Spring 2019

Randomization Analysis Driven Software

Steph-Yves Louis

RANDOMIZATION ANALYSIS DRIVEN SOFTWARE

by

Steph-Yves Louis

Bachelor of Science
Belmont Abbey College, 2015

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science in Public Health in

Biostatistics

The Norman J. Arnold School of Public Health

University of South Carolina

2019

Accepted by:

James Hardin, Director of Thesis

James Hussey, Reader

Robert Moran, Reader

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

ACKNOWLEDGEMENTS

I would like to thank my God Yahweh and Savior Jesus-Christ for their merciful grace during my studies and throughout this project. It is my hope that they find joy and take glory in the accomplished work that I hereby dedicate to them. Indeed, neither the work nor the completion of this project would have been possible without the two of them. Again, thank you!

I would like to thank Dr. James Hardin who stood as the best mentor I could have obtained for my thesis. I thank him for his guidance, support, and recommendations. Particularly, I am mostly grateful for his continuous feedbacks and motivation to combine my acquired knowledge in Biostatistics, sense of creativity, and love of coding to obtain this finished piece of art. Additionally, I would like to also thank Dr. Hussey and Dr. Moran for their advice, support, and suggestions so as to further improve the quality of the resulting work. Thank you all for having agreed to be on my committee.

I would like to thank my family and former co-workers at S.C.T.C.S who have encouraged me throughout this hard but rewarding journey. My beautiful wife Christina, my mother Hermione, and my former mentors Dr. Frazier and Mrs. Maria especially played a key role for they never stopped pushing me to finish the race! Finally, I would like to thank my father Alain, Christian, Harnel, Doubea, Cameron, Jean-Luc, Dr. Ortaglia, and Mrs. Jamie for either their prayers, feedbacks, or support. Even if you don’t find your name mentioned, do not be offended, for if you made any contribution, thank you!

ABSTRACT

The application of a method of randomization for a clinical trial frequently summarizes to using Simple Randomization. Even though the latter method provides favorable characteristics, if the collected sample is not large enough, it still presents the highest chance of imbalance both marginally in the treatment groups and locally in terms of the covariates. Methods of Permuted Block Randomization, Urn Randomization, Stratified Permuted Block Randomization, and Minimization represent popular alternative methods that one should consider depending on the goal of the study. A comparison of the previously mentioned methods is carried to evaluate their performance with samples that are not considered large. Additional goals of our study are to also assess the performance of our newly implemented methods of Minimizations based on the performances of the established methods.

The found results show that the existing Minimization had the lowest imbalance amongst the previously established methods. Our newly implemented methods of Kolmogorov-Smirnov Minimization and Minimization with increasing Factor showed to be superior to the already established methods when the objective is to randomize on subjects' variables. The found results also served the purpose of additional reference to build a free software that any user may employ to appropriately randomize subjects.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

I-MIN ..... Minimization with Increasing Factor

KS-MIN ..... Kolmogorov-Smirnov Minimization

MIN ..... Minimization

PB ..... Permuted Block Randomization

R-MIN ..... Mixed Rank-Minimization

RCT ..... Randomized Clinical Trial

SRA ..... Simple Randomization

SPB ..... Stratified Permuted Block Randomization

UD ..... Urn Design

CHAPTER 1

INTRODUCTION

In many fields, especially in Public Health, the utilization of a randomization method for the recruitment of subjects in a study is extremely important. Randomization helps decreasing the chances of bias while also increasing the likelihood that each subject will receive an equal chance of enrolling into any treatment group (Suresh, 2011). With the use of an appropriate randomization method, one should expect the following effects: better comparability among the treatment groups marginally, better comparability of the treatment groups with respect to known and unknown covariates, and improved validity for the statistical tests (Suresh, 2011).

There exist numerous procedures that one may apply to randomize subjects. Depending on the aimed number of subjects to be collected and the importance of balance, the randomization methods will differ with the consideration of their algorithms, their benefits, and their disadvantages. Amongst the known randomization methods and their implemented algorithms figure the simple or complete randomization (SRA), the permuted block randomization (PB), the urn design (UD), the stratified permuted block randomization (SPB), and the minimization (MIN) (Suresh, 2011).

The simplest and most popular randomization method is the simple randomization (SRA). In general, its application simply resolves to using either a coin, or a dice, or any simple computer generating sequence to randomly allocate subjects to treatment groups. (Kim, 2014). Though easily implemented, the application of simple randomization

does not consider the information from prior assignments or covariates. Permuted block Randomization (PB) describes another randomization method which consists of creating boxes filled with randomly generated assignments. As in the simple randomization, the permuted block randomization does not make any adjustments for the subjects' variables (Kim, 2014).

The urn design (UD) represents a restricted method of the simple randomization. Due to its flexible properties, one may classify the former as an appropriate compromise between the simple randomization and the permuted block randomization (JM, JP, & LJ, 1988). The stratified permuted block randomization (SPB) defines a well-known procedure of randomization that is considered when one desires the randomization to also affect the subjects' variables. Though relatively simple, the application of the stratified permuted block randomization leads to increasing inefficiency and complexity as the number of variables increases. Lastly, minimization (MIN) is a widely studied and implemented randomization method which considers all prior information to assign a treatment to a new subject. Various recommendations can be found in the literature on the use of the five previously mentioned randomization methods.

Neither the simple randomization nor the Urn Design randomization is recommended when the sample is small (Dettori, 2010) (JM, JP, & LJ, 1988). With the implementation of block randomization, one should adjust for the possibilities of selection-bias and possible dissimilarity of the treatment groups with respect to the covariates (Suresh, 2011). Given a study in which the investigator wants to also randomize on the subjects' variables, then one ought to consider applying stratified permuted block randomization or minimization. However, one should limit the number of

variables and the levels contained within each variable before using the stratified permuted block randomization. On the other hand, Pocock and Simon's minimization method is recommended for use in studies with especially small samples (Tu, Shalay, & Pater, 2000).

In Table 1.1, we classify the different randomization schemes based on their attributes for being predictable, adaptive (use prior information), and adoptive of variables information (use variables). Because the performances of the five randomization methods have not been exhaustively investigated on their handling of random data showing a strong defined pattern, we carry a simulation study and here forth report the results obtained.

Table 1.1 Classification of the existing randomization

	SRA	PB	UD	SPB	MIN
Predictable	X	✓	X	X	X
Adaptive	X	X	✓	X	✓
Uses Variables information	X	X	X	✓	✓

We have also implemented two new methods based on the existing minimization procedure. The first implemented method is called minimization with increasing factor (I-MIN). The latter is obtained by slightly modifying the algorithm of the existing minimization to make its constant random factor of 0.75 increase as the number of enrolled subjects increases. The other implemented randomization method is called Kolmogorov-Smirnov minimization (KS-MIN) which is obtained by incorporating a 2-sample Kolmogorov-Smirnov test to also randomize covariates of continuous data.

The final goal of this study is to combine the recommendations found in the literature and the results obtained from our project to create a user-friendly randomization software. Named R.A.D.S. for Randomization Analysis Driven Software, this software would request very little input information to provide the user a recommended method that one can automatically use in that software. Made for both Windows and Mac, the beta version of the software is available for download and usage.

CHAPTER 2

LITERATURE REVIEW

SRA, PB, UD, SPB, and MIN describe well-known randomization methods that one may apply to randomize subjects. Depending on the objective of the investigator, the goal of minimal imbalance may be negatively affected with the application of the inappropriate procedure. For the sake of simplicity, the imbalance of the randomization methods is only evaluated for 2 treatment groups. We hereby define imbalance as the absolute value of the difference in number of subjects between the two groups. For the number of subjects N and the groups A and B, the marginal imbalance I can also be formulated as:

$$I = |N_A - N_B| \tag{1}$$

Below we fully describe the five randomization methods on which we based our project.

2.1 Simple Randomization

Simple randomization stands as the most used randomization method because of its attractive properties and simple implementation in practice. SRA provides 3 main benefits: fully random outcome, very simple algorithm, and freedom to use any statistical analysis on the resulting randomized data. Notably, SRA reduces to recording the results of a fair coin toss to afterward assign a treatment according to that toss result (Hedden, Woolson, & Malcom, 2006). Considering 10 sequential tosses of a fair coin to randomly allocate 10 subjects in either group A or group B, resulting heads (H) could mean

allocations to group A while resulting tails (T) allocations to group B. A possible sequence of coin toss results can be seen in Table 2.1.

As stated in the literature, the consideration of SRA also comes with the important detriment of considerable imbalance when the sample collected is relatively small <100 (Kang, Ragan, & Park, 2008). Another disadvantage of SRA derives from the fact that it cannot use any of the subjects' information to do its randomization. For instance, in a sample of 50 subjects, a plausible effect of applying this randomization scheme could be that only 9 subjects are assigned in group A while 41 subjects to group B. In this scenario, the calculation for the marginal imbalance $|41 - 9|$, results to 32.

Table 2.1. Possible treatment assignment from simple randomization

	Subj. 1	Subj. 2	Subj. 3	Subj. 4	Subj. 5	Subj. 6	Subj. 7	Subj. 8	Subj. 9	Subj. 10
Fair-Coin Toss Result	H	H	H	H	T	T	T	H	T	H
Corresponding Allocation	A	A	A	A	B	B	B	A	B	A

2.2 Permuted Block Randomization

Permuted block randomization defines a popular procedure that generally prevents the issue of considerable imbalance which results from SRA. Additionally, the implementation of its simple algorithm in practice further contributes to making this method appear more favorable than SRA.

To implement the method of PB in a study, one first selects an appropriate block size, a multiple of the treatment number, that will dictate how many treatments that a

single block may contain. In the literature, it is advised to opt for smaller block sizes because the risks of imbalance are much higher with the use larger blocks (JM, JP, & LJ, 1988). For instance, if one considers a study with only 2 possible treatments, smaller block sizes could include 4, 6, or 8 while large block sizes could describe 16 or 24. Afterwards, the process resolves to randomly filling each block with an equal number of each treatment type. The former step may be repeated as often as needed, as demonstrated in Figure 2.1, to enroll the right number of subjects in a study.

Block #1	
Allocation for subjects 1-6	
Subj.	***Assignment***
#1	*A*
#2	*A*
#3	*A*
#4	*B*
#5	*B*
#6	*B*

Block #2	
Allocation for subjects 7-12	
Subj.	***Assignment***
#7	A
#8	A
#9	B
#10	A
#11	B
#12	B

Block #3	
Allocation for subjects 13-18	
Subj.	***Assignment***
#13	A
#14	B
#15	B
#16	A
#17	B
#18	A

Block #4	
Allocation for subjects 19-24	
Subj.	***Assignment***
#19	A
#20	B
#21	A
#22	B
#23	A
#24	B

Figure 2.1. Illustration of 4 blocks with size 6

The PB method requires that the blocks are made prior to the subjects' enrollment in the study. Hence, as subjects enroll, each will receive the assignment matching one's order of enrollment. Considering a scenario of a two-armed randomized clinical trial (RCT) in Figure 2.1, with groups A and B and the block size fixed at 6, the random allocation to group B would be given to the 9^{th} enrolled subject, A for the 10^{th}, B for the 11^{th}, etc.

The first downside of PB describes the fact that it does not utilize any of the subjects' information to provide treatment allocations. In addition to ignoring subjects' data, PB also presents the drawback of high predictability for the last assignment when a

constant block size is known. As a result, the latter disadvantage introduces the risks of selection bias into the study (Efird, 2011).

Again referring to the study in Figure 2.1, if treatments A, B, A, B, and A, from block #4, had already been assigned, then one can determine that the next assignment of B would be allocated to the following subject (Tu, Shalay, & Pater, 2000). Possible solutions to the problem of high predictability involve either the usage of random block sizes as seen in Figure 2.2 or the usage of larger block sizes. While the application of random or larger block sizes makes it harder to accurately determine correctly determine the next assignment, larger blocks present the disadvantage of increasing the risks of marginal imbalance. Nevertheless, the risks of imbalance in terms of the covariates remain regardless of which type of PB is applied.

Block #1	
Allocation for subjects 1-6	
Subj.	***Assignment***
#1	*A*
#2	*A*
#3	*A*
#4	*B*
#5	*B*
#6	*B*

Block #2	
Allocation for subjects 7-12	
Subj.	***Assignment***
#7	A
#8	A
#9	B
#10	B

Block #3	
Allocation for subjects 11-16	
Subj.	***Assignment***
#11	A
#12	B
#13	B
#14	A
#15	B
#16	A

Block #4	
Allocation for subjects 17-24	
Subj.	***Assignment***
#17	A
#18	B
#19	A
#20	B
#21	A
#22	B
#23	A
#24	B

Block #5	
Allocation for subjects 25-28	
Subj.	***Assignment***
#25	A
#26	A
#27	B
#28	B

Figure 2.2. Illustration of 5 blocks with random sizes

For instance, we assume that males (M) and females (F) get enrolled in a two-armed RCT, with set block size of 6. At the end of the 11^{th} subject's enrollment, a probable sequence of the subjects' gender along with the treatment is given below in Table 2.2. Even though the marginal imbalance of the treatments is kept to a minimum, $I = |6 - 5| = 1$, one may observe the resulting non-comparability of the treatments with

respect to the covariate of gender. Notably, the only 2 females in the sample are enrolled in group B and one also notices that there are twice as much males in group A than there are treatment B.

Table 2.2. Possible treatment assignment from permuted block randomization

	Subj. 1	Subj. 2	Subj. 3	Subj. 4	Subj. 5	Subj. 6	Subj. 7	Subj. 8	Subj. 9	Subj. 10	Subj. 11
Gender	M	M	M	F	F	M	M	M	M	M	M
Treatment Assignment	A	A	A	B	B	B	A	A	B	A	B

2.3 Urn Design

Considered as the most studied method of randomization from the Bias-Coin Randomization class, UD involves a procedure that provides more flexibility on ways to influence the outcome of balance for the two branches of the R.C.T. study (Wei & Lachin, 1988). The literature frequently refers to Friedman's urn model to describe this randomization scheme (Wei & Lachin, 1988). Simply put, the algorithm of UD consists of selecting two numbers X and Y and then use them in the following manner.

