Data validation and exploration

Abhijit Dasgupta

Fall, 2019

Plan today

- Dynamic exploration of data
- Data validation
- Missing data evaluation

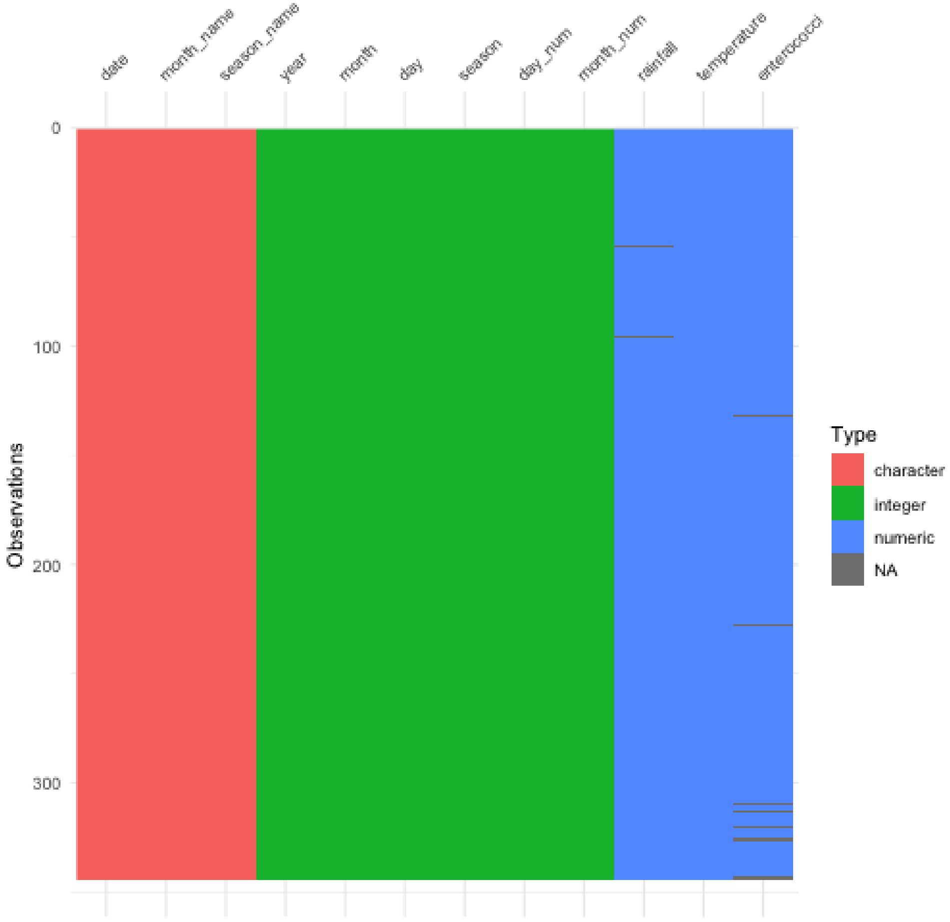
Why go back to this?

This is important!!

- Most of the time in an analysis is spent understanding and cleaning data
- Recognize that unless you've ended up with good-quality data, the rest of the analyses are moot
- This is tedious, careful, non-sexy work
    - Hard to tell your boss you're still fixing the data
    - No real results yet
    - But essential to understanding the appropriate analyses and the tweaks you may need.

