Eksploracja tekstu i wyszukiwanie informacji w mediach społecznościowych

LABORATORIUM 10

- Twitter
- Google Maps
- Tramwaje w Warszawie
- Zanieczyszczenie powietrza

Twitter

Jak wszyscy wiedzą, z mediów społecznościowych można się o każdym dowiedzieć wszystkiego. Powszechny dostęp do wielu z tych serwisów został w ciągu ostatniego roku mocno ograniczony (np. Facebook) albo też obwarowany zdobyciem serii kodów dostępu etc. Na potrzeby dzisiejszych zajęć zajmiemy się Twitterem; dostęp do niego jest możliwy dzięki bibliotece **rtweet**, niestety, prawdopodobnie nie uda się jej uruchomić na kompuetrach w Pracowni... Twitter wymaga aż czterech różnych kodów, aby można było korzystać z API - te poniżej zostały stworzone specjalnie na potrzeby zajęć TEXT i po około tygodniu zostaną zresetowane.

```
# PRZYKŁAD 10.1
library(rtweet)

create_token(
  app = "TEXTclass",
  consumer_key = "mjBZhJWhlcYYkjTNQX1zC9z1t",
  consumer_secret = "J8KjngzwuimbpanNzGuWJOhx6flNID5D9wyhguCpxI5HMzjWTt",
  access_token = "414634084-T9H7IJR7nX0k2wYuJE8SWydgyaPduV4L0CweL3M1",
  access_secret = "EKQBhBnAf6Ew9Q9adeY1bn7YJEykKdsYpUbTUuyknp4ZW")
```

```
## <Token>
## <oauth_endpoint>
##  request:   https://api.twitter.com/oauth/request_token
##  authorize: https://api.twitter.com/oauth/authenticate
##  access:    https://api.twitter.com/oauth/access_token
## <oauth_app> TEXTclass
##   key:    mjBZhJWhlcYYkjTNQX1zC9z1t
##   secret: <hidden>
## <credentials> oauth_token, oauth_token_secret
## ---
```

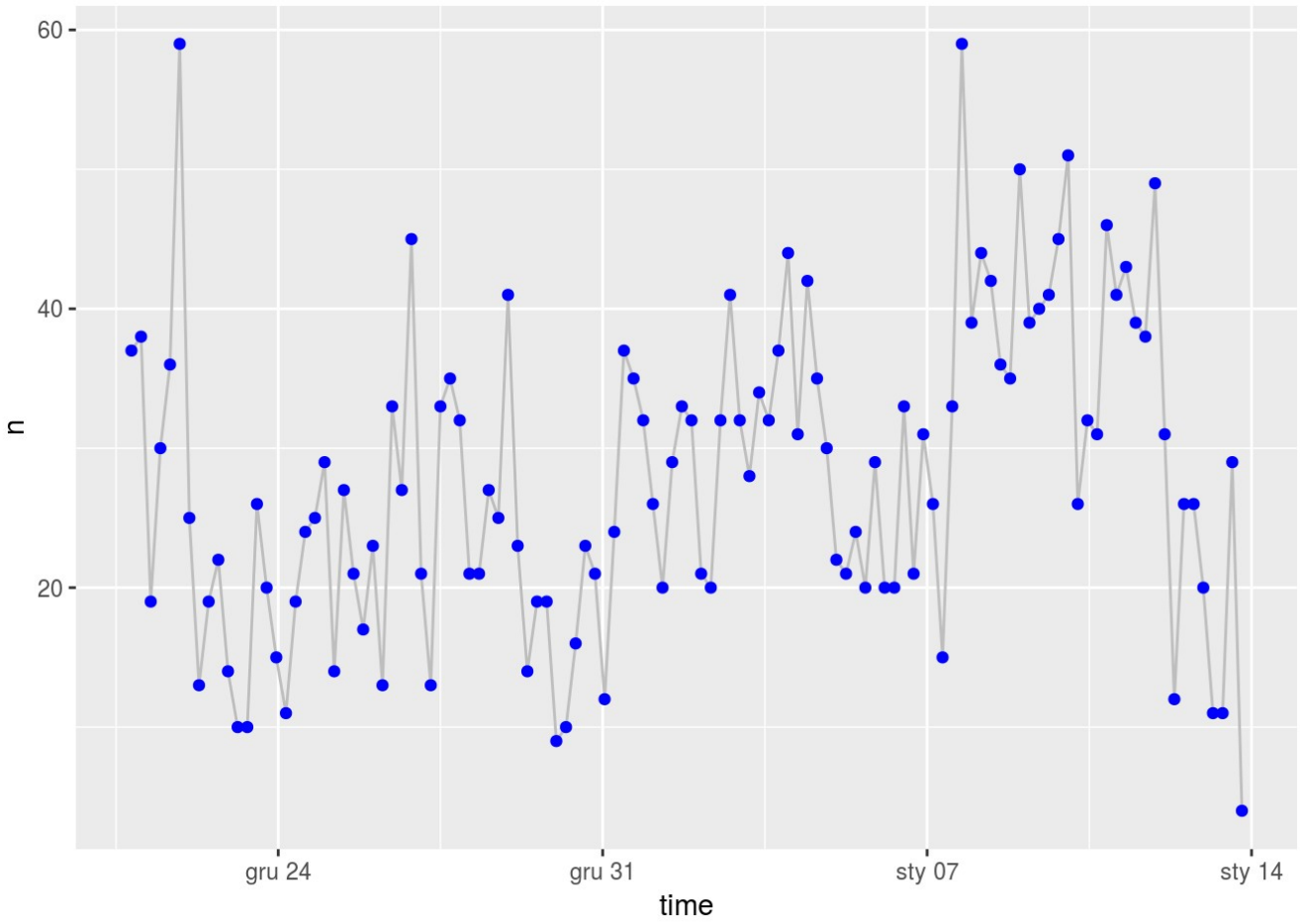
Po tym uwierzytelnaniu można już popbrać dane - poniżej zbierzemy 10000 tweetów w konta CNN

```
# PRZYKŁAD 10.2

tw <- get_timeline("cnn", n = 10000)
tw
```

```
## # A tibble: 3,234 x 88
##    user_id status_id created_at          screen_name text  source
##    <chr>   <chr>     <dttm>              <chr>       <chr> <chr>
##  1 759251  10845368. 2019-01-13 19:45:05 CNN         "GOP. Socia.
##  2 759251  10845330. 2019-01-13 19:30:05 CNN         "DNA. Socia.
##  3 759251  10845292. 2019-01-13 19:15:07 CNN         Peop. Socia.
##  4 759251  10845255. 2019-01-13 19:00:07 CNN         Amer. Socia.
##  5 759251  10845217. 2019-01-13 18:45:09 CNN         Puer. Socia.
##  6 759251  10845179. 2019-01-13 18:30:05 CNN         "Rep. Socia.
##  7 759251  10845141. 2019-01-13 18:15:07 CNN         Pres. Socia.
##  8 759251  10845104. 2019-01-13 18:00:06 CNN         Walt. Socia.
##  9 759251  10845066. 2019-01-13 17:45:05 CNN         Miss. Socia.
## 10 759251  10845034. 2019-01-13 17:32:23 CNN         A ma. Socia.
## # . with 3,224 more rows, and 82 more variables: display_text_width <dbl>,
## #   reply_to_status_id <chr>, reply_to_user_id <chr>,
## #   reply_to_screen_name <chr>, is_quote <lgl>, is_retweet <lgl>,
## #   favorite_count <int>, retweet_count <int>, hashtags <list>,
## #   symbols <list>, urls_url <list>, urls_t.co <list>,
## #   urls_expanded_url <list>, media_url <list>, media_t.co <list>,
## #   media_expanded_url <list>, media_type <list>, ext_media_url <list>,
## #   ext_media_t.co <list>, ext_media_expanded_url <list>,
## #   ext_media_type <chr>, mentions_user_id <list>,
## #   mentions_screen_name <list>, lang <chr>, quoted_status_id <chr>,
## #   quoted_text <chr>, quoted_created_at <dttm>, quoted_source <chr>,
## #   quoted_favorite_count <int>, quoted_retweet_count <int>,
## #   quoted_user_id <chr>, quoted_screen_name <chr>, quoted_name <chr>,
## #   quoted_followers_count <int>, quoted_friends_count <int>,
## #   quoted_statuses_count <int>, quoted_location <chr>,
## #   quoted_description <chr>, quoted_verified <lgl>,
## #   retweet_status_id <chr>, retweet_text <chr>,
## #   retweet_created_at <dttm>, retweet_source <chr>,
## #   retweet_favorite_count <int>, retweet_retweet_count <int>,
## #   retweet_user_id <chr>, retweet_screen_name <chr>, retweet_name <chr>,
## #   retweet_followers_count <int>, retweet_friends_count <int>,
## #   retweet_statuses_count <int>, retweet_location <chr>,
## #   retweet_description <chr>, retweet_verified <lgl>, place_url <chr>,
## #   place_name <chr>, place_full_name <chr>, place_type <chr>,
## #   country <chr>, country_code <chr>, geo_coords <list>,
## #   coords_coords <list>, bbox_coords <list>, status_url <chr>,
## #   name <chr>, location <chr>, description <chr>, url <chr>,
## #   protected <lgl>, followers_count <int>, friends_count <int>,
## #   listed_count <int>, statuses_count <int>, favourites_count <int>,
## #   account_created_at <dttm>, verified <lgl>, profile_url <chr>,
## #   profile_expanded_url <chr>, account_lang <chr>,
## #   profile_banner_url <chr>, profile_background_url <chr>,
## #   profile_image_url <chr>
```

