Re-construction of Reference Population and Generating Weights by Decision Tree[1]

Wei Wan, Ph.D.

Junior Faculty Research Fellow, DEOMI

Summer 2017

Claflin University

Orangeburg, SC 29115

DEFENSE EQUAL OPPORTUNITY MANAGEMENT INSTITUTE

RESEARCH, DEVELOPMENT, AND STRATEGIC INITIATIVES DIRECTORATE, Patrick AFB, FL

Submitted to Dr. Daniel P. McDonald, Executive Director of Research

July21, 2017

Technical Report #11-17

[1] The opinions expressed in this report ore those of the author and should not be construed to represent the official position of the U.S. military services, or the Department of Defense, or DEOMI.

Abstract

The DEOCS received responder data, which does not contain non-responses, directly through the survey as well as unit population data through DMDC. To estimate statistical characteristic of the population, the DEOCS team has merged the unit population data into survey data, which is a dataset of ~260,000 cases. However, the non-responses rate is more than 60%, so the responder data may not be representative of population. In order to compensate for non-responses, weighting is needed to avoid bias. In order for computing post-stratification weights, the first step is to design and realize an algorithm by Python to re-construct the population. The second step is to compute weights. The last step is to weight response cases and analyze. Two methods were adopted in the process of computing weights. The first weighting method is to compute post-stratification weights from crosstabs. This method is used to compute two types of weights. The type 1 is weighting with respect to unit reference population. The type 2 is weighting with respect to the whole reference population. The second method is to use Logistic Regression approach to compute weights. SPSS decision tree with CHAID module has been used to compute the probabilities of predict factors for Logistic Regression. In the end, we compare the effects of the weights from these two different methods on distribution of variables.

Table of Contents

List of Tables in Appendix A

List of Figures in Appendix B

1. Introduction

Purpose/Statement of Problem: The DEOCS team has merged the unit population data into survey data. It is a dataset of 241,027cases, 310 variables as follows. Each row is for one responder. The data in red dotted box is group x gender crosstab, which is the reference population of unit.

Table 1: Original Dataset

AFEOCA ID	E1E3M	…	OtherF	…	Group	Gender	Indicator
129933							1
⋮							1
129933							1
135253							1
⋮							1
135253							1
⋮							⋮

In this dataset, there are several demographical variables, but we are interested in only two variables: “group”, which is rank of responders, and “gender”. The variable information is as follows:

Table 2: Variable-Group

Value	1	2	3	4	5	6	7	8	10
Label	E1-E3	E4-E6	E7-E9	W1-W5	O1-O3	O4-O6	Grade 1-8	Grade 9 -15 & SES	Other

Table 3: Variable-Gender

Value	**1**	**2**
Label	Male	Female

However, in above dataset there exists a lot of non-responses. The responses rate is low (< 40%). How can we make our analysis meaningful in terms of whole population based on those responses? Or how can we generalize our analysis results into whole population? Having a representative sample of the population is of paramount importance. It is not unusual that a certain demographic characteristics of the sample is distributed different from the population. This difference introduces bias into any estimate we obtain from sample data because statistical procedures will give greater weight to these oversampled people.

Design/methodology/approach: The solution to solve above “bias” is solved by post-stratification weight. However, in order to calculate a post-stratification weight, we need an auxiliary dataset, that is reference population, to which we can compare the sample data. The dataset of this project has the reference population on unit level for group and gender, which is in red-dotted box in Table 1.

The first and most import step of project is to reconstruct the whole population from unit reference population by Python in SPSS.

The second step is to calculate weights. Two methods were adopted in the process of computing weights. The first method is to compute post-stratification weights from crosstabs. This method is used to compute two types of weights. The type 1 is computing weights with respect to unit reference population. The type 2 is computing weights with respect to the whole reference population. The second method is to use Logistic Regression approach to compute weight. SPSS decision tree has been used to compute the probabilities of predict factors for Logistic Regression. The dataset of re-constructed population was used to build the decision trees. Two variables were selected as the inputs for the decision trees. Two techniques were employed to build the decision trees - Chi-square automatic interaction detection (CHAID), the exhaustive chi-squared automatic interaction detector (ECHAID). The output of the decision trees was the classification of responders and non-responders based on independent variables - group, gender, and probabilities for responders and non-responders. Next, Logistic Regression model which has these probabilities as input, and a binary variable (responders-1;

non-responders-2) as dependent variable was adopted to curve these two probabilities. Then, the weights were calculated using these probabilities. At last but optional, the weights are usually rescaled so as to add to the responding sample numbers.

In the end, we compare these weights from these different methods, weight each case and analyze.

Findings: 1. The python is the necessary tool for the data manipulation in this project.

2. Weights do bring changes to the distribution of survey dataset, but choosing the proper type of weight will depend on the original goal of survey.

3. Missing values and inconsistence values lead to errors. The missing value in population data led to no rows would be generated for non-responses for that unit. Inconsistence between unit population data and survey data will lead to extra row for non-responses.

4. A “bigger” decision tree is recommended by using more input variables.

5. The weight from second method is easier to applied, but may be less accurate.

Originality/value:

1. The complexity and size of data in SPSS decided the complexity of data manipulation. An algorithm has been designed and realized by Python to re-construct the whole population. The Python codes for this algorithm can be used as a template for future similar work.

2. Once the population data is generated or available, “decision trees” module is an efficient tool to classify responders and non-responders, compute their probability, and compute the weights for each node. The complexity of the decision trees theoretically depends on the number of input variables, instead of number of cases. If more independent variables are needed, other statistical techniques may be adopted to decrease the number of input variables and thereby reduce the complexity of the decision trees.

2. Weighting Methods

2.1 What is a Survey Weight?

When analyzing a survey, having a representative sample of the population is very important. We may accidentally (or sometimes intentionally) oversample some types of people and under-sample others. In other words, the distribution of a certain characteristic such as age, rank, race, gender, etc. of the sample may be different from their distribution in the population. Thus, **weight** is defined to be a positive value assigned to each respondent record in a survey data file in order to make the weighted records represent the population of inference as closely as possible, so that the analysis results from response data can be applied to the whole population. For example: A weight of 2 means that the case counts in the dataset as two identical cases. A weight of 1 means that the case counts in the dataset as one identical cases. A weight of 1/2 means that the case counts in the dataset as 1/2 identical cases.

The weights are usually developed in a series of stages to compensate for unequal selection probabilities, non-response, non-coverage, and sampling fluctuations from known population values [13]. In terms of progress in time of weighting process, it can be divided into three stages as follows [9].

2.2 Typical Stages of Weighting

In terms of weight process, we can divide this process into three stages. The **first stage** of weighting for unequal selection probabilities is generally straightforward. Each sampled element (whether respondent or non-respondent) is assigned a base weight that is either the inverse of the element's selection probability or proportional to that inverse. With probability sampling, the selection probabilities are known, and the base weights are generally readily determined. A difficulty that occurs with the base weights in this project arises from the lack of sampling frame and population data, which result in the probabilities of sampled elements being selected is unknown.

The **second stage** of weight development is usually to attempt to compensate for unit or total, non-response. The base weights of responding elements are adjusted to compensate for the non-responding elements. The general strategy is to identify respondents who are similar to the non-respondents in terms of auxiliary information that is available for both respondents and non-respondents, and then to increase the base weights of respondents so that they represent similar non-respondents. In many cases little is known about the non-respondents (often only their stratum and cluster), in which case a simple cell weighting adjustment may be used. In this project, the auxiliary information of non-respondents available or computed is group and gender. Respondents and non-respondents are sorted into weighting cells, and the weights of the respondents in each cell are increased by a multiplying factor so that the respondents represent the non-respondents in that cell. This method works well when there is limited auxiliary information available for the non-respondents. However, when a sizeable amount of auxiliary information is available, and the researcher wants to incorporate much of it in the non-response weighting adjustments, then other alternative methods may be needed.

The **third stage** of weight development involves a further adjustment to the weights to make the resultant weighted estimates from the sample conform to known population values for some key variables. For voluntary surveys non-response is the greatest factor to affect the accuracy of the survey estimates. Different surveys achieve different response rates, the surveys with the highest response rates tending to be those that ask questions that seem relevant and interesting to respondents. But, even with ‘popular’ surveys, response rates have been declining in recent years, and, as a direct consequence, worries about survey bias have been increasing. Non-response is only a problem if the non-respondents are a non-random sample of the total sample. Unfortunately, this seems almost always to be the case [21]. Thus, another form of adjustment is needed to force the sample joint distribution of certain variables (“group” and “gender” in this project) to match the known population joint distribution. This type of adjustment is often called **post-stratification**. It is called a post-stratification weight because it can only be computed after all data are collected. The stratification part comes from the fact that various known strata (such as age group or gender distribution) of the population are needed to adjust the sample data to conform more to the population's parameters. This stage of adjustment serves two purposes: to compensate for non-coverage and to improve the precision of the survey estimates. It can also be used to compensate for non-response. It should be noted that the theory for post-stratification presented in survey sampling texts assumes full response and perfect coverage. In this situation, the adjustments are generally relatively small provided that the sample sizes in the post-strata are reasonably large, and on average post-stratification can be expected to lead to gains in precision for the survey estimates [9]. However, when there is sizeable non-coverage and/or non-response involved, the adjustments can be substantial; in this case the adjustments are used to reduce the bias of the survey estimates, but standard errors for estimates unrelated to the adjustment variables may be increased.

2.3 Typical Types of Weighting and Computing Methods

From above three stages in weighting process, we can see that two most common types of survey weights are: Design Weights, Post-Stratification or non-response weights. **Design Weight** belongs to Stage 1 and 2, and is normally used to compensate for over- or under-sampling of specific cases or for disproportionate stratification. The **post-stratification weight** belongs to Stage 3, and is used to compensate for that fact that persons with certain characteristics are not as likely to respond to the survey. The following discuss of computing methods will be divided into three parts.

2.3.1 Computing Design Weights

It is straightforward to calculate design weights. Supposing we know the sampling fraction for each case, the weight is the inverse of the sampling fraction [5]:

$$\text{Design weight} = \frac{1}{\text{sampling fraction}}$$

The sampling fraction could also be the over-sampling amount for a given group or area. For example: If we oversampled African Americans at a rate 5 times greater than the rate for Whites, than the design weight for an African American would be $\frac{1}{5}$ and for a White respondent would be 1.

2.3.2Computing Post-Stratification Weights

However, it is normally more difficult to weight for non-response with post-stratification. This type of weight is calculated **using population data**. It requires the use of auxiliary information about the population and may take a number of different variables into account. Specifically, we need population estimates of the distribution of a set of demographic characteristics that have also been measured in the sample. This is essentially a two-step procedure[21]:

Step 1: identify a set of 'control totals' (a set of demographic characteristics) for the population that the survey ought to match;

Step 2: calculate weights to adjust the sample totals to the control totals.

If there is only one characteristic to balance with the population, then computing weight is one by following formula:

$$weight = \frac{population\ proportion}{sample\ proportion}$$

Table 4: Example for Calculating Post-Stratification Weights

Gender	Population Proportion	Sample Proportion	$\frac{Population}{Sample}$	Weight
Female	.5	.7	.5 /.7	**.714285**
Male	.5	.3	.5 /.3	**1.66667**
Total	1	1		

In analysis, only one weight per case can be used. If there were more than one characteristic to balance with the population. It is not unusual we weight for different factors, these weights must be combined together into one weight. There are following options to deal with multiple characteristics:

option 1: using one big N-way crosstab. In order to use this option, these crosstab tables available must be available from the population source. The number of cases in each cell in the sample cannot be too small. This project will adopt this option because there are only two characteristic to balance (group and gender); we are able to compute the 9X2 crosstab; the number of cases in each cell is not small.

option 2: using several separate frequency tables for each characteristics for the population. The advantage of this option is that single variable frequency tables are more likely to be available for the population; Using of frequency tables may reduce unstable weights due to small values in the sample in the cells of N-way crosstabs. The disadvantage of this method is to combine the weights for each characteristic. According to [5], there are

a. Compute a weight for each characteristic independently and then multiply all these weights together. This method is not recommended since it will usually not yield good weights.
b. Compute weights separately but sequentially. Supposing there are three characteristics A, S, E. The method is done by following iterative process:

   1. Compute A weight (wA) and weight data by this weight. Generate the weighted frequency table for S

   2. Compute S weight (wS) and weight by wA*wS. Generate the weighted frequency table for E

   3. Compute E weight (wE) and weight by wA*wS*wE. Generate the weighted frequency for A

   4. Compute a second A weight(wA2) and weight by wA*wS*wE*wA’. Generate the weighted frequency for S

5. Compute a second S weight (wS2) and weight by wA*wS*wE*wA2*wS2. Generate the weighted frequency for E

6. Compute a second E weight (wE2) and weight by wA*wS*wE*wA2*wS2*wE2

Continue process until the weighted frequencies and the population frequencies don't change. Usually converge after two or three iterations (or less).

There are also several software to conduct above iterative procedure automatically, such as SAS Raking macro Stata ado.

c. Using Logistic Regression approach to weighting. This approach requires that the dataset in use has those information for the population figures. In this project, we will adopt Logistic Regression approach with support of Decision Tree. That is why it is necessary to reconstruct the population data from the non-response data. This method is as follows [5].

1. Supposing reference population data set includes age, education, race (in categories), gender, and metropolitan status variables.
2. Assume we have the same variables measured in the same way in the dataset we want to weight to increase representativeness.
3. Create a subset of the Reference Population with just these variables and add an indicator called "Sample" set equal to 0. Also create of subset from your survey with the same variables formatted the same as the CPS data, but set the "Sample" equal to 1.
4. Combine the cases from the two data sets together.
5. Use "sample" as a dependent variable in a logistic regression with each of the other characteristics as independent variables. Set the regression program to save the predicted probability (pprob) from the regression for each case and include it in the dataset.
6. The weight would be the inverse of this predicted probability: $weight = \frac{1}{pprob}$.

7. Yields weights that are highly correlated with those obtained in raking.

The main constraint on using post-stratification is that the population distributions must be known. This automatically limits that the only control totals that can be used are the ones available and known to be accurate. For most DEOCS surveys, control totals tend to be rank, service, gender and enlisted.

We divide non-response weights and design weights into different types, but they look similar, at least in terms of mathematics. A possible difference is that design weights are known exactly but non-response weights are only estimated.

For design weights we know how many units were selected and how many were in the sampling frame. Non-response weights are estimated by comparing responding units to totals from the population or from the sampling frame. If we repeated the sampling procedure many times we would get different numbers of non-responding units in each post-strata. This would give different non-response weights in each possible sample. This uncertainty in the exact value of non-response weights should be reflected in the standard errors of the non-response adjusted analyses. Only replication methods such as jackknives and bootstrap methods include this adjustment. If the post-stratification is based on a simple model the contribution to the standard error from the uncertainty in the weights will be very small.

2.3.3 Computing Weights using Survey Information from Sampling Frame

Sometimes non-response re-weighting can be carried out by comparing the characteristics of those who responded to a survey with the whole group who the survey attempted to reach. This will not be very helpful when (as in most household surveys) we don’t know much about the people who do not respond. This approach is most helpful when we are selecting a sample form an informative sampling frame. Examples might be surveys of a workforce, where we know the grade, age, length of service of all employees. Another circumstance when this is used is in the context of longitudinal surveys when a survey is re-contacting people who responded at a previous wave of the survey. This is done in three steps. This is done in three steps [21].

