

# Unsupervised and Supervised Propensity Scoring in “R”:

Nearest Neighbor, Instrumental Variable, Parametric Prediction and Smoothing Methods of Adjustment for Treatment Selection Bias in Non-Standard Studies.

**USPSinR, Version 1.1-0, Aug 2007**

**Bob Obenchain, PhD, Senior Research Advisor  
US Medical, Outcomes Research  
Eli Lilly & Company, Indianapolis IN, 46285**

This user’s guide defines syntax and illustrates use of updated “R” functions that perform a variety of alternative approaches to Propensity Scoring (PS) analyses. These functions implement a variety of relatively new methods for statistical inference that use either “local control” patient clustering or else traditional “covariate adjustment” (parametric prediction) to analyze data from different types of non-standard studies ...such as observational studies, retrospective database analyses and poorly randomized (chaotic) studies. USPS methods do not rely on the “balance” that is “expected” when using traditional randomized assignment of patients to treatments. After all, this “balance” frequently fails to result even **approximately** in finite samples!!! Instead, the USPS methods implement various forms of a-posteriori “matching” or “stratification” of patients who received only one of the two treatments that are being compared.

## Table of Contents

	Page
<b>Introduction</b>	2
Distinctions between Alternative PS Approaches	5
The Fundamental (Balancing) Theorem of Propensity Scoring	5
Numerical Illustrations: Effects of Abciximab use on Survival and Total Cardiac Billing	7
Preliminary Setup of R Functions	8

	<b>Part I: Unsupervised Propensity Scoring Methods</b>	11
Step One:	Specify X Covariates and Compute Hierarchical Clustering Tree using USPhclus( )	11
Step Two:	Specify Treatment and Outcome Variables with UPSaccum( ) to start Accumulating NN and/or IV information	14
Step Three:	Compute NN Local Treatment Differences (LTD) for a specified Number of Cluster-Bins with UPSnnltd( )	14
Step Four:	Compute IV Local Treatment Averages (LTA) for a specified Number of Cluster-Bins with UPSivadj( )	16
Step Five:	Graphically Summarize Sensitivity of NN and/or IV Analyses to Choice of Number of Cluster-Bins with UPSgraph( )	19
Step Six:	Compare the UPS LTD Distribution for a given number of Clusters with the Corresponding “Artificial” (potentially biased) Distribution from Random Clusters with UPSaltd( )	20
	<b>Part II: Supervised Propensity Scoring Methods</b>	22
Step One:	Estimate Parametric Propensity Scores and Form Bins using SPSlogit( ) and SPSnbins( )	22
Step Two:	Check for Within-Bin Balance on Covariates using SPSbalan( )	24
Step Three:	Display Outcome Differences within Bins using SPSoutco( )	27
Step Four:	Display Outcome Differences expressed as smooth (lowess or cubic spline) functions using SPSloess( ) and SPSsmoot( )	30
	<b>Technical Matters</b>	
R and S-Plus functions that are (or could be) called in NN and IV analyses		36
Summary		37
References		38
Software Update History		39
Brief Synopsis of UPS and SPS Functions and Their Arguments		39

## 1. Introduction.

Here we describe “R” language functions implementing rather new methods for “propensity score” (PS) and/or “instrumental variable” (IV) adjustment to estimates of treatment effects. These approaches adjust for treatment selection bias, characterized by imbalance in patient baseline characteristics between treatment groups (arms, cohorts) in either nonrandomized or poorly randomized studies. Traditional “supervised” PS methods can be categorized as follows:

The traditional “big three” Propensity Scoring methods require an estimate of the PS = (conditional probability of treatment) for each patient ...usually from a fitted logit or probit model.

- [a] PS matching of patients in any fixed ratio (as in case/control studies),
- [b] PS binning of patients (sub-classification / stratification) and
- [c] regression modeling using Heckman effects or inverse Mills’ ratios (nonlinear functions of PS).

See D’Agostino(1998) for a gentle introduction to these three relatively well-know methods of supervised propensity scoring.

Two alternative (rather more technical) methods are:

- [d] inverse probability weighting ( $IPW = 1 / PS$ ) and
- [e] econometric simultaneous equations / instrumental variable models.

For key references on these methods, see my “white paper,” Obenchain(2006a).

“Unsupervised” PS strategies start by clustering patients in baseline covariate X-space. The two new approaches of this type that are implemented in “R” here are:

- [f] Nearest Neighbor / Local Treatment Differences (NN/LTD) plotting.

NN/LTD focuses on characterizing the full distribution of truly “local” treatment differences within clusters of relatively well-matched patients.

- [g] Instrumental Variable / Local Outcome Averages (IV/LOA) plotting.

IV/LOA focuses on how within-cluster outcome averages (regardless of treatment) vary when clusters are plotted versus a within-cluster PS estimate, the observed treatment percentage. This approach requires all X covariates used to define clusters to be “instrumental variables;” i.e. to effect outcome only (indirectly) through choice of treatment. Use of X covariates which quantify disease severity or patient frailty is then questionable because such characteristics may also have direct effects on outcomes, invalidating the IV/LOA plotting approach.

Our “Unsupervised” PS designation comes from familiar jargon in literature on artificial intelligence and data mining; see Barlow(1989). Specifically, PS clustering methods proceed without receiving any “hints” from a designated outcome measure or a treatment indicator variable to help “guide” formation of patient subgroups. In addition to clustering, multivariate probability density estimation is another example of an unsupervised statistical method.

The PS clustering approaches implemented here are like some early suggestions of Rubin(1980), which apparently have not been nearly as widely used as “supervised” methods. After all, they tend to be much more computationally intensive.

Some supervised methods do have to resort to numerical search methods over a p-dimensional space of parameters (e.g. estimation of a logit or probit regression model) rather than use a closed form expression (such as that for ordinary least squares estimates in a linear regression model.) On the other hand, unsupervised methods typically have to make the  $n \times (n-1)/2$  pairwise comparisons of patients needed to hierarchically cluster all study subjects. The resulting increases in computing time and memory allocation due to use of unsupervised (Non-Polynomial Hard) methods can be enormous, especially when n is much larger than p ( $n \gg p$ .) Besides, clustering results can also be highly sensitive to user choices of patient similarity metric and/or specific clustering algorithm.

In spite of their almost frustrating flexibility and sharply increased demand on computing power, “clustering” approaches to adjustment for treatment selection bias end up offering two main advantages over supervised PS methods...

[1] At least when the number of clusters is relatively large (and thus the number of patients in most clusters is relatively small), there is no real need to test whether patients are relatively “well matched” on their X characteristics within individual clusters. Clustering of patients on their X characteristics has assured that almost any model for predicting propensity scores that is relatively smooth over X space would confirm that any within-cluster comparisons of treatment outcome differences are relatively “fair” comparisons.

[2] Results from clustering lend themselves well to use of graphical visualization and sensitivity analysis techniques. For example, once the full hierarchical clustering tree has been constructed, displays using alternative numbers of total clusters can be generated relatively quickly. Thus clustering approaches can provide not only fundamental, robust (non-parametric) insights but also highly relevant information about sensitivity of results to “tuning” parameters. Furthermore, the resulting graphical displays can dramatically illustrate how traditional parametric modeling approaches, such as simultaneous equations models, tend to emphasize some aspects of the data while de-emphasizing other aspects.

Thanks to the considerable computing and graphical display power of modern workstations, clustering methods for PS adjustment are now practically useful. My key strategy is to emphasize their use in systematic sensitivity analyses: “What is the full range of quantitative treatment effect estimates supported by the available data?” And, “Which patient subsets tend to have extreme outcomes?”

Finally, “smoothing” of outcomes along a PS axis defined by a parametric prediction model is a simple and intuitively appealing alternative (initially suggested to me by Professor Frank Harrell of Vanderbilt) to the earlier PS “binning” approach.

The computing algorithms discussed here are written in a dialect of Version 3 of the “S” language that is processed by the “R” interpreter, Version 1.7+, available for download from <http://www.r-project.org>. R is a GNU (open source) implementation of the “S language and environment for data analysis and graphics.” See Becker, Chambers and Wilks(1988) and Chambers and Hastie(1992) for information about S; see Ihaka and Gentleman(1996) for the original description of R as being “not unlike” S.

## 2. Fundamental Distinctions between Alternative PS Approaches.

Traditional “Supervised” forms of Propensity Scoring (PS) typically start with a parametric logit (or probit) model to estimate the conditional probability of receiving treatment given certain patient characteristics. [An average across some randomForest() of classification trees may well be a superior way to “score” patients, but no R functions for this are currently provided here.]

The mandatory second step in all Supervised approaches is to verify that one’s estimated propensity scores behave, at least approximately, like (unknown) true propensity scores. Specifically, they must conditionally “balance” patients on all relevant X characteristic distributions within all bins. Sections 3 and 11 (Supervised PS Step 2), below, discuss this topic in detail.

The steps in actually using estimated propensity scores (PSs) involve: rank ordering patients on their estimated scores, sub-classifying patients into “bins” (strata), computing within-bin outcome differences between treatment subgroups, and averaging these differences across bins.

The “unsupervised” propensity scoring approaches introduced here stress graphical methods for detection of Local Treatment Differences (LTDs) using Nearest Neighbor (NN) Clustering in the X-space of observed patient baseline characteristics. Obenchain(2006a) argues that cluster membership can provide better (less “coarse”) patient matchings than propensity scores. This general approach is definitely not new. Obenchain(1979) used the underlying concept to develop an AT&T measurement plan (PRRAP) that made “customer trouble report rate” comparisons among only relatively well-matched Bell System plant maintenance districts.

## 3. The Fundamental “Balancing” Theorem of Propensity Scoring

Our basic notation for variables will be...

- $y$  = observed outcome variable(s)
- $x$  = observed baseline patient characteristic(s) / covariate(s) / instrumental variable(s)
- $t$  = observed treatment assignment (binary, 0 or 1; may be non-random)
- $z$  = unobserved explanatory variable(s)

Note that unobserved  $z$  variables may provide unknown, causal effects on outcomes,  $y$ . In statistical/econometric models, it is the existence of  $z$  variables (as well as uncertainty in measuring  $y$  and  $x$  variables) that necessitates inclusion of “error terms” in models.

NOTE: Patient level genome information is mostly a gigantic  $z$ -vector today (2006.) Some day

soon, more and more of this sort of information will become routine  $x$  variables.

The Propensity Score for the  $i^{\text{th}}$  patient is then defined to be the following function of his/her  $x$  characteristics...

$$\text{PS} = p(x_i) = \Pr( t_i = 1 \mid x_i ) = E( t_i \mid x_i ). \quad [1]$$

By definition, propensity scores are (conditional) probabilities, which are numbers between zero and one, inclusive. In words, the true propensity score of a patient is the conditional probability that he/she will receive treatment number one given his/her vector of observed, baseline characteristics,  $x$ .

In many practical applications, only the rank orders of (estimated) propensity scores are needed. In this sense, any monotone transformation of a set of propensity scores are another set of propensity scores “equivalent” to the first set. For example, when only one (univariate)  $x$  variable is found to be predictive of treatment choice/assignment, that single  $x$  variable may (itself) be called a propensity score.

One requirement of the above PS formulation is that each patient receives one and only one treatment. In other words, the formulation here does not cover chronic conditions where a cross-over design may access two (or more) treatments using sequential treatment periods for each patient ...separated by adequate “wash-out” periods.

In the following (cartoon) proof of the “fundamental theorem” of propensity scoring, it will be essential to view a patient’s propensity score as the conditional probability, [1]. We will not attempt any sort of a full-blown proof in the sense that our notation only makes sense when  $x$  is discrete and  $t$  has only 2 levels. In other words, we will not worry here about notational details for cases where components of  $x$  have continuous distributions or  $t$  has more than 2 levels.

The “fundamental theorem” of propensity scoring, Rosenbaum and Rubin(1984), states that, conditional upon a given value of the propensity score,  $p(x)$ , the distribution of baseline patient characteristics must be statistically independent of treatment choice. Mathematically, this theorem simply implies that the joint distribution of  $x$  and  $t$  given  $p$  must factor as follows:

$$\begin{aligned} \Pr( x, t \mid p ) &= \Pr( x \mid p ) \Pr( t \mid x, p ) \\ &= \Pr( x \mid p ) \Pr( t \mid x ) \\ &= \Pr( x \mid p ) \text{ times } p \text{ or } (1-p) \\ &= \Pr( x \mid p ) \Pr( t \mid p ) \end{aligned} \quad [2]$$

PROOF: In the first line of [2], we use the very definition of conditional probability to factor the joint conditional distribution of  $x$  and  $t$ . In the second line, we note that  $p(x)$  cannot contain any information not contained in  $x$  itself. In the third line, we note that  $\Pr( t \mid x )$  is either  $p(x)$  or  $1 - p(x)$ . In the fourth and final line, we conclude that  $\Pr( t \mid x )$  must depend upon  $x$  only through  $p(x)$ .

