6-3 Matching Methods - Solutions

March 12, 2024

```r
# libraries
xfun::pkg_attach2(c("tidyverse", # load all tidyverse packages
                    "here",      # set file path
                    "MatchIt",   # for matching
                    "optmatch",  # for matching
                    "cobalt"))   # for matching assessment


# chunk options ----------------------------------------------------------------
knitr::opts_chunk$set(
  warning = FALSE            # prevents warning from appearing after code chunk
)

# prevent scientific notation
# ----------
options(scipen = 999)
```

As we saw in last week's lab, an important advantage of randomized experiments are that they allow researchers to ensure independence between the exposure variable and other covariates, or rather that treatment and control groups have similar covariate distributions and differ only randomly.

The same cannot be said of observational studies, *no matter how large the sample size.* Thus, researchers often use a variety of matching methods to try to replicate this matching of covariate distributions between exposure groups.

In this lab we will consider some of these matching methods. Note that these methods are all implemented in the analysis stage (i.e. after the study has already been completed), and are distinct from (though may be similar to) methods of conducting studies which are matched from the outset.

Furthermore, matching should **not** be seen as an alternative to modeling adjustments such as regression, but instead are often used together.

Simulation

We will again use the simulated example from last week assessing the effectiveness of AspiTyleCedrin at treating migraines. As a reminder, this dataset contained the following variables:

- `A`: Treatment variable indicating whether individual i:
  - **DID** take AspiTyleCedrin ($A_i = 1$)
  - **DID NOT** take AspiTyleCedrin ($A_i = 0$)
- `Y_obs`: Outcome variable indicating whether individual i:
  - **DID** experienced a migraine ($Y_{i_{obs}} = 1$)
  - **DID NOT** experience a migraine ($Y_{i_{obs}} = 0$)
- `W1`: Variable representing sex assigned at birth:
  - $W1 = 0$ indicating AMAB (assigned male at birth)
  - $W1 = 1$ indicating AFAB (assigned female at birth)
  - $W1 = 2$ indicating an X on the birth certificate, intersex individual, or left blank

- `W2`: Variable representing simplified racial category:
  - $W2 = 0$ indicating White
  - $W2 = 1$ indicating Black or African American
  - $W2 = 2$ indicating Non-White Hispanic or Latinx
  - $W2 = 3$ indicating American Indian or Alaska Native
  - $W2 = 4$ indicating Asian
  - $W2 = 5$ indicating Native Hawaiian or Other Pacific Islander

Say that there is concern among providers that AspiTyleCedrin may be less effective among individuals with a higher Body Mass Index (BMI). To simulate this, we will modify the code we used to create the original AspiTyleCedrin dataset to also include the variable `W3` representing an individual's BMI. (We'll also modify the treatment and observed outcomes to be confounded by this variable.)

```r
# set seed
# ----------
set.seed(42) # set so that random process of generating data is reproducible

# set the number of individuals for simulated dataset
# ----------
n = 1e4 # Number of individuals (smaller than last time)

# NOTE: Again, don't worry too much about how we're creating this dataset,
# this is just an example.

# W3 scaled to have mu=24 and sigma=4 a la
# https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4789291/
# where k = mu^2/sigma^2 and theta = sigma^2/mu

# Also make treatment less likely so that there are more controls,
# and add ID column
df <- data.frame(ID = seq.int(n),
                 W1 = sample(0:2, size = n, replace = TRUE,
                             prob = c(0.49,0.50,0.01)),
                 W2 = sample(0:5, size = n, replace = TRUE,
                             prob = c(0.60,0.13,0.19,0.06, 0.015, 0.005)),
                 W3 = rgamma(n,
                    shape = 36,
                    scale = (2/3)))
df <- df %>%
  mutate(W3 = W3 + 8*(W1 == 1)+ 12*(W2==2) +
           8*(W2==3) + 4*(W2==4) + (-4)*(W2 == 5),
         A = as.numeric(rbernoulli(n,
                                   p = (0.16 + 0.07*(W1 > 0) + 0.21*(W2 == 0) -
                                          0.1*(W3 > 25) ))),
         Y_0 = as.numeric(rbernoulli(n,
                                     p = (0.87 + 0.035*(W1 > 0) + 0.05*(W2 > 0)) +
                                       abs((W3 - 22)/100))),
         Y_1 = as.numeric(rbernoulli(n,
                                     p = (0.34 + 0.035*(W1 > 0) + 0.3*(W2 > 0)) +
                                       abs((W3 - 22)/100) + 0.2*(W3 > 30))),
         ITE = Y_1 - Y_0,
         Y_obs = as.numeric((A & Y_1) | (!A & Y_0)))

ATE_true <- mean(df$ITE)
df_a1 <- df %>% filter(A == 1)
```

```r
ATT_true <- mean(df_a1$ITE)

df <- df %>% select(-Y_0, -Y_1, -ITE)
df_a1 <- df_a1 %>% select(-Y_0, -Y_1, -ITE)
df_a0 <- df %>% filter(A == 0)

head(df)
```

```
##   ID W1 W2       W3 A Y_obs
## 1  1  0  0 28.51215 0     1
## 2  2  0  2 34.91706 0     1
## 3  3  1  1 31.11242 0     1
## 4  4  0  0 26.56770 0     1
## 5  5  0  2 29.66014 0     1
## 6  6  0  2 38.53180 0     1
```

```r
summary(df)
```

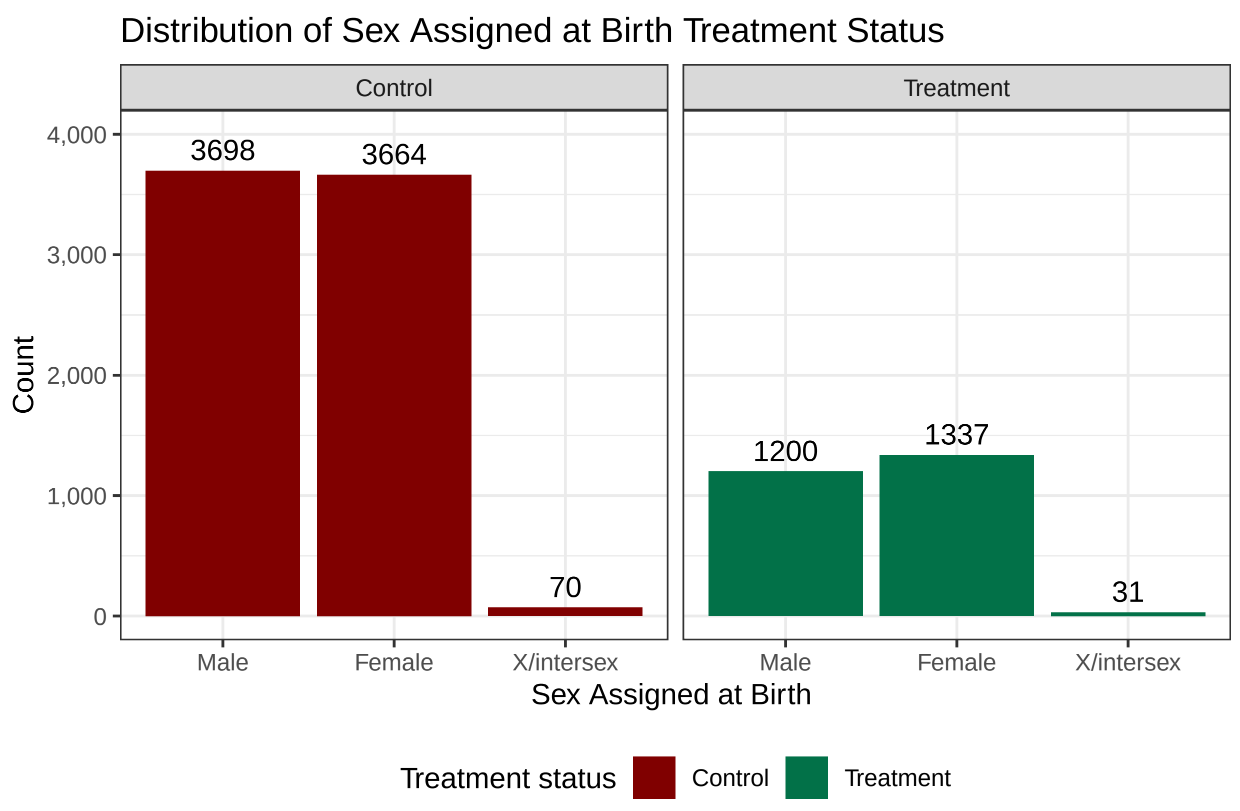
```
##        ID              W1               W2               W3
##  Min.   :    1   Min.   :0.0000   Min.   :0.0000   Min.   :12.84
##  1st Qu.: 2501   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:25.25
##  Median : 5000   Median :1.0000   Median :0.0000   Median :30.44
##  Mean   : 5000   Mean   :0.5203   Mean   :0.7677   Mean   :30.79
##  3rd Qu.: 7500   3rd Qu.:1.0000   3rd Qu.:2.0000   3rd Qu.:35.55
##  Max.   :10000   Max.   :2.0000   Max.   :5.0000   Max.   :57.38
##        A              Y_obs
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:1.0000
##  Median :0.0000   Median :1.0000
##  Mean   :0.2568   Mean   :0.8666
##  3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000
```

