Managing Data in R

KEL - Quantitative Methods

While you are waiting for class to begin…

- Navigate to https://github.com/VTQuantMethodsEEB/klangwig
- Scroll down to week 2
- Download the slides and the R script for today
- Open the R script so you can run it while I talk

Housekeeping

- You need some data for this class (assignment one)
- If you still do not have data, and do not have a plan to acquire data (e.g. chatting with your advisor, surfing dryad, using some from a cool paper you recently read), we need to speak about your options ASAP.
- Please email me klangwig@vt.edu if you are worried about this.
- **I need your github username turned in before next class**

Assignment Reminder

- Please turn in your GitHub name and the paragraph about your data on canvas.
    - There is a text entry box to do this under the “assignments” tab. (You don’t need to send me an email or a canvas message.)

Goals

Improve data management through:

- Building pipelines
- Datasheet format
- File management
- Maintaining relational structure between data types
- Meta-data tracking

Why pipelines?

Pipelines are ways of carefully recording and systematizing the steps you take to work with your data

Ideally your project will depend on:

- Some data files: spreadsheets storing data (.csv files from Excel)
- Some scripts (more on this later)
- Something that tells you how these things go together: README file

Managing data files and folders

after you collect samples or receive results

Proper file (electronic and hard copy) management will save you TONS of time, headaches, and **most importantly downstream errors**.

Always…

- scan datasheets to save electronically and make copies
- retain unaltered original copies of datasheets or output files
- enter data from scanned sheets
- develop a quality control procedure

Consider…

- organizing as if you will continuously build the dataset and anticipate you will need to go back
- "*if someone continued this project, is my data pipeline in a place I could hand it over seamlessly?*"

Metadata and data dictionaries

preserving tangential information

How do you store data that is important but not specific to a single 'observation'?

- Net effort, environmental conditions during sampling event, site characteristics, etc
- Create a separate metadata file that includes the same variables as observational data (e.g., site, date, species)

Metadata and *data dictionaries*

preserving tangential information

How would someone stepping into the project understand the data you collect?

- We often collect data that isn't intuitive to everyone (wing score, RFA, reproductive condition, etc)
- ***Make a data dictionary***:
    - A table of definitions in an Excel spreadsheet (or other program)
    - Each variable from the datasheet is described thoroughly, including precisely what it measures

10/64

Record keeping

long-term pipeline success

You will inevitably forget your procedure.

- Write a protocol for how your files are organized/linked together
- Track any modifications using your README file or other notebook.
- ***Advantages***:
    - Allows more flexibility and time saving if you need someone to help
    - Provides an easy reference for yourself
    - Reminds you how you set up your pipeline if you need to be away from the project for any time

Now, data is collected, entered to spreadsheets, and filed appropriately!

Now you are ready to work with your data in R!

Read in your data

read.csv() is the function to use when reading your spreadsheets into R.

Using .csv files to store and read-in your data is important!

- Stores data in a text format (human-readable; transferrable)
- Can store large amounts of data in simple format
- Can be read by most programs

You can read in batdat, each row is a bat

```
# in this dataset, each row is a bat that
# is positive or negative for P. destructans
batdat=read.csv("/Users/klangwig/Desktop/VT/teaching/quant grad course/github/kla
head(batdat)
```

```
##      swab_id gd        gdL swab_type state          site    date species temp
## 1 KL15WI0002  1 0.00007560       BAT    WI HORSESHOE BAY 2/27/15    MYSE   NA
## 2 KL15WI0003  1 0.47879100       BAT    WI HORSESHOE BAY 2/27/15    MYLU   NA
## 3 KL15WI0004  0         NA       BAT    WI HORSESHOE BAY 2/27/15    MYLU   NA
## 4 KL15WI0005  1 0.00000551       BAT    WI HORSESHOE BAY 2/27/15    MYLU   NA
## 5 KL15WI0006  1 0.00003560       BAT    WI HORSESHOE BAY 2/27/15    MYLU   NA
## 6 KL15WI0007  1 0.00003160       BAT    WI HORSESHOE BAY 2/27/15    MYLU   NA
##   country count
## 1    u.s.     3
## 2    u.s.  1110
## 3    u.s.  1110
```

