Copy Number Analysis using CNVpytor

February, 2022

Table of Contents

# About this Course

Copy number analysis is the process of detecting copy number information from a genome. This course will guide one to get copy number information from alignment file using CNVpytor.

# 1 Introduction

CNVpytor is a Python package and command line tool for copy number variation (CNV)/Copy Number alteration (CNA) analysis from depth-of-coverage by mapped reads developed in Abyzov Lab, Mayo Clinic.

## 1.1 Motivation

## 1.2 Target Audience

The course is intended for …

## 1.3 Curriculum

The course covers…

# 2 Installation and Setting Reference genome

## 2.1 Learning Objectives

This chapter will cover:

* Installation
  + using setuptools
  + using pip
* Steps for setting reference genome
  + Create GC and mask file for new reference genome

## 2.2 Libraries

CNVpytor is written in python and it works on both python 2 and 3. Please install python before proceeding with the installation steps.

The following code can be used to check python version

python --version

## 2.3 Installation

### 2.3.1 Install by cloning from GitHub

The following lines of codes can be used to install directly from GitHub

> git clone https://github.com/abyzovlab/CNVpytor.git  
> cd CNVpytor  
> pip install .

For single user (without admin privileges), can use:

> pip install --user .

### 2.3.2 Install using pip

The following code will download the latest code from GitHub and use pip to install.

pip install git+https://github.com/abyzovlab/CNVpytor.git

## 2.4 Steps for setting reference genome

Commonly used human reference genomes (HG19 and HG38) are integrated and comes with default GitHub installation. Its detects the genome by comparing the chromosome lengths. Although, other reference genomes are also frequently used in practice. This section will guide one to add a new reference genome.

Now, we will create example configuration file for mouse reference genome MGSCv37 which contains a list of chromosomes and chromosome lengths:

# Filename: example\_ref\_genome\_conf.py  
  
import\_reference\_genomes = {  
 "mm9": {  
 "name": "MGSCv37",  
 "species": "Mus musculus",  
 "chromosomes": OrderedDict(  
 [("chr1", (197195432, "A")), ("chr2", (181748087, "A")), ("chr3", (159599783, "A")),  
 ("chr4", (155630120, "A")), ("chr5", (152537259, "A")), ("chr6", (149517037, "A")),  
 ("chr7", (152524553, "A")), ("chr8", (131738871, "A")), ("chr9", (124076172, "A")),  
 ("chr10", (129993255, "A")), ("chr11", (121843856, "A")), ("chr12", (121257530, "A")),  
 ("chr13", (120284312, "A")), ("chr14", (125194864, "A")), ("chr15", (103494974, "A")),  
 ("chr16", (98319150, "A")), ("chr17", (95272651, "A")), ("chr18", (90772031, "A")),  
 ("chr19", (61342430, "A")), ("chrX", (166650296, "S")), ("chrY", (15902555, "S")),  
 ("chrM", (16299, "M"))]),  
 "gc\_file": "/..PATH../MGSCv37\_gc\_file.pytor",  
 "mask\_file": "/..PATH../MGSCv37\_mask\_file.pytor"  
 }  
}

Letter next to chromosome length denote type of a chromosome: A - autosome, S - sex chromosome, M - mitochondria. The instruction for creating MGSCv37\_gc\_file.pytor and MGSCv37\_mask\_file.pytor is in the next section.

To use CNVpytor with new reference genome us -conf option in each cnvpytor command, e.g.

cnvpytor -conf REL\_PATH/example\_ref\_genome\_conf.py -root file.pytor -rd file.bam

CNVpytor will use chromosome lengths from alignment file to detect reference genome. However, if you configured reference genome after you had already run -rd step you could assign reference genome using -rg:

cnvpytor -conf REL\_PATH/example\_ref\_genome\_conf.py -root file.pytor -rg mm9

To avoid typing -conf REL\_PATH/example\_ref\_genome\_conf.py each time you run cnvpytor, you can create an bash alias or make configuration permanent by copying example\_ref\_genome\_conf.py to ~/.cnvpytor/reference\_genomes\_conf.py.

### 2.4.1 Create GC and mask file for new reference genome

CNVpytor also has optional features for GC correction and masking (i.e., commonly known false positive regions). One can setup their reference genome by adding its related content in the gc\_file and mask\_file field of the configuration file.

