Student Growth Percentiles Calculations

Adam Van Iwaarden
Center for Assessment

Leslie Keng
Center for Assessment

Emma Klugman
Harvard University

November 2022

Abstract

This appendix contains example code used to conduct Phases 1 and 2 of the **"Re-evaluating the Efficiency and Efficacy of Annual Census Standardized Testing for Accountability Purposes"** study. In Phase 1 we prepare and format the data in a manner that will be consistent across all states and satisfies the requirements for growth analyses using the `SGP` software package. With the properly formatted data, we then proceed with the Student Growth Percentiles (SGP) analyses and school level results aggregations in the Phase 2 code.

Keywords: draft – do not cite.

1. Phase 1 - data cleaning and preparation

For this example analysis we will be using the *sgpData_LONG_COVID* data from the SGPData package. It includes 7 years of annual assessment data in two content areas (ELA and Mathematics). As this data is typically used for testing and research purposes with the SGP package (Betebenner et al. 2022), much of the data cleaning and formatting has already been done.

This section of the appendix assumes the user is operating with their working directory set to the state level directory (e.g., "./*State_A*/".

```
# setwd("./State_A")
```

1.1. Load packages and custom functions.

The following R packages are required for the data source, cleaning and augmentation.

```
require(SGPdata)
require(data.table)
```

1.2. General data setup and cleaning

This example dataset comes with a "built-in" impact in 2021 related to the pandemic as well as an unperturbed version - *SCALE_SCORE_without_COVID_IMPACT*. Here we will first subset the data to include only those years needed for the study.

```
# First load and rename/remove SCALE_SCORE* variables included in the data
State_A_Data_LONG <- copy(SGPdata::sgpData_LONG_COVID)[YEAR < 2023]
```

The variables in each longitudinal data set are renamed to be consistent across all states in the study. The variable names ID, YEAR, GRADE, CONTENT_AREA and SCALE_SCORE are required SGP package conventions. The demographic variable names match the "analysis specification" document. Any extraneous variables are removed before saving.

```
setnames(
  State_A_Data_LONG,
  c("ETHNICITY", "FREE_REDUCED_LUNCH_STATUS", "ELL_STATUS", "IEP_STATUS",
    "SCHOOL_NUMBER"),
  c("Race", "EconDis", "EL", "SWD", "SchoolID")
)
State_A_Data_LONG[, Race := as.character(Race)]
State_A_Data_LONG[Race == "African American", Race := "Black"]
# State_A_Data_LONG[Race == "Other", Race := "Multiracial"]

State_A_Data_LONG[, EconDis := gsub("Free Reduced Lunch", "EconDis", EconDis)]
State_A_Data_LONG[, SWD := gsub("IEP", "SWD", SWD)]
State_A_Data_LONG[, EL := gsub("ELL", "EL", EL)]

State_A_Data_LONG[,
  c("SCALE_SCORE_without_COVID_IMPACT", "GENDER",
    "DISTRICT_NUMBER", "DISTRICT_NAME", "SCHOOL_NAME"
  ) := NULL
]
```

We have also decided to make schools unique by the grade levels that they serve (i.e. elementary and middle)

```
State_A_Data_LONG[, SchoolID := as.character(SchoolID)]
State_A_Data_LONG[GRADE %in% 3:5, SchoolID := paste0(SchoolID, "E")]
State_A_Data_LONG[GRADE %in% 6:8, SchoolID := paste0(SchoolID, "M")]
```

1.2.1. Create VALID_CASE and invalidate duplicates

The `SGP` package requires a variable named `VALID_CASE` with values set to either "VALID_CASE" or "INVALID_CASE". This is helpful in identifying cases that are problematic for any number of reasons (duplicate records, invalid ID types, missing scores or subject/grade values, etc.). Any record flagged as an "INVALID_CASE" will be excluded from any SGP analyses (but remain in the data for possible later use in other aggregations).

```
# Create `VALID_CASE` if it does not exist:
State_A_Data_LONG[, VALID_CASE := "VALID_CASE"]

#  Check for record duplicates:
setkey(State_A_Data_LONG,
    VALID_CASE, CONTENT_AREA, GRADE, YEAR, ID, SCALE_SCORE)
setkey(State_A_Data_LONG,
    VALID_CASE, CONTENT_AREA, GRADE, YEAR, ID)
# Total Dups:
State_A_Data_LONG |> duplicated(by = key(State_A_Data_LONG)) |> sum()
#  If duplicates, keep the highest score (if any non-NA exist)
dupl <- duplicated(State_A_Data_LONG, by = key(State_A_Data_LONG))
State_A_Data_LONG[which(dupl)-1, VALID_CASE := "INVALID_CASE"]

#  Subset the data to remove invalid student records
State_A_Data_LONG <- State_A_Data_LONG[VALID_CASE == "VALID_CASE"]
```

1.3. Save data

```
fname <- "Data/Cleaned_Data/Student_LongTestData_State_A_2016-2022_AVI.csv"
fwrite(State_A_Data_LONG, file = fname)
zip(zipfile = paste0(fname, ".zip"), files = fname, flags = "-mqj")
```

1.4. Summary and notes

- "State A" uses the 2016 to 2022 subset of the *sgpData_LONG_COVID* dataset from the `SGPData` package.
    - Only required variables are retained.
- Variables required by the `SGP` packages were renamed and formatted as necessary.
- Student demographic variables were formatted to be consistent across all states in the study.
- A `VALID_CASE` variable was created and used to identify duplicate records and other problematic cases.

