Datenbereinigung

Pakete laden

```
library(tidyverse)
library(lubridate)
library(spatstat.utils)
rm(list = ls())
path_workfiles <- "/Users/patrickschulze/desktop/inla_research_project/data/workfiles"
```

Daten laden

Zunächst werden die zuvor heruntergeladenen Daten eingelesen. Unter path_workfiles existiert für jeden Sensortyp eine Tabelle mit allen Daten dieses Typs (siehe "Sensordownload"). Alle Sensoren eines Sensortyps werden nun in einem eigenen Tibble gespeichert. Die Tibbles werden in einer Liste zusammengefasst.

```
# Sensortypen in München
types <- c("bme280", "bmp180", "dht22", "sds011")
# Daten laden
# Für jeden Sensortyp ein Tibble
# Tibbles werden in Liste gespeichert
filename_in <- paste(path_workfiles, paste0(types, ".csv"), sep ="/")
sensors_munich <- filename_in %>% purrr::map(readr::read_delim, delim=",")
names(sensors_munich) <- types
```

Variablen entfernen/generieren

Einige Variablen enthalten ausschließlich fehlende Werte bzw. werden nicht benötigt - diese werden gelöscht. Zur Quantifizierung des Feinstaubs stehen die Einheiten PM10 und PM2.5 (Feinstaubpartikel mit 10 bzw. 2.5 oder weniger Mikrometern Durchmesser) zur Verfügung; wir wählen PM10 (entspricht P1 in den heruntergeladenen Daten) aus. Zudem wird aus dem Zeitstempel eine Variable für die Stunde (1,2,. . . ,24), sowie eine für den Tag (1,2,. . . ,21) erstellt, wobei Tag 1 dem 25.07.2019 entspricht, Tag 2 dem 26.07.2019, . . . , und Tag 21 dem 14.08.2019.

```
# Nicht benötigte bzw. Variablen mit fehlenden Messwerten entfernen
# Zur Quantifizierung des Feinstaubs wählen wir PM10(=P1) aus.
sensors_munich$bme280 <-
  sensors_munich$bme280[, c("sensor_id","lat", "lon", "timestamp", "pressure")] %>%
  dplyr::rename(id_bme280 = sensor_id)

sensors_munich$bmp180 <-
  sensors_munich$bmp180[, c("sensor_id","lat", "lon", "timestamp", "pressure")] %>%
  dplyr::rename(id_bmp180 = sensor_id)
```

```r
sensors_munich$dht22 <-
  sensors_munich$dht22[, c("sensor_id","lat", "lon",
                           "timestamp", "temperature", "humidity")]%>%
  dplyr::rename(id_dht22 = sensor_id)

sensors_munich$sds011 <-
  sensors_munich$sds011[, c("sensor_id","P1", "lat", "lon", "timestamp")] %>%
  dplyr::rename(pm10 = P1, id_sds011 = sensor_id)

# Stunde (1,2...,24) und Tag (1,2,...,21) erstellen
create_time_vars <- function(x){
  x %>%
    dplyr::mutate(hour = lubridate::hour(timestamp)+1,
                  date = lubridate::date(timestamp)) %>%
    dplyr::mutate(day = lubridate::day(date)-24) %>%
    dplyr::mutate(day = day+(day<0)*31) %>%
    dplyr::select(-"timestamp")
}
sensors_munich <- sensors_munich %>% purrr::map(create_time_vars)
print(sensors_munich)
```

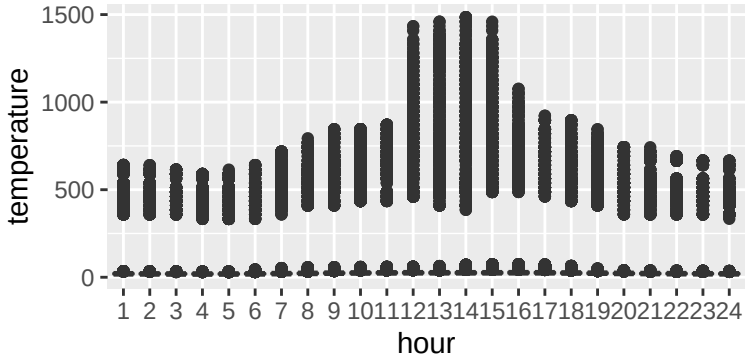
```
## $bme280
## # A tibble: 285,686 x 7
##    id_bme280   lat   lon pressure  hour date         day
##        <dbl> <dbl> <dbl>    <dbl> <dbl> <date>     <dbl>
##  1     10093  48.1  11.5   95283.     1 2019-07-25     1
##  2     10093  48.1  11.5   95285.     1 2019-07-25     1
##  3     10093  48.1  11.5   95284.     1 2019-07-25     1
##  4     10093  48.1  11.5   95284.     1 2019-07-25     1
##  5     10093  48.1  11.5   95286.     1 2019-07-25     1
##  6     10093  48.1  11.5   95288.     1 2019-07-25     1
##  7     10093  48.1  11.5   95291.     1 2019-07-25     1
##  8     10093  48.1  11.5   95294.     1 2019-07-25     1
##  9     10093  48.1  11.5   95292.     1 2019-07-25     1
## 10     10093  48.1  11.5   95290.     1 2019-07-25     1
## # ... with 285,676 more rows
##
## $bmp180
## # A tibble: 54,690 x 7
##    id_bmp180   lat   lon pressure  hour date         day
##        <dbl> <dbl> <dbl>    <dbl> <dbl> <date>     <dbl>
##  1     14023  48.1  11.6    95528     1 2019-07-25     1
##  2     14023  48.1  11.6    95525     1 2019-07-25     1
##  3     14023  48.1  11.6    95528     1 2019-07-25     1
##  4     14023  48.1  11.6    95531     1 2019-07-25     1
##  5     14023  48.1  11.6    95529     1 2019-07-25     1
##  6     14023  48.1  11.6    95527     1 2019-07-25     1
##  7     14023  48.1  11.6    95529     1 2019-07-25     1
##  8     14023  48.1  11.6    95527     1 2019-07-25     1
##  9     14023  48.1  11.6    95524     1 2019-07-25     1
## 10     14023  48.1  11.6    95523     1 2019-07-25     1
## # ... with 54,680 more rows
##
## $dht22
```

```
## # A tibble: 1,258,438 x 8
##    id_dht22   lat   lon temperature humidity  hour date         day
##       <dbl> <dbl> <dbl>       <dbl>    <dbl> <dbl> <date>     <dbl>
##  1    10109  48.2  11.5        27.5     84       1 2019-07-25     1
##  2    10109  48.2  11.5        27.5     84.1     1 2019-07-25     1
##  3    10109  48.2  11.5        27.5     84       1 2019-07-25     1
##  4    10109  48.2  11.5        27.5     84.2     1 2019-07-25     1
##  5    10109  48.2  11.5        27.5     84.1     1 2019-07-25     1
##  6    10109  48.2  11.5        27.5     84       1 2019-07-25     1
##  7    10109  48.2  11.5        27.5     84       1 2019-07-25     1
##  8    10109  48.2  11.5        27.5     84.1     1 2019-07-25     1
##  9    10109  48.2  11.5        27.5     84       1 2019-07-25     1
## 10    10109  48.2  11.5        27.5     84.3     1 2019-07-25     1
## # ... with 1,258,428 more rows
##
## $sds011
## # A tibble: 1,808,999 x 7
##    id_sds011  pm10   lat   lon  hour date         day
##        <dbl> <dbl> <dbl> <dbl> <dbl> <date>     <dbl>
##  1     10092  9.7   48.1  11.5     1 2019-07-25     1
##  2     10092  9.53  48.1  11.5     1 2019-07-25     1
##  3     10092 10.1   48.1  11.5     1 2019-07-25     1
##  4     10092  9.93  48.1  11.5     1 2019-07-25     1
##  5     10092  9.43  48.1  11.5     1 2019-07-25     1
##  6     10092 10     48.1  11.5     1 2019-07-25     1
##  7     10092  9.47  48.1  11.5     1 2019-07-25     1
##  8     10092  9.37  48.1  11.5     1 2019-07-25     1
##  9     10092  9.87  48.1  11.5     1 2019-07-25     1
## 10     10092  9.23  48.1  11.5     1 2019-07-25     1
## # ... with 1,808,989 more rows
```