Let's take a look at the covariate distributions, comparing those that did and did not take AspiTyleCedrin:

Sex Assigned at Birth (SAAB)

For this chunk, there is extra `ggplot` code that illustrates how you might customize a figure for publication. There is a lot more you can do, so be sure to delve into the `ggplot` documentation to see all that is possible.

```r
#
# treatment status by sex
# --------------------------------------------------
df %>%

  # processing
  # ----------
  mutate(sex = case_when(W1 == 0 ~ "Male",        # assigned male at birth
                         W1 == 1 ~ "Female",      # assigned female at birth
                         W1 == 2 ~ "X/intersex"), # an X on the birth certificate,representing an inter
         sex = fct_relevel(sex, "Male", "Female", "X/intersex"),
         treatment = case_when(A == 0 ~ "Control",
                               A == 1 ~ "Treatment")
         ) %>%

  # plot
  # ----------
  ggplot(aes(x = sex, fill = treatment)) +
    # create a bar plot using geom_bar()
    geom_bar() +
    geom_text(stat = "count", aes(label = ..count..), # calculate count and pass to label parameter usi
              vjust = -0.5) +                         # vjust to add space between bar and text

    # facet grid controls number of panels - prefer this to facet_wrap
    facet_grid(
               #rows= vars(treatment), # facet variable in the rows
               cols = vars(treatment)  # facets variable in the column
               ) +
   # theme
   theme_bw() +                        # set base black and white theme
   theme(legend.position = "bottom") + # theme functions manipulate different elements of the plots app


   # scales
   scale_fill_manual(values=c("#800000","#027148")) +               # assign colors using hex code
   scale_y_continuous(breaks=seq(0, 4000, 1000),                    # y axis floor, ceiling, step
                      labels = scales::label_number(scale = 1,       # scale the variable
                                                    accuracy = 1,    # decimal points
                                                    big.mark = ",", # add "," or "."
                                                    prefix = "",     # add "$"
                                                    suffix = ""),    # add suffix, e.g., "%" or "k"
                      limits = c(0, 4000)) +                         # set floor and ceiling
    # labels
    labs(x = "Sex Assigned at Birth ",  # x-axis label
           y = "Count",                   # y-axis label
           fill = "Treatment status",     # legend label
         caption = "Note: ",            # add a caption
         title = "Distribution of Sex Assigned at Birth Treatment Status") # title
```



Note:

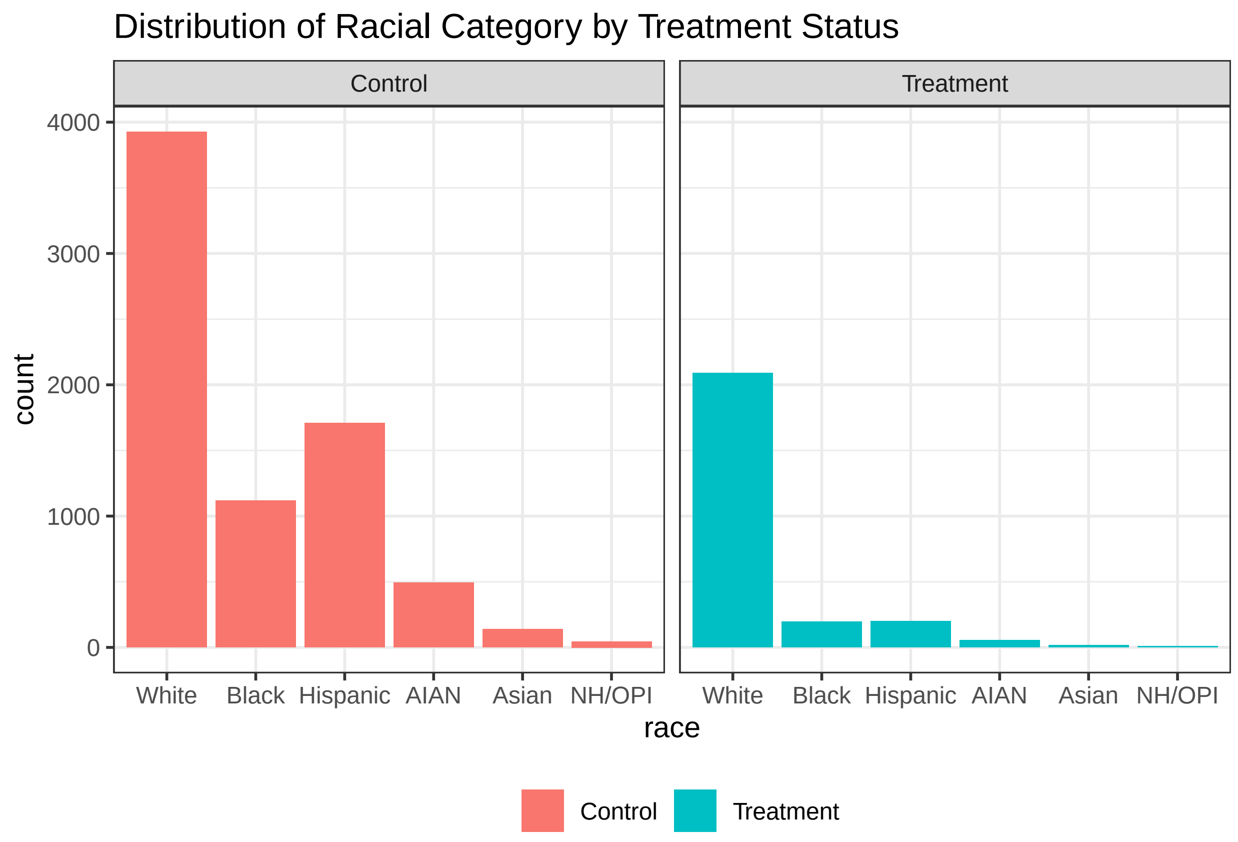
```
# chi-squared to test difference
# ----------
chisq.test(table(df$A, df$W1))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(df$A, df$W1)
## X-squared = 7.8191, df = 2, p-value = 0.02005
```

