China Contamination

Data Science: Capstone Proyect

Jorge Haces

16/6/2020

Contents

1 Overview

This is the final project required for **Data Science: Capstone** course offered by *edX HarvardX for Professional Certificate Program in Data Science*. The theme of this proyect is the Contamination, specifically in China and the aim is to predict the air quality in the fastest growing country nowadays.

1.1 Introduction

Contanimation is defined as the presence of materials in the air that cause serious harm or discomfort to people. The contamination has increased since the Industrial Revolution began, in the second half of the 18th century, with production processes in factories, the development of transportation and the use of fuels.

According to the World Health Organization (WHO), the state of the current atmosphere causes, by the simple act of breathing, the death of around seven million people a year (fine particle respiration).

The most common air pollutant gases are carbon monoxide, sulfur dioxide, chlorofluorocarbons, and nitrogen oxides. Photochemicals such as ozone and smog are increased in the air by nitrogen oxides and hydrocarbons reacting with sunlight.

Contaminants are classified into:

- Primaries are those that are emitted directly into the atmosphere such as sulfur dioxide, carbon monoxide
- Secondary are those that are formed by atmospheric chemical processes that act on primary contaminants such as sulfuric acid, which is formed by the oxidation of sulfur dioxide, nitrogen dioxide that is formed by oxidizing the primary pollutant nitric oxide and ozone that is formed from oxygen. [1]

1.2 Project Description

An analysis of the data will be carried out based on the following models: k Nearest Neighbors, Logistic Regression, Support Vector Machines (SVM), Random Forests and Neural Network to help us predict if pollution will grow even more (2.3% in 2018 almost at double compared to 2010). To answer the question, will china be able to comply with the Paris agreement signed in 2015? Whose goal is to reduce the global temperature to 2°C in 2050.

For this we will divide the data into two: training data and test data. Later, we will train the different models in the first set and then will be evaluated in the second set. Finally, we will use the **Root-Mean-Square-Error (RMSE)** to rate the performance of each model and thus identify the best for this project.

1.3 DataSet

The Dataset used in this project is *Beijing Multi-Site Air-Quality Data Data Set*, available at the UCI Machine Learning Repository [2].

This data set includes hourly air pollutants data from 12 nationally-controlled air-quality monitoring sites. The air-quality data are from the Beijing Municipal Environmental Monitoring Center. The meteorological data in each air-quality site are matched with the nearest weather station from the China Meteorological Administration. The time period is from March 1st, 2013 to February 28th, 2017. Missing data are denoted as NA.

The Attribute Information is the following:

- No: row number
- year: year of data in this row
- month: month of data in this row
- day: day of data in this row
- hour: hour of data in this row
- PM2.5: PM2.5 concentration (ug/mˆ3)
- PM10: PM10 concentration (ug/mˆ3)
- SO2: SO2 concentration (ug/mˆ3)
- NO2: NO2 concentration (ug/mˆ3)
- CO: CO concentration (ug/mˆ3)
- O3: O3 concentration (ug/mˆ3)
- TEMP: temperature (degree Celsius)
- PRES: pressure (hPa)
- DEWP: dew point temperature (degree Celsius)
- RAIN: precipitation (mm)
- wd: wind direction
- WSPM: wind speed (m/s)
- station: name of the air-quality monitoring site

The data is contained in a Zip file named *PRSA2017_Data_20130301-20170228.zip* containing 12 files (one for each municipality), as follow:

- PRSA_Data_Aotizhongxin_20130301-20170228.csv
- PRSA_Data_Changping_20130301-20170228.csv
- PRSA_Data_Dingling_20130301-20170228.csv
- PRSA_Data_Dongsi_20130301-20170228.csv
- PRSA_Data_Guanyuan_20130301-20170228.csv
- PRSA_Data_Gucheng_20130301-20170228.csv
- PRSA_Data_Huairou_20130301-20170228.csv
- PRSA_Data_Nongzhanguan_20130301-20170228.csv
- PRSA_Data_Shunyi_20130301-20170228.csv
- PRSA_Data_Tiantan_20130301-20170228.csv
- PRSA_Data_Wanliu_20130301-20170228.csv
- PRSA_Data_Wanshouxigong_20130301-20170228.csv

2 Methods and Analysis

2.1 Data Stage

Next, the UCI data will be downloaded in ZIP format to decompress them and load the 12 files in a single variable called PRSA, identifying the data to be analyzed with 420,768 records and 18 attributes (columns).

```
## [1] 420768     18
```

Subsequently, we execute the nearZeroVar function to identify the attributes that have no significant variation in our data set; In this case, the attribute "Rain" does not present significant variations, so we remove that attribute.

```
# Structure of the data (data type, numbers of rows, number of attributes)
str(PRSA)
```

```
## tibble [420,768 x 18] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ No     : num [1:420768] 1 2 3 4 5 6 7 8 9 10 ...
##  $ year   : num [1:420768] 2013 2013 2013 2013 2013 ...
##  $ month  : num [1:420768] 3 3 3 3 3 3 3 3 3 3 ...
##  $ day    : num [1:420768] 1 1 1 1 1 1 1 1 1 1 ...
##  $ hour   : num [1:420768] 0 1 2 3 4 5 6 7 8 9 ...
##  $ PM2.5  : num [1:420768] 4 8 7 6 3 5 3 3 3 3 ...
##  $ PM10   : num [1:420768] 4 8 7 6 3 5 3 6 6 8 ...
##  $ SO2    : num [1:420768] 4 4 5 11 12 18 18 19 16 12 ...
##  $ NO2    : num [1:420768] 7 7 10 11 12 18 32 41 43 28 ...
##  $ CO     : num [1:420768] 300 300 300 300 300 400 500 500 500 400 ...
##  $ O3     : num [1:420768] 77 77 73 72 72 66 50 43 45 59 ...
##  $ TEMP   : num [1:420768] -0.7 -1.1 -1.1 -1.4 -2 -2.2 -2.6 -1.6 0.1 1.2 ...
##  $ PRES   : num [1:420768] 1023 1023 1024 1024 1025 ...
##  $ DEWP   : num [1:420768] -18.8 -18.2 -18.2 -19.4 -19.5 -19.6 -19.1 -19.1 -19.2 -19.3 ...
##  $ RAIN   : num [1:420768] 0 0 0 0 0 0 0 0 0 0 ...
##  $ wd     : chr [1:420768] "NNW" "N" "NNW" "NW" ...
##  $ WSPM   : num [1:420768] 4.4 4.7 5.6 3.1 2 3.7 2.5 3.8 4.1 2.6 ...
##  $ station: chr [1:420768] "Aotizhongxin" "Aotizhongxin" "Aotizhongxin" "Aotizhongxin" ...
```

```
#Near zero variance ( identify the attributes that do not give us valuable data )
nzv <- nearZeroVar(PRSA)

#Remove the nzv columns, NA data and save in another variable
data <- PRSA[,-nzv]
data<- na.omit(data)
```

Therefore, we will continue with 17 attributes and 382,172 rows.

```
# Dimensions of the valuable data (rows,columns)
dim(data)
```

```
## [1] 382172     17
```

It was identified that the attribute "No" is a consecutive attribute and not contribute anything to the data, therefore it will also be discarded. It is identified that all the attributes are numerical type with the exception of "station" which is character type, therefore, the attributes will be transformed to factor type to make the data set more efficient.

```
# Show the data in a transposed version to see more data
glimpse(data)
```

```
## Rows: 382,172
```

```
## Columns: 17
## $ No      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ...
## $ year    <dbl> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013,...
## $ month   <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ day     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ hour    <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1...
## $ PM2.5   <dbl> 4, 8, 7, 6, 3, 5, 3, 3, 3, 3, 3, 3, 3, 3, 6, 8, 9, 10, 11, ...
## $ PM10    <dbl> 4, 8, 7, 6, 3, 5, 3, 6, 6, 8, 6, 6, 6, 6, 9, 15, 19, 23, 20...
## $ SO2     <dbl> 4, 4, 5, 11, 12, 18, 18, 19, 16, 12, 9, 9, 7, 7, 7, 7, 9, 1...
## $ NO2     <dbl> 7, 7, 10, 11, 12, 18, 32, 41, 43, 28, 12, 14, 13, 12, 11, 1...
## $ CO      <dbl> 300, 300, 300, 300, 300, 400, 500, 500, 500, 400, 400, 400,...
## $ O3      <dbl> 77, 77, 73, 72, 72, 66, 50, 43, 45, 59, 72, 71, 74, 76, 77,...
## $ TEMP    <dbl> -0.7, -1.1, -1.1, -1.4, -2.0, -2.2, -2.6, -1.6, 0.1, 1.2, 1...
## $ PRES    <dbl> 1023.0, 1023.2, 1023.5, 1024.5, 1025.2, 1025.6, 1026.5, 102...
## $ DEWP    <dbl> -18.8, -18.2, -18.2, -19.4, -19.5, -19.6, -19.1, -19.1, -19...
## $ wd      <chr> "NNW", "N", "NNW", "NW", "N", "N", "NNE", "NNW", "NNW", "N"...
## $ WSPM    <dbl> 4.4, 4.7, 5.6, 3.1, 2.0, 3.7, 2.5, 3.8, 4.1, 2.6, 3.6, 3.7,...
## $ station <chr> "Aotizhongxin", "Aotizhongxin", "Aotizhongxin", "Aotizhongx...
```

```r
#Remove the "no" variable because is a consecutive (also do not give us valuable data)
data <- data[,-c(1)]

# Cast the attributes year, month, day, hour, wd and station from "dbl" type to "factor" type
data$year <- as.factor(data$year)
data$month <- as.factor(data$month)
data$day <- as.integer(data$day)
data$hour <- as.integer(data$hour)
data$wd <- as.factor(data$wd)
data$station <- as.factor(data$station)
```

