**The PIAAC Variable Finder: An interactive Shiny app for cleaning, interpreting and analyzing Programme for the International Assessment of** **Adult Competencies data**

Nate Breznau, [breznau.nate@gmail.com](mailto:breznau.nate@gmail.com), [](https://orcid.org/0000-0003-4983-3137)

German Institute for Adult Education – Leibniz Centre for Lifelong Learning

Abstract

Keywords: PIAAC, data analysis, shiny, text scraping, app development

Funding: This project was partly funded from a German Science Foundation (Deutsches Forschungsgemeinschaft) project number XXXXXX

The most widely used data for comparative research on adult skills in rich and middle income countries is the *Programme for the International Assessment of Adult Competencies* (PIAAC). The various local and international surveys fielded as part of this Organisation for Economic Co-Operation and Development (OECD) project measure human capital. Specifically literacy, numeracy and problem-solving in technology-rich environments in adults aged 16-65, with some countries sampling from older populations. With the release of PIAAC Cycle 2 in 2025, there are now two waves of data from 24 countries with an additional 19 that participated in one of the two Cycles[[1]](#footnote-1). Since 2018 researchers produced over 150 publications per year using PIAAC data in some form (Maehler et al., 2025).

The PIAAC survey is large and complex. It has sophisticated item-response methods to predict adult skill scores, population weighting, and a detailed background questionnaire asking about education, work, demographics, behaviors and attitudes. The data is available in open access Public Use Files (PUFs), requiring only name, email, and usage goals prior to downloading[[2]](#footnote-2). Changes between the two Cycles of PIAAC increase the complexity of this massive data project. To fully understand all aspects of the data and the differences between Cycles, a user needs at least four separate documents each in a different format. For data analysts, it can be quite challenging and confusing to have some or all of these documents open simultaneously, and to cycle through them searching for variables or other information.

In my own attempts to understand and analyze PIAAC data, it became clear to me that these diverse documents could be integrated. They all contain stable variable names. This stable “ID” would allow for compilation of a dataset with metadata linked to each variable. I scraped the documents and combined their data, then built an interactive app for searching through the combined data that I could run locally to support my data cleaning, wrangling, analysis and visualization efforts. This allowed me to work much more efficiently, for example to quickly find any word that was used in the questionnaire during the survey, identify response categories and their values, identify if a variable was a trend variable between the Cycles or not, and check if a variable contained non-missing values (i.e., that it was not withheld to protect the identities of survey respondents).

I relied consistently on Gen AI to develop the app, in this case ChatGPT about 95% of the time. It was crucial that I had strong pre-existing skills in the R statistical programming language, the Shiny package, and basic knowledge of working with text and scraping. The app development was so successful that it seemed prudent to make it a public resource in the name of open science so that others could avoid the challenges and frustrations I faced working with PIAAC data. This is not to say that the many teams of researchers and administrators producing PIAAC documentation failed in any way. They overcame monumental hurdles in producing documentation with diverse purposes. It is only to say that I saw an opportunity for a scientific tool to improve on their work. Thus, this paper details both, A) the development of that tool – a demonstration of how any researcher with foundational R skills could build something similar, and B) a user’s guide to the tool, which I named the “PIAAC Variable Finder”.

Readers can use the interactive app via a Web browser[[3]](#footnote-3), fork the entire workflow to construct the app[build new repo], or simply download a zipped folder to run the app locally[xxx]. This open source approach allows users to expand and customize the app for their purposes. For example, I only scraped the British English and German versions of the background questionnaire, but more languages could easily be incorporated.

## Building an interactive app, circa 2025

Background knowledge was necessary to fuel my idea of having an interactive app. First, basic knowledge of how document text scraping works, and software that could perform this scraping. In this case I know R (R Core Team, 2024), and use it in the R Studio IDE (Posit Team, 2025) which has several advantages: It was specifically designed to run R statistical software and is compatible with Git allowing for easy version control and open source sharing. The programming language R is also ideal because it was developed by scientists.

