ADEP WORKING PAPER SERIES

Statistical Language Performance at the U.S. Census Bureau: An Example Using Full Count 1990 Decennial Census Data

Cecile Murray
U.S. Census Bureau

Katie Genadek
U.S. Census Bureau

Krista Chan
U.S. Census Bureau

Virginia Gwengi
U.S. Census Bureau

Christian Moscardi
U.S. Census Bureau

Working Paper 2021-01
April 2021

Associate Directorate for Economic Programs
U.S. Census Bureau
Washington DC 20233

Statistical Language Performance at the U.S. Census Bureau: An Example Using Full Count 1990 Decennial Census Data

Cecile Murray, U.S. Census Bureau
*Katie Genadek, U.S. Census Bureau
Krista Chan, U.S. Census Bureau
Virginia Gwengi, U.S. Census Bureau
Christian Moscardi, U.S. Census Bureau
*Contact Author for questions

ADEP Working Paper 2021-01
April 2021

Abstract

Large-scale data processing and analysis is not a new challenge for the U.S. Census Bureau, but the number of statistical programming languages and tools available to perform such work has expanded in recent years. We evaluate how statistical programming languages perform on a common data management task within the Census's Bureau's high-performance computing cluster. Specifically, we develop Python, SAS, Stata, and R scripts that merge the person, household, and geographic microdata from the full-count 1990 Census microdata files. We then use these merged data to perform basic analyses such as counting the number of individuals per household and calculating the average household size for every county in the U.S. We compare the different language implementations of these scripts based on runtime for each task. We find that there is wide variation between languages in runtime, and the speed of the programming language depends most heavily on the file format of the input data file. As the data files at the Census Bureau are stored in SAS format, SAS was the fastest programing language.

Keywords: Programming, decennial census
JEL Classification Codes: C80, C88

I. Introduction

Researchers at the U.S. Census Bureau use large demographic and economic datasets in a restricted use environment on approved research projects. The computing and storage environment, known as the Integrated Research Environment (IRE), is a cluster of Linux servers for researchers to access and use Census Bureau data. The IRE has more than fifteen statistical applications available for researchers to manage and analyze data using the computing resources in this space. All statistical jobs are scheduled through a job queuing system in order to best disperse and use resources in the IRE across all users.

Researchers generally choose statistical programming languages based on the available features, their familiarity with the program or language, the program used by other team members, and their sense of typical runtimes. Such languages vary in how they store and use data. While some of these differences are well-known, it was not clear which statistical programing language would be the fastest at performing basic manipulation of large, long decennial census data in the Census Bureau's IRE. The decennial census microdata file available to researchers generally includes responses for all individuals residing in the U.S. at the time of the census. - The 1990 Decennial Census produced more than 250 million individual responses.

We developed scripts to benchmark how long it took to perform the same set of analytical tasks in R, Python, SAS, and Stata using a subset of the 1990 Decennial Census short form microdata. While there are a host of quantitative and qualitative factors to consider determining the best program for this type of job, runtime is a particularly important factor for most researchers. The datasets at the Census Bureau are stored in a SAS format, so we performed these tests starting with the data in SAS formant, but we also converted all files to CSV and performed the tasks starting with a CSV file as well.

The results of our test suggest that there is wide variation between languages in runtime, and the speed of the programming language depends most heavily on the file format of the input data file. As the Census Bureau stores all data in the SAS data format, SAS has the shortest run time. When storing the files as CSVs, the runtimes were similar across programing languages.

II. Data

The Census Bureau makes anonymized decennial census microdata available to researchers on approved projects in the restricted use IRE environment, which can be accessed through the Federal Statistical Research Data Centers (FSRDCs). The decennial censuses are household surveys, and in 1990 the decennial census included a “short form” questionnaire with basic information that all respondents filled out, and a “long form” went to about 20% of the population. The short form includes basic demographic information on age, race, sex, and relationship to household head. The microdata also include information on the housing unit and geographic information. For each year, the data are stored by state, and there are three data files for each state: a person file, a household file, and a geography file. Respectively, they contain information about individuals, the households in which those individuals lived, and the geographic area where those households were located.

