MAY 08, 2023

OPEN ACCESS

DOI:
dx.doi.org/10.17504/protocols.io.j8nlkwpdxl5r/v1

External link:
https://github.com/markziemann/enrichment_recipe

Protocol Citation: Mark Ziemann, Anusuiya Bora 2023. A recipe for extremely reproducible enrichment analysis. **protocols.io** https://dx.doi.org/10.17504/protocols.io.j8nlkwpdxl5r/v1

Protocol status: Working
We use this protocol and it's working

Created: Mar 30, 2023

Last Modified: May 08, 2023

PROTOCOL integer ID: 79725

A recipe for extremely reproducible enrichment analysis

Anusuiya
Mark Ziemann[1], Bora[1]

[1]Deakin University, Geelong, Australia.

Mark Ziemann

ABSTRACT

Enrichment analysis is a popular computational biology technique for interpreting omics data, but typically these are conducted irreproducibly with web-based and graphical interface tools, which risks omitting important methodological information. To enable complete reproducibility, the analysis needs to be conducted non-interactively, recording the versions of all dependancies. This is achieved using an **Rmarkdown** script running inside a **docker container**. This allows all instructions to complete the workflow in a sequence, including parameters which sometimes are not described in methods sections. Rmarkdown and other literate programming approaches are useful for such workflows because the end result combines code, outputs (like charts and tables), together with free text, which can be used for extended descriptions of experiment design, input data, interpretation of results, etc. Using R allows to leverage the large ecosystem of bioinformatics software in CRAN and Bioconductor repositories. Containerisation with docker allows packaging of code, data and environment into a single reproducible unit. This means the workflow can be run on different types of computers (windows PC, Mac, server, cloud, etc) and yield the same result. This protocol also guides users through other best practices in computational research such as **source control, documentation and data archiving**. This protocol is designed for Linux users who want to modify and remix the provided templates to undertake their own enrichment analysis. It requires a moderate level of shell scripting, some knowledge about docker containers, and moderate R scripting.

Keywords: enrichment analysis, Rmarkdown, bioinformatics, Docker, pathway analysis, gene ontology, clusterProfiler, Reactome

MATERIALS

An internet-connected computer with Ubuntu 22.04 LTS installed. This guide might also work for other Ubuntu versions or similar Linux distributions like Debian or Mint (untested). You will need "sudo" or administrator permission to install and run docker.

On the hardware side:

- CPU with 2 or more threads
- 8 GB of available system memory
- 20 GB of available system storage

You'll also need to create an account for the following services:

- https://github.com/
- https://hub.docker.com/

Also you will need some omics data to analyse. In this protocol we will begin with RNA-seq count data, but it can be customised to work with a differential expression table or even a list of genes.

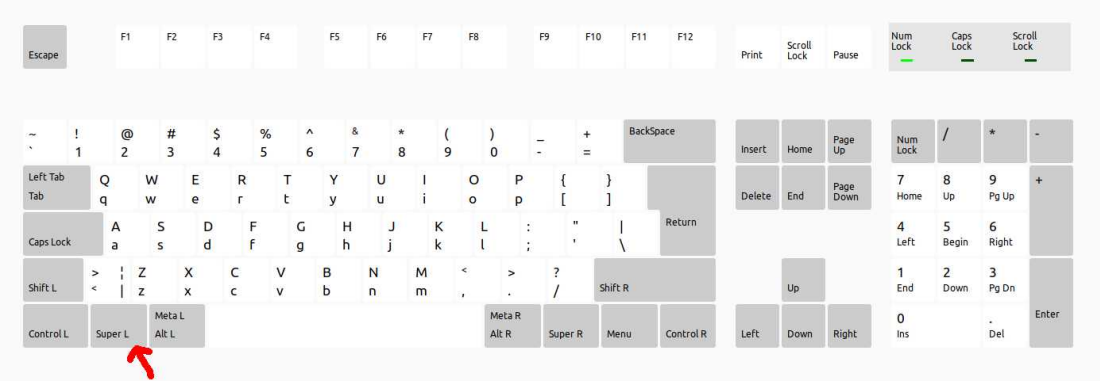
Install Docker

7m

1 1m

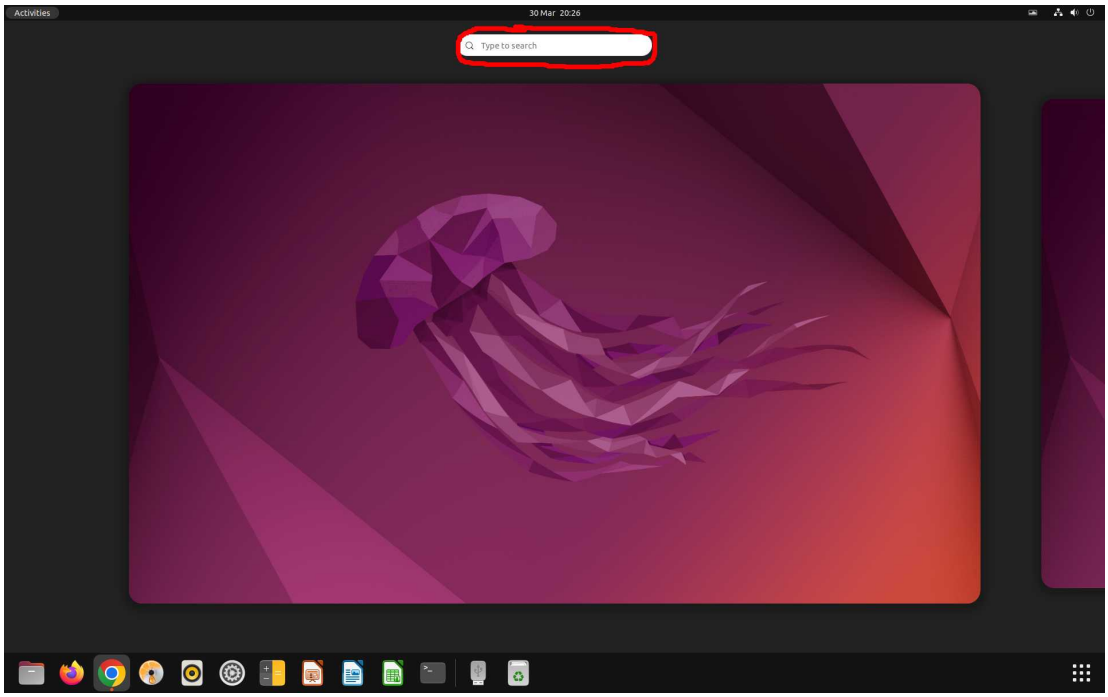
Open a terminal. Docker is a tool for working with containers, including building and running them. It is essential for ensuring reprducibility. We will install it using the command line interface, also known as the "terminal". We're assuming this is the first time using Ubuntu, so you'll need know how to open a terminal. This step only applies the the Ubuntu desktop, if you are using a server and only get a command line interface you can skip to the next step.