1. Carry out an investigation of which factors predict that a response has been received.

2. Apply a weight to the **responding classes** that is proportional to $\frac{1}{\text{probability of responding}}$.

3. Finally, the weights, at this stage generally above 1.0, are usually rescaled so as to add to the responding sample numbers.

At the first step a response variable is attached to the sampling frame that is coded as 1 for responders and 0 for non-responders. Where the sampling frame only tells us a few things about the units we can divide the sample up into groups (called response classes) and the probability of response is simply the proportion who respond in each response class.

When the sampling frame contains more detailed information about the non-responders the factors that influence non-response are often investigated via logistic regression. The resulting model is then used to calculate the probability of response at Step 1 above, and the subsequent steps are carried out in the same way as above.

When this type of regression model is used we need to strike a balance between having a powerful model to predict non-response (and so reduce bias) and the introduction of extreme weights that will affect precision. Models are often simplified at the final stage to avoid extreme weights (either large or small). Another practice that some surveys employ is to cap the weights, for example by replacing all weights above 2.5 with the value of 2.5.

2.3.4 Combine Different Types of Weights

As what has been discussed above, it is usual to compute different types of weights for same dataset, since we need to weight for different factors. However, only one weight will be allowed to analyze the dataset, so these weights must be combined together into one weight before use. Suppose that a design weight (Dwate) and a post-stratification (PSwate) weight have been computed for each case. Then a total weight will be multiplication of these two weights:

$$\text{Total Weight} = \text{Dwate} \times \text{PSwate}$$

Furthermore, a weight cannot be equal to zero unless we want the case excluded from the analysis. The default value is set to 1.

3. Decision Tree

Decision tree models enable to develop classification systems that predict or classify future observations based on a set of decision rules. If we have data divided into classes that interest us, we can use the data to build rules that we can use to classify old or new cases with maximum accuracy. For example, we might build a tree that classifies credit risk or purchase intent based on age and other factors [2,29].

Tree building Algorithm [10]. Four algorithms are available for performing classification and segmentation analysis. These algorithms all perform basically the same thing: they examine all of the fields of your dataset to find the one that gives the best classification or prediction by splitting the data into subgroups. The process is applied recursively, splitting subgroups into smaller and smaller units until the tree is finished (as defined by certain stopping criteria). The target and input fields used in tree building can be continuous (numeric range) or categorical, depending on the algorithm used. If a continuous target is used, a regression tree is generated; if a categorical target is used, a classification tree is generated.

Table 5: Decision Tree Modules

Feature\Algorithm	**C&R Tree**	**QUEST**	**CHAID**	**C5.0**
Input fields(predictors)	continuous, categorical, flag, nominal or ordinal	continuous, categorical, flag, nominal or ordinal	continuous, categorical, flag, nominal or ordinal	continuous, categorical, flag, nominal or ordinal

Target fields	continuous, categorical, flag, nominal or ordinal	categorical, flag or nominal	continuous, categorical, flag, nominal or ordinal	flag, nominal or ordinal
Type of split	binary splits	binary splits	more than two branches at a time	more than two branches at a time
Method used for splitting	For categorical output, a dispersion measure is used (by default the Gini coefficient). For continuous targets, the least squared deviation method is used	chi-square test for categorical predictors, and analysis of variance for continuous inputs	chi-square test	information theory measure is used, the information gain ratio
Missing value handling	use substitute prediction fields, where needed, to advance a record with missing values through the tree during training	use substitute prediction fields, where needed, to advance a record with missing values through the tree during training	makes the missing values a separate category and enables them to be used in tree building.	uses a fractioning method, which passes a fractional part of a record down each branch of the tree from a node where the split is based on a field with a missing value.
Pruning	offer the option to grow the tree fully and then prune it back by removing bottom-level splits that do not contribute significantly to the accuracy of the tree	offer the option to grow the tree fully and then prune it back by removing bottom-level splits that do not contribute significantly to the accuracy of the tree		offer the option to grow the tree fully and then prune it back by removing bottom-level splits that do not contribute significantly to the accuracy of the tree
Interactive tree building	provide an option to launch an interactive session.	provide an option to launch an interactive session.	provide an option to launch an interactive session.	No this option
Prior probabilities	support the specification of prior probabilities for categories when predicting a categorical target field.	support the specification of prior probabilities for categories when predicting a categorical target field.	do not support specifying prior probabilities	do not support specifying prior probabilities
Rule sets				

In this project, we will use re-constructed population to run decision tree module with CHAID. “Group” and “gender” are independent variables, and “indicator” is dependent variable. The output of decision tree are probabilities of responder and non-responder in each node of tree.

Then, we use “indicator” as a dependent variable in a logistic regression with probability for responses, which is output from decision tree, as independent variables. We set the regression program to save the predicted probability (pprob) from the regression for each node and include it in the dataset.

Next, compute: weight=$\frac{1}{\text{predicted probability}}$.

The last step is optional. Re-scale the weights, which at this stage generally are above 1.0, are usually rescaled so as to add to the responding sample numbers.

4. Python

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems[17, 25].

The IBM SPSS Statistics - Integration Plug-in for Python provides two interfaces for programming with the Python language within IBM SPSS Statistics on Windows, Linux, Mac OS, and for IBM SPSS Statistics Server[11].

Python Integration Package: The Python Integration Package provides functions that operate on the IBM SPSS Statistics processor, extending IBM SPSS Statistics command syntax with the full capabilities of the Python programming language. With this interface, we can access IBM SPSS Statistics variable dictionary information, case data, and procedure output. We can submit command syntax to IBM SPSS Statistics for processing, create new variables and new cases in the active dataset, or create new datasets. We can also create output in the form of pivot tables and text blocks, all from within Python code.

Scripting Facility: The Scripting Facility provides Python functions that operate on user interface and output objects. With this interface, you can customize pivot tables, and export items such as charts and tables in various formats. We can also start IBM SPSS Statistics dialog boxes, and manage connections to instances of IBM SPSS Statistics Server, all from within Python code.

In this project, the only available software is SPSS. Thus, Integration Plug-in for Python is the tool for us to manipulate data.

5. Design and Analysis of Algorithm

In sections, we have analyzed the features of dataset of this project: large scale, low responses rate, lack of reference population. Thus, we need to weight for non-responses. In order for weighting, we need re-construct reference population. The design of the project is as follows:

First step: design and realize an algorithm by Python to re-construct the population.

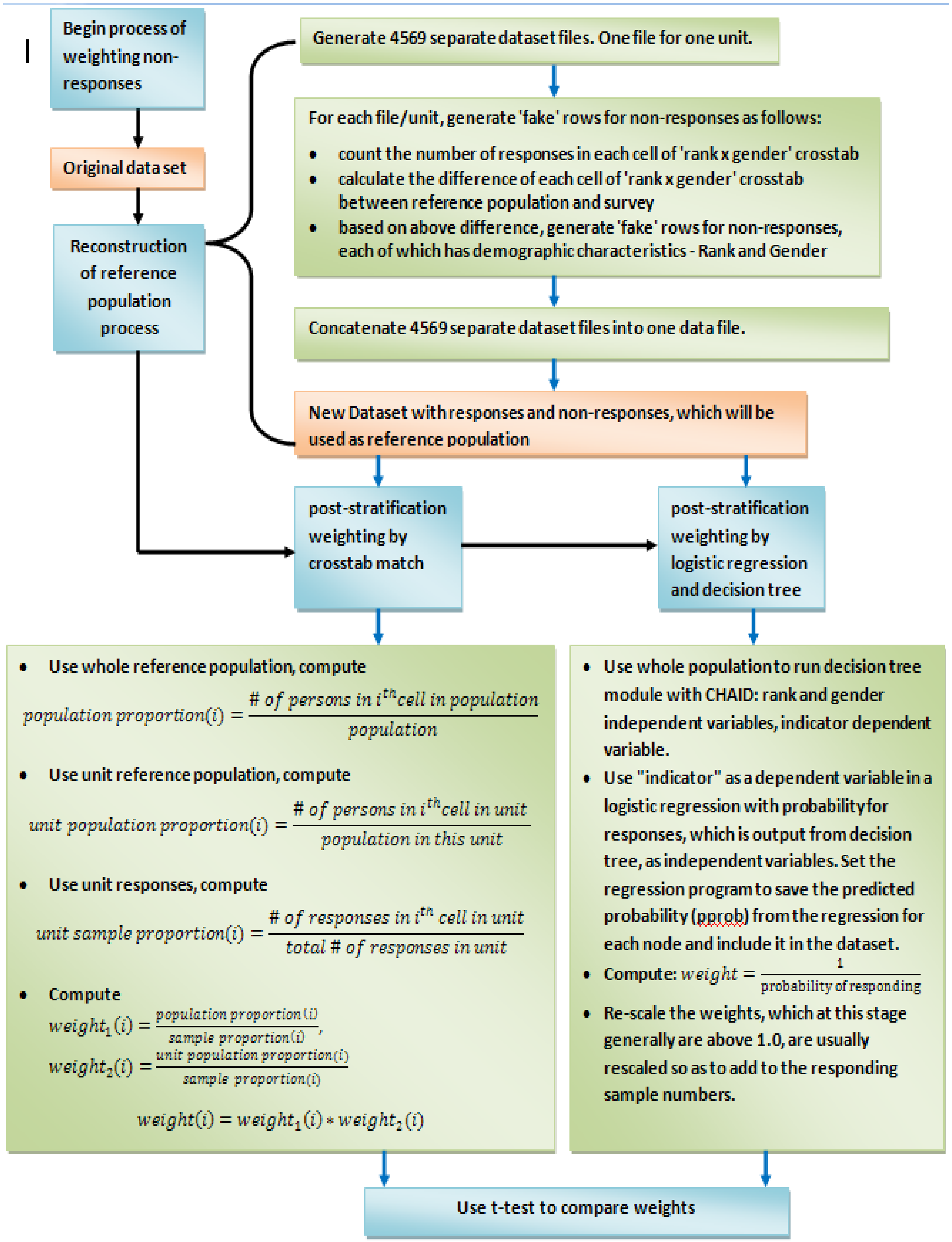
Second step: Two methods were adopted to weight non-responses:

- The first method is to compute Post-Stratification weights by matching crosstabs.
- The second is to use Logistic Regression approach to weight. SPSS decision tree with CHAID module has been used to compute the probabilities of predict factors.

Last step: we compare weights from different methods, and further suggestion and discussion will be conducted.

The flow and Components of Project is as follows:

Figure 1: Flow and Components of Project



The algorithm for re-construction can be broken into five components as follows:

Table 6: Components for Re-construction and Computing Weights

<table>
<tr><th colspan="3">Components</th><th>Operation Time</th></tr>
<tr><td rowspan="5">Component 1</td><td>Purpose</td><td>Generate unit dataset files.
Compute and add all non-responder cases for each unit dataset file.</td><td rowspan="5">20-24 hours</td></tr>
<tr><td>Input</td><td>The original dataset file with new variable “indicator”.</td></tr>
<tr><td>Procedure</td><td>Iterate on units:
1. generate one unit dataset file
2. compute the missing responses for each cell of “group and gender” crosstab, generate non-responses rows, and attach all non-responder cases to this unit dataset file
3. output this unit dataset file, and then close it.</td></tr>
<tr><td>Output</td><td>Separate 4500+ dataset files. One file for one unit.</td></tr>
<tr><td>Comments</td><td>1. I have added a new variable “indicator” to original dataset. The name of new dataset file is “AlldatawithIndicator.sav”.
2. After opening original data file, it is necessary to compute another new variable $CASENUM, and set it as last variable.
3. Indicator’s values are 1, which means all cases are responders. Its value for non-response is 0.
4. If reference population is missing, then no rows for non-response will be added.
5. Error 1: In the crosstab, the value of each cell of reference population is supposed to be greater or equal to the corresponding value of responses. However, some responses may input wrong information, then this algorithm will generate extra rows. For example, this algorithm will generate one extra non-response row for following unit:
<table>
<tr><th></th><th colspan="2">Population
Number of rows
Total: 10</th><th></th><th colspan="2">Sample
Number of rows
Total: 10</th></tr>
<tr><td></td><td colspan="2">gender</td><td></td><td colspan="2">gender</td></tr>
<tr><td>Rank 1</td><td>4</td><td>3</td><td></td><td>5</td><td>2</td></tr>
<tr><td>Rank 2</td><td>1</td><td>2</td><td></td><td>1</td><td>2</td></tr>
</table>
6. Error 2: missing value from “group” or “gender”. Someone submits his survey, but he does not fill out gender or rank. For example, in following survey there are 10 responses, but one person does not input his rank, so by computing, it misses one person, then algorithm generates an extra row.
<table>
<tr><th></th><th colspan="2">Population
Total Number of rows: 10</th><th></th><th colspan="2">Sample
Total Number of rows: 10</th></tr>
<tr><td></td><td colspan="2">gender</td><td></td><td colspan="2">gender</td></tr>
<tr><td>Rank 1</td><td>4</td><td>3</td><td></td><td>4</td><td>2</td></tr>
<tr><td>Rank 2</td><td>1</td><td>2</td><td></td><td>1</td><td>2</td></tr>
</table>
Above two types of error will lead to extra cases, which will affect following analysis including decision tree and computing weights.</td></tr>
</table>

Component 2	**Purpose**	**automatically concatenate files**	**1 hour**
	Input	**The unit dataset files (4500+)**	
	Procedure	**Iterate on files:** **1. count 50 files** **2. use "ADD FILES" to concatenate every 50 files** **concatenate the leftover files**	
	Output	**One dataset file "allcombined.sav", which includes all responses and non-responses.**	
	Comments	**The maximum number of files that SPSS –"ADD FILES" command can take is 50, so we have to set up loop to concatenate 50 files at one time.**	
Component 3	**Purpose**	**Generate decision tree, and then run logistic regress to get predict probability**	**Less than 10 minutes**
	Input	**The input of decision tree is one dataset file "allcombined.sav" from Component 2.** **The input of regression is the probability of responses from Decision Tree**	
	Procedure	**Run SPSS-Tree first, and run SPSS - LOGISTIC REGRESSION**	
	Output	**Decision tree: "Output_07012017_1_treeoutput.spv"** **Predicted probability for responders:"NodeIDbyProb_07032017.sav", "allcombined_07012017_Tree1.sav"**	
	Comments	**The input of Decision Tree module is the whole population dataset file. Output is: decision tree, and a new variable "predicted probability".**	
Component 4-1	**Purpose**	**Compute weights by using unit reference population**	**About 25 minutes**
	Input	**One dataset file: "alldatawithoutindicator.sav". This is in fact the original dataset.**	
	Procedure	**Iterate on units:** **1. Count the number of responses in each cell of group x gender crosstab.** **2. Compute: respondersratio =** $\frac{Number\ of\ responses\ in\ each\ cell}{Total\ responses\ in\ this\ unit}$ **3. Compute: referenceratio** $=\frac{Number\ in\ each\ cell\ of\ unit\ reference\ population}{Population\ of\ this\ unit}$ **4. Compute: weight=** $\frac{referenceratio}{respondersratio}$	
	Output	**One dataset file "weights_%(name)s.sav".**	
	Comments	**1. Please make sure that the variable $casenum is the last variable since I will use it to generate the dictionary for all units.** **2. The output of this syntax is a dataset file with "UnitNumber, UnitID, and 18 weights for those 18 cells in crosstab"** **3. In fact this component can be imbedded in Component 1, but because of memory problem, I have to separate.**	
Compo	**Purpose**	**Compute weights by using whole reference population**	**About 25 minutes**
	Input	**One dataset file: "alldatawithoutindicator.sav". This is in fact the original dataset.**	