In other words,  $x$  and  $t$  are, necessarily, conditionally independent variables given the propensity score,  $p = \Pr( t = 1 | x )$ . This is really a very simple theorem in statistics / probability that requires only rather weak assumptions. In fact, the real “problem” in applications is simply that the functional form of the true PS is usually unknown and, thus, needs to be estimated from data!

When the conditional distributions of baseline patient characteristics and treatment choices “fail to factor” as dictated by the fundamental theorem of PS, [2], this is rightfully interpreted as clear evidence that one’s estimates of the unknown, true PSs are not even approximately correct.

#### 4. Case Study Example: Effects of Abciximab use on both Survival and Cardiac Billing.

The (numerical and graphical) output illustrations provided here use the data from Kereiakes et al. (2000). The corresponding command file and data are distributed along with the UPS and SPS “R” functions in the files ABCIXini.R and ABCIX.CSV. In this prospective study, outcomes variables (survival and cardiac related costs) were collected via follow-up for at least 6 months on 996 PCI patients treated at the Lindner Center, Christ Hospital, Cincinnati. Rather than randomize patients to treatment, the Lindner interventionists practiced “evidence based medicine” in choosing between either augmenting or not augmenting their “usual care” for Percutaneous Coronary Intervention (PCI) with abciximab (Reopro®), a relatively expensive IIb/IIIa cascade inhibitor. Ability-to-pay was not a factor in this treatment choice in the sense that Lindner interventionists had access to “research use” abciximab.

Our objective in this “R user’s manual” documentation for the UPS and SPS functions is not to fully discuss and illustrate all aspects of the abciximab case study. Rather, we simply wish to illustrate some example UPS and SPS function invocations as well as the tabular and/or graphical output that results. Readers interested in reading more about UPS and SPS analyses using the abciximab case study are referred to my “white paper,” Obenchain(2006a).

### Variables in the Kereiakes et al (2000) Abciximab / Lindner Data:

Description	Name	Values
Life Years Preserved = 0 if died within 6 Months or 11.6 Years given Survival for at least 6 Months	lifepres	Either 0 or 11.6 Years
Total Cardiac Related Billing within 12 Months of Patient’s Initial PCI at Lindner Center	cardbill	\$2,216 to \$178,534 in 1997 US Dollars
Was “Usual PCI Care” augmented with Abciximab?	abcix	0 => No, 1 => Yes
Was a Stent (anti-collapse device) Deployed?	stent	0 => No, 1 => Yes
Patient Height in Centimeters	height	108 cm to 196 cm
Patient Gender	female	0 => No, 1 => Yes
Was the patient Diabetic?	diabetic	0 => No, 1 => Yes
Had patient suffered an Acute Myocardial Infarction within the Last Seven Days?	acutemi	0 => No, 1 => Yes
Left Ventricular Ejection Fraction	ejecfrac	0% to 90%
Number of Vessels involved in first PCI procedure	veslproc	0 to 5

## 5. Installing and Loading the “USPS” Package.

First, obtain the USPS\_1\_0.ZIP archive from the web site of the Central Indiana ASA chapter, [www.math.iupui.edu/~indyasa/bobodown.htm](http://www.math.iupui.edu/~indyasa/bobodown.htm) , or from CRAN. Then use the R ‘Packages’ menu and the “Install from local ZIP file” menu item. Then use the “Load packages” sub-menu to load “USPS.” The required “cluster”, “lattice” and “spline” R packages will then be automatically loaded for you.

All of the abciximab analyses and graphics used as illustrations in this documentation (and several more!) can then be generated by reading in the abcix.R source code file and executing the resulting abcix() function.

```
"abcix" <- function() {  
  
  #input the abciximab study data of Kereiakes et al. (2000).  
  Load(lindner)  
  
  # outcomes: lifepres & cardbill  
  # Define Divisive Cluster Hierarchy for UNSUPERVISED analyses...  
  UPSxvars <- c("stent", "height", "female", "diabetic", "acutemi",  
               "ejefrac", "veslproc")  
  UPSsharch <- UPSshclus(lindner, UPSxvars, method="diana")  
  plot(UPSsharch) # Top figure on page 13.  
  
  # Save UPSpars settings for NN/IV analyses of the "lifepres" outcome.  
  # Although the "lifepres" variable assumes only two different numerical values, it is not to be treated here as  
  # a “factor.” It’s average value is to be interpreted as “proportion surviving” times 11.6 = expected years.  
  UPSaccum(UPSsharch, lindner, abcix, lifepres, faclev=1, accobj="ABClife")  
  
  lif001nn <- UPSnnltd( 1)  
  lif002nn <- UPSnnltd( 2)  
  lif005nn <- UPSnnltd( 5)  
  lif010nn <- UPSnnltd( 10)  
  lif020nn <- UPSnnltd( 20)  
  plot(lif020nn) # graphical display  
  lif030nn <- UPSnnltd( 30)  
  lif040nn <- UPSnnltd( 40)  
  lif050nn <- UPSnnltd( 50)  
  lif060nn <- UPSnnltd( 60)  
  summary(lif060nn) # brief console output  
  lif070nn <- UPSnnltd( 70)  
  lif080nn <- UPSnnltd( 80)  
  lif090nn <- UPSnnltd( 90)  
  plot(lif090nn) # graphical display  
  
  lif030iv <- UPSivadj( 30)  
  lif040iv <- UPSivadj( 40)  
  lif050iv <- UPSivadj( 50)  
  lif060iv <- UPSivadj( 60)  
  lif070iv <- UPSivadj( 70)  
  lif080iv <- UPSivadj( 80)  
  lif090iv <- UPSivadj( 90)  
  plot(lif090iv) # graphical display
```

```

lif100iv <- UPSivadj(100)
lif200iv <- UPSivadj(200)
lif300iv <- UPSivadj(300)
lif996iv <- UPSivadj(996)

# Overall "Sensitivity Analysis" Summary...
UPSgraph()

cat("\n\nPress ENTER to add low-precision IV estimates...\n\n")
scan()
lif003iv <- UPSivadj( 3)
lif005iv <- UPSivadj( 5)
lif010iv <- UPSivadj( 10)
lif020iv <- UPSivadj( 20)
# Display Augmented "Sensitivity Analysis" Summary...
UPSgraph()

# Display contents of UPSdf...
ABClife

cat("\n\nPress ENTER to Analyze Costs...\n\n")
scan()

# Save UPSpars settings for NN/IV analyses of the "cardbill" outcome
UPSaccum(UPSharch, lindner, abcix, cardbill, accobj="ABCcost")

cst001nn <- UPSnnltd( 1)
cst002nn <- UPSnnltd( 2)
cst005nn <- UPSnnltd( 5)
cst010nn <- UPSnnltd( 10)
cst020nn <- UPSnnltd( 20)
cst030nn <- UPSnnltd( 30)
plot(cst030nn) # graphical display (Figure on page 15.)

cst040nn <- UPSnnltd( 40)
cst050nn <- UPSnnltd( 50)
cst060nn <- UPSnnltd( 60)
cst070nn <- UPSnnltd( 70)
cst080nn <- UPSnnltd( 80)
cst090nn <- UPSnnltd( 90)
plot(cst090nn) # graphical display (Figure on page 16.)

cst030iv <- UPSivadj( 30) # plot cst030iv is top figure on page 18.
cst040iv <- UPSivadj( 40)
cst050iv <- UPSivadj( 50)
cst060iv <- UPSivadj( 60)
cst070iv <- UPSivadj( 70)
cst080iv <- UPSivadj( 80)
cst090iv <- UPSivadj( 90)
plot(cst090iv) # graphical display (Bottom figure on page 18.)
cst100iv <- UPSivadj(100)
cst200iv <- UPSivadj(200)
cst300iv <- UPSivadj(300)
cst996iv <- UPSivadj(996)

```

```

# Overall "Sensitivity Analysis" Summary (Figure on page 19)...
UPSgraph()

cat("\n\nPress ENTER to add low-precision IV estimates...\n\n")
scan()
cst003iv <- UPSivadj( 3)
cst005iv <- UPSivadj( 5)
cst010iv <- UPSivadj( 10)
cst020iv <- UPSivadj( 20)
# Augmented "Sensitivity Analysis" Summary (Figure on page 20)...
UPSgraph()

# Display contents of UPSdf...
ABCcost

# End of UNSUPERVISED analyses.

# Define Logit Model for Treatment Choice in SUPERVISED analyses
PStreat <- abcix~stent+height+female+diabetic+acutemi+ejecfrac+veslproc

# Store Propensity Score info (default=5 bins) in frame named "lindSPS"
logtSPS <- SPSlogit(lindner, PStreat, PSfit, PSrnk, PSbin,
                    appn="lindSPS")
logtSPS

# Testing for Within-Bin Balance on Continuous Covariates...
SPSbalht <- SPSbalan(lindSPS, abcix, PSbin, height)
plot(SPSbalht)
print(SPSbalht)
SPSbalej <- SPSbalan(lindSPS, abcix, PSbin, ejecfrac)
plot(SPSbalej) # Figure on page 25.
print(SPSbalej)
SPSbalvs <- SPSbalan(lindSPS, abcix, PSbin, veslproc)
plot(SPSbalvs)
print(SPSbalvs)

# Testing for Within-Bin Balance on Binary (Dichotomous) Covariates...
SPSbalst <- SPSbalan(lindSPS, abcix, PSbin, stent)
print(SPSbalst)
SPSbalfm <- SPSbalan(lindSPS, abcix, PSbin, female)
print(SPSbalfm)
SPSbaldi <- SPSbalan(lindSPS, abcix, PSbin, diabetic)
print(SPSbaldi)
SPSbalam <- SPSbalan(lindSPS, abcix, PSbin, acutemi)
print(SPSbalam)

# Test for Within-Bin Outcome Differences...
lindlife <- SPSoutco(lindSPS, abcix, PSbin, lifepres, faclev=1)
plot(lindlife)
print(lindlife)
scan()

lindcost <- SPSoutco(lindSPS, abcix, PSbin, cardbill)
plot(lindcost) # Both Figures on page 28.
print(lindcost)

```

```

# Cubic smoothing spline analyses...
SPScbill <- SPSsmoot(lindSPS, abcix, PSfit, cardbill)
plot(SPScbill)
SPScbil7 <- SPSsmoot(lindSPS, abcix, PSfit, cardbill, df=7)
plot(SPScbil7) # Figures on pages 33 and 34.
SPScbil3 <- SPSsmoot(lindSPS, abcix, PSfit, cardbill, df=3)
plot(SPScbil3)

# Loess "symmetric" smoothing analyses...
SPScblss <- SPSloess(lindSPS, abcix, PSfit, cardbill)
plot(SPScblss)
SPScbls5 <- SPSloess(lindSPS, abcix, PSfit, cardbill, span=.5)
plot(SPScbls5) # Top Figure on page 33.
SPScbls9 <- SPSloess(lindSPS, abcix, PSfit, cardbill, span=.9)
plot(SPScbls9)
}

```

## Part I: Unsupervised PS Analyses

In data mining terminology, the approach here is an “unsupervised” form of propensity score binning, which (in turn) is a form of retrospective patient sub-classification or stratification (matching without using any fixed ratio of treated-to-untreated patients.) Specifically, we will use observed patient X-characteristics to cluster similar patients together rather than to directly predict (non-random) treatment choice. In fact, here our unsupervised (cluster-binning) strategy will typically use...

- many more bins (say, 50 instead of 5) and/or
- bins with a potentially wide range of different sizes (wide variation in total numbers of patients.)

### 7. Unsupervised Step One: Compute the Hierarchical Clustering Tree using `UPShclus()`

The UPS hierarchical clustering function, `UPShclus()`, must be invoked before any of the four other UPS functions. The R calling syntax for this function is

```
UPShclus(dframe, xvars, method="diana"),
```

where the first two arguments are required and the third is optional (because a default value is provided.) The R invocation of `UPShclus()` used in the abciximab case study is

```
UPSharch <- UPShclus(lindner, UPSxvars)
```

...which implies that the default (divisive) clustering method = "diana" will be used and that all relevant hierarchical clustering information is to be saved in an R object of class "UPShclus" named "UPSharch."