In the Ubuntu desktop, the terminal can be accessed from the applications menu, which can be accessed by the "super" key, which is located between the ctrl and alt keys on a standard English (US) keyboard.



Keyboard layout indicating the "Super" key for accessing the "Activities overview"

Now that you're in the activities overview, you can enter "terminal" in the search box, and the terminal app should appear.



The activities layout has a search box that allows you to quickly find apps and files.

Click on the terminal app icon to get a terminal window. You can also drag the terminal icon to the dock to keep it handy for later.

If you prefer keyboard shortcuts, ctrl + alt + t will also get you a terminal window.

2 **Update the system and install Docker.** Once in the terminal, type the following to update the sources and update all currently installed packages. Note that you will need "sudo" permissions for this to work. 5m

```
sudo apt update && sudo apt upgrade -y
```

When doing this, it is also good practice to remove unneeded packages with the following:

```
sudo apt autoremove -y && sudo apt autoclean -y
```

Now we can install the docker engine:

```
sudo apt install docker.io
```

This will install a stable version of docker that is packaged in the Ubuntu apt package manager.

3 **Add users to the docker group.** In order to use docker without constantly needing to use "sudo", it is recommended to add yourself and any other intended docker users to the docker group. 1m

```
sudo usermod -aG docker ${USER}
```

But this will only take effect for new logins, so if you want to try docker immediately you should type in the following which will log you in again.

```
su - ${USER}
```

Type the "groups" command in to check that you are member of the docker group.

```
groups
```

Validate the example workflow 14m

4 As docker is now installed, it is now possible to try to reproduce the example workflow described in our article. The first step is to fetch the workflow from dockerhub. 3m

```
docker pull mziemann/enrichment_recipe
```

5 Now run the docker container, getting a bash prompt. 1m

```
docker run -it --entrypoint /bin/bash mziemann/enrichment_recipe
```

If you get a permission denied error at this stage, it could be that you don't have membership in the docker group. Go back and complete the commands in step 3, then try again.

If it has started the container, run 'ls' here, you will see the project's Rmd scripts and the Reactome
gmt file.

6 Now open R. 5m

```
R
```

Once inside R, run the example.Rmd script.

```
rmarkdown::render("example.Rmd")
```

This will regenerate the analysis that was shown in the article. The report is saved to "example.html".

If the workflow completed successfully, you can exit R with "q()" and the container with "exit".

7 Visually examine the html file. It is easiest to do this outside of the container. This command copies the newly created html from the last running container into the current directory. 5m

```
docker cp $(docker ps -aql):/enrichment_recipe/example.html .
```

From there, you can use your favourite web browser to view the report.

```
firefox example.html
```

Create a GitHub repository 29m

8 **Install git.** git is a source control system that allows users to manage versions of software over time. Use the following code to install git on your local machine: 1m

```
sudo apt install git -y
```

9 **Create an account for GitHub.** GitHub.com is a website that hosts these software repositories, which has the benefit of making the code widely available. Visit the website and create an account. 5m

10 **Set up SSH keys for GitHub.** GitHub requires authentication with SSH keys. SSH (Secure Shell) keys are an access credential that is used in the SSH protocol. Using SSH keys is much more secure than a password because the SSH key file is equivalent to a password with >1000 characters. We will use the SSH protocol to generate public and private keys. Keep the private keys secret, and upload the public key to GitHub. If you don't have any ssh keys in your ~/.ssh directory, type the following to generate them: 5m

```
ssh-keygen
```

At the prompt it will ask about the location. Continue with the default location. It will ask about whether you want password to protect the keys. It may increase security to add a password. If

you don't want a password, just leave it blank and hit enter. It will then generate a randomart and a fingerprint. You can safely ignore that. Next, copy the public key text. You can do this from the terminal with the following:

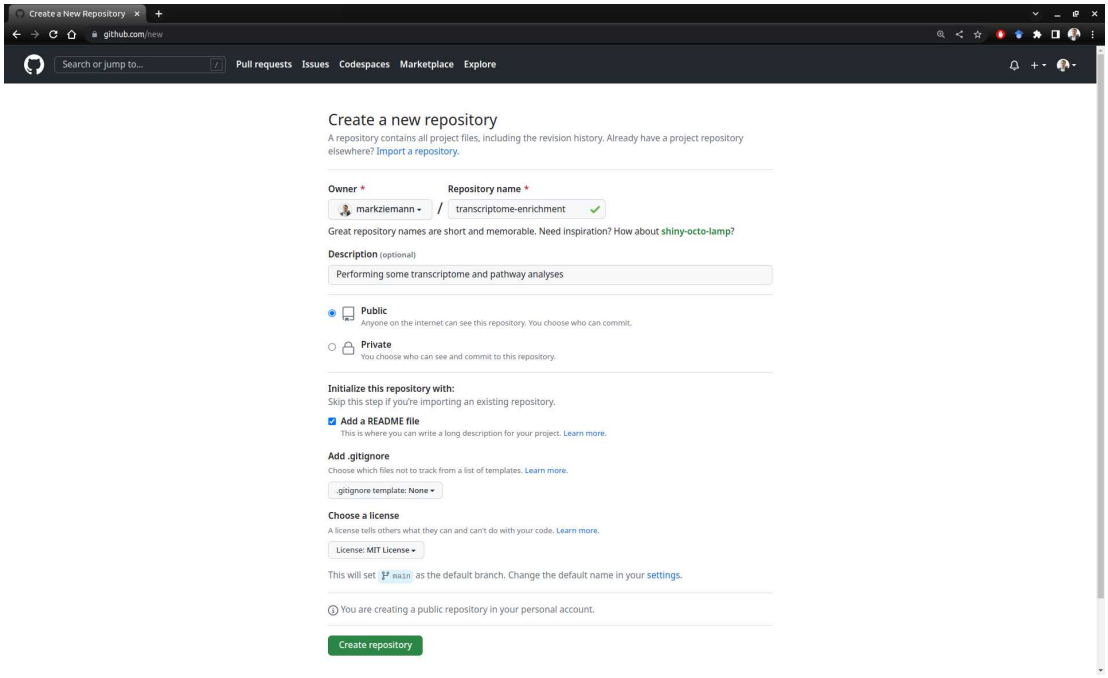
```
cat ~/.ssh/id_rsa.pub
```

It will show the contents of the public key. Now return to GitHub.com, log in and in the top right of the window you will find the personal menu, click on "Settings". This will bring up a whole range of settings on the left side of the screen. Under the "Access" heading you will see "SSH and GPG keys". Click on that then you will see a green button labeled "New SSH key". Hit it and paste the public key in the designated area and hit the green "Add SSH key" button. Do not upload the private key.

