MICHIGAN JUSTICE STATISTICS CENTER
SCHOOL OF CRIMINAL JUSTICE
MICHIGAN STATE UNIVERSITY

DEC.
2016

CLEANING AND ANALYZING MICHIGAN INCIDENT CRIME REPORTING DATA (MICR) IN THE R STATISTICAL COMPUTING ENVIRONMENT

MICHIGAN STATE
UNIVERSITY

Cleaning and Analyzing Michigan Incident Crime Reporting Data (MICR) in the R Statistical Computing Environment

Jason Rydberg, Ph.D.

School of Criminology and Justice Studies

Center for Program Evaluation

University of Massachusetts Lowell

Michigan Justice Statistics Center

Michigan State University

December, 2016

Michigan Justice Statistics Center
The School of Criminal Justice at Michigan State University, through the Michigan Justice Statistics Center, serves as the Statistical Analysis Center (MI-SAC) for the State of Michigan. The mission of the Center is to advance knowledge about crime and justice issues in the state of Michigan while also informing policy and practice. The Center works in partnership with the Michigan State Police, Michigan's State Administering Agency (SAA), as well as with law enforcement and criminal justice agencies serving the citizens of Michigan. For further information please visit the Michigan Justice Statistics Center.

About the Author
Jason Rydberg is an assistant professor in the School of Criminology and Justice Studies at the University of Massachusetts Lowell, where he is also an associate with the Center for Program Evaluation. Jason received his PhD from the School of Criminal Justice at Michigan State University in 2014. His research interests include sex offender policy, prisoner reentry and recidivism, the evaluation of criminal justice programs, and quantitative methods. His research has recently been featured in the *Journal of Quantitative Criminology*, the *Journal of Criminal Justice*, *Criminology and Public Policy*, and *Justice Research and Policy*.

Author Contact Information
Jason Rydberg, Ph.D.
Assistant Professor, School of Criminology and Justice Studies
Associate, Center for Program Evaluation
University of Massachusetts Lowell
113 Wilder Street
Lowell, MA 01854
E: Jason_Rydberg@uml.edu

Acknowledgements
This guide is inspired by the Introduction to R Workshop given by Dr. Dave Armstrong at the ICPSR Summer Program. It also draws on materials developed by doctoral students at the University of Massachusetts Lowell - Scott Walfield and Kyleigh Clark.

Last compiled in LaTeX Friday 2nd December, 2016

Contents

1 Preface

This guide has been developed to support researchers interested in working with incident-based crime data within the **R** statistical computing environment. Specifically, this tutorial focuses on the Michigan Incident Crime Reporting (MICR) system, but the practices described here could be applied to the National Incident-Based Reporting System (NIBRS) more broadly. Historically, criminological research has been heavily influenced by summary-based reporting systems, particularly the Uniform Crime Reports (UCR).[1] As data only reflecting crimes reported to the police, the limitations associated with these data are well known among researchers and students of criminology alike.[2] Because the NIBRS program data is generated in a similar fashion - as crimes reported to the police - it does not solve many of the dilemmas posed by UCR data. However, as an incident-based reporting system, NIBRS is able to make major improvements on the UCR program as it related to the unit of analysis.

As a summary-based system, the UCR program reports data utilizing the reporting agency-year as the unit of analysis. That is, a researcher accessing these data would be capable of describing the number of specific violent or property offenses reported by a given law enforcement agency in a given year. This data structure still has wide utility in criminological research, particularly for describing variation in long-term crime trends. However, in many respects this data structure is limiting. For instance, if a researcher were interested in describing variation in intimate partner violence across jurisdictions, the UCR data are not capable of differentiating aggravated assaults or homicides committed by intimate partners from any other aggravated assault or homicide. In the late 1980s, the U.S. Department of Justice began preparations for a new data collection system that would utilize incident-level data. This system eventually became what we known today as NIBRS.[3] The NIBRS system remedies the unit of analysis limitations associated with summary data by reporting data at multiple units of analysis nested within crime *incidents*. That is, within each incident participating agencies report data concerning the characteristics of the *reporting agency*, the *victim(s)* involved, the *offenders* involved, any *arresttees*, and details concerning *property* lost, recovered, or seized. Each of these nested units are detailed within data files known as *segments*.

The additional units of analysis and variables provided by NIBRS allow for tremendous

[1]See Maxfield, M. (1999). The National Incident-Based Reporting System: Research and Policy Implications. *Journal of Quantitative Criminology 15*(2), 119-149.

[2]See Loftin, C., & McDowall, D. (2010). The use of official records to measure crime and delinquency. *Journal of Quantitative Criminology 26*(4), 527-532.

[3]See Addington, L. (2008). Assessing the extent of Nonresponse Bias on NIBRS estimates of violent crime. *Journal of Contemporary Criminal Justice 24*(1), 32-49.

flexibility in addressing a variety of research questions. However, this feature of NIBRS also presents a double-edged sword. The complex file structure of NIBRS does not easily lend itself to analysis. For example, within a given incident there may be multiple victims who may or may not be associated with multiple offenders who may or may not have committed a variety of offenses against said victims. To this extent, building data files from NIBRS segments that align with the principles of *tidy data* can be difficult. Tidy data requires that each row of a data file will represent a unique unit (e.g., a person, a person observed at a particular time point, an agency, etc.), and that each variable describing those units will make up a column of the data set.[4]. This guide will briefly introduce some of the tools within the **R** environment that can help facilitate the cleaning and statistical analyses of these incident-based data.

[4]See Wickham, H. (2014). Tidy data. *Journal of Statistical Software 59*(10), 1-23

2 A Gentle Introduction to R

2.1 Getting Started with R

R is an open source statistical computing enviroment. This means that it is free to download and use, compared to other environments - such as SPSS, SAS, and Stata - for which licenses must be purchased. **R** can be downloaded from http://www.r-project.org/. It will likely be helpful to download **RStudio** as well, which is a free-to-use interface for **R**.[1] **RStudio** is available for download from http://www.rstudio.com/. **RStudio** provides an interface that is more similar to Stata, which users old and new are likely to find more informative and user friendly than the base **R** interface. As opposed to being a point-and-click interface, both **R*** and **RStudio** are command line based, meaning that they are controlled through scripted commands entered into a prompt.

In regards to what constitutes **R**, specifically, it is an implementation of the **S** statistical programming language developed at Bell Labs. Because **R** is open source, this means that it is open to the community to develop new techniques and tools to be used in the environment. Fortunately, the **R** community is large, active, and growing - which means that when new statistical techniques are developed, they are often implemented in **R** before they are incorporated into SPSS and the like. Additionally, the user community continually develops new tools designed to make data cleaning, manipulation, and graphics easier and more powerful. Hadley Wickham's `dplyr` and John Fox's `car` packages being excellent examples that will be utilized in this guide

2.1.1 Installing R

Installing **R** on your computer is free and *relatively* simple. The following will help you install **R** on a Windows computer. The procedure is likely somewhat different for installation on a Mac.

1. **R** is available from http://www.r-project.org/. Navigate to this URL.

2. On the homepage, there is a link for CRAN on the left under the heading "Downloads, Packages".

3. A list of servers to download **R** from should appear. Choose your favorite, or at random - they all produce the same result.

[1]In order for **RStudio** to work properly, **R** must already be installed.

4. Choose the link for your operating system.
5. If this is your first time installing **R**, click the "base" link, and then the link to download the most recent version of the program. You should now have downloaded an .exe file to setup **R**.
6. Execute the file (i.e., double-click, or click "Run" if prompted). You should now be well on your way to having **R** installed on your machine (...Bam!).
7. An optional, but highly recommended final step is to download and install **RStudio**. It is available from http://www.rstudio.com/ and will only function once **R** is already installed.

2.1.2 Installing and Loading R Packages

When a statistical computing environment such as SPSS or Stata is installed, it already contains all (or most) of the tools and techniques at its disposal. When **R** is installed, it contains some basic capabilities, but user-written statistical tools and techniques (referred to as *packages*), must be downloaded and installed. **R** packages are available from CRAN (Comprehensive R Archive Network) and can be installed by using the `install.packages` function. For instance, the command `install.packages("car", dependencies = TRUE)` will install the `car` package (standing for 'Companion to Applied Regression'), as well as any packages that `car` requires to function. Packages only need to be installed on a computer once. As with installing the program, you may be prompted to select a CRAN mirror the first time you are installing packages - any one will do. There are currently thousands packages for free on CRAN, all of which are tested before they are accepted into the CRAN repository.

After a package has been installed, it must be loaded by utilizing the `library` function. For instance, the command `library(car)` will load the `car` package. Entering `search()` will display all of the packages that are loaded for the current session. Note that if multiple packages are loaded which share a common function (e.g., `car` and `psych` each contain a function called `logit()`), the package loaded last will be utilized.

2.1.3 Setting up R Script (.R Files)

The **R** and **RStudio** interfaces allow for two methods of entering commands. There is manual command entry, where commands are entered directly into the console, and there is also the possibility of running commands via a .R script. Script is conceptually similar to a SPSS syntax file, or a Stata .do file. As a general piece of advice, you should primarily enter commands via a script since this maintains a record of the work that you have done - which aids reproducibility and enables you to quickly and easily identify and correct errors without significantly altering your original data file.

For any given project, a .R script will generally begin with three sets of commands. First, loading necessary packages via the `library()` function. For instance, the following will load the `car` and `dplyr` packages.[2]

[2]The semi-colon ; is the equivalent of hitting Enter, and tells **R** to proceed to the next command as if it were a new line of code.

```
> library(car) ; library(dplyr)
```

Second, setting the working directory. This is the file path directory that **R** will look for files and save output. It makes sense to create a new directory for any given project/analysis. Entering ‘getwd()‘ will display the current working directory. The ‘setwd()‘ function sets the working directory to the desired filepath. For instance, the following command would set the working directory to my Dropbox cloud storage.

```
> setwd("C:/Users/Jason/Dropbox/R")
```

Note that **R** uses forward slashes "/" in the file path, as opposed to the Windows default, which is the back slash. Third, options allows you to change some of the defaults in R (https://stat.ethz.ch/R-manual/R-devel/library/base/html/options.html). Uncomfortable working with scientific notation? You can start off your script with a command to change that option:

```
> options(scipen = 999)
```

2.2 R as an Object-Oriented Programming Language

R commands can be entered directly into the console, or run through script. When a command is entered either way, it is printed to the console. For instance, we can use **R** like a calculator.

```
> 5 + 5
[1] 10
```

Here, we told **R** to add `5 + 5`, and it gave us the output of `10`. If we wanted to get this result again, we would be required to enter `5 + 5` again. **R** is an *object-oriented* programming language, in that it is possible to save the results of our commands to named objects. For instance, rather than simply printing the previous command to the console, let's save it to an object called `x`.

```
> x = 5 + 5
```

Now, `x` contains the result of our command. If we were to enter `x` as a command, it will print the result to the console.[3]

```
> x
[1] 10
```