To create GC file, we need sequence of the reference genome in fasta.gz file:

> cnvpytor -root MGSCv37\_gc\_file.pytor -gc ~/hg19/mouse.fasta.gz -make\_gc\_file

This command will produce MGSCv37\_gc\_file.pytor file that contains information about GC content in 100-base-pair bins. For reference genomes where we have strict mask in the same format as [1000 Genomes Project strict mask](http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000_genomes_project/working/20160622_genome_mask_GRCh38/), we can create mask file using command:

> cnvpytor -root MGSCv37\_mask\_file.pytor -mask ~/hg19/mouse.strict\_mask.whole\_genome.fasta.gz -make\_mask\_file

If you do not have mask file, You can skip this step. Mask file contains information about regions of the genome that are more accessible to next generation sequencing methods using short reads. CNVpytor uses P marked positions to filter SNP-s and read depth signal. If reference genome configuration does not contain mask file, CNVpytor will still be fully functional, apart from the filtering step. You may also generate your own mask file by creating fasta file that contains character “P” if corresponding base pair passes the filter and any character different than “P” if not.

# 3 Calling CNV from Alignment file using terminal

This section will guide to use CNVpytor for calling CNV using Read depth file and incorporating variant information

## 3.1 Learning Objectives

This chapter will cover:

* Steps to process alignment file
* Steps to process variant information.

## 3.2 Calling CNV from Alignment file

The following steps can be used to process read depth information from alignment file

> cnvpytor -root file.pytor -rd file.bam

If the reference genome is human than there is no need to set the reference genome and one can run the following steps.

> cnvpytor -root file.pytor -his 1000 10000 100000  
> cnvpytor -root file.pytor -partition 1000 10000 100000  
> cnvpytor -root file.pytor -call 1000 10000 100000

For non human reference genome, please have a look at the section for setting reference genome.

## 3.3 Importing and using variant information:

> cnvpytor -root file.pytor -snp file.vcf -sample sample\_name  
> cnvpytor -root file.pytor -pileup file.bam # OPTIONAL  
> cnvpytor -root file.pytor -mask\_snps # OPTIONAL   
> cnvpytor -root file.pytor -baf 10000 100000

# 4 Data import

## 4.1 Learning Objectives

* Import read depth signal
* Import variant information

# 5 Import read depth signal

Make sure that you have indexed alignment (SAM, BAM or CRAM) file. Initialize your CNVpytor project by running:

> cnvpytor -root file.pytor -rd file.bam [-chrom name1 ...] [-T ref.fa.gz]

where:

\* file.pytor -- specifies output CNVpytor file (HDF5 file)  
\* name1 ... -- specifies chromosome name(s).  
\* file.bam -- specifies bam/sam/cram file name.  
\* -T ref.fa.gz -- specifies reference genome file (only for cram file without reference genome).

Chromosome names must be specified the same way as they are described in the sam/bam/cram header, e.g., chr1 or 1. One can specify multiple chromosomes separated by space. If no chromosome is specified, read mapping is extracted for all chromosomes in the sam/bam file. Note that the pytor file is not being overwritten.

**Examples:**

> cnvpytor -root NA12878.pytor -chrom 1 2 3 -rd NA12878\_ali.bam

for bam files with a header like this:

@HD VN:1.4 GO:none SO:coordinate  
@SQ SN:1 LN:249250621  
@SQ SN:2 LN:243199373  
@SQ SN:3 LN:198022430

or

> cnvpytor -root NA12878.pytor -chrom chr1 chr2 chr3 -rd NA12878\_ali.bam

for bam files with a header like this:

@HD VN:1.4 GO:none SO:coordinate  
@SQ SN:chr1 LN:249250621  
@SQ SN:chr2 LN:243199373  
@SQ SN:chr3 LN:198022430

After -rd step file file.pytor is created and read depth data binned to 100 base pair bins will be stored in *pytor* file.

Chromosome names and lengths are parsed from the input file header and used to detect reference genome.

## 5.1 check reference genome

To check is reference genome detected use:

> cnvpytor -root file.pytor -ls

CNVpytor will print out details about file.pytor including line that specify which reference genome is used and are there available GC and mask data:

Using reference genome: hg19 [ GC: yes, mask: yes ]

Command -ls is useful if you want to check content of *pytor* file but also date and version of CNVpytor that created it.

## 5.2 Predicting CNV regions

First we have to chose bin size. By CNVpytor design it have to be divisible by 100. Here we will use 10 kbp and 100 kbp bins.

