STUDENT GROWTH PERCENTILES CALCULATIONS

RE-EVALUATING ANNUAL CENSUS TESTING

NOVEMBER 2022

ADAM VAN IWAARDEN
LESLIE KENG
EMMA KLUGMAN, HARVARD UNIVERSITY

Center for Assessment

National Center for the Improvement
of Educational Assessment
Dover, NH

STUDENT GROWTH PERCENTILES CALCULATIONS

RE-EVALUATING ANNUAL CENSUS TESTING

Submitted to:

Bill Gates and the Walton Family
The Gates-Walton Foundations
November 2022

Author(s):

Adam Van Iwaarden,
Leslie Keng and
Emma Klugman

Acknowledgements:

We want to thank the State A Department of Education for their help in obtaining the data for this report, the **Gates-Walton Foundations** for their ongoing support and the Walmart greeter who told us to never (EVER) use Windows OS.

Suggested Citation:

The National Center for the Improvement of Educational Assessment (2022). Student Growth Percentiles Calculations. Submitted to The Gates-Walton Foundations, Anywhere, U.S.A..

REPORT CONTENTS

Executive summary 1

Data cleaning and preparation 2

- Load packages and custom functions. 2
- General data setup and cleaning 2
- Save data 3
- Summary and notes 3

Student Growth Percentiles Analysis 5

- Load SGP package and modify `SGPstateData` 5
- Simulation Condition 0 5
- Simulation Condition 1b 7
- Simulation Condition 1c 9
- Simulation Condition 2 11
- Simulation Condition 3 13
- Save data 14

Growth and Achievement Aggregations 15

- Additional variables for aggregated results 16
- Condition specific summary tables - all students by schools 16
- Achievement Improvement Aggregations 21
- School level aggregations by demographics 24
- A single file with all aggregations 26
- Summary and notes 27

References 28

EXECUTIVE SUMMARY

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.

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")
```

LOAD PACKAGES AND CUSTOM FUNCTIONS.

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

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

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 < 2023]
```

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",
    "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", "FRL", 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
]
```

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"]
```

SAVE DATA

```
fwrite(State_A_Data_LONG,
    file = "Data/Cleaned_Data/Student_LongTestData_State_A_2016-2022_AVI.csv"
)
```

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.

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.

LOAD SGP PACKAGE AND MODIFY SGPSTATEDATA

The SGP package is required for all growth percentile analyses.

```
require(SGP)
require(data.table)
```

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
```

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

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.

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
    )
```

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("..")
```

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.

```
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
]

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

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.

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
    )
```

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,
        sgp.projections.lagged.baseline = FALSE,
        simulate.sgps = FALSE,
        parallel.config = list(
            BACKEND = "PARALLEL",
            WORKERS = workers
        )
    )
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
]

Condition_1b_CoefMatrices <- copy(State_A_SGP@SGP[["Coefficient_Matrices"]])
save(Condition_1b_CoefMatrices,
     file = "./Condition_1b/Condition_1b_CoefMatrices.rda")
```

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.

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
    )
```

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 = workers
        )
    )
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
]

Condition_1c_CoefMatrices <- copy(State_A_SGP@SGP[["Coefficient_Matrices"]])
save(Condition_1c_CoefMatrices,
     file = "./Condition_1c/Condition_1c_CoefMatrices.rda")
```

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.

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
    )
```

CALCULATE CONDITION 2 SGPS

The call to theabcSGP 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.lagged.baseline = FALSE,
        simulate.sgps = FALSE,
        parallel.config = list(
            BACKEND = "PARALLEL",
            WORKERS = workers
        )
    )
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
]

Condition_2_CoefMatrices <- copy(State_A_SGP@SGP[["Coefficient_Matrices"]])
save(Condition_2_CoefMatrices,
     file = "./Condition_2/Condition_2_CoefMatrices.rda")
```

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.panel.years = c("2017", "2019"),
        sgp.grade.sequences = list(
            c("3", "5"), #  Elementary Grades
            c("6", "8")  #  Middle Grades
        )
    )
)
```

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
]
```

SAVE DATA

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

save("State_A_Data_LONG", file = "Data/Student_Growth/State_A_Data_LONG.rda")

fwrite(State_A_Data_LONG,
    file = "Data/Student_Growth/Student_LongTestData_State_A_2016-2019_AVI.csv"
)
```

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

[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

    - 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

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[2].

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

CONDITION SPECIFIC SUMMARY TABLES - ALL STUDENTS BY SCHOOLS

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…

[2] The unstandardized `SCALE_SCORE` variable is used in the SGP calculations.

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),
          MeanScore = round(mean(Z_SCORE, na.rm = TRUE), 2),
          PcntProf = round(mean(PROFICIENCY, na.rm = TRUE), 3)*100,
          StatusZ = round(mean(Z_PROFICIENCY, na.rm = TRUE), 3),
          GrowthZ = round(mean(qnorm(SGP_Cnd_0/100), na.rm = TRUE), 3)
        ),
        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 and my additional 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()
)

#   Write files for each year separately as per the specifications:
fwrite(sch_summary_cnd_0w[YEAR == 2018][, YEAR := NULL],
    file = "Data/School_Summaries/School_Condition_0_State_A_2018_AVI.csv"
)
fwrite(sch_summary_cnd_0w[YEAR == 2019][, YEAR := NULL],
    file = "Data/School_Summaries/School_Condition_0_State_A_2019_AVI.csv"
)
```