Similarly, it is validated that the PM (2.5 and 10) attributes; in this case, only the attribute "CO" will be transformed to an integer.

```
##   [1]   4.0   8.0   7.0   6.0   3.0   5.0   9.0  10.0  11.0  12.0  15.0  24.0
##  [13]  22.0  14.0  13.0  18.0  26.0  25.0  37.0  44.0  54.0  61.0  67.0  74.0
##  [25]  81.0  93.0 112.0 109.0 110.0 105.0 106.0 101.0  91.0  79.0  77.0  83.0
##  [37]  94.0  96.0  98.0  72.0  48.0  28.0  19.0  60.0 117.0  46.0  42.0  49.0
##  [49]  16.0  20.0  31.0  34.0  40.0  51.0  58.0  71.0 100.0 103.0 115.0 104.0
##  [61] 111.0 114.0 131.0 138.0 142.0 163.0 168.0 165.0 166.0 190.0 203.0 210.0
##  [73] 205.0 215.0 219.0 226.0 284.0 272.0 242.0 212.0 188.0 186.0 194.0 184.0
##  [85] 175.0 179.0 192.0 201.0 230.0 246.0 248.0 239.0 254.0 266.0 260.0 250.0
##  [97] 240.0 238.0 243.0 275.0 306.0 320.0 292.0 251.0 228.0 217.0 255.0 258.0
## [109] 277.0 282.0 267.0 216.0 211.0 227.0 363.0 376.0 344.0 339.0 315.0 310.0
## [121] 309.0 326.0 121.0 330.0 172.0  82.0  84.0  90.0  89.0  97.0 102.0 108.0
## [133] 113.0 145.0  69.0  47.0  21.0  17.0  27.0  30.0  36.0  38.0  70.0  64.0
## [145]  95.0 120.0 141.0 140.0 149.0 150.0 132.0 144.0 156.0 174.0 152.0 125.0
## [157] 146.0 116.0 123.0 130.0 129.0 139.0 135.0 133.0 124.0 137.0 160.0  88.0
## [169]  50.0  23.0  35.0  62.0  65.0  73.0 119.0 162.0 180.0 183.0 182.0 173.0
## [181] 164.0 236.0 256.0 287.0 278.0 269.0 286.0 304.0 312.0 316.0 353.0 355.0
## [193] 328.0 308.0 314.0 276.0 264.0 270.0 185.0 197.0 176.0 206.0 257.0 293.0
## [205] 322.0 294.0 332.0 352.0 362.0 356.0 357.0 364.0 392.0 423.0 434.0 450.0
## [217] 463.0  78.0  55.0  59.0  45.0  52.0  39.0  68.0  33.0  41.0  92.0 178.0
## [229] 181.0 187.0 202.0  63.0  56.0  75.0  66.0  76.0 118.0 209.0 220.0 223.0
## [241] 214.0 200.0 199.0 204.0 222.0 229.0 143.0 128.0  57.0  80.0  86.0 151.0
## [253] 221.0 225.0 235.0 233.0  43.0 126.0 161.0 136.0 134.0 147.0 154.0 153.0
```

```
## [265] 157.0 208.0 122.0  85.0  32.0  99.0  53.0 107.0 127.0 159.0 171.0 189.0
## [277] 198.0 232.0 263.0 237.0 169.0 193.0  29.0  87.0 195.0 366.0 170.0 252.0
## [289] 262.0 279.0 288.0 273.0 311.0 303.0 167.0 665.0 158.0 148.0 155.0 218.0
## [301] 191.0 464.0 494.0 485.0 510.0 430.0 281.0 247.0 271.0 245.0 224.0 213.0
## [313] 249.0 177.0 253.0 234.0 207.0 265.0 283.0 290.0 297.0 321.0 295.0 280.0
## [325] 340.0 349.0 334.0 289.0 305.0 261.0 345.0 327.0 313.0 302.0 196.0 268.0
## [337] 274.0 231.0 244.0 342.0 367.0 307.0 241.0 361.0 341.0 317.0 285.0 300.0
## [349] 388.0 385.0 378.0 371.0 413.0 420.0 368.0 335.0 350.0 337.0 296.0 402.0
## [361] 259.0 436.0 472.0 499.0 511.0 530.0 538.0 535.0 521.0 475.0 398.0 383.0
## [373] 389.0 396.0 418.0 422.0 333.0 298.0 343.0 481.0 501.0 584.0 525.0 414.0
## [385] 455.0 505.0 466.0 401.0 403.0 412.0 503.0 500.0 488.0 407.0 358.0 379.0
## [397] 375.0 323.0 324.0 291.0 299.0 325.0 377.0 427.0 416.0 394.0 346.0 483.0
## [409] 426.0 445.0 440.0 461.0 518.0 319.0 318.0 301.0 374.0 370.0 410.0 354.0
## [421]  87.8  84.6 225.6  26.8 102.3 113.6  81.3 381.0 393.0 384.0 336.0 338.0
## [433] 446.0 435.0 351.0 329.0 360.0 432.0 431.0 409.0   8.6 404.0 419.0 411.0
## [445] 347.0 365.0 390.0 408.0 497.0 469.0 438.0 425.0 399.0 114.1  66.2  67.5
## [457] 397.0 359.0 380.0 331.0 369.0 492.0 453.0 473.0 529.0 439.0 635.0 546.0
## [469] 476.0 477.0 428.0 527.0 523.0 382.0 519.0 489.0 498.0 541.0 532.0 547.0
## [481] 713.0 615.0 585.0 577.0 544.0 348.0 433.0 406.0 386.0 448.0 395.0 373.0
## [493] 400.0 387.0 405.0 451.0 454.0 421.0 424.0 542.0 568.0 543.0 522.0 487.0
## [505] 459.0 417.0 429.0 437.0 447.0 467.0 441.0 576.0 641.0 651.0 682.0 697.0
## [517] 462.0 443.0 581.0 524.0 444.0  74.5  75.8  80.7  80.4 154.2 540.0  26.9
## [529]  51.7  76.7  15.5  18.9 112.4  79.5 479.0 470.0 458.0 471.0 415.0 560.0
## [541] 662.0 495.0 478.0 491.0 456.0 539.0 442.0 548.0  77.4  83.3  78.7  88.6
## [553] 139.7   9.6   7.9   7.2  15.7 120.4   8.4  82.1  72.4 391.0   4.3   8.5
## [565] 117.9 103.8  61.2  12.6  13.8 468.0 515.0 506.0 496.0 614.0 632.0 617.0
## [577] 647.0 604.0 594.0 513.0 509.0 490.0 564.0 610.0 474.0 372.0 493.0 536.0
## [589] 520.0 531.0 553.0 598.0 593.0 597.0 603.0 590.0 555.0 449.0 516.0 105.4
## [601]  92.8  94.1 238.6  23.7  42.3  97.9  45.4 120.7  84.7 147.9   4.4  10.7
## [613] 174.3  74.3 460.0 512.0 596.0 624.0 626.0 629.0 638.0 666.0 670.0 685.0
## [625] 507.0 628.0 637.0 678.0 671.0 482.0 569.0 625.0 558.0 561.0 606.0 612.0
## [637] 557.0 575.0 573.0 622.0 660.0 554.0 504.0 642.0 695.0 661.0 583.0 537.0
## [649] 508.0 480.0 457.0 484.0 572.0 580.0 589.0 563.0 533.0 514.0 654.0 646.0
## [661] 681.0 528.0 595.0 578.0 549.0 486.0 502.0 116.7  23.8  99.1 111.3  71.4
## [673] 121.7  14.3 256.9 136.2  62.8 556.0 609.0 636.0 599.0 592.0 645.0 680.0
## [685] 550.0 605.0 633.0 627.0 465.0 526.0 534.0 639.0 517.0 570.0 545.0 664.0
## [697] 567.0 663.0   2.0 602.0 770.0 110.9 156.7  88.9  78.4  80.5  40.3  15.9
## [709]  38.6  20.6  58.8  70.7 123.7  48.8 452.0 114.6 652.0 574.0  70.3 565.0
## [721] 675.0 733.0 741.0 588.0 649.0 705.0 559.0 600.0 571.0 551.0 634.0 608.0
## [733] 640.0 767.0 566.0  68.9  72.5  77.2 193.3  13.5  42.5  92.6  65.3  53.5
## [745]  12.7 144.6  55.1 659.0  83.7  81.8  55.7 683.0 844.0 809.0 781.0 687.0
## [757]  91.3  78.2  32.2  85.6  23.6  81.7  91.7 100.9 623.0 197.1  71.2 601.0
## [769] 611.0 586.0 552.0 835.0 744.0  41.2 208.1  13.7  67.4  99.4  92.4  14.7
## [781]  52.8  83.9  11.2   4.6 613.0 127.2  38.1  20.8 689.0 631.0 644.0 707.0
## [793] 587.0 762.0 650.0  18.3  28.1  92.1 116.9  86.8 145.8 667.0 591.0 618.0
## [805] 821.0 801.0 758.0 720.0 712.0 691.0 125.7  81.2 197.2  19.3 104.1 150.8
## [817] 115.5  71.5  85.2  20.7  11.5 105.6 708.0 619.0 657.0 692.0  12.5 607.0
## [829] 704.0 616.0 655.0 111.8  98.5  89.1 224.5  13.4  92.9 113.3  77.9 125.3
## [841]   8.8 106.4 153.8 669.0 748.0 579.0 804.0 730.0 823.0 621.0

##    [1]   4.0   8.0   7.0   6.0   3.0   5.0   9.0  15.0  19.0  23.0  20.0  14.0
##   [13]  17.0  18.0  24.0  13.0  11.0  10.0  29.0  30.0  33.0  35.0  40.0  46.0
##   [25]  58.0  79.0  86.0  96.0 103.0 113.0 120.0 130.0 132.0 129.0 136.0 135.0
##   [37] 142.0 116.0 110.0 119.0 122.0 117.0 108.0 134.0 106.0 114.0 100.0  82.0
```

```
##   [49]  71.0 175.0 181.0 105.0  94.0  83.0  80.0  32.0  36.0  28.0  12.0  44.0
##   [61]  63.0  84.0  85.0  91.0 127.0 151.0 153.0 145.0 146.0 147.0 159.0 177.0
##   [73] 184.0 193.0 182.0 171.0 186.0 218.0 248.0 255.0 244.0 252.0 253.0 269.0
##   [85] 315.0 300.0 265.0 229.0 194.0 196.0 205.0 198.0 212.0 219.0 233.0 277.0
##   [97] 285.0 276.0 291.0 338.0 396.0 380.0 335.0 360.0 319.0 297.0 294.0 293.0
##  [109] 304.0 337.0 366.0 374.0 344.0 282.0 311.0 283.0 284.0 326.0 310.0 257.0
##  [121] 287.0 298.0 377.0 452.0 426.0 389.0 370.0 346.0 327.0 318.0 371.0 362.0
##  [133] 225.0 348.0 508.0 128.0 123.0 104.0  87.0 102.0  67.0  97.0 125.0 118.0
##  [145] 143.0 139.0 166.0 272.0 587.0 628.0 662.0 443.0 263.0 251.0 107.0  34.0
##  [157]  16.0  25.0  21.0  74.0  75.0  59.0  56.0  52.0  54.0 115.0 140.0 156.0
##  [169] 138.0 155.0 207.0 189.0 133.0 162.0 214.0 246.0 124.0 161.0 237.0 165.0
##  [181] 121.0  90.0  93.0 131.0 101.0  31.0  50.0  22.0  38.0  47.0  39.0  45.0
##  [193]  88.0  66.0  53.0  77.0  81.0  95.0 112.0 126.0 137.0 158.0 179.0 176.0
##  [205] 211.0 200.0 170.0 174.0 185.0 331.0 261.0 279.0 305.0 343.0 320.0 324.0
##  [217] 367.0 430.0 411.0 357.0 330.0 316.0 289.0 273.0 243.0 169.0 167.0 204.0
##  [229] 217.0 223.0 242.0 192.0 195.0 188.0 210.0 259.0 321.0 347.0 372.0 392.0
##  [241] 383.0 369.0 395.0 390.0 442.0 462.0 455.0 494.0 476.0  68.0  76.0  42.0
##  [253]  61.0  57.0  49.0  78.0  27.0  51.0  48.0  89.0 141.0 163.0 172.0 187.0
##  [265] 202.0 221.0 240.0 236.0 234.0 209.0 168.0 152.0 157.0 203.0 260.0 292.0
##  [277] 264.0 256.0 241.0 274.0 302.0 230.0 206.0  70.0  26.0  73.0  99.0  98.0
##  [289]  92.0 111.0 231.0 222.0 247.0 250.0 245.0 239.0 280.0 268.0 267.0 232.0
##  [301]  41.0 249.0 178.0 227.0 238.0 197.0 275.0 351.0 358.0 191.0  62.0 144.0
##  [313] 190.0 149.0  43.0  69.0  72.0  55.0  60.0 215.0 150.0 109.0 164.0 173.0
##  [325] 312.0  37.0 154.0  64.0 183.0 199.0 213.0 254.0 216.0 148.0 309.0 299.0
##  [337] 303.0 308.0 359.0 354.0 336.0 365.0 306.0 228.0 325.0 258.0 160.0  65.0
##  [349] 224.0 180.0 201.0 235.0 208.0 655.0 353.0 341.0 340.0 296.0 388.0 278.0
##  [361] 328.0 364.0 407.0 271.0 262.0 226.0 290.0 270.0 333.0 528.0 544.0 527.0
##  [373] 564.0 375.0 317.0   2.0 220.0 266.0 281.0 314.0 323.0 345.0 295.0 313.0
##  [385] 382.0 301.0 409.0 406.0 397.0 334.0 286.0 307.0 416.0 400.0 408.0 419.0
##  [397] 339.0 332.0 288.0 350.0 460.0 539.0 423.0 594.0 470.0 516.0 536.0 540.0
##  [409] 549.0 555.0 640.0 610.0 612.0 598.0 499.0 427.0 431.0 445.0 401.0 356.0
##  [421] 483.0 329.0 415.0 386.0 561.0 654.0 393.0 492.0 510.0 501.0 481.0 379.0
##  [433] 373.0 349.0 352.0 322.0 434.0 428.0 418.0 479.0 446.0 421.0 458.0 520.0
##  [445] 466.0 438.0 518.0 478.0 342.0 417.0 471.0 429.0 486.0 469.0 424.0 477.0
##  [457] 368.0 402.0 463.0 447.0 436.0 485.0 521.0 554.0 603.0 502.0 404.0 530.0
##  [469] 948.0 548.0 114.6 131.9 125.3 399.0 341.5 391.0 385.0 601.0 526.0 725.0
##  [481]  26.8 170.1  81.3 381.0 403.0 384.0 435.0 394.0 448.0 441.0 432.0 433.0
##  [493] 378.0 414.0 413.0 376.0  37.8 440.0 363.0 405.0 450.0 439.0 568.0 629.0
##  [505] 609.0 500.0 482.0 437.0 361.0 633.0 634.0 506.0 538.0 355.0 387.0 578.0
##  [517] 497.0 475.0 489.0 464.0 472.0 465.0 137.6 570.0 533.0 585.0 844.0 721.0
##  [529] 845.0 862.0 637.0 412.0 812.0 874.0 504.0 456.0 588.0 552.0 514.0 474.0
##  [541] 488.0 420.0 398.0  36.9  64.7   6.4  92.8  33.9 468.0 630.0 453.0 473.0
##  [553] 529.0 410.0 647.0 557.0 537.0 571.0 580.0 558.0 550.0 590.0 604.0 523.0
##  [565] 513.0 582.0 565.0 884.0 762.0 722.0 693.0 666.0 873.0 638.0 509.0 777.0
##  [577] 625.0 507.0 642.0 827.0 834.0 575.0 449.0 567.0 422.0 444.0 496.0 425.0
##  [589] 542.0 572.0 543.0 498.0 491.0 467.0 484.0 493.0 815.0 773.0 799.0 858.0
##  [601] 754.0 653.0 579.0 595.0 584.0 531.0 451.0 546.0 195.5 169.1  90.4  15.5
##  [613] 522.0 770.0 652.0 487.0 112.4 720.0 793.0 930.0 992.0 747.0 608.0 480.0
##  [625] 976.0 757.0  79.8 714.0 641.0 503.0 562.0 665.0 702.0 691.0 688.0 674.0
##  [637] 683.0 657.0 663.0 686.0 685.0 677.0 660.0 632.0 692.0 576.0 454.0 541.0
##  [649] 635.0 706.0 563.0 517.0 581.0 933.0 457.0 597.0 675.0 775.0 559.0 631.0
##  [661] 895.0 801.0 112.2  99.5  91.5  96.6 152.7  26.9   7.9  21.1  15.7 171.2
##  [673]   8.4 131.5   9.6 623.0 117.9 731.0 842.0 904.0 905.0 782.0 776.0 611.0
##  [685] 646.0 556.0  91.9  60.9  14.5   5.6 138.5  61.4  49.3 545.0 490.0 676.0
```

```
##  [697]  36.5  17.1  30.4  42.4  45.5 645.0 650.0 461.0 534.0 515.0 671.0 734.0
##  [709] 771.0 737.0 699.0 717.0 690.0 673.0 644.0 532.0 864.0 678.0 792.0 524.0
##  [721] 811.0 828.0 553.0 602.0 495.0 551.0 589.0 599.0 560.0 566.0 459.0 600.0
##  [733] 319.8  23.7  42.3  45.4 201.1 114.1 147.9  22.2 659.0 672.0 574.0 511.0
##  [745] 794.0 185.9 164.9  16.3 622.0 593.0 669.0 649.0 816.0 680.0 512.0 606.0
##  [757] 583.0 547.0  84.1 525.0 643.0 573.0 596.0 626.0 807.0 876.0 787.0 758.0
##  [769] 708.0 888.0 891.0 830.0 848.0 800.0 586.0 710.0 715.0 701.0 535.0 618.0
##  [781] 605.0 613.0 607.0 695.0 703.0 591.0 619.0 694.0 955.0 907.0 915.0 796.0
##  [793] 825.0 987.0 743.0 519.0 627.0 616.0 505.0 814.0 322.1  23.8 145.4 179.9
##  [805]  71.4 121.7 305.4 157.7 136.2 117.6 124.4  57.8  62.9 961.0 661.0 656.0
##  [817] 658.0 592.0 614.0 802.0 744.0 636.0 711.0 806.0 769.0 906.0 617.0 624.0
##  [829] 577.0 651.0 917.0 820.0 947.0 983.0 957.0 639.0 804.0 697.0 849.0 195.2
##  [841] 186.6  88.9 126.5 134.1  40.3  15.9  38.6  58.8 123.7 142.6 116.4 112.3
##  [853] 707.0 817.0 890.0 941.0  47.3  53.7 733.0 741.0 705.0 620.0 689.0 826.0
##  [865] 772.0 986.0 883.0 681.0 664.0 700.0 728.0 805.0 724.0 679.0 994.0 738.0
##  [877] 835.0 887.0 193.3  42.5  99.9  53.5  34.2 993.0 180.2  84.7  55.1 991.0
##  [889] 973.0  60.3  50.3  15.8  11.8 104.4   9.8 107.4 111.5 125.6 748.0 569.0
##  [901] 750.0 648.0 740.0 615.0 687.0  32.2  85.6  23.6 139.4 146.1 117.2 100.9
##  [913]  30.7 742.0 668.0 899.0  79.3  16.4   9.5 126.3 169.7 153.3  93.3  81.6
##  [925]  55.3  35.9  70.6 103.9  72.5 214.3 909.0 995.0 939.0 878.0 766.0 778.0
##  [937] 764.0  41.2 208.1  67.4  99.4  92.4  14.7  52.8  28.4 903.0 127.2  38.1
##  [949] 914.0 912.0  41.8  84.4  42.7  26.6  33.5  59.3 106.1 194.6 118.8  51.8
##  [961] 999.0 790.0 857.0  87.8  20.8  54.5 823.0 951.0 786.0 789.0 839.0 667.0
##  [973] 894.0  28.1 175.1 135.2  30.1 108.9 159.4 172.6  45.6   8.7   7.7   5.4
##  [985]   8.2  67.9  68.9  68.3 121.5 198.3 144.5 242.7 229.6 180.6 169.5 156.4
##  [997] 135.9 108.6  74.1  28.8  31.7  99.2  78.4 102.8 215.2 988.0 927.0 870.0
## [1009] 781.0 893.0 863.0 829.0 107.5 923.0 335.1 158.8 170.3 161.1  81.4  85.2
## [1021]  77.5  11.5 712.0 783.0 751.0 736.0 785.0 621.0 784.0 902.0  24.5 732.0
## [1033] 670.0 768.0 886.0 730.0 836.0 145.5  89.1 176.4 133.1  28.9 682.0 150.4
## [1045] 931.0 704.0 929.0 745.0 952.0 950.0
```

```
##   [1]   300   400   500   600   700   800   900  1000  1200  1100  1300  1399
##  [13]  1500  1700  1899  2200  2399  2500  2799  2100  2000  2299  2600  1800
##  [25]  1600  2899  2700  3100  3200  4400  4000  3500  3700  4200  3799  4099
##  [37]  5000  5599  5700  4599  3299  3399  4900  5200  3600  3000   200  1400
##  [49]   100  1900  2300  2400  2800  3300  2900  3400  5400  5800  5900  5300
##  [61]  4500  3800  5100  5600  4800  4600  6300  6400  4300  4100  4700  6000
##  [73]  6700  7300  7000  7100  7700  8300  8900  8100  7600  6800  3900  6500
##  [85]  7400  6900  6200  5500  6600  6100  7800  7200  7900  8000  9000  7500
##  [97]  8800  9300  9100  8400  9600 10000  9500  9900  9400  8500  8200  8600
## [109]  8700  9700  9200  4799  3899  5099  5299  6299  9800   350   950  1150
## [121]   150  4299  6599  6799  6099  7299  8199  8099  7099  5799  7599  8599
```

```
# Review the first 6 rows
head(data)
```

```
##                  date   CO PM2.5 PM10 SO2 NO2 O3 TEMP   PRES  DEWP  wd WSPM
## 1 2013-03-02 00:00:00  500    22   24  24  44 44 -0.4 1031.0 -17.6 ENE  1.4
## 2 2013-03-02 06:00:00  500     4   10  28  46 39 -2.5 1029.6 -17.7  NW  0.7
## 3 2013-03-02 12:00:00  900    26   30  25  76 22  2.7 1027.3 -16.4 WSW  2.7
## 4 2013-03-02 18:00:00 1100    54   79  50  80 24  1.2 1023.2 -12.3   W  0.9
## 5 2013-03-04 00:00:00 1300    42   83  51  86  4  7.7 1015.7 -11.1   N  2.6
## 6 2013-03-04 06:00:00  500    13   32  24  65 22  3.8 1018.9 -11.4   N  2.3
##        station
## 1 Aotizhongxin
```

```
## 2 Aotizhongxin
## 3 Aotizhongxin
## 4 Aotizhongxin
## 5 Aotizhongxin
## 6 Aotizhongxin
```

We visualize the data set with the changes made (Filtered for performance reasons: half days and only 3 rows per day)

```
# Review the previous changes in a transposed version to see more data
glimpse(data)
```

```
## Rows: 31,530
## Columns: 13
## $ date    <dttm> 2013-03-02 00:00:00, 2013-03-02 06:00:00, 2013-03-02 12:00...
## $ CO      <int> 500, 500, 900, 1100, 1300, 500, 400, 600, 4000, 2600, 2299,...
## $ PM2.5   <dbl> 22, 4, 26, 54, 42, 13, 13, 25, 284, 186, 192, 239, 344, 310...
## $ PM10    <dbl> 24, 10, 30, 79, 83, 32, 28, 44, 315, 194, 219, 291, 389, 32...
## $ SO2     <dbl> 24, 28, 25, 50, 51, 24, 12, 13, 133, 94, 124, 142, 107, 45,...
## $ NO2     <dbl> 44, 46, 76, 80, 86, 65, 14, 43, 174, 131, 113, 177, 195, 14...
## $ O3      <dbl> 44, 39, 22, 24, 4, 22, 77, 75, 28, 22, 22, 17, 84, 84, 84, ...
## $ TEMP    <dbl> -0.4, -2.5, 2.7, 1.2, 7.7, 3.8, 14.2, 11.6, 4.9, 4.5, 11.1,...
## $ PRES    <dbl> 1031.0, 1029.6, 1027.3, 1023.2, 1015.7, 1018.9, 1018.9, 101...
## $ DEWP    <dbl> -17.6, -17.7, -16.4, -12.3, -11.1, -11.4, -13.9, -11.3, -6....
## $ wd      <fct> ENE, NW, WSW, W, N, N, N, SSW, NE, NE, ENE, WSW, ESE, WSW, ...
## $ WSPM    <dbl> 1.4, 0.7, 2.7, 0.9, 2.6, 2.3, 2.7, 3.4, 1.2, 2.0, 0.4, 0.9,...
## $ station <fct> Aotizhongxin, Aotizhongxin, Aotizhongxin, Aotizhongxin, Aot...
```

Finally, we run the display of the attribute statistics. It is important to note that the mean and median values, in every attribute, are not very far from each other, therefore, there are not many scattered data.

