python in R

python environment setting

```r
use_condaenv("final")
bartpy <- import("bartpy2.sklearnmodel")
time_py <- import("time")
numpy <- import("numpy")

# unnormalize function from [-0.5,0.5]
unnormalize_x <- function(y_train,y_new){
  x <- data.frame()
  y_min <- min(y_train)
  y_max <- max(y_train)
  for (i in 1:nrow(y_new)) {
    for (j in 1:ncol(y_new)) {
      x[i,j] <- (y_max-y_min)*(y_new[i,j]+0.5)+y_min
    }
  }
  return(x)
}
```

create dataset

```r
linear_dgp_fun <- function(ratio,n, p, noise_sd) {
  set.seed(123)
  n_train <- n*ratio
  beta <-  sample(1:100, p, replace = FALSE)

  #n <- n_train + n_test
  X <- matrix(rnorm(n * p), nrow = n, ncol = p)
  y <- X %*% beta + rnorm(n, sd = noise_sd)
  data_list <- list(
    X_train = X[1:n_train, , drop = FALSE],
    y_train = y[1:n_train],
    X_test = X[(n_train + 1):n, , drop = FALSE],
    y_test = y[(n_train + 1):n]
  )
  return(data_list)
}
linear_dgp <- create_dgp(
  .dgp_fun = linear_dgp_fun, .name = "Linear DGP",
  # additional named parameters to pass to .dgp_fun()
  ratio = 0.8, n = 500, p = 4,  noise_sd = 1
)

dataset_dgp_fun <- function(datasetname){

  address <- "C:/Users/pyk/Desktop/nus/RA/project/imodels-data-master/data_cleaned/"
  file <- paste0(datasetname,".csv")
  file_path <- paste0(address,file)
  df <- read.csv(file_path)
  x <- df[, -ncol(df)]
  y <- df[, ncol(df)]

  train_indices <- createDataPartition(y, p = 0.8, list = FALSE)

  data_list <- list(
    X_train <- x[train_indices, ],
    y_train <- y[train_indices],
    X_test <- x[-train_indices, ],
    y_test <- y[-train_indices]
  )
  return(data_list)
}
dataset_dgp <- create_dgp(.dgp_fun = dataset_dgp_fun,.name = 'heart',
                          datasetname = "heart")
```

