Methods:

The Methods section outlines the variable selection approaches adopted to address the research questions.

Bross Formula Method:

Bross formula is a statistical method for calculating the bias contributed by not adjusting for a covariate in larger datasets. It is employed to assess the importance of individual features. As the Bross formula would rank all proxy variables without excluding any, the top 100 proxies are manually selected for further modeling.

To implement the Bross Formula method, the following code is used: (`k = 100` specifies the manually selected top-ranked variables)

```
out3 <- get_prioritised_covariates(df = out2,
                                   patientIdVarname = "idx",
                                   exposureVector = basetable$exposure,
                                   outcomeVector = basetable$outcome,
                                   patientIdVector = patientIds,
                                   k = 100)
hdps.dim <- out3$autoselected_covariate_df
```

Lasso Regression Method:

Lasso is a variable selection technique that limits the number of variables in a linear model by adding a penalty L_1 which is proportional to both the absolute value of the model coefficients and a hyperparameter lambda. Cross-validation (CV) in Lasso is applied to identify variables with non-zero coefficients in the best model with using the optimal value of lambda, for further modeling.

The CV of Lasso is fitted using the function `cv.glmnet`, with hyperparameters `alpha = 1` indicating the L_1 penalty of lasso and `nfolds = 5` indicating five-fold CV to determine the optimal lambda (i.e., five runs, each with a different 1/5 of the data points as validation set and the remaining 4/5 as training set). `covar.mat` is the design matrix containing the data and the full formula (i.e., outcome ~ exposure + all covariates + all proxy variables).

```
lasso.fit <- glmnet::cv.glmnet(y = hdps.data$outcome,
                               x = covar.mat,
                               type.measure='mse',
                               family="binomial",
                               alpha = 1,
                               nfolds = 5)
```

Hybrid Bross Formula & Lasso:

A combination of Bross formula and Lasso is also employed. This hybrid approach first picks the proxies prioritized by the Bross formula (i.e. the first 100 features) and then applies Lasso CV on these variables.

Elastic Net (EN) Regression:

Another method used is EN variable selection, which is similar to Lasso as it also includes the L_1 penalty. However, EN also incorporates another quadratic expression (L_2) in the penalty, which allows it to outperform than Lasso in the context of handling multicollinearity by grouping correlated features and selecting the most representative ones.

The CV of EN is fitted using the function `cv.glmnet`, with hyperparameters `alpha = 0.5` indicating the both L_1 and L_2 penalty of EN and `nfolds = 5` indicating five-fold CV to determine the optimal lambda.

```
elas.fit <- glmnet::cv.glmnet(y = hdps.data$outcome,
                              x = covar.mat,
                              type.measure='mse',
                              family="binomial",
                              alpha = 0.5,
                              nfolds = 5)
```

```
bounds <- t(data.frame(log2lambda1=c(-10, 10), log2lambda2=c(-10,10)))
colnames(bounds)<-c("lower", "upper")
svm.model <- svmfs(x=covar.mat,
           y = svmTrainOutcome,
           fs.method = "scad+L2",
           bounds=bounds,
           grid.search = "interval",
           inner.val.method = "cv",
           show = "none",
           parms.coding = "none",
           seed=42)
```

Genetic Algorithm:

GA is an evolutionary algorithm inspired by the theory of natural selection. It operates by evolving offspring from a population of the fittest individuals over several generations. GA evaluates and selects the best combination of features that maximizes the prediction accuracy.

To implement the GA, the following code is used:

```
gaOpt <- function(vars, IV.train, DV.train) {
  varNames <- colnames(IV.train) #getting names of all variables
  selectedVarNames <- varNames[vars == "1"] # getting names of selected vars from GA
  gaSolutionData <- IV.train[,selectedVarNames] # keeping only those selected vars

  gaDat <- cbind(gaSolutionData,DV.train) # combining selected variables with outcome variable
  gaMod <- glm(DV.train ~ ., family = "binomial", data = gaDat) #build model
  gaProb <- predict(gaMod, IV.train, type = "response") # get probabilities
  gaPred <- ifelse(gaProb >= .8, 1, 0) # get predicted 0s and 1s

  ari <- adjustedRandIndex(gaPred, DV.train)
  return(ari)
}

ga.solution <- ga(fitness = function(vars)
 gaOpt(vars=vars,
       IV.train=data.frame(covar.mat),
       DV.train=hdps.data$outcome),
 type = "binary",
 nBits = ncol(covar.mat),
 names = colnames(covar.mat),
 seed = 42,
 run=5)
```

XGBoost:

XGBoost is a gradient boosting algorithm for optimizing machine learning models. It is based on decision trees that make splits with maximum decrease in impurity, and derives the importance score for each feature by calculating the mean decrease in impurity.

