Problem Set 1

Mitchell Linegar

2020-04-30

Contents


```
#### SETUP ####
```

echo=TRUE

```
# set global options
dataset_name <- "welfare"
outcome_family <- "binomial" # based on whether your outcome is binary or not; input to glm call
outcome_type <- "class"
n_sims <- 20
prop_to_keep <- 1.0 # if you want to only run on a random sample of the data, if want to run on full da

# lambda <- c(0.0001, 0.001, 0.01, 0.1, 0.3, 0.5) #, 0.7, 1, 5, 10, 50, 100, 1000)
# lambda <- c(0.0001, 0.01)
lambda <- c(0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 1, 5, 10, 50, 100, 1000)
prop_drop_rf <- c(0.01, 0.02, 0.04, 0.1, 0.2, 0.3, 0.4, .5, .7, .8, .9)

library(here)
# devtools::install_github("hrbrmstr/hrbrthemes")
# library(hrbrthemes)
library(ggplot2)
theme_set(theme_classic())
library(data.table)
library(tidyverse)
library(broom)
```

```
library(grf)
library(sandwich)
devtools::install_github("swager/amlinear") # install amlinear package
```

```
## Skipping install of 'amlinear' from a github remote, the SHA1 (83ee1d18) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
library(amlinear)

#### KNITR SETUP ####
```

Part One

This problem set examines the welfare data set. Throughout, the treatment variable will be referred to as W and the outcome variable will be referred to as Y.
Note that models with the suffix `.int` refer to our interacted data; we work with this expanded dataset to demonstrate the usefulness of machine learning methods in higher dimensions.
Collaborators I worked closely on this problem set with Ayush Kanodia. I also worked with Kaleb Javier and Haviland Sheldahl-Thomason, and indicate where we collaborated on code.

Pre-Processing the Data

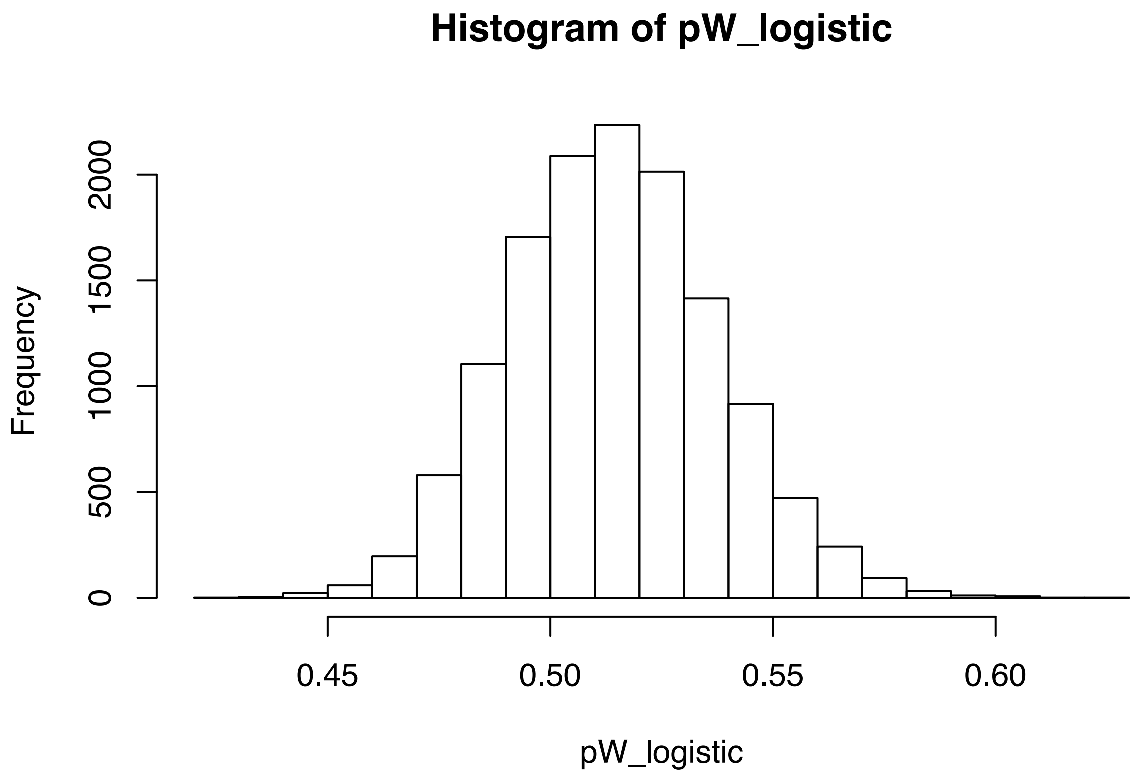
I use code from a set of AtheyLab tutorials, which include the following note:
> The datasets in our github webpage have been prepared for analysis so they will not require a lot of cleaning and manipulation, but let's do some minimal housekeeping. First, we will drop the columns that aren't outcomes, treatments or (pre-treatment) covariates, since we won't be using those. Specifically, we keep only a subset of predictors and drop observations with missing information.

Testing Assumptions

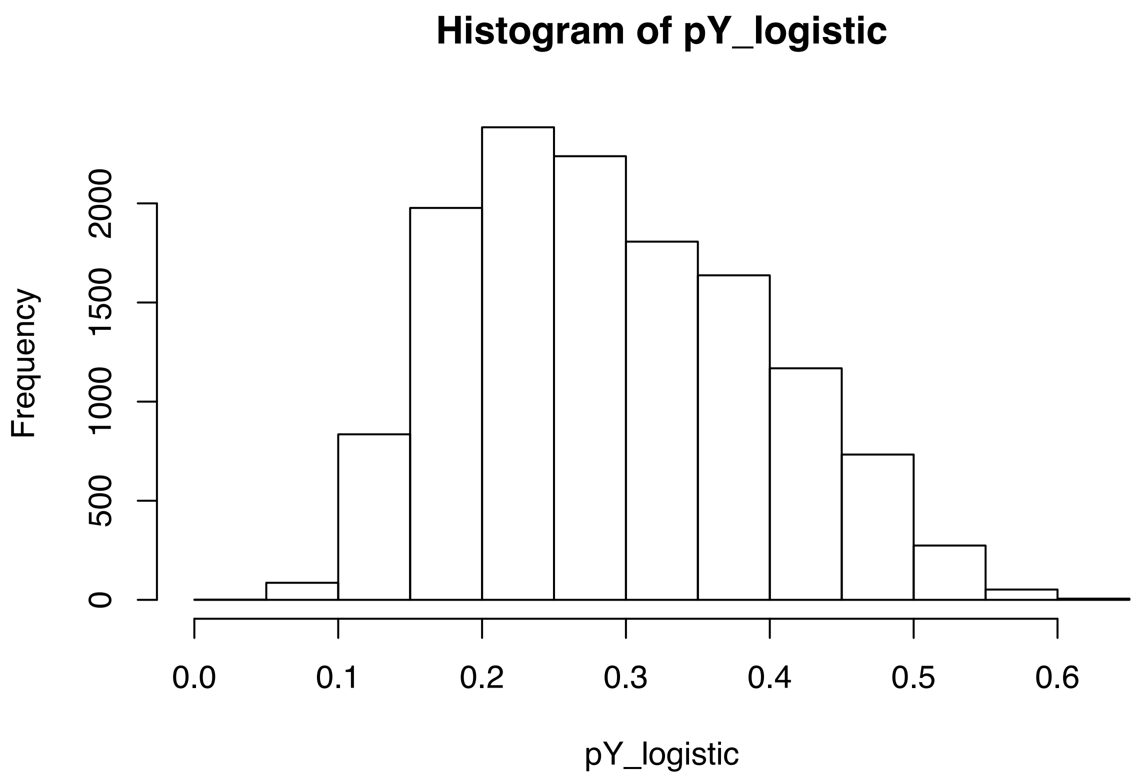
Here we test some of our traditional causal inference assumptions.

As a first step, we plot logistic predictions of the probabilities our treatment pW and our outcome pY. We see that treatment assignment appears to follow a normal distribution, and that our outcome has an average unconditional probability of 0.3.

```
pW_logistic.fit <- glm(Wmod ~ as.matrix(Xmod), family = "binomial")
pW_logistic <- predict(pW_logistic.fit, type = "response")
pW_logistic.fit.tidy <-  pW_logistic.fit %>% tidy()
hist(pW_logistic)
```



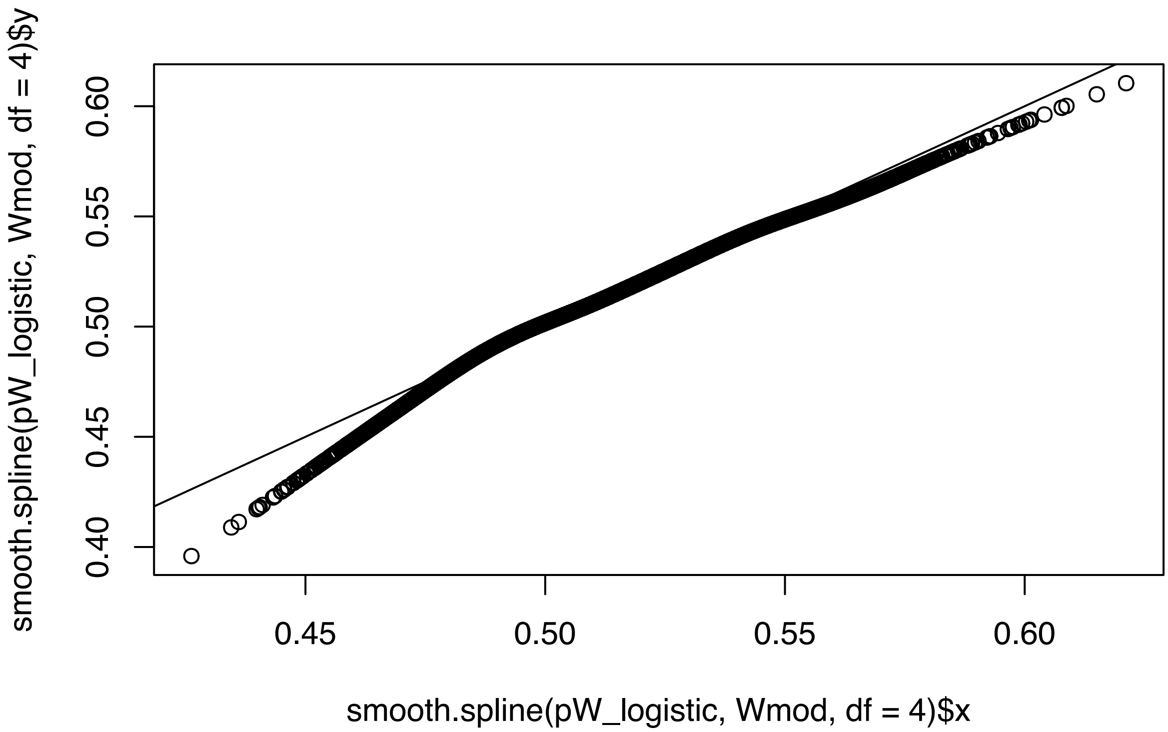

```
pY_logistic.fit <- glm(Ymod ~ as.matrix(Xmod), family = "binomial")
pY_logistic <- predict(pY_logistic.fit, type = "response")
pY_logistic.fit.tidy <-  pY_logistic.fit %>% tidy()
hist(pY_logistic)
```




```
df_mod[, `:=`(p_Y = pY_logistic,
              p_W = pW_logistic)]
```

We now produce a plot comparing predicted and actual treatment assignment. This plot is provided mostly for comparison (this is the plot the tutorial has); future plots of this nature will be done with `ggplot` to make their options more explicit.

```
{plot(smooth.spline(pW_logistic, Wmod, df = 4))
  abline(0, 1)}
```




```
#### RCT ANALYSIS ####
```

RCT Analysis

We now report the (presumably true) treatment effect $\hat{\tau}$ from the randomized experiment:

```
tauhat_rct <- difference_in_means(df)
print(tauhat_rct)
```

```
##      ATE lower_ci upper_ci
##   -0.370   -0.384   -0.355
```

```
#### SAMPLING BIAS ####
```

Introducing Bias

We now introduce sampling bias in order to simulate the situation we would be in if our data was from an observational study rather than a randomized experiment. This situation might arise due to sampling error or selection bias, and we will be able to see how various methods correct for this induced bias. To do, so, we under-sample treated units matching the following rule, and under-sample control units in its complement: - Independents on closer to the Democratic party (`partyid < 4`) - Who have at least a college degree (`educ >= 16`)

We remove 40.0 percent of observations in these sets.