Ausreißer entfernen

Die Sensoren messen zum Teil falsche Werte. Siehe zum Beispiel Temperatur:

```
ggplot(data=sensors_munich$dht22, aes(x=as.factor(hour), y=temperature)) +
  geom_boxplot() +
  labs(x = "hour")
```



Um Ausreißer zu erkennen, versuchen wir für die verschiedenen gemessenen Variablen jeweils auffällige Sensoren zu finden, da diese möglicherweise generell fehlerhaft messen (d.h. auch für Messwerte dieser Sensoren welche in einem realistischen Bereich liegen). Diese Sensoren werden dann ggf. entfernt. Zudem

werden alle Einzelmesswerte außerhalb eines realistischen Bereichs entfernt. Dieses Vorgehen wenden wir in ähnlicher Form für alle Variablen an.

```
# Liste für ausreißerbereinigte Daten initialisieren
sensors_munich_cleaned <- list(bme280 = NULL, bmp180 = NULL, dht22 = NULL, sds011 = NULL)
```

Ausreißer Temperatur

Laut Wetterbericht bewegte sich die Temperatur im Beobachtungszeitraum zwischen 8°C und 32°. Wir kontrollieren, ob Werte außerhalb dieses Bereichs vorkommen und überprüfen Sensoren welche fehlerhaft messen. Wir verwenden eine Toleranz von 5° (nach unten jedoch nicht nötig, da keine Ausreißer vorhanden).

```
# Keine Temperatur unter 8° gemessen
sensors_munich$dht22 %>%
  dplyr::filter(spatstat.utils::inside.range(temperature, c(-Inf,7)))
```

```
## # A tibble: 0 x 8
## # ... with 8 variables: id_dht22 <dbl>, lat <dbl>, lon <dbl>,
## #   temperature <dbl>, humidity <dbl>, hour <dbl>, date <date>, day <dbl>
```

```
# Ausreißer nach oben aber vorhanden
(dht22_outliers_temperature <- sensors_munich$dht22 %>%
   dplyr::filter(spatstat.utils::inside.range(temperature, c(38, Inf))))
```

```
## # A tibble: 39,373 x 8
##    id_dht22   lat   lon temperature humidity  hour date         day
##       <dbl> <dbl> <dbl>       <dbl>    <dbl> <dbl> <date>     <dbl>
##  1     1015  48.1  11.6        38.3     62.4     8 2019-07-25     1
##  2     1015  48.1  11.6        38.5     60.8     8 2019-07-25     1
##  3     1015  48.1  11.6        38.1     57.4     8 2019-07-25     1
##  4     1015  48.1  11.6        38       56.5     8 2019-07-25     1
##  5     1015  48.1  11.6        38.4     55.9     8 2019-07-25     1
##  6     1015  48.1  11.6        38.6     54.7     8 2019-07-25     1
##  7     1015  48.1  11.6        38.9     54.1     8 2019-07-25     1
##  8     1015  48.1  11.6        38.4     54.2     8 2019-07-25     1
##  9    10534  48.1  11.7        38.7     53.1     8 2019-07-25     1
## 10    10534  48.1  11.7        38.5     52.6     8 2019-07-25     1
## # ... with 39,363 more rows
```

Wir berechnen nun für alle Sensoren, d.h. nicht nur für die Ausreißer, die durschnittliche Temperatur je Sensor. Anschließend wird in einer Tabelle zusammengefasst, welche Sensoren prozentual die meisten Ausreißer aufweisen und wie hoch die durchschnittlich gemessene Temperatur des jeweiligen Sensors ist.

```
sensor_mean_temperature <- sensors_munich$dht22 %>% dplyr::group_by(id_dht22) %>%
  dplyr::summarize(mean_temperature=mean(temperature))

n_defect <- dht22_outliers_temperature %>%
  dplyr::group_by(id_dht22) %>% dplyr::tally() %>%
  dplyr::rename(n_defect = n) %>% dplyr::arrange(desc(n_defect))
n_total <- sensors_munich$dht22 %>%
  dplyr::group_by(id_dht22) %>% dplyr::tally() %>%
  dplyr::filter(id_dht22 %in% n_defect$id_dht22) %>%
  dplyr::rename(n_total = n) %>% dplyr::arrange(desc(n_total))
percent_defect <- n_defect %>% dplyr::inner_join(n_total, by = "id_dht22") %>%
  dplyr::inner_join(sensor_mean_temperature, by = "id_dht22") %>%
  dplyr::mutate(percent_defect = 100*n_defect/n_total) %>%
```

```
  dplyr::arrange(desc(percent_defect))
print(percent_defect, n=5)
```

```
## # A tibble: 79 x 5
##   id_dht22 n_defect n_total mean_temperature percent_defect
##      <dbl>    <int>   <int>            <dbl>          <dbl>
## 1    24900    11631   11631            591.           100
## 2    26897     6479   46679             29.7           13.9
## 3    16527     1592   11655             28.1           13.7
## 4     3442      181    1337             29.3           13.5
## 5     7806     1371   11756             24.5           11.7
## # ... with 74 more rows
```

Fazit: Der Sensor mit ID 24900 ist defekt, da 100% der Messungen über 37° liegen. Die anderen Sensoren scheinen im Allgemeinen zu funktionieren. Wir entfernen alle Ausreißer (und somit u.a. den Sensor mit ID 24900 komplett). Da der gleiche Sensor (dht22) auch Luftfeuchtigkeit misst, werden zuerst die Indices der Temperatur-Ausreißer gespeichert und die Ausreißer später gemeinsam mit den Luftdruck-Ausreißern gelöscht.

```
outliers_dht22_temperature <-
  !spatstat.utils::inside.range(sensors_munich$dht22$temperature, c(0, 38))
```

Ausreißer Luftfeuchtigkeit

Laut Wetterbericht betrug die Luftfeuchtigkeit im Beobachtungszeitraum zwischen 26% und 100%. Zudem war die Luftfeuchtiegkeit laut Wetterbericht im Durchschnitt hoch, was durch die Daten bestätigt wird: Das 60% Quantil ist bei fast 100%. Unser Vorgehen ist equivalent zum Vorgehen bei den Temperatur-Ausreißern, d.h. auch hier erstellen wir zunächst eine Tabelle, welche die schlechtesten Sensoren und dessen durchschnittlichen Messwerte auflistet.

```
quantile(sensors_munich$dht22[,"humidity", drop=TRUE], 0.6, na.rm = TRUE)
```

```
##  60%
## 99.9
```

```
# Ausreißer filtern
# Auch hier etwas Toleranz berücksichtigen (nach oben jedoch nicht, da mehr als 100% unmöglich)
(dht22_outliers_humidity <- sensors_munich$dht22 %>%
dplyr::filter(!spatstat.utils::inside.range(humidity, c(21, 100))))
```