This feature of the **R** language may seem simple, but it turns out to be quite powerful and important. For instance, we could now use this object in other commands:

[3]Note that simply entering the name of an object (here it is `x`) into the command line is the equivalent of the command `print(x)`.

```
> x + 2 # Addition (+)
[1] 12
> x / 2 # Division (/)
[1] 5
> x * 2 # Multiplication (*)
[1] 20
> x + x # Adding two objects
[1] 20
> x ^2 # Exponents (^)
[1] 100
```

2.3 Loading and Manipulating Data

In the previous example, we introduced 'objects' to demonstrate that we can store output. In the example above, we stored a single value in an object called `x`. This opens a path to reading and importing datasets into **R**. To load an existing dataset into **R**, we can save it to a special kind of object called a `data.frame`. So, how does one get their data into a data frame? There are a variety of functions for reading data into **R**, and it is capable of reading numerous types of data files. There are several functions for reading types of data in the base **R** installation (e.g., `read.table` for reading text files, and `read.csv` for reading comma-separated values spreadsheets). In order to read files built by other statistical programs, such as SPSS or Stata, the `foreign` package will be necessary.

```
> install.packages("foreign", dependencies = TRUE) # Install the package
```

```
> library(foreign) # Load the package
```

`foreign` contains functions for reading numerous different data types, but for this example we will focus on reading SPSS data into **R**. This is accomplished using the `read.spss` function, which has several arguments with default values.[4]

```
> args(read.spss)
function (file, use.value.labels = TRUE, to.data.frame = FALSE,
    max.value.labels = Inf, trim.factor.names = FALSE, trim_values = TRUE,
    reencode = NA, use.missings = to.data.frame)
NULL
```

[4] A function with an argument with a default value will simply use that argument if it is not explicitly called. If you decide that the default value is not that you want, you will need to specify that argument in the function call.

Typing `help(read.spss)` will give additional details on these arguments and their possible values. The first argument asks for the file name. If the data file is not in your working directory, then you will need to specify the entire file path. If the file is in your working directory then all you will need is the file name. Other arguments take logical values, such as `TRUE` if you would like to switch that argument on. For instance, setting ‘to.data.frame = TRUE‘ will return a data frame when we call the function (instead of a list object, which is the default setting). Additionally, it was noted earlier that SPSS often applies value labels to the levels of factors. We would like to use those value labels as actual for our dataset, otherwise we will not know what the numbers stand for. We will set `use.value.labels` to `TRUE`. In this case, I will use the 1998 Pennsylvania Commission on Sentencing dataset (ICPSR 3450) as an example. These data are publicly available, describing characteristics of individuals convicted in Pennsylvania criminal courts.

```
> pcs = read.spss("PCS 1998 Data.sav", to.data.frame = TRUE,
                 use.missings = TRUE, use.value.labels = TRUE)
> names(pcs) # Lists all variable names within the data frame
 [1] "ID"           "CID"          "DOB"
 [4] "DOS"          "DOSAGE"       "SEX"
 [7] "RACE"         "COUNTY"       "PCSOFF"
[10] "OFFLABEL"     "GRAD"         "OGS"
[13] "PRS"          "MANMIN"       "DISP"
[16] "INCSTR"       "INCTYPE"      "INCMIN"
[19] "INCMAX"       "CONFORM"      "INCARCERATED"
```

When working with datasets we save them to objects. In this case, we have saved our dataset to a `data.frame` object called `pcs`. There are several functions that are helpful for exploring the contents of a `data.frame`. The ‘str‘ function tells us the structure of an object, including it’s class (this one is a ‘data.frame‘), the number of observations (units) and variables, as well as the name and class of each variable. More on classes coming up in the next section.

```
> summary(pcs) # Descriptive statistics on every variable
> str(pcs) # Dataset and variable structure
> View(pcs) # View the first 100 columns of the data as a spreadsheet
> utils::View(pcs) # Creates a popup window with the entire spreadsheet
```

There are numerous online tutorials for reading data into **R**, in case this example does not demonstrate the necessary information. Helpful guides are maintained by CRAN, Quick-R, and UCLA. For other common forms of data, `read.dta` is a function in the ‘foreign‘ package which reads Stata files into **R**.[5] In essence, to read a type of data file into **R**, you will need to match the correct function to the file type. Data also commonly comes in Excel form (.xls or .csv); R works best with the latter and does not require the aforementioned package. To read a CSV file, we would use the following: `read.csv("mydata.csv", header=TRUE)`. When in doubt, Google ”how to open [.extension] files in R”.

[5]This raises an issue with the open source nature of **R**. The `read.dta` function has not be updated recently, and can only open Stata files from version 12 or earlier. Fortunately, some intrepid programmers have written a new package ‘readstata13‘, which can open more recent versions of Stata files.

2.4 Working with Data Frame Variables

Data frames are comprised of observations of variables. In order to access individual variables for analysis, the most common option is to use the `$` operator. For instance, `pcs` is the data frame. `SEX` is a variable in the `pcs` data frame. In order to access the `SEX` variable, it is formally `pcs$SEX`.

```
> head(pcs$SEX)
[1] Male    Male    Male    Male    Male    Male
Levels: Female  Male
```

R data frames store different sorts of variables. The different kinds of variables are called ”classes”, which *roughly* correspond to the level of measurement of the variable. Using the above example, `SEX` is a vector of class `factor`, or nominally measured data. For instance,

```
> class(pcs$SEX)
[1] "factor"
```

Alternatively, we could use the `str` function on a variable which combines the above two. This also alerts us to a potential issue for later, but we will cross that bridge when we get to it.

```
> str(pcs$SEX)
 Factor w/ 2 levels "Female ","Male   ": 2 2 2 2 2 2 2 2 1 2 ...
```

In **R**, `factor` variables represent categorical, or nominal data. Factors have different values called `levels`. In this case, there are 2 levels to `SEX` (`FEMALE` and `MALE`). These have no intrinsic quantitative meaning. Unlike in SPSS or Stata, where it is common to give factor variables numeric values, but then label those numeric values, in **R** it is more appropriate to use the labels themselves as the data entered so that we know that the variable is categorical, and each category has a meaning.[6] On the other hand, variables of the `numeric` class contain quantitative data. For instance, the variable `DOSAGE` contains the defendant’s age at sentencing.

```
> head(pcs$DOSAGE)
[1] 39 28 34 18 19 37
> class(pcs$DOSAGE)
[1] "numeric"
```

There are several other common classes of data, which will be covered to varying extents later on. `character` variables contain character strings, `integer` variables contain whole

[6]It is also worth noting that factor variables have underlying numeric values. In this case, `FEMALE` is equal to `1`, and `MALE` is equal to `2`.

number values and are essentially equivalent to `numeric` variables for all intents and purposes. `logical` variables contain values of `TRUE` and `FALSE`, which can be used to represent dichotomous variables, among other things.

We can perform functions on variables. For instance, we can gather simple descriptive statistics. The `mean` function will produce the mean, and the `sd` function will produce the standard deviation. When we use `summary` on a numeric variable, it produces the five number summary, plus the mean.

```
> mean(pcs$DOSAGE) #the mean - use help(mean) for additional arguments
[1] 32.38741
> sd(pcs$DOSAGE) #the standard deviation
[1] 12.86041
> summary(pcs$DOSAGE) #the five number summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00   23.00   30.00   32.39   38.00   99.00 
> mean(pcs$SEX) #not surprisingly, the mean doesn't work on a factor
[1] NA
```

2.5 Data Management and Manipulation in R

This section of the presentation concerns basic data management and manipulation techniques using **R**. As it turns out, **R** is a very powerful tool for data management, and proficient users are able to accomplish complicated tasks in a short period of time.

2.5.1 Subsetting (Indexing) Data

So far, we have been working with the entire `pcs` file. Given the nature of our research questions, we may be in a position where we are actually not interested in working with the entire dataset, and instead only want to work with a selection (or subset) of those data. In **R** there are several means of subsetting data frames.

The first means is to use what I will call *bracketing.* To do this, we have to understand something about matrices. A matrix is a rectangular array of numbers, which can be described by the number of its rows and columns (i.e., it's *dimensions*). A data frame in **R** is essentially a matrix, which can contain all sorts of information (not just numbers). Let's take a look at the dimensions of our data frame.

```
> dim(pcs)
[1] 48881    21
```

When we take a look at the dimensions of `pcs`, we see that it is a data frame with 48881 rows and 21 columns. As we have been working with variables by using the `$` operator, we can actually work with variables by referring to their column placement. In `R`, this is accomplished by using the square brackets `[ , ]`. Inside the brackets, the information on the left side of the comma refers to the rows, and the information on the right side of the comma refers to the columns.

We can use brackets (i.e., `object[ , ]`) to identify the subset of rows or variables that we would like. For instance,

```
> pcs[1, 1] #This returns the value in the first row and the first column
[1] 1
> mean(pcs[, 5]) #Takes the mean of the 5th column, which is DOSAGE
[1] 32.38741
> #This creates a new data frame our of the first 10 rows and first 5 variables
> pcs.sub = pcs[1:10, 1:5]
> pcs.sub
   ID CID        DOB        DOS DOSAGE
1   1  NA 04/13/1959 06/02/1998     39
2   2  NA 10/02/1969 04/22/1998     28
3   3  NA 06/18/1964 09/14/1998     34
4   4  NA 06/22/1980 12/29/1998     18
5   5  NA 02/06/1979 07/09/1998     19
6   6  NA 07/08/1961 10/26/1998     37
7   7  NA 08/05/1973 04/06/1998     24
8   8  NA 03/28/1977 09/21/1998     21
9   9  NA 01/09/1979 09/28/1998     19
10 10  NA 12/03/1951 09/08/1998     46
```

Here, we saved a small number of observations and variables to a new data.frame object, `pcs.sub`. We can also use `c()` (concatenate) to be very specific about the rows and variables we want. For instance,

```
> pcs.sub = pcs[c(2, 4, 6, 8, 10), c(2, 8, 10)]
> pcs.sub
   CID       COUNTY
2   NA Philadelphia
4   NA Armstrong
6   NA Bucks
8   NA Allegheny
10  NA Beaver
                                              OFFLABEL
2  Retaliate Against Witness/Victim
4
6  Simple Assault
8  Possession:Small Amt of Marij(30g marij or 8g hash
10 Simple Assault
> pcs.sub = pcs[c(1:99, 200:299), ] #keeps all variables
> pcs.sub = pcs[ , c(1:10)] #All rows, but first 100 variables
```

These forms of subscripting can be helpful if particular row or column numbers are meaningful. But it is more likely that we will want to select rows that have particular values of particular variables. For instance, only cases that were sentenced to a period of incarceration. We can use brackets to achieve this as well.

```
> table(pcs$INCARCERATED)
    0     1
22098 26783
> pcs.sub = pcs[pcs$INCARCERATED == 1, ]
> table(pcs.sub$INCARCERATED)
    1
26783
```

In the above example, we created a subset of the original data frame, selecting only the rows in which `INCARCERATED` is equal to `1`. Notice that in the above example I used the `==` operator. This is a logical operator in **R** which means "is equal to". There are a number of logical operators that can help us select and subset data.