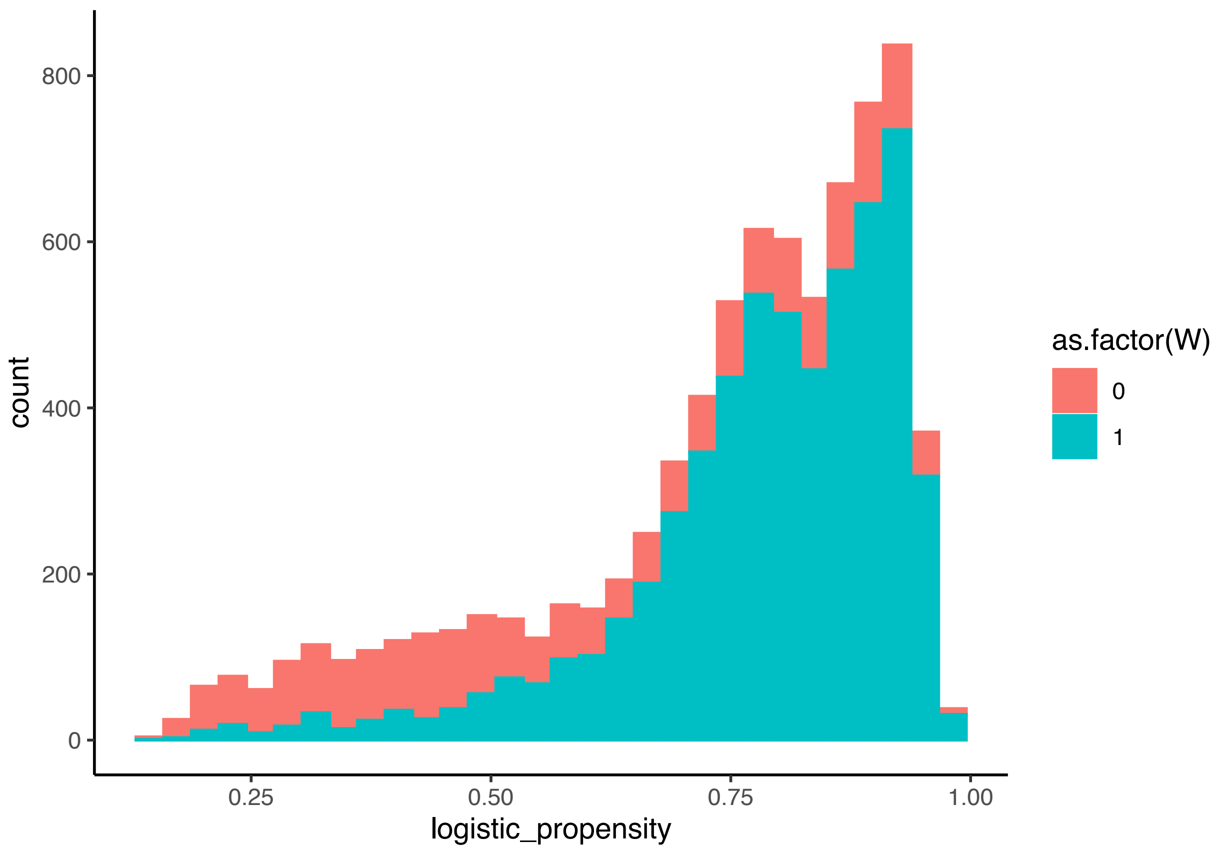
The difference in means is now biased, and significantly outside the confidence interval indicated by the RCT. Check if difference in treatment effect estimates is substantial

```
##      ATE lower_ci upper_ci
##   -0.259   -0.273   -0.245
```

```
# df_mod <- copy(df)
Xmod = df_mod[,.SD, .SDcols = names(df_mod)[!names(df_mod) %in% c("Y", "W")]] %>% as.matrix()
Ymod = df_mod$Y
Wmod = df_mod$W
XWmod = cbind(Xmod, Wmod)

# Computing the propensity score by logistic regression of W on X.
pW_logistic.fit <- glm(Wmod ~ as.matrix(Xmod), family = "binomial")
pW_logistic <- predict(pW_logistic.fit, type = "response")

df_mod[, logistic_propensity := pW_logistic]

#### OVERLAP ####
```

We now plot (logistic) propensity scores, showing that we still have overlap after removing observations. We may be somewhat concerned about the small number of observations with propensities close to one; however, they are small in number and not exactly one so we are leaving them in for now.

```
overlap <- df_mod %>% ggplot(aes(x=logistic_propensity,color=as.factor(W),fill=as.factor(W)))+ geom_his
overlap
```



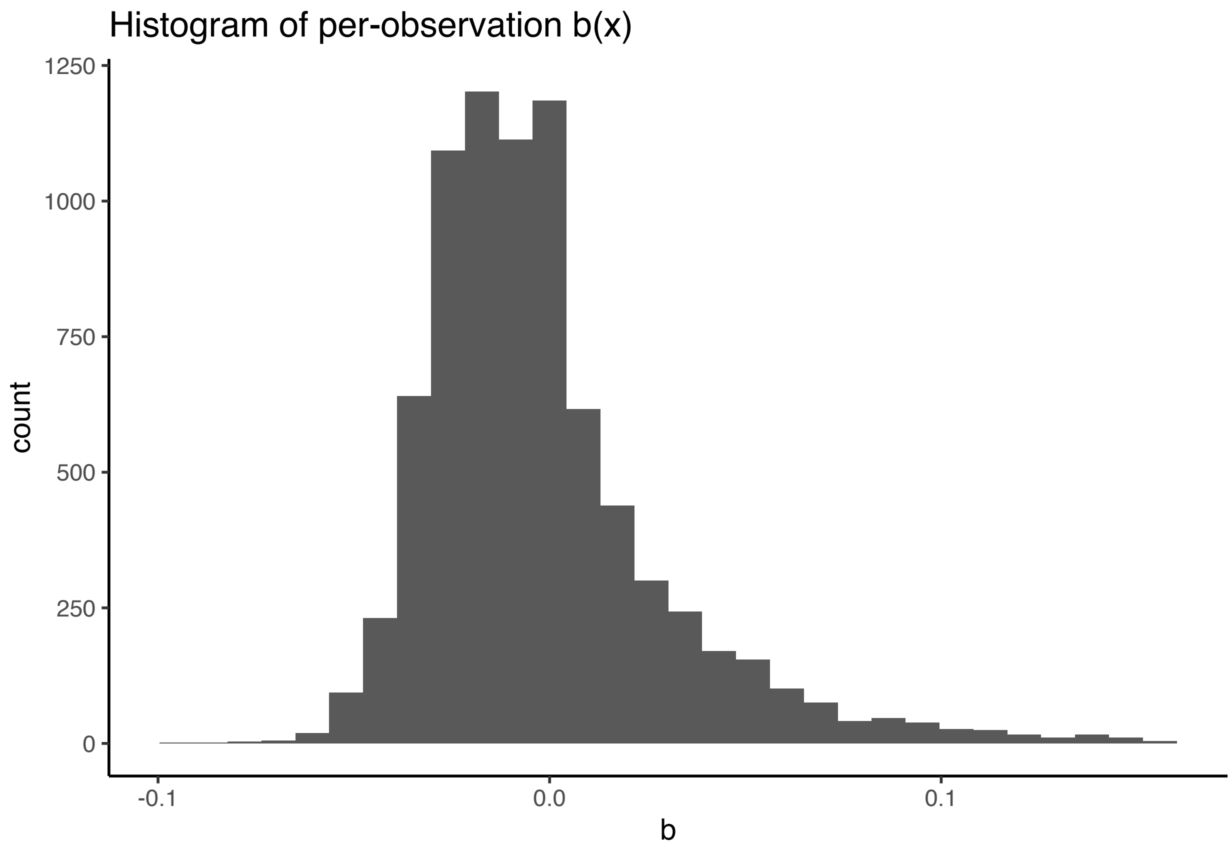

```
#### PREDICTING PROPENSITIES AND OUTCOMES, ORIGINAL AND EXPANDED DATA ####
# some of this is used to calculate bias function, hence the ordering

# expand data
Xmod.int = model.matrix(~ . * ., data = as.data.frame(Xmod))
XWmod.int = cbind(Xmod.int, Wmod)
# make expanded df
```

```r
df_mod.int <- Xmod.int %>% as.data.frame %>% setDT()
df_mod.int[,`:=`(W = Wmod, Y = Ymod)]

# logistic model
# original data
pW_logistic.fit <- glm(Wmod ~ Xmod, family = "binomial")
pW_logistic <- predict(pW_logistic.fit, type = "response")
# expanded data
pW_logistic.fit.int <- glm(Wmod ~ Xmod.int, family = "binomial")
pW_logistic.int <- predict(pW_logistic.fit.int, type = "response")

# original data
pY_logistic.fit <- glm(Ymod ~ XWmod, family = outcome_family)
pY_logistic <- predict(pY_logistic.fit, type = "response")
# pY_logistic2 <- predict(pY_logistic.fit, newdata = as.data.frame(XWmod), type = "response")
# expanded data
pY_logistic.fit.int <- glm(Ymod ~ Xmod.int, family = outcome_family)
pY_logistic.int <- predict(pY_logistic.fit.int, type = "response")

# lasso expanded data, code provided by TA
# original data
pW_glmnet.fit.model = glmnet::cv.glmnet(Xmod, Wmod, lambda = lambda, family = "binomial", type.measure =
pY_glmnet.fit.model = glmnet::cv.glmnet(Xmod, Ymod, lambda = lambda, family = outcome_family, type.meas
# expanded data
pW_glmnet.fit.model.int = glmnet::cv.glmnet(Xmod.int, Wmod, lambda = lambda, family = "binomial", type.m
pY_glmnet.fit.model.int = glmnet::cv.glmnet(Xmod.int, Ymod, lambda = lambda, family = outcome_family, ty

# demonstration of lasso fit across lambdas:
pW_lasso = pW_glmnet.fit.model$fit.preval[, pW_glmnet.fit.model$lambda == pW_glmnet.fit.model$lambda.mi
pW_lasso.min = pW_glmnet.fit.model$fit.preval[, pW_glmnet.fit.model$lambda == min(pW_glmnet.fit.model$l
pW_lasso.max = pW_glmnet.fit.model$fit.preval[, pW_glmnet.fit.model$lambda == max(pW_glmnet.fit.model$l
pW_lasso.rand = pW_glmnet.fit.model$fit.preval[, pW_glmnet.fit.model$lambda == base::sample(pW_glmnet.f

# glmnet.fit.model = glmnet::cv.glmnet(Xmod.int, Wmod, family = "binomial",keep=TRUE)
pW_lasso.int = pW_glmnet.fit.model.int$fit.preval[, pW_glmnet.fit.model.int$lambda == pW_glmnet.fit.mod
pW_lasso.int.min = pW_glmnet.fit.model.int$fit.preval[, pW_glmnet.fit.model.int$lambda == min(pW_glmnet
pW_lasso.int.max = pW_glmnet.fit.model.int$fit.preval[, pW_glmnet.fit.model.int$lambda == max(pW_glmnet
pW_lasso.int.rand = pW_glmnet.fit.model.int$fit.preval[, pW_glmnet.fit.model.int$lambda == base::sample

# FIXME:
# problems calculating probabilities:
# this "should" work:
# pW_lasso.int2 <- predict(pW_glmnet.fit.model.int, newx = Xmod.int, type = "response")
# however, it returns identical predictions for every entry. This is the same if we specify a lambda, t
# pW_lasso.int3 <- predict(pW_glmnet.fit.model.int, newx = Xmod.int,  s=pW_glmnet.fit.model.int$lambda.
# pW_lasso.int4 <- predict(pW_glmnet.fit.model.int, newx = Xmod.int,  s="lambda.min", type = "response"
# pW_lasso.int5 <- predict(pW_glmnet.fit.model.int, newx = Xmod.int,  s=pW_glmnet.fit.model.int$lambda[
# pW_lasso.int6 <- predict(pW_glmnet.fit.model.int, newx = Xmod.int,  s=pW_glmnet.fit.model.int$lambda[

# random forest
pW_rf.fit = regression_forest(Xmod, Wmod, num.trees = 500)
pY_rf.fit = regression_forest(Xmod, Ymod, num.trees = 500)
```

```r
# pW_rf = pW_rf.fit$predictions
# pY_rf = pY_rf.fit$predictions
pW_rf = predict(pW_rf.fit, newdata = Xmod) %>% as.matrix
pY_rf = predict(pY_rf.fit, newdata = Xmod) %>% as.matrix

# random forest, expanded data
pW_rf.fit.int = regression_forest(Xmod.int, Wmod, num.trees = 500)
pY_rf.fit.int = regression_forest(Xmod.int, Ymod, num.trees = 500)

# pW_rf.int = pW_rf.fit.int$predictions
# pY_rf.int = pY_rf.fit.int$predictions
pW_rf.int = predict(pW_rf.fit.int, newdata = Xmod.int) %>% as.matrix
pY_rf.int = predict(pY_rf.fit.int, newdata = Xmod.int) %>% as.matrix

# CF
cf = causal_forest(Xmod, Ymod, Wmod, num.trees = 500)
cf.int = causal_forest(Xmod.int, Ymod, Wmod, num.trees = 500)


# hist(pW_rf)
#### BIAS FUNCTION ####
```