build BART model

```r
BART_fun <- function(X_train, y_train, X_test, y_test, df,k,q) {
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark(fit <- wbart(x.train = train_X,
                                y.train = y_train,
                                x.test = test_X,
                                k = k,
                                sigdf = df,
                                sigquant = q
                                ))
  time <- mean(t$time[[1]])
  predictions <- colMeans(fit$yhat.test)
  mse_score <- mean((y_test - predictions)^2)

  lower_bounds <- apply(fit$yhat.test, 2, quantile, probs = 0.025)
  upper_bounds <- apply(fit$yhat.test, 2, quantile, probs = 0.975)

  coverage <- mean(y_test >= lower_bounds & y_test <= upper_bounds)
  return(list(time = time, mse=mse_score,coverage = coverage))
}

dbarts_fun <- function(X_train, y_train, X_test, y_test, df,k,q){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark(bart_model <- bart(x.train = train_X,
                                       y.train = y_train,
                                       x.test = test_X,
                                       k = k,
                                       sigdf = df,
                                       sigquant = q))
  time <- mean(t$time[[1]])
  predictions <- colMeans(bart_model$yhat.test)
  mse_score <- mean((y_test - predictions)^2)

  lower_bounds <- apply(bart_model$yhat.test, 2, quantile, probs = 0.025)
  upper_bounds <- apply(bart_model$yhat.test, 2, quantile, probs = 0.975)

  coverage <- mean(y_test >= lower_bounds & y_test <= upper_bounds)

  return(list(time = time, mse=mse_score,coverage = coverage))
}
```

```r
bartMachine_fun <- function(X_train, y_train, X_test,y_test,df,k,q){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark(bart_model <- bartMachine(
          X = train_X,
          y = y_train,
          k = k,
          nu = df,
          q=q))
        # The value of calculating the time required for modeling
  time <- mean(t$time[[1]])
  predictions <- predict(bart_model,test_X,type = "prob")
  mse_score <- mean((y_test - predictions)^2)

  CI <- calc_credible_intervals(bart_model,test_X)
  coverage <- mean(y_test >= CI[,1] & y_test <= CI[,2])

  return(list(time = time, mse=mse_score,coverage = coverage))
}

SoftBart_fun<- function(X_train, y_train, X_test,y_test,num_trees,alpha,beta){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <-  bench::mark({bart_model <- softbart(X = train_X, Y = y_train, X_test = test_X, hypers

  time <- mean(t$time[[1]])
  predictions <- bart_model$y_hat_test_mean
  mse_score <- mean((y_test - predictions)^2)

  lower_bounds <- apply(bart_model$y_hat_test, 2, quantile, probs = 0.025)
  upper_bounds <- apply(bart_model$y_hat_test, 2, quantile, probs = 0.975)

  coverage <- mean(y_test >= lower_bounds & y_test <= upper_bounds)

  return(list(time = time, mse=mse_score,coverage = coverage))
}

RF_fun <- function(X_train, y_train, X_test,y_test){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark({rf_model <- randomForest(x=train_X, y=y_train)})
  time <- mean(t$time[[1]])
  predictions <- predict(rf_model, test_X)
  mse_score <- mean((y_test - predictions)^2)
  return(list(time = time, mse=mse_score))
}

bartpy_fun <- function(X_train, y_train, X_test,y_test){
  train_x <- numpy$array(X_train)
  train_y <- numpy$array(y_train)
  test_x <- numpy$array(X_test)
  test_y <- numpy$array(y_test)

  bart_model <- bartpy$SklearnModel(n_jobs=1)

  #start_time <- time_py$time()
  t <- bench::mark({yk <- bart_model$fit(train_x,train_y)})
  #time <- time_py$time-start_time
  time <- mean(t$time[[1]])
  predictions <- yk$predict(test_x)
  mse_score <- mean((test_y - predictions)^2)

  ## calculate coverage
  extract <- yk$extract
  model_samples <- extract[[1]][[1]]
  a <- data.frame()
  for (model in model_samples) {
    a <- rbind(a,model$predict(test_x))
  }
  a_new <- unnormalize_x(train_y,a)

  lower_bounds <- apply(a_new, 2, quantile, probs = 0.025)
  upper_bounds <- apply(a_new, 2, quantile, probs = 0.975)

  coverage <- mean(test_y >= lower_bounds & test_y <= upper_bounds)


  return(list(time = time, mse=mse_score,coverage = coverage))
}
```

create evaluation

```r
posterior_mse <- function(fit_results,truth_col,estimate_col){
  y_test = fit_results$truth_col
  pred = fit_results$estimate_col
  return(mean((y_test - pred)^2))
}

pred_err <- create_evaluator(
  .eval_fun = posterior_mse, .name = 'Posterior MSE',
  # additional named parameters to pass to .eval_fun()
  truth_col = "y_test", estimate_col = "predictions"
)
```