Getting Started with Data

- Save files as .csv
- IMPORTANT – saving an excel file as a CSV means that you will lose some data
- For example, if you used excel to calculate a formula, the formula will be gone as R will just store this as plain text
- DON'T USE EXCEL TO DO CALCULATIONS – JUST ADD THIS TO YOUR CODE IN R
- Use smart column names. R can't handle spaces in your column names, so get rid of those. Also don't use a bunch of capitals unnecessarily because it slows down your coding. e.g. use "species" not "Species"

Making your excel file

- Excel files should have a list of column names at the top only and variable values
- A reminder: your excel file should not look like your field data sheet!

Missing values

When you input data, you need to be aware of `NA` ("not available"). Your read function has an option called `na.strings` which you can use to communicate between R and your CSV files, for example. You need to know that

- use `is.na()` to test for `NA` values, `na.omit()` to drop them, and the optional `na.rm` argument in some functions (`mean`, `sum`, `median` ...)
- in the tidyverse, you can use `drop_na()` to remove `NA`

Examination

You should think creatively, and early on, about how to check your data. Is it internally consistent? Are there extreme outliers? Are there typos? Are there certain values that really mean something else?

An American Airlines memo about fuel reporting from the 1980s complained of multiple cases of:

- Reported departure fuel greater than aircraft capacity
- Reported departure fuel less than minimum required for trip
- Reported arrival fuel greater than reported departure fuel

You should think about what you can test, and what you can fix if it's broken.

Things to fix in excel

- naming inconsistencies (see maple example last lecture)
- column name issues (spaces)
- use excel's find and replace and filter function to find these

How do you clean data?

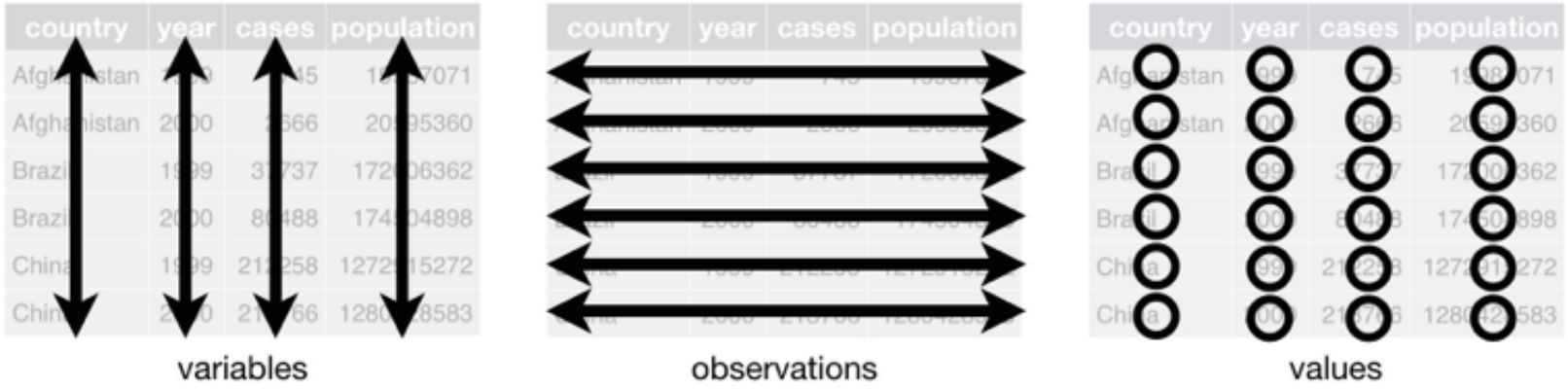
What R functions do you know that are useful for examination?
What are your strategies?

Tidy(ing) data

Hadley Wickham has defined a concept of tidy data, and has introduced the `tidyverse` package.

