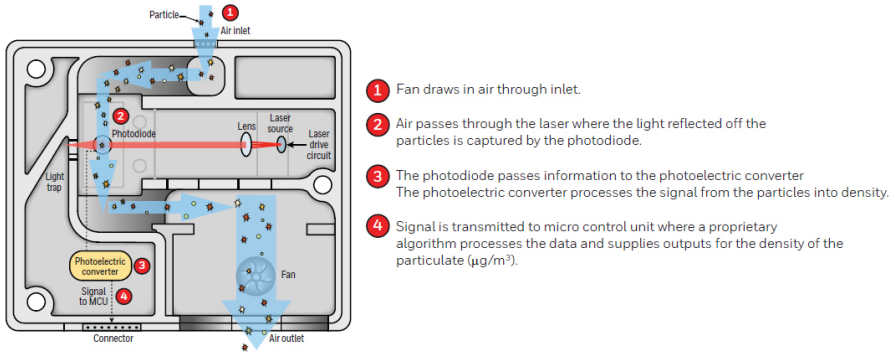


Figure 1: Schematic of Honeywell Particulate Matter Sensor. A laser light source illuminates a particle as it is pulled through the detection chamber. As particles pass through the laser beam, the light reflects off the particles and is recorded on the photo or light detector. The light is then analyzed and converted to an electrical signal to calculate particle concentration.

1 Introduction

1.1 How is particulate matter analyzed?

The PM sensor has several components to measure the particles in the air – remember, we don't know what the make up of these particles might be – but we can get an estimate of their size distribution. We have written the code to collect particles smaller than 1μm, 2.5μm and 10μm.

Figure 1 is a schematic of the particle sensor made by Honeywell (HPM Series — Particulate Matter Sensors 32322550HP). The one we are using is made by Plantower. As a Chinese company, most of their literature is in Chinese, so I didn't find something that I could interpret for us.

Our sensors can generate several categories of PM data. Using a laser and light defraction, the number and size of the particles can be estimated (Figure 2).

2 Data Collection

2.1 What data is collected by the PMS5003?

The PMS5003 generates the following data:

Data 1 refers to PM1.0 concentration unit μg/m^{3}CF=1standard particle)[1]

Data 2 refers to PM2.5 concentration unit μg/m^{3}CF=1standard particle

Data 3 refers to PM10 concentration unit μg/m^{3}CF=1standard particle

[1]The Asterisk on Data 1 and Data 4 is defined on the same page as "Note: CF=1 should be used in the factory environment"

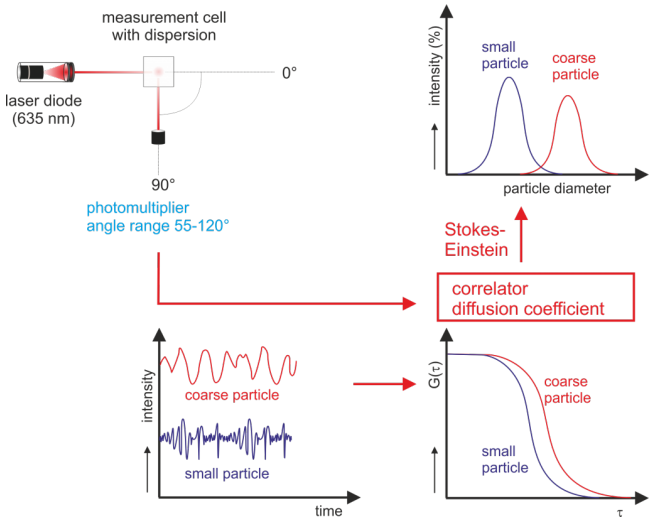


Figure 2: In particle size measurements using light scattering (DLS), a laser beam is scattered on very small, finely dispersed particles in a highly diluted matrix. The scattered light of each particle will then interfere with each other. Since the particles constantly change locations, the position of the scattering centers changes with respect to each other and the interferences lead to small fluctuations in scattering intensity (this explains the name "dynamic" light scattering).

Data 4 refers to PM1.0 concentration unit $\mu g/m^3$under atmospheric environment

Data 5 refers to PM2.5 concentration unit $\mu g/m^3$under atmospheric environment

Data 6 refers to concentration unit (under atmospheric environment) $\mu g/m^3$

Data 7 indicates the number of particles with diameter beyond 0.3 μm in 0.1 L of air.

Data 8 indicates the number of particles with diameter beyond 0.5 μm in 0.1 L of air.

Data 9 indicates the number of particles with diameter beyond 1.0 μm in 0.1 L of air.

Data 10 indicates the number of particles with diameter beyond 2.5 μm in 0.1 L of air.

Data 11 indicates the number of particles with diameter beyond 5.0 μm in 0.1 L of air.

Data 12 indicates the number of particles with diameter beyond 10 μm in 0.1 L of air

The first set are labeled "standard", while the second set are labeled "atmospheric environment". What do these measure and how do the "standard" ones differ from the "atmospheric environment" ones?

It has to do with the density of the air used for calculations. "Standard" refers to the concentration "corrected" to the "standard atmosphere" which in the US is DEFINED as "having a temperature of 288.15 K at the sea level 0 km geo-potential height and 1013.25 hPa". NOTE: 288.15 K equals 25 C and Pa are "pascals" which is a meausure of pressure.

On the other hand, the "ambient conditions" are just as the air is "now" (whatever temperature and pressure there is). Now what does that mean...

Air being a gas, it is compressible which means that it changes its volume when the pressure changes so when you report concentrations as mass per volume of air it is relevant at what pressure that volume is calculated. For example, if you have a bunch of particles rising in the air in a bubble (no loss of particles, no addition, they're just riding a bubble up in the air) then, as they rise, the pressure drops so what was 1cc at the ground it is now 2cc so the concentration of what was in the bubble is now half without anything actually changing other than the ambient pressure. So, it is common to report concentrations (of anything) as "x mg per standard m3" and because we scientist don't like to write much (current example excluded) you'll usually see the "standard" being dropped because it is "implicit".

