ETC1010: Data Modelling and Computing

Lecture 3: Wrangling your data

Di Cook (dicook@monash.edu, @visnut)

Week 3

Overview

- Data structures, variable types
- Wrangling verbs: filter, arrange, select, mutate, summarise
- Making joins
- Working with dates

Data structures

- Data frames
- Matrices/vectors
- Lists
- Tibbles

data.frame/tibble vs matrices/vectors

Mostly what we have seen so far are data.frame or tibble

- Rectangular format
- Each column might be a different type of variable
- Each column is same length

Matrices/vectors

- Original data format
- All columns have numerical values

Lists

- Not necessarily rectangular
- Anything can be packed into a list
- We will see more of these when we fit models, because model summaries are typically shared as a list
- Trickier to work with

How do you know what you have?

```
genes <- read_csv("data/genes.csv")
genes
# A tibble: 3 x 12
      id `WI-6.R1` `WI-6.R2` `WI-6.R4` `WM-6.R1` `WM-6.R2` `WI-12.R1`
   <chr>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dbl>
1 Gene 1  2.182424 2.2042219  4.195636 2.6273345  5.063641   4.540002
2 Gene 2  1.464224 0.5854472  1.859238 0.5152242  2.882808   1.364037
3 Gene 3  2.031792 0.8701078  3.281983 0.5330452  4.627315   2.182192
# ... with 5 more variables: `WI-12.R2` <dbl>, `WI-12.R4` <dbl>,
#   `WM-12.R1` <dbl>, `WM-12.R2` <dbl>, `WM-12.R4` <dbl>
is_tibble(genes)
[1] TRUE
is.data.frame(genes)
[1] TRUE
is.matrix(genes)
[1] FALSE
is.list(genes)
[1] TRUE
```

```
melbtemp <- read.fwf("data/ASN00086282.dly",
   c(11, 4, 2, 4, rep(c(5, 1, 1, 1), 31)), fill=T)
head(melbtemp[,c(1,2,3,4,seq(5,100,4))])
           V1   V2 V3   V4  V5  V9 V13 V17 V21 V25 V29 V33 V37 V41 V45 V49
1 ASN00086282 1970  7 TMAX 141 124 113 123 148 149 139 153 123 108 119 112
2 ASN00086282 1970  7 TMIN  80  63  36  57  69  47  84  78  49  42  48  56
3 ASN00086282 1970  7 PRCP   3  30   0   0  36   3   0   0  10  23   3   0
4 ASN00086282 1970  8 TMAX 145 128 150 122 109 112 116 142 166 127 117 127
5 ASN00086282 1970  8 TMIN  50  61  75  67  41  51  48  -7  56  62  47  33
6 ASN00086282 1970  8 PRCP   0  66   0  53  13   3   8   0   0   0   3   5
  V53 V57 V61 V65 V69 V73 V77 V81 V85 V89 V93 V97
1 126 112 115 133 134 126 104 143 141 134 117 142
2  51  36  44  39  40  58  15  33  51  74  39  66
3   5   0   0   0   0   0   8   0  18   0   0   0
4 159 143 114  65 113 125 129 147 161 168 178 161
5  67  84  11  41  18  50  22  28  74  94  73  88
6   0   0  64   3  99  36   8   0   0   0   8  36
is_tibble(melbtemp)
[1] FALSE
is.data.frame(melbtemp)
[1] TRUE
is.matrix(melbtemp)
[1] FALSE
is.list(melbtemp)
[1] TRUE
```

Variable types

- `int` integer.
- `dbl` doubles, or real numbers
- `num` numeric, includes integer or double
- `chr` characters, or strings
- `lgl` or `logi` logical, true or false
- `fctr` or `Factor` fancy character, contains additional information about levels and labels
- `date` and `dttm` time and date structures

Converting formats

```
melbtemp <- melbtemp[,c(1,2,3,4,seq(5,128,4))]
colnames(melbtemp) <- c("id", "year", "month", "var", paste0("V",1:31))
melbtemp <- melbtemp %>%
  gather(day, value, V1:V31) %>%
  mutate(day = sub("V", "", day)) %>%
  mutate(value=ifelse(value==-9999, NA, value)) %>%
  filter(var %in% c("PRCP", "TMAX", "TMIN")) %>%
  spread(var, value) %>%
  mutate(PRCP=PRCP/10, TMAX=TMAX/10, TMIN=TMIN/10) %>%
  as_tibble()
melbtemp
# A tibble: 16,399 x 7
           id  year month   day  PRCP  TMAX  TMIN
        <fctr> <int> <int> <chr> <dbl> <dbl> <dbl>
 1 ASN00086282  1970     7     1   0.3  14.1   8.0
 2 ASN00086282  1970     7    10   2.3  10.8   4.2
 3 ASN00086282  1970     7    11   0.3  11.9   4.8
 4 ASN00086282  1970     7    12   0.0  11.2   5.6
 5 ASN00086282  1970     7    13   0.5  12.6   5.1
 6 ASN00086282  1970     7    14   0.0  11.2   3.6
 7 ASN00086282  1970     7    15   0.0  11.5   4.4
 8 ASN00086282  1970     7    16   0.0  13.3   3.9
 9 ASN00086282  1970     7    17   0.0  13.4   4.0
10 ASN00086282  1970     7    18   0.0  12.6   5.8
# ... with 16,389 more rows
```

