Practical R: Data Munging

Abhijit Dasgupta

BIOF 339

Data munging


stringr
forcats
tidyverse
www.rstudio.com
dplyr
tidyr
readr
purrr
Horst

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

The tidyverse.org site and the R4DS book are the definitive sources for tidyverse information.
The packages are united in a common philosophy of how data analysis should be done.

Tidying data

Tidy data

Tidy datasets are all alike,
but every messy data is messy in its own way

Tidy data

Tidy data is a computer-friendly format based on the following characteristics:

- Each row is one observation
- Each column is one variable
- Each set of observational unit forms a table

All other forms of data can be considered messy data.

Let us count the ways

There are many ways data can be messy. An incomplete list....

- Column headers are values, not variables
- Multiple variables are stored in a single column
- Variables are stored in both rows and columns
- Multiple types of observational units are saved in the same table
- A single observational unit is stored in multiple tables

Ways to have messy (i.e. not tidy) data

1. Column headers contain values

Country	< $10K	$10-20K	$20-50K	$50-100K	> $100K
India	40	25	25	9	1
USA	20	20	20	30	10

Ways to have messy (i.e. not tidy) data

Column headers contain values

Country	Income	Percentage
India	< $10K	40
USA	< $10K	20

This is a case of reshaping or melting

Ways to have messy (i.e. not tidy) data

Multiple variables in one column

Country	Year	M_0-14	F_0-14	M_ 15-60	F_15-60	M_60+	F_60+
UK	2010						
UK	2011						

Separating columns into different variables

Country	Year	Gender	Age	Count

Tidying data

The typical steps are

- Transforming data from wide to tall (`pivot_longer`) and from tall to wide (`pivot_wider`)
- Separating columns into different columns (`separate`)
- Putting columns together into new variables (`unite`)

> The functions `pivot_longer` and `pivot_wider` supercede the older functions `gather` and `spread`, which I have used in previous iterations of this class. However, if you are familiar with `gather` and `spread`, they aren't gone and can still be used in the current tidyr package.

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: Tidy Messy Data
- 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 × 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 × 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 × 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 × 3
  country     `1999` `2000`
  <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

```
table4b # population
```

```
# A tibble: 3 × 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 × 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 = "/",
         convert = TRUE) # convert type if possible
```

```
# A tibble: 6 × 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
```

> 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
```

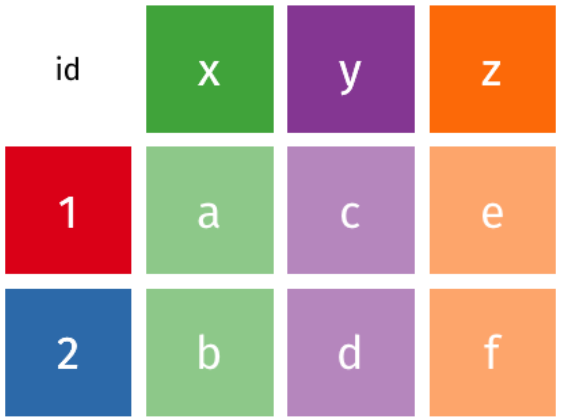
```
# A tibble: 12 × 4
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population    19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population    20595360
 5 Brazil       1999 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. This function is now called `pivot_wider`.

wide




```
pivot_wider(table2, names_from =
```

```
# A tibble: 6 × 4
  country      year  cases popula
  <chr>       <int>  <int>      <
1 Afghanistan  1999    745   1998
2 Afghanistan  2000   2666   2059
3 Brazil       1999  37737  17200
4 Brazil       2000  80488  17450
5 China        1999 212258 127291
6 China        2000 213766 128042
```

