Shapley Values & Principal Component Analysis

Lars H. B. Olsen

2024-01-22

Idea

In this file we look at the idea about using Principal Component Analysis to speed up the computations of the Shapley value computations.

The traditional setup

1. We are given a model $f(x)$ and the corresponding training set $\{X_{\text{train}}, y_{\text{train}}\}$.
   - We assume that there are some dependency structure between the M features. For example, let us assume that $x \sim \mathcal{N}_M(\mu, \Sigma)$, where we for the moment let $\mu = \mathbf{0}$ and Σ being 1 on the diagonal and $\rho = 0.7$ off-diagonal.
2. We are asked to explain the prediction of $f(x^*)$ for a new observation/explicand x^*.
3. Either we can compute the conditional Shapley value explanations $\boldsymbol{\phi}$ using the `shapr` library where we specify that the `approach = "gaussian"`, meaning that we assume that data follows a multivariate Gaussian distributions. Denote the corresponding Shapley values for $\phi_{\boldsymbol{x}}$. Conditional Shapley values can be/are computationally expensive to compute.

The new setup

1. The new idea is to use principal component analysis to transform the features $\boldsymbol{x}$ into independent features $\boldsymbol{z}$ and then use marginal Shapley values, i.e., `approach = "independence"` in `shapr`. Marginal Shapley values are faster to compute as they create the Monte Carlo samples by sampling observations from th training data.
   - That is, we have a the principal component function $h : \boldsymbol{x} \mapsto \boldsymbol{z}$, where $\boldsymbol{z}$ are the principal components and h is a linear function. Linear in the sense that $\boldsymbol{z} = h(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$. Here $\boldsymbol{A}$ is a square matrix of dimension $M \times M$ and is the rotation matrix (note that `scale` and `center` will influence the PCA transformation). The function h also has an inverse $h^{-1} : \boldsymbol{z} \mapsto \boldsymbol{x}$.
2. In the first place, we will use all M principal components. Later we can try to reduce it.
3. Instead of letting the contribution/reward function be $v(\mathcal{S}) = \mathbb{E}[f(\boldsymbol{x})|\boldsymbol{x}_{\mathcal{S}}] = \mathbb{E}[f(\{\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}}\})|\boldsymbol{x}_{\mathcal{S}}]$, we compute $\tilde{v}(\mathcal{S}) = \mathbb{E}[f(h^{-1}(\boldsymbol{z}))|\boldsymbol{z}_{\mathcal{S}}] = \mathbb{E}[f(h^{-1}(\{\boldsymbol{z}_{\bar{\mathcal{S}}}, \boldsymbol{z}_{\mathcal{S}}\}))|\boldsymbol{z}_{\mathcal{S}}] = \mathbb{E}[f(\{\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}}\})|\boldsymbol{z}_{\mathcal{S}}]$.
4. Then we compute the corresponding Shapley values $\phi_{\boldsymbol{z}}$ for the $\boldsymbol{z}$ features/principal components.
5. The idea is then to transform $\phi_{\boldsymbol{z}}$ into $\phi_{\boldsymbol{x}}$ using h^{-1}.
   - I think the problem is here. Why should the transformations on the feature space also be valid for the Shapley value space.
   - Indirectly, it think we are saying that $\phi_{\boldsymbol{z}}$ are independent and when we apply the h^{-1}, we reintroduce the dependence structure that $\boldsymbol{x}$ had (?)

Conclusion

I might have done something wrong in the code or misunderstood some ideas. But I do not get the same Shapley values with the two procedures. I am open for setting up a meeting and discussing this more.

I am unsure why using h^{-1}, which is $h^{-1} : \boldsymbol{z} \mapsto \boldsymbol{x}$, should also work on $\phi_{\boldsymbol{z}} \mapsto \phi_{\boldsymbol{x}}$. I feel like these are two different spaces.

Also, due to the efficiency axiom, we have that $\sum_{j=1}^{M} \phi^*_{\boldsymbol{z},j} = f(x^*)$. But when we are sending $\phi_{\boldsymbol{z}}$ through h^{-1} to get the Shapley values in the original feature space we do NOT have that the efficiency axiom still holds.

Is there any way to ensure that it still holds?

Maybe if we assume that f is linear that we are able to derive some results. Since both the PC transformation and the model is linear.

Code

Load the needed libraries. The github branch is a branch where I have altered the `shapr` code to transform the principal components back to original feature space before sending the Monte Carlo samples to the model.

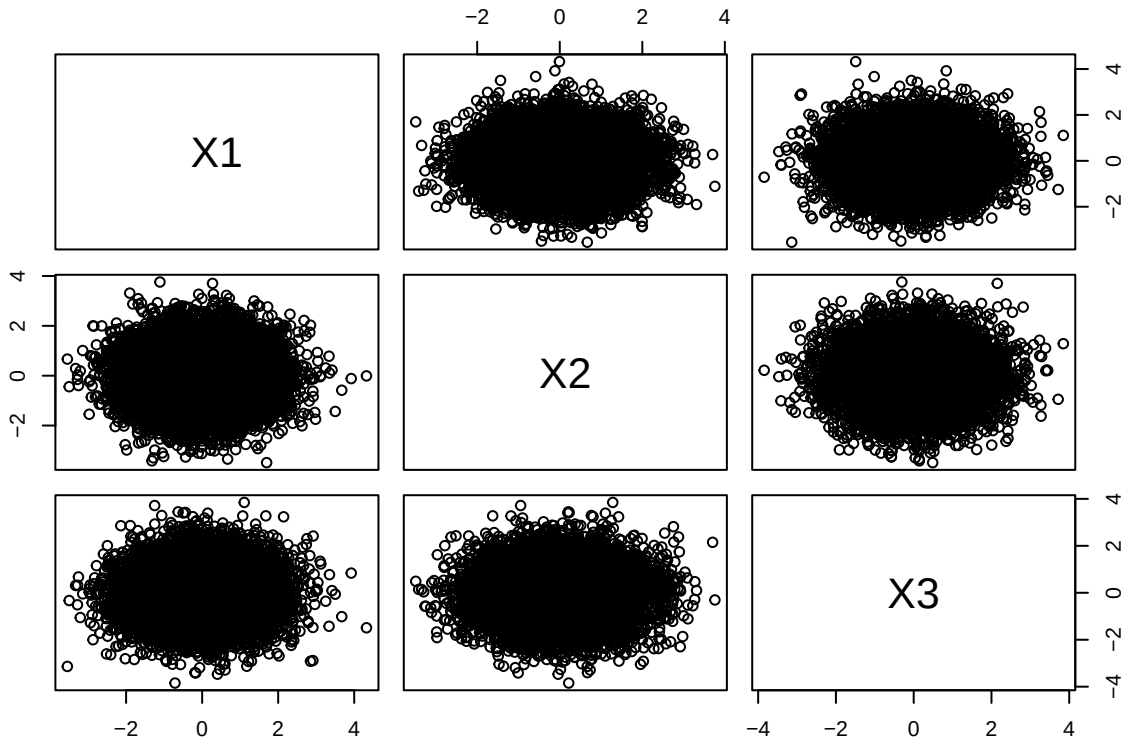
```
library(MASS)
library(data.table)
library(progressr)
suppressMessages(remotes::install_github("LHBO/shapr", ref = "Lars/PCA_SHAP"))
library(shapr)
# devtools::load_all(".")
```

We first set the parameters of our simulation experiment.

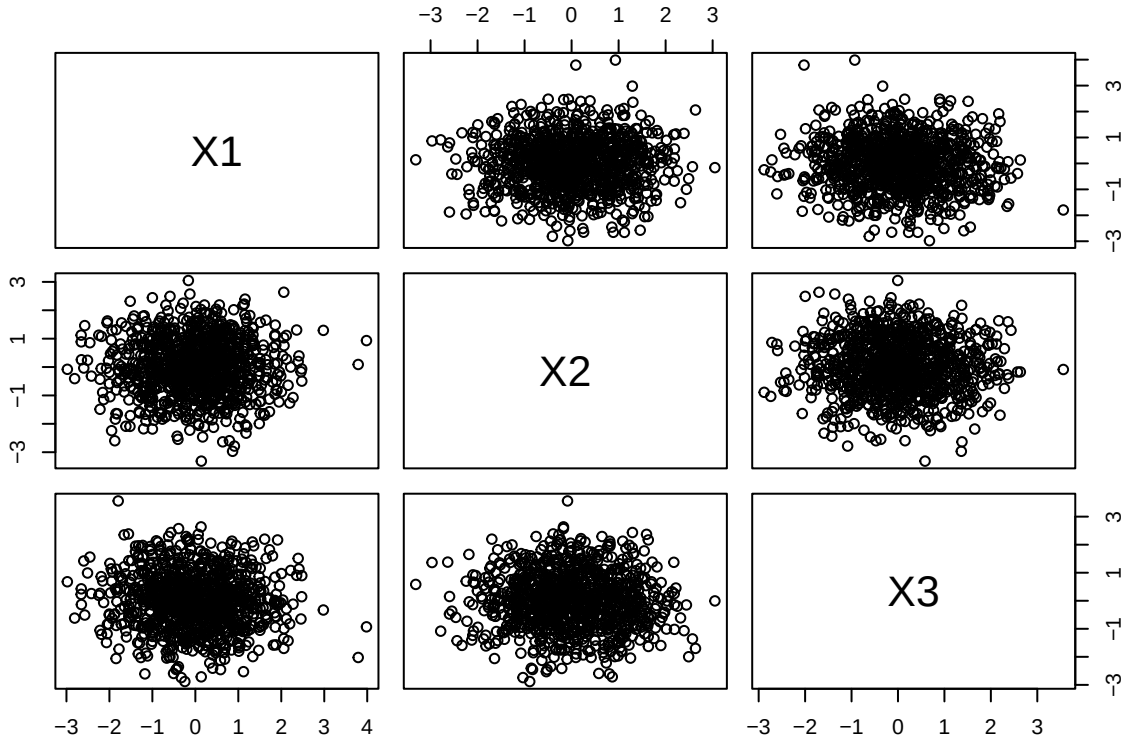
```
# Set seed for reproducibility
set.seed(123)

# Parameters
n_train <- 10000 # Number of training samples
n_explain <- 1000 # Number of explicands
n_features <- 3 # Number of features

# Mean and covariance matrix for the multivariate Gaussian distribution
mean_vector <- rep(0, n_features)
rho <- 0.00
cov_matrix <- matrix(rho, nrow = n_features, ncol = n_features)
diag(cov_matrix) <- 1

# Another option
# cov_matrix <-
#   matrix(c(1, 0.5, 0.75, 0.5, 1, 0.2, 0.75, 0.2, 1), nrow = n_features, byrow = TRUE)
```

