Interpretability Examples

First we load in the crabs data set. This contains physical measurements of several species of crabs collected at Fremantle, West Australia.

```
library(MASS)
library(data.table)
library(ggplot2)
library(interpret)
library(Rforestry)
```

```
## ##
## ##  Rforestry (Version 0.9.0.70, Build Date: R 4.1.1; x86_64-apple-darwin17.0; 2022-01-20 00:01:36 U
## ##  See https://github.com/forestry-labs for additional documentation.
## ##  Please cite software as:
## ##    Soren R. Kunzel, Theo F. Saarinen, Edward W. Liu, Jasjeet S. Sekhon. 2019.
## ##    ''Linear Aggregation in Tree-based Estimators.'' arXiv preprint
## ##    arXiv:1906.06463. https://arxiv.org/abs/1906.06463
## ##
```

```
# source("predictor.R")
# source("interpret.R")
source("~/Dropbox/interpretability_sandbox/R/plotter.R")
set.seed(491)

data <- MASS::crabs
levels(data$sex) <- list(Male = "M", Female = "F")
levels(data$sp) <- list(Orange = "O", Blue = "B")
colnames(data) <- c("Species","Sex","Index","Frontal Lobe",
                    "Rear Width", "Carapace Length","Carapace Width","Body Depth")
```

We can train a random forest to estimate the Carapace Width of the crabs based on the other features. In order to use the interpretability features, we must create a `Predictor` class for the estimator we want to interpret. This class standardizes the predictions, tracks the outcome feature, and stores the training data.

```
# try with different dataset
set.seed(491)
test_ind <- sample(1:nrow(data), nrow(data)%/%5)
train_reg <- data[-test_ind,]
test_reg <- data[test_ind,]

# Train a random forest on the data set
forest <- forestry(x=train_reg[,-which(names(train_reg)=="Carapace Width")],
                   y=train_reg[,which(names(train_reg)=="Carapace Width")])

# Create a predictor wrapper for the forest
```

```
# this allows us to use a standard wrapper for querying any
# trained estimator
forest_predictor <- Predictor$new(model = forest,
                                  data=train_reg,
                                  y="Carapace Width",
                                  task = "regression")
```

Once we have initialized a `Predictor` object for the forest, we can pass this to the `Interpretor` class. In the future, this will have several methods implemented as different options, but now it defaults to creating PDP functions + plots for the estimator. Examining the `Interpretor`, we can see the current method selected: "pdp", the feature names, the training data indices, and the lists of pdp functions.

```
# Now given a predictor object, we can create an interpretor class
forest_interpret <- Interpreter$new(predictor = forest_predictor)

print(forest_interpret)
```

```
## <Interpreter>
##   Public:
##     clone: function (deep = FALSE)
##     data.points: 17 59 105 8 18 51 157 37 102 44 119 131 107 75 7 148 60  ...
##     features: Species Sex Index Frontal Lobe Rear Width Carapace Lengt ...
##     functions.1d: list
##     functions.2d: list
##     initialize: function (predictor = NULL, samples = NULL, data.points = NULL,
##     method: pdp
##     predictor: Predictor, R6
```

```
# Check to see that the specified functions are behaving
#print(forestinterpret$functions.1d$`Body Depth`(1))
```

The pdp functions are stored in two lists, one for 1-d pdp functions and one for 2-d pdp functions. For any feature, we can retrieve the pdp function by selecting the entry in the list with that feature name.

