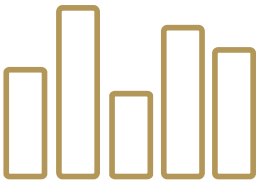


STRATEGIC DATA PROJECT

CONNECT: DATA LINKING GUIDE FOR COLLEGE-GOING

SDP TOOLKIT

FOR EFFECTIVE DATA USE IN EDUCATION AGENCIES

www.gse.harvard.edu/sdp/toolkit

Toolkit Documents

An Introduction to the SDP Toolkit for Effective Data Use

Identify: Data Specification Guide

Clean: Data Building Guide for College-Going

Connect: Data Linking Guide for College-Going

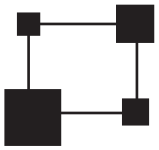
Analyze: College-Going Success Analysis Guide

Adopt: Coding Style Guide

SDP Stata Glossary

VERSION: 1.1
Last Modified: October 29, 2012

| Authored by Todd Kawakita and the SDP Research Team



Connect: SDP Data Linking Guide

Now that you identified and cleaned data, you will merge it together to create an analysis file.

Purpose

Connect links data elements from across your system into one analysis file. The file allows you to execute analyses inspired by the SDP College-Going Diagnostic to examine students' progression through high school and college.

After completing connect, you will have:

- Produced student-level files that track high school completion and graduation
- Linked postsecondary college enrollment and persistence data from the National Student Clearinghouse (NSC), to your agency's student achievement records
- Merged disparate data files to create a single analysis file to support **Analyze**.

The National Student Clearinghouse collects information on postsecondary enrollment for students across the country. To access your agency's data, please visit: **studentclearinghouse.org**.

At the end of **Connect**, you will have merged 7 files and generated the necessary variables (shaded in green) for the analysis file. A preview of the end product is below:

CG_Analysis																			
1	sid	15	first_hs_code	24	ontime_grad	30	test_math_8_raw	41-43	first_college_opeid_*	89-92	cum_credits_yr*
2	male	16	first_hs_name	25	late_grad	31	test_math_8	44-46	first_college_name_*	93-95	cum_credits_yr*_ela
3	race_ethnicity	17	last_hs_code	26	stil_enrl	32	test_math_8_std	47-49	enrl_1oct_grad_yr1_*	97-100	cum_credits_yr_math
4	hs_diploma	18	last_hs_name	27	transferout	33	test_ela_8_raw	50-52	enrl_1oct_grad_yr2_*	101-104	ontrack_endyr*
5	hs_diploma_type	19	longest_hs_code	28	dropout	34	test_ela_8	53-55	enrl_1oct_grad_yr3_*	105-108	status_after_yr*
6	hs_diploma_date	20	longest_hs_name	29	disappear	35	test_ela_8_std	56-58	enrl_1oct_grad_yr4_*	109	ontrack_hsgrad_sample
7	frpl_ever	21	last_wd_group			36	test_composite_8	59-61	enrl_ever_w2_grad_*	110-113	cum_gpa_yr*
8	iep_ever	22	chrt_ninth			37	test_composite_8_std	62-64	enrl_grad_persist_*	114	cum_gpa_final
9	ell_ever	23	chrt_grad			38	qrt_8_math	65-67	enrl_grad_all4_*	115	sat_act_concordance
10	gifted_ever					39	qrt_8_ela	68-70	enrl_1oct_ninth_yr1_*	116	highly_qualified
11	frpl_ever_hs					40	qrt_8_composite	71-73	enrl_1oct_ninth_yr2_*	117	ontrack_sample
12	iep_ever_hs							74-76	enrl_1oct_ninth_yr3_*		
13	ell_ever_hs							77-79	enrl_1oct_ninth_yr4_*		
14	gifted_ever_hs							80-82	enrl_ever_w2_ninth_*		
								83-85	enrl_grad_all4_*		
								86-88	enrl_ninth_all4_*		

Note: This guide references **Identify** and requires output from **Clean**. To move through **Connect**, you should review these stages of the toolkit.

Data and Structure

Connect consists of 10 steps to build one analysis files from various sources. After the first ten steps, there is also a section on producing indicators to on-track gradudation. The steps in **Connect** require data files from **Clean**.

Files needed in **Connect**:	Step in Connect:	High-Level Description:
Output file from **Clean** Task 5: Prior Achievement	1. Prior Achievement: Part 1	In Steps 1 and 2, you load 8th grade test scores and perform a "crosswalk" between numeric school codes and names.
School research file from **Identify**	2. School Crosswalk	
Output file from **Clean** Task 1: Student Attributes, Student_ Attributes_Clean.dta	3. Student Attributes	In Steps 3-5, you load Student Attributes data and then merge this with Student School Year and Student School Enrollment information. This creates a single file with information from all three datasets.
Output file from **Clean** Task 3: Defining the Grade Cohort	4. Student School Year	
Output file from **Clean** Task 4: Student School Enrollment	5. Student School Enrollment	
Output file from **Connect** Steps 3-4	6. High School Indicators and Outcomes	In Step 6, you use the file from Steps 3-4 to define key high school indicators and outcomes. These include first and last high school attended and graduation outcomes. You will also merge the School Crosswalk from Step 2 onto this file to obtain high school names using school codes.
Output file from **Connect** Steps 1 and 6	7. Prior Achievement: Part 2	In Step 7, you merge Prior Achievement data from Step 1 with data in memory.
	8. Examining the Analysis File: Part 1	In Step 8, you examine the analysis file that captures information through high school.
Student NSC Enrollment research file from **Identify**	9. National Student Clearinghouse Data	In Step 9, you load NSC data on college enrollment and persistence.
	10. Examining the Analysis File: Part 2	In Step 10, you merge the NSC data to the analysis file from Step 8. You will now have an analysis file that captures information through high school and college.
	Connect: On-Track Indicators	In Connect: On-Track Indicators, you will generate variables that will allow you to analyze on-track to graduation status.

Throughout **Connect** the term "merge" indicates that two files will be linked. Merging allows you to combine datasets horizontally and add new columns (or variables) from one dataset to another based on identifier(s) present in both. To merge you must start with a dataset "loaded" in memory.

Step Components

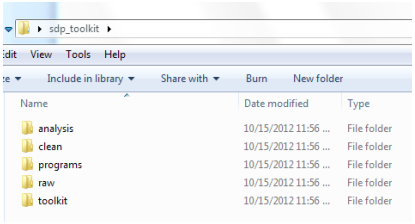
For each step, you will find the following:

- **Purpose**: an overview of each step;
- **Files Needed**: data elements or files required to complete each step;
- **After this step**: an overview of "output" generated by each step.

Also, throughout **Connect**, you will find Stata code to explain each of the sub-steps. Code appears in blue boxes, like below:

```
preserve

// keep only observations if 8th grade math score is not missing
keep if test_math_8 ~= .

// check to see if the file is unique by student id
isid sid
```

Infrastructure



In the infrastructure folder you unzipped from **www.gse.harvard.edu/sdp/toolkit,** look for the files in the **clean** folder and the files Connect.do and Connect_On_Track.do in the **programs** folder. The files in the **clean** folder are those you produced via Clean and that we have also provided for you. The do files provide a shell for you to fill in the ten steps of Connect and the section on On-Track indicators. Doing so will allow you to produce the CG_Analysis file which will be saved to the **analysis** folder.

As always, if you would like additional support from the friendly SDP team, please email us at **sdp@gse.harvard.edu**.

STEP 1: Prior Achievement Part 1

Purpose: Prepare 8th grade test scores for the analysis file.

Files needed: Prior_Achievement output file from Task 5 in **Clean**

After this step you will have a temporary file `tests' that contains prior achievement information for math, ELA, and math-ELA composite score.

1.1 Load the File and Check Uniqueness

Begin Step 1 of Connect by loading the Prior_Achievement file resulting from Task 5.

```
// load Prior Achievement
use "${clean}/Prior_Achievement.dta", clear
isid sid
```

Note the structure of the file. Thanks to Task 5, the file is unique by sid (highlighted in blue) and contains test scores for only 8th grade math , ELA and math-ELA composite. If your data is not structured like this, please review Task 5.

Raw scores (math_raw_score and ela_raw_score from Task 5) are not shown. You will primarily use scaled or standardized scores in future analyses. However, keep raw scores in your file to compare results between scaled, standardized, or raw scores later on.

1.2 Rename variables to indicate that they are 8th grade scores.

PRIOR ACHIEVEMENT
sid
school_year
grade_level
math_scaled_score
ela_scaled_score
composite_scaled_score
std_scaled_math
std_scaled_ela
std_composite_scaled_score

PRIOR ACHIEVEMENT
sid
school_year
grade_level
test_math_8
test_ela_8
test_composite_8
test_math_8_std
test_ela_8_std
test_composite_8_std

```
rename raw_score_math              test_math_8_raw
rename raw_score_ela               test_ela_8_raw
rename scaled_score_math           test_math_8
rename scaled_score_ela            test_ela_8
rename scaled_score_composite      test_composite_8
rename scaled_math_std             test_math_8_std
rename scaled_ela_std              test_ela_8_std
rename scaled_score_composite_std  test_composite_8_std
```

1.3 Define Prior Achievement Quartiles in Each Subject

Prior achievement quartiles have to be created by subject and by year. Since the process is the same for each subject, you can create a loop to get this done.

```
foreach subject in math ela composite {
```

Start by isolating the data fi le to a single exam subject (Math, ELA or Composite scores). In Stata syntax, this means you begin with a `preserve` command (This allows you to unisolate using `restore` later)to fi rst isolate 8th grade math scaled scores.

```
    preserve

    // keep only observations if 8th grade score is not missing
    keep if test_`subject'_8 != .
```

STEP 1: Prior Achievement Part 1

Next, create a variable to capture the quartile of an eighth grader's score in each subject, relative to peers who took the same test, the same school year.

This allows you to compare performance of each student to peers in the same year.

```
    // create a local variable containing all values of  school year in the test score file
    levelsof school_year, local(year)

    // capture the quartile of students' eighth grade test scores in each school year
    foreach yr of local year {
      xtile qrt_8_`subject'_`yr' = test_`subject'_8 if school_year == `yr' , nq(4)
    }
```

Because quartile variables are year specific (e.g. qrt_8_math_2007, qrt_8_math_2008, etc), you must create another variable, e.g. qrt_8_math, to compile year-specific quartiles into one variable across years.

```
    // create a variable that compiles all year-specific quartile variables into a single
    variable across years
    gen qrt_8_`subject' = .

    foreach yr of local year {
      replace qrt_8_`subject' = qrt_8_`subject'_`yr' if missing(qrt_8_`subject')
      drop qrt_8_`subject'_`yr'
    }
```

Save the data as a separate tempfile and then restore the data to its previous state. This is the end of the loop, so we close the bracket.

```
    // create and save a tempfile
    tempfile `subject'test
    save ``subject'test'

    restore
}
```

1.4 Bring the Subjects Together into a Single File

Now, you constructed three temporary files `mathtest', `elatest', and `compositetest' with appropriate quartile variables.

Merge the files together using the sid included in each temporary file. Because each temporary file is subject specific, identified by student ID (sid), and includes only one score per student, you may link these files with a `1:1` (one-to-one) relationship.

```
// merge the three subjects together into one test file
use `mathtest', clear
merge 1:1 sid using `elatest', nogen
merge 1:1 sid using `compositetest', nogen
```

You will not need the school year and grade level variables, so drop those.

```
drop school_year grade_level
```

`tests'
sid
test_math_8_raw
test_math_8
test_math_8_std
test_ela_8_raw
test_ela_8
test_ela_8_std
test_composite_8
test_composite_8_std
qrt_8_math
qrt_8_ela
qrt_8_composite

STEP 1: Prior Achievement Part 1

Order the variables in a sensical order

```
order sid  test_math* test_ela* test_composite* qrt*
```

Now, save the data as a tempfile `tests' and set the file aside for use later (Step 8).

```
tempfile tests
save `tests'
```

STEP 2: School Crosswalk

Purpose: Prepare a crosswalk* between school codes and names. This allows you to link a high school students' high school graduation with their college enrollment outcomes.

Files needed: School research file from **Identify**

After this step you will have created a temporary file `highschoolinfo' that contains a crosswalk between school codes and school names.