To calculate read depth histograms, GC correction and statistics type:

> cnvpytor -root file.pytor -his 10000 100000

Next step is partitioning using mean-shift method:

> cnvpytor -root file.pytor -partition 10000 100000

Finally we can call CNV regions using commands:

> cnvpytor -root file.pytor -call 10000 > calls.10000.tsv  
> cnvpytor -root file.pytor -call 100000 > calls.100000.tsv

Result is stored in tab separated files with following columns:

\* CNV type: "deletion" or "duplication",  
\* CNV region (chr:start-end),  
\* CNV size,  
\* CNV level - read depth normalized to 1,  
\* e-val1 -- e-value (p-value multiplied by genome size divided by bin size) calculated using t-test statistics between RD statistics in the region and global,  
\* e-val2 -- e-value (p-value multiplied by genome size divided by bin size) from the probability of RD values within the region to be in the tails of a gaussian distribution of binned RD,  
\* e-val3 -- same as e-val1 but for the middle of CNV,  
\* e-val4 -- same as e-val2 but for the middle of CNV,  
\* q0 -- fraction of reads mapped with q0 quality in call region,  
\* pN -- fraction of reference genome gaps (Ns) in call region,  
\* dG -- distance from closest large (>100bp) gap in reference genome.

Using viewer mode we can filter calls based on five parameters:

CNV size, e-val1, q0, pN and dG:

> cnvpytor -root file.pytor [file2.pytor ...] -view 10000  
  
cnvpytor> set Q0\_range -1 0.5 # filter calls with more than half not uniquely mapped reads  
cnvpytor> set p\_range 0 0.0001 # filter non-confident calls   
cnvpytor> set p\_N 0 0.5 # filter calls with more than 50% Ns in reference genome   
cnvpytor> set size\_range 50000 inf # filter calls smaller than 50kbp   
cnvpytor> set dG\_range 100000 inf # filter calls close to gaps in reference genome (<100kbp)  
cnvpytor> print calls # printing calls on screen (tsv format)  
...  
...  
cnvpytor> set print\_filename file.xlsx # output filename (xlsx, tsv or vcf)  
cnvpytor> set annotate # turn on annotation (optional - takes a lot of time)  
cnvpytor> print calls # generate output file with filtered calls   
cnvpytor> quit

Upper bound for parameters size\_range and dG\_range can be *inf* (infinity).

If there are multiple samples (pytor files) there will be an additional column with sample name in tsv format, multiple sheets in Excel format, and multiple sample columns in vcf format.

## 5.3 Import SNP data

### 5.3.1 From variant file

To import variant data from VCF file use following command:

> cnvpytor -root file.pytor -snp file.vcf.gz [-sample sample\_name] [-chrom name1 ...] [-ad AD\_TAG] [-gt GT\_TAG] [-noAD]

where:

\* file.pytor -- specifies cnvpytor file,  
\* file.vcf -- specifies variant file name.  
\* sample\_name -- specifies VCF sample name,  
\* name1 ... -- specifies chromosome name(s),  
\* -ad AD\_TAG -- specifies AD tag used in vcf file (default AD)  
\* -gt GT\_TAG -- specifies GT tag used in vcf file (default GT)  
\* -noAD -- ref and alt read counts will not be readed (see next section)

Chromosome names must be specified the same way as they are described in the vcf header, e.g., chr1 or 1. One can specify multiple chromosomes separated by space. If no chromosome is specified, all chromosomes from the vcf file will be parsed.

If chromosome names in variant and alignment file are different in prefix chr (e.g. in “1” and “chr1”) CNVpytor will detect it and match the names using first imported name for both signals.

### 5.3.2 Using SNP positions from variant file and counts from alignment file

In some cases it is useful to read positions of SNPs from vcf file and extract read counts from bam file. For example if we have two samples, normal tissue and cancer, normal can be used to call germline SNPs, while samtools mpileup procedure can be used to calculate read counts in cancer sample at the positions of SNPs. CNVpytor have implemented this procedure. After reading SNP positions (previous step) type:

> cnvpytor -root file.pytor -pileup file.bam [-T ref.fa.gz]

where

\* file.pytor -- specifies cnvpytor file,  
\* file.bam -- specifies bam/sam/cram file,  
\* -T ref.fa.gz -- specifies reference genome file (only for cram file without reference genome).

