Practical R: Data Ingestion and Munging

Abhijit Dasgupta

Fall, 2019

A quick refresh

- We talked about various data structures in R
- The primacy of the `data.frame`
  - Extracting individual variables from a data frame
  - `breast_cancer$ER.Status`, `breast_cancer[,'ER.Status']`, `breast_cancer[['ER.Status']]`
  - Extracting rows of a `data.frame`
- Identifying data classes using the `class` function
- Recognizing different classes: `numeric`, `character`, `factor`, `Date`, ..
  - testing for a class: `is.numeric`
  - converting to a class: `as.numeric`

A note on factors

Factors

- Factors are stored internally as integers, with *meta-data* in the form of text labels
    - There is an inherent ordering of labels, by default alphabetically
- Individual levels of a factor are treated as *separate* but related variables (dummy variables)

```
breast_cancer <- read_csv('data/clinical_data_breast_cancer_modified.csv')
names(breast_cancer) <- make.names(names(breast_cancer))
breast_cancer$ER.Status.f <- factor(breast_cancer$ER.Status)
summary(breast_cancer$ER.Status)
```

```
#>      Length      Class       Mode
#>         105 character character
```

```
summary(breast_cancer$ER.Status.f)
```

```
#>  Indeterminate      Negative      Positive
#>              1            36            68
```

Factors

```
breast_cancer$ER.Status.f <- fct_relevel(breast_cancer$ER.Status.f, 'Negative')
summary(breast_cancer$ER.Status.f)
```

```
#>      Negative Indeterminate      Positive
#>            36             1            68
```

This is manipulating the meta-data, not the actual data itself

Factors

```
breast_cancer$ER.Status.n <- as.numeric(breast_cancer$ER.Status.f)
summary(breast_cancer$ER.Status.n)
```

```
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#>    1.000    1.000    3.000    2.305    3.000    3.000
```

Logistic regression of death status on ER status

```
#>  # A tibble: 2 x 2
#>    term        estimate
#>    <chr>          <dbl>
#>  1 (Intercept)    1.81
#>  2 ER.Status.n    0.148
```

Only one coefficient, since levels are modeled as numeric, with one slope being estimated

```
#>  # A tibble: 3 x 2
#>    term                     estimate
#>    <chr>                       <dbl>
#>  1 (Intercept)                 2.08
#>  2 ER.Status.fIndeterminate  -17.6
#>  3 ER.Status.fPositive         0.256
```

One coefficient for all but one factor level

RMarkdown tip of the day

You can add options to each R chunk to add or suppress output

Option	Property
echo=T/F	Does the document show the R code
eval=T/F	Does the chunk get evaluated by R
message=T/F	Do messages get printed
warning=T/F	Do warnings get printed

You can also set these once per session by putting the following in a R chunk:

```
knitr::opts_chunk(echo=T, eval=T, message=F, warning=F)
```

See here for the full gory details

Data ingestion

Data ingestion

Unlike Excel, you have to pull data into R for R to operate on it

Typically your data is in some sort of file (Excel, csv, sas7bdat, dta, txt)

You need to find a way to pull it into R

The GUI you've used is one way, but not very programmatic

Data ingestion

Type	Function	Package	Notes
csv	read_csv	readr	Takes care of formatting
csv	read.csv	base	Built in
csv	fread	data.table	Fastest
Excel	read_excel	readxl	
sas7bdat	read_sas	haven	SAS format
sav	read_spss	haven	SPSS format
dta	read_dta	haven	Stata format

