**The Flat Earth**

YouTube contains thousands of videos from people that claim the earth is flat and that science as we know it is a “globalist conspiracy”. While their claims are hilarious, people that believe our planet is flat are often passionate and even conduct simple experiments and engage in naive flat earth mathematics.

Adherents to the idea that the world is flat often propose [Gleason’s Map](https://amzn.to/2Pxh3I7) (shown above) as their version of the correct representation of our planet. This map is not a map of the flat earth but a polar [Azimuthal equidistant projection](https://en.wikipedia.org/wiki/Azimuthal_equidistant_projection) of the globe. Even though the map itself states that it is a projection of the world, flat earth believers nevertheless see it as a literal representation of the earth.

Projecting the spherical earth on a flat surface is a complex task which will always require compromise as it is impossible to draw the surface of the globe on a piece of paper truthfully.Â  The video below provides a practical introduction to map projections.

We can recreate Gleason’s map with ggplot, which incorporates the mapproj package to show maps in various projections. The Azimuthal equidistant method is popularised in the flag of the United Nations. Antarctica is in this projection displayed as a ring around the world. Flat earth evangelists believe that the South Pole a ring of ice that prevents us from proceeding beyond the disc.

library(tidyverse)

library(tidyverse)

world <- map\_data("world")

worldmap <- ggplot(world) +

geom\_path(aes(x = long, y = lat, group = group)) +

theme(axis.text = element\_blank(),

axis.ticks = element\_blank()) +

labs(x = "", y = "")

worldmap +

coord\_map("azequidistant", orientation = c(90, 0, 270))

The [coord\_map](https://ggplot2.tidyverse.org/reference/coord_map.html) function allows you to change projections. The orientation argument defines the centre point of the projection. The most common location is the North Pole (90, 0). The third value gives the clockwise rotation in degrees. Gleason’s map uses a rotation of 270 degrees.

Coord\_map Function

coord\_map() projects a portion of the earth, which is approximately spherical, onto a flat 2D plane using any projection defined by the mapproj package. Map projections do not, in general, preserve straight lines, so this requires considerable computation. coord\_quickmap() is a quick approximation that does preserve straight lines. It works best for smaller areas closer to the equator.

coord\_map(

projection = "mercator",

...,

parameters = NULL,

orientation = NULL,

xlim = NULL,

ylim = NULL,

clip = "on"

)

coord\_quickmap(xlim = NULL, ylim = NULL, expand = TRUE, clip = "on")

Arguments

|  |  |
| --- | --- |
| **projection** | projection to use, see [mapproj::mapproject()](https://rdrr.io/pkg/mapproj/man/mapproject.html) for list |
| **..., parameters** | Other arguments passed on to [mapproj::mapproject()](https://rdrr.io/pkg/mapproj/man/mapproject.html). Use ... for named parameters to the projection, and parameters for unnamed parameters. ... is ignored if the parameters argument is present. |
| **orientation** | projection orientation, which defaults to c(90, 0, mean(range(x))). This is not optimal for many projections, so you will have to supply your own. See [mapproj::mapproject()](https://rdrr.io/pkg/mapproj/man/mapproject.html) for more information. |
| **xlim, ylim** | Manually specific x/y limits (in degrees of longitude/latitude) |
| **clip** | Should drawing be clipped to the extent of the plot panel? A setting of "on" (the default) means yes, and a setting of "off" means no. For details, please see [coord\_cartesian()](https://ggplot2.tidyverse.org/reference/coord_cartesian.html). |
| **expand** | If TRUE, the default, adds a small expansion factor to the limits to ensure that data and axes don't overlap. If FALSE, limits are taken exactly from the data or xlim/ylim. |

Details

In general, map projections must account for the fact that the actual length (in km) of one degree of longitude varies between the equator and the pole. Near the equator, the ratio between the lengths of one degree of latitude and one degree of longitude is approximately 1. Near the pole, it tends towards infinity because the length of one degree of longitude tends towards 0. For regions that span only a few degrees and are not too close to the poles, setting the aspect ratio of the plot to the appropriate lat/lon ratio approximates the usual mercator projection. This is what coord\_quickmap() does, and is much faster (particularly for complex plots like [geom\_tile()](https://ggplot2.tidyverse.org/reference/geom_tile.html)) at the expense of correctness.

Examples

if ([require](https://rdrr.io/r/base/library.html)("maps")) {

nz <- [map\_data](https://ggplot2.tidyverse.org/reference/map_data.html)("nz")

# Prepare a map of NZ

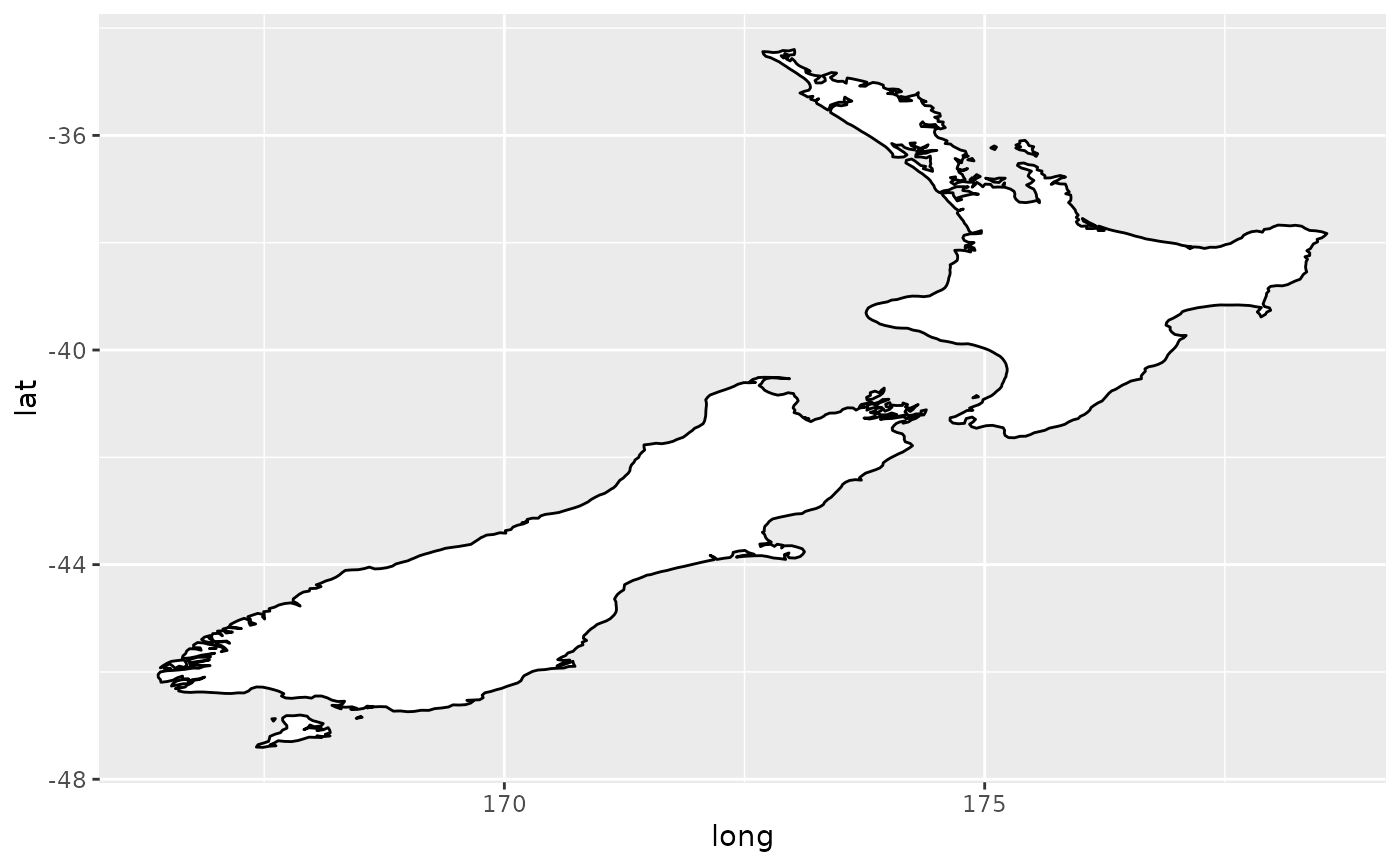
nzmap <- [ggplot](https://ggplot2.tidyverse.org/reference/ggplot.html)(nz, [aes](https://ggplot2.tidyverse.org/reference/aes.html)(x = long, y = lat, group = group)) +