11 **Create the repository.** A simple way to get git/GitHub working is to create a repo with Github and then clone it to the local computer. Return to your github personal profile page, eg: https://github.com/yourname and click on the tab called "Repositories". Click the green button labeled "New". This will open up a new screen where you can set some options. 5m

1. It is important to give your repository a good, descriptive, but short name. Avoid spaces.
2. Provide a one sentence description of your repository based on its purpose.
3. You can select to keep the repository private or to make it public. Public is easier to work with if you don't mind that other people could be snooping at your work in progress.
4. It is recommended to add a README file to the new repo which can be worked on later.
5. Ignore the .gitignore for now (this is a list of file types that might be in the project working directory but we don't want to have tracked by git).
6. Choose a license. If you want your code to be made available to others with out restrictive conditions you can pick a license such as the MIT license. If there is no licence in the repo, then it is assumed to be covered by copyright.

Once you are happy with your selections, hit the "Create repository" button



When creating a Github repository, select the options carefully.

Then you will be greeted with the repository page.

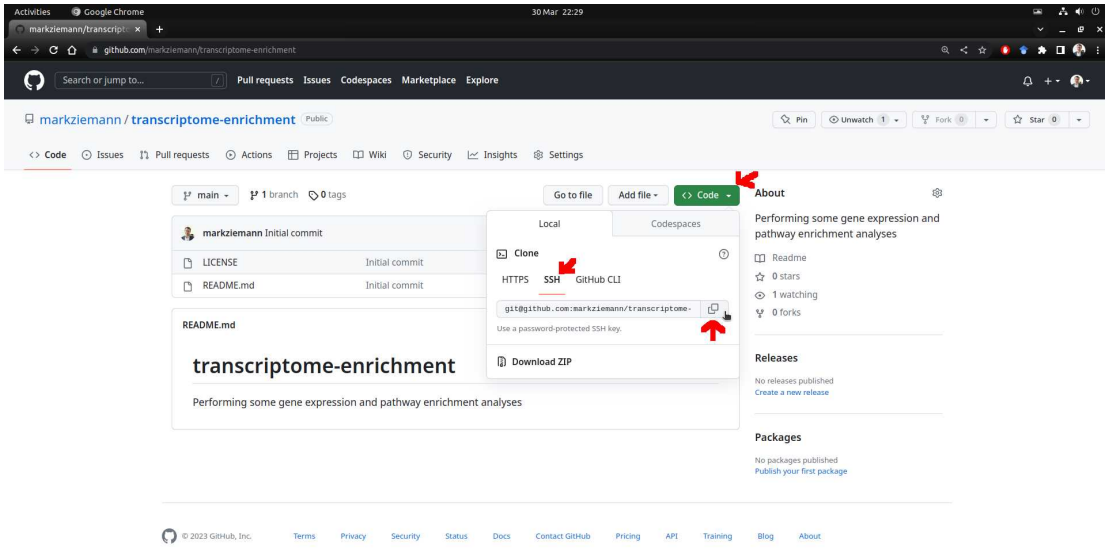
12 **Start working with the repo locally.** So far the repository exists on Github.com but not on the local machine. Fix that by cloning the repo. You probably don't want to do this in the home directory, rather in an area where you work. My work folder is called "projects". You can create a projects folder with the following command: 3m

```
mkdir  ~/projects
```

Then change to this directory:

```
cd ~/projects
```

Now clone the repository. Click the green "<> Code" button and you will see that cloning can be done three different ways. We will be using "SSH". Copy the git URL as indicated in the image below:



When cloning your github repo, use the SSH option and copy the git address.

Go back to your terminal and type "git clone" and then paste your git URL so it looks like this:

```
git clone git@github.com:yourname/your-repo-name.git
```

And git should be able to complete the task for you.

13 **Try a commit/push.** If you are new to git, then it can be a bit confusing. So it is best to try modifying the repo now, before working with code or data. The first thing to do is to change directory (cd) into the project folder, then you can list the contents (ls) and you will see the README and LICENSE if you selected one. 10m

```
cd your-repo-name
ls
```

At this point I would recommend adding another few sentences to the README to explain more about the project background. To edit files in the terminal you can use a program called nano:

```
nano README.md
```

Use the arrow keys to navigate to the bottom of the file and use the enter key to add more lines if necessary. Enter the sentences about the project here. If you want to add lots more information like subheadings, links, images, tables and other stuff, all of that is possible with the markdown syntax, and can be found on this "cheatsheet". Once you're finished adding content to the README, exit nano using Ctrl + X. It will prompt you to save, where you should select "y" and you will return to a normal command prompt.

Although the README file is updated, the changes haven't been recorded on the local repository nor on Github. In order to update the local git repository we use the following commands:

```
git add README.md
git commit -m "added some details to the README"
```

The git add command specifies to git that the file should be tracked for changes, and commit recognises these changes and stores them in the repository.

If you get an error like "Please tell me who you are", follow the instructions in the error message to specify to git your user.email and user.name. I'd recommend providing the email address used to create the Github account and the Github username.

In order to make these changes effective on the "origin" repository on Github, we need to "push":

```
git push origin main
```

At this step, git will use the SSH private key for authentication and if all goes well you will be able to visit the Github repo URL and see that the changes you made with nano have propagated there.

Build a custom docker image for your project

45m

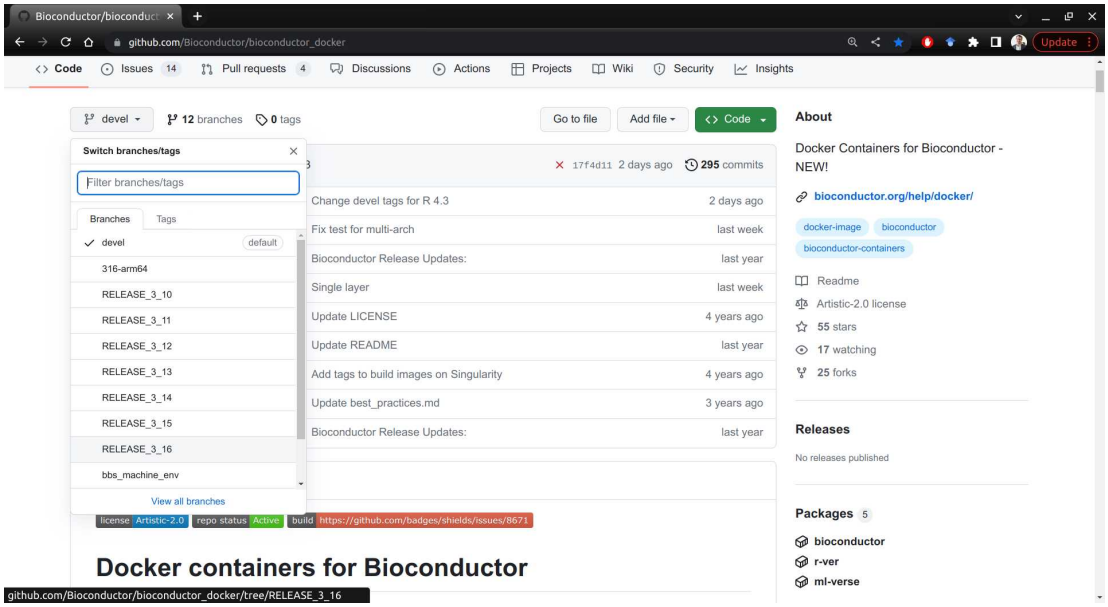
14 **Understand the Dockerfile and modify it to your needs.** In this step you will build a docker image for your project. This will contain the operating system, R/Rstudio and the packages you need for your analysis. I have provided a Dockerfile you can use as a template, and remix as you need, for example by adding and removing R packages or changing the bioconductor version. Copy the contents of the Dockerfile (text below), and then create a new file in your project directory called "Dockerfile" using the command "nano Dockerfile". Paste the text using ctrl + shift + v. 20m

