Data Management and Processing

Zack Jaggers

22 April, 2020
handout and files: http://zjaggers.com/resources/DataMgmtWorkshop/

Intro

This workshop: how to work with the data you've collected, and make it analyzable.

- Keeping the end state in mind, from the beginning
  - Ultimately, you want your data to be in a state that can be submitted to analysis.
  - So, at every step, you'll want to think about
    - what that ideal end state is, and
    - how to help the data you collect resemble or be able to get to that state.

- Pragmatics: the real world
  - The data you collect won't be in that state already. (i.e., why this workshop exists)
  - So, at every step, you'll want to make sure that
    - you have the pieces of information you'll need in your analysis,
    - you can trace it to originals (in case things are separated, altered, or moved), and
    - you can easily connect pieces to each other that will need to be connected.

Foreshadow/summary

- the goal state: one dataset
  - unique rows per observation (data points)
  - unique columns per factor of interest (ways to splice/dice/group the observations)
  - including, while preferably limited to, all observations of interest

- getting there
  - Know your factors of interest and identify their values for each observation, somewhere.
    - usually factors of the stimulus, study condition, and subject
  - Have reference datasets where these factors are treated as observations w/ info.
    - Make these just like another dataset — spreadsheet format.
    - Keep a separate record of how you've labeled and moved your data — can be prose.
  - Combine and restructure your raw data (as necessary) to reach the goal state.

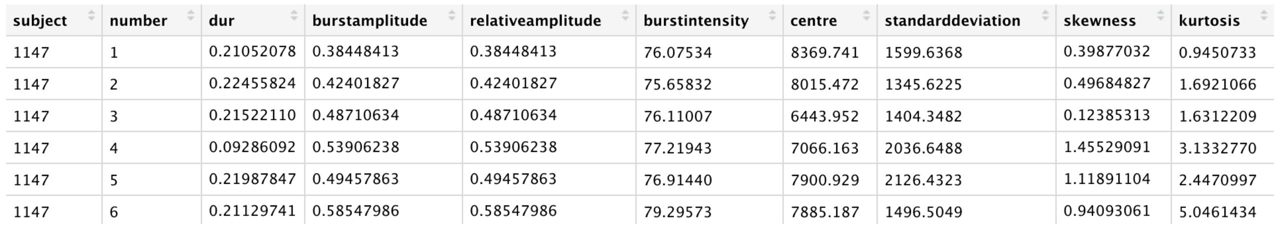
Tools

- best practices for tracking information, considering ease for joining it all later on
- binding/concatenating: joining complementary sets of <u>observations</u> of same info structure
- joining/merging complementary sets of <u>information</u> that observation sets don't fully specify
- recoding, separating: dealing (w/in-script) with labels that weren't ideal
- subsetting/filtering: limiting dataset only to observations of interest
- reshaping: e.g., multiple observations per row in starting point (e.g., experiment output)

1. Compiling your data — tracking all the information to be combined

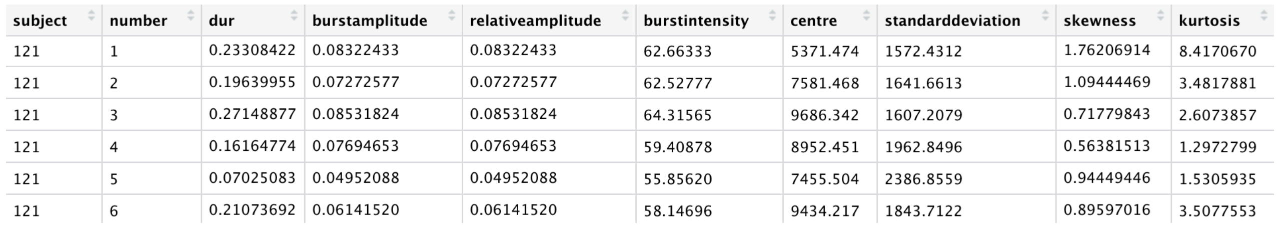
Often, your data output doesn't already have <u>all</u> the information of interest to your analysis neatly lumped into one tidy spreadsheet.

- You probably want to analyze the observations from multiple subjects together, but they're likely in different output spreadsheets.
- And those spreadsheets might not have all the information you want for each observation:
  - Say you have acoustic measurements of subjects' utterances from a Praat script.
  - But, the script output didn't include information like subject's language background, sex.
  - And, if you were using particular audio stimuli to elicit the productions, you might care about aspects of those stimuli, which the script might also not have given you.



subject	number	dur	burstamplitude	relativeamplitude	burstintensity	centre	standarddeviation	skewness	kurtosis
1147	1	0.21052078	0.38448413	0.38448413	76.07534	8369.741	1599.6368	0.39877032	0.9450733
1147	2	0.22455824	0.42401827	0.42401827	75.65832	8015.472	1345.6225	0.49684827	1.6921066
1147	3	0.21522110	0.48710634	0.48710634	76.11007	6443.952	1404.3482	0.12385313	1.6312209
1147	4	0.09286092	0.53906238	0.53906238	77.21943	7066.163	2036.6488	1.45529091	3.1332770
1147	5	0.21987847	0.49457863	0.49457863	76.91440	7900.929	2126.4323	1.11891104	2.4470997
1147	6	0.21129741	0.58547986	0.58547986	79.29573	7885.187	1496.5049	0.94093061	5.0461434

...

