Package: maxcovr (via r-universe)

September 24, 2024

Type Package

Title A Set of Tools For Solving The Maximal Covering Location Problem

Version 0.1.3.9200

Maintainer Nicholas Tierney <`nicholas.tierney@gmail.com`>

Description Solving the ``maximal covering location problem'' as described by Church can be difficult for users not familiar with linear programming. maxcovr provides functions to make it easy to solve this problem, and tools to calculate facility coverage.

Depends R (>= 3.2.2)

License GPL-3

RoxygenNote 6.1.1

LazyData true

ByteCompile TRUE

URL `https://github.com/njtierney/maxcovr`

BugReports `https://github.com/njtierney/maxcovr/issues`

Imports lpSolve, dplyr, tidyr, tibble, readr, Rcpp, purrr, magrittr, Rglpk

LinkingTo Rcpp

Suggests testthat (>= 2.1.0), covr, gurobi, knitr, rmarkdown, modelr, leaflet, ggplot2

Encoding UTF-8

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

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

RemoteUrl https://github.com/njtierney/maxcovr

RemoteRef HEAD

RemoteSha 2e905223183560ffaed4e89425459204e51400d2

Contents

`augment_facility_relocated` *Find distance from relocated and proposed new sites*

Description

This takes the proposed sites and the existing sites, with additional information from the model, and then returns a dataframe of all of the existing facilities that were relocated, and provides the distance to the nearest facility, which is presumably the location to which it was relocated to.

Usage

```
augment_facility_relocated(proposed_facility, existing_facility)
```

Arguments

`proposed_facility`	facilities proposed for the model - but this data has extra information (`is_installed`) in it.
`existing_facility`	facilities existing for the model - but this data has extra information (`is_relocated`) in it.

Value

dataframe

Examples

```
## Not run:

mc_cv_n100_test %>%
  mutate(facility_distances = map2(
    .x = proposed_facility,
    .y = existing_facility,
    .f = augment_facility_relocated)) %>%
  select(facility_distances) %>%
  .[[1]]


## End(Not run)
```

augment_user *Augment users data; add useful information*

Description

This returns the user dataframe, with added columns containing distance between that user and a given facility - IDs are generated for IDs and facilities that correspond to their row number.

Usage

```
augment_user(facilities_selected, existing_facilities, existing_users)
```

Arguments

facilities_selected	dataframe of facilities selected, obtained from extract_facility_selected
existing_facilities	existing facilities
existing_users	existing users

Value

tibble of users, with distances between each user and facility

Examples

```
## Not run:

mc_facilities_selected <-extract_facility_selected(
  solution_vector = x$lp_solution$solution,
  A_mat = x$A,
  proposed_facilities = x$proposed_facility)

augmented_users <- augment_user(
    facilities_selected = mc_facilities_selected,
    existing_facilities = mc_cv_fit_n20_test_1$existing_facility,
    existing_users = mc_cv_fit_n20_test_1$existing_user
    )

## End(Not run)
```

augment_user_tested *Nearest wrapper*

Description

This function is wrapper to `nearest`, adding `is_covered` to the model. This function is explicit about inputs, and is useful in cross validation - evaluating how test data performs against suggested facilities in the training set. This might be added to `nearest`, and may become obsolete.

Usage

```
augment_user_tested(all_facilities, test_data, distance_threshold = 100)
```

Arguments

`all_facilities`	data.frame Facilities selected in maxcovr model
`test_data`	data.frame test data (but it could be any `user`-type data)
`distance_threshold`	numeric

Value

dataframe containing distances between each test data observation and the nearest facility.

Examples

```
## Not run:

mc_cv_relocate_n100_cut %>%
  mutate(user_nearest_test = map2(
    .x = facilities_selected,
    .y = test,
    .f = augment_user_tested
    ))


## End(Not run)
```

`binary_distance_matrix` *(Internal) Create a binary distance matrix*

Description

This is a wrapper function that returns a logical matrix, of 1 if distance between element i, j is less than or equal to the distance_cutoff, and 0 otherwise.

