

# Package ‘eDNAfuns’

May 8, 2026

**Type** Package

**Title** Working with Metabarcoding Data in a Tidy Format

**Version** 0.1.0

**Description** A series of R functions that come in handy while working with metabarcoding data. The reasoning of doing this is to have the same functions we use all the time stored in a curated, reproducible way.

In a way it is all about putting together the grammar of the 'tidyverse' from Wickham et al.(2019) <[doi:10.21105/joss.01686](https://doi.org/10.21105/joss.01686)>

with the functions we have used in community ecology compiled in packages like 'vegan' from Dixon (2003) <[doi:10.1111/j.1654-1103.2003.tb02228.x](https://doi.org/10.1111/j.1654-1103.2003.tb02228.x)>

and 'phyloseq' McMurdie & Holmes (2013) <[doi:10.1371/journal.pone.0061217](https://doi.org/10.1371/journal.pone.0061217)>.

The package includes functions to read sequences from FAST(A/Q) into a tibble ('fasta\_reader' and 'fastq\_reader'),

to process 'cutadapt' Martin (2011) <[doi:10.14806/ej.17.1.200](https://doi.org/10.14806/ej.17.1.200)> 'info-

file' output. When it comes to sequence counts across samples, the package works with the long format in mind (a three column 'tibble' with Sample, Sequence and counts ), with functions to move from there to the wider format.

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyData** true

**Suggests** insect, testthat, tidyverse, knitr, rmarkdown

**Imports** dplyr, googlesheets4, phyloseq, purrr, rlang, tibble, tidyr, vegan, ggplot2, vroom, Biostrings, stringr, tidyselect, readr

**Depends** R (>= 4.1.0)

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Ramón Gallego Simón [aut, cre]

**Maintainer** Ramón Gallego Simón <[ramongallego@gmail.com](mailto:ramongallego@gmail.com)>

**Repository** CRAN

**Date/Publication** 2025-10-01 07:00:09 UTC

## Contents

AMPURE . . . . .	2
ASV_table . . . . .	3
count_stop_codons . . . . .	3
eDNAindex . . . . .	5
example_hashes . . . . .	6
fasta_reader . . . . .	6
fasta_writer . . . . .	7
Index_PCR . . . . .	8
metadata . . . . .	8
molarity.data . . . . .	9
mutation . . . . .	10
ng2nM . . . . .	10
OTU_taxonomy . . . . .	11
plot_seq_len_hist . . . . .	12
ps . . . . .	13
read_indexing_PCR . . . . .	13
read_info_file . . . . .	15
tally_wide . . . . .	16
template_PCR . . . . .	16
test_seqs . . . . .	17
tibble_to_comm . . . . .	17
tidy2phyloseq . . . . .	19
training.ASV.table . . . . .	20
training.metadata . . . . .	21
tree . . . . .	21
UDI_Indices . . . . .	22
write_indexing_PCR . . . . .	22
<b>Index</b>	<b>24</b>

---

AMPURE	<i>AMPURE</i>
--------	---------------

---

### Description

Description: Placeholder for AMPURE dataset.

### Usage

AMPURE

### Format

A tibble/data.frame with X rows and Y columns:

**Column1** Description of column1

**Column2** Description of column2 ...

**Source**

Generated by the Author

---

ASV_table	<i>ASV_table</i>
-----------	------------------

---

**Description**

An abundance table in long format.

**Usage**

ASV\_table

**Format**

A data frame with 9 rows and 3 variables:

**sample\_name** Character vector, last digit includes PCR replicate

**Hash** The sequence found, hashed using sha1 for consistency

**nReads** Number of times that hash was found in that sample

**Details**

Contains the number of reads from unique ASVs across different samples.

**Source**

Generated by the Author

---

count_stop_codons	<i>Count stop codons or return translated sequence</i>
-------------------	--

---

**Description**

Given a DNA sequence (either a 'DNAStrng' object or a character string), this function translates it using a specified genetic code and counts the number of stop codons ('\*') in the resulting amino acid sequence. Alternatively, the translated sequence itself can be returned.

