Working with Data in R

Mauricio Romero
(Based on Nick C. Huntington-Klein's notes)

Working with Data

- R is all about working with data!
- data.frames:
  - data.frames are an object type
  - Most of the time, you'll be doing calculations using them
  - Conceptually, data.frames are basically spreadsheets
  - Technically, they're a list of vectors

data.frame

- It's a collection of vectors of the same length

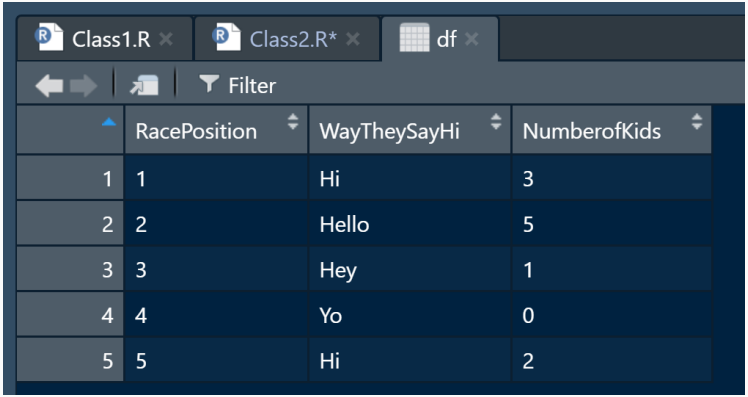
- (Note the use of $=$ here, not $<$-)

```
df <- data.frame(
RacePosition = 1:5,
WayTheySayHi = as.factor(c('Hi','Hello','Hey','Yo','Hi')),
NumberofKids = c(3,5,1,0,2))
df
```

```
> df <- data.frame(
+     RacePosition = 1:5,
+     WayTheySayHi = as.factor(c('Hi','Hello','Hey','Yo','Hi')),
+     NumberofKids = c(3,5,1,0,2))
> df
  RacePosition WayTheySayHi NumberofKids
1            1           Hi            3
2            2        Hello            5
3            3          Hey            1
4            4           Yo            0
5            5           Hi            2
> 
```

Looking Over Data

- Now that we have our data, how can we take a look at it?
- We name it in the Console and look at the whole thing
    - Usually too much data
- Clicking on it in Environment to open it up



	RacePosition	WayTheySayHi	NumberofKids
1	1	Hi	3
2	2	Hello	5
3	3	Hey	1
4	4	Yo	0
5	5	Hi	2

Glancing at Data

- What if we just want a quick overview?
    - Arrow in the Environment tab
    - ‘head()‘ (look at the head of the data - first six rows)
    - ‘str()‘ (structure)

```
str(df)
```

```
> str(df)
'data.frame':   5 obs. of  3 variables:
 $ RacePosition: int  1 2 3 4 5
 $ WayTheySayHi: Factor w/ 4 levels "Hello","Hey",..: 3 1 2 4 3
 $ NumberofKids: num  3 5 1 0 2
>
```

What do we want to know about our data?

- What is this data **of**? (won't get that with 'str()')
- Data types
- The kinds of values it takes
- How many observations
- Variable names
- Summary statistics and observation level (we'll get to that later)

Getting at Data

- Now we have a data frame, ‘df‘. How do we use it?
- We can pull the vectors back out with ‘$‘! Note autocompletion of variable names.
- We can treat it just like the vectors we had before

```
df$NumberofKids
df$NumberofKids[2]
df$NumberofKids >= 3
```

```
> df
  RacePosition WayTheySayHi NumberofKids
1            1           Hi            3
2            2        Hello            5
3            3          Hey            1
4            4           Yo            0
5            5           Hi            2
> View(df)
> View(df)
> View(df)
> str(df)
'data.frame':   5 obs. of  3 variables:
 $ RacePosition: int  1 2 3 4 5
 $ WayTheySayHi: Factor w/ 4 levels "Hello","Hey",..: 3 1 2 4 3
 $ NumberofKids: num  3 5 1 0 2
> df$NumberofKids
[1] 3 5 1 0 2
> df$NumberofKids[2]
[1] 5
> df$NumberofKids >= 3
[1]  TRUE  TRUE FALSE FALSE FALSE
>
```