Data ingestion

We will use this csv data and this Excel data for the following:

```
brca_clinical <- readr::read_csv('data/BreastCancer_Clinical.csv')
brca_clinical2 <- data.table::fread('data/BreastCancer_Clinical.csv')
```

```
str(brca_clinical)
```

```
#>  Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.
#>   $ Complete TCGA ID                   : chr  "TCG
#>   $ Gender                             : chr  "FEM
#>   $ Age at Initial Pathologic Diagnosis: num  66 4
#>   $ ER Status                          : chr  "Neg
#>   $ PR Status                          : chr  "Neg
#>   $ HER2 Final Status                  : chr  "Neg
#>   $ Tumor                              : chr  "T3"
#>   $ Tumor--T1 Coded                    : chr  "T_O
#>   $ Node                               : chr  "N3"
#>   $ Node-Coded                         : chr  "Pos
#>   $ Metastasis                         : chr  "M1"
#>   $ Metastasis-Coded                   : chr  "Pos
#>   $ AJCC Stage                         : chr  "Sta
#>   $ Converted Stage                    : chr  "No_
#>   $ Survival Data Form                 : chr  "fol
#>   $ Vital Status                       : chr  "DEC
#>   $ Days to Date of Last Contact       : num  240
```

```
str(brca_clinical2)
```

```
#>  Classes 'data.table' and 'data.frame':    105 obs
#>   $ Complete TCGA ID                   : chr  "TCG
#>   $ Gender                             : chr  "FEM
#>   $ Age at Initial Pathologic Diagnosis: int  66 4
#>   $ ER Status                          : chr  "Neg
#>   $ PR Status                          : chr  "Neg
#>   $ HER2 Final Status                  : chr  "Neg
#>   $ Tumor                              : chr  "T3"
#>   $ Tumor--T1 Coded                    : chr  "T_O
#>   $ Node                               : chr  "N3"
#>   $ Node-Coded                         : chr  "Pos
#>   $ Metastasis                         : chr  "M1"
#>   $ Metastasis-Coded                   : chr  "Pos
#>   $ AJCC Stage                         : chr  "Sta
#>   $ Converted Stage                    : chr  "No_
#>   $ Survival Data Form                 : chr  "fol
#>   $ Vital Status                       : chr  "DEC
#>   $ Days to Date of Last Contact       : int  240
```

A note on two "super"-data.frame objects

A `tibble`

```
#>  # A tibble: 6 x 30
#>    `Complete TCGA … Gender `Age at Initial… `ER St
#>    <chr>            <chr>             <dbl> <chr>
#>  1 TCGA-A2-A0T2     FEMALE               66 Negati
#>  2 TCGA-A2-A0CM     FEMALE               40 Negati
#>  3 TCGA-BH-A18V     FEMALE               48 Negati
#>  4 TCGA-BH-A18Q     FEMALE               56 Negati
#>  5 TCGA-BH-A0E0     FEMALE               38 Negati
#>  6 TCGA-A7-A0CE     FEMALE               57 Negati
#>  # … with 25 more variables: `HER2 Final Status` <
#>  #   `Tumor--T1 Coded` <chr>, Node <chr>, `Node-Co
#>  #   Metastasis <chr>, `Metastasis-Coded` <chr>, `
#>  #   `Converted Stage` <chr>, `Survival Data Form`
#>  #   Status` <chr>, `Days to Date of Last Contact`
#>  #   Death` <dbl>, `OS event` <dbl>, `OS Time` <db
#>  #   `SigClust Unsupervised mRNA` <dbl>, `SigClust
#>  #   `miRNA Clusters` <dbl>, `methylation Clusters
#>  #   Clusters` <chr>, `CN Clusters` <dbl>, `Integr
#>  #   PAM50)` <dbl>, `Integrated Clusters (no exp)`
#>  #   Clusters (unsup exp)` <dbl>
```

A `data.table`

```
#>     Complete TCGA ID Gender Age at Initial Patholo
#>  1:     TCGA-A2-A0T2 FEMALE
#>  2:     TCGA-A2-A0CM FEMALE
#>  3:     TCGA-BH-A18V FEMALE
#>  4:     TCGA-BH-A18Q FEMALE
#>  5:     TCGA-BH-A0E0 FEMALE
#>  6:     TCGA-A7-A0CE FEMALE
#>     PR Status HER2 Final Status Tumor Tumor--T1 Co
#>  1:  Negative          Negative    T3          T_Ot
#>  2:  Negative          Negative    T2          T_Ot
#>  3:  Negative          Negative    T2          T_Ot
#>  4:  Negative          Negative    T2          T_Ot
#>  5:  Negative          Negative    T3          T_Ot
#>  6:  Negative          Negative    T2          T_Ot
#>     Metastasis Metastasis-Coded AJCC Stage Convert
#>  1:         M1         Positive   Stage IV   No_Co
#>  2:         M0         Negative  Stage IIA       S
#>  3:         M0         Negative  Stage IIB   No_Co
#>  4:         M0         Negative  Stage IIB   No_Co
#>  5:         M0         Negative Stage IIIC   No_Co
#>  6:         M0         Negative  Stage IIA       S
#>     Survival Data Form Vital Status Days to Date o
#>  1:           followup     DECEASED
#>  2:           followup     DECEASED
#>  3:         enrollment     DECEASED
```