### 5.3.3 Calculating BAF histograms

To apply 1000 genomes strict mask filter:

> cnvpytor -root file.pytor -mask\_snps

To calculate baf histograms for maf, baf and likelihood function for baf use:

> cnvpytor -root file.pytor -baf 10000 100000 [-nomask]

### 5.3.4 Predicting CNV regions using joint caller (prototype)

Finally we can call CNV regions using commands:

> cnvpytor -root file.pytor -call combined 10000 > calls.combined.10000.tsv  
> cnvpytor -root file.pytor -call combined 100000 > calls.combined.100000.tsv

Result is stored in tab separated files with following columns:

\* CNV type: "deletion", "duplication", or ”cnnloh",   
\* CNV region (chr:start-end),  
\* CNV size,  
\* CNV level - read depth normalized to 1,  
\* e-val1 -- e-value (p-value multiplied by genome size divided by bin size) calculated using t-test statistics between RD statistics in the region and global,  
\* e-val2 -- e-value (p-value multiplied by genome size divided by bin size) from the probability of RD values within the region to be in the tails of a gaussian distribution of binned RD,  
\* e-val3 -- same as e-val1 but for the middle of CNV,  
\* e-val4 -- same as e-val2 but for the middle of CNV,  
\* q0 -- fraction of reads mapped with q0 quality in call segments,  
\* pN -- fraction of reference genome gaps (Ns) within call region,  
\* dNS -- fraction of reference genome gaps (Ns) within call segments,  
\* pP -- fraction of P bases (1kGP strict mask) within call segments,  
\* bin\_size – size of bins  
\* n – number of bins within call segments,  
\* delta\_BAF – change in BAF from ½,  
\* e-val1 -- e-value RD based (repeted, reserved for future upgrades),  
\* baf\_eval – e-value BAF based,  
\* hets – number of HETs,  
\* homs – number of HOMs,  
\* cn\_1 – most likely model copy number,  
\* genotype\_1 - most likely model genotype,  
\* likelihood\_1 – most likely model likelihood,  
\* cf\_1 -- most likely model cell fraction,  
\* cn\_2 – the second most likely model copy number,  
\* genotype\_2 - the second most likely model genotype,  
\* likelihood\_2 – the second most likely model likelihood,  
\* cf\_2 -- the second most likely model cell fraction.

Using viewer mode we can filter calls based on five parameters:

CNV size, e-val1, q0, pN and dG:

> cnvpytor -root file.pytor [file2.pytor ...] -view 10000  
cnvpytor> set caller combined\_mosaic # IMPORTANT, default caller is mean shift  
cnvpytor> set Q0\_range -1 0.5 # filter calls with more than half not uniquely mapped reads  
cnvpytor> set p\_range 0 0.0001 # filter non-confident calls   
cnvpytor> set p\_N 0 0.5 # filter calls with more than 50% Ns in reference genome   
cnvpytor> set size\_range 50000 inf # filter calls smaller than 50kbp   
cnvpytor> set dG\_range 100000 inf # filter calls close to gaps in reference genome (<100kbp)  
cnvpytor> print calls # printing calls on screen (tsv format)  
...  
...  
cnvpytor> set print\_filename file.xlsx # output filename (xlsx, tsv or vcf)  
cnvpytor> set annotate # turn on annotation (optional - takes a lot of time)  
cnvpytor> print calls # generate output file with filtered calls   
cnvpytor> quit

Upper bound for parameters size\_range and dG\_range can be *inf* (infinity).

If there are multiple samples (pytor files) there will be an additional column with sample name in tsv format, multiple sheets in Excel format, and multiple sample columns in vcf format.

Comparison between CNVnator and CNVpytor callers output format:

## 5.4 Genotyping genomic regions

Using -genotype option followed by bin\_sizes you can enter region and genotype calculation for each bin size will be performed:

> cnvpytor -root file.pytor -genotype 10000 100000  
12:11396601-11436500  
12:11396601-11436500 1.933261 1.937531  
22:20999401-21300400  
22:20999401-21300400 1.949186 1.957068

Genotyping with additional information:

> cnvpytor -root file.pytor -genotype 10000 -a [-rd\_use\_mask] [-nomask]  
12:11396601-11436500  
12:11396601-11436500 2.0152 1.629621e+04 9.670589e+08 0.0000 0.0000 4156900 1.0000 50 4 0.0000 1.000000e+00

Output columns are:

1. region,  
1. cnv level -- mean RD normalized to mean autosomal RD level,  
1. e\_val\_1 -- p value calculated using t-test statistics between RD statistics in the region and global,  
1. e\_val\_2 -- p value from the probability of RD values within the region to be in the tails of a gaussian distribution of binned RD,  
1. q0 – fraction of reads mapped with q0 quality within call region,  
1. pN – fraction of reference genome gaps (Ns) within call region,  
1. dG -- distance from closest large (>100bp) gap in reference genome,  
1. proportion of bins used in RD calculation (with option \_-rd\_use\_mask\_ some bins can be filtered out),  
1. Number of homozygous variants within region,  
1. Number of heterozygous variants,  
1. BAF level (difference from 0.5) for HETs estimated using maximum likelihood method,  
1. p-value based on BAF signal.  
  
  
Option \_-rd\_use\_mask\_ turns on P filtering (1000 Genome Project strict mask) for RD signal.  
  
Option \_-nomak\_ turns off P filtering of SNPs (1000 Genome Project strict mask) for BAF signal.

**Example:**

Genotype all called CNVs:

> awk '{ print $2 }' calls.10000.tsv | cnvpytor -root file.pytor -genotype 10000 100000

# 6 Visualization

## 6.1 Learning Objectives

This chapter will cover:

* Visualization in command line mode
* Interactive visualization

### 6.1.1 Plot from command line

Chromosome wide plots:

> cnvpytor -root file.pytor -plot [rd BIN\_SIZE] [likelihood BIN\_SIZE] [baf BIN\_SIZE] [snp] [-o IMAGE\_FILENAME]

where \* rd BIN\_SIZE – plots RD signal for all chromosomes \* likelihood BIN\_SIZE – plots baf likelihood for all chromosomes \* baf BIN\_SIZE – plots baf/maf/likelihood peak position for all chromosomes \* snp – plots baf for each snp for all chromosomes \* -o IMAGE\_FILENAME – if specified, saves plot in file instead to show on the screen

Manhattan plot:

> cnvpytor -root file.pytor -plot manhattan BIN\_SIZE [-chrom name1 ...] [-o IMAGE\_FILENAME]

Circular plot:

> cnvpytor -root file.pytor -plot circular BIN\_SIZE [-chrom name1 ...] [-o IMAGE\_FILENAME]

Plot genomic regions:

> cnvpytor -root file.pytor -plot regions [reg1[,| ]...] BIN\_SIZE [-panels [rd] [likelihood] [baf] [snp] ...] [-o IMAGE\_FILENAME]

where \* reg1 – comma or space separated regions in form CHR[:START-STOP], e.g. 1:1M-20M 2 3:200k-80000010 \* if regions are comma separated they will be plotted in the same subplot \* space will split regions in different subplots \* -panels – specify which panels to plot: rd likelihood baf snp \* -o IMAGE\_FILENAME – if specified, saves plot in file instead to show on the screen

### 6.1.2 Plot from interactive mode

The best way to visualize CNVpytor results is interactive mode. Enter interactive mode by typing:

cnvpytor -root file.pytor -view BIN\_SIZE

There is tab completion and help similar to man pages. Type double tab or help to start.

# 7 Visualize CNVpytor data inside JBrowse

## 7.1 Learning Objectives

This chapter will cover:

* How to visualize CNVpytor data in JBrowse.

## 7.2 Libraries

JBrowse version: https://github.com/GMOD/jbrowse/archive/1.16.6-release.tar.gz  
 Plugins:   
 - multibigwig (https://github.com/elsiklab/multibigwig )  
 - multiscalebigwig (https://github.com/cmdcolin/multiscalebigwig)  
  
 \*\*Note:\*\* The JBrowse development version is required as integration of different jbrowse plugins are needed.

## 7.3 Steps to process CNVpytor data for JBrowse integration

To generate CNVpytor file for JBrowse visualization:

cnvpytor -root [pytor files] -export jbrowse [optional argument: output path]  
  
Default export directory name:   
 - For single pytor file input: jbrowse\_[pytor file name]  
 - For multiple pytor file input: cnvpytor\_jbrowse\_export

The above command creates all the necessary files that are required to visualize the CNVpytor data.

To view CNVpytor file using JBrowse, users need to install JBrowse and required plugins (See JBrowse version and plugins section). http://localhost/jbrowse/?data=[export directory]

# Example usage  
cnvpytor -root test.pytor -export jbrowse  
http://localhost/jbrowse/?data=jbrowse\_test

#### 7.3.0.1 Data Description

There are mainly two types of data CNVpytor processes. i.e.; Read depth data from alignment file and SNP data from variant file. Depending on the availability of these two input data, the export function works.

