Package ‘ggmap’

February 5, 2019

Version 3.0.0
Title Spatial Visualization with ggplot2
Description A collection of functions to visualize spatial data and models on top of static maps from various online sources (e.g. Google Maps and Stamen Maps). It includes tools common to those tasks, including functions for geolocation and routing.

URL https://github.com/dkahle/ggmap
BugReports https://github.com/dkahle/ggmap/issues
Depends R (>= 3.1.0), ggplot2 (>= 2.2.0)
Imports RgoogleMaps, png, plyr, rjson, jpeg, digest, scales, dplyr, bitops, grid, glue, httr, stringr, purrr, magrittr, tibble, tidyr
Suggests MASS, hexbin, testthat
License GPL-2
LazyData true
RoxygenNote 6.1.0
Encoding UTF-8
NeedsCompilation no
Author David Kahle [aut, cre] (<https://orcid.org/0000-0002-9999-1558>), Hadley Wickham [aut] (<https://orcid.org/0000-0003-4757-117X>), Scott Jackson [aut], Mikko Korpela [ctb]
Maintainer David Kahle <david.kahle@gmail.com>
Repository CRAN
Date/Publication 2019-02-05 10:19:04

R topics documented:

bb2bbox .......................................................... 2
calc_zoom ......................................................... 4
Convert a bb specification to a bbox specification

In ggmap, all maps (class ggmap) have the bb attribute, a data frame bounding box specification in terms of the bottom left and top right points of the spatial extent. This function converts this specification to a named double vector (with names left, bottom, right, top) specification that is used in some querying functions (e.g. get_stamenmap).
Usage

bb2bbox(bb)

Arguments

bb  a bounding box in bb format (see examples)

Value

a bounding box in bbox format (see examples)

Author(s)

David Kahle <david.kahle@gmail.com>

Examples

## Not run: # cut down on R CMD check time

# grab a center/zoom map and compute its bounding box
gc <- geocode("white house, washington dc")
map <- get_map(gc)
(bb <- attr(map, "bb"))
(bbox <- bb2bbox(bb))

# use the bounding box to get a stamen map
stamMap <- get_stamenmap(bbox)

ggmap(map) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )

ggmap(stamMap) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )

## End(Not run)
Description

calc_zoom can calculate a zoom based on either (1) a data frame with longitude and latitude variables, (2) a longitude range and latitude range, or (3) a bounding box (bbox specification). The specification for (1) is identical to that of most R functions, for (2) simply put in a longitude range into lon and a latitude range into lat, and for (3) put the bounding box in for the lon argument.

Usage

calc_zoom(lon, lat, data, adjust = 0, f = 0.05)

Arguments

lon    longitude, see details
lat    latitude, see details
data    (optional) a data frame containing lon and lat as variables
adjust number to add to the calculated zoom
f    argument to pass to make_bbox

See Also

make_bbox, bb2bbox

Examples

# From data
calc_zoom(lon, lat, wind)

# From range
lon_range <- extendrange( wind$lon )
lat_range <- extendrange( wind$lat )
calc_zoom(lon_range, lat_range)

# From bounding box
box <- make_bbox(lon, lat, data = crime)
calc_zoom(box)
Description

Lightly cleaned Houston crime from January 2010 to August 2010 geocoded with Google Maps

Author(s)

Houston Police Department, City of Houston

References

http://www.houstontx.gov/police/cs/stats2.htm

geocode

Description

Geocodes (finds latitude and longitude of) a location using the Google Geocoding API. Note: To use Google’s Geocoding API, you must first enable the API in the Google Cloud Platform Console. See ?register_google.

Usage

geocode(location, output = c("latlon", "latlona", "more", "all"),
source = c("google", "dsk"), force = ifelse(source == "dsk", FALSE, TRUE), urlonly = FALSE, override_limit = FALSE,
nameType = c("long", "short"), ext = "com", inject = ",", ...)

mutate_geocode(data, location, ...)

geocodeQueryCheck()

Arguments

location a character vector of street addresses or place names (e.g. "1600 pennsylvania avenue, washington dc" or "Baylor University")
output amount of output, "latlon", "latlona", "more", or "all"
source "google" for Google (note: "dsk" is defunct)
force force online query, even if cached (previously downloaded)
urlonly return only the url?
override_limit override the current query rate
geocode

nameType in some cases, Google returns both a long name and a short name. this parameter allows the user to specify which to grab.

ext top level domain (e.g. "com", "co.nz"); helpful for non-US users

inject character string to add to the url or named character vector of key-value pairs to be injected (e.g. c("a" = "b") get converted to "a=b" and appended to the query)

... ...

data a data frame or equivalent

Value

If output is "latlon", "latlona", or "more", a tibble (classed data frame). If "all", a list.

Author(s)

David Kahle <david.kahle@gmail.com>

See Also


Examples

## Not run: requires Google API key, see ?register_google

## basic usage

# geocoding is most commonly used for addresses
geocode("1600 Amphitheatre Parkway, Mountain View, CA")
geocecode("1600 Amphitheatre Parkway, Mountain View, CA", urlonly = TRUE)

# google can also geocode colloquial names of places
geocode("the white house")

# geocode can also accept character vectors of places
geocode(c("the white house", "washington dc"))

## types of output

geocode("waco texas")
geocecode("waco texas", output = "latlona")
geocecode("waco texas", output = "more")
str(geocecode("waco texas", output = "all"))
```r
geocode(c("waco, texas", "houston, texas"))
geocode(c("waco, texas", "houston, texas"), output = "latlon")
geocode(c("waco, texas", "houston, texas"), output = "all") %>% str()

## mutate_geocode

# mutate_geocode is used to add location columns to an existing dataset
# that has location information

df <- data.frame(
  address = c("1600 Pennsylvania Avenue, Washington DC", ",", "houston texas"),
  stringsAsFactors = FALSE
)
mutate_geocode(df, address)
df %>% mutate_geocode(address)

## known issues

# in some cases geocode finds several locations
geocode("waco city hall")

## End(Not run)
```

---

```r
geom_leg

Single line segments with rounded ends

Description

This is ggplot2’s segment with rounded ends. It’s mainly included in ggmmap for historical reasons.
Single line segments with rounded ends

Usage

geom_leg(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", arrow = NULL, lineend = "round",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)

geom_leg(mapping = NULL, data = NULL, stat = "identity",
  position = "identity", arrow = NULL, lineend = "round",
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, ...)
```
Arguments

mapping    mapping
data       data
stat       stat
position   position
arrow      arrow
lineend    Line end style (round, butt, square).
na.rm      If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

Details

only intended for use in ggmap package. only designed for mercator projection.
only intended for use in ggmap package. only designed for mercator projection.

See Also

geom_segment in ggplot2, inspired by http://spatialanalysis.co.uk/2012/02/great-maps-ggplot2/, route
geom_segment in ggplot2, inspired by http://spatialanalysis.co.uk/2012/02/great-maps-ggplot2/, route

Examples

## Not run: # removed for R CMD check speed

map <- get_map(
    location = c(-77.0425, 38.8925), # painfully picked by hand
    source = "google", zoom = 14, maptype = "satellite"
)
ggmap(map)

(legs_df <- route(
    "the white house, dc",
    "lincoln memorial washington dc",
    alternatives = TRUE
))
```r
ggplot(data = legs_df) +
geom_leg(aes(
  x = startLon, xend = endLon,
  y = startLat, yend = endLat
)) +
coord_map()

ggplot(data = legs_df) +
geom_leg(aes(
  x = startLon, xend = endLon,
  y = startLat, yend = endLat,
  color = route
)) +
coord_map()

ggmap(map) +
geom_leg(
  aes(
    x = startLon, xend = endLon,
    y = startLat, yend = endLat
  ),
  data = legs_df, color = "red"
)

# adding a color aesthetic errors because of a base-layer problem
# gggmap(map) +
#  geom_leg(
#    aes(
#      x = startLon, xend = endLon,
#      y = startLat, yend = endLat,
#      color = route
#    )
#  )

# this is probably the easiest hack to fix it
ggplot(data = legs_df) +
inset_ggmap(map) +
geom_leg(
  aes(
    x = startLon, xend = endLon,
    y = startLat, yend = endLat,
    color = route
  ),
  data = legs_df
) +
coord_map()

## End(Not run)
```
## Not run: requires Google API key, see ?register_google

```r
map <- get_map(
  location = c(-77.0425, 38.8925), # painfully picked by hand
  source = "google", zoom = 14, maptype = "satellite"
)
ggmap(map)

(legs_df <- route(
  "the white house, dc",
  "lincoln memorial washington dc",
  alternatives = TRUE
))

ggplot(data = legs_df) +
  geom_leg(aes(
    x = start_lon, xend = end_lon,
    y = start_lat, yend = end_lat
  )) +
  coord_map()

ggplot(data = legs_df) +
  geom_leg(aes(
    x = start_lon, xend = end_lon,
    y = start_lat, yend = end_lat,
    color = route
  )) +
  coord_map()

ggmap(map) +
  geom_leg(
    aes(
      x = start_lon, xend = end_lon,
      y = start_lat, yend = end_lat
    ),
    data = legs_df, color = "red"
  )

# adding a color aesthetic errors because of a base-layer problem
# ggmap(map) +
# geom_leg(~
#   aes(
#     x = start_lon, xend = end_lon,
#     y = start_lat, yend = end_lat,
#     color = route
#   )
# )

# this is probably the easiest hack to fix it
```
get_cloudmademap

Description

get_cloudmademap accesses a tile server for Stamen Maps and downloads/stitches map tiles/formats a map image. This function requires an api key which can be obtained for free from http://cloudmade.com/user/show (defunct?). Thousands of maptypes ("styles"), including create-your-own options, are available from http://maps.cloudmade.com/editor (defunct).

Usage

get_cloudmademap(bbox = c(left = -95.80204, bottom = 29.38048, right = -94.92313, top = 30.14344), zoom = 10, api_key = , maptype = 1, highres = TRUE, crop = TRUE, messaging = FALSE, urlonly = FALSE, filename = NULL, color = c("color", "bw"), ...)