Filter

Pick observations by their values. For example,

```
filter(var %in% c("PRCP", "TMAX", "TMIN"))
```

took the column named `var` and keeps rows that have one of three values `PRCP`, `TMAX`, `TMIN`. All other rows are removed.

Logical operators

- Comparisons
  - `<`, `<=`: less than, less than or equal
  - `>`, `>=`: greater than, greater than or equal
  - `==`, `!=`: equal to, not equal to
  - Cautions: computers have a hard time with equals, `near()` is a function that can be useful for real numbers, will check within a small neighbourhood of a value
- Logicals
  - `&` and, both checks have to be true
  - `|` or, either check is true
  - `%in%` checks the elements of a collection, multiple or's
- Missings: `is.na()`

Arrange

Orders a data frame or tibble by the values in one column

```
library(lubridate)
melbtemp %>%
  mutate(date=ymd(paste(year, month, day, sep="-"))) %>%
  arrange(date)
# A tibble: 16,399 x 8
            id  year month   day  PRCP  TMAX  TMIN       date
        <fctr> <int> <int> <chr> <dbl> <dbl> <dbl>     <date>
 1 ASN00086282  1970     7     1   0.3  14.1   8.0 1970-07-01
 2 ASN00086282  1970     7     2   3.0  12.4   6.3 1970-07-02
 3 ASN00086282  1970     7     3   0.0  11.3   3.6 1970-07-03
 4 ASN00086282  1970     7     4   0.0  12.3   5.7 1970-07-04
 5 ASN00086282  1970     7     5   3.6  14.8   6.9 1970-07-05
 6 ASN00086282  1970     7     6   0.3  14.9   4.7 1970-07-06
 7 ASN00086282  1970     7     7   0.0  13.9   8.4 1970-07-07
 8 ASN00086282  1970     7     8   0.0  15.3   7.8 1970-07-08
 9 ASN00086282  1970     7     9   1.0  12.3   4.9 1970-07-09
10 ASN00086282  1970     7    10   2.3  10.8   4.2 1970-07-10
# ... with 16,389 more rows
```

Select

Choose some of the **variables**.

```
airport <- read_csv("data/airports.csv")
airport
# A tibble: 13,094 x 29
   AIRPORT_SEQ_ID AIRPORT_ID AIRPORT     DISPLAY_AIRPORT_NAME
            <int>      <int>   <chr>                    <chr>
 1        1000101      10001     01A     Afognak Lake Airport
 2        1000301      10003     03A  Bear Creek Mining Strip
 3        1000401      10004     04A          Lik Mining Camp
 4        1000501      10005     05A     Little Squaw Airport
 5        1000601      10006     06A             Kizhuyak Bay
 6        1000701      10007     07A    Klawock Seaplane Base
 7        1000801      10008     08A Elizabeth Island Airport
 8        1000901      10009     09A          Augustin Island
 9        1001001      10010     1B1          Columbia County
10        1001002      10010     1B1          Columbia County
# ... with 13,084 more rows, and 25 more variables:
#   DISPLAY_AIRPORT_CITY_NAME_FULL <chr>, AIRPORT_WAC <int>,
#   AIRPORT_COUNTRY_NAME <chr>, AIRPORT_COUNTRY_CODE_ISO <chr>,
#   AIRPORT_STATE_NAME <chr>, AIRPORT_STATE_CODE <chr>,
#   AIRPORT_STATE_FIPS <chr>, CITY_MARKET_ID <int>,
#   DISPLAY_CITY_MARKET_NAME_FULL <chr>, CITY_MARKET_WAC <int>,
#   LAT_DEGREES <int>, LAT_HEMISPHERE <chr>, LAT_MINUTES <int>,
#   LAT_SECONDS <int>, LATITUDE <dbl>, LON_DEGREES <int>,
#   LON_HEMISPHERE <chr>, LON_MINUTES <int>, LON_SECONDS <int>,
```

```
airport %>%
  select(AIRPORT, LATITUDE, LONGITUDE, AIRPORT_IS_LATEST, DISPLAY_AIRPORT_NAM
# A tibble: 13,094 x 5
   AIRPORT LATITUDE  LONGITUDE AIRPORT_IS_LATEST     DISPLAY_AIRPORT_NAME
     <chr>    <dbl>      <dbl>              <int>                    <chr>
 1     01A 58.10944 -152.90667                  1     Afognak Lake Airport
 2     03A 65.54806 -161.07167                  1  Bear Creek Mining Strip
 3     04A 68.08333 -163.16667                  1          Lik Mining Camp
 4     05A 67.57000 -148.18389                  1     Little Squaw Airport
 5     06A 57.74528 -152.88278                  1             Kizhuyak Bay
 6     07A 55.55472 -133.10167                  1    Klawock Seaplane Base
 7     08A 59.15694 -151.82917                  1 Elizabeth Island Airport
 8     09A 59.36278 -153.43056                  1          Augustin Island
 9     1B1 42.28889  -73.71028                  0          Columbia County
10     1B1 42.29139  -73.71028                  1          Columbia County
# ... with 13,084 more rows
```

