Preparing Data and Code for Publication

Brooke Anderson

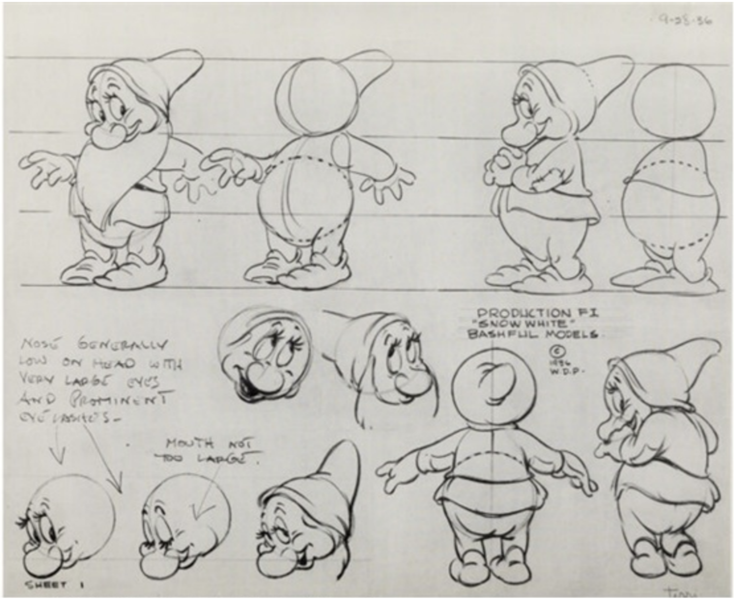
Barriers to reproducible papers

In theory, some of the barriers to making published papers computationally reproducible are:

- Necessary to know what to include and how to share it
- Data are very large
- Data cannot ethically or legally be shared (e.g. private health data)
- Necessary to know how to share important information about the enviroment in which the code was run
- Must address randomness introduced by random number generation

We will cover these topics in this session.

Barriers to reproducible papers


9-28-36
PRODUCTION F1 "SNOW WHITE" BASHFUL MODELS
© 1936 W.D.P.
NOSE GENERALLY LOW ON HEAD WITH VERY LARGE EYES AND PROMINENT EYE LASHES-
MOUTH NOT TOO LARGE.
SHEET 1

Barriers to reproducible papers

In practice, though, some of the barriers keeping people from making published papers computationally reproducible are actually:

- I don't want to share the code / data
  - The code's too messy
  - Someone might scoop my work
  - Someone might find a bug
- The code's available, but very hard to understand
  - I'll put the code up, but it's not worth taking the time to clean it first
  - Subconciously creating a barrier to prevent easy reproduction

The other sessions of the workshop have focused on approaches that address these barriers.

What to include with a computationally reproducible paper?

Remember, to be computationally reproducible, the reader must be able to start from the data that were collected and arrive at the same results, including tables and figures.

- All data used in analysis in the paper
- All code to get from these data to the results shown in the paper
- Any environmental information (e.g., package versions) that are necessary for the code to generate the results

It's also helpful to include:

- Adequate documentation for the user to recreate the results from the data using the code and environmental information
- Guidance on how the code can reused (license)

Including data for the paper

Ideally, you will include all the data that were used to generate the paper's results.

However, sometimes there are challenges with including some or all of the data. These include:

- Data cannot be legally or ethically shared
- Data are very large

Sensitive or protected data

If your research includes human health data or other types of sensitive or protected data, you may not be able to publish that data.

As a result, you will not be able to make your paper fully computationally reproducible.

However, if you consider reproducibility as a spectrum, there are steps you can take to make the paper *more* computationally reproducible, even if you cannot share all the elements required to be fully reproducible.