**Usage**

```
count_stop_codons(
  sequence = NULL,
  format = "DNAStrng",
  codon = 1,
  dictionary = 5,
  return = "count"
)
```

**Arguments**

sequence	A DNA sequence, either as a [Biostrings::DNASTring] object or as a character string.
format	Input format, either "DNASTring" (default) or "character". If "character", the sequence must consist of 'A', 'C', 'G', 'T' only.
codon	Integer giving the starting codon position (usually 1, 2, or 3).
dictionary	Integer specifying which translation code to use. See [Biostrings::GENETIC_CODE_TABLE] for all options. Common examples:

id	name
1	Standard
2	Vertebrate Mitochondrial
3	Yeast Mitochondrial
4	Mold/Protozoan/Coelenterate/Mycoplasma/Spiroplasma Mitochondrial
5	Invertebrate Mitochondrial
6	Ciliate/Dasycladacean/Hexamita Nuclear
9	Echinoderm/Flatworm Mitochondrial
10	Euplotid Nuclear
11	Bacterial, Archaeal, and Plant Plastid
12	Alternative Yeast Nuclear
13	Ascidian Mitochondrial
14	Alternative Flatworm Mitochondrial
15	Blepharisma Macronuclear
16	Chlorophycean Mitochondrial
21	Trematode Mitochondrial
22	Scenedesmus obliquus Mitochondrial
23	Thraustochytrium Mitochondrial
24	Pterobranchia Mitochondrial
25	Candidate Division SR1 and Gracilibacteria
26	Pachysolen tannophilus Nuclear

return	Either "count" (default) to return the number of stop codons, or any other value to return the translated amino acid sequence.
--------	--

**Details**

The function uses [Biostrings::translate()] with the specified starting position and genetic code. Translation warnings are suppressed.

**Value**

If 'return = "count"', an integer giving the number of stop codons. Otherwise, a character string with the translated amino acid sequence.

**Author(s)**

Ramon Gallego, 2021

**Examples**

```

if (requireNamespace("Biostrings", quietly = TRUE)) {
  library(Biostrings)
  seq <- DNASTring("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG")
  count_stop_codons(seq, codon = 1, dictionary = 1)
  count_stop_codons(seq, codon = 1, dictionary = 1, return = "translation")
}

```

---

eDNAindex	<i>Create scaled relative proportions of the number of reads of taxa in different samples</i>
-----------	---

---

**Description**

This function takes a long ASV table (tibble) and creates a new column with the relative proportions scaled to their maximum, to avoid the dominance of species with better primer efficiency

**Usage**

```

eDNAindex(
  tibble,
  Sample_column = Sample,
  OTU_column = Hash,
  Counts_column = nReads,
  Biological_replicate_column = NULL,
  ...
)

```

**Arguments**

tibble	A tibble the ASV table in a long format, with at least three columns, Sample_column, OTU_column, Counts_column
Sample_column	The column indicating the sample.
OTU_column	The column indicating the OTU/ASV.
Counts_column	The column (numeric) with the number of sequences from that OTU in that sample.
Biological_replicate_column	The column representing replicate measurements of Sample_column.
...	Any extra columns that want to be added to the final dataset (either taxonomical information about OTUs, or metadata information about the samples)

**Value**

A tibble with at least the columns Sample\_column, OTU\_column and Normalized.reads

**Examples**

```
data("training.ASV.table")
eDNAindex(training.ASV.table, Sample_column = Sample_name)
```

---

example_hashes	<i>example_hashes</i>
----------------	-----------------------

---

**Description**

Placeholder description for example\_hashes dataset.

**Usage**

```
example_hashes
```

**Format**

A vector or data frame with X rows/columns

**Source**

Generated by the Author

---

fasta_reader	<i>Read FASTA or FASTQ files into a tibble</i>
--------------	--

---

**Description**

Functions to read sequence files into a tidy data frame with one row per sequence.

**Usage**

```
fasta_reader(path_to_fasta)

fastq_reader(path_to_fastq, keepQ = FALSE)
```

**Arguments**

path_to_fasta	Character. Path to a FASTA file.
path_to_fastq	Character. Path to a FASTQ file.
keepQ	Logical. If 'TRUE', keep a third column with quality scores when reading FASTQ files. Default is 'FALSE'.

