STRATEGIC DATA PROJECT

CLEAN: DATA BUILDING TASKS FOR HUMAN CAPITAL EDITION 1.0

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 Human Capital BETA

Connect: Data Linking Guide for Human Capital BETA

Analyze: Human Capital Analysis Guide BETA

Adopt: Coding Style Guide

SDP Stata Glossary

VERSION: 1.0
Last Modified: December 18th, 2013

2. Clean: Data Building Guide

Clean and process data files you identify.

Clean: Data Building Tasks for Human Capital is a series of tasks with step-by-step instructions to build clean data files from raw data you identify. In each task, analysts take a raw input file and produce a cleaned output file.

Raw input file ⟶ Task ⟶ Cleaned output file

Each raw input file consists of the data elements required to produce a cleaned output file that will be used in the next step of the toolkit, **Connect**.

DATA BUILDING GUIDE

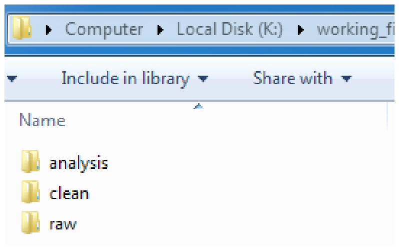
Congratulations on identifying the data to conduct rigorous analyses via **Identify**. **Clean** is the next stage in the **SDP Toolkit**. After this stage, you will have clean data files that allow you to **Connect** and **Analyze** data related to human capital in your agency.

The Tasks

Clean consists of one short task with school data, four tasks with staff data, and five tasks with student data:

1. **School Task 1:** School
2. **Staff Task 1:** Staff Attributes
3. **Staff Task 2:** Staff Certifications
4. **Staff Task 3:** Staff Degrees and Job Codes
5. **Staff Task 4:** Staff School Year
6. **Student Task 1:** Student Attributes
7. **Student Task 2:** Student School Year
8. **Student Task 3:** Student Test Scores
9. **Student Task 4:** Core Courses and Student Class Enrollment

Each task uses a raw input file and produces a cleaned output file. Download these raw input files along with everything else you need for the toolkit as a ZIP file at **www.gse.harvard.edu/sdp/toolkit**. When unzipped, this file will reveal an infrastructure including all the steps of the toolkit, the data files you need, and template files that serve as a shell for Stata code.



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

How to Start

The beginning of the Data Building Guide provides snapshots of all the clean files you will produce through **Clean**. Take a glance at these to understand how the files you'll be producing will be structured.

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

Some of the tasks contain data snippets in the left column of the page. These may help you get a visual sense of the changes occurring at each step.

Task Structure

The tasks follow a logical sequence going from **School Task 1** to **Staff Tasks 1–4** and **Student Tasks 1–4.** Each task comes with its own raw input file and results in a cleaned output file. We also provide all cleaned output files so you can check your answers after completing each task. If you have followed the task instructions correctly, you should arrive at the same cleaned output file.

In each task, you will also find:

- **Purpose:** clarifies the importance of the task,
- **How to start:** identifies the input file for the task,
- **Data description:** describes data elements for the task, and
- **Instructions:** provides instructions to transform data. These instructions include:
  - Stata code to help you execute the instructions through code.
  - Data snapshots to help you visualize changes to the data at each step.

Summary

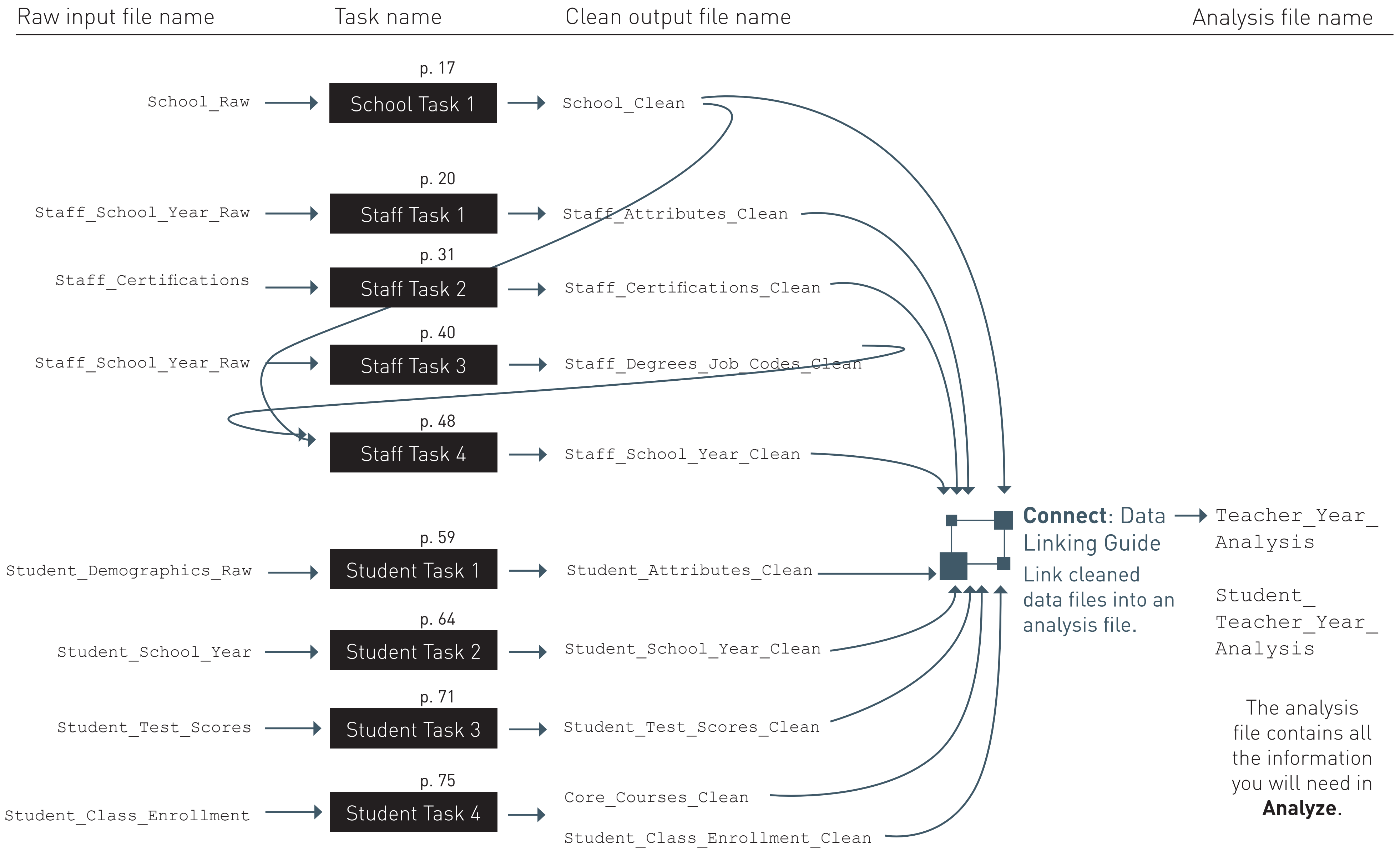
Through the tasks, you will learn effective practices for data transformation, variable construction, and implementation of key decision rules.

The **Task Map** on the next page summarizes the inputs and outputs of each task and shows how the outputs are used in **Connect** to produce an analysis file. The Task Map also serves as a Table of Contents.

If you need additional guidance, the friendly research team at SDP is available to help: **sdp@gse.harvard.edu**.

Task Map

This map summarizes the inputs and outputs for each task, and shows how the outputs are used in **Connect** to produce the college-going analysis and college-going analysis on-track file. The map also serves as a linked Table of Contents.



The analysis file contains all the information you will need in **Analyze**.

CLEAN DATA TABLES

The following tables show the structure and content of the clean data files. These "clean" files are obtained by using the tasks described in the chapters in **Clean**.

These clean files will then be used in **Connect** to create the student- and teacher-level analysis files.

School

Information about schools.

Identifies unique observation: **`school_code`**

Field Name	Values or Data Type	Definition
`school_code`	numeric	Unique school identifier.
`school_name`	string	School name. Some agencies include and assign codes to other locations, such as administrative, central office, juvenile detention center, etc.
`school_lvl`	string	Description of the school level (elementary, middle, high).
`elementary`	0 = Not elementary school 1 = Elementary school	Indicates whether the school is an elementary school.
`middle`	0 = Not middle school 1 = Middle school	Indicates whether the school is a middle school.
`high`	0 = Not high school 1 = High School	Indicates whether the school is a high school.
`alternative`	string	Indicates whether the school is an alternative school.

Staff_Attributes

Time invariant demographic and recruitment data related to staff.

Identifies unique observation: **tid**

Field Name	Values or Data Type	Definition
tid	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
male	0 = female 1 = male	Staff gender.
race_ethnicity	1 = Black 2 = Asian 3 = Latino 4 = Native American 5 = White, not Latino 6 = Other 7 = Multiple	For systems where race and ethnicity are treated as a combined variable. If the system allows multiple categories (e.g., African American and White), report "multiple."
certification_pathway	1 = Standard Certification 2 = Alternative Certification 3 = TFA	The teacher's certification pathway.
birth_date	date format	Staff birth date.

Staff_Certifications

Teaching certifications received by staff.

Identifies unique observation: **`tid + school_year`**

Field Name	Values or Data Type	Definition
`tid`	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
`school_year`	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
`certification_esl`	0 = not certified 1 = certified	Indicates if a teacher has English as a Second Language (ESL) certification in the given school year.
`certification_nbct`	0 = not certified 1 = certified	Indicates if a teacher has National Board certification in the given school year.
`certification_sped`	0 = not certified 1 = certified	Indicates if a teacher has Special Education certification in the given school year.

Staff_Degrees_Job_Codes

Educational achievement for staff. Each degree a staff member has received should be recorded once.

Identifies unique observation:
tid + school_year + school code

Field Name	Values or Data Type	Definition
tid	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
school_code	numeric	Local identifier of schools.
job_code	numeric	Indicates standardized job codes within the agency.
degree	1 = bachelor's degree 2 = master's degree 3 = doctorate degree	Highest degree earned by the teacher.
t_is_teacher	0 = not a teacher 1 = teacher	Indicates if the staff member is a teacher in that school year.
experience	numeric	Years of teaching experience.
hire_date	string, showing a date	Hire date.
termination_date	string, showing a date	Termination date.

Staff_School_Year_Clean

Educational achievement for staff. Each degree a staff member has received should be recorded once.

Identifies unique observation: **`tid + school_year`**

Field Name	Values or Data Type	Definition
`tid`	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
`school_year`	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
`school_code`	numeric	Local identifier of schools.
`job_code`	numeric	Indicates standardized job codes within the agency.
`degree`	1 = bachelor's degree 2 = master's degree 3 = doctorate degree	Highest degree earned by the teacher.
`t_is_teacher`	0 = not a teacher 1 = teacher	Indicates if the staff member is a teacher in that school year.
`experience`	numeric	Years of teaching experience.
`hire_date`	date format	Hire date.
`termination_date`	date format	Termination date.

Student_Attributes

Gender and race/ethnicity data for students.

Identifies unique observation: **sid**

Field Name	Values or Data Type	Definition
sid	numeric	Student identifier unique to each student. This identification number is typically assigned to students upon enrollment in your agency. State agencies may have different identification numbers than district agencies for the same student.
male	0 = female 1 = male	Student gender.
race_ethnicity	1 = Black 2 = Asian 3 = Latino 4 = Native American 5 = White, not Latino 6 = Other 7 = Multiple	Student race and ethnicity. For systems where race and ethnicity are treated as a combined variable.

Student_School_Year

Yearly classification and attendance data for students. Identifies unique observation: **sid** + **school_year**

Field Name	Values or Data Type	Definition
sid	numeric	Student identifier unique to each student. This identification number is typically assigned to students upon enrollment in your agency. State agencies may have different identification numbers than district agencies for the same student.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
grade_level	-9 = ungraded -1 = any pre-kindergarten 0 = kindergarten 1-12 = grades 1–12 13+ = additional grade levels	Student grade level.
retained	0 = not retained 1 = retained	Indicates if the student was retained (is in the same grade level as last year).
frpl	0 = not participating 1 = reduced lunch 2 = free lunch null = no status	Status in the free or reduced price lunch program.
iep	0 = not IEP 1 = IEP null = no status	Indicator for students who have an individualized education plan (IEP).
ell	0 = not ELL 1 = ELL null = no status	Indicator for students who are classified as English Language Learners (ELL). Some systems refer to this category as Limited English Proficient (LEP) or English as a Second Language (ESL).
gifted	0 = not gifted 1 = gifted null = no status	Indicator variable for students enrolled in gifted and talented education programs.
days_enrolled	numeric	Number of days the student was enrolled during the school year.
days_absent	numeric	Number of days the student was absent during the school year.
absence_high	0 = not high absence 1 = high absence	Indicates if the student had a high absence rate (missed more than 10% of days enrolled).
absence_miss	0 = not missing absence data 1 = missing absence data	Indicates if absence data for the student is missing or not available.

Student_Test_Scores

Test data for students.

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

Field Name	Values or Data Type	Definition
sid	numeric	Student identifier unique to each student. This identification number is typically assigned to students upon enrollment in your agency. State agencies may have different identification numbers than district agencies for the same student.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
grade_level	-9 = ungraded -1 = any pre-kindergarten 0 = kindergarten 1-12 = grades 1–12 13+ = additional grade levels	Student grade level.
scaled_score_math	numeric	Math scale score.
scaled_score_ela	numeric	ELA scale score.
scaled_score_composite	numeric	Composite scaled score (average of Math and ELA scores, if the student has both scores).
std_scaled_score_math	numeric	Standardized score in Math. Scores are standardized by school year and grade level, with a mean of 0 and standard deviation of 1.
std_scaled_score_ela	numeric	Standardized score in ELA. Scores are standardized by school year and grade level, with a mean of 0 and standard deviation of 1.
std_scaled_score_composite	numeric	Standardized composite score. Scores are standardized by school year and grade level, with a mean of 0 and standard deviation of 1.

language_version_ela	E = English
	S = Spanish
		Language of the ELA test.

language_version_math	E = English
	S = Spanish
		Language of the MA test.