A note on two "super"-data.frame objects

- A `tibble` works pretty much like any `data.frame`, but the printing is a little saner
- A `data.table` is faster, has more inherent functionality, but has a ver different syntax

We'll work almost entirely with `tibble`'s and not `data.table`

Suggested modifications:

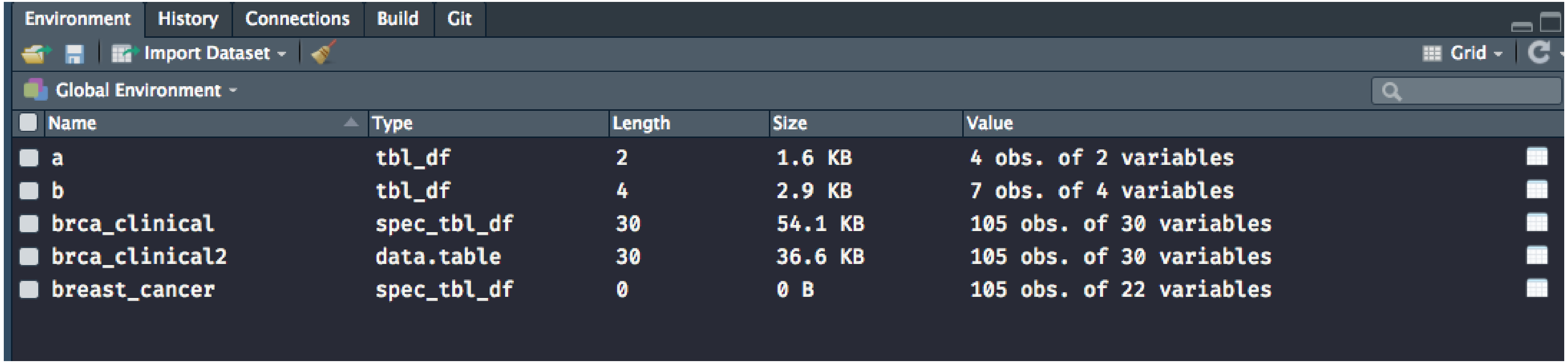
- If using `fread`, convert the resulting object to a `data.frame` or `tibble` using `as_data_frame()` or `as_tibble`
- Convert the column names to not have spaces using, for example,

```
names(brca_clinical) <- make.names(names(brca_clinical))
```

Data ingestion

Note that you **have** to give a name to what you're importing using `read_*` or whatever you're using, otherwise it won't stay in R

```
brca_clinical <- readr::read_csv('data/BreastCancer_Clinical.csv')
```



Reading Excel

```
excel_sheets('data/BreastCancer.xlsx')
```

```
#>  [1] "Cllinical"  "Expression"
```

```
brca_expression <- readxl::read_excel('data/BreastCancer.xlsx', sheet='Expression')
```

Data export

Data export

Type	Function	Package	Notes
csv	write_csv	readr	Takes care of formatting
csv	write.csv	base	Built in
csv	fwrite	data.table	Fastest
Excel	write.xlsx	openxlsx	
sas7bdat	write_sas	haven	SAS format
sav	write_spss	haven	SPSS format
dta	write_dta	haven	Stata format

We'll often save tabular results using these functions

Simplifying import/export

We'll be using a package that makes this easier.

It's called `rio` and it has two basic functions: `import` and `export`.

