Propensity Score Matching

This notebook illustrates how to do propensity score matching in Python. Original dataset available at: http://biostat.mc.vanderbilt.edu/wiki/Main/DataSets (http://biostat.mc.vanderbilt.edu/wiki/Main/DataSets) (search for rhs, download csv file for data, html file for file description) Fro a presentation and key results on the topic, see: http://www.mc.vanderbilt.edu/crc/workshop_files/2008-04-11.pdf (http://www.mc.vanderbilt.edu/crc/workshop_files/2008-04-11.pdf)

Import key packages

In [61]:

```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import seaborn as sns
```

In [62]:

```
# allow graphs
%matplotlib inline
```

Import data to a dataframe (called df)

In [63]:

```
#df = pd.read_csv(r'rhc.csv')
df = pd.read_csv(r'http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/rhc.csv')
```

Have a look at the data

In [64]:

```
df.head(5)
```

Out[64]:

	Unnamed: 0	cat1	cat2	ca	sadmdte	dschdte	dthdte	lstctdte	death	ca
0	1	COPD	NaN	Yes	11142	11151.0	NaN	11382	No	
1	2	MOSF w/Sepsis	NaN	No	11799	11844.0	11844.0	11844	Yes	
2	3	MOSF w/Malignancy	MOSF w/Sepsis	Yes	12083	12143.0	NaN	12400	No	
3	4	ARF	NaN	No	11146	11183.0	11183.0	11182	Yes	
4	5	MOSF w/Sepsis	NaN	No	12035	12037.0	12037.0	12036	Yes	

5 rows × 63 columns

Clean the Data

In [65]:

```
df=df.replace(to_replace = 'Yes', value = 1)
df=df.replace(to_replace = 'No', value = 0)
```

In [66]:

```
df.head()
```

Out[66]:

	Unnamed: 0	cat1	cat2	ca	sadmdte	dschdte	dthdte	lstctdte	death	car
0	1	COPD	NaN	1	11142	11151.0	NaN	11382	0	
1	2	MOSF w/Sepsis	NaN	0	11799	11844.0	11844.0	11844	1	
2	3	MOSF w/Malignancy	MOSF w/Sepsis	1	12083	12143.0	NaN	12400	0	
3	4	ARF	NaN	0	11146	11183.0	11183.0	11182	1	
4	5	MOSF w/Sepsis	NaN	0	12035	12037.0	12037.0	12036	1	

5 rows × 63 columns

In [67]:

```
# how many received treatment?
df.swang1.value_counts(normalize=True)
```

Out[67]:

```
No RHC    0.61918
RHC       0.38082
Name: swang1, dtype: float64
```

In [68]:

```
# how many died in the treatment group (percent)
df['treated'] = 0
df['treated'] = df['treated'].where(df.swang1=='No RHC',1)
```

In [69]:

```
df['treated'] = np.where(df.swang1 == 'RHC', 1, 0)
```

In [70]:

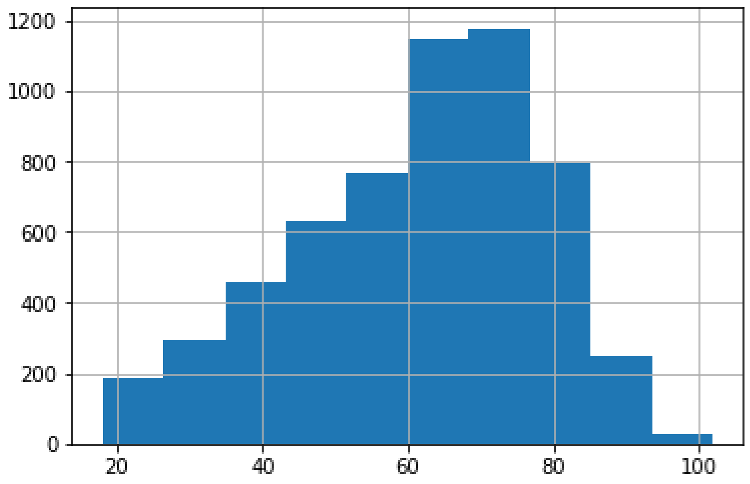
```
df.groupby('treated')['death'].mean()
```

Out[70]:

```
treated
0    0.629682
1    0.680403
Name: death, dtype: float64
```

In [71]:

```
# age distribution
df.age.hist();
```



In [72]:

```
pd.cut(df.age, 5).head(5)
```

Out[72]:

```
0    (68.326, 85.087]
1    (68.326, 85.087]
2    (34.803, 51.564]
3    (68.326, 85.087]
4    (51.564, 68.326]
Name: age, dtype: category
Categories (5, interval[float64]): [(17.958, 34.803] < (34.803, 51.564] <
(51.564, 68.326] < (68.326, 85.087] < (85.087, 101.848]]
```

