Data Science with R
Maps

Graham.Williams@togaware.com

19th August 2014

Visit http://HandsOnDataScience.com/ for more Chapters.

Done right maps can be a very effective communications tool. Numerous R packages work together to bring us a sophisticated mapping and spatial analysis capability.

The required packages for this module include:

```
library(ggplot2)      # Plotting maps.
library(maps)         # Map data.
library(oz)           # Map data for Australia.
library(scales)       # Functions: alpha() comma()
library(ggmap)        # Google maps.
```

As we work through this chapter, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the *help=* option of `library()`:

```
library(help=rattle)
```

This chapter is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

1 Google Maps: Geocoding

One of the fundamental things about spatial data and mapping is the geographic coordinate system used to uniquely identify locations. We use longitude (x axis, abbreviated lon) and latitude (y axis, abbreviated lat) for locations on our planet. The longitude is the angle from the meridian through Greenwich and the latitude is the angle from the equator. We can use ggmap (Kahle and Wickham, 2013) to `geocode()` street addresses and locations. Here are a few examples.

```
library(ggmap)
geocode("New York")
```

```
##         lon      lat
## 1 -74.00594 40.71278
```

```
geocode("Qiushi Road, Shenzhen")
```

```
##        lon      lat
## 1 113.9751 22.59073
```

```
geocode("Canberra")
```

```
##        lon     lat
## 1 149.1287 -35.282
```

```
geocode("Gus' Cafe, Garema Place, Canberra, Australia")
```

```
##        lon       lat
## 1 149.1321 -35.27839
```

```
geocode("9 Bunda Street, Canberra")
```

```
##        lon       lat
## 1 149.1312 -35.27757
```

```
geocode("11 Bunda Street, Canberra")
```

```
##        lon       lat
## 1 149.1313 -35.27758
```

For later use we will save some locations.

```
(syd <- as.numeric(geocode("Sydney")))
```

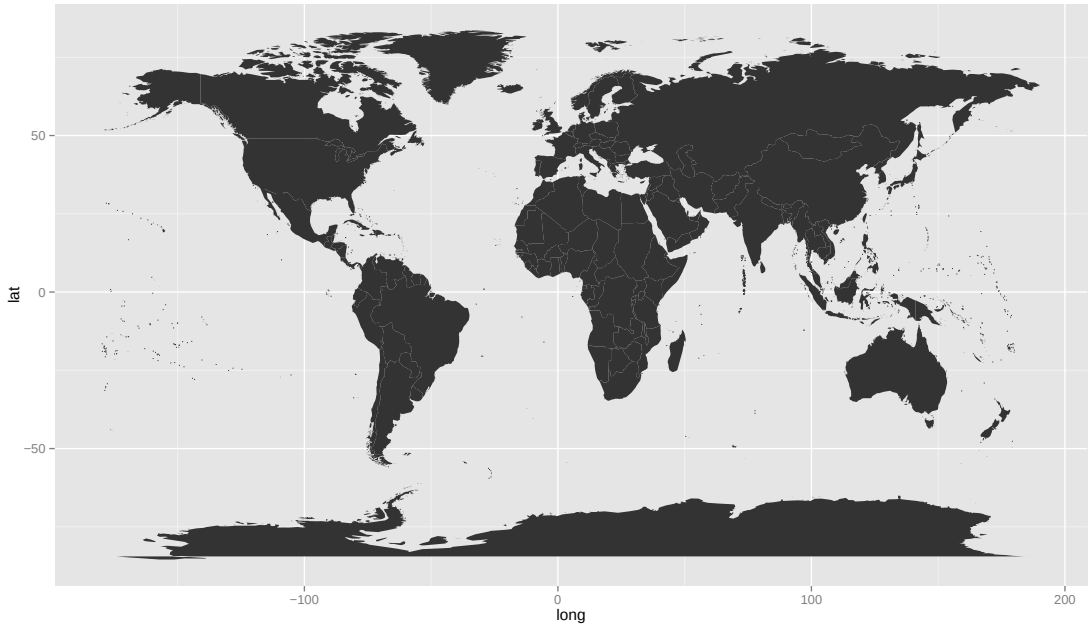
```
## [1] 151.20699 -33.86749
```

```
(cbr <- as.numeric(geocode("Canberra")))
```

```
## [1] 149.1287 -35.2820
```

```
sydbb <- c(151.15, -33.88, 151.25, -33.84)
```

2 World Map



The data here comes from the maps (Brownrigg, 2014) package. We load the vector data for plotting a world map using `map_data()`.

```
ds <- map_data("world")
class(ds)
```

```
## [1] "data.frame"
```

```
str(ds)
```

```
## 'data.frame': 25553 obs. of  6 variables:
##  $ long     : num  -133 -132 -132 -132 -130 ...
##  $ lat      : num  58.4 57.2 57 56.7 56.1 ...
##  $ group    : num  1 1 1 1 1 1 1 1 1 1 ...
## ....
```

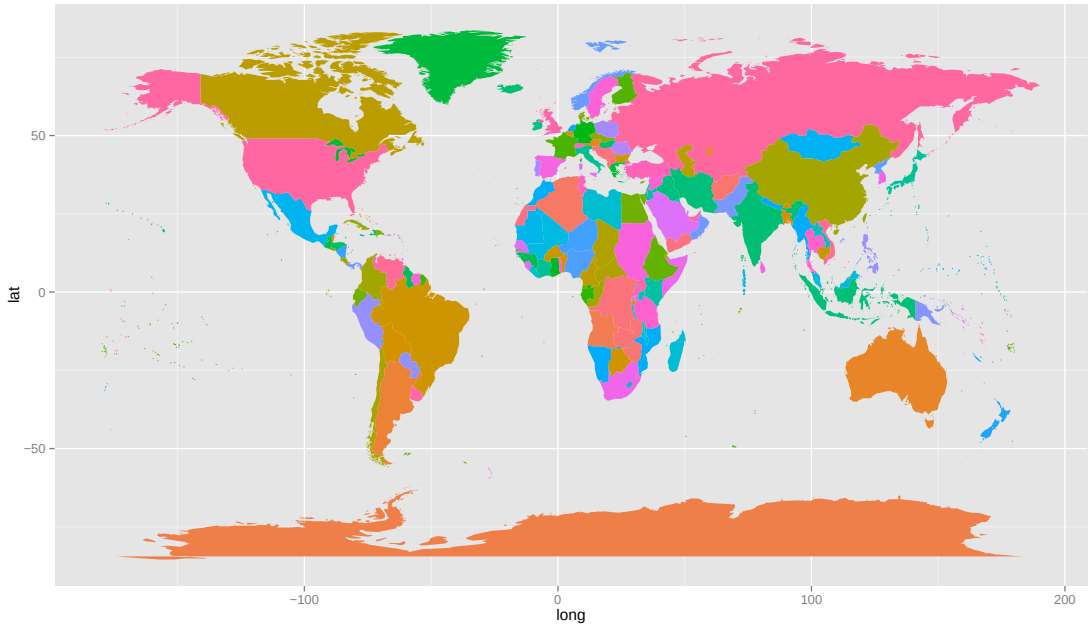
```
head(ds)
```

```
##     long   lat group order region subregion
## 1 -133.4 58.42     1     1 Canada      <NA>
## 2 -132.3 57.16     1     2 Canada      <NA>
## 3 -132.0 56.99     1     3 Canada      <NA>
## ....
```

It is then quite simple to plot the world map using ggplot2 (Wickham and Chang, 2014).

```
p <- ggplot(ds, aes(x=long, y=lat, group=group))
p <- p + geom_polygon()
p
```

3 World Map with Coloured Regions



We can specify a fill for the map, based on the regions.

```
p <- ggplot(ds, aes(x=long, y=lat, group=group, fill=region))
p <- p + geom_polygon()
p <- p + theme(legend.position = "none")
p
```

Note there are very many regions, and so the legend would be too large, so we turn that off.

```
length(unique(ds$region))
```

```
## [1] 234
```

4 Obtain Map Data

Visit http://www.gadm.org/country to download administrative vector data for any region of the world. The coordinate reference system is latitude/longitude and the WGS84 datum. The file formats include shapefiles, ESRI data files, Google Earth, and RData files.

From the website: A "shapefile" consist of at least three actual files. This is a commonly used format that can be directly used in Arc-anything, DIVA-GIS, and many other programs. Unfortunately, many of the non standard latin (roman / english) characters are lost in the shapefile. Even if you use the shapefile for mapping, you can use the .csv file that comes with the shapefiles, or the attribute data in the geodatabase for the correct attributes (the geodatabase is a MS Access database that (on windows) can be accessed via ODBC).

An "ESRI personal geodatabase" is a MS Access file that can be opened in ArcGIS (version 10). One of its advantages, compared to a shapefile, is that it can store non-latin characters (e.g. Cyrillic and Chinese characters). You can also query the (attribute) data in Access or via ODBC.

An "ESRI file geodatabase" can be opened in ArcGIS (version 10). It can also store non-latin characters (e.g. Cyrillic and Chinese characters).

A "Google Earth .kmz" file can be opened in Google Earth.

A "RData" file can be used in R (with the sp package loaded). See the CRAN spatial task view

5 Australian Map Data

A data frame version of the Australian map data that comes originally from oz (?) is available at http://www.elaliberte.info/software. We load the CSV version of the data frame here.

```
ds <- read.csv(file.path("data", "ozdata.csv"))
dim(ds)
```

```
## [1] 5852    7
```

```
head(ds, 2)
```

```
##   X  long    lat group order state border
## 1 1 129.0 -31.58     1     1    WA  coast
## 2 2 128.7 -31.69     1     2    WA  coast
```

```
tail(ds, 2)
```

```
##         X  long    lat group order state border
## 5851 5851 146.9 -43.59     7   660   TAS  coast
## 5852 5852 146.9 -43.59     7   661   TAS  coast
```

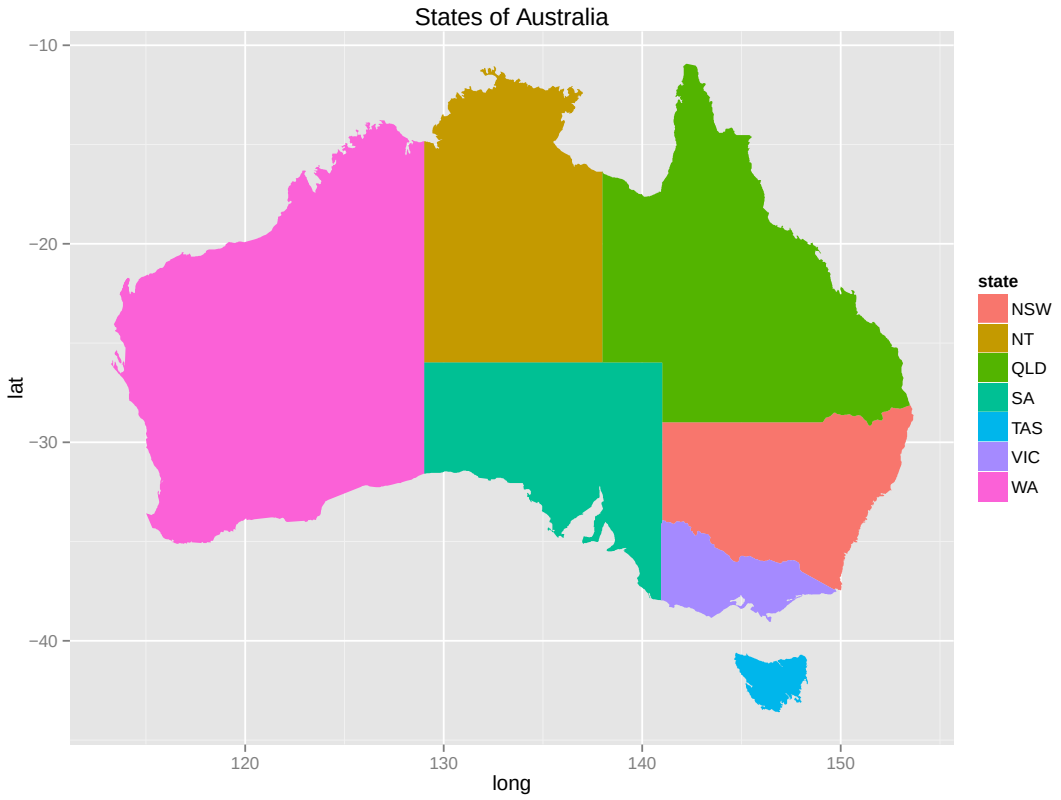
```
str(ds)
```

```
## 'data.frame': 5852 obs. of  7 variables:
##  $ X     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ long  : num  129 129 129 128 128 ...
##  $ lat   : num  -31.6 -31.7 -31.8 -31.8 -31.9 ...
##  $ group : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ order : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ state : Factor w/ 7 levels "NSW","NT","QLD",..: 7 7 7 7 7 7 7 7 7 7 ...
##  $ border: Factor w/ 19 levels "coast","NSW.QLD",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(ds)
```

```
##        X             long          lat            group      
##  Min.   :   1   Min.   :113   Min.   :-43.6   Min.   :1.00  
##  1st Qu.:1464   1st Qu.:131   1st Qu.:-35.6   1st Qu.:2.00  
##  Median :2926   Median :142   Median :-28.9   Median :3.00  
##  Mean   :2926   Mean   :139   Mean   :-27.2   Mean   :3.52  
##  3rd Qu.:4389   3rd Qu.:147   3rd Qu.:-16.1   3rd Qu.:5.00  
##  Max.   :5852   Max.   :154   Max.   :-10.9   Max.   :7.00  
##
##      order      state          border      
##  Min.   :   1   NSW: 856   coast  :4904  
##  1st Qu.: 210   NT : 743   NSW.VIC: 282  
##  Median : 418   QLD:1118   VIC.NSW: 282  
##  Mean   : 459   SA : 504   NSW.QLD: 176  
##  3rd Qu.: 648   TAS: 661   QLD.NSW: 176  
##  Max.   :1298   VIC: 672   QLD.SA :   3  
##                 WA :1298   (Other):  29
```

