Package 'STRprofilerR'

November 7, 2026

Title Compare Short Tandem Repeat Profiles for Model Authentication

Version 0.99.1

Description Compares short tandem repeat (STR) profiles to authenticate biomedical models such as cell lines and xenografts against the primary tissue samples they were derived from. Provides the Tanabe (Sorenson-Dice), Masters (versus query), and Masters (versus reference) similarity scores, heuristics for detecting sample mixing, flexible ingest of wide and long profile formats, comparison against user-supplied reference databases, and querying of the Cellosaurus knowledge base through the CLASTR REST API. A command line application is included for use in automated workflows, and a Shiny application for interactive queries.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/j-andrews7/STRprofilerR

BugReports https://github.com/j-andrews7/STRprofilerR/issues

biocViews Software, QualityControl, Genetics, CellBasedAssays

Depends R (>= 4.6.0)

Imports httr2, IRanges, Matrix, methods, S4Vectors, stats, tools, utils

Suggests BiocStyle, bslib (>= 0.7.0), curl, DT, htmlwidgets, jsonlite, knitr, Rapp, readxl, rmarkdown, shiny (>= 1.8.1), SummarizedExperiment, testthat (>= 3.0.0), withr, writexl

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

git_url https://git.bioconductor.org/packages/STRprofilerR

git_branch devel

git_last_commit fa9daea

git_last_commit_date 2026-10-13

Repository Bioconductor 3.24

Date/Publication 2026-11-06

Author Jared Andrews [aut, cre] (ORCID: <https://orcid.org/0000-0002-0780-6248>)

Maintainer Jared Andrews <jared.andrews07@gmail.com>

Contents

STRprofilerR-package . . . 2
alleles . . . 3
amelogeninMarkers . . . 4
classifyMarkers . . . 5
clastrBatchQuery . . . 6
clastrMarkers . . . 7
clastrQuery . . . 8
cleanAlleles . . . 10
compareProfiles . . . 11
flagMixedSamples . . . 13
markerAliases . . . 14
pdxSTRProfiles . . . 15
readSTRProfiles . . . 16
scoreProfiles . . . 18
scoreQuery . . . 20
scores . . . 21
STRComparison-class . . . 23
strHTMLTable . . . 24
STRprofilerApp . . . 25
STRProfiles . . . 27
STRProfiles-class . . . 28
STRProfiles-methods . . . 29
summarizeMatches . . . 30
writeSTRProfiles . . . 32
writeSTRResults . . . 33

Index **35**

`STRprofilerR-package` *STRprofilerR: Compare Short Tandem Repeat Profiles for Model Authentication*

Description

Compares short tandem repeat (STR) profiles to authenticate biomedical models such as cell lines and xenografts against the primary tissue samples they were derived from. Provides the Tanabe (Sorenson-Dice), Masters (versus query), and Masters (versus reference) similarity scores, heuristics for detecting sample mixing, flexible ingest of wide and long profile formats, comparison against user-supplied reference databases, and querying of the Cellosaurus knowledge base through the CLASTR REST API. A command line application is included for use in automated workflows, and a Shiny application for interactive queries.

Author(s)

Maintainer: Jared Andrews <jared.andrews07@gmail.com> (ORCID)

Authors:

- Jared Andrews <jared.andrews07@gmail.com> (ORCID)

See Also

Useful links:

- https://github.com/j-andrews7/STRprofilerR
- Report bugs at https://github.com/j-andrews7/STRprofilerR/issues

---

`alleles` *Accessors for STRProfiles objects*

Description

Extract and replace the parts of a STRProfiles object.

Usage

```
alleles(x, ...)

alleles(x, ...) <- value

markers(x, ...)

markerData(x, ...)

markerData(x, ...) <- value

sampleData(x, ...)

sampleData(x, ...) <- value

provenance(x, ...)

## S4 method for signature 'STRProfiles'
alleles(x, ...)

## S4 replacement method for signature 'STRProfiles'
alleles(x, ...) <- value

## S4 method for signature 'STRProfiles'
markers(x, ...)

## S4 method for signature 'STRProfiles'
markerData(x, ...)

## S4 replacement method for signature 'STRProfiles'
markerData(x, ...) <- value

## S4 method for signature 'STRProfiles'
sampleData(x, ...)

## S4 replacement method for signature 'STRProfiles'
```

```
sampleData(x, ...) <- value

## S4 method for signature 'STRProfiles'
provenance(x, ...)
```

Arguments

x	A STRProfiles object.
...	Ignored.
value	Replacement value.

Value

`alleles()` returns a S4Vectors::DataFrame of IRanges::CharacterList columns, one column per marker.

`markers()` returns a character vector of marker names.

`markerData()` and `sampleData()` return a S4Vectors::DataFrame of marker and sample annotations respectively.

`provenance()` returns a list describing how the object was built.

The replacement forms return an updated STRProfiles object.

Author(s)

Jared Andrews

Examples

```
f <- system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR")
p <- readSTRProfiles(f, sampleCol = "Sample Name")

markers(p)
markerData(p)
alleles(p)[["marker1"]]
provenance(p)$timestamp

# Correct a marker classification by hand.
markerData(p)$class[markers(p) == "marker1"] <- "y-linked"
markerData(p)
```

---

amelogeninMarkers *Recognised amelogenin marker names*

Description

Amelogenin is a sex-determining marker rather than a polymorphic STR, so it is excluded from similarity scoring by default. This function returns the spellings recognised as amelogenin.

Usage

```
amelogeninMarkers()
```

Details

The strprofiler Python package identifies amelogenin by matching a single user-supplied column name (amel_col, default "AMEL"), which silently fails when a file uses a different spelling. This package instead classifies every marker at ingest time and records the result in markerData(), so AMEL, Amel, and Amelogenin all work without configuration.

Value

A character vector of recognised amelogenin marker names.

Author(s)

Jared Andrews

See Also

classifyMarkers(), which uses this to tag markers at ingest.

Examples

```
amelogeninMarkers()
```

---

classifyMarkers *Classify STR markers*

Description

Tags each marker as amelogenin, Y-linked, or autosomal. The classification is stored in the class column of markerData() and is what scoreProfiles() uses to decide which markers to drop from scoring.

Usage

```
classifyMarkers(markers)
```

Arguments

markers	Character vector of marker names.

Details

Y-linked markers are recognised by a DYS prefix (for example DYS391). Everything that is neither amelogenin nor Y-linked is reported as "autosomal"; the classification is a naming heuristic, not a lookup against a curated marker panel, so unusual marker names may need correcting by hand via markerData(x)$class <- ....

Value

A character vector the same length as markers, with values "amelogenin", "y-linked", or "autosomal".

Author(s)

Jared Andrews

Examples

```
classifyMarkers(c("AMEL", "DYS391", "vWA", "PentaD"))
```

`clastrBatchQuery` *Query CLASTR in batch and save the XLSX result*

Description

Submits several profiles to CLASTR's batch endpoint and writes the workbook it returns, one sheet per query.

