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 the code used to prepare and format the data required for each condition of the **"Re-evaluating the Efficiency and Efficacy of Annual Census Standardized Testing for Accountability Purposes"** study. Each condition requires the data to be formatted in a consistent manner, and addition variables to be created. With the properly formatted and amended data, we then proceed with the Student Growth Percentiles (SGP) analyses and results aggregations.

Keywords: draft – do not cite.

1. Data cleaning and preparation

For this simulation 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 (Betebenner et al. 2022) package, 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, and then remove the perturbed score version and use the original scale score.

```
# First load and rename/remove SCALE_SCORE* variables included in the data
State_A_Data_LONG <- copy(SGPdata::sgpData_LONG_COVID)[YEAR < 2020]
State_A_Data_LONG[, SCALE_SCORE := NULL]
setnames(State_A_Data_LONG, "SCALE_SCORE_without_COVID_IMPACT", "SCALE_SCORE")
```

NOTE TO LESLIE & EMMA

We will need to either come to an agreement on the longitudinal data naming or rename according to the SGP package conventions. Here I rename the demographic variables to match the "analysis specification" documents and remove some of the variables we will not be looking at or using.

```
setnames(
  State_A_Data_LONG,
  c("ETHNICITY", "FREE_REDUCED_LUNCH_STATUS", "ELL_STATUS", "IEP_STATUS"),
  c("Race", "EconDis", "EL", "SWD")
)
State_A_Data_LONG[, Race := as.character(Race)]
State_A_Data_LONG[Race == "African American", Race := "Black"]
State_A_Data_LONG[, EconDis := gsub("Free Reduced Lunch", "FRL", EconDis)]

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

1.3. Additional variables for aggregated results

A standardized score variable and an achievement proficiency indicator are required for school level aggregations, final analyses and results comparisons. The standardized scale score variable is scaled by each ***year by subject by grade*** test mean and standard deviation[1].

NOTE: I am doing this here, but it could easily be done before the aggregation/summarization step. It is NOT required as any part of the growth analyses.

```
##    Standardize SCALE_SCORE by CONTENT_AREA and GRADE using 2019 norms
State_A_Data_LONG[,
  Z_SCORE := scale(SCALE_SCORE),
```

[1]The original SCALE_SCORE variable is used in the SGP calculations.

```
by = c("YEAR", "CONTENT_AREA", "GRADE")
]
```

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
  )
]

State_A_Data_LONG[,
  Z_PROFICIENCY := scale(PROFICIENCY),
  by = c("YEAR", "CONTENT_AREA", "GRADE")
]
```

1.4. Summary and notes

- "State A" uses the 2016 to 2019 subset of the *sgpData_LONG_COVID* dataset from the SGPData package.
  - The "original", unperturbed version of the scaled score is retained.
- A standardized scale score variable is added (scaled by unique grade, content area and annual assessment).
- A binary indicator variable for proficiency status is added.

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 the R code to the same set of data, thus only producing growth measures for 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 SGP analysis step. Only cohort-referenced SGPs are calculated (SGP projections and targets are omitted). The goal of this step is simply to create growth percentiles and merge them into the longitudinal data before aggregation and investigation of the impact non-census testing has on school accountability measures.

2.1. Load SGP package and modify SGPstateData

The SGP package is required for all growth percentile analyses.

```
require(SGP)
```

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"]]
SGPstateData[["State_A"]][["Growth"]][["Levels"]] <-
  SGPstateData[["State_A"]][["Growth"]][["Cutscores"]] <-
    SGPstateData[["State_A"]][["SGP_Configuration"]][["percentile.cuts"]] <-
      NULL
```

2.2. 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.

2.2.1. 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
    )
```

2.2.2. 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 = parallel::detectCores(logical = FALSE)
        )
    )
```

2.2.3. 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.

```
rm(State_A_Data_LONG)
State_A_Data_LONG <- copy(State_A_SGP@Data)
```

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

2.3. 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.

2.3.1. 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")

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

2.3.2. 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.

