

# SDP Data Building Tasks

*Strategic Data Project*

*October 31, 2016*

## Contents

<b>SDP Data Building Tasks</b>	<b>1</b>
Overview . . . . .	1
Introduction . . . . .	2
Task 1: STUDENT ATTRIBUTES . . . . .	3
Task 2: STUDENT SCHOOL YEAR . . . . .	22
Task 3: IDENTIFYING THE NINTH-GRADE COHORT . . . . .	26
- TODO - . . . . .	27
- TODO Task 3 Slightly Different Numbers than SDP ##- . . . . .	32
Task 4: STUDENT SCHOOL ENROLLMENT . . . . .	33
Task 4: STUDENT TEST SCORES . . . . .	37
Task 6: STUDENT CLASS ENROLLMENT . . . . .	43
Task 7 STUDENT NSC ENROLLMENT . . . . .	51

## SDP Data Building Tasks

```
library(tidyverse) # main suite of R packages to ease data analysis
library(magrittr) # allows for some easier pipelines of data
# Read in some R functions that are convenience wrappers
source("R/functions.R")
library(haven) # required for importing .dta files
```

## Overview

Housed at the Center for Education Policy Research at Harvard University, the Strategic Data Project (SDP) partners with school districts, school networks, and state agencies across the US. **Our mission is to transform the use of data in education to improve student achievement.** We believe that with the right people, the right data, and the right analyses, we can significantly improve the quality of strategic policy and management decisions.

## Core Strategies

To achieve our mission, SDP pursues three core strategies:

1. Placing top-notch analytic leaders as “Fellows” for two years with our partner agencies; SDP supports more than 40 Data and Agency Fellows serving partner educational agencies – districts, states, and charter management organizations–across the nation. This number will grow to nearly 70 in 2012.
2. Conducting rigorous diagnostic analyses of teacher effectiveness and college-going success using existing agency data; and

We have completed diagnostics in teacher effectiveness and / or college-going success in seven districts, with more diagnostics currently underway or planned in additional district and state partner agencies.

3. Disseminating our tools, methods, and lessons learned to many more education agencies.

Through the diagnostic analyses, we have developed a body of knowledge around effective data use. The release of this toolkit reflects SDP's third core strategy to spread knowledge and build capacity within educational agencies for effective data use.

## SDP DIAGNOSTICS

Our second core strategy, conducting rigorous diagnostic analyses using existing agency data, focuses on two core areas: **(1) college-going success and attainment for students** and **(2) human capital** (primarily examining teacher effectiveness).

The diagnostics are a set of analyses that frame actionable questions for education leaders. By asking questions such as, *How well do students transition to postsecondary education?* or *How successfully is an agency recruiting effective teachers?* we support education leaders to develop a deep understanding of student achievement in their agency. In an effort to make these analyses accessible and more widely used, this toolkit helps analysts collect data and produce analyses associated with the SDP College-Going and Human Capital diagnostics.

Notably, the diagnostic analyses in this release of our toolkit are specific to the College-Going diagnostic. The data collection (Identify), data cleaning (Clean), and best practices (Adopt) stages of the toolkit, however, are applicable to either diagnostic and convey general data use guidelines valuable to any analysts interested in increasing the quality and rigor of their analyses. Later releases will address the analyses in our Human Capital diagnostic.

## Introduction

### SDP Data Building Tasks

Congratulations on identifying the data elements that are essential for conducting rigorous analyses in your organization. **Clean** is the next stage in the SDP Toolkit for Effective Data Use. To successfully move through the **Clean** stage, you should review the **Identify** component of this toolkit.

Upon completing this stage, you will have produced clean research files that will allow you to **Connect** and **Analyze** data related to college-going success in your agency.

## THE TASKS

**Clean** consist of five tasks that share a similar structure. The tasks are geared toward analysts with at least moderately strong data background and comfort with statistics. Each task provides hands-on experience building specific components of the research file used for the SDP CollegeGoing Diagnostic Analyses.

The tasks are listed as follows: - Task 1 Student Attributes - Task 2 Student School Year - Task 3 Identifying the Ninth Grade Cohort - Task 4 Student School Enrollment - Task 5 Prior Achievement

Each task is accompanied by a practice file dataset upon which all data snapshots and output are based. These datasets consist of simulated data that have been fully de-identified. We strongly recommend that you use these datasets to work through the tasks and check your answers. The datasets are available for download at ([www.gse.harvard.edu/sdp/tools](http://www.gse.harvard.edu/sdp/tools))[[www.gse.harvard.edu/sdp/tools](http://www.gse.harvard.edu/sdp/tools)]. Note that the tasks follow a logical sequence from Task 1 to Task 5, and some tasks require the output of previous tasks. However, because we provide all necessary practice files for each task, you may also choose to work on the tasks out of order. For instance, you may be first interested in identifying the ninth-grade cohort for students in your agency with Task 3.

To successfully complete all parts of this toolkit, however, you should work your way through all five tasks. The output of each task will be needed to successfully complete the Connect and Analyze stages of the

toolkit. Lastly, it is important to note that the tasks do not show you how to develop every single component and detail of the files to be used in Connect and Analyze. Our goal is to equip you with an understanding for the core process of constructing robust, clean research files. We do, however, aim to explicitly indicate what additional elements are needed in the **DATA DESCRIPTION** section of each task to deliver a fully realized research file. Furthermore, we also provide a **DECISION RULES GLOSSARY** in the Appendix at the end of this document to provide guidance on how to approach the cleaning process for these additional elements.

For those who are less familiar with or who need to brush up on Stata use, we also include a **R GLOSSARY** of commonly used commands in the Appendix at the end of this document. Through this set of tasks, you will learn effective practices for: data transformations, new variable construction, and the implementation of key decision rules.

## TASK STRUCTURE

The core of each task is a set of step-by-step instructions that guide you through the work. For each task you will find:

- Purpose — Clarifies the importance of the task.
- How to Start — Identifies the input file(s) you will need to complete the task and guidelines for apply the task to your own agency's data.
- Data Description — Lists the data elements you will need to complete the task and describes the uniqueness of key data elements.
- Instructions — Provides logical instructions on transforming the data with R code and fill-in-the-blank snapshots that help you visualize changes to your data.
- Solutions — Provides answers for the data snapshot exercises.

After completing these tasks, you will be well-positioned to use your own agency's data to construct similar clean research files needed in the Connect and Analyze stages.

Finally, if you find yourself in need of additional guidance, the friendly research team at SDP is available to help: [sdp@gse.harvard.edu](mailto:sdp@gse.harvard.edu)

## Task 1: STUDENT ATTRIBUTES

### PURPOSE

Through **Task 1: Student Attributes**, you will take the raw Student Attributes file and generate a cleaned Student Attributes output file that has only one observation per student. These data will allow you to examine college-going outcomes by race/ethnicity.

The core assignments of this task are to:

1. Resolve instances in which the same student appears with different values for race/ethnicity in different years. Our goal is to have only one race/ethnicity associated with each student.
2. Drop duplicate observations so the file is unique by student—that is, it contains only one observation per student. Upon completing this task, you will have a clean `Student_Attributes` file that can then be used as to create the analysis file in Connect. From Task 1, Task 2 is a natural next step, in which you will clean the `Student_School_Year` file in preparation for Task 3 and Task 4.

### HOW TO START

To begin, open the provided `Student_Attributes` practice file.

```

# Step 1: Load the college-going analysis file into Stata
# library(haven) # commented out, we've already read it in above
# To read data from a zip file and unzip it in R we can
# create a connection to the path of the zip file
# To read data from a zip file we create a connection to the path of the
# zip file
tmpfileName <- "raw/Student_Demographics_Raw.dta"
# This assumes analysis is a raw subfolder from where the file is read,
# in this case inside the zipfile
con <- unz(description = "data/raw.zip", filename = tmpfileName,
            open = "rb")
# The zipfile is located in the subdirectory data, called raw.zip
stuatt <- read_stata(con) # read data in the data subdirectory
close(con) # close the connection to the zip file, keeps data in memory

```

The input file contains data for school years 2000-01 through 2006-07.

Normally race is considered a time-invariant variable that is unique by student.

In this instance, we deal with a case in which race is stored in a file unique by student and school year, which is instead time-variant. This task aims to take convert the dataset from being time-variant to being time-invariant.

If this is your first time going through the task, we recommend starting with the practice file, rather than your agency's own data file. Doing so will help you learn SDP's cleaning methodology and allow you to easily check your answers from a common dataset. You may then apply these methods to your agency's own `Student_Attributes` data with confidence. To learn more about the data you will need to collect in your agency, refer to `Identify: Data Specification Guide` and the `DATA DESCRIPTION` section of this document.

In addition to the practice file, you may also find it useful to complete the data snapshot exercises provided in the task. These exercises will allow you to visualize changes to the data occurring in each step of the task. Solutions for the exercises are provided at the end of the task.

## DATA DESCRIPTION

In `Identify: Data Specification Guide`, we specify the data elements included in the `Student_Attributes` research file.[^You may be wondering how this specification compares to the version in `Identify: Data Specification Guide`. Here are the primary changes: First, the `race_ethnicity` variable is coded as a string rather than being numeric, as specified in the `Data Specification Guide`. You will correct this in the task as it will facilitate the process of making the file unique by sid. Second, we are examining a time-variant data set. In the `Data Specification Guide`, the `Student_Attributes` file is specified as being unique by sid. In this case, the data are time-variant and unique by sid and `school_year`. Note that some districts may actually store `race_ethnicity` in a time-variant form such as this, and it is our job through this task to make the data time-invariant, i.e. each student only has a single value for `race_ethnicity` across time. Third, we are examining a partial data set including only `sid`, `school_year`, and `race_ethnicity`. We do not include variables such as `male`, `hs_diploma`, or `hs_diploma_type`, or `hs_diploma_date` to simplify the task. These variables are essential for later analyses but are left for you to complete as a further exercise. For guidance on cleaning these additional variables, refer to the `DECISION RULES GLOSSARY` at the end of this document and use this task as a reference.]

In this task, we examine a partial version of the `Student_Attributes` file that includes only `sid`, `school_year`, and `race_ethnicity`. This partial version is presented to help you learn the `Student_Attributes` cleaning process to make a file unique by sid without having to worry about additional `Student_Attributes` variables such as `male`, `hs_diploma`, `hs_diploma_type`, or `hs_diploma_date`.

The relevant variables and definitions you will need to complete the task are illustrated below:

```
glimpse(stuatt)
```

Observations: 87,534

```
Variables: 9
```

```
$ sid <dbl> 1, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 4, 5, 6, 6, 6, 7,
```

```
$ school_year      <dbl> 2004, 2005, 2006, 2007, 2006, 2007, 2006, 2006, 200
```

```
$ male <dbl> 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
```

```
$ race ethnicity
<chr> "B". "H". "H". "H". "W". "B". "H". "B". "B". "B". "
```

```
$ race_ethnicity      <chr>  B ,  H ,  H ,  H ,  W ,  B ,  H ,  B ,  B ,  B ,
$ birth date         <dbl> 10869, 10869, 10869, 10869, 11948, 11948, 11724, 11
```

```
$ birth_date      <dbl> 10889, 10889, 10889, 10889, 11948, 11948, 11724, 11
$ first 9th school year reported <dbl> 2004 2004 2004 2004 NaN NaN 2007 2007 2007
```

```
$ first_9th_school_year_reported <dbl> 2004, 2004, 2004, 2004, NaN, NaN, 2007, 2007, 2007,
$ hs_diploma <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1
```

```
$ hs_diploma      <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
$ hs_diploma_type <chr> "", "", "", "", "Standard Diploma", "College Prep, A
```

```
$ hs_diploma_type      <chr> "", "", "", "", "Standard Diploma", "College Prep D
$ hs_diploma_date      <date> NA, NA, NA, NA, 2000-06-05, 2000-05-04, NA, NA, NA
```

```
$ hs_diploma_date      <date> NA, NA, NA, NA, 2008-06-05, 2009-05-24, NA, NA, NA
```

**Uniqueness:** ideally, the data in its raw form would be unique by `sid`.

However, this may not be the case as some agencies might record `race_ethnicity` in a time-variant manner,

However, this may not be the case as some agencies might record `race_chemistry` in a time variant manner, such as by school year. To address this, we explain how to take the raw research file from being unique by

such as by school year. To address this, we explain how to take the raw research file from being unique by sid and school\_year to being unique by sid alone. Once the file is unique by sid alone, it is ready to be

incorporated into the analysis file in the Connect stage.

Examine your `Student_Attributes` raw research file input dataset. According to the data specification, the

file should be unique by **sid**. Examine the snapshot below to determine if it is unique as described.

```
head(stuatt)
```

```
# A tibble: 6 × 9
```

```
sid school_year male race_ethnicity birth_date first_9th_school_year_reported hs_dipl
```

```

      <dbl>      <dbl> <dbl>      <chr>      <dbl>      <dbl>      <d

```

1	1	2004	1	B	10869	2004
---	---	------	---	---	-------	------

1	1	2001	1	D	10000	2001
2	1	2005	1	H	10869	2004

2	1	2005	1	H	10869	2004
3	1	2006	1	H	10869	2004

3	1	2006	1	H	10869	2004
4	1	2007	1	H	10869	2004

4	1	2007	1	H	10869	2004
5	2	2006	0	H	11048	N=N

5	2	2006	0	W	11948	NaN
6	0	2007	0	D	11948	NaN

6	2	2007	0	B	11948	NaN
---	---	------	---	---	-------	-----

```
# ... with 1 more variables: hs_diploma_date <date>
```

```
# A quick way to test this in R
```

```
nvals(stuatt$sid) == nrow(stuatt)
```

```
[1] FALSE
```

```
# Checks that number of unique values of `sid` equals number of rows
```

## Step 0: Read in the Data

```
# Read in Stata
```

```
# library(haven) # already loaded so we comment it out
```

```
tmpfileName <- "raw/Student Demographics Raw.dta"
```

```
con <- unz(description = "data/raw.zip", filename = tmpfileName,
           open = "rb")
```

```
stuatt <- read_stata(con) # read data in the data subdirectory
```

```
stuatt <- read_stata(con) # read data in the stata binary
stuatt <- as.data.frame(stuatt)
```

```
Student <- as.data.frame(Student)
close(con)
```

Now drop the `first_9th_school_year_reported` variable. You will create a `first_9th_school_year_reported` variable in Task 3 that also imputes this variable for transfer-ins.

```
# In R one way to drop a variable is by assigning it a NULL value
stuatt$first_9th_school_year_reported <- NULL
# For testing purposes, let's specify a variable which indexes the SIDs
# we will use to check our work
idx <- c(2, 8552, 12506) # Specify which SIDs are interesting
# Now we can easily view only relevant data
stuatt[stuatt$sid %in% idx,]
```

	sid	school_year	male	race_ethnicity	birth_date	hs_diploma	hs_diploma_type	hs_diploma_date
5	2	2006	0	W	11948	1	Standard Diploma	2008-06-05
6	2	2007	0	B	11948	1	College Prep Diploma	2009-05-24
34290	8552	2005	1	W	12334	0		<NA>
34291	8552	2006	0	A	12334	0		<NA>
34292	8552	2006	1	W	12334	0		<NA>
34293	8552	2007	1	W	12334	0		<NA>
34294	8552	2009	1	W	12334	0		<NA>
50064	12506	2004	1	H	11803	0		<NA>
50065	12506	2005	0	H	11803	0		<NA>

## Step 1

Create one consistent value for gender for each student across years

```
# Create one consistent value for gender for each student across years
# View the data
```

```
stuatt %>% arrange(sid, school_year) %>%
  select(sid, school_year, male) %>%
  filter(sid %in% idx)
```

	sid	school_year	male
1	2	2006	0
2	2	2007	0
3	8552	2005	1
4	8552	2006	0
5	8552	2006	1
6	8552	2007	1
7	8552	2009	1
8	12506	2004	1
9	12506	2005	0

1. Create a variable that shows how many unique values male assumes for each student. Name this variable `nvals_male`. Tabulate the variable and browse the relevant data.

```
# Create an intermediate variable that counts the number of unique
# values observed for `male` per student
```

```
stuatt <- stuatt %>% group_by(sid) %>%
  mutate(nvals_male = length(unique(male))) %>% ungroup()
table(stuatt$nvals_male)
```