Second, knowledge of the Shiny package was necessary. Shiny is a powerful set of tools designed to enable interactive app development and deployment with minimal data science knowledge. I used Shiny tools in R previously, and therefore had a strong understanding of how it works, and what it can and cannot do, in addition to how much (how little) coding is required to produce powerful interactive apps with it (Chang et al., 2025).

Third, based on my previous knowledge and experiences with Gen AI, I had enough motivation to build an app. My experience learning and deploying Shiny apps prior to AI was positive, but also a hurdle given the amount of time it could take to develop and troubleshoot. My familiarity with how powerful Gen AI tools like ChatGPT have become, and how effective they are helping the user build applications in languages like R and Python that are well known and widely discussed on the Internet was a necessary condition to build this app. My assumptions were correct: It took one day of dedicated work to build a rudimentary but working app, and 2-3 more to have something that already could improve my PIAAC data analysis goals. This would not have been possible prior to the rise of Gen AI in 2021(Breznau & Nguyen, 2025).

Fourth, of course, was that I had enough familiarity with both Cycles of PIAAC data to know where to find the documentation, and experience with survey data analysis to identify what I need for optimizing PIAAC work.

### Building the database

There are four key document types that I needed to best perform exploratory scientific analyses using PIAAC Cycles 1 and 2 in combination. These are the *questionnaires*, *variable labels*, longitudinal *trend identification*, and *derived variable* details for each cycle.

Any skilled survey data analyst will tell you that the *questionnaire* is essential because it gives verbatim questions and response categories from the survey. A social scientist cannot determine the face validity or meaning of questions without reading them. For all major languages in all countries surveyed there is a different questionnaire document provided in HTML format[[4]](#footnote-4). As I operate in English and German, I built the app using the German and United Kingdom (British English) questionnaire documents which are available as HTML files, but this could be easily expanded.

When working with the PIAAC data there are variable names and variable labels. After an analyst downloads the PUFs as CSV files, they only contain variable names. Thus, they need a second type of documentation for *variable labels*. They can do this via Stata (Cycle 1 and 2) and R (Cycle 2 only) routines[[5]](#footnote-5). This variable label documentation therefore needs to be added in some form to have all meta data in one place. Reliance on the variable labels along without the questionnaire can be misleading. An example taken from PIAAC sheds light on this. In both Cycles there is a “political efficacy” question. The variable label suggests that the question asks about influence on government. But only by looking at the questionnaire is it possible to identify exactly what this means.

In Cycle 1 the question is “To what extent do you agree or disagree with the following statements? People like me don't have any say about what the government does?” and in Cycle 2 it is “How much would you say the political system in [country] allows people like you to have a say in what the government does?”. These questions both ask about political self-efficacy, but in very different ways. We have known for most of the history of survey research that even small wording differences in survey questions tap different ranges of meanings among respondents (Bishop et al., 1978). In this case the questions are completely different in wording.

This is a case where an analyst might try to compare these variables over time without checking the questionnaire, and generate non-sensical output. Thus, a third document is necessary to easily identify *longitudinal trends*. The PIAAC released a PDF document “PIAAC Cycle 2 BQ Draft Conceptual Framework” (ROA & GESIS, 2025) which details the comparability of variables across the two cycles. It identifies items that are identical, or so close that they constitute a “hard” trend – in other words that they have both face and content validity, or a “soft” trend with only face validity on top of slight changes to the contents, such as question wording, response wordings or both. Also, this document identifies which variables are non-trends, like the political efficacy variable. Finally, it identifies which questions and corresponding variables were dropped after Cycle 1. Of course, an analyst could use the questionnaire to compare between Cycles to identify trend variables, but this is tedious work. Moreover, there are many derived variables, some with extremely complex coding schemes and information that is not fully in the questionnaire.