Usage

```
binary_distance_matrix(facility, user, distance_cutoff,
  d_proposed_user = NULL)
```

Arguments

`facility`	data.frame of facilities
`user`	data.frame of users
`distance_cutoff`	integer of distance to use for cutoff
`d_proposed_user`	Option distance matrix between proposed facilities and users (see Examples).

Value

a logical matrix, of 1 if distance between element i, j is less than or equal to the distance_cutoff, and 0 otherwise.

`binary_matrix_cpp` *Create a binary matrix TRUE if distance satisfies a condition*

Description

Create a binary matrix TRUE if distance satisfies a condition

Usage

```
binary_matrix_cpp(facility, user, distance_cutoff)
```

Arguments

`facility`	a matrix with longitude and latitude in the first two columns
`user`	a matrix with longitude and latitude in the first two columns
`distance_cutoff`	numeric indicating the distance cutoff (in metres) you are interested in. If a number is less than distance_cutoff, it will be 1, if it is greater than it, it will be 0.

Value

a logical matrix 1 if distance between element i, j is less than or equal to the distance_cutoff, and 0 otherwise

---

`coverage` *Create a summary of the coverage between two dataframes*

Description

In the york building and york crime context, writing `nearest(york_crime,york)` reads as "find the nearest crime in york to each building in york, and returns a dataframe with every building in york, the nearest york_crime to each building, and the distance in metres between the two."

Usage

```
coverage(nearest_df, to_df, distance_cutoff = 100, ...)
```

Arguments

`nearest_df`	dataframe containing latitude and longitude
`to_df`	dataframe containing latitude and longitude
`distance_cutoff`	integer the distance threshold you are interested in assessing coverage at
`...`	extra arguments to pass to nearest

Value

a dataframe containing information about the distance threshold uses (distance_within), the number of events covered and not covered (n_cov, n_not_cov), the percentage covered and not covered (pct_cov, pct_not_cov), and the average distance and sd distance.

Examples

```
library(dplyr)

# already existing locations
york_selected <- york %>% filter(grade == "I")

# proposed locations
york_unselected <- york %>% filter(grade != "I")
coverage(york_selected, york_crime)
coverage(york_crime, york_selected)
```

`deg2rad` *Convert from degrees to radians*

Description

Convert from degrees to radians

Usage

```
deg2rad(deg)
```

Arguments

`deg`	A numeric vector in units of degrees.

Value

The input numeric vector, converted to units of radians.

`deg2rad_cpp` *Convert degrees to radians*

Description

Convert degrees to radians

Usage

```
deg2rad_cpp(deg)
```

Arguments

`deg`	degrees

Value

radians

`distance_matrix_cpp` *Create a matrix of distances between two areas*

Description

Create a matrix of distances between two areas

Usage

```
distance_matrix_cpp(facility, user)
```

Arguments

`facility`	a matrix with longitude and latitude in the first two columns
`user`	a matrix with longitude and latitude in the first two columns

Value

a matrix of distances in metres between each user and facility, with nrow(user) rows and nrow(facility) columns.

`extract_existing_coverage` *Extract the existing coverage*

Description

Extract the existing coverage

Usage

```
extract_existing_coverage(existing_facilities, existing_users,
  distance_cutoff)
```

Arguments

`existing_facilities`	the existing facilities
`existing_users`	the existing users
`distance_cutoff`	the distance cutoffs

Value

tibble of existing coverage

Examples

```
## Not run:
extract_existing_coverage(existing_facilities = x$existing_facility,
   existing_users = x$existing_user,
   distance_cutoff = x$distance_cutoff)

## End(Not run)
```

---

extract_facility_selected

Extract Selected Facilities

Description

This takes the linear programming solution, the A matrix, and the proposed facilities. It returns a tibble, which contains the facilities chosen from the proposed facilities.