The bar plot above clearly shows a difference in the distribution of SAAB among the two groups, and this is confirmed by the very small p-value from the χ^2 test.

```
#
# treatment status by race
# --------------------------------------------------
df %>%

  # processing
  # ----------
  mutate(race = case_when(W2 == 0 ~ "White",    # non-Hispanic White
                          W2 == 1 ~ "Black",    # non-Hispanic Black
                          W2 == 2 ~ "Hispanic", # Latinx
                          W2 == 3 ~ "AIAN",     # American Indian or Alaska Native
                          W2 == 4 ~ "Asian",    # Asian
                          W2 == 5 ~ "NH/OPI"),  # Native Hawaiian or Other Pacific Islander
         race = fct_relevel(race, "White", "Black", "Hispanic", "AIAN", "Asian", "NH/OPI"), # relevel f
```

```
        treatment = case_when(A == 0 ~ "Control",
                              A == 1 ~ "Treatment")
        ) %>%

  # plot
  # ----------
  ggplot(aes(x = race, fill = treatment)) +
  geom_bar() +
  facet_grid(cols = vars(treatment)) +  # facets variable in the column

  # theme
   theme_bw() +                         # set base black and white theme
   theme(legend.position = "bottom") + # theme functions manipulate different elements of the plots app

  # labels
    labs(title = "Distribution of Racial Category by Treatment Status",
        fill = "")
```



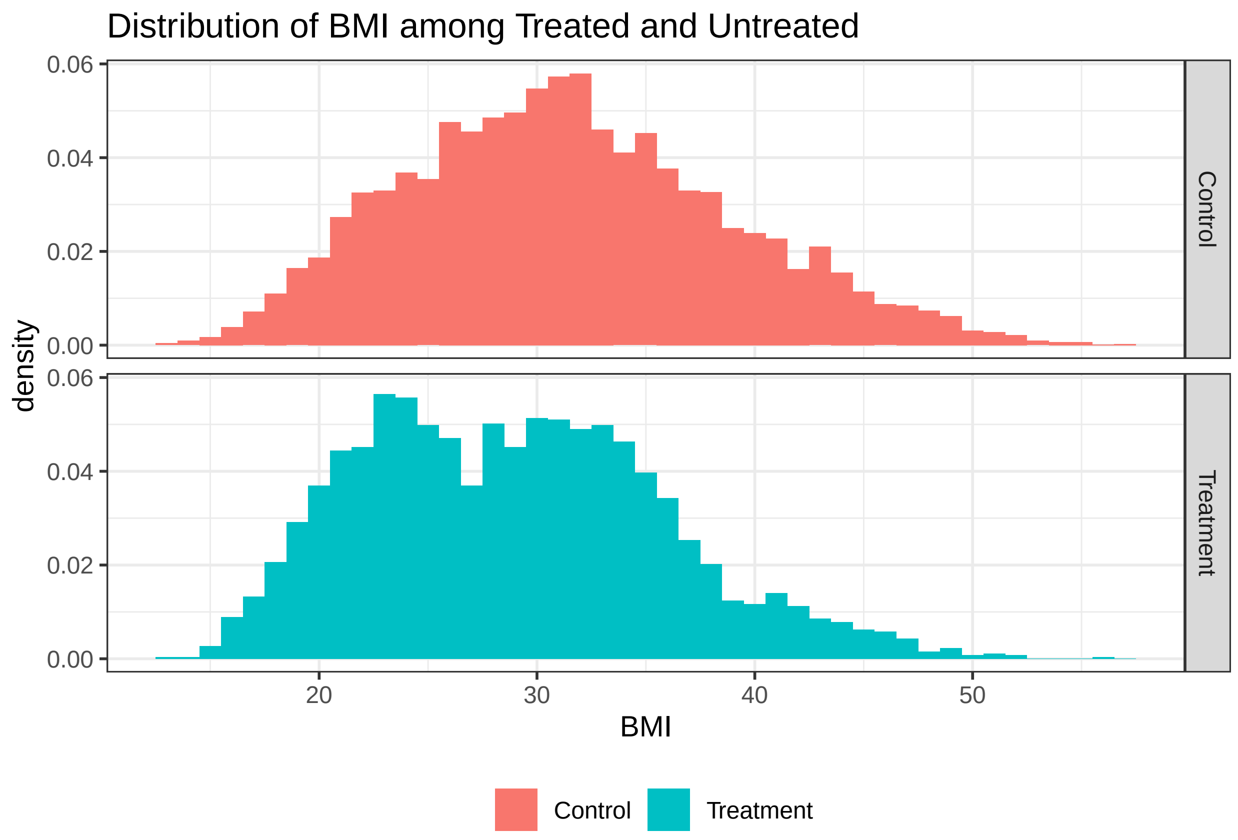
```
# chi-squared to test difference
# ----------
chisq.test(table(df$A, df$W2))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(df$A, df$W2)
## X-squared = 661.27, df = 5, p-value < 0.00000000000000022
```

The bar plot above again shows a difference in the distribution of simplified racial category among the two groups, and this is again confirmed by the very small p-value from the χ^2 test. You can find more documentation for the plotting parameters here.

Finally, we can use `geom_hist` to view the distribution of BMI by treatment status, which is a continuious variable.

```
#
# treatment status by BMI
# --------------------------------------------------
df %>%

  # processing
  # ----------
  mutate(treatment = case_when(A == 0 ~ "Control",
                               A == 1 ~ "Treatment")
         ) %>%
  # plot
  # ----------
  ggplot(aes(x = W3, fill = treatment)) +
  geom_histogram(binwidth = 1, aes(y = ..density..)) +
  facet_grid(rows = vars(treatment)) +  # facets variable in the column

  # theme
   theme_bw() +                        # set base black and white theme
   theme(legend.position = "bottom") + # theme functions manipulate different elements of the plots app

  labs(title = "Distribution of BMI among Treated and Untreated",
       x = "BMI",
       fill = "")
```



```
# t-test
# ---------
t.test(W3 ~ A, data = df)
```

```
##
##  Welch Two Sample t-test
##
## data:  W3 by A
## t = 15.686, df = 4735.2, p-value < 0.00000000000000022
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  2.243671 2.884605
## sample estimates:
## mean in group 0 mean in group 1
##        31.45203        28.88789
```

While it may be difficult to determine from the histogram above how the distribution of BMI differs among the two groups, the very small p-value from the t-test shows evidence of a clear difference.

Thus we can see the need to improve the matching of these covariate distributions.

Matching Considerations

There are a number of factors to consider when choosing a matching method, including the following:

- Distance Metric
- Greediness

- Control:Treatment Ratio
- Caliper Width
- Replacement
- Estimand

Distance Metric

The goal of matching is to match together each treatment unit (in our case, each individual who took AspiTyleCedrin, `A == 1`) to one or more "control" unit (in our case, individuals who did not take AspiTyleCedrin, `A == 0`) based on baseline covariates (in our case, `W1, W2, W3`). **Conceptually, this means we are trying to find the control unit(s) that most closely resemble the counterfactual for each treatment unit.**

Exact Matching

Ideally, we would like to find the control unit(s) which have all identical covariate values. This is called "exact matching".

For our dataset, this would mean each individual who took AspiTyleCedrin (`A == 1`) would be matched with individual(s) who did not take AspiTyleCedrin (`A == 0`) with the *exact* same SAAB (`W1`), racial category (`W2`), and BMI (`W3`).

In other words, the exact distance between two points X_i, X_j, where $X_i = \{W1_i, W2_i, W3_i\}$ and $X_j = \{W1_j, W2_j, W3_j\}$ is defined as:

$$\text{Distance}(X_i, X_j) = \begin{cases} 0, & \text{if } X_i = X_j \\ \infty, & \text{if } X_i \neq X_j \end{cases}$$

Question 1: The data frame `df_a0` contains all the individuals that did not take AspiTyleCedrin, and the data frame `df_a1` contains all those who did. In the `R` code chunk below, use the first ten rows of `df_a0` and the first five rows of `df_a1` to find the exact distance of the first ten individuals who did not take AspiTyleCedrin from *each* of the first five individuals who did. (Hint: How many comparisons should you be making?)

```r
#
# calculate exact matches by hand
# --------------------------------------------------

# create dataframe
# ---------
df_a0_small <- df_a0[1:10,]
df_a1_small <- df_a1[1:5,]
cols <- c("W1", "W2", "W3")

# create function to only keep where observations are equal
# # ---------
dist.exact <- function(x,y) {
  ifelse(all(x == y), 0, NA) # NA means no match
}

# funciton to calculate distances
# ---------
calculate.dist <- function(x, y, dist.method, xnames = df_a1_small$ID, ynames = df_a0_small$ID) {
  dists <- apply(y, 1, function(j) {apply(x, 1, function(i) {dist.method(i,j)})})
  rownames(dists) <- xnames
  colnames(dists) <- ynames
  return(dists)
}

# apply to data and save as new object
# ---------
dists_ex <- calculate.dist(df_a1_small[, cols], # x
                           df_a0_small[, cols], # y
                           dist.exact)          # distance metric
dists_ex
```

```
##     1  2  3  4  5  6  7  8 10 12
## 9  NA NA NA NA NA NA NA NA NA NA
## 11 NA NA NA NA NA NA NA NA NA NA
## 15 NA NA NA NA NA NA NA NA NA NA
## 16 NA NA NA NA NA NA NA NA NA NA
## 17 NA NA NA NA NA NA NA NA NA NA
```

While exact matching is ideal, it is not always possible, such as in the case of continuous variables, such as our BMI variable, `W3`.

