imputation

October 16, 2018

0.1 Handling Missing Data

In this Jupyter notebook we will attempt question 1 of Assignment 2, which is to work on imputing Age and Gender. We will be using pandas to load the data, and also to perform our imputation for us, while also discussing what approach we will be taking.

Let us start with reading the data and loading our files

```
In [1]: import pandas as pd

In [2]: bestincome = pd.read_csv('BestIncome.txt')

In [3]: surveyincome = pd.read_csv('SurvIncome.txt')

In [4]: bestincome
```

Out[4]:

	lab_inc	cap_inc	height	weight
0	52655.605507	9279.509829	64.568138	152.920634
1	70586.979225	9451.016902	65.727648	159.534414
2	53738.008339	8078.132315	66.268796	152.502405
3	55128.180903	12692.670403	62.910559	149.218189
4	44482.794867	9812.975746	68.678295	152.726358
5	55394.631435	10769.461417	67.370550	151.602678
6	62627.896258	9730.261003	64.547689	151.421983
7	54936.555868	8712.627564	63.080352	153.917777
8	52730.248945	9260.989766	63.417904	147.327536
9	60525.267381	10310.988854	65.310226	154.179314
10	52939.306726	9318.170569	66.577360	138.514938
11	65368.966785	8137.377983	65.416442	150.882211
12	64518.650046	15055.395681	64.776401	152.626530
13	50441.688765	9743.244116	69.019113	153.272130
14	60680.505706	13336.610305	67.642602	156.561194
15	51182.193744	7828.334577	66.508997	151.376282
16	51778.464236	5090.959542	64.424632	157.941301
17	43148.362586	9113.554143	68.265576	160.202517
18	56445.182706	9893.480254	62.625146	168.247523
19	53665.754187	10050.356411	62.728510	154.734221
20	42274.625095	10942.178080	65.740133	168.984497
21	49039.112709	11970.307373	68.185663	172.660528

```
22    55675.003397  10131.222466  63.173962  145.048888
23    52500.211333  10557.126469  65.554760  155.855650
24    41768.686282   9946.434932  60.990848  151.661267
25    55797.114339   7171.218631  66.873181  145.007796
26    66916.703649   9723.487751  62.115249  134.842964
27    57757.960553   9018.991672  63.689874  139.842300
28    53841.112391  11251.930995  66.697991  157.004523
29    62163.018768   7788.967219  64.431583  144.984515
...            ...           ...        ...         ...
9970  59397.036181   9365.956541  62.665072  139.588793
9971  49324.118612   8861.021667  65.716819  157.220513
9972  50747.219007   9478.404450  65.638505  153.771960
9973  53200.651406   7463.569890  66.915694  150.945506
9974  54247.298670   6968.734057  63.844270  133.346272
9975  52002.140627   7508.570636  67.320795  157.471407
9976  49283.022302  10430.520851  66.574066  150.883483
9977  59174.106493   7358.315166  68.464577  150.248427
9978  48741.143171   8435.860615  64.951102  149.913997
9979  55760.987949   5295.896397  67.651121  146.308846
9980  47430.407011   8193.535129  64.515317  145.410390
9981  55614.658063  10963.837631  63.254930  119.497027
9982  66326.901487  14463.127375  65.106509  138.779036
9983  65408.846526  11427.320320  64.451020  163.693557
9984  47206.177567   8711.462491  64.159775  161.054623
9985  51435.183670  12596.929959  66.053950  149.094935
9986  67002.137376  10403.604084  64.941448  147.752410
9987  58637.877655  15076.887813  63.232478  132.209348
9988  60984.232324  10182.804787  62.806832  166.376524
9989  57024.365041   6874.385067  64.804038  146.795999
9990  56251.964522  10826.684057  68.232857  151.760895
9991  52591.679552  10734.561169  60.951158  144.530906
9992  53130.264646  10062.310008  61.546878  137.753285
9993  81154.055874   8505.572396  66.400586  149.112161
9994  71192.465909  10943.497348  62.950019  156.766404
9995  51502.225233  14786.050723  66.781187  154.645212
9996  52624.117104  11048.811747  64.499036  165.868002
9997  50725.310645  13195.218100  64.508873  154.657639
9998  56392.824076   8470.592718  62.161556  145.498194
9999  44274.098164  12765.748454  64.974145  135.936862

[10000 rows x 4 columns]
```

