Matching in R using the optmatch and RItools packages

Ben Hansen, Mark Fredrickson and Yevgeniya Pinelis

May 17, 2013

1 The R environment

All the software used in this worksheet is freely available. The R statistical package is installed for you in the lab, but you may download and install R for Windows, Mac, and Linux systems from: http://www.r-project.org.

The following document walks through a common propensity score matching workflow in R. Example R code will appear as italics with a > indicating the command prompt. You may type this code yourself — each line is a command to R. Output will follow in a typewriter font. For example:

```
> 2 + 2

[1] 4
```

R stores data in named variables using the arrow operator:

```
> my.variable <- 2 + 2
> my.variable * 3

[1] 12
```

2 Setup

2.1 Outfitting your R with the proper add-ons

R add-on packages are available to install directly from R:

```
> install.packages("optmatch")
> install.packages("RItools", dependencies=T)
```

These commands will ask you to select a CRAN server. Any server will do. You may also be asked whether you'd like to set up a "personal library to install packages into"; if so, answer yes. (The default personal library location that R will suggest should be OK.) You'll only need to run these commands the first time you want to use optmatch or RItools on a particular computer, or when you install a new version of R.

2.2 Setting up the R environment for matching

Attach extension packages that we'll be using for matching and associated diagnostics:

```
> library(optmatch)
> library(RItools)
```

You'll do this each time you want to run matching commands.

The nuclear plants data: load it by entering

```
> data(nuclearplants)
```

To see the first five rows:

```
> head(nuclearplants)
```

For more on the variables here, enter

```
> help('nuclearplants')
```

You can directly access a variable within this data frame as follows. (Try typing in the commands to see what they do.)

```
> nuclearplants$pt
> table(nuclearplants$pt)
> with(nuclearplants, table(pt))
```

The variable you will have just viewed and tabulated, `pt`, is a dummy for whether the plant was built with "partial turnkey guarantees." These plants were not comparable to the others in terms of construction costs. Let's exclude them for the time being, for simplicitly. To do this we'll create a data table (in `R` jargon, a "data frame") of just those observations for which `pt` is 0:

```
> nuke.nopt <- subset(nuclearplants, pt==0)
```

To inspect its first and last 5 entries, do

```
> head(nuke.nopt)
> tail(nuke.nopt)
```

To view this as presenting a matching problem, we'll think of plants built on the site of a previously existing plant (`pr==1`) as the treatment group and plants on new sites (`pr==0`) as comparisons.

3 Optimal pair matching and 1:k matching

3.1 pair matching

To check the number of treated and control plants:

```
> table(nuke.nopt$pr)
>
```

To get the pair match minimizing the mean paired distance on `cap`, among all collections of 7 nonoverlapping pairs, do

```
> pairmatch(match_on( pr~cap, data=nuke.nopt ) )
```

For matching on both `date` and `cap`, you'd type `match_on( pr~cap+date, ...)` instead of `match_on( pr~cap, ...)`. We'll talk later about what "match_on" stands for and what role it plays here. For now, note the form of the output this command generates: a variable of the same length as the variables making up `nuke.nopt`, assigning a distinct name to each macthed set. To fix your intuition, you might try connecting up in Figure 1 the units that `pairmatch()` has placed in the same matched sets.

For basic summary information about this match, try

```
> summary( pairmatch(match_on( pr~cap, data=nuke.nopt ) ) )
```

Existing site

	date	capacity
A	2.3	660
B	3.0	660
C	3.4	420
D	3.4	130
E	3.9	650
F	5.9	430
G	5.1	420

"date" is date of construction, in years after 1965; "capacity" is net capacity of the power plant, in MWe above 400.

New site

	date	capacity
H	3.6	290
I	2.3	660
J	3.0	660
K	2.9	110
L	3.2	420
M	3.4	60
N	3.3	390
O	3.6	160
P	3.8	390
Q	3.4	130
R	3.9	650
S	3.9	450
T	3.4	380
U	4.5	440
V	4.2	690
W	3.8	510
X	4.7	390
Y	5.4	140
Z	6.1	730

Figure 1: New- versus existing-site plants. "date" is `date`-65; "capacity" is `cap`-400.