The `rio` package uses the different packages mentioned earlier but unifies it into a single syntax

Classwork

Open an Rmd file, and create a R chunk where you use the function `import` from `rio` to load the clinical breast cancer data into R

- Note, you have to "activate" the `rio` package in the chunk
- You have to save the imported object by giving it a name

10:00

Data munging

The tidyverse

What is the tidyverse?

> The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures. -- Dr. Hadley Wickham

- A human-friendly syntax and semantics to make code more understandable
- The functions in the tidyverse often wraps harder-to-understand functions into simpler, more understandable forms
- We're taking an opinionated choice here
    - Covers maybe 85% of the cases you'll ever face
    - Takes a particular viewpoint about how data *should* be organized
- But this makes things easier and simpler

What's tidy here?

The way data is organized in a data frame is **tidy** in this framework.

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

In practical terms:

1. Put data in a data frame / *tibble*
2. Make sure each variable is in its own column

Tidy data

A first step in the tidyverse is to activate the `tidyverse` meta-package

```
library(tidyverse)
```

- **ggplot2**: Create Elegant Data Visualisations Using the Grammar of Graphics
- **purrr**: Functional Programming Tools
- **readr**: Read Rectangular Text Data
- **tidyr**: Easily Tidy Data with 'spread()' and 'gather()' Functions
- **dplyr**: A Grammar of Data Manipulation
- **forcats**: Tools for Working with Categorical Variables (Factors)
- **lubridate**: Make Dealing with Dates a Little Easier
- **stringr**: Simple, Consistent Wrappers for Common String Operations

Tidy data

The common feature of all these packages is that their functions take a data frame (which the tidyverse calls a `tibble`) as their first argument.

So the starting point for any analysis is the data set.

Tidy data

```
table1
```

```
#>  # A tibble: 6 x 4
#>    country      year  cases population
#>    <chr>       <int>  <int>      <int>
#>  1 Afghanistan  1999    745   19987071
#>  2 Afghanistan  2000   2666   20595360
#>  3 Brazil       1999  37737  172006362
#>  4 Brazil       2000  80488  174504898
#>  5 China        1999 212258 1272915272
#>  6 China        2000 213766 1280428583
```

Is this tidy?

Tidy data

```
table2
```

```
#>  # A tibble: 12 x 4
#>     country      year type            count
#>     <chr>       <int> <chr>           <int>
#>   1 Afghanistan  1999 cases             745
#>   2 Afghanistan  1999 population   19987071
#>   3 Afghanistan  2000 cases            2666
#>   4 Afghanistan  2000 population   20595360
#>   5 Brazil       1999 cases           37737
#>   6 Brazil       1999 population  172006362
#>   7 Brazil       2000 cases           80488
#>   8 Brazil       2000 population  174504898
#>   9 China        1999 cases          212258
#>  10 China        1999 population 1272915272
#>  11 China        2000 cases          213766
#>  12 China        2000 population 1280428583
```

Is this tidy?

Tidy data

```
table3
```

```
#>  # A tibble: 6 x 3
#>    country      year rate
#>  * <chr>       <int> <chr>
#>  1 Afghanistan  1999 745/19987071
#>  2 Afghanistan  2000 2666/20595360
#>  3 Brazil       1999 37737/172006362
#>  4 Brazil       2000 80488/174504898
#>  5 China        1999 212258/1272915272
#>  6 China        2000 213766/1280428583
```

Is this tidy?

Tidy data

```
table4a # cases
```

```
#>   # A tibble: 3 x 3
#>     country     `1999` `2000`
#>   * <chr>        <int>  <int>
#>   1 Afghanistan    745   2666
#>   2 Brazil       37737  80488
#>   3 China       212258 213766
```

```
table4b # population
```

```
#>   # A tibble: 3 x 3
#>     country         `1999`     `2000`
#>   * <chr>            <int>      <int>
#>   1 Afghanistan   19987071   20595360
#>   2 Brazil       172006362  174504898
#>   3 China       1272915272 1280428583
```

Are these tidy?

Can we make datasets tidy?

