Getting / cleaning data 2

More with `dplyr`

unite

The `unite` function does the reverse of the `separate` function: it lets you join several columns into a single column. For example, say you have data where year, month, and day are split into different columns:

```
## # A tibble: 4 x 3
##    year month   day
##   <dbl> <dbl> <int>
## 1  2016    10     1
## 2  2016    10     2
## 3  2016    10     3
## 4  2016    10     4
```

unite

You can use `unite` to join these into a single column:

```
date_example %>%
  unite(col = date, year, month, day, sep = "-")
```

```
## # A tibble: 4 x 1
##   date
##   <chr>
## 1 2016-10-1
## 2 2016-10-2
## 3 2016-10-3
## 4 2016-10-4
```

unite

If the columns you want to unite are in a row (and in the right order), you can use the `:` syntax with `unite`:

```
date_example %>%
  unite(col = date, year:day, sep = "-")
```

```
## # A tibble: 4 x 1
##   date
##   <chr>
## 1 2016-10-1
## 2 2016-10-2
## 3 2016-10-3
## 4 2016-10-4
```

Grouping with `mutate` versus `summarize`

So far, we have never used `mutate` with grouping.

You can use `mutate` after grouping– unlike `summarize`, the data will not be collapsed to fewer columns, but the summaries created by `mutate` will be added within each group.

For example, if you wanted to add the mean time by team to the `worldcup` dataset, you could do that with `group_by` and `mutate` (see next slide).

Grouping with `mutate` versus `summarize`

```r
worldcup %>%
  group_by(Position) %>%
  mutate(mean_time = mean(Time)) %>%
  slice(1:2) %>% select(Team:Time, mean_time)
```

```
## # A tibble: 8 x 4
## # Groups:   Position [4]
##   Team        Position    Time mean_time
##   <fct>       <fct>      <int>     <dbl>
## 1 France      Defender     180      242.
## 2 Ghana       Defender     138      242.
## 3 Cameroon    Forward       46      167.
## 4 Uruguay     Forward       72      167.
## 5 Ivory Coast Goalkeeper   270      315.
## 6 Switzerland Goalkeeper   270      315.
## 7 Algeria     Midfielder    16      192.
## 8 Japan       Midfielder   351      192.
```

slice

You can also group by a factor first using `group_by`. Then, when you use `slice`, you will get the first few rows for each level of the group.

```
worldcup %>%
  group_by(Position) %>%
  slice(1:2)
```

```
## # A tibble: 8 x 7
## # Groups:   Position [4]
##   Team        Position    Time Shots Passes Tackles Saves
##   <fct>       <fct>      <int> <int>  <int>   <int> <int>
## 1 France      Defender     180     0     91       6     0
## 2 Ghana       Defender     138     0     51       2     0
## 3 Cameroon    Forward       46     2     16       0     0
## 4 Uruguay     Forward       72     0     15       0     0
## 5 Ivory Coast Goalkeeper   270     0     23       0     8
## 6 Switzerland Goalkeeper   270     0     75       0    11
## 7 Algeria     Midfielder    16     0      6       0     0
```

arrange with group_by

You can also group by a factor before arranging. In this case, all data for the first level of the factor will show up first, in the order given in arrange, then all data from the second level will show up in the specified order, etc.

```r
worldcup %>%
  group_by(Team) %>%
  arrange(desc(Saves)) %>%
  slice(1) %>%
  head(n = 4)
```

```
## # A tibble: 4 x 7
## # Groups:   Team [4]
##   Team      Position    Time Shots Passes Tackles Saves
##   <fct>     <fct>      <int> <int>  <int>   <int> <int>
## 1 Algeria   Goalkeeper   180     0     30       0    12
## 2 Argentina Goalkeeper   450     0     47       0    10
## 3 Australia Goalkeeper   270     0     51       0    13
## 4 Brazil    Goalkeeper   450     0     69       0    10
```

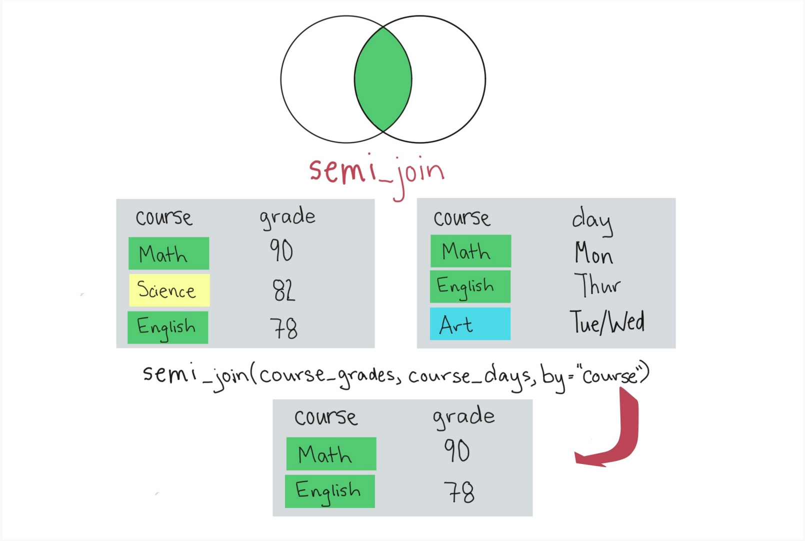
`semi_join` and `anti_join`

There are two more `*_join` functions we'll look at.

These functions allow you to filter one dataframe on only values that **do** have a match in a second dataframe (`semi_join`) or **do not** have a match in a second dataframe (`anti_join`).

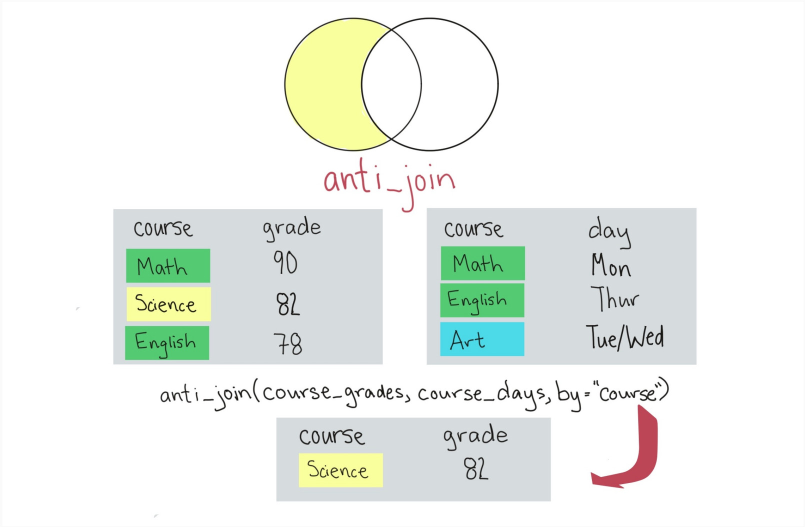
These functions do **not** bring in columns from the second dataset. Instead, they check the second dataset to decide whether or not to keep certain rows in the first dataset.

`semi_join`



The `semi_join` function filters to observations that **do** have a match in a second dataframe.

anti_join



course	grade
Math	90
Science	82
English	78

course	day
Math	Mon
English	Thur
Art	Tue/Wed

```
anti_join(course_grades, course_days, by="course")
```

course	grade
Science	82

The `anti_join` function filters to observations that **do not** have a match in a second dataframe.