Project 8 Morales

```
# Add to this package list for additional SL algorithms
pacman::p_load(
  tidyverse,
  ggthemes,
  ltmle,
  tmle,
  SuperLearner,
  tidymodels,
  caret,
  furrr,
  parallel,
  dagitty,
  ggdag,
  here)

#install.packages("xgboost")
#install.packages("biglasso")
#install.packages("randomForest")


library("xgboost")
```

```
##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library("biglasso")
```

```
## Loading required package: bigmemory

## Loading required package: ncvreg
```

```
library("randomForest")
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
heart_disease <- read_csv(here('Project 8/heart_disease_tmle.csv'))
```

```
## Rows: 10000 Columns: 14

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (14): age, sex_at_birth, simplified_race, college_educ, income_thousands...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
set.seed(283)
```

Introduction

Heart disease is the leading cause of death in the United States, and treating it properly is an important public health goal. However, it is a complex disease with several different risk factors and potential treatments. Physicians typically recommend changes in diet, increased exercise, and/or medication to treat symptoms, but it is difficult to determine how effective any one of these factors is in treating the disease. In this project, you will explore SuperLearner, Targeted Maximum Likelihood Estimation (TMLE), and Longitudinal Targeted Maximum Likelihood Estimation (LTMLE). Using a simulated dataset, you will explore whether taking blood pressure medication reduces mortality risk.

Data

This dataset was simulated using R (so it does not come from a previous study or other data source). It contains several variables:

- **blood_pressure_medication**: Treatment indicator for whether the individual took blood pressure medication (0 for control, 1 for treatment)
- **mortality**: Outcome indicator for whether the individual passed away from complications of heart disease (0 for no, 1 for yes)
- **age**: Age at time 1
- **sex_at_birth**: Sex assigned at birth (0 female, 1 male)
- **simplified_race**: Simplified racial category. (1: White/Caucasian, 2: Black/African American, 3: Latinx, 4: Asian American,
  5: Mixed Race/Other)
- **income_thousands**: Household income in thousands of dollars

- **college_educ**: Indicator for college education (0 for no, 1 for yes)
- **bmi**: Body mass index (BMI)
- **chol**: Cholesterol level
- **blood_pressure**: Systolic blood pressure
- **bmi_2**: BMI measured at time 2
- **chol_2**: Cholesterol measured at time 2
- **blood_pressure_2**: BP measured at time 2
- **blood_pressure_medication_2**: Whether the person took treatment at time period 2

For the "SuperLearner" and "TMLE" portions, you can ignore any variable that ends in "_2", we will reintroduce these for LTMLE.

SuperLearner

Modeling

Fit a SuperLearner model to estimate the probability of someone dying from complications of heart disease, conditional on treatment and the relevant covariates. Do the following:

1. Choose a library of at least 5 machine learning algorithms to evaluate. **Note**: We did not cover how to hyperparameter tune constituent algorithms within SuperLearner in lab, but you are free to do so if you like (though not required to for this exercise).
2. Split your data into train and test sets.
3. Train SuperLearner
4. Report the risk and coefficient associated with each model, and the performance of the discrete winner and SuperLearner ensemble
5. Create a confusion matrix and report your overall accuracy, recall, and precision

```
# Fit SuperLearner Model

## sl lib

listWrappers()
```

```
## All prediction algorithm wrappers in SuperLearner:

##  [1] "SL.bartMachine"      "SL.bayesglm"         "SL.biglasso"
##  [4] "SL.caret"            "SL.caret.rpart"      "SL.cforest"
##  [7] "SL.earth"            "SL.gam"              "SL.gbm"
## [10] "SL.glm"              "SL.glm.interaction"  "SL.glmnet"
## [13] "SL.ipredbagg"        "SL.kernelKnn"        "SL.knn"
## [16] "SL.ksvm"             "SL.lda"              "SL.leekasso"
## [19] "SL.lm"               "SL.loess"            "SL.logreg"
## [22] "SL.mean"             "SL.nnet"             "SL.nnls"
```

```
## [25] "SL.polymars"          "SL.qda"               "SL.randomForest"
## [28] "SL.ranger"            "SL.ridge"             "SL.rpart"
## [31] "SL.rpartPrune"        "SL.speedglm"          "SL.speedlm"
## [34] "SL.step"              "SL.step.forward"      "SL.step.interaction"
## [37] "SL.stepAIC"           "SL.svm"               "SL.template"
## [40] "SL.xgboost"

##
## All screening algorithm wrappers in SuperLearner:

## [1] "All"
## [1] "screen.corP"            "screen.corRank"        "screen.glmnet"
## [4] "screen.randomForest"    "screen.SIS"            "screen.template"
## [7] "screen.ttest"           "write.screen.template"
```

```r
# set seed
set.seed(987)

# multiple models
# ----------
 # I chose the following models: 'SL.mean','SL.glmnet','SL.randomForest','SL.glm','SL.xgboost','SL.bigl
```

```r
## Train/Test split
# initial split
# ----------
heart_split <-
  initial_split(heart_disease, prop = 3/4) # create initial split (tidymodels)

train <- training(heart_split)

# y_train
y_train <-
  train  %>%
  # pull and save as vector
  pull(mortality)

# x_train
x_train <-
  train %>%
  # drop the target variable
  select(-mortality)

# Testing
# ----------
test <-
  # Declare the training set with rsample::training()
  testing(heart_split)

# y test
y_test <-
  test %>%
  pull(mortality)
```

```r
# x test
x_test <-
  test %>%
  select(-mortality)
```

```r
library(doParallel)
```

```
## Loading required package: iterators
```

```r
library(foreach)

# Set up parallel processing
num_cores <- detectCores() - 1  # Reserve one core for system stability
cl <- makeCluster(num_cores)
registerDoParallel(cl)

# Define the Super Learner model with parallel processing
sl <- SuperLearner(Y = y_train,
                   X = x_train,
                   family = binomial(),  # Enable multicore processing
                   SL.library = c('SL.mean',    # if you just guessed the average - serves as a baselin
                                  'SL.glm',
                                  'SL.randomForest',
                                  'SL.xgboost'))

# Stop the cluster after processing
stopCluster(cl)

# Display the Super Learner model details
sl
```

```
##
## Call:
## SuperLearner(Y = y_train, X = x_train, family = binomial(), SL.library = c("SL.mean",
##     "SL.glm", "SL.randomForest", "SL.xgboost"))
##
##
##                           Risk       Coef
## SL.mean_All          0.2497082 0.02635167
## SL.glm_All           0.2355568 0.33562966
## SL.randomForest_All 0.2309637 0.61356845
## SL.xgboost_All       0.2467163 0.02445022
```

```r
## Discrete winner and superlearner ensemble performance

# predictions
# ----------
preds <-
  predict(sl,             # use the superlearner not individual models
          x_test,         # prediction on test set
          onlySL = TRUE)  # use only models that were found to be useful (had weights)
```

```r
# start with y_test
validation <-
  y_test %>%
  # add our predictions - first column of predictions
  bind_cols(preds$pred[,1]) %>%
  # rename columns
  rename(obs = `...1`,       # actual observations
         pred = `...2`) %>% # predicted prob
  # change pred column so that obs above .5 are 1, otherwise 0
  mutate(pred = ifelse(pred >= .5,
                           1,
                           0))
```

```
## New names:
## * `` -> `...1`
## * `` -> `...2`
```