Arguments

bbox a bounding box in the format c(lowerleftlon, lowerleftlat, upperrightlon, upperrightlat).
zoom a zoom level
api_key character string containing cloud made api key, see details
maptype an integer of what cloud made calls style, see details
highres double resolution
crop crop raw map tiles to specified bounding box
messaging turn messaging on/off
urlonly return url only
filename destination file for download (file extension added according to format). Default NULL means a random tempfile.
color color or black-and-white
... ...
get_googlemap

Value

a ggmap object (a classed raster object with a bounding box attribute)

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

http://maps.cloudmade.com/ (defunct), ggmap

Examples

```r
#api_key <- 'your api key here'
#api_key <- 'b23b0358e87c4ff9f81029eda25c903'

#map <- get_cloudmademap(api_key = api_key
#ggmap(map)

#map <- get_cloudmademap(maptype = 997, api_key = api_key)
#ggmap(map)
```

get_googlemap

Get a Google Map.

Description

get_googlemap queries the Google Maps Static API version 2 to download a static map. Note that in most cases by using this function you are agreeing to the Google Maps API Terms of Service at https://cloud.google.com/maps-platform/terms/. Note that as of mid-2018, registering with Google Cloud to obtain an API key is required to use any of Google’s services, including get_googlemap. Usage and billing may apply, see the links under See Also further down in this documentation for more details.

Usage

```r
googlemap(center = c(lon = -95.3632715, lat = 29.7632836),
  zoom = 10, size = c(640, 640), scale = 2, format = c("png8",
  "gif", "jpg", "jpg-baseline", "png32"), maptype = c("terrain",
  "satellite", "roadmap", "hybrid"), language = "en-EN",
  messaging = FALSE, urlonly = FALSE, filename = NULL,
  color = c("color", "bw"), force = FALSE, where = tempdir(),
  archiving = FALSE, ext = "com", inject = "", region, markers, path,
  visible, style, ...)
```
Arguments

center  the center of the map; either a longitude/latitude numeric vector or a string containing a location, in which case geocode is called with source = "google". (default: c(lon = -95.3632715, lat = 29.7632836), houston, texas)

zoom  map zoom; an integer from 3 (continent) to 21 (building), default value 10 (city)

size  rectangular dimensions of map in pixels - horizontal x vertical - with a max of c(640, 640). this parameter is affected in a multiplicative way by scale.

scale  multiplicative factor for the number of pixels returned possible values are 1, 2, or 4 (e.g. size = c(640,640) and scale = 2 returns an image with 1280x1280 pixels). 4 is reserved for google business users only. scale also affects the size of labels as well.

format  character string providing image format - png, jpeg, and gif formats available in various flavors

maptype  character string providing google map theme. options available are "terrain", "satellite", "roadmap", and "hybrid"

language  character string providing language of map labels (for themes with them) in the format "en-EN". not all languages are supported; for those which aren’t the default language is used

messaging  turn messaging on/off

urlonly  return url only

filename  destination file for download (file extension added according to format). Default NULL means a random tempfile.

color  color or black-and-white

force  if the map is on file, should a new map be looked up?

where  where should the file drawer be located (without terminating "/")

archiving  use archived maps. note: by changing to TRUE you agree to the one of the approved uses listed in the Google Maps API Terms of Service : http://developers.google.com/maps/terms.

ext  domain extension (e.g. "com", "co.nz")

inject  character string to add to the url

region  borders to display as a region code specified as a two-character ccTLD ("top-level domain") value, see http://en.wikipedia.org/wiki/List_of_Internet_top-level_domains#Country_code_top-level_domains

markers  data.frame with first column longitude, second column latitude, for which google markers should be embedded in the map image, or character string to be passed directly to api

path  data.frame (or list of data.frames) with first column longitude, second column latitude, for which a single path should be embedded in the map image, or character string to be passed directly to api

visible  a location as a longitude/latitude numeric vector (or data frame with first column longitude, second latitude) or vector of character string addresses which should be visible in map extent
get_googlemap

style character string to be supplied directly to the api for the style argument or a named vector (see examples). this is a powerful complex specification, see https://developers.google.com/maps/documentation/staticmaps/

Value

a ggmap object (a classed raster object with a bounding box attribute)

Author(s)

David Kahle <david.kahle@gmail.com>

See Also


Examples

## Not run: requires Google API key, see ?register_google

## basic usage

```
(map <- get_googlemap(c(-97.14667, 31.5493)))
gmap(map)
```

# plotting based on a colloquial name
# this requires a geocode() call, and needs that API
get_googlemap("waco, texas") %>% ggmap()

# different maptypes are available
get_googlemap("waco, texas", maptype = "satellite") %>% ggmap()
get_googlemap("waco, texas", maptype = "hybrid") %>% ggmap()

# you can get the url as follows
# see ?register_google if you want the key printed
get_googlemap(urlonly = TRUE)

## other usage

```
# markers and paths are easy to access
d <- function(x=-95.36, y=29.76, n,r,a){
  round(data.frame(
    lon = jitter(rep(x,n), amount = a),
    lat = jitter(rep(y,n), amount = a)
  ))
```

get_map

), digits = r
}
df <- rdata = n=50, r=3, a=.3
map <- get_googlemap(markers = df, path = df, scale = 2)
ggmap(map)
geom_point(aes(x = lon, y = lat), data = df, size = 3, colour = "black") +
ageom_path(aes(x = lon, y = lat), data = df)

gc <- geocode("waco, texas", source = "google")
center <- as.numeric(gc)
ggmap(get_googlemap(center = center, color = "bw", scale = 2), extent = "device")

# the scale argument can be seen in the following
# (make your graphics device as large as possible)
ggmap(get_googlemap(center, scale = 1), extent = "panel") # pixelated
ggmap(get_googlemap(center, scale = 2), extent = "panel") # fine

# archiving; note that you must meet google's terms for this condition
map <- get_googlemap(archiving = TRUE)
map <- get_googlemap()
map <- get_googlemap()
ggmap(map)

# style
map <- get_googlemap(
  maptype = "roadmap",
  style = c(feature = "all", element = "labels", visibility = "off"),
  colour = "bw"
)
ggmap(map)

## End(Not run)

get_map

Grab a map.

Description

get_map is a smart wrapper that queries the Google Maps, OpenStreetMap, Stamen Maps or Naver Map servers for a map.
Usage

get_map(location = c(lon = -95.3632715, lat = 29.7632836),
  zoom = "auto", scale = "auto", maptype = c("terrain",
  "terrain-background", "satellite", "roadmap", "hybrid", "toner",
  "watercolor", "terrain-labels", "terrain-lines", "toner-2010",
  "toner-2011", "toner-background", "toner-hybrid", "toner-labels",
  "toner-lines", "toner-lite"), source = c("google", "osm", "stamen"),
  force = ifelse(source == "google", TRUE, FALSE), messaging = FALSE,
  urlonly = FALSE, filename = NULL, crop = TRUE, color = c("color",
  "bw"), language = "en-EN", ...)

Arguments

location an address, longitude/latitude pair (in that order), or left/bottom/right/top bounding box
zoom map zoom, an integer from 3 (continent) to 21 (building), default value 10 (city).
openstreetmaps limits a zoom of 18, and the limit on stamen maps depends on the maptype. "auto" automatically determines the zoom for bounding box specifications, and is defaulted to 10 with center/zoom specifications. maps of the whole world currently not supported.
scale scale argument of get_googlemap or get_openstreetmap
maptype character string providing map theme. options available are "terrain", "terrain-background", "satellite", "roadmap", and "hybrid" (google maps), "terrain", "watercolor", and "toner" (stamen maps), or a positive integer for cloudmade maps (see ?get_cloudmademap)
source Google Maps ("google"), OpenStreetMap ("osm"), Stamen Maps ("stamen")
force force new map (don’t use archived version)
messaging turn messaging on/off
urlonly return url only
filename destination file for download (file extension added according to format). Default NULL means a random tempfile.
crop (stamen and cloudmade maps) crop tiles to bounding box
color color ("color") or black-and-white ("bw")
language language for google maps
...

Value

a ggmap object (a classed raster object with a bounding box attribute)

Author(s)

David Kahle <david.kahle@gmail.com>
See Also

`ggmap`, `GetMap` in package RgoogleMaps

Examples

```r
## Not run: some requires google API key, see ?register_google

## basic usage

# lon-lat vectors automatically use google:
(map <- get_map(c(-97.14667, 31.5493)))
str(map)
ggmap(map)

# bounding boxes default to stamen
(map <- get_map(c(left = -97.1268, bottom = 31.536245, right = -97.099334, top = 31.559652)))
ggmap(map)

# characters default to google
(map <- get_map("orlando, florida"))
ggmap(map)

## basic usage

(map <- get_map(maptype = "roadmap"))
(map <- get_map(source = "osm"))
(map <- get_map(source = "stamen", maptype = "watercolor"))

map <- get_map(location = "texas", zoom = 6, source = "stamen")
ggmap(map, fullpage = TRUE)

## End(Not run)
```

get_navermap  Get a Naver Map

Description

This is (at least) temporarily unavailable as the Naver API changed.

Usage

```r
get_navermap(center = c(lon = 126.9849208, lat = 37.5664519), zoom = 4,
size = c(640, 640), format = c("png", "jpeg", "jpg"),
```
get_navermap


Arguments

center  the center of the map. this can be longitude/latitude numeric vector.
zoom  map zoom, an integer from 1 to 14 (building), default value 10
size  rectangular dimensions of map in pixels - horizontal x vertical - with a max of c(640, 640).
format  character string providing image format - png, jpeg(jpg) formats available in various flavors
crs  Coordinate system, this currently supports EPSG:4326
baselayer  base layer, this can be either "default", "satellite".
color  color or black-and-white
overlayers  overlay layers, this can be "anno_satellite","bicycle","roadview","traffic".
markers  data.frame with first column longitude, second column latitude, for which naver markers should be embedded in the map image, or character string to be passed directly to api
key  key code from naver api center
uri  registered host url
filename  destination file for download (file extension added according to format). Default NULL means a random tempfile.
messaging  turn messaging on/off
urlonly  return url only
force  if the map is on file, should a new map be looked up?
where  where should the file drawer be located (without terminating "/")
arcing  use archived maps, note: by changing to TRUE you agree to abide by any of the rules governing caching naver maps

Details

get_navermap accesses the Naver Static Maps API version 1.1 to download a static map. Note that in most cases by using this function you are agreeing to the Naver Maps API Terms of Service.