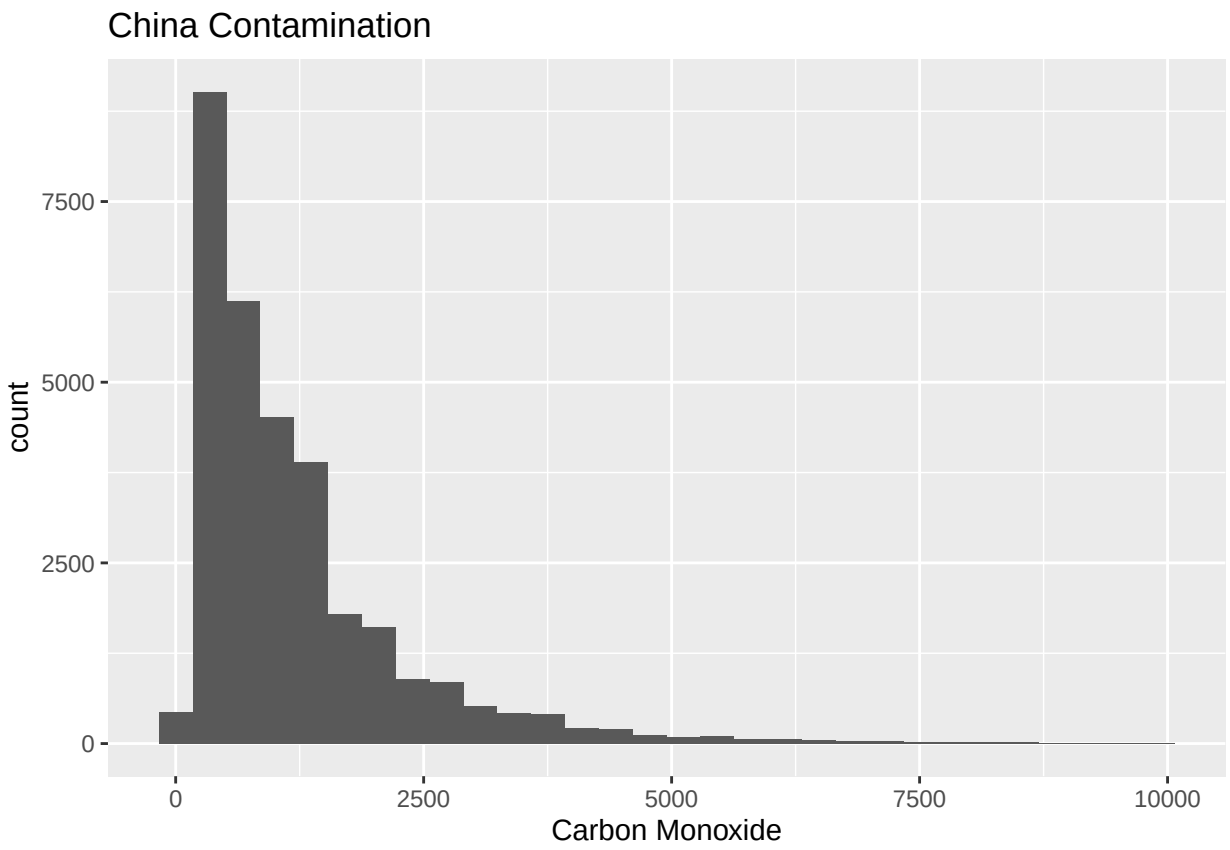
```
# Review the statistics of each attribute
summary(data)
```

```
##       date                          CO            PM2.5             PM10
##  Min.   :2013-03-02 00:00:00   Min.   :  100   Min.   :  3.00   Min.   :  2
##  1st Qu.:2014-04-04 00:00:00   1st Qu.:  500   1st Qu.: 19.00   1st Qu.: 35
##  Median :2015-03-22 06:00:00   Median :  900   Median : 53.00   Median : 80
##  Mean   :2015-03-16 16:57:45   Mean   : 1213   Mean   : 78.48   Mean   :103
##  3rd Qu.:2016-03-08 00:00:00   3rd Qu.: 1500   3rd Qu.:108.00   3rd Qu.:142
##  Max.   :2017-02-28 18:00:00   Max.   :10000   Max.   :720.00   Max.   :986
##
##       SO2              NO2                O3                TEMP
##  Min.   :  1.0   Min.   :  2.00   Min.   :   0.2142   Min.   :-18.90
##  1st Qu.:  2.0   1st Qu.: 22.00   1st Qu.:  11.0000   1st Qu.:  3.00
##  Median :  7.0   Median : 42.00   Median :  47.0000   Median : 14.00
##  Mean   : 15.3   Mean   : 49.73   Mean   :  58.4333   Mean   : 13.33
##  3rd Qu.: 19.0   3rd Qu.: 70.00   3rd Qu.:  84.0000   3rd Qu.: 22.80
##  Max.   :224.0   Max.   :271.00   Max.   :1071.0000   Max.   : 37.10
##
##       PRES             DEWP              wd             WSPM
##  Min.   : 982.8   Min.   :-32.30   NE     : 3290   Min.   :0.000
##  1st Qu.:1002.4   1st Qu.: -9.10   NW     : 2506   1st Qu.:0.900
##  Median :1010.5   Median :  2.70   ENE    : 2491   Median :1.400
##  Mean   :1010.8   Mean   :  2.35   E      : 2305   Mean   :1.718
##  3rd Qu.:1019.0   3rd Qu.: 15.30   SW     : 2301   3rd Qu.:2.200
##  Max.   :1041.4   Max.   : 28.80   N      : 2148   Max.   :9.700
```

```
##                                   (Other):16489
##           station
##  Nongzhanguan : 2713
##  Tiantan      : 2713
##  Changping    : 2700
##  Wanshouxigong: 2675
##  Gucheng      : 2659
##  Guanyuan     : 2648
##  (Other)      :15422
```

We can verify in the following histogram in this case with the attribute CO

```
# Histogram of Carbon Monoxide
data %>% ggplot(aes(CO)) + geom_histogram() +
         labs(title = "China Contamination", x = "Carbon Monoxide")
```



2.2 Exploratory Data Analisys

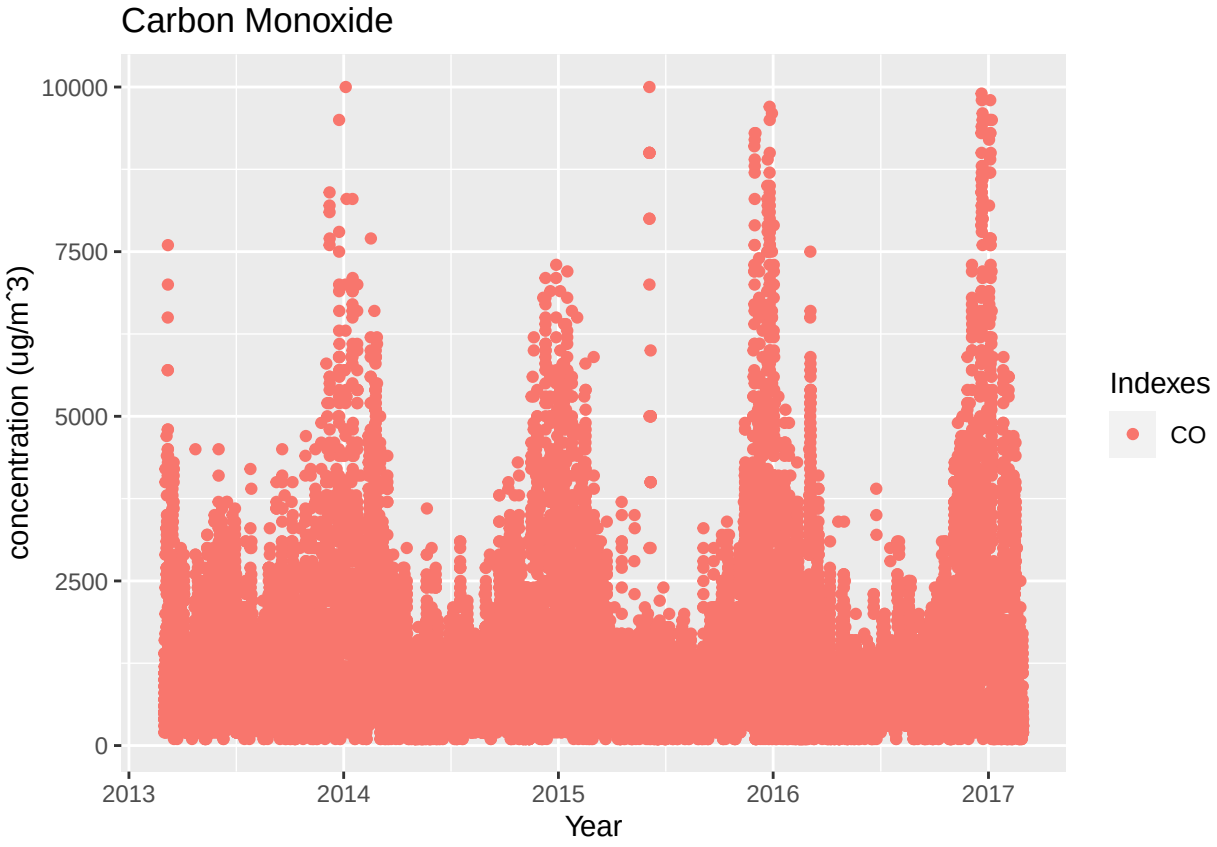
To better understand the dataset i will perform a Exploratory Data Analysis (EDA) which will facilitate the generation of the models later.

2.2.1 Data Throught Time

First, the amount of pollutants over time is analyzed (2013-2017). The amount of carbon monoxide is shown in the following graph.

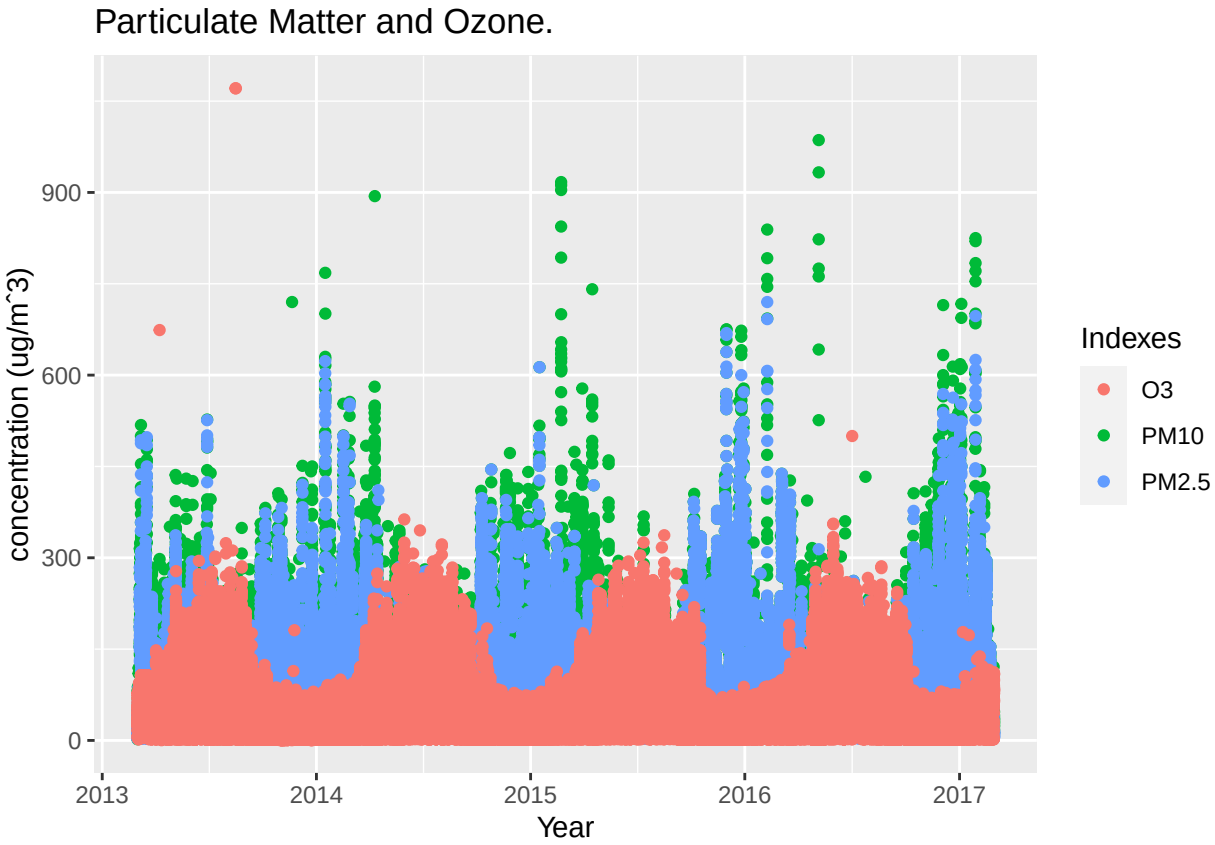
```
# Carbon Monoxide
data_grph <- data %>%
```

```
  select(date,CO) %>%
  gather(key = "Indexes", value = "value", -date)

data_grph %>%
  ggplot(aes(x=date,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Carbon Monoxide", x = "Year",
       y = "concentration (ug/m^3)")
```



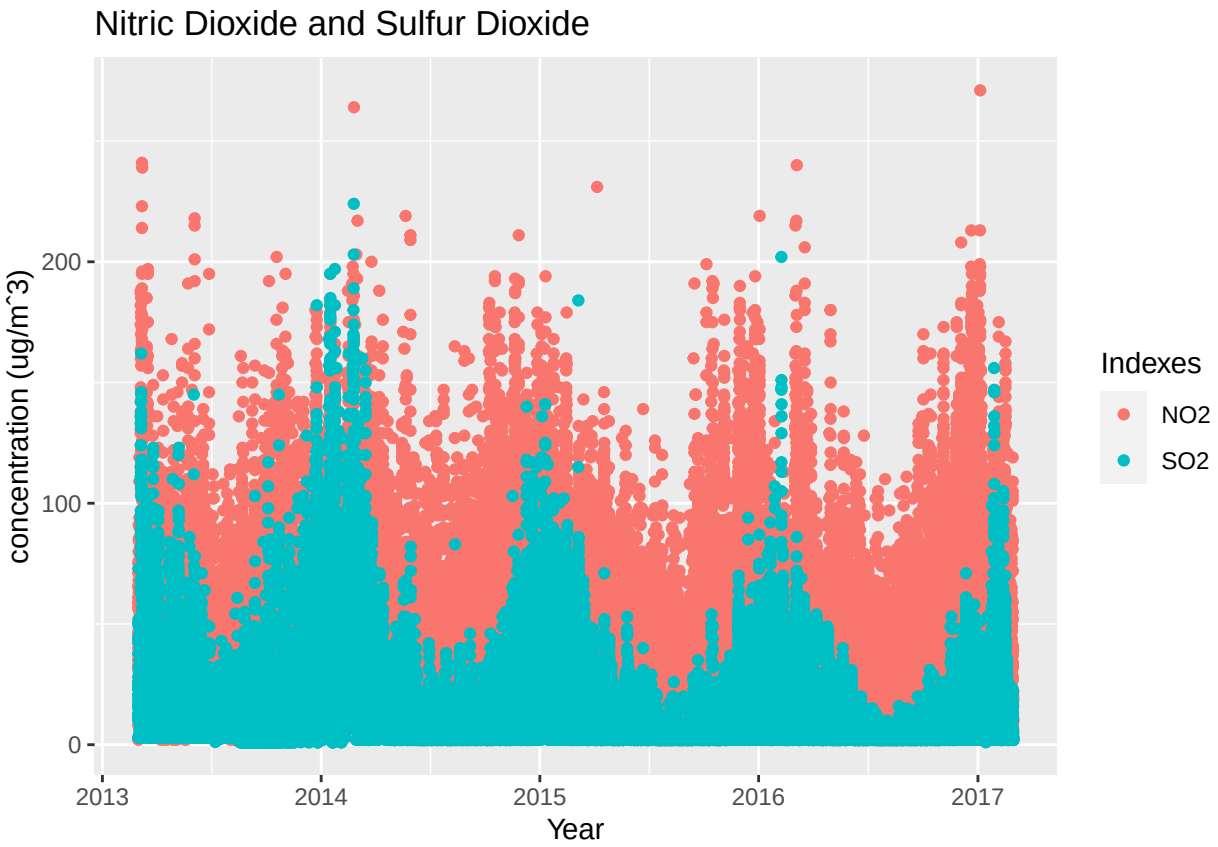
Below is the Particulate Matter and Ozone.

```
# Particulate Matter and Ozono
data_grph <- data %>%
  select(date,PM10,PM2.5,O3) %>%
  gather(key = "Indexes", value = "value", -date)

data_grph %>%
  ggplot(aes(x=date,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Particulate Matter and Ozone.", x = "Year",
       y = "concentration (ug/m^3)")
```