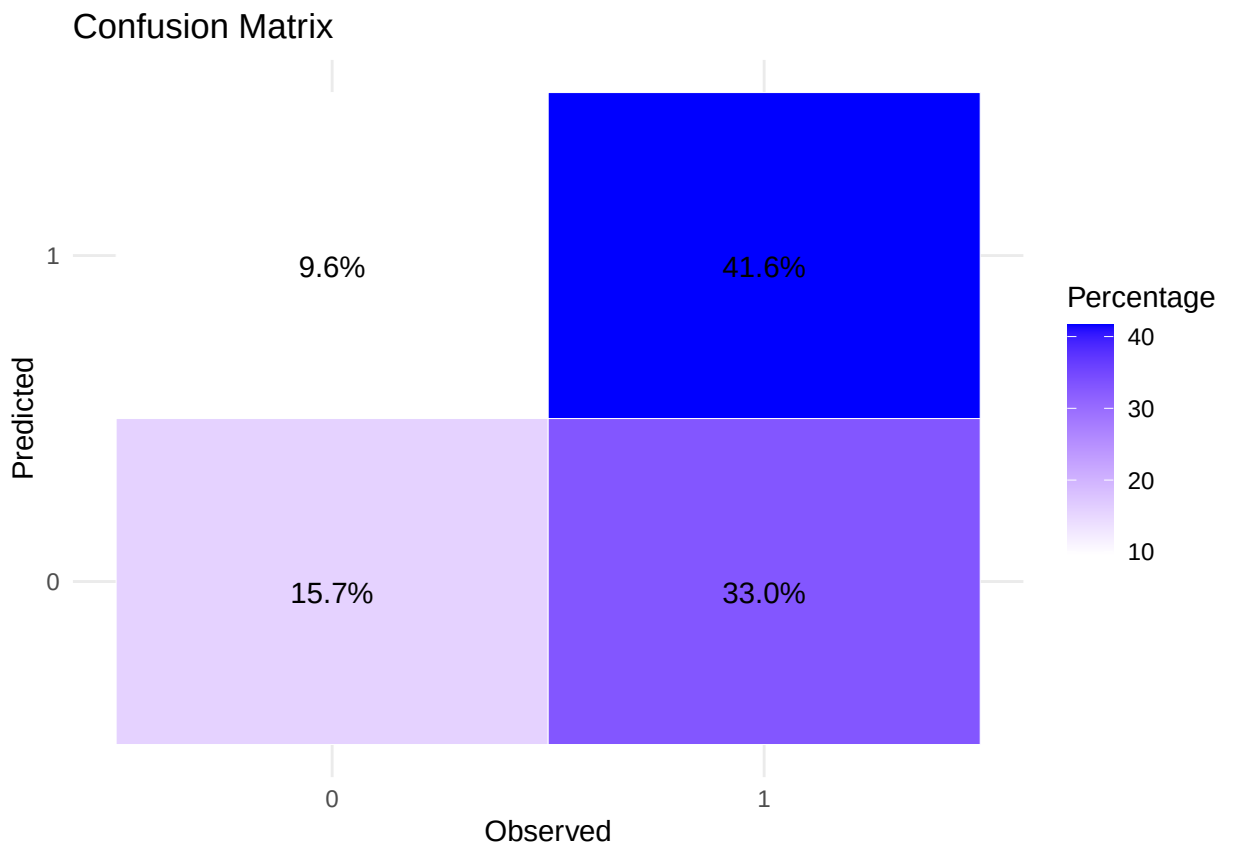
```r
# view
head(validation)
```

```
## # A tibble: 6 x 2
##     obs  pred
##   <dbl> <dbl>
## 1     1     1
## 2     0     1
## 3     1     1
## 4     1     1
## 5     0     1
## 6     1     1
```

```r
## Confusion Matrix
```

```r
# confusion matrix
# ----------
conf_matrix <- confusionMatrix(as.factor(validation$pred),
                               as.factor(validation$obs))


# Extract the table from the confusion matrix
matrix_data <- as.data.frame(as.table(conf_matrix$table))

# Rename the columns for easier understanding
colnames(matrix_data) <- c("Reference", "Prediction", "Frequency")

# Convert frequencies to percentages
total_observations <- sum(matrix_data$Frequency)
matrix_data$Percentage <- (matrix_data$Frequency / total_observations) * 100

# Plot using ggplot2 with percentages
ggplot(matrix_data, aes(x = Reference, y = Prediction, fill = Percentage)) +
  geom_tile(color = "white") + # Use geom_tile for the heatmap effect
  scale_fill_gradient(low = "white", high = "blue") + # Color gradient
```

```
  geom_text(aes(label = sprintf("%.1f%%", Percentage)), vjust = 1) + # Add percentage text
  theme_minimal() + # Use a minimal theme
  labs(title = "Confusion Matrix", x = "Observed", y = "Predicted") # Labels
```




```
overall_stats <- conf_matrix$overall
accuracy <- overall_stats['Accuracy']

# Display the metrics
cat("Accuracy: ", accuracy * 100, "%\n")
```

```
## Accuracy:  57.36 %
```