Pakiet **rtweet** ma wbudowaną funkcję **ts_plot()**, która korzysta z pakietu **ggplot2** i do której, podobnie jak do ggplota, można dodawać kolejne warstwy. Argumentem funkcji ts_plot jest rozdzielczość z jaką chcemy wykreślić aktywnośc (czyli liczbę tweetów) w danych.

```
# PRZYKŁAD 10.3

ts_plot(tw, "5 hour", color = "gray") + geom_point(color = "blue")
```



Biblioteka oferuje całą masę różnych opcji (polecam prezentację (https://mkearney.github.io/nicar_tworkshop/#1) oraz dokumentację (https://rtweet.info/)). My zrobimy tylko jeden prosty wykres, ale wykorzystamy do niego wcześniej zdobyte metody - poliyczymy walencję w kolejnych tweetach, uzywając klasyfikatora słownikowego, opartego na zbiorze Warriner et al. z Laboratorium 6 (http://www.if.pw.edu.pl/~julas/TEXT/lab/text06.html).

```
# PRZYKŁAD 10.4

library(tidytext)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
emo <- as_tibble(read.csv("http://www.fizyka.pw.edu.pl/~julas/TEXT/lab/Ratings_Warrine
r_et_al.csv", stringsAsFactors = F))

emo <- emo %>%
  select(word = Word, valence = V.Mean.Sum, arousal = A.Mean.Sum)

tw.data <- tw %>% transmute(id = row_number(), text = text) %>% unnest_tokens(word, te
xt)

tw.sent <- tw.data %>%
  inner_join(emo) %>%
  group_by(id) %>%
  summarise(valence = mean(valence))
```

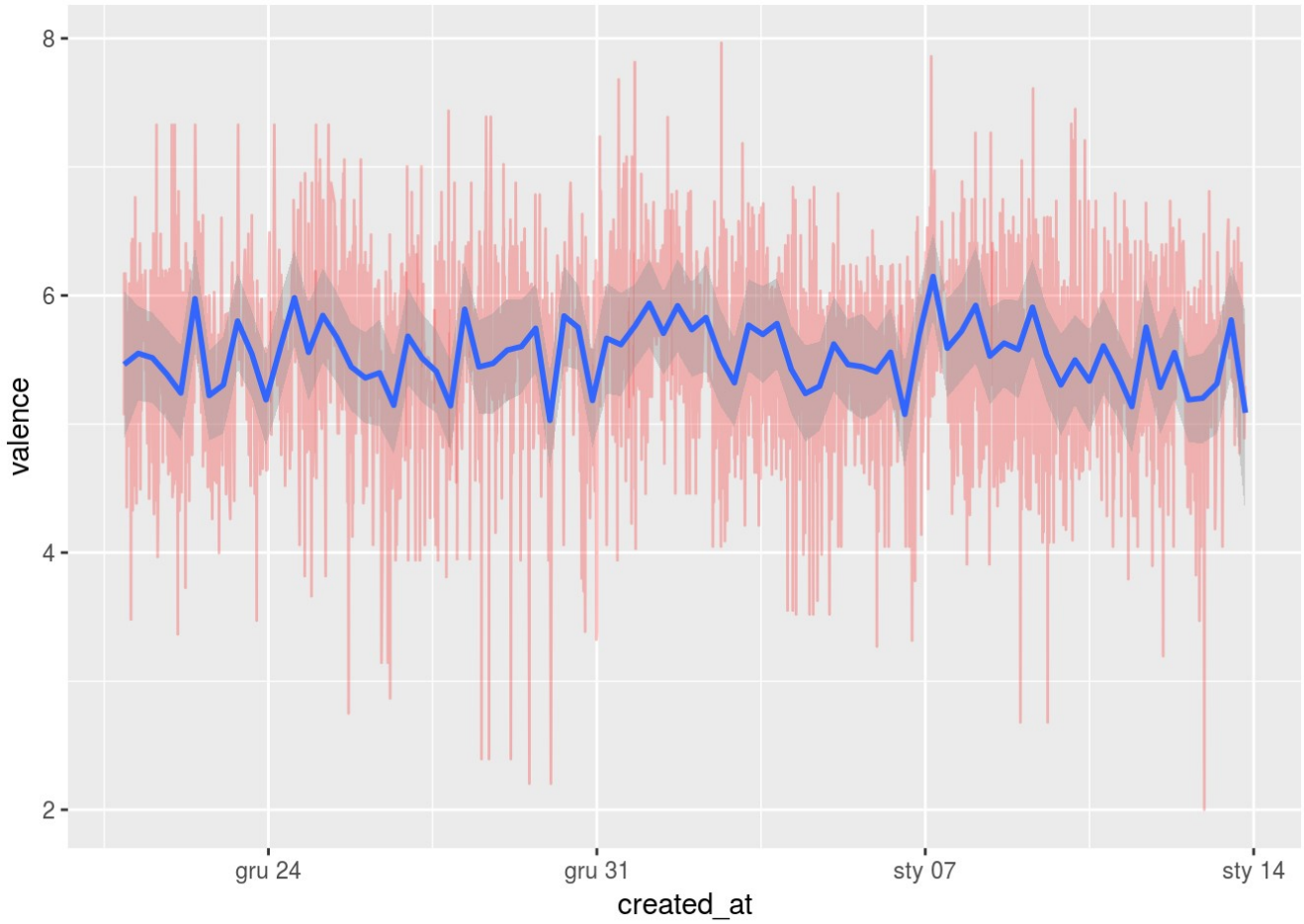
```
## Joining, by = "word"
```

```
tw$valence[tw.sent$id] <- tw.sent$valence
```

```
## Warning: Unknown or uninitialised column: 'valence'.
```

```
ggplot(tw, aes(x = created_at, y = valence)) +
  geom_line(alpha=0.25, color = "red") +
  geom_smooth(method = "loess", span = 0.01)
```

```
## Warning: Removed 15 rows containing non-finite values (stat_smooth).
```