Author(s)
Heewon Jeon <madjakarta@gmail.com>
See Also

- ggmap

get_openstreetmap

Get an OpenStreetMap

Description

get_openstreetmap accesses a tile server for OpenStreetMap and downloads/formats a map image. This is simply a wrapper for the web-based version at http://www.openstreetmap.org/. If you don't know how to get the map you want, go there, navigate to the map extent that you want, click the export tab at the top of the page, and copy the information into this function.

Usage

get_openstreetmap(bbox = c(left = -95.80204, bottom = 29.38048, right = -94.92313, top = 30.14344), scale = 606250, format = c("png", "jpeg", "svg", "pdf", "ps"), messaging = FALSE, urlonly = FALSE, filename = NULL, color = c("color", "bw"), ...)

Arguments

- bbox: a bounding box in the format c(lowerleftlon, lowerleftlat, upperrightlon, upperrightlat)
- scale: scale parameter, see http://wiki.openstreetmap.org/wiki/MinScaleDenominator. smaller scales provide a finer degree of detail, where larger scales produce more coarse detail. The scale argument is a tricky number to correctly specify. In most cases, if you get an error when downloading an openstreetmap the error is attributable to an improper scale specification. OSM_scale_lookup can help; but the best way to get in the correct range is to go to http://www.openstreetmap.org/, navigate to the map of interest, click export at the top of the page, click 'map image' and then copy down the scale listed.
- format: character string providing image format - png, jpeg, svg, pdf, and ps formats
- messaging: turn messaging on/off
- urlonly: return url only
- filename: destination file for download (file extension added according to format). Default NULL means a random tempfile.
- color: color or black-and-white

Details

receive an error message from download.file with the message HTTP status '503 Service Unavailable'. You can confirm this by setting urlonly = TRUE, and then entering the URL in a web browser. the solution is either (1) change sources or (2) wait for the OSM servers to come back up. See http://www.openstreetmap.org/copyright for license and copyright information.
get_stamenmap

Value

a ggmap object (a classed raster object with a bounding box attribute)

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

http://www.openstreetmap.org/, ggmap

Examples

# get_openstreetmap(urlonly = TRUE)

# osm servers get overloaded, which can result in
# erroneous failed checks

# osm <- get_openstreetmap()
# ggmap(osm)

generate_stamenmap

get_stamenmap

Get a Stamen Map

description

get_stamenmap accesses a tile server for Stamen Maps and downloads/stitches map tiles/formats a
map image. Note that Stamen maps don’t cover the entire world.

Usage

get_stamenmap(bbox = c(left = -95.80204, bottom = 29.38048, right =
-94.92313, top = 30.14344), zoom = 10, mtype = c("terrain",
"terrain-background", "terrain-labels", "terrain-lines", "toner",
"toner-2010", "toner-2011", "toner-background", "toner-hybrid",
"toner-labels", "toner-lines", "toner-lite", "watercolor"),
crop = TRUE, messaging = FALSE, urlonly = FALSE,
color = c("color", "bw"), force = FALSE, where = tempdir(), ...)

generate_stamenmap_tile_download_fail_log()

retry_stamen_map_download()
get_stamenmap

Arguments

bbox  a bounding box in the format c(lowerleftlon, lowerleftlat, upperrightlon, upperrightlat).
zoom  a zoom level
maptype terrain, terrain-background, terrain-labels, terrain-lines, toner, toner-2010, toner-2011, toner-background, toner-hybrid, toner-labels, toner-lines, toner-lite, or watercolor.
crop  crop raw map tiles to specified bounding box. if FALSE, the resulting map will more than cover the bounding box specified.
messaging turn messaging on/off
urlonly return url only
color  color or black-and-white (use force = TRUE if you’ve already downloaded the images)
force if the map is on file, should a new map be looked up?
where  where should the file drawer be located (without terminating "/")

Value

a ggmap object (a classed raster object with a bounding box attribute)

See Also

httpZOOmapsNstamenNcomOCwatercolor, ggmap

Examples

## Not run: some requires Google API key, see ?register_google; heavy network/time load

## basic usage

bbox <- c(left = -97.1268, bottom = 31.536245, right = -97.099334, top = 31.559652)

ggmap(get_stamenmap(bbox, zoom = 13))
ggmap(get_stamenmap(bbox, zoom = 14))
ggmap(get_stamenmap(bbox, zoom = 15))
ggmap(get_stamenmap(bbox, zoom = 16))

place <- "mount everest"
(google <- get_googlemap(place, zoom = 9))
ggmap(google)
bbox_everest <- c(left = 86.05, bottom = 27.21, right = 87.81, top = 28.76)
ggmap(get_stamenmap(bbox_everest, zoom = 9))
## get_stamenmap

### map types

```r
place <- "rio de janeiro"
google <- get_googlemap(place, zoom = 10)
ggmap(google)

bbox <- bb2bbox(attr(google, "bb"))

get_stamenmap(bbox, maptype = "terrain") %>% ggmap()
get_stamenmap(bbox, maptype = "terrain-background") %>% ggmap()
get_stamenmap(bbox, maptype = "terrain-labels") %>% ggmap()
get_stamenmap(bbox, maptype = "terrain-lines") %>% ggmap()
get_stamenmap(bbox, maptype = "toner") %>% ggmap()
get_stamenmap(bbox, maptype = "toner-2010") %>% ggmap()
get_stamenmap(bbox, maptype = "toner-2011") %>% ggmap()
get_stamenmap(bbox, maptype = "toner-background") %>% ggmap()
get_stamenmap(bbox, maptype = "toner-hybrid") %>% ggmap()
get_stamenmap(bbox, maptype = "toner-labels") %>% ggmap()
get_stamenmap(bbox, maptype = "toner-lines") %>% ggmap()
get_stamenmap(bbox, maptype = "toner-lite") %>% ggmap()
get_stamenmap(bbox, maptype = "watercolor") %>% ggmap()
```

### zoom levels

```r
get_stamenmap(bbox, maptype = "watercolor", zoom = 11) %>% ggmap(extent = "device")
get_stamenmap(bbox, maptype = "watercolor", zoom = 12) %>% ggmap(extent = "device")
# get_stamenmap(bbox, maptype = "watercolor", zoom = 13) %>% ggmap(extent = "device")
# get_stamenmap(bbox, maptype = "watercolor", zoom = 14) %>% ggmap(extent = "device")
# get_stamenmap(bbox, maptype = "watercolor", zoom = 15) %>% ggmap(extent = "device")
# get_stamenmap(bbox, maptype = "watercolor", zoom = 16) %>% ggmap(extent = "device")
# get_stamenmap(bbox, maptype = "watercolor", zoom = 17) %>% ggmap(extent = "device")
get_stamenmap(bbox, maptype = "watercolor", zoom = 18) %>% ggmap(extent = "device")
```

### more examples

```r
gc <- geocode("rio de janeiro")

get_stamenmap(bbox, zoom = 10) %>% ggmap() +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

get_stamenmap(bbox, zoom = 10, crop = FALSE) %>% ggmap() +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

get_stamenmap(bbox, zoom = 10, maptype = "watercolor") %>% ggmap() +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)
```
ggimage

```r
get_stamenmap(bbox, zoom = 10, maptype = "toner") %>% ggmap() +
geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

# continental united states labels
c("left" = -125, "bottom" = 25.75, "right" = -67, "top" = 49) %>%
get_stamenmap(zoom = 5, maptype = "toner-labels") %>%
ggmap()

# accuracy check - white house
gc <- geocode("the white house")
qmap("the white house", zoom = 16) +
geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 3)
qmap("the white house", zoom = 16, source = "stamen", maptype = "terrain") +
geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 3)

## known issues
#################################################################
# in some cases stamen's servers will not return a tile for a given map
# this tends to happen in high-zoom situations, but it is not always
# clear why it happens. these tiles will appear as blank parts of the map.

# ggmap provides some tools to try to recover the missing tiles, but the
# servers seem pretty persistent at not providing the maps.
bbox <- c(left = -97.1268, bottom = 31.536245, right = -97.099334, top = 31.559652)
ggmap(get_stamenmap(bbox, zoom = 17))
get_stamen_tile_download_fail_log()
retry_stamen_map_download()

## End(Not run)
```

---

**ggimage**  
*Plot an image using ggplot2*
Description

ggimage is the near ggplot2 equivalent of image.

Usage

ggimage(mat, fullpage = TRUE, coord_equal = TRUE, scale_axes = FALSE)

Arguments

mat a matrix, imagematrix, array, or raster (something that can be coerced by as.raster)
fullpage should the image take up the entire viewport?
coord_equal should the axes units be equal?
scale_axes should the axes be [0,ncol(mat)-1]x[0,nrow(mat)-1] (F) or [0,1]x[0,1] (T)

Value

a ggplot object

Author(s)

David Kahle <david.kahle@gmail.com>

Examples

img <- matrix(1:16, 4, 4)
image(img)
ggimage(t(img[,4:1]), fullpage = FALSE, scale_axes = TRUE)
ggimage(t(img[,4:1]), fullpage = FALSE)

## Not run:
# not run due to slow performance

data(hadley)
ggimage(hadley)
ggimage(hadley, coord_equal = FALSE)

x <- seq(1, 438, 15); n <- length(x)
df <- data.frame(x = x, y = -(120*(scale((x - 219)^3 - 25000*x) + rnorm(n)/2 - 3)))
qplot(x, y, data = df, geom = c('smooth','point'))
ggimage(hadley, fullpage = FALSE) +
  geom_smooth(aes(x = x, y = y), fill = I('gray60'), data = df,
              colour = I('green'), size = I(1)) +
  geom_point(aes(x = x, y = y), data = df,
             colour = I('green'), size = I(3), fill = NA)

## End(Not run)
**gglocator**  
*Locator for ggplot objects*

**Description**
Locator for ggplot objects (Note: only accurate when extent = "normal" when using ggmap.)

**Usage**
```r
gglocator(n = 1, message = FALSE, mercator = TRUE, ...)
```

**Arguments**
- `n`: number of points to locate.
- `message`: unused.
- `mercator`: logical flag; should the plot be treated as using the projection common to most web map services? Set to FALSE if the axes on the plot use a linear scale.
- `...`: additional arguments (including deprecated, e.g. `xexpand`)

**Value**
a data frame with columns according to the x and y aesthetics

**Author(s)**
Tyler Rinker, Baptiste Auguie, DWIn, David Kahle, @Nikolai-Hlubek and @mvkorpel.

**Examples**
```r
if (interactive()) {

  # only run for interactive sessions
  df <- expand.grid(x = 0:-5, y = 0:-5)

  ggplot(df, aes(x, y)) + geom_point() +
    annotate(geom = "point", x = -2, y = -2, colour = "red")

  (pt <- gglocator(mercator = FALSE)) # click red point

  last_plot() +
    annotate("point", pt$x, pt$y, color = "blue", size = 3, alpha = .5)

  hdf <- get_map("houston, texas")
  ggrepal(hdf, extent = "normal")
  (pt <- gglocator(mercator = TRUE))
  last_plot() +
    annotate("point", pt$lon, pt$lat, color = "blue", size = 3, alpha = .5)
```
ggmap  

Plot a ggmap object

Description

ggmap plots the raster object produced by get_map.