Discussion Questions

1. Why should we, in general, prefer the SuperLearner ensemble to the discrete winner in cross-validation? Or in other words, what is the advantage of "blending" algorithms together and giving them each weights, rather than just using the single best algorithm (with best being defined as minimizing risk)?

2. The Superlearner ensamble is always better or equal than any of the algorithms in it, and it can adjust also for expert specifications. As usual, the big improvement also comes from reducing the overfitting of individual models by weightening against the others and using cross-validation. So pretty much, it mitigates the risk of sticking with one model that might have some issues, and it takes the best out of all of them. As we learned on the course, each model might have strengths depending on the data and specifications, superLearners provides a tool to make the best out of all.

Targeted Maximum Likelihood Estimation

Causal Diagram

TMLE requires estimating two models:

1. The outcome model, or the relationship between the outcome and the treatment/predictors, $P(Y|(A, W)$.
2. The propensity score model, or the relationship between assignment to treatment and predictors $P(A|W)$

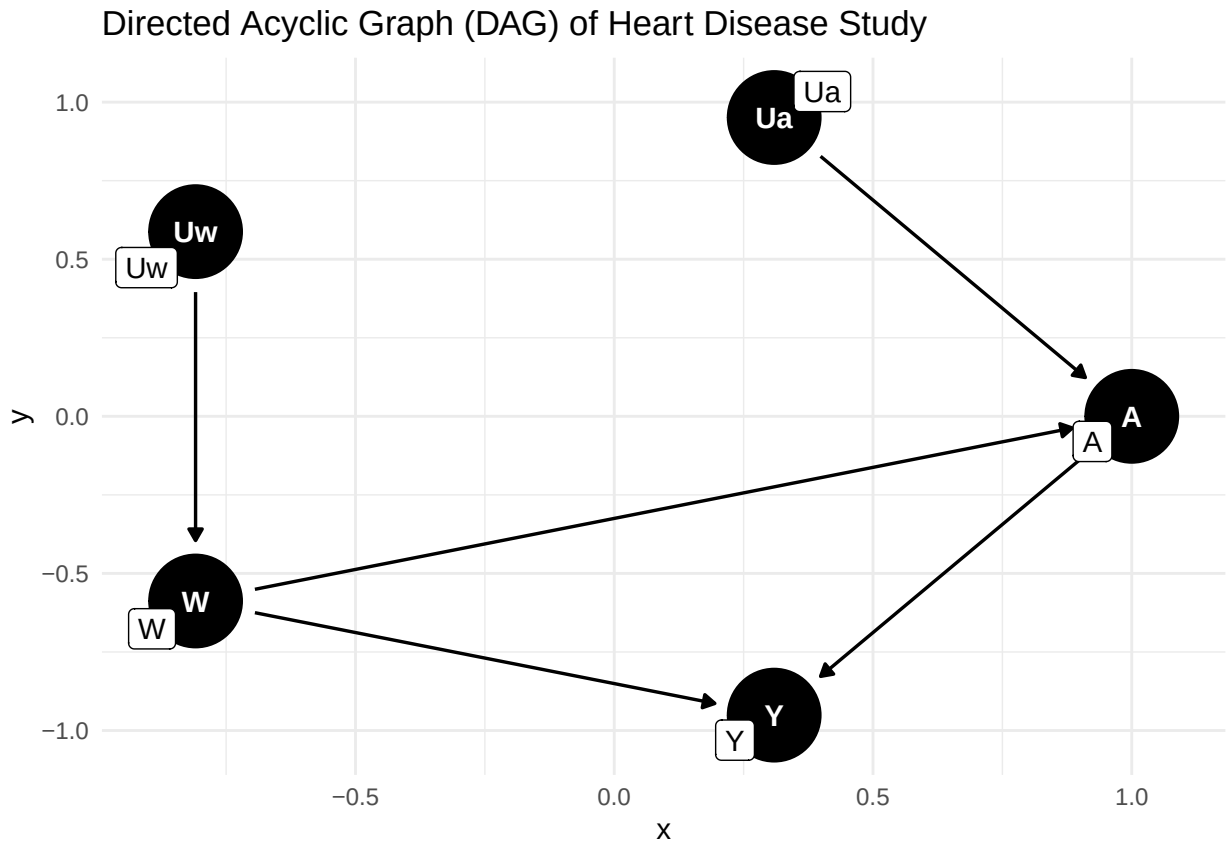
Using ggdag and daggity, draw a directed acylcic graph (DAG) that describes the relationships between the outcome, treatment, and covariates/predictors. Note, if you think there are covariates that are not related to other variables in the dataset, note this by either including them as freestanding nodes or by omitting them and noting omissions in your discussion.