**The three arguments of UPShclus() are as follows:**

The first argument, `dframe = lindner`, must be the name of an existing data.frame object.

SPSlogit's second argument, `xvars = UPSxvars`, must be a list (concatenation) of "quoted" names (in character format) of variables existing within the data.frame that will be used to discover clusters representing "nearest neighbors in X-space."

```
UPSxvars <- c("stent","height","female","diabetic","acutemi","ejefrac","ves1proc")
```

The final (optional) argument defines the clustering algorithm to be used by `UPShclus()`. The (default) method is "diana" for the DIVisive ANALysis method of Kaufman and Rousseeuw (1990), and the other two options currently available are: "agnes" for the Agglomerative NESTing method of Kaufman and Rousseeuw(1990) and the original "hclus" method implemented in Fortran code contributed to STATLIB by F. Murtagh.

Unfortunately, the current implementation of `UPShclus` provides only one possible metric, Mahalanobis distance, Rubin(1980), for determining patient dissimilarity in X-space. Thus each X variable used with `UPShclus()` needs to be coded as either a continuous (interval) or a "dummy" (0 or 1) variable. Factors with more than two unordered (but numerical levels) would be inappropriately analyzed by the current `UPShclus()` implementation.

The object list returned by `UPShclus()` will contain an object (`x$upshcl`) of class `diana`, `agnes` or `hclus` and must be saved to some name because this `UPShclus()` "output list" must be specified as the first, required argument to the `UPSaccum()` function.

The graphical output generated for an `UPShclus()` output object is a "dendrogram" display of the resulting hierarchical clustering tree, such as the following "diana" tree for the `abciximab` data.



## 8. Unsupervised Step Two: Specify Treatment and Outcome variables to prepare for Accumulation of NN and/or IV information with `UPSaccum()`

Invoking `UPSaccum()` creates an R object named “UPSpars” or overwrites any existing object with this name.

```
UPSaccum(hiclus, dframe, trtm, yvar, faclev=3, accobj="UPSframe")
```

The six possible arguments of `UPSaccum()` are as follows:

`UPSaccum`'s first argument, `hiclus=ABChclus`, must be an R object of class `diana`, `agnes` or `hclust` ...usually the object returned by an invocation of `UPShclus()`.

`UPSaccum`'s second argument, `dframe=lindner`, is usually the `data.frame` that provided X-covariates to the most recent `UPShclus()` invocation.

`UPSaccum`'s third and fourth arguments, `trtm=abcix` and `yvar=cardbill`, are almost always the treatment and outcome variable names within the `data.frame` that also provided the X-covariates in the most recent `UPShclus()` invocation.

The two optional arguments to `UPSaccum()` are

`faclev` = the maximum number of distinct numerical values that an outcome variable can assume within the input `data.frame` and yet still be treated as a discrete R “factor” variable. The default values for this optional parameter is `faclev=3`.

`accobj` = the quoted name of the R object that will hold "UPSframe"

When `yvar` takes on more than `faclev` distinct numerical values within the specified `data.frame`, `yvar` will be considered to be a “continuous” variable.

## 9. Unsupervised Step Three: Use `UPSnnltd()` to Compute Nearest Neighbor / Local Treatment Differences (NN/LTD) for a specified Number of Cluster-Bins.

In these “Step 3” analyses, we calculate Nearest-Neighbor (within cluster-bin) statistics describing “Local Treatment Differences” via R function calls of the form:

```
out###nn <- UPSnnltd(numclust)
```

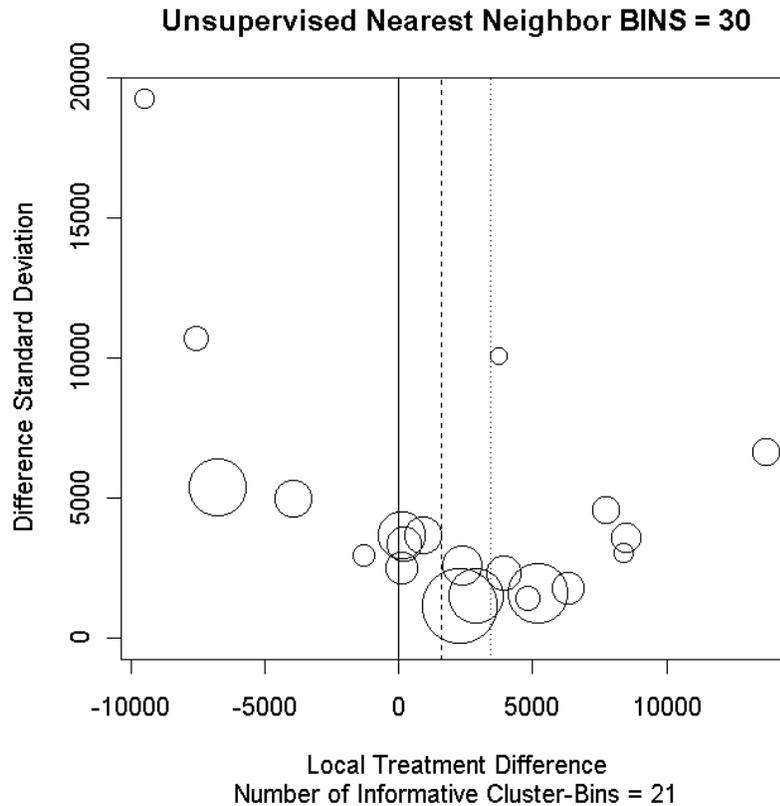
Here are some example invocations of the `UPSnnltd()` family of functions:

```
cst030nn <- UPSnnltd(30)
cst030nn  # shorthand for... print(cst030nn)
```

```
summary(cst030nn)
plot(cst030nn)
```

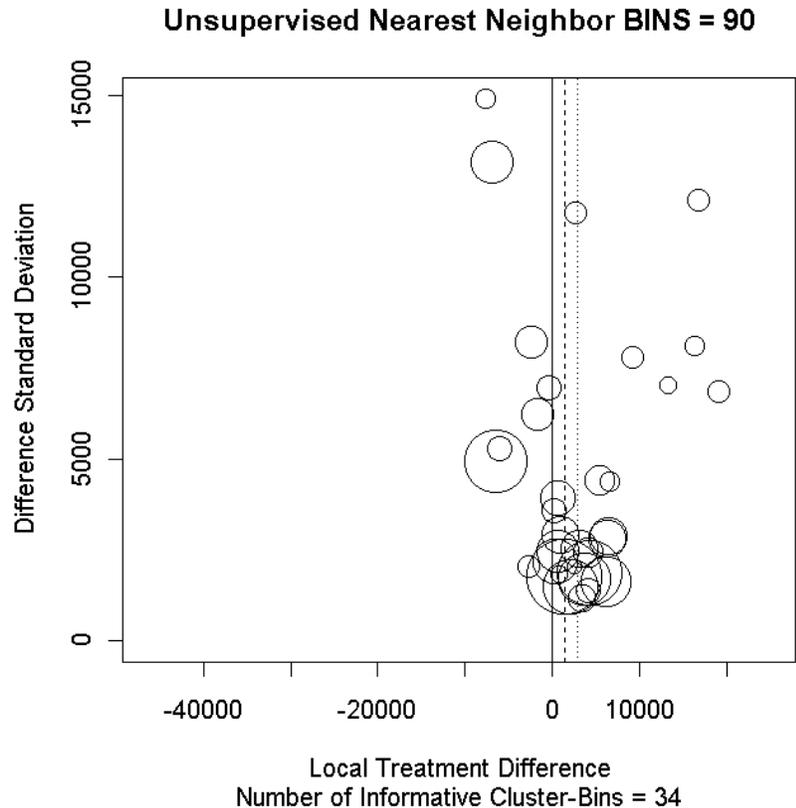
The graphical output generated by “plot.UPSnnltd()” is called a “Nearest Neighbors: Local Treatment Differences” (NN/LTD) plot. This graphic displays a circular symbol representing each “informative” cluster-bin ...using horizontal and vertical coordinates that convey within-cluster-bin mean and precision information about local treatment differences. Specifically, the horizontal coordinate conveys the average outcome difference (treated minus untreated) within a single cluster-bin, the vertical coordinate conveys the corresponding outcome difference standard error (treated minus untreated) within that same cluster-bin, and the area of the circular plotting symbol denotes the total number of patients (either treated or untreated) within that cluster-bin. To be fully “informative,” a cluster-bin must contain at least two patients on each treatment. After all, at least 2 patients on each treatment are needed to provide not only a treatment difference point estimate but also an observed standard error for that treatment difference!

Here is the NN/LTD plot for 30 “diana” cluster-bins for the `cardbill` outcome within the `abciximab` case study. Note that only 21 of these 30 clusters are fully informative about local treatment differences.



For comparison with the above NN/LTD plot based upon 30 cluster-bins, here is the corresponding PS plot for 90 diana cluster-bins. Note that only 34 of these 90 cluster-bins are

fully “informative” about treatment differences. Twenty additional cluster-bins out of the original 90 contained just one patient on one (or both) of the two treatments and thus provided no “local” standard error information to supplement its local treatment difference point estimate.



Note that each `UPSnnltd()` plot also displays 3 vertical lines. A solid vertical line is always drawn at the position denoting an outcome treatment difference of zero. A dashed vertical line is then drawn at the position corresponding to the overall weighted average outcome treatment difference, with weights proportional to the total number of patients within each cluster-bin. Finally, a dotted vertical line is drawn at the position corresponding to the supposedly “optimally” weighted average outcome treatment difference, with weights inversely proportional to estimated within-cluster-bin variance ...as dictated by the Gauss-Markov theorem for the known variance case.

**10. Unsupervised Step Four: Use `UPSivadj()` to Compute Instrumental Variable / Local Outcome Averages (IV/LOA) for a specified Number of Cluster-Bins**

In this step, we generate “IV plots” of within-cluster Local-Outcome-Averages (regardless of treatment) versus within-cluster treatment percentages (PS estimates.) A key assumption here is that all X-variables used to define clusters are instrumental variables in the sense that they influence expected outcome only through treatment choice. I.E. none of the X-variables used to define clusters should have direct influence on the outcome to be analyzed. The appropriate R function calls are then of the form:

`UPSivadj(numclust)`

Any two IV cluster-bins can be connected by a straight line without any apparent lack-of-fit, and there is no way to define such a line with a single cluster. Thus `numclust` must be at least 3 in each `UPSivadj()` invocation.

Possible invocations of the `UPSivadj()` family of functions for the abciximab case study are of the form:

```
lindcost <- UPSivadj(90)
lindcost # or print(lindcost)
summary(lindcost)
plot(lindcost)
```

**WARNING:** Several of the X-variables used to define clusters in the abciximab study are likely to have direct effects on survival and/or cost as well as (demonstrated) influence upon treatment choice. In other words, these X-variables are **highly unlikely** to be pure “instruments.”

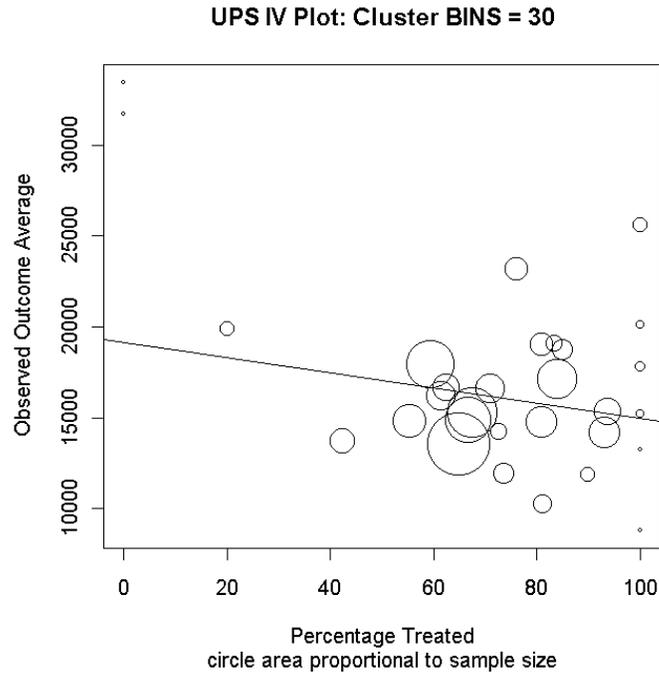
The graphical output that can be requested from a `UPSivadj()` output object is called an “IV plot” ...as in Figure 1 of McClellan, McNeil and Newhouse (1994.) This graphic displays a circular plotting symbol representing each cluster-bin using horizontal and vertical coordinates that convey how within-cluster-bin outcome averages vary across clusters. Specifically, the horizontal coordinate for each cluster displays the within-cluster-bin PS estimate (fraction of treated patients relative to total patients), the vertical coordinate conveys the corresponding average outcome (regardless of treatment) within that same cluster-bin, and the area of the circular plotting symbol depicts the relative size of the total number of patients (treated plus untreated) within that cluster-bin.