Question 2: Explain why matching on a continuous variable would likely be impossible.

The probability of any exact value of a continuous variable is by definition zero, so even taking rounding into account, the probability of finding exact matches on a continuous variable is very low.

Question 3: Modify your code above to only check the distance for `W1` and `W2` values.

```
# check again but omit W3 (BMI)
# ---------
dists_ex_lim <- calculate.dist(df_a1_small[, cols[1:2]], df_a0_small[, cols[1:2]], dist.exact)
dists_ex_lim
```

```
##     1  2  3  4  5  6  7  8 10 12
## 9   0 NA NA  0 NA NA  0 NA  0 NA
## 11 NA NA NA NA NA NA NA NA NA NA
## 15 NA NA NA NA NA NA NA NA NA NA
## 16 NA NA NA NA NA NA NA NA NA  0
## 17  0 NA NA  0 NA NA  0 NA  0 NA
```

Since exact matching is not always possible, there are a variety of alternative distance metrics which may be used to determine how similar a potential match is. A few of these methods are discussed below.

Mahalanobis Distance Matching

The Mahalanobis distance in general is a "multi-dimensional generalization of the idea of measuring how many standard deviations away [some point] P is from the mean of [some distribution] D." However, in the context of matching, the Mahalanobis distance measures this distance between the two points X_i, X_j rather than that between one point and a distribution.

Mathematically, this version of the Mahalanobis distance is defined as follows:

$$\text{Distance}(X_i, X_j) = \sqrt{(X_i - X_j)^T S^{-1} (X_i - X_j)}$$

where S^{-1} is the covariance matrix of X_i and X_j.

Question 4: Using the `cov()` function to find the covariance matrix of $\{W1, W2, W3\}$ from the *whole dataset*, modify your code from **Question 1** to instead find the Mahalanobis distance of the first ten individuals who did not take AspiTyleCedrin from *each* of the first five individuals who did. (Hint: The `t()` function will transpose a vector or matrix, and matrix multiplication uses the `%*%` character, not `*`)

```
#
# calculate Mahalanobis distance metric
# --------------------------------------------------

# calculate covariance matrix
# ---------
cov_df <- cov(df[,cols])

# create a function to calculate mahalanobis distance
# ---------
dist_mahalanobis <- function(x,y) {
  diff <- (x - y)                          # return the difference of x-matrix from y-matrix
  sqrt( t(diff) %*% cov_df %*% (diff) ) # transpose difference and multiply by the covariance and the d
}

# apply function to calculate Mahalanobis distance
# ---------
dists_ma <- calculate.dist(df_a1_small[, cols], # x
                           df_a0_small[, cols], # y
                           dist_mahalanobis)    # distance
# return
dists_ma
```

```
##            1         2        3         4        5         6        7        8
## 9  53.711777 102.53125 73.88371 39.210885 63.33835 129.48478 66.22732 192.1228
## 11 36.824702  12.16181 16.68337 51.324811 27.30501  38.99333 24.31058 101.5984
## 15  4.977753  53.79966 25.14157  9.554379 14.70509  80.74327 17.47859 143.3723
## 16 50.456454  99.25355 70.61490 35.959232 60.05258 126.20982 62.97019 188.8519
## 17 57.306452 106.12532 77.47812 42.805559 66.93147 133.07908 69.82200 195.7173
##           10       12
## 9   18.47621 78.78527
## 11 109.01085 11.81294
## 15  67.23230 30.04789
## 16  21.76174 75.51503
## 17  14.88153 82.37969
```

Propensity Score Matching

The propensity score of an individual is a measure of the probability of that individual receiving the treatment based upon the baseline covariates. That is, given a set of covariate values ($\{W1_i, W2_i, W3_i\}$ in our case), the propensity score represents the estimated probability of treatment ($A_i = 1$). The propensity score is often estimated using a logit model and is therefore defined as follows:

$\pi_i = P(A_i = 1|X_i) = \frac{1}{1+e^{-X_i\beta}}$

We can estimate these propensity scores using logistic regression, by regressing the treatment A on the baseline covariates X, like so:

```
#
# fit a logit model
# --------------------------------------------------
model_ps <-                    # save logit model as an object
```

```
  glm(A ~ W1 + W2 + W3,      # regress A (treatment) on covariates (W1, W2, W3)
      family = binomial(),   # specifying binomial calls a logit model
      data = df)             # specify data for regression

# print summary
summary(model_ps)
```

```
##
## Call:
## glm(formula = A ~ W1 + W2 + W3, family = binomial(), data = df)
##
## Coefficients:
##              Estimate Std. Error z value             Pr(>|z|)
## (Intercept)  0.015900   0.121459   0.131                0.896
## W1           0.358689   0.056395   6.360      0.0000000002014 ***
## W2          -0.531363   0.033945 -15.654 < 0.0000000000000002 ***
## W3          -0.031861   0.004817  -6.614      0.0000000000373 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11394  on 9999  degrees of freedom
## Residual deviance: 10709  on 9996  degrees of freedom
## AIC: 10717
##
## Number of Fisher Scoring iterations: 4
```

We can then use this model and the `predict()` function to add all of the estimated propensity scores for each data point in `df`:

```
# predict
# ---------
df <-                                    # save over df dataframe object
  df %>%                                 # pass data
  mutate(prop_score = predict(model_ps)) # create a new variable that predicts propensity score based o

# update the subsetted datasets - WOULD NOT DO THIS IN YOUR WORK
# ---------
df_a0 <- df %>% filter(A == 0) # save anything under control as a dataframe
df_a1 <- df %>% filter(A == 1) # save anything under treatment as a dataframe
df_a0_small <- df_a0[1:10,]    # further subsetting
df_a1_small <- df_a1[1:5,]     # further subsetting
```

Propensity score *matching* uses the absolute difference between two propensity scores as its distance metric, or rather:

$$\text{Distance}(X_i, X_j) = |\pi_i - \pi_j|$$

Question 5: Again modify your previous code to find the propensity score distance of the first ten individuals who did not take AspiTyleCedrin from *each* of the first five individuals who did.

```
# calculate distances based on propensity scores
# ---------
dist.prop.score <- function(x,y) {
  abs(x-y)  # distance based on absolute value
}

# apply function
# ---------
dists_ps <- calculate.dist(as.matrix(df_a1_small[, "prop_score"]), # x
                           as.matrix(df_a0_small[, "prop_score"]), # y
                           dist.prop.score)                         # method
# view
dists_ps
```

```
##            1         2          3         4         5        6         7
## 9  0.2294698 1.4962611 0.48498975 0.1675185 1.3287721 1.611429 0.2829393
## 11 0.2024718 1.4692631 0.45799176 0.1405205 1.3017741 1.584431 0.2559413
## 15 0.3809625 1.6477538 0.63648243 0.3190112 1.4802647 1.762922 0.4344320
## 16 0.3136261 0.9531652 0.05810617 0.3755774 0.7856761 1.068333 0.2601566
## 17 0.2448272 1.5116184 0.50034708 0.1828758 1.3441294 1.626786 0.2982966
##            8         10        12
## 9  1.5192731 0.07893488 0.8657146
## 11 1.4922751 0.10593288 0.8387166
## 15 1.6707658 0.07255779 1.0172072
## 16 0.9761772 0.62203081 0.3226186
## 17 1.5346305 0.06357755 0.8810719
```

Double Robustness A key advantage of propensity score matching is that, when used in conjunction with outcome regression, provides a "doubly robust" estimator. That is,

> "When used individually to estimate a causal effect, both outcome regression and propensity score methods are unbiased only if the statistical model is correctly specified. The doubly robust estimator combines these 2 approaches such that only 1 of the 2 models need be correctly specified to obtain an unbiased efficient estimator."