	Procedure	iterate on units: 1. Count the number of responses in each cell of group x gender crosstab. 2. Compute: respondersratio $=\frac{\textit{Number of responses in each cell}}{\textit{Total responses in this unit}}$ 3. Compute: rratioij$=\frac{\textit{Number in each cell of whole reference population}}{\textit{The whole population}}$ Compute: weight$=\frac{\text{rratioij}}{\text{respondersratio}}$	
	Output	One dataset file "Myweights0703201701_%(name)s.sav"	
	Comments	1. Please make sure that the variable $casenum is the last variable since I will use it to generate the dictionary for all units. 2. The output of this syntax is a dataset file with "UnitNumber, UnitID, and 18 weights for those 18 cells in crosstab" 3. In fact this component can be imbedded in Component 1, but because of memory problem, I have to separate. 4. The variables "rratio11", etc. are ratio = population proportion of cell 11=(total number of "rank 1 gender 1")/whole population. However, I do not need to compute this ration sinceI just read it from the outcome of decision tree.	
Component 5-1	Purpose	Attach the unit-based weights to each case in the original dataset file	16 hours
	Input	Original dataset file: "originaldatawithweightsr.sav". This is in fact the original dataset file. "weights_xDataset1_unitpop.sav". It is the weights based on unit reference population. It is the output from Component 4-1.	
	Procedure	Iterate on cases: 1. Compute rank and gender value 2. Compute a new variable "localweights"	
	Output	One dataset file: originaldatawithlocalweights.sav	
	Comments	The component can be imbedded in Components 5-1, 5-2, 5-3, but because of memory, I separate it.	
Component 5-2	Purpose	Attach the global weights into each case.	13 hours
	Input	"originaldatawithlocalweights.sav". This is the output dataset file from Component 5-1 "Myweights0703201701_xDataset1_wholepop.sav". This is the output dataset file from Component 4-2	
	Procedure	Iterate on cases: 1. Compute rank and gender value 2. Compute a new variable "globalweights"	
	Output	One dataset file: originaldatawithlocalandglobalweights.sav	
	Comments	The component can be imbedded in Components 5-1, 5-2, 5-3, but because of memory, I separate it.	
Compone	Purpose	Attach the decision tree weights into each case.	15 hours
	Input	"originaldatawithlocalandglobalweights.sav". This is in fact the output dataset file from Component 5-2	

		"NodeIDbyProb_07032017.sav". This is the output dataset file from Component 3. It is the weights from decision tree.	
	Procedure	**Iterate on cases:** **1. Compute rank and gender value** **2. Compute a new variable "decisiontreeweights"**	
		One dataset file: originaldatawithlocalandglobalanddecisiontreeweights.sav	
	Comments	**The component can be imbedded in Component 1, but because of memory, I separate it.**	

The Python codes for above algorithms of each component are at appendix. The challenging and time-consuming component is Component 1. The design of algorithm is as follow:

Step 1: Read in the whole dataset and compute a new variable casenum:

```
GET FILE='E:\WEI WAN\My SPSS\TestFiles\AlldatawithIndicator.sav'.
DATASET NAME alldata.
SORT CASES BY AFEOCAID.
compute casen = $CASENUM.
formats casen(f12.0).
VARIABLE LEVEL casen (SCALE).
EXECUTE.
```

The outcome is the following dataset for all responses in memory.

AFEOCA ID	E1E3M	…	OtherF	…	Group	Gender	Indicator	casenum
129933								1
⋮								⋮
129933								62
135253								63
⋮								⋮
135253								78
⋮								⋮

Step 2: Generate a list of variable names

```
ordlist=[]
ordlist2=[]
for i in range(spss.GetVariableCount()):
        ordlist.append(spss.GetVariableName(i))
        ordlist2.append(spss.GetVariableType(i))

totalvar = spss.GetVariableCount()
```

The outcome will be two lists. One is the list of all variable names, the other is the list of all variable type.

```
[u'PopGrp', u'AFEOCAID', u'DEOCSID', u'DEOCSID7', u'UICCODE', u'MainUicCode', u'titleoforganization', u'commanderemail', u'commandername', u'star
tdate', u'enddate', u'reportdate', u'submitdate', u'E1E3M', u'E1E3F', u'E4E6M', u'E4E6F', u'E7E9M', u'E7E9F',
u'WOM', u'WOF', u'O1O3M', u'O1O3F', u'O4AboveM', u'O4AboveF', u'GS1GS8M', u'GS1GS8F', u'GS9_SESM', u'GS9_SESF', u'OtherM', u'OtherF', u'SERVICE',
 u'COMPONENT', u'accessCodeID', u'gender', u'hispanic', u'race1', u'race2', u'race3', u'race4', u'race5', u'Ra
ce', u'Maj_Min', u'reside', u'deployed', u'fedcat', u'paygrad', u'branch', u'type', u'Q1', u'Q2', u'Q3', u'Q4', u'Q5', u'Q6', u'Q7', u'Q8', u'Q9'
, u'Q10', u'Q11', u'Q12', u'Q13', u'Q14', u'Q15', u'Q16', u'Q17', u'Q18', u'Q19', u'Q20', u'Q21', u'Q22', u'Q2
3', u'Q24', u'Q25', u'Q26', u'Q27', u'Q28', u'Q29', u'Q30', u'Q31', u'Q32', u'Q33', u'Q34', u'Q35', u'Q36', u'Q37', u'Q38', u'Q39', u'Q39A', u'Q4
0', u'Q41', u'Q42', u'Q43', u'Q44', u'Q45', u'Q46', u'Q47', u'Q48', u'Q49', u'Q50', u'Q51', u'Q52', u'Q53', u'
Q54', u'Q55', u'Q56', u'Q57', u'Q58', u'Q59', u'Q60', u'Q61', u'Q62', u'Q63', u'Q64', u'Q65', u'Q66', u'Q67', u'Q68', u'Q69', u'Q70', u'Q71', u'Q
72A', u'Q72B', u'Q72C', u'Q72D', u'Q72E', u'Q72F', u'Q72G', u'Q72H', u'Q72I', u'Q72J', u'Q73A', u'Q73B', u'Q73
C', u'Q73D', u'Q73E', u'Q73F', u'Q73G', u'Q73H', u'Q73I', u'Q73J', u'Q73K', u'Q74', u'Q75', u'Q76', u'Q77A', u'Q77B', u'Q77C', u'Q77D', u'Q77E',
u'Q78', u'Q78A', u'Q79', u'Q80', u'Q81', u'Q82', u'Q83', u'Q84', u'Q85', u'IsPaper', u'LogID', u'Input_date',
u'Q74_1', u'Q74_2', u'Q74_3', u'Q74_4', u'Q74_5', u'Q74_6', u'Q74_7', u'Q74_8', u'Q74_9', u'Q74_10', u'Q79_1', u'Q79_2', u'Q79_3', u'Q79_4', u'Q7
9_5', u'Q79_6', u'Q79_7', u'Q79_8', u'Q79_9', u'Q79_10', u'Q79_11', u'Q81_1', u'Q81_2', u'Q81_3', u'Q81_4', u'
Q81_5', u'Q81_6', u'Q81_7', u'Q84_1', u'Q84_2', u'Q84_3', u'Q84_4', u'Q84_5', u'Q84_6', u'Q84_7', u'Month', u'Quarter', u'DoD', u'Group', u'Rank'
, u'RankJr', u'RankJrMil', u'OFFvsENL', u'MILvsCIV', u'Organization', u'ServComponent', u'BroadJrEnlGender', u
'JrEnlGender', u'TypeTotal', u'ActiveMilCiv', u'ArmyVsAir', u'INTENDSTAY', u'FAVORITISM', u'RQ19', u'RQ25', u'RQ32', u'RQ36', u'RQ40', u'RQ44', u
'RQ47', u'RQ56', u'RQ62', u'RQ65', u'OrgCom', u'TrustLead', u'OrgPerf', u'OrgCoh', u'LeadCoh', u'JobSat', u'Di
vMgt', u'OrgProc', u'HelpSeek', u'Exhaust', u'Hazing', u'Demean', u'Raci[illegible]exDisc', u'RelDisc', u'SexHar', u'Racist', u'Sexist', u'AgeDisc'
, u'DisDisc', u'Safety1', u'Safety2', u'CoCSupport1', u'CoCSupport2', u[illegible]3', u'CoCSupport4', u'CoCSup
port5', u'Publicity1', u'Publicity2', u'Publicity3', u'CoCSupport6', u'CoCSupport7', u'URC1', u'URC2', u'URC3', u'URC4', u'URC5', u'URC6', u'URC7
', u'URC8', u'URC9', u'URC10', u'SafetyPercep', u'SafeLiveUF', u'SafeWorkUF', u'CoCSupport', u'Publicity', u'P
ublicityUF1', u'PublicityUF2', u'PublicityUF3', u'URC7rev', u'URC9rev', u'URC10rev', u'URC', u'BarriersTotal', u'BarriersTri', u'Bystander1', u'B
ystander1UF', u'Bystander2', u'BystanderScale', u'BystanderObs1', u'BystanderObs2', u'Knowledge1', u'Knowledge
2', u'Knowledge3', u'Knowledge4', u'Knowledge5', u'KnowledgeScale', u'JrEnlvNCOvAll', u'JrEnandNCO', u'BystanderObs2Dich', u'Metric4Composite', u
'Metric9CompositeR', u'Metric9NEWComposite', u'Metric11Composite', u'SenorityGender', u'PasswordsRequested', u
'FirstDEOCS', u'Occurrence', u'ACTSERVICE', u'WRank', u'PopNum', u'filter_$', u'TotalN', u'indicator', u'casen']
[0, 0, 9, 0, 25, 25, 27, 40, 19, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 25, 0, 0, 25, 0, 0, 25, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Step 3: Change from Python unicode to string. (Strings received by Python from IBM SPSS Statistics are converted from UTF-8 to Python Unicode, which is UTF-16.)

```
length = len(ordlist)
newordlist=[]
for i in range(length):
    newordlist.append([ordlist[i].encode('ascii','ignore'),ordlist2[i]])
```

The outcome will be one list as follows. Each element of this list is a pair: (VariableName, VariableType).

```
[['PopGrp', 0], ['AFEOCAID', 0], ['DEOCSID', 9], ['DEOCSID7', 0], ['UICCODE', 25], ['MainUicCode', 25], ['titleoforganization', 27], ['commandere
mail', 40], ['commandername', 19], ['startdate', 0], ['enddate', 0], ['reportdate', 0], ['submitdate', 0], ['E
1E3M', 0], ['E1E3F', 0], ['E4E6M', 0], ['E4E6F', 0], ['E7E9M', 0], ['E7E9F', 0], ['WOM', 0], ['WOF', 0], ['O1O3M', 0], ['O1O3F', 0], ['O4AboveM',
 0], ['O4AboveF', 0], ['GS1GS8M', 0], ['GS1GS8F', 0], ['GS9_SESM', 0], ['GS9_SESF', 0], ['OtherM', 0], ['Other
F', 0], ['SERVICE', 0], ['COMPONENT', 0], ['accessCodeID', 0], ['gender', 0], ['hispanic', 0], ['race1', 0], ['race2', 0], ['race3', 0], ['race4'
, 0], ['race5', 0], ['Race', 0], ['Maj_Min', 0], ['reside', 0], ['deployed', 0], ['fedcat', 0], ['paygrad', 0]
, ['branch', 0], ['type', 0], ['Q1', 0], ['Q2', 0], ['Q3', 0], ['Q4', 0], ['Q5', 0], ['Q6', 0], ['Q7', 0], ['Q8', 0], ['Q9', 0], ['Q10', 0], ['Q1
1', 0], ['Q12', 0], ['Q13', 0], ['Q14', 0], ['Q15', 0], ['Q16', 0], ['Q17', 0], ['Q18', 0], ['Q19', 0], ['Q20'
, 0], ['Q21', 0], ['Q22', 0], ['Q23', 0], ['Q24', 0], ['Q25', 0], ['Q26', 0], ['Q27', 0], ['Q28', 0], ['Q29', 0], ['Q30', 0], ['Q31', 0], ['Q32',
 0], ['Q33', 0], ['Q34', 0], ['Q35', 0], ['Q36', 0], ['Q37', 0], ['Q38', 0], ['Q39', 0], ['Q39A', 0], ['Q40',
0], ['Q41', 0], ['Q42', 0], ['Q43', 0], ['Q44', 0], ['Q45', 0], ['Q46', 0], ['Q47', 0], ['Q48', 0], ['Q49', 0], ['Q50', 0], ['Q51', 0], ['Q52', 0
], ['Q53', 0], ['Q54', 0], ['Q55', 0], ['Q56', 0], ['Q57', 0], ['Q58', 0], ['Q59', 0], ['Q60', 0], ['Q61', 0],
 ['Q62', 0], ['Q63', 0], ['Q64', 0], ['Q65', 0], ['Q66', 0], ['Q67', 0], ['Q68', 0], ['Q69', 0], ['Q70', 0], ['Q71', 0], ['Q72A', 0], ['Q72B', 0]
, ['Q72C', 0], ['Q72D', 0], ['Q72E', 0], ['Q72F', 0], ['Q72G', 0], ['Q72H', 0], ['Q72I', 0], ['Q72J', 0], ['Q7
3A', 0], ['Q73B', 0], ['Q73C', 0], ['Q73D', 0], ['Q73E', 0], ['Q73F', 0], ['Q73G', 0], ['Q73H', 0], ['Q73I', 0], ['Q73J', 0], ['Q73K', 0], ['Q74'
, 25], ['Q75', 0], ['Q76', 0], ['Q77A', 0], ['Q77B', 0], ['Q77C', 0], ['Q77D', 0], ['Q77E', 0], ['Q78', 0], ['
Q78A', 0], ['Q79', 25], ['Q80', 0], ['Q81', 25], ['Q82', 0], ['Q83', 0], ['Q84', 25], ['Q85', 0], ['IsPaper', 0], ['LogID', 25], ['Input_date', 0
], ['Q74_1', 0], ['Q74_2', 0], ['Q74_3', 0], ['Q74_4', 0], ['Q74_5', 0], ['Q74_6', 0], ['Q74_7', 0], ['Q74_8',
 0], ['Q74_9', 0], ['Q74_10', 0], ['Q79_1', 0], ['Q79_2', 0], ['Q79_3', 0], ['Q79_4', 0], ['Q79_5', 0], ['Q79_6', 0], ['Q79_7', 0], ['Q79_8', 0],
 ['Q79_9', 0], ['Q79_10', 0], ['Q79_11', 0], ['Q81_1', 0], ['Q81_2', 0], ['Q81_3', 0], ['Q81_4', 0], ['Q81_5',
 0], ['Q81_6', 0], ['Q81_7', 0], ['Q84_1', 0], ['Q84_2', 0], ['Q84_3', 0], ['Q84_4', 0], ['Q84_5', 0], ['Q84_6', 0], ['Q84_7', 0], ['Month', 0],
['Quarter', 0], ['DoD', 0], ['Group', 0], ['Rank', 0], ['RankJr', 0], ['RankJrMil', 0], ['OFFvsENL', 0], ['MIL
vsCIV', 0], ['Organization', 0], ['ServComponent', 0], ['BroadJrEnlGender', 0], ['JrEnlGender', 0], ['TypeTotal', 0], ['ActiveMilCiv', 0], ['Army
VsAir', 0], ['INTENDSTAY', 0], ['FAVORITISM', 0], ['RQ19', 0], ['RQ25', 0], ['RQ32', 0], ['RQ36', 0], ['RQ40',
 0], ['RQ44', 0], ['RQ47', 0], ['RQ56', 0], ['RQ62', 0], ['RQ65', 0], ['OrgCom', 0], ['TrustLead', 0], ['OrgPerf', 0], ['OrgCoh', 0], ['LeadCoh',
 0], ['JobSat', 0], ['DivMgt', 0], ['OrgProc', 0], ['HelpSeek', 0], ['Exhaust', 0], ['Hazing', 0], ['Demean',
0], ['RacDisc', 0], ['SexDisc', 0], ['RelDisc', 0], ['SexHar', 0], ['Racist', 0], ['Sexist', 0], ['AgeDisc', 0], ['DisDisc', 0], ['Safety1', 0],
['Safety2', 0], ['CoCSupport1', 0], ['CoCSupport2', 0], ['CoCSupport3', 0], ['CoCSupport4', 0], ['CoCSupport5'
, 0], ['Publicity1', 0], ['Publicity2', 0], ['Publicity3', 0], ['CoCSupport6', 0], ['CoCSupport7', 0], ['URC1', 0], ['URC2', 0], ['URC3', 0], ['U
RC4', 0], ['URC5', 0], ['URC6', 0], ['URC7', 0], ['URC8', 0], ['URC9', 0], ['URC10', 0], ['SafetyPercep', 0],
['SafeLiveUF', 0], ['SafeWorkUF', 0], ['CoCSupport', 0], ['Publicity', 0], ['PublicityUF1', 0], ['PublicityUF2', 0], ['PublicityUF3', 0], ['URC7r
ev', 0], ['URC9rev', 0], ['URC10rev', 0], ['URC', 0], ['BarriersTotal', 0], ['BarriersTri', 0], ['Bystander1',
 0], ['Bystander1UF', 0], ['Bystander2', 0], ['BystanderScale', 0], ['BystanderObs1', 0], ['BystanderObs2', 0], ['Knowledge1', 0][illegible]dge2',
 0], ['Knowledge3', 0], ['Knowledge4', 0], ['Knowledge5', 0], ['KnowledgeScale', 0], ['JrEnlvNCOvAll', 0], ['J
rEnandNCO', 0], ['BystanderObs2Dich', 0], ['Metric4Composite', 0], ['Metric9CompositeR', 0], ['Metric9NEWComposite', 0], ['Metric11Composite', 0]
, ['SenorityGender', 0], ['PasswordsRequested', 0], ['FirstDEOCS', 0], ['Occurrence', 0], ['ACTSERVICE', 0], [
'WRank', 0], ['PopNum', 0], ['filter_$', 0], ['TotalN', 0], ['indicator', 0], ['casen', 0]]
```


Step 4: Generate a dictionary {unit number: [unitID, begincasenum, endcasenum]}.

with spss.DataStep() starts a block to manipulate data.