One example of this is participation in adult education and learning (ALE). The coding rules for derived variables are in a third document to which I will return shortly. Measurement of ALE derivers from various combinations of 20 questions. For example, the variable “NFE12” appears in Cycle 1, which refers to participation in any organized work or non-work related non-formal education (‘NFE’), meaning that is not part of initial educational qualifications (primary, secondary, tertiary) even if these are being sought as an adult. The interviewer gives a show card to demonstrate a concept of what qualifies as ALE for the respondent. Then they are asked to report on four types of ALE (distance, on-the-job, seminars or workshops, and private or other courses not mentioned in the previous three). In each of the four cases a follow-up question is asked if they answered “yes” to determine the number of courses. Then after determining the most recent, several other follow ups are asked about the format, such as whether it related directly to their work and their motivation for participating. The FET12 variable is “yes”, “no” or “missing” for participation, but the question cannot be understood without digging into all the derived variables.

In Cycle 2 a nearly identical variable appears “NFE12C2”. For many trend variables a “C2” is added to denote the same variable in Cycle 2. However, in this case the variable is not considered a trend variable according to the longitudinal trend document. To unpack this requires tracing the complex question wording and variable derivation in Cycle 2. To determine that they should not be compared as a trend, in other words cannot easily be understood by reading casually through the questionnaire. Thus, understanding derived variables such as NFE12, requires the *codebooks* which are in Word Doc formats. Each Cycle as a derived variables codebook with corresponding coding rules to generate the derived variables (OECD, 2015, 2025). The only information I needed for app development was to identify if a variable was derived or not.

Therefore, I use scraping methods to get verbatim question wording from the HTML files into a dataframe (files parse\_questionnaires\_cycle1.R and parse\_quesionnaires\_cycle2.R). I then take conversion of the Stata (Cycle 1 saved as label\_data\_PIAAC\_Cy1.csv) and R (Cycle 2 saved as a dataframe label\_data\_PIAAC.RDS) variable label documents into CSV files which are imported and merged with the dataframe (routing parse\_labels.R), conversion of the longitudinal trend PDF into a Word document using Word which preserved the tables containing the trend data with minor conversion bugs requiring hand editing and then saved as CSV (trend\_vars.csv imported and cleaned via trend\_var.R), and finally copying a list from the derived variables document and saving it as a CSV file (derived\_vars.csv) and putting everything together with the main routine prep \_shiny.R.

### Building the Shiny app

I had two general goals with the app. To have the most information for any given variable in one place, and to find any information in PIAAC by searching text. I imagined a search bar on the left-hand side, where users could put in variable names or any text that appears anywhere in variable labels, question wording and responses in English or German. I also imagined a list on the right-hand side for the search results with the variable name, label, and which Cycle it appeared in (Cycle 1, Cycle 2 or both).

Given my skill set I was confident I could build this; however, I wanted to prompt Gen AI first for two reasons. The first is simply to save time. I am not an expert R Shiny programmer. I can work my way through it, as I have in other apps which I built prior to using Gen AI[[6]](#footnote-6), but if Gen AI can build it through prompting I could save potentially days if not weeks of work. Second, prompting Gen AI could yield features in Shiny app design that I am not aware of, and that might contribute to a better tool. Appendix Prompt 1 lists the exact prompt and corresponding output. Although the entire Gen AI prompting history is quite long, this single prompt might have saved the most time, because it built a framework for a working Shiny app that could then be de-bugged and tweaked.

I faced many further issues for development, but my motivation to push forward was very strong after such a successful first prompt. I discovered that R does not yet have a package specifically designed to deploy a Boolean search. Therefore, I added two search boxes, the second of which searches within the results of the first search. I added the most important meta-data based on my needs, so that the user can find the variable label, question text and responses in English or German, identify if it is a trend variable, and get a list of related variables for example the name of the trend in the other Cycle, variables measuring similar things or those used to construct the variable.