- Each variable is in a column
- Each observation is in a row
- “Long” rather than “wide” form
- Sometimes duplicates data
- Statistical modeling tools and graphical tools (especially the **ggplot2** package) in R work best with long form

An example of tidy data



Learning about the tidyverse

- https://www.tidyverse.org

Piping

- tidyverse syntax is VERY different from base R
- it relies on using %>%
- this is called a pipe
- it says the word “then”

Critical tools for managing data in tidyverse

- `pivot_longer`, `pivot_wider`
- `mutate`: add a column
- `select`: select columns
- `filter`: select rows
- `group_by`: group then do something (usually mutate or summarise)
- `summarise`: make a summary table
- `arrange`: sort
- `bind_rows`: combines dataframes by stacking them on top of each other - keeps all columns; will make duplicates if the

First load the package

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1
```

```
## ✓ ggplot2 3.3.5     ✓ purrr   0.3.4
## ✓ tibble  3.1.6     ✓ dplyr   1.0.7
## ✓ tidyr   1.1.4     ✓ stringr 1.4.0
## ✓ readr   2.1.1     ✓ forcats 0.5.1
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Group by

- `group_by` is my favorite tidyverse command which has cut my need to write loops in half
- `group_by` allows you to do calculations on groups of things, for example, by species or year
- `group_by` is kind of like using Excel's `filter`
- For example, if I group_by `species, date`, this is the equivalent in Excel of selecting a species (e.g. MYLU) and a specific date (e.g. March 1 2016, 3/1/16) except group_by does this for every species and date combination in your dataset
- This is incredibly useful because in other programs, you might need to write a for loop to have this capability

Group by species

```
batdat$lgdL = log10(batdat$gdL)
batdat %>%
  group_by(species) %>%
    summarise(mean.fungal.loads=mean(lgdL,na.rm=TRUE))
```

```
## # A tibble: 5 × 2
##   species   mean.fungal.loads
##   <chr>                 <dbl>
## 1 EPFU                  -3.64
## 2 MYLU                  -3.03
## 3 MYSE                  -3.69
## 4 PESU                  -2.04
## 5 SUBSTRATE             -4.11
```

Summarise versus Mutate

- `summarise` creates a new dataframe
- `mutate` does a calculation where it add a new column to your existing dataframe

Importance of assigning

```
fungal.load.table = batdat %>%
  group_by(species) %>%
  summarise(mean.fungal.loads=mean(lgdL,na.rm=TRUE))

fungal.load.table
```

```
## # A tibble: 5 × 2
##   species   mean.fungal.loads
##   <chr>                 <dbl>
## 1 EPFU                  -3.64
## 2 MYLU                  -3.03
## 3 MYSE                  -3.69
## 4 PESU                  -2.04
## 5 SUBSTRATE             -4.11
```

Assigning

- When using summarise, it's best to call your summarised object a new name (e.g. fungal.load.table)
- This is typically not necessary when using mutate which just adds a column to an existing dataset

```
batdat = batdat %>%
  #take batdat, then group by somethings
  #we re-assign batdat to batdat because we want to add the column to that datafi
    group_by(site,species,date) %>%
  #you can group_by multiple things
    mutate(sample.size=n())
#this adds a column to the dataframe
#using the function n(), which counts things (e.g n rows in the group)
```

What does our dataframe look like now?

```
head(batdat %>%
       select(c("swab_id", "site","species","date","gd","sample.size")))
```

```
## # A tibble: 6 × 6
## # Groups:   site, species, date [2]
##   swab_id    site           species date       gd sample.size
##   <chr>      <chr>          <chr>   <chr>   <int>       <int>
## 1 KL15WI0002 HORSESHOE BAY MYSE    2/27/15     1           4
## 2 KL15WI0003 HORSESHOE BAY MYLU    2/27/15     1          20
## 3 KL15WI0004 HORSESHOE BAY MYLU    2/27/15     0          20
## 4 KL15WI0005 HORSESHOE BAY MYLU    2/27/15     1          20
## 5 KL15WI0006 HORSESHOE BAY MYLU    2/27/15     1          20
## 6 KL15WI0007 HORSESHOE BAY MYLU    2/27/15     1          20
```