In [5]:
```
surveyincome
```

Out[5]:
```
          tot_inc      weight        age  gender
0    63642.513655  134.998269  46.610021     1.0
1    49177.380692  134.392957  48.791349     1.0
2    67833.339128  126.482992  48.429894     1.0
```

3	62962.266217	128.038121	41.543926	1.0
4	58716.952597	126.211980	41.201245	1.0
5	45708.541543	116.023500	44.504057	1.0
6	73594.659828	111.142211	52.943038	1.0
7	66831.999693	140.317751	34.138021	1.0
8	63690.343092	138.023277	40.662877	1.0
9	59965.928189	124.794773	32.909093	1.0
10	71616.698088	133.904314	58.352106	1.0
11	58792.143221	128.032797	42.939641	1.0
12	52206.065942	122.555781	35.746997	1.0
13	61864.024802	129.626024	40.584096	1.0
14	41106.890518	134.002953	51.502829	1.0
15	56756.679667	121.764020	47.304075	1.0
16	75706.319716	127.030443	44.741404	1.0
17	64184.541556	140.253942	35.897930	1.0
18	67915.377742	127.781118	47.234274	1.0
19	73308.611215	122.160024	44.745897	1.0
20	55638.978117	121.722019	54.227057	1.0
21	62916.446966	117.204411	46.066650	1.0
22	70096.498550	124.125143	45.009500	1.0
23	64223.891611	140.520811	45.203856	1.0
24	60974.984877	117.634542	48.440532	1.0
25	61249.757831	132.866153	38.031742	1.0
26	55010.940184	129.319867	45.205707	1.0
27	59613.411553	136.649460	44.407637	1.0
28	51778.294378	113.475392	50.269154	1.0
29	58876.674426	129.532950	43.544025	1.0
..	...	...	...	...
970	86171.809808	172.955760	38.925270	0.0
971	75800.419643	168.760920	41.133830	0.0
972	71932.668909	171.379317	43.237950	0.0
973	63256.544651	173.791050	43.733118	0.0
974	65075.052136	157.065695	42.014467	0.0
975	62453.362212	162.226990	43.904954	0.0
976	89481.011621	177.978784	40.022632	0.0
977	71450.812161	159.018055	43.379238	0.0
978	70658.655015	164.406240	43.709366	0.0
979	68286.997107	175.945021	39.507692	0.0
980	79030.131179	181.274495	56.044716	0.0
981	66643.032203	166.845651	41.779962	0.0
982	71876.613837	178.691654	39.302694	0.0
983	73758.398881	160.069073	50.678034	0.0
984	73233.041415	168.557464	51.787934	0.0
985	72069.356943	147.239351	53.287615	0.0
986	77328.407327	167.167770	31.053877	0.0
987	53072.797015	165.618622	51.823863	0.0
988	80487.848762	187.368164	50.303873	0.0
989	62523.793331	164.994353	51.128326	0.0

```
990  75459.504804  171.963494  42.993941     0.0
991  76917.829315  164.063802  33.689170     0.0
992  75608.257582  169.640352  34.965142     0.0
993  63863.192160  173.806056  37.365372     0.0
994  60956.258144  170.239463  45.765461     0.0
995  61270.538697  184.930002  46.356881     0.0
996  59039.159876  180.482304  50.986966     0.0
997  67967.188804  156.816883  40.965268     0.0
998  79726.914251  158.935050  41.190371     0.0
999  71005.223603  169.067695  48.480007     0.0

[1000 rows x 4 columns]
```

0.2 Imputation Technique

There are many ways to go about handling imputation problems. Some more straightforward techniques might involve just dropping missing values, or replacing it with the mean value found. The most popular method may just be using a linear regression model, and then using the coefficients to later simulate the values missing. We will attempt this method! The formula for the same is

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i \tag{1}$$

Here, Y_i is the variable we wish to impute, and β_0 and β_1 are the coefficients we will use to estimate our variable.

Let us use scikit-learn for our regression model.

