Demo

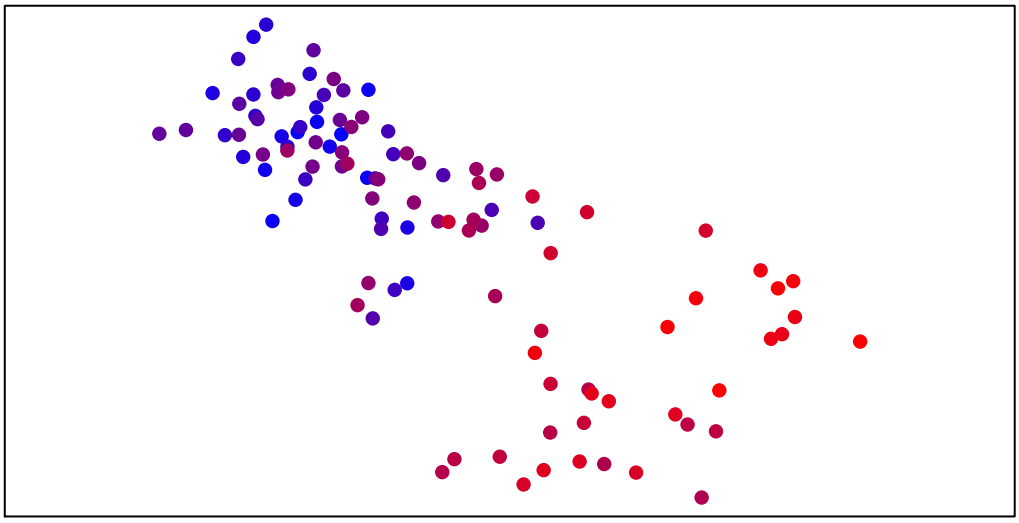
Loading the data

The dataset we will be using is a bivaraite dataset with a phenotype (numeric) with 107 samples. The goal is to access whether or not there is dependency between the two variables after the phenotype has been accounted for. This dataset was subsetted from the BrainSpan (microarray) dataset, so we do not have access to whether or not they are dependent in truth.

```
library(statCognition)
dat <- statCognition::dat
```

For purely illustrative purposes, let's plot the data so we know what it looks like. Typically (in an actual usage), we would discourage the user from looking at the data in any way prior to completing the system. We code the phenotype `Days` so large values are red and small values are blue.

```
compute_color <- function(x){
  sapply(1:nrow(x$pheno), function(y){
    rgb(min(max((x$pheno[y,1])/107,0),1), 0, min(max((107-x$pheno[y,1])/107,0), 1))
  })
}

plot(dat$mat[,1:2], xlab = "", ylab = "", pch = 16, yaxt = "n", xaxt = "n", asp = T,
     col = compute_color(dat))
```



Setting up the system

Let us decide what the stages of our analysis will be as well as the actions and state functions for each stage. We will consider 3 stages: the first stage to remove phenotype effects, the second stage to remove outliers, and the third stage to estimate whether or not there is dependency.

```
action_ll <- vector("list", 3)
action_ll[[1]] <- list(RC_none = RC_none, RC_linear_regression = RC_linear_regression,
                       RC_pairing_difference = RC_pairing_difference)
action_ll[[2]] <- list(SS_none = SS_none, SS_cook = SS_cook,
                       SS_neighborhood = SS_neighborhood)
```