Usage

```
extract_facility_selected(solution_vector, A_mat, proposed_facilities)
```

Arguments

solution_vector	vector from lp_solution$solution
A_mat	The "A" matrix from the solver
proposed_facilities	Dataframe of proposed facilities

Value

dataframe of selected facilities

Examples

```
# assuming that you've run max_coverage using lpSolve, then you
# will save the model output before the extraction process
# as `x`.
## Not run:
mc_facilities_selected <- extract_facility_selected(
  solution_vector = x$lp_solution$solution,
  A_mat = x$A,
  proposed_facilities = x$proposed_facility)

## End(Not run)
```

`extract_mc_results` *(Internal) Summarise maxcovr model using facility and user information*

Description

`extract_mc_results` takes a fitted `max_coverage` object and returns useful summary information from the model. It exists so that the manipulation functions for the outcomes from the solver have another home - this makes it easier to maintain this package, and heeds to this idea of having functions that are specialised. The name of this function is likely to change in the near future.

Usage

```
extract_mc_results(x)
```

Arguments

`x`	the fitted model from `max_coverage`.

Value

a list containing multiple dataframes summarising the model

`extract_mc_results_relocation` *(Internal) Summarise maxcovr relocation model with facility and user info*

Description

`extract_mc_results_relocation` takes a fitted max_coverage object and returns useful summary information from the model, specifically for the relocation method.

Usage

```
extract_mc_results_relocation(x)
```

Arguments

`x`	the fitted model from max_coverage_relocation

Value

a list containing multiple dataframes summarising the model

extract_model_coverage *Extract a one-row summary of the model coverage*

Description

This function takes the users information, the distance cutoff, and the number of facilities added, and then returns a one-row dataframe containing summary information about the coverage.

Usage

```
extract_model_coverage(augmented_user, distance_cutoff, n_added)
```

Arguments

`augmented_user`	dataframe obtained from `augment_user()`
`distance_cutoff`	numeric of the distance cutoff
`n_added`	numeric of the number of facilities added

Value

tibble of summary coverage info

Examples

```
## Not run:

augmented_users <- augment_user(
    facilities_selected = mc_facilities_selected,
    existing_facilities = x$existing_facility,
    existing_users = x$existing_user)

extract_model_coverage(
    augmented_user = augmented_users,
    distance_cutoff = x$distance_cutoff,
    n_added = x$n_added)

## End(Not run)
```

`extract_users_affected` *Extract users affected*

Description

Extract additional users affected by new coverage from the new facilities

Usage

```
extract_users_affected(A_mat, solution_vector, user_id, users_not_covered)
```

Arguments

`A_mat`	A matrix
`solution_vector`	The vector of solutions
`user_id`	The IDs of the individuals
`users_not_covered`	those users not covered by original AEDs

Value

tibble taken from `users`, those who are affectd by new placements

Examples

```
## Not run:
extract_users_affected(
    A_mat = x$A,
    solution_vector = x$lp_solution$solution,
    user_id = x$user_id,
    users_not_covered = x$user_not_covered)

## End(Not run)
```

`facility_user_dist` *facility_user_dist*

Description

Uses haversines formula to calculate the distance between lat/long co-ordinates of every facility and every user, returning a data_frame. You can think of "facilities" as something like mobile towers, police centres, or AED locations, and "users" as something like individual houses, crime locations, or heart attack locations. The motivating example for this function was finding the distance from Automatic Electronic Defibrillators (AEDs) to each Out of Hospital Cardiac Arrest (OHCA), where the locations for AEDs and OHCAs are in separate dataframes. Currently facifacility_user_dist makes the strict assumption that the facility and user dataframes have columns named aed_id, lat, and long, and ohca_id, lat, and long. This will be updated soon.