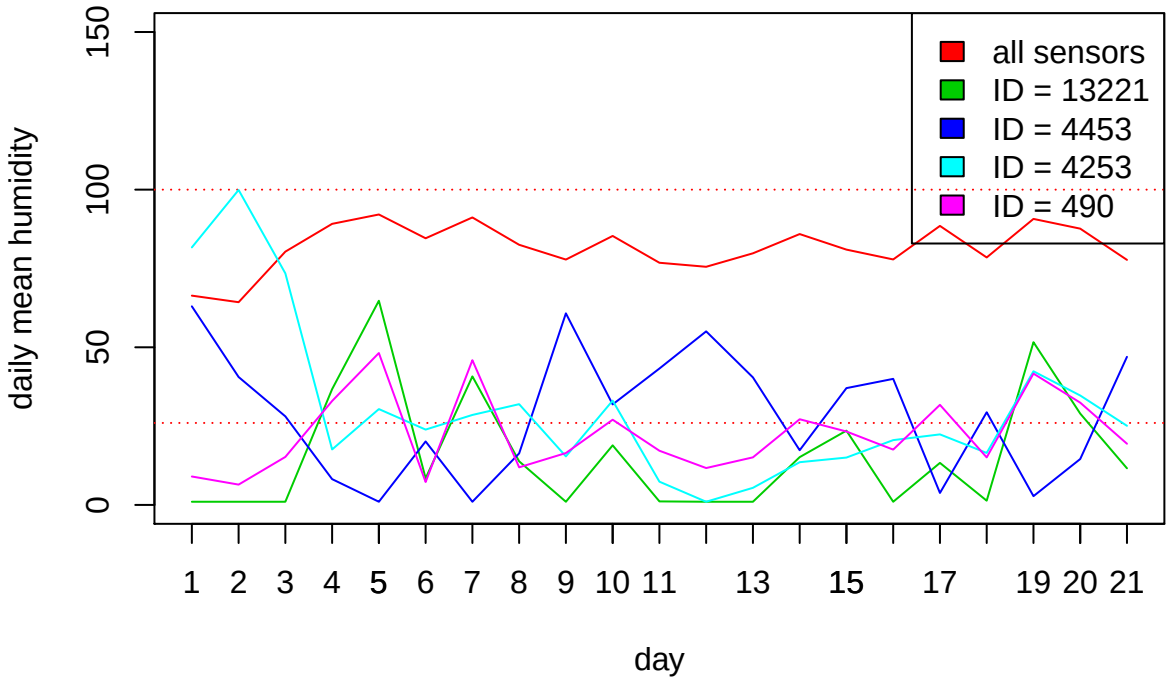
```
## # A tibble: 54,361 x 8
##    id_dht22   lat   lon temperature humidity  hour date         day
##       <dbl> <dbl> <dbl>       <dbl>    <dbl> <dbl> <date>     <dbl>
##  1    12166  48.1  11.5        42.8     20.1     9 2019-07-25     1
##  2    12166  48.1  11.5        42.9     19.8     9 2019-07-25     1
##  3    12166  48.1  11.5        42.2     20.2     9 2019-07-25     1
##  4    12166  48.1  11.5        42.3     20.3     9 2019-07-25     1
##  5    12166  48.1  11.5        42.2     19.1     9 2019-07-25     1
##  6    12166  48.1  11.5        41.8     18       9 2019-07-25     1
##  7    12166  48.1  11.5        41.4     19.2     9 2019-07-25     1
##  8    12166  48.1  11.5        41.3     20.1     9 2019-07-25     1
##  9    12166  48.1  11.5        41.1     19.3     9 2019-07-25     1
## 10    12166  48.1  11.5        41.2     19.7     9 2019-07-25     1
## # ... with 54,351 more rows
```

```r
sensor_mean_humidity <- sensors_munich$dht22 %>% dplyr::group_by(id_dht22) %>%
  dplyr::summarize(mean_humidity=mean(humidity, na.rm = TRUE))

n_defect <- dht22_outliers_humidity %>%
  dplyr::group_by(id_dht22) %>% dplyr::tally() %>%
  dplyr::rename(n_defect = n) %>% dplyr::arrange(desc(n_defect))
n_total <- sensors_munich$dht22 %>%
  dplyr::group_by(id_dht22) %>% dplyr::tally() %>%
  dplyr::filter(id_dht22 %in% n_defect$id_dht22) %>%
  dplyr::rename(n_total = n) %>% dplyr::arrange(desc(n_total))
percent_defect <- n_defect %>% dplyr::inner_join(n_total, by = "id_dht22") %>%
  dplyr::inner_join(sensor_mean_humidity, by = "id_dht22") %>%
  dplyr::mutate(percent_defect = 100*n_defect/n_total) %>%
  dplyr::arrange(desc(percent_defect))
print(percent_defect, n = 8)
```

```
## # A tibble: 24 x 5
##   id_dht22 n_defect n_total mean_humidity percent_defect
##      <dbl>    <int>   <int>         <dbl>          <dbl>
## 1    24900    11631   11631        1990.          100
## 2    13221     8958   11683          16.0          76.7
## 3     4453     8208   11651          28.7          70.4
## 4     4253     8026   11662          30.7          68.8
## 5      490     5021    9828          21.0          51.1
## 6     1339     3000   11647          38.7          25.8
## 7    26897     5272   46679          41.6          11.3
## 8      937      915   11830          64.5           7.73
## # ... with 16 more rows
```

Fazit: Sensor mit ID 24900 defekt; dieser war auch für die Temperatur defekt. Die zweit- bis fünftschlechtesten Sensoren haben jedoch auch sehr viele Ausreißer (zu niedrige Werte).Daher überprüfen wir diese Sensoren nun noch etwas genauer.

```r
# Mittelwert je Tag dieser Sensoren im Vergleich
# zum Gesamtmittelwert (um eindeutige Ausreißer bereinigt) plotten
sensors_munich$dht22 %>%
  dplyr::filter(id_dht22 != "24000", spatstat.utils::inside.range(humidity, c(21, 100))) %>%
  dplyr::group_by(day) %>%
  dplyr::summarize(mean_humidity = mean(humidity, na.rm = TRUE)) %>%
  plot(type = "l", ylim=c(0,150), col = 2, ylab = "daily mean humidity")
axis(1,at=1:24)
abline(h = 100, col = "red", lty = 3)
abline(h = 26, col = "red", lty = 3)

ids_defect <- percent_defect[2:5,] %>% dplyr::pull(id_dht22)
i=3
for (id in ids_defect) {
  sensors_munich$dht22 %>% dplyr::filter(id_dht22 == id) %>%
    dplyr::group_by(day) %>%
    dplyr::summarize(mean_humidity = mean(humidity, na.rm =TRUE)) %>%
    lines(col = i)
i <- i+1 }
legend("topright", c("all sensors", paste0("ID = ", ids_defect)), fill = 2:6)
```



