# Analyzing Pollution Data in Madrid

**MBD - Group E**

**17/12/2019**

# 1st step: Hourly Dataset

To create the hourly dataset with all the csv files, we use a for loop.

Before starting the iterations of the for loop, variables used in the loop need to be defined. - path where all the CSV files are stored (csv_path) - common part of the names of the CSV files (csv_name) - first year (y) in the first csv file - first month (m) in the first csv file - empty dataframe that will be filled in the loop - iterator i (not compulsory but clearer this way) - row counter (c_r) of each csv to make sure the final dataframe's number of rows is equal to the sum of the rows in each csv file (start at 0)

```
#clean environment
rm(list=ls())

#define variables
csv_path<-'/Users/anna/Desktop/IE BUSINESS SCHOOL/Data Science/R/GroupWork/workgroup_
data/' #change
csv_name<-'hourly_data_'  #only common part of the csv files' names
n_csv<-72   #total number of csv files to be opened
y<-11   #first year in csv files
m<-1    #first month in csv files
bind_df<-data.frame()   #new hourly dataframe that will be filled by row binding each
dataframe
i<-1  #iterator's first value (not compulsory)
c_r<-0  #row counter to check how many rows final dataframe should have
```

In each iteration of the loop, a csv file will be read and put into a dataframe.

Once the dataframe is filled, two new columns are added: - one to enter the corresponding year of the csv file - one to input the corresponding month of the csv file

We then reorder the dataframe's columns to put the year and month columns first.

We also remove any rows containing NAs. This is because when the csv file is read and put in the dataframe some empty rows are added. They do not contain any information (double checked before) so we can remove them.

Once the csv dataframe is all ready, it is row binded with the final hourly dataframe (bind_df) - which for the first iteration of the for loop is empty.

The dataframes of each csv file are still kept and named in case it is needed later on.

We then added an IF loop to check which value the month variable (m) and the year variable (y) has to take for the next for iteration. Basically if the month value is smaller than 12 then month value for the next iteration will be m+1 and the year value will remain the same. However, if the month value is equal to 12, for the next iteration the year value will change to y+1 and the month value will restart to 1.

```
for ( i in 1:n_csv ){

  #put csv file in dataframe df
  df<-read.csv(paste0(csv_path,csv_name,y,'_',m,'.csv'),header=TRUE, sep=",")

  #add year and month column inside dataframe df
  df$month<-m
  df$year<-2000+y

  #reorder columns of dataframe df
  df<-df[colnames(df)[c(7,6,1:5)]]
  ########rownames(df1)<-paste0(df1$year,'/',df1$month,'/',df1$day,'-',df1$station)

  #count rows inside df dataframe
  c_r<-c_r+nrow(df)

  #remove NA rows (double-checked -> NA rows have NAs in all columns)
  df<-df[complete.cases(df),]

  #bind df with the new hourly dataframe bind_df
  bind_df<-rbind(bind_df,df)

  #give a name to save df on its own too
  assign(paste0('df_',i),df)

  #check which value m and y have to take for the next iteration
  if (m<12) {
    m<-m+1
  } else if (m==12) {
    m<-1
    y<-y+1
  }

#close for loop
}

#check structure dataframe
str(bind_df)
```

```
## 'data.frame':    6470763 obs. of  7 variables:
##  $ year     : num  2011 2011 2011 2011 2011 ...
##  $ month    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ day      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ hour     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ station  : int  28079004 28079008 28079017 28079018 28079024 28079035 28079036
28079038 28079040 28079057 ...
##  $ parameter: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ value    : num  6 12 12 10 7 11 22 13 9 12 ...
```

```
#bind_df shoud have c_r obs. - NA's and 7 variables
```

# 2nd step: Daily Dataset

First, we create a new column in the bind_df dataframe which will contain the corresponding date string by concatenating Year, Month, and Day columns. The column containing the date is converted to a date variable.

Afterwards, we need to install (if not already installed) the Reshape package that we will use to cast the data.

The Cast function is used to put each parameter as a new column that will be filled with the average value of the parameter for each day (average of the hourly parameter's values) for each corresponding parameter.

The new dataframe created with the cast function is called daily_df. We sorte it by dates and add a year column which will be used to ease one of the visualizations. We also keep only the information needed for the analysis: NO2, SO2, SO3, PM2.5, date, and year.

Each column names for the parameters is changed to its scientific nomenclature for better comprehension. This is done using a for loop looping through each column name. And an if loop, changing the names of the parameters columns: NO2, SO2, O3, PM2.5.

```
#create date column concatenating year-month-day for each row and convert new string
 to date
bind_df$date<-paste0(bind_df$year,'-',bind_df$month,'-',bind_df$day)
bind_df$date<-as.Date(bind_df$date)

#install reshape package if not already installed - this is used to cast data
if(!'reshape'%in%installed.packages()){
  install.packages('reshape')
} else {
  print('reshape is already installed')
}
```

```
## [1] "reshape is already installed"
```

```
library(reshape)

#put each parameter as a columns that will be filled with the average of each hour fo
r corresponding parameter
daily_df<-cast(bind_df,date~parameter,mean)
daily_df$year<-format(as.Date(daily_df$date, format="%d/%m/%Y"),"%Y")

#order dataframe / keep only wanted columns
daily_df<-daily_df[order(daily_df$date),c('date','year','1','8','9','14')]

#change the name of each column with parameter nomenclature
for (i in 2:ncol(daily_df)){
  if (colnames(daily_df)[i]==1){
    colnames(daily_df)[i]<-'SO2'
  }else if (colnames(daily_df)[i]==8){
    colnames(daily_df)[i]<-'NO2'
  }else if (colnames(daily_df)[i]==9){
    colnames(daily_df)[i]<-'PM2.5'
  }else if (colnames(daily_df)[i]==14){
    colnames(daily_df)[i]<-'O3'
  }
}


#check structure new dataframe
str(daily_df)
```

```
## List of 6
##  $ date : Date[1:2192], format: "2011-01-01" "2011-01-02" ...
##  $ year : chr [1:2192] "2011" "2011" "2011" "2011" ...
##  $ SO2  : num [1:2192] 10.71 11.93 11.91 8.84 9.51 ...
##  $ NO2  : num [1:2192] 41.5 48.5 63.6 46.3 51.5 ...
##  $ PM2.5: num [1:2192] 9.36 9.08 11.94 9.4 10.51 ...
##  $ O3   : num [1:2192] 20.47 15.56 9.45 13.34 10.88 ...
##  - attr(*, "row.names")= int [1:2192] 1 2 3 4 5 6 7 8 9 10 ...
```

# 3rd step: Daily Dataset with Parameters & Weather information

First, we need to read the weather excel and put it into a dataframe. To do so, the “readxl” package needs to be installed (if not already installed).

We remove columns humidity, temp_max, temp_min, which we are not using for our visualisation.

Once the weather data are in the dataframe, the date column (which appears as a POSIXct) is converted to a date variable to ensure it matches with our daily dataframe’s date information.

The daily dataframe and weather dataframe are then merged thanks to the date column of each dataframe.

We call this merged dataframe complete_df.

We display complete_df’s structure to double check our data. We also check there are no NA’s.

```
#install reshape package if not already installed
if(!'readxl'%in%installed.packages()){
  install.packages('readxl')
} else {
  print('readxl is already installed')
}
```

```
## [1] "readxl is already installed"
```

```
library('readxl')


#add weather variable by merging
#read file weather
weather_excel<-'/Users/anna/Desktop/IE BUSINESS SCHOOL/Data Science/R/GroupWork/weath
er.xlsx'
weather_df<-data.frame()
weather_df<-read_excel(weather_excel)

#remove unwanted columns
weather_df$humidity<-NULL
weather_df$temp_max<-NULL
weather_df$temp_min<-NULL

#check variable types
str(weather_df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2192 obs. of  4 variables:
##  $ date          : POSIXct, format: "2011-01-01" "2011-01-02" ...
##  $ temp_avg      : num  8.3 8.6 4.2 6.5 8.9 12.2 10.9 9.8 8.4 6.9 ...
##  $ precipitation : num  0 0 0 0 0 ...
##  $ wind_avg_speed: num  5.2 5.4 3.5 6.3 10.4 15.7 15.6 14.3 6.5 7.4 ...
```

```
#dates in date format in weather_df and merge with daily_df
weather_df$date<-as.Date(weather_df$date)
df_complete<-merge(x=daily_df,y=weather_df,all.x=TRUE, by='date')

#check structure and check NA's
str(df_complete)
```

```
## 'data.frame':    2192 obs. of  9 variables:
##  $ date          : Date, format: "2011-01-01" "2011-01-02" ...
##  $ year          : chr  "2011" "2011" "2011" "2011" ...
##  $ SO2           : num  10.71 11.93 11.91 8.84 9.51 ...
##  $ NO2           : num  41.5 48.5 63.6 46.3 51.5 ...
##  $ PM2.5         : num  9.36 9.08 11.94 9.4 10.51 ...
##  $ O3            : num  20.47 15.56 9.45 13.34 10.88 ...
##  $ temp_avg      : num  8.3 8.6 4.2 6.5 8.9 12.2 10.9 9.8 8.4 6.9 ...
##  $ precipitation : num  0 0 0 0 0 ...
##  $ wind_avg_speed: num  5.2 5.4 3.5 6.3 10.4 15.7 15.6 14.3 6.5 7.4 ...
```

```
sum(is.na(df_complete))
```

```
## [1] 0
```

# 4th step: Exploratory Visualisation

The first step of our visualisation is to check the behaviour of each parameter (NO2, SO2, PM2.5, O3) throughout time.

