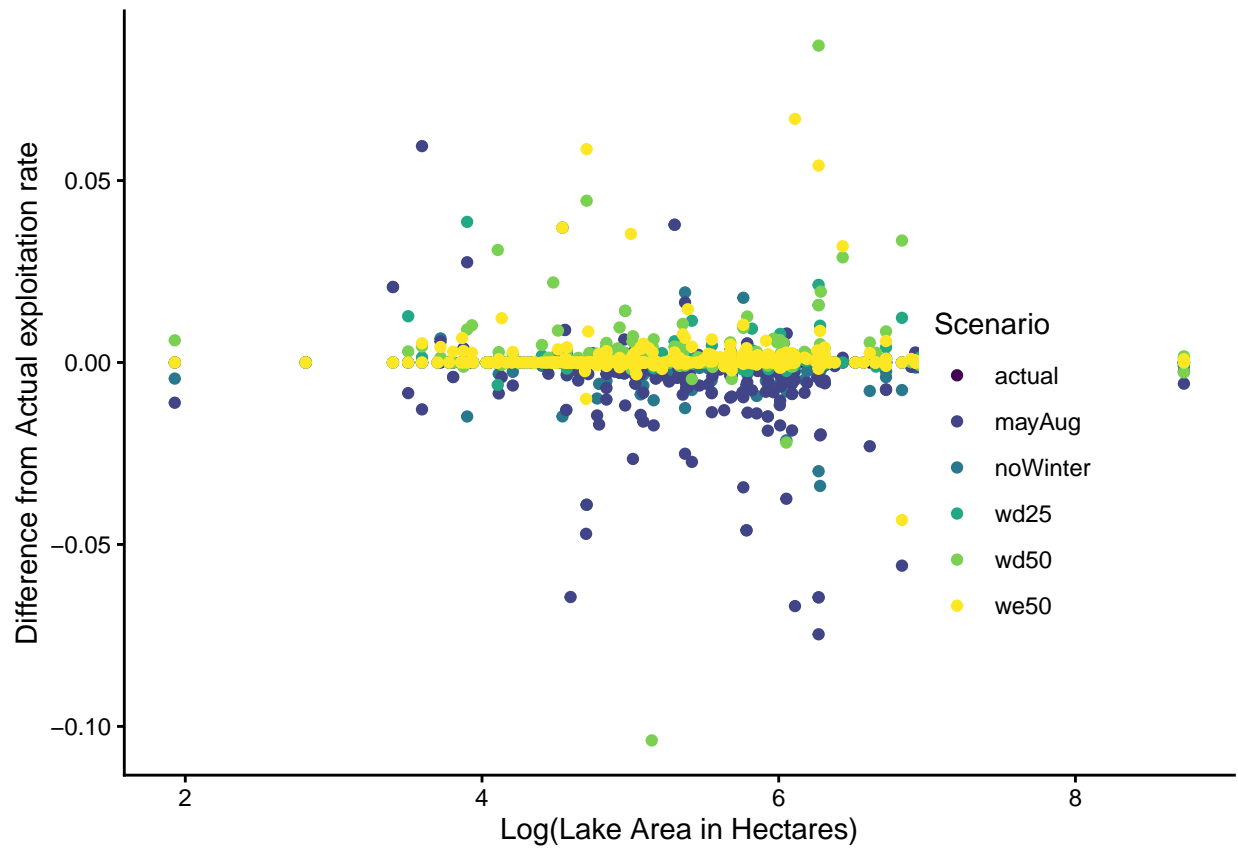
Figure 8: Comparison of the magnitude of the difference between the mean actual exploitation rate and the estimate derived from each reduced data set. Note the y-axis scale, many of the differences are small.