We use data from the 1990 Decennial Census full-count short form. We chose three states, Delaware (DE), Nevada (NV), and Kansas (KS), as small, medium, and large states. In 1990, the census counted 666,168 people in DE, 1,201,833 people in NV, and 2,477,574 people in KS. Thus, each successively larger state has about twice as many observations as the last, which allowed us to measure how performance changes with the number of rows in a dataset. These data are not particularly wide, with 77 variables in the geography files, 63 variables in the household files, and 35 variables in the person files. Table 1 shows the file sizes for each of the files used in the companions.

Table 1. Size of 1990 full-count microdata files for select states in mebibytes

	.sas7bdat			**.csv**		
	DE	NV	KS	DE	NV	KS
Geography	1.3	7.5	38.9	6.5	13.3	69.6
Household	45.8	88.7	177.1	97.7	176.6	351.8
Person	89.2	183.2	372.9	155.7	282.4	578.1

Notes: All file sizes were approved for release by the U.S. Census Bureau Disclosure Review Board, authorization number CBDRB-FY21-ERD002-014.

III. Methodology

To compare the four programing languages, we simulated typical data loading and preparation steps that researchers perform in order to use the decennial census data and computed some basic analytical measures researchers might perform to ensure the data is ready for analysis. Comparisons were done on the same machine, IRE, with Linux Red Hat v.6.10

operating system, with 250 GB of memory.[1] We wrote one script in each language that would execute three general conceptual tasks: read in data, merge it together, and produce a few summary counts. There are myriad ways to approach these tasks and optimize code in any language. To keep the exercise simple, we focused on writing scripts that performed these conceptual steps in this order using an approach that a fluent user of a given language might take.[2]

Table 2. Machine Specification

Specification	IRE
Processor	Intel Xeon CPU E5-2698 v3 @2.30GHz
HD size	926TB
HD free space	89TB
RAM	250GB
OS	Linux Red Hat v 6.10

Table 3. Software Specification

Software	Version
SAS	9.4
Stata	16.1
R	3.5.2
Python	3.6.7

In our first step, “Read,” we needed to read the selected data sets into the statistical program. As mentioned, the default storage format for these files is the proprietary SAS format (extension .sas7bdat), but we also produced CSV versions of each file for testing. To read SAS

[1] We submitted each script as a separate job to the IRE resource scheduler, which queues jobs until the requested resources become available. Because IRE is a shared resource, other users' behavior could affect script runtime. To minimize the effect of this interference, we submitted all jobs to the queue one at the same time on a weekday evening after normal business hours. We took the fact that jobs moved almost immediately from "queued" to "running" as an indication that our chosen window was not a high-usage time.

[2] For example, to read in data from CSV in Python and R, we used read_csv() methods from the pandas and readr packages, respectively. While more optimized methods exist, a typical user would likely use these more generic ones as a first pass. The full implementation of all scripts is available on GitHub here.

files in R and Python, we used the read_sas methods from the haven and pandas packages, respectively. To read the files in Stata, the script looped through each SAS file, ingesting it with the "import sas" command available in Stata 16, and saved a Stata.dta format file that Stata could read directly. Second, we merged the person, household, and geographic datasets in each state using unique household and geographic identifiers to prepare the data for analysis, as is often necessary to use these data for statistical analyses. Thus the result of our second step, "Merge," is a flat file ready for analysis. Finally, in our third step, "Count," we performed some simple analytical checks to ensure the datasets are merged properly and prepared for more complex analyses. Specifically, we counted the number of people in each household and computed the average number of people per household by state, and we also computed the average age of individuals in each county.

We measured runtime using two different strategies for all four languages. Our first approach measures total runtime using a Python wrapper that repeatedly runs the Linux "ps" command to identify the status of a running process.[3] Each script maps to one process, so we can use this tool to measure precisely how long each script took from start to finish. However, this approach does not allow us to measure how long each phase of the script takes. Consequently, we also developed a second approach: we simply logged print statements and timestamps at key points in each script. We then parsed this text output to produce the runtime data. While this method allows us to break out task runtimes, its simplicity could come with reduced accuracy because print statements do not necessarily map perfectly to the order in which scripts are executed "under the hood." Overall runtimes were similar but not identical for both methods, and we rely on the more precise approach wherever possible.