[geom\_polygon](https://ggplot2.tidyverse.org/reference/geom_polygon.html)(fill = "white", colour = "black")

# Plot it in cartesian coordinates

nzmap

}

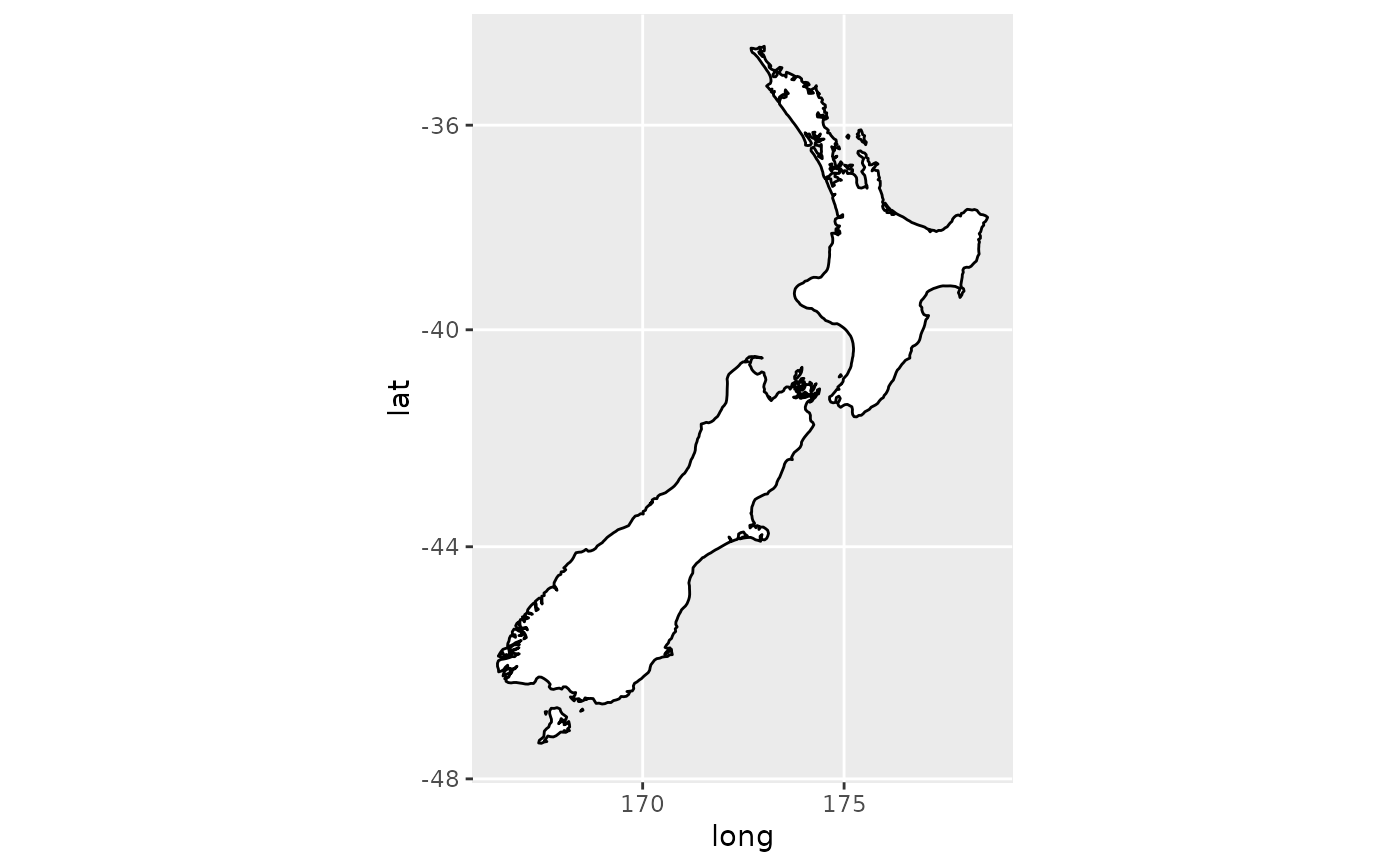


if ([require](https://rdrr.io/r/base/library.html)("maps")) {

# With correct mercator projection

nzmap + coord\_map()