#this is just showing a few columns for effect

Joining

- Joining datasets together is a useful skill, especially if we have two datasets we need to match on a specific column or set of columns
- https://dplyr.tidyverse.org/reference/mutate-joins.html
- Always call your new dataframe something new; don't write over and old dataframe in case you make a mistake joining

Join functions

- `inner_join()`: includes all rows in x and y.
- when inner joining, non-matching rows will be dropped!
- `left_join()`: includes all rows in x.
- when left joining, every row in x is kept, but only those matching x are kept in y
- `right_join()`: includes all rows in y.
- when right joining, every row in y is kept, but only those matching y are kept in x
- `full_join()`: includes all rows in x or y.
- every row is kept in both x and y

Joining actual datasets

- We frequently have data coming from different sources (lab, field, etc.)
- Let's join our infection data with our count data
- We might want to do this to ask 'How does the count of one species influence another infection in that species?'
- A note - a previous merge I did added this to your data but let's pretend it isn't there…

Example with bat count data

- We are making a dataset here using aggregate but imagine this came from a different file

```
batcount<-aggregate(count~species+site+date,data=batdat, FUN=mean)
head(batcount)
```

```
##   species          site     date count
## 1    MYLU      ST. JOHN 11/20/15    87
## 2    MYLU HORSESHOE BAY  11/7/15   646
## 3    MYSE HORSESHOE BAY  11/7/15     1
## 4    MYLU    BEAR CREEK  11/9/15   116
## 5    MYSE    BEAR CREEK  11/9/15     7
## 6    PESU    BEAR CREEK  11/9/15    50
```

Join batdat and batcount

```
batdat_count = left_join(
  x = batdat,
  y = batcount,
  by = c("site","species","date")
  #columns to join on
)
head(batdat_count %>%
       select(c("swab_id","site","date","species","count.y")))
```

```
## # A tibble: 6 × 5
## # Groups:   site, species, date [2]
##   swab_id    site          date    species count.y
##   <chr>      <chr>         <chr>   <chr>     <dbl>
## 1 KL15WI0002 HORSESHOE BAY 2/27/15 MYSE          3
## 2 KL15WI0003 HORSESHOE BAY 2/27/15 MYLU       1110
## 3 KL15WI0004 HORSESHOE BAY 2/27/15 MYLU       1110
```

Pivoting

- https://tidyr.tidyverse.org/articles/pivot.html
- sometimes our datasets are not in the format we want for an analysis

Why pivot?

```
head(batdat_count)
```

```
## # A tibble: 6 × 14
## # Groups:   site, species, date [2]
##   swab_id       gd         gdL swab_type state site  date  species  temp count
##   <chr>      <int>       <dbl> <chr>     <chr> <chr> <chr> <chr>   <dbl> <chr>
## 1 KL15WI0002     1  0.0000756  BAT       WI    HORS… 2/27… MYSE       NA u.s.
## 2 KL15WI0003     1  0.479      BAT       WI    HORS… 2/27… MYLU       NA u.s.
## 3 KL15WI0004     0 NA          BAT       WI    HORS… 2/27… MYLU       NA u.s.
## 4 KL15WI0005     1  0.00000551 BAT       WI    HORS… 2/27… MYLU       NA u.s.
## 5 KL15WI0006     1  0.0000356  BAT       WI    HORS… 2/27… MYLU       NA u.s.
## 6 KL15WI0007     1  0.0000316  BAT       WI    HORS… 2/27… MYLU       NA u.s.
## # … with 4 more variables: count.x <int>, lgdL <dbl>, sample.size <int>,
## #   count.y <dbl>
```



```
# right now, species is in long format
```

