Package 'wunderscraper'

January 13, 2018

Title Scrape the 'Wunderground' API

Version 0.1.0

Description A function for randomly sampling from the 'Wunderground' API <http://api.wunderground.com>. Provides multistage sampling strategies, respects API usage limits, and collects and saves 'Wunderground' data.

Depends R (>= 3.4.0)

Imports httr, jsonlite, sf, tigris

Suggests testthat

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Chandler Armstrong [aut, cre]

Maintainer Chandler Armstrong <omni.armstrong@gmail.com>

Repository CRAN

Date/Publication 2018-01-13 17:31:21 UTC

R topics documented:

plan *Plan a schedule for executing a task*

Description

Sets a schedule using strptime

Usage

```
plan(scheduler, ...)

## S3 method for class 'scheduler'
plan(scheduler, ...)
```

Arguments

scheduler	A scheduler object.
...	Arguments passed to seq.POSIXt.

Value

The schedule

Methods (by class)

- scheduler: convenience wrapper around seq.POSIXt.

See Also

strptime, seq.POSIXt

Examples

```
plan(scheduler(), '1 hour') # sample every hour
plan(scheduler(), '30 min') # sample every 30 minutes
```

`scheduler` *Schedules wunderscraper*

Description

Schedule sampling and ensure wunderscraper remains within API usage limits.

Usage

```
scheduler(plan = "developer", day = NA, minute = NA)
```

Arguments

`plan`	API usage plan. Possible values are "developer" (500 calls a day 10 calls a minute, "drizzle" (5000 calls a day 100 calls a minute) "shower" (100000 calls a day 1000 a minute), or "custom" (see parameters `day` and `minute`).
`day`	Custom daily API usage limit.
`minute`	Custom minute API usage limit.

Details

Scheduler is a constructor function that returns a scheduler object for use in a wunderscrape process.

Scheduler has methods for managing the schedule: `plan`, and `sync`.

Value

Returns a scheduler object.

See Also

`plan.scheduler`, `sync.scheduler`

Examples

```
scheduler(plan='drizzle')
```

scrape *Scrape Wunderground API*

Description

Randomly samples from Wunderground API with a sampling strategy based on states, counties, zip codes, or a grid.

Usage

```
scrape(scheduler, id, size = NA, strata = NA, weight = NA,
  cellsize = NA, sampleFrame = wunderscraper::zctaRel, form = "json",
  o = NA)
```

Arguments

scheduler	A scheduler object.
id	A vector of strings specifying variable names for cluster identifiers. The id of the last stage must be "id". If "id" is missing but 'scrape' can unambiguously assume the last stage is "id" then it will do so with a warning, otherwise 'scrape' will raise an error message. The unit identifiers of the second-to-last stage will also supply the 'q' parameters for Wunderground geolookups. The 'q' parameter must be a zip code, city name, or latitude/longitude. Zip codes must have 5 digits. City names must be strings with underscores for spaces. Latitude/longitude must be a string of two floating point numbers separated by a comma. Data that does not meet these requirements may find no results from the Wunderground API, or may cause an error.
size	A vector of integers specifying sample size at each stage. NA values specify complete sampling. If not specified for all stages then unspecified stages are assumed complete sampling.
strata	A vector of strings specifying variable names for strata. NA values indicate simple sampling. Wunderscraper will repeat sampling in each strata.
weight	A vector of strings specifying variable names for numeric variables that indicate sampling weights. NA values specify unweighted sampling.
cellsize	A vector of numerics specifying cellsize for adding grids to TIGER county geometries; grids larger than the scale of a county should be added directly to the sampleFrame. TIGER geometries are in the unit of latitude-longitude degrees. value of NA specifies no grid. The grids will be available to the next stage with the identifying variable GRID.
sampleFrame	A dataframe representing the sampling frame. The dataframe must contain columns named "STATEFP" and "GEOID", along with columns for any data required by the sampling strategy. Defaults to zctaRel.
form	A character string specifying output format. An NA value sends output to standard out and will always be in JSON format. Possible formats are: "json"; any other value will currently result in an error with the message "not implemented".

o — A character string specifying output directory or file. If `form='json'` then this will be a directory with each station, other formats are not yet supported.

Details

The sampling strategy has two constraints: 1) the next to last stage of the strategy must be: zip code, city name, or latitude/longitude, and 2) the last stage must sample individual weather stations. Wunderscraper sends the values of the next to last stage identifier as queries to the Wunderground API. In addition to stages users may specify weights or strata, and may also generate spatial grids to use as stages or strata.

Users specify a sampling strategy through a set of vector-valued arguments that indicate the sampling stages, sizes, strata, and weights. All sampling parameter vectors are in stage order, from the first to the last, and must be fully nested.

Wunderscraper is limited to the following stage and strata identifiers: states, counties, and arbitrary spatial grids; indicated in sampling parameter vectors as `"STATEFP"`, `"GEOID"`, and `"GRID"` respectively.

