

CLEAN: DATA BUILDING GUIDE

Strategic Data Project

Center for Education Policy Research at Harvard University

Contents

CLEAN: DATA BUILDING GUIDE	3
Introduction	3
SDP DATA BUILDING GUIDE	3
The Tasks	3
How to Start	3
Task Structure	3
Summary	4
Task Map	4
Decision Rules Glossary	5
Task 1: STUDENT ATTRIBUTES	8
PURPOSE	8
HOW TO START	8
DATA DESCRIPTION	9
Uniqueness	9
Step 0: Load and Inspect the Data	9
Step 1: Create one consistent value for gender across years	11
Step 3: Create consistent values for high school diploma	21
Step 4: Clean up and save data file	26
Task 2: STUDENT SCHOOL YEAR	27
PURPOSE	27
HOW TO START	28
Step 0: Load the data file	28
Step 1: Create one consistent grade level	28
Step 2: Create one consistent FRPL value	29
Step 3: Create one consistent IEP value	30
Step 4: Create one consistent ELL value	31
Step 5: Create one consistent gifted value	31
Step 6: Drop any unneeded variables and save the file	31
Task 3: IDENTIFYING THE NINTH-GRADE COHORT	32
PURPOSE	32
HOW TO START	32
DATA DESCRIPTION	32
Uniqueness	32
Step 0: Load the Student_School_Year data file	32
Step 1: Flag the first school year a student enrolls each grade	33
Step 2: Identify when a student was first observed in 9th grade	34
Step 3: Impute the grade 9 school year for transfer students	35
Step 4: Adjust the imputation for some students	36
Step 5: Drop unneeded variables and save the file	38
Task 4: STUDENT SCHOOL ENROLLMENT	39
PURPOSE	39
HOW TO START	39
Step 0: Load the Student_School_Enrollment_Raw data file	39
Step 1: Create a school_start and school_end variable	40
Step 2: Remove abnormal enrollment observations	40

Step 3: Consolidate overlapping enrollments by student by school	41
Step 4: Update days_enrolled based on consolidated enrollments	42
Step 5: Determine the last withdrawal code for each student	42
Step 6: Drop unneeded variables and save	43
Task 5: STUDENT TEST SCORES	44
PURPOSE	44
HOW TO START	44
DATA DESCRIPTION	45
Uniqueness	45
Part I: Clean Prior Achievement Scores	45
Step 0: Load the Student_Test_Scores data file	45
Step 1: Keep only state test scores in 8th grade	46
Step 2: Clean up raw and scaled scores	46
Step 3: Identify same-year repeat test takers	46
Step 4: Reshape the data	47
Step 5: Compute standardized test scores	47
Step 6: Identify different-year repeat test takers	48
Step 7: Verify the data and drop unneeded variables	48
Step 8: Generate composite scaled and standardized scores	48
Step 9: Save the current file as Prior_Achievement.dta.	48
Part II: Clean SAT Scores	49
Part III: Clean ACT Scores	50
Task 6: STUDENT CLASS ENROLLMENT	51
PURPOSE	51
HOW TO START	51
DATA DESCRIPTION FOR RAW FILE	51
Uniqueness	52
Part I: Clean the Class File	52
Step 0: Load the Class_Raw data file	52
Step 1: Identify the critical variables that identify a class	52
Step 2: Flag core math and English courses	53
Step 4: Drop any unneeded variables and drop duplicates	54
Part II: Clean the Student_Class_Enrollment file	55
Step 0: Load the Student_Class_Enrollment data file	55
Step 1: Merge on the temporary Class file	55
Step 2: Evaluate course marks	55
Step 3: Evaluate course completion	57
Step 4: Evaluate course enrollment	57
Step 5: Drop any unneeded variables and save the file	59
Task 7 STUDENT NSC ENROLLMENT	59
PURPOSE	59
HOW TO START	60
DATA DESCRIPTION	60
Uniqueness	60
Clean	60
Step 0: Load the Student_NSC_Enrollment data file	60
Step 1: Rename variables and format them for analysis	61
Step 2: Identify first college attended by type	62
Step 3: Drop any unneeded variables, and save the file	65

CLEAN: DATA BUILDING GUIDE

Introduction

SDP DATA BUILDING GUIDE

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 College Going 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: Student Test Scores
- Task 6: Student Class Enrollment
- Task 7: NSC (National Student Clearinghouse) Data

Each task uses a raw input file and produces a cleaned output file that matches **Identify**.

Download these raw input files along with everything else you need for the toolkit as a zip file at sdp.cepr.harvard.edu/toolkit. When unzipped, this file will reveal an infrastructure including all the steps of the toolkit, the data files you need, and template files with R code.

In particular, in **Clean**, you will be working with the files in the **raw** folder. If you are using R, you can fill in the corresponding .R scripts in **programs** to go through the tasks.

How to Start

The beginning of the Data Building Guide is a Decision Rules Glossary (p. 6). This glossary provides decision rules for resolving data problems associated with particular variables. It is meant to be a quick-reference guide of rules that can be used with any software platform. These decision rules are then implemented in the step-by-step instructions the tasks provide.

SDP has also created a detailed **SDP R Glossary**, available as a separate document, that covers the R commands used throughout the toolkit. Commands are listed alphabetically and by subject.

As you go through a task, be sure to consult the data and code snippets to get a visual sense for the changes occurring at each step.

Task Structure

The tasks follow a logical sequence from **1** to **7**. Each task comes with its own raw input file that results in a cleaned output file that matches or extends the file **Identify**. We also provide all cleaned output files so you can check your answers after completing each task. If you have followed the task instructions correctly, you should arrive at the same cleaned output file.

In each task, you will also find:

- **Purpose:** — Clarifies the importance of the task.
- **How to Start:** — Identifies the input file(s) for the task.
- **Data Description:** — Describes data elements for the task.
- **Instructions:** — Provides instructions to transform data. These instructions include:
 - R code to help you execute the instructions through code
 - Data snapshots to help you visualize changes to the data at each step

Summary

Through the tasks, you will learn effective practices for: data transformation, variable construction, and implementation of key decision rules. The **Task Map** on the next page summarizes the inputs and outputs of each task and how the outputs are used in **Connect** to produce an analysis file. The Task Map also serves as a Table of Contents. If you need additional guidance, the friendly research team at SDP is available to help: ****sdp@gse.harvard.edu****.

Task Map

This map summarizes the inputs and outputs for each task and how the outputs are used in Connect to produce the college-going analysis and college-going analysis on-track file.

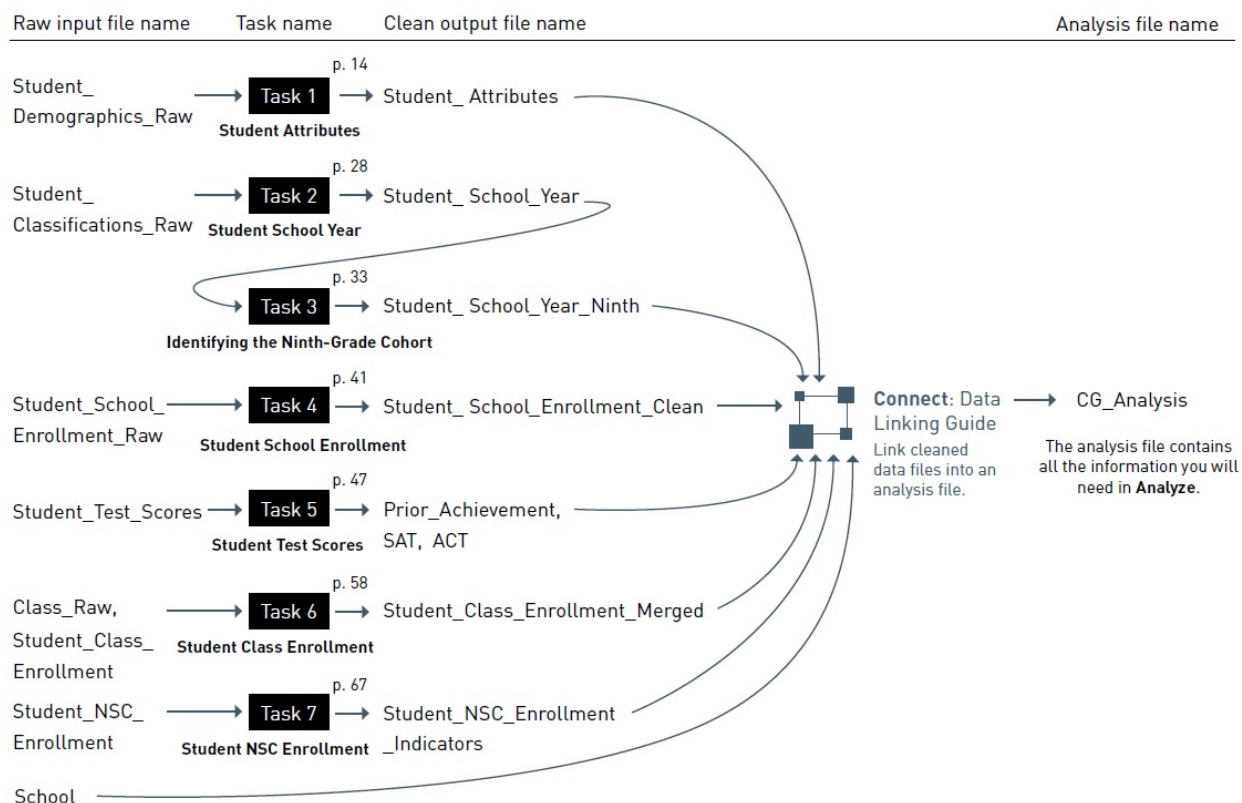


Figure 1: Taskmap

Decision Rules Glossary

Student_Attributes

Demographic, cohort, and graduation data for students.