Now I will walk you through the dockerfile which consists of just 6 commands.

```
# Docker inheritance
# this is a ubuntu image with R, Rstudio and bioC
FROM bioconductor/bioconductor_docker:RELEASE_3_16

# Update apt-get
RUN apt-get update \
  && apt-get install -y nano git \
  ## Install the python package magic wormhole to send files
  && pip install magic-wormhole \
  ## Remove packages in '/var/cache/' and 'var/lib'
        ## to remove side-effects of apt-get update
  && apt-get clean \
  && rm -rf /var/lib/apt/lists/*

# Install required CRAN packages
RUN R -e
'install.packages(c("kableExtra","vioplot","gplots","eulerr"))'

# Install required Bioconductor package
RUN R -e
'BiocManager::install(c("getDEE2","DESeq2","fgsea","clusterProfiler"
,"mitch"))'

# Clone the repository that contains the research code and execute
it
RUN git clone https://github.com/markziemann/enrichment_recipe.git

# Set the container working directory
ENV DIRPATH /enrichment_recipe WORKDIR $DIRPATH
```

Note that each line starting with a hash (#) is a comment, so it is ignored by the program, but can contain some useful information for the user.

The first command starting in "FROM" specifies the base image. In this case it is the Bioconductor release version 3.16, which has R version 4.2.2. These are maintained by the Bioconductor team and the Dockerfiles are kept here. When making your own docker image, you are suggested to use the most up-to-date release available. That means taking a look at what Bioconductor images are available. Do not use a "devel" branch. On the Bioconductor_docker page, there are several branches, each representing a different version. There will be a "devel" image and several "release" versions, as shown by the image below.



Bioconductor project maintains several branches of Dockerfiles that will install R, Rstudio and Bioconductor.

The next command in the Dockerfile installs some system utilities that may be useful including nano, git and magic-wormhole. Nano is for text file editing, git is for source control, and magic-wormhole is for transferring files between computers. This command also demonstrates how to install utilities with the apt and pip package managers, so you can easily add your favourite utilities. If you don't need any such utilities, the command can be safely deleted.

The next command installs some packages from CRAN that might be useful. You can add your favourite CRAN packages to this. The packages included in this Dockerfile are kableExtra for creating nice-looking tables, vioplot for creating violin plots, gplots for simple heatmaps, and eulerr which makes nice Euler diagrams.

The next command uses BiocManager::install() to install Bioconductor packages. The packages it installs are:

- getDEE2: which is a programmatic tool for fetching RNA-seq data from the dee2.io resource. It is used to fetch data in the example workflow, but you might not need it for your project if you have your own data file.
- DESeq2: a widely used differential expression tool.
- fgsea: a widely used tool for functional class sorting, similar to GSEA.
- clusterProfiler: a widely used tool for over-representation analysis.
- mitch: a tool for multi-contrast enichment analysis.

Again, you may need to add and remove packages here to your project's requirements.

The next step runs a command to clone your git repository. This ensures that the docker image will already have a copy of your code when you launch a container. You will need to change this line so it matches your Github alias and the name of your Github repo. Note that this command

will make a clone of the repo in the root directory, which will be used later as the project working directory.

The last command in the Dockerfile sets the working directory, you should change this to the name of your Github repo.

15 **Think about how you will link your data and code.** Linking is ensuring that the code "knows" where to obtain the data from, so that users don't get "file not found" errors. The template workflow provided uses public data that is fetched from a website. It is likely that you will need to change this so that you can analyse your own gene expression data. There are a few options worth discussing: 10m

1. Include the data in the Github repo. If the data is very small, like a simple list of genes in a text file or an RNA-seq expression count matrix, then it might be appropriate to commit and push these to the Github repo. This only works well if the total data size is less than 100 MB, as large files can bloat the repo and make it slow to work with.

2. Copy the data into the Docker image during the build. If the data is a bit larger, it might be viable to copy it into the Docker image during the build process with the following command:

```
COPY /source/file/path  /yourreponame
```

Where you will need to change /source/file/path to the actual path of the file. For example if the current working directory is "/home/john/enrichment1", the data file is called "genelist1.txt", and the repo/project name is "myenrichment", the command would look like this:

```
COPY /home/john/enrichment1/genelist1.txt  /myenrichment
```

3. Deposit the data to an archive. While the above two options are the most convenient, this is the most robust option for long term reproducibility. Data archives like Zenodo and FigShare will accept large general data sets, and NCBI Sequence Read Archive and Gene Expression Omnibus will accept high throughput sequence and microarray data. This has the dual benefit of enhancing data reuse.

Of the above options, only option 2 will require you to make any changes to the Dockerfile.

16 **Build the new docker image.** Use the following command to build the new image. Where it says "yourname", put your Dockerhub alias. Where it says "yourprojectname", put the name of your project. I'd suggest the same name as your Github repo. 30m

```
docker build -t yourname/yourprojectname .
```

It may take up to 25 minutes to build it.

To confirm that the image has been built successfully, you can run the following command to view the available images.

```
docker images
```

17 **Check that the docker image works.** The following command will run the image in a container and give you a bash shell prompt. 10m

```
docker run -it --entrypoint /bin/bash yourname/yourprojectname
```

Here are the things you should check:

- The present working directory is indeed your cloned repo, and not the root directory
- That R can be opened, by typing "R" on the command line (record the version).
- That some of the CRAN and Bioconductor packages can be loaded using the library() command.

Once verified, quit R with the "q()" command and once back on the shell prompt type "exit" to leave the docker container.