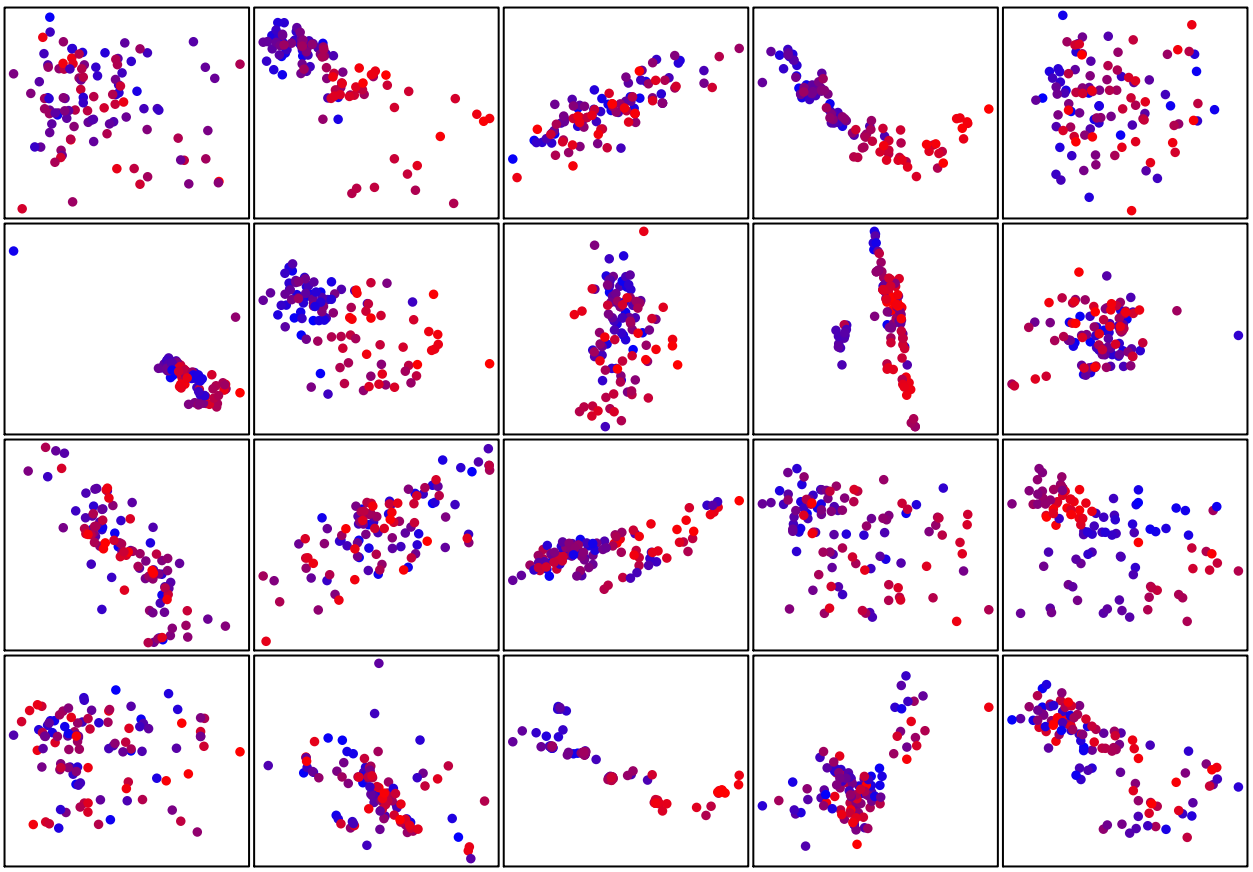
```
action_ll[[3]] <- list(PD_pearson = PD_pearson, PD_kendall = PD_kendall,
                       PD_energy = PD_energy)
names(action_ll) <- c("Remove_confounder", "Sample_selection", "Pairwise_dependency")
```

```
state_ll <- vector("list", 3)
state_ll[[1]] <- list(state_pheno_MI = state_pheno_MI,
                      state_pheno_residual = state_pheno_residual)
state_ll[[2]] <- list(state_variance = state_variance,
                      state_interpoint = state_interpoint)
state_ll[[3]] <- list(state_samples = state_samples, state_linearity = state_linearity)
names(state_ll) <- c("Phenotype", "Outlier", "Dependency")
```

Creating synthetic data

For the system to learn our preferences on how to analyze data, we'll need to generate synthetic data. Let's generate 20 synthetic datasets and plot them all. Each synthetic dataset is labeled by a seed, so the cognition system can reconstruct exactly the same synthetic datasets without you explicitly needing to store them.

```
syn_init <- synthetic_initializer(lambda = 4)
syn_list <- vector("list", 20)
seed_vec <- rep(0, 20)

par(mfrow = c(4,5), mar = rep(0.1, 4))
for(i in 1:20){
  set.seed(i)
  res <- synthetic_generator(dat, syn_init)
  plot(res$mat[,1:2], xlab = "", ylab = "", pch = 16, yaxt = "n", xaxt = "n", asp = T,
       col = compute_color(res))

  syn_list[[i]] <- res
  seed_vec[i] <- get_seed(res)
}
```



```
seed_vec
```

```
##  [1] 266 185 168 586 200 606 989 466 222 507 277  69 710 254 602 683 155
## [18] 823 117 878
```

Recording the pipelines

After some investigation, we record the results

```
response_vec <- c(1,2,1,   2,1,3,  1,1,1,   2,1,3,  1,1,1,
                  1,3,3,   2,1,1,  1,1,1,   3,2,1,  1,2,1,
                  1,1,3,   1,1,1,  3,1,1,   1,1,1,  3,2,1,
                  1,1,1,   1,3,3,  2,3,3,   3,1,1,  1,1,3)
```

Now we feed them into the statistical cognition system.

```
cog_init <- stat_cognition_initializer(action_ll = action_ll,
                                                    state_ll = state_ll,
                                                    generator_init = syn_init)

res_cog <- stat_cognition(dat, cog_init, seed_vec, response_vec, store = T)
```

Now we can evaluate the value functions.

