python in R

python environment setting

```
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

```
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)
```

```r
  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 <- dbarts::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]])
```

```r
  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))
}

stochtree_fun <- function(X_train, y_train, X_test, y_test, q){
  train_X <- data.frame(X_train)
  test_X <- data.frame(X_test)
  t <- bench::mark(bart_model <- stochtree::bart(X_train = train_X,
```

```r
                                       y_train = y_train,
                                       X_test = test_X,
                                       q = q,
                                       num_burnin = 100))
  time <- mean(t$time[[1]])
  predictions <- rowMeans(bart_model$y_hat_test)
  mse_score <- mean((y_test - predictions)^2)

  lower_bounds <- apply(bart_model$y_hat_test, 1, quantile, probs = 0.025)
  upper_bounds <- apply(bart_model$y_hat_test, 1, quantile, probs = 0.975)

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

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

create evaluation

```r
plot_mse <- function(fit_results){
  fit_results$time_numeric <- as.numeric(fit_results$time)

# Calculate MSE for each group
  summary <- fit_results %>%
    group_by(fit_results$.dgp_name, fit_results$.method_name,n,p,noise_sd) %>%
    summarise(
      Mean_MSE = mean(mse),
      Var_MSE = sd(mse),

      .groups = 'keep')
  plt <- ggplot(summary, aes(x = `fit_results$.method_name`, y = Mean_MSE
                    #fill = Category
                    )) +
    geom_bar(stat = "identity") +
    facet_grid(~n+p+noise_sd)+
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))+

    labs(y = "MSE", x = "method")
    facet_wrap(~ `fit_results$.dgp_name`)
  return(plt)
```

```r
  }

plot_time <- function(fit_results){
  fit_results$time_numeric <- as.numeric(fit_results$time)

# Calculate MSE for each group
  summary <-  fit_results %>%
    group_by(fit_results$.dgp_name, fit_results$.method_name,n,p,noise_sd) %>%
    summarise(
      Mean_time = mean(time_numeric),
      Var_time = sd(time_numeric),
      )

  plt <- ggplot(summary, aes(x = `fit_results$.method_name`, y = Mean_time
                    #fill = Category
                    )) +
    geom_bar(stat = "identity") +

    facet_wrap(~n+p+noise_sd) +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))+
    labs(y = "time", x = "method")
    facet_wrap(~ `fit_results$.dgp_name`)
  return(plt)
  }

plot_coverage <- function(fit_results){
  fit_results$time_numeric <- as.numeric(fit_results$time)

# Calculate MSE for each group
    summary <- fit_results %>%
    group_by(fit_results$.dgp_name, fit_results$.method_name,n,p,noise_sd) %>%
    summarise(

      Mean_coverage=mean(coverage),
      SD_coverage = sd(coverage))

  plt <- ggplot(summary, aes(x = `fit_results$.method_name`, y = Mean_coverage)) +
    geom_bar(stat = "identity") +

    facet_grid(~n+p+noise_sd)+
    theme_minimal() +
```

```
    theme(axis.text.x = element_text(angle = 45, hjust = 1))+
    labs(y = "coverage", x = "method")
    facet_wrap(~ `fit_results$.dgp_name`)
  return(plt)
}

coverage_plot <- create_visualizer(
  .viz_fun = plot_coverage, .name = 'coverage Plot',
  # additional named parameters to pass to .viz_fun()

)

time_plot <- create_visualizer(
  .viz_fun = plot_time, .name = 'time Plot',
  # additional named parameters to pass to .viz_fun()

)
mse_plot <- create_visualizer(
  .viz_fun = plot_mse, .name = 'MSE Plot',
  # additional named parameters to pass to .viz_fun()

)
```

model fitting

```
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")
stochtree <- create_method(.method_fun = stochtree_fun, .name = "stochtree",q=0.95)
# Create experiment
```

```r
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_method(stochtree)%>%
  add_visualizer(mse_plot)%>%
  add_visualizer(time_plot)%>%
  add_visualizer(coverage_plot)%>%
  add_vary_across(
    .dgp = "Linear DGP",
    noise_sd = c(0.1, 0.5),
    n=c(200),
    p=c(4,6)
  )
  #add_evaluator(pred_err)



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