2. Student Growth Percentiles Analysis

This section presents and explains the code used to conduct the Student Growth Percentiles (SGP) analyses. Each simulated testing condition is applied, via `R` code configurations, to the same set of data, thus only producing growth measures for students with valid records in the appropriate grades, content areas and years. At the end of each condition-specific analysis, the SGP variable is renamed to indicate the simulated condition before proceeding to the next condition-specific growth analysis step. Only cohort-referenced SGPs are calculated (growth projections and targets are omitted). The goal of this step is simply to create growth percentiles and merge them into a single longitudinal dataset before aggregation and investigation of the impact non-census testing has on school accountability measures.

The code presented below is provided as an example of how the growth calculations are conducted for all states in the study. The student data for each state will be cleaned and formatted according to the study guidelines, as was done with the exemplar data from the `SGP` package (Betebenner et al. (2022)) in the example Phase 1 code in the previous section.

2.1. Load SGP package and modify `SGPstateData`

The `SGP` package is required for all growth percentile analyses.

```
require(SGP)
require(data.table)
source("../functions/calculate_SGPs.R")
```

We will use the assessment meta-data from the "Demonstration_COVID" (abbreviated "DEMO_COVID") dataset stored in the `SGPstateData` object. This meta-data is required to use various functions in the `SGP` package.

```
SGPstateData[["State_A"]] <- SGPstateData[["DEMO_COVID"]]
```

3. Condition-Specific Growth Analyses

```
#  Load cleaned, merged and formatted data
if (!exists("State_A_Data_LONG")) {
  source("../functions/freadZIP.R")
  State_A_Data_LONG <-
    freadZIP(
      "Data/Cleaned_Data/Student_LongTestData_State_A_2016-2022_AVI.csv.zip"
    )[YEAR < 2020]
} else {
  State_A_Data_LONG <- State_A_Data_LONG[YEAR < 2020]
}
```

3.1. Simulation Condition 0

In this simulation condition, we want to replicate the base condition of typical census-level testing with the base data set. Growth analyses will include grades 4 to 8, with consecutive-year assessment patterns. Students with a valid score from the previous year and grade level in their historical data will be included in the growth calculations and receive a SGP. Up to two prior scores will be used as available in the data.

3.1.1. Load and combine SGP config scripts

The growth calculation functions of the `SGP` software package allow users to manually specify which test progressions to run. That is, we can define the unique **year-by-grade-by-content area** cohorts of students included in each analysis.

As an example, the 2019 ELA analyses/cohorts are specified with this code:

```
ELA_2019.config <- list(
    ELA.2019 = list(
        sgp.content.areas = rep("ELA", 3),
        sgp.panel.years = c("2017", "2018", "2019"),
        sgp.grade.sequences = list(
            c("3", "4"), c("3", "4", "5"), # Elementary Grades
            c("4", "5", "6"), c("5", "6", "7"), c("6", "7", "8") # Middle
        )
    )
)
```

All configurations are housed in condition specific `R` code scripts. Here we read these in and combine them into a single list object, `state.a.config`, that will be supplied to the `abcSGP` function.

```
source("SGP_CONFIG/Condition_0.R")

state.a.config <-
    c(ELA_2019.config,
      MATHEMATICS_2019.config,
      ELA_2018.config,
      MATHEMATICS_2018.config
    )
```

3.1.2. Calculate condition 0 SGPs

We use the `abcSGP` function from the `SGP` package to produce 2018 and 2019 student growth percentiles. We provide the function with the longitudinal data that was previously cleaned and formatted, as well as the list of analysis configurations and other relevant arguments to tailor the analyses to our specifications.

The SGP analysis section of the appendix assumes the user is operating with their working directory set to "*./Condition_0*".

```
setwd("./Condition_0")
State_A_SGP <-
    abcSGP(
        sgp_object = State_A_Data_LONG,
        state = "State_A",
        steps = c("prepareSGP", "analyzeSGP", "combineSGP"),
        sgp.config = state.a.config,
        sgp.percentiles = TRUE,
        sgp.projections = FALSE,
        sgp.projections.lagged = FALSE,
        sgp.percentiles.baseline = FALSE,
        sgp.projections.baseline = FALSE,
        sgp.projections.lagged.baseline = FALSE,
        simulate.sgps = FALSE,
        parallel.config = list(
            BACKEND = "PARALLEL",
            WORKERS = workers
        )
    )
setwd("..")
```

3.1.3. Re-name and remove the SGP variables as necessary

In order to keep all growth results in the same longitudinal dataset, we will add a `Cnd_0` tag to growth related variables of interest. Extraneous variables will be removed as well before moving on to the next simulation condition. We will save the coefficient matrices from all separate analyses in case we need to replicate or compare results from this condition analysis.

```
setnames(x = State_A_SGP@Data, old = "SGP", new = "SGP_Cnd_0")
State_A_SGP@Data[,
    c("SGP_NORM_GROUP", "SGP_NORM_GROUP_SCALE_SCORES",
      "SCALE_SCORE_PRIOR", "SCALE_SCORE_PRIOR_STANDARDIZED"
    ) := NULL
]

Condition_0_CoefMatrices <- copy(State_A_SGP@SGP[["Coefficient_Matrices"]])
save(Condition_0_CoefMatrices,
     file = "./Condition_0/Condition_0_CoefMatrices.rda")
State_A_SGP@SGP <- NULL
```

3.2. Simulation Condition 1b

In this condition, students test twice per grade span (elementary and middle grades) in both subjects. Tests are administered every year in 3rd, 5th, 6th and 8th grades. Subsequently, all growth analyses will use a single prior score, and can be done with a either consecutive- or skipped-year assessment patterns.