For gases it is also common to report concentrations as "parts per million" or "ppm" and that metric is independent of the volume of air as it represents

the number of molecules of the gas in a million molecules of air (including the target gas).

In conclusion, I use the "standard" readings for reporting but keep the "ambient conditions" for analysis. I haven't used these things in high altitudes so for all my deployments the standard and ambient are very similar. I have written code to extract the standard values only – but you can certain add the ambient if you want to see if there is a difference – could be important if you are above 1000 ft in elevation.

3 Collecting the data

3.1 Extract Data From Pi

Once the data have been collected, you can extract from the Pi – Unfortunately, there are several ways to do this and one of them are easier using the VNC interface.

3.1.1 Using RealVNC to transfer data files

1. Use RealVNC and VNC into the Raspberry Pi using the Pi's IP adress in the RealVNC search toolbar.

2. Controlling the Raspberry Pi, right click on the VNCServer icon in the toolbar in the upper left near the clock.

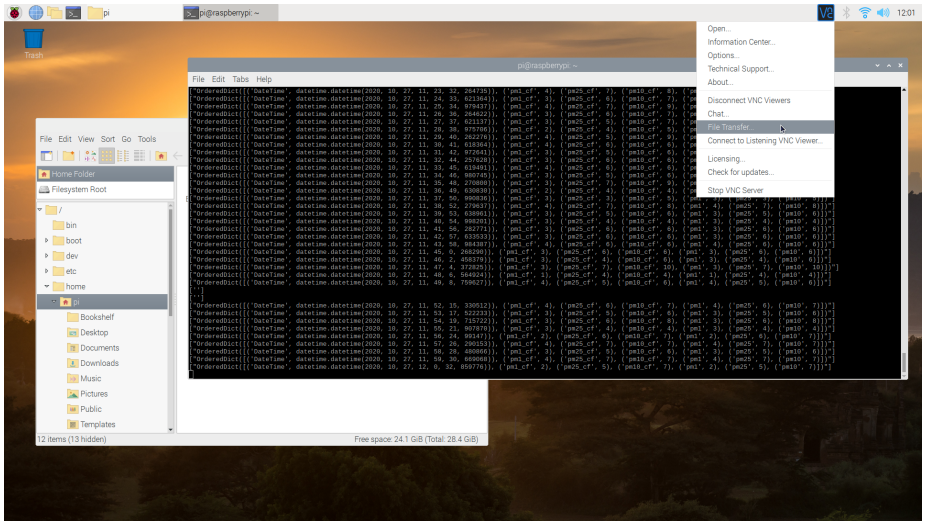


Figure 3:

3. Select **"File Transfer..."**

4. In the "File Transfer" popup window, select **"Send Files"** in the lower left.

Figure 4:

5. Then navigate to the file you want to send in the "Send Files" popup and click **"OK"**.

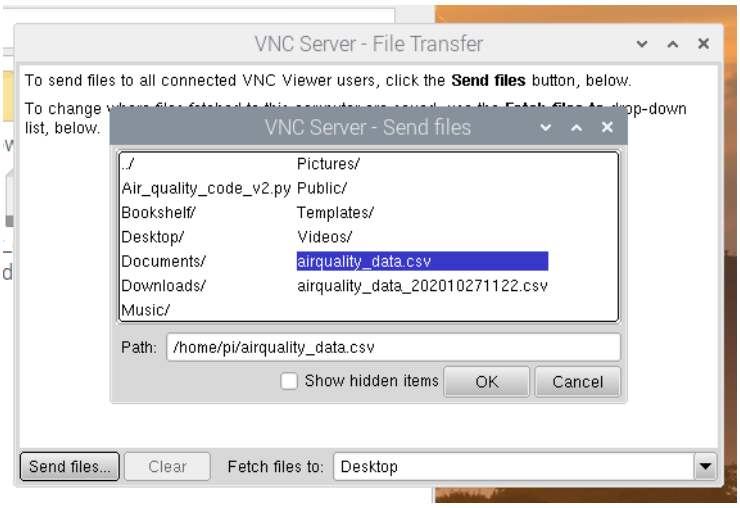
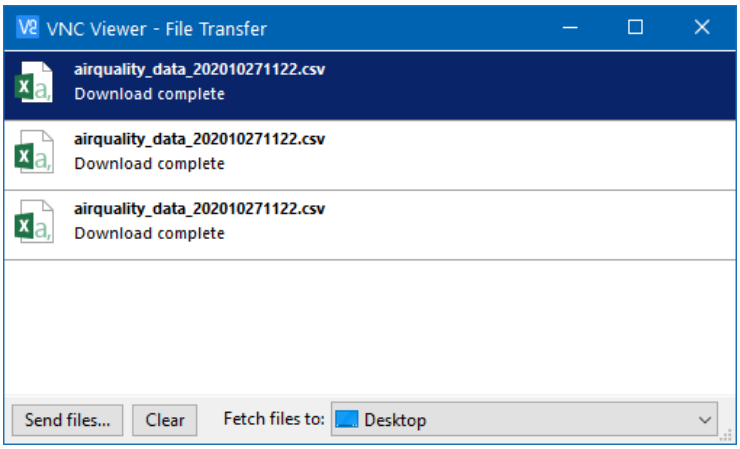


Figure 5:

6. There should be another popup, on your main computer, that will show download manager with your file in it. By default, RealVNC transfers files to your desktop.



3.1.2 Alternative Methods For PCs

Alternatively, there serveral other alternatives. For example, this webpage describes the issues and 5 not great options to get data.

After several attempts to create a cloud folder that you could upload, I found it was far to slow and unreliable. Thus, I recommend using Putty and the following line commands:

```
psftp> open Pi@XXX.XXX.XXX.XXX
```

and you will be prompted to enter the password. Once you are in, you can see your files by entering

```
psftp>ls *.csv
```

where all your csv files should be shown.