Mutate

Create new, or transform existing, variables, e.g. for the Melbourne temperature data, we put the temperature and precipitation values into Celsius and mm, byt dividing by 10.

```
mutate(PRCP=PRCP/10, TMAX=TMAX/10, TMIN=TMIN/10)
```

Arithmetic

- Typical calculations: `+`, `-`, `*`, `/`; `^` (power)
- Modular:
  - `%/%` integer division
  - `%%` remainder
- Transformations: `log()`, `log10()`, `sqrt()`
- Temporal: `lead()`, `lag()` create variables which are temporal lags of existing
- Stats: `rank()`, `cumsum()`

Summarise

Calculate a quantity on a column, producing a single number, e.g. for the audio data:

```
summarise(m = mean(value), s = sd(value),
          mx = max(value), mn = min(value))
```

- Stats: `mean()`, `median()`, `sd()`, `min()`, `max()`, `sum()`
- Rounding: `floor()`, `ceiling()`, `trunc()`, `round()`, `signif()`

Grouping and ungrouping

Summarise is most commonly called on subgroups of data. We might want to calculate means and standard deviation across genders, or countries or schools. For the audio data, we calculated the summary statistics by the word spoken:

```
group_by(word) %>%
summarise(m = mean(value), s = sd(value),
          mx = max(value), mn = min(value))
```

If we want to more operations on the variable `word` after these calculations, you would need to do an `ungroup()` step.

Counts and tallies

Dealing with categorical data, means wanting to count or tally. The function `count()` allows calculate the number of objects in levels of a variable, and the function `tally()` calculates how many objects there are. We used `count()` for the tweets example:

```
tweets %>%
  count(source, hour = hour(with_tz(created, "EST")))
```

to calculate the number of tweets from the different devices, android or iphone, each hour.

Putting it together for the french fries

10 week sensory experiment, 12 individuals assessed taste of french fries on several scales (how potato-y, buttery, grassy, rancid, paint-y do they taste?), fried in one of 3 different oils, replicated twice. First few rows:

```
# A tibble: 696 x 9
     time treatment subject   rep potato buttery grassy rancid painty
 * <fctr>    <fctr>  <fctr> <dbl>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
 1      1         1       3     1    2.9     0.0    0.0    0.0    5.5
 2      1         1       3     2   14.0     0.0    0.0    1.1    0.0
 3      1         1      10     1   11.0     6.4    0.0    0.0    0.0
 4      1         1      10     2    9.9     5.9    2.9    2.2    0.0
 5      1         1      15     1    1.2     0.1    0.0    1.1    5.1
 6      1         1      15     2    8.8     3.0    3.6    1.5    2.3
 7      1         1      16     1    9.0     2.6    0.4    0.1    0.2
 8      1         1      16     2    8.2     4.4    0.3    1.4    4.0
 9      1         1      19     1    7.0     3.2    0.0    4.9    3.2
10      1         1      19     2   13.0     0.0    3.1    4.3   10.3
# ... with 686 more rows
```

What would we like to know?

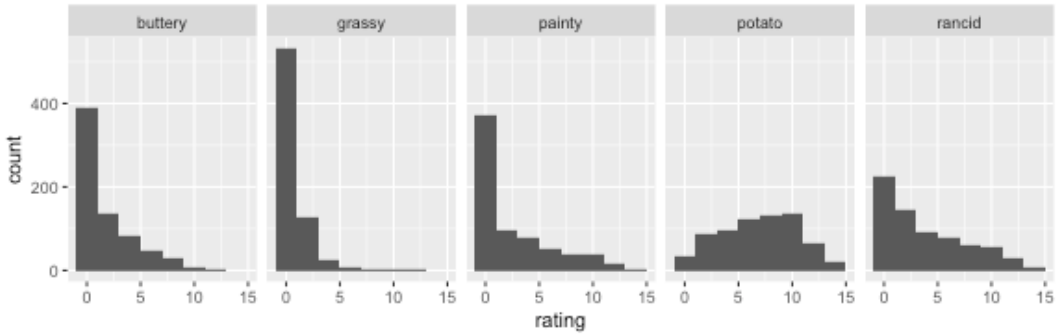
- Is the design complete?
- Are replicates like each other?
- How do the ratings on the different criteria differ?
- Are raters giving different scores on average?
- Do ratings change over the weeks?