6 Australian Map with States



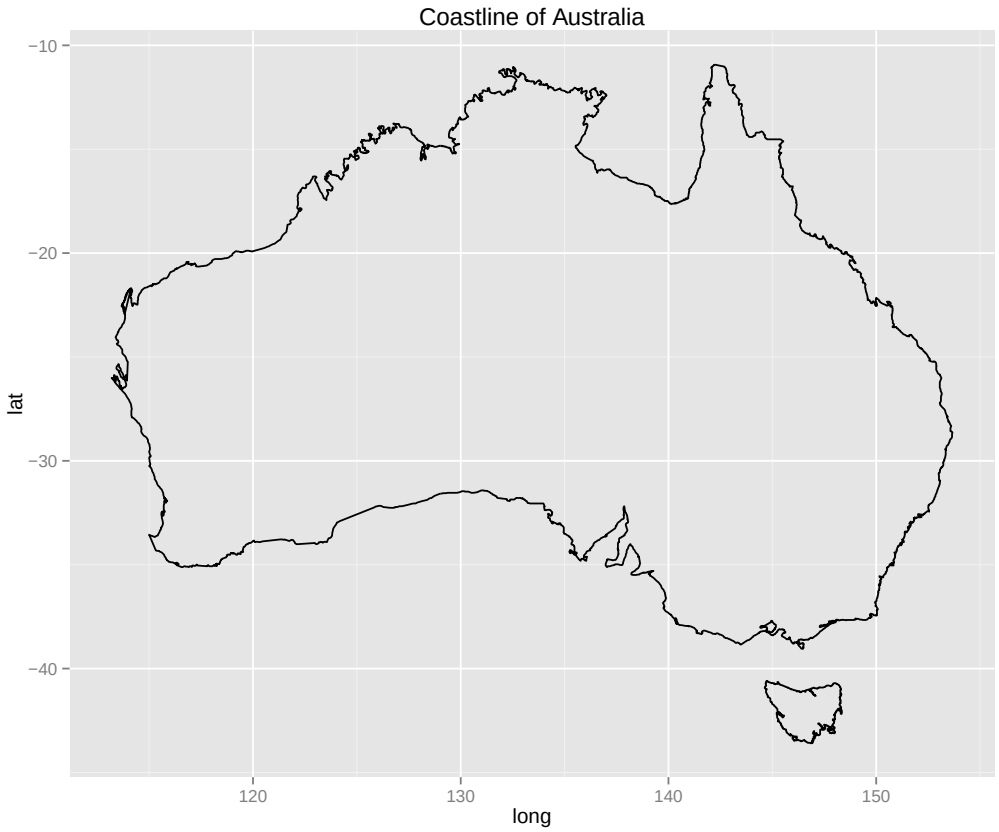
The map is drawn with a different fill colour for each state.

```
p <- ggplot(ds, aes(long, lat, fill=state))
p <- p + geom_polygon()
p <- p + coord_equal()
p <- p + ggtitle("States of Australia")
p
```

Note the use `coord_equal()` to ensure a properly proportioned map.

This example was motivated by the example using `qqplot()` at http://www.elaliberte.info/software.

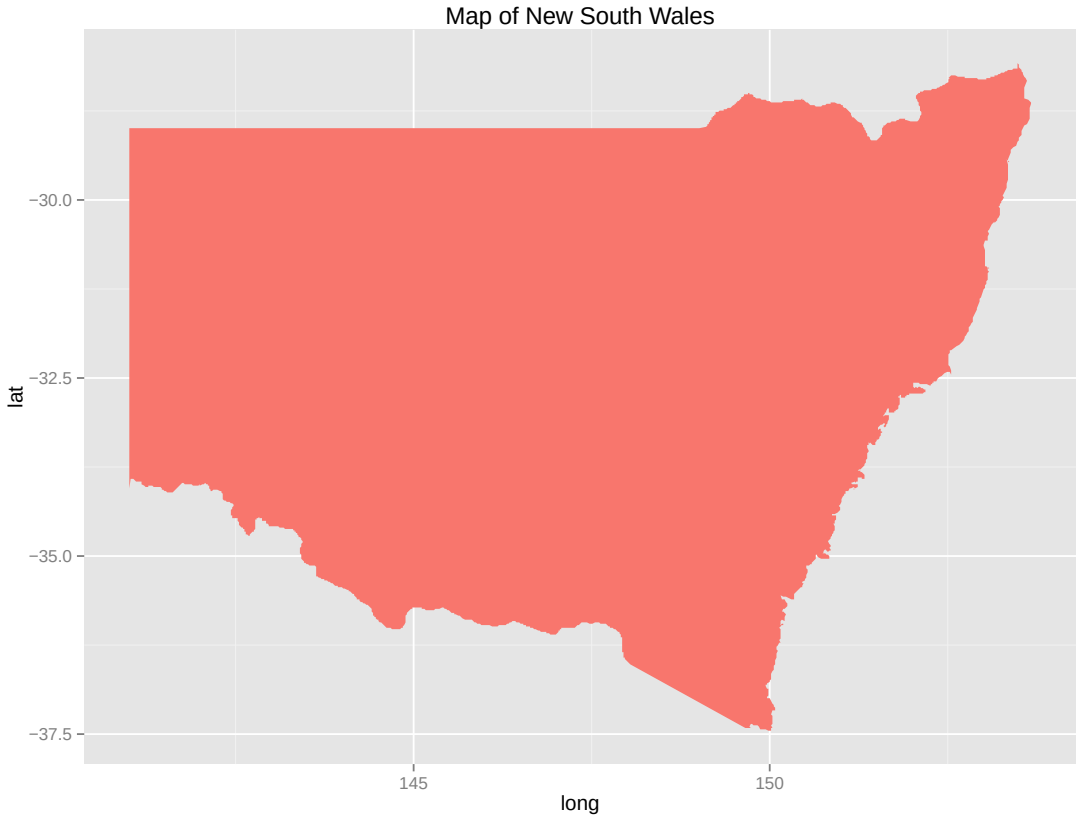
7 Map a Subset — Australian Coastline




```
p <- ggplot(subset(ds, border=="coast"), aes(long, lat, fill=state))
p <- p + geom_path()
p <- p + coord_equal()
p <- p + ggtitle("Coastline of Australia")
p
```

This example was motivated by the example using `qqplot()` at http://www.elaliberte.info/software.

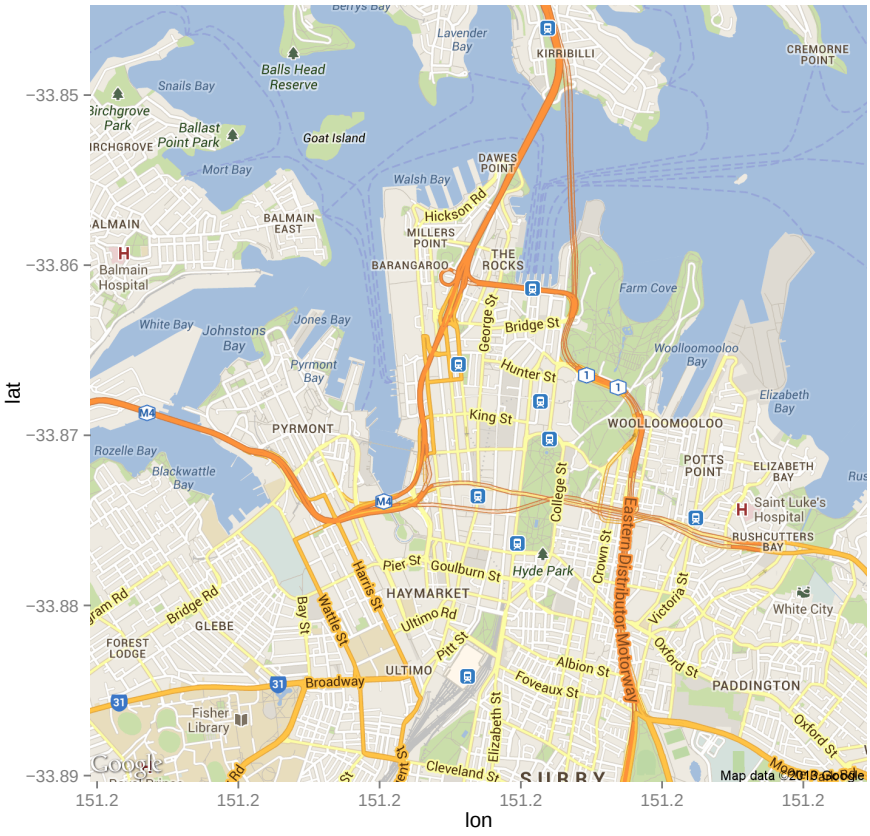
8 Map a Subset — New South Wales




```
p <- ggplot(subset(ds, state=="NSW"), aes(long, lat, fill=state))
p <- p + geom_polygon()
p <- p + coord_equal()
p <- p + ggtitle("Map of New South Wales")
p <- p + theme(legend.position="none")
p
```

This example was motivated by the example using `qqplot()` at http://www.elaliberte.info/software.

9 Google Map of Sydney



Here we download the map data for Sydney from Google using `get_map()` from `ggmap` (Kahle and Wickham, 2013). The data is transformed into a raster object for plotting.

```
map <- get_map(location="Sydney", zoom=14, maptype="roadmap", source="google")
p <- ggmap(map)
p
```

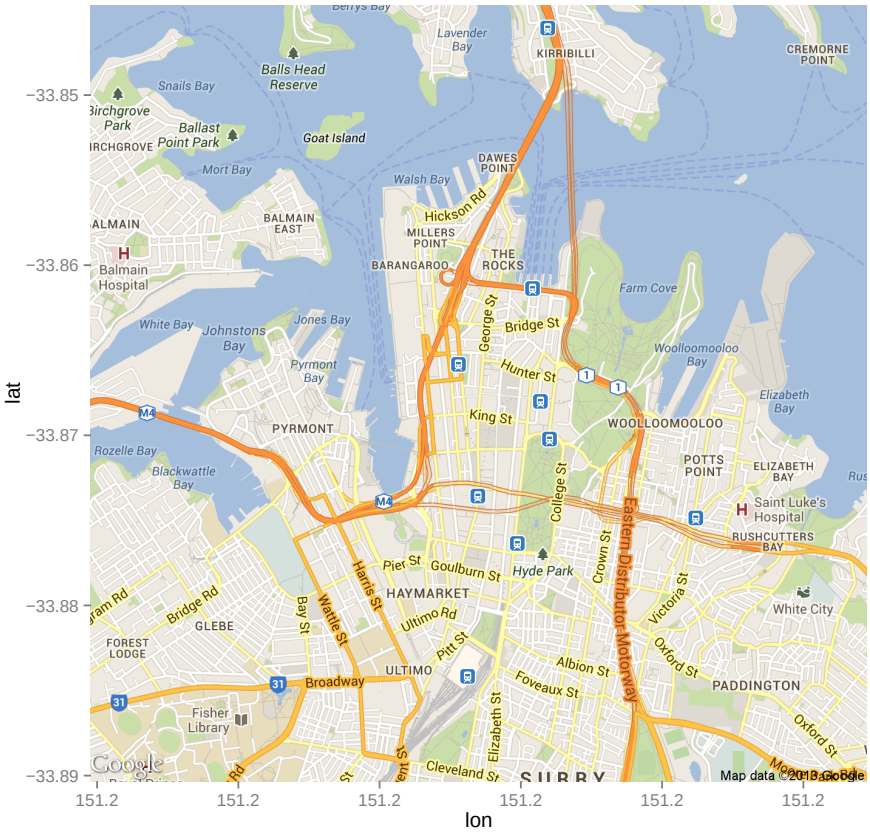
The `zoom=` option is an integer. A value of 0 returns a map of the whole world, centred around the location. The maximum value is 21 and returns a map down to the building. Choosing 4 is usually good for continents, 14 for a city.