Usage

```
facility_user_dist(facility, user, coverage_distance = 100,
  nearest = "facility")
```

Arguments

`facility`	a dataframe containing columns named "lat", and "long".
`user`	a dataframe containing columns "lat", and "long".
`coverage_distance`	numeric indicating the coverage level for the facilities to be within in metres to a user. Default value is 100 metres.
`nearest`	character Can be "facility", "user", and "both". Defaults to "facility". When set to "facility", returns a dataframe where every row refers to every user, and the closest facility to each user. When set to "user", it returns a dataframe where every row is every facility, and the closest user to each facility. When set to "both", which will return every pairwise combination of distances. Be careful when default is "facility".

Value

a data frame containing the two datasets joined together with columns named facility_id, lat_facility, long_facility, user_id, lat_user, long_user, distance in meters between each the given facility and user in a row.

`facility_user_indic` *facility_user_indic*

Description

This is a data manipulation function for facility_user_dist. This function creates a spread matrix of the distances between each ohca and each aed. There is an ohca_id column, and then a column for each aed_id, with a given cell being the distance between an ohca in a row, and that column. This distance is converted into an indicator variable, based upon whether that distance is less than the provided dist_indic parameter. In the future I might change the dist_indic function to be optional, but this whole function mainly exists to make it easier to do the computation in the max_coverage function.

Usage

```
facility_user_indic(df_dist, dist_indic)
```

Arguments

`df_dist`	dataframe from facility_user_dist. Requires nearest = "both"
`dist_indic`	an indicator of the distance you want to be TRUE / FALSE

Value

dataframe with variables ohca_id, and aed_id_number, with the id from each aed_id being transposed into each column name.

`find_users_not_covered` *(Internal) Create a dataframe of the users not covered*

Description

(Internal) Create a dataframe of the users not covered

Usage

```
find_users_not_covered(existing_facility, user, distance_cutoff,
  d_existing_user = NULL)
```

Arguments

`existing_facility`	data.frame of existing facilities
`user`	data.frame of existing users
`distance_cutoff`	integer of distance cutoff
`d_existing_user`	Optional distance matrix between existing facilities and users.

Value

data.frame of those users not covered by current facilities

`is.maxcovr` *Test if the object is a maxcovr object*

Description

Test if the object is a maxcovr object

Usage

```
is.maxcovr(x)
```

Arguments

`x`	An object

Value

`TRUE` if the object inherits from the `maxcovr` class.

`is.maxcovr_relocation` *Test if the object is a maxcovr_relocation object*

Description

Test if the object is a maxcovr_relocation object

Usage

```
is.maxcovr_relocation(x)
```

Arguments

`x`	An object

Value

`TRUE` if the object inherits from the `maxcovr_relocation` class.

`maxcovr` *maxcovr*

Description

maxcovr

`max_coverage` *Solve the Maximal Covering Location Problem*

Description

`max_coverage` solves the binary optimisation problem known as the "maximal covering location problem" as described by Church (http://www.geo .ucsb.edu/~forest/G294download/MAX_COVER_RLC_CSR.pdf). This package was implemented to make it easier to solve this problem in the context of the research initially presented by Chan et al (http://circ.ahajournals.org/content/127/17/1801.short) to identify ideal locations to place AEDs.

Usage

```
max_coverage(existing_facility, proposed_facility, user, distance_cutoff,
  n_added, d_existing_user = NULL, d_proposed_user = NULL,
  solver = "glpk")
```

Arguments

`existing_facility`	data.frame containing the facilities that are already in existing, with columns names lat, and long.
`proposed_facility`	data.frame containing the facilities that are being proposed, with column names lat, and long.
`user`	data.frame containing the users of the facilities, along with column names lat, and long.
`distance_cutoff`	numeric indicating the distance cutoff (in metres) you are interested in. If a number is less than distance_cutoff, it will be 1, if it is greater than it, it will be 0.
`n_added`	the maximum number of facilities to add.
`d_existing_user`	Optional distance matrix between existing facilities and users. Default distances are direct (geospherical ellipsoidal) distances; this allows alternative measures such as street-network distances to be submitted (see Examples).
`d_proposed_user`	Option distance matrix between proposed facilities and users (see Examples).
`solver`	character "glpk" (default) or "lpSolve". "gurobi" is currently in development, see `https://github.com/njtierney/maxcovr/issues/25`