Can we make datasets tidy?

```
table4a
```

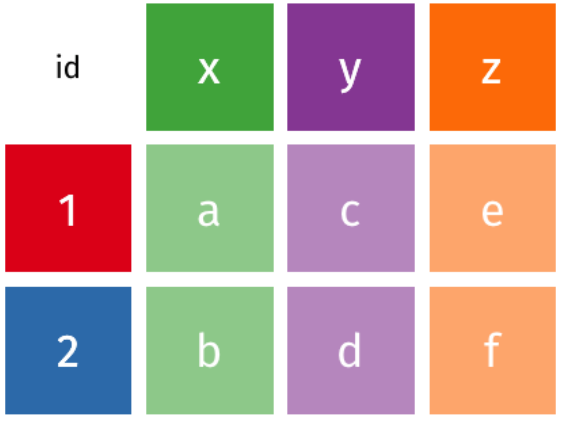
```
# A tibble: 3 × 3
  country     `1999` `2000`
  <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

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. This function is now called `pivot_longer`

Can we make datasets tidy?

wide




```
pivot_longer(table4a, names_to =
    cols = c(`1999`, `2000`))
```

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

Making data tidy

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

1. `pivot_longer` 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. `pivot_wider` 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

Tidying the weather data

```
library(tidyverse)
weather_data <- rio::import('../data/weather.csv')
```

```
       id year month element d1   d2   d3 d4   d5 d6
1 MX17004 2010     1    tmax NA   NA   NA NA   NA NA
2 MX17004 2010     1    tmin NA   NA   NA NA   NA NA
3 MX17004 2010     2    tmax NA 27.3 24.1 NA   NA NA
4 MX17004 2010     2    tmin NA 14.4 14.4 NA   NA NA
5 MX17004 2010     3    tmax NA   NA   NA NA 32.1 NA
6 MX17004 2010     3    tmin NA   NA   NA NA 14.2 NA
  d14 d15  d16 d17 d18 d19 d20 d21 d22  d23 d24 d25
1  NA  NA   NA  NA  NA  NA  NA  NA  NA   NA  NA  NA
2  NA  NA   NA  NA  NA  NA  NA  NA  NA   NA  NA  NA
3  NA  NA   NA  NA  NA  NA  NA  NA  NA 29.9  NA  NA
4  NA  NA   NA  NA  NA  NA  NA  NA  NA 10.7  NA  NA
5  NA  NA 31.1  NA  NA  NA  NA  NA  NA   NA  NA  NA
6  NA  NA 17.6  NA  NA  NA  NA  NA  NA   NA  NA  NA
```

1. Days are in separate columns
2. Temperatures for each day is in two rows, max and min
3. Don't worry about missing values. Just work on getting the shape right

Tidying the weather data

```
weather1 <- pivot_longer(weather_data, names_to='day', values_to='temp',
                         cols = c(-(1:4)))
head(weather1,5)
```

```
# A tibble: 5 × 6
  id       year month element day    temp
  <chr>   <int> <int> <chr>   <chr> <dbl>
1 MX17004  2010     1 tmax    d1       NA
2 MX17004  2010     1 tmax    d2       NA
3 MX17004  2010     1 tmax    d3       NA
4 MX17004  2010     1 tmax    d4       NA
5 MX17004  2010     1 tmax    d5       NA
```

Tidying the weather data

```
weather1 <- pivot_longer(weather_data, names_to='day', values_to='temp',
                         cols = c(-(1:4)))
weather2 <- pivot_wider(weather1, names_from='element', values_from = 'temp')
head(weather2, 5)
```

```
# A tibble: 5 × 6
  id       year month day    tmax  tmin
  <chr>   <int> <int> <chr> <dbl> <dbl>
1 MX17004  2010     1 d1       NA    NA
2 MX17004  2010     1 d2       NA    NA
3 MX17004  2010     1 d3       NA    NA
4 MX17004  2010     1 d4       NA    NA
5 MX17004  2010     1 d5       NA    NA
```

Tidying the weather data

```
weather1 <- pivot_longer(weather_data, names_to='day', values_to='temp',
                         cols = c(-(1:4)))
weather2 <- pivot_wider(weather1, names_from='element', values_from = 'temp')
weather3 <- separate(weather2, col='day', into=c('symbol','day'), sep=1)
head(weather3,5)
```