An equal number of balls of each type is initially placed into an urn from which selections will occur. For our given scenario, we pick two types of balls (blue and purple: one corresponding to each treatment group). Hence, the urn will contain X blue balls and X purple balls. Specifically, we assign a selection of a blue ball (B) to the allocation to group A while the selection of a purple ball (P) to the allocation to group B. With the random selection of a ball with replacement, the corresponding subject is allocated to the

appropriate treatment, and then Y balls of the opposite type are added into the urn. This method of randomization is referred to as $UD(X, Y)$.

The choice X dictates how similarly the urn design should behave compared to complete randomization during the early stages of enrollment. On the other hand, the choice of Y dictates how the algorithm preserves balance (Wei & Lachin, 1988). Unless Y equals to 0, the probability of enrollment into any group will vary and it will be higher for the treatment group with the smaller number. In Figure 2.3, we illustrate an example of implementing the urn design for 6 subjects using $UD(2,2)$.

Before the enrollment of the first subject, there are 2 blue and 2 purple balls inside the urn. Once the first subject gets enrollment into the RCT study, a random Purple ball is picked and placed back in the urn, an allocation for group B is given to the subject, and then two blue balls are added into urn. This process increases the chances of the next subject to get allocated to treatment B from 50% to 67%. The same algorithm applies to randomly allocate a treatment group to each subsequent subject. It is recommended to use the urn design when the aim of marginal balance is wanted, yet not warrantied (JM, JP, & LJ, 1988).

Urn Adaptive Design
UD(2,2)

Entry of Subj.	Urn	Random ball	Assignment
Start	BBPP	None	None
Subj. #1	BBPP	P	group B
Subj. #2	BBBBPP	B	group A
Subj. #3	BBBBPPPP	P	group B
Subj. #4	BBBBBBPPPP	P	group B
Subj. #5	BBBBBBBBPPPP	P	group B
Subj. #6	BBBBBBBBBBPPPP	B	group A

Figure 2.3 Illustration of an urn design with x and y fixed at 2 and 2 for 6 subjects

2.4 Stratified Permuted Block Randomization

The method of stratified block randomization is analogous to the block randomization in the aspect that it uses assignment blocks of a pre-determined size (Suresh, 2011). The main difference between PB and SPB is that the latter adjusts for use of covariates. Besides the selection of block sizes, one must also identify the covariates of interest that one desires to incorporate in the study. The next step requires the combination of the covariates of interest, which would create all the strata by which the randomization will occur.

Considering the same RCT study described in the methods above, one could use the stratified randomization to make the randomization use the information for the designated covariates. To explain this method, we decide to use 2 binary covariates: gender (Male and Female) and diabetes (yes: if someone suffers from it and no: otherwise). Hence, the combination of the binary variables of sex and diabetes result in the four following strata: Male & Diabetes, Male & No-Diabetes, Female & Diabetes, and Female & No-Diabetes; each stratum contains the same block size set at 6. For each one of the strata, we then create many blocks containing an equal number of allocations. Table 2.3. below provides an illustration of the blocks within each stratum.

Once the strata are generated, subjects subsequently enroll into the study, they are first matched into a stratum and then, they get assigned the treatment group that corresponds to the order of their enrollment within that stratum. Notably, the method of stratified randomization reduces to applying PB inside each stratum. The disadvantage of this method is that it cannot accommodate a large number of strata; in fact, one should

limit the number of strata to a maximum of N/B where N is total sample size and B the designated block size (Tu, Shalay, & Pater, 2000).

Table 2.3. Possible treatment assignment for stratified block randomization

Strata	Block 1	Block 2	Block 3	Block 4	Block 5	…
Male & Diabetes	BABABA	AAABBB	BABABA	BAABBA	…	…
Male & No-Diabetes	BBABAA	ABABBA	ABBBAA	ABBAAB	…	…
Female & Diabetes	BABBAA	BABBAA	BABAAB	BAABAB	…	…
Female & No-Diabetes	BBBAAA	BBBAAA	BABABA	ABBBAA	…	…

2.5 Minimization

First developed by Taves in 1974, Pocock and Simon later generalized this randomization method by modifying its algorithm (Tu, Shalay, & Pater, 2000). Only Pocock's and Simon's minimization are used in our project. The procedure can be explained in the following manner. Assume, for a given study, that we decide to collect the subjects' information on Race (3 levels), Gender (2 levels), and disease (2 levels). We may further assume that the goal is to randomly allocate a treatment to a new subject who is a Male, Black, with no-disease. Figure 2.4. displays the summarized information of the first 10 subjects already allocated in the hypothetical study.

The process involves the consideration of separately assigning the new subject group A and B, calculating an imbalance score by each variable, calculating a total imbalance score, and then finally assign the group that has the smallest imbalance, with

respect to a randomly generated probability. In Figure 2.5., we provide an illustration of the calculation in MIN to determine the next treatment.

		Group A	Group B
Race	Female	3	2
	Male	3	2
Gender	Black	1	0
	Other	2	3
	White	2	2
Disease	Yes	2	2
	No	2	4
Total		6	4

Figure 2.4 Summary of covariates information for 10 subjects

If assigned to group A

		Group A	Group B	Imbalance
Race	Male	3+1	2	| (3+1) - 2 |
Gender	Black	1+1	0	| (1+1) - 0 |
Disease	No	2+1	4	| (2+1) - 4 |
		Total Imbalance		2+2+1=5

If assigned to group B

		Group A	Group B	Imbalance
Race	Male	3	2+1	| 3 - (2+1) |
Gender	Black	1	0+1	| 1 - (0+1) |
Disease	No	2	4+1	| 2 - (4+1) |
		Total Imbalance		0+0+3=3

Figure 2.5 Calculation in the minimization process

The first step requires the imbalance measure for each covariate assuming that the subject is assigned to a particular group. Once all the local imbalance scores are calculated, they are subsequently summed to obtain a total imbalance which leads to the

comparison of those total imbalances (for both group A and B). If the total imbalances are different, the group that presents the smallest total imbalance is preferred. Lastly, a random value P from a Uniform distribution $(0, 1)$ gets generated for the new subject.

The aforementioned P is then compared to a fixed probability P^* , ranging from 0 to 1, that should be set by the investigator before the study begins. In our scenario, we set P^* at 0.75, the lowest recommended value by the literature (Pocock & Simon, 1975). Henceforth, the new subject gets assigned to either group A or group B based on the following criteria:

a) If $P \leq P^*$, then finally assign the preferred group to the subject

b) Else, do a coin-toss to randomly assign either group A or group B to the subject.

Assuming that the randomly generated P equals 0.63, applying MIN leads to the assignment of group B for the 11^{th} subject. While this method appears favorable, its algorithm presents the downfall that one must categorize all continuous data. An existing alternative to this issue is the method of method of R-MIN which is briefly described below.

2.6 Rank Minimization

The rank-minimization represents a set of simple calculations to carry to avoid the required step of discretization for numerical data. Thanks to its simplicity and allowance of more variability within the data, the algorithm for the rank-minimization is preferred to minimization when dealing with continuous data (Stigsby & Taves, 2010). In our project, we slightly modify the algorithm of the rank-minimization to accommodate both categorical and continuous data. In Figure 2.6., we provide a sample of continuous

variables that we should use to explain the use of Rank Minimization to allocate a treatment to the 7th subject enrolling in a study.

Given a new subject with a weight of 145 lbs. and height of 61.2 in., the rank minimization involves the calculations of an imbalance score based on the rank-sums. Similarly to the method of minimization, the first step of the algorithm requires the consideration of two independent scenarios corresponding to an allocation to group A and another to group B. Figure 2.7. illustrates the calculation of the imbalance for each scenario.

Assignment		Weight (lbs)	Rank of weights	Height (in)	Rank of heights
group A	Subj. #1	145.5	4	66	5
group B	Subj. #2	128.9	1	45	1
group A	Subj. #3	167.2	6	60	3
group A	Subj. #4	159.2	5	70	6
group B	Subj. #5	202.3	7	76	7
group A	Subj. #6	130	2	52	2
	Subj. #7	145	3	61.2	4

Figure 2.6 Illustration of assignments, weights, and height for 7 subjects

In both of the assignment considerations, one must rank all of the numerical values, including the new subject's covariates. The new subject's weight of 145 lbs. and height of 61.2 in. correspond to ranks of 3 and 4. The next step involves summing the ranks for each covariate by group assignment.

For instance, considering the assignment to group A, the sum of ranks of weight for group A is 20 and group B is 8. The sum of ranks of height for group A is also 20 while the one for group B is 8. On the other hand, the sum of the ranks when considering

group B are 17 and 11 for weight, and 16 and 12 for height. The subsequent steps involve the calculation of the average of the sum of the ranks, the calculation of the deviation from the mean for the sums of ranks, the squaring of the latter numbers, and finally the summation of the squared deviations within each group assignment consideration. Since the resulting imbalance scores for group A and B are 144 and 26, group B would be preferred. Even though the rank-minimization method presents the benefits of allowing the investigator to randomize continuous data without discretization, its calculations become more complex as more patients enroll.

	If Assigned to group A				If Assigned to group B			
	Weights		Heights		Weights		Heights	
	group A	group B	group A	group B	group A	group B	group A	group B
	4		5		4		5	
		1		1		1		1
	6		3		6		3	
	5		6		5		6	
		7		7		7		7
	2		2		2		2	
	3		4			3		4
Sum of ranks	20	8	20	8	17	11	16	12
Mean rank-sum	14				14			
Deviation from mean	6	-6	6	-6	3	-3	2	-2
Squared deviation	36	36	36	36	9	9	4	4
Imbalance: sum of sqrd. Deviations	144				26			

Figure 2.7 Calculation in Rank Minimization

CHAPTER 3

DESCRIPTION OF METHODS

3.1 Strong Pattern

The first goal of this project is to investigate the performance on the 5 randomization methods of SRA, PB, SPB, UD, and MIN in the presence of data with a strong pattern. Our term of strong pattern defines an apparent pattern in the subjects' covariates that is observed throughout the enrollment process. An example of a strong pattern is illustrated in Table 3.1. For instance, an example of a strong pattern in the enrollment of 250 subjects for a given study could define the repetitive enrollment of approximately 6 females for every male. Though very hypothetical, an investigator may indeed have to deal with a similar case in the real practice.

Table 3.1 Illustration of strong pattern

	Subj. 1	Subj. 2	Subj. 3	Subj. 4	Subj. 5	Subj. 6	Subj. 7	Subj. 8	Subj. 9	Subj. 10	…
Sequence	F	F	F	F	F	F	M	F	F	F	…

3.2 Variables

To carry our investigation, we use methods that are comparable to the ones found in the literature (Stigsby & Taves, 2010). Our benchmark of good performance is defined

as the perfect balance amongst treatment groups, both marginally and with respect to the covariates. For simplicity, the performances of the randomization methods are only evaluated for 2 treatment groups for the sample sizes of 25, 50, 105, 150, and 215. The choice of our designed sample sizes corresponds accordingly to the types of sample sizes described in the literature as small, moderate, and large (small [25, 50]; moderate [105, 150]; and large [215]). Additionally, the 3 variables used for our research describe a binary variable, a categorical variable of 5-levels, and a continuous variable (JM, JP, & LJ, 1988).

For the data with a strong pattern: the binary variable was generated so that the weights of 0.75 and 0.25 be given randomly to the two levels. The variable with 5 levels had weights of 0.1, 0.15, 0.45, 0.25, and 0.05 also randomly given to the different values. Lastly, we generated the continuous variable from a skewed-normal distribution (skewness of 0.632, mean of 50, and standard deviation of 5.5). Additionally, the latter was also discretized into a categorical variable of 4 levels corresponding to the quartiles of the generated sample.

For the data with a truly random pattern: neither the binary nor the 5-level categorical variables were given weights. We also generated the continuous variable from a normal distribution (mean of 50, and standard deviation of 5.5) which was likewise discretized into 4 levels according to the quartiles of the sample.

3.3 Simulation

We used the general-purpose programming language of Python. All the python modules created for this study are written in Python 3.x and the codes used can be found

below in Appendix A. Our results can be reproduced by using the Random module seed of 10621 and the Numpy module seed of 1921. Once we prepared our program, we carried each simulation 1000 times. For each simulation run, we recorded the marginal imbalance score for the groups and for the 3 covariates. We applied the same procedure on our randomly generated data that were not forced to show patterns.

In the three figures below: Figure 3.1, Figure 3.2, and Figure 3.3, we provide an illustration of 1 sample of 1000 observations for the binary variable, the 5-level categorical variable, and the continuous variable.