If you've already typed in the `pairmatch(...)` part, you can use the up-arrow, Home and End keys to avoid having to re-type. Alternatively, to assign the name "`pm`" to the matching result, do

```
>   pm <- pairmatch(match_on( pr~cap, data=nuke.nopt ) )
```

— after this, you could just type `summary(pm)` to view the matching summary.

The following would give a basic matched analysis of the effect of new or existing site on construction costs is given with the help of R's linear modeling function. In effect, the existing site effect is estimated as one "way" in a two-way ANOVA, the other "way" being the factor variable that represents the matching result, i.e. `pm`.

```
> summary(lm(cost~pr+pm, data=nuke.nopt))
```

3.2 Matching with multiple controls

There are other types of matches you might want to try. Here's how to create matched triples:

```
> tm <- pairmatch(match_on( pr~cap, data=nuke.nopt ), controls=2 )
>
```

There will be further variations suggested on the slides.

3.3 Distances

You'll often want to do several variations on a match. It may save computation and typing time to store the distance you're using to match, if you're going to re-use that distance.

```
> cap.dist <- match_on( pr~cap, data=nuke.nopt )
> summary(pairmatch(cap.dist))

Matching failed in subclasses containing 12 of 26 observations.
Reporting on subclasses where matching worked. (Enter ?matchfailed for more info.)
Structure of matched sets:
```

```
1:1
  7
Effective Sample Size:  7
(equivalent number of matched pairs).

sum(matched.distances)=0.22
(within 0.02 of optimum).
Percentiles of matched distances:
  0%  50%  95% 100%
0.00 0.00 0.12 0.14
```

```
> identical(pm, pairmatch(cap.dist))
```

```
[1] TRUE
```

```
> summary(pairmatch(cap.dist, 2))
```

```
Matching failed in subclasses containing 5 of 26 observations.
Reporting on subclasses where matching worked. (Enter ?matchfailed for more info.)
Structure of matched sets:
1:2
  7
Effective Sample Size:  9.3
(equivalent number of matched pairs).

sum(matched.distances)=1.8
(within 0.015 of optimum).
Percentiles of matched distances:
   0%   50%   95%  100%
0.000 0.093 0.430 0.653
```

```
> identical(tm, pairmatch(cap.dist,2))
```

```
[1] TRUE
```

(Don't be misled by statement in the above about matching having "failed": looking below a bit, seven 1:1 pairs were successfully created, with the inevitable by-product that 12 controls could not be used.)

What does a matching distance look like? Here's the upper-left corner of one of them:

```
> cap.dist[1:3, 1:3]
```

```
         control
treatment   H   I   J
        A 1.8 0.0 0.0
        B 1.8 0.0 0.0
        C 0.6 1.2 1.2
```

(Note the use of square brackets, "[" and "]", for specifying rows and columns of the distance matrix.)

Matching with a caliper of 2 pooled s.d.s on the cap variable:

```
> cal.dist=cap.dist+caliper(cap.dist, 2)

> cal.dist

       control
treated   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W   X   Y
      A 1.8 0.0 0.0 Inf 1.2 Inf 1.3 Inf 1.3 Inf 0.1 1.0 1.4 1.0 0.1 0.7 1.3 Inf
      B 1.8 0.0 0.0 Inf 1.2 Inf 1.3 Inf 1.3 Inf 0.1 1.0 1.4 1.0 0.1 0.7 1.3 Inf
      C 0.6 1.2 1.2 1.5 0.0 1.7 0.1 1.2 0.2 1.4 1.1 0.1 0.2 0.1 1.3 0.4 0.2 1.4
      D 0.7 Inf Inf 0.1 1.4 0.3 1.2 0.1 1.2 0.0 Inf 1.5 1.2 1.5 Inf 1.8 1.2 0.0
      E 1.7 0.1 0.1 Inf 1.1 Inf 1.2 Inf 1.2 Inf 0.0 1.0 1.3 1.0 0.2 0.7 1.3 Inf
      F 0.7 1.1 1.1 1.5 0.0 1.8 0.2 1.3 0.2 1.4 1.1 0.1 0.2 0.1 1.2 0.4 0.2 1.4
      G 0.6 1.2 1.2 1.5 0.0 1.7 0.1 1.2 0.1 1.4 1.1 0.1 0.2 0.1 1.3 0.4 0.2 1.3
       control
treated   Z
      A 0.3
      B 0.3
      C 1.5
      D Inf
      E 0.4
      F 1.4
      G 1.5
```