Next we plot the bias function $b(X)$ following Athey, Imbens, Pham and Wager (AER P&P, 2017, Section IIID). We plot $b(x)$ for all units in the sample, and see that the bias seems evenly distributed around zero. We see that bias for most observations is close to zero.

```r
mu_avg <- function(treated, df){df[W==treated, mean(Y)]}
mu <- function(treated, df){df[W==treated, mean(pY)]}

B <- function(df, treatment_model, outcome_model, outcome_type = "response"){
  # have to supply models so that can estimate counterfactual predictions given an alternative treatmen
  # note that this will NOT work for lasso model, attempt to warn of this misbehavior:
  if (grepl("lasso|rf|cf", deparse(quote(treatment_model)))){
    simpleMessage("The predict method appears to be broken for lasso models estimated using glmnet::cv.
                  You may want to try another predictive model.")
  }
  df = copy(df)
  p = df[,mean(W)]
  mu0 <- df[W==0,mean(Y)]
  mu1 <- df[W==1,mean(Y)]

  pY_w0 <- predict(outcome_model, newdata = df[,.SD, .SDcols = !c('W', 'Y')][, W := 0], type = outcome_
  pY_w1 <- predict(outcome_model, newdata = df[,.SD, .SDcols = !c('W', 'Y')][, W := 1], type = outcome_
  pW <- predict(treatment_model, newdata = df[,.SD, .SDcols = !c('W', 'Y')], type = "response")
  df[, `:=`(W = NULL, pY_w0 = pY_w0, pY_w1 = pY_w1, pW = pW)]


  df[, b := (pW - p) * (p * (pY_w0 - mu0) + (1 - p) * (pY_w1 - mu1))]

  return(df[,.(b)])
  }

df_mod_bias <- B(df_mod, pW_logistic.fit, pY_logistic.fit)
ggplot(df_mod_bias, aes(x = b)) + geom_histogram() + labs(title = "Histogram of per-observation b(x)")
```

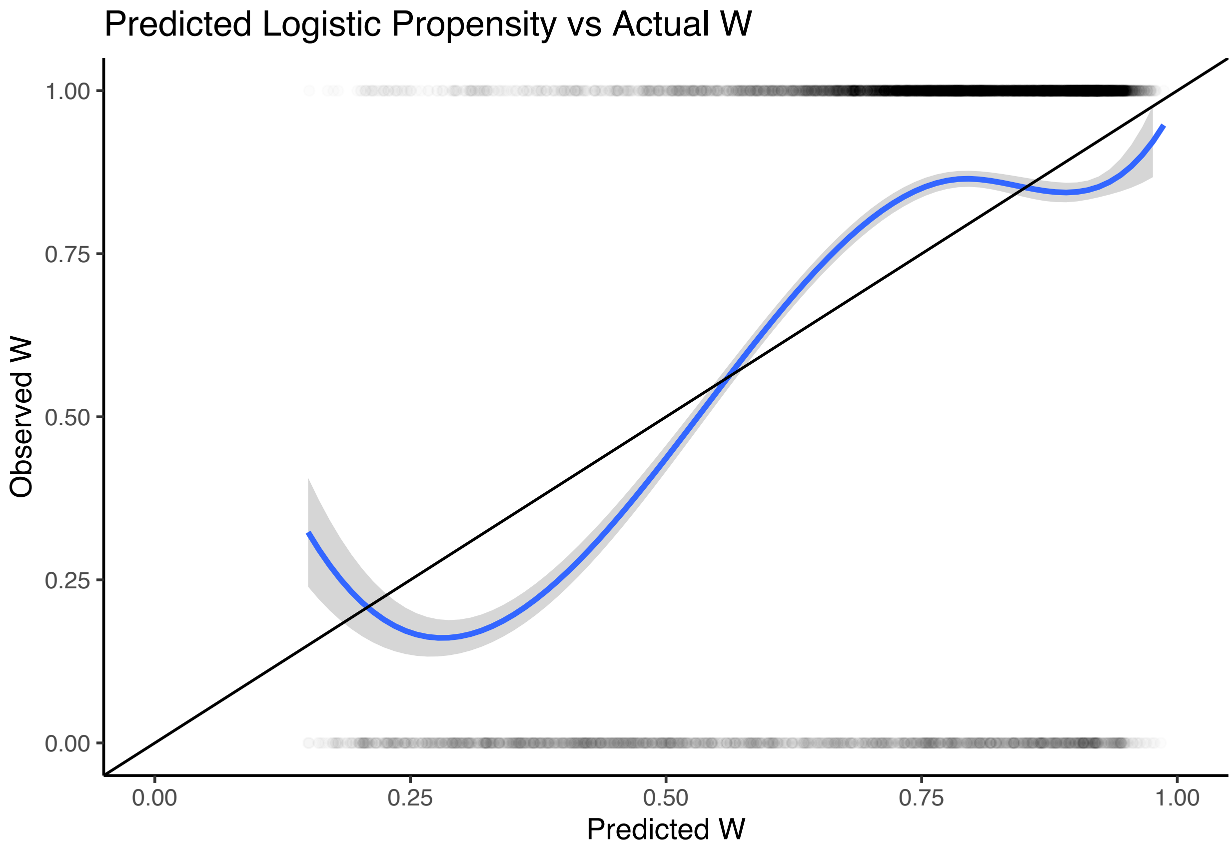


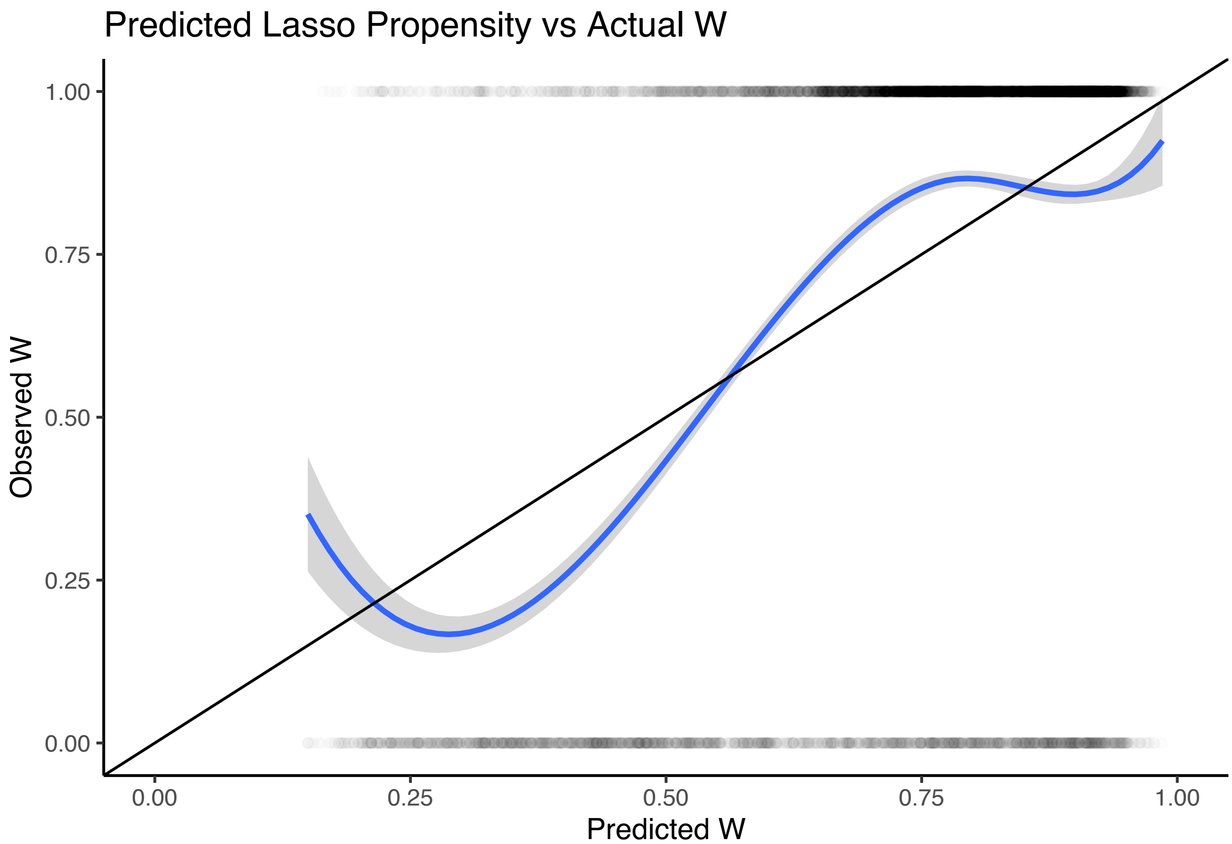

```
#### ESTIMATING ATE INTRO ####
```

Estimating the ATE

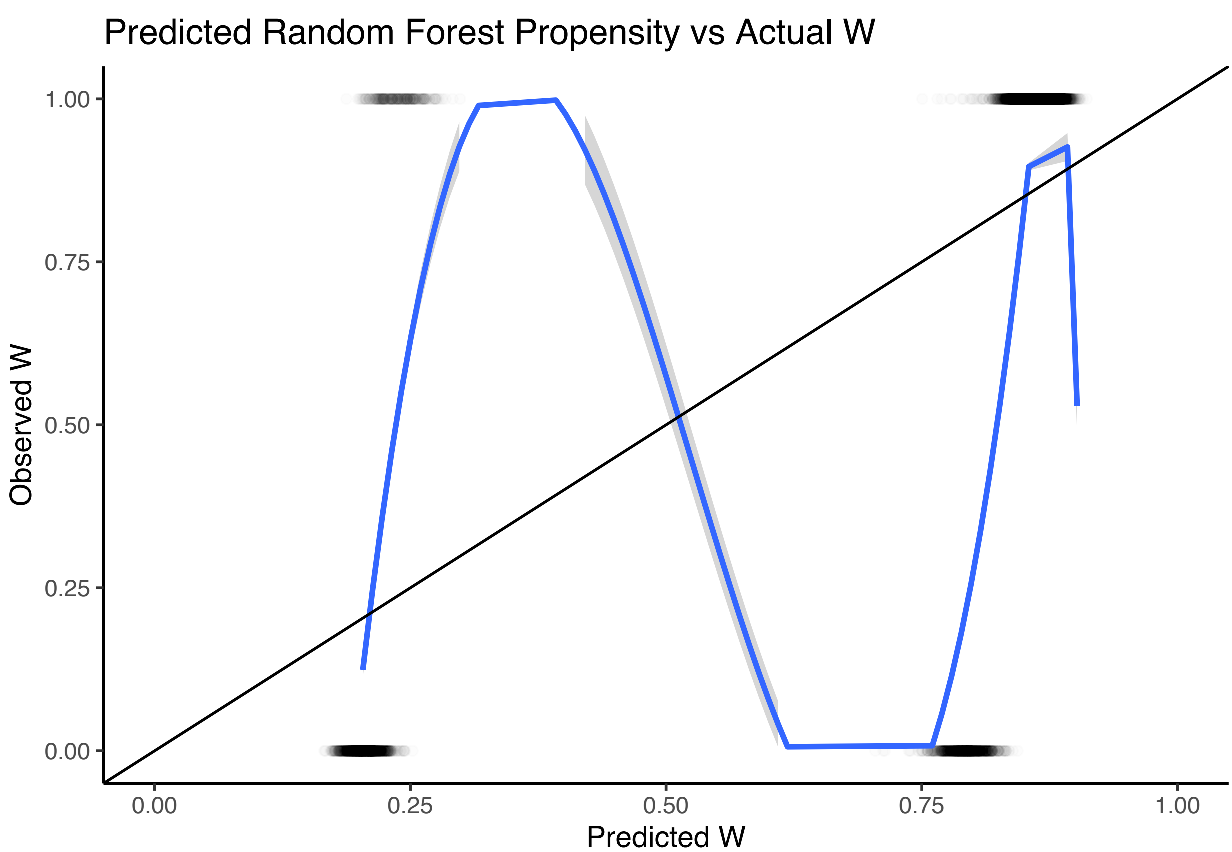
In this section we explore various methods for estimating the ATE. We explore the following methods:
1. inverse propensity weighting via logistic regression
2. direct regression analysis via OLS
3. traditional double robust analysis via augmented inverse-propensity score weighting that combines the above two estimators.
We also re-run the above methods after expanding the data to include all interactions of all of the covariates, and re-estimate outcome and proensity models using the original linear model, as well as running lasso and random forest models on the expanded data.

We first plot propensity scores against treatment status on the original and expanded set of coefficients. Models closer to the 45-degree line are better.
We see that logistic propensity scores perform surprisingly well!