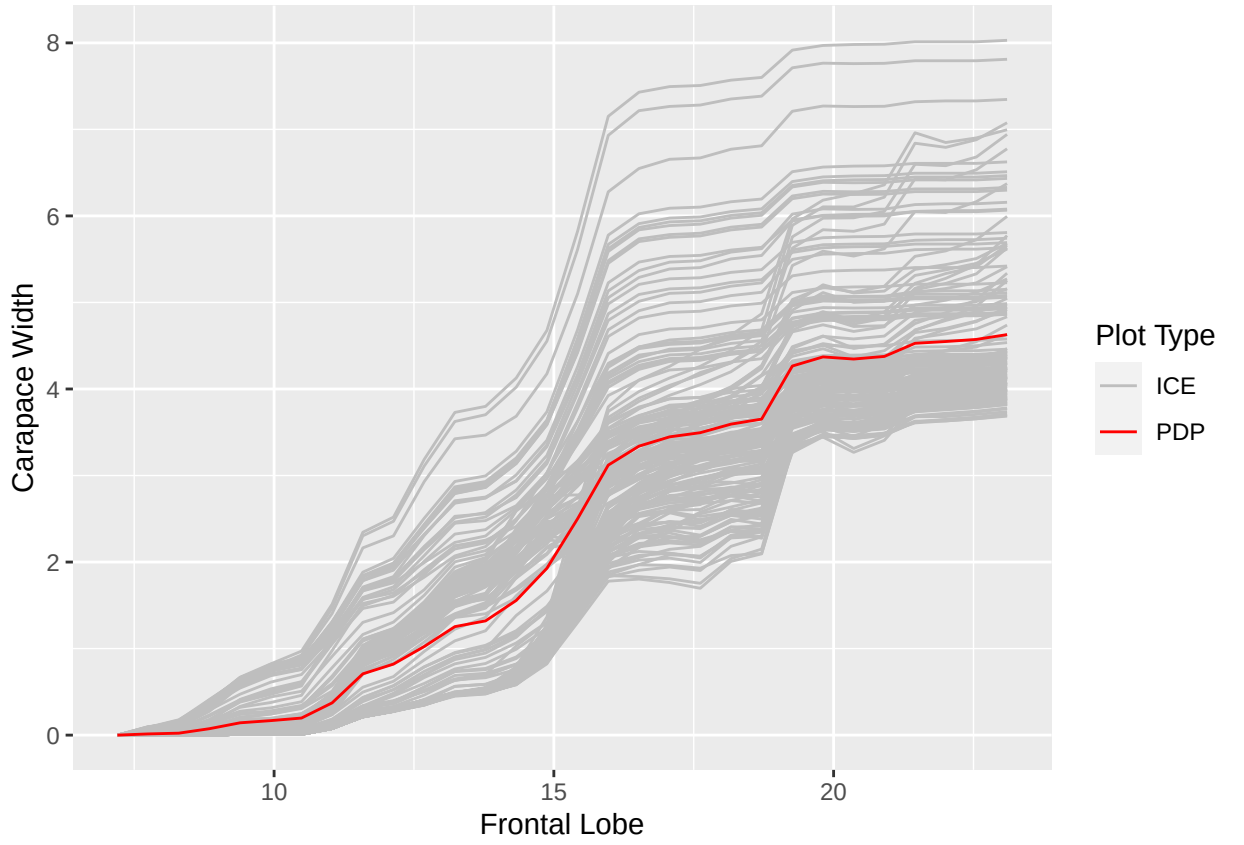
```
# We can also access the Partial dependence functions themselves

one_feat <- train_reg$`Frontal Lobe`
preds_pdp <- forest_interpret$functions.1d$`Frontal Lobe`(one_feat)
print(preds_pdp)
```

```
##   [1] 34.30249 34.41734 34.44234 34.44921 34.67518 35.00046 35.02284 35.11743
##   [9] 35.47968 35.47968 35.49736 35.53507 35.54499 35.62712 35.83010 36.02498
##  [17] 36.41389 36.41389 36.75502 36.78658 37.37735 37.45747 37.45747 37.47773
##  [25] 37.50593 37.66866 37.70930 37.71698 37.73481 37.73481 37.76440 37.81859
##  [33] 37.86082 37.87467 37.96698 38.58168 38.58168 38.65140 38.65630 38.65630
##  [41] 38.81511 34.28389 34.37910 34.37724 34.43134 34.44921 34.45287 34.46049
##  [49] 34.63168 34.67942 34.97021 35.00046 35.01691 35.04928 35.27399 35.47968
##  [57] 35.47968 35.52060 35.53507 35.53654 35.54889 35.62712 36.06094 36.22803
##  [65] 36.41389 36.49817 36.49817 36.75502 36.78056 36.78658 36.80397 36.81986
##  [73] 36.86280 37.33438 37.47773 37.56105 37.76435 37.77059 38.46212 34.37724
##  [81] 34.45578 34.49384 35.12596 35.30982 35.53654 35.54889 35.59126 35.62522
##  [89] 35.62751 35.62751 35.64753 35.64753 35.64753 36.02498 36.06094 36.49817
```

