Outlier Identification and Imputation using Robust Regression Trees

Zhao Xinqiang and Ray Chambers
Department of Social Statistics
University of Southampton

28 March 2002

EUREEDIT DELIVERABLES

D4.2.1: See Sections 1 and 2
D4.2.2: See Appendices A, B
1. Overview of the Approach

1.1 The WAID Toolkit

WAID is software for building regression and classification trees. The original version of this software was intended for automatic imputation of missing data in censuses and surveys, and was developed as a C++ Windows application under the AUTIMP project (R.L. Chambers, J. Hoogland, S. Laaksonen, D.M. Mesa, J. Pannekoek, P. Piela, P. Tsai and T. De Waal, 2001, The AUTIMP-Project: Evaluation of Imputation Software. Report, Statistics Netherlands, Voorburg). Under the EUREDIT project a "toolkit" of Splus programs has been created that emulates and extends the capabilities of WAID. These programs work both under Splus and under R, a public domain statistical software package that is compatible with Splus. Availability of this toolkit means that it is now easy to create statistical applications that "build" on the WAID algorithm. Similar software products are CART from Salford Systems, the Splus tree() function and the CHAID module in SPSS. The code for the programs in the WAID toolkit is set out in Appendix 1.

The basic idea behind WAID (and other tree modelling software packages) is to sequentially divide the original data set into subgroups (or nodes) that are increasingly more homogeneous with respect to the values of a response variable. The splits themselves are defined in terms of the values of a set of categorical covariates. The WAID splitting algorithm is described in the next section. By definition, WAID is a nonparametric statistical procedure. It also has the capacity to implement outlier robust splitting based on M-estimation methodology. In this case outliers are downweighted when calculating the measure of within node heterogeneity (weighted residual sum of squares) used to decide whether a node should be split or not. The weights used for this purpose are themselves based on outlier robust influence functions.

1.2 Outlier Identification using WAID

The basic idea behind the WAID outlier identification strategy is straightforward. Each time WAID splits the data set to create two new nodes it creates a new set of robust weights for the units making up those nodes. These weights are scaled so that they sum to the number of observations in the node, with outliers receiving weights close to zero and inliers receiving weights around one. These weights reflect distance from a robust estimate of location for the values in the node. Consequently a value that is not immediately identifiable as an outlier within "larger" nodes is more likely to become identified as such as it is classified into smaller and smaller nodes. In effect, the weights associated with such units tend to move towards zero. The WAID outlier identification algorithm defines an outlier as an observation with an average weight over all node splits that is less than a specified threshold. This threshold is defined as the value at which the most observations that are "real" errors are identified as outliers and where the least number of error-free observations are classified as outliers. See Section 2 for details. The process itself can be defined in terms of the following three steps.

- Step 1: Build a robust tree using WAID.
- Step 2: Define a weight "threshold".
• Step 3: Identify outliers as those with average weights less than this threshold.

In practice of course we do not know which data values are errors and which are not. Consequently it is difficult to identify the optimal threshold. Where historical data are available on errors, the threshold can be defined in terms of this historical information. This is the approach taken with the ABI application reported on in section 3. An alternative approach is to randomly perturb the current data and to then define the optimal threshold in terms of maximal correct identification of perturbed values and minimal incorrect identification of non-perturbed values. Such a perturbation scheme could be created based on the residuals generated by the robust WAID tree. This is an area that needs further investigation.

1.3 Outlier and Missing Data Imputation using WAID

Once a set of outliers have been identified, the robust tree structure generated by the WAID algorithm can be used to impute replacement values for these units. These imputed values can be derived in many different ways. However, the lessons learned in the AUTIMP project indicate that two imputation methods in particular are suited for use with WAID trees. These are mean imputation and random donor imputation within terminal nodes. In the context of outlier contaminated data, these imputation methods need to be modified as follows:

• Use a robust weighted mean within a terminal node as the "mean" imputation.
• Identify a donor within a terminal node by randomly selecting a value from those cases within the node whose average weights are greater than a specified threshold.

Note that imputation for missing (rather than outlier) values proceeds in exactly the same way. That is, the record with a missing value is "dropped" down the WAID tree until it reaches a terminal node. It is then imputed using either the weighted mean for the node or via a donor value obtained from a non-outlier in the node.

2. The WAID Outlier Identification Algorithm

2.1 Introduction

In this section we describe the three main components of WAID outlier identification methodology. The first is the algorithm used to build a robust regression tree in WAID. The second is the procedure for identifying potential outliers based on a robust WAID tree. The last component describes the algorithm used to find the weight threshold.

WAID assumes a rectangular data set containing n observations, values \{y_i\} of a scalar response variable Y and values \{x_{1i}, ..., x_{pi}\} of p covariates X_1, ..., X_p. The values of Y are assumed to contain outliers. In contrast, the covariates X_1, ..., X_p are all assumed to be categorical. No missing X-values are allowed in the current R version of WAID. For a scalar response variable WAID builds a regression tree. If the response variable is categorical, WAID builds a classification tree. The only difference between these two types of trees is the heterogeneity measure used to
determine tree splitting behaviour. Since our focus is outlier identification, we are concerned with scalar response variables only and so we restrict consideration to WAID's regression tree algorithm.

2.2 The WAID Regression Tree Algorithm

The basic idea used in WAID (as well as other tree-based methods) is to split the original data set into smaller subsets or nodes in which the Y-values are more homogeneous. In WAID this is accomplished by sequential binary splitting. At each step in the splitting process, all nodes created up to that point are examined in order to identify the one with minimal homogeneity. An optimal binary split of this "parent" node is then carried out. This is based on identifying a set of values of one of the covariates $X_1, \ldots, X_p$ such that a split of the parent node into one child node containing only cases possessing these values and another child node containing the remaining cases maximizes (minimizes) the homogeneity (heterogeneity) of these child nodes. The measure of heterogeneity used is the weighted sum of squares of residuals (WSSR) defined with respect to a robust measure of the location of the Y-values in the parent node. The splitting process continues until a suitable stopping criterion is met. At present this is when either (i) all candidate parent nodes are effectively homogeneous; (b) all candidate parent nodes are too small to split further; or (c) the maximum number of nodes is reached. There is no attempt to find an "optimal" tree. The set of nodes defining the final tree are typically referred to as the terminal nodes of the tree.

The two key algorithms used to determine a node split are the algorithm for calculating the robust weight (used in calculation of WSSR) and the algorithm for finding the optimal split. We describe them below.

2.2.1 Calculation of WSSR and Robust Weights

We denote the values of the response variable Y in node k by $y_1, \ldots, y_{n_k}$, where $n_k$ is the number of observations in node k. The weighted sum of squared residuals within this node (WSSR$_k$) can be written as follows

$$WSSR_k = \sum_{i=1}^{n_k} w_i (y_i - \bar{y}_{wk})^2$$

where $w_i$ is the weight attached to $i^{th}$ case in node k and $\bar{y}_{wk}$ is the weighted mean of Y in node k,

$$\bar{y}_{wk} = \frac{\sum_{i=1}^{n_k} w_i y_i}{\sum_{i=1}^{n_k} w_i}.$$ 

The weight $w_i$ is calculated as the ratio

\[ \text{ratio} \]
\[ w_i = \frac{\psi(y_i - \overline{y}_{wk})}{y_i - \overline{y}_{wk}}. \]

where \( \psi(x) \) denotes an appropriately chosen influence function. The Splus/R toolkit version of WAID computes these weights by first calling the robust regression function `rlm()` in the MASS robust statistics library (available for both R and Splus). This function returns weight values which are then rescaled within WAID to sum to the number \( n_k \) of cases within node \( k \).

A standard (nonrobust) regression tree in WAID uses the Ordinary Least Squares (OLS) influence function, \( \psi(x) = x \), in which case \( w_i = 1 \). Besides OLS, WAID currently has three robust weighting schemes, corresponding to the weighting options available for `rlm()`. These correspond to use of Huber's Min/Max, Tukey's Biweight and Hampel's Redescending influence functions. These are defined as follows:

**Huber's Min/Max**: \[ \psi(x) = \min(|x|, c) \]
where \( c \) is a tuning constant. With \( c = 1.345 \) this influence function achieves 95\% efficiency when estimating the mean of a normal distribution.

**Tukey's Biweight**: \[ \psi(t) = t \left(1 - \frac{t^2}{R^2}\right)^2 \]
where \( R \) is a tuning constant. The value \( R = 4.685 \) gives 95\% efficiency at the normal (Ripley, 1994).

**Hampel's Redescending**: \[ \psi(t) = \begin{cases} 
  t & |t| \leq a \\
  a \text{sgn}(t) & a < |t| \leq b \\
  a(c \text{sgn}(t) - t) / (c - b) & b < |t| \leq c \\
  0 & |t| > c 
\end{cases} \]
where \( a \), \( b \) and \( c \) are tuning constants (Hampel, 1986). In passing we observe that it is a straightforward matter to add extra weight functions for use by `rlm()` and hence by WAID.

### 2.2.2 Optimal Splitting

Without loss of generality, we consider optimal splitting of the \( k^{th} \) parent node. Suppose there are \( n_k \) observations in this node and \( p \) covariates \( X_1, \ldots, X_p \). The best split for each covariate \( X_j \), \( j = 1, \ldots, p \) is defined by the subset of values of \( X_j \) that generates child nodes with a WSSR value smaller than that generated by any other split of the parent node based on \( X_j \). The optimal split overall is then defined as the best (in terms of minimising the WSSR value generated by the resulting child nodes) among these covariate-specific optimal splits.

For convenience, we let \( V_j \) denote the set containing the \( m_j \) unique values of \( X_j \) for the cases in the parent node. The search procedure for the best split based on \( X_j \) is then as follows.
Step I: Sort the $m_j$ values in $V_j$. WAID allows a covariate to be declared as monotone or non-monotone. If $X_j$ is monotone then the values in $V_j$ are sorted in ascending order. If $X_j$ is non-monotone, then the values in $V_j$ are sorted as \{v_1, v_2, \ldots, v_{m_j}\} where $\bar{y}_{wk}(v_1) < \bar{y}_{wk}(v_2) < \cdots < \bar{y}_{wk}(v_{m_j})$ and

$$\bar{y}_{wk}(v_q) = \frac{\sum_{i \in \text{Node } L_{qkj}} w_i y_i}{\sum_{i \in \text{Node } L_{qkj}} w_i} \quad \text{and} \quad \bar{y}_{wk}(v_q) = \frac{\sum_{i \in \text{Node } R_{qkJ}} w_i y_i}{\sum_{i \in \text{Node } R_{qkJ}} w_i}$$

Let $L_{qj}$ denote the values in $V_j$ that are to the "left" of $v_q$ with $R_{qj}$ denoting the corresponding set of values in $V_j$ that are to the "right" of $v_q$. We can then split the parent node into two child nodes at this value $v_q$. This split corresponds to a left child node (Node$_{Lqkj}$) and right child node (Node$_{RqkJ}$) defined by

$$\text{Node}_{Lqkj} = \{i \in \text{node } k, x_{ij} = v_s, 1 \leq s \leq q\}$$
$$\text{Node}_{RqkJ} = \{i \in \text{node } k, x_{ij} = v_s, q < s \leq m_j\}$$

Step II: Calculate a WSSR value for each of the $m_j - 1$ possible splits of $V_j$. For each $v_q \in V_j$ the WSSR values for the child nodes corresponding to a split at this value are

$$WSSR_{Lqkj} = \sum_{i \in \text{Node}_{Lqkj}} w_i (y_i - \bar{y}_{Lwkj})^2$$
$$WSSR_{RqkJ} = \sum_{i \in \text{Node}_{RqkJ}} w_i (y_i - \bar{y}_{Rwkj})^2$$

where $\bar{y}_{Lwkj} = \frac{\sum_{i \in \text{Node}_{Lqkj}} w_i y_i}{\sum_{i \in \text{Node}_{Lqkj}} w_i}$ and $\bar{y}_{Rwkj}$ is defined similarly. The within-group WSSR associated with a split at $v_q \in V_j$ is then

$$WSSR_{qkj} = WSSR_{Lqkj} + WSSR_{RqkJ}$$

Step III: Find the best splitting value $v_q \in V_j$. This is the value $v_q$ that generates the minimum value of $WSSR_{qkj}$ among all values $v_q$ in $V_j$. The overall best split among the candidate best splits defined by each covariate is then the split that generates the minimum value of $WSSR_{qkj}$ over both $q$ and $j$.

2.3 Outlier identification using WAID

We can calculate an "average" weight for every case in the original data set by averaging its weight over all splits defining the final tree. If a case is not involved in a particular split (it is not in the node that is selected for splitting) its weight remains unchanged from the last split in which it was involved. Outlier identification using a
robust WAID tree is then based on the fact that outliers are likely to have small weights in most of the nodes in which they appear in such a tree. Consequently they will also have small average weights across all node splits, and we therefore can identify them as outliers on the basis that their average weight lies below a defined threshold. Note that we do not distinguish between outliers that appear early on in the construction of the tree and then gradually become "less" outlier-like because of progressive refinement by the tree-building process and outliers that are "hidden" early on in the tree-building process and then become more and more outlying relative to their within-node comparators in later stages of the tree.

The task is therefore one of finding a threshold value such that "important" outliers are identified as having average weights less than this threshold. The set of identified outliers for a given threshold w can be denoted

\[ \text{out}(w) = \{ i, \ \bar{w}_i < w, i=1, ..., n \} \]

where \( \bar{w}_i = \frac{1}{m} \sum_{k=1}^{m} w_{i}^{(k)} \) is the average weight of the \( i^{th} \) case, with \( w_{i}^{(k)} \) denoting the weight of the \( i^{th} \) unit at the \( k^{th} \) split, and \( m \) is the total number of splits defining the tree.

The main problem with this approach therefore reduces to finding a threshold \( w \) that successfully identifies outliers without also mis-identifying non-outliers. The optimal threshold value \( w \) is therefore the one that maximises the proportion of "true" outliers identified and minimises the proportion of "false" outliers identified. This requires information about true outliers to be available. In practice this information will not be available. However, information is often available about known errors in the data, most of which are associated with outlying values. Consequently, we can choose the optimal value of \( w \) to optimise identification of these errors.

Put \( N_{\text{errors}} \) equal to the total number of true errors, and, for a given threshold \( w \), put \( N_{\text{outliers}}(w) \) equal to the total number of outliers identified by WAID on the basis of the specified threshold \( w \), \( n_{\text{errors}}(w) \) equal to the corresponding number of errors identified as outliers, and \( n_{\text{non-errors}}(w) \) equal to the total number of non-errors identified as outliers. The proportion of error-generated outliers identified by WAID is then

\[ R_1(w) = \frac{n_{\text{errors}}(w)}{N_{\text{errors}}} \]

while the proportion of non-errors identified as outliers using the threshold \( w \) is

\[ R_2(w) = \frac{n_{\text{non-errors}}(w)}{N_{\text{outliers}}(w)} = 1 - \frac{n_{\text{errors}}(w)}{N_{\text{outliers}}(w)}. \]

We denote the optimal threshold value as \( w_{\text{opt}} \), where
\[ w_{\text{opt}} = \arg \max_{w} \{ R_1(w)(1 - R_2(w)) \}. \]

This definition is a simple implementation of the idea that at the optimal threshold value WAID identifies most of the error-generated outliers as well as minimises the number of non-errors identified as outliers. In the next section we illustrate this approach using the 1997 ABI data.