Load the File and Check Uniqueness

SCHOOL
school_code
school_year
school_name
grade_span

Load the School research file from **Identify**. Restrict the 'universe' of schools to only include high schools.

```
// load School
use "${clean}/School.dta", clear

// keep only schools that contain high school grades
keep if grade_span == "High"
drop grade_span
```

A crosswalk table ensures the final file is unique by school_code and that one school_code maps to one school_name. For example, in an uncleaned file, Albert Einstein High School might be spelled three ways, "A. Einstein HS," "Einstein High School," or "A.E. HS," but have one school_code. Alternatively, "Jones High School" might have a code of 153 and 154. You must fix these issues before moving on.

```
// keep only the school code and school name
keep school_code school_name
duplicates drop

// check that the file is unique by school_code
isid school_code
```

Generate First, Last, and Longest High School Variables

Next, generate three variables, **first_hs_code**, **last_hs_code**, and **longest_hs_code**. Set them equal to the school_code of each high school.

```
// creates first / last / longest hs id variables

foreach type in first last longest {
  gen `type'_hs_code = school_code
}
```

first_hs_code: When you attempt to understand high school outcomes, you traditionally assign students to their **first high school** attended. This is because these schools assume responsibility for students at the beginning of their high school career.

last_hs_code: When examining college enrollment and persistence outcomes, you assign students to their **last high school**. This is because these schools assume responsibility for preparing a student for college.

longest_hs_code: You will allow flexibility to change student assignment to schools by capturing students' **longest high school** attended. Using alternative assignments may be appropriate if student mobility is a concern in your system. (It is also useful to explore how school-level results vary by assignment for sensitivity analyses).

You will return to these variables in Step 6 to link high school names to students in the Student School Enrollment file. For now, save the file as a tempfile named 'highschoolinfo'.

```
// create and save a tempfile
tempfile highschoolinfo
save 'highschoolinfo'
```

`highschoolinfo'
school_code
school_name
first_hs_code
last_hs_code
longest_hs_code

STEP 3: Student Attributes

Purpose: Load Student Attributes data to obtain time-invariant information for students in the system.

Files needed: Student_Attributes output file from Task 1 in **Clean**

After this step you will have loaded the Student Attributes data into memory.

Load the File and Check Uniqueness

STUDENT ATTRIBUTES
sid
male
race_ethnicity
hs_diploma
hs_diploma_type
hs_diploma_date

Load the Student Attributes output file from Task 1. The file contains time-invariant info on students (i.e. high school graduation status, ninth grade cohort, gender, and race) and should be unique by sid.

```
// load the Student_Attributes output from Task 1
use "${clean}/Student_Attributes.dta", clear

// check that the file is unique by sid
isid sid
```

STEP 4: Student School Year

Purpose: Merge Student School Year data with Student Attributes data in memory and generate program participation status variables.

Files needed: Student_School_Year_Ninth output file from Task 3 in **Clean**

After this step: You will have merged Student School Year data with Student Attributes data into memory and generated variables that indicate if a student has ever been classified as FRPL (Free and Reduced Price Lunch), IEP(Individualized Education Plan), ELL(Engish Language Learner) or Gifted in the system or during high school.

Merge on Student School Year

STUDENT ATTRIBUTES
sid
male
race_ethnicity
hs_diploma
hs_diploma_type
hs_diploma_date

STUDENT SCHOOL YEAR
sid
school_year
grade_level
first_9th_school_year_observed
frpl
iep
ell
gifted
days_enrolled
days_absent
days_suspended_out_of_school

Merge the Student School Year file with the Student Attributes file in memory. This adds student-level information that may change from year to year (i.e FRPL,IEP, ELL or Gifted). This is a `1:m` (one-to-many) merge because the Student Attributes file is unique by sid and the Student School Year file is unique by sid + school_year.

```
// merge Student School Year from Task 3
onto Student Attributes
merge 1:m sid using "${clean}/Student_School_Year_
Ninth.dta"
```

Before conducting the merge, the Student School Year output file should be unique by sid and school_year and contain information on all available grades. The Student School Year data should also include the first_9th_school_year_observed variable.

STEP 4: Student School Year

Checking the Merge

In an ideal world records match perfectly. However, administrative records are often messy. Perfect merges rarely occur. Therefore, consider a merge satisfactory if at least 95% of students appear in both files.

The results of the merge can be checked by running a tabulation of the `_merge` variable. The `_merge` variable is automatically created when data files are merged.

Keep only students at the intersection of the two files.

```
// check the number and percentage of students appearing in both files
sort sid
tab _merge if sid!=sid[_n-1]

// (if sid!=sid[_n-1] counts unique students instead of observations; this should be familiar from
the tasks)
```

```
// once you have checked the _merge, keep only students at the intersection of both files
keep if _merge==3
drop _merge
```

Generate Program Participation Variables

Now, create binary variables (variables that assume values of 0 or 1) to indicate if a student ever:

1. qualified to participate in FRPL;
2. qualified for an IEP;
3. classified as ELL (or LEP);
4. qualified for gifted program.

These variables, (frpl_ever, iep_ever, ell_ever, and gifted_ever) allow you to explore high school and college outcomes for students that participated in these programs for one or more school years.

Create analogous variables to capture students' program participation status in high school.

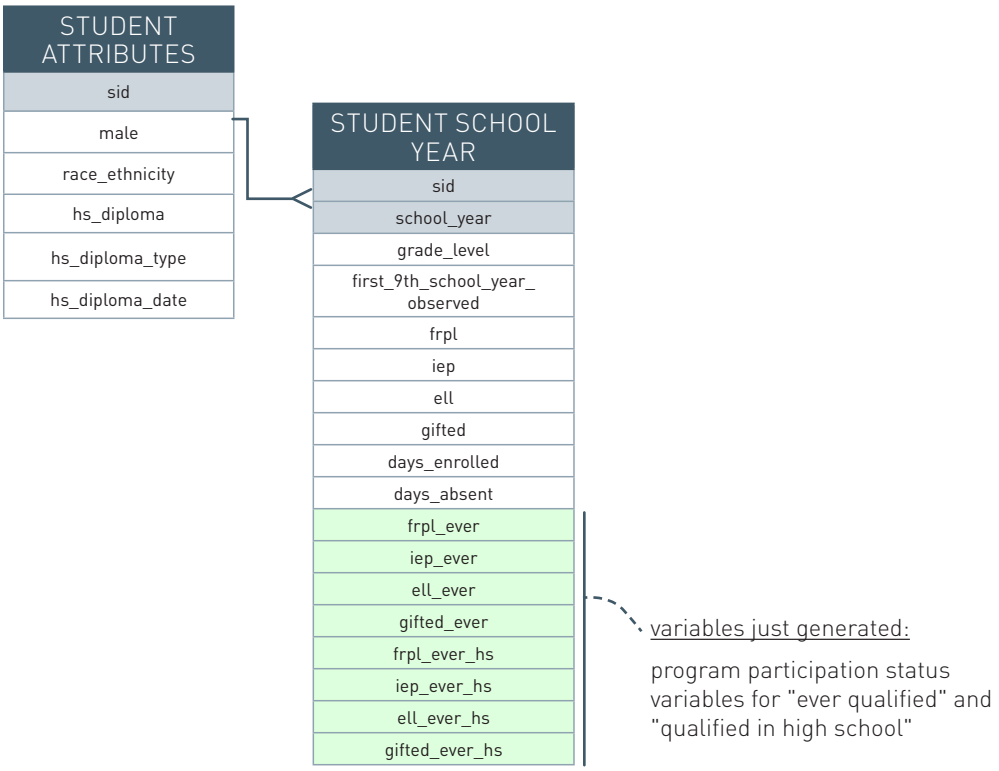
```
foreach var of varlist frpl iep ell gifted {

    // detect if the student has ever been flagged with a value of 1 for frpl / iep / ell / gifted
    bys sid: egen `var'_ever = max(`var')

    // detect for only high school grades
    gen temp_`var'_hs = `var' if grade_level >= 9 & grade_level <= 12

    // this populates the high school only variables across all observations within each student
    bys sid: egen `var'_ever_hs = max(temp_`var'_hs)
    replace `var'_ever_hs =0 if `var'_ever_hs ==.

    // drop the temporary variable
    drop temp_`var'_hs
}
```

STEP 4: Student School Year

High school status variables (frpl_ever_hs, iep_ever_hs, ell_ever_hs and gifted_ever_hs) allow flexibility in defining student subgroups. This is useful if participation data is missing non-randomly before a student enters high school, or if variables that capture participation are overly inclusive across years.

For example, a student who demonstrates limited English proficiency in 4th grade may be fluent in English by 9th grade. It may or may not be appropriate to categorize the student as ELL in analyses that examine high school outcomes.

Current State of the Analysis File



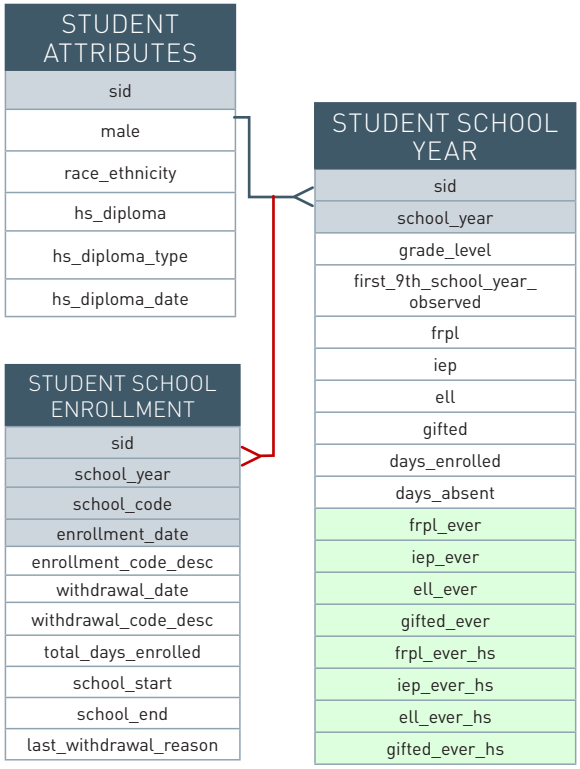
STEP 5: Student School Enrollment

Purpose: Merge Student School Enrollment with data in memory.

Files needed: Student_School_Enrollment_Clean output from Task 4 in **Clean**

After this step you will have merged Student School Enrollment data with data in memory.

Merge on Student School Enrollment



Merge the Student School Enrollment file onto the analysis file. This allows you to identify high schools students enrolled at different times.

This is a `1:m` merge, as the data file from the previous two steps is unique by sid + school_year and the Student School Enrollment file is unique by sid, school_year, school_code, and enrollment_date.

```
// merge on Student School Enrollment from Task 4
merge 1:m sid school_year using "${clean}/Student_
School_Enrollment_Clean.dta"
```

Before the merge, the Student School Enrollment file should be unique by sid, school_year, school_code, and enrollment_ date.

Checking the Merge

```
// check the number and percentage of students appearing in both files
sort sid
tab _merge if sid!=sid[_n-1]
```

Keep only students at the intersection of the two files.

```
// once you have checked the _merge, keep only students at the intersection of both files
keep if _merge==3
drop _merge
```

STEP 6: High School Indicators and Outcomes

Purpose: Generate high school indicators and outcomes in two categories.

Files needed: The file in memory from Steps 3-5 and `highschoolinfo' from Step 2

After this step you will have created a number of high school indicators and outcomes: 1) first, last, and longest high school; 2) 9th grade and graduation cohorts; and 3) end of high school outcomes: ontime and late graduates and high school enrollment outcomes for non-graduates.

6.1 Define First, Last and Longest High School

To begin, make sure that the data includes only student observations in high school. (You have done this in Clean already; we are checking it again here).

```
// restrict to only high school
keep if grade_level >= 9 & grade_level!=.
```

There might be students who are assigned to high schools but whose attendance duration is 0. Drop these school assignments to ensure that you assign students to high schools they actually attended.

```
drop if days_enrolled == 0
```

Define First High School

To identify a student's first high school, determine the first enrollment episode for the student.

In some cases, students enroll in more than one school at the same time. In such cases, assign them to the school where they attended longest.

Should students have multiple first enrollments of the same length, randomly assign them to one of these schools.

```
gsort sid school_year enrollment_date -days_enrolled
bys sid: gen temp_first_hs_code = school_code if _n==1
egen first_hs_code = max(temp_first_hs_code), by(sid)
```

Note, that by not specifying a school code in the sort, you allow the program to randomly sort schools among student records that have the same school year, enrollment date and total days enrolled.