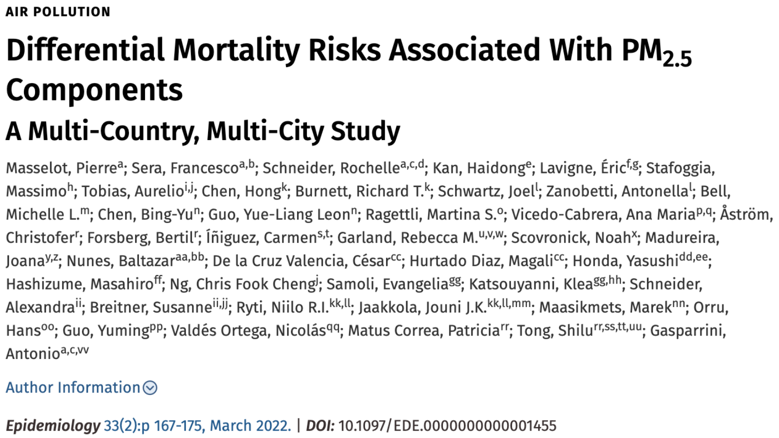
Sensitive or protected data

If you cannot include the data with other reproducbility materials, consider the following steps to make the paper as reproducible as possible:

- Include mock dataset (same format, different numbers) to use to test and demonstrate the function of the code
- Describe the data that you were not able to include, so that readers can understand how the code would work with the data
- Use .gitignore to include the data in a git repo on a local computer (or a secure server you're working on) but not push it to remote versions of that repo, like one on GitHub
- When working on a server for security reasons, consider having RStudio Server installed, as this provides a friendlier interface for working with git version control on the server (avoids having to do much from the terminal)

Sensitive or protected data

In the next lab, we'll explore a paper that has taken steps to be computationally reproducible, even those it cannot share its data.



AIR POLLUTION

Differential Mortality Risks Associated With $PM_{2.5}$ Components

A Multi-Country, Multi-City Study

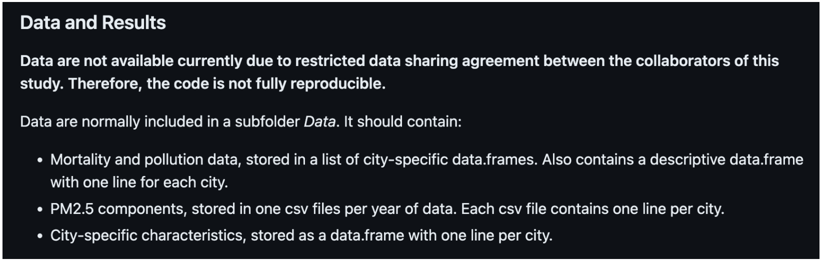
Masselot, Pierre[a]; Sera, Francesco[a,b]; Schneider, Rochelle[a,c,d]; Kan, Haidong[e]; Lavigne, Éric[f,g]; Stafoggia, Massimo[h]; Tobias, Aurelio[i,j]; Chen, Hong[k]; Burnett, Richard T.[k]; Schwartz, Joel[l]; Zanobetti, Antonella[l]; Bell, Michelle L.[m]; Chen, Bing-Yu[n]; Guo, Yue-Liang Leon[n]; Ragettli, Martina S.[o]; Vicedo-Cabrera, Ana Maria[p,q]; Åström, Christofer[r]; Forsberg, Bertil[r]; Íñiguez, Carmen[s,t]; Garland, Rebecca M.[u,v,w]; Scovronick, Noah[x]; Madureira, Joana[y,z]; Nunes, Baltazar[aa,bb]; De la Cruz Valencia, César[cc]; Hurtado Diaz, Magali[cc]; Honda, Yasushi[dd,ee]; Hashizume, Masahiro[ff]; Ng, Chris Fook Cheng[j]; Samoli, Evangelia[gg]; Katsouyanni, Klea[gg,hh]; Schneider, Alexandra[ii]; Breitner, Susanne[ii,jj]; Ryti, Niilo R.I.[kk,ll]; Jaakkola, Jouni J.K.[kk,ll,mm]; Maasikmets, Marek[nn]; Orru, Hans[oo]; Guo, Yuming[pp]; Valdés Ortega, Nicolás[qq]; Matus Correa, Patricia[rr]; Tong, Shilu[rr,ss,tt,uu]; Gasparrini, Antonio[a,c,vv]