```
setwd("./Condition_1b")
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,
```

```
parallel.config = list(
            BACKEND = "PARALLEL",
            WORKERS = parallel::detectCores(logical = FALSE)
        )
    )
setwd("..")

rm(State_A_Data_LONG)
State_A_Data_LONG <- copy(State_A_SGP@Data)

setnames(x = State_A_Data_LONG, old = "SGP", new = "SGP_Cnd_1b")
State_A_Data_LONG[,
    c("SGP_NORM_GROUP", "SGP_NORM_GROUP_SCALE_SCORES",
      "SCALE_SCORE_PRIOR", "SCALE_SCORE_PRIOR_STANDARDIZED"
    ) := NULL
]
```

2.4. 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.

2.4.1. 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")

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

2.4.2. Calculate condition 1c SGPs

The call to the`abcSGP` function here is identical to that made for conditions 1b and 2. The data object `State_A_Data_LONG` now includes the results from conditions 0 and 1b, and the configurations have been updated.

```
setwd("./Condition_1c")
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 = parallel::detectCores(logical = FALSE)
        )
    )
setwd("..")

rm(State_A_Data_LONG)
State_A_Data_LONG <- copy(State_A_SGP@Data)

setnames(x = State_A_Data_LONG, old = "SGP", new = "SGP_Cnd_1c")
State_A_Data_LONG[,
    c("SGP_NORM_GROUP", "SGP_NORM_GROUP_SCALE_SCORES",
      "SCALE_SCORE_PRIOR", "SCALE_SCORE_PRIOR_STANDARDIZED"
    ) := NULL
]
```

2.5. 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.

2.5.1. 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")

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

2.5.2. Calculate condition 2 SGPs

The call to the`abcSGP` function here is identical to that made for conditions 1b and 1c. The data object `State_A_Data_LONG` now includes the results from conditions 0 through 1c, and the configuration object, `state.a.config`, has been updated.

```
setwd("Condition_2")
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 = parallel::detectCores(logical = FALSE)
        )
    )
setwd("..")

rm(State_A_Data_LONG)
State_A_Data_LONG <- copy(State_A_SGP@Data)

setnames(x = State_A_Data_LONG, old = "SGP", new = "SGP_Cnd_2")
State_A_Data_LONG[,
    c("SGP_NORM_GROUP", "SGP_NORM_GROUP_SCALE_SCORES",
      "SCALE_SCORE_PRIOR", "SCALE_SCORE_PRIOR_STANDARDIZED"
    ) := NULL
]
```

2.6. 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
        )
    )
)
```

2.6.1. 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_Data_LONG[
    GRADE %in% c(5, 8),
    SGP_Cnd_3 := SGP_Cnd_1b
]

if (!dir.exists("Data")) dir.create("Data")
save("State_A_Data_LONG", file = "Data/State_A_Data_LONG.rda")
```

3. 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[2].
- *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

3.1. Condition specific summary tables

[2] **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 (as well as my variable additions such as `Z_PROFICIENCY` and `Z_SCORE`) are my first attempts to both interpret and implement what I've read in the "Analysis Specification" document. Since I have some extensive experience in aggregating growth and achievement data, this is how I would approach it at this early stage…

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),
          Mean_Score = round(mean(Z_SCORE, na.rm = TRUE), 2),
          Pcnt_Prof = round(mean(PROFICIENCY, na.rm = TRUE), 3)*100,
          Z_Status = round(mean(Z_PROFICIENCY, na.rm = TRUE), 3),
          Z_Growth = round(mean(qnorm(SGP_Cnd_0/100), na.rm = TRUE), 3)
        ),
        keyby = c("YEAR", "CONTENT_AREA", "SCHOOL_NUMBER")
    ]
```

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:

```
sch_summary_cnd_0[SCHOOL_NUMBER %in% c(1001, 3801)] |>
    setkey(SCHOOL_NUMBER) |> print()
```

```
 YEAR CONTENT_AREA SCHOOL_NUMBER TotalN ProfN GrowthN  MGP Mean_Score Pcnt_Prof Z_Status Z_Growth
 2018          ELA          1001    120   109      73 49.7       0.62      90.8    0.511   -0.045
 2018  MATHEMATICS          1001    120   107      73 61.5       0.79      89.2    0.553    0.386
 2019          ELA          1001    162   146      90 54.0       0.66      90.1    0.501    0.151
 2019  MATHEMATICS          1001    162   145      90 55.1       0.77      89.5    0.565    0.188
 2018          ELA          3801    170    44     103 43.4      -0.89      25.9   -0.870   -0.264
 2018  MATHEMATICS          3801    170    31     103 40.7      -0.91      18.2   -0.855   -0.317
 2019          ELA          3801    151    67     101 53.4      -0.61      44.4   -0.467    0.142
 2019  MATHEMATICS          3801    151    52     101 54.3      -0.66      34.4   -0.531    0.109
```