3.2.1. Load and combine SGP config scripts

In order to avoid errors in specification of our analysis configurations, we first remove all previous configuration related objects before reading in the code for condition 1b and proceeding as before. Unlike the other simulation conditions, 1b requires *both* consecutive- and skipped-year configuration scripts.

The 2019 ELA configuration code is provided here as an example and for comparison with the code provided above for condition 0:

```
ELA_2019.config <- list(
    ELA.SKIP.2019 = list(
        sgp.content.areas = rep("ELA", 2),
        sgp.panel.years = c("2017", "2019"),
        sgp.grade.sequences = list(
            c("3", "5"), #  Elementary Grades
            c("6", "8")  #  Middle Grades
```

```
)
    ),
    ELA.2019 = list(
        sgp.content.areas = rep("ELA", 2),
        sgp.panel.years = c("2018", "2019"),
        sgp.grade.sequences = list(c("5", "6")) # Middle Only
    )
)
```

```
rm(list = grep(".config", ls(), value = TRUE))
source("SGP_CONFIG/Condition_1b.R")

sgp_config.c1b <-
    c(ELA_2019.config,
      MATHEMATICS_2019.config,
      ELA_2018.config,
      MATHEMATICS_2018.config
    )
```

3.2.2. Calculate condition 1b SGPs

We again use the `abcSGP` function to compute the student growth percentiles for this simulation condition. Here we use the data with results from condition 0. The updated list of analysis configurations is now provided, and all other relevant arguments remain the same.

```
State_A_SGP <-
    calculate_SGPs(
      sgp_data = State_A_SGP,
      config = sgp_config.c1b,
      condition = "1b",
      workers = list(TAUS = 15)
    )
```

3.3. Simulation Condition 1c

In this condition, students alternate testing in each subject across grade levels. In this simulation, students in grades 3, 5, and 7 take ELA and students in grades 4, 6, 7 take mathematics each year. As with condition 1b, all growth analyses will be conditioned on a single prior score, but only skipped-year assessment patterns can be analyzed.

3.3.1. Load and combine SGP config scripts

We again remove all previous configuration related objects before reading in the condition 1c course progression code. The 2019 ELA configurations are once again provided here for comparison with other simulation conditions.

```
ELA_2019.config <- list(
    ELA.SKIP.2019 = list(
        sgp.content.areas = rep("ELA", 2),
        sgp.panel.years = c("2017", "2019"),
        sgp.grade.sequences = list(
            c("3", "5"), # Elementary Grades
            c("5", "7")  # Middle Grades
        )
    )
)
```

The mathematics configurations are nearly identical to the ELA code, with the exception of the `sgp.grade.sequences` element, which specifies the grades 4 to 6 and grades 6 to 8 progressions. Note that this particular testing pattern means traditional elementary schools will only have growth measures for grade 5 ELA, while traditional middle schools will have growth indicators in all three grades and both content areas. The only contribution mathematics makes to a school's accountability calculation is through grade 4 proficiency (status).

```
rm(list = grep(".config", ls(), value = TRUE))
source("SGP_CONFIG/Condition_1c.R")

sgp_config.c1c <-
    c(ELA_2019.config,
      MATHEMATICS_2019.config,
      ELA_2018.config,
      MATHEMATICS_2018.config
    )
```

3.3.2. Calculate condition 1c SGPs

The call to the`calculate_SGPs` function here performs the same calculation tasks as those made for conditions 1b. The `config` list specifies the cohort data to use in the growth analyses, and the `condition` argument will be used to create and change directories in which the analyses are run, and rename the SGP variable for this condition calculations.

```
State_A_SGP <-
    calculate_SGPs(
      sgp_data = State_A_SGP,
      config = sgp_config.c1c,
      condition = "1c",
      workers = list(TAUS = 15)
    )
```

3.4. Simulation Condition 2

In this condition, all students are tested every two years in each grade and subject on the state's assessments. There are two instances of this condition to simulate:

- Testing only occurs in even years - (e.g., 2016, 2018, etc.)
- Testing only occurs in even years - (e.g., 2017, 2019, etc.)

In both instances, in a year that testing occurs, all students are tested in every grade and subject. As with condition 1c, all growth analyses will be conditioned on a single prior score with skipped-year patterns.

3.4.1. Load and combine SGP config scripts

We again remove all previous configuration related objects before reading in the condition 2 course progression code. The 2019 ELA configurations are once again provided here for comparison with other simulation conditions.

```
ELA_2019.config <- list(
    ELA.SKIP.2019 = list(
        sgp.content.areas = rep("ELA", 2),
        sgp.panel.years = c("2017", "2019"),
        sgp.grade.sequences = list(
            c("3", "5"), # Elementary Grades
            c("4", "6"), c("5", "7"), c("6", "8") # Middle Grades
        )
    )
)
```

```
rm(list = grep(".config", ls(), value = TRUE))
source("SGP_CONFIG/Condition_2.R")

sgp_config.c2 <-
    c(ELA_2019.config,
      MATHEMATICS_2019.config,
      ELA_2018.config,
      MATHEMATICS_2018.config
    )
```

3.4.2. Calculate condition 2 SGPs

The call to the`calculate_SGPs` function here performs the same calculation tasks as those made for conditions 1b and 1c. The `config` list specifies the cohort data to analyze for condition 2, and the data object `State_A_SGP@Data` will now include the results from conditions 0 through 2 when the calculations are complete.

```
State_A_SGP <-
    calculate_SGPs(
      sgp_data = State_A_SGP,
      config = sgp_config.c2,
      condition = "2",
      workers = list(TAUS = 15)
    )
```

3.5. Simulation Condition 3

In this condition, all students are tested every two years at specific grade and subject on the state's assessments. As with Condition 2, there are two instances of this condition to simulate:

- Testing only occurs in even years - (e.g., 2016, 2018, etc.)
- Testing only occurs in even years - (e.g., 2017, 2019, etc.)