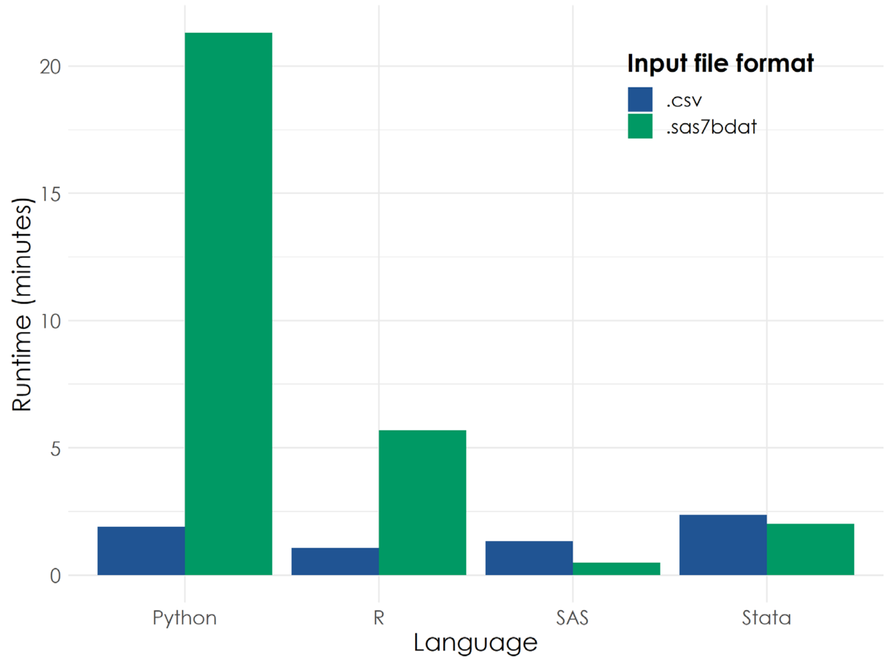
III. Results

Figure 1 summarizes our overall runtime results. When using a SAS file format, SAS was fastest and Python was slowest by a wide margin. Python took more than 20 minutes to perform all three steps, while R took a little over 5 minutes, Stata took about 2 minutes, and SAS performed the whole exercise in just under 30 seconds. When the file format was CSV, R was the fastest, but the difference between runtimes narrowed considerably. The R script ran the fastest, finishing in just over one minute, while Stata ran the slowest, taking about 2.3 minutes.

[3] We use the open-source syrupy package.

SAS runtimes lengthened substantially when reading from CSV than when reading from its native format, but even then, its performance was still the second-quickest of the four languages. The Python runtime has the most dramatic difference when comparing the CSV to native SAS data ingest and analyses, with the exercise taking 10 times longer when Python read in SAS data.