Each of these questions involves different summaries of the data.

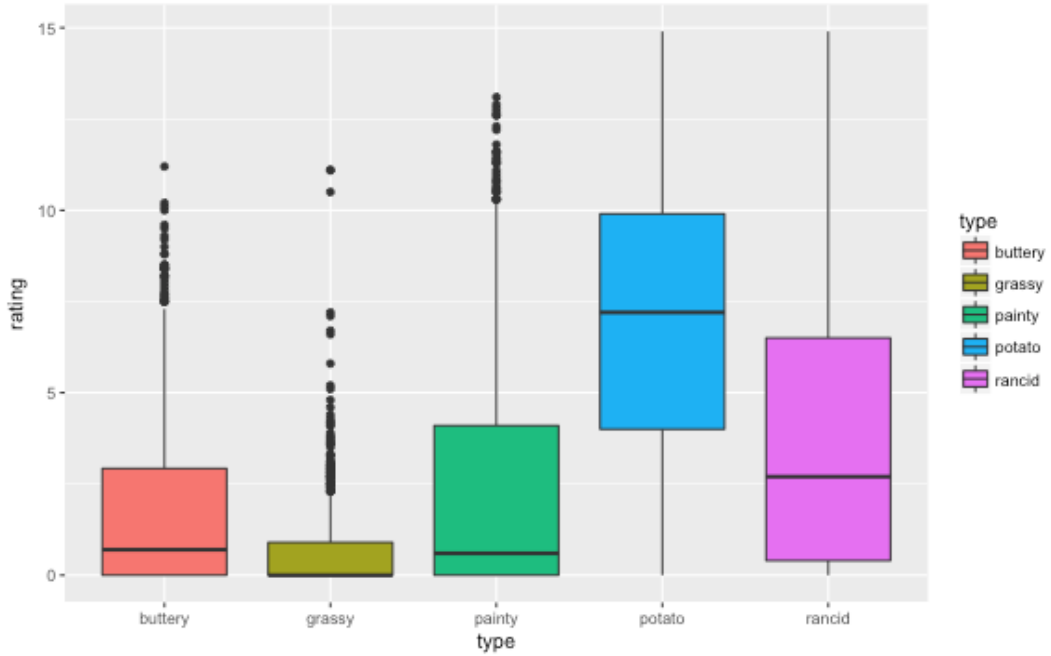
Answer some Questions

- Easiest question is whether the ratings are similar on the different scales, potato'y, buttery, grassy, rancid and painty.
- We need to gather the data into long form, and make plots facetted by the scale.

```
ff.m <- french_fries %>%
  gather(type, rating, -subject, -time, -treatment, -rep)
ggplot(data=ff.m, aes(x=rating)) + geom_histogram(binwidth=2) +
  facet_wrap(~type, ncol=5)
```



Look at it a different way



Comparison of the distributions of criteria

Ratings on the different criteria are quite different.

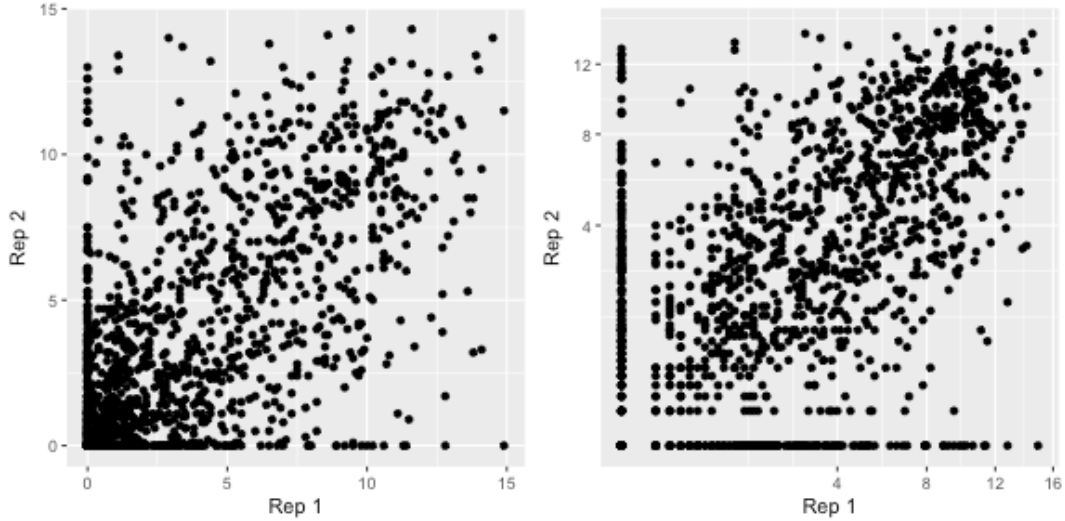
- Potato'y scores relatively highly.
- Grassy are mostly o's
- Buttery and painty are skewed right, mostly low values a few high ratings
- Rancid is quite varied, we would hope that this relates to time, that the chips get more rancid as the weeks go by.