Wunderscraper may use population or land area as a weighting variable. County population and state population are `"COPOP"` and `"STPOP"` respectively. Similarly, county and state area are `"COAREA"` and `"STAREA"`, respectively, where `"COLAND"` and `"STLAND"` are land areas without water. See `zctaRel` for more details on available weighting variables.

The sampling parameter vectors will be padded on the right with NA values to the length of the longest parameter vector. NA for all sampling parameters results in a complete unweighted unstratified sample for that stage.

Wunderscraper uses the following template for building api queries: `http://api.wunderground.com/conditions/q/<que` wunderscrape returns the value of each query, and can either write the raw json to a file or collect the sample and save all stations as a dataframe in rds format.

Value

Wunderscrape may output the data directly to a file or to standard out. All file output is named by the station identification code and date in epoch time.

See Also

`conditions`

Examples

```
## Not run:
## ?setApiKey before running examples
schedulerMMDD <- scheduler()
## select random county and sample one station from each 0.01 arc degrees
## (roughly 1km^2 at the equator)
scrape(schedulerMMDD, c("GEOID", "ZCTA5"), size=c(1, NA, 1), strata=c(NA, NA, "GRID"),
       weight="COPOP", cellsize=c(NA, 0.01))
## same, but limit sampling to southeastern US
data(zctaRel)
SE <- c("01", "05", "12", "13", "21", "22", "24", "28", "37", "45", "47", "51", "54")
scrape(schedulerMMDD, c("GEOID", "ZCTA5"), size=c(1, NA, 1), strata=c(NA, NA, "GRID"),
```

```
     weight="COPOP", cellsize=c(NA, 0.01), sampleFrame=zctaRel[zctaRel $STATEFP %in% SE, ])
## select two states and in each state select a 1 arc degree area (roughly
## 100km^2 at the equator) and sample five zip codes, each stratified into
## 0.01 arc degree areas
scrape(schedulerMMDD, c("STATEFP", "GRID", "ZCTA5"), size=c(2, 1, 5, 1),
       strata=c(NA, "STATEFP", "GRID", "GRID"), cellsize=c(1, NA, 0.01))
## periodically resample one location
sampleFrame <- with(zctaRel, zctaRel[GEOID==sample(GEOID, 1, weight=COPOP), ])
plan(schedulerMMDD, '2 hours')
repeat {
  scrape(schedulerMMDD, "ZCTA5", strata=c(NA, "GRID"), cellsize=0.01, sampleFrame=sampleFrame)
  sync(schedulerMMDD) # sync schedule after each sample to wait for next scheduled sample
}
## stratify by rural and urban to ensure both types of areas recieve adequate representation
zctaRel $RURAL <- log(zctaRel $COPOP) < 10
scrape(schedulerMMDD, c("GEOID", "ZCTA5"), size=c(1, 8, 1), strata=c("RURAL", "RURAL", "GRID"),
       weight="COPOP", cellsize=c(NA, 0.01), sampleFrame=zctaRel)

## End(Not run)
```

`setApiKey` *Set API key.*

Description

Sets the API key to the value of the argument.

Usage

```
setApiKey(key, f = NULL)
```

Arguments

`key`	A valid Wunderground API key
`f`	A string indicating a text file containing only an API key

Details

If `f` is not missing, then reads the API key from a file, else sets key to the `key` parameter. Does not check if key is valid. Please visit <https://www.wunderground.com/weather/api/d/pricing.html> to sign up for a free API key.

Value

The API key

Examples

```
setApiKey('q1w2e3r4t5y6u7i8') # not a valid key
```

sync *Sync scheduler's schedule to current time*

Description

Syncs scheduler's schedule to current time

Usage

```
sync(scheduler)

## S3 method for class 'scheduler'
sync(scheduler)
```

Arguments

scheduler	A scheduler object.

Value

The synced schedule

Methods (by class)

- scheduler: Sync scheduler's schedule to current time.

See Also

Sys.time

Examples

```
sync(scheduler())
```

wunderscraper *wunderscraper: A package for sampling weather stations via Wunderground*

Description

wunderscraper provides a function for scraping Wunderground's api and objects and methods to schedule scraping and ensure wunderscraper remains within api usage limits.

Wunderscraper functions

scrape

Wunderscraper objects

scheduler

Wunderscraper methods

plan.scheduler sync.scheduler

Wunderscraper data

zctaRel

---

zctaRel *Relationships for zip codes*

Description

A dataset containing population and land area of zip codes and administrative boundaries

Usage

```
zctaRel
```

Format

A data frame with 44410 rows and 8 variables:

ZCTA5 zip code

COUNTYFP 3 character string indicating county Federal Information Processing Standard (FIPS) code

STATEFP 2 character string indicating state FIPS code

GEOID 5 character string concatenating STATEFP and COUNTYFP

STPOP state population

COPOP county population

ZPOP zip code population

POPPT partial zip code population, one value for each county intersecting zip code

Source

https://www2.census.gov/geo/docs/maps-data/data/rel/zcta_county_rel_10.txt

Examples

```
data(zctaRel)
summary(zctaRel)
```

Index