3. An Evaluation Using ABI Data

3.1 The ABI data

There are five ABI datasets available for evaluation in EUREDIT. See abimeta.xls for a general description of these datasets. Three of them are derived from 1997 ABI data. The remaining two are derived from 1998 ABI data. The first of these 1997 datasets (sec197(true)) is contains true values. The second (sec197(y2)) is derived from sec197(true) and only contains missing values. The third dataset (sec197(y3)) contains both missing values and adding errors. Similarly, one of the two evaluation datasets for the 1998 survey is a dataset containing only missing values, while the other is a dataset with missing value and errors.

In this section we focus on the 1997 ABI dataset that contains both missing values and errors (sec197(y3)). We present results from an empirical study based on this dataset that evaluates the error and outlier identification and imputation performance of a WAID-based approach. The dataset has a total of 6099 records, with 30 variables. The descriptions of these variables (from abimeta.xls) are given in Table 1.

Our interest is in the variables corresponding to totals in this dataset. There are seven such variables in the ABI dataset, TURNOVER, EMPTOTC, PURTOT, TAXBOT, ASSACQ, ASSDISP and EMPLOY. Initially, however we focus on total business turnover (TURNOVER), which we select as our response variable. Similarly, we consider two covariates corresponding to the register size information available on the ABI dataset. These are TURNREG (a positive-valued continuous variable corresponding to the register value of total turnover for a unit) and EMPREG (a ordinal variable corresponding to the employee size band of a unit). Both these covariates, being register variables, have no missing values and no errors and therefore could be considered as "natural" covariates for explaining variation in ABI variables. There are 42 missing values and 241 "error" values (i.e. values different from the corresponding "true" values in sec197(true)) for TURNOVER in sec197(true). In what follows the 42 missing values are excluded, to be imputed after the tree is formed. As an aside we note that the version of WAID implemented in the toolkit cannot handle missing values in the covariates used to define a tree. Consequently records containing one or more missing values in the covariates need to be excluded from the dataset used to build a WAID tree.
Table 1. Variable descriptions

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Variable label</th>
</tr>
</thead>
<tbody>
<tr>
<td>REF</td>
<td>Case reference number</td>
</tr>
<tr>
<td>CLASS</td>
<td>Anonymised industrial classification. The classification is hierarchical with the first digit indicating the higher level of classification. Numerically adjacent digits do not indicate similarity between classes/subclasses e.g. 3.1 is similar to 3.2 but not necessarily closer to 3.2 than 3.3.</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>Design weight (N/n) for category (1st digit of CLASS) and employment size band (EMPREG)</td>
</tr>
<tr>
<td>TURNOVER</td>
<td>Total turnover</td>
</tr>
<tr>
<td>EMPWAG</td>
<td>Wages and salaries paid</td>
</tr>
<tr>
<td>EMPNI</td>
<td>Employers NI contributions</td>
</tr>
<tr>
<td>EMPNIOTH</td>
<td>Employers NI contributions and other employment costs</td>
</tr>
<tr>
<td>EMPENS</td>
<td>Contributions to pension funds</td>
</tr>
<tr>
<td>EMPRED</td>
<td>Redundancy and severance payments to employees</td>
</tr>
<tr>
<td>EMPTOTC</td>
<td>Total employment costs</td>
</tr>
<tr>
<td>PUREN</td>
<td>Purchases of energy, water and materials</td>
</tr>
<tr>
<td>PURENOOTH</td>
<td>Purchases of energy and other goods for own consumption</td>
</tr>
<tr>
<td>PURCOTH</td>
<td>Purchases of other goods and materials for own consumption</td>
</tr>
<tr>
<td>PURESALE</td>
<td>Purchases of goods bought for resale</td>
</tr>
<tr>
<td>PURHIRE</td>
<td>Payments of hiring, leasing or renting</td>
</tr>
<tr>
<td>PURINS</td>
<td>Commercial insurance premiums paid</td>
</tr>
<tr>
<td>PURTRANS</td>
<td>Purchases of road transport services</td>
</tr>
<tr>
<td>PURTELE</td>
<td>Purchases of telecommunication services</td>
</tr>
<tr>
<td>PURCOMP</td>
<td>Purchases of computer and related services</td>
</tr>
<tr>
<td>PURADV</td>
<td>Purchases of advertising and marketing</td>
</tr>
<tr>
<td>PUROTHSE</td>
<td>Other services purchased</td>
</tr>
<tr>
<td>PUROTHAL</td>
<td>All other purchases of goods and services</td>
</tr>
<tr>
<td>PURTOT</td>
<td>Total purchases of goods and services</td>
</tr>
<tr>
<td>TAXRATES</td>
<td>Amounts paid for national non-domestic rates</td>
</tr>
<tr>
<td>TAXDUTY</td>
<td>Amounts paid for export duty</td>
</tr>
<tr>
<td>TAXOTHE</td>
<td>Other amounts paid for taxes and levies</td>
</tr>
<tr>
<td>TAXOTHD</td>
<td>Other amounts for taxes and levies excluding duty</td>
</tr>
<tr>
<td>TAXTOT</td>
<td>Total taxes paid</td>
</tr>
<tr>
<td>STOCKBEG</td>
<td>Value of stocks held at beginning of year</td>
</tr>
<tr>
<td>STOCKEND</td>
<td>Value of stocks held at end of year</td>
</tr>
<tr>
<td>ASSACQ</td>
<td>Total cost of all capital assets acquired</td>
</tr>
<tr>
<td>ASSDISP</td>
<td>Total proceeds from capital asset disposal</td>
</tr>
<tr>
<td>CAPWORK</td>
<td>Value of work of a capital nature</td>
</tr>
<tr>
<td>EMPLOY</td>
<td>Total number of employees</td>
</tr>
<tr>
<td>TURNREG</td>
<td>Register turnover</td>
</tr>
<tr>
<td>EMPREG</td>
<td>Employment size group from register: 1 = 0 to 9 employees, 2 = 10 to 19 employees, 3 = 20 to 49 employees, 4 = 50 to 99 employees, 5 = 100 to 249 employees, 6 = 250 or more employees</td>
</tr>
<tr>
<td>FORMTYPE</td>
<td>1 = long form, 2 = short form</td>
</tr>
</tbody>
</table>
3.2 Numerical Results

3.2.1 Tree Fitting

WAID assumes all covariates are categorical. Consequently, before building a WAID tree using TURNREG as a covariate, this variable was categorized into its 100 percentile classes. Furthermore, since the distribution of TURNOVER is highly skewed (see Figure 1) the tree was built using log(TURNOVER + 1) as the response variable. The addition of one to the observed TURNOVER value was to ensure that all non-missing cases, including the three businesses with zero values for TURNOVER, were used in building the tree.

Figure 2 shows the “nodemean tree” defined by WAID for non-missing values of log(TURNOVER + 1) from the error and missing contaminated dataset sec197(y3). This plot shows the robust means of the log-transformed response variable for the nodes defined at each split of the tree-fitting process. The tree was fitted using a maximum of 50 terminal nodes, with robust weights defined using Tukey's Biweight weighting scheme. No nodes containing less than 5 cases were allowed to be created in the fitting process. A maximum of 100 iterations were used in the iterative procedure rlm() that calculates the robust means shown in this plot. Also, both covariates TURNREG and EMPREG were declared as "monotone" in the tree-fitting process. In practice, this means that any splits defined using the categories of these variables must be in terms of contiguous categories, i.e. into subgroups defined by X ≤ x and X > x. Also, virtually all the splits in the tree were determined by TURNREG, with only the 17th split determined by EMPREG. This shows clearly on the nodemean plot.

For comparison, Figure 3 shows the corresponding WAID tree defined by the "true" values of TURNOVER in the file sec197(true). It is clear that the outliers and errors in the dataset sec197(y3) have little or no effect on the robust tree structure, as one would expect.
Figure 2: Robust node-mean tree for y3 values of \( \log(TURNOVER+1) \)
Figure 3: Robust node mean tree for true values of $\log(\text{TURNOVER}+1)$
3.2.2 Finding an Optimal Weight Threshold for TURNOVER

In Table 2 we illustrate the error detection performance of the robust regression tree displayed in Figure 2. For a range of values of a threshold $w$ this table shows:

- The number $N_{\text{out}}$ of outliers identified using that threshold
- The number $N_{\text{error}}$ of errors identified using that threshold. There are a total of 241 errors in this dataset.
- The number $N_{\text{sig}}$ of "significant" errors contained in these identified errors, where a "significant" error is one whose relative difference from its corresponding true value is greater than one. There are a total of 208 such significant errors in this dataset.
- The proportion $R_1$ of errors contained in the identified outliers
- The proportion $R_{\text{sig}}$ of significant errors contained in the identified outliers.
- The proportion $R_2$ of "non-errors" identified as outliers at this value of $w$
- The value $R_1(1-R_2)$

For TURNOVER we see that the optimal threshold (in terms of minimising $R_1(1-R_2)$) then lies between 0.015 and 0.016. These values are highlighted in Table 2. Observe that at this optimum over 80% (90%) of errors (significant errors) are detected. The 3% of identified outliers that are not errors are essentially "true outliers", which, in normal survey processing, would definitely need to be queried.
Table 2 Outlier and error detection performance for different threshold values

<table>
<thead>
<tr>
<th></th>
<th>N_{\text{out}}</th>
<th>N_{\text{error}}</th>
<th>N_{\text{sig}}</th>
<th>R_1</th>
<th>R_{\text{sig}}</th>
<th>R_2</th>
<th>R_1(1-R_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>0.145</td>
<td>0.168</td>
<td>0.000</td>
<td>0.145228</td>
</tr>
<tr>
<td>0.002</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>0.270</td>
<td>0.312</td>
<td>0.000</td>
<td>0.269709</td>
</tr>
<tr>
<td>0.003</td>
<td>83</td>
<td>82</td>
<td>82</td>
<td>0.340</td>
<td>0.394</td>
<td>0.012</td>
<td>0.336150</td>
</tr>
<tr>
<td>0.004</td>
<td>101</td>
<td>100</td>
<td>100</td>
<td>0.415</td>
<td>0.481</td>
<td>0.010</td>
<td>0.410829</td>
</tr>
<tr>
<td>0.005</td>
<td>121</td>
<td>120</td>
<td>120</td>
<td>0.498</td>
<td>0.577</td>
<td>0.008</td>
<td>0.493810</td>
</tr>
<tr>
<td>0.006</td>
<td>145</td>
<td>142</td>
<td>142</td>
<td>0.589</td>
<td>0.683</td>
<td>0.021</td>
<td>0.577021</td>
</tr>
<tr>
<td>0.007</td>
<td>163</td>
<td>160</td>
<td>159</td>
<td>0.664</td>
<td>0.764</td>
<td>0.018</td>
<td>0.651681</td>
</tr>
<tr>
<td>0.008</td>
<td>172</td>
<td>169</td>
<td>168</td>
<td>0.701</td>
<td>0.808</td>
<td>0.017</td>
<td>0.689014</td>
</tr>
<tr>
<td>0.009</td>
<td>186</td>
<td>182</td>
<td>180</td>
<td>0.755</td>
<td>0.865</td>
<td>0.022</td>
<td>0.738946</td>
</tr>
<tr>
<td>0.010</td>
<td>189</td>
<td>185</td>
<td>183</td>
<td>0.768</td>
<td>0.880</td>
<td>0.021</td>
<td>0.751389</td>
</tr>
<tr>
<td>0.011</td>
<td>193</td>
<td>189</td>
<td>187</td>
<td>0.784</td>
<td>0.899</td>
<td>0.021</td>
<td>0.767979</td>
</tr>
<tr>
<td>0.012</td>
<td>194</td>
<td>190</td>
<td>188</td>
<td>0.788</td>
<td>0.904</td>
<td>0.021</td>
<td>0.772126</td>
</tr>
<tr>
<td>0.013</td>
<td>197</td>
<td>192</td>
<td>190</td>
<td>0.797</td>
<td>0.913</td>
<td>0.025</td>
<td>0.776460</td>
</tr>
<tr>
<td>0.014</td>
<td>198</td>
<td>192</td>
<td>190</td>
<td>0.797</td>
<td>0.913</td>
<td>0.030</td>
<td>0.772539</td>
</tr>
<tr>
<td>0.015</td>
<td>199</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.030</td>
<td>0.776684</td>
</tr>
<tr>
<td>0.016</td>
<td>199</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.030</td>
<td>0.776684</td>
</tr>
<tr>
<td>0.017</td>
<td>202</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.045</td>
<td>0.765149</td>
</tr>
<tr>
<td>0.018</td>
<td>202</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.045</td>
<td>0.765149</td>
</tr>
<tr>
<td>0.019</td>
<td>203</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.049</td>
<td>0.761380</td>
</tr>
<tr>
<td>0.020</td>
<td>204</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.054</td>
<td>0.757648</td>
</tr>
<tr>
<td>0.021</td>
<td>207</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.068</td>
<td>0.746667</td>
</tr>
<tr>
<td>0.022</td>
<td>207</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.068</td>
<td>0.746667</td>
</tr>
<tr>
<td>0.023</td>
<td>207</td>
<td>193</td>
<td>191</td>
<td>0.801</td>
<td>0.918</td>
<td>0.068</td>
<td>0.746667</td>
</tr>
<tr>
<td>0.024</td>
<td>211</td>
<td>195</td>
<td>192</td>
<td>0.809</td>
<td>0.923</td>
<td>0.076</td>
<td>0.747773</td>
</tr>
<tr>
<td>0.025</td>
<td>212</td>
<td>196</td>
<td>193</td>
<td>0.813</td>
<td>0.928</td>
<td>0.075</td>
<td>0.751899</td>
</tr>
<tr>
<td>0.026</td>
<td>212</td>
<td>196</td>
<td>193</td>
<td>0.813</td>
<td>0.928</td>
<td>0.075</td>
<td>0.751899</td>
</tr>
<tr>
<td>0.027</td>
<td>212</td>
<td>196</td>
<td>193</td>
<td>0.813</td>
<td>0.928</td>
<td>0.075</td>
<td>0.751899</td>
</tr>
<tr>
<td>0.028</td>
<td>214</td>
<td>197</td>
<td>194</td>
<td>0.817</td>
<td>0.933</td>
<td>0.079</td>
<td>0.752492</td>
</tr>
<tr>
<td>0.029</td>
<td>214</td>
<td>197</td>
<td>194</td>
<td>0.817</td>
<td>0.933</td>
<td>0.079</td>
<td>0.752492</td>
</tr>
<tr>
<td>0.030</td>
<td>215</td>
<td>197</td>
<td>194</td>
<td>0.817</td>
<td>0.933</td>
<td>0.084</td>
<td>0.748992</td>
</tr>
<tr>
<td>0.035</td>
<td>221</td>
<td>198</td>
<td>194</td>
<td>0.822</td>
<td>0.933</td>
<td>0.104</td>
<td>0.736073</td>
</tr>
<tr>
<td>0.040</td>
<td>225</td>
<td>202</td>
<td>194</td>
<td>0.838</td>
<td>0.933</td>
<td>0.102</td>
<td>0.752494</td>
</tr>
<tr>
<td>0.045</td>
<td>228</td>
<td>202</td>
<td>194</td>
<td>0.838</td>
<td>0.933</td>
<td>0.114</td>
<td>0.742593</td>
</tr>
<tr>
<td>0.050</td>
<td>233</td>
<td>203</td>
<td>195</td>
<td>0.842</td>
<td>0.938</td>
<td>0.129</td>
<td>0.733870</td>
</tr>
<tr>
<td>0.055</td>
<td>242</td>
<td>208</td>
<td>199</td>
<td>0.863</td>
<td>0.957</td>
<td>0.140</td>
<td>0.741813</td>
</tr>
<tr>
<td>0.060</td>
<td>248</td>
<td>211</td>
<td>201</td>
<td>0.876</td>
<td>0.966</td>
<td>0.149</td>
<td>0.744897</td>
</tr>
<tr>
<td>0.065</td>
<td>252</td>
<td>211</td>
<td>201</td>
<td>0.876</td>
<td>0.966</td>
<td>0.163</td>
<td>0.733073</td>
</tr>
<tr>
<td>0.070</td>
<td>259</td>
<td>212</td>
<td>201</td>
<td>0.880</td>
<td>0.966</td>
<td>0.181</td>
<td>0.720037</td>
</tr>
<tr>
<td>0.075</td>
<td>265</td>
<td>215</td>
<td>204</td>
<td>0.892</td>
<td>0.981</td>
<td>0.189</td>
<td>0.723792</td>
</tr>
<tr>
<td>0.080</td>
<td>275</td>
<td>216</td>
<td>204</td>
<td>0.896</td>
<td>0.981</td>
<td>0.215</td>
<td>0.703976</td>
</tr>
<tr>
<td>0.085</td>
<td>292</td>
<td>217</td>
<td>205</td>
<td>0.900</td>
<td>0.986</td>
<td>0.257</td>
<td>0.669144</td>
</tr>
<tr>
<td>0.090</td>
<td>300</td>
<td>218</td>
<td>206</td>
<td>0.905</td>
<td>0.990</td>
<td>0.273</td>
<td>0.657317</td>
</tr>
<tr>
<td>0.095</td>
<td>309</td>
<td>220</td>
<td>206</td>
<td>0.913</td>
<td>0.990</td>
<td>0.288</td>
<td>0.649935</td>
</tr>
<tr>
<td>0.100</td>
<td>315</td>
<td>221</td>
<td>207</td>
<td>0.917</td>
<td>0.995</td>
<td>0.298</td>
<td>0.643364</td>
</tr>
</tbody>
</table>
In Figure 4 we show the behaviour of both $R_1$ and $R_1(1-R_2)$ as the threshold value $w$ changes. Here we see that if a more conservative approach to error detection is required, then the optimal threshold would be set higher in order to detect more errors, even if this leads to more "correct" values being identified as outliers. Based on an inspection of Figure 4 a reasonable conservative optimal threshold for TURNOVER is then 0.08, at which (see Table 2), nearly 90% (98%) of all errors (significant errors) are detected. The cost, of course, is that this leads to the identification of a much larger number of correct TURNOVER values as outliers (59 compared with 6). However, it is clear from inspection of the TURNOVER data that these values are ones that we would want to query anyway.