In both instances, when testing occurs, students are tested specific grades in both subject areas. As with condition 1c, all growth analyses will be conditioned on a single prior score with skipped-year patterns. ### SGP config scripts

The pattern of testing for this condition is identical to that of condition 1b, with the exception of skipping years. This means that we have already calculated the SGPs for these patterns and do not need to reanalyze the data to get these results. Instead we can simply copy the results from simulation condition 1b that use the skipped-year progressions (i.e. results for grades 5 and 8, but not 6th grade).

For the sake of completeness, however, the 2019 ELA configurations for this condition would be a subset of the condition 1b code, such as this:

```
ELA_2019.config <- list(
    ELA.SKIP.2019 = list(
        sgp.content.areas = rep("ELA", 2),
        sgp.panel.years = c("2017", "2019"),
        sgp.grade.sequences = list(
            c("3", "5"), #  Elementary Grades
            c("6", "8")  #  Middle Grades
        )
    )
)
```

3.5.1. Use condition 1b growth for condition 3

Here we will simply copy the results from condition 1b to a new variable for condition 3. The grade 6 SGPs, which were consecutive-year (grade 5 to grade 6) will be omitted.

```
State_A_SGP@Data[
    GRADE %in% c(5, 8),
    SGP_Cnd_3 := SGP_Cnd_1b
]
```

3.6. Save data

```
if (!dir.exists("Data/Student_Growth"))
    dir.create("Data/Student_Growth", recursive = TRUE)

State_A_Data_LONG <- copy(State_A_SGP@Data)
save(State_A_Data_LONG, file = "Data/Student_Growth/State_A_Data_LONG.rda")

fname <- "Data/Student_Growth/Student_LongTestData_State_A_2016-2019_AVI.csv"
fwrite(State_A_SGP@Data, file = fname)
zip(zipfile = paste0(fname, ".zip"), files = fname, flags = "-mqj")
```

4. Growth and Achievement Aggregations

To simplify the analysis and enable comparisons of results across participating states, we plan to simulate a standard "prototype" accountability model with the following features.

Reporting

- The minimum n-count for computing scores for schools and disaggregated student groups is varied depending on the simulated condition.
- The disaggregated student groups should include economically disadvantaged students, students from racial and ethnic groups, children with disabilities, and English learners, as long as they meet the minimum n-count threshold in the simulated condition.

Indicators

- **Academic achievement** is the percentage of students in the school meeting the proficiency in ELA and mathematics (as defined by the 'Proficient' cut score on the statewide assessment).
- The computation of the ELA and math proficiency rates are adjusted if a school or student group does not have at least 95% participation.
- For the other academic indicator, we apply the following rules:
    - If student-level academic growth (consecutive-year or skip-year) can be computed, then we will use it for this indicator. For consistency, we will calculate student growth percentiles (SGPs) using the student-level assessment data.
    - If student-level academic growth (consecutive-year or skip-year) cannot be computed, then we will use an improvement measure, defined as the change in average scale scores for each grade-level subject area test between administrations for the school or student group.

Summative Rating Computation

All indicator scores are standardized by transforming into z-scores. Use the following means and standard deviations (SD) for the z-score computations (of all schools and student groups):

- *Academic achievement*
    - Mean: mean student-level proficiency rate for the focus year
    - SD: student-level proficiency rate SD for the focus year
    - standardized by year, subject and grade.
- *Other academic indicator* - ***growth***
    - SGPs, being percentiles, can be converted directly to a standardized metric[1].
- *Other academic indicator* - ***improvement***
    - Mean: mean student-level scale score changes for the focus year, calculated separately for each grade level and subject area
    - SD: SD of student-level scale scores for the focus year, calculated separately for each grade level and subject area
    - standardized by year, subject and grade.
- *Graduation rates, progress in ELP, and SQSS*
    - Mean: mean school-level indicator scores for the focus year
    - SD: SD of school-level indicator scores for the focus year

4.1. Additional variables for aggregated results

A simple '`1/0`' binary indicator for proficiency will allow us to compute descriptive statistics (e.g., percent proficient) easily and consistently across all states included in the report.

```
##    Proficient/Not (1/0) binary indicator.
State_A_Data_LONG[,
  PROFICIENCY := fcase(
    ACHIEVEMENT_LEVEL %in% c("Partially Proficient", "Unsatisfactory"), 0L,
    ACHIEVEMENT_LEVEL %in% c("Advanced", "Proficient"), 1L
  )
]
```

4.2. Condition specific summary tables - all students by schools

[1] **Ex. in R:** `qnorm(c(1, 10, 25, 50, 75, 90, 99)/100)` gives the z-score for the 1st, 10th, 25th, … etc., percentiles. For more on mapping percentiles on to the standard-normal distribution, see this site

NOTE TO LESLIE & EMMA These tables and aggregations are my first attempts to both interpret and implement what I've read in the "Analysis Specification" document. This is how I would approach this (sub-)Phase given my past experience in aggregating growth and achievement data.