1	2
87517	17

```
# Look at the values where more than one value is observed
stuatt %>% select(sid, school_year, male, nvals_male) %>%
  filter(nvals_male > 1)
```

```
# A tibble: 17 × 4
  sid school_year male nvals_male
  <dbl>      <dbl> <dbl>     <int>
1     7         2004     1         2
2     7         2005     1         2
3     7         2006     1         2
4     7         2007     0         2
5     7         2008     1         2
6  8078         2004     1         2
7  8078         2005     0         2
8  8078         2006     1         2
9  8078         2007     1         2
10 8078         2008     1         2
11 8552         2005     1         2
12 8552         2006     0         2
13 8552         2006     1         2
14 8552         2007     1         2
15 8552         2009     1         2
16 12506        2004     1         2
17 12506        2005     0         2
```

```
# Or interactively in RStudio
# stuatt %>% select(sid, school_year, male, nvals_male) %>%
#   filter(nvals_male > 1) %>% View
```

2. Identify the modal gender. If multiple modes exist for a student, report the most recent gender recorded

```
# Step 2: Identify the modal gender, if multiple modes exist, report the most
# recent gender
```

```
# Here is an example mode function in R that mimics Stata
# We can read this function in or load it from another package
# library(eeptools)
# statamode creates a list of the modal values and assigns NA, missing,
# if more than one mode exists
```

```
statamode <- function(x) {
  z <- table(as.vector(x))
  m <- names(z)[z == max(z)]
  if(length(m) == 1){
    if(class(x) %in% c("numeric", "integer", "logical")){
      class(m) <- class(x)
    } else {
      class(m) <- "character"
    }
    return(m)
  }
  return(NA)
}
```

```
# Apply statamode to the data grouped by sid
stuatt <- stuatt %>% group_by(sid) %>%
```

```

mutate(nvals_male = length(unique(male)),
       male_mode = statamode(male)) %>% ungroup()

# Check our work
stuatt %>% select(sid, male, male_mode, nvals_male) %>%
  filter(sid %in% idx)

# A tibble: 9 × 4
   sid  male male_mode nvals_male
<dbl> <dbl>   <dbl>       <int>
1     2     0         0           1
2     2     0         0           1
3 8552     1         1           2
4 8552     0         1           2
5 8552     1         1           2
6 8552     1         1           2
7 8552     1         1           2
8 12506     1        NA           2
9 12506     0        NA           2

# Replace male with male_mode where male_mode is not missing
stuatt$male[!is.na(stuatt$male_mode)] <-
  stuatt$male_mode[!is.na(stuatt$male_mode)]

# In R we replace by vector so both sides of the <- have to have the same filter
# so they are the same length, otherwise R will recycle the elements on the
# right hand side and we will have the wrong values in place

idx <- c(8552, 12506)

stuatt %>% select(sid, school_year, male, nvals_male, male_mode) %>%
  filter(sid %in% idx)

# A tibble: 7 × 5
   sid school_year  male nvals_male male_mode
<dbl>   <dbl> <dbl>       <int>   <dbl>
1 8552     2005     1           2         1
2 8552     2006     1           2         1
3 8552     2006     1           2         1
4 8552     2007     1           2         1
5 8552     2009     1           2         1
6 12506     2004     1           2        NA
7 12506     2005     0           2        NA

# If multiple modes exist, report the most recent gender recorded
stuatt %<>% arrange(sid, school_year) %>%
  group_by(sid) %>%
  mutate(temp_male_last = male[school_year == max(school_year)])

# Show sid 12506
stuatt %>% select(sid, school_year, male, nvals_male, male_mode, temp_male_last) %>%
  filter(sid == 12506)

```

Source: local data frame [2 x 6]  
Groups: sid [1]



```

      sid school_year  male nvals_male male_mode temp_male_last
<dbl>      <dbl> <dbl>      <int>      <dbl>      <dbl>
1 12506        2004     1          2         NA          0
2 12506        2005     0          2         NA          0

# Assign temp_male_last to the male variable in cases where no mode exists
stuatt$male[is.na(stuatt$male_mode)] <- stuatt$temp_male_last[is.na(stuatt$male_mode)]

# Check our work again
stuatt %>% select(sid, school_year, male, nvals_male, male_mode, temp_male_last) %>%
  filter(sid == 12506)

```

Source: local data frame [2 x 6]  
Groups: sid [1]

```

      sid school_year  male nvals_male male_mode temp_male_last
<dbl>      <dbl> <dbl>      <int>      <dbl>      <dbl>
1 12506        2004     0          2         NA          0
2 12506        2005     0          2         NA          0

# Drop temporary variables
stuatt %<>% select(-nvals_male, -male_mode, -temp_male_last)

```

Let's check we got it right

```
table(stuatt$male)
```

```

  0     1
43660 43874

```

```

# Check nvals without creating the variable
stuatt %>% ungroup %>%
  group_by(sid) %>%
  summarize(nvals = nvals(male)) %>% select(nvals) %>%
  table

```

```

.
  1
21803

```

```
nvals(stuatt$sid)
```

```
[1] 21803
```

## Step 2

Create one consistent value for race\_ethnicity for each student across years

1. Recode the raw `race_ethnicity` variable as a numeric variable and label it. Replace the string `race_ethnicity` variable with the numeric one.
  - 1 = African American, not Hispanic
  - 2 = Asian American
  - 3 = Hispanic
  - 4 = American Indian
  - 5 = White, not Hispanic
  - 6 = Multiple / Other

```
# When R reads in Stata files using haven it creates a data type called
# labelled, for compatibility with Stata and most R functions, we convert
# this into a more standard factor variable
```

```
# Create a copy
stuatt$race_num <- stuatt$race_ethnicity
stuatt$race_ethnicity <- as_factor(stuatt$race_ethnicity)
table(stuatt$race_ethnicity) #check current values
```

```
      A      B      H      M/O      NA      W
7303 25321 30444 2809 1129 20528
```

```
stuatt$race_num <- NA
stuatt$race_num[stuatt$race_ethnicity=='B'] <- 1
stuatt$race_num[stuatt$race_ethnicity=='A'] <- 2
stuatt$race_num[stuatt$race_ethnicity=='H'] <- 3
stuatt$race_num[stuatt$race_ethnicity=='NA'] <- 4
stuatt$race_num[stuatt$race_ethnicity=='W'] <- 5
stuatt$race_num[stuatt$race_ethnicity=='M/O'] <- 6
table(stuatt$race_num)
```

```
      1      2      3      4      5      6
25321 7303 30444 1129 20528 2809
```

```
idx <- c(8552)
```

```
stuatt %>% filter(sid %in% idx) %>%
  select(sid, school_year, race_ethnicity, race_num)
```

Source: local data frame [5 x 4]

Groups: sid [1]

	sid	school_year	race_ethnicity	race_num
	<dbl>	<dbl>	<fctr>	<dbl>
1	8552	2005	W	5
2	8552	2006	A	2
3	8552	2006	W	5
4	8552	2007	W	5
5	8552	2009	W	5

```
# In R categorical variables are best represented as factors
# Factors can have values, and labels
# Create a labeled factor for the new race_num variable
stuatt$race_num2 <- factor(stuatt$race_num,
                           labels = c('Black', 'Asian', 'Hispanic',
                                       'Native American', 'White', 'MultipleOther'))
```

```
# Compare them to check using a cross-tabulation
```

```
table(stuatt$race_ethnicity, stuatt$race_num2)
```

	Black	Asian	Hispanic	Native American	White	MultipleOther
A	0	7303	0		0	0
B	25321	0	0		0	0

```
H      0      0      30444      0      0      0
M/O    0      0      0      0      0      2809
NA     0      0      0      1129    0      0
W      0      0      0      0      20528    0
```

```
# Replace them
stuatt$race_ethnicity <- stuatt$race_num2
stuatt$race_num2 <- NULL

table(stuatt$race_ethnicity) # counts
```

```
Black      Asian      Hispanic Native American      White      MultipleOther
25321      7303      30444      1129      20528      2809
```

```
prop.table(table(stuatt$race_ethnicity))*100 #percentages
```

```
Black      Asian      Hispanic Native American      White      MultipleOther
28.927045   8.343044   34.779628   1.289785   23.451459   3.209039
```

Check: What does the distribution of your `race_ethnicity` variable look like? Let's redraw the tables above in a more readable format.

```
library(pander) # library to beautify output
pander(prop.table(table(stuatt$race_ethnicity))*100, style = "rmarkdown")
```

Table 1: Table continues below

Black	Asian	Hispanic	Native American	White
28.93	8.343	34.78	1.29	23.45

MultipleOther
3.209

```
pander(table(stuatt$race_ethnicity), style = "rmarkdown")
```

Table 3: Table continues below

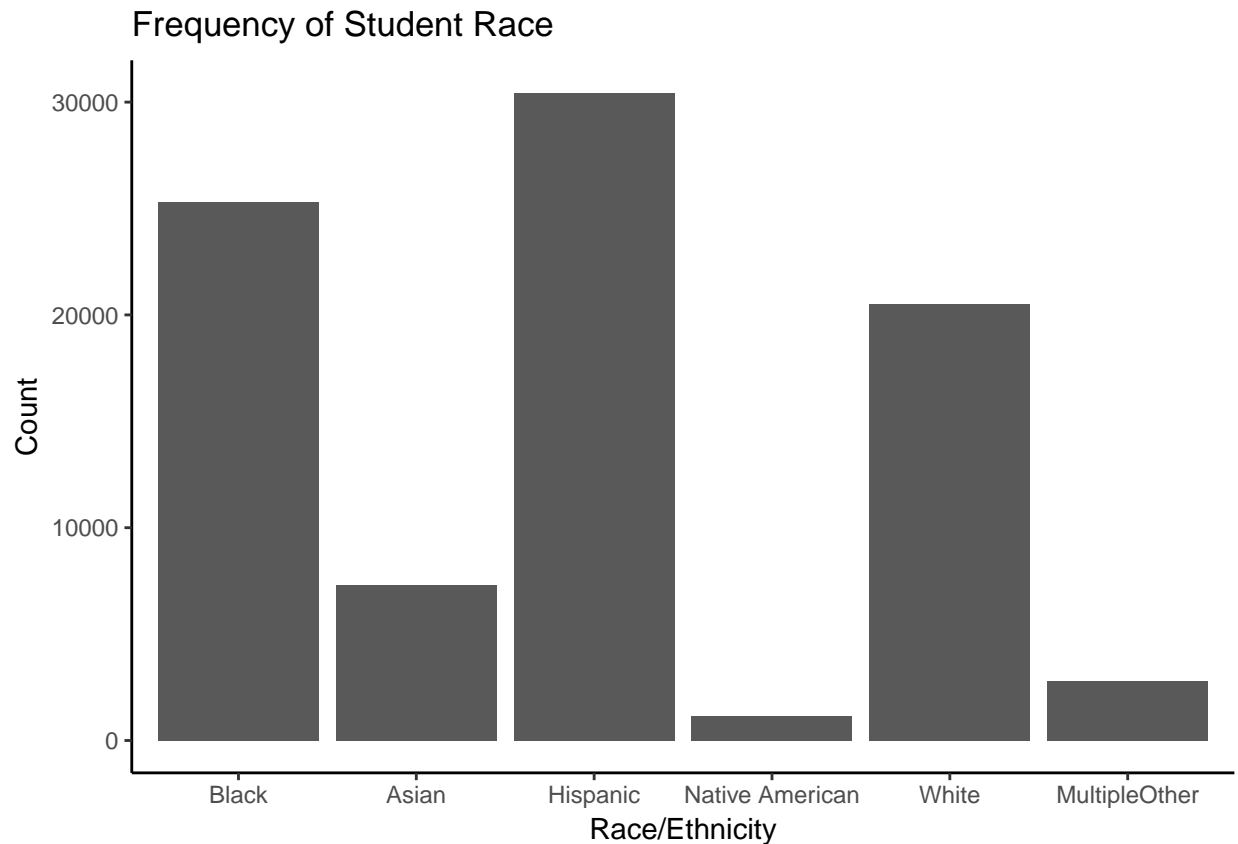
Black	Asian	Hispanic	Native American	White
25321	7303	30444	1129	20528

MultipleOther
2809

Let's also draw a figure to show this distribution.

```
library(ggplot2) # the best R library for plotting
qplot(stuatt$race_ethnicity, geom='bar') +
```

```
theme_classic() + labs(x = 'Race/Ethnicity', y = 'Count',
                        title = "Frequency of Student Race")
```



2. Create a variable indicating how many unique values *race\_ethnicity* assumes for each student called *nvals\_race*.

```
# Create a variable indicating how many unique values `race_ethnicity` takes
# for each student
```

```
stuatt <- stuatt %>% group_by(sid) %>%
  mutate(nvals_race = nvals(race_ethnicity))
```

```
table(stuatt$nvals_race)
```

```
1      2      3
87176 328    30
```

3. Create a variable that shows how many unique values *race\_ethnicity* assumes for each student and *school\_year*. Name this variable *nvals\_race\_yr*. Tabulate the variable and browse the relevant data.

```
# Create a variable that shows how many unique values `race_ethnicity`
# assumes for each student and school year.
```

```
stuatt <- stuatt %>% group_by(sid, school_year) %>%
  mutate(nvals_race_yr = nvals(race_ethnicity))
```

```
#Make a table
```

```
table(stuatt$nvals_race_yr)
```

```

  1      2
87528    6

```

```
# Browse the results
```

```
stuatt %>% select(sid, school_year, race_ethnicity, nvals_race, nvals_race_yr) %>%
  filter(nvals_race_yr > 1)
```

Source: local data frame [6 x 5]

Groups: sid, school\_year [3]

	sid	school_year	race_ethnicity	nvals_race	nvals_race_yr
	<dbl>	<dbl>	<fctr>	<int>	<int>
1	3	2006	Hispanic	2	2
2	3	2006	Black	2	2
3	8552	2006	Asian	2	2
4	8552	2006	White	2	2
5	11382	2005	Hispanic	2	2
6	11382	2005	MultipleOther	2	2

4. If more than one race is reported in the same school\_year, report students as multiracial, unless one of their reported race\_ethnicity values is Hispanic. Report the student as Hispanic in that case.

```
# Generate a temporary hispanic variable
```

```
# Use ifelse function to recode variable
```

```
stuatt$temp_ishispanic <- ifelse(stuatt$race_num == 3 &
  stuatt$nvals_race_yr > 1, 1, 0)
```

```
stuatt %>% select(sid, school_year, race_ethnicity, nvals_race,
  nvals_race_yr, temp_ishispanic) %>%
  filter(nvals_race_yr > 1)
```

Source: local data frame [6 x 6]

Groups: sid, school\_year [3]

	sid	school_year	race_ethnicity	nvals_race	nvals_race_yr	temp_ishispanic
	<dbl>	<dbl>	<fctr>	<int>	<int>	<dbl>
1	3	2006	Hispanic	2	2	1
2	3	2006	Black	2	2	0
3	8552	2006	Asian	2	2	0
4	8552	2006	White	2	2	0
5	11382	2005	Hispanic	2	2	1
6	11382	2005	MultipleOther	2	2	0

```
# Take the maximum value of temp_ishispanic by student by school_year
```

```
# This is creating a variable indicating if the student was ever
```

```
# listed as hispanic in a given school year
```

```
stuatt %<>% group_by(sid, school_year) %>%
  mutate(ishispanic = max(temp_ishispanic, na.rm=TRUE))
```

```
stuatt %>% select(sid, school_year, race_ethnicity, nvals_race, nvals_race_yr,
  temp_ishispanic, ishispanic) %>%
```

```
filter(nvals_race_yr > 1)
```

Source: local data frame [6 x 7]