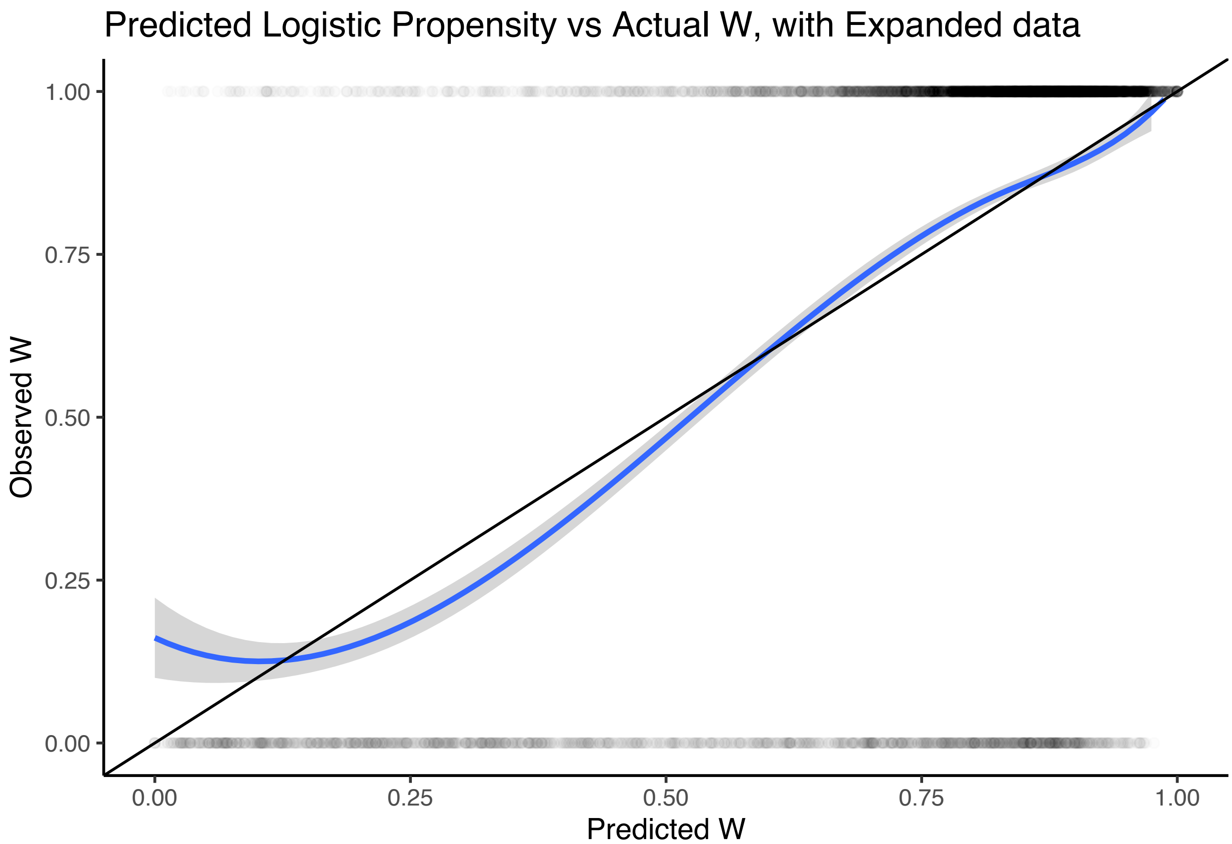


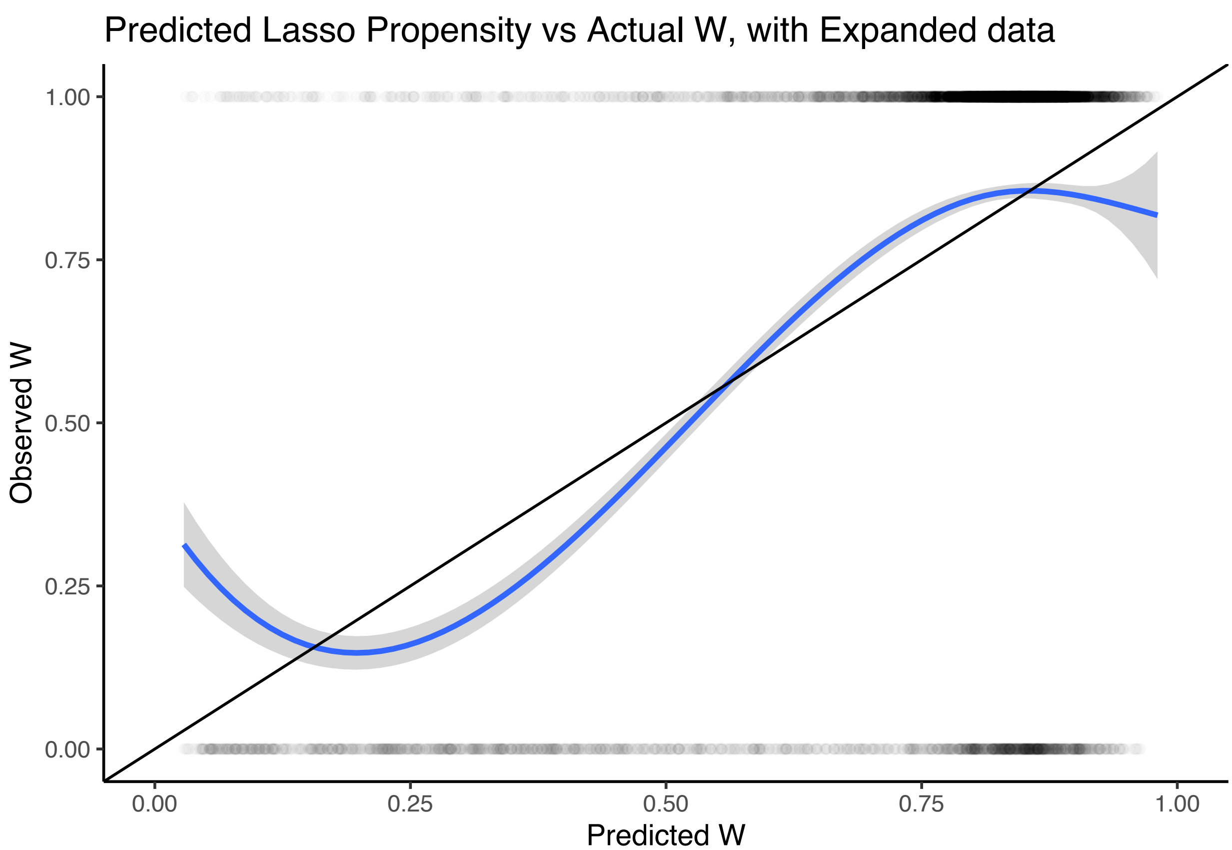


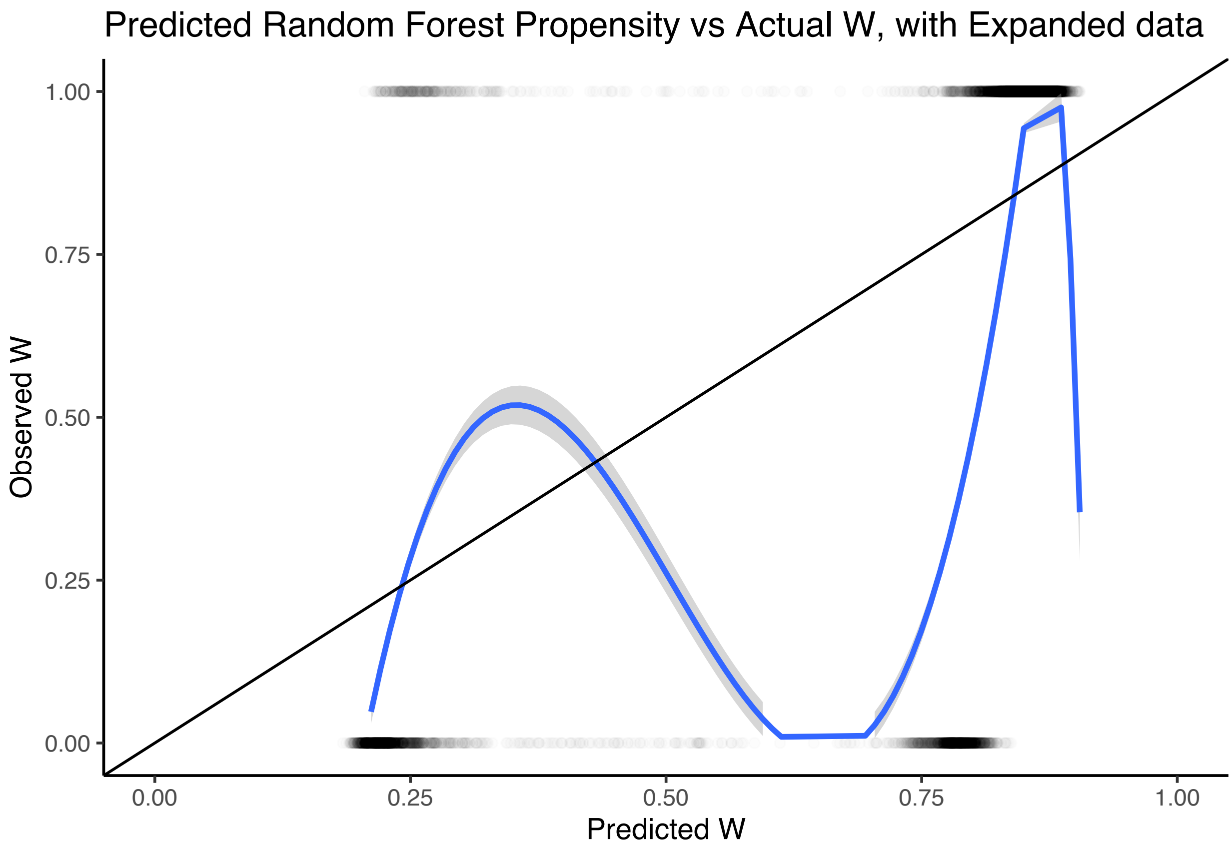
Predicted Lasso Propensity vs Actual W
Observed W
Predicted W
0.00
0.25
0.50
0.75
1.00



Predicted Random Forest Propensity vs Actual W
Observed W
Predicted W
0.00
0.25
0.50
0.75
1.00


Predicted Logistic Propensity vs Actual W, with Expanded data
Observed W
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Predicted W

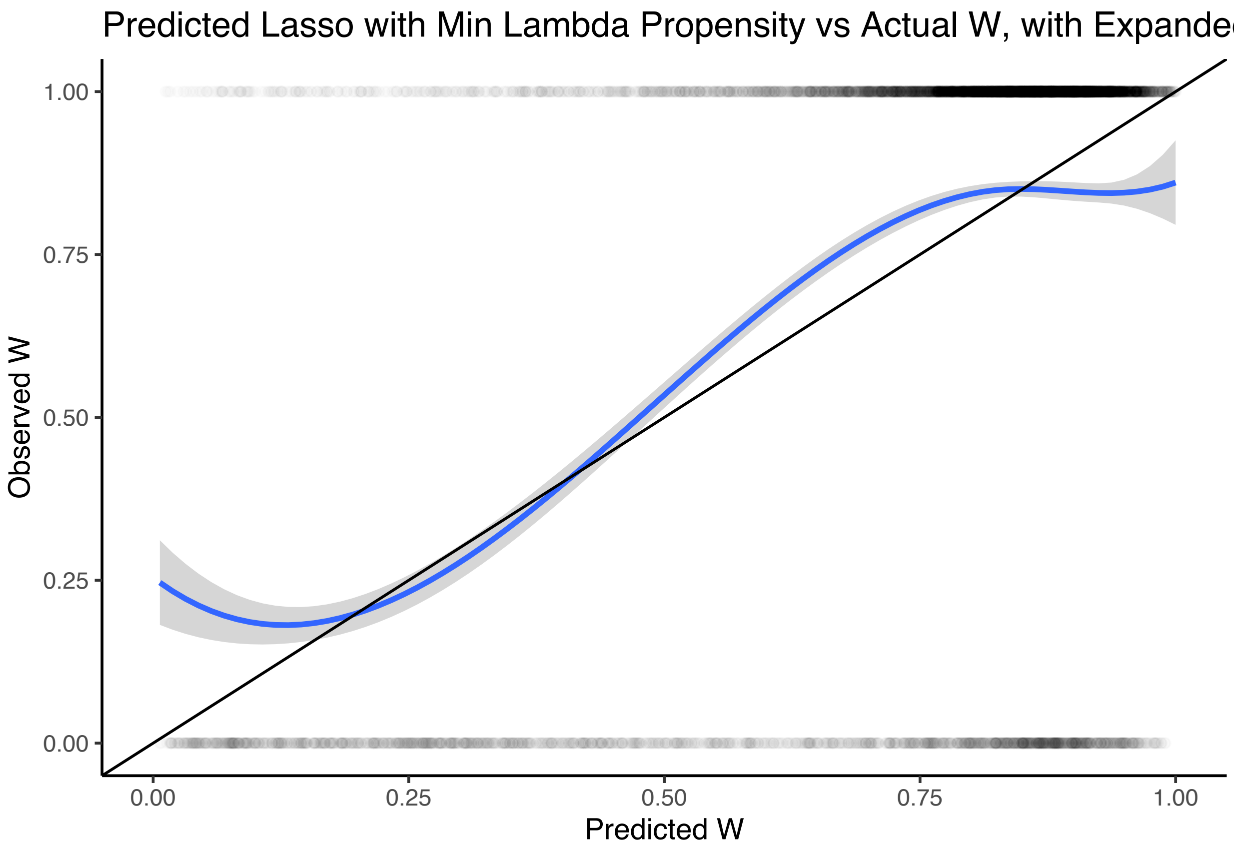

Predicted Lasso Propensity vs Actual W, with Expanded data
Observed W
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Predicted W