Usage

ggmap(ggmap, extent = "panel", base_layer, maprange = FALSE, 
legend = "right", padding = 0.02, darken = c(0, "black"), b, 
fullpage, expand, ...)

Arguments

- **ggmap**: an object of class ggmap (from function get_map)
- **extent**: how much of the plot should the map take up? "normal", "device", or "panel" (default)
- **base_layer**: a ggplot(aes(...), ...) call; see examples
- **maprange**: logical for use with base_layer; should the map define the x and y limits?
- **legend**: "left", "right" (default), "bottom", "top", "bottomleft", "bottomright", "topleft", "topright", "none" (used with extent = "device")
- **padding**: distance from legend to corner of the plot (used with legend, formerly b)
- **darken**: vector of the form c(number, color), where number is in [0, 1] and color is a character string indicating the color of the darken. 0 indicates no darkening, 1 indicates a black-out.
- **b**: Deprecated, renamed to ‘padding’. Overrides any ‘padding’ argument.
- **fullpage**: Deprecated, equivalent to ‘extent = "device“’ when ‘TRUE’. Overrides any ‘extent’ argument.
- **expand**: Deprecated, equivalent to ‘extent = "panel“’ when ‘TRUE’ and ‘fullpage’ is ‘FALSE’. When ‘fullpage’ is ‘FALSE’ and ‘expand’ is ‘FALSE’, equivalent to ‘extent="normal“’. Overrides any ‘extent’ argument.

... ...

Value

- a ggplot object
Author(s)
David Kahle <david.kahle@gmail.com>

See Also
get_map, qmap

Examples

## Not run: ## map queries drag R CMD check

## extents and legends

hdf <- get_map("houston, texas")
ggmap(hdf, extent = "normal")
ggmap(hdf) # extent = "panel", note qmap defaults to extent = "device"
ggmap(hdf, extent = "device")

# make some fake spatial data
mu <- c(-95.3632715, 29.7632836); nDataSets <- sample(4:10,1)
chkpts <- NULL
for(k in 1:nDataSets){
a <- rnorm(2); b <- rnorm(2);
si <- 1/3000 * (outer(a,a) + outer(b,b))
chkpts <- rbind(chkpts,

  cbind(MASS::mvrnorm(rpois(1,50), jitter(mu,.01), si), k)
)
}
chkpts <- data.frame(chkpts)
names(chkpts) <- c("lon", "lat","class")
chkpts$class <- factor(chkpts$class)
qplot(lon, lat, data = chkpts, colour = class)

# show it on the map
ggmap(hdf, extent = "normal") +
  geom_point(aes(x = lon, y = lat, colour = class), data = chkpts, alpha = .5)

  geom_point(aes(x = lon, y = lat, colour = class), data = chkpts, alpha = .5)

  geom_point(aes(x = lon, y = lat, colour = class), data = chkpts, alpha = .5)

  theme_set(theme_bw())
  ggrepel::geom_point(aes(x = lon, y = lat, colour = class), data = chkpts, alpha = .5)
ggmap(hdf, extent = "device", legend = "topleft") +
    geom_point(aes(x = lon, y = lat, colour = class), data = chkpts, alpha = .5)

# qmplot is great for this kind of thing...
qmplot(lon, lat, data = chkpts, color = class, darken = .6)
qmplot(lon, lat, data = chkpts, geom = "density2d", color = class, darken = .6)

## maprange

hdf <- get_map()
mu <- c(-95.3632715, 29.7632836)
points <- data.frame(MASS::mvrnorm(1000, mu = mu, diag(c(.1, .1))))
names(points) <- c("lon", "lat")
points$class <- sample(c("a", "b"), 1000, replace = TRUE)

ggmap(hdf) + geom_point(data = points) # maprange built into extent = panel, device

ggmap(hdf, extent = "normal") + geom_point(data = points)
# note that the following is not the same as extent = panel

ggmap(hdf, extent = "normal", maprange = TRUE) + geom_point(data = points)

# and if you need your data to run off on a extent = device (legend included)

 ggmap(hdf, extent = "normal", maprange = TRUE) +
    geom_point(aes(colour = class), data = points) +
    theme_nothing(legend = TRUE) + theme(legend.position = "right")

# again, qmplot is probably more useful
qmplot(lon, lat, data = points, color = class, darken = .4, alpha = I(.6))
qmplot(lon, lat, data = points, color = class, maptype = "toner-lite")

## cool examples

# contour overlay

ggmap(get_map(maptype = "satellite"), extent = "device") +
    stat_density2d(aes(x = lon, y = lat, colour = class), data = chkpts, bins = 5)

# adding additional content

library(grid)
baylor <- get_map("one bear place, waco, texas", zoom = 15, maptype = "satellite")
ggmap(baylor)

# use gglocator to find lon/lat"s of interest

(clicks <- gglocator(2))
ggmap(baylor) +
    geom_point(aes(x = lon, y = lat), data = clicks, colour = "red", alpha = .5)
expand.grid(lon = clicks$lon, lat = clicks$lat)

ggmap(baylor) + theme_bw() +
    annotate("segment", x=-97.110, xend=-97.1188, y=31.5450, yend=31.5485,
```r
colour = I("red"), arrow = arrow(length=unit(0.3,"cm"), size = 1.5) +
annotate("label", x=-97.113, y=31.5445, label = "Department of Statistical Science",
colour = I("red"), size = 3.5) +
labs(x = "Longitude", y = "Latitude") + ggttitle("Baylor University")

baylor <- get_map("marrs mclean science, waco, texas", zoom = 16, maptype = "satellite")

# a shapefile like layer
data(zips)
ggmap(get_map(maptype = "satellite", zoom = 8), extent = "device") +
gem_polyon(aes(x = lon, y = lat, group = plotOrder),
  data = zips, colour = NA, fill = "red", alpha = .2) +
gem_path(aes(x = lon, y = lat, group = plotOrder),
  data = zips, colour = "white", alpha = .4, size = .4)

library(plyr)
zipsLabels <- ddply(zips,.(zip), function(df){
  df[,c("area", "perimeter", "zip", "lonCent", "latCent")]
})
ggmap(get_map(maptype = "satellite", zoom = 9),
  extent = "device", legend = "none", darken = .5) +
gem_text(aes(x = lonCent, y = latCent, label = zip, size = area),
  data = zipsLabels, colour = I("red")) +
  scale_size(range = c(1.5,6))

qgplot(lonCent, latCent, data = zipsLabels, geom = "text",
  label = zip, size = area, maptype = "toner-lite", color = I("red")
)

## crime data example

# only violent crimes
violent_crimes <- subset(crime,
  offense != "auto theft" &
  offense != "theft" &
  offense != "burglary"
)

# rank violent crimes
violent_crimes$offense <-
  factor(violent_crimes$offense,
    levels = c("robbery", "aggravated assault",
```
"rape", "murder")

# restrict to downtown
violent_crimes <- subset(violent_crimes,
          -95.39681 <= lon & lon <= -95.34188 &
          29.73631 <= lat & lat <= 29.78400)

# get map and bounding box
theme_set(theme_bw())
HoustonMap <- qmap("houston", zoom = 14, color = "bw",
          extent = "device", legend = "topleft")
HoustonMap <- ggmap(
          get_map("houston", zoom = 14, color = "bw"),
          extent = "device", legend = "topleft")

# the bubble chart
HoustonMap +
  geom_point(aes(x = lon, y = lat, colour = offense, size = offense), data = violent_crimes) +
  scale_colour_discrete("Offense", labels = c("Robbery","Aggravated Assault","Rape","Murder")) +
  scale_size_discrete("Offense", labels = c("Robbery","Aggravated Assault","Rape","Murder"),
          range = c(1.75,6)) +
  guides(size = guide_legend(override_aes = list(size = 6))) +
  theme(
      legend.key.size = grid::unit(1.8,"lines"),
      legend.title = element_text(size = 16, face = "bold"),
      legend.text = element_text(size = 14)
  ) +
  labs(colour = "Offense", size = "Offense")

# doing it with qmplot is even easier
qmplot(lon, lat, data = violent_crimes, maptype = "toner-lite",
          color = offense, size = offense, legend = "topleft")

# or, with styling:
qmplot(lon, lat, data = violent_crimes, maptype = "toner-lite",
          color = offense, size = offense, legend = "topleft")

  scale_colour_discrete("Offense", labels = c("Robbery","Aggravated Assault","Rape","Murder")) +
  scale_size_discrete("Offense", labels = c("Robbery","Aggravated Assault","Rape","Murder"),
          range = c(1.75,6)) +
  guides(size = guide_legend(override_aes = list(size = 6))) +
  theme(
      legend.key.size = grid::unit(1.8,"lines"),
      legend.title = element_text(size = 16, face = "bold"),
      legend.text = element_text(size = 14)
  ) +
  labs(colour = "Offense", size = "Offense")
# a contour plot
HoustonMap +
  stat_density2d(aes(x = lon, y = lat, colour = offense),
                  size = 3, bins = 2, alpha = 3/4, data = violent_crimes) +
  scale_colour_discrete("Offense", labels = c("Robbery", "Aggravated Assault", "Rape", "Murder")) +
  theme(
    legend.text = element_text(size = 15, vjust = .5),
    legend.title = element_text(size = 15, face = "bold"),
    legend.key.size = grid::unit(1.8, "lines")
)