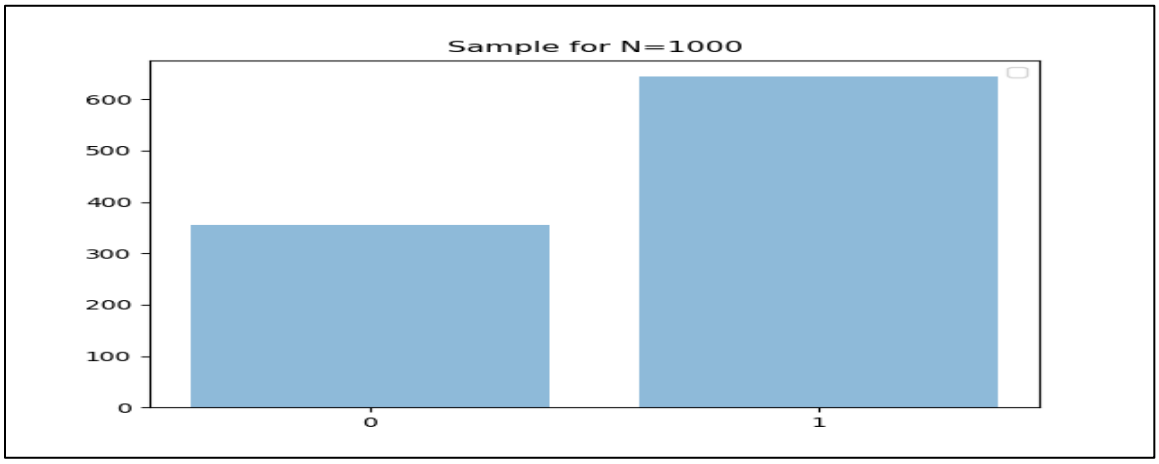

Figure 3.1 Sample distribution of binary covariate for 1000 subjects

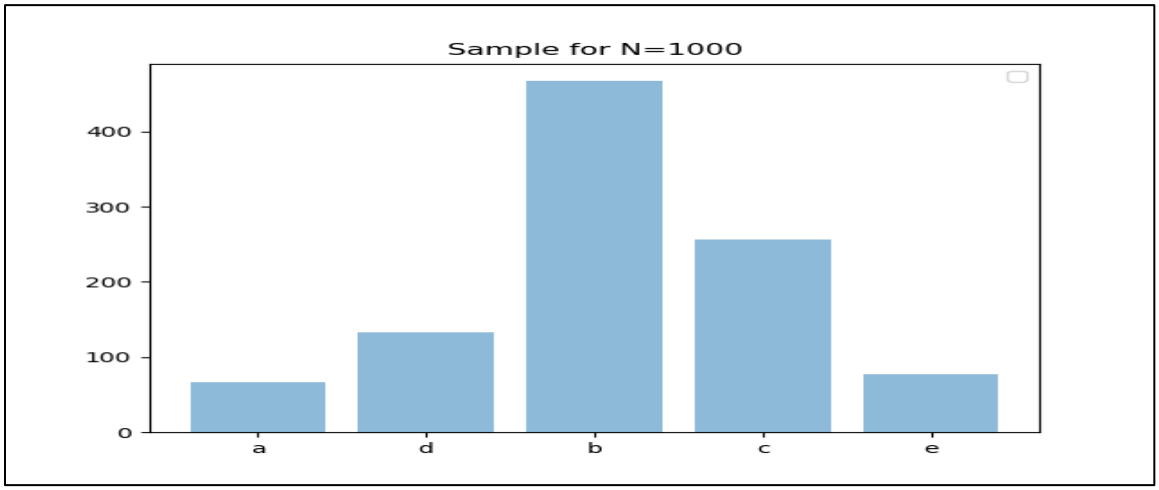

Figure 3.2 Sample distribution of 5-level covariate for 1000 subjects

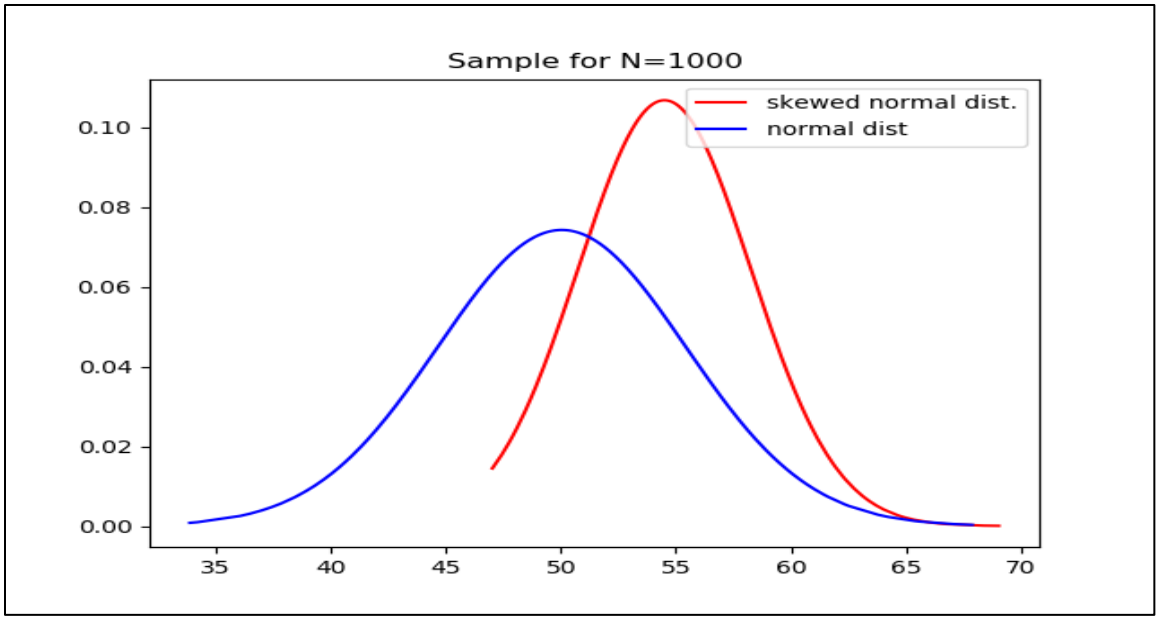

Figure 3.3 sample distribution of continuous covariate for 1000 subjects

3.4 Performance Indicator

Throughout our study, we employ the simple mathematical method of the range to obtain the groups marginal imbalance and the imbalance scores for the covariates (Pocock & Simon, 1975). We may consider a simple study for which the information on gender, disease, and race of 501 enrolled subjects are collected and listed below in Table 3.2. For this example, we do not make any consideration on the particular method of randomization that was employed.

Table 3.2 Summary of enrollment for any given study

	Gender		Disease		Race			Total
	Male	Female	Yes	No	Black	Other	White	
Group A	105	140	49	196	60	90	95	245
Group B	115	141	51	205	60	80	116	256
Total	220	281	100	401	120	170	211	501

We calculate the imbalance for the groups by taking the absolute value of the difference between the numbers in each group: $R_{group} = |N_a - N_b| = |245 - 256| =$

11; N_a & N_b the numbers in each assignment group and R_{group} is the group range. The same concept also applies for each level within a given covariate. The range measure for Males is given by $r_{males} = \left|N_{males,a} - N_{males,b}\right| = |105 - 115| = 10$; r_{males} is the range of males in the sex covariate. The range of non-diseased subjects is evaluated as: $r_{nodisease} = \left|N_{nodisease,a} - N_{nodisease,b}\right| = |196 - 205| = 9$. Once we measure the range of each level within a covariate, we then sum all of its ranges. This overall measure is obtained by applying the summation below:

$$\sum_{i=1}^{k} r_i \tag{2}$$

where r is the range of a level within a covariate and i: 1,2, … , k are the different levels. Thus, the total group imbalance is the range of the groups and the imbalance for a given covariate is the sum of the ranges for each one of its level.

In our given example, the total imbalance for the groups is given as $I_{group} = 11$; I is the imbalance measure. The imbalance for gender is measured as: $I_{gender} = |105 - 115| + |140 - 141| = 10 + 1 = 11$. The imbalance for disease is: $I_{disease} = |49 - 51| + |196 - 205| = 2 + 9 = 11$. Last, we get the race imbalance as: $I_{race} = |60 - 60| + |90 - 80| + |95 - 116| = 0 + 10 + 21 = 31$.

In our study, we further scale these imbalance scores as a ratio by dividing them by the total number of subjects in the study: $D = \frac{I}{N}$; I is the imbalance and N is the total number of subjects. Hence, the marginal imbalance ratio for the groups is given as $D = \frac{|256-245|}{501} = \frac{11}{501}$. The imbalance ratio for gender is:

$D_{gender} = \frac{|105-115|+|140-141|}{501} = \frac{11}{501}$. The imbalance ratio for disease is:

$D_{disease} = \frac{|49-51|+|196-205|}{501} = \frac{11}{501}$. The imbalance ratio for race is:

$D_{gender} = \frac{|60-60|+|90-80|+|95-116|}{501} = \frac{31}{501}$.

To measure the imbalance of the numerical covariate, we performed a 2-Sample Kolmogorov Smirnov test on the sample obtained for group 0 and for group 1, and then

used $1 - p.value$ as indicator of the dissimilarity. Assuming that a given covariate of weight listed values of 101.41, 95.7, 110 for group 0 and values of 116.12, 103.5, 73.4, 85.8, 106 for group 1; then doing a Kolmogorov Smirnov test would yield a p-value of 0.8254. Hence, an imbalance ratio between the groups 0 and 1, in terms of weight is: $D_{weight} = 1 - 0.8254 = 0.1746$.

CHAPTER 4

NEW RANDOMIZATION PROCEDURES

4.1 Minimization with Increasing Factor

In the literature, it is recommended to set the value of P^*, as seen in the existing minimization method, between 0.75 and 1.00; implying that the algorithm behaves more deterministically as that value increases. Our custom version of Pocock and Simon's minimization differs from the original in the fact that we allow our value of P^* to keep increasing as the number of patients who enroll in the study approaches a fixed number N^* that one sets prior to the enrollment of the first patient. For this altered version of minimization, we initially set P^* at 0.75 and then at each subject's enrollment P^* increases by either 0 or by $\frac{3}{4\times N^*}$. In other words, after each new patient's allocation to either group A or group B, P^* may only randomly increase by $\frac{0.25}{N^*}$. Notably, this method is nearly identical to earlier minimization's scheme.

4.2 Important Details on Custom Rank-Minimization

For our custom Rank minimization (R-MIN), we combine the regular minimization and the rank minimization. At the enrollment of a new subject, we calculate the imbalance scores for the binary covariate and the 5-level covariate through the minimization algorithm and the imbalance scores for the continuous covariate via the

rank minimization algorithm. To aggregate a single imbalance score for our mixed minimization, we summed the imbalance scores of the binary covariate (I_1), the score of the 5-level categorical covariate (I_2), and the log for the sum of the score of the continuous covariate and $\frac{1}{10}$ $(I_3 + 0.1)$. Here, the addition of $0.1\ or \frac{1}{10}$ to the imbalance score of I_3 avoids taking the log of 0. Hence, the total imbalance for a given group allocation equals to

$$I_1 + I_2 + \log(I_3 + 0.1) \quad (3)$$

To allocate a random group, we apply the same criteria that were explained in the minimization algorithm.

4.3 Kolmogorov-Smirnov Minimization

The Kolmogorov-Smirnov Minimization (KS-MIN) test is almost identical to R-MIN. The only difference between the two methods is that in the latter, we use a 2-sample Kolmogorov-Smirnov test on the continuous variable (ENGMANN & COUSINEAU). The 2-sample KS test is a non-parametric test for continuous data that can may be used for two samples. Notably, the KS test assesses whether two independent samples come from the same distribution. Referring to the same 7 subjects in Figure 2.7, we explain our methodology on the application of the KS test for the covariate of height in the Figure 4.1. below.

KS-MIN requires that the KS-test be applied on both sides, the imbalance scores are then measured by subtracting the obtained p-values from 1. To obtain a single imbalance score, we combine the imbalance scores of the categorical covariates and the

exponentiated imbalance score for the continuous variable. Contrasting the scenario of our custom R-MIN above, a total imbalance for a given group allocation would equal to

$$I_1 + I_2 + \exp(I_3) \tag{4}$$

Beyond this point, the same steps that are stated above in our implemented R-MIN apply.

Assignment	Height (in)
group A	66
group B	45
group A	60
group A	70
group B	76
group A	52
	61.2

If Assigned to group A	
Heights	
group A	group B
66	
	45
60	
70	
	76
52	
61.2	

If Assigned to group B	
Heights	
group A	group B
66	
	45
60	
70	
	76
52	
	61.2

	pvalue=0.70518306	pvalue=0.96103385
Imbalance: 1 - pvalue	0.29481694	0.03896615

Figure 4.1 Illustration of application for the KS-test

CHAPTER 5

RESULTS

Below we provide the results from our simulation study, in which we evaluate the performances of the randomization methods of simple randomization, block randomization, stratified block randomization, bias-coin randomization, and the original minimization. Furthermore, we also compare our new minimization with increasing random factor and mixed Kolmogorov-Smirnov minimization to the performances of minimization and mixed rank-minimization. We present the results in two sections: a) for data forced to follow a pattern and b) data following any random pattern.
Furthermore, we also use the following notations to identify the randomization methods studied as 1 of 3 types:

- Method that does not take covariates into account (*)
- Method that only takes discretized covariates into account (+)
- Method that takes both categorical and continuous covariates (*x*)

In the following tables, we list the results obtained for our simulation study considering small samples (25 and 50), moderately small sample sizes (105 and 150), and a large sample size of 215.

5.1 Performance on Data with Strong Pattern

Table 5.1 lists the results of the different methods considered in our simulation for the imbalance ratio, between group 0 and group 1, without taking any covariates into consideration. Table 5.2 and 5.3 present the imbalance ratio between group 0 and group 1

for the two categorical covariates (binary and 5-level), while Tables 5.4 and 5.5 list the imbalance for the continuous covariate. In general, when only considering marginal imbalance, the application of any Permuted Block Randomization method returned the smallest results amongst all the methods across all sample sizes. The Minimization procedures come in second with very similar results. The subsequent methods in order of performance are Urn Randomization, Stratified Permuted Block Randomization, and then Simple Randomization. As the sample sizes increase, one notices that the marginal imbalance ratio decreases for all considered methods. In fact, the choice of a randomization method beyond a sample size of 150 becomes less important since the difference in imbalance ratio for different methods becomes negligible (see Table 5.1).

In Table 5.2 and Table 5.3, the imbalance for the categorical covariates are listed. The Minimization Methods considerably outperformed the other randomization procedures. Similarly to the marginal imbalance ratios, the imbalance ratios for the categorical covariates also decrease as the sample sizes increase.

In Table 5.4 and Table 5.5, the imbalance ratio for the continuous covariate are listed. Particularly, in Table 5.4, the values were discretized into 4 quartiles (discretized)*, while in Table 5.5 the values were left as numerical. Except for the Kolmogorov Smirnov Randomization, none of the other randomization algorithms returned improved scores. The Kolmogorov Smirnov outperformed all the other methods and showed decreasing scores as size increased.