```
# A tibble: 5 × 7
  id       year month symbol day    tmax  tmin
  <chr>   <int> <int> <chr>  <chr> <dbl> <dbl>
1 MX17004  2010     1 d      1        NA    NA
2 MX17004  2010     1 d      2        NA    NA
3 MX17004  2010     1 d      3        NA    NA
4 MX17004  2010     1 d      4        NA    NA
5 MX17004  2010     1 d      5        NA    NA
```

This gets us into the right shape for the data.

There still is some work to do, but the format is tidy

Data transformation (dplyr)

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, e.g., `select(weather, month)` or `select(weather, 3)`

You can select consecutive columns using `:` notation, e.g. `select(weather, d1:d31)`

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

- `starts_with("___")`, `ends_with("___")`, `contains("____")`
- `one_of("____","_____","______")`
- `everything()`

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

These selection methods work in all tidyverse functions

> Note that for `select` the names of the columns don't need to be quoted. This is called *non-standard evaluation* and is a convenience. However for the criteria-based selectors within `select`, you **do** need to quote the criteria

select

```
weather1 <- select(weather_data, year, month, d1:d31
head(weather1, 20)
```

```
   year month d1   d2   d3   d4   d5 d6   d7   d8 d9
1  2010     1 NA   NA   NA   NA   NA NA   NA   NA NA
2  2010     1 NA   NA   NA   NA   NA NA   NA   NA NA
3  2010     2 NA 27.3 24.1   NA   NA NA   NA   NA NA
4  2010     2 NA 14.4 14.4   NA   NA NA   NA   NA NA
5  2010     3 NA   NA   NA   NA 32.1 NA   NA   NA NA
6  2010     3 NA   NA   NA   NA 14.2 NA   NA   NA NA
7  2010     4 NA   NA   NA   NA   NA NA   NA   NA NA
8  2010     4 NA   NA   NA   NA   NA NA   NA   NA NA
9  2010     5 NA   NA   NA   NA   NA NA   NA   NA NA
10 2010     5 NA   NA   NA   NA   NA NA   NA   NA NA
11 2010     6 NA   NA   NA   NA   NA NA   NA   NA NA
12 2010     6 NA   NA   NA   NA   NA NA   NA   NA NA
13 2010     7 NA   NA 28.6   NA   NA NA   NA   NA NA
14 2010     7 NA   NA 17.5   NA   NA NA   NA   NA NA
15 2010     8 NA   NA   NA   NA 29.6 NA   NA 29.0 NA
16 2010     8 NA   NA   NA   NA 15.8 NA   NA 17.3 NA
17 2010    10 NA   NA   NA   NA 27.0 NA 28.1   NA NA
18 2010    10 NA   NA   NA   NA 14.0 NA 12.9   NA NA
19 2010    11 NA 31.3   NA 27.2 26.3 NA   NA   NA NA
20 2010    11 NA 16.3   NA 12.0  7.9 NA   NA   NA NA
    d15  d16  d17 d18 d19 d20 d21 d22  d23 d24  d25
1    NA   NA   NA  NA  NA  NA  NA  NA   NA  NA   NA
2    NA   NA   NA  NA  NA  NA  NA  NA   NA  NA   NA
3    NA   NA   NA  NA  NA  NA  NA  NA 29.9  NA   NA
4    NA   NA   NA  NA  NA  NA  NA  NA 10.7  NA   NA
5    NA 31.1   NA  NA  NA  NA  NA  NA   NA  NA   NA
6    NA 17.6   NA  NA  NA  NA  NA  NA   NA  NA   NA
7    NA   NA   NA  NA  NA  NA  NA  NA   NA  NA   NA
```

select