Then you'll need to change your local drive (the default one is write protected, creating a signficant obstacle. So, use

```
psftp>lpwd
```

to see your default directly. We are doing to use the download directory and we can change the directory using:

```
psftp>lcd c:/Downloads
```

Then we will get the files...

```
psftp>get airquality_data.csv
```

and it will be transfered to your local computer... ! Yay! It only took me 5 hours to figure out and hopefully for you less than 5 min.

3.1.3 FOR Macs

I can't find an easy solution for MACs, use the file transfer function in RealVNC.

3.2 Processing the data

We create a script to process the data and allow you to create a reasonably user-friendly dataframe to analyze the data.

3.3 Remove Double Quotes in CSV

However, before running the code below, we need to remove all the double quotes, which get in the way of how "csv" files are read.

Opened the document up in Rstdudio. Use the edit menu and select "search and replace". Search for double quotes (") and leave the place for "Replace" blank, i.e. don't enter anythning. Then replace all. Then you can read the csv into R.

```
filepath.csv = "/home/CAMPUS/mwl04747/github/EJnPi/data/Air_Quality.csv"
filepath.csv = "/home/CAMPUS/mwl04747/github/EJnPi/data/Complete_airquality_data2.csv"
filepath.csv = "/home/CAMPUS/mwl04747/github/EJnPi/data/201104_Complete_airquality.csv"
rawdata = read.csv(filepath.csv)

names(rawdata)= c("X1", "X2", "Month", "Day", "Hour",
    "Minute", "Second", "X3", "X4", "pm1_cf", "X5", "pm25_cf", "X6",
    "pm10_cf", "X7", "pm1", "X8", "pm25", "pm10.", "X9")

rawdata$pm1_cf = as.numeric(gsub('[)]', '', rawdata$pm1_cf))

## Warning:  NAs introduced by coercion

rawdata$pm25_cf = as.numeric(gsub('[)]', '', rawdata$pm25_cf))

## Warning:  NAs introduced by coercion

rawdata$pm10_cf = as.numeric(gsub('[)]', '', rawdata$pm10_cf))

## Warning:  NAs introduced by coercion

as.Date(with(rawdata, paste("2020", Month, Day,sep="-")), "%Y-%m-%d")[1]

## [1] "2020-10-21"

library(lubridate)

##
## Attaching package:  'lubridate'
## The following object is masked from 'package:base':
##
##    date

rawdata$DateTime = with(rawdata, ymd_hms(paste("2020", Month,
  Day, Hour, Minute, Second, sep= '-')))

## Warning:  2564 failed to parse.
```

As always, I check my data!

```
rawdata[sample(1:nrow(rawdata), 5),] # random 10 rows, all columns
```

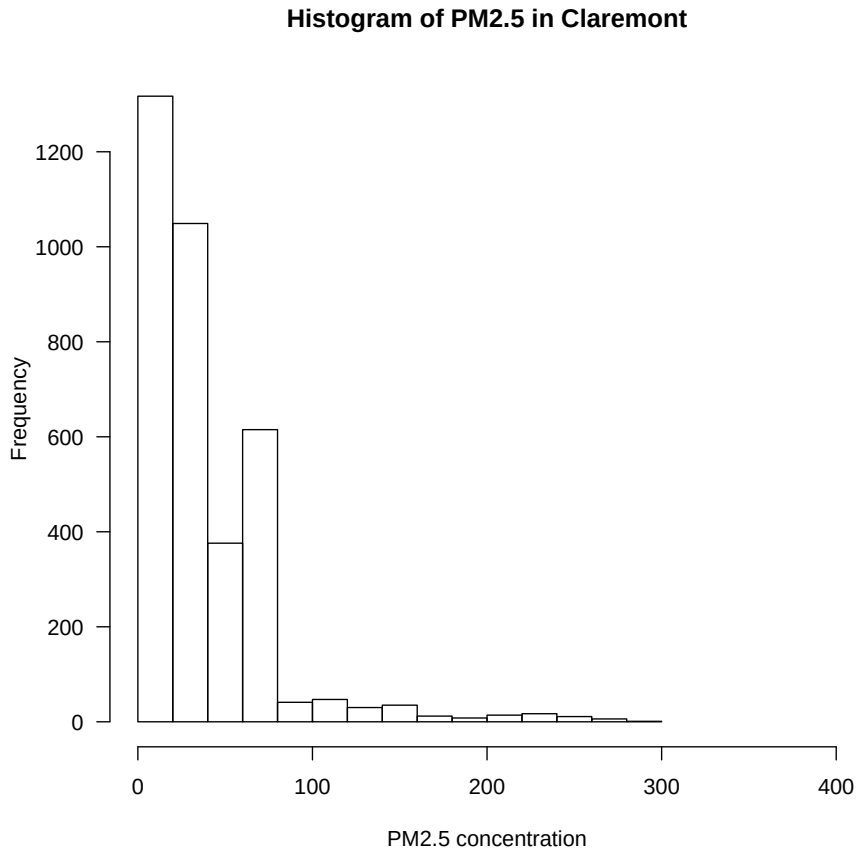
```
##                            X1                       X2 Month Day Hour Minute
## 1991                                                    NA  NA   NA     NA
## 745  OrderedDict([('DateTime'  datetime.datetime(2020    10  22    5     26
## 1263 OrderedDict([('DateTime'  datetime.datetime(2020    10  24    2     48
## 3681 OrderedDict([('DateTime'  datetime.datetime(2020    10  30   13     12
## 1036 OrderedDict([('DateTime'  datetime.datetime(2020    10  23    7     50
##      Second        X3         X4 pm1_cf          X5 pm25_cf          X6 pm10_cf
## 1991     NA                          NA                  NA                  NA
## 745       3  910686))  ('pm1_cf'     32  ('pm25_cf'      69  ('pm10_cf'      80
## 1263      2  364166))  ('pm1_cf'     19  ('pm25_cf'      39  ('pm10_cf'      41
## 3681     26  749344))  ('pm1_cf'      3  ('pm25_cf'       5  ('pm10_cf'       5
## 1036     53  373274))  ('pm1_cf'     20  ('pm25_cf'      39  ('pm10_cf'      48
##           X7  pm1       X8 pm25    pm10.     X9            DateTime
## 1991                                                          <NA>
## 745   ('pm1'  27)  ('pm25'  49)  ('pm10'  64)]) 2020-10-22 05:26:03
## 1263  ('pm1'  19)  ('pm25'  35)  ('pm10'  41)]) 2020-10-24 02:48:02
## 3681  ('pm1'   3)  ('pm25'   5)  ('pm10'   5)]) 2020-10-30 13:12:26
## 1036  ('pm1'  20)  ('pm25'  35)  ('pm10'  47)]) 2020-10-23 07:50:53
```

```
# Remove Variables
cleandata = subset(rawdata, select=c(DateTime, pm1_cf, pm25_cf, pm10_cf))
```