W = age, sex_at_birth, simplified_race, income_thousands, college_educ, bmi, chol, blood_pressure A = blood_pressure_medication Y = mortality Uw = unobserved variables affecting covariates Ua = unobserved variables affecting treatment

Notice that almost all covariets have an effect on the prescription and take-up of the medication: due to guidelines we know there are differential recomemndations based on age, sex, race, and bmi, chol, blood presure levels; and then we see that education, income also play a role in how many people will follow the treatment and adhere.

Similarly, all these variables are related to mortality

```
# DAG for TMLE
library(ggdag)
library(dagitty)

# Define DAG

dag <- dagitty::dagitty("
dag {
  Uw [unobserved]
  Ua [unobserved]
  W -> A -> Y
  W -> Y
  Uw -> W
  Ua -> A
}
")

ggdag <- ggdag::ggdag(dag, text = TRUE, use_labels = "name", layout = "circle") +
  theme_minimal() +
  ggtitle("Directed Acyclic Graph (DAG) of Heart Disease Study")

print(ggdag)
```



TMLE Estimation

Use the `tmle` package to estimate a model for the effect of blood pressure medication on the probability of mortality. Do the following:

1. Use the same SuperLearner library you defined earlier
2. Use the same outcome model and propensity score model that you specified in the DAG above. If in your DAG you concluded that it is not possible to make a causal inference from this dataset, specify a simpler model and note your assumptions for this step.
3. Report the average treatment effect and any other relevant statistics

```
# Set seed for reproducibility
set.seed(876)

# Super Learner libraries
sl_libs <- c('SL.mean', 'SL.randomForest', 'SL.glm', 'SL.xgboost')

# Define the outcome, treatment, and covariates
Y <- "mortality"
A <- "blood_pressure_medication"
W <- c("age", "sex_at_birth", "simplified_race", "income_thousands", "college_educ", "bmi", "chol", "bl

# Setup parallel processing
```

```r
num_cores <- detectCores() - 1  # Reserve one core for system stability
cl <- makeCluster(num_cores)
registerDoParallel(cl)

# Run the TMLE
tmle_fit <- tmle(
  Y = heart_disease[[Y]],
  A = heart_disease[[A]],
  W = heart_disease[W],
  family = "binomial",  # Assuming Y is binary
  Q.SL.library = sl_libs,
  g.SL.library = sl_libs
)

# Stop the cluster after processing
stopCluster(cl)

# View results
tmle_fit
```

```
##  Additive Effect
##    Parameter Estimate:  -0.35469
##    Estimated Variance:  6.6335e-05
##               p-value:  <2e-16
##     95% Conf Interval:  (-0.37066, -0.33873)
##
##  Additive Effect among the Treated
##    Parameter Estimate:  -0.31793
##    Estimated Variance:  0.000146
##               p-value:  <2e-16
##     95% Conf Interval:  (-0.34161, -0.29425)
##
##  Additive Effect among the Controls
##    Parameter Estimate:  -0.36181
##    Estimated Variance:  5.9307e-05
##               p-value:  <2e-16
##     95% Conf Interval:  (-0.3769, -0.34671)
##
##  Relative Risk
##    Parameter  Estimate: 0.37417
##    Variance(log scale): 0.00093814
##                 p-value: <2e-16
##      95% Conf Interval: (0.35237, 0.39732)
##
##  Odds Ratio
##    Parameter  Estimate: 0.20574
##    Variance(log scale): 0.0018358
##                 p-value: <2e-16
##      95% Conf Interval: (0.18917, 0.22376)
```

