Package ‘ggmap’

August 29, 2016

Version 2.6.1
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 (>= 2.14.0), ggplot2 (>= 2.0.0)
Imports proto, RgoogleMaps, png, plyr, reshape2, rjson, mapproj, jpeg,
geosphere, digest, scales
Suggests MASS, stringr, hexbin, dplyr
License GPL-2
LazyData true
RoxygenNote 5.0.1
NeedsCompilation no
Author David Kahle [aut, cre],
Hadley Wickham [aut]
Maintainer David Kahle <david.kahle@gmail.com>
Repository CRAN
Date/Publication 2016-01-23 00:37:27

R topics documented:

bb2bbox ................................................................. 2
calc_zoom ............................................................ 3
crime ................................................................. 4
distQueryCheck ...................................................... 5
geocode ............................................................... 5
g geom_leg .......................................................... 7
bb2bbox

**Convert a bb specification to a bbox specification**

**Description**

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)
calc_zoom

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`, `bbRbbox`

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)
```

---

crime  

**Crime data**

Description

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

Author(s)

Houston Police Department, City of Houston
distQueryCheck

References

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

distQueryCheck Check Google Maps Distance Matrix API query limit

Description

Check Google Maps Distance Matrix API query limit

Usage

distQueryCheck()

Value

a data frame

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

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

Examples

distQueryCheck()

gencode Geocode

Description

Geocodes a location (find latitude and longitude) using either (1) the Data Science Toolkit (http://www.datasciencetoolkit.org/about) or (2) Google Maps. Note that when using Google you are agreeing to the Google Maps API Terms of Service at https://developers.google.com/maps/terms.

Usage

gencode(location, output = c("latlon", "latlon", "more", "all"),
source = c("google", "dsk"), messaging = FALSE, force = ifelse(source ==
"dsk", FALSE, TRUE), sensor = FALSE, override_limit = FALSE,
client = "", signature = "", nameType = c("long", "short"), data)

gencodeQueryCheck(userType = "free")
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  "dsk" for Data Science Toolkit or "google" for Google
messaging  turn messaging on/off
force  force online query, even if previously downloaded
sensor  whether or not the geocoding request comes from a device with a location sensor
override_limit  override the current query count (GoogleGeocodeQueryCount)
client  client ID for business users, see https://developers.google.com/maps/documentation/business/webservices/auth
signature  signature for business users, see https://developers.google.com/maps/documentation/business/webservices/auth
nameType  in some cases, Google returns both a long name and a short name. this parameter allows the user to specify which to grab.
data  deprecated in 2.5, use mutate_geocode
usertype  User type, "free" or "business"

Details

Note that the Google Maps api limits to 2500 queries a day. Use geocodeQueryCheck to determine how many queries remain.

Value

If output is "latlon", "latlona", or "more", a data frame. If all, a list.

Author(s)

David Kahle <david.kahle@gmail.com>

See Also


Examples

## Not run:  # Server response can be slow; this cuts down check time.

# types of input
geocode("houston texas")
geocode("baylor university") # see known issues below
geocode("1600 pennsylvania avenue, washington dc")
geocode("the white house")
geocode(c("baylor university", "salvation army waco"))
# types of output
geocode("houston texas", output = "latlon")
geocode("houston texas", output = "more")
geocode("Baylor University", output = "more")
str(geocode("Baylor University", output = "all"))

# see how many requests we have left with google
geocodeQueryCheck()
geocode("one bear place, waco, texas")
geocode("houston texas", force = TRUE)

# known issues :
# (1) source = "dsk" can't reliably geocode colloquial place names
geocode("city hall houston")
geocode("rice university")

## End(Not run)

---

### geom_leg

**Single line segments with rounded ends**

**Description**

This is ggplot2’s segment with rounded ends. It’s mainly included in ggmap for historical reasons.

Single line segments with rounded ends

**Usage**

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

```r
gemm_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), removes missing values with a warning. If TRUE silently removes missing values.
- **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.
- **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 ggmaps package. only designed for mercator projection.