3.4 Plot Data

As usual, it's a good idea to get a sense of the variability of the data.

```
hist(cleandata$pm25_cf, xlab="PM2.5 concentration",
    main="Histogram of PM2.5 in Claremont", las=1, xlim=c(0,400))
```



In my case, I have a very odd bimodal distribution and an long tail to boot. So, I am going plot on a time series to see if I can figure out what in the world is going on (Figure 6).

Except for some breaks in the data, we can certainly see a diurnal pattern, where the concentrations seem highest after midnight.

Let's see if we can add times to the plot with some rectancles for "night time"! Furthermore, let's see how different size particles behave (Figure 7).

Figure **??** is not perfect. Here's a few issues:

- Dates and Days would be ideal
- y-axis isn't quite correct
- moving average might be nice.
- remove "dates" from top two figures because it's redundant.

For now, this is good enough – and I'll keep working on this. Overall, Claremont's Air Quality was terrible on Wednesday night and Thursday morning.

```
plot(pm25_cf~DateTime, data=cleandata, pch=20, cex=.5)
```

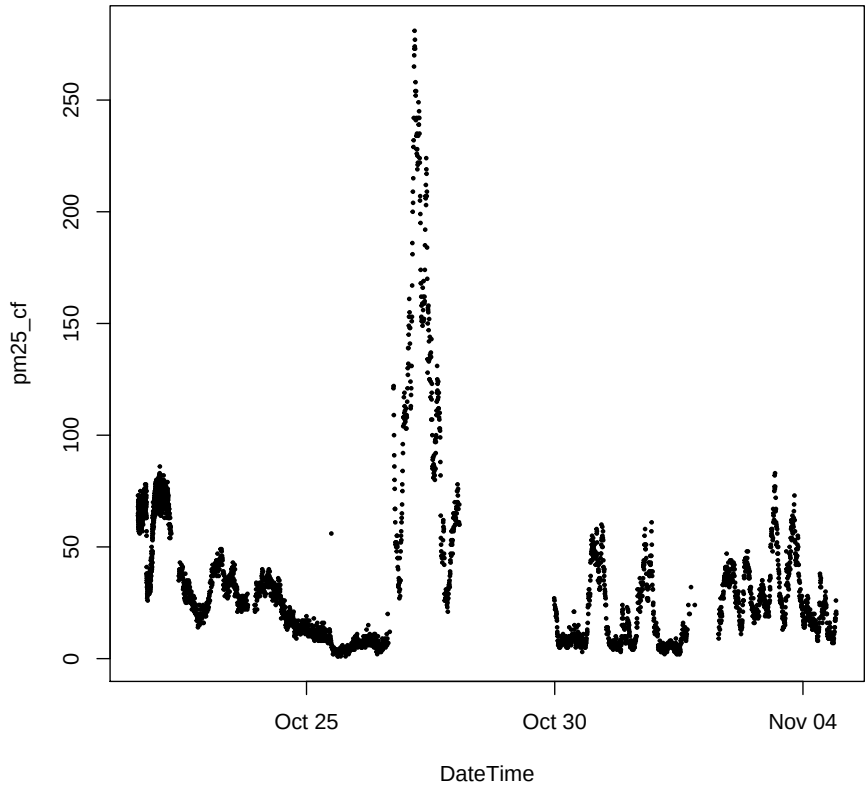


Figure 6: Draft Image – needs work, but intersting patterns!