18 **Commit and push the Dockerfile to your repo.** Now that we know that the Dockerfile works, it is a good time to add it to your Github repo. If you are still working in the project (git) directory, you can run the following to update the Github repo. Just to refresh your memory these commands do the following: 5m
Pull: sync your local machine with GitHub. This is especially important if you are working in a team, or you like to make changes to the GitHub repo using the browser.
Add: This will activate track changes for the files selected.
Commit: Will recognise and confirm the changes made locally.
Push: Will propagate changes from local to Github repos, so they are in sync again.

```
git pull
git add Dockerfile
git commit -m "adding dockerfile"
git push origin main
```

Customise the Rmarkdown workflow to your needs

12h

19 **Understand the structure of the Rmd file.** In this section I will be walking you through the process of adapting the provided template Rmd workflow for your needs. If you are new to this sort of thing, then stay with me as I dissect the structure of the Rmd and describe its working parts. If you are an advanced Rmarkdown user, you can skip over this. 10m

Rmarkdown is a literate programming script language that combines R together with markdown, a lightweight markup language. It allows us to combine documentation together with R code and the results of the R analysis such as charts and tables. The ability to add documentation is really important as it allows us to write thorough descriptions of the analysis such as background, methods, findings, conclusions and bibliography. In fact it makes it possible to write a whole manuscript, just using Rmarkdown (and this is becoming more popular).

Fetch a copy of example.Rmd to act as your template.

```
wget https://raw.githubusercontent.com/markziemann/enrichment_recipe/main/example.Rmd
```

Once downloaded, you can edit it with nano. If you only want to view the file, you can use the "less" command. If you want to change the file name, you can do that with the "mv" command.

```
mv example.Rmd myworkflow.Rmd
```

20 **Change the header to your needs.** The header is a section the very top of the Rmd file delineated with three hyphens "---" 10m

```
---
title: "An example of extremely reproducible enrichment analysis"
author: "Mark Ziemann & Anusuiya Bora"
date: "`r Sys.Date()`"
output:
  html_document:
    toc: true
    toc_float: true
    code_folding: hide
    fig_width: 5
    fig_height: 5
theme: cosmo
---
```

Naturally, you should change the title and author according to your project needs. The Sys.Date() command ensures that the date of execution is recorded in the final report. The other settings, like theme and figure size you should keep as-is for now until you have a working workflow.

21 **Understand markdown text.** Let's take a look at the next section in the Rmd file immediately below the header, which is markdown. 20m

```
Source: https://github.com/markziemann/enrichment_recipe

## Introduction

This guide is a Rmarkdown script that conducts differential
expression and enrichment analysis, which are very popular workflows
for transcriptome data.

This is meant to be a boilerplate template, which you can remix and
modify to suit your analytical needs.

In the code chunk below called `libs`, you can add and remove
required R library dependancies. Check that the libraries listed
here match the Dockerfile, otherwise you might get errors.
```

There is a link to the source repository, which is good practice because it allows the reader to inspect the raw code and its history. Under that, we have a subheading called introduction, as denoted by hashes. The more hashes, the smaller the subheading. These are useful to structure your work, and the Rmarkdown engine recognises these to create a table of contents. Free text can be used to write descriptions of background information, methods and results, which allows comprehensive documentation of workflows. Your reports should have introduction, methods, results and conclusions sections. A bibliography is recommended. There are many formatting options provided by markdown such as italics, bold, block quotes, numbered and bullet points, tables, and it is possible to embed images and videos.

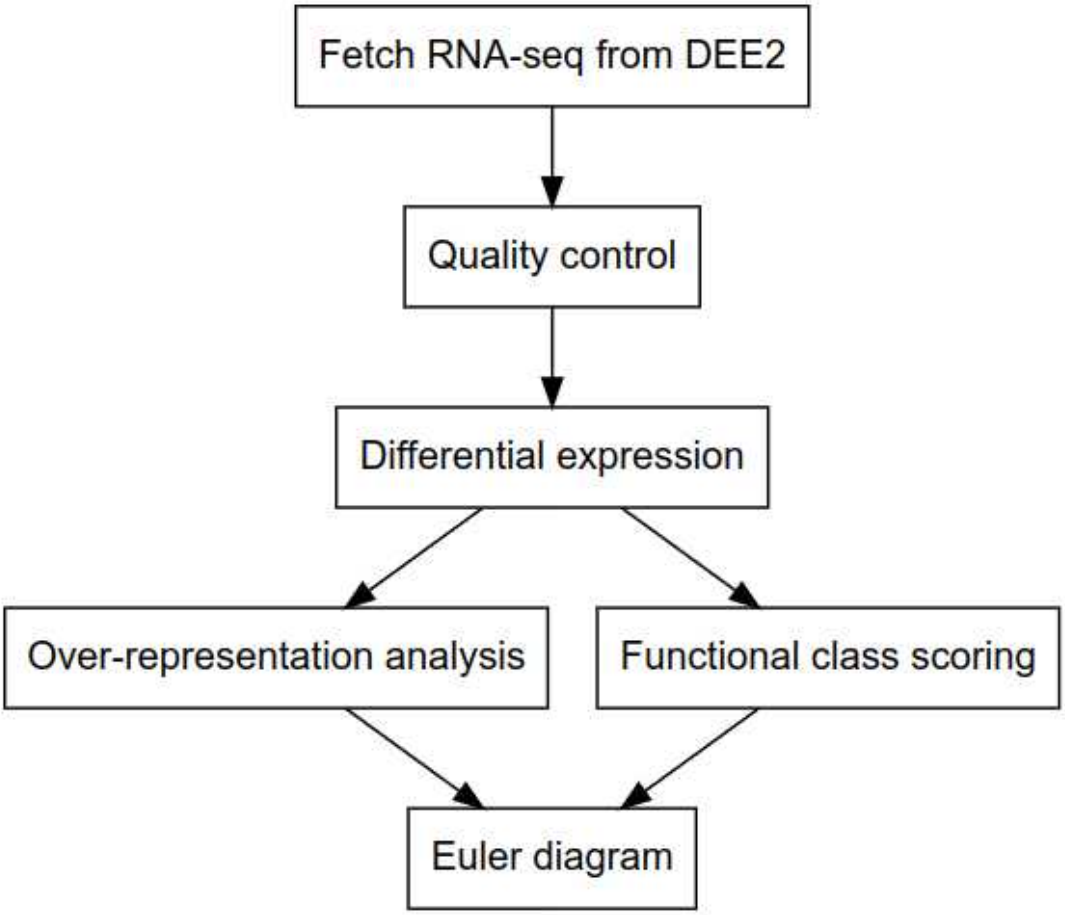
To understand the capabilities of Markdown further, visit this guide.