Groups: sid, school\_year [3]

	sid	school_year	race_ethnicity	nvals_race	nvals_race_yr	temp_ishispanic	ishispanic
	<dbl>	<dbl>	<fctr>	<int>	<int>	<dbl>	<dbl>
1	3	2006	Hispanic	2	2	1	1
2	3	2006	Black	2	2	0	1
3	8552	2006	Asian	2	2	0	0
4	8552	2006	White	2	2	0	0
5	11382	2005	Hispanic	2	2	1	1
6	11382	2005	MultipleOther	2	2	0	1

```
# Replace hispanic values
```

```
stuatt$race_num[stuatt$nvals_race_yr > 1 & stuatt$ishispanic == 1] <- 3
stuatt$race_ethnicity[stuatt$nvals_race_yr > 1 & stuatt$ishispanic == 1] <- "Hispanic"
stuatt$race_num[stuatt$nvals_race_yr > 1 & stuatt$ishispanic != 1] <- 6
stuatt$race_ethnicity[stuatt$nvals_race_yr > 1 & stuatt$ishispanic != 1] <- "MultipleOther"
```

```
# Drop the temporary variables
```

```
stuatt <- select(stuatt, -ishispanic, -temp_ishispanic)
```

```
# Drop the duplicates resulting from fixing student with different race_ethnicity
# within a school year
```

```
# bind_rows allows us to bind two data frames with the same columns together
# The first data.frame will be all rows where the student-school_year race
# is consistent
# The second data.frame is all rows where student race varies by school_year,
# but we have corrected it and drop all duplicated rows using the distinct
# command
```

```
#NROW 87534
```

```
stuatt <- bind_rows(stuatt %>% filter(nvals_race_yr < 2),
                    stuatt %>% filter(nvals_race_yr > 1) %>%
                      distinct(sid, school_year, race_ethnicity, .keep_all=TRUE))
stuatt <- select(stuatt, -nvals_race_yr)
```

```
# Re arrange after binding the rows
stuatt %<>% arrange(sid, school_year)
```

```
# Before we fixed the data we had 87534 rows
# We had 3 students with 2 different races, so we had 6 rows where we needed 3
# This means we had 3 extra rows
```

```
nrow(stuatt) == 87534 - 3
```

```
[1] TRUE
```

5 Report the modal race. If multiple modes exist for a student, report the most recent race recorded.

```
# Calculate the modal race for a student over time, if multiple modes exist
# report the most recent
```

```
stuatt %<>% group_by(sid) %>%
  mutate(race_mode = statamode(race_ethnicity))
```

```
# tab1 <- table(modes$race_temp,modes$nvals)
# addmargins(tab1, FUN=list(Total=sum), quiet=TRUE)
```

```
stuatt %>% filter(sid == 8552) %>%
  select(sid, school_year, race_ethnicity, nvals_race, race_mode)
```

Source: local data frame [4 x 5]

Groups: sid [1]

	sid	school_year	race_ethnicity	nvals_race	race_mode
	<dbl>	<dbl>	<fctr>	<int>	<chr>
1	8552	2005	White	2	White
2	8552	2006	MultipleOther	2	White
3	8552	2007	White	2	White
4	8552	2009	White	2	White

```
stuatt$race_ethnicity[!is.na(stuatt$race_mode)] <- stuatt$race_mode[!is.na(stuatt$race_mode)]
```

```
stuatt %>% filter(sid == 8552) %>%
  select(sid, school_year, race_ethnicity, nvals_race, race_mode)
```

Source: local data frame [4 x 5]

Groups: sid [1]

	sid	school_year	race_ethnicity	nvals_race	race_mode
	<dbl>	<dbl>	<fctr>	<int>	<chr>
1	8552	2005	White	2	White
2	8552	2006	White	2	White
3	8552	2007	White	2	White
4	8552	2009	White	2	White

```
# Consider cases where the mode is not unique
```

```
stuatt %>% filter(sid == 2) %>%
  select(sid, school_year, race_ethnicity, nvals_race, race_mode)
```

Source: local data frame [2 x 5]

Groups: sid [1]

	sid	school_year	race_ethnicity	nvals_race	race_mode
	<dbl>	<dbl>	<fctr>	<int>	<chr>
1	2	2006	White	2	<NA>
2	2	2007	Black	2	<NA>

```
# Define the most recent value of race observed
```

```
stuatt %>% group_by(sid) %>%
  mutate(race_last = race_ethnicity[school_year == max(school_year)])

stuatt %>% filter(sid == 2) %>%
  select(sid, school_year, race_ethnicity, nvals_race, race_mode, race_last)
```

Source: local data frame [2 x 6]

Groups: sid [1]

	sid	school_year	race_ethnicity	nvals_race	race_mode	race_last
	<dbl>	<dbl>	<fctr>	<int>	<chr>	<fctr>
1	2	2006	White	2	<NA>	Black

```

2      2      2007      Black      2      <NA>      Black
stuatt$race_ethnicity[is.na(stuatt$race_mode)] <- stuatt$race_last[is.na(stuatt$race_mode)]

stuatt %>% filter(sid %in% c(8552, 2)) %>%
  select(sid, school_year, race_ethnicity)

```

Source: local data frame [6 x 3]  
Groups: sid [2]

	sid	school_year	race_ethnicity
	<dbl>	<dbl>	<fctr>
1	2	2006	Black
2	2	2007	Black
3	8552	2005	White
4	8552	2006	White
5	8552	2007	White
6	8552	2009	White

```

# Drop temporary variables
stuatt %<>% select(-nvals_race, -race_mode, -race_last, -race_num)

```

```
table(stuatt$race_ethnicity)
```

Black	Asian	Hispanic	Native American	White	Multiple	Other
25323	7262	30443	1132	20553		2818

### Step 3

Create consistent values for high school diploma variables.

1. Recode the `hs_diploma_type` variable as a numeric variable and label it. Replace the string `hs_diploma_type` variable with the numeric one. Use lower numbers for more competitive diploma types.

```

# 1. Recode the `hs_diploma_type` variable` as a numeric variable and label it.
# Replace the string `hs_diploma_type` variable with the numeric one. Use lower
# numbers for more competitive diploma types.

```

```

# In R a factor variable behaves like a labeled numeric variable in Stata
# When reading the data in from a .dta file we can recover the numeric
# labels and ordering by using the `as_factor` function
stuatt$dipl_num <- as_factor(stuatt$hs_diploma_type)

```

```

# To show the work this saves if the data has already been labeled,
# the alternative method for manually recreating this is shown below
stuatt$dipl_num <- 4
stuatt$dipl_num <- ifelse(stuatt$hs_diploma_type == "College Prep Diploma",
  1, stuatt$dipl_num)
stuatt$dipl_num <- ifelse(stuatt$hs_diploma_type == "Standard Diploma",
  2, stuatt$dipl_num)
stuatt$dipl_num <- ifelse(stuatt$hs_diploma_type == "Alternative Diploma",
  3, stuatt$dipl_num)

stuatt %>% select(sid, school_year, hs_diploma, hs_diploma_date,

```



```
      hs_diploma_type, dipl_num) %>%
filter(sid == 16)
```

Source: local data frame [2 x 6]

Groups: sid [1]

	sid	school_year	hs_diploma	hs_diploma_date	hs_diploma_type	dipl_num
	<dbl>	<dbl>	<dbl>	<date>	<chr>	<dbl>
1	16	2007	1	2008-05-14	Standard Diploma	2
2	16	2008	1	2008-05-14	College Prep Diploma	1

```
stuatt$hs_diploma_type <- NULL
stuatt$hs_diploma_type <- stuatt$dipl_num
stuatt$dipl_num <- NULL
```

```
stuatt %>% select(sid, school_year, hs_diploma, hs_diploma_date,
                 hs_diploma_type) %>%
filter(sid == 16)
```

Source: local data frame [2 x 5]

Groups: sid [1]

	sid	school_year	hs_diploma	hs_diploma_date	hs_diploma_type
	<dbl>	<dbl>	<dbl>	<date>	<dbl>
1	16	2007	1	2008-05-14	2
2	16	2008	1	2008-05-14	1

*# Now identify the first diploma date reported*

```
stuatt %<>% arrange(sid, hs_diploma_date)
```

```
stuatt %<>% group_by(sid) %>%
mutate(earliest_diploma_date = min(hs_diploma_date, na.rm=TRUE))
```

```
stuatt %>% select(sid, school_year, hs_diploma, hs_diploma_date,
                 hs_diploma_type, earliest_diploma_date) %>%
filter(sid == 16)
```

Source: local data frame [2 x 6]

Groups: sid [1]

	sid	school_year	hs_diploma	hs_diploma_date	hs_diploma_type	earliest_diploma_date
	<dbl>	<dbl>	<dbl>	<date>	<dbl>	<date>
1	16	2007	1	2008-05-14	2	2008-05-14
2	16	2008	1	2008-05-14	1	2008-05-14

*# 3 Create a variable that shows the earliest diploma type*

*## This won't work because there may be ties*

```
# stuatt %<>% group_by(sid) %>%
```

```
# mutate(earliest_dipl_type = hs_diploma_type[hs_diploma_date==earliest_diploma_date])
```

```
stuatt %<>% group_by(sid) %>%
mutate(earliest_dipl_type_mode = statamode(hs_diploma_type[hs_diploma_date==earliest_diploma_date]))
```

```
stuatt %>% select(sid, school_year, hs_diploma, hs_diploma_date,
                 hs_diploma_type, earliest_diploma_date, earliest_dipl_type_mode) %>% filter(sid == 16)
```

Source: local data frame [2 x 7]

Groups: sid [1]

	sid	school_year	hs_diploma	hs_diploma_date	hs_diploma_type	earliest_diploma_date	earliest_dipl_type
	<dbl>	<dbl>	<dbl>	<date>	<dbl>	<date>	
1	16	2007	1	2008-05-14	2	2008-05-14	
2	16	2008	1	2008-05-14	1	2008-05-14	

```
#4 Create a variable that shows the number of unique diploma types recorded  
# for the first diploma date
```

```
stuatt %<>% group_by(sid) %>%  
  mutate(nvals_dipl_type =  
    length(unique(hs_diploma_type[hs_diploma_date==earliest_diploma_date])))  
  
stuatt %>% select(sid, school_year, hs_diploma_type, earliest_diploma_date,  
  earliest_dipl_type_mode, nvals_dipl_type) %>%  
  filter(sid %in% c(16, 20, 80))
```

Source: local data frame [8 x 6]

Groups: sid [3]

	sid	school_year	hs_diploma_type	earliest_diploma_date	earliest_dipl_type_mode	nvals_dipl_type
	<dbl>	<dbl>	<dbl>	<date>	<dbl>	<int>
1	16	2007	2	2008-05-14	NA	2
2	16	2008	1	2008-05-14	NA	2
3	20	2008	2	2008-05-14	NA	2
4	20	2008	1	2008-05-14	NA	2
5	80	2005	1	2008-05-14	2	2
6	80	2006	2	2008-05-14	2	2
7	80	2007	2	2008-05-14	2	2
8	80	2008	2	2008-05-14	2	2

```
# 5. Identify the modal diploma type. If multiple modes exist for a  
# student, report the diploma type in the earliest school year for  
# the first diploma date
```

```
stuatt %<>% group_by(sid) %>%  
  mutate(earliest_dipl_type_syear = hs_diploma_type[school_year == min(school_year)])  
  
stuatt %>% select(sid, school_year, hs_diploma_type, earliest_diploma_date,  
  earliest_dipl_type_mode, nvals_dipl_type,  
  earliest_dipl_type_syear) %>%  
  filter(sid %in% c(16, 20, 80))
```

Source: local data frame [8 x 7]

Groups: sid [3]

	sid	school_year	hs_diploma_type	earliest_diploma_date	earliest_dipl_type_mode	nvals_dipl_type
	<dbl>	<dbl>	<dbl>	<date>	<dbl>	<int>
1	16	2007	2	2008-05-14	NA	2
2	16	2008	1	2008-05-14	NA	2
3	20	2008	2	2008-05-14	NA	2
4	20	2008	1	2008-05-14	NA	2
5	80	2005	1	2008-05-14	2	2

	sid	school_year	hs_diploma_type	earliest_diploma_date	earliest_dipl_type_mode	nvals_dipl_type
6	80	2006	2	2008-05-14	2	2
7	80	2007	2	2008-05-14	2	2
8	80	2008	2	2008-05-14	2	2

# ... with 1 more variables: earliest\_dipl\_type\_year <dbl>

```

stuatt %<>% group_by(sid) %>%
  mutate(earliest_dipl_type_year_mode = statamode(earliest_dipl_type_year))

stuatt %>%
  select(sid, school_year, hs_diploma_type, earliest_diploma_date,
         earliest_dipl_type_mode, nvals_dipl_type,
         earliest_dipl_type_year,
         earliest_dipl_type_year_mode) %>%
  filter(sid %in% c(16, 20, 80))

```

Source: local data frame [8 x 8]  
Groups: sid [3]

	sid	school_year	hs_diploma_type	earliest_diploma_date	earliest_dipl_type_mode	nvals_dipl_type
	<dbl>	<dbl>	<dbl>	<date>	<dbl>	<int>
1	16	2007	2	2008-05-14	NA	2
2	16	2008	1	2008-05-14	NA	2
3	20	2008	2	2008-05-14	NA	2
4	20	2008	1	2008-05-14	NA	2
5	80	2005	1	2008-05-14	2	2
6	80	2006	2	2008-05-14	2	2
7	80	2007	2	2008-05-14	2	2
8	80	2008	2	2008-05-14	2	2

# ... with 2 more variables: earliest\_dipl\_type\_year <dbl>, earliest\_dipl\_type\_year\_mode <dbl>

*# 6. If multiple diploma types were recorded for the same school year and first  
# diploma date, report the most competitive diploma type*

```

stuatt %<>% group_by(sid) %>%
  mutate(temp_most_compet = min(earliest_dipl_type_year))

stuatt %>%
  select(sid, school_year, hs_diploma_type, earliest_diploma_date,
         earliest_dipl_type_mode, nvals_dipl_type,
         earliest_dipl_type_year,
         earliest_dipl_type_year_mode, temp_most_compet) %>%
  filter(sid %in% c(16, 20, 80)) %>% as.data.frame()

```

	sid	school_year	hs_diploma_type	earliest_diploma_date	earliest_dipl_type_mode	nvals_dipl_type
1	16	2007	2	2008-05-14	NA	2
2	16	2008	1	2008-05-14	NA	2
3	20	2008	2	2008-05-14	NA	2
4	20	2008	1	2008-05-14	NA	2
5	80	2005	1	2008-05-14	2	2
6	80	2006	2	2008-05-14	2	2
7	80	2007	2	2008-05-14	2	2
8	80	2008	2	2008-05-14	2	2

	earliest_dipl_type_year	earliest_dipl_type_year_mode	temp_most_compet
1	2	2	2
2	2	2	2
3	2	NA	1

```

4           1           NA           1
5           1           1           1
6           1           1           1
7           1           1           1
8           1           1           1

# Replace original diploma type variable starting with most specific case, and
# working backward

```

```

stuatt$ivals_dipl_type <- NULL

stuatt$hs_diploma_type[!is.na(stuatt$temp_most_compet)] <-
  stuatt$temp_most_compet[!is.na(stuatt$temp_most_compet)]

stuatt$hs_diploma_type[!is.na(stuatt$earliest_dipl_type_syear_mode)] <-
  stuatt$earliest_dipl_type_syear_mode[!is.na(stuatt$earliest_dipl_type_syear_mode)]

stuatt$hs_diploma_type[!is.na(stuatt$earliest_dipl_type_mode)] <-
  stuatt$earliest_dipl_type_mode[!is.na(stuatt$earliest_dipl_type_mode)]

stuatt %>%
  select(sid, school_year, hs_diploma_type, earliest_diploma_date,
         earliest_dipl_type_mode,
         earliest_dipl_type_syear,
         earliest_dipl_type_syear_mode, temp_most_compet) %>%
  filter(sid %in% c(16, 20, 80)) %>% as.data.frame()

```

	sid	school_year	hs_diploma_type	earliest_diploma_date	earliest_dipl_type_mode	earliest_dipl_type_syear
1	16	2007	2	2008-05-14	NA	
2	16	2008	2	2008-05-14	NA	
3	20	2008	1	2008-05-14	NA	
4	20	2008	1	2008-05-14	NA	
5	80	2005	2	2008-05-14	2	
6	80	2006	2	2008-05-14	2	
7	80	2007	2	2008-05-14	2	
8	80	2008	2	2008-05-14	2	

	earliest_dipl_type_syear_mode	temp_most_compet
1	2	2
2	2	2
3	NA	1
4	NA	1
5	1	1
6	1	1
7	1	1
8	1	1

```

# 7. If there are any missing diploma types, mark these as an unknown
# diploma type