Figure 9: Comparison of the magnitude of the difference between the mean actual exploitation rate and the estimate derived from each reduced data set. Note the y-axis scale, many of the differences are small.
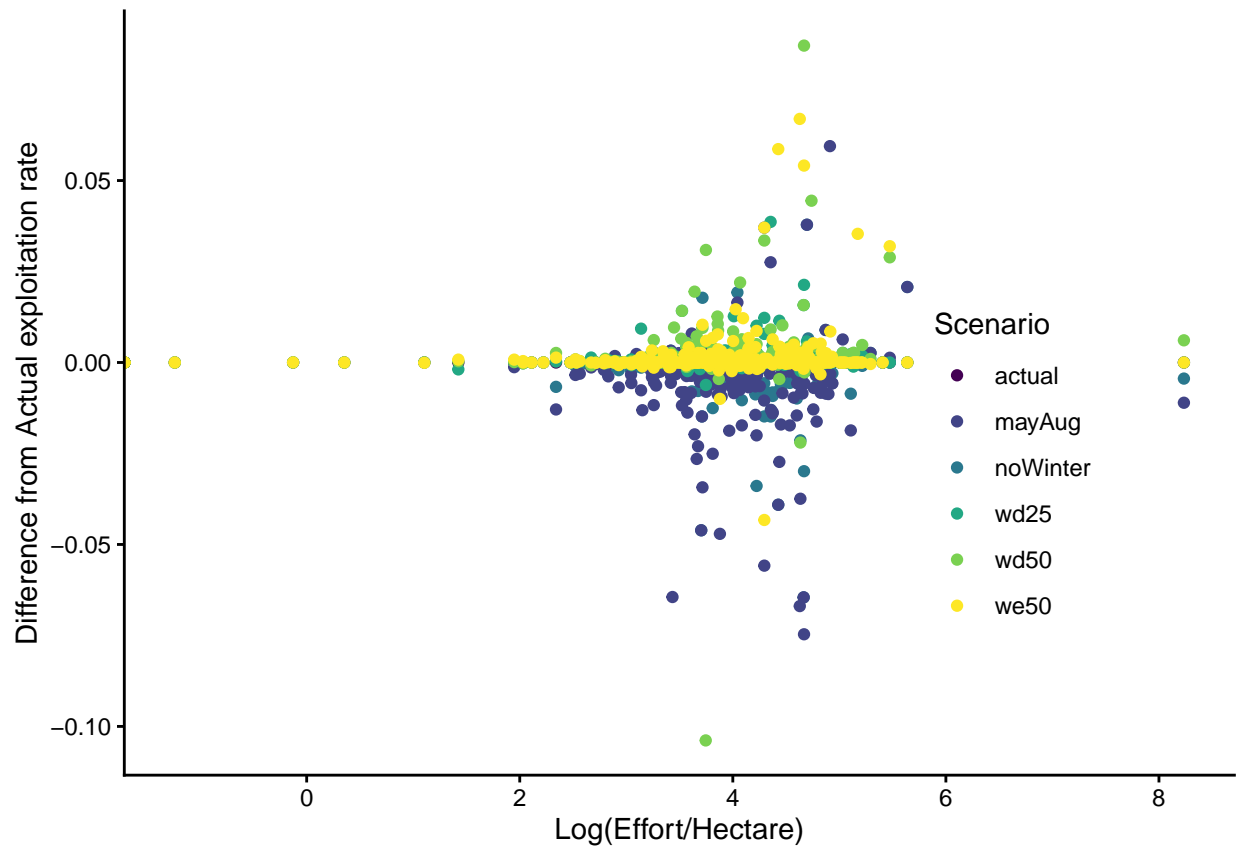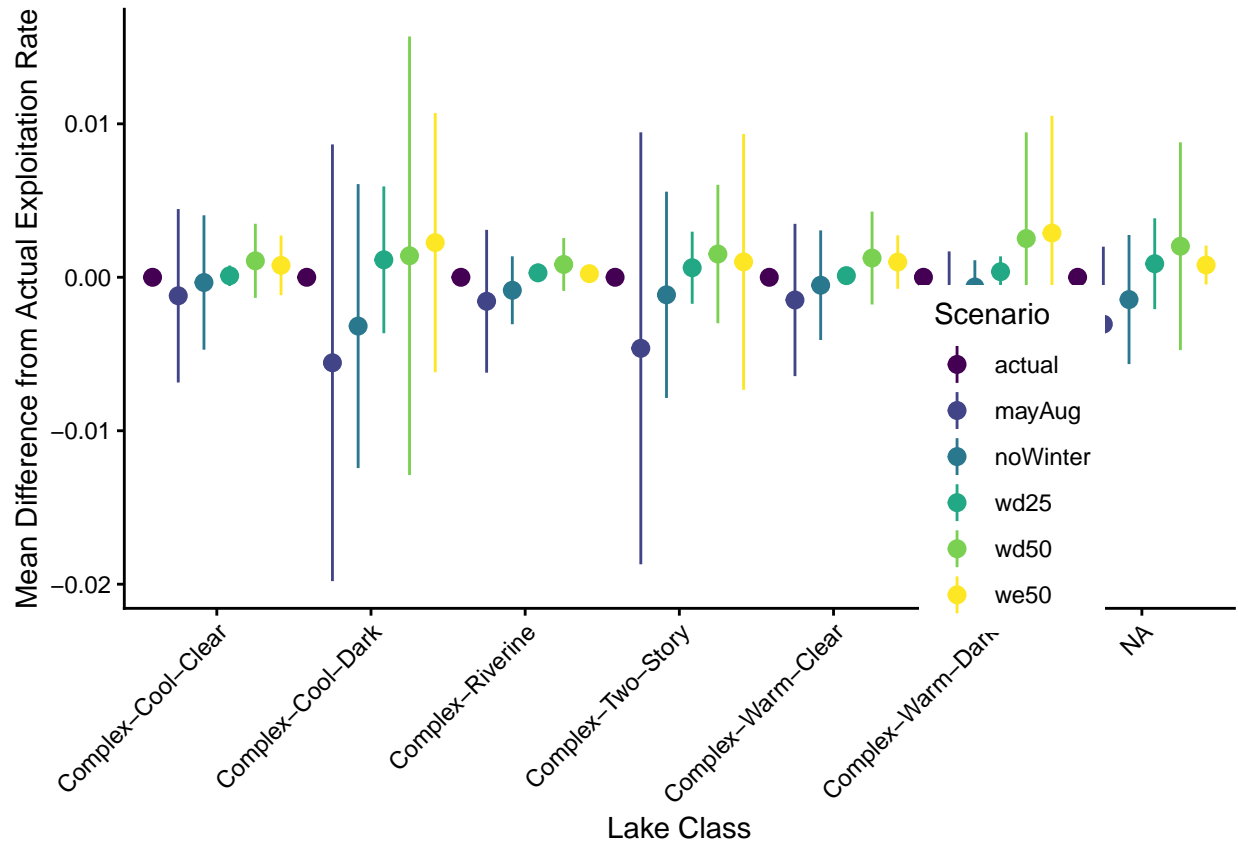
Figure 10: Comparison of the magnitude of the difference between the mean actual exploitation rate and the estimate derived from each reduced data set. Vertical lines represent 1 standard deviation. Note the y-axis scale, many of the differences are very small. Lake classes are according to Rypel et al. 2019, https://doi.org/10.1002/fsh.10228 .

```r
    us = rlnorm(nrow(tdat[tdat$treat == "mayAug", ]), meanlog = alpha,
        sdlog = beta)
    ll = dlnorm(tdat$exp.rate[tdat$treat == "mayAug"], meanlog = mean(log(us)),
        sdlog = sd(log(us)), log = T)
    return(sum(ll))
}
uLL.nw = function(param) {
    alpha = param[1]
    beta = param[2]

    us = rlnorm(nrow(tdat[tdat$treat == "noWinter", ]), meanlog = alpha,
        sdlog = beta)
    ll = dlnorm(tdat$exp.rate[tdat$treat == "noWinter"], meanlog = mean(log(us)),
        sdlog = sd(log(us)), log = T)