![Figure 4 Plot of $R_1(w)$ and $R_1(w)(1-R_2(w))$ for TURNOVER](image)

### 3.2.3 Imputing for Missing, Error and Outlier values in TURNOVER

There are 42 records with missing TURNOVER in the dataset sec197(y3). Furthermore, if we set the outlier detection threshold at $w = 0.02$ in the robust WAID tree constructed for the non-missing values in this dataset then there are 204 identified outliers, out of which 193 are errors. All these values can be imputed by identifying their terminal nodes and then replacing them by TURNOVER values donated by randomly selected "inliers" in the node, or by the robust estimates of the corresponding node means for TURNOVER. These two options are denoted "Random" imputation and "Mean" imputation in what follows. Note that the Random imputation option requires specification of what constitutes an inlier in a terminal node. The default option for this is observations with average tree weight of one or greater, and this is the approach that was taken to obtain the results set out below.
**Missing Data Imputation**

Table 3 shows the Reference number, TURNREG class, EMREG class, true value of log(TURNOVER+1) and corresponding imputed value under both Random and Mean imputation for the 42 records in sec197(y3) with missing values for TURNOVER. In the original scale, the unweighted mean error of imputation for these cases is 114 for Random imputation and -16 for Mean imputation. The corresponding unweighted root mean square error values are 1409 for Random imputation and 1189 for Mean imputation.

**Error Values Imputation**

Table 4 shows Reference number, TURNREG class, EMREG class, true value of log(TURNOVER+1) and corresponding imputed value under both Random and Mean imputation for the 193 records with errors in TURNOVER in sec197(y3) that were identified as outliers using a threshold of \( w = 0.02 \). In the original scale, the unweighted mean error of imputation for these cases is 60670 for Random imputation and 54506 for Mean imputation. The corresponding unweighted root mean square error values are 881109 for Random imputation and 858946 for Mean imputation. Inspection of these records showed that one of them (Reference number 17816) is clearly an outlier as well as an error. When this record is excluded, the unweighted mean error of imputation drops sharply to -2703 for Random imputation and to -7209 for Mean imputation. The corresponding unweighted root mean square error values are then 39787 for Random imputation and 60006 for Mean imputation.

"Correct" Outlier Imputation

Table 4 shows Reference number, TURNREG class, EMREG class, true value of log(TURNOVER+1) and corresponding imputed value under both Random and Mean imputation for the 11 records with correct values for TURNOVER in sec197(y3) that were identified as outliers using a threshold of \( w = 0.02 \). In the original scale, the unweighted mean error of imputation for these cases is 46518 for Random imputation and 46543 for Mean imputation. The corresponding unweighted root mean square error values are 89834 for Random imputation and 89882 for Mean imputation. Note that we expect to generate large mean errors and large root mean square errors for these imputations, since the comparisons in this case are with outlying true values of TURNOVER.
Table 3 Imputations for missing y3 values of log(TURNOVER+1)

<table>
<thead>
<tr>
<th>REF</th>
<th>TURNREG CLASS</th>
<th>EMPREG CLASS</th>
<th>TRUE VALUE</th>
<th>RANDOM IMPUTE</th>
<th>MEAN IMPUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>788</td>
<td>97</td>
<td>5</td>
<td>10.7011</td>
<td>10.6339</td>
<td>10.5759</td>
</tr>
<tr>
<td>4064</td>
<td>88</td>
<td>4</td>
<td>7.9983</td>
<td>8.1173</td>
<td>8.0892</td>
</tr>
<tr>
<td>5403</td>
<td>96</td>
<td>6</td>
<td>9.9889</td>
<td>10.0297</td>
<td>10.1933</td>
</tr>
<tr>
<td>8348</td>
<td>93</td>
<td>5</td>
<td>8.8343</td>
<td>9.3270</td>
<td>9.1466</td>
</tr>
<tr>
<td>9816</td>
<td>36</td>
<td>1</td>
<td>5.0814</td>
<td>5.0304</td>
<td>5.0955</td>
</tr>
<tr>
<td>9844</td>
<td>54</td>
<td>2</td>
<td>5.5175</td>
<td>5.8406</td>
<td>5.7337</td>
</tr>
<tr>
<td>9866</td>
<td>78</td>
<td>2</td>
<td>7.3232</td>
<td>6.9147</td>
<td>7.0157</td>
</tr>
<tr>
<td>10141</td>
<td>85</td>
<td>2</td>
<td>7.5735</td>
<td>7.8034</td>
<td>7.7851</td>
</tr>
<tr>
<td>10781</td>
<td>86</td>
<td>3</td>
<td>7.9424</td>
<td>7.9306</td>
<td>7.7851</td>
</tr>
<tr>
<td>11164</td>
<td>71</td>
<td>1</td>
<td>6.3886</td>
<td>6.4265</td>
<td>6.5741</td>
</tr>
<tr>
<td>11670</td>
<td>38</td>
<td>1</td>
<td>5.0239</td>
<td>5.0938</td>
<td>5.0955</td>
</tr>
<tr>
<td>12058</td>
<td>9</td>
<td>1</td>
<td>4.0254</td>
<td>3.7612</td>
<td>4.0141</td>
</tr>
<tr>
<td>12757</td>
<td>40</td>
<td>1</td>
<td>4.9972</td>
<td>5.2149</td>
<td>5.1875</td>
</tr>
<tr>
<td>12793</td>
<td>24</td>
<td>1</td>
<td>4.8040</td>
<td>4.5643</td>
<td>4.6375</td>
</tr>
<tr>
<td>12902</td>
<td>2</td>
<td>1</td>
<td>5.6630</td>
<td>2.5649</td>
<td>3.2615</td>
</tr>
<tr>
<td>12995</td>
<td>33</td>
<td>1</td>
<td>5.3845</td>
<td>5.0814</td>
<td>4.8907</td>
</tr>
<tr>
<td>14185</td>
<td>43</td>
<td>1</td>
<td>5.3613</td>
<td>5.2523</td>
<td>5.3080</td>
</tr>
<tr>
<td>14198</td>
<td>78</td>
<td>3</td>
<td>7.3957</td>
<td>7.1253</td>
<td>7.0157</td>
</tr>
<tr>
<td>14843</td>
<td>63</td>
<td>2</td>
<td>6.4184</td>
<td>6.0730</td>
<td>6.1132</td>
</tr>
<tr>
<td>14953</td>
<td>44</td>
<td>1</td>
<td>5.2311</td>
<td>5.4638</td>
<td>5.3080</td>
</tr>
<tr>
<td>15086</td>
<td>74</td>
<td>2</td>
<td>6.8855</td>
<td>6.7154</td>
<td>6.7363</td>
</tr>
<tr>
<td>15494</td>
<td>19</td>
<td>1</td>
<td>4.5747</td>
<td>4.4427</td>
<td>4.4676</td>
</tr>
<tr>
<td>15809</td>
<td>42</td>
<td>1</td>
<td>5.0876</td>
<td>5.3033</td>
<td>5.3080</td>
</tr>
<tr>
<td>15826</td>
<td>42</td>
<td>1</td>
<td>5.3327</td>
<td>5.3083</td>
<td>5.3080</td>
</tr>
<tr>
<td>16014</td>
<td>60</td>
<td>1</td>
<td>6.0753</td>
<td>6.1203</td>
<td>6.0088</td>
</tr>
<tr>
<td>16281</td>
<td>12</td>
<td>1</td>
<td>4.2627</td>
<td>4.1109</td>
<td>4.2034</td>
</tr>
<tr>
<td>16334</td>
<td>43</td>
<td>1</td>
<td>5.2311</td>
<td>5.4249</td>
<td>5.3080</td>
</tr>
<tr>
<td>16685</td>
<td>19</td>
<td>1</td>
<td>3.4965</td>
<td>4.3944</td>
<td>4.4676</td>
</tr>
<tr>
<td>16806</td>
<td>81</td>
<td>3</td>
<td>7.5569</td>
<td>7.1546</td>
<td>7.2567</td>
</tr>
<tr>
<td>16894</td>
<td>65</td>
<td>1</td>
<td>6.2025</td>
<td>6.2166</td>
<td>6.2344</td>
</tr>
<tr>
<td>17113</td>
<td>20</td>
<td>1</td>
<td>2.5649</td>
<td>4.3944</td>
<td>4.5513</td>
</tr>
<tr>
<td>17201</td>
<td>57</td>
<td>1</td>
<td>5.8021</td>
<td>5.8861</td>
<td>5.8261</td>
</tr>
<tr>
<td>17222</td>
<td>24</td>
<td>1</td>
<td>5.4931</td>
<td>4.7185</td>
<td>4.6375</td>
</tr>
<tr>
<td>17339</td>
<td>11</td>
<td>3</td>
<td>4.1744</td>
<td>4.2341</td>
<td>4.2034</td>
</tr>
<tr>
<td>17681</td>
<td>19</td>
<td>1</td>
<td>4.5109</td>
<td>4.2627</td>
<td>4.4676</td>
</tr>
<tr>
<td>17794</td>
<td>69</td>
<td>2</td>
<td>6.7405</td>
<td>6.3835</td>
<td>6.3690</td>
</tr>
<tr>
<td>18084</td>
<td>59</td>
<td>1</td>
<td>5.8608</td>
<td>5.9480</td>
<td>6.0088</td>
</tr>
<tr>
<td>18107</td>
<td>67</td>
<td>2</td>
<td>6.2860</td>
<td>6.2166</td>
<td>6.3690</td>
</tr>
<tr>
<td>18188</td>
<td>96</td>
<td>4</td>
<td>10.1914</td>
<td>9.9637</td>
<td>10.1933</td>
</tr>
<tr>
<td>18894</td>
<td>24</td>
<td>1</td>
<td>4.3041</td>
<td>4.4427</td>
<td>4.6375</td>
</tr>
<tr>
<td>18963</td>
<td>16</td>
<td>1</td>
<td>4.4773</td>
<td>4.2767</td>
<td>4.3603</td>
</tr>
</tbody>
</table>
Table 4 Imputations for identified errors in y3 values of log(TURNOVER+1)