- `<` less than
- `<=` less than or equal to
- `>` greater than
- `>=` greater than or equal to
- `==` exactly equal to
- `!=` does not equal
- `x!` not x
- `x | y` x OR y
- `x & y` x AND y

For instance, let's create a dataset that is comprised of just those sentenced to incarceration, and with no prior record (a PRS of zero).

```
> pcs.sub = pcs[pcs$INCARCERATED == 1 & pcs$PRS == 0, ]
```

As you may have noticed, it can be a little cumbersome to continually call for the name of the data frame (`pcs$`) when subsetting, particularly if you have a rather complicated set of conditions that you are specifying. One way around this is to use the dedicated `subset` function. It works similarly to bracketing, but only requires calling for the data frame once. This time, we will call for any prior record score lower than 3.

```
> pcs.sub = subset(pcs, INCARCERATED == 1 & PRS < 3)
```

2.5.2 Creating and Recoding Variables

In **R**, although we can look at our dataset using `View`, there is no intuitive spreadsheet-like data editor, such as in Excel or SPSS. Instead, what we will do is create variables as separate objects, which we can then append to our dataset. To demonstrate, let's say that we would like to create a transformation of our age variable, for instance, taking its square root (accomplished with the `sqrt()` function). Let's store this in an object called `sqrt.age`:

```
> sqrt.age = sqrt(pcs$DOSAGE)
> head(sqrt.age)
[1] 6.244998 5.291503 5.830952 4.242641 4.358899 6.082763
> summary(sqrt.age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   4.796   5.477   5.599   6.164   9.950
> length(sqrt.age)
[1] 48881
```

This object exists on its own, separate from our data frame. There are a few ways that we can add it to the `pcs` data frame. First, we can bind the column to `pcs` using the `cbind()` function (or "column bind"). This will append the `sqrt.age` object to the data frame. Notice that we overwrite `pcs` by doing this.

```
> pcs = cbind(pcs, sqrt.age)
> summary(pcs$sqrt.age) # Recognizes the new variable
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   4.796   5.477   5.599   6.164   9.950
```

Alternatively, we can initially create the variable as already being part of the data frame. This time, standardize age and add it to the `pcs` data frame (using the function `scale`). We will call this variable `z.age`. Notice that at this moment there is no variable called `z.age` in the dataset. The code below will create a this new variable by specifying `pcs$z.age`:

```
> pcs$z.age = scale(pcs$DOSAGE)
> summary(pcs$z.age)
       V1
 Min.   :-2.5184
 1st Qu.:-0.7299
 Median :-0.1856
 Mean   : 0.0000
 3rd Qu.: 0.4364
 Max.   : 5.1797
```

Because we specified a variable in `pcs` that did not yet exist, it created a new variable. If you give it the name if an existing variable, **R** will overwrite it (and will not warn you about it!). So be careful. So far we have created new variables that are statistical transformations of others. How about changing the values of an existing variable? Recoding can be accomplished through a variety of means. The first is by use of bracketing. Let's take a look at the variable 'INCMIN', which measures the minimum sentence length in months.

```
> summary(pcs$INCMIN)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     2.0    36.0   481.9  1000.0  1000.0
```

Wow, this is *really* skewed. In fact, that maximum value of 1,000 months (i.e., 83 years) looks a little suspicious (check them out using `View`, they are actually `999.9999`). Let's see how many of those there are. Here we will use the function `length` to count the number of observations `which` meet a specified criteria.

```
> length(which(pcs$INCMIN == 999.9999)) #A function within a function!
[1] 23364
```

Ok, so it looks like those values of `999.9999` are some sort of missing value, but **R** is not recognizing them as such. Let's recode `INCMIN` to change all of those values to missing. We can accomplish this by use of the brackets. Because we are changing the values of `pcs$INCMIN`, there is no column to specify (because it is a single variable, by definition it is a single column of data).

```
> pcs$INCMIN[pcs$INCMIN == 999.9999] = NA
> summary(pcs$INCMIN)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.000   0.490   3.000   7.459   9.000 300.000   23364
```

What this code translates to is "For the variable `INCMIN`, take all the rows for `INCMIN` that are equal to `999.9999` and change them to `NA`." This will change the value to the **R** missing data class. If you added quotation marks around the `NA` (i.e., `"NA"`), this would change the class to `character`. We can also change values from `numeric` to `factor`, or vice versa. For instance, right now the variable `INCARCERATED` is coded as numeric, where `0` presumably means "No" and `1` presumably means "Yes". Let's make this explicit. There are (at least) two ways to accomplish this. First, we will use brackets. Let's save it to a new variable called `incarc.fac` just to be safe. We will start by creating a blank variable that is comprised only of `NA` values, and then we will copy over those values. We will then convert all of the `‘1‘` values to `Yes`, and `0` to `‘No‘`.

```
> pcs$incarc.fac = NA
> pcs$incarc.fac[pcs$INCARCERATED == 1] = 'Yes'
> pcs$incarc.fac[pcs$INCARCERATED == 0] = 'No'
> table(pcs$incarc.fac)
   No   Yes
22098 26783
```

```
> str(pcs$incarc.fac)
 chr [1:48881] "No" "No" "No" "No" "No" ...
```

Alright, so we have our new variable, but notice that it is a `character` variable, and not a `factor`. To finish things off, we can use the `factor` function to convert the variable to a factor, specifying it's levels and setting the reference category. The first value of the `levels` argument will always be the reference category.

```
> pcs$incarc.fac = factor(pcs$incarc.fac, levels = c('No', 'Yes'))
> table(pcs$incarc.fac)
   No   Yes
22098 26783
> str(pcs$incarc.fac)
 Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Note, We can also use the `within` function to avoid having to call for the data frame over and over.

```
> pcs = within(pcs, {
   incarc.fac = NA
   incarc.fac[INCARCERATED == 1] = 'Yes'
   incarc.fac[INCARCERATED == 0] = 'No'
   incarc.fac = factor(incarc.fac, levels = c('No', 'Yes'))
 })
> table(pcs$incarc.fac)
   No   Yes
22098 26783
> str(pcs$incarc.fac)
 Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

The second way to recode variables from numeric to factor, or even from factor to factor is to use the `recode` function, which is part of the `car` package (`help(recode)`). The function works similarly to the recode function in SPSS. We can specify values or groups of values to become other values. To demonstrate, let's convert the `RACE` variable to one with fewer categories. This will also give us a chance for a teaching moment concerning character strings in **R**. First, let's call up a frequency table for `RACE` to see its values, and then use the `recode` function to create a `race.recode` variable. Notice that it has several arguments that we can specify.

```
> table(pcs$RACE, useNA = "ifany")
Am Indi Asian   Black   Hispani Other   Unknown White
     22     164   13752    2756     115    1809   30263
```

```
> library(car)
> pcs$race.recode = recode(pcs$RACE, "'White' = 'White';
                                       'Black' = 'Black'; 'Hispani' = 'Hispanic';
                                       c('Am Indi', 'Asian', 'Other') = 'Other';
                                       else = NA",
                            as.factor.result = TRUE,
                            levels = c('White', 'Black', 'Hispanic', 'Other'))
> table(pcs$race.recode, useNA = "ifany")
   White    Black Hispanic    Other     <NA>
       0        0     2756       22    46103
```

Whoa, something isn't right. Let's get to the bottom of this.

```
> str(pcs$RACE) #Notice the trailing white space following "Asian  "
 Factor w/ 7 levels "Am Indi","Asian  ",..: 3 4 7 7 7 3 7 3 7 3 ...
```

This is a common issue with datasets maintained in other platforms, as trailing or leading white space may exist but goes unnoticed. We will use a handy function in the `stringr` package called `str_trim` to trim all the leading and trailing white space from `RACE`.

```
> #install.packages("stringr", dependencies = TRUE)
> library(stringr)
> pcs$RACE = str_trim(pcs$RACE)
> pcs$race.recode = recode(pcs$RACE, "'White' = 'White';
                                       'Black' = 'Black'; 'Hispani' = 'Hispanic';
                                       c('Am Indi', 'Asian', 'Other') = 'Other';
                                       else = NA",
                            as.factor.result = TRUE,
                            levels = c('White', 'Black', 'Hispanic', 'Other'))
> table(pcs$race.recode, useNA = "ifany") #That's more like it!
   White    Black Hispanic    Other     <NA>
   30263    13752     2756      301     1809
```

2.5.3 Arranging and Merging

Data frames can be arranged by values of a single variable or multiple variables. In the base installation, sorting is not very intuitive. For instance, we could order the data by the name of sentencing county using brackets and the `order` function.

```
> pcs = pcs[order(pcs$COUNTY), ]
> head(pcs$COUNTY)
[1] Adams        Adams        Adams        Adams
[5] Adams        Adams
67 Levels: Adams        Allegheny    ... York
```

Fortunately, the `dplyr` package has a variety of functions for manipulating data frames. This is accomplished with the `arrange` function. Lets order the data in ascending order by `COUNTY` and in descending order by offense seriousness and then ascending by offender age.

```
> #install.packages("dplyr", dependencies = TRUE)
> library(dplyr)
> pcs = arrange(pcs, COUNTY, -OGS, DOSAGE)
```

Now that we have arranged the data by the sentencing county, perhaps it would be useful to merge some county-level data onto our dataset. This can be accomplished by merging data frames. First, let's read in a new data frame containing the county data. It is a .csv file called "PA County Data", so we will use the `read.csv` function to read it in.

```
> county = read.csv("PA County Data.csv")
> str(county$COUNTY) #No trailing space
 Factor w/ 67 levels "Adams","Allegheny",..: 1 2 3 4 5 6 7 8 9 10 ...
```

To make sure the merging works properly, let's use `str_trim` on the `pcs$COUNTY` variable. That way we have equivalent values across the datasets. We will use the `merge` function to merge the two data frames. We specify the name of the matching variable using 'by ='.[7] The `all.x` argument specifies that we want the new data frame to keep all the cases in `pcs`, even if there are not matches found in the second dataset. We could then also calculate the violent crime rate by dividing the crimes by the total population.

```
> pcs$COUNTY = str_trim(pcs$COUNTY)
> pcs.county = merge(pcs, county, by = "COUNTY", all.x = TRUE)
> pcs.county$vio.rate = with(pcs.county, (violent.crime / total.pop)*100000)
> head(pcs.county[, c('COUNTY', 'vio.rate')])
  COUNTY vio.rate
1  Adams 96.39399
2  Adams 96.39399
3  Adams 96.39399
4  Adams 96.39399
5  Adams 96.39399
6  Adams 96.39399
```

2.6 Moving Forward . . .

This chapter of the tutorial covered some basic **R** functionality necessary for working with the MICR data. Other tools will be necessary to reach the final product. The following sections will describe the R script that cleans and merges the various MICR files. When novel techniques are utilized in this process, they will be given a brief description. In the (inevitable) event that the author forgets to describe one of these functions, please do not hesitate to use the `help` function, or search for other examples of the function in action online.