The following is an example of a preliminary school-level aggregation table for condition 0. Each condition will have a similar table, generally with only the appropriate `SGP` variable substituted for `SGP_Cnd_0` and changes to the inclusion criteria (`YEAR`, `GRADE` and sometimes `CONTENT_AREA`).

```
sch_summary_cnd_0 <-
    State_A_Data_LONG[
        YEAR %in% c(2018, 2019) &
        GRADE %in% 3:8,
        .(TotalN = .N,
          ProfN = sum(PROFICIENCY == 1L),
          GrowthN = sum(!is.na(SGP_Cnd_0)),
          MGP = round(mean(SGP_Cnd_0, na.rm = TRUE), 1),
          PctProf = round(mean(PROFICIENCY, na.rm = TRUE), 3)*100
        ),
        keyby = c("SchoolID", "YEAR", "CONTENT_AREA")
    ]
```

Per the specifications doc, this table would be widened to have a column for each subject. This can be achieved with this code (retaining all specified descriptive statistics).

```
sch_summary_cnd_0w <-
    dcast(
        data = sch_summary_cnd_0,
        formula = SchoolID + YEAR ~ CONTENT_AREA,
        sep = "..",
        value.var = names(sch_summary_cnd_0) %w/o% key(sch_summary_cnd_0)
    )
setnames(
    sch_summary_cnd_0w,
    sapply(
      names(sch_summary_cnd_0w),
      \(f) {
        tmp.name <- strsplit(f, "[.][.]")[[1]] |> rev() |> paste(collapse = "_")
        gsub("MATHEMATICS", "Math", tmp.name)
      }
    ) |> unlist()
)
```

Creating summary tables for the other conditions would only require changing the data records selected (year, grade and content areas as defined for each condition) and the growth variable specified in the aggregation code above. Adding in the demographic variables is a simple addition of the variable of interest into the `keyby` argument of the `data.table` aggregation. Since we are going to be doing this numerous times, we will use a custom function to create these tables, rather than copying the code for each use case.

You may notice that there are more summary calculations than what will be used (e.g., percent proficient and mean standardized scale scores). Those are included for our review - so we can easily see what a z-score, of for example 0.5, corresponds to in the actual percent proficient or mean SGP. Here are two schools from the condition 0 table:

```
school_aggregation_all_students[
    Condition == 0 & SchoolID %in% c("1001E", "3801M"), # rows to keep
    c(key(school_aggregation_all_students)[-1],
      grep("ELA", names(school_aggregation_all_students), value = TRUE)
    ),  #  columns to keep
    with = FALSE
  ] |>
    setkey(SchoolID) |> print()
```

```
 YEAR SchoolID ELA_TotalN ELA_ProfN ELA_GrowthN ELA_MGP ELA_PctProf
 2018    1001E        120       109          73  49.740      90.833
 2019    1001E        162       146          90  54.044      90.123
 2018    3801M         49        15          44  50.727      30.612
 2019    3801M         26        17          23  71.652      65.385
```

We can look at these tables in a number of ways to make sure we are getting what is expected. A simply cross-tab by year shows that many schools do not get a summary in condition 1c given the testing pattern:

```
table(school_aggregation_all_students[, .(Condition, YEAR), is.na(Math_MGP)])
```

```
, , YEAR = 2018

       Condition
is.na     0  1b  1c   2   3
  FALSE 334 335 149 335 250
  TRUE    2   1 187   1  86

, , YEAR = 2019

       Condition
is.na     0  1b  1c   2   3
  FALSE 334 334 149 334 251
  TRUE    2   2 187   2  85
```

```
fwrite(school_aggregation_all_students,
    file = "Data/School_Summaries/School_Condition_AllGrowth_AllStudents_State_A_AllYears_AVI.csv"
)
```

4.3. Achievement Improvement Aggregations

The simulation condition 1a does not allow for growth calculations and will instead use an indicator of status improvement. This **improvement** measure is defined as the change in average scale scores for each grade-level content area test between administrations for the school or student group.

For this aggregation we will create status summaries in a similar way as the other conditions, but include all available years. Lagged values are then created and the change scores calculated.

```
sch_summary_cnd_1a <-
    State_A_Data_LONG[
        GRADE %in% c(5, 8),
        .(N = sum(!is.na(SCALE_SCORE)),
          MeanScore = mean(SCALE_SCORE, na.rm = TRUE),
          ScoreSD = sd(SCALE_SCORE, na.rm = TRUE)
        ),
        keyby = c("YEAR", "CONTENT_AREA", "GRADE", "SchoolID")
    ]

#  Create lagged variables (1 year lag):
setkeyv(
  sch_summary_cnd_1a,
  c("SchoolID", "CONTENT_AREA", "YEAR", "GRADE")
)
cfaTools::getShiftedValues(
    sch_summary_cnd_1a,
    shift_group = c("SchoolID", "CONTENT_AREA"),
    shift_variable = c("N", "MeanScore", "ScoreSD"),
    shift_amount = 1L
)

#  Subset the data for the two focus years:
sch_summary_cnd_1a <-
    sch_summary_cnd_1a[YEAR %in% c(2018, 2019)]

#  Calculate Change Score (Z - effect size?)
# ELA_G5_ZDiff = (ELA_G5AvgScore_<FYear> - ELA_G5AvgScore_<PYear>) / [(ELA_G5StdDev_<FYear>  ELA_G5N_<
```

```
FYear> + ELA_G5StdDev_<PYear>  ELA_G5N_<PYear>) / (ELA_G5N_<FYear> + ELA_G5N_<PYear>)]

sch_summary_cnd_1a[,
    ZDiff := (MeanScore - MeanScore_LAG_1) / ((ScoreSD*N + ScoreSD_LAG_1*N_LAG_1)/(N + N_LAG_1))
]
```