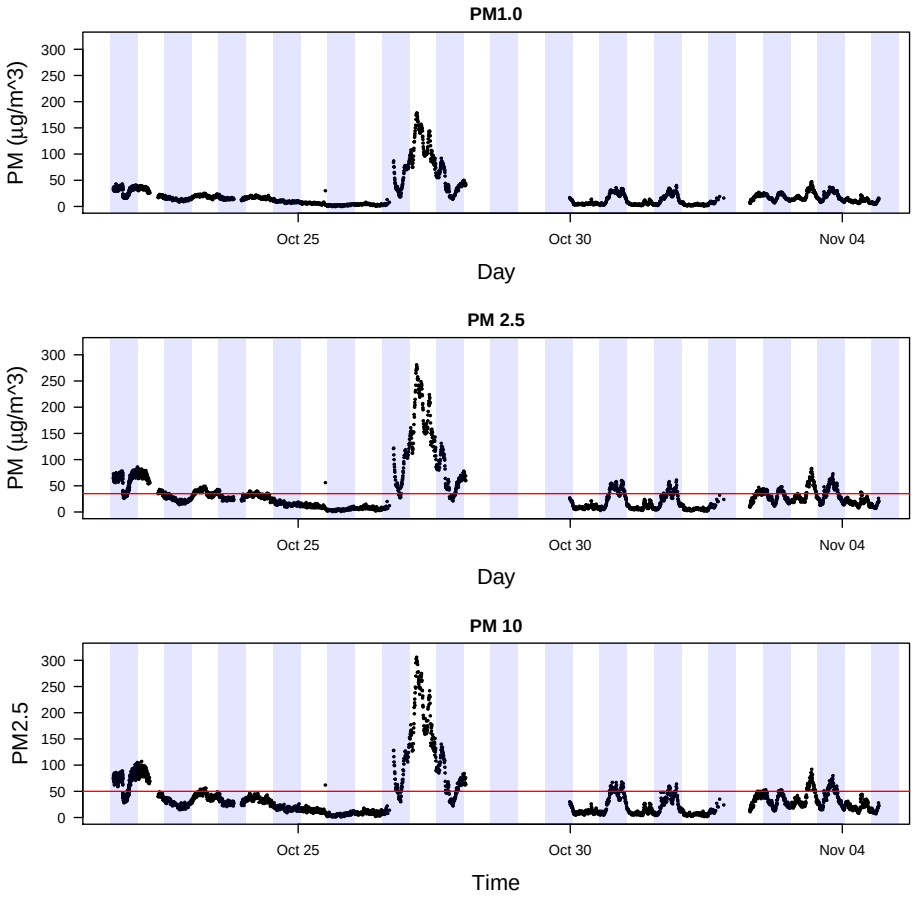


Figure 7: $PM_{1.0}$, $PM_{2.5}$ and $PM_{2.5}$ in Claremont California between Oct 21–Oct. 28, PM2.5 24 hour standard = 35, and PM10 24 hour standard = 50.

4 Comparing Data to EPA Stations

4.1 Using previous collected data

```
# TBD
```

4.2 Comparing μ to EPA Data

The population mean (monthly average from the EPA data) is a parameter estimate. We can determine if the values you collected fall within the parameter estimate and it's 95% confidence intervals.

```
mean(cleandata$pm1_cf)

## [1] NA
```

Okay, now let's talk about signficant figures. The Pi only reports to the nearest whole value. Read about the rules in Wikipedia – https://en.wikipedia.org/wiki/Significant_figures. In general, we should report to the whole number as well. However, some statiticians suggest that the mean and standard deviation can include one extra significant figure, allowing the reader to see the "directionality" of the mean toward the precision. However, for our purposes, I suggest we follow the standard rule and round to the nearest whole number and avoid implying additional precision.

Finally, what happens if you get a missing value when using the function mean()? Likely, this is due to missing values in the vector! the mean function will not give you you an NA if NA is in the dataset. You can do a mean(DF, na.rm=T) to override the default...

fyi: same as sd()

R wants users to be sure to understand the implications of missing data! For example, what if we had missing data in cold temperatures – then our mean would be biased! When you decide to override the default, you want to be sure you are not introducing unknown biases...

5 More Intricate Analysis– Based on Slack and Zooom Conversations :-)

You could look to see if the there is a different during the time of the day or day of the week. You could move you sensor around in and outside your house and see if air quality is better / worse or the same....

We can help you with r code if you'd like to follow up with these additional questions.

5.1 Removing Missing Values

Our Pi code seems to create missing data when the sensor doesn't respond in a timely fashion.

```
summary(cleandata)

##     DateTime                       pm1_cf          pm25_cf
##  Min.   :2020-10-21 14:32:25   Min.   :  0.00   Min.   :  1.00
##  1st Qu.:2020-10-22 22:27:17   1st Qu.:  7.00   1st Qu.: 13.00
##  Median :2020-10-26 09:43:06   Median : 17.00   Median : 29.00
##  Mean   :2020-10-27 10:31:56   Mean   : 22.48   Mean   : 38.87
##  3rd Qu.:2020-10-31 18:07:44   3rd Qu.: 32.00   3rd Qu.: 57.50
##  Max.   :2020-11-04 16:09:13   Max.   :178.00   Max.   :281.00
```

```
##  NA's   :2564                  NA's   :2574     NA's   :2574
##     pm10_cf
##  Min.   :  1.00
##  1st Qu.: 14.00
##  Median : 32.00
##  Mean   : 43.56
##  3rd Qu.: 64.00
##  Max.   :306.00
##  NA's   :2574
```

In my case, I have 2564 missing values! So, here's code to remove these missing values. Be sure you have not created a biased dataset by removing values that are systematically changing the accuracy!

```
cleandata = na.omit(cleandata)
summary(cleandata)
```

```
##     DateTime                      pm1_cf          pm25_cf
##  Min.   :2020-10-21 14:32:25   Min.   :  0.00   Min.   :  1.00
##  1st Qu.:2020-10-22 22:14:44   1st Qu.:  7.00   1st Qu.: 13.00
##  Median :2020-10-26 09:15:25   Median : 17.00   Median : 29.00
##  Mean   :2020-10-27 10:13:20   Mean   : 22.48   Mean   : 38.87
##  3rd Qu.:2020-10-31 17:23:24   3rd Qu.: 32.00   3rd Qu.: 57.50
##  Max.   :2020-11-04 16:09:13   Max.   :178.00   Max.   :281.00
##     pm10_cf
##  Min.   :  1.00
##  1st Qu.: 14.00
##  Median : 32.00
##  Mean   : 43.56
##  3rd Qu.: 64.00
##  Max.   :306.00
```

5.2 Subset for Dates or Times

Let's say I want to compare day and night times. Let's say we compare 6:01 AM to 6 PM and 6:01 PM to 6:00 AM.

I use a simple criteria to select data based on the hour – using the libridate library.

```
library(lubridate)
str(cleandata)
```

```
## 'data.frame': 3579 obs. of  4 variables:
##  $ DateTime: POSIXct, format: "2020-10-21 14:32:25" "2020-10-21 14:33:26" ...
##  $ pm1_cf  : num  32 36 33 35 31 34 35 34 33 35 ...
```

```
##  $ pm25_cf : num  64 73 65 68 62 68 70 67 61 63 ...
##  $ pm10_cf : num  75 77 73 80 73 82 78 79 75 68 ...
##  - attr(*, "na.action")= 'omit' Named int  79 93 104 176 180 212 246 325 339 377 ...
##   ..- attr(*, "names")= chr  "79" "93" "104" "176" ...

day = with(cleandata,
     cleandata[hour(DateTime)> 6 &
               hour(DateTime)<= 18,])

night = with(cleandata,
     cleandata[hour(DateTime)<= 6 &
               hour(DateTime)< 18,])
```

Figure 8 suggests that night time PM2.5 is higher, but let's check with a statistical test in section 5.4.