Usage

```
clastrBatchQuery(
  x,
  file,
  algorithm = c("tanabe", "mastersQuery", "mastersRef"),
  scoringMode = c("nonEmpty", "query", "reference"),
  scoreFilter = 80,
  minMarkers = 8,
  maxResults = 200,
  includeAmelogenin = FALSE,
  url = "https://www.cellosaurus.org/str-search/api/batch/"
)
```

Arguments

`x`	A STRProfiles object, or a named character vector for a single profile.
`file`	Path to write the XLSX to.
`algorithm`	Similarity score CLASTR should rank by.
`scoringMode`	How CLASTR handles markers missing from one side: `"nonEmpty"` uses markers present in both, `"query"` all query markers, `"reference"` all reference markers.
`scoreFilter`	Minimum score for a match to be returned.
`minMarkers`	Minimum number of shared markers for a match to be returned.
`maxResults`	Maximum number of matches to return per query.
`includeAmelogenin`	Logical scalar. Include amelogenin in CLASTR's scoring.
`url`	API endpoint. Exposed for testing.

Details

The batch endpoint returns a formatted XLSX rather than JSON, so this is the route to take when you want CLASTR's own presentation of the results. Use `clastrQuery()` instead when you want the hits as data to work with.

This function needs network access, and **is for research use only**.

Value

Invisibly, the path written to.

Author(s)

Jared Andrews

See Also

`clastrQuery()` for parsed results.

Examples

```
p <- readSTRProfiles(
    system.file("extdata", "Example_clastr_input.csv", package = "STRprofilerR")
)

out <- file.path(tempdir(), "clastr.xlsx")
clastrBatchQuery(p, out)
```

---

clastrMarkers *Marker names accepted by the CLASTR API*

Description

The controlled vocabulary of marker names the Cellosaurus CLASTR similarity search accepts, and a helper for checking a profile against it.

Usage

```
clastrMarkers()

validateClastrMarkers(markers)
```

Arguments

`markers`	Character vector of marker names to check.

Details

CLASTR silently ignores markers it does not recognise, so a profile using a local marker spelling can be scored on far fewer markers than intended without any warning. Check first with `validateClastrMarkers()`.

Note that CLASTR wants the *spaced* Penta spellings (`Penta D`), which is the opposite of this package's internal convention; `clastrQuery()` converts them for you via `harmonizeMarkers()` with `reverse = TRUE`.

Value

`clastrMarkers()` returns the accepted marker names.

`validateClastrMarkers()` returns the subset of `markers` that CLASTR does not recognise, as a character vector, empty if all are valid.

Author(s)

Jared Andrews

References

Robin T, Capes-Davis A, Bairoch A (2020). CLASTR: The Cellosaurus STR similarity search tool - A precious help for cell line authentication. *International Journal of Cancer* 146(5):1299-1306. doi:10.1002/ijc.32639

See Also

`clastrQuery()`, which validates automatically before querying.

Examples

```
head(clastrMarkers())

validateClastrMarkers(c("Amelogenin", "vWA", "marker1", "NotAMarker"))
```

`clastrQuery` *Query the Cellosaurus knowledge base via CLASTR*

Description

Searches the human cell line profiles in Cellosaurus for matches to one or more STR profiles, using the CLASTR REST API.

Usage

```
clastrQuery(
  x,
  algorithm = c("tanabe", "mastersQuery", "mastersRef"),
  scoringMode = c("nonEmpty", "query", "reference"),
  scoreFilter = 80,
  minMarkers = 8,
  maxResults = 200,
  includeAmelogenin = FALSE,
  url = "https://www.cellosaurus.org/str-search/api/query/"
)
```

Arguments

`x`	A STRProfiles object, or a named character vector giving one profile's markers and comma-separated alleles.
`algorithm`	Similarity score CLASTR should rank by.
`scoringMode`	How CLASTR handles markers missing from one side: `"nonEmpty"` uses markers present in both, `"query"` all query markers, `"reference"` all reference markers.
`scoreFilter`	Minimum score for a match to be returned.

`minMarkers` Minimum number of shared markers for a match to be returned.

`maxResults` Maximum number of matches to return per query.

`includeAmelogenin`
Logical scalar. Include amelogenin in CLASTR's scoring.

`url` API endpoint. Exposed for testing.

Details

Each profile is submitted separately and the JSON responses are parsed into one tidy table, with a `query` column naming the profile each row answers. A result carrying two profiles (CLASTR returns at most two) is reported as two rows, with `" (Best)"` and `" (Worst)"` appended to the accession, as the Python package does.

Marker names are converted to CLASTR's spellings on the way out and back to this package's on the way in, so `PentaD` round-trips correctly. Markers CLASTR does not recognise are reported with a warning and ignored by the search; check ahead of time with `validateClastrMarkers()`.

This function needs network access, and **is for research use only**.

Divergences from the Python package:

The response is parsed directly from the nested JSON rather than through a flatten-and-pivot chain over generated column names. `algorithm = 3` is sent for Masters (reference) in batch mode, as `strprofiler` does from 0.5.0; through 0.4.2 it sent `2`, which silently runs a Masters (query) search.

Value

A S4Vectors::DataFrame with one row per returned profile and columns `query`, `accession`, `name`, `species`, `score`, `accessionLink`, `problem`, followed by one column per marker. Zero rows if nothing matched.

Author(s)

Jared Andrews

References

Bairoch A (2018). The Cellosaurus, a cell-line knowledge resource. *Journal of Biomolecular Techniques* 29(2):25-38. doi:10.7171/jbt.182902002

Robin T, Capes-Davis A, Bairoch A (2020). CLASTR: The Cellosaurus STR similarity search tool. *International Journal of Cancer* 146(5):1299-1306. doi:10.1002/ijc.32639

See Also

`clastrBatchQuery()` for CLASTR's own multi-sheet XLSX output, `compareProfiles()` to compare against a local database instead.

Examples

```
profile <- c(
    Amelogenin = "X", CSF1PO = "13,14", D5S818 = "13", D7S820 = "8,9",
    D13S317 = "12", FGA = "24", TH01 = "8", TPOX = "11", vWA = "16"
)

hits <- clastrQuery(profile, scoreFilter = 80)
```

```
as.data.frame(hits)[, c("accession", "name", "score")]
```

`cleanAlleles` *Clean and split a vector of allele calls*

Description

Splits comma-separated allele strings into their component alleles, de-duplicating and ordering them consistently.

Usage

```
cleanAlleles(x, keepCalls = c("X", "Y"))
```

Arguments

`x`	Character vector of comma-separated allele calls. `NA` is treated as an empty call.
`keepCalls`	Character vector of non-numeric calls that count as alleles, compared case-insensitively against each whole token and stored in the spelling given here. Exports are not consistent about capitalisation. Pass `character(0)` to accept repeat counts only.