Here is our example school's improvement numbers

```
sch_summary_cnd_1a[SchoolID == "1001E",
    c(key(sch_summary_cnd_1a)[-1], "ZDiff"), with = FALSE
]
```

```
 CONTENT_AREA YEAR GRADE      ZDiff
          ELA 2018     5  0.4892425
          ELA 2019     5 -0.2357825
  MATHEMATICS 2018     5  0.3524293
  MATHEMATICS 2019     5 -0.3911958
```

One important factor to consider with the "school improvement" indicator is the issue of "regression to the mean" that is expected to occur. There are a strong relationships between current/prior scores and change scores that may require correction:

```
cor(
  sch_summary_cnd_1a[, MeanScore, ZDiff],
  use = 'na.or.complete'
) |> round(3)
```

```
          ZDiff MeanScore
ZDiff     1.000     0.401
MeanScore 0.401     1.000
```

```
cor(
  sch_summary_cnd_1a[, MeanScore_LAG_1, ZDiff],
  use = 'na.or.complete'
) |> round(3)
```

```
                ZDiff MeanScore_LAG_1
ZDiff            1.00           -0.47
MeanScore_LAG_1 -0.47            1.00
```

```
sch_summary_cnd_1a[,
  .(Current_Year = cor(MeanScore, ZDiff, use = 'na.or.complete'),
    Prior_Year = cor(MeanScore_LAG_1, ZDiff, use = 'na.or.complete')
  ),
  keyby = c("YEAR", "CONTENT_AREA", "GRADE")
]
```

```
 YEAR CONTENT_AREA GRADE Current_Year Prior_Year
 2018          ELA     5    0.3703319 -0.5039010
 2018          ELA     8    0.5324401 -0.2896151
 2018  MATHEMATICS     5    0.3675901 -0.6119717
 2018  MATHEMATICS     8    0.4814938 -0.4118177
 2019          ELA     5    0.4679060 -0.4311257
 2019          ELA     8    0.3405961 -0.6064881
 2019  MATHEMATICS     5    0.5002665 -0.4175789
 2019  MATHEMATICS     8    0.3384279 -0.4943981
```

Per the specifications doc, here is how we could reshape this summary table so that there are separate columns for each grade and subject.

```
#  Reshape by subject
sch_summary_cnd_1a[, GRADE := paste0("G", GRADE)]
setkeyv(
  sch_summary_cnd_1a,
  c("SchoolID", "CONTENT_AREA", "YEAR", "GRADE")
)
sch_summary_cnd_1a <-
    dcast(
```

```
data = sch_summary_cnd_1a,
        formula = SchoolID + YEAR ~ GRADE + CONTENT_AREA,
        sep = "..",
        value.var = c("N", "ZDiff")
    )
setnames(
    sch_summary_cnd_1a,
    sapply(
      names(sch_summary_cnd_1a),
      \(f) {
        tmp.name <- strsplit(f, "[.][.]")[[1]] |> rev() |> paste(collapse = "_")
        gsub("MATHEMATICS", "Math", tmp.name)
      }
    ) |> unlist()
)
```

If schools have both 5th and 8th grades, then take a weighted average of their _G5_ZDiff and _G8_ZDiff to create the ELA_Improve and Math_Improve variable, otherwise _Improve is equal to the grade level _G5_ZDiff or _G8_ZDiff they DO have in the school.

NOTE: This is not necessary if we break up the schools into Elementary and Middle grade only schools.

```
sch_summary_cnd_1a[,
  ELA_Improve := ELA_G5_ZDiff
][
  is.na(ELA_Improve),
  ELA_Improve := ELA_G8_ZDiff
][
  !is.na(ELA_G5_ZDiff) & !is.na(ELA_G8_ZDiff),
  ELA_Improve := ((ELA_G5_ZDiff * ELA_G5_N) + (ELA_G8_ZDiff * ELA_G8_N))/(ELA_G5_N + ELA_G8_N)
][,
  Math_Improve := Math_G5_ZDiff
][
  is.na(Math_Improve),
  Math_Improve := Math_G8_ZDiff
][
  !is.na(Math_G5_ZDiff) & !is.na(Math_G8_ZDiff),
  Math_Improve := ((Math_G5_ZDiff * Math_G5_N) + (Math_G8_ZDiff * Math_G8_N))/(Math_G5_N + Math_G8_N)
]

sch_summary_cnd_1a[, (grep("ZDiff", names(sch_summary_cnd_1a))) := NULL]
```

Finally, we add the `Group` variable to indicate this data set is for all students, and save files for each year.

```
sch_summary_cnd_1a[, Group := "All"]
setcolorder(sch_summary_cnd_1a, c("YEAR", "SchoolID", "Group"))

fwrite(sch_summary_cnd_1a[YEAR == 2018][, YEAR := NULL],
    file = "Data/School_Summaries/School_Condition_1a_State_A_2018_AVI.csv"
)
fwrite(sch_summary_cnd_1a[YEAR == 2019][, YEAR := NULL],
    file = "Data/School_Summaries/School_Condition_1a_State_A_2019_AVI.csv"
)
```

Note that the above code can be accomplished with the `cond1aAggrGator` function:

```
sch_smry_cnd_1a <- cond1aAggrGator(data_table = State_A_Data_LONG)
```

This function is used below for the condition 1a summaries that are disaggregated by demographic subgroups.

4.4. School level aggregations by demographics

Adding in the demographic variables is a simple addition of the variable of interest into the `keyby` argument of the `data.table` aggregation.