```
weather1 <- select(weather_data, starts_with('d'))
head(weather1, 20)
```

```
   d1   d2   d3   d4   d5 d6   d7   d8 d9  d10  d11
1  NA   NA   NA   NA   NA NA   NA   NA NA   NA   NA
2  NA   NA   NA   NA   NA NA   NA   NA NA   NA   NA
3  NA 27.3 24.1   NA   NA NA   NA   NA NA   NA 29.7
4  NA 14.4 14.4   NA   NA NA   NA   NA NA   NA 13.4
5  NA   NA   NA   NA 32.1 NA   NA   NA NA 34.5   NA
6  NA   NA   NA   NA 14.2 NA   NA   NA NA 16.8   NA
7  NA   NA   NA   NA   NA NA   NA   NA NA   NA   NA
8  NA   NA   NA   NA   NA NA   NA   NA NA   NA   NA
9  NA   NA   NA   NA   NA NA   NA   NA NA   NA   NA
10 NA   NA   NA   NA   NA NA   NA   NA NA   NA   NA
11 NA   NA   NA   NA   NA NA   NA   NA NA   NA   NA
12 NA   NA   NA   NA   NA NA   NA   NA NA   NA   NA
13 NA   NA 28.6   NA   NA NA   NA   NA NA   NA   NA
14 NA   NA 17.5   NA   NA NA   NA   NA NA   NA   NA
15 NA   NA   NA   NA 29.6 NA   NA 29.0 NA   NA   NA
16 NA   NA   NA   NA 15.8 NA   NA 17.3 NA   NA   NA
17 NA   NA   NA   NA 27.0 NA 28.1   NA NA   NA   NA
18 NA   NA   NA   NA 14.0 NA 12.9   NA NA   NA   NA
19 NA 31.3   NA 27.2 26.3 NA   NA   NA NA   NA   NA
20 NA 16.3   NA 12.0  7.9 NA   NA   NA NA   NA   NA
    d17 d18 d19 d20 d21 d22  d23 d24  d25  d26  d27
1    NA  NA  NA  NA  NA  NA   NA  NA   NA   NA   NA
2    NA  NA  NA  NA  NA  NA   NA  NA   NA   NA   NA
3    NA  NA  NA  NA  NA  NA 29.9  NA   NA   NA   NA
4    NA  NA  NA  NA  NA  NA 10.7  NA   NA   NA   NA
5    NA  NA  NA  NA  NA  NA   NA  NA   NA   NA   NA
6    NA  NA  NA  NA  NA  NA   NA  NA   NA   NA   NA
7    NA  NA  NA  NA  NA  NA   NA  NA   NA   NA 36.3
```

select

The flexibility of the `select` function, which is also used in other tidyverse functions like `pivot_longer`, and others we'll see presently, is quite powerful.

Suppose you have a large genomic data where the columns are different genes, and suppose that the housekeeping genes all start with "HK". Then, in order to *remove* the housekeeping genes, you could just do

```
new_data <- select(old_data, -starts_with("HK"))
```

Here, the `-` sign means, remove those columns.

Also note that we have to assign the selected dataset to a new (or old) name in order to preserve it in the workspace.

select

I always prefer naming my columns well and using the capabilities of `select` to grab columns.

However, you can use `select` with column numbers. For example, if you wanted to grab the first 4 columns of a dataset, you could do

```
new_data <- select(old_data, 1:4)
```

The notation `1:4` is a short hand for the sequence `1,2,3,4`. Generally, the notation `m:n` means the set of consecutive integers between `m` and `n`.

mutate

`mutate`, as the name suggests, either creates a new column in your data set or transforms an existing column.

```
weather4 <- mutate(weather3,
                   num_day = as.numeric(day))
as_tibble(weather4)
```

```
# A tibble: 341 × 8
   id       year month symbol day    tmax  tmin num_
   <chr>   <int> <int> <chr>  <chr> <dbl> <dbl>   <d
 1 MX17004  2010     1 d      1        NA    NA
 2 MX17004  2010     1 d      2        NA    NA
 3 MX17004  2010     1 d      3        NA    NA
 4 MX17004  2010     1 d      4        NA    NA
 5 MX17004  2010     1 d      5        NA    NA
 6 MX17004  2010     1 d      6        NA    NA
 7 MX17004  2010     1 d      7        NA    NA
 8 MX17004  2010     1 d      8        NA    NA
 9 MX17004  2010     1 d      9        NA    NA
10 MX17004  2010     1 d      10       NA    NA
# … with 331 more rows
```

mutate

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