In [73]:

```
df['agegrp'] = pd.qcut(df.age,10)
```

In [74]:

```
df.groupby('agegrp').size()
```

Out[74]:

```
agegrp
(18.041, 36.972]     574
(36.972, 46.196]     573
(46.196, 53.386]     574
(53.386, 59.545]     573
(59.545, 64.047]     574
(64.047, 68.068]     573
(68.068, 71.899]     573
(71.899, 76.025]     574
(76.025, 80.883]     573
(80.883, 101.848]    574
dtype: int64
```

Logit regression (to compare the results with Propensity matching method)

In [75]:

```
model = 'death ~ age + sex + edu + treated'
reg_results = smf.logit(formula=model, data=df).fit()
reg_results.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.622953
         Iterations 5
```

Out[75]:

Logit Regression Results

Dep. Variable:	death	**No. Observations:**	5735
Model:	Logit	**Df Residuals:**	5730
Method:	MLE	**Df Model:**	4
Date:	Wed, 07 Nov 2018	**Pseudo R-squ.:**	0.03875
Time:	12:29:50	**Log-Likelihood:**	-3572.6
converged:	True	**LL-Null:**	-3716.7
		LLR p-value:	4.112e-61

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-1.2474	0.168	-7.406	0.000	-1.577	-0.917
sex[T.Male]	0.1074	0.057	1.878	0.060	-0.005	0.219
age	0.0277	0.002	15.970	0.000	0.024	0.031
edu	0.0030	0.009	0.327	0.743	-0.015	0.021
treated	0.2526	0.059	4.278	0.000	0.137	0.368

In [76]:

```
np.exp(reg_results.params)
```

Out[76]:

```
Intercept      0.287255
sex[T.Male]    1.113332
age            1.028086
edu            1.003032
treated        1.287403
dtype: float64
```

In [77]:

```
df.edu.describe()
```

Out[77]:

```
count    5735.000000
mean       11.678461
std         3.145831
min         0.000000
25%        10.000000
50%        12.000000
75%        13.000000
max        30.000000
Name: edu, dtype: float64
```

Estimate propensity score

In [78]:

```
df.sex = df.sex.replace('Male', 0)
df.sex = df.sex.replace('Female', 1)
```

In [79]:

```
df['male'] = np.where(df.sex == 0, 1, 0)
```

In [80]:

```
model = 'treated ~ age + male +edu'
propensity = smf.logit(formula=model, data = df).fit()
propensity.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.662209
         Iterations 4
```

Out[80]:

Logit Regression Results

Dep. Variable:	treated	**No. Observations:**	5735
Model:	Logit	**Df Residuals:**	5731
Method:	MLE	**Df Model:**	3
Date:	Wed, 07 Nov 2018	**Pseudo R-squ.:**	0.003394
Time:	12:29:50	**Log-Likelihood:**	-3797.8
converged:	True	**LL-Null:**	-3810.7
		LLR p-value:	1.017e-05

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.7441	0.160	-4.645	0.000	-1.058	-0.430
age	-0.0027	0.002	-1.648	0.099	-0.006	0.001
male	0.1858	0.055	3.374	0.001	0.078	0.294
edu	0.0273	0.009	3.111	0.002	0.010	0.045

Check overlap

In [81]:

```
df.groupby('treated').size()
df.groupby('treated').male.mean()
```

Out[81]:

```
treated
0    0.539003
1    0.585165
Name: male, dtype: float64
```

In [109]:

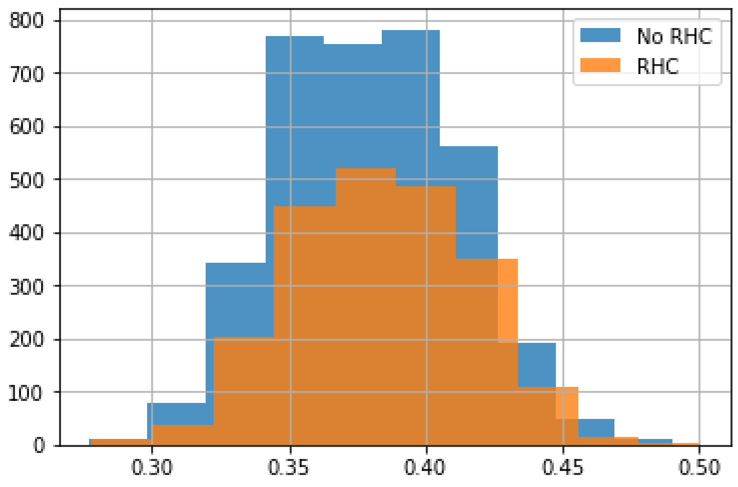
```
df['propensity'] = propensity.predict()
```

In [83]:

```
import matplotlib.pyplot as plt
df.groupby('treated')['propensity'].hist(alpha=0.8)
plt.legend(['No RHC','RHC'])
```

Out[83]:

```
<matplotlib.legend.Legend at 0x7f367ea43400>
```



In [84]:

```
df.groupby('treated')['propensity'].mean()
```

Out[84]:

```
treated
0    0.379108
1    0.383603
Name: propensity, dtype: float64
```