Define Last High School

To identify a student's last high school, determine the last enrollment episode for the student.
In cases of joint enrollment, use the school where the student attended longest.
Where joint enrollment duration is the same, randomly assign the last high school.

```
gsort sid -school_year -withdrawal_date -days_enrolled
bys sid: gen temp_last_hs_code = school_code if _n==1
egen last_hs_code = max(temp_last_hs_code), by(sid)
```

STEP 6: High School Indicators and Outcomes

Define Longest High School

To determine the longest enrolling HS, you first have to add up all enrollments within a HS.

Since in Clean you ensured that there are no overlapping enrollments within a school, you can add enrollments up.

In cases where students enrolled in more than one school for the same amount of time, randomly assign the longest high school

```
egen total_days_enrolled_in_school = sum(days_enrolled), by(sid school_code)
gsort sid -total_days_enrolled_in_school
bys sid: gen temp_longest_hs_code = school_code if _n==1
egen longest_hs_code = max(temp_longest_hs_code), by(sid)
```

Question 6.1 Test your understanding by filling the shaded areas below.

sid	school_ code	enrollment_ date	total_ days_en- rolled	temp_ first_hs_ code	first_hs_ code	temp_ last_hs_ code	last_hs_ code	total_days_ enrolled_ in_school	temp_ longest_hs_ code	longest_ hs_code
41	430	10-Aug-04	344							
41	460	10-Aug-04	287							
41	460	14-Aug-05	281							
41	460	12-Aug-05	283							
41	460	20-Aug-07	282							

Now, you can drop the temporary variables.

```
// delete temporary variables created
drop temp*
```

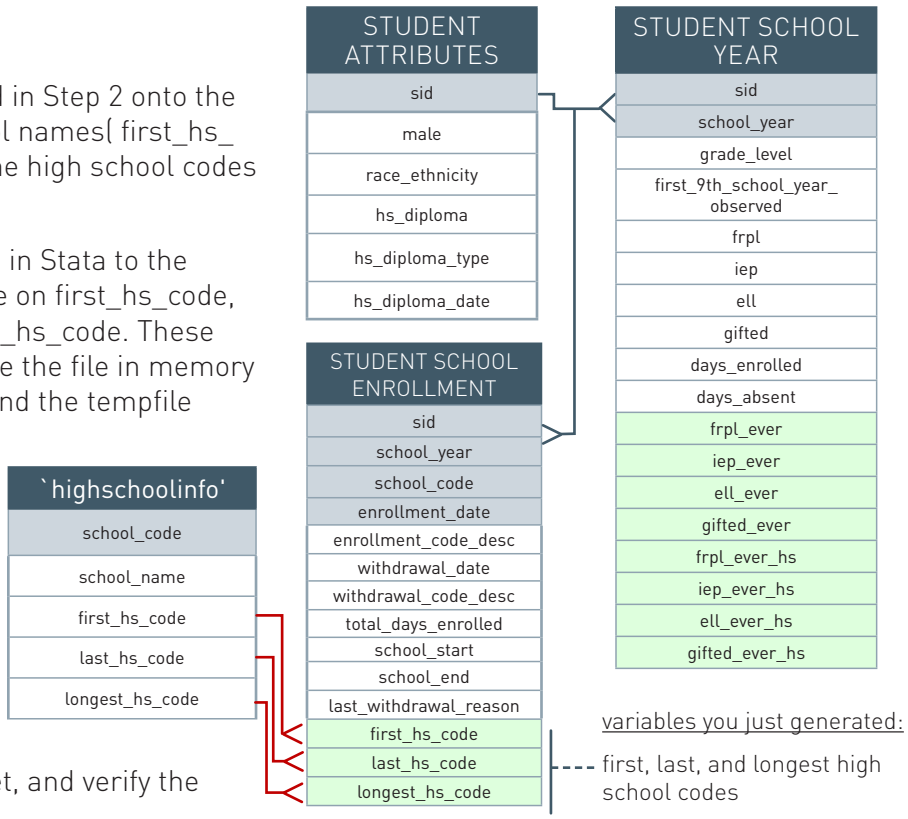
STEP 6: High School Indicators and Outcomes

Merge on `highschoolinfo'

Merge the `highschoolinfo' tempfile created in Step 2 onto the current file. This allows you to obtain school names(first_hs_ name and last_hs_name) associated with the high school codes just captured.

This requires merging data currently loaded in Stata to the `highschoolinfo' tempfile three times – once on first_hs_code, then on last_hs_code, and finally on longest_hs_code. These merges will all be `m:1` (many to one) because the file in memory contains multiple observations per school and the tempfile contains only one per school.

```
// Merge on `highschoolinfo'
foreach type in first last longest {
     merge m:1 `type'_hs_code using
     `highschoolinfo', gen(_m`type')
     rename school_name `type'_hs_
     name
}
```



By specifying gen(_m`type'), you can keep all three merge variables in the dataset, and verify the return of each merge.

```
Result                           # of obs.
-----------------------------------------
not matched                             434
    from master                         434  (_mfirst==1)
from using                                0  (_mfirst==2)

matched                              53,140  (_mfirst==3)
-----------------------------------------


Result                           # of obs.
-----------------------------------------
not matched                              11
    from master                          11  (_mlast==1)
from using                                0  (_mlast==2)

matched                              53,563  (_mlast==3)
-----------------------------------------


Result                           # of obs.
-----------------------------------------
not matched                              55
    from master                          55  (_mlongest==1)
from using                                0  (_mlongest==2)

matched                              53,519  (_mlongest==3)
-----------------------------------------
```

STEP 6: High School Indicators and Outcomes

After each merge drop observations that do not show up in the enrollment data, as well as observations for which you cannot assign high school names (students who have school_codes not listed in the `highschoolinfo' file). Also, drop students that had neither a first nor last high school defined. These restrictions should eliminate very few students from the analysis file.

```
// Keep only observations that merged (Drop all observations only from the school file,
// and drop all observations for which a first, last, or longest high school cannot be defined

keep if _mfirst == 3 & _mlast == 3 & _mlongest == 3
drop _m*
```

6.2 Assign Ninth Grade and Graduation Cohorts

Assigning students to cohorts will allow you to calculate various indicators (e.g. high school graduation, college enrollment) using a different set of students in the denominator. For example, when calculating college enrollment, you could use the ninth grade cohort to illustrate how high schools prepared their incoming freshmen for future success, or you could use the graduating cohort to illustrate the percentage of a high school's graduates enrolling in college.

Since the ninth grade cohort is equal to first_9th_school_year_observed in the student attributes file, just rename first_9th_school_year_observed to chrt_ninth.

```
// define ninth grade cohort
rename first_9th_school_year_observed chrt_ninth
```

The graduation cohort variable, chrt_grad, is the school year in which a student graduated. If a student obtained a diploma prior to September 1st, the chrt_grad variable is the same as the year of hs_diploma_date. If a student received a diploma between September 1st and December 31st treat them as graduates for the next school year.

```
// define graduation cohort
gen chrt_grad =.
replace chrt_grad = year(hs_diploma_date)      if month(hs_diploma_date) < 9
replace chrt_grad = year(hs_diploma_date) + 1  if month(hs_diploma_date) >=9
```

Question 6.2 Test your understanding by filling the shaded areas below.

sid	hs_diploma_date	first_9th_school_year_observed	chrt_ninth	chrt_grad
16305	.	2007		
16306	14-May-08	2005		
16307	21-Dec-08	2005		

Note, that in Clean, you assigned every student a first_ninth_school_year_observed. You either had their first night grade in the data, or if they transferred into the district later in high school, you backward mapped them to an appropriate ninth grade school year. Thus, chrt_ninth should never be missing. Chrt_grad, however, could be missing, because not all students graduated, thus they will not be assigned a graduating cohort.

STEP 6: High School Indicators and Outcomes

6.3 Define High School Outcomes

To determine end of high school outcomes, you need a student's last withdrawal code (withdrawal code at their last high school). Use the last withdrawal code to determine if the student graduated, transferred out, dropped out, or has another outcome.

6.3a Group last withdrawal codes together into four end of high school outcomes:

1 = Graduated (define graduated using withdrawal data and hs_diploma in the Student Attributes file, or any other source of graduation information used)

2 = Transfer Out

3 = Drop Out

4 = Other (all other reasons for withdrawal)

Outcomes are captured using the last_wd_group variable. Assigning the last withdrawal code to last_wd_group requires an understanding of decision rules in your agency. Some withdrawal codes may be ambiguous or redundant and need to be combined to fit under the four categories. Therefore, it is important to elicit help from those knowledgeable of local graduation, transfer, and dropout policies in your agency. For example, there may be special codes for students who are incarcerated or pass away that are not well-documented. We provide an example below, but you will have to customize this script based on values your agency uses. Particularly, for dropouts you should make sure the agency is not being penalized for something it does not have control over.

First, examine the values for last_withdrawal_reason. You will have to make sure that you capture all these values in defining the last withdrawal groups. Should your data have any missing values for last_withdrawal_reason, be sure to assign those to an appropriate category. In some agencies, a missing withdrawal code indicates that the student is still enrolled, so assign them as still enrolled.

```
sort sid
tab last_withdrawal_reason if sid != sid[_n-1], m
```

```
       last_withdrawal_reason |      Freq.     Percent        Cum.
------------------------------+-----------------------------------
                   Absenteeism |        709        3.32        3.32
                         Death |         20        0.09        3.41
                     Expulsion |         39        0.18        3.60
GED/Certificate of Completion |        629        2.95        6.54
       Graduated with Diploma |      4,906       22.98       29.52
                   Home School |         17        0.08       29.60
                       No Show |        650        3.04       32.64
                         Other |        528        2.47       35.12
                Other Transfer |        525        2.46       37.57
            Promoted End Year |      6,371       29.84       67.41
            Retained in Grade |      1,679        7.86       75.28
                    Suspension |        201        0.94       76.22
          Transfer In District |        427        2.00       78.22
     Transfer Out of District |      4,651       21.78      100.00
------------------------------+-----------------------------------
                        Total |     21,352      100.00
```

STEP 6: High School Indicators and Outcomes

```
gen last_wd_group = .
label define lastwd 1 "Graduated" 2 "Transfer Out" 3 "Drop Out" 4 "Other"
label values last_wd_group lastwd

replace last_wd_group = 2 if      last_withdrawal_reason == "Home School" | ///
                                  last_withdrawal_reason == "Other Transfer" | ///
                                  last_withdrawal_reason == "Transfer Out of District" | ///
                                  last_withdrawal_reason == "Death"

replace last_wd_group = 3 if      last_withdrawal_reason == "Absenteeism" | ///
                                  last_withdrawal_reason == "No Show" | ///
                                  last_withdrawal_reason == "Expulsion"

replace last_wd_group = 1 if      hs_diploma == 1

replace last_wd_group = 4 if      last_wd_group == .

assert !mi(last_wd_group)
```

Note that we populated the graduating group last; this is because the evidence of a high school diploma overrides any other value for last withdrawal reason. Any remaining last_withdrawal_reason values were then classified as 4, Other.

STEP 6: High School Indicators and Outcomes

6.3b Define High School Outcomes

Identify On-Time and Late High School Graduates

First, identify students who graduated within 4 years of entering high school (on-time graduates) as well as students who took more than 4 years (late graduates). These variables allow you to examine time taken to complete high school and explore how this varies across high schools within a system. These two variables have to add up to the total graduates.

```
// define on-time graduates
gen ontime_grad = (chrt_ninth >= (chrt_grad - 3) & chrt_ninth !=. & chrt_grad!=. & hs_diploma==1)

// define late graduates
gen late_grad = (ontime_grad==0 & chrt_ninth !=. & chrt_grad!=. & hs_diploma==1)

assert ontime_grad + late_grad == hs_diploma
```

Identifying High School Enrollment Outcomes for Non-Graduates

Next, assign high school enrollment outcomes for students who have not graduated by a point in time. You may define this point, but typically it is the current year if data is up to date.