`pivot_wider` is how we take long data, and make it wide

```
batcounts.wide<- batcount %>%
  #this says - make a new df called batcounts.wide using bat counts
  pivot_wider(names_from = species,
              values_from = count
              )
##make columns for each of the values in the species column and fill those colum
head(batcounts.wide)
```

```
## # A tibble: 6 × 6
##   site          date      MYLU  MYSE  PESU  EPFU
##   <chr>         <chr>    <dbl> <dbl> <dbl> <dbl>
## 1 ST. JOHN      11/20/15    87    NA    NA    NA
## 2 HORSESHOE BAY 11/7/15    646     1    NA    NA
```

adding 0s into a `pivot`

- when we perform this, it automatically fills missing with NAs but we know that missing actually mean 0

```
batcounts.wide = batcount %>%
  pivot_wider(
  names_from = species,
  values_from = count,
  values_fill = 0
)
head(batcounts.wide)
```

```
## # A tibble: 6 × 6
##   site          date      MYLU  MYSE  PESU  EPFU
##   <chr>         <chr>    <dbl> <dbl> <dbl> <dbl>
## 1 ST. JOHN      11/20/15    87     0     0     0
## 2 HORSESHOE BAY 11/7/15    646     1     0     0
```

Going from wide to long format

- more often, we might be working in the opposite direction
- A more common issue is that programs output information in columns that we would rather have in rows

```
batcounts.long = batcounts.wide %>%
  pivot_longer(
    cols = c("EPFU","MYLU","MYSE","PESU"),
    #what are the existing columns I want to make into rows?
    names_to =   "species",
    #put the names of the columns in a column called 'species'
    values_to = "count"
    #the values that were in each of the columns get moved to a column called 'c
  )
```

Check out new batcounts.long

```
head(batcounts.long)
```

```
## # A tibble: 6 × 4
##   site          date     species count
##   <chr>         <chr>    <chr>   <dbl>
## 1 ST. JOHN      11/20/15 EPFU        0
## 2 ST. JOHN      11/20/15 MYLU       87
## 3 ST. JOHN      11/20/15 MYSE        0
## 4 ST. JOHN      11/20/15 PESU        0
## 5 HORSESHOE BAY 11/7/15  EPFU        0
## 6 HORSESHOE BAY 11/7/15  MYLU      646
```

So how do we create tidy datasets?

- Make your data as tidy as possible
- Learn to manipulate data in R and hardcode these changes into your scripts
- There is no perfect method - each dataset is unique
- Manipulating data in R is hard, sometimes harder than excel. But learning to do it SO worth it because you will save hours of time for each project you do.

Managing Pipelines in R

- Pipelines are ways of carefully recording and systematizing the steps you take to work with your data
- The idea is that you should be able to delete any results of computer calculations and be able to quickly re-do them
- Ideally your project will depend on:
    - Some data files
    - Some scripts
    - Something that tells you how these things go together (RMarkdown is helpful for this), at minimum a README file

Advantages of this approach

- Clarity: we aren't confused about the 600 pages of information stored with our projects
- Reproducibility: we can always re-do something we did
- Flexibility : we can use different data and re-create the same thing

Spreadsheets

- Spreadsheets are a useful tool for working with R
- `read.csv` and `write.csv` are very useful commands for working with spreadsheets
- when using `write.csv` use `row.names=F` to avoid line numbers
- Importantly, spreadsheets are for storing data, NOT FOR MANIPULATING DATA
    - Your goal should be to take data from a spreadsheet and manipulate it entirely using scripts.
    - Avoid spreadsheet addiction: http://www.burns-stat.com/documents/tutorials/spreadsheet-addiction/
    - The jist is: friends don't let friends use excel for statistics.

47/64

Database

- Your spreadsheet is a database (just because it isn't stored in microsoft access doesn't mean it isn't!)
- "small" databases are usually considered to be fewer than 1000 observations of 10-20 vars
- "medium" databases are about 1000 to 100,000 observations of about 10-50 vars. These are most helpful with data handling packages.
- "large" means millions of observations and potentially 1000s of variables. These may need to be stored in an external application.