Table 5.1. Comparison of results for marginal imbalance ratio

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215

*	Block Randomization of size: 4	4.00%	1.28%	0.95%	0.46%	0.47%
*	Block Randomization of size: 6	4.00%	1.52%	1.12%	0.00%	0.47%
*	Block Randomization of size: 8	4.00%	1.70%	0.95%	0.59%	0.47%
*	Block Randomization of size: 16	6.74%	1.82%	1.63%	0.98%	0.79%
*	Block Randomization of random sizes	4.30%	1.10%	1.06%	0.40%	0.50%
*	Urn Design: UD (1,2)	9.28%	6.41%	4.55%	3.69%	3.05%
*	Urn Design: UD (2,1)	10.21%	6.80%	4.62%	3.75%	3.22%
*	Urn Design: UD (10,2)	10.77%	6.97%	4.62%	3.79%	3.28%
*	Simple Randomization	16.21%	11.24 %	7.71%	6.44%	5.37%
+	Stratified Block Randomization of size: 4	14.40%	8.94%	4.52%	3.03%	2.17%
+	Stratified Block Randomization of size: 6	14.39%	9.96%	5.61%	3.79%	2.52%
+	Stratified Block Randomization of size: 12	15.70%	10.49 %	6.86%	5.10%	3.93%
+	Minimization	5.45%	2.23%	1.31%	0.69%	0.66%
+	Minimization with Increasing Factor	4.89%	1.78%	1.12%	0.58%	0.58%
x	Kolmogorov Smirnov Minimization	5.35%	2.08%	1.28%	0.70%	0.65%
x	Rank-Minimization	5.24%	2.25%	1.36%	0.79%	0.71%

Table 5.2. Comparison of results for imbalance ratio in binary covariate

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215
*	Block Randomization of size: 4	16.14 %	11.59 %	7.84%	6.87%	5.54%

*	Block Randomization of size: 6	15.89 %	11.35 %	7.53%	6.63%	5.35%
*	Block Randomization of size: 8	16.01 %	11.02 %	7.67%	6.65%	5.43%
*	Block Randomization of size: 16	17.03 %	11.44 %	7.86%	6.37%	5.75%
*	Block Randomization of random sizes	16.27 %	11.07 %	8.27%	6.29%	5.45%
*	Urn Design: UD (1,2)	18.23 %	12.78 %	9.19%	7.37%	6.38%
*	Urn Design: UD (2,1)	18.19 %	13.52 %	9.10%	7.68%	6.41%
*	Urn Design: UD (10,2)	19.28 %	13.55 %	8.90%	7.68%	6.38%
*	Simple Randomization	22.59 %	15.88 %	10.75 %	9.28%	7.57%
+	Stratified Block Randomization of size: 4	19.95 %	12.63 %	6.21%	4.29%	2.97%
+	Stratified Block Randomization of size: 6	21.10 %	14.03 %	7.99%	5.34%	3.63%
+	Stratified Block Randomization of size: 12	22.01 %	14.82 %	9.53%	7.30%	5.50%
+	Minimization	7.77%	3.94%	1.95%	1.29%	0.93%
+	Minimization with Increasing Factor	6.86%	3.32%	1.61%	1.10%	0.80%
x	Kolmogorov Smirnov Minimization	6.76%	3.52%	1.68%	1.15%	0.85%
x	Rank-Minimization	7.24%	3.75%	1.75%	1.25%	0.87%

Table 5.3. Comparison of results for imbalance ratio in 5-level covariate

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215
*	Block Randomization of size: 4	31.22 %	22.54 %	15.45 %	13.00 %	10.83 %

*	Block Randomization of size: 6	31.77 %	22.75 %	15.35 %	13.19 %	10.75 %
*	Block Randomization of size: 8	31.10 %	22.41 %	15.79 %	12.90 %	10.75 %
*	Block Randomization of size: 16	31.34 %	22.43 %	15.76 %	12.98 %	10.96 %
*	Block Randomization of random sizes	31.50 %	22.55 %	15.39 %	13.02 %	10.69 %
*	Urn Design: UD (1,2)	32.91 %	23.13 %	16.13 %	13.41 %	11.22 %
*	Urn Design: UD (2,1)	32.84 %	22.90 %	16.10 %	13.58 %	11.17 %
*	Urn Design: UD (10,2)	32.95 %	23.67 %	16.42 %	13.29 %	11.45 %
*	Simple Randomization	34.72 %	24.90 %	17.38 %	14.56 %	12.15 %
+	Stratified Block Randomization of size: 4	31.92 %	19.75 %	9.88%	6.82%	4.73%
+	Stratified Block Randomization of size: 6	33.56 %	21.87 %	12.48 %	8.35%	5.58%
+	Stratified Block Randomization of size: 12	33.70 %	23.57 %	15.14 %	11.65 %	8.75%
+	Minimization	19.22 %	10.63 %	5.26%	3.62%	2.66%
+	Minimization with Increasing Factor	17.49 %	9.34%	4.48%	3.18%	2.20%
x	Kolmogorov Smirnov Minimization	16.33 %	8.62%	4.09%	2.83%	2.01%
x	Rank-Minimization	17.48 %	9.01%	4.24%	2.95%	2.06%

Table 5.4 Comparison of results for imbalance ratio in continuous covariate

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215
*	Block Randomization of size: 4	26.84 %	20.03 %	13.37 %	11.28 %	9.30%

*	Block Randomization of size: 6	26.15 %	19.86 %	13.48 %	11.33 %	9.44%
*	Block Randomization of size: 8	27.78 %	19.50 %	13.58 %	11.11 %	9.44%
*	Block Randomization of size: 16	27.97 %	19.62 %	13.53 %	11.57 %	9.16%
*	Block Randomization of random sizes	27.70 %	19.40 %	13.22 %	11.06 %	9.41%
*	Urn Design: UD (1,2)	28.46 %	20.79 %	14.05 %	11.90 %	9.93%
*	Urn Design: UD (2,1)	29.15 %	20.71 %	14.26 %	12.07 %	10.05 %
*	Urn Design: UD (10,2)	28.63 %	20.86 %	13.90 %	11.85 %	10.07 %
*	Simple Randomization	31.18 %	22.93 %	15.60 %	12.91 %	10.93 %
+	Stratified Block Randomization of size: 4	27.50 %	18.18 %	8.81%	6.13%	4.18%
+	Stratified Block Randomization of size: 6	28.62 %	19.96 %	11.01 %	7.65%	4.91%
+	Stratified Block Randomization of size: 12	30.93 %	21.58 %	13.53 %	10.57 %	7.81%
+	Minimization	13.38 %	8.59%	3.77%	2.83%	1.76%
+	Minimization with Increasing Factor	11.88 %	7.32%	2.95%	2.40%	1.41%

Table 5.5 Comparison of results for imbalance ratio in continuous covariate

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215

*	Block Randomization of size: 4	43.64 %	44.95 %	47.53 %	48.51 %	48.84 %
*	Block Randomization of size: 6	46.84 %	48.48 %	46.82 %	48.30 %	49.47 %
*	Block Randomization of size: 8	47.07 %	46.74 %	47.83 %	47.76 %	48.17 %
*	Block Randomization of size: 16	46.26 %	46.67 %	49.12 %	47.95 %	47.92 %
*	Block Randomization of random sizes	45.97 %	46.92 %	48.11 %	46.67 %	48.97 %
*	Urn Design: UD (1,2)	46.05 %	48.26 %	47.92 %	47.64 %	49.33 %
*	Urn Design: UD (2,1)	45.58 %	46.66 %	48.63 %	47.33 %	47.53 %
*	Urn Design: UD (10,2)	46.47 %	47.12 %	46.76 %	47.38 %	47.24 %
*	Simple Randomization	46.44 %	47.44 %	49.02 %	46.72 %	46.74 %
x	Kolmogorov Smirnov Minimization	27.89 %	22.02 %	14.40 %	11.06 %	8.54%
x	Rank-Minimization	44.47 %	42.82 %	42.98 %	43.38 %	46.12 %

5.2 Performance on Data with Random Pattern

Table 5.6 lists the results marginal imbalance of the different methods between group 0 and group 1, without taking any covariates into consideration. Table 5.7 and 5.8 present the imbalance ratio between group 0 and group 1 for the two categorical covariates (binary and 5-level), while Table 5.9 and 5.10 list the imbalance for the continuous covariate.

The results obtained from the randomization of data that do not follow any specifically assigned pattern closely matched the results of the data with strong pattern. As seen in the previous section of Performance for strong –patterned data, imbalance scores improved for nearly all considered criteria, except for the continuous data. Minimization provided the least imbalance scores when considering the categorical and discretized covariates. Permuted Block Randomization had the lowest scores for the marginal imbalance. The application of the Kolmogorov Smirnov Minimization returned the best imbalance scores which consistently decreased as sample sizes increased.

Table 5.6. Comparison of results for marginal imbalance ratio

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215
*	Block Randomization of size: 4	4.00%	1.36%	0.95%	0.45%	0.47%
*	Block Randomization of size: 6	4.00%	1.56%	1.15%	0.00%	0.47%
*	Block Randomization of size: 8	4.00%	1.75%	0.95%	0.61%	0.47%
*	Block Randomization of size: 16	6.59%	2.01%	1.63%	1.00%	0.79%
*	Block Randomization of random sizes	4.31%	1.04%	1.04%	0.37%	0.51%
*	Urn Design: UD (1,2)	9.62%	6.41%	4.57%	4.02%	3.06%
*	Urn Design: UD (2,1)	10.50 %	6.45%	4.65%	3.81%	3.19%
*	Urn Design: UD (10,2)	11.02 %	6.63%	4.76%	3.88%	3.29%
*	Simple Randomization	16.79 %	11.26 %	7.62%	6.50%	5.33%
+	Stratified Block Randomization of size: 4	14.98 %	9.18%	4.41%	3.00%	2.14%

+	Stratified Block Randomization of size: 6	14.93 %	9.68%	5.65%	3.91%	2.52%
+	Stratified Block Randomization of size: 12	15.62 %	10.52 %	6.81%	5.30%	3.99%
+	Minimization	5.40%	2.14%	1.29%	0.70%	0.64%
+	Minimization with Increasing Factor	4.71%	1.81%	1.21%	0.54%	0.57%
x	Kolmogorov Smirnov Minimization	5.10%	2.00%	1.30%	0.71%	0.63%
x	Rank-Minimization	5.32%	2.38%	1.36%	0.75%	0.71%

Table 5.7 Comparison of results for imbalance ratio in binary covariate

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215
*	Block Randomization of size: 4	16.01 %	11.44 %	7.58%	6.67%	5.46%
*	Block Randomization of size: 6	16.14 %	11.27 %	7.76%	6.61%	5.47%
*	Block Randomization of size: 8	16.00 %	11.45 %	7.99%	6.73%	5.50%
*	Block Randomization of size: 16	17.05 %	11.20 %	8.14%	6.75%	5.57%
*	Block Randomization of random sizes	16.28 %	11.33 %	7.95%	6.31%	5.32%
*	Urn Design: UD (1,2)	18.33 %	12.83 %	9.09%	7.62%	6.24%
*	Urn Design: UD (2,1)	18.35 %	13.61 %	8.96%	7.30%	6.54%
*	Urn Design: UD (10,2)	19.23 %	13.03 %	9.14%	7.49%	6.38%
*	Simple Randomization	22.98 %	16.08 %	11.06 %	9.26%	7.59%
+	Stratified Block Randomization of size: 4	20.46 %	12.85 %	6.25%	4.35%	3.01%

+	Stratified Block Randomization of size: 6	20.89 %	13.83 %	7.85%	5.34%	3.56%
+	Stratified Block Randomization of size: 12	21.43 %	15.19 %	9.49%	7.54%	5.53%
+	Minimization	7.73%	4.12%	1.81%	1.31%	0.93%
+	Minimization with Increasing Factor	6.75%	3.55%	1.71%	1.11%	0.80%
x	Kolmogorov Smirnov Minimization	6.61%	3.46%	1.68%	1.20%	0.80%
x	Rank-Minimization	7.06%	3.82%	1.76%	1.21%	0.88%

Table 5.8. Comparison of results for imbalance ratio in 5-level covariate

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215
*	Block Randomization of size: 4	31.24 %	22.43 %	15.36 %	13.18 %	10.74 %
*	Block Randomization of size: 6	31.18 %	22.28 %	15.30 %	12.83 %	10.87 %
*	Block Randomization of size: 8	31.12 %	22.35 %	15.26 %	12.89 %	10.74 %
*	Block Randomization of size: 16	31.81 %	22.33 %	15.55 %	13.27 %	10.93 %
*	Block Randomization of random sizes	30.85 %	22.42 %	15.50 %	12.97 %	10.56 %
*	Urn Design: UD (1,2)	32.64 %	23.24 %	16.04 %	13.74 %	11.26 %
*	Urn Design: UD (2,1)	32.94 %	22.97 %	16.33 %	13.37 %	11.51 %
*	Urn Design: UD (10,2)	32.58 %	23.60 %	16.67 %	13.37 %	11.45 %
*	Simple Randomization	35.02 %	25.02 %	17.29 %	14.69 %	11.95 %

+	Stratified Block Randomization of size: 4	31.86 %	20.03 %	9.89%	6.87%	4.86%
+	Stratified Block Randomization of size: 6	32.89 %	21.83 %	12.17 %	8.60%	5.53%
+	Stratified Block Randomization of size: 12	34.10 %	23.24 %	15.05 %	11.76 %	8.79%
+	Minimization	19.14 %	10.57 %	5.33%	3.76%	2.56%
+	Minimization with Increasing Factor	17.58 %	9.49%	4.66%	3.27%	2.28%
x	Kolmogorov Smirnov Minimization	15.92 %	8.65%	4.16%	2.89%	2.03%
x	Rank-Minimization	17.34 %	9.18%	4.26%	3.04%	2.00%

Table 5.9. Comparison of results for imbalance ratio in continuous covariate

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215
*	Block Randomization of size: 4	27.42 %	19.59 %	13.29 %	11.21 %	9.48%
*	Block Randomization of size: 6	27.44 %	19.72 %	13.39 %	11.35 %	9.48%
*	Block Randomization of size: 8	27.26 %	19.52 %	13.40 %	11.36 %	9.26%
*	Block Randomization of size: 16	28.01 %	20.11 %	13.78 %	11.21 %	9.40%
*	Block Randomization of random sizes	27.12 %	19.80 %	13.65 %	11.24 %	9.65%
*	Urn Design: UD (1,2)	28.54 %	20.76 %	14.25 %	12.12 %	10.05 %
*	Urn Design: UD (2,1)	28.22 %	21.03 %	14.16 %	12.29 %	9.95%
*	Urn Design: UD (10,2)	29.42 %	20.62 %	14.25 %	11.97 %	10.06 %