Details

Each element is split on commas and each token trimmed. A token survives only if it is an allele:

- a finite repeat count, de-duplicated numerically, sorted ascending, and rendered without a trailing `.0`, so `"10.0"` and `"10"` collapse to a single `"10"` while `"9.3"` is preserved; or
- one of the calls named in `keepCalls`, matched case-insensitively and stored in the spelling given there, so `"x"` and `"X"` are one allele rather than two that never match. These sort after the numeric alleles.

Everything else is discarded: the empty token left by a trailing comma (`"12,"`), the codes a capillary-electrophoresis export uses for a peak it could not call (`OL` for one outside the ladder, `?` for one it declined to type, `NR`, `ND` and the like), and free text. None of them is an allele, and keeping them makes two profiles that merely failed at the same marker score as sharing a value; discarding them leaves the marker untyped, which lowers the shared-marker count instead. `"nan"` and `"inf"` go the same way, though `as.numeric()` accepts both.

`keepCalls` defaults to the amelogenin sex markers, the only allele calls that are not repeat counts. `readSTRProfiles()` and `STRProfiles()` are stricter still and honour `keepCalls` at amelogenin markers only, since a letter at any other marker is a failed call rather than a sex call. `strprofiler` (from 0.5.0) keeps `X/Y` at every marker; through 0.4.2 it scored every one of these codes as an ordinary allele.

Value

A IRanges::CharacterList the same length as `x`, each element the cleaned allele vector for the corresponding input.

Author(s)

Jared Andrews

Examples

```
cleanAlleles("10.0,10,13,13.0,14,14 ")

# The sex markers are kept, in one spelling, and sort after numeric alleles.
cleanAlleles(c("Y,X", "17.3, 12", "", NA))
cleanAlleles("x,X,y")

# Uncallable peaks and free text are discarded, leaving a marker untyped.
cleanAlleles("OL,11")
cleanAlleles(c("OL", "12,NR,ND"))

# Repeat counts only.
cleanAlleles("12,X", keepCalls = character(0))
```

`compareProfiles` *Compare STR profiles end to end*

Description

Scores profiles, flags potential mixing, and summarises the hits in one call. This is the R equivalent of the Python package's `strprofiler compare` command, minus the file writing, which `writeSTRResults()` handles.

Usage

```
compareProfiles(
  query,
  reference = NULL,
  useAmel = FALSE,
  excludeMarkers = NULL,
  tanThreshold = 80,
  masQThreshold = 80,
  masRThreshold = 80,
  threeAlleleThreshold = 3,
  minScore = NULL,
  minScoreType = c("tanabe", "mastersQuery", "mastersRef"),
  chunkSize = 2000L
)
```

Arguments

`query`	A STRProfiles object holding the profiles to score.
`reference`	A STRProfiles object to score against. If `NULL`, `query` is compared against itself and self-pairs are dropped.
`useAmel`	Logical scalar. Include amelogenin markers in scoring.
`excludeMarkers`	Character vector of additional marker names to leave out of scoring.

tanThreshold	Minimum Tanabe score to report in tanabeMatches.
masQThreshold	Minimum Masters (query) score to report in mastersQueryMatches.
masRThreshold	Minimum Masters (reference) score to report in mastersRefMatches.
threeAlleleThreshold	Passed to flagMixedSamples(). Ignored when profiles is NULL.
minScore	Optional numeric scalar. Drop pairs scoring below this on minScoreType. Applied per block, so it also bounds memory.
minScoreType	Which score minScore applies to.
chunkSize	Number of query profiles to score per block.

Details

With reference supplied, each query is compared against the reference set only, never against the other queries. This is the "batch against database" mode. With reference = NULL, queries are compared against each other and self-comparisons are dropped.

Unlike strprofiler, this returns an object rather than writing files, so results can be inspected, filtered, or plotted before anything is committed to disk. Pass the result to writeSTRResults() to produce the same file set the Python CLI does.

Value

A STRComparison object.

Author(s)

Jared Andrews

See Also

scoreProfiles() and summarizeMatches() for the individual steps, writeSTRResults() to write the output files.

Examples

```
q <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)
ref <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)

cmp <- compareProfiles(q, ref)
cmp
summary(cmp)

# All-against-all within a single set.
compareProfiles(ref)
```

`flagMixedSamples` *Flag potentially mixed samples*

Description

Flags samples showing signs of mixing or contamination, based on how many markers carry more alleles than a single diploid genome can explain.

Usage

```
flagMixedSamples(x, threeAlleleThreshold = 3)
```

Arguments

`x`	A STRProfiles object.
`threeAlleleThreshold`	Number of markers with more than two alleles tolerated before a sample is flagged.

Details

A marker is counted when it has **more than two** alleles. A sample is flagged when the number of such markers is **strictly greater than** `threeAlleleThreshold`. With the default of 3, a sample needs four or more three-allele markers before it is flagged.

This is a screening heuristic, not a test. A flagged sample warrants a look at its electropherogram; an unflagged one is not evidence of purity.

Value

A named logical vector, one element per sample.

Author(s)

Jared Andrews

See Also

`compareProfiles()`, which reports this alongside the scores.

Examples

```
p <- readSTRProfiles(
    system.file("extdata", "Example_Batch_File.csv", package = "STRprofilerR")
)

flagMixedSamples(p)

# A stricter threshold flags fewer samples.
flagMixedSamples(p, threeAlleleThreshold = 5)
```

markerAliases *Marker name harmonisation*

Description

STR marker names are spelled inconsistently across instruments, vendors, and databases. `markerAliases()` returns the alias table used to harmonise them, and `harmonizeMarkers()` applies it to a vector of marker names.

Usage

```
markerAliases(reverse = FALSE, extra = NULL)

harmonizeMarkers(markers, reverse = FALSE, extra = NULL)
```

Arguments

`reverse`	Logical scalar. If `TRUE`, return the inverse table mapping compact spellings onto the spaced spellings expected by CLASTR.
`extra`	Named character vector of additional aliases, where names are the aliases to replace and values are the canonical names to use.
`markers`	Character vector of marker names to harmonise.

Details

The default table resolves the common Penta marker spellings (`Penta C`, `Penta_C`, and so on) onto the compact forms (`PentaC`) used throughout this package. This generalises the hard-coded `_pentafix()` helper in the `strprofiler` Python package: pass `extra` to harmonise any other marker names your instrument produces.

The CLASTR API expects the *spaced* spellings, so `reverse = TRUE` inverts the table for outbound queries.

Value

`markerAliases()` returns a named character vector whose names are aliases and whose values are canonical marker names.

`harmonizeMarkers()` returns a character vector of the same length as `markers`, with any recognised aliases replaced.