}

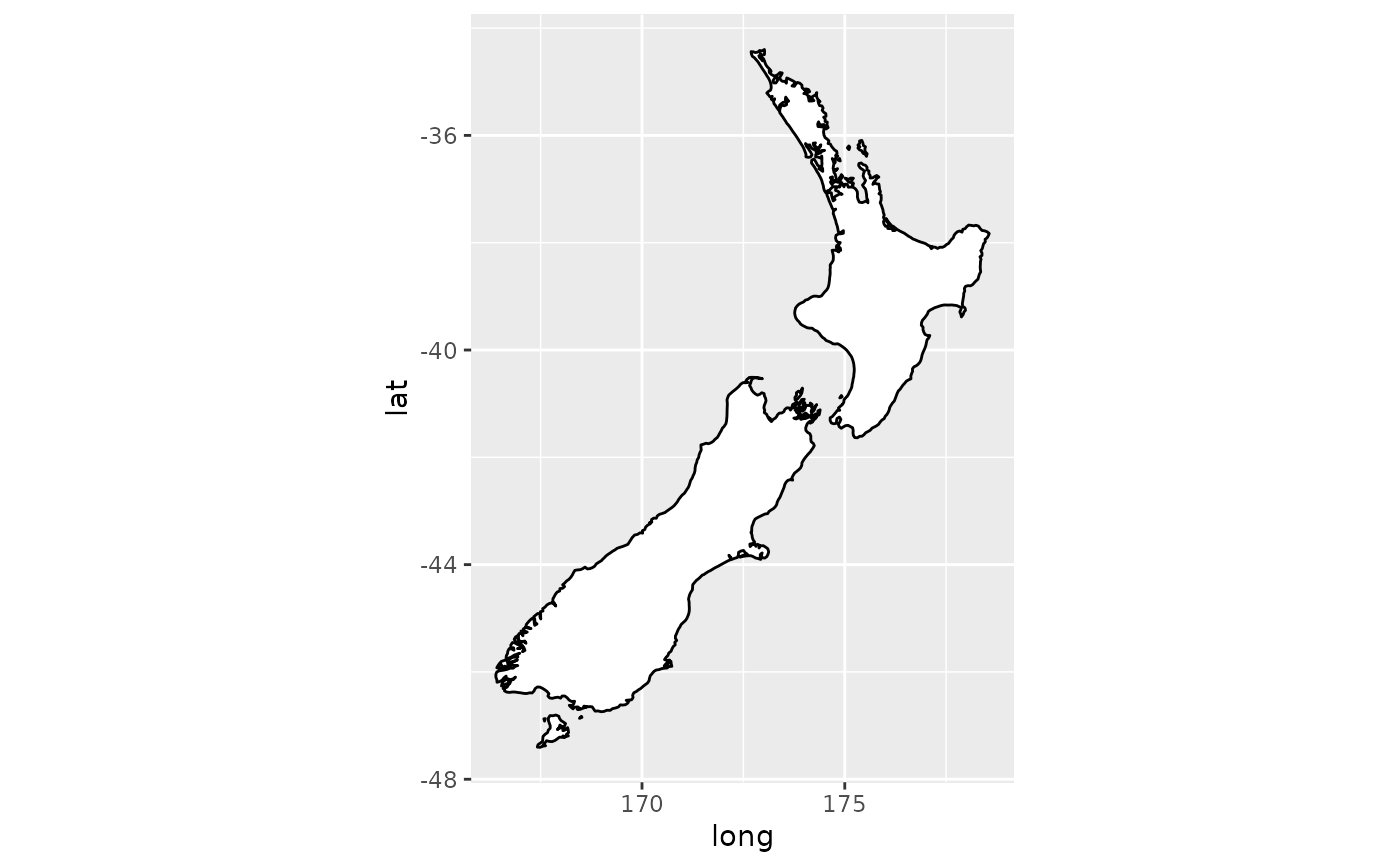


if ([require](https://rdrr.io/r/base/library.html)("maps")) {

# With the aspect ratio approximation

nzmap + coord\_quickmap()

}

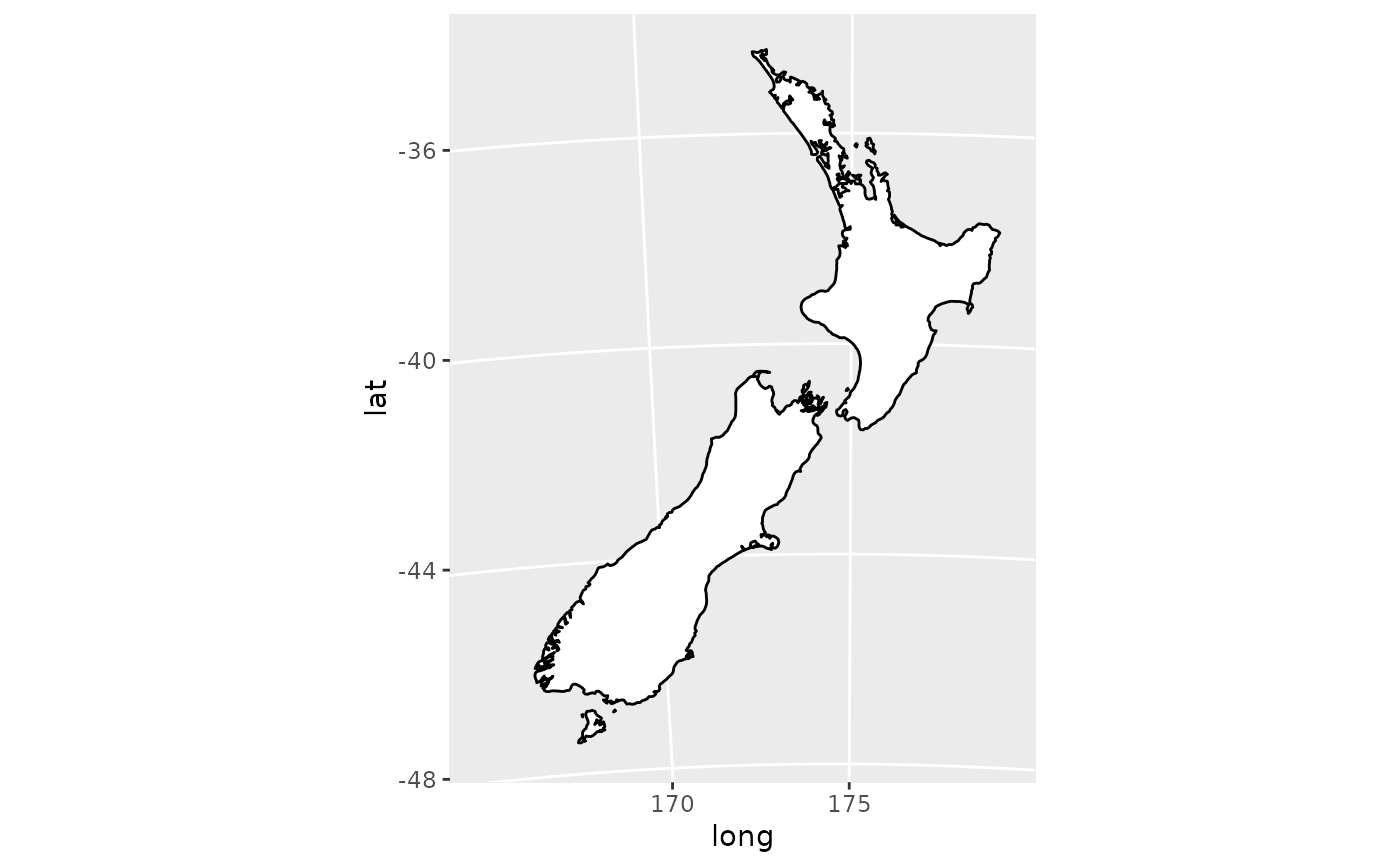


if ([require](https://rdrr.io/r/base/library.html)("maps")) {

# Other projections

nzmap + coord\_map("azequalarea", orientation = [c](https://rdrr.io/r/base/c.html)(-36.92, 174.6, 0))