Google Maps

Google jest prawdziwym potentatem i w zasadzie monopolistą jeśli chodzi o szereg usług -- możemy albo z nich nie korzystać albo też zgadzać się na różne praktyki. Nie da się jednak ukryć, że poniższe usługi, do których dedykowane są konkretne API są dość przydatne:

- Distance Matrix,
- Maps Static,
- Directions,
- Geocoding,
- Maps

Nas interesuje w tym momencie szczególnie ta ostatnia pozycja. R oferuje kilka pakietów do obsługi Google Maps - my skorzystamy z **ggmap**. Jak sama nazwa wskazuje, pakiet korzysta z **ggplot2** i jest z nim kompatybilny pod względem idei użycia warstw. Na początek wygenerujemy prostą mapę scentrowaną na środku Warszawy. Do obsługi map potrzebny jest nam klucz API - poniższy jest wygenerowany na użytek zajęć i zostanie zresetowany po dwóch tygodniach. Największym problemem jet to, że pakiet **ggmap** trzeba zainstalować z odrębnego źródła, a nie z ogólnego repozytorium i dodatkowo zajmuje to sporo czasu, bo niektóre pakiety są instalowane od nowa ze źródła.

```
# PRZYKŁAD 10.5

if(!requireNamespace("devtools")) install.packages("devtools")
devtools::install_github("dkahle/ggmap", ref = "tidyup")
```

```
# PRZYKŁAD 10.6

library(ggmap)
```

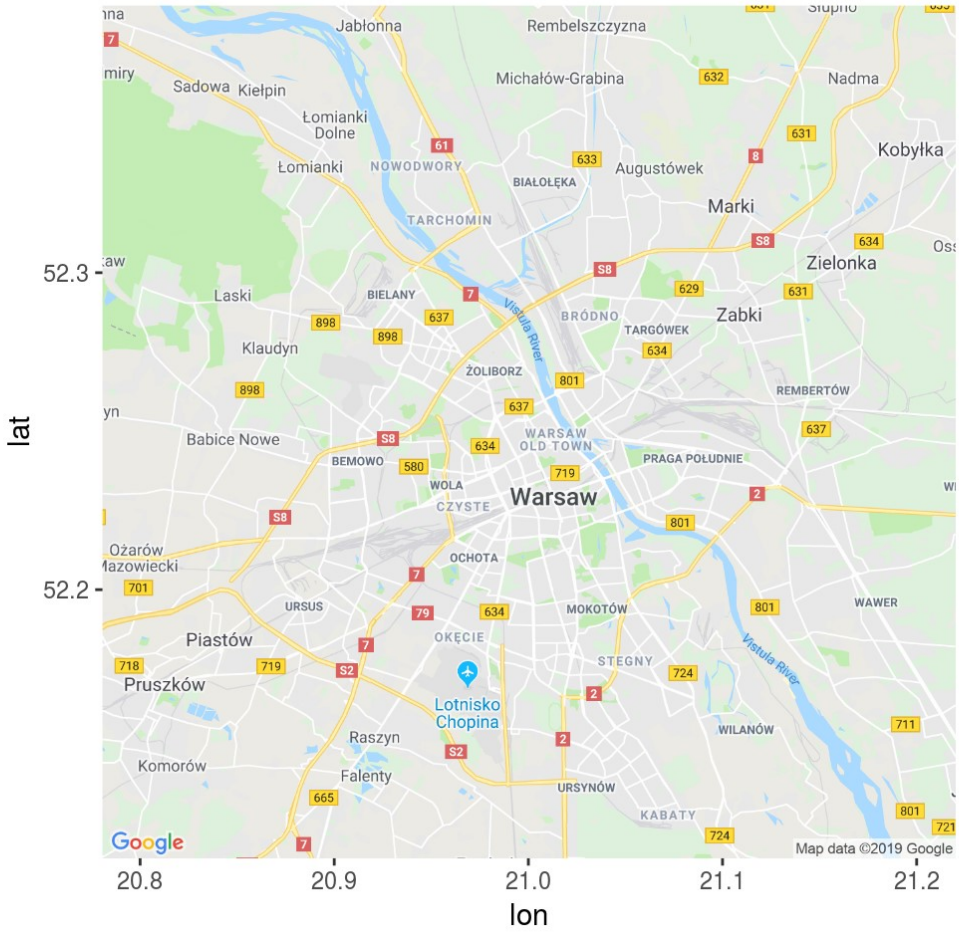
```
## Google Maps API Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it: see citation("ggmap") for details.
```

```
register_google("AIzaSyCVaYyyqIPta_LCss-P-P73k2fqAZPp7Ps")

map <- get_googlemap(center = c(lon = 21.00, lat = 52.25), zoom = 11, maptype = "road"
)
```

```
## Source : https://maps.googleapis.com/maps/api/staticmap?center=52.25,21&zoom=11&siz
e=640x640&scale=2&maptype=roadmap&key=xxx-P-P73k2fqAZPp7Ps
```

```
ggmap(map)
```



Na mapę naniesiemy teraz pozycje przystanków sieci transportu miejskiego - zbiór wejściowy zosta pobrany ze serwera ZTM (ftp://rozklady.ztm.waw.pl/).

```
# PRZYKŁAD 10.7

stops <- readLines("http://www.if.pw.edu.pl/~julas/TEXT/lab/stops.txt")
head(stops)
```

```
## [1] "   1001   KIJOWSKA,                        --  WARSZAWA"
## [2] "      *PR  9"
## [3] "         100101   2      Ul./Pl.: TARGOWA,                          Kier.: AL.
ZIELENIECKA,                  Y= 52.248670     X= 21.044260  "
## [4] "            L  6  - stały:              125   135   138   166   509   517  "
## [5] "            L  1  - na żądanie:         N21  "
## [6] "         100102   2      Ul./Pl.: TARGOWA,                          Kier.: ZĄB
KOWSKA,                       Y= 52.249020     X= 21.044540  "
```

Na oko zbiór ma dość nieuporządkowaną postać, na szczęście okazuje się, że poszczególne dane, które chcielibyśmy na nasze potrzeby otrzymać (numer przystanku oraz współrzędne) łatwo poddają się ekstrakcji za pomocą prostych wyrażeń regularnych