In [85]:

```
# Are the treated different in terms of age, sex and edu?
df.groupby('treated')['age', 'male', 'edu'].mean()
```

Out[85]:

	age	male	edu
treated			
0	61.760926	0.539003	11.569005
1	60.749836	0.585165	11.856428

Do matching (ten groups)

In [86]:

```
df.propensity.head()
```

Out[86]:

```
0    0.396208
1    0.347837
2    0.381078
3    0.331200
4    0.384368
Name: propensity, dtype: float64
```

In [92]:

```
# Everybody between 0 and 0.09999999 will end up in group 0, 0.1 to 0.1999999999 in gro
up 1

df['group'] = (df.propensity*10).astype(int)
```

In [93]:

```
# percentages who die in the different groups
df.groupby('group')['death'].mean()
```

Out[93]:

```
group
2    0.695652
3    0.670981
4    0.595983
5    1.000000
Name: death, dtype: float64
```

In [94]:

```
# distinguish between treated and untreated in the different groups and see how many wh
o die

df.groupby(['group', 'treated'])['death'].mean()
```

Out[94]:

```
group  treated
2      0          0.538462
       1          0.900000
3      0          0.649882
       1          0.707317
4      0          0.579317
       1          0.619799
5      1          1.000000
Name: death, dtype: float64
```

In [95]:

```
# same thing, but easier to see if we stack it
df.groupby(['group', 'treated'])['death'].mean().unstack('treated')
```

Out[95]:

treated	0	1
group		
2	0.538462	0.900000
3	0.649882	0.707317
4	0.579317	0.619799
5	NaN	1.000000

Calculate overall average effect of treatment (on the treated)

In [96]:

```
psTable=df.groupby(['group', 'treated'])['death'].mean().unstack('treated')
psTable
```

Out[96]:

treated	0	1
group		
2	0.538462	0.900000
3	0.649882	0.707317
4	0.579317	0.619799
5	NaN	1.000000

In [97]:

```
psTable.columns = ['untreated', 'treated']
```

In [98]:

```
psTable['difference'] = psTable.treated - psTable.untreated
```

In [99]:

```
psTable
```

Out[99]:

	untreated	treated	difference
group			
2	0.538462	0.900000	0.361538
3	0.649882	0.707317	0.057435
4	0.579317	0.619799	0.040482
5	NaN	1.000000	NaN

In [100]:

```
# end result, average effect of treatment on mortality

psTable.difference.mean()
```

Out[100]:

```
0.1531518073699852
```

In [101]:

```
psTable.mean()
```

Out[101]:

```
untreated     0.589220
treated       0.806779
difference    0.153152
dtype: float64
```

In [102]:

```
# how many individuals are in the different groups?
df.groupby(['group', 'treated']).size().unstack('treated')
```

Out[102]:

treated	0	1
group		
2	13.0	10.0
3	2542.0	1476.0
4	996.0	697.0
5	NaN	1.0

In [103]:

```
# what is the balance (age, edu etc in the different groups)

df.groupby(['group', 'treated'])['age', 'sex', 'edu'].mean().unstack('treated')
```

Out[103]:

	age		sex		edu	
treated	**0**	**1**	**0**	**1**	**0**	**1**
group						
2	79.668124	70.251468	1.000000	1.000000	2.846154	1.700000
3	66.084293	64.332149	0.619591	0.590786	10.725295	10.819763
4	50.493061	53.020573	0.049197	0.034433	13.836182	14.175709
5	NaN	65.535950	NaN	0.000000	NaN	27.000000

In [104]:

```
# overall group average

df.groupby(['group', 'treated'])['age', 'sex', 'edu'].mean().unstack('treated').mean()
```

Out[104]:

```
     treated
age  0          65.415159
     1          63.285035
sex  0           0.556263
     1           0.406305
edu  0           9.135877
     1          13.423868
dtype: float64
```

Examples on how to group variables

In [105]:

```
grp_name =['0-29','30-59','60-'] #name of groups
bins = [-1,29,59,222] #prespecified age intervals
df['agegrp'] = pd.cut(df.age, bins = bins,labels = grp_name)
```

In [106]:

```
df['agegrp'] = (df.age/10).astype(int)
df['agegrp_label'] = pd.cut(df.age, [-1,20,50,70,999], labels = ['Young','Adults','Old'
, 'Super old']) #prespecified age intervals
```

In [107]:

```
df['agegrp'] = pd.qcut(df.age, 3) #equal number of patients in each of the groups
df['agegrp_label'] = pd.qcut(df.age, 3, labels = ['Young','Adults','Old'])
```

Deciding which category in a categorical variable that should be the reference category. Example show how to make male the reference category in the sex variable

In [ ]:

```
model = 'death ~ age + C(sex, Treatment(reference="Male")) + edu + treated'
reg_results = smf.logit(formula=model, data=df).fit()
reg_results.summary()
```

sex refers to the column while male refers to your choice of reference category

In [ ]:

```
```

In [ ]:

```
```