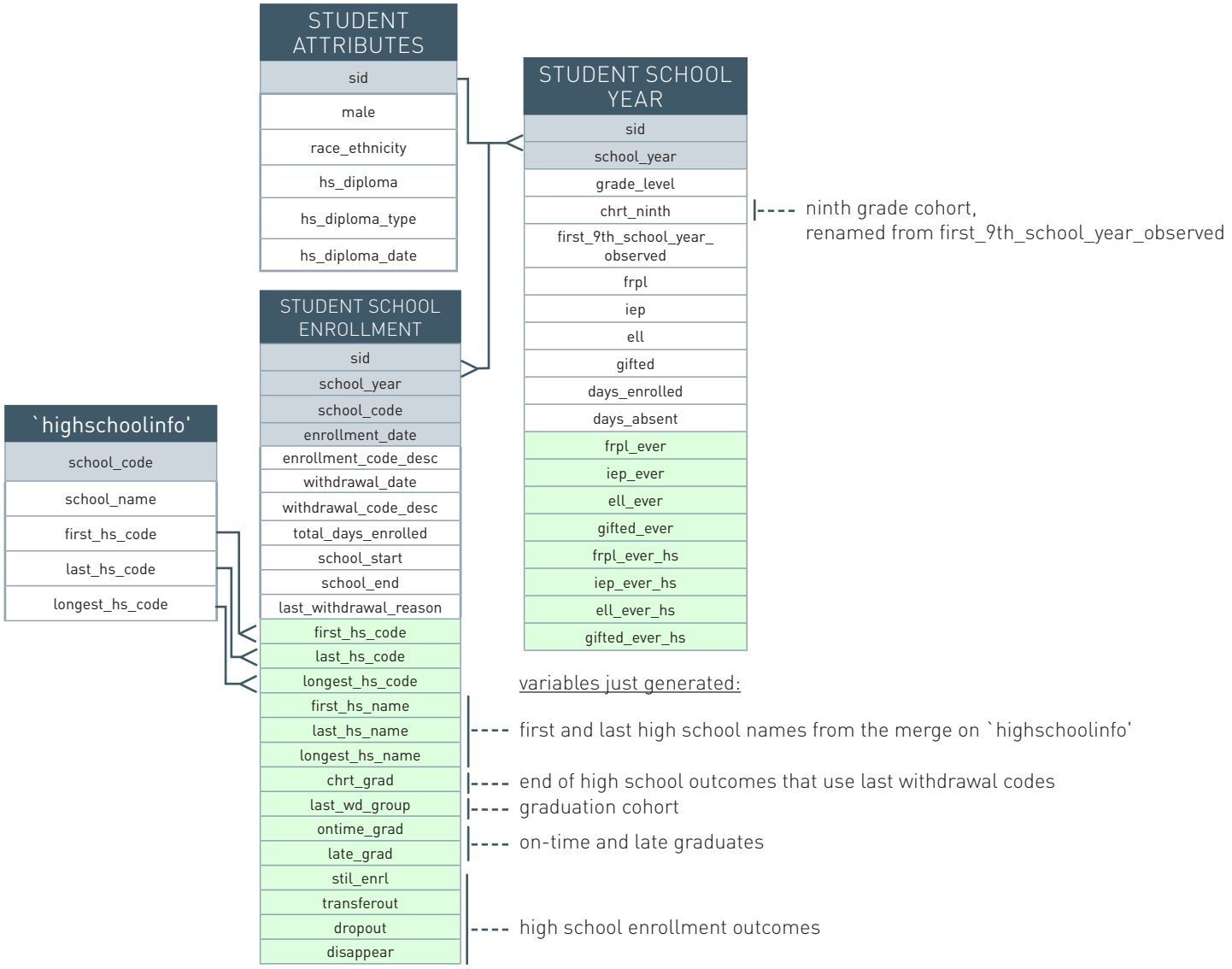
It is important that each student is either marked as a graduate or assigned to only one of the following categories. Notice how the definition of each category is conditional on all previous categories.

```
// still enrolled
egen last_schyr_stu = max(school_year), by(sid)
egen last_schyr_data = max(school_year)
gen still_enrl = (last_schyr_stu == last_schyr_data & hs_diploma!=1)

// transfer out
gen transferout = last_wd_group == 2 & hs_diploma!=1 & still_enrl!=1

// drop out
gen dropout = last_wd_group == 3 & hs_diploma!=1 & still_enrl!=1 & transferout!=1

// disappear
gen disappear = (hs_diploma!=1 & still_enrl!=1 & transferout!=1 & dropout!=1)

assert (hs_diploma + still_enrl +  transferout + dropout + disappear) == 1
```

STEP 6: High School Indicators and Outcomes

Current State of the Analysis File

Congratulations on generating a plethora of high school indicators and outcomes!



STEP 6: High School Indicators and Outcomes

You have generated most of the key high school indicators and outcomes, so you no longer need all source variables. Keep only the variables listed here.

	CURRENT ANALYSIS FILE
1	sid
2	male
3	race_ethnicity
4	hs_diploma
5	hs_diploma_type
6	hs_diploma_date
7	chrt_ninth
8	frpl_ever
9	frpl_ever_hs
10	iep_ever
11	iep_ever_hs
12	ell_ever
13	ell_ever_hs
14	gifted_ever
15	gifted_ever_hs
16	first_hs_code
17	last_hs_code
18	longest_hs_code
19	first_hs_name
20	last_hs_name
21	longest_hs_name
22	chrt_grad
23	late_wd_group
24	ontime_grad
25	late_grad
26	stil_enrl
27	transferout
28	dropout
29	disappear

Based on these time-invariant variables, the file is now unique by sid. To drop other variables, first keep the 30 then drop duplicates.

```
// keep time-invariant variables
// the "*" symbol is a wildcard to indicate multiple variable names
keep sid male race_ethnicity hs_diploma* *_ever *_ever_* *_hs_code ///
last_wd_group *_hs_name chrt_* *_grad still_enrl transferout dropout
disappear

// drop any duplicates
duplicates drop
```

Next, do one last check to make sure the file is unique by sid and then save the file as `analysis'.

```
// make sure the file is unique by sid
isid sid
```

STEP 6: High School Indicators and Outcomes

SOLUTIONS

6.1

sid	school_ code	enrollment_ date	total_ days_en- rolled	temp_ first_hs_ code	first_hs_ code	temp_ last_hs_ code	last_hs_ code	total_days_ enrolled_ in_school	temp_ longest_hs_ code	longest_ hs_code
41	430	10-Aug-04	344	430	430		460	344		460
41	460	10-Aug-04	287		430		460	1133		460
41	460	14-Aug-05	281		430		460	1133	460	460
41	460	12-Aug-05	283		430		460	1133		460
41	460	20-Aug-07	282		430	460	460	1133		460

6.2

sid	hs_diploma_date	first_9th_school_year_observed	chrt_ninth	chrt_grad
16305	.	2007	2007	.
16306	14-May-08	2005	2005	2008
16307	21-Dec-08	2005	2005	2009

STEP 7: Prior Achievement Part 2

Purpose: Merge prior achievement test scores onto the analysis file.

Files needed: the analysis file in memory from Step 6 and `tests' from Step 1

After this step you will have merged prior achievement data with the current analysis file.

	CURRENT ANALYSIS FILE
1	sid
2	male
3	race_ethnicity
4	hs_diploma
5	hs_diploma_type
6	hs_diploma_date
7	chrt_ninth
8	frpl_ever
9	frpl_ever_hs
10	iep_ever
11	iep_ever_hs
12	ell_ever
13	ell_ever_hs
14	gifted_ever
15	gifted_ever_hs
16	first_hs_code
17	last_hs_code
18	longest_hs_code
19	first_hs_name
20	last_hs_name
21	longest_hs_name
22	chrt_grad
23	late_wd_group
24	ontime_grad
25	late_grad
26	stil_enrl
27	transferout
28	dropout
29	disappear

`tests'
sid
test_math_8_raw
test_math_8
test_math_8_std
test_ela_8_raw
test_ela_8
test_ela_8_std
test_composite_8
test_composite_8_std
qrt_8_math
qrt_8_ela
qrt_8_composite

Merge on `tests'

You have a data set unique by sid that contains key student attributes, high school indicators and outcomes. All you need now are 8th grade test scores.

To add prior achievement scores to the file, merge the `tests' tempfile from Step 1 onto the current analysis file. This is a `1:1` merge on sid.

```
// merge `tests' onto the current file
merge 1:1 sid using `tests'
```

Next, drop students who do not appear in the analysis file but have 8th grade test scores. You may expect to capture prior achievement for most students, but not all students will have score information. For example, students who first enroll in the system during high school (after 8th grade) or were exempt from tests will not have prior test scores.

```
// drop students who do not appear in the analysis file
but have 8th grade test scores
drop if _merge==2
```

Review the merge results, to see what percentage of students have prior scores.

```
tab _m
```

```
        _merge |      Freq.     Percent        Cum.
---------------+-----------------------------------
master only (1) |      6,518       30.52       30.52
    matched (3) |     14,835       69.48      100.00
---------------+-----------------------------------
          Total |     21,353      100.00
```

In this case, almost 70% of high school students have prior achievement data.

Drop the merge variable

```
drop _m
```

STEP 8: Examining the Analysis File Part 1

Purpose: Admire your work and familiarize yourself with the variables generated.

Congratulations! You now finished working with agency administrative records and can save a preliminary analysis file to generate analyses on student transitions through high school completion and college going success. First, order the variables in a sensible way.

```
order sid male race_ethnicity hs_diploma hs_diploma_type hs_diploma_date ///
    frpl_ever iep_ever ell_ever gifted_ever frpl_ever_hs iep_ever_hs ell_ever_hs gifted_ever_hs ///
    first_hs_code last_hs_code longest_hs_code first_hs_name last_hs_name longest_hs_name ///
    last_wd_group chrt_ninth chrt_grad ontime_grad late_grad still_enrl transferout dropout
disappear
```

Then, save the analysis file.

```
save "${analysis}/Student_Collegegoing.dta", replace
```

All that is left to do is process college enrollment records from the National Student Cleainghouse (NSC), and merge these data onto the Student_CollegeGoing file. This creates a single analysis file to generate analyses on student transitions through high school and college.

Before moving on, take a moment to admire your work and refamiliarize yourself with sources and processes for each of these variables. Ask yourself: What research files were the variables produced from? How were high school indicators and outcomes created?

Student_CollegeGoing							
1	sid	12	iep_ever_hs	22	chrt_ninth	30	test_math_8_raw
2	male	13	ell_ever_hs	23	chrt_grad	32	test_math_8
3	race_ethnicity	14	gifted_ever_hs	24	ontime_grad	33	test_math_8_std
4	hs_diploma	15	first_hs_code	25	late_grad	34	test_ela_8_raw
5	hs_diploma_type	16	first_hs_name	26	stil_enrl	35	test_ela_8
6	hs_diploma_date	17	last_hs_code	27	transferout	36	test_ela_8_std
7	frpl_ever	18	last_hs_name	28	dropout	37	test_composite_8
8	iep_ever	19	longest_hs_code	29	disappear	38	test_composite_8_std
9	ell_ever	20	longest_hs_name			39	qrt_8_math
10	gifted_ever	21	last_wd_group			40	qrt_8_ela
11	frpl_ever_hs					41	qrt_8_composite

Use the following questions for the numbered variables above to guide your thoughts:

1. (1-6): Which research file are these variables from?
2. (7-14): Which research file are these variables from?
3. (15-17): How are first, last, and longest high school codes identified?
4. (18-20): From what research file are first, last, and longest high school names obtained?
5. (21): What does the last_wd_group variable describe?
6. (22-23): How are 9th grade and graduation cohorts defined?
7. (24-29): How are graduation and high school enrollment outcomes defined?
8. (30-40): Which research file are these variables from?

STEP 9: National Student Clearinghouse Data

Purpose: Generate college enrollment and persistence indicators

Files needed: The Student_NSC_Enrollment_Indicators data from Task 7 in Clean, and the Student_CollegeGoing file you saved in Step 8.

After this step you have created the indicators that will be used for college going analysis.

For all NSC related analyses, we create two types of indicators: one bases on the graduating cohort, and another based on the ninth grade cohort. Each of these indicators serves a different purpose, and can be used to answer different questions. For example, if you are interested in how high schools or the entire agency is doing in enrolling their graduates to college, you would be using the indicators based on the graduating cohort. If, however, you want to evaluate how the high schools or agency is preparing their incoming freshman to go through high school and enroll in college, you will use the indicators bases on the night grade cohort.

In addition, we create separate indicators to evaluate how soon after high school students enroll in college. One set of indicators is based on enrollment on October 1st. The second set of indicators is based on enrollment within two years of graduation. This latter indicator is calculated based on the calendar date of the student's high school graduation (or, in case of the ninth grade cohort, the expected on-time high school graduation).