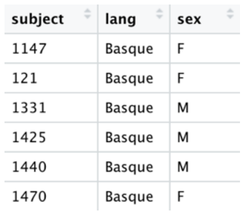


subject	number	dur	burstamplitude	relativeamplitude	burstintensity	centre	standarddeviation	skewness	kurtosis
121	1	0.23308422	0.08322433	0.08322433	62.66333	5371.474	1572.4312	1.76206914	8.4170670
121	2	0.19639955	0.07272577	0.07272577	62.52777	7581.468	1641.6613	1.09444469	3.4817881
121	3	0.27148877	0.08531824	0.08531824	64.31565	9686.342	1607.2079	0.71779843	2.6073857
121	4	0.16164774	0.07694653	0.07694653	59.40878	8952.451	1962.8496	0.56381513	1.2972799
121	5	0.07025083	0.04952088	0.04952088	55.85620	7455.504	2386.8559	0.94449446	1.5305935
121	6	0.21073692	0.06141520	0.06141520	58.14696	9434.217	1843.7122	0.89597016	3.5077553

What do you do? Hopefully, you've thought ahead some about being able to track this information and combine it into a unified dataset in the future:

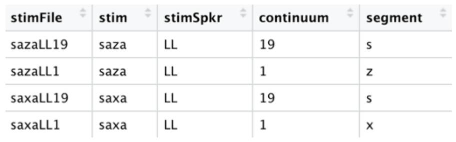
- ID: Ideally each subject has a separate ID (see 'subject' column above), so when you combine their observations together, you can still tell them apart.
  - This way, you can associate each Praat script observation with its subject.
  - There's a way you can get this from the filename, which probably is or at least includes the subject ID. (Most Praat scripts produce this. See ID/filename extraction below.)

- Subject info tracking: Ideally, you gathered the information you care about (e.g., language background, sex) for each subject.
  - And ideally, you have this associated with the same ID that your other observations are associated with.
    - Now (whew!), you know that even if you don't have all that information together right now, it's possible to, say, create new info columns for each observation in the output from your Praat script, and fill them in with this info you have stored.
  - Better/easier yet, if you have this information stored in a structure just like another dataset, you can automate that process.



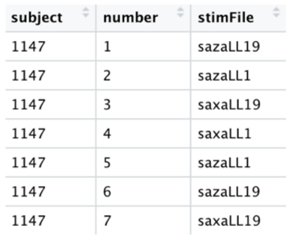
subject	lang	sex
1147	Basque	F
121	Basque	F
1331	Basque	M
1425	Basque	M
1440	Basque	M
1470	Basque	F

- The same would go for stimulus information: If it will be relevant to your analysis, but it wasn't included in your observation output...
  - You should have a way to ID the stimulus (maybe it's word, or audio stimulus filename).
  - And you should be able to track which one corresponds to each observation utterance:
    - Maybe this is by having segmented the word and stored the info of all words segmented as a dataset to refer to.



stimFile	stim	stimSpkr	continuum	segment
sazaLL19	saza	LL	19	s
sazaLL1	saza	LL	1	z
saxaLL19	saxa	LL	19	s
saxaLL1	saxa	LL	1	x

    - Maybe this is by having kept a record of the order of stimulus presentation each subject was exposed to. In this case, you'll need to keep both subject and observation order in mind when combining this information. (This is also totally doable, so long as you've kept records, in the form of datasets: see below.)



subject	number	stimFile
1147	1	sazaLL19
1147	2	sazaLL1
1147	3	saxaLL19
1147	4	saxaLL1
1147	5	sazaLL1
1147	6	sazaLL19
1147	7	saxaLL19

Takeaway:

- Your data might not all come out ready-to-analyze. But your life will be easier the more you make sure you track relevant information, and think about how you'll plug it into analysis.
  - Observation responses/measurements
  - Subject info
  - Stimulus info
  - Condition or exposure presentation info
  - ***Ways of connecting all of these things.***

- Tips
  - The fewer chain links, the better.
    - (e.g., How would you identify which 'segment' the subject was producing for each observation from the Praat script output in the first spreadsheets above?)
  - Format the complementary pieces of info like separate datasets to be combined (see "Combining your data" below).
  - Try to keep factor columns consistent across these datasets (e.g., 'subject', 'stimFile').
  - If there's anything else you need to note, but you're having a hard time formatting it as a dataset, have a separate document where you can write it out as prose: e.g.,
    - work progress
    - instruction guides
    - logs of file inventory, changes, or movements
    - writeup of questions or issues w/ data processing and how they were resolved
    - keys for codes or abbreviations used in datasets

- R
  - For the next steps, I'm going to walk you through some R tools and commands. This isn't meant as a full-on intro to R, but a data management tutorial using R. Most everything should be doable with just basic **R and RStudio**, and the **`tidyverse` package**.

2. Combining your data — concatenating and merging

Why did you want your separate info sets structured like datasets? So you can easily combine them!