Core_Courses

Identifies Core Courses

Identifies unique observation: **school_code + cid**

Field Name	Values or Data Type	Definition
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
cid	numeric	Course identifier.
tid	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
school_code	numeric	Local identifier of schools.
math	0 = not math 1 = math	Indicates if the course is a math course.
ela	0 = not ELA 1 = ELA	Indicates if the course is an ELA course.
core	0 = not core 1 = core	Indicates if the course is a core course.
section_code	numeric	Section code for the course.
section_code_desc	string	Description of the section.
course_code	numeric	Course number.

Student_Class_Enrollment

Class enrollment and grades for students.

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

Field Name	Values or Data Type	Definition
sid	numeric	Student identifier unique to each student. This identification number is typically assigned to students upon enrollment in your agency. State agencies may have different identification numbers than district agencies for the same student.
cid	numeric	Course identifier.
final_grade_mark	string	The final grade or mark the student received in the class ("final" means last, cumulative grade assigned). Grades can range from "Alpha Plus" (A+) through F.
final_grade_mark_ numeric	numeric	The final grade or mark the student received in the class ("final" means last, cumulative grade assigned). Grades can range from 0 to 5.

PURPOSE

In **School Task 1** you will take the `School _ Raw` input file and generate a `School _ Clean` file that contains information about each school in your agency.

The core of this task:

1. Ensure that you have one record per school.

2. Create variables to indicate school level.

After this task, you will have a cleaned school file. This file will be used to create your analysis files in **Connect** and later in **Staff Task 4**.

HOW TO START

To begin, open the `School _ Raw` file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

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

DATA DESCRIPTION FOR RAW FILE

To create the `School _ Clean` file, you will use the following variables from the `School _ Raw` data. Note that `School _ Raw` does not exactly match the specifications set in **Identify**, which specifies the school file as being unique by `school _ code` and `school _ year`. Also, the `school _ lvl` variable has not yet been split into separate binary indicators for elementary, middle, and high schools. This action will be performed in this task.

Field Name	Values or Data Type	Definition
school_code	numeric	Unique school identifier.
school_name	string	School name. Some agencies include and assign codes to other locations, such as administrative, central office, juvenile detention center, etc.
school_lvl	string	Description of the school level (elementary, middle, high).
alternative	string	Indicates whether the school is an alternative school.

Uniqueness: The school file should be unique by the `school _ code`, and should contain relevant information about the school.

Unique By:
school_code

School_Raw

Unique By:
school_code

School_Clean
ready for **Connect**

SCHOOL

Step 0: Load the Staff_School_Year data file; keep only necessary variables and drop duplicates.

Process

1. Load the file and drop possible duplicates.

Stata Code

```
use "${raw}/School.dta", clear
duplicates drop
```

Step 1: Clean the file and ensure that the file is unique by school code.

1. Assert that dummy variables, school names, and levels coincide.

```
assert school_lvl == "Elem" if regexm(school_name, "Elementary")
== 1

assert school_lvl == "Mid" if regexm(school_name, "Middle") == 1

assert school_lvl == "High" if regexm(school_name, "High") == 1
```

2. Create dummy variables to indicate the school level.

```
gen elementary = (school_lvl == "Elem")
gen middle = (school_lvl == "Middle")
gen high = (school_lvl == "High")
```

3. Convert the variable indicating if a school is an alternative school into a numeric indicator.

```
tab alternative, mi
gen alt_num =.
replace alt_num = 1 if alternative == "Yes"
replace alt_num = 0 if alternative == "No"

tab alt_num alternative, mi
drop alternative
rename alt_num alternative
```

4. Ensure that the file is unique by school code.

```
isid school_code
```

Step 2: Save the file.

1. Order the variables, sort, and save the current file as `School_Clean.dta`.

```
order school_code school_name school_lvl elementary middle high
alternative
sort school_code
save "${clean}/School_Clean.dta", replace
```

Staff Task 1 STAFF ATTRIBUTES

PURPOSE

In **Staff Task 1** you will take the `Staff_School_Year_Raw` input file and generate a `Staff_Attributes_Clean` file that contains one observation per teacher.

The core of this task:

1. Resolve instances when teachers appear with inconsistent attributes over available years.

2. Ensure that you have one record per teacher.

After this task, you will have the `Staff_Attributes_Clean` file. This file will be used to create your analysis files in **Connect**.

HOW TO START

To begin, open the `Staff_School_Year_Raw` file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

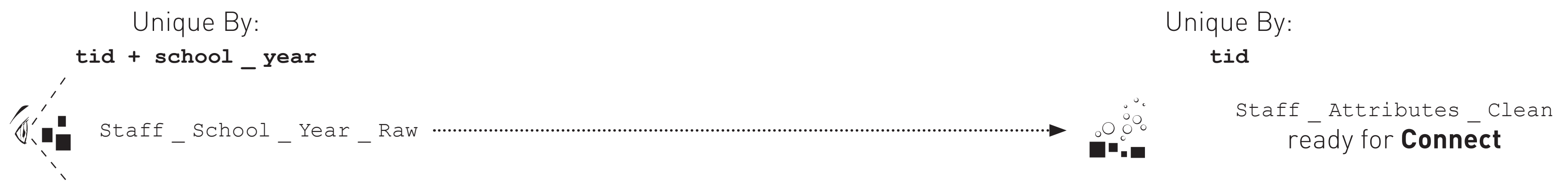
If this is your first time attempting this task, start with the input file we have provided. This file teaches you SDP's cleaning methodology and allows you to check answers from a common dataset.

DATA DESCRIPTION FOR RAW FILE

To create the `Staff_Attributes_Clean` file, you will use the relevant variables from the `Staff_School_Year_Raw` data. Note that `Staff_School_Year_Raw` does not exactly match the specifications set in **Identify**, which specifies `Staff_School_Year` as containing only variables that may change on a year-to-year basis, such as `job_code` or `school_code`, and not other time-invariant characteristics, such as gender or race/ethnicity. `Staff_School_Year_Raw` here contains both of these types of variables, which you will use in **Staff Tasks 1** and **3**. For **Staff Task 1**, you will pull out only the time-invariant characteristics from `Staff_School_Year_Raw`, which are shown below.

Field Name	Values or Data Type	Definition
tid	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
male	0 = female 1 = male	Staff gender.
race_ethnicity	string	For systems where race and ethnicity are treated as a combined variable. If the system allows multiple categories (e.g., African American and White), report "multiple."

Uniqueness: Ideally, data for `Staff_Attributes` are unique by `tid`. However, some agencies may record `race_ethnicity` and/or gender under each school year. Additionally, teachers may have multiple records for a certification pathway and may also have multiple offer, hire, and termination dates. To resolve this issue, you will create a `Staff_Attributes` file unique by tid alone. Once the file is unique by `tid`, it is ready for **Connect**, the next stage of the toolkit.



STAFF ATTRIBUTES

Step 0. Load the Staff_School_Year data file and keep only necessary variables.

Process

1. Load the `Staff _ School _ Year` raw input file.

```
use "${raw}\Staff _ School _ Year _ Raw.dta", clear
```

2. Keep only the variables needed to construct the `Staff_Attributes` file.

```
keep tid school_year male race_ethnicity certification_path
```

3. Drop duplicate records. Because you dropped variables, your remaining data may contain duplicate observations.

```
duplicates drop
```

Stata Code

Step 1: Create one consistent gender value for each student across years.

1. Tabulate the `male` variable to see its values and whether any are missing.

```
tab male, mi

        male |      Freq.     Percent        Cum.
------------+-----------------------------------
     Female |     26,498       49.29       49.29
       Male |     27,002       50.22       99.51
          . |        263        0.49      100.00
------------+-----------------------------------
      Total |     53,763      100.00
```

2. Create a variable that shows how many unique values `male` assumes for each teacher. Name this variable `nvals_male`.

```
bys tid: egen nvals _ male = nvals(male)
```

STAFF ATTRIBUTES

3. Tabulate `nvals_male` to determine the number of instances in which teachers have more than one unique gender reported.

```
tab nvals_male, mi
```

```
 nvals_male |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |     52,927       98.45       98.45
          2 |        573        1.07       99.51
          . |        263        0.49      100.00
------------+-----------------------------------
      Total |     53,763      100.00
```

Note that 690 observations indicate teachers with more than one gender reported. For example, the teacher with `tid` 985 is reported female in 2008 and male in all other years.

tid	school_year	male	nvals_male
985	2004	1	2
985	2005	1	2
985	2006	1	2
985	2007	1	2
985	2008	0	2
985	2009	1	2

4. Generate a variable called `male_mode` that reports the modal gender for each teacher.

```
by tid: egen male_mode = mode(male)
```

5. For teachers who have a mode, replace their gender with `male_mode`.

```
replace male = male_mode if !mi(male_mode)
```

Since the teacher with `tid` 3853 does not have a single modal gender, Stata generates missing values for the `male_mode` variable.

tid	school_year	male	nvals_male	male_mode
3853	2011	0	2	.
3853	2012	1	2	.

6. In instances where there are multiple modes, take the most recent value reported.

Sort the data so that the most recent school year is the first observation for each teacher.

```
gsort tid -school_year
```

STAFF ATTRIBUTES

7. Generate a variable called `temp_male_last` that identifies the gender for the first (most recent) observation for each teacher.

```
bys tid: gen temp_male_last = male if _n==1
```

tid	school_year	male	nvals_ male	male_mode	temp_male_ last
3853	2012	1	2	.	1
3853	2011	0	2	.	.

8. Generate a variable that assigns the highest value of `temp_male_last` (the only non-missing value) to all observations for a teacher. Call this variable `male_last`.

```
egen male_last = max(temp_male_last), by(tid)
```

tid	school_ year	male	nvals_ male	male_ mode	temp_male_ last	male_last
3853	2012	1	2	.	1	1
3853	2011	0	2	.	.	1

9. Replace male with `male_last` if `male_mode` is missing (e.g., a teacher has multiple modes for gender).

```
replace male = male_last if mi(male_mode)
```

tid	school_ year	male	nvals_ male	male_ mode	temp_male_ last	male_ last
3853	2012	1	2	.	1	1
3853	2011	1	2	.	.	1

10. Drop the temporary variables you created.

```
drop nvals_male male_mode temp_male_last male_last
```

Step 2. Create one consistent certification pathway for each teacher across years.

1. Tabulate the `certification_pathway` variable to display its values and identify whether any are missing.

Note that there are several different spellings for similar certification pathways that should be brought together into a common definition.

```
tab certification_pathway, mi

     certification_pathway |      Freq.     Percent        Cum.
---------------------------+-----------------------------------
                       ALT |          2        0.00        0.00
                   ALTCERT |         78        0.15        0.15
ALTERNATIVE CERTIFICATION |      7,937       14.76       14.91
   STANDARD CERTIFICATION |     43,916       81.68       96.60
                  STD CERT |      1,499        2.79       99.38
                       TAF |          2        0.00       99.39
                       TFA |        193        0.36       99.75
                   altcert |          1        0.00       99.75
alternative certification |        24        0.04       99.79
   standard certification |       108        0.20       99.99
                  std cert |          3        0.01      100.00
---------------------------+-----------------------------------
                     Total |     53,763      100.00
```

2. Create consistent values for each `certification_pathway`.

```
replace certification_pathway = "ALTERNATIVE CERTIFICATION" if
inlist(certification_pathway, "ALT", "ALTCERT", "alternative
certification", “altcert”)

replace certification_pathway = "STANDARD CERTIFICATION" if
inlist(certification_pathway, "STD CERT", "standard certification",
"std cert")

replace certification_pathway = "TFA" if inlist(certification_
pathway, "TAF", "tfa")
```

3. Check that there is only one value for each `certification_pathway`.

```
tab certification_pathway


certification_pathway |      Freq.     Percent        Cum.
---------------------------+-----------------------------------
ALTERNATIVE CERTIFICATION |      8,042       14.96       14.96
   STANDARD CERTIFICATION |     45,526       84.68       99.64
                       TFA |        195        0.36      100.00
---------------------------+-----------------------------------
                     Total |     53,763      100.00
```

4. Replace `certification_pathway`'s values with numeric values.

```
replace certification_pathway = "1" if certification_pathway ==
"STANDARD CERTIFICATION"

replace certification_pathway = "2" if certification_pathway ==
"ALTERNATIVE CERTIFICATION"

replace certification_pathway = "3" if certification_pathway ==
"TFA"
```

5. Destring `certification_pathway`.

```
destring certification_pathway, replace
```

6. Label the values of `certification_pathway`.

```
label define cert 1 "Standard Certification" 2 "Alternative
Certification" 3 "TFA"

label values certification_pathway cert
```

7. Create a variable that shows how many unique values `certification_pathway` assumes for each teacher. Name this variable `nvals_cert`.

```
bys tid: egen nvals_cert = nvals(certification_pathway)
```

8. Tabulate `nvals_cert` to determine how many instances there are in the data where teachers have more than one unique `certification_pathway` reported.

```
tab nvals_cert
```

9. Generate a variable called `cert_mode` that reports the modal `certification_pathway` for each teacher.

```
egen cert_mode = mode(certification_pathway), by(tid)
```

10. For teachers who have a mode, replace their `certification_pathway` with `cert_mode`.

```
replace certification_pathway = cert_mode if !mi(cert_mode)
```

This data set does not have any instances where there are multiple modes for teacher certification. As an exercise, fill in the remaining code as if there were multiple modes using Step 1 (creating a consistent gender) as a model.

11. Sort the data so that the most recent school year is the first observation for each teacher.

```
gsort ...
```

12. Generate a variable called `temp_cert_last` that identifies the `certification_pathway` for the first (most recent) observation for each teacher.

```
bys tid: gen ...
```

13. Generate a variable that assigns the highest value of `temp_cert_last` (the only non-missing value) to all observations for a teacher. Call this variable `cert_last`.

```
egen ...
```

14. Replace `certificaton_path` with `cert_last` if `cert_mode` is missing (a teacher has multiple modes for `certification_path`).

```
replace ...
```

15. Drop the temporary variables you created.

```
drop ...
```

Step 3. Create one consistent value for race_ethnicity for each teacher across years. This process is similar to that of gender and certification_ pathway, so data snapshots are not provided in this section.

To be consistent with federal guidelines, teachers who have more than one value for `race_ethnicity` will be reported as multiracial, unless one of the values for their `race_ethnicity` is Latino, in which case they will be reported Latino.

Some districts allow the indication of multiple categories simultaneously, so you will first create consistent `race_ethnicity` values by school year for each teacher based on the guidelines above, and then create a consistent value for each teacher across years.