Finally, Nitric Dioxide and Sulfur Dioxide is shown.

```
# Nitrogen Dioxide and Sulfur Dioxide
data_grph <- data %>%
  select(date, NO2, SO2) %>%
  gather(key = "Indexes", value = "value", -date)

data_grph %>%
  ggplot(aes(x=date,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Nitric Dioxide and Sulfur Dioxide", x = "Year",
       y = "concentration (ug/m^3)")
```

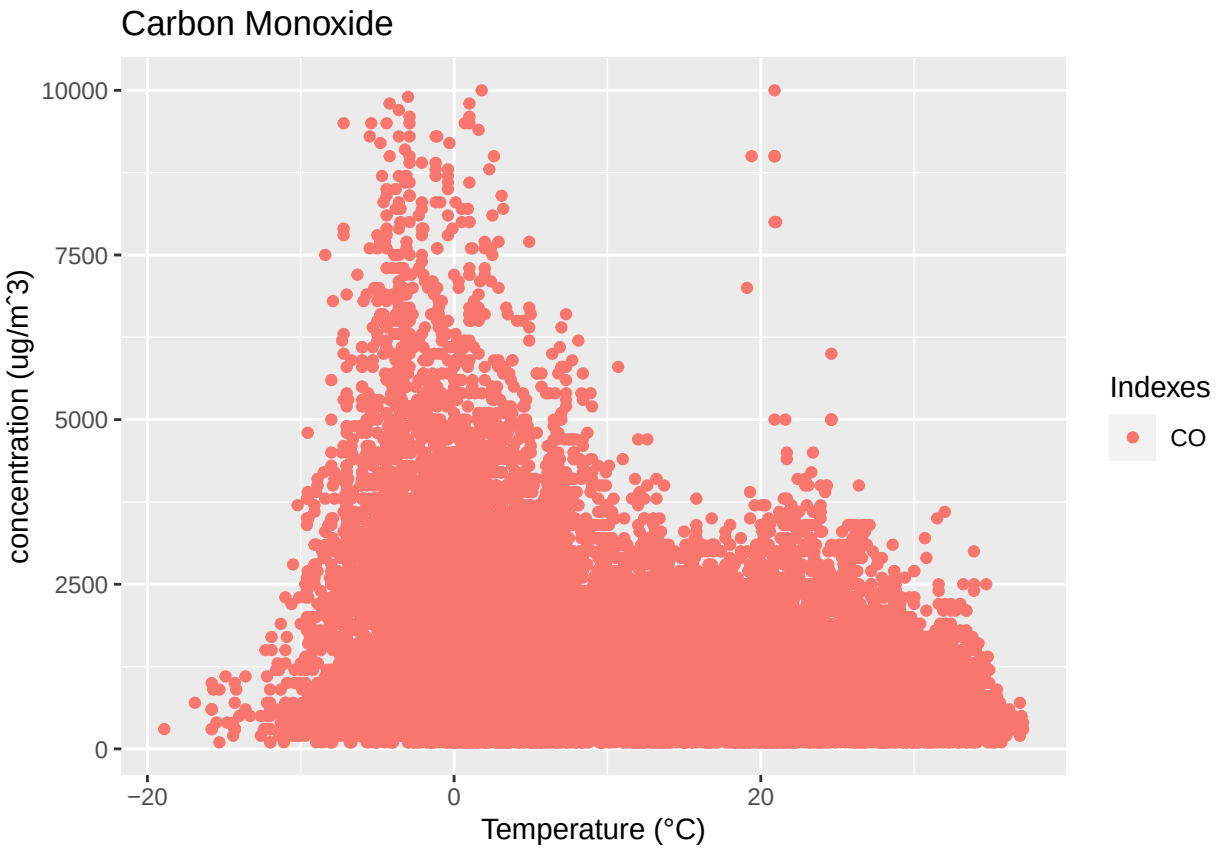


It is observed in the past graphs that the concentration levels tend to increase throughout the years, but during some months (different for each contaminant) will be maximus and minimous

2.2.2 Data Throught Temperature

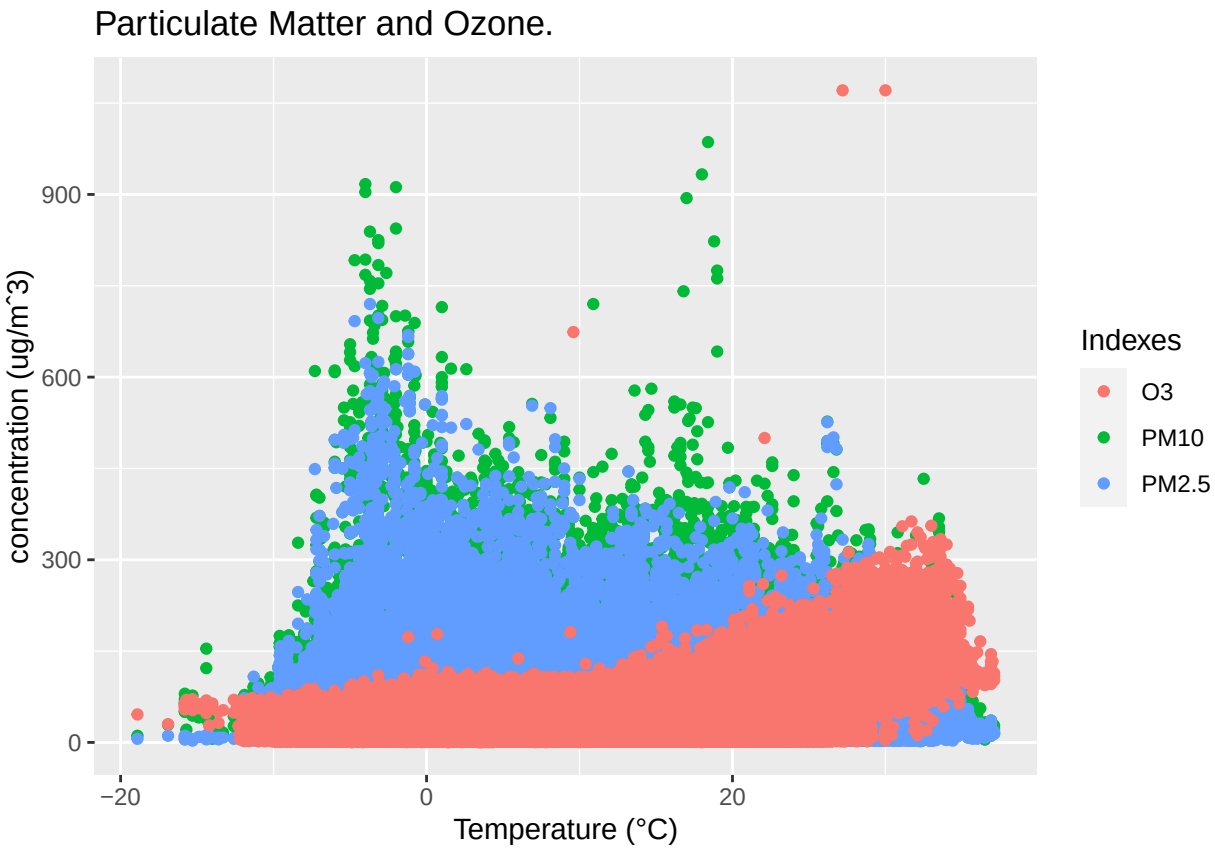
Then, the amount of pollutants over temperature is analyzed (-20°C to 40 °C). The amount of carbon monoxide is shown in the following graph.

```
# Carbon Monoxide
data_grph <- data %>%
  select(TEMP,CO) %>%
  gather(key = "Indexes", value = "value", -TEMP)

data_grph %>%
  ggplot(aes(x=TEMP,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Carbon Monoxide", x = "Temperature (°C)",
       y = "concentration (ug/m^3)")
```



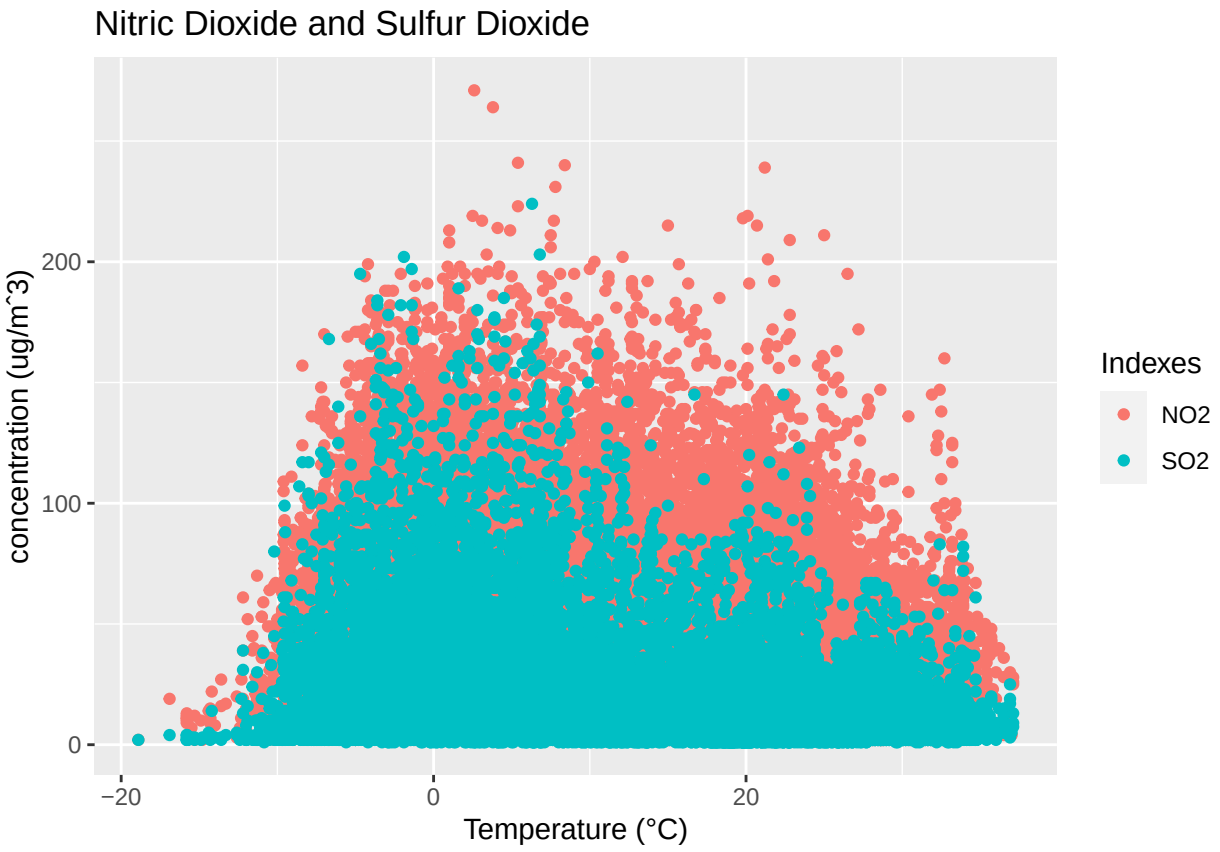
Below is the Particulate Matter and Ozone.

```
# Particulate Matter and Ozono
data_grph <- data %>%
  select(TEMP,PM10,PM2.5,O3) %>%
  gather(key = "Indexes", value = "value", -TEMP)

data_grph %>%
  ggplot(aes(x=TEMP,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Particulate Matter and Ozone.", x = "Temperature (°C)",
       y = "concentration (ug/m^3)")
```



Finally, Nitric Dioxide and Sulfur Dioxide is shown.

```
# Nitrogen Dioxide and Sulfur Dioxide
data_grph <- data %>%
  select(TEMP, NO2, SO2) %>%
  gather(key = "Indexes", value = "value", -TEMP)

data_grph %>%
  ggplot(aes(x=TEMP,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Nitric Dioxide and Sulfur Dioxide", x = "Temperature (°C)",
       y = "concentration (ug/m^3)")
```

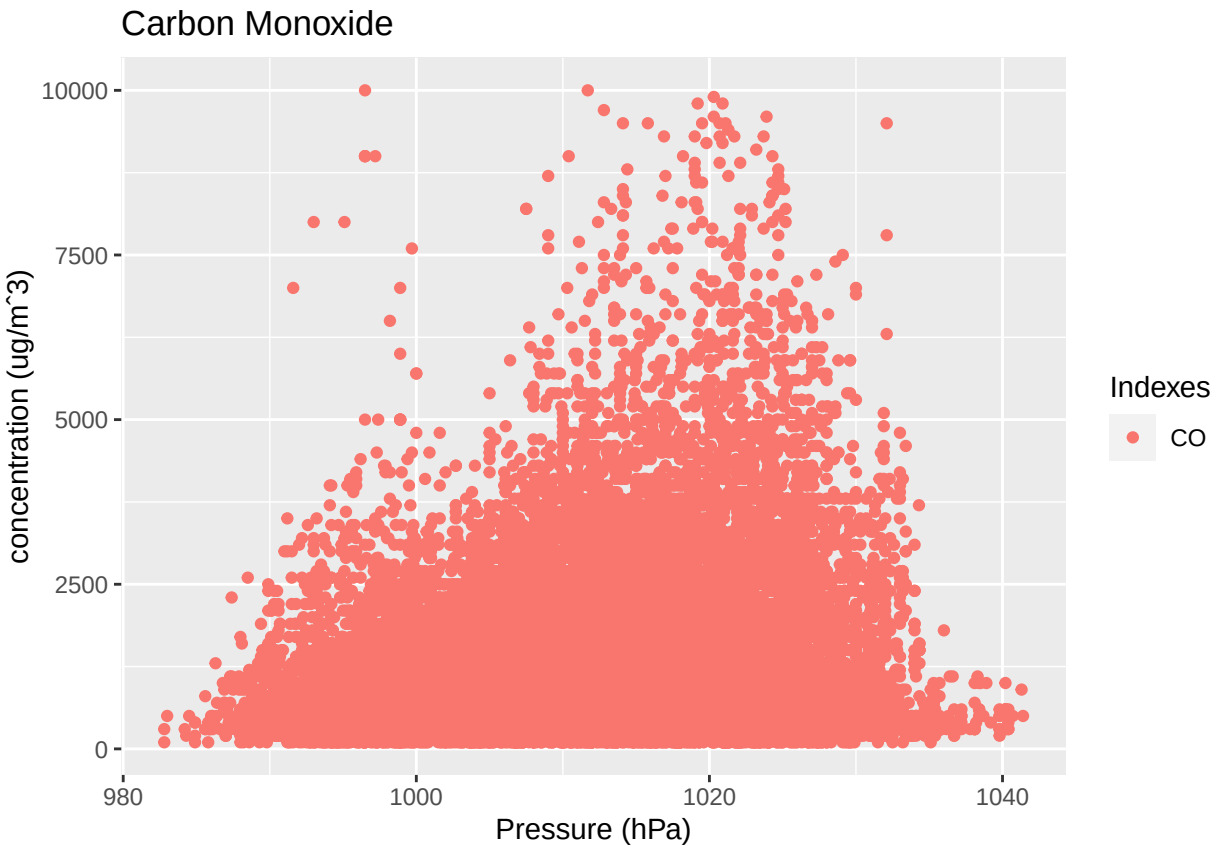


It is observed in the past graphs that the concentration levels tend to increase for the Particulate Matter over 0°C and also increase for Ozono over 35°C.

