Proposal

Page • 1 backlink • Tag

Project info

Project title: grepreaper R Package Development

Project short title (30 characters): grepreaper R Package Development

URL of project idea page: https://github.com/rstats-gsoc/gsoc2025/wiki/grepreaper-R-Package-Development

Bio of Contributor

I'm Aastik, a CS undergrad in my junior year at VIT Vellore, Tamil Nadu. I have worked as a developer(and still am) with multiple stacks for both mobile applications, full stack web apps and even have cli/tui writing experience. I was first introduced to the R language an year ago. Working with it for college assignments, I found it really intuitive and simple to use. The concepts were highly practical and easy to grasp. It felt like a language built to be used for practical applications. So for the past 2 months I have been learning R, trying out it's different data handling libraries like dplyr, tidyr, ggplot2 or libraries like plumber for building REST APIs. I use Fedora Linux 41 (Workstation edition) as my daily driver and for R related projects I prefer R studio for R projects and nvim for quick edits. I am quite fluent with the terminal and that is why this project particularly piqued my interest. grep is something I use often but using it in R wasn't a combination I had even imagined previously but makes so much sense when given a proper thought.

I am quite confident that my experience working on real-life projects and experience with big coding projects sets me apart and gives me an edge over a lot of other candidates. I have experience working with tight deadlines, delays and unforeseen problems that might arise.

Contact Information

Contributor name: Aastik Narang

Contributor postal address: Hno. 2454, Sector -16, Faridabad, Haryana, India

Telephone(s): +91 8383827865

Email(s): aastik839@gmail.com(registered email), aastikn@outlook.com,

Other communications channels:

- Whatsapp : +91 8383827865
- Discord: thegnaran

Contributor affiliation

Institution: Vellore Institute of Technology,Vellore,Tamil Nadu

Program: Bachelors in Technology, Computer Science Engineering

Stage of completion: 3 years out of 4 completed

Contact to verify: aastik.narang2022@vitstudent.ac.in

Schedule Conflicts:

I also plan to spend ~10 -15 hours a week on my personal learning, but that would be highly flexible. These hours can also be adjusted.

On 9th July, my uni would be reopening and the hours I can give would fall to ~15-20 hours a week.

Before returning to college I plan to commit ~50 hours a week just to working on the project. Since the time I can give after coming back to uni would be significantly lesser I will be focusing on finishing most of it beforehand.

Mentors

Evaluating mentor name and email: David Shilane david.shilane@columbia.edu

Co-mentor name(s) and email(s): Toby Dylan Hocking tdhock5@gmail.com

Have you been in touch with the mentors? When and how?:

As per the contributor guidelines on the wiki I have emailed David upon completing the tasks on 2nd April.

Coding Plan and Methods

This is the project description provided on the Idea page:

The package will include development and revision of a number of functions:

```
grep.count: Count the number of relevant rows of data in one or more
files, overall or matching search patterns.

grep.read: Read and aggregate data from one or more files while allowing
for pattern matching.
```

The coding process will involve using R to perform a few tasks: a) use string-printing techniques to construct the grep statement, b) build variations on the coding statement to match different types of pattern-matching (such as full or partial patterns, along with regular or inverted matches), and c) read the files through data.table's fread() command. Ideally these functions will also provide options to show either a) the resulting counts/data, b) the command-line grep statement, or c) both. You will begin with some partially-developed code that can be revised and expanded upon.

It makes it pretty clear that there'll be 2 main functions as listed above.

I plan to divide time equally. My plan would be to segment the function into all the possible overloading functions and develop each at a time while working on it's documentation and testing as soon as it is completed. I feel testing after completing an entire function would be highly inefficient and the complexity would also increase.

