

Interpreting Black-Box Machine Learning Models Using Partial Dependence and Individual Conditional Expectation Plots

Ray Wright, SAS Institute Inc.

ABSTRACT

One of the key questions a data scientist asks when interpreting a predictive model is “How do the model inputs work?” Variable importance rankings are helpful for identifying the strongest drivers, but these rankings provide no insight into the functional relationship between the drivers and the model’s predictions.

Partial dependence (PD) and individual conditional expectation (ICE) plots are visual, model-agnostic techniques that depict the functional relationships between one or more input variables and the predictions of a black-box model. For example, a PD plot can show whether estimated car price increases linearly with horsepower or whether the relationship is another type, such as a step function, curvilinear, and so on. ICE plots enable data scientists to drill much deeper to explore individual differences and identify subgroups and interactions between model inputs.

This paper shows how PD and ICE plots can be used to gain insight from and compare machine learning models, particularly so-called “black-box” algorithms such as random forest, neural network, and gradient boosting. It also discusses limitations of PD plots and offers recommendations about how to generate scalable plots for big data. The paper includes SAS® code for both types of plots.

INTRODUCTION

After assessing a model’s accuracy, data scientists often want to know how the model’s predictions vary depending on the values of the inputs. This knowledge can help data scientists identify flaws in their models, select from among competing models, and explain their models to stakeholders such as consulting clients, credit card applicants, and medical patients.

In the days of small data sets and relatively simple models, interpreting predictive models was fairly straightforward. For example, the coefficients from a linear regression model indicate the strength and direction of the relationship between a model input and the model’s predictions. Small decision trees are also easily understood by data analysts. But although so-called “black box” machine learning algorithms such as neural network, gradient boosting, and random forest are capable of highly accurate predictions, their inner workings can be very difficult to grasp because these algorithms are enormously complex.

Partial dependence (PD) plots (Friedman 2001) and individual conditional expectation (ICE) plots (Goldstein et al. 2014) are highly visual, model-agnostic tools that can help you interpret modern machine learning models. PD plots show how values of model inputs affect the model’s predictions. ICE plots, which are closely related to PD plots, let you drill down further to identify individual differences, interesting subgroups, and interactions among model variables.

PD and ICE are post hoc methods of model interpretation, meaning that they do not reveal a model’s inner workings; rather, they show how the model behaves in response to changing inputs. The difference is similar to the difference between looking under the hood of a sports car and observing how the car responds when you operate the driver’s controls. Nonetheless, PD and ICE plots are popular and highly visual tools for obtaining a working understanding of increasingly complicated machine learning models.

PARTIAL DEPENDENCE PLOTS

A partial dependence plot depicts the functional relationship between a small number of model inputs (generally one or two inputs) and a model’s predictions. PD plots are thus named because they show how the model’s predictions partially depend on values of the input variables of interest.

ONE-WAY PD PLOTS

The simplest PD plots are one-way plots, which show how a model's predictions depend on a single input to the model. For example, Figure 1 shows the relationship between horsepower and predicted MSRP (manufacturer's suggested retail price in US dollars) for automobile models.

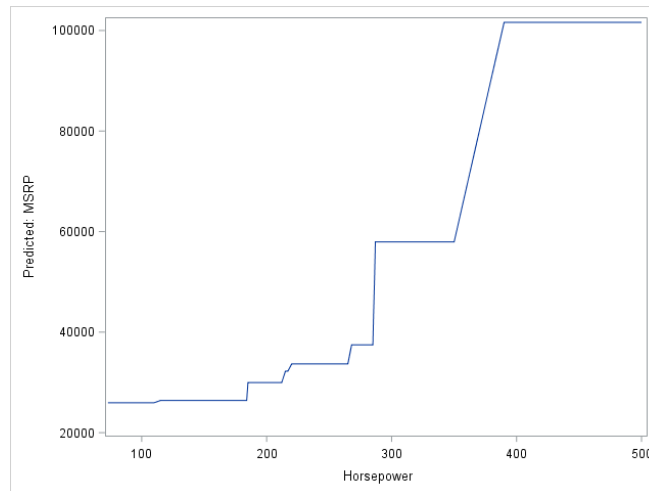


Figure 1. Partial Dependence Plot for Horsepower

Here the model's estimate of MSRP is a step function: MSRP tends to increase with horsepower, but there are sharp increases in expected MSRP for certain values of horsepower. There is a price premium of approximately \$20,000 for cars that have more than 285 horsepower, and an additional (and even greater) premium of about \$40,000 for cars that have about 400 horsepower. Expected MSRP levels off for higher values of horsepower.

PD plots can be used with any supervised learning algorithm. The following block of code uses a decision tree to predict MSRP. As is often true, the model includes several other inputs such as engine size, number of cylinders, and origin. Each value shown in a PD plot represents a prediction for a particular value of horsepower while averaging out the effects of the other (complementary) model variables.

```
proc hpsplit data=sashelp.cars leafsize = 10;
  target MSRP / level = interval;
  input horsepower engineSize length cylinders weight
  MPG_highway MPG_city wheelbase / level = int;
  input make driveTrain type / level = nominal;
  code file="treeCode.sas";
run;
```

The next block calls the %PDFunction macro (described in detail in a later section) to compute the partial dependence function for horsepower:

```
%PDFunction(
  dataset=sashelp.cars,
  target=MSRP,
  PDVars=horsepower,
  otherIntervalInputs=engineSize length cylinders weight
  MPG_highway MPG_city wheelbase,
  otherClassInputs=origin make driveTrain type,
  scorecodeFile=treeCode.sas,
  outPD=partialDependence
);
```

Finally, the next block calls the SGPLOT procedure to plot the partial dependence function, which is shown as a series plot in Figure 1:

```
proc sgplot data=partialDependence;
  series x = horsepower y = AvgYHat;
run;
quit;
```

You can create PD plots for model inputs of both interval and classification variables. Figure 2 shows the partial dependence for the nominal (classification) variable Make.

