Package: vcf2mafR (via r-universe)

May 21, 2026

Title What the Package Does (One Line, Title Case)

Version 0.0.0.9000

Description What the package does (one paragraph).

License MIT + file LICENSE

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports assertions (>= 0.0.0.9000), cli, data.table, dplyr, mutationtypes (>= 0.0.0.9000), rlang, stringr, tibble, vcfR

Remotes selkamand/assertions, selkamand/mutationtypes

URL https://github.com/selkamand/vcf2mafR

BugReports https://github.com/selkamand/vcf2mafR/issues

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev

Repository https://ccicb.r-universe.dev

Date/Publication 2023-11-29 12:31:10 UTC

RemoteUrl https://github.com/CCICB/vcf2mafR

RemoteRef HEAD

RemoteSha 5be8185d179a04014e5ed72624e4cf4413367506

Contents

`cdna_to_transcript_length` *Convert cDNA_position to numeric indication of transcript length*

Description

This function just pulls out digits from the cDNA_position column. Note transcript length isn't encoded into cDNA position column UNLESS –total_length flag is on when running vep (turns format into cdna_position/total_length) It is not possible to turn this flag on using VEP online. If we can't find total length - we just return 0, so for online version we just won't have a transcript sort.

Usage

```
cdna_to_transcript_length(cdna)
```

Arguments

`cdna`	vector representing cDNA_position VEP annotations.

Value

numeric transcript length passed from VEP cDNA_position

`df2maf` *Convert VCF-like data.frame to MAF*

Description

Converts a VEP-like (chr-pos-ref-alt-consequence-sample style) dataframe with to a minimal MAF dataframe. **Input dataframe should include the columns:**

- **chr**
- **pos** (1-based)
- **sample**
- **ref**
- **alt**
- **consequence** (Sequence Ontology terms)
- **gene**

Usage

```
df2maf(
  data,
  ref_genome,
  keep_all = TRUE,
  col_chrom = "chr",
  col_pos = "pos",
  col_sample_identifier = "sample",
  col_ref = "ref",
  col_alt = "alt",
  col_consequence = "consequence",
  consequence_dictionary = c("SO", "PAVE", "AUTO"),
  missing_to_silent = FALSE,
  col_gene = "gene",
  col_center = NULL,
  col_entrez_gene_id = NULL,
  col_dbSNP_rsid = NULL,
  col_dbSNP_validation_status = NULL,
  col_matched_normal_sample_identifier = NULL,
  col_sequencer = NULL,
  col_sequence_source = NULL,
  col_mutation_status = NULL,
  verbose = TRUE
)
```

Arguments

`data`	a data.frame with 1 row per mutation in a cohort (data.frame)
`ref_genome`	name of the reference genome used to call variants (string)
`keep_all`	keep all columns in the original data.frame? (flag). If FALSE, only includes the minimal required columns and any column explicitly mapped using `col_<name>` arguments.
`col_chrom`	name of column describing chromosome of the mutation (string)
`col_pos`	name of column describing the 1-based position of the mutation (string)
`col_sample_identifier`	name of column describing the sample containing the mutation (string)
`col_ref`	name of column describing the reference allele (string)
`col_alt`	name of column describing the alternate allele (string)
`col_consequence`	name of column describing the consequence of the mutation (in SO ontology terms e.g. those that VEP would use OR PAVE ontology terms) containing the mutation (string)
`consequence_dictionary`	What dictionary is to describe variant consequences (SO / PAVE / etc)
`missing_to_silent`	Assyne any missing (NA) or empty (") consequences are 'Silent' mutations

`col_gene` name of column containing Hugo_Symbol of the gene affected by the mutation (string)

`col_center` name of column containing the genome sequencing center reporting the variant (string)

`col_entrez_gene_id` name of column containing entrez gene IDs (string)

`col_dbSNP_rsid` name of column describing the dbSNP rsid of the variant, or "novel" if there is no dbSNP record (string)

`col_dbSNP_validation_status` name of column describing the validation status of the variant. Elements must be one of by1000genomes;by2Hit2Allele; byCluster; byFrequency; byHapMap; byOtherPop; bySubmitter; alternate_allele (string)

`col_matched_normal_sample_identifier` name of column describing the matched normal sample identifier

`col_sequencer` name of column describing the instrument used to produce data (string)

`col_sequence_source` name of column describing the molecular assay type used to produce the analytes used for sequencing (string). Elements are usually one of 'WGS', 'WGA', 'WXS', 'RNA-seq', etc

`col_mutation_status` name of column describing the mutation status (string). Elements must be one of: None, Germline, Somatic, LOH, Post-transcriptional modification, or Unknown

`verbose` verbose (flag)

Details

Alternate column names can be used, but require **col_chrom**, **col_pos**, **col_** arguments to be supplied.

Value

a maf-like data.frame (data.table)

Examples

```
# Start with a vcf/maf-like 'chr-pos-ref-alt-consequence-sample' data.frame
df = read.csv(system.file(package = "vcf2mafR", "testfiles/test_so.tsv"), sep = "\t")

# Convert to a MAF dataframe
df2maf(df, ref_genome = "hg38")
```

`paths_to_sample_names` *Extract Sample Names From VCF filepath*

Description

Extract Sample Names From VCF filepath

Usage

```
paths_to_sample_names(
  filenames,
  extract = c("before_dot", "before_underscore")
)
```

Arguments

`filenames`	paths to vcf files
`extract`	what text do we extract as sample name. See details

Details

Extraction	Explanation
before_dot	**sample_name**.otherinfo.vcf
before_underscore	**samplename**_other_info.vcf

Value

sample names extracted from filename (character)

