ChatGPT

Technical Solutions for Ethical Issues in the zoomstudentengagement R Package

Introduction: The **zoomstudentengagement** package must adhere to privacy and security best practices to ensure it **defaults to protecting student data** and complies with laws like FERPA. This means implementing **"privacy by default"** (applying the strictest privacy settings automatically [1]) and incorporating features for **data minimization, retention limits, and secure deletion** [2]. Below we address each critical issue—Privacy-First design, FERPA compliance, Security measures, and Ethical use—providing actionable implementation steps, **R code examples**, recent best practices (2023–2025), and a two-week development roadmap.

1. Privacy-First Defaults and Data Anonymization

Goal: Ensure all package functions default to anonymized outputs, preventing exposure of student names unless explicitly authorized. This aligns with the "privacy-by-default" principle, where the strictest privacy settings are enabled by default [2] [1]. In practice, users should get anonymized results (e.g. *"Student 01, Student 02"* instead of real names) unless they opt out.

Key Implementation Steps:

- **Global Privacy Setting:** Introduce a `set_privacy_defaults()` function to set a global privacy level (e.g., using an R option). All ~40 exported functions will read this default unless overridden by an argument. By default, this should be the highest privacy level (e.g., `"full"` anonymity) [2]. Users can still override per call (for special cases), but the out-of-the-box behavior is privacy-safe.

- `privacy_level` **Parameter:** Add a `privacy_level` argument to **every** user-facing function. This parameter can accept `"full"`, `"partial"`, `"individual"`, or `"none"`. In function definitions, default it to the global option (set by `set_privacy_defaults`). For example:

```
some_function <- function(data, privacy_level =
getOption("zoomengagement.privacy_level", "full"), ...) {
    # ... function code ...
}
```

This ensures that if the user hasn't specified anything, the default from `options()` (set via `set_privacy_defaults`) is applied.

- `set_privacy_defaults()` **Implementation:** This function will update the global option and possibly other package-wide settings. For example:

```
set_privacy_defaults <- function(level =
c("full","partial","individual","none"), focus_student = NULL) {
    level <- match.arg(level)
    options(zoomengagement.privacy_level = level)
    options(zoomengagement.focus_student = focus_student)
    if(level != "none") {
        message("Privacy default set to '", level, "' – outputs will be
anonymized by default.")
    } else {
        message("Warning: privacy_level 'none' will show real names (ensure
authorization).")
    }
}
```

This code uses `options()` to store the default privacy level and (if needed) a target student name for `"individual"` mode. By providing an optional `focus_student` argument, an instructor can specify which student's name *may* be revealed under `"individual"` privacy (all others remain masked). The function also prints a message, with a **strong warning if** `"none"` is chosen, reminding users that actual names will be shown **only when authorized** (e.g. instructor has consent).

- **Name Masking Utility:** Expand the existing `mask_user_names_by_metric()` function into a more general **anonymization utility** that all analysis functions call when needed. For example, create an internal helper `anonymize_names(data, level, focus=NULL)` that handles different levels:

- `"full"` – replace **all** participant names with generic labels (e.g. "Student 01", "Student 02", ...). We can reuse the logic of `mask_user_names_by_metric()` here: perhaps rank or sort the names and then assign `"Student 01"` etc. The key is **consistent pseudonymization** – the same person should get the same label within a session or study [3]. We can achieve consistency by maintaining a mapping (e.g., in a package environment) from real name to pseudonym. A simple approach: sort unique names alphabetically or by some stable metric so that label assignment is deterministic, or use a hash. For example:

```
anonymize_names <- function(df, level, focus=NULL, roster=NULL) {
    if(level == "none") return(df)  # no anonymization
    # Identify instructor(s) if roster provided or by assuming a known
Instructor name in data
    instructor_names <- c()
    if(!is.null(roster) && "role" %in% names(roster)) {
        instructor_names <- roster$name[roster$role == "Instructor"]
    }
    # Build name mapping
    student_names <- setdiff(unique(df$Name), instructor_names)
    mapping <- setNames(character(length(student_names)), student_names)
    if(level %in% c("full","individual","partial")) {
```

```r
        # Generate pseudonyms for each student
        anon_labels <- paste("Student", formatC(1:length(student_names),
width=2, flag="0"))
        mapping[student_names] <- anon_labels
    }
    # If partial or individual, handle instructor or focus student
differently
    if(level == "partial" && length(instructor_names)>0) {
        # Map instructor name to a generic "Instructor" label (to retain
role without identifying)
        for(inst in instructor_names) {
            mapping[inst] <- "Instructor"
        }
    }
    if(level == "individual" && !is.null(focus)) {
        # Ensure the one focus student remains identified (don't anonymize
focus student)
        mapping <- mapping[names(mapping) != focus]
    }
    # Apply mapping to data
    df$Name <- ifelse(df$Name %in% names(mapping), mapping[df$Name],
df$Name)
    return(df)
}
```

In this pseudocode, we replace names depending on level: - For **full**: everyone becomes *Student XX*. - For **partial**: students become *Student XX*, but instructors (identified via a `roster` or known host name) are labeled as "Instructor" (preserving their role without revealing personal identity). - For **individual**: all students except one "focus" student are anonymized. The `focus` name could be provided via the `focus_student` option or parameter; that student's real name remains (assuming consent), others get *Student XX*. This level is useful if analyzing a specific student's engagement (with permission) in context of anonymized peers. - **None**: returns data unmodified (real names visible). We should log or warn when this is used, as it's highly sensitive.

Note: The mapping ensures consistent pseudonyms within the dataset. If you need consistency across multiple datasets/sessions (so *Student 01* in session 1 refers to the same person in session 2), consider using a stable mapping strategy (e.g., hash of the name with a secret salt, or a package like **simplanonym** that anonymizes consistently across datasets [4]). For initial implementation, focus on per-session consistency. Document that cross-session analysis requires using the same mapping (perhaps by saving the mapping or using a consistent seeding method).

- **Default Behavior and Overrides:** Once the above is in place, by default all functions will anonymize outputs. For example, a function `summarize_participation()` might do:

```r
summarize_participation <- function(transcript_df, privacy_level =
getOption("zoomengagement.privacy_level", "full"), ...) {
    # Apply privacy masking unless privacy_level is "none"
    if(privacy_level != "none") {
        transcript_df <- anonymize_names(transcript_df, level = privacy_level,
focus = getOption("zoomengagement.focus_student"))
    }
    # ... compute participation metrics on transcript_df ...
    return(results_df)
}
```

This ensures that unless `privacy_level="none"` is explicitly set, the data is anonymized. If the user has globally set `"partial"` or `"full"`, that automatically takes effect. If a user explicitly calls `summarize_participation(..., privacy_level="none")`, we might issue a caution (e.g., using `warning()` or message) that real names are being used, reminding of authorization needs. This double-confirmation approach prevents accidental exposure.