1. Tabulate the `race_ethnicity` variable to see its values and check if any are missing.

```
tab race_ethnicity, mi
```

2. Create a numeric variable that has consistent values for each race/ethnicity. Use a `for` loop to standardize values for Black and Latino, which have several different spelling variations.

```
gen race_num = .
foreach afam in "AFAM" "AFRICAN AMERICAN" "BLACK" "afam" "african
american" "black" {
        replace race_num = 1 if race_ethnicity=="`afam'"
}
replace race_num = 2 if race_ethnicity=="ASIAN" | race_
ethnicity=="asian"
foreach hisp in "HISP" "HISPANIC" "hisp" "hispanic" {
        replace race_num = 3 if race_ethnicity=="`hisp'"
}
replace race_num = 4 if race_ethnicity=="NATIVE AMERICAN" | race_
ethnicity=="native american"
replace race_num = 5 if race_ethnicity=="WHITE" | race_
ethnicity=="white"
replace race_num = 6 if race_ethnicity=="6"
```

3. Destring the `race_num` variable.

```
destring race_num, replace
```

4. Label the values of `race_num`.

```
label define race 1 "Black" 2 "Asian" 3 "Latino" 4 "Native
American" 5 "White" 6 "Multiple/Other"
label values race_num race
```

5. Check that the values for `race_num` correspond to the values for `race_ethnicity`. Tabulate both variables.

```
tab race_ethnicity race_num, mi
```

6. Replace the string `race_ethnicity` variable with the numeric one.

```
drop race_ethnicity
rename race_num race_ethnicity
```

STAFF ATTRIBUTES

7. Create a variable that shows how many unique values `race _ ethnicity` assumes for each teacher in each school year. Name this variable `nvals _ race _ year`.

```
bys tid school_year: egen nvals_race_year = nvals(race_ethnicity)
```

8. Tabulate `nvals _ race _ year` to determine how many instances there are in the data where `race _ ethnicity` is not consistent within `tid` and `school _ year`.

```
tab nvals_race_year
```

9. If a teacher has more than one race in a single school year, replace the teacher's race as multiracial.

```
replace race_ethnicity = 6 if nvals_race_year>1 & !mi(nvals_race_
year)
```

10. Generate a variable called `temp_islatino` to identify observations where a teacher is reported to be Latino.

```
gen temp_islatino = .
replace temp_islatino = 1 if race_ethnicity == 3
```

11. Generate a variable called `islatino` that indicates the maximum value of `temp_islatino` for all observations within each teacher and year.

```
egen islatino = max(temp_ishispanic), by(tid school_year)
```

Latino teachers should have a value of "1" across all observations and all other teachers should have missing values across all observations for islatino.

```
tab islatino, mi
tab islatino if race_ethnicity==3, mi
tab islatino if race_ethnicity!=3, mi
```

12. Replace `race_ethnicity` with 3 (Latino) if one of the `race_ethnicity` values within the same teacher and year is 3 (Latino).

```
replace race_ethnicity = 3 if nvals_race_year > 1 & !mi(nvals_
race_year) & islatino == 1 & nvals_race > 1
```

Replace `race_ethnicity` with 6 (Multiracial) if a teacher has more than one race in the same year and none of the `race_ethnicity` values are 3 (Latino).

```
replace race_ethnicity = 6 if nvals_race_year > 1 & !mi(nvals_
race_year) & islatino != 1
```

13. Drop the temporary variables you created.

```
drop nvals_race nvals_race_year temp_islatino islatino
```

14. Next, make race_ethnicity consistent by tid.

Check if race_ethnicity is consistent by tid by creating a variable that shows how many unique values race_ethnicity assumes for each teacher. Name this variable nvals_race.

```
bys tid: egen nvals_race = nvals(race_ethnicity)
```

15. Tabulate nvals_race to determine how many instances there are in the data where race_ethnicity is not consistent within tid.

```
tab nvals_race

nvals_race |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |     52,588       98.12       98.12
          2 |        982        1.83       99.95
          3 |         25        0.05      100.00
------------+-----------------------------------
      Total |     53,595      100.00
```

16. Generate a variable called race_mode that indicates the mode for each teacher's race_ethnicity.

```
bys tid: egen race_mode = mode(race_ethnicity)
```

17. For teachers who have a mode, replace their race_ethnicity with race_mode.

```
replace race_ethnicity = race_mode if !mi(race_mode)
```

18. In instances where there are multiple modes, take the most recent value reported.

Sort the data so that the most recent school year is the first observation for each teacher.

```
gsort tid -school_year
```

19. Generate a variable called temp_race_last that identifies the race_ethnicity for the first (most recent) observation for each teacher.

```
bys tid: gen temp_race_last = race_ethnicity if _n==1
```

20. Generate a variable that assigns the highest value of temp_race_last (the only non-missing value) to all observations within a each teacher. Call this variable race_last.

```
egen race_last = max(temp_race_last), by(tid)
```

21. Replace race_ethnicity with race_last if race_mode is missing (a teacher has multiple modes for race_ethnicity).

```
replace race_ethnicity = race_last if mi(race_mode)
```

22. Drop the temporary variables you created.

```
drop nvals_race race_mode temp_race_last race_last
```

Step 4: Create one consistent value for birth_date for each teacher across years.

1. Convert `birth_date` to a numeric variable in Stata date format.

```
gen temp_birth_date = date(birth_date, "MDY")
drop birth_date
rename temp_birth_date birth_date
format birth_date %d
```

2. Check the number of unique birth dates recorded for each teacher.

```
egen nvals_birth_date = nvals(birth_date), by(tid)
```

3. Check the number of missing birth dates.

```
count if mi(birth_date)
```

4. Since each teacher has only one birthday, this variable is clean, so drop the temporary variables you created.

```
drop nvals_birth_date
```

Step 5: Make the data unique by tid, check the data, and save the file.

1. At this point, we have cleaned the data. We no longer need the school year variable.

```
drop school_year
```

2. Drop duplicate observations.

```
duplicates drop
```

3. Check that the file is unique by tid.

```
isid tid
```

4. Check the distribution and range of values for each variable; check for missing data.

```
drop nvals_birth_date
```

Count the number of observations

```
count
```

Produce a detailed summary of `male`, `race_ethnicity`, and `certification_path`, check the number of non-missing observations, range, and distribution.

```
summ male race_ethnicity certification_path, detail
```

5. Order the variables, sort, and save the current file as `Staff_Attributes_Clean.dta`.

```
order tid male race_ethnicity certification_pathway birth_date
sort tid
save "${clean}/Staff_Attributes_Clean.dta", replace
```

STAFF CERTIFICATIONS

PURPOSE

In **Staff Task 2** you will clean the `Staff _ Certifications` file from **Identify** to produce a `Staff _ Certifications _ Clean` file that is unique by `tid` and `school _ year`.

The core of this task:

1. Identify teachers with special certifications, such as English as a Second Language (ESL), Special Education, and National Board Certification.

2. Use teachers' effective certification and effective expiration dates to determine the school years in which teachers became certified and their certifications expired.

3. Ensure that you have one record per teacher-year.

After this task, you will have produced the `Staff _ Certifications _ Clean` file. This will be used to create an analysis file in **Connect**.

HOW TO START

To begin, open the `Staff _ Certifications` file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

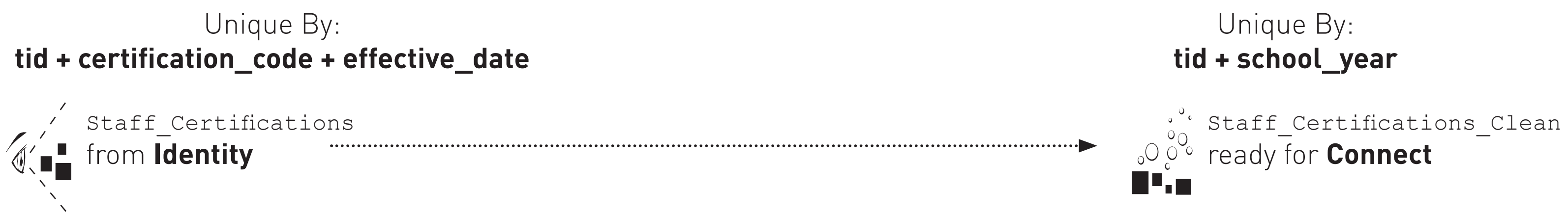
If this is your first time attempting this task, start with the input file we have provided. This file teaches you SDP's cleaning methodology and allows you to check answers from a common dataset.

DATA DESCRIPTION FOR RAW FILE

To create the clean Staff_Certifications file, you will use the relevant variables from the Staff_Certifications data from **Identify**. Here, the input file matches the specification from **Identify**, as the file is unique by tid, certification_code, and effective_date.

Field Name	Values or Data Type	Definition
tid	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff member/teacher.
certification_code	use local codes	One or more variables (e.g., grade level may be separate from subject area). Local code for the type or class of certification or license.
effective_date	string, showing a date	The date the certification took effect.
expiration_date	string, showing a date	The date when certification expired or expires.

Uniqueness: Ideally, data for Staff_Certifications would be unique by tid and school year. However, some agencies record multiple certifications for each teacher in each school year. It is also possible that school year is not included in the file, making it difficult to determine, for example, the school year in which a teacher became National Board certified. To fix these issues, we demonstrate how to create a Staff_Certifications file unique by tid and school year. Once the file is unique by tid and school year, it is ready for **Connect.**



STAFF CERTIFICATIONS

Step 0. Load the Staff_Certifications data file.

Process

1. Load the Staff_Certifications raw input file.

Stata Code

```
use "${raw}\Staff_Certifications_Raw.dta", clear
```

Step 1. Create consistent values for certification_code.

1. String fields entered by hand often contain spelling errors and nonstandard terms. Tabulate `certification_code` to examine the data.

```
tab certification_code, mi

                     certification_code |      Freq.     Percent        Cum.
----------------------------------------+-----------------------------------
                        BOARD CERTIFIED |          2        0.04        0.04
                                    ELS |         22        0.43        0.47
ENGLISH AS A SECOND LANGUAGE CERTIFIC.. |        637       12.37       12.84
                                    ESL |      1,800       34.96       47.80
            NATIONAL BOARD CERTIFICATION |        393        7.63       55.43
                             NATL BOARD |         15        0.29       55.72
                                     NB |        142        2.76       58.48
                             SPECIAL ED |         50        0.97       59.45
         SPECIAL EDUCATION CERTIFICATION |      1,218       23.66       83.10
                                   SPED |        852       16.55       99.65
 english as a second language certific.. |          1        0.02       99.67
                                    esl |          8        0.16       99.83
            national board certification |          3        0.06       99.88
                                     nb |          1        0.02       99.90
         special education certification |          4        0.08       99.98
                                   sped |          1        0.02      100.00
----------------------------------------+-----------------------------------
                                  Total |      5,149      100.00
```

2. There are 16 unique values for `certification_code`, but many appear to indicate the same certifications. Some of these can be combined by changing all values to uppercase.

```
replace certification_code = upper(certification_code)
```

STAFF CERTIFICATIONS

3. Use a `for` loop with the `index` command to replace the remaining nonstandard terms with one consistent value.

The `index` command assigns one value to all values that contain a specified phrase. For example, every value that contains the word "ESL" will be re-assigned the value "ENGLISH AS A SECOND LANGUAGE" in the command at right.

```
foreach esl in "ELS" "ESL" {
       replace certification _ code = "ENGLISH AS A SECOND LANGUAGE
CERTIFICATION" if index(certification _ code, "`esl’")
}

foreach nb in "BOARD" "NATL" "NB" {
       replace certification _ code = "NATIONAL BOARD CERTIFICATION" if
index(certification _ code, "`nb’")
}

foreach sped in "SPECIAL" "SPED" {
       replace certification _ code = "SPECIAL EDUCATION CERTIFICATION"
if index(certification _ code, "`sped’")
}
```

4. Check that there is only one unique value for each certification.

```
tab certification_code

                     certification_code |      Freq.     Percent        Cum.
----------------------------------------+-----------------------------------
ENGLISH AS A SECOND LANGUAGE CERTIFICAT |      2,468       47.93       47.93
           NATIONAL BOARD CERTIFICATION |        556       10.80       58.73
        SPECIAL EDUCATION CERTIFICATION |      2,125       41.27      100.00
----------------------------------------+-----------------------------------
                                  Total |      5,149      100.00

assert certification_code == "ENGLISH AS A SECOND LANGUAGE
CERTIFICATION" | certification_code == "NATIONAL BOARD
CERTIFICATION" | certification_code == "SPECIAL EDUCATION
CERTIFICATION"
```

STAFF CERTIFICATIONS

Step 2. Identify the school years in which a teacher's certification is valid.

1. Reformat effective and expiration dates to Stata date format.

Dates in raw data:

tid	effective_date
1	1/6/2009
4	8/29/2009
4	8/9/2005
5	7/5/2006
8	3/22/2008
16	8/10/2003
19	7/12/2004
23	8/25/2008
24	8/1/2007
27	10/4/2010

After transformation:

tid	effective_date
1	06jan2009
4	29aug2009
4	09aug2005
5	05jul2006
8	22mar2008
16	10aug2003
19	12jul2004
23	25aug2008
24	01aug2007
27	04oct2010

```
foreach var of varlist effective_date expiration_date {
      gen `var'_num = date(`var', "MDY")
      format `var'_num %td
      drop `var'
      rename `var'_num `var'
}
```

STAFF CERTIFICATIONS

2. Identify the school years in which a certificate is valid.

Generate variables that indicate the school year and month a certificate became effective and a certificate expired.

tid	effective_date	effective_date_year	effective_date_month
1	06jan2009	2009	1
4	29aug2009	2009	8
4	09aug2005	2005	8
5	05jul2006	2006	7
8	22mar2008	2008	3
16	10aug2003	2003	8
19	12jul2004	2004	7
23	25aug2008	2008	8
24	01aug2007	2007	8
27	04oct2010	2010	10

```
foreach date in effective_date expiration_date {
      gen `date'_year = year(`date')
      gen `date'_month = month(`date')
}
```

STAFF CERTIFICATIONS

3. A certification is valid during a particular school year if the certification effective date lies between May 1 of the previous school year and April 30 of the current school year. To align the year and month variables just created with the standard of defining a school year by the spring year, add one year to `effective_date_year` if certification effective dates fall between May and December. No changes need to be made for certifications with effective dates between January and April.