The XGBoost model is fitted with hyperparameters: `max.depth = 20` controls the maximum depth of each tree; `eta = 1` specifies the step weight of each boosting step, or the learning rate; `nthread = 2` sets the number of thread used in training; `nrounds = 5` limits the max number of iterations; ` objective = "binary:logistic"` is for binary classification tasks.

```
xgb_fit <- xgboost(data = covar.mat, label = hdps.data$outcome,
                   max.depth = 20, eta = 1, nthread = 2, nrounds = 5,
                   objective = "binary:logistic")
```

Random Forest Algorithm:

Random forest algorithm is an ensemble learning method that builds multiple decision trees to obtain a mode for classification. It would calculate the importance measure for each feature based on the decrease of impurity or Gini importance, providing a ranking of features. As the Random Forest algorithm would rank all proxy variables without excluding any, the top 100 variables are manually selected for further modeling.

The following code is used to implement the Random Forest algorithm, get all the proxy variables ranked by the importance. The top 100 proxies are manually selected for further modeling.

```
rf_model <- randomForest(outcome ~ ., data = hdps.data, importance = TRUE)
rf_importance <- importance(rf_model)
important_features_rf <- rownames(rf_importance)[order(rf_importance[, 1], decreasing = TRUE)]
fs_rf <- important_features_rf[important_features_rf %in% proxy.list][1:100]
```

Stepwise Regression using function `regsubsets()`:

Stepwise selection is a progressive feature selection method for linear regression models. It selects features by evolving the model in different directions, forward or backward. Forward selection starts feature selection with an initial model and uses a series of steps to allow features to enter the model one at a time. Backward selection starts with a full model and uses a series of steps to allow features to leave the regression model one at a time.

For the Stepwise Forward Algorithm, the full formula contains all investigator selected features and all proxy variables. The model with maximum adjusted R^2 is selected as the best model and its corresponding proxies are selected for further analysis. The following code is used:

```
full.formula <- as.formula(paste0("outcome~exposure+",
                                  rhsformula.full,
                                  collapse = "+"))

stepwise_forward <- regsubsets(full.formula,
                               data = hdps.data,
                               method = "forward",
                               nvmax = length(proxy.list),
                               nbest = 10)

summary_stepwise <- summary(stepwise_forward)
best_model <- which.max(summary_stepwise$adjr2)
sel.variables <- names(summary_stepwise$which[best_model,])[summary_stepwise$which[best_model,]]
```

For the Stepwise Backward Algorithm, the following code is used:

```
stepwise_backward <- regsubsets(full.formula,
                                data = hdps.data,
                                method = "backward",
                                nvmax = length(proxy.list),
                                nbest = 10)

summary_stepwise <- summary(stepwise_backward)
best_model <- which.max(summary_stepwise$adjr2)
sel.variables <- names(summary_stepwise$which[best_model,])[summary_stepwise$which[best_model,]]
fs.backward <- proxy.list[proxy.list %in% sel.variables]
```

Results:

Figure 1 shows: (Column 1,) the number of proxy variables selected by each variable selection method; (Column 2,) the number of common features selected by each method and the Bross formula method, and (Column 3,) the percentage of the common features in the total number of features of each method.

	numFeature	numInCommon	pctCommon
bross	100	*NA*	*NA*
hybrid	49	49	1.0000000
lasso	60	47	0.7833333
elastic	69	54	0.7826087
GA	64	44	0.6875000
xgboost	48	38	0.7916667
rf	100	72	0.7200000
forward	59	45	0.7627119
backward	59	45	0.7627119

Figure 1, Comparison of the number & percentage of proxy variables selected by different methods in common with that by the Bross Formula

The Hybrid method shows a feature common ratio of 1 with the Bross formula method. As the hybrid approach performs lasso variable selection on the variables selected by the Bross formula method, the ratio would always be 1 regardless of the number of variables selected.