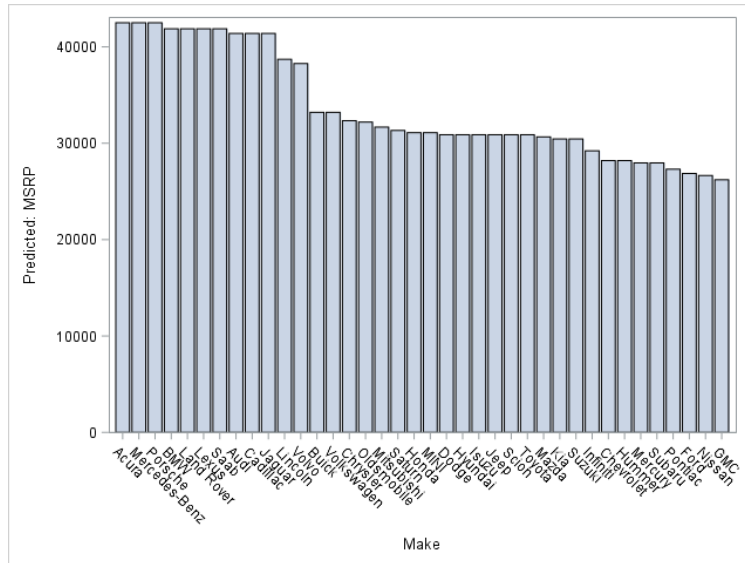


Figure 2. Partial Dependence Function for Make

As you might expect, predicted MSRP is highest for luxury brands such as Acura, Mercedes-Benz, and Porsche. The following code generates the bar chart:

```
%PDFunction(
  dataset=sashelp.cars,
  target=MSRP,
  PDVars=make,
  otherIntervalInputs=horsepower engineSize length cylinders
  weight MPG_highway MPG_city wheelbase,
  otherClassInputs=origin driveTrain type,
  scorecodeFile=treeCode.sas,
  outPD=partialDependence
);

proc sgplot data=partialDependence;
  vbar make / response = AvgYHat categoryorder = respdesc;
run;
quit;
```

TWO-WAY PD PLOTS

Because one-way PD plots display one variable at a time, they are valid only if the variable of interest does not interact strongly with other model inputs. However, interactions are common in actual practice; in these cases, you can use higher-order (such as two- and three-way) partial dependence plots to check for interactions among specific model variables. For example, Figure 3 shows an interaction between the model variables Horsepower and Origin:

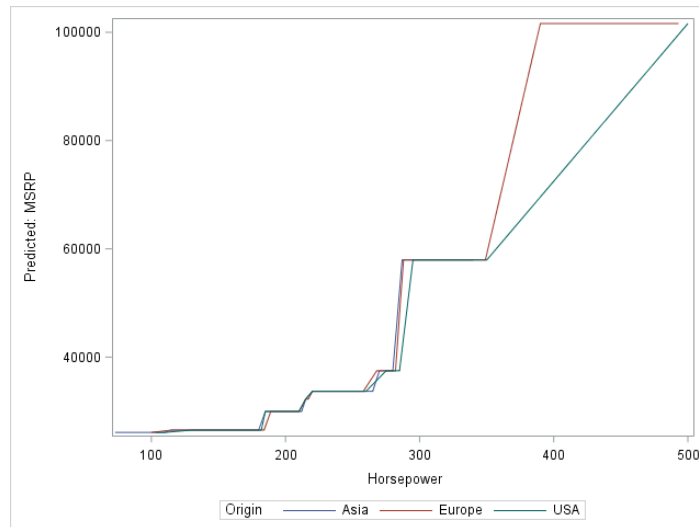


Figure 3. Partial Dependence for Horsepower by Origin

Figure 3 shows a separate PD function for each region (Asia, Europe, and USA). Consistent with the one-way plot for Horsepower in Figure 1, MSRP increases monotonically with horsepower for each region. But the two-way plot shows that powerful European cars (more than 350 horsepower) have much higher expected prices than their American counterparts, an interaction that was not apparent in the one-way plot.

Two-way PD plots are computed much like one-way PD plots. The difference is that two-way plots compute average prediction for each *combination* of values of the plot variables. Here are the %PDFunction and PROC SGLOT calls that generate Figure 3:

```
%PDFunction(  
  
    dataset=sashelp.cars,  
    target=MSRP,  
    PDVars=horsepower origin,  
    otherIntervalInputs=engineSize length cylinders weight  
        MPG_highway MPG_city wheelbase,  
    otherClassInputs=make driveTrain type,  
    scorecodeFile=treeCode.sas,  
    outPD=partialDependence  
  
);  
  
proc sgplot data=partialDependence;  
    series x = horsepower y = AvgYHat / group = origin;  
run;  
quit;
```

Figure 3 plots an interval input by a nominal input. Figure 4 is a scatter plot for two interval variables, Horsepower and MPG_City. PD plots of two interval variables are typically gradient scatter plots, response surfaces, or 3-D plots.

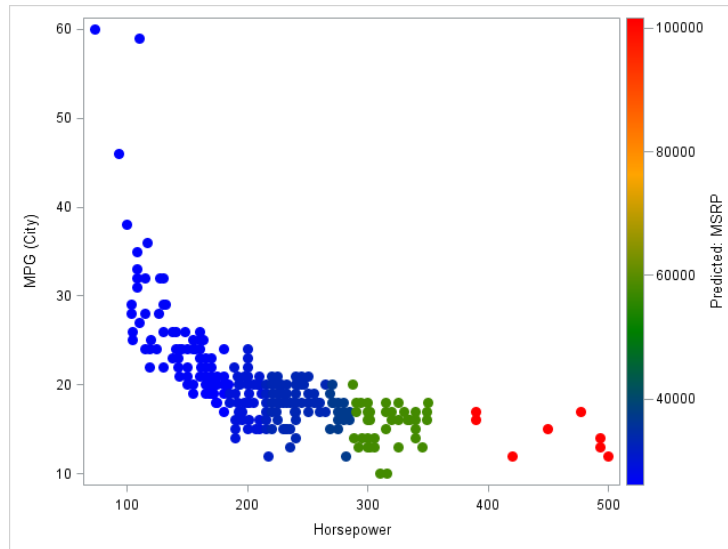


Figure 4. Partial Dependence for Horsepower by MPG_City

No interaction is apparent among the plot variables in Figure 4: MSRP appears to increase with horsepower regardless of the car's mileage estimate. The following %PDFunction macro call and PROC SGPLOT code generate the PD plot:

```
%PDFunction(

    dataset=sashelp.cars,
    target=MSRP,
    PDVars= horsepower MPG_City,
    otherIntervalInputs= engineSize cylinders MPG_highway wheelbase
        length weight,
    otherClassInputs= make origin type driveTrain,
    scorecodeFile=treeCode.sas,
    outPD=partialDependence

);

proc sgplot data=partialDependence;
    scatter x = horsepower y = MPG_City /
        colorresponse = avgYHat
        colormodel=(blue green orange red)
        markerattrs=(symbol=CircleFilled size=10) ;
run;
quit;
```

COMPUTING THE PD FUNCTION

To create a traditional PD plot (such as Figure 1), you must first compute the PD function by using the following steps. These steps are illustrated using hypothetical data in Figure 5 for a one-way plot.