```

```

stuatt$hs_diploma_type[is.na(stuatt$hs_diploma_type) &
  !is.na(stuatt$hs_diploma_date)] <- 4

```

```

# 8. Finally, replace hs_diploma_date with the first hs_diploma_date

```

```

stuatt$hs_diploma_date <- stuatt$earliest_diploma_date

```

```

# 9. Make sure that diploma is set to 1 if there is a diploma date reported

stuatt$hs_diploma[!is.na(stuatt$hs_diploma_date)] <- 1

# 10. Drop all temporary variables we created

stuatt %<>% select(-earliest_diploma_date, -earliest_dipl_type_mode,
                 -earliest_dipl_type_syear, -earliest_dipl_type_syear_mode,
                 -temp_most_compet)

stuatt %>%
  select(sid, school_year, hs_diploma_type) %>%
  filter(sid %in% c(16, 20, 80)) %>% as.data.frame()

```

	sid	school_year	hs_diploma_type
1	16	2007	2
2	16	2008	2
3	20	2008	1
4	20	2008	1
5	80	2005	2
6	80	2006	2
7	80	2007	2
8	80	2008	2

Step 4: Drop any unneeded variables, drop duplicates, check the data, and save the file

```

# Drop school year as you no longer need it
stuatt %<>% select(-school_year, -birth_date)

# Drop duplicate values

tmp <- stuatt[!duplicated(stuatt),]

# Check that the file is unique by sid

nrow(tmp) == length(unique(stuatt$sid))

```

```
[1] TRUE
```

```

# Deduplicate

rm(tmp)
stuatt <- stuatt[!duplicated(stuatt),]

# Save the current file as Student_Attributes.rda

# save(stuatt, file = "Student_Attributes.rda")

# Clean up the workspace
rm(con, idx, tmpfileName, stuatt)

```

## Task 2: STUDENT SCHOOL YEAR

### PURPOSE

In Task 2: Student School Year, you will take the `Student_Classifications_Raw` file and generate a clean `Student_School_Year` output file that matches the specification in Identify with one observation per student and school year. To do so, you will first ensure only one grade level is assigned per student per school year. Then, you will process the free or reduced price lunch (FRPL) variable (a proxy for students' poverty status), individualized education program (IEP) variable, English language learner (ELL) variable, and gifted variable. You will also examine the total days enrolled, days absent, and days suspended variables. The core of this task:

1. Resolve instances when students have more than one grade level in a school year
2. Keep the highest value of FR PL reported by student by school year
3. If a student has both "has IE P" and "no IE P" reported in a school year, keep "has IEP"
4. If a student has both "has ELL" and "no ELL" reported in a school year, keep "has ELL"
5. If a student is observed as both gifted eligible and not eligible, report eligible
6. Explore `days_enrolled`, `days_absent` and `days_suspended`
7. Drop duplicate observations to make the file unique by student and school year After this, you will have a data set unique by student and school year that allows you to assign students to the appropriate ninth grade cohort in Task 3.

### HOW TO START

To begin, open the `Student_Classifications_Raw` file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions and data snippets we have provided.

If this is your first time attempting Task 2, start with the provided raw input file. This file teaches you SDP's cleaning methodology and allows you to check answers from a common dataset.

```
# Read in Stata
library(haven) # required for .dta files

# To read data from a zip file we create a connection to the path of the
# zip file
tmpfileName <- "raw/Student_Classifications_Raw.dta"
con <- unz(description = "data/raw.zip", filename = tmpfileName,
           open = "rb")
stuclass <- read_stata(con) # read data in the data subdirectory
glimpse(stuclass)
```

Observations: 88,260

Variables: 10

```
$ sid      <dbl> 1, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 6, 6, 6, 7, 7, 7, 7, 7, 8
$ school_year <dbl> 2004, 2005, 2006, 2007, 2006, 2007, 2006, 2007, 2007, 2005, 2006, 2
$ grade_level <dbl> 9, 9, 10, 11, 10, 11, 10, 8, 9, 11, 10, 9, 3, 2, 8, 9, 10, 8, 9, 1
$ frpl      <chr> "N", "N", "R", "R", "F", "F", "F", "F", "F", "N", "N", "N", "R", "I
$ iep       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0
$ ell       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
$ gifted    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
$ total_days_enrolled <dbl> 210, 210, 210, 210, 172, 172, 228, 228, 228, 184, 184, 184, 209, 2
$ total_days_absent <dbl> 14, 6, 1, 5, 22, 57, 7, 15, 15, 7, 7, 60, 35, 35, 8, 7, 95, 16, 7,
$ days_suspended_out_of_school <dbl> 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 7, 0, 0, 0, 0,
```

## Step 1 Create One Consistent Grade Level for Each Student Within the Same Year

*# Step 1: Create one consistent grade level for each student within the same year*

*# Keep the highest grade\_level when a student has multiple grade levels  
# within the same year*

*# distinct values function*

```
nvals <- function(x){  
  length(unique(x))  
}
```

```
varIdx <- c("sid", "school_year", "grade_level", "nvals_grade",  
           "max_grade_level")
```

```
stuclass %<>% group_by(sid, school_year) %>%  
  mutate(nvals_grade = nvals(grade_level),  
         max_grade_level = max(grade_level))
```

```
stuclass %>% select(one_of(varIdx)) %>%  
  filter(sid == 3)
```

Source: local data frame [3 x 5]

Groups: sid, school\_year [2]

	sid	school_year	grade_level	nvals_grade	max_grade_level
	<dbl>	<dbl>	<dbl>	<int>	<dbl>
1	3	2006	10	1	10
2	3	2007	8	2	9
3	3	2007	9	2	9

```
stuclass$grade_level[stuclass$nvals_grade > 1] <- stuclass$max_grade_level[stuclass$nvals_grade > 1]
```

```
stuclass %>% select(one_of(varIdx)) %>%  
  filter(sid == 3)
```

Source: local data frame [3 x 5]

Groups: sid, school\_year [2]

	sid	school_year	grade_level	nvals_grade	max_grade_level
	<dbl>	<dbl>	<dbl>	<int>	<dbl>
1	3	2006	10	1	10
2	3	2007	9	2	9
3	3	2007	9	2	9

```
stuclass %<>% select(-nvals_grade, -max_grade_level)
```

Step 2: Create one consistent FRPL value for each student in the same student-year

*# 1. Recode raw frpl variable with string type to numeric type*

```
stuclass$frpl_num <- NA  
stuclass$frpl_num[stuclass$frpl == "N"] <- 0  
stuclass$frpl_num[stuclass$frpl == "R"] <- 1
```

```
stuclass$frpl_num[stuclass$frpl == "F"] <- 2
```

```
stuclass %>% select(sid, school_year, grade_level, frpl, frpl_num) %>%
  filter(sid == 80)
```

Source: local data frame [5 x 5]

Groups: sid, school\_year [4]

	sid	school_year	grade_level	frpl	frpl_num
	<dbl>	<dbl>	<dbl>	<chr>	<dbl>
1	80	2005	9	N	0
2	80	2005	9	R	1
3	80	2006	10	N	0
4	80	2007	11	N	0
5	80	2008	12	N	0

```
stuclass$frpl <- NULL
stuclass$frpl <- stuclass$frpl_num
stuclass$frpl_num <- NULL
```

```
stuclass %>% select(sid, school_year, grade_level, frpl) %>%
  filter(sid == 80)
```

Source: local data frame [5 x 4]

Groups: sid, school\_year [4]

	sid	school_year	grade_level	frpl
	<dbl>	<dbl>	<dbl>	<dbl>
1	80	2005	9	0
2	80	2005	9	1
3	80	2006	10	0
4	80	2007	11	0
5	80	2008	12	0

*# 2. Ensure that frpl is consistent by sid and school\_year. In cases where  
# multiple values exist, report the highest value. Follow the same procedure  
# as Step 1 for grade\_level.*

*# Check if there are any cases where different values of frpl status are reported  
# in a year*

```
stuclass %<>% group_by(sid, school_year) %>%
  mutate(nvals_frpl = nvals(frpl))
```

```
table(stuclass$nvals_frpl)
```

1	2	3
87773	430	57

*# Report the highest value of frpl by year for each student, selecting  
# free over reduced over not participating*

```
stuclass %<>% group_by(sid, school_year) %>%
  mutate(highest_frpl = max(frpl))
```



```
stuclass$frpl <- stuclass$highest_frpl
```

```
# Label the values so they are easy to understand
```

```
# drop the temporary values we created
```

```
stuclass %<>% select(-nvals_frpl, -highest_frpl)
```

```
** Step 3: Create one consistent IEP value for each student within the same year. **
```

```
# Follow the same procedure as Step 1 for grade_level.
```

```
# Report the highest value of iep by year for each student,
```

```
# selecting has iep over not iep.
```

```
stuclass %<>% group_by(sid, school_year) %>%  
  mutate(highest_iep = max(iep)) %>%  
  ungroup() %>%  
  mutate(iep = highest_iep) %>%  
  select(-highest_iep)
```

```
# /** Step 4: Create one consistent ELL value for each student within the same year. **/
```

```
# Follow the same procedure as Step 1 for grade_level.
```

```
# // Report the highest value of ell by year for each student, selecting is ell over not ell.
```

```
stuclass %<>% group_by(sid, school_year) %>%  
  mutate(highest_ell = max(ell)) %>%  
  ungroup() %>%  
  mutate(ell = highest_ell) %>%  
  select(-highest_ell)
```

```
# /** Step 5: Create one consistent gifted value for each student within the same year. **/
```

```
# Follow the same procedure as Step 1 for grade_level.
```

```
# // Report the highest value of gifted by year for each student, selecting is enrolled in gifted program
```

```
stuclass %<>% group_by(sid, school_year) %>%  
  mutate(highest_gifted = max(gifted)) %>%  
  ungroup() %>%  
  mutate(gifted = highest_gifted) %>%  
  select(-highest_gifted)
```

```
# /** Step 6: Drop any unneeded variables, drop duplicates, and save the file ***/
```

```
# // 1. Drop duplicate observations
```

```
stuclass <- stuclass[!duplicated(stuclass),]
```

```
# // 2. Make sure your file is now unique by student and school year
```

```
nrow(stuclass) == nvals(paste0(stuclass$sid, stuclass$school_year))
```

```
[1] TRUE
```

```
# // 3. Save the current file as Student_School_Year.dta which you will need for Task 3.
```

```
# save(stuclass, file = "Student_School_Year.rda")
```

```
# Clean up the workspace
```

```
rm(con, tmpfileName, stuclass, varIdx)
```



```
$ total_days_absent          <dbl> 14, 6, 1, 5, 22, 57, 7, 15, 7, 7, 60, 35, 8, 7, 95, 16, 7, 7, 3, 1
$ days_suspended_out_of_school <dbl> 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 11, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0,
```

## - TODO -

Check on p. 35-36 of the guide, inconsistency on whether observed\_g indicator is unique by student, or by student-grade. I've coded it unique by student.

```
# /** Step 1: Flag the first school year a student enrolls in grades 9, 10, 11, or 12. */
# // Create four binary indicators to flag the first school year a student enrolls in grades 9, 10, 11,
```

```
stusy %>% filter(sid == 1) %>% select(sid, school_year, grade_level)
```

```
# A tibble: 4 × 3
```

```
  sid school_year grade_level
<dbl>      <dbl>      <dbl>
1     1         2004          9
2     1         2005          9
3     1         2006         10
4     1         2007         11
```

```
stusy %<>% group_by(sid, grade_level) %>%
  mutate(tmpG = ifelse(school_year == min(school_year), 1, NA),
         observed_g = 1)
```

```
stusy %>% filter(sid == 1) %>% select(sid, school_year, grade_level,
                                     tmpG, observed_g)
```

```
Source: local data frame [4 x 5]
```

```
Groups: sid, grade_level [3]
```

```
  sid school_year grade_level tmpG observed_g
<dbl>      <dbl>      <dbl> <dbl>      <dbl>
1     1         2004          9     1          1
2     1         2005          9    NA          1
3     1         2006         10     1          1
4     1         2007         11     1          1
```

```
# Use tidyr to spread first_flag and observed_g out
```

```
library(tidyr)
```

```
stusy$first_flag <- stusy$grade_level
```

```
stusy <- spread(stusy, key = first_flag, value = tmpG, sep = "") %>%
  select(-one_of("first_flag3", "first_flag5", "first_flag6", "first_flag7",
                 "first_flag8", "first_flag13", "first_flag17"))
```

```
stusy$observed <- stusy$grade_level
```

```
# Fill in a 1 because we want the observed vectors to populate all values
# for a student
```

```
stusy <- spread(stusy, key = observed, value = observed_g, sep = "_") %>%
  select(-one_of("observed_3", "observed_5", "observed_6",
                 "observed_7", "observed_8", "observed_13",
                 "observed_17")) %>%
```

```

group_by(sid) %>%
mutate(observed_9 = max(observed_9, na.rm=TRUE),
       observed_10 = max(observed_10, na.rm=TRUE),
       observed_11 = max(observed_11, na.rm=TRUE),
       observed_12 = max(observed_12, na.rm=TRUE)) %>%
mutate(observed_9 = ifelse(is.finite(observed_9), 1, 0),
       observed_10 = ifelse(is.finite(observed_10), 1, 0),
       observed_11 = ifelse(is.finite(observed_11), 1, 0),
       observed_12 = ifelse(is.finite(observed_12), 1, 0))

# Check how many students are identified as enrolled in grades 9, 10, 11, or 12

tmp <- stusy %>%
  select(num_range(prefix= "observed_", range = 9:12)) %>%
  distinct(.keep_all=TRUE)

table(tmp$observed_9)

```

```

      0      1
2959 18844
table(tmp$observed_10)

```

```

      0      1
6590 15213
table(tmp$observed_11)

```

```

      0      1
12510 9293
table(tmp$observed_12)

```

```

      0      1
16277 5526
rm(tmp)

```

**\*\* Step 2: Identify the school year in which the student was first observed in 9th grade. \*\***

*# // Create a variable that lists the first school year a student is observed as enrolled in grade 9.*

```

stusy %<>% group_by(sid) %>%
  mutate(first_9th_schyear_obs = min(school_year[grade_level == 9]))

# work around weird way R handles minimum of an empty vector

stusy$first_9th_schyear_obs[!is.finite(stusy$first_9th_schyear_obs)] <- NA

stusy %>% filter(sid == 1) %>%
  select(sid, school_year, grade_level, first_flag9, observed_9, first_9th_schyear_obs)

```

Source: local data frame [4 x 6]

Groups: sid [1]

	sid	school_year	grade_level	first_flag9	observed_9	first_9th_schyear_obs
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1	2004	9	1	1	2004
2	1	2005	9	NA	1	2004
3	1	2006	10	NA	1	2004
4	1	2007	11	NA	1	2004

```
stusy %>% ungroup %>% distinct(sid, first_9th_schyear_obs) %>%  
  select(first_9th_schyear_obs) %>% unlist %>% table
```

```
.  
2004 2005 2006 2007 2008 2009  
    1 4884 4405 4524 5018    12
```

*# Say something about missing values in the list...*

**\*\* Step 3: Impute the school year in which transfer students would have been in grade 9. \*\***

*# // Impute first\_9th\_school\_year\_observed as school\_year - 1, school\_year - 2, or school\_year - 3 for*

```
stusy$first_flag10[!is.finite(stusy$first_flag10)] <- 0  
stusy$first_flag11[!is.finite(stusy$first_flag11)] <- 0  
stusy$first_flag12[!is.finite(stusy$first_flag12)] <- 0  
  
stusy$tempfirst9year <- ifelse(stusy$first_flag10 == 1,  
                              stusy$school_year - 1,  
                              ifelse(stusy$first_flag11 == 1,  
                                      stusy$school_year - 2,  
                                      ifelse(stusy$first_flag12 == 1,  
                                              stusy$school_year - 3,  
                                              NA)))  
  
stusy %>% filter(sid == 2) %>%  
  select(sid, school_year, grade_level, first_9th_schyear_obs,  
         tempfirst9year)
```

Source: local data frame [2 x 5]