In order to do all the demographic summaries at once, we can combine calls to the function (rather than creating separate tables and THEN combining) using the `rbindlist` function from `data.table`. Note that the demographic subgroup indicator is changed (from the demographic variable name) to the generic "Group" for each individual table.:

All of the demographic aggregation and combination code chunks can be run for each of the `SGP_Cnd*` growth fields. At that point, we can then combine those objects in a wide format, stacked into a long format (with an added "Condition" variable for each table) or written to separate .csv files as described in the specification doc.

The code below creates a stacked long format version with all conditions that contain growth.

We can obtain the condition 1a by demographic subgroup using the following call to the `cond1aAggrGator` function with `group` argument supplied via `lapply`:

```
sch_summary_cnd_1a_demogs <-
  lapply(
    c("Race", "EconDis", "EL", "SWD"),
    \(f) {
      cond1aAggrGator(
        data_table = State_A_Data_LONG,
        group = f
      )
    }
  ) |>
    rbindlist()
```

4.5. A single file with all aggregations

I may be missing something about why all the various .csv files are needed for this step, but if we wanted to save all the conditions and summaries as a single file, we could do something like this (and then save as a single .csv using `fwrite` or similar):

```
setcolorder(
    school_aggregation_all_students,
    names(school_aggregation_student_demogs)
)
school_aggregation_all <-
    rbindlist(
        list(
          school_aggregation_all_students,
          school_aggregation_student_demogs
        )
    )
setkeyv(school_aggregation_all, c("Condition", "SchoolID", "YEAR"))

fwrite(school_aggregation_all,
    file = "Data/School_Summaries/School_Condition_AllGrowth_Demographics_State_A_AllYears_AVI.csv"
)
```

4.5.1. Save the condition/year specific aggregation files

Here is how to split up the single file created above into the individual files per the specification document.

```
school_aggregation_all[
  Condition %in% c("2", "3") & YEAR == 2018,
  Condition := paste0(Condition, "_E")
][
  Condition %in% c("2", "3") & YEAR == 2019,
  Condition := paste0(Condition, "_O")
]

fprefix <- "./Data/School_Summaries/School_Condition_"

for (cond in c("0", "1a", "1b", "1c", "2_E", "2_O", "3_E", "3_O")) {
  for (yr in 2018:2019) {
    tmp.tbl <-
```

```
school_aggregation_all[
        Condition == cond & YEAR == yr
      ][,
        c("Condition", "YEAR") := NULL
      ][
        -(grep(": No", Group)),
      ][,
        Group := gsub(": Yes", "", Group)
      ]

##  School_Condition_<c> _<State>_<FYear>_ <init>.csv
    if (nrow(tmp.tbl) > 0L) {
      fwrite(
        x = tmp.tbl,
        file = paste0(fprefix, cond, "_State_A_", yr, "_AVI.csv")
      )
    }
  }
}
```

Repeat this process of combining, sub-setting and saving for condition 1a.

```
sch_summary_cnd_1a_all <-
    rbindlist(
        list(
          sch_summary_cnd_1a,
          sch_summary_cnd_1a_demogs
        )
    )[
      -(grep(": No", Group)),
    ][,
      Group := gsub(": Yes", "", Group)
    ]

fwrite(sch_summary_cnd_1a_all[YEAR == 2018][, YEAR := NULL],
    file = "Data/School_Summaries/School_Condition_1a_State_A_2018_AVI.csv"
)
fwrite(sch_summary_cnd_1a_all[YEAR == 2019][, YEAR := NULL],
    file = "Data/School_Summaries/School_Condition_1a_State_A_2019_AVI.csv"
)
```

4.6. Summary and notes

- A binary indicator variable for proficiency status is added.
- Methods for using the `data.table` package for calculating school level aggregations for all students and by demographic subgroups were outlined and discussed with examples.

References

Betebenner, Damian W., Adam VanIwaarden, Ben Domingue, and Yi Shang. 2022. *SGP: Student Growth Percentiles & Percentile Growth Trajectories.*
Betebenner, Damian W., Adam VanIwaarden, Ben Domingue, and Yi Shang. 2022. *SGP: Student Growth Percentiles & Percentile Growth Trajectories.*

Appendix R

1. Computational Environment

Since R and R packages are constantly evolving, it is critical to document information such as software package versions (primary and auxilary) and the computer system platform used in data analyses and report generation. This appendix provides the R and system specifications used in the creation of this report.

1.1. General R software and system information

Table R1: Platform Information for R Session

Setting	Value
version	R version 4.2.2 (2022-10-31)
os	macOS Ventura 13.1
system	x86_64, darwin17.0
ui	X11
language	(EN)
collate	en_US.UTF-8
ctype	en_US.UTF-8
tz	America/Denver
date	2022-12-28
pandoc	2.19.2 @ /usr/local/bin/ (via rmarkdown)

1.2. Attached and loaded R packages

The following packages (non-base R) were attached:

Table R2: Attached R Packages for R Session

Package	Version	Source	Date Installed
data.table	1.14.7	Github (Rdatatable/data.table@cb8aeff)	11/21/2022
SGP	2.0-1.3	Github (centerforassessment/SGP@38499af)	12/15/2022
SGPdata	26.0-0.0	CRAN (R 4.2.0)	05/27/2022