Code Example – Setting Privacy and Usage:

```r
# Set package-wide default to full anonymization
set_privacy_defaults("full")

# Example transcript data frame
transcript <- data.frame(Name=c("Alice Smith","Dr. John Doe","Bob Lee","Alice
Smith"),
                         Role=c("Student","Instructor","Student","Student"),
                         Speech=c("I have a question", "Let’s discuss that",
"...", "..."))

# Summarize participation with default privacy (full)
result <- summarize_participation(transcript)
# The result$Name would contain "Student 01", "Student 02", etc., instead of
"Alice Smith", "Bob Lee".
print(result)
```

If we print `result`, we might see an output where Alice and Bob’s names have been replaced by *Student 01* and *Student 02*, whereas the instructor might appear as “Instructor” (for **partial** level) or also as a generic label under **full**. The actual mapping should be hidden from the end-user to maintain anonymity. (If needed, an instructor could be given a way to retrieve the mapping from an internal environment for debugging, but that should be tightly controlled or discouraged.)

Privacy by Design (Recent Best Practices): Modern data privacy practices emphasize *pseudonymization* and *data masking* for any personal identifiers [3]. Our approach implements pseudonymization (consistent replacement of names with random/ID strings) as recommended. By ensuring **privacy is the default**, we follow global standards (like GDPR’s “Privacy by Default”) which mandate that users shouldn’t have to take action to secure their data [1]. The default “full” or “partial” anonymization meets this criterion. Only on deliberate request (opt-in) can real names be shown, and even then, usage should be logged (see Audit Logging below) and accompanied by warnings.

Testing Privacy Features: We will write unit tests to verify that no function returns identifiable names when `privacy_level` is not `"none"`. For example, test that `plot_participation()` output labels are anonymized by default. A simple test could create a small transcript with a fake name and check that the fake name does not appear in the function output. This ensures we haven't missed any function. We can also implement a **package self-check** that scans all exported functions to confirm they have a `privacy_level` argument, reducing the chance of omission.

2. FERPA Compliance Features (Data Retention, Deletion, Consent, Audit)

FERPA (Family Educational Rights and Privacy Act) imposes obligations on educational records like Zoom transcripts (which **are** considered education records if students are identifiable [5]). To comply, our package needs features for **consent management, controlled data retention, secure deletion, and audit trails**. Recent FERPA guidance (2023–2025) stresses that you **cannot just keep data "any way you want"** – some data must be retained for a period, and **personally identifiable information (PII) must be destroyed when no longer needed** [6] [7]. Compliance also involves documenting consent and providing transparency.

We implement FERPA compliance in the package via the following components:

- **Consent Tracking and Enforcement**
  It's crucial to **ensure only data from students who have consented to analysis is used**, especially if data will be used for research or shared beyond the classroom. We introduce:

- A way to annotate the class roster with consent status. For example, a data frame `roster` with columns like `name`, `email`, `consent` (TRUE/FALSE), `role` etc. Instructors can provide this roster to the package (perhaps via a function `load_roster()` or as part of data loading).

- **Consent check in analysis functions:** Every analysis function should cross-reference the roster (if provided) and **exclude or flag students without consent**. For example, when loading a transcript, filter out utterances by students who did not consent, or replace their name with "[REDACTED]". This prevents inadvertent analysis of unauthorized data. The function could issue a warning like: *"3 students without consent were removed from analysis."*
- If no roster/consent info is provided, assume all students are to be treated cautiously. We should document that by default, all student data is considered sensitive and only those with proper authority should use it. For research usage, we'll encourage integrating an IRB approval and consent process outside the package, but our package can facilitate by having these hooks for consent data.

Code example: Suppose `transcript_df` has a column `Name`. After reading the .vtt file, do:

```r
if(exists("roster") && "consent" %in% names(roster)) {
    no_consent <- roster$name[roster$consent == FALSE]
    transcript_df <- subset(transcript_df, !(Name %in% no_consent))
    if(length(no_consent) > 0) {
        message(length(no_consent), " participants without consent were removed
from the data.")
    }
}
```

Additionally, we might provide a utility `verify_consent(transcript, roster)` that returns an informative summary (e.g., listing any names present in the transcript that have no recorded consent). This can be used by instructors to double-check compliance before analysis.

- **Documentation & Warnings:** We will update documentation to clearly state that **written consent** is required to use student data beyond normal class operations [8]. If an instructor tries to use `privacy_level="none"` (revealing names) or to export identified data (say, by writing results to CSV), the package can prompt: *"Have you obtained consent from students to use identifiable data?*". This can be a simple `warning()` on such operations. The aim is to remind users of their legal responsibilities.

- **Data Retention Controls**
  FERPA doesn't set a fixed retention period, but it requires that PII **not be kept longer than necessary** for the purpose [9] [10]. Our package will encourage and facilitate proper data retention:

- **Configurable Retention Period:** Provide an option (perhaps via `set_data_retention(days = X)`) that the user (or institution) can set, indicating how long they intend to keep transcript data. For example, an instructor might set 30 days or one term. This information could be used to trigger reminders or automated deletions.

- **Auto-Deletion Reminder:** When the package loads or when certain functions run, it can check timestamps of stored data (if the package had any caching mechanism or if we record when a file was loaded) and remind the user to delete data that exceeds the retention period. E.g., *"Note: Transcript file* `Class1.vtt` *is older than 60 days. Consider deleting to comply with data retention policies."* We can store metadata (maybe in a small CSV in the package's user directory or an R environment) noting when files were loaded.

- **Secure Deletion Function:** Implement a function `secure_delete(file_path)` that **permanently deletes** a transcript or any sensitive file. Simply using `file.remove()` is not truly secure (it just unlinks the file, but data may remain on disk). Instead, we overwrite the file content with random data before deletion. In R, we can do this as follows:

```
secure_delete <- function(file_path, passes = 3) {
    if(!file.exists(file_path)) {
        warning("File not found: ", file_path)
        return(FALSE)
    }
    size <- file.size(file_path)
    # Overwrite file with random bytes multiple times
    con <- file(file_path, "r+")
    for(i in 1:passes) {
        seek(con, where = 0)
```

```
            writeBin(as.raw(sample(0:255, size, replace=TRUE)), con)
        }
        close(con)
        # Remove file name from directory
        file.remove(file_path)
        message("Securely deleted file: ", file_path)
        return(TRUE)
    }
```

This function opens the file in read-write mode, fills it with random bytes for a given number of passes (default 3), closes it, then deletes it. Overwriting multiple times helps mitigate simple recovery techniques. (Note: On SSDs or certain filesystems, true secure deletion is complex, but this is a good-faith effort. We will note in docs that extremely sensitive data might require additional measures.) Using this function, an instructor or researcher can confidently erase transcripts after analysis is done.