```
weather4 <- mutate(weather3, day = as.numeric(day))
as_tibble(weather4)
```

```
# A tibble: 341 × 7
   id       year month symbol   day  tmax  tmin
   <chr>   <int> <int> <chr>  <dbl> <dbl> <dbl>
 1 MX17004  2010     1 d          1    NA    NA
 2 MX17004  2010     1 d          2    NA    NA
 3 MX17004  2010     1 d          3    NA    NA
 4 MX17004  2010     1 d          4    NA    NA
 5 MX17004  2010     1 d          5    NA    NA
 6 MX17004  2010     1 d          6    NA    NA
 7 MX17004  2010     1 d          7    NA    NA
 8 MX17004  2010     1 d          8    NA    NA
 9 MX17004  2010     1 d          9    NA    NA
10 MX17004  2010     1 d         10    NA    NA
# … with 331 more rows
```

mutate

`mutate` can also be used to deal with missing values, by replacing them with a value, for example

```
mutate(weather4, tmax = replace_na(tmax, 0))
```

You wouldn't want to do exactly this, of course

```
# A tibble: 341 × 7
   id       year month symbol   day  tmax  tmin
   <chr>   <int> <int> <chr>  <dbl> <dbl> <dbl>
 1 MX17004  2010     1 d          1     0    NA
 2 MX17004  2010     1 d          2     0    NA
 3 MX17004  2010     1 d          3     0    NA
 4 MX17004  2010     1 d          4     0    NA
 5 MX17004  2010     1 d          5     0    NA
 6 MX17004  2010     1 d          6     0    NA
 7 MX17004  2010     1 d          7     0    NA
 8 MX17004  2010     1 d          8     0    NA
 9 MX17004  2010     1 d          9     0    NA
10 MX17004  2010     1 d         10     0    NA
# … with 331 more rows
```

across

dplyr version 1.0 introduced a new verb, `across` to allow functions like `mutate` (and `summarize`, which we shall see in the statistics module) to act on a selection of columns which can be chosen using the same syntax as `select`, or by condition.

```
mutate(mpg,
       cty = cty * 1.6/3.8,
       hwy = hwy * 1.6/3.8)
```

```
mutate(mpg,
       across(c(cty, hwy),
                     function(x) {x * 1.6/3.8}))
```

```
mutate(mpg,
       across(is.character, as.factor)) # select based on condition
```

filter

`filter` extracts rows based on criteria

So if we wanted to just grab January data, we could use

```
january <- filter(weather4, month==1)
head(january)
```

```
# A tibble: 6 × 7
  id          year month symbol   day  tmax  tmin
  <chr>      <int> <int> <chr>  <dbl> <dbl> <dbl>
1 MX17004     2010     1 d          1    NA    NA
2 MX17004     2010     1 d          2    NA    NA
3 MX17004     2010     1 d          3    NA    NA
4 MX17004     2010     1 d          4    NA    NA
5 MX17004     2010     1 d          5    NA    NA
6 MX17004     2010     1 d          6    NA    NA
```

filter

Some comparison operators for filtering

Operator	Meaning
==	Equals
!=	Not equals
> / <	Greater / less than
>= / <=	Greater or equal / Less or equal
!	Not
%in%	In a set

Combining comparisons

Operator	Meaning
&	And
|	Or

filter

Some comparison operators for filtering

Strings: `str_detect(<variable>, "<string>")` or `str_detect(<variable>, "<regex>")`

Regex or regular expression basics:

Expression	Meaning
[a,b,c]	Matches "a", "b" or "c"
[a-z]	Matches letters between "a" and "z"
[^abc]	Matches anything except "a", "b" and "c"
[:alpha:]	letters
[:digit:]	digits
[:alnum:]	letters or numbers
[:punct:]	punctuation

Many more details are available here and a cheatsheet is available here

filter

Let's use the `mpg` dataset from the `ggplot2` package