1. Find the unique values of the plot variable (for a one-way plot) or variables (for a higher-order plot) in the training set and identify the complementary variables.
2. Create one replicate of the training set for each unique value of the plot variable, and fix the value of the plot variable. For complementary variables, use the same values as in the training set.
3. Score each replicate by using your predictive model.
4. Compute the average predicted value within each replicate.

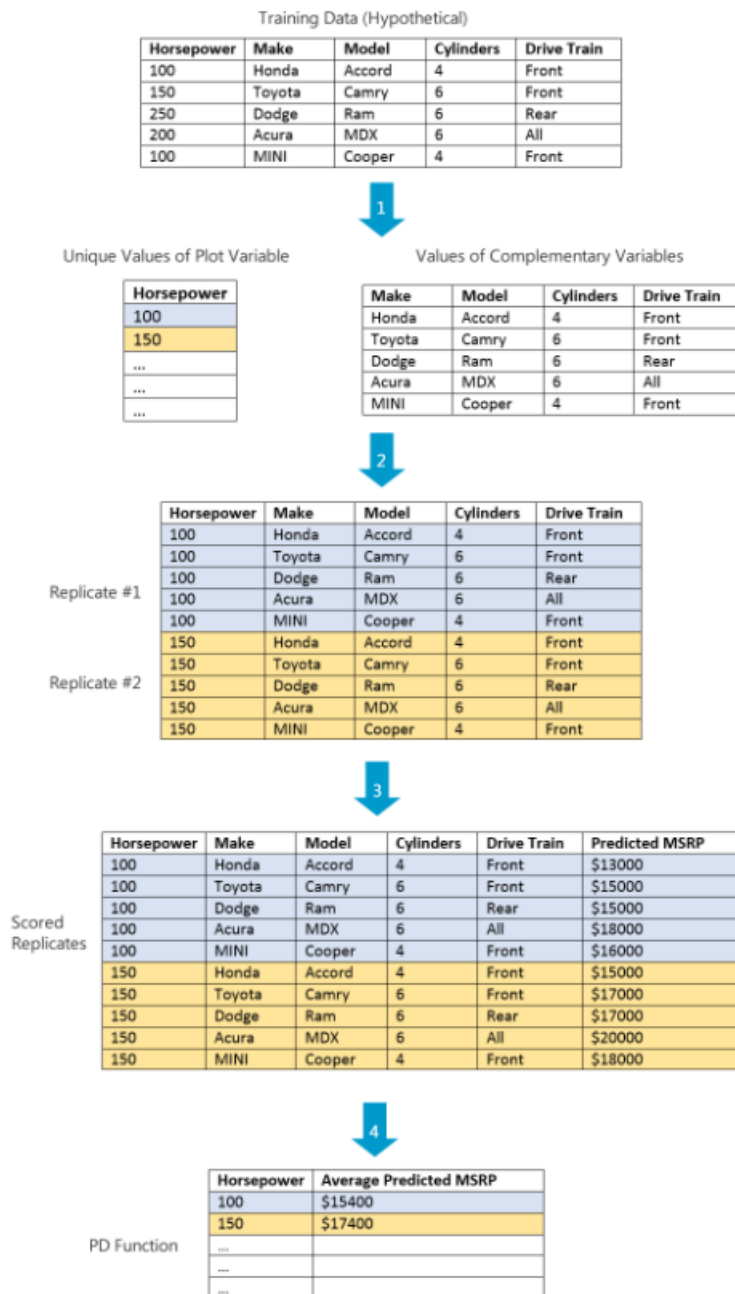


Figure 5. Computing the PD Function

As you can see from the last step in Figure 5, each value of the PD function represents an average prediction for a particular value of the plot variable. A simpler way to depict the functional relationship between the horsepower and MSRP variables might be to fix the values of the complementary variables at their average (or modal) values. But PD functions have the advantage that each value reflects the actual joint distribution of the complementary variables in the training set.

THE %PDFUNCTION MACRO

The following %PDFunction macro is called in the examples:

```

%macro PDFunction(
    dataset=,
    target=,
    PDVars=,
    otherIntervalInputs=,
    otherClassInputs=,
    scoreCodeFile=,
    outPD=
);

%let PDVar1 = %sysfunc(scan(&PDVars,1));
%let PDVar2 = %sysfunc(scan(&PDVars,2));

%let numPDVars = 1;
%if &PDVar2 ne %str() %then %let numPDVars = 2;

/*Obtain the unique values of the PD variable */
proc summary data = &dataset.;
    class &PDVar1. &PDVar2.;
    output out=uniqueXs
        %if &numPDVars = 1 %then
            %do;
            (where=( _type_ = 1))
            %end;
        %if &numPDVars = 2 %then
            %do;
            (where=( _type_ = 3))
            %end;
    ;
run;

/*Create data set of complementary Xs */
data complementaryXs;
    set &dataset(keep= &otherIntervalInputs. &otherClassInputs.);
    obsID = _n_;
run;

/*For every observation in uniqueXs, read in each observation
   from complementaryXs */
data replicates;
    set uniqueXs (drop=_type_ _freq_);
    do i=1 to n;
        set complementaryXs point=i nobs=n;
        %include "&scoreCodeFile.";
    output;
end;
run;

```

```

/*Compute average yHat by replicate*/
proc summary data = replicates;
  class &PDVar1. &PDVar2.;
  output out=&outPD.
    %if &numPDVars = 1 %then
      %do;
      (where=(_type_ = 1))
      %end;
    %if &numPDVars = 2 %then
      %do;
      (where=(_type_ = 3))
      %end;
  mean(p_&target.) = AvgYHat;
run;

%mend PDFunction;

```

The macro requires the following input parameters:

- dataset: Specify the training set.
- target: Specify the target variable to use in the predictive model.
- PDVars: For one-way plots, specify one variable; for two-way plots, specify two model variables.
- otherIntervallInputs: Specify the complementary model variables whose measurement level is interval.
- otherClassInputs: Specify the complementary model variables whose measurement level is nominal or binary.
- scoreCodeFile: Specify the score code from the machine learning model.
- outPD: Name the output data set to contain the PD function.