The first plot on page 18 displays calculations from `UPSivadj()` for 30 diana cluster-bins on the `cardbill` outcome in the abciximab case study. Note that “pure” cluster-bins (with extreme within-bin PS estimates of either 0% or 100%) are not only informative in this type of analysis ...they are potential “high-leverage” points!

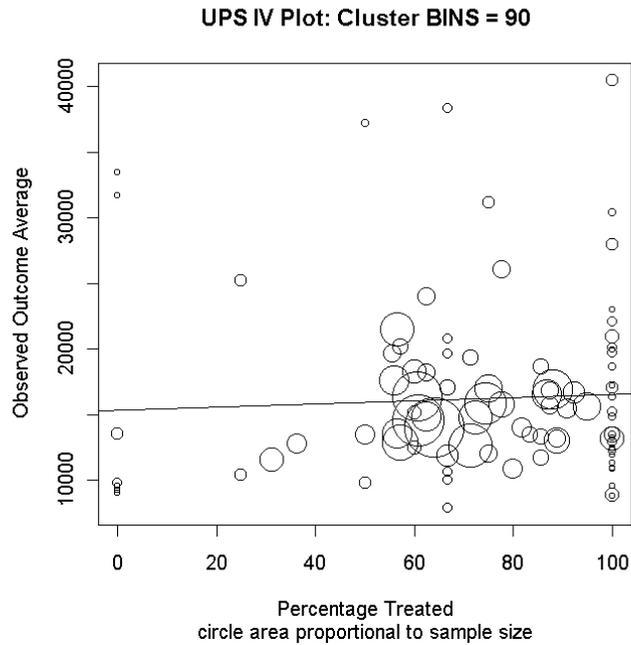
In sharp contrast with “NN/LTD plot” analyses, a relatively much larger number of cluster-bins may be highly desirable in the “IV plotting” approach. For example, the first IV plot below with “only” 30 cluster-bins suggests that treatment with abciximab might be expected to result in a net cost savings! On the other hand, the second IV plot on page 18, with 90 cluster-bins, suggest an average overall treatment effect much more like estimates from NN/LTD plots with a relatively wide range of numbers of cluster-bins (either small or large.)

Again, the fundamental problem with the abciximab examples of IV/LOA analyses may well be that some of the X covariates used here are not “instrumental variables.” Encouragingly, the `UPSgraph()` “systematic sensitivity summary” plot will dramatically depict the instability in numerical signs of potential answers resulting from alternative IV analyses for this example!

**Is the cardbill cost difference (abcix minus usual care) negative?**



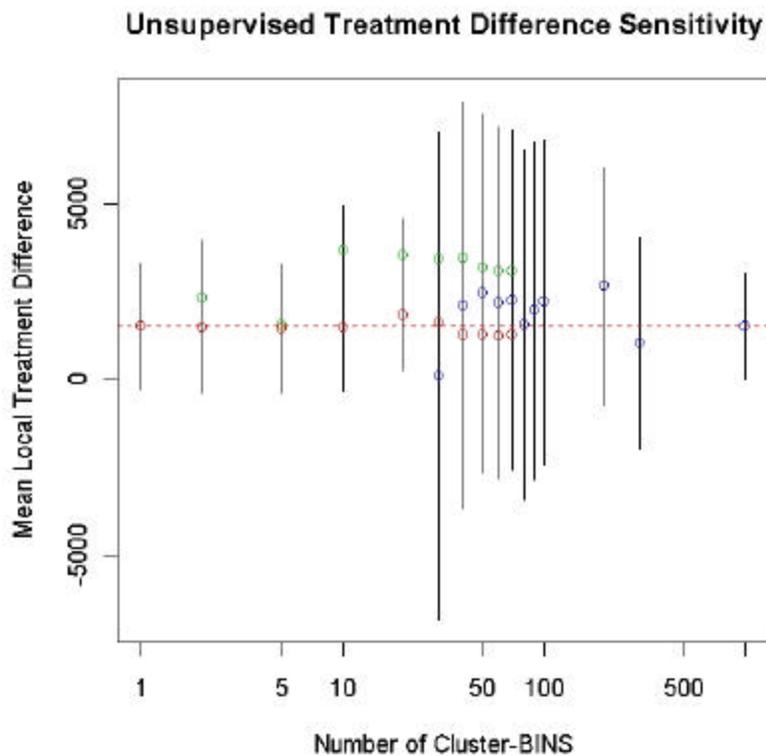
**Or is this cost difference (abcix minus usual care) positive?**



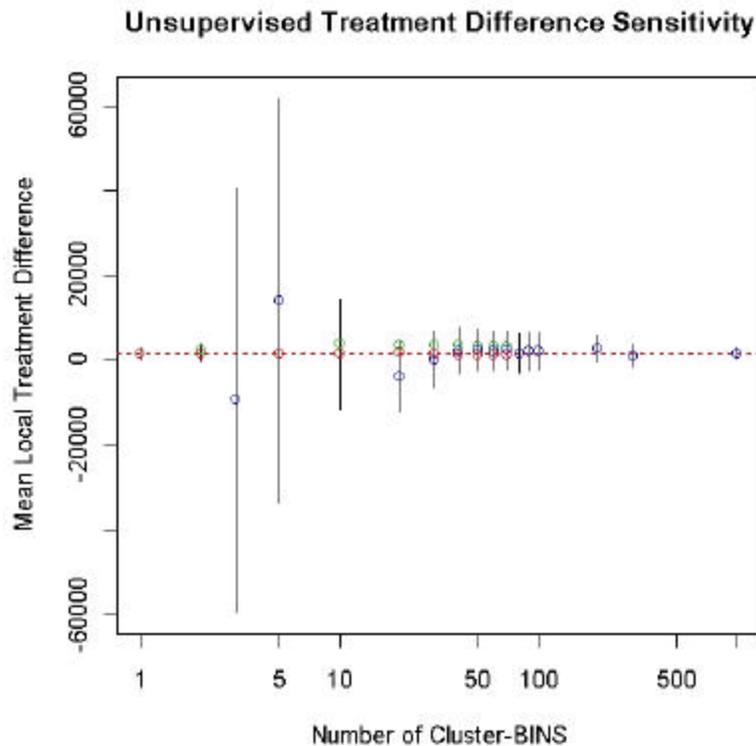
## 11. Unsupervised Step Five: Use `UPSgraph()` to Graphically Display a Systematic Sensitivity Summary of NN and/or IV Analyses to choice of Number of Cluster-Bins

While the user of the UPS functions certainly has the option to make a NN/LTD or IV/LOA plot immediately after each call to `UPSnnltd()` or `UPSivadj()` for a specified number of cluster-bins, the fundamental strategy advocated here is to generate results over a sufficiently wide range for number-of-clusters to generate a meaningful “systematic sensitivity summary” plot resulting from an invocation of `UPSgraph()`. For example, it is always a good idea to invoke both `UPSnnltd(1)` and `UPSivadj(Cmax)`, where `Cmax` = maximum possible number of clusters = total number of subjects in the input R data.frame. After all, these seemingly “extreme” cases are actually guaranteed to give the same (“unadjusted”) estimates of the overall average outcome difference ...and of its precision!

Here is what the `UPSgraph()` summary for the `cardbill` outcome in the `abciximab` case study looks like before (very low precision) `UPSivadj()` results have been accumulated for fewer than 30 cluster-bins.



Calculating `UPSivadj()` results for 20, 10, 5 and 3 cluster-bins forces the `UPSgraph()` summary plot for the `cardbill` outcome in the `abciximab` / `lindner` case study to “Zoom Out” so as to accommodate the much wider (vertical) range of treatment difference uncertainties displayed in the updated `UPSgraph()` below.



**12. Unsupervised Step Six: Use `UPSaltdd()` to Compare the NN/LTD Distribution for a given number of Clusters in X-space with the Corresponding “Artificial” Distribution from Random Clusters.**

After displaying outcome LTD sensitivity over a range of alternative numbers-of-clusters using `UPSgraph()`, the user will usually wish to focus on visualizing details of the NN/LTD distribution for some specific number-of-clusters. The `UPSaltdd()` function displays Cumulative Distribution Functions (CDFs) and Histogram pairs that are quite helpful in doing this.

`UPSnnltd()` makes only the “more” or “most” relevant patient comparisons in the specified X-space for a given requested number-of-clusters. `UPSaltdd()` allows users to compare and contrast this NN/LTD distribution with the corresponding “Artificial” LTD distribution from random clusterings (ignoring the specified X variables.) This allows the user to literally see how much treatment-selection-bias (imbalance) has been “detected” using the specified X-space clustering metric, clustering algorithm and number-of-clusters. If the NN/LTD and Artificial LTD distributions are not clearly “different,” then no meaningful “adjustment” for X-variables has occurred.

For example, consider the following R code fragment for the abciximab / lindner example. Here, we specify `faclev=1` because the expected life years preserved (continuous) variable, **lifepres**, assumes only two distinct values (0 if died within 6 months, or 11.6 years otherwise):

```
abcdf <- UPSaltddd(lindner, abcix, lifepres, faclev=1, NNobj=lif050nn)
```

The R command `print(abcdf)` then produces the following output:

```
UPSaltddd Object: Artificial Distribution of LTDs for random clusters...
Data Frame: lindner
Outcome Variable: lifepres
Treatment Factor: abcix
Scedasticity assumption: homo
Number of Replications: 10
Number of Clusters per Replication: 50
Total Number of Informative Clusters = 494
```

```
Mean Artificial Treatment Difference = 0.4275955
Number of Smoothed Sample Quantiles = 309
Mean of Smoothed Sample Quantiles = 0.4251765
Std. Deviation of Sample Quantiles = 1.254298
```

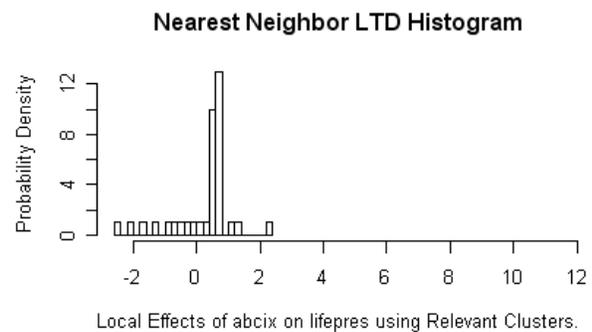
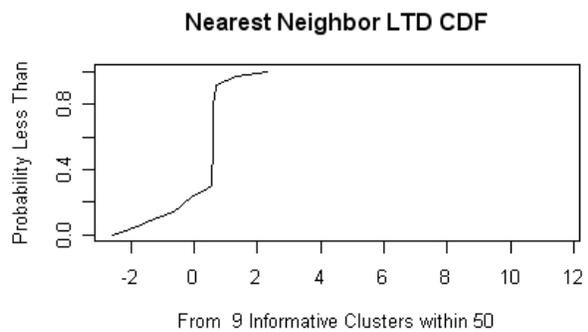
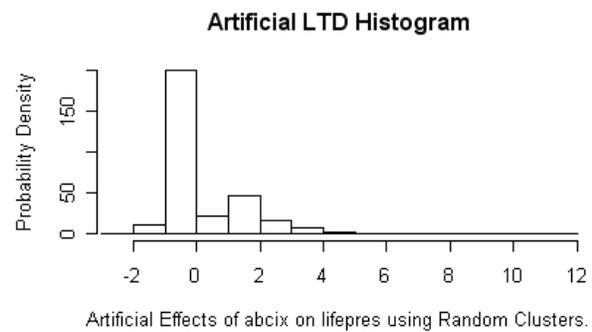
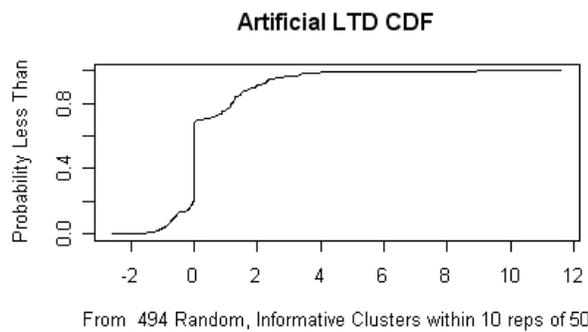
```
UPSnnltd Object: lif050nn
Number of Informative Clusters = 9
```

```
Mean of Observed LTD Distribution = 0.3592107
Number of Smoothed Sample Quantiles = 37
Mean of Smoothed Sample Quantiles = 0.2331547
Std. Deviation of Sample Quantiles = 0.9594896
```

Unfortunately, the above printout is rather misleading. As we will see, the two distributions to be compared are skewed and have different ranges. Thus the order of their mean values is relatively meaningless because means are poor measures of location in this situation. We do learn that the NN/LTD distribution has a slightly lower mean (and slightly higher precision) than the Artificial LTD distribution. Much more informative and meaningful insights are provided by the R `plot(abcdf, breaks=20)` command that produces the 4 plots on the next page.