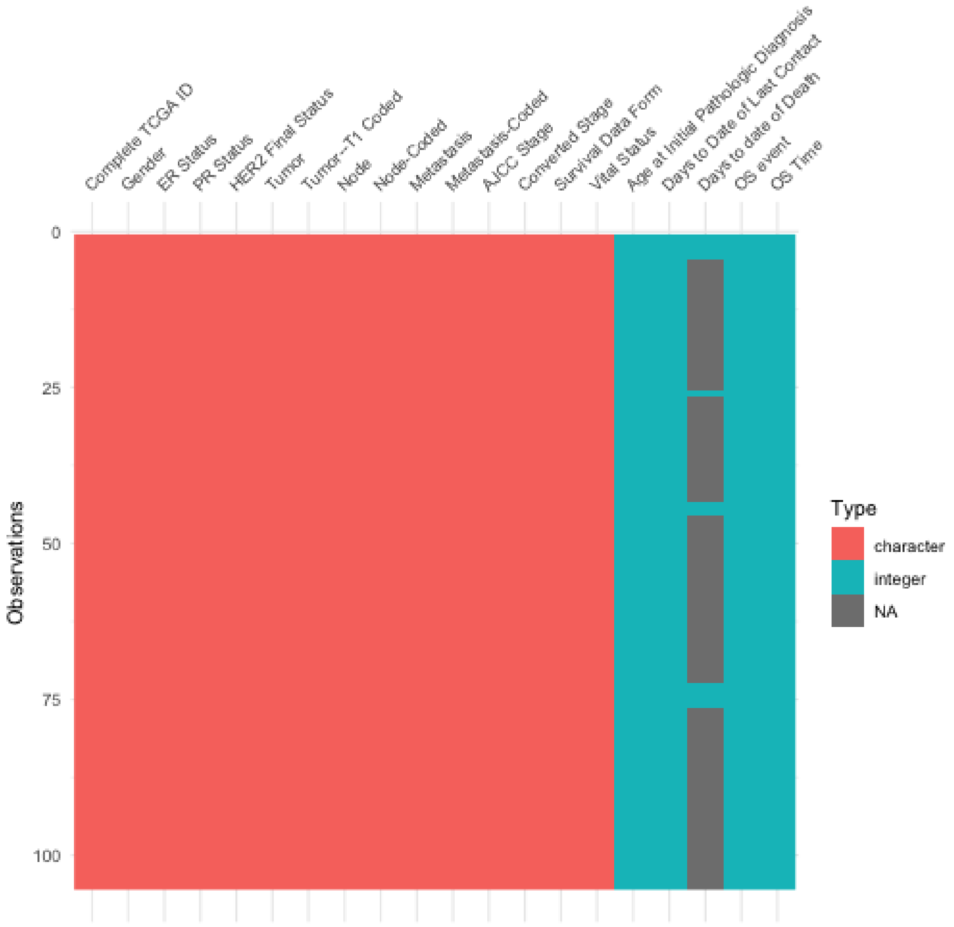
What does a dataset look like?

```r
library(tidyverse)
library(visdat)
beaches <- rio::import('data/sydneybeaches3.csv')
vis_dat(beaches)
```



What does a dataset look like?

```
brca <- rio::import('data/clinical_data_breast_cancer
vis_dat(brca)
```