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

See Also

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

Examples

```r
## 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
))
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
# ggmap(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: # removed for R CMD check speed
```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 = 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
# ggmmap(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) +
```
### Description

`get_cloudmademap` accesses a tile server for Stamen Maps and downloads/stitches map tiles/formats into 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

```r
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 = "ggtemp.png", 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)
- **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

## Not run: # in what follows, enter your own api key

api_key <- '<your api key here>'

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

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

## End(Not run)

---

get_googlemap Get a Google Map.

Description

googlemap accesses the Google Static Maps 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://developers.google.com/maps/terms.

Usage

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", sensor = FALSE, messaging = FALSE, urlonly = FALSE, filename = "ggmapTemp", color = c("color", "bw"), force = FALSE, where = tempdir(), archiving = FALSE, key = "", region, markers, path, visible, style, ...)
Arguments

center
the center of the map. Either a longitude/latitude numeric vector, a string address
(note that the latter uses geocode with source = "google").

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

sensor
specifies whether the application requesting the static map is using a sensor to
determine the user’s location

messaging
turn messaging on/off

urlonly
return url only

filename
destination file for download (file extension added according to format)

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 ap-
proved uses listed in the Google Maps API Terms of Service : http://developers.google.com/maps/terms.

date
an api_key for business users

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 char-
acter 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

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

https://developers.google.com/maps/documentation/staticmaps/, ggmap

Examples

```r
## Not run:  # to diminish run check time

googlemap(urlonly = TRUE)
googlemap()

# 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)
    ), digits = r)
) df <- d(n=50, r=3, a=.3)
map <- googlemap(markers = df, path = df, , scale = 2)
googlemap(map)
googlemap(map, extent = "device") +
  geom_point(aes(x = lon, y = lat), data = df, size = 3, colour = "black") +
  geom_path(aes(x = lon, y = lat), data = df)

googlemap(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)
googlemap(googlemap(center, scale = 1), extent = "panel") # pixelated
googlemap(googlemap(center, scale = 2), extent = "panel") # fine

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

# style
map <- googlemap(style = c(feature = "all", element = "labels", visibility = "off"))
```
get_map

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**

```r
```

**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). OpenStreetMap 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"), or CloudMade Maps ("cloudmade").
- **force**: force new map (don’t use archived version).
- **messaging**: turn messaging on/off.
get_navermap

Description

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 at http://dev.naver.com/openapi/apis/map/staticmap.

urlonly

return url only

filename

destination file for download (file extension added according to format)

crop

(stamen and cloudmade maps) crop tiles to bounding box

color

color ("color") or black-and-white ("bw")

language

language for google maps

api_key

an api key for cloudmade 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

map <- get_map()

map

str(map)

ggmap(map)

## Not run:
# not run by check to reduce time; also,
# osm may error due to server overload

(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

Usage

get_navermap(center = c(lon = 126.9849208, lat = 37.5664519), zoom = 4, size = c(640, 640), format = c("png", "jpeg", "jpg"),
    baselayer = c("default", "satellite"), color = c("color", "bw"),
    overlayers = c("anno_satellite", "bicycle", "roadview", "traffic"), markers,
    key, uri, filename = "ggmapTemp", messaging = FALSE, urlonly = FALSE,
    force = FALSE, where = tempdir(), archiving = TRUE, ...)

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)
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 ")"
archiving use archived maps. note: by changing to TRUE you agree to abide by any of the rules governing caching naver maps
... ...

Author(s)

