Big Data Regression Computational Techniques Execution Time Comparison

We have previously seen how to do regression for comparatively small sets of data. What if, however, we had a much larger dataset that could not even be loaded into R directly? How could we estimate the regression coefficients in this case? These are all questions that this exercise will give us answers to.

First of all, we create a huge *Excel* file that contains our data. In total we have 2000000 rows and 116 columns (due to the Student ID number). The size of it is 3.92 GB, which means that should we run it directly in *R*, it will crash our program (and probably our system as well). What we need to do, is to get the estimates of coefficients, but the size of the data is making it a more difficult task than it should be. Thankfully, there are techniques and *R* packages which we can use to make this errand much easier for us.

Some of these methods are explored in the questions below.

Part 1: **We shall use the command $\boldsymbol{biglm.ffdf}()$ of the $\boldsymbol{ffbase}$ library to estimate the regression coefficients.**

This method uses specific libraries (all listed in the *R Script*) to estimate the coefficients. After we read the data, without loading it into *R* and running our regression, we get the following output:

```
Large data regression model: bigglm(as.formula(myFormula), data = brgdata, 
chunksize = 5000,
    sandwich = FALSE)
Sample size =  2e+06
               Coef    (95%     CI)    SE     p
(Intercept) -1.7092 -1.7106 -1.7078 7e-04 0e+00
x1           3.1800  3.1786  3.1814 7e-04 0e+00
x2           0.6495  0.6481  0.6509 7e-04 0e+00
x3          -0.9343 -0.9357 -0.9329 7e-04 0e+00
x4           0.2779  0.2765  0.2793 7e-04 0e+00
x5           0.8452  0.8438  0.8466 7e-04 0e+00
x6          -0.4417 -0.4431 -0.4403 7e-04 0e+00
x7           0.7341  0.7327  0.7355 7e-04 0e+00
x8           2.3973  2.3959  2.3988 7e-04 0e+00
x9           0.0184  0.0170  0.0198 7e-04 0e+00
x10          0.1105  0.1091  0.1119 7e-04 0e+00
x11          1.0034  1.0020  1.0048 7e-04 0e+00
x12         -0.6052 -0.6066 -0.6038 7e-04 0e+00
x13          1.4784  1.4770  1.4799 7e-04 0e+00
x14          0.1752  0.1738  0.1766 7e-04 0e+00
x15          1.1472  1.1457  1.1486 7e-04 0e+00
x16         -1.8666 -1.8680 -1.8652 7e-04 0e+00
x17          1.1229  1.1214  1.1243 7e-04 0e+00
x18         -0.1800 -0.1814 -0.1785 7e-04 0e+00
```