<table>
<thead>
<tr>
<th>REF</th>
<th>TURNREG</th>
<th>EMPREG</th>
<th>TRUE VALUE</th>
<th>RANDOM IMPUTE</th>
<th>MEAN IMPUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>284</td>
<td>95</td>
<td>6</td>
<td>9.5599</td>
<td>9.5932</td>
<td>9.7412</td>
</tr>
<tr>
<td>303</td>
<td>98</td>
<td>6</td>
<td>10.9272</td>
<td>11.0953</td>
<td>11.1149</td>
</tr>
<tr>
<td>709</td>
<td>98</td>
<td>6</td>
<td>10.8215</td>
<td>10.8881</td>
<td>11.1149</td>
</tr>
<tr>
<td>775</td>
<td>96</td>
<td>5</td>
<td>9.9404</td>
<td>10.1301</td>
<td>10.1933</td>
</tr>
<tr>
<td>1315</td>
<td>96</td>
<td>6</td>
<td>10.3678</td>
<td>10.2942</td>
<td>10.1933</td>
</tr>
<tr>
<td>2497</td>
<td>90</td>
<td>4</td>
<td>8.4000</td>
<td>8.4029</td>
<td>8.4258</td>
</tr>
<tr>
<td>2902</td>
<td>87</td>
<td>3</td>
<td>8.3195</td>
<td>8.2022</td>
<td>8.0892</td>
</tr>
<tr>
<td>3111</td>
<td>80</td>
<td>4</td>
<td>7.4448</td>
<td>7.1709</td>
<td>7.2567</td>
</tr>
<tr>
<td>3097</td>
<td>100</td>
<td>6</td>
<td>12.7652</td>
<td>13.6733</td>
<td>13.6365</td>
</tr>
<tr>
<td>3489</td>
<td>96</td>
<td>6</td>
<td>10.1254</td>
<td>10.2583</td>
<td>10.1933</td>
</tr>
<tr>
<td>3572</td>
<td>71</td>
<td>2</td>
<td>6.4118</td>
<td>6.5610</td>
<td>6.5741</td>
</tr>
<tr>
<td>3676</td>
<td>84</td>
<td>4</td>
<td>7.5585</td>
<td>7.6971</td>
<td>7.5587</td>
</tr>
<tr>
<td>4052</td>
<td>98</td>
<td>6</td>
<td>10.7908</td>
<td>11.1701</td>
<td>11.1149</td>
</tr>
<tr>
<td>4370</td>
<td>100</td>
<td>6</td>
<td>16.3603</td>
<td>13.1454</td>
<td>13.6365</td>
</tr>
<tr>
<td>5803</td>
<td>94</td>
<td>5</td>
<td>9.2641</td>
<td>9.5485</td>
<td>9.4123</td>
</tr>
<tr>
<td>5937</td>
<td>91</td>
<td>5</td>
<td>8.8471</td>
<td>8.8051</td>
<td>8.8157</td>
</tr>
<tr>
<td>6155</td>
<td>76</td>
<td>4</td>
<td>6.9187</td>
<td>6.8967</td>
<td>6.8667</td>
</tr>
<tr>
<td>6326</td>
<td>97</td>
<td>6</td>
<td>10.5906</td>
<td>10.6339</td>
<td>10.5759</td>
</tr>
<tr>
<td>6629</td>
<td>79</td>
<td>3</td>
<td>7.1693</td>
<td>7.3784</td>
<td>7.1014</td>
</tr>
<tr>
<td>7009</td>
<td>99</td>
<td>6</td>
<td>11.9795</td>
<td>12.1266</td>
<td>12.0154</td>
</tr>
<tr>
<td>7022</td>
<td>100</td>
<td>6</td>
<td>12.7164</td>
<td>13.0603</td>
<td>13.6365</td>
</tr>
<tr>
<td>7125</td>
<td>97</td>
<td>6</td>
<td>10.3577</td>
<td>10.4862</td>
<td>10.5759</td>
</tr>
<tr>
<td>7912</td>
<td>69</td>
<td>3</td>
<td>6.3953</td>
<td>6.3986</td>
<td>6.3690</td>
</tr>
<tr>
<td>8593</td>
<td>100</td>
<td>6</td>
<td>12.8892</td>
<td>12.6550</td>
<td>13.6365</td>
</tr>
<tr>
<td>9651</td>
<td>30</td>
<td>1</td>
<td>4.8283</td>
<td>4.8828</td>
<td>4.8907</td>
</tr>
<tr>
<td>9714</td>
<td>53</td>
<td>1</td>
<td>5.6768</td>
<td>5.7652</td>
<td>5.5979</td>
</tr>
<tr>
<td>9754</td>
<td>48</td>
<td>1</td>
<td>5.4510</td>
<td>5.3613</td>
<td>5.4603</td>
</tr>
<tr>
<td>9826</td>
<td>32</td>
<td>1</td>
<td>5.8608</td>
<td>4.8903</td>
<td>4.8907</td>
</tr>
<tr>
<td>9863</td>
<td>71</td>
<td>5</td>
<td>6.9460</td>
<td>6.5917</td>
<td>6.5741</td>
</tr>
<tr>
<td>9938</td>
<td>46</td>
<td>1</td>
<td>4.8752</td>
<td>5.1475</td>
<td>5.3080</td>
</tr>
<tr>
<td>10215</td>
<td>17</td>
<td>1</td>
<td>4.5326</td>
<td>4.3041</td>
<td>4.3603</td>
</tr>
<tr>
<td>10420</td>
<td>44</td>
<td>1</td>
<td>5.3799</td>
<td>5.1705</td>
<td>5.3080</td>
</tr>
<tr>
<td>10532</td>
<td>91</td>
<td>4</td>
<td>8.5960</td>
<td>8.7751</td>
<td>8.8157</td>
</tr>
<tr>
<td>10559</td>
<td>36</td>
<td>1</td>
<td>5.1417</td>
<td>4.9127</td>
<td>5.0955</td>
</tr>
<tr>
<td>10635</td>
<td>22</td>
<td>1</td>
<td>4.5643</td>
<td>4.4067</td>
<td>4.5513</td>
</tr>
<tr>
<td>10763</td>
<td>85</td>
<td>4</td>
<td>7.6714</td>
<td>7.7820</td>
<td>7.7851</td>
</tr>
<tr>
<td>10829</td>
<td>93</td>
<td>4</td>
<td>9.2265</td>
<td>9.0508</td>
<td>9.1466</td>
</tr>
<tr>
<td>10893</td>
<td>62</td>
<td>2</td>
<td>5.7301</td>
<td>6.2324</td>
<td>6.1132</td>
</tr>
<tr>
<td>10903</td>
<td>50</td>
<td>2</td>
<td>5.5452</td>
<td>5.3566</td>
<td>5.5979</td>
</tr>
<tr>
<td>10992</td>
<td>41</td>
<td>1</td>
<td>5.0173</td>
<td>5.1874</td>
<td>5.1875</td>
</tr>
<tr>
<td>11089</td>
<td>90</td>
<td>4</td>
<td>5.3230</td>
<td>8.3772</td>
<td>8.4258</td>
</tr>
<tr>
<td>11119</td>
<td>12</td>
<td>1</td>
<td>4.2627</td>
<td>4.2485</td>
<td>4.2034</td>
</tr>
<tr>
<td>11143</td>
<td>93</td>
<td>5</td>
<td>8.8557</td>
<td>8.9274</td>
<td>9.1466</td>
</tr>
<tr>
<td>11174</td>
<td>29</td>
<td>1</td>
<td>4.2627</td>
<td>4.7005</td>
<td>4.7833</td>
</tr>
<tr>
<td>REF</td>
<td>TURNREG CLASS</td>
<td>EMPREG CLASS</td>
<td>TRUE VALUE</td>
<td>RANDOM IMPUTE</td>
<td>MEAN IMPUTE</td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
<td>--------------</td>
<td>------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>11225</td>
<td>73</td>
<td>1</td>
<td>6.8438</td>
<td>6.9930</td>
<td>6.7356</td>
</tr>
<tr>
<td>11269</td>
<td>67</td>
<td>1</td>
<td>6.3630</td>
<td>6.2823</td>
<td>6.3690</td>
</tr>
<tr>
<td>11374</td>
<td>17</td>
<td>1</td>
<td>4.3944</td>
<td>4.2627</td>
<td>4.3603</td>
</tr>
<tr>
<td>11382</td>
<td>89</td>
<td>4</td>
<td>8.2340</td>
<td>8.3624</td>
<td>8.4258</td>
</tr>
<tr>
<td>11400</td>
<td>52</td>
<td>1</td>
<td>5.6971</td>
<td>5.5568</td>
<td>5.5979</td>
</tr>
<tr>
<td>11418</td>
<td>51</td>
<td>1</td>
<td>5.6384</td>
<td>5.5872</td>
<td>5.5979</td>
</tr>
<tr>
<td>11419</td>
<td>77</td>
<td>1</td>
<td>6.9903</td>
<td>7.1381</td>
<td>7.0157</td>
</tr>
<tr>
<td>11476</td>
<td>90</td>
<td>4</td>
<td>8.4532</td>
<td>8.3347</td>
<td>8.4258</td>
</tr>
<tr>
<td>11485</td>
<td>13</td>
<td>1</td>
<td>4.0604</td>
<td>4.3041</td>
<td>4.2559</td>
</tr>
<tr>
<td>11581</td>
<td>37</td>
<td>1</td>
<td>4.9488</td>
<td>5.0562</td>
<td>5.0955</td>
</tr>
<tr>
<td>11631</td>
<td>72</td>
<td>1</td>
<td>6.3456</td>
<td>6.5352</td>
<td>6.5741</td>
</tr>
<tr>
<td>11695</td>
<td>71</td>
<td>3</td>
<td>6.0822</td>
<td>6.5058</td>
<td>6.5741</td>
</tr>
<tr>
<td>11710</td>
<td>20</td>
<td>1</td>
<td>4.1897</td>
<td>4.5326</td>
<td>4.5513</td>
</tr>
<tr>
<td>11776</td>
<td>36</td>
<td>1</td>
<td>4.8828</td>
<td>5.2983</td>
<td>5.0955</td>
</tr>
<tr>
<td>11784</td>
<td>40</td>
<td>1</td>
<td>5.0562</td>
<td>5.3230</td>
<td>5.1875</td>
</tr>
<tr>
<td>11814</td>
<td>74</td>
<td>2</td>
<td>5.9269</td>
<td>6.4313</td>
<td>6.7356</td>
</tr>
<tr>
<td>11913</td>
<td>99</td>
<td>6</td>
<td>12.3400</td>
<td>11.9281</td>
<td>12.0154</td>
</tr>
<tr>
<td>11924</td>
<td>8</td>
<td>1</td>
<td>2.8904</td>
<td>3.9703</td>
<td>4.0141</td>
</tr>
<tr>
<td>11992</td>
<td>58</td>
<td>1</td>
<td>5.8551</td>
<td>5.8111</td>
<td>5.9028</td>
</tr>
<tr>
<td>12116</td>
<td>20</td>
<td>1</td>
<td>4.6250</td>
<td>4.6151</td>
<td>4.5513</td>
</tr>
<tr>
<td>12172</td>
<td>38</td>
<td>1</td>
<td>4.9345</td>
<td>5.1985</td>
<td>5.0955</td>
</tr>
<tr>
<td>12193</td>
<td>43</td>
<td>2</td>
<td>7.2591</td>
<td>5.3279</td>
<td>5.3080</td>
</tr>
<tr>
<td>12305</td>
<td>64</td>
<td>2</td>
<td>6.4102</td>
<td>6.3190</td>
<td>6.2344</td>
</tr>
<tr>
<td>12351</td>
<td>59</td>
<td>1</td>
<td>6.0913</td>
<td>6.0426</td>
<td>6.0088</td>
</tr>
<tr>
<td>12372</td>
<td>17</td>
<td>1</td>
<td>4.3820</td>
<td>4.4543</td>
<td>4.3603</td>
</tr>
<tr>
<td>12420</td>
<td>11</td>
<td>1</td>
<td>4.3944</td>
<td>4.3694</td>
<td>4.2034</td>
</tr>
<tr>
<td>12425</td>
<td>72</td>
<td>3</td>
<td>6.3699</td>
<td>6.6412</td>
<td>6.5741</td>
</tr>
<tr>
<td>12428</td>
<td>54</td>
<td>1</td>
<td>5.6904</td>
<td>5.5835</td>
<td>5.7337</td>
</tr>
<tr>
<td>12437</td>
<td>42</td>
<td>1</td>
<td>5.5255</td>
<td>5.2883</td>
<td>5.3080</td>
</tr>
<tr>
<td>12460</td>
<td>77</td>
<td>4</td>
<td>7.0707</td>
<td>7.1131</td>
<td>7.0157</td>
</tr>
<tr>
<td>12530</td>
<td>41</td>
<td>1</td>
<td>4.4308</td>
<td>5.2470</td>
<td>5.1875</td>
</tr>
<tr>
<td>12685</td>
<td>36</td>
<td>1</td>
<td>4.7362</td>
<td>5.2311</td>
<td>5.0955</td>
</tr>
<tr>
<td>12701</td>
<td>51</td>
<td>2</td>
<td>5.7104</td>
<td>5.4806</td>
<td>5.5979</td>
</tr>
<tr>
<td>12742</td>
<td>30</td>
<td>1</td>
<td>4.7622</td>
<td>4.7274</td>
<td>4.8907</td>
</tr>
<tr>
<td>12935</td>
<td>39</td>
<td>1</td>
<td>5.4723</td>
<td>5.3706</td>
<td>5.1875</td>
</tr>
<tr>
<td>12970</td>
<td>31</td>
<td>1</td>
<td>4.8828</td>
<td>4.6821</td>
<td>4.8907</td>
</tr>
<tr>
<td>12988</td>
<td>14</td>
<td>1</td>
<td>4.4067</td>
<td>4.3307</td>
<td>4.3603</td>
</tr>
<tr>
<td>13045</td>
<td>49</td>
<td>2</td>
<td>5.0434</td>
<td>5.5568</td>
<td>5.5979</td>
</tr>
<tr>
<td>13150</td>
<td>79</td>
<td>3</td>
<td>7.2542</td>
<td>7.1317</td>
<td>7.1014</td>
</tr>
<tr>
<td>13194</td>
<td>56</td>
<td>1</td>
<td>5.8111</td>
<td>5.7398</td>
<td>5.8261</td>
</tr>
<tr>
<td>13239</td>
<td>3</td>
<td>1</td>
<td>3.4340</td>
<td>3.5264</td>
<td>3.5262</td>
</tr>
<tr>
<td>13251</td>
<td>83</td>
<td>2</td>
<td>7.5126</td>
<td>7.5000</td>
<td>7.5587</td>
</tr>
<tr>
<td>13333</td>
<td>59</td>
<td>2</td>
<td>5.5491</td>
<td>5.8665</td>
<td>6.0088</td>
</tr>
<tr>
<td>13392</td>
<td>15</td>
<td>1</td>
<td>3.7377</td>
<td>4.3175</td>
<td>4.3603</td>
</tr>
<tr>
<td>13449</td>
<td>5</td>
<td>1</td>
<td>3.8067</td>
<td>3.9318</td>
<td>3.7430</td>
</tr>
<tr>
<td>13562</td>
<td>41</td>
<td>1</td>
<td>5.0304</td>
<td>5.1240</td>
<td>5.1875</td>
</tr>
<tr>
<td>REF</td>
<td>TURNREG CLASS</td>
<td>REF</td>
<td>EMPREG CLASS</td>
<td>TRUE VALUE</td>
<td>RANDOM</td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td>-----</td>
<td>--------------</td>
<td>------------</td>
<td>--------</td>
</tr>
<tr>
<td>13599</td>
<td>15</td>
<td>1</td>
<td>1</td>
<td>4.2485</td>
<td>4.4308</td>
</tr>
<tr>
<td>13680</td>
<td>78</td>
<td>3</td>
<td>1</td>
<td>6.9976</td>
<td>6.9344</td>
</tr>
<tr>
<td>13861</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>4.5643</td>
<td>4.6151</td>
</tr>
<tr>
<td>13894</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2.8904</td>
<td>2.7081</td>
</tr>
<tr>
<td>13964</td>
<td>64</td>
<td>1</td>
<td>1</td>
<td>6.2166</td>
<td>6.1675</td>
</tr>
<tr>
<td>14086</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td>4.4886</td>
<td>4.3041</td>
</tr>
<tr>
<td>14131</td>
<td>53</td>
<td>2</td>
<td>1</td>
<td>5.7203</td>
<td>5.6454</td>
</tr>
<tr>
<td>14188</td>
<td>49</td>
<td>1</td>
<td>1</td>
<td>5.5607</td>
<td>5.6490</td>
</tr>
<tr>
<td>14225</td>
<td>51</td>
<td>1</td>
<td>1</td>
<td>5.7398</td>
<td>5.5491</td>
</tr>
<tr>
<td>14379</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>5.5759</td>
<td>3.8712</td>
</tr>
<tr>
<td>14451</td>
<td>73</td>
<td>1</td>
<td>1</td>
<td>6.0981</td>
<td>6.6958</td>
</tr>
<tr>
<td>14460</td>
<td>40</td>
<td>1</td>
<td>1</td>
<td>5.6021</td>
<td>5.1985</td>
</tr>
<tr>
<td>14473</td>
<td>28</td>
<td>1</td>
<td>1</td>
<td>4.9972</td>
<td>4.9836</td>
</tr>
<tr>
<td>14486</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3.5835</td>
<td>3.9703</td>
</tr>
<tr>
<td>14490</td>
<td>91</td>
<td>4</td>
<td>1</td>
<td>9.0101</td>
<td>8.7178</td>
</tr>
<tr>
<td>14496</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3.4657</td>
<td>3.4965</td>
</tr>
<tr>
<td>14535</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>5.7071</td>
<td>5.5452</td>
</tr>
<tr>
<td>14564</td>
<td>46</td>
<td>1</td>
<td>1</td>
<td>5.1180</td>
<td>5.2523</td>
</tr>
<tr>
<td>14649</td>
<td>85</td>
<td>3</td>
<td>1</td>
<td>7.6123</td>
<td>7.7341</td>
</tr>
<tr>
<td>14699</td>
<td>73</td>
<td>2</td>
<td>1</td>
<td>6.6214</td>
<td>6.7534</td>
</tr>
<tr>
<td>14713</td>
<td>55</td>
<td>1</td>
<td>1</td>
<td>5.8319</td>
<td>6.0014</td>
</tr>
<tr>
<td>14728</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>4.1431</td>
<td>4.0431</td>
</tr>
<tr>
<td>14847</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>3.9512</td>
<td>3.9120</td>
</tr>
<tr>
<td>14858</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td>4.4188</td>
<td>4.3944</td>
</tr>
<tr>
<td>14901</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8.1259</td>
<td>3.2189</td>
</tr>
<tr>
<td>14961</td>
<td>23</td>
<td>1</td>
<td>1</td>
<td>4.5951</td>
<td>4.5850</td>
</tr>
<tr>
<td>15003</td>
<td>85</td>
<td>2</td>
<td>1</td>
<td>7.7437</td>
<td>7.7231</td>
</tr>
<tr>
<td>15007</td>
<td>61</td>
<td>2</td>
<td>1</td>
<td>5.8636</td>
<td>6.1312</td>
</tr>
<tr>
<td>15017</td>
<td>37</td>
<td>1</td>
<td>1</td>
<td>5.0434</td>
<td>5.0499</td>
</tr>
<tr>
<td>15068</td>
<td>75</td>
<td>2</td>
<td>1</td>
<td>7.2284</td>
<td>6.9537</td>
</tr>
<tr>
<td>15072</td>
<td>25</td>
<td>1</td>
<td>1</td>
<td>4.0073</td>
<td>4.6052</td>
</tr>
<tr>
<td>15078</td>
<td>49</td>
<td>1</td>
<td>1</td>
<td>5.4337</td>
<td>5.6131</td>
</tr>
<tr>
<td>15094</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3.8501</td>
<td>4.1897</td>
</tr>
<tr>
<td>15222</td>
<td>59</td>
<td>1</td>
<td>1</td>
<td>5.8081</td>
<td>5.9965</td>
</tr>
<tr>
<td>15226</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>5.6384</td>
<td>5.4931</td>
</tr>
<tr>
<td>15239</td>
<td>79</td>
<td>3</td>
<td>1</td>
<td>6.5525</td>
<td>7.2218</td>
</tr>
<tr>
<td>15263</td>
<td>41</td>
<td>1</td>
<td>1</td>
<td>5.1818</td>
<td>5.2204</td>
</tr>
<tr>
<td>15276</td>
<td>40</td>
<td>1</td>
<td>1</td>
<td>4.6151</td>
<td>5.1120</td>
</tr>
<tr>
<td>15300</td>
<td>57</td>
<td>1</td>
<td>1</td>
<td>5.9940</td>
<td>5.8230</td>
</tr>
<tr>
<td>15333</td>
<td>60</td>
<td>1</td>
<td>1</td>
<td>5.8777</td>
<td>6.1924</td>
</tr>
<tr>
<td>15377</td>
<td>71</td>
<td>2</td>
<td>1</td>
<td>6.6567</td>
<td>6.5280</td>
</tr>
<tr>
<td>15462</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3.4965</td>
<td>3.5835</td>
</tr>
<tr>
<td>15501</td>
<td>26</td>
<td>1</td>
<td>1</td>
<td>4.1897</td>
<td>4.7536</td>
</tr>
<tr>
<td>15513</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>3.9703</td>
<td>4.2195</td>
</tr>
<tr>
<td>15541</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>4.6250</td>
<td>4.5951</td>
</tr>
<tr>
<td>15588</td>
<td>64</td>
<td>1</td>
<td>1</td>
<td>6.2364</td>
<td>6.1549</td>
</tr>
<tr>
<td>REF</td>
<td>TURNREG</td>
<td>EMPREG</td>
<td>TRUE VALUE</td>
<td>RANDOM IMPUTE</td>
<td>MEAN IMPUTE</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>--------</td>
<td>------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>15594</td>
<td>54</td>
<td>1</td>
<td>5.9940</td>
<td>5.8721</td>
<td>5.7337</td>
</tr>
<tr>
<td>15674</td>
<td>95</td>
<td>5</td>
<td>9.5331</td>
<td>9.7092</td>
<td>9.7412</td>
</tr>
<tr>
<td>15749</td>
<td>69</td>
<td>2</td>
<td>6.6606</td>
<td>6.3172</td>
<td>6.3690</td>
</tr>
<tr>
<td>15972</td>
<td>13</td>
<td>1</td>
<td>4.2047</td>
<td>4.3944</td>
<td>4.2559</td>
</tr>
<tr>
<td>16011</td>
<td>34</td>
<td>1</td>
<td>5.0938</td>
<td>5.0039</td>
<td>5.0040</td>
</tr>
<tr>
<td>16028</td>
<td>56</td>
<td>1</td>
<td>5.5607</td>
<td>6.0450</td>
<td>5.8261</td>
</tr>
<tr>
<td>16057</td>
<td>37</td>
<td>1</td>
<td>4.7536</td>
<td>5.0938</td>
<td>5.0955</td>
</tr>
<tr>
<td>16087</td>
<td>2</td>
<td>1</td>
<td>4.0431</td>
<td>4.1109</td>
<td>3.2615</td>
</tr>
<tr>
<td>16134</td>
<td>59</td>
<td>1</td>
<td>6.0426</td>
<td>6.0403</td>
<td>6.0088</td>
</tr>
<tr>
<td>16181</td>
<td>18</td>
<td>1</td>
<td>4.4998</td>
<td>4.3820</td>
<td>4.4676</td>
</tr>
<tr>
<td>16200</td>
<td>68</td>
<td>1</td>
<td>5.2149</td>
<td>6.4313</td>
<td>6.3690</td>
</tr>
<tr>
<td>16239</td>
<td>47</td>
<td>1</td>
<td>5.2983</td>
<td>5.4249</td>
<td>5.4603</td>
</tr>
<tr>
<td>16278</td>
<td>17</td>
<td>1</td>
<td>4.4067</td>
<td>4.3567</td>
<td>4.3603</td>
</tr>
<tr>
<td>16292</td>
<td>11</td>
<td>1</td>
<td>4.3307</td>
<td>4.4773</td>
<td>4.2034</td>
</tr>
<tr>
<td>16400</td>
<td>62</td>
<td>1</td>
<td>4.6728</td>
<td>6.2538</td>
<td>6.1132</td>
</tr>
<tr>
<td>16479</td>
<td>66</td>
<td>1</td>
<td>6.2166</td>
<td>6.1612</td>
<td>6.2344</td>
</tr>
<tr>
<td>16526</td>
<td>3</td>
<td>1</td>
<td>2.9444</td>
<td>3.4340</td>
<td>3.5262</td>
</tr>
<tr>
<td>16531</td>
<td>41</td>
<td>2</td>
<td>5.2523</td>
<td>5.2627</td>
<td>5.1875</td>
</tr>
<tr>
<td>16564</td>
<td>58</td>
<td>1</td>
<td>6.0137</td>
<td>5.8665</td>
<td>5.9028</td>
</tr>
<tr>
<td>16678</td>
<td>48</td>
<td>2</td>
<td>5.4116</td>
<td>5.6276</td>
<td>5.4603</td>
</tr>
<tr>
<td>16853</td>
<td>73</td>
<td>2</td>
<td>6.6958</td>
<td>6.9622</td>
<td>6.7356</td>
</tr>
<tr>
<td>16952</td>
<td>49</td>
<td>1</td>
<td>5.5658</td>
<td>5.4848</td>
<td>5.5979</td>
</tr>
<tr>
<td>16959</td>
<td>54</td>
<td>1</td>
<td>5.6384</td>
<td>5.7071</td>
<td>5.7337</td>
</tr>
<tr>
<td>16985</td>
<td>13</td>
<td>1</td>
<td>4.4659</td>
<td>4.2047</td>
<td>4.2559</td>
</tr>
<tr>
<td>17055</td>
<td>11</td>
<td>1</td>
<td>4.1897</td>
<td>4.0943</td>
<td>4.2034</td>
</tr>
<tr>
<td>17104</td>
<td>31</td>
<td>1</td>
<td>4.8752</td>
<td>5.0304</td>
<td>4.8907</td>
</tr>
<tr>
<td>17148</td>
<td>25</td>
<td>1</td>
<td>4.5539</td>
<td>4.6821</td>
<td>4.7053</td>
</tr>
<tr>
<td>17210</td>
<td>77</td>
<td>2</td>
<td>6.8298</td>
<td>7.2027</td>
<td>7.0157</td>
</tr>
<tr>
<td>17317</td>
<td>29</td>
<td>1</td>
<td>4.7622</td>
<td>4.8828</td>
<td>4.7833</td>
</tr>
<tr>
<td>17397</td>
<td>37</td>
<td>2</td>
<td>4.9904</td>
<td>4.8752</td>
<td>5.0955</td>
</tr>
<tr>
<td>17528</td>
<td>2</td>
<td>1</td>
<td>3.5264</td>
<td>4.1109</td>
<td>3.2615</td>
</tr>
<tr>
<td>17559</td>
<td>15</td>
<td>1</td>
<td>4.4308</td>
<td>4.5109</td>
<td>4.3603</td>
</tr>
<tr>
<td>17816</td>
<td>11</td>
<td>1</td>
<td>10.9331</td>
<td>4.0073</td>
<td>4.2034</td>
</tr>
<tr>
<td>17875</td>
<td>62</td>
<td>1</td>
<td>6.0638</td>
<td>6.0981</td>
<td>6.1132</td>
</tr>
<tr>
<td>17920</td>
<td>13</td>
<td>1</td>
<td>4.2767</td>
<td>4.3567</td>
<td>4.2559</td>
</tr>
<tr>
<td>18015</td>
<td>1</td>
<td>1</td>
<td>4.1271</td>
<td>2.4849</td>
<td>2.5510</td>
</tr>
<tr>
<td>18102</td>
<td>67</td>
<td>2</td>
<td>6.2344</td>
<td>6.3630</td>
<td>6.3690</td>
</tr>
<tr>
<td>18113</td>
<td>76</td>
<td>3</td>
<td>6.1862</td>
<td>6.9187</td>
<td>6.8667</td>
</tr>
<tr>
<td>18175</td>
<td>80</td>
<td>3</td>
<td>7.2862</td>
<td>7.2682</td>
<td>7.2567</td>
</tr>
<tr>
<td>18182</td>
<td>1</td>
<td>1</td>
<td>4.6151</td>
<td>1.9459</td>
<td>2.5510</td>
</tr>
<tr>
<td>18203</td>
<td>72</td>
<td>2</td>
<td>6.4599</td>
<td>6.5889</td>
<td>6.5741</td>
</tr>
<tr>
<td>18334</td>
<td>10</td>
<td>1</td>
<td>3.7842</td>
<td>4.1109</td>
<td>4.0913</td>
</tr>
<tr>
<td>18340</td>
<td>57</td>
<td>1</td>
<td>5.9687</td>
<td>5.9713</td>
<td>5.8261</td>
</tr>
<tr>
<td>18428</td>
<td>36</td>
<td>1</td>
<td>3.9120</td>
<td>5.3279</td>
<td>5.0955</td>
</tr>
<tr>
<td>18434</td>
<td>32</td>
<td>1</td>
<td>4.8283</td>
<td>4.7958</td>
<td>4.8907</td>
</tr>
<tr>
<td>18469</td>
<td>2</td>
<td>1</td>
<td>3.8286</td>
<td>2.5649</td>
<td>3.2615</td>
</tr>
</tbody>
</table>
### Table 5: Imputations for identified "correct" outliers in $y_3$ values of $\log(\text{TURNOVER}+1)$