In addition to the attached packages in the table above, the following packages were loaded via a namespace (and not attached):

abind *(1.4-5)*

backports *(1.4.1)*, **base64enc** *(0.1-3)*, **bit** *(4.0.5)*, **bit64** *(4.0.5)*, **blob** *(1.2.3)*, **bookdown** *(0.31)*, **boot** *(1.3-28.1)*, **brio** *(1.1.3)*, **broom** *(1.0.2)*, **bslib** *(0.4.2)*

cachem *(1.0.6)*, **Cairo** *(1.6-0)*, **callr** *(3.7.3)*, **car** *(3.1-1)*, **carData** *(3.0-5)*, **cfaDocs** *(0.0-1.11)*, **cfaTools** *(0.0-1.994)*, **checkmate** *(2.1.0)*, **chromote** *(0.1.1)*, **class** *(7.3-20)*, **cli** *(3.5.0)*, **clipr** *(0.8.0)*, **cluster** *(2.1.4)*, **codetools** *(0.2-18)*, **colorspace** *(2.0-3)*, **crayon** *(1.5.2)*

DBI *(1.1.3)*, **deldir** *(1.0-6)*, **DEoptimR** *(1.0-11)*, **devtools** *(2.4.5)*, **digest** *(0.6.31)*, **doParallel** *(1.0.17)*, **doRNG** *(1.8.3)*, **dplyr** *(1.0.10)*

e1071 *(1.7-12)*, **ellipsis** *(0.3.2)*, **equate** *(2.0.8)*, **evaluate** *(0.19)*

fansi *(1.0.3)*, **fastmap** *(1.1.0)*, **foreach** *(1.5.2)*, **foreign** *(0.8-84)*, **Formula** *(1.2-4)*, **fs** *(1.5.2)*

generics *(0.1.3)*, **ggplot2** *(3.4.0)*, **glue** *(1.6.2)*, **gridBase** *(0.4-7)*, **gridExtra** *(2.3)*, **gtable** *(0.3.1)*, **gtools** *(3.9.4)*

HDInterval *(0.2.4)*, **highr** *(0.10)*, **Hmisc** *(4.7-2)*, **htmlTable** *(2.4.1)*, **htmltools** *(0.5.4)*, **htmlwidgets** *(1.6.0)*, **httpuv** *(1.6.7)*, **httr** *(1.4.4)*

interp *(1.1-3)*, **iterators** *(1.0.14)*

jpeg *(0.1-10)*, **jquerylib** *(0.1.4)*, **jsonlite** *(1.8.4)*

knitr *(1.41)*

laeken *(0.5.2)*, **later** *(1.3.0)*, **lattice** *(0.20-45)*, **latticeExtra** *(0.6-30)*, **lazyeval** *(0.2.2)*, **lifecycle** *(1.0.3)*, **lmtest** *(0.9-40)*

magrittr *(2.0.3)*, **MASS** *(7.3-58.1)*, **Matrix** *(1.5-3)*, **MatrixModels** *(0.5-1)*, **matrixStats** *(0.63.0)*, **memoise** *(2.0.1)*, **mice** *(3.15.0)*, **miceadds** *(3.15-21)*, **mime** *(0.12)*, **miniUI** *(0.1.1.1)*, **mitools** *(2.4)*, **mnormt** *(2.1.1)*, **munsell** *(0.5.0)*

nnet *(7.3-18)*, **numDeriv** *(2016.8-1.1)*

pagedown *(0.20.1)*, **pillar** *(1.8.1)*, **pkgbuild** *(1.4.0)*, **pkgconfig** *(2.0.3)*, **pkgload** *(1.3.2)*, **plotly** *(4.10.1)*, **png** *(0.1-8)*, **prettyunits** *(1.1.1)*, **processx** *(3.8.0)*, **profvis** *(0.3.7)*, **promises** *(1.2.0.1)*, **proxy** *(0.4-27)*, **ps** *(1.7.2)*, **purrr** *(1.0.0)*

quantreg *(5.94)*

R.methodsS3 *(1.8.2)*, **R.oo** *(1.25.0)*, **R.utils** *(2.12.2)*, **R6** *(2.5.1)*, **randomNames** *(1.5-0.0)*, **ranger** *(0.14.1)*, **RColorBrewer** *(1.1-3)*, **Rcpp** *(1.0.9)*, **remotes** *(2.4.2)*, **rlang** *(1.0.6)*, **rmarkdown** *(2.19)*, **rngtools** *(1.5.2)*, **robustbase** *(0.95-0)*, **rpart** *(4.1.19)*, **RSQLite** *(2.2.20)*, **rstudioapi** *(0.14)*

sass *(0.4.4)*, **scales** *(1.2.1)*, **sessioninfo** *(1.2.2)*, **shiny** *(1.7.4)*, **sn** *(2.1.0)*, **sp** *(1.5-1)*, **SparseM** *(1.81)*, **stringi** *(1.7.8)*, **stringr** *(1.5.0)*, **survival** *(3.4-0)*, **svglite** *(2.1.0)*, **systemfonts** *(1.0.4)*

testthat *(3.1.6)*, **tibble** *(3.1.8)*, **tidyr** *(1.2.1)*, **tidyselect** *(1.2.0)*, **toOrdinal** *(1.3-1.0)*

urlchecker *(1.0.1)*, **usethis** *(2.1.6)*, **utf8** *(1.2.2)*

vcd *(1.4-10)*, **vctrs** *(0.5.1)*, **VIM** *(6.2.2)*, **viridisLite** *(0.4.1)*

webshot2 *(0.1.0)*, **websocket** *(1.4.1)*

xaringan *(0.28.1)*, **xfun** *(0.36)*, **xtable** *(1.8-4)*

yaml *(2.3.6)*, **ymlthis** *(0.1.7)*

zoo *(1.8-11)*