- First, what about all those different Praat script outputs from different subjects? You want those together in one dataset if you're going to compare subjects with each other.
  - You can row-bind data sets, vertically stacking them if they have identical column structures. **I've provided a script (CombineFiles.R)** that combines all datasets located in one directory. The main content of it is duplicated here:

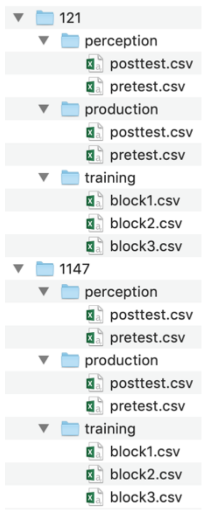
```
#Set working directory to that where script resides
sourcedir <- dirname(rstudioapi::getSourceEditorContext()$path)
setwd(sourcedir)

#identify "...csv" files in working directory
filelist <- list.files(path = getwd(), pattern = "*.csv")
#read and combine, storing filename as first column
d <- data_frame(filename = filelist) %>% #new dataframe with filenames
  mutate(file_contents = map(filename, #ready expansion: one contents cell
per filename
                             ~ read_csv(file.path(getwd(), .))) #read
contents into nested cells
  ) %>%
  unnest(.) #unnest dataframe cells into one combined dataframe

#Save output as csv in the directory
write.csv(d, file = "CombinedOutput.csv", row.names=FALSE)
```

- So now, the observations of subjects 1147, 121, etc. will be combined in one dataset, but you still have 'subject' as a column that uses their subject IDs to keep their observations distinct.

nb: This script actually also includes a new 'filename' column, in case you didn't have something like 'subject' before.

A note on file storage and folder structure:

- This script and others like it work best if your files are all in the same folder/'directory'.
  - Ideally, that's why your filenames also have information that uniquely distinguishes the files from each other, and you've set up your experiment or analysis scripts to store and label files in this way.
  - But, say, you have separate *folders*, rather than *files* for each subject, and multiple files per subject that would otherwise have identical file names (see image example).
    - If you add a `recursive=TRUE` argument to the `list.files( )` command, you can search subdirectories and store filepath information as well: e.g., `121/training/block1.csv`
    - (Below we'll talk about parsing and splitting column info.)
    - You'll want to make sure that your folder structures, labels, and contents within are consistent if taking this approach.
  - Other datasets containing complementary reference information, like subject info or stimulus info, should be stored outside the directory housing the observation files you want to combine.



Filling out your observations with complementary information:

- Now you can join your datasets: Observations can be combined with complementary information you stored about subjects and stimuli.

(R base calls this 'merge'; dplyr and SQL call this 'join')

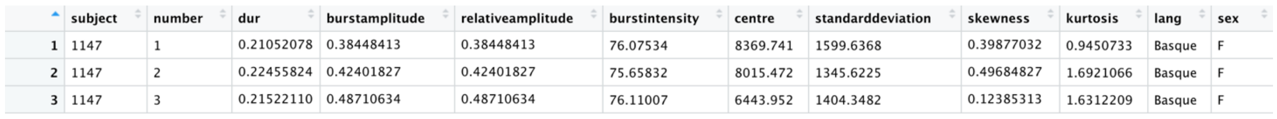
  - Using the `left_join( )` command from dplyr, you can add to a dataset of observations (first argument) columns of complementary information (second argument) so long as there is a matching ID column.

```
> d <- left_join(d.praatObsvs, d.subjInfo)
Joining, by = "subject"
```

    - Notice how, above, the matching 'subject' column between the two datasets was found automatically. You can manually specify it with a third `by=" "` argument.

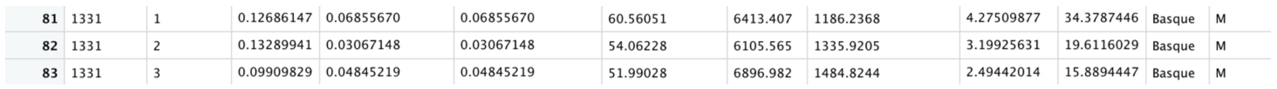
(You can also manually specify if the columns to be matched across datasets differ in their titles; but it's much better if you've been consistent from the start.)

- So now, subject info columns are added, with the values matching their subject IDs.



	subject	number	dur	burstamplitude	relativeamplitude	burstintensity	centre	standarddeviation	skewness	kurtosis	lang	sex
1	1147	1	0.21052078	0.38448413	0.38448413	76.07534	8369.741	1599.6368	0.39877032	0.9450733	Basque	F
2	1147	2	0.22455824	0.42401827	0.42401827	75.65832	8015.472	1345.6225	0.49684827	1.6921066	Basque	F
3	1147	3	0.21522110	0.48710634	0.48710634	76.11007	6443.952	1404.3482	0.12385313	1.6312209	Basque	F

...



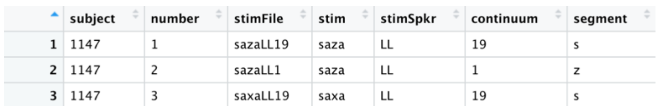
81	1331	1	0.12686147	0.06855670	0.06855670	60.56051	6413.407	1186.2368	4.27509877	34.3787446	Basque	M
82	1331	2	0.13289941	0.03067148	0.03067148	54.06228	6105.565	1335.9205	3.19925631	19.6116029	Basque	M
83	1331	3	0.09909829	0.04845219	0.04845219	51.99028	6896.982	1484.8244	2.49442014	15.8894447	Basque	M

Chain-joins:

- Notice that, for our dataset, we have the utterance number of each observation per speaker.
  - Or, if you don't, so long as you're sure your rows are in the right order:

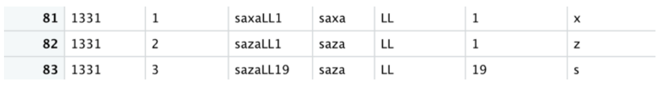
```
d <- d %>% group_by(subject) %>% mutate(number = row_number())
```

- We also have a separate record per subject of the order of stimuli that elicited each utterance ('stimFile').
- We have a record with more stimulus details; but it's separate from the stimulus presentation record, and it's not directly traceable to the full observation dataset: It doesn't share any ID columns with it.
  - Combine your stimulus presentation records per subject, row-binding them like observation datasets.
  - Join them with the additional stimulus information, giving them more detail.



	subject	number	stimFile	stim	stimSpkr	continuum	segment
1	1147	1	sazaLL19	saza	LL	19	s
2	1147	2	sazaLL1	saza	LL	1	z
3	1147	3	saxaLL19	saxa	LL	19	s

...



81	1331	1	saxaLL1	saxa	LL	1	x
82	1331	2	sazaLL1	saza	LL	1	z
83	1331	3	sazaLL19	saza	LL	19	s

- Now you can join this dataset with the dataset of Praat script observations...

Joining by multiple conditions:

- Notice that you'll need both subject ID and utterance number to match each utterance observation's Praat script measurements with its stimulus details: i.e.,
  - It's not the case that each subject saw only one stimulus;
  - and it's not the case that stimulus presentation order ('number' and how it matched with 'stimFile') was the same for every subject.

- You can specify a set of multiple columns as criteria for your `join( )` command:

```
d <- left_join(d, d.subjStimOrder, by=c("subject", "number"))
```

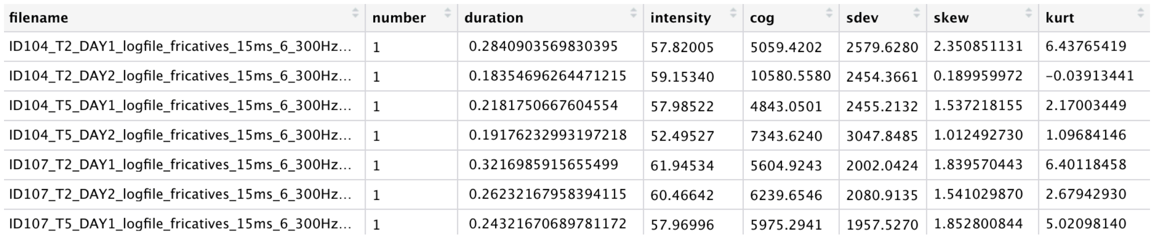
nb: Be mindful about which dataset you're specifying as the left argument. That one is treated as the base that the other is being added to, and it shouldn't be missing observations of interest. (See online guides about dplyr and SQL join types.)

3. Recoding and separating values

In actuality, the data I showed you above had already been pre-cleaned a bit.

Remember: Keep the ideal in mind and aim for it.

Instead of having subject IDs neatly provided by the Praat script, the output looked more like this (a similar dataset). It just spat out the filename shared by both the .wav and .textgrid files we segmented (ID###_T#_DAY#), and appended a bunch of details about the script at the end.



filename	number	duration	intensity	cog	sdev	skew	kurt
ID104_T2_DAY1_logfile_fricatives_15ms_6_300Hz...	1	0.2840903569830395	57.82005	5059.4202	2579.6280	2.350851131	6.43765419
ID104_T2_DAY2_logfile_fricatives_15ms_6_300Hz...	1	0.18354696264471215	59.15340	10580.5580	2454.3661	0.189959972	-0.03913441
ID104_T5_DAY1_logfile_fricatives_15ms_6_300Hz...	1	0.2181750667604554	57.98522	4843.0501	2455.2132	1.537218155	2.17003449
ID104_T5_DAY2_logfile_fricatives_15ms_6_300Hz...	1	0.19176232993197218	52.49527	7343.6240	3047.8485	1.012492730	1.09684146
ID107_T2_DAY1_logfile_fricatives_15ms_6_300Hz...	1	0.3216985915655499	61.94534	5604.9243	2002.0424	1.839570443	6.40118458
ID107_T2_DAY2_logfile_fricatives_15ms_6_300Hz...	1	0.26232167958394115	60.46642	6239.6546	2080.9135	1.541029870	2.67942930
ID107_T5_DAY1_logfile_fricatives_15ms_6_300Hz...	1	0.24321670689781172	57.96996	5975.2941	1957.5270	1.852800844	5.02098140

Rows in this sample are just sorted first by 'number', then 'filename', so you can see different filenames:

```
d <- d %>% arrange(number, filename)
```

- As recommended above, these filenames were informative to keep the files distinct:
  - ID### = subject ID
  - T2 vs. T5 = pretest vs. posttest
  - DAY1 vs. DAY2 (you get it)

- So we could safely put them in the same directory and combine them (as described above).
- Now, we can make use of these informative variables embedded in the filenames...

Separating values into distinct columns:

- We thought ahead (You should too.) and separated each informative variable in our filenames by "_": a common, useful practice. This is why...