<table>
<thead>
<tr>
<th>REF</th>
<th>TURNREG CLASS</th>
<th>EMPREG CLASS</th>
<th>TRUE VALUE</th>
<th>RANDOM IMPUTE</th>
<th>MEAN IMPUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>18478</td>
<td>71</td>
<td>3</td>
<td>6.6503</td>
<td>6.7696</td>
<td>6.5741</td>
</tr>
<tr>
<td>18535</td>
<td>41</td>
<td>2</td>
<td>5.2575</td>
<td>5.1648</td>
<td>5.1875</td>
</tr>
<tr>
<td>18554</td>
<td>68</td>
<td>2</td>
<td>6.3886</td>
<td>6.2672</td>
<td>6.3690</td>
</tr>
<tr>
<td>18696</td>
<td>59</td>
<td>1</td>
<td>5.9738</td>
<td>5.9713</td>
<td>6.0088</td>
</tr>
<tr>
<td>18697</td>
<td>17</td>
<td>1</td>
<td>4.8203</td>
<td>4.2767</td>
<td>4.3603</td>
</tr>
<tr>
<td>18707</td>
<td>40</td>
<td>1</td>
<td>5.3706</td>
<td>5.2204</td>
<td>5.1875</td>
</tr>
<tr>
<td>18720</td>
<td>34</td>
<td>1</td>
<td>4.8978</td>
<td>5.0689</td>
<td>5.0040</td>
</tr>
<tr>
<td>18770</td>
<td>91</td>
<td>5</td>
<td>8.9141</td>
<td>8.7038</td>
<td>8.8157</td>
</tr>
<tr>
<td>18798</td>
<td>42</td>
<td>1</td>
<td>5.2523</td>
<td>5.1705</td>
<td>5.3080</td>
</tr>
<tr>
<td>18861</td>
<td>50</td>
<td>1</td>
<td>5.4848</td>
<td>5.6240</td>
<td>5.5979</td>
</tr>
<tr>
<td>18886</td>
<td>88</td>
<td>2</td>
<td>7.9700</td>
<td>8.2825</td>
<td>8.0892</td>
</tr>
</tbody>
</table>

### 3.2.3 Finding the Optimal Weight Thresholds for other ABI Variables

The approach outlined in the previous subsection was applied to the other 1997 ABI total variables EMPTOTC, PURTOT, TAXTOT, ASSACQ, ASSDISP and EMPLOY. Below we discuss results for each of these variables in turn. In each case a robust WAID tree based on the register covariates TURNREG and EMPREG was fitted.
There are 41 missing values of EMPTOTC in the 1997 ABI dataset. Of the remaining 6058 values, 658 are zero. These values were not treated specially in fitting the robust WAID tree for this variable. Figure 5 shows the values of $R_1$ and $R_1(1-R_2)$ generated by this tree. We note that in this case the optimal and conservative choices for the optimal threshold $w$ are the same, corresponding to $w = 0.385$. At this value, 75% of errors in EMPTOTC are detected and 16% of identified outliers are correct values.