These plots allow the user to literally “see” that the NN/LTD and Artificial LTD distributions are quite different and also skewed in opposite directions. The Artificial LTD CDF and Histogram (in the top 2 plots) show that this distribution has **Mode < Median = 0** < Mean = 0.43 years and contains a large, outlying value of +11.6 years (a random cluster where all abciximab augmented patients survived for at least 6 months and all unaugmented patients died.) Meanwhile, the NN/LTD CDF and Histogram (on the bottom of the figure) show that this distribution has Mean = 0.36 years < **Median = 0.59 years** < **Mode** with an upper limit of ~2.5 years. In other words, relative to the (biased) Artificial LTD distribution, the LTD distribution resulting from NN “adjustment” using 50 clusters of relatively well-matched patients in the X-space defined by seven patient characteristics (**stent, height, female, diabetic, acutemi, ejecfrac & veslproc**) strikes me as being **clearly more favorable** to demonstrating the “effectiveness” of augmenting “usual PCI care” with abciximab.

Again, means are not “robust” measures of location; the means of these two distributions are in the “wrong order” simply due to a single outlier in the simulated Artificial LTD distribution!



## Part II: Supervised PS Analyses

### 13. Supervised Step One: Estimate Propensity Scores and Form Bins

The propensity scoring “definition” function, `SPSlogit()`, provides...

- (1) estimates a logistic regression model for predicting observed treatment choice (zero or one) from specified covariates observed on all patients,
- (2) estimates the probability (propensity score) that each patient would have been selected to receive treatment number one (rather than treatment number zero),
- (3) rank orders all patients (allowing for ties) according to these estimated probabilities, and
- (4) groups patients into “bins” defined by equal ranges of PS ranks.

NOTE: Defining bins using tied rank ranges assures that no two patients with identical estimated propensity scores will end up in different bins. The bin number for the  $i^{\text{th}}$  patient will be  $1 + \text{floor}(\text{bins} * \text{rank}[i] / (1 + \text{total number of patients}))$ .

The R calling syntax for this “define” function is

```
SPSlogit( dframe, form, psfit, psrnk, qbin, bins=5, appn="" )
```

SPSlogit's first five arguments are required while the last two are optional. The R invocation of SPSlogit() illustrated in the numerical example is

```
SPSobj <- SPSlogit(PCI, PSform, PSFIT, PSRNK, QUINT)
```

The arguments of SPSlogit() are as follows:

SPSlogit's first argument, dframe=PCI, must be an existing data.frame name.

SPSlogit's second argument, form=PSform, must be an S "formula" for predicting the treatment factor (ABCIX) using available data.frame covariates.

```
PSform <- ABCIX ~ STENT + HEIGHT + FEMALE + DIABETIC +  
ACUTEMI + EJEFCFRAC + VES1PROC
```

SPSlogit's third, fourth and fifth arguments, psfit = PSFIT, psrnk = PSRNK and qbin = QUINT, supply names for the fitted propensity scores (estimated treatment probabilities), the propensity score ranks and the propensity score bin number factor that are created by SPSlogit.

Note 1: Although missing values ("NA"s) are allowed, it is best to make predictions for as many patients as practically possible. Adding a covariate with NAs for patients who do not have NAs in any other current covariate or in the response (treatment) variable, will cause additional NAs in the psfit, psrnk and qbin variables created by SPSlogit().

Note 2: SPSlogit() does not use regression subset selection methods because this tactic can create "ties" between propensity score predictions. Ties can cause the total number of patients per bin to vary from bin to bin. Also, see comments in the section "R and S-Plus functions that are (or could be) called in PS analyses."

The optional arguments to SPSlogit() are

bins = the number of bins to be formed, and  
appn = name of output data.frame with appended columns, as a quoted string.

The default value for bins is 5. Thus, users need to specify a value for "bin" only when they wish SPSlogit() to form more than 5 or fewer than 5 bins.

The default value for appn is "", which means that the user wishes to revise the input data.frame by appending the calculated psfit, psrnk and qbin variables to it. Thus, the user needs to specify a non-empty quoted string (such as appn="SPSdf2") if he/she

does not wish to modify / overwrite the input data.frame named in the first argument to `SPSlogit()`. For example, the treatment indicator variable will have been declared to be a “factor” in the revised data.frame.

The output list returned by `SPSlogit()` needs to be saved to avoid printing of summary() information about the “glm” object created by `SPSlogit()`. This summary information as well as details on the `SPSlogit()` invocation can then be printed later from the saved object using `print(PSObj)`.

**NOTE:** The `SPSnbins()` function creates a new (or modifies an existing) propensity score bin number factor variable to change the number of PS bins; typical R invocations are of the general form...

```
PSframe <- SPSnbins( PSframe, psrnk, octile, bins=8)
```

or

```
PSframe <- SPSnbins( PSframe, psrnk, decile, bins=10)
```

All four arguments to `SPSnbins()` are usually specified; the first three are required, and the default value for the fourth is `bins=8`. `SPSnbins()` does not re-calculate PS fitted values or re-rank patients. Specifically, the first argument to `SPSnbins()` is almost always the data.frame output value of a previous `SPSlogit()` invocation, and the second argument to `SPSnbins()` is then the `psrnk` (fourth) argument from that same `SPSlogit()` invocation. Finally, the third (required) argument of `SPSnbins()` is a variable name for a (new or already existing) propensity score bin number factor variable.

#### 14. Supervised Step Two: Test for Within-Bin “Balance” on Covariates

The “fundamental theorem” of PS states that, if an appropriate propensity scoring algorithm has been found, then there will be no difference in the distributions of covariate measurements between treatment groups with the same given propensity score. In other words, although this distribution may be different at different numerical values for propensity score, treated and untreated patients have been relatively “well matched” when their propensity scores are nearly equal. Specifically, treated and untreated patients can be expected to display essentially identical covariate distributions within each bin.

The `SPSbalan()` function is designed to detect “violations” of this fundamental PS balancing theorem, thereby implying that the current PS model is inadequate to explain treatment selection. Every covariate used in the second, “formula” argument to `SPSlogit()` of SPS Step ONE is a candidate for the sort of testing performed here in SPS Step TWO.

The R calling syntax for the function to detect treatment differences in the within-bin distribution of a single X covariate is...

```
SPStest <- SPSbalan(dframe, xvar, trtm, qbin, faclev=3)
```

The first four arguments of `SPSbalan()` are required, and the final two are optional. For example, three consecutive R invocations could be

```
SPSout1 <- SPSbalan(PSframe, AGE, ABCIX, QUINT)
SPSout2 <- SPSbalan(PSframe, EJECTFMI, ABCIX, QUINT)
SPSout3 <- SPSbalan(PSframe, VES1PROC, ABCIX, QUINT)
```

The arguments of `SPSbalan()` are as follows:

`SPSbalan`'s first argument, `dframe=PSframe`, is almost always a returned `data.frame` from `SPSlogit()`.

`SPSbalan`'s second and third arguments, `xvar` and `trtm`, are usually terms from the S "formula" used in `SPSlogit`. In fact, `trtm` must be the "response" (first) term of that formula, while `xvar` is almost always one of the "covariate" terms.

`SPSbalan`'s fourth argument, `qbin=QUINT`, is almost always the bin indicator variable name as in the previous `SPSlogit()` or `PSbinnum()` invocation.

The optional argument to `SPSbalan()` is

`faclev` = the maximum number of distinct numerical values that a variable can assume and yet still be treated as an S "factor."

The default value for this optional parameter is `faclev=3`.

If the `xvar` named in this invocation takes on no more than `faclev` distinct numerical values, no graphics will be displayed. On the other hand, when `xvar` takes on more than `faclev` distinct numerical values, `xvar` is considered "continuous," and a "SPSbalan" will be created.

`SPSbalan()` returns a list of objects containing analysis details. However, in addition to the optional box plot "side effects" discussed above, `PSdifcov()` also prints out summaries of overall and within-bin analyses of `trtm` effects on `xvar`.

When the `xvar` data contain `faclev` or fewer levels, a summary of contingency table ChiSquare tests for `trtm` effects is printed.

When the `xvar` data contain more than `faclev` levels, `xvar` is considered "continuous," and one-way and two-way ANOVA summaries are printed (`trtm %in% bin` identifies treatment within bin effects.)

If the above test results and/or box-plots indicate that the fundamental theorem of propensity scoring is not even approximately satisfied, then a revised model formula should be tried in `SPSlogit()`. Typically, one would try adding powers or interaction terms between currently

used covariates, adding more covariates, or even defining propensity scores using a generalized additive model rather than a generalized linear model.

Here are some examples of `SPSbalan()` output for the abciximab case study:

```
> SPSbalan(PSframe, age, abcix, QUINT)
```

```
Test for Raw / Unadjusted Differences by Treatment
```

```
AGE ~ ABCIX
```

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)	
ABCIX	1	0.0	0.0443	0.0003315842	0.9854754	<= NOT significant!
Residuals	1009	134822.4	133.6198			

```
Test for Treatment Differences within Paired PS Bins
```

```
AGE ~ QUINT + ABCIX %in% QUINT
```

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)	
QUINT	4	686.7	171.6847	1.295466	0.2699387	
ABCIX %in% QUINT	5	1475.8	295.1598	2.227162	0.0496073	<= significant?
Residuals	1001	132659.9	132.5274			

In other words, binning can appear to create a covariate difference!!!

```
> SPSbalan(PSframe, ves1proc, abcix, QUINT)
```

```
Test for Raw / Unadjusted Differences by Treatment
```

```
VES1PROC ~ ABCIX
```

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)	
ABCIX	1	14.3135	14.31349	34.06864	7.164788e-009	<= Significant!!!
Residuals	1009	423.9180	0.42014			

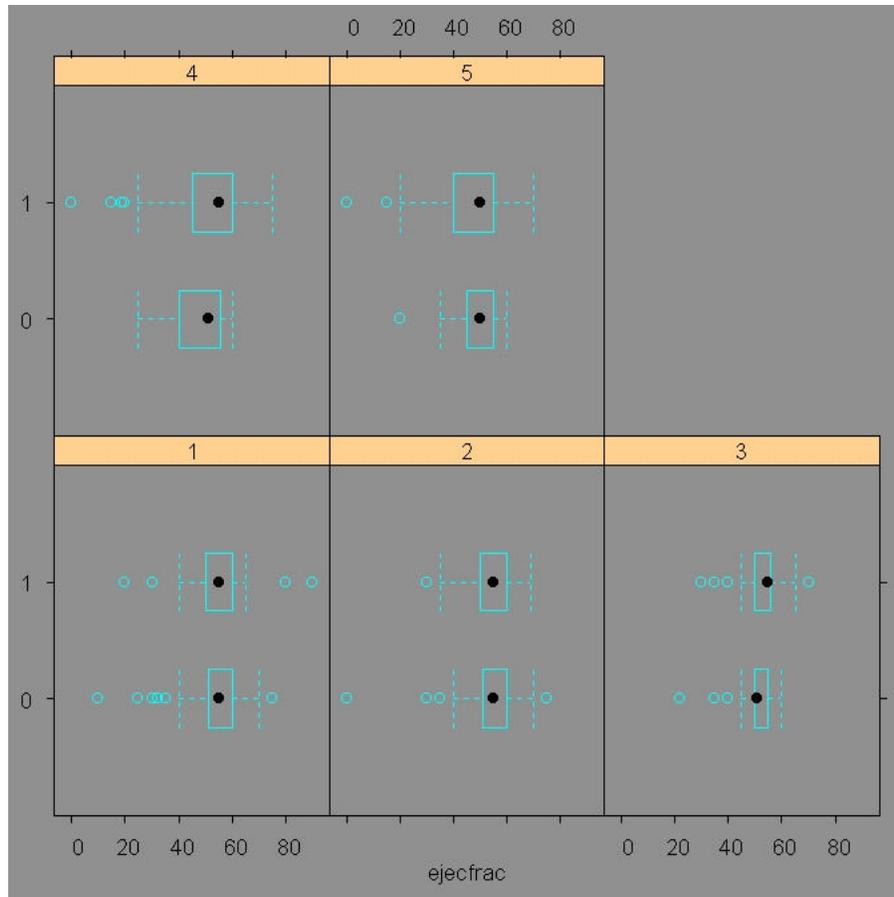
```
Test for Treatment Differences within Paired PS Bins
```

```
VES1PROC ~ QUINT + ABCIX %in% QUINT
```

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)	
QUINT	4	164.5746	41.14364	151.0037	0.0000000	
ABCIX %in% QUINT	5	0.9167	0.18333	0.6729	0.6441004	<= NO PROBLEM!!!
Residuals	1001	272.7402	0.27247			

Much more commonly, appropriate estimates of propensity scores eliminate all within-bin covariate differences!