"Correctly specified" means that a model accurately represents the relationship between the variables. E.g. a linear model between x and y is correctly specified if and only if x and y truly do have a linear relationship to each other.

This means that only one of the two models (the model of treatment to covariates or the model of outcome to treatment and covariates) needs to accurately represent the relationships among the respective variables in order for the estimate to be unbiased.

Greediness

Once deciding upon a distance metric, we must also choose a matching algorithm. That is, how shall the computed distances be used to determine a match? The various matching algorithms fall into two general categories: "greedy" and optimal.

"Greedy" Matching

Greedy algorithms in general are used to reduce larger problems to smaller ones by taking the best option at the time and repeating, while never returning to earlier choices to make changes. In the context of matching, this means that a greedy matching algorithm chooses the best single match first and removes that chosen match. It then repeats this process by choosing the best single match still remaining and removing that match, and so on.

There are a number of different ways to decide which match to deem "best", including but not limited to:

- Choose the treatment participant with the highest propensity score first, and match it to the "control" participant with the closest propensity score (shortest propensity score distance).

- Same as above but start with lowest rather than highest propensity score.
- The best overall match (minimum of all match distances) in the entire dataset.
- Random selection.

Most greedy matching algorithms in common use (including those listed above) are “nearest neighbor” algorithms, which choose a treatment individual first and match to a control individual rather than the reverse.

Question 6: Using the propensity score distances you made in Question 5, find the greedy matching of this subset using highest to lowest propensity score. Report the IDs of both elements of each matched pair. (Hint: You may find the `which.min()` and `which.max()` functions helpful)

```
#
# use greedy matching - subset on highest to lowest propensity
# --------------------------------------------------

# create new datasets
# ---------
treat <- c()   # create empty treatment vector
control <- c() # create empty control vector
df_a1_small_copy <- as.data.frame(df_a1_small) # create a copy to prevent overwrite within cell
dists_ps_copy <- as.data.frame(dists_ps)       # create a copy to prevent overwrite within cell

# loop through to grab matches based on propensity scores
# ---------
for(i in 1:nrow(df_a1_small)) {
  max_treat <- which.max(df_a1_small_copy$prop_score)# %>% select(-ID)) # save max propensity score
  treat[i] <- names(max_treat)                                         # add max_treat names
  df_a1_small_copy <- df_a1_small_copy %>% slice(-max_treat)           # remove it from the dataframe

  match_control <- which.min(dists_ps_copy[max_treat,])                # find it's match in control
  control[i] <- names(all_of(match_control))                           # store names as control
  dists_ps_copy <- dists_ps_copy %>%                                   # drop what we have just select
      select(-match_control) %>%
      slice(-max_treat)
}

# print
# ---------
treat
```

```
## [1] "15" "17" "9"  "11" "16"
```

```
control
```

```
## [1] "10" "4"  "1"  "7"  "3"
```

Question 7: Same as Question 6, but now find the greedy matching of this subset using lowest to highest propensity score.

```
#
# use greedy matching - subset on lowest to highest propensity
# --------------------------------------------------


# create new datasets
# ---------
treat <- c()
```

```
control <- c()
df_a1_small_copy <- as.data.frame(df_a1_small)
dists_ps_copy <- as.data.frame(dists_ps)

# loop through to grab matches based on propensity scores
# ---------
for(i in 1:nrow(df_a1_small)) {
  min_treat <- which.min(df_a1_small_copy$prop_score)
  treat[i] <- names(min_treat)
  df_a1_small_copy <- df_a1_small_copy %>% slice(-min_treat)

  match_control <- which.min(dists_ps_copy[min_treat,])
  control[i] <- names(match_control)
  dists_ps_copy <- dists_ps_copy %>%
      select(-match_control) %>%
      slice(-min_treat)
}

# print
# ---------
treat
```

```
## [1] "16" "11" "9"  "17" "15"
```

```
control
```

```
## [1] "3"  "10" "4"  "1"  "7"
```

Question 8: Same as in the previous two problems, but now find the greedy matching of this subset using best overall match.

```
#
# use greedy matching - subset using best overall
# --------------------------------------------------

# create new datasets
# ---------
treat <- c()
control <- c()
dists_ps_copy <- as.data.frame(dists_ps)

# loop through to grab matches based on propensity scores
# ---------
for(i in 1:nrow(df_a1_small)) {
  best <- which(dists_ps_copy == min(dists_ps_copy), arr.ind = TRUE)
  treat[i] <- rownames(dists_ps_copy)[best[1]]
  control[i] <- colnames(dists_ps_copy)[best[2]]

  dists_ps_copy <- dists_ps_copy %>%
    slice(-(best[1])) %>%
    select(-(best[2]))
}

# print
# ---------
treat
```

```
## [1] "16" "17" "11" "9"  "15"
```

```
control
```

```
## [1] "3"  "10" "4"  "1"  "7"
```

Question 9: Were there any differences in the matchings you found in the previous three problems?

ANSWER: There were significant differences across each one, meaning the matching algorithm can have a big impact.

Optimal Matching

Optimal matching, as the name implies, seeks to find an optimal matching scheme in which the overall match difference is minimized. For example, if we were to add the distances of all match pairs chosen, an optimal matching would seek the set of match pairs which produces the smallest sum. A disadvantage of optimal matching is that it can be computationally intensive without providing sufficient improvements over greedy matching.

Control:Treatment Ratio

You may have noticed that in the previous examples we only selected one "control" individual for each treatment individual, often called $1:1$ matching. However, in some cases we may prefer to match more than one control to each treatment, often called $k:1$ matching, where k is the number of control individuals desired per treatment individual. (Note: while we are not considering them here, there are matching algorithms which discard treatment individuals rather than control individuals)

Question 10: Modify your code from Question 6 to perform a 2:1 matching rather than 1:1. That is, find the two best "control" matches for each treatment individual, using highest to lowest propensity score.

```r
#
# manual matching - using 2:1 ratio
# --------------------------------------------------
#
# create new datasets
# ---------
treat <- c()
control_1 <- c()
control_2 <- c()
df_a1_small_copy <- as.data.frame(df_a1_small)
dists_ps_copy <- as.data.frame(dists_ps)

# loop through to grab matches based on propensity scores
# ---------
for(i in 1:nrow(df_a1_small)) {
  max_treat <- which.max(df_a1_small_copy$prop_score)
  treat[i] <- names(max_treat)
  df_a1_small_copy <- df_a1_small_copy %>% slice(-max_treat)

  match_control_1 <- which.min(dists_ps_copy[max_treat,])
  control_1[i] <- names(all_of(match_control_1))
  dists_ps_copy <- dists_ps_copy %>% select(-match_control_1)

  if(ncol(dists_ps_copy) > 1) {
    match_control_2 <- which.min(dists_ps_copy[max_treat,])
    control_2[i] <- names(all_of(match_control_2))
    dists_ps_copy <- dists_ps_copy %>%
      select(-match_control_2) %>%
      slice(-max_treat)
  } else {
    control_2[i] <- names(dists_ps_copy)
  }

}

# print
# ---------
treat
```

```
## [1] "15" "17" "9"  "11" "16"
```

```
control_1
```

```
## [1] "10" "1"  "3"  "5"  "8"
```

```
control_2
```

```
## [1] "4"  "7"  "12" "2"  "6"
```

Question 11: Did any of the matches you made in Question 6 change in Question 10?