0.3 Fitting our Model

```
In [6]: from sklearn import linear_model

In [7]: model_age = linear_model.LinearRegression()
        model_age.fit(surveyincome[['tot_inc', 'weight']], surveyincome['age'])

/Users/bhargavvader/open_source/persp-analysis_A18/venv/lib/python3.5/site-packages/sklearn/line
  linalg.lstsq(X, y)


Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                 normalize=False)

In [8]: model_age.coef_

Out[8]: array([ 2.52022048e-05, -6.72214424e-03])
```

We can see the coefficients of our model for predicting age. These will be used in our estimation.

```
In [9]: model_gender = linear_model.LogisticRegression()
        model_gender.fit(surveyincome[['tot_inc', 'weight']], surveyincome['gender'])
```

```
/Users/bhargavvader/open_source/persp-analysis_A18/venv/lib/python3.5/site-packages/sklearn/line
  FutureWarning)
```

```
Out[9]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                intercept_scaling=1, max_iter=100, multi_class='warn',
                n_jobs=None, penalty='l2', random_state=None, solver='warn',
                tol=0.0001, verbose=0, warm_start=False)
```

```
In [10]: model_gender
```

```
Out[10]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                 intercept_scaling=1, max_iter=100, multi_class='warn',
                 n_jobs=None, penalty='l2', random_state=None, solver='warn',
                 tol=0.0001, verbose=0, warm_start=False)
```

We can see the coefficients of our model for predicting age. These will be used in our estimation.

0.4 Imputing our Values

We will now use our models and our coefficients to impute the values. Let us create a temporary column called Total Income to be used when creating our fit. We will be using our models predict function to do the imputation.

```
In [11]: bestincome['tot_inc'] = bestincome['cap_inc'] + bestincome['lab_inc']
```

```
In [12]: bestincome['imputed_age'] = model_age.predict(bestincome[['tot_inc', 'weight']])
```

```
In [13]: bestincome['imputed_gender'] = model_gender.predict(bestincome[['tot_inc', 'weight']])
```

```
In [14]: bestincome = bestincome.drop(columns=['tot_inc'])
```

```
In [15]: bestincome
```

```
Out[15]:            lab_inc       cap_inc     height      weight  imputed_age  \
         0     52655.605507   9279.509829  64.568138  152.920634    44.742614
         1     70586.979225   9451.016902  65.727648  159.534414    45.154387
         2     53738.008339   8078.132315  66.268796  152.502405    44.742427
         3     55128.180903  12692.670403  62.910559  149.218189    44.915836
         4     44482.794867   9812.975746  68.678295  152.726358    44.551391
         5     55394.631435  10769.461417  67.370550  151.602678    44.858053
         6     62627.896258   9730.261003  64.547689  151.421983    45.015372
         7     54936.555868   8712.627564  63.080352  153.917777    44.779109
         8     52730.248945   9260.989766  63.417904  147.327536    44.781626
         9     60525.267381  10310.988854  65.310226  154.179314    44.958481
         10    52939.306726   9318.170569  66.577360  138.514938    44.847575
         11    65368.966785   8137.377983  65.416442  150.882211    45.047937
         12    64518.650046  15055.395681  64.776401  152.626530    45.189131
         13    50441.688765   9743.244116  69.019113  153.272130    44.696142
```