x19	1.5741	1.5727	1.5755	7e-04	0e+00
x20	0.2798	0.2783	0.2812	7e-04	0e+00
x21	-0.1574	-0.1588	-0.1559	7e-04	0e+00
x22	-0.1426	-0.1440	-0.1412	7e-04	0e+00
x23	-2.7099	-2.7114	-2.7085	7e-04	0e+00
x24	-1.0259	-1.0273	-1.0245	7e-04	0e+00
x25	-0.2834	-0.2848	-0.2820	7e-04	0e+00
x26	-0.4077	-0.4091	-0.4063	7e-04	0e+00
x27	1.0675	1.0661	1.0690	7e-04	0e+00
x28	-1.2465	-1.2479	-1.2451	7e-04	0e+00
x29	1.8646	1.8632	1.8660	7e-04	0e+00
x30	0.0028	0.0014	0.0042	7e-04	1e-04
x31	0.4559	0.4545	0.4573	7e-04	0e+00
x32	2.0724	2.0710	2.0738	7e-04	0e+00
x33	-0.7346	-0.7360	-0.7332	7e-04	0e+00
x34	0.8299	0.8285	0.8314	7e-04	0e+00
x35	0.8859	0.8845	0.8873	7e-04	0e+00
x36	-0.4807	-0.4822	-0.4793	7e-04	0e+00
x37	1.0476	1.0462	1.0490	7e-04	0e+00
x38	-0.8034	-0.8048	-0.8020	7e-04	0e+00
x39	1.2253	1.2239	1.2267	7e-04	0e+00
x40	-0.8244	-0.8258	-0.8230	7e-04	0e+00
x41	-1.3336	-1.3350	-1.3322	7e-04	0e+00
x42	1.6627	1.6613	1.6641	7e-04	0e+00
x43	0.1419	0.1404	0.1433	7e-04	0e+00
x44	-0.7958	-0.7973	-0.7944	7e-04	0e+00
x45	0.9542	0.9528	0.9557	7e-04	0e+00
x46	-0.1105	-0.1119	-0.1091	7e-04	0e+00
x47	0.1087	0.1073	0.1102	7e-04	0e+00
x48	0.3277	0.3263	0.3291	7e-04	0e+00
x49	0.9692	0.9678	0.9706	7e-04	0e+00
x50	-0.5099	-0.5113	-0.5085	7e-04	0e+00
x51	0.2285	0.2271	0.2299	7e-04	0e+00
x52	4.1000	4.0986	4.1014	7e-04	0e+00
x53	0.5252	0.5238	0.5266	7e-04	0e+00
x54	0.7558	0.7544	0.7572	7e-04	0e+00
x55	-2.2579	-2.2594	-2.2565	7e-04	0e+00
x56	-2.5020	-2.5034	-2.5006	7e-04	0e+00
x57	-0.5881	-0.5895	-0.5866	7e-04	0e+00
x58	-1.8587	-1.8601	-1.8573	7e-04	0e+00
x59	0.0605	0.0591	0.0619	7e-04	0e+00
x60	0.2646	0.2632	0.2661	7e-04	0e+00
x61	4.5802	4.5788	4.5816	7e-04	0e+00
x62	0.0832	0.0818	0.0847	7e-04	0e+00
x63	-0.2352	-0.2366	-0.2338	7e-04	0e+00
x64	-1.3702	-1.3716	-1.3688	7e-04	0e+00
x65	1.3503	1.3489	1.3517	7e-04	0e+00
x66	0.3057	0.3043	0.3071	7e-04	0e+00
x67	2.2040	2.2026	2.2054	7e-04	0e+00
x68	-0.4783	-0.4797	-0.4768	7e-04	0e+00
x69	-0.3913	-0.3927	-0.3899	7e-04	0e+00
x70	0.9986	0.9972	1.0000	7e-04	0e+00
x71	-0.4374	-0.4388	-0.4360	7e-04	0e+00

```
x72          0.6544  0.6530  0.6558 7e-04 0e+00
x73          0.5708  0.5693  0.5722 7e-04 0e+00
x74         -0.8792 -0.8806 -0.8778 7e-04 0e+00
x75          0.9439  0.9425  0.9453 7e-04 0e+00
x76         -5.5141 -5.5155 -5.5127 7e-04 0e+00
x77          0.1967  0.1952  0.1981 7e-04 0e+00
x78         -2.2753 -2.2767 -2.2739 7e-04 0e+00
x79         -0.4998 -0.5012 -0.4983 7e-04 0e+00
x80         -0.8087 -0.8101 -0.8073 7e-04 0e+00
x81         -1.3883 -1.3897 -1.3869 7e-04 0e+00
x82          0.2325  0.2311  0.2339 7e-04 0e+00
x83         -0.4575 -0.4589 -0.4561 7e-04 0e+00
x84          0.4527  0.4513  0.4541 7e-04 0e+00
x85         -1.6430 -1.6444 -1.6416 7e-04 0e+00
x86         -0.0644 -0.0658 -0.0630 7e-04 0e+00
x87          0.4795  0.4781  0.4809 7e-04 0e+00
x88         -2.2914 -2.2928 -2.2900 7e-04 0e+00
x89         -2.2269 -2.2283 -2.2255 7e-04 0e+00
x90         -0.6374 -0.6389 -0.6360 7e-04 0e+00
x91         -3.0174 -3.0188 -3.0160 7e-04 0e+00
x92         -0.0141 -0.0155 -0.0127 7e-04 0e+00
x93          0.7740  0.7726  0.7754 7e-04 0e+00
x94         -1.0298 -1.0312 -1.0284 7e-04 0e+00
x95          0.0459  0.0445  0.0473 7e-04 0e+00
x96          2.5820  2.5806  2.5834 7e-04 0e+00
x97         -0.4834 -0.4849 -0.4820 7e-04 0e+00
x98         -2.0905 -2.0919 -2.0891 7e-04 0e+00
x99          0.2130  0.2116  0.2144 7e-04 0e+00
x100        -1.3311 -1.3325 -1.3297 7e-04 0e+00
x101        -0.1636 -0.1650 -0.1621 7e-04 0e+00
x102        -1.8021 -1.8035 -1.8007 7e-04 0e+00
x103        -0.8984 -0.8999 -0.8970 7e-04 0e+00
x104        -0.2064 -0.2078 -0.2050 7e-04 0e+00
x105        -0.6809 -0.6823 -0.6795 7e-04 0e+00
x106        -1.7670 -1.7684 -1.7656 7e-04 0e+00
x107         0.7991  0.7977  0.8005 7e-04 0e+00
x108         1.7183  1.7169  1.7197 7e-04 0e+00
x109         0.0298  0.0284  0.0312 7e-04 0e+00
x110        -0.1510 -0.1524 -0.1496 7e-04 0e+00
x111        -3.8276 -3.8290 -3.8262 7e-04 0e+00
x112         0.7852  0.7838  0.7866 7e-04 0e+00
x113        -2.6076 -2.6090 -2.6062 7e-04 0e+00
x114         0.0778  0.0763  0.0792 7e-04 0e+00
x115         1.3444  1.3430  1.3458 7e-04 0e+00
```