Identifies unique observation: **sid**

Data Element	Possible Scenario	SDP Decision Rule	Reference in Data Building Tasks
sid			
male	Some students with conflicting genders may be observed.	If more than one gender observed, report the modal gender. If multiple modes, report the most recent gender recorded.	Task 1: Student Attributes
race_ethnicity	Some students with more than one race_ethnicity may be observed. For the purposes of analysis, SDP considers race_ethnicity time invariant.	If more than one category selected serially in time, report the modal race/ethnicity. If multiple modes are observed, report the most recent race/ethnicity recorded.	Task 1: Student Attributes
race	Some students with more than one race may be observed. For the purposes of analysis, SDP considers race time invariant.	If more than one category selected serially in time, report the modal race. If multiple modes are observed, report the most recent race recorded.	
ethnicity	Some students with more than one ethnicity may be observed. For the purposes of analysis, SDP considers ethnicity time invariant.	If more than one ethnicity observed, report the modal ethnicity. If multiple modes are observed, report the most recent ethnicity recorded.	
birth_date	Some students with more than one birth date may be observed. For the purposes of analysis, SDP considers birth_date time invariant.	If more than one birth date observed, report the modal birth date. If multiple modes, report the most recent birth date recorded. When evaluating modal birth date exclude birth dates that fall outside +/- four years of the expected birth date given grade level and school year.	
first_9th_school_year_reported			
hs_diploma			Task 1: Student Attributes
hs_diploma_date	Some students with more than one diploma date may be observed. For the purposes of analysis, SDP considers hs_diploma_date time invariant.	If more than one diploma date is observed, report the first diploma date.	Task 1: Student Attributes
hs_diploma_type	Some students with more than one diploma type may be observed. For the purposes of the analysis, SDP considers hs_diploma_type to be time invariant.	If more than one type is observed, report the type associated with the first diploma date. If multiple diploma types are observed for the first diploma date, report the modal value. If there is no mode, report the most competitive diploma type. If there is a diploma date but no diploma type, report the diploma type as "Unknown."	Task 1: Student Attributes
zip_code			

Student_School_Year

Yearly classification and attendance data for students.

Identifies unique observation: **sid + school_year**

Data Element	Possible Scenario	SDP Decision Rule	Reference in Data Building Tasks
sid			
school_year			
grade_level	Some students with more than one grade level may be observed during a given school year.	If more than one grade level observed during a school year, report the highest grade level recorded.	Task 2: Student School Year
frpl	Some students with more than one free or reduced price lunch status may be observed during a given school year.	If more than one status is observed during the school year, report the highest non-null value.	Task 2: Student School Year
iep	Some students with more than one individualized education plan status may be observed during a given school year.	If both "no IEP" and "has IEP" are observed during the same school year, report "has IEP".	Task 2: Student School Year
iep_classification	Some students with more than one individualized education plan classification may be observed during a given school year.	If more than one classification is observed during the school year, report the last non-null classification reported.	
ell	Some students with more than one English language learner status may be observed during a given school year.	If both "not ell" and "ell" are observed in during the same school year, report "ell".	Task 2: Student School Year
ell_classification	Some students with more than one individualized education plan classification may be observed during a given school year.	If more than one classification is observed during the school year, report the last non-null classification reported.	
gifted	Some students with more than one gifted status may be observed during a given school year.	If both "not enrolled" and "enrolled" are observed during the same school year, report "enrolled".	Task 2: Student School Year
gifted_classification	Some students with more than one gifted classification may be observed during a given school year.	If more than one classification is observed during the school year, report the last non-null classification reported.	
total_days_enrolled		If not reported, value may be calculated by number of school days between enrollment_date and withdrawal_date, or days_present + days_absent.	Task 2: Student School Year
total_days_absent		Cannot exceed the number of days enrolled.	Task 2: Student School Year
days_suspended_out_of_school		Cannot exceed the number of days enrolled.	Task 2: Student School Year
days_suspended_in_school		Cannot exceed the number of days enrolled.	Task 2: Student School Year

Student_School_Enrollment

School enrollment/withdrawal data for students.

Identifies unique observation: **sid + school_year + school_code + enrollment_date**

Data Element	Possible Scenario	SDP Decision Rule	Reference in Data Building Tasks
sid			
school_year			
school_code			
enrollment_date	Many different scenarios may be encountered: - Some enrollment spells may overlap at the same school. - Some enrollment spells may have a missing enrollment_date and withdrawal_date. - Some enrollment spells may have the same enrollment_date and withdrawal_date. - Some enrollment spells may begin after the withdrawal_date. - Some enrollment spells may begin after the end of the current school_year or before the beginning of the current school_year.	If enrollment spells overlap at the same school, consolidate enrollment observations into one enrollment period, using the earliest enrollment date and last withdrawal date and their corresponding codes and descriptions. Drop any other abnormal enrollment observations.	Task 4: Student School Enrollment
enrollment_code			
enrollment_code_desc			
withdrawal_date			
withdrawal_code			
withdrawal_code_desc			
days_enrolled		Update days_enrolled after you have consolidated overlapping enrollment observations.	Task 4: Student School Enrollment
days_present			

Student_Test_Scores

Standardized test data for students (state standardized tests, advanced placement, SAT, ACT, etc). Every attempt at a test by a student should be recorded.

Identifies unique observation: **sid + test_code + test_date**

Data Element	Possible Scenario	SDP Decision Rule	Reference in Data Building Tasks
sid			
test_code			
test_date	Students who re-take tests in the same year or are retained in a grade and re-take the same test in different years may have multiple observations for a single test code.	Take the earliest test in both cases.	Task 5: Student Test Scores
test_code_desc			
test_type			Task 5: Student Test Scores
grade_level			Task 5: Student Test Scores
test_subject			Task 5: Student Test Scores
test_version			
language_version			Task 5: Student Test Scores
raw_score	The raw score or scaled score may be missing or take on values outside the accepted range. Additionally, a test observation may have a missing raw test score but not a missing scaled, and vice versa.	Set scores outside the accepted range to missing. For instance, in the synthetic data, a test score of 0 is considered outside the accepted range and is set to missing. Based on knowledge of your agency, decide if all test observations should have both raw and scaled scores. One option would be to drop test observations missing either of the test scores. Here we only drop test observations that are missing both test scores.	Task 5: Student Test Scores
scaled_score			Task 5: Student Test Scores
performance_level			Task 5: Student Test Scores
standardized_score	The system may not provide a standardized score (i.e., mean zero, s.d. one).	Generate a standardized test score by subject, test_type, and school_year.	

Class

Class level scheduling data.

Identifies unique observation: **cid**

Data Element	Possible Scenario	SDP Decision Rule	Reference in Data Building Tasks
cid			Task 6: Student Class Enrollment
school_year			Task 6: Student Class Enrollment
school_code			Task 6: Student Class Enrollment
course_code			Task 6: Student Class Enrollment
course_code_desc	Agencies often have a very large number of course names. In some cases other criteria (e.g. department the course listed in, or length of course) is needed to identify a course.	Use agency rules to flag math and English courses.	Task 6: Student Class Enrollment
section_code			
period_bell			
room_number			
tid			
semester_term_year			Task 6: Student Class Enrollment
graduation_requirement			
credits_possible			Task 6: Student Class Enrollment
instructional_level			
subject			Task 6: Student Class Enrollment

Student_Class_Enrollment

Class enrollment, grades, and credits earned data for students.