```
with spss.DataStep():
    ds = spss.Dataset() #---create a pointer to point at current active dataset---
    i =1
    d1=dict()
    #-i: unit number--j = [deptid, beginingcasenum, endingcasenum]---------------
    j=[ds.cases[0,1][0], ds.cases[0,totalvar-1][0],ds.cases[0,totalvar-1][0]]
    d1={i:j}
    iteration = 0
    for r in ds.cases:
        if (r[1]!=d1[i][0]):
            d1[i][2]=d1[i][1]+ iteration-1
            i=i+1
            newj=[r[1], d1[i-1][2]+1, d1[i-1][2]+1]
            d1[i]=newj
            iteration=0
        iteration=iteration+1
    d1[i][2]=ds.cases[-1,totalvar-1][0]
```

Outcome will be as follows:

```
{1: [129933.0, 1.0, 62.0], 2: [135253.0, 63.0, 78.0], 3: [161338.0, 79.0, 99.0], 4:
[161339.0, 100.0, 174.0], 5: [161730.0, 175.0, 195.0], 6: [162772.0, 196.0, 215.0],
7: [165204.0, 216.0, 260.0], 8: [166080.0, 261.0, 284.0], 9: [166370.0, 285.0, 304.0
],
10: [166757.0, 305.0, 418.0], 11: [167247.0, 419.0, 435.0], 12: [167560.0, 436.0, 46
2.0], 13: [167713.0, 463.0, 553.0], 14: [167716.0, 554.0, 604.0], 15: [167988.0, 605
.0, 624.0], 16: [168840.0, 625.0, 643.0], 17: [168841.0, 644.0, 659.0], 18: [168872.
0,
660.0, 685.0], 19: [169151.0, 686.0, 748.0], 20: [169154.0, 749.0, 779.0], 21: [1691
55.0, 780.0, 794.0], 22: [169834.0, 795.0, 810.0], 23: [170115.0, 811.0, 845.0], 24:
 [170175.0, 846.0, 863.0], 25: [171053.0, 864.0, 904.0], 26: [171092.0, 905.0, 918.0
],
27: [171345.0, 919.0, 950.0], 28: [171583.0, 951.0, 970.0], 29: [171908.0, 971.0, 98
8.0], 30: [171940.0, 989.0, 1010.0], 31: [172149.0, 1011.0, 1034.0], 32: [172887.0,
1035.0, 1118.0], 33: [172920.0, 1119.0, 1167.0], 34: [172930.0, 1168.0, 1183.0], 35:
```


Step 5: Iterate on units: generate a dataset for an unit, compute each cell in group X gender crosstab, and compute the difference between responses and reference population, then generate non-responses rows, and at last output this file.

```
for key in d1:
    with spss.DataStep():
        ds1 = spss.Dataset(name="alldata")
# Create a new dataset for each unit--------------------------------------------------
        newds1 = spss.Dataset(name=None)
# Add variables to this new dataset------------------------------------------------
        for i in range(length):
            newds1.varlist.append(newordlist[i][0],newordlist[i][1])

        deptid = d1[key][0]
        dsNames = {newds1.name : deptid}
        begincasenum=int(d1[key][1])-1
        endcasenum=int(d1[key][2])
#working on here--------------------.
        templist=[0]*18
        v11,v12,v21,v22,v31,v32,v41,v42,v51,v52,v61,v62,v71,v72,v81,v82,v101,v102=templist
        for row in ds1.cases[begincasenum : endcasenum]:
# add this case to the new file ------------------------------------------------------------
            newds1.cases.append(row)
# count the number of each cell in Group and gender crosstab--------------------
            if row[ds1.varlist['Group'].index] == 1 and row[ds1.varlist['gender'].index]==1:
                v11 = v11+1
            if row[ds1.varlist['Group'].index] == 1 and row[ds1.varlist['gender'].index]==2:
                v12 = v12+1
            if row[ds1.varlist['Group'].index] == 2 and row[ds1.varlist['gender'].index]==1:
                v21 = v21+1
            if row[ds1.varlist['Group'].index] == 2 and row[ds1.varlist['gender'].index]==2:
                v22 = v22+1
            if row[ds1.varlist['Group'].index] == 3 and row[ds1.varlist['gender'].index]==1:
                v31 = v31+1
            if row[ds1.varlist['Group'].index] == 3 and row[ds1.varlist['gender'].index]==2:
                v32 = v32+1
            if row[ds1.varlist['Group'].index] == 4 and row[ds1.varlist['gender'].index]==1:
                v41 = v41+1
            if row[ds1.varlist['Group'].index] == 4 and row[ds1.varlist['gender'].index]==2:
                v42 = v42+1
            if row[ds1.varlist['Group'].index] == 5 and row[ds1.varlist['gender'].index]==1:
                v51 = v51+1
            if row[ds1.varlist['Group'].index] == 5 and row[ds1.varlist['gender'].index]==2:
                v52 = v52+1
            if row[ds1.varlist['Group'].index] == 6 and row[ds1.varlist['gender'].index]==1:
                v61 = v61+1
            if row[ds1.varlist['Group'].index] == 6 and row[ds1.varlist['gender'].index]==2:
                v62 = v62+1
            if row[ds1.varlist['Group'].index] == 7 and row[ds1.varlist['gender'].index]==1:
                v71 = v71+1
            if row[ds1.varlist['Group'].index] == 7 and row[ds1.varlist['gender'].index]==2:
                v72 = v72+1
            if row[ds1.varlist['Group'].index] == 8 and row[ds1.varlist['gender'].index]==1:
                v81 = v81+1
```

```
                if row[ds1.varlist['Group'].index] == 8 and row[ds1.varlist['gender'].index]==2:
                    v82 = v82+1
                if row[ds1.varlist['Group'].index] == 9 and row[ds1.varlist['gender'].index]==1:
                    v81 = v81+1
                if row[ds1.varlist['Group'].index] == 9 and row[ds1.varlist['gender'].index]==2:
                    v82 = v82+1
                if row[ds1.varlist['Group'].index] == 10 and row[ds1.varlist['gender'].index]==1:
                    v101 = v101+1
                if row[ds1.varlist['Group'].index] == 10 and row[ds1.varlist['gender'].index]==2:
                    v102 = v102+1
        print("This is the key: ", key)
        print("The number of cases in each cell of crosstable in this unit is:/n", v11,v12,v21,
                                 v22,v31,v32,v41,v42,v51,v52,v61,v62,v71,v72,v81,v82,v101,v102)
        total = v11+ v12+v21+v22+v31+v32+v41+v42+v51+v52+v61+v62+v71+v72+v81+v82+v101+v102
        print("The summation of all cells in above crosstable:", total)
# compute the difference between reference population and reponses-----------------------------
        if ( ( isinstance(ds1.cases[begincasenum,ds1.varlist['E1E3M'].index][0], int) ) or
                    ( isinstance(ds1.cases[begincasenum,ds1.varlist['E1E3M'].index][0], float) ) ):
            diff11=ds1.cases[begincasenum,ds1.varlist['E1E3M'].index][0] - v11
            diff12=ds1.cases[begincasenum,ds1.varlist['E1E3F'].index][0] - v12
            diff21=ds1.cases[begincasenum,ds1.varlist['E4E6M'].index][0] - v21
            diff22=ds1.cases[begincasenum,ds1.varlist['E4E6F'].index][0] - v22
            diff31=ds1.cases[begincasenum,ds1.varlist['E7E9M'].index][0] - v31
            diff32=ds1.cases[begincasenum,ds1.varlist['E7E9F'].index][0] - v32
            diff41=ds1.cases[begincasenum,ds1.varlist['WOM'].index][0] - v41
            diff42=ds1.cases[begincasenum,ds1.varlist['WOF'].index][0] - v42
            diff51=ds1.cases[begincasenum,ds1.varlist['O1O3M'].index][0] - v51
            diff52=ds1.cases[begincasenum,ds1.varlist['O1O3F'].index][0] - v52
            diff61=ds1.cases[begincasenum,ds1.varlist['O4AboveM'].index][0] - v61
            diff62=ds1.cases[begincasenum,ds1.varlist['O4AboveF'].index][0] - v62
            diff71=ds1.cases[begincasenum,ds1.varlist['GS1GS8M'].index][0] - v71
            diff72=ds1.cases[begincasenum,ds1.varlist['GS1GS8F'].index][0] - v72
            diff81=ds1.cases[begincasenum,ds1.varlist['GS9_SESM'].index][0] - v81
            diff82=ds1.cases[begincasenum,ds1.varlist['GS9_SESF'].index][0] - v82
            diff101=ds1.cases[begincasenum,ds1.varlist['OtherM'].index][0] - v101
            diff102=ds1.cases[begincasenum,ds1.varlist['OtherF'].index][0] - v102
            difflist=['diff11', 'diff12','diff21', 'diff22','diff31', 'diff32','diff41', 'diff42',
            'diff51', 'diff52','diff61', 'diff62','diff71', 'diff72','diff81', 'diff82','diff101', 'diff102']

            samplecase=ds1.cases[begincasenum]
            indexofgroup = ds1.varlist['Group'].index
            indexofgender = ds1.varlist['gender'].index
#generate non-resonsese rows---------------------------------------------------------
            for var in difflist:
                  num1=var.replace("diff","")
                  num2=int(num1)
                  gendernum = int(num2%10)
                  groupnum = int((num2-gendernum)/10)

                  valueofvar= int(eval(var))
                  if valueofvar < 0:
```

```
           print("The value of 'wrong' difference is:", valueofvar)
           print("there exist error at unit:", d1[key][0])
           continue
         if valueofvar == 0:
           continue
         for i in range(valueofvar):
           for j in range(totalvar):
             if j == indexofgender:
               samplecase[j] = gendernum
             elif (j >=49) and (j<=199):
               samplecase[j] = None
             elif j == indexofgroup:
               samplecase[j] = groupnum
             elif (j >=201) and (j<=305):
               samplecase[j] = None
             elif j == (totalvar-2):
               samplecase[j] = 0
             elif j == (totalvar-1):
               samplecase[j] = None
           newds1.cases.append(samplecase)

#-----------------------------------------
 strdept = str(int(deptid))
 name=list(dsNames.keys())[0]
 spss.Submit(r"""
    DATASET ACTIVATE %(name)s.
    SAVE OUTFILE='E:\WEI WAN\My SPSS\unitfile3\unit_%(strdept)s.sav'.
    DATASET CLOSE %(name)s.
           """ %locals())

spss.Submit(r"""
  DATASET ACTIVATE alldata.
  DATASET CLOSE ALL.
            """ %locals())