Figure 1. Total runtime by input file type and language

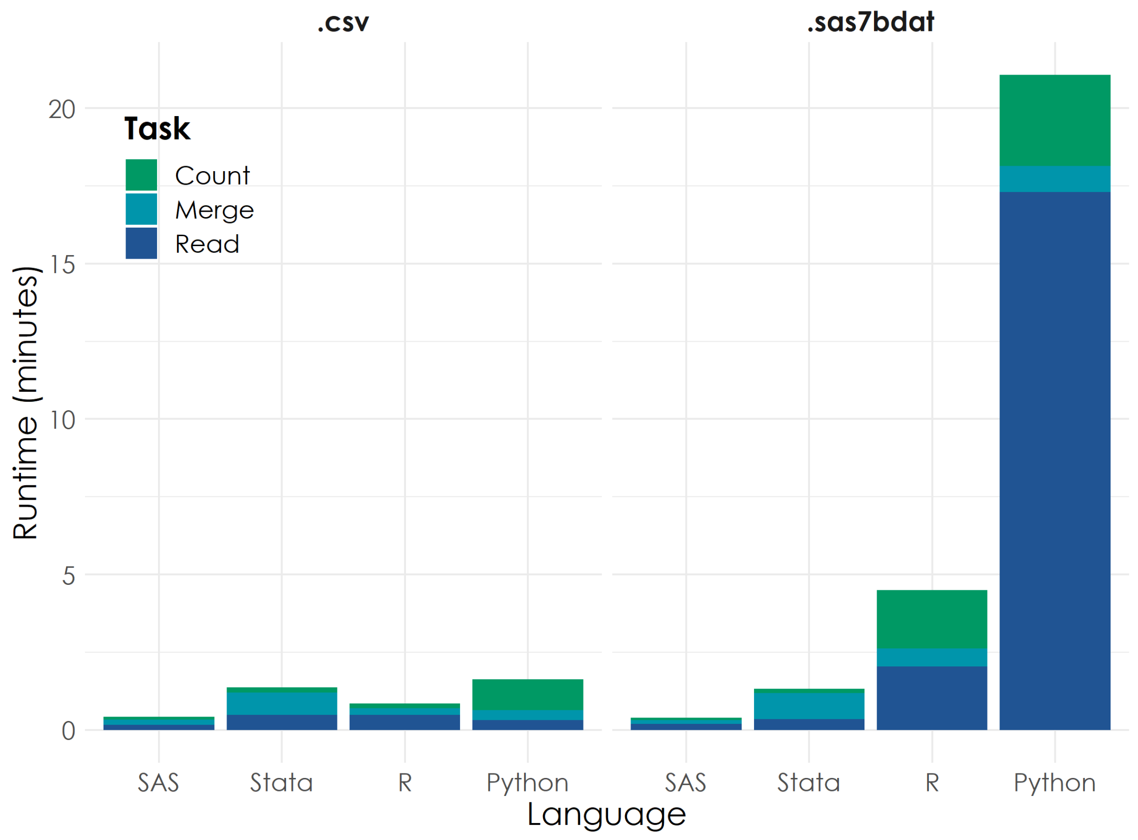


Notes: Runtime for each program was calculated for full task described in text using the 1990 decennial census. Results were approved for release by the U.S. Census Bureau Disclosure Review Board, authorization number CBDRB-FY21-ERD002-014.

The slow run times for Python and R when using the native SAS data suggest that the programs are inefficient when reading in the .sas7bdat file. Comparing the timing of each task in the exercise, Figure 2 shows the approximate time spent on each of the three conceptual stages.[4] As expected, the read step with the SAS input file was the slowest for Python.