Sometimes. The functions in the `tidyr` package can help

- `separate` is a function that can split a column into multiple columns
    - When there are multiple variables together in a column

```
table3
```

```
#>  # A tibble: 6 x 3
#>    country      year rate
#>  * <chr>       <int> <chr>
#>  1 Afghanistan  1999 745/19987071
#>  2 Afghanistan  2000 2666/20595360
#>  3 Brazil       1999 37737/172006362
#>  4 Brazil       2000 80488/174504898
#>  5 China        1999 212258/1272915272
#>  6 China        2000 213766/1280428583
```

We need to separate `rate` into two variables, cases and population

Can we make datasets tidy?

```
separate(table3, col = rate, into = c("cases", "population"),
         sep = "/")
```

```
#>  # A tibble: 6 x 4
#>    country      year cases  population
#>    <chr>       <int> <chr>  <chr>
#>  1 Afghanistan  1999 745    19987071
#>  2 Afghanistan  2000 2666   20595360
#>  3 Brazil       1999 37737  172006362
#>  4 Brazil       2000 80488  174504898
#>  5 China        1999 212258 1272915272
#>  6 China        2000 213766 1280428583
```

> I've been explicit about naming all the options. R functions can work by position as well, so `separate(table3, rate, c('cases','population'), '/')` would work, but it's not very clear, is it?

Can we make datasets tidy?

```
table2
```

```
#>  # A tibble: 12 x 4
#>     country      year type            count
#>     <chr>       <int> <chr>           <int>
#>   1 Afghanistan  1999 cases             745
#>   2 Afghanistan  1999 population   19987071
#>   3 Afghanistan  2000 cases            2666
#>   4 Afghanistan  2000 population   20595360
#>   5 Brazil       1999 cases           37737
#>   6 Brazil       1999 population  172006362
#>   7 Brazil       2000 cases           80488
#>   8 Brazil       2000 population  174504898
#>   9 China        1999 cases          212258
#>  10 China        1999 population 1272915272
#>  11 China        2000 cases          213766
#>  12 China        2000 population 1280428583
```

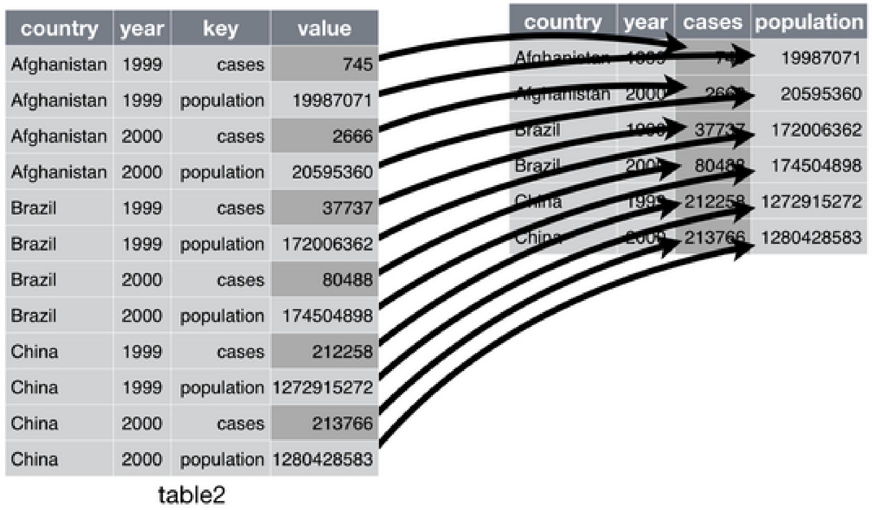
Here there are observations on two variables in successive rows

Can we make datasets tidy?