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

```
##   SchoolID YEAR CONTENT_AREA TotalN ProfN GrowthN  MGP MeanScore PcntProf StatusZ Growt
##       1001 2018          ELA    120   109      73 49.7      0.62     90.8   0.511  -0.0
##       1001 2018  MATHEMATICS    120   107      73 61.5      0.79     89.2   0.553   0.3
##       1001 2019          ELA    162   146      90 54.0      0.66     90.1   0.501   0.1
##       1001 2019  MATHEMATICS    162   145      90 55.1      0.77     89.5   0.565   0.1
##       3801 2018          ELA    170    44     103 43.4     -0.89     25.9  -0.870  -0.2
##       3801 2018  MATHEMATICS    170    31     103 40.7     -0.91     18.2  -0.855  -0.3
##       3801 2019          ELA    151    67     101 53.4     -0.61     44.4  -0.467   0.1
##       3801 2019  MATHEMATICS    151    52     101 54.3     -0.66     34.4  -0.531   0.1
```

At some point we will 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:
```

```
grep("PcntProf|Mean_", names(composite_summary), value = TRUE) :=
      NULL
]

#  School No. '1001' - changes in Growth N count
composite_summary[SchoolID == 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[SchoolID == 1001,
    c("YEAR", "CONTENT_AREA",
# All relevant aggregations at once:
# sort(grep("GrowthZ|StatusZ", names(composite_summary), value = TRUE))
      grep("GrowthZ", names(composite_summary), value = TRUE) # just z-growth
    ), with = FALSE
]
```

```
##  YEAR CONTENT_AREA GrowthZ GrowthZ_1b GrowthZ_1c GrowthZ_2 GrowthZ_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[SchoolID == 1001,
    c("YEAR", "CONTENT_AREA",
      grep("StatusZ", names(composite_summary), value = TRUE)
    ), with = FALSE
]
```

```
##  YEAR CONTENT_AREA StatusZ StatusZ_1b StatusZ_1c StatusZ_2 StatusZ_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
```

Another way to combine all the conditions is to create a long summary table with the same summary variable names for each condition and an added column indicating the condition they belong to.

Because creating summary tables for all growth conditions only requires 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, and we are going to be doing this numerous times, we will create a custom function to create these tables, rather than copying the code for each use case. This will also help in the demographic level summaries, which require the addition of the variable of interest into the `keyby` argument of the `data.table` aggregation.

```
schoolAggrGator =
  function(
    data_table,
    growth.var,
    groups = c("SchoolID", "YEAR", "CONTENT_AREA")
  ) {
    aggr.names <-
      c("TotalN", "ProfN", "GrowthN", "MGP",
        "GrowthZ", "MeanScore", "PcntProf", "StatusZ")
    frmla <-
      paste0(
        paste(groups %w/o% "CONTENT_AREA", collapse = " + "),
        " ~ CONTENT_AREA"
      ) |> as.formula()
    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),
        GrowthZ = round(mean(qnorm(get(growth.var)/100), na.rm = TRUE), 3),
        MeanScore = round(mean(Z_SCORE, na.rm = TRUE), 2),
        PcntProf = round(mean(PROFICIENCY, na.rm = TRUE), 3)*100,
        StatusZ = round(mean(Z_PROFICIENCY, na.rm = TRUE), 3)
      ),
      keyby = groups
    ] |>
      dcast(
        formula = frmla,
        sep = "..",
        value.var = aggr.names
      ) |>
        setnames(
c(groups %w/o% "CONTENT_AREA",
              paste(
                rep(c("ELA", "Math"), length(aggr.names)),
                rep(aggr.names, each = 2),
                sep = "_"
            )),
            \(f) {
              tmp.name <-
                strsplit(f, "[.][.]")[[1]] |>
                  rev() |> paste(collapse = "_")
              gsub("MATHEMATICS", "Math", tmp.name)
            }
          ) |> unlist()
        )
  }
```

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_GrowthZ)])
```

```
## , , YEAR = 2018
##
##        Condition
## is.na     0  1b  1c   2   3
##   FALSE 231 231 149 231 231
##   TRUE    0   0  82   0   0
##
## , , YEAR = 2019
##
##        Condition
## is.na     0  1b  1c   2   3
##   FALSE 231 231 149 231 231
##   TRUE    0   0  82   0   0
```

```
fwrite(school_aggregation_all_students,
    file = "Data/School_Summaries/School_Condition_AllGrowth_AllStudents_State_A_AllYears
)
```

