Package: bcaquiferdata (via r-universe)

August 18, 2024

Title BC Aquifer data tools

Version 0.0.3

Description Set of tools for processing BC Aquifer lithology and yield data.

License Apache License (>= 2)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports bcmaps (>= 1.2.0), dplyr (>= 1.0.7), httr (>= 1.4.2), janitor (>= 2.1.0), lubridate (>= 1.9.2), magrittr, purrr (>= 0.3.4), rappdirs (>= 0.3.3), readr (>= 2.1.1), rlang (>= 0.4.12), shiny (>= 1.7.2), shinyFiles (>= 0.9.3), bslib (>= 0.5.0), sf (>= 0.9.8), stars (>= 0.5.3), stringdist (>= 0.9.8), stringr (>= 1.4.0), tidyr (>= 1.1.4), fs (>= 1.6.1), DT (>= 0.23), ggplot2 (>= 3.3.5), ggthemes (>= 4.2.4), shinyjs (>= 2.1.0), htmltools (>= 0.5.7), markdown, rmarkdown, wk (>= 0.9.1)

Depends R (>= 4.0)

Suggests knitr, gt, testthat (>= 3.0.0)

LazyData true

Config/testthat/edition 3

URL `http://bcgov.github.io/bcaquiferdata/`, `https://github.com/bcgov/bcaquiferdata`

VignetteBuilder knitr

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

RemoteUrl https://github.com/bcgov/bcaquiferdata

RemoteRef HEAD

RemoteSha e3b3a443606eb45f4ef0ce027e3b285215b7de6d

Contents

`aq_app` *Launch Aquifer Data Shiny App*

Description

This app allows you to load a shapefile and filter aquifer/well data according to region, explore data, and export cleaned files.

Usage

```
aq_app()
```

Examples

```
aq_app()
```

`cache_clean` *Clean cache*

Description

Removes data cache

Usage

```
cache_clean(bcmaps_cded = FALSE)
```

Arguments

`bcmaps_cded`	Logical. Whether or not to also remove CDED files cached with the bcmaps package. These are used by bcaquifertools for acquiring TRIM data, but may also be cached for use by other workflows.

Examples

```
# cache_clean()
# cache_clean(bcmaps_cded = TRUE)
```

`data_read` *Download, Update, and/or load data*

Description

This function downloads, updates or loads locally stored data. Currently this function returns `wells`, `wells_sf`, or `lithology` data. Note that these data are originally from GWELLS, but are cleaned and summarized for use in the bcaquiferdata package. For example `wells_sf` is a spatial version of the data, and `lithology` is a cleaned and standardized version of lithology. `wells` also contains the new standardized `lithology` data, along with the original lithology observations and intermediate classification steps to simplify error tracing.

Usage

```
data_read(type, update = FALSE, permission = FALSE)
```

Arguments

`type`	Character. Type of data to return, one of `wells`, `wells_sf`, or `lithology`
`update`	Logical. Force update of the data?
`permission`	Logical. Permission to create the cache folder. If `FALSE`, user is asked for permission, if `TRUE`, permission is implied.

Details

Under normal circumstances, users will not need to use this function as it is used internally by the main workflow functions. However, users may wish to overview entire datasets.

Bear in mind that the lithology cleaning and standardizing, while better than the original data, will almost certainly still have errors!

Value

Data frame or spatial features object of the requested data.

Examples

```
wells <- data_read("wells")
```

`data_update` *Update cached data*

Description

Update the GWELLs data stored locally.

Usage

```
data_update(type = "all", download = TRUE, permission = FALSE)
```

Arguments

`type`	Character. Type of data to update. One of "all", "wells", "lithology"
`download`	Logical. Whether to re-download and process the data (`TRUE`), or just re-process it (`FALSE`).
`permission`	Logical. Permission to create the cache folder. If `FALSE`, user is asked for permission, if `TRUE`, permission is implied.

Examples

```
data_update(type = "lithology")
```

`dem_region` *Fetch and trim DEM of a region*

Description

This function takes a shape file of a region and creates a DEM of the region. Lidar data is stored locally as tiles. Tiles are only downloaded if they don't already exist unless `only_new = FALSE`. TRIM data is obtained via the `bcmaps` package and stored locally as tiles. **Note:** TRIM elevation is coarser than Lidar Use Lidar unless it is missing for your region of interest.

Usage

```
dem_region(
  region,
  type = "lidar",
  buffer = 1,
  lidar_dir = NULL,
  only_new = TRUE,
  progress = httr::progress()
)
```

Arguments

`region`	sf simple features object. Shape file of the region of interest.
`type`	Character. Type of DEM to download, either "lidar" or "trim". Use Lidar unless unavailable.
`buffer`	Numeric. Percent buffer to apply to the `region` spatial file before cropping the DEM data to match. Increase this value if you find that wells on the edge of your area aren't been matched to elevations when using `wells_elev()`.
`lidar_dir`	Character. File path of where Lidar tiles should be stored. Defaults to the cache directory. Only applies when `type = "lidar"`.
`only_new`	Logical. Whether to download all Lidar tiles, or only new tiles that don't exist locally. Defaults to TRUE. Only apples when `type = "lidar"`.
`progress`	Function. Progress bar to use. Generally leave as is.

Details

Lidar tiles are the newest tile available. If you have reason to need a historical file, contact the team to discuss your use case.

Value

stars spatiotemporal array object

Data Source

Lidar data is obtained from the LidarBC portal. The `tiles` data frame contains is an internally created data frame listing tiles and their respective download locations. Tiles to download are selected based on overlap between map tiles and the provided shapefile (`region`). These Lidar tiles can be browsed and downloaded manually via the LidarBC Open LiDAR Data Portal

The grid of map tiles is obtained from the BC Data Catalogue, BCGS 1:20,000 Grid

TRIM data is obtained via the `bcmaps` package from the BC government Data Catalogue based on overlap between map tiles and the provided shapefile (`region`).