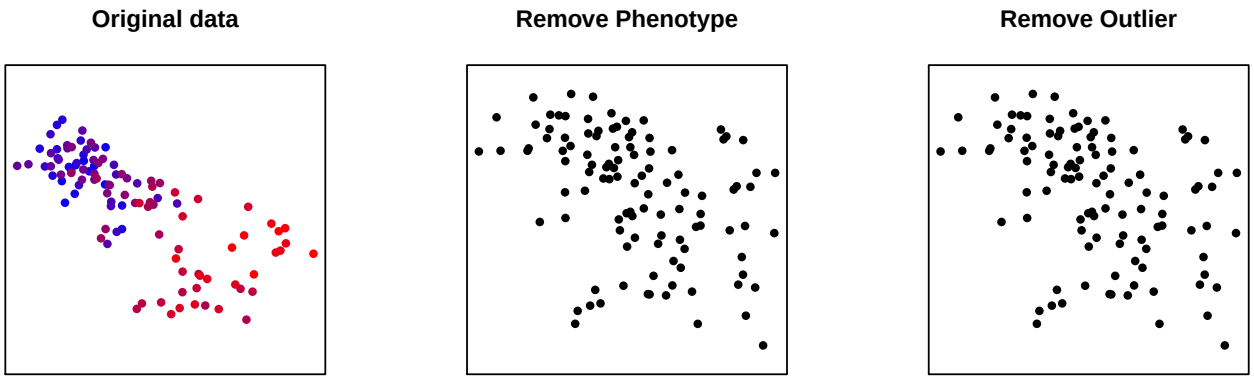
```
result <- evaluate(res_cog, dat)
result
```

```
## $pipeline
```

```
## RC_linear_regression              SS_none            PD_energy
##                    2                    1                    3
##
## $result
## [1] TRUE
##
## attr(,"class")
## [1] "stat_cognition_result"
```

We see that our pipeline returns that there is a dependency between the two variables. Here is how the learned pipelined analyzed the data.

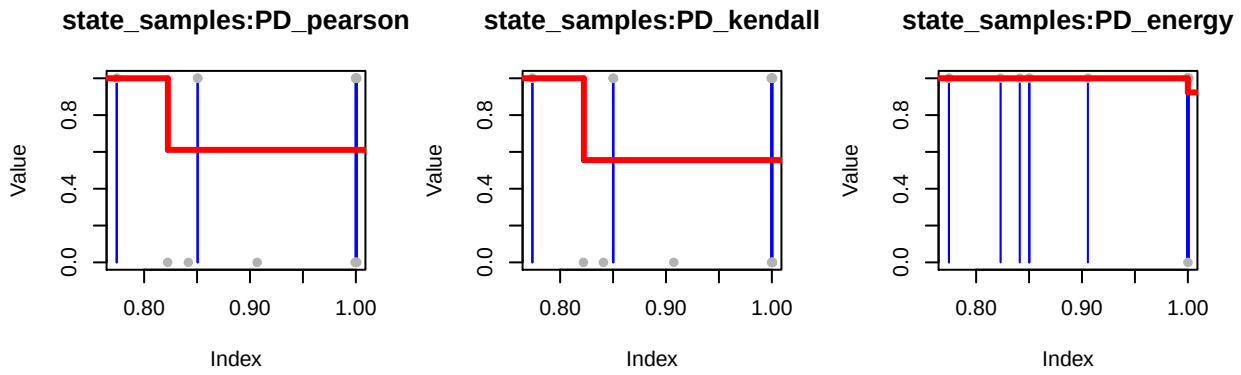
```
par(mfrow = c(1,3))

plot(dat$mat[,1:2], xlab = "", ylab = "", pch = 16, yaxt = "n", xaxt = "n", asp = T,
     col = compute_color(dat), main = "Original data")
dat2 <- action_ll[[1]][[result$pipeline[1]]](dat)
plot(dat2$mat[,1:2], xlab = "", ylab = "", pch = 16, yaxt = "n", xaxt = "n", asp = T,
     main = "Remove Phenotype")
dat2 <- action_ll[[2]][[result$pipeline[2]]](dat2)
plot(dat2$mat[,1:2], xlab = "", ylab = "", pch = 16, yaxt = "n", xaxt = "n", asp = T,
     main = "Remove Outlier")
```