Author(s)

Jared Andrews

Examples

```
markerAliases()

harmonizeMarkers(c("Penta D", "Penta_E", "vWA"))

# The CLASTR API wants the spaced spellings back.
harmonizeMarkers(c("PentaD", "PentaE"), reverse = TRUE)
```

```
# Harmonise a vendor-specific spelling too.
harmonizeMarkers("AMELOGENIN", extra = c(AMELOGENIN = "AMEL"))
```

`pdxSTRProfiles` *STR profiles of patient-derived models from JAX and the NCI PDMR*

Description

Reference STR profiles for 1,258 patient-derived models: 435 from The Jackson Laboratory PDX program and 823 from the NCI Patient-Derived Models Repository (PDMR). This is the default database of `STRprofilerApp()`, and can be passed as the `reference` of `compareProfiles()` or `scoreProfiles()` to check models against it.

Usage

```
data(pdxSTRProfiles)
```

Format

A STRProfiles object of 1,258 samples typed at 18 markers: amelogenin and 17 autosomal STRs (the 13 CODIS core loci plus D2S1338, D19S433, Penta D, and Penta E). `sampleData()` holds

`Center` Source of the model, `"JAX"` or `"PDMR"`.

`Passage` Passage of the profiled material, where recorded.

`nDroppedCalls` Calls dropped at ingest as not being alleles (see `readSTRProfiles()`).

Details

Built by `inst/scripts/make-pdxSTRProfiles.R`, which reads `inst/extdata/main_database.csv` with `readSTRProfiles()` defaults. The raw file is kept alongside, as an example of a database file with metadata columns:

```
system.file("extdata", "main_database.csv", package = "STRprofilerR")
```

Twenty-two samples each carry one call that is not an allele (mostly `OL`, off-ladder), dropped at ingest and counted in `nDroppedCalls`.

Source

Profiles provided by The Jackson Laboratory PDX program and the NCI PDMR, as compiled for the default database of the `strprofiler` Python package (0.5.1, `strprofiler/shiny_app/www/main_database.csv`), and redistributed here under the package's MIT license. See `inst/scripts/README.md`.

References

Andrews JM, Lloyd MW, Neuhauser SB, Bundy M, Jocoy EL, Airhart SD, Bult CJ, Evrard YA, Chuang JH, Baker S (2024). STRprofiler: efficient comparisons of short tandem repeat profiles for biomedical model authentication. *Bioinformatics*, btae713. doi:10.1093/bioinformatics/btae713.

Examples

```
data(pdxSTRProfiles)
pdxSTRProfiles
table(sampleData(pdxSTRProfiles)$Center)

# Check a batch of models against it.
batch <- readSTRProfiles(
    system.file("extdata", "Example_Batch_File.csv", package = "STRprofilerR")
)
cmp <- compareProfiles(batch, pdxSTRProfiles)
summary(cmp)[, c("Sample", "mixed", "topHit")]
```

---

readSTRProfiles *Read STR profiles from file*

Description

Reads one or more STR profile files, in either wide or long layout, into a single STRProfiles object.

Usage

```
readSTRProfiles(
  files,
  sampleCol = "Sample",
  markerCol = "Marker",
  sampleMap = NULL,
  pentaFix = TRUE,
  metadataCols = c("Center", "Passage"),
  format = c("auto", "wide", "long"),
  extraAliases = NULL,
  keepCalls = c("X", "Y")
)
```

Arguments

files	Character vector of paths. Supported extensions are csv, tsv, txt (tab-separated), and xlsx (first sheet; needs readxl).
sampleCol	Name of the sample identifier column.
markerCol	Name of the marker identifier column. Used only for wide files.
sampleMap	Optional sample renaming table: either a path to a headerless two-column CSV, or a data.frame/matrix with at least two columns. The first column holds names as they appear in the input, the second the names to use instead.
pentaFix	Logical scalar. Harmonise the common Penta marker spellings via markerAliases().
metadataCols	Character vector of non-marker column names to route into sampleData(). Columns not present in the input are ignored.
format	One of "auto", "wide", or "long". Applied to every file.
extraAliases	Named character vector of additional marker aliases, passed to harmonizeMarkers().

keepCalls — Character vector of non-numeric calls that count as alleles, passed to `cleanAlleles()` and honoured at amelogenin markers only. Defaults to the sex markers `X` and `Y`. Every other non-numeric call is an uncallable peak or free text, and the per-sample count discarded is recorded in `sampleData()` as `nDroppedCalls`.

Details

Layouts:

Long files carry one row per sample, with every column other than the sample column (and any `metadataCols`) treated as a marker:

```
Sample,  D1S1656, DYS391, AMEL
Line1,   "12,14", 12,     X
```

Wide files carry one row per sample/marker pair, with the alleles spread across several columns. Any column whose name contains `Allele` is collected; size and height columns are ignored:

```
Sample, Marker,  Allele 1, Size 1, Allele 2
Line1,  D3S1358, 16,       128.29, 18
```

With `format = "auto"` a file is read as wide when it has a `markerCol` column *and* at least one column containing `Allele`, and long otherwise.

Combining files:

Files may mix layouts and formats. Markers are unioned across files, with samples missing a marker recorded as untyped rather than as an empty allele. Sample names must be unique across all files.

Divergences from the Python package:

`metadataCols` are held in `sampleData()` rather than alongside the markers. `strprofiler` (from 0.5.0) carries them in the profile and skips them at scoring and mixing time, so a custom metadata column has to be declared to each of those functions rather than once at ingest. (Through 0.4.2 it had no such argument, and two samples sharing a `Center` of `"JAX"` scored as sharing a marker.)

Row order follows the input files. `strprofiler` returns wide-format samples in sorted order because it groups with `pandas`.

All columns are read as text, so alleles are never coerced to numbers and back. This removes a class of bug that `strprofiler` patched twice through 0.4.2 (alleles ending in zero being truncated, for example `10` becoming `1`); since 0.5.0 it instead parses every call and renders it back.

Value

A STRProfiles object.

Author(s)

Jared Andrews

See Also

`STRProfiles()` to build one from an in-memory `data.frame`, `scoreProfiles()` to compare the result.

Examples

```
long <- system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR")
p <- readSTRProfiles(long, sampleCol = "Sample Name")
p

# Penta spellings are harmonised by default.
markers(p)
markers(readSTRProfiles(long, sampleCol = "Sample Name", pentaFix = FALSE))

# A database file with Center and Passage metadata.
db <- system.file("extdata", "main_database.csv", package = "STRprofilerR")
ref <- readSTRProfiles(db)
sampleData(ref)[1:3, ]

# Rename samples on the way in.
smap <- system.file("extdata", "SampleMap_exp.csv", package = "STRprofilerR")
xlsx <- system.file("extdata", "ExampleSTR.xlsx", package = "STRprofilerR")
if (requireNamespace("readxl", quietly = TRUE)) {
    rownames(readSTRProfiles(xlsx, sampleCol = "Sample Name", sampleMap = smap))
}
```

`scoreProfiles` *Score STR profiles against each other*

Description

Computes the Tanabe and both Masters similarity scores for every query/reference pair.