Entries of `Inf` of `NaN` in a distance matrix are interpreted as forbidden matches. Thus

```
> cm=pairmatch(cal.dist)
> cm
```

matches on `cap`, insisting that paired units not differ by more than a pooled s.d. in `cap`[1].

3.4 Did matching work?

It's possible to give the software an impossible list of requirements for a match. For instance, the following fails, generating an error:

```
> pairmatch(cap.dist,3)
```

The problem here is that the data don't have 3 comparison units to go with each treatment unit.

Matching can also fail because the distance matrix embodies matching contstraints that are impossible to meet. In these cases the matching function will generally run without complaint, although it won't create any matches. Here is an example, where the caliper is so narrow as to forbid all possible matches:

```
> try( pairmatch(cap.dist+caliper(cap.dist, width=.0001)) )
>
```

[1] If you would prefer to set a requirement on how much paired units can differ in `cap` in its original units, rather than standard units, then you would have to create `cap.dist` in a somewhat different way. If you have verson 0.8-3 or later of optmatch, you'll find an example on the help page for `caliper()`. If you don't see such an example than you probably have an earlier verison of the package; those have closely related examples on the help pages for `match_on()` or `mdist()`, particularly the example of "Absolute differences on a scalar". Enter `!caliper`, `?match_on` or `?mdist` at the R command line.

If before matching you want to remove just the subjects lacking a counterpart within caliper distance, you can do `pairmatch(<...>, remove.unmatchables=T)`. That won't help with the miniscule caliper above, but with less extreme calipers it helps you salvage a few matches.

3.5 How closely did I match?

Getting back to a matching that succeeded, note that `summary()` reports information about how close the matches are.

```
> summary(pm)
```

```
Matching failed in subclasses containing 12 of 26 observations.
Reporting on subclasses where matching worked. (Enter ?matchfailed for more info.)
Structure of matched sets:
1:1
  7
Effective Sample Size:  7
(equivalent number of matched pairs).

sum(matched.distances)=0.22
(within 0.02 of optimum).
Percentiles of matched distances:
  0%  50%  95% 100%
0.00 0.00 0.12 0.14
```

In this default report, the measure of closeness is just whatever distance you matched on. For closeness in terms of some other distance, use `matched.distances`.

```
> matched.distances(pm, match_on(pr~date, data=nuke.nopt))
```

```
 1.1  1.2  1.3  1.4  1.5  1.6  1.7
0.68 0.68 0.25 0.00 0.00 1.45 1.78
```

```
> summary( matched.distances(pm, match_on(pr~date, data=nuke.nopt)) )
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.13    0.68    0.69    1.06    1.78
```

3.6 Did matching balance the covariate?

Comparing overt biases before and after matching. An assessment of the unmatched difference between the groups on `cap` can be had via:

```
> cap.noadj <- lm(cap ~ pr, nuke.nopt)
> summary(cap.noadj)
```

or, more directly,

```
> summary(lm(cap ~ pr, nuke.nopt))$coeff["pr",]
```

(Note again the use of square brackets, "[" and "]", for specifying subsets of a matrix. With `R` one has to carefully distinguish square brackets, curly brackets and parentheses.)

Here is a parallel calculation that takes the match `pm` into account.

```
> summary(lm(cap ~ pr + pm, nuke.nopt))$coeff["pr",]
```

Exercises.

1. Compare `pm`, `tm` and the unmatched samples in terms of balance on `t2`.
2. Compare `pm`, `tm` and the unmatched samples in terms of balance on `date`.
3. Compare `pm` to Mahalanobis pair matching on `t1` in terms of balance on `date`.
4. Compare Mahalanobis pair matching on `cap` and `date` to Mahalanobis pair matching on `cap`, `date` and each of `t1,t2`. Add the last two variables in one at a time, so that you're comparing a total of three matches. Compare on balance in `cap` and `t2`.