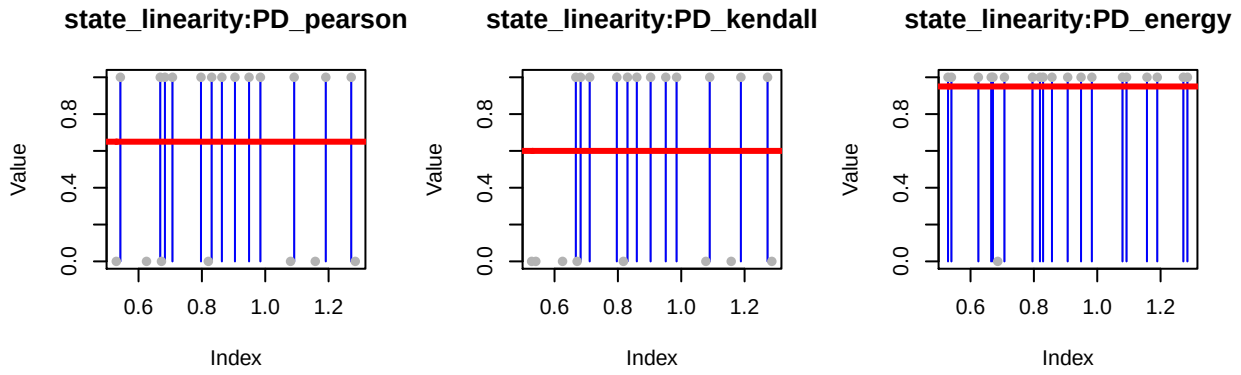


Interpreting the value function

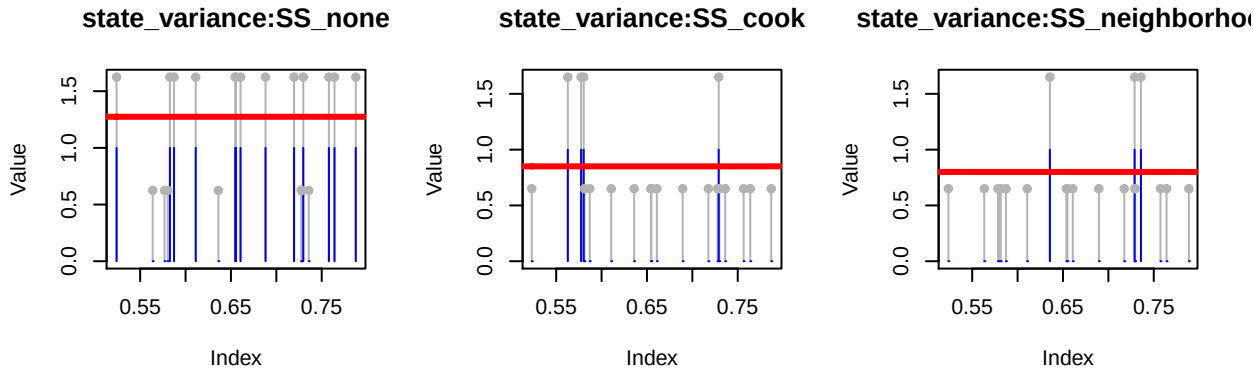
We can plot the contribution functions. We start with the last one.

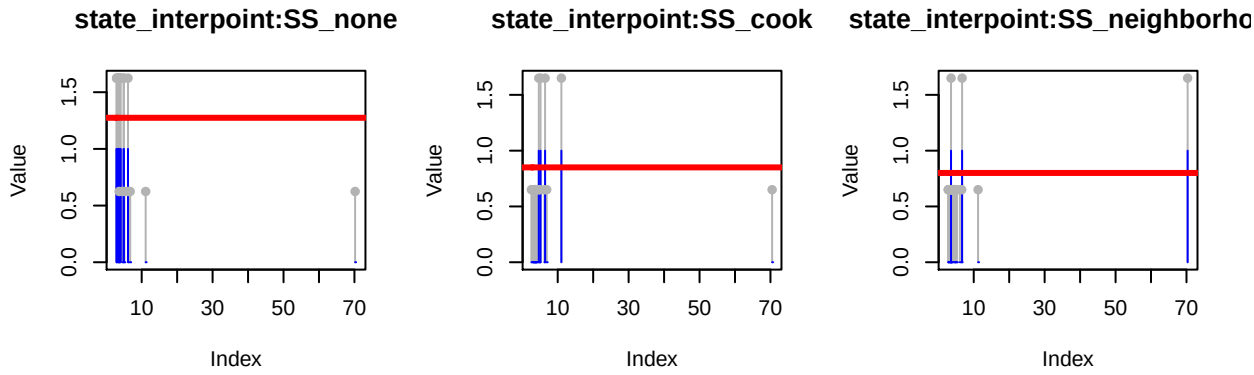
```
plot(res_cog$value_list[[3]], samples = T, contribution = T, ylab = "Value", pch = 16)
```