```
Fitting Test Experiment...
Saving fit results...
Fit results saved | time taken: 0.112590 seconds
4 reps completed (totals: 4/4) | time taken: 12.458266 minutes
==============================
No evaluators to evaluate. Skipping evaluation.
==============================
Visualizing Test Experiment...
`summarise()` has grouped output by 'fit_results$.dgp_name', 'fit_results$.method_name', 'n'
`summarise()` has grouped output by 'fit_results$.dgp_name', 'fit_results$.method_name', 'n'
Visualization completed | time taken: 0.001795 minutes
Saving viz results...
Viz results saved | time taken: 0.092358 seconds
==============================
```

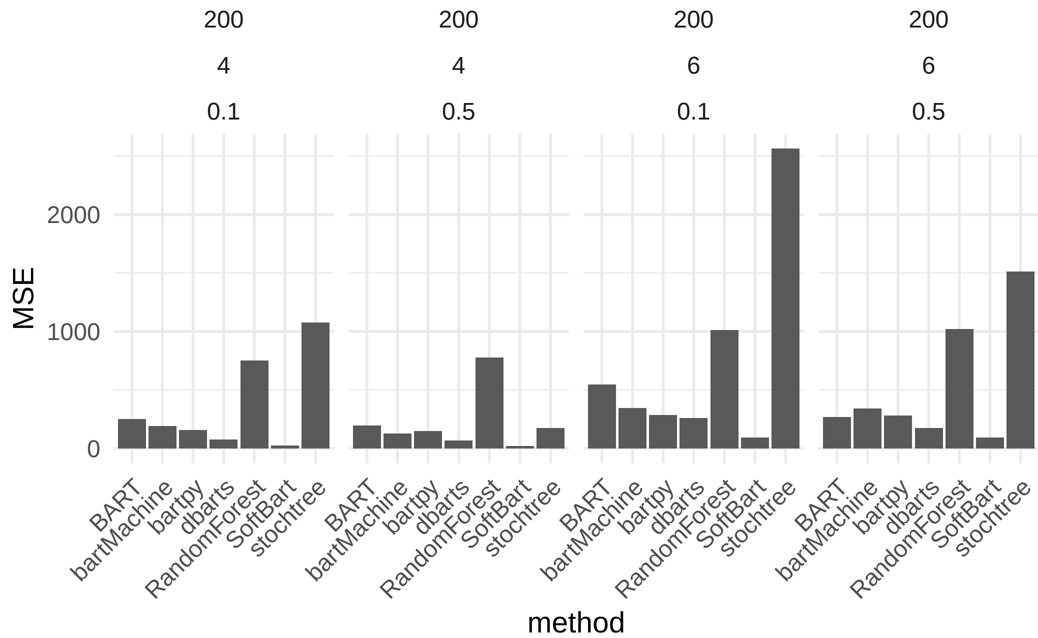
```
# Render automated documentation and view results
#render_docs(experiment)
```

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

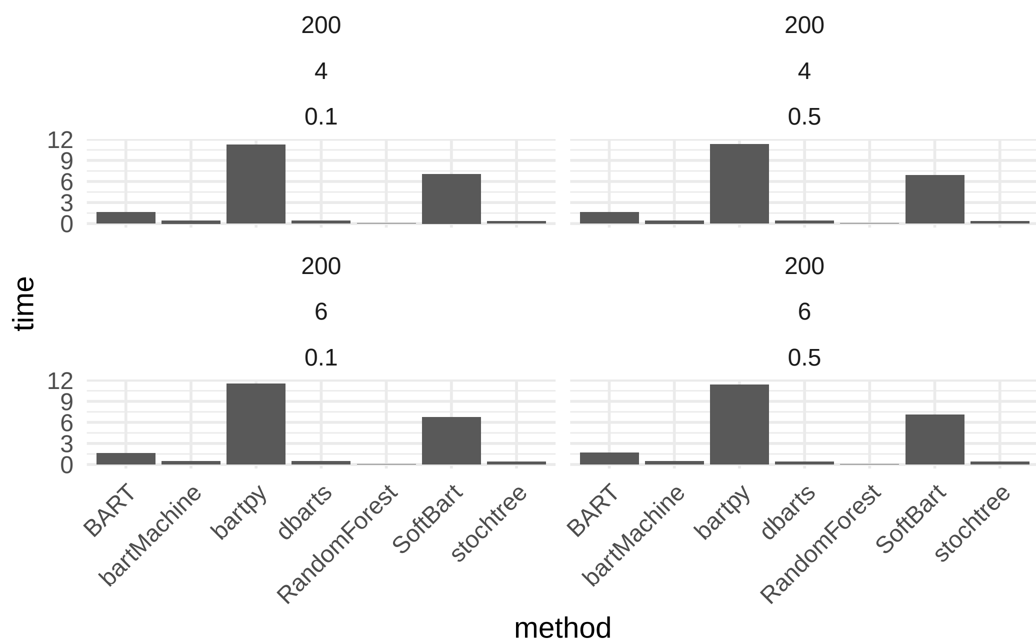
```
# A tibble: 112 x 9
   .rep  .dgp_name  .method_name noise_sd     n     p time           mse coverage
   <chr> <chr>      <chr>           <dbl> <dbl> <dbl> <list>       <dbl>    <dbl>
 1 1     Linear DGP BART              0.1   200     4 <bench_tm>  251.      0.925
 2 1     Linear DGP BART              0.1   200     6 <bench_tm>  546.      0.575
 3 1     Linear DGP BART              0.5   200     4 <bench_tm>  194.      0.975
 4 1     Linear DGP BART              0.5   200     6 <bench_tm>  269.      0.725
 5 1     Linear DGP RandomForest      0.1   200     4 <bench_tm>  700.     NA
 6 1     Linear DGP RandomForest      0.1   200     6 <bench_tm> 1024.     NA
 7 1     Linear DGP RandomForest      0.5   200     4 <bench_tm>  771.     NA
 8 1     Linear DGP RandomForest      0.5   200     6 <bench_tm> 1024.     NA
 9 1     Linear DGP SoftBart          0.1   200     4 <bench_tm>   24.5     0.775
10 1     Linear DGP SoftBart          0.1   200     6 <bench_tm>   99.2     0.75
# i 102 more rows
```

