Spatial, Temporal and Spatio-Temporal Data in R

L. Torgo

ltorgo@fc.up.pt
Faculdade de Ciências / LIAAD-INESC TEC, LA
Universidade do Porto

Jan, 2017

Jožef Stefan Institute

Motivation

- We leave in a world where data collection devices abound
- Frequently these devices are location- and time-aware (e.g. smartphones)
- Data sets where observations are tagged by time and/or location are increasing at a fast rate

Motivation - 2

- If we suspect our observations my have some form of time and or space correlation we should not neglect these effects in our analysis
  - Do observations of X at time t depend on the values at recent past times ?
  - Do observations of X at location z depend on values at neighboring locations ?
  - Do observations of X at location z on time t depend on the values at neighboring locations currently or in recent past times?

Jožef Stefan Institute

Some Tasks/Problems Addressed in Spatio-Temporal Data Mining

- Exploratory analysis of the data with the goal of understanding the eventual spatial and temporal patterns
- Prediction
  - Spatial interpolation
  - Time series forecasting
  - Spatio-temporal forecasting

Jožef Stefan Institute

Why do we need something specific?

- Spatial coordinates are just numbers...
- Temporal tags can be seen as strings following some rules...
- Special purpose classes facilitate the manipulation of this type of data
- e.g. temporal or spatial queries on a data set

Jožef Stefan Institute

Some Packages and Classes

- Temporal data
  - Package **xts**
- Spatial data
  - Package **sp**
- Spatio-temporal data
  - Package **spacetime**

Jožef Stefan Institute

Handling Time Series in R

- R includes several data structures that can be used to store time series
- In this illustration we will use the infra-structured provided in package `xts`
  Note: this is an extra package that must be installed.
- The function `xts()` can be used to create a time series,

```
library(lubridate)
library(xts)
sp500 <- xts(c(1102.94,1104.49,1115.71,1118.31),
             ymd(c("2010-02-25","2010-02-26","2010-03-01","2010-03-02")))
sp500

##                [,1]
## 2010-02-25 1102.94
## 2010-02-26 1104.49
## 2010-03-01 1115.71
## 2010-03-02 1118.31
```


Creating time series

- The function `xts` has 2 arguments: the time series values and the temporal tags of these values
- The second argument must contain dates
- The function `ymd()` from package **lubridate** can be used to convert strings into dates
- If we supply a matrix on the first argument we will get a multivariate time series, with each column representing one of the variables

Indexing Time Series

- We may index the objects created by the function `xts()` as follows,

```
sp500[3]

##                  [,1]
## 2010-03-01 1115.71
```

- However, it is far more interesting to make “temporal queries”,

```
sp500["2010-03-02"]

##                  [,1]
## 2010-03-02 1118.31

sp500["2010-03"]

##                  [,1]
## 2010-03-01 1115.71
## 2010-03-02 1118.31
```


Indexing Time Series (2)

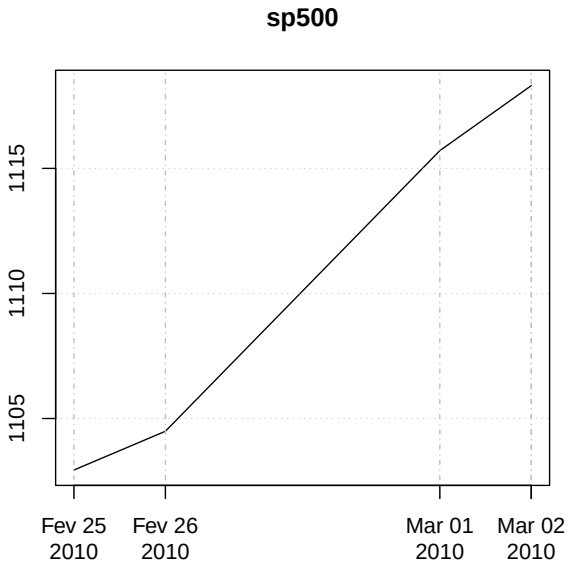
```
sp500["2010-02-26/"]

##                  [,1]
## 2010-02-26 1104.49
## 2010-03-01 1115.71
## 2010-03-02 1118.31

sp500["2010-02-26/2010-03-01"]

##                  [,1]
## 2010-02-26 1104.49
## 2010-03-01 1115.71
```

- The index is a string that may represent intervals using the symbol `/` or by omitting part of a date. You may also use `::` instead of `/`.