**Value**

- 'fasta\_reader()': A tibble with columns: - 'header': sequence identifiers (without the '>'). - 'seq': nucleotide sequences.
- 'fastq\_reader()': A tibble with columns: - 'header': sequence identifiers (without the '@'). - 'seq': nucleotide sequences. - 'Qscores' (optional): quality scores, if 'keepQ = TRUE'.

**Examples**

```
fasta_df <- fasta_reader(system.file("extdata", "test.fasta", package="eDNAfuns"))
fastq_df <- fastq_reader(system.file("extdata", "test.fastq", package="eDNAfuns"), keepQ = TRUE)
```

---

fasta_writer	<i>Read FASTA or FASTQ files into a tibble</i>
--------------	--

---

**Description**

Functions to read sequence files into a tidy data frame with one row per sequence.

**Usage**

```
fasta_writer(df, sequence, header, file.out)
```

```
fastq_writer(df, sequence, header, Qscores, file.out)
```

**Arguments**

df	A dataframe where the sequence information is stored, one sequence per row
sequence	The name (unquoted) of the column where the sequence information (as characters) is stored
header	The name (unquoted) of the column where the header information is stored
file.out	Character. Path to the location where to write the file.
Qscores	The name (unquoted) of the column where the Quality information (encoded as characters) is stored

**Value**

- 'fasta\_reader()': A tibble with columns: - 'header': sequence identifiers (without the '>'). - 'seq': nucleotide sequences.
- 'fastq\_reader()': A tibble with columns: - 'header': sequence identifiers (without the '@'). - 'seq': nucleotide sequences. - 'Qscores' (optional): quality scores, if 'keepQ = TRUE'.

**Examples**

```

fasta_file <- tempfile(fileext = ".fasta")

fasta_df <- fasta_reader(system.file("extdata", "test.fasta", package="eDNAfuns"))

fasta_writer(fasta_df, sequence=seq,
             header = header,
             file.out = fasta_file)

fastq_file <- tempfile(fileext = ".fastq")

fastq_df <- fastq_reader(system.file("extdata", "test.fastq", package="eDNAfuns"), keepQ = TRUE)

fastq_writer(fastq_df, sequence=seq,
            header = header, Qscores= Qscores,
            file.out = fastq_file)

```

---

Index\_PCR

*Index\_PCR*


---

**Description**

Placeholder description for IndexPCR template web address.

**Usage**

Index\_PCR

**Format**

A character vector

**Source**

Generated by the Author, stored in googledrive

---

metadata

*metadata*


---

**Description**

Placeholder description for metadata dataset.

**Usage**

metadata

**Format**

A data frame with X rows and Y columns:

**sample\_name** Name of sample

**Treatment** either control or treatment

**Day** either 1 or 2

**Source**

Generated by the Author

---

*molarity.data*

*molarity.data*

---

**Description**

Placeholder description for *molarity.data* dataset.

**Usage**

*molarity.data*

**Format**

A tibble/data.frame with X rows and Y columns

**Sample** Name of sample

**Molarity** in nM

**Amp\_len** length of amplicon in bp

**mass** in ng/ $\mu$ l

**Source**

Generated by the Author

---

mutation	<i>Generate mutated DNA sequences</i>
----------	---------------------------------------

---

### Description

This function takes DNA sequences and generates mutated variants. Useful for testing classification algorithms on sequences with either PCR-induced or naturally occurring mutations.

### Usage

```
mutation(sequence = NULL, format = "bin", n.mutations = NA, prob.mutation = NA)
```

### Arguments

sequence	A list of DNA sequences, either as "DNABin" objects or character vectors.
format	Character. Format of the input sequences. "bin" for DNABin, "char" for character vectors.
n.mutations	Integer. Number of mutations to introduce per sequence. Exclusive with prob.mutation.
prob.mutation	Numeric. Probability of mutation per position. Exclusive with n.mutations.

### Value

A list of mutated sequences of the same class as the input.