Predicted Random Forest Propensity vs Actual W, with Expanded data
Observed W
Predicted W
0.00
0.25
0.50
0.75
1.00
0.00
0.25
0.50
0.75
1.00

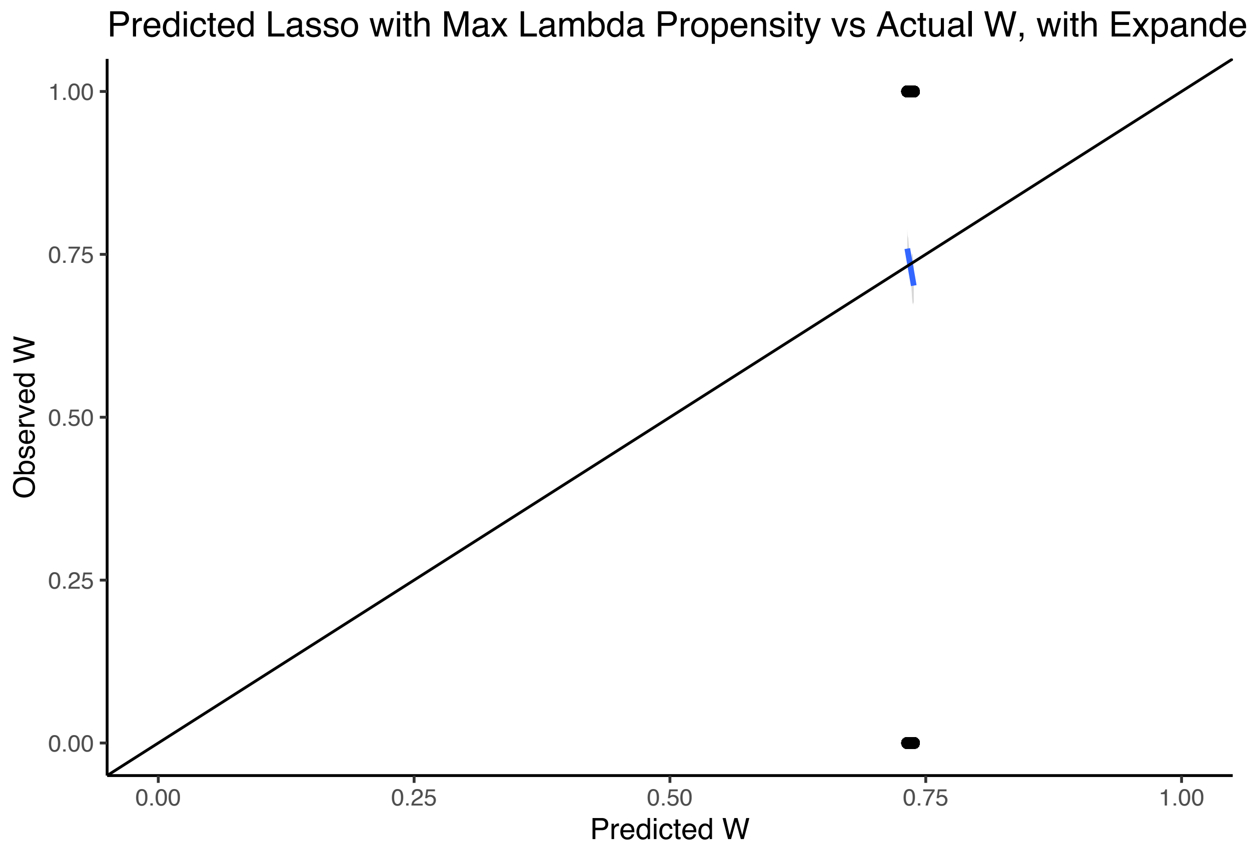
Exploring the Lasso Model Along Lambda

To show how cross-validating lambda is important for the lasso, we compare predicted and actual treatment status for the minimum, maximum, a randomly selected lambda. The lasso with the best lamda is the one closest to the 45-degree line.

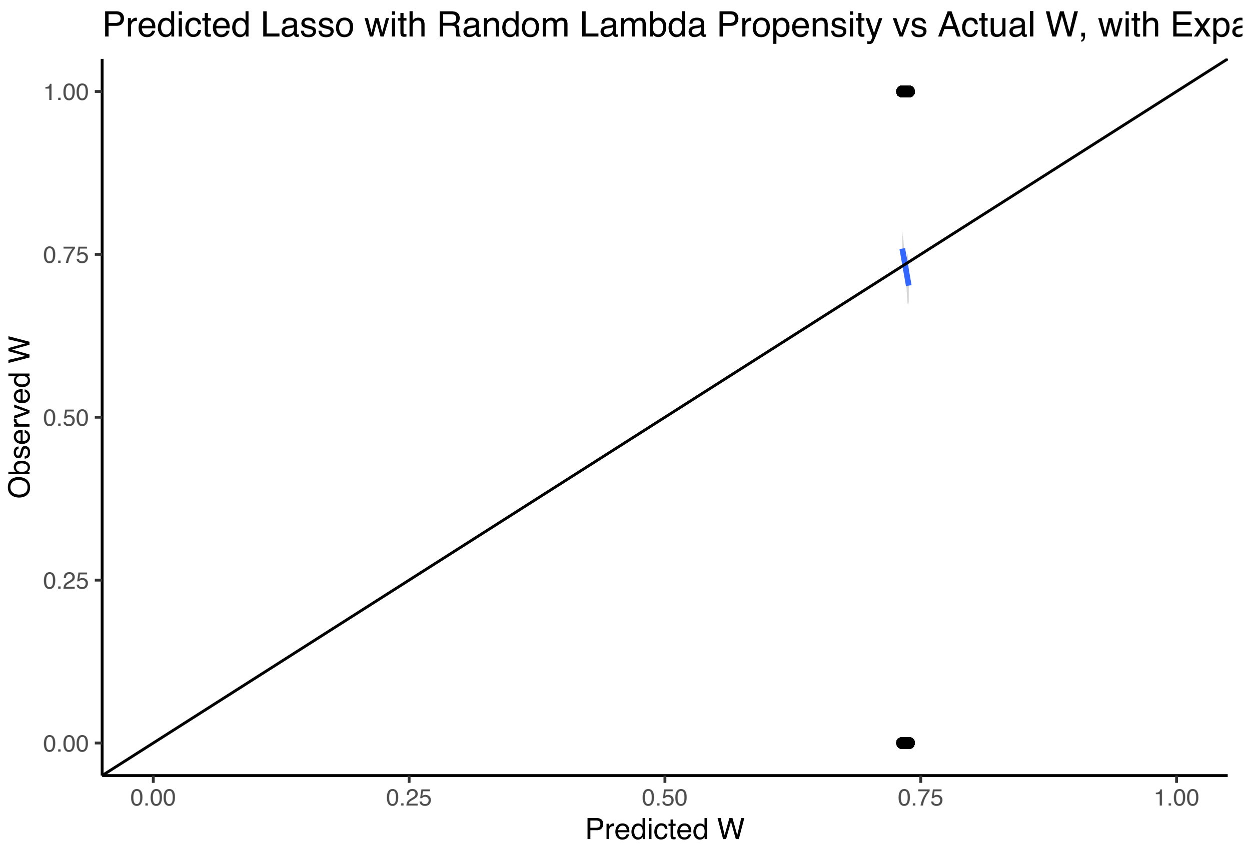
```
plot_prob(pW_lasso.int.min, Wmod, "Lasso with Min Lambda", "Expanded")
```




```
plot_prob(pW_lasso.int.max, Wmod, "Lasso with Max Lambda", "Expanded")
```



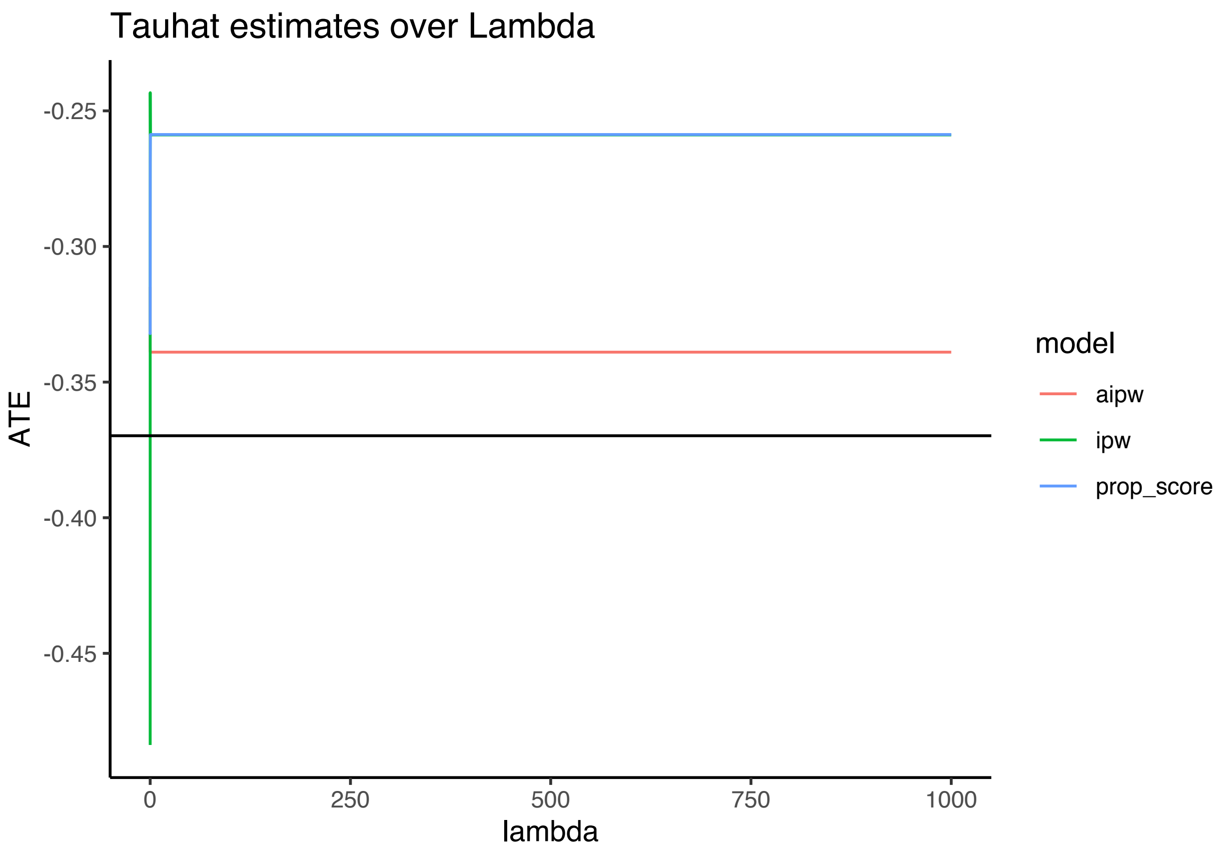

```r
plot_prob(pW_lasso.int.rand, Wmod, "Lasso with Random Lambda", "Expanded")
```