```
results$viz_results
```

```
$`MSE Plot`
```

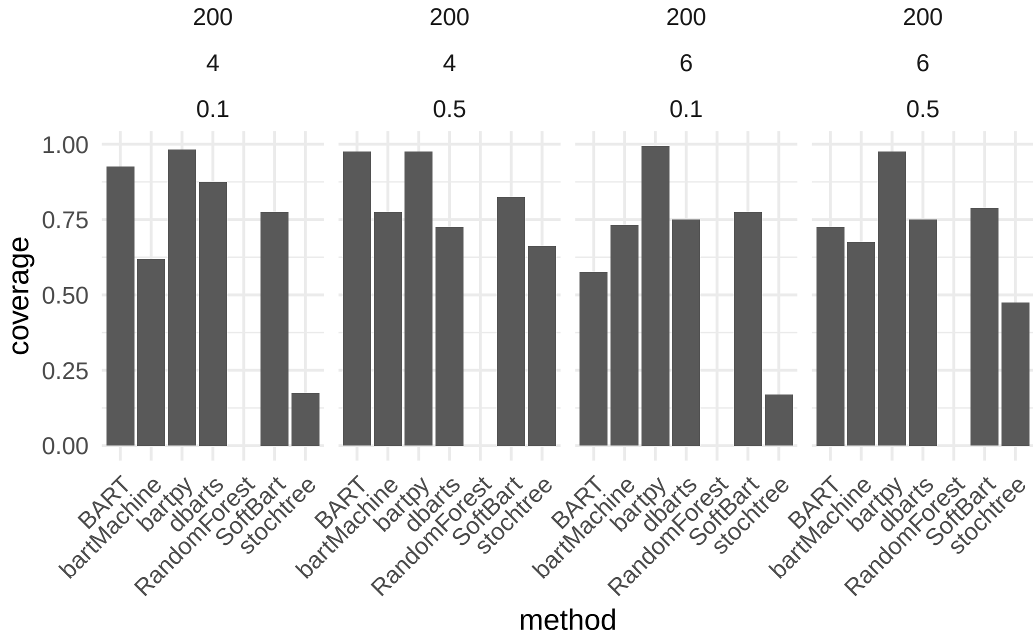

200
4
0.1
200
4
0.5
200
6
0.1
200
6
0.5
MSE
2000
1000
0
BART
bartMachine
bartpy
dbarts
RandomForest
SoftBart
stochtree
method

$`time Plot`


200
4
0.1
200
4
0.5
200
6
0.1
200
6
0.5
time
12
9
6
3
0
BART
bartMachine
bartpy
dbarts
RandomForest
SoftBart
stochtree
method

```
$`coverage Plot`
```

```
Warning: Removed 4 rows containing missing values or values outside the scale range
(`geom_bar()`).
```




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

# Calculate MSE for each group
summary <- result %>%
  group_by(result$.dgp_name, result$.method_name,n,p,noise_sd) %>%
  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: 28 x 11
# Groups:   result$.dgp_name, result$.method_name, n, p, noise_sd [28]
   `result$.dgp_name` `result$.method_name`     n     p noise_sd Mean_MSE
   <chr>              <chr>                 <dbl> <dbl>    <dbl>    <dbl>
 1 Linear DGP         BART                    200     4      0.1    251.
 2 Linear DGP         BART                    200     4      0.5    194.
 3 Linear DGP         BART                    200     6      0.1    546.
 4 Linear DGP         BART                    200     6      0.5    269.
 5 Linear DGP         RandomForest            200     4      0.1    751.
 6 Linear DGP         RandomForest            200     4      0.5    777.
 7 Linear DGP         RandomForest            200     6      0.1   1012.
 8 Linear DGP         RandomForest            200     6      0.5   1023.
 9 Linear DGP         SoftBart                200     4      0.1     24.5
10 Linear DGP         SoftBart                200     4      0.5     17.9
# i 18 more rows
# i 5 more variables: Var_MSE <dbl>, Mean_time <dbl>, Var_time <dbl>,
#   Mean_coverage <dbl>, SD_coverage <dbl>
```