[7] You can merge on multiple variables using the `c()` function to specify the multiple matches, for instance `by = c(COUNTY, YEAR)`.

3 Working with MICR in R

3.1 Understanding the MICR File Structure

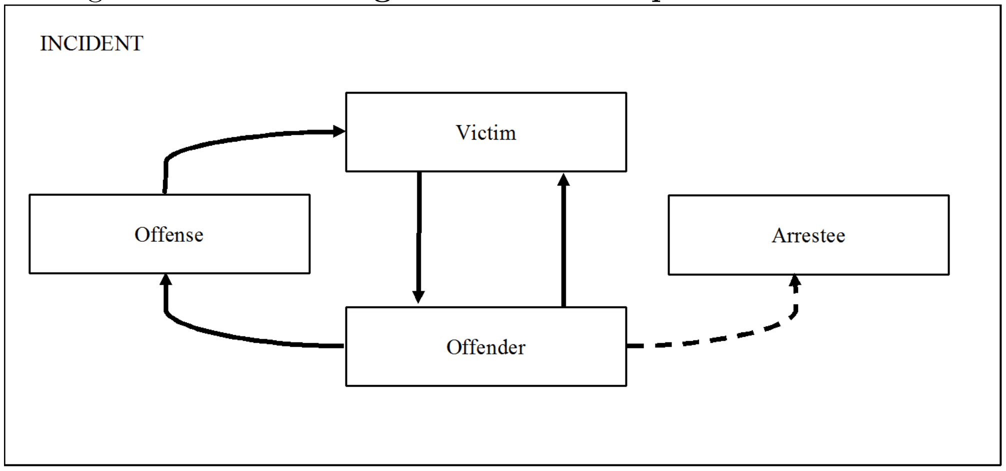
As opposed to summary-based reporting systems, such as the Uniform Crime Reports (UCR), incident-based reporting systems have a considerably more complex file structure. This complexity derives from the nested relationships for the various parties involved in each incident (e.g., victim, offender), the various outcomes or processes involved with each incident (e.g., arrests of offenders, offense committed, weapons/items used), and the contexts in which the incidents take place (e.g., incidents involving multiple victims/offenders/offenses occurring at a particular location in space and time). As such, incident-based reporting systems such as the National Incident-Based Reporting System (NIBRS) are maintained at the National Archive of Criminal Justice Data (NACJD) as multiple files, where each one represents a separate unit of analysis (e.g., victim-level file, offender-level file, etc.). MICR is characterized by a similar complex file structure. When MICR is extracted and exported for analysis by external researchers, it is divided up into numerous comma-separated values files. Specifically, the MICR download is packaged into relevant file segments including:

Table 3.1: MICR File Segments

File Name	File Description	Unit of Analysis
MICR 1	Administrative Segment	Incident
MICR 3	Offender Segment	Offender
MICR 5	Victim Segment	Victim
MICR 7	Arrestee Segment	Offender (Arrested)
MICR 9	Property Segment	Incident
MICR ADDRESS	Address Segment	Incident
MICR ARMED	Arrestee Armed	Offender (Arrested)
MICR ATYPE	Offense Activity Type	Offense
MICR LEOKA	Law Enforcement Officer Killed or Assaulted	Victim (LE Officer)
MICR NDRUG	Drugs Recovered	Incident
MICR NPROP	Property Lost / Recovered	Incident
MICR OFFNS	Offense Segment	Offense
MICR OUSED	Drugs/Equipement Used	Offense
MICR VOFNS	Victim-Offense Segment	Victim-Offense Dyads
MICR VOR	Victim-Offender Relationship Segment	Victim-Offender Dyads

These various files describe different units of analysis nested within incidents. Within each incident, different units of analysis may be linked to one another on the basis of various transactions. For instance, Figure 2.1 highlights how victims, offenders, offenses, and arrestees may be connected in a given incident. In this case, each victim may have been offended against by a particular offender, who would have committed one or more offenses against the victim(s). These offender(s) may have subsequently been arrested. Clearly, not all incidents will represent such a complex web of relationships. For instance, in an incident in which the offense is ”obstructing justice”, there may only be an offender present, and all other segments will be null. Representing these data in a tidy format - one in which each row represents a specific unit of analysis and each column represents an observation of those units - would be impossible without utilizing multiuple active datasets. And even at this point these multiple datasets would still need to be linked in order to conduct a variety of analyses.

Figure 3.1: **Visualizing the MICR Complex File Structure**



One alternative to maintaining separate datasets for each unit is to create a dataset in which each row represents a relationship between units. For instance, each row could be offense-based, representing a victim-offender transaction. Producing such a dataset requires careful management of identifiers for units within each dataset. Specifically, as it is exported, MICR utilizes a unique identifier for each incident, known as the `MIC1_NUMBER`. Even though the multiple file segments in Table 2.1 detail different units of analysis, most of them also contain the `MIC1_NUMBER` for the incident to which they correspond. The following section describes a strategy for linking the MICR files by combining the `MIC1_NUMBER` with other variables from these files to create a system of common identifiers.

3.2 Linking the MICR Files

The following describes a specific strategy for linking several MICR files in order to produce an offense-based dataset, in which each row represents an offense against a victim, committed by a given offender (i.e., a victim-offender-offense triad). This process is facilitated by creating common identifiers across multiple datasets which can then be used for merging purposes. For instance, the victim segment file (MICR5) contains a variable called `MICR5_VICTIMNO` (victim number) which identifies unique victims within a given incident. Said another way, for every unique `MIC1_NUMBER`, the victim number variable identifies victim #1, #2, through victim #n. To this extent, the `MIC1_NUMBER` can be combined with the `MICR5_VICTIMNO` to create a unique victim identifier (e.g., for `MIC1_NUMBER - 123456` and `MICR5_VICTIMNO - 1`, the Victim ID would be 123456_1).

The victim number that is identified in the MICR5 file is replicated in other datasets. In the MICR VOR file (victim-offender relationship) the variable `VOR_VICTIMNO` refers to the same victim number as in the MICR5 file. This means that a similarly constructed victim ID in the VOR file can be linked to the MICR5 file on the basis of the `MIC1_NUMBER` and the corresponding victim number. The files and variables necessary to make victim-offender-offense likages are detailed in Table 2.2.

Table 3.2: Unique Identifier Creation

Identifier (File Name)	Component Variables
Victim ID (MICR5)	`MIC1_NUMBER + MICR5_VICTIMNO`
Victim ID (MICR VOFNS)	`MIC1_NUMBER + VOFNS_VICTIMNO`
Victim ID (MICR VOR)	`MIC1_NUMBER + VOR_VICTIMNO`
Offender ID (MICR3)	`MIC1_NUMBER + MICR3_OFFENDERNO`
Offender ID (MICR VOR)	`MIC1_NUMBER + VOR_OFFENDERNO`
Offender ID (MICR7)	`MIC1_NUMBER + MICR7_ARRESTNO`
Offender ID (MICR ARMED)	`MIC1_NUMBER + ARMED_ARRESTNO`
Offense ID (MICR OUSED)	`MIC1_NUMBER + OUSED_OFFENSE_CO`
Offense ID (MICR OFFNS)	`MIC1_NUMBER + OFFNS_OFFENSE_CO`
Offense ID (MICR ATYPE)	`MIC1_NUMBER + ATYPE_OFFENSE_CO`
Offense ID (MICR VOFNS)	`MIC1_NUMBER + VOFNS_OFFENSE_CO`

As the complete **R** script will demonstrate (see the Appendix), this linking can be accomplished using the `paste()` function, which combines multiple variables to create character strings (similar to the concatenate function in SPSS). Once these identifiers have been created within each dataset, they can be linked in a sequential process to produce the final victim-offender-offense datafile. This process is detailed in Table 2.3.

Following the production of the triads file, multiple tools are available for reducing the overall file (which will likely exceed 1 million rows of data, 2015 alone is 1.26 million rows) to useful products for pursuing research questions. The following section will briefly describe some of these tools.

Table 3.3: Sequential MICR File Linking Process

Step	File Created	File A Linked to	. . . File B	Linking IDs
1	admin.addrs	MICR ADDRESS	MICR1	`MIC1_NUMBER`
2	victim.vofns	MICR VOFNS	MICR5	`MIC1_NUMBER` `Victim ID`
3	victim.vofns.vor	MICR VOR	victim.vofns	`MIC1_NUMBER` `Victim ID`
4	ofnd.oused	MICR OUSED	MICR3	`MIC1_NUMBER`
5	arrest.armed	MICR ARMED	ofnd.oused	`MIC1_NUMBER` `Offender ID`
6	ofnd.oused. arrest.armed	arrest.armed	ofnd.oused	`MIC1_NUMBER` `Offender ID`
7	offns.atype	MICR ATYPE	MICR OFFNS	`MIC1_NUMBER` `Offense ID`
8	victim.vofns.vor. ofnd.oused. arrest.armed	ofnd.oused. arrest.armed	victim.vofns.vor	`MIC1_NUMBER` `Offender ID` `Offense ID`
9	victim.vofns.vor. ofnd.oused.arrest armed.offns.atype	offns.atype	victim.vofns.vor. ofnd.oused. arrest.armed	`MIC1_NUMBER` `Offense ID`
10	Full Triads File	admin.addrs	victim.vofns.vor. ofnd.oused.arrest. armed.offns.atype	`MIC1_NUMBER`

3.3 Subsetting and Aggregating MICR

The full victim-offender-offense triads file will be quite large and it is unlikely that the entire file will be useful for any given research question. Instead, specific research questions will require extracting specific rows or variables. The purpose of this section is to demonstrate some possible tasks towards this end. Note that this section will make use of the full file that is produced by the **R** script in the Appendix, and will not cover the requisite steps in producing the file.

3.3.1 Selecting Particular Units

Now that unique ID numbers have been created for each distinct unit within the MICR data (e.g., victims, offenders, offenses), it is possible to extract a dataset consisting of unique units. Unfortunately, due to the complex file structure, such a dataset will necessarily represent something of a compromise. For instance, a common strategy in the analysis of NIBRS data is to use unique victims or offenders as the units of analysis, and each variable represents a summary of the offenders or victims, or offenses that were associated with those units. Said another way, each row may represent a unique offender, and there will be a variable for the average age of all of the victims they offended against in a given incident, a variable counting the number of female victims, white victims, and so on.