    return(sum(ll))
}
uLL.a = function(param) {
    alpha = param[1]
    beta = param[2]

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

# removing 0s
ttrExp = ttrExp[ttrExp$exp.rate != 0, ]
loopY = sort(unique(ttrExp$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.prsf = NA,
    gr.par1 = NA, gr.par2 = NA)

for (y in 1:length(loopY)) {
    # first get to year-specific data
    tdat = ttrExp[ttrExp$year == loopY[y], ]
    # removing 0s,
    tdat = tdat[tdat$exp.rate != 0, ]
    # Bayesian Model Fitting ACTUAL

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

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

    settings = list(iterations = 10000, nrChains = 3, message = F,
        burnin = 5000)
    set.seed(10)
```

```r
t.a = runMCMC(bayesianSetup = setup.a, sampler = "DEzs",
    settings = settings)
# NW

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

setup.nw = createBayesianSetup(uLL.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$exp.rate[ttrExp$treat ==
    "mayAug"]))), sd(log(ttrExp$exp.rate[ttrExp$treat == "mayAug"]))),
    sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.ma = createBayesianSetup(uLL.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$exp.rate[ttrExp$treat ==
    "wd25"]))), sd(log(ttrExp$exp.rate[ttrExp$treat == "wd25"]))),
    sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.wd25 = createBayesianSetup(uLL.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$exp.rate[ttrExp$treat ==
    "wd50"]))), sd(log(ttrExp$exp.rate[ttrExp$treat == "wd50"]))),
    sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

setup.wd50 = createBayesianSetup(uLL.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$exp.rate[ttrExp$treat ==
        "we50"])), sd(log(ttrExp$exp.rate[ttrExp$treat == "we50"]))),
        sd = c(1, 1), lower = c(-15, 0), upper = c(15, 5))

    setup.we50 = createBayesianSetup(uLL.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$exp.rate[tdat$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
```

```r
    actual.sd = sd(log(tdat$exp.rate[tdat$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$exp.rate[tdat$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$exp.rate[tdat$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$exp.rate[tdat$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$exp.rate[tdat$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)) {