To begin this step, open the Student_NSC_Enrollment_Indicators file from Task 7, and merge the Student_CollegeGoing file you saved in Step 8.

```
use "${clean}/Student_NSC_Enrollment_Indicators.dta", clear

// merge on variables needed from Student_College_Going
merge m:1 sid using Student_CollegeGoing.dta, keepusing(hs_diploma_date hs_diploma chrt_grad chrt_
ninth)
```

Only keep students who appear in both dataset.

```
keep if _m==3
drop _m
```

9.1 Create a variable to indicate if the student enrolled in college within two years of graduating from high school.

Start with the graduating cohort. If the student enrolled in college within 2*365 days after high school graduation, set the indicator to 1.

```
// create and indicator to show if the student enrolled within two years of HS graduation
gen enrl_ever_w2_grad = (first_enrl_date_any < (hs_diploma_date + (365 * 2))) & ///
             !mi(hs_diploma_date) & !mi(first_enrl_date_any)
```

For the ninth grade cohort, first create a variable that represents on-time graduation. Set this date to September 1st the fourth year after the student's ninth grade cohort. Then create the enrollment indicator using this date.

STEP 9: National Student Clearinghouse Data

```
// identify the date that would represent ontime high school graduation for students
gen ontime_yr = (chrt_ninth + 3) if !mi(chrt_ninth)
gen ontime_date = mdy(9,1,ontime_yr)
format ontime_date %d

// create and indicator to show if the student enrolled within two years of expected HS graduation
gen enrl_ever_w2_ninth = (first_enrl_date_any < (ontime_date + (365 * 2))) & ///
             !mi(ontime_date) & !mi(first_enrl_date_any)
```

Question 9.1 Test your understanding by filling the shaded areas below.

sid	chrt_ ninth	chrt_grad	hs_diploma_ date	first_enrl_ date_any	enrl_ever _w2_grad	ontime _yr	ontime _date	enrl_ever _w2_ninth
15647	2005	2008	14-May-08	30-Aug-08				
15656	2005	2007	9-May-07	17-Jan-10				
15658	2006	2009	17-May-09					

9.2 Create variables to indicate if the student was enrolled in college by Oct 1

This variable indicates if a student enrolled in college immediately after high school. In addition, we use indicators based on October 1st college enrollment to track if student persisted in college. For this, we need to create variables to indicate enrollment on October 1st the 1st, 2nd, 3rd and 4th year after high school graduation.

First, create placeholder variables for both the ninth and graduating cohort, for each of the four years.

```
// Create the 4 enrollment outcomes of interest by October 1st
foreach num of numlist 1/4 {
     gen enrl_1oct_grad_yr`num' = .
     gen enrl_1oct_ninth_yr`num' = .
}
```

Now, loop through these year values. We replace the above created placeholder variables with 1 if the student enrolled in college as of October 1st and the student's enrollment for that year ended after October 1st. We do this for the 1st, 2nd, 3rd and 4th year after high school graduation. As we loop through the variables, we have to make sure that we are only replacing values for students who graduated in the year in the current loop, so we have to set a condition that ensures that the record is for a student whose diploma date falls in that school year.

STEP 9: National Student Clearinghouse Data

```
    // loop through the graduation years (actual and expected) that exist in your data.

      // actual
      levelsof chrt_grad, local(chrt_grad_values)

      foreach yr of local chrt_grad_values {

            // assign the outcome of interest to assume a value of 1 when students enroll on or
before October 1st
            local yr1 = `yr'
            local yr2 = `yr' + 1
            local yr3 = `yr' + 2
            local yr4 = `yr' + 3
            local yr5 = `yr' + 4

            foreach num of numlist 1/5 {
                  replace enrl_1oct_grad_yr`num' = 1 if (n_enroll_begin_date <=
td(1oct`yr`num'')  & (td(1oct`yr`num'')  <= n_enroll_end_date) & ((year(hs_diploma_date) ==
(`yr') & month(hs_diploma_date) <=9) | ((year(hs_diploma_date) == `yr'-1 & month(hs_diploma_
date)>9)))
            }
      }

      // expected
      levelsof ontime_yr, local(chrt_ninth_values)

      foreach yr of local chrt_ninth_values {

            //assign the outcome of interest to assume a value of 1 when students enroll on or
before October 1st
            local yr1 = `yr'
            local yr2 = `yr' + 1
            local yr3 = `yr' + 2
            local yr4 = `yr' + 3
            local yr5 = `yr' + 4

            foreach num of numlist 1/5 {
                  replace enrl_1oct_ninth_yr`num' = 1 if (n_enroll_begin_date <=
td(1oct`yr`num'')  & (td(1oct`yr`num'')  <= n_enroll_end_date) & chrt_ninth == (`yr' - 3)
            }
      }
```

Question 9.2 Test your understanding by filling the shaded areas below.

sid	chrt_ grad	hs_diploma_ date	first_enrl_ date_any	n_enrl_ begin_date	enrl_1oct _grad_yr1	enrl_1oct _grad_yr2	enrl_1oct _grad_yr3	enrl_1oct _grad_yr4
16011	2008	14-May-08	23-Aug-08	23-Aug-08				
16011	2008	14-May-08	23-Aug-08	17-Jan-09				
16011	2008	14-May-08	23-Aug-08	22-Aug-09				
16011	2008	14-May-08	23-Aug-08	16-Jan-10				
16016	2009	14-May-08	23-Aug-09	22-Aug-09				
16016	2009	14-May-08	23-Aug-09	16-Jan-10				

STEP 9: National Student Clearinghouse Data

9.3 Collapse and reshape the data to make it unique by student id

At this point, you have all the values that were depended on individual college enrollment episodes. Now we will make the data unique by student, ensuring that we have indicators for 2-year, 4-year and any college.

Create an indicator that specifies the type of college. Make this the highest level of enrollment.

```
gen type = ""
replace type = "_none" if n_college_2yr == 0 & n_college_4yr == 0
replace type = "_2yr" if n_college_2yr == 1
replace type = "_4yr" if n_college_4yr == 1
```

Collapse the data by the invariant variables to populate the enrollment indicators for each record by student and college type.

Note, that when you collapse, only variables specified in the collapse command remain in your data.

```
collapse (max) enrl_*, by(sid  chrt_grad chrt_ninth hs_diploma_date first_college* type)
```

Reshape the data to have one record per student, and separate indicators for the type of college.

```
reshape wide enrl* , i(sid chrt_grad chrt_ninth hs_diploma_date first_college*) j(type) string
```

We do not need variables that refer to no college, so we can drop those.

```
drop *_none
```

Ensure that your data is unique by student id.

```
isid sid
```

STEP 9: National Student Clearinghouse Data

Question 9.3 Test your understanding by filling the shaded areas after each operation.

Original data:

sid	type	enrl_ever _w2_ninth	enrl_1oct _ninth_yr1	enrl_1oct _ninth_yr2	enrl_1oct _ninth_yr3	enrl_1oct _ninth_yr4
41	_2yr	1	1	.	.	.
41	_2yr	0	.	1	.	.
41	_2yr	1	.	1	.	.
41	_4yr	1	.	.	1	.

After collapse:

sid	type	enrl_ever _w2_ninth	enrl_1oct _ninth_yr1	enrl_1oct _ninth_yr2	enrl_1oct _ninth_yr3	enrl_1oct _ninth_yr4
41	_2yr					
41	_4yr					

After reshape:

sid	type	enrl_ever _w2_ ninth _2yr	enrl_1oct _ninth_ yr1 _2yr	enrl_1oct _ninth_ yr2 _2yr	enrl_1oct _ninth_ yr3 _2yr	enrl_1oct _ninth_ yr4 _2yr	enrl_ever _w2_ ninth _4yr	enrl_1oct _ninth_ yr1 _4yr	enrl_1oct _ninth_ yr2 _4yr	enrl_1oct _ninth_ yr3 _4yr	enrl_1oct _ninth_yr4 _4yr
41											

9.4 Ensure mutual exclusivity of 2-year and 4-year college enrollment

To ensure that in analyses that compare list both 2-year and 4-year enrollment students are not double counted, you have to ensure that these indicators are mutually exclusive. If have a student enrolled in both 2-year and 4-year college, report the 4-year.

```
foreach chrt in grad ninth {
    local variablelist "enrl_ever_w2_`chrt'  enrl_1oct_`chrt'_yr1  enrl_1oct_`chrt'_yr2
enrl_1oct_`chrt'_yr3  enrl_1oct_`chrt'_yr4"
    foreach var of local variablelist {
            replace `var'_2yr = 0 if `var'_2yr==1 & `var'_4yr==1
            assert (`var'_2yr + `var'_4yr)==1 | (`var'_2yr + `var'_4yr)==0 | (`var'_2yr +
            `var'_4yr)==.
    }
}
```

STEP 9: National Student Clearinghouse Data

```
// set missing values to 0
foreach chrt in grad ninth {
     foreach var of varlist enrl_ever_w2* enrl* {
               replace `var' = 0 if `var'==.
     }
}
```

9.5 Create "any college" version of the variables

At this point, we have 2-year and 4-year variables. We create "any college" variables by setting them to 1 if either the 2-year or the 4-year variable is 1.

```
// create an any college version of the year-by-year college enrollment outcome
foreach chrt in grad ninth {
     foreach i of numlist 1/4 {
               egen enrl_1oct_`chrt'_yr`i'_any = rowmax(enrl_1oct_`chrt'_yr`i'_2yr
               enrl_1oct_`chrt'_yr`i'_4yr)
     }
}

// create an any college version of the within 2 years enrollment outcome foreach chrt in grad
ninth {
     egen enrl_ever_w2_`chrt'_any = rowmax(enrl_ever_w2_`chrt'_2yr enrl_ever_w2_`chrt'_4yr)
}
```

9.6 Create persistence outcomes for graduates and ninth graders

We consider that a student has persisted to the second year of college if they were enrolled in college on October 1st after graduation, and were also enrolled on October 1st one year later.

Apply the same logic to creating a variable that indicates continuous enrollment over four years in college.

```
foreach chrt in grad ninth {
     // create persistence outcomes to the second year of college
     foreach type in 4yr 2yr any {
               gen enrl_`chrt'_persist_`type' = (enrl_1oct_`chrt'_yr1_`type' == 1 &
               enrl_1oct_`chrt'_yr2_any == 1) if !mi(chrt_`chrt')

               // create persistence outcomes that denote continuous enrollment over four
               consecutive years in college
               gen enrl_`chrt'_all4_`type' =  (enrl_1oct_`chrt'_yr1_`type' == 1 & ///
                         enrl_1oct_`chrt'_yr2_`type' == 1 & ///
                         enrl_1oct_`chrt'_yr3_`type' == 1 & ///
                         enrl_1oct_`chrt'_yr4_`type' == 1) if !mi(chrt_`chrt')
     }
}
```

Save your file as a temporary file to merge it with the Student_Collegegoing file you saved in Step 8.

```
tempfile nsc
save `nsc'
```

STEP 9: National Student Clearinghouse Data

SOLUTIONS

9.1

sid	chrt_ninth	chrt_grad	hs_diploma_date	first_enrl_date_any	enrl_ever_w2_grad	ontime_yr	ontime_date	enrl_ever_w2_ninth
15647	2005	2008	14-May-08	30-Aug-08	1	2008	1-Sep-08	1
15656	2005	2007	9-May-07	17-Jan-10	0	2008	1-Sep-08	1
15658	2006	2009	17-May-09		0	2009	1-Sep-09	0

9.2

sid	chrt_grad	hs_diploma_date	first_enrl_date_any	n_enrl_begin_date	enrl_1oct_grad_yr1	enrl_1oct_grad_yr2	enrl_1oct_grad_yr3	enrl_1oct_grad_yr4
16011	2008	14-May-08	23-Aug-08	23-Aug-08	1	.	.	.
16011	2008	14-May-08	23-Aug-08	17-Jan-09	.	1	.	.
16011	2008	14-May-08	23-Aug-08	22-Aug-09	.	1	.	.
16011	2008	14-May-08	23-Aug-08	16-Jan-10	.	.	1	.
16016	2009	14-May-08	23-Aug-09	22-Aug-09	1	.	.	.
16016	2009	14-May-08	23-Aug-09	16-Jan-10	.	1	.	.

9.3

Original data:

sid	type	enrl_ever_w2_ninth	enrl_1oct_ninth_yr1	enrl_1oct_ninth_yr2	enrl_1oct_ninth_yr3	enrl_1oct_ninth_yr4
41	_2yr	1	1	.	.	.
41	_2yr	0	.	1	.	.
41	_2yr	1	.	1	.	.
41	_4yr	1	.	.	1	.

STEP 9: National Student Clearinghouse Data

After collapse:

sid	type	enrl_ever _w2_ninth	enrl_1oct _ninth_yr1	enrl_1oct _ninth_yr2	enrl_1oct _ninth_yr3	enrl_1oct _ninth_yr4
41	_2yr	1	1	1	.	.
41	_2yr	1	.	.	1	.