### Examples

```
data("test_seqs")
mutation(test_seqs, n.mutations = 2)
mutation(test_seqs, prob.mutation = 0.1)
seqs <- fasta_reader(system.file("extdata", "test.fasta", package="eDNAfuns"))
mutation(seqs$seq, format = "char", n.mutations = 1)
```

---

ng2nM	<i>functions to translate mass into molarity and vice versa, given we are talking about double stranded DNA it requires two inputs, the mass (or molarity) and the length of the DNA fragment It works with the two most common concentrations used in Molecular Ecology labs ng/<math>\mu</math>l for mass nM for molarity</i>
-------	---

---

### Description

functions to translate mass into molarity and vice versa, given we are talking about double stranded DNA it requires two inputs, the mass (or molarity) and the length of the DNA fragment It works with the two most common concentrations used in Molecular Ecology labs ng/ $\mu$ l for mass nM for molarity

**Usage**

```
ng2nM(ng, length_amplicon)
```

```
nM2ng(nM, length_amplicon)
```

**Arguments**

```
ng          Numeric. the concentration in ng per  $\mu$ L
length_amplicon Integer. The length of the DNA fragment in base pairs.
nM          Numeric. The concentration in nmoles per litre
```

**Value**

Numeric. The equivalent concentration in nmoles per litre

Numeric. The equivalent concentration in ng per  $\mu$ L

**Examples**

```
data("molarity.data")
ng2nM(ng=molarity.data$mass, length_amplicon = molarity.data$Amp_len)
```

```
data("molarity.data")
nM2ng(nM=molarity.data$Molarity, length_amplicon = molarity.data$Amp_len)
```

---

OTU\_taxonomy

*OTU\_taxonomy*

---

**Description**

Placeholder description for OTU\_taxonomy dataset.

**Usage**

```
OTU_taxonomy
```

**Format**

A dataframe of DNA sequences names and their taxonomy, with 3 rows and 7 columns

**Hash** Sequence identifier

**Kingdom** taxon info at that rank

**Phylum** taxon info at that rank

**Class** taxon info at that rank

**Order** taxon info at that rank

**Family** taxon info at that rank

**Genus** taxon info at that rank

### Source

Generated by the Author

---

plot\_seq\_len\_hist      *Plot sequence length distribution*

---

### Description

This function takes a tibble produced by ‘fasta\_reader()’ or ‘fastq\_reader()’ and plots the distribution of sequence lengths.

### Usage

```
plot_seq_len_hist(Hash_tibble, length_col, binwidth = 1, label_interval = 5)
```

### Arguments

**Hash\_tibble**      A tibble containing a numeric column with sequence lengths.

**length\_col**      The column of ‘Hash\_tibble’ containing sequence lengths (unquoted).

**binwidth**        Width of histogram bins. Default = 1.

**label\_interval**   Interval for x-axis labels. Default = 5.

### Value

A ggplot object.

### Examples

```
data("example_hashes")  
plot_seq_len_hist(example_hashes, seq_len)
```

---

 ps

 ps
 

---

**Description**

Placeholder description for phyloseq object.

**Usage**

ps

**Format**

Formal class 'phyloseq' with five slots

**Source**

Generated by the Author

---

 read\_indexing\_PCR

*Read Indexing PCR Spreadsheet*


---

**Description**

Reads data from a Google Sheets-based indexing PCR template. You can access the template [here](#), create a copy in your Google Drive, and then create copies for each of your multiplexing PCR experiments.

Reads data from a Google Sheets-based PCR template. You can access the template [here](#), create your own copy, and then create one copy per PCR reaction you want to keep track of.

**Usage**

read\_indexing\_PCR(ss)

read\_step1\_PCR(ss, trim = TRUE, name = TRUE)

**Arguments**

ss	Google Sheet ID or URL of the indexing PCR spreadsheet.
trim	Logical. If TRUE, removes rows where 'Sample' is NA (default: TRUE).
name	Logical. If TRUE, adds the spreadsheet name as a 'PCR' column in the sample sheet (default: TRUE).

**Details**

The function extracts sample information across all plates and returns a tidy dataframe.

Captures reagent mix, cycling conditions, and sample information.