model fitting

```r
BART <- create_method(
  .method_fun = BART_fun, .name = "BART",
  # additional named parameters to pass to .method_fun()
  k=2.5,q=0.95,df=4
)
dbarts <- create_method(.method_fun = dbarts_fun,.name = "dbarts",
                        k=2.5,q=0.95,df=4)
bartMachine <- create_method(.method_fun = bartMachine_fun,.name = "bartMachine",
                        k=2.5,q=0.95,df=4)
SoftBart <- create_method(.method_fun = SoftBart_fun,.name = "SoftBart",
                        num_trees=50,alpha=0.95,beta=2)
RF <- create_method(.method_fun = RF_fun,.name = "RandomForest")
bartpy2 <- create_method(.method_fun = bartpy_fun,.name = "bartpy")
# Create experiment
experiment <- create_experiment(name = "Test Experiment") %>%
  add_dgp(linear_dgp) %>%
  add_dgp(dataset_dgp) %>%
  add_method(dbarts) %>%
  add_method(BART) %>%
  add_method(bartMachine) %>%
  add_method(SoftBart) %>%
  add_method(RF)%>%
  add_method(bartpy2)%>%
  add_vary_across(
    .dgp = "Linear DGP",
    noise_sd = c(0.1, 0.5, 1, 2),
    n=c(200,500,1000),
    p=c(4,6,8)
  )
  #add_evaluator(pred_err)



results <- run_experiment(experiment, n_reps = 4, save = TRUE)
```

```
Fitting Test Experiment...

Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
```

```
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
```

```
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
```

```
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
```

```
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
```

```
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.


Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?


Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
```

```
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.

Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?

Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
```

```
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
Warning: Some expressions had a GC in every iteration; so filtering is
disabled.
```

```
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?
Warning in randomForest.default(x = train_X, y = y_train): The response has
five or fewer unique values.  Are you sure you want to do regression?


Saving fit results...
Fit results saved | time taken: 0.039731 seconds
4 reps completed (totals: 4/4) | time taken: 198.093059 minutes
==============================
No evaluators to evaluate. Skipping evaluation.
==============================
No visualizers to visualize. Skipping visualization.
==============================
```

```
# Render automated documentation and view results
#render_docs(experiment)
```

```
result <- results$fit_results
result
```

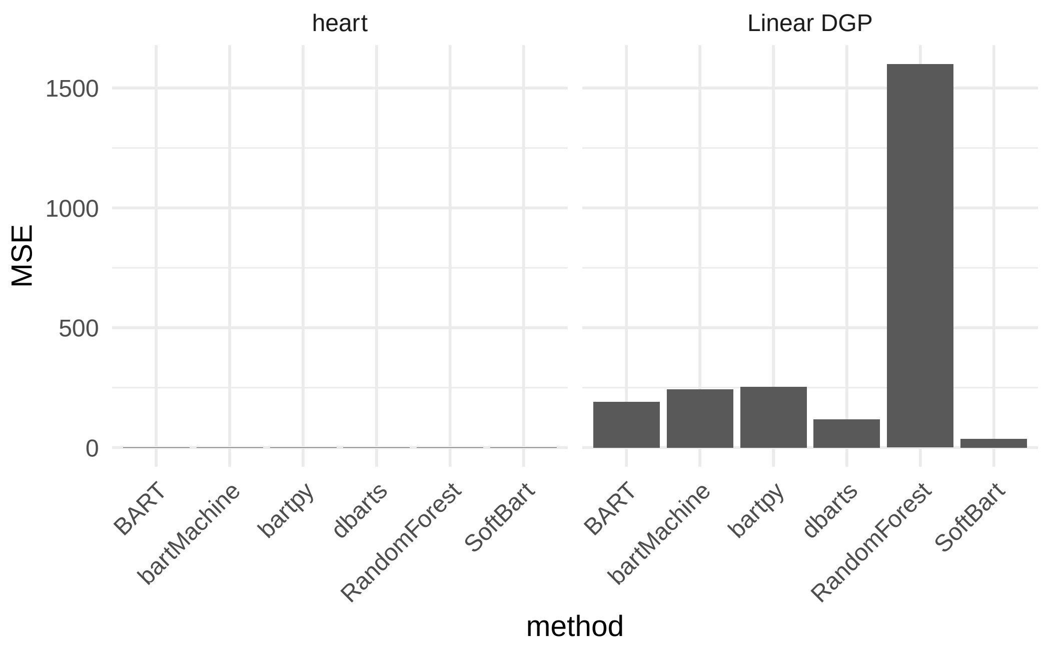
```
# A tibble: 888 x 9
   .rep  .dgp_name  .method_name noise_sd     n     p time           mse coverage
   <chr> <chr>      <chr>           <dbl> <dbl> <dbl> <list>       <dbl>    <dbl>
 1 1     Linear DGP BART              0.1   200     4 <bench_tm> 251.       0.925
 2 1     Linear DGP BART              0.1   200     6 <bench_tm> 546.       0.575
 3 1     Linear DGP BART              0.1   200     8 <bench_tm> 870.       0.4
 4 1     Linear DGP BART              0.1   500     4 <bench_tm>  38.0      0.76
 5 1     Linear DGP BART              0.1   500     6 <bench_tm>  72.2      0.84
 6 1     Linear DGP BART              0.1   500     8 <bench_tm> 137.       0.85
 7 1     Linear DGP BART              0.1  1000     4 <bench_tm>  62.1      0.765
 8 1     Linear DGP BART              0.1  1000     6 <bench_tm>  40.5      0.785
 9 1     Linear DGP BART              0.1  1000     8 <bench_tm>  77.3      0.755
10 1     Linear DGP BART              0.5   200     4 <bench_tm> 194.       0.975
# i 878 more rows
```

```r
result$time_numeric <- as.numeric(result$time)
```