*	Simple Randomization	31.82 %	22.65 %	15.25 %	13.05 %	10.90 %
+	Stratified Block Randomization of size: 4	27.91 %	17.82 %	8.93%	6.08%	4.13%
+	Stratified Block Randomization of size: 6	28.68 %	19.64 %	11.03 %	7.59%	4.92%
+	Stratified Block Randomization of size: 12	30.31 %	21.35 %	13.62 %	10.68 %	7.80%
+	Minimization	13.86 %	8.32%	3.63%	2.83%	1.77%
+	Minimization with Increasing Factor	12.14 %	7.31%	2.91%	2.40%	1.42%

Table 5.10. Comparison of results for imbalance ratio in continuous covariate

	Method of Randomization	Ratio for size 25	Ratio for size 50	Ratio for size 105	Ratio for size 150	Ratio for size 215
*	Block Randomization of size: 4	47.16%	45.98%	46.82%	45.45%	45.06%
*	Block Randomization of size: 6	46.37%	46.21%	47.45%	46.75%	47.45%
*	Block Randomization of size: 8	45.82%	47.52%	47.43%	47.48%	48.09%
*	Block Randomization of size: 16	46.24%	47.01%	47.02%	48.57%	46.54%
*	Block Randomization of random sizes	45.88%	45.57%	48.60%	47.44%	48.02%

*	Urn Design: UD (1,2)	46.18%	46.10%	48.36%	48.89%	46.80%
*	Urn Design: UD (2,1)	47.19%	46.97%	47.88%	47.84%	49.64%
*	Urn Design: UD (10,2)	46.37%	48.12%	47.93%	46.95%	48.24%
*	Simple Randomization	47.41%	46.23%	47.52%	47.51%	48.17%
x	Kolmogorov Smirnov Minimization	26.66%	21.62%	13.63%	10.62%	8.61%
x	Rank-Minimization	42.63%	44.83%	43.05%	42.17%	43.63%

CHAPTER 6

SOFTWARE

Based on the obtained results, we implemented a software named RADS, which stands for Randomization Analysis Driven Software. RADS is a stand-alone application available for Windows and Mac OS X. Our freeware implements all the methods studied in our project and it offers numerous options of utilization. Notably, one can use RADS to randomly allocate subjects into a two-branch study. The main features of the software are listed below:

- Group Allocation of subjects on a 1-to-1 basis (continuous enrollment)
- Group Allocation of subjects whose information are already recorded
- Save Randomization Allocation Project to later import and keep enrollment
- Blind Allocation Results to prevent Bias
- Equal Balance of Groups marginally and with respect to designated covariates
- Designated balance of groups, i.e. 4 subjects in group 0 to 2 subjects in group 1.
- Provides user with the most best suggestion of method to employ for desired purpose
- Export the results as a csv file
- Allow user to provide weights of importance for selected covariates
- Allow user to provide specific seed number to reproduce results