# 2d histogram...
HoustonMap +
  stat_bin_2d(aes(x = lon, y = lat, colour = offense, fill = offense),
               size = .5, bins = 30, alpha = 2/4, data = violent_crimes) +
  scale_colour_discrete("Offense",
                        labels = c("Robbery", "Aggravated Assault", "Rape", "Murder"),
                        guide = FALSE) +
  scale_fill_discrete("Offense", labels = c("Robbery", "Aggravated Assault", "Rape", "Murder")) +
  theme(
    legend.text = element_text(size = 15, vjust = .5),
    legend.title = element_text(size = 15, face = "bold"),
    legend.key.size = grid::unit(1.8, "lines")
)

# changing gears (get a color map)
hoon <- get_map("houston", zoom = 14)
HoustonMap <- ggmap(houston, extent = "device", legend = "topleft")

# a filled contour plot...
HoustonMap +
  stat_density2d(aes(x = lon, y = lat, fill = ..level.., alpha = ..level..),
                 size = 2, bins = 4, data = violent_crimes, geom = "polygon") +
  scale_fill_gradient("Violent\nCrime\nDensity") +
  scale_alpha(range = c(.4, .75), guide = FALSE) +
  guides(fill = guide_colorbar(barwidth = 1.5, barheight = 10))

# ... with an insert
overlay <- stat_density2d(aes(x = lon, y = lat, fill = ..level.., alpha = ..level..),
                           bins = 4, geom = "polygon", data = violent_crimes)
attr(houston,"bb") # to help finding (x/y)(min/max) vals below

HoustonMap +
  stat_density2d(aes(x = lon, y = lat, fill = ..level.., alpha = ..level..),
    bins = 4, geom = "polygon", data = violent_crimes) +
  scale_fill_gradient("Violent\nCrime\nDensity") +
  scale_alpha(range = c(.4,.75), guide = FALSE) +
  guides(fill = guide_colorbar(barwidth = 1.5, barheight = 10)) +
  inset(
    grob = ggplotGrob(ggplot()) + overlay +
      scale_fill_gradient("Violent\nCrime\nDensity") +
      scale_alpha(range = c(.4,.75), guide = FALSE) +
      theme_inset()
  ),
  xmin = -95.35877, xmax = -95.34229,
  ymin = 29.73754, ymax = 29.75185
)

## more examples

# you can layer anything on top of the maps (even meaningless stuff)
df <- data.frame(
  lon = rep(seq(-95.39, -95.35, length.out = 8), each = 20),
  lat = sapply(
    rep(seq(29.74, 29.78, length.out = 8), each = 20),
    function(x) rnorm(1, x, .002)
  ),
  class = rep(letters[1:8], each = 20)
)

qplot(lon, lat, data = df, geom = "boxplot", fill = class)

HoustonMap +
  geom_boxplot(aes(x = lon, y = lat, fill = class), data = df)

## the base_layer argument - faceting

df <- data.frame(
  x = rnorm(1000, -95.36258, .2),
  class = rep(letters[1:8], each = 20)

HoustonMap +
  geom_boxplot(aes(x = lon, y = lat, fill = class), data = df)


```r

# no apparent change because ggmap sets maprange = TRUE with extent = "panel"
ggmap(get_map(), base_layer = ggplot(aes(x = x, y = y), data = df)) +
  geom_point(colour = "red")

# ... but there is a difference
ggmap(get_map(), base_layer = ggplot(aes(x = x, y = y), data = df), extent = "normal") +
  geom_point(colour = "red")

# maprange can fix it (so can extent = "panel")
ggmap(get_map(), maprange = TRUE, extent = "normal",
      base_layer = ggplot(aes(x = x, y = y), data = df)) +
  geom_point(colour = "red")

# base_layer makes faceting possible
df <- data.frame(
  x = rnorm(10*100, -95.36258, .075),
  y = rnorm(10*100, 29.76196, .075),
  year = rep(paste("year", format(1:10)), each = 100)
)
ggmap(get_map(), base_layer = ggplot(aes(x = x, y = y), data = df)) +
  geom_point() + facet_wrap(~ year)

ggmap(get_map(), base_layer = ggplot(aes(x = x, y = y), data = df), extent = "device") +
  geom_point() + facet_wrap(~ year)

ggmap(get_map(), base_layer = ggplot(aes(x = x, y = y), data = df), extent = "device") +
  geom_point() + facet_wrap(~ year)

ggmap(get_map(), base_layer = ggplot(aes(x = x, y = y), data = df)) +
  stat_density2d(aes(fill = ..level.., alpha = ..level..),
                 bins = 4, geom = "polygon") +
  scale_fill_gradient2(low = "white", mid = "orange", high = "red", midpoint = 10) +
  scale_alpha(range = c(.2, .75), guide = FALSE) +
  facet_wrap(~ year)
```

---

```r

y = rnorm(1000, 29.76196, .2)

# simulated example

df <- data.frame(
  x = rnorm(10*100, -95.36258, .05),
  y = rnorm(10*100, 29.76196, .05),
  year = rep(paste("year", format(1:10)), each = 100)
)

for(k in 0:9){
  df$x[1:100 + 100*k] <- df$x[1:100 + 100*k] + sqrt(.05)*cos(2*pi*k/10)
  df$y[1:100 + 100*k] <- df$y[1:100 + 100*k] + sqrt(.05)*sin(2*pi*k/10)
}

ggmap(get_map(),
      base_layer = ggplot(aes(x = x, y = y), data = df)) +
      stat_density2d(aes(fill = ..level.., alpha = ..level..),
                      bins = 4, geom = "polygon") +
      scale_fill_gradient2(low = "white", mid = "orange", high = "red", midpoint = 10) +
      scale_alpha(range = c(.2, .75), guide = FALSE) +
      facet_wrap(~ year)
```
# crime example by month
levels(violent_crimes$month) <- paste(
  toupper(substr(levels(violent_crimes$month),1,1)),
  substr(levels(violent_crimes$month),2,20), sep = ""
)

houston <- get_map(location = "houston", zoom = 14, source = "osm", color = "bw")

HoustonMap <- ggmap(houston,
  base_layer = ggplot(aes(x = lon, y = lat), data = violent_crimes)
)

HoustonMap +
  stat_density2d(aes(x = lon, y = lat, fill = .level.., alpha = .level..),
    bins = I(5), geom = "polygon", data = violent_crimes) +
  scale_fill_gradient2("Violent\nCrime\nDensity",
    low = "white", mid = "orange", high = "red", midpoint = 500) +
  labs(x = "Longitude", y = "Latitude") + facet_wrap(~ month) +
  scale_alpha(range = c(.2, .55), guide = FALSE) +
  ggtitle("Violent Crime Contour Map of Downtown Houston by Month") +
  guides(fill = guide_colorbar(barwidth = 1.5, barheight = 10))

## darken argument

ggmap(get_map())

## End(Not run)
ggmapplot

Details
The following are defunct ggmap functions:

- get_cloudermap
- get_navermap
- get_openstreetmap

Don’t use this function, use ggmap.

Description
ggmap plots the raster object produced by get_map.

Usage

ggmapplot(ggmap, fullpage = FALSE, base_layer, maprange = FALSE,
expand = FALSE, ...)

Arguments

- ggmap: an object of class ggmap (from function get_map)
- fullpage: logical; should the map take up the entire viewport?
- base_layer: a ggplot(aes(...), ...) call; see examples
- maprange: logical for use with base_layer; should the map define the x and y limits?
- expand: should the map extend to the edge of the panel? used with base_layer and maprange=TRUE.

Value

- a ggplot object

Author(s)

- David Kahle <david.kahle@gmail.com>

See Also

- get_map, qmap

Examples

## Not run:
this is a deprecated function, use ggmap.

## End(Not run)
### ggmap_options

**Description**

`ggmap` stores options as a named list in R’s global options, i.e. `getOption("ggmap")`. It currently stores two such options, one for Google credentialing and one to suppress private API information in the URLs printed to the screen when web queries are placed. For both of those, see `register_google`.

**Usage**

```r
set_ggmap_option(...)

ggmap_credentials()
```

**Arguments**

```r
...
```

- `...`: a named listing of options to set

**Author(s)**

David Kahle <david.kahle@gmail.com>

**See Also**

- `register_google`

**Examples**

```r
getOption("ggmap")
```

----

### hadley

**Highly unofficial ggplot2 image**

**Description**

Highly unofficial ggplot2 image

**Author(s)**

Garrett Grolemund <grolemund@gmail.com>
### inset

**Add ggplot2 insets to a map**

**Description**

This is identical to `ggplot2::annotation_custom` for use with `ggmap`.

**Usage**

```
inset(grob, xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = Inf)
```

**Arguments**

- `grob`: grob to display
- `xmin`, `xmax`: x location (in data coordinates) giving horizontal location of raster
- `ymin`, `ymax`: y location (in data coordinates) giving vertical location of raster

**Details**

Most useful for adding tables, inset plots, and other grid-based decorations.

**Note**

`annotation_custom` expects the grob to fill the entire viewport defined by `xmin`, `xmax`, `ymin`, `ymax`. Grobs with a different (absolute) size will be center-justified in that region. Inf values can be used to fill the full plot panel.

### inset_raster

**Create a (ggplot2) raster layer**

**Description**

This is a special version of `ggplot2::annotation_raster` for use with `ggmap`. (It simply removes the requirement for cartesian coordinates.) The only difference between `inset_raster` and `inset_ggmap` is their arguments. `inset_ggmap` is simply a wrapper of `inset_raster` with `xmin`, ..., `ymax` arguments equal to the map's bounding box.

**Usage**

```
inset_raster(raster, xmin, xmax, ymin, ymax, interpolate = TRUE)
inset_ggmap(ggmap)
```
Arguments

raster | raster object to display
xmin, xmax | x location (in data coordinates) giving horizontal location of raster
ymin, ymax | y location (in data coordinates) giving vertical location of raster
interpolate | interpolate the raster? (i.e. antialiasing)
ggmap | a ggmap object, see ggplot

Examples

# see ?bb2bbox

legs2route

Convert a leg-structured route to a route-structured route

Description

Convert a leg-structured route to a route-structured route

Usage

legs2route(legsdf)

Arguments

legsdf | a legs-structured route, see route

See Also

geom_path in ggplot2

Examples

## Not run: requires Google API key, see ?register_google

(legs_df <- route("houston", "galveston"))
legs2route(legs_df)

(legs_df <- route(
  "marrs mclean science, baylor university",
  "220 south 3rd street, waco, tx 76701", # ninefa's
  alternatives = TRUE))
LonLat2XY

Convert a lon/lat coordinate to a tile coordinate

Description
Convert a lon/lat coordinate to a tile coordinate for a given zoom. Decimal tile coordinates (x, y) are reported.

