Problem Set 1

Mitchell Linegar

2020-04-30

Contents


```
#### SETUP ####
# 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
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_to_keep <- 1.0 # if you want to only run on a random sample of the data, if want to run on full da

prop_drop_rf <- c(0.01, 0.02, 0.04, 0.1, 0.2, 0.3, 0.4, .5)

library(here)
# devtools::install_github("hrbrmstr/hrbrthemes")
# library(hrbrthemes)
library(ggplot2)
```

```r
theme_set(theme_classic())
library(data.table)
library(tidyverse)
library(broom)
library(grf)
library(sandwich)
devtools::install_github("swager/amlinear") # install amlinear package
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.

Collaborators

I worked closely on this problem set with Mitchell Linegar, who is my team mate for the course. We also worked with Kaleb Javier Pena and Haviland Sheldahl-Thomason, and indicate where we collaborated on code.

```r
#### DATA WORK ####
# follows tutorial exactly
```

Pre-Processing the Data

We 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.

```r
all_variables_names <- c(outcome_variable_name, treatment_variable_name, covariate_names)
df <- df[, which(names(df) %in% all_variables_names)]
# Drop rows containing missing values
df <- na.omit(df)
# Rename variables
names(df)[names(df) == outcome_variable_name] <- "Y"
names(df)[names(df) == treatment_variable_name] <- "W"
# Converting all columns to numerical
df <- data.frame(lapply(df, function(x) as.numeric(as.character(x))))

# coerce to data.table for future analysis
setDT(df)

# if option supplied to randomly subset, do so
df <- df[base::sample(1:nrow(df), prop_to_keep * nrow(df))]
```

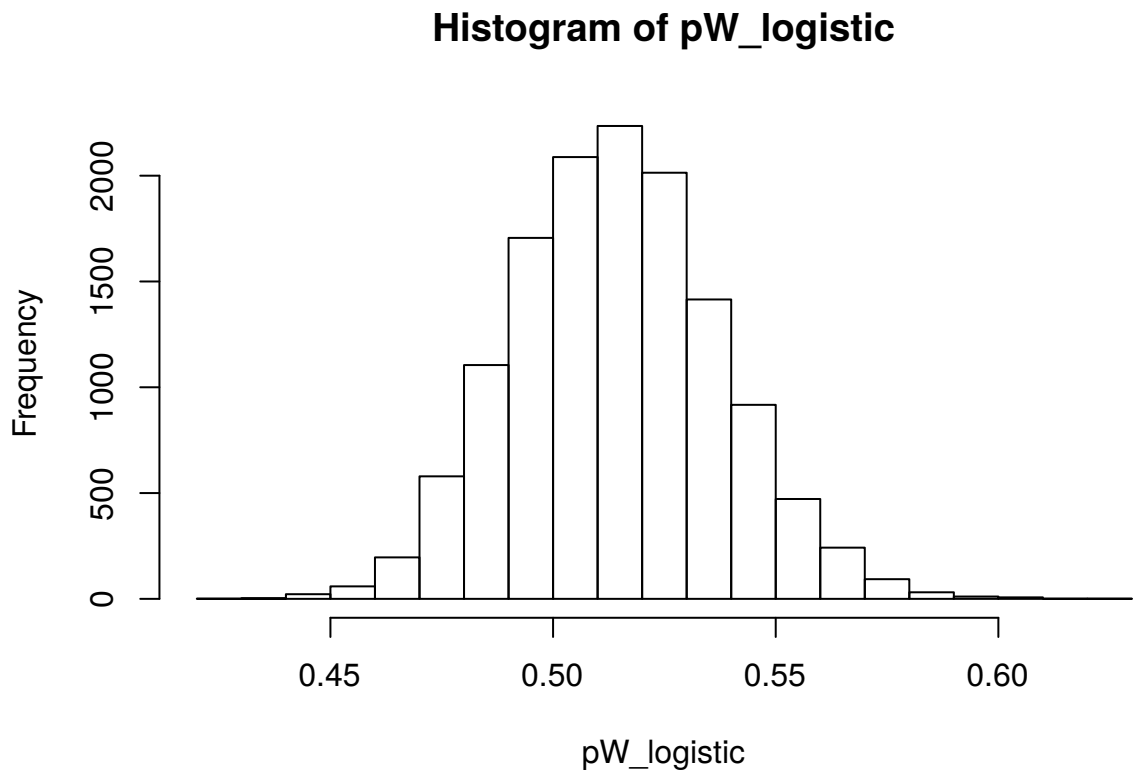
```
#### Outcome And Treatment Summary ####
```

Outcome and Treatment Summary

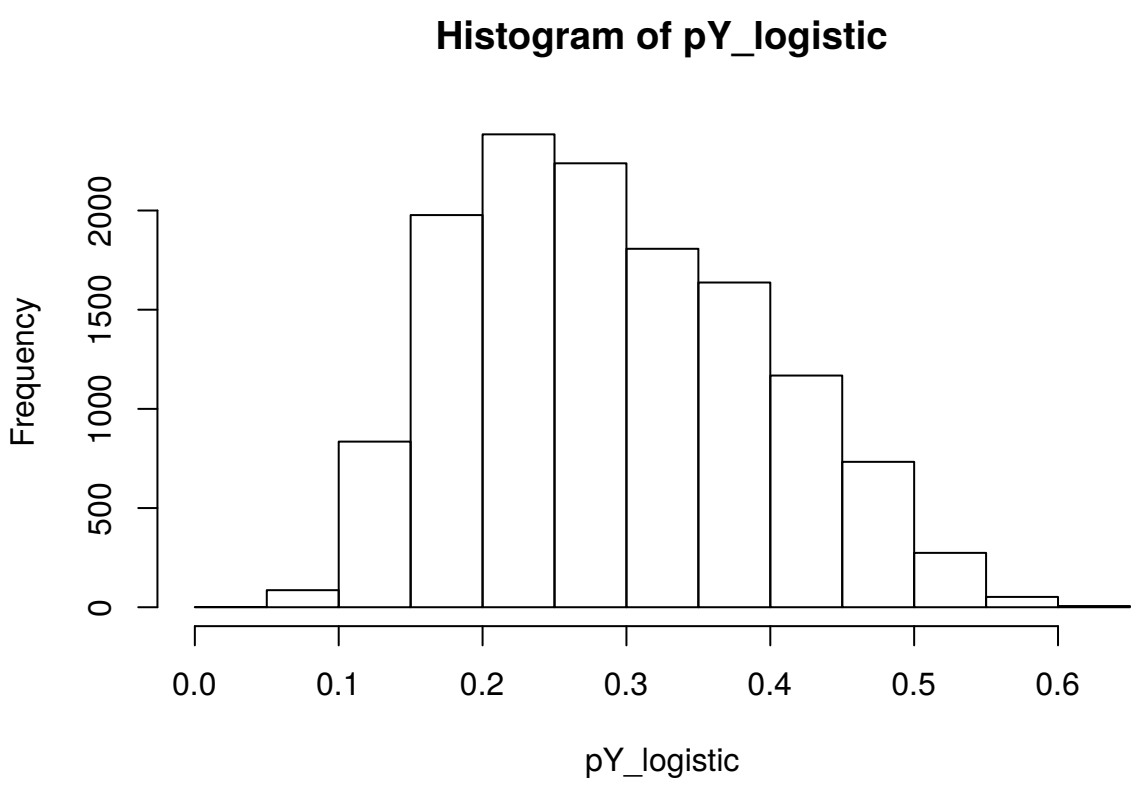
Here is a summary of our outcome and treatment variables

We plot a histogram of treatment propensities and outcome probabilities, since our outcome is binary 0-1. These are learnt using a logistic regression model. Treatment assignment is roughly normally distributed as per this model, with a mean of 0.5. Outcome is skewed right with a mean 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)
```

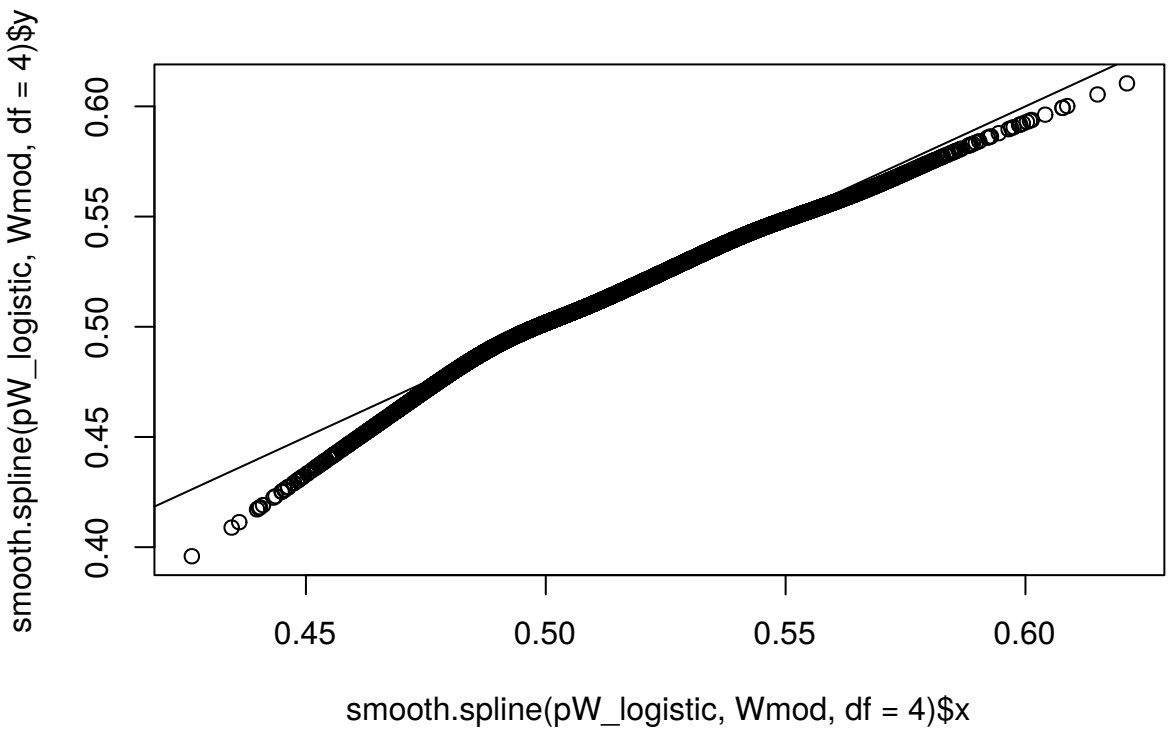



```
pY_logistic.fit <- glm(Ymod ~ as.matrix(Xmod), family = "binomial")
pY_logistic <- predict(pY_logistic.fit, type = "response")
pY_logistic.fit.tidy <-  pY_logistic.fit %>% tidy()
hist(pY_logistic)
```




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

The following plot compares predicted and actual treatment assignment probabilities. We see that the predicted probability tracks the true probability well.



RCT Analysis

We now report the gold standard treatment effect and its confidence intervals

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

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

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

Introducing Bias

Now we introduce bias, as per the assignment, to introduce confounding, as if our data is from an observational study. Our biasing rule is simple. We analyzed the data using rpart (a tree) to check which covariates affect outcomes the most. We took hints from it to drop higher outcomes from one group, and lower outcomes from the other group. Then, we analyze how various methods correct for this confounding.

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`)

```
    # - Republican-leaning independent or closer to the Republican party (`partyid >= 4 & partyid < 7
    # - More than slightly conservative (`polviews >= 5.5 & polviews < 8`)
    # - with self-reported prestige of occuption above the bottom quartile (`prestg80 >= 34`)
```

We remove 40.0 percent of observations in these sets.

We report the new difference in means estimator and note that is signficantly biased. The gold standard difference in means also lies outside the confidence intervals that are reported

```
tauhat_confounded <- difference_in_means(df_mod)
tauhat_confounded
```

```
##      ATE lower_ci upper_ci
##   -0.280   -0.294   -0.265
```

```
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)
pW_logistic.fit <- glm(Wmod ~ as.matrix(Xmod), family = "binomial")
pW_logistic <- predict(pW_logistic.fit, type = "response")
# 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)
# tauhat_lin_logistic_both <- aipw_both(df_mod, pW_logistic, pY_logistic.fit)

# tauhat_logistic_ipw
# tauhat_pscore_ols
# tauhat_lin_logistic_aipw

#### LOGISTIC PROPENSITY SCORES ####
# 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")

# hist(pW_logistic)

df_mod[, logistic_propensity := pW_logistic]
```

We now show summary statistics of the modified dataset

```
library('stargazer')
stargazer(df_mod)
```