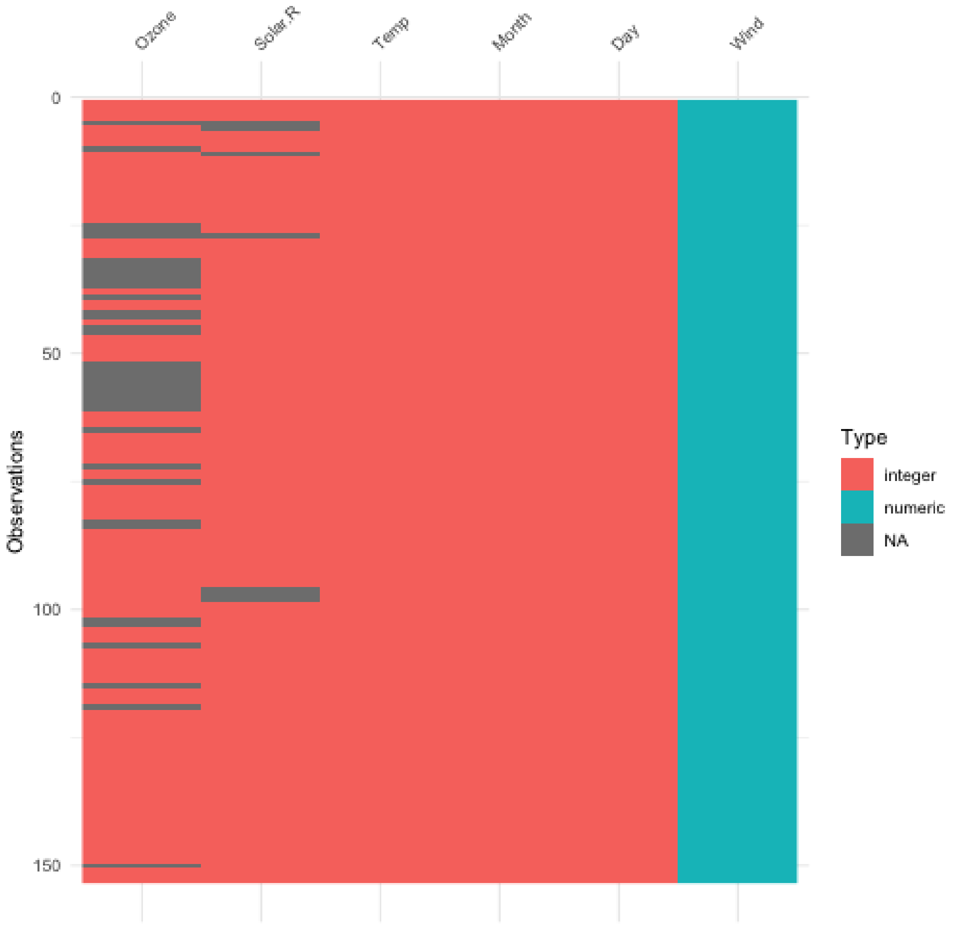
What does a dataset look like?

```
vis_dat(airquality)
```

These plots give a nice insight into

1. data types

2. Missing data patterns (more on this later)



Let's get a bit more quantitative

`summary` and `str`/`glimpse` are a first pass

```
summary(airquality)
```

```
#>      Ozone           Solar.R           Wind
#>  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700
#>  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400
#>  Median : 31.50   Median :205.0   Median : 9.700
#>  Mean   : 42.13   Mean   :185.9   Mean   : 9.958
#>  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500
#>  Max.   :168.00   Max.   :334.0   Max.   :20.700
#>  NA's   :37       NA's   :7
#>      Month            Day
#>  Min.   :5.000   Min.   : 1.0
#>  1st Qu.:6.000   1st Qu.: 8.0
#>  Median :7.000   Median :16.0
#>  Mean   :6.993   Mean   :15.8
#>  3rd Qu.:8.000   3rd Qu.:23.0
#>  Max.   :9.000   Max.   :31.0
#>
```

```
glimpse(airquality)
```

```
#> Observations: 153
#> Variables: 6
#> $ Ozone   <int> 41, 36, 12, 18, NA, 28, 23, 19, 8
#> $ Solar.R <int> 190, 118, 149, 313, NA, NA, 299,
#> $ Wind    <dbl> 7.4, 8.0, 12.6, 11.5, 14.3, 14.9,
#> $ Temp    <int> 67, 72, 74, 62, 56, 66, 65, 59, 6
#> $ Month   <int> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
#> $ Day     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
```

Validating data values

- We can certainly be reactive by just describing the data and looking for anomalies.
- For larger data or multiple data files it makes sense to be proactive and catch errors that you want to avoid, before exploring for new errors.
- The `assertthat` package provides nice tools to do this

> **Note to self:** I don't do this enough. This is a good defensive programming technique that can catch crucial problems that aren't always automatically flagged as errors