Author Information

Epidemiology 33(2):p 167-175, March 2022. | *DOI:* 10.1097/EDE.0000000000001455

Sensitive or protected data

In the documentation of the GitHub repository that the authors use to share the code for the paper, they describe why they cannot includes the health data:



Data and Results

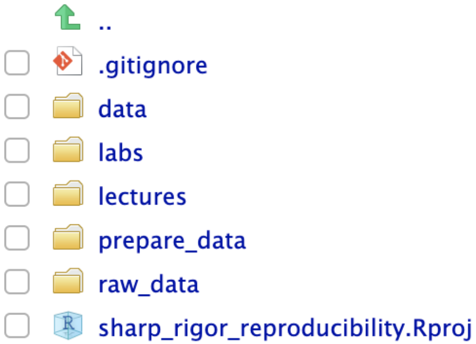
Data are not available currently due to restricted data sharing agreement between the collaborators of this study. Therefore, the code is not fully reproducible.

Data are normally included in a subfolder *Data*. It should contain:

- Mortality and pollution data, stored in a list of city-specific data.frames. Also contains a descriptive data.frame with one line for each city.
- PM2.5 components, stored in one csv files per year of data. Each csv file contains one line per city.
- City-specific characteristics, stored as a data.frame with one line per city.

.gitignore file

When you track a directory with `git` version control, you will have a file added to the directory called ".gitignore".



..
.gitignore
data
labs
lectures
prepare_data
raw_data
sharp_rigor_reproducibility.Rproj

.gitignore file

This is a plain text file. You can edit it to specify the files or groups of files that you want `git` to ignore as it tracks changes to that directory.

If you are sharing the `git` repo through GitHub, you can avoid posting certain files online by listing them in this ".gitignore" file.

.gitignore file

If you look through the repository for the sample paper for the lab, you'll see that they listed all data in the ".gitignore" file. This allows them to have the data on their local computer but not share it publicly with the rest of the repo.

```
# Session Data files
.RData
**/*.RData

# RStudio files
.Rproj.user/

# Other
Data/
```

Extremely large data

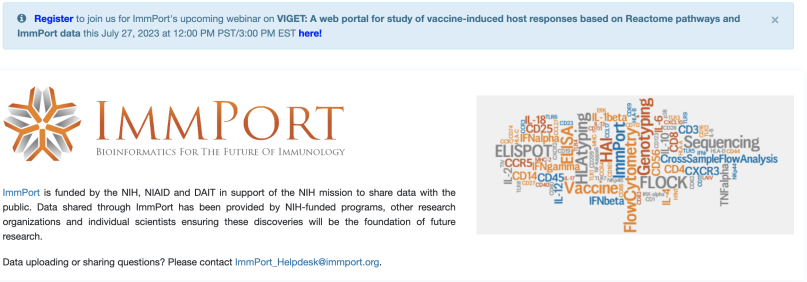
Sometimes, the paper will include results based on datasets that are very large.

These data sizes can exceed the limits for some platforms that you otherwise might use to share the reproducibility materials (e.g., GitHub, the journal's website, the author's university website).

Repositories built for research data curation might be better suited for storing very large datasets.

Research data repositories

One example is the ImmPort data repository, which is sponsored by US agencies including the National Institute of Allergy and Infectious Diseases.



Register to join us for ImmPort's upcoming webinar on **VIGET: A web portal for study of vaccine-induced host responses based on Reactome pathways and ImmPort data** this July 27, 2023 at 12:00 PM PST/3:00 PM EST **here!**

IMMPORT
BIOINFORMATICS FOR THE FUTURE OF IMMUNOLOGY

ImmPort is funded by the NIH, NIAID and DAIT in support of the NIH mission to share data with the public. Data shared through ImmPort has been provided by NIH-funded programs, other research organizations and individual scientists ensuring these discoveries will be the foundation of future research.