Examples

```
path <- "/path/to/sample_name.otherinfo.vcf"
paths_to_sample_names(path)
```

valid_maf_columns *Maf Column Names*

Description

List all valid MAF columns based on GDC specification

Usage

```
valid_maf_columns()
```

Value

Names of each GDC MAF column in the order they appear (character)

Examples

```
valid_maf_columns()
```

vcf2df *Convert a vepped VCF into a table ready to be passed into df2maf*

Description

Expects a vepped VCF. VEP must be run with a couple of options:

1. Identifiers: Gene symbol & Transcript Version & (HGVS: optional)
2. Transcript annotation: Transcript biotype & 'Identify canonical transcripts'
3. (Optional) If using the commandline version of vep also make sure to use –total_length (helps in transcript choice)

Usage

```
vcf2df(
  vcf,
  tumor_id = vcf_tumor_id,
  normal_id = vcf_normal_id,
  vcf_tumor_id = "TUMOR",
  vcf_normal_id = "NORMAL",
  debug_mode = FALSE,
  verbose = TRUE
)
```

Arguments

vcf	path to a VEP-annotated VCF file
tumor_id	desired value of Tumor_Sample_Barcode in maf. By default will use the name of the tumor sample in the VCF (string)
normal_id	desired value of Matched_Norm_Sample_Barcode in maf. By default will use the name of the normal sample in the VCF (string)
vcf_tumor_id	the sample ID describing the tumor in the (string)
vcf_normal_id	the sample ID describing the normal in the (string)
debug_mode	run in debug mode (flag)
verbose	(flag)

Details

There are a couple of different types of inputs we supp

Value

data.frame

Examples

```
path_vcf <- system.file(package = "vcf2mafR", "testfiles/test_b38.vepgui.vcf")
vcf2df(path_vcf)
```

---

vcf2maf *Convert Annotated VCF to MAF-compatible data.frame*

Description

Expects a vepped VCF. VEP must be run with a couple of options:

1. Identifiers: Gene symbol & Transcript Version & (HGVS: optional)
2. Transcript annotation: Transcript biotype & 'Identify canonical transcripts'
3. (Optional) If using the commandline version of vep also make sure to use –total_length (helps in transcript choice)

Usage

```
vcf2maf(
  vcf,
  ref_genome,
  tumor_id = vcf_tumor_id,
  normal_id = vcf_normal_id,
  vcf_tumor_id = "TUMOR",
  vcf_normal_id = "NORMAL",
```

```
  missing_to_silent = TRUE,
  verbose = TRUE,
  debug_mode = FALSE
)
```

Arguments

vcf	path to a VEP-annotated VCF file
ref_genome	name of the reference genome used to call variants (string)
tumor_id	desired value of Tumor_Sample_Barcode in maf. By default will use the name of the tumor sample in the VCF (string)
normal_id	desired value of Matched_Norm_Sample_Barcode in maf. By default will use the name of the normal sample in the VCF (string)
vcf_tumor_id	the sample ID describing the tumor in the (string)
vcf_normal_id	the sample ID describing the normal in the (string)
missing_to_silent	Assyne any missing (NA) or empty (") consequences are 'Silent' mutations
verbose	(flag)
debug_mode	run in debug mode (flag)

Value

a maf compatible data.frame

Examples

```
path_vcf_vepped <- system.file("testfiles/test_b38.vepgui.vcf", package = "vcf2mafR")
vcf2maf(vcf = path_vcf_vepped, ref_genome = "b38")
```

---

vcfs2maf *Convert several VCF files into a cohort-level MAF*

Description

Take multiple VCF files, each representing a single tumour-normal file and combine into one big maf

Usage

```
vcfs2maf(
  vcfs,
  ref_genome,
  tumor_id = paste0("Tumor", seq_len(length(vcfs)), times = length(vcfs)),
  vcf_tumor_id = "TUMOR",
```

```
  vcf_normal_id = "NORMAL",
  parse_tumor_id_from_filename = TRUE,
  extract = c("before_dot", "before_underscore"),
  verbose = TRUE
)
```

Arguments

vcfs	path to vcf files (character)
ref_genome	name of the reference genome used to call variants (string)
tumor_id	desired value of Tumor_Sample_Barcode in maf. By default will use the name of the tumor sample in the VCF (string)
vcf_tumor_id	a vector describing what the tumor_names to expect in a VCF are
vcf_normal_id	the sample ID describing the normal in the (string)
parse_tumor_id_from_filename	should tumour id be parsed from filename (flag)
extract	what text do we extract as sample name. See details
verbose	(flag)

Value

a maf-compatible data.frame

Examples

```
vcf_filepaths = dir(system.file(package='vcf2mafR', 'testfiles/cohort_of_vcfs/'), full.names = TRUE)
vcfs2maf(vcf_filepaths, ref_genome = "b38")
```

---

vep_rank_biotypes *Rank VEP biotypes*

Description

This function takes a vector of VEP annotated biotypes and returns their ranks based on priority.

Usage

```
vep_rank_biotypes(biotypes)
```

Arguments

biotypes	A character vector of VEP annotated biotypes.

Value

A numeric vector where values represent the priority of each biotype (lower number = higher priority).

Examples

```
vep_rank_biotypes(c("protein_coding", "miRNA", "pseudogene"))
```

`vep_rank_consequences` *Rank VEP consequences*

Description

This function takes a vector of VEP annotated biotypes and returns their ranks based on priority. If there are multiple consequences in one string (e.g. splice_region_variant&TFBS_ablation) this function will automatically return the priority for the most severe event

Usage

```
vep_rank_consequences(consequences)
```

Arguments

`consequences`	A character vector of VEP annotated consequences.

Value

A numeric vector where values represent the priority of each consequence (lower number = higher priority).

Examples

```
vep_rank_consequences(c("protein_coding", "miRNA", "pseudogene"))
```

Index