The %PDFunction macro is intended to introduce the concept of partial dependence and might not scale well to large data sets. As the number of unique values and observations increase, the number of replicated observations can grow out of hand. To avoid creating too many replicates for larger data sets, you can consider alternative approaches such as the following:

- Bin unique values of high-cardinality inputs such as income.
- Sample or cluster observations.
- Avoid stacking the replicates altogether: process the replicates one at a time, keeping only the average predicted value for each replicate.

Subsequent examples use one or more of these strategies to greatly reduce the number of rows that are replicated.

INDIVIDUAL CONDITIONAL EXPECTATION PLOTS

Whereas PD plots provide a coarse view of a model's workings, ICE plots enable you to drill down to the level of individual observations. Essentially, ICE plots disaggregate the PD function (which, after all, is an average) to reveal interactions and individual differences. To avoid visualization overload, ICE plots show one model variable at a time.

This section shows an example that uses generated data. The PD function in Figure 6 is essentially flat, giving the impression that there is no relationship between X1 and the model's predictions.

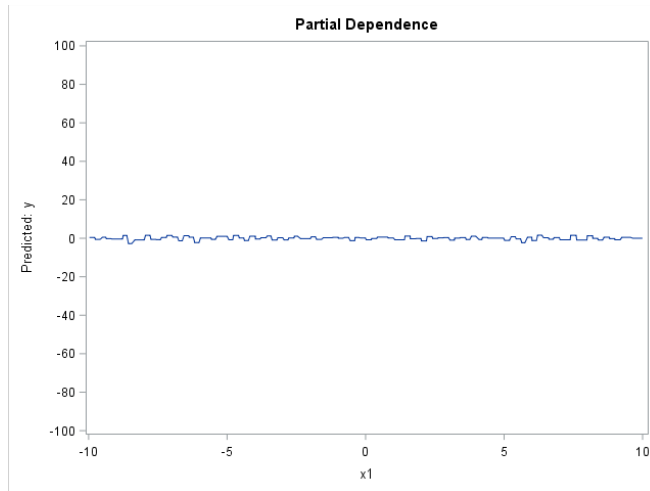


Figure 6. PD Plot for X1

Figure 7 is an ICE plot for two observations in the same data set.

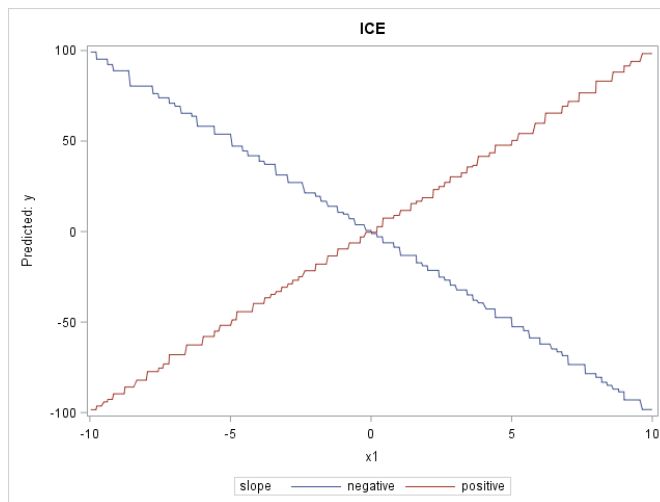


Figure 7. ICE Plot for X1

The ICE plot presents a much different picture: the relationship is strongly positive for one observation, but strongly negative for the other observation. So despite the PD plot, the ICE plot shows that X1 is actually related to the target; it's just that there are strong individual differences in the nature of that relationship.

Traditional ICE plots display one curve for each observation in the training set, but plotting a curve for every observation can result in visualization overload even for data sets of moderate size. Fortunately, you can manage the number of curves that are displayed by sampling or clustering. Figure 8 shows the dependence of predicted MSRP on horsepower for 10 randomly selected observations.

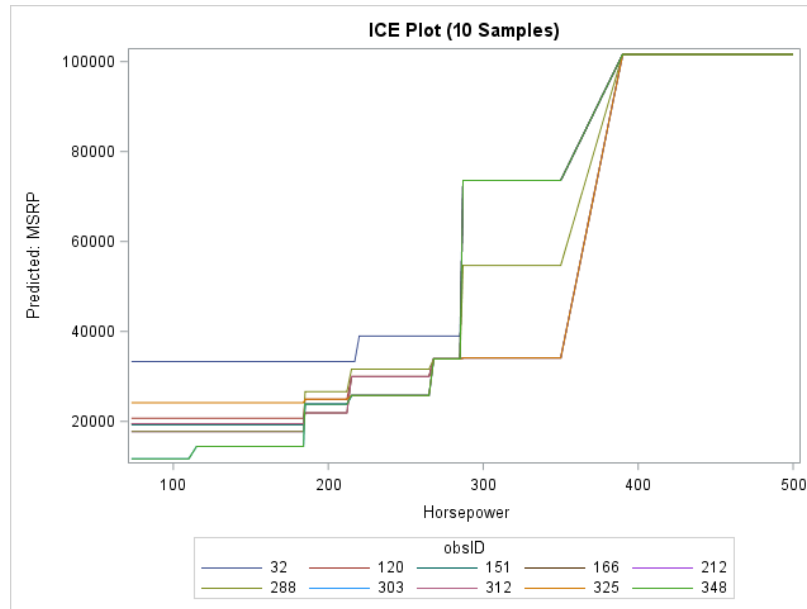


Figure 8. ICE Plot for Horsepower

Each series in the plot represents a different car model. Although some overlap is apparent among the individual curves, expected MSRP clearly diverges in the 280–350 horsepower range. The predictive model indicates that if horsepower were varied (leaving the other characteristics of each car unchanged), the same change in horsepower would have a different effect on the MSRP of the various car models. Although testing this in real life might not be practical, the model’s prediction makes intuitive sense. It suggests that car manufacturers believe that customers are willing to pay for increased horsepower for some car models but not others.

COMPUTING THE ICE FUNCTION

You can think of each ICE curve as a kind of simulation that shows what would happen to the model’s prediction if you varied one characteristic of a particular observation. As illustrated in Figure 9, the ICE curve for one observation is obtained by replicating the individual observation over the unique values of the plot variable and scoring each replicate.