End Program.
```

The outcome is as follows:

unit_180478	unit_181109	unit_182200	unit_182631	unit_183170
unit_180479	unit_181110	unit_182267	unit_182632	unit_183173
unit_180480	unit_181111	unit_182295	unit_182633	unit_183191
unit_180481	unit_181142	unit_182341	unit_182634	unit_183192
unit_180482	unit_181204	unit_182342	unit_182639	unit_183193
unit_180483	unit_181237	unit_182343	unit_182640	unit_183195
unit_180484	unit_181346	unit_182344	unit_182655	unit_183216
unit_180485	unit_181421	unit_182345	unit_182668	unit_183253
unit_180493	unit_181443	unit_182346	unit_182670	unit_183293
unit_180543	unit_181478	unit_182401	unit_182679	unit_183297
unit_180628	unit_181495	unit_182418	unit_182713	unit_183298
unit_180645	unit_181544	unit_182454	unit_182767	unit_183299
unit_180647	unit_181695	unit_182457	unit_182768	unit_183300
unit_180648	unit_181736	unit_182465	unit_182769	unit_183301
unit_180649	unit_181778	unit_182466	unit_182770	unit_183302
unit_180651	unit_181818	unit_182467	unit_182771	unit_183311
unit_180652	unit_181819	unit_182469	unit_182772	unit_183312
unit_180655	unit_181820	unit_182476	unit_182794	unit_183313
unit_180671	unit_181821	unit_182481	unit_182823	unit_183338
unit_180694	unit_181822	unit_182483	unit_182838	unit_183355
unit_180732	unit_181825	unit_182484	unit_182868	unit_183363
unit_180764	unit_181843	unit_182485	unit_182915	unit_183364
unit_180786	unit_181942	unit_182487	unit_182938	unit_183366
unit_180819	unit_181947	unit_182488	unit_182939	unit_183368
unit_180857	unit_181960	unit_182510	unit_182940	unit_183369
unit_180863	unit_181961	unit_182549	unit_182941	unit_183377
unit_180865	unit_181964	unit_182562	unit_182942	unit_183387
unit_180868	unit_181965	unit_182601	unit_182943	unit_183389
unit_180913	unit_181968	unit_182603	unit_182944	unit_183391

Occurrence	ACTSERVICE	WRank	PopNum	filter_$	Group	gender	indicator	casen	totalcases
1.00	7.00	4.00	15930.00	.00	2.00	1.00	1.00	10972.00	1392.00
1.00	7.00	4.00	.00	.00	2.00	1.00	1.00	10973.00	1392.00
1.00	7.00	4.00	47106.00	.00	2.00	1.00	1.00	10974.00	1392.00
1.00	7.00	4.00	47106.00	.00	2.00	1.00	1.00	10975.00	1392.00
1.00	7.00	4.00	47106.00	.00	2.00	1.00	1.00	10976.00	1392.00
1.00	7.00	4.00	47106.00	.00	2.00	1.00	1.00	10977.00	1392.00
1.00	7.00	4.00	47106.00	.00	2.00	1.00	1.00	10978.00	1392.00
1.00	7.00	4.00	8100.00	.00	2.00	2.00	1.00	10979.00	1392.00
1.00	7.00	4.00	8100.00	.00	2.00	2.00	1.00	10980.00	1392.00
1.00	7.00	4.00	8100.00	.00	2.00	2.00	1.00	10981.00	1392.00
1.00	7.00	4.00	8100.00	.00	2.00	2.00	1.00	10982.00	1392.00
1.00	7.00	4.00	9122.00	.00	2.00	2.00	1.00	10983.00	1392.00
1.00	7.00	5.00	10021.00	.00	3.00	1.00	1.00	10984.00	1392.00
1.00	7.00	5.00	10021.00	.00	3.00	1.00	1.00	10985.00	1392.00
.	.	.	.	.	1.00	1.00	.00	.	1392.00
.	.	.	.	.	1.00	1.00	.00	.	1392.00
.	.	.	.	.	1.00	1.00	.00	.	1392.00
.	.	.	.	.	1.00	1.00	.00	.	1392.00
.	.	.	.	.	1.00	2.00	.00	.	1392.00
.	.	.	.	.	1.00	2.00	.00	.	1392.00
.	.	.	.	.	1.00	2.00	.00	.	1392.00
.	.	.	.	.	2.00	1.00	.00	.	1392.00
.	.	.	.	.	2.00	1.00	.00	.	1392.00
.	.	.	.	.	2.00	1.00	.00	.	1392.00
.	.	.	.	.	2.00	1.00	.00	.	1392.00

6. Weighting and Outcome

We have three ways to compute weights. The first one is Option 1 at section 2.3.2. The reference population is unit population. Follow table is the outcome of this method.

Table 7: Weights from Unit Reference Population

weights_xDataset1_unitpop.sav [DataSet1] - IBM SPSS Statistics Data Editor

File Edit View Data Transform Analyze Direct Marketing Graphs Utilities Extensions Window Help

	unitcasenumber	unitid	weight_0	weight_1	weight_2	weight_3	weight_4	weight_5	weight_6	weight_7	weight_8	weight_9	weight_10	weight_11	weight_12
1	1.00	129933.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	2.00	135253.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	3.00	161338.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
4	4.00	161339.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
5	5.00	161730.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
6	6.00	162772.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
7	7.00	165204.00	3.75	.80	.97	.69	1.07	1.00	1.00	1.00	1.07	1.00	1.00	1.00	1.00
8	8.00	166080.00	1.00	.83	1.32	.51	.46	.28	1.00	1.00	.28	1.00	1.00	1.00	1.52
9	9.00	166370.00	4.27	1.00	.44	.98	1.73	1.00	1.00	1.00	1.00	.13	1.00	1.00	1.00
10	10.00	166757.00	1.49	.61	.89	.60	3.90	1.00	.78	1.00	1.56	1.00	1.00	1.00	1.00
11	11.00	167247.00	1.00	1.00	1.00	1.00	.50	1.00	.94	1.00	2.41	1.26	.16	1.00	.84
12	12.00	167560.00	1.12	1.00	.92	1.00	1.00	1.00	1.00	1.00	2.05	1.00	1.00	1.00	1.00
13	13.00	167713.00	1.17	1.00	.99	1.00	.92	1.00	1.00	1.00	.71	1.00	1.00	1.00	1.00
14	14.00	167716.00	1.45	1.00	.79	1.00	1.10	1.00	1.00	1.00	1.65	1.00	1.00	1.00	1.00
15	15.00	167988.00	1.00	1.00	.42	1.08	1.89	1.08	1.00	1.00	1.00	1.00	1.00	1.00	1.00
16	16.00	168840.00	1.64	2.03	.66	.94	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
17	17.00	168841.00	1.00	1.00	.74	.28	1.21	.55	1.00	.55	.55	.28	3.86	1.00	1.00
18	18.00	168872.00	.60	1.00	1.60	1.00	1.24	1.00	1.00	1.00	.83	1.00	1.00	1.00	1.00
19	19.00	169151.00	3.47	.30	2.19	1.09	.82	1.00	1.00	1.00	.89	1.00	1.00	1.00	1.00
20	20.00	169154.00	.62	.86	1.24	.86	.70	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
21	21.00	169155.00	1.33	1.00	.95	2.09	1.01	1.00	1.00	1.00	1.00	1.00	1.00	.19	1.00
22	22.00	169834.00	1.00	1.00	.91	1.00	1.27	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
23	23.00	170115.00	.71	1.00	1.54	2.50	3.04	1.43	1.00	1.00	1.00	1.00	1.00	1.00	1.00
24	24.00	170175.00	.70	1.00	.95	.84	1.13	.56	1.00	1.00	1.69	.28	1.00	1.00	1.00
25	25.00	171053.00	1.00	1.00	1.24	1.71	.82	.43	1.00	1.00	.43	1.00	1.00	1.00	1.00

The second one is also Option 1 at section 2.3.2, but the reference population is the whole population.

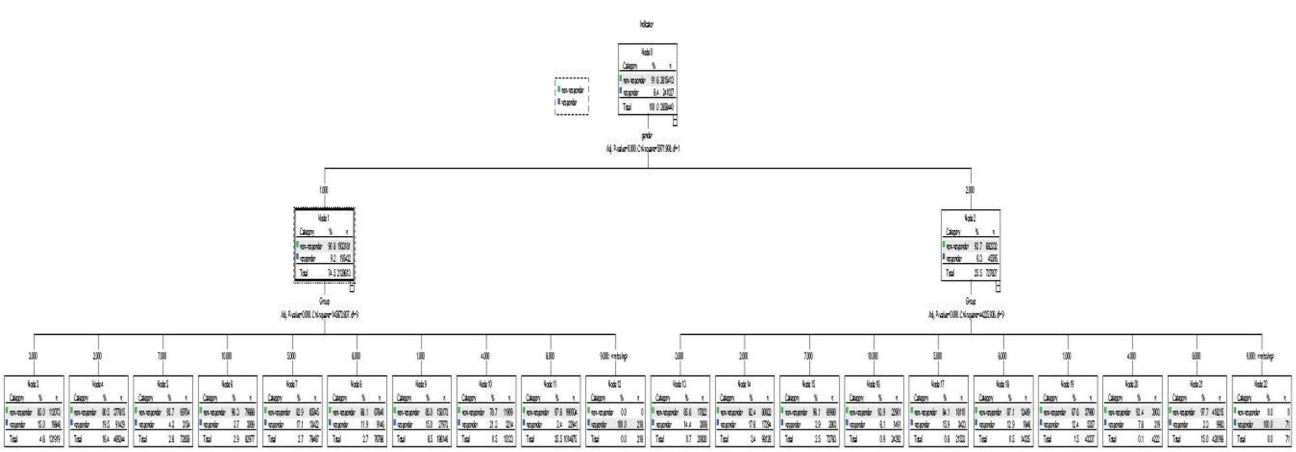
Table 8: Weights from Whole Reference Population

Myweights0703201701_xDataset1_wholepop.sav [DataSet3] - IBM SPSS Statistics Data Editor

	unitcasenumber	unitid	Pweight_0	Pweight_1	Pweight_2	Pweight_3	Pweight_4	Pweight_5	Pweight_6	Pweight_7	Pweight_8	Pweight_9	Pweight_10	Pweight_11	Pweight_12
163	163.00	180469.00	.29	1.27	.31	1.44	.43	.61	.44	1.00	.77	1.00	.75	1.00	1.00
164	164.00	180474.00	1.38	.32	.48	.23	.43	.27	.08	.22	.25	.22	.66	.74	1.00
165	165.00	180478.00	.37	1.00	.31	.40	1.06	.08	1.00	1.00	.32	1.00	1.00	1.00	1.00
166	166.00	180479.00	1.00	1.00	.44	.40	.19	.17	1.00	.07	.14	1.00	.62	1.00	1.00
167	167.00	180480.00	.55	.51	.26	.29	1.03	.49	1.00	1.00	1.00	.25	1.00	1.00	1.71
168	168.00	180481.00	.27	.48	.28	.72	1.45	1.00	1.00	1.00	.43	1.00	1.00	1.00	1.00
169	169.00	180482.00	.74	.52	.25	.21	3.14	.50	1.00	1.00	1.87	1.00	1.00	1.00	1.00
170	170.00	180483.00	.28	.18	.35	.36	.63	1.00	.50	1.00	.87	1.00	1.00	1.00	1.00
171	171.00	180484.00	.32	.21	.32	.30	.90	1.00	.51	.14	1.33	1.00	1.00	1.00	1.00
172	172.00	180485.00	.47	.39	.26	.44	1.00	1.00	.09	1.00	1.00	.38	1.00	1.00	1.00
173	173.00	180493.00	.33	1.00	.51	.12	1.15	1.00	1.00	1.00	.23	.19	1.00	1.00	1.00
174	174.00	180543.00	1.00	1.00	.52	.65	.18	1.00	.10	.03	.52	1.00	.51	1.00	1.00
175	175.00	180628.00	1.00	1.00	.53	.18	.37	1.00	.04	1.00	1.00	1.00	.11	1.00	1.00
176	176.00	180645.00	1.00	1.00	3.45	1.00	.18	1.00	.02	1.00	.58	.32	.07	1.00	1.00
177	177.00	180647.00	1.00	1.00	.72	.15	.25	.05	1.00	1.00	.60	.17	.59	1.00	1.00
178	178.00	180648.00	1.00	1.00	.63	.18	.65	.15	1.00	1.00	.58	.06	.13	.11	1.00
179	179.00	180649.00	1.50	1.00	.28	.40	.27	.50	1.00	1.00	1.00	1.00	1.85	1.00	1.76
180	180.00	180651.00	.98	1.00	.33	.29	.21	.22	1.00	1.00	.82	1.00	1.00	.30	1.00
181	181.00	180652.00	1.00	1.03	.33	.29	.21	.12	1.00	1.00	1.87	.51	1.83	1.00	1.00
182	182.00	180655.00	1.00	1.00	1.12	.07	1.00	.08	1.00	1.00	1.00	.09	.30	1.00	1.00
183	183.00	180671.00	2.15	1.00	.20	.57	1.52	1.00	1.00	1.00	.60	1.00	1.00	1.00	1.00
184	184.00	180694.00	.18	.24	.34	1.06	.95	1.00	1.00	1.00	1.70	1.00	1.00	1.00	1.00
185	185.00	180732.00	1.63	1.00	.46	.34	.26	.18	.13	1.00	.20	.38	.45	1.00	1.00
186	186.00	180764.00	1.00	.34	.74	.14	.69	.11	1.00	1.00	.62	.17	.30	.03	1.00
187	187.00	180786.00	1.00	1.00	1.22	.21	.21	.04	.05	1.00	1.02	.07	.99	.19	1.00

The third one is done by Component 3 (Decision Tree method in Section 3). Following is outcome of decision tree.

Figure 2: Decision Tree



Following table is from decision tree. The column "WeightfromDecisionTree" is the weight.

Table 9: Weights from Decision Tree

NodeID	Total	responses	ResponseProbability	Weight	PredictedProbability_2_mean	PRE_1_mean	Total_sum	responses_sum	RescaledWeightRate	Rescaled Weight	WeightfromDecisionTree	RescaledWeightfromDecisionTree
3	131919	19846	.1504	6.65	.15	.13	2856440	236649	.08	.55	7.69	.64
4	469244	91429	.1948	5.13	.19	.21	2856440	236649	.08	.43	4.72	.39
5	72858	3154	.0433	23.10	.04	.04	2856440	236649	.08	1.91	28.51	2.36
6	82977	3089	.0372	26.86	.04	.03	2856440	236649	.08	2.23	30.80	2.55
7	78467	13422	.1711	5.85	.17	.16	2856440	236649	.08	.48	6.10	.51
8	76786	9146	.1191	8.40	.12	.09	2856440	236649	.08	.70	11.11	.92
9	186146	27973	.1503	6.65	.15	.13	2856440	236649	.08	.55	7.70	.64
10	15123	3214	.2125	4.71	.21	.25	2856440	236649	.08	.39	3.95	.33
11	1014875	23941	.0236	42.39	.02	.03	2856440	236649	.08	3.51	36.68	3.04
12	218	218	1.0000	1.00	1.00	1.00	2856440	236649	.08	.08	1.00	.08
13	20830	3008	.1444	6.92	.14	.12	2856440	236649	.08	.57	8.24	.68
14	98126	17294	.1762	5.67	.18	.17	2856440	236649	.08	.47	5.76	.48
15	72783	2803	.0385	25.97	.04	.03	2856440	236649	.08	2.15	30.30	2.51
16	24392	1491	.0611	16.36	.06	.04	2856440	236649	.08	1.36	22.74	1.88
17	21533	3423	.1590	6.29	.16	.14	2856440	236649	.08	.52	6.98	.58
18	14335	1846	.1288	7.77	.13	.10	2856440	236649	.08	.64	9.90	.82
19	43337	5357	.1236	8.09	.12	.09	2856440	236649	.08	.67	10.53	.87
20	4222	319	.0756	13.24	.08	.05	2856440	236649	.08	1.10	18.97	1.57
21	428198	5357	.0125	79.93	.02	.03	2856440	236649	.08	6.62	36.81	3.05
22	71	319	4.4930	.22	1.00	1.00	2856440	236649	.08	.02	1.00	.08

After we computed these three types of weights, we use algorithm Component 5 to attach these weights to each case.

Table 10: Three Weights in Original Dataset

originaldatawithlocalandglobalanddecisiontreeweights2.sav [DataSet6] - IBM SPSS Statistics Data Editor

File Edit View Data Transform Analyze Direct Marketing Graphs Utilities Extensions Window Help

	k	PopNum	filter_$	TotalN	indicator	localweight	globalweight	weightfromdecisiontree2	localtimesglobal
10924	JO	2796	Not Selected	1392.00	1	.12	.09	6.98	.01
10925	JO	2796	Not Selected	1392.00	1	.12	.09	6.98	.01
10926	SO	2569	Not Selected	1392.00	1	.04	.03	9.90	.00
10927	SO	1339	Not Selected	1392.00	1	.07	.09	11.11	.01
10928	SO	418	Not Selected	1392.00	1	.07	.09	11.11	.01
10929	SO	8510	Not Selected	1392.00	1	.07	.09	11.11	.01
10930	SO	8510	Not Selected	1392.00	1	.07	.09	11.11	.01
10931	SO	8510	Not Selected	1392.00	1	.07	.09	11.11	.01
10932	SO	8510	Not Selected	1392.00	1	.07	.09	11.11	.01
10933	SO	834	Not Selected	1392.00	1	.04	.03	9.90	.00
10934	SO	1717	Not Selected	1392.00	1	.04	.03	9.90	.00
10935	SO	1717	Not Selected	1392.00	1	.04	.03	9.90	.00
10936	ICO	47106	Not Selected	1392.00	1	2.21	3.94	4.72	8.70
10937	ICO	10226	Not Selected	1392.00	1	.28	.55	7.69	.15
10938	ICO	4200	Not Selected	1392.00	1	.28	.55	7.69	.15
10939	ICO	0	Not Selected	1392.00	1	.24	.18	8.24	.04
10940	WO	168	Not Selected	1392.00	1	.07	.13	3.95	.01
10941	ICO	.	.	1392.00	1	.09	.18	7.69	.02
10942	ICO	.	.	1392.00	1	.09	.18	7.69	.02
10943	ICO	.	.	1392.00	1	.21	.16	5.76	.03
10944	JE	5117	Not Selected	1392.00	1	.05	1.24	7.70	.07
10945	ICO	15930	Not Selected	1392.00	1	.19	.35	4.72	.07
10946	ICO	15930	Not Selected	1392.00	1	.19	.35	4.72	.07
10947	ICO	15930	Not Selected	1392.00	1	.19	.35	4.72	.07
10948	ICO	15930	Not Selected	1392.00	1	.19	.35	4.72	.07
10949	ICO	0	Not Selected	1392.00	1	.19	.35	4.72	.07

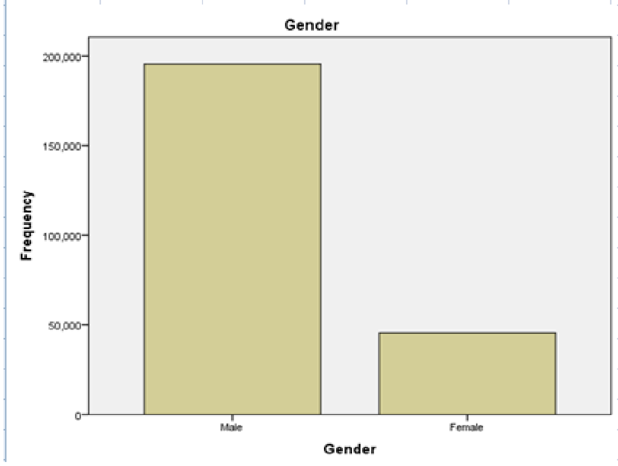
In the following, we would like show effects of weights some group and gender variables:

Figure 3: Effects of Weight on Distribution of Variables

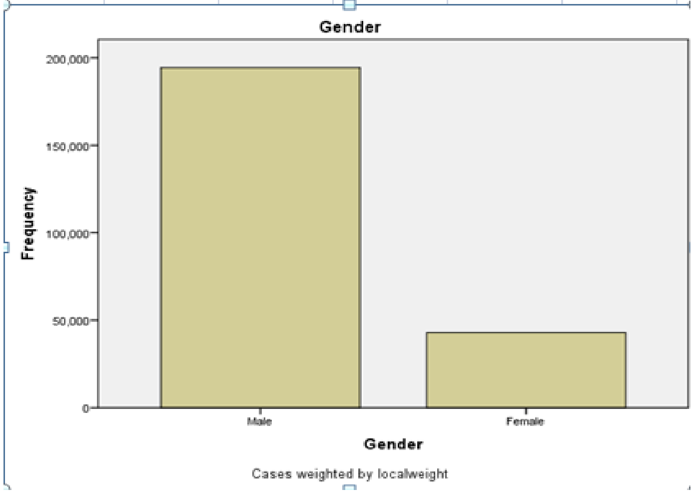
	B	C	D	E	F
1	Group				
2		Frequency	Percent	Valid Percent	Cumulative Percent
3	1.00 E1 -	33330	13.8	13.8	13.8
4	2.00 E4 -	108723	45.1	45.1	59.0
5	3.00 E7 -	22854	9.5	9.5	68.4
6	4.00 W1 -	3533	1.5	1.5	69.9
7	5.00 O1 -	16845	7.0	7.0	76.9
8	6.00 O4 -	10992	4.6	4.6	81.5
9	7.00 Grade	5957	2.5	2.5	83.9
10	8.00 Grade	33924	14.1	14.1	98.0
11	9.00 SES	195	0.1	0.1	98.1
12	10.00 Other	4580	1.9	1.9	100.0
13	Total	240933	100.0	100.0	
14	System	94	0.0		
15		241027	100.0		

O	P	Q	R	S	T
Group					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1.00 E1 - E3	36138	15.2	15.2	15.2
	2.00 E4 - E6	106118	44.7	44.7	60.0
	3.00 E7 - E9	21538	9.1	9.1	69.0
	4.00 W1 -	3255	1.4	1.4	70.4
	5.00 O1 - O3	15648	6.6	6.6	77.0
	6.00 O4 - O6	9370	3.9	3.9	80.9
	7.00 Grade 1 -	6865	2.9	2.9	83.8
	8.00 Grade 9 -	29104	12.3	12.3	96.1
	9.00 SES	195	0.1	0.1	96.2
	10.00 Other	9061	3.8	3.8	100.0
	Total	237292	100.0	100.0	
Missing	System	94	0.0		
Total		237386	100.0		

A	B	C	D	E	F
gender Gender					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 Male	195432	81.1	81.1	81.1
	2 Female	45595	18.9	18.9	100.0
	Total	241027	100.0	100.0	



O	P	Q	R	S	T
gender Gender					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1 Male	194369	81.9	81.9	81.9
	2 Female	43017	18.1	18.1	100.0
	Total	237386	100.0	100.0	



Following we used T-test to compare these three types of weights.

Figure 4: Comparing Weights by t-Test

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper			
Pair 1	newlocal - weightfromdecisiontree	.02058	1.14790	.00234	.01600	.02517	8.804	241026	.000
Pair 2	newglobal - weightfromdecisiontree	-.46624	1.08309	.00221	-.47056	-.46191	-211.336	241026	.000
Pair 3	newlocalandglobal - weightfromdecisiontree	-.43887	1.33023	.00271	-.44418	-.43356	-161.973	241026	.000
Pair 4	newlocal - newglobal	.48682	1.00459	.00205	.48281	.49083	237.910	241026	.000

From above results and comparison study, we can make conclusion:

- In Decision Tree, "Gender" and "Gender" are used to predict a response. More variables will lead to better weights.
- Logistic and Decision Tree method is most helpful when we are selecting a sample form an informative sampling frame.
- Weight by using unit reference population is "similar" on average to weight from decision tree.
- Weights do lead to changes of distribution.

7. References

(1) Barbara Lepidus Carlson, Stephen Williams, "A COMPARISON OF TWO METHODS TO ADJUST WEIGHTS FOR NON-RESPONSE: PROPENSITY MODELING AND WEIGHTING CLASS ADJUSTMENTS", Proceedings of the Annual Meeting of the American Statistical Association, August 5-9, 2001

(2) Che-Chern Lin, Hung-Jen Yang and Lung-Hsing Kuo, "Behaviour analysis of internet survey completion using decision trees An exploratory study", Online Information Review, 1 July 2008, pp. 117-130.

(3) Chris Skinner, “What is Survey Weighting?”, http://eprints.ncrm.ac.uk/1358/1/Weighting%20Festival%202010.pdf

(4) Cosma Shalizi, “Logistic Regression”, www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12.pdf

(5) David R. Johnson, “Using Weights in the Analysis of Survey Data”, http://web.pop.psu.edu/projects/help_archive/help.pop.psu.edu/help-by-statistical-method/weighting/Introduction%20to%20survey%20weights%20pri%20version.ppt/at_download/Introduction%20to%20survey%20weights%20pri%20version.ppt, November 2008.

(6) Deng, P.-S. (1996), “Using case-based reasoning approach to the support of ill-structured decisions”, European Journal of Operational Research, Vol. 93, pp. 511-21.

(7) Eun Sul Lee, Ronald N. Forthofer, “Analyzing Complex Survey Data”, SAGE Publications, Inc, 2nd edition, September 22, 2005.

(8) Floyd J. Fowler, “Survey Research Methods”, 5th Edition SAGE Publications, Inc., 5 edition, September 18, 2013.

(9) Graham Kalton and Ismael Flores-Cervantes, “Weighting Methods”, Journal of Official Statistics, Vol. 19, No. 2, 2003, pp. 81-97.

(10) IBM, “IBM SPSS Decision Trees 20”.

(11) IBM, “Python Reference Guide for IBM SPSS Statistics”.

(12) IBM, “IBM SPSS Modeler 18.0 User's Guide”.

(13) Ibrahim S. Yansaneh, “Construction and use of sample weights”, Expert Group Meeting to Review the Draft Handbook on Designing of Household Sample Surveys, 3-5 December 2003.

(14) Indurkhya, N. and Weiss, S.M. (1998), “Estimating performance gains for voted decision trees”, Intelligent Data Analysis, Vol. 2, pp. 303-10.

(15) Jae Kwang Kim, C. J. Skinner, “Weighting in survey analysis under informative sampling”,

Volume 100, Issue 2, June 2013

(16) Leslie Kish, Survey Sampling. New York: John Wiley and Sons, 1965, 1995.

(17) Mark Lutz, “Learning Python”, Fifth Edition, O’Reilly Media, Inc., June 2013

(18) Mendonca, L.F., Vieira, S.M. and Sousa, J.M.C. (2007), “Decision tree search methods in fuzzy modeling and classification”, International Journal of Approximate Reasoning, Vol. 44, pp. 106-23.

(19) Mugambi, E.M., Hunter, A., Oatley, G. and Kennedy, L. (2004), “Polynomial-fuzzy decision tree structures for classifying medical data”, Knowledge Based Systems, Vol. 17, pp. 81-7.

(20) Neil Malhotra, Annie Franco, Gabor Simonovits, L.J. Zigerell, “Developing Standards for Post-Stratification Weighting in Population-Based Survey Experiments”, web.stanford.edu/~neilm/weights_may1_final_identified.pdf

(21) PEAS, "Adjusting for non-response by weighting", http://www.restore.ac.uk/PEAS/nonresponse.php.

(22) Richard J. Harris, “Mini-Report: Use of Weights with a Sample from the DEOCS”, Version 3.3, 3/17-3/24, 20111, DEOMI, Summer 2014.

(23) Robert M. Groves, et al., Survey Methodology, 2nd edition, Hoboken, NJ: John Wiley and Sons, 2009.

(24) Sarasin, F.P. (2001), “Decision analysis and its application in clinical medicine”, European Journal of Obstetrics & Gynecology and Reproductive Biology, Vol. 94, pp. 172-9.

(25) Swaroop C H, “A Byte of Python”, ebshelf Inc., September 29, 2013

(26) Tsujino, K. (1995), “Implementation and refinement of decision trees using neural networks for hybrid knowledge acquisition”, Artificial Intelligence in Engineering, Vol. 9, pp. 265-75.

(27) Wang, J.-L. and Chan, S.-H. (2006), “Stock market trading rule discovery using two-layer bias

decision tree", Expert Systems with Applications, Vol. 30, pp. 605-11.

(28) Wikipedia, "Python (programming language)", https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-About-25

(29) Yan-yan SONG,Ying LU, "Decision tree methods: applications for classification and prediction", Shanghai Arch Psychiatry, 2015; 27(2): 130-135.

Appendix A: Algorithm Component 1