Alternatively, let's select a period where a fire near Diamond Bar caused major air quality issue in Claremont.

```
str(cleandata)

## 'data.frame': 3579 obs. of  4 variables:
##  $ DateTime: POSIXct, format: "2020-10-21 14:32:25" "2020-10-21 14:33:26" ...
##  $ pm1_cf  : num  32 36 33 35 31 34 35 34 33 35 ...
##  $ pm25_cf : num  64 73 65 68 62 68 70 67 61 63 ...
##  $ pm10_cf : num  75 77 73 80 73 82 78 79 75 68 ...
##  - attr(*, "na.action")= 'omit' Named int  79 93 104 176 180 212 246 325 339 377 ...
##   ..- attr(*, "names")= chr  "79" "93" "104" "176" ...

fire = subset(cleandata,
  subset=
    DateTime >= as.POSIXct("2020-10-26 04:10:00", tz="PST") &
    DateTime <= as.POSIXct("2020-10-28 00:00:00", tz="PST"))
str(fire)

## 'data.frame': 453 obs. of  4 variables:
##  $ DateTime: POSIXct, format: "2020-10-26 04:13:42" "2020-10-26 04:19:09" ...
##  $ pm1_cf  : num  3 4 4 4 5 5 5 5 4 6 ...
##  $ pm25_cf : num  6 6 5 10 9 8 9 7 10 13 ...
##  $ pm10_cf : num  6 10 5 10 12 11 10 7 16 18 ...
```

```
boxplot(day$pm25_cf, night$pm25_cf, names=c("Day", "Night"), las=1)
```

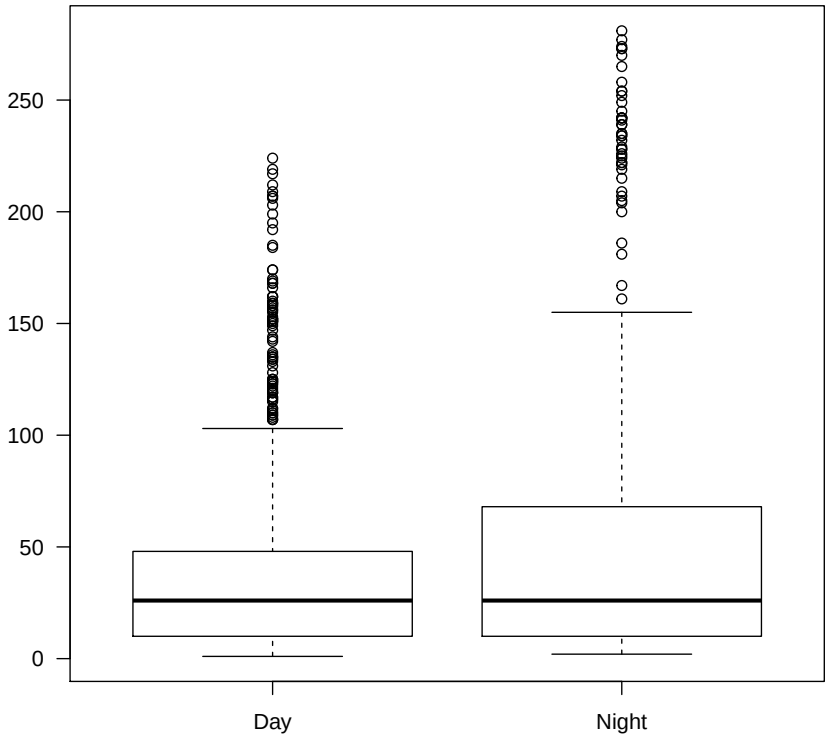
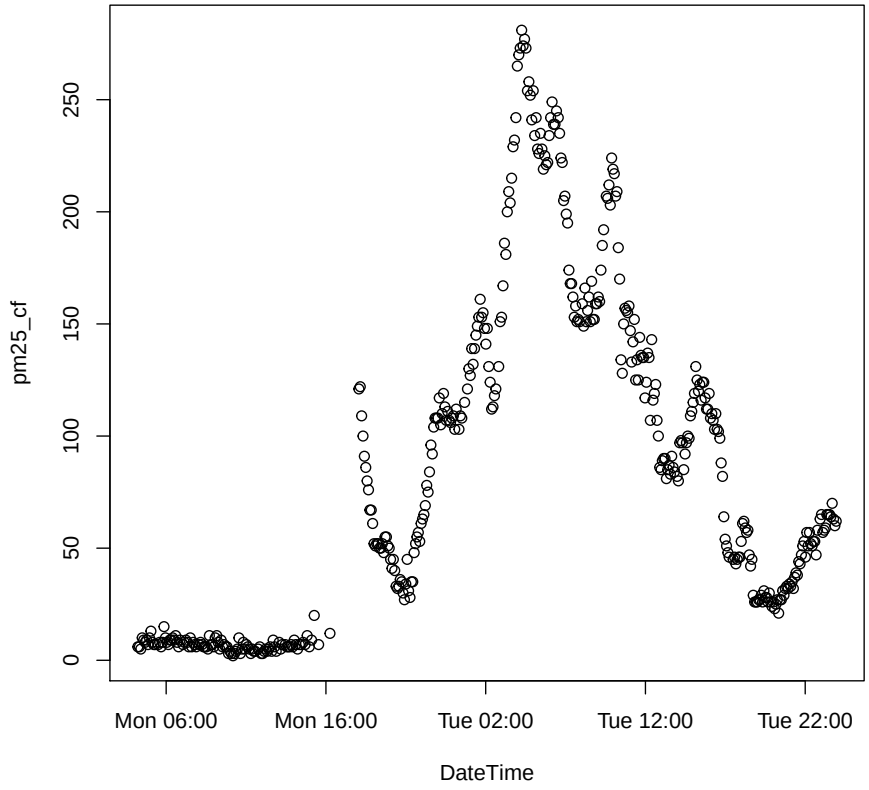


Figure 8: PM2.5 at day and night.