A certification that expires before October 1 of the current school year is valid only through the end of the previous school year. No changes need to be made to `expiration_date_year` for certifications that expire between January and September.

A certification that expires between October 1 and April 30 of the current school year will be valid until the end of the current school year. Add one year to `expiration_date_year` if a certification expires between October and December to align with defining a school year by the spring year.

```
replace effective_date_year = effective_date_year + 1 if effective_
date_month>=5 & effective_date_month<=12

replace expiration_date_year = expiration_date_year+1 if expiration_
date_month>=10 & expiration_date_month<=12
```

tid	effective _ date	effective_date_year
1	06jan2009	2009
4	29aug2009	2010
4	09aug2005	2006
5	05jul2006	2007
8	22mar2008	2008
16	10aug2003	2004
19	12jul2004	2005
23	25aug2008	2008
24	01aug2007	2008
27	04oct2010	2011

4. Drop observations that are missing both `effective_date_year` and `expiration_date_year`. These observations pertain to teachers and certification codes with no evidence of an effective date and expiration year.

```
drop if mi(effective_date_year) & mi(expiration_date_year)
```

5. If a teacher has only either an effective date or an expiration date for a certification, the most we know about that teacher's certification is the year in which it was effective or expired.

Assign the same year to both effective and expiration dates if one or the other is missing.

```
replace effective _ date _ year = expiration _ date _ year if effective _
date _ year==. & expiration _ date _ year!=.

replace expiration _ date _ year = effective _ date _ year if expiration _
date _ year==. & effective _ date _ year!=.
```

6. Drop the month variables, as they are no longer necessary.

```
drop *month
```

7. Create an observation row for every combination of teacher, `certification_code`, and `expiration_date_year`. "fillin" creates a variable called `_fillin` that indicates if the row was newly created (`1`) or original to the dataset (`0`).

```
fillin tid certification _ code expiration _ date _ year
```

8. Create variables that identify the effective and expiration years for the observations that are original to the dataset.

```
gen effective_year = effective_date_year if _fillin==0
gen expire_year = expiration_date_year if _fillin==0
```

9. Generate the maximum value for `expire_year` and `effective_year` within teacher and `certification_code`.

```
foreach yr in effective_year expire_year {
     egen max_`yr' = max(`yr'), by(tid certification_code)
}
```

10. Generate a `school_year` variable that is equal to the `expiration_date_year`. Drop observations that have school years before the `max_effective_year` and after the `max_expire_year`.

This should leave one observation for each school year in which a teacher held a valid certificate.

```
gen school_year = expiration_date_year
drop if school_year<max_effective_year
drop if school_year>max_expire_year
```

11. Drop unnecessary variables.

```
drop max* _fillin expire_year effective_year effective_date_year
expiration_date_year effective_date expiration_date
```

STAFF CERTIFICATIONS

Step 3. Create indicators for special certifications.

1. Create temporary variables that indicate if a teacher has a certain certification in a school year.

```
gen temp_esl = (certification_code == "ENGLISH AS A SECOND LANGUAGE
CERTIFICATION")

gen temp_nbct = (certification_code == "NATIONAL BOARD CERTIFICATION")

gen temp_sped = (certification_code == "SPECIAL EDUCATION
CERTIFICATION")
```

2. Populate the existence of a certification in a school year for each teacher/school year observation.

```
foreach cert in esl nbct sped {
      egen certification_`cert' = max(temp_`cert'), by(tid school_year)
}
```

3. Label the certification variables and values.

```
label var certification_esl        "ESL certification"
label var certification_nbct       "National Board certification"
label var certification_sped       "Special Education certification"

label define cert 0 "Not certified" 1 "Certified"

label values certification_esl certification_nbct certification_sped cert
```

4. Drop the unnecessary variables and drop duplicates.

```
drop certification_code temp*
duplicates drop
```

5. Verify that the file is unique by `tid` and `school_year`.

```
isid tid school_year
```

Step 4. Keep relevant data and save file.

1. Limit the range of school years to those in the range of the `Staff_School_Year` file.

First, load the `Staff_School_Year` and assign the first and last school year values to local variables.

Then drop variables that are outside of the range needed.

```
preserve
      use "${raw}/Staff_School_Year_Raw.dta", clear
      sort school_year
            local first_yr = school_year
      gsort -school_year
            local last_yr = school_year
restore

drop if school_year < `first_yr' | school_year > `last_yr'
```

2. Save the current file as `Staff_Certifications_Clean.dta`.

```
save "${clean}\Staff_Certifications_Clean.dta", replace
```

STAFF DEGREES AND JOB CODES

PURPOSE

In **Staff Task 3** you will take the `Staff_School_Year_Raw` file and generate the `Staff_Degrees_Job_Codes_Clean` output file with one unique observation per teacher, school year, and school code. First, you will assign a single degree level for each individual in each school year. Then you will identify a single job code for each individual in each school year.

The core of this task:

1. Resolve instances in which a teacher has multiple degree levels in a single school year.

2. Identify a single job code for each individual within each school year.

After this, you will have a dataset unique by teacher, school year, and school code. This file will be used in **Staff Task 4**.

HOW TO START

To begin, open the `Staff_School_Year_Raw` practice file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

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

DATA DESCRIPTION FOR RAW FILE

To create the `Staff_Degrees_Job_Codes_Clean` file, you will use the relevant variables from the `Staff_School_Year_Raw` data. Note that `Staff_School_Year_Raw` does not exactly match the specifications set in Identify, which specifies `Staff_School_Year` as containing only variables that may change on a year-to-year basis, such as `job_code` or `school_code`, and not other time-invariant characteristics, such as gender or race/ethnicity. `Staff_School_Year_Raw` here contains both of these types of variables, which you used in **Staff Task 1** and will use again now in 3. For **Staff Task 3**, you will pull out only the characteristics that vary from year to year from `Staff_School_Year_Raw`, which are shown below.

Field Name	Values or Data Type	Definition
tid	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
school_code	numeric	Local identifier of schools.
degree	string	Local values of teachers' degrees.
job_code	use local values	Local values that describe jobs.
job_code_desc	string	Description of the job codes.
experience	numeric	Years of teaching experience.
hire_date	string, showing a date	Hire date.
termination_date	string, showing a date	Termination date.

Uniqueness: Ideally, this dataset is unique by `tid` + `school_year` + `school_code`. However, this may not be the case. Some teachers hold multiple job descriptions in the same school year, and/or may appear to have more than one degree. To address this issue, you will take a raw research file and make it unique by `tid`, `school_year`, and `school_code`.

Unique By:
tid + school_year + school_code + degree + job code

Unique By:
tid + school_year + school_code

$$\begin{tikzcd} \text{Staff\_School\_Year\_Raw} \arrow[r, dotted] & \text{Staff\_Degrees\_Job\_Codes\_Clean} \end{tikzcd}$$

ready for **Staff Task 4**

Staff Task 3

STAFF DEGREES AND JOB CODES

Step 0: Load the Staff_School_Year data file, keep only necessary variables and drop duplicates.

Process

Stata Code

1. Load the `Staff _ School _ Year` raw input file.

```
use "${raw}\Staff _ School _ Year.dta", clear
```

2. Keep only the variables needed.

```
keep tid school_year school_code degree job_code job_code_desc
experience hire_date termination_date
```

You will clean `experience`, `hire _ date`, and `termination _ date` in the next task. For now, keep them in the file.

3. Drop duplicate records. Because you dropped variables, your remaining data may contain duplicate observations.

```
duplicates drop
```

Step 1. Assign one unique value for each teacher degree level within each school year.

1. String fields entered by hand often contain spelling errors and nonstandard terms. Tabulate `job _ code _ desc` to examine the data.

Note that about 9.5% of the data are missing values for degree.

```
tab degree, mi

              degree |      Freq.     Percent        Cum.
---------------------+-----------------------------------
                     |      5,963       10.06       10.06
                A.B. |         10        0.02       10.08
                  BA |      4,669        7.88       17.96
        BACH OF ARTS |          3        0.01       17.97
 BACHELOR OF SCIENCE |         74        0.12       18.09
    BACHELORS DEGREE |     23,277       39.29       57.38
    DOCTORATE DEGREE |        725        1.22       58.60
                  MA |        265        0.45       59.05
     MASTER'S DEGREE |        815        1.38       60.43
      MASTERS DEGREE |     23,235       39.22       99.65
                 PHD |         46        0.08       99.72
               Ph.D. |         31        0.05       99.78
                  ba |         12        0.02       99.80
    bachelors degree |         50        0.08       99.88
    doctorate degree |          5        0.01       99.89
     master's degree |          2        0.00       99.89
      masters degree |         64        0.11      100.00
---------------------+-----------------------------------
               Total |     59,246      100.00
```

STAFF DEGREES AND JOB CODES

2. There are many unique values for `degree`, but many appear to indicate the same degree. Some of these can be combined by changing all values to uppercase.

```
replace degree = upper(degree)
```

3. There are many variations of spellings for each degree level. The index command assigns one value to all values that contain a specified phrase. For example, every value that contains the letters "MA" will be reassigned the value "MASTERS DEGREE" in the command.

```
replace degree = "MASTERS DEGREE" if index(degree, "MA")

replace degree = "DOCTORATE DEGREE" if index(degree, "PH")
```

4. There are many variations of `degree` for teachers with bachelor's degrees. Use a `for` loop with the index command to replace the remaining nonstandard terms with one consistent value.

```
foreach ba in "A.B." "BA" "BACH"  {
replace degree = "BACHELORS DEGREE" if index(degree, "`ba'")
}
```

5. Check that there is one unique value for each degree level.

```
tab degree, mi

             degree |      Freq.     Percent        Cum.
--------------------+-----------------------------------
                    |      5,963       10.06       10.06
   BACHELORS DEGREE |     28,095       47.42       57.49
   DOCTORATE DEGREE |        807        1.36       58.85
     MASTERS DEGREE |     24,381       41.15      100.00
--------------------+-----------------------------------
              Total |     59,246      100.00
```

Step 2. Recode the degree variable as a numeric variable and label it. Replace the string degree variable with the numeric one.

1. Generate a variable that assigns a numeric value for each degree level.

```
gen degree_num = .
replace degree_num = 1 if degree=="BACHELORS DEGREE"
replace degree_num = 2 if degree=="MASTERS DEGREE"
replace degree_num = 3 if degree=="DOCTORATE DEGREE"
```

2. Define value labels for `degree _ num` that correspond to the string values in `degree`.

```
label define degree 1 "Bachelor's Degree" 2 "Master's Degree" 3
"Doctorate Degree"

label values degree_num degree
```

3. Check that the values for `degree _ num` match those in `degree`. Then, drop `degree` and rename `degree _ num` to degree.

```
tab degree degree _ num, mi
drop degree
rename degree _ num degree
```

Step 3: Resolve cases in which teachers are missing values for degree or have values less than the degree recorded in a prior year.

STAFF DEGREES AND JOB CODES

1. Some teachers are missing values for degree in some years but not others. Also, some have a value for a degree that is less than the degree they recorded in a prior year.

Use a `for` loop to identify the first and last years in which a teacher held a particular degree and fill in missing or incorrect values.

Replace missing values for degree if no degree is recorded for the last year(s) the teacher is in the data.

Begin with the highest degree value (doctorate) to impute the last observation(s) for each teacher with the highest degree if the last values are missing.

```
foreach degree in 3 2 1 {
       egen temp_last_degree_year`degree' = max(school_year) if
degree==`degree', by(tid)
       egen last_degree_year`degree' = max(temp_last_degree_
year`degree'), by(tid)

       egen temp_first_degree_year`degree' = min(school_year) if
degree==`degree', by(tid)
       egen first_degree_year`degree' = max(temp_first_degree_
year`degree'), by(tid)

       replace degree = `degree' if school_year<=last_degree_
year`degree' & school_year>=first_degree_year`degree'
       replace degree = `degree' if school_year>last_degree_
year`degree' & degree==. & !mi(last_degree_year`degree')

       drop *first_degree* *last_degree*
}
```

Step 4: Create consistent values for job_code_desc.

1. String fields entered by hand often contain spelling errors and nonstandard terms. Tabulate `job _ code _ desc` to examine the data.

```
tab job_code_desc job_code
```

2. There are many unique values for `job _ code _ desc`, but many appear to indicate the same job description. Some of these can be combined by changing all values to uppercase.

```
replace job_code_desc = upper(job_code_desc)
```

3. Some job code descriptions have only two unique values. A replace command will resolve these cases.

```
replace job _ code _ desc = "PRINCIPAL / ASSISTANT PRINCIPAL" if job _
code _ desc=="AP"
```

4. The index command assigns one value to all values that contain a specified phrase. For example, every value that contains the word "COACH" will be reassigned the value "COACH" in the command to the right.

```
replace job _ code _ desc = "COACH" if index(job _ code _ desc, "COACH")
```

5. There are many variations of `job _ code _ desc` for teachers. Use a `for` loop with the index command to replace the remaining nonstandard terms with one consistent value.

```
foreach teach in "TEACHER" "teacher" "tchr" "TCHR" "TAECHER" {
       replace job_code_desc = "TEACHER" if index(job_code_desc,
"`teach'")
}
```

6. Tabulate `job _ code _ desc` to check that each job description has one unique value.

```
tab job _ code _ desc, mi

                   job _ code _ desc |      Freq.     Percent        Cum.
-------------------------------+-----------------------------------
           CLASSROOM ASSISTANT |      2,041        3.44        3.44
                         COACH |        914        1.54        4.99
                     COUNSELOR |        892        1.51        6.49
PRINCIPAL / ASSISTANT PRINCIPAL |      1,035        1.75        8.24
                  SCHOOL STAFF |      1,070        1.81       10.05
   SPECIAL EDUCATION ASSISTANT |        829        1.40       11.45
                    SUBSTITUTE |      3,350        5.65       17.10
                       TEACHER |     46,742       78.89       95.99
                          TEMP |      2,373        4.01      100.00
-------------------------------+-----------------------------------
                         Total |     59,246      100.00
```

Step 5: Create one consistent value for job_code for each teacher and school year.