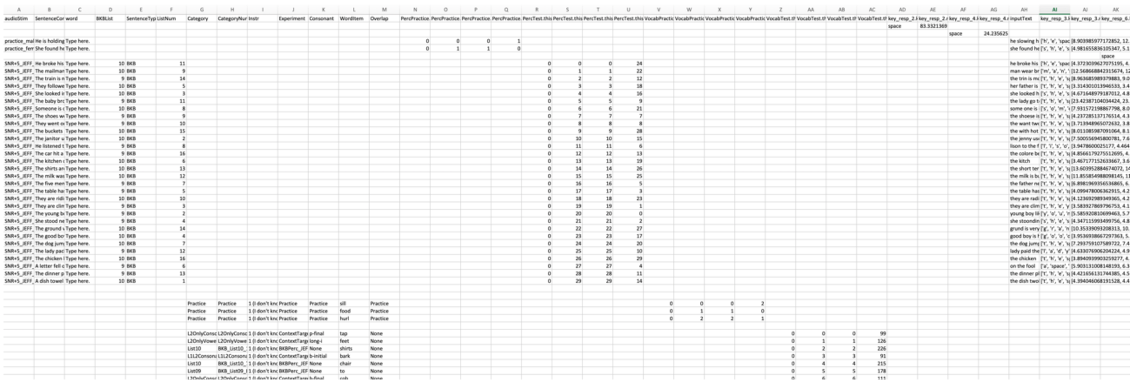
```
d <- d %>% separate(filename,c("subject","phase","day"),"_",extra="drop")
```

subject	phase	day	number	duration	intensity	cog	sdev	skew	kurt
ID104	T2	DAY1	1	0.28409036	57.82005	5059.4202	2579.6280	2.350851131	6.43765419
ID104	T2	DAY2	1	0.18354696	59.15340	10580.5580	2454.3661	0.189959972	-0.03913441
ID104	T5	DAY1	1	0.21817507	57.98522	4843.0501	2455.2132	1.537218155	2.17003449
ID104	T5	DAY2	1	0.19176233	52.49527	7343.6240	3047.8485	1.012492730	1.09684146
ID107	T2	DAY1	1	0.32169859	61.94534	5604.9243	2002.0424	1.839570443	6.40118458
ID107	T2	DAY2	1	0.26232168	60.46642	6239.6546	2080.9135	1.541029870	2.67942930
ID107	T5	DAY1	1	0.24321671	57.96996	5975.2941	1957.5270	1.852800844	5.02098140
ID107	T5	DAY2	1	0.08980949	62.87627	7064.9545	1749.5779	1.476542613	6.46937707

Saying `extra= "merge"` and specifying a fourth column name would have kept all the extra material in one column and ignored further "_" separations: useful for keeping around a 'comments' column (e.g., segmenting and noting comments w/ "_").

- This separation method would also be useful for column values like "`121/training/block1.csv`" — if you didn't have unique filenames with your files all in one place but, instead, had used folder structure to distinguish your files that you combined.
- (And dplyr's `unite()` command is a useful counterpart to `separate()`. It can be useful for cases like PsychoPy data, combining multiple response columns that are distinct only because they are in a different condition block, which may already be noted elsewhere in the row.)

Though see an introduction to `pivot()` below, converting column info into useful row material.



Recoding values:

- Now, it would just be nice if our values for pretest vs. posttest phase were a little clearer...

```
d <- d %>% mutate(phase= recode(phase, T2= "pretest", T5= "posttest"))
```

subject	phase	day	number	duration	intensity	cog	sdev	skew	kurt
ID104	pretest	DAY1	1	0.28409036	57.82005	5059.4202	2579.6280	2.350851131	6.43765419
ID104	pretest	DAY2	1	0.18354696	59.15340	10580.5580	2454.3661	0.189959972	-0.03913441
ID104	posttest	DAY1	1	0.21817507	57.98522	4843.0501	2455.2132	1.537218155	2.17003449
ID104	posttest	DAY2	1	0.19176233	52.49527	7343.6240	3047.8485	1.012492730	1.09684146
ID107	pretest	DAY1	1	0.32169859	61.94534	5604.9243	2002.0424	1.839570443	6.40118458
ID107	pretest	DAY2	1	0.26232168	60.46642	6239.6546	2080.9135	1.541029870	2.67942930
ID107	posttest	DAY1	1	0.24321671	57.96996	5975.2941	1957.5270	1.852800844	5.02098140
ID107	posttest	DAY2	1	0.08980949	62.87627	7064.9545	1749.5779	1.476542613	6.46937707

  - You could also have performed these commands at the same time for the same result:

```
d <- d %>%
     separate(filename,c("subject","phase","day"),"_",extra="drop") %>%
     mutate(phase= recode(phase, T2= "pretest", T5= "posttest"))
```

dplyr's `%>%` syntax allows you to immediately carry over the results of one command as the state and argument for subsequent commands.

4. Filtering/subsetting your data

Say you want to only look at some of the data you collected.

- Conditions of interest
  - For example, Say we want to just look at Day 1 of our experiment. The following command will limit our dataset to just those observations.

    ```
    d <- d %>% filter(day == "DAY1")
    ```

  - Or, Say we want to look at Day 1, and also posttest from Day2: i.e.,
    - We'll include all Day 1 data (pre- and post-test), and all posttest data.

      ```
      d <- d %>% filter(day == "DAY1" | phase == "posttest")
      ```

      Translation: Limit data to rows where day = "DAY1" or (|) where phase = "posttest".

  - Or, Say we want to look at only Day 1's posttest results: Either of the following commands will achieve this.