Diese Sensoren scheinen ebenso defekt, da die Messwerte für Großteil des Zeitraums zu gering sind und teils stark schwanken. Daher löschen wir auch diese Sensoren komplett.

```
del_12345 <- sensors_munich$dht22$id_dht22 %in% unlist(percent_defect[1:5,"id_dht22"])
```

Zudem löschen wir alle einzelnen Ausreißer und bestimmen die gesamte Menge der Ausreißer, d.h. die Vereinigungsmenge der vollständig entfernten Sensoren und der einzelnen Ausreißer.

```
del_single <- !spatstat.utils::inside.range(sensors_munich$dht22$humidity, c(21, 100))
# Alle Ausreißer für Luftfeuchtigkeit
outliers_dht22_humidity <- del_12345 | del_single
```

Nun können können wir alle Ausreißer des Typs dht22, welcher Temperatur und Luftfeuchtigkeit misst, bestimmen. Wir entfernen eine Beobachtung sicherheitshalber auch, wenn nur einer der beiden Werte falsch ist.

```
 # Alle Ausreißer des Typs dht22 (Temperatur und Luftfeuchtigkeit)
outliers_dht22 <- (outliers_dht22_humidity | outliers_dht22_temperature)
length(which(outliers_dht22))/length(sensors_munich$dht22$humidity)
```

```
## [1] 0.06995259
```

Insgesamt knapp 7% Asureißer werden für Sensor dht22 entfernt. Die bereinigten Daten werden in der Liste sensors_munich_cleaned gespeichert.

```
# Entfernen
sensors_munich_cleaned$dht22 <- sensors_munich$dht22[!outliers_dht22,]
```

Ausreißer Luftdruck

Der Luftdruck wird von zwei Sensoren gemessen. Wir führen für jeden der beiden Sensoren eine Ausreißeranalyse durch und kombinieren die beiden Werte später (siehe nächste Sektion).

Zuerst betrachten wir den Sensor bme280 und erstellen auch hier wieder die Tabelle analog zu den bisher betrachteten Variablen. Da die exakt gemessenen Luftdruckdaten in diesem Zeitraum schwierig zu überprüfen

sind, klassifizieren wir vereinfachend Werte außerhalb der 2%- und 98% Quantile als Ausreißer.

```
(interval_outliers1 <- quantile(
  sensors_munich$bme280[,"pressure", drop=TRUE], c(0.02,0.98), na.rm = TRUE))
```

```
##       2%      98%
## 93984.22 96023.58
```

```
bme280_outliers_pressure <- sensors_munich$bme280 %>%
  dplyr::filter(!(spatstat.utils::inside.range(pressure,interval_outliers1)))

sensor_mean_bme280 <- sensors_munich$bme280 %>% dplyr::group_by(id_bme280) %>%
  dplyr::summarize(mean_bme280=mean(pressure, na.rm = TRUE))

n_defect <- bme280_outliers_pressure %>%
  dplyr::group_by(id_bme280) %>% dplyr::tally() %>%
  dplyr::rename(n_defect = n) %>% dplyr::arrange(desc(n_defect))
n_total <- sensors_munich$bme280 %>%
  dplyr::group_by(id_bme280) %>% dplyr::tally() %>%
  dplyr::filter(id_bme280 %in% n_defect$id_bme280) %>%
  dplyr::rename(n_total = n) %>% dplyr::arrange(desc(n_total))
percent_defect <- n_defect %>% dplyr::inner_join(n_total, by = "id_bme280") %>%
  dplyr::inner_join(sensor_mean_bme280, by = "id_bme280") %>%
  dplyr::mutate(percent_defect = 100*n_defect/n_total) %>%
  dplyr::arrange(desc(percent_defect))
print(percent_defect, n=5)
```

```
## # A tibble: 19 x 5
##   id_bme280 n_defect n_total mean_bme280 percent_defect
##       <dbl>    <int>   <int>       <dbl>          <dbl>
## 1     29239     2836    9839      95702.          28.8
## 2     27731     1081   11738      95543.           9.21
## 3      6218      993   11662      95537.           8.51
## 4      7662      801   10528      95559.           7.61
## 5     10093      666   11609      94973.           5.74
## # ... with 14 more rows
```

Fazit: Sensor mit ID 29239 hat deutlich mehr Ausreißer als die anderen Sensoren, welche im Allgemeinen in Ordnung zu sein scheinen. Daher entfernen wir diesen Sensor komplett. Zudem löschen wir alle einzelnen Ausreißer.

```
# Entferne Sensor mit ID 29239 komplett
del1 <- sensors_munich$bme280$id_bme280 == "29239"
# Entferne zudem alle einzelnen Ausreißer
del_single <- !spatstat.utils::inside.range(sensors_munich$bme280$pressure,
                                            interval_outliers1)
del <- del1 | del_single
```

Insgesamt ca. 7% der Asureißer werden für Sensor bme280 entfernt. Die bereinigten Daten werden in der Liste sensors_munich_cleaned gespeichert.

```
# Ca. 7% der Daten entfernt
length(which(del))/length(sensors_munich$bme280$pressure)
```

```
## [1] 0.06450089
```

```
# Löschen
sensors_munich_cleaned$bme280 <- sensors_munich$bme[!del,]
```

Die gleiche Prozedur wiederholen wir nun für den anderen Luftdruck-Sensortyp.

```
(interval_outliers2 <- quantile(
  sensors_munich$bmp180[,"pressure", drop=TRUE], c(0.02,0.98), na.rm = TRUE))
```

```
##       2%      98%
## 93880.00 96234.22
```

```
bmp180_outliers_pressure <- sensors_munich$bmp180 %>%
  dplyr::filter(!(spatstat.utils::inside.range(pressure,interval_outliers2)))

sensor_mean_bmp180 <- sensors_munich$bmp180 %>% dplyr::group_by(id_bmp180) %>%
  dplyr::summarize(mean_bmp180=mean(pressure, na.rm = TRUE))

n_defect <- bmp180_outliers_pressure %>%
  dplyr::group_by(id_bmp180) %>% dplyr::tally() %>%
  dplyr::rename(n_defect = n) %>% dplyr::arrange(desc(n_defect))
n_total <- sensors_munich$bmp180 %>%
  dplyr::group_by(id_bmp180) %>% dplyr::tally() %>%
  dplyr::filter(id_bmp180 %in% n_defect$id_bmp180) %>% dplyr::rename(n_total = n) %>%
  dplyr::arrange(desc(n_total))
percent_defect <- n_defect %>% dplyr::inner_join(n_total, by = "id_bmp180") %>%
  dplyr::inner_join(sensor_mean_bmp180, by = "id_bmp180") %>%
  dplyr::mutate(percent_defect = 100*n_defect/n_total) %>%
  dplyr::arrange(desc(percent_defect))
print(percent_defect, n=5)
```

```
## # A tibble: 6 x 5
##   id_bmp180 n_defect n_total mean_bmp180 percent_defect
##       <dbl>    <int>   <int>       <dbl>          <dbl>
## 1     28574     1094   10254      95795.          10.7
## 2      4069      237    2978      94695.           7.96
## 3      3695      274   11237      95181.           2.44
## 4     14023      258   11655      95168.           2.21
## 5      8112      150    6826      95114.           2.20
## # ... with 1 more row
```