We will integrate `secure_delete()` into workflows. For example, after processing a transcript, the instructor could call `secure_delete("session1_transcript.vtt")`. We might even add an argument to `load_zoom_transcript(..., delete_file = FALSE)` – if set to TRUE, the package will automatically call `secure_delete` on the source file after loading it into memory (with a confirmation prompt to avoid accidents). This ensures no local copy lingers beyond its need.

- **Retention Policy Documentation:** We will provide guidance (in vignette or README) on appropriate retention. For example, *"It's recommended to delete or anonymize transcripts at the end of the term or when no longer needed* [7] [11]*. Use* `secure_delete()` *for secure removal. If your institution requires a specific retention period (e.g., 1 year), set that via* `set_data_retention()` *and the package will remind you."* Moreover, incorporate the principle that *personal data should be deleted or anonymized once the purpose is fulfilled* [2].

- **Audit Logging**
  FERPA and general compliance best practices call for **audit trails** of data access and processing [12]. While our R package is a local tool (no server component), we can still implement logging of significant events to a local file. This serves two purposes: (1) **Accountability** – if questions arise, one can review how data was used; (2) **Security** – identifying if any unauthorized action was attempted.

What to log: We should log events such as: loading a transcript, generating a report (especially if containing PII), exporting data, or calling `secure_delete`. The log entry should include timestamp, user (if available via system info), and action details. **Importantly, do not log actual sensitive content** (e.g., don't record actual student names or transcript text in the log). Instead, log metadata like number of records, what was done, etc. For example:

```
2025-08-05 19:21:43 - Loaded transcript "Class123.vtt" (45 entries, 10
participants)
2025-08-05 19:22:10 - Ran summarize_participation() [privacy_level=full]
2025-08-05 19:30:00 - Exported engagement report to
```

```
"engagement.csv" [anonymized]
2025-08-05 20:00:00 - Securely deleted "Class123.vtt"
```

Such a log gives a timeline of data handling without exposing PII.

Implementation approach: We can create a simple logging utility using base R file I/O. For example:

```r
.engagement_log_file <- NULL  # package-level variable for log file path

start_audit_log <- function(file = "zoomengagement_audit.log") {
    .engagement_log_file <<- normalizePath(file, mustWork = FALSE)
    # Initialize log file with header
    writeLines(paste("== ZoomStudentEngagement Audit Log ==",
Sys.time()), .engagement_log_file)
}

log_event <- function(event) {
    if(is.null(.engagement_log_file)) return(FALSE)  # logging not started or
disabled
    timestamp <- format(Sys.time(), "%Y-%m-%d %H:%M:%S")
    entry <- paste(timestamp, "-", event)
    write(entry, file = .engagement_log_file, append = TRUE)
    return(TRUE)
}
```

We use a hidden variable `.engagement_log_file` to store the path once `start_audit_log()` is called. We won't enable logging by default (to respect users who might not want any file output), but we will **recommend enabling it** in research or multi-user contexts.

Throughout the package, pepper calls to `log_event()`. For example, in `load_zoom_transcript()`:

```r
load_zoom_transcript <- function(file, ...) {
    # ... read the file ...
    transcript <- parse_vtt(file)
    # After successful load:
    n <- nrow(transcript); m <- length(unique(transcript$Name))
    log_event(sprintf('Loaded transcript "%s" (%d entries, %d participants)',
file, n, m))
    return(transcript)
}
```

Similarly, after anonymizing or summarizing, log those actions:

```
result <- summarize_participation(transcript_df)
log_event('Ran summarize_participation() [privacy_level=' %s% privacy_level %s%
']')
```

When `secure_delete()` is called, it should log that the file was deleted. If a user tries to set `privacy_level="none"`, we could log an event like *"Privacy override: names exposed in output of X function."* Audit logs like these align with compliance norms by creating an **"audit-ready"** trail [12]. They help demonstrate that the software user (e.g., a school) is taking steps to protect student data (should they ever need to show this to an auditor or during an incident investigation).

The log file itself should be protected (the user should store it in a secure location, since it may contain some meta-information). We will note that in docs. We won't log highly sensitive info, but even knowing a file name or number of students could be sensitive in certain cases. We leave it to users to choose log location (by default we put in working directory or allow them to specify a path, possibly on a secure drive).

- **FERPA Compliance Documentation:** We will include a **"FERPA Compliance" section** in the package documentation, summarizing how to use these features. It will list steps like: *"Maintain a consent record; Always anonymize data by default; Delete raw data after use; Keep audit logs of data processing."* It will reference official guidance that *data must be destroyed when no longer needed for its purpose* [11], and remind that **Zoom recordings/transcripts are protected by FERPA** [5].

Recent FERPA Developments: In the last couple of years, there's been an increased focus on **data governance in education**. For instance, the U.S. Department of Education's Student Privacy office (PTAC) released guidance in 2025 emphasizing that you must manage data retention and destruction properly – *"Neither the law nor best practices allow you to treat data 'Any Way You Want It.'"* [6]. Our package's retention and deletion features directly address this by preventing indefinite storage. Additionally, institutions are expected to have clear **data retention policies and audit logs as part of compliance** [12], which is why we include those capabilities.

FERPA also requires documenting any disclosure of PII. While our tool itself doesn't "disclose" to third parties (it's used by authorized instructors on their own data), the audit log can serve as internal documentation of data handling. If an instructor were to share anonymized results (which is generally safe under FERPA if no PII), there's no violation; if they ever needed to share identifiable data (not recommended), they should obtain consent and the log would reflect that action, providing accountability.

Finally, we note that FERPA's "studies exception" allows use of PII for research to improve instruction **if** certain conditions are met, one being that *data is destroyed when no longer needed for the study* [9] [11]. By building in data deletion and tracking, our package helps meet those conditions in a technical sense.

3. Security Compliance and Secure Data Handling

Beyond privacy, we must ensure the package itself handles data securely and is robust against misuse or attacks. While the package runs locally (no external facing server), it's still important to follow **secure coding practices**: validate inputs, avoid creating vulnerabilities through file handling, and document any security considerations. Educational data breaches are on the rise – e.g., the **PowerSchool breach in Dec**

2024 exposed millions of student records [13] – underscoring that even seemingly internal tools must be built defensively.