The we create the multivariate Gaussian distributed features. Both the training and explain data sets.

```
# Generate x_train from multivariate Gaussian
x_train <- MASS::mvrnorm(n = n_train, mu = mean_vector, Sigma = cov_matrix)

# Generate x_explain from multivariate Gaussian with
# the same mean and covariance matrix as x_train
x_explain <- MASS::mvrnorm(n = n_explain, mu = mean_vector, Sigma = cov_matrix)

# Set column names
colnames(x_train) <- colnames(x_explain) <- paste0("X", seq(n_features))

# Display x_train and x_explain
pairs(x_train)
```




```
pairs(x_explain)
```



Time to create the response and the model we want to explain. We let the response be a linear combination of the features and we let the model we a simple linear regression model.

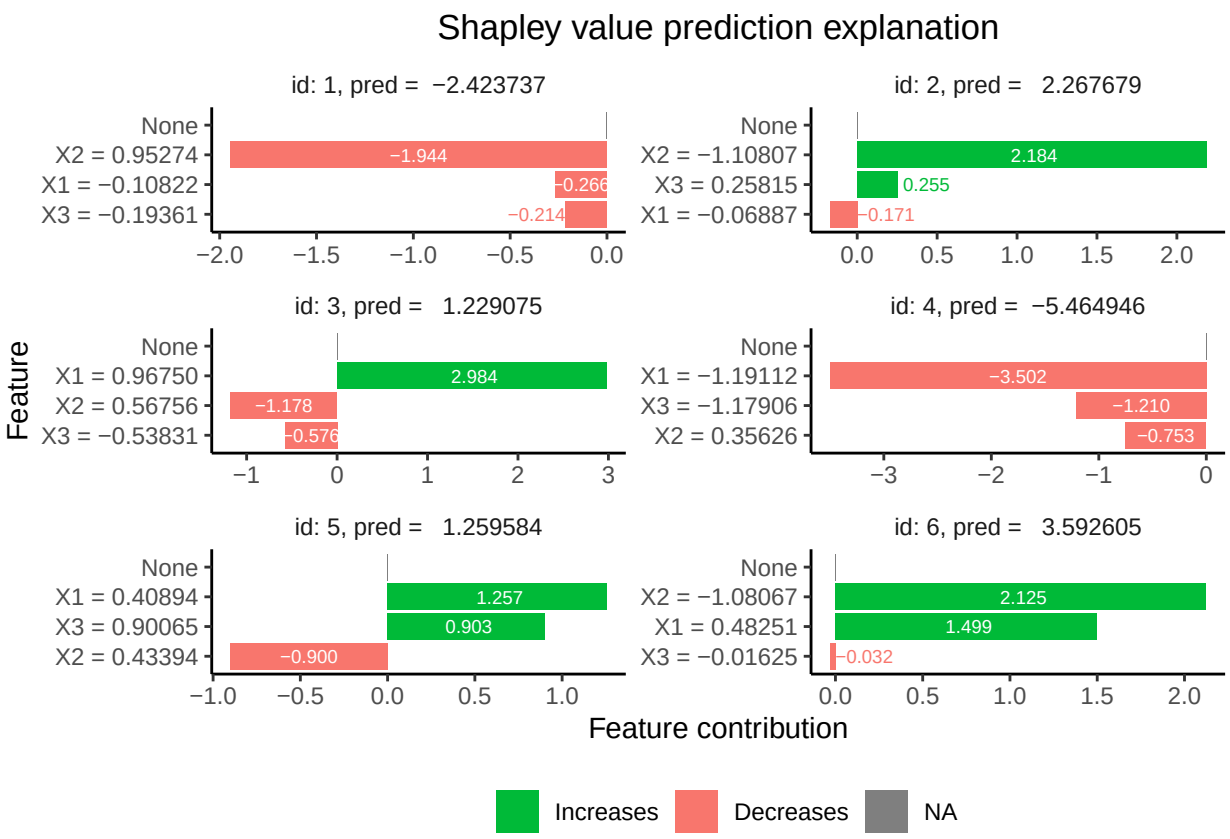
```
# Create a very simple linear response. To keep it simple, we do not add any noise now.
beta <- c(3, -2, 1)
y_train <- x_train %*% beta
```

```r
y_explain <- x_explain %*% beta

# Make data tables out of the training and test features + response
data_train <- as.data.table(cbind(y_train, x_train))
data_explain <- as.data.table(cbind(y_explain, x_explain))
colnames(data_train) <- colnames(data_explain) <- c("y", paste0("X", seq(n_features)))

# Train a linear model
model <- lm(y ~ ., data = data_train)
```

Conditional Shapley values

We compute the conditional Shapley values using the `shapr` package where we use the `gaussian` approach that assumes that the data follows a multivariate Gaussian distribution.

```r
# # Set up progressr, so we get feedback from shapr
# progressr::handlers(global = TRUE)
# progressr::handlers('cli') # requires the 'cli' package

# Explain the model using conditional shapley values
explanation <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  n_batches = 5,
  prediction_zero = 0, # Set \phi_0 to be 0 as we only want $M$ dimensional Shapley values
  n_samples = 2500 # The number of Monte Carlo samples to use
)

plot(explanation, index_x_explain = 1:6)
```



Principal components

We now start to look at the PCA + marginal Shapley value idea.

We start by computing the PC for the training data. We see that the PCs are independent.

```
# Compute principal components for x_train
pca_train <- prcomp(x_train, center = TRUE, scale. = TRUE)
x_train_pca <- pca_train$x

# Look at some summary to see the importance of each principal component
summary(pca_train)
```

```
## Importance of components:
##                           PC1    PC2   PC3
## Standard deviation     1.0101 1.0008 0.989
## Proportion of Variance 0.3401 0.3339 0.326
## Cumulative Proportion  0.3401 0.6740 1.000
```

```
# Compute PC means and covariance
colMeans(x_train_pca) # Technically zero
```

```
##           PC1           PC2           PC3
##  2.199976e-18 -1.898481e-17 -2.763761e-17
```

```
cov(x_train_pca)
```

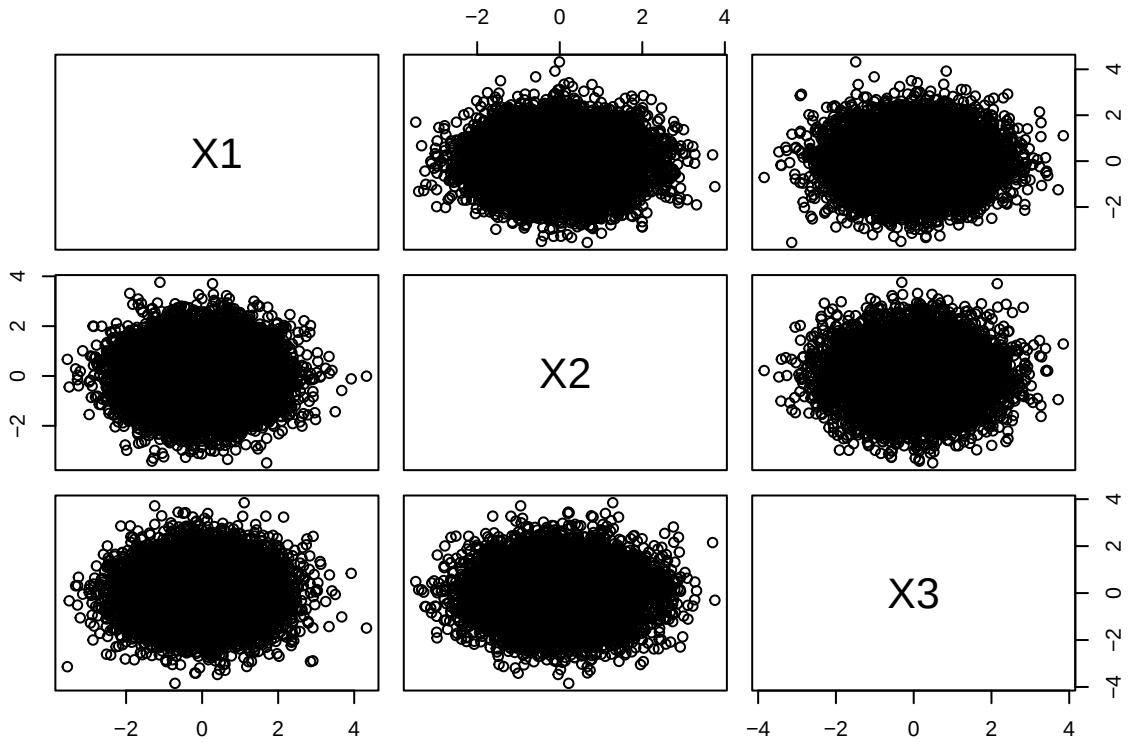
```
##               PC1          PC2           PC3
## PC1  1.020232e+00 4.697675e-16 -1.793835e-15
## PC2  4.697675e-16 1.001668e+00  5.535650e-16
```

```
## PC3 -1.793835e-15 5.535650e-16  9.780998e-01
```

