imodel_dataset

create dataset

```
linear_dgp_fun <- function(n_train, n_test, p, beta, noise_sd) {
  set.seed(123)
  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()
  n_train = 350, n_test = 120, p = 4, beta = c(1,2,1.5,3), 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)
```

```r
  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))
}

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
```

```r
  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))
}
```

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")
# 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_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 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 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 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 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 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: 0.051202 seconds
4 reps completed (totals: 4/4) | time taken: 4.716003 minutes
==============================
Evaluating Test Experiment...

Warning: Unknown or uninitialised column: `truth_col`.
```

```
Warning: Unknown or uninitialised column: `estimate_col`.

Evaluation completed | time taken: 0.000032 minutes
Saving eval results...
Eval results saved | time taken: 0.044698 seconds
==============================
No visualizers to visualize. Skipping visualization.
==============================
```

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

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

```
# A tibble: 40 x 6
   .rep  .dgp_name  .method_name time               mse coverage
   <chr> <chr>      <chr>        <list>           <dbl>    <dbl>
 1 1     Linear DGP BART         <bench_tm [1]> 1.35     0.758
 2 1     Linear DGP RandomForest <bench_tm [1]> 3.11    NA
 3 1     Linear DGP SoftBart     <bench_tm [1]> 1.30     0.517
 4 1     Linear DGP bartMachine  <bench_tm [1]> 1.48     0.667
 5 1     Linear DGP dbarts       <bench_tm [1]> 1.34     0.783
 6 1     heart      BART         <bench_tm [1]> 0.144    0.222
 7 1     heart      RandomForest <bench_tm [1]> 0.133   NA
 8 1     heart      SoftBart     <bench_tm [1]> 0.142    0.5
 9 1     heart      bartMachine  <bench_tm [1]> 0.141    0.333
10 1     heart      dbarts       <bench_tm [1]> 1.19     0.593
# i 30 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),
```

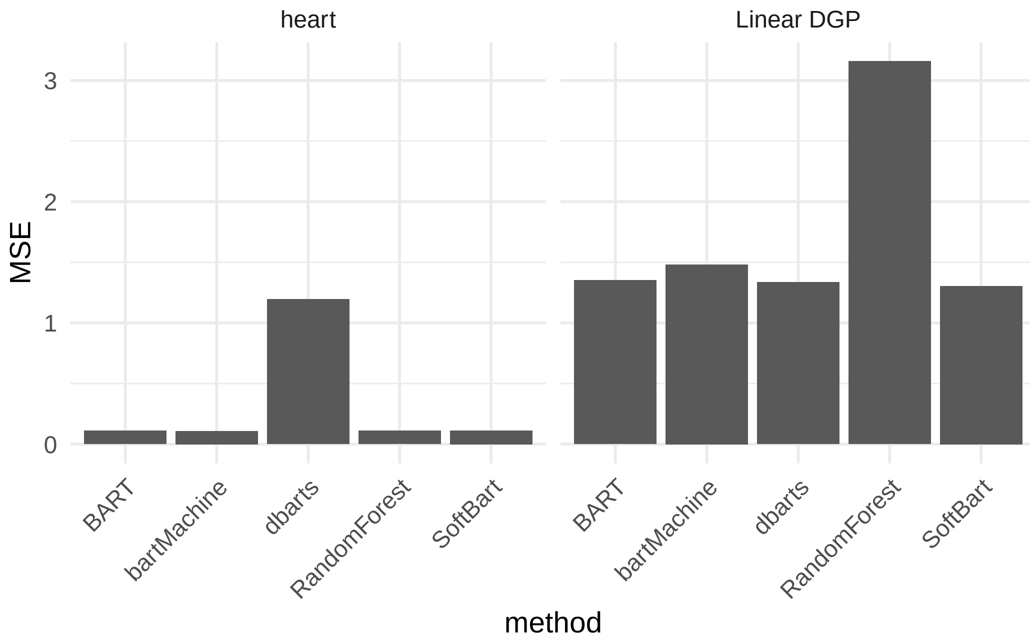
```r
    Var_time = sd(time_numeric),
    Mean_coverage=mean(coverage),
    SD_coverage = sd(coverage),
    .groups = 'keep')