Being assertive

```
library(assertthat)
```

```
#>
#>  Attaching package: 'assertthat'
```

```
#>  The following object is masked from 'package:tibble':
#>
#>      has_name
```

```
assert_that(all(between(airquality$Day, 1, 31) ))
```

```
#>  [1] TRUE
```

```
assert_that(is.factor(mpg$manufacturer))
```

```
#>  Error: mpg$manufacturer is not a factor
```

```
assert_that(all(beaches$season_name %in% c('Summer','Winter','Spring', 'Fall')))
```

```
#>  Error: Elements 11, 12, 13, 14, 15, ... of beaches$season_name %in% c("Summer", "Winter", "Spring", "Fall") ar
```

Being assertive

- `assert_that` generates an error, which will stop things
- `see_if` does the same validation, but just generates a `TRUE/FALSE`, which you can capture

```
see_if(is.factor(mpg$manufacturer))
```

```
#>  [1] FALSE
#>  attr(,"msg")
#>  [1] "mpg$manufacturer is not a factor"
```

- `validate_that` generates `TRUE` if the assertion is true, otherwise generates a string explaining the error

```
validate_that(is.factor(mpg$manufacturer))
```

```
#>  [1] "mpg$manufacturer is not a factor"
```

```
validate_that(is.character(mpg$manufacturer))
```

Being assertive

You can even write your own validation functions and custom messages

```
is_odd <- function(x){
    assert_that(is.numeric(x), length(x)==1)
    x %% 2 == 1
}
assert_that(is_odd(2))
```

```
#>  Error: is_odd(x = 2) is not TRUE
```

```
on_failure(is_odd) <- function(call, env) {
  paste0(deparse(call$x), " is even") # This is a R trick
}
assert_that(is_odd(2))
```

```
#>  Error: 2 is even
```

```
is_odd(1:2)
```

```
#>  Error: length(x) not equal to 1
```

Missing data

Missing data

R denotes missing data as `NA`, and supplies several functions to deal with missing data.

The most fundamental is `is.na`, which gives a TRUE/FALSE answer

```
is.na(NA)
```

```
#>  [1] TRUE
```

```
is.na(25)
```

```
#>  [1] FALSE
```

Missing data

When we get a new dataset, it's useful to get a sense of the missingness

```
mpg %>% summarize_all(function(x) sum(is.na(x)))
```

```
#>  # A tibble: 1 x 11
#>    manufacturer model displ  year   cyl trans   drv   cty   hwy    fl class
#>           <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
#>  1            0     0     0     0     0     0     0     0     0     0     0
```

Missing data

The `naniar` package allows a tidyverse-compatible way to deal with missing data

```
library(naniar)
weather <- rio::import('data/weather.csv')
all_complete(mpg)
```

```
#>  [1] TRUE
```

```
all_complete(weather)
```

```
#>  [1] FALSE
```

```
weather %>% summarize_all(pct_complete)
```

```
#>     id year month element       d1       d2       d3       d4       d5
#>   1 100  100   100     100 9.090909 18.18182 18.18182 9.090909 36.36364
#>           d6       d7       d8 d9      d10      d11 d12      d13      d14
#>   1 9.090909 9.090909 9.090909  0 9.090909 9.090909   0 9.090909 18.18182
#>          d15      d16      d17 d18 d19 d20 d21 d22      d23 d24      d25
#>   1 9.090909 9.090909 9.090909   0   0   0   0   0 18.18182   0 9.090909
#>          d26      d27      d28      d29      d30      d31
#>   1 9.090909 27.27273 9.090909 18.18182 9.090909 9.090909
```

Missing data

```
gg_miss_case(weather, show_pct = T)
```

Missing data

```
gg_miss_var(weather, show_pct=T)
```

Are there patterns to the missing data

- Most analyses assume that data is either
  - Missing completely at random (MCAR)
  - Missing at random (MAR)
- MCAR means
  - The missing data is just a random subset of the data
- MAR means
  - Whether data is missing is related to values of some other variable(s)
  - If we control for those variable(s), the missing data would form a random subset of each of those data subsets defined by unique values of these variables

Are there patterns to the missing data

MAR or MCAR allows us to ignore the missing data, since it doesn't bias our analyses

If data are not MCAR or MAR, we really need to understand the issing data mechanism and how that might affect our results.