```
mpg1 <- filter(mpg,
       (year==1999) &
         (class %in% c('minivan','suv')))
select(mpg1, manufacturer, cty, hwy, class, year)
```

```
# A tibble: 35 × 5
   manufacturer   cty   hwy class    year
   <chr>        <int> <int> <chr>   <int>
 1 chevrolet       13    17 suv      1999
 2 chevrolet       11    15 suv      1999
 3 chevrolet       14    17 suv      1999
 4 dodge           18    24 minivan  1999
 5 dodge           17    24 minivan  1999
 6 dodge           16    22 minivan  1999
 7 dodge           16    22 minivan  1999
 8 dodge           15    22 minivan  1999
 9 dodge           15    21 minivan  1999
10 dodge           13    17 suv      1999
# … with 25 more rows
```

filter

A common use of `filter` is to remove rows with missing values from your dataset

```
weather5 <- filter(weather4,
                   !is.na(tmax) & !is.na(tmin))
head(weather5, 20)
```

`is.na` is a *function* that tests whether a value is missing or not.

So `!is.na` is the opposite of that.

```
# A tibble: 20 × 7
   id       year month symbol   day  tmax  tmin
   <chr>   <int> <int> <chr>  <dbl> <dbl> <dbl>
 1 MX17004  2010     1 d         30  27.8  14.5
 2 MX17004  2010     2 d          2  27.3  14.4
 3 MX17004  2010     2 d          3  24.1  14.4
 4 MX17004  2010     2 d         11  29.7  13.4
 5 MX17004  2010     2 d         23  29.9  10.7
 6 MX17004  2010     3 d          5  32.1  14.2
 7 MX17004  2010     3 d         10  34.5  16.8
 8 MX17004  2010     3 d         16  31.1  17.6
 9 MX17004  2010     4 d         27  36.3  16.7
10 MX17004  2010     5 d         27  33.2  18.2
11 MX17004  2010     6 d         17  28    17.5
12 MX17004  2010     6 d         29  30.1  18
13 MX17004  2010     7 d          3  28.6  17.5
14 MX17004  2010     7 d         14  29.9  16.5
15 MX17004  2010     8 d          5  29.6  15.8
16 MX17004  2010     8 d          8  29    17.3
17 MX17004  2010     8 d         13  29.8  16.5
18 MX17004  2010     8 d         23  26.4  15
19 MX17004  2010     8 d         25  29.7  15.6
20 MX17004  2010     8 d         29  28    15.3
```

Important distinction

The `filter` function affects rows of a dataset

The `select` function affects columns of a dataset

slice

You can use `slice` and siblings to subset rows of a data set by index.

- `slice(mpg, 1,2,5)` grabs rows 1, 2 and 5
- `slice_head(mpg)` / `slice_tail(mpg)` grabs first/last row of data set
  - You can specify an argument `n` for the number of rows to grab
  - You can specify an argument `prop` for the proportion of rows to grab
- `slice_sample(mpg, 10)` grabs 10 rows at random, without replacement
- `slice_min(mpg, hwy)` / `slice_max(mpg, hwy)` gives the `n`/`prop` rows with the lowest/highest values of `hwy`.

arrange

`arrange` reorders rows of a data set according to the values of one or more variables

```
arrange(weather5, day)
```

Not quite.

```
# A tibble: 33 × 7
   id       year month symbol   day  tmax  tmin
   <chr>   <int> <int> <chr>  <dbl> <dbl> <dbl>
 1 MX17004  2010    12 d          1  29.9  13.8
 2 MX17004  2010     2 d          2  27.3  14.4
 3 MX17004  2010    11 d          2  31.3  16.3
 4 MX17004  2010     2 d          3  24.1  14.4
 5 MX17004  2010     7 d          3  28.6  17.5
 6 MX17004  2010    11 d          4  27.2  12
 7 MX17004  2010     3 d          5  32.1  14.2
 8 MX17004  2010     8 d          5  29.6  15.8
 9 MX17004  2010    10 d          5  27    14
10 MX17004  2010    11 d          5  26.3   7.9
# … with 23 more rows
```

arrange