Diesmal gibt es keine so deutlichen Unterschiede wie zuvor. Wir löschen deshalb nur die einzelnen Ausreißer und nicht den ganzen Sensor. Insgesamt ca. 4% der Ausreißer werden für Sensor bmp180 entfernt. Die bereinigten Daten werden wie zuvor in der Liste sensors_munich_cleaned gespeichert.

```
# Einzelne Ausreißer
del <- !spatstat.utils::inside.range(sensors_munich$bmp180$pressure, interval_outliers2)
# ca. 4% werden geläscht
length(which(del))/length(sensors_munich$bmp180$pressure)
```

```
## [1] 0.03995246
```

```
# Löschen
sensors_munich_cleaned$bmp180 <- sensors_munich$bmp180[!del,]
```

Ausreißer Feinstaub (PM10)

Laut der Stadt München wird der PM10-Tagesmittelwert von $50 \mu g/m3$ pro Jahr 16 mal überschritten, siehe https://www.muenchen.de/rathaus/Stadtverwaltung/Referat-fuer-Gesundheit-und-Umwelt/Luft_und_

Strahlung/Stickstoffdioxidmessungen.html. Daher halten wir eine Klassifizierung von Ausreißern ab Werten von größer als $150 \mu g/m3$ für gerechtfertigt. Dies entspricht ca. dem 99.4% Quantil:

```
quantile(sensors_munich$sds011[,"pm10", drop=TRUE], 0.994)
```

```
##    99.4%
## 163.2004
```

Um verdächtige Sensoren zu erkennen, suchen wir zunächst bereits ab PM10 >= 100. Wieder verfahren wir wie oben, d.h. wir erstellen zunächst eine Tabelle der schlechtesten Sensoren.

```
pm10_outliers <- sensors_munich$sds011 %>%
  dplyr::filter(spatstat.utils::inside.range(pm10, c(100, Inf)))

sensor_mean_pm10 <- sensors_munich$sds011 %>% dplyr::group_by(id_sds011) %>%
  dplyr::summarize(mean_pm10=mean(pm10))

n_defect <- pm10_outliers %>%
  dplyr::group_by(id_sds011) %>% dplyr::tally() %>%
  dplyr::rename(n_defect = n) %>% dplyr::arrange(desc(n_defect))
n_total <- sensors_munich$sds011 %>%
  dplyr::group_by(id_sds011) %>% dplyr::tally() %>%
  dplyr::filter(id_sds011 %in% n_defect$id_sds011) %>% dplyr::rename(n_total = n) %>%
  dplyr::arrange(desc(n_total))
(percent_defect <- n_defect %>% dplyr::inner_join(n_total, by = "id_sds011") %>%
    dplyr::inner_join(sensor_mean_pm10, by = "id_sds011") %>%
    dplyr::mutate(percent_defect = 100*n_defect/n_total) %>%
    dplyr::arrange(desc(percent_defect)))
```

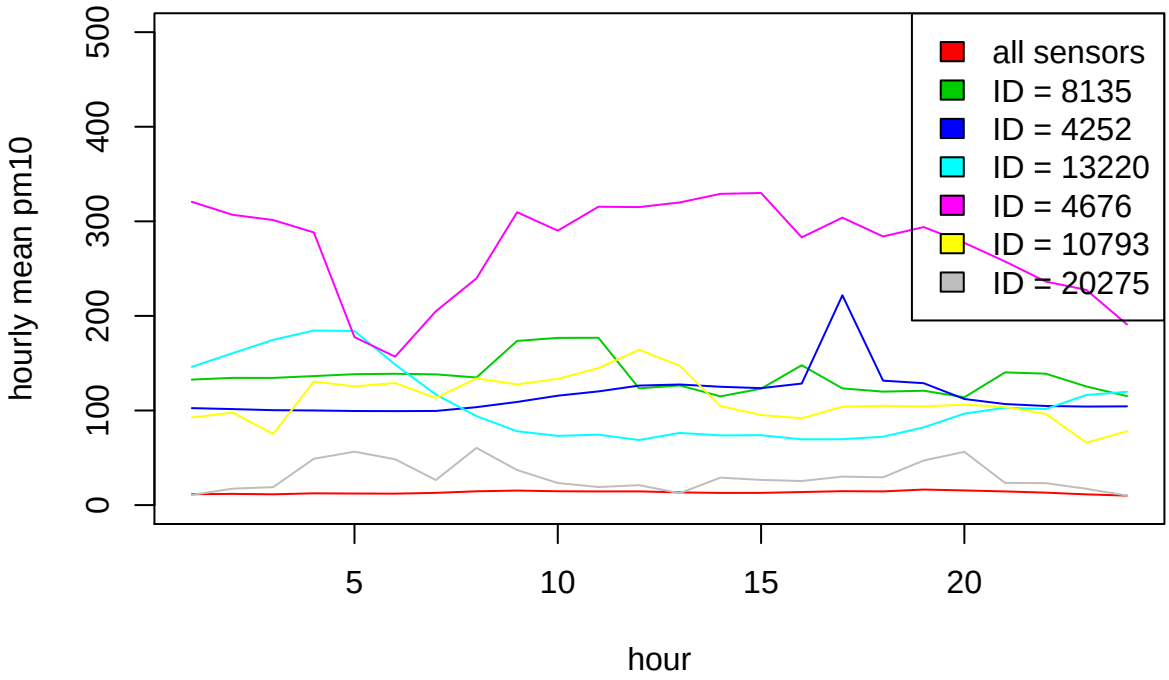
```
## # A tibble: 105 x 5
##    id_sds011 n_defect n_total mean_pm10 percent_defect
##        <dbl>    <int>   <int>     <dbl>          <dbl>
##  1     14860     1768    1768   2000.          100
##  2      8135     6384   10174    135.           62.7
##  3      4252     7186   11603    117.           61.9
##  4     13220     3664   11617    107.           31.5
##  5      4676     1067    6097    273.           17.5
##  6     10793     1381   11422    111.           12.1
##  7     20275     1243   26540     30.3           4.68
##  8     26896     1616   47811     21.5           3.38
##  9     20859       59    5747      9.74          1.03
## 10      4893       19    2268     16.2           0.838
## # ... with 95 more rows
```

Fazit: Die 6 schlechtesten Sensoren messen einen deutlich höheren Durchschnittswert als die restlichen Sensoren. Prozentual haben diese Sensoren auch deutlich mehr Ausreißer. Der Sensor mit ID 14860 ist defekt und wird deshalb vollständig entfernt. Die anderen Sensoren untersuchen wir nun genauer, um zu beurteilen, ob diese ebenfalls entfernt werden müssen.