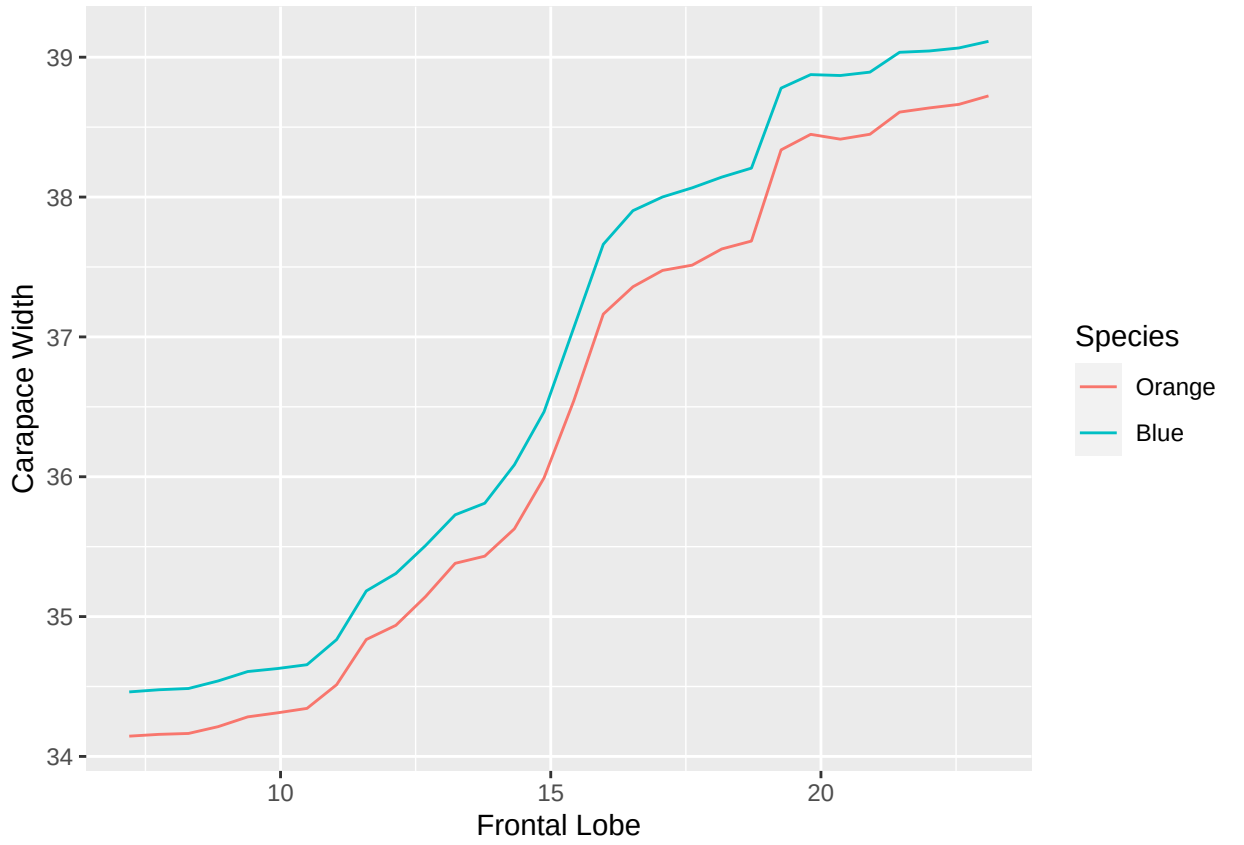
```
##  [97] 36.49817 36.78658 36.86280 37.47773 37.50593 37.73481 37.76435 37.77059
## [105] 37.77059 37.83315 37.86082 37.87467 37.87603 37.87604 37.96698 37.96698
## [113] 38.59398 38.62473 38.64342 38.71064 38.81142 38.82872 38.83958 38.90604
## [121] 38.91239 34.49384 34.74866 35.12596 35.27399 35.62522 35.83010 36.06094
## [129] 36.22803 36.41389 36.81986 36.81986 36.86280 37.47773 37.68615 37.77059
## [137] 37.77059 37.77548 37.87467 37.87467 37.87433 37.87703 37.87703 37.87604
## [145] 37.96698 38.00429 38.08885 38.08885 38.64094 38.62600 38.62473 38.62878
## [153] 38.63950 38.64342 38.66018 38.81511 38.80835 38.82872 38.85389 38.91239
```

In order to use these pdp functions to create plots, we can use the `Plotter` function. This allows us to make standard pdp plots as well as conditional pdp plots with two variables. Here we show a single variable pdp plot and two conditional pdp plots with second categorical variables.