Value

dataframe of results

Examples

```
library(dplyr)

# already existing locations
york_selected <- york %>% filter(grade == "I")

# proposed locations
york_unselected <- york %>% filter(grade != "I")

mc_result <- max_coverage(existing_facility = york_selected,
                          proposed_facility = york_unselected,
                          user = york_crime,
                          distance_cutoff = 100,
                          n_added = 20)

mc_result

summary(mc_result)

# get the facilities chosen
mc_result$facility_selected

# get the users affected
mc_result$user_affected

# get the summaries
mc_result$summary

# Example of street-network distance calculations
## Not run:
library(dodgr)
net <- dodgr_streetnet_sf ("york england") %>%
    weight_streetnet (wt_profile = "foot")

from <- match_points_to_graph (v, york_selected [, c ("long", "lat")])
to <- match_points_to_graph (v, york_crime [, c ("long", "lat")])
d_existing_user <- dodgr_dists (net, from = from, to = to)

from <- match_points_to_graph (v, york_unselected [, c ("long", "lat")])
d_proposed_user <- dodgr_dists (net, from = from, to = to)

mc_result <- max_coverage(existing_facility = york_selected,
                          proposed_facility = york_unselected,
                          user = york_crime,
                          distance_cutoff = 100,
                          n_added = 20,
                          d_existing_user = d_existing_user,
                          d_proposed_user = d_proposed_user)


## End(Not run)
```

`max_coverage_relocation` *Maximum Coverage when considering relocation*

Description

This function adds a relocation step

Usage

```
max_coverage_relocation(existing_facility = NULL, proposed_facility,
  user, distance_cutoff, cost_install, cost_removal, cost_total,
  solver = "lpSolve", return_early = FALSE)
```

Arguments

`existing_facility`	data.frame containing the facilities that are already in existing, with columns names lat, and long.
`proposed_facility`	data.frame containing the facilities that are being proposed, with column names lat, and long.
`user`	data.frame containing the users of the facilities, along with column names lat, and long.
`distance_cutoff`	numeric indicating the distance cutoff (in metres) you are interested in. If a number is less than distance_cutoff, it will be 1, if it is greater than it, it will be 0.
`cost_install`	integer the cost of installing a new facility
`cost_removal`	integer the cost of removing a facility
`cost_total`	integer the total cost allocated to the project
`solver`	character "glpk" (default) or "lpSolve". "gurobi" is currently in development, see `https://github.com/njtierney/maxcovr/issues/25`
`return_early`	logical TRUE if I do not want to run the extraction process, FALSE if I want to just return the lpsolve model etc.

Value

dataframe of results

Examples

```
## Not run:

library(dplyr)
# subset to be the places with towers built on them.

york_selected <- york %>% filter(grade == "I")

york_unselected <- york %>% filter(grade != "I")

# OK, what if I just use some really crazy small data to optimise over.

#

mc_relocate <-  max_coverage_relocation(existing_facility = york_selected,
                                        proposed_facility = york_unselected,
                                        user = york_crime,
                                        distance_cutoff = 100,
                                        cost_install = 5000,
                                        cost_removal = 200,
                                        cost_total = 600000)

mc_relocate

summary(mc_relocate)


## End(Not run)
```

`nearest` *Find the nearest lat/long to another lat/long*

Description

This function finds the nearest lat/long pairs to another lat/long pair. So in the york building and york crime context, writing `nearest(york_crime,york)` reads as "find the nearest crime in york to each building in york, and returns a dataframe with every building in york, the nearest york_crime to each building, and the distance in metres between the two. Likewise, you could write `nearest(york, york_crime)`, and this would return the nearest building to every crime. `nearest` assumes that the names of the latitude and longitude are "lat" and "long", but you can provide these names.