Different packages need to be installed, if not already installed, and libraries need to be called.

```
#all packages intalled for visualisation ----
#install reshape package if not already installed
if(!'ggpubr'%in%installed.packages()){
  install.packages('ggpubr')
} else {
  print('ggpubr is already installed')
}
```

```
## [1] "ggpubr is already installed"
```

```
#install ggplot2 package if not already installed
if(!'ggplot2'%in%installed.packages()){
  install.packages('ggplot2')
} else {
  print('ggplot2 is already installed')
}
```

```
## [1] "ggplot2 is already installed"
```

```
#install plotly package if not already installed
if(!'plotly'%in%installed.packages()){
  install.packages('plotly')
} else {
  print('plotly is already installed')
}
```

```
## [1] "plotly is already installed"
```

```
#install ggcorrplot package if not already installed
if(!'ggcorrplot'%in%installed.packages()){
  install.packages('ggcorrplot')
} else {
  print('ggcorrplot is already installed')
}
```

```
## [1] "ggcorrplot is already installed"
```

```
#install ggally package if not already installed
if(!'GGally'%in%installed.packages()){
  install.packages('GGally')
} else {
  print('GGally is already installed')
}
```

```
## [1] "GGally is already installed"
```

```
#Install ggthemes package if not already installed
if(!'ggthemes'%in%installed.packages()){
  install.packages('ggthemes')
} else {
  print('ggthemes is already installed')
}
```

```
## [1] "ggthemes is already installed"
```

```
library(ggpubr)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: magrittr
```

```
library(ggplot2)
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:reshape':
##
##     rename
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

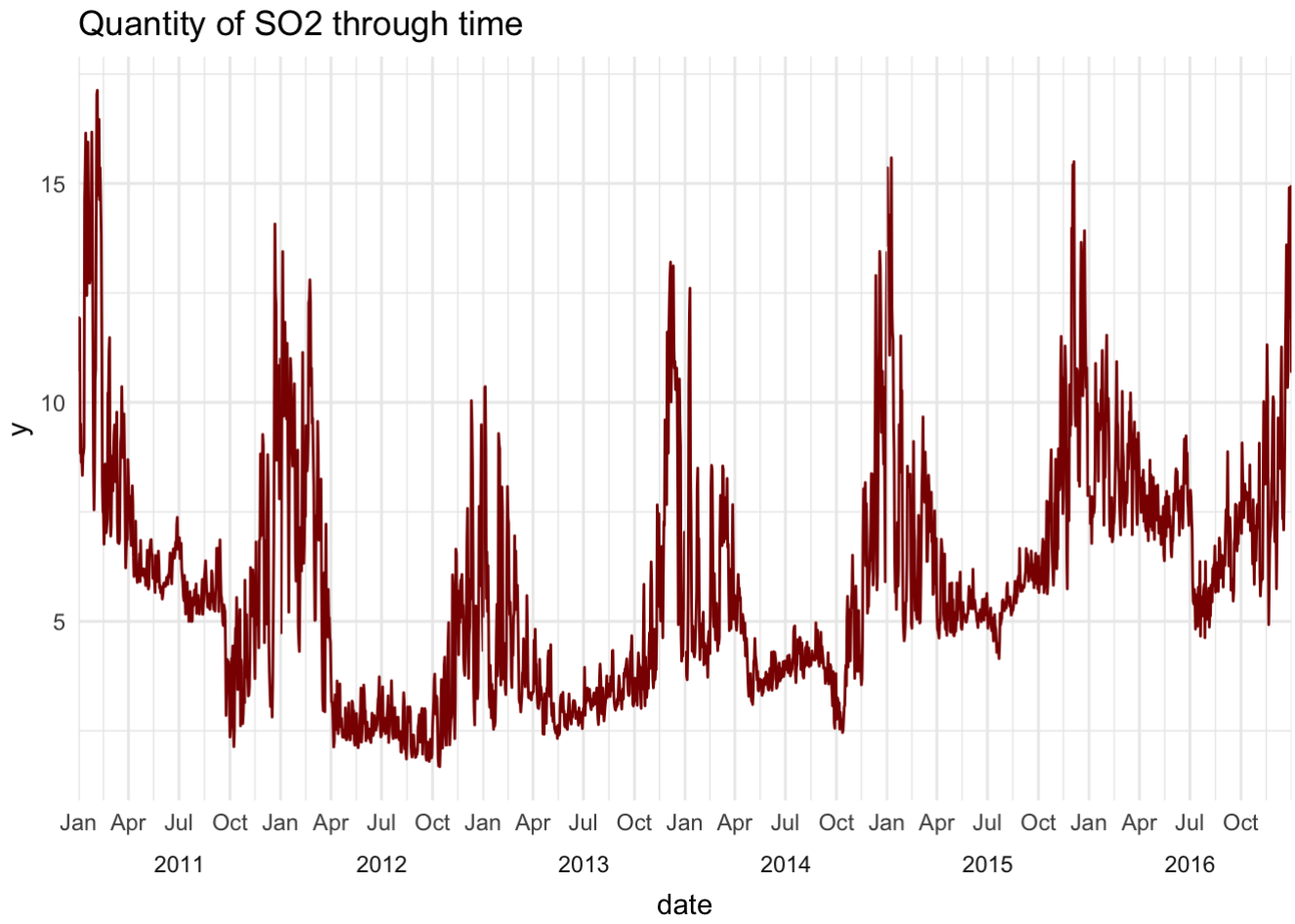
```
library(ggthemes)
```

Once packages are installed and libraries are called, we create a function to automate the plotting of different variables' timeseries.

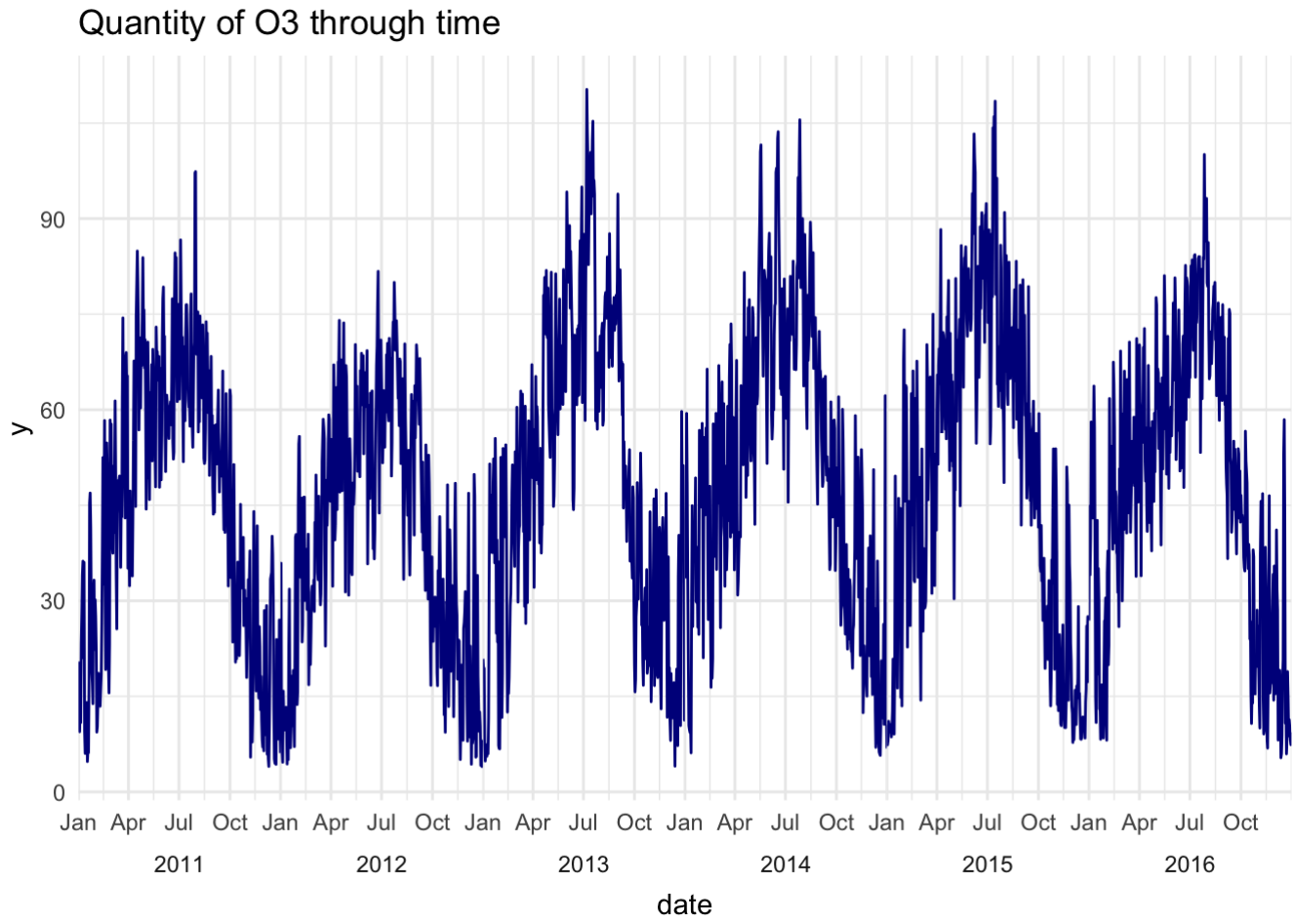
```
timeseries_fun = function(y,nom, c) {
  ggplot(df_complete, aes(date)) +
    geom_line(aes(y = y, colour = nom), color = c) +
    facet_grid(.~year, scale="free", space="free", switch = "x") +
    scale_x_date(date_breaks = "3 month", date_labels = "%b", expand = c(0,0)) +
    theme_minimal() +
    theme(panel.spacing.x = unit(0,"line"), strip.placement = "outside", legend.posit
ion = "none") +
    ggtitle(paste0(nom, ' through time'))
}
```

The function is called for the different parameters: O2, O3, PM2.5, NO2. The plots are not presented on the same grid since it is less readable for comments.

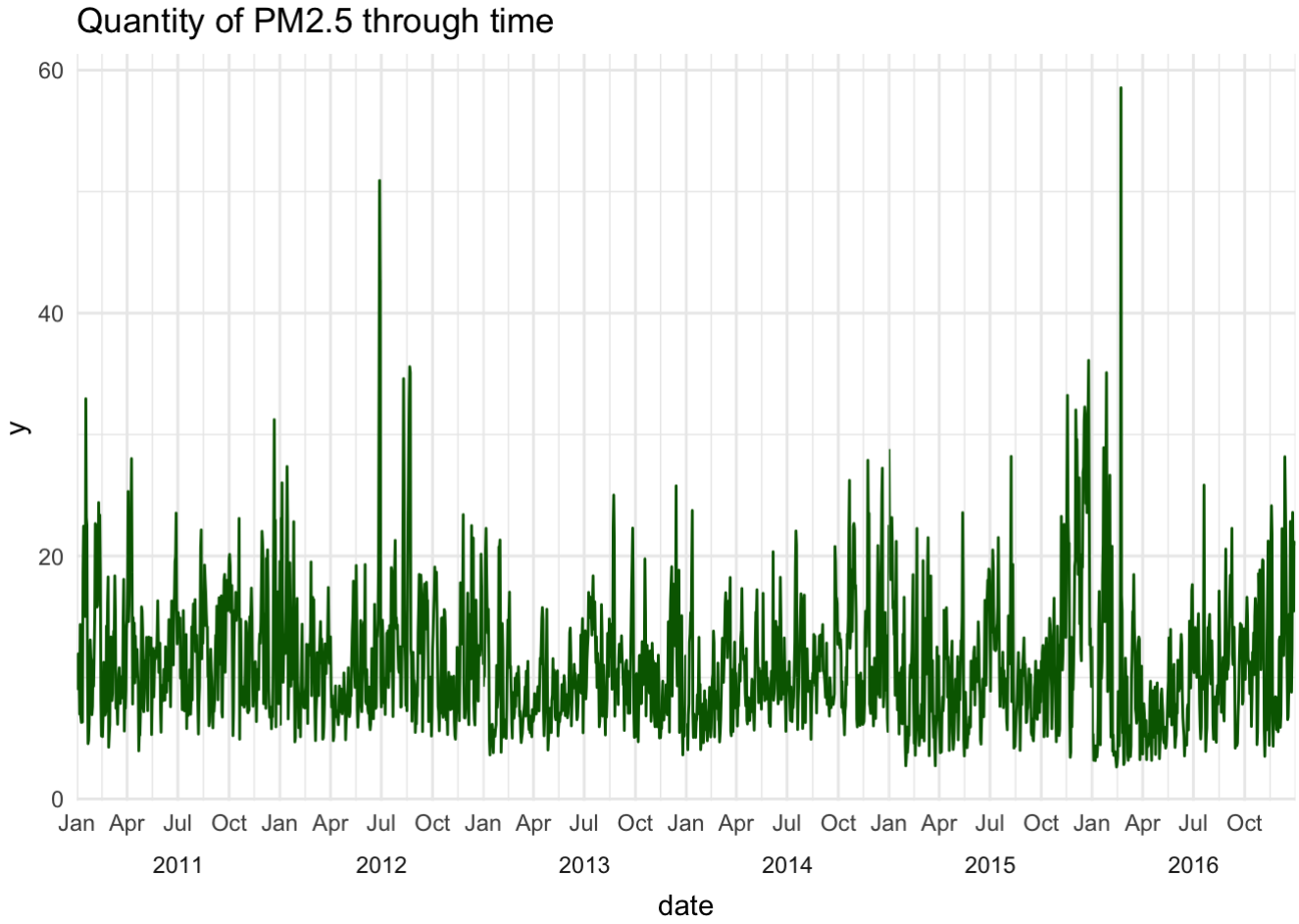
```
timeseries_fun(df_complete$SO2, 'Quantity of SO2', 'darkred')
```



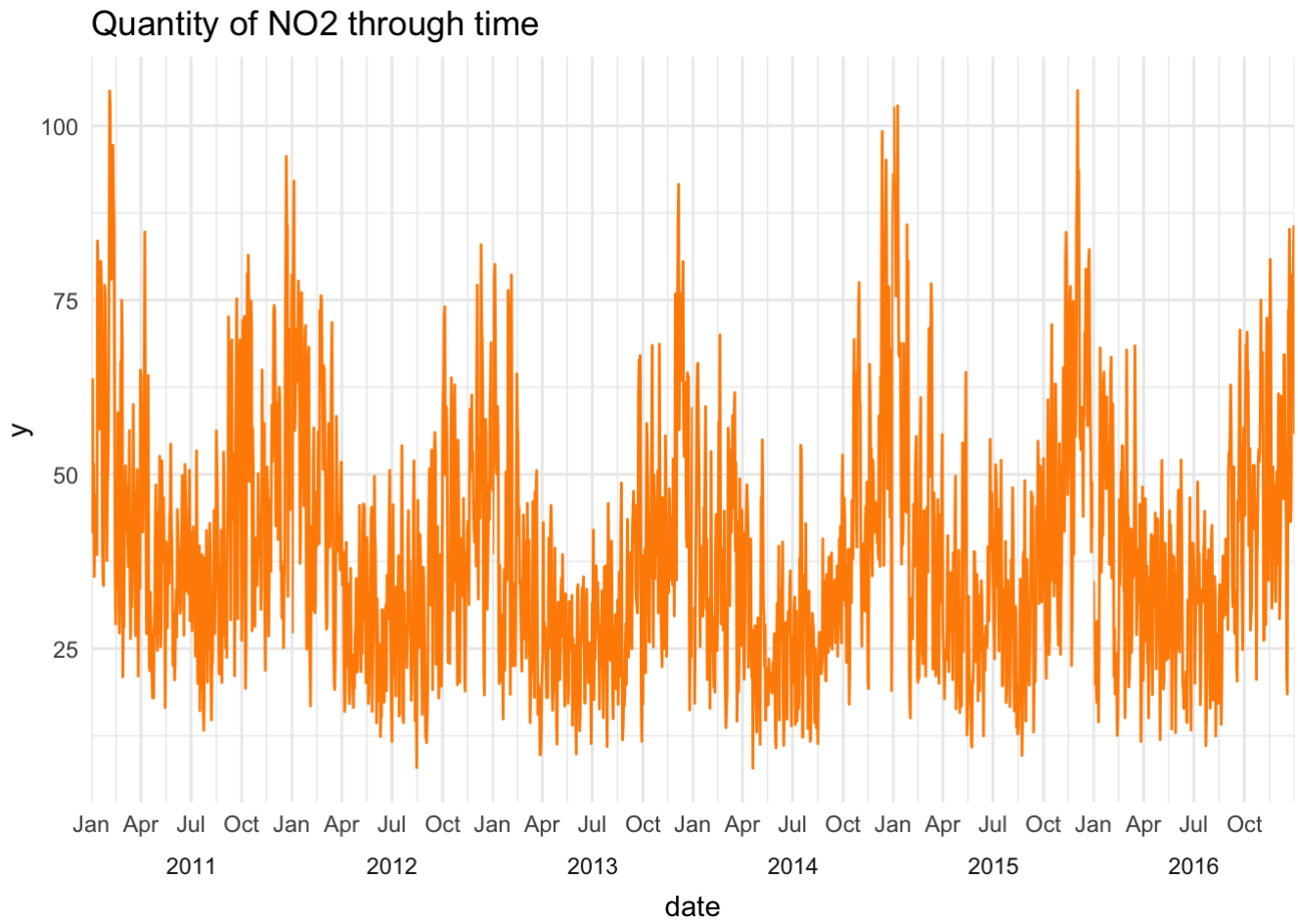