22 **Understand how code chunks are formatted.** R code is embedded in an Rmarkdown script inside "chunks". There can be any number of chunks. They are delineated with triple backticks (```). The chunk header comes after the triple backtics and it is bordered with curly braces ({}). Inside the braces, you will find a lower case "r" which specifies to the render engine that this is R code. There is also an option to provide other information, such as a chunk name. In the below example, the chunk name is "libs" because in that chunk, libraries (packages) are being loaded. Providing a unique name for each chunk is a good idea, and helps in troubleshooting errors if they occur. 20m

There are many options that can be specified in the chunk header, from figure size, to captions, and many other behaviours. See this guide for a deep dive into them.

````
```{r,libs}

suppressPackageStartupMessages({
  library("getDEE2")
  library("DESeq2")
  library("kableExtra")
  library("clusterProfiler")
  library("fgsea")
  library("eulerr")
  library("gplots")
})

```
````

23 **Understand the workflow.** Before you start making changes to the workflow, it is a good idea to understand the process first. The example.Rmd script contains instruction for the five-step workflow. 3h



General overview of the example.Rmd script.

1. It begins with fetching some RNA-seq data from a website called Digital Expression Explorer 2 (dee2.io), in the form of a count matrix. The dataset is described at SRA under accession SRP038101, and was selected as it shows a robust effect of treating AML3 acute myeloid leukemia cells with 5-azacytidine, an analog of cytidine which is thought to inhibit DNA methyltransferases.
2. It performs some rudimentary quality control including tabulating read counts for each sample and creating a multidimensional scaling plot (MDS). MDS is similar to principal component analysis which is used to examine the variation between samples, both inter-group and intra-group variation.
3. Then it conducts differential expression analysis with DESeq2. Just before running DESeq2, genes with fewer than 10 reads per sample are discarded. Setting a detection threshold like this is helpful for removing noise from lowly expressed genes. Genes with false discovery rate adjusted p-values (FDR)<0.05 were considered significant. The extent of differential expression was visualised with a smear plot.
4. Step 4 is parallel enrichment analysis with two different algorithms; over-representation analysis with clusterProfiler, and functional class scoring with fgsea. For clusterprofiler the significant up- and down-regulated genes were considered in separate tests using the

enricher() function. The list of all detected genes was used as the "background". The top clusterprofiler results are visualised as a barchart, but there are other visualisation options. For fgsea, the Wald statistic was used as a ranking metric. Top fgsea results are also depicted in a bar chart. Reactome pathways (downloaded 2023-03-06) were used for both ORA and FCS. Gene sets with FDR<0.05 were considered as significant.

5. Then a comparison of the two enrichment approaches is conducted. This includes an Euler diagram, which is a bit like a Venn diagram but the areas are proportional. The jaccard index is calculated, which is a measure of similarity. Finally, there are two additional visualisations, enrichment plots (from the fgsea analysis) for two strongly enriched gene sets, and heatmaps to show the expression of member genes in each sample.

At the end of the Rmd script, there is a section called "Session Information" which describes the environment including R version and version of other packages.

24 **Make targeted changes to your Rmd script.** It isn't within the scope of this protocol to delve deeply into the details of each chunk, rather understand the modular nature of these chunks. The types of modifications you might make are: 8h

- Instead of fetching from DEE2, you analyse a count matrix that you have on disk or saved on a data archive.
- Instead of RNA-seq, you have microarray data which would be better analysed with limma.
- Instead of performing differential analysis in R, you already have a list of genes in a text file which you want to perform enrichment analysis with.
- The format of the gene identifiers in the example.Rmd workflow is the ensembl identifier followed by the gene symbol separated with a space (eg: "ENSG00000165949 IFI27"). This could be different in your data, and some work may be required to adapt the code for your gene identifiers.

To do this, you can use the command line interface inside your container as detailed below. If you prefer the more visual Rstudio integrated development environment (IDE), a guide for that is given below in step 24.1.

```
docker run -it -v ${pwd}/enrichment_recipe --entrypoint /bin/bash
yourname/yourprojectname
```

Then once in the container, type "R" to get a R prompt.

In a separate window (not in a container) navigate to the project directory, use nano to open up the Rmd script.

The general process is to begin at the top of the script and customise all the sections to meet your needs, starting at the header. To ensure that the code works, copy the code chunks or individual lines from the nano window and paste them to the R prompt to execute them in sequence. There will be many modifications necessary to get your data workflow just right.

The first type of challenge is to get the data you have loaded into R. Use the suggestions in step 15. If you have uploaded the data to a persistent archive, the script can fetch it from that location with commands like "download.file()". If you have added the data file to the Github repo, it will be available to be read with functions like "read.table()", "read.csv()", "readLines()" or similar depending on the file format.

The next set of changes will be around differences in the format of the input datasets. This includes the gene identifiers, but also ensuring that your data is in the correct structure. For example DESeq2 requires the counts in a matrix, not a data frame. Commands like "str()", "class()" and "head()" can be used to diagnose the data structure.

If you're importing sets of gene identifiers for clusterprofiler, you can safely delete unnecessary sections such as differential expression and fgsea. Naturally you can add additional analytics steps to the workflow.

Throughout this process, errors will arise. Use google and other search engines to help you fix these problems. AI chatbots might also be helpful.

Take another look at the libraries that are loaded near the top of the Rmd file, and remove the ones that that are unnecessary.