Quick Note

- There are actually many many ways to do this

- All these are equivalent:

```
df$NumberofKids >= 3
df[,3] >= 3
df[,'NumberofKids']>=3
```

```
> df$NumberofKids >= 3
[1]  TRUE  TRUE FALSE FALSE FALSE
> df[,3] >= 3
[1]  TRUE  TRUE FALSE FALSE FALSE
> df[,'NumberofKids']>=3
[1]  TRUE  TRUE FALSE FALSE FALSE
>
```

We can run the same calculations on these vectors as we were doing before

```
mean(df$RacePosition)
df$WayTheySayHi[4]
sum(df$NumberofKids <= 1)
```

```
> mean(df$RacePosition)
[1] 3
> df$WayTheySayHi[4]
[1] Yo
Levels: Hello Hey Hi Yo
> sum(df$NumberofKids <= 1)
[1] 2
>
```

Practice

- Create ‘df2 <- data.frame(a = 1:20, b = 0:19*2,‘ ‘c = sample(101:200,20,replace=TRUE))‘
- What is the average of ‘c‘?
- What is the sum of ‘a‘ times ‘b‘?
- Did you get any values of ‘c‘ 103 or below? (make a logical)
- What is on the 8th row of ‘b‘?
- How many rows have ‘b‘ above 10 AND ‘c‘ below 150?

Practice Answers

```
df2 <- data.frame(
a = 1:20,
b = 0:19*2,
c = sample(101:200,20,replace=TRUE))

mean(df2$c)

sum(df2$a*df2$b)

sum(df2$c <= 103) > 0

df2$b[8]

sum(df2$b > 10 & df2$c < 150)
```

```
> df2 <- data.frame(
+     a = 1:20,
+     b = 0:19*2,
+     c = sample(101:200,20,replace=TRUE))
>
> mean(df2$c)
[1] 147.55
>
> sum(df2$a*df2$b)
[1] 5320
>
> sum(df2$c <= 103) > 0
[1] TRUE
>
> df2$b[8]
[1] 14
>
> sum(df2$b > 10 & df2$c < 150)
[1] 11
>
```

The Importance of Rows

- So far we've taken data frames and pulled the vectors (columns) back out
- So... why not just stick with the vectors?
- We're not just interested in the columns one at a time
- We want to keep track of how the **row** (an observation)
- Goal: How do variables (columns) relate to each other for the same observation (row)

The Importance of Rows

- Going back to ‘df‘, that fourth row says that
  - The person in the fourth position...
  - Says hello by saying "Yo"
  - And has no kids
- Is there a relationship between having kids and your position in the race?
- Or a relationship between the number of kids relates to how you say hello?

Working With Data Frames

- We can manipulate data frames!
- Let's figure out how we can:
  - Create new variables
  - Change variables
  - Rename variables
- It's very common that you'll have to work with data a little before analyzing it
- "data cleaning" is super important and a big part of statistical analysis

Creating New Variables

- data.frames are just lists of vectors
- So create a vector and tell R where in that list to stick it!
- Use descriptive names so you know what the variable is

```
df$State <- c('Alaska','California',
                          'California','Maine',
                          'Florida')
df
```

```
> df$State <- c('Alaska','California',
+               'California','Maine',
+               'Florida')
> df
  RacePosition WayTheySayHi NumberofKids      State
1            1          Hi            3     Alaska
2            2       Hello            5 California
3            3         Hey            1 California
4            4          Yo            0      Maine
5            5          Hi            2    Florida
> 
```

Another approach - DPLYR and Tidyverse

- We just saw the base-R way to do it
- Can use **dplyr** (data pliers) for data manipulation instead
- dplyr syntax is inspired by SQL
    - Learning dplyr will give you a leg up if you want to learn SQL later
    - Plus some people find it more intuitive/better

Packages

- tidyverse isn't a part of base R. It's in a package, so we'll need to install it
- We can install packages using 'install.packages('nameofpackage')'

```
install.packages('tidyverse')
```

- We can then check whether it's installed in the Packages tab

Packages

- Before we can use it we must then use the ‘library()‘ command to open it up
- Need to run ‘library()‘ every time we open up R if we want to use the package

```
library(tidyverse)
```