Figure 5 Plot of $R_1(w)$ and $R_1(w)(1-R_2(w))$ for EMPTOTC
There are 28 missing values of PURTOT in the 1997 ABI dataset. Of the remaining 6071 values, 5 are zero. These values were not treated specially in fitting the robust WAID tree for this variable. Figure 6 shows the values of $R_1$ and $R_1(1-R_2)$ generated by this tree. For this variable there is a clear distinction between the optimal and conservative values for the outlier detection threshold. The optimal threshold is $w = 0.018$, where 37% of errors in PURTOT are detected and 4% of identified outliers are correct values. The conservative threshold is $w = 0.067$, where 43% of errors are detected and 18% of identified outliers are correct values. This can be seen in Figure 5 where the increase in the rate of detection of errors slows down considerably after this conservative threshold value.

Figure 5 Plot of $R_1(w)$ and $R_1(w)(1-R_2(w))$ for PURTOT
TAXTOT

There are 45 missing values of TAXTOT in the 1997 ABI dataset. Of the remaining 6054 values, 435 are zero. These values were not treated specially in fitting the robust WAID tree for this variable. Figure 6 shows the values of $R_1$ and $R_1(1-R_2)$ generated by this tree. In this case there is a degree of uncertainty about the choice between the optimal and conservative choices for the weight threshold for outlier detection based on this tree. The optimal threshold is $w = 0.559$, where 68% of TAXTOT errors are detected, with 4% of identified outliers corresponding to true values. In contrast, at $w = 0.750$, 80% of errors are identified, with 39% of identified outliers corresponding to true values of TAXTOT.

Figure 6 Plot of $R_1(w)$ and $R_1(w)(1-R_2(w))$ for TAXTOT
There are 57 missing values of ASSACQ in the 1997 ABI dataset. In addition, there are 908 records with ASSACQ = -9. Both types of record were ignored when fitting the robust WAID tree for this variable. Of the remaining 5134 values, 2106 are zero. This high proportion of zero values is quite different from the preceding ABI total variables. On the basis that so many zero values cannot all be outliers, we therefore also excluded records with ASSACQ = 0 from the tree fitting process.

When zero values excluded, we obtain the plots for $R_1$ and $R_1(1-R_2)$ shown in Figure 7. Here the optimal threshold is $w = 0.565$, where 76% of non-zero errors are identified, with 7% of identified outliers corresponding to true values. We also see that there is no difference between the optimum and conservative thresholds in this case.

Since in practice some of the zero values in ASSACQ can be errors, it is necessary to have a strategy that can deal with these. However, since only four of the 248 errors in this variable corresponded to zero values in 1997, there seems little lost by ignoring these errors.

Figure 7 Plot of $R_1(w)$ and $R_1(w)(1-R_2(w))$ for positive values of ASSACQ
ASSDISP

There are 63 missing values of ASSDISP in the 1997 ABI dataset. In addition, there are 1389 records with ASSDISP = -9. Both types of record were ignored when fitting the robust WAI D tree for this variable. Of the remaining 4647 values, 3208 are zero. This very large number of zero values made it impossible to fit a robust WAI D tree that included them. The same argument as used with ASSACQ led to records with ASSDISP = 0 being excluded from the tree fitting process. Figure 8 shows the values of $R_1$ and $R_1(1-R_2)$ generated by these positive values of ASSDISP. Here the optimal and conservative thresholds are clearly identical, at $w = 0.773$. At this value 59% of errors are identified, with 16% of identified outliers corresponding to true values.

Note that, as with ASSACQ, the problem then becomes one of how to identify errors in the zero values of ASSDISP. However, since there are only 2 such cases in 1997, there does not seem to be too much lost by ignoring them.

Figure 8 Plot of $R_1(w)$ and $R_1(w)(1-R_2(w))$ for positive values of ASSDISP
This variable contained just 49 errors and 35 missing values in 1997. With missing values excluded, Figure 9 shows the values of $R_1$ and $R_1(1-R_2)$ generated by the robust tree defined by the remaining non-missing cases. Here again we see that optimal and conservative thresholds are identical, at $w = 0.312$. At this value 61% of errors are identified. However, then 69% of identified outliers also correspond to true values.

Figure 9 Plot of $R_1(w)$ and $R_1(w)(1-R_2(w))$ for EMPLOY
In theory a robust WAID tree for a particular ABI variable can be constructed using any or all of the other ABI variables, including the register variables TURNREG and EMPREG. It is therefore of interest to investigate the outlier and error detection performance of alternative robust WAID trees based on different covariate combinations. Table 6 shows the values of outlier and error detection performance measures (the same as in Table 2) generated by these trees. Both single covariate trees and multiple covariate trees were investigated, with "A" defining the tree based only on the EMPLOY variable, "B" defining the tree based on all the other ABI total variables (except the analysis variable) and "C" defining the tree based on all the ABI total variables plus the two register variables TURNREG and EMPREG. Note that when other ABI total variables are used as covariates these variables are used with their observed (y3) values, with missing values coded to -1.

The results in Table 6 show that of the three types of trees A, B, C, those based on EMPLOY alone (A) is invariably the better performer, with outlier/error detection performance similar to that of robust trees based on the register variables TURNREG and EMPREG. Furthermore, in almost all cases trees based on a "bigger" set of covariates perform worse (sometimes considerably worse) than trees based on a "smaller" set of covariates. This is an interesting result. However, since all trees were constrained to 50 terminal nodes, it may be due to the fact that more terminal nodes are required when constructing a tree based on multiple covariates. This is currently being investigated.
Table 6 Outlier identification performance of robust WAID trees based on different covariate combination. Here "A" denotes a tree built using EMPLOY as the only covariate, "B" denotes a tree built using the other total variables as the covariates, and "C" denotes a tree based on the register covariates TURNREG and EMPREG plus all the other total variables. Note that where another total variable is used as a covariate, it is categorized into its percentile groups, based on observed (y3) values and with missing values coded to -1. Note also that zero values were excluded when building the trees for ASSACQ and ASSDISP.

<table>
<thead>
<tr>
<th>y</th>
<th>TREE type</th>
<th>w</th>
<th>N_{out}</th>
<th>N_{error}</th>
<th>N_{sig}</th>
<th>R_1</th>
<th>R_{sig}</th>
<th>R_2</th>
<th>R_1(1-R_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TURNOVER</td>
<td>A</td>
<td>0.0307</td>
<td>196</td>
<td>190</td>
<td>187</td>
<td>0.7884</td>
<td>0.8990</td>
<td>0.0306</td>
<td>0.7642</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.0949</td>
<td>52</td>
<td>40</td>
<td>32</td>
<td>0.1660</td>
<td>0.1538</td>
<td>0.2308</td>
<td>0.1775</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0.1432</td>
<td>112</td>
<td>61</td>
<td>49</td>
<td>0.2531</td>
<td>0.2356</td>
<td>0.4554</td>
<td>0.1379</td>
</tr>
<tr>
<td>EMPTOTC</td>
<td>A</td>
<td>0.3635</td>
<td>275</td>
<td>251</td>
<td>245</td>
<td>0.7560</td>
<td>0.9176</td>
<td>0.0873</td>
<td>0.6900</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.6179</td>
<td>227</td>
<td>108</td>
<td>94</td>
<td>0.3253</td>
<td>0.3521</td>
<td>0.5242</td>
<td>0.1548</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0.4606</td>
<td>139</td>
<td>83</td>
<td>73</td>
<td>0.2500</td>
<td>0.2734</td>
<td>0.4029</td>
<td>0.1493</td>
</tr>
<tr>
<td>PURTOT</td>
<td>A</td>
<td>0.0487</td>
<td>245</td>
<td>234</td>
<td>231</td>
<td>0.3720</td>
<td>0.9094</td>
<td>0.0449</td>
<td>0.3553</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.2326</td>
<td>169</td>
<td>110</td>
<td>71</td>
<td>0.1749</td>
<td>0.2795</td>
<td>0.3491</td>
<td>0.1138</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0.2201</td>
<td>166</td>
<td>111</td>
<td>73</td>
<td>0.1765</td>
<td>0.2874</td>
<td>0.3313</td>
<td>0.1180</td>
</tr>
<tr>
<td>TAXTOT</td>
<td>A</td>
<td>0.3588</td>
<td>300</td>
<td>272</td>
<td>271</td>
<td>0.5643</td>
<td>0.6145</td>
<td>0.0933</td>
<td>0.5116</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.7255</td>
<td>503</td>
<td>247</td>
<td>241</td>
<td>0.5124</td>
<td>0.5465</td>
<td>0.5089</td>
<td>0.2516</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0.7246</td>
<td>502</td>
<td>248</td>
<td>243</td>
<td>0.5145</td>
<td>0.5510</td>
<td>0.5060</td>
<td>0.2542</td>
</tr>
<tr>
<td>ASSACQ</td>
<td>A</td>
<td>0.5590</td>
<td>202</td>
<td>186</td>
<td>186</td>
<td>0.7620</td>
<td>0.7980</td>
<td>0.0790</td>
<td>0.7019</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.4900</td>
<td>61</td>
<td>48</td>
<td>48</td>
<td>0.1970</td>
<td>0.2060</td>
<td>0.2130</td>
<td>0.1548</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0.5940</td>
<td>96</td>
<td>60</td>
<td>59</td>
<td>0.2460</td>
<td>0.2530</td>
<td>0.3750</td>
<td>0.1537</td>
</tr>
<tr>
<td>ASSDISP</td>
<td>A</td>
<td>0.7674</td>
<td>149</td>
<td>136</td>
<td>136</td>
<td>0.6154</td>
<td>0.6326</td>
<td>0.0872</td>
<td>0.5617</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.7293</td>
<td>98</td>
<td>51</td>
<td>51</td>
<td>0.2308</td>
<td>0.2372</td>
<td>0.4796</td>
<td>0.1201</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0.7085</td>
<td>96</td>
<td>53</td>
<td>53</td>
<td>0.2398</td>
<td>0.2465</td>
<td>0.4479</td>
<td>0.1324</td>
</tr>
<tr>
<td>EMPLOY</td>
<td>B</td>
<td>0.0924</td>
<td>27</td>
<td>13</td>
<td>13</td>
<td>0.2653</td>
<td>0.4333</td>
<td>0.5185</td>
<td>0.1277</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0.1148</td>
<td>34</td>
<td>17</td>
<td>17</td>
<td>0.3469</td>
<td>0.5667</td>
<td>0.5000</td>
<td>0.1735</td>
</tr>
</tbody>
</table>
4. Further Research

The influence of tree size on outlier and error detection performance is being investigated using the 1997 ABI dataset. In addition, application of the WAID-based outlier detection and imputation approach to the Swiss EPE dataset is being carried out. Finally, the extension of the WAID algorithm to allow trees based on multivariate response variables is being implemented.
Appendix A The WAID Toolkit

This appendix contains a complete listing of the Splus/R code for the different functions that make up the WAID Toolkit. Appendix B illustrates the use of this Toolkit to carry out an edit and imputation analysis for the TURNOVER variable in the sec197(y3) ABI dataset.

```
RTWaid.basic <-
function(y,x,xlabels,mono,delta=0.00001,minn=5,maxnogroups=20,robust=TRUE,method="M",psi=psi.bisquare,c=4.685,maxit=10,verbose=0) {
# This function fits a robust regression tree
# y = response variable data values (numeric, interval scaled)
# x = predictor variable values (numeric, categorical), in matrix form (rows = observations, columns = variables)
# each distinct value in x is taken as defining a category
# there is a limit of 10 categories for any non-monotone x-variable (otherwise GenerateAllSamples function needs modification)
# xlabels = vector of column numbers (or column names) for x-variables
# mono = logical vector identifying which of the x-variables is monotone (TRUE) or non-monotone (FALSE)
# delta = minimum heterogeneity (relative to input y) required before a node can be split
# minn = minimum size of node that can be created
# maxnogroups = maximum number of terminal nodes that can be created
# robust = logical flag (TRUE = use robust WSS to split, FALSE = use unweighted WSS to split)
# method, psi, c, maxit = parameters used to control robust fit (type help(rlm.default) for details).
# verbose = control value for printing out of results
  if(robust==TRUE) library(MASS)
  y <- c(y)
  if(!is.matrix(x)) x <- matrix(x,ncol=1)
  N <- length(y)
  P <- maxnogroups*2+1
  SplitHistory <- vector("list",length=maxnogroups)
  ObsNodeNo <- vector(mode="integer",length=N)
  ObsWt <- vector(mode="double",length=N)
  NodeHetero <- vector(mode="double",length=P)
  NodeSplit <- vector(mode="integer",length=P)
  NodeSize <- vector(mode="integer",length=P)
  NodeLoc <- vector(mode="integer",length=P)

# Initialise variables
  ParentNodeNo <- 1
  NumberNodes <- 1
  NodeSize[1] <- N
  NodeSize[-1] <- 0
  ObsNodeNo[ ] <- 1
  ObsWt[ ] <- 1
  Finish <- FALSE
```
TermHistory <- 1
NodeHistory <- rep(1,N)
WtHistory <- NULL
# GO
if( verbose>0) cat("Starting Regression Tree Analysis\n")
for (nstep in 1:maxnogroups) {
  if (Finish == TRUE) break
  NewNode <- NumberNodes-1
  if (nstep == 1) {
      NewNode <- 1
  }
  for (j in NewNode:NumberNodes) {
    NodeSize[j] <- sum(ObsNodeNo==j)
    Fit <- WeightedSumSquares(x=y[ObsNodeNo==j],robust=robust,method=method,psi=psi,c=c,maxit=maxit)
    NodeHetero[j] <- Fit$wss
    NodeLoc[j] <- Fit$loc
    ObsWt[ObsNodeNo==j] <- Fit$w
    # Test if node is allowed to be split
    if (nstep <= maxnogroups) {
      # Check that node has at least twice the minimum number of cases allowed in a node
      if (NodeSize[j] >= (2*minn)) {
        NodeSplit[j] <- 1
      }
      # If not write message
      if(NodeSplit[j] == 0) {
        if(verbose>2) cat("Node ",j," cannot be split, since it is too small\n")
      }
      # Check NodeHetero fraction for node
      if (zdiv(NodeHetero[j],NodeHetero[1]) <= delta) {
        NodeSplit[j] <- 0
        # NodeHetero fraction is too small, so write message
        if(verbose > 2) cat("Node ",j," cannot be split, since it is too homogeneous\n")
      }
      # Print statistics if node cannot be NodeSplit
      if (NodeSplit[j] == 0) {
        if(verbose > 2) {
          cat("Node ",j," N ",NodeSize[j]," Heterogeneity\n")
        }
      }
  }
  # End SplitCriteria
candidates <- unique(ObsNodeNo)
repeat {
  Finish <- TRUE
  for (j in candidates) {
    if (NodeSplit[j] == 1) {
      Finish <- FALSE
    }
  }
  if(Finish == TRUE) {
    if( verbose > 1) cat("There are no parent nodes left to split\n");
    break
  }
}
# Parent group statistics
if( verbose > 1) cat("Candidate nodes are as follows - \n")
if( verbose > 1) cat("Node N Heterogeneity\n")
for (k in candidates) {
  if (NodeSplit[k] == 1) {
    if( verbose > 1) cat(k," N",NodeSize[k]," Heterogeneity",NodeHetero[k],"\n")
  }
}
# After first split find splittable group with maximum NodeHetero
if( NumberNodes > 1) {
  WSSM <- 0.0
  for (k in candidates) {
    if (NodeSplit[k] == 1) {
      if (NodeHetero[k] >= WSSM) {
        ParentNodeNo <- k
        WSSM <- NodeHetero[k]
      }
    }
  }
}
if ( verbose > 0) cat("Step ",nstep," New parent node = ",ParentNodeNo,"\n")
# Find best split of new parent group based on each X-variable
xsplit <- vector("list",length=ncol(x))
if( verbose > 2) cat("Predictor Best Split Heterogeneity Left Codes Right Codes\n")
for(k in 1:ncol(x)) {
ksplit <-
BestRTSplit(y=y[ObsNodeNo==ParentNodeNo],x=x[ObsNodeNo==ParentNodeNo,k],mono=mono[k],w=ObsWt[ObsNodeNo==ParentNodeNo],min=minn,ws=NodeHetero[ParentNodeNo],loc=NodeLoc[ParentNodeNo])
xsplit[[k]] <- ksplit
if( verbose > 2) {
  if(xsplit[[k]]$split == FALSE) {
    cat("Cannot split on predictor ",xlabels[k]," \\
  }
  else {
    cat(xlabels[k]," \\
",xsplit[[k]]$wss," \\
",xsplit[[k]]$left," \\
",xsplit[[k]]$right," \\
  }
}
}