In figures 6.1, 6.2, 6.3, 6.4, 6.5, and 6.6 we provide snapshots of our software

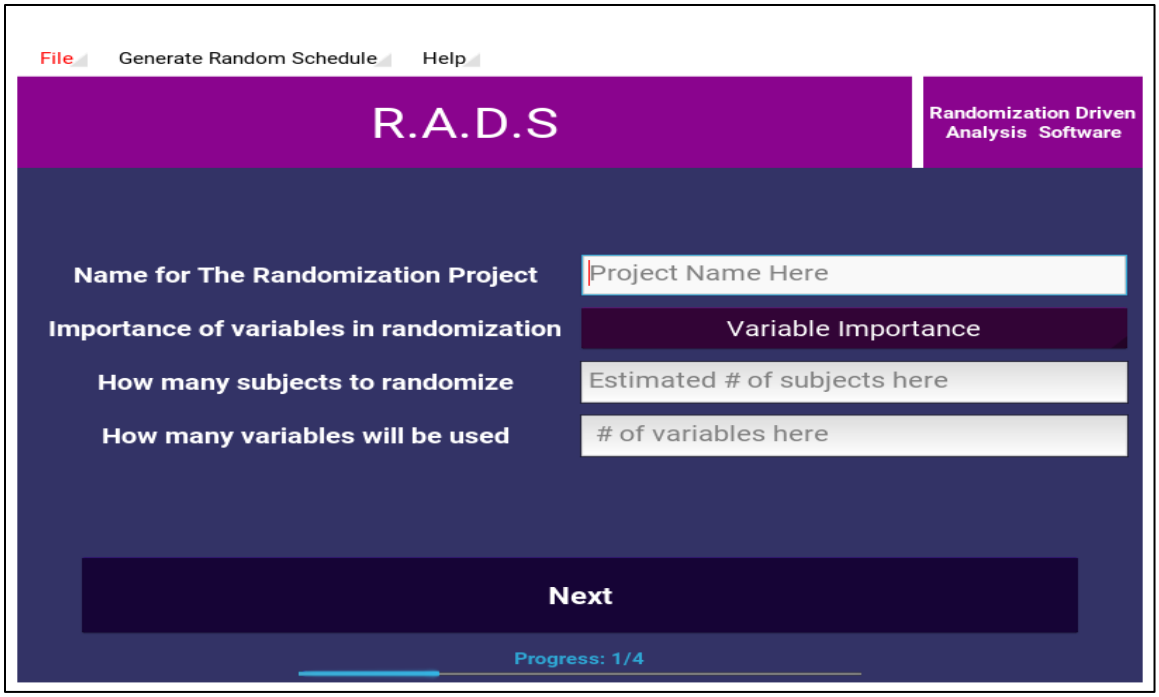

Figure 6.1 Illustration for the Homepage in RADS

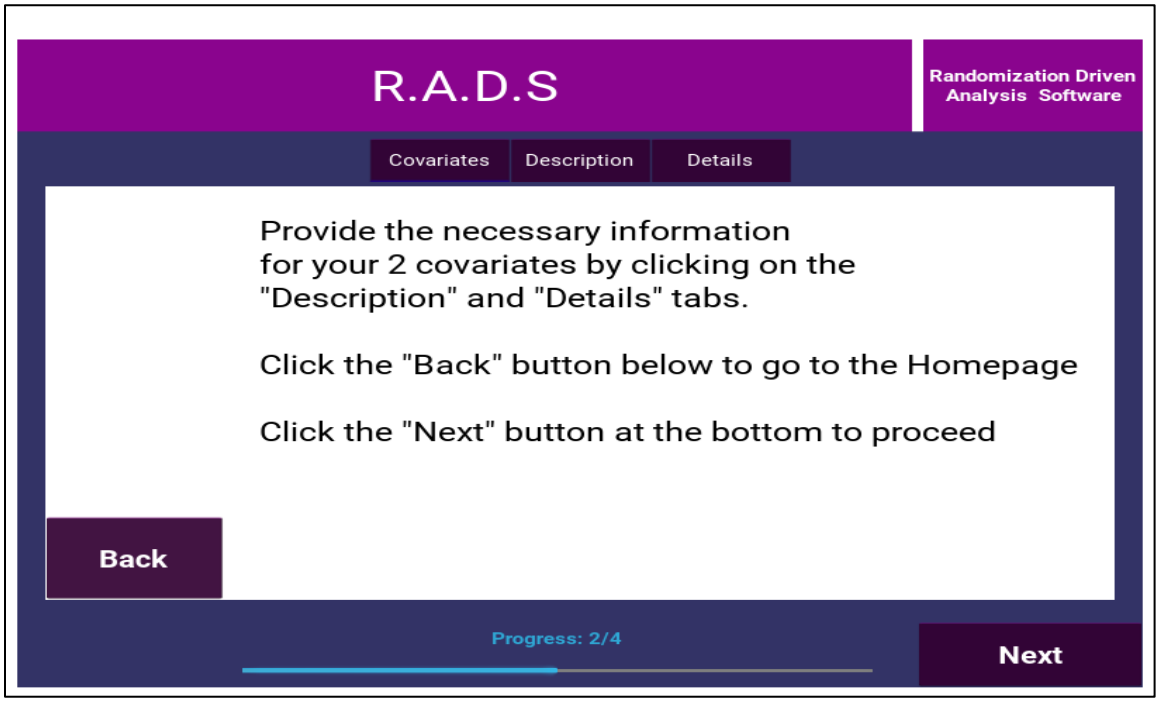

Figure 6.2 Illustration of input details and descriptions in RADS

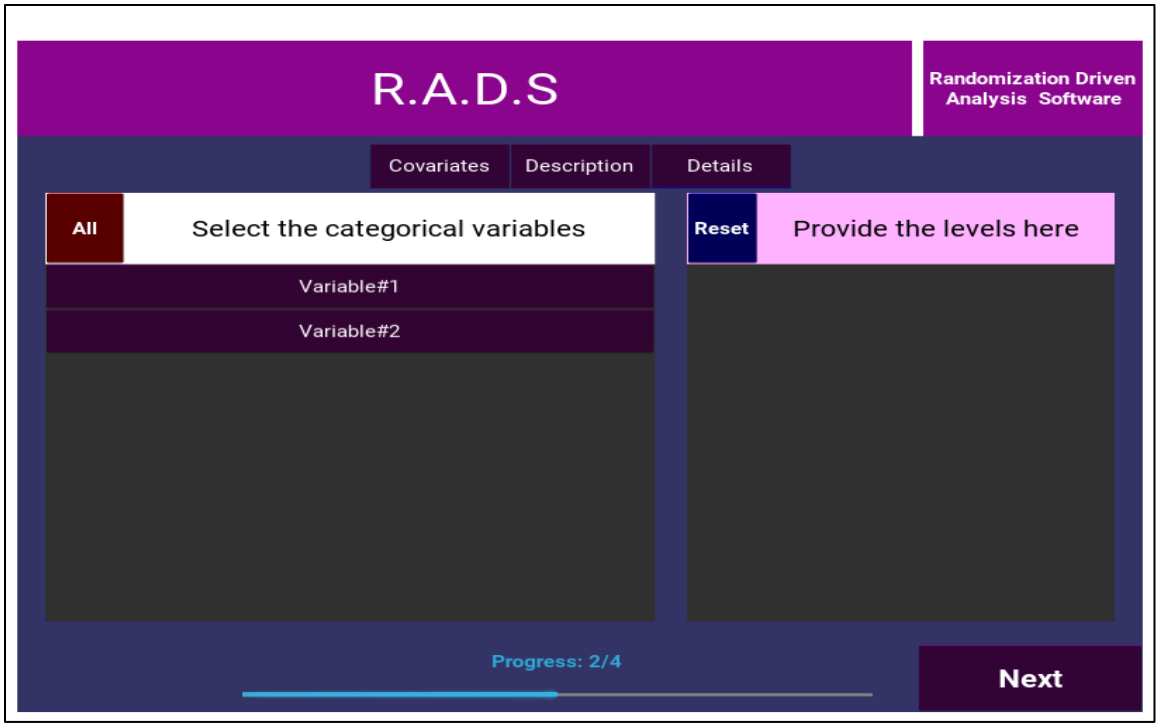

Figure 6.3 Illustration of additional options of data entry in RADS

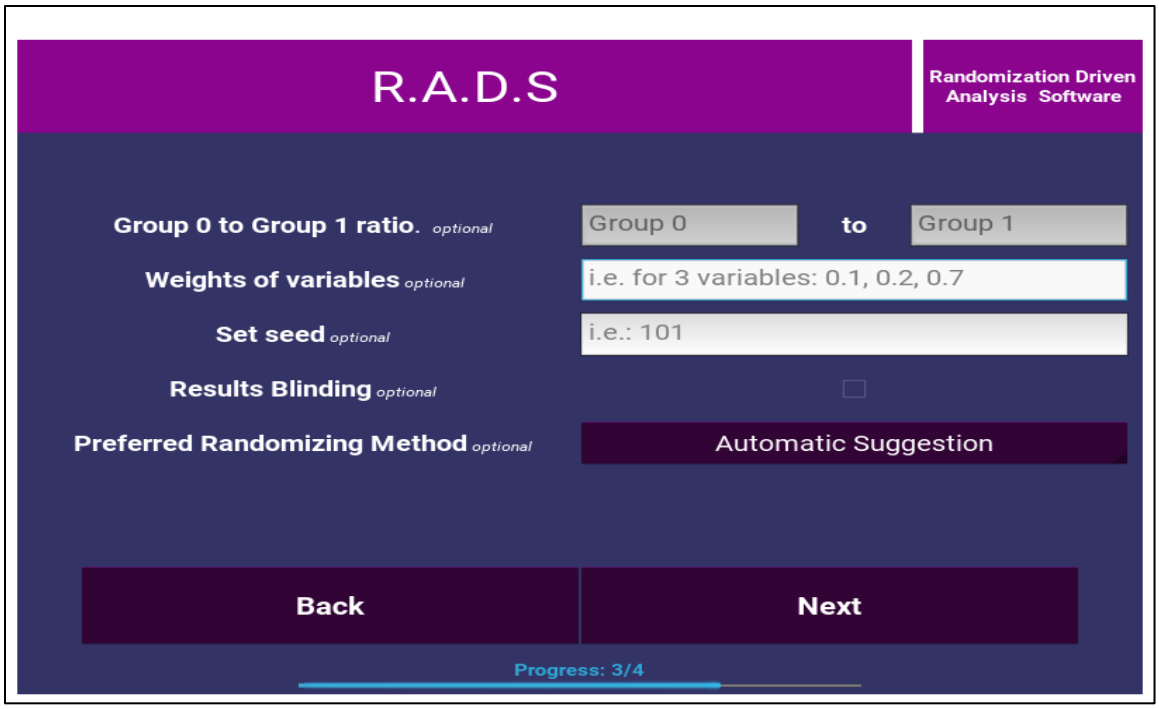

Figure 6.4 Illustration of randomization in RADS on 1-to-1 basis

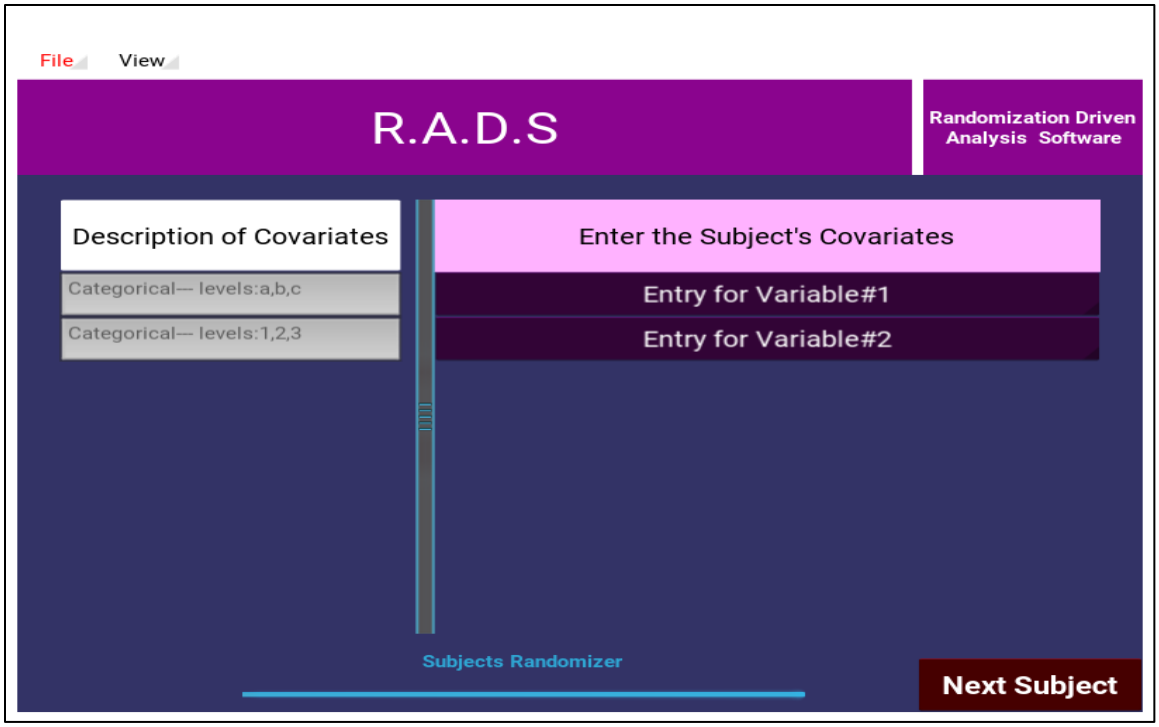

Figure 6.5 Illustration of subject randomizer in action

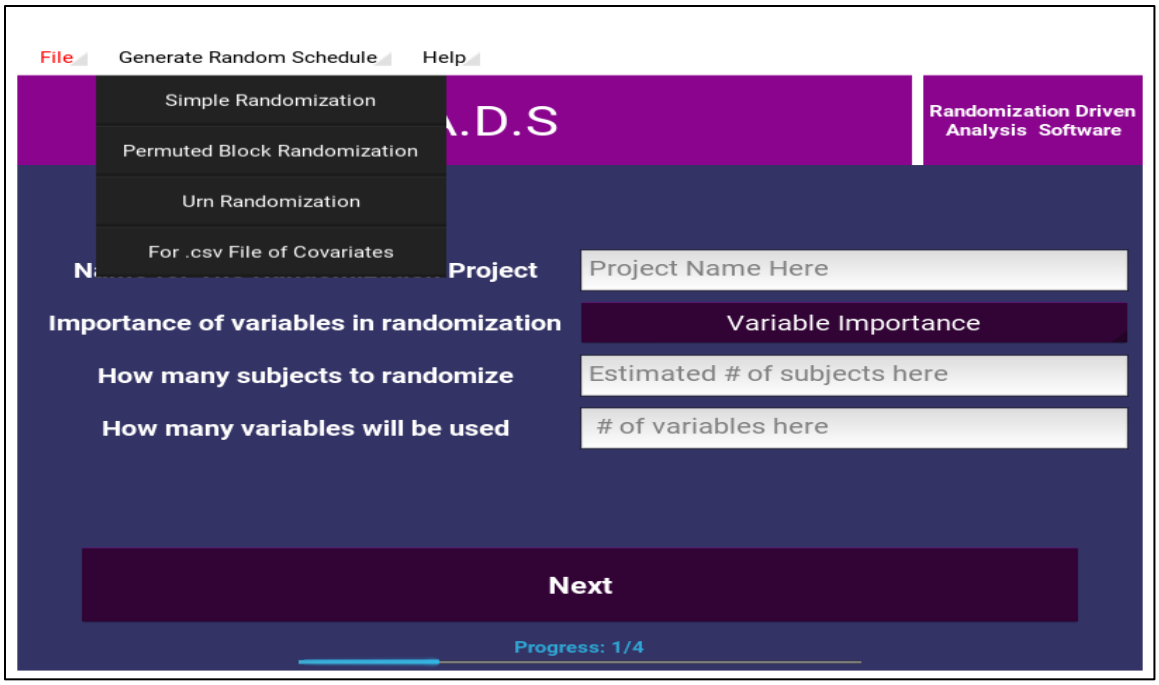

Figure 6.6 Illustration of randomization in RADS for pre-recorded information

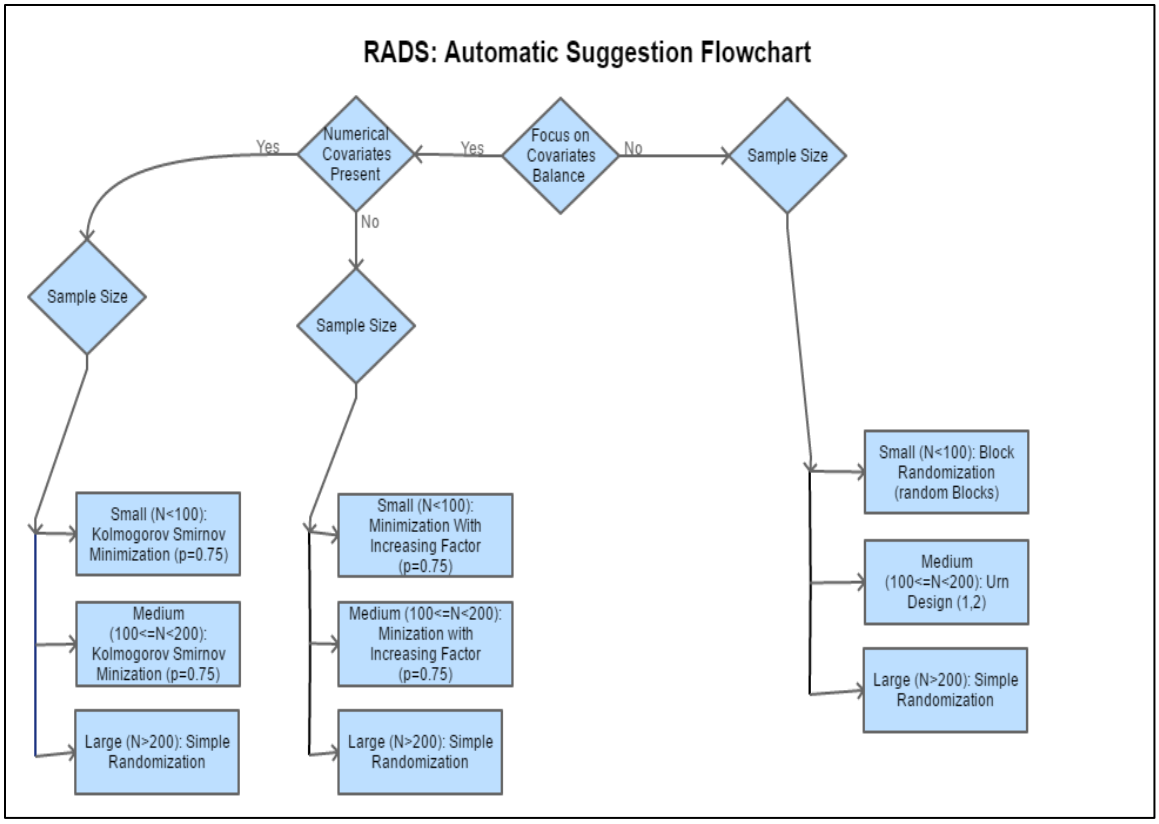

Figure 6.7 Flowchart used for the backbone of RADS randomizer engine

CHAPTER 7

DISCUSSION

The results obtained from our simulation study closely match the results found in the literature. In our simulation, we evaluated the performance of the randomization methods of SRA, PB, UD, SPB, and MIN.

In terms of marginal imbalance, PB performed the best. Amongst our selection of choices for the block size, block size of 4 and 6 returned the smallest marginal imbalance and block size 16 the largest, however, as stated in the literature, (JM, JP, & LJ, 1988), we recommend the utilization of random blocks. Applying random blocks sizes represent the best compromise for they considerably reduce the chances of selection bias and still offer optimal performance. The difference between the mean marginal imbalance for random blocks and the mean marginal imbalance of blocks 4 and 6 is very minimal.

Even though the Minimization methods returned slightly higher imbalance ratios than PB, their performance was considerably better than the next UD. For methods that are independent of subjects' covariates, UD represents a good compromise between the restricted PB and the non-restricted SRA. SPB and SRA in general had the worst marginal imbalance. Since marginal imbalance decreases as sample size increases, it is recommended to use SRA when the sample is larger than 200 for it is unlikely that a considerable imbalance will occur (JM, JP, & LJ, 1988).

For the categorical covariates, the application of the Minimization-based methods returned the best results. Particularly, our method of I-MIN slightly improved the

imbalance scores obtained from MIN that has a fixed standard probability of 0.75. Additionally, we also observed a noticeable superior performance from the application of our KS-MIN compared to the application of R-MIN. Notably, our implemented KS-MIN provided better results than I-MIN.

For the continuous covariate, we compared the performances of the methods that do not use the subjects' covariates to the methods of Minimization. The first type of Minimization (MIN and I-MIN) involved the discretization of the continuous covariate in different quartiles while the second type of Minimization (KS-MIN and R-MIN) randomized the continuous covariate as it was. For the discretized numerical covariate, our I-MIN returned better results. The results obtained from SPB do not differ from the methods of randomization independent of the covariates' information until the sample size of 105. Beyond a sample size of 100 (small size), we notice a considerable improvement of performance for SPB for blocks 4 and 6. However, the application of SPB with block size of 16 do not differ from the methods of reference. For the second type of Randomization, the best method when comparing the distribution of the continuous variable is our method of KS-MIN. The imbalance scores of the latter were significantly lower than all other methods. Additionally, the scores obtained from the KS-MIN method were the only ones to decrease over time.

Beyond the issues of performance, the choice of a randomization should involve the possibilities of bias (selection and accidental), the complexity of the algorithm, and the applicable statistical methods. Our implemented software RADS provides the means to reduce selection bias while carrying all operations on the user's behalf. While our simulation study points that the methods of Minimization offer significantly better results

than the other methods in most scenarios, one must be aware that an important reason why there is still some reluctance in their acceptation is due to the uncertainty of their statistical analysis (Stigsby & Taves, 2010). In the case of the UD, PB, and SPB it is possible to ignore the method of randomization and carry any analysis if the sample comes from a homogenous population (JM, JP, & LJ, 1988). With SRA, one may carry any statistical test for samples larger than 200 (JM, JP, & LJ, 1988). In our project we only focused on the balance of the groups because, using a population based-model, the equality of the groups would maximize power.

Our choice of the Kolmogorov Smirnov test to implement our method of minimization and to also measure the imbalance of the numerical covariate relies on the fact that its implementation in Python was reliable. Unlike other more powerful and recommended tests like the Anderson-Darling test, the implementation of the Kolmogorov Smirnov test did not have issues.

Our results have demonstrated the difference in performance of different randomization methods under multiple scenarios. We advise one to more thoroughly think of the implications of the desired method of randomization prior to carrying that method, especially for the statistical methods that would be applicable for that method. Additional investigation should also be done on the statistical implication of our methods of Kolmogorov Smirnov Minimizations and Minimization with Increasing-Factor.