1. Destring `job _ code`, turning it into a numeric variable.

```
destring job_code, replace
```

2. Tabulate `job _ code` and `job _ code _ desc` to check that each job description is assigned one job code.

```
tab job_code job_code_desc
```

3. Notice that `job _ code` 3 has two job descriptions: "CLASSROOM ASSISTANT" and "SUBSTITUTE". These appear to be different positions. Assign a new, unique job code to classroom assistants.

```
replace job_code = 9 if job_code_desc == "CLASSROOM ASSISTANT"
```

4. Individuals in a school district often serve multiple roles in the same school year. Create a variable that shows how many unique values `job _ code` assumes for each teacher and school year. Name this variable `nvals _ job`.

```
bys tid school_year: egen nvals_job = nvals(job_code)
```

5. Tabulate `nvals _ job` to determine how many instances there are in the data where teachers have more than one unique job code reported in the same school year.

There are 11,568 teacher observations who have more than one unique job code reported in the same school year.

```
tab nvals_job, mi

  nvals_job |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |     49,462       83.49       83.49
          2 |      9,784       16.51      100.00
------------+-----------------------------------
      Total |     59,246      100.00
```

STAFF DEGREES AND JOB CODES

6. Create a variable using `job _ code` that flags an individual as a teacher.

Using parentheses around the "if statement" tells Stata to make a binary variable that assigns a value of 1 to all observations where the statement is true and 0 to all other observations.

```
gen teacher = (job_code == 1)
```

7. Create a variable that indicates whether an individual is a teacher in that school year.

```
bys tid school_year: egen t_is_teacher = max(teacher)
```

8. For individuals who are teachers and have other job descriptions in the same school year, replace their job code with "TEACHER" in all instances.

```
replace job_code_desc = "TEACHER" if t_is_teacher == 1
replace job_code = 1 if t_is_teacher == 1
```

9. For individuals who have multiple job codes in the same school year and are not teachers, keep the observation that is "PRINCIPAL / ASSISTANT PRINCIPAL."

The standard SDP analysis does not differentiate teachers by the jobs they take after they leave teaching if they stay in the district.

A possible extension to the SDP Human Capital Diagnostic is to determine the average teacher effect scores of teachers who become principals.

```
gen principal = 0
replace principal = 1 if job_code == 2
bys tid school_year: egen is_principal = max(principal)
replace job_code_desc = "PRINCIPAL / ASSISTANT PRINCIPAL" if is_
principal == 1 & t_is_teacher == 0
replace job_code = 2 if is_principal == 1 & t_is_teacher == 0
```

10. Some individuals still have more than one job code per school year but are neither teachers nor principals. Resolve these cases using the decision rules below.

First, give priority to permanent roles (drop instances where individuals are temps or substitutes).

Then, drop observations in which the position is non-academic (coaches and school staff).

Give preference to special education assistants—a more specialized position—over classroom assistants.

Finally, give preference to counselors over special education assistants.

Before we remove any observations, check that teacher experience is unique within `tid` and `school _ year`.

Use a `for` loop to accomplish this process efficiently. List the job codes in the order specified above, drop cases in which the role has lower priority for teachers with more than one job in the same school year, and re-identify teachers who still have more than one job code after executing each decision rule.

```
bys tid school_year: egen max_exp = max(experience)
assert max_exp == experience

foreach job in 7 3 4 8 9 6 {
	drop if nvals_job > 1 & job_code==`job'
	drop nvals_job
	bys tid school_year: egen nvals_job = nvals(job_code)
}
```

11. Label the values of `job _ code` and drop `job _ code _ desc`.

```
label define job 1 "Teacher" 2 "Principal / Assistant Principal" 3
"Substitute" 4 "Coach" 5 "Counselor" 6 "Special Education Assistant"
7 "Temp" 8 "School Staff" 9 "Classroom Assistant"

label values job_code job
```

12. Drop unnecessary variables. Keep the `t _ is _ teacher` variable to identify teachers for analysis samples.

```
drop nvals_job teacher principal is_principal
```

Step 6. Make the data unique by tid, school_year, and school_code, check the data, and save the file.

1. Drop duplicate observations.

```
duplicates drop
```

2. Check that the file is unique by `tid`, `school _ year`, and `school _ code`.

```
isid tid school_year school_code
```

3. Save the current file as `Staff _ Degrees _ Job _ Codes _ Clean.dta`.

```
save "{$clean}\Staff_Degrees_Job_Codes_Clean.dta", replace
```

STAFF SCHOOL YEAR

PURPOSE

In **Staff Task 4**, you will take the `Staff _ Degrees _ Job _ Codes _ Clean` file from **Staff Task 3** and the `School _ Clean` file from **School Task 1** and generate a `Staff _ School _ Year _ Clean` file that contains one observation for each school year a teacher is in the data.

The core of this task:

1. Identify one unique school code per teacher within each school year.

2. Resolve inconsistencies in years of teacher experience across school years.

3. Assign one hire and termination date to each employment period.

After this task, you will have the `Staff _ School _ Year _ Clean` file to create an analysis file in **Connect**.

HOW TO START

To begin, open the `Staff _ Degrees _ Job _ Codes _ Clean` file you created in **Staff Task 3** and merge on the school file. If you do not have Stata, you can follow the steps of the task by looking at the instructions and data snippets we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

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

DATA DESCRIPTION FOR RAW FILE

For this task, you will use variables from `Staff_Degrees_Job_Codes_Clean` from **Staff Task 3** and `School_Clean` from **School Task 1**.

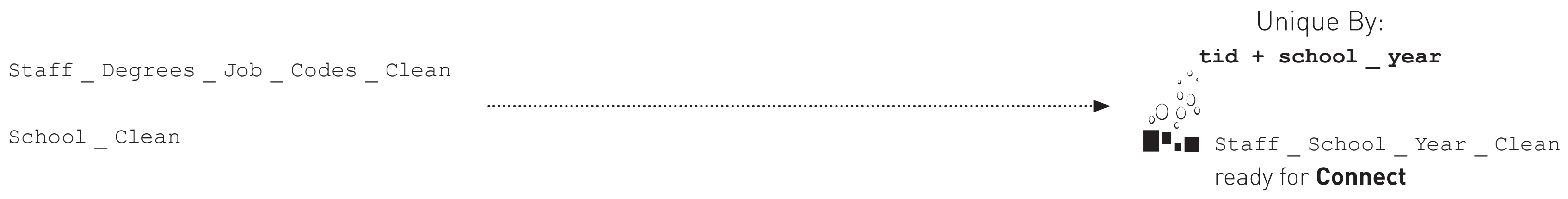
Field Name	Values or Data Type	Definition
tid	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
school_code	numeric	Local identifier of schools.
job_code	numeric	Indicates standardized job codes within the agency.
degree	1 = bachelor's degree 2 = master's degree 3 = doctorate degree	
t_is_teacher	0 = not a teacher 1 = teacher	Indicates if the staff member is a teacher in that school year.
experience	numeric	Years of teaching experience.
hire_date	string, showing a date	Hire date.
termination_date	string, showing a date	Termination date.

Field Name	Values or Data Type	Definition
school_code	numeric	Local identifier of schools.
elementary	0 = not elementary school 1 = elementary school	Indicates whether the school is an elementary school.
middle	0 = not middle school 1 = middle school	Indicates whether the school is a middle school.
high	0 = not high school 1 = high school	Indicates whether the school is a high school.
alternative	0 = not alternative school 1 = alternative school	Indicates whether the school is an alternative school.

STAFF SCHOOL YEAR

Uniqueness: Ideally, data for `Staff _ School _ Year` would be unique by `tid + school _ year`. However, some teachers teach at multiple schools in the same year. One reason it is important to identify a primary school for each teacher is to be able to describe teacher characteristics at the school level without duplicating teachers. Teachers may also have recorded years of experience that are inconsistent (e.g., decrease over time, skip many years). It is especially important to identify which teachers are novice teachers. Many of the analyses SDP conducts describe patterns for novice teachers (e.g., the percentage of novice teachers who leave the district each year, the characteristics of students assigned to novice teachers).

To fix these issues, we demonstrate how to create the `Staff _ School _ Year _ Clean` file unique by `tid` and `school _ year` alone. Once the file is unique by `tid` and `school _ year`, it is ready for **Connect**.


Staff _ Degrees _ Job _ Codes _ Clean
School _ Clean
Unique By:
tid + school _ year
Staff _ School _ Year _ Clean
ready for Connect

STAFF SCHOOL YEAR

Step 0. Load the Staff_Degrees_Job_Codes_Clean data file. Merge school variables from school file.

Process

1. Open `Staff_School_Year_Clean.dta`.

Stata Code

```
use "${raw}\Staff_School_Year_Clean.dta", clear
```

2. Merge school variables from `School_Clean.dta` file. Assert that all observations match.

```
merge m:1 school_code  using “${clean}/School_Clean.dta”,
keepusing(elementary middle high alternative) assert(3) nogen
```

Step 1. Identify one unique school code per teacher within each school year.

1. A school code that is associated with many non-teachers is a sign that the school code might not be an actual school, but some other entity.

Notice that there are many non-teachers associated with school codes 9 and 800.

Tabulate `job_code_desc` for school codes 9 and 800.

Most individuals in school 9 are substitutes and many in school 800 are temporary employees. It is reasonable to assume that a substitute teacher or temporary employee would not earn teaching experience.

This information will be important later when we make decisions about which school code to assign to teachers who have multiple codes in each school year. If one of a teacher’s schools is 9 or 800, we will keep the other one.

```
tab school_code t_is_teacher, mi

foreach school in 9 800 {
      tab job_code_desc if school_code == `school’
}
```

2. Create a variable that shows how many unique values `school_code` assumes for each individual and school year. Name this variable `nvals_school`.

```
bys tid school_year: egen nvals_school = nvals(school_code)
```

3. Tabulate `nvals_school` to determine how many instances there are in the data where individuals have more than one unique school code reported in the same school year.

```
tab nvals_school
```

STAFF SCHOOL YEAR

4. If an individual has more than one school code within the same year and one of them is 9 or 800, drop the observation with `school_code` 9.

```
foreach school in 9 800 {
       drop if nvals _ school > 1 & school _ code==`school' & t _ is _
teacher == 1
}
```

5. If an individual has more than one school code within the same year and one of them is a high school and the other is either a middle school or elementary school, keep the middle or elementary school observation, since SDP does not calculate teacher effects for high school teachers.

First, generate a variable that indicates whether one or both of the school codes within a teacher and school year refer to a middle school or elementary school.

Drop the high school observation for teachers who have more than one school code in the same school year and also teach at a middle or elementary school.

```
gen temp_ms_es = (middle == 1 | elementary == 1)

bys tid school_year: egen ms_es = max(temp_ms_es)

drop if nvals_school > 1 & high == 1 & ms_es == 1
```

6. If an individual has more than one school code within the same year and one of them is a nontraditional school and the other is a traditional school, keep the traditional school observation.

First, determine if teachers teach in non-alternative schools. Generate a variable called `non_alternative` that assigns "`0`" to all observations of teachers within school years if they ever taught at a traditional school.

Drop the alternative school observation for teachers who have more than one school code in the same school year and also teach at a traditional school.

Drop variables that are no longer needed.

```
bys tid school_year: egen non_alternative = min(alternative)

drop if nvals_school > 1 & alternative == 1 & non_alternative == 0

drop high middle elementary *alternative *ms_es
```

7. Since we have resolved some cases where teachers have more than one school code in a single school year, the `nvals_school` variable no longer reflects which teachers have more than one school code per school year. Therefore, we need to create a new variable that identifies how many cases we still need to resolve. Call this new variable `nvals_school2`.

```
bys tid school_year: egen nvals_school2 = nvals(school_code)

tab nvals_school2
```

8. For teachers with more than one school code per year, use the school code from the following (n+1) year. Use a for loop to sequentially identify the maximum and minimum school code and replace its value if it is missing.

First, if one of the school codes matches the school code from the following year for the same teacher (a more recent year), choose that school year.

For teachers who have more than one school code in the same school year, create a separate variable for each school code with values of `school_code` that are constant among all observations within teacher and school year.

Generate a variable whose value is the school code that matches the school code in the following year. The value of this variable will be missing if neither school code matches the value of the school code for the following school year.

Generate a variable that fills in all of the observations within teacher and school year with the value of `next_school_is_min` and `next_school_is_max` for teachers who have more than one school code in the same school year.

Replace school_code with the value of `next_school_code_is_min` or `next_school_code_is_max` if the value is not missing and a teacher has more than one school code in the same school year.

Drop the variables no longer needed.

```
foreach code in max min {
        egen `code'_school = `code'(school_code) if nvals_school2 > 1,
by(tid school_year)

        gen temp_next_school_is_`code' = `code'_school if `code'_school ==
school_code[_n+1] & school_year < school_year[_n+1] & tid == tid[_n+1]

        egen next_school_is_`code' = `code'(temp_next_school_is_`code'),
by(tid school_year)

        replace school_code = next_school_is_`code' if nvals_school2>1 &
next_school_is_`code' !=.

        drop `code'_school temp_next_school_is_`code' next_school_
is_`code'
}
```

9. Drop duplicate observations.

```
duplicates drop
```

10. Again, create a variable that identifies how many teachers still have more than one school code in the same school year.

```
bys tid school_year: egen nvals_school3 = nvals(school_code)

tab nvals_school3
```

STAFF SCHOOL YEAR

11. Repeat the process for Step 9, but instead of selecting the school code that matches the one for the following year, choose the one that matches the school code from the prior year for the same teacher.

```
foreach code in max min {
       egen `code'_school = `code'(school_code) if nvals_school3 > 1,
by(tid school_year)

       gen temp_last_school_is_`code' = `code'_school if `code'_
school==school_code[_n-1] & school_year>school_year[_n-1] & tid ==
tid[_n-1]

       egen last_school_is_`code' = `code'(temp_last_school_is_`code'),
by(tid school_year)

       replace school_code = last_school_is_`code' if
nvals_school3>1 & last_school_is_`code' !=.

       drop `code'_school temp_last_school_is_`code'
last_school_is_`code'
}
```

12. Drop duplicate observations.

```
duplicates drop
```

13. Now the only teachers assigned to more than one school are assigned to two schools that do not repeat in the years following or preceding the year in which the teacher has two schools. For these remaining cases, keep an observation at random.

```
bys tid school_year: egen nvals_school4 = nvals(school_code)
tab nvals_school4
bys tid school_year: gen n = _n
drop if n == 2 & nvals_school4 > 1
```

14. Check that the data file is unique by teacher and school year.

```
isid tid school_year
```

15. Drop unnecessary variables.

```
drop n nvals _ school*
```

16. Use a `for` loop to format both `hire_date` and `termination_date` as a date format.

```
foreach date in hire_date termination_date {
       gen `date'_num = date(`date', "MDY")
       format `date'_num %td
       drop `date'
       rename `date'_num `date'
}
```

17. Drop variables that still need to be cleaned and save this file as a temporary file that we will merge onto the data afterward.

```
drop experience hire_date termination_date
tempfile clean_school
save `clean_school'
```

Step 2. Resolve inconsistencies in years of teacher experience across school years.