Working in Github

- Git is version control system, with the original purpose of allowing groups to work collaboratively on software projects
- Git manages the evolution of a set of files - called a repository
- A repository is essentially a folder where you store your stuff
- Version control works a bit like “Track Changes” in word, Git will track the changes we make to our code so we can return to previous versions
- It also allows collaboration so I can look at your code and make changes - a bit like a more complicated version of Google Docs

Will this hurt?

- Maybe!
- But, I think this important enough that we NEED exposure to this. This is the future!

But I only code alone!

- You need to carefully document your steps if the only person you are sharing code with is the future version of yourself
- In addition, most journals require publicly available data and code - open code is the norm, not the exception.
- Using Git has gotten easier. We used to have to use command line to communicate with Git, but now we can just use RStudio!

Terminology

- repository: A directory or storage space where your projects can live. Sometimes GitHub users shorten this to “repo.” (If you’re cool like that.) It is usually a local folder on your computer. You can keep code files, text files, image files, you name it, inside a repository.

- commit: This is the command that gives Git its power. When you commit, you are taking a “snapshot” of your repository at that point in time, giving you a checkpoint to which you can reevaluate or restore your project to any previous state.When you first start “commiting”, it is important to remember this is taking the picture, not SENDING the picture. (Sending is called “pushing”)

Terminology cont.

- branch: How do multiple people work on a project at the same time without Git getting them confused? Usually, they "branch off" of the main project with their own versions full of changes they themselves have made. After they're done, it's time to "merge" that branch back with the "master," the main directory of the project. Because we'll be working within our own repos, we don't need to worry too much about branching but is good to know for future.

- push: This is how you upload your file to GitHub. Remember, you need to both commit and push for your file to be sent to GitHub.

Sending your files to our class repository

- We have an “organization” account for our class
- Normally, we would have to pay for private repositories,github is giving us UNLIMITED private repositories. That’s pretty awesome.
- Why should we want things open-source? Why not?

Installing Git

- Please try to start this before our next class.
- Here is a link: http://happygitwithr.com/install-git.html#install-git
- Please follow instructions to get started with git.
- Try to install git in the most scientific way possible - if one way doesn't work, try the next, and google your mistakes!

Here are some other example using "pivot" - skipped for time

Look at some example data that comes with the tidyr package:

```
fish_encounters
```

```
## # A tibble: 114 × 3
##    fish  station  seen
##    <fct> <fct>   <int>
##  1 4842  Release     1
##  2 4842  I80_1       1
##  3 4842  Lisbon      1
##  4 4842  Rstr        1
##  5 4842  Base_TD     1
##  6 4842  BCE         1
##  7 4842  BCW         1
```

Pivot_wider

Using `pivot_wider()`

```
fish_encounters %>%
  pivot_wider(names_from = station, values_from = seen)
```

```
## # A tibble: 19 × 12
##    fish  Release I80_1 Lisbon  Rstr Base_TD   BCE   BCW  BCE2  BCW2   MAE   MA
##    <fct>   <int> <int>  <int> <int>   <int> <int> <int> <int> <int> <int> <int
##  1 4842        1     1      1     1       1     1     1     1     1     1
##  2 4843        1     1      1     1       1     1     1     1     1     1
##  3 4844        1     1      1     1       1     1     1     1     1     1
##  4 4845        1     1      1     1       1    NA    NA    NA    NA    NA     N
##  5 4847        1     1      1    NA      NA    NA    NA    NA    NA    NA     N
##  6 4848        1     1      1     1      NA    NA    NA    NA    NA    NA     N
##  7 4849        1     1     NA    NA      NA    NA    NA    NA    NA    NA     N
##  8 4850        1     1     NA     1       1     1     1    NA    NA    NA     N
```