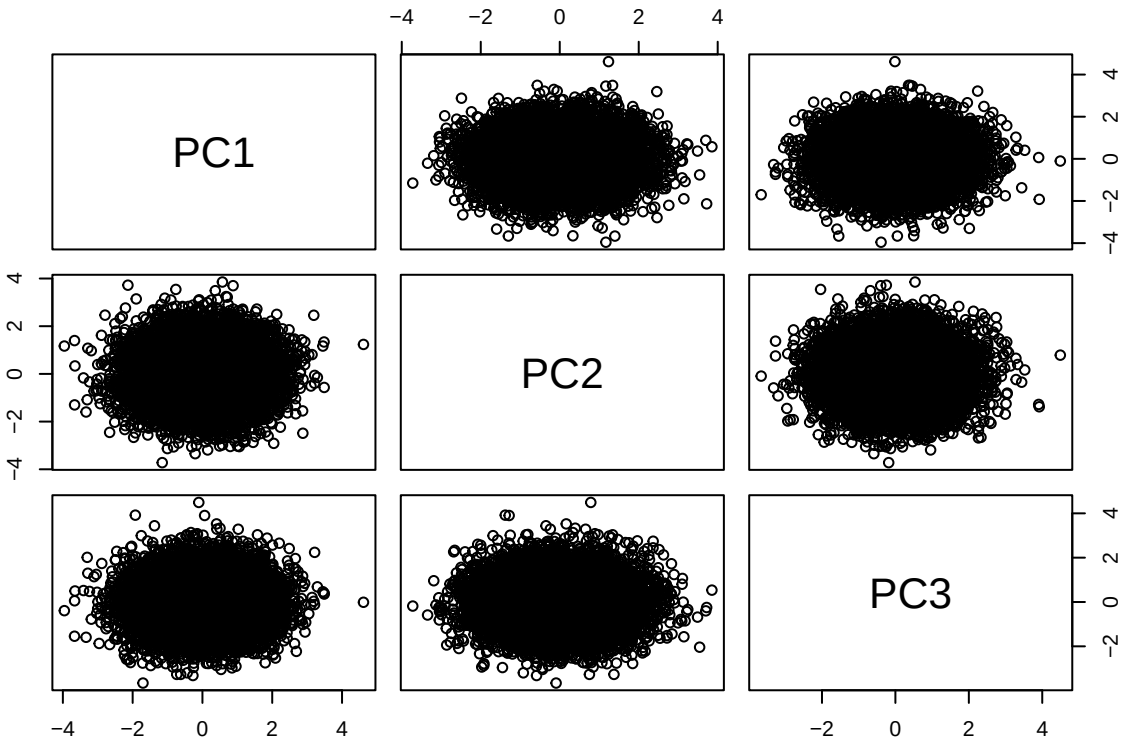
```
cor(x_train_pca)
```

```
##                PC1           PC2           PC3
## PC1  1.000000e+00 4.646988e-16 -1.795731e-15
## PC2  4.646988e-16 1.000000e+00  5.592618e-16
## PC3 -1.795731e-15 5.592618e-16  1.000000e+00
```

```
# Can also look at the pairs plot
pairs(x_train)
```




```
pairs(x_train_pca)
```



Then it is time to apply the same transformation on the explicands.

```
# Use the same transformations on x_explain
x_explain_pca <- predict(pca_train, newdata = x_explain)
# Now, x_explain_pca contains the principal components of x_explain
# based on the transformations learned from x_train

# x_explain_tilde = t(t(x_explain_pca %*% t(pca_rotation)) * pca_scale + pca_center)
# x_explain[1:10,]
# x_explain_tilde[1:10,]

# Compute PC means and covariance
# The means should be close to zero, but not exactly as we
# use the training data transformations
colMeans(x_explain_pca)
```

```
##           PC1           PC2           PC3
## -0.004368588  0.012913568  0.007046506
```

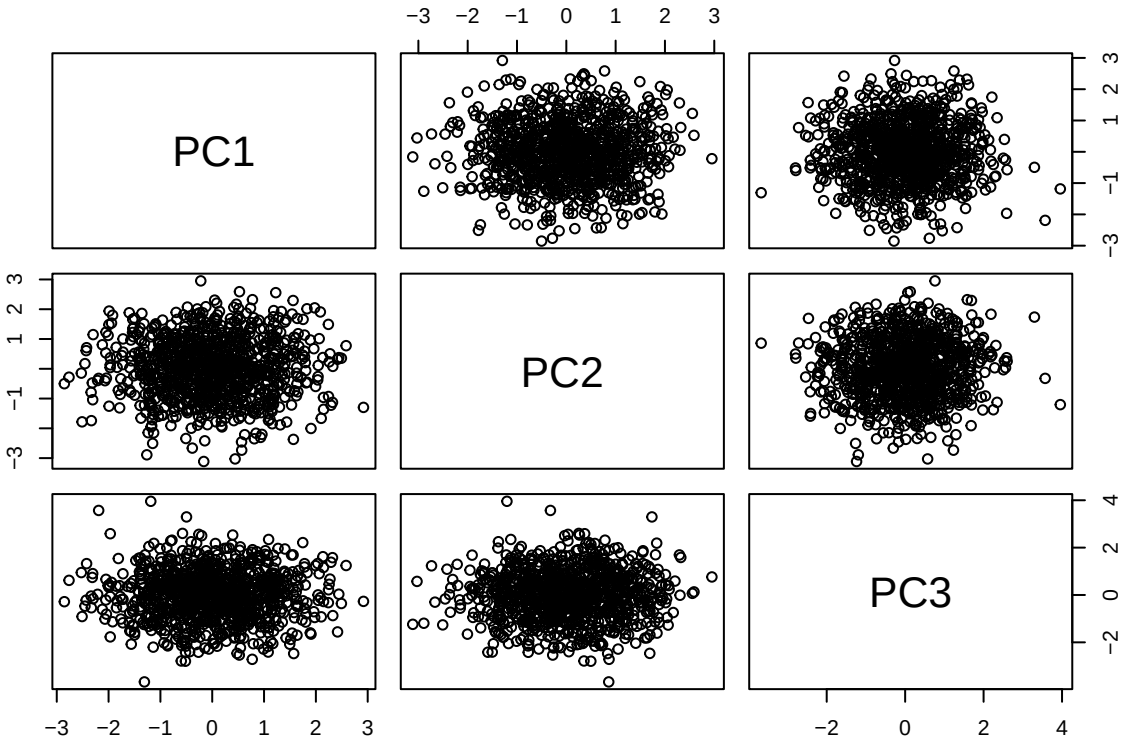
```
cov(x_explain_pca)
```

```
##              PC1         PC2         PC3
## PC1 0.910646139 0.006354762 0.006665884
## PC2 0.006354762 0.959548366 0.016410718
## PC3 0.006665884 0.016410718 1.049811770
```

```
cor(x_explain_pca)
```

```
##              PC1         PC2         PC3
## PC1 1.000000000 0.006798154 0.006817532
## PC2 0.006798154 1.000000000 0.016350778
## PC3 0.006817532 0.016350778 1.000000000
```

```r
# Can also look at the pairs plot
pairs(x_explain_pca)
```



To go from feature space to principal component space I use the `predict` function as above. This constitutes the $h : \boldsymbol{x} \mapsto \boldsymbol{z}$ function discussed above.

To go from PC space back to feature space, i.e., $h^{-1} : \boldsymbol{z} \mapsto \boldsymbol{x}$, I use the following computations:

```r
# If we are to center or scale the data.
# REMEMBER TO USE THE SAME AS ABOVE TRUE AND TRUE
center <- TRUE
scale. <- TRUE

# Apply the h(x) function on x to get to z
pca_train <- prcomp(x_train, center = center, scale. = scale.)

# Extract the rotation matrix and if we centered or not
pca_rotation <- pca_train$rotation
pca_center <- if (any(pca_train$center == FALSE)) rep(0, ncol(pca_train$x)) else pca_train$center
pca_scale <- if (any(pca_train$scale == FALSE)) rep(1, ncol(pca_train$x)) else pca_train$scale

# Apply the h^{-1}(z) function on z to get to x
h_inverse_of_z <- t(t(pca_train$x %*% t(pca_rotation)) * pca_scale + pca_center)

# This yields equal results for all combinations of `center` and `scale.`
all.equal(x_train, h_inverse_of_z)
```

```
## [1] TRUE
```

We can then compute the marginal Shapley values using `shapr` by specifying that we want to use the `independence` approach. The input to the `shapr` function is now the model $f(\boldsymbol{x})$, the principal component representation of training data, and the principal component representation of explicands. Furthermore, we

provide the `pca_rotation`, `pca_center` and `pca_scale`.

In this github branch, `shapr` will recognize that the latter three parameters are provided and will then transform the Monte Carlo samples of the principal components before sending them to the model f using the inverse transformation showed above in the previous code block.