To “visualize” within-cell balance, plot the `SPSbalan()` output for a continuous variable:



### 15. Supervised Step Three: Display Within-Bin Treatment Differences by Outcome using `SPSoutco()`

The calculations performed in Step THREE by `SPSoutco()` assume that the PS model of Step ONE was found to (approximately) satisfy the fundamental balancing theorem in Step TWO. In other words, treated and untreated patients have now been relatively “well matched” on covariates within bins. As a result, within-bin mean outcome differences (treated minus untreated) can be expected to be relatively free of bias, at least compared with the corresponding overall mean outcome difference between treatment groups.

An overall summary statistic estimating any true (treated minus untreated) outcome difference is usually desired. As a result, within-bin estimates need to be averaged across bins using some weighting scheme. `SPSoutco()` displays two such weighted averages:

- weighted proportional to the total number of patients in each bin, and
- weighted inversely proportional to the estimated variance of the within-bin difference.

The overall difference from the latter option will always appear to be more precise, but this weighting typically downweights results from the outer (first and last) bins. The overall

difference using weights proportional to total numbers of patients (usually nearly equal across bins) may be much less biased, especially when the data contain outliers. After all, outliers can greatly inflate within-bin variances because within-bin sample sizes are reduced by a factor of five or more.

The R calling syntax for the function to compute “adjusted” outcome differences is...

```
SPSoutco(dframe, yvar, trtm, qbin, faclev=3)
```

The first four arguments of `SPSoutco` are required, and the other two are optional. For example, two consecutive R invocations could be

```
PSdie6mo <- SPSoutco(PSframe, lifepres, abcix, QUINT)
PScrdbil <- SPSoutco(PSframe, cardbill, abcix, QUINT)
```

The arguments to `SPSoutco()` are as follows:

`SPSoutco`'s first argument, `dframe=PSframe`, is almost always a returned `data.frame` from `SPSlogit`.

`SPSoutco`'s second argument, `yvar`, is an outcome measure for patients. Outcomes are results that were unknown at the time when patients were assigned (possibly non-randomly) to treatments. “NA”s are allowed in this `yvar`.

`SPSoutco`'s third argument, `trtm`, is almost always the “response” (first) term from the S “formula” used in `SPSlogit()`.

`SPSoutco`'s fourth argument, `qbin = QUINT`, is almost always the same bin indicator variable name as in the previous `SPSlogit()` or `SPSnbins()` invocation.

The optional argument for `SPSoutco()` is

`faclev` = the maximum number of distinct numerical values that a variable can assume and yet still be treated like an S “factor.”

The default value for this optional parameter is `faclev=3`.

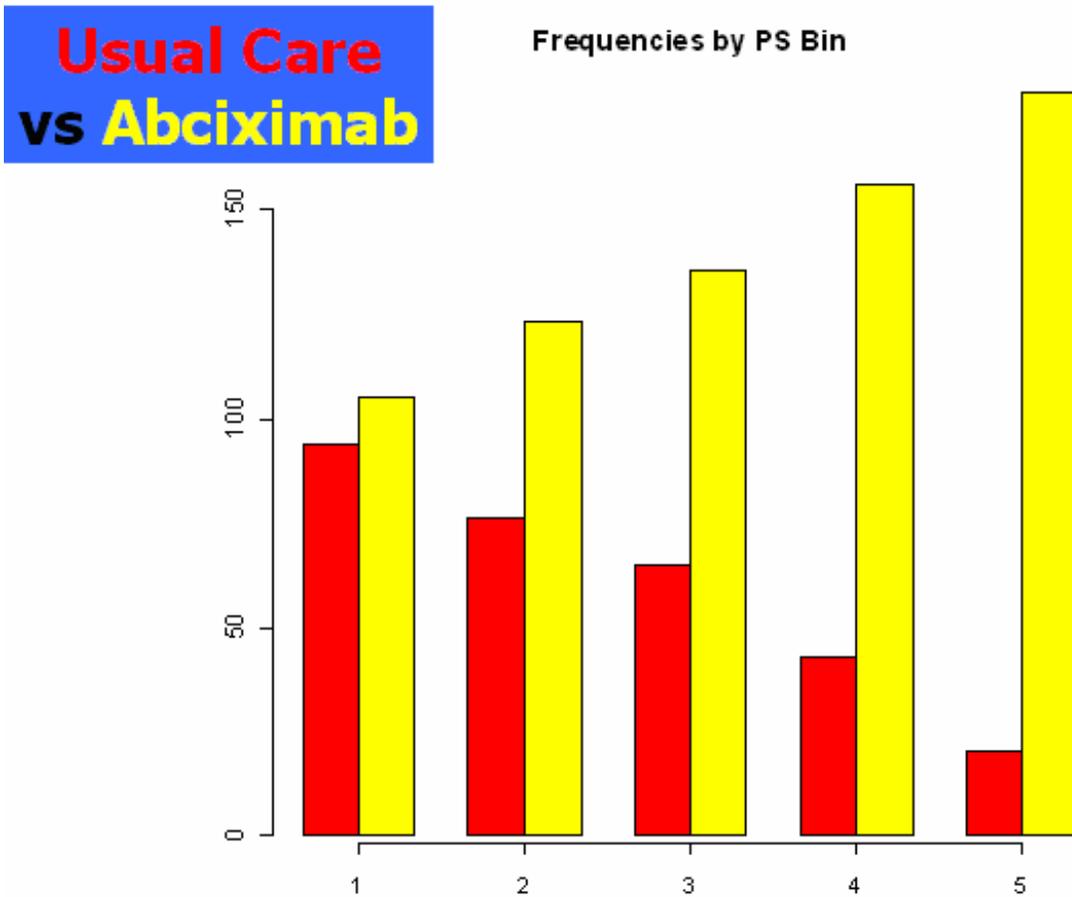
`SPSoutco()` returns a list of objects containing analysis details. However, in addition to the optional histogram plot “side effects” discussed above, `SPSoutco()` also prints out summaries of overall and within-bin analyses of `trtm` effects on `yvar`.

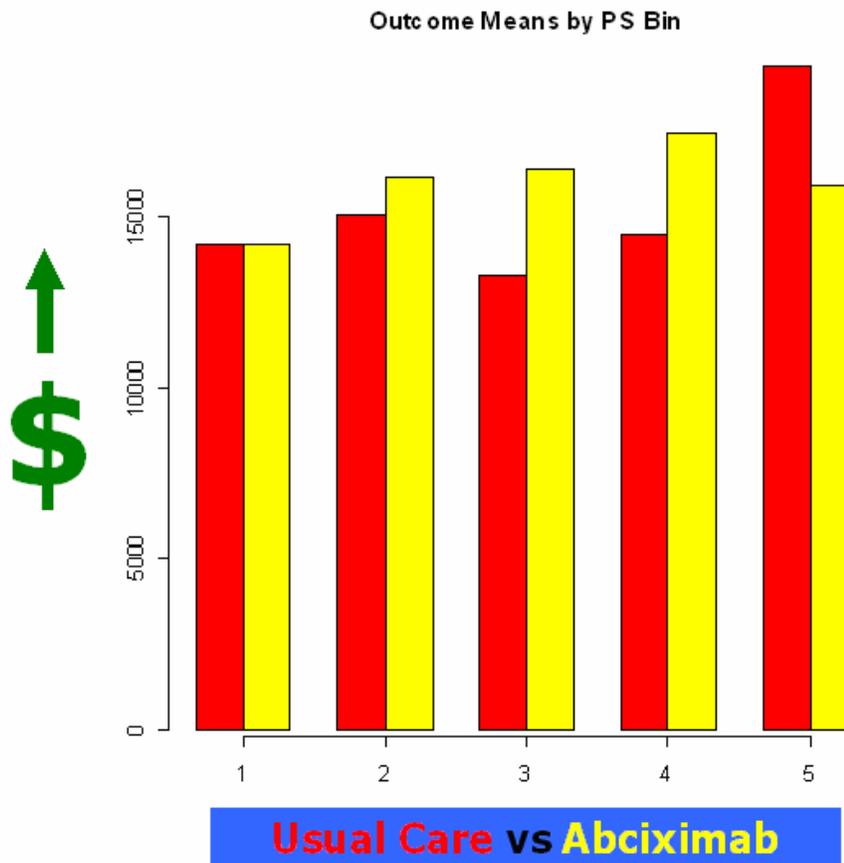
When the `yvar` data contain `faclev` or fewer levels, a summary of contingency table ChiSquare tests for `trtm` effects is printed. On the other hand, if this `yvar` actually is an R factor (character) variable, then `SPSoutco()` histograms will display mean values computed as if the numerical values for factor levels are 1, ..., `faclev`. As a result,

any `yvar` taking on only the numerical values 0 and 1 (meaning that outcome was not or was observed, respectively), should usually not be declared an R factor variable (with values “0”=1 and “1”=2.)

When the `yvar` data contain more than `faclev` levels, `yvar` is considered “continuous” and one-way and two-way ANOVA summaries are printed. (`trtm %in% bin` identifies “treatment within bin” effects.)

Note that `SPSoutco()` describes treatment differences in the distributions of outcomes using the same methodologies (ChiSquare or ANOVA) used by `SPSbalan()` on covariates. The key distinction here is that any outcome differences remaining after propensity scoring are called “adjusted” differences and do NOT signal problems with assumptions in the current PS analysis.





#### 16. Supervised Step Four: Explore Outcome Differences Expressed as Smooth Loess or Spline Functions of Propensity Score

The calculations currently performed in Step FOUR by `SPSloess()` and `SPSsmoot()` can only be applied to outcome measures that are continuous. And the logistic regression model fit in Step ONE using `SPSlogit()` needs to have been found satisfactory in Step TWO. After all, when one starts treating assigned propensity scores as continuous variables (rather than forming discrete bins of similar scores), it becomes much more difficult to test / verify the implications of the PS matching theorem (i.e. that the distribution of covariates is independent of treatment selection.)

MOTIVATION: Suppose one has fitted a somewhat smooth (loess or spline) curve through the observed outcome (Y) versus fitted propensity score (X) scatter for each of the two treatment groups. Now, consider the question:

**“Over the range where both smooth curves are defined (i.e. their common support), what is the (weighted) average signed difference between these two curves?”**

If the distribution of patients (either treated or untreated) were UNIFORM over this range, the (unweighted) average signed difference (treated minus untreated) would be an appropriate

estimate of the overall difference in outcome due to choice of treatment.

Histogram patient counts within 100 cells of width 0.01 provide a naive “non-parametric density estimate” for the distribution of total patients (treated or untreated) along the propensity score axis. The weighted average difference (and standard error) displayed by `SPSloess()` and `SPSsmoot()` are based on an `Rdensity()` smooth of these counts.

In situations where the propensity scoring distribution for all patients in a therapeutic class is known to differ from that of the patients within the current study, that population weighted average would also be of interest. Thus the `SPSloess()` returned value contains two data frames, `logrid` and `lofit`, useful in further computations; the corresponding data frames returned by `SPSsmoot()` are named `ssgrid` and `ssfit`.