Data uploading or sharing questions? Please contact ImmPort_Helpdesk@immport.org.

Extremely large data

Some things to keep in mind about these repositories include:

- Some repositories might have APIs—data can be downloaded programatically (from the code script)
- Some repositories allow options to toggle private versus public
- Some repositories have suggested or required data models
- Some repositories include platforms to work interactively with the data

dkNET has a curated list of repositories for research data at https://dknet.org/rin/suggested-data-repositories

Data to include

It can be useful to save intermediate versions of the data, as the code moves from raw data to the final figures, tables, and other results. Intermediate data can include:

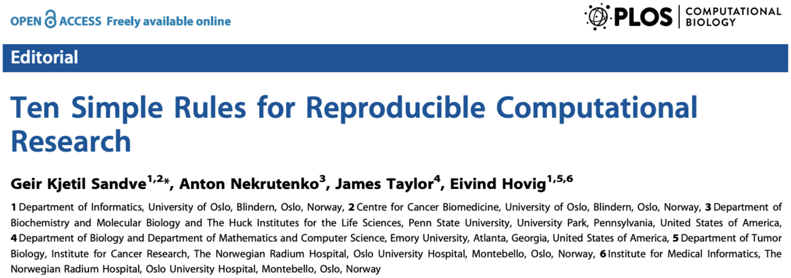
- Processed data (e.g., if raw data are from flow cytometry, the results after gating and counting the data)
- Data points that are plotted in figures (i.e., data at the point immediately before it is graphed)

Note that this is in addition to the raw data.

For these intermediate data sets, it's helpful if you save them in plain text formats (e.g., ".csv" or ".txt" file).

Data to include

There's a nice discussion of why to include all intermediate results in an article available through PLoS Computational Biology.


OPEN ACCESS Freely available online

PLOS | COMPUTATIONAL BIOLOGY

Editorial

Ten Simple Rules for Reproducible Computational Research

Geir Kjetil Sandve[1,2]*, Anton Nekrutenko[3], James Taylor[4], Eivind Hovig[1,5,6]

1 Department of Informatics, University of Oslo, Blindern, Oslo, Norway, **2** Centre for Cancer Biomedicine, University of Oslo, Blindern, Oslo, Norway, **3** Department of Biochemistry and Molecular Biology and The Huck Institutes for the Life Sciences, Penn State University, University Park, Pennsylvania, United States of America, **4** Department of Biology and Department of Mathematics and Computer Science, Emory University, Atlanta, Georgia, United States of America, **5** Department of Tumor Biology, Institute for Cancer Research, The Norwegian Radium Hospital, Oslo University Hospital, Montebello, Oslo, Norway, **6** Institute for Medical Informatics, The Norwegian Radium Hospital, Oslo University Hospital, Montebello, Oslo, Norway

Sharing code and environment information

Usually, you will not have any problems with the code and environmental information being sensitive or extremely large, as with the research data.

However, there are some other things you should consider as you aim to share code to improve the computational reproducbility of a paper.

- The code will not work correctly if you use absolute rather than relative pathnames for files.
- The code might not work correctly if it is run on software with a different version.
- The code will not, unless you take appropriate steps, recreate the same results if some of the steps in the code use random number generation.

Absolute versus relative pathnames

When you want to tell R about a file, you can either give it the file's full address (from the root directory of your computer) or you can give it directions from the working directory. The first is an **absolute** filepath and the second is a **relative** filepath.

```
read_csv("/Users/ganders/my_proj/data/my_data.csv")
read_csv("data/my_data.csv")
```

Which will be robust when someone else tries to run the code on their own computer?