Thus, these are the regression coefficients that we needed to estimate. Notice that almost all p-values are 0 (except for *x30*). That is due to the size of our data. When the size of the data increases, the p-values tend to go towards 0 and thus are unreliable for significance testing of these said variables.

Part 2: **Next, we shall compute X^TX and X^Ty using a recursive approach and then use the $solve()$ command in order to obtain the least squares estimate:**

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T y}.$$

We already know that the formula above gives us the least squares estimates for the coefficients. Under normal circumstances, when the whole dataset can be loaded into *R*, this application of the formula would be straightforward. Now however, we are limited by the sheer size of our data. Hence, what we should do is to find an updating, recursive algorithm to calculate these same coefficients which we found in *Part 1*.

Let us say that we currently have a small set of 5000 observations and 116 parameters. If we wanted to estimate the coefficients of this dataset, then of course we would use the following formula:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Now let us suppose that on top of the old information, we suddenly get a new set of yet another 5000 observations and naturally, we want to include this new set in our data and re-estimate our coefficients so as to get more accurate results. Then, the formula above would be updated in the following manner:

$$\hat{\beta}_{NEW} = [(X_{OLD})^T X_{OLD} + (X_{NEW})^T X_{NEW}]^{-1}[(X_{OLD})^T y_{OLD} + (X_{NEW})^T y_{NEW}]$$

where:

- X_{OLD} is the design matrix of the initial set of data.
- X_{NEW} is the design matrix of the new, additional set of data.
- y_{OLD} is the response vector of the initial set of data.
- y_{NEW} is the response vector of the new, additional set of data.

And in this manner, we can continue updating our coefficients as many times as we like. Thus, it is a perfect way to calculate recursively least squares coefficients for large datasets. In order to make this a bit easier, notice that the only thing that gets updated each time are the products X^TX and X^Ty. Knowing that inversion is a costly procedure, we shall only do it once, after we have included all the necessary updates to those aforementioned products.

Selecting to repeat this algorithm for chunks of 5000, we will have to repeat this procedure 400 times, since we have a total of 2000000 observations. After we do that, we shall get the results for the Recursive Least Squares (*RLS*) procedure. Let us compare it with the results from *Part 1* in the following output:

```
             BigGLM Method   RLS Method
(Intercept)  -1.709229882 -1.709229882
x1            3.179990874  3.179990874
x2            0.649516689  0.649516689
x3           -0.934266553 -0.934266553
x4            0.277920247  0.277920247
x5            0.845215298  0.845215298
x6           -0.441726214 -0.441726214
x7            0.734079326  0.734079326
```