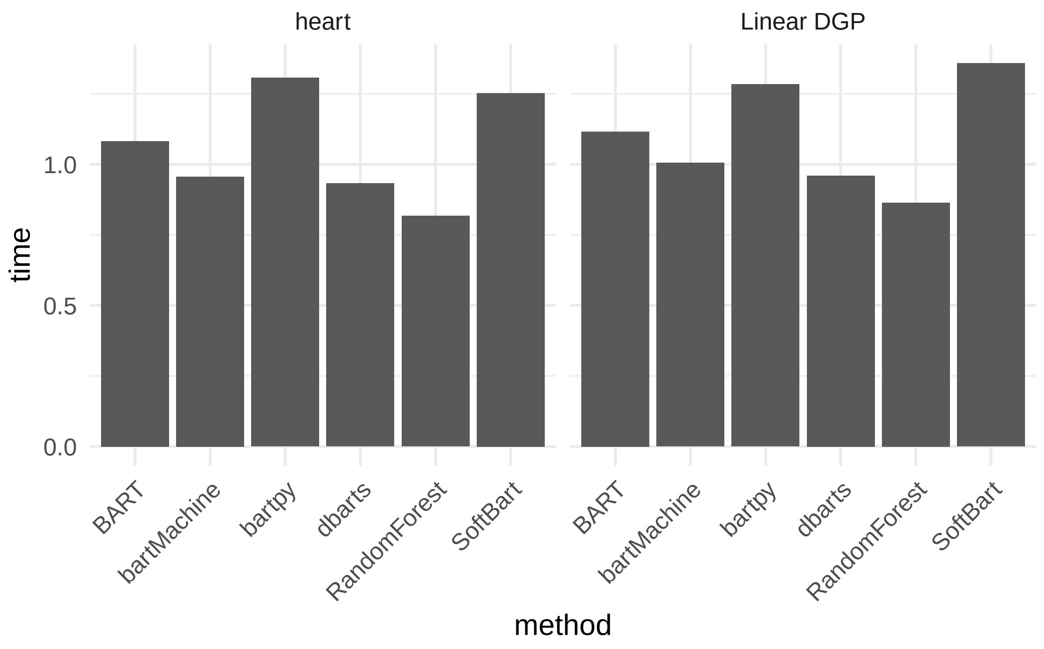
```r
result$Resource <- paste(result$.dgp_name, result$.method_name, sep="_")

# Calculate MSE for each group
summary <- result %>%
  group_by(result$.dgp_name, result$.method_name) %>%
  summarise(
    Mean_MSE = mean(mse),
    Var_MSE = sd(mse),
    Mean_time = mean(time_numeric),
    Var_time = sd(time_numeric),
    Mean_coverage=mean(coverage),
    SD_coverage = sd(coverage),
    .groups = 'keep')

print(summary)
```

```
# A tibble: 12 x 8
# Groups:   result$.dgp_name, result$.method_name [12]
   `result$.dgp_name` `result$.method_name` Mean_MSE  Var_MSE Mean_time Var_time
   <chr>              <chr>                    <dbl>    <dbl>     <dbl>    <dbl>
 1 Linear DGP         BART                   192.     2.22e+2     2.99   1.31
 2 Linear DGP         RandomForest          1599.     1.04e+3     0.232  0.162
 3 Linear DGP         SoftBart                36.8    3.84e+1    21.4   13.8
 4 Linear DGP         bartMachine            243.     2.50e+2     1.05   0.579
 5 Linear DGP         bartpy                 254.     1.91e+2    12.2    0.708
 6 Linear DGP         dbarts                 118.     1.40e+2     0.667  0.230
 7 heart              BART                     0.144  1.70e-2     2.20   0.0386
 8 heart              RandomForest             0.151  2.47e-2     0.133  0.00323
 9 heart              SoftBart                 0.151  2.73e-2     9.55   0.114
10 heart              bartMachine              0.146  1.99e-2     0.638  0.0227
11 heart              bartpy                   0.144  1.85e-2    14.5    0.324
12 heart              dbarts                   1.50   3.78e-1     0.497  0.0102
# i 2 more variables: Mean_coverage <dbl>, SD_coverage <dbl>
```

```r
ggplot(summary, aes(x = `result$.method_name`, y = Mean_MSE
                    #fill = Category
                    )) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+

  labs(y = "MSE", x = "method") +
  facet_grid(~ `result$.dgp_name`)
```