We need to `spread` these rows out into different columns

```
spread(table2, key = type, value = count)
```

```
#>  # A tibble: 6 x 4
#>    country      year  cases population
#>    <chr>       <int>  <int>      <int>
#>  1 Afghanistan  1999    745   19987071
#>  2 Afghanistan  2000   2666   20595360
#>  3 Brazil       1999  37737  172006362
#>  4 Brazil       2000  80488  174504898
#>  5 China        1999 212258 1272915272
#>  6 China        2000 213766 1280428583
```



Can we make datasets tidy?

```
table4a
```

```
#>  # A tibble: 3 x 3
#>    country     `1999` `2000`
#>  * <chr>        <int>  <int>
#>  1 Afghanistan    745   2666
#>  2 Brazil       37737  80488
#>  3 China       212258 213766
```

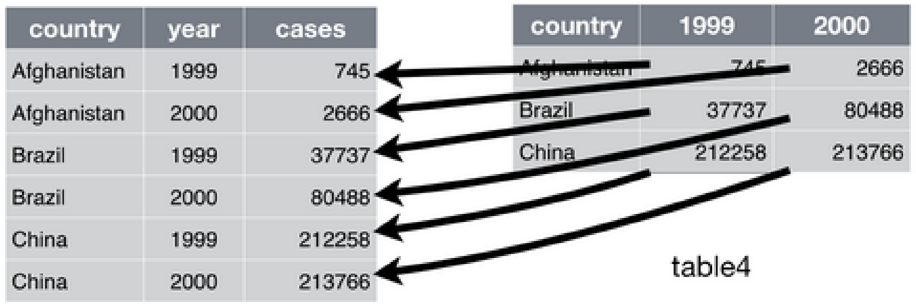
Here, the variable for year is stored as a header, not as data in a cell.

We need to `gather` that data and put it into a column

Can we make datasets tidy?

```
tidyr::gather(table4a, key = year, value = cases, `19
```

```
#>  # A tibble: 6 x 3
#>    country     year   cases
#>    <chr>       <chr>  <int>
#>  1 Afghanistan 1999     745
#>  2 Brazil      1999   37737
#>  3 China       1999  212258
#>  4 Afghanistan 2000    2666
#>  5 Brazil      2000   80488
#>  6 China       2000  213766
```



Making data tidy

Admittedly, `spread` and `gather` are not easy concepts, but we'll practice with them more.

1. `gather` collects multiple columns into 2, and only 2 columns
    - One column represents the data in the column headers
    - One column represents the values in the column
    - All other columns are repeated to keep all the data properly associated
2. `spread` takes two columns and makes them multiple columns
    - The values in one column form the headers to different new columns
    - The values in the other column represent the values in the corresponding cells
    - The other columns are repeated to start with, but reduce repetitions to make all associated data stay together

Progress check

Load the data from this link. You can look the structure by `head(____)` where ___ is what you named the dataset.

What do you think you would need to do to make this data tidy? (Hint: look at the column headers)

What function would you want to use?

Fill in the blanks:

```
gather(______, key = _____, value = ________, ______,______,_______,
              _______,________,_________,__________,__________,____)
```

This is a lot of writing. There's gotta be something simpler

05:00

A friendly way of selecting columns

The tidyverse gives us a nice way of selecting, or not selecting columns

Instead of all the writing, we could simply say

```
pew <- read_csv('data/pew.csv')
tidyr::gather(pew, key = income, value = count, -religion)
```

```
#>  # A tibble: 180 x 3
#>     religion                income count
#>     <chr>                   <chr>  <dbl>
#>   1 Agnostic                <$10k     27
#>   2 Atheist                 <$10k     12
#>   3 Buddhist                <$10k     27
#>   4 Catholic                <$10k    418
#>   5 Don’t know/refused      <$10k     15
#>   6 Evangelical Prot        <$10k    575
#>   7 Hindu                   <$10k      1
#>   8 Historically Black Prot <$10k    228
#>   9 Jehovah's Witness       <$10k     20
#>  10 Jewish                  <$10k     19
#>  # … with 170 more rows
```

`dplyr` verbs in the tidyverse

The `dplyr` package gives us a few verbs for data manipulation

Function	Purpose
select	Select columns based on name or position
mutate	Create or change a column
filter	Extract rows based on some criteria
arrange	Re-order rows based on values of variable(s)
group_by	Split a dataset by unique values of a variable
summarize	Create summary statistics based on columns

select

You can select columns by name or position, of course.

You can also select columns based on some criteria, which are encapsulated in functions.