Input Validation & Sanitization: Every function that takes user input (file paths, data frames, or parameters) must validate them to prevent malicious inputs or inadvertent errors. Specifically for our package:

- **File Path Validation:** Functions like `load_zoom_transcript(path)` should ensure the `path` is a valid file and of the expected format. We'll implement checks such as:
- Confirm the file exists and has a ".vtt" extension (to avoid someone passing a different file type).
- Use `normalizePath()` to get an absolute path and perhaps check it's not pointing to an unexpected location. (For example, if we wanted to restrict to a project directory, though likely not necessary in this context.)
- Reject paths containing traversal patterns like `"../"` that could be used maliciously. For instance:

  ```
  if(grepl("\\.\\.", path)) stop("Parent directory traversal in path is not
  allowed.")
  ```

  This is a guard against an unlikely scenario (since the user is typically the one providing the path, not an external attacker), but it's a cheap safeguard.

- **File Content Validation:** A Zoom transcript (.vtt file) has a known structure (often starting with "WEBVTT" and containing timestamps). We will parse it with a robust parser (writing a small parser or using any existing library if available). We should validate that the content meets expectations:

- For example, ensure that after parsing, each entry has a timestamp and speaker name. If the file is malformed or contains unexpected data, we handle it gracefully (throw an error or warning, rather than, say, writing it directly to output).

- If our parser encounters HTML or script tags (unlikely in a .vtt, but if someone tampered with it), we should strip or ignore them. We won't `eval()` anything from the file, so code injection risk is low, but it's good practice to **treat the transcript as untrusted text**. That means not using functions like `parse()` on the content.

- **Example – Safe File Loader:**

```
load_zoom_transcript <- function(path) {
    # Basic validation
    if(!file.exists(path)) stop("Transcript file not found: ", path)
    if(!grepl("\\.vtt$", tolower(path))) {
        stop("Invalid file type. Please provide a .vtt transcript file.")
    }
    # (Potentially) Check for excessive file size to prevent memory issues
    # or intentionally large input attacks.
```

```
    if(file.size(path) > 100*1024^2) { # 100 MB as an arbitrary cutoff
        stop("Transcript file is too large and may not be a valid Zoom
transcript.")
    }
    # Read file content
    lines <- readLines(path, warn=FALSE, encoding="UTF-8")
    if(length(lines) == 0) {
        stop("Transcript file is empty or unreadable.")
    }
    # Simple format sanity check:
    if(!grepl("^WEBVTT", lines[1])) {
        warning("File does not start with WEBVTT header – it may not be a
standard Zoom VTT file.")
    }
    # ... proceed to parse the lines into a structured data frame ...
    transcript_df <- parse_vtt_lines(lines)
    # Logging the event
    log_event(sprintf('Loaded transcript "%s" (%d lines, %d speakers)',
                      basename(path), nrow(transcript_df),
length(unique(transcript_df$Name))))
    return(transcript_df)
}
```

Here we enforce correct extension and basic structure. We also guard against extremely large files (which could be a sign of something wrong or a possible DoS vector). The parsing function `parse_vtt_lines` would include logic to split speaker names and text based on the VTT timestamp format. If at any point we find unexpected content (e.g., lines that don't match the pattern "HH:MM:SS --> HH:MM:SS" etc.), we handle it (maybe skip or report it).

- **Data Sanitization:** If the package generates any outputs or reports (e.g., a CSV report or an HTML summary), we must escape any user-provided content. For example, if a student's name is taken as-is to generate a plot title or HTML output, we should sanitize it to prevent issues. Since the data primarily comes from Zoom (student names, spoken text), the risk of things like script injection is low, but consider if a student's name was something like `"><script>alert('x')</script>` (unlikely, but for completeness): if we output that directly in an HTML context, it could execute. Our solution is to **escape special characters** in any context that could interpret them (HTML, LaTeX, etc.). R has functions like `htmlEscape` (in `shiny`) or we can manually gsub `<` to `&lt;`, etc., in any HTML outputs.

- **Preventing Injection Attacks:** Although our package doesn't involve SQL or system commands ordinarily, secure coding practice is to never trust inputs. We will refrain from using `system()` or `shell()` with any input-derived strings. One place we might use system command is in `secure_delete()` if we decided to use a system tool like `shred`. But we chose to implement it in R to avoid invoking shell altogether. This eliminates the risk of command injection through file names. We will still double-check file names (as above) to ensure they don't contain dangerous characters.

R can be vulnerable to injection if user input is passed to certain functions without sanitization [14]. For instance, if we allowed user to provide an expression to filter data, we'd be cautious to use non-eval approaches (like using `dplyr::filter()` with data rather than `eval(parse(...))` on user strings). Our package doesn't currently evaluate arbitrary code from users, which is good – we will keep it that way.

- **Secure Handling of Temporary Files:** If our package uses any temporary files (e.g., writing a temp CSV or plot image), we will ensure they are properly cleaned up after use. Use `on.exit()` in functions to remove temps, or use R's `tempfile()` which generates unique file names and store in secure temp directory. And if those files might contain sensitive data (e.g., an intermediate CSV of participation counts by name), treat them like the raw data: delete them or, even better, avoid writing to disk if possible by using in-memory structures.

- **Use of Encryption (if needed):** Since this is a purely local package, we are not transmitting data over a network (no need for TLS, etc., within the package). However, if the instructor chooses to store results (like an RDS file of processed data), we will advise that they store it securely (e.g., on an encrypted drive or at least not in a publicly accessible location). We could provide an option to **encrypt outputs** – for example, exporting an analysis could optionally produce an encrypted archive. But that might be overkill for now. Instead, we document steps like: *"If you save analysis outputs that contain sensitive data, consider using encryption or password-protected files."* (One could use the `openssl` package in R to encrypt a file with a key if needed.)

- **Security Documentation:** We will include a **SECURITY.md** or a section in the vignette that outlines:

- The security measures implemented (so users or IT departments can review). For example: *"This package validates file inputs and does not transmit any data externally. It provides secure deletion of files. It does not collect any analytics or phone home."*
- Any known vulnerabilities or limitations. For instance, we might note: *"Secure deletion may not completely wipe data on SSD drives due to wear leveling – for extremely sensitive data, consult your IT team for appropriate data destruction policies."* Or *"The package relies on the user's R environment security; ensure your system is secure and up to date (e.g., use R >= 4.4.0 to avoid known R serialization vulnerabilities)."*

- Guidance for users to maintain security: e.g., *"Do not share audit logs or output files that contain student information without anonymization."* and *"Keep your R environment updated to get security patches [15]."*

- **Recent Security Issues & Best Practices:** In early 2024, a vulnerability in R's deserialization (CVE-2024-27322) was discovered that could allow malicious RDS files to execute code [16]. Our package doesn't use RDS files by default, but this highlights the need to be careful with any saved objects. We will encourage using plain text (CSV) for exports, or if using RDS, ensure they are not loaded from untrusted sources. Also, the trend in EdTech is to adopt **strong encryption and access control** even in internal tools [17]. For example, if multiple people use this package on a shared machine, ensure operating system file permissions restrict who can read the transcript files and logs. This is outside our package's direct control, but we will mention it.