```
timeseries_fun(df_complete$O3, 'Quantity of O3', 'darkblue')
```



```
timeseries_fun(df_complete$PM2.5, 'Quantity of PM2.5', 'darkgreen')
```




```
timeseries_fun(df_complete$NO2, 'Quantity of NO2','darkorange')
```




Looking at the time series of each parameter, we can see the quantity of NO2, SO2, O3, and PM2.5 in the air of Madrid varies a lot throughout a year.

For O3, quantities vary from 10 to a 100 ug/m3. Reaching a low around winter (january) and reaching a peak around summer (peak during July month). 2012 seems to have been a year with a lower peak (80 againt usual 100 ug/m3).

For NO2, quantities vary from 10 to 110 ug/m3 on the same year. The quantity was lower from 2012 to 2014 than in 2011, and went back up in 2015. At a yearly scale, the quantity of SO2 in the air is higher during winter (peak during January month usually around 100).

For SO2, the quantity swings from 2.5 ug/m3 to around 15 ug/m3 throughout a year. SO2 peaks that seem to usually happen during the January month, have been lower from 2012 to 2014 (10 ug/m3 in 2013 against 18 ug/m3 in 2011). Since 2015, the trend seems to have gone back up.

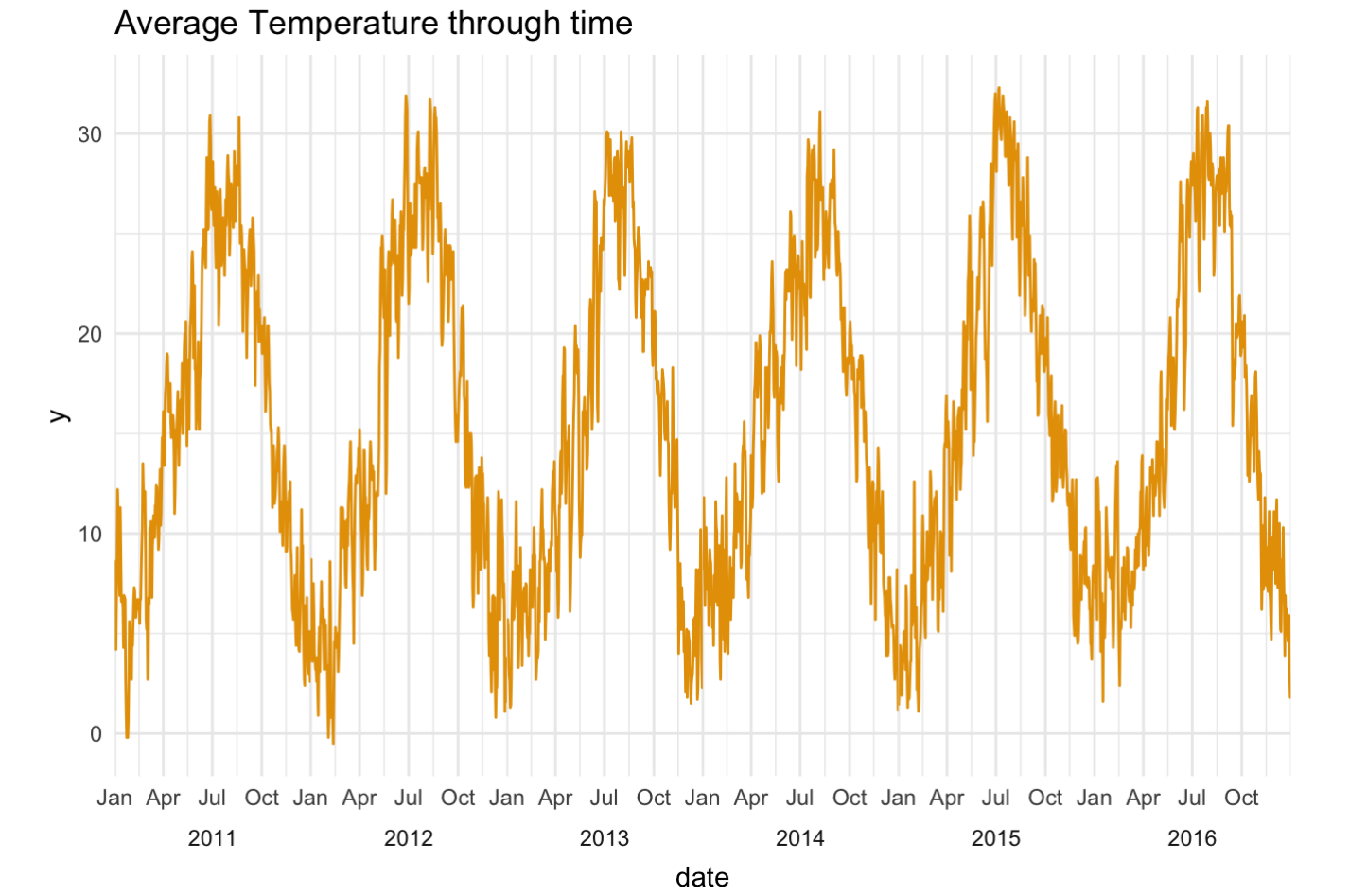
For PM2.5, the quantity varies a lot throughout the year between 20 and 35 ug/m3. Two higher peaks are noticeable for July 2012 where it reaches 53 ug/m3, and March 2016 where it reaches almost 60 ug/m3.

Overall, the difference in pollutant quantities remains the similar between 2011 and 2016 at a yearly level (eventhough for some years we observe lower quantities), but varies throughout the year in a cyclical manner.

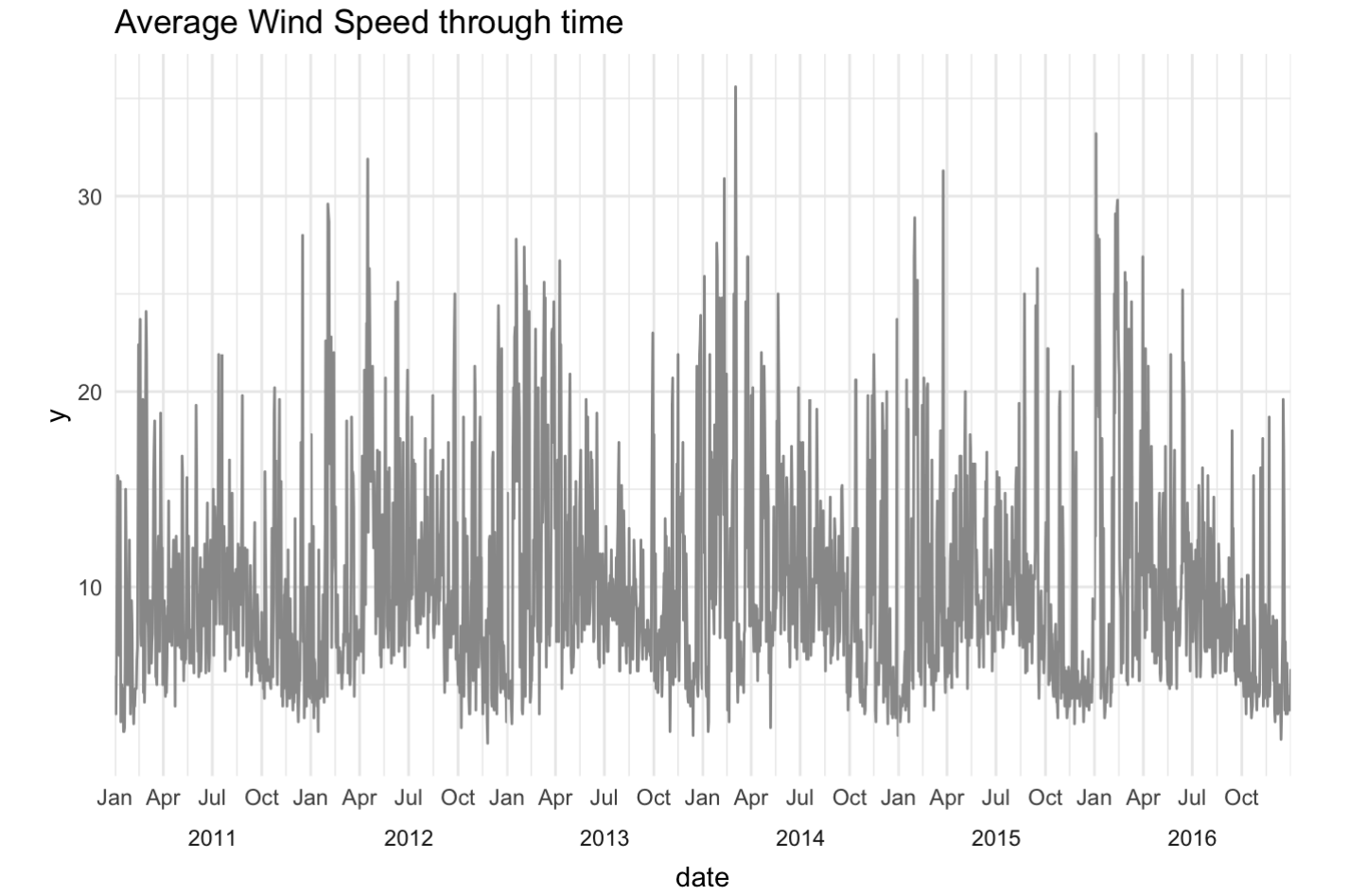
Later, it will be interesting to understand what variables influence the variation in NO2 in the air of madrid.

Before conducting any linear regression to understand the behaviour of NO2 quantity in the air, we also had a look at the behaviour of the temperature average, wind speed average, and precipitation throughout time.

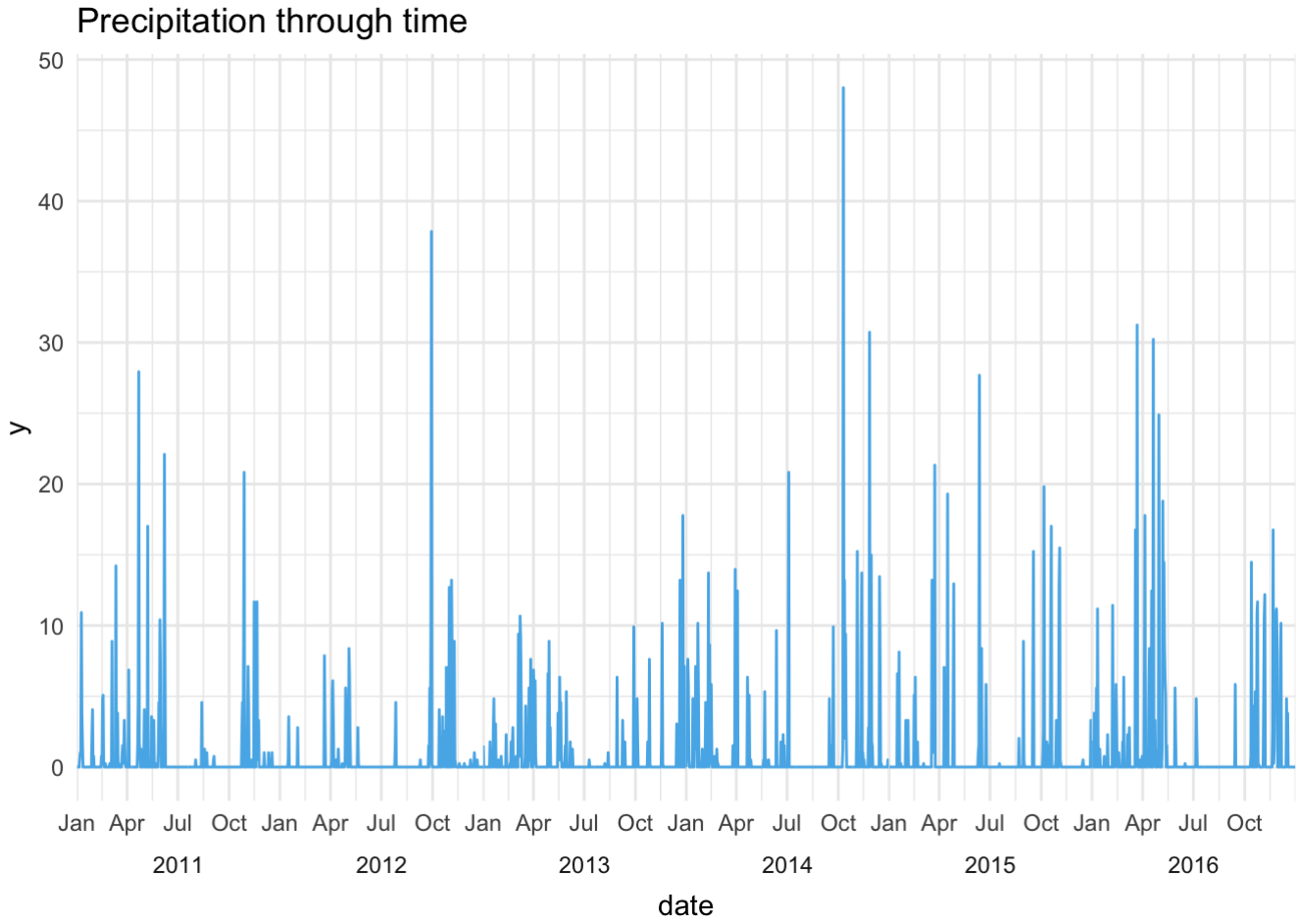
```
timeseries_fun(df_complete$temp_avg, 'Average Temperature', "#E69F00")
```




```
timeseries_fun(df_complete$wind_avg_speed,'Average Wind Speed', "#999999")
```



```
timeseries_fun(df_complete$precipitation, 'Precipitation', "#56B4E9")
```




Average temperature, average wind speed and precipitation also have a clear cycical behaviour throughout a year.

Average temperature, is usually lower in winter, reaching a low of approximately 0 degrees. And higher in summer, reaching around 30 degrees.

Wind speed variation are much closer to each other, with usual peaks between february and march.

Precipitations also seem to have a cyclical behaviour which appear much less symmetric year on year. From 2011 to 2013, around july is a dry period. Months with the highest amount of precipitation is usually around october. In 2014, the october peak has reached a record with almost 50 mm against usual 30 mm.

All in all, we can also say that the variations in average temperatures, average wind speed, and precipitations are important within a year but do not alter significantly year on year, with some exceptions.

To have a deeper understanding of the behaviour of NO2. We created two interactive timeseries.

The first interactive time-series is for nitrogen dioxyde (NO2) with colors for different temperature scales.

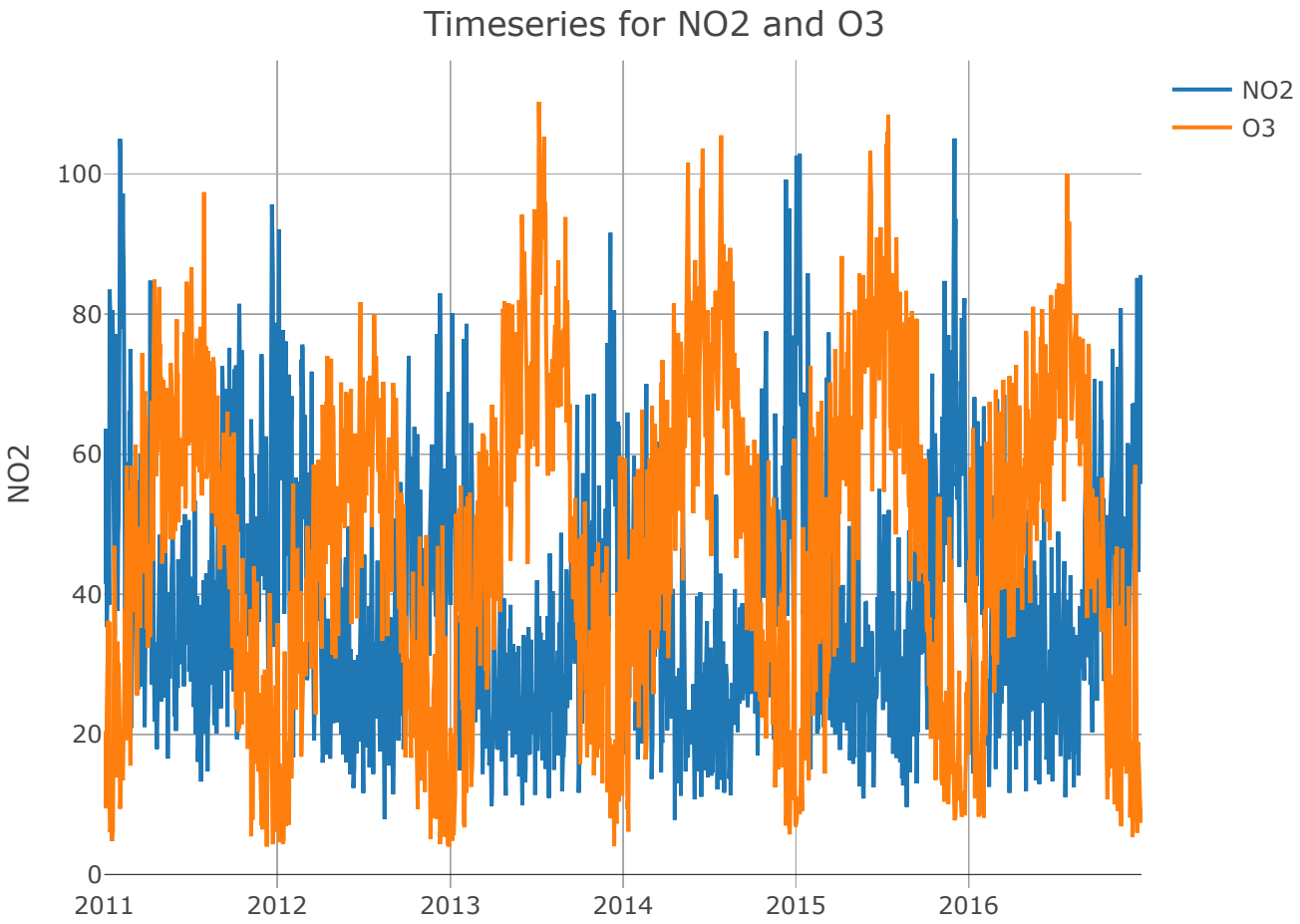
It is possible to click and zoom on the graph. Also by moving the mouse along the graph, NO2 level, temperature and date will be displayed for the corresponding coordinates.

```
time_series <-ggplot(df_complete, aes(x=date,y=NO2, color=temp_avg)) +
  geom_point()  +
  geom_abline() +
  ggtitle("Timeseries for NO2 coloured by temperature")
```

Overall, this plots confirms the statement made earlier when commenting the timerseries of NO2: Quantities of NO2 in the air are higher for colder month (aka. lower temperatures).

The second interactive timeseries is focused on quantities NO2 and O3. As discussed above when reading the independent timeseries, quantities in NO2 seem to reach peaks on cold month while quantities in O3 reach peaks on hotter months.