We might think of this as an event that might need to be analzyed in a different way!

5.3 Calculate 5/30 min averages

How would you make a dataset of five minute intervals? We can use the dplyr library.

Then we can use the function cut(), to create groups (based on time windows – interestingly enough reponds to various word phrased for time, where "5 min" for five minute windows and "30 min" for 30 minute windows.

```
str(cleandata)

## 'data.frame': 3579 obs. of  4 variables:
##  $ DateTime: POSIXct, format: "2020-10-21 14:32:25" "2020-10-21 14:33:26" ...
##  $ pm1_cf  : num  32 36 33 35 31 34 35 34 33 35 ...
##  $ pm25_cf : num  64 73 65 68 62 68 70 67 61 63 ...
```

```
##  $ pm10_cf : num  75 77 73 80 73 82 78 79 75 68 ...
##  - attr(*, "na.action")= 'omit' Named int  79 93 104 176 180 212 246 325 339 377 ...
##   ..- attr(*, "names")= chr  "79" "93" "104" "176" ...

require(dplyr)

## Loading required package:  dplyr
##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:lubridate':
##
##    intersect, setdiff, union
## The following objects are masked from 'package:stats':
##
##    filter, lag
## The following objects are masked from 'package:base':
##
##    intersect, setdiff, setequal, union

mean5min = cleandata %>%
  group_by(DateTime = cut(DateTime, breaks="5 min")) %>%
  summarize(pm25_cf = mean(pm25_cf))
```

Or for 30 min means...

```
require(dplyr)
mean30min = cleandata %>%
  group_by(DateTime = cut(DateTime, breaks="30 min")) %>%
  summarize(pm25_cf = mean(pm25_cf))
```

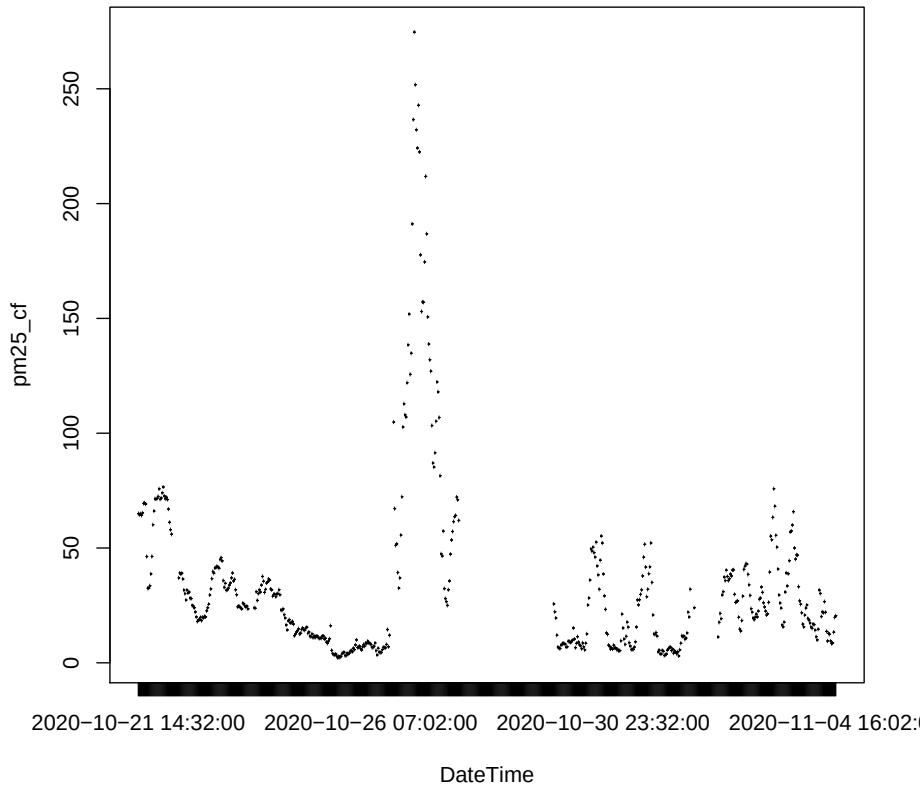
So, what did we create? Let's look at the structure and head...

```
head(mean30min)

## # A tibble: 6 x 2
##   DateTime            pm25_cf
##   <fct>                 <dbl>
## 1 2020-10-21 14:32:00    65.0
## 2 2020-10-21 15:02:00    64.5
## 3 2020-10-21 15:32:00    64.9
## 4 2020-10-21 16:02:00    64.2
## 5 2020-10-21 16:32:00    65.3
## 6 2020-10-21 17:02:00    69.4

str(mean30min)
```

```
## tibble [546 x 2] (S3: tbl_df/tbl/data.frame)
##  $ DateTime: Factor w/ 676 levels "2020-10-21 14:32:00",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ pm25_cf : num [1:546] 65 64.5 64.9 64.2 65.3 ...
```

This is a rather unique type of data set – we are no looking at a "tibble", which is a particular type of data.frame that is structured as a table too. We also have lost our DateTime date formats, so this will not graph easily! Ugh!!

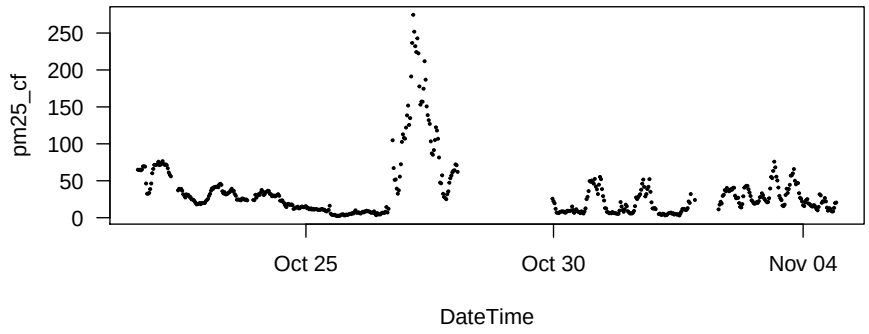
```
plot(pm25_cf ~ DateTime, data=mean30min)
```