Usage
LonLat2XY(lon_deg, lat_deg, zoom, xpix = 256, ypix = 256)
Arguments

- `lon_deg` longitude in degrees
- `lat_deg` latitude in degrees
- `zoom` zoom
- `xpix` width of tile in pixels
- `ypix` length of tile in pixels

Value

a data frame with columns X, Y, x, y

Author(s)

David Kahle <david.kahle@gmail.com>, based on function `LatLon2XY` by Markus Loecher, Sense Networks <markus@sensenetworks.com> in package `RgoogleMaps`

See Also

http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames

Examples

```r
## Not run:
gc <- geocode('baylor university')
LonLat2XY(gc$lon, gc$lat, 10)

## End(Not run)
```

make_bbox compute a bounding box

Description

Compute a bounding box for a given longitude / latitude collection.

Usage

```r
make_bbox(lon, lat, data, f = 0.05)
```
Arguments

- **lon**: longitude
- **lat**: latitude
- **data**: (optional) a data frame containing lon and lat as variables
- **f**: number specifying the fraction by which the range should be extended

Examples

```r
make_bbox(lon, lat, data = crime)

(lon <- sample(crime$lon, 10))
(lat <- sample(crime$lat, 10))
make_bbox(lon, lat)
make_bbox(lon, lat, f = .10) # bigger box
```

mapdist

Compute map distances using Google

Description

Compute map distances using Google’s Distance Matrix API. Note: To use Google’s Distance Matrix API, you must first enable the API in the Google Cloud Platform Console. See `register_google`.

Usage

```r
mapdist(from, to, mode = c("driving", "walking", "bicycling", "transit"),
         output = c("simple", "all"), urlonly = FALSE,
         override_limit = FALSE, ext = "com", inject = "", ...)

distQueryCheck()
```

Arguments

- **from**: name of origin addresses in a data frame (vector accepted)
- **to**: name of destination addresses in a data frame (vector accepted)
- **mode**: driving, bicycling, walking, or transit
- **output**: amount of output
- **urlonly**: return only the url?
- **override_limit**: override the current query count (.google_distance_query_times)
- **ext**: domain extension (e.g. "com", "co.nz")
- **inject**: character string to add to the url
- **...**: ...
Details

if parameters from and to are specified as geographic coordinates, they are reverse geocoded with revgeocode. note that the google maps api limits to 2500 element queries a day.

Value

a data frame (output="simple") or all of the geocoded information (output="all")

Author(s)

David Kahle <david.kahle@gmail.com>

See Also


Examples

## Not run: requires Google API key, see ?register_google

## basic usage

```r
mapdist(\text{"waco, texas"}, \text{"houston, texas"})
```

```r
from <- c(\text{"houston, texas"}, \text{"dallas"})
to <- c(\text{"waco, texas"})
mapdist(from, to)
```

```r
mapdist(from, to, mode = "bicycling")
```

```r
mapdist(from, to, mode = "walking")
```

```r
from <- c(\text{"1600 Amphitheatre Parkway, Mountain View, CA"},
\text{"3711 World Drive Walt Disney World, Orlando, Fl"})
to <- \text{"1600 Pennsylvania Avenue, Washington DC"}
mapdist(from, to)
```

```r
from <- \text{"st lukes hospital houston texas"}
to <- \text{"houston zoo, houston texas"}
mapdist(from, to, mode = "transit")
```

```r
from <- c(\text{"houston"}, \text{"houston"}, \text{"dallas"})
to <- c(\text{"waco, texas"}, \text{"san antonio", \text{"houston"}})
mapdist(from, to)
```

## geographic coordinates are accepted as well

```r
(wh <- as.numeric(geocode("the white house, dc")))
```
OSM_scale_lookup

Look up OpenStreetMap scale for a given zoom level.

Description

Look up OpenStreetMap scale for a given zoom level.

Usage

OSM_scale_lookup(zoom = 10)

Arguments

zoom google zoom

Details

The calculation of an appropriate OSM scale value for a given zoom level is a complicated task. For details, see http://wiki.openstreetmap.org/wiki/FAQ

Value

scale

Author(s)

David Kahle <david.kahle@gmail.com>

Examples

OSM_scale_lookup(zoom = 3)
OSM_scale_lookup(zoom = 10)

## Not run:
# these can take a long time or are prone to crashing
# if the osm server load is too high

# these maps are were the ones used to tailor fit the scale
# the zooms were fixed
ggmap(get_map(zoom = 3, source = 'osm', scale = 47500000), extent = "device")
ggmap(get_map(zoom = 4, source = 'osm', scale = 32500000), extent = "device")
ggmap(get_map(zoom = 5, source = 'osm', scale = 15000000), extent = "device")
print.ggmap

Description

Print a console description of a map

Usage

## S3 method for class 'ggmap'
print(x, ...)

Arguments

x           an object of class elicite
...
          additional parameters

Value

Invisibly returns x.
qmap

Examples

gmap_stamenmap(zoom = 9)

qmap

Quick map plot

Description
qmap is a wrapper for ggmap and get_map.

Usage
qmap(location = "houston", ...)

Arguments

location character; location of interest

... stuff to pass to ggmap and get_map.

Value

a ggplot object

Author(s)
David Kahle <david.kahle@gmail.com>

See Also
ggmap and get_map.

Examples

## Not run: some requires Google API key; heavy network/time load

location <- "marrs mclean science, waco, texas"
qmap(location)
qmap(location, zoom = 14)
qmap(location, zoom = 14, source = "osm")
qmap(location, zoom = 14, source = "osm", scale = 20000)
qmap(location, zoom = 14, maptype = "satellite")
qmap(location, zoom = 14, maptype = "hybrid")
qmap(location, zoom = 14, maptype = "toner", source = "stamen")
qmap(location, zoom = 14, maptype = "watercolor", source = "stamen")
qmplot

Quick map plot

Description

qmplot is the ggmap equivalent to the ggplot2 function qplot and allows for the quick plotting of maps with data/models/etc.

Usage

```r
qmplot(x, y, ..., data, zoom, source = "stamen",
      maptype = "toner-lite", extent = "device", legend = "right",
      padding = 0.02, force = FALSE, darken = c(0, "black"),
      mapcolor = "color", facets = NULL, margins = FALSE,
      geom = "auto", stat = list(NULL), position = list(NULL),
      xlim = c(NA, NA), ylim = c(NA, NA), main = NULL, f = 0.05,
      xlab = "Longitude", ylab = "Latitude")
```

Arguments

- **x**: longitude values
- **y**: latitude values
- **...**: other aesthetics passed for each layer
- **data**: data frame to use (optional). If not specified, will create one, extracting vectors from the current environment.
- **zoom**: map zoom, see `get_map`
- **source**: map source, see `get_map`
- **maptype**: map type, see `get_map`
- **extent**: how much of the plot should the map take up? "normal", "panel", or "device" (default)
legend "left", "right" (default), "bottom", "top", "bottomleft", "bottomright", "topleft", "topright", "none" (used with extent = "device")
padding distance from legend to corner of the plot (used with extent = "device")
force force new map (don’t use archived version)
darken vector of the form c(number, color), where number is in [0, 1] and color is a character string indicating the color of the darken. 0 indicates no darkening, 1 indicates a black-out.
mapcolor color ("color") or black-and-white ("bw")
facets faceting formula to use. Picks facet_wrap or facet_grid depending on whether the formula is one sided or two-sided
margins whether or not margins will be displayed
gem character vector specifying geom to use. defaults to "point"
stat character vector specifying statistics to use
position character vector giving position adjustment to use
xlim limits for x axis
ylim limits for y axis
main character vector or expression for plot title
f number specifying the fraction by which the range should be extended
xlab character vector or expression for x axis label
ylab character vector or expression for y axis label

Examples

```r
## Not run: # these are skipped to conserve R check time
qmplot(lon, lat, data = crime)

# only violent crimes
violent_crimes <- subset(crime, 
  offense != "auto theft" & 
  offense != "theft" & 
  offense != "burglary"
)

# rank violent crimes
violent_crimes$offense <- factor( 
  violent_crimes$offense, 
  levels = c("robbery", "aggravated assault", "rape", "murder")
)

# restrict to downtown
violent_crimes <- subset(violent_crimes, 
  -95.39681 <= lon & lon <= -95.34188 & 
  29.73631 <= lat & lat <= 29.78400
```
theme_set(theme_bw())

qmplot(lon, lat, data = violent_crimes, colour = offense, 
       size = I(3.5), alpha = I(.6), legend = "topleft")

qmplot(lon, lat, data = violent_crimes, geom = c("point", "density2d"))
qmplot(lon, lat, data = violent_crimes) + facet_wrap(~ offense)
qmplot(lon, lat, data = violent_crimes, extent = "panel") + facet_wrap(~ offense)
qmplot(lon, lat, data = violent_crimes, extent = "panel", colour = offense, darken = .4) + 
   facet_wrap(~ month)

qmplot(long, lat, xend = long + delta_long, 
       color = I("red"), yend = lat + delta_lat, data = seals, 
       geom = "segment", zoom = 5)
qmplot(long, lat, xend = long + delta_long, maptype = "watercolor", 
       yend = lat + delta_lat, data = seals, 
       geom = "segment", zoom = 6)
qmplot(long, lat, xend = long + delta_long, maptype = "terrain", 
       yend = lat + delta_lat, data = seals, 
       geom = "segment", zoom = 6)

qmplot(lon, lat, data = wind, size = I(.5), alpha = I(.5)) + 
   ggtitle("NOAA Wind Report Sites")

# thin down data set...
# s = seq(1, 227, 8)
thinwind <- subset(wind, 
   lon %in% unique(wind$lon)[s] &
   lat %in% unique(wind$lat)[s])