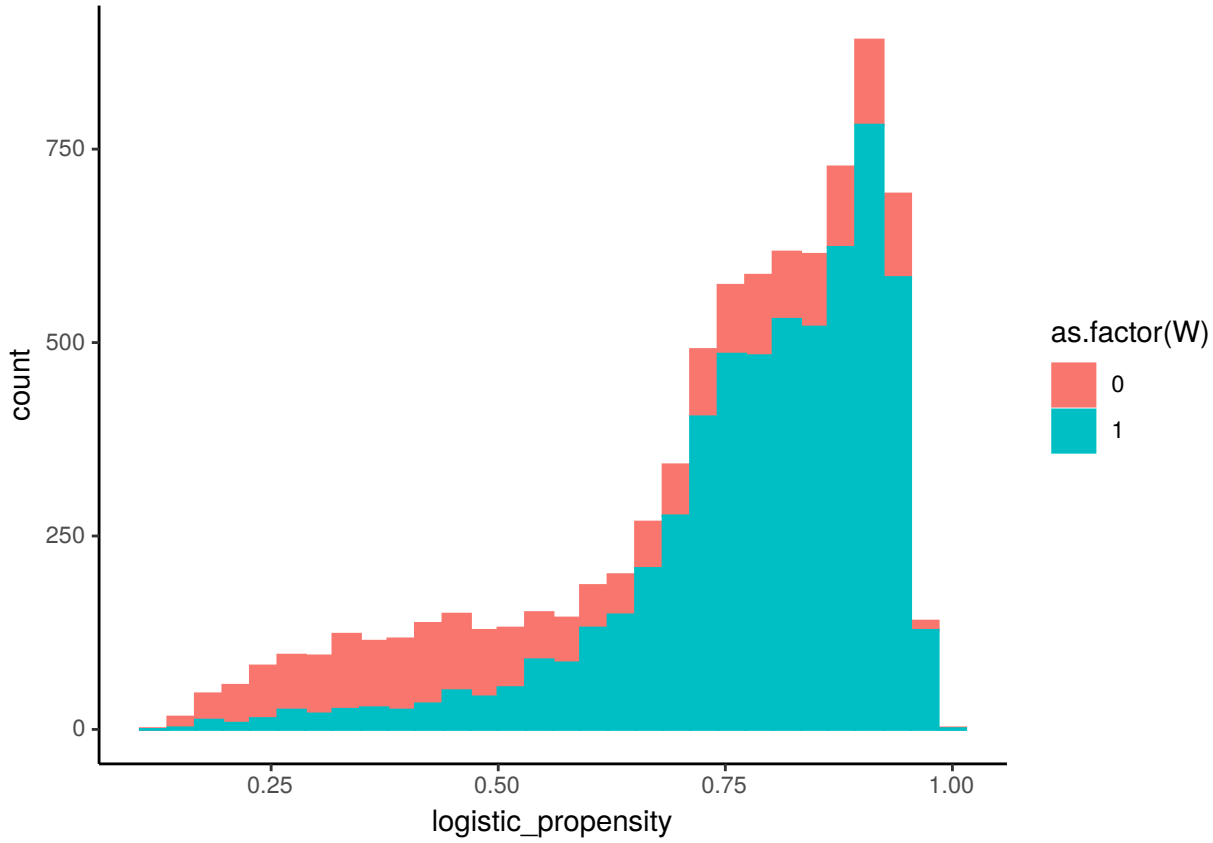
```
##
```

```
## % Table created by stargazer v.5.2.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harva
## % Date and time: Thu, Apr 30, 2020 - 23:08:32
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}}lccccccc}
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
## Statistic & \multicolumn{1}{c}{N} & \multicolumn{1}{c}{Mean} & \multicolumn{1}{c}{St. Dev.} & \multi
## \hline \\[-1.8ex]
## wrkstat & 7,918 & 1.160 & 0.365 & 1 & 1 & 1 & 2 \\
## hrs1 & 7,918 & 42.200 & 14.100 & 0 & 38 & 50 & 89 \\
## wrkslf & 7,918 & 1.870 & 0.331 & 1 & 2 & 2 & 2 \\
## occ80 & 7,918 & 336.000 & 245.000 & 5 & 156 & 467 & 889 \\
## prestg80 & 7,918 & 45.100 & 13.900 & 17 & 34 & 52 & 86 \\
## indus80 & 7,918 & 603.000 & 273.000 & 10 & 410 & 840 & 932 \\
## marital & 7,918 & 2.450 & 1.690 & 1 & 1 & 4 & 5 \\
## sibs & 7,918 & 3.460 & 2.980 & 0 & 2 & 4 & 37 \\
## childs & 7,918 & 1.580 & 1.490 & 0 & 0 & 2 & 8 \\
## age & 7,918 & 40.500 & 12.200 & 18 & 31 & 49 & 85 \\
## educ & 7,918 & 14.000 & 2.760 & 0 & 12 & 16 & 20 \\
## maeduc & 7,918 & 11.700 & 3.330 & 0 & 11 & 13 & 20 \\
## degree & 7,918 & 1.730 & 1.180 & 0 & 1 & 3 & 4 \\
## sex & 7,918 & 1.500 & 0.500 & 1 & 1 & 2 & 2 \\
## race & 7,918 & 1.240 & 0.559 & 1 & 1 & 1 & 3 \\
## res16 & 7,918 & 3.540 & 1.530 & 1 & 3 & 5 & 6 \\
## reg16 & 7,918 & 4.390 & 2.620 & 0 & 2 & 6 & 9 \\
## mobile16 & 7,918 & 1.950 & 0.854 & 1 & 1 & 3 & 3 \\
## family16 & 7,918 & 1.840 & 1.670 & 0 & 1 & 1 & 8 \\
## born & 7,918 & 1.090 & 0.287 & 1 & 1 & 1 & 2 \\
## parborn & 7,918 & 0.897 & 2.440 & 0 & 0 & 0 & 8 \\
## hompop & 7,918 & 2.660 & 1.410 & 1 & 2 & 4 & 11 \\
## babies & 7,918 & 0.232 & 0.550 & 0 & 0 & 0 & 4 \\
## preteen & 7,918 & 0.310 & 0.663 & 0 & 0 & 0 & 5 \\
## teens & 7,918 & 0.218 & 0.532 & 0 & 0 & 0 & 4 \\
## adults & 7,918 & 1.900 & 0.773 & 1 & 1 & 2 & 8 \\
## earnrs & 7,918 & 1.740 & 0.832 & 0 & 1 & 2 & 8 \\
## income & 7,918 & 11.300 & 1.660 & 1 & 11 & 12 & 12 \\
## rincome & 7,918 & 10.200 & 2.770 & 1 & 9 & 12 & 12 \\
## partyid & 7,918 & 2.930 & 2.030 & 0 & 1 & 5 & 7 \\
## polviews & 7,918 & 4.090 & 1.360 & 1 & 3 & 5 & 7 \\
## W & 7,918 & 0.735 & 0.442 & 0 & 0 & 1 & 1 \\
## Y & 7,918 & 0.194 & 0.396 & 0 & 0 & 0 & 1 \\
## logistic\_propensity & 7,918 & 0.735 & 0.191 & 0.110 & 0.652 & 0.884 & 0.988 \\
## \hline \\[-1.8ex]
## \end{tabular}
## \end{table}
```

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

We calculate propensity of treatment based on pre treatment covariates. We show that we have significant overlap. There are some propensity scores close to 1. We remove all observations with propensity score $>$ 0.95 to fix this and relearn the propensity model. We show overlap both before and after. Overlap before truncating

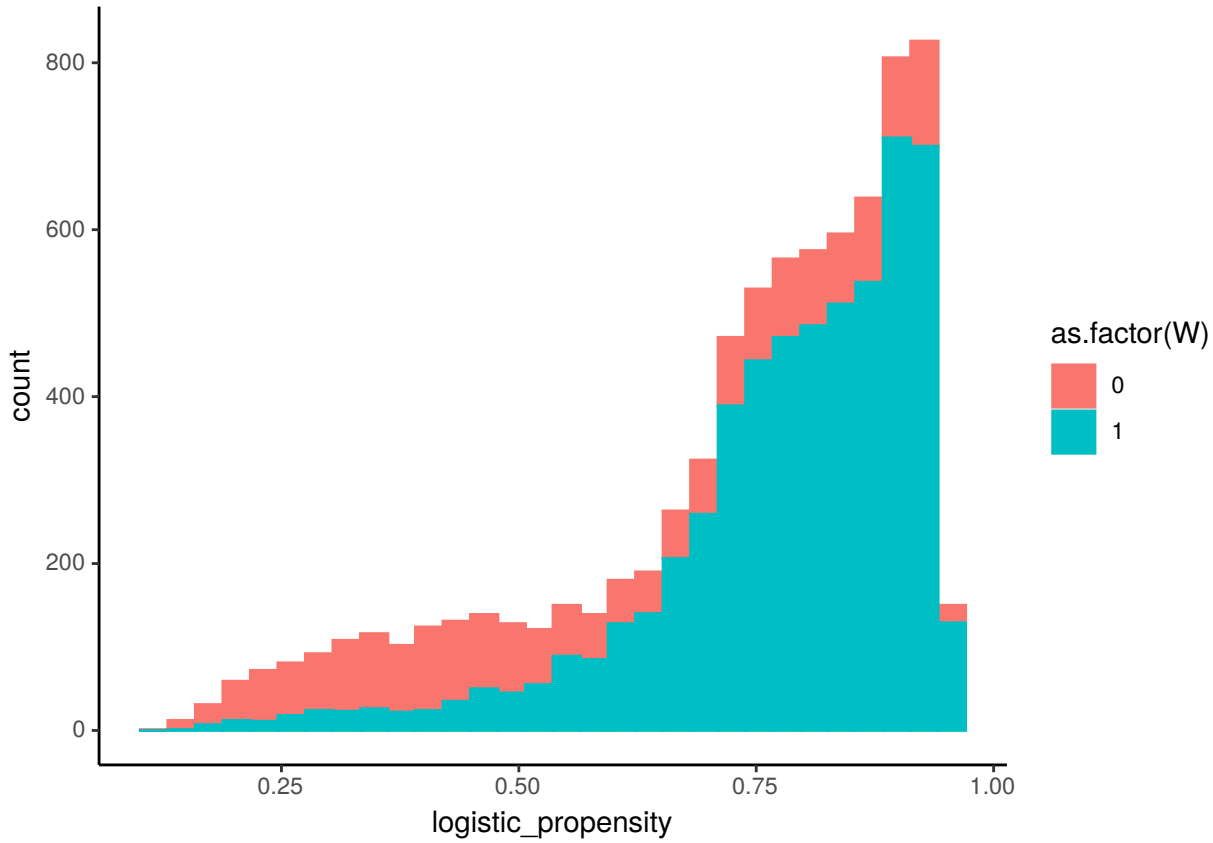
```
overlap <- df_mod %>% ggplot(aes(x=logistic_propensity,color=as.factor(W),fill=as.factor(W)))+ geom_his
overlap
```




```
df_mod <- df_mod[logistic_propensity %between% c(0.05, 0.95)]

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")

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

Overlap after truncating

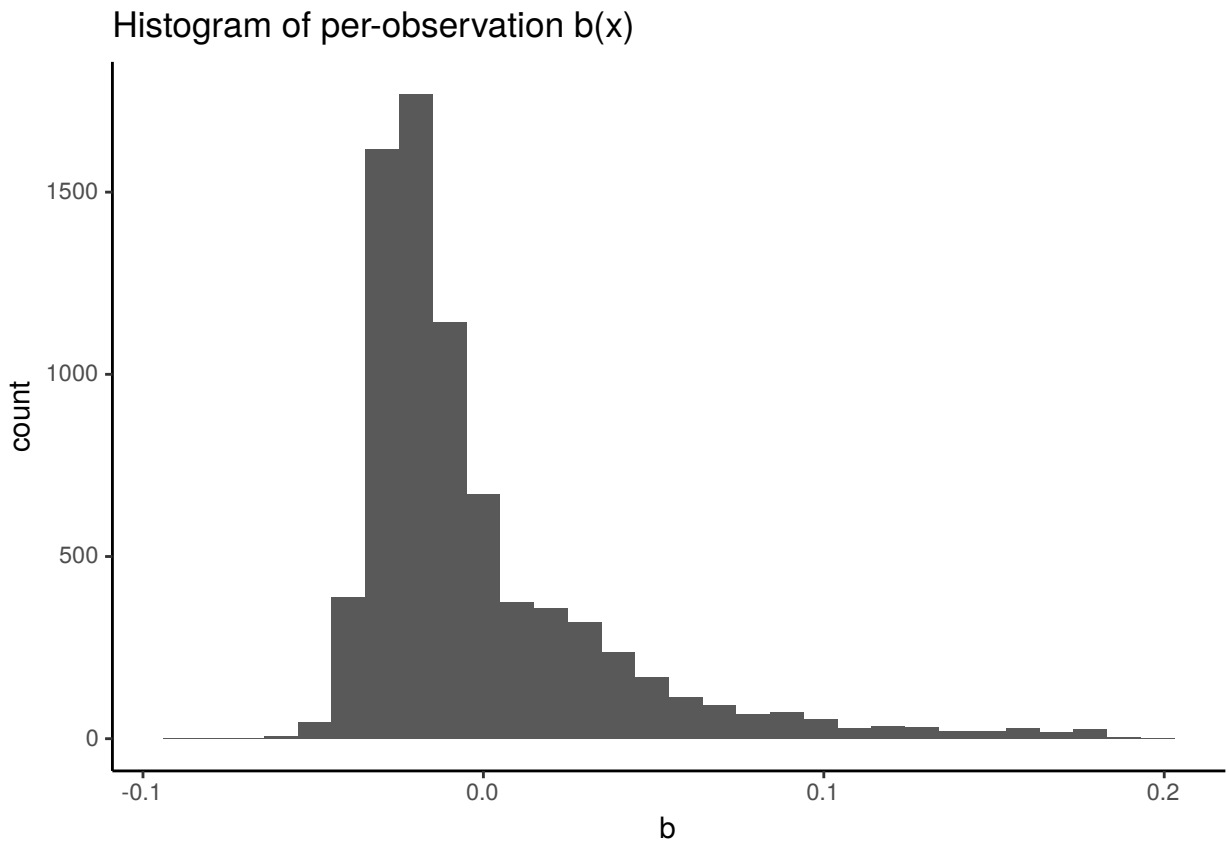
```
overlap
```




```r
#### 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
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
```

```r
# original data
pW_glmnet.fit.propensity = glmnet::cv.glmnet(Xmod, Wmod, lambda = lambda, family = "binomial", type.meas
pY_glmnet.fit.propensity = glmnet::cv.glmnet(Xmod, Ymod, lambda = lambda, family = outcome_family, type
# expanded data
pW_glmnet.fit.propensity.int = glmnet::cv.glmnet(Xmod.int, Wmod, lambda = lambda, family = "binomial",
pY_glmnet.fit.propensity.int = glmnet::cv.glmnet(Xmod.int, Ymod, lambda = lambda, family = outcome_famil

# demonstration of lasso fit across lambdas:
pW_lasso = pW_glmnet.fit.propensity$fit.preval[, pW_glmnet.fit.propensity$lambda == pW_glmnet.fit.proper
pW_lasso.min = pW_glmnet.fit.propensity$fit.preval[, pW_glmnet.fit.propensity$lambda == min(pW_glmnet.fi
pW_lasso.max = pW_glmnet.fit.propensity$fit.preval[, pW_glmnet.fit.propensity$lambda == max(pW_glmnet.fi
pW_lasso.rand = pW_glmnet.fit.propensity$fit.preval[, pW_glmnet.fit.propensity$lambda == base::sample(pW

# glmnet.fit.propensity = glmnet::cv.glmnet(Xmod.int, Wmod, family = "binomial",keep=TRUE)
pW_lasso.int = pW_glmnet.fit.propensity.int$fit.preval[, pW_glmnet.fit.propensity.int$lambda == pW_glmne
pW_lasso.int.min = pW_glmnet.fit.propensity.int$fit.preval[, pW_glmnet.fit.propensity.int$lambda == min
pW_lasso.int.max = pW_glmnet.fit.propensity.int$fit.preval[, pW_glmnet.fit.propensity.int$lambda == max
pW_lasso.int.rand = pW_glmnet.fit.propensity.int$fit.preval[, pW_glmnet.fit.propensity.int$lambda == bas

# FIXME:
# problems calculating probabilities:
# this "should" work:
# pW_lasso.int2 <- predict(pW_glmnet.fit.propensity.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.propensity.int, newx = Xmod.int,  s=pW_glmnet.fit.propensity.i
# pW_lasso.int4 <- predict(pW_glmnet.fit.propensity.int, newx = Xmod.int,  s="lambda.min", type = "resp
# pW_lasso.int5 <- predict(pW_glmnet.fit.propensity.int, newx = Xmod.int,  s=pW_glmnet.fit.propensity.i
# pW_lasso.int6 <- predict(pW_glmnet.fit.propensity.int, newx = Xmod.int,  s=pW_glmnet.fit.propensity.i