Co-patterns of missingness

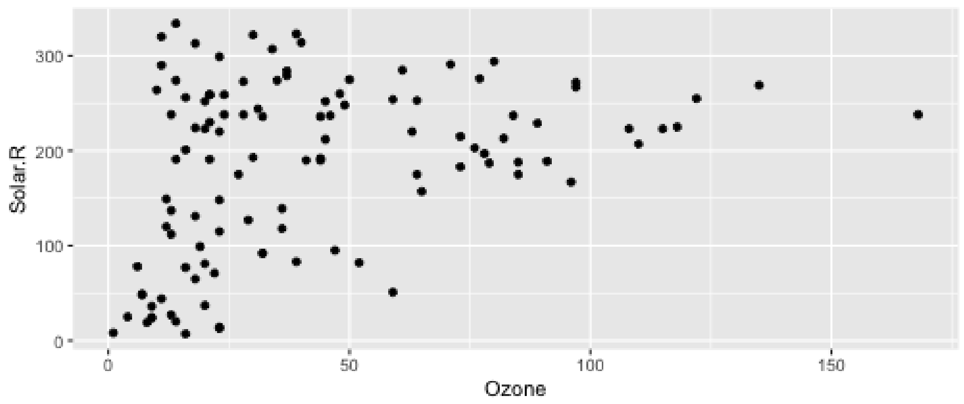
```
gg_miss_upset(airquality)
```

```
gg_miss_upset(riskfactors)
```

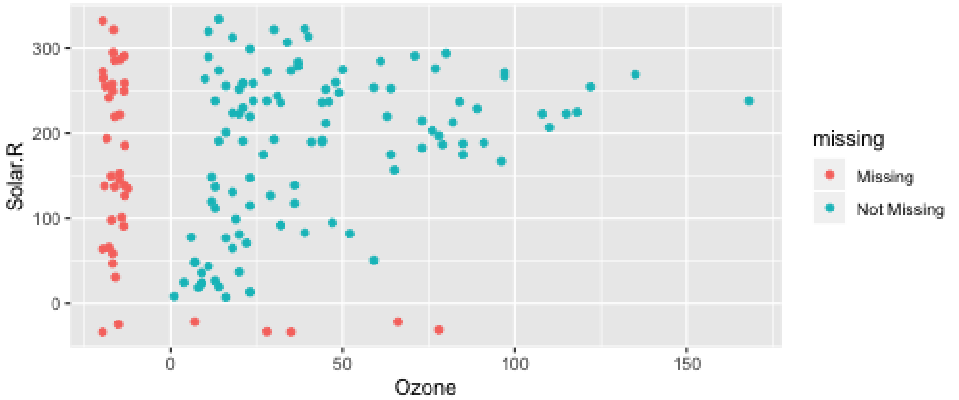
Co-patterns of missingness

```
ggplot(airquality,
       aes(x = Ozone,
           y = Solar.R)) +
geom_point()
```

```
#>  Warning: Removed 42 rows containing missing value
```




```
ggplot(airquality,
       aes(x = Ozone,
           y = Solar.R)) +
 geom_miss_point()
```



Co-patterns of missingness

```
gg_miss_fct(x = riskfactors, fct = marital)
```

Replacing missing data

`tidyr` has a function `replace_na` which will replace all missing values with some particular value.

In the weather dataset, values are missing generally because there wasn't recorded rainfall on a day. So these values should really be 0

```
weather1 <- weather %>% mutate(d1 = replace_na(d1, 0))
pct_miss(weather1$d1)
```

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

Question: How would you replace all the missing values with 0?

```
weather %>% mutate_all(function(x) replace_na(x, 0))
```

How would you replace the missing values with the mean of the variable?

```
weather %>% mutate_if(is.numeric, function(x) replace_na(x, mean(x, na.rm=T)))
```