Now let's dive into how I plan to create the functions and their various overloads:

Grep.count:

```
grep.count(data='<dataset_name>',pattern
='<pattern>',column='<columnName>',ignoreCase='<bool>',partial='<bool>',
inverted='<no,only,both>)
```

here's what I mean each argument to b used for:

- Path: The file path to access the data,
- Pattern: The pattern or regular expression to look for while matching in the given dataset
- Column: Specify column in the dataset if the user wants to search specifically
- IgnoreCase: To match case with the pattern/regex or not
- Partial: To count partial matches or not
- Inverted: I plan for this to have no, only and both.
    - No is for no inverted matches, only is to count only inverted ones and both is for matching both inverted and normal

Grep.read

Plain Text

```
grep.read(data='<dataset_name>',pattern
='<pattern>',column='<columnName>',ignoreCase='<bool>',partial='<bool>',
inverted='<no,only,both>)
```

I feel that the arguments for both the functions would be mostly same even though the implementations and functionality might differ.

Describe perceived obstacles and challenges, and how you plan to overcome them.

For each function there will be the following challenges:

- Each function can be overloaded to perform multiple tasks
- Developing tests for each task and comparing with similar functions to ensure competence in search and aggregation times
- Documenting the functions, it's various usages and application scenarios

Other than these specific challenges some overall challenges that I might face are:

- Building the package structure for a comprehensive and official package for the R language.(I imagine the level of details will have to be far greater than the hard task.)
- I believe testing and developing tests would take up most of the time. Identification of edge cases and implementation scenarios would be key in getting the project completed in time.

Timeline

Provide a detailed timeline of how you plan to spend your summer. Don't leave testing and documentation for last, as that's almost a guarantee of a failed project.

Make sure to identify what you plan to accomplish in each month of the project. Google has asked organizations to be more directive about milestones. Also be sure to identify what you will accomplish between acceptance and the official start of coding. (Hint: establishing your build environment and researching existing work shouldn't wait till project start)

Community Bonding Period (May 8 - June 1)

Week 1 - 2 (May 8 - May 22):

- Setting up the environment (includes setting up the package structure and making sure everything is working)
- Familiarizing myself with the codebase
    - Fixing pre existing bugs
    - Exploring the project structure and where each function is
- Getting familiar with the maintainers and the people involved

Week 3 + 3 days:

- Discussing the project details
- Creating a feature list
- Establishing specifications

Coding Period (1st part: June 2 - July 14):

Week 3 - 5 (June 2 - June 22):

- Spending the first 10 days, working on the code
- The next week is to be spent on testing and debugging
- The last few days (4 days) are to be spent on documentation

Week 6 (June 23 - June 29):

- Maintainer evaluation of the current progress
- Deciding specificatio and path to follow for the next week
- Preparing for midterm evaluation
- A short break before getting to the second function

Week 7-8 (June 30 - July 13):

- Get started on the second code
- Using the entire two weeks to write the code

Note: By this time I'll be back in uni so my hours daily would fall significantly

Week 8 (July 13 - July 19):

- Prepare for the midterm evaluation
- Working on the proposal and midterm submission
- If possible then code through that as well

Week 9 - 11 (July 20 - August 10):

- Coding, testing and documentation for the second function
- This period is longer than the time taken for the first function, since the hours I'm able to contribute per week would be lesser.

Week 12 - 14 (August 11 - September 1):

- Consolidating and testing overall project documentation and working
- The final week (August 26 - September 1) are included in this period as well.

Contingency plans

What is your contingency plan for things not going to schedule? We understand things change, but how are you planning to address changes and setbacks?

I understand that things don't always go as per schedule that's why I have accommodated various breaks and buffer periods within the plan. I have kept breaks and entire weeks just for submissions, these can be utilized as buffer. In case of contingency those buffers can be utilized for the coding and development process. This will be my first proper R package and I will give it all I can to make sure it completed properly.

If you have other time commitments that will interfere with GSoC, we highly recommend explaining how you will front-load the work before coding start or work extra early on to build a cushion!

I do have other commitments that will interfere and I have mentioned them above but in the beginning I will be able to devote most of my time to GSoC. I plan on utitlizing that time to build a cushion for later when I return to uni.

Management of Coding Project

How do you propose to ensure code is submitted / tested?

For the testing portion I plan to use custom testthat files checking for all edge cases like large input, invalid parameters etc. I'll be regularly getting these reviewed by the maintainer to ensure their relevance and to keep them as comprehensive as possible.

How often do you plan to commit? What changes in commit behavior would indicate a problem?

I plan to commit upon completion of every small feature and by completion I mean for it to be fully working and locally tested. I usually follow conventional commits while committing, too many fix commits or break in the format would indicate a problem.