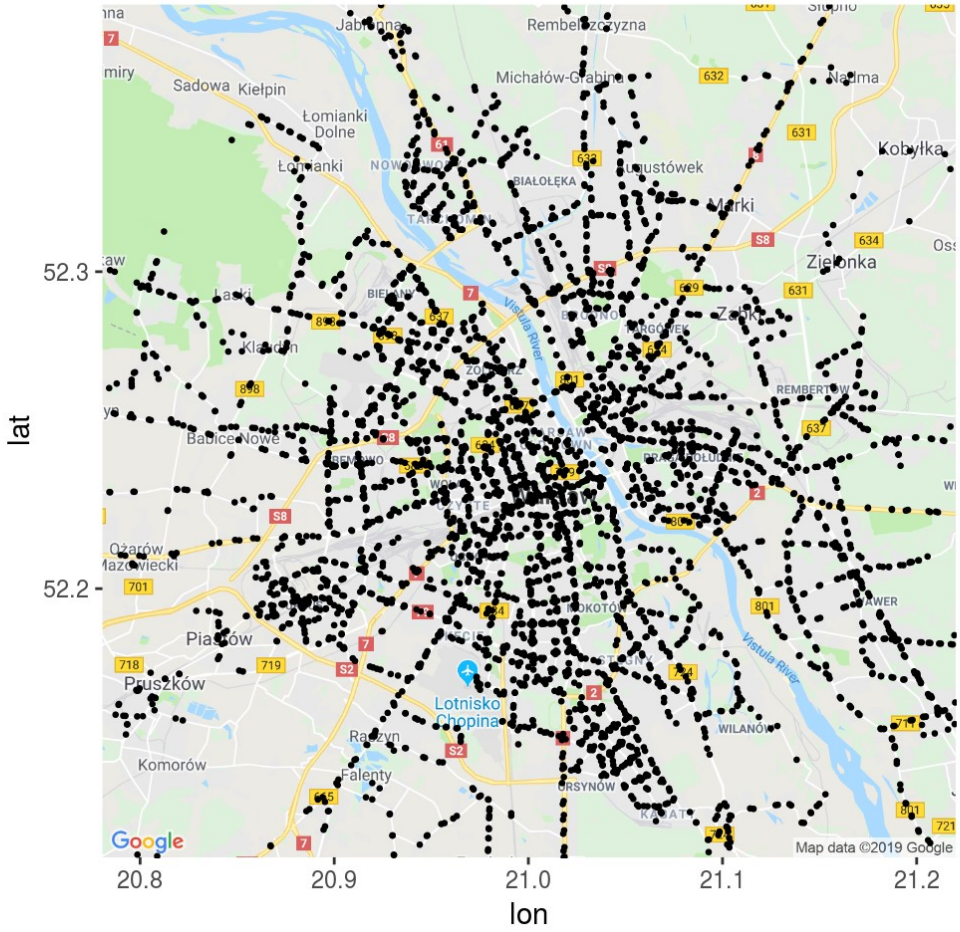
```
# PRZYKŁAD 10.8

get.coor <- function(line) {

p <- regexpr("[0-9]{6}", line)
y <- regexpr("Y= 52.[0-9]{5,6}", line)
x <- regexpr("X= 2[0-1].[0-9]{5,6}", line)

p <- substr(line, p[1], p[1] + attr(p, "match.length") - 1)
y <- substr(line, y[1] + 3, y[1] + attr(y, "match.length") - 1)
x <- substr(line, x[1] + 3, x[1] + attr(x, "match.length") - 1)

data.frame(stop_id = as.numeric(p), long = as.numeric(y), lat = as.numeric(x))

}

stops <- get.coor(stops[grep("Y= ", stops)])
head(stops)
```

```
##   stop_id     long      lat
## 1  100101 52.24867 21.04426
## 2  100102 52.24902 21.04454
## 3  100103 52.24900 21.04398
## 4  100104 52.24990 21.04173
## 5  100105 52.25035 21.04386
## 6  100106 52.25001 21.04371
```

Korzystając z powyższej ramki danych, możemy teraz nanieść poszczególne punkty na mapę, dodając je jako kolejną warstwę ggplota.

```
# PRZYKŁAD 10.9
ggmap(map) + geom_point(data = stops, aes(x = lat, y = long), size=0.5, shape = 19)
```

```
## Warning: Removed 1286 rows containing missing values (geom_point).
```



Na bazie takich danych można się pokusić o np. naniesienie warstwy z interpolowaną gestością przystanków (warstwa **stat_density2d**). Przy okazji dokonamy również drobnej zmiany podstawowej mapy, ustawiając język na polski oraz kolor na czarno-biały.

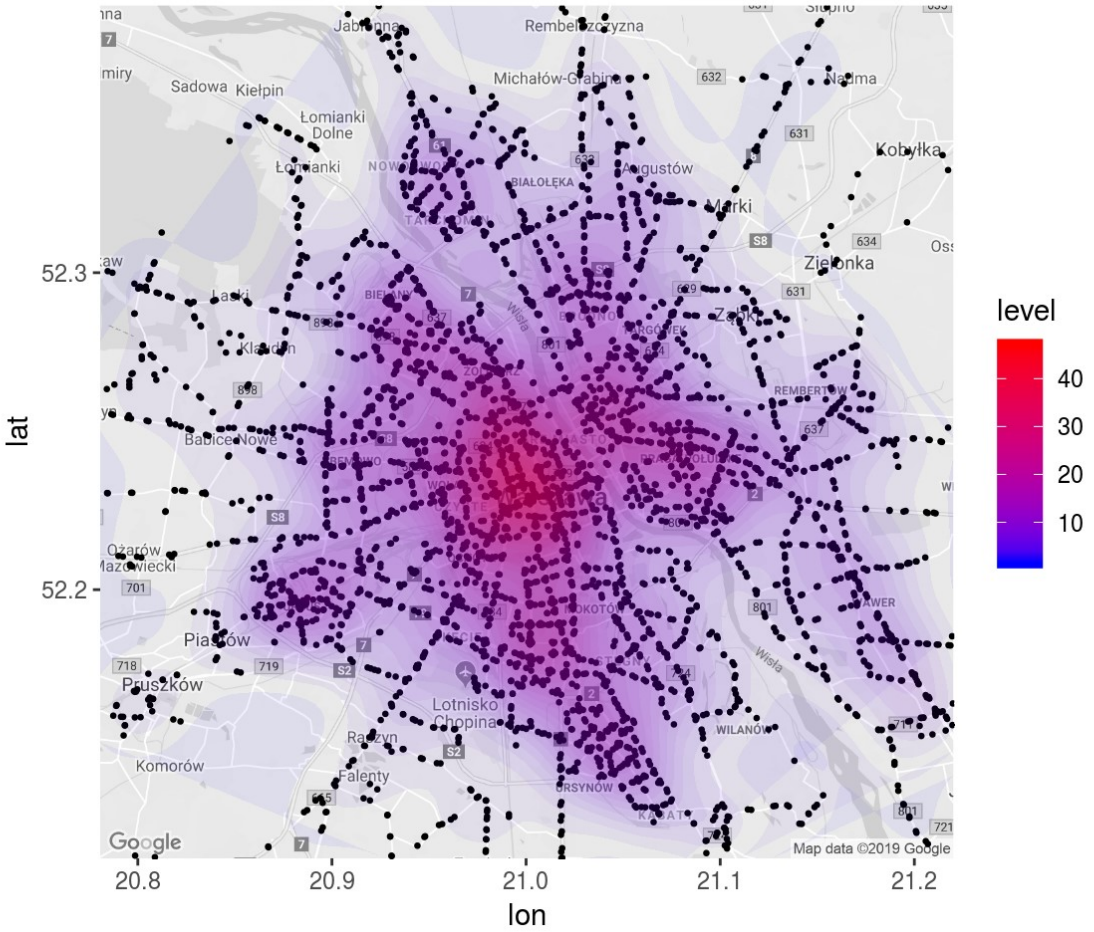
```
# PRZYKŁAD 10.10

map <- get_googlemap(center = c(lon = 21.00, lat = 52.25), zoom = 11, maptype = "road"
, color = "bw", language = "PL")
```

```
## Source : https://maps.googleapis.com/maps/api/staticmap?center=52.25,21&zoom=11&siz
e=640x640&scale=2&maptype=roadmap&language=PL&key=xxx-P-P73k2fqAZPp7Ps
```

```
ggmap(map) +
  geom_point(data = stops, aes(x = lat, y = long), size=0.5, shape = 19) +
  stat_density2d(data = stops, aes(x = lat, y = long, fill = stat(level)), geom = "pol
ygon", alpha = 0.05, bins = 30) + scale_fill_gradient(low = "blue", high = "red")
```

```
## Warning: Removed 1286 rows containing non-finite values (stat_density2d).
```

```
## Warning: Removed 1286 rows containing missing values (geom_point).
```