ANSWER: Yes.

It is also possible to have a variable number of control individuals per treatment individual in "full" matching. Full matching assures that every individual in the dataset is paired. Full matching can only by achieved using an optimal matching algorithm.

Caliper Width

As seen in $1:1$ and $k:1$ matching, some data may be pruned in favor of other priorities. We may also choose to prune data for which a sufficiently close match can be found. For this method we choose a threshold, or "caliper", and only consider matches whose distance is within this caliper width, discarding any individuals left unmatched.

Replacement

Another consideration when deciding upon a matching algorithm is whether matches are made with or without replacement. That is, can the same control individual be matched to more than one treatment individual. You may notice that so far we have only considered matching without replacement.

Question 12: Write code to perform the same greedy matching as in Question 6 but **with** replacement. (Hint: This code will likely be much simpler!)

```
#
# implement with replacement
# --------------------------------------------------

row_mins <- apply(dists_ps, 1, which.min)
treat <- names(row_mins)
control <- colnames(dists_ps)[row_mins]

# view
treat
```

```
## [1] "9"  "11" "15" "16" "17"
```

```
control
```

```
## [1] "10" "10" "10" "3"  "10"
```

Question 13: Compare these matches to those you found in Question 6.

> Your answer here.

Estimand

Depending on the matching algorithm used, you may be limited in whether it is possible to estimate the Average Treatment Effect (ATE) or the Average Treatment Effect on the Treated (ATT) only. For example, 1:1 nearest neighbor matching almost always estimates the ATT and cannot estimate the ATE.

Question 14: Briefly explain why 1:1 nearest neighbor matching may not be able to estimate the ATE.

It may not be able to estimate the ATE because it may not be able to find appropriate controls to match to the treatment.

Matching Algorithm Examples

As we've seen using our small subset of the data, implementing matching algorithms from scratch can be rather complex. Thankfully, we can use the `MatchIt` package which can implement many different matching algorithm variations for us.

The main `matchit()` function of this package includes the following arguments:

- `formula` : A formula object specifying the the treatment variable `A` and the covariates to be matched upon `X`, X2,`...in the following format:`A ~ X1 + X2 + ...`.
- `data` : The data frame.
- `method`: Matching method to be used. Options include (but are not limited to): "nearest" (i.e. Nearest Neighbor), "optimal", "full", "exact".
- `distance`: Distance metric to be used. Options include (but are not limited to): "glm" (e.g.Propensity score matching using a generalized linear model such as regression), "mahalanobis", a numeric vector containing already calculated distances.
- `link`: The link function used with the option chosen in `distance`. (e.g. "logit" if using logistic regression for propensity score matching)
- `estimand`: The value to be estimated. Options include (but are not limited to): "ATE", "ATT". Note that "ATE" is not available for all matching methods.
- `discard`: Which type of units may be discarded. Options are: "control" (i.e. most of the examples we have considered so far), "treatment", "none", "both".
- `replace`: Whether matching should be done with (`TRUE`) or without (`FALSE`) replacement.
- `caliper`: The caliper widths to use for each variable (if any) while matching.
- `ratio`: How many control units should be matched to each treatment unit.

Exact Matching Example

ATE

For example, for an exact matching on our dataset ignoring BMI we would do the following to estimate ATE:

```
#
# ATE using matchit for exact
# --------------------------------------------------
match_exact_ate <- matchit(formula = A ~ W1 + W2, # formula (leaving out W3 bc it is continuous)
                           data = df,             # data
                           method = "exact",      # specify method to use
                           estimand = "ATE")      # specify estimand you want
```

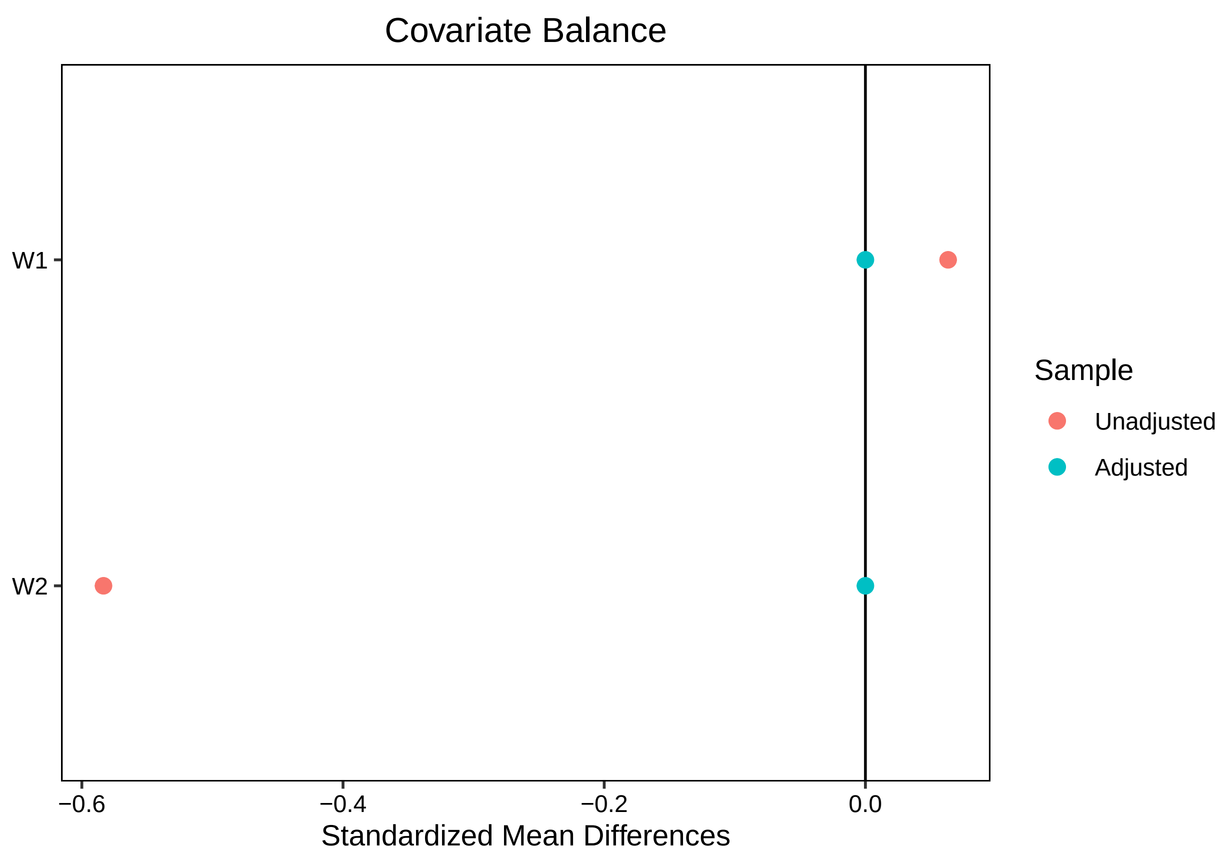
```r
# view
summary(match_exact_ate)
```

```
##
## Call:
## matchit(formula = A ~ W1 + W2, data = df, method = "exact", estimand = "ATE")
##
## Summary of Balance for All Data:
##    Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean eCDF Max
## W1        0.5448        0.5118          0.0633     1.0131    0.0110   0.0303
## W2        0.3403        0.9154         -0.5834     0.5128    0.0958   0.2860
##
## Summary of Balance for Matched Data:
##    Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean eCDF Max
## W1        0.5203        0.5203               0     1.0004         0        0
## W2        0.7677        0.7677               0     1.0004         0        0
##    Std. Pair Dist.
## W1               0
## W2               0
##
## Sample Sizes:
##               Control Treated
## All           7432.   2568.
## Matched (ESS) 7268.96 1794.71
## Matched       7432.   2568.
## Unmatched        0.      0.
## Discarded        0.      0.
```

We can see from the summary how much the balance has improved after matching, but remember that this is only the balance on `W1` and `W2`.