NOTE: The difference in average smooth (loess or spline) predictions (treated minus untreated) is not an appropriately weighted average, in the above sense. While this sort of computation would use propensity scores to make a cost prediction for each patient, no “matching” of treated and untreated patients (with nearly equal propensity scores) is used in this sort of calculation.

SYNTAX: The R calling syntax for the functions to compute outcome differences (treatment=1 minus treatment=0) under the assumption that expected outcome is a smooth (loess or spline) function of propensity score within each treatment cohort are...

```
SPSloess(dframe, yvar, ps, trtm, faclev=3, display = T, deg=2,  
         sp=0.75, fam="symmetric", tcol="black", ucol="red")
```

and

```
SPSsmoot(dframe, yvar, ps, trtm, faclev=3, df=5, spar=NULL,  
         cv=F, penalty=1, display = T, tcol="black", ucol="red")
```

The first four arguments of `SPSloess` and `SPSsmoot` are required, and the other five or six are optional. For example, two consecutive R invocations could be

```
PScbillo <- SPSloess(PSframe, cardbill, PSFIT, ABCIX)  
PSframe$TRIMBILL <- pmin( PSframe$cardbill, 50000)  
PStbillo <- SPSloess(PSframe, TRIMBILL, PSFIT, ABCIX)
```

The `fam="symmetric"` default option of `SPSloess` tends to be fairly robust to outlying outcomes, at least when the loess span (default `sp = 1/10`) is wide enough. Thus reducing (Winsorizing) outlying `cardbill` values to \$50K (as illustrated above) should have little effect on a fitted loess smooth with an appropriate span. Looking for the effects of Winsorizing on `SPSloess()` or `SPSsmoot()` results is a form of “sensitivity analysis.”

The arguments to `SPSloess()` are as follows:

`SPSloess`'s first argument, `dframe=PSframe`, is almost always a returned `data.frame` from `SPSlogit()`.

`SPSloess`'s second argument, `yvar`, must be a continuous outcome measure. Outcomes are results that were unknown at the time when patients were assigned (possibly non-randomly) to treatments. "NA"s are allowed in this `yvar`.

`SPSloess`'s third argument, `ps=PSFIT`, is almost always a set of fitted propensity scores from a previous `SPSlogit()` invocation.

`SPSloess`'s fourth argument, `trtm`, is almost always the "response" (first) term from the R "formula" used in `SPSlogit()`, which is a "factor" variable taking on only two different levels.

The seven optional arguments of `SPSloess()` are

`faclev` = the maximum number of distinct numerical values that a variable can assume and yet still be automatically converted into an R "factor",

`display` = display graphical output (T or F.)

`deg` = degree (1=linear or 2=quadratic) of the local fit.

`sp` = span (zero to two) of the local regression fit, and

`fam` = "gaussian" or "symmetric."

`tcol` = color loess curve for treated group (default "black".)

`ucol` = color loess curve for untreated group (default "red".)

The arguments to `SPSmoot()` are as follows:

`SPSmoot`'s first argument, `dframe=PSframe`, is almost always a returned `data.frame` from `SPSlogit`.

`SPSmoot`'s second argument, `yvar`, must be a continuous outcome measure. Outcomes are results that were unknown at the time when patients were assigned (possibly non-randomly) to treatments. "NA"s are allowed in this `yvar`.

`SPSmoot`'s third argument, `ps = PSFIT`, is almost always a set of fitted propensity scores from a previous `SPSlogit()` invocation.

SPSsmoot's fourth argument, `trtm`, is almost always the "response" (first) term from the R "formula" used in `SPSlogit()`, which is a "factor" variable taking on only two different levels.

The eight optional arguments of `SPSsmoot()` are

`faclev` = the maximum number of distinct numerical values that a variable can assume and yet still be treated as an S "factor",

`cv` = ordinary cross-validation (T) or generalized cross-validation, GCV (F).

`df` = degrees-of-freedom of B-spline fit (5 bins .

`spar` = `spar` of the `smooth.spline()` function, and

`penalty` = coefficient of penalty for `df` in the GCV criterion.

`display` = display graphical output (T or F.)

`tcol` = color of spline for treated group (default "black".)

`ucol` = color of spline for untreated group (default "red".)

This scatterplot displays patient propensity score along the horizontal axis and his/her corresponding observed (continuous) outcome along the vertical axis. Patients receiving the "standard" treatment (`trtm=0`) are represented by cyan circles, while patients receiving the "new" treatment (`trtm=1`) are represented by magenta triangles. The smooth fits of outcome to propensity score within treatment cohorts are shown as cyan (`trtm=0`) and magenta (`trtm=1`) curves, respectively, superimposed upon the scatter.

The smooth fits can be difficult to see when the scatters contain many points. Thus `SPSloess` and `SPSsmoot` each draw a second plot rescaled to show only the two smooth (lowess or spline) fits, again using cyan (`trtm=0`) and magenta (`trtm=1`) curves. (For details, see the returned `lofit` and `ssfit` data frames.)

Finally, `SPSloess` and `SPSsmoot` each draw a third plot to show total patient frequencies (black circles) within a 100-cell histogram along the propensity score axis as well as the corresponding `density()` smooth in red. (For details, see the returned `logrid` and `ssgrid` data frames.)

In addition to the graphs described above, the primary "side effects" of `SPSloess()` and `SPSsmoot()` consist of printouts of outcome differences (unadjusted and adjusted) and their standard deviations.

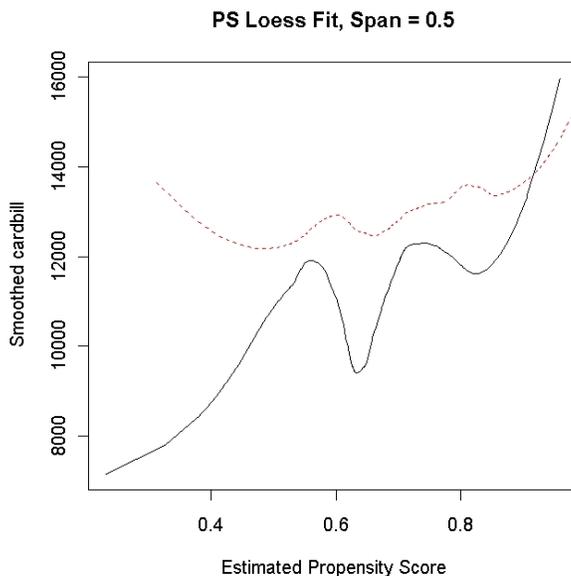
The `SPSloess( )` or `SPSsmoot( )` returned value is a list of two data frames, a **grid** frame and a **fit** frame, plus other objects giving analysis details.

**logrid** and **ssgrid** each contain 11 variables and 100 observations. The **PS** variable (column one) contains propensity score “cell means” of 0.005 to 0.995 in steps of 0.010. Variables **F0**, **S0** and **C0** for treatment 0 and variables **F1**, **S1** and **C1** for treatment 1 contain fitted smooth (lowess or spline) values, standard error estimates and patient counts, respectively. Observations with “NA” for variables **F0**, **S0**, **F1** or **S1** represent “extremes” where the lowess fits could not be extrapolated because no observed outcomes were available. The **DIF** variable is simply  $(F1-F0)$ , the **SED** variable is  $\sqrt{S1^2+S0^2}$ , the **HST** variable is proportional to  $(C0+C1)$ , and the **DEN** variable is the estimated probability density of patients along the PS axis.

**lofit** contains 4 variables for all observations in data frame = `dframe` that have no “NA” values in the `yvar`, `ps` or `trtm` variables. These 4 variables are named **PS**, **YVAR**, **TRT** (with levels 0 and 1 recoded to 1 and 2, respectively) and **FIT** = loess prediction for the specified “span” (default `sp=1/10`.)

**ssfit** contains 4 variables for each distinct PS value in data frame = `dframe`. These 4 variables are named **PS**, **YAVG**, **TRT** (with levels 0 and 1 recoded to 1 and 2, respectively) and **FIT** = spline prediction for the specified degrees-of-freedom (default `df=10`.)

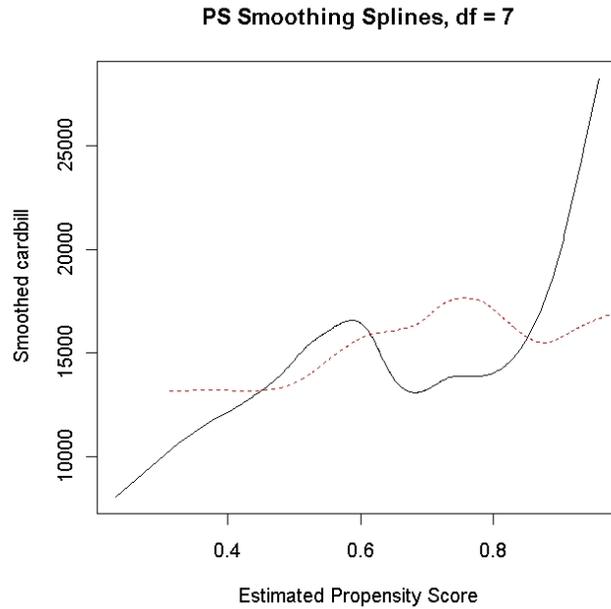
### Example SPSloess fit for cardbill versus Propensity Score



**Solid loess fit gives smoothed cardbill estimates for ABCIX patients.**

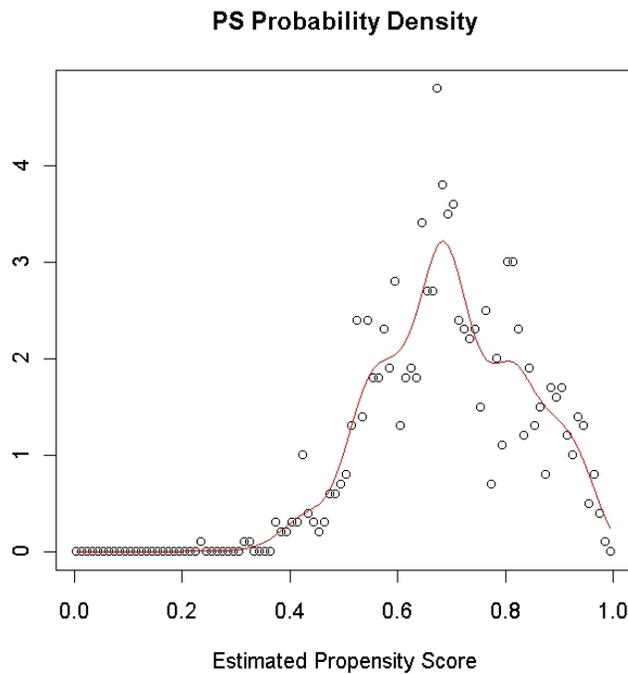
**Dashed loess fit gives smoothed cardbill estimates for Usual-Care-Only patients.**

### Example SPSSmoot fit for cardbill versus Propensity Score



**Solid spline gives smoothed cardbill estimates for ABCIX patients.**  
**Dashed spline gives smoothed cardbill estimates for Usual-Care-Only patients.**

### Example Patient Distribution (abcix + usual care) along fitted Propensity Score Axis



**Black circles give normalized histogram estimates for 100 cells (0.005 to 0.995).**  
**Red curve gives Gaussian kernel density estimator for the PS distribution of patients.**

Note that the above distribution of patients at the Lindner Center is somewhat shifted to the right because almost 70% of all Lindner PCI patients did receive abciximab in 1997.

### **17. R and S-Plus functions that are (or could be) called in PS/NN/IV analyses.**

`SPSlogit()` currently models propensity scores via a call to `glm()` with `family = binomial()` (and thus default `link=logit`), but a variety of alternatives could be quite useful in applications. For example, fits from `gam()` or a classification `tree()` model would relax the “linear functional” requirement of `glm()` or automatically incorporate interactions among covariates, respectively. The `lrm()` function from the Harrell(1997) “design” library could be used to penalize logistic regression parameters, but it is debatable whether inter-correlations (or even non-linear relationships) among covariates can be harmful in propensity score estimation. After all, our primary interest here is restricted to simply making predictions; all we need are fitted values within the closed interval  $0 \leq PS \leq 1$  that estimate the probability of treatment choice given all available covariates. Any potential problems with significance testing or (causal) interpretations for parameters are almost irrelevant. In fact, D’Agostino(1998) essentially recommends drastic over-fitting by including all potentially relevant covariates in one’s PS model.