**Value**

A tibble with columns:

**Well** Well position

**Sample** Sample name

**Barcode** Barcode assigned to the sample

**Set** Barcode set identifier

A named list with three elements:

**PCR\_mix** tibble of reagents and volumes.

**Cycling** tibble of PCR cycling conditions.

**Samples** tibble of samples with columns 'Well', 'Sample', 'Success', 'Notes', and optionally 'PCR'.

**See Also**

[read\_step1\_PCR()] for reading normal PCR spreadsheets.

[read\_indexing\_PCR()] for reading multiplexing PCR spreadsheets.

**Examples**

```
# Examples are not executed because the function requires identification in Google
## Not run:
data("Index_PCR")
read_indexing_PCR(Index_PCR)

## End(Not run)

# Examples are not executed because the function requires identification in Google
## Not run:
data("template_PCR")
read_step1_PCR(template_PCR)

## End(Not run)
```

---

read_info_file	<i>Read cutadapt info files</i>
----------------	---------------------------------

---

## Description

Read the ‘-info-file’ output generated by cutadapt for adapter trimming. Column structure differs depending on whether the input came from Nanopore (ONT) or Illumina sequencing.

## Usage

```
read_info_file(file, delim = "\t", col_select = NULL, ...)
```

## Arguments

file	Path to the cutadapt info file (TSV).
delim	Field delimiter (default = "\t").
col_select	Optional tidyselect specification of which columns to read.
...	Additional arguments passed to [vroom::vroom()].

## Details

- ‘read\_info\_file()’ (Illumina) returns columns: ‘Seq.id’, ‘n\_errors’, ‘start\_adap’, ‘end\_adap’, ‘seq\_before\_adap’, ‘matching\_seq’, ‘seq\_after\_adap’, ‘adap\_name’, ‘QScores\_seq\_before’, ‘QScores\_matching’, ‘QScores\_after’.

## Value

A tibble with the parsed ‘cutadapt’ information. Column names are standardized.

## Examples

```
test_info <- system.file("extdata", "cutadapt_info_illumina.txt", package="eDNAfuns")
df_illumina <- read_info_file(test_info)
```

---

tally_wide	<i>Create contingency tables with two variables</i>
------------	---

---

**Description**

This function takes a tibble and create human readable contingency tables from two variables, either by showing number of cases in each combination or weighted by the sum of a numerical variable

**Usage**

```
tally_wide(tibble, rows, cols, wt = NULL, ...)
```

**Arguments**

tibble	A tibble containing at least two columns
rows	The column with the levels included as rows in the final table.
cols	The column with the levels included as columns in the final table.
wt	The column (numeric) whose values to add in order to fill the cells. If wt = NULL (the default), counts are returned instead of weighted sums.
...	Any parameters that can be passed to tally_wide 'values_fill' is a useful one

**Value**

A tibble

**Examples**

```
df <- tibble::tibble(
  group = c("A", "A", "B", "B", "B"),
  outcome = c("yes", "no", "yes", "yes", "no")
)
tally_wide(df, rows = group, cols = outcome)
```

---

template_PCR	<i>template_PCR</i>
--------------	---------------------

---

**Description**

Placeholder description for Normal PCR template web address.

**Usage**

```
template_PCR
```

**Format**

A character vector

**Source**

Generated by the Author, stored in googledrive

---

test_seqs	<i>test_seqs</i>
-----------	------------------

---

**Description**

Placeholder description for test\_seqs dataset.

**Usage**

test\_seqs

**Format**

An object of class DNABin of length 3.

**Source**

Generated by the Author

---

tibble_to_comm	<i>Convert a long tibble to a community matrix</i>
----------------	--

---

**Description**

Converts a long-format table of sequence counts into a wide community matrix (samples  $\times$  taxa) suitable for vegan or other community ecology tools.

Produces a distance matrix between samples using vegan's `vegdist()`. Optionally applies a data transformation before distance calculation.

Returns the environmental data frame (sample metadata) from a long sequence/OTU tibble by removing taxon and abundance columns.