At some point we will probably want to combine the condition aggregations into a single table so that we can do direct condition comparisons. We can also clean up some of the extra descriptive statistics, re-order the columns or anything else.

```
#  Combine all condition-specific tables into one:
composite_summary <-
    sch_summary_cnd_0[
    sch_summary_cnd_1b][
    sch_summary_cnd_1c][
    sch_summary_cnd_2][
    sch_summary_cnd_3]

#  Remove extraneous aggregations:
composite_summary[,
    grep("Pcnt_Prof|Mean_", names(composite_summary), value = TRUE) :=
      NULL
]

#  School No. '1001' - changes in Growth N count
composite_summary[SCHOOL_NUMBER == 1001,
  c("YEAR", "CONTENT_AREA",
```

```
grep("GrowthN", names(composite_summary), value = TRUE)
  ), with = FALSE
]
```

```
 YEAR CONTENT_AREA GrowthN GrowthN_1b GrowthN_1c GrowthN_2 GrowthN_3
 2018          ELA      73         40         40        40        40
 2018  MATHEMATICS      73         40          0        40        40
 2019          ELA      90         38         38        38        38
 2019  MATHEMATICS      90         38          0        38        38
```

```
#  School No. '1001' - Growth (Z-SGP) summaries
composite_summary[SCHOOL_NUMBER == 1001,
    c("YEAR", "CONTENT_AREA",
# All relevant aggregations at once:
# sort(grep("Z_Growth|Z_Status", names(composite_summary), value = TRUE))
      grep("Z_Growth", names(composite_summary), value = TRUE) # just z-growth
    ), with = FALSE
]
```

```
 YEAR CONTENT_AREA Z_Growth Z_Growth_1b Z_Growth_1c Z_Growth_2 Z_Growth_3
 2018          ELA   -0.045       0.214       0.214      0.214      0.214
 2018  MATHEMATICS    0.386       0.459         NaN      0.459      0.459
 2019          ELA    0.151      -0.128      -0.128     -0.128     -0.128
 2019  MATHEMATICS    0.188       0.130         NaN      0.130      0.130
```

```
#  School No. '1001' - Status (Z-proficient %) summaries
composite_summary[SCHOOL_NUMBER == 1001,
    c("YEAR", "CONTENT_AREA",
      grep("Z_Status", names(composite_summary), value = TRUE)
    ), with = FALSE
]
```

```
 YEAR CONTENT_AREA Z_Status Z_Status_1b Z_Status_1c Z_Status_2 Z_Status_3
 2018          ELA    0.511       0.533       0.533      0.511      0.533
 2018  MATHEMATICS    0.553       0.513       0.657      0.553      0.513
 2019          ELA    0.501       0.493       0.493      0.501      0.493
 2019  MATHEMATICS    0.565       0.536       0.622      0.565      0.536
```

3.2. 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),
        .(TotalN = .N,
          Mean_Score = round(mean(Z_SCORE, na.rm = TRUE), 2),
          Z_Status = round(mean(Z_PROFICIENCY, na.rm = TRUE), 3)
        ),
        keyby = c("YEAR", "CONTENT_AREA", "GRADE", "SCHOOL_NUMBER")
    ]

#  Create lagged variables (1 year lag):
setkeyv(
  sch_summary_cnd_1a,
  c("SCHOOL_NUMBER", "CONTENT_AREA", "YEAR", "GRADE")
)
cfaTools::getShiftedValues(
    sch_summary_cnd_1a,
    shift_group = c("SCHOOL_NUMBER", "CONTENT_AREA"),
    shift_variable = c("TotalN", "Mean_Score", "Z_Status"),
    shift_amount = 1L
)
```

```
sch_summary_cnd_1a[YEAR %in% c(2018, 2019)]