Versions of software

When you run R code, your computer not only runs the code that you wrote, but also code within your installation of base R as well as code from packages that you have installed.

You code essentially is built on top of these other pieces of code.

At an even deeper level, your computer is also using code from its operating system to execute your R code script.

Versions of software

All of these pieces of software are evolving. When you install a new version of any of these pieces of software, it will change the environment in which your R code runs.

Similarly, if someone else tries to run your code, it may fail if they have a different version of one or more of these pieces of software.

Versions of software

There is a gradient in how quickly different pieces of software evolve (in terms of updates that affect the execution of R code). Code for your operating system and base R tend to evolve more slowly (in terms of things that affect R code), while package extensions to R tend to change more quickly.

Some R package projects are in stages of significant evolution. Two types of R packages which I've repeatedly seen introduce reproducibility issues based on the package version are the tidyverse suite of packages and the Bioconductor collection of packages.

Versions used in original analysis

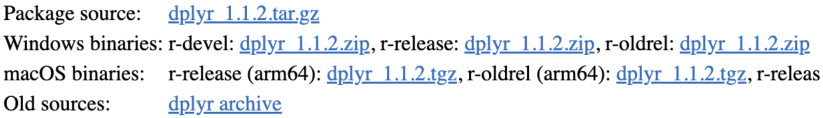
There are a few approaches you can take to address this issue. Your aim is to provide the user with enough information that they can recreate the environment you had when you applied the code to the data to generate the paper's results.

- Provide an in-depth description of the versions of software used when running the paper's analysis
- Help the user recreate the original environment
- Save and share the full environment

Providing session information when running code

The simplest was to provide environment information is to share the versions of all software that was used when the paper's analysis was conducted.

Most open-source software will archive older versions. A user can obtain an older version of the software from these archives and install it on R. Here's an example of how to find old sources on CRAN:



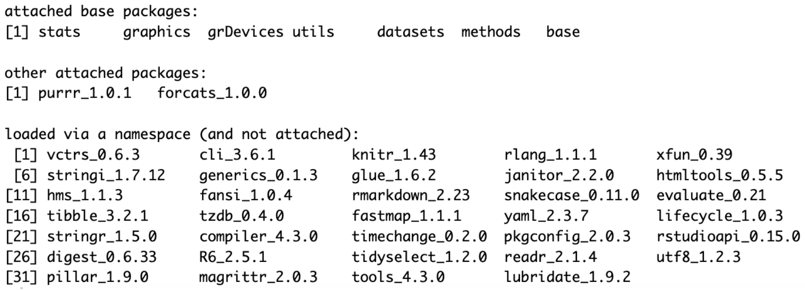
Package source:	dplyr_1.1.2.tar.gz
Windows binaries:	r-devel: dplyr_1.1.2.zip, r-release: dplyr_1.1.2.zip, r-oldrel: dplyr_1.1.2.zip
macOS binaries:	r-release (arm64): dplyr_1.1.2.tgz, r-oldrel (arm64): dplyr_1.1.2.tgz, r-releas
Old sources:	dplyr archive

This approach may not work for proprietary software (e.g., SAS, MATLAB), if they do not make prior software versions available.

Providing session information when running code

R has some functions that will print information about the software in its working environment, including the names and versions of all loaded packages.

These functions include `sessionInfo` from base R and `session_info` from `devtools`.



```
attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] purrr_1.0.1   forcats_1.0.0

loaded via a namespace (and not attached):
 [1] vctrs_0.6.3       cli_3.6.1         knitr_1.43        rlang_1.1.1       xfun_0.39
 [6] stringi_1.7.12    generics_0.1.3    glue_1.6.2        janitor_2.2.0     htmltools_0.5.5
[11] hms_1.1.3         fansi_1.0.4       rmarkdown_2.23    snakecase_0.11.0  evaluate_0.21
[16] tibble_3.2.1      tzdb_0.4.0        fastmap_1.1.1     yaml_2.3.7        lifecycle_1.0.3
[21] stringr_1.5.0     compiler_4.3.0    timechange_0.2.0  pkgconfig_2.0.3   rstudioapi_0.15.0
[26] digest_0.6.33     R6_2.5.1          tidyselect_1.2.0  readr_2.1.4       utf8_1.2.3
[31] pillar_1.9.0      magrittr_2.0.3    tools_4.3.0       lubridate_1.9.2
```

Providing session information from running code

You can include a call to `sessionInfo` or `session_info` at the end of an RMarkdown document.