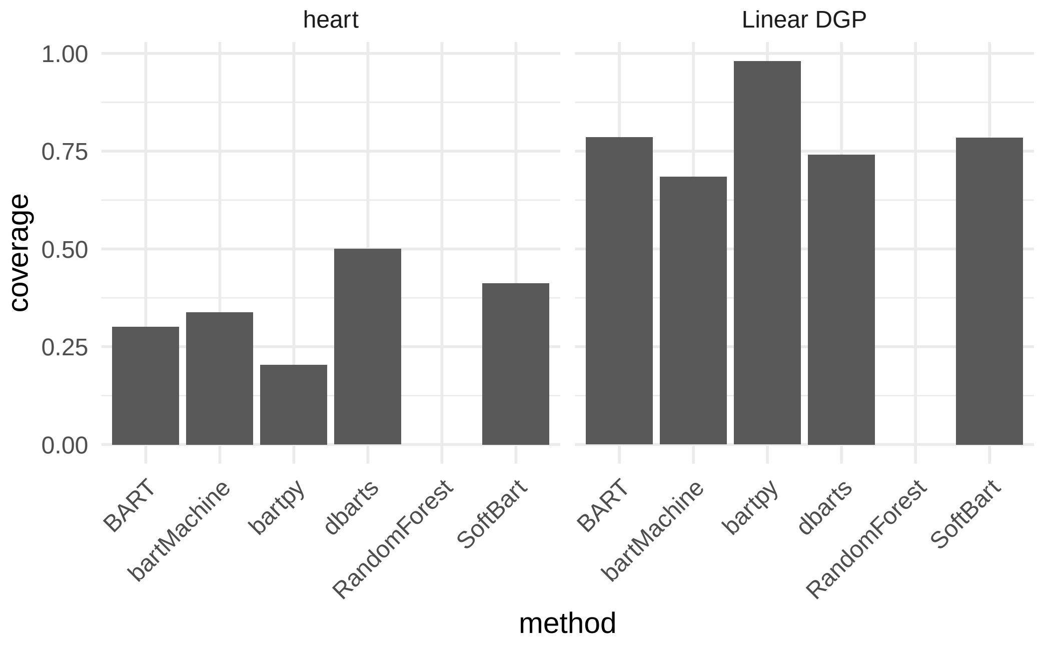


```
ggplot(summary, aes(x = `result$.method_name`, y = Mean_time**(0.1)
                     #fill = Category
                     )) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+

  labs(y = "time", x = "method") +
  facet_wrap(~ `result$.dgp_name`)
```



```
ggplot(summary, aes(x = `result$.method_name`, y = Mean_coverage
                    #fill = Category
                    )) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+

  labs(y = "coverage", x = "method") +
  facet_wrap(~ `result$.dgp_name`)
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_bar()`).
```


heart
Linear DGP
coverage
1.00
0.75
0.50
0.25
0.00
BART
bartMachine
bartpy
dbarts
RandomForest
SoftBart
method