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

# 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

# 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 see skew in the bias, and note that the mean lies at NA

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

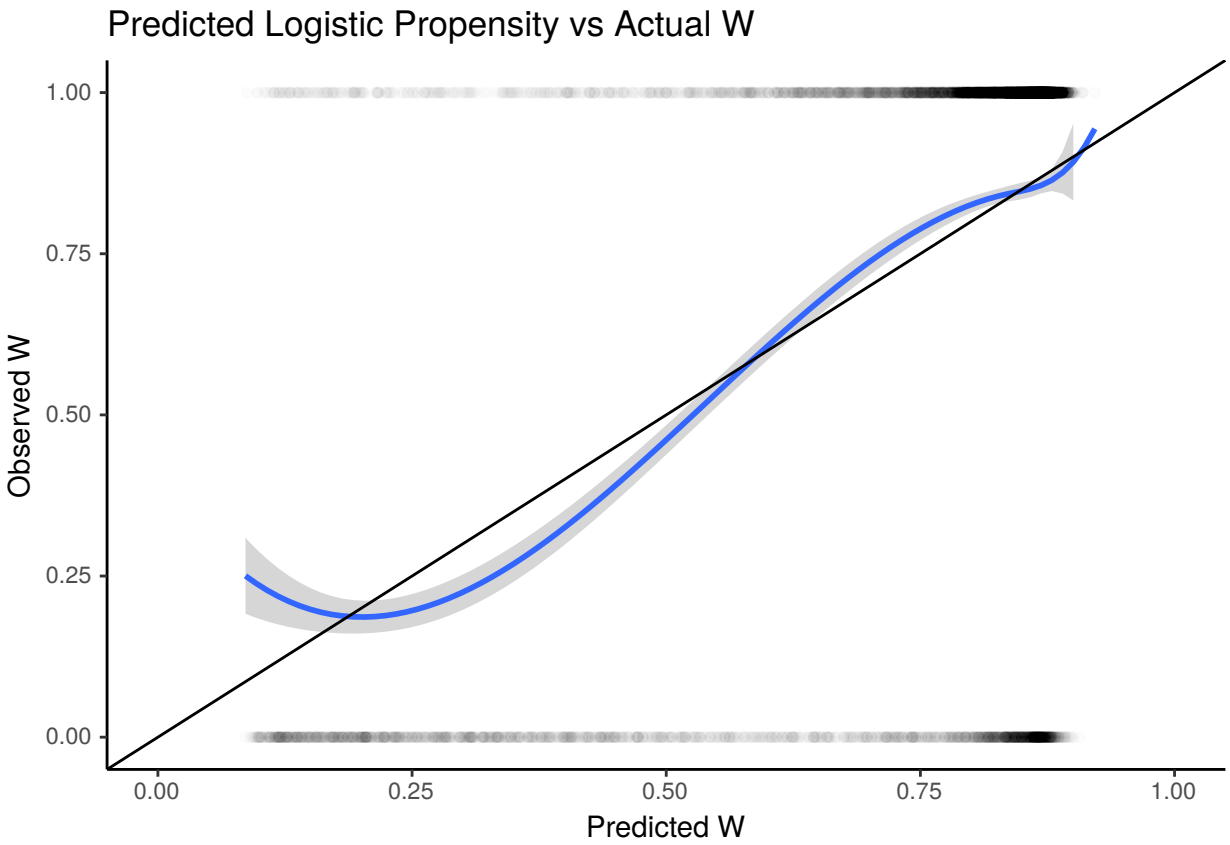
ATE on biased data

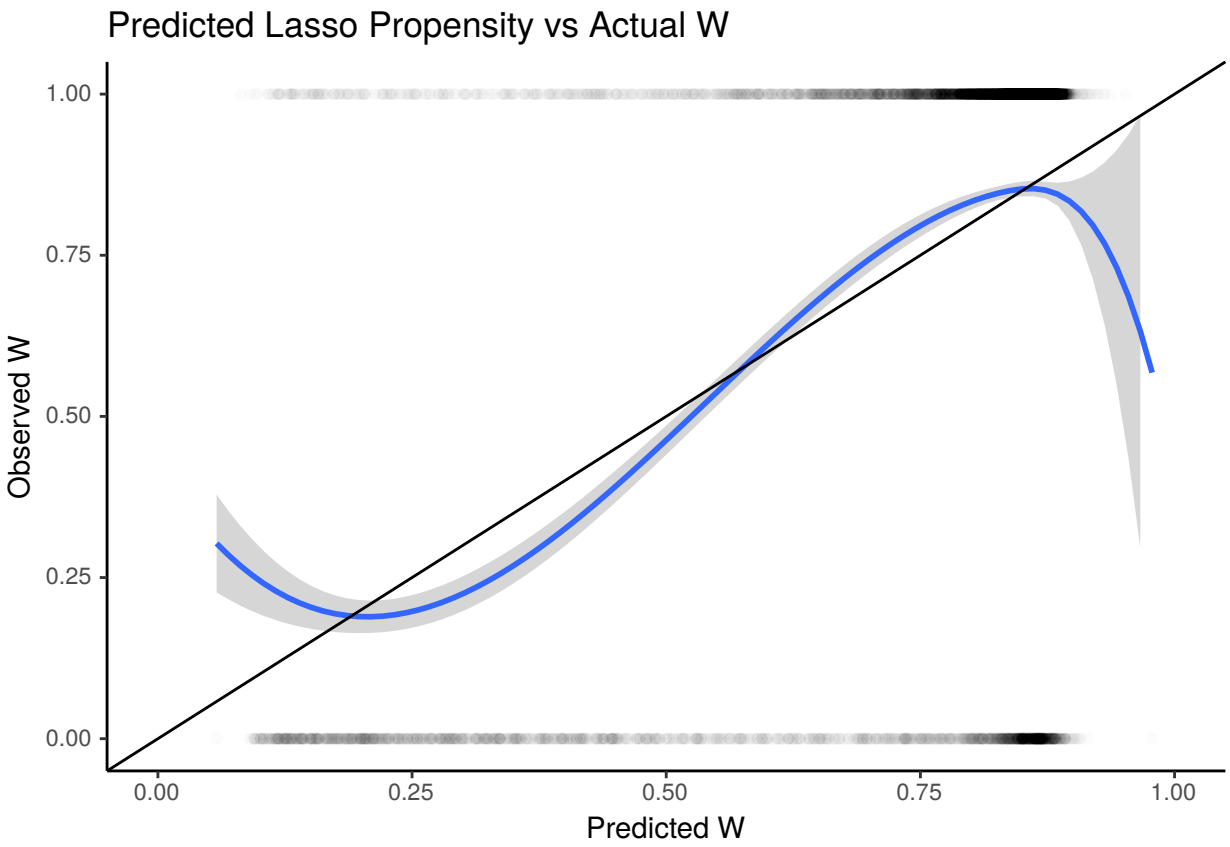
```
#` We measure ATE and confidence intervals on our biased data using the following methods, to explore t
```

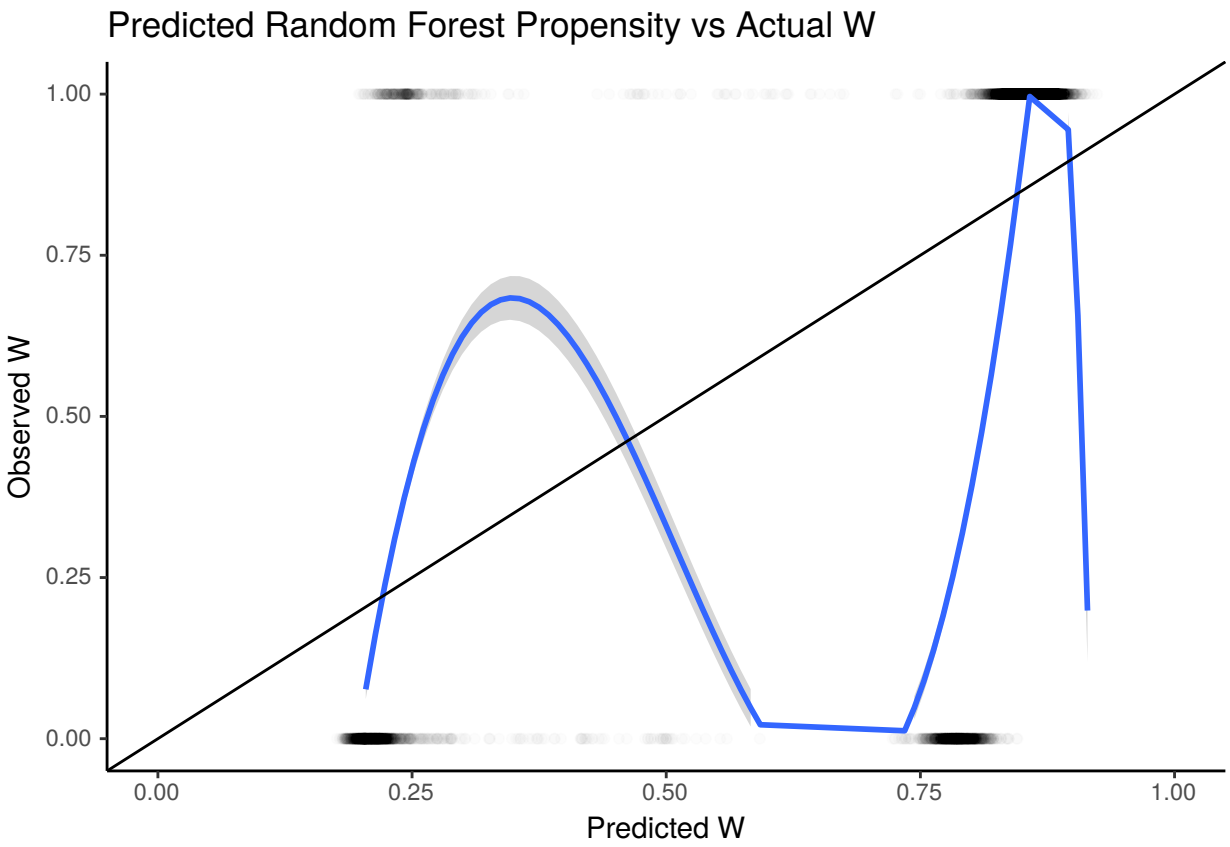
1. IPW with various propensity models
2. OLS regression with propensity weights using various propensity models
3. AIPW (Augmented inverse propensity score), doubly robust methods, using various propensity and outcome models.

```
#### PLOTTING PROPENSITY PERFORMANCE ####
```

Now we analyze each propensity model by plotting how they track the true propensity of treatment. Models close to the Y=X line are better. Logistic and Lasso Propensity scores perform the best in our original dataset.

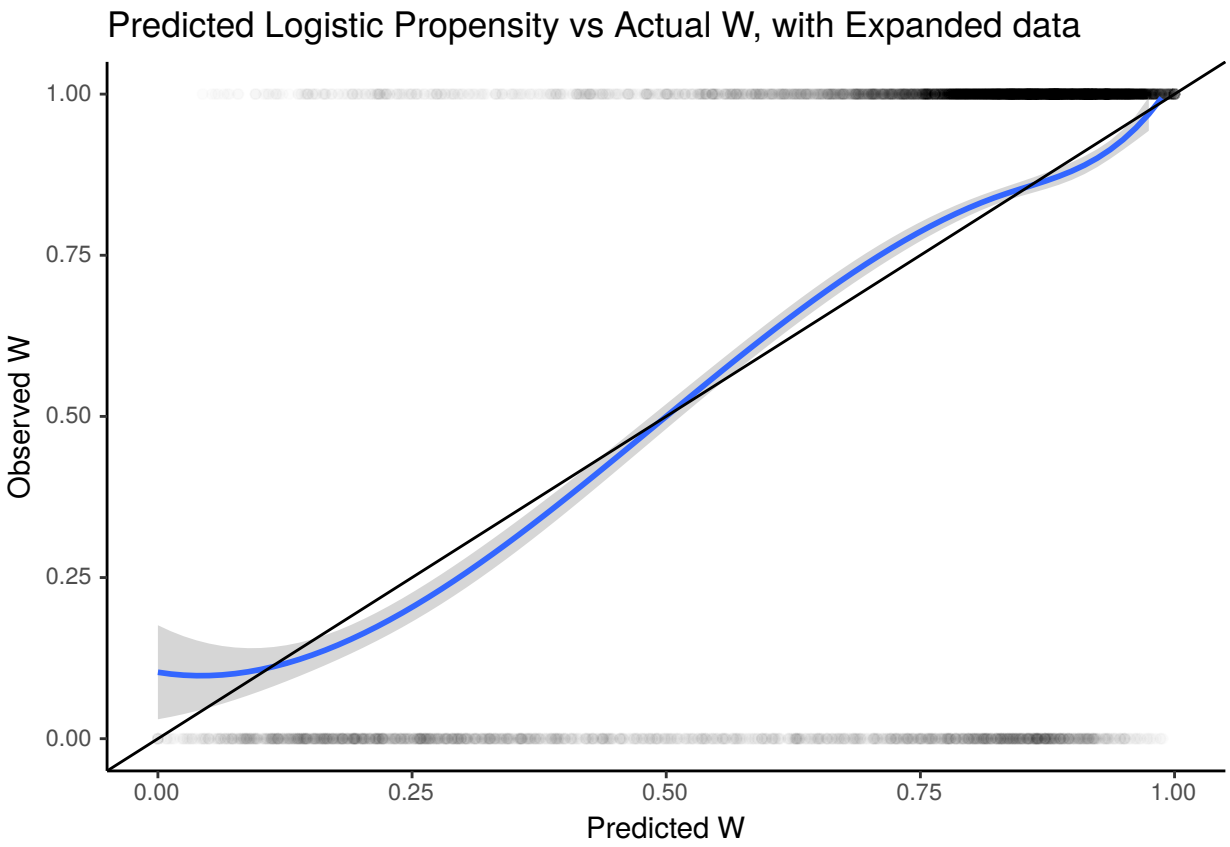



Predicted Lasso Propensity vs Actual W
Observed W
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Predicted W