#  Calculate changes (current year minus 1 year lag)
sch_summary_cnd_1a[,
    TotalN_Change := TotalN - TotalN_LAG_1
][,
    Mean_Score_Change := Mean_Score - Mean_Score_LAG_1
][,
    Z_Status_Change := Z_Status - Z_Status_LAG_1
]
```

Here is our example school's improvement numbers

```
sch_summary_cnd_1a[SCHOOL_NUMBER == 1001,
    c(key(sch_summary_cnd_1a)[-1],
      "TotalN_Change", "Mean_Score_Change", "Z_Status_Change"
    ), with = FALSE
]
```

```
 CONTENT_AREA YEAR GRADE TotalN_Change Mean_Score_Change Z_Status_Change
          ELA 2018     5            24              0.56           0.307
          ELA 2019     5            -3             -0.20          -0.113
  MATHEMATICS 2018     5            24              0.54           0.059
  MATHEMATICS 2019     5            -3             -0.41           0.164
```

3.3. 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. Since we are going to be doing this numerous times, it might be smart to create a custom function to create these tables, rather than copying the code for each use case.

```
schoolAggrGator =
  function(
    data_table,
    growth.var,
    groups = c("YEAR", "CONTENT_AREA", "SCHOOL_NUMBER")
  ) {
    data_table[,
      # the list of summaries can be reduced/increased/amended as needed:
      .(TotalN = .N,
        ProfN = sum(PROFICIENCY==1L),
        GrowthN = sum(!is.na(get(growth.var))),
        MGP = round(mean(get(growth.var), na.rm = TRUE), 1),
        Mean_Score = round(mean(Z_SCORE, na.rm = TRUE), 2),
        # Pcnt_Prof = round(mean(PROFICIENCY, na.rm = TRUE), 3)*100,
        Z_Status = round(mean(Z_PROFICIENCY, na.rm = TRUE), 3),
        Z_Growth = round(mean(qnorm(get(growth.var)/100), na.rm = TRUE), 3)
      ),
      keyby = groups
    ][]
  }
```

Our original "base" condition table can be reproduced now with this call to our function:

```
schoolAggrGator(
    data_table =
      State_A_Data_LONG[YEAR %in% c(2018, 2019) & GRADE %in% 3:8,],
    growth.var = "SGP_Cnd_0",
)
```

Our function used for demographics (Economic Disadvantage):

```
schoolAggrGator(
    data_table =
      State_A_Data_LONG[YEAR %in% c(2018, 2019) & GRADE %in% 3:8,],
    growth.var = "SGP_Cnd_0",
    groups = c("YEAR", "CONTENT_AREA", "SCHOOL_NUMBER", "EconDis")
)
```

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):

```
demog_cond_0 <-
    rbindlist(
      list(
        schoolAggrGator(
          data_table =
            State_A_Data_LONG[YEAR %in% c(2018, 2019) & GRADE %in% 3:8,],
          growth.var = "SGP_Cnd_0",
          groups = c("YEAR", "CONTENT_AREA", "SCHOOL_NUMBER", "Race")
        ) |> setnames("Race", "Group"),
        schoolAggrGator(
          data_table =
            State_A_Data_LONG[YEAR %in% c(2018, 2019) & GRADE %in% 3:8,],
          growth.var = "SGP_Cnd_0",
          groups = c("YEAR", "CONTENT_AREA", "SCHOOL_NUMBER", "EconDis")
        ) |> setnames("EconDis", "Group"),
        schoolAggrGator(
          data_table =
            State_A_Data_LONG[YEAR %in% c(2018, 2019) & GRADE %in% 3:8,],
          growth.var = "SGP_Cnd_0",
          groups = c("YEAR", "CONTENT_AREA", "SCHOOL_NUMBER", "EL")
        ) |> setnames("EL", "Group"),
        schoolAggrGator(
          data_table =
            State_A_Data_LONG[YEAR %in% c(2018, 2019) & GRADE %in% 3:8,],
          growth.var = "SGP_Cnd_0",
          groups = c("YEAR", "CONTENT_AREA", "SCHOOL_NUMBER", "SWD")
        ) |> setnames("SWD", "Group")
      )
    )
```

Here a subset of the output from the example school:

```
demog_cond_0[
  SCHOOL_NUMBER == 1001 & YEAR == 2019
][,
  c("YEAR", "SCHOOL_NUMBER", "MGP", "Mean_Score") := NULL
][]
```

CONTENT_AREA	Group	TotalN	ProfN	GrowthN	Z_Status	Z_Growth
ELA	Asian	30	27	21	0.499	0.225
ELA	Black	12	10	8	0.357	-0.274
ELA	Hispanic	30	24	18	0.287	-0.046
ELA	Other	10	9	7	0.499	0.559
ELA	White	80	76	36	0.604	0.222
MATHEMATICS	Asian	30	23	21	0.316	0.064
MATHEMATICS	Black	12	10	8	0.445	-0.117
MATHEMATICS	Hispanic	30	24	18	0.379	-0.082
MATHEMATICS	Other	10	10	7	0.800	0.400
MATHEMATICS	White	80	78	36	0.717	0.423
ELA	FRL: No	146	133	83	0.522	0.140
ELA	FRL: Yes	16	13	7	0.312	0.280
MATHEMATICS	FRL: No	146	132	83	0.586	0.166
MATHEMATICS	FRL: Yes	16	13	7	0.377	0.455
ELA	ELL: No	159	143	89	0.497	0.143
ELA	ELL: Yes	3	3	1	0.709	0.878
MATHEMATICS	ELL: No	159	142	89	0.562	0.199
MATHEMATICS	ELL: Yes	3	3	1	0.753	-0.739
ELA	IEP: No	152	146	83	0.627	0.215
ELA	IEP: Yes	10	0	7	-1.408	-0.608
MATHEMATICS	IEP: No	152	143	83	0.660	0.265
MATHEMATICS	IEP: Yes	10	2	7	-0.869	-0.725
CONTENT_AREA	Group	TotalN	ProfN	GrowthN	Z_Status	Z_Growth

Another example of how to combine the aggregations along with output from a different school:

```
demog_cond0 <-
  lapply(
    c("Race", "EconDis", "EL", "SWD"),
    \(f) {
      schoolAggrGator(
          data_table =
            State_A_Data_LONG[YEAR %in% c(2018, 2019) & GRADE %in% 3:8, ],
          growth.var = "SGP_Cnd_0",
          groups = c("YEAR", "CONTENT_AREA", "SCHOOL_NUMBER", f)
        ) |> setnames(f, "Group")
    }
  ) |> rbindlist()