In sum, the security compliance work will make the package **robust against common pitfalls**. By validating inputs and sanitizing outputs, we prevent accidental exposures or exploitation. By documenting security features and known risks, we enable users to deploy the tool within their institutional policies confidently. The package will contain **no hardcoded secrets**, no external data calls, and minimal dependencies, reducing its attack surface. All these measures align with general secure coding guidelines and ensure that our package doesn't become the weak link in protecting student data.

4. Ethical Use Guidelines and Bias Mitigation

Finally, we address the overarching ethical mandate: ensure the tool promotes **equitable participation** and isn't misused for surveillance or punitive monitoring. Technology in education should be implemented carefully – research has shown that heavy-handed surveillance can erode trust and chill student engagement [18] . Our package must include guidelines and features to keep the focus on positive, fair use.

Ethical Guidelines Documentation: We will create a detailed **"Ethical Use Guidelines"** section (likely a vignette or wiki page) that covers principles for using **zoomstudentengagement** responsibly. Key points will include: - **Transparency:** Instructors should inform students if they are analyzing class transcripts for participation insights. Being transparent can actually foster a dialogue on participation. It also aligns with emerging ethics guidelines that call for openness about data use in learning analytics. - **Consent and Agency:** Reiterate that students (or their guardians) should consent to this analysis, and they should ideally have the option to opt out without penalty. (Our consent-tracking features support this by enabling excluding those who opt out.) - **Focus on Improvement, Not Punishment:** The tool should be used to **identify and support** under-engaged students, not to penalize them. For instance, if a student rarely speaks, the ethical response is to check in and encourage them, not to publicly shame them with these metrics. We will explicitly caution against using the data for grading or disciplinary action. In our documentation, we might include scenarios or case studies illustrating positive uses (e.g., "Professor uses engagement analysis to adjust teaching methods and include quieter students") vs. negative uses (e.g., "Using transcript data to punish students could backfire by causing anxiety"). - **Equity and Bias Awareness:** A critical ethical aspect is acknowledging and mitigating **biases in the data**. Zoom transcripts and participation metrics can be biased. For example: - **Technical Bias:** Automatic speech recognition (ASR) may have different accuracy for different speakers. Studies have found higher error rates for speakers with accents or from certain racial backgrounds – one study showed ASR made nearly **twice as many errors for Black speakers compared to white speakers** [19] . Also, non-native English speakers might have some of their speech transcribed incorrectly (Zoom's own metrics showed a ~3.6% lower accuracy for non-native vs. native speakers) [20] . These errors mean the raw transcript might undercount contributions from some students. Our guidelines will alert users to this possibility: *"If a student has an accent or the transcript quality is poor, their participation may be underreported. Cross-verify important insights with actual observations or recordings."* We could also suggest using the Zoom recording to manually check if needed, or encourage students to review their portions of transcripts for accuracy. - **Behavioral Bias:** Some students participate in ways not captured by transcripts (e.g., chat messages, collaborative documents, or they may communicate non-verbally). If an instructor only focuses on spoken contributions, they might overlook those other forms. Ethically, we advise instructors to consider multiple dimensions of participation. We might integrate features in the package in the future for analyzing chat logs as well, but for now we at least mention it. - **Algorithmic Bias:** Our analysis functions themselves should be reviewed to ensure they don't inadvertently favor or disfavor any group. For instance, if we rank students by talk time, that's a neutral metric, but interpreting it requires care (maybe the one with least talk time is actually engaged through chat or was absent, etc.). We'll include guidance on interpreting results cautiously and within context.

- **Bias Assessment Tools:** As a technical aid, we could implement a simple **bias check** function. For example, `check_transcript_quality(transcript_df)` could analyze the transcript for anomalies like large sections of "[inaudible]" or very short responses that might indicate transcription issues. It could flag: *"Audio quality for Student 03 might be low (many undeciphered segments), results could be biased."* While we may not reliably know why something is missing, we can at least detect if one student has significantly fewer words but also many blanks, etc. Another idea: if we have access to confidence scores from Zoom (some APIs provide per-word confidence), use that to identify segments of low confidence aligned with certain speakers.

In absence of such detailed data, we advise manual bias checks: e.g., *the instructor should reflect on whether any student's low participation metric might be due to factors like technology issues, language barriers, or an intimidating environment.* The ethical guidelines will emphasize: **numbers don't tell the whole story** – they are conversation starters, not final judgments.

- **Positive Outcomes Focus:** We will include positive use-cases and perhaps code examples that encourage equity:
- For instance, show how to use the package to detect if you (the instructor) are dominating the conversation. A high instructor word count vs. students might prompt the instructor to adjust and create more space for students. This flips the script from surveilling students to improving teaching, which is ethically sound.

- Show how to aggregate participation by demographic (if such data is available in roster) to see if certain groups are less engaged – not to single them out, but to identify systemic issues. For example, if we notice that female students spoke 30% of the time in a class that's 50% female, that imbalance could spark a reflection on class dynamics (perhaps certain voices are being unintentionally favored). Of course, such analysis must be done carefully and respectfully, but it demonstrates an equity focus. (Any demographic analysis should never identify individuals and should be done only in aggregate with appropriate privacy; we'd mention that if doing this kind of analysis, ensure it's covered by consent and IRB if research.)

- **Package Features to Reinforce Ethics:** Aside from documentation, a few technical measures can remind users of ethical usage:

- **Startup Message:** When the package is loaded (`.onAttach` in R), we can display a brief message: *"Remember: use zoomstudentengagement to support learning, not to surveil. Ensure student privacy and consent."* This gentle nudge every time they load the package keeps ethics in mind.

- **Function Warnings for Misuse:** If a user tries to do something like export identifiable data, we could include a prompt: *"You are about to save data with student names. Please ensure FERPA compliance and ethical use."* Requiring them to set a flag like `override = TRUE` to actually proceed can inject a moment of reflection. This is similar to how some dangerous functions require `confirm=TRUE`. For example:

```
export_results <- function(data, file, privacy_level =
getOption("zoomengagement.privacy_level", "full"), override = FALSE) {
    if(privacy_level == "none" && !override) {
        stop("Exporting identifiable data is discouraged. If you have
proper authorization, call again with override=TRUE.")
    }
    # proceed to write.csv or similar
}
```

This makes it harder (or at least very intentional) to export PII, aligning with an ethical stance.