```

```r
    tempdat = rlnorm(n = length(tdat$exp.rate[tdat$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$exp.rate[tdat$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$exp.rate[tdat$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$exp.rate[tdat$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$exp.rate[tdat$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
```

```r
# of the real data

pval.we50$sdExceed = 0

we50.sd = sd(log(tdat$exp.rate[tdat$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.prsf = 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_explRate_9.30.25.RData')
```
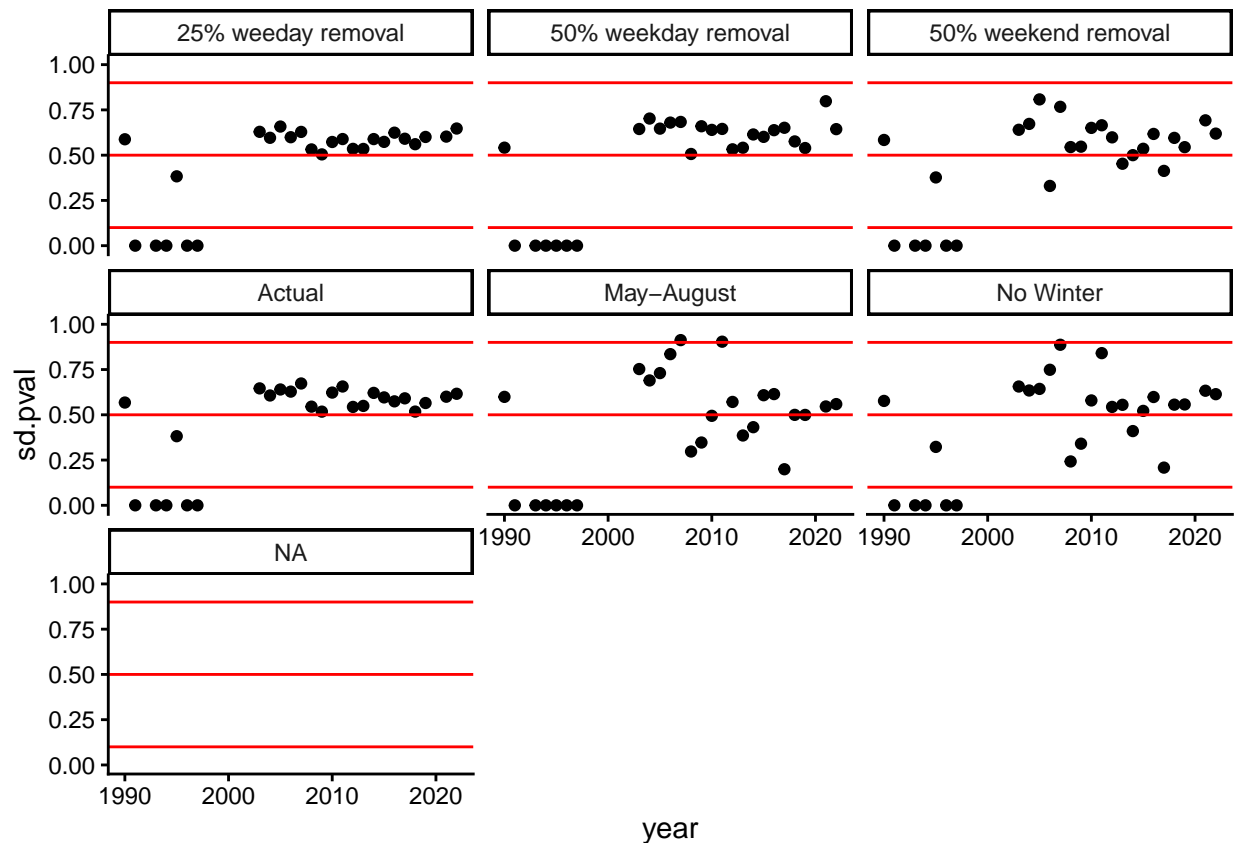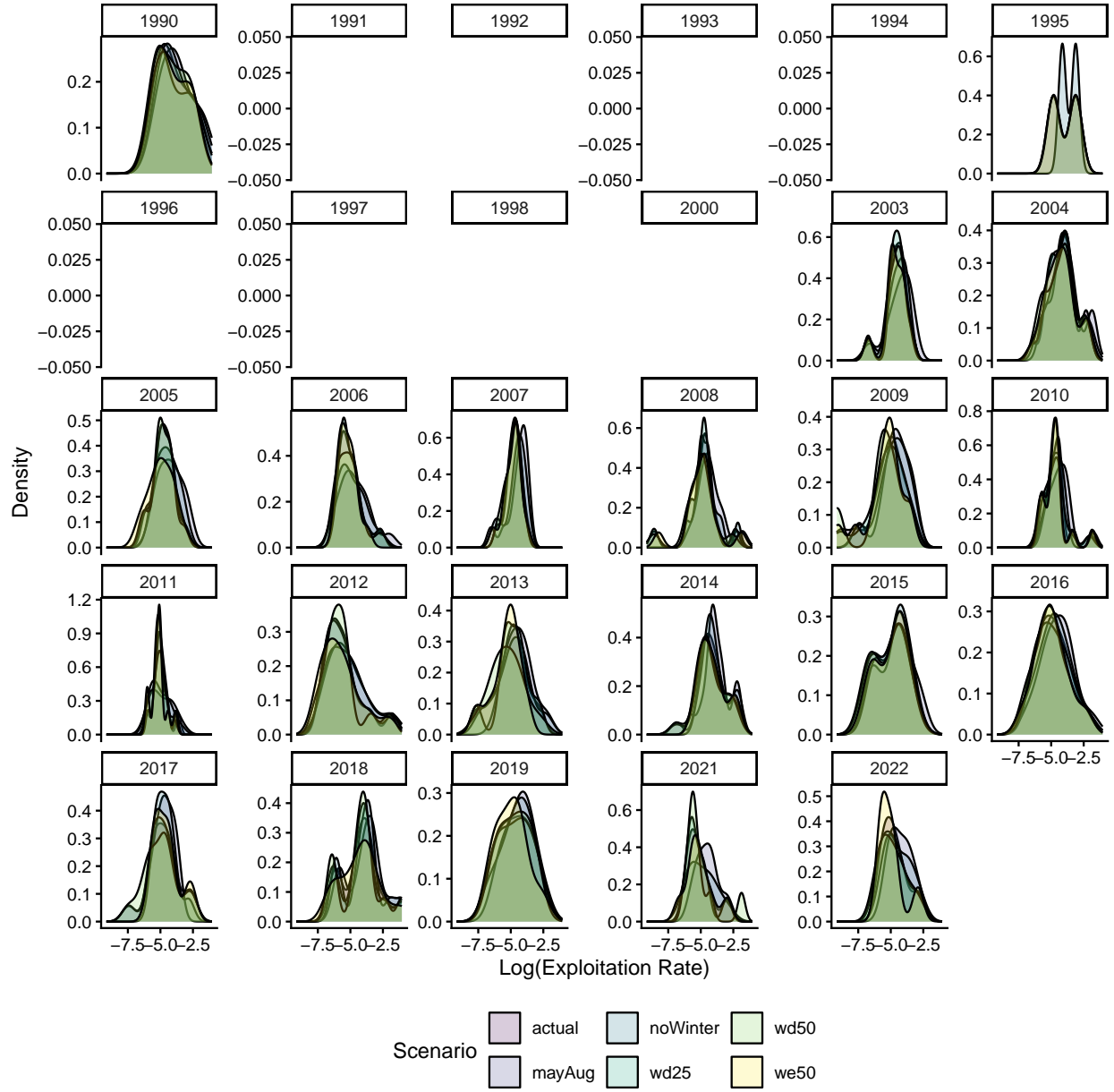
Figure 11: Distribution of exploitation rates across years in the creel data set. Different data reduction scenarios are noted with varying colors.
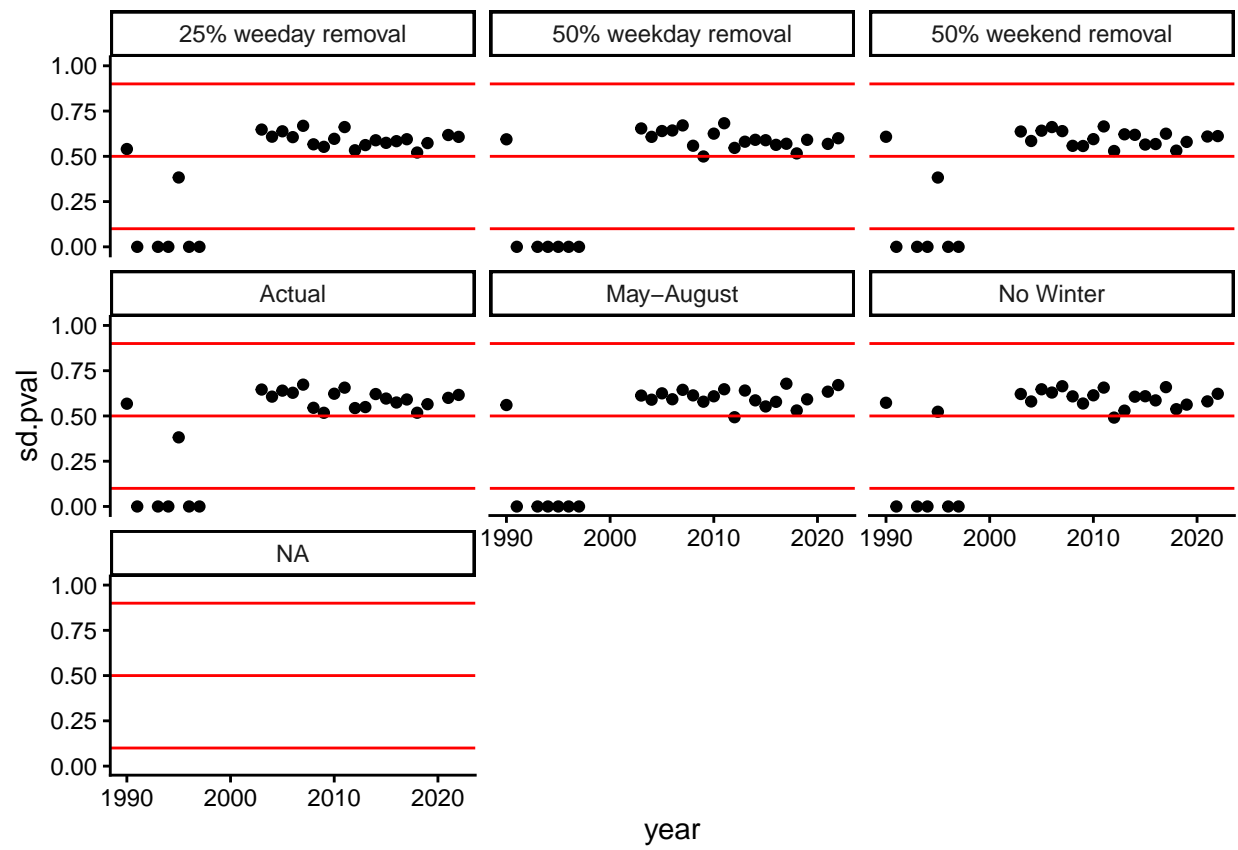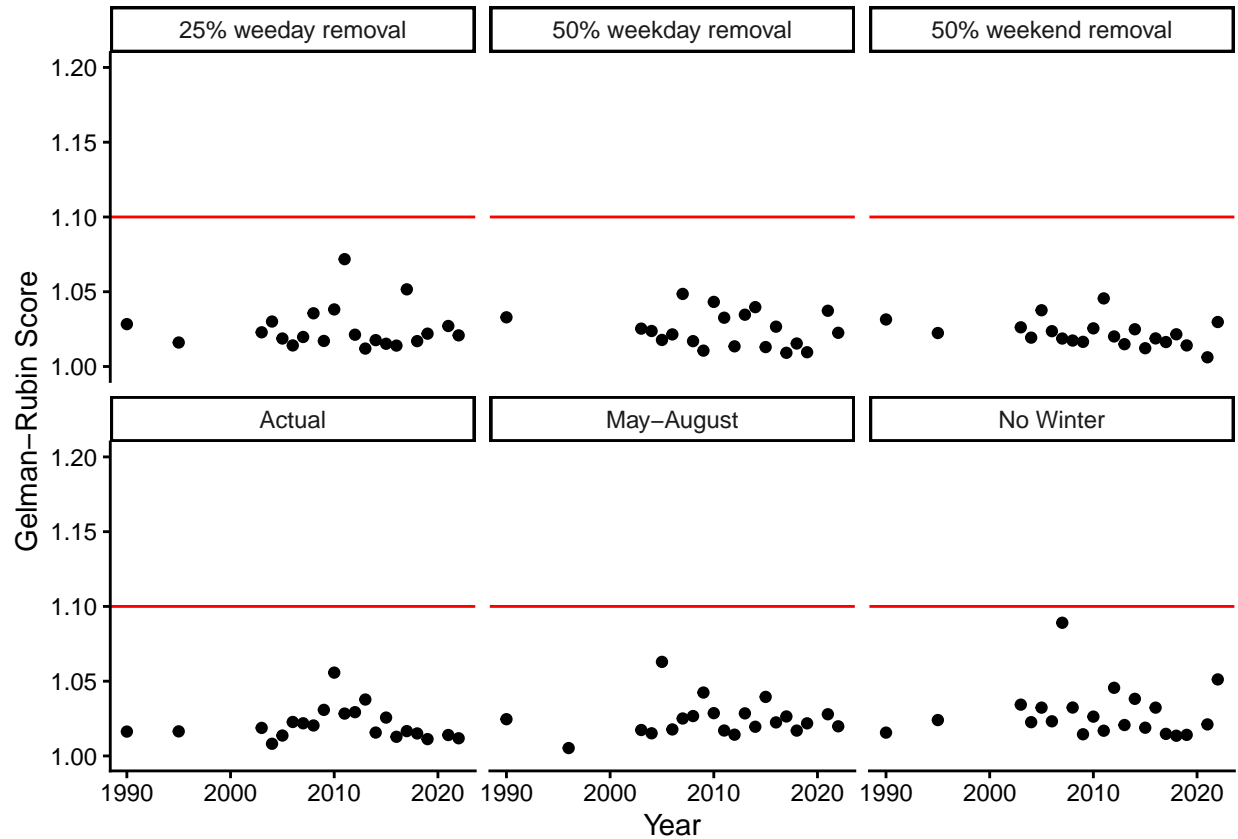
Figure 12: Comparison of standard deviations for bayesian p-values comparing the actual data to the simulated data for the creel year and data reduction scenario.