demog_cond0[
  SCHOOL_NUMBER == 3801 & YEAR == 2019
][,
  c("YEAR", "SCHOOL_NUMBER", "MGP", "Mean_Score") := NULL
][]
```

CONTENT_AREA	Group	TotalN	ProfN	GrowthN	Z_Status	Z_Growth
ELA	Asian	13	5	12	-0.590	0.111
ELA	Black	4	2	2	-0.350	-2.054
ELA	Hispanic	36	23	22	-0.053	0.870
ELA	Other	8	4	6	-0.347	1.012
ELA	White	90	33	59	-0.630	-0.137
MATHEMATICS	Asian	13	4	12	-0.567	0.184
MATHEMATICS	Black	4	2	2	-0.276	-1.555
MATHEMATICS	Hispanic	36	19	22	-0.138	0.489
MATHEMATICS	Other	8	3	6	-0.426	0.520
MATHEMATICS	White	90	24	59	-0.704	-0.033
ELA	FRL: No	19	13	9	0.043	0.751
ELA	FRL: Yes	132	54	92	-0.540	0.082
MATHEMATICS	FRL: No	19	11	9	-0.042	0.274
MATHEMATICS	FRL: Yes	132	41	92	-0.601	0.093
ELA	ELL: No	109	61	69	-0.221	0.134
ELA	ELL: Yes	42	6	32	-1.104	0.159
MATHEMATICS	ELL: No	109	46	69	-0.372	0.276
MATHEMATICS	ELL: Yes	42	6	32	-0.944	-0.251
ELA	IEP: No	134	65	86	-0.379	0.322
ELA	IEP: Yes	17	2	15	-1.158	-0.888
MATHEMATICS	IEP: No	134	52	86	-0.439	0.213
MATHEMATICS	IEP: Yes	17	0	15	-1.260	-0.485
CONTENT_AREA	Group	TotalN	ProfN	GrowthN	Z_Status	Z_Growth

Either of the demographic aggregation and combination code chunks above can be run for each of the `SGP_Cnd*` growth fields. At that point, we can then combine those objects in a wide format (similar to what was done for the `composite_summary` object - this would require re-naming the aggregate variables), stacked into a long format (with an added "Condition" variable for each table - probably what I would do) or written to separate .csv files as described in the specification doc.

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 4.2.1 (2022-06-23)
os	macOS Ventura 13.0.1
system	x86_64, darwin17.0
ui	X11
language	(EN)
collate	en_US.UTF-8
ctype	en_US.UTF-8
tz	America/Denver
date	2022-11-11
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	1.14.5	Github (Rdatatable/data.table@e956716)	11/09/2022
SGP	2.0-1.1	Github (centerforassessment/SGP@d8c59ae)	10/27/2022
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** *(4.0.4)*, **bit64** *(4.0.5)*, **blob** *(1.2.3)*, **bookdown** *(0.30)*, **boot** *(1.3-28)*, **brio** *(1.1.3)*, **broom** *(1.0.1)*, **bslib** *(0.4.1)*

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** *(3.4.1)*, **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** *(0.6.30)*, **doParallel** *(1.0.17)*, **doRNG** *(1.8.2)*, **dplyr** *(1.0.10)*

e1071 *(1.7-12)*, **ellipsis** *(0.3.2)*, **equate** *(2.0.8)*, **evaluate** *(0.18)*

fansi *(1.0.3)*, **fastmap** *(1.1.0)*, **foreach** *(1.5.2)*, **foreign** *(0.8-83)*, **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** *(3.9.3)*

HDInterval *(0.2.2)*, **highr** *(0.9)*, **Hmisc** *(4.7-1)*, **htmlTable** *(2.4.1)*, **htmltools** *(0.5.3)*, **htmlwidgets** *(1.5.4)*, **httpuv** *(1.6.6)*, **httr** *(1.4.4)*

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

jpeg *(0.1-9)*, **jquerylib** *(0.1.4)*, **jsonlite** *(1.8.3)*

knitr *(1.40)*

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** *(1.5-1)*, **MatrixModels** *(0.5-1)*, **matrixStats** *(0.62.0)*, **memoise** *(2.0.1)*, **mice** *(3.14.8)*, **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 *(0.19.1)*, **pillar** *(1.8.1)*, **pkgbuild** *(1.3.1)*, **pkgconfig** *(2.0.3)*, **pkgload** *(1.3.1)*, **plotly** *(4.10.1)*, **png** *(0.1-7)*, **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** *(0.3.5)*

quantreg *(5.94)*

R.methodsS3 *(1.8.2)*, **R.oo** *(1.25.0)*, **R.utils** *(2.12.1)*, **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** *(2.18)*, **rngtools** *(1.5.2)*, **robustbase** *(0.95-0)*, **rpart** *(4.1.19)*, **RSQLite** *(2.2.18)*, **rstudioapi** *(0.14)*

sass *(0.4.2)*, **scales** *(1.2.1)*, **sessioninfo** *(1.2.2)*, **shiny** *(1.7.3)*, **sn** *(2.1.0)*, **sp** *(1.5-1)*, **SparseM** *(1.81)*, **stringi** *(1.7.8)*, **stringr** *(1.4.1)*, **survival** *(3.4-0)*, **svglite** *(2.1.0)*, **systemfonts** *(1.0.4)*

testthat *(3.1.5)*, **tibble** *(3.1.8)*, **tidyr** *(1.2.1)*, **tidyselect** *(1.2.0)*, **toOrdinal** *(1.3-0.0)*

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

vcd *(1.4-10)*, **vctrs** *(0.5.0)*, **VIM** *(6.2.2)*, **viridisLite** *(0.4.1)*

webshot2 *(0.1.0)*, **websocket** *(1.4.1)*, **withr** *(2.5.0)*

xaringan *(0.27)*, **xfun** *(0.34)*, **xtable** *(1.8-4)*

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

zoo *(1.8-11)*