Unique Values of Plot Variable

Horsepower
100
150
200
250
300
...

Complementary Variables for One Observation

Make	Model	Cylinders	Drive Train
Honda	Accord	4	Front



Replicates

Horsepower	Make	Model	Cylinders	Drive Train
100	Honda	Accord	4	Front
150	Honda	Accord	4	Front
200	Honda	Accord	4	Front
250	Honda	Accord	4	Front
300	Honda	Accord	4	Front
...	Honda	Accord	4	Front



Scored Replicates

Horsepower	Make	Model	Cylinders	Drive Train	Predicted MSRP
100	Honda	Accord	4	Front	\$12,000
150	Honda	Accord	4	Front	\$15,000
200	Honda	Accord	4	Front	\$20,000
250	Honda	Accord	4	Front	\$26,000
300	Honda	Accord	4	Front	\$35,000
...	Honda	Accord	4	Front

Figure 9. Computing an ICE Curve for One Observation

THE %ICEPLOT MACRO

ICE plots are essentially plots of raw replicates. Thus, if you have computed the PD function for a single variable, you need to add only a few more steps to produce an ICE plot for that variable:

5. Sample individuals from the replicates data set that is created by the %PDFFunction macro.
6. Select the replicates that correspond to the sampled individuals.
7. Plot the sampled replicates as overlaid series.

The following %ICEPlot macro is used to plot the ICE curves:

```
%macro ICEPlot(
    ICEVar=,
    samples=10,
    YHatVar=
);

/*Select a small number of individuals at random*/
proc summary data = replicates;
    class obsID;
    output out=individuals (where=( _type_ = 1));
run;

data individuals;
    set individuals;
    random = ranuni(12345);
run;
```

```

proc sort data = individuals;
    by random;
run;

data sampledIndividuals;
    set individuals;
    if _N_ LE &samples.;
run;

proc sort data = sampledIndividuals;
    by obsID;
run;

proc sort data = replicates;
    by obsID;
run;

data ICEReplicates ;
    merge replicates sampledIndividuals (in = s);
    by obsID;
    if s;
run;

/*Plot the ICE curves for the sampled individuals*/
title "ICE Plot (&samples. Samples)";
proc sgplot data = ICEReplicates;
    series x=&ICEVar. y = &yHatVar. / group=obsID;
run;

%mend ICEPlot;

```

Figure 8 is created by the following call of the %ICEPlot macro, which specifies a plot variable, the number of individual curves to sample, and the variable to contain the model's predicted values:

```

%ICEPlot(
    ICEVar=horsepower,
    samples=10,
    YHatVar=p_MSRP
);

```

EXAMPLE: NBA BASKETBALL SHOTS

This section presents an extended example that uses a larger data set and more complicated machine learning models. The data are a sample of 16,934 basketball shots taken in the NBA 2015–2016 regular season.

An autotuned neural network model is used to predict shot success. As shown in Table 1, model inputs include both shot and player characteristics.

Variable	Role	Measurement Level	Values
Shot outcome	Target	Binary	0=made,1=missed
Distance to basket	Input	Interval	In feet
Player experience	Input	Interval	In years
Player height	Input	Interval	In inches
Player weight	Input	Interval	In pounds
Player position	Input	Nominal	Center, guard, or forward
Shot style	Input	Nominal	Jump, layup, hook, or dunk
Shot location	Input	Nominal	Right, left, center, left center, or right center
Shot area	Input	Nominal	Mid-range, restricted area (RA), in the paint (non-RA), above the break 3, right corner 3, left corner 3

Table 1. Model Variables

Because predictive models can have many inputs, it is customary to focus on the most important inputs when interpreting the model. According to a decision tree model, the most important predictors for the shot success model are distance to basket, shot style, and shot location (in that order). Figure 10 through

Figure 12 show partial dependence for the top model inputs in order of their relative importance.