```
output close all.
GET FILE='E:\WEI WAN\My SPSS\TestFiles\AlldatawithIndicator.sav'.
DATASET NAME alldata.
SORT CASES BY AFEOCAID.
compute casen = $CASENUM.
formats casen(f12.0).
VARIABLE LEVEL casen (SCALE).
EXECUTE.

BEGIN PROGRAM.
import spss
#---generate a list of variable names------------
ordlist=[]
ordlist2=[]
for i in range(spss.GetVariableCount()):
        ordlist.append(spss.GetVariableName(i))
        ordlist2.append(spss.GetVariableType(i))

totalvar = spss.GetVariableCount()
#-------change from unicode to string------------
length = len(ordlist)
newordlist=[]
for i in range(length):
    newordlist.append([ordlist[i].encode('ascii','ignore'),ordlist2[i]])
#--end-----------------------------------------------------
#-generate a dictionary {unit number: [unitID, begincasenum, endcasenum]}-
# with spss.DataStep() starts a block to manipulate data------------------------
with spss.DataStep():
    ds = spss.Dataset() #---create a pointer to point at current active dataset--
    i =1
    d1=dict()
    #-i: unit number--j = [deptid, beginingcasenum, endingcasenum]--------------
    j=[ds.cases[0,1][0], ds.cases[0,totalvar-1][0],ds.cases[0,totalvar-1][0]]
    d1={i:j}
    iteration = 0
    for r in ds.cases:
        if (r[1]!=d1[i][0]):
            d1[i][2]=d1[i][1]+ iteration-1
            i=i+1
            newj=[r[1], d1[i-1][2]+1, d1[i-1][2]+1]
            d1[i]=newj
            iteration=0
        iteration=iteration+1
    d1[i][2]=ds.cases[-1,totalvar-1][0]
#following is a long process, for each unit compute each cell in cross tab,------
#and compute the difference---
for key in d1:
  with spss.DataStep():
        ds1 = spss.Dataset(name="alldata")
# Create a new dataset for each unit---------------------------------------------
        newds1 = spss.Dataset(name=None)
# Add variables to this new dataset----------------------------------------------
        for i in range(length):
                 newds1.varlist.append(newordlist[i][0],newordlist[i][1])
```

```

        deptid = d1[key][0]
        dsNames = {newds1.name : deptid}
        begincasenum=int(d1[key][1])-1
        endcasenum=int(d1[key][2])

        templist=[0]*18
        v11,v12,v21,v22,v31,v32,v41,v42,v51,v52,v61,v62,v71,v72,v81,v82,v101,v102=templist
        for row in ds1.cases[begincasenum : endcasenum]:
# add this case to the new file ---------------------------------------------------
            newds1.cases.append(row)
# count the number of each cell in Group and gender crosstab-----------------------
            if row[ds1.varlist['Group'].index] == 1 and row[ds1.varlist['gender'].index]==1:
                v11 = v11+1
            if row[ds1.varlist['Group'].index] == 1 and row[ds1.varlist['gender'].index]==2:
                v12 = v12+1
            if row[ds1.varlist['Group'].index] == 2 and row[ds1.varlist['gender'].index]==1:
                v21 = v21+1
            if row[ds1.varlist['Group'].index] == 2 and row[ds1.varlist['gender'].index]==2:
                v22 = v22+1
            if row[ds1.varlist['Group'].index] == 3 and row[ds1.varlist['gender'].index]==1:
                v31 = v31+1
            if row[ds1.varlist['Group'].index] == 3 and row[ds1.varlist['gender'].index]==2:
                v32 = v32+1
            if row[ds1.varlist['Group'].index] == 4 and row[ds1.varlist['gender'].index]==1:
                v41 = v41+1
            if row[ds1.varlist['Group'].index] == 4 and row[ds1.varlist['gender'].index]==2:
                v42 = v42+1
            if row[ds1.varlist['Group'].index] == 5 and row[ds1.varlist['gender'].index]==1:
                v51 = v51+1
            if row[ds1.varlist['Group'].index] == 5 and row[ds1.varlist['gender'].index]==2:
                v52 = v52+1
            if row[ds1.varlist['Group'].index] == 6 and row[ds1.varlist['gender'].index]==1:
                v61 = v61+1
            if row[ds1.varlist['Group'].index] == 6 and row[ds1.varlist['gender'].index]==2:
                v62 = v62+1
            if row[ds1.varlist['Group'].index] == 7 and row[ds1.varlist['gender'].index]==1:
                v71 = v71+1
            if row[ds1.varlist['Group'].index] == 7 and row[ds1.varlist['gender'].index]==2:
                v72 = v72+1
            if row[ds1.varlist['Group'].index] == 8 and row[ds1.varlist['gender'].index]==1:
                v81 = v81+1
            if row[ds1.varlist['Group'].index] == 8 and row[ds1.varlist['gender'].index]==2:
                v82 = v82+1
            if row[ds1.varlist['Group'].index] == 9 and row[ds1.varlist['gender'].index]==1:
                v81 = v81+1
            if row[ds1.varlist['Group'].index] == 9 and row[ds1.varlist['gender'].index]==2:
                v82 = v82+1
            if row[ds1.varlist['Group'].index] == 10 and row[ds1.varlist['gender'].index]==1:
                v101 = v101+1
            if row[ds1.varlist['Group'].index] == 10 and row[ds1.varlist['gender'].index]==2:
                v102 = v102+1
        print("This is the key: ", key)
```