x8	2.397336338	2.397336338
x9	0.018364976	0.018364976
x10	0.110482633	0.110482633
x11	1.003409866	1.003409866
x12	-0.605186396	-0.605186396
x13	1.478449236	1.478449236
x14	0.175172466	0.175172466
x15	1.147158684	1.147158684
x16	-1.866583946	-1.866583946
x17	1.122851686	1.122851686
x18	-0.179963367	-0.179963367
x19	1.574117470	1.574117470
x20	0.279759515	0.279759515
x21	-0.157358998	-0.157358998
x22	-0.142565266	-0.142565266
x23	-2.709944790	-2.709944790
x24	-1.025865604	-1.025865604
x25	-0.283391147	-0.283391147
x26	-0.407706794	-0.407706794
x27	1.067546316	1.067546316
x28	-1.246514355	-1.246514355
x29	1.864583678	1.864583678
x30	0.002833325	0.002833325
x31	0.455925336	0.455925336
x32	2.072399096	2.072399096
x33	-0.734585316	-0.734585316
x34	0.829945107	0.829945107
x35	0.885918062	0.885918062
x36	-0.480736813	-0.480736813
x37	1.047594191	1.047594191
x38	-0.803391867	-0.803391867
x39	1.225314137	1.225314137
x40	-0.824435847	-0.824435847
x41	-1.333631570	-1.333631570
x42	1.662663936	1.662663936
x43	0.141854162	0.141854162
x44	-0.795843627	-0.795843627
x45	0.954246133	0.954246133
x46	-0.110476629	-0.110476629
x47	0.108737677	0.108737677
x48	0.327706308	0.327706308
x49	0.969222045	0.969222045
x50	-0.509911525	-0.509911525
x51	0.228497674	0.228497674
x52	4.099966015	4.099966015
x53	0.525235686	0.525235686
x54	0.755789701	0.755789701
x55	-2.257947817	-2.257947817
x56	-2.501968250	-2.501968250
x57	-0.588050732	-0.588050732
x58	-1.858684834	-1.858684834
x59	0.060526440	0.060526440
x60	0.264645699	0.264645699

x61	4.580213742	4.580213742
x62	0.083248840	0.083248840
x63	-0.235219720	-0.235219720
x64	-1.370224006	-1.370224006
x65	1.350285983	1.350285983
x66	0.305723488	0.305723488
x67	2.204014366	2.204014366
x68	-0.478250923	-0.478250923
x69	-0.391265187	-0.391265187
x70	0.998592874	0.998592874
x71	-0.437406845	-0.437406845
x72	0.654378324	0.654378324
x73	0.570759655	0.570759655
x74	-0.879170336	-0.879170336
x75	0.943915204	0.943915204
x76	-5.514117922	-5.514117922
x77	0.196658447	0.196658447
x78	-2.275334936	-2.275334936
x79	-0.499754406	-0.499754406
x80	-0.808721267	-0.808721267
x81	-1.388303714	-1.388303714
x82	0.232511623	0.232511623
x83	-0.457522565	-0.457522565
x84	0.452709951	0.452709951
x85	-1.643030373	-1.643030373
x86	-0.064390947	-0.064390947
x87	0.479498776	0.479498776
x88	-2.291432058	-2.291432058
x89	-2.226875373	-2.226875373
x90	-0.637440692	-0.637440692
x91	-3.017405380	-3.017405380
x92	-0.014084559	-0.014084559
x93	0.774017150	0.774017150
x94	-1.029774716	-1.029774716
x95	0.045902985	0.045902985
x96	2.582028617	2.582028617
x97	-0.483437770	-0.483437770
x98	-2.090476554	-2.090476554
x99	0.212999061	0.212999061
x100	-1.331117313	-1.331117313
x101	-0.163553388	-0.163553388
x102	-1.802082854	-1.802082854
x103	-0.898441643	-0.898441643
x104	-0.206380997	-0.206380997
x105	-0.680933609	-0.680933609
x106	-1.767021323	-1.767021323
x107	0.799089298	0.799089298
x108	1.718331446	1.718331446
x109	0.029824298	0.029824298
x110	-0.150969271	-0.150969271
x111	-3.827621604	-3.827621604
x112	0.785201144	0.785201144
x113	-2.607580773	-2.607580773

```
x114           0.077761542  0.077761542
x115           1.344404935  1.344404935
```

As we can plainly see, we get identical results, just like we suspected we would. Let us also see which method was the fastest of the two, by comparing the run times of each procedure:

Method Used	Time Elapsed
BigGLM Algorithm	64.5977 seconds
Recursive Least Squares (*RLS*)	29.0449 seconds

Therefore, we conclude that the second method (*RLS*) was as accurate as the first one and twice as fast, making it a better choice among the two.