    ```
    d <- d %>% filter(day == "DAY1" & phase == "posttest")
    ```

    Translation: Limit data to rows where day = "DAY1" and phase = "posttest".

    ```
    d <- d %>% filter(day == "DAY1", phase == "posttest")
    ```

    Translation: Limit data to rows where day = "DAY1". Also limit data to rows where phase = "posttest".

  - Or, Say we want to look at 2-day learning: i.e., Say we want to compare Day 1's pretest and Day2's posttest.

    ```
    d <- d %>% filter( (day=="DAY1" & phase=="pretest") | (day=="DAY2" &
        phase=="posttest") )
    ```

    Can you provide the translation?

There's a whole range of ways that you can filter your data, which I won't fully expound here.

- You can use syntax like < and > to limit your data to ranges of numeric factor values.
  - useful for things like excluding outliers
- You can use syntax like `%in% c("...", "...")` to limit your data to values that are within a pre-specified set.
  - useful for narrowing your dataset to only certain categories
    - such as looking at only a certain set of sounds, while you've segmented and coded for a wide variety of sounds
- Or you can use `!=` if you know you just need to exclude rows matching a particular value.

Just browse online for dplyr functions and logical operators.

Sometimes you want to keep your data, but group it.

- There are other useful functions like `group_by()` and `summarize()`, which you can look up and learn more about. These can be useful for performing functions on your data that depend on certain conditions: e.g.,
  - creating a new variable (piped `%>%` before the `mutate()` function) dependent on criteria
  - collapsing condition groups into averages

5. Reshaping your data

Sometimes you'll be forced to deal with a data output that's very from the ideal state to work with.

Qualtrics and PsychoPy, for example, are two platforms that we use to get our data, but we don't have much say on what the output looks like.

For example, here is a Qualtrics study with a randomized set of stimulus exposures that we used the Loop & Merge tool to automatically plug in from a spreadsheet we uploaded.

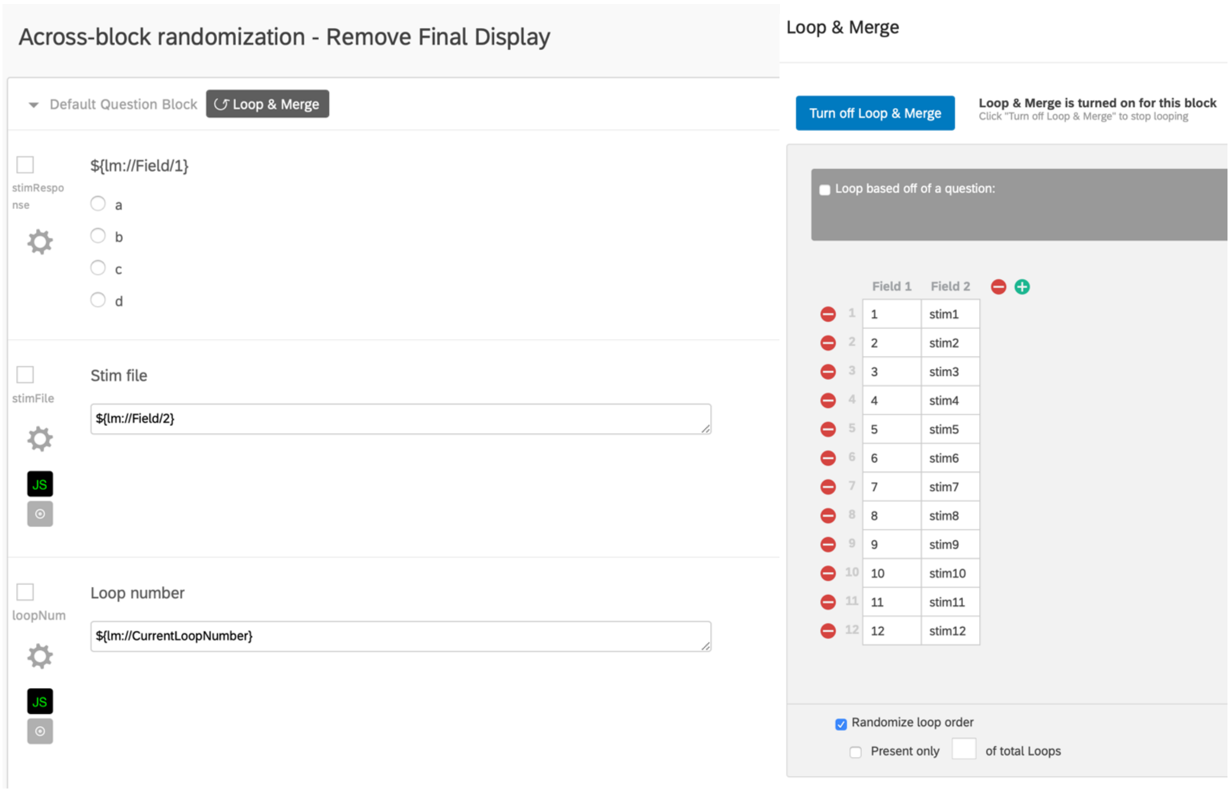
See my Qualtrics Workshop handout, and a specific example on using a randomized Loop & Merge, and tracking display order with invisible questions:

http://zjaggers.com/resources/QualtricsWorkshop/

http://zjaggers.com/resources/QualtricsWorkshop/Append1_LoopMergeWBlockBreaks.pdf