Figure 2. Runtime by language by task, by format

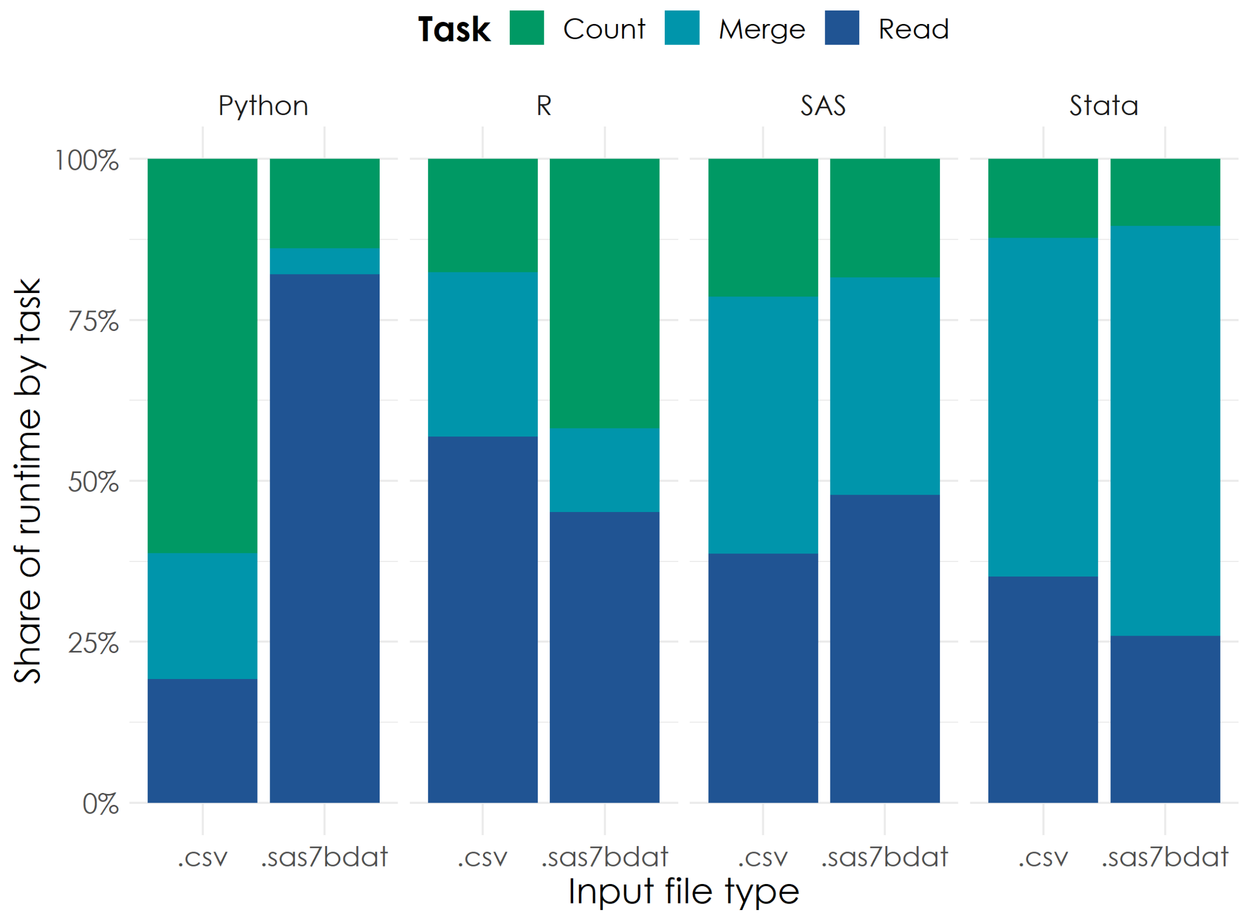
[4] Note that total runtime shown in Figure 1 is not comparable to the sum of times for each task in Figure 2. The former incorporates the full time for the entire process of a job based on a logging utility (Linux ps), whereas the latter is derived from timestamps logged from within the script itself.



Notes: Runtime for each program was calculated for full task described in text using the 1990 decennial census. Results were approved for release by the U.S. Census Bureau Disclosure Review Board, authorization number CBDRB-FY21-ERD002-014.

Figure 3 shows the percent share of runtime each language spent on each task using each file format. Not only did the CSV format reduce the absolute amount of time that Python spent reading in data, it also reduced the share of time it spent on that part of the script. By contrast, while the absolute read time for R went down from two minutes to just under 30 seconds, the relative time it spent on the read step increased. Figure 3 also illustrates how a key difference in how Stata operates might affect runtime. We are holding only one dataset in memory at a time in Stata, so Stata merges implicitly require reading data from disk. This feature might explain why Stata spends relatively more time on the merge task than any of the other languages. However, there is a newly available "frames" feature in Stata, which allows Stata to hold multiple tables in memory at once, might avoid this issue. Finally, Figure 3 reveals a puzzling result: runtimes for R and Python during the "count" stage were longer with the SAS input file than with the CSV file. Since the operations during this phase should be being executed in memory, it is not clear why the input file format would make a difference. One possible explanation for this result is that Python and R could be performing “lazy evaluation,” in which statements are evaluated at the moment in a script when their results are needed rather than when they are first called; our print statement-based logging strategy would not capture this complexity.

Figure 3. Percent of runtime by task, by language and input file format



Notes: Share of runtime for each program was calculated for full task described in text using the 1990 decennial census. Results were approved for release by the U.S. Census Bureau Disclosure Review Board, authorization number CBDRB-FY21-ERD002-014.