1.2.5 Alternative modeling distributions

Instead of the lognormal distribution there are two 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.
Beta	Continuous random variable than can take on values between 0 and 1. Any random variable that can be expressed as a proportion.

The following analyses will compare the ability of models using the lognormal and beta distributions to fit the full data and each of the reduced data set. The gamma distribution is not included in this analysis. 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 13). The purpose of the comparison between lognormal and beta family models is not to see which model provided the most convenient answer

26

Table 6: Bayesian p-values for a beta 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 devidation (SD) was the variance metric assessed here.

Scenario	SD p-value
Actual	0.001
No Winter	0.011
May-August	0.088
25% weeday removal	0.000
50% weekday removal	0.001
50% weekend removal	0.000

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 data set and **not** across data sets which is what is necessary to gain inference on the effect of any data reductions.
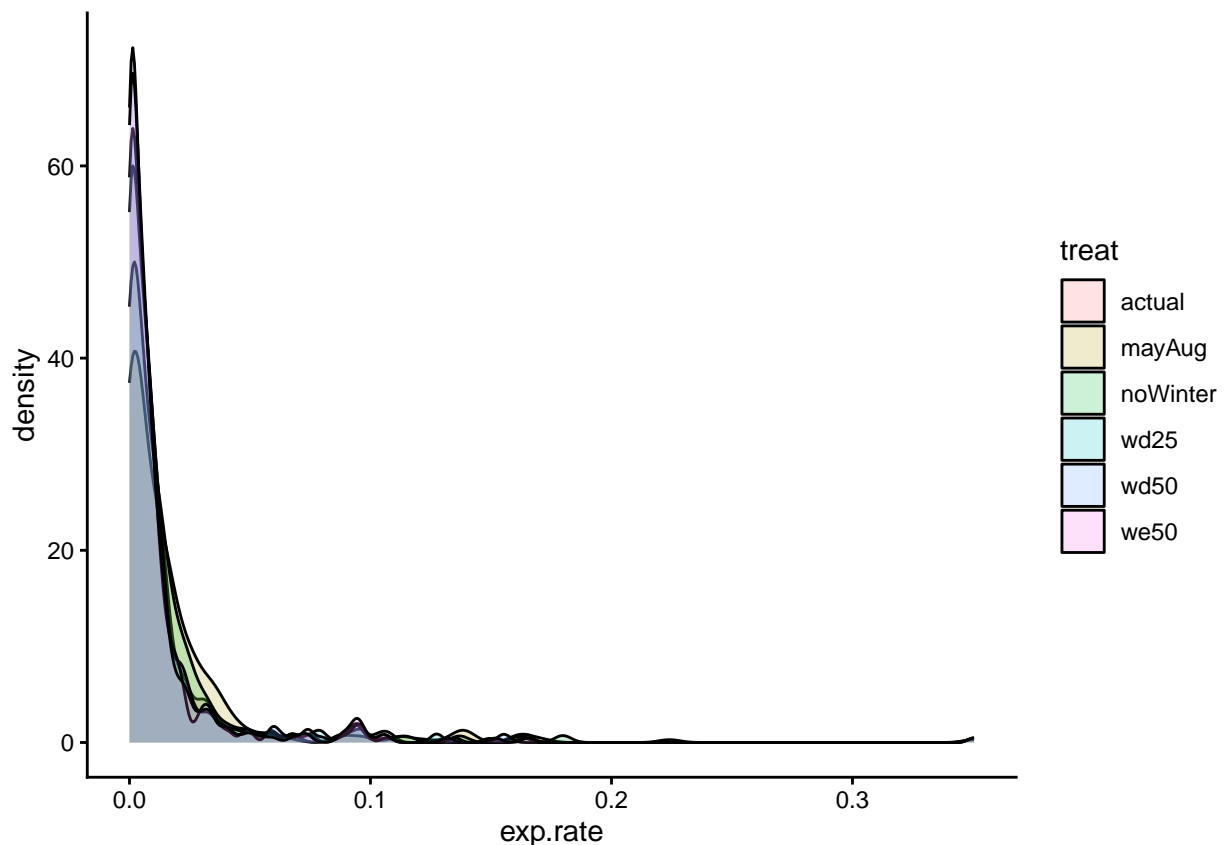

```r
### MODEL CHECKNG ####

gelmanDiagnostics(expR.actual.beta)   # converged
gelmanDiagnostics(expR.nw.beta)   # converged
gelmanDiagnostics(expR.ma.beta)   # converged
gelmanDiagnostics(expR.25wd.beta)   # NOT converged
gelmanDiagnostics(expR.50wd.beta)   # converged
gelmanDiagnostics(expR.50we.beta)   # converged
```
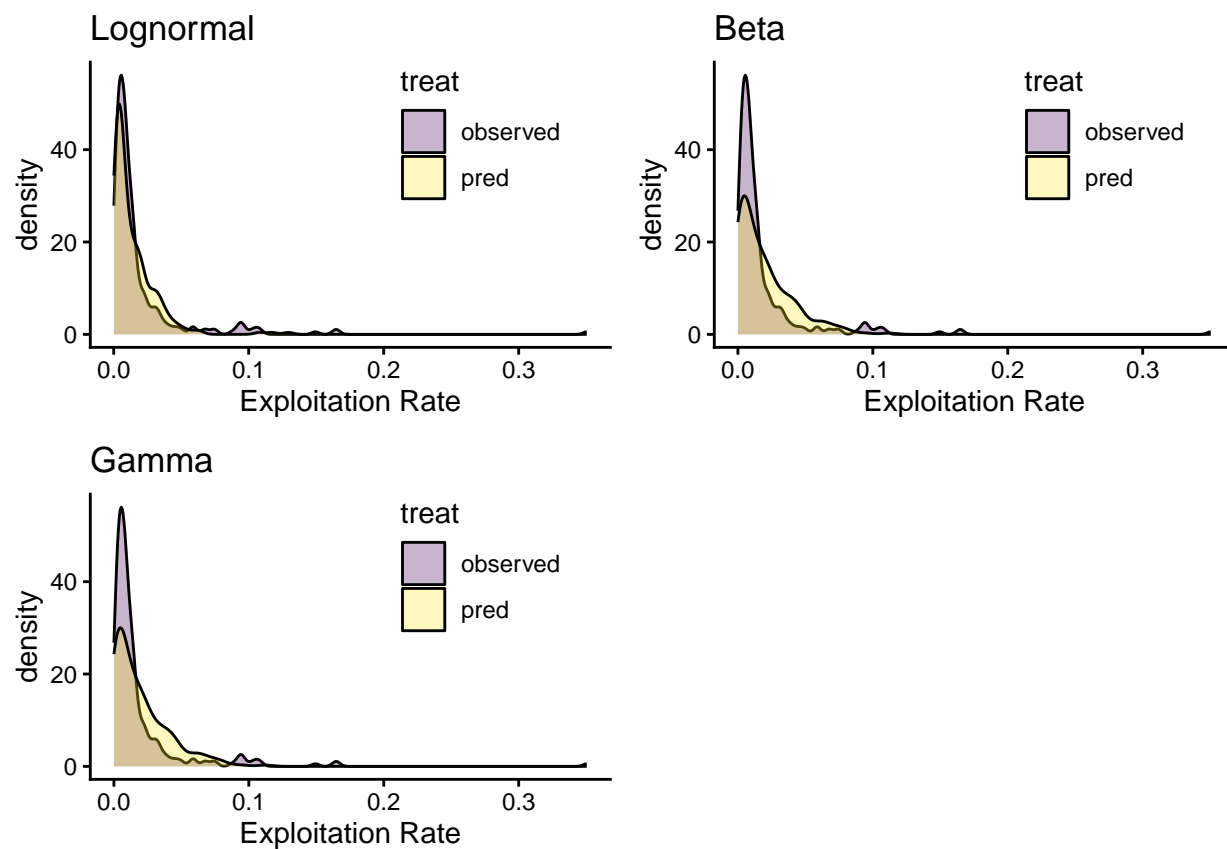
Figure 13: Comparison of model fits for lognormal, gamma, and beta family of models. This rough look at the models suggests there are no obvious differences between the choice of modeling distribution. This will be evaluated statistically using bayesian p values later on. Based on what I see here I've dropped the gamma family of models from further analysis since the definition of gamma data is not really different from the lognormal data and beta better fits the data description than gamma.
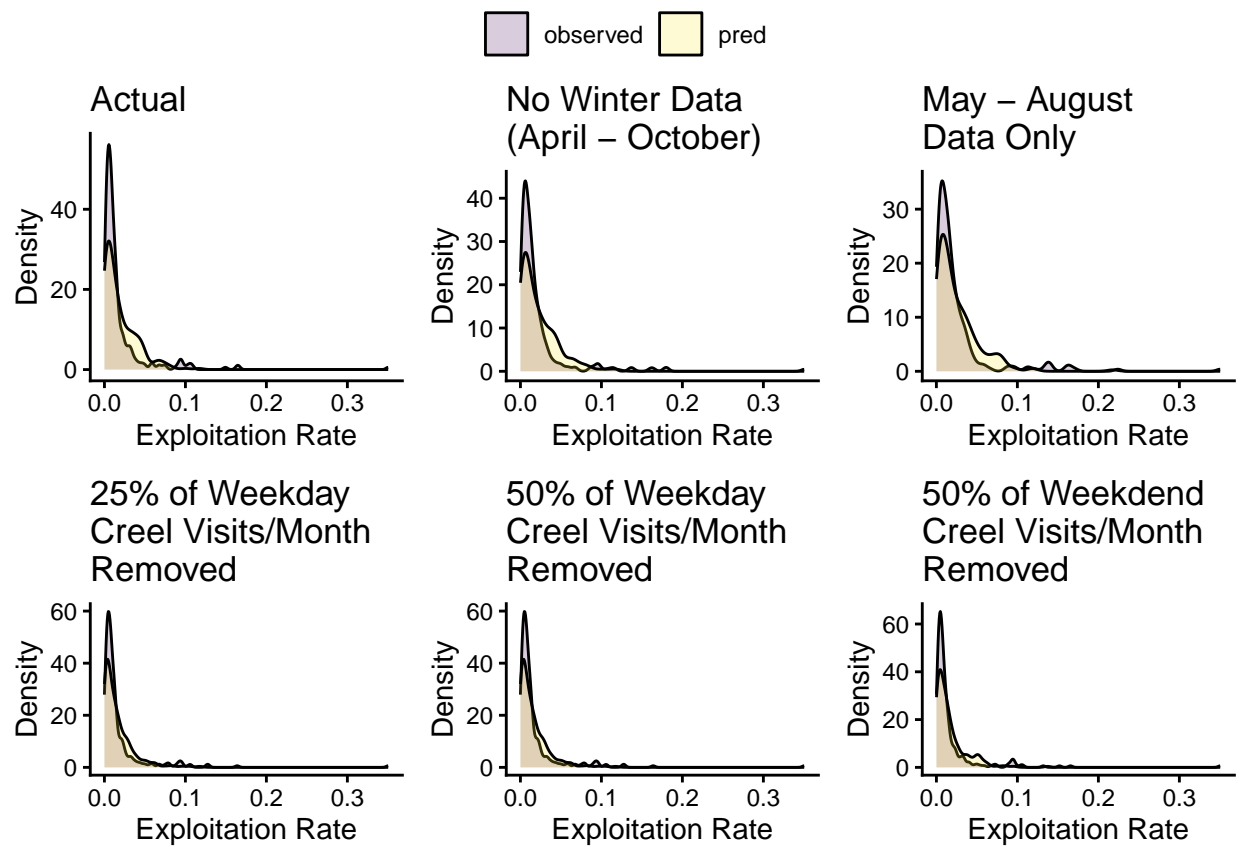
Figure 14: Visualization of the model fit to the data for each scenario using a beta distribution instead of a lognormal distribution.