Heewon Jeon <madjakarta@gmail.com>

See Also

http://dev.naver.com/openapi/apis/map/staticmap/, ggmap
**get_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/](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

```r
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 = "ggmapTemp", 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](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/](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)
- **color**
  - color or black-and-white
- **...**
  - ...
get_stamenmap

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.

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

## Not run:
# osm servers get overloaded, which can result in
# erroneous failed checks

osm <- get_openstreetmap(urlonly = TRUE)
ggmap(osm)

## End(Not run)

---

**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, e.g. http://tile.stamen.com/terrain/#/4/30.28/-87.21

**Usage**

get_stamenmap(bbox = c(left = -95.80204, bottom = 29.38048, right = -94.92313, top = 30.14344), zoom = 10, maptype = 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(), ...)

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
messaging turn messaging on/off
urlonly return url only
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 "/")
...
...

Value

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

See Also

http://maps.stamen.com/#watercolor.ggmap

Examples

## Not run: # to diminish run check time
gc <- geocode("baylor university")
google <- get_googlemap("baylor university", zoom = 15)
ggmap(google) +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

bbox <- c(left = -97.132, bottom = 31.536, right = -97.105, top = 31.560)
ggmap(get_stamenmap(bbox, zoom = 13))
ggmap(get_stamenmap(bbox, zoom = 14))
ggmap(get_stamenmap(bbox, zoom = 15))
# ggmap(get_stamenmap(bbox, zoom = 16))
# ggmap(get_stamenmap(bbox, zoom = 17))

# note that the osm code may not run due to overloaded # servers.
# various maptypes are available. bump it up to zoom = 15 for better resolution.
```r
ggmap(get_stamenmap(bbox, maptype = "terrain", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "terrain-background", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "terrain-labels", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "terrain-lines", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "toner", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "toner-2010", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "toner-2011", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "toner-background", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "toner-hybrid", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "toner-labels", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "toner-lines", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "toner-lite", zoom = 14))
ggmap(get_stamenmap(bbox, maptype = "watercolor", zoom = 14))

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

stamen <- get_stamenmap(bbox, zoom = 15)
ggmap(stamen) +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

stamen <- get_stamenmap(bbox, zoom = 15, crop = FALSE)
ggmap(stamen) +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

osm <- get_openstreetmap(bbox, scale = OSM_scale_lookup(15))
ggmap(osm) +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

ggmap(get_stamenmap(bbox, zoom = 15, maptype = "watercolor") +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

ggmap(get_stamenmap(bbox, zoom = 15, maptype = "toner") +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 2)

# here's an interesting example:
us <- c(left = -125, bottom = 25.75, right = -67, top = 49)
map <- get_stamenmap(us, zoom = 5, maptype = "toner-labels")
ggmap(map)

# 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)

# accuracy check - statue of liberty
# see https://github.com/dkahle/ggmap/issues/32

gc <- geocode("statue of liberty")
googMapZ10 <- get_googlemap(center = as.numeric(gc))
bbZ10 <- attr(googMapZ10, "bb")
stamMapZ10 <- get_stamenmap(bb2bbox(bbZ10))

ggmap(googMapZ10) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )

# using a higher zoom

# accuracy check - statue of liberty
# see https://github.com/dkahle/ggmap/issues/32

gc <- geocode("statue of liberty")
googMapZ10 <- get_googlemap(center = as.numeric(gc))
bbZ10 <- attr(googMapZ10, "bb")
stamMapZ10 <- get_stamenmap(bb2bbox(bbZ10))

ggmap(googMapZ10) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )

# using a higher zoom

# accuracy check - statue of liberty
# see https://github.com/dkahle/ggmap/issues/32

gc <- geocode("statue of liberty")
googMapZ10 <- get_googlemap(center = as.numeric(gc), zoom = 15)
bbZ15 <- attr(googMapZ15, "bb")
stamMapZ15 <- get_stamenmap(bb2bbox(bbZ15),
  zoom = calc_zoom(bb2bbox(bbZ15))

ggmap(googMapZ15) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )

# accuracy check - statue of liberty
# see https://github.com/dkahle/ggmap/issues/32

gc <- geocode("statue of liberty")
googMapZ10 <- get_googlemap(center = as.numeric(gc), zoom = 15)
bbZ15 <- attr(googMapZ15, "bb")
stamMapZ15 <- get_stamenmap(bb2bbox(bbZ15),
  zoom = calc_zoom(bb2bbox(bbZ15))

ggmap(stamMapZ15) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )
# using a lower zoom
googMapZ5 <- get_googlemap(center = as.numeric(gc), zoom = 4)
bbZ5 <- attr(googMapZ5, "bb")
stamMapZ5 <- get_stamenmap(bb2bbox(bbZ5),
          zoom = calc_zoom(bb2bbox(bbZ5))
)

ggmap(googMapZ5) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )

ggmap(stamMapZ5) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )

stamMapZ5uncropped <- get_stamenmap(bb2bbox(bbZ5),
                                    zoom = calc_zoom(bb2bbox(bbZ5)),
                                    crop = FALSE)

ggmap(stamMapZ5uncropped) +
  geom_point(
    aes(x = lon, y = lat),
    data = gc, colour = "red", size = 3
  )

qmap(location = c(lon = -74.0445, lat = 40.68925),
      zoom = 16, source = "stamen") +
  geom_point(aes(x = lon, y = lat), data = gc, colour = "red", size = 3)