R is a flexible environment for performing such a data reduction task. In the following example, I will use the `data.table` package to create a file consisting of unique robbery offenders. Unfortunately, the script is not very straightforward for novice users, but once the user understands the general structure, it becomes relatively simple to adapt to a given problem. First, I will use the `subset` function to reduce the full 2015 MICR file to victims that were individual human beings and offenses that consisted of robbery.

```
> dta = read.csv("SAC MICR 2015 Total File.txt", header = TRUE)
> dta.rob = subset(dta, vic.type == "Individual" & offense == "Robbery")
> nrow(dta.rob) # Number of Observations
[1] 10946
```

Once this subsetting is complete, the remaining dataset consists of victim-offender dyads for robbery offenses. At this point, the data frame will be converted to a `data.table` and then the aggregation will begin. Detailed guides on using `data.table` are available here, here, and here. Essentially, the strategy here consists of first creating an aggregated data frame with the average of all numeric variables across each offender ID. Then, separate data tables are created for each categorical variable desired, and then merged onto the aggregated dataset. This process can be repeated until all required variables have been incorporated.

```
> library(data.table)
> #Convert to data.table
> dt = data.table(dta.rob)
> #Create aggregate file, with summaries for
> #number of victims, offender age, and victim age
> dt.agg = dt[, list(n.victims = .N, off.age = mean(off.age, na.rm = TRUE),
                    vic.age = mean(vic.age, na.rm = TRUE)), by = ofnd.id]
> #Create separate files with counts of
> #categorical variables, victim and offender sex
> dt.vsex = dt[, as.list(table(vic.sex)), by = ofnd.id];
>   names(dt.vsex) = c("ofnd.id", "v.female", "v.male", "v.unknown")
> dt.osex = dt[, as.list(table(off.sex)), by = ofnd.id];
>   names(dt.osex) = c("ofnd.id", "o.female", "o.male", "o.unknown")
> #Merge the categorical variables onto the aggregated data file
> dt.agg = merge(dt.agg, dt.vsex, by = "ofnd.id")
> dt.agg = merge(dt.agg, dt.osex, by = "ofnd.id")


> head(dt.agg)
      ofnd.id n.victims off.age vic.age v.female v.male
1: 10000166_1         1      17      18        0      1
2: 10000211_1         1      17      29        0      1
3: 10000648_1         1      29      20        1      0
```

```
4: 10000657_1          1      28      22          1      0
5: 10000663_1          1      18      17          0      1
6: 10000663_2          1      17      17          0      1
   v.unknown o.female o.male o.unknown
1:         0        0      1         0
2:         0        0      1         0
3:         0        0      1         0
4:         0        0      1         0
5:         0        0      1         0
6:         0        0      1         0
```

3.3.2 Counting Unique Units within Geographies

Another common task when analyzing MICR will be to summarize unique counts of units within geographies, such as the number of aggravated assaults within counties. This can be accomplished in a similar fashion to the example above, using `data.table`. However, for this example I will use the `dplyr` package, to demonstrate an alternative. The following describes creating a county-level dataset, counting the number of aggravated assault victims. Similar to the example above, I will begin by subsetting the overall file to human victims and aggravated assault offenses.

```
> dta.aa = subset(dta, vic.type == "Individual" &
                  offense == "Aggravated/Felonious Assault")
> nrow(dta.aa) # Number of Observations
[1] 30119
```

The `dplyr` package is a set of tools for manipulating data frames. One helpful feature is the ability to ”chain” or ”pipe” functions into a sequence which (for instance), takes a given data frame, selects particular variables, then filters particular rows, then groups output by a given categorical variable, and then performs some manner of summary function. Guides on using `dplyr` are available here and here. Specifically, the operator `%>%` is used to chain sequential function calls. The example below, I will start with the aggravated assault data frame (`dta.aa`), group the results by the county of the incident (`inc.county`), and will count the number of unique victims within each county, using the `length` and `unique` functions together.

```
> library(dplyr)
> county.aa = dta.aa %>% #start with the dta.aa data.frame, then...
   group_by(inc.county) %>% #group by county, then...
   summarize(n.aggs = length(unique(victim.id))) #count unique victims
> head(county.aa)
# A tibble: 6  2
  inc.county n.aggs
       <int>  <int>
1          1      5
2          2     13
3          3    147
4          4     23
5          5     27
6          6     15
```

Further, it is possible to use additional filters to count the number of aggravated assaults that meet a certain criteria, such as involving a firearm.

```
> county.gunaa = dta.aa %>% #start with the dta.aa data.frame, then...
   filter(ofns.weptype == "Firearm") %>% #select gun offenses, then...
   group_by(inc.county) %>% #group by county, then...
   summarize(n.gunaggs = length(unique(victim.id))) #count unique victims
> head(county.gunaa)
# A tibble: 6  2
  inc.county n.gunaggs
       <int>     <int>
1          2         1
2          3        21
3          4         4
4          5        12
5          6         1
6          7         3
```

These datasets can then be linked to Census data, or any other data pertaining to the geographic unit of analysis (e.g., city, ORI).

3.4 Moving Forward . . .

This tutorial has laid the groundwork for merging the MICR files into analyzable datasets in the **R** statistical computing environment. Further steps in analysis may include merging multiple years of data. This can be easily accomplished by running the cleaning script (see the Appendix) on the data files for each year, and then using a simple `merge` function. For instance, `combined.data = merge(data.2014, data.2015, all.x = TRUE, all.y = TRUE)` would accomplish such a task. However, it is recommended that the individual years be subsetted to only the necessary rows/structure, since the overall files are quite large on their own.

One point to consider is variation in reporting agencies in multiple years of data. Although MICR is fortunate to have a high participation rate among law enforcement agencies, there may be slight variations in which agencies report or do not report from year to year. One means of checking for this variation is to use the `setdiff` function. This function will identify the elements of one vector which do not appear in another. To this extent, one can create vectors of the unique ORI numbers reporting data in a given year and then compare them using the `setdiff` function. Just to demonstrate,

```
> set.a = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
> set.b = c(2, 4, 6, 8, 10)
> setdiff(set.a, set.b) # The elements of A that are not in B
[1] 1 3 5 7 9
```

The Appendix includes the entire code for cleaning, recoding, and merging the MICR files. Each year the variables seem to change very slightly, so please check the individual year files for consistency in case running the script produces any errors.

3.4.1 Building on this Tutorial

One of the strengths of using an open-source statistical computing platform is the vibrancy of its community. **R** users benefit from the experiences and advice of others tacking similar problems in different contexts. To this extent, this guide was written in the spirit of a living document that would continue to grow with new advice as researchers continue to work with incident-based data in **R**. If you believe you have helpful advice for working with MICR/NIBRS data in **R** that other users could potentially benefit, please contact the Michigan Justice Statistics Center so that this guide can continue to grow.