```
        print("The number of cases in each cell of crosstable in this unit is:/n", v11,v12,v21,
          v22,v31,v32,v41,v42,v51,v52,v61,v62,v71,v72,v81,v82,v101,v102)
        total = v11+ v12+v21+v22+v31+v32+v41+v42+v51+v52+v61+v62+v71+v72+v81+v82+v101+v102
        print("The summation of all cells in above crosstable:", total)
# compute the difference between reference population and reponses-----------------------------
        if (( isinstance(ds1.cases[begincasenum,ds1.varlist['E1E3M'].index][0], int) ) or
           ( isinstance(ds1.cases[begincasenum,ds1.varlist['E1E3M'].index][0], float) )):
            diff11=ds1.cases[begincasenum,ds1.varlist['E1E3M'].index][0] - v11
            diff12=ds1.cases[begincasenum,ds1.varlist['E1E3F'].index][0] - v12
            diff21=ds1.cases[begincasenum,ds1.varlist['E4E6M'].index][0] - v21
            diff22=ds1.cases[begincasenum,ds1.varlist['E4E6F'].index][0] - v22
            diff31=ds1.cases[begincasenum,ds1.varlist['E7E9M'].index][0] - v31
            diff32=ds1.cases[begincasenum,ds1.varlist['E7E9F'].index][0] - v32
            diff41=ds1.cases[begincasenum,ds1.varlist['WOM'].index][0] - v41
            diff42=ds1.cases[begincasenum,ds1.varlist['WOF'].index][0] - v42
            diff51=ds1.cases[begincasenum,ds1.varlist['O1O3M'].index][0] - v51
            diff52=ds1.cases[begincasenum,ds1.varlist['O1O3F'].index][0] - v52
            diff61=ds1.cases[begincasenum,ds1.varlist['O4AboveM'].index][0] - v61
            diff62=ds1.cases[begincasenum,ds1.varlist['O4AboveF'].index][0] - v62
            diff71=ds1.cases[begincasenum,ds1.varlist['GS1GS8M'].index][0] - v71
            diff72=ds1.cases[begincasenum,ds1.varlist['GS1GS8F'].index][0] - v72
            diff81=ds1.cases[begincasenum,ds1.varlist['GS9_SESM'].index][0] - v81
            diff82=ds1.cases[begincasenum,ds1.varlist['GS9_SESF'].index][0] - v82
            diff101=ds1.cases[begincasenum,ds1.varlist['OtherM'].index][0] - v101
            diff102=ds1.cases[begincasenum,ds1.varlist['OtherF'].index][0] - v102
            difflist=['diff11', 'diff12','diff21', 'diff22','diff31', 'diff32','diff41', 'diff42',
            'diff51', 'diff52','diff61', 'diff62','diff71', 'diff72','diff81', 'diff82','diff101', 'diff102']

            samplecase=ds1.cases[begincasenum]
            indexofgroup = ds1.varlist['Group'].index
            indexofgender = ds1.varlist['gender'].index
#generate non-resonsese rows--------------------------------------------------------------
            for var in difflist:
                  num1=var.replace("diff","")
                  num2=int(num1)
                  gendernum = int(num2%10)
                  groupnum = int((num2-gendernum)/10)

                  valueofvar= int(eval(var))
                  if valueofvar < 0:
                      print("The value of 'wrong' difference is:", valueofvar)
                      print("there exist error at unit:", d1[key][0])
                      continue
                  if valueofvar == 0:
                      continue
                  for i in range(valueofvar):
                      for j in range(totalvar):
                          if j == indexofgender:
                              samplecase[j] = gendernum
                          elif (j >=49) and (j<=199):
                              samplecase[j] = None
                          elif j == indexofgroup:
                              samplecase[j] = groupnum
                          elif (j >=201) and (j<=305):
                              samplecase[j] = None
                          elif j == (totalvar-2):
                              samplecase[j] = 0
                          elif j == (totalvar-1):
                              samplecase[j] = None
                      newds1.cases.append(samplecase)

  strdept = str(int(deptid))
  name=list(dsNames.keys())[0]
  spss.Submit(r"""
        DATASET ACTIVATE %(name)s.
        SAVE OUTFILE='E:\WEI WAN\My SPSS\unitfile3\unit_%(strdept)s.sav'.
        DATASET CLOSE %(name)s.
                      """ %locals())
spss.Submit(r"""
    DATASET ACTIVATE alldata.
    DATASET CLOSE ALL.
                        """ %locals())
End Program.
```

Appendix B: Algorithm Component 2

```
OUTPUT CLOSE ALL.
begin program.
import glob
import spss

# join all the sav files in the directory specified below.
# Remember to save the resulting file at the end
# The Data Editor may not show the last partial block of files
# until the SAVE or another procedure is run.
cmd = []
i = 0
first = True
for fcount, f in enumerate(glob.glob("E:/WEI WAN/My SPSS/unitfile3/*.sav")):
# specification for files to join
    if i >= 49:
        if first:
            cmdroot = ["ADD FILES"]
            first = False
        else:
            cmdroot = ["ADD FILES /FILE=*"]
        cmdroot.extend(cmd)
        spss.Submit(cmdroot)
        i = 0
        cmd = []

    cmd.append("""/FILE = "%s" """ % f)
    i += 1

# leftovers from last block
if cmd:
    cmdroot = ["ADD FILES /FILE=*"]
    cmdroot.extend(cmd)
    spss.Submit(cmdroot)

print "Files merged: %i, leftover count: %i" % (fcount+1, i)

spss.Submit(r"""
        SAVE OUTFILE='E:/WEI WAN/My SPSS/unitfile3/allcombined.sav'.
                    """)
end program.
```

Appendix C: Algorithm Component 3

```
OUTPUT CLOSE ALL.
* Decision Tree.
TREE indicator [n] BY Group [n] gender [n]
  /TREE DISPLAY=TOPDOWN NODES=STATISTICS BRANCHSTATISTICS=YES NODEDEFS=YES SCALE=AUTO
  /DEPCATEGORIES USEVALUES=[.00 1.00] TARGET=[.00 1.00]
  /PRINT MODELSUMMARY CLASSIFICATION RISK
  /GAIN CATEGORYTABLE=YES TYPE=[NODE] SORT=DESCENDING CUMULATIVE=NO
  /PLOT GAIN INDEX RESPONSE INCREMENT=5
  /SAVE NODEID PREDVAL PREDPROB
  /METHOD TYPE=EXHAUSTIVECHAID
  /GROWTHLIMIT MAXDEPTH=AUTO MINPARENTSIZE=100 MINCHILDSIZE=20
  /VALIDATION TYPE=NONE OUTPUT=BOTHSAMPLES
  /CHAID ALPHASPLIT=0.01 SPLITMERGED=YES CHISQUARE=PEARSON CONVERGE=0.001 MAXITERATIONS=200
    ADJUST=BONFERRONI
  /COSTS EQUAL
  /MISSING NOMINALMISSING=MISSING.


LOGISTIC REGRESSION VARIABLES indicator
  /METHOD=ENTER PredictedProbability_2
  /SAVE=PRED
  /CRITERIA=PIN(.05) POUT(.10) ITERATE(20) CUT(.5).
```

Appendix D: Algorithm Component 4-1

```
output close all.

GET FILE='E:\WEI WAN\My SPSS\TestFiles\AlldatawithoutIndicator.sav'.
DATASET NAME alldata.
SORT CASES BY AFEOCAID.
compute casen = $CASENUM.
formats casen(f12.0).
VARIABLE LEVEL casen (SCALE).
EXECUTE.

BEGIN PROGRAM.
import spss

totalvar = spss.GetVariableCount()
#-works:generate a dictionary {unit numberm: [unitID, begincasenum, endcasenum]}
#make sure you have case number--
with spss.DataStep():
    ds = spss.Dataset(name="alldata")
    i =1
    d1=dict()
    #-i: unit number--j = [deptid, beginingcasenum, endingcasenum]---make sure you have case number-
    j=[ds.cases[0,1][0], ds.cases[0,totalvar-1][0],ds.cases[0,totalvar-1][0]]
    d1={i:j}
    iteration = 0
    for r in ds.cases:
        if (r[1]!=d1[i][0]):
            d1[i][2]=d1[i][1]+ iteration-1
            i=i+1
            newj=[r[1], d1[i-1][2]+1, d1[i-1][2]+1]
            d1[i]=newj
            iteration=0
        iteration=iteration+1
    d1[i][2]=ds.cases[-1,totalvar-1][0]
#----------------------
weightdic=dict()
for key in d1:

  weight = [1.0]*18
  with spss.DataStep():
        ds1 = spss.Dataset(name="alldata")
#-----------------------------------------------
        begincasenum=int(d1[key][1])-1
        endcasenum=int(d1[key][2])
```

```
#-----working on here-------------------------
        v11,v12,v21,v22,v31,v32,v41,v42,v51,v52,v61,v62,v71,v72,v81,v82,v101,v102=[0.0]*18
        for row in ds1.cases[begincasenum : endcasenum]:
                if  row[ds1.varlist['Group'].index] == 1 and row[ds1.varlist['gender'].index]==1:
                    v11 = v11+1
                if  row[ds1.varlist['Group'].index] == 1 and row[ds1.varlist['gender'].index]==2:
                    v12 = v12+1
                if  row[ds1.varlist['Group'].index] == 2 and row[ds1.varlist['gender'].index]==1:
                    v21 = v21+1
                if  row[ds1.varlist['Group'].index] == 2 and row[ds1.varlist['gender'].index]==2:
                    v22 = v22+1
                if  row[ds1.varlist['Group'].index] == 3 and row[ds1.varlist['gender'].index]==1:
                    v31 = v31+1
                if  row[ds1.varlist['Group'].index] == 3 and row[ds1.varlist['gender'].index]==2:
                    v32 = v32+1
                if  row[ds1.varlist['Group'].index] == 4 and row[ds1.varlist['gender'].index]==1:
                    v41 = v41+1
                if  row[ds1.varlist['Group'].index] == 4 and row[ds1.varlist['gender'].index]==2:
                    v42 = v42+1
                if  row[ds1.varlist['Group'].index] == 5 and row[ds1.varlist['gender'].index]==1:
                    v51 = v51+1
                if  row[ds1.varlist['Group'].index] == 5 and row[ds1.varlist['gender'].index]==2:
                    v52 = v52+1
                if  row[ds1.varlist['Group'].index] == 6 and row[ds1.varlist['gender'].index]==1:
                    v61 = v61+1
                if  row[ds1.varlist['Group'].index] == 6 and row[ds1.varlist['gender'].index]==2:
                    v62 = v62+1
                if  row[ds1.varlist['Group'].index] == 7 and row[ds1.varlist['gender'].index]==1:
                    v71 = v71+1
                if  row[ds1.varlist['Group'].index] == 7 and row[ds1.varlist['gender'].index]==2:
                    v72 = v72+1
                if  row[ds1.varlist['Group'].index] == 8 and row[ds1.varlist['gender'].index]==1:
                    v81 = v81+1
                if  row[ds1.varlist['Group'].index] == 8 and row[ds1.varlist['gender'].index]==2:
                    v82 = v82+1
                if  row[ds1.varlist['Group'].index] == 9 and row[ds1.varlist['gender'].index]==1:
                    v81 = v81+1
                if  row[ds1.varlist['Group'].index] == 9 and row[ds1.varlist['gender'].index]==2:
                    v82 = v82+1
                if row[ds1.varlist['Group'].index] == 10 and row[ds1.varlist['gender'].index]==1:
                    v101 = v101+1
                if row[ds1.varlist['Group'].index] == 10 and row[ds1.varlist['gender'].index]==2:
                    v102 = v102+1

        total = float(v11+ v12+v21+v22+v31+v32+v41+v42+v51+v52+v61+v62+v71+v72+v81+v82+v101+v102)
        respondernum=[v11, v12, v21, v22, v31, v32, v41, v42, v51, v52, v61,
        v62, v71, v72, v81, v82, v101, v102]
        respondersratio=[v11/total, v12/total, v21/total, v22/total, v31/total, v32/total, v41/total,
        v42/total, v51/total, v52/total, v61/total, v62/total, v71/total, v72/total, v81/total,
        v82/total, v101/total, v102/total]

        referencepop=[0.0]*18;referenceratio=[0.0]*18
        if ( isinstance(ds1.cases[begincasenum,ds1.varlist['E1E3M'].index][0], (int, float))
            and isinstance(ds1.cases[begincasenum,ds1.varlist['E1E3F'].index][0], (int, float))
```

```
             and isinstance(ds1.cases[begincasenum,ds1.varlist['E4E6M'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['E4E6F'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['E7E9M'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['E7E9F'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['WOM'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['WOF'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['O1O3M'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['O1O3F'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['O4AboveM'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['O4AboveF'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['GS1GS8M'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['GS1GS8F'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['GS9_SESM'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['GS9_SESF'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['OtherM'].index][0], (int, float))
             and isinstance(ds1.cases[begincasenum,ds1.varlist['OtherF'].index][0], (int, float))
             ):
            referencepop[0] = ds1.cases[begincasenum,ds1.varlist['E1E3M'].index][0]
            referencepop[1] = ds1.cases[begincasenum,ds1.varlist['E1E3F'].index][0]
            referencepop[2] = ds1.cases[begincasenum,ds1.varlist['E4E6M'].index][0]
            referencepop[3] = ds1.cases[begincasenum,ds1.varlist['E4E6F'].index][0]
            referencepop[4] = ds1.cases[begincasenum,ds1.varlist['E7E9M'].index][0]
            referencepop[5] = ds1.cases[begincasenum,ds1.varlist['E7E9F'].index][0]
            referencepop[6] = ds1.cases[begincasenum,ds1.varlist['WOM'].index][0]
            referencepop[7] = ds1.cases[begincasenum,ds1.varlist['WOF'].index][0]
            referencepop[8] = ds1.cases[begincasenum,ds1.varlist['O1O3M'].index][0]
            referencepop[9] = ds1.cases[begincasenum,ds1.varlist['O1O3F'].index][0]
            referencepop[10] = ds1.cases[begincasenum,ds1.varlist['O4AboveM'].index][0]
            referencepop[11] = ds1.cases[begincasenum,ds1.varlist['O4AboveF'].index][0]
            referencepop[12] = ds1.cases[begincasenum,ds1.varlist['GS1GS8M'].index][0]
            referencepop[13] = ds1.cases[begincasenum,ds1.varlist['GS1GS8F'].index][0]
            referencepop[14] = ds1.cases[begincasenum,ds1.varlist['GS9_SESM'].index][0]
            referencepop[15] = ds1.cases[begincasenum,ds1.varlist['GS9_SESF'].index][0]
            referencepop[16] = ds1.cases[begincasenum,ds1.varlist['OtherM'].index][0]
            referencepop[17] = ds1.cases[begincasenum,ds1.varlist['OtherF'].index][0]
#---------------------------------------------
            refsum=0.0
            for element in  referencepop:
                refsum = refsum+element
#---------------------------------------------
            e=0
            for ele in referencepop:
                if refsum !=0.0:
                    referenceratio[e] = ele/refsum
                else:
                    referenceratio[e] = 0.0
                e=e+1
#------------------------------------------------------
            for d in range(len(referencepop)):
                if respondersratio[d] !=0.0 and  referenceratio[d] != 0.0:
                    weight[d] = referenceratio[d]/respondersratio[d]
                else:
                    weight[d] = 1.0
        listb=[d1[key][0]] + weight
        weightdic[key]=listb
```

```

print(weightdic)
# Create a new dataset for each value of the variable 'unit'
with spss.DataStep():
    newds1 = spss.Dataset(name=None)
    name=newds1.name
    newds1.varlist.append('unitcasenumber',0)
    newds1.varlist.append('unitid',0)
    for v in range(18):
        templist = ['weight',v]
        templist[1] = str(templist[1])
        varname = '_'.join(templist)
        newds1.varlist.append(varname,0)

    for i, j in weightdic.items():
        templist= [ ]
        templist = [i]+j
        newds1.cases.append(templist)

spss.Submit(r"""
    DATASET ACTIVATE %(name)s.
    SAVE OUTFILE='E:\WEI WAN\My SPSS\TestFiles\weights_%(name)s.sav'.
    DATASET CLOSE ALL.
                """ %locals())
End Program.
```

Appendix E: Algorithm Component 4-2

```
output close all.