Section 4.2, below, presents convenient ways to do balance assessment for many variables at once. Before getting to that let's discuss try matching with propensity scores.

4 Propensity Score Matching

4.1 Propensity score fitting in R

Logistic regression models are fit in R using the function `glm()`, with "`family`" argument set to "`binomial`." Example:

```
> psm <- glm(pr~ date + t1 + t2 + cap + ne + ct + bw + cum.n + pt,
+            family=binomial, data=nuclearplants)
```

The fitted logistic regression is then stored in the object "`psm`." The propensity scores are then stored as `psm$fitted.values` (estimated probabilities) or `psm$linear.predictors` (estimated logits of probabilities).

It's often a good idea to compare the groups' distributions on the propensity score.

```
> pscore <- psm$linear.predictors
> boxplot(pscore~nuclearplants$pr)
```

The groups do overlap, if not greatly. It may be wise to restrict the sample to the region of overlap, at least roughly. A propensity caliper would help with this.

To extract a matrix of distances on the propensity score (and store it as "`psd`"), do

```
> psd <- match_on(psm)
```

From this point you could match directly on the propensity score,

```
> ps.pm = pairmatch(psd)
> summary(ps.pm)
```

```
Matching failed in subclasses containing 12 of 32 observations.
Reporting on subclasses where matching worked. (Enter ?matchfailed for more info.)
Structure of matched sets:
1:1
 10
Effective Sample Size:  10
(equivalent number of matched pairs).

sum(matched.distances)=11
(within 0.03 of optimum).
Percentiles of matched distances:
   0%   50%   95%  100%
0.018 0.873 2.630 2.920
```

or match on the propensity score within propensity score calipers,

```
> ps.pm2 = pairmatch(psd + caliper(psd, 2))
> summary(ps.pm2)

Matching failed in subclasses containing 12 of 32 observations.
Reporting on subclasses where matching worked. (Enter ?matchfailed for more info.)
Structure of matched sets:
1:1
 10
Effective Sample Size:  10
(equivalent number of matched pairs).

sum(matched.distances)=11
(within 0.028 of optimum).
Percentiles of matched distances:
   0%   50%   95%  100%
0.018 1.140 1.820 1.850
```

Here individual matches are required to be no more separated than 2 pooled SDs in the estimated propensity score.

Or you could match within calipers of the propensity score on some other distance, perhaps Mahalanobis distances based on selected covariates as recommended by Rubin and Thomas (2000, JASA) and others. For Mahalanobis matching on `date`, `cap` and the propensity score, for instance, combined with a propensity caliper of 1 pooled sd[2]:

```
> mhd1 <- match_on(pr~date+cap+pscore, data=nuclearplants)
> mhpc.pm <- pairmatch(mhd1+caliper(psd,1))
> summary(mhpc.pm) # Oops!

Matching failed.  (Restrictions impossible to meet?)
Enter ?matchfailed for more info.

> mhpc.pm <- pairmatch(mhd1+caliper(psd,2))
> summary(mhpc.pm) # That's better

Matching failed in subclasses containing 12 of 32 observations.
Reporting on subclasses where matching worked. (Enter ?matchfailed for more info.)
Structure of matched sets:
1:1
 10
Effective Sample Size:  10
(equivalent number of matched pairs).

sum(matched.distances)=12
(within 0.026 of optimum).
Percentiles of matched distances:
  0%  50%  95% 100%
0.18 0.98 2.30 2.37
```

[2]This assumes you ran the above command assigning a value to "`pscore`" (as was suggested near the beginning of § 4.1).