As I am a practicing social scientist rather than a developer, I simply kept a list of bugs that I encountered or ideas that would improve the app, while using it for my own research. This means that I simply added lines of code to the script that would recode values in the dataframe. A developer would probably go back and adjust the scraping routine or improve automation, but I have much simpler goals and am not interested in using up my time resources in such activities. Moreover, I wanted to show that even non-expert R users can build practical tools.

Some notable benefits of working with Shiny are fluid layouts. The infrastructure of Shiny means that all the heavy development lifting is done. Simply packages like fluidPage() generate outputs that seamlessly adjust to screen sizes and move text and fields around to fit. Using base R and tidyverse commands allows for filtering the dataset reactively. Thus, as soon as users type anything in the search or tick boxes, it starts filtering results. Also, it allows the users to click (or toggle on a phone) a single result in the results list and this displays in a lower panel all the meta-data about that variable.

I was able to add some basic instructions and an attribution, along with a logo for branding. With the help of Gen AI I embedded CSS for improved wrapping on screen without the need for horizontal scrolling. Advanced filtering options with check boxes were added to allow users to view only trend variables and filter out results for variables that have no non-missing in either of the two countries or Cycles – this occurs due to data protection policies that got much stricter with Cycle 2. This information was extracted from the actual PIAAC PUFs in the routine prep\_shiny.R.

The app runs locally on any recent R version so long as the user installs all the necessary packages, but I also thought that for the benefit of non-R-users it is helpful to have a publicly accessible tool. For this, shinyapps.io provides a low cost solution to host the app so that users can access it via the Internet and a Web browser. The shinyapps.io service builds a virtual computing environment ensuring that the app will always function even if updates to packages or R itself reduce or remove functionality. This has the benefit that if I no loner maintain the app, or have not yet updated it, users can still access it.

## The PIAAC Variable Finder, a user’s guide

This section is a shortened version of the “User’s Guide” which users will reach by click on the identically named link in the left-hand panel of the Shiny app. The link is dynamic, so it will work either on the Web-based version by linking directly to the guide on Github, or on the local version by opening the PDF file in the /www folder inside the project folder.

Figure 1. PIAAC Variable Finder User Interface

## Appendix

Prompt 1

My first go at a Shiny app:

Ok I have my final dataframe "df" which I will use to build the shiny app. I want a shiny app with two panels, side-by-side. The one on the left will be much smaller and a user interactive panel. It will have a Boolean search bar. In this search bar, users can write Boolean search terms in English or German. The app will then search through the values in the variables "variable", "question\_text\_en", "question\_text\_de", "responses\_en", "responses\_de", "generic\_label\_en", "trend\_var" and "generic\_label\_de". If any hits are found in any of these variables, the corresponding row should be included in the output. The right panel will be all output and be split into two halves and upper half and a lower half. It will report the results of the search in the upper half, which could be more than one hit. Results will be displayed in rows and the top row will be selected by default, but the user themselves can click on different rows. The rows will consist of the values for "variable", "generic\_label\_en" (which will be replaced by "question\_text\_en" if "generic\_label\_en" is NA, and will be replaced by "question\_text\_de" in case both generic\_label\_en and question\_text\_en are NA) and "cycle". The lower half of the right panel will be a report of the selected row (top row by default, or user clicked row). This report will consist of "Variable:" which will out put the value for df$variable, "Cycle:" which will output df$cycle, "Trend status:" which will output df$trend, "Question text EN:" which will output df$question\_text\_en, "Question text DE:" which will output df$question\_text\_de, "Responses:" which will output df$responses\_en (but if this is NA then it will output df$responses\_de), "Related variables:" which will combine all values from the following separated by a comma df$trend\_var, df$ref\_variables and df$constructed\_vars.