This will print out all the information for the environment in which the RMarkdown file was rendered, including:

- The version of base R
- The platform (this provides information about the operating system and its version)
- Versions of other packages of routines (e.g., LAPACK for FORTRAN)
- Attached and loaded packages, as well as the version of each

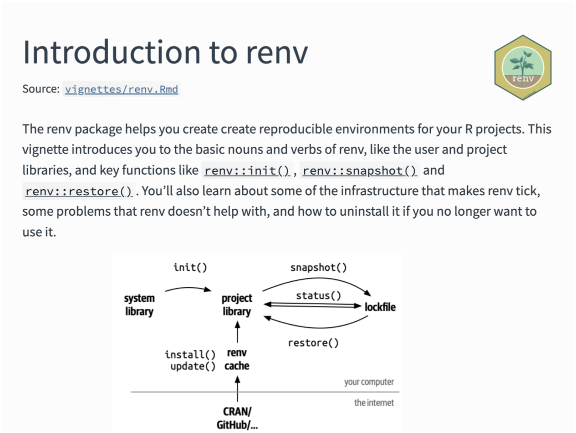
Recreating the original environment

When you record the session information, it still leaves a lot of work on a later user to find and install all the old versions of packages.

To alleviate this, there are some R packages that automate part of the process. These help the user recreate the environment that code ran in originally. In particular, they can help when a user wants to recreate the environments for several different projects on one computer, with different software versions for each project.

Recreating the original environment

The `renv` package is one of these packages that helps automating the process of recreating an environment. It has extensive help documentation online:



Introduction to renv

Source: `vignettes/renv.Rmd`

The renv package helps you create create reproducible environments for your R projects. This vignette introduces you to the basic nouns and verbs of renv, like the user and project libraries, and key functions like `renv::init()`, `renv::snapshot()` and `renv::restore()`. You'll also learn about some of the infrastructure that makes renv tick, some problems that renv doesn't help with, and how to uninstall it if you no longer want to use it.

Saving and sharing the environment

A number of researchers take the approach of saving and sharing the environment in which the code originally ran. One approach is to use a container (e.g., a Docker container).

Container

A container includes all files and directory organization. Also includes the software needed to run code. It does not include all the code for the operating system, but it does provide information to help recreate that part of the environment.

This set-up allows the user to rerun the analysis *in its original environment* by running it within the container.

Container

Downsides:

- File size of container is often large
- Higher threshold at present for both creating and using containers with R code versus scripts paired with `renv`
- Can be somewhat of a black box, in terms of dissecting what's in the container
- Can be hard to adapt parts for new tasks (often relies on code that might be out of date)

Future-proof your code

You can also take some steps to try to avoid or delay problems that are created as the packages that the code depends on evolves.

One is to try to avoid using superseded or deprecated functions in your code.

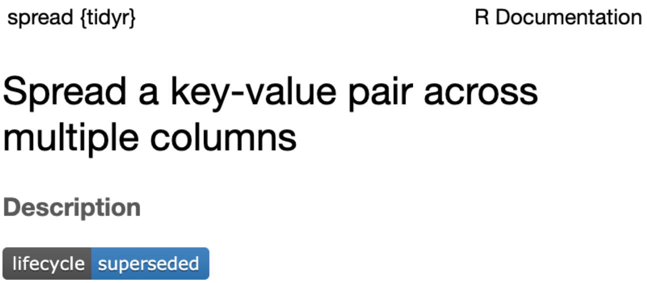
These are functions where an improved version exists, with plans that the new version will become the default.