Usage

```
scoreProfiles(
  query,
  reference = NULL,
  useAmel = FALSE,
  excludeMarkers = NULL,
  minScore = NULL,
  minScoreType = c("tanabe", "mastersQuery", "mastersRef"),
  chunkSize = 2000L
)
```

Arguments

`query`	A STRProfiles object holding the profiles to score.
`reference`	A STRProfiles object to score against. If `NULL`, `query` is compared against itself and self-pairs are dropped.
`useAmel`	Logical scalar. Include amelogenin markers in scoring.
`excludeMarkers`	Character vector of additional marker names to leave out of scoring.
`minScore`	Optional numeric scalar. Drop pairs scoring below this on `minScoreType`. Applied per block, so it also bounds memory.
`minScoreType`	Which score `minScore` applies to.
`chunkSize`	Number of query profiles to score per block.

Details

The scores:

Let M be the markers at which *both* profiles were typed (markers with no alleles in either profile are ignored), s the number of alleles shared across those markers, and q and r the number of alleles the query and reference carry across those same markers. Then:

$$\text{Tanabe} = 100 \times \frac{2s}{q + r}$$

$$\text{Masters (query)} = 100 \times \frac{s}{q}$$

$$\text{Masters (reference)} = 100 \times \frac{s}{r}$$

Tanabe is the Sorenson-Dice coefficient and is symmetric. The two Masters scores are not: Masters (query) asks how much of the query is explained by the reference, which is the question to ask when the query may be a contaminated or drifted derivative of a known line.

Amelogenin is a sex marker rather than a polymorphic STR, so it is excluded by default; set `useAmel = TRUE` to include it.

Implementation:

Scoring is expressed as four sparse matrix products rather than a loop over pairs, which is what makes whole-database comparison practical. With Q_A/R_A the binary sample-by-(marker, allele) incidence matrices, Q_M/R_M the binary sample-by-marker "typed" matrices, and Q_C/R_C the sample-by-marker allele counts:

```
s <- QA %*% t(RA)    # shared alleles
q <- QC %*% t(RM)    # query alleles,     over shared markers only
r <- QM %*% t(RC)    # reference alleles, over shared markers only
```

An allele can only be shared at a marker where both profiles were typed, so the restriction to shared markers falls out of the products.

Results are computed in blocks of `chunkSize` queries so that comparing a large batch against a large database does not need the full query-by-reference matrix in memory at once.

Divergence from the Python package:

A pair with no markers in common gives a zero denominator. `strprofiler` raises `ZeroDivisionError` (which its Shiny app catches and reports as `FALSE`); this function returns `NA` for the affected scores and warns once with a count.

Value

A S4Vectors::DataFrame with one row per scored pair and columns `query`, `reference`, `nSharedMarkers`, `nSharedAlleles`, `nQueryAlleles`, `nReferenceAlleles`, `tanabeScore`, `mastersQueryScore`, and `mastersRefScore`. Rows are ordered by query (input order), then by Tanabe score descending.

Author(s)

Jared Andrews

References

Tanabe H, et al. (1999). Cell line individualization by STR multiplex system in the cell bank found cross-contamination between ECV304 and EA.hy926. *Tissue Culture Research Communications* 18:329-338. doi:10.11418/jtca1981.18.4_329

Masters JR, et al. (2001). Short tandem repeat profiling provides an international reference standard for human cell lines. *PNAS* 98(14):8012-8017. doi:10.1073/pnas.121616198

See Also

`compareProfiles()` for scoring plus a per-sample summary, `summarizeMatches()` to condense the output.

Examples

```
q <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)
ref <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)

scores <- scoreProfiles(q, ref)
head(as.data.frame(scores))

# All-against-all within one set.
scoreProfiles(ref)[1:3, ]

# Keep only strong Tanabe hits.
scoreProfiles(q, ref, minScore = 90)
```

`scoreQuery` *Score a single profile against a reference set*

Description

A convenience wrapper around `scoreProfiles()` for scoring one hand-entered profile, as an interactive session or a query form would.

Usage

```
scoreQuery(
  x,
  reference,
  sample = "Query",
  useAmel = FALSE,
  excludeMarkers = NULL,
  pentaFix = TRUE,
  keepCalls = c("X", "Y")
)
```

Arguments

x	A named character vector or list mapping marker names to comma-separated allele calls.
reference	A STRProfiles object to score against.
sample	Name to label the query with.
useAmel	Logical scalar. Include amelogenin markers in scoring.
excludeMarkers	Character vector of additional marker names to leave out of scoring.
pentaFix	Logical scalar. Harmonise Penta marker spellings in x.
keepCalls	Character vector of non-numeric calls that count as alleles, passed to cleanAlleles() and honoured at amelogenin markers only. Defaults to the sex markers X and Y. Every other non-numeric call is an uncallable peak or free text, and the per-sample count discarded is recorded in sampleData() as nDroppedCalls.

Value

A S4Vectors::DataFrame as returned by scoreProfiles().

Author(s)

Jared Andrews

See Also

scoreProfiles(), which this calls.

Examples

```
ref <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)

scoreQuery(
    c(marker1 = "12,14", marker2 = "12", marker4 = "13", AMEL = "X"),
    ref
)
```

---

scores *Accessors for STRComparison objects*

Description

Extract the parts of the STRComparison object returned by compareProfiles().

Usage

```
scores(x, ...)

params(x, ...)

queryProfiles(x, ...)

referenceProfiles(x, ...)

## S4 method for signature 'STRComparison'
scores(x, ...)

## S4 method for signature 'STRComparison'
params(x, ...)

## S4 method for signature 'STRComparison'
queryProfiles(x, ...)

## S4 method for signature 'STRComparison'
referenceProfiles(x, ...)

## S4 method for signature 'STRComparison'
summary(object, ...)

## S4 method for signature 'STRComparison'
show(object)
```

Arguments

x, object	A STRComparison object.
...	Ignored.

Value

`scores()` returns the pair-level S4Vectors::DataFrame, `summary()` the per-query one, `queryProfiles()` and `referenceProfiles()` the STRProfiles that were compared, and `params()` a list of the scoring and threshold settings used.

Author(s)

Jared Andrews

Examples

```
q <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)
ref <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)
cmp <- compareProfiles(q, ref)
```

```
cmp
summary(cmp)
head(as.data.frame(scores(cmp)))
params(cmp)$tanThreshold
```

STRComparison-class *STRComparison: the result of comparing STR profiles*

Description

An S4 container returned by `compareProfiles()`, holding every pairwise score alongside a per-query summary and the parameters used to produce them.

Value

An object of class `STRComparison`.

Slots

`scores` A S4Vectors::DataFrame with one row per query/reference pair.

`summary` A S4Vectors::DataFrame with one row per query sample.

`query` The query STRProfiles.