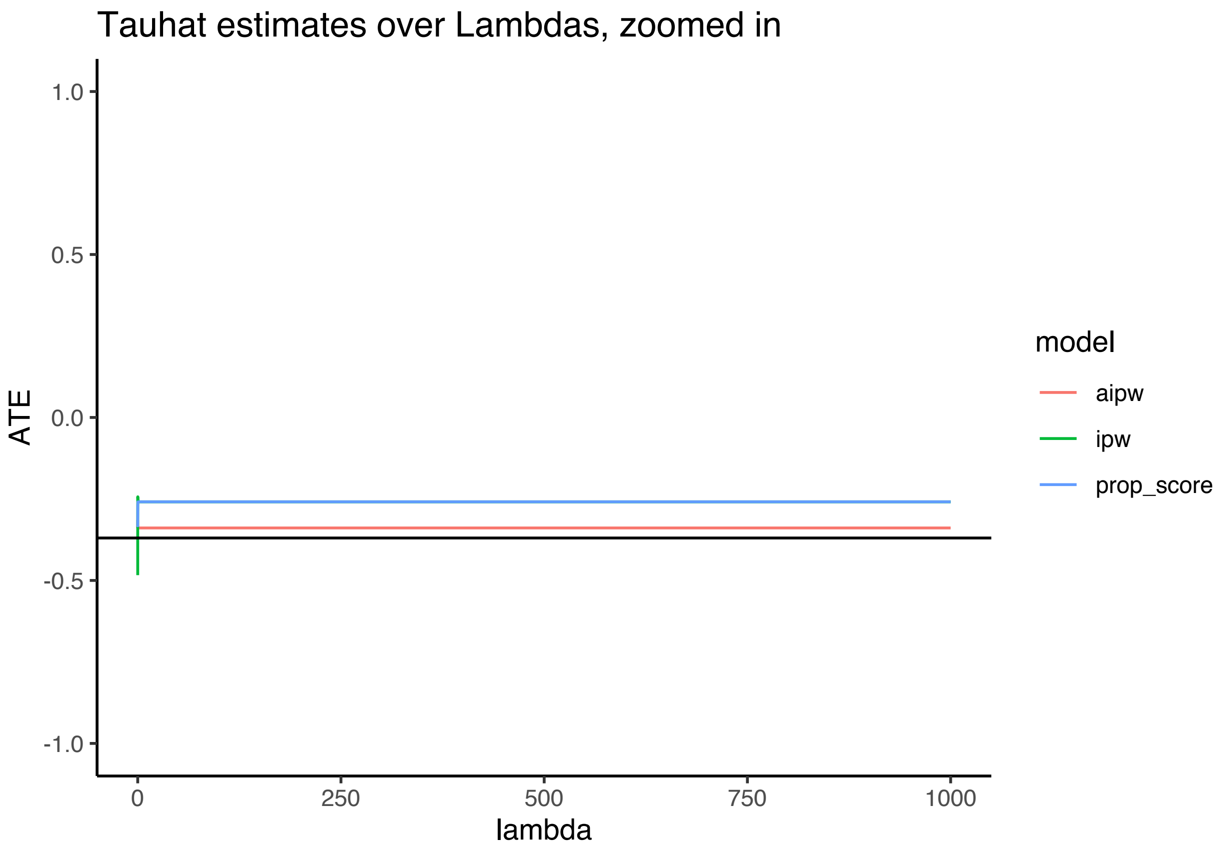
We also plot our various estimates of $\hat{\tau}$ over our lambdas.

```
# plot lasso over grid of lambdas
pW_glmnet.fit.model.int.lambda_preds <- as.data.table(pW_glmnet.fit.model.int$fit.preval)
pW_glmnet.fit.model.int.lambda_preds <- pW_glmnet.fit.model.int.lambda_preds[
  # see discussion in FIXME above about using convert_to_prob here
  ,lapply(.SD, convert_to_prob), .SDcols = names(pW_glmnet.fit.model.int.lambda_preds)]

# credit to Kaleb for this formulation
 tauhat_lasso_ipw.lambdas <- rbindlist(lapply(1:ncol(pW_glmnet.fit.model.int.lambda_preds),
                                              function(p){
  data.frame(lambda=pW_glmnet.fit.model.int$lambda[p],
             "ATE"=ipw(df_mod.int, as.matrix(pW_glmnet.fit.model.int.lambda_preds[,..p]))["ATE"])
}))[, model := "ipw"]

 tauhat_lasso_prop_score.lambdas <- rbindlist(lapply(1:ncol(pW_glmnet.fit.model.int.lambda_preds),
                                                     function(p){
   data.frame(lambda=pW_glmnet.fit.model.int$lambda[p],
              "ATE"=prop_score_ols(df_mod.int, as.matrix(pW_glmnet.fit.model.int.lambda_preds[,..p]))["A
 }))[, model := "prop_score"]

 tauhat_lasso_aipw.lambdas <- rbindlist(lapply(1:ncol(pW_glmnet.fit.model.int.lambda_preds),
                                               function(p){
   data.frame(lambda=pW_glmnet.fit.model.int$lambda[p],
              "ATE"=aipw_ols(df_mod.int, as.matrix(pW_glmnet.fit.model.int.lambda_preds[,..p]))["ATE"])
 }))[, model := "aipw"]

tauhat_lasso_estimates.lambdas <- rbindlist(list(tauhat_lasso_ipw.lambdas,
```

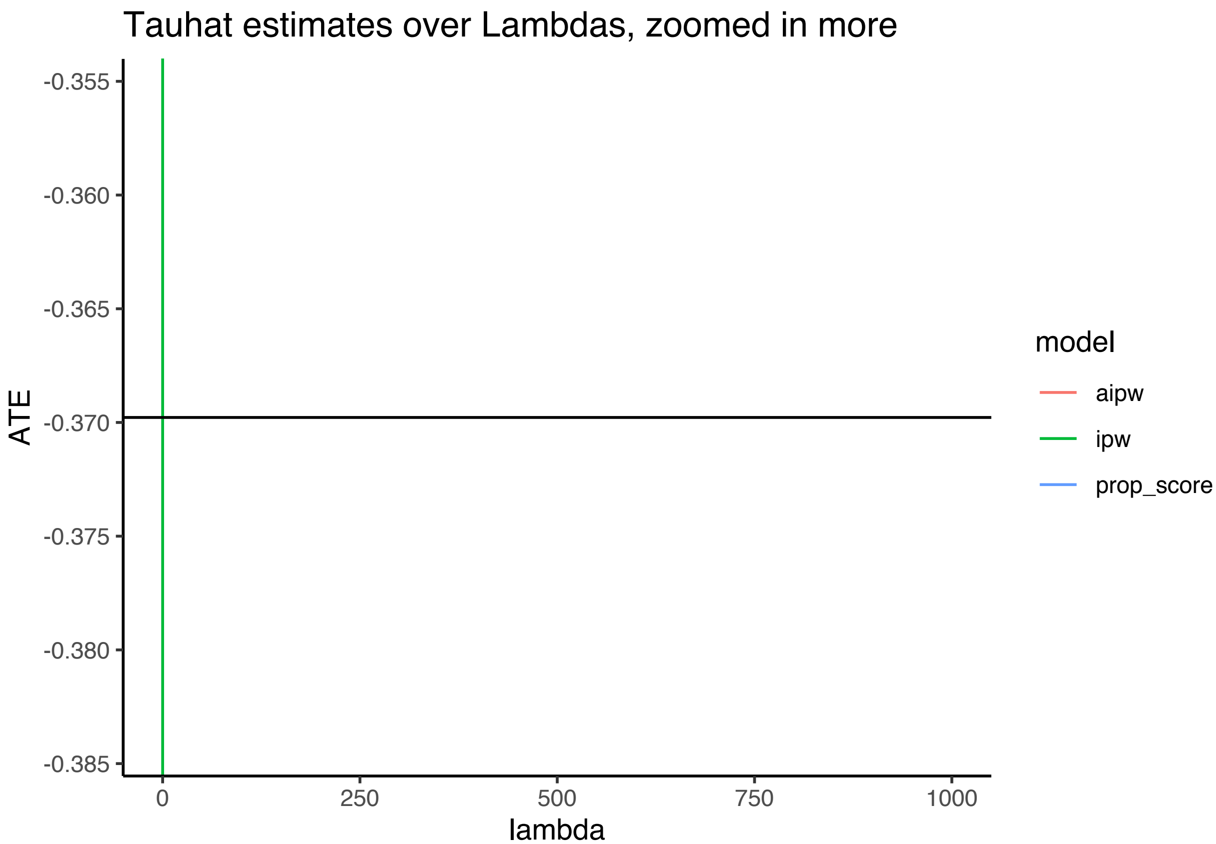
```r
                                                    tauhat_lasso_prop_score.lambdas,
                                                    tauhat_lasso_aipw.lambdas))

ggplot(tauhat_lasso_estimates.lambdas, aes(x = lambda, y = ATE, color = model)) +
  geom_line() +
  geom_abline(aes(slope = 0, intercept = tauhat_rct["ATE"])) +
  ggtitle("Tauhat estimates over Lambda")
```




```r
ggplot(tauhat_lasso_estimates.lambdas, aes(x = lambda, y = ATE, color = model)) +
  geom_line() +
  geom_abline(aes(slope = 0, intercept = tauhat_rct["ATE"])) +
  coord_cartesian(ylim = c(-1, 1)) +
  ggtitle("Tauhat estimates over Lambdas, zoomed in")
```

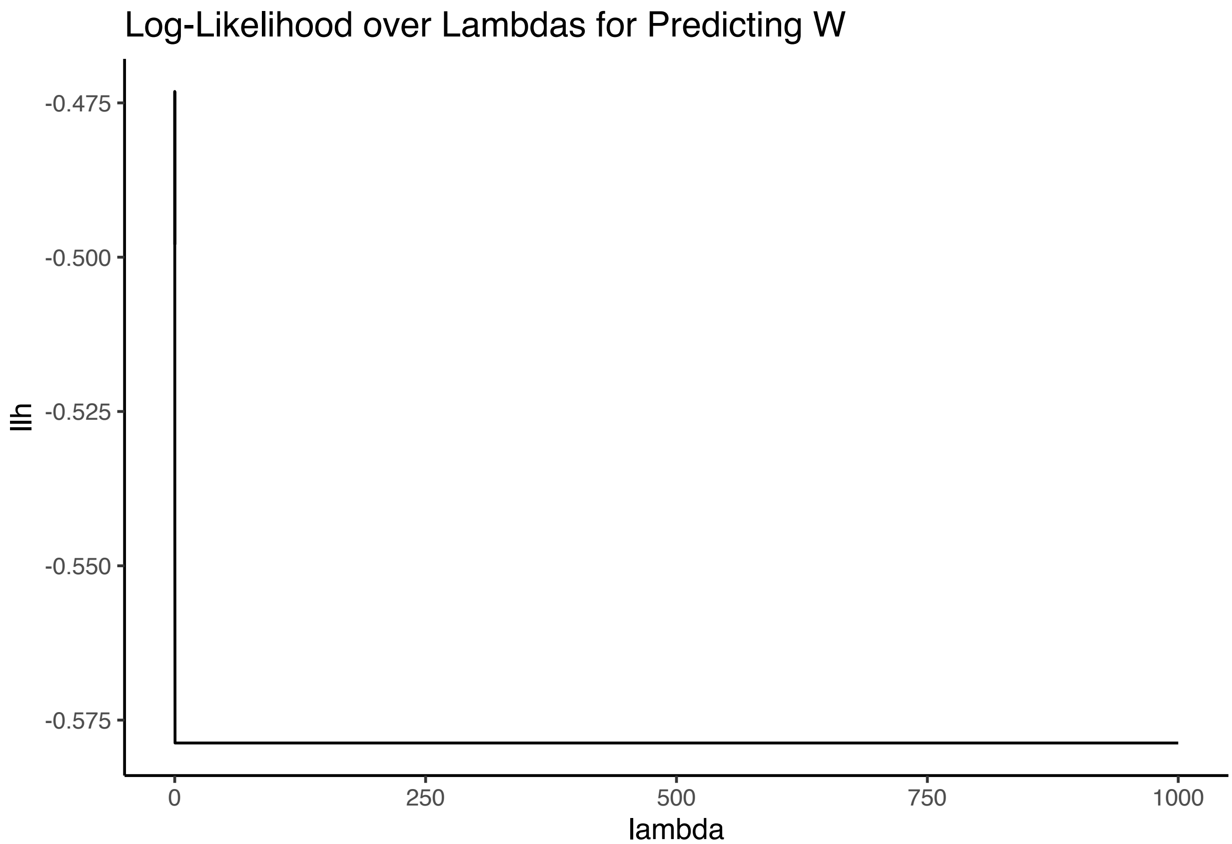



```r
ggplot(tauhat_lasso_estimates.lambdas, aes(x = lambda, y = ATE, color = model)) +
  geom_line() +
  geom_abline(aes(slope = 0, intercept = tauhat_rct["ATE"])) +
  coord_cartesian(ylim = c(tauhat_rct["lower_ci"], tauhat_rct["upper_ci"])) +
  ggtitle("Tauhat estimates over Lambdas, zoomed in more")
```