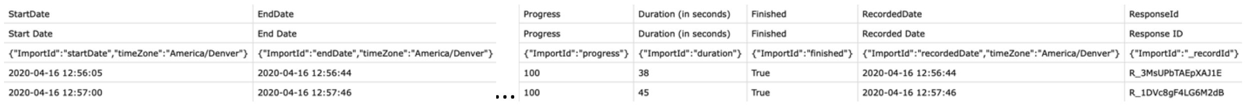
In the example experiment below, I've

- created a question that will display a stimulus (Field 1) and elicit a response {a,b,c,d}
- created an invisible question named 'stimFile' that will record a response as if the subject entered the contents of Field 2
- created another invisible question named 'loopNum' that will record a response as if the subject entered the current loop number



	Field 1	Field 2
1	1	stim1
2	2	stim2
3	3	stim3
4	4	stim4
5	5	stim5
6	6	stim6
7	7	stim7
8	8	stim8
9	9	stim9
10	10	stim10
11	11	stim11
12	12	stim12

I ran through this short demo experiment a couple times to show you the output:



StartDate	EndDate		Progress	Duration (in seconds)	Finished	RecordedDate	ResponseId
Start Date	End Date		Progress	Duration (in seconds)	Finished	Recorded Date	Response ID
{"ImportId":"startDate","timeZone":"America/Denver"}	{"ImportId":"endDate","timeZone":"America/Denver"}		{"ImportId":"progress"}	{"ImportId":"duration"}	{"ImportId":"finished"}	{"ImportId":"recordedDate","timeZone":"America/Denver"}	{"ImportId":"_recordId"}
2020-04-16 12:56:05	2020-04-16 12:56:44		100	38	True	2020-04-16 12:56:44	R_3MsUPbTAEpXAJ1E
2020-04-16 12:57:00	2020-04-16 12:57:46	...	100	45	True	2020-04-16 12:57:46	R_1DVc8gF4LG6M2dB

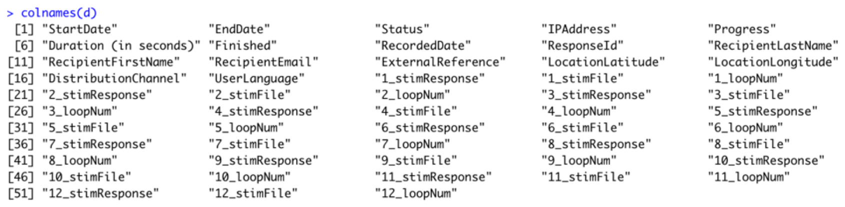
- You get a bunch of ID info, and for some reason 3 header rows, then your observation info.

- First, you can get rid of your first two ‘rows’, keeping just what R processed as the actual header and everything else: in this case, 2 separate subjects.

  ```
  d <- d %>% slice(3:n())
  ```

  Translation: Slice your data to keep only that starting from row 3 to the end (the row numbered with the total row count).

- Also, you can `select()` certain columns if you know only some are of interest.
  - You could run the command `colnames(d)` to see a numbered list of column names.



```
> colnames(d)
 [1] "StartDate"            "EndDate"             "Status"              "IPAddress"           "Progress"
 [6] "Duration (in seconds)" "Finished"           "RecordedDate"        "ResponseId"          "RecipientLastName"
[11] "RecipientFirstName"   "RecipientEmail"      "ExternalReference"   "LocationLatitude"    "LocationLongitude"
[16] "DistributionChannel"  "UserLanguage"        "1_stimResponse"      "1_stimFile"          "1_loopNum"
[21] "2_stimResponse"       "2_stimFile"          "2_loopNum"           "3_stimResponse"      "3_stimFile"
[26] "3_loopNum"            "4_stimResponse"      "4_stimFile"          "4_loopNum"           "5_stimResponse"
[31] "5_stimFile"           "5_loopNum"           "6_stimResponse"      "6_stimFile"          "6_loopNum"
[36] "7_stimResponse"       "7_stimFile"          "7_loopNum"           "8_stimResponse"      "8_stimFile"
[41] "8_loopNum"            "9_stimResponse"      "9_stimFile"          "9_loopNum"           "10_stimResponse"
[46] "10_stimFile"          "10_loopNum"          "11_stimResponse"     "11_stimFile"         "11_loopNum"
[51] "12_stimResponse"      "12_stimFile"         "12_loopNum"
```

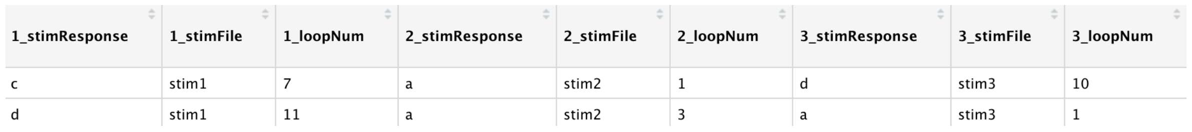
  - Then you could use `d <- d %>% select(1,9:53)` to pick columns 1 and 9-53.
  - Or you could identify a set like `colsOfInterest` and either include them by selecting that set or exclude them by selecting the negated set: e.g.,

    ```
    colsOfInterest <- c(“ResponseId”, “RecordedDate”, “UserLanguage”)
    d <- d %>% select(colsOfInterest)  #to include; Or...
    d <- d %>% select(-colsOfInterest) #to exclude
    ```

    Pop quiz: How does `select()` differ from `filter()`?