Tramwaje w Warszawie

Pozycje przystanków są na pewno interesujące, ale zwykle więcej uwagi poświęca się danym dynamicznym, zmieniającym się w czasie. Takie możliwości oferuje np. serwis Otwarte Dane (https://api.um.warszawa.pl/) pod egidą Urzędu Miasta Warszawy. Rejestrując tam konto otrzymujemy klucz API do różnych danych - nas interesuje serwis udostępniający w czasie rzeczywistym pozycje tramajów w Warszawie.

Do opracowania wyników zapytań potrzebujemy biblioteki **jsonlite**, która obsługuje format JSON zwracany przez API

```
# PRZYKŁAD 10.11
library(jsonlite)
```

```
##
## Attaching package: 'jsonlite'
```

```
## The following object is masked from 'package:rtweet':
##
##     flatten
```

```
um.waw.api <- "41f78414-bc00-4099-b607-42bb80a9d9d5"
um.waw.url <- "https://api.um.warszawa.pl/api/action/wsstore_get/?id=c7238cfe-8b1f-4c3
8-bb4a-de386db7e776&apikey="

url.api <- paste(um.waw.url, um.waw.api, sep = "")
trams <- fromJSON(url.api)

head(trams$result)
```

```
##    Status FirstLine      Lon           Lines                 Time      Lat
## 1 RUNNING       28  20.92525 28              2019-01-13T20:48:56 52.26143
## 2 RUNNING       6   20.94394 6               2019-01-13T20:48:57 52.29683
## 3 RUNNING       6   20.92996 6               2019-01-13T20:48:59 52.29200
## 4 RUNNING       27  20.97193 27              2019-01-13T20:48:59 52.27275
## 5 RUNNING       28  20.90473 28              2019-01-13T20:48:58 52.23906
## 6 RUNNING       35  20.93497 35              2019-01-13T20:48:57 52.27289
##   LowFloor Brigade
## 1    FALSE    3
## 2    FALSE    6
## 3    FALSE    8
## 4    FALSE    3
## 5    FALSE    6
## 6    FALSE    7
```

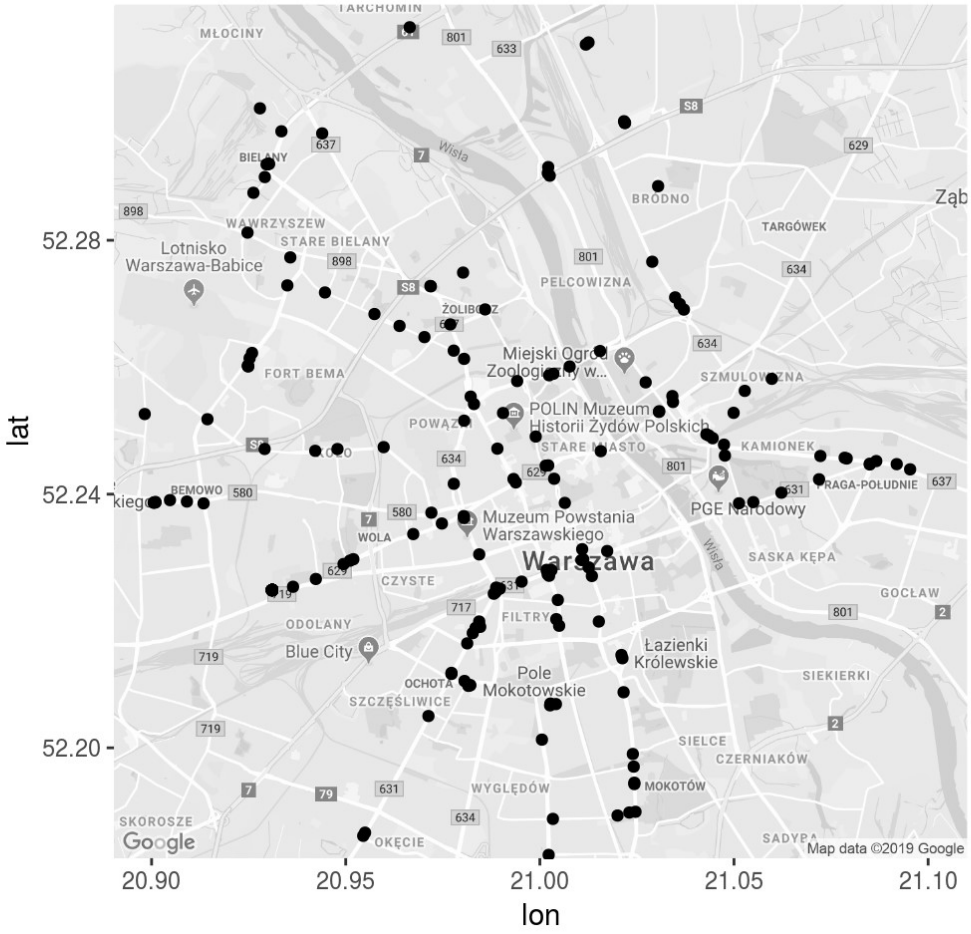
Teraz swobodnie wykorzystujemy naszą mapę Google, aby nanieść na nią pozycje tramwajów.

```
# PRZYKŁAD 10.12
map <- get_googlemap(center = c(lon = 21.00, lat = 52.25), zoom = 12, maptype = "road"
, color="bw", language = "PL")
```

```
## Source : https://maps.googleapis.com/maps/api/staticmap?center=52.25,21&zoom=12&siz
e=640x640&scale=2&maptype=roadmap&language=PL&key=xxx-P-P73k2fqAZPp7Ps
```

```
ggmap(map) +
  geom_point(data = trams$result, aes(x = Lon, y = Lat))
```

```
## Warning: Removed 18 rows containing missing values (geom_point).
```



Mapę można trochę "uatrakcyjnić", nakładając na punkty numer linii oraz dodatkowo kolorując według tych numerów.