Test

Describe the qualification test that you have submitted to your project mentors. If feasible, include code, details, output, and example of similar coding problems that you have solved.
For the qualification test we had to count a specified pattern within the diamonds dataset. For the easy task we had to use the grep function to count all occurrences. For the medium task we were supposed to call fread and feed it a command-line grep call using cmd to count the rows. For the hard task we were asked to write a package with Rd files tests and vignettes.

click here to go the task

Easy Task

I used the grep() function to identify rows in the diamonds dataset that match the pattern 'VS'.

```R
# Load required packages
library(ggplot2)
library(data.table)

# Load the diamonds dataset
data(diamonds)

# Convert the data to a character format for pattern matching
diamonds_text <- apply(diamonds, 1, paste, collapse = ",")

# Use grep to find rows with pattern 'VS'
matching_rows <- grep("VS", diamonds_text)

# Count the matching rows
count_vs <- length(matching_rows)
print(count_vs)
```

Medium Task

I used the system's grep command through fread() as specified:

```R
# Load required packages
library(data.table)
library(ggplot2)

# Load the diamonds dataset
data(diamonds)

# Write the data to a temporary CSV file
tmp_file <- tempfile(fileext = ".csv")
write.csv(diamonds, file = tmp_file, row.names = FALSE)

# Use fread with cmd parameter to count rows with 'VS'
vs_rows <- fread(cmd = paste0("grep 'VS' ", tmp_file))
```

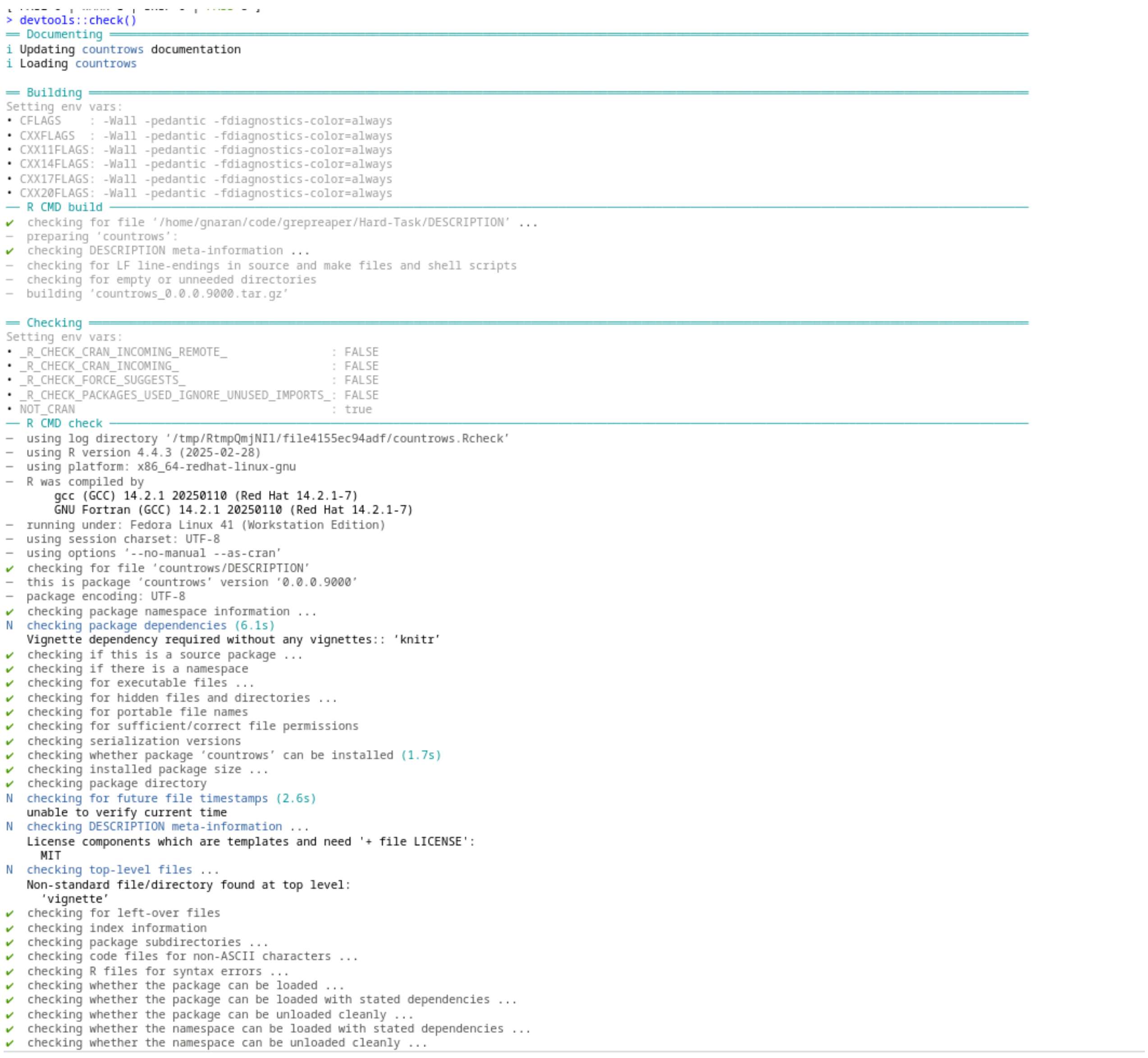
```r
# Count the rows
count_vs <- nrow(vs_rows)
print(count_vs)

# Clean up
unlink(tmp_file)
```