- There are thousands of useful packages for R, and we’re going to be using a few!
- tidyverse will just be our first of many
- Google R package “X” to look for packages that do “X”

Variable creation with dplyr

- The **mutate** command will "mutate" our data frame to have a new column in it
- The pipe '%>%' says "take df and send it to that mutate command to use"
- Or we can stick the data frame itself in the 'mutate' command
- Thus these two are equivalent:

```
library(tidyverse)

df1 <- df %>% mutate(State = c('Alaska','California',
                               'California','Maine','Florida'))

df2 <- mutate(df,State = c('Alaska','California',
                           'California','Maine','Florida'))

identical(df1,df2)

df <- df1
```

```
>
> #Creating a new variable with mutate/tidiverse
>
> df1 <- df %>% mutate(State = c('Alaska','California',
+                              'California','Maine','Florida'))
>
> df2 <- mutate(df,State = c('Alaska','California',
+                            'California','Maine','Florida'))
>
> identical(df1,df2)
[1] TRUE
>
```

Creating New Variables

- We can use all the tricks we already know about creating vectors

- We can create multiple new variables in one mutate command

```
df <- df %>% mutate(
MoreThanTwoKids = NumberofKids > 2,
One = 1,
KidsPlusPosition = NumberofKids + RacePosition)

df
```

```
> df <- df %>% mutate(
+   MoreThanTwoKids = NumberofKids > 2,
+   One = 1,
+   KidsPlusPosition = NumberofKids + RacePosition)
>
> df
  RacePosition WayTheySayHi NumberofKids      State MoreThanTwoKids One KidsPlusPosition
1            1          Hi            3     Alaska            TRUE   1                4
2            2       Hello            5 California            TRUE   1                7
3            3         Hey            1 California           FALSE   1                4
4            4          Yo            0      Maine           FALSE   1                4
5            5          Hi            2    Florida           FALSE   1                7
>
```

Manipulating Variables

- We can't really **change** variables, but we can overwrite them!
- We can drop variables with '-' in the dplyr 'select' command
- Note we chain multiple dplyr commands with '%>%'

```
df <- df %>%
  select(-KidsPlusPosition,-WayTheySayHi,-One) %>%
  mutate(State = as.factor(State),
         RacePosition = RacePosition - 1)
df$State[3] <- 'Alaska'
str(df)
```

```
> #manipulating some variables
> df <- df %>%
+   select(-KidsPlusPosition,-WayTheySayHi,-One) %>%
+   mutate(State = as.factor(State),
+          RacePosition = RacePosition - 1)
> df$State[3] <- 'Alaska'
> str(df)
'data.frame':   5 obs. of  4 variables:
 $ RacePosition   : num  0 1 2 3 4
 $ NumberofKids   : num  3 5 1 0 2
 $ State          : Factor w/ 4 levels "Alaska","California",..: 1 2 1 4 3
 $ MoreThanTwoKids: logi  TRUE TRUE FALSE FALSE FALSE
>
```

Renaming Variables

- Sometimes it will make sense to change the names of the variables we have.
- Names are stored in ‘names(df)‘ which we can edit directly
- Or the ‘rename()‘ command in dplyr has us covered

```
names(df)
#two ways of renaming
#names(df) <- c('Pos','Num.Kids','State','mt2Kids')

df <- df %>% rename(Pos = RacePosition, Num.Kids=NumberofKids,
                    mt2Kids = MoreThanTwoKids)

names(df)
```

```
> #Renaming variaables
> names(df)
[1] "RacePosition"    "NumberofKids"    "State"           "MoreThanTwoKids"
> #two ways of renaming
>
> #names(df) <- c('Pos','Num.Kids','State','mt2Kids')
>
> df <- df %>% rename(Pos = RacePosition, Num.Kids=NumberofKids,
+                     mt2Kids = MoreThanTwoKids)
>
> names(df)
[1] "Pos"      "Num.Kids" "State"    "mt2Kids"
>
```