Do the replicates look like each other?

- We will start to tackle this by plotting the replicates against each other using a scatterplot.
- If raters give the same rating to the replicates, we would expect something close to this, then all values would lie on the X=Y line
- We need to
  - gather the data into long form, and
  - then get the replicates spread into separate columns.

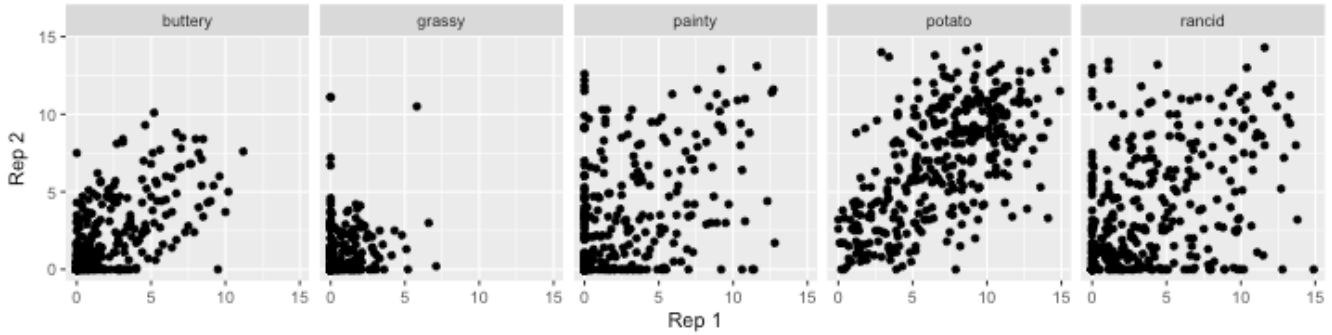
```
ff.s <- ff.m %>% spread(rep, rating)
head(ff.s)
# A tibble: 6 x 6
    time treatment subject    type   `1`   `2`
  <fctr>    <fctr>  <fctr>   <chr> <dbl> <dbl>
1      1         1       3 buttery   0.0   0.0
2      1         1       3  grassy   0.0   0.0
3      1         1       3  painty   5.5   0.0
4      1         1       3  potato   2.9  14.0
5      1         1       3  rancid   0.0   1.1
6      1         1      10 buttery   6.4   5.9
```

Check Replicates



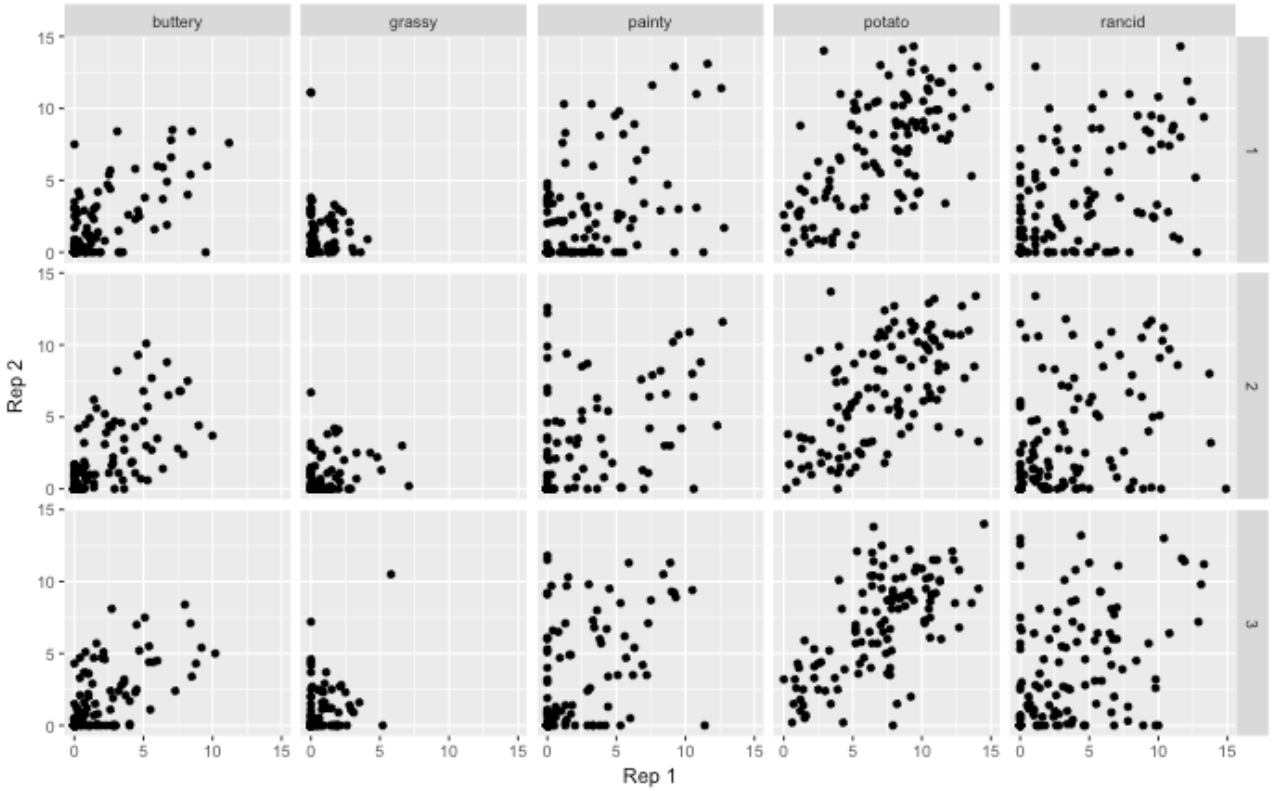
They have some positive linear association, but there is a lot more variation than expected. One rep might have scored 15 and the other 0, that's the same batches, same oil, same criteria, same week, same rater!