```
ts_o3_no2 <- plot_ly(df_complete, x= df_complete$date, y= ~NO2, name= 'NO2', type= 's
catter', mode='lines') %>%
  add_trace(y= ~O3, name= 'O3', mode='lines') %>% layout (title= "Timeseries for NO2
 and O3")
ts_o3_no2
```




This time series for nitrogen dioxide (NO2) and ozone (O3) indeed illustrates a potential inverse relationship between the two variables.

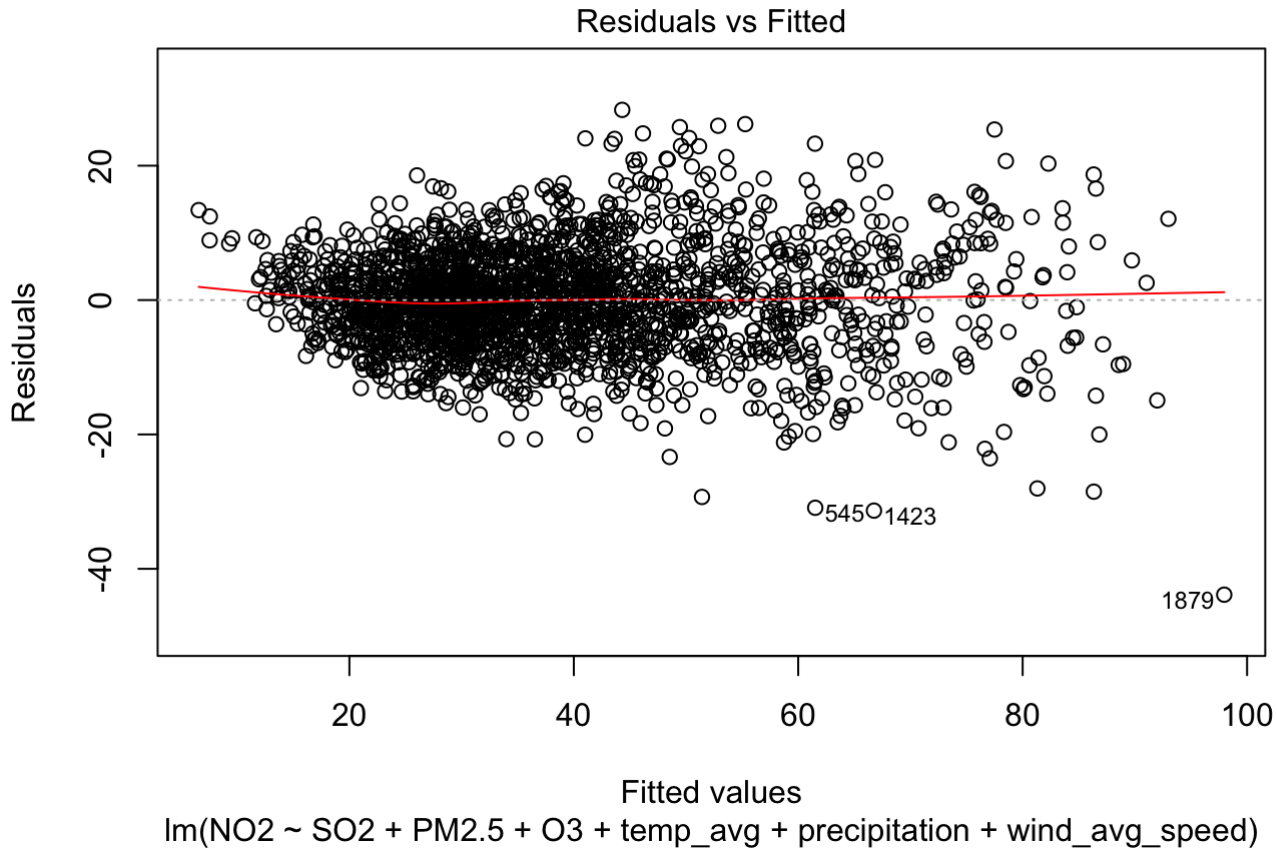
# 5th Step: Linear Regressions on NO2

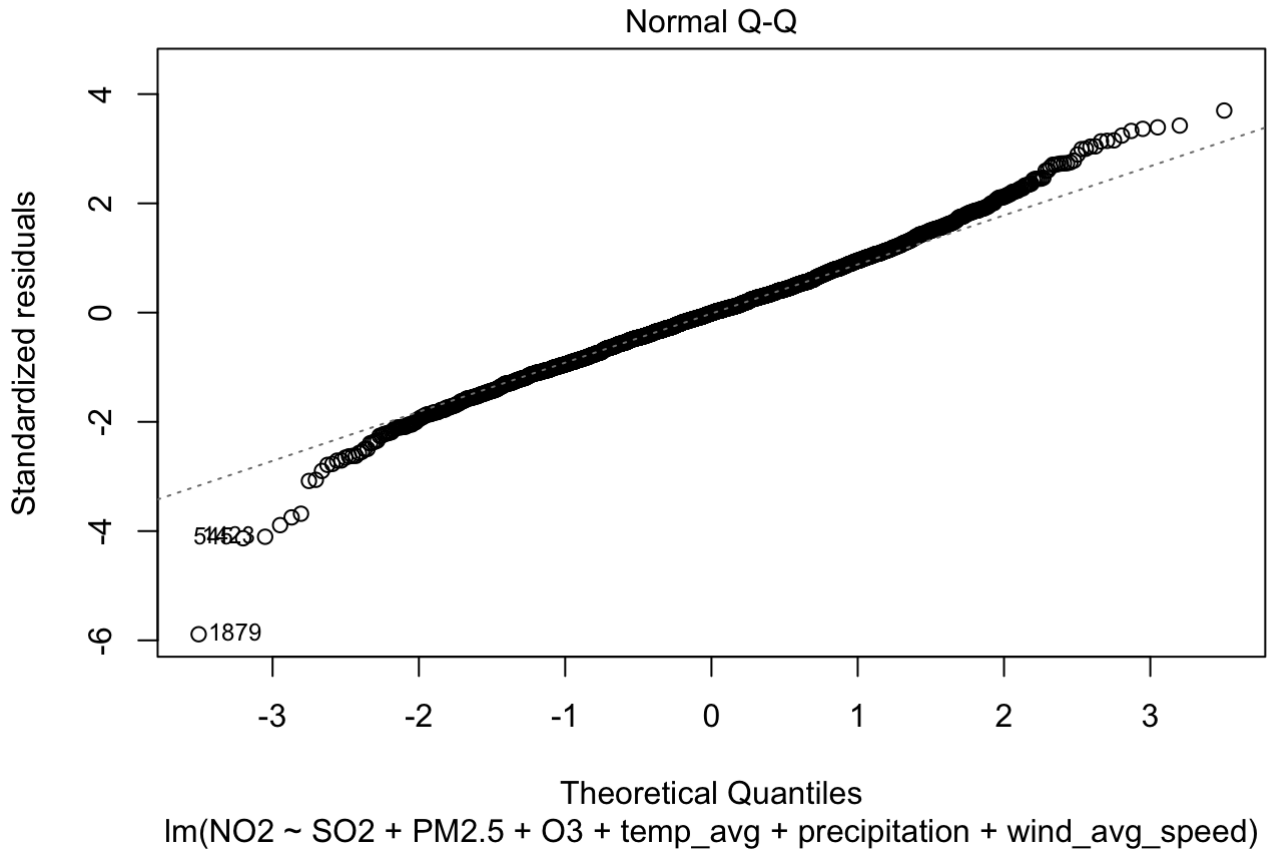
A first regression model is run with all the parameters included, to get an overview of the impact of different parameters on nitrogen dioxide. Later we will adjust the model.

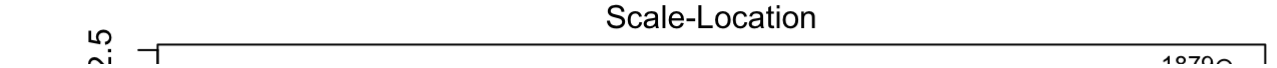
```
regression <- lm(NO2~SO2+PM2.5+O3+temp_avg+precipitation+wind_avg_speed, data = df_co
mplete)
summary(regression)
```

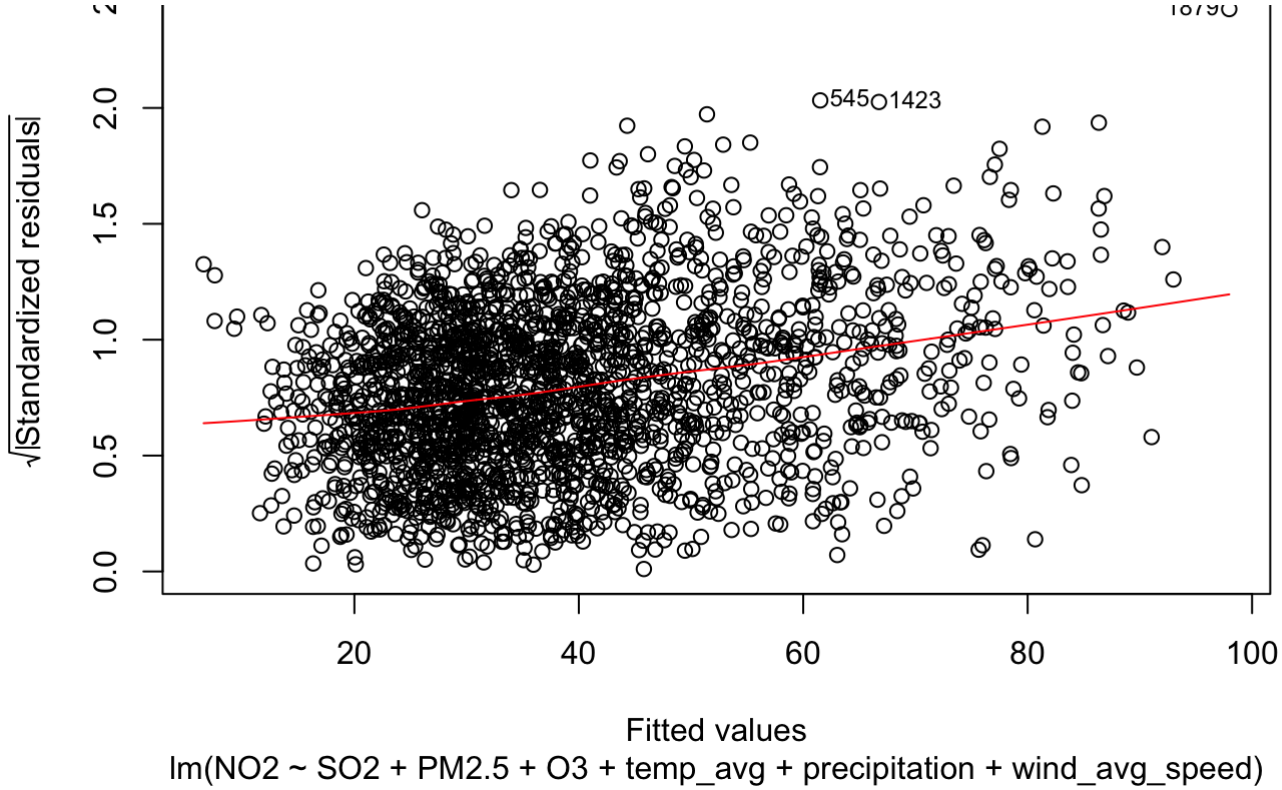
```
##
## Call:
## lm(formula = NO2 ~ SO2 + PM2.5 + O3 + temp_avg + precipitation +
##     wind_avg_speed, data = df_complete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -43.850  -4.771  -0.075   4.502  28.309
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    37.62630    0.99447  37.836  < 2e-16 ***
## SO2             1.93820    0.08041  24.104  < 2e-16 ***
## PM2.5           0.95238    0.03916  24.318  < 2e-16 ***
## O3             -0.29971    0.01165 -25.733  < 2e-16 ***
## temp_avg        0.03797    0.03292   1.154  0.24881
## precipitation  -0.14403    0.05207  -2.766  0.00572 **
## wind_avg_speed -0.68046    0.03731 -18.240  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.66 on 2185 degrees of freedom
## Multiple R-squared:  0.8009, Adjusted R-squared:  0.8004
## F-statistic:  1465 on 6 and 2185 DF,  p-value: < 2.2e-16
```

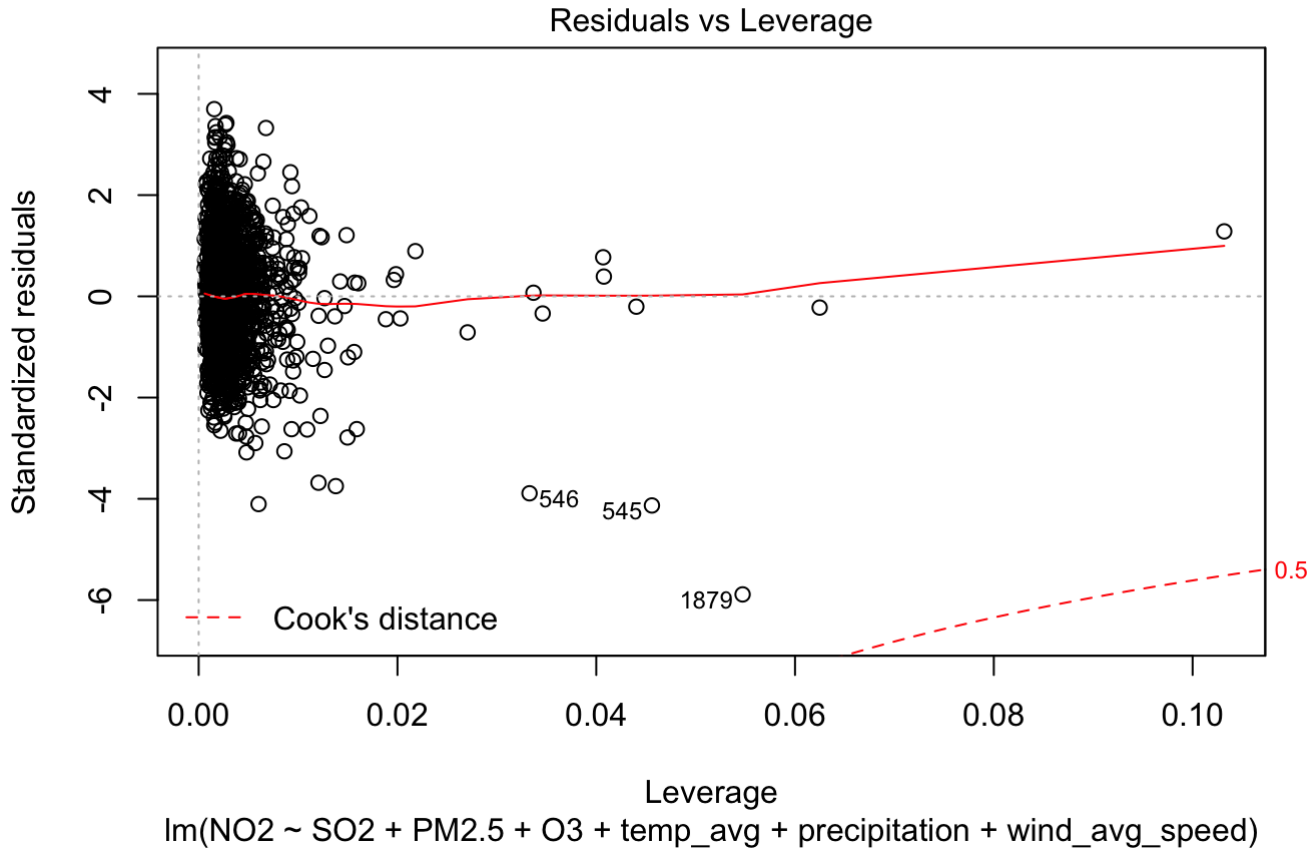
```
plot(regression)
```


Residuals vs Fitted
Residuals
20
0
-20
-40
545
1423
1879
20
40
60
80
100
Fitted values
lm(NO2 ~ SO2 + PM2.5 + O3 + temp_avg + precipitation + wind_avg_speed)


Normal Q-Q
Standardized residuals
4
2
0
-2
-4
-6
1879
-3
-2
-1
0
1
2
3
Theoretical Quantiles
lm(NO2 ~ SO2 + PM2.5 + O3 + temp_avg + precipitation + wind_avg_speed)


Scale-Location




In order to improve the model, we asked ourselves the following questions: - How much of the variance can our model explain? - Is there any unnecessary variables in our initial model? - Are the parameters significant? - Are some explanatory variables correlated in our initial model?

The R square = 0.8 means that around 80% of the variance of nitrogen dioxide is explained by the model. This is a good score, however, two parameters appear to be statistically less significant than others: precipitation and average temperature.

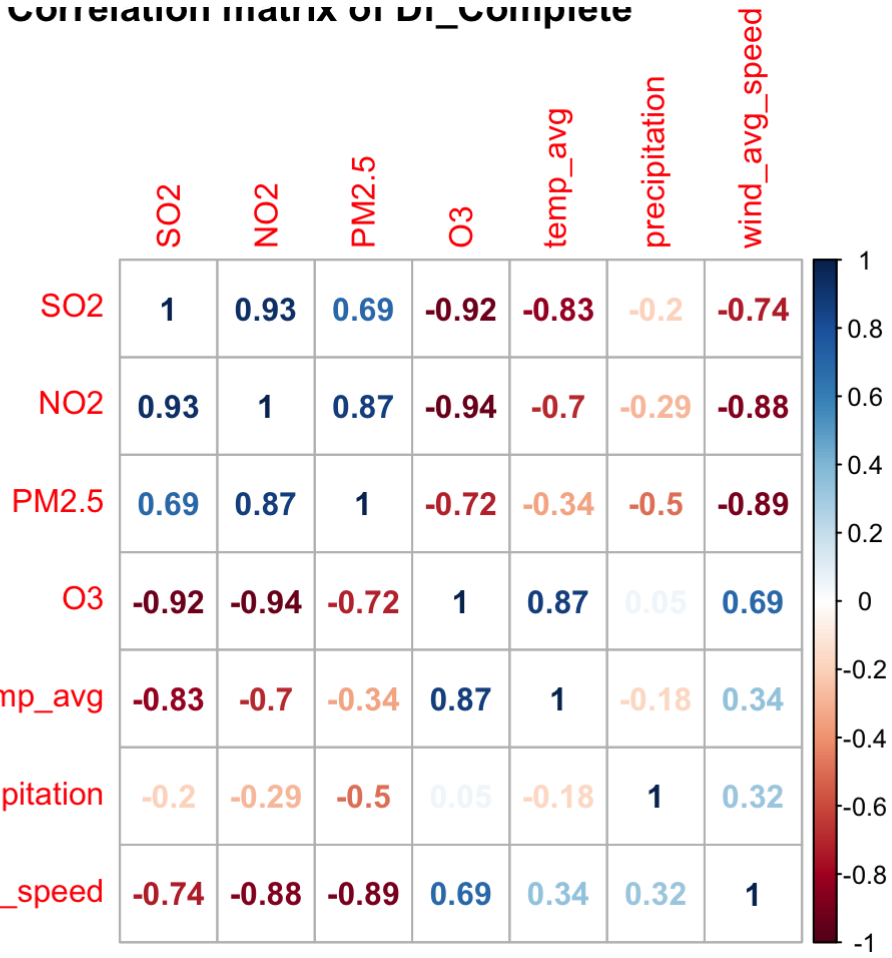
Average temperature has a p-value of 0.24881, which is not significant at all.

The correlation between explanatory variables might cause one of them to be “non-significant” in the model. To understand whether the lack of significance in average temperature is due to multicolinearity, we have a quick look at the correlation matrix.

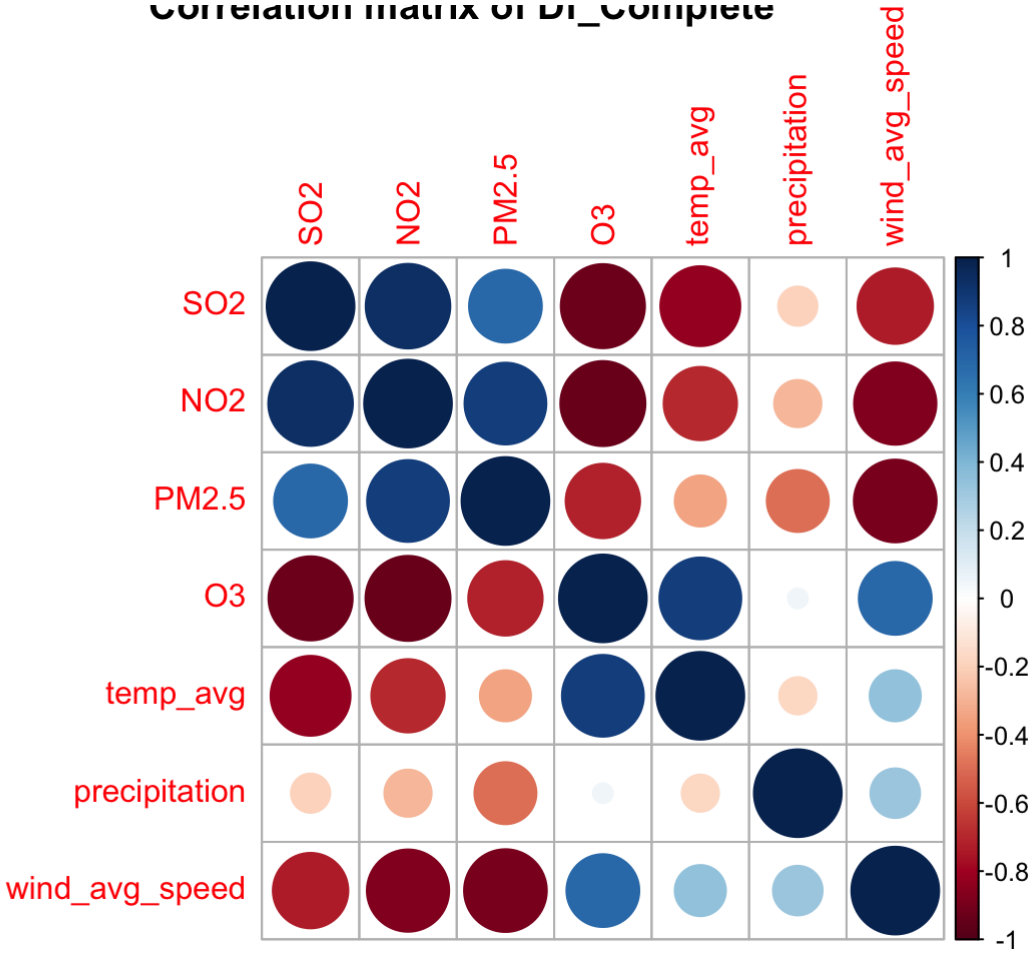
```
#Create correlation matrix for all parameters, excluding NAs, rounding at 2 decimals
a <- cor(df_complete[, sapply(df_complete, is.numeric)], use = 'complete.obs')
cormat <- round(cor(a),2)
cormat
```