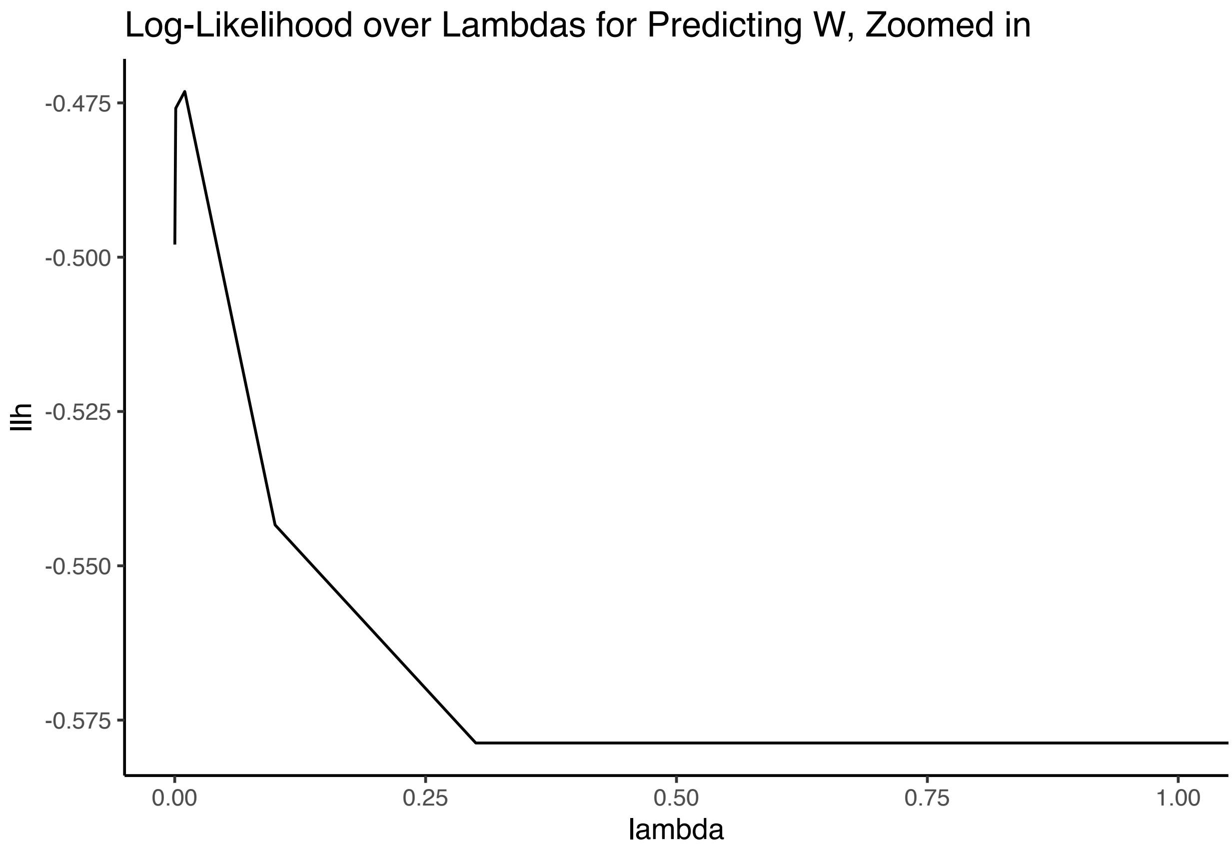


We now plot log likelihood over different values of lambda on the expanded data.

```
lambda_log_liks <- pW_glmnet.fit.model.int.lambda_preds[
  ,lapply(.SD, loglike, Wmod), .SDcols = names(pW_glmnet.fit.model.int.lambda_preds)]

colnames(lambda_log_liks) <- as.character(pW_glmnet.fit.model.int$lambda)
lambda_log_liks.long <- melt(lambda_log_liks, variable.name = "lambda", value.name = "llh")
lambda_log_liks.long[, lambda := as.numeric(as.character(lambda))]

ggplot(lambda_log_liks.long, aes(x = lambda, y = llh)) +
  geom_line() +
  labs(title = "Log-Likelihood over Lambdas for Predicting W")
```




```r
ggplot(lambda_log_liks.long, aes(x = lambda, y = llh)) +
  geom_line() +
  labs(title = "Log-Likelihood over Lambdas for Predicting W, Zoomed in") +
  coord_cartesian(x = c(0,1))
```




```
#### EXPLORING RF ####
```

Exploring Random Forest Performance with Sample Size

To show how sample size is is important for random forest performance, we compare ATE estimates across sample size.

```
ate_rf_aipw.int = average_treatment_effect(cf.int)
tauhat_rf_aipw.int = c(ATE=ate_rf_aipw.int["estimate"],
                       lower_ci=ate_rf_aipw.int["estimate"] - 1.96 * ate_rf_aipw.int["std.err"],
                       upper_ci=ate_rf_aipw.int["estimate"] + 1.96 * ate_rf_aipw.int["std.err"])
tauhat_rf_ipw.int = ipw(df_mod.int, pW_rf.int)
tauhat_ols_rf_aipw.int = aipw_ols(df_mod.int, pW_rf.int)

tauhat_rf_ipw.int
```

```
##      ATE lower_ci upper_ci
##   -0.217   -0.243   -0.191
```

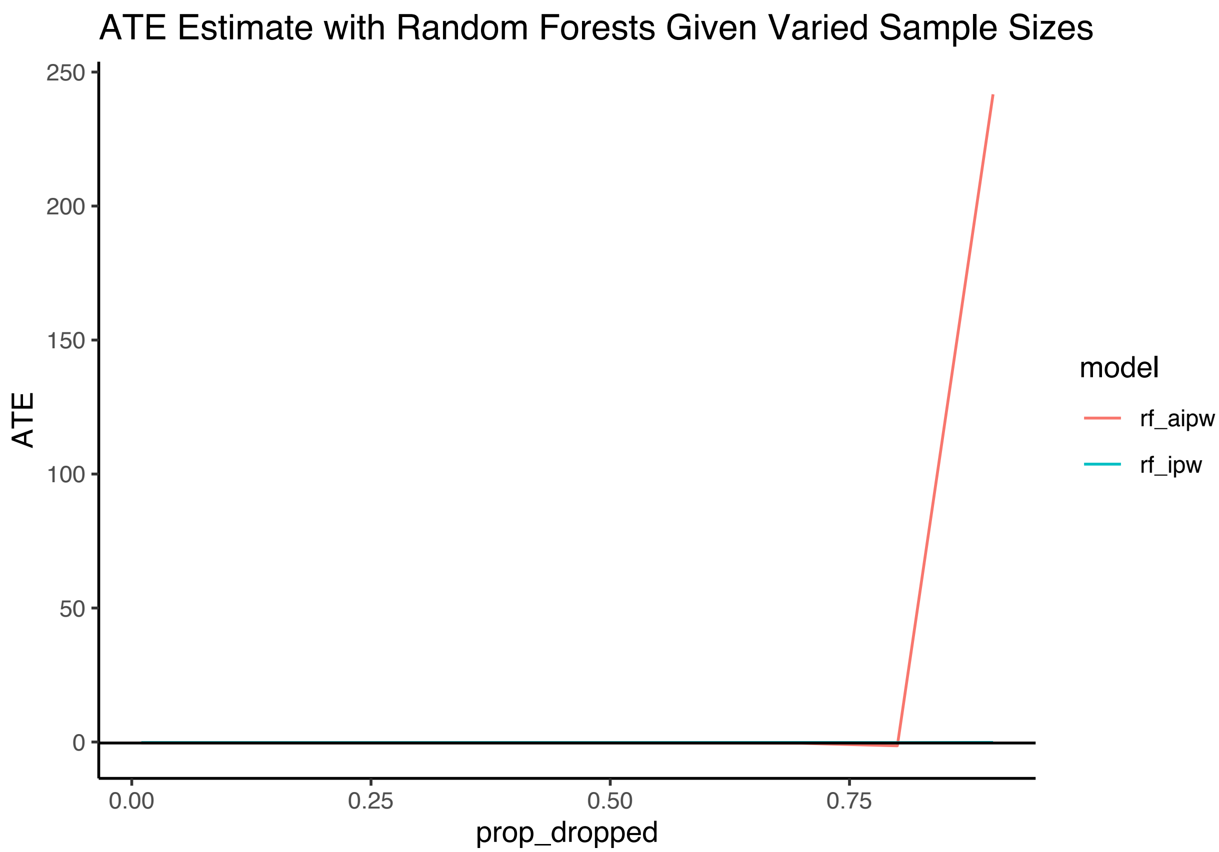
```
tauhat_ols_rf_aipw.int
```

```
##      ATE lower_ci upper_ci
##   -0.339   -0.358   -0.320
```

We now repeat this analysis over different sample sizes.

```
tauhat_rf_list <- data.frame()
tauhat_ols_rf_aipw_list <- data.frame()
for (prob_temp in prop_drop_rf) {
  # drop same proportion of treated and control units
  drop_from_treat_temp <- base::sample(which(df_mod$W == 1), round(prob_temp * sum(df_mod$W == 1)))
  drop_from_control_temp <- base::sample(which(df_mod$W == 0), round(prob_temp * sum(df_mod$W == 0)))
  df_mod_temp <- df_mod[-c(drop_from_treat_temp, drop_from_control_temp),]
  # df_mod_temp <- copy(df_mod)
  Xmod_temp = df_mod_temp[,.SD, .SDcols = names(df_mod_temp)[!names(df_mod_temp) %in% c("Y", "W")]] %>%
  Ymod_temp = df_mod_temp$Y
  Wmod_temp = df_mod_temp$W
  XWmod_temp = cbind(Xmod_temp, Wmod_temp)
  pW_rf_temp.fit = regression_forest(Xmod_temp, Wmod_temp, num.trees = 500)
  # pY_rf_mod.fit = regression_forest(Xmod_temp, Ymod_temp, num.trees = 500)
  Xmod_temp.int = model.matrix(~ . * ., data = as.data.frame(Xmod_temp))
  #XWmod_temp.int = cbind(Xmod_temp.int, Wmod_temp)
  df_mod_temp.int <- Xmod_temp.int %>% as.data.frame %>% setDT()
  df_mod_temp.int[,`:=`(W = Wmod_temp, Y = Ymod_temp)]
  pW_rf_temp.int.fit = regression_forest(Xmod_temp.int, Wmod_temp, num.trees = 500)
  # pY_rf_temp.int.fit = regression_forest(Xmod_temp.int, Ymod_temp, num.trees = 500)
  pW_rf_temp.int = predict(pW_rf_temp.int.fit, newdata = Xmod_temp.int) %>% as.matrix
  # pY_rf_temp.int = predict(pY_rf.fit.int, newdata = Xmod_temp.int)

  # pW_rf = pW_rf.fit$predictions

  tauhat_rf_temp.int = ipw(df_mod_temp.int, pW_rf_temp.int) %>% as.list() %>% data.frame()
  tauhat_rf_temp.int$"prop_dropped" <- prob_temp
  tauhat_rf_temp.int$model <- "rf_ipw"
  tauhat_ols_rf_aipw_temp.int = aipw_ols(df_mod_temp.int, pW_rf_temp.int) %>% as.list() %>% data.frame(
  tauhat_ols_rf_aipw_temp.int$"prop_dropped" <- prob_temp
  tauhat_ols_rf_aipw_temp.int$model <- "rf_aipw"
```

```r
  # print(tauhat_rf_temp.int)
  # print(tauhat_ols_rf_aipw_temp.int)
  tauhat_rf_list <- rbind(tauhat_rf_list, tauhat_rf_temp.int)
  tauhat_ols_rf_aipw_list <- rbind(tauhat_ols_rf_aipw_list, tauhat_ols_rf_aipw_temp.int)
}

tauhat_rf_list <- rbind(tauhat_rf_list, tauhat_ols_rf_aipw_list)

ggplot(tauhat_rf_list, aes(x = prop_dropped, y = ATE, color = model)) + geom_line() +
  ggtitle("ATE Estimate with Random Forests Given Varied Sample Sizes") +
  geom_abline(aes(slope = 0, intercept = tauhat_rct["ATE"]))
```




```r
#### ATE CALCULATIONS: ORIGINAL DATA ####

# linear models
tauhat_ols <- ate_condmean_ols(df_mod)

# linear models
tauhat_logistic_ipw <- ipw(df_mod, pW_logistic)
tauhat_pscore_ols <- prop_score_ols(df_mod, pW_logistic)
tauhat_lin_logistic_aipw <- aipw_ols(df_mod, pW_logistic)

# lasso
tauhat_lasso_ipw <- ipw(df_mod, pW_lasso)
tauhat_pscore_lasso <- prop_score_ols(df_mod, pW_lasso)
tauhat_lasso_logistic_aipw <- aipw_ols(df_mod, pW_lasso)
# FIXME: add this
```

```r
# prior code to add:
# Xmod.for.lasso = cbind(Wmod, Xmod, (2 * Wmod - 1) * Xmod)
# glmnet.fit.outcome = amlinear:::crossfit.cv.glmnet(Xmod.for.lasso, Ymod,
#                                                      penalty.factor = c(0, rep(1, ncol(Xmod.for.lasso)
# lasso.yhat.control = amlinear:::crossfit.predict(glmnet.fit.outcome,
#                                                    cbind(0, Xmod, -Xmod))
# lasso.yhat.treated = amlinear:::crossfit.predict(glmnet.fit.outcome,
#                                                    cbind(1, Xmod, Xmod))
# The lasso AIPW estimator. Here, the inference is justified via
# orthogonal moments.
# G = lasso.yhat.treated - lasso.yhat.control +
#   Wmod / pW_lasso * (Ymod - lasso.yhat.treated) -
#   (1 - Wmod) / (1 - pW_lasso) * (Ymod - lasso.yhat.control)
# tau.hat = mean(G)
# se.hat = sqrt(var(G) / length(G))
# tauhat_lasso_aipw = c(ATE=tau.hat,
#                        lower_ci=tau.hat-1.96*se.hat,
#                        upper_ci=tau.hat+1.96*se.hat)

# FIXME: add this
# balancing.weights = amlinear::balance_minimax(Xmod, Wmod, zeta = 0.5)
# G.balance = lasso.yhat.treated - lasso.yhat.control +
#   balancing.weights * (Ymod - Wmod * lasso.yhat.treated
#                         - (1 - Wmod) * lasso.yhat.control)
# tau.hat = mean(G.balance)
# se.hat = sqrt(var(G.balance) / length(G.balance))
# tauhat_lasso_balance = c(ATE=tau.hat,
#                           lower_ci=tau.hat-1.96*se.hat,
#                           upper_ci=tau.hat+1.96*se.hat)

# RF
tauhat_rf_ipw = ipw(df_mod, pW_rf)
ate_rf_aipw = average_treatment_effect(cf)
tauhat_rf_aipw = c(ATE=ate_rf_aipw["estimate"],
                   lower_ci=ate_rf_aipw["estimate"] - 1.96 * ate_rf_aipw["std.err"],
                   upper_ci=ate_rf_aipw["estimate"] + 1.96 * ate_rf_aipw["std.err"])
tauhat_ols_rf_aipw = aipw_ols(df_mod, pW_rf)

#### ATE CALCULATIONS: INTERACTED DATA ####

# linear models
tauhat_ols.int <- ate_condmean_ols(df_mod.int)

# linear models
tauhat_logistic_ipw.int <- ipw(df_mod.int, pW_logistic.int)
tauhat_pscore_ols.int <- prop_score_ols(df_mod.int, pW_logistic.int)
tauhat_lin_logistic_aipw.int <- aipw_ols(df_mod.int, pW_logistic.int)

# lasso
tauhat_lasso_ipw.int <- ipw(df_mod.int, pW_lasso.int)
tauhat_pscore_lasso.int <- prop_score_ols(df_mod.int, pW_lasso.int)
tauhat_lasso_logistic_aipw.int <- aipw_ols(df_mod.int, pW_lasso.int)
# FIXME: add this
```

```
# prior code to add:
# Xmod.for.lasso = cbind(Wmod, Xmod.int, (2 * Wmod - 1) * Xmod.int)
# glmnet.fit.outcome = amlinear:::crossfit.cv.glmnet(Xmod.for.lasso, Ymod,
#                                                     penalty.factor = c(0, rep(1, ncol(Xmod.for.lasso)
# lasso.yhat.control = amlinear:::crossfit.predict(glmnet.fit.outcome,
#                                                   cbind(0, Xmod.int, -Xmod.int))
# lasso.yhat.treated = amlinear:::crossfit.predict(glmnet.fit.outcome,
#                                                   cbind(1, Xmod.int, Xmod.int))
# The lasso AIPW estimator. Here, the inference is justified via
# orthogonal moments.
# G = lasso.yhat.treated - lasso.yhat.control +
#   Wmod / pW_lasso * (Ymod - lasso.yhat.treated) -
#   (1 - Wmod) / (1 - pW_lasso) * (Ymod - lasso.yhat.control)
# tau.hat = mean(G)
# se.hat = sqrt(var(G) / length(G))
# tauhat_lasso_aipw = c(ATE=tau.hat,
#                       lower_ci=tau.hat-1.96*se.hat,
#                       upper_ci=tau.hat+1.96*se.hat)

# FIXME: add this
# balancing.weights = amlinear::balance_minimax(Xmod.int, Wmod, zeta = 0.5)
# G.balance = lasso.yhat.treated - lasso.yhat.control +
#   balancing.weights * (Ymod - Wmod * lasso.yhat.treated
#                        - (1 - Wmod) * lasso.yhat.control)
# tau.hat = mean(G.balance)
# se.hat = sqrt(var(G.balance) / length(G.balance))
# tauhat_lasso_balance = c(ATE=tau.hat,
#                          lower_ci=tau.hat-1.96*se.hat,
#                          upper_ci=tau.hat+1.96*se.hat)

#### COMPARING ATE ACROSS MODELS ####
```