Separately by criteria ...



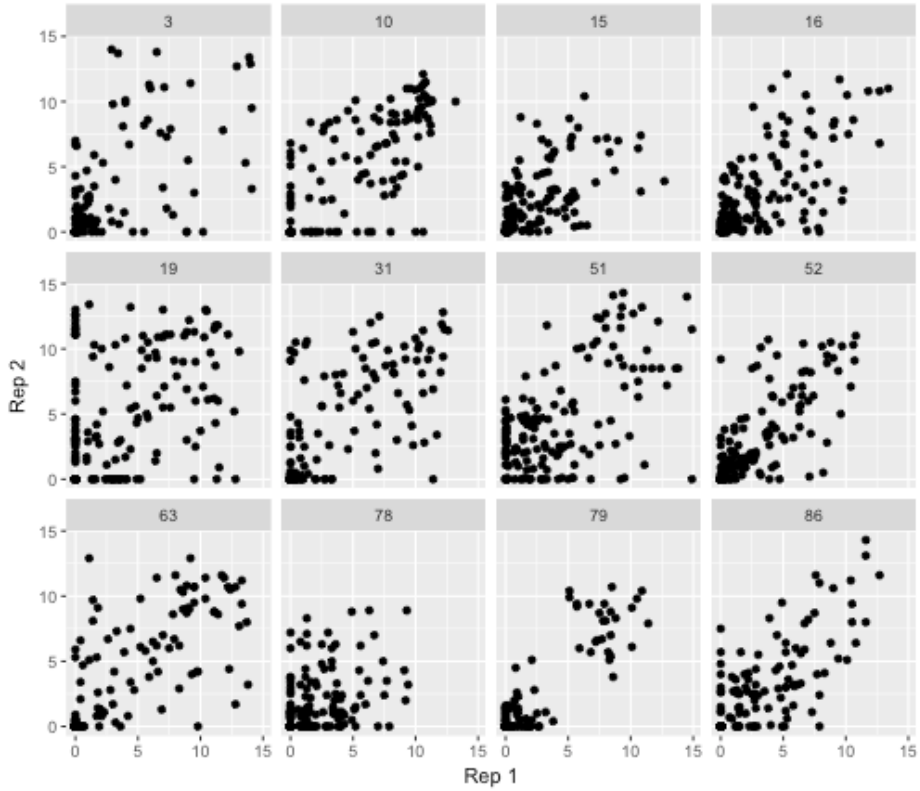
buttery and potato'y look a bit better. The rest are still terrible.

by oil ...



Not much improvement here.

By subject ...



Some subjects may be more experienced than others?

Because the replicates do not look like each other, the quality of the data might be questioned at this point.

Nevertheless, lets push on with some of the other questions.

Completeness of experimental design

If the data is complete it should be 12 x 10 x 3 x 2, that is, 6 records for each person. (Assuming that each person rated on all scales.)