4 Appendix

4.1 Annotated .R Script for Cleaning and Merging MICR

```
> ###Workspace Setup
>
> library(plyr) ; library(dplyr) ; library(lubridate); library(car)
> ###Main Administrative File - Date/Time/Location
>
> admin = read.csv("MICR1 2015.txt", header = TRUE)
> #698083 Records, 698083 Unique MIC1 Numbers
>
> #Date and Time Recodes
>
> admin$MIC1_INC_DATE = as.character(admin$MIC1_INC_DATE)
> admin$MIC1_INC_DATE = ifelse(nchar(admin$MIC1_INC_DATE) < 6,
                              paste0("0", admin$MIC1_INC_DATE),
                              admin$MIC1_INC_DATE)
> admin$inc.date = as.Date(admin$MIC1_INC_DATE, format = "%m%d%y")
> admin$inc.month = month(admin$inc.date, abbr = FALSE)
> admin$inc.hour = ifelse(admin$MIC1_HOUR_OCCUR >= 0 &
                           admin$MIC1_HOUR_OCCUR <= 23,
                         admin$MIC1_HOUR_OCCUR, "NA")
> admin$inc.hour = factor(admin$inc.hour,
                         levels = c("0", "1", "2", "3", "4", "5", "6",
                                    "7", "8", "9", "10", "11", "12",
                                    "13", "14", "15", "16", "17", "18",
                                    "19", "20", "21", "22", "23"))
> #6.2% Missing the Hour of Occurrence
>
> #MIC1_COUNTY Indicator
>
> admin$inc.county = factor(admin$MIC1_COUNTY)
> #Geocodes
>
> addrs = read.csv("MICR ADDRESS 2015.txt", header =  TRUE)
> #364193 Records, 363918 Unique MIC1 Numbers
>
> addrs$lon = addrs$LONGITUDE
> addrs$lat = addrs$LATITUDE
> addrs$lon[addrs$lon < -90.418611 | addrs$lon > -82.122778] = NA
> addrs$lat[addrs$lat < 41.696111 | addrs$lat > 48.305833] = NA
> addrs$missing.geo = 0
> addrs$missing.geo[is.na(addrs$lon) | is.na(addrs$lat)] = 1
> #Invalid geocodes for 0.9% of address records
>
> ###Victim and Victim Offender Relationship File
>
> victim = read.csv("MICR5 2015.txt", header = TRUE)
> #699921 Records, #650257 Unique MIC1 Numbers
>
> victim$vic.ethnicity = recode(victim$MICR5_ETHNICITY,
                               "'A' = 'Arab' ; 'H' = 'Hispanic' ; 'O' = 'Other' ;
                               'U' = 'Unknown' ; else = NA")
> victim$vic.race = recode(victim$MICR5_RACE,
                          "'A' = 'Asian/Pacific Islander' ; 'B' = 'African American' ;
                          'I' = 'American Indian' ; 'W' = 'White' ; 'U' = 'Unknown' ;
                          else = NA")
> victim$vic.reside = recode(victim$MICR5_RESIDENT,
                            "'C' = 'Same MIC1_COUNTY' ;
                            'R' = 'Same Community' ;
                            'S' = 'Same State' ; 'O' = 'Out of State' ;
                            'U' = 'Unknown' ; else = NA")
> victim$vic.sex = recode(victim$MICR5_SEX,
                         "'F' = 'Female' ; 'M' = 'Male' ;
                         'U' = 'Unknown' ; else = NA")
> victim$vic.female = recode(victim$vic.sex,
                            "'Female' = 1 ; 'Male' = 0 ; else = NA")
> victim$vic.type = recode(victim$MICR5_TYPE,
                          "'B' = 'Business' ; 'F' = 'Financial Institution' ;
                           'G' = 'Government' ;
                           'I' = 'Individual' ; 'O' = 'Other' ;
                           'P' = 'Police Officer' ; 'R' = 'Religious Organization' ;
                           'S' = 'Society/Public' ; 'U' = 'Unknown' ; else = NA")
> victim$vic.individual = recode(victim$vic.type,
                                "'Individual' = 1 ; NA = NA ; else = 0")
> victim$vic.injury = recode(victim$MICR5_INJURY,
                           "'B' = 'Broken Bones' ; 'I' = 'Possible Internal Injury' ;
                            'F' = 'Fatal' ; 'L' = 'Severe Laceration' ;
                            'M' = 'Apparent Minor Injury' ; 'N' = 'None' ;
                            'O' = 'Other Major Injury' ; 'T' = 'Loss of Teeth' ;
                            'U' = 'Unconscious' ;
                            else = NA")
> victim$vic.inj.any = recode(victim$MICR5_INJURY,
                             "'N' = 'No Injury' ;
                             c('B', 'I', 'L', 'M', 'O', 'T', 'U') = 'Any Injury' ;
                             'F' = 'Fatal' ; else = NA")
> victim$vic.inj.sev = recode(victim$MICR5_INJURY,
                             "'N' = 'No Injury' ; c('M', 'O') = 'Minor Injury' ;
                             c('B', 'L', 'T', 'U', 'I') = 'Severe Injury' ;
                             'F' = 'Fatal' ; else = NA")
> victim$MICR5_VICTIM_AGE = as.character(victim$MICR5_VICTIM_AGE)
> victim$vic.age = as.numeric(victim$MICR5_VICTIM_AGE)
> victim$ofns.circumstance = recode(victim$MICR5_CIRCUMSTAN,
                                   "1 = 'Argument' ; 2 = 'Assault on LE' ;
                                   3 = 'Drug Dealing' ; 4 = 'Gangland' ;
                                   5 = 'Juvenile Gang' ; 6 = 'Lovers Quarrel' ;
                                   7 = 'Mercy Killing' ; 8 = 'Other Felony Involved' ;
                                   9 = 'Other Circumstances' ;
                                   10 = 'Unknown Circumstances' ;
                                   20 = 'Criminal Killed by Citizen' ;
                                   21 = 'Criminal Killed by Citizen' ;
                                   33 = 'Negligent Weapon Handling' ;
                                   34 = 'Other Negligent Killing' ; else = NA")
> victim$justify = recode(victim$MICR5_JUSTIFY,
                         "c(1, 2) = 'Police Kill Attacker' ;
                         3 = 'Criminal Attack Civilian' ;
                         5 = 'Criminal Killed During Commission of Crime'")
> victim$ofns.domvio = recode(victim$MICR5_DOM_VIOLENCE,
                             "'Y' = 'Domestic Violence' ; 'N' = 'Not DV' ; else = NA")
> ##VOFNS Table
>
> vofns = read.csv("MICR VOFNS 2015.txt", header = TRUE)
> #740546 records, 650257 unique MIC1 Numbers
> vor = read.csv("MICR VOR 2015.txt", header = TRUE)
> #244266 Records, 191160 unique MIC1 Numbers
>
> vor$vo.rel.known = recode(vor$VOR_VOREL,
                           "c(NA, 99) = 'Unknown VO Rel' ; else = 'Known VO Rel'")
> vor$vor.cat = recode(vor$VOR_VOREL,
                      "c(1, 2, 24, 26) = 'Current Intimate Partner' ;
                       c(13, 27, 32) = 'Former Intimate Partner' ;
                       c(3, 4, 5, 6, 7) = 'Blood Relative' ;
                       c(8, 9, 10, 11, 12) = 'Non-Blood Relative' ;
                       c(20, 21, 22, 23, 25, 28, 29, 33, 34) = 'Acquaintance' ;
                       c(30, 31) = 'Other Known' ;
                       98 = 'Stranger' ; 99 = 'Unknown' ; else = NA")
> vor$vor.cat = factor(vor$vor.cat)
> ###Law Enforcement Officers Killed or Assaulted
>
> leoka = read.csv("MICR LEOKA 2015.txt", header = TRUE)
> ###Offender File
>
```

```
> ofnd = read.csv("MICR3 2015.txt", header = TRUE)
> #588631 Records, 504528 Unique MIC1
>
> ofnd$off.race = recode(ofnd$MICR3_RACE,
                        "'A' = 'Asian/Pacific Islander' ; 'B' = 'African American' ;
                        'I' = 'American Indian' ; 'W' = 'White' ;
                        'U' = 'Unknown' ; else = NA")
> ofnd$off.sex = recode(ofnd$MICR3_SEX,
                       "'F' = 'Female' ; 'M' = 'Male' ; 'U' = 'Unknown' ; else = NA")
> ofnd$off.female = recode(ofnd$off.sex,
                          "'Female' = 1 ; 'Male' = 0 ; else = NA")
> ofnd$off.age = as.numeric(ofnd$MICR3_AGE)
> ###Offender Used Table
>
> oused = read.csv("MICR OUSED 2015.txt", header = TRUE)
> #748769 records, 682972 unique MIC1 Numbers
>
> oused$off.used = recode(oused$OUSED_USED,
                         "'A' = 'Alcohol' ; 'C' = 'Computer Equipment' ;
                         'D' = 'Drugs/Narcotics' ;
                         'N' = 'Not Applicable/None' ; else = NA")
> ofnd$VOR_OFFENDERNO = ofnd$MICR3_OFFENDERNO
> ofnd$ofnd.id = with(ofnd, paste0(MIC1_NUMBER, "_", VOR_OFFENDERNO))
> ofnd = arrange(ofnd, MIC1_NUMBER, VOR_OFFENDERNO)
> ###Offense Table
>
> act = read.csv("MICR OFFNS 2015.txt", header = TRUE)
> #758552 Records, 697904 Unique MIC1 Numbers
>
> act$offense.id = with(act, paste0(MIC1_NUMBER, "_", OFFNS_OFFENSE_CO))
> act$ofns.attempt = recode(act$OFFNS_ATTEMPT,
                           "'A' = 1 ; 'C' = 0 ; else = NA")
> act$ofns.weapon = recode(act$OFFNS_WEAPON,
                         "'00' = 'Unarmed' ; '11' = 'Firearm' ;
                         '11A' = 'Automatic Firearm' ;
                          '12' = 'Handgun' ; '12A' = 'Automatic Handgun' ;
                          '13' = 'Rifle' ; '13A' = 'Automatic Rifle' ;
                          '14' = 'Shotgun' ; '14A' = 'Automatic Shotgun' ;
                          '15' = 'Other Firearm' ; '15A' = 'Other Automatic Firearm' ;
                          '20' = 'Knife/Blade' ; '30' = 'Blunt Object' ;
                          '35' = 'Motor Vehicle' ; '40' = 'Personal Weapons' ;
                          '50' = 'Poison' ; '60' = 'Explosives' ;
                          '65' = 'Fire/Incendiary Devices' ; '70' = 'Drugs/Narcotics' ;
                          '85' = 'Asphyxiation' ; '88' = 'Other' ;
                          '99' = 'Unknown' ; else = NA")
> act$ofns.weptype = recode(act$OFFNS_WEAPON,
                           "c('00', '40') = 'Unarmed' ;
```

```
                              c('11', '11A', '12', '12A', '13', '13A', '14', '14A',
                              '15', '15A') = 'Firearm' ;
                              c('20', '30') = 'Melee Weapon' ;
                              c('35', '50', '60', '65', '70', '85', '88') = 'Other Weapons' ;
                              else = NA")
> act$ofns.biastype = recode(act$OFFNS_BIAS,