- **Ethics Review Checklist:** We can provide a simple checklist (in documentation or as a function that prints it) that users can go through before using the tool each term. For example: `ethics_checklist()` could print:
    - Have you informed students about this analysis?
    - Do you have consent from students (or is this within allowable educational use)?
    - Is your goal to help students (not penalize them)?
    - Have you considered potential biases in the data?
    - Do you have a plan to follow up with students who might be struggling, in a supportive way? This could just be a markdown in the documentation, but framing it as a function or a vignette makes it more likely to be seen.

- **Updated Ethical Guidelines (2023-2025):** The field of learning analytics has been actively discussing ethics. Recent guidelines (such as those by EDUCAUSE or Every Learner Everywhere) stress **student-centric approaches, transparency, and fairness**. For instance, an EDUCAUSE article in 2019 noted a “push to monitor student data is met with concerns about privacy and equity,” urging caution and clarity in purpose (B. Koenig, EdSurge, 2019). A 2021 UNESCO report on AI in education also highlighted protecting student rights and avoiding biases. Our implementation echoes those by ensuring we’re not creating a surveillance tool but a reflective aid. Furthermore, there’s a push for **student agency** – some advanced systems even allow students to view and correct their own data. While that’s beyond our scope (we’re not building a student-facing app), we do encourage involving students in interpreting the data. For example, an instructor could share anonymized class participation visualizations with the class and discuss what they mean, rather than using them in a secretive manner. That fosters trust and a collective effort to improve participation, rather than fear.

- **Preventing Misinterpretation:** We will include notes that low participation metrics do not necessarily mean a student is disengaged or “bad” – there could be external factors (illness, technical issues, personal preferences). Similarly, a student with high talk time isn’t necessarily contributing more substantively. The tool cannot measure quality of contributions. We caution users to use the metrics as a **starting point for further inquiry**. For example, if Student X has spoken very little, the instructor might reach out privately to check if they’re comfortable in class or if something is affecting their ability to participate. This ensures the data is used compassionately and constructively.

In summary, the ethical guidelines coupled with the technical safeguards (privacy defaults, consent requirements, difficulty of exporting PII) form a comprehensive approach to keep the package's use aligned with its intended purpose: **supporting equitable student engagement**.

By explicitly addressing these points, we signal to CRAN (and users) that we have thoroughly considered the ethical implications. This not only helps pass the submission (CRAN is increasingly attentive to privacy/ethics for data-related packages) but more importantly ensures the tool, once in use, will have a positive impact on teaching and learning.

As a final measure, we might seek an external review of our guidelines by an educational ethics expert or run an informal workshop with some instructors and students to get feedback on whether our proposed use is acceptable to them. This kind of community engagement, while not required for CRAN, demonstrates our commitment to ethical development.

---

5. Implementation Roadmap (2-Week Timeline)

With the requirements clarified, we outline a **2-week (14-day) development plan** with daily tasks. This plan prioritizes legal compliance features first (FERPA/privacy) and then moves to security and documentation, aligning with the principle that compliance and ethics are not afterthoughts but integral to the development:

Day 1: Project Setup and Planning – Update the project repository with a new branch for "compliance_update". Review all 40+ exported functions and list where changes are needed (e.g., which functions produce outputs requiring anonymization, which save files, etc.). Draft the `privacy_level` API design and `set_privacy_defaults` interface. **Output:** Detailed design notes for implementing privacy and consent in each function.

Day 2: Implement Global Privacy Infrastructure – Code the `set_privacy_defaults()` function and test setting/getting the global option. Begin adding the `privacy_level` argument to a few core functions as a prototype (e.g., the transcript loading and a summary function). Implement the `anonymize_names()` helper and test it on a small dummy transcript. Ensure that `"full"`, `"partial"`, `"individual"`, `"none"` behaviors work as expected (write a quick test script). **Output:** Core privacy functions in place.

Day 3: Apply Privacy Defaults to All Functions – Methodically go through each exported function (perhaps in alphabetical order or by category) and add `privacy_level` in the arguments and corresponding anonymization logic. This is a bulk of mechanical work. Use search (e.g., find where `mask_user_names_by_metric` was used and replace/improve those sections). After code changes, run `R CMD check` to catch any syntax issues. **Output:** All functions now have privacy controls (but not fully tested yet).

Day 4: Test & Refine Privacy Implementation – Create several test scenarios: one with a full class transcript, one with an instructor present, one focusing on an individual. Test that default calls yield anonymized outputs (e.g., print a summary, see no real names). Test overriding the global default (set default to "full", but call one function with `privacy_level="none"` to ensure it actually does show names when asked). Fix any bugs (e.g., if factors were used for names, ensure our replacement doesn't produce NA factor levels, etc.). Write unit tests for anonymization (if using a testing framework). **Output:** Privacy feature verified to work correctly across the package.

Day 5: Develop Consent Tracking Features – Design how the roster/consent information will be provided. Possibly implement a function `attach_roster(dataframe)` that stores the roster internally (or simply instruct users to keep a roster and pass it to functions that need it). Implement modifications in `load_zoom_transcript()` or analysis functions to filter out non-consenting students. Test with a sample roster (mark one student as no-consent and see that their data is dropped or masked). Also implement any **consent warnings**: e.g., if a transcript contains a name not in the roster (or whose consent is FALSE), perhaps `load_zoom_transcript` could warn *"X appears in transcript but has no consent record; excluding from analysis."* Ensure this behavior can be toggled (maybe an argument `enforce_consent=TRUE` by default). **Output:** Consent enforcement integrated into data loading/processing.

Day 6: Implement Data Retention & secure_delete – Write the `secure_delete()` function as designed. Test it on a dummy file (create a temp file, write data, secure_delete it, then check file no longer exists). Also, implement a simple tracking of file load times: e.g., modify `load_zoom_transcript` to record an entry in an internal list (filename -> Sys.time()). Then implement `check_retention()` that goes through that list and prints any files older than the retention period set (if none set, default maybe 30 days). We might integrate this check to run whenever the package is loaded or when `load_zoom_transcript` is called (to remind about older files). Ensure that these checks only produce messages, not hard stops. **Output:** `secure_delete` working and retention checks in place.

Day 7: Implement Audit Logging – Add the `start_audit_log()` and `log_event()` functions. Test by calling `start_audit_log()` in the R console and then running a few package functions, then open the log file to verify entries. Integrate `log_event` calls at key points in the code: after loading data, after analysis, on export, on deletion. This requires scanning the code; use a consistent phrasing for log messages. Also, ensure that if logging is not started, our `log_event` calls do nothing (so by default there's no error). **Output:** Logging feature integrated, with sample log verified.