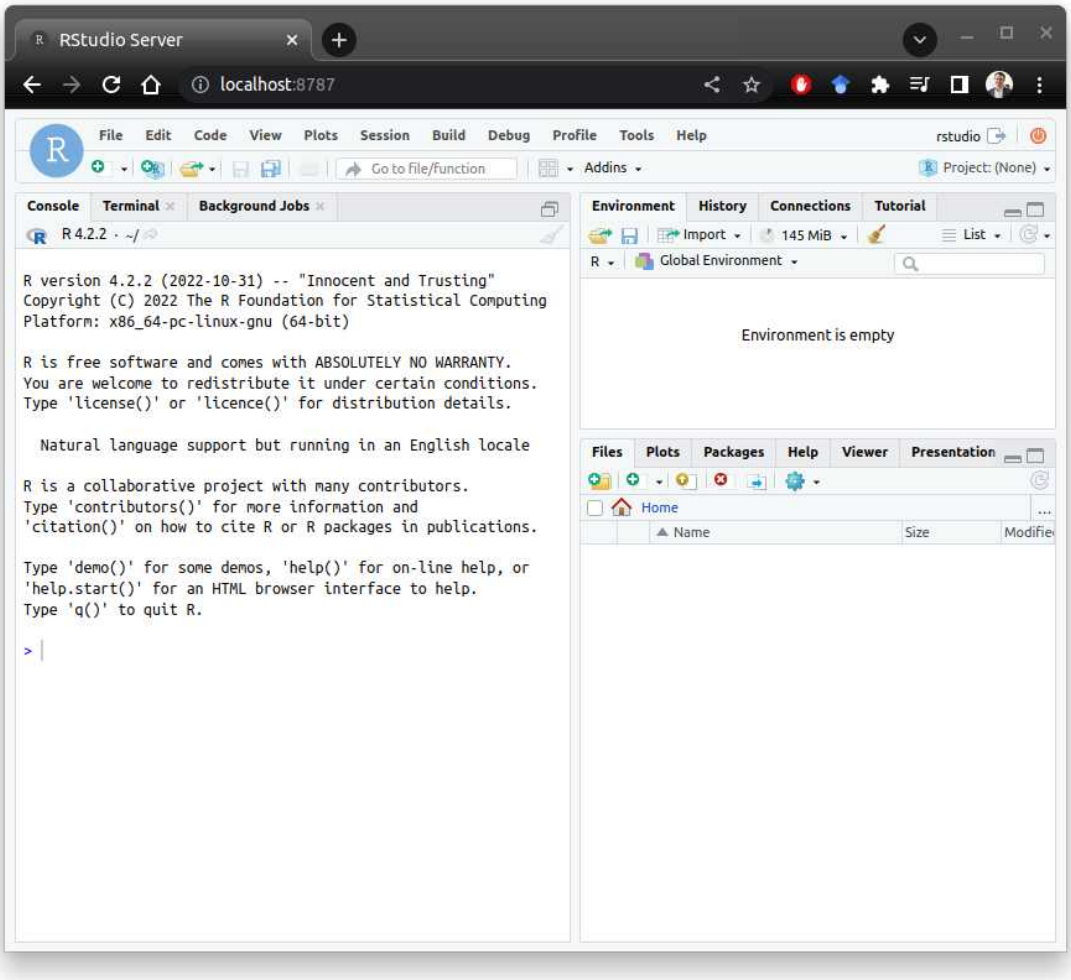
Once you are happy with the performance of each chunk, it is important to "knit" the entire Rmd script. This ensures the Rmd has no critical code errors (but doesn't guarantee the analytical validity).

24.1 **Development using Rstudio.** There are a lot of people who would rather develop their scripts with the Rstudio graphical interface. Rstudio is included in the docker image, so running it this way shouldn't be that difficult. You may select this approach as an alternative to step 24. If you prefer the command line approach, you can safely skip this step and jump to step 25.

1. Run the docker container. This will spawn a web service which can be accessed using your web browser.

```
docker run -it -v  -e PASSWORD=bioc  -p 8787:8787
mziemann/enrichment_recipe
```

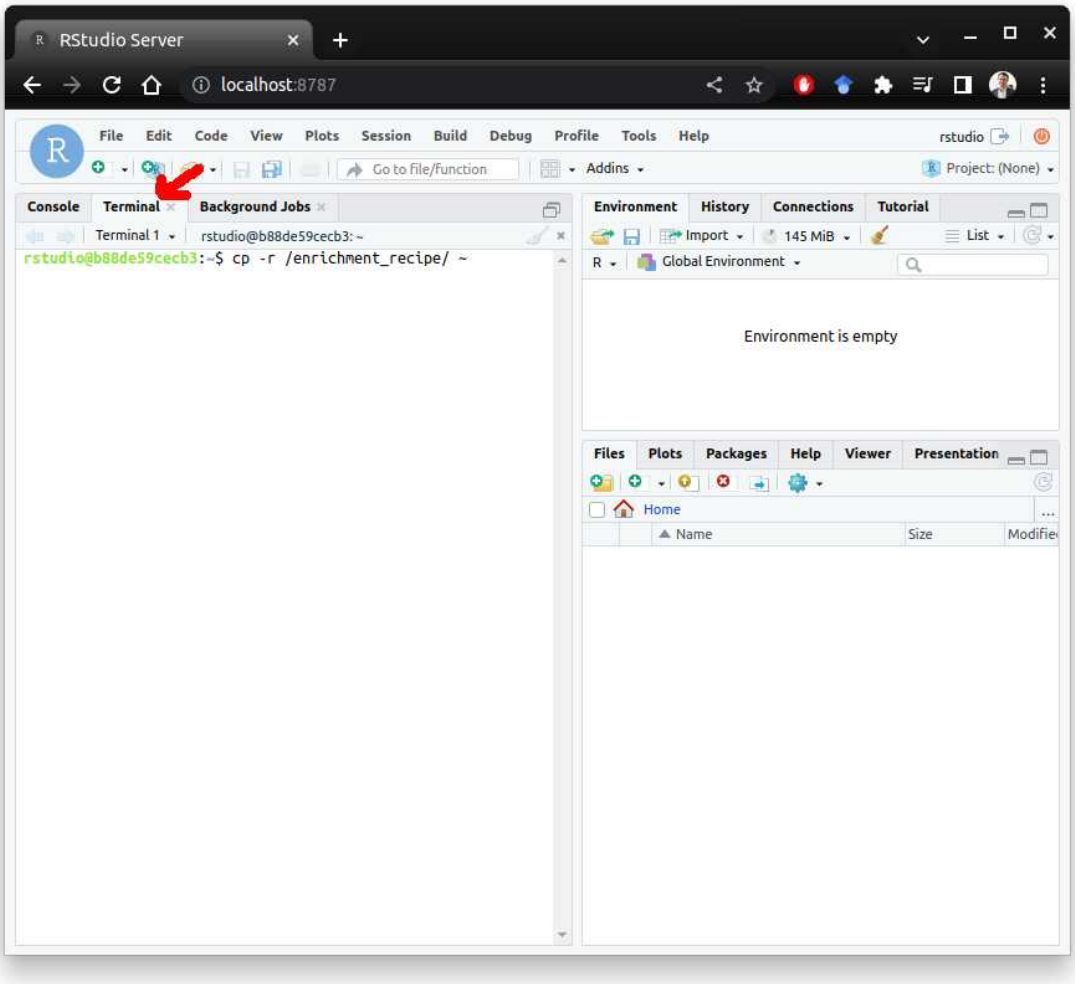
2. In a web browser, visit localhost:8787/ it will prompt you for the username "rstudio" and password "bioc". Then you will be greeted with the Rstudio window.



Rstudio window.

As you can see there are no files in the home directory.

3. To get a copy of the working script into your home directory where you can work with it we will need to use the terminal built into the Rstudio interface.