Groups: sid [1]

	sid	school_year	grade_level	first_9th_schyear_obs	tempfirst9year
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2	2006	10	NA	2005
2	2	2007	11	NA	2005

*# What is up with 2003 in the table here in Stata documentation*

```
stusy %<>% group_by(sid) %>%  
  mutate(tempfirst9year = min(tempfirst9year, na.rm=TRUE))  
  
stusy$first_9th_schyear_obs[is.na(stusy$first_9th_schyear_obs) & !is.na(stusy$tempfirst9year)] <- stusy$tempfirst9year  
  
stusy$tempfirst9year <- NULL  
  
#  
# stusy <- bind_rows(stusy %>% filter(observed_9 == 0),
```

```
#           stusy %<% filter(observed_9 == 1) %>%
#   group_by(sid) %>%
#   mutate(first_9th_schyear_obs = max(first_9th_schyear_obs))
# )
# stusy %<% arrange(sid, school_year)

# // Review the distribution of first_9th_school_year_observed for students who transferred in grades 1

stusy %>% ungroup %>%
  filter(first_flag10 > 0) %>%
  filter(observed_9 == 0) %>%
  distinct(sid, first_9th_schyear_obs) %>%
  select(first_9th_schyear_obs) %>% unlist %>% table
```

```
.
2004 2005 2006 2007 2008
   16  400  321  380  444
```

```
stusy %>% ungroup %>%
  filter(first_flag11 ==1) %>%
  filter(observed_9 == 0 & observed_10 == 0 & observed_11 == 1) %>%
  distinct(sid, first_9th_schyear_obs) %>%
  select(first_9th_schyear_obs) %>% unlist %>% table
```

```
.
2004 2005 2006 2007
    2  288  285  318
```

```
stusy %>% ungroup %>%
  filter(first_flag12 ==1) %>%
  filter(observed_9 == 0 & observed_10 == 0 & observed_11 == 0 &
         observed_12 == 1) %>%
  distinct(sid, first_9th_schyear_obs) %>%
  select(first_9th_schyear_obs) %>% unlist %>% table
```

```
.
2004 2005 2006
    2  137  145
```

```
# Consider max of this?
stusy %>% select(sid, school_year, grade_level,
               first_9th_schyear_obs, observed_9) %>%
  filter(sid == 3)
```

```
Source: local data frame [2 x 5]
Groups: sid [1]
```

	sid	school_year	grade_level	first_9th_schyear_obs	observed_9
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	3	2006	10	2007	1
2	3	2007	9	2007	1

```
# Should this be 2005 or 2007?
```

\*\* Step 4: Adjust the imputation of first\_9th\_school\_year\_observed for students who appear in a lower grade in a later school year. \*\*

```
# // 1. Flag students who are observed to be in a lower grade in a subsequent school year.
```

```
stusy %<>% arrange(sid, school_year) %>%
  group_by(sid) %>%
  mutate(grade_lag = lag(grade_level, order_by = school_year)) %>%
  mutate(grade_flag = ifelse(grade_lag > grade_level & !is.na(grade_lag > grade_level), 1, 0)) %>%
  mutate(grade_flag_max = max(grade_flag, na.rm=TRUE)) %>%
  select(-grade_lag)

stusy %>% select(sid, school_year, grade_level,
                first_9th_schyear_obs, grade_flag, grade_flag_max) %>%
  filter(sid == 3)
```

Source: local data frame [2 x 6]  
Groups: sid [1]

	sid	school_year	grade_level	first_9th_schyear_obs	grade_flag	grade_flag_max
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	3	2006	10	2007	0	1
2	3	2007	9	2007	1	1

```
# // 2. Flag the first school year in which students appear in high school grades
```

```
stusy %<>% group_by(sid) %>%
  mutate(first_9th_flag = ifelse(school_year == min(school_year[grade_level %in% c(9:12)]), 1, 0))

stusy %>% select(sid, school_year, grade_level,
                first_9th_schyear_obs, grade_flag, grade_flag_max,
                first_9th_flag) %>%
  filter(sid == 3)
```

Source: local data frame [2 x 7]  
Groups: sid [1]

	sid	school_year	grade_level	first_9th_schyear_obs	grade_flag	grade_flag_max	first_9th_flag
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	3	2006	10	2007	0	1	1
2	3	2007	9	2007	1	1	0

```
# // 4. Replace the first_9th_school_year_observed with the correctly imputed values.
```

```
# Need to drop NAs
stusy$temp4_first9year <- NA
stusy$temp4_first9year[stusy$grade_flag_max == 1 &
  stusy$first_9th_flag == 1 &
  stusy$grade_level == 10] <- stusy$school_year[stusy$grade_flag_max == 1 &
  stusy$first_9th_flag == 1 &
  stusy$grade_level == 10] - 1

stusy$temp4_first9year[stusy$grade_flag_max == 1 &
  stusy$first_9th_flag == 1 &
  stusy$grade_level == 11] <- stusy$school_year[stusy$grade_flag_max == 1 &
  stusy$first_9th_flag == 1 &
```

```

      stusy$grade_level == 11] - 2

stusy$temp4_first9year[stusy$grade_flag_max == 1 &
  stusy$first_9th_flag == 1 &
  stusy$grade_level == 12] <- stusy$school_year[stusy$grade_flag_max == 1 &
  stusy$first_9th_flag == 1 &
  stusy$grade_level == 12] - 3

stusy %<>% group_by(sid) %>%
  mutate(temp5_first9year = min(temp4_first9year, na.rm=TRUE))

stusy %>% select(sid, school_year, grade_level,
  first_9th_schyear_obs, grade_flag, grade_flag_max,
  first_9th_flag, temp4_first9year, temp5_first9year) %>%
  filter(sid == 3)

```

Source: local data frame [2 x 9]

Groups: sid [1]

	sid	school_year	grade_level	first_9th_schyear_obs	grade_flag	grade_flag_max	first_9th_flag	temp4_fi
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	3	2006	10	2007	0	1	1	
2	3	2007	9	2007	1	1	0	

# ... with 1 more variables: temp5\_first9year <dbl>

```

stusy$first_9th_schyear_obs[stusy$grade_flag_max == 1 &
  !is.na(stusy$temp5_first9year)] <- stusy$temp5_first9year[stusy$grade_flag_max == 1 &
  !is.na(stusy$temp5_first9year)]

stusy %>% ungroup %>% distinct(sid, first_9th_schyear_obs) %>%
  select(first_9th_schyear_obs) %>% unlist %>% table

```

```

.
2002 2003 2004 2005 2006 2007 2008 2009
    4    9   22 5706 5154 5217 5459   11

```

## - TODO Task 3 Slightly Different Numbers than SDP ##-

\*\* Step 5: Keep only variables relevant to future analyses, and save the file. \*\*

```

# // Keep relevant variables

stusy %<>% select(sid, school_year, grade_level, frpl, iep, ell, gifted,
  total_days_enrolled, total_days_absent,
  days_suspended_out_of_school, first_9th_schyear_obs)

# // Save the current file as Student_School_Year_Ninth.dta.
# save "${clean}\\Student_School_Year_Ninth.dta", replace
rm(stusy)

```



## Task 4: STUDENT SCHOOL ENROLLMENT

### PURPOSE

In Task 4: Student School Enrollment, you will take the `Student_School_Enrollment_Raw` file and generate the `Student_School_Enrollment` file that matches the specification in Identify. After matching Identify, you will take your dataset a few steps further by consolidating overlapping enrollment spells and determining the last withdrawal code for each student to yield the file `Student_School_Enrollment_Clean`. The core of this task: 1. Create a `school_start` and `school_end` variable. 2. Remove abnormal enrollment observations with missing enrollment and withdrawal dates along with enrollment or withdrawal dates that are not in the right order. 3. Consolidate overlapping enrollments by student by school. 4. Update `days_enrolled` based on the consolidated enrollments using the new enrollment and withdrawal dates. 5. Determine the last withdrawal code for each student. You will use this data in later analyses to determine a student's end of high school outcomes. After completing this, you will have a clean `Student_School_Enrollment` file. This process sets up our analyses for high school graduation and college enrollment and persistence outcomes.

### HOW TO START

To begin, open the `Student_School_Enrollment_Raw` file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions and data snippets we have provided. If this is your first time attempting Task 4, start with the provided input file. This file teaches you SDP's cleaning methodology and allows you to check answers from a common dataset.

```
# Read in Stata  
library(haven) # required for .dta files
```

```
# To read data from a zip file we create a connection to the path of the  
# zip file  
tmpfileName <- "raw/Student_School_Enrollment_Raw.dta"  
con <- unz(description = "data/raw.zip", filename = tmpfileName,  
           open = "rb")  
stuenr <- read_stata(con) # read data in the data subdirectory  
glimpse(stuenr)
```

Observations: 95,935

Variables: 8

```
$ sid          <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 4, 5, 5, 6, 6, 6, 7, 7, 7,  
$ school_year  <dbl> 2004, 2004, 2005, 2005, 2006, 2006, 2007, 2007, 2006, 2007, 2007, 2006, 200  
$ school_code  <dbl> 486, 486, 485, 486, 485, 485, 485, 485, 486, 486, 486, 480, 480, 475, 470,  
$ enrollment_date <date> 2003-06-25, 2003-08-29, 2005-01-28, 2004-09-02, 2005-08-28, 2005-08-28, 2  
$ enrollment_code_desc <chr> "Grade 9", "Grade 9", "Grade 9", "Grade 9", "Grade 10", "Grade 10", "Grade  
$ withdrawal_date <date> 2003-07-05, 2004-06-08, 2005-06-07, 2005-02-10, 2005-08-28, 2006-06-07, 2  
$ withdrawal_code_desc <chr> "Promoted End Year", "Retained in Grade", "Transfer In District", "Retaine  
$ days_enrolled  <dbl> 10, 284, 130, 161, 0, 283, 284, NaN, 282, 191, 124, 361, 361, -822, 299, 3
```

```
# /** Step 1: Create a school_start and school_end variable **/
```

```
# // In this example, school start is August 1, and school end is July 31 of each school year. This may
```

```
library(lubridate) # handle dates and times in R correctly
```

```
stuenr$school_start <- mdy(paste0("08", "01", stuenr$school_year-1))  
stuenr$school_end <- mdy(paste0("07", "31", stuenr$school_year))
```

```
# In R we have to create a character string that we convert to a date
```

```

# Converting numerics to dates and times can introduce errors

# /** Step 2: Remove abnormal enrollment observations. */
# // 1. Drop observations missing both enrollment and withdrawal dates.

stuenr %<>% filter(!is.na(enrollment_date) & !is.na(withdrawal_date))

# // 2. Drop observations with enrollment and withdrawal dates on same day.

stuenr %<>% filter(!enrollment_date == withdrawal_date & !is.na(enrollment_date))

# // 3. Drop observations with withdrawal date earlier than enrollment date.

stuenr %<>% filter(!is.na(withdrawal_date) & !withdrawal_date < enrollment_date)

# // 4. Drop observations with enrollment date after the end of the current school year.

stuenr %<>% filter(school_end > enrollment_date)

# // 5. Drop observations with enrollment date before the beginning of the current school year.

stuenr %<>% filter(school_start <= enrollment_date)

# // 6. Drop observations with withdrawal date more than one month after the end of the school year.

stuenr %<>% filter(withdrawal_date <= (school_end + 31) & !is.na(withdrawal_date))

# // 7. Check to make sure enrollment dates are in the correct school year.

table(stuenr$enrollment_date >= stuenr$school_start)

```

```

TRUE
93772

```

```

table(stuenr$enrollment_date <= stuenr$school_end)

```

```

TRUE
93772

```

```

/** Step 3: Consolidate overlapping enrollments by student by school. */

```

```

# // 1. Sort enrollment spells in ascending order and then check how many overlapping enrollment spells

```

```

stuenr %<>% arrange(sid, school_code, enrollment_date)

```

```

stuenr %<>% group_by(sid, school_code) %>%
  mutate(lag_withdrawal_date = lag(withdrawal_date)) %>% ungroup %>%
  group_by(sid, school_code, school_year) %>%
  mutate(min_enroll_date = min(enrollment_date))

```

```

# table(stuenr$enrollment_date <= stuenr$lag_withdrawal_date &

```

```

# !is.na(stuenr$lag_withdrawal_date))
# 682?

tmp <- stuenr %>% filter(sid == 2) %>%
  select(sid, school_year, school_code, enrollment_date,
         enrollment_code_desc, withdrawal_date, lag_withdrawal_date,
         withdrawal_code_desc, min_enroll_date)

# // 2. For overlapping observations, replace the enrollment date and enrollment code description of all
# observations with the enrollment date and enrollment code description of the latest withdrawal date.

stuenr$enrollment_date[stuenr$enrollment_date <= stuenr$lag_withdrawal_date &
  !is.na(stuenr$lag_withdrawal_date)] <- stuenr$min_enroll_date[stuenr$enrollment_date <=
  !is.na(stuenr$lag_withdrawal_date)]

stuenr %>% filter(sid == 2) %>%
  select(sid, school_year, school_code, enrollment_date,
         enrollment_code_desc, withdrawal_date,
         withdrawal_code_desc)

```

Source: local data frame [3 x 7]

Groups: sid, school\_code, school\_year [2]

	sid	school_year	school_code	enrollment_date	enrollment_code_desc	withdrawal_date	withdrawal_code_desc
	<dbl>	<dbl>	<dbl>	<date>	<chr>	<date>	<chr>
1	2	2006	486	2005-08-27	Grade 10	2006-06-05	Promoted End Year
2	2	2007	486	2006-08-29	Grade 11	2007-03-08	Retained in Grade
3	2	2007	486	2006-08-29	Grade 11	2007-06-05	Promoted End Year

// 3. Replace the withdrawal date and withdrawal code description of the earliest enrollment spell with the latest withdrawal date.

```

# // Sort the data first so that latest withdrawal
# information appears as the first record.

stuenr %<>% arrange(sid, school_code, enrollment_date, withdrawal_date)

stuenr %>% filter(sid == 2) %>%
  select(sid, school_year, school_code, enrollment_date,
         enrollment_code_desc, withdrawal_date,
         withdrawal_code_desc)

```

Source: local data frame [3 x 7]

Groups: sid, school\_code, school\_year [2]

	sid	school_year	school_code	enrollment_date	enrollment_code_desc	withdrawal_date	withdrawal_code_desc
	<dbl>	<dbl>	<dbl>	<date>	<chr>	<date>	<chr>
1	2	2006	486	2005-08-27	Grade 10	2006-06-05	Promoted End Year
2	2	2007	486	2006-08-29	Grade 11	2007-03-08	Retained in Grade
3	2	2007	486	2006-08-29	Grade 11	2007-06-05	Promoted End Year

```

# // Replace withdrawal_date
# // Replace withdrawal_code_description

stuenr %<>% group_by(sid, school_code, enrollment_date) %>%
  mutate(withdrawal_date = last(withdrawal_date),

```

```
withdrawal_code_desc = last(withdrawal_code_desc))
```

\*\* Step 4: Update days\_enrolled based on the consolidated enrollments using the new enrollment and withdrawal dates. \*\*

```
stuenr$days_enrolled <- stuenr$withdrawal_date - stuenr$enrollment_date

stuenr %>% filter(sid == 2) %>%
  select(sid, school_year, school_code, enrollment_date,
         enrollment_code_desc, withdrawal_date,
         withdrawal_code_desc, days_enrolled)
```

Source: local data frame [3 x 8]

Groups: sid, school\_code, enrollment\_date [2]

	sid	school_year	school_code	enrollment_date	enrollment_code_desc	withdrawal_date	withdrawal_code_desc
	<dbl>	<dbl>	<dbl>	<date>	<chr>	<date>	<chr>
1	2	2006	486	2005-08-27	Grade 10	2006-06-05	Promoted End Year
2	2	2007	486	2006-08-29	Grade 11	2007-06-05	Promoted End Year
3	2	2007	486	2006-08-29	Grade 11	2007-06-05	Promoted End Year

\*\* Step 5: Determine the last withdrawal code for each student. You will use this data in later analyses to determine a student's end of high school outcomes. \*\*

```
stuenr %<>% arrange(sid, withdrawal_date)

stuenr %>% filter(sid == 16) %>%
  select(sid, school_year, school_code, enrollment_date,
         enrollment_code_desc, withdrawal_date,
         withdrawal_code_desc)
```