This method is more efficient for large datasets as it leverages the system's grep command, which is optimized for pattern matching operations.

Hard Task

For the hard task I tried to implement parallel processing. I wanted to split the data into chunks and I run grep for all chunks simultaneously. This approach in theory should've made the function faster and more stable for implementing but while implementing it ended up being counter intuitive and slowed the function. So in the end I ended up using a direct grep command with a lot of error handling. I did run devtools::check() and devtools::test() on it. Here are the results:



```
[illegible]
> devtools::check()
══ Documenting ═══════════════════════════════════════════════════════════════════
i Updating countrows documentation
i Loading countrows

══ Building ══════════════════════════════════════════════════════════════════════
Setting env vars:
• CFLAGS    : -Wall -pedantic -fdiagnostics-color=always
• CXXFLAGS  : -Wall -pedantic -fdiagnostics-color=always
• CXX11FLAGS: -Wall -pedantic -fdiagnostics-color=always
• CXX14FLAGS: -Wall -pedantic -fdiagnostics-color=always
• CXX17FLAGS: -Wall -pedantic -fdiagnostics-color=always
• CXX20FLAGS: -Wall -pedantic -fdiagnostics-color=always
── R CMD build ───────────────────────────────────────────────────────────────────
✔  checking for file '/home/gnaran/code/grepreaper/Hard-Task/DESCRIPTION' ...
─  preparing 'countrows':
✔  checking DESCRIPTION meta-information ...
─  checking for LF line-endings in source and make files and shell scripts
─  checking for empty or unneeded directories
─  building 'countrows_0.0.0.9000.tar.gz'

══ Checking ══════════════════════════════════════════════════════════════════════
Setting env vars:
• _R_CHECK_CRAN_INCOMING_REMOTE_               : FALSE
• _R_CHECK_CRAN_INCOMING_                      : FALSE
• _R_CHECK_FORCE_SUGGESTS_                     : FALSE
• _R_CHECK_PACKAGES_USED_IGNORE_UNUSED_IMPORTS_: FALSE
• NOT_CRAN                                     : true
── R CMD check ───────────────────────────────────────────────────────────────────
─  using log directory '/tmp/RtmpQmjNI1/file4155ec94adf/countrows.Rcheck'
─  using R version 4.4.3 (2025-02-28)
─  using platform: x86_64-redhat-linux-gnu
─  R was compiled by
       gcc (GCC) 14.2.1 20250110 (Red Hat 14.2.1-7)
       GNU Fortran (GCC) 14.2.1 20250110 (Red Hat 14.2.1-7)
─  running under: Fedora Linux 41 (Workstation Edition)
─  using session charset: UTF-8
─  using options '--no-manual --as-cran'
✔  checking for file 'countrows/DESCRIPTION'
─  this is package 'countrows' version '0.0.0.9000'
─  package encoding: UTF-8
✔  checking package namespace information ...
N  checking package dependencies (6.1s)
   Vignette dependency required without any vignettes:: 'knitr'
✔  checking if this is a source package ...
✔  checking if there is a namespace
✔  checking for executable files ...
✔  checking for hidden files and directories ...
✔  checking for portable file names
✔  checking for sufficient/correct file permissions
✔  checking serialization versions
✔  checking whether package 'countrows' can be installed (1.7s)
✔  checking installed package size ...
✔  checking package directory
N  checking for future file timestamps (2.6s)
   unable to verify current time
N  checking DESCRIPTION meta-information ...
   License components which are templates and need '+ file LICENSE':
     MIT