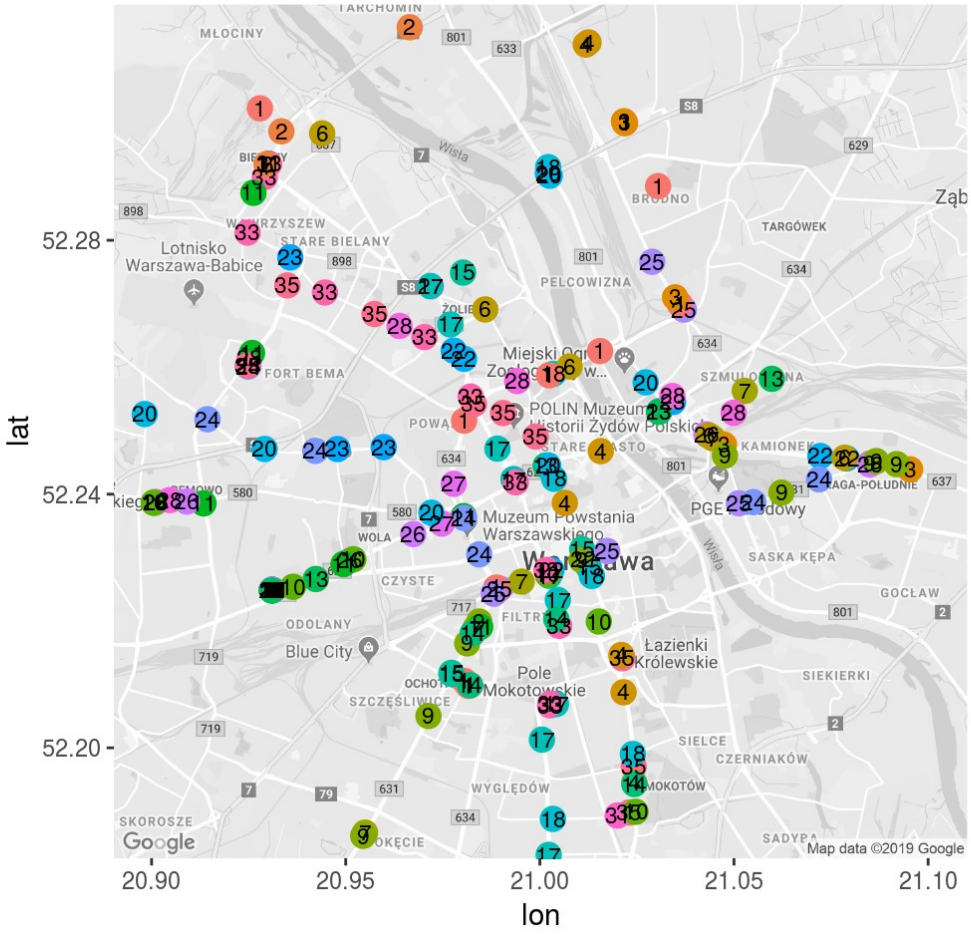
```
# PRZYKŁAD 10.13

trams$result$FirstLine <- as.numeric(gsub(" ", "", trams$result$FirstLine))
```

```
## Warning: NAs introduced by coercion
```

```
ggmap(map) +
  geom_point(data = trams$result, aes(x = Lon, y = Lat, color = as.factor(FirstLine)),
size = 4) +
  geom_text(data = trams$result, aes(x = Lon, y = Lat, label = FirstLine), size = 3) +
  scale_color_discrete(guide = FALSE)
```

```
## Warning: Removed 18 rows containing missing values (geom_point).
```

```
## Warning: Removed 19 rows containing missing values (geom_text).
```



Zanieczyszczenie powietrza

Jak wiadomo, zanieczyszczenie powietrza (czyli popularny smog) stanowi coraz większy problem w Polsce - jest to wypadkowa natężonego ruchu pojazdów oraz ogrzewania domów za pomocą kiespkiej jakości paliw, przy czym ten drugi czynnik jest raczej decydujący. W rezultacie takiej sytuacji, pojawiają się coraz to nowe przedsiębiorstwa, oferujące dostęp do metod moniotorowania poziomu zanieczyszczenia. Jedną z takich form jest Airly (https://airly.eu/pl/), która udostępnia mapę (https://airly.eu/map/pl/) oraz apki, umożliwiające ocene stanu powietrza w danej chwili, jak również obejrzenie 24-godzinnej historii takich pomiarów oraz prognoze na kolejną dobę. Airly dostarcza również API (https://developer.airly.eu/api): po zarejestrowaniu się, otrzymujemy klucz dający umożliwiający wykonanie maks. 50 zapytań na minutę oraz do 1000 na dobę.

Naszym pierwszym krokiem będzie zdobycie danych 45 czujników z obszaru Warszawy.

```
# PRZYKŁAD 10.14

airly.api = "0oRd9wGrqtnhN91H1UaAhPwKjRhGrmxy"
airly.url1 <- "https://airapi.airly.eu/v2/installations/nearest?lat=52.25&lng=21.00&ma
xDistanceKM=30&maxResults=45&apikey="

query <- paste(airly.url1, airly.api, sep = "")

czujniki <- fromJSON(query)
head(czujniki)
```

```
##     id location.latitude location.longitude address.country address.city
## 1 2191          52.25574           20.99337          Poland     Warszawa
## 2 1074          52.24233           20.99543          Poland     Warszawa
## 3 2350          52.25445           20.98454          Poland     Warszawa
## 4 3190          52.24577           21.01822          Poland     Warszawa
## 5 2525          52.23689           20.99439          Poland     Warszawa
## 6  729          52.26425           20.98856          Poland     Warszawa
##           address.street address.number address.displayAddress1
## 1              Inflancka             4b                Warszawa
## 2   aleja "Solidarności"            113                Warszawa
## 3    aleja Jana Pawła II             80                Warszawa
## 4                  Sowia              4                Warszawa
## 5             Krochmalna              1                Warszawa
## 6 aleja Wojska Polskiego             20                Warszawa
##   address.displayAddress2 elevation airly sponsor.id    sponsor.name
## 1            Inflancka 4A    101.07  TRUE         22           Aviva
## 2  aleja Solidarności 113    114.64  TRUE         49        eurobank
## 3        Jana Pawła II 80    108.31  TRUE         22           Aviva
## 4                   Sowia     88.35  TRUE         22           Aviva
## 5              Krochmalna    114.79  TRUE         10 WarszawaOddycha
## 6  aleja Wojska Polskiego     98.09  TRUE         10 WarszawaOddycha
##              sponsor.description
## 1         Airly Sensor's sponsor
## 2         Airly Sensor's sponsor
## 3         Airly Sensor's sponsor
## 4         Airly Sensor's sponsor
## 5 Airly Sensor is part of action
## 6 Airly Sensor is part of action
##                                                sponsor.logo
## 1 https://cdn.airly.eu/logo/Aviva_1538146740542_399306786.jpg
## 2                     https://cdn.airly.eu/logo/eurobank.jpg
## 3 https://cdn.airly.eu/logo/Aviva_1538146740542_399306786.jpg
## 4 https://cdn.airly.eu/logo/Aviva_1538146740542_399306786.jpg
## 5               https://cdn.airly.eu/logo/WarszawaOddycha.jpg
## 6               https://cdn.airly.eu/logo/WarszawaOddycha.jpg
##                   sponsor.link
## 1 https://wiemczymoddycham.pl/
## 2                         <NA>
## 3 https://wiemczymoddycham.pl/
## 4 https://wiemczymoddycham.pl/
## 5                         <NA>
## 6                         <NA>
```

Jak widać, każdy z czujników ma unikalny numer (**id**) oraz podaną lokalizajcę. Będziemy teraz chcieli odpytać każdy z czujników - naszym celem jest stan powietrza w danej chwili (reprezentowany przez tzn indeks ACQ), ale przy okazji otrzymamy też całą masę innych danych, które wykorzystamy później. W tym momencie stworzymy sobie ramkę danych, zawierającą wartości indeksu oraz pozycje czujników, a następnie korzystając z ggmap naniesiemy te dane na mapę.