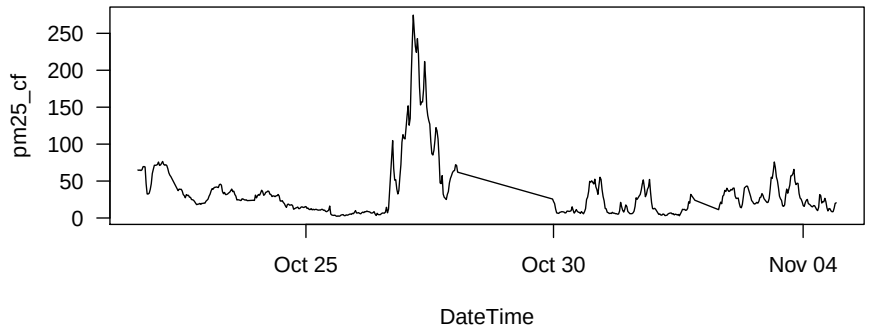


As a factor, we can't control the axis display very well – and it looks like crap! The type of object doesn't seem to be bothering us much. So, let's march forward.

```
mean30min$DateTime = as.POSIXct(mean30min$DateTime, format="%Y-%m-%d %H:%M:%S")
```

```
par(mfrow=c(2,1), las=1)
plot(pm25_cf ~ DateTime, data=mean30min, ty="p", pch=20, cex=.4)
plot(pm25_cf ~ DateTime, data=mean30min, ty="l")
```





Note: when I make this a line graph (ty="l"), the missing data are a problem! I will see how to fix that is someone asks, but I do like the display better than the point graphic.

let's see if we can limit the dates... (Figure 9), where I have used a second window to look at the fire in Diamond Bar and then put lines for the date changes.

```
par(mfrow=c(1,2), las=1)
plot(pm25_cf ~ DateTime, data=mean30min, ty="l", ylab="Conc", xlab="Date")
plot(pm25_cf ~ DateTime, data=mean30min, ty="p",
  pch=20, cex=.6, xlim=c(as.POSIXct("2020-10-26 20:00:00"),
  as.POSIXct("2020-10-27 23:59:59")), ylab="Conc",
  xaxt = "n", xlab="Time", col="blue")

r <- as.POSIXct(round(range(as.POSIXct("2020-10-26 00:00:00"),
  as.POSIXct("2020-10-28 00:00:00")), "hours"))

axis.POSIXct(3, at = seq(r[1], r[2],
  by = "day"), format = "%D")

abline(v=as.POSIXct(c("2020-10-26 00:00:00",
  "2020-10-27 00:00:00", "2020-10-28 00:00:00")), lwd=2, col="brown")
axis.POSIXct(1, at = seq(r[1], r[2],
  by = "hour"), format = "%H")
```

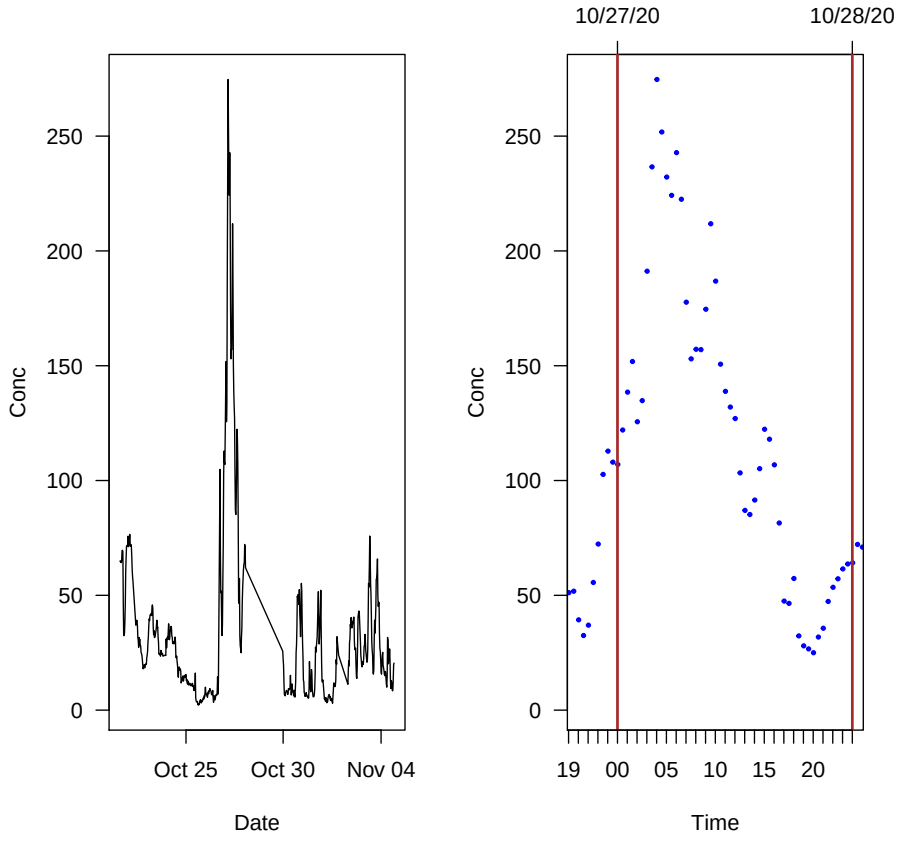


Figure 9: Focused Figure on PM change during a fire.

5.4 Test test – Comparing Two Populations

Now, let's formally test the null hypothesis that there is no difference between two populations of data – night PM2.5 and day PM2.5

```
t.test(day$pm25_cf, night$pm25_cf, pair=FALSE)

##
##  Welch Two Sample t-test
##
## data:  day$pm25_cf and night$pm25_cf
## t = -4.4286, df = 1669, p-value = 1.01e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.234988  -4.337892
## sample estimates:
## mean of x mean of y
##  35.68789  43.47433
```