Identifies unique observation: **sid + cid + enrollment_date**

Data Element	Possible Scenario	SDP Decision Rule	Reference in Data Building Tasks
sid			Task 6: Student Class Enrollment
cid			Task 6: Student Class Enrollment
class_enrollment_date	Some class enrollment spells may overlap: students may have more than one class_enrollment_date and/or more than one class_withdrawal_date for the same cid, school_year and marking_period.	For overlapping class enrollment spells, report the earliest enrollment date and latest withdrawal date	Task 6: Student Class Enrollment
class_withdrawal_date	Some class enrollment spells may overlap: students may have more than one class_enrollment_date and/or more than one class_withdrawal_date for the same cid, school_year and marking_period.	For overlapping class enrollment spells, report the earliest enrollment date and latest withdrawal date	Task 6: Student Class Enrollment
credits_earned			Task 6: Student Class Enrollment
final_grade_mark			Task 6: Student Class Enrollment

Student_NSC_Enrollment

National Student Clearinghouse Student Tracker student-level data that provides information on postsecondary outcomes. Identifies unique observation: `sid + college_code_branch + enrollment_begin + enrollment_end`

Data Element	Possible Scenario	SDP Decision Rule	Reference in Data Building Tasks
<code>sid</code>			Task 7: Student NSC Enrollment
<code>college_code_branch</code>			Task 7: Student NSC Enrollment
<code>enrollment_begin</code>		Identify the student's first college by the earliest enrollment date.	Task 7: Student NSC Enrollment
<code>enrollment_end</code>			Task 7: Student NSC Enrollment
<code>record_found_yn</code>			Task 7: Student NSC Enrollment
<code>high_school_code</code>			
<code>high_school_grad_dt</code>			
<code>college_name</code>			Task 7: Student NSC Enrollment
<code>college_state</code>		Create an indicator for in-state and one for out of state colleges	Task 7: Student NSC Enrollment
<code>year4year</code>		Create two indicators to mark 4-year colleges and 2-year colleges. Combine "2-year" and "Less Than 2 Years".	Task 7: Student NSC Enrollment
<code>public_private</code>		Create two indicators for the two types.	Task 7: Student NSC Enrollment
<code>enrollment_status</code>		Sometimes students enroll in more than one college at the same time. When identifying first college, report the one with the highest enrollment status (F, H, L, in order of importance) and then by longest duration.	Task 7: Student NSC Enrollment
<code>graduated</code>			Task 7: Student NSC Enrollment
<code>graduation_date</code>			Task 7: Student NSC Enrollment
<code>college_sequence</code>			Task 7: Student NSC Enrollment
<code>degree_title</code>			Task 7: Student NSC Enrollment
<code>major</code>			Task 7: Student NSC Enrollment

Task 1: STUDENT ATTRIBUTES

PURPOSE

In **Task 1: Student Attributes**, you will take the `Student_Demographics_Raw` file and generate the clean `Student_Attributes` file that matches the specification in **Identify** with one observation per student.

The core of this task:

1. Create consistent gender indicators for students across years.
2. Create consistent race/ethnicity values for students across years.
3. Create consistent values for high school diploma indicators.

HOW TO START

To begin, open the `Student_Demographics_Raw` file in R. If you do not have R, 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 1**, 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.

DATA DESCRIPTION

The clean `Student_Attributes` file includes `sid`, `male`, `race_ethnicity`, `first_9th_school_year_reported`, `hs_diploma`, `hs_diploma_date`, and `hs_diploma_type`. Later analyses do not currently make use of birth dates and zip codes, and these variables are thus excluded. This file contains the combined `race_ethnicity` variable rather than separate variables for race and ethnicity.

The raw input file, `Student_Demographics_Raw`, varies from the clean `Student_Attributes` file in a number of ways. In `Student_Demographics_Raw`, `race_ethnicity` is coded as a string rather than numeric and does not distinguish between the designations multiple, "M", and other, "O". `Student_Demographics_Raw` is also a time-variant data set including `school_year` so the data is unique by `sid` and `school_year`. `Student_Attributes`, however, is unique by `sid` alone. The aim of this task will be to match `Student_Attributes` to be unique by `sid` only.

Uniqueness

Some agencies may record **race_ethnicity** and/or **gender** each school year. Alternatively, students may have multiple records for having attended ninth grade or multiple diploma dates and/or types. To fix this issue, you will create a `Student_Attributes` research file unique by `sid` alone starting from a `Student_Demographics_Raw` file that is unique by `sid` and `school_year`. Once the file is unique by `sid` as shown in **Identify**, it is ready for **Connect**.

Step 0: Load and Inspect the Data

```
## Step 0: Load the packages and prepare your R environment
```

```
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 useful for toolkit tasks, see SDP R Glossary
# for details

source("R/functions.R")
library(haven) # required for importing .dta files
```

```
## Step 0: Load the college-going analysis file into Stata
## using the haven library
```

```
# 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
```

```
glimpse(stuatt)
```

```
Observations: 87,534
```

```
Variables: 9
```

```
$ sid           <dbl> 1, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 4, ...
$ school_year   <dbl> 2004, 2005, 2006, 2007, 2006, 2007, ...
$ male          <dbl> 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, ...
$ race_ethnicity <chr> "B", "H", "H", "H", "W", "B", "H", "...
$ birth_date    <dbl> 10869, 10869, 10869, 10869, 11948, 1...
$ first_9th_school_year_reported <dbl> 2004, 2004, 2004, 2004, NaN, NaN, 20...
$ hs_diploma     <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, ...
$ hs_diploma_type <chr> "", "", "", "", "Standard Diploma", ...
$ hs_diploma_date <date> NA, NA, NA, NA, 2008-06-05, 2009-05...
```

```
head(stuatt)
```

```
# A tibble: 6 × 9
```

```
  sid school_year male race_ethnicity birth_date
  <dbl>    <dbl> <dbl>         <chr>    <dbl>
1     1        2004     1           B      10869
2     1        2005     1           H      10869
3     1        2006     1           H      10869
4     1        2007     1           H      10869
5     2        2006     0           W      11948
6     2        2007     0           B      11948
```

```
# ... with 4 more variables: first_9th_school_year_reported <dbl>,
```

```
#   hs_diploma <dbl>, hs_diploma_type <chr>, hs_diploma_date <date>
```

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

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

```
n_distinct(stuatt$sid) == nrow(stuatt) #n_distinct function is in dplyr package
```

```
[1] FALSE
```

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,]
```

```
# A tibble: 9 × 8
```

```
  sid school_year male race_ethnicity birth_date hs_diploma
  <dbl>    <dbl> <dbl>         <chr>    <dbl>    <dbl>
1     2        2006     0           W      11948         1
2     2        2007     0           B      11948         1
3 8552        2005     1           W      12334         0
4 8552        2006     0           A      12334         0
5 8552        2006     1           W      12334         0
6 8552        2007     1           W      12334         0
7 8552        2009     1           W      12334         0
8 12506        2004     1           H      11803         0
9 12506        2005     0           H      11803         0
```

```
# ... with 2 more variables: hs_diploma_type <chr>, hs_diploma_date <date>
```

Step 1: Create one consistent value for gender 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)
```

```
# A tibble: 9 × 3
  sid school_year male
<dbl>      <dbl> <dbl>
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
```

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.

```
# Step 1: 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

```

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 taht 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
# 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
stuatt$male[!is.na(stuatt$male_mode)] <-
  stuatt$male_mode[!is.na(stuatt$male_mode)]

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)
```

Now check our work

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

```

  0      1
43660 43874
```

```
# Check nvals without creating the variable
```

```
stuatt %>% ungroup %>%
```

```
  group_by(sid) %>%
```

```
  summarize(nvals = n_distinct(male)) %>% select(nvals) %>%
```

```
  table
```

```

.
  1
21803
```

```
n_distinct(stuatt$sid)
```

```
[1] 21803
```

Step 2: Create one consistent value for race_ethnicity

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

```

```

# If the data were not coming from Stata, we would need to create a factor
# variable ourselves
# In R categorical variables are best represented as factors
# Factors can have values, order, 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
M/O	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
25321		7303	30444	1129	20528
MultipleOther					
2809					

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

Black	Asian	Hispanic	Native American	White
-------	-------	----------	-----------------	-------

```

28.927045      8.343044      34.779628      1.289785      23.451459
MultipleOther
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