Temporal Plots

- The `plot()` function can be used to obtain a temporal plot of a time series
- R takes care of selecting the proper axes,

```
plot(sp500)
```



Jožef Stefan Institute

Hands On Time Series

Package **quantmod** (an extra package that you need to install) contains several facilities to handle financial time series. Among them, the function `getMetals` allows you to download the prices of metals from `oanda.com`. Explore the help page of the function to try to understand how it works, and the answer the following:

1. Obtain the prices of gold of the current year
2. Show the prices in January
3. Show the prices from February 10 till March 15
4. Obtain the prices of silver in the last 30 days
   Tip: explore the function `seq.Date()`
5. Plot the prices of silver in the last 7 days
   Tip: explore the function `last()` on package **xts**

Jožef Stefan Institute

Geo-referenced Data in R

- Package **sp** defines a series of classes of objects that can be used to store geo-referenced data sets
- Many other R packages build upon the classes defined in this package
- A broad overview of the (many) packages existing in R for this type of data can be found in
  Analysis of Spatial Data Task View

Some of the Classes Defined in **sp**

- `SpatialPointsDataFrame`
  A data frame like structure that can be used to store data (values of a set of variables) about geographic locations (a spatial point)
- `SpatialGridDataFrame`
  A data frame like structure that can be used to store data (values of a set of variables) about a regular grid of spatial points
- `SpatialLinesDataFrame` and `SpatialPolygonsDataFrame`
  A data frame like structure that can be used to store data (values of a set of variables) about lines and polygons (closed sequences of lines)

A Simple Illustration

Fires data of 500m^2 regions of Portugal

- Official data on fires in different regions of Portugal
- The data set we will use contains information on 25000 locations

```
df <- read.csv("firesnew_25000_500m.txt")
```

```
df[1:3,]
```

```
##   FID_ CID ano1991 ano1992 ano1993 ano1994 ano1995 ano1996 ano1997 ano1998
## 1   NA   1       0       0       0       0       0       0       0       0
## 2   NA   2       0       0       0       0       0       0       0       0
## 3   NA   3       0       0       0       0       0       0       0       0
##   ano1999 ano2000        x       y
## 1       0       0 -7.31924 38.5406
## 2       0       0 -7.63557 40.5022
## 3       0       0 -7.90273 40.3418
```

Jožef Stefan Institute

A Simple Illustration

Creating a `SpatialPointsDataFrame` object with 2000 data

```
library(sp)
spatialCoords <- cbind(long=df$x, lat=df$y)
coordRefSys <- CRS("+proj=longlat +ellps=WGS84")
fires2000 <- SpatialPointsDataFrame(spatialCoords,
                                    df[, "ano2000", drop=FALSE],
                                    proj4string=coordRefSys)
fires2000[1:3,]
```

```
##           coordinates ano2000
## 1 (-7.31924, 38.5406)       0
## 2 (-7.63557, 40.5022)       0
## 3 (-7.90273, 40.3418)       0
```

Jožef Stefan Institute

A Simple Illustration

Some utility functions

```
bbox(fires2000)
```

```
##           min      max
## long -9.49174 -6.20743
## lat  36.98050 42.14360
```

```
coordinates(fires2000)[1:3,]
```

```
##          long     lat
## [1,] -7.31924 38.5406
## [2,] -7.63557 40.5022
## [3,] -7.90273 40.3418
```

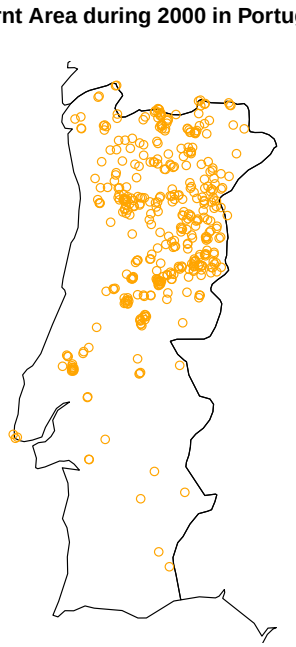
```
summary(fires2000)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##           min      max
## long -9.49174 -6.20743
## lat  36.98050 42.14360
## Is projected: FALSE
## proj4string : [+proj=longlat +ellps=WGS84]
## Number of points: 25000
## Data attributes:
##     ano2000
## Min.   :0.00000
##  Median :0.00000
##  Mean   :0.01612
##  3rd Qu.:0.00000
```