Another useful plot is the `love.plot` that comes from the `cobalt` package. It's an easy way to visualize key information from the table above, specifically comparing the standardized mean differences for "All Data" (unadjusted) with the "Matched Data" (adjusted). You can read more about it here. We want the "Adjusted" sample to have a standardized mean difference of zero.

```r
# plot
# ---------
love.plot(match_exact_ate)
```



To use this matching to estimate the ATE we first get the matched data using the `match.data()` function. We can then use linear regression to estimate the ATE.

```
#
# estimate the ATE using linear regression
# ---------

# construct a matched dataset from the matchit object
match_exact_ate_data <- match.data(match_exact_ate)

# uncomment to see the difference between original dataset and new matchit dataset
#head(df)
#head(match_exact_ate_data)


# specify a linear model
lm_exact_ate <- lm(Y_obs ~ A + W1 + W2 + W3,     # specify the linear model
                   data = match_exact_ate_data, # specify the data
                   weights = weights)           # specify the weights

# view summary of results
lm_exact_ate_summ <- summary(lm_exact_ate)
lm_exact_ate_summ
```

```
##
## Call:
## lm(formula = Y_obs ~ A + W1 + W2 + W3, data = match_exact_ate_data,
##     weights = weights)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -1.51271 -0.06388  0.02962  0.12762  0.76135
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  0.6994727  0.0133966  52.213 < 0.0000000000000002 ***
## A           -0.3072286  0.0063243 -48.579 < 0.0000000000000002 ***
## W1           0.0373288  0.0064028   5.830        0.00000000571 ***
## W2           0.0299750  0.0030030   9.982 < 0.0000000000000002 ***
## W3           0.0072121  0.0005059  14.256 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.276 on 9995 degrees of freedom
## Multiple R-squared:  0.2511, Adjusted R-squared:  0.2508
## F-statistic: 837.7 on 4 and 9995 DF,  p-value: < 0.00000000000000022
```

The ATE estimate is the coefficient estimate on the treatment variable `A`:

```
#
# pull out ATE
# ---------
ATE_exact <- lm_exact_ate_summ$coefficients["A", "Estimate"]
ATE_exact
```

```
## [1] -0.3072286
```

ATT

We could also have estimated the ATT using this method.

```
# ATT using matchit for exact
# --------------------------------------------------
match_exact_att <- matchit(formula = A ~ W1 + W2, data = df,  # formula
                           method = "exact",                  # method
                           estimand = "ATT")                  # estimand

# summary
summary(match_exact_att, un = FALSE)
```

```
##
## Call:
## matchit(formula = A ~ W1 + W2, data = df, method = "exact", estimand = "ATT")
##
## Summary of Balance for Matched Data:
##    Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean eCDF Max
## W1        0.5448        0.5448              -0     1.0002         0        0
## W2        0.3403        0.3403              -0     1.0002         0        0
##    Std. Pair Dist.
## W1               0
## W2               0
##
## Sample Sizes:
```

```
##                Control Treated
## All            7432.      2568
## Matched (ESS) 5545.73    2568
## Matched        7432.      2568
## Unmatched         0.         0
## Discarded         0.         0
```

```
#
# estimate the ATT using linear regression
# ---------

# construct a matched dataset from the matchit object
match_exact_att_data <- match.data(match_exact_att)

# specify a linear model
lm_exact_att <- lm(Y_obs ~ A + W1 + W2 + W3,     # specify linear regression
                   data = match_exact_att_data, # data
                   weights = weights)           # weights

# view summary of results
lm_exact_att_summ <- summary(lm_exact_att)
lm_exact_att_summ
```

```
##
## Call:
## lm(formula = Y_obs ~ A + W1 + W2 + W3, data = match_exact_att_data,
##     weights = weights)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -1.28042 -0.06129  0.01112  0.13846  0.56940
##
## Coefficients:
##                Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  0.7023342  0.0154093  45.579 < 0.0000000000000002 ***
## A           -0.3791960  0.0067949 -55.806 < 0.0000000000000002 ***
## W1           0.0500977  0.0071045   7.052   0.0000000000018869 ***
## W2           0.0307669  0.0040749   7.550   0.0000000000000472 ***
## W3           0.0073089  0.0005953  12.278 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2964 on 9995 degrees of freedom
## Multiple R-squared:  0.2823, Adjusted R-squared:  0.282
## F-statistic: 982.9 on 4 and 9995 DF,  p-value: < 0.00000000000000022
```

```
#
# pull out ATT
# ---------
ATT_exact <- lm_exact_att_summ$coefficients["A", "Estimate"]
ATT_exact
```

```
## [1] -0.379196
```

k Nearest Neighbor Matching Example

ATT

Now let's perform a 2:1 nearest neighbor matching using (logistic regression) propensity scores on all three covariates. Remember that we can only estimate ATT in this case.

```r
#
# ATT using matchit for k-nearest neighbor
# --------------------------------------------------
match_ps_att <- matchit(formula = A ~ W1 + W2 + W3, # formula
                        data = df,                  # data
                        method = "nearest",         # method
                        distance = "glm",           # use glm, which by default is logistic regression
                        link = "logit",             # specify we want a logit model, default when dista
                        discard = "control",        # obs to be discarded that are outside region of co
                        replace = FALSE,            # whether matching should be done with replacement
                        ratio = 2)                  # k:1 matching

# view summary results
summary(match_ps_att)
```

```
##
## Call:
## matchit(formula = A ~ W1 + W2 + W3, data = df, method = "nearest",
##     distance = "glm", link = "logit", discard = "control", replace = FALSE,
##     ratio = 2)
##
## Summary of Balance for All Data:
##          Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance        0.3035        0.2407          0.7512     0.5718    0.1663
## W1              0.5448        0.5118          0.0631     1.0131    0.0110
## W2              0.3403        0.9154         -0.7086     0.5128    0.0958
## W3             28.8879       31.4520         -0.3655     0.8767    0.0975
##          eCDF Max
## distance   0.2937
## W1         0.0303
## W2         0.2860
## W3         0.1422
##
## Summary of Balance for Matched Data:
##          Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance        0.3035        0.2954          0.0969     1.0313    0.0359
## W1              0.5448        0.5292          0.0299     1.0162    0.0052
## W2              0.3403        0.3832         -0.0528     1.0444    0.0097
## W3             28.8879       29.1966         -0.0440     1.1725    0.0252
##          eCDF Max Std. Pair Dist.
## distance   0.1055          0.0971
## W1         0.0129          0.8874
## W2         0.0506          0.0988
## W3         0.0705          0.7968
##
## Sample Sizes:
##           Control Treated
## All          7432    2568
## Matched      5136    2568
## Unmatched    2289       0
## Discarded       7       0
```

```
#
# estimate the ATT using linear regression
# ---------

# construct a matched dataset from the matchit object
match_ps_att_data <- match.data(match_ps_att)

# specify linear model
lm_ps_att <- lm(Y_obs ~ A + W1 + W2 + W3,  # formula
                data = match_ps_att_data,  # data
                weights = weights)         # weights

# view summary results
lm_ps_att_summ <- summary(lm_ps_att)
lm_ps_att_summ
```

```
##
## Call:
## lm(formula = Y_obs ~ A + W1 + W2 + W3, data = match_ps_att_data,
##     weights = weights)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.98752 -0.05100  0.03524  0.15290  0.59344
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  0.6665793  0.0185857  35.865 < 0.0000000000000002 ***
## A           -0.3818981  0.0076443 -49.959 < 0.0000000000000002 ***
## W1           0.0590631  0.0086265   6.847     0.00000000000814 ***
## W2           0.0461488  0.0049705   9.285 < 0.0000000000000002 ***
## W3           0.0082899  0.0007234  11.460 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.316 on 7699 degrees of freedom
## Multiple R-squared:  0.2957, Adjusted R-squared:  0.2954
## F-statistic: 808.3 on 4 and 7699 DF,  p-value: < 0.00000000000000022
```

```
#
# pull out ATT
# ---------
ATT_ps <- lm_ps_att_summ$coefficients["A", "Estimate"]
ATT_ps
```

```
## [1] -0.3818981
```