N  checking top-level files ...
   Non-standard file/directory found at top level:
     'vignette'
✔  checking for left-over files
✔  checking index information
✔  checking package subdirectories ...
✔  checking code files for non-ASCII characters ...
✔  checking R files for syntax errors ...
✔  checking whether the package can be loaded ...
✔  checking whether the package can be loaded with stated dependencies ...
✔  checking whether the package can be unloaded cleanly ...
✔  checking whether the namespace can be loaded with stated dependencies ...
✔  checking whether the namespace can be unloaded cleanly ...
```

```
N  checking top-level files ...
   Non-standard file/directory found at top level:
     ‘vignette’
✔  checking for left-over files
✔  checking index information
✔  checking package subdirectories ...
✔  checking code files for non-ASCII characters ...
✔  checking R files for syntax errors ...
✔  checking whether the package can be loaded ...
✔  checking whether the package can be loaded with stated dependencies ...
✔  checking whether the package can be unloaded cleanly ...
✔  checking whether the namespace can be loaded with stated dependencies ...
✔  checking whether the namespace can be unloaded cleanly ...
✔  checking loading without being on the library search path ...
✔  checking dependencies in R code (966ms)
✔  checking S3 generic/method consistency ...
✔  checking replacement functions ...
✔  checking foreign function calls ...
✔  checking R code for possible problems (1.5s)
✔  checking Rd files ...
✔  checking Rd metadata ...
✔  checking Rd line widths ...
✔  checking Rd cross-references ...
✔  checking for missing documentation entries ...
✔  checking for code/documentation mismatches (499ms)
✔  checking Rd \usage sections ...
✔  checking Rd contents ...
✔  checking for unstated dependencies in examples ...
─  checking examples ... NONE
✔  checking for unstated dependencies in ‘tests’ ...
─  checking tests ...
✔  Running ‘testthat.R’ (1.6s)
✔  checking for non-standard things in the check directory
✔  checking for detritus in the temp directory

   See
     ‘/tmp/RtmpQmjNI1/file4155ec94adf/countrows.Rcheck/00check.log’
   for details.

── R CMD check results ─────────────────────────────────────────────── countrows 0.0.0.9000 ────
Duration: 19.8s

❯ checking package dependencies ... NOTE
  Vignette dependency required without any vignettes:: ‘knitr’

❯ checking for future file timestamps ... NOTE
  unable to verify current time

❯ checking DESCRIPTION meta-information ... NOTE
  License components which are templates and need '+ file LICENSE':
    MIT

❯ checking top-level files ... NOTE
  Non-standard file/directory found at top level:
    ‘vignette’

0 errors ✔ | 0 warnings ✔ | 4 notes ✖
> devtools::test()
ℹ Testing countrows
✔ | F W  S  OK | Context
✔ |   1      8 | count_patterns
────────────────────────────────────────────────────────────────────────────────────────────────
Warning (test_count_patterns.R:27:3): count_rows_with_pattern handles missing ggplot2
`with_mock()` was deprecated in testthat 3.3.0.
ℹ Please use `with_mocked_bindings()` instead.
Backtrace:
    ▆
 1. └─testthat::with_mock(...) at test_count_patterns.R:27:3
────────────────────────────────────────────────────────────────────────────────────────────────

══ Results ═════════════════════════════════════════════════════════════════════════════════════
[ FAIL 0 | WARN 1 | SKIP 0 | PASS 8 ]
> 
```