`reference` The reference STRProfiles. Identical to `query` for an all-against-all comparison.

`params` A list of the scoring and threshold parameters used.

Author(s)

Jared Andrews

See Also

`compareProfiles()` to build one, `writeSTRResults()` to write it out.

Examples

```
f <- system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR")
db <- system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR")
cmp <- compareProfiles(
    readSTRProfiles(f, sampleCol = "Sample Name"),
    readSTRProfiles(db, sampleCol = "Sample Name")
)
cmp
```

`strHTMLTable` *Write a data frame as a standalone HTML table*

Description

Writes a table to a self-contained HTML file, interactive where the optional packages allow and static otherwise.

Usage

```
strHTMLTable(x, file, title = "STRprofiler Results")
```

Arguments

`x`	A `data.frame`.
`file`	Path to write to.
`title`	Page title.

Details

When `DT`, `htmlwidgets`, and `pandoc` are all available the table is written as a `DT` widget: sortable, searchable, and paged. Otherwise a plain HTML table with embedded styling is written. Both forms are self-contained and render offline.

Value

Invisibly, the path written to.

Author(s)

Jared Andrews

See Also

`writeSTRResults()`, which calls this for the summary table.

Examples

```
f <- file.path(tempdir(), "table.html")
strHTMLTable(head(iris), f)
file.exists(f)
```

`STRprofilerApp` *Launch the STRprofiler Shiny application*

Description

Builds the interactive STRprofiler application, for comparing STR profiles against a reference database or Cellosaurus from a web browser without writing any code.

Usage

```
STRprofilerApp(database = NULL, sampleCol = "Sample", markerCol = "Marker")
```

Arguments

`database`	The reference database. `NULL`, the default, uses pdxSTRProfiles, the database bundled with the package, of models from The Jackson Laboratory PDX program and the NCI Patient-Derived Models Repository. Otherwise a path to a file `readSTRProfiles()` can read, or a STRProfiles object.
`sampleCol`	Name of the sample identifier column, used when reading `database` from a file and for every file uploaded in the application.
`markerCol`	Name of the marker column in wide-layout files, used as `sampleCol` is.

Details

The application is a port of the one in the `strprofiler` Python package, aligned with its 0.5.1 release. It has four tabs:

Single Query Type a profile in marker by marker and score it against the loaded database, or search Cellosaurus with it through CLASTR. The report lists every reference passing the chosen score threshold, best first, with alleles that differ from the query highlighted.

Batch Query Upload a file of profiles and compare them against the database, against each other ("Within File Query"), or against Cellosaurus. Database and within-file searches give one summary row per profile, as `compareProfiles()` does.

Database File Management Swap in a custom reference database for the session, or restore the one the application started with.

Usage Guide How to use the application, and what to cite.

Uploaded files are read with `readSTRProfiles()`, so they may be csv, tsv, tab-separated txt, or xlsx, in the wide or long layout.

CLASTR searches need network access. As with the rest of this package, the application **is for research use only**.

Deploying:

The return value is an ordinary Shiny application object. To host it with your own database, on Posit Connect or a Shiny Server for example, deploy an `app.R` containing:

```
STRprofilerR::STRprofilerApp(database = "my_database.csv")
```

A ready-made one ships with the package, at `system.file("app", "app.R", package = "STRprofilerR")`. It also attaches the application's suggested packages so deployment tools bundle them.

From a shell, `strprofilerr app --database my_database.csv` runs the same application locally.

Divergences from the Python package:

Checked against the application in `strprofiler` 0.5.1:

- An upload that fails to load leaves the current database in place. `strprofiler` loads its stock database instead, while still displaying the failed file's name.
- Changing the batch search type clears the results. `strprofiler` re-runs the query, which for CLASTR means an unprompted API call.
- Batch CLASTR results are read from CLASTR's JSON API, shown per profile, and downloadable as csv, with CLASTR's own xlsx workbook still offered. `strprofiler` reads the workbook back, keying sheets by sample name, which fails for names longer than Excel's 31-character sheet name limit.
- Uploads may be csv, tsv, txt, or xlsx. `strprofiler`'s upload controls accept csv only.
- Single query results show whichever metadata columns the database has. `strprofiler` always shows `Center` and `Passage`, empty if the database lacks them.
- Amelogenin is recognised under any of the spellings in `amelogeninMarkers()`. `strprofiler`'s application only recognises a marker named `Amelogenin`.

Value

A `shiny.appobj`. Print it, or pass it to `shiny::runApp()`, to launch the application.

Author(s)

Jared Andrews

References

Andrews JM, Lloyd MW, Neuhauser SB, Bundy M, Jocoy EL, Airhart SD, Bult CJ, Evrard YA, Chuang JH, Baker S (2024). STRprofiler: efficient comparisons of short tandem repeat profiles for biomedical model authentication. *Bioinformatics*, btae713. doi:10.1093/bioinformatics/btae713.

See Also

`compareProfiles()` and `clastrQuery()`, which the application calls.

Examples

```
if (requireNamespace("shiny", quietly = TRUE) &&
    requireNamespace("bslib", quietly = TRUE) &&
    requireNamespace("DT", quietly = TRUE)) {
    app <- STRprofilerApp()

    # With a custom database.
  custom <- system.file("extdata", "Example_Custom_Database.csv", package = "STRprofilerR")
    app <- STRprofilerApp(database = custom)

    if (interactive()) {
        shiny::runApp(app)
    }
}
```

`STRProfiles` *Build STRProfiles from a data.frame*

Description

Constructs a STRProfiles object from a wide `data.frame` already in memory: one row per sample, one column per marker, alleles as comma-separated strings. This is the inverse of `as.data.frame()` on a `STRProfiles` object.

Usage

```
STRProfiles(
  x,
  sampleCol = "Sample",
  pentaFix = TRUE,
  metadataCols = c("Center", "Passage"),
  extraAliases = NULL,
  keepCalls = c("X", "Y")
)
```

Arguments

`x`	A `data.frame` with a sample identifier column and one column per marker.
`sampleCol`	Name of the sample identifier column.
`pentaFix`	Logical scalar. Harmonise the common Penta marker spellings via `markerAliases()`.
`metadataCols`	Character vector of non-marker column names to route into `sampleData()`. Columns not present in the input are ignored.
`extraAliases`	Named character vector of additional marker aliases, passed to `harmonizeMarkers()`.
`keepCalls`	Character vector of non-numeric calls that count as alleles, passed to `cleanAlleles()` and honoured at amelogenin markers only. Defaults to the sex markers `X` and `Y`. Every other non-numeric call is an uncallable peak or free text, and the per-sample count discarded is recorded in `sampleData()` as `nDroppedCalls`.

Value

A STRProfiles object.

Author(s)

Jared Andrews

See Also

`readSTRProfiles()` to read from disk instead.