}



if ([require](https://rdrr.io/r/base/library.html)("maps")) {

states <- [map\_data](https://ggplot2.tidyverse.org/reference/map_data.html)("state")

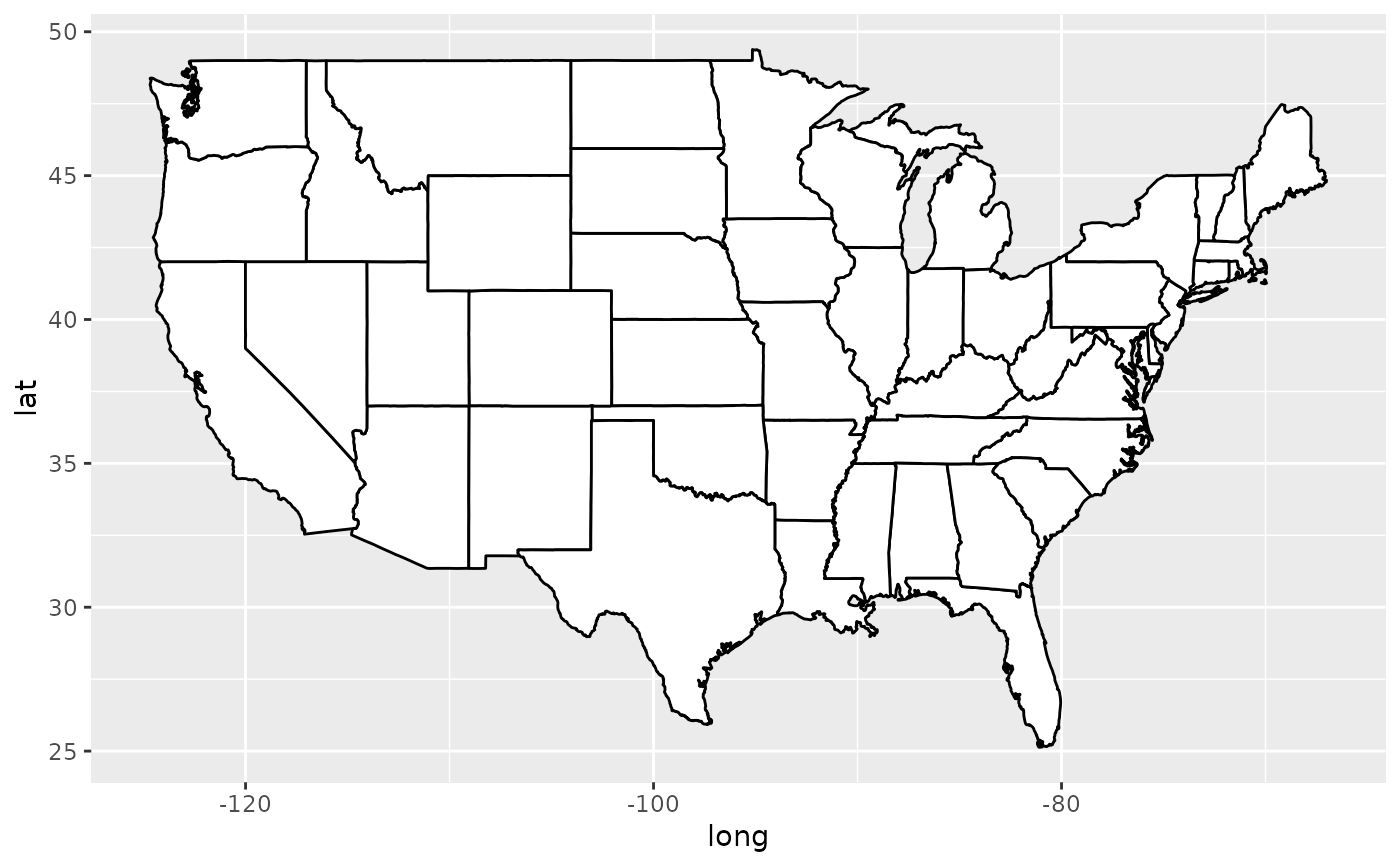
usamap <- [ggplot](https://ggplot2.tidyverse.org/reference/ggplot.html)(states, [aes](https://ggplot2.tidyverse.org/reference/aes.html)(long, lat, group = group)) +

[geom\_polygon](https://ggplot2.tidyverse.org/reference/geom_polygon.html)(fill = "white", colour = "black")