<u>Part 3:</u> **We shall use a sub-sampling approach in order to estimate the regression coefficients.**

Yet another method that we can use in order to estimate our model's coefficients, is meta-analysis. As the question itself describes, we split our data randomly into smaller, manageable parts and then for each part we calculate both an estimate of the coefficients and the standard error for each coefficient. After we aggregate many such estimates, we use the following weighted formu-la, in order to get the regression coefficients:

$$\tilde{\beta} = \frac{\sum_{i=1}^{m} w_i \cdot \hat{\beta}_i}{\sum_{i=1}^{m} w_i}$$

where:

- $\hat{\beta}_i$ are the estimated coefficients at each iteration i.
- m is the total number of iterations.
- w_i are the weights at each iteration i, which are derived as the inverse of the estimated variance for each coefficient, meaning that:

$$w_i = \frac{1}{(SE_i)^2}$$

In our algorithm, this procedure will be an iterative one, as we do not want to create a huge matrix neither for the coefficients $\left(\hat{\beta}_i\right)$, nor for the weights (w_i). Therefore, we shall take this algorithm step by step and repeat it m times. We shall take samples of 5000 random observations each and repeat the procedure for a total of 100 times. After our algorithm runs and gives us the coefficients, what we naturally want to see is how well it compares to the other two procedures. This is synopsized in the following output:

```
             BigGLM Method   RLS Method Meta-Analysis
(Intercept)  -1.709229882 -1.709229882 -1.7100357564
x1            3.179990874  3.179990874  3.1820247646
x2            0.649516689  0.649516689  0.6487034095
```

x3	-0.934266553	-0.934266553	-0.9364479276
x4	0.277920247	0.277920247	0.2769108058
x5	0.845215298	0.845215298	0.8456535482
x6	-0.441726214	-0.441726214	-0.4420916497
x7	0.734079326	0.734079326	0.7361695754
x8	2.397336338	2.397336338	2.3959025261
x9	0.018364976	0.018364976	0.0164333893
x10	0.110482633	0.110482633	0.1102020831
x11	1.003409866	1.003409866	1.0022949681
x12	-0.605186396	-0.605186396	-0.6050775860
x13	1.478449236	1.478449236	1.4789194193
x14	0.175172466	0.175172466	0.1733105699
x15	1.147158684	1.147158684	1.1451015858
x16	-1.866583946	-1.866583946	-1.8664302300
x17	1.122851686	1.122851686	1.1222464508
x18	-0.179963367	-0.179963367	-0.1786808288
x19	1.574117470	1.574117470	1.5743587876
x20	0.279759515	0.279759515	0.2818395842
x21	-0.157358998	-0.157358998	-0.1546516427
x22	-0.142565266	-0.142565266	-0.1433523784
x23	-2.709944790	-2.709944790	-2.7082970094
x24	-1.025865604	-1.025865604	-1.0260063077
x25	-0.283391147	-0.283391147	-0.2858870770
x26	-0.407706794	-0.407706794	-0.4086503159
x27	1.067546316	1.067546316	1.0690596679
x28	-1.246514355	-1.246514355	-1.2465898667
x29	1.864583678	1.864583678	1.8670805747
x30	0.002833325	0.002833325	0.0009774587
x31	0.455925336	0.455925336	0.4550279084
x32	2.072399096	2.072399096	2.0731178731
x33	-0.734585316	-0.734585316	-0.7346877642
x34	0.829945107	0.829945107	0.8291446116
x35	0.885918062	0.885918062	0.8858349026
x36	-0.480736813	-0.480736813	-0.4826902013
x37	1.047594191	1.047594191	1.0464904537
x38	-0.803391867	-0.803391867	-0.8033981445
x39	1.225314137	1.225314137	1.2231908819
x40	-0.824435847	-0.824435847	-0.8239576822
x41	-1.333631570	-1.333631570	-1.3339009351
x42	1.662663936	1.662663936	1.6600298343
x43	0.141854162	0.141854162	0.1419023111
x44	-0.795843627	-0.795843627	-0.7984637711
x45	0.954246133	0.954246133	0.9546132016
x46	-0.110476629	-0.110476629	-0.1112021141
x47	0.108737677	0.108737677	0.1069000392
x48	0.327706308	0.327706308	0.3269002013
x49	0.969222045	0.969222045	0.9687788852
x50	-0.509911525	-0.509911525	-0.5101884442
x51	0.228497674	0.228497674	0.2281934968
x52	4.099966015	4.099966015	4.1018919244
x53	0.525235686	0.525235686	0.5227930742
x54	0.755789701	0.755789701	0.7598825370
x55	-2.257947817	-2.257947817	-2.2584989204