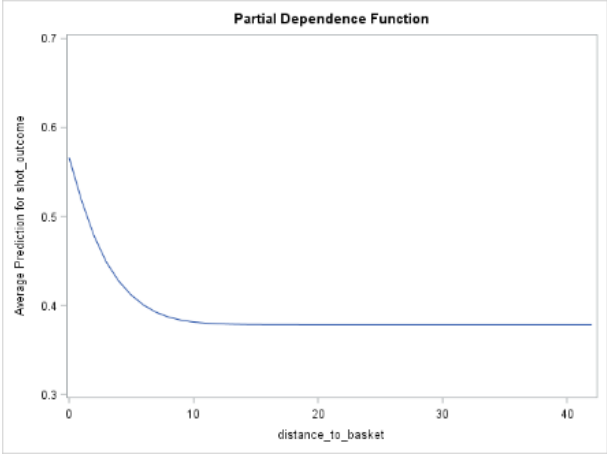


Figure 10. Distance to Basket

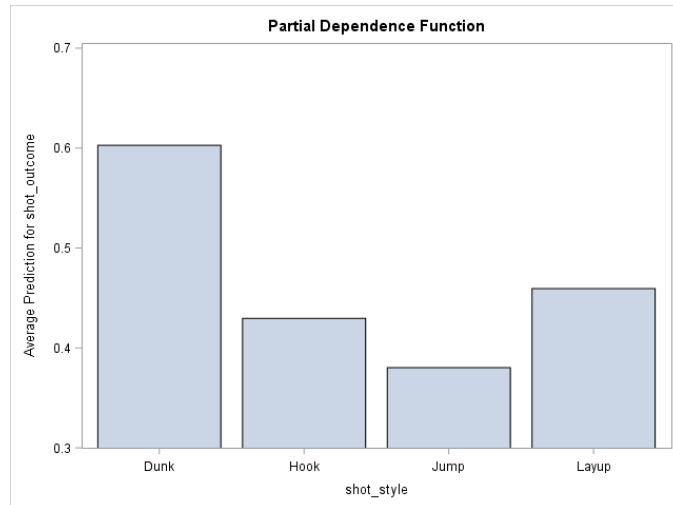


Figure 11. Shot Style

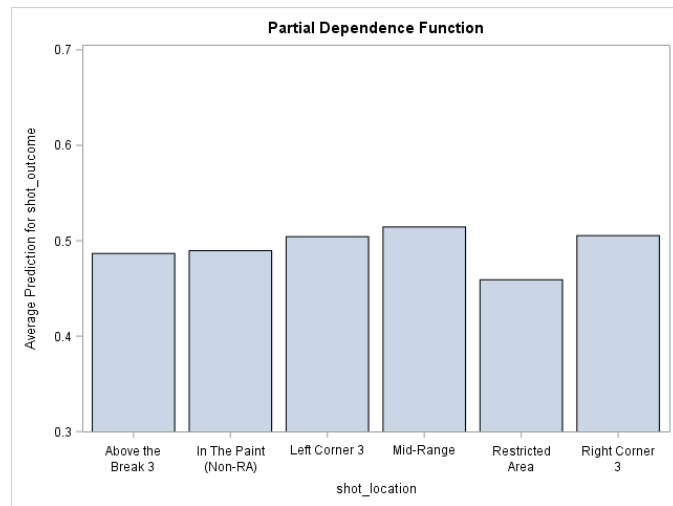


Figure 12. Shot Location

Figure 10 and

Figure 11 show that there is a success advantage for shots taken near the basket and for dunk shots, respectively. Because the plots use a consistent Y-axis scale, you can compare the success rate for each type of shot. For example, shots taken near the basket are approximately 20 percentage points more successful than those taken 10 feet away. This is about the same advantage as for dunk shots over hook shots.

Figure 12 shows a disadvantage for shots taken in the restricted area, although the effect of shot location is small compared to the difference among the different shot styles.

Typically data scientists build multiple candidate models and choose a champion based on both its accuracy and some assessment of its validity. Figure 13 compares PD functions for a neural network, an autotuned gradient boosting model, and an autotuned random forest.

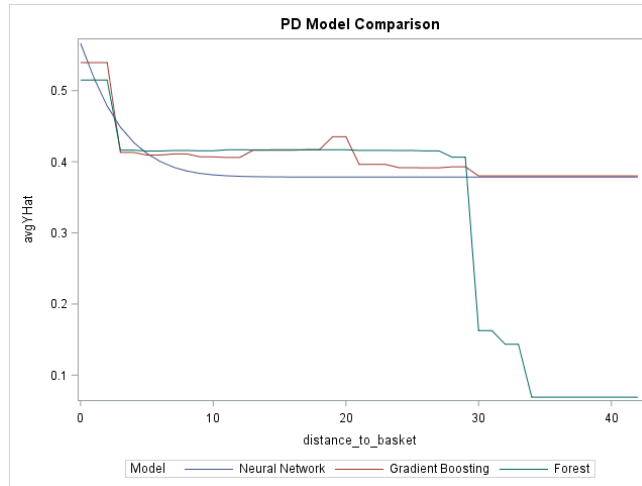


Figure 13. Comparison of Three Models

The models represent three different perspectives: For the neural network, the predicted effect of distance to basket on shot success is nonlinear and asymptotic, whereas the effect is more steplike for the other two models. Only one candidate, the forest, predicts a steep drop-off in success for shots taken from around 28 feet or more. That seems intuitive because long-distance shots are typically taken out of desperation. Overlaying PD functions in this manner can help you choose models that not only are accurate but also make intuitive sense and are acceptable to consumers of the model.

Let's see whether ICE plots reveal any interesting interactions or subgroups. Figure 14 is an ICE plot for distance to basket for the neural network model.

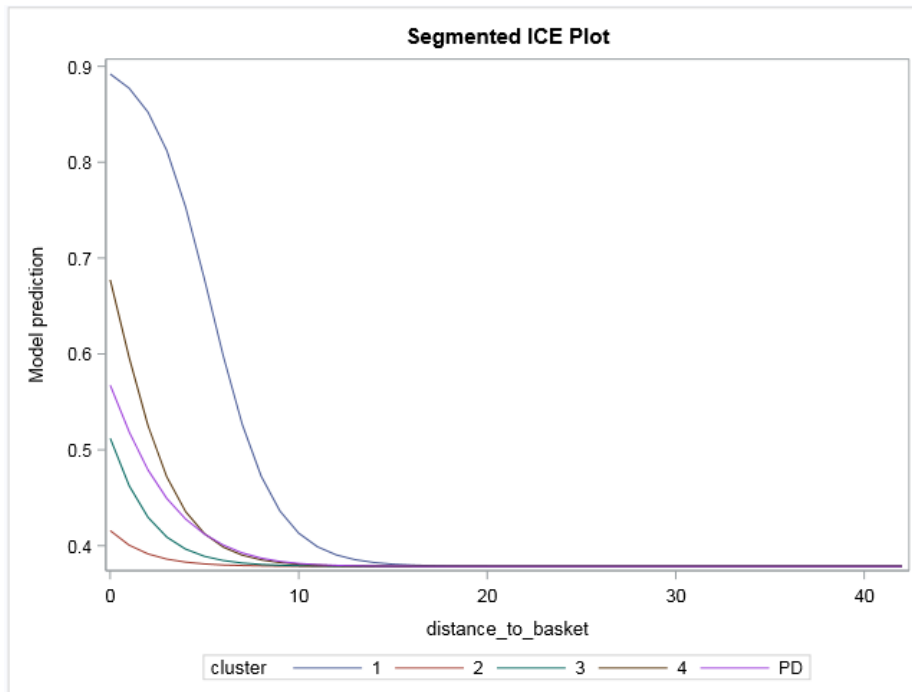


Figure 14. Distance to Basket

This plot actually is a *segmented* ICE plot. Unlike a traditional ICE plot, in which each curve represents a single observation, each curve in a segmented plot represents a cluster of observations whose ICE curves have a similar shape. By plotting representative clusters rather than observations, the number of curves is greatly reduced, making the plot easier to digest. Figure 14 also includes the PD function (the overall average) for reference.

There are two things to look for in ICE plots: intersecting slopes, which indicate interactions between the plot variable and one or more complementary variables, and level differences, which indicate group effects. The slopes of the various clusters are nearly parallel, indicating a lack of strong interaction between distance to basket and the other model variable. But notice the dark blue line (cluster 1): it indicates a marked success advantage for certain types of shots compared to other shots taken at the same distance to basket. It turns out that this cluster includes a disproportionate number of dunk shots taken in the restricted area at center court. These shots had an extremely high success rate of 84.6%.

An alternative to clustering the individual curves is to group them by values of other model variables.