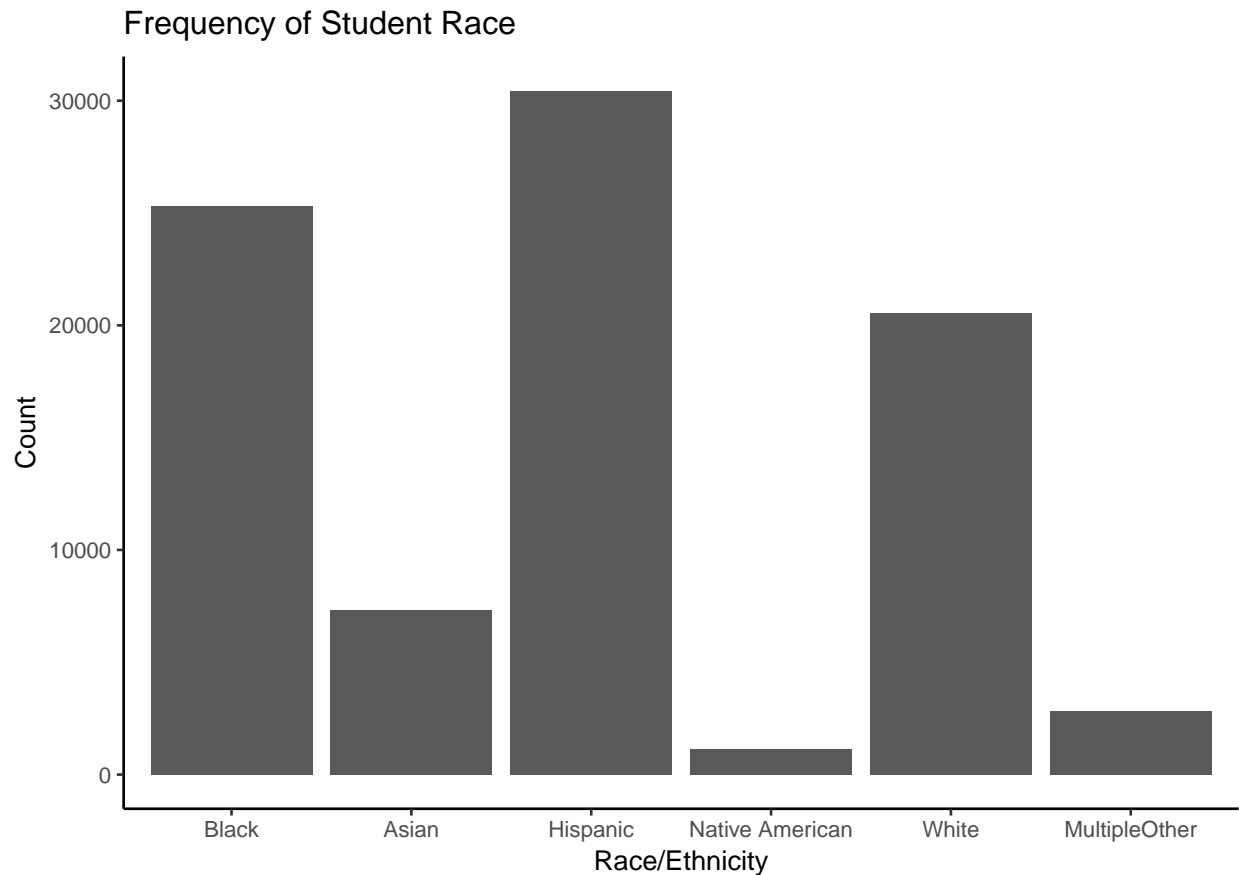
Let's also draw a figure to show this distribution.

```

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")

```



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 = n_distinct(race_ethnicity))

table(stuatt$nvals_race)
```

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

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 = n_distinct(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

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
<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
# ... with 1 more variables: ishispanic <dbl>

# 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

```

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>

Find the most recent race.

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)

```

Check your work.

```

table(stuatt$race_ethnicity)

```

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

Step 3: Create consistent values for high school diploma

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 in Stata,
 # 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

Identify the first diploma date reported

```
# 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
	<dbl>	<dbl>	<dbl>	<date>	<dbl>
1	16	2007	1	2008-05-14	2
2	16	2008	1	2008-05-14	1

... with 1 more variables: earliest_diploma_date <date>

Create a variable that shows the earliest diploma type

```
# Create a variable that shows the earliest diploma type
# This statement is extra long and includes the mode because it needs to avoid
# ties in the 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
  <dbl>      <dbl>      <dbl>      <date>      <dbl>
1     16         2007          1    2008-05-14          2
2     16         2008          1    2008-05-14          1
# ... with 2 more variables: earliest_diploma_date <date>,
#   earliest_dipl_type_mode <dbl>
```

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

```
# Number of unique diploma types 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
  <dbl>      <dbl>      <dbl>      <date>
1     16         2007          2    2008-05-14
2     16         2008          1    2008-05-14
3     20         2008          2    2008-05-14
4     20         2008          1    2008-05-14
5     80         2005          1    2008-05-14
6     80         2006          2    2008-05-14
7     80         2007          2    2008-05-14
8     80         2008          2    2008-05-14
# ... with 2 more variables: earliest_dipl_type_mode <dbl>,
#   nvals_dipl_type <int>
```

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

```
# 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
    <dbl>    <dbl>      <dbl>         <date>
1     16       2007          2      2008-05-14
2     16       2008          1      2008-05-14
3     20       2008          2      2008-05-14
4     20       2008          1      2008-05-14
5     80       2005          1      2008-05-14
6     80       2006          2      2008-05-14
7     80       2007          2      2008-05-14
8     80       2008          2      2008-05-14
# ... with 3 more variables: earliest_dipl_type_mode <dbl>,
#   nvals_dipl_type <int>, earliest_dipl_type_syear <dbl>

```

```

stuatt %<>% group_by(sid) %>%
  mutate(earliest_dipl_type_syear_mode = statamode(earliest_dipl_type_syear))

stuatt %>%
  select(sid, school_year, hs_diploma_type, earliest_diploma_date,
         earliest_dipl_type_mode, nvals_dipl_type,
         earliest_dipl_type_syear,
         earliest_dipl_type_syear_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
    <dbl>    <dbl>      <dbl>         <date>
1     16       2007          2      2008-05-14
2     16       2008          1      2008-05-14
3     20       2008          2      2008-05-14
4     20       2008          1      2008-05-14
5     80       2005          1      2008-05-14
6     80       2006          2      2008-05-14
7     80       2007          2      2008-05-14
8     80       2008          2      2008-05-14
# ... with 4 more variables: earliest_dipl_type_mode <dbl>,
#   nvals_dipl_type <int>, earliest_dipl_type_syear <dbl>,
#   earliest_dipl_type_syear_mode <dbl>

```

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

*# 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_syear))

stuatt %>%
  select(sid, school_year, hs_diploma_type, earliest_diploma_date,
         earliest_dipl_type_mode, nvals_dipl_type,
         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
1 16 2007 2 2008-05-14 NA
2 16 2008 1 2008-05-14 NA
3 20 2008 2 2008-05-14 NA
4 20 2008 1 2008-05-14 NA
5 80 2005 1 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
nvals_dipl_type earliest_dipl_type_syear earliest_dipl_type_syear_mode
1 2 2 2
2 2 2 2
3 2 2 NA
4 2 1 NA
5 2 1 1
6 2 1 1
7 2 1 1
8 2 1 1
temp_most_compet
1 2
2 2
3 1
4 1
5 1
6 1
7 1
8 1

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

stuatt$nvals_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
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	earliest_dipl_type_syear_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

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

```
# 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_dipoma_date)] <- 4
```

Finally, replace hs_diploma_date with the first hs_diploma_date

```
# Finally, replace hs_diploma_date with the first hs_diploma_date
```

```
stuatt$hs_diploma_date <- stuatt$earliest_diploma_date
```

```
# 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
```

```
# 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: Clean up and save data 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
# Create a clean directory
# dir.create("clean")
# save(stuatt, file = "clean/Student_Attributes.rda")
# Or if you want to save the Stata file
# write_dta(stuatt, file = "clean/Student_Attributes.dta")

# 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 R. If you do not have R, 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.

Step 0: Load the 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_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,...
$ school_year   <dbl> 2004, 2005, 2006, 2007, 2006, 2007, 20...
$ grade_level   <dbl> 9, 9, 10, 10, 11, 10, 11, 10, 8, 9, 11, 10...
$ frpl          <chr> "N", "N", "R", "R", "F", "F", "F", "F"...
$ iep           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ ell           <dbl> 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,...
$ total_days_enrolled <dbl> 210, 210, 210, 210, 172, 172, 228, 228...
$ total_days_absent  <dbl> 14, 6, 1, 5, 22, 57, 7, 15, 15, 7, 7, ...
$ days_suspended_out_of_school <dbl> 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 11, 0...
```

Step 1: Create one consistent grade level

Keep the highest `grade_level` when a student has multiple grade levels 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 = n_distinct(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

Recode raw frpl variable with string type to numeric type

```
# 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

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.

```
# 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 = n_distinct(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

```
# 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

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

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 over not enrolled.