# Find the overall best split, if one exists
splitok <- FALSE
for(k in 1:ncol(x)) {
  if(xsplit[[k]]$split == TRUE) {
    splitok <- TRUE
    best.xsplit <- xsplit[[k]]
    best.x <- k
    break
  }
}

# Where no predictor can be used to split the parent node, flag this parent and try next parent
if(splitok == FALSE) {
  NodeSplit[ParentNodeNo] <- 0;
  if( verbose > 2) cat("Group ",ParentNodeNo," cannot be split. Decrease in heterogeneity or child node size is always too \\
  \\
small\n")
} else {
  if((ncol(x) > 1) & (best.x < ncol(x))) {
    for(k in (best.x+1):ncol(x)) {
      if((xsplit[[k]]$wss < best.xsplit$wss) & (xsplit[[k]]$split == TRUE)) {
        best.xsplit <- xsplit[[k]]
        best.x <- k
      }
    }
    if( verbose > 1) cat("Best split is on predictor ",xlabels[best.x]," \\
    \\
  )
  }
}
if(NodeSplit[ParentNodeNo] == 1) {
for(k in best.xsplit$left) {
    ObsNodeNo[(ObsNodeNo==ParentNodeNo)&(x[,best.x]==k)] <- NumberNodes+1
}
for(k in best.xsplit$right) {
    ObsNodeNo[(ObsNodeNo==ParentNodeNo)&(x[,best.x]==k)] <- NumberNodes+2
}
NumberNodes <- NumberNodes+2
break
}
TermHistory <- TermHistoryUpdate(x=TermHistory,p=ParentNodeNo,c1=NumberNodes-1,c2=NumberNodes)
NodeHistory <- cbind(NodeHistory,ObsNodeNo)
WtHistory <- cbind(WtHistory,ObsWt)
SplitHistory[[nstep]] <- list(parent=ParentNodeNo,child.left=NumberNodes-1,child.right=NumberNodes,predictor=xlabels[best.x],codes.left=best.xsplit$left,codes.right=best.xsplit$right)
if (Finish != TRUE) {
    # End of splitting since maximum number of terminal nodes has being reached
    if(verbose > 0) cat("End of run since number of terminal nodes >= ", maxnogroups,"\n")
}
NumberTerminals <- nrow(TermHistory)
TerminalHistory <- vector("list",length=NumberTerminals)
for(i in 1:NumberTerminals) {
    TerminalHistory[[i]] <- unique(TermHistory[i,])
}
# Output results
list(nodes = NodeHistory, splits = SplitHistory, terminals = TerminalHistory, weights = WtHistory)

CTWaid.basic <- function(y,x,xlabels,mono,delta=0.00001,minn=5,maxnogroups=20,gini=TRUE,verbose=0) {
    # this function fits a classification tree
    # y = response variable data values (numeric, categorical)
    # x = predictor variable values (numeric, categorical), in matrix form (rows = observations, columns = variables)
    # each distinct value in y (and x) is taken as defining a category
    # there is a limit of 10 categories for any non-monotone x-variable (otherwise GenerateAllSamples function needs modification)
    # xlabels = vector of column numbers (or column names) for x-variables
    # mono = logical vector identifying which of the x-variables is monotone (TRUE) or non-monotone (FALSE)
    # delta = minimum heterogeneity (relative to input y) required before a node can be split
    # minn = minimum size of node that can be created
    # maxnogroups = maximum number of terminal nodes that can be created
    # gini = logical flag (TRUE = use Gini heterogeneity index, FALSE = use Kullback-Leibler heterogeneity index)
# verbose = control value for printing out of results
y <- c(y)
if(!is.matrix(x)) x <- matrix(x,ncol=1)
N <- length(y)
P <- maxnogroups*2+1
SplitHistory <- vector("list",length=maxnogroups)
ObsNodeNo <- vector(mode="integer",length=N)
NodeHetero <- vector(mode="double",length=P)
NodeSplit <- vector(mode="integer",length=P)
NodeSize <- vector(mode="integer",length=P)

# Initialise variables
ParentNodeNo <- 1
NumberNodes <- 1
NodeSize[1] <- N
NodeSize[-1] <- 0
ObsNodeNo[ ] <- 1
Finish <- FALSE
TermHistory <- 1
NodeHistory <- rep(1,N)

# GO
if(verbose>0) cat("Starting Classification Tree Analysis\n")
for (nstep in 1:maxnogroups) {
  if (Finish == TRUE) break
  NewNode <- NumberNodes-1
  if (nstep == 1) {
    NewNode <- 1
  }
  for (j in NewNode:NumberNodes) {
    NodeSize[j] <- sum(ObsNodeNo==j)
    if (nstep <= maxnogroups) {
      # Check that node has at least twice the minimum number of cases allowed in a node
      if (NodeSize[j] >= (2*minn)) {
        NodeSplit[j] <- 1
      }
    }
    NodeHetero[j] <- GiniCoefficient(y[ObsNodeNo==j])$gini
  }
}

# Test if node is allowed to be NodeSplit
# If not write message
if(NodeSplit[j] == 0) {
    if(Verbose>2) cat("Node ",j," cannot be split, since it is too small\n")
}

# Check NodeHetero fraction for node
if (zdiv(NodeHetero[j],NodeHetero[1]) <= delta) {
    NodeSplit[j] <- 0
    # NodeHetero fraction is too small, so write message
    if(Verbose > 2) cat("Node ",j," cannot be split, since it is too homogeneous\n")
}

# Print statistics if node cannot be NodeSplit
if (NodeSplit[j] == 0) {
    if(Verbose > 2) {
        cat("Node N Heterogeneity\n")
        cat(j," ",NodeSize[j]," ",NodeHetero[j],"\n")
    }
}

# End SplitCriteria
}

candidates <- unique(ObsNodeNo)
repeat {
    Finish <- TRUE
    for (j in candidates) {
        if (NodeSplit[j] == 1) {
            Finish <- FALSE
        }
    }
    if(Finish == TRUE) {
        if(Verbose > 1) cat("There are no parent nodes left to split\n");
        break
    }
}

# Parent group statistics
if(Verbose > 1) cat("Candidate nodes are as follows - \n")
if(Verbose > 1) cat("Node N Heterogeneity\n")
for (k in candidates) {
    if (NodeSplit[k] == 1) {
        if(Verbose > 1) cat(k," ",NodeSize[k]," ",NodeHetero[k],"\n")
    }
}

# After first split find splittable group with maximum NodeHetero
if(NumberNodes > 1) {
    TGINIM <- 0.0
    for (k in candidates) {
        if (NodeSplit[k] == 1) {
            if (NodeHetero[k] >= TGINIM) {
                ParentNodeNo <- k
                TGINIM <- NodeHetero[k]
            }
        }
    }
}

if((verbose > 0) cat("Step ",nstep," New parent node = ",ParentNodeNo,"\n")

# Find best split of new parent group based on each X-variable
xsplit <- vector("list",length=ncol(x))

if(verbose > 2) cat("Predictor Best Split Heterogeneity Left Codes Right Codes\n")
for(k in 1:ncol(x)) {
    ksplit <-
    BestCTSplit(y=y[ObsNodeNo==ParentNodeNo],x=x[ObsNodeNo==ParentNodeNo,k],mono=mono[k],min=minn,gini=gini,gini.all=NodeHetero[ParentNodeNo])
    xsplit[[k]] <- ksplit
    if(verbose > 2) {
        if(xsplit[[k]]$split == FALSE) {
            cat("Cannot split on predictor ",xlabels[k],"\n")
        } else {
            cat(xlabels[k]," ",xsplit[[k]]$gini," ",xsplit[[k]]$left," ",xsplit[[k]]$right,"\n")
        }
    }
}

# Find the overall best split, if one exists
splitok <- FALSE
for(k in 1:ncol(x)) {
    if(xsplit[[k]]$split == TRUE) {
        splitok <- TRUE
        best.xsplit <- xsplit[[k]]
        best.x <- k
        break
    }
}

# Where no predictor can be used to split the parent node, flag this parent and try next parent
if(splitok == FALSE) {
NodeSplit[ParentNodeNo] <- 0;
if(Verbose > 2) cat("Group ",ParentNodeNo," cannot be split. Decrease in heterogeneity or child node size is always too
small\n")
}
else {
if((ncol(x) > 1) & (best.x < ncol(x))) {
for(k in (best.x+1):ncol(x)) {

if((xsplits[[k]]$gini < best.xsplits$gini) & (xsplits[[k]]$split == TRUE)) {

best.xsplits <- xsplits[[k]]

best.x <- k

}

}

}
if(Verbose > 1) cat("Best split is on predictor ",xlabels[best.x],"\n")
}
if(NodeSplit[ParentNodeNo] == 1) {
for(k in best.xsplits$left) {

ObsNodeNo[(ObsNodeNo==ParentNodeNo)&(x[,best.x]==k)] <- NumberNodes+1

}
for(k in best.xsplits$right) {

ObsNodeNo[(ObsNodeNo==ParentNodeNo)&(x[,best.x]==k)] <- NumberNodes+2

}
NumberNodes <- NumberNodes+2
break
}

}
TermHistory <- TermHistoryUpdate(x=TermHistory,p=ParentNodeNo,c1=NumberNodes-1,c2=NumberNodes)
NodeHistory <- cbind(NodeHistory,ObsNodeNo)
SplitHistory[[nstep]] <- list(parent=ParentNodeNo,child.left=NumberNodes-1,child.right=NumberNodes,predictor=xlabels[best.x],codes.left=best.xsplits$left,codes.right=best.xsplits$right)
}
if (Finish != TRUE) {
# End of splitting since maximum number of terminal nodes has being reached
if(Verbose > 0) cat("End of run since number of terminal nodes >= ", maxnogroups,"\n")
}
NumberTerminals <- nrow(TermHistory)
TerminalHistory <- vector("list",length=NumberTerminals)
for(i in 1:NumberTerminals) {

TerminalHistory[[i]] <- unique(TermHistory[i,])
}
# Output results
list(nodes = NodeHistory, splits = SplitHistory, terminals = TerminalHistory)

WeightedSumSquares <- function(x, robust=TRUE, method="M", psi=psi.bisquare, c=4.685, maxit=10) {
  # this function computes within node heterogeneity for regression trees
  n <- length(c(x))
  if(robust==TRUE) {
    meanfit <- rlm.default(x=rep(1,n), y=c(x), method=method, psi=psi, c=c, maxit=maxit)
    w <- meanfit$w
    w <- w*zdiv(n, sum(w))
    loc <- zdiv(sum(x*w), n)
  } else {
    w <- rep(1,n)
    loc <- mean(x)
  }
  wss <- sum(w*x^2) - n*loc^2
  list(loc = loc, wss = wss, w = w)
}

GiniCoefficient <- function(x) {
  # this function computes within node heterogeneity for classification trees
  n <- length(x)
  values <- unique(x)
  p <- rep(0, length(values))
  gini <- 0
  info <- 0
  for(i in 1:length(values)) {
    freq <- sum(x==values[i])
    p[i] <- freq/n
    gini <- gini + (p[i]*(1-p[i]))
    info <- info - (p[i]*log(p[i]))
  }
  gini <- n*gini
  info <- n*info
  list(p = p, gini = gini, info=info)
}

BestRTSplit <- function(y, x, v=unique(x), mono=FALSE, w=rep(1, length(y)), min=5, wss=0, loc=0) {
  # this regression tree function determines the optimal split of a node based on the values of a covariate
  splittable <- FALSE

```r
best.split <- list(left=v,right=NULL)
better.split.wss <- 0
if(length(v) > 1) {
  if(mono==TRUE) {
    v.sort <- sort(v)
  } else {
    v.ybar <- rep(0,length(v))
    for(i in 1:length(v)) {
      v.ybar[i] <- zdiv(sum(y[x==v[i]]*w[x==v[i]]),sum(w[x==v[i]]))
    }
    v.sort <- v[order(v.ybar)]
  }
  left <- v.sort
  right <- NULL
  best.split <- list(left=left,right=right)
  best.split.wss <- wss
  for(i in 1:(length(v)-1)) {
    left <- v.sort[1:i]
    y.left <- Subset(y=y,x=x,v=left)
    w.left <- Subset(y=w,x=x,v=left)
    n.left <- sum(w.left)
    ybar.left <- zdiv(sum(y.left*w.left),n.left)
    right <- v.sort[-(1:i)]
    y.right <- Subset(y=y,x=x,v=right)
    w.right <- Subset(y=w,x=x,v=right)
    n.right <- sum(w.right)
    ybar.right <- zdiv(sum(y.right*w.right),n.right)
    if((length(y.left) >= min) & (length(y.right) >= min)) {
      split.wss <- wss-(n.left*(ybar.left-loc)^2)-(n.right*(ybar.right-loc)^2)
      if(split.wss < best.split.wss) {
        best.split <- list(left=left,right=right)
        best.split.wss <- split.wss
        splittable <- TRUE
      }
    }
  }
}
list(left=best.split$left,right=best.split$right,wss=best.split.wss,splittable=TRUE)
```
BestCTSplit <- function(y,x,v=unique(x),mono=FALSE,min=5,gini=TRUE,gini.all=0) {
  # this classification tree function determines the optimal split of a node based on the values of a covariate
  splittable <- FALSE
  best.split <- list(left=v,right=NULL)
  best.split.gini <- 0
  if(length(v) > 1) {
    if(mono==TRUE) {
      v.sort <- sort(v)
      left <- v.sort
      right <- NULL
      gini.left <- gini.all
      gini.right <- 0
      best.split <- list(left=left,right=right)
      best.split.gini <- gini.left+gini.right
      for(i in 1:(length(v)-1)) {
        left <- v.sort[1:i]
        y.left <- Subset(y=y,x=x,v=left)
        right <- v.sort[-(1:i)]
        y.right <- Subset(y=y,x=x,v=right)
        if((length(y.left) >= min) & (length(y.right) >= min)) {
          if(gini==TRUE) {
            gini.left <- GiniCoefficient(y.left)$gini
            gini.right <- GiniCoefficient(y.right)$gini
          } else {
            gini.left <- GiniCoefficient(y.left)$info
            gini.right <- GiniCoefficient(y.right)$info
          }
          split.gini <- gini.left+gini.right
          if(split.gini < best.split.gini) {
            best.split <- list(left=left,right=right)
            best.split.gini <- split.gini
            splittable <- TRUE
          }
        }
      }
    } else {
      split <- GenerateAllBinarySplits(v=v)
      left <- v
      right <- NULL
    }
  }
  else {
    split <- GenerateAllBinarySplits(v=v)
    left <- v
    right <- NULL
  }
}

gini.left <- gini.all
gini.right <- 0
best.split <- list(left=left,right=right)
best.split.gini <- gini.left+gini.right
for(i in 1:length(split)) {
  left <- split[[i]]
y.left <- Subset(y=y,x=x,v=left)
right <- Delete(s=left,v=v)
y.right <- Subset(y=y,x=x,v=right)
if((length(y.left) >= min) & (length(y.right) >= min)) {
  if(gini==TRUE) {
    gini.left <- GiniCoefficient(y.left)$gini
    gini.right <- GiniCoefficient(y.right)$gini
  } else {
    gini.left <- GiniCoefficient(y.left)$info
    gini.right <- GiniCoefficient(y.right)$info
  }
split.gini <- gini.left+gini.right
  if(split.gini < best.split.gini) {
    best.split <- list(left=left,right=right)
    best.split.gini <- split.gini
    splittable <- TRUE
  }
}
}
list(left=best.split$left,right=best.split$right,gini=best.split.gini,split=splittable)