# Use cartesian coordinates

usamap

}

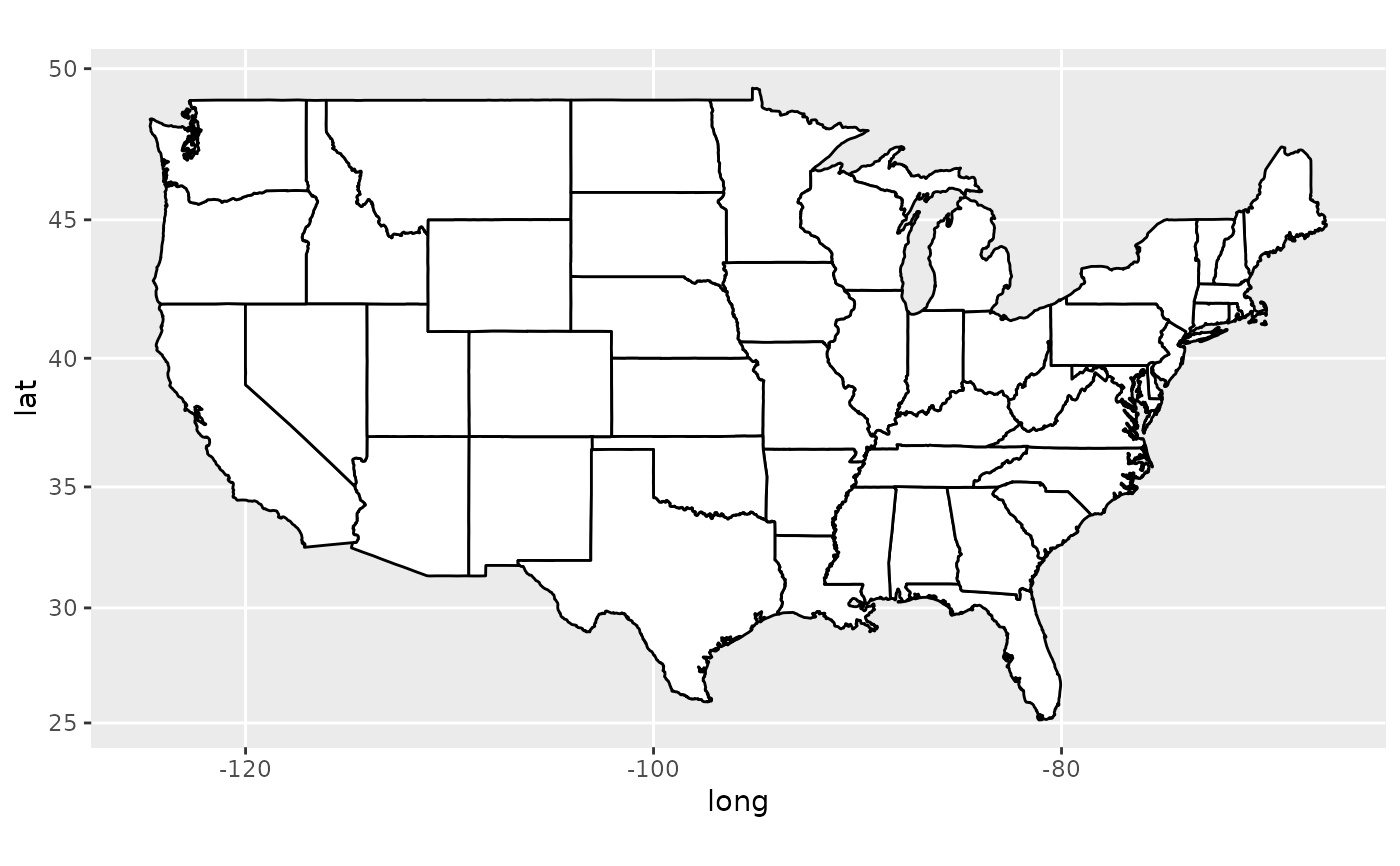


if ([require](https://rdrr.io/r/base/library.html)("maps")) {

# With mercator projection

usamap + coord\_map()

}

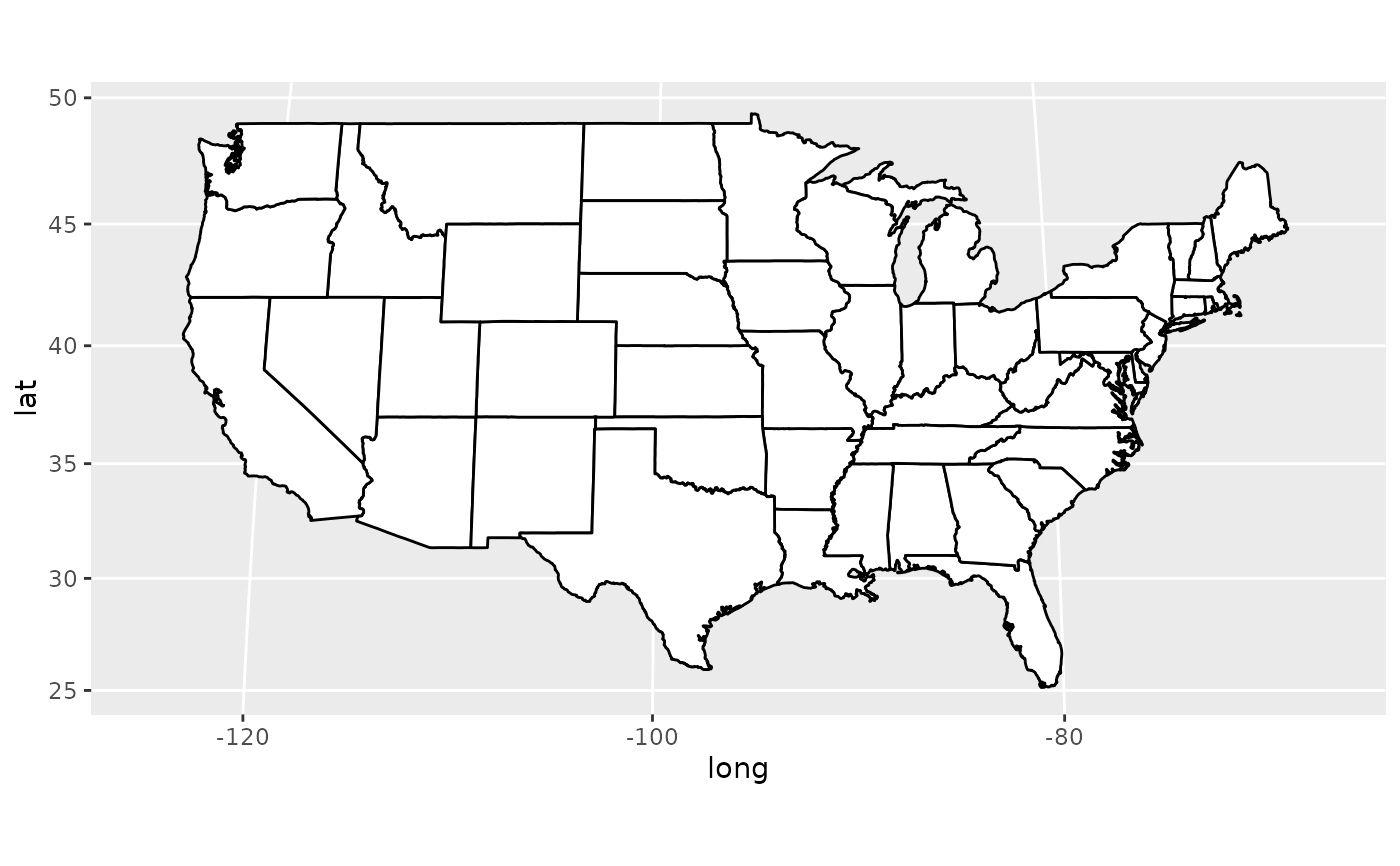


if ([require](https://rdrr.io/r/base/library.html)("maps")) {

# See ?mapproject for coordinate systems and their parameters

usamap + coord\_map("gilbert")