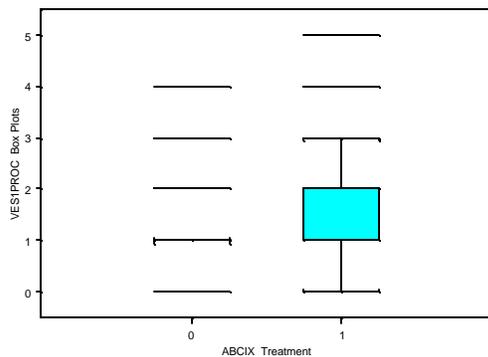
`SPSloess()` and `SPSsmoot()` use the R `loess()` and `smooth.spline()` functions, respectively. Cleveland’s original `lowess()` function could be used here because only one X variable (namely, fitted propensity score) is involved, but I choose `loess()` to give users flexibility to choose between `fam="gaussian"` and `fam="symmetric"`, which provides some resistance to outlying outcome values.

The `df` parameter of `SPSsmoot()` brings considerable intuitive appeal to one’s choice of smoothness; see Hastie and Tibshirani(1990). For example, the `SPSoutco()` approach with `bins=5` clearly corresponds to `df=5`, but “binning” outcome analyses correspond to fits that are discontinuous at cut points. `SPSsmoot()` fits cubic smoothing splines that are not only continuous but also have continuous first and second derivatives.

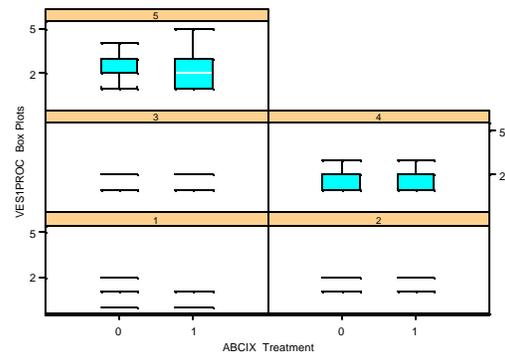
`SPSloess()` and `SPSsmoot()` both call the R `density()` function to generate a non-parametric probability density estimate for the distribution of patients along the fitted PS axis. This density is evaluated at 100 points evenly spaced between `PS=0.005` and `PS=0.995`, and signed differences between the (lowess or cubic spline) smooths at these same points are weighted proportional to this density. Bandwidth for this Gaussian kernel density estimator is chosen using the R default `bw="nrd0"` option. Alternatively, see Venables and Ripley(1999), page 137.

Like “Lattice” graphics in R, S-Plus “Trellis Graphics” can be extremely useful in visualizing within-bin balance achieved via propensity scoring. For example, the plots below illustrate balance issues related to the “ves1proc” X-variable within the abciximab case study. Even major overall distributional differences will, ideally, almost “disappear” as a result of PS binning.

## Overall “ves1proc” Distributions



## Within-Bin “ves1proc” Distributions



## 18. Summary

As is clear from the above descriptions and examples, the `SPSlogit()`, `SPSbalan()` and `SPSoutco()` R functions provide only relatively simple and straight-forward analyses. I started out performing such computations primarily in JMP and Stata. I then used text files to port tables of means and sample sizes to Excel to draw histograms. Although possibly pedagogical, I quickly realized that my original “mostly-point-and-click” analysis process was actually highly repetitive, tedious, error-prone and produced no satisfactory audit-trail for reproducing analyses.

Like many other interesting forms of data analysis, I think that propensity scoring and instrumental variables adjustment methods need to be viewed as highly iterative, discovery processes. Successfully making ones way through SPS step TWO can require several returns to SPS step ONE. Only then can results from SPS steps THREE and FOUR be considered meaningful. And convincing UPS analyses require exploring alternative clustering algorithms and patient dissimilarity metrics in step ONE as well as a varying numbers of cluster-bins in steps TWO and/or THREE. Outliers and missing values (NAs) can provide frustrations throughout these journeys. The R functions described here hopefully provide enough basic support (if only relief from tedium) to encourage PS practitioners to persevere and end up feeling confident that their “sensitivity analyses” have been thorough.

The `SPSloess()` and `SPSsmoot()` functions are still in their early stages of development. `SPSloess()` fits can tend to look rather “rough” compared to `SPSsmoot()` fits. Cubic spline smoothing appears to give answers that are interpretable as smoothed mean values for highly skewed distributions. Loess smoothing, at least when `fam="symmetric,"` tends to give answers more easily interpretable as modes or medians of highly skewed distributions. This median versus mean analogy may help explain why the weighted average signed treatment differences from `SPSloess()` tend to seem more precise than those from `SPSsmoot()` for highly skewed distributions.

## References

- Alzola C, Harrell FE. *An Introduction to S-Plus and the Hmisc and Design Libraries*. School of Medicine, University of Virginia, Charlottesville, VA. 1999.
- Angrist JD, Imbens GW, Rubin DB. "Identification of Causal Effects Using Instrumental Variables." *J Amer Stat Assoc* 1996; 91: 444-472.
- Barlow HB. "Unsupervised learning." *Neural Computation* 1989; 1: 295-311.
- Becker RA, Chambers JM, Wilks AR. *The New S Language*. Chapman & Hall, New York. 1988.
- Chambers JM, Hastie TJ, eds. *Statistical Models in S*. Chapman & Hall, New York. 1992.
- Cleveland WS, Devlin SJ (1988) "Locally-weighted regression: an approach to regression analysis by local fitting." *J Amer Stat Assoc* 83, 596-610.
- Cochran WG. "The effectiveness of adjustment by subclassification in removing bias in observational studies." *Biometrics* 1968; 24: 205-213.
- D'Agostino RB Jr. "Tutorial in Biostatistics: Propensity score methods for bias reduction in the comparison of a treatment to a non-randomized control group." *Stat Med* 1998; 17, 2265-2281.
- Hastie TJ, Tibshirani RJ. *Generalized Additive Models*. Chapman and Hall, London. 1990.
- Harrell FE (1997) *Predicting Outcomes: Applied Survival Analysis and Logistic Regression*. University of Virginia, Charlottesville, VA.
- Ihaka R, Gentleman R. "R: A language for data analysis and graphics." *J Comp & Graph Stat* 1996; 5(3): 299-314.
- Kaufman L, Rousseeuw PJ. *Finding Groups in Data. An Introduction to Cluster Analysis*. New York: John Wiley and Sons. 1990.
- Kereiakes DJ, Obenchain RL, Barber BL, Smith A, McDonald M, Broderick TM, Runyon JP, Shimshak TM, Schneider JF, Hattamer CH, Roth EM, Whang DD, Cocks DL, Abbottsmith CW. Abciximab provides cost effective survival advantage in high volume interventional practice. *Am Heart J* 2000; 140: 603-610.
- McClellan M, McNeil BJ, Newhouse JP. "Does More Intensive Treatment of Myocardial Infarction in the Elderly Reduce Mortality?: Analysis Using Instrumental Variables." *JAMA* 1994; 272: 859-866.

- Obenchain RL. "Nearest Neighbors Analysis for PRRAP, the Probable Report Rate Analysis Plan." **Bell Laboratories** TM-79-1711-4. 1979. Holmdel, NJ.
- Obenchain RL, Melfi CA. "Propensity Score and Heckman Adjustments for Treatment Selection Bias in Database Studies." **Proceedings of the Biopharmaceutical Section**, 1997; 297-306. Washington, DC: American Statistical Association.
- Obenchain RL. "Unsupervised Propensity Scoring: NN and IV Plots." **2004 Proceedings of the American Statistical Association** (on CD.) 8 pages.
- Obenchain RL. "Unsupervised Propensity Scoring: Nearest Neighbor and Instrumental Variable Adjustment for Treatment Selection Bias. **Lilly US Medical Outcomes Research White Paper**. 45 pages. 2006a.
- Obenchain RL. "USPS\_1\_01.zip: An R package for unsupervised and supervised propensity score adjustment." 2006b. <http://www.math.iupui.edu/~indyasa/download.htm>.
- Rosenbaum PR, Rubin RB. "The Central Role of the Propensity Score in Observational Studies for Causal Effects." *Biometrika* 1983; 70: 41-55.
- Rosenbaum PR, Rubin DB. "Reducing Bias in Observational Studies Using Subclassification on a Propensity Score." *J Amer Stat Assoc* 1984; 79: 516-524.
- Rosenbaum PR, Rubin DB. Constructing a control group using multivariate matched sampling methods that incorporate the propensity score. *Amer Statist* 1985; 39: 33-38.
- Rosenbaum PR. "Optimal matching in observational studies." *J Amer Stat Assoc* 1989; 84: 1024-1032.
- Rosenbaum PR. "Multivariate matching methods." In: Kotz S, Read CR, Banks D, eds. *Encyclopedia of Statistical Sciences*, Update Volume 2. New York: J Wiley 1998: 435-438.
- Rosenbaum PR. *Observational Studies, Second Edition*. New York: Springer-Verlag 2002.
- Rubin DB. "Bias reduction using Mahalanobis metric matching." *Biometrics* 1980; 36: 293-298.
- S-PLUS User's Guide* (1999) and "S-PLUS 2000 On-Line Help" for the loess(), smooth.spline(), predict() and density() functions. Data Analysis Products Division, Insightful Corporation, Seattle, WA.
- Venables WN, Ripley BD. *Modern Applied Statistics with S-Plus, Third Edition*. New York; SpringerVerlag. 1999.

## Software Updates:

- 2003.08      Fix error in standard deviation computation for the “weighted by bin size” treatment difference estimate in `SPSoutco( )` and `UPSnnltd( )`.
- 2004.01      Major upgrade to “object oriented” style; add UPS “sensitivity analysis” overview `UPSgraph( )`; fix `SPSbalan( )` trellis-style (lattice library) graphic; unfortunately, new argument sequencing not backward compatible with earlier versions.
- 2006.09      Upgrade functions (fix bugs and clarify terminology / titles / labels) and add `UPSaltdd()` functionality for computing and visualizing the Artificial Distribution of LTDs due to random patient clusterings.

## Unsupervised R functions:

=====

```
xvars <- c("x1", "x2", ..., "xN")

UPlinint <- function(q, xmin, n, x, w)

UPShclus <- function(dframe, xvars, method=diana)
  plot.UPShclus <- function(x)
  print.UPShclus <- function(x)

UPSaccum <- function(hiclus, dframe, trtm, yvar, faclev=3,
  accobj="UPSframe")

UPSaltdd <- function(dframe, trtm, yvar, faclev=3,
  scedas="homo", NNobj=NA, clus=50,
  reps=10, seed=12345)
  plot.UPSsldd <- function(x, breaks="Sturges")
  print.UPSaltdd <- function(x)
```

```
UPSnnltd <- function(numclust)
  plot.UPSnnltd <- function(x)
  print.UPSnnltd <- function(x)
  summary.UPSnnltd <- function(x)
```

```
UPSivadj <- function(numclust)
  plot.UPSivadj <- function(x)
  print.UPSivadj <- function(x)
  summary.UPSivadj <- function(x)
```

```
UPSgraph <- function(nncol="red", nwcol="green3", ivcol="blue")
```

## Supervised R functions:

=====

```
form <- trtm~x1+x2+...+xN
```

```
SPSlogit <- function(dframe, form, pfit, prnk, qbin, bins=5,
  appn="")
  print.SPSlogit <- function(x)
```

```
SPSbalan <- function(dframe, trtm, qbin, xvar, faclev=3)
  plot.SPSbalan <- function(x)
  print.SPSbalan <- function(x)
```

```
SPSoutco <- function(dframe, trtm, qbin, yvar, faclev=3)
  plot.SPSoutco <- function(x)
  print.SPSoutco <- function(x)
  summary.SPSoutco <- function(x)
```

```
SPSsmoot <- function(dframe, trtm, pscr, yvar, faclev=3, df=5,
  spar=NULL, cv=F, penalty=1)
  plot.SPSsmoot <- function(x, tcol="blue", ucol="red",
  dcol="green3")
  print.SPSsmoot <- function(x)
```

```
SPSloess <- function(dframe, trtm, pscr, yvar, faclev=3, deg=2,
  span=0.75, fam=symmetric)
  plot.SPSloess <- function(x, tcol="blue", ucol="red",
  dcol="green3")
  print.SPSloess <- function(x)
```