A Simple Illustration

Visualizing the data

```
library(rworldmap)
m <- getMap(resolution="low")
b <- bbox(fires2000)
plot(m,xlim=b[1,],ylim=b[2,])
points(fires2000[fires2000$ano2000==1,],
       col="orange")
title("Burnt Area during 2000 in Portugal")
```



Jožef Stefan Institute

Spatio-temporal Data with **spacetime**

- Spatio-temporal data frequent formats
  - **Time-wide** - different columns have different measurements across time
  - **Space-wide** - different columns have measurements across different locations
  - **Long formats** - each data record contains measurements for a space-time combination
- Our previous fires data example used a time-wide format

Jožef Stefan Institute

Spatio-temporal Data with **spacetime** (cont.)

- Some types of spatio-temporal data sets
  - **Full grids** - we have values for all combinations of location and time stamps
    number of values = number of locations $\times$ number of time stamps
  - **Sparse grids** - similar but only non-missing values are stored
  - **Irregular data** - Each value is measured on a certain location and time, without regularity
  - **Time intervals, moving objects and trajectories** - e.g. spatial features constant but values collected on a time interval; trajectories where irregular space-time points form a sequence; etc.

Jožef Stefan Institute

Spatio-temporal Data with **spacetime** (cont.)

- The package **spacetime** defines different classes for these situations
  - **Full grids** - class `STFDF`
  - **Sparse grids** - class `STSDF`
  - **Irregular data** - class `STIDF`
  - **Time intervals, moving objects and trajectories** - class `STTDF`

Jožef Stefan Institute

A Simple Illustration with the Fires Data

The data again...

```
df <- read.csv("firesnew_25000_500m.txt")
```

```
df[1:3,]
```

```
##   FID_ CID ano1991 ano1992 ano1993 ano1994 ano1995 ano1996 ano1997 ano1998
## 1   NA   1       0       0       0       0       0       0       0       0
## 2   NA   2       0       0       0       0       0       0       0       0
## 3   NA   3       0       0       0       0       0       0       0       0
##   ano1999 ano2000        x       y
## 1       0       0 -7.31924 38.5406
## 2       0       0 -7.63557 40.5022
## 3       0       0 -7.90273 40.3418
```

Jožef Stefan Institute

Creating a STFDF object

- First let us put the data in Long Format

```
library(tidyr)
x <- gather(df[,3:14],variable, value, ano1991:ano2000)
head(x,3)

##          x       y variable value
## 1 -7.31924 38.5406  ano1991     0
## 2 -7.63557 40.5022  ano1991     0
## 3 -7.90273 40.3418  ano1991     0

x$variable <- substr(x$variable,4,7)
head(x,3)

##          x       y variable value
## 1 -7.31924 38.5406     1991     0
## 2 -7.63557 40.5022     1991     0
## 3 -7.90273 40.3418     1991     0
```


Creating a STFDF object (cont.)

- We are now ready to create the object (we have a full grid)

```
library(sp)
library(spacetime)
spatialCoords <- cbind(long=df$x,lat=df$y)
coordRefSys <- CRS("+proj=longlat +ellps=WGS84")
timeStamps <- as.POSIXct(unique(x$variable),
                         format="%Y",tz="GMT")
sp <- SpatialPoints(spatialCoords,coordRefSys)
## STFDF assumes that space "moves faster" than time in the data
std <- STFDF(sp,timeStamps,data=x[,"value",drop=FALSE])
```

Jožef Stefan Institute

Querying the spatio-temporal data set

- Data for a certain location

```
std[1,]

##            value timeIndex
## 1991-01-12     0         1
## 1992-01-12     0         2
## 1993-01-12     0         3
## 1994-01-12     0         4
## 1995-01-12     0         5
## 1996-01-12     0         6
## 1997-01-12     0         7
## 1998-01-12     0         8
## 1999-01-12     0         9
## 2000-01-12     0        10
```