# for some reason adding arrows to the following plot bugs
theme_set(theme_bw(18))

qmplot(lon, lat, data = thinwind, geom = "tile", fill = spd, alpha = spd, 
       legend = "bottomleft") + 
   geom_leg(aes(xend = lon + delta_lon, yend = lat + delta_lat)) + 
   scale_fill_gradient2("Wind Speed\nand\nDirection", 
   low = "green", mid = scales::muted("green"), high = "red") + 
   scale_alpha("Wind Speed\nand\nDirection", range = c(.1, .75)) + 
   guides(fill = guide_legend(), alpha = guide_legend())
## kriging

# the below examples show kriging based on undeclared packages
# to better comply with CRAN's standards, we remove it from
# executing, but leave the code as a kind of case-study
# they also require the rgdal library

library(lattice)
library(sp)
library(rgdal)

# load in and format the meuse dataset (see bivand, pebesma, and gomez-rubio)
data(meuse)
coordinates(meuse) <- c("x", "y")
proj4string(meuse) <- CRS("+init=epsg:28992")
meuse <- spTransform(meuse, CRS("+proj=longlat +datum=WGS84"))

# plot
plot(meuse)

m <- data.frame(slot(meuse, "coords"), slot(meuse, "data"))
names(m)[1:2] <- c("lon", "lat")

qmplot(lon, lat, data = m)
qmplot(lon, lat, data = m, zoom = 14)

qmplot(lon, lat, data = m, size = zinc, zoom = 14, source = "google", maptype = "satellite",
       alpha = I(.75), color = I("green"),
       legend = "topleft", darken = .2
     ) + scale_size("Zinc (ppm)")

# load in the meuse.grid dataset (looking toward kriging)
library(gstat)
data(meuse.grid)
coordinates(meuse.grid) <- c("x", "y")
proj4string(meuse.grid) <- CRS("+init=epsg:28992")
meuse.grid <- spTransform(meuse.grid, CRS("+proj=longlat +datum=WGS84"))

# plot it
plot(meuse.grid)

mg <- data.frame(slot(meuse.grid, "coords"), slot(meuse.grid, "data"))
names(mg)[1:2] <- c("lon", "lat")
```r
qmplot(lon, lat, data = mg, shape = I(15), zoom = 14, legend = "topleft") +
  geom_point(aes(size = zinc), data = m, color = "green") +
  scale_size("Zinc (ppm)"
)

# interpolate at unobserved locations (i.e. at meuse.grid points)
# pre-define scale for consistency
scale <- scale_color_gradient("Predicted\nZinc (ppm)",
  low = "green", high = "red", lim = c(100, 1850)
)

data(idw) <- idw(log(zinc) ~ 1, meuse, meuse.grid, idp = 2.5)
mg$idw <- exp(slot(idw, "data")$var1.pred)
qmplot(lon, lat, data = mg, shape = I(15), color = idw,
  zoom = 14, legend = "topleft", alpha = I(.75), darken = .4
) + scale

# linear regression
lin <- krig(log(zinc) ~ 1, meuse, meuse.grid, degree = 1)
mg$lin <- exp(slot(lin, "data")$var1.pred)
qmplot(lon, lat, data = mg, shape = I(15), color = lin,
  zoom = 14, legend = "topleft", alpha = I(.75), darken = .4
) + scale

# trend surface analysis
tsa <- krig(log(zinc) ~ 1, meuse, meuse.grid, degree = 2)
mg$tsa <- exp(slot(tsa, "data")$var1.pred)
qmplot(lon, lat, data = mg, shape = I(15), color = tsa,
  zoom = 14, legend = "topleft", alpha = I(.75), darken = .4
) + scale

# ordinary kriging
gram <- variogram(log(zinc) ~ 1, meuse)  # plot(gram)
gramFit <- fit.variogram(gram, vgm(1, "Exp", .2, .1))
ordKrig <- krig(log(zinc) ~ 1, meuse, meuse.grid, gramFit)
mg$ordKrig <- exp(slot(ordKrig, "data")$var1.pred)
qmplot(lon, lat, data = mg, shape = I(15), color = ordKrig,
  zoom = 14, legend = "topleft", alpha = I(.75), darken = .4
)
```
register_google

) + scale

# universal kriging
vgram <- variogram(log(zinc) ~ 1, meuse) # plot(vgram)
vgramFit <- fit.variogram(vgram, vgm(1, "Exp", .2, .1))
univKriGe <- krig(log(zinc) ~ sqrt(dist), meuse, meuse.grid, vgramFit)
mg$univKriGe <- exp(slot(univKriGe, "data")$var1.pred)

qplot(lon, lat, data = mg, shape = I(15), color = univKriGe,
      zoom = 14, legend = "topleft", alpha = I(.75), darken = .4
) + scale

# adding observed data layer
qplot(lon, lat, data = mg, shape = I(15), color = univKriGe,
      zoom = 14, legend = "topleft", alpha = I(.75), darken = .4
) +
  geom_point(
    aes(x = lon, y = lat, size = zinc),
    data = m, shape = 1, color = "black"
  ) +
  scale +
  scale_size("Observed\nLog Zinc")

## End(Not run) # end dontrun

---

**register_google**  
**Register a Google API**

**Description**
This page contains documentation for tools related to enabling Google services in R. See the Details section of this file for background information.

**Usage**

showing_key()

ggmap_show_api_key()
ggmap_hide_api_key()

scrub_key(string, with = "xxx")

register_google(key, account_type, client, signature, second_limit, day_limit, write = FALSE)

## S3 method for class 'google_credentials'
print(x, ...)

google_key()

has_google_key()

has_google_account()

google_account()

google_client()

has_google_client()

google_signature()

has_google_signature()

google_second_limit()

google_day_limit()

Arguments

string a url string to be scrubbed. currently key, signature, and client keywords are scrubbed from the url and replace with the with argument

with a string to replace

key an api key

account_type "standard" or "premium"

client client code

signature signature code

second_limit query limit per second (default 50)

day_limit query limit per day (default 2500 for standard accounts, 100000 for premium accounts)

write if TRUE, stores the secrets provided in the .Renviron file

x a google credentials class object

... a dumped formal argument to the generic print method
register_google

Details

As of mid-2018, the Google Maps Platform requires a registered API key. While this alleviates previous burdens (e.g. query limits), it creates some challenges as well. The most immediate challenge for most R users is that ggmap functions that use Google’s services no longer function out of the box, since the user has to setup an account with Google, enable the relevant APIs, and then tell R about the user’s setup.

To obtain an API key and enable services, go to https://cloud.google.com/maps-platform/. This documentation shows you how to input the requisite information (e.g. your API key) into R, and it also shows you a few tools that can help you work with the credentialing.

To tell ggmap about your API key, use the `register_google()` function, e.g. `register_google(key = "mQkzTpiAYjQvXQ")` (that’s a fake key). This will set your API key for the current session, but if you restart R, you’ll need to do it again. You can set it permanently by setting `write = TRUE`, see the examples. If you set it permanently it will be stored in your .Renviron file, and that will be accessed by ggmap persistently across sessions.

Users should be aware that the API key, a string of jumbled characters/numbers/symbols, is a PRIVATE key - it uniquely identifies and authenticates you to Google’s services. If anyone gets your API key, they can use it to masquerade as you to Google and potentially use services that you have enabled. Since Google requires a valid credit card to use its online cloud services, this also means that anyone who obtains your key can potentially make charges to your card in the form of Google services. So be sure to not share your API key. To mitigate against users inadvertently sharing their keys, by default ggmap never displays a user’s key in messages displayed to the console.

Users should also be aware that ggmap has no mechanism with which to safeguard the private key once registered with R. That is to say, once you register your API key, any function R will have access to it. As a consequence, ggmap will not know if another function, potentially from a compromised package, accesses the key and uploads it to a third party. For this reason, when using ggmap we recommend a heightened sense of security and self-awareness: only use trusted packages, do not save API keys in script files, routinely cycle keys (regenerate new keys and retire old ones), etc. Google offers features to help in securing your API key, including things like limiting queries using that key to a particular IP address, as well as guidance on security best practices. See https://cloud.google.com/docs/authentication/api-keys#securing_an_api_key for details.

Author(s)

David Kahle <david.kahle@gmail.com>

See Also


Examples

# this sets your google map for this session
# register_google(key = "[your key]")
reversegeocode

# this sets your google map permanently
# register_google(key = "[your key]", write = TRUE)

has_google_key()
googkey()  
has_google_client()
has_google_signature()

geocode("waco, texas", urlonly = TRUE)
geocodeshowapikey()
geocodehideapikey()
geocode("waco, texas", urlonly = TRUE)

scrub_key("key=d_5iD")
scrub_key("key=d_5iD", "[your \1""")
scrub_key("signature=d_5iD")
scrub_key("client=a_5sS&signature=d_5iD")

---

Reverse geocode

Description

Reverse geocodes (looks up the address of) a longitude/latitude location using the Google Geocoding API. Note: To use Google’s Geocoding API, you must first enable the API in the Google Cloud Platform Console. See ?register_google.

Usage

revgeocode(location, output = c("address", "all"), force = FALSE,
urlonly = FALSE, override_limit = FALSE, ext = "com",
inject = "", ...)

Arguments

location a location in longitude/latitude format
output "address" or "all"
force force online query, even if cached (previously downloaded)
urlonly return only the url?
override_limit override the current query rate
ext domain extension (e.g. "com", "co.nz")
inject character string to add to the url
...
...
route

Value

a character(1) address or a list (the parsed json output from Google)

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

http://code.google.com/apis/maps/documentation/geocoding/

Examples

## Not run: requires Google API key, see register_google

## basic usage

```r
(gc <- as.numeric(geocode("the white house")))
revgeocode(gc)
str(revgeocode(gc, output = "all"), 3)
```

## End(Not run)

---

route  

Grab a route from Google

Description

Route two locations: determine a sequence of steps (legs) between two locations using the Google Directions API. Note: To use Google’s Directions API, you must first enable the API in the Google Cloud Platform Console. See register_google.