Source: local data frame [2 x 7]

Groups: sid, school\_code, enrollment\_date [2]

	sid	school_year	school_code	enrollment_date	enrollment_code_desc	withdrawal_date	withdrawal_code_desc
	<dbl>	<dbl>	<dbl>	<date>	<chr>	<date>	<chr>
1	16	2007	450	2007-01-07	Grade 11	2007-05-22	Promoted End Year
2	16	2008	450	2007-08-20	Grade 12	2008-05-28	Graduated with Diploma

*# p.46 withdrawal date wrong in table, or needs to be updated*

```
stuenr %<>% group_by(sid) %>%
  mutate(last_withdrawal_reason = last(withdrawal_code_desc))

stuenr %>% filter(sid == 16) %>%
  select(sid, school_year, school_code, enrollment_date,
         enrollment_code_desc, withdrawal_date,
         withdrawal_code_desc, last_withdrawal_reason)
```

Source: local data frame [2 x 8]

Groups: sid [1]

	sid	school_year	school_code	enrollment_date	enrollment_code_desc	withdrawal_date	withdrawal_code_desc
	<dbl>	<dbl>	<dbl>	<date>	<chr>	<date>	<chr>
1	16	2007	450	2007-01-07	Grade 11	2007-05-22	Promoted End Year
2	16	2008	450	2007-08-20	Grade 12	2008-05-28	Graduated with Diploma

# ... with 1 more variables: last\_withdrawal\_reason <chr>

**\*\* Step 6: Drop any unneeded variables, drop duplicates, and save the file \*\***

```
# // 1. Drop duplicate records

stuenr %<>% select(-min_enroll_date, -lag_withdrawal_date)

stuenr <- ungroup(stuenr) %>% distinct(sid, school_year, school_code,
                                       enrollment_date, .keep_all = TRUE)

# // 2. Confirm that file is unique by student, school_year, school_code, and enrollment_date

nvals(paste0(stuenr$sid, stuenr$school_year, stuenr$school_code,
             stuenr$enrollment_date)) == nrow(stuenr)

[1] TRUE

# // 3. Save the current file as Student_School_Enrollment_Clean
# save("${clean}\\Student_School_Enrollment_Clean.dta", replace
# save(stuenr, file = "clean/Student_School_Enrollment_Clean.rda")
rm(tmp, stuenr); gc()
```

	used (Mb)	gc trigger (Mb)	max used (Mb)
Ncells	663699 35.5	1770749 94.6	1770749 94.6
Vcells	1023151 7.9	6542064 50.0	10212593 78.0

## Task 4: STUDENT TEST SCORES

### PURPOSE

structure of Identify. Through this task, you will generate three different clean output files that contain a single score and test-taking instance for each student: • Prior Achievement (one 8th grade state test score per student), • SAT scores (one SAT score per student), and • ACT scores (one ACT score per student). The file for Prior Achievement will contain students' achievement on state standardized Math and English Language Arts tests in 8th grade. This will allow you to control for prior academic achievement when you examine college-going outcomes. The SAT and ACT score files will be used for defining highly qualified high school graduates. The core of this task: • Prior Achievement 1. Clean state test scores and resolve instances where students took the same test multiple times. 2. Standardize test scores to a mean of 0 and a standard deviation of 1. This allows you to compare across tests and years when different score scales were used. 3. Generate a composite math and English score for scaled and standardized test scores in eighth grade. • SAT 1. Clean SAT test scores and resolve instances where students took the same test multiple times. 2. Generate a total SAT score based on math, verbal, and writing scores. • ACT 1. Clean ACT test scores and resolve instances where students took the same test multiple times. After completing this, you will have a Prior\_Achievement file with 8th grade test scores. You will also have SAT and ACT files. All three files will be used in Connect.

### HOW TO START

the steps of the task by looking at the instructions and data snippets we have provided. If this is your first time attempting Task 5, start with the provided input file. This file teaches you SDP's cleaning methodology and allows you to check answers from a common dataset

## DATA DESCRIPTION

The input file, `Student_Test_Scores`, follows the structure of `Student_Test_Scores` in `Identify` so it is unique by `sid`, `test_code`, and `test_date`. The aim of this task will be to create three separate clean output files, `Prior_Achievement`, `SAT`, and `ACT`, that report only one test score per student. This means that for eight grade prior achievement duplicates of the same test taken in the same and different years will need to be resolved. Also any duplicates of SAT or ACT scores will need to be resolved as well.

Prior Achievement (8th grade state test scores), Ideally, state test data in its raw form is unique by `sid`, `test_subject`, `grade_level`, and `school_year`. However, some students re-take the same test for the same grade in the same year. To fix this, you will make the 8th grade test score data in `Student_Test_Scores` unique by `sid`, `test_subject`, `grade_level`, and `school_year` by removing any same year repeat test taking instances. Then, you will manipulate the data so tests for different subjects in the same `grade_level` fall on the same row so the data is unique by `sid`, `test_subject`, and `grade_level`. As a final step, if a student took the same test in different years (e.g. by repeating a grade), you will take the earliest instance. The data will finally be unique by `sid` and is considered a clean file and ready to be incorporated into the analysis file in `Connect`.

SAT Ideally, SAT test data in its raw form is unique by sid. However, some students re-take the SAT. To fix this, you will take the data unique by sid, test\_subject, and test\_date and reshape it so the data will finally be unique by sid and is considered a clean file and ready to be incorporated into the analysis file in Connect.

ACT Ideally, ACT test data in its raw form is unique by sid. However, some students re-take the ACT. To fix this, you will take the data unique by sid, test\_subject, and test\_date and reshape it so the data will finally be unique by sid and is considered a clean file and ready to be incorporated into the analysis file in Connect.

**\*\* Part I: Clean Prior Achievement Scores \*\***

```
# Read in Stata
library(haven) # required for .dta files

# // 0. Load the Student_Test_Scores data file.
# To read data from a zip file we create a connection to the path of the
# zip file
tmpfileName <- "raw/Student_Test_Scores.dta"
con <- unz(description = "data/raw.zip", filename = tmpfileName,
            open = "rb")
stutest <- read_stata(con) # read data in the data subdirectory
glimpse(stutest)
```

Observations: 100,705

Variables: 10

[illegible]

```
# Convert to R style
stutest$test_subject <- as_factor(stutest$test_subject)
stutest$test_subject <- tolower(as.character(stutest$test_subject))
```

```
# // 1. Keep only the variables you need and limit the sample to state test scores in 8th grade.
```

```
statetest <- statetest %>%
  select(sid, test_type, test_subject, school_year,
         grade_level, scaled_score, raw_score, test_date) %>%
  filter(test_type == "State" & grade_level == 8)
```

```
# Leave the original stutest, we will come back to this later. In R
# we can keep multiple datasets open in the workset at the same time.
```

```
# // 2. Clean up raw and scaled scores.
# // Change raw and scaled scores to missing if zero.
```

```
for(var in c("raw_score", "scaled_score")){
  statetest[, var][statetest[,var] == 0] <- NA
}
```

```
# //Drop observations missing both a raw and scaled test score.
```

```
statetest %<>% filter(!is.na(raw_score) | !is.na(scaled_score))
```

```
# 3. Identify same-year repeat test takers and take the highest test score.
# For ties in scores, take the last date tested
```

```
statetest %<>% arrange(sid, test_subject,
                      grade_level, school_year, scaled_score)
```

```
statetest %>% filter(sid == 595) %>%
  select(sid, test_type, school_year, test_date, grade_level,
         test_subject, scaled_score, raw_score)
```

```
# A tibble: 3 × 8
```

	sid	test_type	school_year	test_date	grade_level	test_subject	scaled_score	raw_score
	<dbl>	<chr>	<dbl>	<date>	<dbl>	<chr>	<dbl>	<dbl>
1	595	State	2007	2007-04-15	8	ela	789	49
2	595	State	2007	2007-04-15	8	ela	799	50
3	595	State	2007	2007-04-15	8	math	770	45

```
statetest %<>% group_by(sid, test_subject, school_year, grade_level) %>%
  mutate(keep_flag = scaled_score == max(scaled_score) &
         test_date == max(test_date)) %>%
  ungroup %>%
  filter(keep_flag) %>%
  select(-keep_flag)
```

```
statetest %>% filter(sid == 595) %>%
  select(sid, test_type, school_year, test_date, grade_level,
         test_subject, scaled_score, raw_score)
```

```
# A tibble: 2 × 8
```

	sid	test_type	school_year	test_date	grade_level	test_subject	scaled_score	raw_score
	<dbl>	<chr>	<dbl>	<date>	<dbl>	<chr>	<dbl>	<dbl>
1	595	State	2007	2007-04-15	8	ela	799	50
2	595	State	2007	2007-04-15	8	math	770	45

```

# // Verify that each student has only one state test in a
# subject in a school year.

statetest %>% distinct(sid, test_subject, grade_level, school_year) %>%
  nrow == nrow(statetest)

[1] TRUE

# // 4. Reshape the data so math and ELA tests appear on the same row.

statetest <- reshape(as.data.frame(statetest),
  v.names = c("raw_score", "scaled_score"),
  timevar = c("test_subject"),
  idvar = c("sid", "test_type", "test_date",
    "school_year", "grade_level"),
  direction = "wide",
  sep = "_")

# // 5. Compute standardized test scores with mean 0 and standard deviation 1.

statetest$scaled_math_std <- scale(statetest$scaled_score_math)
statetest$scaled_ela_std <- scale(statetest$scaled_score_ela)

# // 6. Identify different-year repeat test takers and take the earliest test score.
#
# // First process ELA scores
# preserve
# drop *_math*
# drop if scaled_score_ela==.
# // Keep only the earliest instance in which the student
# took the test
# sort sid grade_level school_year
# drop if sid==sid[_n-1] & grade_level==grade_
# level[_n-1] & school_year>= school_year[_n-1] & scaled_
# score_ela[_n-1]!=.
# bys sid: gen count=_n
# tab count
# drop count
# // Save the ela_scores as a tempfile to be merged on
# tempfile ela_scores
# save `ela_scores'
# restore
#
# // Next process math scores
# drop *_ela*
# drop if scaled_score_math==.
# // Keep only the earliest instance in which the student took the test
# sort sid grade_level school_year
# drop if sid==sid[_n-1] & grade_level==grade_level[_n-1] &
# school_year>= school_year[_n-1] & scaled_score_math[_n-1]!=.
# bys sid: gen count=_n
# tab count
# drop count

```



```

# // Merge the ela_scores tempfile onto the math scores
# merge 1:1 sid using `ela_scores`, nogen

statetest %<>% group_by(sid) %>%
  mutate(keep_flag = test_date == min(test_date)) %>%
  filter(keep_flag) %>% select(-keep_flag)

# Not sure in R we need to process these separately at all!

# // 7. Verify that each student has only one state test, and drop unneeded variables.
nrow(statetest) == nvals(statetest$sid)

[1] TRUE
statetest %<>% select(-test_date, -test_type)

# // 8. Generate composite scaled and standardized scores that average ELA and math scores.
statetest$scaled_score_composite <- (statetest$scaled_score_ela + statetest$scaled_score_math) /2
statetest$scaled_score_composite_std <- (statetest$scaled_math_std + statetest$scaled_ela_std) /2

# // 9. Save the current file as Prior_Achievement.dta.
statetest %<>% arrange(sid, school_year, grade_level) %>%
  select(sid, school_year, grade_level, raw_score_math, raw_score_ela,
         scaled_score_math, scaled_score_ela, scaled_score_composite,
         scaled_math_std, scaled_ela_std, scaled_score_composite_std)

# save "${clean}/Prior_Achievement.dta", replace

** Part II: Clean SAT Scores **

# // 1. Keep only the variables and limit the sample to SAT.
satatest <- statest %>% filter(test_type == "SAT")

satatest %<>% select(sid, test_subject, test_date, scaled_score)
# // 2. Drop duplicate observations and any observations missing test scores.

satatest %<>% distinct()
satatest %<>% filter(!is.na(scaled_score))

# // 3. Reshape the data so that math, ELA, and writing scores appear on one row by student and test da

satatest <- reshape(as.data.frame(satatest),
  v.names = c("scaled_score"),
  timevar = c("test_subject"),
  idvar = c("sid", "test_date"),
  direction = "wide",
  sep = "_")

# Rename for convenience
names(satatest) <- c("sid", "sat_test_date", "sat_math_score",
  "sat_verbal_score", "sat_writing_score")

```

```

sattest %<>% arrange(sid, sat_test_date)
# // 4. Identify repeat test takers and take the earliest test score.

sattest %<>% group_by(sid) %>%
  mutate(keep_flag = sat_test_date == min(sat_test_date)) %>%
  filter(keep_flag) %>% select(-keep_flag)

# // Verify that the file is now unique by student.

nrow(sattest) == nvals(sattest$sid)

[1] TRUE

# // 5. Verify that test scores from the component subjects are not missing and generate total scores.

table(!is.na(sattest$sat_math_score) & !is.na(sattest$sat_verbal_score))

TRUE
271

sattest$sat_total_score <- sattest$sat_math_score + sattest$sat_verbal_score

table(!is.na(sattest$sat_math_score) & !is.na(sattest$sat_verbal_score) &
      !is.na(sattest$sat_writing_score))

TRUE
271

sattest$sat_total_score_plus_writing <- sattest$sat_math_score +
  sattest$sat_verbal_score + sattest$sat_writing_score

# // 6. Save the current file as SAT.dta.

** Part III: Clean ACT Scores **

# // 1. Keep only the variables you need and limit the sample to ACT.

acttest <- stutest %>% filter(test_type == "ACT")
acttest %<>% select(sid, test_subject, test_date, scaled_score)

# // 2. Identify repeat test takers and take the earliest test score.

acttest %<>% group_by(sid) %>%
  mutate(keep_flag = test_date == min(test_date)) %>%
  filter(keep_flag) %>% select(-keep_flag)

# // 3. Keep and rename the relevant variables.
acttest %>% select(sid, test_date, scaled_score)

Source: local data frame [2,544 x 3]
Groups: sid [2,544]

  sid test_date scaled_score
<dbl>    <date>    <dbl>

```

```

1      10 2008-04-06      14
2      16 2008-02-07      17
3      30 2008-04-06      17
4      38 2008-02-07      19
5      40 2008-04-06      29
6      67 2008-04-06      16
7      73 2008-02-07      13
8      74 2007-10-07      28
9      77 2008-04-06      20
10     80 2008-04-06      18
# ... with 2,534 more rows

names(acttest) <- c("sid", "act_test_date", "act_composite_score")

# // Verify that the file is now unique by student.
nrow(acttest) == nvals(acttest$sid)

[1] TRUE

# // 4. Save the current file as ACT.dta.
rm(stutest, acttest, satest, statetest)

```

## Task 6: STUDENT CLASS ENROLLMENT

### PURPOSE

In Task 6: Student Class Enrollment, you will take the Class\_Raw file and the Student\_Class\_Enrollment file to create the Student\_Class\_Enrollment\_Merged file that combines these two files together. The combined file will identify a unique observation by student and class id. To obtain this file, you will first clean the Class\_Raw file to identify core courses in math and ELA based on the course description variable and match the specification in Identify. This will make the class file unique by class id. Second, you will merge the Class file and the Student Class Enrollment file and make it unique by student id and class id. The core of this task: 1. Using the Class file: a. Drop incomplete observations b. Flag core math and English courses based on the course description 2. Merging the Student Class Enrollment file: a. Merge the Class file onto the Student\_Class\_Enrollment\_Raw file b. Evaluate course marks and drop courses with no record of completion c. Evaluate course enrollment so that each student has only one enrollment record for a course The Student\_Class\_Enrollment\_Merged file will be used in Connect to create on-track indicators for students. On-track indicators explore year-by-year academic progress towards high school graduation and college readiness. For instance, using course credit and course grade information, one might ask what percent of students earn the minimum number of credits in their core courses to satisfy agency graduation requirements?

### HOW TO START

To begin, open the Class\_Raw file in Stata. This file contains data linking students to teachers. If you do not have Stata, you can follow the steps of the task by looking at the instructions and data snippets we have provided. In the second part of this task, you will then use the Student\_Class\_Enrollment file. If this is your first time attempting Task 6, start with the provided input file. This file teaches you SDP's cleaning methodology and allows you to check answers from a common dataset.