Jožef Stefan Institute

Querying the spatio-temporal data set

- Data for all locations on a certain time stamp

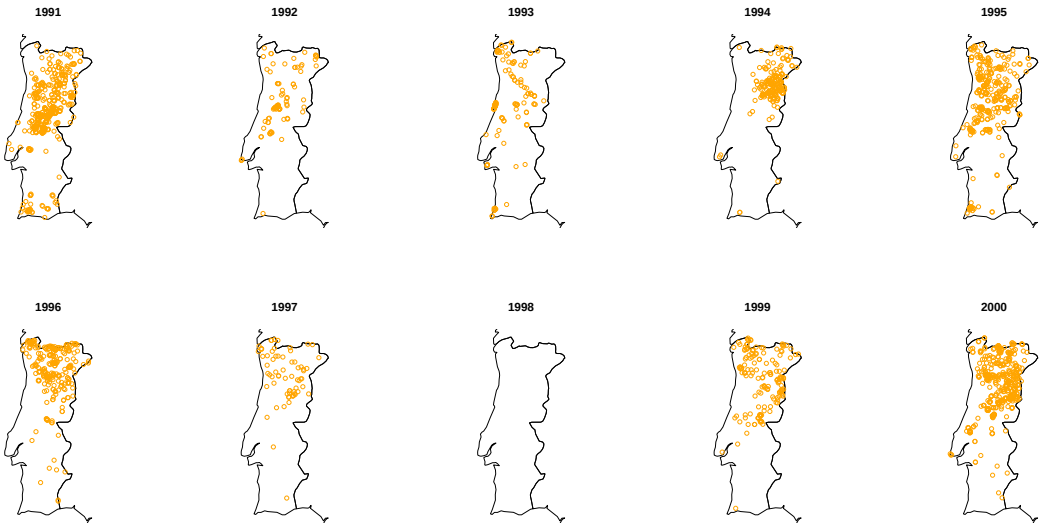
```
dataSince1995 <- std[,"1995/"]
data1991 <- std[,"1991"]
data1991[1:4,]

##           coordinates value
## 1 (-7.31924, 38.5406)     0
## 2 (-7.63557, 40.5022)     0
## 3 (-7.90273, 40.3418)     0
## 4 (-7.25657, 39.2572)     0
```

Jožef Stefan Institute

Plotting the spatio-temporal data set

```
library(rworldmap)
m <- getMap(resolution="low")
par(mfrow=c(2,5))
for(year in 1991:2000) {
    y <- std[,as.character(year)]
    bb <- bbox(y)
    plot(m,xlim=bb[1,],ylim=bb[2,])
    points(y[y$value ==1,],
           col="orange")
    title(year)
}
par(mfrow=c(1,1))
```


Plotting the spatio-temporal data set (cont.)



The Package **ggmap**

- Spatial visualization is a key step on spatio-temporal data analysis
- **ggmap** is a recent package that greatly enhances spatial visualization in R
- The package is able to combine spatial information from providers of maps (e.g. Google Maps) with the beautiful graphics of the **ggplot2** package
- On top of spatial visualization tools the package also includes also other nice utility functions

Jožef Stefan Institute

What is **ggplot2**

- An alternative graphics system for R based on the notion of *layered grammar of graphics*
- Every plot is formed by five components:
  - A data set and a set of aesthetic mappings (the way variables in the data are mapped into things we see in the graph)
  - One or more layers, each with a geometric object (*geom*) and a statistical transformation (*stat*)
  - A scale for each aesthetic mapping (frequently automatically generated)
  - A coordinate systems
  - A facet specification

Jožef Stefan Institute

ggmap and **ggplot2**

- As **ggmap** uses **ggplot2** graphics, the graphs also have the same components
- However, some of these are fixed to map components:
  - The *x* aesthetic is fixed to longitude
  - The *y* aesthetic is fixed to latitude
  - The coordinate system is fixed to the Mercator projection