Notice that the object returned by `ggmap()` is a ggplot object, and so all the usual `ggplot2` (Wickham and Chang, 2014) functions apply.

```
class(p)
```

```
## [1] "gg"     "ggplot"
```

10 Google Map by Geocode of Sydney



We will generally provide geo-codes to extract maps.

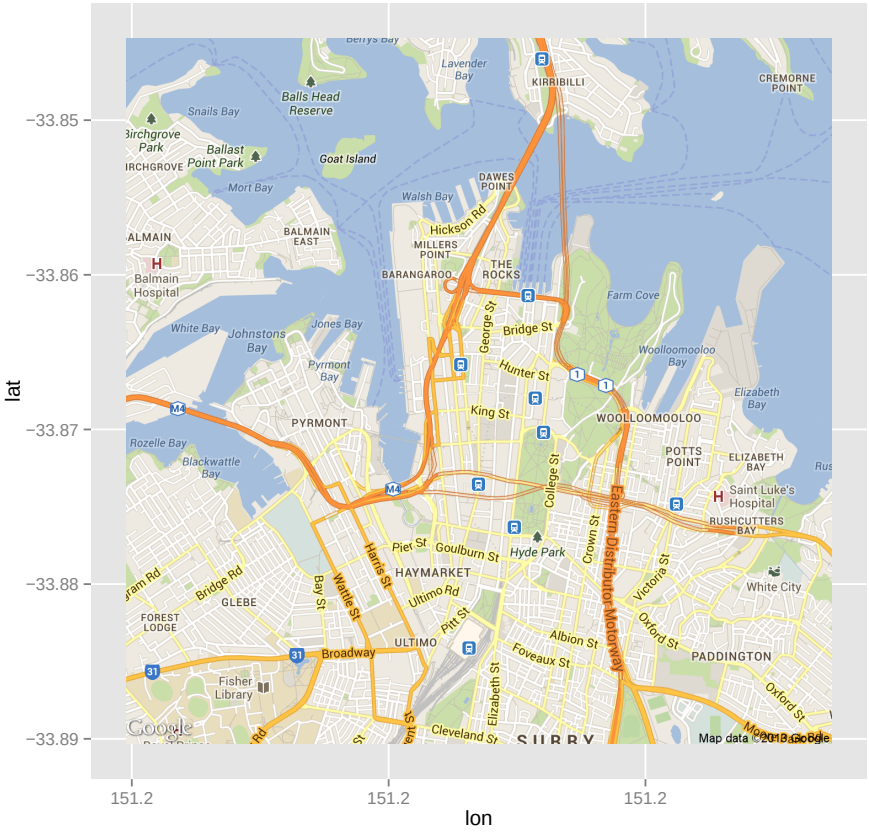
```
map <- get_map(location=syd, zoom=14, maptype="roadmap", source="google")
p <- ggmap(map)
p
```

Notice we have previously saved the location of Sydney into the variable *syd* and we have used that here to extract the same map.

```
(syd <- as.numeric(geocode("Sydney")))
```

```
## [1] 151.21 -33.87
```

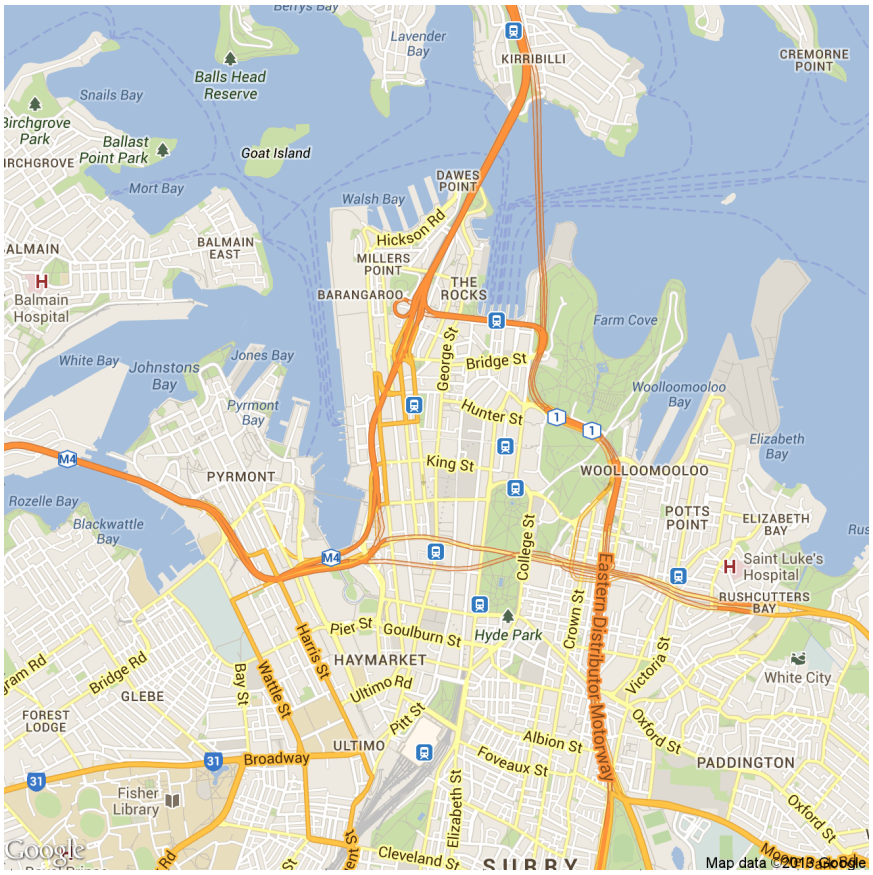
11 Map with Normal Extent of Sydney




```
map <- get_map(location="Sydney", zoom=14, maptype="roadmap", source="google")
p <- ggmap(map, extent="normal")
p
```

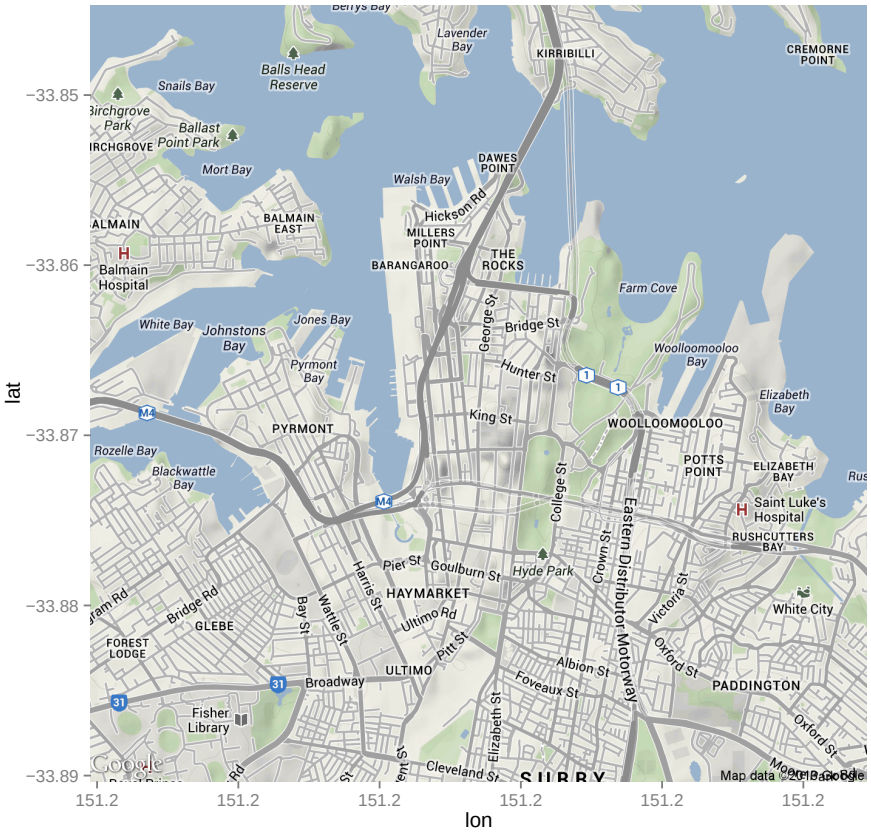
The default is `extent="panel"`.

12 Map with Device Extent of Sydney




```
map <- get_map(location="Sydney", zoom=14, maptype="roadmap", source="google")
p <- ggmap(map, extent="device")
p
```

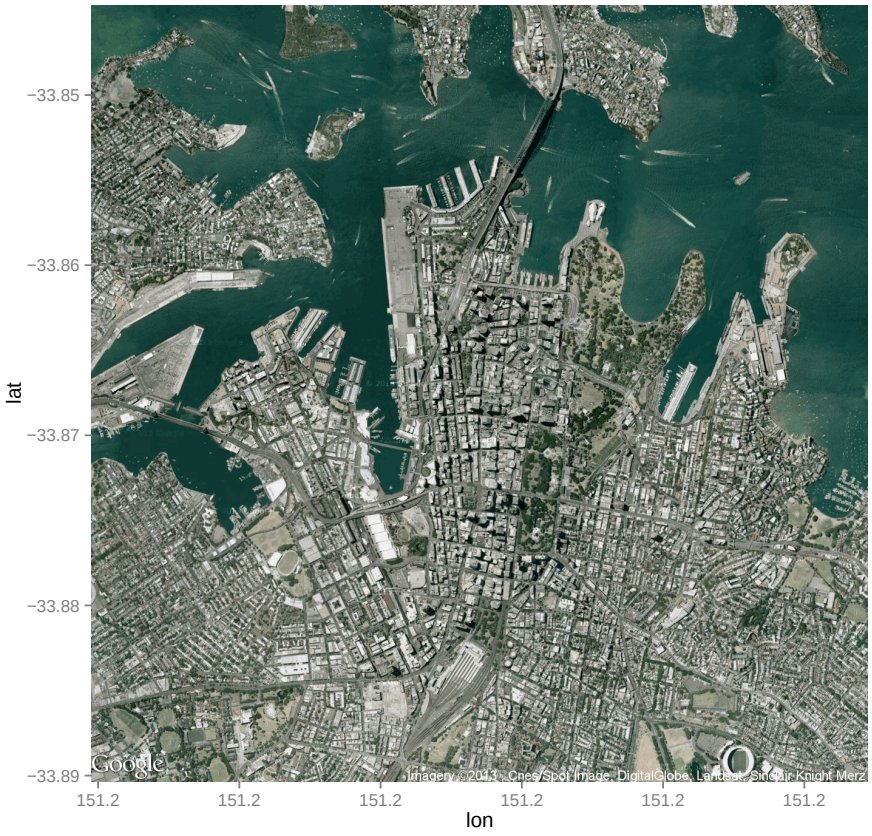
13 Google Terrain Map of Sydney



Here we see another type of map available from Google.