To check this we want to tabulate the number of records for each subject, time and treatment. That is,

- select appropriate columns,
- tabulate,
- count and
- spread it out to give a nice table.

```
french_fries %>%
  select(subject, time, treatment) %>%
  count(subject, time) %>%
  spread(time, n)
# A tibble: 12 x 11
   subject   `1`   `2`   `3`   `4`   `5`   `6`   `7`   `8`   `9`  `10`
 *  <fctr> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
 1       3     6     6     6     6     6     6     6     6     6    NA
 2      10     6     6     6     6     6     6     6     6     6     6
 3      15     6     6     6     6     6     6     6     6     6     6
 4      16     6     6     6     6     6     6     6     6     6     6
 5      19     6     6     6     6     6     6     6     6     6     6
 6      31     6     6     6     6     6     6     6     6    NA     6
 7      51     6     6     6     6     6     6     6     6     6     6
 8      52     6     6     6     6     6     6     6     6     6     6
 9      63     6     6     6     6     6     6     6     6     6     6
10      78     6     6     6     6     6     6     6     6     6     6
11      79     6     6     6     6     6     6     6     6     6    NA
12      86     6     6     6     6     6     6     6     6    NA     6
```

Its pretty good, but subjects, 3, 31, 79, 86 all missed a rating session.

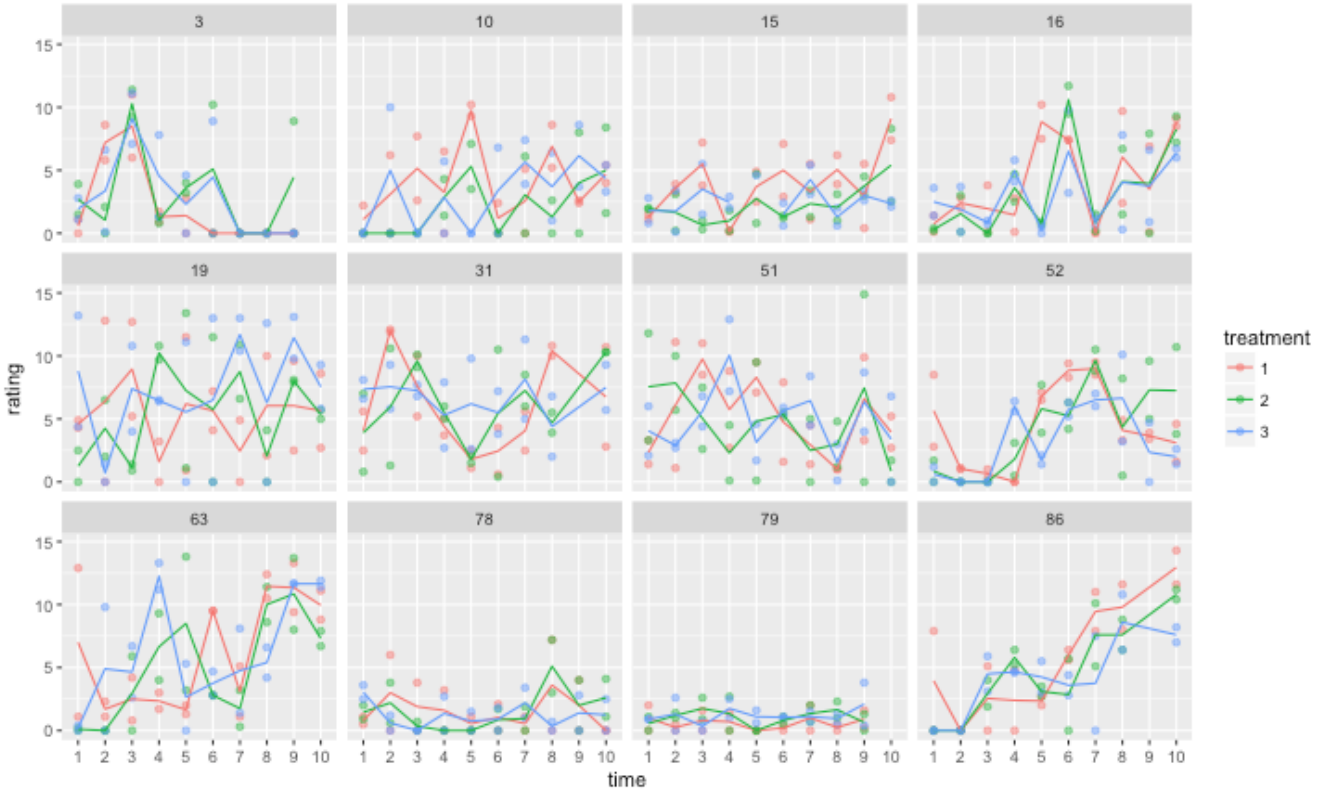
By criteria

```
# A tibble: 12 x 11
   subject   `1`   `2`   `3`   `4`   `5`   `6`   `7`   `8`   `9`  `10`
 *  <fctr> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
 1       3    30    30    30    30    30    30    30    30    30    NA
 2      10    30    30    30    30    30    30    30    30    30    30
 3      15    30    30    30    30    30    30    30    30    30    30
 4      16    30    30    30    30    30    30    30    30    30    30
 5      19    30    30    30    30    30    30    30    30    30    30
 6      31    30    30    30    30    30    30    30    30    NA    30
 7      51    30    30    30    30    30    30    30    30    30    30
 8      52    30    30    30    30    30    30    30    30    30    30
 9      63    30    30    30    30    30    30    30    30    30    30
10      78    30    30    30    30    30    30    30    30    30    30
11      79    30    30    30    30    30    30    30    30    30    NA
12      86    30    30    30    30    30    30    30    30    NA    30
```