Jožef Stefan Institute

The process of using **ggmap**

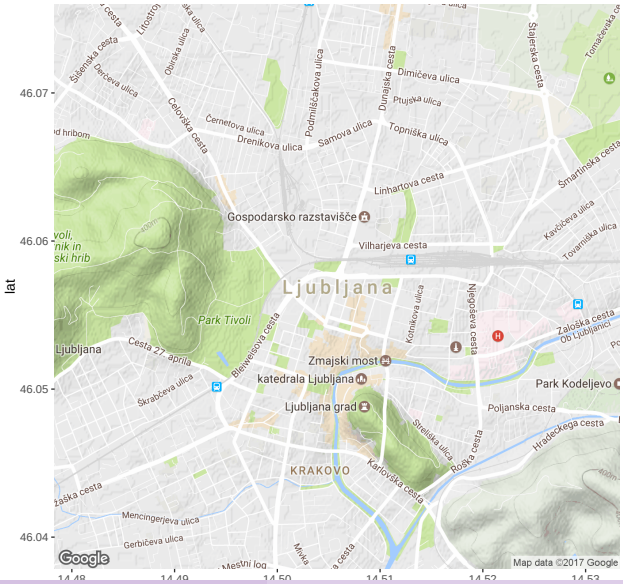
- Download a map image
- Plot it as basic layer using **ggplot2**
- Plot additional layers of data, statistics or models on top
- In **ggmap** this is done using:
  - `get_map()` to obtain the map image
  - `ggmap()` to make the actual plot

Jožef Stefan Institute

The function get_map()

- **ggmap** can use several map providers
  Google Maps, OpenStreet Maps, Stamen Maps and CloudMade Maps
- A simple example

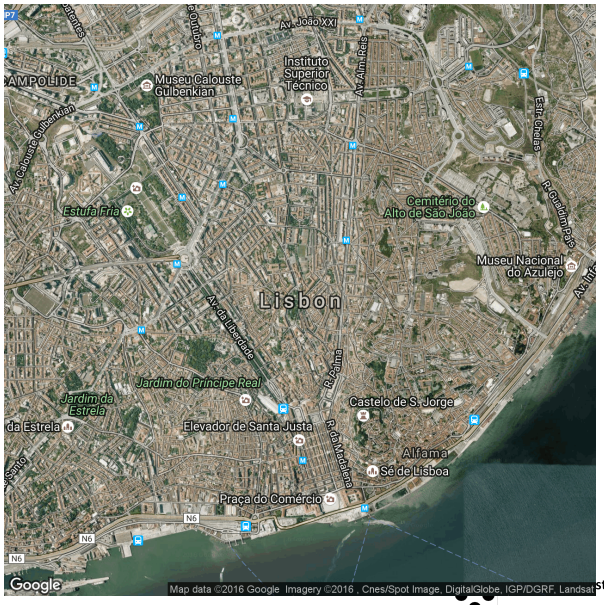
```
library(ggmap)
map <- get_map("Ljubljana",
               zoom=14)
ggmap(map)
```



The function get_map() - a few examples

```
library(ggmap)
map <- get_map("London",zoom=14,
               maptype="roadmap")
ggmap(map)
```

```
map <- get_map("Lisbon",zoom=14,
               maptype="hybrid")
ggmap(map,extent="device")
```



A Simple Illustration

Fires data of 500m^2 regions of Portugal

- Official data on fires in different regions of Portugal
- The data set we will use contains information on 25000 locations

```
df <- read.csv("firesnew_25000_500m.txt")
```

```
df[1:3,]
```

```
##   FID_ CID ano1991 ano1992 ano1993 ano1994 ano1995 ano1996 ano1997 ano1998
## 1   NA   1       0       0       0       0       0       0       0       0
## 2   NA   2       0       0       0       0       0       0       0       0
## 3   NA   3       0       0       0       0       0       0       0       0
##   ano1999 ano2000        x       y
## 1       0       0 -7.31924 38.5406
## 2       0       0 -7.63557 40.5022
## 3       0       0 -7.90273 40.3418
```

Jožef Stefan Institute

Putting the data in long format

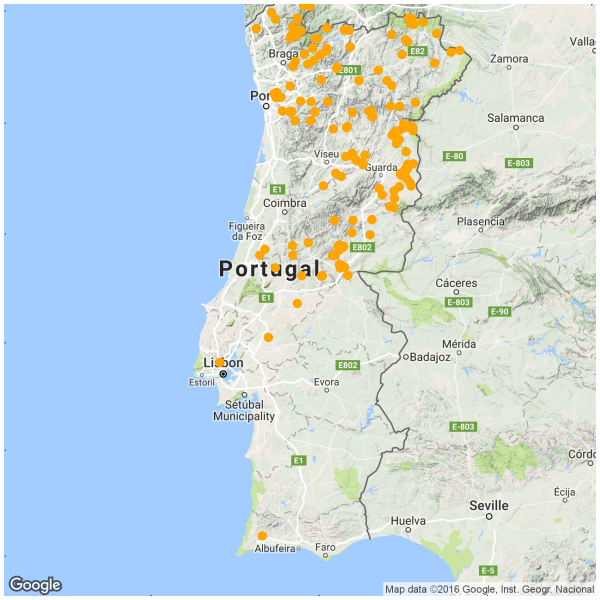
```
library(tidyr)
dat <- gather(df[,3:14],year, burnt, ano1991:ano2000)
dat$year <- substr(dat$year,4,7)
head(dat,4)
```

```
##           x       y year burnt
## 1 -7.31924 38.5406 1991     0
## 2 -7.63557 40.5022 1991     0
## 3 -7.90273 40.3418 1991     0
## 4 -7.25657 39.2572 1991     0
```