## DATA DESCRIPTION FOR RAW FILE

The input file, `Class_Raw`, varies from `Class` in `Identify` in a number of key ways. Most importantly, the data is not unique by `cid` as shown in `Identify`. For instance, there may be more than one course description that describes the same course. Also, a `tid` is not included as it is not required for the questions later asked in this toolkit. Support for a `Class` file with `tid` will come with the Human Capital version of the toolkit. The aim of this task then is to eliminate any duplicate course code descriptions and match the `Class` file in `Identify` in its structure and uniqueness so it is unique by `cid` alone.

```
# Read in Stata
library(haven) # required for .dta files

# To read data from a zip file we create a connection to the path of the
# zip file
tmpfileName <- "raw/Class_Raw.dta"
con <- unz(description = "data/raw.zip", filename = tmpfileName,
            open = "rb")
classRaw <- read_stata(con) # read data in the data subdirectory
glimpse(classRaw)
```

Observations: 135,969

Variables: 8

```
$ cid          <dbl> 541631401, 432349312, 802451252, 831688206, 34366305, 208062771, 31066135, 4
$ credits_possible <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 3.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.0, 1.0, 1.0, 0
$ school_year    <dbl> 2007, 2005, 2007, 2009, 2007, 2008, 2008, 2008, 2008, 2008, 2008, 2007, 200
$ school_code    <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, N
$ section_code   <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, N
$ instructional_level <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, N
$ course_code_desc <chr> "ELECTIVE II", "ELECTIVE II", "ELECTIVE II", "ELECTIVE II", "ELECTIVE II", "
$ course_code    <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, N
```

```
# TODO - File is named incorrectly on p.59
tmpfileName <- "raw/Student_Class_Enrollment.dta"
con <- unz(description = "data/raw.zip", filename = tmpfileName,
            open = "rb")
stuclass <- read_stata(con) # read data in the data subdirectory
glimpse(stuclass)
```

Observations: 1,010,819

Variables: 8

```
$ sid          <dbl> 13281, 18950, 18950, 17817, 4739, 4739, 6737, 4897, 19940, 1809, 2675, 27
$ cid          <dbl> 227008230, 488826242, 441147758, 64721603, 592975099, 986162419, 47962993
$ class_enrollment_date <date> 2006-08-13, 2007-08-13, 2007-08-13, 2005-08-12, 2005-08-12, 2005-08-12, 2
$ class_withdrawal_date <date> 2007-07-01, 2008-06-30, 2008-06-30, 2006-07-01, 2006-07-01, 2006-07-01, 2
$ marking_period <chr> "S2", "S1", "S2", "S2", "Q4", "Q3", "S2", "S2", "S1", "S2", "S1", "S1", "S
$ final_grade_mark <chr> "B-", "A", "A", "D", "C", "D", "A-", "C+", "F", "B+", "F", "B", "F", "A-"
$ final_grade_mark_num <dbl> 2.7, 4.0, 4.0, 1.0, 2.0, 1.0, 3.7, 2.3, 0.0, 3.3, 0.0, 3.0, 0.0, 3.7, 1.9
$ credits_earned <dbl> 0.5, 0.5, 0.5, 1.0, 0.5, 0.5, 0.5, 0.5, 0.0, 0.5, 0.0, 0.5, 0.0, 0.5, 0.5
```

\*\* Part I: Clean the Class file \*\*

```
# // 1. Identify the critical variables that identify a class.
```

```
local_ids <- c("cid", "school_year", "school_code", "section_code",
              "course_code")
```

```
# // 2. Drop the observations where any of the critical variables are missing
classRaw %<>%
  filter(complete.cases(.[, local_ids]))
```

**\*\* Step 1: Flag core math and English courses. \*\***

```
# // Note that agencies may have varying consistency in course
# names and use different criteria to identify a core course
# vs an elective.
# // In some cases, other criteria may have to be applied to
# identify core courses (e.g. the department the course is
# listed in, or length of the course.)
# // We provide a simplified version of the cleaning process
# for the class file: work within your agency to determine the
# best criteria.

# // 1. Tabulate course names
table(classRaw$course_code_desc)
```

ALGEBRA 6202	ALGEBRA II 21	CALCULUS 218	ELECTIVE I 27468	ELECTIVE II 23679	ELECTIVE III 23702	ELECTIVE IV 32184	ENG 10 101
ENGLISH 10 177	ENGLISH 11 418	ENGLISH 12 287	ENGLISH 9 7913	GEOM 2444	GEOMETRY 2409	OTHER ELA 151	OTHER MATH 310
TRIGONOMETRY 44							

```
# // 2. Flag math courses based on the tabulation results
# // Generate a flag variable

classRaw$math_flag <- NA

# // Use the regexm function to identify course names that contain common word stems, but slightly diff
# Algebra I and Algebra-I

# In R the patterns need to not have spaces in the grep command
# The spaces will be matched
# | = OR
# grep does partial matching
# grpl returns TRUE/FALSE, as.numeric converts this to 1/0
classRaw$math_flag <- as.numeric(grepl("GEOM|ALGEBRA|MATH|STAT|CALC|TRIG",
                                       classRaw$course_code_desc))
# // Check the results of flagging your variables

table(classRaw$course_code_desc, classRaw$math_flag)
```

	0	1
ALGEBRA	0	6202
ALGEBRA II	0	21
CALCULUS	0	218
ELECTIVE I	27468	0
ELECTIVE II	23679	0
ELECTIVE III	23702	0
ELECTIVE IV	32184	0

ENG 10	101	0
ENGLISH 09	7799	0
ENGLISH 10	177	0
ENGLISH 11	418	0
ENGLISH 12	287	0
ENGLISH 9	7913	0
GEOM	0	2444
GEOMETRY	0	2409
OTHER ELA	151	0
OTHER MATH	0	310
STATISTICS	0	68
TRIGONOMETRY	0	44

*# // 3. Repeat this process for flagging ELA courses*

```
classRaw$ela_flag <- NA
```

*# In R the patterns need to not have spaces in the grep command*

*# The spaces will be matched*

*# | = OR*

*# grep does partial matching*

*# grpl returns TRUE/FALSE, as.numeric converts this to 1/0*

```
classRaw$ela_flag <- as.numeric(grepl("ENG|ELA",
                                     classRaw$course_code_desc))
```

*# // Check the results of flagging your variables*

```
table(classRaw$course_code_desc, classRaw$ela_flag)
```

	0	1
ALGEBRA	6202	0
ALGEBRA II	21	0
CALCULUS	218	0
ELECTIVE I	27468	0
ELECTIVE II	23679	0
ELECTIVE III	23702	0
ELECTIVE IV	32184	0
ENG 10	0	101
ENGLISH 09	0	7799
ENGLISH 10	0	177
ENGLISH 11	0	418
ENGLISH 12	0	287
ENGLISH 9	0	7913
GEOM	2444	0
GEOMETRY	2409	0
OTHER ELA	0	151
OTHER MATH	310	0
STATISTICS	68	0
TRIGONOMETRY	44	0

\*\*\* Step 3: Drop any unneeded variables, drop duplicates, and save the temporary file \*\*

*# // 1. Drop the course\_code\_desc, as it is no longer needed.*

```
classRaw %<>% select(-course_code_desc)
```

```
classRaw %<>% distinct()
```

```
# // 2. Verify that the data is unique by cid, and also unique by school year, school code, section code
nrow(classRaw) == nvals(classRaw$cid)
```

```
[1] TRUE
```

```
classRaw %>% distinct(school_year, school_code,
                      section_code, course_code) %>%
  nrow == nrow(classRaw)
```

```
[1] TRUE
```

```
** Part II: Clean the Student_Class_Enrollment file. **
```

```
** Step 0: Load the Student_Class_Enrollment data file. **
```

```
** Step 1: Merge on the temporary Class file you saved earlier to the Student_Class_Enrollment file **
```

```
# Merging is fun
# keep only files merged from both files
stuclass <- inner_join(stuclass, classRaw, by = "cid")
```

```
# /** Step 2: Evaluate course marks. **/
```

```
table(stuclass$final_grade_mark, stuclass$credits_possible)
```

	0	0.1	0.125	0.13	0.17	0.2	0.25	0.3	0.33	0.333	0.34	0.35	0.4	0.5
A	3168	1	3	2	1	2	2176	2	4	18	0	0	2	18477
A-	397	0	0	0	0	0	654	0	1	25	0	0	0	8407
A+	591	0	1	0	0	0	884	3	0	1	0	0	0	5669
B	577	0	2	1	2	0	898	0	2	32	0	1	0	12779
B-	131	0	0	0	0	0	340	0	0	8	0	0	0	6500
B+	235	0	0	0	0	0	440	0	0	23	0	0	0	5334
C	311	0	1	1	3	0	466	0	3	12	1	0	1	9381
C-	66	0	0	0	0	0	160	0	0	2	0	0	0	4652
C+	69	0	0	1	0	0	182	0	0	5	0	0	0	4017
D	157	0	0	1	0	0	209	0	1	2	0	0	0	5565
D-	33	0	0	0	0	0	75	0	0	0	0	0	0	3291
D+	25	0	0	0	0	0	67	0	0	2	0	0	0	1973
DF	0	0	0	0	0	0	0	0	0	0	0	0	0	5
F	304	3	1	0	0	0	308	0	0	0	0	0	0	8992
NGPA	3773	0	0	0	0	0	107	0	0	1	0	0	0	967
P	4942	1	1	0	0	0	870	0	0	0	0	0	0	1097

	0.75	0.9	1	1.2	1.5	1.67	1.8	2	3	5	130
A	4	0	3347	4	125	1	1	1	0	0	1
A-	0	0	1153	0	27	0	1	1	0	0	0
A+	1	0	634	0	15	0	0	2	0	0	0
B	2	1	2329	2	39	1	0	1	0	0	0
B-	0	0	1033	0	9	0	0	1	0	0	0
B+	0	0	874	0	9	0	0	1	1	0	0
C	0	0	1672	0	37	0	1	2	0	0	0
C-	0	0	786	0	5	0	0	0	0	0	0

C+	0	0	669	0	5	0	0	0	0	1	0
D	1	0	925	1	7	0	0	0	1	4	0
D-	0	0	510	0	3	0	0	0	0	0	0
D+	0	0	356	0	0	0	0	0	0	1	0
DF	0	0	0	0	0	0	0	0	0	0	0
F	0	0	1284	0	24	0	0	0	1	2	0
NGPA	0	0	163	0	12	0	0	0	0	0	0
P	0	0	396	0	2	0	0	0	0	0	0

```
table(stuclass$final_grade_mark, stuclass$final_grade_mark_num)
```

	0	0.4	0.6	0.7	1	1.3	1.9	2	2.3	2.7	3	3.3	3.7	4
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A-	0	0	0	0	0	0	0	0	0	0	0	0	0	19364
A+	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	131688	0	0	0
B-	0	0	0	0	0	0	0	0	0	66526	0	0	0	0
B+	0	0	0	0	0	0	0	0	0	0	0	54934	0	0
C	0	0	0	0	0	0	0	96326	0	0	0	0	0	0
C-	0	0	0	0	0	0	47541	0	0	0	0	0	0	0
C+	0	0	0	0	0	0	0	0	41110	0	0	0	0	0
D	0	0	0	0	56969	0	0	0	0	0	0	0	0	0
D-	0	0	0	33538	0	0	0	0	0	0	0	0	0	0
D+	0	0	0	0	0	20183	0	0	0	0	0	0	0	0
DF	0	0	5	0	0	0	0	0	0	0	0	0	0	0
F	91855	0	0	0	0	0	0	0	0	0	0	0	0	0
NGPA	0	687	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
#
# // Some letter marks (NGPA and P) indicate that they do not count toward GPA, so you may leave the num

# ** Step 3: Evaluate course completion. **
# // Drop observations that have no record of course completion

stuclass %<>% filter(!is.na(final_grade_mark) &
                    !is.na(final_grade_mark_num) &
                    !is.na(credits_earned))

# ** Step 4: Evaluate course enrollment. **
# // Fix cases where a student has multiple observations for the same course with the same year and mar

# // 1. Remove enrollment and withdrawal dates that are not in the current school year.

library(lubridate) # handle dates and times in R correctly

stuclass$school_start <- mdy(paste0("08", "01", stuclass$school_year-1))
stuclass$school_end <- mdy(paste0("07", "31", stuclass$school_year))

stuclass$class_enrollment_date[stuclass$class_enrollment_date <
                                stuclass$school_start |
```



```

      stuclass$class_enrollment_date >
      stuclass$school_end] <- NA
stuclass$class_withdrawal_date[stuclass$class_withdrawal_date <
      stuclass$school_start |
      stuclass$class_withdrawal_date >
      stuclass$school_end] <- NA

stuclass %<>% select(-school_start, -school_end)

stuclass %>% filter(sid == 2251 & cid == 78150780) %>%
  select(sid, cid, school_code, school_year, class_enrollment_date,
         class_withdrawal_date)

# A tibble: 4 × 6
  sid      cid school_code school_year class_enrollment_date class_withdrawal_date
  <dbl>   <dbl>   <dbl>     <dbl>         <date>                <date>
1  2251 78150780     540       2006       2005-08-12          2005-08-17
2  2251 78150780     540       2006       2005-09-21          2005-11-02
3  2251 78150780     540       2006       2005-12-23              <NA>
4  2251 78150780     540       2006       2005-09-13          2005-08-27

# // 2. Identify the variables that identify a course

local_ids <- c("sid", "cid", "school_year", "marking_period")

# // 3. Populate all enrollments with the earliest enrollment date

# use group_by_ to pass character values to dplyr
stuclass %<>% ungroup %>%
  group_by(sid, cid, school_year, marking_period) %>%
  arrange(class_enrollment_date) %>%
  mutate(first_enroll = min(class_enrollment_date, na.rm=TRUE))

# There is a bug here in the enrollment date
# If you use group_by_ and mutate without the underscore
# in a pipe, and then calculate max or min of a date, you
# get the wrong time

stuclass$class_enrollment_date <- stuclass$first_enroll
stuclass %<>% select(-first_enroll)

stuclass %>% ungroup %>% filter(sid == 2251 & cid == 78150780) %>%
  select(sid, cid, school_code, school_year, class_enrollment_date,
         class_withdrawal_date)

# A tibble: 4 × 6
  sid      cid school_code school_year class_enrollment_date class_withdrawal_date
  <dbl>   <dbl>   <dbl>     <dbl>         <date>                <date>
1  2251 78150780     540       2006       2005-08-12          2005-08-17
2  2251 78150780     540       2006       2005-08-12          2005-08-27
3  2251 78150780     540       2006       2005-08-12          2005-11-02

```

```
4 2251 78150780          540          2006          2005-08-12          <NA>
```

```
# // 4. Populate all enrollments with the latest withdrawal date
```

```
stuclass %<>% ungroup %>%
  arrange(sid, cid, school_year, marking_period, class_withdrawal_date) %>%
  group_by(sid, cid, school_year, marking_period) %>%
  mutate(last_withdraw = max(class_withdrawal_date, na.rm=TRUE))

stuclass %>% ungroup %>% filter(sid == 2251 & cid == 78150780) %>%
  select(sid, cid, class_enrollment_date,
         class_withdrawal_date, last_withdraw)
```

```
# A tibble: 4 × 5
```

	sid	cid	class_enrollment_date	class_withdrawal_date	last_withdraw
	<dbl>	<dbl>	<date>	<date>	<date>
1	2251	78150780	2005-08-12	2005-08-17	2005-11-02
2	2251	78150780	2005-08-12	2005-08-27	2005-11-02
3	2251	78150780	2005-08-12	2005-11-02	2005-11-02
4	2251	78150780	2005-08-12	<NA>	2005-11-02

```
stuclass$class_withdrawal_date <- stuclass$last_withdraw
stuclass$last_withdraw <- NULL
```

```
stuclass %>% ungroup %>% filter(sid == 2251 & cid == 78150780) %>%
  select(sid, cid, school_code, school_year, marking_period,
         section_code, class_enrollment_date,
         class_withdrawal_date, class_withdrawal_date)
```