2.2.3 Data Throught Pressure

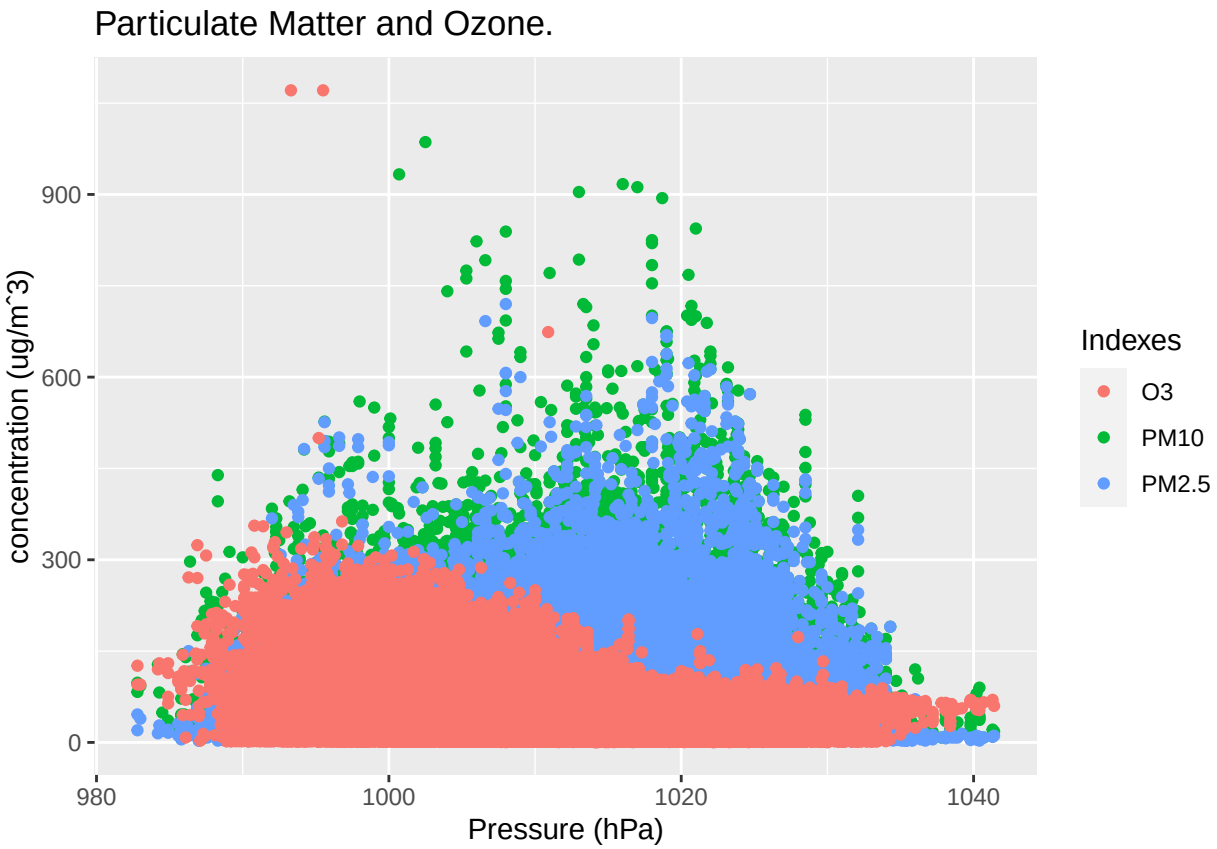
Then, the amount of pollutants over pressure is analyzed (982 hPa to 1042 hPa). The amount of carbon monoxide is shown in the following graph.

```
# Carbon Monoxide
data_grph <- data %>%
  select(PRES,CO) %>%
  gather(key = "Indexes", value = "value", -PRES)

data_grph %>%
  ggplot(aes(x=PRES,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Carbon Monoxide", x = "Pressure (hPa)",
       y = "concentration (ug/m^3)")
```



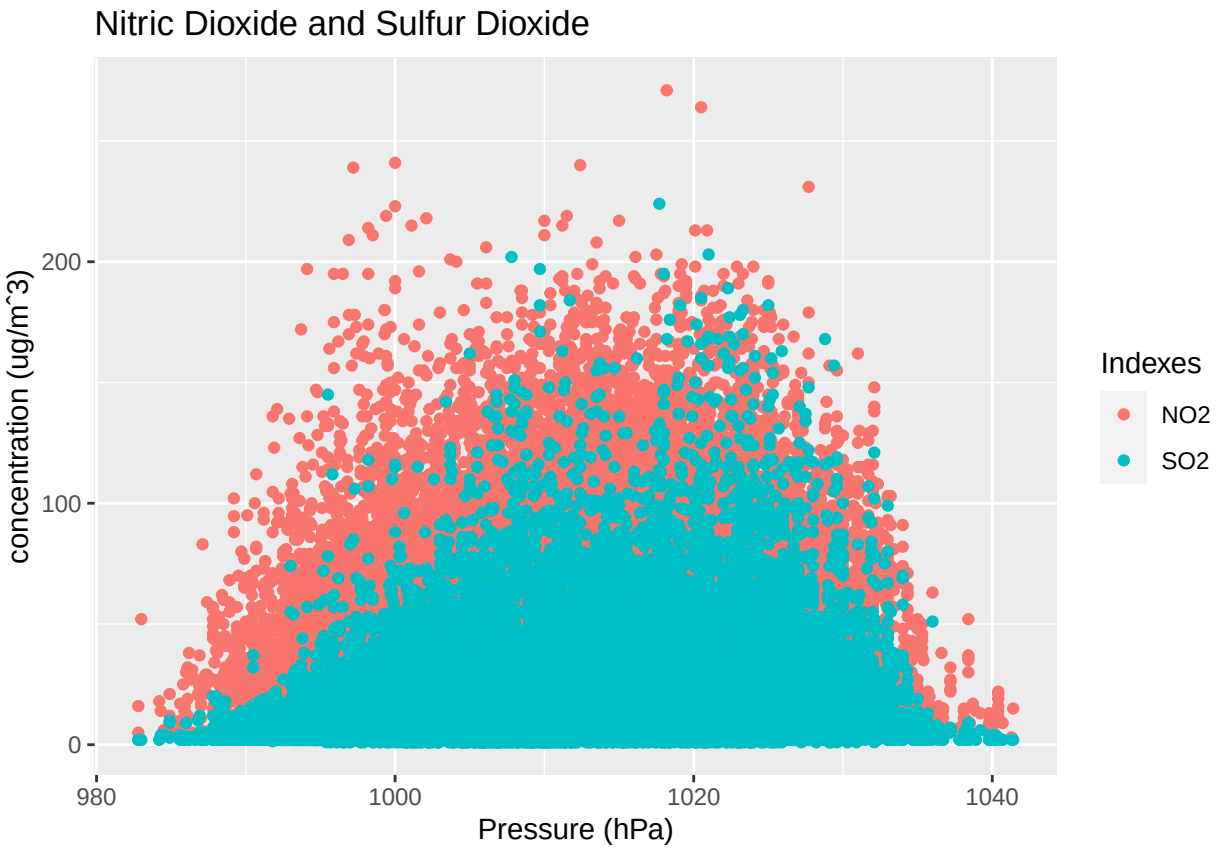
Below is the Particulate Matter and Ozone.

```
# Particulate Matter and Ozono
data_grph <- data %>%
  select(PRES,PM10,PM2.5,O3) %>%
  gather(key = "Indexes", value = "value", -PRES)

data_grph %>%
  ggplot(aes(x=PRES,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Particulate Matter and Ozone.", x = "Pressure (hPa)",
       y = "concentration (ug/m^3)")
```



Finally, Nitric Dioxide and Sulfur Dioxide is shown.

```
# Nitrogen Dioxide and Sulfur Dioxide
data_grph <- data %>%
  select(PRES, NO2, SO2) %>%
  gather(key = "Indexes", value = "value", -PRES)

data_grph %>%
  ggplot(aes(x=PRES,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Nitric Dioxide and Sulfur Dioxide", x = "Pressure (hPa)",
       y = "concentration (ug/m^3)")
```

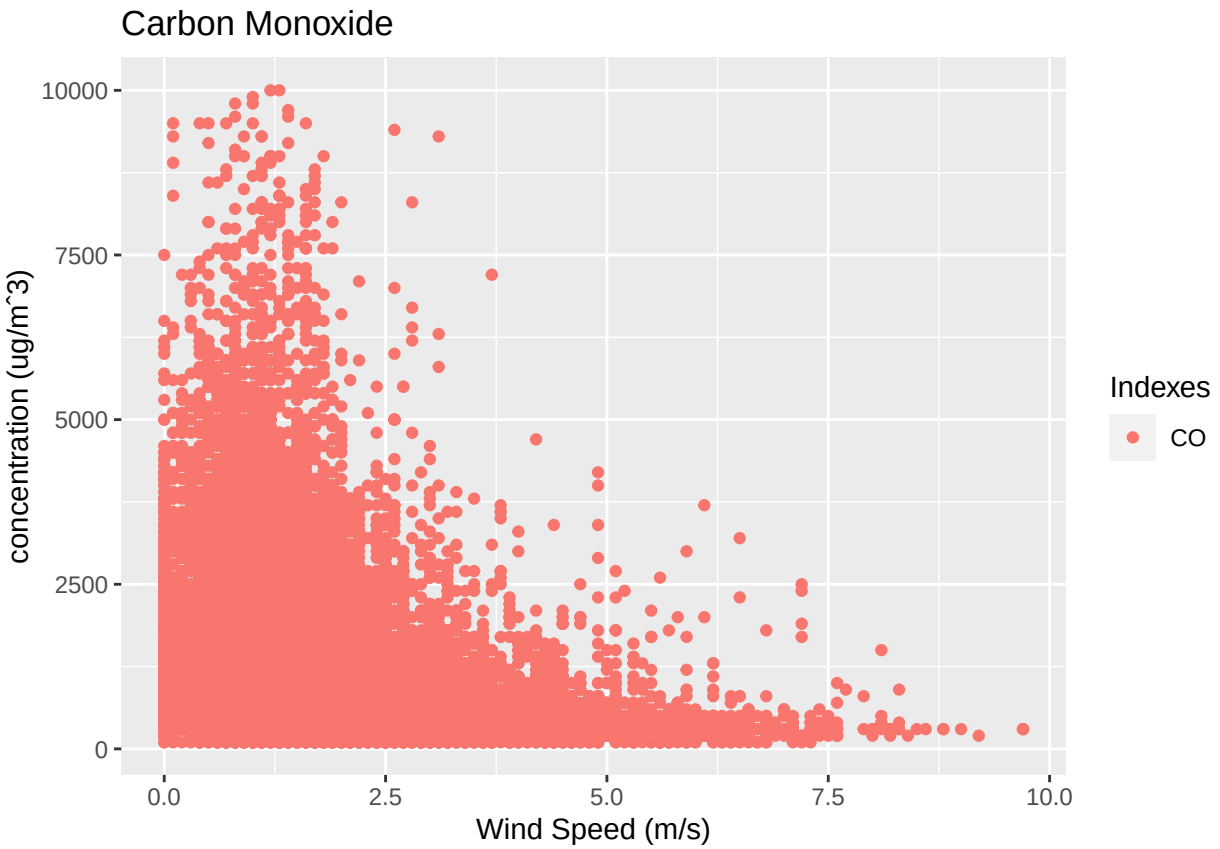


It is observed in the past graphs that the concentration levels tend to increase between 1000 and 1020 hPa. In other pressures a low concentration is registered.

2.2.4 Data Throught Wind Speed

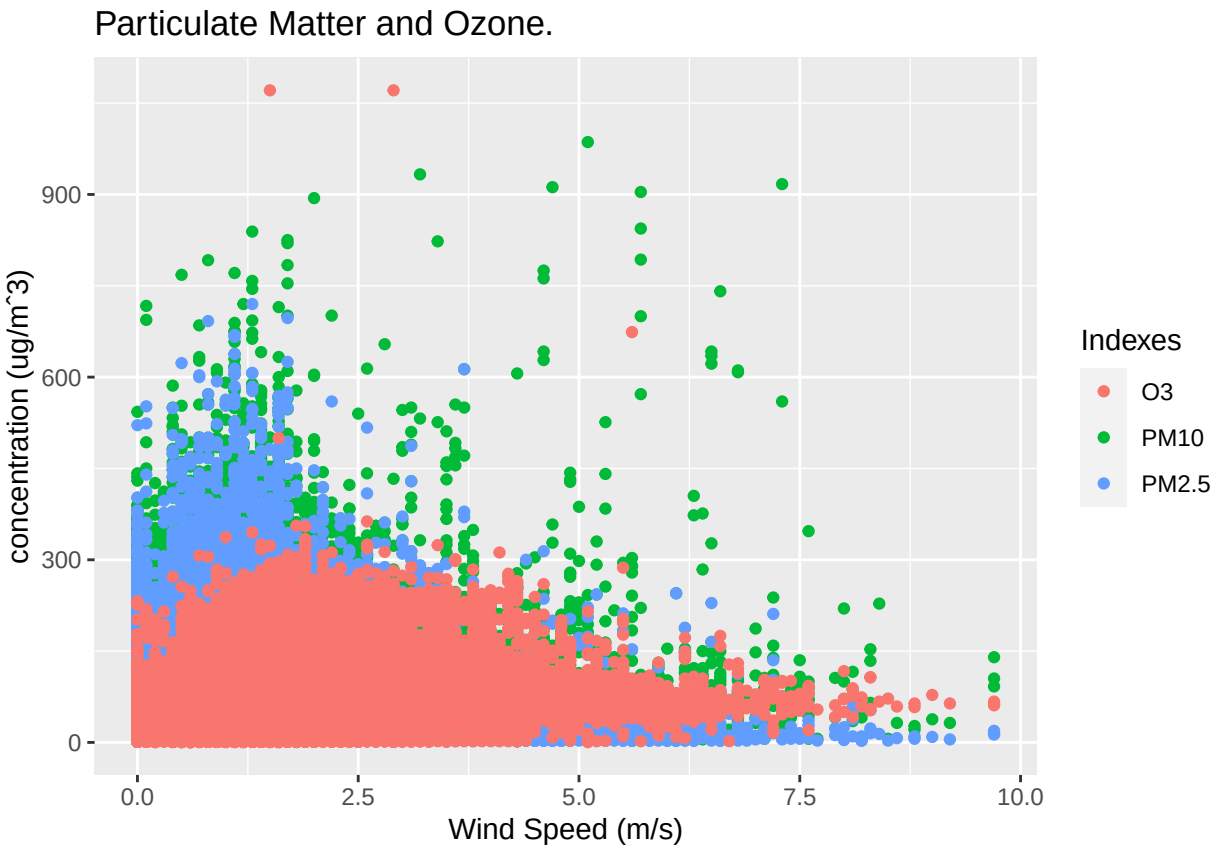
Finally, the amount of pollutants over wind speed is analyzed (0 m/s to 13.2 m/s). The amount of carbon monoxide is shown in the following graph.

```
# Carbon Monoxide
data_grph <- data %>%
  select(WSPM,CO) %>%
  gather(key = "Indexes", value = "value", -WSPM)

data_grph %>%
  ggplot(aes(x=WSPM,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Carbon Monoxide", x = "Wind Speed (m/s)",
       y = "concentration (ug/m^3)")
```



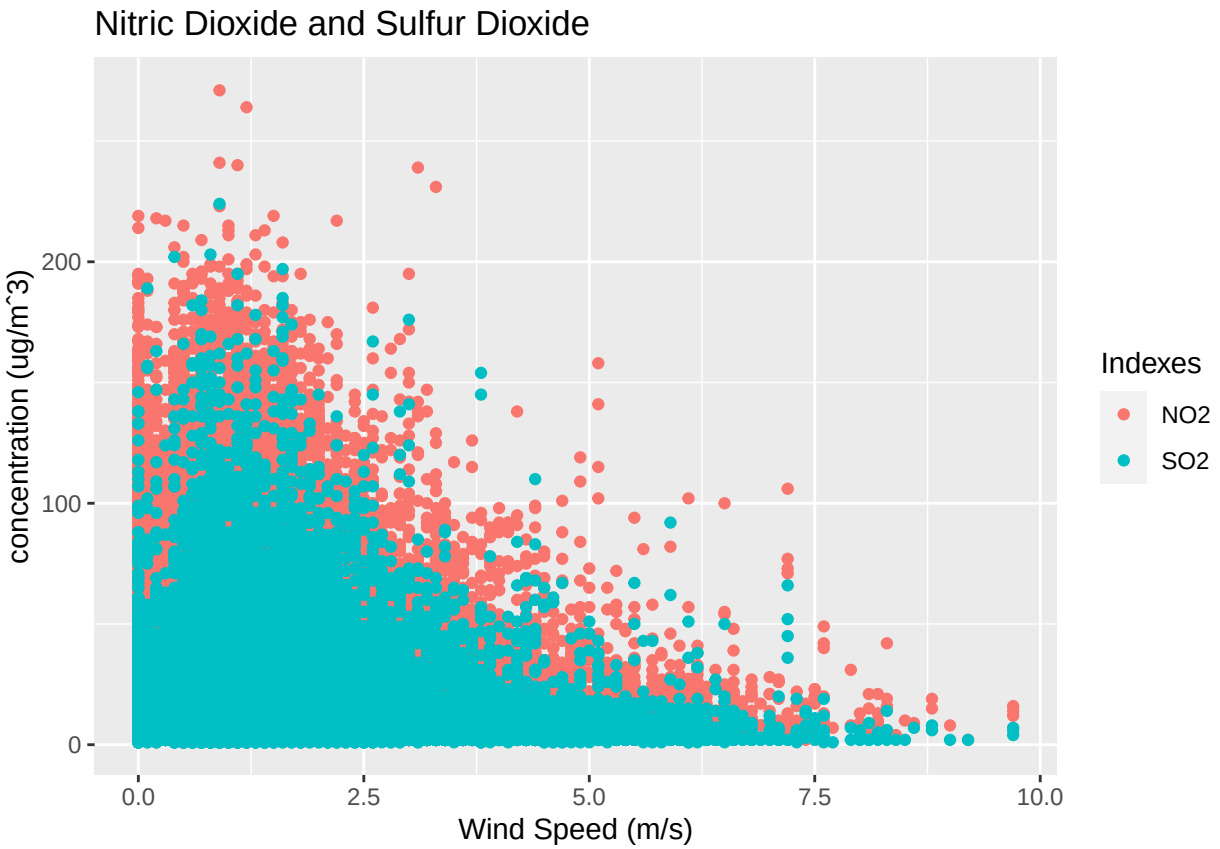
Below is the Particulate Matter and Ozone.

```
# Particulate Matter and Ozono
data_grph <- data %>%
  select(WSPM,PM10,PM2.5,O3) %>%
  gather(key = "Indexes", value = "value", -WSPM)

data_grph %>%
  ggplot(aes(x=WSPM,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Particulate Matter and Ozone.", x = "Wind Speed (m/s)",
       y = "concentration (ug/m^3)")
```