A function that is **superseded** will be maintained, but won't be a key part of the evolving package (in particular, it won't receive any new updates, only enough changes to still work).

A function that is **deprecated** will eventually be removed from the package, and a replacement is already available and preferred.

Future-proof your code

You can find often find information about deprecation in the helpfiles for a file.

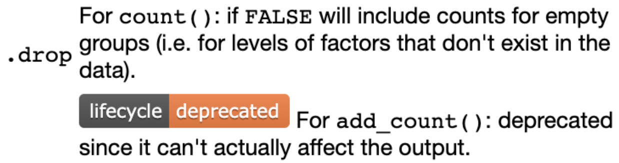


spread {tidyr} R Documentation

Spread a key-value pair across multiple columns

Description

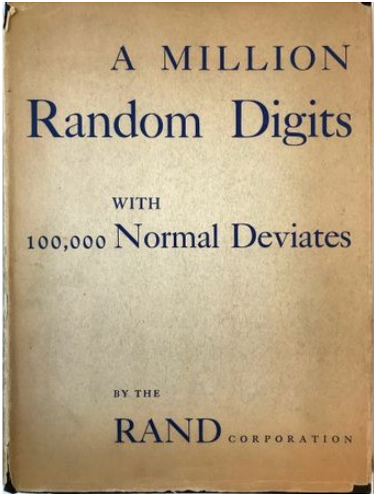
lifecycle superseded



`.drop`	For `count()`: if `FALSE` will include counts for empty groups (i.e. for levels of factors that don't exist in the data). lifecycle deprecated For `add_count()`: deprecated since it can't actually affect the output.

Reproducing randomness

Many statistical methods require randomization, implemented through the generation of random numbers.