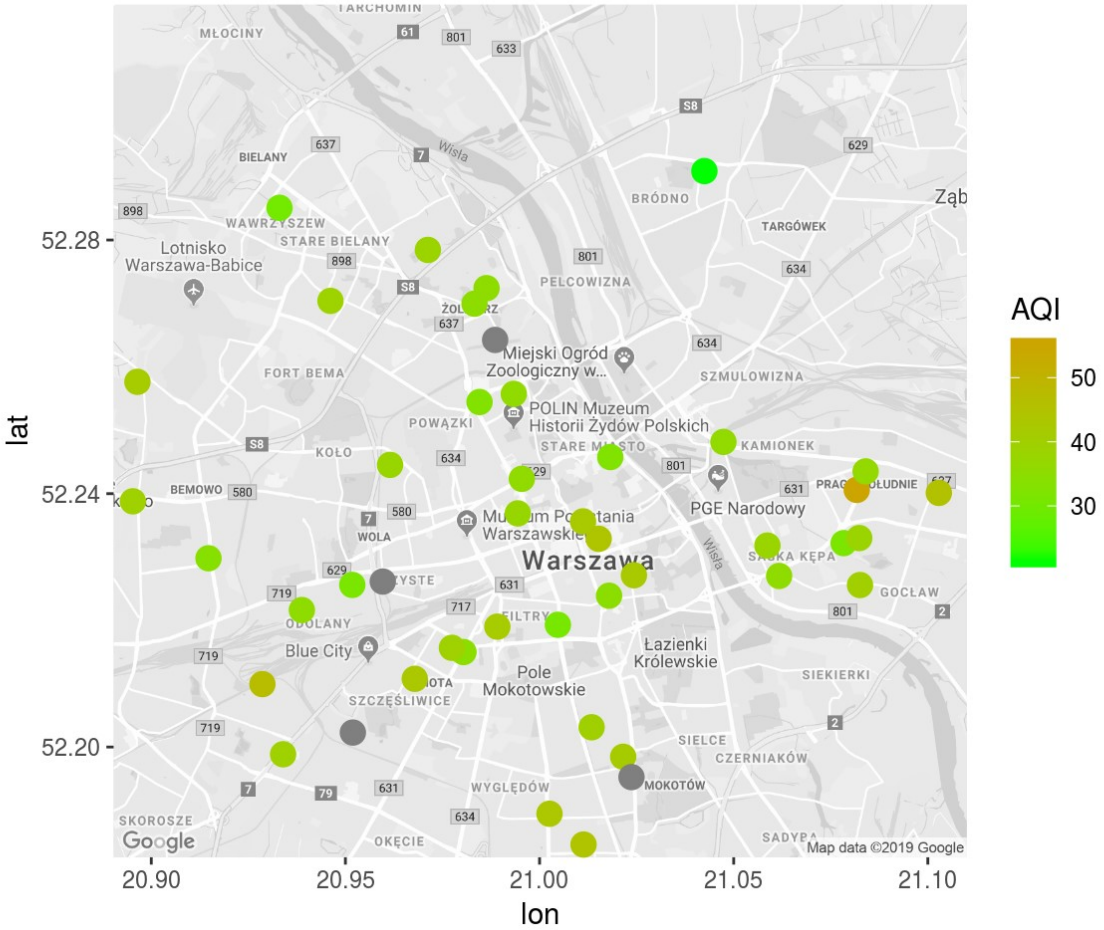
```
# PRZYKŁAD 10.15

get.air.mes <- function(id, api.key) {

  query <- sprintf("https://airapi.airly.eu/v2/measurements/installation?installationI
d=%d&apikey=%s", id, api.key)
  print(id)
  #Sys.sleep(1)
  fromJSON(query)

}

pomiary <- lapply(czujniki$id, get.air.mes, api.key = airly.api)
```

```
## [1] 2191
## [1] 1074
## [1] 2350
## [1] 3190
## [1] 2525
## [1] 729
## [1] 337
## [1] 658
## [1] 2724
## [1] 6615
## [1] 6532
## [1] 6864
## [1] 956
## [1] 6961
## [1] 26
## [1] 3410
## [1] 3507
## [1] 2325
## [1] 6959
## [1] 2679
## [1] 2330
## [1] 7448
## [1] 817
## [1] 528
## [1] 6630
## [1] 2720
## [1] 3328
## [1] 27
## [1] 2143
## [1] 812
## [1] 2205
## [1] 3511
## [1] 6963
## [1] 2067
## [1] 1071
## [1] 650
## [1] 1091
## [1] 1066
## [1] 2129
## [1] 6954
## [1] 2214
## [1] 3465
## [1] 732
## [1] 1016
## [1] 6996
```

```
indexes <- sapply(1:length(pomiary), function(i) pomiary[[i]]$current$indexes$value)

data <- cbind(czujniki$location, indexes)

head(data)
```