Figure 15 and Figure 16 show ICE curves for distance to basket for 250 randomly selected shots that are grouped by shot style and location, respectively:

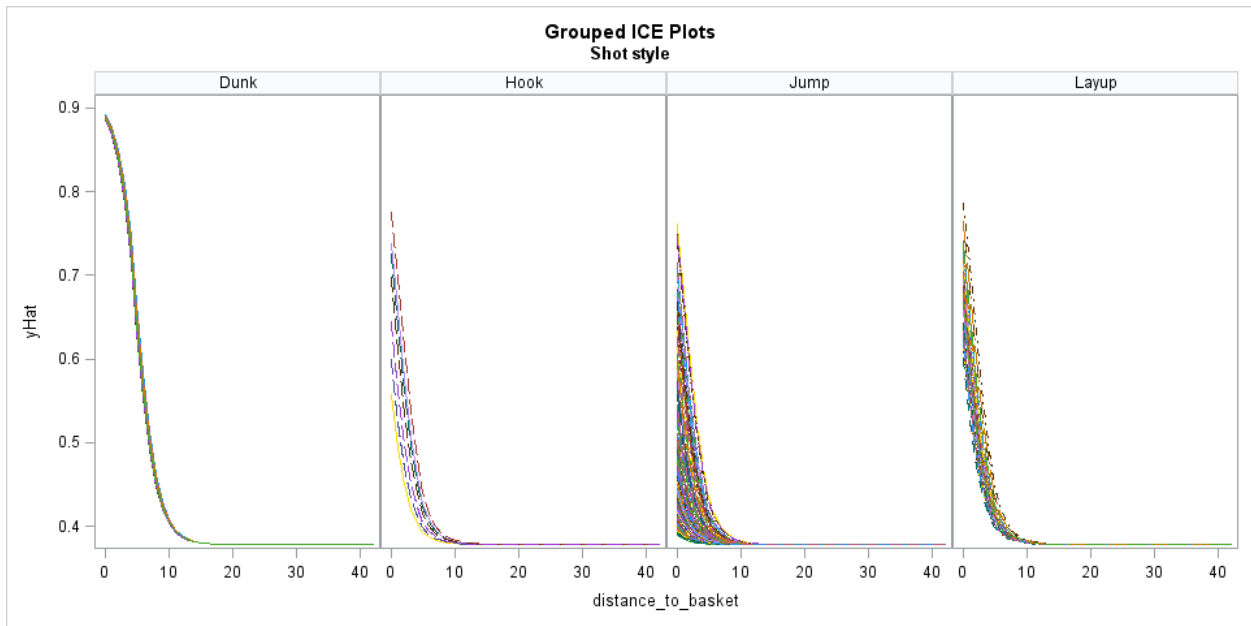


Figure 15. Grouped ICE Plot: Shot Style

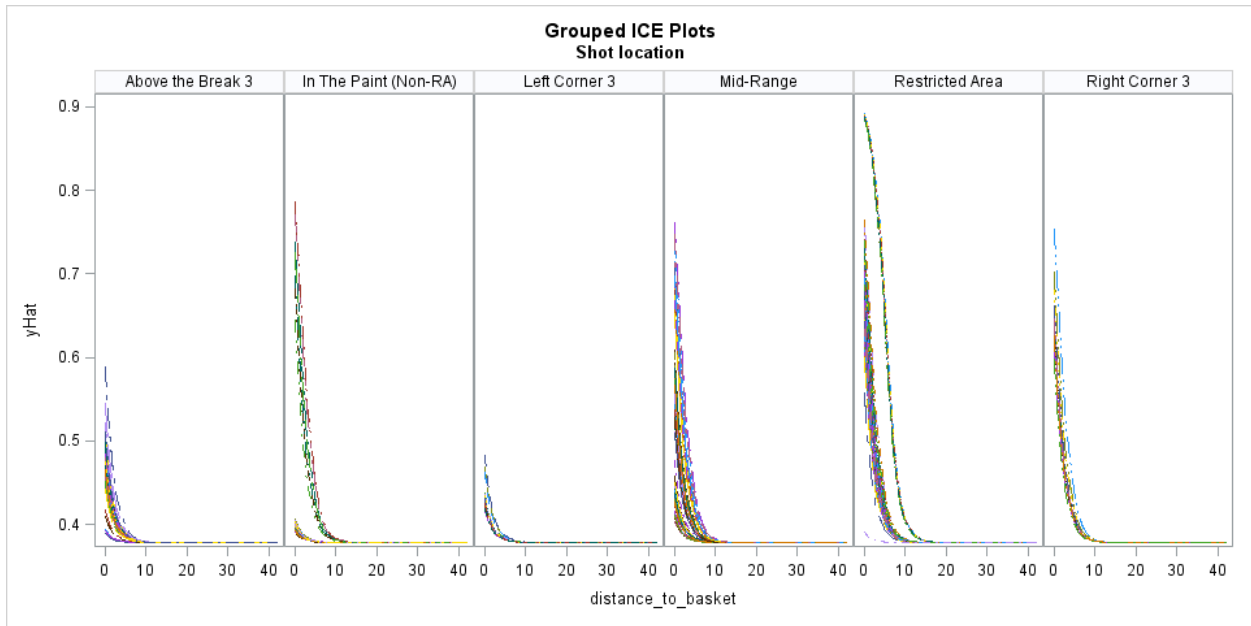


Figure 16. Grouped ICE Plot: Shot Location

When ICE curves are grouped by shot style, it appears that dunk shots are more advantageous than other shot styles. This is consistent with the PD plot for shot style. But when ICE curves are grouped by shot location, it appears that shots taken in the restricted area include high-, medium-, and low-probability shots. In other words, some shots in the restricted area are actually advantageous—a finding that is not apparent from the PD plot. After segregating the ICE curves in this manner, you could drill even deeper to explore why some shots in the restricted area are more successful than others.

SAS® VISUAL DATA MINING AND MACHINE LEARNING

SAS Visual Data Mining and Machine Learning enables you to run machine learning pipelines that build powerful supervised learning models. Figure 17 shows a pipeline that runs three autotuned black-box algorithms to predict shot success: neural network, gradient boosting, and forest.