First we choose a low value of `n_samples` as I want to make some pair plots to illustrate that we transform the Monte Carlo principal components. We can chose which combination/coalition we want to look at.

```
explanation$inter$objects$S
```

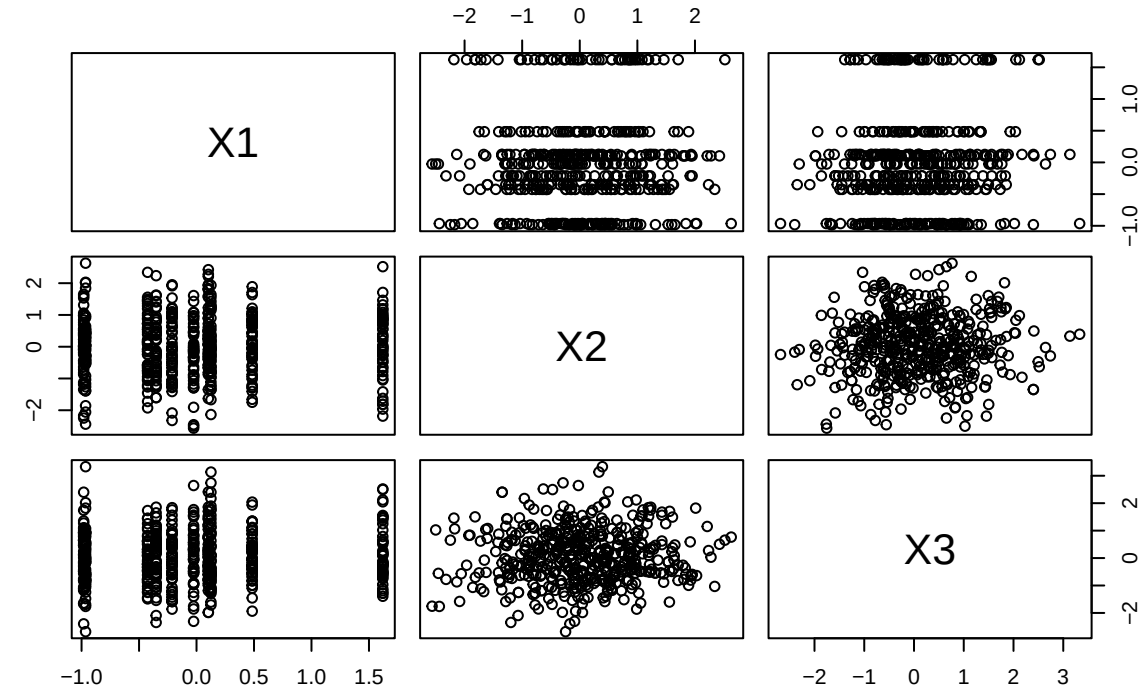
```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    1    0    0
## [3,]    0    1    0
## [4,]    0    0    1
## [5,]    1    1    0
## [6,]    1    0    1
## [7,]    0    1    1
## [8,]    1    1    1
```

That is, `id_combination = 2` means that we just condition on the first feature. Meaning that each row indicates the features that are included in the coalition $\mathcal{S}$. So in `id_combination = 5`, only the third feature is unknown.

For `id_combination = 2`, we see `plot_ggpairs_n_explicands = 10` "lines" in X1, while the other two features are random.

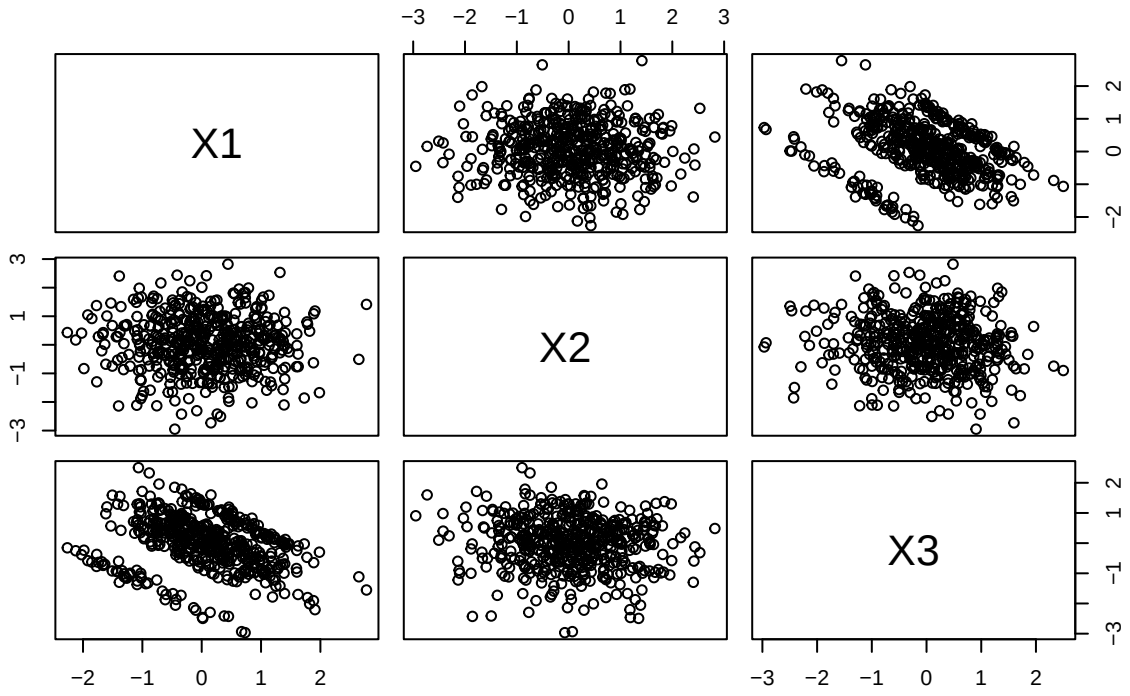
```
# HACK NOW. SHAPR needs column names to match, so we rename PC1 -> X1, PC2 -> X2 and so on.
# NEED TO FIX THIS IN A FINAL VERSION IF PCA_SHAP WORKS.
colnames(x_train_pca) <- colnames(x_explain_pca) <- colnames(x_train)

# Create the explanations
explanation_PCA <- explain(
  model = model,
  x_explain = x_explain_pca,
  x_train = x_train_pca,
  approach = "independence", # As we are doing marginal Shapley values
  prediction_zero = 0,
  n_batches = 1, # One batch as we want to plot the data
  n_samples = 50,
  pca_rotation = pca_rotation,
  pca_scale = pca_scale,
  pca_center = pca_center,
  plot_ggpairs = TRUE,
  plot_ggpairs_n_explicands = 10,
  plot_ggpairs_id_combination = c(2),
  keep_samp_for_vS = TRUE
)
```

Principal component feature space (10 explicands, id_comb = [2])




```
## Using the PCA-SHAP idea...
```

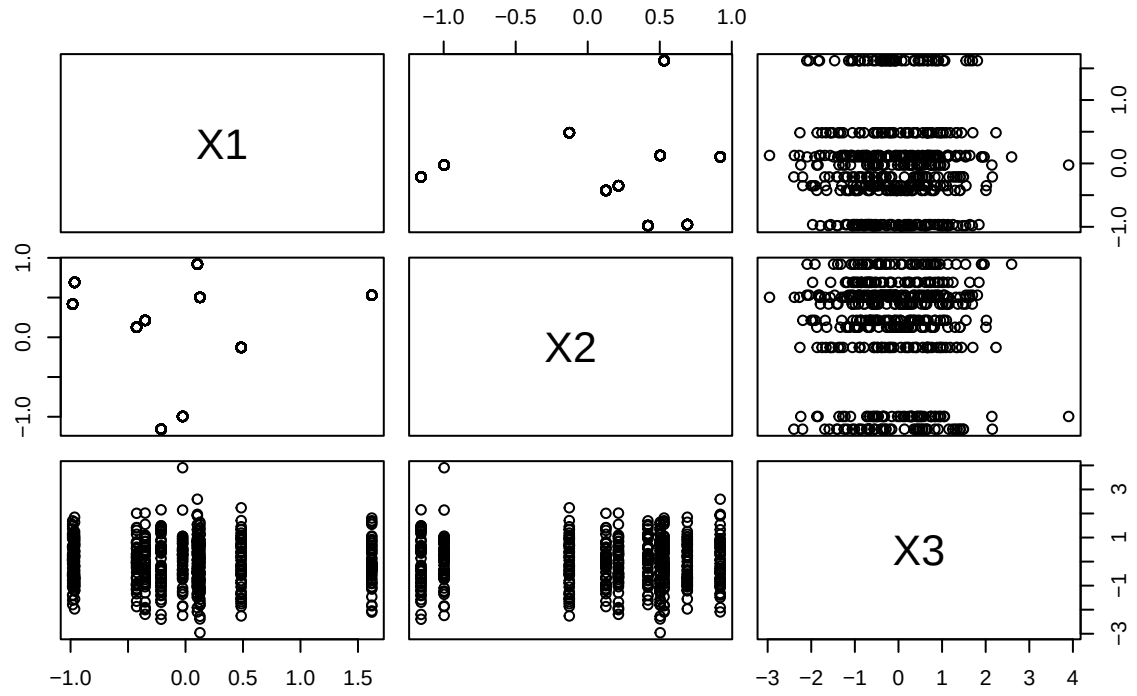
Original feature space (10 explicands, id_comb = [2])