After reshape:

sid	enrl_ever _w2_ ninth _2yr	enrl_1oct _ninth_ yr1 _2yr	enrl_1oct _ninth_ yr2 _2yr	enrl_1oct _ninth_ yr3 _2yr	enrl_1oct _ninth_ yr4 _2yr	enrl_ever _w2_ninth _4yr	enrl_1oct _ninth_ yr1 _4yr	enrl_1oct _ninth_ yr2 _4yr	enrl_1oct _ninth_ yr3 _4yr	enrl_1oct _ninth_yr4 _4yr
41	1	1	1	.	.	1	.	.	1	.

STEP 10: Merge the Collegegoing and NSC file

You're Almost There!

You are almost finished creating the college-going analysis file. All that is left is to merge the Student_CollegeGoing file with the NSC college going indicators you created in Step 9 and drop any students classified as `transferout` from the analysis. This allows your analysis file to contain student characteristics, high school outcomes, and college enrollment and persistence outcomes. This is a 1:1 merge on student id and results in observations that match, as well as observations that come only from the Student_CollegeGoing file.

```
use "${analysis}/Student_CollegeGoing.dta", clear
merge 1:1 sid using `nsc', keep(1 3)
```

Ensure that your file is unique by student id, drop any students classified as `transferout`, and save.

```
isid sid
drop _m

drop if transferout == 1

compress
save "${analysis}/CG_Analysis.dta", replace
```

Keep both sets of records from this merge since the Student_CollegeGoing file captures the full analytic sample of students upon which your analyses rely. Note that the merge results in missing values for college enrollment and persistence outcomes because some students in the data do not match to NSC enrollment records. This occurs if a student did not enroll in college and is therefore not included in the NSC files. To fix this issue, ensure all college enrollment and persistence outcomes are binary by setting missing values to 0. At this point, label all variables with clear label definitions (in the event you forget how you defined variables inthe analysis file). Finally, save the final analysis file. Save the file as CG_Analysis.

This file can be used for most of the college going analyses and graphs. We provide a separate instruction and code file to create indicators relative to students being on track.

STEP 10: Examining the Analysis File Part 2

Creating the college-going analysis file is an accomplishment. This undertaking integrates disparate administrative data with student-level post-secondary records, and creates a dataset that allows education agencies to explore, identify and address critical barriers to student transitions through high school and college. We hope this guide facilitates the investment of time and energy required to convert data into a resource that drives evidence based decision-making and advances student achievment. Admire your handiwork. Here is the final list of variables you will have in your analysis file, which includes the NSC variables from the last step.

CG_Analysis

1	sid	15	first_hs_code	24	ontime_grad	30	test_math_8_raw	41-43	first_college_opeid_*
2	male	16	first_hs_name	25	late_grad	31	test_math_8	44-46	first_college_name_*
3	race_ethnicity	17	last_hs_code	26	stil_enrl	32	test_math_8_std	47-49	enrl_1oct_grad_yr1_*
4	hs_diploma	18	last_hs_name	27	transferout	33	test_ela_8_raw	50-52	enrl_1oct_grad_yr2_*
5	hs_diploma_type	19	longest_hs_code	28	dropout	34	test_ela_8	53-55	enrl_1oct_grad_yr3_*
6	hs_diploma_date	20	longest_hs_name	29	disappear	35	test_ela_8_std	56-58	enrl_1oct_grad_yr4_*
7	frpl_ever	21	last_wd_group			36	test_composite_8	59-61	enrl_ever_w2_grad_*
8	iep_ever	22	chrt_ninth			37	test_composite_8_std	62-64	enrl_grad_persist_*
9	ell_ever	23	chrt_grad			38	qrt_8_math	65-67	enrl_grad_all4_*
10	gifted_ever					39	qrt_8_ela	68-70	enrl_1oct_ninth_yr1_*
11	frpl_ever_hs					40	qrt_8_composite	71-73	enrl_1oct_ninth_yr2_*
12	iep_ever_hs							74-76	enrl_1oct_ninth_yr3_*
13	ell_ever_hs							77-79	enrl_1oct_ninth_yr4_*
14	gifted_ever_hs							80-82	enrl_ever_w2_ninth_*
								83-85	enrl_grad_all4_*
								86-88	enrl_ninth_all4_*

Student Attributes + Student School Year

First, last, and longest hs; ninth grade and graduation cohorts

End of high school outcomes for high school graduates and non-graduates

Prior Achievement

College Enrollment and Persistence Outcomes

* consists of a 2-yr, 4-yr, and any college type version of the variable

Connect: On Track Indicators

Purpose: Generate variables that indicate students' on track status at the end of each high school year.

Files needed: Student_School_Year_Ninth, Student_School_Enrollment_Clean and Student_Class_Enrollment_Merged from Clean, and Student_CollegeGoing from Step 8 in Connect.

After this step the variables created will be merged onto the CG_Analysis file to create a CG_Analysis_Ontrack file.

The process of creating the on track indicators consists of three major steps:

1. Create the sample used for the on-track analyses
2. Create the on-track variables
3. Create GPA test variables.

Before you begin, specify the current school year. For our example, we assume that we are in 2010.

```
global current_schyr = "2010"
```

Step 1: Create the On-Track Sample

1.1 Keep the students who have all records to be in the sample

The on-track indicator file is based on a subset of the college going file. In order to be included, a student has to be enrolled in the district in the first semester of 9th grade, and has to have continuous enrollment.

If a student ever transferred out, they could have earned credits in another district that we don't have records us. Thus, we have to drop them from the sample. We use the Student School Year file and the Student School Enrollment file to determine enrollment.

```
use "${clean}/Student_School_Year_Clean_Ninth", clear
merge 1:m sid school_year using "${clean}/Student_School_Enrollment_Clean", keep(1 3) nogen
```

Create a new withdrawal-code-based binary variable that identifies transfer-out codes.

This variable will be 1 for all withdrawal codes related with transfer-out, not just last withdrawal code observed for the student.

```
// Create an indicator for students ever observed with a transfer out code.
gen ever_transferout_temp = 0

replace ever_transferout_temp = 1 if withdrawal_code_desc == "Home School" | ///
                    withdrawal_code_desc == "Left District" | ///
                    withdrawal_code_desc == "Other Transfer" | ///
                    withdrawal_code_desc == "Transfer Out of District" | ///
                    withdrawal_code_desc == "Death"
```

Connect: On Track Indicators

```
bys sid: egen ever_transferout = max(ever_transferout_temp)

label define tr 0 "Never transfer-out" 1 "Ever transfer-out" , modify
label values ever_transferout tr
label var ever_transferout "Ever transfer-out"
drop ever_transferout_temp
```

Now the ever_transferout variable is consistent by student.

Omit students who ever transfer out of the district since we can't determine their total credit accumulation in any year in which they left the district.

```
drop if ever_transferout == 1
drop ever_transferout
```

Keep the relevant variables, and ensure that your file is unique by student id and school year. Save the file for use later.

```
keep sid school_year grade_level
duplicates drop

isid sid school_year
tempfile student_school_year
save `student_school_year', replace
```

Next, load the Student Class Enrollment file, and merge with the college going analysis file.

```
use "${clean}/Student_Class_Enrollment_Merged" , clear
merge m:1 sid using "${analysis}/Student_CollegeGoing"
```

We can only assess if a student is on track if we have course information for them. Keep only records that appear in both files.

```
keep if _m==3
drop _m
```

Merge the file with the Student School Year fiel you saved above, keeping students who appear in both files.

```
merge m:1 sid school_year using `student_school_year', keep(3) nogen
```

Keep only students who were in the district in first semester of 9th grade, to ensure that we have complete record for them.

```
bys sid: egen enrolled_grade9 = max(grade_level == 9 & (marking_period == "S1" | marking_period ==
"Q1" | marking_period == "YL"))
keep if enrolled_grade9 == 1
```

Connect: On Track Indicators

Restrict to cohorts that have had time to graduate. We assume here that you have complete records until the school year before the current year.

```
keep if chrt_ninth <= $current_schyr - 4
```

Identify students who don't enroll subsequently from one year to the next and omit those from our sample. To do this, create an variable that indicates if a student is enrolled in one year, not enrolled in the next, and then enrolled again.

```
preserve

    keep sid school_year
    duplicates drop
    sort sid school_year
    gen temp_nonlin = 1 if sid[_n] == sid[_n-1] & school_year[_n] != (school_year[_n-1] + 1)
    egen nonlin_enrl = max(temp_nonlin), by(sid)
    drop temp
    tab nonlin_enrl if sid!= sid[_n-1], m

    tempfile nonlin
    save `nonlin', replace

restore
```

Merge this indicator onto your current file, and drop all students who have any episodes of non-linear enrollment.

```
merge m:1 sid school_year using `nonlin', assert(2 3) keep(3) nogen
drop if nonlin_enrl == 1
drop nonlin_enrl
```

1.2. Resolve inconsistencies with credit variables

Now, you can move on to resolving inconsistencies with the credit variables.

In Clean, you ensured that there are no missing credits possible and credits earned. Confirm this here.

```
assert !mi(credits_possible)
assert !mi(credits_earned)

tab credits_possible credits_earned, m
```

credits_po	credits_earned						
ssible	0	.25	.5	1	1.5	2	Total
0	1,063	4	155	25	0	0	1,248
.25	58	664	14	0	0	0	736
.5	15,767	11	250,851	87	7	1	266,725
.58	0	0	1	0	0	0	1
1	133	0	14	1,814	0	0	1,961
1.5	11	0	0	2	151	0	164
2	0	0	0	0	0	2	2
3	1	0	0	0	0	0	3
Total	17,033	679	251,035	1,928	158	3	270,840

Connect: On Track Indicators

credits_possible	credits_earned 3	3.5	4	Total
0	0	1	0	1,248
.25	0	0	0	736
.5	0	0	1	266,725
.58	0	0	0	1
1	0	0	0	1,961
1.5	0	0	0	164
2	0	0	0	2
3	2	0	0	3
Total	2	1	1	270,840

In our data, there are a handful have 2 or 3 credits_possible. This may be due to block scheduling.

Some students with 0 credits_possible have received credits_earned > 0

Typically a course has 0 credits possible if it does not count towards GPA. In most cases students receive a "P" for these courses.

```
tab final_grade_mark if credits_possible == 0 & credits_earned != 0
tab final_grade_mark credits_earned if credits_possible == 0 & credits_earned != 0
```

final_grade_mark	credits_earned .25	.5	1	3.5	Total
A	2	11	0	0	13
A+	0	1	0	0	1
A-	0	1	0	0	1
B	0	1	2	0	3
B+	0	0	1	0	1
B-	0	1	0	0	1
C	0	4	1	0	5
C+	0	1	0	0	1
C-	0	1	0	0	1
D-	0	2	0	0	2
P	2	132	21	1	156
Total	4	155	25	1	185

However, notice that in some cases a student has a normal final grade mark, for these courses. We will change the credits possible to credits earned for these.

To do this, we are looping through the possible final grade mark letters, and replacing credits possible with credits earned if the final grade mark contains any of those letters.

```
foreach gl in A B C D E {
    replace credits_possible = credits_earned if credits_possible == 0 & credits_earned != 0 &
regexm(final_grade_mark, "`gl'") & final_grade_mark != "NGPA"
}
```

Note, that our data contains a final grade mark of “NGPA”. These are courses that do not earn GPA credits. In the code above, we had to exclude "NGPA" specifically, because it contains the letter "A", and as such, it would be replaced in the loop for “A”.

Connect: On Track Indicators

Review final grade marks and credits earned.

```
tab final_grade_mark credits_earned
```

Make sure that final grade marks of F and non credit-carrying letter grades (in our case, NGPA) have 0 credits earned.

```
replace credits_earned = 0 if final_grade_mark == "F" | final_grade_mark == "NGPA"
```

For the most part, students with credits_earned == 0 receive an "F" letter grade.

However, some students receive passing grades and should receive credits_earned. Review and fix these cases.

```
tab final_grade_mark if credits_earned == 0
tab final_grade_mark credits_possible if credits_earned == 0
```

final_grad	credits_possible						
e_mark	0	.25	.5	1	1.5	3	Total
A	184	0	69	0	0	0	253
A+	27	0	17	0	0	0	44
A-	11	0	2	0	0	0	13
B	17	0	17	1	0	0	35
B+	2	0	6	0	0	0	8
B-	0	0	6	0	0	0	6
C	2	0	19	0	0	0	21
C+	0	0	3	0	0	0	3
C-	0	0	10	0	0	0	10
D	1	0	30	0	0	0	31
D+	0	0	3	0	0	0	3
D-	1	0	25	0	0	0	26
F	8	39	13,807	68	9	1	13,932
NGPA	384	17	1,340	29	4	0	1,774
P	426	2	608	37	0	0	1,073
Total	1,063	58	15,962	135	13	1	17,232

Assign these students non-zero credits earned = credits_possible.

```
replace credits_earned = credits_possible if credits_earned == 0 & credits_possible != 0 &
regexm(final_grade_mark, "A|B|C|D|E") & final_grade_mark != "NGPA"
```

If credits_possible in that observation is also zero, then replace with mode of course.