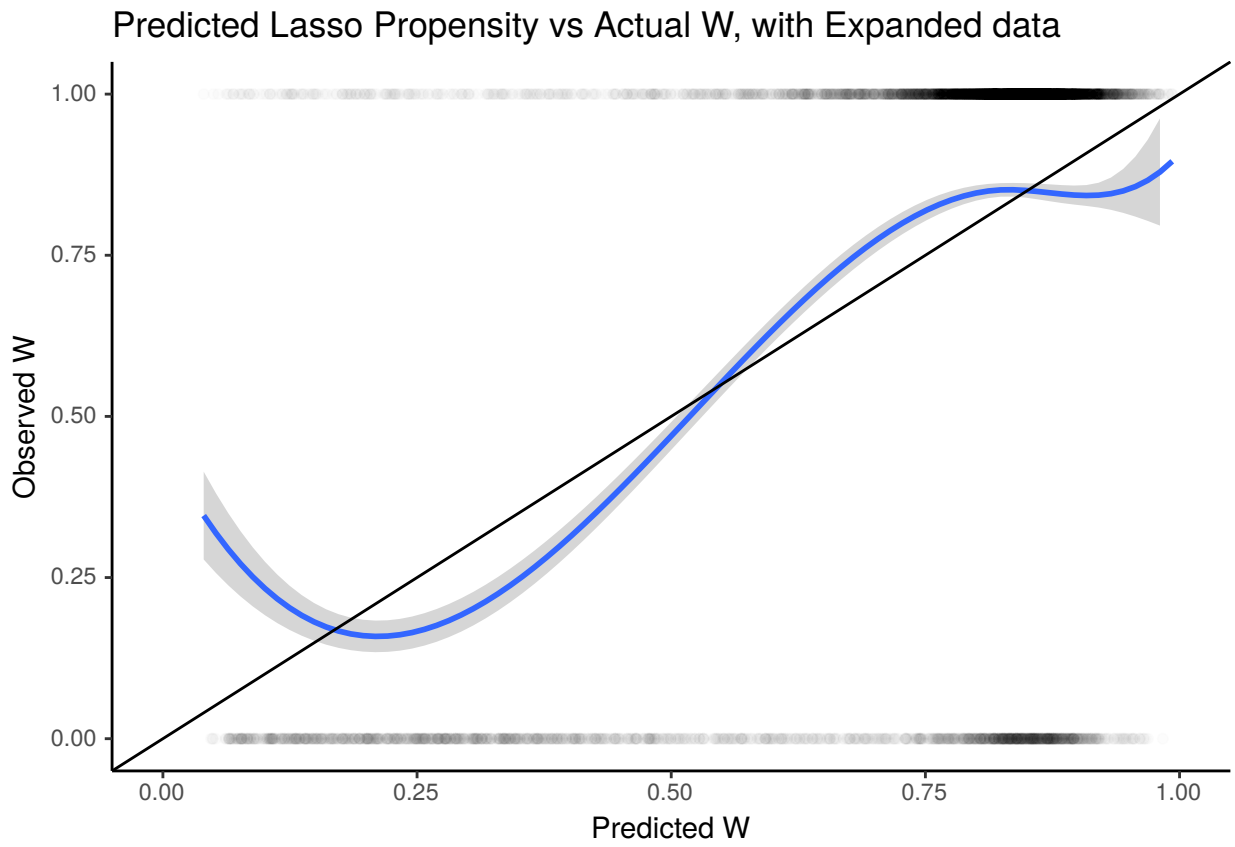


We create an expanded dataset by adding two way interaction effects for all our pre treatment covariates We then add plots for treatment propensity models using this dataset. Once again, Logistic Regression and Lasso perform the best.

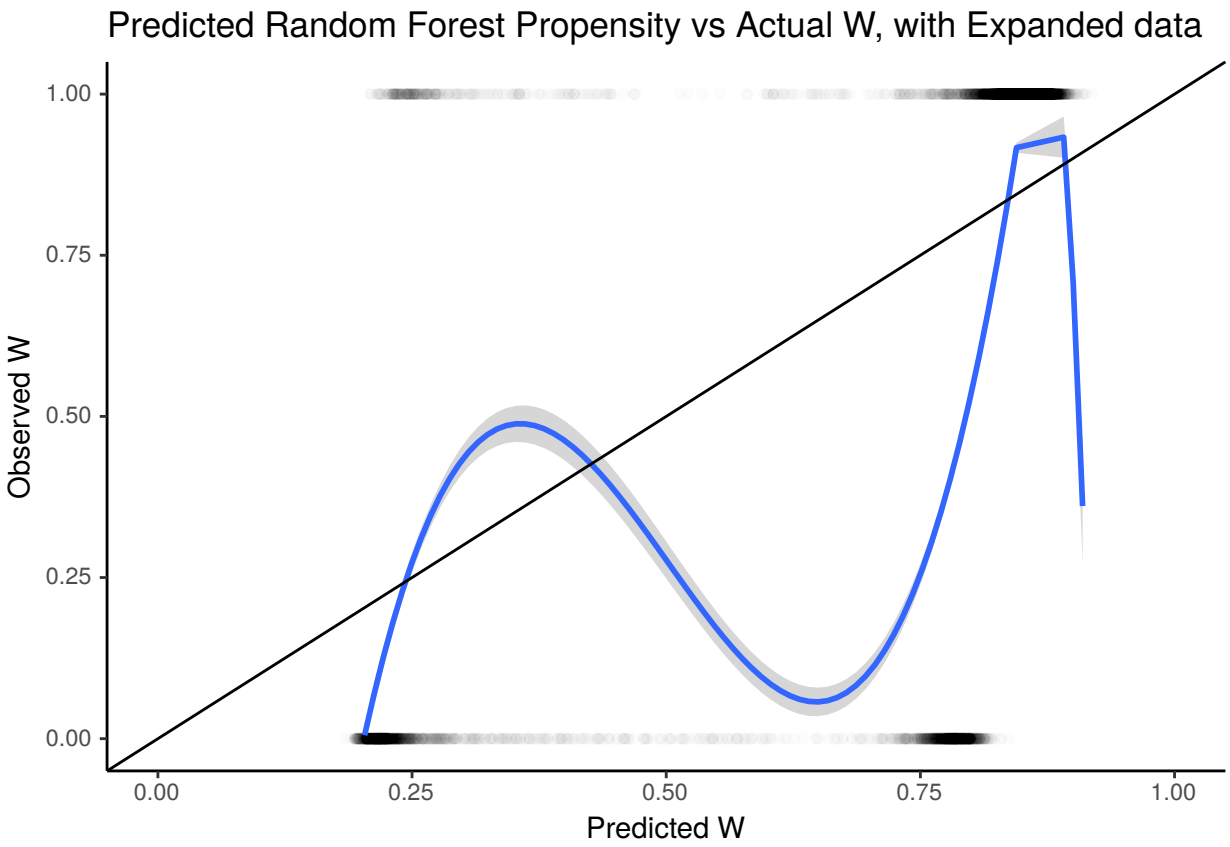
```
plot_prob(pW_logistic.int, Wmod, "Logistic", "Expanded")
```




```
plot_prob(pW_lasso.int, Wmod, "Lasso", "Expanded")
```




```
plot_prob(pW_rf.int, Wmod, "Random Forest", "Expanded")
```

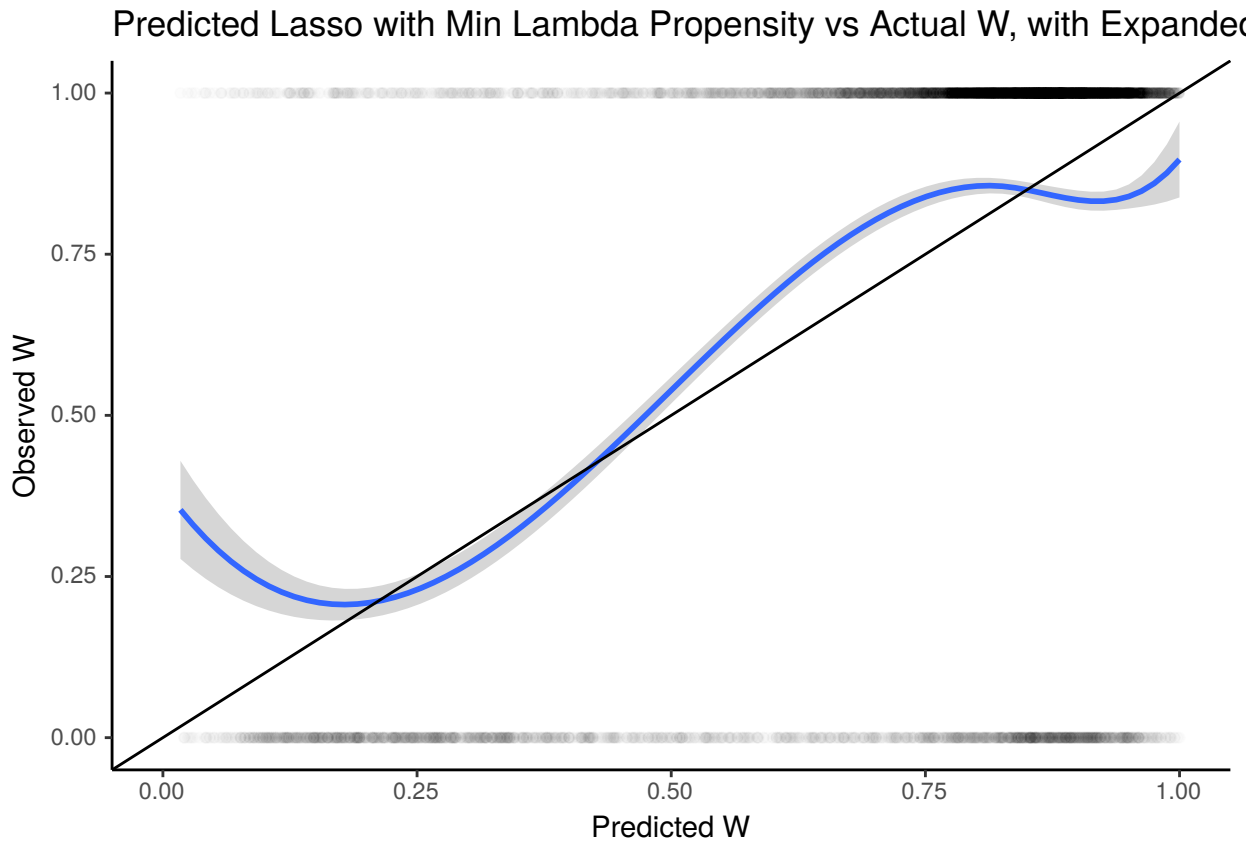



```
#### EXPLORING LASSO ####
```

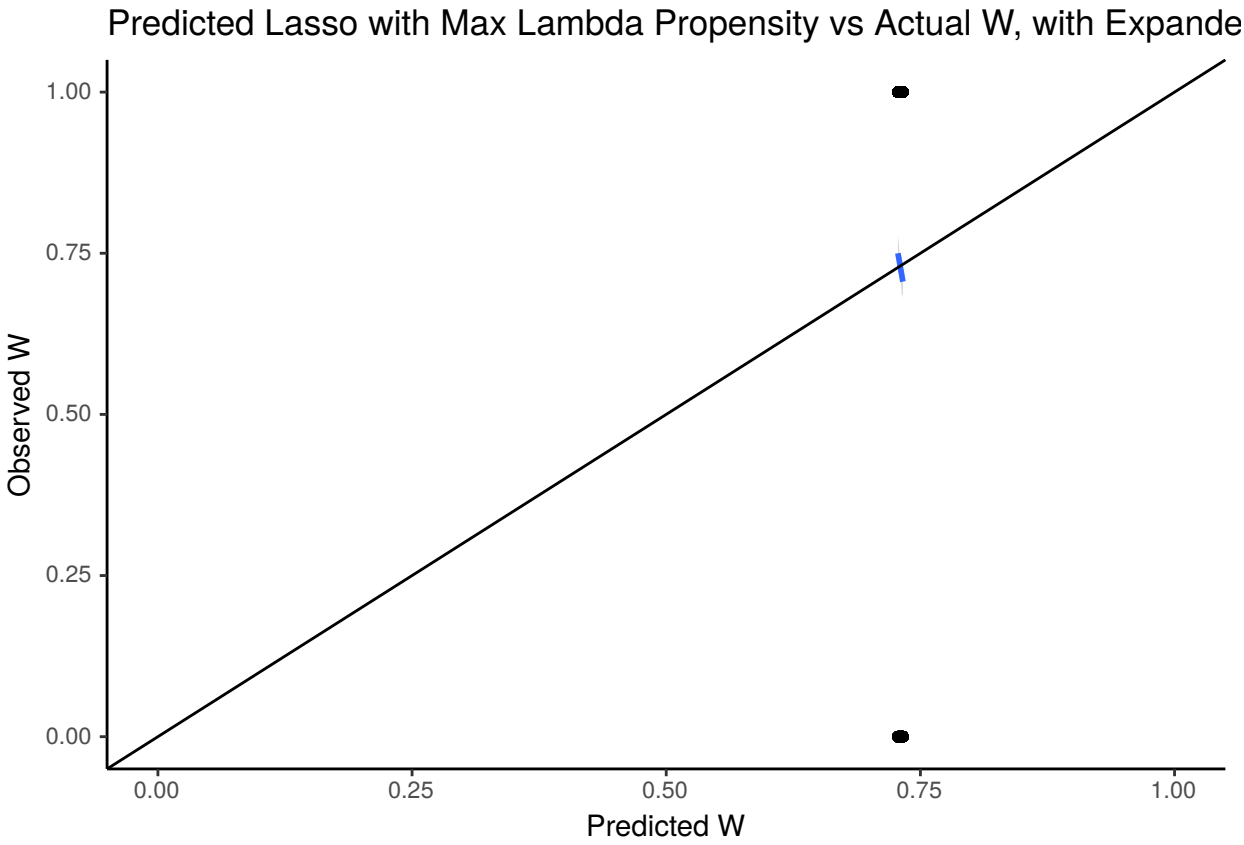
Lasso Cross Validation

We plot how the lasso propensity models track the true propensity with various cross validation parameters. Here the lasso is plotted for the larger, interacted, covariate set. The lasso with the best cross validation parameter performs the best.