**Usage**

```
tibble_to_comm(long.table, taxon, Abundance, sample.name)
```

```
tibble_to_dist(
  long.table,
  taxon,
  Abundance,
  sample.name,
  distance = "bray",
  transformation = NULL,
  ...
)
```

```
tibble_to_env(long.table, taxon, Abundance, sample.name, ...)
```

**Arguments**

long.table	A tibble with sample, taxon, abundance, and metadata columns.
taxon	Column containing taxa/OTU IDs (unquoted).
Abundance	Column with counts/abundance values (unquoted).
sample.name	Column with sample IDs (unquoted).
distance	Distance metric to use (default = "bray").
transformation	Optional transformation (e.g. "hellinger", "log"). See <code>vegan::decostand</code> documentation for a great explanation of all transformations
...	Additional metadata columns to retain.

**Value**

A numeric matrix with taxa as columns and samples as row names.

A dist object.

A tibble of unique sample-level metadata.

**Examples**

```
data("ASV_table")
tibble_to_comm(ASV_table,
               taxon = Hash,
               Abundance = nReads,
               sample.name = sample_name)
data("ASV_table")
tibble_to_dist(ASV_table,
               taxon = Hash,
               Abundance = nReads,
               sample.name = sample_name,
               distance = "bray",
               transformation = "hellinger")
```

```

data("ASV_table")
data("metadata")
dplyr::inner_join(ASV_table,metadata) |>
  tibble_to_env(taxon = Hash,
               Abundance = nReads,
               sample.name = sample_name)

```

---

tidy2phyloseq

---

*Convert a tidy ASV table to a phyloseq object*


---

## Description

This function converts a tidy ASV table, along with optional taxonomy and metadata, into a ‘phyloseq’ object.

Extracts ASV counts, taxonomy, and metadata from a ‘phyloseq’ object into tidy data frames.

## Usage

```

tidy2phyloseq(
  ASV_table,
  OTU_taxonomy = NULL,
  metadata = NULL,
  Taxa = "sseqid",
  Sample = "sample_name",
  Reads = "nr",
  tree = NULL
)

```

```

phyloseq2tidy(phylo_obj, Taxa = "sseqid", Sample = "sample_name", Reads = "nr")

```

## Arguments

ASV_table	A tidy data.frame/tibble of ASV counts.
OTU_taxonomy	A data.frame with OTU taxonomy. Optional.
metadata	A data.frame with sample metadata. Optional.
Taxa	Column name for OTU IDs in the output (default: "sseqid").
Sample	Column name for sample IDs in the output (default: "sample_name").
Reads	Column name for read counts in the output (default: "nr").
tree	A phylogenetic tree of class ‘phylo’. Optional.
phylo_obj	A phyloseq object.

**Value**

A ‘phyloseq’ object combining OTU table, taxonomy, metadata, and optionally a tree.

A list with three tibbles:

**ASV\_table** Long-format tibble of counts: Sample, Taxa, Reads.

**taxonomy** Taxonomy table as tibble (NULL if none in phyloseq).

**metadata** Sample metadata as tibble (NULL if none in phyloseq).

**Examples**

```
data("ASV_table")
data("metadata")
data("OTU_taxonomy")
ps <- tidy2phyloseq(ASV_table = ASV_table,
                   OTU_taxonomy = OTU_taxonomy,
                   metadata = metadata,
                   Taxa = "Hash",
                   Reads = "nReads")
```

```
data("ps")
tidy_list <- phyloseq2tidy(ps)
```

---

training.ASV.table	<i>training.ASV.table</i>
--------------------	---------------------------

---

**Description**

Placeholder description for training.ASV.table dataset.

**Usage**

```
training.ASV.table
```

**Format**

A data frame with 1909 rows and 4 variables:

**Sample\_name** Character vector, last digit includes PCR replicate

**Locus** Which locus the sequences are from, numeric

**Hash** The sequence found, hashed using sha1 for consistency

**nReads** Number of times that hash was found in that sample

**Source**

Generated by the Author

---

training.metadata	<i>training.metadata</i>
-------------------	--------------------------

---

**Description**

Placeholder description for training.metadata dataset.