Most of the methods show a feature common ratio around 0.77 with the Bross formula method, while GA and random forest show a slightly lower common rate of 0.688 and 0.72, respectively.

Figure 2 shows the one-to-one number of proxy features in common between each two variable selection methods. Every cell represents the number of common proxies of its row name method and its column name method. The diagonal, with the same row name and column name, represents the number of proxy variables selected by the method. Half of the table is blank to avoid duplication of information.

	bross	hybrid	lasso	elastic	GA	xgboost	rf	forward	backward
bross	100	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
hybrid	49	49	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
lasso	47	47	60	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
elastic	54	48	60	69	*NA*	*NA*	*NA*	*NA*	*NA*
GA	44	28	36	40	64	*NA*	*NA*	*NA*	*NA*
xgboost	38	24	28	30	25	48	*NA*	*NA*	*NA*
rf	72	37	42	50	39	36	100	*NA*	*NA*
forward	45	41	51	54	35	25	43	59	*NA*
backward	45	41	51	54	35	25	43	59	59

Figure 2, One-to-one comparison of the number of proxy variables selected by different methods

One pair of methods of some comparable interests are the Stepwise Regression in different directions. It is noticeable that the subsets of proxy variables selected by Forward and Backward Regression are exactly the same, 59 out of 142 proxy variables.

Figure 3 shows the estimated odds ratios (ORs) and the corresponding confidence intervals (CIs) of models with all investigator specified covariates and proxy variables selected by each method.

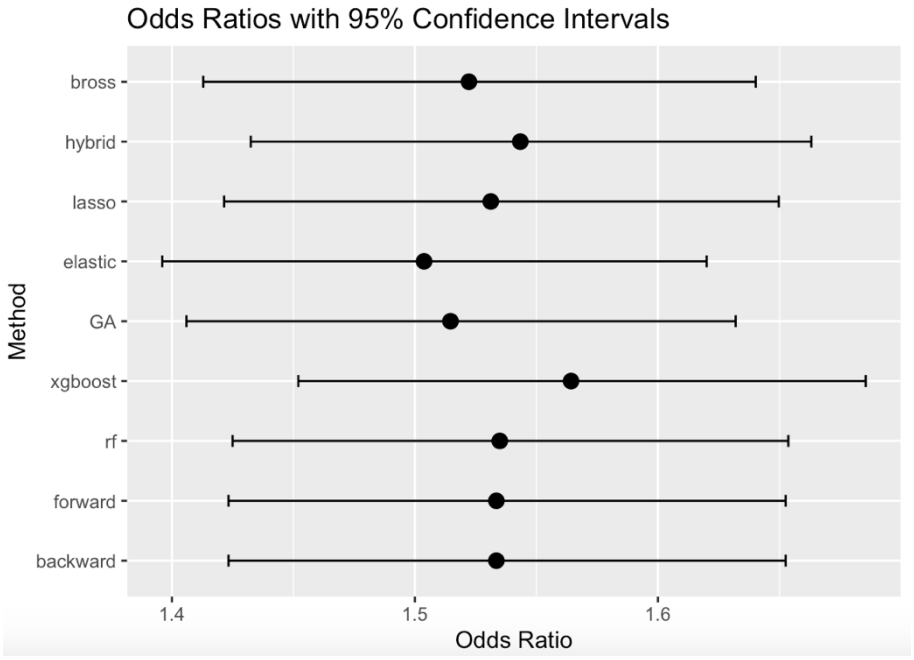

Figure 3, Comparison of odds ratios (ORs) and the confidence intervals (CIs) of models by different proxy selection methods

It is noticeable that none of the confidence intervals contain 1, and most of the estimated ORs are around 1.54, except for the OR of Elastic Net SVM, which contains only 1 proxy variable (and all investigator specified covariates).

The same plot as Figure 3, with additional information: the number of proxies selected.