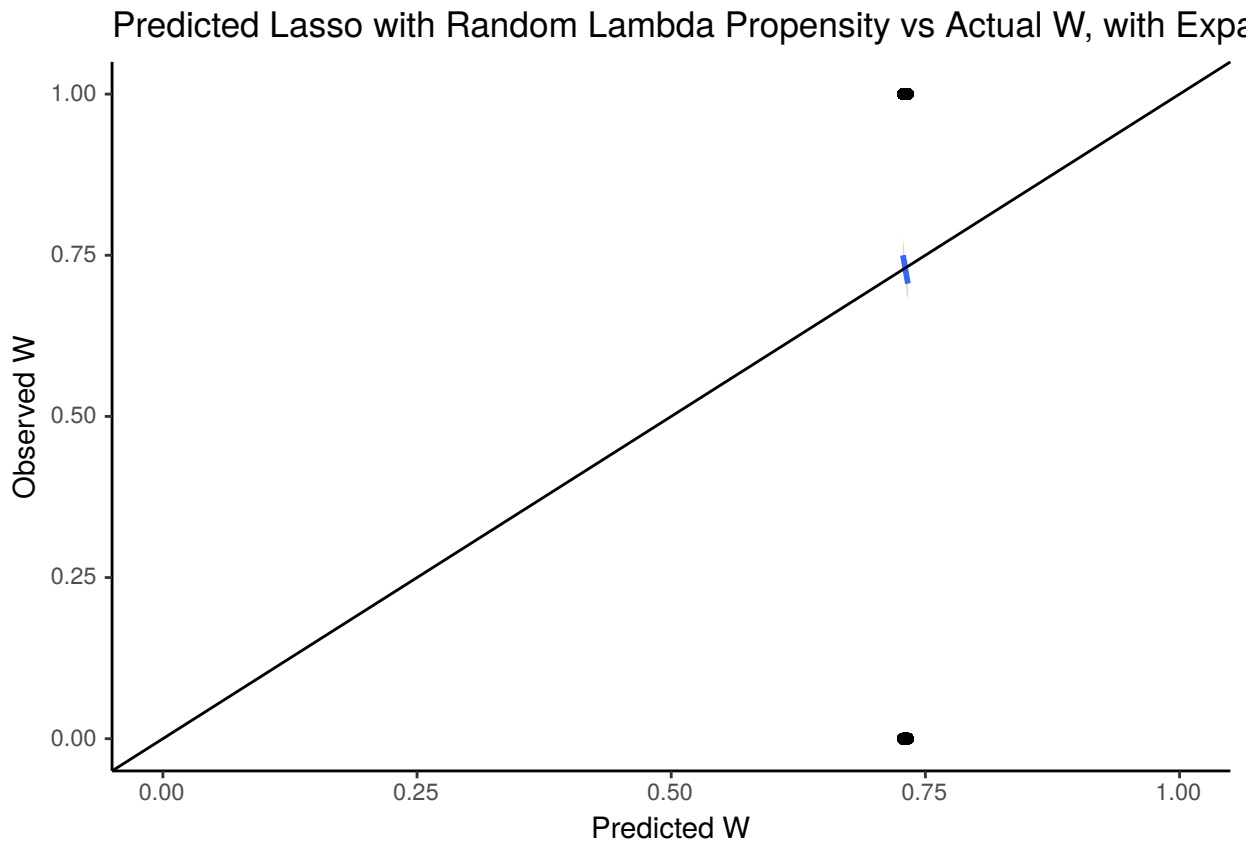
```
plot_prob(pW_lasso.int.min, Wmod, "Lasso with Min Lambda", "Expanded")
```




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

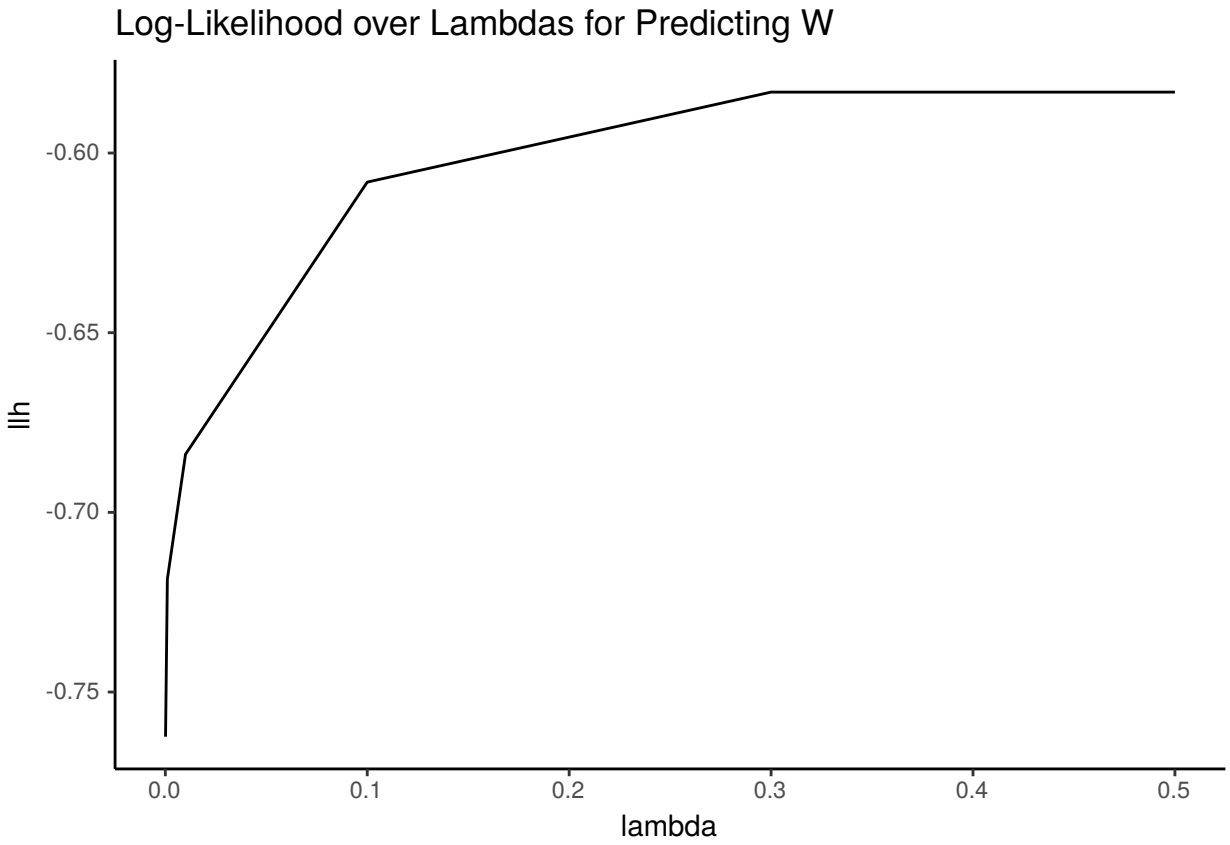


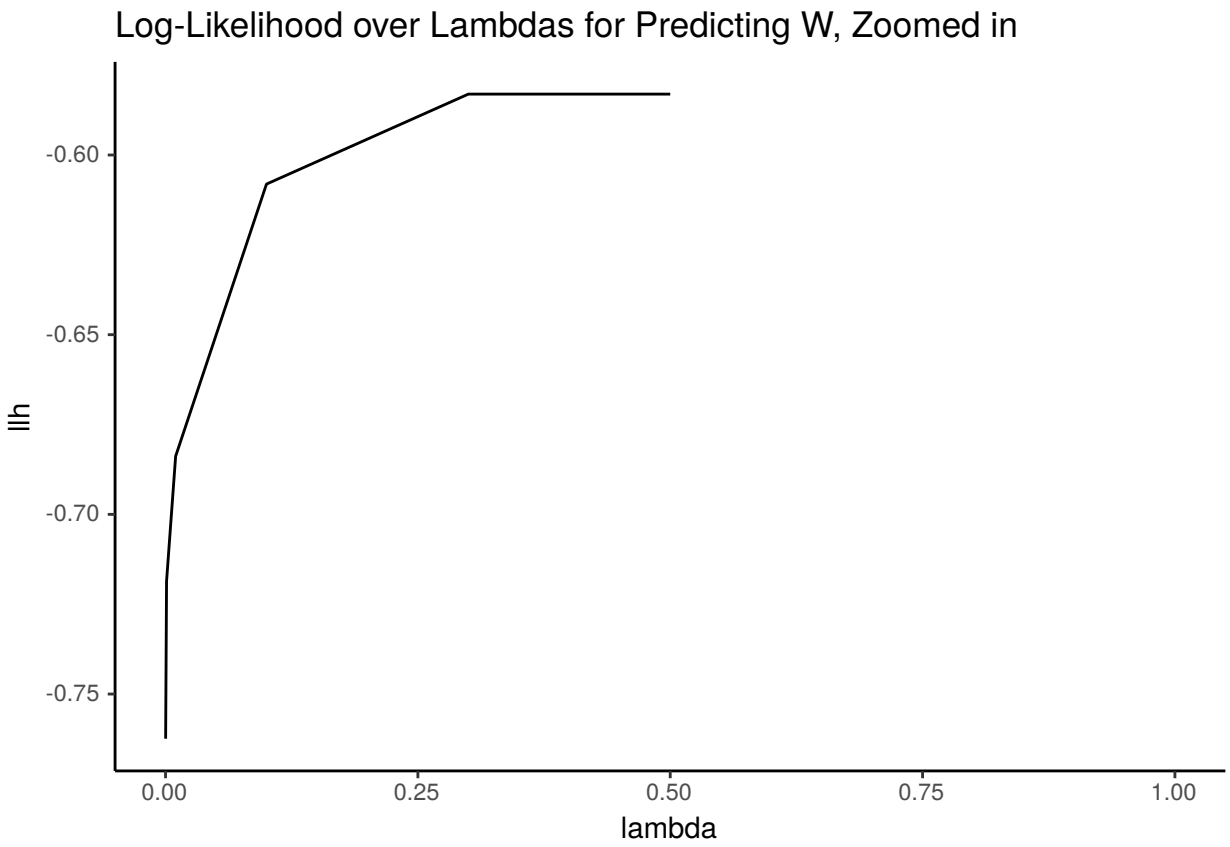

```
plot_prob(pW_lasso.int.rand, Wmod, "Lasso with Random Lambda", "Expanded")
```




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

# likelihoods over lambdas
lambda_log_liks <- pW_glmnet.fit.propensity.int.lambda_preds[
  ,lapply(.SD, loglike, Wmod), .SDcols = names(pW_glmnet.fit.propensity.int.lambda_preds)]

colnames(lambda_log_liks) <- as.character(pW_glmnet.fit.propensity.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))]
```

We now plot log likelihood over different values of lambda on the expanded data. We see later that the IPW and OLS with propensity score weighting models above perform the best when the log likelihoods of the lasso propensity models are high. This is consistent with our conclusions downstream.

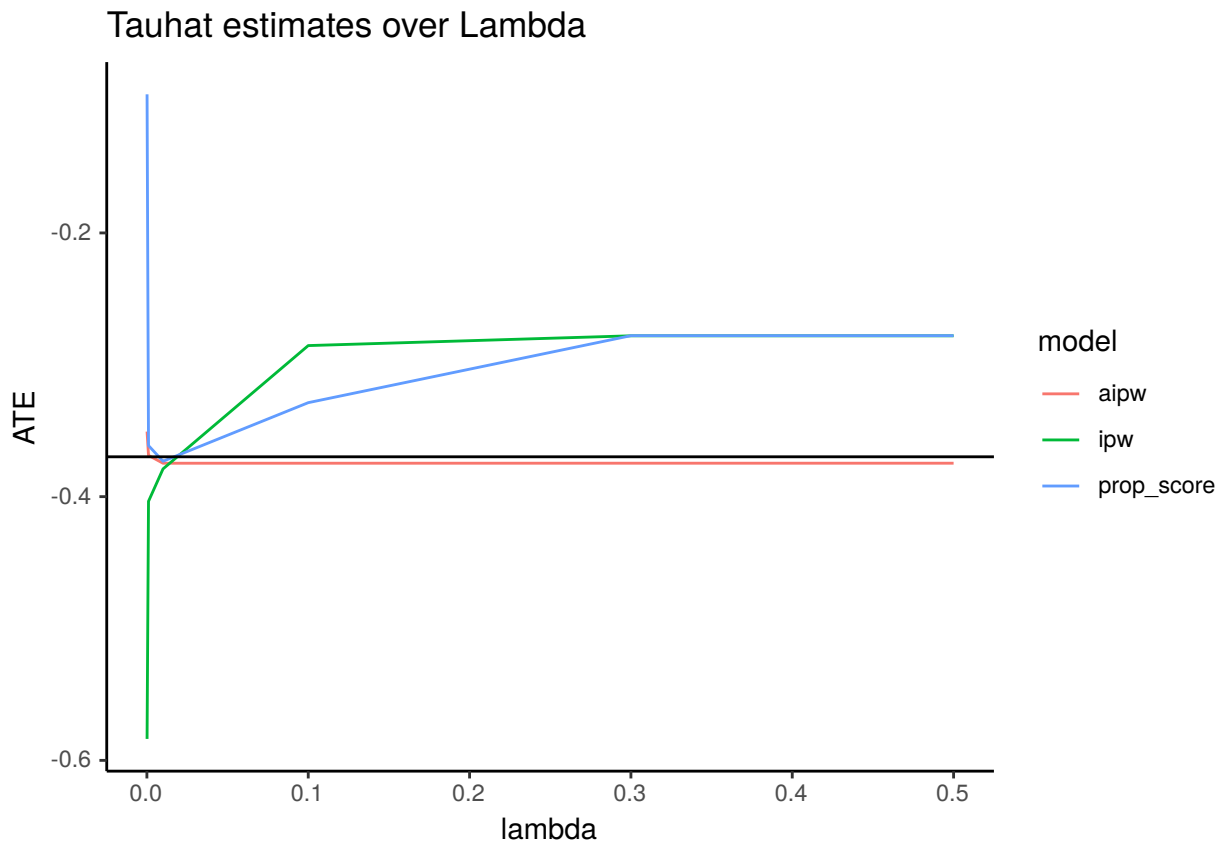

Log-Likelihood over Lambdas for Predicting W
-0.60
-0.65
-0.70
-0.75
llh
0.0
0.1
0.2
0.3
0.4
0.5
lambda



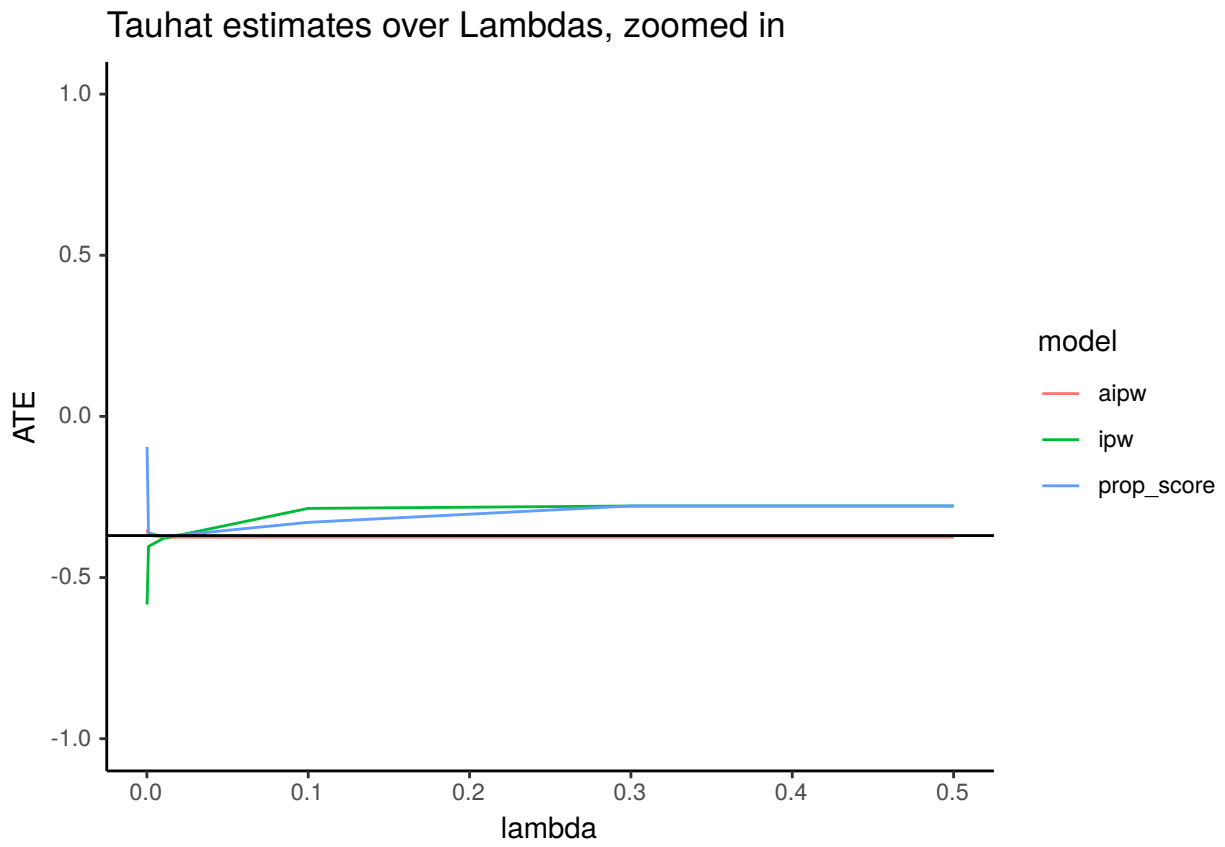
We show the results from the different methods by using propensity models with different lambdas. We note that we only vary the propensity model; even for the case with AIPW we use ordinary OLS along with our lasso propensity models for varying lambdas. We tried to also very the outcome model; we could not get the predict function to work with our glmnet model; hence the limitation

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

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

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

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

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