GET FILE='E:\WEI WAN\My SPSS\TestFiles\AlldatawithoutIndicator.sav'.
DATASET NAME alldata.
SORT CASES BY AFEOCAID.
compute casen = $CASENUM.
formats casen(f12.0).
VARIABLE LEVEL casen (SCALE).
EXECUTE.

BEGIN PROGRAM.
import spss

totalvar = spss.GetVariableCount()
#-works:generate a dictionary {unit numberm: [unitID, begincasenum, endcasenum]}-
#make sure you have case number--
with spss.DataStep():
    ds = spss.Dataset(name="alldata")
    i =1
    d1=dict()
    #-i: unit number--j = [deptid, beginingcasenum, endingcasenum]---make sure you have case number---
    j=[ds.cases[0,1][0], ds.cases[0,totalvar-1][0],ds.cases[0,totalvar-1][0]]
    d1={i:j}
    iteration = 0
    for r in ds.cases:
        if (r[1]!=d1[i][0]):
            d1[i][2]=d1[i][1]+ iteration-1
            i=i+1
            newj=[r[1], d1[i-1][2]+1, d1[i-1][2]+1]
            d1[i]=newj
            iteration=0
        iteration=iteration+1
    d1[i][2]=ds.cases[-1,totalvar-1][0]
#-----------------------
weightdic=dict()
for key in d1:

  weight = [1.0]*18
  with spss.DataStep():
        ds1 = spss.Dataset(name="alldata")
#---------------------------------------------------
        begincasenum=int(d1[key][1])-1
        endcasenum=int(d1[key][2])
#-----working on here-------------------------
        v11,v12,v21,v22,v31,v32,v41,v42,v51,v52,v61,v62,v71,v72,v81,v82,v101,v102=[0.0]*18
        for row in ds1.cases[begincasenum : endcasenum]:
                 if  row[ds1.varlist['Group'].index] == 1 and row[ds1.varlist['gender'].index]==1:
                     v11 = v11+1
                 if  row[ds1.varlist['Group'].index] == 1 and row[ds1.varlist['gender'].index]==2:
                     v12 = v12+1
                 if  row[ds1.varlist['Group'].index] == 2 and row[ds1.varlist['gender'].index]==1:
                     v21 = v21+1
                 if  row[ds1.varlist['Group'].index] == 2 and row[ds1.varlist['gender'].index]==2:
                     v22 = v22+1
```

```
                if  row[ds1.varlist['Group'].index] == 3 and row[ds1.varlist['gender'].index]==1:
                    v31 = v31+1
                if  row[ds1.varlist['Group'].index] == 3 and row[ds1.varlist['gender'].index]==2:
                    v32 = v32+1
                if  row[ds1.varlist['Group'].index] == 4 and row[ds1.varlist['gender'].index]==1:
                    v41 = v41+1
                if  row[ds1.varlist['Group'].index] == 4 and row[ds1.varlist['gender'].index]==2:
                    v42 = v42+1
                if  row[ds1.varlist['Group'].index] == 5 and row[ds1.varlist['gender'].index]==1:
                    v51 = v51+1
                if  row[ds1.varlist['Group'].index] == 5 and row[ds1.varlist['gender'].index]==2:
                    v52 = v52+1
                if  row[ds1.varlist['Group'].index] == 6 and row[ds1.varlist['gender'].index]==1:
                    v61 = v61+1
                if  row[ds1.varlist['Group'].index] == 6 and row[ds1.varlist['gender'].index]==2:
                    v62 = v62+1
                if  row[ds1.varlist['Group'].index] == 7 and row[ds1.varlist['gender'].index]==1:
                    v71 = v71+1
                if  row[ds1.varlist['Group'].index] == 7 and row[ds1.varlist['gender'].index]==2:
                    v72 = v72+1
                if  row[ds1.varlist['Group'].index] == 8 and row[ds1.varlist['gender'].index]==1:
                    v81 = v81+1
                if  row[ds1.varlist['Group'].index] == 8 and row[ds1.varlist['gender'].index]==2:
                    v82 = v82+1
                if  row[ds1.varlist['Group'].index] == 9 and row[ds1.varlist['gender'].index]==1:
                    v81 = v81+1
                if  row[ds1.varlist['Group'].index] == 9 and row[ds1.varlist['gender'].index]==2:
                    v82 = v82+1
                if row[ds1.varlist['Group'].index] == 10 and row[ds1.varlist['gender'].index]==1:
                    v101 = v101+1
                if row[ds1.varlist['Group'].index] == 10 and row[ds1.varlist['gender'].index]==2:
                    v102 = v102+1
        print("This is the key: ", key)
        total = float(v11+ v12+v21+v22+v31+v32+v41+v42+v51+v52+v61+v62+v71+v72+v81+v82+v101+v102)
        print("The summation of all cells in above crosstable:", total)
        respondernum=[v11, v12, v21, v22, v31, v32, v41, v42, v51, v52, v61, v62, v71, v72, v81, v82, v101, v102]
        respondersratio=[v11/total, v12/total, v21/total, v22/total, v31/total, v32/total, v41/total,
        v42/total, v51/total, v52/total, v61/total, v62/total, v71/total, v72/total, v81/total,
        v82/total, v101/total, v102/total]
        print("This is respondersratio:", respondersratio)

        referencepop=[0.0]*18;referenceratio=[0.0]*18
        rratio11 = 0.065167; rratio12 = 0.015172; rratio21 = 0.164276; rratio22 = 0.034353;
        rratio31 = 0.046183; rratio32 = 0.007292; rratio41 = 0.005294; rratio42 = 0.0014788;
        rratio51 = 0.02747; rratio52 = 0.007538;
        rratio61 = 0.026882; rratio62 = 0.005018; rratio71 = 0.0255074; rratio72 = 0.02548;
        rratio81 = 0.35537; rratio82 = 0.149931; rratio101 = 0.029049; rratio102 = 0.008539
        referenceratio=[rratio11, rratio12, rratio21, rratio22, rratio31, rratio32, rratio41, rratio42, rratio51,
        rratio52, rratio61, rratio62, rratio71, rratio72, rratio81, rratio82, rratio101, rratio102]
#-------------------------------------------------------
        for d in range(len(referencepop)):
            if respondersratio[d] !=0.0 and  referenceratio[d] != 0.0:
                weight[d] = referenceratio[d]/respondersratio[d]
            else:
```

```
                    weight[d] = 1.0
        listb=[d1[key][0]] + weight
        weightdic[key]=listb

print(weightdic)

# Create a new dataset for each value of the variable 'unit'
with spss.DataStep():
        newds1 = spss.Dataset(name=None)
        name=newds1.name
        newds1.varlist.append('unitcasenumber',0)
        newds1.varlist.append('unitid',0)
        for v in range(18):
                templist = ['Pweight',v]
                templist[1] = str(templist[1])
                varname = '_'.join(templist)
                newds1.varlist.append(varname,0)

        for i, j in weightdic.items():
              templist= [ ]
              templist = [i]+j
              newds1.cases.append(templist)


spss.Submit(r"""
        DATASET ACTIVATE %(name)s.
        SAVE OUTFILE='E:\WEI WAN\My SPSS\TestFiles\Myweights0703201701_%(name)s.sav'.
        DATASET CLOSE ALL.
                      """ %locals())
End Program.
```

Appendix F: Algorithm Component 5-1

```
output close all.

GET FILE='E:\WEI WAN\My SPSS\TestFiles\originaldatawithweightsr.sav'.
DATASET NAME weighteddata.
EXECUTE.

GET FILE='E:\WEI WAN\My SPSS\TestFiles\weights_xDataset1_unitpop.sav'.
DATASET NAME alldata.
EXECUTE.

BEGIN PROGRAM.
import spss

totalvar = spss.GetVariableCount()
with spss.DataStep():
    ds = spss.Dataset(name="alldata")
    ds1=spss.Dataset(name="weighteddata")
    ds1.varlist.append('localweight',0)

    d1=dict()
    for r in ds.cases:
        d1[int(r[1])] = r[2:totalvar]

    totalcase=len(ds1.cases)
    for m in range(totalcase):
        kkey = ds1.cases[m,ds1.varlist['AFEOCAID'].index][0]
        if ds1.cases[m,ds1.varlist['Group'].index][0] == 1 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][0]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 1 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][1]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 2 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][2]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 2 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][3]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 3 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][4]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 3 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][5]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 4 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][6]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 4 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][7]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 5 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][8]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 5 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][9]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 6 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][10]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 6 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][11]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 7 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][12]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 7 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][13]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 8 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][14]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 8 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][15]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 9 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = 1
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 9 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = 1
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 10 and ds1.cases[m,ds1.varlist['gender'].index][0] == 1:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][16]
        elif ds1.cases[m,ds1.varlist['Group'].index][0] == 10 and ds1.cases[m,ds1.varlist['gender'].index][0] == 2:
            ds1.cases[m,ds1.varlist['localweight'].index] = d1[kkey][17]
        else:
            ds1.cases[m,ds1.varlist['localweight'].index] =1

spss.Submit(r"""
        DATASET ACTIVATE weighteddata.
        SAVE OUTFILE='E:\WEI WAN\My SPSS\TestFiles\originaldatawithlocalweights.sav'.
        DATASET CLOSE ALL.
                """)
End Program.
```

Appendix G: Algorithm Component 5-2

```
output close all.

GET FILE='E:\WEI WAN\My SPSS\TestFiles\originaldatawithlocalweights.sav'.
DATASET NAME weighteddata.
EXECUTE.

GET FILE='E:\WEI WAN\My SPSS\TestFiles\Myweights0703201701_xDataset1_wholepop.sav'.
DATASET NAME alldata.
EXECUTE.

BEGIN PROGRAM.
import spss

totalvar = spss.GetVariableCount()
with spss.DataStep():
    ds = spss.Dataset(name="alldata")
    ds1=spss.Dataset(name="weighteddata")
    ds1.varlist.append('globalweight',0)

    d1=dict()
    for r in ds.cases:
        d1[int(r[1])] = r[2:totalvar]

    totalcase=len(ds1.cases)
    for m in range(totalcase):
        kkey = ds1.cases[m,ds1.varlist['AFEOCAID'].index][0]
        groupvalue  = ds1.cases[m,ds1.varlist['Group'].index][0]
        gendervalue = ds1.cases[m,ds1.varlist['gender'].index][0]
        if groupvalue  == 1 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][0]
        elif  groupvalue  == 1 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][1]
        elif  groupvalue  == 2 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][2]
        elif  groupvalue  == 2 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][3]
        elif  groupvalue  == 3 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][4]
        elif  groupvalue  == 3 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][5]
        elif  groupvalue  == 4 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][6]
        elif  groupvalue  == 4 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][7]
        elif  groupvalue  == 5 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][8]
        elif  groupvalue  == 5 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][9]
        elif  groupvalue  == 6 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][10]
        elif  groupvalue  == 6 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][11]
        elif  groupvalue  == 7 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][12]

        elif  groupvalue  == 7 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][13]
        elif  groupvalue  == 8 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][14]
        elif  groupvalue  == 8 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][15]
        elif  groupvalue  == 9 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = 1
        elif  groupvalue  == 9 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = 1
        elif  groupvalue  == 10 and gendervalue == 1:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][16]
        elif  groupvalue  == 10 and gendervalue == 2:
                ds1.cases[m,ds1.varlist['globalweight'].index] = d1[kkey][17]
        else:
                ds1.cases[m,ds1.varlist['globalweight'].index] =1

spss.Submit(r"""
        DATASET ACTIVATE weighteddata.
        SAVE OUTFILE='E:\WEI WAN\My SPSS\TestFiles\originaldatawithlocalandglobalweights.sav'.
        DATASET CLOSE ALL.
                """)
End Program.
```

Appendix H: Algorithm Component 5-3

```
output close all.

GET FILE='E:\WEI WAN\My SPSS\TestFiles\originaldatawithlocalandglobalweights.sav'.
DATASET NAME weighteddata.
EXECUTE.

GET FILE='E:\WEI WAN\My SPSS\unitfile3\NodeIDbyProb_07032017.sav'.
DATASET NAME alldata.
EXECUTE.

BEGIN PROGRAM.
import spss

totalvar = spss.GetVariableCount()

with spss.DataStep():
    ds = spss.Dataset(name="alldata")
    ds1=spss.Dataset(name="weighteddata")
    ds1.varlist.append('weightfromdecisiontree',0)

    vec=[ ]
    for r in ds.cases:
        vec.append(r[totalvar-1:totalvar][0])


    totalcase=len(ds1.cases)
    for m in range(totalcase):
        groupvalue  = ds1.cases[m,ds1.varlist['Group'].index][0]
        gendervalue = ds1.cases[m,ds1.varlist['gender'].index][0]
        if  groupvalue  == 1 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[6]
        elif  groupvalue  == 1 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[16]
        elif  groupvalue  == 2 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[1]
        elif  groupvalue  == 2 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[11]
        elif  groupvalue  == 3 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[0]
        elif  groupvalue  == 3 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[10]
        elif  groupvalue  == 4 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[7]
        elif  groupvalue  == 4 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[17]
        elif  groupvalue  == 5 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[4]
        elif  groupvalue  == 5 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[14]
        elif  groupvalue  == 6 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[5]
        elif  groupvalue  == 6 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[15]
        elif  groupvalue  == 7 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[2]
        elif  groupvalue  == 7 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[12]
        elif  groupvalue  == 8 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[8]
        elif  groupvalue  == 8 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[18]
        elif  groupvalue  == 9 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = 1
        elif  groupvalue  == 9 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = 1
        elif  groupvalue  == 10 and gendervalue == 1:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[3]
        elif  groupvalue  == 10 and gendervalue == 2:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] = vec[13]
        else:
                    ds1.cases[m,ds1.varlist['weightfromdecisiontree'].index] =1

spss.Submit(r"""
        DATASET ACTIVATE weighteddata.
        SAVE OUTFILE='E:\WEI WAN\My SPSS\TestFiles\
                            originaldatawithlocalandglobalanddecisiontreeweights.sav'.
        DATASET CLOSE ALL.
                    """)
End Program.
```