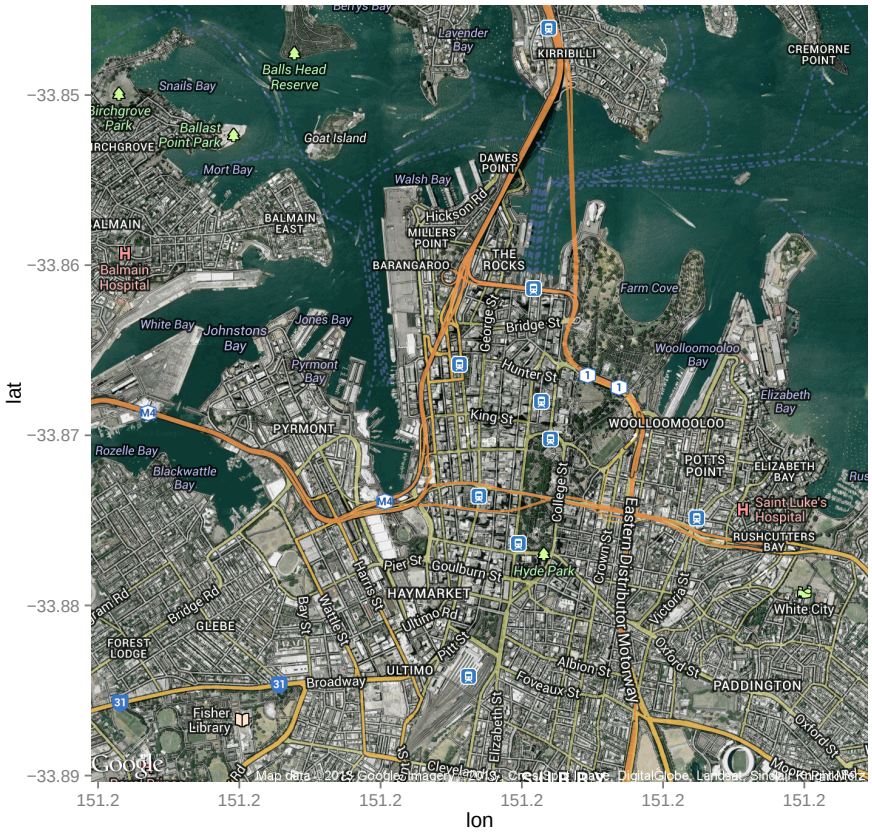
```
map <- get_map(location="Sydney", zoom=14, maptype="terrain", source="google")
p <- ggmap(map)
p
```

14 Google Satellite Map of Sydney



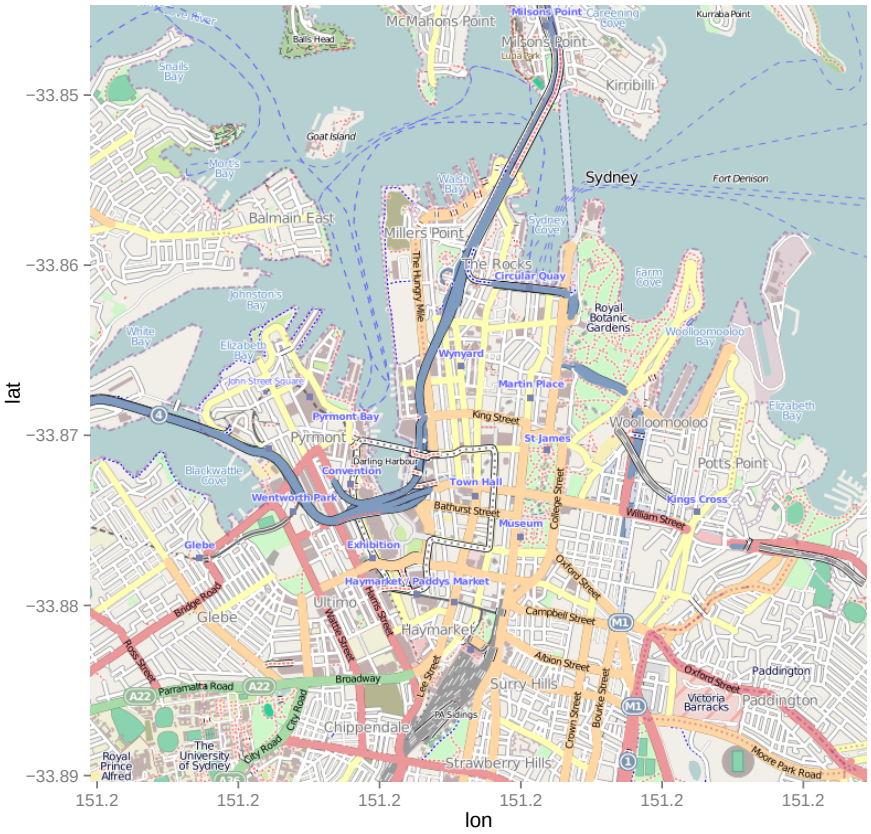

```
map <- get_map(location="Sydney", zoom=14, maptype="satellite", source="google")
p <- ggmap(map)
p
```

15 Google Hybrid Map of Sydney



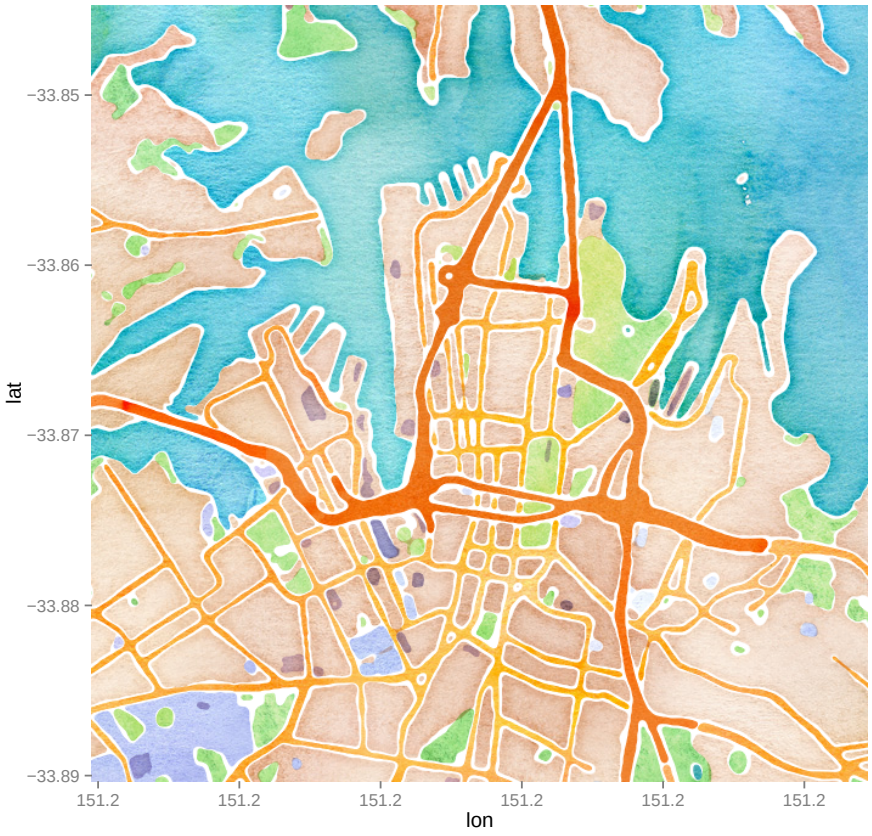

```
map <- get_map(location="Sydney", zoom=14, maptype="hybrid", source="google")
p <- ggmap(map)
p
```

16 OpenStreetMap of Sydney




```
map <- get_map(location="Sydney", zoom=14, source="osm")
p <- ggmap(map)
p
```

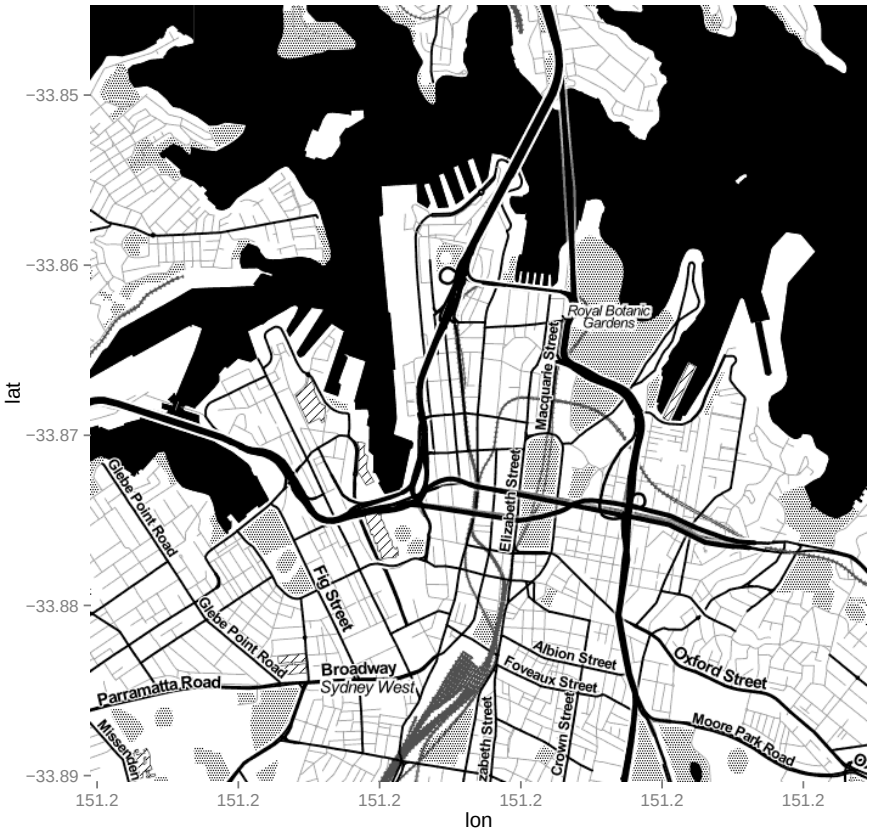
17 Stamen Watercolour Map of Sydney



Here we downloaded the map data for Sydney from other sources. This watercolour comes from Stamen.

```
map <- get_map(location="Sydney", zoom=14, maptype="watercolor", source="stamen")
p <- ggmap(map)
p
```

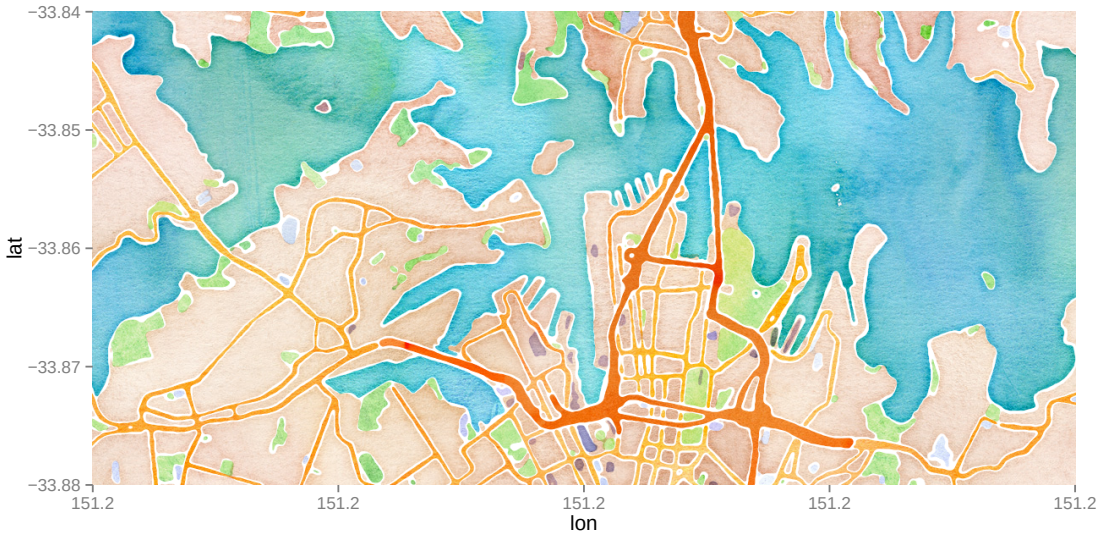
18 Stamen Toner Map of Sydney




```
map <- get_map(location="Sydney", zoom=14, maptype="toner", source="stamen")
p <- ggmap(map)
p
```

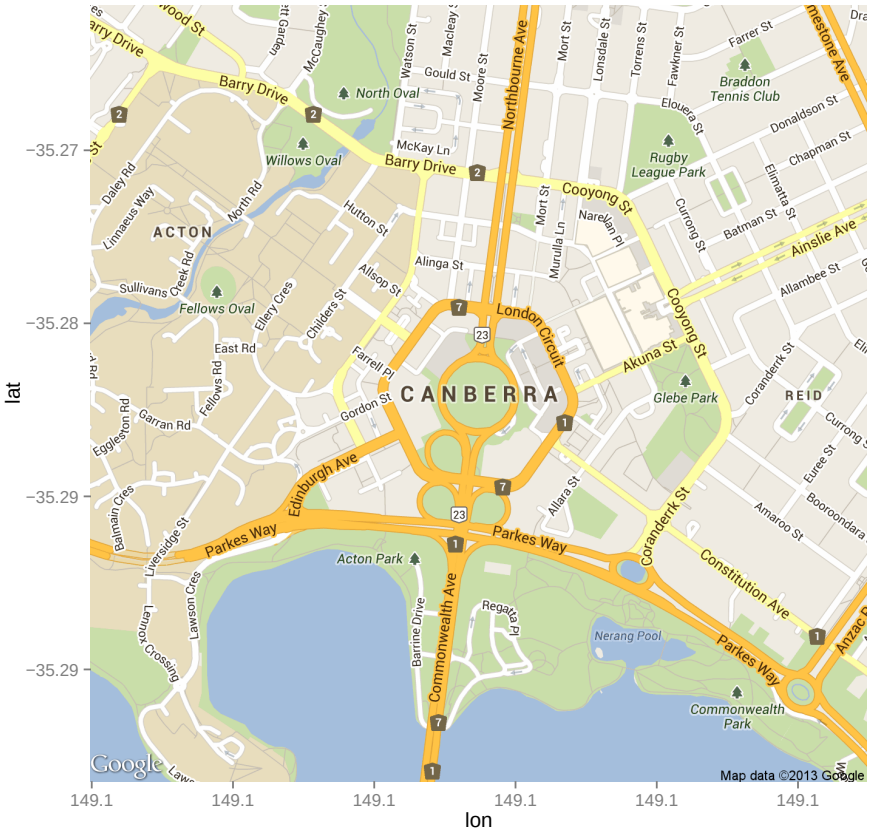
19 Bounding Box to Specify Sydney Map

Some sources also support bounding boxes rather than a centroid and zoom. Stamen, for example, supports bounding boxes, but Google does not. Here we use a bounding box (*sydbb*) that we defined earlier.