Discussion Questions

1. What is a "double robust" estimator? Why does it provide a guarantee of consistency if either the outcome model or propensity score model is correctly specified? Or in other words, why does mispecifying one of the models not break the analysis? **Hint**: When answering this question, think about how your introductory statistics courses emphasized using theory to determine the correct outcome model, and in this course how we explored the benefits of matching.

2. Doubly robust estimators are two step models, where a first model estimates the probability of treatment, and the second uses that probability to estimate the target estimator of the outcomes. The biggest advantage of such estimators is that if either of the two stages of the model are correctly specified there will be consistency in the results. The first part of the model addresses the potential biases coming from self-selection to the treatment variable, by matching observations based on observable characteristics, whereas the second part addressed the misspecification of the model by adjusting the estimate based on treatment and covariates. The advantage of the first part is that finds a common support and weights the observations to create the best counterfactual by letting ML do the best match based on predictions. Theory, as the creator of TMLE, Mark VanDerLaan says it is a great way to have a first outcome that will help to target the estimator, and you can include as many theoretically based models to the SuperLearner, so the model will be as good or better than theory informed. But in doubly robust estimators like TMLE, it can be discovered that researchers might have their own biases and might not be getting the best estimate of treatment, or their best specification of the regression model for the outcome. At the end of the day, we still need theory to build our DAGs, specially to address any potential missing variable, that could be biasing the results. DR methods can't improve the estimates if there are a lot of missing variables or unobservables, which highlights why this methods are still prone to similar weaknesses than other models.

LTMLE Estimation

Now imagine that everything you measured up until now was in "time period 1". Some people either choose not to or otherwise lack access to medication in that time period, but do start taking the medication in time period 2. Imagine we measure covariates like BMI, blood pressure, and cholesterol at that time for everyone in the study (indicated by a "_2" after the covariate name).