Practice

- Create a data set ‘data‘ with three variables: ‘a‘ is all even numbers from 2 to 20, ‘b‘ is ‘c(0,1)‘ over and over, and ‘c‘ is any ten-element numeric vector of your choice.
- Rename them to ‘EvenNumbers‘, ‘Treatment‘, ‘Outcome‘.
- Add a logical variable called Big that’s true whenever EvenNumbers is greater than 15
- Increase Outcome by 1 for all the rows where Treatment is 1.
- Create a logical AboveMean that is true whenever Outcome is above the mean of Outcome.
- Display the data structure

Practice Answers

```
data <- data.frame(a = 1:10*2,
                   b = c(0,1),
                   c = sample(1:100,10,replace=FALSE)) %>%
  rename(EvenNumbers = a, Treatment = b, Outcome = c)

data <- data %>%
  mutate(Big = EvenNumbers > 15,
         Outcome = Outcome + Treatment,
         AboveMean = Outcome > mean(Outcome))

str(data)
```

```
> #Some practice
> data <- data.frame(a = 1:10*2,
+                    b = c(0,1),
+                    c = sample(1:100,10,replace=FALSE)) %>%
+   rename(EvenNumbers = a, Treatment = b, Outcome = c)
>
> data <- data %>%
+   mutate(Big = EvenNumbers > 15,
+          Outcome = Outcome + Treatment,
+          AboveMean = Outcome > mean(Outcome))
>
> str(data)
'data.frame':   10 obs. of  5 variables:
 $ EvenNumbers: num  2 4 6 8 10 12 14 16 18 20
 $ Treatment  : num  0 1 0 1 0 1 0 1 0 1
 $ Outcome    : num  44 61 29 33 58 37 48 80 97 50
 $ Big        : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ AboveMean  : logi  FALSE TRUE FALSE FALSE TRUE FALSE ...
>
```

Other Ways to Get Data

- Of course, most of the time we aren't making up data
- We get it from the real world!
- Two main ways to do this are the ‘data()‘ function in R
- Or reading in files, usually with one of the ‘read‘ commands like ‘read.csv()‘

data()

- R has many baked-in data sets, and more in packages!
- Just type in ‘data(‘ and see what options it autocompletes
- We can load in data and look at it
- Many of these data sets have ‘help‘ files too

```
data(LifeCycleSavings)
help(LifeCycleSavings)
head(LifeCycleSavings)
```

Intercountry Life-Cycle Savings Data

Description

Data on the savings ratio 1960–1970.

Usage

```
LifeCycleSavings
```

Format

A data frame with 50 observations on 5 variables.

[,1]	sr	numeric aggregate personal savings
[,2]	pop15	numeric % of population under 15
[,3]	pop75	numeric % of population over 75
[,4]	dpi	numeric real per-capita disposable income
[,5]	ddpi	numeric % growth rate of dpi

Details

Under the life-cycle savings hypothesis as developed by Franco Modigliani, the savings ratio (aggregate personal saving divided by disposable income) is explained by per-capita disposable income, the percentage rate of change in per-capita disposable income, and two demographic variables: the percentage of population less than 15 years old and the percentage of the population over 75 years old. The data are averaged over the decade 1960–1970 to remove the business cycle or other short-term fluctuations.

Source

The data were obtained from Belsley, Kuh and Welsch (1980). They in turn obtained the data from Sterling (1977).

```
> data(LifeCycleSavings)
> help(LifeCycleSavings)
> head(LifeCycleSavings)
             sr pop15 pop75     dpi ddpi
Australia 11.43 29.35  2.87 2329.68 2.87
Austria   12.07 23.32  4.41 1507.99 3.93
Belgium   13.17 23.80  4.43 2108.47 3.82
Bolivia    5.75 41.89  1.67  189.13 0.22
Brazil    12.88 42.19  0.83  728.47 4.56
Canada     8.79 31.72  2.85 2982.88 2.43
> 
```

read

- Often there will be data files on the internet or your computer
- You can read this in with one of the many ‘read‘ commands, like ‘read.csv‘
- “csv” is a very basic spreadsheet format stored in a text file
  - You can create it from Excel or Sheets (or just write it)
- There are different ‘read‘ commands for different file types
- Make sure your working directory is set to where the data is
  - Tell R what the path is!
  - Use “setwd(‘mypath’)”
  - See example at the top of the .R files I posted
  - Can also read data from an url directly
- Documentation will usually be in a different file

```
datafromCSV <- read.csv('mydatafile.csv')
```

Practice

- Use ‘data()‘ to open up a data set
  - Any data set that is a ‘data.frame‘
  - Try again if you get something else
- Use ‘str()‘ and ‘help()‘ to examine that data set
- What is it data *of* (help file)? How was it collected and what are the variables?
- What kinds of variables/values do they have (‘str()‘ and ‘head()‘)?
- Create a new variable using the variables that are already in there
- Take a mean of one of the variables
- Rename a variable to be more descriptive based on what you saw in ‘help()‘.