}



if ([require](https://rdrr.io/r/base/library.html)("maps")) {

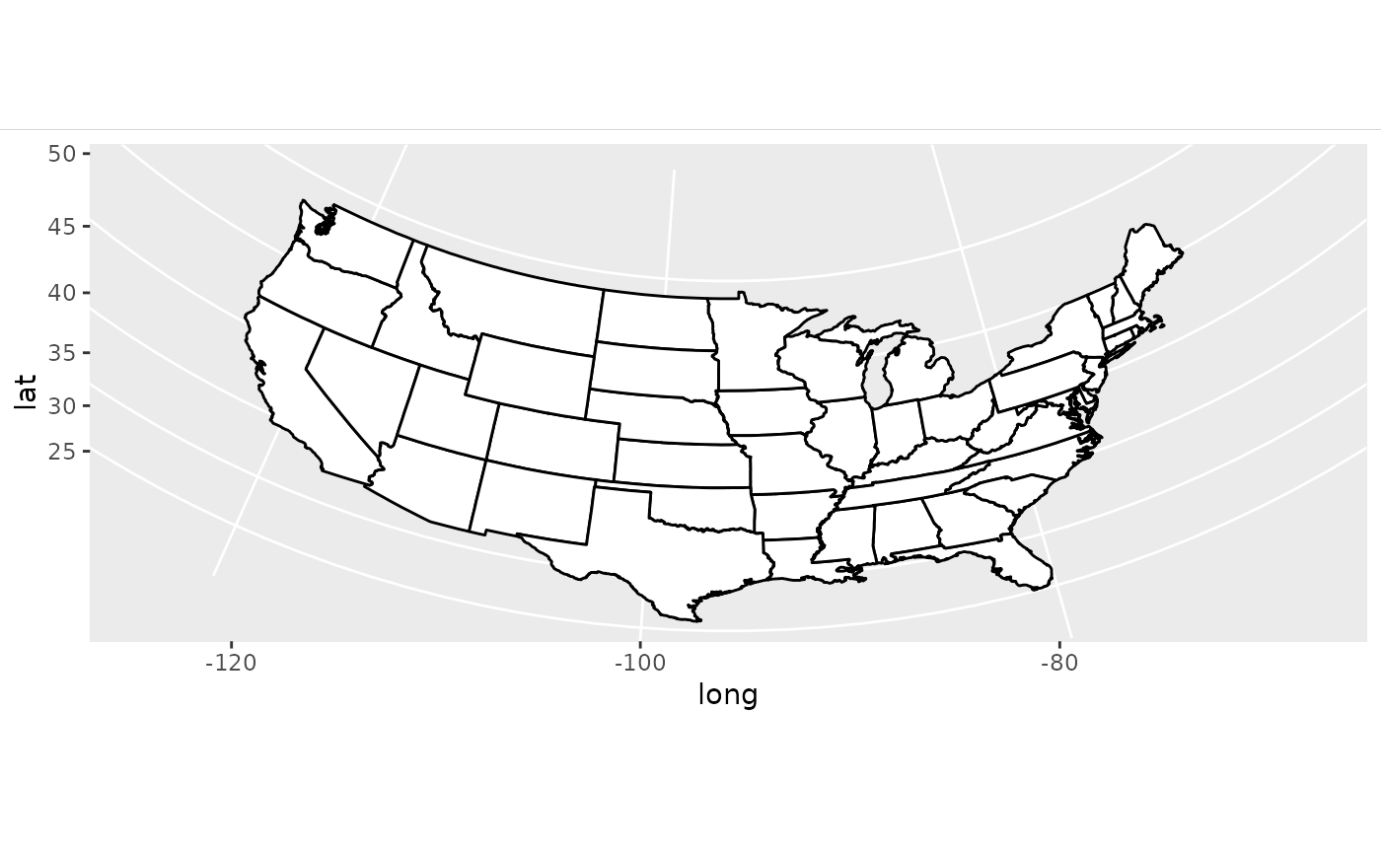
# For most projections, you'll need to set the orientation yourself

# as the automatic selection done by mapproject is not available to

# ggplot

usamap + coord\_map("orthographic")

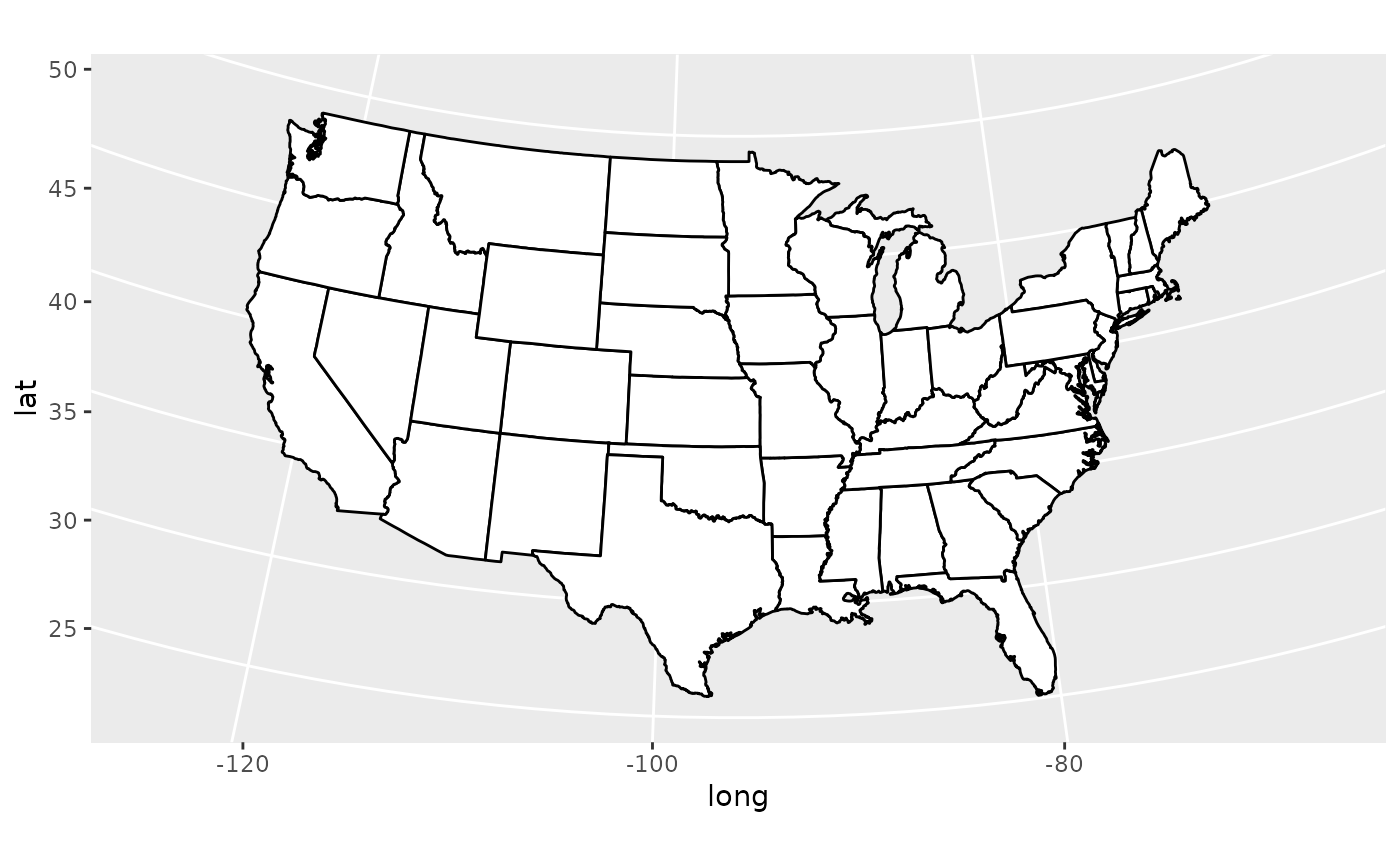
}



if ([require](https://rdrr.io/r/base/library.html)("maps")) {

usamap + coord\_map("conic", lat0 = 30)