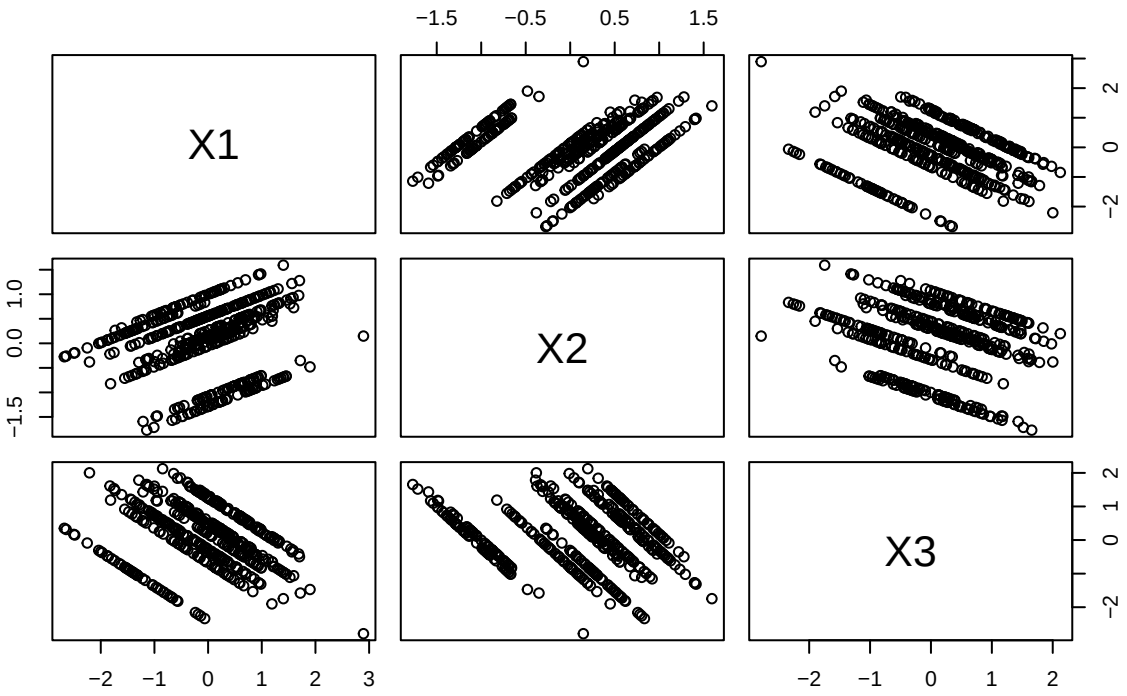
For `id_combination = 5`, we see `plot_ggpairs_n_explicands = 5` "lines" in X1 and X2, while the last feature is random.

```
explanation_PCA <- explain(
  model = model,
  x_explain = x_explain_pca,
  x_train = x_train_pca,
  approach = "independence", # As we are doing marginal Shapley values
  prediction_zero = 0,
  n_batches = 1, # One batch as we want to plot the data
  n_samples = 50,
  pca_rotation = pca_rotation,
  pca_scale = pca_scale,
  pca_center = pca_center,
  plot_ggpairs = TRUE,
  plot_ggpairs_n_explicands = 10,
  plot_ggpairs_id_combination = c(5),
  keep_samp_for_vS = TRUE
)
```

Principal component feature space (10 explicands, id_comb = [5])



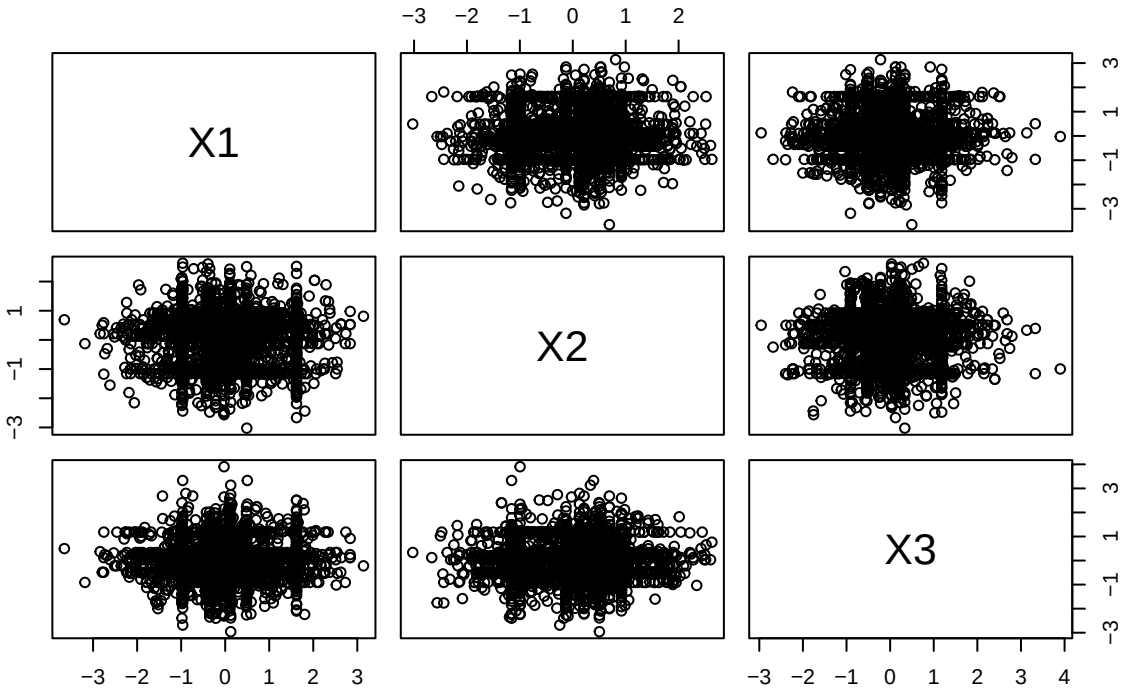

```
## Using the PCA-SHAP idea...
```



Can also look at all coalitions. Then it is easier to see that we go from something that is independent to having the same form as our original features.

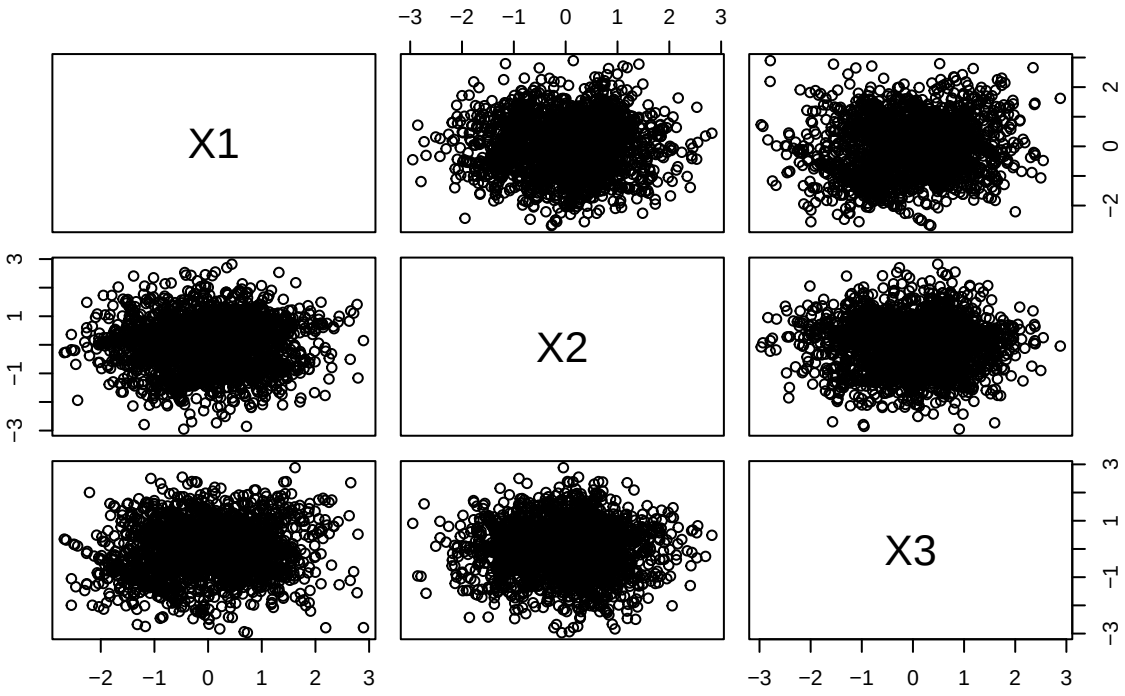
```
explanation_PCA <- explain(
  model = model,
  x_explain = x_explain_pca,
  x_train = x_train_pca,
  approach = "independence", # As we are doing marginal Shapley values
  prediction_zero = 0,
  n_batches = 1, # One batch as we want to plot the data
  n_samples = 50,
  pca_rotation = pca_rotation,
  pca_scale = pca_scale,
  pca_center = pca_center,
  plot_ggpairs = TRUE,
  plot_ggpairs_n_explicands = 10,
  plot_ggpairs_id_combination = NULL,
  keep_samp_for_vS = TRUE
)
```

pal component feature space (10 explicands, id_comb = [1, 2, 3, 4, 5, 6,




```
## Using the PCA-SHAP idea...
```

Original feature space (10 explicands, id_comb = [1, 2, 3, 4, 5, 6, 7, 8])



Then we skip the plotting and increase `n_samples`. `pairs` is computationally expensive to call when `n_samples` is high.

```
explanation_PCA <- explain(
  model = model,
  x_explain = x_explain_pca,
  x_train = x_train_pca,
  approach = "independence", # As we are doing marginal Shapley values
  prediction_zero = 0,
  n_batches = 1,
  n_samples = 2500,
  pca_rotation = pca_rotation,
  pca_scale = pca_scale,
  pca_center = pca_center,
  plot_ggpairs = FALSE,
  keep_samp_for_vS = TRUE
)
```

```
## Using the PCA-SHAP idea...
```

We can now look at the Shapley value in the two set ups and compare them.

```
# Phi_x
shapley_regular <- as.matrix(explanation$shapley_values[, -1])

# Phi_z
shapley_pca <- as.matrix(explanation_PCA$shapley_values[, -1])

# Phi_z transformed to \tilde{Phi}_x using the same transformation as above
shapley_pca_to_regular <- t(t(shapley_pca %*% t(pca_rotation)) * pca_scale + pca_center)

# Look at the RMSE between the Shapley values and the largest absolute difference
sqrt(mean((shapley_pca_to_regular - shapley_regular)^2))
```

```
## [1] 2.901534
```

```
max(abs(shapley_pca_to_regular - shapley_regular))
```

```
## [1] 14.39639
```