## End(Not run) # end dontrun

---

### 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, imagenatrix, 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 \([0, \text{ncol(mat)}-1] \times [0, \text{nrow(mat)}-1]\) (F) or \([0.1] \times [0.1]\) (T)

Value

- a ggplot object

Author(s)

David Kahle <david.kahle@gmail.com>

Examples

```r
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, coord_equal = FALSE)

```r
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 ggplots.

Description

Locator for ggplots. (Note : only accurate when extent = "normal" when using ggmap.)
Usage

```
gglocator(n = 1, message = FALSE, xexpand = c(0.05, 0),
         yexpand = c(0.05, 0))
```

Arguments

- `n`: number of points to locate.
- `message`: turn messaging from grid.ls on/off
- `xexpand`: expand argument in scale_x_continuous
- `yexpand`: expand argument in scale_y_continuous

Value

A data frame with columns according to the x and y aesthetics

Author(s)

Tyler Rinker with help from Baptiste Auguie and StackOverflow user DWin with additions and
canning by David Kahle <david.kahle@gmail.com>. Updated by @Nikolai-Hlubek

Examples

```
if(interactive()){
  # only run for interactive sessions

  df <- expand.grid(x = 0:-5, y = 0:-5)
  (p <- qplot(x, y, data = df) +
   annotate(geom = 'point', x = -2, y = -2, colour = 'red'))
  ggl()ter()
  p +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0))
  ggl(1, xexpand = c(0,0), yexpand = c(0,0))
}
```
ggmap

Plot a ggmap object

Description

ggmap plots the raster object produced by get_map.

Usage

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

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.

Value

a ggplot object

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

`get_map`, `qmap`
Examples

```r
## 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缺点
    cbind(MASS::mvtnorm(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())

    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)

# 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::mvnrnorm(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) + geom_point(aes(colour = class), data = points)

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, darken = 0,
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("baylor university", zoom = 15, maptype = "satellite")

ggmap(baylor)

# use gglocator to find lon/lat's of interest
(clicks <- clicks <- gglocator(2))
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,
colour=I("red"), arrow = arrow(length=unit(0.3,"cm")), size = 1.5) +
annotate("rect", xmin=-97.122, ymin=31.5439, xmax=-97.1050, ymax=31.5452,
fill = I("white"), alpha = I(3/4)) +
annotate("text", x=-97.113, y=31.5445, label = "Department of Statistical Science",
colour = I("red"), size = 3.5) +
labs(x = "Longitude", y = "Latitude") + ggtitle("Baylor University")
baylor <- get_map("baylor university", zoom = 16, maptype = "satellite")

ggmap(baylor, extent = "panel") +
  annotate("segment", x=97.1175, xend=97.1188, y=31.5449, yend=31.5485,
            colour=I("red"), arrow = arrow(length=unit(0.4, "cm")), size = 1.5) +
  annotate("rect", xmin=97.122, ymin=31.5441, xmax=97.113, ymax=31.5449,
           fill = I("white"), alpha = 1/3) +
  annotate("text", x=-97.1175, y=31.5445, label = "Department of Statistical Science",
            colour = I("red"), size = 4)

# a shapefile like layer

data(zips)
ggmap(get_map(maptype = "satellite", zoom = 8), extent = "device") +
  geom_polygon(aes(x = lon, y = lat, group = plotOrder),
               data = zips, colour = NA, fill = "red", alpha = .2) +
  geom_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[1, c("area", "perimeter", "zip", "lonCent", "latCent")]
})
ggmap(get_map(maptype = "satellite", zoom = 9),
      extent = "device", legend = "none", darken = .5) +
  geom_text(aes(x = lonCent, y = latCent, label = zip, size = area),
            data = zipsLabels, colour = I("red")) +
  scale_size(range = c(1.5,6))

qplot(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

```r
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...

```r
HoustonMap +
stat_bin2d(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")
)
```

# ... with hexagonal bins

```r
HoustonMap +
stat_binhex(aes(x = lon, y = lat, colour = offense, fill = offense),
   size = .5, binwidth = c(.00225,.00225), 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)

```r
houston <- get_map("houston", zoom = 14)
HoustonMap <- ggmap(houston, extent = "device", legend = "topleft")
```

# a filled contour plot...