```
arrange(weather5, month, day)
```

```
# A tibble: 33 × 7
   id       year month symbol   day  tmax  tmin
   <chr>   <int> <int> <chr>  <dbl> <dbl> <dbl>
 1 MX17004  2010     1 d         30  27.8  14.5
 2 MX17004  2010     2 d          2  27.3  14.4
 3 MX17004  2010     2 d          3  24.1  14.4
 4 MX17004  2010     2 d         11  29.7  13.4
 5 MX17004  2010     2 d         23  29.9  10.7
 6 MX17004  2010     3 d          5  32.1  14.2
 7 MX17004  2010     3 d         10  34.5  16.8
 8 MX17004  2010     3 d         16  31.1  17.6
 9 MX17004  2010     4 d         27  36.3  16.7
10 MX17004  2010     5 d         27  33.2  18.2
# … with 23 more rows
```

arrange

1. I use `arrange` sparingly in my workflow
    - For spiffying up final presentation tables
    - If order is really important
2. Sorting data is one of the most computationally expensive operations you can do
    - It can crash your computer for big data

Cluttering up our workspace

We've done a bit, but lets see all the objects we've created

```
ls()
```

```
 [1] "dat"          "datadir"      "hexes"        "icons_left"
 [5] "imgdir"       "january"      "mpg1"         "start_box"
 [9] "stop_box"     "tidy_pkgs"    "update_header" "weather_data"
[13] "weather1"     "weather2"     "weather3"     "weather4"
[17] "weather5"
```

We see a lot of intermediate datasets we've created, that we aren't going to really use anymore

Workflow pipes in the tidyverse

Intermediate data sets

Recall how we cleaned the weather dataset yesterday

```
weather_data <- rio::import('../data/weather.csv')
weather1 <- pivot_longer(weather_data, names_to='day', values_to = 'temp', cols = starts_with('d'))
weather2 <- pivot_wider(weather1, names_from = element, values_from=temp)
weather3 <- separate(weather2, day, c('symbol','day'), sep = 1, convert=TRUE)
weather4 <- select(weather3, -symbol)
# weather4 <- mutate(weather2, day = readr::parse_number(day))
weather5 <- mutate(weather4,
                   tmax = replace_na(tmax, 0),
                   tmin = replace_na(tmin, 0))
weather6 <- arrange(weather5,year,month,day)
```

This required us to create and keep track of several intermediate datasets

These datasets are essentially temporary datasets which do not hold the final result

What we did is a series of sequential steps to process the data

The `parse_number` function extracts the first number out of a character string. For example, `parse_number('abc254')` outputs 254.

Pipes

Pipes

Pipes are a method in R to create analytic pipelines utilizing tidyverse functions.

The pipe operator (denoted `%>%`, spoken as "then") is what creates the pipes.

You start with a dataset, and then progressively add functions to the pipe. Typically you save the result to a new object.

Each element of the pipe takes as its first argument the results of the previous step, which typically is a data frame.

Pipes are just a different representation of an analytic process that we can do in separate steps anyway.

The keyboard shortcut for the pipe operator in RStudio is `Ctrl/Cmd + Shift + m`

Pipes

Without pipes

```
mpg1 <- mutate(mpg, id = 1:n())
mpg2 <- select(mpg1, id, year, trans, cty, hwy)
mpg_final <- mutate(mpg2,
                    across(c(cty, hwy),
                           function(x) {x * 1.6/3.8}
```

With pipes

```
mpg_final <- mpg %>%
  mutate(id = 1:n()) %>%
  select(id, year, trans, cty, hwy) %>%
  mutate(across(c(cty, hwy),
                function(x){x*1.6/3.8}))
```

The important things to note here are:

1. When using pipes, the results of one operation are automatically entered into the first argument of the next function, so the actual specification omits the first argument
2. If you need the results of one step to go to some other argument of the next function, you can represent that input by `.`, for example, `mpg %>% lm(cty ~ hwy, data = .)` takes the dataset `mpg` and places it in the argument for `data` in the `lm` function.