Jožef Stefan Institute

Plotting the Fires in 1999

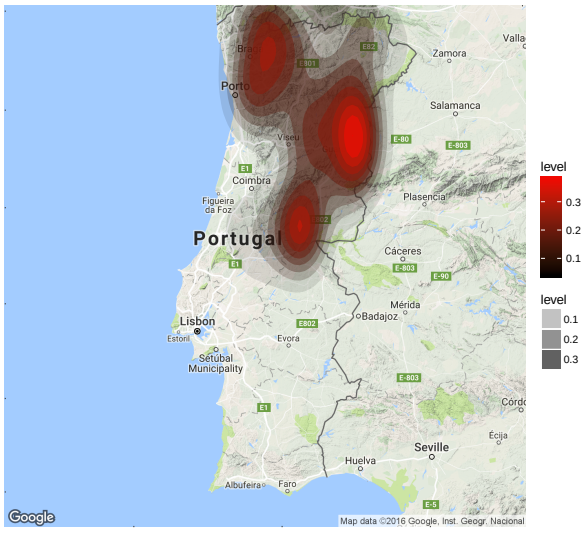
```
library(ggmap)
pt <- get_map("Portugal",zoom=7)
data2plot <- dat[dat$year==1999 & dat$burnt == 1,]
ggmap(pt,extent="device") +
    geom_point(data=data2plot,
               aes(x=x,y=y),
               color="orange",size=3)
```



Jožef Stefan Institute

Adding some spatial interpolation

```
ggmap(pt,extent="device",
      base_layer=ggplot(data2plot,aes(x=x,y=y))
      ) +
    stat_density2d(data=data2plot,
                   aes(x=x,y=y,
                       fill=..level..,
                       alpha=..level..),
                   bins=10,
                   geom="polygon") +
    scale_fill_gradient(low="black",high="red")
```