Zuerst plotten wir die durchschnittlichen Werte dieser Sensoren je Stunde und vergleichen mit dem Durchschnitt aller Sensoren:

```
# Plotte Mittelwert je Stunde für potentiell defekte Sensoren
sensors_munich$sds011 %>%
  dplyr::group_by(hour) %>% dplyr::summarize(mean_pm10 = mean(pm10)) %>%
  plot(type = "l", ylim=c(0,500), col = 2, ylab = "hourly mean pm10")
ids_defect <- percent_defect[2:7,] %>% pull(id_sds011)
```

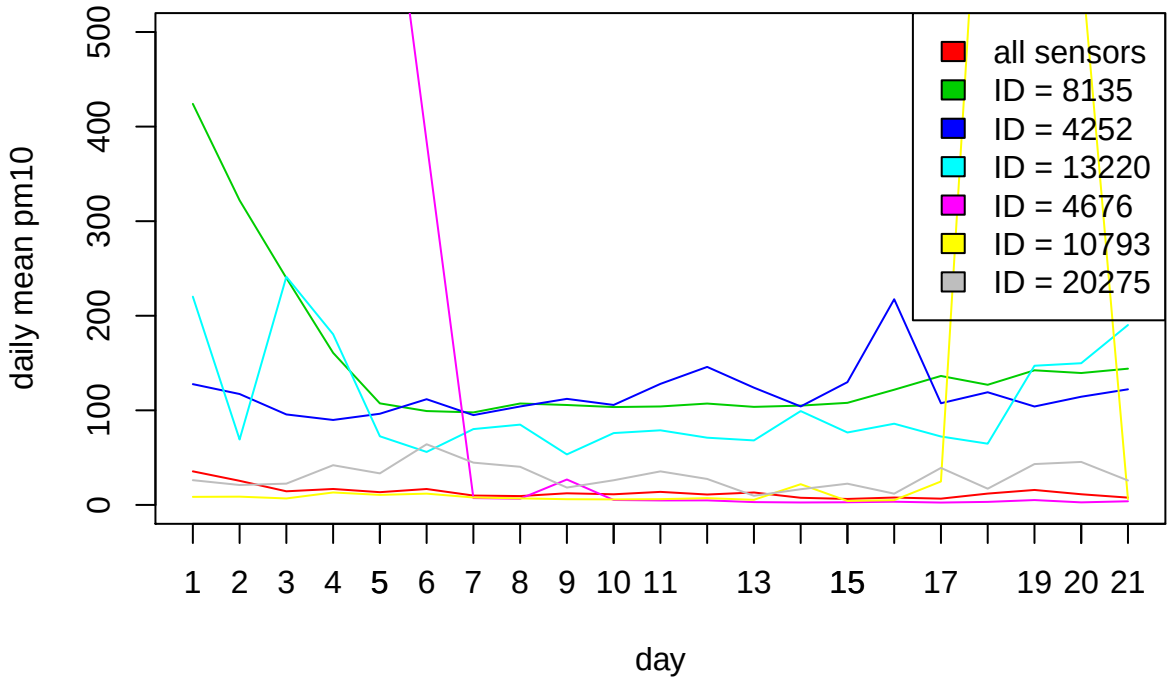
```r
i=3
for (id in ids_defect) {
  sensors_munich$sds011 %>% dplyr::filter(id_sds011 == id) %>%
    dplyr::group_by(hour) %>% dplyr::summarize(mean_pm10 = mean(pm10)) %>%
    lines(col = i)
  i <- i+1
}
legend("topright", c("all sensors", paste0("ID = ", ids_defect)), fill = 2:8)
```



Ergebnis: Die Werte sind konstant hoch. Dies ist sehr unrealistisch, z.B. kommen Sonneneinstrahlung oder Autoverkehr als mögliche Gründe nun kaum mehr in Frage. Ab einschließlich Sensor 7 (ID 20275) sind die Werte dann aber im möglichen Bereich.

Nun plotten wir die durchschnittlichen Werte je Tag und vergleichen erneut mit dem Durchschnitt aller Sensoren:

```r
# Gleiche Prozedur für Mittelwert je Tag
sensors_munich$sds011 %>%
  dplyr::group_by(day) %>% dplyr::summarize(mean_pm10 = mean(pm10)) %>%
  plot(type = "l", ylim=c(0,500), col = 2, ylab = "daily mean pm10")
axis(1,at=1:24)

ids_defect <- percent_defect[2:7,] %>% pull(id_sds011)
i=3
for (id in ids_defect) {
  sensors_munich$sds011 %>% filter(id_sds011 == id) %>%
    dplyr::group_by(day) %>% dplyr::summarize(mean_pm10 = mean(pm10)) %>%
    lines(col = i)
  i <- i+1
}
legend("topright", c("all sensors", paste0("ID = ", ids_defect)), fill = 2:8)
```



Ergebnis: Die Sensoren 4676 und 10793 sind in bestimmten Perioden defekt.

Wir gehen daher wie folgt vor: Wir entfernen die Sensoren 8135, 4252 und 13220, da die Werte sehr unrealistisch erscheinen (konstant hoch). Sensor 4676 wird für Tage 1-6 entfernt und Sensor 10793 für Tage 18-21.

```
# Sensoren 1-4 (IDs 8135, 4252, 13220)
del_1234 <- sensors_munich$sds011$id_sds011 %in% ids_defect[1:4]
# Sensor 5 (ID 4676) für Tage 1-6
del_5 <- (sensors_munich$sds011$id_sds011 %in% ids_defect[5]) & (sensors_munich$sds011$day %in% 1:6)
# Sensor 6 (ID 10793) für Tage 18-21
del_6 <- (sensors_munich$sds011$id_sds011 %in% ids_defect[6]) & (sensors_munich$sds011$day %in% 18:21)
```

Nun werden, wie oben bereits erläutert, noch alle Einzelmesswerte >= 150 entfernt.

```
del_single <- sensors_munich$sds011$pm10 >= 150
```

Wir bilden die Vereinigungsmenge und löschen alle Ausreißer. Insgesamt ca. 3% der Ausreißer werden für Sensor sds011 entfernt.

```
# Vereinigunsmenge bilden
outliers_sds011 <- del_1234 | del_5 | del_6 | del_single
# Anteil der Ausreißer bestimmen
length(which(outliers_sds011))/length(sensors_munich$sds011$pm10)
```

```
## [1] 0.02962135
```

```
# Enfternen
sensors_munich_cleaned$sds011 <- sensors_munich$sds011[!outliers_sds011,]
```

Tibbles der verschiedenen Sensortypen mergen

Nun werden die bisher separaten Tibbles (pro Sensortyp ein Tibble) zu einem Datensatz zusammengefasst. Dazu werden je Stunde die durchschnittlichen Werte von Feinstaub, Temperatur und Feuchtigkeit pro

Sensor gebildet. Der Feinstaubsensor (sds011) wird dann an den Temperatur-/Feuchtigkeitssensor (dht22) gespielt, wobei als eindeutiger Schlüssel der Ort (lat, lon) und die Zeit (day, hour) verwendet wird. Dadurch gehen alle Beobachtungen dieser beiden Sensortypen verloren, welche nicht am gleichen Ort und/oder zur gleichen Zeit gemessen wurden. Von den ca. 69000 stündlich aggregierten Feinstaubdaten und den ca. 48100 Temperatur-/Luftfeuchtigkeitsdaten bleiben am Ende ca. 43800 Daten übrig.