Usage

```
nearest(nearest_df, to_df, nearest_lat = "lat", nearest_long = "long",
  to_lat = "lat", to_long = "long")
```

Arguments

`nearest_df`	a dataframe containing latitude and longitude
`to_df`	a dataframe containing latitude and longitude
`nearest_lat`	name of latitude in nearest_df
`nearest_long`	name of longitude in nearest_df
`to_lat`	name of latitude in to_df
`to_long`	name of longitude in to_df

Value

dataframe of "to_df" along with the nearest "nearest_df" to each row, along with the distance between the two, and the nearest_id, the row position of the nearest_df closest to that row.

Examples

```
library(maxcovr)

nearest(nearest_df = york_crime,
        to_df = york)

# you can use the pipe as well

## Not run:

library(magrittr)
york_crime %>% nearest(york)


## End(Not run)
```

`nearest_facility_dist` *nearest facility + distance to a user*

Description

nearest facility + distance to a user

Usage

```
nearest_facility_dist(facility, user)
```

Arguments

`facility`	a matrix with longitude and latitude in the first two columns
`user`	a matrix with longitude and latitude in the first two columns

Value

matrix with 3 columns: user_id, facility_id, distance, where the user_id is the identifier for the user, the facility_id is the identifier for the facility that is closest to that user, and the distance is the distance in metres from that user to that facility.

`nearest_facility_distances` *(Internal) Calculate the nearest facility distances*

Description

This function is a wrapper for the similarly named, `nearest_facility_dist` function used inside `max_coverage` to calculate distances so that the nearest facilities can be found.

Usage

```
nearest_facility_distances(existing_facility, user)
```

Arguments

`existing_facility`	dataframe of existing facilities
`user`	dataframe of users to place facilities to cover

Value

A tibble with 3 columns: user_id, facility_id, distance, where the user_id is the identifier for the user, the facility_id is the identifier for the facility that is closest to that user, and the distance is the distance in metres from that user to that facility.

`n_installed` *Extract the number of facilities installed*

Description

Using the model-modified dataframe of `proposed_facility`, count the number of events installed.

Usage

```
n_installed(proposed_facility)
```

Arguments

`proposed_facility`	dataframe from the mc_model, of facilities proposed with the additional information about whether the facility was installed or not - `is_installed`

Value

datafrmae

Examples

```
## Not run:

mc_cv_n100_test %>%
    mutate(n_installed = map(
        .x = proposed_facility,
        .f = n_installed
    )) %>%
    select(n_installed) %>%
    .[[1]]


## End(Not run)
```

`n_relocated` *Extract the number of facilities relocated.*

Description

Extract the number of facilities relocated.

Usage

```
n_relocated(existing_facility)
```

Arguments

`existing_facility`
: the facilities originally existing, as output from the model (e.g., `model$existing_facility[[1]]`)

Value

dataframe containing one column of the number of things relocated

Examples

```
## Not run:

mc_cv_n100_test %>%
  mutate(n_relocated = map(
    .x = existing_facility,
    .f = n_relocated)) %>%
  select(n_relocated) %>%
  .[[1]]
```

```
## End(Not run)
```

---

spherical_distance *Calculate the distance between two locations*

Description

This function uses the haversine formula to calculate the great circle distance between two locations, identified by their latitudes and longitudes. It is borrowed from rnoaa (https://github.com/ropenscilabs/rnoaa/blob/master/R/m and included here as rnoaa is a large package that is rather unrelated to maxcovr. I have renamed it from meteo_spherical_distance to spherical_distance

Usage

```
spherical_distance(lat1, long1, lat2, long2)
```

Arguments

`lat1`	Latitude of the first location.
`long1`	Longitude of the first location.
`lat2`	Latitude of the second location.
`long2`	Longitude of the second location.