Change in rancid ratings over weeks

- Filter on criteria
- Compute means by subject, week and oil

```
ff.av <- ff.m %>%
  filter(type == "rancid") %>%
  group_by(subject, time, treatment) %>%
  summarise(rating=mean(rating))
```



Oh, its awful data! Only subject 86 is seeing the chips get more rancid over time, with maybe oil 1 being worse. Subject 63, shows some trend. Nothing is rancid for subjects 78, 79. Subject 53 thinks they taste better after many weeks in the same old oil!

Joins

It's rare that a data analysis involves only a single table of data. Typically you have many tables of data, and you must combine them to answer the questions that you're interested in. Collectively, multiple tables of data are called relational data because it is the relations, not just the individual datasets, that are important.

```
load("data/plane_N4YRAA.rda")
plane_N4YRAA %>% glimpse()
Observations: 145
Variables: 8
$ FL_DATE  <date> 2017-05-26, 2017-05-02, 2017-05-05, 2017-05-11, 2017...
$ CARRIER  <chr> "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA",...
$ FL_NUM   <int> 2246, 2276, 2278, 2287, 2288, 2291, 2297, 2297, 2297,...
$ ORIGIN   <chr> "CVG", "DFW", "DFW", "STL", "IND", "CHS", "DFW", "DFW...
$ DEST     <chr> "DFW", "IND", "OKC", "ORD", "DFW", "DFW", "MKE", "MKE...
$ DEP_TIME <chr> "0748", "2020", "0848", "0454", "0601", "0807", "0700...
$ ARR_TIME <chr> "0917", "2323", "0941", "0600", "0719", "0947", "0905...
$ DISTANCE <dbl> 812, 761, 175, 258, 761, 987, 853, 853, 853, 853, 447...
```

```
airport <- read_csv("data/airports.csv")
airport %>% select(AIRPORT, LATITUDE, LONGITUDE, AIRPORT_STATE_NAME) %>%
  glimpse()
Observations: 13,094
Variables: 4
$ AIRPORT            <chr> "01A", "03A", "04A", "05A", "06A", "07A", "...
$ LATITUDE           <dbl> 58.10944, 65.54806, 68.08333, 67.57000, 57....
$ LONGITUDE          <dbl> -152.90667, -161.07167, -163.16667, -148.18...
$ AIRPORT_STATE_NAME <chr> "Alaska", "Alaska", "Alaska", "Alaska", "Al...
```

Joining the two tables

- Purpose is to show flight movement on the map
- Key is the airport three letter code,
  - called ORIGIN or DEST in plane_N4YRAA table
  - called AIRPORT in the airport table
- One table, plane_N4YRAA, has less airports than the other
  - Only want to keep the rows of airport table, for those that appear in the plane_N4YRAA table

```
airport <- airport %>%
  select(AIRPORT, LATITUDE, LONGITUDE, AIRPORT_IS_LATEST, DISPLAY_AIRPORT_NAM
  filter(AIRPORT_IS_LATEST == 1) %>%
  select(-AIRPORT_IS_LATEST)

N4YRAA_latlon <- left_join(plane_N4YRAA, airport,
                           by = c("ORIGIN"="AIRPORT")) %>%
  rename("ORIGIN_LATITUDE"="LATITUDE",
         "ORIGIN_LONGITUDE"="LONGITUDE")
N4YRAA_latlon %>%
  select(ORIGIN, ORIGIN_LATITUDE, ORIGIN_LONGITUDE,
         DISPLAY_AIRPORT_NAME)
# A tibble: 146 x 4
   ORIGIN ORIGIN_LATITUDE ORIGIN_LONGITUDE
    <chr>           <dbl>            <dbl>
 1    CVG        39.04889        -84.66778
 2    DFW        32.89722        -97.03778
 3    DFW        32.89722        -97.03778
 4    STL        38.74861        -90.37000
 5    IND        39.71722        -86.29472
 6    CHS        32.89861        -80.04056
 7    DFW        32.89722        -97.03778
 8    DFW        32.89722        -97.03778
 9    MKE        42.94694        -87.89694
10    MKE        42.94694        -87.89694
# ... with 136 more rows, and 1 more variables: DISPLAY_AIRPORT_NAME <chr>
```

The variables ORIGIN_LATITUDE, ORIGIN_LONGITUDE, DISPLAY_AIRPORT_NAME are added to corresponding row in the plane_N4YRAA table.

- Added the spatial coordinates (lat, lon) for the origin airport
- The same needs to be done for the destination airport
- Then the airports can be drawn over a map

Share and share alike