Für den Luftdruck (bme280 und bmp180) gibt es jedoch deutlich weniger Sensoren. Deshalb berechnen wir hier einen stündlichen Mittelwert über alle Sensoren hinweg. Dieser Mittelwert wird dann über den Zeitschlüssel (day, hour) an die beiden anderen zuvor zusammengefügten Sensoren gespielt. Hierbei gehen keine Daten verloren. Da es zwei verschiedene Luftdrucksensoren gibt, die jedoch die gleiche Einheit (Pascal) messen, mitteln wir die Werte beider Sensoren.

```
# Funktion um Mittelwert pro Stunde und Sensor für Variable var eines tibbles x zu berechn
hourly_mean_per_sensor <- function(x, var){
  x %>%
    dplyr::group_by(lat, lon, day, hour) %>%
    dplyr::summarize_at(dplyr::vars(var), mean, na.rm=TRUE) %>%
    dplyr::ungroup()
}
# Funktion um Mittelwert pro Stunde für Variable var eines tibbles x zu berechnen
hourly_mean_agg <- function(x, var){
  x %>%
    dplyr::group_by(day, hour) %>%
    dplyr::summarize_at(dplyr::vars(var), mean, na.rm=TRUE) %>%
    dplyr::ungroup()
}
# Schlüssel kreieren, um verschiedene Sensortypen anzuspielen
make_key1 <- function(x){
  x %>%
    dplyr::mutate(key1 = paste(lat, lon, day, hour, sep="-"))
}

make_key2 <- function(x){
  x %>%
    dplyr::mutate(key2 = paste(day, hour, sep="-"))
}
# Funktion anwenden, um Mittelwert pro Stunde und Sensor für Sensortypen
# sds011 (PM10) und dht22 (Temperatur, Feuchtigkeit) zu berechnen
new_data1 <- sensors_munich_cleaned[c("sds011", "dht22")] %>%
  purrr::map2(list("pm10", c("temperature", "humidity")),
              hourly_mean_per_sensor) %>%
  purrr::map(make_key1) %>%
  purrr::map(make_key2)
# Aggregierte PM10- und Temperatur-/Feuchtigkeitsdaten vor dem Mergen:
new_data1[c("sds011", "dht22")]
```

```
## $sds011
## # A tibble: 69,039 x 7
##      lat   lon   day  hour  pm10 key1               key2
##    <dbl> <dbl> <dbl> <dbl> <dbl> <chr>              <chr>
##  1  48.1  11.5     1     1 39.4  48.078-11.516-1-1  1-1
##  2  48.1  11.5     1     2 10.6  48.078-11.516-1-2  1-2
##  3  48.1  11.5     1     3 10.7  48.078-11.516-1-3  1-3
##  4  48.1  11.5     1     4 11.6  48.078-11.516-1-4  1-4
##  5  48.1  11.5     1     5 12.2  48.078-11.516-1-5  1-5
```

```
##  6  48.1  11.5     1     6 11.9  48.078-11.516-1-6  1-6
##  7  48.1  11.5     1     7 12.2  48.078-11.516-1-7  1-7
##  8  48.1  11.5     1     8 11.1  48.078-11.516-1-8  1-8
##  9  48.1  11.5     1     9  8.09 48.078-11.516-1-9  1-9
## 10  48.1  11.5     1    10  4.77 48.078-11.516-1-10 1-10
## # ... with 69,029 more rows
##
## $dht22
## # A tibble: 48,122 x 8
##      lat   lon   day  hour temperature humidity key1               key2
##    <dbl> <dbl> <dbl> <dbl>       <dbl>    <dbl> <chr>              <chr>
##  1  48.1  11.5     1     1        22.3     99.9 48.078-11.516-1-1  1-1
##  2  48.1  11.5     1     2        21.7     99.9 48.078-11.516-1-2  1-2
##  3  48.1  11.5     1     3        21.1     99.9 48.078-11.516-1-3  1-3
##  4  48.1  11.5     1     4        20.5     99.9 48.078-11.516-1-4  1-4
##  5  48.1  11.5     1     5        20.9     99.9 48.078-11.516-1-5  1-5
##  6  48.1  11.5     1     6        21.9     99.9 48.078-11.516-1-6  1-6
##  7  48.1  11.5     1     7        23.5     99.9 48.078-11.516-1-7  1-7
##  8  48.1  11.5     1     8        26.5     98.9 48.078-11.516-1-8  1-8
##  9  48.1  11.5     1     9        29.2     92.0 48.078-11.516-1-9  1-9
## 10  48.1  11.5     1    10        32.0     73.3 48.078-11.516-1-10 1-10
## # ... with 48,112 more rows
```

```
# Für bmp180 und bme280 zu wenige Sensoren vorhanden.
# Beim mergen auf Sensorebene würden daher fast alle Sensoren verloren gehen.
# Deshalb Berechnung des Mittelwerts für diese Sensortypen aggregiert über alle Sensoren
sensors_munich_cleaned$bme280 <- sensors_munich_cleaned$bme280 %>%
  dplyr::rename(pressure1 = pressure)
sensors_munich_cleaned$bmp180 <- sensors_munich_cleaned$bmp180 %>%
  dplyr::rename(pressure2 = pressure)
new_data2 <- sensors_munich_cleaned[c("bme280", "bmp180")] %>%
  purrr::map2(list("pressure1", "pressure2"),
              hourly_mean_agg) %>%
  purrr::map(make_key2)
# Da beide Sensoren Luftdruck messen, wird Mittelwert beider Sensortypen verwendet
# Falls für einen der beiden Sensoren Wert fehlt, nehme anderen Wert
new_data2 <- new_data2$bme280 %>%
  dplyr::full_join(new_data2$bmp180) %>%
  dplyr::mutate(pressure = 0.5*pressure1+0.5*pressure2) %>%
  dplyr::select(-c("pressure1", "pressure2"))
```

```
## Joining, by = c("day", "hour", "key2")
```

```
# Tibbles Mergen
new_data <- new_data1$sds011 %>%
  dplyr::inner_join(new_data1$dht22) %>%
  dplyr::select(-"key1") %>%
  dplyr::left_join(new_data2)
```

```
## Joining, by = c("lat", "lon", "day", "hour", "key1", "key2")
## Joining, by = c("day", "hour", "key2")
```

Transformationen und zeitlich-räumliche Indizierung der Daten

Abschließend kreieren bzw. transformieren wir noch einige Variablen. Da wir für das Modell in der folgenden Sektion einen AR(1)-Prozess annehmen werden, erstellen wir einen chronologischen Zeitindex t. Dieser beginnt mit Wert t = 1 bei Tag 1, Stunde 1, und endet bei Tag 21 Stunde 24 mit Wert t = 504. Wir erstellen zudem für alle Messungen am selben Ort eine ID.