Finally, Tables 4 presents the runtimes for each task for each language by state for the CSV and SAS file formats (.csv and .sas7bdat), respectively. These runtimes are an indicator of how a language’s performance at a task deteriorates as dataset size increases. Given that the number of observations in the medium state (NV) is approximately double that in the small state (DE), one might expect the runtime for the medium state to roughly double as well. Similarly, since the number of observations in the large state (KS) is also roughly twice that of the medium state (NV), one might have the analogous expectation for their respective runtimes. Such a linear increase in runtime would indicate that a language's performance remains steady as dataset size increases, while a greater than linear increase would indicate potential problems at larger scales.

Table 3 present a mixed picture of language performance. For both file formats, total runtimes for all languages for NV are less than twice what they are for DE, indicating that scaling the dataset from roughly 650,000 rows to 1.2 million rows did not have a noticeable effect on performance. However, moving from NV to KS, these total runtimes all at least double, suggesting that scale may begin affecting performance between 1.2 and 2.4 million observations. Interestingly, comparing the increase in runtime from DE to NV to that from NV to KS shows that SAS slowed down more than the other languages when working with the largest dataset for both input formats.

Table 4. Runtime by language and state dataset size, SAS vs. CSV format

Language	Input format	DE runtime (sec)	NV runtime (sec)	KS runtime (sec)	Runtime scale, DE-NV	Runtime scale, NV-KS
Python	.csv	13.9	26.8	57.3	1.9	2.1
R	.csv	9.0	14.0	28.0	1.6	2.0
SAS	.csv	13.1	16.8	44.0	1.3	2.6
Stata	.csv	15.7	21.2	45.0	1.3	2.1
Python	.sas7bdat	184.9	330.0	749.3	1.8	2.3
R	.sas7bdat	294.5	522.7	1192.2	1.8	2.3
SAS	.sas7bdat	5.5	6.1	12.0	1.1	2.0
Stata	.sas7bdat	12.3	21.3	45.5	1.7	2.1

Notes: Runtime for each program was calculated using the 1990 decennial census. Results were approved for release by the U.S. Census Bureau Disclosure Review Board, authorization number CBDRB-FY21-ERD002-014.

IV. Conclusion

The primary practical conclusion from this exercise is that the native file format has a huge impact on how quickly open-source scripts, Python and R, will run. While the CSV files make the run times comparable across all programing languages, the Census Bureau stores all data files as SAS files. Thus, the conversion to CSV files is an important step in the data analysis process. As far as which statistical programming language is "best," this exercise offers evidence that using SAS to perform initial data management on native SAS files in the Census Bureau's computing environment is the fastest option. If you are working in Python or R, runtimes will be much shorter if you convert the original SAS data files to CSV and store them as CSV files.

The fact that the original file format has a large impact on programming language performance is also an important consideration regarding the adoption of open-source tools at the Census Bureau. Slow data input due to storage file format choice could represent an important barrier to broader use of such tools. Moreover, this paper has compared one proprietary format versus one open-source data format, but there are several additional open-source data formats that are optimized for large-scale data analysis. For example, the Parquet storage format allows fast reads at large scale when users only need access to a subset of columns. Exploring whether open-source formats reduce the read times for all languages without incurring substantially greater storage costs could be useful for Census Bureau employees and researchers.

The datasets used for this exercise were not particularly large compared to some of the research done at the Census Bureau. Often researchers at the Census Bureau are using full-count census data (248 million respondents in 2019) or large administrative records. However, these results provide insights into using larger data with various statistical languages, and this is something that could be measured in the future. Moreover, the decennial short form data is long with its large number of observations, but it does not have many variables compared to other data sets (wide data sets). This exercise could also be performed using wide data sets. In addition to testing these scripts on larger data, we also plan to explore parallelizing our code to more fully leverage the high-performance cluster environment in which we work, and to see how these results hold up for more complicated analytical tasks. Runtime is also just one measure of a statistical language, and there are many trade-offs to consider in addition to speed. We plan to measure how memory usage patterns vary by language. Finally, we will be able to use the same toolset to benchmark the performance of different functions and analysis implementations, which will allow us to identify opportunities to optimize our work.