Causal Diagram

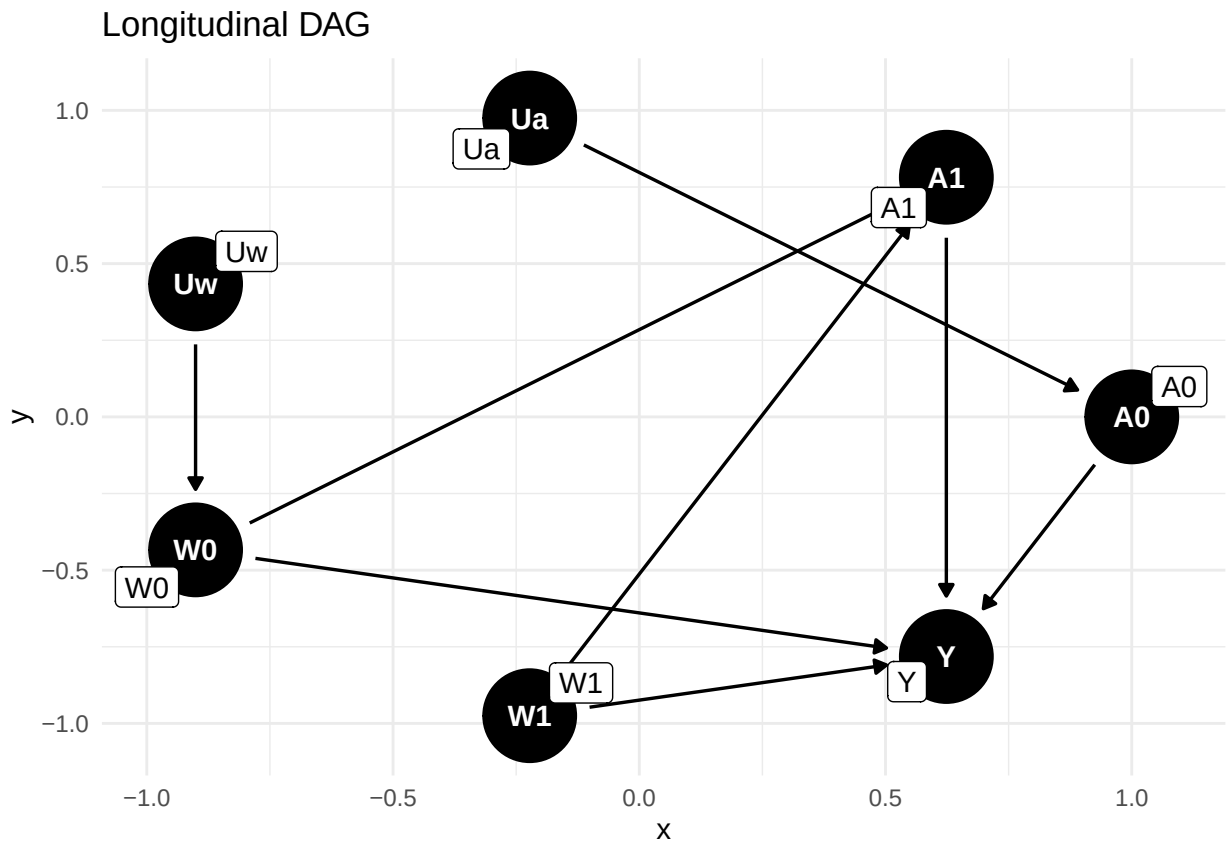
Update your causal diagram to incorporate this new information. **Note**: If your groups divides up sections and someone is working on LTMLE separately from TMLE then just draw a causal diagram even if it does not match the one you specified above.

Hint: Check out slide 27 from Maya's lecture, or slides 15-17 from Dave's second slide deck in week 8 on matching.

Hint: Keep in mind that any of the variables that end in "_2" are likely affected by both the previous covariates and the first treatment when drawing your DAG.

```
# DAG for TMLE


dag <- dagitty::dagitty("
dag {
  Uw [unobserved]
  Ua [unobserved]
  W1-> A1 -> Y
```

```
  W1 -> Y
  W0 -> Y
  W0 -> A1
  A0 -> Y
  Uw -> W0
  Ua -> A0
}
")

ggdag <- ggdag::ggdag(dag, text = TRUE, use_labels = "name", layout = "circle") +
  theme_minimal() +
  ggtitle("Longitudinal DAG")

# Print the DAG
print(ggdag)
```



LTMLE Estimation

Use the `ltmle` package for this section. First fit a "naive model" that **does not** control for the time-dependent confounding. Then run a LTMLE model that does control for any time dependent confounding. Follow the same steps as in the TMLE section. Do you see a difference between the two estimates?

```
library(foreach)
```

```r
# Load your dataset heart_disease and ensure it is properly structured
# Example: heart_disease <- read.csv("path_to_your_data.csv")

# Naive Model (no time-dependent confounding) estimate
data_obs <-
  heart_disease %>%
  rename(Y = mortality, A = blood_pressure_medication, W1=age, W2=sex_at_birth, W3=simplified_race, W4=

data_obs_ltmle <-
  data_obs %>%
  select(W1, W2, W3,W4,W5,W6,W7,W8, A, Y)

# Set up parallel processing
num_cores <- detectCores() - 1  # Reserve one core for system stability
cl <- makeCluster(num_cores)
registerDoParallel(cl)

# Implement ltmle with parallel processing
result <- ltmle(data_obs_ltmle, # dataset
                Anodes = "A",   # vector that shows treatment
                Ynodes = "Y",   # vector that shows outcome
                abar = 1)
```

```
## Qform not specified, using defaults:

## formula for Y:

## Q.kplus1 ~ W1 + W2 + W3 + W4 + W5 + W6 + W7 + W8 + A

##

## gform not specified, using defaults:

## formula for A:

## A ~ W1 + W2 + W3 + W4 + W5 + W6 + W7 + W8

##

## Estimate of time to completion: < 1 minute
```

```r
# Stop the cluster after processing
stopCluster(cl)

# View results
result
```

```
## Call:
## ltmle(data = data_obs_ltmle, Anodes = "A", Ynodes = "Y", abar = 1)
##
## TMLE Estimate:  0.2041733
```