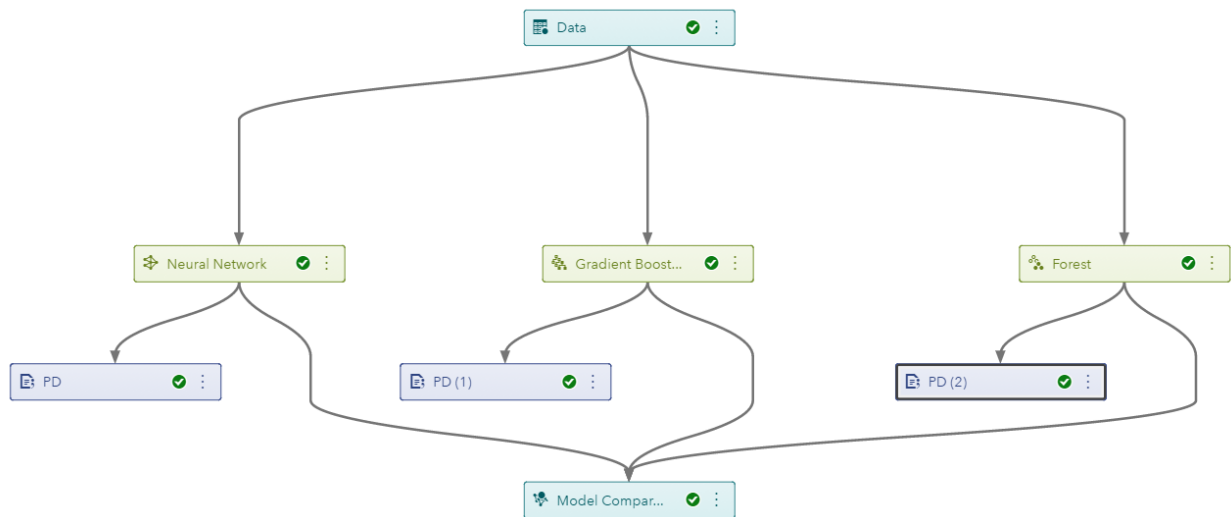



Figure 17. Model Studio Pipeline

Model Studio, the visual interface to SAS Visual Data Mining and Machine Learning, enables you to use a Code node to generate partial dependence plots. Each node that immediately follows a modeling node is a Code node that produces PD plots for the predecessor model. (The Code nodes are identified by the ) For example, Figure 18 shows Code node output for the forest model.

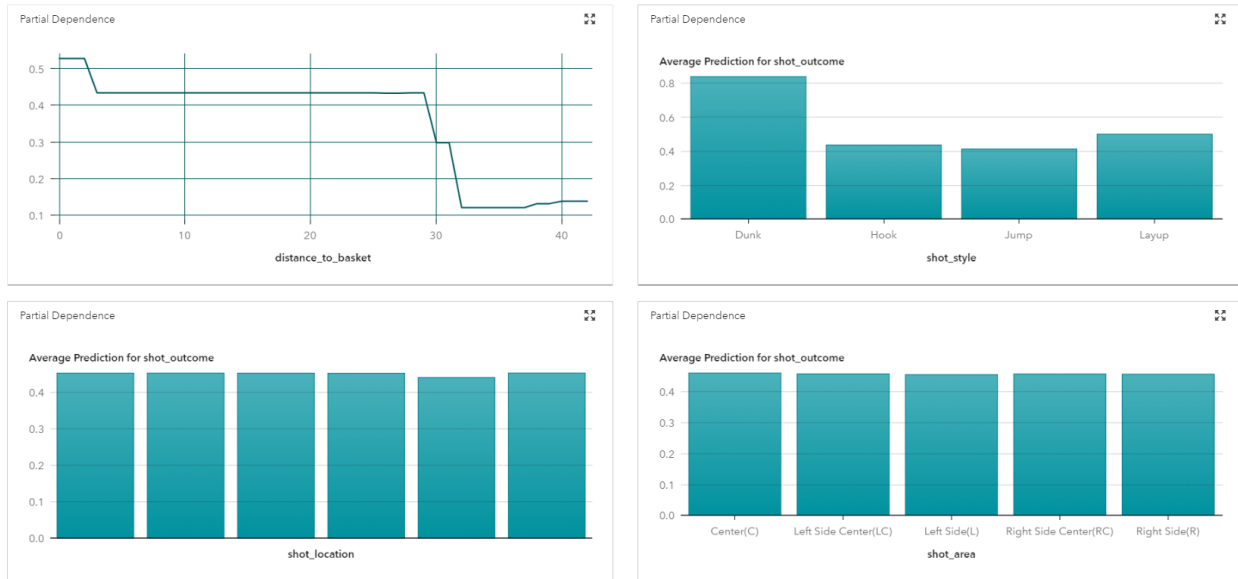


Figure 18. Partial Dependence Results

Each Code node calls the %partialDep macro (available at <https://github.com/sassoftware/sas-viya-machine-learning/tree/master/interpretability>) to create for each requested input a PD plot that is appropriate to that input's measurement level. Depending on the modeling algorithm used in the pipeline, the macro obtains either score code or the analytics store from the preceding modeling node. To limit the number of replicated observations, the macro samples observations by default.

Using the macro is straightforward: simply download the macro, paste it into a Code node, and then call it, specifying the number of important inputs to plot and the proportion of observations to sample. For example, the following macro call requests PD plots for four inputs and uses a 10% sample of the observations:

```
%partialDep(
    importantInputs=distance_to_basket shot_style shot_location,
    obsSampProp=.10
);
```

CONCLUSION

Modern machine learning algorithms can be incredibly powerful predictors but their inner workings are often difficult for data scientists to digest. PD plots help you understand how your model works by depicting how changes in input values affect a model's predictions, and you can use them to evaluate competing models. ICE plots enable you to drill deeper to find interactions among model variables and unusual subgroups in your data.

Although you can easily compute traditional PD and ICE plots, you might need to make some adjustments for efficient computation with large data sets. Several options are available, including binning of plot variables, sampling of rows, and sampling or clustering of observations and ICE curves. You can use these techniques individually or in combination to produce reasonable approximations of traditional plots in a fraction of the time.

Although PD and ICE plots provide only indirect approximations of a model's workings, they are popular and highly visual tools for obtaining a working understanding of increasingly complicated machine learning models.

REFERENCES

Friedman, J.H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, 29, pp. 1189–1232.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2014). "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation." arXiv:1309.6392v2.

ACKNOWLEDGMENTS

Thanks to Funda Gunes for critical feedback.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Ray Wright
SAS Institute Inc.
Ray.wright@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.