Usage

```r
route(from, to, mode = c("driving", "walking", "bicycling", "transit"),
structure = c("legs", "route"), output = c("simple", "all"),
alternatives = FALSE, units = "metric", urlonly = FALSE,
override_limit = FALSE, ext = "com", inject = ",", ...)
```

routeQueryCheck()
Arguments

from  
name of origin addresses in a data frame
to  
name of destination addresses in a data frame
mode  
driving, bicycling, walking, or transit
structure  
structure of output, "legs" or "route", see examples
output  
amount of output ("simple" or "all")
alternatives  
should more than one route be provided?
units  
"metric"
urlonly  
return only the url?
override_limit  
override the current query count
ext  
domain extension (e.g. "com", "co.nz")
inject  
character string to add to the url
...  
...

Value

a data frame (output="simple") or all of the geocoded information (output="all")

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

https://developers.google.com/maps/documentation/directions/, trek, legs2route, routeQueryCheck,
geom_leg, register_google

Examples

## Not run: requires Google API key, see ?register_google

## basic usage
################################

from <- "houston, texas"
to <- "waco, texas"

route(from, to, structure = "legs")  
route(from, to, structure = "route")  
route(from, to, alternatives = TRUE)

## comparison to trek
################################

(route_df <- route(from, to, structure = "route"))
theme_inset

(theme_inset <- trek(from, to, structure = "route"))

qmap("college station, texas", zoom = 8) +
  geom_path(
    aes(x = lon, y = lat), colour = "red",
    size = 1.5, alpha = .5,
    data = route_df, lineend = "round"
  ) +
  geom_path(
    aes(x = lon, y = lat), colour = "blue",
    size = 1.5, alpha = .5,
    data = trek_df, lineend = "round"
  )

qmap("college station, texas", zoom = 6) +
  geom_path(
    aes(x = lon, y = lat), colour = "red", size = 1.5,
    data = route_df, lineend = "round"
  )

## End(Not run)

theme_inset  Make a ggplot2 inset theme.

Description
theme_inset is a ggplot2 theme geared towards making inset plots.

Usage
theme_inset(base_size = 12)

Arguments
base_size  base size, not used.

Value
a ggplot2 theme (i.e., a list of class options).

Author(s)
David Kahle <david.kahle@gmail.com>
Examples

```r
library(ggplot2)

## Not run:

n <- 50
df <- expand.grid(x = 1:n, y = 1:n)[sample(n^2, .5*n^2),]
qplot(x, y, data = df, geom = 'tile')
qplot(x, y, data = df, geom = 'tile') + theme_nothing()

qplot(1:10, 1:10) +
  annotation_custom(
    grob = ggplotGrob(qplot(1:10, 1:10)),
    8, Inf, -Inf, 2
  )

qplot(1:10, 1:10) +
  annotation_custom(
    grob = ggplotGrob(qplot(1:10, 1:10) + theme_nothing()),
    8, Inf, -Inf, 2
  )

qplot(1:10, 1:10) +
  annotation_custom(
    grob = ggplotGrob(qplot(1:10, 1:10) + theme_inset()),
    8, Inf, -Inf, 2
  )

## End(Not run)
```

---

**theme_nothing**

*Make a blank ggplot2 theme.*

**Description**

theme_nothing simply strips all thematic element in ggplot2.

**Usage**

```r
theme_nothing(base_size = 12, legend = FALSE)
```

**Arguments**

- **base_size** base size, not used.
- **legend** should the legend be included?
Value

a ggplot2 theme (i.e., a list of class options).

Author(s)

David Kahle <david.kahle@gmail.com>

Examples

```r
# no legend example
n <- 50
df <- expand.grid(x = 1:n, y = 1:n)[sample(n^2, .5*n^2),]
p <- ggplot(df, aes(x, y)) + geom_raster()
p
p + theme_nothing()
p + theme_nothing(legend = TRUE)  # no difference
p +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0)) +
  theme_nothing()

# legend example
df$class <- factor(sample(0:1, .5*n^2, replace = TRUE))
p <- ggplot(df, aes(x, y)) + geom_raster(aes(fill = class))
p
p + theme_nothing()
p + theme_nothing(legend = TRUE)

p <- p +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0))
p
p + theme_nothing()
p + theme_nothing(legend = TRUE)
```

Description

Sequence treks (latitude-longitude sequences following ordinary paths, e.g. roads) between two locations using the Google Directions API. Note: To use Google's Directions API, you must first enable the API in the Google Cloud Platform Console. See ?register_google.
Usage

`trek(from, to, mode = c("driving", "walking", "bicycling", "transit"),
output = c("simple", "all"), alternatives = FALSE,
units = "metric", urlonly = FALSE, override_limit = FALSE,
ext = "com", inject = "", ...)`

Arguments

- `from`: name of origin addresses in a data frame
- `to`: name of destination addresses in a data frame
- `mode`: driving, bicycling, walking, or transit
- `output`: amount of output ("simple" or "all")
- `alternatives`: should more than one route be provided?
- `units`: "metric"
- `urlonly`: return only the url?
- `override_limit`: override the current query count
- `ext`: domain extension (e.g. "com", "co.nz")
- `inject`: character string to add to the url
- `...`: ...

Value

A tibble

Author(s)

David Kahle <david.kahle@gmail.com> with the key decoding algorithm due to Stack Overflow user akhmed

See Also


Examples

```r
## Not run: requires Google API key, see ?register_google

## basic usage

# "houston, texas"
# "waco, texas"

(route_df <- route(from, to, structure = "route"))
```
(trek_df <- trek(from, to, structure = "route"))

qmap("college station, texas", zoom = 8) +
  geom_path(
    aes(x = lon, y = lat), colour = "red",
    size = 1.5, alpha = .5,
    data = route_df, lineend = "round"
  ) +
  geom_path(
    aes(x = lon, y = lat), colour = "blue",
    size = 1.5, alpha = .5,
    data = trek_df, lineend = "round"
  )

from <- "rice university houston texas"
to <- "1001 Bissonnet St, Houston, TX 77005"
trek_df <- trek(from, to)
qmplot(lon, lat, data = trek_df, geom = "path", maptype = "terrain",
       color = I("red"), size = I(2), alpha = I(.5))

trek_df <- trek(from, to, mode = "walking")
qmplot(lon, lat, data = trek_df, geom = "path", maptype = "terrain",
       color = I("red"), size = I(2), alpha = I(.5))

trek_df <- trek(from, to, mode = "transit")
qmplot(lon, lat, data = trek_df, geom = "path", maptype = "terrain",
       color = I("red"), size = I(2), alpha = I(.5))

## neat faceting example

from <- "houston, texas"; to <- "waco, texas"
trek_df <- trek(from, to, alternatives = TRUE)
qmplot(lon, lat, data = trek_df, geom = "path",
       color = route, size = I(2), maptype = "terrain",
       alpha = I(.5))
qmplot(lon, lat, data = trek_df, geom = "path",
       color = route, size = I(2), maptype = "terrain",
       zoom = 8)
+ facet_grid(. ~ route)
# Wind data from Hurricane Ike

## Description

Wind data from Hurricane Ike

## Details


## Author(s)

Atlantic Oceanographic and Meteorological Laboratory (AOML), a division of the National Oceanic and Atmospheric Administration (NOAA)

## References


---

## XY2LonLat

### Convert a tile coordinate to a lon/lat coordinate

## Description

Convert a tile coordinate to a lon/lat coordinate for a given zoom. Decimal tile coordinates are accepted.

## Usage

XY2LonLat(X, Y, zoom, x = 0, y = 0, xpix = 255, ypix = 255)

### Arguments

- **X**: horizontal map-tile coordinate (0 is map-left)
- **Y**: vertical map-tile coordinate (0 is map-top)
- **zoom**: zoom
- **x**: within tile x (0 is tile-left)
- **y**: within tile y (0 it tile-top)
- **xpix**: width of tile in pixels
- **ypix**: length of tile in pixels
Zip code data for the Greater Houston Metropolitan Area from the 2000 census

Description
Zip code data for the Greater Houston Metropolitan Area from the 2000 census

Author(s)
U.S. Census Bureau, Geography Division, Cartographic Products Management Branch

References
Downloaded from http://www.census.gov/geo/www/cob/z52000.html (now defunct).
Index

bb2bbox, 2, 4
borders(), 8
calc_zoom, 4
crime, 5
distQueryCheck (mapdist), 41
facet_grid, 47
facet_wrap, 47

gecode, 5, 13
gecodeQueryCheck (geocode), 5
gem_lem, 7, 56
get_cloudmademap, 11
get_googlemap, 12, 16
get_map, 15, 26, 27, 35, 38, 45, 46
get_navermap, 17
get_openstreetmap, 16, 19
get_stamen_tile_download_fail_log
  (get_stamenmap), 20
get_stamenmap, 20
GetMap, 17
ggimage, 23
gglocator, 25
ggmap, 12, 14, 17, 19–21, 26, 45
ggmap-defunct, 34
ggmap-package (ggmap), 26
ggmap_credentials (ggmap_options), 36
ggmap_hide_api_key (register_google), 51
ggmap_options, 36
ggmap_show_api_key (register_google), 51
ggmapPlot, 35
goog_le_account (register_google), 51
goog_le_client (register_google), 51
goog_le_day_limit (register_google), 51
goog_le_key (register_google), 51
goog_le_second_limit (register_google), 51
goog_le_signature (register_google), 51
hadley, 36
has_google_account (register_google), 51
has_google_client (register_google), 51
has_google_key (register_google), 51
has_google_signature (register_google), 51
inset, 37
inset_ggmap (inset_raster), 37
inset_raster, 37
legs2route, 38, 56
LonLat2XY, 39
make_bbox, 4, 40
mapdist, 41
mutate_geocode (geocode), 5
OSM_scale_lookup, 19, 43
package-ggmap (ggmap), 26
print.ggmap, 44
print.google_credentials
  (register_google), 51
qmap, 27, 35, 45
qmplot, 46
register_google, 14, 36, 51, 56, 60
retry_stamen_map_download
  (get_stamenmap), 20
revgeocode, 54
route, 8, 38, 55, 60
routeQueryCheck, 56, 60
routeQueryCheck (route), 55
scrub_key (register_google), 51
set_ggmap_option (ggmap_options), 36
showing_key (register_google), 51
tempfile, 11, 13, 16, 18, 19
INDEX

theme_inset, 57
theme_nothing, 58
trek, 56, 59

wind, 62

XY2LonLat, 62

zips, 63