                              "0 = 'None' ; 11 = 'Anti-White' ; 12 = 'Anti-Black' ;
                               13 = 'Anti-Amer Indian' ; 14 = 'Anti-Asian' ;
                               15 = 'Anti-MultiRace' ; 21 = 'Anti-Jewish' ;
                               22 = 'Anti-Catholic' ; 23 = 'Anti-Protestant' ;
                               24 = 'Anti-Islamic' ; 25 = 'Anti-Other Religion' ;
                               26 = 'Anti-MultiRelig' ; 27 = 'Anti-Atheism' ;
                               32 = 'Anti-Hispanic' ;
                               33 = 'Anti-Other Ethnicity' ; 41 = 'Anti-Male Homosexual' ;
                               42 = 'Anti-Female Homosexual' ; 43 = 'Anti-Homosexual' ;
                               44 = 'Anti-Heterosexual' ; 45 = 'Anti-Bisexual' ;
                               51 = 'Anti-Female' ; 52 = 'Anti-Male' ;
                               61 = 'Anti-PhysDisability' ;
                               62 = 'Anti-MentalDisab' ; 88 = 'Other Bias' ;
                               99 = 'Unknown' ; else = NA")
> act$ofns.biascat = recode(act$OFFNS_BIAS,
                              "0 = 'None' ; c(11, 12, 13, 14, 15, 32, 33) = 'Racial/Ethnic' ;
                              c(21, 22, 23, 24, 25, 26, 27) = 'Religious' ;
                              c(41, 42, 43, 44, 45) = 'Sexual Orientation' ;
                              c(51, 52) = 'Gender' ; c(61, 62) = 'Disability' ;
                              88 = 'Other' ; 99 = 'Unknown' ; else = NA")
> act$ofns.location = recode(act$OFFNS_LOCATION,
                               "20 = 'Residence' ; c(2, 3, 5, 7, 8, 12, 14, 17, 19,
                               21, 23, 24, 34, 36, 37, 38, 39, 41,
                               44, 46, 55) = 'Business' ;
                               c(1, 4, 9, 11, 15, 31, 32, 45, 48,
                               49, 54) = 'Govt/School/Transport' ;
                               c(6, 10, 13, 16, 18, 33, 35, 40, 50,
                               56) = 'Public - Outdoors' ;
                               88 = 'Other' ; 99 = 'Unknown' ; else = NA")
> ###ATYPE Table
>
> atype = read.csv("MICR ATYPE 2015.txt", header = TRUE)
> #143045 Records, 130733 Unique MIC1 Numbers
>
> atype$act.type = recode(atype$ATYPE_TYPE,
                            "'B' = 'Buying/Receiving' ;
                            'c' = 'Cultivating/Manufacturing/Publishing' ;
                            'D' = 'Distributing/Selling' ; 'E' = 'Exploiting Children' ;
                            'G' = 'Other Gang' ;
                            'J' = 'Juvenile Gang' ; 'N' = 'None/Unknown' ;
                            'O' = 'Operating/Promoting/Assisting' ;
```

```
                           'P' = 'Possessing/Concealing' ;
                           'T' = 'Transporting/Transmitting/Importing' ;
                           'U' = 'Using/Consuming' ; else = NA")
> ###Arrest File
>
> arrest = read.csv("MICR7 2015.txt", header = TRUE)
> #260482 records, 236682 unique MIC1 Numbers
>
> arrest$arr.type = recode(arrest$MICR7_ARREST_TYP,
                           "'O' = 'On-View' ; 'S' = 'Cited - No Custody' ;
                           'T' = 'Custody - Previous Warrant'")
> arrest$arr.clearance = recode(arrest$MICR7_CLEARANCE,
                                "'Y' = 'Yes' ; 'N' = 'Previously Cleared' ; else = NA")
> arrest$dispo.und18 = recode(arrest$MICR7_DISPOSITIO,
                              "'H' = 'Handled within Dept.' ;
                              'R' = 'Referred to Other Authorities'")
> arrest$multi.arr = recode(arrest$MICR7_MULTI_ARR,
                            "c('C', 'M') = 'Person Arr Multi Incidents' ;
                            'N' = 'Person Arr One Incident'")
> arrest$arr.race = recode(arrest$MICR7_RACE,
                           "'A' = 'Asian/Pacific Islander' ; 'B' = 'African American' ;
                           'I' = 'American Indian' ; 'W' = 'White' ; 'U' = 'Unknown' ;
                           else = NA")
> arrest$arr.ethnicity = recode(arrest$MICR7_ETHNIC,
                                "'A' = 'Arab' ; 'H' = 'Hispanic' ; 'O' = 'Other' ;
                                'U' = 'Unknown' ; else = NA")
> arrest$arr.sex = recode(arrest$MICR7_SEX,
                          "'F' = 'Female' ; 'M' = 'Male' ; 'U' = 'Unknown' ; else = NA")
> arrest$arr.female = recode(arrest$arr.sex,
                             "'Female' = 1 ; 'Male' = 0 ; else = NA")
> arrest$arr.age = as.numeric(arrest$MICR7_AGE)
> arrest$arr.reside = recode(arrest$MICR7_RESIDENCE,
                             "'C' = 'Same MIC1_COUNTY' ; 'R' = 'Same Community' ;
                             'S' = 'Same State' ; 'O' = 'Out of State' ;
                             'U' = 'Unknown' ; else = NA")
> arrest$VOR_OFFENDERNO = arrest$MICR7_ARRESTNO
> ###Arrestee Armed
>
> armed = read.csv("MICR ARMED 2015.txt", header = TRUE)
> #255316 Records, 236682 Records
>
> armed$arr.weapon = recode(armed$ARMED_ARREST_ARM,
                            "'00' = 'Unarmed' ; '11' = 'Firearm' ;
                            '11A' = 'Automatic Firearm' ;
                            '12' = 'Handgun' ; '12A' = 'Automatic Handgun' ;
                            '13' = 'Rifle' ; '13A' = 'Automatic Rifle' ;
                            '14' = 'Shotgun' ; '14A' = 'Automatic Shotgun' ;
```

```
                              '15' = 'Other Firearm' ; '15A' = 'Other Automatic Firearm' ;
                              '20' = 'Knife/Blade' ; '30' = 'Blunt Object' ;
                              '35' = 'Motor Vehicle' ; '40' = 'Personal Weapons' ;
                              '50' = 'Poison' ; '60' = 'Explosives' ;
                              '65' = 'Fire/Incendiary Devices' ; '70' = 'Drugs/Narcotics' ;
                              '85' = 'Asphyxiation' ; '88' = 'Other' ;
                              '99' = 'Unknown' ; else = NA")
> ###Initial Merging Attempt
>
> #Link Addresses to Incidents
>
> admin = arrange(admin, MIC1_NUMBER)
> addrs = arrange(addrs, MIC1_NUMBER)
> admin.addrs = merge(admin, addrs, by = "MIC1_NUMBER", all.x = TRUE)
> rm(admin, addrs) #Clean Up Workspace
> #Missing Geocodes by MIC1_COUNTY
>
> m = admin.addrs %>%
   group_by(inc.county) %>%
   summarize(N = n(),
             Missing.Total = sum(missing.geo, na.rm = TRUE),
             Missing.PCT = sum(missing.geo, na.rm = TRUE)/n())
> summary(m$Missing.PCT) #Mean = 0.1%, Median = 0.0%, Max = 3.3%
> rm(m)
> #Link VOR and VOFNS to Victims
>
> victim$victim.id = with(victim, paste0(MIC1_NUMBER, "_", MICR5_VICTIMNO))
> vofns$victim.id = with(vofns, paste0(MIC1_NUMBER, "_", VOFNS_VICTIMNO))
> vor$victim.id = with(vor, paste0(MIC1_NUMBER, "_", VOR_VICTIMNO))
> vor$ofnd.id = with(vor, paste0(MIC1_NUMBER, "_", VOR_OFFENDERNO))
> victim = arrange(victim, MIC1_NUMBER, victim.id)
> vofns = arrange(vofns, MIC1_NUMBER, victim.id)
> vor = arrange(vor, MIC1_NUMBER, victim.id, ofnd.id)
> setdiff(vofns$victim.id, victim$victim.id)
> # All VOFNS Victim IDs in the Victim File
> setdiff(vor$victim.id, victim$victim.id)
> # All VOR Victim IDs in the Victim File
> victim.vofns = merge(victim, vofns, by = c("MIC1_NUMBER", "victim.id"),
                      all.x = TRUE, all.y = TRUE)
> vic.vor = merge(victim.vofns, vor, by = c("MIC1_NUMBER", "victim.id"),
                 all.x = TRUE, all.y = TRUE)
> #773390 Records, 650257 Unique MIC1 Numbers
> vic.vor$offense.id = with(vic.vor, paste0(MIC1_NUMBER, "_", VOFNS_OFFENSE_CO))
> rm(victim, vofns, vor, victim.vofns)
> #Link OUSED to Offenders
>
> ofnd$ofnd.id = with(ofnd, paste0(MIC1_NUMBER, "_", MICR3_OFFENDERNO))
> oused$offense.id = with(oused, paste0(MIC1_NUMBER, "_", OUSED_OFFENSE_CO))
> ofnd = arrange(ofnd, MIC1_NUMBER)
> oused = arrange(oused, MIC1_NUMBER)
> length(setdiff(ofnd$MIC1_NUMBER, oused$MIC1_NUMBER))
> ofnd.oused = merge(ofnd, oused, by = "MIC1_NUMBER",
                    all.x = TRUE, all.y = TRUE)
> #867438 Records, 699332 Unique MIC1 Numbers
> ofnd.oused$MIC1_ORIG_YEAR = 2015L
> rm(ofnd, oused)
> #Link Activity Type to Offenses
>
> act$offense.id = with(act, paste0(MIC1_NUMBER, "_", OFFNS_OFFENSE_CO))
> atype$offense.id = with(atype, paste0(MIC1_NUMBER, "_", ATYPE_OFFENSE_CO))
> act = arrange(act, MIC1_NUMBER, offense.id)
> atype = arrange(atype, MIC1_NUMBER, offense.id)
> length(setdiff(atype$offense.id, act$offense.id))
> # All atype offense numbers in act file
> act.atype = merge(act, atype, by = c("MIC1_NUMBER", "offense.id"),
                   all.x = TRUE, all.y = TRUE)
> #762188 Records, 697904 Unique MIC1 Numbers
> rm(act, atype)
> #Link Arrests to Offenders
>
> arrest$ofnd.id = with(arrest, paste0(MIC1_NUMBER, "_", MICR7_ARRESTNO))
> armed$ofnd.id = with(armed, paste0(MIC1_NUMBER, "_", ARMED_ARRESTNO))
> arrest = arrange(arrest, MIC1_NUMBER, ofnd.id)
> armed = arrange(armed, MIC1_NUMBER, ofnd.id)
> length(setdiff(arrest$ofnd.id, armed$ofnd.id))
> arr = merge(arrest, armed, by = c("MIC1_NUMBER", "ofnd.id"),
             all.x = TRUE, all.y = TRUE)
> #264222 Records, 239808 Unique MIC1 Numbers
> rm(arrest, armed)
> ofnd.oused = arrange(ofnd.oused, MIC1_NUMBER, ofnd.id)
> arr = arrange(arr, MIC1_NUMBER, ofnd.id)
> length(setdiff(ofnd.oused$ofnd.id, arr$ofnd.id))
> ofnd.arr = merge(ofnd.oused, subset(arr, select = -MIC1_ORIG_YEAR),
                  by = c("MIC1_NUMBER", "ofnd.id"), all.x = TRUE, all.y = TRUE)
> #876683 Records, 702341 Unique MIC1 Numbers, 597534 Unique Offender IDs
> ofnd.arr$MIC1_ORIG_YEAR = 2015
> rm(ofnd.oused, arr)
> #Link Offenders and Arrests to Victims and VOR
>
> vic.vor = arrange(vic.vor, MIC1_NUMBER, victim.id, ofnd.id, offense.id)