Fill in 0's

```
fish_encounters %>%
  pivot_wider(
    names_from = station,
    values_from = seen,
    values_fill = list(seen = 0)
  )
```

```
## # A tibble: 19 × 12
##    fish  Release I80_1 Lisbon  Rstr Base_TD   BCE   BCW  BCE2  BCW2   MAE   MA
##    <fct>   <int> <int>  <int> <int>   <int> <int> <int> <int> <int> <int> <int
##  1 4842        1     1      1     1       1     1     1     1     1     1
##  2 4843        1     1      1     1       1     1     1     1     1     1
##  3 4844        1     1      1     1       1     1     1     1     1     1
##  4 4845        1     1      1     1       1     0     0     0     0     0
##  5 4847        1     1      1     0       0     0     0     0     0     0
##  6 4848        1     1      1     1       0     0     0     0     0     0
```

Making a dataframe long (e.g. tidy)

Let's look at an example of an untidy dataframe.

```
head(relig_income)
```

```
## # A tibble: 6 × 11
##   religion  `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-10(
##   <chr>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dl
## 1 Agnostic       27        34        60        81        76       137        1
## 2 Atheist        12        27        37        52        35        70
## 3 Buddhist       27        21        30        34        33        58
## 4 Catholic      418       617       732       670       638      1116        9
## 5 Don't kn…      15        14        15        11        10        35
## 6 Evangeli…     575       869      1064       982       881      1486        9
## # … with 3 more variables: $100-150k <dbl>, >150k <dbl>,
## #   Don't know/refused <dbl>
```

59/64

Make a row for the number of individuals for each religion by income category

```
relig_income %>%
  pivot_longer(-religion, names_to = "income", values_to = "count")
```

```
## # A tibble: 180 × 3
##    religion income            count
##    <chr>    <chr>             <dbl>
##  1 Agnostic <$10k                27
##  2 Agnostic $10-20k              34
##  3 Agnostic $20-30k              60
##  4 Agnostic $30-40k              81
##  5 Agnostic $40-50k              76
##  6 Agnostic $50-75k             137
##  7 Agnostic $75-100k            122
##  8 Agnostic $100-150k           109
```

Another example using temporal data:

The billboard dataset has a row for every week and the rank of that song

```
billboard
```

```
## # A tibble: 317 × 79
##    artist   track   date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   w
##    <chr>    <chr>   <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
##  1 2 Pac    Baby D… 2000-02-26      87    82    72    77    87    94    99
##  2 2Ge+her  The Ha… 2000-09-02      91    87    92    NA    NA    NA    NA
##  3 3 Doors… Krypto… 2000-04-08      81    70    68    67    66    57    54
##  4 3 Doors… Loser   2000-10-21      76    76    72    69    67    65    55
##  5 504 Boyz Wobble… 2000-04-15      57    34    25    17    17    31    36
##  6 98^0     Give M… 2000-08-19      51    39    34    26    26    19     2
##  7 A*Teens  Dancin… 2000-07-08      97    97    96    95   100    NA    NA
##  8 Aaliyah  I Don'… 2000-01-29      84    62    51    41    38    35    35
```

We want week to be temporal data, but it has letters in it

- We want names to be a variable called “week” and the values to be a variable called “rank”
- We want to remove NAs because not all songs stay on the charts for 76 weeks

Billboard

```
billboard %>%
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    names_prefix = "wk",
    values_to = "rank",
    values_drop_na = TRUE
  )
```

```
## # A tibble: 5,307 × 5
##    artist  track                   date.entered week   rank
##    <chr>   <chr>                   <date>       <chr> <dbl>
##  1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   1        87
##  2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   2        82
##  3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   3        72
##  4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   4        77
```

Changing something to an integer

- We want to turn week into an integer so we can easily determine how long a song was on the charts

```
billboard %>%
  pivot_longer (
    cols = starts_with("wk"),
    names_to = "week",
    names_prefix = "wk",
    values_to = "rank",
    values_drop_na = TRUE
)
```

```
## # A tibble: 5,307 × 5
##    artist  track                   date.entered week   rank
##    <chr>   <chr>                   <date>       <chr> <dbl>
##  1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   1        87
```