- starts*with("*__"), ends*with("*__"), contains("__")
- one*of("","","*__")

There are others; see `help(starts_with)`.

Example

Load this file. This contains daily temperature data in 2010 for some location.

```
weather <- rio::import('data/weather.csv')
# weather <- readr::read_csv(here::here('slides','lectures','data','FSI','weather.csv'))
```

```
head(weather, 2)
```

```
#>        id year month element d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13
#> 1 MX17004 2010     1    tmax NA NA NA NA NA NA NA NA NA  NA  NA  NA  NA
#> 2 MX17004 2010     1    tmin NA NA NA NA NA NA NA NA NA  NA  NA  NA  NA
#>   d14 d15 d16 d17 d18 d19 d20 d21 d22 d23 d24 d25 d26 d27 d28 d29  d30 d31
#> 1  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 27.8  NA
#> 2  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 14.5  NA
```

How would you just select the columns with the daily data?

```
select(weather, starts_with("d"))
```

mutate

`mutate` can either transform a column in place or create a new column in a dataset

We'll use the in-built `mpg` dataset for this example, We'll select only the city and highway mileages. To use this selection later, we will need to assign it to a new name

```
mpg1 <- select(mpg, cty, hwy)
```

mutate

We'll change the city and highway mileage to km/l from mpg. This will involve multiplying it by 1.6 and dividing by 3.8

```
mutate(mpg1, cty = cty * 1.6 / 3.8, hwy = hwy * 1.6/3
```

```
#>  # A tibble: 234 x 2
#>       cty   hwy
#>     <dbl> <dbl>
#>   1  7.58  12.2
#>   2  8.84  12.2
#>   3  8.42  13.1
#>   4  8.84  12.6
#>   5  6.74  10.9
#>   6  7.58  10.9
#>   7  7.58  11.4
#>   8  7.58  10.9
#>   9  6.74  10.5
#>  10  8.42  11.8
#>  # … with 224 more rows
```

This is in-place replacement

```
mutate(mpg1, cty1 = cty * 1.6/3.8, hwy1 = hwy * 1.6/3
```

```
#>  # A tibble: 234 x 4
#>       cty   hwy  cty1  hwy1
#>     <int> <int> <dbl> <dbl>
#>   1    18    29  7.58  12.2
#>   2    21    29  8.84  12.2
#>   3    20    31  8.42  13.1
#>   4    21    30  8.84  12.6
#>   5    16    26  6.74  10.9
#>   6    18    26  7.58  10.9
#>   7    18    27  7.58  11.4
#>   8    18    26  7.58  10.9
#>   9    16    25  6.74  10.5
#>  10    20    28  8.42  11.8
#>  # … with 224 more rows
```

This creates new variables

filter

`filter` extracts rows based on criteria

```
filter(mpg, cyl == 4)
```

```
#>  # A tibble: 81 x 11
#>     manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
#>     <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
#>   1 audi         a4      1.8  1999     4 auto… f        18    29 p     comp…
#>   2 audi         a4      1.8  1999     4 manu… f        21    29 p     comp…
#>   3 audi         a4      2    2008     4 manu… f        20    31 p     comp…
#>   4 audi         a4      2    2008     4 auto… f        21    30 p     comp…
#>   5 audi         a4 q…   1.8  1999     4 manu… 4        18    26 p     comp…
#>   6 audi         a4 q…   1.8  1999     4 auto… 4        16    25 p     comp…
#>   7 audi         a4 q…   2    2008     4 manu… 4        20    28 p     comp…
#>   8 audi         a4 q…   2    2008     4 auto… 4        19    27 p     comp…
#>   9 chevrolet    mali…   2.4  1999     4 auto… f        19    27 r     mids…
#>  10 chevrolet    mali…   2.4  2008     4 auto… f        22    30 r     mids…
#>  # … with 71 more rows
```

This extracts only 4 cylinder vehicles

Other choices might be `cyl != 4`, `cyl > 4`, `year == 1999`, `manufacturer=="audi"`

Exercise

We already saw the weather data. It's not tidy. Let's work to make it tidy.