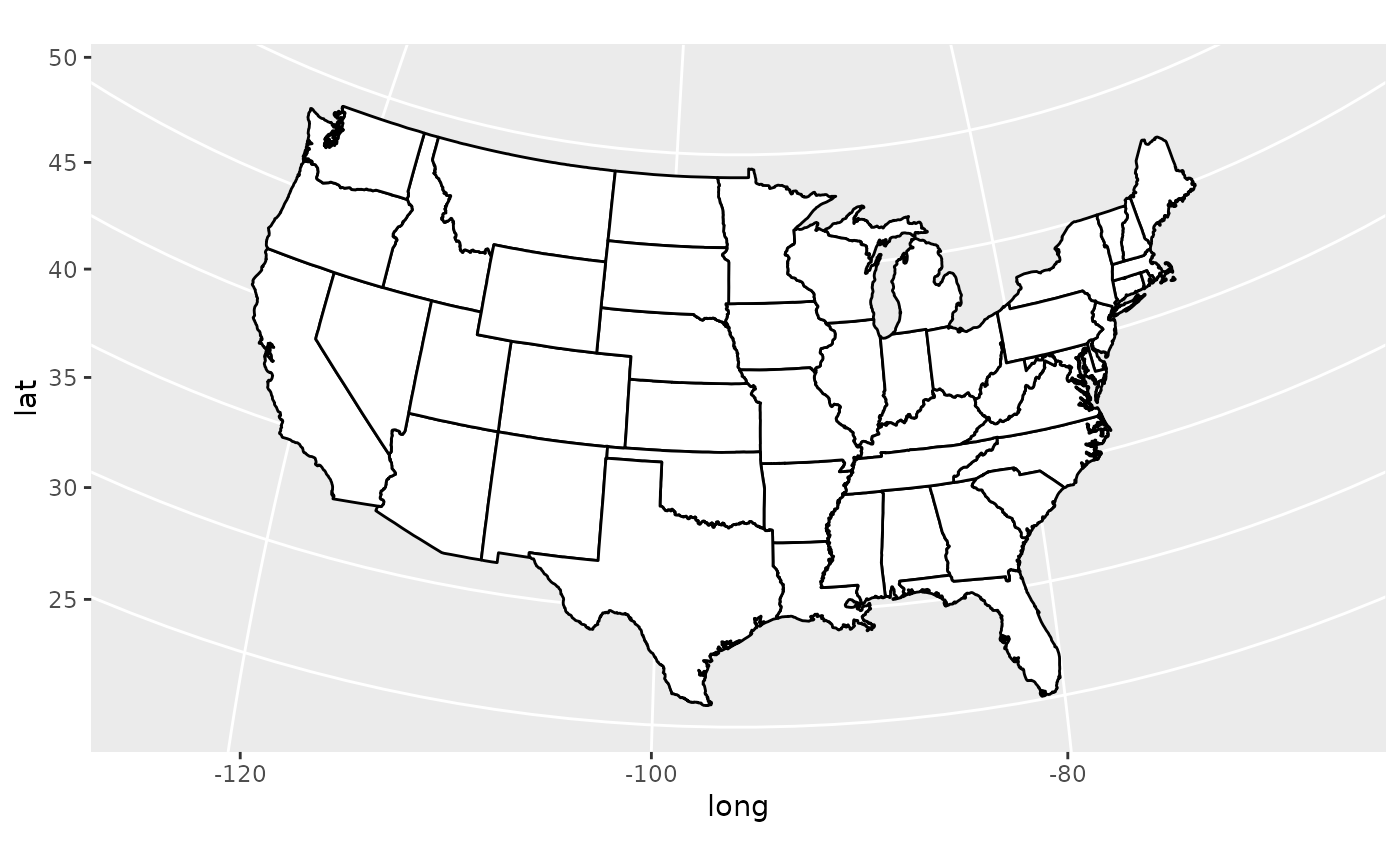
}



if ([require](https://rdrr.io/r/base/library.html)("maps")) {

usamap + coord\_map("bonne", lat0 = 50)

}



if (FALSE) {

if ([require](https://rdrr.io/r/base/library.html)("maps")) {

# World map, using geom\_path instead of geom\_polygon

world <- [map\_data](https://ggplot2.tidyverse.org/reference/map_data.html)("world")

worldmap <- [ggplot](https://ggplot2.tidyverse.org/reference/ggplot.html)(world, [aes](https://ggplot2.tidyverse.org/reference/aes.html)(x = long, y = lat, group = group)) +

[geom\_path](https://ggplot2.tidyverse.org/reference/geom_path.html)() +

[scale\_y\_continuous](https://ggplot2.tidyverse.org/reference/scale_continuous.html)(breaks = (-2:2) \* 30) +

[scale\_x\_continuous](https://ggplot2.tidyverse.org/reference/scale_continuous.html)(breaks = (-4:4) \* 45)

# Orthographic projection with default orientation (looking down at North pole)

worldmap + coord\_map("ortho")

}

if ([require](https://rdrr.io/r/base/library.html)("maps")) {

# Looking up up at South Pole

worldmap + coord\_map("ortho", orientation = [c](https://rdrr.io/r/base/c.html)(-90, 0, 0))

}

if ([require](https://rdrr.io/r/base/library.html)("maps")) {

# Centered on New York (currently has issues with closing polygons)

worldmap + coord\_map("ortho", orientation = [c](https://rdrr.io/r/base/c.html)(41, -74, 0))

}

}

**My Round-the-World Itinerary**

My trip will take me from Australia to Japan, from where I go to the Netherlands. The last two legs will bring me back to Australia via San Francisco.

The itinerary is stored in a data frame, and the ggmap package geocodes the longitude and latitude of each of the locations on my trip.