Recap

- Construct ‘data.frame‘s by making them with ‘data.frame()‘, or reading in data with ‘data()‘ or ‘read.csv‘
- ‘data.frame‘s are a list of vectors - we know vectors!
- We can pull the vectors back out with ‘$‘
- We can assign new variables, or update them, using ‘$‘ as well

Subsetting your data

- Selecting only **part** of the data that we have (i.e., **subset** the data)
- Why would we want to do this?
- We might be interested in how a variable **differs** for two different groups
- Or how one variable is **related** to another
- Or how those relationships differ for different groups
- We might only be interested in a particular group

Subset

- ‘filter()‘ and ‘select()‘ allow you to pick certain parts of your data
- You can select certain rows/observations using logicals with ‘filter()‘
- And you can select certain columns/variables with ‘select()‘
- The syntax is:

```
data.frame %>% filter(logical.for.rows)
filter(data.frame, logical.for.rows)
data.frame %>% select(variables,you,want)
select(data.frame,variables,you,want)
```

Example

- Let's read in some data on pollution in CDMX
- Mexico has lots of open data sources
  - Mexico city pollution: `http://www.aire.cdmx.gob.mx/`
  - Mexico city crime: `https://datos.cdmx.gob.mx/explore/dataset/carpetas-de-investigacion-pgj-cdmx/`
  - More open data CDMX: `https://datos.cdmx.gob.mx/pages/home/`
  - More open data Mexico: `https://datos.gob.mx/`
  - A lot of cool data can be "webscrapped" or requested to INAI

Example

```
df <- read.csv(
'http://www.aire.cdmx.gob.mx/opendata/red_manual/red_manual_particulas_susp.csv',
skip=8)
str(df)
df$Date=as.Date(df$Date,format="%d/%m/%Y") #convert date, to date format
```

```
> #Download pollution data
> df <- read.csv('http://www.aire.cdmx.gob.mx/opendata/red_manual/red_manual_particulas_susp.csv',
+                skip=8,stringsAsFactors = F) #skip first 8 lines (how do I know 8 lines, open the original csv and check it out)
> str(df)
'data.frame':   37715 obs. of  5 variables:
 $ Date         : chr  "02/01/1989" "02/01/1989" "02/01/1989" "02/01/1989" ...
 $ cve_station  : chr  "CES" "MER" "PED" "TLA" ...
 $ cve_parameter: chr  "PM10" "PM10" "PM10" "PM10" ...
 $ value        : int  258 142 95 164 249 304 163 91 177 288 ...
 $ unit         : int  2 2 2 2 2 2 2 2 2 2 ...
>
```

Example

- Always look at the data before you use it!
- It has date as a string, but R has a special "date" data type
- More info on the data at `https://tinyurl.com/y6mhmo6f`

Example

- There is data for TSP, PM2.5 and PM10. What are thse? See the "'catalogo de parametros"
- Let's keep the data only for PM10

```
 #these produce the same result
dfsub1 <- df %>% filter(cve_parameter %in% c('PM10'))
dfsub2 <- df[df$cve_parameter=='PM10',]
dfsub <- dfsub1
```

- What is the mean PM10?
- How many observations do we have per station?

```
#what is the mean of PM10?
mean(dfsub$value)
#how many observations per station?
table(dfsub$cve_station)
```

```
> #Keep only Pm10 -- Two ways to do it
> dfsub1 <- df %>% filter(cve_parameter %in% c('PM10'))
> dfsub2 <- df[df$cve_parameter=='PM10',]
> #Ok lets just keep one of them
> dfsub <- dfsub1
> #what is the mean of PM10?
> mean(dfsub$value)
[1] 64.41394
> #how many observations per station?
> table(dfsub$cve_station)

 CES  HAN  LOM  LPR  MCM  MER  NEZ  PED  SHA  TLA  UIZ  XAL  XCH
1355  248 1164 1117  265 1834 1283 1802  902 1848 1232 1816  124
>
```