Examples

```
library(sf)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Fetch Lidar DEM
creek_lidar <- dem_region(creek_sf)

plot(creek_lidar)

# Fetch TRIM DEM
creek_trim <- dem_region(creek_sf, type = "trim")
```

```
plot(creek_trim)
```

flags *Flags*

Description

A glossary of flag terms

Usage

```
flags
```

Format

`flags`**:**

A data frame with 10 rows and 2 columns:

Flag flag name

Description Flag description

lith_fix *Fix lithology descriptions*

Description

Clean and categorize lithology descriptions into primary, secondary, tertiary and final lithology categories. Generally this function is used internally when loading and cleaning GWELLS lithology.

Usage

```
lith_fix(file = "lithology.csv", desc = NULL)
```

Arguments

`file`	Character. Lithology file name stored in cache
`desc`	Character. Text string to convert (overrides `file`).

Details

However statements can be tested directly with this function to see how it works and for troubleshooting.

Value

Data frame of lithology categorizations

Examples

```
lith_fix(desc = "sandy gravel")

# basic spell checks
lith_fix(desc = "saandy gravel")
```

`tiles` *tiles*

Description

A spatial data frame of map tiles with corresponding links to Lidar tiles.

Usage

```
tiles
```

Format

`tiles`**:**

A data frame with 7,129 rows and 5 columns:

map_tile Tile name

geometry Spatial data

utm Projection

tile_name Lidar tile name

url Link to Lidar tile

Details

The spatial grid of map tiles is obtained from the BC Data Catalogue, BCGS 1:20,000 Grid

Links to Lidar tile urls are extracted from the list at the LidarBC Open LiDAR Data Portal

wells_elev *Subset wells and add elevation*

Description

This function takes a region shape file and the DEM of a region (output of `dem_region()`), subsets the wells data (from GWELLS) to this region and adds the elevation data.

Usage

```
wells_elev(wells_sub, dem, update = FALSE)
```

Arguments

`wells_sub`	sf spatial data frame. Subset of wells data output by `wells_subset()`
`dem`	stars simple features object. Output of `dem_region()`.
`update`	Logical. Force update of the data?

Value

sf spatial data frame

Examples

```
library(sf)
library(ggplot2)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek_sf)

# Fetch Lidar DEM
creek_lidar <- dem_region(creek_sf)

# Collect wells in this region with added elevation from Lidar
creek_wells <- wells_elev(creek_wells, creek_lidar)

ggplot() +
  geom_sf(data = creek_sf) +
  geom_sf(data = creek_wells, aes(colour = elev), size = 0.5,
          fill = "NA", show.legend = FALSE) +
 coord_sf(datum = st_crs(3005)) # BC Albers

# OR Fetch TRIM DEM
creek_trim <- dem_region(creek_sf, type = "trim")
```

```
# Collect wells in this region with added elevation from Lidar
creek_wells <- wells_elev(creek_wells, creek_trim)

ggplot() +
  geom_sf(data = creek_sf) +
  geom_sf(data = creek_wells, aes(colour = elev), size = 0.5,
          fill = "NA", show.legend = FALSE) +
 coord_sf(datum = st_crs(3005)) # BC Albers
```

wells_export *Export wells data for use in Strater and Voxler*

Description

Export wells data for use in Strater and Voxler

Usage

```
wells_export(wells_sub, id, type, dir = ".", preview = FALSE)
```

Arguments

`wells_sub`	Data frame. Output of `wells_elev()`
`id`	Character. Id to prepend to all output files e.g., "id_lith.csv"
`type`	Character. Format in which to export. One of "strater", "voxler", "archydro", "leapfrog", or "surfer" (case-insensitive).
`dir`	Character. Directory where files should be exported to. Defaults to working directory.
`preview`	Logical. Whether to preview the exports (`TRUE`, return a list of data frames) or to actually export the data (`FALSE`, write the necessary files to the `dir` folder.

Value

If `preview` = `FALSE`, a vector of file names, if `preview` = `TRUE`, a list of data frames.

Examples

```
library(sf)

# Load a shape file defining the region of interest
creek <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek)

# Fetch Lidar DEM
```

```
creek_lidar <- dem_region(creek)

# Collect wells in this region with added elevation from Lidar
creek_wells <- wells_elev(creek_wells, creek_lidar)

# Preview data for Strater
p <- wells_export(creek_wells, id = "clinton", type = "strater", preview = TRUE)
names(p)
p[["strater_lith"]]
p[["strater_collars"]]
p[["strater_wells"]]

# Export data for Strater
wells_export(creek_wells, id = "clinton", type = "strater")

# Export Arc Hydro
wells_export(creek_wells, id = "clinton", type = "archydro")

# Export Surver
wells_export(creek_wells, id = "clinton", type = "surfer")
```

wells_subset *Subset wells to region*

Description

Filter the GWELLS data returning only wells within the provided shapefile.

Usage

```
wells_subset(region, update = FALSE)
```

Arguments

`region`	sf simple features object. Shape file of the region of interest.
`update`	Logical. Force update of the data?

Examples

```
library(sf)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek_sf)
```

`wells_yield` *Add yield lithology data to wells subset*

Description

Yield records are extracted from lithology observations and added to the wells data.

Usage

```
wells_yield(wells_sub)
```

Arguments

`wells_sub`	sf spatial data frame. Subset of wells data output by `wells_subset()`

Value

Data frame or sf spatial data frame with wells data and added yield from lithology.

Examples

```
library(sf)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek_sf)

# Get yield data for these wells
creek_yield <- wells_yield(creek_wells)
```

Index