Flight Map Air Travel Route Implementation with R

## Init

**library**(tidyverse)

**library**(ggmap)

**library**(ggrepel)

api <- **readLines**("google.api") # Text file with the API key

**register\_google**(key = api)

## Read flight and airports lists

#flights <- read\_csv("flights.csv")

flights <- **read\_csv**("flight-map/flights.csv")

airports\_file <- "flight-map/airports.csv"

**if** (**file.exists**(airports\_file)) {

airports <- **read\_csv**(airports\_file)

} else {

airports <- **tibble**(airport = **NA**, lon = **NA**, lat= **NA**)

}

## Lookup coordinates

## Some airports need country names to ensure Google finds the correct location

## The geocoding keeps looping till all coordinates have been found

destinations <- **unique**(**c**(flights$From, flights$To))

new\_destinations <- destinations[!destinations %in% airports$airport]

**while** (**length**(new\_destinations) > **0**) {

new\_airports <- **geocode**(new\_destinations) %>%

**mutate**(airport = new\_destinations) %>%

**select**(airport, lon, lat)

airports <- **rbind**(airports, new\_airports) %>%

**filter**(!**is.na**(lon) | !**is.na**(lat))

new\_destinations <- destinations[!destinations %in% airports$airport]

}

**write\_csv**(airports, "flight-map/airports.csv")

## Remove country names

airports$airport <- **as.character**(airports$airport)

comma <- **regexpr**(",", airports$airport)

airports$airport**[which**(comma > **0**)] <- **substr**(airports$airport**[which**(comma > **0**)], **1**, comma[comma > **0**] - **1**)

## Remove return flights

d <- **vector**()

**for** (i in **1**:**nrow**(flights)) {

d <- **which**(**paste**(flights$From, flights$To) %in%

**paste**(flights$To[i], flights$From[i]))

flights$From[d] <- "R"

}

flights2 <- flights %>%

**filter**(From != "R") %>%

**select**(From, To)

## Add coordinates to flight list

flights <- **merge**(flights, airports, by.x = "From", by.y = "airport")

flights <- **merge**(flights, airports, by.x = "To", by.y = "airport")

flights <- flights %>%

**select**(From, To, lon.x, lat.x, lon.y, lat.y) %>%

**as\_tibble**()

## Split Circumnaviation Flights at -180/180 degrees

circ <- **which**(**abs**(flights$lon.y - flights$lon.x) > **180**)

flights[circ,]

flights$lon.y[circ] <- **ifelse**(flights$lon.y[circ] < **0**, **180**, **-180**)

flights$lat.y[circ] <- **rowSums**(flights[circ, **c**("lat.x", "lat.y")]) / **2**

leg2 <- airports %>%

**filter**(airport %in% flights$To[circ]) %>%

**mutate**(From = **rep**("", **length**(circ))) %>%

**mutate**(lon.x = -flights$lon.y[circ], lat.x = flights$lat.y[circ]) %>%

**select**(From, To = airport, lon.x, lat.x, lon.y = lon, lat.y = lat)

flights <- **rbind**(flights, leg2)

## Plot flight routes

worldmap <- **borders**("world2", colour="#efede1", fill="#efede1")

**ggplot**() + worldmap +

**geom\_point**(data = airports, **aes**(x = lon, y = lat), col = "#970027") +

**geom\_text\_repel**(data=airports, **aes**(x = lon, y = lat, label = airport), col = "black", size = **2**, segment.color = **NA**) +

**geom\_curve**(data = flights, **aes**(x = lon.x, y = lat.x, xend = lon.y, yend = lat.y), col = "#b29e7d", size = **.4**) +

**theme\_void**()

**ggsave**("flights\_map.png", dpi = **300**)

The itinerary is visualised using the same method as above but centring on Antarctica. The geosphere package helps to estimate the total travel distance, which is approximately 38,995 km, slightly less than a trip around the equator. This distance is the [great circle distance](https://en.wikipedia.org/wiki/Great-circle_distance), which is the shortest distance between two points on a sphere, adjusted for a spheroid (a flattened sphere).

The flight paths on this map are curved because of the inevitable distortions when projecting a sphere on a flat surface.

## Define itinerary

library(ggmap)

airports <- tibble(city = c("Melbourne", "Tokyo", "Amsterdam", "San Francisco", "Melbourne"))

## Find coordinates

## Where you receive an OVER\_QUERY\_LIMIT error, repeat the code

itinerary <- geocode(airports$city) %>%

mutate(location = airports$city)

## Split travel past dateline

dl <- which(diff(itinerary$lon) > 180)

dr <- ifelse(itinerary$lon[dl] < 0, -180, 180)

dateline <- tibble(lon = c(dr, -dr),

lat = rep(mean(itinerary$lat[dl:(dl + 1)]), 2),

location = "dateline")

itinerary <- rbind(itinerary[1:dl, ], dateline, itinerary[(dl + 1):nrow(itinerary), ]) itinerary ## Visualise worldmap + geom\_point(data = itinerary, aes(lon, lat), colour = "red", size = 4) + geom\_path(data = itinerary, aes(lon, lat), colour = "red", size = 1) + coord\_map("azequidistant", orientation = c(-90, 0, 270)) ggsave("rtw.png", dpi = 150) ## Great Circle Distance library(geosphere) sapply(1:(nrow(itinerary) - 1), function(l) distVincentyEllipsoid(itinerary[l, 1:2], itinerary[(l + 1), 1:2]) / 1000) %>%

sum()

Round the world trip in polar Azimuthal equidistant projection.

**Flat Earth Mathematics**

If the Gleason map were in reality a map of the flat earth, then the flight paths on the map would show as straight lines. I was unable to plot these lines on the map because ggplot corrects for the projection effects.

The mapproj package contains the mapproject function to calculate the projected coordinates based on longitude and latitude. The output of this function is a grid with limits from -\pito \pi.

On the polar Azimuthal equidistant projection, distances from any point to the centre of the map are not distorted. A line from the lon/lat 0,0 to the north pole has a projected distance on \pi/2, which in the spherical world is \pi / 2 * 6378.137Â  = 10018.75km. In other words, we need to multiply the Euclidean distance with the radius of the Earth to estimate the length on a flat map.

This last code snippet converts the itinerary to projected coordinates, calculates the Euclidean distance between the points and multiplies this with the Earth’s diameter. The outcome is a staggering 83,675 km, more than twice the length over a sphere. The main reason for this difference is the distortion of this projection. When taking Antartica as the centre of the projection, East-West lengths on the Northern hemisphere are massively distorted. When undertaking the same calculation with the North pole as the center, the total distance is 46,348 km, still a lot more than the actual length of the journey. The only leg with a similar distance to reality is from Brisbane to Tokyo because this route is almost a radial line without distortion.

This article proves that the earth cannot be flat because aeroplanes would have to break the sound barrier to fly these distances in the time it takes to travel. But these facts will not stop people believing in a flat earth, I am after all an engineer and thus part of the globalist science conspiracy.

library(mapproj)

coords <- mapproject(itinerary$lon, itinerary$lat, "azequidistant",

orientation = c(-90, 0, 270))

coords <- tibble(x = coords$x, y = coords$y)

sum(sqrt(diff(coords$x)^2 + diff(coords$y)^2) \* 6378.137)