```r
HoustonMap +
```
```r
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)

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 = attr(houston,"bb")$ll.lon +
        (7/10) * (attr(houston,"bb")$ur.lon - attr(houston,"bb")$ll.lon),
    xmax = Inf,
    ymin = -Inf,
    ymax = attr(houston,"bb")$ll.lat +
        (3/10) * (attr(houston,"bb")$ur.lat - attr(houston,"bb")$ll.lat)
)

## 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)
)
```
```
ggmap

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

```r
df <- data.frame(
  x = rnorm(1000, -95.36258, .2),
  y = rnorm(1000, 29.76196, .2)
)

# 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)

gmplot(x, y, data = df)
mqplot(x, y, data = df, facets = ~ year)

## neat faceting examples

```r

## 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 = "\n"
)

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())
ggmap(get_map(), darken = .5)
ggmap(get_map(), darken = c(.5,"white"))
ggmap(get_map(), darken = c(.5,"red")) # silly, but possible
```

## End(Not run)
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

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

## End(Not run)
```
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 get_map

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

gem_path in ggplot2

Examples

## Not run:

```r
(legs_df <- route("houston","galveston"))
legs2route(legs_df)
(legs_df <- route(
  "marrs mclean science, baylor university",
  "220 south 3rd street, waco, tx 76701", # ninfas
  alternatives = TRUE))
legs2route(legs_df)
```

from <- "houston, texas"
to <- "waco, texas"
legs_df <- route(from, to)

```r
qmap("college station, texas", zoom = 8) +
  geom_segment(
    aes(x = startLon, y = startLat, xend = endLon, yend = endLat),
    colour = "red", size = 1.5, data = legs_df
  )
  # notice boxy ends
qmap("college station, texas", zoom = 8) +
  geom_leg(
    aes(x = startLon, y = startLat, xend = endLon, yend = endLat),
    colour = "red", size = 1.5, data = legs_df
  )
  # notice overshooting ends
route_df <- legs2route(legs_df)
qmap("college station, texas", zoom = 8) +
  geom_path(
    aes(x = lon, y = lat),
    colour = "red", size = 1.5, data = route_df, lineend = "round"
  )
```

## End(Not run)
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

## 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 Maps. Note that in most cases by using this function you are agreeing to the Google Maps API Terms of Service at https://developers.google.com/maps/terms.

**Usage**

```r
mapdist(from, to, mode = c("driving", "walking", "bicycling"),
  output = c("simple", "all"), messaging = FALSE, sensor = FALSE,
  language = "en-EN", override_limit = FALSE)
```
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, or walking
output: amount of output
messaging: turn messaging on/off
sensor: whether or not the geocoding request comes from a device with a location sensor
language: language
override_limit: override the current query count (.GoogleDistQueryCount)

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

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

Examples

```r
## Not run: online queries draw R CMD check times

mapdist("waco, texas", "houston, texas")

from <- c("houston, texas", "dallas")
to <- "waco, texas"
mapdist(from, to)
mapdist(from, to, mode = "bicycling")
mapdist(from, to, mode = "walking")

from <- c("houston", "houston", "dallas")
to <- c("waco, texas", "san antonio", "houston")
mapdist(from, to)

# geographic coordinates are accepted as well
(wh <- as.numeric(geocode("the white house, dc")))
(lm <- as.numeric(geocode("lincoln memorial washington dc")))
```
mutate_geocode

Geocode a dataset

Description

mutate_geocode geocodes a data frame and appends the new information to the data frame provided.

Usage

mutate_geocode(data, location, ...)