1. In cases where non-teachers have years of teaching experience and appear as novice teachers (with one year of experience) in a later year, replace years of experience as missing.

```
sort tid school _ year
replace experience = . if tid==tid[ _ n+1] & t _ is _ teacher==0 & t _ is _
teacher[ _ n+1]==1 & experience!=. & experience!=0 & experience[ _ n+1]==1
```

2. Keep only observations where the individual is a teacher.

```
keep if t_is_teacher == 1
```

3. It is vital to have only one occurrence of the first year of experience teaching. Force experience to equal `2` for all but the earliest instance of `1` for a given teacher.

```
egen min_novice_year = min(school_year) if experience == 1, by(tid)
replace experience = 2 if experience == 1 & school_year != min_novice_year
drop min_novice_year
```

4. Write a command named `drops` (this is called a `program`) to fix drops in experience over time, which is theoretically impossible and likely occurs due to typos in the raw data.

```
program drops

        4a. Flag every instance when a value of experience is less than the
prior value for a given teacher and count the total such instances in the
whole data.
        bys tid (school _ year): gen neg = 1 if experience <
experience[ _ n-1] & _ n != 1 & !mi(experience) & !mi(experience[ _ n-1])
        tab neg
        local neg = r(N)

        4b. If the total is zero (no instances of drops in experience for
any teacher in the data), finish the program.
        if `neg' == 0{
                drop neg
                exit
        }

        4c. If such instances still exist in the data, replace experience
with the prior value of experience, and re-run this process.
        else{
                replace experience = experience[ _ n-1] if neg == 1
                drop neg
                drops
        }

end
```

5. Now that the program has been created, execute it like any other Stata command.

```
drops
```

6. Write a command jumps to fix jumps in experience that are too large given the number of years that have elapsed (for example, a teacher's experience increases by two in one year), and count all such instances in the whole data.

```
program jumps

	6a. Flag every instance when a value of experience has increased
by more than the number of school years that have elapsed for a given
teacher.
	bys tid (school _ year): gen jump = 1 if (school _ year - school _
year[ _ n-1]) < (experience - experience[ _ n-1]) & !mi(experience) &
!mi(experience[ _ n-1])
	tab jump
	local jump = r(N)

	6b. If no such instances exist for any teacher, finish the program.
	if `jump' == 0{
		drop jump*
		exit
	}

	6c. If such instances exist, subtract 1 year of experience in the
latter observation where the jump happens, and re-run this process.
	else{
		replace experience = experience - 1 if jump == 1 &
experience == experience[ _ n+1]
		gen jump2 = jump[ _ n+1]
		replace experience = experience[ _ n+1] - 1 if jump2 == 1
		drop jump*
		jumps
	}

end
```

7. Execute the command written above.

```
jumps
```

8. Note that most teachers are missing experience in 2012. Replace years of experience in 2012 for missing observations, assuming that the teacher gains one year of experience from the prior year.

```
replace experience = (experience[ _ n-1]+1) if tid==tid[ _ n-1] & t _ is _
teacher==1 & school _ year==2012 & experience==.
```

9. Keep only the variables needed.

```
keep tid school _ year school _ code job _ code experience degree t _ is _
teacher hire _ date termination _ date
```

Step 3. Assign one hire and termination date to each employment period.

STAFF SCHOOL YEAR

1. Create a variable to identify each employment period for a teacher.

```
sort tid hire_date termination_date
bys tid hire_date termination_date: gen employment_period =
(_n==1)
replace employment_period = employment_period[_n-1] +
employment_period if tid == tid[_n-1]
```

2. Identify the hire and termination dates for each employment period. Replace hire and termination dates with the mode for the employment period. If there is no mode for the employment period, replace the hire and termination dates with the teacher's overall hire and termination date modes.

```
foreach var of varlist hire_date termination_date {
        bys tid tid_employment_period: egen period_mode_`var’ = mode(`var’)
        replace `var’ = period_mode_`var’ if !mi(period_mode_`var’)
        bys tid : egen total_mode_`var’ = mode(`var’)
        replace `var’ = total_mode_`var’ if mi(period_mode_`var’) &
!mi(total_mode_`var’)
}
```

3. If there is no hire date or termination date mode overall, choose the earliest hire date and the latest termination date.

```
bys tid: egen min_hire_date = min(hire_date)
replace hire_date = min_hire_date if mi(period_mode_hire_date) & mi(total_
mode_hire_date)

bys tid: egen max_termination_date = max(termination_date)
replace termination_date = max_termination_date if mi(period_mode_
termination_date) & mi(total_mode_termination_date)
```

4. Assert that there is only one value for hire and termination dates within an employment period.

```
foreach var of varlist hire_date termination_date {
        bys tid employment_period: egen nvals_`var’ = nvals(`var’)
        assert nvals_`var’ == 1 | nvals_`var’ ==.
        drop nvals_`var’
}
```

Step 4. Merge the temporary file with cleaned school codes to the current data file, check the data, and save the file.

1. Merge data from the temporary file we created earlier. This file contains information about non-teachers.

Note that experience, hire date, and termination date are missing for non-teachers. SDP does not include these variables in any analysis for non-teachers.

```
merge 1:1 tid school_year using `clean_school’, nogen
```

2. Keep only the variables needed.

```
keep tid school_year school_code job_code experience degree
t_is_teacher hire_date termination_date
```

3. Check that data is unique by `tid` and `school_year`.

```
isid tid school_year
```

STAFF SCHOOL YEAR

4. Order the variables, sort the data, and save the data file.	`order  tid school_year school_code job_code degree t_is_teacher experience hire_date termination_date`
	`sort tid school_year`
	`save "${clean}/Staff_School_Year_Clean.dta", replace`

Student Task 1 STUDENT ATTRIBUTES

PURPOSE

In **Student Task 1** you will take the `Student _ Demographics _ Raw` file and generate the `Student _ Attributes _ Clean` file.

The core of this task:

1. Create consistent gender indicators for students across years.

2. Create consistent race/ethnicity values for students across years.

HOW TO START

To begin, open the `Student _ Demographics _ Raw` file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

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

Student Task 1 STUDENT ATTRIBUTES

DATA DESCRIPTION FOR RAW FILE

To create the `Student _ Attributes _ Clean` file, you will use the `Student _ Demographics _ Raw` file. Note that `Student _ Demographics _ Raw` varies from the `Student _ Attributes` file presented in **Identify** in a number of ways. In `Student _ Demographics _ Raw`, `race _ ethnicity` is coded as a string rather than numeric and does not distinguish between the designations multiple, "M," and other, "O." `Student _ Demographics _ Raw` is also a time-variant dataset including `school _ year`, so the data is unique by `sid` and `school _ year`. `Student _ Attributes`, however, is unique by `sid` alone. The aim of this task will be to match `Student _ Attributes` in Identify to be unique by `sid` only.

Field Name	Values or Data Type	Definition
sid	numeric	Student identifier unique to each student. This identification number is typically assigned to a student upon enrollment in your agency. State agencies may have different identification numbers than district agencies for the same student.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
gender	string	Student gender.
race_ethnicity	"B" = Black "A" = Asian "H" = Latino "NA" = Native American "WH" = White, not Latino "M/O"= Multiple/Other	If the system allows the indication of multiple categories simultaneously (e.g., African American and White), report "multiple," unless one of the categories is "Hispanic," in which case Hispanic will be reported. (This rule follows federal standards for reporting race and ethnicity.)

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

Unique By:
sid + school_year

`Student_Demographics_Raw` ···► `Student_Attributes_Clean` ready for **Connect**

Unique By:
sid

Student Task 1 STUDENT ATTRIBUTES

Step 0: Load the Student_Demographics_Raw data file.

Process

1. Load the data into memory.

Stata Code

```
use "${raw}\Student_Demographics_Raw.dta", clear
```

Step 1: Create one consistent value for gender for each student across years.

1. Recode the `gender` variable as a numeric variable and label it.

```
tab gender
gen male = (gender == "Male")

label define s_male 0 "Female" 1 "Male"
label values male s_male

drop gender
```

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

```
bys sid: egen nvals_male = nvals(male)
tab nvals_male
```

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

Define the modal gender. For students who have a mode, replace `male` with the modal value (`male_mode` will be missing if there is no single mode).

```
bys sid: egen male_mode = mode(male)
replace male = male_mode if !mi(male_mode)
```

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

```
gsort sid -school_year
bys sid: gen temp_male_last = male if _n==1
bys sid: egen male_last = max(temp_male_last)
replace male = male_last if mi(male_mode)
```

Drop temporary variables.

```
drop nvals_male male_mode temp_male_last male_last
```

Step 2: Create one consistent value for race_ethnicity for each student across years.

1. Recode the raw `race_ethnicity` variable as a numeric variable and label it. Replace the string `race_ethnicity` variable with the numeric one.

```
tab race_ethnicity, mi

generate race_num=.
replace race_num = 1 if race_ethnicity=="B"
replace race_num = 2 if race_ethnicity =="A"
replace race_num = 3 if race_ethnicity =="H"
replace race_num = 4 if race_ethnicity =="NA"
replace race_num = 5 if race_ethnicity =="W"
replace race_num = 6 if race_ethnicity =="M/O"
replace race_num = 7 if race_ethnicity ==""

label define race 1 "Black" 2 "Asian" 3 "Latino" 4 "Native
American" 5 "White" 6 "Multiple/Other" 7 "Missing"
label val race_num race

tab race_num, mi
drop race_ethnicity
rename race_num race_ethnicity
```

2. Create a variable that shows how many unique values `race_ethnicity` assumes for each student. Name this variable `nvals_race`. Tabulate the variable.

```
bys sid: egen nvals_race = nvals(race_ethnicity)
tab nvals_race
```

3. Adjust `nvals` if a student has a missing race value.

```
egen race_miss = max(race_ethnicity == 7), by(sid)
replace nvals_race = nvals_race - 1 if race_miss == 1 & nvals_race
> 1
```

4. If more than one race is reported, report the student as multiracial, unless one of the reported `race_ethnicity` values is Latino.

Report the student as `islatino` in that case.

If the student has only one race but also missing values, use the race that was reported.

Drop the temporary variables you created.

```
gen temp_islatino = .
replace temp_islatino = 1 if race_ethnicity == 3 & nvals_race > 1
bys sid: egen islatino = max(temp_islatino)

replace race_ethnicity = 3 if nvals_race > 1 & nvals_race !=. &
islatino == 1
replace race_ethnicity = 6 if nvals_race > 1 & nvals_race !=. &
islatino != 1

egen race_nonmissing = min(race_ethnicity) if race_miss == 1 &
nvals_race == 1, by(sid)
replace race_ethnicity = race_nonmissing if !mi(race_nonmissing)

drop temp_islatino islatino nvals_race race_miss race_nonmissing
```

STUDENT ATTRIBUTES

Step 3: Drop any unneeded variables, drop duplicates, check the data, and save the file.

1. Drop `school_year`, as you no longer need it.

```
drop school_year
```

2. Drop duplicate values.

```
duplicates drop
```

3. Check that the file is unique by `sid`.

```
isid sid
```

4. Save the current file as `Student_Attributes_Clean.dta`.

```
save "${clean}\Student_Attributes_Clean.dta", replace
```

Student Task 2 STUDENT SCHOOL YEAR

PURPOSE

In **Student Task 2**, you will take the `Student _ Classifications _ Raw` file and generate a `Student _ School _ Year _ Clean` output file that has one observation per student and school year.

The core of this task:

1. Create one consistent grade level for each student within the same year.
2. Create a variable to indicate if a student was retained in the next school year.
3. Create one consistent free or reduced price lunch (FRPL) value for each student within the same year.
4. Create one consistent individualized education plan (IEP) value for each student within the same year.
5. Create one consistent English language learner (ELL) value for each student within the same year.
6. Create one consistent gifted value for each student within the same year.
7. Tag students who had high absence rates (> =10% of enrolled days) during the school year.

HOW TO START

To begin, open the `Student _ Classifications _ Raw` file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

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

Student Task 2 STUDENT SCHOOL YEAR

DATA DESCRIPTION FOR RAW FILE

The raw input file, `Student _ Classifications _ Raw`, varies from the clean `Student _ School _ Year` file in that it is not unique by `sid` and `school _ year` as shown in **Identify**. For the same `sid` and `school _ year`, a student may have more than one `grade _ level`, `frpl`, `iep`, `ell`, or gifted status reported. The aim of this task will be to match `Student _ School _ Year` in **Identify** in its structure and uniqueness so it is unique by `sid` and `school _ year`.

Field Name	Values or Data Type	Definition
sid	numeric	Student identifier unique to each student. This identification number is typically assigned to a student upon enrollment in your agency. State agencies may have different identification numbers than district agencies for the same student.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
grade_level	-9 = ungraded -1 = any pre-kindergarten 0 = kindergarten 1-12 = grades 1–12 13+ = additional grade levels (e.g., vocational training, special education past year 12)	Grade level of student.
frpl	string	Status in the free or reduced price lunch program.
ell	string	Indicator for students who are classified as English Language Learners (ELL). Some systems refer to this category as Limited English Proficient (LEP) or English as a Second Language (ESL).
iep	string	Indicator for students who have an individualized education plan (IEP).
gifted	string	Indicator variable for students enrolled in gifted and talented education programs.
days_enrolled	number of days	Number of days over the school year a student was enrolled at a school.

Uniqueness: Some students transfer grades mid-year and therefore report two grade levels within a school year. Furthermore, as they transfer, other attributes might change. There may also be duplicates due to clerical errors. To address this issue, you will take the `Student _ Classifications _ Raw` file and make it unique by `sid` and `school _ year`.