ChatGPT 3.5o results:

library(shiny)

library(dplyr)

library(DT)

library(stringr)

# Load Data

df <- readRDS("piaac\_final\_df.RDS") # Ensure this is the correct file name

# Define UI

ui <- fluidPage(

titlePanel("PIAAC Questionnaire Search"),

sidebarLayout(

sidebarPanel(

width = 3, # Smaller panel

textInput("search\_term", "Boolean Search (English/German)", value = ""),

actionButton("search\_btn", "Search")

),

mainPanel(

width = 9,

fluidRow(

column(12, DTOutput("search\_results"))

),

fluidRow(

column(12, verbatimTextOutput("selected\_row"))

)

)

)

)

# Define Server

server <- function(input, output, session) {

# Reactive: Perform Boolean Search

filtered\_data <- reactive({

req(input$search\_btn) # Only search when button is clicked

search\_term <- input$search\_term

if (search\_term == "") {

return(df) # Show all if empty

}

# Filter: Search across multiple columns

df %>%

filter(

str\_detect(variable, regex(search\_term, ignore\_case = TRUE)) |

str\_detect(question\_text\_en, regex(search\_term, ignore\_case = TRUE)) |

str\_detect(question\_text\_de, regex(search\_term, ignore\_case = TRUE)) |

str\_detect(responses\_en, regex(search\_term, ignore\_case = TRUE)) |

str\_detect(responses\_de, regex(search\_term, ignore\_case = TRUE)) |

str\_detect(generic\_label\_en, regex(search\_term, ignore\_case = TRUE)) |

str\_detect(generic\_label\_de, regex(search\_term, ignore\_case = TRUE)) |

str\_detect(trend\_var, regex(search\_term, ignore\_case = TRUE))

)

})

# Output: Display search results in a table

output$search\_results <- renderDT({

df\_filtered <- filtered\_data()

df\_filtered %>%

mutate(

display\_label = coalesce(generic\_label\_en, question\_text\_en, question\_text\_de)

) %>%

select(variable, display\_label, cycle) %>%

datatable(selection = "single", options = list(pageLength = 5))

})

# Reactive: Get selected row data

selected\_row\_data <- reactive({

req(input$search\_results\_rows\_selected)

selected\_index <- input$search\_results\_rows\_selected

df\_filtered <- filtered\_data()

df\_filtered[selected\_index, ]

})

# Output: Display details of selected row

output$selected\_row <- renderPrint({

row <- selected\_row\_data()

if (nrow(row) == 0) {

return("No row selected.")

}

cat("\*\*Variable:\*\*", row$variable, "\n")

cat("\*\*Cycle:\*\*", row$cycle, "\n")

cat("\*\*Trend status:\*\*", row$trend, "\n")

cat("\*\*Question text EN:\*\*", row$question\_text\_en, "\n")

cat("\*\*Question text DE:\*\*", row$question\_text\_de, "\n")

cat("\*\*Responses:\*\*", coalesce(row$responses\_en, row$responses\_de), "\n")

# Combine related variables

related\_vars <- paste(na.omit(c(row$trend\_var, row$ref\_variables, row$constructed\_vars)), collapse = ", ")

cat("\*\*Related variables:\*\*", related\_vars, "\n")

})

}

# Run the application

shinyApp(ui = ui, server = server)

1. There are also two predecessors to the PIAAC that measure similar adult skills across a handful of countries: The *International Adult Literacy Skills Survey* (IALS) and the *Adult Literacy and Life Skills Survey* (ALL). [↑](#footnote-ref-1)
2. Access here: <https://www.oecd.org/en/about/programmes/piaac/piaac-data.html>. Note: There are Scientific Use Files (SUFs) as well which contain fewer redacted variables, but require minor security clearance and a usage agreement for access. [↑](#footnote-ref-2)
3. <https://nate-breznau.shinyapps.io/PIAAC_Variable_Finder/> [↑](#footnote-ref-3)
4. For each country’s questionnaire <https://www.oecd.org/en/data/datasets/piaac-2nd-cycle-database.html> [↑](#footnote-ref-4)
5. Or download the files in SPSS format and convert them. [↑](#footnote-ref-5)
6. <https://nate-breznau.shinyapps.io/shiny/> [↑](#footnote-ref-6)