```
# A tibble: 4 × 8
```

	sid	cid	school_code	school_year	marking_period	section_code	class_enrollment_date	class_withdrawal_date
	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<dbl>	<date>	<date>
1	2251	78150780	540	2006	S1	7	2005-08-12	2005-11-02
2	2251	78150780	540	2006	S1	7	2005-08-12	2005-11-02
3	2251	78150780	540	2006	S1	7	2005-08-12	2005-11-02
4	2251	78150780	540	2006	S1	7	2005-08-12	2005-11-02

```
** Step 5: Drop any unneeded variables, drop duplicates, and save the file **
```

```
# // 1. Drop duplicate values
```

```
stuclass %<>% ungroup %>% distinct()
```

```
# // 2. Verify that the file is unique by sid and cid
```

```
nrow(stuclass) == nvals(paste0(stuclass$sid, stuclass$cid, sep = "_"))
```

```
[1] TRUE
```

```
# // 3. Order the variables
```

```
stuclass %<>% select(sid, cid, school_year, school_code, course_code,
                    marking_period, section_code, instructional_level,
                    credits_possible, math_flag, ela_flag,
                    class_enrollment_date, class_withdrawal_date,
                    final_grade_mark, final_grade_mark_num,
                    credits_earned)
```

```
# // 4. Sort the data

stuclass %<>% ungroup() %>%
  arrange(sid, school_year, marking_period, cid)

# // 5. Save the current file as Student_Class_Enrollment_Merged.dta.
```

## Task 7 STUDENT NSC ENROLLMENT

### PURPOSE

In Task 7: Student NSC Enrollment, you will take the Student\_NSC\_Enrollment file that matches the specification in Identify and produce a Student\_NSC\_Enrollment\_Indicators file that includes some of the first college enrollment indicators you will need for further analysis. College enrollment data is obtained from the National Student Clearinghouse (NSC). NSC matches students from a file your agency sends, including student id, student name, high school from where the student graduated, graduation date, and some other variables. For more information on the NSC matching process and requirements, visit [http://www.studentclearinghouse.org/high\\_schools/studenttracker](http://www.studentclearinghouse.org/high_schools/studenttracker) The core of this task: 1. Rename the variables typically returned by NSC 2. Format the date values 3. Standardize the variables that reflect the type of college the student enrolls in 4. Create a college graduation indicator 5. Interpret the college enrollment status 6. Identify the first college the student attended After this task, you will merge the Student\_NSC\_Indicators file onto the college-going analysis file from Connect. You will use this file and the high school graduation variables you will also create in Connect to then to generate further college-going variables, such as variables that indicating if a student enrolled in college the fall after graduation, enrolled in college a year after graduation, and persisted through subsequent years of college.

### HOW TO START

To begin, open the Student\_NSC\_Enrollment file in Stata. This file contains data on college enrollment and persistence for students in your agency. If you do not have Stata, you can follow the steps of the task by looking at the instructions and data snippets we have provided. If this is your first time attempting Task 7, start with the provided input file. This file teaches you SDP's cleaning methodology and allows you to check answers from a common dataset.

### DATA DESCRIPTION

The input file, Student\_NSC\_Enrollment, follows the structure of Student\_NSC\_Enrollment in Identify so it is unique by sid, college\_code\_branch, enrollment\_begin, and enrollment\_end. This usually equates to a semester. Though the exact structure of the data you receive from NSC may vary, it will likely look something like this. The aim of this task then is to become familiar with the NSC data and start building college enrollment outcomes that will be expanded upon in Connect.

**\*\* Step 0: Load the Student\_NSC\_Enrollment data file. \*\***

```
# Read in Stata
library(haven) # required for .dta files

# To read data from a zip file we create a connection to the path of the
# zip file
tmpfileName <- "raw/Student_NSC_Enrollment.dta"
con <- unz(description = "data/raw.zip", filename = tmpfileName,
  open = "rb")
```

```
stunsc <- read_stata(con) # read data in the data subdirectory
glimpse(stunsc)
```

Observations: 11,985

Variables: 15

```
$ sid          <dbl> 7, 10, 10, 10, 10, 10, 16, 20, 24, 24, 30, 33, 33, 33, 33, 33, 34, 36, 36,
$ record_found_yn <chr> "N", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y",
$ enrollment_begin <dbl> NaN, 20100109, 20090523, 20090110, 20090829, 20080830, 20090829, 20080819,
$ enrollment_end   <dbl> NaN, 20100503, 20090814, 20090505, 20091215, 20081216, 20091215, 20081012,
$ college_code_branch <chr> "", "746460-00", "746460-00", "746460-00", "746460-00", "746460-00", "86045
$ college_name      <chr> "", "COMMUNITY COLLEGE 400", "COMMUNITY COLLEGE 400", "COMMUNITY COLLEGE 400
$ college_state     <chr> "", "FL", "FL", "FL", "FL", "FL", "MA", "FL", "NY", "NY", "IL", "TX", "MA",
$ yr2_yr4          <chr> "", "2-year", "2-year", "2-year", "2-year", "2-year", "2-year", "2-year", "4-year",
$ public_private    <chr> "", "Public", "Public", "Public", "Public", "Public", "Public", "Public", "Private",
$ enrollment_status <chr> "", "L", "L", "H", "L", "F", "F", "W", "F", "F", "F", "L", "F", "F", "F", "F", "F",
$ graduated         <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N",
$ graduation_date   <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN,
$ college_sequence  <dbl> NaN, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, NaN, 1, 1, 1, 1, 1, 1,
$ degree_title      <chr> "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",
$ major            <chr> "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",
```

```
# // 1. Rename variables to indicate that they are NSC variables.
```

```
names(stunsc) <- c("sid", "n_record_found_yn", "n_enrollment_begin",
  "n_enrollment_end", "n_college_otheid",
  "n_college_name", "college_state", "yr2_yr4",
  "public_private", "n_enrollment_status",
  "graduated", "n_degree_date",
  "n_enrl_sequence", "degree_title", "major")
```

```
# // 2. Format the date values as dates.
```

```
library(lubridate)
```

```
stunsc <- as.data.frame(stunsc)
```

```
for(i in c("n_enrollment_begin", "n_enrollment_end",
  "n_degree_date")){
  stunsc[, i] <- lubridate::ymd(as.character(stunsc[, i]))
}
```

```
stunsc %>% filter(sid == 13047) %>%
```

```
  select(sid, n_record_found_yn, n_enrollment_begin,
    n_enrollment_end, n_college_name, yr2_yr4,
    public_private, n_enrollment_status, graduated)
```

	sid	n_record_found_yn	n_enrollment_begin	n_enrollment_end	n_college_name	yr2_yr4	public_private
1	13047	Y	2009-01-10	2009-05-05	B COMMUNITY COLLEGE	2-year	Public
2	13047	Y	2008-08-30	2008-12-17	UNIVERSITY OF B	4-year	Private
3	13047	Y	2009-08-29	2009-12-15	B COMMUNITY COLLEGE	2-year	Public
4	13047	Y	2008-08-30	2008-12-16	B COMMUNITY COLLEGE	2-year	Public

	n_enrollment_status	graduated
1	F	N
2	F	N
3	H	N
4	H	N

```

### Drop missing
# stunsc %<>% filter(stunsc$college_state != "")

# // 3. Standardize types of college by:
# // 2-year and 4-year college

stunsc$n_college_4yr <- ifelse(stunsc$yr2_yr4 == "4-year", 1, 0)
stunsc$n_college_2yr <- ifelse(stunsc$yr2_yr4 == "2-year" |
                              stunsc$yr2_yr4 == "Less Than 2 Years",
                              1, 0)

stunsc$yr2_yr4 <- NULL

# // Public and private college
table(stunsc$public_private)

      Private  Public
1088      2660   8237

stunsc$n_college_public <- ifelse(stunsc$public_private == "Public", 1, 0)
stunsc$n_college_private <- ifelse(stunsc$public_private == "Private", 1, 0)
stunsc$public_private <- NULL

# // In-state and out-of-state college
table(stunsc$college_state)

      CA  FL  IL  MA  NY  TX
1088 1915 1905 1683 1865 1779 1750

stunsc$n_college_instate <- ifelse(stunsc$college_state == "MA", 1, 0)
stunsc$n_college_outstate <- ifelse(stunsc$college_state != "MA", 1, 0)
stunsc$college_state <- NULL

# // 4. Create a college graduation indicator.
stunsc$n_degree <- ifelse(stunsc$graduated == "Y", 1, 0)
stunsc$graduated <- NULL
# // 5. Interpret enrollment status.

table(stunsc$n_enrollment_status)

      F  H  L  W
1118 8693 1350 551 273

stunsc$n_enrl_status <- factor(stunsc$n_enrollment_status,
                              levels = c("F", "H", "L", "W",
                                          "A", "D"))

stunsc$n_enrollment_status <- NULL

```

\*\* Step 2: Identify first college attended by type (any, 4-year and 2-year) that didn't result in a withdrawal.  
 \*\*

```
# // 1. Specify these types of college (any, 4-year, 2-year) in globals.
# global condition_any "n_college_4yr == 1 | n_college_2yr == 1"
# global condition_4yr "n_college_4yr == 1"
# global condition_2yr "n_college_2yr == 1"
# // 2. Calculate the days enrolled.
```

```
stunsc$days_enrolled <- stunsc$n_enrollment_end - stunsc$n_enrollment_begin
```

```
# // 3. Identify the first college a student enrolled in by type (any, 2-year, and 4-year).
```

```
stunsc %>% filter(sid == 13047) %>%
  select(sid, n_record_found_yn, n_enrollment_begin,
         n_enrollment_end, n_college_name,
         n_enrl_status, n_college_4yr, n_college_2yr)
```

	sid	n_record_found_yn	n_enrollment_begin	n_enrollment_end	n_college_name	n_enrl_status	n_college_4yr	n_college_2yr
1	13047	Y	2009-01-10	2009-05-05	B COMMUNITY COLLEGE	F	1	1
2	13047	Y	2008-08-30	2008-12-17	UNIVERSITY OF B	F	0	0
3	13047	Y	2009-08-29	2009-12-15	B COMMUNITY COLLEGE	H	1	1
4	13047	Y	2008-08-30	2008-12-16	B COMMUNITY COLLEGE	H	1	1

```
stunsc %<>% group_by(sid) %>%
  mutate(flag_status = ifelse(n_enrl_status %in% c("F", "H", "L"), 1, 0))

stunsc %>% filter(sid == 13047) %>%
  select(sid, n_record_found_yn, n_enrollment_begin,
         n_enrollment_end, n_college_name,
         n_enrl_status, n_college_4yr, n_college_2yr, flag_status)
```

Source: local data frame [4 x 9]

Groups: sid [1]

	sid	n_record_found_yn	n_enrollment_begin	n_enrollment_end	n_college_name	n_enrl_status	n_college_4yr	n_college_2yr
	<dbl>	<chr>	<date>	<date>	<chr>	<fctr>	<dbl>	<dbl>
1	13047	Y	2009-01-10	2009-05-05	B COMMUNITY COLLEGE	F	1	1
2	13047	Y	2008-08-30	2008-12-17	UNIVERSITY OF B	F	0	0
3	13047	Y	2009-08-29	2009-12-15	B COMMUNITY COLLEGE	H	1	1
4	13047	Y	2008-08-30	2008-12-16	B COMMUNITY COLLEGE	H	1	1

# ... with 2 more variables: n\_college\_2yr <dbl>, flag\_status <dbl>

```
stunsc %<>% group_by(sid, n_college_4yr) %>%
  mutate(first_enr_date_4yr = min(n_enrollment_begin[flag_status > 0])) %>%
  ungroup %>%
  group_by(sid, n_college_2yr) %>%
  mutate(first_enr_date_2yr = min(n_enrollment_begin[flag_status > 0])) %>%
  ungroup %>%
  group_by(sid) %>%
  mutate(first_enr_date_any = min(n_enrollment_begin[flag_status > 0]))
```

```
stunsc %>% filter(sid == 13047) %>%
```

```
  select(sid, n_enrollment_begin,
         n_enrollment_end,
         n_enrl_status, n_college_4yr, n_college_2yr, flag_status,
         first_enr_date_2yr, first_enr_date_4yr,
         first_enr_date_any) %>% as.data.frame
```

	sid	n_enrollment_begin	n_enrollment_end	n_enrl_status	n_college_4yr	n_college_2yr	flag_status	first_enr_date_2yr	first_enr_date_4yr	first_enr_date_any
1	13047	2009-01-10	2009-05-05	F	0	1	1	2008-08-30	2008-08-30	2008-08-30
2	13047	2008-08-30	2008-12-17	F	1	0	1	2008-08-30	2008-08-30	2008-08-30
3	13047	2009-08-29	2009-12-15	H	0	1	1	2008-08-30	2008-08-30	2008-08-30
4	13047	2008-08-30	2008-12-16	H	0	1	1	2008-08-30	2008-08-30	2008-08-30

```
stunsc %>% filter(sid == 13047) %>%
```

```
  select(sid, n_college_opeid,
         n_enrl_status, n_college_4yr, n_college_2yr, flag_status,
         first_enr_date_2yr, first_enr_date_4yr,
         first_enr_date_any) %>% as.data.frame
```

	sid	n_college_opeid	n_enrl_status	n_college_4yr	n_college_2yr	flag_status	first_enr_date_2yr	first_enr_date_4yr	first_enr_date_any
1	13047	164039-00	F	0	1	1	2008-08-30	2008-08-30	2008-08-30
2	13047	416739-00	F	1	0	1	2008-08-30	2008-08-30	2008-08-30
3	13047	164039-00	H	0	1	1	2008-08-30	2008-08-30	2008-08-30
4	13047	164039-00	H	0	1	1	2008-08-30	2008-08-30	2008-08-30

```
stunsc %<>% group_by(sid) %>%
```

```
  mutate(first_college_any_opeid = n_college_opeid[first_enr_date_any == n_enrollment_begin][1],
         first_college_4yr_opeid = n_college_opeid[first_enr_date_4yr == n_enrollment_begin & n_college_4yr == 1],
         first_college_2yr_opeid = n_college_opeid[first_enr_date_2yr == n_enrollment_begin & n_college_2yr == 1],
  ungroup
```

```
## // Get the college name and id for the first enrollment date
```

```
##
```

```
## stunsc %>% filter(sid == 13047) %>%
```

```
##   select(sid, n_college_opeid,  
##         n_enrl_status, n_college_4yr, n_college_2yr,  
##         first_enr_date_2yr, first_enr_date_4yr,  
##         first_enr_date_any,  
##         first_college_any_opeid,  
##         first_college_4yr_opeid, first_college_2yr_opeid) %>% as.data.frame  
##
```

```
## // Count how many first college names and ids you got for each student
```

```
## bys sid: egen nvals_first_college_`var`_`type` = nvals(temp_first_college_`var`_`type`)
```

```
##
```

```

# tmp <- stunsc %>% select(sid, first_college_any_opeid,
#                           first_college_4yr_opeid, first_college_2yr_opeid)
#
# tmp <- tmp[!duplicated(tmp), ]
# tmp$nunique <- apply(tmp[, 2:4], 1, function(x) length(unique(na.omit(x))))
#
# stunsc <- left_join(stunsc, tmp[, c(1,5)], "sid")
# rm(tmp); gc()
#
#
# stunsc %>% filter(nunique > 1) %>%
#   select(sid, n_enrl_status, n_college_4yr, n_college_2yr,
#           n_enrollment_begin,
#           n_college_opeid, first_college_any_opeid,
#           first_college_4yr_opeid, first_college_2yr_opeid,
#           days_enrolled) %>%
#   filter(sid == 13653) %>% as.data.frame
#

# // If a student started at multiple colleges of the same type on the same date, indicate this by repl

# // these values with a dummy value (">1") for processing.
# replace temp_first_college_`var`_`type' = ">1" if nvals_first_college_`var`_`type' > 1 & nvals_first_
# !=.

** Step 3: Drop any unneeded variables, and save the file **

# // 1. Drop the unneeded variables
# drop temp* nvals* days_enrolled
# // 2. Save the current file as Student_NSC_Enrollment_Indicators
# save "${clean}\\Student_NSC_Enrollment_Indicators.dta", replace

```