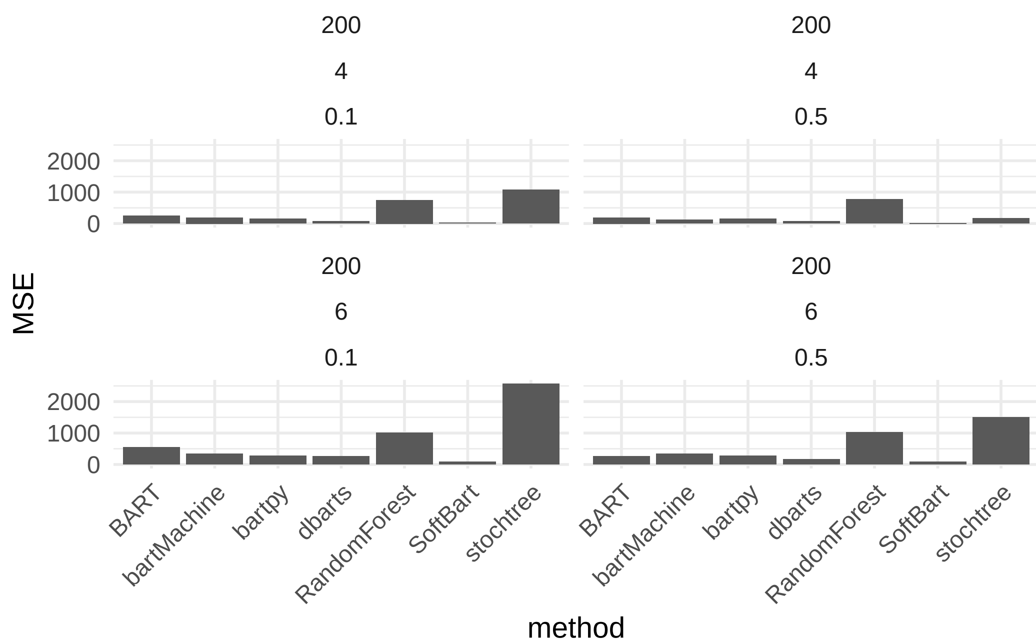
```r
plt <- ggplot(summary, aes(x = `result$.method_name`, y = Mean_MSE
                    #fill = Category
                    )) +
  geom_bar(stat = "identity") +
  facet_wrap(~n+p+noise_sd)+
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+

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

```
<ggproto object: Class FacetGrid, Facet, gg>
    compute_layout: function
    draw_back: function
    draw_front: function
    draw_labels: function
    draw_panels: function
    finish_data: function
    init_scales: function
    map_data: function
    params: list
    setup_data: function
    setup_params: function
    shrink: TRUE
```

```
    train_scales: function
    vars: function
    super:  <ggproto object: Class FacetGrid, Facet, gg>
```