Examples

```
df <- data.frame(
    Sample = c("Line1", "Line2"),
    AMEL = c("X,Y", "X"),
    vWA = c("16,18", "17"),
    "Penta D" = c("9,10", ""),
    check.names = FALSE
)
p <- STRProfiles(df)
p
alleles(p)[["PentaD"]]

# Round-trips through as.data.frame().
identical(as.data.frame(STRProfiles(as.data.frame(p))), as.data.frame(p))
```

STRProfiles-class *STRProfiles: a set of short tandem repeat profiles*

Description

An S4 container holding STR profiles for a set of samples, the markers they were typed at, per-sample metadata, and a record of how the object was built.

Details

Alleles are stored **pre-split**: each column of the `alleles` slot is an IRanges::CharacterList whose i-th element is the character vector of unique alleles called for sample i at that marker. A sample that was not typed at a marker has a zero-length entry.

This is the main structural departure from the `strprofiler` Python package, which stores comma-joined strings and re-splits them on every single pairwise comparison. Splitting once at ingest is what lets `scoreProfiles()` express scoring as a handful of sparse matrix products.

Build one with `readSTRProfiles()` (from files) or `STRProfiles()` (from a `data.frame` already in memory).

Value

An object of class `STRProfiles`.

Slots

`alleles` A S4Vectors::DataFrame with one row per sample and one column per marker. Every column is a IRanges::CharacterList.

`markerData` A S4Vectors::DataFrame with one row per marker, row names matching `colnames(alleles)`. Always carries a `class` column (see `classifyMarkers()`) and an `nTyped` column.

`sampleData` A S4Vectors::DataFrame with one row per sample, row names matching `rownames(alleles)`. Holds non-marker columns such as `Center` and `Passage`.

`provenance` A list recording the source files, ingest options, timestamp, and package version.

Author(s)

Jared Andrews

See Also

`readSTRProfiles()` to build one, `scoreProfiles()` to compare them.

Examples

```
f <- system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR")
p <- readSTRProfiles(f, sampleCol = "Sample Name")
p
```

STRProfiles-methods *Dimensions, subsetting, and coercion for STRProfiles*

Description

Standard R idioms for inspecting and reshaping a STRProfiles object.

Usage

```
## S4 method for signature 'STRProfiles'
dim(x)

## S4 method for signature 'STRProfiles'
dimnames(x)

## S4 method for signature 'STRProfiles,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'STRProfiles'
x$name

## S4 method for signature 'STRProfiles'
show(object)

## S4 method for signature 'STRProfiles'
c(x, ..., recursive = FALSE)

## S4 method for signature 'STRProfiles'
as.data.frame(x, row.names = NULL, optional = FALSE, ..., sampleCol = "Sample")
```

Arguments

x	A STRProfiles object.
i	Sample index: names, positions, or a logical vector.
j	Marker index: names, positions, or a logical vector.
drop	Ignored; present for compatibility with the [generic.

name	A marker or sample annotation name.
object	A STRProfiles object.
recursive	Ignored; present for compatibility with the c() generic.
row.names, optional, ...	Passed through to as.data.frame().
sampleCol	Name to give the sample identifier column.

Details

A STRProfiles object is samples-by-markers, so dim() reports c(n_samples, n_markers) and x[i, j] subsets samples by i and markers by j, both of which accept names, indices, or logical vectors.

as.data.frame() flattens the object back to the wide layout readSTRProfiles() accepts, with alleles collapsed to comma-separated strings, so profiles round-trip through disk without loss. Annotations the package computes at ingest rather than reading from the input — currently nDroppedCalls — are left out, since every column emitted is read back as a marker and a QC statistic is not one.

Value

dim() returns an integer vector of length two; dimnames() a list of two character vectors. [and c() return a STRProfiles object. as.data.frame() returns a data.frame. $ returns the named column of sampleData().

Author(s)

Jared Andrews

Examples

```
f <- system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR")
p <- readSTRProfiles(f, sampleCol = "Sample Name")

dim(p)
colnames(p)

# Subset to three samples and drop amelogenin.
sub <- p[1:3, markerData(p)$class != "amelogenin"]
dim(sub)

as.data.frame(sub)
```

---

summarizeMatches *Summarise pairwise scores into one row per query*

Description

Condenses the pair-level output of scoreProfiles() into a per-query overview: the best hits, every hit passing each score threshold, and an optional mixing flag and copy of the query's alleles.

Usage

```
summarizeMatches(
  scores,
  profiles = NULL,
  tanThreshold = 80,
  masQThreshold = 80,
  masRThreshold = 80,
  threeAlleleThreshold = 3,
  includeAlleles = TRUE
)
```

Arguments

scores	A S4Vectors::DataFrame of pairwise scores from scoreProfiles().
profiles	Optional STRProfiles of the query profiles. When supplied, a mixed column and the query's allele columns are added.
tanThreshold	Minimum Tanabe score to report in tanabeMatches.
masQThreshold	Minimum Masters (query) score to report in mastersQueryMatches.
masRThreshold	Minimum Masters (reference) score to report in mastersRefMatches.
threeAlleleThreshold	Passed to flagMixedSamples(). Ignored when profiles is NULL.
includeAlleles	Logical scalar. Append the query's allele columns. Ignored when profiles is NULL.

Details

Hits are reported as "name: score", joined by "; ", with scores shown to two decimal places. Each of the three match columns is ordered by its own score, descending. topHit and nextBest use the Tanabe score.

If scores was filtered (via the minScore argument of scoreProfiles()), the summary reflects only the pairs that survived.

Divergences from the Python package:

strprofiler reads the top two hits positionally out of a table whose first row is the query itself, reporting an empty string when a query has fewer than two comparisons (through 0.4.2 it raised an IndexError). Here a query with no comparisons gets NA for both, and one with a single comparison gets NA for nextBest.

strprofiler orders all three match columns by Tanabe score, so the Masters columns come out in an order unrelated to their own values. Each column is sorted on its own score here.

Value

A S4Vectors::DataFrame with one row per query sample and columns Sample, mixed (optional), topHit, nextBest, tanabeMatches, mastersQueryMatches, mastersRefMatches, followed by the allele columns.

Author(s)

Jared Andrews

See Also

scoreProfiles() to produce scores, compareProfiles() to do both in one call.

Examples

```
q <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)
ref <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)

scores <- scoreProfiles(q, ref)
summarizeMatches(scores, q)

# Scores only, without the mixing flag or alleles.
summarizeMatches(scores)
```

---

writeSTRProfiles *Write STR profiles to file*

Description

Writes a STRProfiles object to a CSV in the wide layout that readSTRProfiles() reads back, with alleles collapsed to comma-separated strings.

Usage

```
writeSTRProfiles(x, file, sampleCol = "Sample")
```

Arguments

x	A STRProfiles object.
file	Path to write to.
sampleCol	Name to give the sample identifier column.

Value

Invisibly, the path written to.