REFERENCES

Dettori J. (2010). The random allocation process: two things you need to know. EBSJ *Evidence Based Spine-Care Journal*, 7-9

Efird, J. (2011). Blocked Randomization with Randomly Selected Block Sizes. *International Journal of Environmental Research and Public Health, 15-20*

E, S., & Cousineau, D. (n.d.). Comparing Distributions: The two-Sample Anderson-Darling Test as an alternative to the Kolmogorov-Smirnov Test. *Journal of Applied Quantitative Methods.*

Hedden, S.L., Woolson, R.F., & Malcom, R. (2006). Randomization in substance abuse clinical trials. *Substance Abuse Treatment, Prevention, and Policy*

JM, L., JP, M., & LJ, W. (1988). Randomization in clinical trials: conclusions and recommendations. *PubMed, 365-374.*

Kang, M., Ragan, B.G., & Park, J.-H. (2008). Issues in Outcomes Research: An Overview of Randomization Techniques for Clinical Trials. *Journal of Athletic Training, 215-221*

Kim, J. (2014). How to Do Random Allocation (Randomization). *Clinics Orthopedic Surgery,* 103-109

Pocok, S.J., & Simon, R. (1975). Sequential Treatment Assignment with Balancing for Prognostic Factors in the Controlled Clinical Trial. *Biometrics, 103-115*

Stigsby, B., & Taves, D. R. (2010). Rank-Minimization for balanced assignment of subjects in clinical trials. *Contemporary Clinical Trials, 147-150.*

Suresh, K. (2011). An overview of randomization techniques: An unbiased assessment of outcome in clinical research. *Journal of Human Reproductive Services, 8-11*

Tu, D., Shalay, K. & Pater, K. (2000). Adjustment of treatment effect for covariates in clinical trials: Statistical and regulatory issues. *Drug Information Journal, 511-523*

Wei, L. J. & Lachin, J. M. (1988). Properties of the Urn Randomization. *Controlled Clinical Trials, 345-364*

APPENDIX A

PYTHON CODES USED

simulator.py

```
from random import seed; seed(10621)
from numpy.random import seed as np_seed; np_seed(1921)
from simple_randomization import *
from block_randomization import *
from stratified_block_randomization import *
from biased_coin_randomization_urn import *
from minimization import *
from minimization_increasing import *
from rank_minimization import *
from minimization_kolmogorov_smirnov import *
from populator import *
import pandas as pd
from random import randint, sample, choice
from pandas import DataFrame
from collections import OrderedDict
from numpy import mean


def machine_run(sample_type,sim_num,sample_size):
    sim_counter=0
    all_names, all_marg_imb, all_margratio_imb, all_cov_imb = [],[],[],[]
    all_cov1, all_cov2, all_cov3        = [],[],[]
    type_of, size_of=[],[]
    m1,m2='categ','numer'
    while sim_counter < sim_num:
        test=[]
        if sample_type== 'random':
            temp=populate_random(sample_size);test=temp[0];test2=temp[1]
        elif sample_type=='irregular':
            temp=populate_irregular(sample_size);test=temp[0];test2=temp[1]
        total_indx_lst_var=list(range(1,len(test[1:])+1))
        method1=sra()
```

```
    method2=block()
    method3=stratified_block()
    method4=biased_coin_urn()
    method5=minimization()
    method6=minimization_growing()
    method7=rank_minimization()
    method8=kolmo_smir_minimization()

    # SIMPLE RANDOMIZATION
    result1=method1.randomize(m1,test[:])
    result2=method1.randomize(m2,test2[:])
    # BLOCK RANDOMIZATION
    result3=method2.randomize(m1,2,test[:])
    result4=method2.randomize(m2,2,test2[:])
    result5=method2.randomize(m1,3,test[:])
    result6=method2.randomize(m2,3,test2[:])
    result7=method2.randomize(m1,4,test[:])
    result8=method2.randomize(m2,4,test2[:])
    result9=method2.randomize(m1,8,test[:])
    result10=method2.randomize(m2,8,test2[:])
    result11=method2.randomize(m1,'random',test[:])
    result12=method2.randomize(m2,'random',test2[:])
    # STRATIFIED RANDOMIZATION
    result13=method3.randomize(total_indx_lst_var,4,test[:])
    result14=method3.randomize(total_indx_lst_var,6,test[:])
    result15=method3.randomize(total_indx_lst_var,12,test[:])
    # URN DESIGN
    result16=method4.randomize(m1,1,2,test[:])
    result17=method4.randomize(m2,1,2,test2[:])
    result18=method4.randomize(m1,10,2,test[:])
    result19=method4.randomize(m2,10,2,test2[:])
    result20=method4.randomize(m1,2,1,test[:])
    result21=method4.randomize(m2,2,1,test2[:])
    # MINIMIZATION
    result22=method5.randomize(total_indx_lst_var,test[:])
    # GROWING MINIMIZATION
    result23=method6.randomize(total_indx_lst_var,test[:])
    # RANK MINIMIZATION
    result24=method7.randomize(total_indx_lst_var,test2[:])
    # KOLMOGOROV-SMIRNOV MINIMIZATION
    result25=method8.randomize(total_indx_lst_var,test2[:])

    # all_result=[result2]

```

```
        all_result=[result1,result2,result3,result4,result5,
            result6,result7,result8,result9,result10,
            result11,result12,result13,result14,result15,
            result16,result17,result18,result19,result20,
            result21,result22,result23,result24,result25]

        name_lst=[];all_imb=[];ratio_imb=[];covar_imb=[]

        for j in list(range(len(all_result))):
            name_lst.append(all_result[j][0])
            all_imb.append(all_result[j][1])
            ratio_imb.append(all_result[j][2])
            covar_imb.append(all_result[j][4])

        sim_counter+=1
        all_names.extend(name_lst)
        all_marg_imb.extend(all_imb)
        all_margratio_imb.extend(ratio_imb)
        all_cov_imb.extend(covar_imb)

    for j in list(range(len(all_cov_imb))):
        all_cov1.append(all_cov_imb[j][0])
        all_cov2.append(all_cov_imb[j][1])
        all_cov3.append(all_cov_imb[j][2])
        type_of.append(sample_type)
        size_of.append(sample_size)

    dict_of_results= OrderedDict([('method name', all_names),
        ('imbalance range', all_marg_imb),
        ('imbalance-range ratio ', all_margratio_imb),
        ('Variable 1', all_cov1),
        ('Variable 2', all_cov2),
        ('Variable 3', all_cov3),
        ('type', type_of),
        ('size', size_of)])
    df=DataFrame(dict_of_results)
    # df=df.set_index('method name')
    return df
sim_num=1000
irr_25=machine_run('irregular',sim_num,25)
irr_50=machine_run('irregular',sim_num,50)
irr_100=machine_run('irregular',sim_num,105)
irr_150=machine_run('irregular',sim_num,150)
irr_200=machine_run('irregular',sim_num,215)
```

```
rand_25=machine_run('random',sim_num,25)
rand_50=machine_run('random',sim_num,50)
rand_100=machine_run('random',sim_num,105)
rand_150=machine_run('random',sim_num,150)
rand_200=machine_run('random',sim_num,215)


full_result=pd.concat([irr_25,irr_50,irr_100,irr_150,irr_200,
rand_25,rand_50,rand_100,rand_150,rand_200])
# full_result=pd.concat([irr_100,irr_200])

full_result.to_csv("C:\\Users\\StephYves\\OneDrive - University of South Carolina\\Biostatistics\\MASTER's Thesis Work\\thesis\\results\\results.csv")

df0=full_result.loc[(full_result['method name']=='Simple Randomization (numer)')&(full_result['type']=='irregular')&(full_result['size']==150)]
df1=df0['Variable 2']
print(df1)
print(mean(df1.tolist()))
# full_result.loc[full_result['Variable 2']][full_result['method name'=='Simple Randomization (numer)']].tolist()
# g_ratio_1=sorted(df1.iloc[:,2][(df2['size']==25)&(df2['method name']=='Simple Randomization')])
# df1=df.loc[df['type']=='irregular']
path="C:\\Users\\StephYves\\Downloads\\test_results_csv"
df0.to_csv(path+"\\res.csv")
```

populator.py

```
from random import shuffle
from numpy.random import choice as np_choice
from numpy.random import normal as np_normal
from pandas import qcut
from scipy.stats import skewnorm

def populate_random(obs_num):
    entire_lst=[]
    all_obs_lst=list(range(obs_num))
    cov1=np_choice(list('01'),size=obs_num).tolist()                # binary
    cov2=np_choice(list('abcde'),size=obs_num).tolist()              # categorical 5 levels
```

```
    cov3=np_normal(loc=50,scale=5.5,size=obs_num).tolist()            # continuous mu
=50 and sd=5.5 ---normal
    cat_cov3=qcut(cov3,4,labels=['c1','c2','c3','c4']).tolist()       # discretized continuo
us variable
    entire_lst=[[all_obs_lst,cov1,cov2,cat_cov3],[all_obs_lst,cov1,cov2,cov3]]
    return entire_lst
def populate_irregular(obs_num):
    entire_lst=[];  all_cov_lst=[]
    all_obs_lst=list(range(obs_num))
    entire_lst.append(all_obs_lst)
    cov1=np_choice(list('01'),size=obs_num,p=shuffle([0.75,0.25])).tolist()
      # binary
    cov2=np_choice(list('abcde'),size=obs_num,p=shuffle([0.1,0.15,0.45,0.25,0.05])).t
olist()       # categorical 5 levels
    cov3=skewnorm.rvs(4,loc=50,scale=5.5,size=obs_num)
 # continuous mu=50 and sd=5.5 --- skewnormal
    cat_cov3=qcut(cov3,4,labels=['c1','c2','c3','c4']).tolist()                  # dis
cretized continuous variable
    entire_lst=[[all_obs_lst,cov1,cov2,cat_cov3],[all_obs_lst,cov1,cov2,cov3]]
    return entire_lst
```

variable.py

```
from scipy.stats import ks_2samp as ks2
def measure_all_cov1(result_list):
    var_lst=result_list[1:-1]; var_counter=0; all_stratum_imbal=[]
    while var_counter < len(var_lst):
        unique_val=list(set(var_lst[var_counter]))
        val_counter=0;lev_imbal=[];tot_lev_imbal=[]
        while val_counter < len(unique_val):
            temp_indx=[i for i, e in enumerate(var_lst[var_counter]) if e==unique_val[val_counter]]
            count_0=len([result_list[-1][i] for i in temp_indx if result_list[-1][i]==0])
            count_1=len([result_list[-1][i] for i in temp_indx if result_list[-1][i]==1])
            imbal_var=abs(count_0-count_1)
            lev_imbal.append(imbal_var)
            tot_lev_imbal.append(len(temp_indx))
            val_counter+=1
        all_stratum_imbal.append((sum(lev_imbal)/sum(tot_lev_imbal)))
        var_counter+=1
    return all_stratum_imbal

def measure_all_cov2(result_list):                                  # for mixed of both categ and numer
    categ_lst, var_counter, all_imbal =result_list[1:-2], 0, []
    numer_lst=result_list[-2]
    while var_counter < len(categ_lst):
        unique_val=list(set(categ_lst[var_counter]))
        val_counter, lev_imbal, max_lev_imbal=0, [], len(categ_lst[var_counter])
        while val_counter < len(unique_val):
            temp_indx=[i for i, e in enumerate(categ_lst[var_counter]) if e==unique_val[val_counter]]
            count_0=len([result_list[-1][i] for i in temp_indx if result_list[-1][i]==0])
            count_1=len([result_list[-1][i] for i in temp_indx if result_list[-1][i]==1])
            imbal_var=abs(count_0-count_1)
            lev_imbal.append(imbal_var)
            val_counter+=1
        all_imbal.append(sum(lev_imbal)/max_lev_imbal)
        var_counter+=1
    numer_0=[numer_lst[i] for i in list(range(len(result_list[-1]))) if result_list[-1][i]==0]
    numer_1=[numer_lst[i] for i in list(range(len(result_list[-1]))) if result_list[-1][i]==1]
    ans=1-list(ks2(numer_0,numer_1))[-1]
    all_imbal.append(ans)
```

```
    return all_imbal

def measure_categ(indx,variables,assignment):
    cov_0_imbal,cov_1_imbal=[],[]
    var_counter=0;
    while var_counter<len(variables):
        covariate=variables[var_counter]
        same_value=covariate[indx]
        same_value_lst_indx=[i for i, e in enumerate(covariate[:indx]) if e==same_valu
e]
        if len(same_value_lst_indx)==0:
            cov_0_imbal.append(1); cov_1_imbal.append(1)
        else:
            assignment_b4_lst=[assignment[i] for i in same_value_lst_indx]
            g0=assignment_b4_lst.count(0)
            g1=assignment_b4_lst.count(1)
            dif_0, dif_1=abs((g0+1)-g1),abs((g1+1)-g0)
            cov_0_imbal.append(dif_0); cov_1_imbal.append(dif_1)
        var_counter+=1
    return [sum(cov_0_imbal),sum((cov_1_imbal))]
```

biased_coin_randomization_urn.py

```
from random import randint, shuffle, choice
from variable_measure import measure_all_cov1, measure_all_cov2

class biased_coin_urn(object):
    def __init__(self):
        self.type='Urn Design'

    def randomize(self,cov3,num,augmenter,some_list):
        imbalance,group_factor,type_rand,assignment_group =None,[0,1],'',[]
        bgin_lst_0, bgin_lst_1=[0]*num, [1]*num
        bgin_lst_0.extend(bgin_lst_1)
        assignment_pond=bgin_lst_0
        assignment_group.append(choice(group_factor))
        augmenter_lst=[x for x in group_factor if x not in assignment_group]*augmente
r
        assignment_pond.extend(augmenter_lst)
        for person_i in some_list[0][1:]:
            ind_augmenter_lst=[]
            assignment=choice(assignment_pond)
            assignment_group.append(assignment)
            ind_augmenter_lst=[x for x in group_factor if x!=assignment]*augmenter
            assignment_pond.extend(ind_augmenter_lst)
        some_list.append(assignment_group)
        imbalance=abs(assignment_group.count(0)-assignment_group.count(1))

        if cov3=='categ':
            var_imbal=measure_all_cov1(some_list);ratio_imbal=imbalance/len(assignm
ent_group)
        elif cov3=='numer':
            var_imbal=measure_all_cov2(some_list);ratio_imbal=imbalance/len(assignm
ent_group)

        ratio_imbal=imbalance/len(assignment_group)
        type_rand=self.type+" ({},{})--{}".format(num,augmenter,cov3)

        return [type_rand,imbalance,ratio_imbal,[],var_imbal]
```

block_randomization.py

```
from random import randint, shuffle, choice
from math import ceil as roundup
from variable_measure import measure_all_cov1,measure_all_cov2

class block(object):
    def __init__(self):
        self.type='Block Randomization'

    def randomize(self,cov3,num,some_list):
        group_factor,type_rand=[0,1],''
        num_total_assignments,assignment_group=len(some_list[0]),[]
        indx=num_total_assignments

        if isinstance(num,str):
            bk_type=num
            counter=0;block_size_lst=[]
            while counter < num_total_assignments:
                rand_block_size=2*choice([2,3,4])
                block_size_lst.append(rand_block_size)
                counter+=rand_block_size
            for size in block_size_lst:
                block_unit=int(size/2)*group_factor
                shuffle(block_unit)
                assignment_group.extend(block_unit)
        else:
            blk_num=num*2; bk_type=num*2
            full_blocks=int(num_total_assignments/blk_num)
            for unit_block in list(range(roundup(num_total_assignments/blk_num))):
                block_unit=int(blk_num/2)*group_factor
                shuffle(block_unit)
                assignment_group.extend(block_unit)
        assignment_group=assignment_group[:indx]
        some_list.append(assignment_group)
        imbalance=abs(assignment_group.count(0)-assignment_group.count(1))
        ratio_imbal=imbalance/len(assignment_group)
        if cov3=='categ':
            var_imbal=measure_all_cov1(some_list)
        elif cov3=='numer':
            var_imbal=measure_all_cov2(some_list)

        type_rand=self.type+' ({})--{}'.format(bk_type,cov3)