Finally, Nitric Dioxide and Sulfur Dioxide is shown.

```
# Nitrogen Dioxide and Sulfur Dioxide
data_grph <- data %>%
  select(WSPM, NO2, SO2) %>%
  gather(key = "Indexes", value = "value", -WSPM)

data_grph %>%
  ggplot(aes(x=WSPM,y=value)) +
  geom_point(aes(color = Indexes)) +
  labs(title = "Nitric Dioxide and Sulfur Dioxide", x = "Wind Speed (m/s)",
       y = "concentration (ug/m^3)")
```

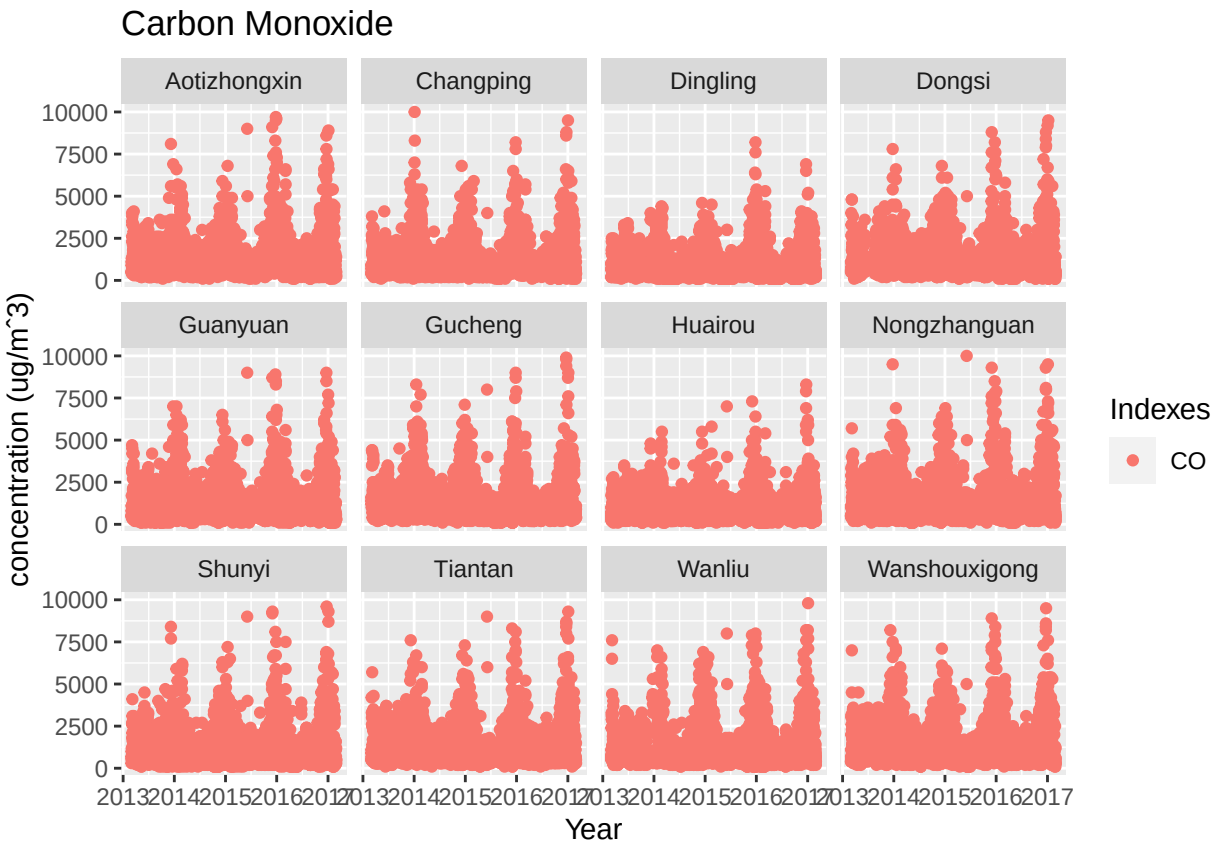


It is observed in the past graphs that the concentration levels tend to decrease when the Wind Speed increase. Since the wind carries the pollutants out the area, drastically reducing the presence of these.

2.2.5 Data Throught Station

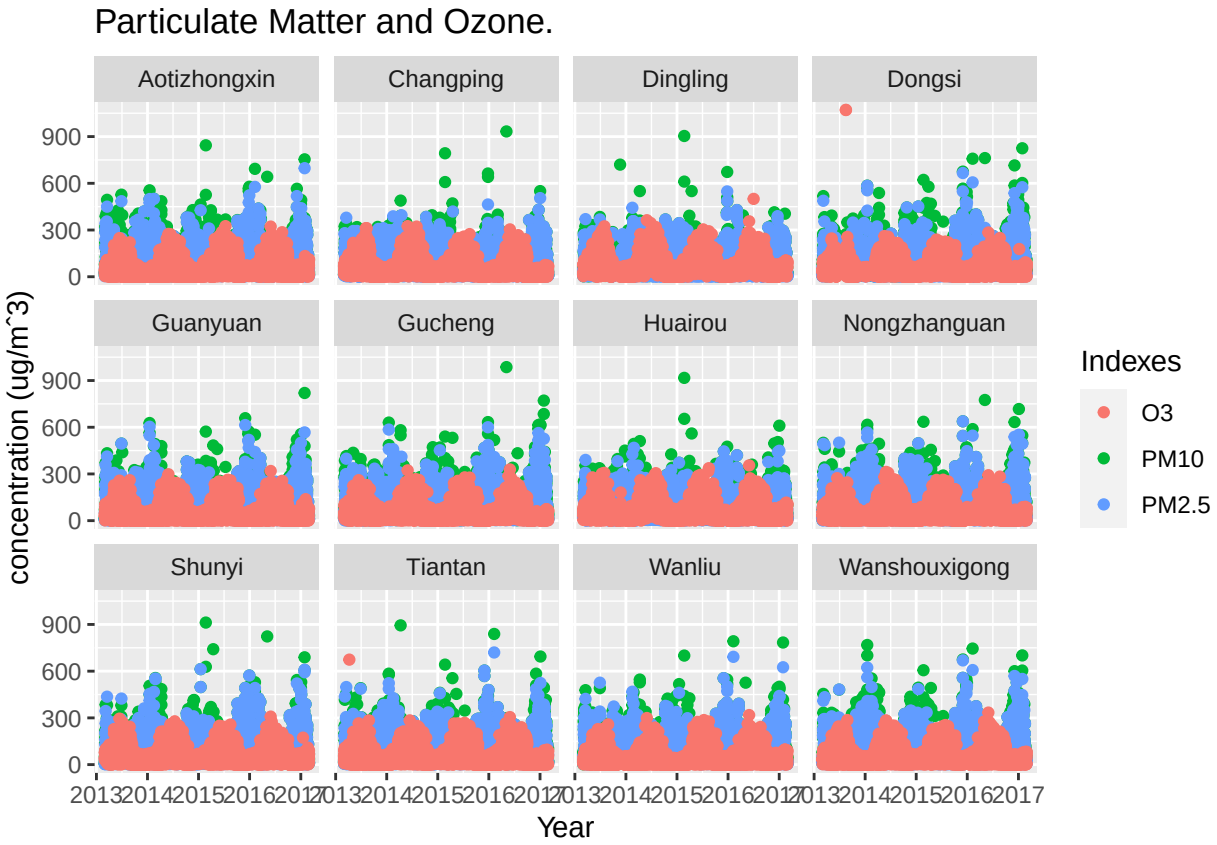
Then, the amount of pollutants over station is analyzed (12 different). The amount of carbon monoxide is shown in the following graph.

```
# Carbon Monoxide
data_grph <- data %>%
  select(date,station,CO) %>%
  gather(key = "Indexes", value = "value", "CO")

data_grph %>%
  ggplot(aes(x=date,y=value)) +
  geom_point(aes(color = Indexes)) +
  facet_wrap(~ station) +
  labs(title = "Carbon Monoxide", x = "Year",
       y = "concentration (ug/m^3)")
```



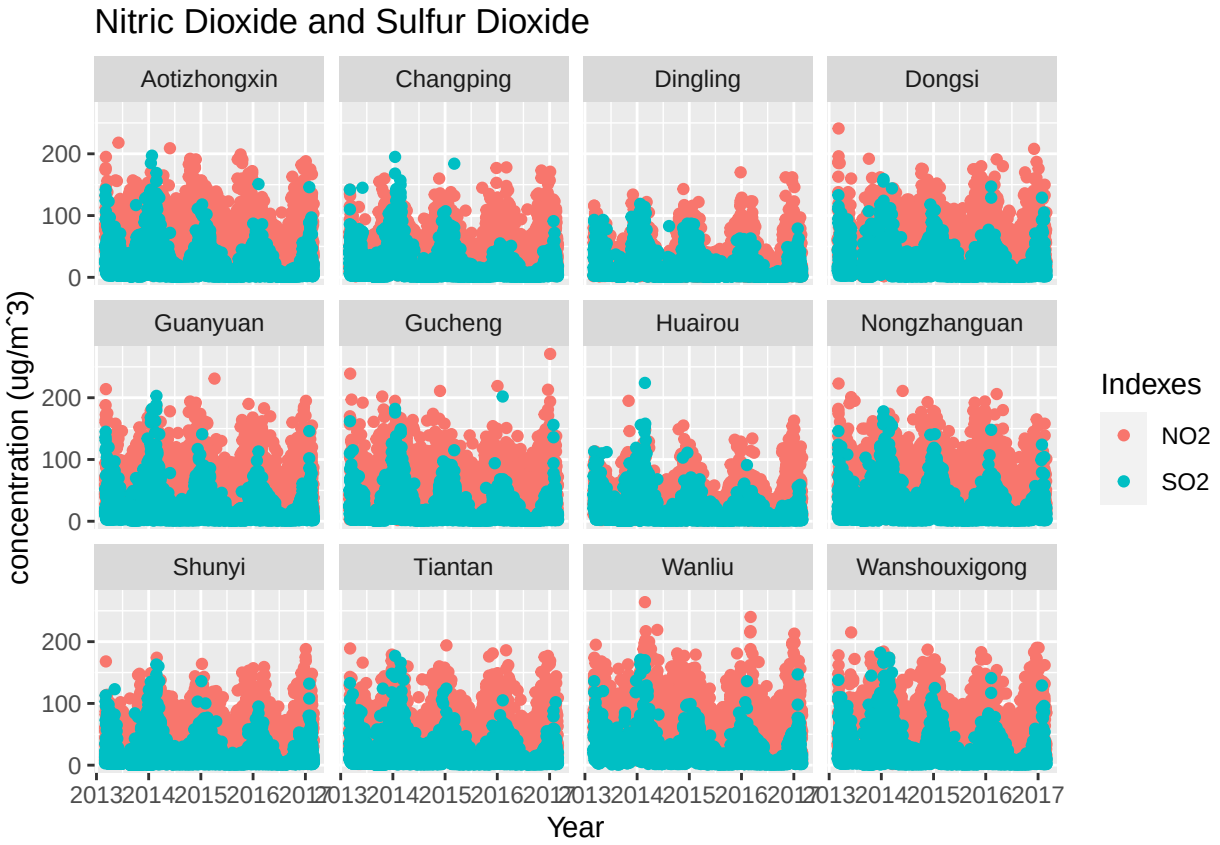
Below is the Particulate Matter and Ozone.

```
# Particulate Matter and Ozono
data_grph <- data %>%
  select(date,station,PM10,PM2.5,O3) %>%
  gather(key = "Indexes", value = "value", "PM10":"O3")

data_grph %>%
  ggplot(aes(x=date,y=value)) +
  geom_point(aes(color = Indexes)) +
  facet_wrap(~ station) +
  labs(title = "Particulate Matter and Ozone.", x = "Year",
       y = "concentration (ug/m^3)")
```



Finally, Nitric Dioxide and Sulfur Dioxide is shown.

```
# Nitrogen Dioxide and Sulfur Dioxide
data_grph <- data %>%
  select(date,station, NO2, SO2) %>%
  gather(key = "Indexes", value = "value", "NO2":"SO2")

data_grph %>%
  ggplot(aes(x=date,y=value)) +
  geom_point(aes(color = Indexes)) +
  facet_wrap(~ station) +
  labs(title = "Nitric Dioxide and Sulfur Dioxide", x = "Year",
       y = "concentration (ug/m^3)")
```



It is observed in the past graphs that the concentration levels tend to increase throughout the years, but during some months (different for each contaminant) will be maximus and minimous. And the most important thing is that the pollution of each station presents very similar values to each other in the same period of time.

So, based on the above, it will be modeled based on Carbon Monoxide (CO) because it presents higher concentration averages on the dataset.

2.3 Models

For the purpose of analise the data in every model, the data set is partitioned in two sets: training and test in a proportion of 90:10. We train the model using the first data set (training), then test it using the second set (test) and then calculate the RMSE.

Train Set Dimension

```
set.seed(2000, sample.kind="Rounding")
test_prt <- createDataPartition(data$CO, times = 1, p = 0.1, list = FALSE)
train_dts <- data[-test_prt,]
test_dts <- data[test_prt,]
dim(train_dts)
```

```
## [1] 28375    13
```

Test Set Dimension

```
dim(test_dts)
```

```
## [1] 3155   13
```

```
real_CO <- test_dts$CO
```

Once the data to be analyzed and the method to be used have been identified; we must evaluate the performance of the method and we will do it in this case with the Root Mean Square Error (RMSE) which is a precision measure, used to compare prediction errors of different models for a particular data set, and among more the closer the value is to zero, the more accurate the method will be. The RMSE formula is as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(x_t - x_p)^2}{n}}$$

Where:

$RMSE$ is the Root Mean Square Error

$\sum_{i=1}^{n}$ is the sum of the quadratic difference

x_t is the real data

x_p is the predicted data

```
RMSE <- function(real_data, predicted_data){
                  sqrt(mean((real_data - predicted_data)^2))
                  }
```

2.3.1 Simple Average

In order to enhance the processing times, the first the model is this project is "Simple Average", which is described with the following formula:

$$\hat{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

Where:

$\hat{X}$ is the average rating of all observations

$\sum_{i=1}^{n}$ is the summation of data

X_i is the rating given by a user to a movie at a certain time

n is the number of observations

We start with calculating the average, we obtain:

```
# Calculation of the average from the training set
avg <-mean(train_dts$CO)
avg
```

```
## [1] 1213.097
```

Now calculating the RMSE, we obtain:

```
# Calculation of the RMSE with the previous avg and validation set
rmse_avg <- RMSE(real_CO, avg)
```

So we save the RMSE: 1142.09

```
options(pillar.sigfig = 6)
models_result <- tibble(Model = "Simple Average", RMSE = rmse_avg)
models_result
```

```
## # A tibble: 1 x 2
##   Model              RMSE
##   <chr>             <dbl>
## 1 Simple Average 1142.09
```

2.3.2 k Nearest Neighbours (kNN)

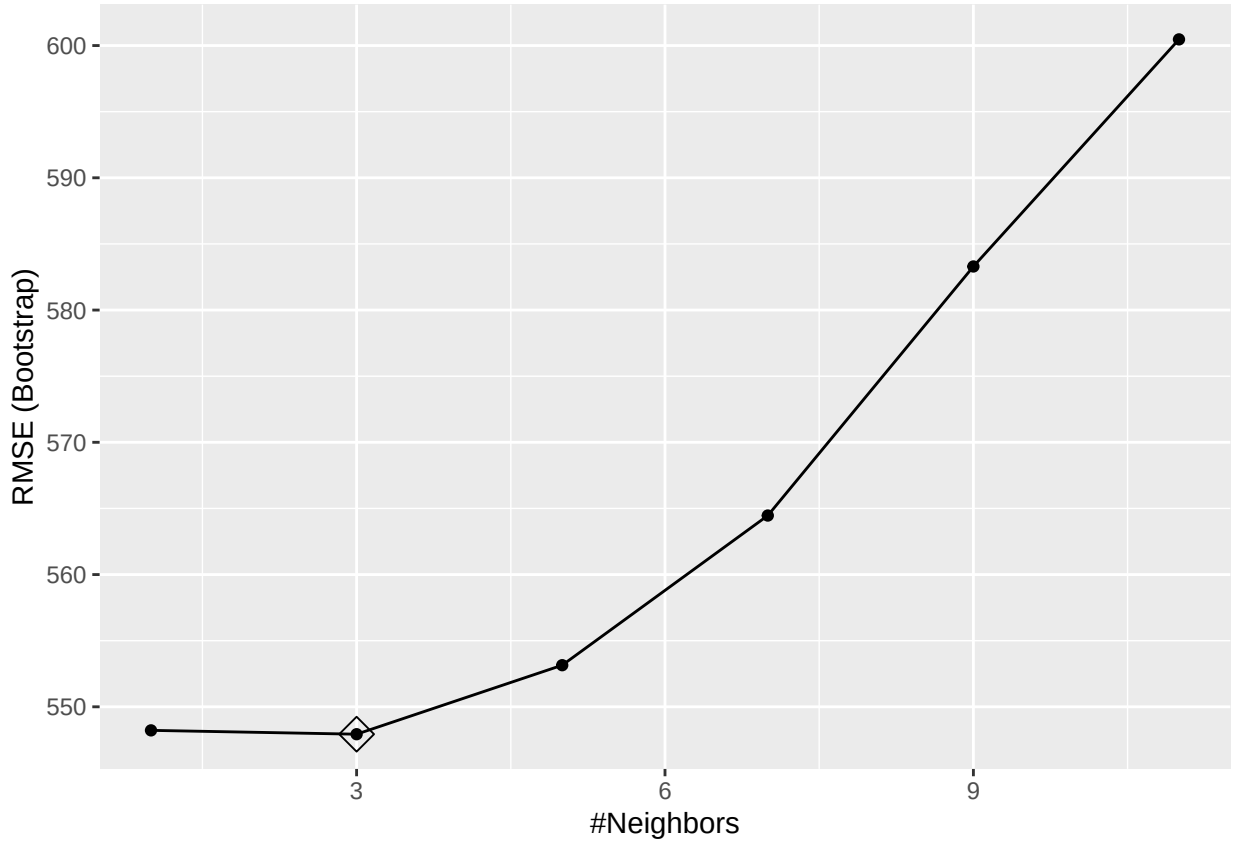
k Nearest Neighbours (kNN) model is a supervised classification method (Learning, estimation based on a training set and prototypes). This is a non-parametric classification method, which estimates the value of the probability density function or directly the posterior probability that an element x belongs to class **C** from the information provided by the prototype set.