Calculate modal credits_earned for each course (cid is unique by school_year school_code section_code course_code)

```
bys cid: egen credits_earned_mode = mode(credits_earned)
replace credits_earned = credits_earned_mode if credits_earned == 0 & credits_earned_mode!=. &
regexm(final_grade_mark, "A|B|C|D|E") & final_grade_mark != "NGPA"
```

Connect: On Track Indicators

Finally, if still 0 for credits_earned and credits_possible, leave as is.

Look at the tabulation and make sure that there are not many cases like this; Otherwise, work with your agency to determine possible causes. In our example, we have 245 cases.

```
tab final_grade_mark credits_possible if credits_earned == 0 & final_grade_mark != "F" & final_grade_
mark != "NGPA"  & final_grade_mark != "P"
```

```
           | credits_
 final_grad | possible
    e_mark |         0 |     Total
-----------+-----------+----------
         A |       184 |       184
        A+ |        27 |        27
        A- |        11 |        11
         B |        17 |        17
        B+ |         2 |         2
         C |         2 |         2
         D |         1 |         1
        D- |         1 |         1
-----------+-----------+----------
     Total |       245 |       245
```

Next, resolve instances in which credits_earned > credits_possible.

```
tab credits_possible credits_earned if credits_earned > credits_possible & final_grade_mark != "P"
& final_grade_mark != "NGPA"
```

Set credits_possible to credits_earned if credits_possible is zero and credits_earned is non-zero.

```
replace credits_possible = credits_earned if credits_possible==0 & credits_earned~=0 & credits_
earned > credits_possible & regexm(final_grade_mark, "A|B|C|D|E") & final_grade_mark != "NGPA"
```

Now set remaining credits_earned AND credits_possible to equal the mode credits earned computed above.

```
gen replace_credits = (credits_possible > 0 & ///
        credits_earned > credits_possible & ///
        regexm(final_grade_mark, "A|B|C|D|E") & final_grade_mark != "NGPA")

replace credits_earned   = credits_earned_mode if replace_credits == 1 & !mi(credits_earned_mode)

replace credits_possible = credits_earned_mode if replace_credits == 1 & !mi(credits_earned_mode)
```

Review remaining mismatched credits_possible and credits_earned. These all have missing modes. We are dropping them at this point.

```
tab credits_possible credits_earned if replace_credits==1 & credits_earned != credits_possible
```

```
credits_po |          credits_earned
    ssible |        .5          1        1.5 |     Total
-----------+---------------------------------+----------
       .25 |         4          0          0 |         4
        .5 |         0         10          3 |        13
-----------+---------------------------------+----------
     Total |         4         10          3 |        17
```

```
drop if credits_earned > credits_possible & regexm(final_grade_mark, "A|B|C|D|E") & final_grade_mark
!= "NGPA"
drop replace_credits
```

Look at the results by credit and by each final grade mark.

```
tab credits_possible credits_earned, m

levelsof final_grade_mark, local(mark)
foreach m of local mark {
    di "Final mark:  `m' "
    tab credits_possible credits_earned if final_grade_mark == "`m'", m
}
```

Make sure that credits_possible and credits_earned are not missing.

```
assert !mi(credits_possible)
assert !mi(credits_earned)
```

1.3. Create variable indicating years in HS

At this point, the only thing left in creating the sample file is to keep only one observation per student/school_year, create a variable indicating how many years the student appears in the high school data, and merge back onto main file.

```
preserve

    keep sid school_year grade_level
    duplicates drop
    isid sid school_year
    sort sid school_year grade_level
    bys sid: gen year_in_hs = _n

    tempfile num_yrs
    save `num_yrs', replace

restore

merge m:1 sid school_year using `num_yrs', assert(3) nogen keepusing(year_in_hs)

tab chrt_ninth chrt_grad  if sid!=sid[_n-1], m

compress
save "${analysis}/Student_OnTrack_Sample.dta", replace
```

Step 2: Create On-Track Variables

In this step you will calculate credits earned in each of the first four high school years, generate on-track indicators and generate outcome indicators at the end of each year in high school.

Connect: On Track Indicators

To start, open the file you saved in Step 1 above.

```
use "${analysis}/Student_OnTrack_Sample.dta", clear
```

2.1 Calculate credits earned in each of a student's first four school years in high school.

First, calculate total credits earned.

```
foreach yr of numlist 1/4 {
    egen temp_credits_earned_yr`yr' = total(credits_earned) if year_in_hs <= `yr', by(sid)
    egen cum_credits_yr`yr' = max(temp_credits_earned_yr`yr'), by(sid)
    assert !mi(cum_credits_yr`yr')
    drop temp*
}
```

Then, calculate total credits earned in Math and English Language Arts. Replace with 0 if student didn't earn credit.

```
foreach yr of numlist 1/4 {
    foreach s in ela math {
      egen temp_earned_yr`yr'_`s' = total(credits_earned) if year_in_hs <= `yr' & `s'_flag == 1,
by(sid)
      egen cum_credits_yr`yr'_`s' = max(temp_earned_yr`yr'_`s'), by(sid)
      replace cum_credits_yr`yr'_`s' = 0 if mi(cum_credits_yr`yr'_`s')
      drop temp_earned_yr`yr'_`s'*
    }
}
```

2.2 Generate on track indicators

In our sample district, 23 credits are needed to graduate from high school with regular diploma, including 4 credits in ELA, 3 in Math.

In addition, promotion to next grade guidelines are as follows:

Promotion from grade to grade should be based on credits earned:

- Promotion to 10th grade – 5 credits
- Promotion to 11th grade – 11 credits
- Promotion to 12th grade - 17 credits

Using this information, define on-track indicators by year enrolled in HS, of graduating within 4 years of initial high school enrollment.

Connect: On Track Indicators

DEFINITION:

- on track by end of 9th: 5 total credits, 1 math, 1 English
- on track by end of 10th: 10 total credits, 1 math, 2 English
- on track by end of 11th: 15 total credits, 2 math, 3 English
- on track by end of 12th: 20 total credits, 3 math, 4 English

```
gen temp_ontrack_endyr1 = (cum_credits_yr1 >= 5  & cum_credits_yr1_math >= 1 & cum_credits_yr1_ela
>= 1) if year_in_hs == 1
gen temp_ontrack_endyr2 = (cum_credits_yr2 >= 10 & cum_credits_yr2_math >= 1 & cum_credits_yr2_ela
>= 2) if year_in_hs == 2
gen temp_ontrack_endyr3 = (cum_credits_yr3 >= 15 & cum_credits_yr3_math >= 2 & cum_credits_yr3_ela
>= 3) if year_in_hs == 3
gen temp_ontrack_endyr4 = (cum_credits_yr4 >= 20 & cum_credits_yr4_math >= 3 & cum_credits_yr4_ela
>= 4) if year_in_hs == 4

// Label on-track variables created above
label define ontrack 0 "Off-Track" 1 "On-Track", replace
```

Now, populate the variables consistently by student.

```
foreach y in 1 2 3 4 {
    bys sid: egen ontrack_endyr`y' = max(temp_ontrack_endyr`y')
    drop temp_ontrack_endyr`y'
    label values ontrack_endyr`y' ontrack
}
```

Keep only relevant variables, and make sure you only have one observation per student/yr observed in high school

```
keep sid school_year hs_diploma* last_wd_group chrt_* ///
    ontime_grad late_grad still_enrl transferout dropout disappear year_in_hs cum_credits_*
ontrack_*

duplicates drop

isid sid school_year
isid sid year_in_hs
```

2.3 Generate outcome indicators at the end of each year in high school.

Create labels that you can apply to the variable created for each high school year.

```
label define status 1 "Enrolled, On-Track" 2 "Enrolled, Off-Track" 3 "Dropped Out" 4 "Disappeared",
replace
```

2.3a Loop through the first 3 years of high school. Determine if the student is on track or off track, or has dropped out, disappeared or earned a General Education Diploma. Make sure that these values are populated consistently for each student.

Connect: On Track Indicators

```
foreach y in 1 2 3 {
    gen      temp_status_after_yr`y' = .
    replace temp_status_after_yr`y' = 1 if    ontrack_endyr`y' == 1
    replace temp_status_after_yr`y' = 2 if    ontrack_endyr`y' == 0
    replace temp_status_after_yr`y' = 3 if mi(ontrack_endyr`y') & dropout == 1
    replace temp_status_after_yr`y' = 4 if mi(ontrack_endyr`y') & disappear == 1

    assert !mi(temp_status_after_yr`y') if year_in_hs == `y'
    egen test = nvals(temp_status_after_yr`y'), by(sid)
    assert test == 1 | mi(test)

    egen status_after_yr`y' = max(temp_status_after_yr`y'), by(sid)

    drop test temp*
    label values status_after_yr`y' status
}
```

Students who graduated in less than 4 years are considered on track. Replace status_after_yr vars to enrolled_ on-track for those students who graduated in less than 4 years.

```
foreach y in 1 2 3 {
    local yp1 = `y' + 1
    replace status_after_yr`y' = 1 if ontime_grad == 1 & mi(ontrack_endyr`yp1') & !mi(chrt_grad)
}
```

2.3b Now, define status after 4th year using diploma information.

```
label define status_yr4 1 "Graduated On-Time" 2 "Enrolled, Not Graduated" 3 "Dropped Out" 4
"Disappeared", replace

tab chrt_ninth late_grad, mi

gen temp_status_after_yr4 = .
replace temp_status_after_yr4 = 1 if ontime_grad == 1 & !mi(chrt_grad)
replace temp_status_after_yr4 = 2 if still_enrl==1 | late_grad==1
replace temp_status_after_yr4 = 3 if mi(hs_diploma_date) & dropout==1
replace temp_status_after_yr4 = 4 if mi(hs_diploma_date) & mi(temp_status_after_yr4) &
disappear==1

assert !mi(temp_status_after_yr4) if year_in_hs == 4, rc0
egen test = nvals(temp_status_after_yr`y'), by(sid)
assert test == 1 | mi(test)

egen status_after_yr4 = max(temp_status_after_yr4), by(sid)

drop test temp*
label values status_after_yr4 status_yr4
```

Label your variables.

```
label var status_after_yr1 "Status After First Year"
label var status_after_yr2 "Status After Second Year"
label var status_after_yr3 "Status After Third Year"
label var status_after_yr4 "Status After Fourth Year"
```

Connect: On Track Indicators

Keep time-invariant variables and ensure that your file is unique by student id. Then, save your on-track variables file.

```
drop year_in_hs school_year

duplicates drop
isid sid

gen ontrack_hsgrad_sample = 1

compress
save "${analysis}/Student_OnTrack_Variables.dta", replace
```

Step 3: Generate GPA and Test variables

In this step you will calculate GPA and credits earned in each year of high school, add SAT and ACT score variables, and identify if a student is considered highly qualified.