```

```

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 and save the file

Drop duplicate observations

```

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

```

Make sure your file is now unique by student and school year

```

nrow(stuclass) == n_distinct(paste0(stuclass$sid, stuclass$school_year))

```

```

[1] TRUE

```

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

```

# dir.create("clean")
# save(stuclass, file = "Student_School_Year.rda")
# Or if you want to save the Stata file
# write_dta(stuclass, file = "clean/Student_Attributes.dta")

```

```
# Clean up the workspace
rm(con, tmpfileName, stuclass, varIdx)
```

Task 3: IDENTIFYING THE NINTH-GRADE COHORT

PURPOSE

In **Task 3: Identifying the Ninth Grade Cohort**, you will identify the school year students first appear in ninth grade using the clean Student_School_Year research file from **Task 2**. This essential step allows you to form student cohorts and examine longitudinal college-going outcomes.

The core of this task:

1. Flag the first school year a student enrolls in grades 9, 10, 11, or 12.
2. Identify the school year in which the student was first observed in 9th grade.
3. Impute the school year in which transfer students would have been in grade 9.
4. Replace the first_9th_school_year_observed with the correctly imputed values.

After completing this task, you will have a clean Student_School_Year file that identifies first-time ninth graders. This file is used both to assemble the analysis file in **Connect** and to complete **Task 4**.

HOW TO START

To begin, open the Student_School_Year file, just created in **Task 2**, in R. *Note if you are doing this in one sitting you can just keep it in your workspace.* If you do not have R, 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 3**, start with the cleaned output file from **Task 2**. This file teaches you SDP's cleaning methodology and allows you to check answers from a common dataset.

DATA DESCRIPTION

The input file in this case, Student_School_Year, also the output from **Task 2**, now follows the structure of Student_School_Year in **Identify** so it is unique by sid and school_year. The aim of this task will be to create a first_9th_school_year_observed variable using the variables in the file.

Uniqueness

This dataset was cleaned in **Task 2** and is now unique by sid and school_year.

Step 0: Load the Student_School_Year 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 <- "clean/Student_School_Year.dta"
con <- unz(description = "data/clean.zip", filename = tmpfileName,
           open = "rb")
stusy <- read_stata(con) # read data in the data subdirectory
glimpse(stusy)
```


Observations: 87,530

Variables: 10

```
$ sid          <dbl> 1, 1, 1, 1, 2, 2, 3, 3, 4, 4, 4, 5, 6,...
$ school_year  <dbl> 2004, 2005, 2006, 2007, 2006, 2007, 20...
$ grade_level  <dbl> 9, 9, 10, 11, 10, 11, 10, 9, 11, 10, 9...
$ frpl         <dbl+lbl> 0, 0, 1, 1, 2, 2, 2, 2, 0, 0, 0, 1...
$ iep         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,...
$ ell         <dbl> 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,...
$ total_days_enrolled <dbl> 210, 210, 210, 210, 172, 172, 228, 228...
$ total_days_absent  <dbl> 14, 6, 1, 5, 22, 57, 7, 15, 7, 7, 60, ...
$ days_suspended_out_of_school <dbl> 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 11, 0, 0...
```

Step 1: Flag the first school year a student enrolls each grade

Create four binary indicators to flag the first school year a student enrolls in grades 9, 10, 11, or 12.

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

A tibble: 4 × 3

	sid	school_year	grade_level
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
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 into multiple indicator variables

```
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, school_year) %>%
  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

# Create a temporary dataframe of only the variables we are interested in for
# tabulation
tmp <- stusy %>%
  select(num_range(prefix= "observed_", range = 9:12)) %>%
  distinct(.keep_all=TRUE)

table(tmp$observed_9)

```

```

      0      1
66643 20887

```

```
table(tmp$observed_10)
```

```

      0      1
71496 16034

```

```
table(tmp$observed_11)
```

```

      0      1
78046 9484

```

```
table(tmp$observed_12)
```

```

      0      1
81910 5620

```

```
rm(tmp) # remove our temporary data, note the original data will stay
```

Step 2: Identify when a 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]))

```

```
# If a student has no values of school_year where grade_level ==9 then
# R will assign this a value of infinite, which is slightly different
# than missing
```

```
# work around weird way R handles minimum of an empty vector
stusys$first_9th_schyear_obs[!is.finite(stusys$first_9th_schyear_obs)] <- NA
```

```
# Check data
stusys %>% 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	0	2004
4	1	2007	11	NA	0	2004

```
stusys %>% 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 grade 9 school year for transfer students

```
# Impute first_9th_school_year_observed as school_year - 1, school_year - 2, or
# school_year - 3 for students first observed in 10th, 11th or 12th grade
# as transfer-ins
```

```
stusys$first_flag10[!is.finite(stusys$first_flag10)] <- 0
stusys$first_flag11[!is.finite(stusys$first_flag11)] <- 0
stusys$first_flag12[!is.finite(stusys$first_flag12)] <- 0

stusys$tempfirst9year <- ifelse(stusys$first_flag10 == 1,
                               stusys$school_year - 1,
                               ifelse(stusys$first_flag11 == 1,
                                       stusys$school_year - 2,
                                       ifelse(stusys$first_flag12 == 1,
                                             stusys$school_year - 3,
                                             NA)))
```

```
stusys %>% 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

```
stusys %>% group_by(sid) %>%
  mutate(tempfirst9year = min(tempfirst9year, na.rm=TRUE))

stusys$first_9th_schyear_obs[is.na(stusys$first_9th_schyear_obs) &
  !is.na(stusys$tempfirst9year)] <-
  stusys$tempfirst9year[is.na(stusys$first_9th_schyear_obs) &
    !is.na(stusys$tempfirst9year)]

stusys$tempfirst9year <- NULL
```

Review the distribution of first_9th_school_year_observed for students who transferred in grades 10-12

*# Review the distribution of first_9th_school_year_observed for students who
transferred in grades 10-12*

```
stusys %>% 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 2009
   17 3764 3402 3722 4307    1
```

```
stusys %>% 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 2008
    3 3068 2965 3238   19
```

```
stusys %>% 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 2007
   18 2836 2643   29
```

Step 4: Adjust the imputation for some students

Flag students who are observed to be in a lower grade in a subsequent school year.

```
# 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

Flag the first school year in which students appear in high school grades

```
# 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
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	3	2006	10	2007	0	1
2	3	2007	9	2007	1	1

... with 1 more variables: first_9th_flag <dbl>

Replace the first_9th_school_year_observed with the correctly imputed values.

```
# Replace the first_9th_school_year_observed with the correctly imputed values.
stusy$temp4_first9year <- NA

for(g in c(10, 11, 12)){
  stusy$temp4_first9year[stusy$grade_flag_max == 1 &
                        stusy$first_9th_flag == 1 &
```

```

      stusy$grade_level == g] <-
    stusy$school_year[stusy$grade_flag_max == 1 &
      stusy$first_9th_flag == 1 &
      stusy$grade_level == g] - (g - 9)
  }

  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
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	3	2006	10	2007	0	1
2	3	2007	9	2007	1	1

... with 3 more variables: first_9th_flag <dbl>, temp4_first9year <dbl>,
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

```

Step 5: Drop unneeded variables 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)

```

Make directory and save

```
# dir.create("clean")
```

```
# save(stusy, file = "clean/Student_School_Year_Ninth.rda")
```

Or if you want to save the Stata file

```
# write_dta(stusy, file = "clean/Student_School_Year_Ninth.dta")
```

```
# Clean up the workspace
```

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

```
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 R. If you do not have R, 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.

Step 0: Load the Student_School_Enrollment_Raw data file

```
# Step 0: Load the Student_School_Enrollment_Raw 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_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...
$ school_year  <dbl> 2004, 2004, 2005, 2005, 2006, 2006, 2007, 2007...
$ school_code  <dbl> 486, 486, 485, 486, 485, 485, 485, 485, 486, 4...
$ enrollment_date <date> 2003-06-25, 2003-08-29, 2005-01-28, 2004-09-0...
$ enrollment_code_desc <chr> "Grade 9", "Grade 9", "Grade 9", "Grade 9", "G...
```

```
$ withdrawal_date      <date> 2003-07-05, 2004-06-08, 2005-06-07, 2005-02-1...
$ withdrawal_code_desc <chr> "Promoted End Year", "Retained in Grade", "Tra...
$ days_enrolled        <dbl> 10, 284, 130, 161, 0, 283, 284, NaN, 282, 191,...
```

Step 1: Create a school_start and school_end variable

```
## 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 be different in your agency.

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))

## - Caution - ##
## In R we have to create a character string that we convert to a date
## Converting numerics to dates and times can introduce errors
## - Caution - ##
```

Step 2: Remove abnormal enrollment observations

```
## Step 2: Remove abnormal enrollment observations.

# Drop observations missing both enrollment and withdrawal dates.
stuenr %<>% filter(!is.na(enrollment_date) & !is.na(withdrawal_date))

# Drop observations with enrollment and withdrawal dates on same day.
stuenr %<>% filter(!enrollment_date == withdrawal_date & !is.na(enrollment_date))

# Drop observations with withdrawal date earlier than enrollment date.

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

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

stuenr %<>% filter(school_end > enrollment_date)

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

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

# 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))

# 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

*# Sort enrollment spells in ascending order and then check how many overlapping
enrollment spells exist for a student at the same school.*

```
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))
```

```
FALSE TRUE
93090 682
```

```
# 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)