Example

- If we limit the data just to the station in Pedregal (PED), what's the mean?

```
mean(filter(df,cve_station %in% c('PED'))$value)
```

- What if we want to compare the stations to each other? We need to split off each station by itself

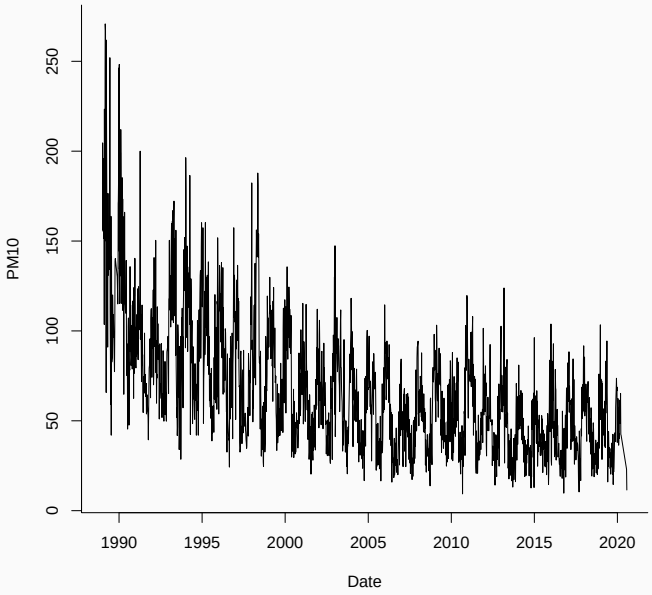
```
dfsub <- df %>% group_by(cve_station) %>%
  summarize(PM10 = mean(value))
```

```
> mean(filter(dfsub,cve_station %in% c('PED'))$value)
[1] 42.45061
> #Mean by station
> dfsub <- dfsub %>% group_by(cve_station) %>%
+   summarize(PM10 = mean(value))
> View(dfsub)
> View(dfsub)
>
```

Example

- What if we want to compare the evolution across time

```
date_mean <- df %>% group_by(Date) %>%
  summarize(PM10 = mean(value))
  plot(date_mean,type="l")
```


PM10
0
50
100
150
200
250
1990
1995
2000
2005
2010
2015
2020
Date

Example

- What questions does this answer?
  - What is average pollution by location?
  - How does average pollution evolve across time?
- What can't we answer yet?
  - What causes these differences across time and place [later!]

Subset

- Let's start by selecting rows from our data
- We do this by creating a logical
- 'filter' will choose all the observations for which that logical is true!
- Here's a logical to pick the 4 stations: 'cve_station %in% c('TLA','PED','MER','XAL')'
- This will be equal to 'TRUE' if the 'cve_station' variable is '%in%' that list of four stations I gave

```
df1<- dfsub %>% filter(cve_station %in% c('TLA','PED','MER','XAL'))
df2 <- dfsub[dfsub$cve_station %in% c('TLA','PED','MER','XAL'),]
identical(df1,df2)
```

Subset

- Subsetting for variables is easy! Just use ‘select()‘ with a vector or list of variables you want!
- Or you can do ‘-‘ a vector of variables you DON’T want!
- We don’t need “unit”, let’s get rid of it

```
str(df %>% select(Date,cve_station,cve_parameter,value,Year))
str(df %>% select(-c(unit)))
```