Comparing ATE Across Models with Original Data

Finally, we compare ATE across various models. We see that AIPW forest methods performs the best across the original and interacted data, though propensity weighted regression performed on the original data.

```
##                                 ATE lower_ci upper_ci ci_length
## RCT_gold_standard            -0.370   -0.384   -0.355     0.029
## naive_observational          -0.259   -0.273   -0.245     0.028
## linear_regression            -0.333   -0.359   -0.307     0.052
## propensity_weighted_regression -0.341   -0.373   -0.308     0.065
## IPW_logistic                 -0.405   -0.455   -0.356     0.099
## AIPW_linear_plus_logistic    -0.329   -0.364   -0.294     0.070
## IPW_forest                   -0.227   -0.255   -0.200     0.055
## AIPW_forest                  -0.335   -0.363   -0.307     0.056
## AIPW_linear_plus_forest      -0.337   -0.357   -0.317     0.041
## IPW_lasso                    -0.408   -0.458   -0.358     0.100
```

Comparing ATE Across Models with Interacted Data

```
##                                 ATE lower_ci upper_ci ci_length
```

```
## RCT_gold_standard              -0.370   -0.384   -0.355    0.029
## naive_observational            -0.259   -0.273   -0.245    0.028
## linear_regression              -0.339      NaN      NaN      NaN
## propensity_weighted_regression -0.337   -0.372   -0.302    0.070
## IPW_logistic                   -0.336   -0.383   -0.289    0.094
## AIPW_linear_plus_logistic      -0.318   -0.349   -0.287    0.062
## IPW_forest                     -0.217   -0.243   -0.191    0.053
## AIPW_forest                    -0.342   -0.369   -0.315    0.054
## AIPW_linear_plus_forest        -0.339   -0.358   -0.320    0.038
## IPW_lasso                      -0.365   -0.407   -0.322    0.085
```

Part Two

Justification of Propensity Stratification

Propensity stratification follows the same principle as stratified random experiments, where we would assign treatment after breaking people into groups based on their observable characteristics. This would ensure balance between treated and control units within strata (it would allow us to avoid overlap problems if done correctly). Propensity-based stratification functions similarly, as we are only comparing units with similar observable characteristics. Propensity scores provide a single-dimensional, unified measure with which to compare units. By comparing only "similar" units, we can ensure that our estimate of the treatment effect should be more accurate. By averaging our predictions over these strata, we can reduce the effect of bias present in only parts of the covariate space. When we increase the number of strata (fixing N), we narrow our comparison to more similar units, and reduce the effect of any poorly-estimated strata.

Propensity Stratification Function

Here we present a function to calculate propensity scores at a strata-level. The user supplies either a dataframe with treatment column W and outcome column Y, as well as a model with which to estimate propensity scores (we don't supply a pre-calculated set of propensity scores in case we want to examine the usefulness of propensity stratification for new data). The function takes options for a number of strata and the function to estimate on the strata - the user could supply something more complex than the simple difference-in-means, for example.

The function checks that each strata has both treatment groups; if it does not then it is not included in the ATE calculation.

We fix the number of strata at 10, following the heuristic discussed in the homework.

Note that the true effect is `mean(W * X[,2])`.

```
propensity_stratification <- function(df, treatment_model, n_strata = 10,
                                      tau_estimator = difference_in_means){
  df <- copy(df)
  df[, pW := predict(treatment_model, newdata = df[,.SD, .SDcols = !c('W', 'Y')], type = "response")]
  df[, pW_strata := ntile(pW, n_strata)]
  # if any strata is empty, automatically ignored
  # if all
  strata_count <- df[, .N, by = .(pW_strata, W)]
  strata_to_keep <- strata_count[, .(n_strata_W_nonempty = sum(N > 0)), by = .(pW_strata)][
    # keep strata if has observations in both treatment
    n_strata_W_nonempty == 2, pW_strata] %>% unique
```

```r
  # sloppy but can't figure out right solution atm
  strata_tau_est <- df[pW_strata %in% strata_to_keep, .(
    tauhat = tau_estimator(.SD)[1],
    lower_ci = tau_estimator(.SD)[2],
    upper_ci = tau_estimator(.SD)[3]), by = .(pW_strata)][
      ,.(tauhat = mean(tauhat),
        lower_ci = mean(lower_ci),
        upper_ci = mean(upper_ci))]
  return(c(ATE = strata_tau_est$tauhat,
           lower_ci = strata_tau_est$lower_ci,
           upper_ci = strata_tau_est$upper_ci))
}

#### SIMULATION ####
```

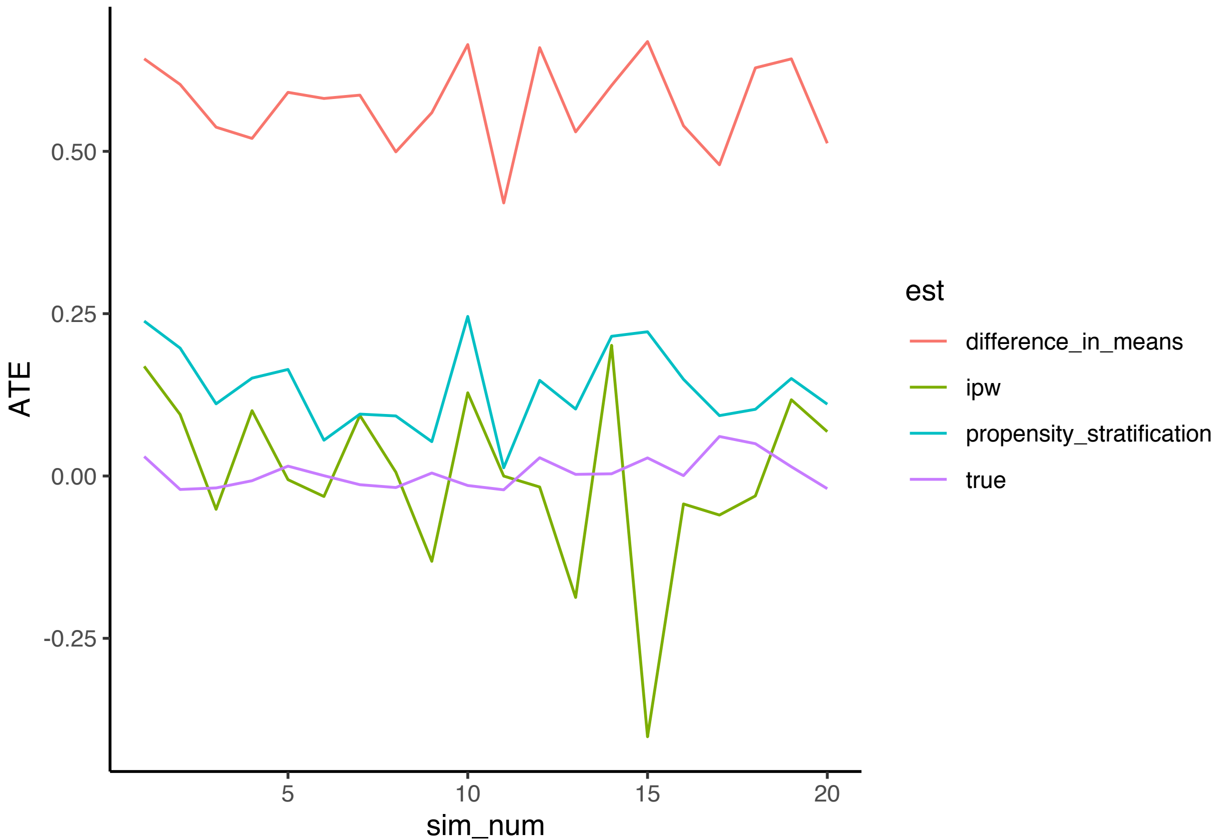
Simulation Exercise

For all discussion that follows, we use the following code to generate a simulated dataset:

```r
make_simulation <- function(){
  n = 1000; p = 20
  X = matrix(rnorm(n * p), n, p)
  propensity = pmax(0.2, pmin(0.8, 0.5 + X[,1]/3))
  W = rbinom(n, 1, propensity)
  Y = pmax(X[,1] + W * X[,2], 0) + rnorm(n)
  df = cbind(X, W, Y) %>% as.data.frame() %>% setDT
  df
}
```

We now run 20.0 simulations of the type described above, and report results over each simulation. We see that propensity stratification and IPW perform similarly, but IPW has lower average MSE.

```r
sim_storage <- data.table()
for (i in 1:n_sims){
  df <- make_simulation()

  sim_pW_logistic.fit <- glm(W ~ ., data = df %>% select(-Y), family = "binomial")
  sim_pW_logistic <- predict(sim_pW_logistic.fit, type = "response")

  tau_ests <- rbind(
    c(df[,mean(W * V2)], NA, NA),
    difference_in_means(df),
    ipw(df, sim_pW_logistic),
    propensity_stratification(df, sim_pW_logistic.fit)
  ) %>% as.data.frame()

  tau_ests$sim_num <- i
  tau_ests$est <- c("true", "difference_in_means", "ipw", "propensity_stratification")
  sim_storage <- rbindlist(list(sim_storage, tau_ests), fill = TRUE)
}

ggplot(sim_storage, aes(x = sim_num, y = ATE, color = est)) + geom_line()
```




```
##### COPY OF OTHER FUNCTIONS ####
```

APPENDIX: Additional Functions Used

In this section, we include a number of other functions necessary to run this code. We do not include any functions defined in any of the tutorials.

```
plot_prob <- function(prob, pred, model_name = "", data_name = ""){
  model = deparse(quote(Wmod)) %>% substr(0,1)
  stopifnot(model %in% c("W", "Y"))
  data_text = ifelse(data_name=="","", sprintf(", with %s data", data_name))
  ggplot(data.frame(prob, pred), aes(x = prob, y = pred)) +
    geom_point(alpha = .01) +
    # geom_smooth() +
    geom_smooth(method = lm, formula = y ~ splines::bs(x, 4), se = TRUE) +
    geom_abline(intercept = 0, slope = 1) +
    labs(title = sprintf("Predicted %s Propensity vs Actual %s%s", model_name, model, data_text),
         x = sprintf("Predicted %s", model),
         y = sprintf("Observed %s", model)) +
    xlim(0,1) +
    ylim(0,1)
}

convert_to_prob <- function(x){
  1/(1 + exp(-x))
}
```

```r
loglike <- function(pred, data){
  mean(data * log(pred) + (1 - data) * log(1 - pred))
}
```