## For overlapping observations, replace the enrollment date and enrollment code
## description of all but the first enrollment spell with the earliest enrollment
## date

stuenr$enrollment_date[stuenr$enrollment_date <= stuenr$lag_withdrawal_date &
  !is.na(stuenr$lag_withdrawal_date)] <-
  stuenr$min_enroll_date[stuenr$enrollment_date <=
    stuenr$lag_withdrawal_date &
    !is.na(stuenr$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
	<dbl>	<dbl>	<dbl>	<date>	<chr>
1	2	2006	486	2005-08-27	Grade 10
2	2	2007	486	2006-08-29	Grade 11

```

3      2      2007      486      2006-08-29      Grade 11
# ... with 2 more variables: withdrawal_date <date>, withdrawal_code_desc <chr>

## 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
	<dbl>	<dbl>	<dbl>	<date>	<chr>
1	2	2006	486	2005-08-27	Grade 10
2	2	2007	486	2006-08-29	Grade 11
3	2	2007	486	2006-08-29	Grade 11

```

# ... with 2 more variables: withdrawal_date <date>, withdrawal_code_desc <chr>

# 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 consolidated enrollments

```

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
	<dbl>	<dbl>	<dbl>	<date>	<chr>
1	2	2006	486	2005-08-27	Grade 10
2	2	2007	486	2006-08-29	Grade 11
3	2	2007	486	2006-08-29	Grade 11

```

# ... with 3 more variables: withdrawal_date <date>,
#   withdrawal_code_desc <chr>, days_enrolled <time>

```

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
	<dbl>	<dbl>	<dbl>	<date>	<chr>
1	16	2007	450	2007-01-07	Grade 11
2	16	2008	450	2007-08-20	Grade 12

... with 2 more variables: withdrawal_date <date>, withdrawal_code_desc <chr>

```
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
	<dbl>	<dbl>	<dbl>	<date>	<chr>
1	16	2007	450	2007-01-07	Grade 11
2	16	2008	450	2007-08-20	Grade 12

... with 3 more variables: withdrawal_date <date>,
withdrawal_code_desc <chr>, last_withdrawal_reason <chr>

Step 6: Drop unneeded variables and save

```
## 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)

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

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

```
[1] TRUE
```

```
## Save the current file as Student_School_Enrollment_Clean
```

```
## Make directory and save
```

```
# dir.create("clean")
# save(stuenr, file = "clean/Student_School_Enrollment_Clean.rda")
## Or if you want to save the Stata file
# write_dta(stuenr, file = "clean/Student_School_Enrollment_Clean.dta")

rm(tmp, stuenr); gc()
```

	used (Mb)	gc trigger (Mb)	max used (Mb)
Ncells	649895 34.8	1770749 94.6	1770749 94.6
Vcells	1010915 7.8	7802288 59.6	9752837 74.5

Task 5: STUDENT TEST SCORES

PURPOSE

In **Task 5: Student Test Scores**, you will take the `Student_Test_Scores` file, containing data on all the tests a student has taken and matching the 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

To begin, open the `Student_Test_Scores` file in R. This file contains data on State assessments, SAT, and ACT scores. If you do not have R, 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 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 eighth 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.

Uniqueness

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

Step 0: Load the `Student_Test_Scores` 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_Test_Scores.dta"
con <- unz(description = "data/raw.zip", filename = tmpfileName,
           open = "rb")
stutest <- read_stata(con) # read data in the data subdirectory
close(con)
# Convert to R style
stutest$test_subject <- as_factor(stutest$test_subject)
stutest$test_subject <- tolower(as.character(stutest$test_subject))
glimpse(stutest)
```

Observations: 100,705

Variables: 10

```
$ sid          <dbl> 6, 6, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9,...
$ test_type    <chr> "State", "State", "State", "State", "State",...
$ school_year  <dbl> 2007, 2007, 2004, 2004, 2005, 2005, 2007, 20...
$ test_date    <date> 2007-04-15, 2007-04-15, 2004-04-15, 2004-04...
```

```

$ grade_level      <dbl> 8, 8, 8, 8, 9, 9, 10, 10, 8, 8, 8, 8, 7, 7, ...
$ test_subject     <chr> "math", "ela", "math", "ela", "math", "ela",...
$ scaled_score     <dbl> 726, 678, 722, 728, 851, 729, 609, 616, 698,...
$ performance_level <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2,...
$ performance_level_desc <chr> "On the Way to Proficient", "On the Way to P...
$ raw_score        <dbl> 35, 31, 40, 36, 59, 41, 18, 19, 35, 46, 40, ...

```

Step 1: Keep only state test scores in 8th grade

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

```

statetest <- stutest %>%
  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.*

Step 2: Clean up raw and scaled scores

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

```

# A forloop in R
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))

```

Step 3: Identify same-year repeat test takers

```
## 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
<dbl>   <chr>      <dbl>   <date>      <dbl>      <chr>      <dbl>
1   595   State      2007 2007-04-15         8        ela        789
2   595   State      2007 2007-04-15         8        ela        799
3   595   State      2007 2007-04-15         8        math        770
# ... with 1 more variables: raw_score <dbl>

```

```

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
<dbl> <chr>      <dbl>      <date>      <dbl>      <chr>      <dbl>
1  595      State      2007 2007-04-15         8      ela        799
2  595      State      2007 2007-04-15         8      math        770
# ... with 1 more variables: raw_score <dbl>

# 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

```

Step 4: Reshape the data

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

```

## 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 = "_")

```

Step 5: Compute standardized test scores

Compute standardized test scores. Standardized means the test score will have a mean 0 and standard deviation 1.

```

## 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)

statetest %>% select(scaled_math_std, scaled_ela_std) %>%
  na.omit %>% summary

scaled_math_std.V1  scaled_ela_std.V1
Min.      :-3.351985  Min.      :-4.141948
1st Qu.   :-0.674740  1st Qu.   :-0.602611
Median    :-0.005428  Median    : 0.095516
Mean      : 0.003436  Mean      : 0.000262

```

```
3rd Qu.: 0.679449    3rd Qu.: 0.679993
Max.    : 5.100017    Max.      : 5.761703
```

Step 6: Identify different-year repeat test takers

Find repeat test takers and take the earliest test score.

```
## Identify different-year repeat test takers and take the earliest test score.

## In R we can do this all at once using group_by
statetest %<>% group_by(sid) %>%
  mutate(keep_flag = test_date == min(test_date)) %>%
  filter(keep_flag) %>% select(-keep_flag)
```

Step 7: Verify the data and drop unneeded variables

Verify that each student has only one state test, and drop unneeded variables.

```
## Verify that each student has only one state test, and drop unneeded variables
nrow(statetest) == n_distinct(statetest$sid)

[1] TRUE

statetest %<>% select(-test_date, -test_type)
```

Step 8: Generate composite scaled and standardized scores

Generate composite scaled and standardized scores that average ELA and math scores.

```
## 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
```

Step 9: Save the current file as Prior_Achievement.dta.

```
## 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)

## Make directory and save
# dir.create("clean")
# save(statetest, file = "clean/Prior_Achievement.rda")
## Or if you want to save the Stata file
# write_dta(statetest, file = "clean/Prior_Achievement.dta")

rm(tmp, statetest); gc()
```

```
          used (Mb) gc trigger (Mb) max used (Mb)
Ncells 659117 35.3   1770749 94.6   1770749 94.6
Vcells 2033209 15.6   6241830 47.7   9752837 74.5
```


Part II: Clean SAT Scores

The steps here are condensed because the process is very similar to the process for state assessment scores.

```
## 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)

## Drop duplicate observations and any observations missing test scores.
satatest %<>% distinct()
satatest %<>% filter(!is.na(scaled_score))

## Reshape the data so that math, ELA, and writing scores appear on one row
## by student and test date.
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")
satatest %<>% arrange(sid, sat_test_date)

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

satatest %<>% 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(satatest) == n_distinct(satatest$sid)

[1] TRUE

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

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

TRUE
271

satatest$sat_total_score <- satatest$sat_math_score + satatest$sat_verbal_score

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

TRUE
271

satatest$sat_total_score_plus_writing <- satatest$sat_math_score +
  satatest$sat_verbal_score + satatest$sat_writing_score
```

```
## Save the current file as SAT.dta.

## Make directory and save
# dir.create("clean")
# save(sattest, file = "clean/SAT.rda")
## Or if you want to save the Stata file
# write_dta(sattest, file = "clean/SAT.dta")
rm(sattest); gc()

      used (Mb) gc trigger (Mb) max used (Mb)
Ncells 659895 35.3   1770749 94.6   1770749 94.6
Vcells 2035708 15.6   6241830 47.7   9752837 74.5
```

Part III: Clean ACT Scores

Steps are condensed because of their similarity to the steps above.