Jožef Stefan Institute

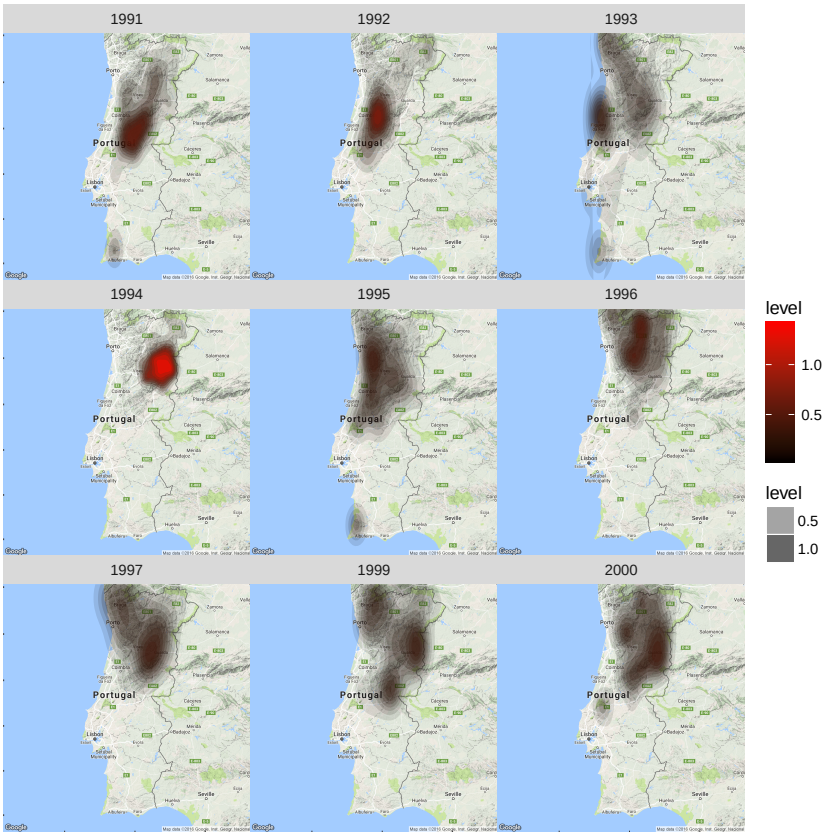
Spatio-temporal Visualization

- We need to add faceting along the year an everything else stays the same

```
data2plot <- dat[dat$burnt == 1,]
ggmap(pt,extent="device",
      base_layer=ggplot(data2plot,aes(x=x,y=y))
      ) +
    stat_density2d(data=data2plot,
                   aes(x=x,y=y,
                       fill=..level..,
                       alpha=..level..),
                   bins=10,
                   geom="polygon") +
    scale_fill_gradient(low="black",high="red") +
    facet_wrap( ~  year)
```

Jožef Stefan Institute

Spatio-temporal Visualization (cont.)



Jožef Stefan Institute

Getting geographic information of a location

- Taking advantage of the Google Maps API we can obtain information on a location

```
geocode("Ljubljana")
```

```
##        lon      lat
## 1 14.50575 46.05695
```

```
geocode("Ljubljana",output="more")
```

```
##        lon      lat     type     loctype             address   north
## 1 14.50575 46.05695 locality approximate ljubljana, slovenia 46.1422
##      south    east    west  locality administrative_area_level_1  country
## 1 45.99007 14.6447 14.4195 Ljubljana                   Ljubljana Slovenia
```

Note: Please note that the Google Maps API has several request limitations. Namely, it has an unspecified short-term rate limit as well as a 24-hour limit of 2500 requests.

Jožef Stefan Institute

Getting physical addresses from a long/lat pair

- Checking the address of one of the fire locations

```
revgeocode(as.numeric(df[1000,c("x","y")]))
```

```
## [1] "Unnamed Road, 7370, Portugal"
```

Jožef Stefan Institute

Calculating distances between locations

- The distance from New York to three other cities
- Note that the default mode is driving
- Again several request limitations are imposed by Google

```
from <- rep("New York",3)
to <- c("Los Angeles","San Francisco","Toronto")
mapdist(from,to)
```

```
##       from            to       m       km     miles seconds  minutes
## 1 New York   Los Angeles 4493003 4493.003 2791.9521  144653 2410.883
## 2 New York San Francisco 4674274 4674.274 2904.5939  152102 2535.033
## 3 New York       Toronto  790159  790.159  491.0048   27837  463.950
##      hours
## 1 40.18139
## 2 42.25056
## 3  7.73250
```

```
mapdist("Chinatown","Times Square",mode="walking")
```

```
##        from           to    m    km    miles seconds minutes    hours
## 1 Chinatown Times Square 5206 5.206 3.235008    3903   65.05 1.084167
```