```
##                  SO2   NO2 PM2.5    O3 temp_avg precipitation wind_avg_speed
## SO2             1.00  0.93  0.69 -0.92    -0.83         -0.20          -0.74
## NO2             0.93  1.00  0.87 -0.94    -0.70         -0.29          -0.88
## PM2.5           0.69  0.87  1.00 -0.72    -0.34         -0.50          -0.89
## O3             -0.92 -0.94 -0.72  1.00     0.87          0.05           0.69
## temp_avg       -0.83 -0.70 -0.34  0.87     1.00         -0.18           0.34
## precipitation  -0.20 -0.29 -0.50  0.05    -0.18          1.00           0.32
## wind_avg_speed -0.74 -0.88 -0.89  0.69     0.34          0.32           1.00
```

```
#Correlation plot
corrplot(cormat, method ='number', title = 'Correlation matrix of Df_Complete')
```



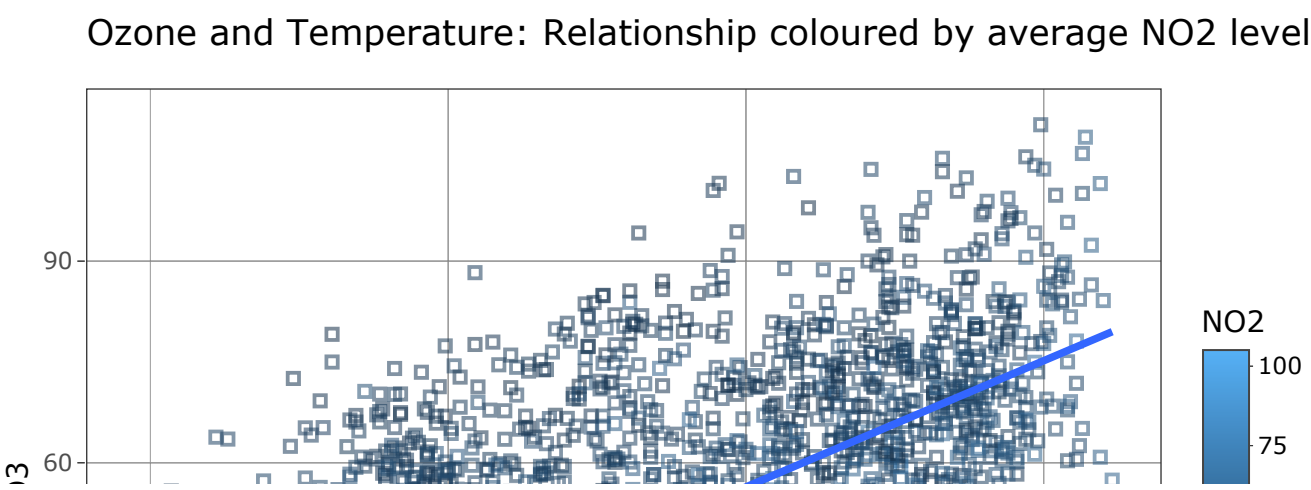

```
corrplot(cormat, title = 'Correlation matrix of Df_Complete')
```

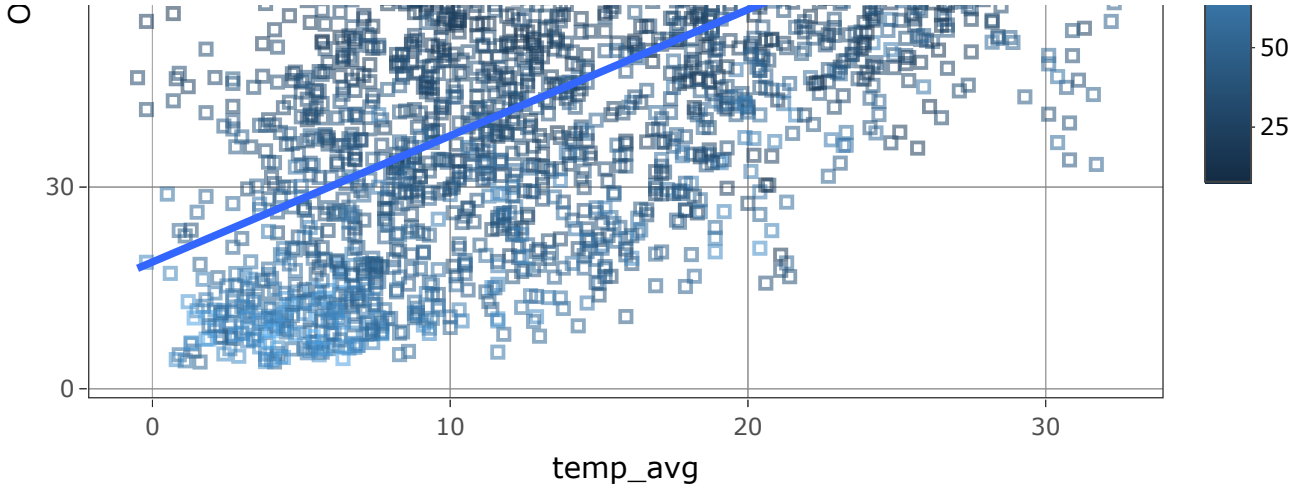



The matrix shows a very weak relation between precipitation and NO2. The lack of correlation between the explained and an explanatory variable might not add value to our model. SOURCE (https://news.harvard.edu/gazette/story/2016/04/the-complex-relationship-between-heat-and-ozone/ (https://news.harvard.edu/gazette/story/2016/04/the-complex-relationship-between-heat-and-ozone/))

The correlation matrix also shows a possible relationship between O3 and temperature: correlation coefficient = O.87. We also know, from existing environmental air pollution research, that temperature and Ozone (O3) are related. We plotted the relationship between ozone and temperature below:

```
#This plot puts forward the relationship between ozone and temperature:
ggplotly(ggplot(df_complete, aes (x=temp_avg, y=O3))+
           geom_point(shape=22, aes(color=NO2, alpha=0.2))+
           theme_bw() +
           ggtitle("Ozone and Temperature: Relationship coloured by average NO2 leve
l") +
           stat_smooth(method='lm', se=FALSE))
```






This plot confirms that a relationship might exist between ozone and temperature.

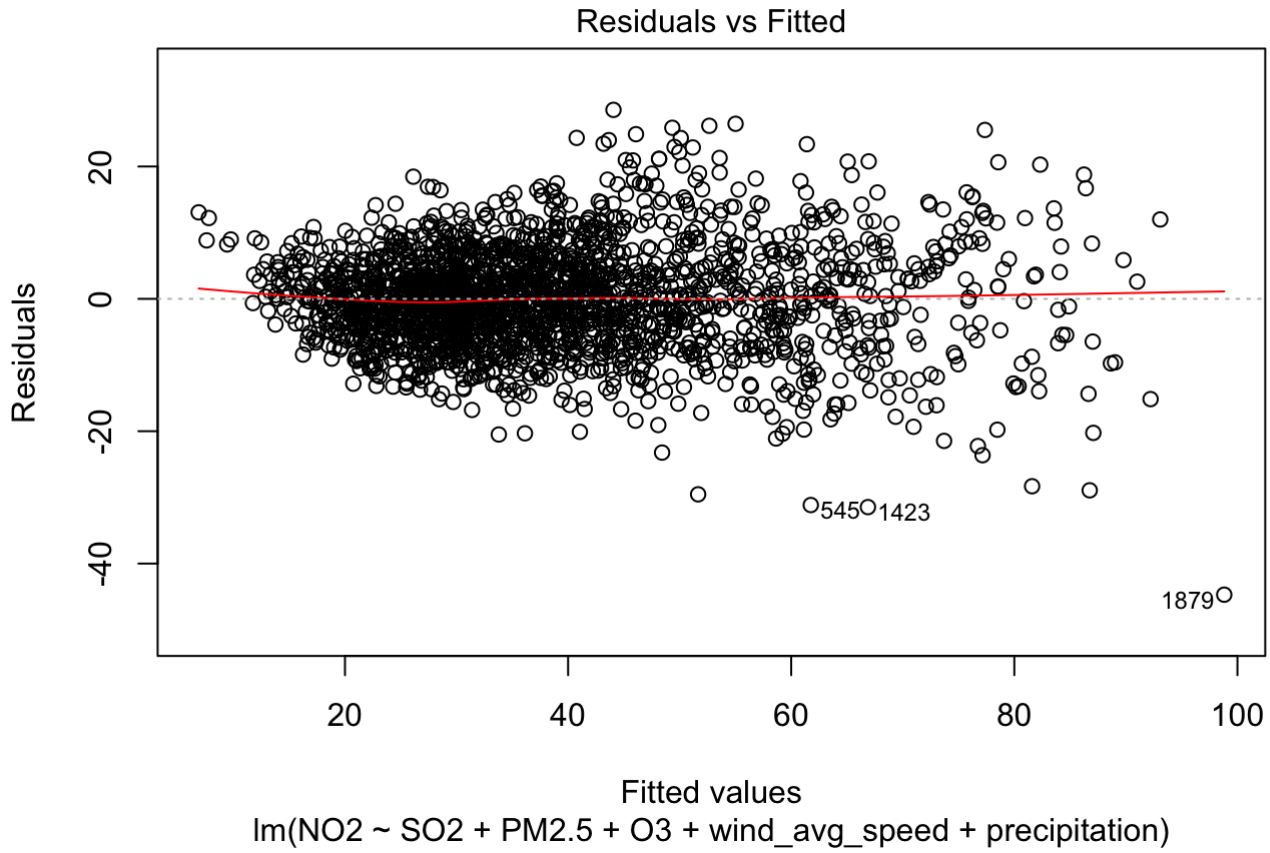
Taking into account that there might be a relationship between O3 and temperature, and that precipitation might not be related to quantities of NO2 in the air, we run 2 additional regression models: -removing temp_avg, keeping all other variables, -removing O3, keeping all other variables,

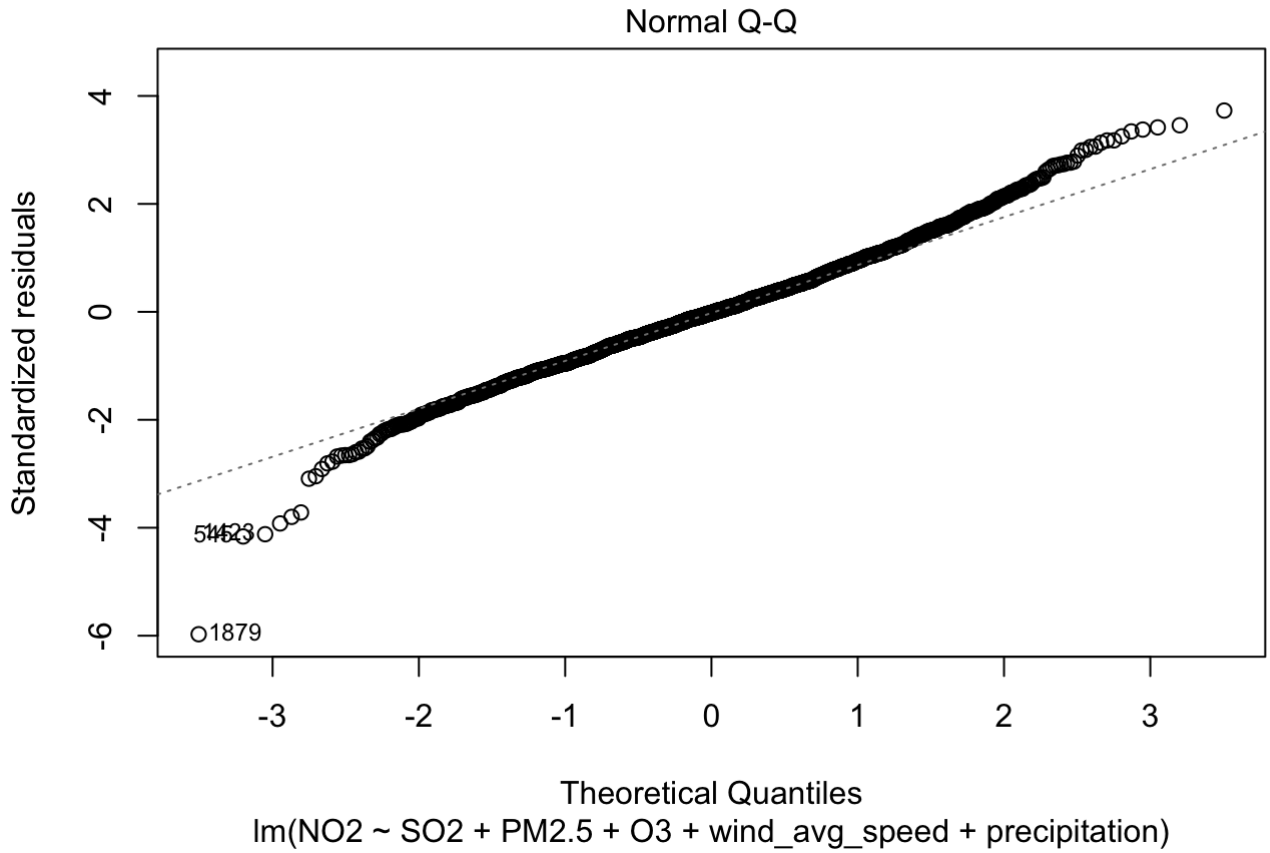
By doing so, we will try to choose a model that fits better. This is the advantage of #doing supervised regression: we can measure the error and try different alternatives.