Value

A numeric value giving the distance in meters between the pair of locations.

Note

This function assumes an earth radius of 6,371 km.

Author(s)

Alex Simmons <`a2.simmons@qut.edu.au`>, Brooke Anderson <`brooke.anderson@colostate.edu`>

Examples

```
spherical_distance(lat1 = -27.4667,
                   long1 = 153.0217,
                   lat2 = -27.4710,
                   long2 = 153.0234)
```

spherical_distance_cpp

Calculate distance using haversines formula

Description

Calculate distance using haversines formula

Usage

```
spherical_distance_cpp(lat1, long1, lat2, long2)
```

Arguments

`lat1`	latitude from the first location
`long1`	longitude from the first location
`lat2`	latitude from the second location
`long2`	longitude from the second location

Value

distance in metres between two locations

spherical_distance_cpp_vec

Calculate (vectorized) distance using haversines formula

Description

Calculate (vectorized) distance using haversines formula

Usage

```
spherical_distance_cpp_vec(lat1, long1, lat2, long2)
```

Arguments

`lat1`	latitude from the first location
`long1`	longitude from the first location
`lat2`	latitude from the second location
`long2`	longitude from the second location

Value

distance in metres between two locations

`summarise_coverage` *summarise_coverage*

Description

Provides summary information of the coverage, using the distance dataframe created by `facility_user_dist`().

Usage

```
summarise_coverage(df_dist, distance_cutoff = 100)
```

Arguments

`df_dist`	distance matrix, as computed by facility_user_dist
`distance_cutoff`	the critical distance range that you would like to know. The default is 100m.

Value

dataframe

`summarise_relocated_dist` *Find the average distance from facilities relocated to their final place*

Description

This takes data from the function `augment_facility_relocated` function of the same name and then summarises it to find the average and sd of the distance between the two.

Usage

```
summarise_relocated_dist(augment_facility_relocated)
```

Arguments

`augment_facility_relocated`	dataframe from function: `augment_facility_relocated`

Value

dataframe

Examples

```
## Not run:

mc_cv_n100_test %>%
    mutate(
        facility_distances = map2(
            .x = proposed_facility,
            .y = existing_facility,
            .f = augment_facility_relocated
        ),
        summary_relocated_dist = map(
            .x = facility_distances,
            .f = summarise_relocated_dist
        )
    ) %>%
    # select(facility_distances) %>%
    select(summary_relocated_dist) %>%
    .[[1]]


## End(Not run)
```

---

summarise_user_cov *Summarise the coverage for users*

Description

This uses a `user` dataframe obtained from something like `augment_user_tested`.

Usage

```
summarise_user_cov(user)
```

Arguments

`user`	dataframe of users with distances between each user and the nearest facility (`distance`), and whether this is within the distance threshold (`is_covered`).

Value

dataframe containing information on the number of users, the number of events covered, the proportion of events covered, and the distance from each

Examples

```
## Not run:

summarise_user_cov(augmented_user_test)



## End(Not run)
```

---

summary_mc_cv *Summary for max_coverage cross validation*

Description

Summary for max_coverage cross validation

Usage

```
summary_mc_cv(model, test_data)
```

Arguments

`model`	the cross validated model
`test_data`	the cross validated test data

Value

a summary dataframe

Examples

```
## Not run:

library(maxcovr)
library(tidyverse)

york_selected <- york %>% filter(grade == "I")
york_unselected <- york %>% filter(grade != "I")

mc_cv_fixed <- modelr::crossv_kfold(york_crime, 5) %>%
                 mutate(test = map(test,as_tibble),
                 train = map(train,as_tibble))

mc_cv_fit <- map_df(mc_cv_fixed$train,
                    ~max_coverage(existing_facility = york_selected,
                    proposed_facility = york_unselected,
                    user = .,
                    n_added = 20,
                    distance_cutoff = 100))

 summary_mc_cv(mc_cv_fit,
              mc_cv_fixed$test)


## End(Not run)
```