```
## 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)

## 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)

## 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) == n_distinct(acttest$sid)
```

```
[1] TRUE
```

```
## Save the current file as ACT.dta.
## Make directory and save
# dir.create("clean")
# save(acttest, file = "clean/ACT.rda")
## Or if you want to save the Stata file
# write_dta(acttest, file = "clean/ACT.dta")
rm(acttest); gc()

      used (Mb) gc trigger (Mb) max used (Mb)
Ncells 659579 35.3   1770749 94.6  1770749 94.6
Vcells 2034322 15.6   6241830 47.7  9752837 74.5
```

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 R. This file contains data linking students to teachers. If you do not have R, 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.

Uniqueness

The input file, `Student_Class_Enrollment_Raw`, follows the structure of `Student_Class_Enrollment` in Identify so it is unique by `sid`, `cid`, and `class_enrollment_date`. The aim of this task then is to take things one step further by consolidating any overlapping enrollment spells for the same student and `cid`.

Part I: Clean the Class File

Step 0: Load the Class_Raw 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/Class_Raw.dta"
con <- unz(description = "data/raw.zip", filename = tmpfileName,
            open = "rb")
classRaw <- read_stata(con) # read data in the data subdirectory
close(con)
glimpse(classRaw)
```

Observations: 135,969

Variables: 8

```
$ cid          <dbl> 541631401, 432349312, 802451252, 831688206, 343...
$ credits_possible <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 3.0, 0.5, 0.5, 0...
$ school_year    <dbl> 2007, 2005, 2007, 2009, 2007, 2008, 2008, 2008,...
$ school_code    <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, Na...
$ section_code   <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, Na...
$ instructional_level <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, Na...
$ course_code_desc <chr> "ELECTIVE II", "ELECTIVE II", "ELECTIVE II", "E...
$ course_code    <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, Na...
```

```
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
close(con)
glimpse(stuclass)
```

Observations: 1,010,819

Variables: 8

```
$ sid          <dbl> 13281, 18950, 18950, 17817, 4739, 4739, 6737,...
$ cid          <dbl> 227008230, 488826242, 441147758, 64721603, 59...
$ class_enrollment_date <date> 2006-08-13, 2007-08-13, 2007-08-13, 2005-08-...
$ class_withdrawal_date <date> 2007-07-01, 2008-06-30, 2008-06-30, 2006-07-...
$ marking_period <chr> "S2", "S1", "S2", "S2", "Q4", "Q3", "S2", "S2...
$ final_grade_mark <chr> "B-", "A", "A", "D", "C", "D", "A-", "C+", "F...
$ final_grade_mark_num <dbl> 2.7, 4.0, 4.0, 1.0, 2.0, 1.0, 3.7, 2.3, 0.0, ...
$ credits_earned <dbl> 0.5, 0.5, 0.5, 1.0, 0.5, 0.5, 0.5, 0.5, 0.0, ...
```

Step 1: Identify the critical variables that identify a class

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

local_ids <- c("cid", "school_year", "school_code", "section_code",
              "course_code")
## Drop the observations where any of the critical variables are missing
classRaw %<>%
  filter(complete.cases(., local_ids))
```

Step 2: 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.

```
## Tabulate course names
table(classRaw$course_code_desc)
```

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

```
## Flag math courses based on the tabulation results
## Generate a flag variable
classRaw$math_flag <- NA

## Use the grep function to identify course names that contain common word
## stems, but slightly different spellings, e.g. Algebra I and Algebra-I

## In R the patterns need to have no spaces in the grep command
## The spaces will be matched
## The | = OR
## grep does partial matching
## grepl 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

```
## Repeat this process for flagging ELA courses
classRaw$ela_flag <- NA
classRaw$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 4: Drop any unneeded variables and drop duplicates

```
## Drop the course_code_desc, as it is no longer needed.
classRaw %<>% select(-course_code_desc)

## Collapse the data
classRaw %<>% distinct()

## Verify that the data is unique by cid, and also unique by school year,
## school code, section code and course code.
```

```
nrow(classRaw) == n_distinct(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

This was done above simultaneously with the class file.

Step 1: Merge on the temporary Class file

Merge on the temporary Class file you saved earlier to the Student_Class_Enrollment file

In R you can merge two datasets using the familiar language of SQL and joins. `inner_join` tells R to retain only observations matched between both datasets.

```
## Merge classRaw and stuclass together
## keep only files merged from both files
stuclass <- inner_join(stuclass, classRaw, by = "cid")
```

Step 2: Evaluate course marks

```
## Evaluate course marks
table(stuclass$final_grade_mark, round(stuclass$credits_possible, digits = 1))
```

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.8	0.9	1
A	3168	6	2179	24	2	184776	1	4	0	3347
A-	397	0	654	26	0	84078	2	0	0	1153
A+	591	1	884	4	0	56697	0	1	0	634
B	577	3	900	34	1	127798	0	2	1	2329
B-	131	0	340	8	0	65004	0	0	0	1033
B+	235	0	440	23	0	53348	3	0	0	874
C	311	2	469	16	1	93813	2	0	0	1672
C-	66	0	160	2	0	46522	0	0	0	786
C+	69	1	182	5	0	40178	0	0	0	669
D	157	1	209	3	0	55659	1	1	0	925
D-	33	0	75	0	0	32917	0	0	0	510
D+	25	0	67	2	0	19732	0	0	0	356
DF	0	0	0	0	0	5	0	0	0	0
F	304	4	308	0	0	89928	0	0	0	1284
NGPA	3773	0	107	1	0	9670	0	0	0	163
P	4942	2	870	0	0	10976	0	0	0	396
	1.2	1.5	1.7	1.8	2	3	5	130		
	0	0	0	0	0	0	0	0		
A	4	125	1	1	1	0	0	1		

A-	0	27	0	1	1	0	0	0
A+	0	15	0	0	2	0	0	0
B	2	39	1	0	1	0	0	0
B-	0	9	0	0	1	0	0	0
B+	0	9	0	0	1	1	0	0
C	0	37	0	1	2	0	0	0
C-	0	5	0	0	0	0	0	0
C+	0	5	0	0	0	0	1	0
D	1	7	0	0	0	1	4	0
D-	0	3	0	0	0	0	0	0
D+	0	0	0	0	0	0	1	0
DF	0	0	0	0	0	0	0	0
F	0	24	0	0	0	1	2	0
NGPA	0	12	0	0	0	0	0	0
P	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
	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0	0
A-	0	0	0	0	0	0	0	0	0	0
A+	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0
B-	0	0	0	0	0	0	0	0	0	66526
B+	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	96326	0	0
C-	0	0	0	0	0	0	47541	0	0	0
C+	0	0	0	0	0	0	0	0	41110	0
D	0	0	0	0	56969	0	0	0	0	0
D-	0	0	0	33538	0	0	0	0	0	0
D+	0	0	0	0	0	20183	0	0	0	0
DF	0	0	5	0	0	0	0	0	0	0
F	91855	0	0	0	0	0	0	0	0	0
NGPA	0	687	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0

	3	3.3	3.7	4	4.3
	0	0	0	0	0
A	0	0	0	193640	0
A-	0	0	86339	0	0
A+	0	0	0	0	58829
B	131688	0	0	0	0
B-	0	0	0	0	0
B+	0	54934	0	0	0
C	0	0	0	0	0
C-	0	0	0	0	0
C+	0	0	0	0	0
D	0	0	0	0	0
D-	0	0	0	0	0
D+	0	0	0	0	0
DF	0	0	0	0	0
F	0	0	0	0	0
NGPA	0	0	0	0	0

P 0 0 0 0 0

Some letter marks (NGPA and P) indicate that they do not count toward GPA, so you may leave the numeric mark as missing.