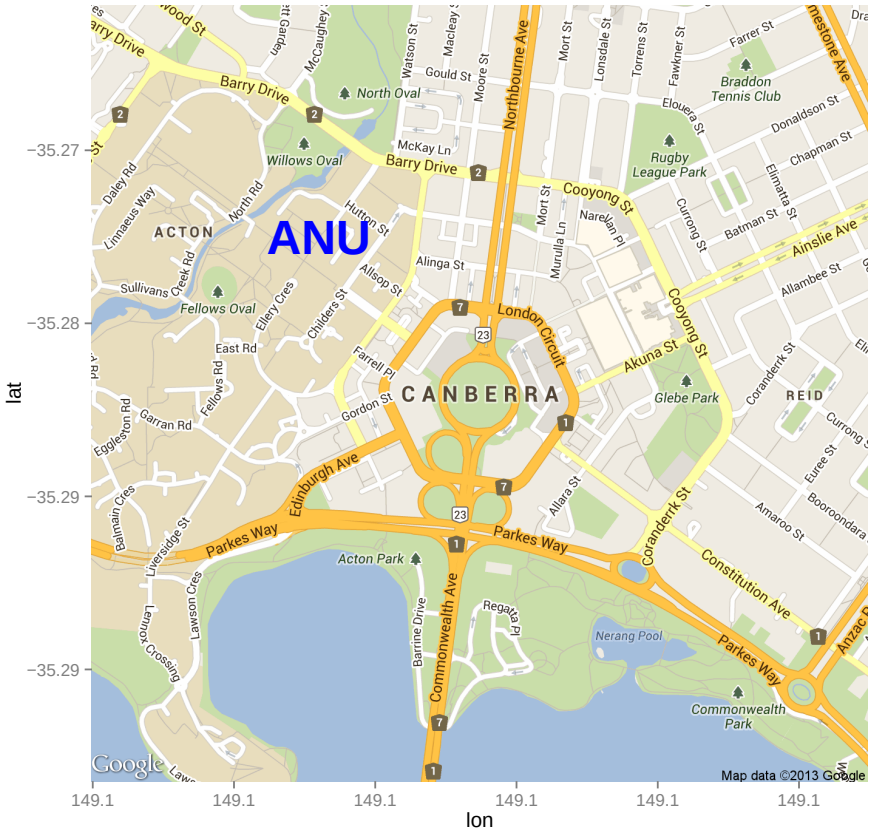

```
map <- get_map(location=sydbb, maptype="watercolor", source="stamen")
p <- ggmap(map)
p
```

20 Google Map of Canberra




```
map <- get_map(location="Canberra", zoom=15, maptype="roadmap", source="google")
p <- ggmap(map)
p
```

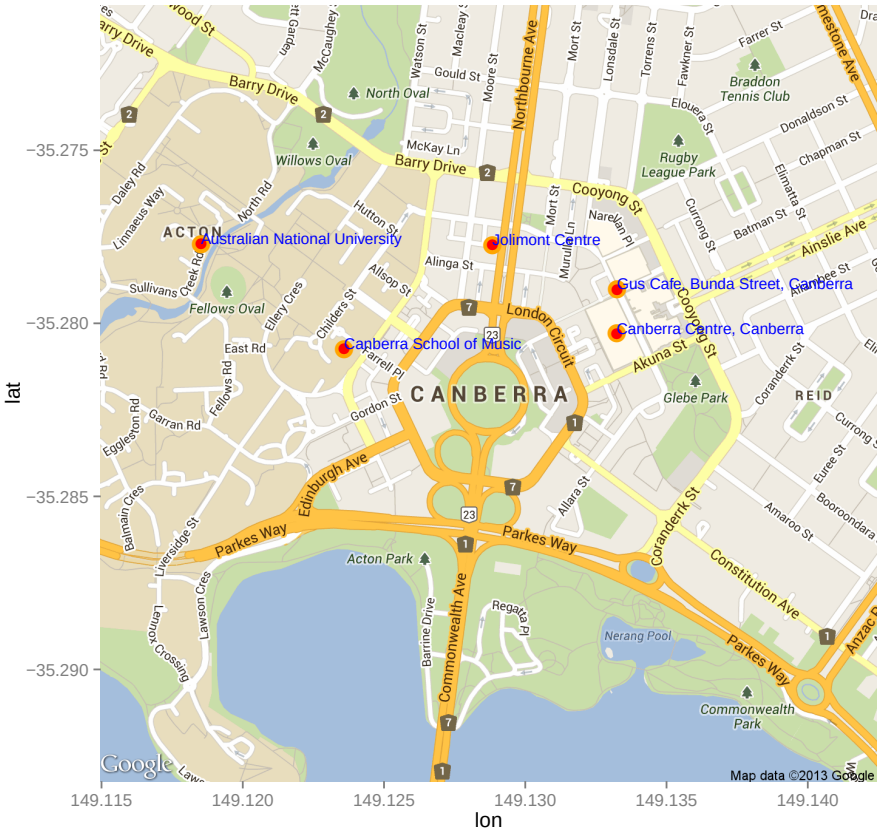
21 Annotating a Map with Text



Here we add a blue ANU label (Australian National University).

```
map <- get_map(location="Canberra", zoom=15, maptype="roadmap", source="google")
dflbl <- data.frame(lon=149.1230, lat=-35.2775, text="ANU")
p <- ggmap(map)
p <- p + geom_text(data=dflbl, aes(x=lon, y=lat, label=text),
                   size=10, colour="blue", fontface="bold")
p
```

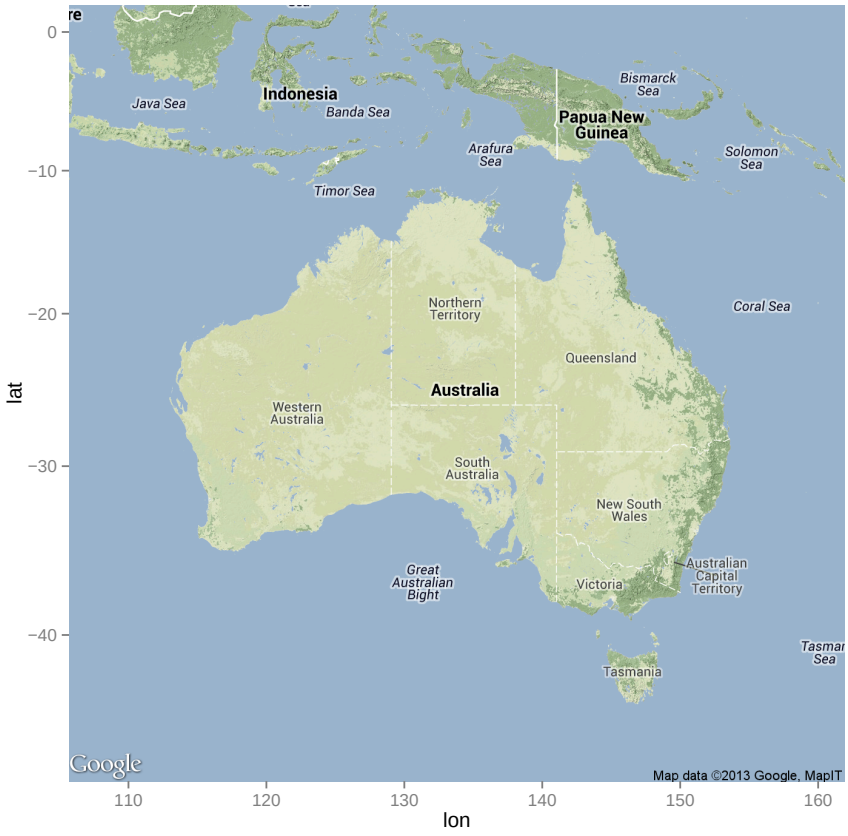
22 Annotating a Map with Landmarks



Here we have geocoded some landmarks and then added them to the map.

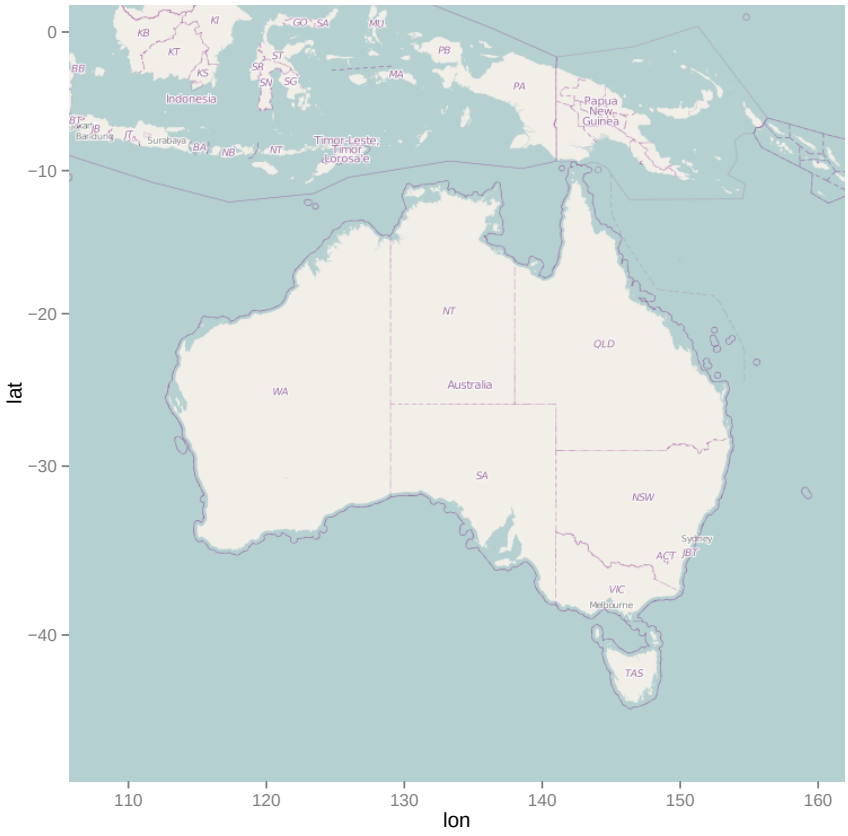
```
landmarks <- c("Gus Cafe, Bunda Street, Canberra", "Canberra Centre, Canberra",
               "Canberra School of Music", "Jolimont Centre",
               "Australian National University")
lbls <- cbind(geocode(landmarks), text=landmarks)
p <- ggmap(map)
p <- p + geom_point(data=lbls, aes(x=lon, y=lat), size=5, colour="orange")
p <- p + geom_point(data=lbls, aes(x=lon, y=lat), size=3, colour="red")
p <- p + geom_text(data=lbls, aes(x=lon, y=lat, label=text),
                   size=3, colour="blue", hjust=0, vjust=0)
p
```