Full Optimal Mahalanobis Matching Example

Now let's perform a full optimal matching on all three covariates using Mahalanobis distances. (We'll need to do this on a smaller subset of the data)

```r
# set seed
set.seed(1000)

# create a smaller dataframe so this runs more quickly
df_small <-
  df %>%
  slice_sample(n = 1000) # SRS of 1000
```

ATE

```r
#
# ATE using matchit for full optimal matching
# --------------------------------------------------
match_full_ate <- matchit(formula = A ~ W1 + W2 + W3,   # specify formula
                          estimand = "ATE",             # specify estimate
                          data = df_small,              # specify data
                          method = "full",              # specify method
                          distance = "mahalanobis")     # specify distance metric

# view summary of results
summary(match_full_ate)
```

```
##
## Call:
## matchit(formula = A ~ W1 + W2 + W3, data = df_small, method = "full",
##     distance = "mahalanobis", estimand = "ATE")
##
## Summary of Balance for All Data:
##    Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean eCDF Max
## W1        0.5484        0.4879          0.1165     1.0518    0.0202   0.0531
## W2        0.3272        0.9106         -0.6017     0.5480    0.0972   0.3043
## W3       28.9497       31.2446         -0.3209     0.8650    0.0871   0.1619
##
## Summary of Balance for Matched Data:
##    Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean eCDF Max
## W1        0.4990        0.5005         -0.0029     0.9968    0.0005   0.0015
## W2        0.7650        0.7938         -0.0297     0.9445    0.0053   0.0089
## W3       30.6723       30.7964         -0.0174     1.0057    0.0109   0.0401
##    Std. Pair Dist.
## W1          0.0097
## W2          0.0377
## W3          0.1037
##
## Sample Sizes:
##                Control Treated
## All              783.   217.
## Matched (ESS)    729.2  104.87
## Matched          783.   217.
## Unmatched          0.     0.
## Discarded          0.     0.
```

```r
#
# estimate the ATE using linear regression
# ---------
# construct a matched dataset from the matchit object
match_full_ate_data <- match.data(match_full_ate)

# specify linear model
lm_full_ate <- lm(Y_obs ~ A + W1 + W2 + W3,   # specify model
                  data = match_full_ate_data, # specify data
                  weights = weights)          # specify weights

# view summary of results
lm_full_ate_summ <- summary(lm_full_ate)
lm_full_ate_summ
```

```
##
## Call:
## lm(formula = Y_obs ~ A + W1 + W2 + W3, data = match_full_ate_data,
##     weights = weights)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -1.32194 -0.04771  0.03568  0.11417  0.69278
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  0.744019   0.041352  17.992 < 0.0000000000000002 ***
## A           -0.278070   0.019833 -14.021 < 0.0000000000000002 ***
## W1           0.038140   0.019840   1.922             0.054847 .
## W2           0.037659   0.009075   4.150            0.0000361 ***
## W3           0.005604   0.001590   3.525             0.000443 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2585 on 995 degrees of freedom
## Multiple R-squared:  0.2278, Adjusted R-squared:  0.2247
## F-statistic:  73.4 on 4 and 995 DF,  p-value: < 0.00000000000000022
```

```r
#
# pull out ATE
# ---------
ATE_full <- lm_full_ate_summ$coefficients["A", "Estimate"]
ATE_full
```

```
## [1] -0.2780698
```

ATT

```r
#
# ATT using matchit for full optimal matching
# ----------------------------------------------------
match_full_att <- matchit(formula = A ~ W1 + W2 + W3,   # specify formula
                          estimand = "ATT",             # specify estimand
                          data = df_small,              # specify data
                          method = "full",              # specify method
                          distance = "mahalanobis")     # specify distance metric
```

```r
# view summary of results
summary(match_full_att)
```

```
##
## Call:
## matchit(formula = A ~ W1 + W2 + W3, data = df_small, method = "full",
##     distance = "mahalanobis", estimand = "ATT")
##
## Summary of Balance for All Data:
##    Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean eCDF Max
## W1        0.5484        0.4879          0.1151     1.0518    0.0202   0.0531
## W2        0.3272        0.9106         -0.7151     0.5480    0.0972   0.3043
## W3       28.9497       31.2446         -0.3332     0.8650    0.0871   0.1619
##
## Summary of Balance for Matched Data:
##    Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean eCDF Max
## W1        0.5484        0.5461          0.0043     1.0174    0.0008   0.0023
## W2        0.3272        0.3724         -0.0555     0.8327    0.0075   0.0157
## W3       28.9497       29.1790         -0.0333     1.0125    0.0105   0.0314
##    Std. Pair Dist.
## W1          0.0096
## W2          0.0448
## W3          0.1076
##
## Sample Sizes:
##               Control Treated
## All            783.       217
## Matched (ESS)  305.03     217
## Matched        783.       217
## Unmatched        0.         0
## Discarded        0.         0
```

```r
#
# estimate the ATT using linear regression
# ---------
# construct a matched dataset from the matchit object
match_full_att_data <- match.data(match_full_att)

# specify linear model
lm_full_att <- lm(Y_obs ~ A + W1 + W2 + W3,
                  data = match_full_att_data,
                  weights = weights)


# view summary of results
lm_full_att_summ <- summary(lm_full_att)
lm_full_att_summ
```

```
##
## Call:
## lm(formula = Y_obs ~ A + W1 + W2 + W3, data = match_full_att_data,
##     weights = weights)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -1.68392 -0.03375  0.01004  0.10623  0.51513
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  0.749222   0.045870  16.334 < 0.0000000000000002 ***
## A           -0.359589   0.021664 -16.599 < 0.0000000000000002 ***
## W1           0.058367   0.022693   2.572              0.01026 *
## W2           0.029790   0.010877   2.739              0.00627 **
## W3           0.005633   0.001816   3.103              0.00197 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2823 on 995 degrees of freedom
## Multiple R-squared:  0.2598, Adjusted R-squared:  0.2568
## F-statistic: 87.31 on 4 and 995 DF,  p-value: < 0.00000000000000022
```

```
#
# pull out ATT
# ---------
ATT_full <- lm_full_att_summ$coefficients["A", "Estimate"]
ATT_full
```

```
## [1] -0.3595888
```

Question 15: Perform a matching algorithm of your own choosing. Report the estimated ATE or ATT where available. (Note: If your chosen algorithm takes too long to run on `df` you may instead use `df_small`)

```
# Your code here
```

Question 16: Compare the estimates of ATE and ATT found above with the true values (saved as `ATE_true` and `ATT_true`). Which method was most accurate? Considering the pros and cons of different methods we have discussed, which method do you prefer?

```
#
# compare ATE and ATT across matching algorithims
# ---------
# compare ATE
ATE_true
```

```
## [1] -0.2965
```

```
c(ATE_exact, ATE_full)
```

```
## [1] -0.3072286 -0.2780698
```

```
# compare ATT
ATT_true
```

```
## [1] -0.3816199
```

```
c(ATT_exact, ATT_ps, ATT_full)
```

```
## [1] -0.3791960 -0.3818981 -0.3595888
```

ANSWER: It seems for the ATT effect, the propensity score model came the closest to the "true ATE", whereas exact matching seemed to come closest to ATE. One of the reason that full matching likely did worse is that we were matching on a smaller sample (1,000 randomly selected cases) instead, so it might perform better if you were to rerun the analysis with the full data (which will take longer to run, of course.)

References

http://www.stephenpettigrew.com/teaching/gov2001/section11_2015.pdf

https://en.wikipedia.org/wiki/Mahalanobis_distance

https://www.statisticshowto.com/greedy-algorithm-matching/

https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Data_Matching-Optimal_and_Greedy.pdf

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2943670/

https://academic.oup.com/aje/article/173/7/761/103691