Day 8: Security Hardening – Go through functions like `load_zoom_transcript` (already did some), any file writing functions, etc., and add input validation and sanitization. Include checks for unusual inputs (as discussed). Also double-check that we're not leaving behind any temp files or unneeded variables containing PII. If the package has plots (maybe generating ggplot objects with student names), consider adding a parameter to anonymize labels in plots by default too. Implement any needed escaping for outputs. Possibly run an R security lint tool or just manual code review for vulnerabilities. For example, search for usage of `eval(parse(` or `system(` to confirm none are present. **Output:** Code is reviewed and updated for secure handling.

Day 9: Documentation – Function Docs and Vignettes – Update each function's documentation (`.Rd` or roxygen comments) to describe the new `privacy_level` argument, its default behavior, and examples of usage. Write a dedicated **"Privacy and Security" vignette** explaining how to use `set_privacy_defaults`, `secure_delete`, audit logs, etc., with sample code (some of which we drafted above). Also write an **"Ethical Use" vignette** that compiles the guidelines: explain FERPA briefly, consent, intended use, avoiding misuse, bias awareness. Use examples or Q&A style for clarity. This day is text-heavy; ensure to keep a professional yet clear tone. **Output:** Updated documentation and two new vignettes (Privacy/Security, and Ethical Guidelines).

Day 10: README and CRAN comments – Update README.md to highlight the privacy-first nature of the package (this can also serve as a selling point to users). The README should mention that the package has features ensuring compliance with student privacy. Also prepare any additional documentation required for CRAN submission: for instance, CRAN might expect a statement on data usage. We will include in the DESCRIPTION or as a CRAN submission comment a note like, *"This package does not collect any personal data; it only processes user-provided transcript files in memory. All privacy-sensitive features are opt-in by the user. The package includes extensive documentation on ethical use."* Basically, proactively assure CRAN that we are aware of and have mitigated the privacy issues (which they flagged). **Output:** Polished README and CRAN submission notes.

Day 11: Testing and QA – Spend this day writing additional **unit tests** and doing manual testing as needed: - Test the package on a realistic sample (perhaps ask a colleague for an anonymized transcript or create a fake one with various names). - Verify that setting `privacy_level` globally and locally works in all combinations. - Test that `secure_delete` truly removes files and that `log_event` entries make sense. - Simulate some edge cases: e.g., no instructor in roster but using partial (should just anonymize everyone in that case), or focus student name not present (should handle gracefully). - Check that warnings and messages are informative but not overly verbose (maybe suppress them in tests if needed). - Run `R CMD check` and address any warnings (for example, writing to a file in examples might need `\dontrun{}` tags to avoid CRAN issues). - Ensure our new vignettes build without errors and are included. - Check that no PII is accidentally hardcoded in examples (use "Alice/Bob" type dummy names only). **Output:** Test results documented, any bugs found are fixed.

Day 12: Performance and Edge Considerations – With new privacy layers, test performance on a large transcript to ensure we haven't introduced a big slowdown. (Masking names is trivial even for large data, but logging every event could slow loops if done excessively – ensure we're not logging inside tight loops, only high-level events.) Optimize if needed (maybe use vectorized operations in anonymize_names, etc.). Also consider memory: our anonymization makes a copy of data; for very large transcripts that might be heavy. But transcripts are text, usually not more than few MBs, so likely fine. Check that the package still works on all platforms (especially file deletion on Windows vs Unix differences). Possibly test secure_delete on Windows (writing random bytes in text mode might need binary mode; ensure to open file in "rb+" if needed). **Output:** Package performs adequately, cross-platform considerations handled.

Day 13: Final Review and Polishing – Review all changes with a fresh eye. Ensure consistency: e.g., the `privacy_level` parameter should have the same default and documented options everywhere. The consent enforcement should be uniformly applied (not in some functions and not others). Resolve any TODOs left in code comments. Double-check compliance: maybe use a FERPA compliance checklist and see that our features tick all the boxes (privacy, consent, rights, security). Possibly have another person review the changes (if available). **Output:** Code ready for release, version bumped (e.g., to 1.0.1 or 1.1.0 as appropriate), NEWS file updated with changes.

Day 14: Package Submission and Follow-up Plan – Submit the updated package to CRAN (or if this is not the first submission, at least get ready for resubmission addressing the issues). Along with submission, include clear explanations in the comments to CRAN about how each issue was addressed: - Issue #125: "Implemented global privacy defaults and anonymization – all functions now default to anonymized output (see vignette on Privacy).” - Issue #126: “Added FERPA compliance features: consent tracking, data retention controls, secure deletion function, and audit logging (documented in detail).” - Issue #84: “Conducted thorough input validation and added security measures; also provided SECURITY.md.” - Issue #85: “Included Ethical Use Guidelines in documentation and package startup messages to prevent misuse.”

After submission, plan for any feedback from CRAN reviewers. If they request changes (perhaps to remove or modify how we write files in examples or how we use options), be ready to address them quickly. Also plan to monitor user feedback once released – e.g., if instructors find something cumbersome (maybe requiring `override=TRUE` too often), be open to iterate, but **never at the cost of violating the core ethical stance**.

Milestone: At the end of two weeks, we expect the package to be compliant and ethical by design, ready for acceptance by CRAN.

Testing & Validation Strategies

To ensure these features truly work and address the issues, we outline how we will test and validate each aspect:

- **Unit Tests:** We’ll create automated tests for critical functions:
- Privacy tests: Input a small transcript with known names and check that after processing (with default privacy), none of those real names appear in the output objects. Also test that with `privacy_level="none"`, the names do appear (ensuring the toggle works).
- Consent tests: Provide a roster with one student’s consent = FALSE and a transcript including that student. Test that analysis results exclude that student’s data. Also test that a warning/message is emitted about exclusion.
- Secure deletion test: Create a temp file, write contents, run `secure_delete`, and then assert `file.exists` is false. (We might not verify the overwrite bytes in tests due to complexity, but we ensure deletion).
- Logging test: Use `tempfile()` for log, call a couple of logged actions, then read the log file and assert that entries contain expected substrings (timestamp and event keywords).
- Input validation test: Try `load_zoom_transcript` with an invalid extension or malformed content and expect it to throw an error or warning appropriately.

- **Compliance Verification:** We will perform a **self-audit** of the package using the checklist of the four issues:

- Scan through all exported functions to confirm presence of `privacy_level` and proper handling.
- Use R’s codetools or lintr to identify any usage of functions that could be security-sensitive (like `sys.getenv` or others; our package likely doesn’t use those).
- Run the package in a simulated environment where we intentionally violate something (like attempt to export data without consent) and ensure the package responds as intended (e.g., stops with an error about missing consent).