Arguments

data a data frame

location a character string specifying a location of interest (e.g. "Baylor University")

... arguments to pass to geocode

Value

data with geocoded information appended as columns

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

geocode

Examples

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

## Not run: # Server response can be slow; this cuts down check time.
mutate_geocode(df, address)
library(dplyr)
  df %>% mutate_geocode(address)

## End(Not run)

### 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](http://wiki.openstreetmap.org/wiki/FAQ) or [http://almien.co.uk/OSM/Tools/Scale/](http://almien.co.uk/OSM/Tools/Scale/).

**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")
print.ggmap

Print a map

Description
Print a console description of a map

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

Arguments
x an object of class elicit
...
additional parameters

Value
Invisible string of the printed object.
qmap

Examples

```r
get_map()
ggmap(get_map())
```

---

qmap

Quick map plot

Description

qmap is a wrapper for \texttt{ggmap} and \texttt{get_map}.

Usage

```r
qmap(location = \text{"houston"}, \ldots)
```

Arguments

- **location** character; location of interest
- \ldots stuff to pass to \texttt{ggmap} and \texttt{get_map}.

Value

a ggplot object

Author(s)

David Kahle <david.kahle@gmail.com>

See Also

- \texttt{ggmap} and \texttt{get_map}.

Examples

```r
## Not run:
# these examples have been excluded for checking efficiency
qmap(location = "baylor university")
qmap(location = "baylor university", zoom = 14)
qmap(location = "baylor university", zoom = 14, source = "osm")
qmap(location = "baylor university", zoom = 14, source = "osm", scale = 20000)
qmap(location = "baylor university", zoom = 14, maptype = "satellite")
qmap(location = "baylor university", zoom = 14, maptype = "hybrid")
qmap(location = "baylor university", zoom = 14, maptype = "toner", source = "stamen")
qmap(location = "baylor university", zoom = 14, maptype = "watercolor", source = "stamen")
```
qmplot

qmap(location = "baylor university", zoom = 14, maaptype = "terrain-background", source = "stamen")
qmap(location = "baylor university", zoom = 14, maaptype = "toner-lite", source = "stamen")

api_key <- "your api key here"
qmap(location = "baylor university", zoom = 14, maaptype = 15434,
       source = "cloudmade", api_key = api_key)

wh <- geocode("the white house")
qmap("the white house", maprange = TRUE,
     base_layer = ggplot(aes(x=lon, y=lat), data = wh)) +
geom_point()

## End(Not run)

### 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", maaptype = "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`
- `maaptype` 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

geom
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

# 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


```r

# 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(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(rgstat)
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")
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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)"

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)
)

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

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

vgramFit <- fit.variogram(vgram, vgm(1, "Exp", .2, 1))

univKrig <- krige(log(zinc) ~ sqrt(dist), meuse, meuse.grid, vgramFit)

mg$univKrig <- exp(slot(univKrig, "data")$var1.pred)

qmapplot(lon, lat, data = mg, shape = I(15), color = univKrig, zoom = 14, legend = "topleft", alpha = I(.75), darken = .4)

# adding observed data layer
qmapplot(lon, lat, data = mg, shape = I(15), color = univKrig, 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

---

**revgeocode**

**Reverse geocode**

**Description**

Revgeocode reverse geocodes a longitude/latitude location using Google Maps. Note that in most cases by using this function you are agreeing to the Google Maps API Terms of Service at https://developers.google.com/maps/terms.

**Usage**