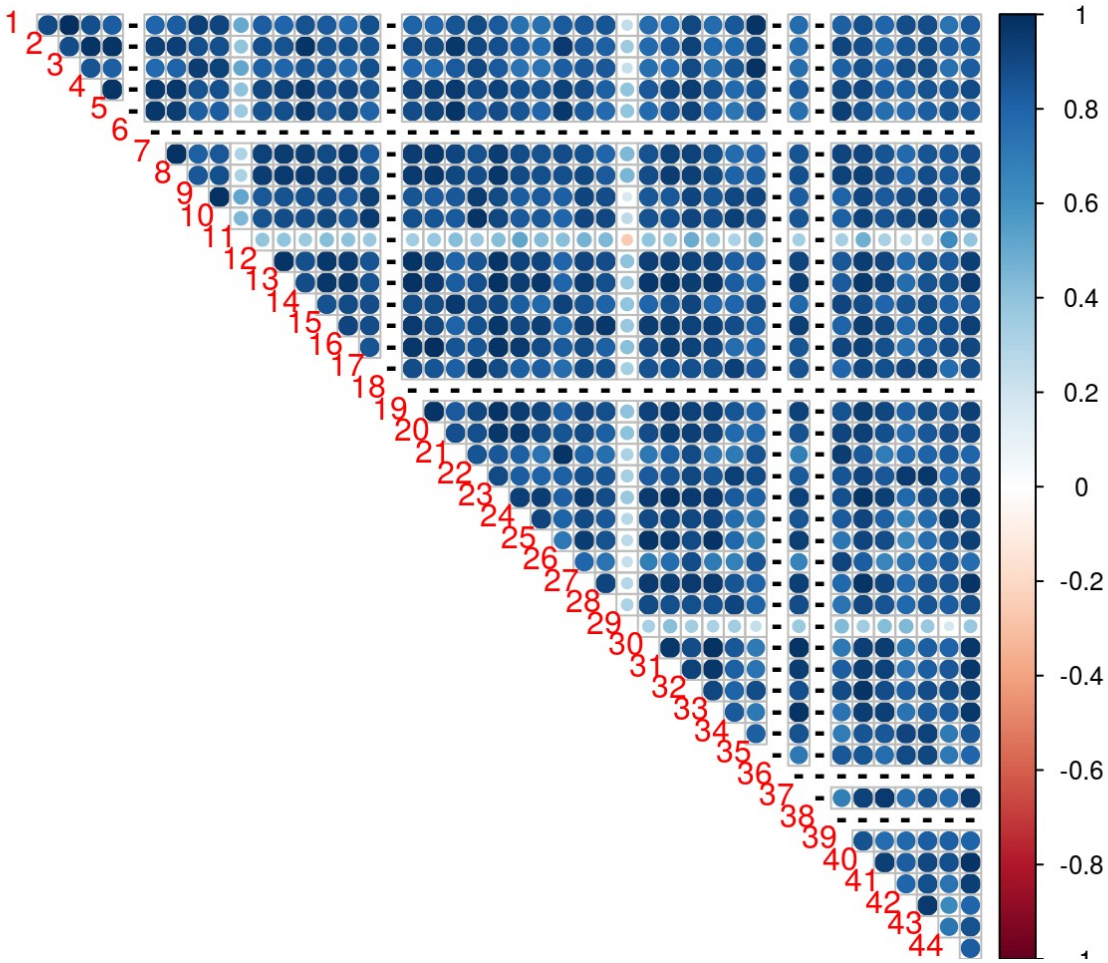
```
##   latitude longitude indexes
## 1 52.25574  20.99337   35.66
## 2 52.24233  20.99543   35.81
## 3 52.25445  20.98454   32.97
## 4 52.24577  21.01822   32.50
## 5 52.23689  20.99439   37.44
## 6 52.26425  20.98856      NA
```

```
ggmap(map) + geom_point(data = data, aes(x = longitude, y = latitude, color = indexes)
, size = 4) +
  scale_color_gradient2("AQI", low = "green", high = "violet", mid = "red", midpoint =
85)
```



Jak wspomiano, nie są to jedyne dane, które przechowujemy w zmiennej **pomiary**. Dla każdego czujnika mamy dostęp do hisorii oraz do prognozy: wykorzystajmy tę pierwszą informację do policzenia korelacji pomiędzy pomiarami poszczególnych czujników.

```
# PRZYKŁAD 10.16
library(corrplot)

historia <- sapply(1:length(pomiary), function(i) sapply(pomiary[[i]]$history$indexes,
function(x) x$value))

C <- cor(historia)
corrplot(C, type = "upper", na.label = "-", diag = F)
```



Jak widać, korelacje są raczej wysokie - tylko jeden czujnik (nr 11) jest wyraźnie inny. Dysponując takimi danymi można się pokusić o sprawdzenie jak korelacja zachowuje się w funkcji odległości. Potrzebujemy do tego funkcji wyznaczającej odegłości pomiędzy punktami określonami przez współrzędne geograficzne (wykorzystamy bibliotekę **geosphere**). Poza tym użyjemy bibliotek **fields** i **Hmisc** do binowania oraz prezentowania słupków niepewności.

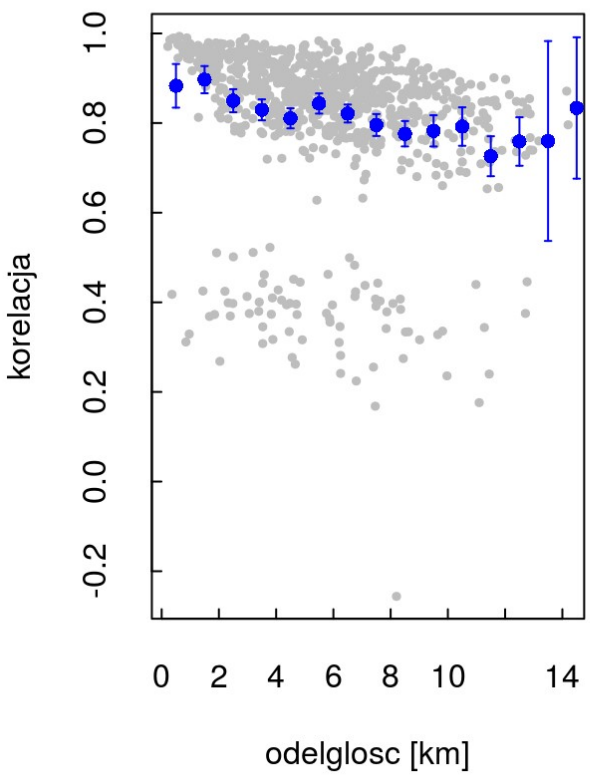
```
# PRZYKŁAD 10.17
library(geosphere, quietly = T, warn.conflicts = F)
library(fields, quietly = T, warn.conflicts = F)
```

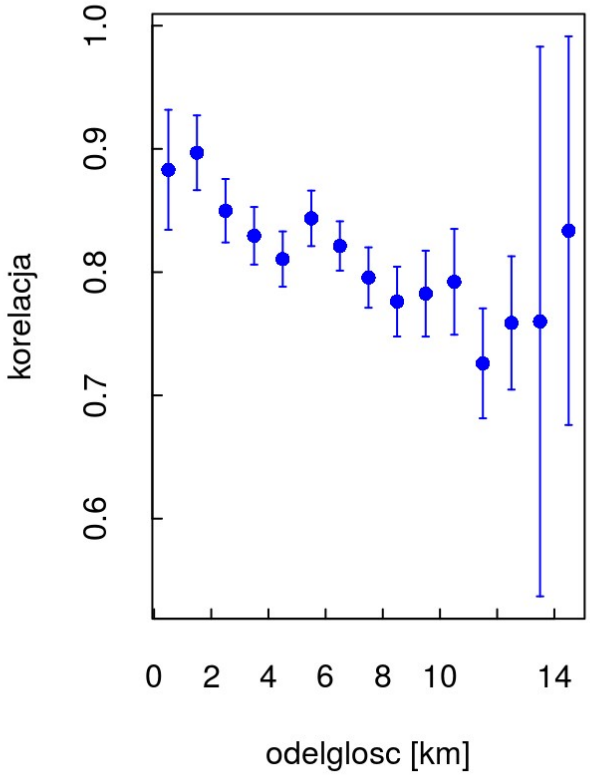
```
## Spam version 2.1-1 (2017-07-02) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

```
library(Hmisc, quietly = T, warn.conflicts = F)

D <- distm(data[c(2, 1)], fun = distHaversine)/1000

d <- stats.bin(D[upper.tri(D)], C[upper.tri(C)])

x <- d$centers
y <- d$stats[2,]
yerr <- d$stats[3] / sqrt(d$stats[1,])

par(mfrow = c(1,2))
plot(D[upper.tri(D)], C[upper.tri(C)], pch = 19, cex = 0.5, col = "gray", tcl = 0.25,
xlab = "odelglosc [km]", ylab = "korelacja")
errbar(x, y, y + yerr, y - yerr, add = T, col = "blue", errbar.col = "blue")
errbar(x, y, y + yerr, y - yerr, col = "blue", errbar.col = "blue", tcl = 0.25, xlab =
"odelglosc [km]", ylab = "korelacja")
```






```
par(mfrow = c(1,1))
```