```
#Regression removing temperature, ceteris paribus
regression_no_temp <- lm(NO2~SO2+PM2.5+ O3+wind_avg_speed+precipitation, df_complete)
summary(regression_no_temp)
```

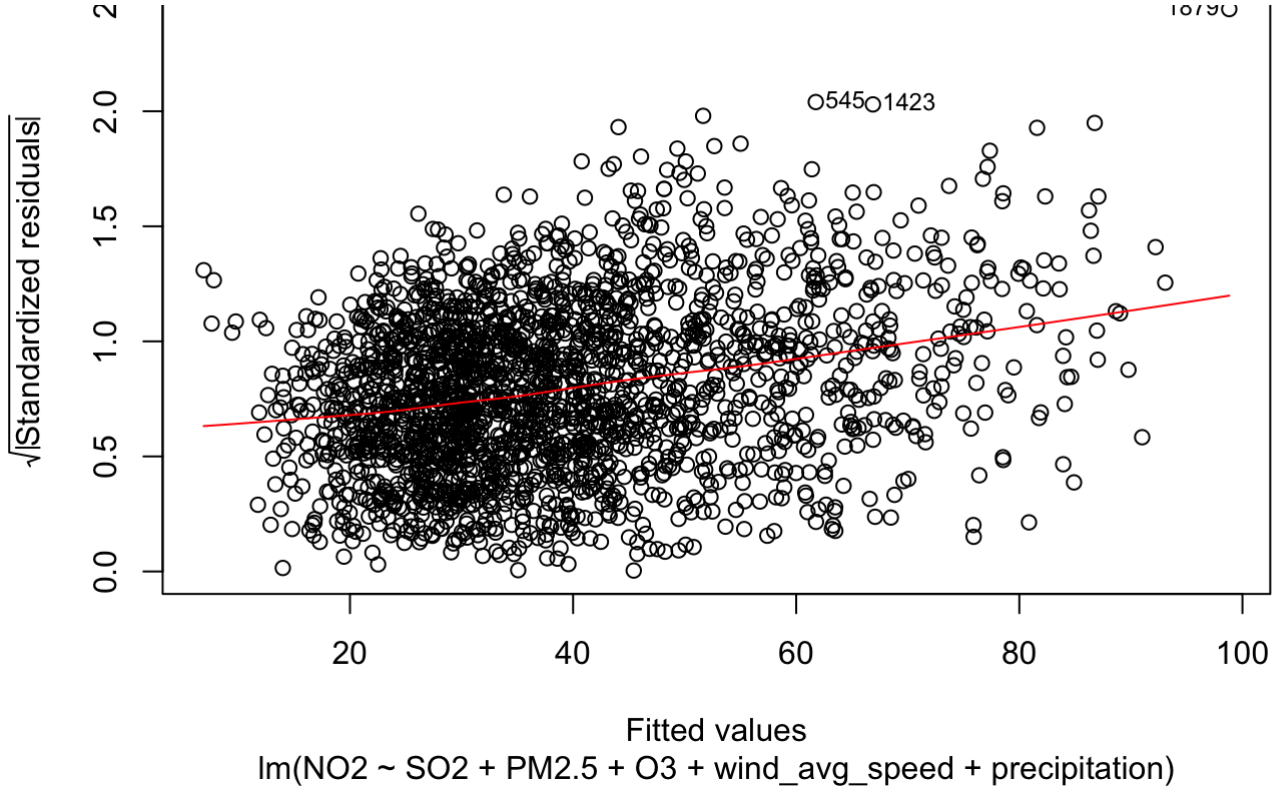
```
##
## Call:
## lm(formula = NO2 ~ SO2 + PM2.5 + O3 + wind_avg_speed + precipitation,
##     data = df_complete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -44.701  -4.743  -0.159   4.424  28.555
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    37.923284   0.960639  39.477  < 2e-16 ***
## SO2             1.900693   0.073549  25.843  < 2e-16 ***
## PM2.5           0.969754   0.036154  26.823  < 2e-16 ***
## O3             -0.290659   0.008608 -33.765  < 2e-16 ***
## wind_avg_speed -0.691562   0.036044 -19.187  < 2e-16 ***
## precipitation  -0.147294   0.051993  -2.833  0.00465 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.66 on 2186 degrees of freedom
## Multiple R-squared:  0.8008, Adjusted R-squared:  0.8003
## F-statistic:  1757 on 5 and 2186 DF,  p-value: < 2.2e-16
```

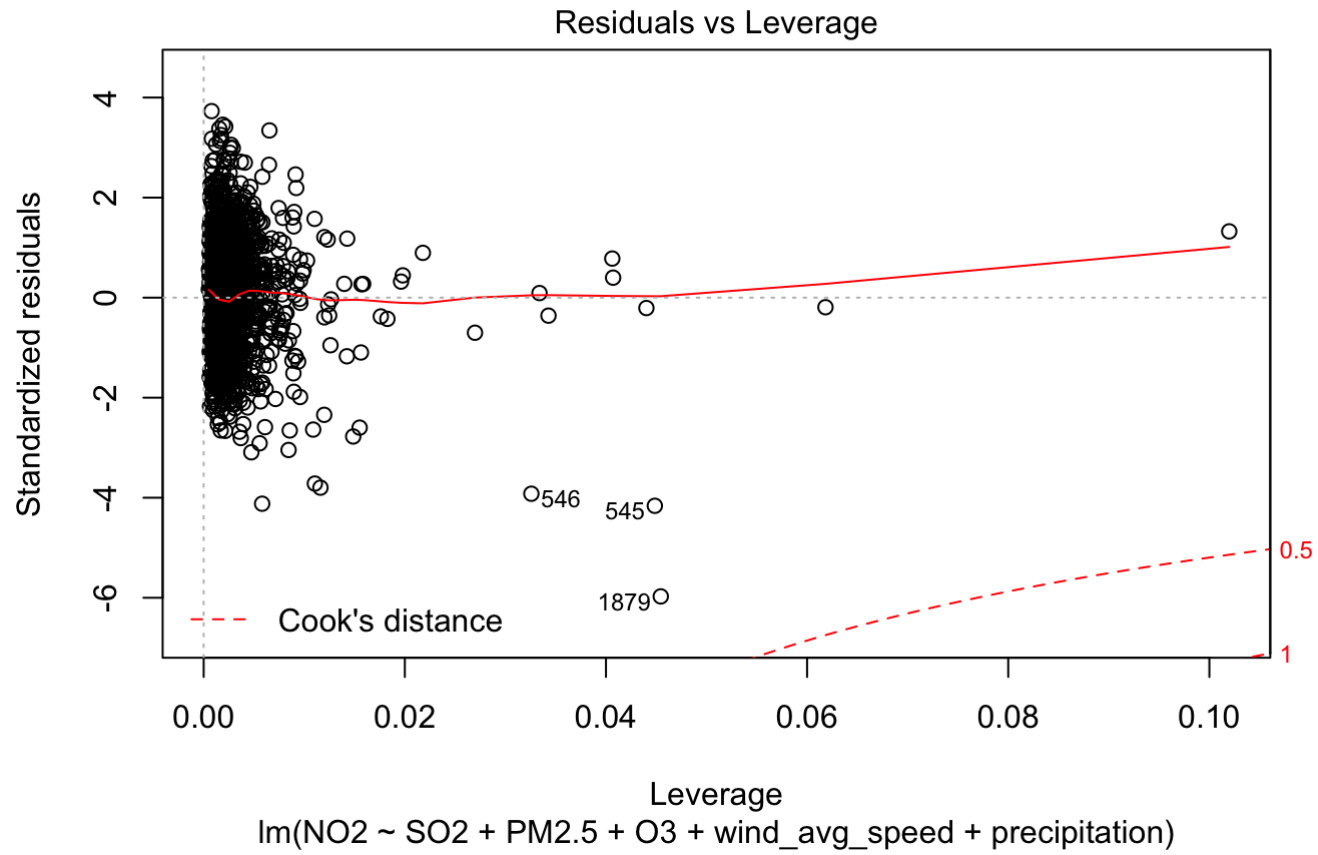
```
plot(regression_no_temp)
```


Residuals vs Fitted
Residuals
20
0
-20
-40
545
1423
1879
20
40
60
80
100
Fitted values
lm(NO2 ~ SO2 + PM2.5 + O3 + wind_avg_speed + precipitation)


Normal Q-Q
Standardized residuals
4
2
0
-2
-4
-6
1423
1879
-3
-2
-1
0
1
2
3
Theoretical Quantiles
lm(NO2 ~ SO2 + PM2.5 + O3 + wind_avg_speed + precipitation)


Scale-Location



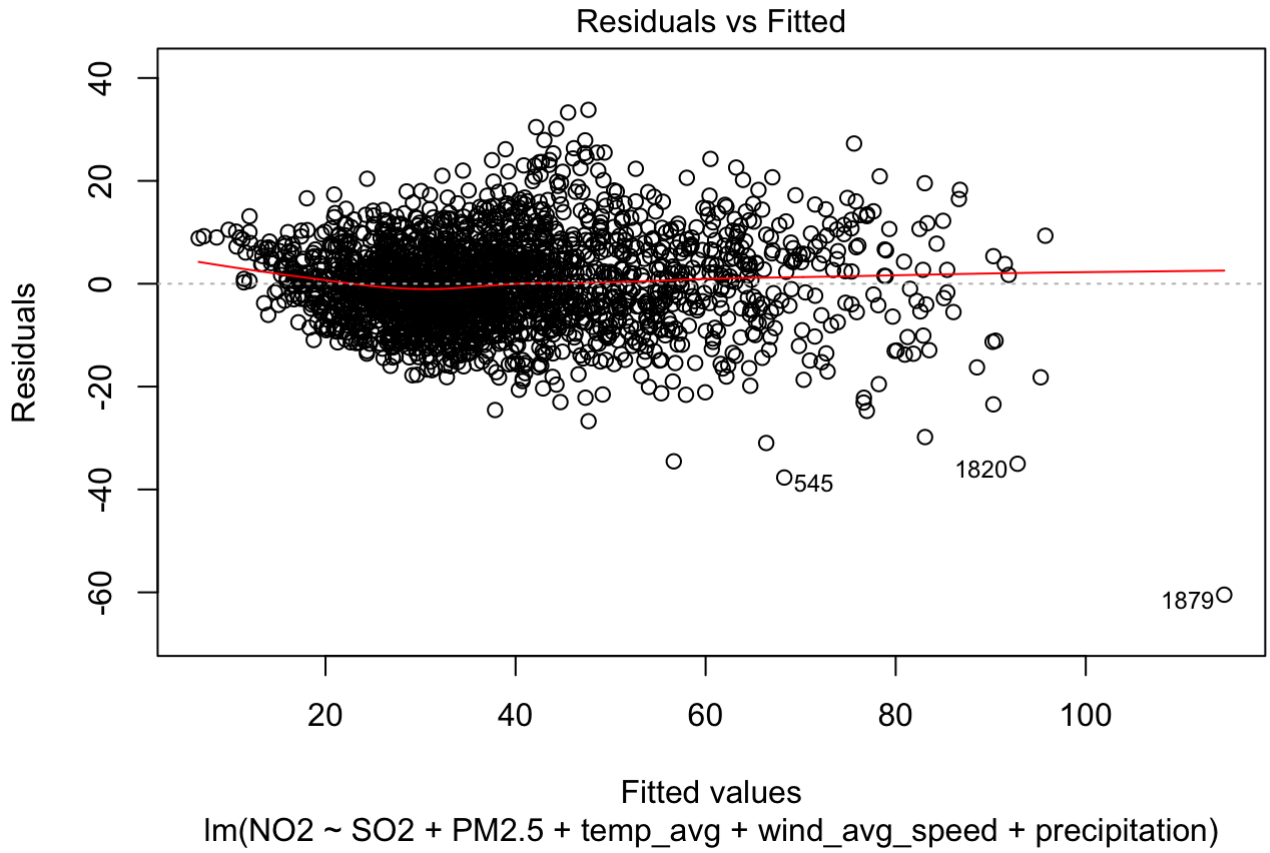



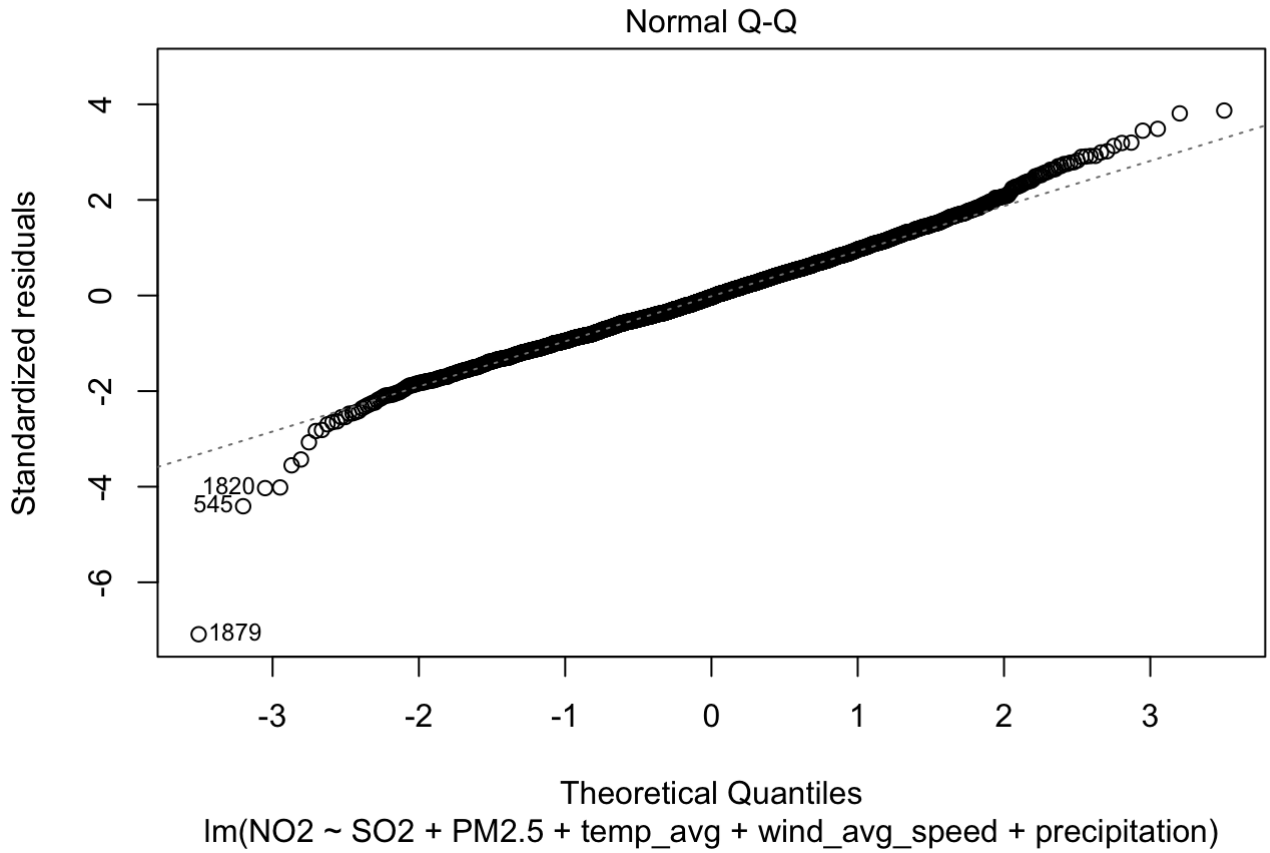

In this regression, all variables are now statistically significant. The lowest score goes to precipitation (with 0.00377). R squared is 0.8002.

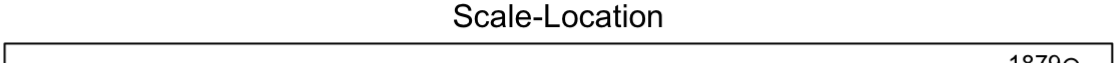
```
#Regression removing O3, ceteris paribus
regression_no_o3 <- lm(NO2~SO2+PM2.5+temp_avg+wind_avg_speed+precipitation, df_comple
te)
summary(regression_no_o3)
```

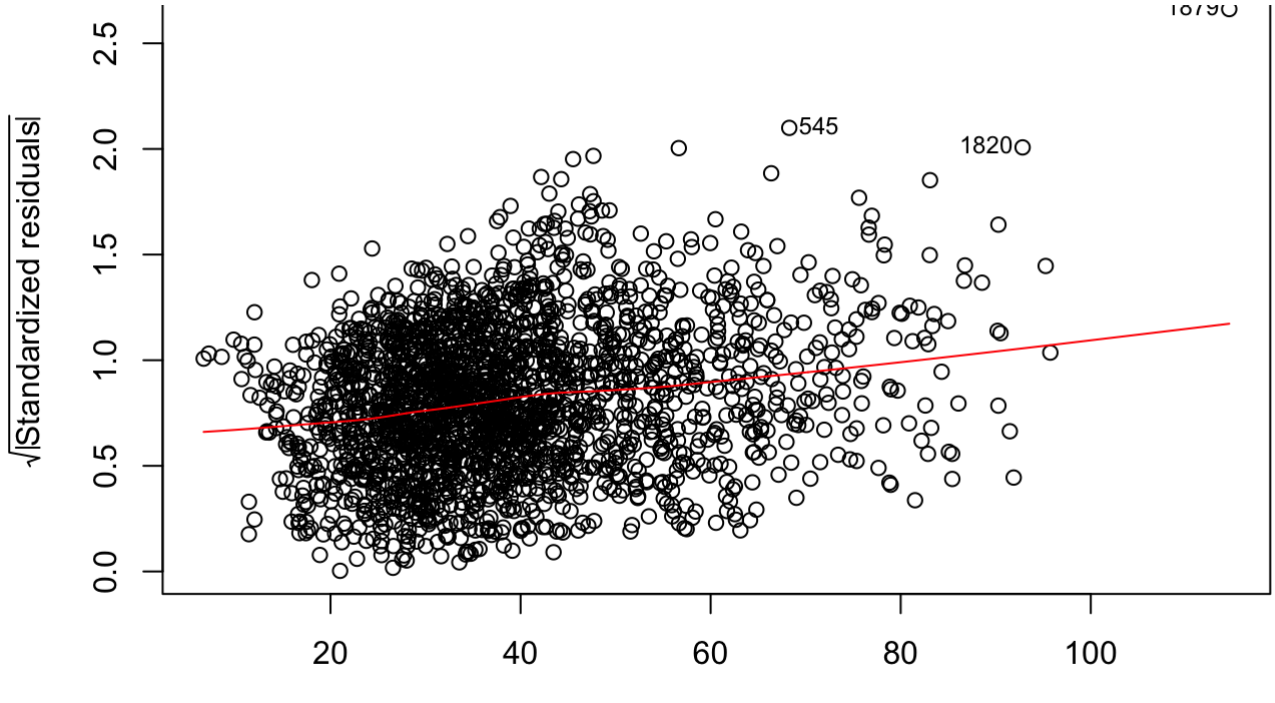
```
##
## Call:
## lm(formula = NO2 ~ SO2 + PM2.5 + temp_avg + wind_avg_speed +
##     precipitation, data = df_complete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -60.454  -5.690  -0.312   5.427  33.805
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    30.94048    1.09553  28.242   <2e-16 ***
## SO2             1.95302    0.09177  21.282   <2e-16 ***
## PM2.5           1.30149    0.04193  31.041   <2e-16 ***
## temp_avg       -0.53264    0.02776 -19.184   <2e-16 ***
## wind_avg_speed -0.96194    0.04070 -23.633   <2e-16 ***
## precipitation  -0.09432    0.05938  -1.588    0.112
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.742 on 2186 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:   0.74
## F-statistic:  1248 on 5 and 2186 DF,  p-value: < 2.2e-16
```