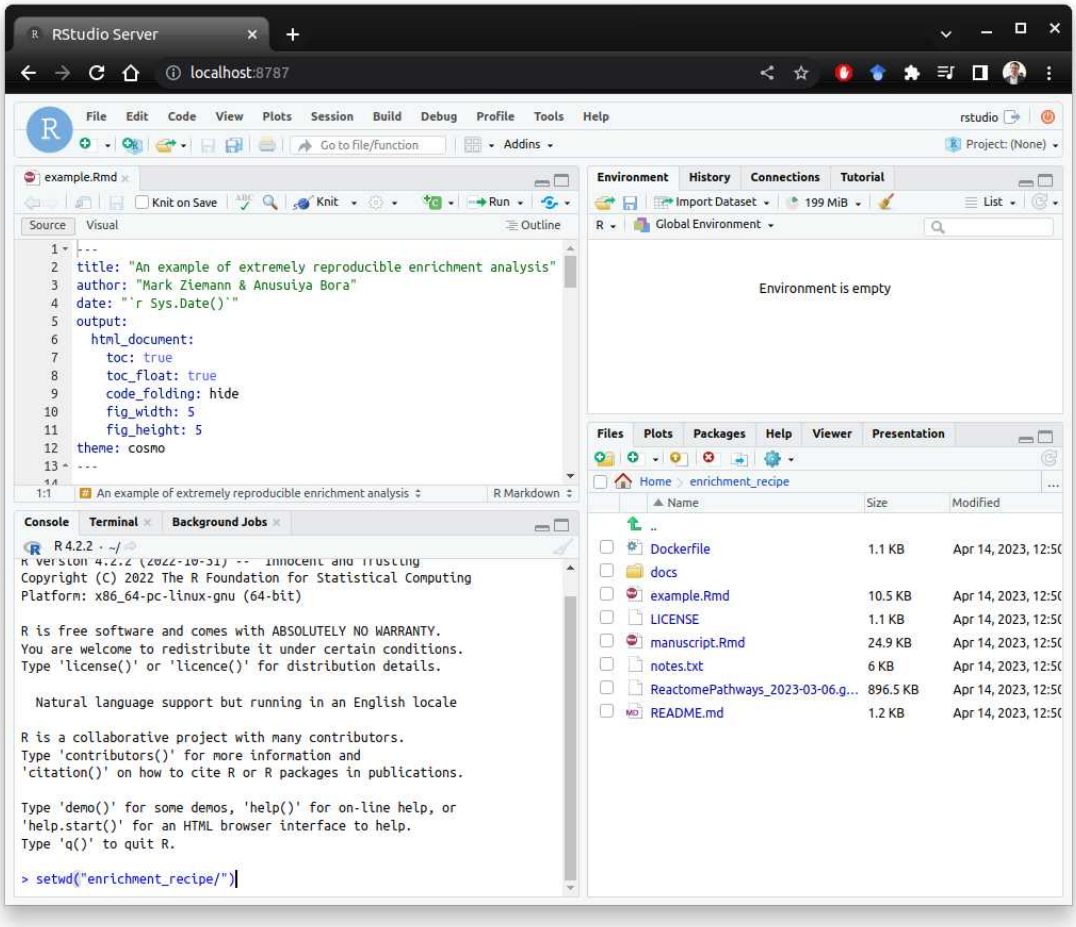
Using the terminal to copy the project folder into the home directory.

The copy command will copy the project folder into your home directory. This is necessary as the user "rstudio" does not have write permissions outside of ther home directory. Naturally if you changed the project folder name/github repo name you should use that name instead.

```
cp -r /enrichment_recipe/ ~
```

4. The folder will now appear in the lower right "Files" panel, which you can double-click on to see the contents and click on the workflow Rmd.

5. Note that the R console still thinks it is located in the home directory "~" . You should change the working directory to the project directory you just copied, so that the console can access the scripts and data.



Change the working directory to avoid "file not found errors".

6. Now you can make edits to the workflow Rmd, and take advantage of the Rstudio shortcuts to execute whole chunks or sections. Continue making amendments until the workflow is to your satisfaction. Be sure to "knit" the entire Rmd to ensure that it runs without any issues. You can also inspect the html report to ensure that all data visualistions, tables and other elements are appearing as desired.

7. Once you are happy with the appearance of the html report, we need to get it out of the container and in the project folder so we can commit and push the changes. Exit the container and use the docker cp command. Be sure to substitute you project/repo name and workflow Rmd name in the command below.

```
docker cp $(docker ps -aql):/enrichment_recipe/myworkflow.Rmd .
```

You may want to copy the html report out as well with the same approach.

Update the software repository

1h 10m

25 **Prepare changes.** Now that the Rmd script is working, it is time to push these changes to the GitHub repo. Now is a good time to update the Dockerfile to remove unnecessary packages and add newly required ones. You may want to rebuild the image, execute the Rmd and inspect the html report for validity. Also take another look at the README and use nano to make any additional updates. 1h

26 **Commit and push changes.** Use the git add command to track the changes of all files you intend to update on the github repo. Git commit to register the changes to the local repo and push to propagate the changes to GitHub. 10m

```
git add Dockerfile myworkflow.Rmd README.md
git commit -m "Rmd workflow working"
git push origin main
```

Then inspect the repo on github.com to ensure that all the neccessary files have been updated.

Push the image to Dockerhub 20m

27 This is an optional step, which uploads your docker image to Dockerhub. This is a good idea if you want to share it openly, however Dockerhub is not a guaranteed data archive and might decide to remove your image in future. If you don't want to push to dockerhub, skip to the next step. 20m

You will need to visit dockerhub.com and make an account the "yourname" alias you used to build your image previously. You will also need to run a command to let the docker command line tool know who you are:

```
docker login -u yourname -p yourdockerhubpassword
```

Next, run "docker images" to bring up a list of images. You should see the image that you built previously named "yourname/yourprojectname". Type the following to push it:

```
 docker push yourname/yourprojectname
```

Now your image is available on any other Linux computer with docker using the following command:

```
docker pull yourname/yourprojectname
```

The image will be publicly visible on Dockerhub (https://hub.docker.com/r/yourname/yourprojectname). You should visit this page and add some details such as an extended description of the purpose of the project, and a link to the GitHub repo.

See step 5 for a guide to reproducing the workflow on another system. Just substitute the names for your docker image and your Rmd script.

Long term archiving

3h

28 **Save the image.** As Dockerhub is not a guaranteed data archive, there's a good chance that over time the company in charge of Dockerhub removes images that are not monetised. To protect against this and give your work a longer reproducibility horizon, you can store your Docker image on a data archive like Zenodo. Use the following command to save the docker image to a tar.gz file. 30m

```
docker save yourname/yourprojectname | gzip > projectname.tar.gz
```

29 Before uploading to a data archive, it is a good idea to check that the tar.gz is also reproducible. Try sending the tar.gz to another computer that has docker installed for reproduction. Sending files can be done with rsync, ssh, wormhole or if you have physical access, a USB flash drive. On the second computer, run the following: 30m

```
gunzip projectname.tar.gz
docker import projectname.tar yourname/yourprojectname
```

Again, use step 5 as a guide for reproduction.

30 **Upload to Zenodo.** If you are new to Zenodo, you'll need to make a new user profile, then upload the image as a new dataset. Provide a thorough metadata about the image such as purpose, design, GitHub link and steps to reproduce so that folks in the future can understand what it's about. Include this link in you research article and it will be reproducible well into the future. 2h