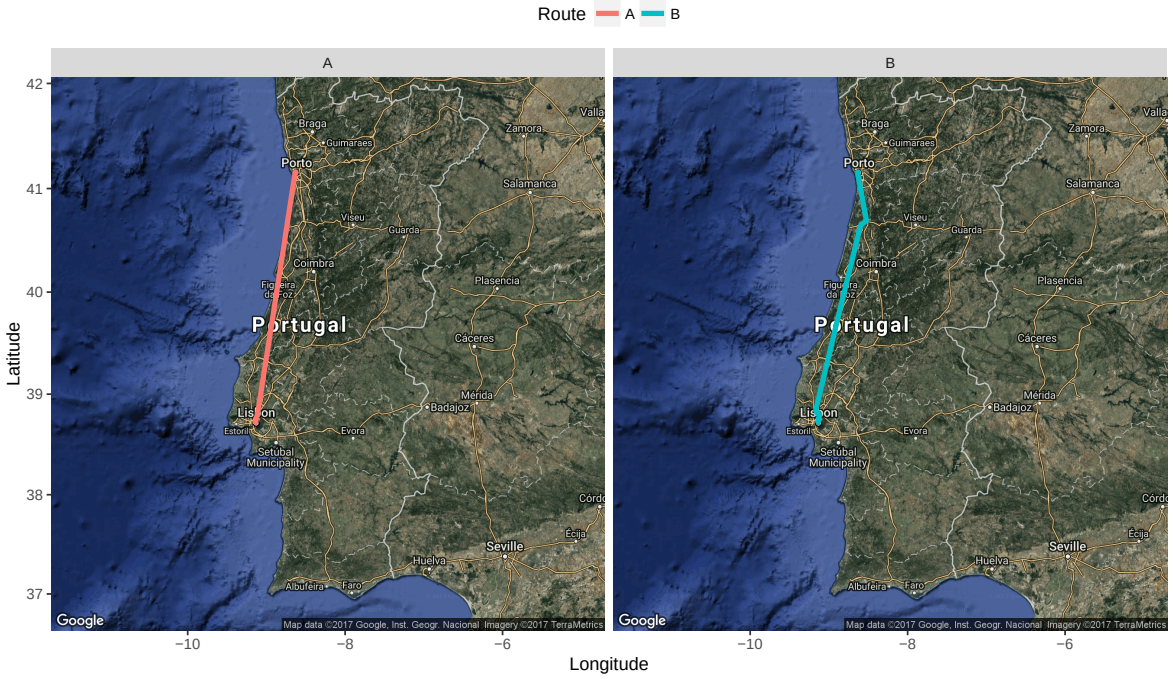

Routes between locations

- Note that the default mode is driving
- Again several request limitations are imposed by Google

```
rtDat <- route("Porto","Lisbon",mode="driving",structure="route",alternatives=TRUE)
mp <- get_map("Portugal",zoom=7,maptype="hybrid")
ggmap(mp,
      base_layer=ggplot(rtDat,aes(x=lon,y=lat,colour=route))) +
      geom_path(size=1.5,lineend="round") +
      facet_wrap(~ route) +
      labs(x = "Longitude", y = "Latitude", colour = "Route") +
      theme(legend.position = "top")
```

Jožef Stefan Institute

Routes between locations (cont.)



Jožef Stefan Institute

Interactive spatial visualization using package `leaflet`

- Leaflet is a very popular open-source JavaScript library for interactive maps
- The R package **leaflet** allows you to create this type of graphs in R
- After installing the package we can try it using the forest fires data

Jožef Stefan Institute

Interactive spatial visualization of the forest fires

```
df <- read.csv("firesnew_25000_500m.txt")
```

- The following shows the fires in 1999 in an interactive map

```
library(sp)
spatialCoords <- cbind(long=df$x, lat=df$y)
coordRefSys <- CRS("+proj=longlat +ellps=WGS84")
fires1999 <- SpatialPointsDataFrame(spatialCoords,
                                    df[, "ano1999", drop=FALSE],
                                    proj4string=coordRefSys)
library(leaflet)
leaflet() %>%
    addTiles() %>%
    addCircleMarkers(data=fires1999[fires1999$ano1999==1,])
```

Jožef Stefan Institute

Improving a bit the visualization

- Getting the addresses of the places where there were fires

```
library(ggmap)
placesWithFires <- which(fires1999$ano1999 == 1)
coord <- coordinates(fires1999)[placesWithFires,]
## Note that the Google API imposes limits on the following...
adds <- apply(coord,1,function(cs) revgeocode(cs))
## Now the interactive map (try clicking on a dot)
leaflet() %>%
    addTiles() %>%
    addCircleMarkers(data=fires1999[fires1999$ano1999==1,],
                     popup=adds,
                     clusterOptions = markerClusterOptions())
```

Jožef Stefan Institute

Learning more about leaflet

You may learn more at `https://rstudio.github.io/leaflet/`

Jožef Stefan Institute

Hands On Spatio-Temporal Data with ggmap

The file `irishWind.Rdata` contains two data frames with information on wind data values collected in several meteorological stations in Ireland along several years. The data frame **wind** contains the wind values for the different stations (in wide format), while the **wind.loc** data frame contains information on the stations. Using this data set answer the following questions:

1. Obtain the geographic coordinates of the stations
2. Reproduce the graph to the right

Jožef Stefan Institute

Hands On Spatio-Temporal Data with ggmap (cont.)

3. Using the functionalities provided by packages **tidyr** and **dplyr** obtain a data frame with the average yearly wind speed for each station.
4. Produce a spatio-temporal showing theses yearly averages on the stations.

Jožef Stefan Institute