- **User Acceptance Testing:** If possible, get an educator colleague to try the updated package on some dummy data to see if the workflow makes sense to them. They can provide feedback on whether the warnings and messages are clear, or if something is too strict/lenient.

- **Documentation Validation:** Ensure that vignettes and examples cover the main use cases so that when someone reads them, they inherently follow the privacy/ethical practices. For example, our examples in documentation will always show use of `set_privacy_defaults("full")` at the start, modeling best practice to the user.

By implementing these testing strategies, we reduce the chance of regression and give confidence that the critical issues are resolved not only on paper but in practice.

Potential Pitfalls and Mitigations

In the course of development and usage, there are some potential pitfalls to watch out for, and our plan addresses how to avoid them:

- **Pitfall 1: Missing Anonymization in Some Outputs.** With 40+ functions, there's a risk we miss one path (maybe a plot title or an error message that inadvertently includes a student name). **Mitigation:** exhaustive search in code for any use of the `Name` field or similar and ensure it's passed through anonymization. Also, peer code review can help catch anything missed. We will also add a note in the package README encouraging users to report any instance of real names appearing unexpectedly, so we can patch it. The comprehensive tests for no real names in outputs help catch these.

- **Pitfall 2: Overzealous Data Deletion.** We provide tools to delete data, but we must be careful not to delete something the user didn't intend. For example, if `load_zoom_transcript(delete_file=TRUE)` is used incorrectly, the user might lose a file without having a backup. **Mitigation:** By default, do not delete automatically; require explicit call to `secure_delete` or explicit `delete_file=TRUE` plus perhaps a confirmation. We included confirmations (like `override=TRUE`) for destructive actions. Also, clearly document that these actions are irreversible.

- **Pitfall 3: Complexity for Users.** Adding privacy_level to every function and requiring roster, etc., could make the package usage feel heavier. There's a chance some users might try to bypass or disable these features out of convenience. **Mitigation:** We make the defaults safe but also convenient (e.g., default anonymity requires no extra work). We'll ensure performance overhead of these features is minimal (so no reason to turn them off). We also plan good documentation and maybe helper functions (like one call to set the roster and privacy that then applies globally). If users have an easy way to "do the right thing," they're less likely to hack around it. Additionally, the messaging we provide about why these features matter can persuade users to cooperate rather than circumvent.

- **Pitfall 4: False Sense of Security.** Users might assume that because they use our package, they are 100% compliant without doing anything else. In reality, they still need to follow institutional policies (like not sharing even anonymized data publicly if not allowed, etc.). **Mitigation:** We explicitly state the scope: *"This package provides tools to help with privacy, but ultimate responsibility lies with the user to use data appropriately."* We mention things like ensuring their computer is secure, etc. Transparency with users prevents complacency.

- **Pitfall 5: Software Maintenance.** Introducing these features means more code to maintain. We must ensure future contributors do not inadvertently break privacy (e.g., someone adding a new function might forget privacy_level). **Mitigation:** We will add contribution guidelines stating that any new analytic function **must** include privacy considerations. Perhaps we'll implement a template or use unit tests that automatically fail if a new function lacks required elements. Continuous integration can run tests on pull requests to catch issues early.

By recognizing these pitfalls, we have already built in solutions to address them. The result should be a robust, secure, and ethical package that not only passes CRAN submission this time but sets a high standard for any future educational data analysis tools.

Sources:

- FERPA requirement for data destruction when no longer needed [7] [11]
- U.S. Dept of Education on data retention vs deletion [6]
- Compliance measures like audit logging & retention policies are essential for FERPA [12]
- GDPR "privacy by default" principle (strictest settings by default) [1] [2]
- *Deident* R package (2025) on pseudonymization and data masking techniques [3]
- *simplanonym* approach for consistent anonymization across datasets [4]
- Zoom class recordings as educational records under FERPA [5]
- Stanford study on speech recognition bias (racial error disparity) [19]
- Difficulty of ASR with accents/dialects [21]
- NASBE warning on surveillance harming student trust [18]
- PowerSchool 2024 breach highlighting need for security in edtech [13]
- R security best practices (input validation to prevent injections) [14]

---

[1] [2] Privacy by Design & Default - Overview - Securiti
https://securiti.ai/blog/privacy-by-design-privacy-by-default/

[3] Deident: An R package for data anonymization
https://www.theoj.org/joss-papers/joss.07157/10.21105.joss.07157.pdf

[4] [PDF] simplanonym: Consistent Anonymisation Across Datasets - CRAN
https://cran.r-project.org/web/packages/simplanonym/simplanonym.pdf

[5] [8] [10] FERPA Training | Education Records Under FERPA | TeachPrivacy
https://teachprivacy.com/what-is-an-education-record-under-ferpa-a-flowchart/

[6] Data Retention and Data Destruction | Protecting Student Privacy
http://studentprivacy.ed.gov/training/data-retention-and-data-destruction

[7] [9] [11] FERPA | Protecting Student Privacy
http://studentprivacy.ed.gov/ferpa

[12] [17] Hardening Your Infra for Classrooms, Campuses, and Compliance
https://criticalcloud.ai/blog/hardening-your-infra-for-classrooms-campuses-and-compliance

[13] PowerSchool Data Breach Exposes Student Information System
https://www.sangfor.com/blog/cybersecurity/powerschool-breach-student-information-system-hack-exposes-data

[14] Blueinfy's blog: Performing Secure Code Review of R Code – A Beginner’s Guide
https://blog.blueinfy.com/2024/02/performing-secure-code-review-of-r-code.html

[15] May 22, 2024: Default R Version Update due to Security Vulnerability
https://bcg.biostat.wisc.edu/2024/05/22/may-22-2024-default-r-version-update-due-to-security-vulnerability/

[16] New Vulnerability in R's Deserialization Discovered - Cyberint
https://cyberint.com/blog/research/new-vulnerability-in-rs-deserialization-discovered/

[18] School Surveillance: The Consequences for Equity and Privacy – NASBE – National Association of State Boards of Education
https://www.nasbe.org/school-surveillance-the-consequences-for-equity-and-privacy/

[19] Automated speech recognition less accurate for blacks
https://news.stanford.edu/stories/2020/03/automated-speech-recognition-less-accurate-blacks

[20] Lost in Transcription: Auto-Captions Often Fall Short on Zoom ...
https://www.consumerreports.org/disability-rights/auto-captions-often-fall-short-on-zoom-facebook-and-others-a9742392879/

[21] [PDF] Biases in a Digital Era: Examining the Accuracy of Transcription Tools
https://repositories.lib.utexas.edu/bitstreams/3662f5d4-273b-42fd-a52b-c2d406c0b73b/download