A MILLION
Random Digits
WITH
100,000 Normal Deviates
BY THE
RAND CORPORATION

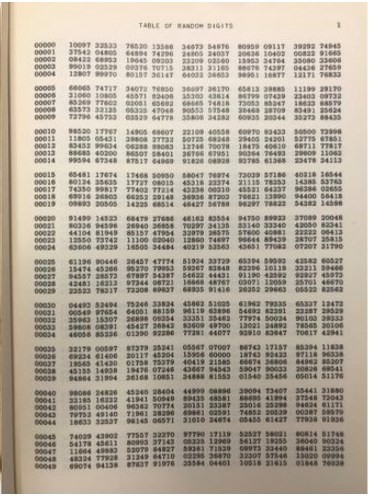


TABLE OF RANDOM DIGITS 1

00000	10097	32533	76520	13586	34673	54876	80959	09117	39292	74945
00001	37542	04805	64894	74296	24805	24037	20636	10402	00822	91665
00002	08422	68953	19645	09303	23209	02560	15953	34764	35080	33606
00003	99019	02529	09376	70715	38311	31165	88676	74397	04436	27659
00004	12807	99970	80157	36147	64032	36653	98951	16877	12171	76833
00005	66065	74717	34072	76850	36697	36170	65813	39885	11199	29170
00006	31060	10805	45571	82406	35303	42614	86799	07439	23403	09732
00007	85269	77602	02051	65692	68665	74818	73053	85247	18623	88579
00008	63573	32135	05325	47048	90553	57548	28468	28709	83491	25624
00009	73796	45753	03529	64778	35808	34282	60935	20344	35273	88435
00010	98520	17767	14905	68607	22109	40558	60970	93433	50500	73998
00011	11805	05431	39808	27732	50725	68248	29405	24201	52775	67851
00012	83452	99634	06288	98083	13746	70078	18475	40610	68711	77817
00013	88685	40200	86507	58401	36766	67951	90364	76493	29609	11062
00014	99594	67348	87517	64969	91826	08928	93785	61368	23478	34113
00015	65481	17674	17468	50950	58047	76974	73039	57186	40218	16544
00016	80124	35635	17727	08015	45318	22374	21115	78253	14385	53763
00017	74350	99817	77402	77214	43236	00210	45521	64237	96286	02655
00018	69916	26803	66252	29148	36936	87203	76621	13990	94400	56418
00019	09893	20505	14225	68514	46427	56788	96297	78822	54382	14598
00020	91499	14523	68479	27686	46162	83554	94750	89923	37089	20048
00021	80336	94598	26940	36858	70297	34135	53140	33340	42050	82341
00022	44104	81949	85157	47954	32979	26575	57600	40881	22222	06413
00023	12550	73742	11100	02040	12860	74697	96644	89439	28707	25815
00024	63606	49329	16505	34484	40219	52563	43651	77082	07207	31790
00025	61196	90446	26457	47774	51924	33729	65394	59593	42582	60527
00026	15474	45266	95270	79953	59367	83848	82396	10118	33211	59466
00027	94557	28573	67897	54387	54622	44431	91190	42592	92927	45973
00028	42481	16213	97344	08721	16868	48767	03071	12059	25701	46670
00029	23523	78317	73208	89837	68935	91416	26252	29663	05522	82562
00030	04493	52494	75246	33824	45862	51025	61962	79335	65337	12472
00031	00549	97654	64051	88159	96119	63896	54692	82391	23287	29529
00032	35963	15307	26898	09354	33351	35462	77974	50024	90103	39333
00033	59808	08391	45427	26842	83609	49700	13021	24892	78565	20106
00034	46058	85236	01390	92286	77281	44077	93910	83647	70617	42941
00035	32179	00597	87379	25241	05567	07007	86743	17157	85394	11838
00036	69234	61406	20117	45204	15956	60000	18743	92423	97118	96338
00037	19565	41430	01758	75379	40419	21585	66674	36806	84962	85207
00038	45155	14938	19476	07246	43667	94543	59047	90033	20826	69541
00039	94864	31994	36168	10851	34888	81553	01540	35456	05014	51176
00040	98086	24826	45240	28404	44999	08896	39094	73407	35441	31880
00041	33185	16232	41941	50949	89435	48581	88695	41994	37548	73043
00042	80951	00406	96382	70774	20151	23387	25016	25298	94624	61171
00043	79752	49140	71961	28296	69861	02591	74852	20539	00387	59579
00044	18633	32537	98145	06571	31010	24674	05455	61427	77938	91936
00045	74029	43902	77557	32270	97790	17119	52527	58021	80814	51748
00046	54178	45611	80993	37143	05335	12969	56127	19255	36040	90324
00047	11664	49883	52079	84827	59381	71539	09973	33440	88461	23356
00048	48324	77928	31249	64710	02295	36870	32307	57546	15020	09994
00049	69074	94138	87637	91976	35584	04401	10518	21615	01848	76938

Reproducing randomness

Your code might be using random number generation if you:

- are sampling
- are using Monte Carlo methods / simulations
- are doing Bayesian statistics

Reproducing randomness

There are ways to generate truly random numbers:

Reproducing randomness

However, computers typically use a pseudorandom number generator.

For R's random number generators, you can force the same random numbers each time you run a piece of code by setting something called a "seed" before any function calls that involve random number generation.

Reproducing randomness

If you don't set a seed, you will get different results when you run code that involves random number generation, because of the randomness involved.

```
sample(1:5)
```

```
## [1] 5 2 1 3 4
```

```
sample(1:5)
```

```
## [1] 2 1 5 4 3
```

Reproducing randomness

If you set the same seed each time before you run that code, on the other hand, you will get the same "random" results each time you generate random numbers:

```r
set.seed(100)
sample(1:5)
```

```
## [1] 2 3 5 4 1
```

```r
set.seed(100)
sample(1:5)
```

```
## [1] 2 3 5 4 1
```