```r
revgeocode(location, output = c("address", "more", "all"),
  messaging = FALSE, sensor = FALSE, override_limit = FALSE,
  client = "", signature = "")
```

**Arguments**

- `location`: a location in longitude/latitude format
route

Grab a route from Google

Description

Grab a route from Google. Note that in most cases by using this function you are agreeing to the Google Maps API Terms of Service at https://developers.google.com/maps/terms.
route

Usage

route(from, to, mode = c("driving", "walking", "bicycling", "transit"),
structure = c("legs", "route"), output = c("simple", "all"),
alternatives = FALSE, messaging = FALSE, sensor = FALSE,
override_limit = FALSE)

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
structure structure of output, see examples
output amount of output
alternatives should more than one route be provided?
messaging turn messaging on/off
sensor whether or not the geocoding request comes from a device with a location sensor
override_limit override the current query count (.GoogleRouteQueryCount)

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/, legs2route, routeQueryCheck, geom_leg

Examples

## Not run: # to cut down on check time

from <- "houston, texas"
to <- "waco, texas"
route_df <- route(from, to, structure = "route")
qumap("college station, texas", zoom = 8) +
  geom_path(
    aes(x = lon, y = lat), colour = "red", size = 1.5,
    data = route_df, lineend = "round"
  )
qumap("college station, texas", zoom = 6) +
  geom_path(
    aes(x = lon, y = lat), colour = "red", size = 1.5,
data = route_df, lineend = "round"
)

routeQueryCheck()

## End(Not run)

---

routeQueryCheck  Check Google Maps Directions API query limit

**Description**

Check Google Maps Directions API query limit

**Usage**

routeQueryCheck()

**Value**

a data frame

**Author(s)**

David Kahle <david.kahle@gmail.com>

**See Also**

https://developers.google.com/maps/documentation/directions/

**Examples**

## Not run:
routeQueryCheck()

## 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

library(ggplot2)
## Not run:

n <- 50
df <- expand.grid(x = 1:n, y = 1:n)[sample(n^2, .5*n^2),]
quplot(x, y, data = df, geom = 'tile')
quplot(x, y, data = df, geom = 'tile') + theme_nothing()
quplot(1:10, 1:10) +
  annotation_custom(
    grob = ggplotGrob(qplot(1:10,1:10)),
    8, Inf, -Inf, 2
  )
quplot(1:10, 1:10) +
  annotation_custom(
    grob = ggplotGrob(qplot(1:10,1:10) + theme_nothing()),
    8, Inf, -Inf, 2
  )
quplot(1:10, 1:10) +
  annotation_custom(
    grob = ggplotGrob(qplot(1:10,1:10) + theme_inset()),
    8, Inf, -Inf, 2
theme_nothing

Make a blank ggplot2 theme.

Description

theme_nothing simply strips all thematic element in ggplot2.

Usage

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

# no legend example
n <- 50
df <- expand.grid(x = 1:n, y = 1:n)[sample(n^2,.5*n^2),]
p <- qplot(x, y, data = df, geom = 'tile')
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
wind

```r
df$class <- factor(sample(0:1, .5*n^2, replace = TRUE))
p <- qplot(x, y, data = df, geom = "tile", 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)
```

---

**wind**

*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

Value

a data frame with columns lon and lat (in degrees)

Author(s)

David Kahle <david.kahle@gmail.com>, based on function XY2LatLon 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:
XYZLonLat(480, 845, zoom = 11)
XYZLonLat(0, 0, zoom = 1)
XYZLonLat(0, 0, 255, 255, zoom = 1)
XYZLonLat(0, 0, 255, 255, zoom = 1)

## End(Not run)
```

---

**zips**

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, 3
crime, 4
distQueryCheck, 5
facet_grid, 47
facet_wrap, 47
geocode, 5, 42
geocodeQueryCheck (geocode), 5
geom_leg, 7, 53
get_cloudfademap, 11
get_googlemap, 12, 15
get_map, 15, 26, 35, 37, 45, 46
get_navermap, 16
get_openstreetmap, 15, 18
get_stamenmap, 19
GetMap, 16
ggimage, 23
gglocator, 24
ggmap, 12, 14, 16, 17, 19, 20, 26, 45
ggmap-package (ggmap), 26
ggmapplot, 35
hadley, 36
inset, 36
inset_ggmap (inset_raster), 37
inset_raster, 37
legs2route, 37, 53
LonLat2XY, 39
make_bbox, 4, 40
mapdist, 40
mutate_geocode, 6, 42
OSM_scale_lookup, 18, 43
package-ggmap (ggmap), 26
print.ggmap, 44
qmap, 26, 35, 45
qmplot, 46
revgeocode, 51
route, 8, 37, 52
routeQueryCheck, 53, 54
theme_inset, 55
theme_nothing, 56
wind, 57
XY2LonLat, 57
zips, 58