```
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")
```




```
# 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 learn that as the lambdas are varied, the ipw and OLS with propensity score weighting models start getting unstable. This is probably because the model is misspecified with extreme values of lambdas. If the propensity model is misspecified, We expect to learn incorrect treatment effects. However, with the AIPW, we conclude that since the model is "doubly robust", the underlying outcome model (ordinary OLS) is always correctly specified. As a result, we get stable estimates across all lambdas for the propensity lasso model This teaches us the benefits of using the AIPW estimator!

```
#### EXPLORING CF ####
```

Machine Learning Model of our choice

As per the assignment, we now choose a machine learning model of our choice, causal forests, to understand how prediciton quality varies with sample size

```
# CF
```

First, we use out of bag predictions (cross fitting)

```
cf.int.fit = causal_forest(Xmod.int, Ymod, Wmod, num.trees = 500)
pW_cf.int = predict(cf.int.fit, newdata = Xmod.int) %>% as.matrix
ate_rf_aipw.int = average_treatment_effect(cf.int.fit)
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_aipw.int
```

```
##      ATE.estimate lower_ci.estimate upper_ci.estimate
##             -0.374            -0.402            -0.347
```

```
tauhat_rf_ipw.int = ipw(df_mod.int, pW_cf.int)
tauhat_ols_rf_aipw.int = aipw_ols(df_mod.int, pW_cf.int)

tauhat_rf_ipw.int
```

```
##      ATE lower_ci upper_ci
##   -0.310   -0.327   -0.293
```

```
tauhat_ols_rf_aipw.int
```

```
##      ATE lower_ci upper_ci
##   -0.379   -0.397   -0.361
```

Now, we do not use cross fitting.

```
cf.int.fit = causal_forest(Xmod.int, Ymod, Wmod, num.trees = 500, compute.oob.predictions = FALSE)
pW_cf.int = predict(cf.int.fit, newdata = Xmod.int) %>% as.matrix
ate_rf_aipw.int = average_treatment_effect(cf.int.fit)
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_aipw.int
```

```
##      ATE.estimate lower_ci.estimate upper_ci.estimate
##             -0.373            -0.400            -0.346
```

```
tauhat_rf_ipw.int = ipw(df_mod.int, pW_cf.int)
tauhat_ols_rf_aipw.int = aipw_ols(df_mod.int, pW_cf.int)

tauhat_rf_ipw.int
```

```
##      ATE lower_ci upper_ci
##   -0.310   -0.327   -0.292
```

```
tauhat_ols_rf_aipw.int
```

```
##      ATE lower_ci upper_ci
##   -0.380   -0.398   -0.362
```

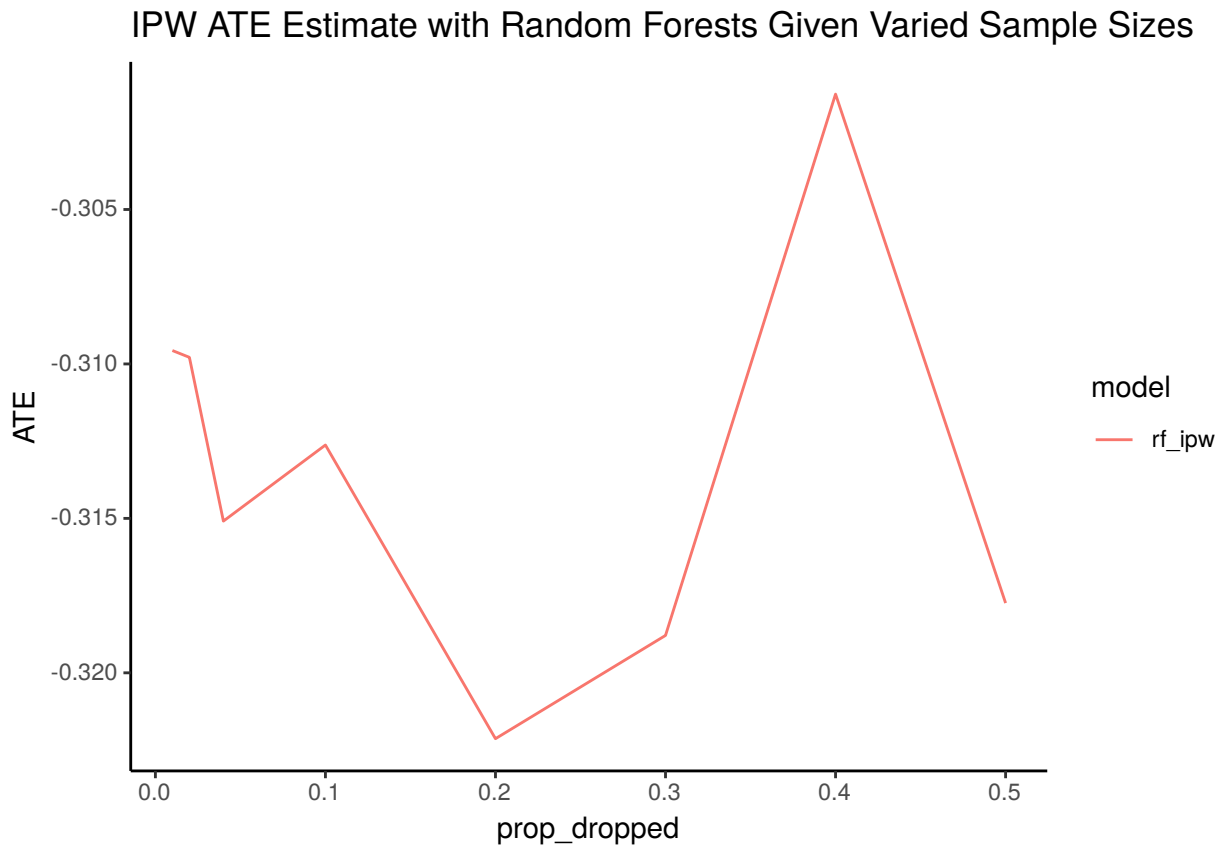
In this setup, we do not see a difference with/without crossfitting Now we vary data sizes with and without cross fitting First, with cross fitting

```
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 = causal_forest(Xmod_temp, Wmod_temp, num.trees = 500)
  #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)
  pW_rf_temp.int.fit = causal_forest(Xmod_temp.int, Ymod_temp, 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"

  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)
}
```

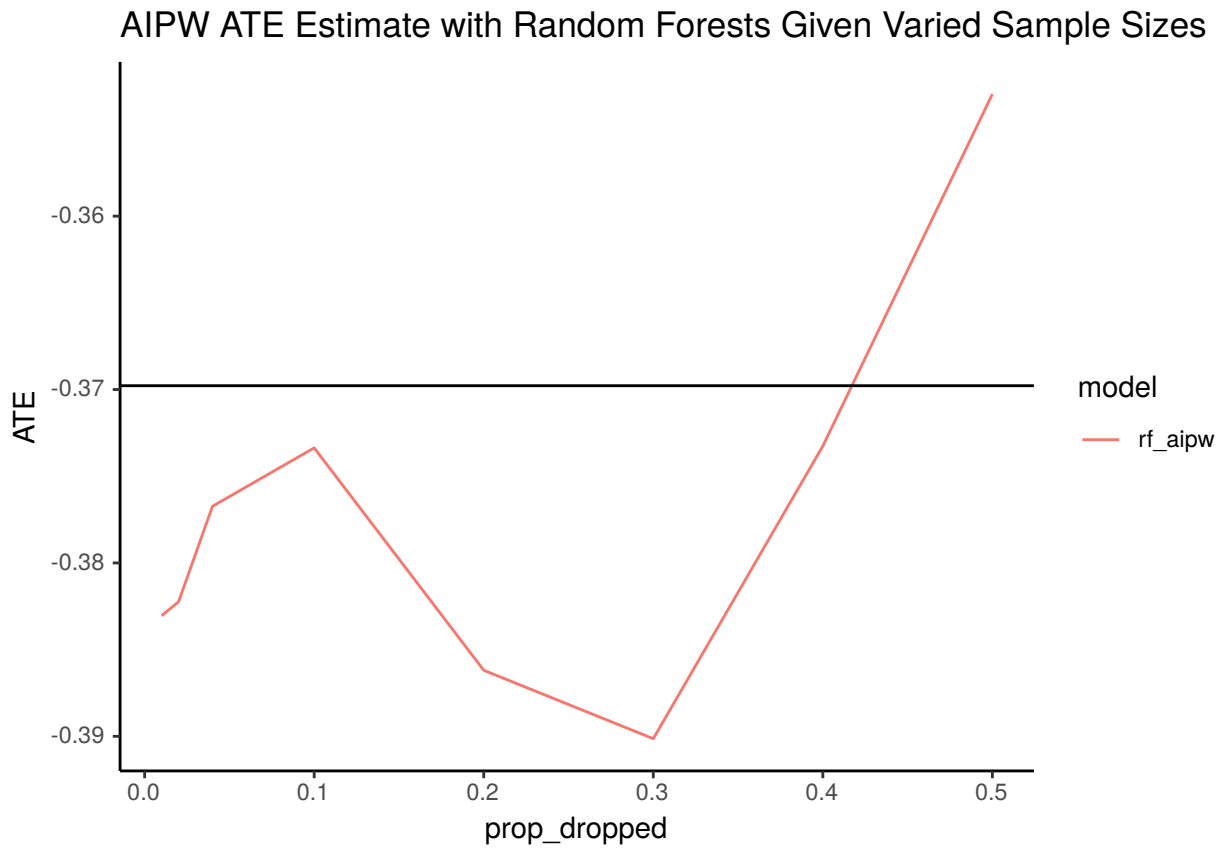
```
##     ATE lower_ci upper_ci prop_dropped  model
```

```
## 1 -0.31   -0.327   -0.292         0.01 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.383   -0.401   -0.365         0.01 rf_aipw
##     ATE lower_ci upper_ci prop_dropped  model
## 1 -0.31   -0.327   -0.292         0.02 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.382     -0.4   -0.364         0.02 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.315   -0.333   -0.297         0.04 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.377   -0.396   -0.357         0.04 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.313   -0.331   -0.294          0.1 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.373   -0.393   -0.354          0.1 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.322   -0.342   -0.302          0.2 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.386   -0.407   -0.365          0.2 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.319    -0.34   -0.298          0.3 rf_ipw
##     ATE lower_ci upper_ci prop_dropped   model
## 1 -0.39   -0.412   -0.368          0.3 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.301   -0.323   -0.279          0.4 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.373   -0.397   -0.349          0.4 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.318   -0.343   -0.292          0.5 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.353   -0.383   -0.323          0.5 rf_aipw
```