For Read depth data, it exports Raw segmented RD, GC corrected Raw Segmented RD, GC corrected RD partition, CNV calling using RD . All of these Read depth signals are plotted on top of each other on a single horizontal track using color gray, black, red and green respectively. For SNP data, it exports Binned BAF, Likelihood of the binned BAF signals. These two signals are plotted on top of each other with gray and red color.

Data

Signal name with color on JBrowse

Read Depth (RD)

Raw Segmented RD (Gray) GC Corrected Raw Segmented RD (Black) GC corrected RD partition (Red) CNV call using RD signals (Green)

SNP

Binned BAF (Gray) Likelihood of the Binned BAF (Red)

CNVpytor does the segmentation for all of the above data based on the user provided bin size. The multiscalebigwig provides the option to show the data based on the visualized area on the reference genome, which means if a user visualizes a small region of the genome it shows small bin data and vice versa.  
!

# About the Authors

These credits are based on our [course contributors table guidelines](https://github.com/jhudsl/OTTR_Template/wiki/How-to-give-credits).

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Lead Content Instructor(s) | [FirstName LastName](link%20to%20personal%20website) |
| Lecturer(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | Delivered the course in some way - video or audio |
| Content Author(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | If any other authors besides lead instructor |
| Content Contributor(s) (include section name/link in parentheses) - make new line if more than one section involved | Wrote less than a chapter |
| Content Editor(s)/Reviewer(s) | Checked your content |
| Content Director(s) | Helped guide the content direction |
| Content Consultants (include chapter name/link in parentheses or word “General”) - make new line if more than one chapter involved | Gave high level advice on content |
| Acknowledgments | Gave small assistance to content but not to the level of consulting |
| **Production** |  |
| Content Publisher(s) | Helped with publishing platform |
| Content Publishing Reviewer(s) | Reviewed overall content and aesthetics on publishing platform |
| **Technical** |  |
| Course Publishing Engineer(s) | Helped with the code for the technical aspects related to the specific course generation |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([ottr](https://github.com/jhudsl/ottr)) | [John Muschelli](https://johnmuschelli.com/), [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| **Art and Design** |  |
| Illustrator(s) | Created graphics for the course |
| Figure Artist(s) | Created figures/plots for course |
| Videographer(s) | Filmed videos |
| Videography Editor(s) | Edited film |
| Audiographer(s) | Recorded audio |
| Audiography Editor(s) | Edited audio recordings |
| **Funding** |  |
| Funder(s) | Institution/individual who funded course including grant number |
| Funding Staff | Staff members who help with funding |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.3 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2022-02-11   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.3)   
## bookdown 0.24 2022-02-11 [1] Github (rstudio/bookdown@88bc4ea)   
## callr 3.4.4 2020-09-07 [1] RSPM (R 4.0.2)   
## cli 2.0.2 2020-02-28 [1] RSPM (R 4.0.0)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.14 2019-05-28 [1] RSPM (R 4.0.3)   
## fansi 0.4.1 2020-01-08 [1] RSPM (R 4.0.0)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## glue 1.6.1 2022-01-22 [1] CRAN (R 4.0.2)   
## htmltools 0.5.0 2020-06-16 [1] RSPM (R 4.0.1)   
## knitr 1.33 2022-02-11 [1] Github (yihui/knitr@a1052d1)   
## lifecycle 1.0.0 2021-02-15 [1] CRAN (R 4.0.2)   
## magrittr 2.0.2 2022-01-26 [1] CRAN (R 4.0.2)   
## memoise 1.1.0 2017-04-21 [1] RSPM (R 4.0.0)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.3.4 2020-08-11 [1] RSPM (R 4.0.2)   
## purrr 0.3.4 2020-04-17 [1] RSPM (R 4.0.3)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## rlang 0.4.10 2022-02-11 [1] Github (r-lib/rlang@f0c9be5)   
## rmarkdown 2.10 2022-02-11 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.2 2020-11-15 [1] CRAN (R 4.0.2)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2022-02-11 [1] Github (R-lib/testthat@e99155a)   
## usethis 2.1.5.9000 2022-02-11 [1] Github (r-lib/usethis@57b109a)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2022-02-11 [1] Github (yihui/xfun@74c2a66)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library

# References