We see that we get quite a large difference.

```
# We see that the Shapley values do not sum to the right predicted value.
explanation$pred_explain[1:10]
```

```
##   p_hat1_1   p_hat1_2   p_hat1_3   p_hat1_4   p_hat1_5   p_hat1_6   p_hat1_7
## -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054  0.9090604
##   p_hat1_8   p_hat1_9  p_hat1_10
##  0.8516805 -1.6001365 -2.0102877
```

```
rowSums(shapley_regular)[1:10]
```

```
##  [1] -2.4237371  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054
##  [7]  0.9090604  0.8516805 -1.6001365 -2.0102877
```

```
rowSums(shapley_pca_to_regular)[1:10]
```

```
##  [1] -1.5129113  1.8999392 -1.7909369  5.2700290 -4.7139563  1.6055126
##  [7] -1.9141479 -5.2927939  1.8950523 -0.5932327
```

```
# Can scale it, but obviously still wrong signs and values
tmp <- sweep(shapley_pca_to_regular, 1, explanation$pred_explain / rowSums(shapley_pca_to_regular), "*")
```

```r
rowSums(tmp)[1:10]
```

```
##  [1] -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054
##  [7]  0.9090604  0.8516805 -1.6001365 -2.0102877
```

```r
explanation$pred_explain[1:10]
```

```
##   p_hat1_1   p_hat1_2   p_hat1_3   p_hat1_4   p_hat1_5   p_hat1_6   p_hat1_7
## -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054  0.9090604
##   p_hat1_8   p_hat1_9  p_hat1_10
##  0.8516805 -1.6001365 -2.0102877
```

```r
shapley_regular[1, ]
```

```
##         X1         X2         X3
## -0.2662762 -1.9439438 -0.2135171
```

```r
tmp[1, ]
```

```
##         X1         X2         X3
##  1.6500774 -3.5798771 -0.4939375
```

```r
sqrt(mean((tmp - shapley_regular)^2))
```

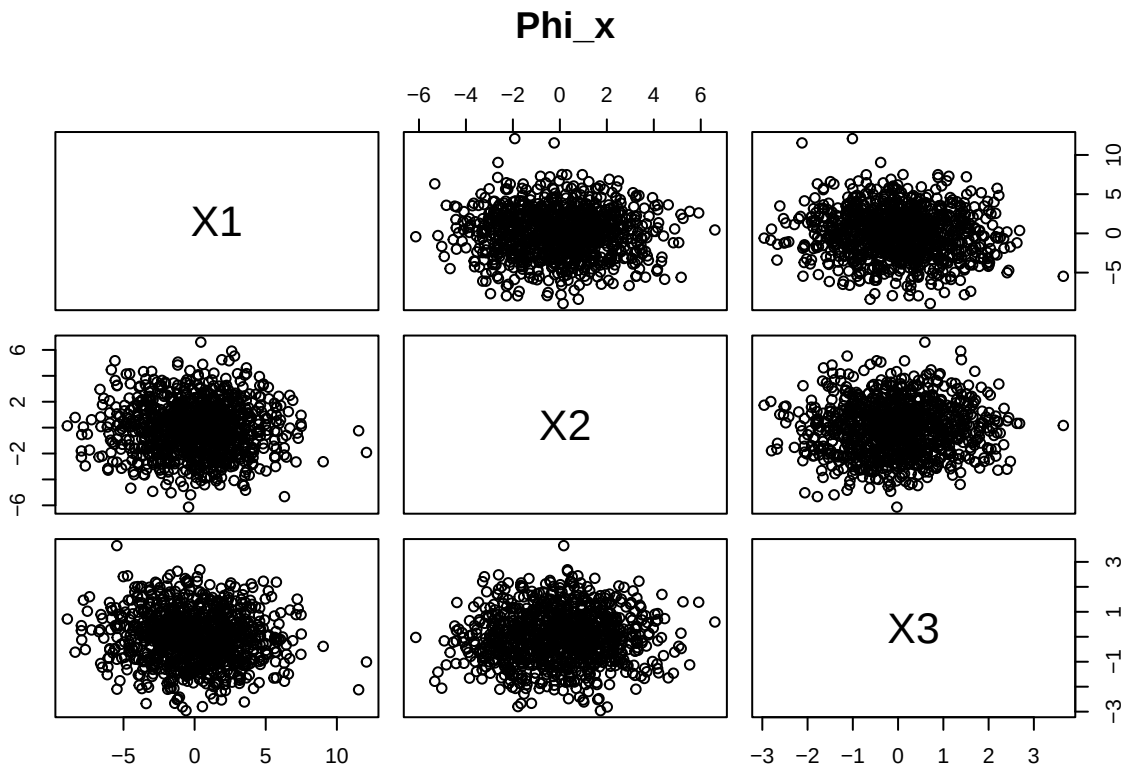
```
## [1] 106.4233
```

```r
max(abs(tmp - shapley_regular))
```

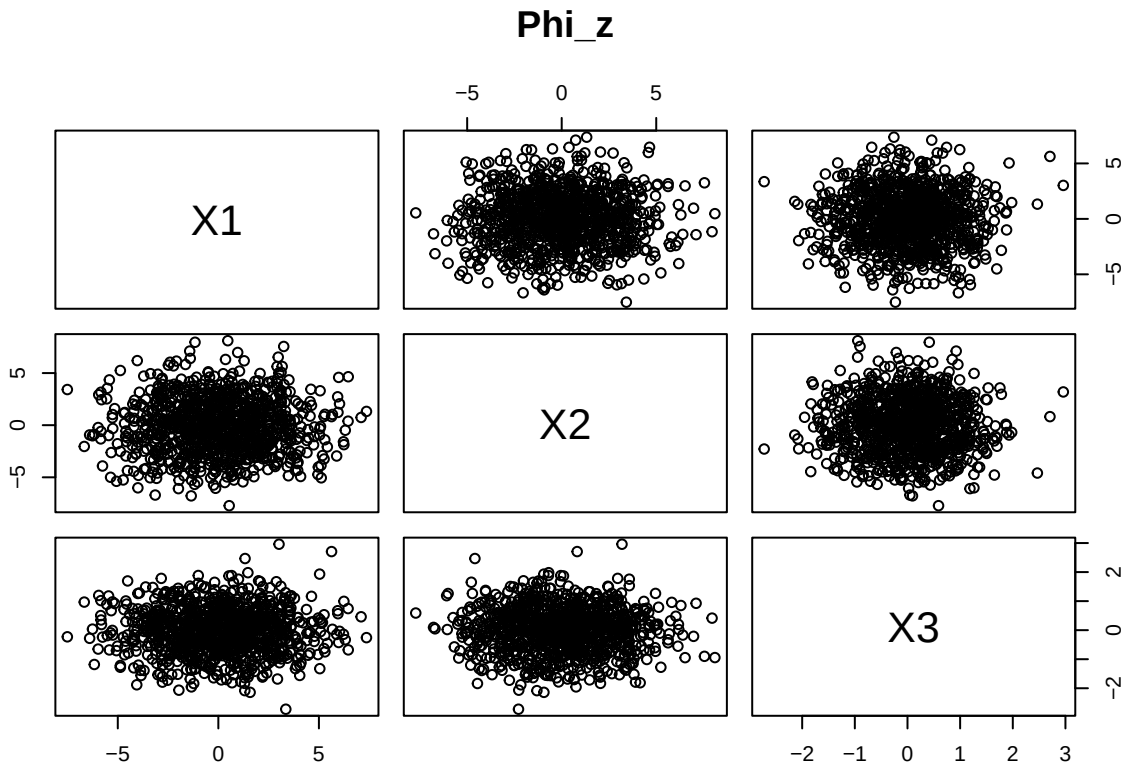
```
## [1] 4637.707
```

We can also look at some plots.

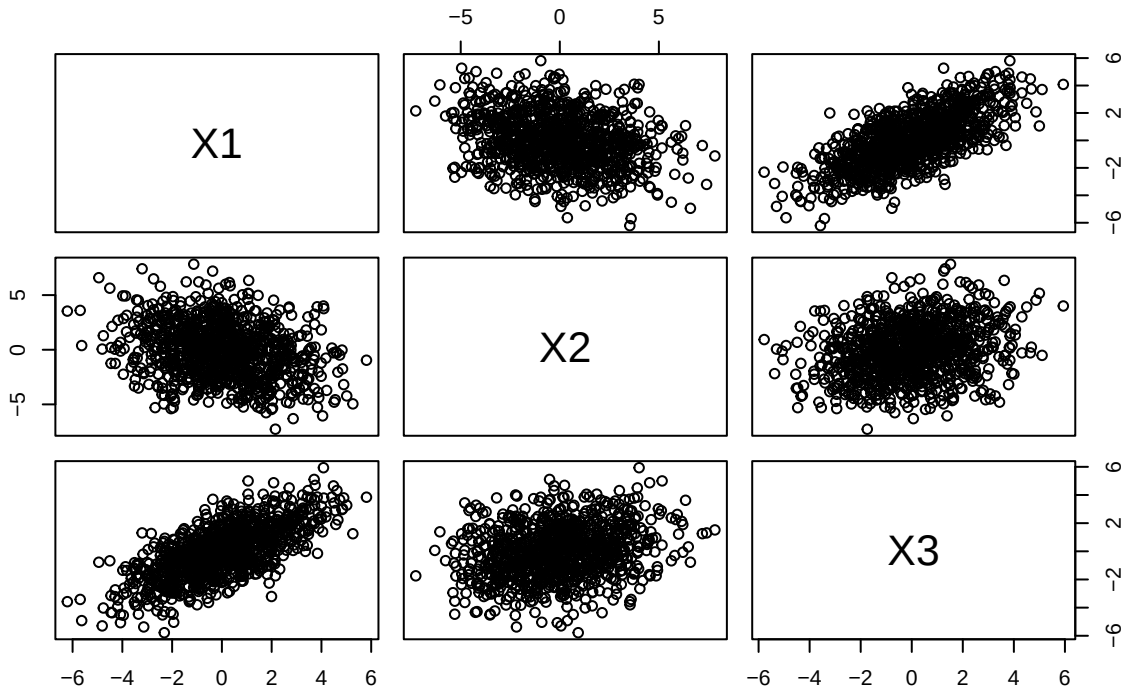
```r
pairs(shapley_regular, main = "Phi_x")
```



```r
pairs(shapley_pca, main = "Phi_z")
```




```r
pairs(shapley_pca_to_regular, main = "Phi_z_to_x")
```