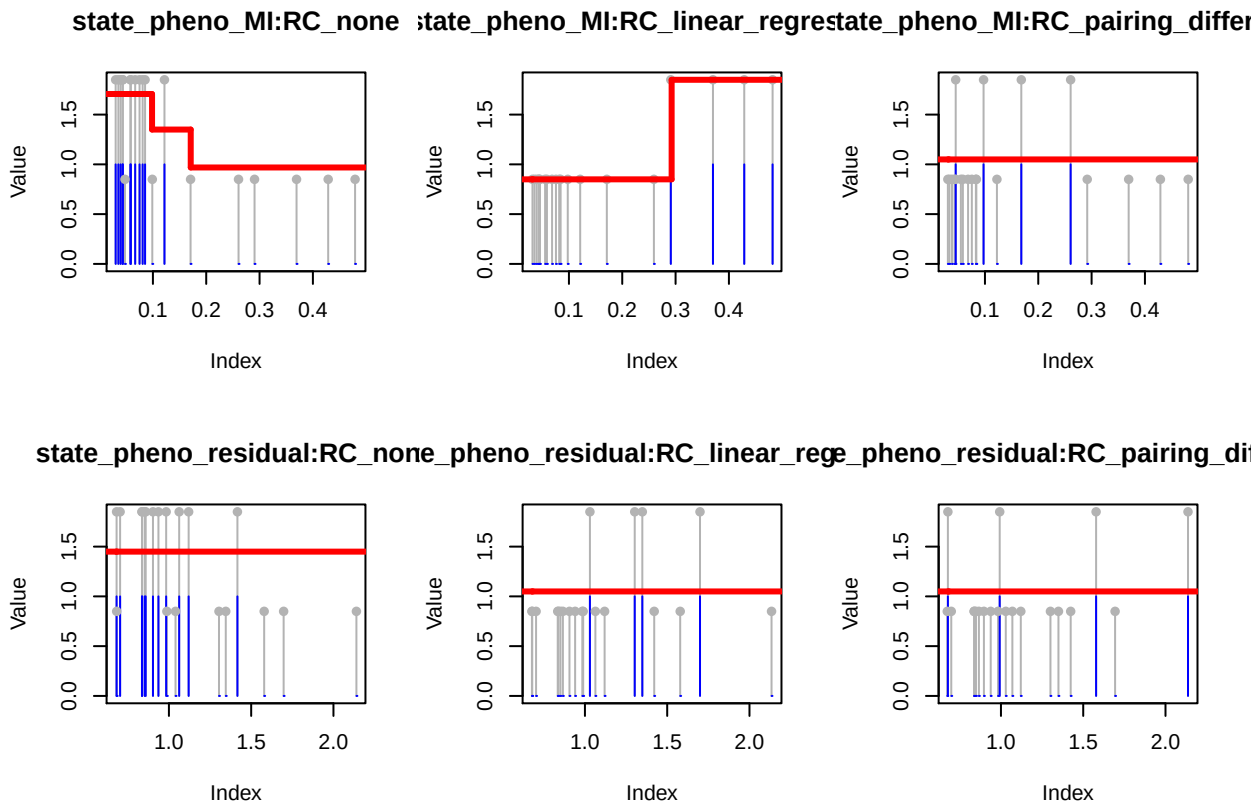



```
plot(res_cog$value_list[[2]], samples = T, contribution = T, ylab = "Value", pch = 16)
```





```
plot(res_cog$value_list[[1]], samples = T, contribution = T, ylab = "Value", pch = 16)
```



We can also plot each individual contribution function.

```
par(mfrow = c(1,3), mar = rep(1,4))
plot(res_cog$value_list[[3]], type = "contribution_surface")
```