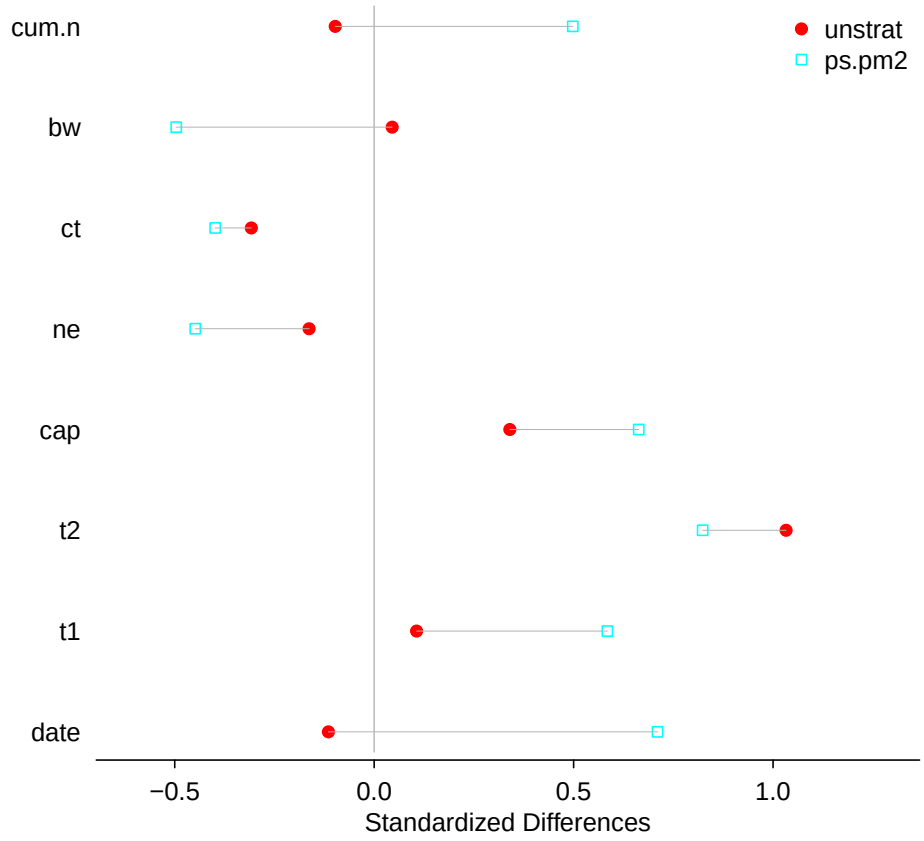
4.2 Checking balance in general

The `RItools` package has a convenient function for checking balance on many variables simultaneously. To get a sense of what it does, try this:

```
> library(RItools)
> xBalance(pr~ date + t1 + t2 + cap + ne + ct + bw + cum.n, data=nuclearplants)
> xBalance(pr~date + t1 + t2 + cap + ne + ct + bw + cum.n + pt, ps.pm2, nuclearplants)
```

It can in the same display compare matching to no matching, on any of a number of axes. Here is a demonstration:

```
> myb=xBalance(pr~ date + t1 + t2 + cap + ne + ct + bw + cum.n,
+          strata=list(unstrat=NULL, ps.pm2=~ps.pm2),
+          data=nuclearplants,
+          report=c("adj.means", "std.diffs",
+          "z.scores", "chisquare.test"))
> plot(myb)
```




```
> myb
```

```
      strata unstrat                                  ps.pm2
      stat       pr=0    pr=1 std.diff     z            pr=0    pr=1 std.diff     z
vars
```

```
date          68.62  68.50    -0.11   -0.31         68.10  68.83    0.71    1.02
t1            13.64  14.00    0.11    0.28          11.80  13.80    0.58    0.69
t2            59.32  69.10    1.03    2.47    *     63.80  71.60    0.82    0.96
cap           805.18 869.80   0.34    0.89          807.00 933.00   0.66    1.13
ne            0.27   0.20     -0.16   -0.43         0.60   0.40     -0.45   -0.58
ct            0.45   0.30     -0.31   -0.81         0.60   0.40     -0.40   -0.45
bw            0.18   0.20     0.05    0.12          0.40   0.20     -0.50   -0.58
cum.n         8.73   8.10     -0.10   -0.26         6.20   9.40     0.50    1.38
---Overall Test---
        chisquare df p.value
unstrat        11  8     0.2
ps.pm2          5  5     0.4
---
Signif. codes:  0 '***' 0.001 '** ' 0.01 '*  ' 0.05 '.  ' 0.1 '   ' 1
```