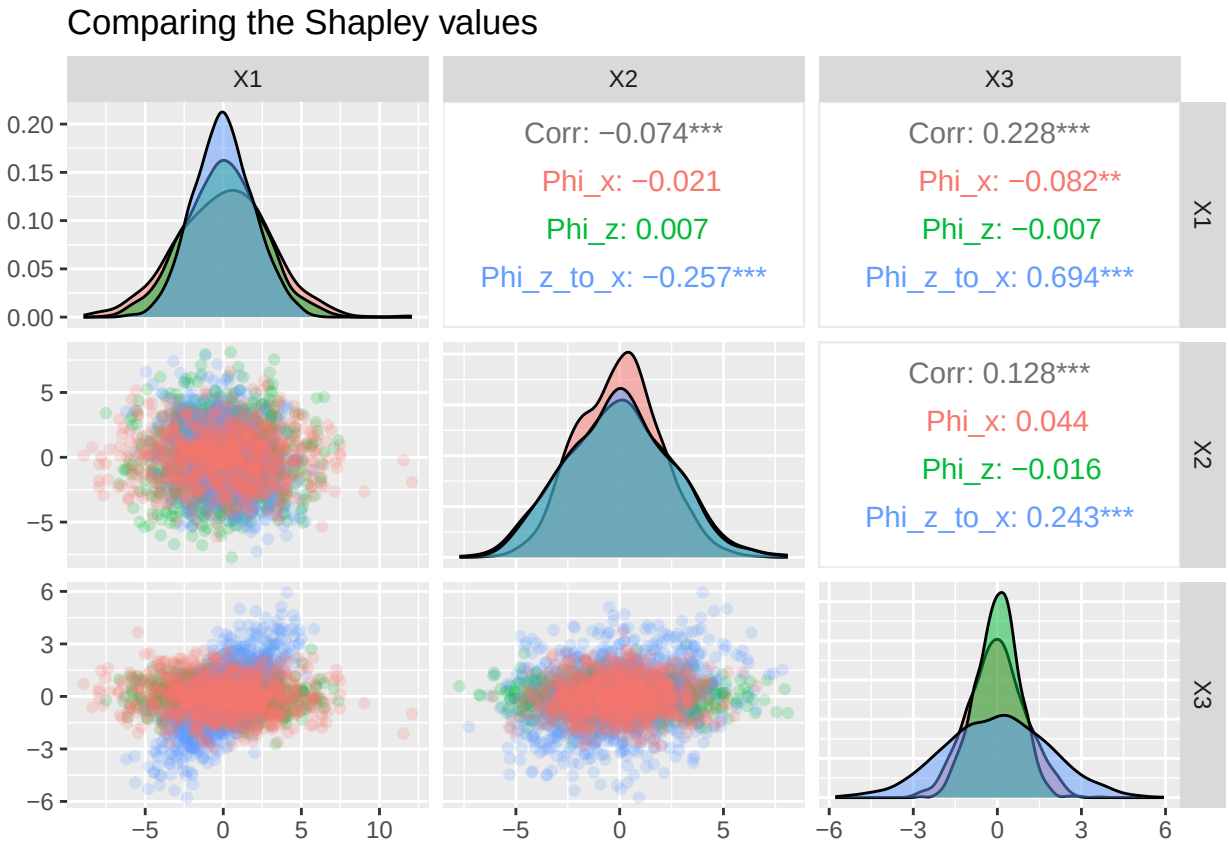


Or we can combine all of them.

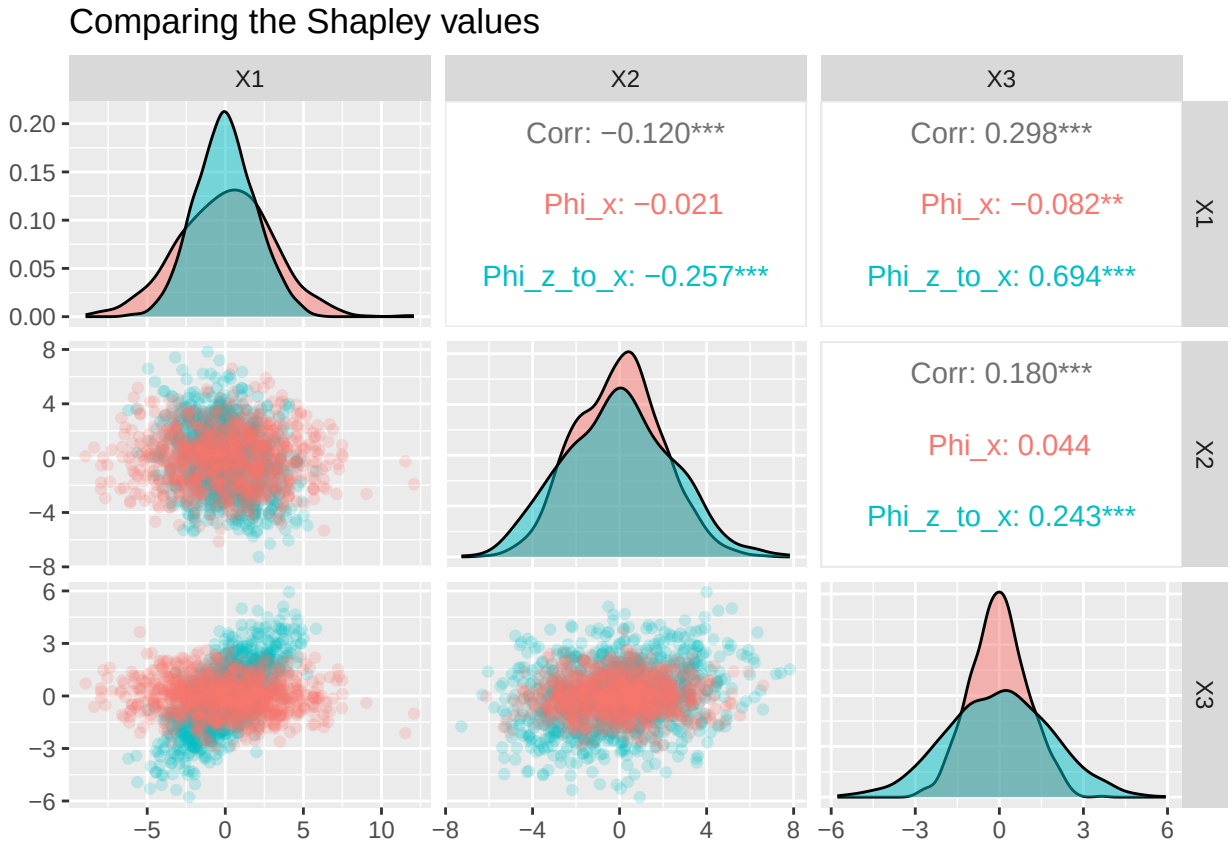
```
library(ggplot2)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
Shapley_value_results <- lapply(list(shapley_pca, shapley_pca_to_regular, shapley_regular), as.data.tabl
names(Shapley_value_results) <- c("Phi_z", "Phi_z_to_x", "Phi_x")
Shapley_value_results <- data.table::rbindlist(Shapley_value_results, idcol = "Shapley_type")
GGally::ggpairs(Shapley_value_results,
  columns = 2:4,
  title = "Comparing the Shapley values",
  ggplot2::aes(color = Shapley_type),
  diag = list(continuous = GGally::wrap("densityDiag", alpha = 0.5)),
  lower = list(continuous = GGally::wrap("points", alpha = 0.2))
)
```

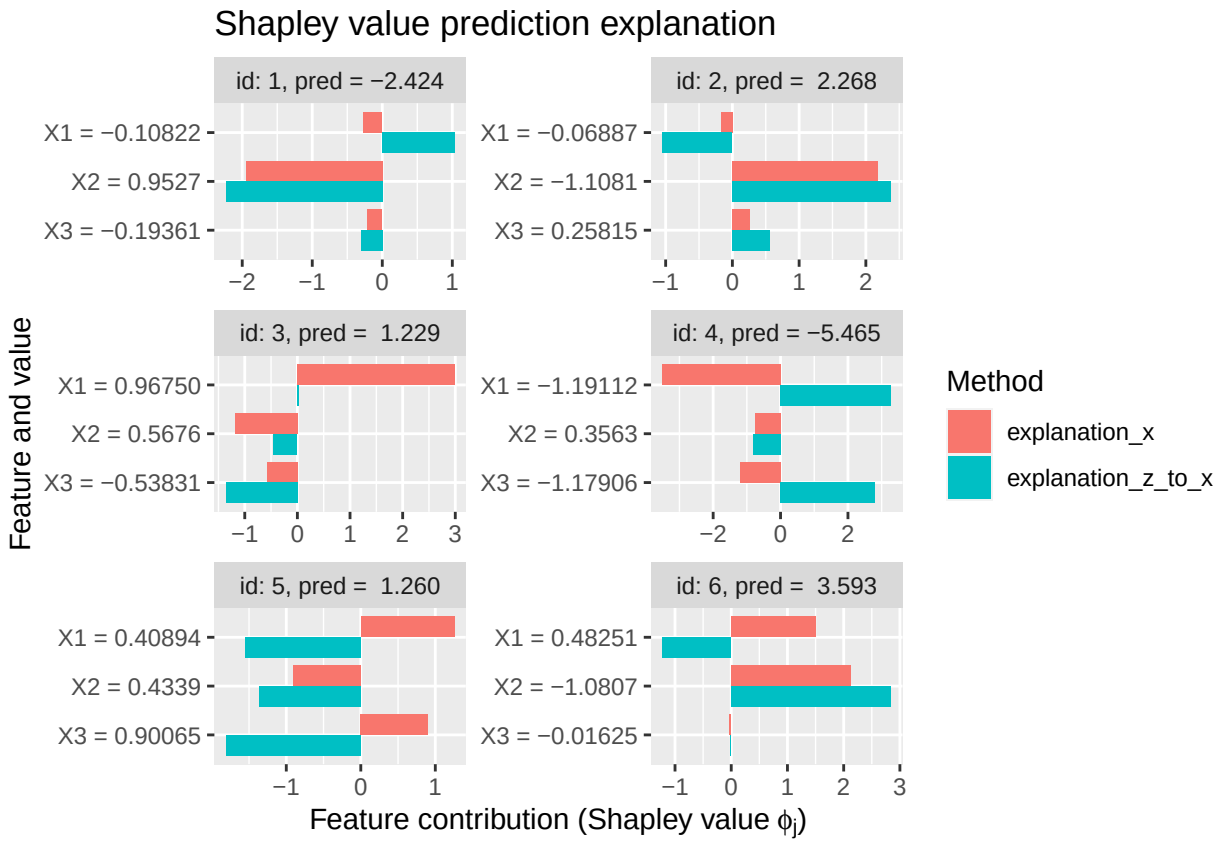



```
# And plot only with original and the new idea
GGally::ggpairs(Shapley_value_results[Shapley_type %in% c("Phi_x", "Phi_z_to_x")],
  columns = 2:4,
  title = "Comparing the Shapley values",
  ggplot2::aes(color = Shapley_type),
  diag = list(continuous = GGally::wrap("densityDiag", alpha = 0.5)),
  lower = list(continuous = GGally::wrap("points", alpha = 0.2))
)
```