NOT Unique By:
sid + school_year

Unique By:
sid + school_year

`Student _ Classifications _ Raw` ....→ `Student _ School _ Year` ready for **Connect**

Student Task 2

STUDENT SCHOOL YEAR

Step 0: Load the Student_School_Year data file.

Process

1. Load the `Student_School_Year_Raw` input file.

Stata Code

```
use "${raw}\Student_School_Year_Raw.dta", clear
```

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

1. Check if there are any instances of multiple grade levels per `sid` per `school_year`.

```
bys sid school_year: egen nvals_grade = nvals(grade_level)
tab nvals_grade
```

2. Keep the highest value per school year.

```
bys sid school_year: egen max_grade_level = max(grade_level)
replace grade_level = max_grade_level
```

3. Drop temporary variables.

```
drop nvals_grade max_grade_level
```

Step 2: Create a variable to indicate if a student was retained in the next school year.

1. Create a separate variable for each school year and populate it with the grade in that school year.

```
levelsof(school_year), local(schyr)

foreach var in `schyr' {
       gen temp_`var' = grade_level if school_year == `var'
       egen grade_yr_`var' = max(temp_`var'), by(sid)
       drop temp
}
```

2. Create a variable to indicate the grade level in the previous school year.

We don't have previous grade info in the first year, so start from the second year.

```
gen grade_level_prevyr =.

local i=1

foreach var in `schyr' {
       local prevyr = `var' - 1
       if `i' > 1 {
              replace grade_level_prevyr = grade_yr_`prevyr' if
school_year == `var'
       }
       local ++i
}
```

3. Indicate if the student was retained.

```
gen retained =.

replace retained = 1 if (grade_level ==  grade_level_prevyr) &
!mi(grade_level_prevyr)

replace retained = 0 if (grade_level !=  grade_level_prevyr) &
!mi(grade_level_prevyr)
```

4. Label the variable.

```
label define yesno10 0 "No" 1 "Yes"
label values retained yesno10
```

5. Drop unnecessary variables.

```
drop grade_yr* grade_level_prevyr
```

Step 3: Create one consistent FRPL value for each student within the same year.

1. Recode raw `frpl` variable with string type to numeric type.

Drop the old string variable and rename the numeric variable.

```
tab frpl

gen frpl_num = .
replace frpl_num = 0 if frpl == "full cost lunch"
replace frpl_num = 1 if frpl == "reduced price lunch"
replace frpl_num = 2 if frpl == "free lunch"

drop frpl
rename frpl_num frpl
```

2. Ensure that `frpl` is consistent by `sid` and `school_year`. In cases where multiple values exist, report the highest value.

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

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

Label the `frpl` values.

Drop the temporary values.

```
bys sid school_year: egen nvals_frpl = nvals(frpl)
tab nvals_frpl

egen highest_frpl = max(frpl), by(sid school_year)
replace frpl = highest_frpl

label define frpl 0 "Not FRPL eligible" 1 "Reduced price lunch
eligible" 2 "Free price lunch eligible"

label values frpl frpl

drop nvals_frpl highest_frpl
```

Step 4: Create one consistent IEP value for each student within the same year.

1. Recode raw `iep` variable with string type to numeric type.

Drop the old string variable and rename the numeric variable.

```
tab iep

gen iep_num = .
replace iep_num = 0 if iep == "not IEP"
replace iep_num = 1 if iep == "IEP"

drop iep
rename iep_num iep
```

2. Ensure that `iep` is consistent by `sid` and `school_year`. In cases where multiple values exist, report the highest value.

Check if there are any cases where different values of `iep` are reported in a year.

```
bys sid school_year: egen nvals_iep = nvals(iep)
tab nvals_iep
```

Report the highest value of `iep` by year for each student, selecting participating over not participating.

```
egen highest_iep = max(iep), by(sid school_year)
replace iep = highest_iep
```

Label the `iep` values.

Drop the temporary values.

```
label define iep 0 "Not IEP" 1 "IEP"
label values iep iep
drop nvals_iep highest_iep
```

Step 5: Create one consistent ELL value for each student within the same year.

1. Recode raw `ell` variable with string type to numeric type.

Drop the old string variable and rename the numeric variable.

```
tab ell, mi

gen ell_num = .
replace ell_num = 0 if ell == "not ELL"
replace ell_num = 1 if ell == "ELL"

drop ell
rename ell_num ell
```

2. Ensure that `ell` is consistent by `sid` and `school_year`. In cases where multiple values exist, report the highest value.

Check if there are any cases where different values of `ell` are reported in a year.

```
bys sid school_year: egen nvals_ell = nvals(ell)
tab nvals_ell
```

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

```
egen highest_ell = max(ell), by(sid school_year)
replace ell = highest_ell
```

Label the `ell` values.

Drop the temporary values.

```
label define ell 0 "Not ELL" 1 "ELL"
label values ell ell
drop nvals_ell highest_ell
```

Step 6: Create one consistent gifted value for each student within the same year.

1. Recode the raw `gifted` variable with string type to numeric type.

Drop the old string variable and rename the numeric variable.

```
tab gifted, mi

gen gifted_num = .
replace gifted_num = 0 if gifted == "not gifted"
replace gifted_num = 1 if gifted == "gifted"

drop gifted
rename gifted_num gifted
```

2. Ensure that `gifted` is consistent by `sid` and `school_year`. In cases where multiple values exist, report the highest value.

Check if there are any cases where different values of gifted status are reported in a year.

Report the highest value of gifted by year for each student, selecting participating over not participating.

Label the gifted values.

Drop the temporary values.

```
bys sid school_year: egen nvals_gifted = nvals(gifted)
tab nvals_gifted

egen highest_gifted = max(gifted), by(sid school_year)
replace gifted = highest_gifted

label define gifted 0 "Not gifted" 1 "gifted"
label values gifted gifted

drop nvals_gifted highest_gifted
```

Step 7: Tag students who had high absence rates (→=10% of enrolled days) during the school year.

1. Create a variable that indicates the 10% cut point above which a student will be marked for high absence.

```
gen abs_cutpoint = floor(days_enrolled * 0.1)
```

2. Tag students who were absent more days than the cutoff.

```
gen absence_high = .
replace absence_high = 1 if days_absent >= abs_cutpoint &
!mi(days_absent)
replace absence_high = 0 if days_absent < abs_cutpoint &
!mi(days_absent)
```

3. Tag observations where the absence data was not available.

```
gen absence_miss = mi(days_absent)
```

4. Drop the temporary variable.

```
drop abs_cutpoint
```

Step 8: Drop any unneeded variables, drop duplicates, and save the file.

1. Drop duplicate observations.

```
duplicates drop
```

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

```
isid sid school_year
```

3. Use a loop to standardize variable names for later merging.

```
foreach var of varlist retained frpl iep ell gifted{
      rename `var' s_`var'
}
```

4. Order, sort, and save the current file as `Student_School_Year_Clean`.

```
order sid school_year grade_level s_retained s_frpl s_iep s_
ell s_gifted days_enrolled days_absent absence_high absence_
miss

sort sid school_year
save "${clean}\Student_School_Year_Clean.dta", replace
```

Student Task 3 STUDENT TEST SCORES

PURPOSE

In **Student Task 3** you will take a `Student _ Test _ Scores _ Raw` file and generate a `Student _ Test _ Scores _ Clean` file unique by student and school year that contains scaled and standardized student achievement test scores for use in calculating teacher value-added estimates.

The core of this task:

1. Ensure that each student has one score per school year and grade level and subject.
2. Generate standardized and composite test scores.

HOW TO START

To begin, open the `Student _ Test _ Scores` practice file. If you do not have Stata, you can follow the steps of the task by looking at the instructions we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

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

Student Task 3 STUDENT TEST SCORES

DATA DESCRIPTION FOR RAW FILE

The input file, `Student _ Test _ Scores _ Raw`, does not match the structure of `Student _ Test _ Scores` in Identify. It is not unique by `sid`, `test _ code` (in this case, the `test _ subject` variable), and `test _ date`. The aim of this task will be to make the file unique by `sid` and `school _ year`.

Variable Name	Values or Data Type	Definition
sid	numeric	Student identifier unique to each student. This identification number is assigned to a student upon enrollment. State agencies may have different identification numbers than district agencies for the same student.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
grade_level	-9 = ungraded -1 = any pre-kindergarten 0 = kindergarten 1-12 = grades 1–12 13+ = additional grade levels (e.g., vocational training, special education past year 12)	Numeric grade level of the test. May be unavailable for SAT, ACT, or AP tests.
test_type	use local values	The category of test, e.g., State Assessment.
test_date	date format (yyyy-mm-dd)	The exact date (or at a minimum school year) the test was completed. Note that students who retake tests or are retained may have multiple observations for a single `test_code`; these should be differentiated by `test_date`.
test_subject	string	Subject of test.

language_version	E = English
	S = Spanish
		Language of test.

scaled_score	numeric	Student's scaled score.

Uniqueness: Ideally, state test data in its raw form is unique by `sid` + `test _ subject` + `school _ year`. However, some students re-take the same test for the same grade in the same school year. To fix this, you will make the raw input file unique by `sid` + `test _ subject` + `school _ year` by selecting the test with the earliest test date or, if both tests are recorded on the same date, selecting the test with the higher score. Then, you will manipulate the data so tests for different subjects fall on the same row to make the data unique by `sid` + `school _ year`.

NOT Unique By:
sid + school_year

Unique By:
sid + school_year

`Student_Test_Scores_Raw` ....► `Student_Test_Scores_Clean`
ready for **Connect**

Student Task 3 STUDENT TEST SCORES

Step 0: Load the raw data file and keep relevant variables and observations.

Process

Stata Code

1. Load the `Student _ Test _ Scores _ Raw` input file.

```
use "${raw}\Student _ Test _ Scores _ Raw.dta", clear
```

2. Keep only the variables you need and limit the sample to state test scores.

```
keep sid test_type test_subject school_year grade_level scaled_
score test_date language_version

keep if test_type == "STATE ASSESSMENT"
```

3. Drop observations with a missing scaled score or a score of zero or a negative score.

```
drop if scaled _ score == . | scaled _ score <= 0
```

Step 1: Ensure that each student has one score per school year and grade level.

Identify same-year repeat test takers and take the earliest score. If more than one on the same date, use the highest.

1. For each student, grade level, school year, and subject, find the earliest date a test was taken.

```
egen earliest_date = min(test_date), by(sid school_year grade_
level test_subject)
```

2. Keep only the test score from the earliest date.

```
keep if test _ date == earliest _ date
drop earliest _ date
```

3. For those with more than one such score on the same date, take the highest score.

```
gsort sid test_subject grade_level school_year -scaled_score

drop if sid==sid[_n-1] & test_subject==test_subject[_n-1] &
school_year==school_year[_n-1] & grade_level==grade_level[_n-1]
```

4. Verify that each student has only one state test in a subject at a given grade level and school year.

```
isid sid test_subject grade_level school_year
```

Step 2: Ensure that each student has one score per school year.

Identify different-year repeat test takes and take the score from the earliest school year.

1. For each student, subject, and grade level, find the earliest school year for which a score exists.

```
egen earliest_year = min(school_year) if !mi(scaled_score), by(sid
grade_level test_subject)
```

2. Keep only the test score from the earliest year.

```
keep if school_year == earliest_year
drop earliest_year
```

3. Verify that each student has only one state test in a subject at a given grade level.

```
isid sid test_subject grade_level
```

Step 3: Reshape the data so math and ELA tests appear on the same row.

1. Reformat the `test_subject` variable for the reshape.

```
replace test_subject = "_math" if test_subject == "MATH"
replace test_subject = "_ela" if test_subject == "ELA"

reshape wide scaled_score test_date language_version, i(sid test_
type school_year grade_level) j(test_subject) string
```

2. Verify that each student has only one state test in each year and drop unneeded variables.

```
isid sid school_year
drop test_date_math test_date_ela test_type
```

Step 4: Generate standardized and composite test scores.

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

```
foreach subject in math ela {
      bys school_year grade_level: center scaled_score_`subject',
standardize gen(std_scaled_score_`subject')
}
```

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

```
gen scaled_score_composite = (scaled_score_math + scaled_score_
ela) / 2 if !mi(scaled_score_math) & !mi(scaled_score_ela)

gen std_scaled_score_composite = (std_scaled_score_math + std_
scaled_score_ela) / 2 if !mi(std_scaled_score_math) & !mi(std_
scaled_score_ela)
```

Step 5: Order the variables, sort the data, and save the file.

1. Order the variables.

```
order sid school_year grade_level scaled_score_math scaled_score_
ela scaled_score_composite std_scaled_score_math std_scaled_score_
ela std_scaled_score_composite
```

2. Sort the data.

```
sort sid school_year
```

3. Save the file.

```
save "${clean}/Student_Test_Scores_Clean.dta", replace
```

STUDENT CLASS ENROLLMENT

PURPOSE

In **Student Task 4** you will take the `Student_Class_Enrollment_Raw` and create a `Core_Courses_Clean` file that contains one observation for each class ID value, as well as indicators for whether that class is a math or ELA class and whether it is a core course. You will also create a `Student_Class_Enrollment_Clean` file that contains each student's grades in each course the student took.

The core of this task:

1. Establish a true one-to-one relationship between course code and course description for all classes.

2. Identify course code/course description pairs that constitute core courses in math and ELA.

3. Identify a single teacher for each course.

HOW TO START

To begin, open the `Student_Class_Enrollment_Raw` file in Stata. If you do not have Stata, you can follow the steps of the task by looking at the instructions we have provided in the left column, separately from the Stata code in the right column. For example, if you are an SPSS user, you should be able to follow the logical instructions and data snippets to see how the data has changed at each step of the data cleaning process.

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

DATA DESCRIPTION FOR RAW FILE

Variable Name	Values or Data Type	Definition
sid	numeric	Student identifier unique to each student. This identification number is assigned to a student upon enrollment. State agencies may have different identification numbers than district agencies for the same student.
school_year	spring calendar year	Academic school year from fall to spring, denoted here as the spring calendar year.
school_code	numeric	Local identifier of schools.
cid	numeric	Unique identifier of courses.
tid	numeric	Unique staff or teacher identifier. State agencies may have different identification numbers than district agencies for the same staff/teacher.
course_code	use local values	Course identifier.
course_code_desc	string	Description of the course (e.g. math, social studies, etc.).
section_code	use local values	Section identifier.
department	string	Description of the department where the course is housed (e.g., ELA, science).
final_grade_mark	string	The final grade or mark the student received in the class ("final" means last, cumulative grade assigned). Grades can range from "Alpha Plus" (A+) through F.
final_grade_mark_numeric	string	The final grade or mark the student received in the class ("final" means last, cumulative grade assigned). Grades can range from 0 to 5.
class_enrollment_date	date format (yyyy-mm-dd)	Date the student was enrolled in the class.