**Usage**

training.metadata

**Format**

A data frame with 95 rows and 8 columns

**Sample\_name** Character vector, last digit includes PCR replicate

**eDNA.sample** Biological sample

**rep** PCR replicate

**Transect** Transect of origin

**position** Stop in Transect

**depth** in the water column

**lat** latitude in decimal degrees

**lon** longitude in decimal degrees

**Source**

Generated by the Author

---

tree	<i>tree</i>
------	-------------

---

**Description**

Placeholder description for tree dataset.

**Usage**

tree

**Format**

An object of class phylo or similar

**Source**

Generated by the Author

---

 UDI\_Indices

*UDI\_Indices*


---

**Description**

Placeholder description for UDI\_Indices dataset.

**Usage**

UDI\_Indices

**Format**

A data frame with 384 rows and 6 columns

**Index** Character vector

**Bases\_in\_Adapter\_i7** Seqs in i7

**Bases\_for\_Sample\_Sheet\_i7** Seqs to include

**Bases\_for\_Sample\_Sheet\_i5** Seqs to include in some illumina platofoms

**Bases\_for\_Sample\_Sheet\_i5\_B** Seqs to include in other illumina platofoms

**Set** A, B, C, or D

**Source**

Generated by the Author, data from Illumina

---

 write\_indexing\_PCR

*Write Indexing PCR Spreadsheet*


---

**Description**

Creates a new Google Sheet for indexing PCRs from a template, fills in the sample information, and writes it into the correct ranges of the sheet.

**Usage**

```
write_indexing_PCR(
  data,
  name,
  ss_template = "1naS-F_dj4SNmND5nJ5TKhMX3TikmRS00ILKjfg_Ucgc"
)
```

## Arguments

data	A tibble or dataframe with at least the columns: <b>Well</b> Well position <b>Sample</b> Sample identifier <b>Column</b> Plate column (used to split data across sheet sections)
name	Character. Name for the new Google Sheet that will be created.
ss_template	Character. ID of the template sheet to copy from (default: "1naS-F_dj4SNmND5nJ5TKhMX3TikmRS00")

## Details

This function: 1. Creates a new Google Sheet with the given 'name'. 2. Copies the template sheet into it. 3. Splits the input 'data' by plate column. 4. Writes each split dataframe into its designated range of the sheet.

## Value

A Google Sheet object (as returned by [googlesheets4::gs4\_create()]) with the data written into the correct plate layout.

## Examples

```
#Examples are not executed because the function requires identification in Google
## Not run:
my_data <- tibble::tibble(
  Well = c("A1", "A2"),
  Sample = c("Sample1", "Sample2"),
  Column = c(1, 2)
)
write_indexing_PCR(my_data, "PCR_001")

## End(Not run)
```

# Index

## \* datasets

- AMPURE, 2
  - ASV\_table, 3
  - example\_hashes, 6
  - Index\_PCR, 8
  - metadata, 8
  - molarity.data, 9
  - OTU\_taxonomy, 11
  - ps, 13
  - template\_PCR, 16
  - test\_seqs, 17
  - training.ASV.table, 20
  - training.metadata, 21
  - tree, 21
  - UDI\_Indices, 22
- AMPURE, 2
- ASV\_table, 3
- count\_stop\_codons, 3
- eDNAindex, 5
- example\_hashes, 6
- fasta\_reader, 6
- fasta\_writer, 7
- fastq\_reader (fasta\_reader), 6
- fastq\_writer (fasta\_writer), 7
- Index\_PCR, 8
- metadata, 8
- molarity.data, 9
- mutation, 10
- ng2nM, 10
- nM2ng (ng2nM), 10
- OTU\_taxonomy, 11
- phyloseq2tidy (tidy2phyloseq), 19
- plot\_seq\_len\_hist, 12
- ps, 13
- read\_indexing\_PCR, 13
- read\_info\_file, 15
- read\_step1\_PCR (read\_indexing\_PCR), 13
- tally\_wide, 16
- template\_PCR, 16
- test\_seqs, 17
- tibble\_to\_comm, 17
- tibble\_to\_dist (tibble\_to\_comm), 17
- tibble\_to\_env (tibble\_to\_comm), 17
- tidy2phyloseq, 19
- training.ASV.table, 20
- training.metadata, 21
- tree, 21
- UDI\_Indices, 22
- write\_indexing\_PCR, 22