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,
          MeanScore = round(mean(Z_SCORE, na.rm = TRUE), 2),
          StatusZ = round(mean(Z_PROFICIENCY, na.rm = TRUE), 3)
        ),
        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("TotalN", "MeanScore", "StatusZ"),
    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 changes (current year minus 1 year lag)
sch_summary_cnd_1a[,
    TotalN_Change := TotalN - TotalN_LAG_1
][,
    MeanScore_Change := MeanScore - MeanScore_LAG_1
][,
    StatusZ_Change := StatusZ - StatusZ_LAG_1
]
```

Here is our example school’s improvement numbers

```
sch_summary_cnd_1a[SchoolID == 1001,
    c(key(sch_summary_cnd_1a)[-1],
      grep("Change", names(sch_summary_cnd_1a), value = TRUE)
), with = FALSE
]
```

```
##   CONTENT_AREA YEAR GRADE TotalN_Change MeanScore_Change StatusZ_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
```

Per the specifications doc, here is how we could reshape this summary table and output by year.

```
#  Reshape by subject
sch_summary_cnd_1a <-
    dcast(
        data = sch_summary_cnd_1a,
        formula = SchoolID + YEAR + GRADE ~ CONTENT_AREA,
        sep = "..",
        value.var = names(sch_summary_cnd_1a) %w/o% key(sch_summary_cnd_1a)
    )
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()
)

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"
)
```

One thought before leaving the "school improvement" indicator section is that the issue of "regression to the mean" should be considered. There is a strong relationship between current/prior scores and change scores that may require a correction:

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

```
##                       Math_MeanScore_Change Math_MeanScore
## Math_MeanScore_Change                 1.000          0.493
## Math_MeanScore                        0.493          1.000
```

```
cor(
  sch_summary_cnd_1a[, Math_MeanScore_LAG_1, Math_MeanScore_Change],
  use = 'na.or.complete'
) |> round(3)
```

```
##                       Math_MeanScore_Change Math_MeanScore_LAG_1
## Math_MeanScore_Change                 1.000               -0.517
## Math_MeanScore_LAG_1                 -0.517                1.000
```

```
cor(
  sch_summary_cnd_1a[, ELA_MeanScore_LAG_1, ELA_MeanScore_Change],
  use = 'na.or.complete'
) |> round(3)
```

```
##                      ELA_MeanScore_Change ELA_MeanScore_LAG_1
## ELA_MeanScore_Change                1.000              -0.466
## ELA_MeanScore_LAG_1                -0.466               1.000
```

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.

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

An example of our function used for (dis)aggregation by student demographics (Economic Disadvantage) for condition 0 results:

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

```
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("SchoolID", "YEAR", "CONTENT_AREA", "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("SchoolID", "YEAR", "CONTENT_AREA", "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("SchoolID", "YEAR", "CONTENT_AREA", "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("SchoolID", "YEAR", "CONTENT_AREA", "SWD")
        ) |> setnames("SWD", "Group")
      )
    )
```

Here a subset of the output from the example school:

```
demog_cond_0[
  SchoolID == 1001 & YEAR == 2019
][,
  grep("YEAR|SchoolID|MGP|MeanScore", names(demog_cond_0), value = TRUE) := NULL
][]
```

```
##        Group ELA_TotalN Math_TotalN ELA_ProfN Math_ProfN ELA_GrowthN Math_-
```

```
GrowthN ELA_GrowthZ Math_GrowthZ ELA_PcntProf Math_PcntProf ELA_StatusZ Math_StatusZ
##        Asian         30          30        27         23          21           21
##        Black         12          12        10         10           8            8
##     Hispanic         30          30        24         24          18           18
##  Multiracial         10          10         9         10           7            7
##        White         80          80        76         78          36           36
##      FRL: No        146         146       133        132          83           83
##     FRL: Yes         16          16        13         13           7            7
##       EL: No        159         159       143        142          89           89
##      EL: Yes          3           3         3          3           1            1
##      SWD: No        152         152       146        143          83           83
##     SWD: Yes         10          10         0          2           7            7
```

Another example of how to combine the aggregations uses calls to the function via `lapply` over the demographic variables and piping the resulting list to `rbindlist`.

```
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("SchoolID", "YEAR", "CONTENT_AREA", f)
      )[, Condition := "0"] |> setnames(f, "Group")
    }
  ) |> rbindlist()
```

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.

The example below shows how to create a stacked long format version with all conditions that contain growth.

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):

```
school_aggregation_all_students[, Group := "All Students"]
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_AllYear
)
```

SUMMARY AND NOTES

- 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.
- 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.*sgp.io.