Step 3: Evaluate course completion

```
## 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

```
## Evaluate course enrollment

## Fix cases where a student has multiple observations for the same course
## with the same year and marking period (i.e. with overlapping enrollment dates)

# 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)

# Check for correct changes
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
<dbl> <dbl> <dbl> <dbl> <date>
1 2251 78150780      540      2006 2005-08-12
2 2251 78150780      540      2006 2005-09-21
3 2251 78150780      540      2006 2005-12-23
4 2251 78150780      540      2006 2005-09-13
# ... with 1 more variables: class_withdrawal_date <date>

# Identify the variables that identify a course
local_ids <- c("sid", "cid", "school_year", "marking_period")

## Populate all enrollments with the earliest enrollment date
stuclass %<>% ungroup %>%
  group_by(sid, cid, school_year, marking_period) %>%
  arrange(class_enrollment_date) %>%
```

```

mutate(first_enroll = min(class_enrollment_date, na.rm=TRUE))

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
<dbl>   <dbl>   <dbl>     <dbl>         <date>
1  2251 78150780     540       2006       2005-08-12
2  2251 78150780     540       2006       2005-08-12
3  2251 78150780     540       2006       2005-08-12
4  2251 78150780     540       2006       2005-08-12
# ... with 1 more variables: class_withdrawal_date <date>

## 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
<dbl>   <dbl>   <dbl>     <dbl>         <chr>         <dbl>
1  2251 78150780     540       2006           S1             7
2  2251 78150780     540       2006           S1             7
3  2251 78150780     540       2006           S1             7
4  2251 78150780     540       2006           S1             7
# ... with 2 more variables: class_enrollment_date <date>,
#   class_withdrawal_date <date>

```

Step 5: Drop any unneeded variables and save the file

```
## Drop any unneeded variables, drop duplicates, and save the file
## Drop duplicate values
stuclass %<>% ungroup %>% distinct()

## Verify that the file is unique by sid and cid
nrow(stuclass) == n_distinct(paste0(stuclass$sid, stuclass$cid, sep = "_"))

[1] TRUE

## 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)

## Sort the data
stuclass %<>% ungroup() %>%
  arrange(sid, school_year, marking_period, cid)

## Save the current file as Student_Class_Enrollment_Merged.dta.
## Make directory and save

# dir.create("clean")
# save(stuclass, file = "clean/Student_Class_Enrollment_Merged.rda")
## Or if you want to save the Stata file
# write_dta(stuclass, file = "clean/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

To learn more about cleaning the NSC data and how to use NSC files, consult the NSC Missing Manual

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 R. This file contains data on college enrollment and persistence for students in your agency. If you do not have R, 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**.

Uniqueness

This dataset matches the specification in **Identify** and is unique by `sid`, `college_code_branch`, `enrollment_begin`, and `enrollment_end`.

Clean

Step 0: Load the Student_NSC_Enrollment data file

```
## 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
close(con)
glimpse(stunsc)
```

Observations: 11,985

Variables: 15

```
$ sid          <dbl> 7, 10, 10, 10, 10, 10, 16, 20, 24, 24, 30, 33, ...
$ record_found_yn <chr> "N", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y...
$ enrollment_begin <dbl> NaN, 20100109, 20090523, 20090110, 20090829, 20...
$ enrollment_end   <dbl> NaN, 20100503, 20090814, 20090505, 20091215, 20...
$ college_code_branch <chr> "", "746460-00", "746460-00", "746460-00", "746...
$ college_name     <chr> "", "COMMUNITY COLLEGE 400", "COMMUNITY COLLEGE...
$ college_state    <chr> "", "FL", "FL", "FL", "FL", "FL", "MA", "FL", "...
$ yr2_yr4         <chr> "", "2-year", "2-year", "2-year", "2-year", "2-...
```

```

$ public_private      <chr> "", "Public", "Public", "Public", "Public", "Pu...
$ enrollment_status  <chr> "", "L", "L", "H", "L", "F", "F", "W", "F", "F"...
$ graduated           <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N", "N"...
$ graduation_date     <dbl> NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, Na...
$ college_sequence    <dbl> NaN, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, ...
$ degree_title        <chr> "", "", "", "", "", "", "", "", "", "", "", "", ...
$ major              <chr> "", "", "", "", "", "", "", "", "", "", "", "", ...

```

Step 1: Rename variables and format them for analysis

```
## Rename variables and format them for analysis
```

```

# 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_opeid",
                  "n_college_name", "college_state", "yr2_yr4",
                  "public_private", "n_enrollment_status",
                  "graduated", "n_degree_date",
                  "n_enrl_sequence", "degree_title", "major")

```

```
# 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	n_enrollment_status	graduated
1	13047	Y	2009-01-10	2009-05-05	B COMMUNITY COLLEGE	2-year	Public	F	N
2	13047	Y	2008-08-30	2008-12-17	UNIVERSITY OF B	4-year	Private	F	N
3	13047	Y	2009-08-29	2009-12-15	B COMMUNITY COLLEGE	2-year	Public	H	N
4	13047	Y	2008-08-30	2008-12-16	B COMMUNITY COLLEGE	2-year	Public	H	N

```
## Drop missing
```

```
stunsc %<>% filter(stunsc$college_state != "")
```

```
## 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
    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
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
```

```
# Create a college graduation indicator.
stunsc$n_degree <- ifelse(stunsc$graduated == "Y", 1, 0)
stunsc$graduated <- NULL
```

```
# Interpret enrollment status.
table(stunsc$n_enrollment_status)
```

```
F  H  L  W
30 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
```

For enrollment status, the following labels apply:

- “F” = full time
- “H” = half time
- “L” = less than half time
- “W” = withdrew
- “A” = leave of absence
- “D” = deceased

Step 2: Identify first college attended by type

Identify first college attended by type (any, 4-year and 2-year) that didn’t result in a withdrawal.

```
## Identify first college attended by type (any, 4-year and 2-year)
## that didn’t result in a withdrawal.
```

```
## Calculate the days enrolled.
stunsc$days_enrolled <- stunsc$n_enrollment_end - stunsc$n_enrollment_begin
```

```

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

# Create a status flag variable that indicates minimum of partial enrollment
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
<dbl>      <chr>      <date>      <date>
1 13047                Y      2009-01-10      2009-05-05
2 13047                Y      2008-08-30      2008-12-17
3 13047                Y      2009-08-29      2009-12-15
4 13047                Y      2008-08-30      2008-12-16
# ... with 5 more variables: n_college_name <chr>, n_enrl_status <fctr>,
#   n_college_4yr <dbl>, n_college_2yr <dbl>, flag_status <dbl>

# In this block of code we are picking the first enrollment date for each of the
# types of schools (4yr, 2yr, any) that is enrolled at all (at least partial),
# and picking the first date that meets the status requirement

stunsc %>% group_by(sid) %>%
  arrange(sid, n_enrollment_begin, n_enrl_status, days_enrolled) %>%
  mutate(first_enr_date_4yr = n_enrollment_begin[flag_status > 0
                                                & n_college_4yr == 1][1]) %>%
  mutate(first_enr_date_2yr = n_enrollment_begin[flag_status > 0
                                                & n_college_2yr == 1][1]) %>%
  mutate(first_enr_date_any = n_enrollment_begin[flag_status > 0][1])

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
1 13047 2008-08-30 2008-12-17 F 1
2 13047 2008-08-30 2008-12-16 H 0
3 13047 2009-01-10 2009-05-05 F 0
4 13047 2009-08-29 2009-12-15 H 0
n_college_2yr flag_status first_enr_date_2yr first_enr_date_4yr
1 0 1 2008-08-30 2008-08-30
2 1 1 2008-08-30 2008-08-30
3 1 1 2008-08-30 2008-08-30
4 1 1 2008-08-30 2008-08-30
first_enr_date_any
1 2008-08-30
2 2008-08-30
3 2008-08-30
4 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
1 13047 416739-00 F 1 0 1
2 13047 164039-00 H 0 1 1
3 13047 164039-00 F 0 1 1
4 13047 164039-00 H 0 1 1
first_enr_date_2yr first_enr_date_4yr first_enr_date_any
1 2008-08-30 2008-08-30 2008-08-30
2 2008-08-30 2008-08-30 2008-08-30
3 2008-08-30 2008-08-30 2008-08-30
4 2008-08-30 2008-08-30 2008-08-30
```

```
# Here we are breaking ties by first picking the highest enrollment status,
# and in the case of a tie there, picking days enrolled
```

```
stunsc %<>% group_by(sid) %>%
  arrange(sid, desc(n_enrl_status), desc(days_enrolled)) %>%
  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 > 0][1],
         first_college_2yr_opeid = n_college_opeid[first_enr_date_2yr ==
                                                    n_enrollment_begin & n_college_2yr > 0][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
```

```
sid n_college_opeid n_enrl_status n_college_4yr n_college_2yr
1 13047 164039-00 H 0 1
2 13047 164039-00 H 0 1
3 13047 164039-00 F 0 1
4 13047 416739-00 F 1 0
first_enr_date_2yr first_enr_date_4yr first_enr_date_any
1 2008-08-30 2008-08-30 2008-08-30
2 2008-08-30 2008-08-30 2008-08-30
3 2008-08-30 2008-08-30 2008-08-30
4 2008-08-30 2008-08-30 2008-08-30
first_college_any_opeid first_college_4yr_opeid first_college_2yr_opeid
1 164039-00 416739-00 164039-00
2 164039-00 416739-00 164039-00
3 164039-00 416739-00 164039-00
4 164039-00 416739-00 164039-00
```

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

```
## Drop the unneeded variables
# drop temp* nvals* days_enrolled
stunsc$flag_status <- NULL

## Sort the data
stunsc %<>% ungroup() %>%
  arrange(sid, n_enrollment_begin)

## Save the current file as Student_NSC_Enrollment_Indicators.dta
## Make directory and save
# dir.create("clean")
# save(stunsc, file = "clean/Student_NSC_Enrollment_Indicators.rda")
## Or if you want to save the Stata file
# write_dta(stunsc, file = "clean/Student_NSC_Enrollment_Indicators.dta")
```