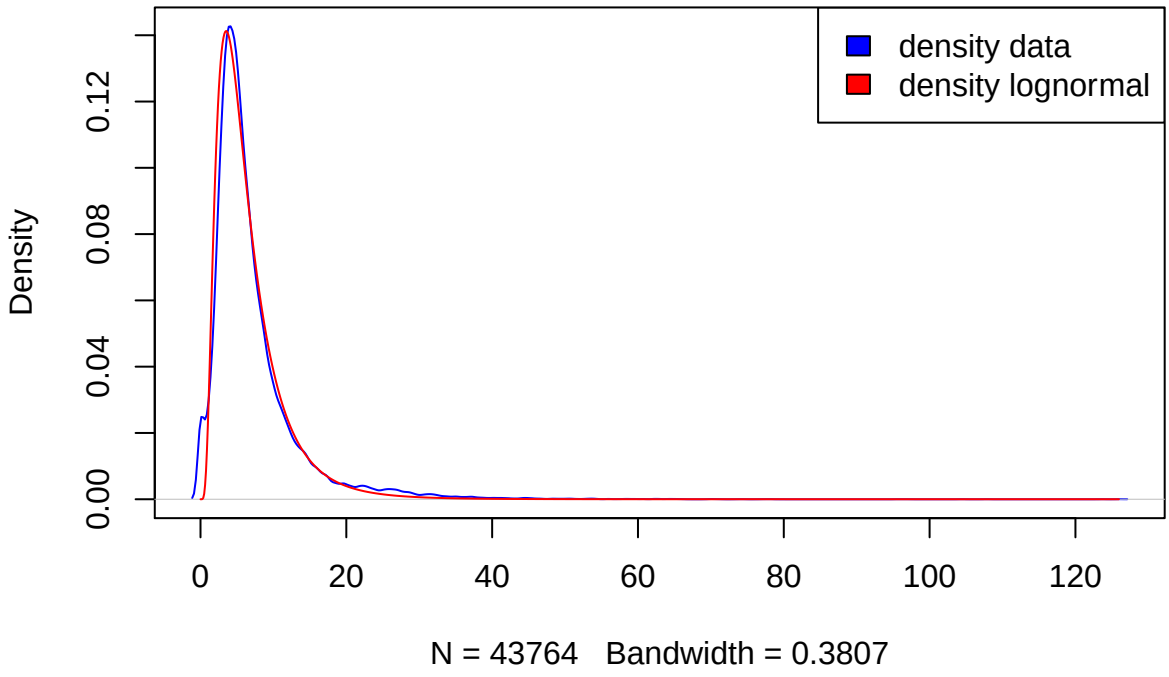
```
# Chronologischen Zeitindex aus Tag und Stunde erstellen,
# da wir später einen AR(1)-Prozess annehmen werden
new_data <- new_data %>% mutate(t = hour+24*(day-1))

# Neue ID kreieren für alle Messungen am gleichen Ort
new_data <- new_data %>% dplyr::group_by(lon,lat) %>%
  mutate(ID = group_indices()) %>% dplyr::ungroup()
```

Die Koordinaten sind als Latitude/Longitude gegeben. Dies ist kein metrisches Koordinatensystem, d.h. insbesondere messen beide Einheiten an verschiedenen Orten der Erde unterschiedliche Distanzen. Um im nächsten Kapitel ein gleichmäßiges Vorhersagegitter zu erzeugen werden daher die Koordinaten in ein metrisches Referenzsystem transformiert. Für München bietet sich hierfür UTM Zone 32 an.

```
# Koordinaten in metrisches Referenzsystem transformieren
coord_metric <- new_data[,c("lon","lat")] %>%
  sf::st_as_sf(coords = c("lon", "lat"), crs = 4326) %>% # lat/lon sf-Objekt erstellen
  sf::st_transform(25832) %>% # CRS von WGS84 zu UTM Zone 32 (Nähe München) transformieren
  sf::st_coordinates() # metrische Koordinaten aus sf-Objekt extrahieren
# Im Datensatz lat/lon Koordinaten durch metrische Koordinaten ersetzen
# Zudem km statt m verwenden, um numerische Probleme beim Fitten zu vermeiden
new_data <- new_data %>% dplyr::mutate(X = coord_metric[,1]/1000,
                                       Y = coord_metric[,2]/1000)
```

Um im nächsten Kapitel eine Normalverteilungsannahme treffen zu können wird die Responsevariable log-transformiert, da die Dichte von pm10 als log-normal angenommen werden kann:

```
pm10 <- new_data$pm10
# Kerndichteschätzung
pm10_density <- density(pm10)
# Plotte geschätzte Dichte vs. Lognormal Dichte
plot(pm10_density, col="blue", main = "Distribution of PM10")
x <- seq(min(pm10),max(pm10), le = 1000)
lines(x, dlnorm(x,meanlog = 1.68, sdlog = 0.65), col = "red")
legend("topright", legend = c("density data", "density lognormal"), fill=c("blue","red"))
```




```
# Log-Transformation der Response
new_data <- new_data %>% mutate(logpm10 = log(pm10+1))
```

Nun haben wir den finalen Datensatz generiert:

```
# Nicht benötigte Variablen entfernen und Datensatz sortieren
(new_data <- new_data %>%
   dplyr::select(c("ID", "t", "logpm10","pm10", "X", "Y", "lon", "lat",
                   "day", "hour", "temperature", "humidity",
                   "pressure")) %>%
   arrange(t,ID))
```

```
## # A tibble: 43,764 x 13
##       ID     t logpm10  pm10     X     Y   lon   lat   day  hour temperature
##    <int> <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>       <dbl>
##  1     1     1    2.25  8.45  678. 5339.  11.4  48.2     1     1        21.3
##  2     2     1    2.43 10.3   680. 5335.  11.4  48.1     1     1        23.2
##  3     3     1    2.67 13.5   682. 5337.  11.4  48.2     1     1        24.5
##  4     4     1    2.74 14.4   683. 5340.  11.5  48.2     1     1        21.5
##  5     5     1    2.44 10.5   684. 5333.  11.5  48.1     1     1        22.7
##  6     6     1    2.55 11.7   685. 5332.  11.5  48.1     1     1        21.3
##  7     7     1    0     0     685. 5333.  11.5  48.1     1     1        21.4
##  8     8     1    2.48 10.9   685. 5334.  11.5  48.1     1     1        20.6
##  9     9     1    2.97 18.4   685. 5335.  11.5  48.1     1     1        21.2
## 10    10     1    2.39  9.94  686. 5334.  11.5  48.1     1     1        24.2
## # ... with 43,754 more rows, and 2 more variables: humidity <dbl>,
## #   pressure <dbl>
```

Die bereinigten Daten werden unter pm_all.csv gespeichert.

```r
# Daten speichern
filename_out <- paste(path_workfiles, "pm_all.csv", sep ="/")
write.table(new_data, file = filename_out, sep = ",", row.names = FALSE)
```

Verkleinerung des Datensatzes & Unterteilung in Trainings- und Validierungsdaten

Aufgrund computationaler Hindernisse verwenden wir für das Fitten des Modells im nächsten Abschnitt nur die erste Woche der Beobachtungen; bei hoher Rechenleistung können jedoch ohne weiteres auch alle Beobachtungen verwendet werden. Somit ist $T = 168$.

```r
# Nur erste Woche verwenden (d.h. t <= 24*7 = 168)
n_t <- 168
new_data_mod <- new_data %>% filter(t <= n_t)
```

Die verbliebenen Beobachtungen unterteilen wir in Trainings- und Validierungsdaten. Zur Validierung des Modells werden zufällig Sensoren ausgewählt - die Vorhersage an diesen Orten wird später mit den tatsächlich gemessenen Werten verglichen.

```r
# Daten in Trainings- und Validierungsdaten unterteilen und nochmal separat abspeichern
idx_val <- sample(1:max(new_data_mod$ID), 10)
pm_data_val <- new_data_mod %>% filter(ID %in% idx_val)
pm_data_train <- new_data_mod %>% filter(!ID %in% idx_val)
filename_val <- paste(path_workfiles, "pm_all_val.csv", sep ="/")
filename_train <- paste(path_workfiles, "pm_all_train.csv", sep ="/")
# Daten zwischenspeichern
write.table(pm_data_val, file = filename_val, sep = ",", row.names = FALSE)
write.table(pm_data_train, file = filename_train, sep = ",", row.names = FALSE)
```

Die Daten werden abgespeichert, damit bei Bedarf verschiedene Modelle anhand der gleichen Validierungsdaten verglichen werden können.