Uniqueness: Ideally, data for student class enrollment would be unique by sid and cid. The purpose of class _ enrollment _ date and class _ withdrawal _ date is to determine whether a student was enrolled in a class for long enough to justify keeping the observation in the dataset. In the practice file that you'll be using, we have performed this step for you, so you can assume that the students were enrolled in their respective classes for long enough. It is not always the case that the data will be unique by sid and cid, however. Classes may consist of multiple sections, and students may be technically enrolled in more than one of these sections, so that a sid/cid pair may appear more than once.

In addition, there may be many misspellings and incorrect values for course description and course code, and oftentimes there is no indicator for whether a class is considered a core course. The cid itself may not be an accurate representation of on-the-ground classes in the agency.

To fix these issues, we demonstrate how to create a core course's research file, unique by cid, that contains math and ELA indicators, a core course indicator, and a single associated teacher with each class. In addition, we demonstrate how to create a clean student class enrollment research file unique by sid and cid that contains students' grades. We also demonstrate how to run the data through some basic checks to ensure its validity.

STUDENT CLASS ENROLLMENT

Step 0: Load the raw Student_Class_Enrollment.dta file.

Process

1. Load the `Student_Class_Enrollment_Raw` file.

Stata Code

```
use "${raw}\Student_Class_Enrollment_Raw.dta", clear
```

Step 1: Fix spelling mistakes and inconsistencies in the course_code and course_code_desc variables.

1. Destring `course_code` and `section_code` for ease of writing `if` conditions sans double quotes in this step and to facilitate modal operations in Step 2.

```
destring course_code, replace
destring section_code, replace
```

2. Combine same-subject departments for ELA and math using the results of the tabulation. Note that the results of the tabulation using your own agency data will be different than those shown here and will require different values to be replaced.

```
tab department

replace department = "ELA" if department == "ENG" | department ==
"ENGL" | department == "LIT"

replace department = "MTH" if department == "MATH"
```

3. Standardize similar `course_code_desc` values using the results of the tabulation.

Note that the results of the tabulation using your own agency data will be different than those shown here and will require different values to be replaced.

```
tab course_code_desc

replace course_code_desc = "FOURTH GRADE" if course_code_desc ==
"4TH GRADE" | course_code_desc == "FORTH GRADE"
replace course_code_desc = "FIFTH GRADE" if course_code_desc ==
"5TH GRADE" | course_code_desc == "GRADE 5"

replace course_code_desc = "ALGEBRA 1" if course_code_desc ==
"AGLEBRA 1"

replace course_code_desc = "LIT & COMP" if course_code_desc ==
"LITCMP"
replace course_code_desc = "ENGLISH 6" if course_code_desc ==
"EGNLISH 6"
replace course_code_desc = "ENGLISH 7" if course_code_desc ==
"EGNLISH7"
```

4. Check the courses labeled "MATH."

```
tab course_code if course_code_desc == "MATH"
tab course_code_desc if course_code == 26
tab course_code_desc if course_code == 27
```

5. Since it's clear that "MATH" should be either "MATH 6" or "MATH 7," change it where possible.

```
replace course_code_desc = "MATH 6" if course_code_desc == "MATH"
& course_code == 26

replace course_code_desc = "MATH 7" if course_code_desc == "MATH"
& course_code == 27
```

6. Check the courses labeled "ENGLISH."

```
tab course_code if course_code_desc == "ENGLISH"
tab course_code_desc if course_code == 11
tab course_code_desc if course_code == 12
```

7. Since it's clear that "ENGLISH" should be either "ENGLISH 6" or "ENGLISH 7," change it where possible.

```
replace course_code_desc = "ENGLISH 6" if course_code_desc ==
"ENGLISH" & course_code == 11

replace course_code_desc = "ENGLISH 7" if course_code_desc ==
"ENGLISH" & course_code == 12
```

Note that not all instances of "MATH" and "ENGLISH" were fixed, and there is no way to know what real course they signify.

```
tab department if course_code_desc == "MATH"
tab department if course_code_desc == "ENGLISH"
```

8. Check that there are no core-course-sounding classes in non-core departments.

```
tab course_code_desc if department != "ELA" & department != "MTH"
& department != "ELEM"
```

Step 2: Merge on grade-level information and browse course descriptions by grade level to identify courses that most students at each grade level are enrolled in.

1. Preserve the data and load the clean Student_School_Year data.

```
preserve
      use ${clean}/Student_School_Year_Clean.dta, clear
```

2. Keep only the merge variables (student ID and school year) and grade level.

```
      keep sid school_year grade_level
```

3. Save as a temp file and restore the data.

```
      tempfile grade_level_data
      save `grade_level_data'

restore
```

4. Merge on the grade-level data by student ID and school year, and ensure that the merge is successful.

```
merge m:1 sid school_year using `grade_level_data', assert(3)
nogen
```

5. Cross-tabulate course description by grade level, restricted to core departments, to determine the courses that most students at each grade level are enrolled in. Note these course descriptions for later.

```
tab course_code_desc grade_level if department == "MTH" |
department == "ELA"
```

For example, in the tab above, it is easy to see that for sixth graders, most of the observations fall under two courses: 40% of observations are for ENGLISH 6 and 42% are for MATH 6. It is reasonable to infer that these are the standard required core math and ELA courses for sixth graders, with the other courses mostly electives, extended pullouts, and advanced courses.

Step 3: Use modal instances within course code to establish 1-to-1 relation between course code and course description, and create indicators for core courses.

Preserve the file.

```
preserve
```

1. Keep only needed variables and non-missing observations.

```
        keep course*
        drop if course_code_desc == "" | course_code == .
```

2. Since we could not accurately label some "ENGLISH" and "MATH" instances, drop them from our true table.

```
        drop if course_code_desc == "MATH" | course_code_desc ==
"ENGLISH"
```

3. Find modal course code for each course description.

```
        egen mode = mode(course_code), by(course_code_desc)
```

4. Drop those observations where the course code does not equal the modal course code for a given course description.

```
        drop if course_code != mode
        drop mode
```

5. Drop duplicate observations.

```
        duplicates drop
```

6. Make sure there is a 1-to-1 relation between course code and course code description.

```
        unique course_code
        local a = r(sum)
        unique course_code_desc
        local b = r(sum)
        assert `a' == `b'
```

7. Create indicators for math and ELA by browsing the list and selecting appropriately.

```
        gen math = inlist(course_code, 2, 3, 13, 16, 26, 27, 28, 37,
51, 52)
        gen ela = inlist(course_code, 10, 11, 12, 13, 16, 22, 25,
40, 47)
```

8. Using the course descriptions identified in Step 2 as core courses, identify core course codes in math and ELA.

```
       gen core = inlist(course_code, 3, 11, 12, 13, 16, 25, 26,
27, 51, 52)
```

Note that you should also use agency knowledge to complete identification of core courses. For example, the earlier tabulations would not indicate that "ALG / GEOM" is a core course because not many eighth graders are enrolled in it. However, it is counted as a core course because it is the only math course for advanced students, who are a relative minority among their peers. This information can be obtained from your agency.

9. Save data in a temporary file for later use.

```
       sort course_code
        tempfile course_catalog
        save `course_catalog'
```

Restore the original file.

```
restore
```

Step 4: Create a class file that contains, for each class ID, a singular teacher ID and indicators for math, ELA, and core courses.

1. Preserve the data and find the modal teacher for each class ID and assign it to each class ID.

Note that it is likely that in your agency, there is not always one teacher per physical classroom (team teaching, for example). In this data, however, there are a very small number of instances where this occurs, likely due to typos in the raw data and perhaps a few specialized classrooms. Thus, the analyses you will perform in the Analyze section use value-added estimates that assume one teacher per classroom. If team teaching is prevalent in your agency, it may be important to consider using weights in the value-added estimation step to properly attribute each teacher's influence on student achievement in such classrooms. This, however, is beyond the scope of this toolkit.

```
preserve
        egen tid_mode = mode(tid), by(cid)
        replace tid = tid_mode
        drop tid_mode
```

2. Investigate the reliability of `course_code` and `course_code_desc` in order to determine which one (or both) to use to create the subject and core indicators.

Note that as there may be typos in the raw data, there may not be a perfect concordance between course code and course description. To extract whether a given class ID is a math and/or ELA course and/or core course, it is necessary to use the more reliable of the available variables that refer to the type of course that class ID is. In cases where there is no obvious difference in reliability, it may be best to use a combination of the variables to assess the most likely "true" status of a given course. In this case, however, it is clear that the combination of course description and department is more reliable than the combination of course code and department. Thus, the indicators will be created based on the course description.

```
        browse course_code if department == "MTH"
        browse course_code_desc if department == "MTH"

        browse course_code if department == "ELA"
        browse course_code_desc if department == "ELA"
```

3. Temporarily rename `course_code` so as to not overwrite it during the merge that follows.

```
        rename course_code raw_course_code
```

4. Using the course description, merge on the indicators from the course catalog constructed in Step 3.9.

```
        merge m:1 course_code_desc using `course_catalog', keep(1 2
        3) nogen
```

5. Drop `course_code` from the course catalog, and rename `raw_course_code` back to its original name.

```
	drop course_code
	rename raw_course_code course_code
```

6. For each class ID, find the modes of the indicators for math, ELA, and core course, and replace each indicator with its corresponding modal value.

```
	foreach indicator in math ela core{
		egen `indicator'_mode = mode(`indicator'), by(cid)
		replace `indicator' = `indicator'_mode
		drop `indicator'_mode
	}
```

Note that a singular course description or course code is not forced on each class ID. The only variables of importance to value-added estimation are the math, ELA, and core course indicators. For example, if there are 20 students in a given class ID, and 19 of them have that class ID marked as an algebra class while the 20th has it marked as photography, the modal values of the indicators will label that class ID as a core math course, even for the student who is supposedly studying photography. It may seem prudent to attribute that discrepancy to a typo and overwrite photography with algebra. However, such overwriting becomes problematic in mixed-level classrooms, when, for example, sixth and seventh graders are studying sixth-and seventh-grade math, respectively, in the same classroom. If the modal course code or description was forced on such a classroom, it may appear that some of the students are either advanced or remedial, when neither is true. However, we can be reasonably sure that in such a classroom, students are pursuing a core math course.

7. Keep only the necessary variables and ensure uniqueness at the class ID level.

```
	keep cid tid school_year school_code math ela core
	duplicates drop
	isid cid
```

8. Save the class ID and indicator data, and restore.

```
	save "${clean}/Core_Courses_Clean.dta", replace
restore
```

Step 5: Save the Student_Class_Enrollment_Clean file, ensuring uniqueness by sid and cid.

1. Keep relevant variables.

```
keep sid cid final_grade_mark final_grade_mark_numeric
```

2. Drop duplicates.

```
duplicates drop
```

3. Find all cases where there is more than one observation for a `sid`/`cid` pair.

```
bys sid cid: gen x = _N
tab x
```

4. Ensure that only one of the observations has grade information.

```
gen grade_info_exists = !mi(final_grade_mark)

egen grade_info_exists_max = max(grade_info_exists) if x >1,
by(sid cid)

egen grade_info_exists_min = min(grade_info_exists) if x >1,
by(sid cid)

assert grade_info_exists_max == 1 & grade_info_exists_min == 0 if
x >1
```

5. For such cases, drop the observations without grade information.

```
drop if mi(final_grade_mark) & x >1
```

6. Clean up the file and save.

```
drop x grade_info*
isid sid cid

save "${clean}/Student_Class_Enrollment_Clean.dta", replace
```

Step 6: Run checks to make sure that the class data is sensible.

1. Merge on the class data.

```
merge m:1 cid using "${clean}/Core_Courses_Clean.dta", keep(1 3)
nogen
```

2. Check that students take a reasonable number of classes in a given year.

```
egen stu_class_per_year = count(cid), by(sid school_year)
tab stu_class_per_year, mi
```

3. Check that students usually have one ELA and one math core course in a given year. First, preserve the data and keep the necessary variables.

```
preserve
	keep sid cid school_year math ela core
```

4. For each subject, count the instances of core math and core ELA courses for each student and year, and fill in for each student.

```
	foreach subj in math ela{
		egen core_`subj'_per_year_temp = count(cid) if `subj'
== 1 & core == 1, by(sid school_year)
		egen core_`subj'_per_year = max(core_`subj'_per_year_
temp), by(sid school_year)
		drop core_`subj'_per_year_temp
	}
```

5. Keep only the necessary variables, drop duplicates, ensure uniqueness by student ID and school year, tab the variables, and then restore the data.

```
	keep sid school_year core_*_per_year
	duplicates drop
	isid sid school_year
	tab core_math_per_year, mi
	tab core_ela_per_year, mi

restore
```

6. Check that class sizes are reasonable.

```
egen class_size = count(sid), by(cid)
summ class_size, detail
```

STUDENT CLASS ENROLLMENT

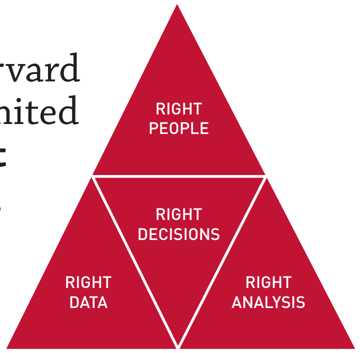
7. Check that teachers teach a reasonable number of classes in a given year.

```
keep tid cid school_year
duplicates drop
egen tch_class_per_year = count(cid), by(tid school_year)
tab tch_class_per_year
```

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 United States. **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

56 AGENCY PARTNERS
34 SCHOOL DISTRICTS
12 STATE EDUCATION DEPARTMENTS
3 CHARTER SCHOOL ORGANIZATIONS
7 NONPROFIT ORGANIZATIONS

107 FELLOWS
65 CURRENT
42 ALUMNI

CORE STRATEGIES

1. Building a network of top-notch data strategists who serve as fellows for two years with our partners

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 the education sector broadly

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 education agencies broadly. This toolkit is meant to help analysts in all education 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.

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