Then the following plot show us the different k values tested

```
#### identifying the best "k"

set.seed(2000, sample.kind="Rounding")
```

```
## Warning in set.seed(2000, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
knn_k <- train(CO ~ date,
                   method = "knn",
                   tuneGrid = data.frame(k = seq(1, 11, 2)),
                   data = train_dts)

## Plot the K's calculated
ggplot(knn_k, highlight = TRUE)
```



So the best "k" is:

```
##Getting the best "k":
best_k<-knn_k$bestTune
best_k
```

```
##   k
## 2 3
```

So, we begin training the first set

```
#### Training the model
train_knn <- train(CO ~ date,
                   method = "knn",
                   tuneGrid = data.frame(k = best_k),
                   data = train_dts)
```

Calculating the prediction

```
## Getting the prediction
pred_knn <- predict(train_knn, test_dts)
```

Now calculating the RMSE, we obtain: 537.208

```
# Calculation of the RMSE with the predicted and test set
rmse_knn <- RMSE(real_CO, pred_knn)
```

So we save the result.

```
models_result <- bind_rows(models_result,
                           tibble(Model = "K-Nearest Neighbors", RMSE = rmse_knn))
models_result
```

```
## # A tibble: 2 x 2
##   Model                  RMSE
##   <chr>                 <dbl>
## 1 Simple Average      1142.09
## 2 K-Nearest Neighbors  537.208
```

2.3.3 Random Forest

The Random Forest Model is a combination of predictor trees such that each tree depends on the values of an independently tested random vector and with the same distribution for each of these.

So, we begin training the first set

```
#### Training the model
set.seed(2000, sample.kind="Rounding")
```

```
## Warning in set.seed(2000, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
train_rnd <- ranger(CO ~ date,
                    data = train_dts,
                    num.trees = 400,
                    respect.unordered.factors = "order",
                    seed = 2000)
print(train_rnd)
```

```
## Ranger result
##
```

```
## Call:
##  ranger(CO ~ date, data = train_dts, num.trees = 400, respect.unordered.factors = "order",      seed
##
## Type:                             Regression
## Number of trees:                  400
## Sample size:                      28375
## Number of independent variables:  1
## Mtry:                             1
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       274155.2
## R squared (OOB):                  0.7873935
```

Calculating the prediction

```
## Getting the prediction
pred_rnd <- predict(train_rnd, test_dts)$predictions
```

Now calculating the RMSE, we obtain: 537.420

```
# Calculation of the RMSE with the predicted and test set
rmse_rnd <- RMSE(real_CO, pred_rnd)
```

So we save the result.

```
models_result <- bind_rows(models_result,
                           tibble(Model = "Random Forest", RMSE = rmse_rnd))
models_result
```

```
## # A tibble: 3 x 2
##   Model                  RMSE
##   <chr>                 <dbl>
## 1 Simple Average      1142.09
## 2 K-Nearest Neighbors  537.208
## 3 Random Forest        537.420
```

2.3.4 Neural Network

The Neural Netwok model consists of a set of units, called artificial neurons, connected together to transmit signals. The input information traverses the neural network (where it undergoes various operations) producing output values.

Each neuron is connected to others through links. In these links, the output value of the previous neuron is multiplied by a weight value. These link weights can increase or inhibit the activation state of adjacent neurons. Similarly, at the exit of the neuron, there may be a limiting function or threshold, which modifies the result value or imposes a limit that must not be exceeded before spreading to another neuron. This function is known as an activation function.