For a very compact representation of the assessment, call `summary()` on the match, passing along the fitted propensity model as a second argument. If you've got `RItools` loaded, then this will call `xBalance` in the background, reporting the summary chi-square test results.

```
> summary(ps.pm2, psm)

Matching failed in subclasses containing 12 of 32 observations.
Reporting on subclasses where matching worked. (Enter ?matchfailed for more info.)
Structure of matched sets:
1:1
 10
Effective Sample Size:  10
(equivalent number of matched pairs).

sum(matched.distances)=11
(within 0.028 of optimum).
Percentiles of matched distances:
   0%   50%   95%  100%
0.018 1.140 1.820 1.850
Balance test overall result:
  chisquare df p.value
          5  5   0.416
```

Exercise. Try out at least 3 different combinations of propensity score and Mahalanobis matching. Identify the matches that gives:

1. the "best balance overall," as measured by chisquare statistics;
2. the "best balance overall," as indicated by the largest standardized difference among the covariates;
3. the best balance on `date` and `cap`, as measured by the larger of the standardized differences for these two variables; and
4. the best balance on `date` and `cap`, as measured by the larger of the standardized differences for these two variables, among those matches for which the overall imbalance p-value is no more than .1.

5 Other topics in matching

5.1 Full matching and matching with a varying number of controls

Try out for yourself and compare:

```
> summary(fullmatch(match_on(pr~date+cap, data=nuke.nopt), min=1))
> summary(fullmatch(match_on(pr~date+cap, data=nuke.nopt), min=2, max=3))
> summary(fullmatch(match_on(pr~date+cap, data=nuke.nopt)))
```

5.2 Subclassification before matching

Recall that the data set `nuclearplants` had 32 observations, 6 of which we excluded. These were plants built under "partial turnkey" guarantees (`pt==1`), for which costs are difficult to compare with other plants. We might include the excluded plants by matching them only among themselves. Then we need to subclassify prior to matching.

This is a common and useful operation, and `optmatch` is designed to help you do it.

```
> ptxm <- exactMatch(pr~pt, data=nuclearplants)
> mhd2 <- match_on(pr~date+cap+pscore, within=ptxm, data=nuclearplants)
> psd2 <- match_on(psm) + ptxm
> mhd2 + caliper(psd2, 1)

$`0`
       control
treated   H    I   J   K   L   M    N   O   P   Q    R    S    T    U    V    W
      A Inf  Inf Inf Inf Inf Inf  Inf Inf Inf Inf  Inf  Inf  Inf  Inf  Inf  Inf
      B Inf 0.79 Inf Inf Inf Inf  Inf Inf Inf Inf  Inf 1.46 1.59 1.91 1.18  Inf
      C Inf 1.69 1.6 Inf Inf   2 0.18 1.5 Inf 1.6 1.30 0.72 0.42  Inf  Inf 0.64
      D Inf  Inf Inf Inf Inf Inf  Inf Inf Inf Inf  Inf  Inf  Inf  Inf 3.26  Inf
      E Inf 1.57 Inf Inf Inf Inf  Inf Inf Inf Inf 0.61 1.06 1.50 1.32 0.59  Inf
      F Inf  Inf Inf Inf Inf Inf  Inf Inf Inf Inf  Inf  Inf  Inf 1.41 2.23  Inf
      G Inf 3.09 Inf Inf Inf Inf  Inf Inf Inf Inf  Inf 1.19  Inf 0.58 1.69  Inf
       control
treated   X   Y   Z
      A Inf Inf Inf
      B Inf Inf Inf
      C 1.3 Inf 3.0
      D Inf Inf Inf
      E Inf Inf 2.3
      F Inf Inf Inf
      G Inf Inf Inf

$`1`
       control
treated    d   e   f
      a 0.97 Inf Inf
      b  Inf Inf Inf
      c 0.92 Inf Inf
```

5.3 Using R for matching and another program for matched analysis

If you prefer to do your main work in another statistical package, you can use R for matching and balance assessment, and then re-import the matched data back into your preferred package for the main analysis. A typical work flow might look something like this:

1. Load data into your primary statistical package.
2. Preprocess data (handle missing values, combine variables into single measures, etc).
3. Export data in a format R can read.
4. Load data into R.
5. Perform matching and balance testing.
6. Append matches to your data and export in a format your statistical package can read.
7. Load appended data into your primary package and perform analyses.