For this step you will start with the Sample file you saved in Step 1.

```
use "${analysis}/Student_OnTrack_Sample.dta", clear
```

3.1 Process final grades and credits

First, ensure final_grade_mark and final_grade_mark_num comport. In the tabulation you should see a pattern that makes sense. In the sample data we provided you will notice that in addition to the regular grades (A through F), we have "P", representing Pass/Fail courses, and "NGPA", representing other non-credit bearing courses.

```
tab final_grade_mark_num final_grade_mark , m
```

For "P" and "NGPA" set credits_possible and numeric grade mark to 0 to exclude from GPA calculation.

```
replace credits_possible   = 0 if inlist(final_grade_mark, "P", "NGPA")
replace final_grade_mark_num      = 0 if inlist(final_grade_mark, "P", "NGPA")
```

Calculate GPA points by multiplying the numeric final grade marks by credits possible. This allows you to weigh higher credit courses in the GPA calculation.

```
gen  gpa_points = final_grade_mark_num * credits_possible
```

Ensure that credits attempted field is populated for everyone with GPA points.

```
assert !mi(credits_possible) if !mi(gpa_points)
assert gpa_points==0 if inlist(final_grade_mark, "P", "NGPA")
```

Connect: On Track Indicators

Determine GPA earned by each student every year. Total yearly GPA is calculated by dividing the total GPA points by total credits possible for the courses the student enrolled in.

```
bys sid school_year: egen tot_gpa_points_yr = sum(gpa_points)
bys sid school_year: egen tot_gpa_credits_yr = sum(credits_possible)
gen gpa_year = tot_gpa_points/tot_gpa_credits
```

Now you have created GPA and credit variables by student and school year. Keep these variables, and ensure your file is unique by student id and school year.

```
keep tot_gpa* gpa_year sid school_year
duplicates drop
drop if gpa_year==.
isid sid school_year
```

Calculate a cumulative GPA. Start by adding GPA points and credits across years until the given year.

```
sort sid school_year

bys sid: gen num = _n
tab num

foreach var in points credits {
    gen total_`var' = tot_gpa_`var' if num == 1
    replace total_`var' = tot_gpa_`var'_yr + total_`var'[_n-1] if sid == sid[_n-1]
    assert total_`var' >= total_`var'[_n-1] if sid == sid[_n-1]
}
```

Then, generate a cumulative GPA by dividing the cumulative GPA points by the cumulative credits.

```
gen cum_gpa_yr = total_points/total_credits
sum cum_gpa_yr

assert cum_gpa_yr == gpa_year if num ==1
drop num total_points tot_gpa_credits_yr tot_gpa_points_yr
```

Identify final cumulative GPA in high school.

```
gsort sid -school_year
bys sid: gen last_yr = _n == 1
gen temp_cum_gpa_final = cum_gpa_yr if last_yr == 1
egen cum_gpa_final = max(temp_cum_gpa_final), by(sid)
assert !mi(cum_gpa_final) if !mi(cum_gpa_yr)

label var cum_gpa_yr "Cum GPA by year"
label var cum_gpa_final "Cum GPA at end of high school"
```

Reshape data to have one record per student, and variables for each high school year.

```
sort sid school_year
bys sid: gen j = _n
drop if j>4
keep sid cum_gpa_yr cum_gpa_final j
reshape wide cum_gpa_yr, i(sid cum_gpa_final) j(j)
```

Connect: On Track Indicators

Now, merge these GPA variables and the Student_OnTrack_Variables file you created in Step 2. Order the variables and save the resulting file in a temporary file.

```
merge 1:1 sid using "Student_OnTrack_Variables.dta", keep(3) nogen

order cum_gpa_yr* cum_gpa_final, last

tempfile cg_student
save `cg_student'
```

3.2 Add SAT and ACT Scores

Now you are ready to add the SAT and/or ACT scores you cleaned in Task 5 in Clean. We are going to convert ACT scores to SAT score equivalent, to ensure that we have the indicators on the same scale for each student. For SAT, we are using the Math and ELA portion (excluding Writing)

Start by adding SAT scores to the temporary file you saved before this step, then save the results in the same temporary file.

```
merge 1:1 sid using "${clean}/SAT.dta", keep(1 3)     nogen
save `cg_student', replace
```

Now, add ACT scores and save the results

```
merge 1:1 sid using "${clean}/ACT.dta", keep(1 3)     nogen
save `cg_student', replace
```

Convert ACT scores to SAT scores for all students according to ACT-SAT Concordance table.

```
tab act_composite_score
    gen sat_act_temp = .
    replace sat_act_temp = 400 if act_composite_score < 11 & !mi(act_composite_score)
    replace sat_act_temp = 530 if act_composite_score == 11
    replace sat_act_temp = 590 if act_composite_score == 12
    replace sat_act_temp = 640 if act_composite_score == 13
    replace sat_act_temp = 690 if act_composite_score == 14
    replace sat_act_temp = 740 if act_composite_score == 15
    replace sat_act_temp = 790 if act_composite_score == 16
    replace sat_act_temp = 830 if act_composite_score == 17
    replace sat_act_temp = 870 if act_composite_score == 18
    replace sat_act_temp = 910 if act_composite_score == 19
    replace sat_act_temp = 950 if act_composite_score == 20
    replace sat_act_temp = 990 if act_composite_score == 21
    replace sat_act_temp = 1030 if act_composite_score == 22
    replace sat_act_temp = 1070 if act_composite_score == 23
    replace sat_act_temp = 1110 if act_composite_score == 24
    replace sat_act_temp = 1150 if act_composite_score == 25
    replace sat_act_temp = 1190 if act_composite_score == 26
    replace sat_act_temp = 1220 if act_composite_score == 27
    replace sat_act_temp = 1260 if act_composite_score == 28
    replace sat_act_temp = 1300 if act_composite_score == 29
    replace sat_act_temp = 1340 if act_composite_score == 30
    replace sat_act_temp = 1380 if act_composite_score == 31
    replace sat_act_temp = 1420 if act_composite_score == 32
    replace sat_act_temp = 1460 if act_composite_score == 33
    replace sat_act_temp = 1510 if act_composite_score == 34
    replace sat_act_temp = 1560 if act_composite_score == 35
    replace sat_act_temp = 1600 if act_composite_score == 36
```

Connect: On Track Indicators

Generate SAT concordance.

SAT_ACT_concordance equal to SAT total_score if student has SAT score and does not have ACT score

```
gen sat_act_concordance = sat_total_score      if !mi(sat_total_score) & mi(act_composite_score)
```

SAT_ACT_concordance equal to mean of SAT and ACT score if student has taken both exams

```
egen sat_act_mean = rowmean(sat_total_score sat_act_temp) if !mi(sat_total_score) & !mi(act_
composite_score)
replace sat_act_concordance = sat_act_mean     if !mi(sat_total_score) & !mi(act_composite_score)
```

SAT_ACT_concordance = concordance score calculated above if a student has only ACT score

```
replace sat_act_concordance = sat_act_temp     if mi(sat_total_score) & !mi(act_composite_score)

assert !mi(sat_act_concordance) if (!mi(sat_total_score) | !mi(act_composite_score))
assert mi(sat_act_concordance) if mi(sat_total_score) & mi(act_composite_score)
```

3.3 Identify highly qualified students (eligible to attend flagship university)

We use the SPI definition of highly qualified. Students are considered highly qualified if they

- have a GPA of 3.7 or higher and an SAT score of 1100 or higher, OR
- have a GPA of 3.3 or higher and an SAT score of 1200 or higher, OR
- have a GPA of 3.0 or higher and an SAT score of 1300 or higher

```
gen highly_qualified = .
replace highly_qualified = 1 if !mi(chrt_grad) & cum_gpa_final >= 3.7 & !mi(cum_gpa_final) & sat_act_
concordance >= 1100 & !mi(sat_act_concordance)
replace highly_qualified = 1 if !mi(chrt_grad) & cum_gpa_final >= 3.3 & !mi(cum_gpa_final) & sat_act_
concordance >= 1200 & !mi(sat_act_concordance)
replace highly_qualified = 1 if !mi(chrt_grad) & cum_gpa_final >= 3.0 & !mi(cum_gpa_final) & sat_act_
concordance >= 1300 & !mi(sat_act_concordance)
replace highly_qualified = 0 if !mi(chrt_grad) & mi(highly_qualified)

assert mi(highly_qualified) if mi(chrt_grad)
assert !mi(highly_qualified) if !mi(chrt_grad)

label var highly_qualified "Student is highly qualified and eligible to attend flagship university"
```

Keep only cohorts who have had a chance to graduate.

```
keep if inlist(chrt_ninth, 2005, 2006)
```

Connect: On Track Indicators

Keep the on-track variables.

```
keep sid ontrack_* cum_credits* cum_gpa* status_* sat_act_concordance highly_qualified

compress
gen ontrack_sample = 1

tempfile ontrack
save `ontrack'
```

Merge the on-track variables with the analysis file and save the final college going analysis file that now includes on-track indicators.

```
use "${analysis}/CG_Analysis.dta", clear
merge 1:1 sid using `ontrack', nogen
save "${analysis}/CG_Analysis.dta", replace
```

Your final file will look like the chart below:

CG_Analysis

1	sid	15	first_hs_code	24	ontime_grad	30	test_math_8_raw	41-43	first_college_opeid_*	89-92	cum_credits_yr*
2	male	16	first_hs_name	25	late_grad	31	test_math_8	44-46	first_college_name_*	93-95	cum_credits_yr*_ela
3	race_ethnicity	17	last_hs_code	26	stil_enrl	32	test_math_8_std	47-49	enrl_1oct_grad_yr1_*	97-100	cum_credits_yr_math
4	hs_diploma	18	last_hs_name	27	transferout	33	test_ela_8_raw	50-52	enrl_1oct_grad_yr2_*	101-104	ontrack_endyr*
5	hs_diploma_type	19	longest_hs_code	28	dropout	34	test_ela_8	53-55	enrl_1oct_grad_yr3_*	105-108	status_after_yr*
6	hs_diploma_date	20	longest_hs_name	29	disappear	35	test_ela_8_std	56-58	enrl_1oct_grad_yr4_*	109	ontrack_hsgrad_sample
7	frpl_ever	21	last_wd_group			36	test_composite_8	59-61	enrl_ever_w2_grad_*	110-113	cum_gpa_yr*
8	iep_ever	22	chrt_ninth			37	test_composite_8_std	62-64	enrl_grad_persist_*	114	cum_gpa_final
9	ell_ever	23	chrt_grad			38	qrt_8_math	65-67	enrl_grad_all4_*	115	sat_act_concordance
10	gifted_ever					39	qrt_8_ela	68-70	enrl_1oct_ninth_yr1_*	116	highly_qualified
11	frpl_ever_hs					40	qrt_8_composite	71-73	enrl_1oct_ninth_yr2_*	117	ontrack_sample
12	iep_ever_hs							74-76	enrl_1oct_ninth_yr3_*		
13	ell_ever_hs							77-79	enrl_1oct_ninth_yr4_*		
14	gifted_ever_hs							80-82	enrl_ever_w2_ninth_*		
								83-85	enrl_grad_all4_*		
								86-88	enrl_ninth_all4_*		

Student Attributes + Student School Year

First, last, and longest hs; ninth grade and graduation cohorts

End of high school outcomes for high school graduates and non-graduates

Prior Achievement

College Enrollment and Persistence Outcomes

* consists of a 2-yr, 4-yr, and any college type version of the variable

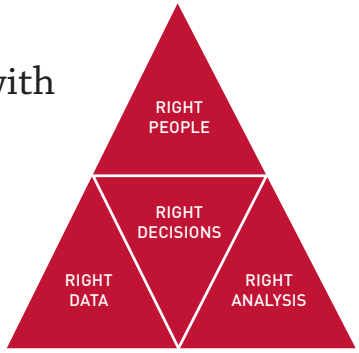
On-Track Outcomes

* consists of a year 1, 2, 3 and 4 version of the variable

The Strategic Data Project

OVERVIEW

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



SDP AT A GLANCE

23 AGENCY PARTNERS
14 SCHOOL DISTRICTS
7 STATE EDUCATION DEPARTMENTS
2 CHARTER SCHOOL ORGANIZATIONS

79 FELLOWS
54 CURRENT
25 ALUMNI

CORE STRATEGIES

1. Placing and supporting top-notch analytic leaders as "Fellows" for two years with our partner agencies

2. Conducting rigorous diagnostic analyses of teacher effectiveness and college-going success using existing agency data

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

SDP DIAGNOSTICS

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

The diagnostics are a set of analyses that frame actionable questions for education leaders. By asking questions such as, "How well do students transition to postsecondary education?" or "How successfully is an agency recruiting effective teachers?" we support education leaders to develop a deep understanding of student achievement in their agency.

ABOUT THE SDP TOOLKIT FOR EFFECTIVE DATA USE

SDP's third core strategy is to disseminate our tools, methods, and lessons learned to many more educational agencies. This toolkit is meant to help analysts in all educational agencies collect data and produce meaningful analyses in the areas of college-going success and teacher effectiveness. Notably, the analyses in this release of our toolkit primarily support questions related to college-going success. The data collection (Identify) and best practices (Adopt) stages of the toolkit, however, are applicable to any sort of diagnostic and convey general data use guidelines valuable to any analysts interested in increasing the quality and rigor of their analyses. Later releases will address analyses relating to teacher effectiveness.

Center for Education Policy Research
HARVARD UNIVERSITY

CENTER FOR EDUCATION POLICY RESEARCH
STRATEGIC DATA PROJECT
50 CHURCH ST., 4TH FLOOR, CAMBRIDGE, MA 02138
VOX 617.496.1563
FAX 617.495.2614
WWW.GSE.HARVARD.EDU/SDP