```r
plt
```



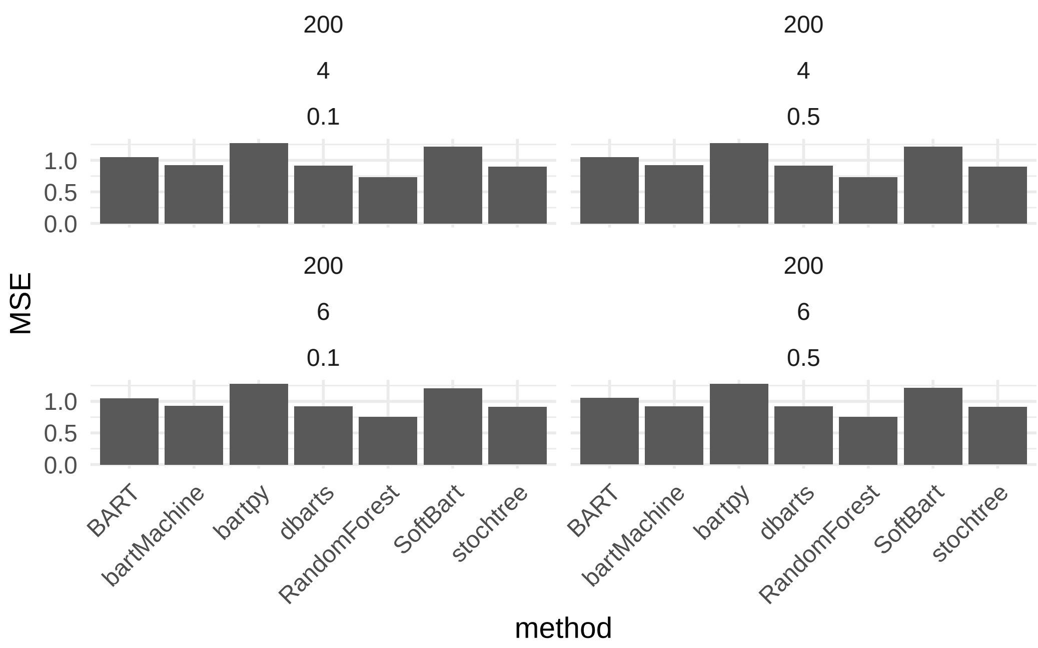

```r
plt <- ggplot(summary, aes(x = `result$.method_name`, y = Mean_time**(0.1)
                    #fill = Category
                    )) +
  geom_bar(stat = "identity") +
  facet_wrap(~n+p+noise_sd)+
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+

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

```
<ggproto object: Class FacetGrid, Facet, gg>
    compute_layout: function
    draw_back: function
    draw_front: function
    draw_labels: function
```

```
draw_panels: function
finish_data: function
init_scales: function
map_data: function
params: list
setup_data: function
setup_params: function
shrink: TRUE
train_scales: function
vars: function
super:  <ggproto object: Class FacetGrid, Facet, gg>
```

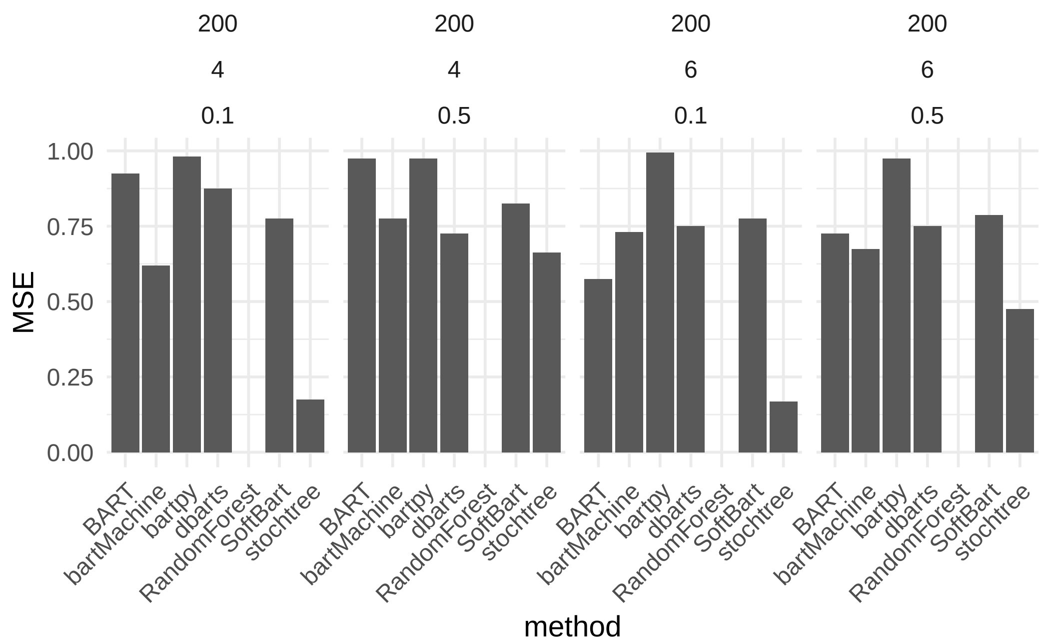
```
plt
```




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

```
labs(y = "MSE", x = "method")
```

```
Warning: Removed 4 rows containing missing values or values outside the scale range
(`geom_bar()`).
```




```
facet_grid(~ `result$.dgp_name`)
```

```
<ggproto object: Class FacetGrid, Facet, gg>
    compute_layout: function
    draw_back: function
    draw_front: function
    draw_labels: function
    draw_panels: function
    finish_data: function
    init_scales: function
    map_data: function
    params: list
    setup_data: function
    setup_params: function
    shrink: TRUE
```

```
train_scales: function
vars: function
super:  <ggproto object: Class FacetGrid, Facet, gg>
```