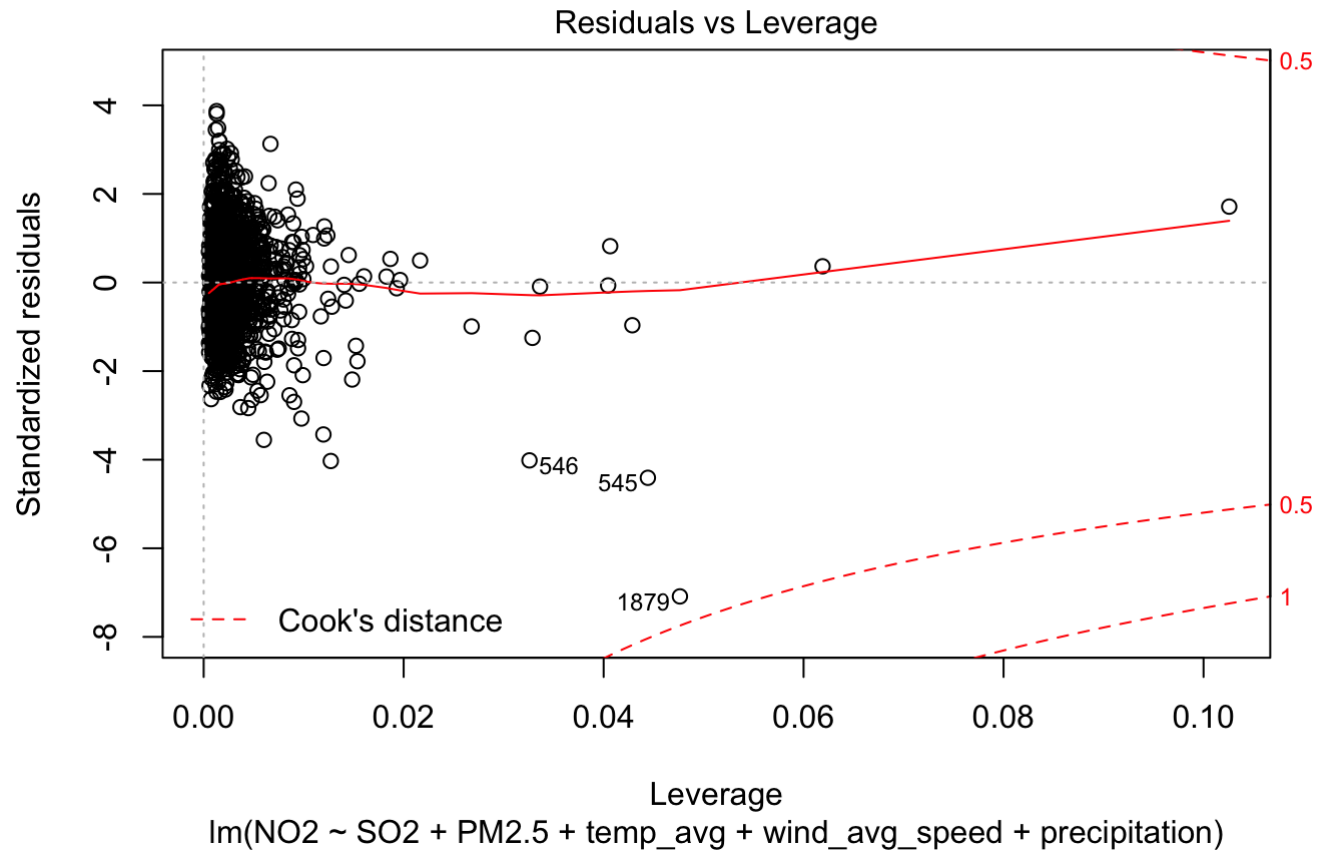
```
plot(regression_no_o3)
```


Residuals vs Fitted
Residuals
Fitted values
lm(NO2 ~ SO2 + PM2.5 + temp_avg + wind_avg_speed + precipitation)
545
1820
1879


Normal Q-Q
Standardized residuals
Theoretical Quantiles
lm(NO2 ~ SO2 + PM2.5 + temp_avg + wind_avg_speed + precipitation)
1820
545
1879


Scale-Location




In this regression, precipitation is “non-significant” with a p-value of 0.1. All other variables are significant. R squared is 0.7398.

The R-squared of the regression without temperature is higher than the regression without O3. Also, in the regression without temperature, all variables are significant.

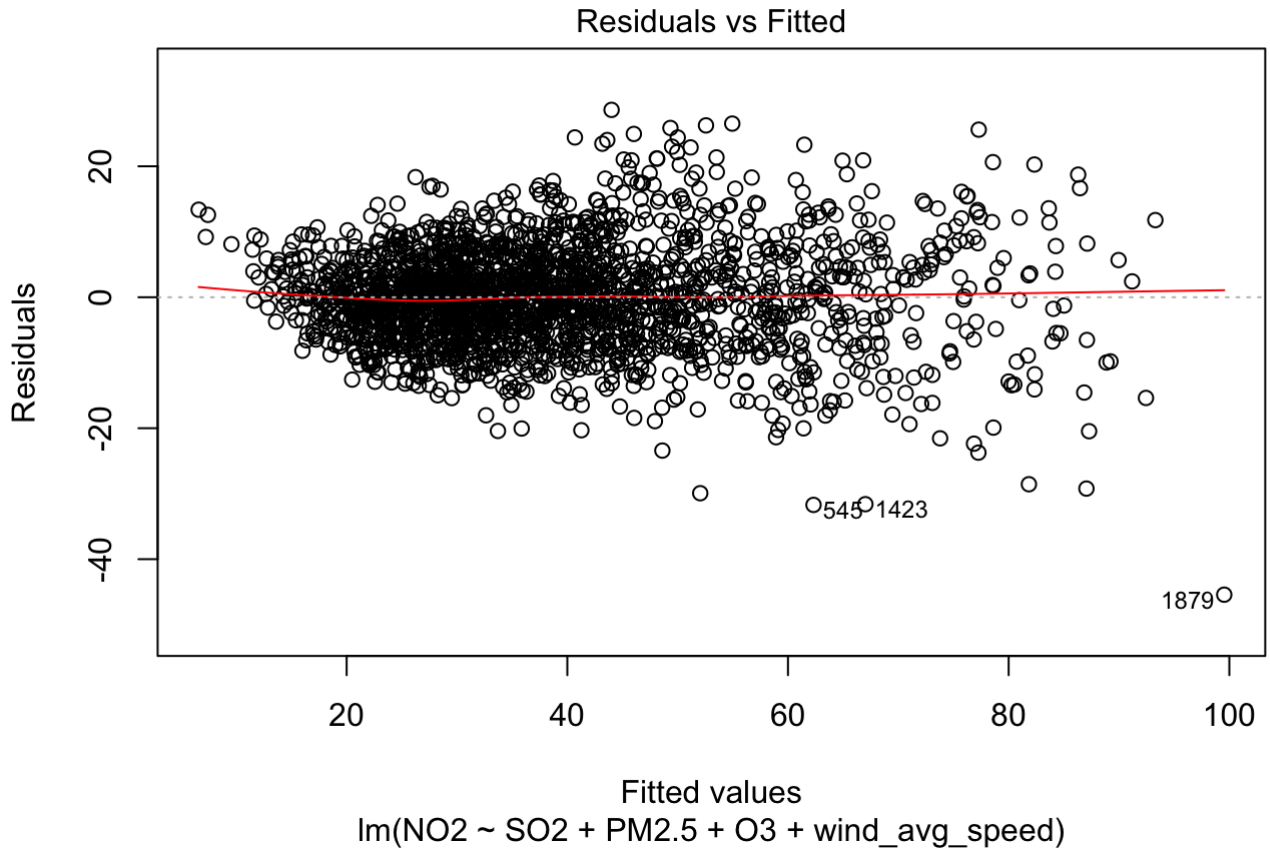
Therefore, if we should choose to keep one variable (O3 or temp_avg), we would keep O3 and discard temp_avg.

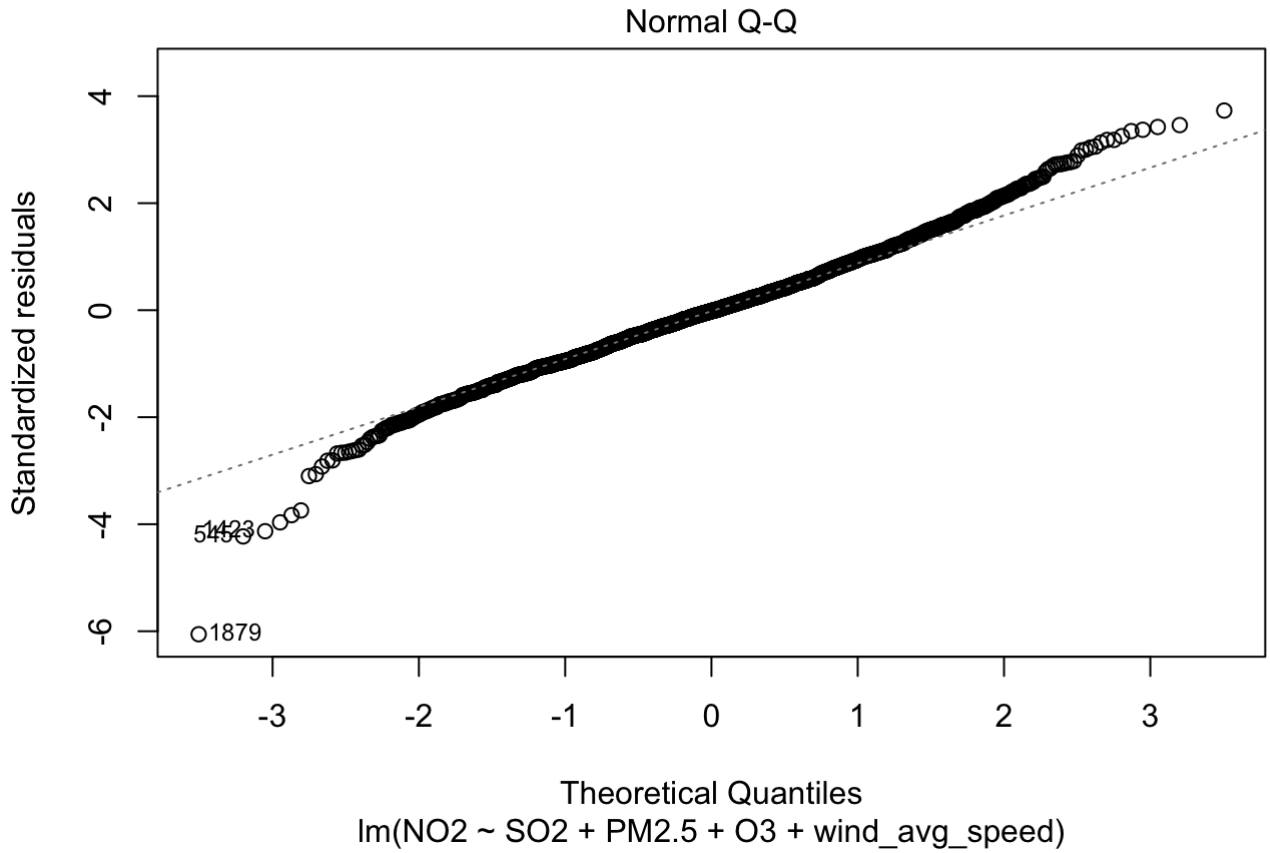
The last step in our work is to see whether precipitation should be removed from the model. Therefore, one last regression is run removing precipitation and temp_avg, keeping all other variables.

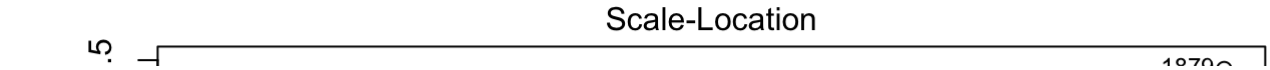
```
#Regression removing precipitation and temp
regression_no_p_no_t<- lm(NO2~SO2+PM2.5+ O3+wind_avg_speed, df_complete)
summary(regression_no_p_no_t)
```

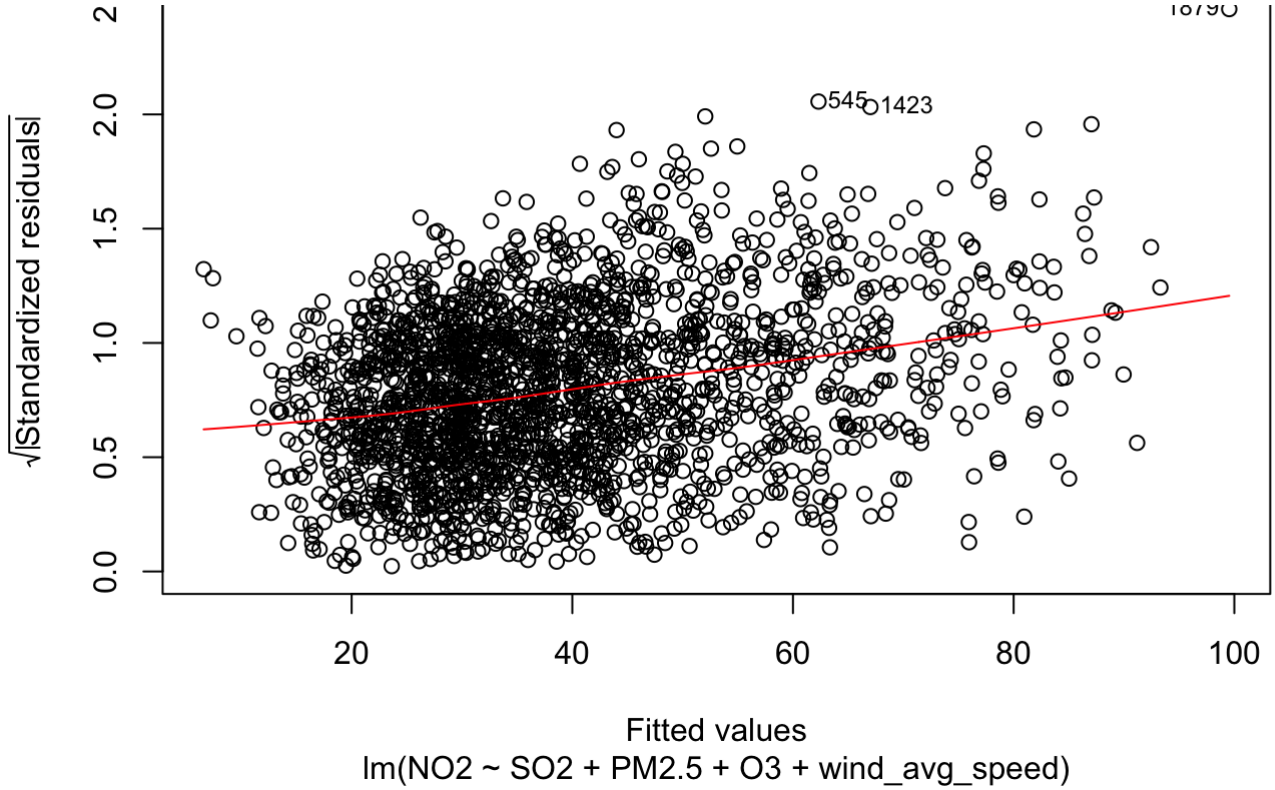
```
##
## Call:
## lm(formula = NO2 ~ SO2 + PM2.5 + O3 + wind_avg_speed, data = df_complete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -45.423  -4.764  -0.123   4.482  28.613
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    37.518858   0.951497   39.43   <2e-16 ***
## SO2             1.903744   0.073659   25.84   <2e-16 ***
## PM2.5           0.987982   0.035634   27.73   <2e-16 ***
## O3             -0.288222   0.008579  -33.60   <2e-16 ***
## wind_avg_speed -0.698251   0.036024  -19.38   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.673 on 2187 degrees of freedom
## Multiple R-squared:  0.8001, Adjusted R-squared:  0.7997
## F-statistic:  2188 on 4 and 2187 DF,  p-value: < 2.2e-16
```

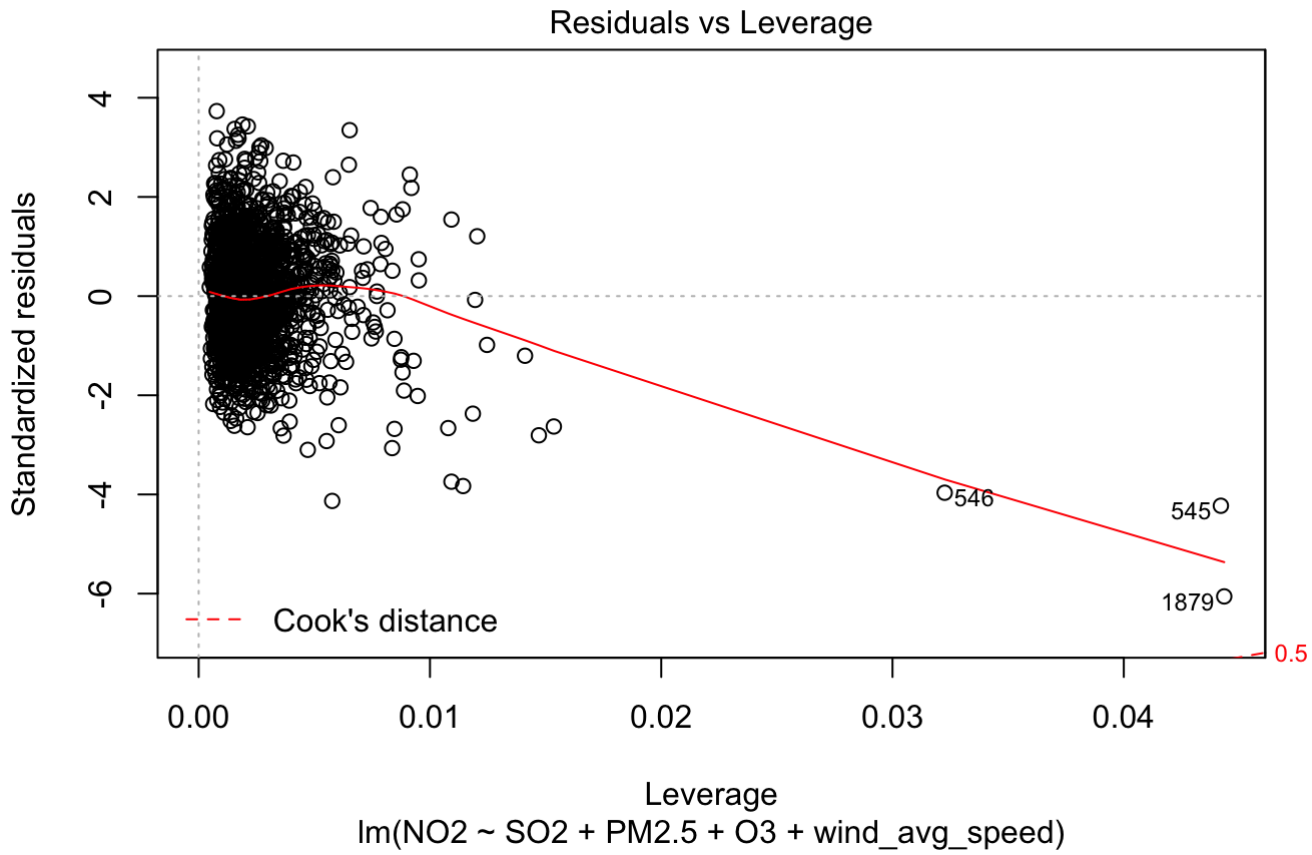
```
plot(regression_no_p_no_t)
```


Residuals vs Fitted
Residuals
20
0
-20
-40
545
1423
1879
20
40
60
80
100
Fitted values
lm(NO2 ~ SO2 + PM2.5 + O3 + wind_avg_speed)


Normal Q-Q
Standardized residuals
4
2
0
-2
-4
-6
1423
545
1879
-3
-2
-1
0
1
2
3
Theoretical Quantiles
lm(NO2 ~ SO2 + PM2.5 + O3 + wind_avg_speed)


Scale-Location




For this final regression, although, all variables are now highly significant, R squared is equal to 0.7995. This result is lower than the one for the regression removing temperature and keeping precipitation.

To conclude, we would choose the model without temperature but with precipitation, keeping all other variables.