14	60680.505706	13336.610305	67.642602	156.561194	45.022634
15	51182.193744	7828.334577	66.508997	151.376282	44.679289
16	51778.464236	5090.959542	64.424632	157.941301	44.581197
17	43148.362586	9113.554143	68.265576	160.202517	44.449878
18	56445.182706	9893.480254	62.625146	168.247523	44.750563
19	53665.754187	10050.356411	62.728510	154.734221	44.775308
20	42274.625095	10942.178080	65.740133	168.984497	44.414909
21	49039.112709	11970.307373	68.185663	172.660528	44.586590
22	55675.003397	10131.222466	63.173962	145.048888	44.893089
23	52500.211333	10557.126469	65.554760	155.855650	44.751167
24	41768.686282	9946.434932	60.990848	151.661267	44.493513
25	55797.114339	7171.218631	66.873181	145.007796	44.821844
26	66916.703649	9723.487751	62.115249	134.842964	45.234735
27	57757.960553	9018.991672	63.689874	139.842300	44.952553
28	53841.112391	11251.930995	66.697991	157.004523	44.794748
29	62163.018768	7788.967219	64.431583	144.984515	44.998004
...	...	...	...	...	...
9970	59397.036181	9365.956541	62.665072	139.588793	45.004310
9971	49324.118612	8861.021667	65.716819	157.220513	44.619202
9972	50747.219007	9478.404450	65.638505	153.771960	44.693808
9973	53200.651406	7463.569890	66.915694	150.945506	44.723861
9974	54247.298670	6968.734057	63.844270	133.346272	44.856073
9975	52002.140627	7508.570636	67.320795	157.471407	44.650922
9976	49283.022302	10430.520851	66.574066	150.883483	44.700319
9977	59174.106493	7358.315166	68.464577	150.248427	44.876439
9978	48741.143171	8435.860615	64.951102	149.913997	44.642910
9979	55760.987949	5295.896397	67.651121	146.308846	44.764926
9980	47430.407011	8193.535129	64.515317	145.410390	44.634043
9981	55614.658063	10963.837631	63.254930	119.497027	45.084315
9982	66326.901487	14463.127375	65.106509	138.779036	45.312861
9983	65408.846526	11427.320320	64.451020	163.693557	45.045736
9984	47206.177567	8711.462491	64.159775	161.054623	44.536282
9985	51435.183670	12596.929959	66.053950	149.094935	44.821180
9986	67002.137376	10403.604084	64.941448	147.752410	45.167249
9987	58637.877655	15076.887813	63.232478	132.209348	45.178711
9988	60984.232324	10182.804787	62.806832	166.376524	44.884826
9989	57024.365041	6874.385067	64.804038	146.795999	44.833272
9990	56251.964522	10826.684057	68.232857	151.760895	44.880038
9991	52591.679552	10734.561169	60.951158	144.530906	44.834070
9992	53130.264646	10062.310008	61.546878	137.753285	44.876262
9993	81154.055874	8505.572396	66.400586	149.112161	45.466934
9994	71192.465909	10943.497348	62.950019	156.766404	45.225868
9995	51502.225233	14786.050723	66.781187	154.645212	44.840730
9996	52624.117104	11048.811747	64.499036	165.868002	44.699376
9997	50725.310645	13195.218100	64.508873	154.657639	44.780974
9998	56392.824076	8470.592718	62.161556	145.498194	44.866308
9999	44274.098164	12765.748454	64.974145	135.936862	44.733410

	imputed_gender
0	0.0
1	1.0
2	0.0
3	1.0
4	0.0
5	0.0
6	1.0
7	0.0
8	0.0
9	1.0
10	0.0
11	1.0
12	1.0
13	0.0
14	1.0
15	0.0
16	0.0
17	0.0
18	0.0
19	0.0
20	0.0
21	0.0
22	1.0
23	0.0
24	0.0
25	0.0
26	1.0
27	1.0
28	0.0
29	1.0
...	...
9970	1.0
9971	0.0
9972	0.0
9973	0.0
9974	1.0
9975	0.0
9976	0.0
9977	0.0
9978	0.0
9979	0.0
9980	0.0
9981	1.0
9982	1.0
9983	1.0
9984	0.0
9985	0.0

```
9986            1.0
9987            1.0
9988            0.0
9989            0.0
9990            0.0
9991            0.0
9992            1.0
9993            1.0
9994            1.0
9995            0.0
9996            0.0
9997            0.0
9998            0.0
9999            0.0

[10000 rows x 6 columns]
```

0.5 Describing Variables

Let us now look at some statistics describing our imputed variables.

```
In [16]: bestincome['imputed_age'].describe()

Out[16]: count    10000.000000
         mean        44.890828
         std          0.219150
         min         43.976495
         25%         44.743776
         50%         44.886944
         75%         45.038991
         max         45.703819
         Name: imputed_age, dtype: float64

In [17]: bestincome['imputed_gender'].describe()

Out[17]: count    10000.000000
         mean         0.471700
         std          0.499223
         min          0.000000
         25%          0.000000
         50%          0.000000
         75%          1.000000
         max          1.000000
         Name: imputed_gender, dtype: float64
```

0.6 Correlation Matrix

Our last step is to print the correlation matrix, which we will do using the `corr` method.

```
In [18]: bestincome.corr()
```

```
Out[18]:                  lab_inc   cap_inc    height    weight  imputed_age  \
         lab_inc         1.000000  0.005325  0.002790  0.004507     0.924053
         cap_inc         0.005325  1.000000  0.021572  0.006299     0.234159
         height          0.002790  0.021572  1.000000  0.172103    -0.045083
         weight          0.004507  0.006299  0.172103  1.000000    -0.300288
         imputed_age     0.924053  0.234159 -0.045083 -0.300288     1.000000
         imputed_gender  0.677675  0.176901 -0.066972 -0.382659     0.784260

                         imputed_gender
         lab_inc                0.677675
         cap_inc                0.176901
         height                -0.066972
         weight                -0.382659
         imputed_age            0.784260
         imputed_gender         1.000000
```