So, we begin training the first set

```
#### Training the model
train_nnt <- train(CO ~ date,
                   method = "nnet",
                   data = train_dts)
```

```
## # weights:  4
## initial  value 76919235277.513031
## final  value 76893164630.000000
```

```
## converged
## # weights:  10
## initial  value 76943503540.685822
## final  value 76893164630.000000
## converged
## # weights:  16
## initial  value 76903025393.520630
## final  value 76893164630.000000
## converged
## # weights:  4
## initial  value 76932734656.981216
## iter  10 value 76893245504.815872
## final  value 76893165154.488022
## converged
## # weights:  10
## initial  value 76919280834.904358
## iter  10 value 76893176700.811554
## final  value 76893165359.052612
## converged
## # weights:  16
## initial  value 76910322528.147629
## iter  10 value 76893174180.007568
## final  value 76893165206.801147
## converged
## # weights:  4
## initial  value 76935317313.062775
## final  value 76893170041.121445
## converged
## # weights:  10
## initial  value 76920341751.128189
## final  value 76893172903.255981
## converged
## # weights:  16
## initial  value 76934422909.432098
## final  value 76893172940.621658
## converged
## # weights:  4
## initial  value 76638050758.150604
## final  value 76593017889.000000
## converged
## # weights:  10
## initial  value 76619893431.819641
## final  value 76593017889.000000
## converged
## # weights:  16
## initial  value 76622616720.494110
## final  value 76593017889.000000
## converged
## # weights:  4
## initial  value 76627142428.900528
## iter  10 value 76593380610.530136
## iter  20 value 76593022070.894516
## final  value 76593018505.645432
## converged
```

```
## # weights:  10
## initial  value 76613237438.826294
## iter  10 value 76593134904.980209
## final  value 76593018647.870102
## converged
## # weights:  16
## initial  value 76606441909.864883
## iter  10 value 76593024607.183090
## final  value 76593018523.007294
## converged
## # weights:  4
## initial  value 76638299765.962601
## final  value 76593077966.415771
## converged
## # weights:  10
## initial  value 76628551902.532150
## final  value 76593026852.553543
## converged
## # weights:  16
## initial  value 76640913671.639099
## final  value 76593022120.262222
## converged
## # weights:  4
## initial  value 76360561098.815170
## final  value 76322965639.000000
## converged
## # weights:  10
## initial  value 76345982791.864563
## final  value 76322965639.000000
## converged
## # weights:  16
## initial  value 76366478553.792145
## final  value 76322965639.000000
## converged
## # weights:  4
## initial  value 76348311933.871689
## iter  10 value 76323112044.526459
## iter  20 value 76322967326.940796
## iter  20 value 76322966719.282104
## final  value 76322966719.282104
## converged
## # weights:  10
## initial  value 76369143490.593933
## iter  10 value 76323094319.613678
## iter  20 value 76322967122.586472
## iter  20 value 76322966588.495346
## iter  20 value 76322966173.091125
## final  value 76322966173.091125
## converged
## # weights:  16
## initial  value 76364039463.405975
## iter  10 value 76323119940.026337
## iter  20 value 76322967417.969711
## iter  20 value 76322966777.540619
```

```
## iter  20 value 76322966279.429092
## final  value 76322966279.429092
## converged
## # weights:  4
## initial  value 76350104472.800827
## final  value 76322968375.759674
## converged
## # weights:  10
## initial  value 76350339845.961792
## final  value 76322968391.641937
## converged
## # weights:  16
## initial  value 76358236016.690079
## final  value 76322971588.655609
## converged
## # weights:  4
## initial  value 77346647158.182693
## final  value 77307432961.000000
## converged
## # weights:  10
## initial  value 77331915787.268814
## final  value 77307432961.000000
## converged
## # weights:  16
## initial  value 77332122571.070145
## final  value 77307432961.000000
## converged
## # weights:  4
## initial  value 77342496445.713898
## iter  10 value 77307601209.611679
## iter  20 value 77307434900.774429
## iter  20 value 77307434202.455627
## final  value 77307434202.455627
## converged
## # weights:  10
## initial  value 77333761876.470001
## iter  10 value 77307584101.158661
## iter  20 value 77307434703.527390
## iter  20 value 77307434076.217529
## iter  20 value 77307433588.309860
## final  value 77307433588.309860
## converged
## # weights:  16
## initial  value 77331162260.487274
## iter  10 value 77307571381.597824
## iter  20 value 77307434556.880844
## iter  20 value 77307433982.363739
## iter  20 value 77307433535.517105
## final  value 77307433535.517105
## converged
## # weights:  4
## initial  value 77328928189.781464
## final  value 77307437408.581726
## converged
```

```
## # weights:  10
## initial  value 77354386329.298187
## final  value 77307437391.957382
## converged
## # weights:  16
## initial  value 77323706100.917496
## final  value 77307433213.025055
## converged
## # weights:  4
## initial  value 79077498007.600235
## final  value 79047527645.000000
## converged
## # weights:  10
## initial  value 79099787611.332199
## final  value 79047527645.000000
## converged
## # weights:  16
## initial  value 79059705431.684006
## final  value 79047527645.000000
## converged
## # weights:  4
## initial  value 79090560425.475250
## iter  10 value 79047602637.954575
## final  value 79047528404.911041
## converged
## # weights:  10
## initial  value 79074699482.527283
## iter  10 value 79047683584.881393
## iter  20 value 79047529442.864426
## iter  20 value 79047528795.633240
## iter  20 value 79047528292.231201
## final  value 79047528292.231201
## converged
## # weights:  16
## initial  value 79088914698.500565
## iter  10 value 79047684656.556427
## iter  20 value 79047529455.220001
## iter  20 value 79047528803.540802
## iter  20 value 79047528296.679199
## final  value 79047528296.679199
## converged
## # weights:  4
## initial  value 79091413992.094131
## final  value 79047592397.657822
## converged
## # weights:  10
## initial  value 79069564204.612473
## final  value 79047534538.584518
## converged
## # weights:  16
## initial  value 79089578836.646225
## final  value 79047538607.721176
## converged
## # weights:  4
```

```
## initial  value 79083063020.420609
## final  value 79040024606.000000
## converged
## # weights:  10
## initial  value 79064551956.590179
## final  value 79040024606.000000
## converged
## # weights:  16
## initial  value 79088302253.914291
## final  value 79040024606.000000
## converged
## # weights:  4
## initial  value 79082605521.966248
## iter  10 value 79040101715.622910
## final  value 79040025387.359451
## converged
## # weights:  10
## initial  value 79079500992.327332
## iter  10 value 79040037834.421173
## final  value 79040025117.340729
## converged
## # weights:  16
## initial  value 79082175514.076279
## iter  10 value 79040704509.026993
## final  value 79040025152.022141
## converged
## # weights:  4
## initial  value 79058916833.012680
## final  value 79040028466.163589
## converged
## # weights:  10
## initial  value 79075969915.146027
## final  value 79040027654.668564
## converged
## # weights:  16
## initial  value 79055881487.073532
## final  value 79040024850.764465
## converged
## # weights:  4
## initial  value 77257798122.501770
## final  value 77228753603.000000
## converged
## # weights:  10
## initial  value 77259583441.772125
## final  value 77228753603.000000
## converged
## # weights:  16
## initial  value 77253186650.630051
## final  value 77228753603.000000
## converged
## # weights:  4
## initial  value 77270416810.886032
## iter  10 value 77228908536.047806
## iter  20 value 77228755389.256424
```

```
## iter  20 value 77228754746.204117
## final  value 77228754746.204117
## converged
## # weights:  10
## initial  value 77279129346.973053
## iter  10 value 77228857615.462784
## final  value 77228754277.539902
## converged
## # weights:  16
## initial  value 77264683821.196060
## iter  10 value 77228770532.457565
## final  value 77228754257.403198
## converged
## # weights:  4
## initial  value 77263942575.734741
## final  value 77228759633.658234
## converged
## # weights:  10
## initial  value 77257085230.677475
## final  value 77228762113.141556
## converged
## # weights:  16
## initial  value 77277766078.801926
## final  value 77228761676.750000
## converged
## # weights:  4
## initial  value 78706435834.583542
## final  value 78676407427.000000
## converged
## # weights:  10
## initial  value 78705412071.078308
## final  value 78676407427.000000
## converged
## # weights:  16
## initial  value 78709744292.677261
## final  value 78676407427.000000
## converged
## # weights:  4
## initial  value 78704023174.427658
## iter  10 value 78676565593.389877
## iter  20 value 78676409250.534317
## iter  20 value 78676408594.061966
## final  value 78676408594.061966
## converged
## # weights:  10
## initial  value 78727947150.262192
## iter  10 value 78676421818.469025
## final  value 78676407983.298004
## converged
## # weights:  16
## initial  value 78708528175.519638
## iter  10 value 78676492390.502899
## final  value 78676407978.003906
## converged
```

```
## # weights:  4
## initial  value 78721503174.878235
## final  value 78676412423.857376
## converged
## # weights:  10
## initial  value 78713197410.876297
## final  value 78676416484.895294
## converged
## # weights:  16
## initial  value 78685017284.531876
## final  value 78676407543.436554
## converged
## # weights:  4
## initial  value 79785179743.442688
## final  value 79748560394.000000
## converged
## # weights:  10
## initial  value 79769752890.013245
## final  value 79748560394.000000
## converged
## # weights:  16
## initial  value 79772568668.231995
## final  value 79748560394.000000
## converged
## # weights:  4
## initial  value 79791280671.427719
## iter  10 value 79748637976.626663
## final  value 79748561180.152451
## converged
## # weights:  10
## initial  value 79784386327.662643
## iter  10 value 79748577315.498947
## final  value 79748561048.095566
## converged
## # weights:  16
## initial  value 79787735382.732819
## iter  10 value 79748561978.905228
## iter  10 value 79748561469.308380
## iter  10 value 79748561431.876633
## final  value 79748561431.876633
## converged
## # weights:  4
## initial  value 79788823409.269363
## final  value 79748566234.027634
## converged
## # weights:  10
## initial  value 79786331598.191925
## final  value 79748560878.591568
## converged
## # weights:  16
## initial  value 79783541767.708023
## final  value 79748569625.558823
## converged
## # weights:  4
```

```
## initial  value 79603640827.361267
## final  value 79565375618.000000
## converged
## # weights:  10
## initial  value 79582934273.777908
## final  value 79565375618.000000
## converged
## # weights:  16
## initial  value 79598121468.683640
## final  value 79565375618.000000
## converged
## # weights:  4
## initial  value 79588728557.528656
## iter  10 value 79565444715.654541
## final  value 79565376318.173386
## converged
## # weights:  10
## initial  value 79609158936.002487
## iter  10 value 79565525274.113968
## iter  20 value 79565377343.417526
## iter  20 value 79565376722.267212
## iter  20 value 79565376239.150314
## final  value 79565376239.150314
## converged
## # weights:  16
## initial  value 79592647723.735809
## iter  10 value 79565391384.113647
## final  value 79565376227.434479
## converged
## # weights:  4
## initial  value 79593655393.511398
## final  value 79565378485.448044
## converged
## # weights:  10
## initial  value 79601754983.805847
## final  value 79565381728.154373
## converged
## # weights:  16
## initial  value 79594488464.852921
## final  value 79565384367.051422
## converged
## # weights:  4
## initial  value 78585336922.168625
## final  value 78552796582.000000
## converged
## # weights:  10
## initial  value 78578132241.289093
## final  value 78552796582.000000
## converged
## # weights:  16
## initial  value 78575710024.123993
## final  value 78552796582.000000
## converged
## # weights:  4
```

```
## initial  value 78587941861.168839
## iter  10 value 78552966594.751160
## iter  20 value 78552798542.113571
## iter  20 value 78552797836.472687
## final  value 78552797836.472687
## converged
## # weights:  10
## initial  value 78573883476.357010
## iter  10 value 78552919771.611053
## iter  20 value 78552798002.279510
## iter  20 value 78552797490.978897
## iter  20 value 78552797093.300629
## final  value 78552797093.300629
## converged
## # weights:  16
## initial  value 78557347206.699020
## iter  10 value 78552822624.782730
## final  value 78552797226.272705
## converged
## # weights:  4
## initial  value 78590301501.398529
## final  value 78552799553.508957
## converged
## # weights:  10
## initial  value 78584985864.917831
## final  value 78552802577.733673
## converged
## # weights:  16
## initial  value 78581232592.897858
## final  value 78552797151.025803
## converged
## # weights:  4
## initial  value 78426913203.334274
## final  value 78403917301.000000
## converged
## # weights:  10
## initial  value 78448129777.245193
## final  value 78403917301.000000
## converged
## # weights:  16
## initial  value 78426182805.093185
## final  value 78403917301.000000
## converged
## # weights:  4
## initial  value 78445576693.579483
## iter  10 value 78404072183.137115
## iter  20 value 78403919086.669464
## iter  20 value 78403918443.828461
## final  value 78403918443.828461
## converged
## # weights:  10
## initial  value 78443284417.369064
## iter  10 value 78403970314.199661
## final  value 78403918126.442444
```

```
## converged
## # weights:  16
## initial  value 78438852128.480682
## iter  10 value 78403930890.648880
## final  value 78403917826.303879
## converged
## # weights:  4
## initial  value 78434716544.579590
## final  value 78403923209.653931
## converged
## # weights:  10
## initial  value 78442501864.883881
## final  value 78403923154.251984
## converged
## # weights:  16
## initial  value 78441578864.916275
## final  value 78403929148.931091
## converged
## # weights:  4
## initial  value 78659727981.500763
## final  value 78629832427.000000
## converged
## # weights:  10
## initial  value 78678405830.537369
## final  value 78629832427.000000
## converged
## # weights:  16
## initial  value 78675438282.378479
## final  value 78629832427.000000
## converged
## # weights:  4
## initial  value 78668717290.120483
## iter  10 value 78629914717.354294
## final  value 78629832960.668045
## converged
## # weights:  10
## initial  value 78656603444.461044
## iter  10 value 78629986528.027771
## iter  20 value 78629834203.663895
## iter  20 value 78629833564.064896
## iter  20 value 78629833066.598999
## final  value 78629833066.598999
## converged
## # weights:  16
## initial  value 78675557283.246063
## iter  10 value 78629967740.420761
## iter  20 value 78629833987.057526
## iter  20 value 78629833425.436813
## iter  20 value 78629832988.620712
## final  value 78629832988.620712
## converged
## # weights:  4
## initial  value 78658299527.570343
## final  value 78629838125.144684
```

```
## converged
## # weights:  10
## initial  value 78672723601.506866
## final  value 78629837759.274857
## converged
## # weights:  16
## initial  value 78664672219.603622
## final  value 78629833031.868240
## converged
## # weights:  4
## initial  value 76075513932.596420
## final  value 76053096594.000000
## converged
## # weights:  10
## initial  value 76071744998.976547
## final  value 76053096594.000000
## converged
## # weights:  16
## initial  value 76090344023.672653
## final  value 76053096594.000000
## converged
## # weights:  4
## initial  value 76080441807.481522
## iter  10 value 76053174393.683838
## final  value 76053097098.545227
## converged
## # weights:  10
## initial  value 76095843589.600784
## iter  10 value 76053245421.750839
## iter  20 value 76053098309.867142
## iter  20 value 76053097692.154968
## iter  20 value 76053097211.712173
## final  value 76053097211.712173
## converged
## # weights:  16
## initial  value 76083349912.254181
## iter  10 value 76053113041.447586
## final  value 76053097229.771240
## converged
## # weights:  4
## initial  value 76078764556.086517
## final  value 76053099230.970673
## converged
## # weights:  10
## initial  value 76091840884.557907
## final  value 76053105289.750336
## converged
## # weights:  16
## initial  value 76066731637.144287
## final  value 76053100257.240570
## converged
## # weights:  4
## initial  value 78400125788.793076
## final  value 78357723400.000000
```

```
## converged
## # weights:  10
## initial  value 78373499803.728210
## final  value 78357723400.000000
## converged
## # weights:  16
## initial  value 78416701683.599045
## final  value 78357723400.000000
## converged
## # weights:  4
## initial  value 78389910943.728378
## iter  10 value 78357807871.161224
## final  value 78357723947.810974
## converged
## # weights:  10
## initial  value 78385677882.908936
## iter  10 value 78357882042.691727
## iter  20 value 78357725229.025711
## iter  20 value 78357724570.576447
## iter  20 value 78357724058.449249
## final  value 78357724058.449249
## converged
## # weights:  16
## initial  value 78405600033.177567
## iter  10 value 78357804451.609375
## final  value 78357723925.634552
## converged
## # weights:  4
## initial  value 78412015118.897034
## final  value 78357790145.563751
## converged
## # weights:  10
## initial  value 78388300210.381714
## final  value 78357729301.644363
## converged
## # weights:  16
## initial  value 78407548293.005325
## final  value 78357729177.631119
## converged
## # weights:  4
## initial  value 77269552389.357132
## final  value 77229971170.000000
## converged
## # weights:  10
## initial  value 77257587709.127167
## final  value 77229971170.000000
## converged
## # weights:  16
## initial  value 77270927354.583237
## final  value 77229971170.000000
## converged
## # weights:  4
## initial  value 77254950363.084793
## iter  10 value 77230043609.756897
```

```
## final  value 77229971904.039246
## converged
## # weights:  10
## initial  value 77264934024.503296
## iter  10 value 77230139354.150009
## iter  20 value 77229973109.031235
## iter  20 value 77229972410.979996
## iter  20 value 77229971868.051239
## final  value 77229971868.051239
## converged
## # weights:  16
## initial  value 77259529287.860947
## iter  10 value 77229980907.395065
## final  value 77229971758.118973
## converged
## # weights:  4
## initial  value 77255168187.108444
## final  value 77229973761.372131
## converged
## # weights:  10
## initial  value 77277298878.092682
## final  value 77229975509.707443
## converged
## # weights:  16
## initial  value 77262837018.703827
## final  value 77229980112.159607
## converged
## # weights:  4
## initial  value 78594730970.311966
## final  value 78555869549.000000
## converged
## # weights:  10
## initial  value 78581402029.195709
## final  value 78555869549.000000
## converged
## # weights:  16
## initial  value 78608298909.027557
## final  value 78555869549.000000
## converged
## # weights:  4
## initial  value 78586858875.216141
## iter  10 value 78556036697.094208
## iter  20 value 78555871476.086319
## iter  20 value 78555870782.335251
## final  value 78555870782.335251
## converged
## # weights:  10
## initial  value 78589751794.936569
## iter  10 value 78555879772.490158
## final  value 78555870166.478134
## converged
## # weights:  16
## initial  value 78583172587.990158
## iter  10 value 78555882054.326736
```

```
## final  value 78555870304.296448
## converged
## # weights:  4
## initial  value 78591817005.198334
## final  value 78555872563.360672
## converged
## # weights:  10
## initial  value 78604218573.171021
## final  value 78555873772.172638
## converged
## # weights:  16
## initial  value 78579809022.450012
## final  value 78555877030.014191
## converged
## # weights:  4
## initial  value 77707670643.018143
## final  value 77665412182.000000
## converged
## # weights:  10
## initial  value 77686941130.247147
## final  value 77665412182.000000
## converged
## # weights:  16
## initial  value 77713540414.024796
## final  value 77665412182.000000
## converged
## # weights:  4
## initial  value 77716173053.163879
## iter  10 value 77665510736.879761
## final  value 77665412821.146469
## converged
## # weights:  10
## initial  value 77700665128.535614
## iter  10 value 77665580297.380020
## iter  20 value 77665414120.238373
## iter  20 value 77665413422.472549
## iter  20 value 77665412879.765808
## final  value 77665412879.765808
## converged
## # weights:  16
## initial  value 77711506862.653030
## iter  10 value 77665427785.240265
## final  value 77665412785.138657
## converged
## # weights:  4
## initial  value 77703278943.598190
## final  value 77665415103.872955
## converged
## # weights:  10
## initial  value 77705583652.944519
## final  value 77665417802.707123
## converged
## # weights:  16
## initial  value 77689056625.041779
```

```
## final  value 77665412769.907288
## converged
## # weights:  4
## initial  value 78571960477.167023
## final  value 78527007343.000000
## converged
## # weights:  10
## initial  value 78552537931.339508
## final  value 78527007343.000000
## converged
## # weights:  16
## initial  value 78556609735.481995
## final  value 78527007343.000000
## converged
## # weights:  4
## initial  value 78567094056.198944
## iter  10 value 78527089318.128143
## final  value 78527007874.623749
## converged
## # weights:  10
## initial  value 78548880785.623489
## iter  10 value 78527015087.512329
## final  value 78527008073.863876
## converged
## # weights:  16
## initial  value 78574958288.398712
## iter  10 value 78527115422.539536
## final  value 78527007503.682587
## converged
## # weights:  4
## initial  value 78559868089.634720
## final  value 78527010391.256744
## converged
## # weights:  10
## initial  value 78575616308.991318
## final  value 78527011625.440826
## converged
## # weights:  16
## initial  value 78561973168.438889
## final  value 78527016531.789001
## converged
## # weights:  4
## initial  value 78488769948.687851
## final  value 78458767731.000000
## converged
## # weights:  10
## initial  value 78470341696.632401
## final  value 78458767731.000000
## converged
## # weights:  16
## initial  value 78495559571.417038
## final  value 78458767731.000000
## converged
## # weights:  4
```

```
## initial  value 78486264552.276047
## iter  10 value 78458846376.694336
## final  value 78458768241.031754
## converged
## # weights:  10
## initial  value 78484087215.639389
## iter  10 value 78458915501.151047
## iter  20 value 78458769434.673843
## iter  20 value 78458768821.351257
## iter  20 value 78458768344.322586
## final  value 78458768344.322586
## converged
## # weights:  16
## initial  value 78498816643.054535
## iter  10 value 78458780678.310013
## final  value 78458768512.991333
## converged
## # weights:  4
## initial  value 78500920091.472916
## final  value 78458773202.688675
## converged
## # weights:  10
## initial  value 78484899318.445984
## final  value 78458773114.157089
## converged
## # weights:  16
## initial  value 78498706865.103195
## final  value 78458779251.690094
## converged
## # weights:  4
## initial  value 77062316192.447861
## final  value 77026485561.000000
## converged
## # weights:  10
## initial  value 77065521826.254211
## final  value 77026485561.000000
## converged
## # weights:  16
## initial  value 77062833329.640472
## final  value 77026485561.000000
## converged
## # weights:  4
## initial  value 77047881592.464951
## iter  10 value 77026610420.585663
## iter  20 value 77026487000.533020
## iter  20 value 77026486482.301132
## final  value 77026486482.301132
## converged
## # weights:  10
## initial  value 77043574911.120590
## iter  10 value 77026580445.099548
## final  value 77026486176.340790
## converged
## # weights:  16
```

```
## initial  value 77061210742.576279
## iter  10 value 77026495709.767426
## final  value 77026486173.965027
## converged
## # weights:  4
## initial  value 77052932118.697632
## final  value 77026490961.116211
## converged
## # weights:  10
## initial  value 77045600990.640427
## final  value 77026491395.879440
## converged
## # weights:  16
## initial  value 77040824725.306320
## final  value 77026485824.400436
## converged
## # weights:  4
## initial  value 79264771640.061035
## final  value 79245514188.000000
## converged
## # weights:  10
## initial  value 79293067962.375381
## final  value 79245514188.000000
## converged
## # weights:  16
## initial  value 79297105611.078751
## final  value 79245514188.000000
## converged
## # weights:  4
## initial  value 79284467503.304291
## iter  10 value 79245597716.733521
## final  value 79245514729.699158
## converged
## # weights:  10
## initial  value 79272036076.696121
## iter  10 value 79245668418.446533
## iter  20 value 79245515966.155991
## iter  20 value 79245515326.019836
## iter  20 value 79245514828.136154
## final  value 79245514828.136154
## converged
## # weights:  16
## initial  value 79282676457.914337
## iter  10 value 79245684791.373413
## iter  20 value 79245516154.922974
## iter  20 value 79245515446.830704
## iter  20 value 79245514896.092270
## final  value 79245514896.092270
## converged
## # weights:  4
## initial  value 79285254396.513321
## final  value 79245520045.865067
## converged
## # weights:  10
```

```
## initial  value 79285894799.717163
## final  value 79245522870.170731
## converged
## # weights:  16
## initial  value 79289345054.484406
## final  value 79245524756.027557
## converged
## # weights:  4
## initial  value 77200929953.134491
## final  value 77169688546.000000
## converged
## # weights:  10
## initial  value 77215008620.534256
## final  value 77169688546.000000
## converged
## # weights:  16
## initial  value 77216308871.369858
## final  value 77169688546.000000
## converged
## # weights:  4
## initial  value 77207968820.390228
## iter  10 value 77169770060.492828
## final  value 77169689074.636444
## converged
## # weights:  10
## initial  value 77190585194.581055
## iter  10 value 77169809784.989059
## iter  20 value 77169689943.790375
## iter  20 value 77169689440.585846
## iter  20 value 77169689049.204529
## final  value 77169689049.204529
## converged
## # weights:  16
## initial  value 77206853576.208923
## iter  10 value 77169701770.702393
## final  value 77169689057.196976
## converged
## # weights:  4
## initial  value 77207402459.122452
## final  value 77169691462.937607
## converged
## # weights:  10
## initial  value 77206690421.335556
## final  value 77169691486.238510
## converged
## # weights:  16
## initial  value 77192730931.517929
## final  value 77169688927.802139
## converged
## # weights:  4
## initial  value 78412463246.679230
## final  value 78387478588.000000
## converged
## # weights:  10
```

```
## initial  value 78417257392.616165
## final  value 78387478588.000000
## converged
## # weights:  16
## initial  value 78420080389.348480
## final  value 78387478588.000000
## converged
## # weights:  4
## initial  value 78412487158.922516
## iter  10 value 78387551834.771240
## final  value 78387479330.216797
## converged
## # weights:  10
## initial  value 78409751245.726578
## iter  10 value 78387609688.377670
## iter  20 value 78387480099.484451
## iter  20 value 78387479555.350052
## iter  20 value 78387479132.134399
## final  value 78387479132.134399
## converged
## # weights:  16
## initial  value 78417897827.954697
## iter  10 value 78387645222.156784
## iter  20 value 78387480509.161026
## iter  20 value 78387479817.543060
## iter  20 value 78387479279.617966
## final  value 78387479279.617966
## converged
## # weights:  4
## initial  value 78415493743.954269
## final  value 78387481420.737885
## converged
## # weights:  10
## initial  value 78412833571.332687
## final  value 78387486482.182281
## converged
## # weights:  16
## initial  value 78425216881.857727
## final  value 78387490541.963516
## converged
## # weights:  4
## initial  value 78449316869.700989
## final  value 78422554331.000000
## converged
## # weights:  10
## initial  value 78445459347.451279
## final  value 78422554331.000000
## converged
## # weights:  16
## initial  value 78463576221.366562
## final  value 78422554331.000000
## converged
## # weights:  4
## initial  value 78454493327.792221
```

```
## iter  10 value 78422723569.964386
## iter  20 value 78422556282.192413
## iter  20 value 78422555579.763138
## final  value 78422555579.763138
## converged
## # weights:  10
## initial  value 78463975964.783463
## iter  10 value 78422632984.496170
## final  value 78422554841.082352
## converged
## # weights:  16
## initial  value 78464873312.861847
## iter  10 value 78422700430.134598
## iter  20 value 78422556015.408386
## iter  20 value 78422555409.021362
## iter  20 value 78422554937.387024
## final  value 78422554937.387024
## converged
## # weights:  4
## initial  value 78455244488.887192
## final  value 78422557352.480896
## converged
## # weights:  10
## initial  value 78460373021.299194
## final  value 78422563269.175110
## converged
## # weights:  16
## initial  value 78443162085.295624
## final  value 78422554758.275238
## converged
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
## # weights:  4
## initial  value 78314179180.598114
## iter  10 value 78276215186.721985
## final  value 78276132325.875473
## converged
```

Calculating the prediction

```
## Getting the prediction
pred_nnt <- predict(train_nnt, test_dts)
```

Now calculating the RMSE, we obtain: 1663.79

```
# Calculation of the RMSE with the predicted and test set
rmse_nnt <- RMSE(real_CO, pred_nnt)
```

So we save the result.

```
models_result <- bind_rows(models_result,
                           tibble(Model = "Neural Network", RMSE = rmse_nnt))
models_result
```

```
## # A tibble: 4 x 2
##   Model                  RMSE
```

```
##   <chr>                  <dbl>
## 1 Simple Average      1142.09
## 2 K-Nearest Neighbors  537.208
## 3 Random Forest        537.420
## 4 Neural Network      1663.79
```

3 Results

For this project 4 models have been tested to predict the air pollution in china considering the CO as the attribute to evaluate. The RMSE was the measure to evaluate the perfomance and viability for every model. For different models the RMSE is enlisted in the following table:

```
models_result
```

```
## # A tibble: 4 x 2
##   Model                   RMSE
##   <chr>                  <dbl>
## 1 Simple Average      1142.09
## 2 K-Nearest Neighbors  537.208
## 3 Random Forest        537.420
## 4 Neural Network      1663.79
```

4 Conclusions

The model **K-Nearest Neighbors** works best with data . The datatest has given us an RMSE of **537.208**, which is the lowest among the models examined.

The Simple Model gave us a fairly high RMSE for this data set (1142.09) which is explained by its simplicity.

The K-Nearest Neighbors Model gave us a lowest RMSE (537.208), due to the closeness of the data.

The Random Forest Model gave a lower RMSE (537.420) so it is a good option to analyze the data.

The Neural Network Model gave us the worst RMSE (1663.79) due to the type of data that was managed in this project.

It is therefore concluded that the model **K-Nearest Neighbors** is the appropriate for predicting or recognizing CO pollution given the required data.

Finally, based on the analyzed data, the growth trends of the contamination are high, so it will take longer to meet the defined goals.