23 Google Map of Australia



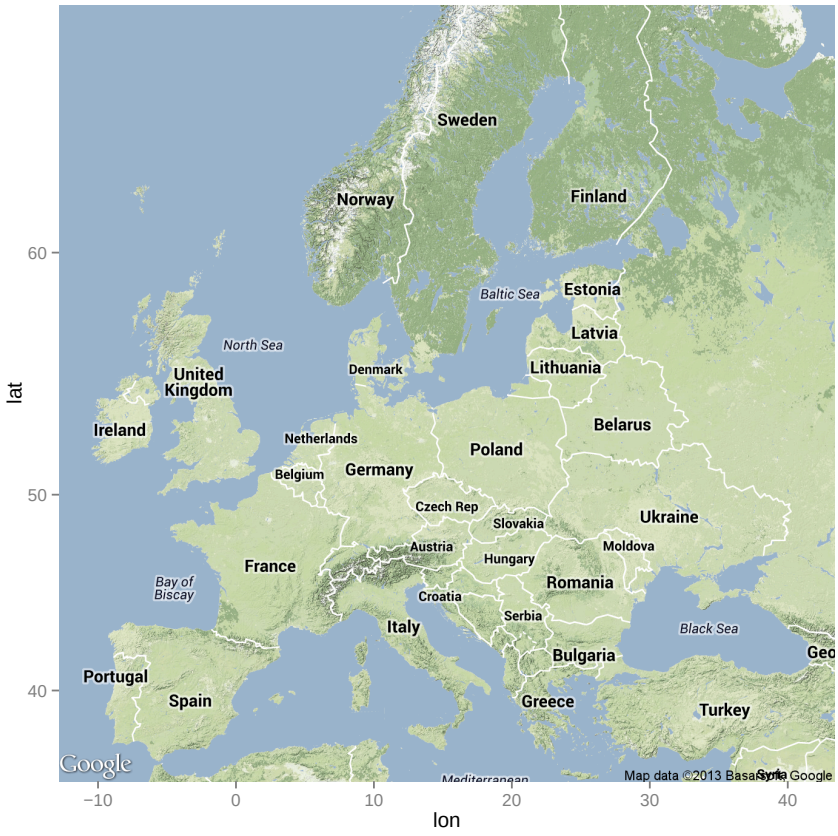

```
map <- get_map(location=as.numeric(geocode("Australia")),
               zoom=4, source="google")
p <- ggmap(map)
p
```

24 OpenStreetMap of Australia



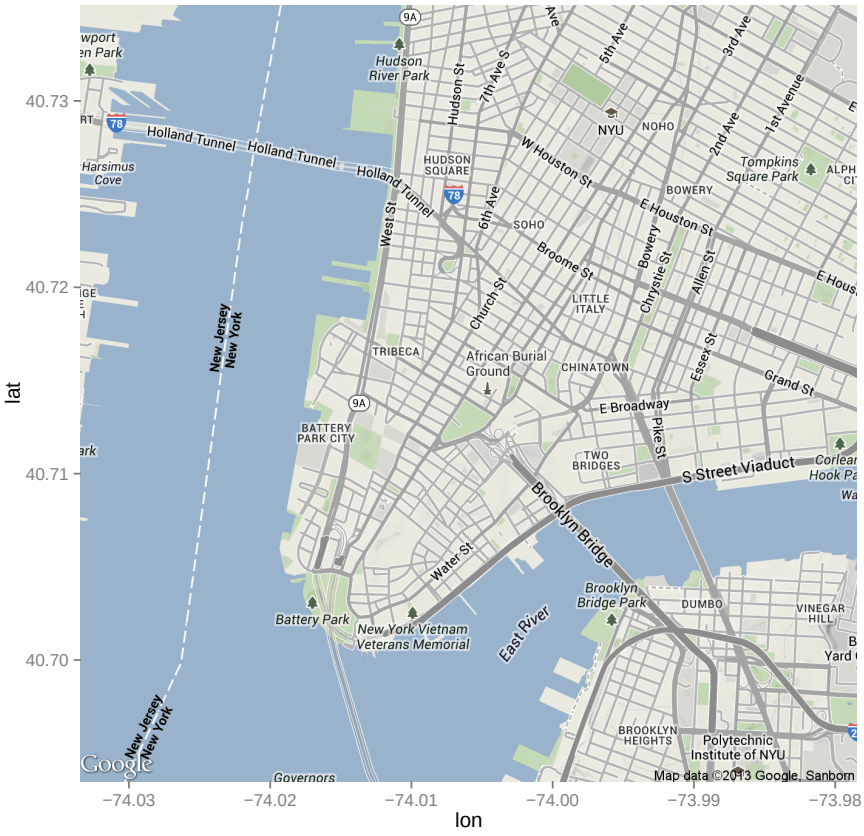

```
map <- get_map(location="Australia", zoom=4, source="osm")
p <- ggmap(map)
p
```

25 Google Map of Europe



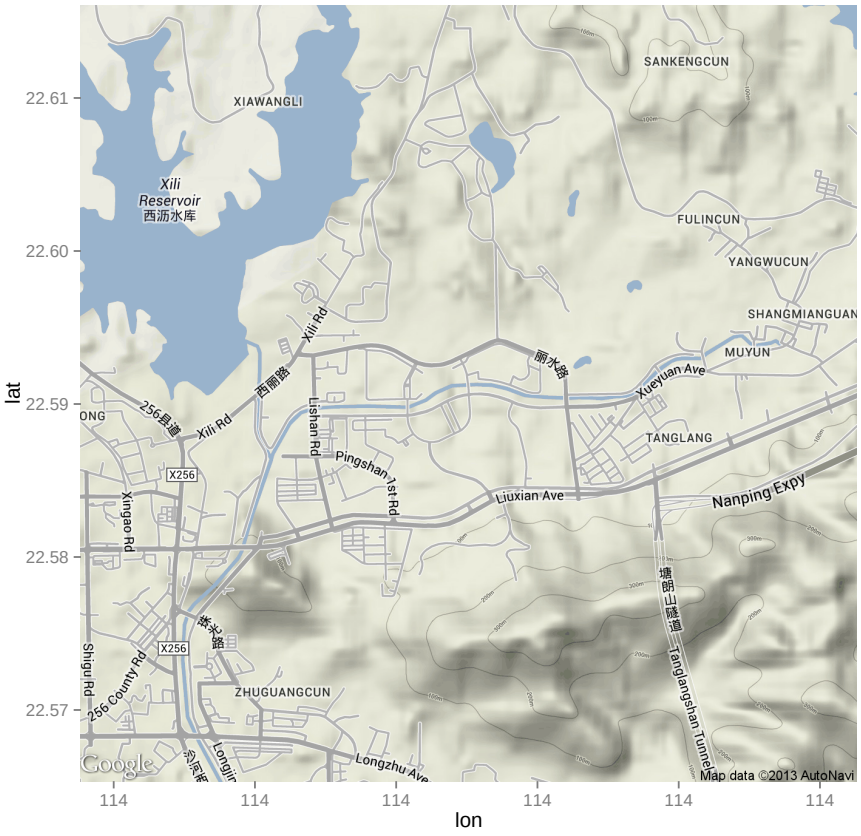

```
map <- get_map(location="Europe", zoom=4)
p <- ggmap(map)
p
```

26 New York: Google Maps



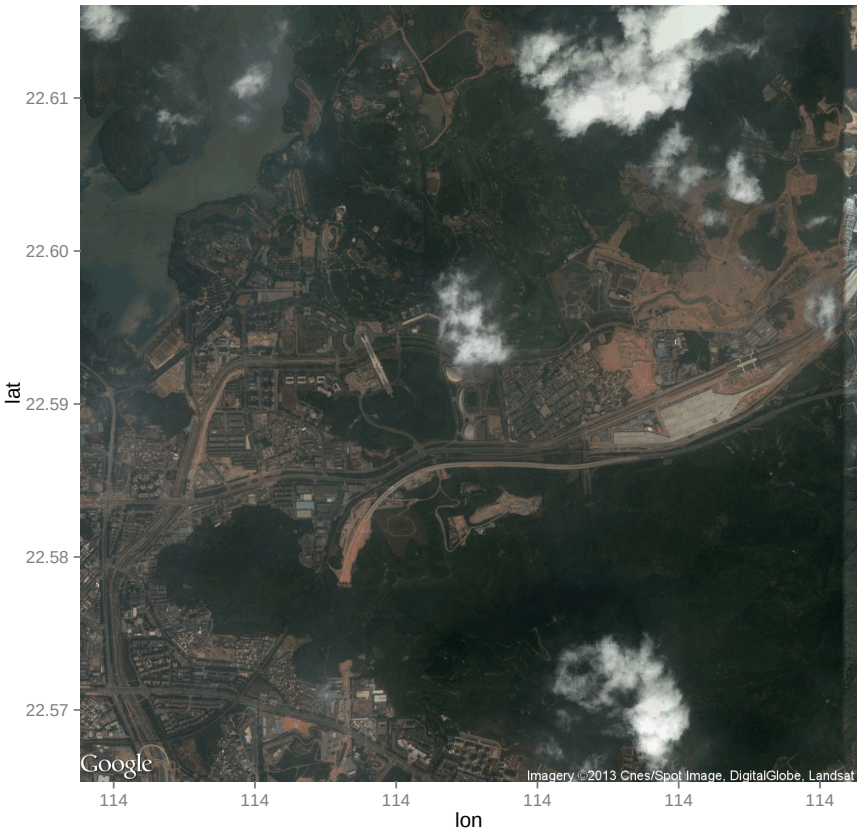

```
map <- get_map(location=as.numeric(geocode("New York")),
               zoom=14, source="google")
p <- ggmap(map)
p
```

27 Google Maps: Shenzhen University Town



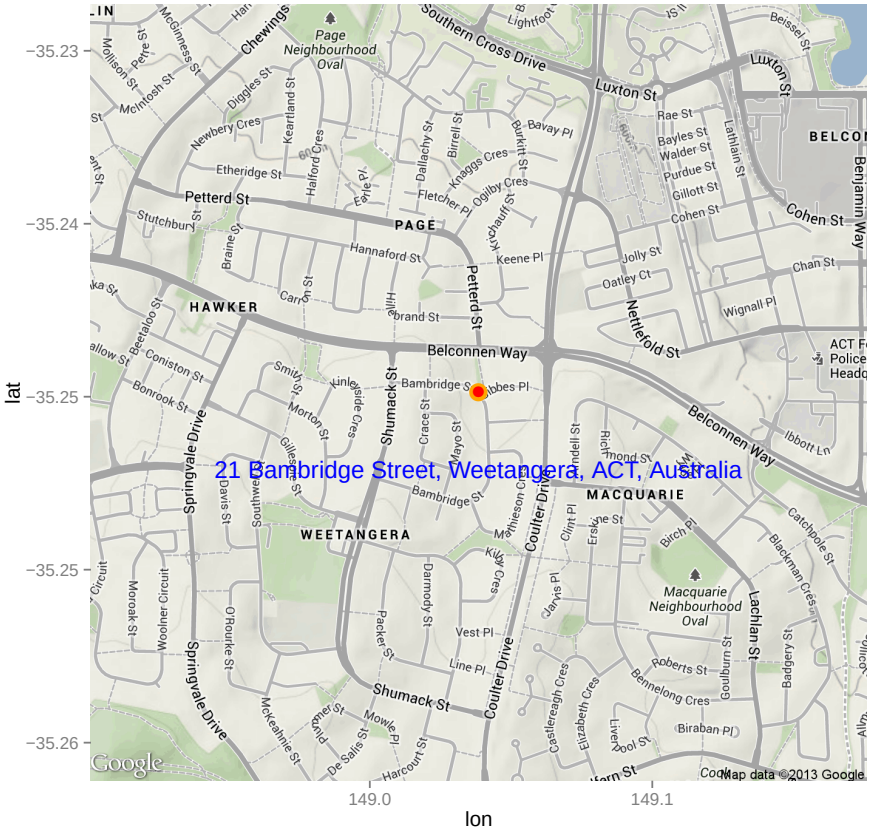

```
map <- get_map(location=as.numeric(geocode("Qiushi Road, Shenzhen")),
               zoom=14, source="google")
p <- ggmap(map)
p
```

28 Google Maps: Shenzhen Satellite




```
map <- get_map(location=as.numeric(geocode("Qiushi Road, Shenzhen")),
               zoom=14, maptype="satellite", source="google")
p <- ggmap(map)
p
```

