README WORD DOC:

How is the file structured, plus notes on where to look for more information.

NOTE: There are google docs (under silent spring, report back, etc…) with many notes

Here's the concise call tree:

1. **MyExposome\_1527\_v6.Rmd**
   * calls: SpecificSubjectAnalysisIntro\_text.Rmd
   * calls: GroupAnalysisIntro\_text.Rmd
   * calls: GroupAndIndividualAnalysisIntro\_text.Rmd
   * calls: StatisticsOnChemicalsFound.Rmd
   * calls: YourChemicalsAndAveragesByGroup\_text.Rmd
   * calls: IndividualChemicalListWithClassification\_text.Rmd
   * calls: IndividualClassificationShowingChemicals\_text.Rmd
2. **MyExp\_support\_functions\_1527\_v6.R**
   * defines: buildMesgVIndividual()
   * defines: buildMesgVGroup()
   * defines: addMesg()
   * defines: makeClickableURL()
3. **MyExp\_data.load.functions\_1527\_v6.R**
   * defines: load.masterParam()
   * defines: load.classification()
   * defines: convert\_to\_new\_reduced\_classifications()
   * defines: updateWithClassSpecificMasterParam()
   * defines: load.testResults\_justReadTable()
   * defines: load.testResults()
   * defines: fixUpTestResults()
   * defines: addAirCalculationInformation()
   * defines: addAirNioshOsha()
   * defines: makeIntoDemoDataResults()
   * defines: pickSubsetOfResults()
   * defines: onlyPickSomeBatchesFromBiggerData()
   * defines: load.riskCalifProp65()
   * defines: load.epaIris()
   * defines: load.IARCRisk()
   * defines: load.chemSourceMitigation2()

That sounds like a complex and rewarding project! Here’s a systematic approach we can take to clean up and refactor the code, ensuring nothing breaks in the process. I’ll outline the steps and how we can collaborate effectively.

**Step 1: Understand the Project Structure**

1. **Directory Structure**: Describe or share the layout of directories and subdirectories so I understand where .Rmd and .R files are located.
2. **Interdependencies**: Provide an overview of how the .Rmd and .R files are connected (e.g., which files source each other, common functions, shared data files).
3. **Goals for Cleanup**: Clarify specific goals for refactoring, such as:
   * Moving reusable functions into separate files or a package.
   * Cleaning up or reordering code within .Rmd files.
   * Optimizing code for performance or readability.

If possible, you could share a list of the main functions or key components, so I have a sense of the code organization.

**Step 2: Identify Key Files and Functions**

1. **Catalog Functions**: We can identify all the unique functions in your .R and .Rmd files. You could use a tool like grep or ag (the Silver Searcher) to quickly list all function definitions (function() across files, or I can guide you on how to do it in R.
2. **Identify Dependencies**: Understanding which files call each other (or use each other's functions) will help us create a dependency graph. I can help you build this using a tool like renv or drake, or we can document it manually if the structure is manageable.

**Step 3: Create a Consolidated "Utilities" File or Package**

1. **Separate Reusable Functions**: As we identify functions that are used across multiple files, we can create a "utility" file (e.g., utils.R) or, if there are many functions, create a custom package.
2. **Refactor to Source the Utility File**: Update .Rmd and .R files to source utils.R or load the custom package, so they don’t directly depend on each other. This minimizes the web of dependencies.

**Step 4: Modularize Code in .Rmd Files**

If some .Rmd files are "calling" others, we can modularize the code to simplify the workflow. Two approaches:

1. **Parameterize Reports**: Instead of one .Rmd calling another, we can refactor each .Rmd file as a self-contained, parameterized report. This way, they don’t depend on each other but can still share code by calling the same functions.
2. **Move Computations to .R Scripts**: Extract any heavy computations or analysis into standalone .R scripts, and call those scripts from the .Rmd files. This keeps the .Rmd focused on reporting and visualization.

**Step 5: Automate and Test**

1. **Set Up Version Control**: If you’re not already using Git, setting up version control will help us manage changes safely. I can guide you through committing logical chunks of changes and creating branches for different aspects of refactoring.
2. **Automate Workflow**: Use rmarkdown::render() or targets/drake to automate rendering of reports and ensure all interdependencies are met.
3. **Test Incrementally**: Make changes one at a time and verify that each step doesn’t break functionality. We can implement unit tests (using testthat) for critical functions if appropriate.

**How We Can Collaborate**

* **Document Sharing**: If you can upload files or share specific pieces of code, I can analyze and refactor them here.
* **Code-by-Code Approach**: Start by identifying specific files or functions that need the most attention. We can go through them together, refactor, and verify each step.
* **Iterative Process**: We’ll work in small iterations—focusing on one file or dependency group at a time—to keep everything manageable and minimize risk.

Let me know if you want to start with a specific function or file, and we can begin organizing!