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




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



Now, without cross fitting

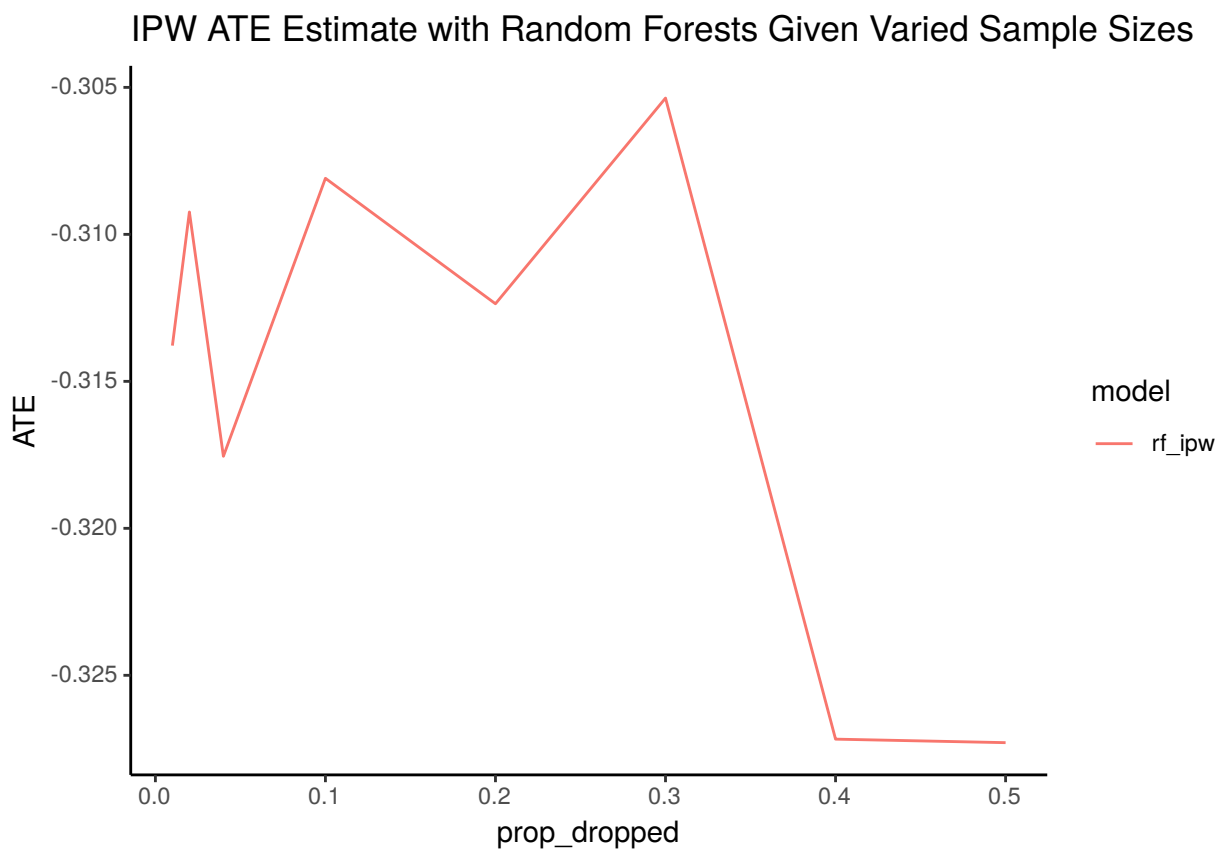
```
tauhat_rf_list_ncf <- data.frame()
tauhat_ols_rf_aipw_list_ncf <- data.frame()

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 = causal_forest(Xmod_temp, Wmod_temp, num.trees = 500)
  #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)
  pW_rf_temp.int.fit = causal_forest(Xmod_temp.int, Ymod_temp, Wmod_temp, num.trees = 500, compute.oob.
  # 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"

  print(tauhat_rf_temp.int)
  print(tauhat_ols_rf_aipw_temp.int)
  tauhat_rf_list_ncf <- rbind(tauhat_rf_list_ncf, tauhat_rf_temp.int)
  tauhat_ols_rf_aipw_list_ncf <- rbind(tauhat_ols_rf_aipw_list_ncf, tauhat_ols_rf_aipw_temp.int)
}
```

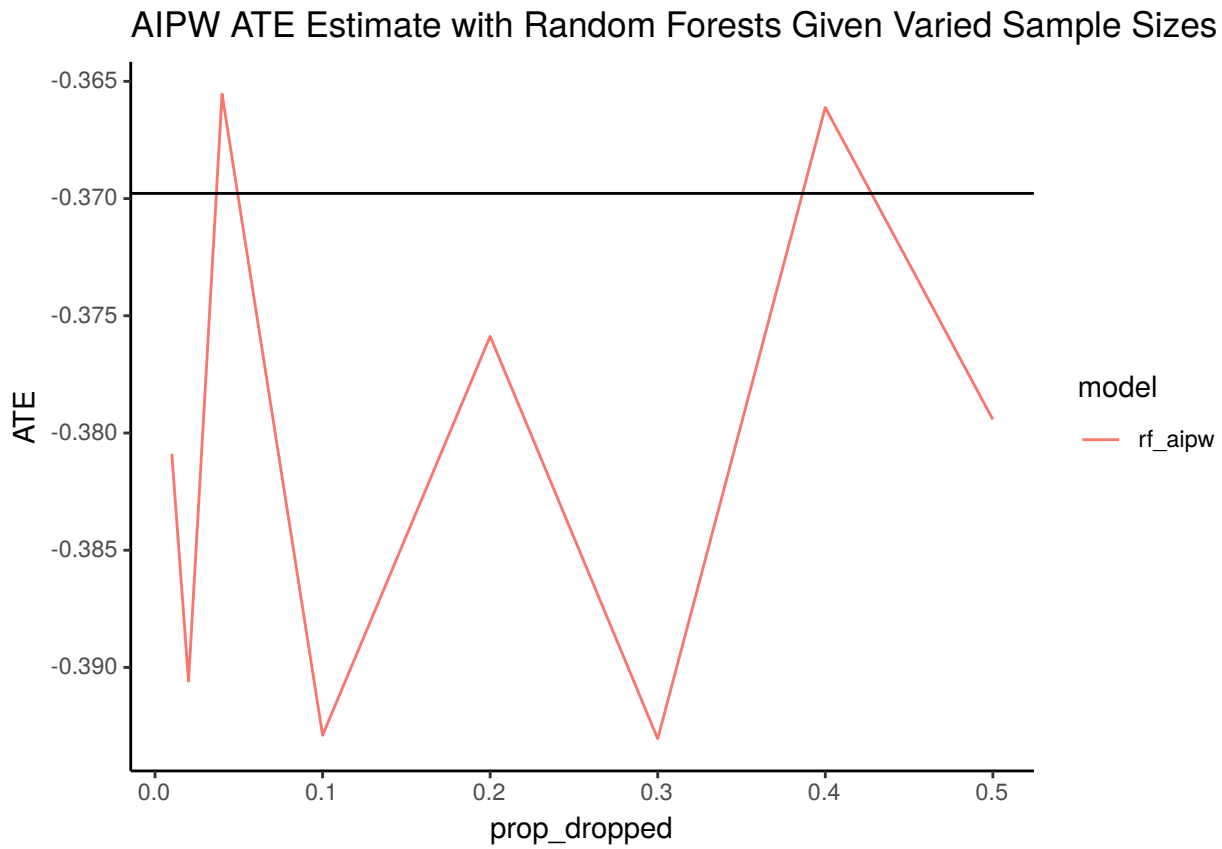
```
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.314   -0.331   -0.296         0.01 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.381   -0.401   -0.361         0.01 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.309   -0.327   -0.292         0.02 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.391   -0.413   -0.369         0.02 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.318   -0.335     -0.3         0.04 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.366   -0.386   -0.345         0.04 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.308   -0.326    -0.29          0.1 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.393   -0.412   -0.374          0.1 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.312   -0.332   -0.293          0.2 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.376   -0.396   -0.355          0.2 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.305   -0.326   -0.285          0.3 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.393   -0.415   -0.372          0.3 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.327    -0.35   -0.304          0.4 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.366   -0.391   -0.342          0.4 rf_aipw
##      ATE lower_ci upper_ci prop_dropped  model
## 1 -0.327   -0.352   -0.303          0.5 rf_ipw
##      ATE lower_ci upper_ci prop_dropped   model
## 1 -0.379   -0.408    -0.35          0.5 rf_aipw
```

```
#tauhat_rf_list <- rbind(tauhat_rf_list, tauhat_ols_rf_aipw_list)

ggplot(tauhat_rf_list_ncf, aes(x = prop_dropped, y = ATE, color = model)) + geom_line() +
```

```
  ggtitle("IPW ATE Estimate with Random Forests Given Varied Sample Sizes") +
  geom_abline(aes(slope = 0, intercept = tauhat_rct["ATE"]))
```




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



We notice that cross fitting improves ATE estimates with increase in size, as the estimates are significantly off and ustable without cross fitting for especially AIPW estimator The AIPW estimator is almost always better than the IPW estimator in this experiment

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

# 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
# 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

We now show ATE estimates and Confidence Intervals across models. AIPW with forest methods provides the most accurate measurements with small confidence intervals. This is true for both the original as well as interacted data

```
##                                    ATE lower_ci upper_ci ci_length
## RCT_gold_standard               -0.370   -0.384   -0.355     0.029
## naive_observational             -0.280   -0.294   -0.265     0.029
## linear_regression               -0.366   -0.392   -0.340     0.052
## propensity_weighted_regression -0.369   -0.396   -0.341     0.055
## IPW_logistic                    -0.370   -0.408   -0.332     0.076
## AIPW_linear_plus_logistic       -0.366   -0.393   -0.340     0.053
## IPW_forest                      -0.250   -0.278   -0.222     0.056
## AIPW_forest                     -0.377   -0.405   -0.349     0.056
## AIPW_linear_plus_forest         -0.372   -0.391   -0.352     0.040
## IPW_lasso                       -0.381   -0.421   -0.341     0.080
```

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             -0.280   -0.294   -0.265     0.029
## linear_regression               -0.375      NaN      NaN       NaN
## propensity_weighted_regression -0.366   -0.402   -0.330     0.073
## IPW_logistic                    -0.386   -0.434   -0.339     0.096
```

```
## AIPW_linear_plus_logistic      -0.366   -0.395   -0.337     0.059
## IPW_forest                     -0.310   -0.327   -0.292     0.034
## AIPW_forest                    -0.373   -0.400   -0.346     0.054
## AIPW_linear_plus_forest        -0.380   -0.398   -0.362     0.036
## IPW_lasso                      -0.404   -0.447   -0.360     0.086
```

Part Two

Why does propensity stratification work? We need the unconfoundedness assumption, that is the potential outcomes are uncorrelated with the treatment assignments given the pretreatment covariates In Propensity Score weighting, we deduce that for units with the same propensity score, once we know the propensity score, the unconfoundedness assumption holds. In other words, instead of equalising on the covariates, we can equalise on the propensity score given covariates. The idea is that even if the covariates are different, but the propensity score of treatment is the same, the latter is a complete statistic summarising all the confounding due to covariates. Put differently and simplistically, if there are two different Xs, if their propensity of treatment is the same, then we are again in the completely randomized design setup where the probability of treatment is independent of covariates (assume there are only these two covariates) As such, stratification on propensity scores resembles a high dimensional version of a stratified random experiment, where the strata are formed over higher dimensions using the Propensity Score, learned from some Machine Learning predictive model. Now, as we grow the number of samples, we should improve upon our estimates even with the same number of strata. If we increase the number of strata, we do finer stratification, reducing inter strata differences as in a stratified random design. Hence, we reduce bias, tending to no bias, as we increase the number of strata with more number of points. More points are required for more strata because otherwise, the effects we calculate in each strata might be too noisy with too few points.

```
#### PROPENSITY STRATIFICATION FUNCTION ####
```

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.

What do we do if some strata has only treatment or only control or no units? We exclude it from our calculations entirely

Note that the true effect is

```
n = 1000; p = 20
X = matrix(rnorm(n * p), n, p)
Y1 = pmax(X[,1] + X[,2], 0)
Y0 = pmax(X[,1], 0)
mean(Y1 - Y0)
```

```
## [1] 0.191
```

```
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

  # 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

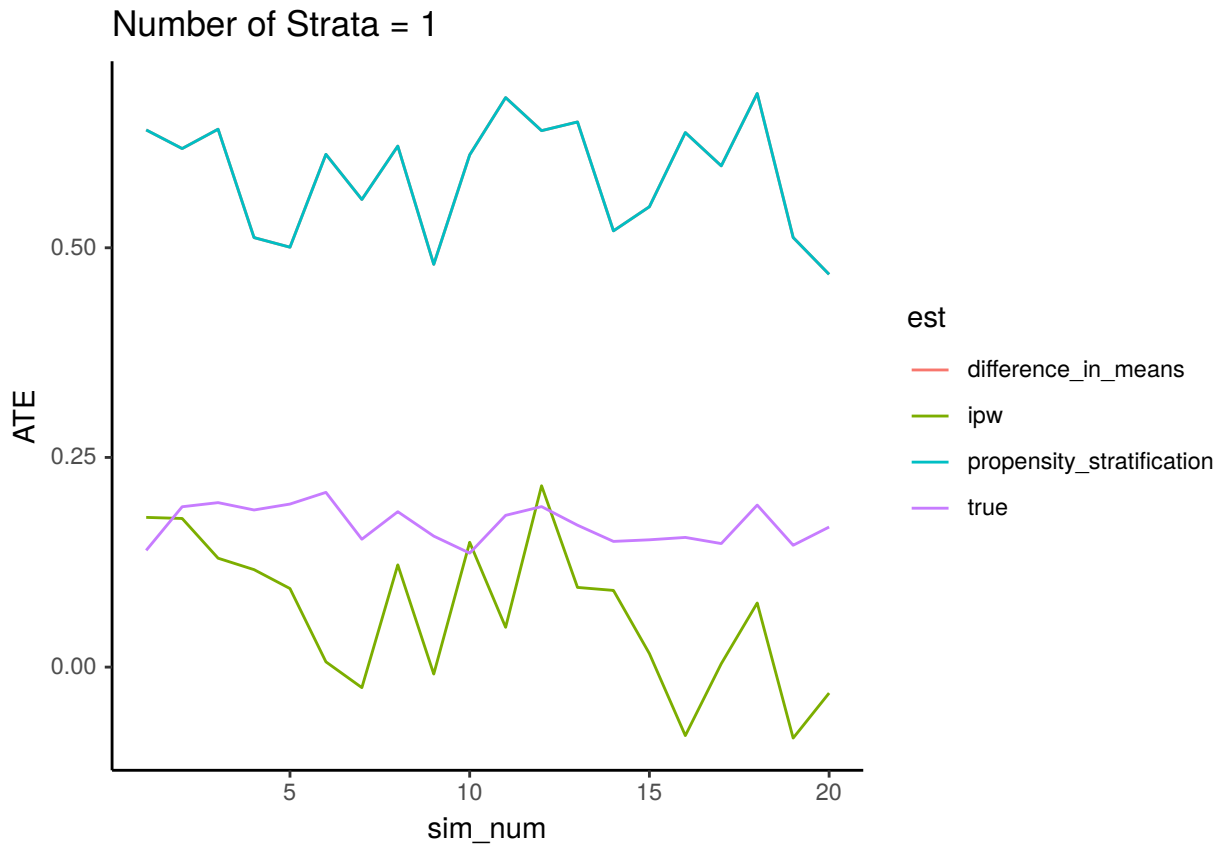
For all discussion that follows, we use the following code to generate a simulated dataset:

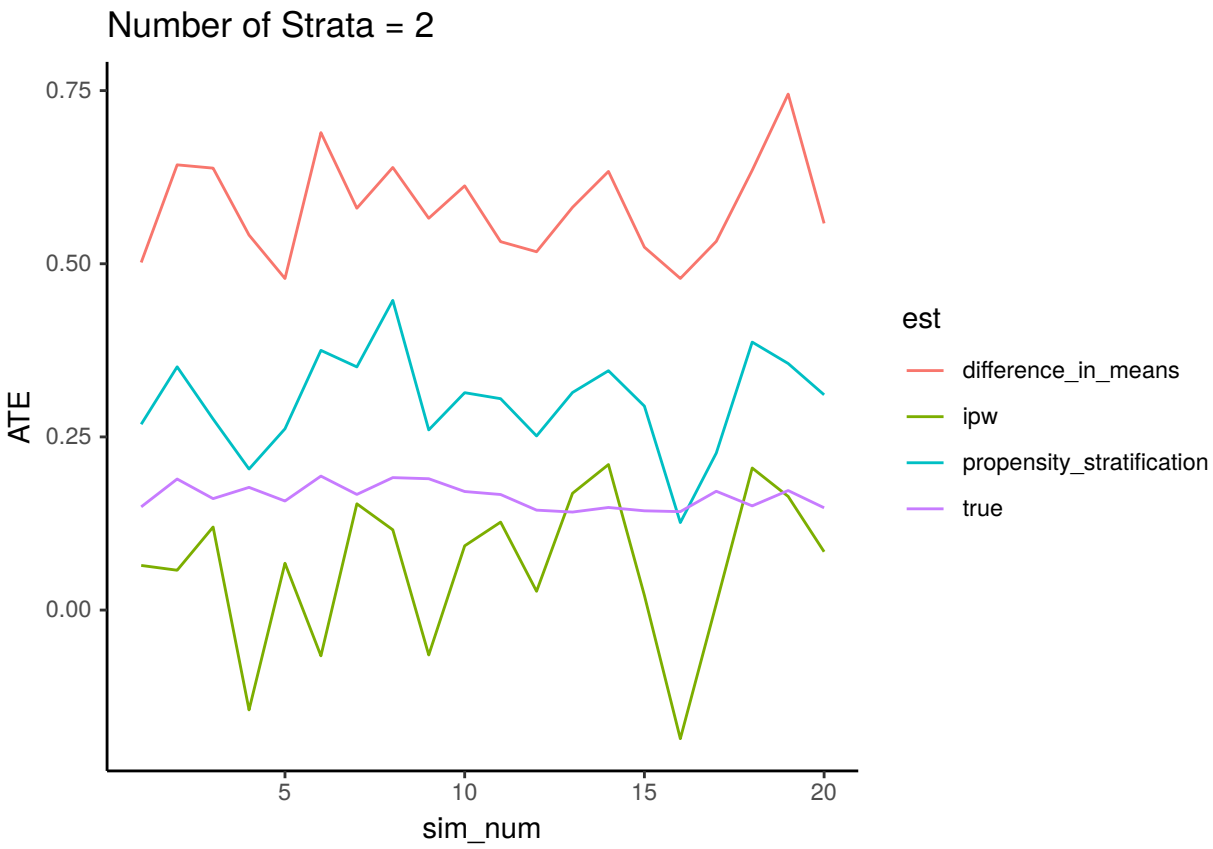
```
make_simulation <- function(){
  n = 1000; p = 20
  X = matrix(rnorm(n * p), n, p)
  propensity = pmax(0.2, pmin(0.8, 0.5 + X[,1]/3))
  W = rbinom(n, 1, propensity)
  Y = pmax(X[,1] + W * X[,2], 0) + rnorm(n)
  df = cbind(X, W, Y) %>% as.data.frame() %>% setDT
  df
}
```

```
# df <- make_simulation()
#
# difference_in_means(df)
#
# tauhat_propensity_stratification <- propensity_stratification(df_mod, pW_logistic.fit)
# tauhat_propensity_stratification
# # to get identical behavior (to be able to supply predictions to newdata for the propensity score mod
# # pW_logistic <- predict(pW_logistic.fit, newdata = df_mod, type = "response")
# tauhat_logistic_ipw <- ipw(df_mod, pW_logistic)
#
#
# sim_pW_logistic.fit <- glm(W ~ ., data = df %>% select(-Y), family = "binomial")
# sim_pW_logistic <- predict(sim_pW_logistic.fit, type = "response")
#
```

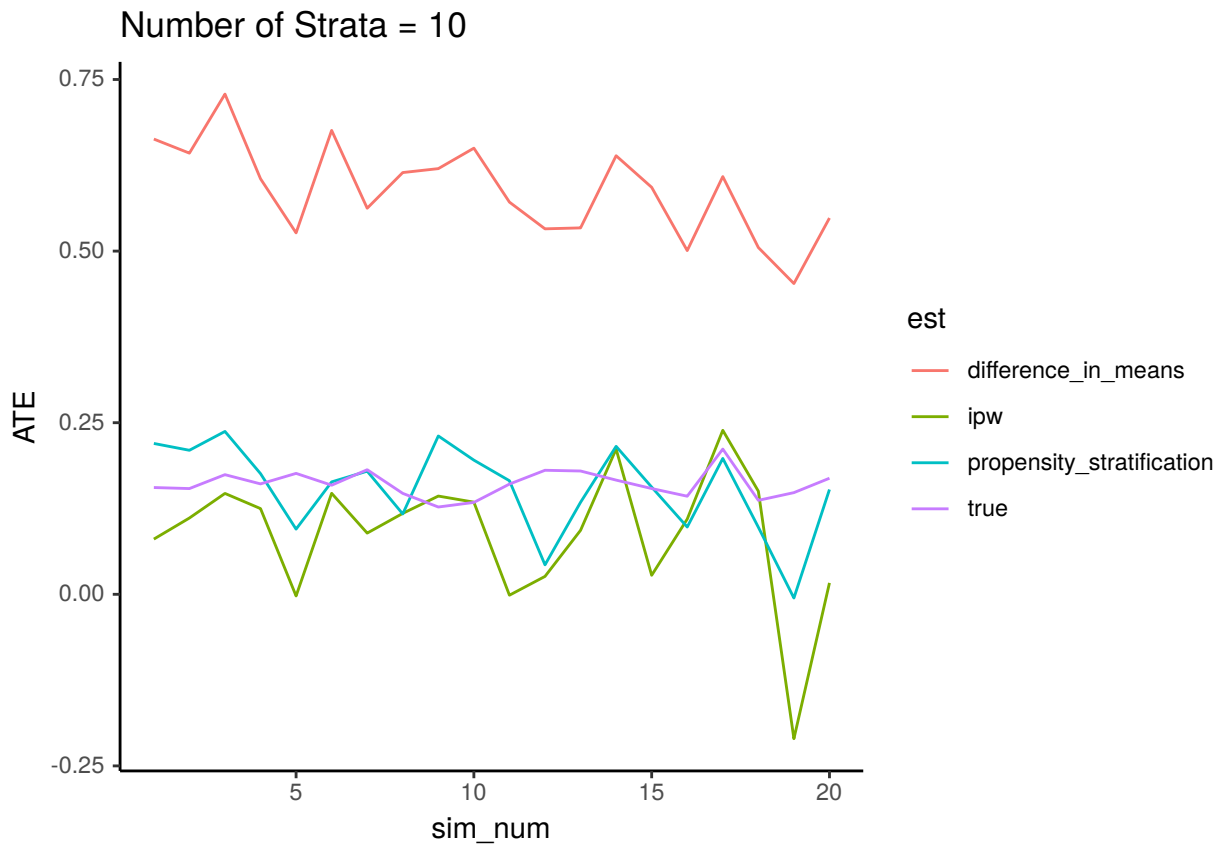
```
# difference_in_means(df)
# ipw(df, sim_pW_logistic)
# propensity_stratification(df, sim_pW_logistic.fit)
```

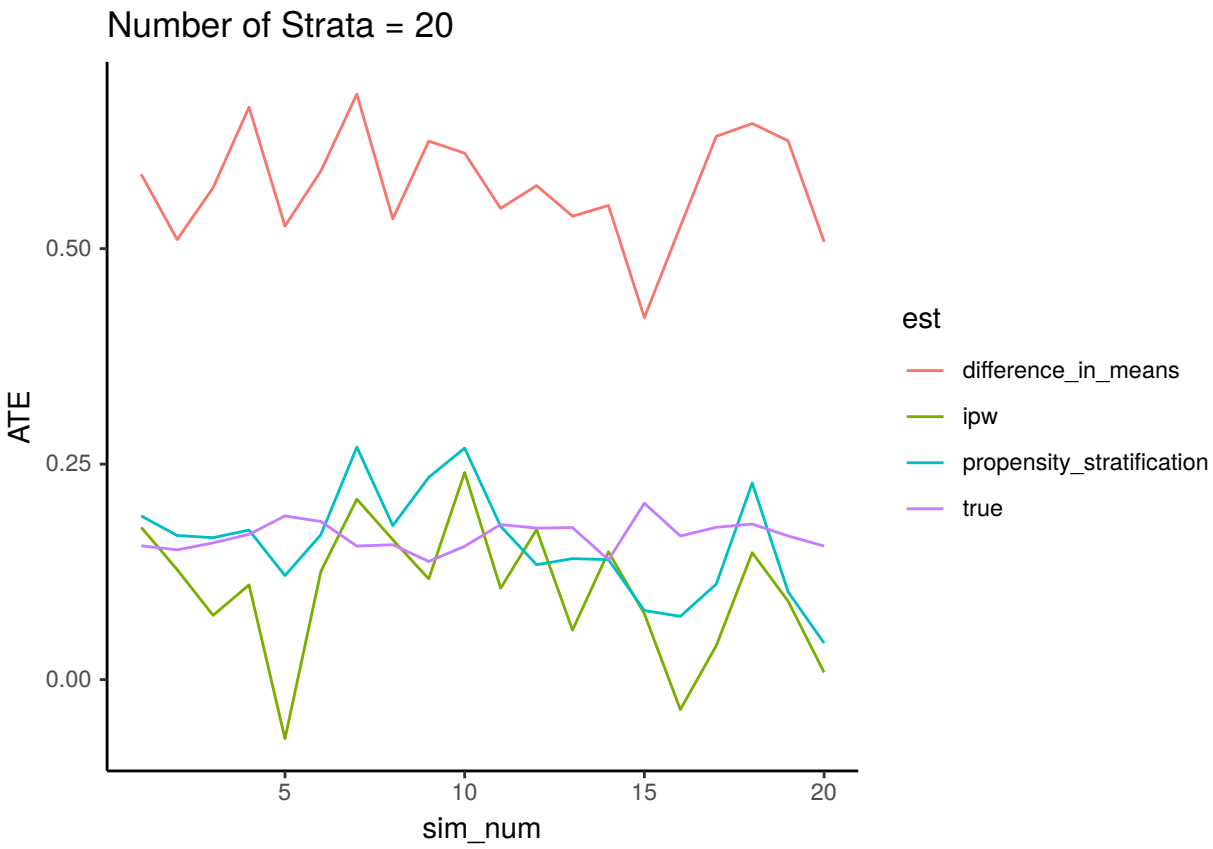
We now run 20.0 simulations of the type described above, and report results over each simulation.
We also run these simulations for different number of strata We see that propensity stratification and IPW perform similarly, but propensity stratification has lower average MSE.
For 1 stratum, we exactly track the difference in means estimator As we increase the number of strata, we start approaching the ipw estimator. Bias falls. We settle on 10 as the ideal number of strata. The reason is that it provides ATE with the least bias. At a high level, having too many strata increases the amount of noise with a fixed number of points, so we start at 1 and reach an optimum

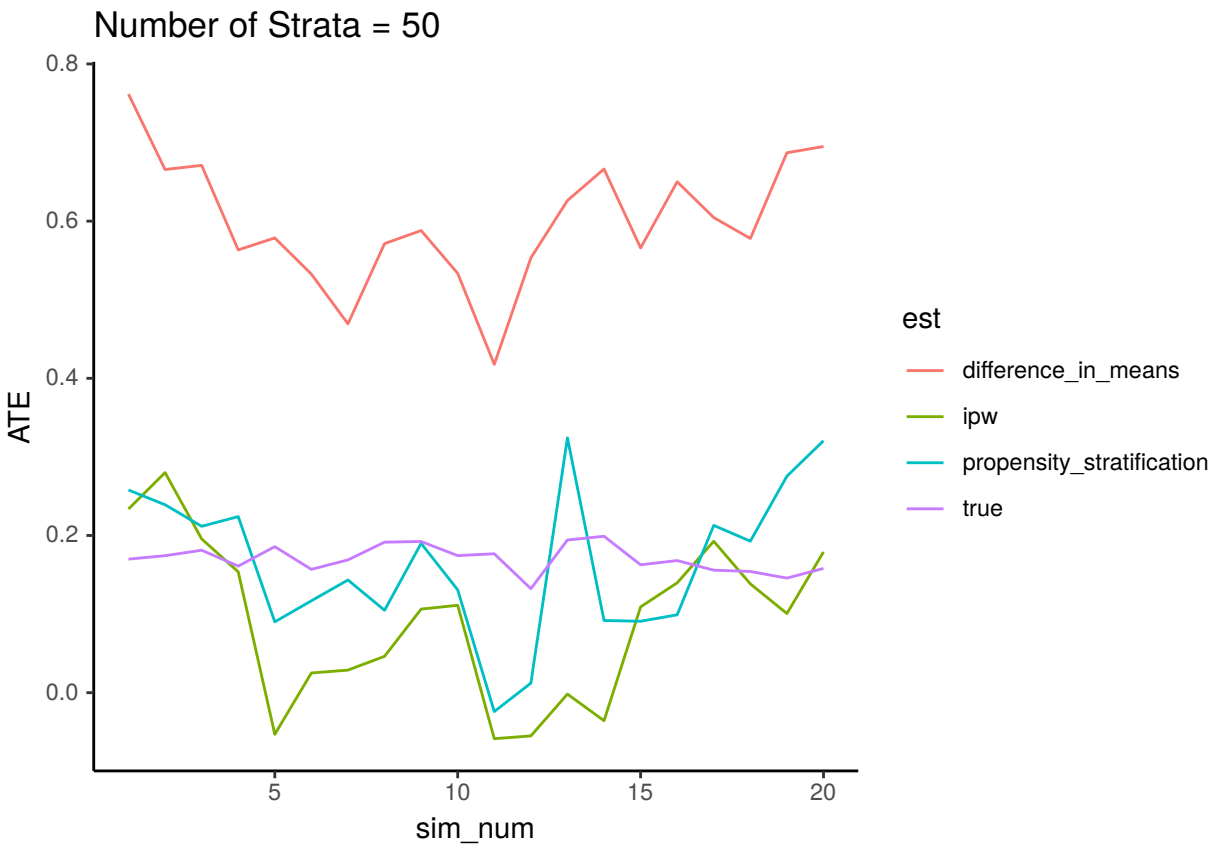



Number of Strata = 2
0.75
0.50
0.25
0.00
ATE
5
10
15
20
sim_num
est
difference_in_means
ipw
propensity_stratification
true


Number of Strata = 5
0.75
0.50
0.25
0.00
ATE
5
10
15
20
sim_num
est
difference_in_means
ipw
propensity_stratification
true


Number of Strata = 10
0.75
0.50
0.25
0.00
-0.25
ATE
5
10
15
20
sim_num
est
difference_in_means
ipw
propensity_stratification
true


Number of Strata = 20
ATE
0.50
0.25
0.00
5
10
15
20
sim_num
est
difference_in_means
ipw
propensity_stratification
true


Number of Strata = 50
0.8
0.6
0.4
0.2
0.0
ATE
5
10
15
20
sim_num
est
difference_in_means
ipw
propensity_stratification
true