x56	-2.501968250	-2.501968250	-2.5014724656
x57	-0.588050732	-0.588050732	-0.5901603933
x58	-1.858684834	-1.858684834	-1.8588071606
x59	0.060526440	0.060526440	0.0607977881
x60	0.264645699	0.264645699	0.2644270004
x61	4.580213742	4.580213742	4.5808889763
x62	0.083248840	0.083248840	0.0841309609
x63	-0.235219720	-0.235219720	-0.2362143660
x64	-1.370224006	-1.370224006	-1.3707706525
x65	1.350285983	1.350285983	1.3517334940
x66	0.305723488	0.305723488	0.3046657658
x67	2.204014366	2.204014366	2.2029333698
x68	-0.478250923	-0.478250923	-0.4784753285
x69	-0.391265187	-0.391265187	-0.3922382672
x70	0.998592874	0.998592874	1.0003933497
x71	-0.437406845	-0.437406845	-0.4366155523
x72	0.654378324	0.654378324	0.6548226588
x73	0.570759655	0.570759655	0.5699189170
x74	-0.879170336	-0.879170336	-0.8775446062
x75	0.943915204	0.943915204	0.9427060918
x76	-5.514117922	-5.514117922	-5.5138318192
x77	0.196658447	0.196658447	0.1975577207
x78	-2.275334936	-2.275334936	-2.2762284706
x79	-0.499754406	-0.499754406	-0.4999955358
x80	-0.808721267	-0.808721267	-0.8106587692
x81	-1.388303714	-1.388303714	-1.3898950478
x82	0.232511623	0.232511623	0.2313068808
x83	-0.457522565	-0.457522565	-0.4544650851
x84	0.452709951	0.452709951	0.4504924627
x85	-1.643030373	-1.643030373	-1.6438405658
x86	-0.064390947	-0.064390947	-0.0647684082
x87	0.479498776	0.479498776	0.4807963898
x88	-2.291432058	-2.291432058	-2.2892685665
x89	-2.226875373	-2.226875373	-2.2266600328
x90	-0.637440692	-0.637440692	-0.6395672683
x91	-3.017405380	-3.017405380	-3.0170462146
x92	-0.014084559	-0.014084559	-0.0123488300
x93	0.774017150	0.774017150	0.7750116122
x94	-1.029774716	-1.029774716	-1.0289246277
x95	0.045902985	0.045902985	0.0465540957
x96	2.582028617	2.582028617	2.5823572354
x97	-0.483437770	-0.483437770	-0.4846540971
x98	-2.090476554	-2.090476554	-2.0916091731
x99	0.212999061	0.212999061	0.2122489352
x100	-1.331117313	-1.331117313	-1.3313580188
x101	-0.163553388	-0.163553388	-0.1646852954
x102	-1.802082854	-1.802082854	-1.8016063393
x103	-0.898441643	-0.898441643	-0.8988522400
x104	-0.206380997	-0.206380997	-0.2083927424
x105	-0.680933609	-0.680933609	-0.6823557252
x106	-1.767021323	-1.767021323	-1.7693543574
x107	0.799089298	0.799089298	0.8005521656
x108	1.718331446	1.718331446	1.7197969175

```
x109          0.029824298  0.029824298  0.0268966699
x110         -0.150969271 -0.150969271 -0.1508189040
x111         -3.827621604 -3.827621604 -3.8298400907
x112          0.785201144  0.785201144  0.7852811770
x113         -2.607580773 -2.607580773 -2.6070827361
x114          0.077761542  0.077761542  0.0788319862
x115          1.344404935  1.344404935  1.3449989204
```

As we can surmise, this method is less accurate than the previous two, however it still gives us pretty good estimates for the regression coefficients. Let us also see how fast it was, when compared to the other two methods:

Method Used	Time Elapsed
BigGLM Algorithm	64.5977 seconds
Recursive Least Squares (*RLS*)	29.0449 seconds
Meta - Analysis	51.3436 seconds

So, timewise, meta-analysis is somewhere in the middle of the previous two methods. Of course, it should be mentioned that both the accuracy of our results and the time it takes to run this algorithm, both depend on two things: the size of our sample and the number of iterations for which we run this procedure. Accuracy can be increased, but the program will take more time to do the calculations, however if we are pressed for time, we can do some alterations at the cost of accuracy.

In conclusion, it must be mentioned that out of all the three procedures, the Recursive Least Squares (*RLS*) was the fastest one, which also had very accurate results, making it the best method to use.