print(summary)
```

```
# A tibble: 10 x 8
# Groups:   result$.dgp_name, result$.method_name [10]
   `result$.dgp_name` `result$.method_name` Mean_MSE Var_MSE Mean_time Var_time
   <chr>              <chr>                    <dbl>   <dbl>     <dbl>    <dbl>
 1 Linear DGP         BART                     1.35  0           2.43   0.0129
 2 Linear DGP         RandomForest             3.16  0.0323      0.129  0.0342
 3 Linear DGP         SoftBart                 1.30  0          14.7    0.114
 4 Linear DGP         bartMachine              1.48  0.0840      0.777  0.0224
 5 Linear DGP         dbarts                   1.34  0           0.623  0.0311
 6 heart              BART                     0.110 0.0223      2.20   0.00964
 7 heart              RandomForest             0.110 0.0154      0.128  0.0158
 8 heart              SoftBart                 0.112 0.0201      9.75   0.0203
 9 heart              bartMachine              0.109 0.0211      0.608  0.0243
10 heart              dbarts                   1.20  0.00475     0.515  0.00582
# i 2 more variables: Mean_coverage <dbl>, SD_coverage <dbl>
```

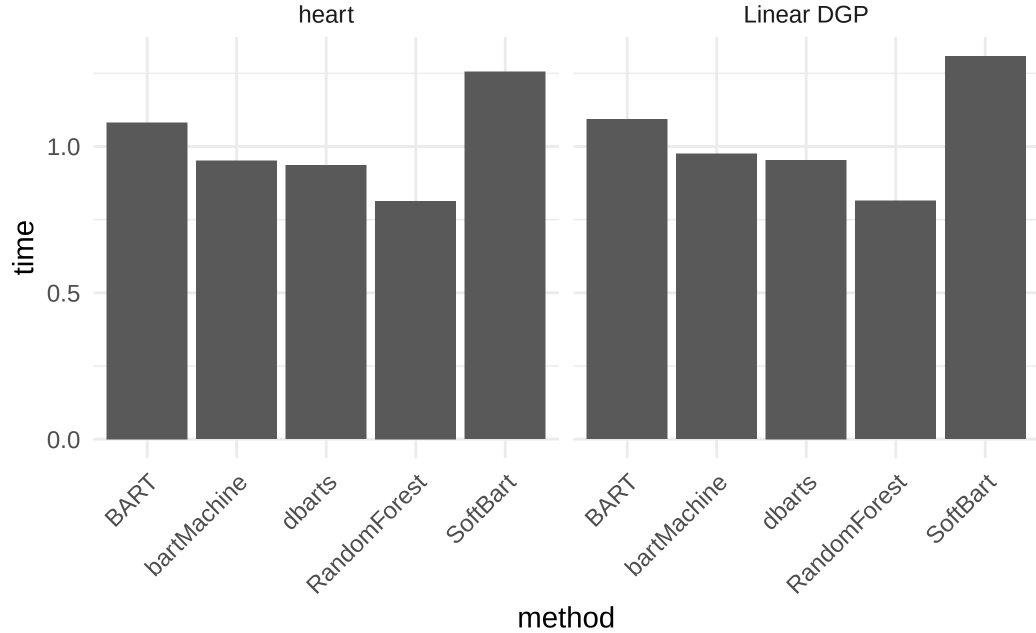
```r
ggplot(summary, aes(x = `result$.method_name`, y = Mean_MSE
                    #fill = Category
                    )) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+

  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`)
```




```
summary$`result$.dgp_name`
```

```
 [1] "Linear DGP" "Linear DGP" "Linear DGP" "Linear DGP" "Linear DGP"
 [6] "heart"      "heart"      "heart"      "heart"      "heart"
```