Author(s)

Jared Andrews

Examples

```
db <- system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR")
p <- readSTRProfiles(db, sampleCol = "Sample Name")

out <- file.path(tempdir(), "profiles.csv")
writeSTRProfiles(p, out)
head(read.csv(out, check.names = FALSE))
```

---

writeSTRResults *Write comparison results to disk*

Description

Writes a STRComparison out as the file set the `strprofiler` command line tool produces: a summary table, one table per query sample, an HTML view of the summary, and a log recording the parameters used.

Usage

```
writeSTRResults(
  x,
  dir,
  formats = c("csv", "html"),
  perSample = TRUE,
  timestamp = Sys.time(),
  prefix = "STRprofilerR"
)
```

Arguments

`x`	A STRComparison object.
`dir`	Directory to write into. Created if it does not exist.
`formats`	Which outputs to write. Any of `"csv"`, `"html"`, and `"xlsx"`. The `xlsx` summary needs the `writexl` package.
`perSample`	Logical scalar. Write the per-query comparison tables. Ignored unless `"csv"` is in `formats`.
`timestamp`	A `POSIXct` used to stamp the file names.
`prefix`	File name prefix.

Details

Files are named with an `STRprofilerR` prefix where the Python package uses `strprofiler`, so results from the two are distinguishable.

Written into `dir`, with `<stamp>` a `YYYYMMDD.HH_MM_SS` timestamp:

`full_summary.STRprofilerR.<stamp>.csv` One row per query: the mixing flag, top two hits, and every hit passing each threshold.

`<sample>.STRprofilerR.<stamp>.csv` One file per query, listing every reference it was compared against with the scores and that reference's alleles. The query itself is the first row.

`full_summary.STRprofilerR.<stamp>.html` The summary as a browsable table.

`STRprofilerR.<stamp>.log` Parameters, input files, and package and R versions.

Sample names are sanitised for use as file names, so a query called `HT-29/P3` is written as `HT-29_P3.STRprofilerR.<st`. The name inside the file is untouched.

The HTML is written with `DT` when `DT`, `htmlwidgets`, and `pandoc` are all available, giving a sortable, searchable, self-contained table. Otherwise a plain styled HTML table is written instead. Either way the file stands alone, unlike the Python package's output, which pulls jQuery and DataTables from a CDN at view time.

Value

Invisibly, a named character vector of the paths written.

Author(s)

Jared Andrews

See Also

`compareProfiles()` to produce `x`.

Examples

```
q <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_long.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)
ref <- readSTRProfiles(
    system.file("extdata", "ExampleSTR_database.csv", package = "STRprofilerR"),
    sampleCol = "Sample Name"
)

out <- file.path(tempdir(), "STRprofilerR-demo")
written <- writeSTRResults(compareProfiles(q, ref), out)
basename(written)
```

Index

∗ **datasets**
pdxSTRProfiles, 15
∗ **internal**
STRprofilerR-package, 2
[,STRProfiles,ANY,ANY,ANY-method *(STRProfiles-methods)*, 29
[,STRProfiles-method *(STRProfiles-methods)*, 29
$,STRProfiles-method *(STRProfiles-methods)*, 29

alleles, 3
alleles,STRProfiles-method *(alleles)*, 3
alleles<- *(alleles)*, 3
alleles<-,STRProfiles-method *(alleles)*, 3
amelogeninMarkers, 4
amelogeninMarkers(), *26*
as.data.frame,STRProfiles-method *(STRProfiles-methods)*, 29

c,STRProfiles-method *(STRProfiles-methods)*, 29
classifyMarkers, 5
classifyMarkers(), *5*, *28*
clastrBatchQuery, 6
clastrBatchQuery(), *9*
clastrMarkers, 7
clastrQuery, 8
clastrQuery(), *6–8*, *26*
cleanAlleles, 10
cleanAlleles(), *17*, *21*, *27*
compareProfiles, 11
compareProfiles(), *9*, *13*, *15*, *20*, *21*, *23*, *25*, *26*, *32*, *34*

dim,STRProfiles-method *(STRProfiles-methods)*, 29
dimnames,STRProfiles-method *(STRProfiles-methods)*, 29

flagMixedSamples, 13
flagMixedSamples(), *12*, *31*

harmonizeMarkers *(markerAliases)*, 14
harmonizeMarkers(), *7*, *16*, *27*

IRanges::CharacterList, *4*, *10*, *28*

markerAliases, 14
markerAliases(), *16*, *27*
markerData *(alleles)*, 3
markerData(), *5*
markerData,STRProfiles-method *(alleles)*, 3
markerData<- *(alleles)*, 3
markerData<-,STRProfiles-method *(alleles)*, 3
markers *(alleles)*, 3
markers,STRProfiles-method *(alleles)*, 3

params *(scores)*, 21
params,STRComparison-method *(scores)*, 21
pdxSTRProfiles, 15, *25*
provenance *(alleles)*, 3
provenance,STRProfiles-method *(alleles)*, 3

queryProfiles *(scores)*, 21
queryProfiles,STRComparison-method *(scores)*, 21

readSTRProfiles, 16
readSTRProfiles(), *10*, *15*, *25*, *27–30*, *32*
referenceProfiles *(scores)*, 21
referenceProfiles,STRComparison-method *(scores)*, 21

S4Vectors::DataFrame, *4*, *9*, *19*, *21–23*, *28*, *31*
sampleData *(alleles)*, 3
sampleData(), *15–17*, *21*, *27*, *30*
sampleData,STRProfiles-method *(alleles)*, 3
sampleData<- *(alleles)*, 3
sampleData<-,STRProfiles-method *(alleles)*, 3
scoreProfiles, 18
scoreProfiles(), *5*, *12*, *15*, *17*, *20*, *21*, *28–32*
scoreQuery, 20

scores, 21
scores,STRComparison-method *(scores)*, 21
shiny::runApp(), *26*
show,STRComparison-method *(scores)*, 21
show,STRProfiles-method
(STRProfiles-methods), 29
STRComparison, *12*, *21*, *22*, *33*
STRComparison *(STRComparison-class)*, 23
STRComparison-accessors *(scores)*, 21
STRComparison-class, 23
strHTMLTable, 24
STRprofilerApp, 25
STRprofilerApp(), *15*
STRprofilerR *(STRprofilerR-package)*, 2
STRprofilerR-package, 2
STRProfiles, *3*, *4*, *6*, *8*, *11*, *13*, *15–18*, *21–23*, *25*, *27*, 27, *29–32*
STRProfiles(), *10*, *17*, *28*
STRProfiles-accessors *(alleles)*, 3
STRProfiles-class, 28
STRProfiles-methods, 29
summarizeMatches, 30
summarizeMatches(), *12*, *20*
summary,STRComparison-method *(scores)*, 21

validateClastrMarkers *(clastrMarkers)*, 7
validateClastrMarkers(), *9*

writeSTRProfiles, 32
writeSTRResults, 33
writeSTRResults(), *11*, *12*, *23*, *24*