```

```
        return [type_rand,imbalance,ratio_imbal,[],var_imbal]
```

minimization.py

```
from random import  choice
from numpy.random import uniform
from variable_measure import measure_all_cov1


class minimization(object):
    def __init__(self):
        self.type='Minimization '

    def randomize(self,cov_indx_lst,some_list):
        group_factor=[0,1];assignment_group=[]
        temp_cov_lst=[some_list[i] for i in cov_indx_lst]
        num_to_randomize=1
        for first_x_ppl in list(range(num_to_randomize)):
            assignment_group.append(choice(group_factor))
        pers_counter=0
        while pers_counter < len(some_list[0][num_to_randomize:]):
            pers_b4_assignment_lst=[]
            full_lst_indx=pers_counter+num_to_randomize
            cov_0_imbal_lst=[]; cov_1_imbal_lst=[]; var_counter=0;
            while var_counter < len(temp_cov_lst):
                same_cov=temp_cov_lst[var_counter][full_lst_indx]
                same_cov_lst=[]
                same_cov_lst_indx=[i for i, e in enumerate(temp_cov_lst[var_counter][:ful
l_lst_indx]) if e==same_cov]
                if len(same_cov_lst_indx)==0:
                    cov_0_imbal_lst.append(1); cov_1_imbal_lst.append(1)
                else:
                    assignment_b4_lst=[]
                    for loc in same_cov_lst_indx:
                        assignment_b4_lst.append(assignment_group[loc])
                    g0=assignment_b4_lst.count(0)
                    g1=assignment_b4_lst.count(1)
                    dif_0=abs((g0+1)-g1);dif_1=abs((g1+1)-g0)
                    cov_0_imbal_lst.append(dif_0)
                    cov_1_imbal_lst.append(dif_1)
                var_counter+=1
            pers_counter+=1
            sum_imbal_0=sum(cov_0_imbal_lst);sum_imbal_1=sum(cov_1_imbal_lst)

```

```
            pers_rand_num=uniform(0,1)
            if pers_rand_num< 0.75:
                if sum_imbal_0>sum_imbal_1:
                    assignment_group.append(1)
                elif sum_imbal_0<sum_imbal_1:
                    assignment_group.append(0)
                else:
                    assignment_group.append(choice(group_factor))
            else:
                assignment_group.append(choice(group_factor))
        some_list.append(assignment_group)
        imbalance=abs(assignment_group.count(0)-assignment_group.count(1))
        var_imbal=measure_all_cov1(some_list);ratio_imbal=imbalance/len(assignment
_group)

    return [self.type,imbalance,ratio_imbal,[],var_imbal]
```

minimization_increasing.py

```
from random import  choice
from numpy.random import uniform
from variable_measure import measure_all_cov1

class minimization_growing(object):
    def __init__(self):
        self.type='Growing Minimization '

    def randomize(self,cov_indx_lst,some_list):
        group_factor=[0,1];assignment_group=[];tot_people=len(some_list[0])
        temp_cov_lst=[some_list[i] for i in cov_indx_lst]
        num_to_randomize=1;start_prob=0.75
        for first_x_ppl in list(range(num_to_randomize)):
            assignment_group.append(choice(group_factor))
        pers_counter=0
        while pers_counter < len(some_list[0][num_to_randomize:]):
```

```
            pers_b4_assignment_lst=[]
            full_lst_indx=pers_counter+num_to_randomize
            cov_0_imbal_lst=[];cov_1_imbal_lst=[]
            all_similar_cov_lst=[]; cov_assignment_dif=[]
            var_counter=0; temp_cov_container=[]
            while var_counter < len(temp_cov_lst):
                same_cov=temp_cov_lst[var_counter][full_lst_indx]
                same_cov_lst=[]
                same_cov_lst_indx=[i for i, e in enumerate(temp_cov_lst[var_counter][
:full_lst_indx]) if e==same_cov]
                if len(same_cov_lst_indx)==0:
                    cov_0_imbal_lst.append(1); cov_1_imbal_lst.append(1)
                else:
                    assignment_b4_lst=[]
                    for loc in same_cov_lst_indx:
                        assignment_b4_lst.append(assignment_group[loc])
                    g0=assignment_b4_lst.count(0)
                    g1=assignment_b4_lst.count(1)
                    dif_0=abs((g0+1)-g1);dif_1=abs((g1+1)-g0)
                    cov_0_imbal_lst.append(dif_0)
                    cov_1_imbal_lst.append(dif_1)
                var_counter+=1
            start_prob+=choice([0,0.25/tot_people])
            pers_counter+=1
            sum_imbal_0=sum(cov_0_imbal_lst);sum_imbal_1=sum(cov_1_imbal_lst
)

            pers_rand_num=uniform(0,1)
            if pers_rand_num<start_prob:
                if sum_imbal_0>sum_imbal_1:
                    assignment_group.append(1)
                elif sum_imbal_0<sum_imbal_1:
                    assignment_group.append(0)
                else:
                    assignment_group.append(choice(group_factor))
            else:
                assignment_group.append(choice(group_factor))
        some_list.append(assignment_group)
        imbalance=abs(assignment_group.count(0)-assignment_group.count(1))
        var_imbal=measure_all_cov1(some_list);ratio_imbal=imbalance/len(assign
ment_group)

    return [self.type,imbalance,ratio_imbal,[],var_imbal]
```

minimization_kolmogorov_smirnov.py

```
from random import  choice, uniform
from scipy.stats import ks_2samp as ks2
from variable_measure import measure_all_cov2, measure_categ
from math import exp


def measure_numer_kolmo(pers_count,variable,assignment):
    cov_0_imbal,cov_1_imbal=0,0
    current_val=[variable[pers_count]]
    values_b4=variable[:pers_count]

    var_0=[values_b4[i] for i in list(range(len(values_b4))) if assignment[i]==0]
    var_1=[values_b4[i] for i in list(range(len(values_b4))) if assignment[i]==1]

    var_0_scene0=var_0+current_val
    var_1_scene1=var_1+current_val

    ans_for_0=1-list(ks2(var_0_scene0,var_1))[-1]
    ans_for_1=1-list(ks2(var_1_scene1,var_0))[-1]

    cov_0_imbal=exp(ans_for_0)
    cov_1_imbal=exp(ans_for_1)
    return [cov_0_imbal,cov_1_imbal]

class kolmo_smir_minimization(object):
    def __init__(self):
        self.type='Kolmogorov Smirnov Minimization'

    def randomize(self,cov_indx_lst,some_list):
        group_factor=[0,1];assignment_group=[]
        num_to_randomize=2
        assignment_group.extend(group_factor) #forced assignment
        pers_counter=0
        while pers_counter < len(some_list[0][num_to_randomize:]):
            full_lst_indx=pers_counter+num_to_randomize
            cov_0_imbal_lst, cov_1_imbal_lst=[],[]
            categ_ans= measure_categ(full_lst_indx,some_list[1:3],assignment_group)
            numer_ans= measure_numer_kolmo(full_lst_indx,some_list[3],assignment_gr
oup)
            cov_0_imbal_lst.append(categ_ans[0]);cov_1_imbal_lst.append(categ_ans[1])
```

```
            cov_0_imbal_lst.append(numer_ans[0]);cov_1_imbal_lst.append(numer_ans[1])

            sum_imbal_0, sum_imbal_1 =sum(cov_0_imbal_lst), sum(cov_1_imbal_lst)
            pers_rand_num=uniform(0,1)
            if pers_rand_num< 0.75:
                if sum_imbal_0>sum_imbal_1:
                    assignment_group.append(1)
                elif sum_imbal_0<sum_imbal_1:
                    assignment_group.append(0)
                else:
                    assignment_group.append(choice(group_factor))
            else:
                assignment_group.append(choice(group_factor))
            pers_counter+=1

        some_list.append(assignment_group)
        imbalance=abs(assignment_group.count(0)-assignment_group.count(1))
        var_imbal=measure_all_cov2(some_list);  ratio_imbal=imbalance/len(assignment_group)
        return [self.type,imbalance,ratio_imbal,[],var_imbal]
```

rank_minimization.py

```
from random import  choice, uniform
from scipy.stats import rankdata as rank
from numpy import mean as avg
from math import log10
from variable_measure import measure_all_cov2, measure_categ

def measure_numer_rank(indx,variable,assignment):
    full_values=variable[:indx+1]
    rank_lst=rank(full_values).tolist()
    ranks_b4=rank_lst[:-1]
    assigned_ind_0=[i for i, e in enumerate(assignment) if e==0]
    assigned_ind_1=[i for i, e in enumerate(assignment) if e==1]

    rank_0=[ranks_b4[i] for i in assigned_ind_0]
    rank_1=[ranks_b4[i] for i in assigned_ind_1]
    if_rank0=rank_0+[rank_lst[-1]]
    if_rank1=rank_1+[rank_lst[-1]]

```

```
    mean_rank_sum_0=avg([sum(if_rank0),sum(rank_1)])
    mean_rank_sum_1=avg([sum(rank_0),sum(if_rank1)])
    d0= (sum(if_rank0)-mean_rank_sum_0)**2
    d1= (sum(if_rank1)-mean_rank_sum_1)**2

    sq0, sq1=[d0]*2,[d1]*2
    imb_0=log10(sum(sq0)+0.1); imb_1=log10(sum(sq1)+0.1)
    cov_0_imbal=imb_0
    cov_1_imbal=imb_1
    return [cov_0_imbal,cov_1_imbal]

class rank_minimization(object):
    def __init__(self):
        self.type='Rank Minimization'

    def randomize(self,cov_indx_lst,some_list):
        group_factor=[0,1];assignment_group=[]
        num_to_randomize=1;assignment_group.append(choice(group_factor))
        pers_counter=0
        while pers_counter < len(some_list[0][num_to_randomize:]):
            full_lst_indx=pers_counter+num_to_randomize
            cov_0_imbal_lst, cov_1_imbal_lst=[],[]
            categ_ans= measure_categ(full_lst_indx,some_list[1:3],assignment_group)
            numer_ans= measure_numer_rank(full_lst_indx,some_list[3],assignment_gro
up)
            cov_0_imbal_lst.append(categ_ans[0]);cov_1_imbal_lst.append(categ_ans[1])

            cov_0_imbal_lst.append(numer_ans[0]);cov_1_imbal_lst.append(numer_ans[
1])
            sum_imbal_0, sum_imbal_1 =sum(cov_0_imbal_lst), sum(cov_1_imbal_lst)
            pers_rand_num=uniform(0,1)
            if pers_rand_num< 0.75:
                if sum_imbal_0>sum_imbal_1:
                    assignment_group.append(1)
                elif sum_imbal_0<sum_imbal_1:
                    assignment_group.append(0)
                else:
                    assignment_group.append(choice(group_factor))
            else:
                assignment_group.append(choice(group_factor))
            pers_counter+=1

        some_list.append(assignment_group)
        imbalance=abs(assignment_group.count(0)-assignment_group.count(1))
```

```
        var_imbal=measure_all_cov2(some_list);  ratio_imbal=imbalance/len(assignment_group)

        return [self.type,imbalance,ratio_imbal,[],var_imbal]
```

simple_randomization.py

```
from random import choice
from variable_measure import measure_all_cov1, measure_all_cov2
from numpy.random import randint


class sra(object):
    def __init__(self):
        self.type='Simple Randomization'

    def randomize(self,cov3,x_list):
        some_list=x_list
        assignment_group=randint(2,size=len(x_list[0])).tolist()
        some_list.append(assignment_group)
        imbalance=abs(assignment_group.count(0)-assignment_group.count(1))
        ratio_imbal=imbalance/len(assignment_group)
        if cov3=='categ':
            var_imbal=measure_all_cov1(some_list)
        elif cov3=='numer':
            var_imbal=measure_all_cov2(some_list)
        type_of=self.type+' ({})'.format(cov3)

        return [type_of,imbalance,ratio_imbal,[],var_imbal]
```

stratified_block_randomization.py

```
from random import randint, shuffle
from math import ceil as roundup
from numpy import prod
from itertools import product as cart_prod
from variable_measure import measure_all_cov1

class stratified_block(object):
    def __init__(self):
        self.type='Stratified Block Randomization'
```

```
    def randomize(self,indx_list_of_covariates,block_size,some_list):
        group_factor,type_rand=[0,1],''
        num_total_assignments=len(some_list[0]);
        temp_lst_string=str([list(set(some_list[i])) for i in indx_list_of_covariates])[1:-1]
        temp_lst_string_function='cart_prod('+temp_lst_string+')'
        temp_lst=eval(temp_lst_string_function)
        all_stratum_lst=[list(item) for item in temp_lst]
        num_of_stratum=len(all_stratum_lst)
        full_block_num=int(roundup(num_total_assignments/block_size))
        assignment_group=[]; strata_assignments=[]

        for indx in list(range(num_of_stratum)):
            temp_subgrp=[]
            for unit_blck in list(range(full_block_num)):
                block_unit=int(block_size/2)*group_factor
                shuffle(block_unit)
                temp_subgrp.extend(block_unit)
            strata_assignments.append(temp_subgrp)
        for i in some_list[0]:
            ind_characteristics=[some_list[j][i] for j in indx_list_of_covariates]
            indx=all_stratum_lst.index(ind_characteristics)
            assignment_group.append(strata_assignments[indx][0])
            del strata_assignments[indx][0]
        some_list.append(assignment_group)
        imbalance=abs(assignment_group.count(0)-assignment_group.count(1))
        var_imbal=measure_all_cov1(some_list);ratio_imbal=imbalance/len(assignment_group)
        type_rand=self.type+" ({})".format(block_size)

        return [type_rand,imbalance,ratio_imbal,[],var_imbal]
```