You already know how to perform the matching and balance testing, so what remains are the import and export steps, along with appending your matches to your original data.

When importing and exporting data, you must select a data format. In a broad sense, you have two options: proprietary data formats (for example .dta for Stata) or open standards (for example .csv — Comma Separated Values). The advantage of proprietary formats is that they may include additional information for R to use, such as the labels on a categorical variable. The potential pitfall is that R might not know how to read your particular file type. Open formats like CSV are easy to read and write, but you may lose variable names, labels, or other special forms of data. For reading and writing proprietary formats, consider the use of the `foreign` library in R. See for example `read.dta`:

```
> library(foreign)
> ?read.dta
```

For much more detail on this topic, see the "R Data Import/Export" manual. If you open R help in an html browser — entering

```
> help.start()
```

at the command line does this, but you'll also find menu options to do it — then this is one of the first options you'll be offerred, at left and near the middle of the page.

If you use CSV files, consult your statistical package's manuals for details on how to export your data as a CSV. Once you have exported your data into a file, open the file (using a text editor, not Microsoft Word) to see if there is a header row of variable names. Then, to import your data use the following:

```
> my.plants <- read.csv("nuclearplants.csv", header = TRUE)
```

If you do not have a header on your data change to `header = FALSE`.

At this point, you may proceed to conduct your matching as usual (for example using an example from earlier in this document). When you are done, you will need to append your match to your original data. The safest way to do this is using the following code snippet:

```
> plant.match <- pairmatch(match_on(pr ~ cap, data = my.plants))
> my.plants.extended <- cbind(my.plants, matches = plant.match[rownames(my.plants)])
```

(If you've got R version 2.13 or later, the second line could be replaced with

```
> my.plants.extended <- data.frame(my.plants, matches = plant.match, check.rows=T)
```

.) Finally, export your data as a .csv file (or a proprietary format if that is what you are using):

```
> write.csv(my.plants.extended, file = "nuclearplants-with-matches.csv")
```

As one final note, many of the balance tests included in `RItools` are available to Stata users directly from Stata. Information is available at the `RItools` webpage: `http://www.jakebowers.org/RItools.html`.

5.4 Trying it out on your own

For another interesting (toy) data set, do

```
> data(tli, package="xtable")
> data(tli, package="xtable")
> head(tli)

  grade sex disadvg ethnicty tlimth
1     6   M     YES HISPANIC     43
2     7   M      NO    BLACK     88
3     5   F     YES HISPANIC     34
4     3   M     YES HISPANIC     65
5     8   M     YES    WHITE     75
6     5   M      NO    BLACK     74
```

You might compare test scores for kids with `disadvg=="YES"` to those of kids with `disadvg=="NO"` using propensity matching, in some combination with mahalanobis matching and caliper matching. A check of propensity overlap may inform your decision as to whether to include a propensity caliper. Be sure to check for balance, and do check the structure of the matched sets.

Two sources of *real* data can be gotten as follows. The "Lalonde" data set, discussed by Lalonde in 1986 and more recently by Dehejia and Wahba (1999, *JASA*) and by Smith and Todd (2005, *J. Econom.*), is bundled with several R packages, including "`arm`" and "`Matching`." To get it:

```
> library(arm)
> data(lalonde)
> help("lalonde")
```

Connors et al's (1996, *J. Am. Med. Assoc.*) brought propensity score methods to the attention of the medical field broadly. It's bundled with the "`Hmisc`." package. You'll need a connection to the internet to get it.

First, make sure the `Hmisc` package is installed: try

```
> library(Hmisc)
```

and if this generates a complaint then do:

```
> install.packages("Hmisc", dep=T)
```

After that's done, just do:

```
> library(Hmisc)
> getHdata(rhc, what="all")
```