- Let’s say we selected the `colsOfInterest` we identified as subject ID info, and cols 18-53, the ones with observation info.

Now, let’s look at the observation info, past the subject ID info:



1_stimResponse	1_stimFile	1_loopNum	2_stimResponse	2_stimFile	2_loopNum	3_stimResponse	3_stimFile	3_loopNum
c	stim1	7	a	stim2	1	d	stim3	10
d	stim1	11	a	stim2	3	a	stim3	1

- For each loop, there are 3 columns: one for each question that had a ‘response’.
  - Each column starts with the row number of the loop.

    nb: This is not the contents of Field 1. This would have been “1_”, “2_”, etc. even if Field 1 of the L&M were “wug”, “schwa”, etc.
  - After “_” comes the question name.
- The cell contents provide the ‘responses’ stored (including automatic-because-hidden ones).
  - The ‘stimResponse’ column provides the actual answer to the multiple choice question.
  - ‘stimFile’ records Field 2: This is how we know the columns go in order of rows.
  - ‘loopNum’ records which loop pass in the randomized order this L&M row was activated.

Really, each of these triplets represents the factor values of a single observation per participant:

- There should just be columns for ‘stimResponse’, ‘stimFile’, and ‘loopNum’.
- And the contents of the ‘1_’, ‘2_’, ‘3_’, etc. groups should be stacked on top of each other
  - ...while repeating all of the subject ID info in each row.

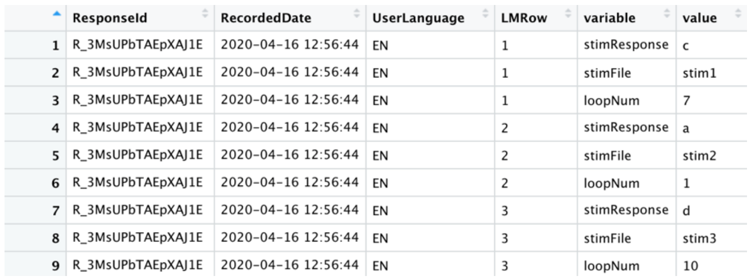
Here’s where we use dplyr’s `pivot( )` function:

- We’re wanting to ‘lengthen’ our dataset in the end.
  - But to do that, we’re actually going to reeeally lengthen it, and then widen it back some. Below, I’ll break this up into two parts. Here’s the full command in advance:

```
d <- d %>%
  pivot_longer(-colsOfInterest,
    names_to = c("LMRow", "variable"),
    names_sep = "_") %>%
  pivot_wider(names_from = "variable",
    values_from = "value")
```

- In the first part, we identify the columns that we want repeated as ID info for every observation. This will take the `value` of each cell in our data and make up only one (very long) column. But we keep track of what `variable` each value belonged to by extracting that from the original column name, as well as the L&M row.

```
pivot_longer(-colsOfInterest, #constant/ID info cols
  names_to = c("LMRow", "variable"), #destination of original col name
  names_sep = "_") #how original col name divided in 2
```

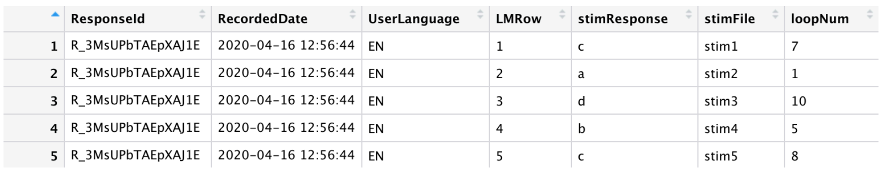


	ResponseId	RecordedDate	UserLanguage	LMRow	variable	value
1	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	1	stimResponse	c
2	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	1	stimFile	stim1
3	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	1	loopNum	7
4	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	2	stimResponse	a
5	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	2	stimFile	stim2
6	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	2	loopNum	1
7	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	3	stimResponse	d
8	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	3	stimFile	stim3
9	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	3	loopNum	10

- Now we've split up the info originally in the columns grouped and repeated per observation, while still keeping it all traceable. In the second part, we regroup the observation values by turning their variables back into columns again, this time leaving L&M rows as separate observation rows (like they should be) rather than column groups.

```
pivot_wider(names_from = "variable", #restore cols w/out LM row numbers
    values_from = "value") #source col of each new col*row's value
```

Remember/notice how we were able to pipe the results of the first part right into the second part.



	ResponseId	RecordedDate	UserLanguage	LMRow	stimResponse	stimFile	loopNum
1	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	1	c	stim1	7
2	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	2	a	stim2	1
3	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	3	d	stim3	10
4	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	4	b	stim4	5
5	R_3MsUPbTAEpXAJ1E	2020-04-16 12:56:44	EN	5	c	stim5	8

6. Conclusion

- This by no means covers all the tools for data management and processing.
- The takeaways are that...
  - You can process your data even when it's not in an ideal state.
  - It probably won't be in an ideal state no matter what you do.
  - There's a lot you can and should do to help your data be manageable.
    - It'll be a lot easier to get to that ideal state if you've kept it in mind and implemented measures along the way that'll facilitate these processes.