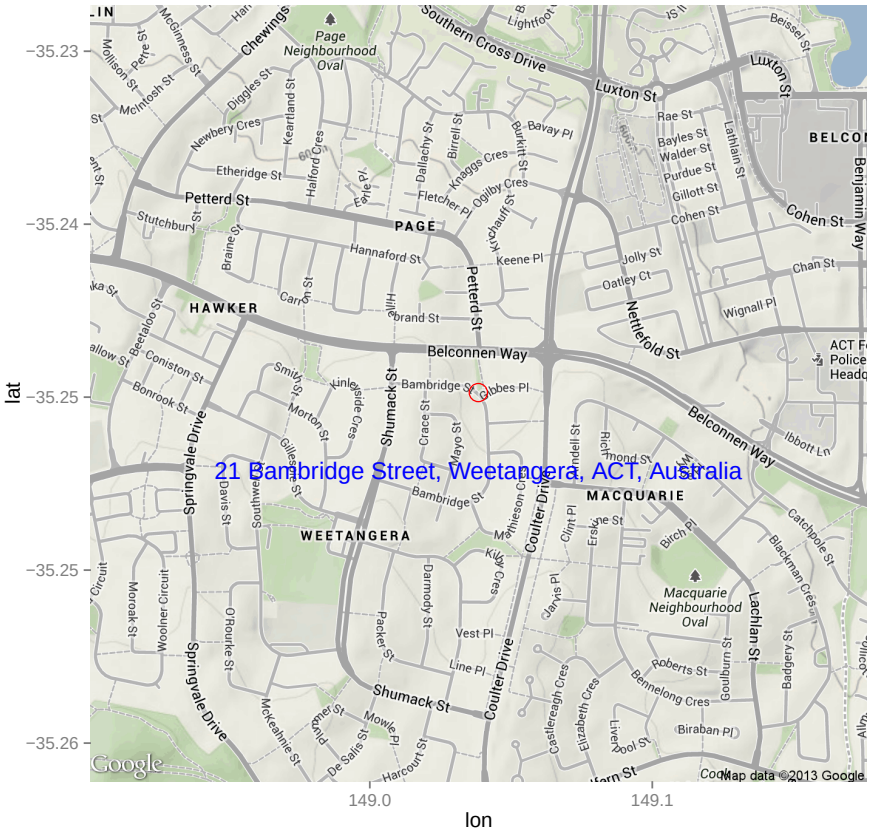
29 Plot an Address




```
addr <- "21 Bambridge Street, Weetangera, ACT, Australia"
loc <- as.numeric(geocode(addr))
lbl <- data.frame(lon=loc[1], lat=loc[2], text=addr)
map <- get_map(location=loc, zoom=15, source="google")
p <- ggmap(map)
p <- p + geom_point(data=lbl, aes(x=lon, y=lat), size=5, colour="orange")
p <- p + geom_point(data=lbl, aes(x=lon, y=lat), size=3, colour="red")
p <- p + geom_text(data=lbl, aes(x=lon, y=lat, label=text),
                   size=5, colour="blue", hjust=0.5, vjust=5)
p
```

The location is pinpointed with a red dot overlaying an orange dot. We also add the actual address to the plot.

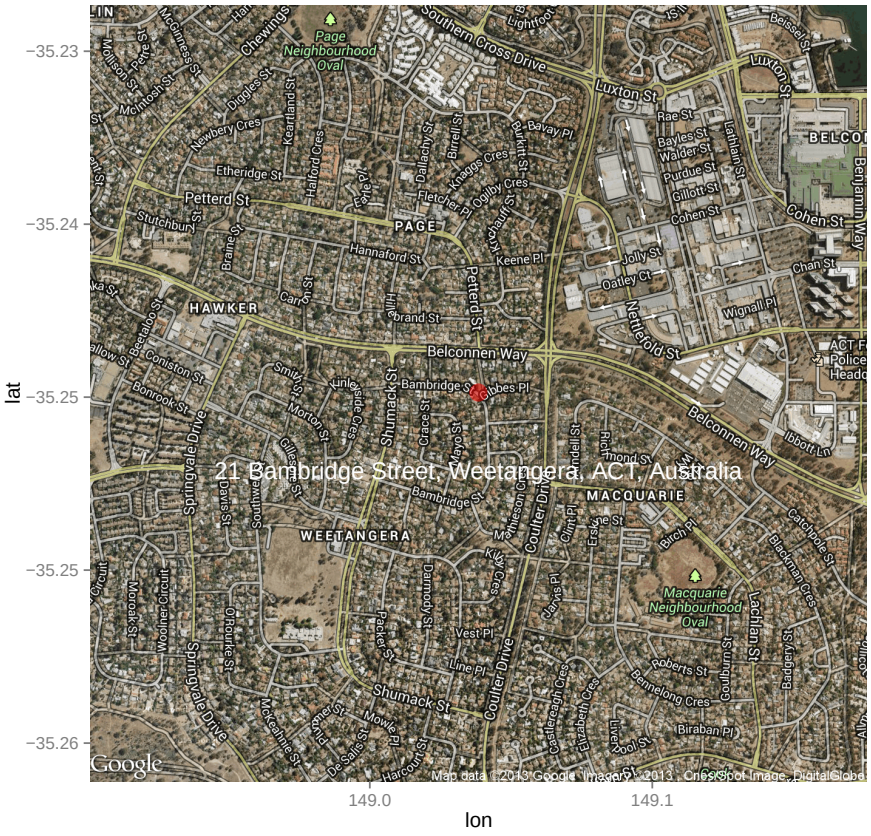
30 Plot an Address Using Circle




```
addr <- "21 Bambridge Street, Weetangera, ACT, Australia"
loc <- as.numeric(geocode(addr))
lbl <- data.frame(lon=loc[1], lat=loc[2], text=addr)
map <- get_map(location=loc, zoom=15, source="google")
p <- ggmap(map)
p <- p + geom_point(data=lbl, aes(x=lon, y=lat), size=5,
                    shape=1, colour="red")
p <- p + geom_text(data=lbl, aes(x=lon, y=lat, label=text),
                   size=5, colour="blue", hjust=0.5, vjust=5)
p
```

Here we have replaced the dot with a circle to pinpoint the location, using `shape=1` to choose a circle rather than a filled dot.

31 Plot an Address on a Satellite Image




```
addr <- "21 Bambridge Street, Weetangera, ACT, Australia"
loc <- as.numeric(geocode(addr))
lbl <- data.frame(lon=loc[1], lat=loc[2], text=addr)
map <- get_map(location=loc, zoom=15, maptype="hybrid", source="google")
p <- ggmap(map)
p <- p + geom_point(data=lbl, aes(x=lon, y=lat),
                    alpha=I(0.5), size=I(5), colour="red")
p <- p + geom_text(data=lbl, aes(x=lon, y=lat, label=text),
                   size=5, colour="white", hjust=0.5, vjust=5)
p
```

The underlying map is now a hybrid satellite and road map, with a transparent red dot to pinpoint the location. The transparency is controlled by `alpha=I(0.5)`.

32 USA Arrests: Assaults per Murder Data

This example comes from the help page for `map_data()` from ggplot2 (Wickham and Chang, 2014). It shows the number of assaults per murder in each US state, though it is quite easy to modify the code to display various statistics from the data.

First we take a copy of the **USArrests** dataset and lowercase the variables and the state names to make the matching across different datasets uniform.

```
arrests <- USArrests
names(arrests) <- tolower(names(arrests))
arrests$region <- tolower(rownames(USArrests))
head(arrests)
```

```
##            murder assault urbanpop rape     region
## Alabama      13.2     236       58 21.2    alabama
## Alaska       10.0     263       48 44.5     alaska
## Arizona       8.1     294       80 31.0    arizona
....
```

Then we merge the statistics with the spatial data, in readiness for mapping.

```
states <- map_data("state")
head(states)
```

```
##     long   lat group order  region subregion
## 1 -87.46 30.39     1     1 alabama      <NA>
## 2 -87.48 30.37     1     2 alabama      <NA>
## 3 -87.53 30.37     1     3 alabama      <NA>
....
```

```
ds <- merge(states, arrests, sort=FALSE, by="region")
head(ds)
```

```
##    region   long   lat group order subregion murder assault urbanpop rape
## 1 alabama -87.46 30.39     1     1      <NA>   13.2     236       58 21.2
## 2 alabama -87.48 30.37     1     2      <NA>   13.2     236       58 21.2
## 3 alabama -87.95 30.25     1    13      <NA>   13.2     236       58 21.2
....
```

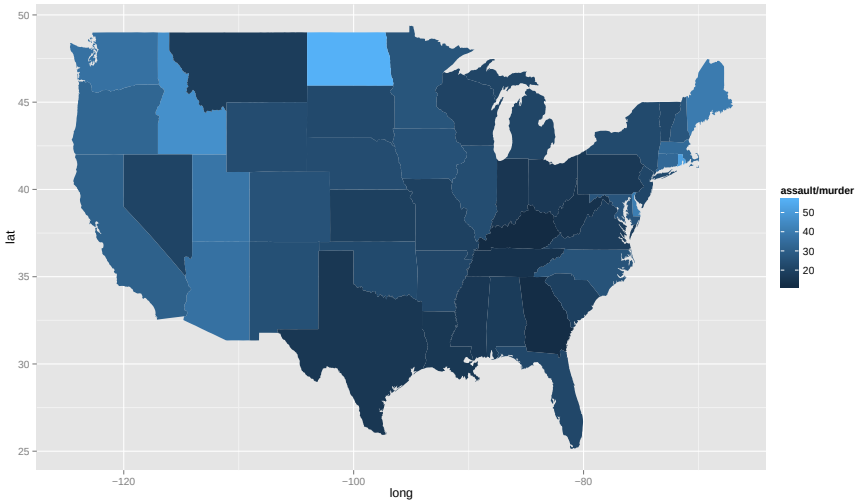
```
ds <- ds[order(ds$order), ]
head(ds)
```

```
##    region   long   lat group order subregion murder assault urbanpop rape
## 1 alabama -87.46 30.39     1     1      <NA>   13.2     236       58 21.2
## 2 alabama -87.48 30.37     1     2      <NA>   13.2     236       58 21.2
## 6 alabama -87.53 30.37     1     3      <NA>   13.2     236       58 21.2
....
```

33 USA Arrests: Assaults per Murder Map

Once we have the data ready, plotting it simply requires nominating the dataset, and identifying the x and y as long and lat respectively. We also need to identify the grouping, which is by state, and so the fill is then specified for each state to indicate the statistic of interest.

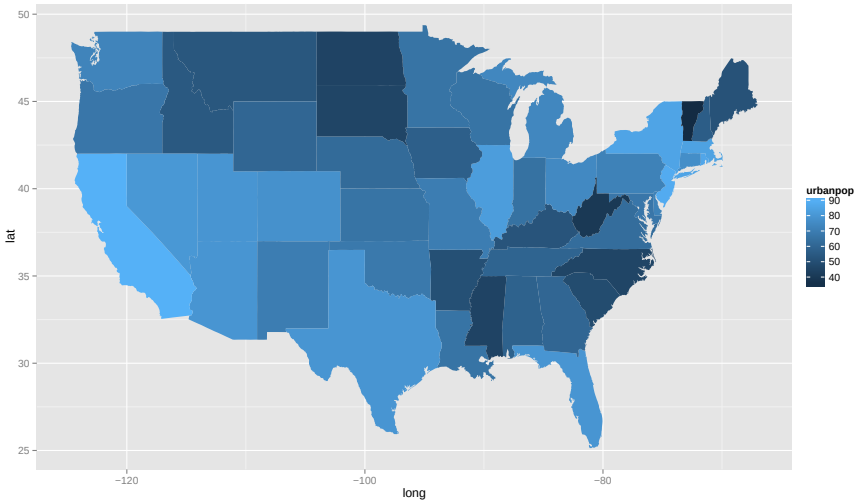
```
g <- ggplot(ds, aes(long, lat, group=group, fill=assault/murder))
g <- g + geom_polygon()
print(g)
```