> ofnd.arr = arrange(ofnd.arr, MIC1_NUMBER, ofnd.id, offense.id)
> length(setdiff(vic.vor$ofnd.id, ofnd.arr$ofnd.id))
> # Zero vic.vor offender IDs that are not in the Offender arrest file
> vic.ofnd = merge(vic.vor, ofnd.arr,
                  by = c("MIC1_NUMBER", "ofnd.id", "offense.id"),
                  all.x = TRUE, all.y = TRUE)
> vic.ofnd = arrange(vic.ofnd, MIC1_NUMBER, victim.id, ofnd.id, offense.id)
> #1,343,338 Records, 706649 Unique MIC1 Numbers,
> #690237 Unique Victims, 710698 Unique Offenders
> vic.ofnd$MIC1_ORIG_YEAR = 2015
> rm(vic.vor, ofnd.arr)
> #Link Offenses to Victims and Offenders
>
> act.atype = arrange(act.atype, MIC1_NUMBER, offense.id)
> length(setdiff(act.atype$offense.id, vic.ofnd$offense.id))
> triads = merge(vic.ofnd, act.atype,
                 by = c("MIC1_NUMBER", "offense.id"), all.x = TRUE, all.y = TRUE)
> triads = arrange(triads, MIC1_NUMBER, victim.id, ofnd.id, offense.id)
> triads$MIC1_ORIG_YEAR = 2015
> #Each row represents a victim x offender x offense triad
> #1,294,000 Triads across 744,104 Incidents
> rm(vic.ofnd, act.atype)
> #Link Victim x Offender x Offense Traids to Administrative File
>
> admin.addrs = arrange(admin.addrs, MIC1_NUMBER)
> length(setdiff(admin.addrs$MIC1_NUMBER, triads$MIC1_NUMBER))
> #178 Admin Address MIC1 NUmbers not in the Triads file
> dta = merge(triads, subset(admin.addrs, select = -MIC1_ORIG_YEAR),
              by = "MIC1_NUMBER", all.x = TRUE, all.y = TRUE)
> dta = arrange(dta, MIC1_NUMBER, victim.id, ofnd.id, offense.id)
> dta$MIC1_ORIG_YEAR = 2015
> #1,362,433 Records, 721808 Unique MIC1 Numbers,
> #690237 Victims, 710698 Offenders, 762934 Offenses
> rm(triads, admin.addrs)
> #Save to CSV
>
> dta = select(dta, MIC1_ORIG_YEAR, MIC1_NUMBER, offense.id, ofnd.id, victim.id,
               MICR5_NUMBER, vic.ethnicity, vic.race, vic.sex, vic.female, vic.age,
               vic.reside, vic.type, vic.individual, vic.injury, vic.inj.any,
               vic.inj.sev, ofns.circumstance, justify, ofns.domvio, VOFNS_NUMBER,
               VOFNS_OFFENSE_CO, VOFNS_VICTIMNO, VOR_NUMBER, VOR_OFFENDERNO,
               VOR_VICTIMNO, vo.rel.known, vor.cat, MICR3_NUMBER, off.race, off.sex,
               off.female, off.age, OUSED_NUMBER, OUSED_OFFENSE_CO, off.used,
               MICR7_ARRESTNO, MICR7_CHARGE, arr.type, arr.clearance, dispo.und18,
               multi.arr, arr.race, arr.ethnicity, arr.sex, arr.female, arr.age,
               arr.reside, ARMED_NUMBER, ARMED_ARRESTNO, arr.weapon, OFFNS_NUMBER,
               OFFNS_OFFENSE_CO, ofns.attempt, ofns.weapon, ofns.weptype,
               ofns.biastype, ofns.biascat, ofns.location, ATYPE_NUMBER,
               ATYPE_OFFENSE_CO, act.type, MIC1_ASSIST_ORI, MIC1_DATE_IND,
               MIC1_GEO_CODE, MIC1_ORI, MIC1_CITY_TWP, MIC1_CLEAR_DATE,
               MIC1_EXCEPT_CLE, inc.date, inc.month,
               inc.hour, inc.county, ZIP, lon, lat, missing.geo)
 > dta = within(dta, {
    offense = NA
    offense[OFFNS_OFFENSE_CO == 14000] = "Abortion"
    offense[OFFNS_OFFENSE_CO == 13002] = "Aggravated/Felonious Assault"
    offense[OFFNS_OFFENSE_CO == 60000] = "Antitrust"
    offense[OFFNS_OFFENSE_CO == 20000] = "Arson"
    offense[OFFNS_OFFENSE_CO == 51000] = "Bribery"
    offense[OFFNS_OFFENSE_CO == 22002] = "Burglary Entry W/Out Force (Intent)"
    offense[OFFNS_OFFENSE_CO == 22001] = "Burglary Forced Entry"
    offense[OFFNS_OFFENSE_CO == 22003] = "Burglary Unlawful Entry (No Intent)"
    offense[OFFNS_OFFENSE_CO == 56000] = "Civil Rights"
    offense[OFFNS_OFFENSE_CO == 40002] =
      "Commercialized Sex - Assisting/Promoting Prostitution"
    offense[OFFNS_OFFENSE_CO == 40001] = "Commercialized Sex - Prostitution"
    offense[OFFNS_OFFENSE_CO == 62000] = "Conservation"
    offense[OFFNS_OFFENSE_CO == 77000] = "Conspiracy"
    offense[OFFNS_OFFENSE_CO == 29000] = "Damage to Property"
    offense[OFFNS_OFFENSE_CO == 53001] = "Disorderly Conduct"
    offense[OFFNS_OFFENSE_CO == 42000] = "Drunkenness"
    offense[OFFNS_OFFENSE_CO == 59000] = "Election Laws"
    offense[OFFNS_OFFENSE_CO == 27000] = "Embezzlement"
    offense[OFFNS_OFFENSE_CO == 49000] = "Escape/Flight"
    offense[OFFNS_OFFENSE_CO == 21000] = "Extortion"
    offense[OFFNS_OFFENSE_CO == 38001] = "Family - Abuse/Neglect Nonviolent"
    offense[OFFNS_OFFENSE_CO == 38002] = "Family - Nonsupport"
    offense[OFFNS_OFFENSE_CO == 38003] = "Family - Other"
    offense[OFFNS_OFFENSE_CO == 25000] = "Forgery/Counterfeiting"
    offense[OFFNS_OFFENSE_CO == 26006] = "Fraud - Bad Checks"
    offense[OFFNS_OFFENSE_CO == 26002] = "Fraud - Credit Card/ATM"
    offense[OFFNS_OFFENSE_CO == 26001] =
      "Fraud - False Pretense/Swindle/Confidence Game"
    offense[OFFNS_OFFENSE_CO == 26003] = "Fraud - Impersonation"
    offense[OFFNS_OFFENSE_CO == 26004] = "Fraud - Welfare"
    offense[OFFNS_OFFENSE_CO == 26005] = "Fraud - Wire"
    offense[OFFNS_OFFENSE_CO == 39001] = "Gambling - Betting/Wagering"
    offense[OFFNS_OFFENSE_CO == 39003] = "Gambling - Equipment Violations"
    offense[OFFNS_OFFENSE_CO == 39002] = "Gambling - Operating/Promoting/Assisting"
    offense[OFFNS_OFFENSE_CO == 39004] = "Gambling - Sports Tampering"
    offense[OFFNS_OFFENSE_CO == 55000] = "Health and Safety"
    offense[OFFNS_OFFENSE_CO == 54001] = "Hit and Run Motor Vehicle Accident"
    offense[OFFNS_OFFENSE_CO == 64001] = "Human Trafficking, Comm. Sex Acts"
    offense[OFFNS_OFFENSE_CO == 64002] = "Human Trafficking, Inv. Servitude"
    offense[OFFNS_OFFENSE_CO == 3000] = "Immigration"
    offense[OFFNS_OFFENSE_CO == 13003] = "Intimidation/Stalking"
    offense[OFFNS_OFFENSE_CO == 57002] = "Invasion of Privacy Other"
    offense[OFFNS_OFFENSE_CO == 9004] = "Justifiable Homicide"
```

```
offense[OFFNS_OFFENSE_CO == 70000] = "Juvenile Runaway"
offense[OFFNS_OFFENSE_CO == 10001] = "Kidnapping/Abduction"
offense[OFFNS_OFFENSE_CO == 23007] = "Larceny - Other"
offense[OFFNS_OFFENSE_CO == 23001] = "Larceny - Pocket Picking"
offense[OFFNS_OFFENSE_CO == 23002] = "Larceny - Purse Snatching"
offense[OFFNS_OFFENSE_CO == 23003] = "Larceny - Theft from Building"
offense[OFFNS_OFFENSE_CO == 23004] =
  "Larceny - Theft from Coin Operated Machine/Device"
offense[OFFNS_OFFENSE_CO == 23005] = "Larceny - Theft from Motor Vehicle"
offense[OFFNS_OFFENSE_CO == 23006] =
  "Larceny - Theft of Motor Vehicle Parts/Accessories"
offense[OFFNS_OFFENSE_CO == 41001] = "Liquor License - Establishment"
offense[OFFNS_OFFENSE_CO == 41002] = "Liquor Violations - Other"
offense[OFFNS_OFFENSE_CO == 2000] = "Military"
offense[OFFNS_OFFENSE_CO == 73000] = "Miscellaneous Criminal Offense"
offense[OFFNS_OFFENSE_CO == 24002] = "Motor Vehicle as Stolen Property"
offense[OFFNS_OFFENSE_CO == 24003] = "Motor Vehicle Fraud"
offense[OFFNS_OFFENSE_CO == 24001] = "Motor Vehicle Theft"
offense[OFFNS_OFFENSE_CO == 9001] = "Murder/Non-negligent Manslaughter"
offense[OFFNS_OFFENSE_CO == 35002] = "Narcotic Equipment Violations"
offense[OFFNS_OFFENSE_CO == 9002] = "Negligent Homicide/Manslaughter"
offense[OFFNS_OFFENSE_CO == 9003] =
  "Negligent Homicide - Vehicle/Boat/Snowmobile/ORV"
offense[OFFNS_OFFENSE_CO == 13001] = "Non-aggravated Assault"
offense[OFFNS_OFFENSE_CO == 37000] = "Obscenity"
offense[OFFNS_OFFENSE_CO == 50000] = "Obstructing Justice"
offense[OFFNS_OFFENSE_CO == 48000] = "Obstructing Police"
offense[OFFNS_OFFENSE_CO == 54002] =
  "Operating Under Influence of Liquor (OUIL) or Drugs (OUID)"
offense[OFFNS_OFFENSE_CO == 30004] = "Organized Retail Fraud"
offense[OFFNS_OFFENSE_CO == 10002] = "Parental Kidnapping"
offense[OFFNS_OFFENSE_CO == 36003] = "Peeping Tom"
offense[OFFNS_OFFENSE_CO == 22004] = "Possession of Burglary Tools"
offense[OFFNS_OFFENSE_CO == 53002] = "Public Peace - Other"
offense[OFFNS_OFFENSE_CO == 40003] = "Purchasing Prostitution"
offense[OFFNS_OFFENSE_CO == 28000] = "Recovery of Stolen Property"
offense[OFFNS_OFFENSE_CO == 30001] = "Retail Fraud - Misrepresentation"
offense[OFFNS_OFFENSE_CO == 30003] = "Retail Fraud - Refund/Exchange"
offense[OFFNS_OFFENSE_CO == 30002] = "Retail Fraud - Theft"
offense[OFFNS_OFFENSE_CO == 12000] = "Robbery"
offense[OFFNS_OFFENSE_CO == 36004] = "Sex Offense - Other"
offense[OFFNS_OFFENSE_CO == 11007] = "Sexual Contact Forcible CSC2"
offense[OFFNS_OFFENSE_CO == 11008] = "Sexual Contact Forcible CSC4"
offense[OFFNS_OFFENSE_CO == 36001] = "Sexual Penetration Non-forcible"
offense[OFFNS_OFFENSE_CO == 36002] = "Sexual Penetration Non-forcible Other"
offense[OFFNS_OFFENSE_CO == 11005] = "Sexual Penetration Object CSC1"
offense[OFFNS_OFFENSE_CO == 11006] = "Sexual Penetration Object CSC3"
```

```
    offense[OFFNS_OFFENSE_CO == 11003] = "Sexual Penetration Oral/Anal CSC1"
    offense[OFFNS_OFFENSE_CO == 11004] = "Sexual Penetration Oral/Anal CSC3"
    offense[OFFNS_OFFENSE_CO == 11001] = "Sexual Penetration Penis/Vagina CSC1"
    offense[OFFNS_OFFENSE_CO == 11002] = "Sexual Penetration Penis/Vagina CSC3"
    offense[OFFNS_OFFENSE_CO == 58000] = "Smuggling"
    offense[OFFNS_OFFENSE_CO == 75000] = "Solicitation"
    offense[OFFNS_OFFENSE_CO == 1000] = "Sovereignty"
    offense[OFFNS_OFFENSE_CO == 61000] = "Tax/Revenue"
    offense[OFFNS_OFFENSE_CO == 57001] = "Trespass"
    offense[OFFNS_OFFENSE_CO == 63000] = "Vagrancy"
    offense[OFFNS_OFFENSE_CO == 35001] = "Violation of Controlled Substance"
    offense[OFFNS_OFFENSE_CO == 52001] = "Weapons Offense - Concealed"
    offense[OFFNS_OFFENSE_CO == 52002] = "Weapons Offense - Explosives"
    offense[OFFNS_OFFENSE_CO == 52003] = "Weapons Offense - Other"
  })
> write.csv(dta, "SAC MICR 2015 Total File.txt")
```