`summary_mc_cv_relocate`
Summary for max_coverage cross validation for relocation models

---

Description

Summary for max_coverage cross validation for relocation models

Usage

```
summary_mc_cv_relocate(model, test_data)
```

Arguments

`model`	the cross validated model
`test_data`	the cross validated test data

Value

a summary dataframe

Examples

```
## Not run:

library(maxcovr)
library(tidyverse)

york_selected <- york %>% filter(grade == "I")
york_unselected <- york %>% filter(grade != "I")

mc_cv <- modelr::crossv_kfold(york_crime, 5) %>%
                 mutate(test = map(test,as_tibble),
                 train = map(train,as_tibble))

mc_cv_relocate <- map_df(mc_cv$train,
                    ~max_coverage_relocation(existing_facility = york_selected,
                               proposed_facility = york_unselected,
                               user = .,
                               cost_install = 2500,
                               cost_removal = 700,
                               cost_total = 50000,
                               distance_cutoff = 100,
                               solver = "gurobi"))

 summary_mc_cv_relocate(mc_cv_relocate, mc_cv$test)


## End(Not run)
```

---

york *York Listed Buildings.*

Description

Listed buildings provided by the City of York Council, made available here: https://data.gov.uk/dataset/listed-buildings24/resource/8c32fb55-0e40-457f-98f9-6494503e283b. This data contains public sector information licensed under the Open Government Licence v3.0: https://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/.

Usage

```
york
```

Format

A data frame with seven variables: `long`, `lat`, `object_id`, `desig_id`, `pref_ref`, `name`, and `grade`.

`long` longitude of the building

`lat` latitude of the building

`object_id` unique identifier for the building

`desig_id` ID related to a feature that is not yet known to me

`pref_ref` ID related to a feature that is not yet known to me

`name` name of the building

`grade` one of the three (I, II, III) cateogories of listed buildings

For further details, see https://www.york.gov.uk/info/20215/conservation_and_listed_buildings/1346/listed_buildings and https://data.gov.uk/dataset/listed-buildings24/resource/8c32fb55-0e40-457f-98f9-6494503e283b

york_crime *York Crime Locations.*

Description

Crime locations obtained via the ukpolice R package: https://github.com/njtierney/ukpolice, which uses the data made available through the uk crime API:<data.police.uk/docs/>. This data contains public sector information licensed under the Open Government Licence v3.0: https://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/.

Usage

```
york_crime
```

Format

A data frame with variables: `category`, `persistent_id`, `date`, `lat`, `long`, `street_id`, `street_name`, `context`, `id`, `location_type`, `location_subtype`, and `outcome_status`. '

- category: Category of the crime (https://data.police.uk/docs/method/crime-street/)
- persistent_id: 64-character unique identifier for that crime. (This is different to the existing 'id' attribute, which is not guaranteed to always stay the same for each crime.)
- date: Date of the crime YYYY-MM
- latitude: Latitude
- longitude: Longitude
- street_id: Unique identifier for the street
- street_name: Name of the location. This is only an approximation of where the crime happened
- context: Extra information about the crime (if applicable)
- id: ID of the crime. This ID only relates to the API, it is NOT a police identifier
- location_type: The type of the location. Either Force or BTP: Force indicates a normal police force location; BTP indicates a British Transport Police location. BTP locations fall within normal police force boundaries.
- location_subtype: For BTP locations, the type of location at which this crime was recorded.
- outcome_status: The category and date of the latest recorded outcome for the crime

Note

more documentation here: https://data.police.uk/docs/method/crime-street/

For further details, see https://www.york.gov.uk/info/20215/conservation_and_listed_buildings/1346/listed_buildings and https://data.gov.uk/dataset/listed-buildings24/resource/8c32fb55-0e40-457f-98f9-6494503e283b

Index