We can look at and compare the Shapley value explanations for the first `6` explicands. We get very different explanations. Note that the headers are not correct as only the Shapley values of `explanation_x` sum to the predicted value `pred`. This is NOT true for `explanation_z_to_x`.

```
explanation_z_to_x <- explanation
explanation_z_to_x$shapley_values <- cbind(explanation_z_to_x$shapley_values[, "none"], shapley_pca_to_
shapr::plot_SV_several_approaches(
  explanation_list = list(
    explanation_x = explanation,
    # explanation_z = explanation_PCA,
    explanation_z_to_x = explanation_z_to_x
  ),
  do_checks = FALSE, # need to set this to false as plot does not work otherwise when including explana
  index_explicands = c(1:6)
)
```



TRASH

Just me checking that things give the same results:

```
x_explain_tilde <- t(t(x_explain_pca %*% t(pca_rotation)) * pca_scale + pca_center)
t(t(as.matrix(explanation_PCA$internal$data$x_explain)[1:10, ] %*% t(pca_rotation)) * pca_scale + pca_c
```

```
##                X1         X2          X3
##  [1,] -0.10821823  0.9527380 -0.19360641
##  [2,] -0.06886765 -1.1080674  0.25814728
##  [3,]  0.96750381  0.5675618 -0.53831257
##  [4,] -1.19112419  0.3562553 -1.17906281
##  [5,]  0.40894066  0.4339425  0.90064742
##  [6,]  0.48250741 -1.0806664 -0.01624955
##  [7,]  0.38661675  0.2147102  0.17863049
##  [8,]  0.63338171  0.8251868  0.60190902
##  [9,] -0.91465685 -0.4413498  0.26113438
## [10,] -0.55842547  0.3195385  0.30406566
```

```
x_explain[1:10, ]
```

```
##                X1         X2          X3
##  [1,] -0.10821823  0.9527380 -0.19360641
##  [2,] -0.06886765 -1.1080674  0.25814728
##  [3,]  0.96750381  0.5675618 -0.53831257
##  [4,] -1.19112419  0.3562553 -1.17906281
##  [5,]  0.40894066  0.4339425  0.90064742
##  [6,]  0.48250741 -1.0806664 -0.01624955
```

```
##  [7,]  0.38661675  0.2147102  0.17863049
##  [8,]  0.63338171  0.8251868  0.60190902
##  [9,] -0.91465685 -0.4413498  0.26113438
## [10,] -0.55842547  0.3195385  0.30406566
```

```
x_explain_tilde[1:10, ]
```

```
##                X1         X2          X3
##  [1,] -0.10821823  0.9527380 -0.19360641
##  [2,] -0.06886765 -1.1080674  0.25814728
##  [3,]  0.96750381  0.5675618 -0.53831257
##  [4,] -1.19112419  0.3562553 -1.17906281
##  [5,]  0.40894066  0.4339425  0.90064742
##  [6,]  0.48250741 -1.0806664 -0.01624955
##  [7,]  0.38661675  0.2147102  0.17863049
##  [8,]  0.63338171  0.8251868  0.60190902
##  [9,] -0.91465685 -0.4413498  0.26113438
## [10,] -0.55842547  0.3195385  0.30406566
```

```
x_explain_pca[1:10, ]
```

```
##                X1         X2          X3
##  [1,]  0.10292153  0.9196523  0.33583407
##  [2,] -0.02528671 -0.9965518 -0.53108448
##  [3,] -0.35128296  0.2132127  1.18306560
##  [4,]  1.62049410  0.5300071  0.12319325
##  [5,] -0.98162584  0.4178644 -0.21927307
##  [6,] -0.21239476 -1.1567501  0.03399111
##  [7,] -0.42555686  0.1264279  0.20143500
##  [8,] -0.96488323  0.6912839  0.24695744
##  [9,]  0.48448596 -0.1278991 -0.90977752
## [10,]  0.12570219  0.5029375 -0.48658453
```

```
as.matrix(explanation_PCA$internal$data$x_explain)[1:10, ]
```

```
##                X1         X2          X3
##  [1,]  0.10292153  0.9196523  0.33583407
##  [2,] -0.02528671 -0.9965518 -0.53108448
##  [3,] -0.35128296  0.2132127  1.18306560
##  [4,]  1.62049410  0.5300071  0.12319325
##  [5,] -0.98162584  0.4178644 -0.21927307
##  [6,] -0.21239476 -1.1567501  0.03399111
##  [7,] -0.42555686  0.1264279  0.20143500
##  [8,] -0.96488323  0.6912839  0.24695744
##  [9,]  0.48448596 -0.1278991 -0.90977752
## [10,]  0.12570219  0.5029375 -0.48658453
```

```
predict(model, as.data.table(x_explain[1:10, ]))
```

```
##          1          2          3          4          5          6          7
## -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054  0.9090604
##          8          9         10
##  0.8516805 -1.6001365 -2.0102877
```

```
predict(model, explanation$internal$data$x_explain[1:10, ])
```

```
##          1          2          3          4          5          6          7
## -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054  0.9090604
```

```
##          8          9         10
##  0.8516805 -1.6001365 -2.0102877
```

```
predict(model, as.data.table(t(t(x_explain_pca %*% t(pca_rotation)) * pca_scale + pca_center))[1:10, ])
```

```
##          1          2          3          4          5          6          7
## -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054  0.9090604
##          8          9         10
##  0.8516805 -1.6001365 -2.0102877
```

```
predict(model, as.data.table(t(t(as.matrix(explanation_PCA$internal$data$x_explain) %*% t(pca_rotation)) *
```

```
##          1          2          3          4          5          6          7
## -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054  0.9090604
##          8          9         10
##  0.8516805 -1.6001365 -2.0102877
```

```
explanation_PCA$pred_explain[1:10]
```

```
##   p_hat1_1   p_hat1_2   p_hat1_3   p_hat1_4   p_hat1_5   p_hat1_6   p_hat1_7
## -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054  0.9090604
##   p_hat1_8   p_hat1_9  p_hat1_10
##  0.8516805 -1.6001365 -2.0102877
```

```
explanation$pred_explain[1:10]
```

```
##   p_hat1_1   p_hat1_2   p_hat1_3   p_hat1_4   p_hat1_5   p_hat1_6   p_hat1_7
## -2.4237372  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054  0.9090604
##   p_hat1_8   p_hat1_9  p_hat1_10
##  0.8516805 -1.6001365 -2.0102877
```

```
max(abs(explanation_PCA$pred_explain - explanation$pred_explain))
```

```
## [1] 5.329071e-15
```

```
x_train_tilde <- t(t(x_train_pca %*% t(pca_rotation)) * pca_scale + pca_center)
x_train[1:10, ]
```

```
##                X1         X2          X3
##  [1,] -0.83629674  2.3707252 -0.56047565
##  [2,] -0.22057299 -0.1668120 -0.23017749
##  [3,] -2.10351477  0.9269614  1.55870831
##  [4,] -1.66780754 -0.5681517  0.07050839
##  [5,] -1.09796286  0.2250901  0.12928774
##  [6,] -1.66562121  1.1319859  1.71506499
##  [7,] -0.04950063  1.3804815  0.46091621
##  [8,]  1.55930293 -0.2328034 -1.26506123
##  [9,] -0.40462394 -1.6006433 -0.68685285
## [10,]  0.78602610 -0.2983692 -0.44566197
```

```
x_train_tilde[1:10, ]
```

```
##                X1         X2          X3
##  [1,] -0.83629674  2.3707252 -0.56047565
##  [2,] -0.22057299 -0.1668120 -0.23017749
##  [3,] -2.10351477  0.9269614  1.55870831
##  [4,] -1.66780754 -0.5681517  0.07050839
##  [5,] -1.09796286  0.2250901  0.12928774
##  [6,] -1.66562121  1.1319859  1.71506499
```

```
##  [7,] -0.04950063  1.3804815  0.46091621
##  [8,]  1.55930293 -0.2328034 -1.26506123
##  [9,] -0.40462394 -1.6006433 -0.68685285
## [10,]  0.78602610 -0.2983692 -0.44566197
```

```
rowSums(shapley_pca)[1:10]
```

```
##  [1] -2.4237371  2.2676792  1.2290752 -5.4649460  1.2595843  3.5926054
##  [7]  0.9090604  0.8516805 -1.6001365 -2.0102877
```

```
rowSums(shapley_pca_to_regular)[1:10]
```

```
##  [1] -1.5129113  1.8999392 -1.7909369  5.2700290 -4.7139563  1.6055126
##  [7] -1.9141479 -5.2927939  1.8950523 -0.5932327
```