Subset

- We can do both at the same time, chaining one to the other

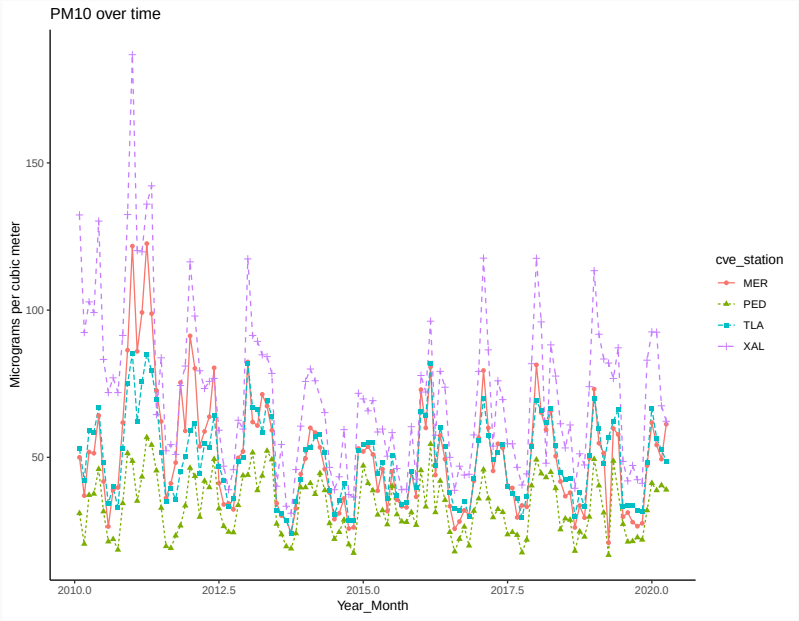
```
df %>% filter(Date == as.Date("2018-05-02",format="%Y-%m-%d")) %>%
  select(Date,cve_station,value)
```

Compare the changes over time across stations

```
#Lets just keep 4 stations, the ones with the most data
dfsub <- dfsub %>% filter(cve_station %in% c('TLA','PED','MER','XAL'))
#Create a month-year indicator
dfsub$Year <- as.numeric(format(as.Date(dfsub$Date), "%Y"))
dfsub$Month <- as.numeric(format(as.Date(dfsub$Date), "%m"))
dfsub$Year_Month=dfsub$Year +dfsub$Month/12

#Keep only data from 2010 onward, and summarize by month/year and station
dfsub_my <- dfsub %>%
  filter(Date>=as.Date("2010-01-01",format="%Y-%m-%d")) %>%
  group_by(Year_Month,cve_station) %>%
  summarize(PM10 = mean(value))

#plot
ggplot(dfsub_my,aes(y=PM10,x=Year_Month,group=cve_station,color=cve_station))+
         geom_line(aes(linetype=cve_station))+
          geom_point(aes(shape=cve_station, color=cve_station))+
         ggtitle("PM10 over time")+
         ylab("Micrograms per cubic meter")+ theme_classic()
```


PM10 over time
Micrograms per cubic meter
150
100
50
2010.0
2012.5
2015.0
2017.5
2020.0
Year_Month
cve_station
MER
PED
TLA
XAL

Example

- Subsetting reveals a more complete story than looking at the aggregated data!
- So how can we do this subsetting?
- There are plenty of ways
- For today we focused on the ‘filter()‘ and ‘select()‘ commands

Practice

- Get the dataset ‘mtcars‘ using the ‘data()‘ function
- Look at it with ‘str()‘ and ‘help()‘
- Limit the dataset to just the variables ‘mpg, cyl‘, and ‘hp‘
- Get the mean ‘hp‘ for cars at or above the median value of ‘cyl‘
- Get the mean ‘hp‘ for cars below the median value of ‘cyl‘
- Do the same for ‘mpg‘ instead of ‘hp‘
- Calculate the difference between above-median and below-median ‘mpg‘ and ‘hp‘
- How do you interpret these differences?

Practice answers

```
data(mtcars)
help(mtcars)
str(mtcars)
mtcars <- mtcars %>% select(mpg,cyl,hp)

mean(filter(mtcars,cyl >= median(cyl))$hp)
mean(filter(mtcars,cyl < median(cyl))$hp)

mean(filter(mtcars,cyl>=median(cyl))$hp)-
mean(filter(mtcars,cyl<median(cyl))$hp)

mean(filter(mtcars,cyl >= median(cyl))$mpg)
mean(filter(mtcars,cyl < median(cyl))$mpg)

mean(filter(mtcars,cyl>=median(cyl))$mpg)-
mean(filter(mtcars,cyl<median(cyl))$mpg)
```