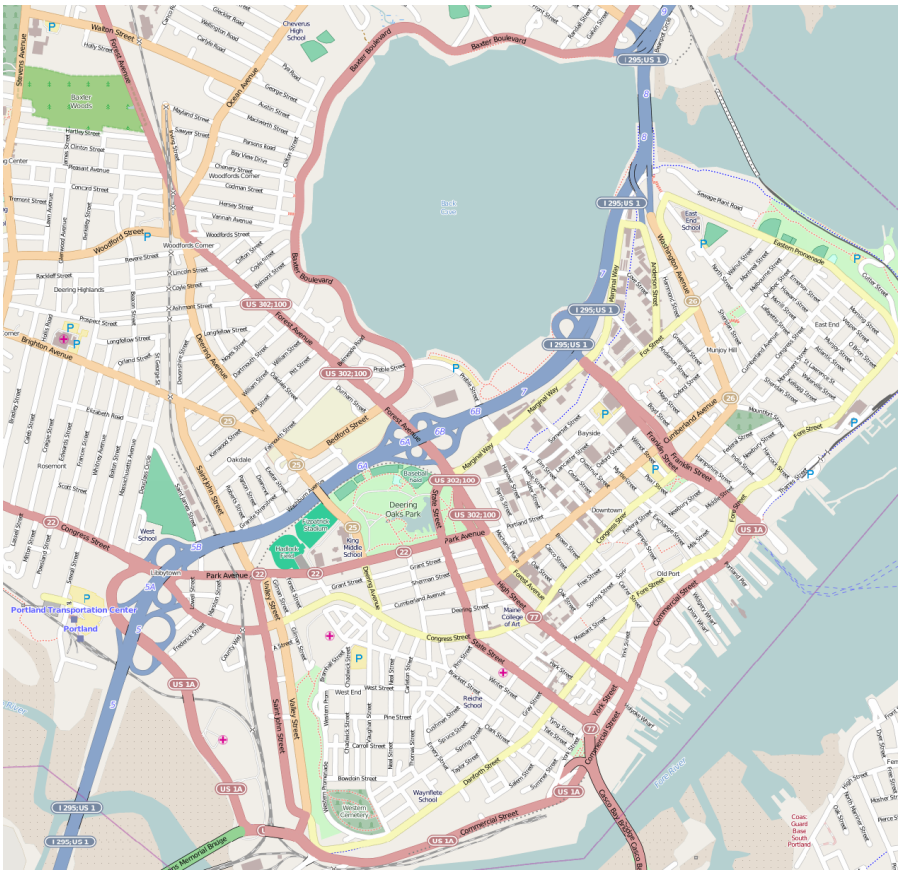
We might also be interested in plotting the percentage of urban population in each state.

Exercise: How can we change the colour of the scale?

```
g <- ggplot(ds, aes(long, lat, group=group, fill=urbanpop))
g <- g + geom_polygon()
print(g)
```

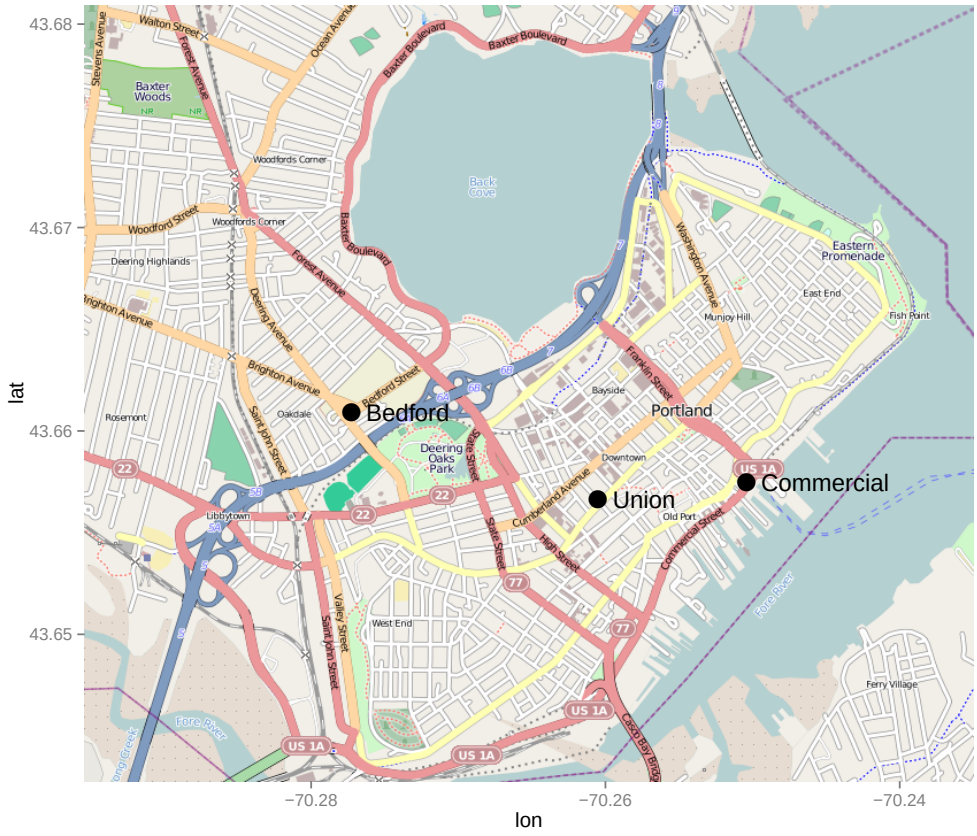


34 Portland: Open Street Maps




```
library(OpenStreetMap)
stores <- data.frame(name=c("Commercial","Union","Bedford"),
                     lon=c(-70.25042295455, -70.26050806045, -70.27726650238),
                     lat=c(43.657471302616, 43.65663299041, 43.66091757424))
lat <- c(43.68093, 43.64278)
lon <- c(-70.29548, -70.24097)
portland <- openmap(c(lat[1],lon[1]),c(lat[2],lon[2]), zoom=15, 'osm')
plot(portland, raster=TRUE)
```

35 Portland: Annotated Maps

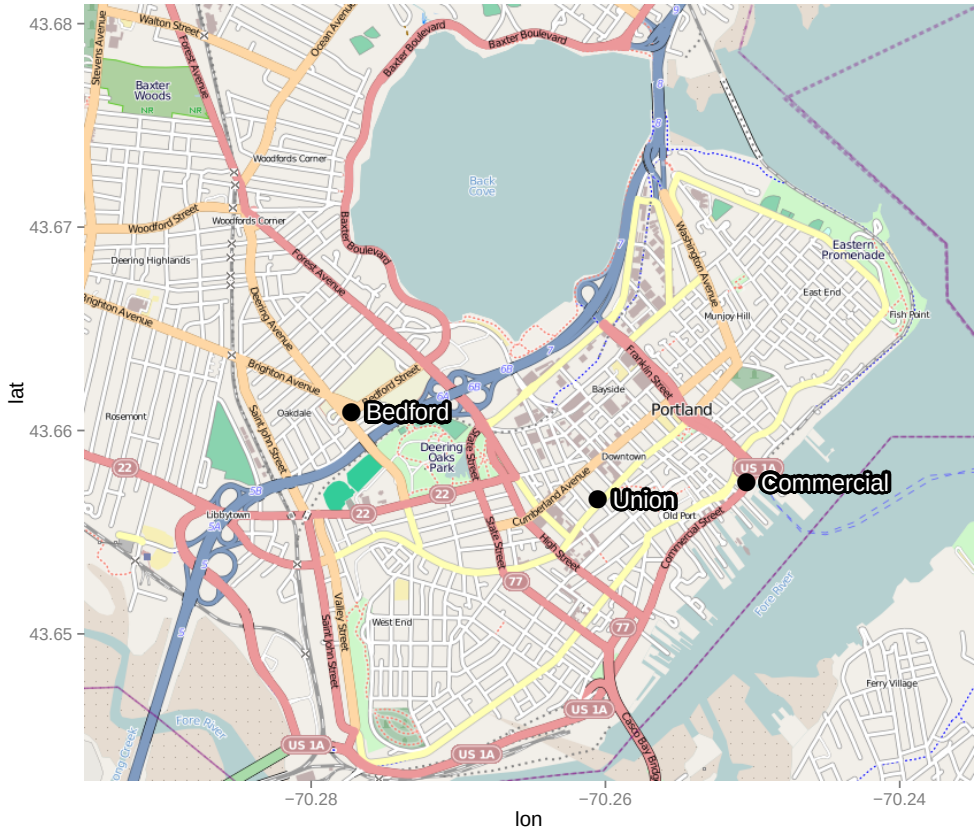


From http://stackoverflow.com/questions/10686054/outlined-text-with-ggplot2

```
stores <- data.frame(name=c("Commercial", "Union", "Bedford"),
                     lon=c(-70.25042295455, -70.26050806045, -70.27726650238),
                     lat=c( 43.65747130261,  43.65663299041,  43.66091757424))

portland <- c(-70.2954, 43.64278, -70.2350, 43.68093)
```

```
library(ggmap)
map <- get_map(location=portland, source="osm")

g <- ggmap(map)
g <- g + geom_point(data=stores, aes(x=lon, y=lat), size=5)
g <- g + geom_text(data=stores, aes(label=name, x=lon+.001, y=lat), hjust=0)
print(g)
```

36 Portland: Standout Text Annotation



This map uses outlined text to ensure we can better read text on the map.

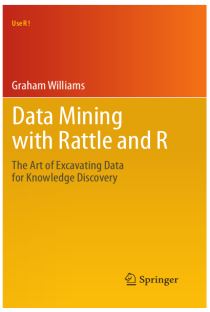
```
g <- ggmap(map)
g <- g + geom_point(data=stores, aes(x=lon, y=lat), size=5)

theta <- seq(pi/16, 2*pi, length.out=32)
xo <- diff(portland[c(1,3)])/250
yo <- diff(portland[c(2,4)])/250

for(i in theta)
  g <- g + geom_text(data=stores, bquote(aes(x=lon + .001 + .(cos(i) * xo),
                        y=lat + .(sin(i) * yo), label = name)),
                     size=5, colour='black', hjust=0)

g <- g + geom_text(data=stores, aes(x=lon+.001, y=lat, label=name),
                   size=5, colour='white', hjust=0)
print(g)
```

37 Animated Maps

See http://rmaps.github.io/blog/posts/animated-choropleths/.

38 Further Reading and Acknowledgements

The Rattle Book, published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from Amazon. Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from `http://datamining.togaware.com`, including the Datamining Desktop Survival Guide.



This chapter is one of many chapters available from `http://HandsOnDataScience.com`. In particular follow the links on the website with a * which indicates the generally more developed chapters.

Other resources include:

- The packages RgoogleMaps (Loecher, 2014) and OpenStreetMap (Fellows and using the JMapViewer library by Jan Peter Stotz, 2013) provide alternative interfaces to Google Maps, etc., whereby we can download maps and annotate them. Much of the functionality we have already seen is provided by ggmap (Kahle and Wickham, 2013). OpenStreetMap provides `automap()`. RgoogleMaps provides `GetMap()` to download a map from Google, and `PlotOnStaticMap()` for displaying the map and overlaying plots on the map.
- The package osmar (Schlesinger and Eugster, 2013) provides an interactive environment based on OpenStreetMap.

Tony's ToDo
stats between locations like distance
calculate an area on a map
colour code regions
regions like ABS has
ability to plot graphs either on map or in a new window with stats as meta data
ability to have a reigon shaed with locations pinpointed on top
dialogue box which describes data on location
have images on a map, and you can filter out, with a simple scroll option.
a 3d options on a property like bing maps has
the ability to make a move to fly through way points on a map
radius zones around a point.
radius points around multiple points, like a distorted ven diagram
stats that change over time like a movie but static location, but graphs change
time change colour for night and day or both

39 References

Brownrigg R (2014). *maps: Draw Geographical Maps*. R package version 2.3-7, URL http://CRAN.R-project.org/package=maps.

Fellows I, using the JMapViewer library by Jan Peter Stotz (2013). *OpenStreetMap: Access to open street map raster images*. R package version 0.3.1, URL http://CRAN.R-project.org/package=OpenStreetMap.

Kahle D, Wickham H (2013). *ggmap: A package for spatial visualization with Google Maps and OpenStreetMap*. R package version 2.3, URL http://CRAN.R-project.org/package=ggmap.

Loecher M (2014). *RgoogleMaps: Overlays on Google map tiles in R*. R package version 1.2.0.6, URL http://CRAN.R-project.org/package=RgoogleMaps.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Schlesinger T, Eugster MJA (2013). *osmar: OpenStreetMap and R*. R package version 1.1-7, URL http://CRAN.R-project.org/package=osmar.

Wickham H, Chang W (2014). *ggplot2: An implementation of the Grammar of Graphics*. R package version 1.0.0, URL http://CRAN.R-project.org/package=ggplot2.

Williams GJ (2009). "Rattle: A Data Mining GUI for R." *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.

Williams GJ (2014). *rattle: Graphical user interface for data mining in R*. R package version 3.1.4, URL http://rattle.togaware.com/.

This document, sourced from Maps.Rnw revision 501, was processed by KnitR version 1.6 of 2014-05-24 and took 66.9 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.1 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-08-19 20:35:16.