GenerateAllBinarySplits <- function(v) {
  # this classification tree function generates all candidate splits of a non-monotone covariate
  # note that there is a limit of 10 categories for the covariate at present
  maxsize <- floor(length(v)/2)
  number <- rep(0, maxsize)
  for(k in 1:maxsize) number[k] <- choose(length(v),k)
  n <- sum(number)
  splits <- vector("list",length=n)
  count <- 0
  for(k in 1:maxsize) {
    ...
groups <- GenerateAllSamples(v,k)
for(j in 1:length(groups)) {
    count <- count+1
    splits[count] <- list(groups[[j]])
}
splits

GenerateAllSamples <- function(x,n) {
    # function called by GenerateAllBinarySplits
    if(n > 5) stop("Sample size > 5")
    N <- length(x)
    sample.matrix <- matrix(0,nrow=choose(N,n),ncol=n)
    count <- 0
    for(i1 in 1:N) {
        if(n==1) {
            count <- count+1
            sample.matrix[count,] <- x[i1]
        } else {
            if(i1<N) {
                for(i2 in (i1+1):N) {
                    if(n==2) {
                        count <- count+1
                        sample.matrix[count,] <- c(x[i1],x[i2])
                    } else {
                        if(i2<N) {
                            for(i3 in (i2+1):N) {
                                if(n==3) {
                                    count <- count+1
                                    sample.matrix[count,] <- c(x[i1],x[i2],x[i3])
                                } else {
                                    if(i3<N) {
                                        for(i4 in (i3+1):N) {
                                            if(n==4) {
                                                count <- count+1
                                                sample.matrix[count,] <- c(x[i1],x[i2],x[i3],x[i4])
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
else {

    if(i4<N) {
        for(i5 in (i4+1):N) {
            count <- count+1
            sample.matrix[count,] <- c(x[i1],x[i2],x[i3],x[i4],x[i5])
        }
    }
}

out <- vector("list",length=nrow(sample.matrix))
for(i in 1:length(out)) out[i] <- list(sample.matrix[i,])
out

Delete <- function(s,v) {
    # utility function
    unique(c(s,v))[-(1:length(s))]
}

Subset <- function(y,x,v) {
    # utility function
    out <- NULL
    for(i in 1:length(v)) out <- c(out,y[x==v[i]])
    out
}

zdiv <- function(x,y){
    # utility function
    if(y==0.0||is.na(x)||is.na(y))
        0.0
    else
        x/y
TermHistoryUpdate <- function(x,p,c1,c2) {
  # function that creates the "history" of the tree splitting process
  if(!is.matrix(x)) x <- matrix(x[1],nrow=1,ncol=1)
  n <- ncol(x)
  if(n == 1) {
    x1 <- NULL
    x2 <- x
  }
  else {
    x1 <- x[x[,n] != p,]
    if(!is.matrix(x1)) x1 <- matrix(x1,nrow=1)
    x2 <- x[x[,n] == p,]
  }
  out <- rbind(x1,x2,x2)
  out <- cbind(out,c(x1[,n],c1,c2))
  out
}

TerminalDefinition <- function(t,SplitHistory,TerminalHistory) {
  # function that returns the sequence of splits that lead to a particular node
  for(i in 1:length(TerminalHistory)) {
    count <- length(TerminalHistory[[i]])
    if(TerminalHistory[[i]][count] == t) {
      trace <- TerminalHistory[[i]]
      definition <- vector("list",length=(length(trace)-1))
      for(j in 2:length(trace)) {
        k <- trace[j]
        split <- ceiling((k-1)/2)
        right <- TRUE
        if((k-1)/2 < split) {
          right <- FALSE
        }
        if(right == FALSE) {
          definition[[j-1]] <-
          list(nodes=c(SplitHistory[[split]]$parent,SplitHistory[[split]]$child.left),
                x=SplitHistory[[split]]$predictor,
                codes=SplitHistory[[split]]$codes.left)
        } else {
          definition[[j-1]] <-
          list(nodes=c(SplitHistory[[split]]$parent,SplitHistory[[split]]$child.right),
                x=SplitHistory[[split]]$predictor,
                codes=SplitHistory[[split]]$codes.right)
        }
      }
    }
  }
}
TerminalFinder <- function(x, xlabels, SplitHistory) {
  # function that returns the terminal node in a WAID tree given a set of values for the covariates that define the tree
  node <- 1
  for(i in 1:length(SplitHistory)) {
    if(SplitHistory[[i]]$parent == node) {
      xval <- x[xlabels == SplitHistory[[i]]$predictor]
      if(is.element(xval, SplitHistory[[i]]$codes.left)) {
        node <- SplitHistory[[i]]$child.left
      } else {
        node <- SplitHistory[[i]]$child.right
      }
    }
  }
  node
}

NodeMeans <- function(y, nodes, weights) {
  # function that calculates the weighted mean values for all nodes in a WAID regression tree
  nodematrix <- nodes
  nsplits <- ncol(weights)
  if(ncol(nodes) > nsplits) nodematrix <- nodes[, 1:nsplits]
  if(ncol(nodes) < nsplits) stop("incompatible node and weight matrix")
  nodemeans <- nodematrix
  for(i in 1:nsplits) {
    nodeset <- unique(nodematrix[, i])
    for(j in nodeset) {
      sumwj <- sum(weights[nodematrix[, i] == j, i])
      meanj <- sum(weights[nodematrix[, i] == j, i] * y[nodematrix[, i] == j]) / sumwj
      nodemeans[nodematrix[, i] == j, i] <- meanj
    }
  }
  nodemeans
NodeMeanPlot <- function(nodes,means,splits=1:ncol(means),points=TRUE,title="NODEMEAN TREE")
# function that plots the nodemean tree corresponding to a WAID regression tree
    if(ncol(nodes) < max(splits)) stop("incompatible nodes matrix")
    if(ncol(means) < max(splits)) stop("incompatible means matrix")
    nodematrix <- nodes[,sort(splits)]
    meanmatrix <- means[,sort(splits)]
    plot(x=c(min(splits),max(splits)),y=c(min(meanmatrix),max(meanmatrix)),type="n",xlab="SPLIT",ylab="NODEMEAN")
    if(points) {
        for(i in 1:ncol(nodematrix)) {
            y <- unique(meanmatrix[,i])
            m <- length(y)
            points(x=rep(splits[i],m),y=y)
        }
    }
    else {
        terminals <- unique(nodematrix[,ncol(nodematrix)])
        for(i in sort(terminals)) {
            nodeindices <- (1:nrow(nodematrix))[nodematrix[, ncol(nodematrix)]==i]
            lines(x=splits,y=meanmatrix[nodeindices[1],])
        }
    }
    title(title)

TreeImputer <- function(ximp,xlabels,yvalues,treesplits,treenodes,treeweights,minwt=1,random=F) {
# function that returns imputed values from terminal nodes associated with input values of covariates
    x <- ximp
    weights <- treeweights
    nodes <- treenodes
    splits <- treesplits
    if(ncol(nodes)!=ncol(weights)) {
        if(ncol(nodes)==(ncol(weights)+1)) {
            nodes <- nodes[,1:(ncol(nodes)-1)]
            splits <- splits[1:(length(splits)-1)]
        }
        else stop("node matrix and weight matrix dimensions incompatible")
    }
    else stop("node matrix and weight matrix dimensions incompatible")
    if(!is.matrix(x)) x <- matrix(x,nrow=1)
    imputes <- rep(0, nrow(x))
for(i in 1:nrow(x)) {
    terminal <- TerminalFinder(x=x[i,],xlabels=xlabels,SplitHistory=splits)
    ydoners <- yvalues[nodes[,ncol(nodes)]==terminal]
    wdoners <- weights[nodes[,ncol(nodes)]==terminal,ncol(weights)]
    mean <- sum(ydoners*wdoners)/sum(wdoners)
    imputes[i] <- mean
    if(random) imputes[i] <- sample(ydoners[wdoners>=minwt],size=1)
}
cbind(imputes,x)

CatMap <- function(x,bounds,groups) {
    # function that maps covariate values to a given set of groups
    if(length(bounds)!=(length(groups)-1)) stop("Incorrect number of groups")
    m <- length(groups)
    xcat <- rep(0, length(x))
    for(i in 1:length(x)) {
        if(x[i] <= bounds[1]) xcat[i] <- groups[1]
        if(x[i] > bounds[m-1]) xcat[i] <- groups[m]
        for(j in 2:(m-1)) {
            if((x[i]>bounds[j-1]) & (x[i]<=bounds[j])) break
        }
    }
    xcat
}
Appendix B Example WAID Outlier Identification and Imputation Analysis

Read in values of total variables for sec197(true+y3) data
abidata <- matrix(scan("totals197(merged).dat",na.strings = "."),ncol=38,byrow=T)

Create individual data vectors that exclude missing values of TURNOVER (these have abidata[,8]==3)
Identifiers
abidata.ref3 <- abidata[(abidata[,8]==1)|(abidata[,8]==2),1]
Values of TURNREG
abidata.turnreg3 <- abidata[(abidata[,8]==1)|(abidata[,8]==2),2]
Values of EMPREG
abidata.empreg3 <- abidata[(abidata[,8]==1)|(abidata[,8]==2),3]
Values of true TURNOVER
abidata.tturnover3 <- abidata[(abidata[,8]==1)|(abidata[,8]==2),4]
Values of observed (with error) TURNOVER
abidata.oturnover3 <- abidata[(abidata[,8]==1)|(abidata[,8]==2),7]
Values of true(1)/error(2) flag for TURNOVER
abidata.turnover.error3 <- abidata[(abidata[,8]==1)|(abidata[,8]==2),8]

Calculate percentiles of TURNREG
abidata.turnreg3.tile <- quantile(abidata.turnreg3,probs=(1:99)/100)
Create classification variable based on TURNREG percentile groups
abidata.turnreg3.class <- cut(abidata.turnreg3,breaks=c(-1,abidata.turnreg3.tile,max(abidata.turnreg3)+1),labels = FALSE)

Calculate robust WAID tree for TURNOVER using TURNREG and EMPREG as covariates
abi.turnover3.waid.wls <- 
RTWaid.basic(y=log(abidata.oturnover3+1),x=cbind(abidata.turnreg3.class,abidata.empreg3),xlabels=c("TURNREG","EMPREG"),mono=c(TRUE,TRUE),mi
nn=5,maxnogroups=50,robust=TRUE,maxit=100)

Calculate robust mean values for all nodes in the tree
abi.turnover3.waid.wtmeans <- NodeMeans(y=log(abidata.oturnover3+1),nodes=abi.turnover3.waid.wls$nodes,weights=abi.turnover3.waid.wls$weights)

Create a nodemean plot for the tree (see Figure 2)
NodeMeanPlot(nodes=abi.turnover3.waid.wls$nodes,means=abi.turnover3.waid.wtmeans,points=FALSE,title="Figure 2: Robust nodemean tree for y3 values of log(TURNOVER+1)")
Calculate average weights for each record defining the tree
abi.turnover3.waid.avwt <- apply(abi.turnover3.waid.wls$weights,1,mean)

Grid search for an optimal threshold weight identifying outliers/errors (This code generates the values in Table 2)
abi.data.turnover3.rdiff <- abs(abi.data.oturnover3-abi.data.tturnover3)/abi.data.tturnover3
abi.data.turnover3.rdiff[abi.data.tturnover3==0] <- 0
TE <- sum(abi.data.turnover.error3==2)
TBE <- sum(abi.data.turnover3.rdiff>1)
for(i in 1:100) {
w <- i/1000
N <- sum(abi.turnover3.waid.avwt<w)
M <- sum(abi.turnover3.waid.avwt<w & abi.data.turnover.error3==2)
Mbig <- sum(abi.turnover3.waid.avwt<w & abi.data.turnover3.rdiff>1)
R1 <- M/TE
R1big <- Mbig/TBE
R2 <- 1-M/N
C <- R1*(1-R2)
print(c(w,N,M,Mbig,R1,R1big,R2,C))
}
Create random imputations for missing values of TURNOVER. Note that donors are selected from cases with average weight > 1

temp <-
cbind(log(abidata[abidata[,8]==3,4]+1),TreeImputer(ximp=cbind(CatMap(x=abidata[abidata[,8]==3,2],bounds=abidata.turnreg3.tile,groups=1:100),abidata[ abidata[,8]==3,3]),xlabels=c("TURNREG","EMPREG"),yvalues=log(abidata.oturnover3+1),treesplits=abi.turnover3.waid.wls$splits,treenodes=abi.turnover 3.waid.wls$nodes,treeweights=abi.turnover3.waid.wls$weights,random=T))

Plot distribution of imputation errors (log scale and raw scale)

hist(temp[,1]-temp[,2])
hist(exp(temp[,1])-exp(temp[,2]))
Create mean imputations for missing values of TURNOVER
```r
temp <- cbind(log(abidata[abidata[,8]==3,4]+1),TreeImputer(ximp=cbind(CatMap(x=abidata[abidata[,8]==3,2],bounds=abidata.turnreg3.tile,groups=1:100),abidata[abidata[,8]==3,3]),xlabels=c("TURNREG","EMPREG"),yvalues=log(abidata.oturnover3+1),treesplits=abi.turnover3.waid.wls$splits,treenodes=abi.turnover3.waid.wls$nodes,treeweights=abi.turnover3.waid.wls$weights,random=F))
```

Plot distribution of imputation errors (log scale and raw scale)
```r
hist(temp[,1]-temp[,2])
hist(exp(temp[,1])-exp(temp[,2]))
```