The Bayesian p-values in the table presented here suggests the beta models do a poor job of reproducing the data they were fit to (Table 6). This inference comes from the majority of p-values being outside the 0.1-0.9 range for the SD metric. The abundance of significant p-values across model scenarios and metrics points to the inability of models fit to the creel data using a beta distribution to be able to reproduce that data. Because each model is unable to reproduce the data it was fit to, we know that this family of models is not a useful family for fitting these data. This means that the beta 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 exploitation rates or not.

1.2.6   P-value sensitivity analysis

1.2.6.1   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.

1.2.6.1.1   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.**

1.2.6.1.2   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.**

1.2.6.1.3   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,

Table 7: 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
Kurtosis	Measure of the width, or 'tailedness' of a distribution. In other words, do the tails of the distribution contain
X2	Chi square test statistic for a 'goodness of fit' test is calculated. Here I want to know if frequency of values in
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 paramter will contain the true value of the data.

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.**

1.2.6.1.4 Chi Square Test (χ^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 χ^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.**

1.2.6.1.5 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**

1.2.6.1.6 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.**
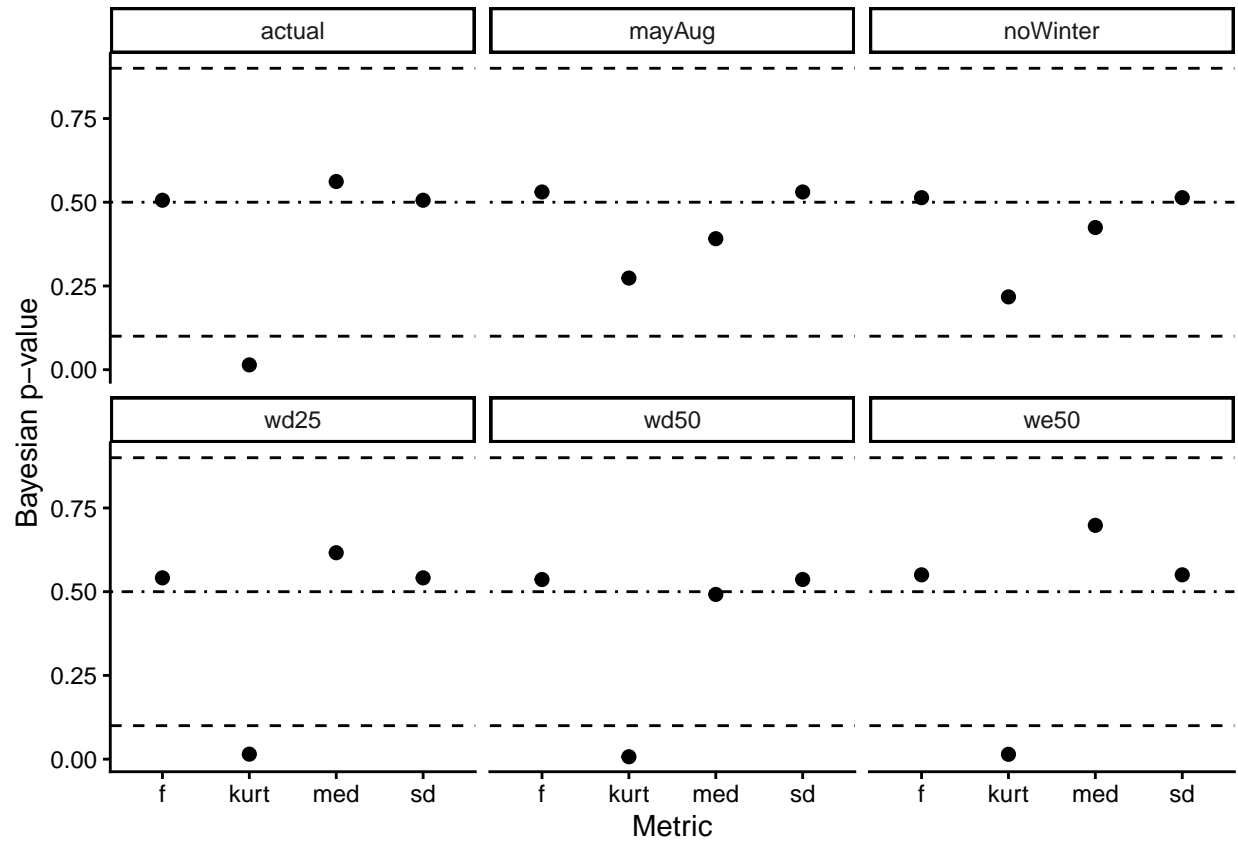
Figure 15: Comparison of multiple metrics for calculating Bayesian p-values. Each panel is a different data scenario. The comparisons in each panel are comparing the model simulated data to the observed data for that same sencario, NOT comparing model simulated data from a reduction scenario tot he actual 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 betweek model simulated data and the observed data).