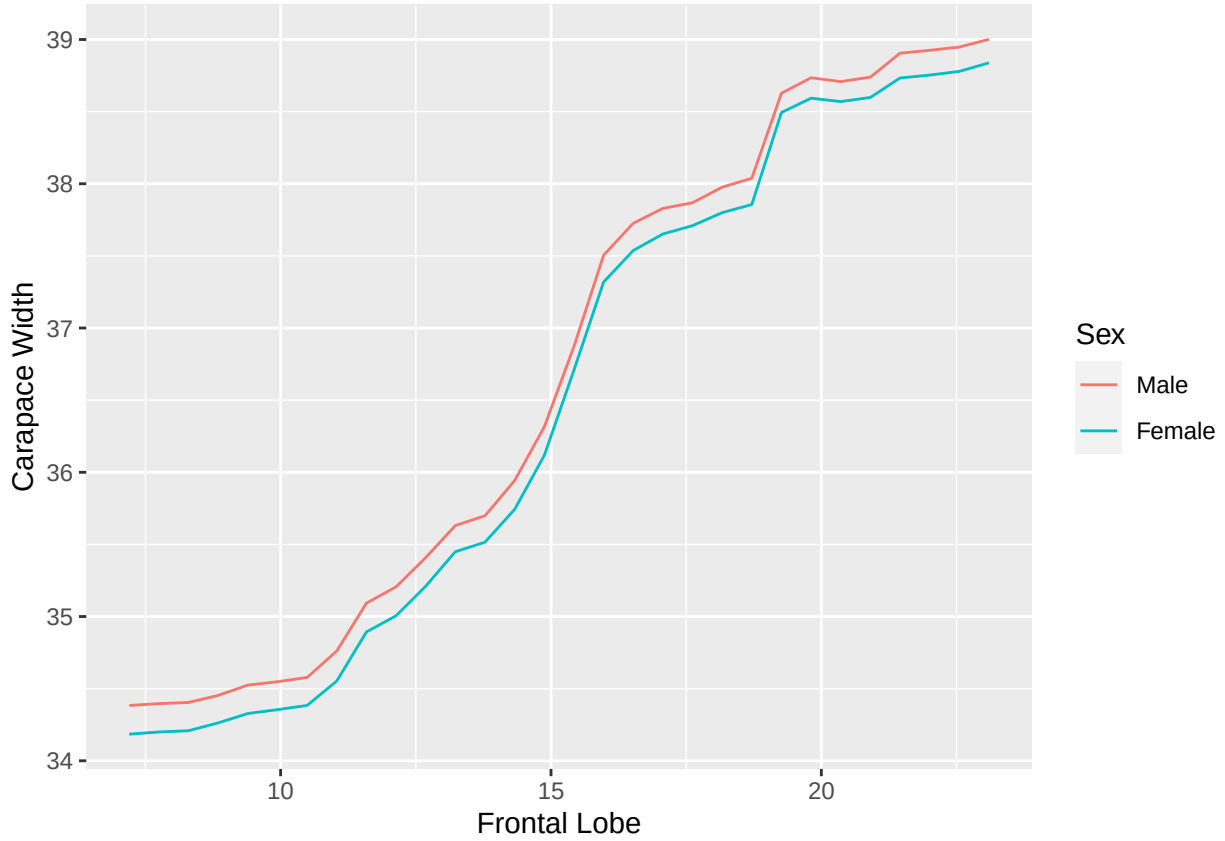
```
# check plotter wrapper
forest_plot <- Plotter$new(forest_interpret, features = c("Frontal Lobe"),
                           features.2d = data.frame(col1 = c("Frontal Lobe", "Frontal Lobe"),
                                                    col2 = c("Species", "Sex")))

plots <- plot.Plotter(forest_plot)
plots[[1]]
```




```
plots[[2]]
```


Carapace Width
39
38
37
36
35
34
10
15
20
Frontal Lobe
Species
Orange
Blue

```
plots[[3]]
```



We can also adjust where the plots are centered at for each feature, and what grid of points are used in the evaluations that create the plots.

```
# test functions for the plotter
forest_plot$center.at$`Frontal Lobe`
```

```
## [1] 7.2
```

```
forest_plot$grid.points$`Frontal Lobe`
```

```
##  [1]  7.200000  7.748276  8.296552  8.844828  9.393103  9.941379 10.489655
##  [8] 11.037931 11.586207 12.134483 12.682759 13.231034 13.779310 14.327586
## [15] 14.875862 15.424138 15.972414 16.520690 17.068966 17.617241 18.165517
## [22] 18.713793 19.262069 19.810345 20.358621 20.906897 21.455172 22.003448
## [29] 22.551724 23.100000
```

```
# We can reset where we center the plot
set.center.at(forest_plot, "Frontal Lobe", 2) # for numeric
forest_plot$center.at$`Frontal Lobe`
```

```
## [1] 2
```