LTMLE estimate

```
# implement ltmle
# Set the seed for reproducibility
set.seed(125)

# Super Learner libraries
sl_libs <- c('SL.mean', 'SL.glm', 'SL.xgboost', 'SL.randomForest')

# Prepare the data
data <- heart_disease %>%
  rename(Y = mortality,
         A1 = blood_pressure_medication,
         A2 = blood_pressure_medication_2,
         W1 = age, W2 = sex_at_birth,
         W3 = simplified_race,
         W4 = college_educ,
         W5 = income_thousands,
         W6 = bmi, W7 = blood_pressure,
         W8 = chol,
         L1 = bmi_2,
         L2 = blood_pressure_2,
         L3 = chol_2)

# Check if data is a data frame (needed for ltmle code)
print(is.data.frame(data))
```

```
## [1] TRUE
```

```
# Set up parallel processing
num_cores <- detectCores() - 1  # Reserve one core for system stability
cl <- makeCluster(num_cores)
registerDoParallel(cl)

# Implement ltmle with a specified Super Learner library in parallel
ltmle_res <- ltmle(data,
                   Anodes = c("A1", "A2"),  # Two treatment variables
                   Lnodes = c("L1", "L2", "L3"),  # L indicators
                   Ynodes = "Y",  # Outcome
                   abar = c(1, 1),  # Treatment indicator in Anodes vector
                   SL.library = sl_libs)
```

```
## Qform not specified, using defaults:

## formula for L1:

## Q.kplus1 ~ W1 + W2 + W3 + W4 + W5 + W6 + W7 + W8 + A1

## formula for Y:

## Q.kplus1 ~ W1 + W2 + W3 + W4 + W5 + W6 + W7 + W8 + A1 + L1 +     L2 + L3 + A2
```

```
##
## gform not specified, using defaults:
## formula for A1:
## A1 ~ W1 + W2 + W3 + W4 + W5 + W6 + W7 + W8
## formula for A2:
## A2 ~ W1 + W2 + W3 + W4 + W5 + W6 + W7 + W8 + A1 + L1 + L2 + L3
##
## Estimate of time to completion: 3 to 8 minutes
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```r
# Stop the cluster after processing to free up resources
stopCluster(cl)

# View the ltmle results
ltmle_res
```

```
## Call:
## ltmle(data = data, Anodes = c("A1", "A2"), Lnodes = c("L1", "L2",
##     "L3"), Ynodes = "Y", abar = c(1, 1), SL.library = sl_libs)
##
## TMLE Estimate:  0.2111822
```

LTMLE results are a little bit larger at 0.21 than the naive ltmle results 0.20, and also a bit larger than the original TMLE results. The change is 5% of the original one so doesn't seem too much but it is also not insignifcant.

NOTE: This results are based on the last results I obtained before knittin, it might change now that I run the last version as I decided to try one more algorithm with parallelization to see how it works. I just got a 11 core Mac and this is a great way to see how it performs. That said, it still take a while and I want to delive on time, so might not be able to run it again.

Discussion Questions

1. What sorts of time-dependent confounding should we be especially worried about? For instance, would we be concerned about a running variable for age the same way we might be concerned about blood pressure measured at two different times?

2. There are confounding variables that are endogenous to the model like blood pressure, that is a function of previous variables like taking medication or even changing the bmi. This variables can't be assume to not have a time dependency, so LTMLE is a better fit. Variables like age are not determined by previous events or variables (unless there is attrition based on age but that's another story) so it is a variable that does not need a time-dependant specification.