Introduction to Data Wrangling II

Summer Institute in Data Science

Rolando J. Acosta

HARVARD
SCHOOL OF PUBLIC HEALTH

June 21, 2022

@RJANunez

What to expected today

- Today we will learn about relational data and how to join tables in R
- Specifically, we will go through different types of join functions
- We will also see simple but useful functions to bind rows and columns
- Lastly, we will go through a bit of set theory and learn useful functions to deal with sets in R

Joins

- Suppose we want to explore relationship between population size for US states and electoral votes in the 2016 presidential election

Joins

- Suppose we want to explore relationship between population size for US states and electoral votes in the 2016 presidential election
- The ***murders*** dataset contains population data for the US states and ***polls_us_election_2016*** has electoral votes data

Joins

- Suppose we want to explore relationship between population size for US states and electoral votes in the 2016 presidential election
- The ***murders*** dataset contains population data for the US states and ***polls_us_election_2016*** has electoral votes data
- Here are samples of the data:

	state	abb	region	population	total
1	Alabama	AL	South	4779736	135
2	Alaska	AK	West	710231	19
3	Arizona	AZ	West	6392017	232
4	Arkansas	AR	South	2915918	93
5	California	CA	West	37253956	1257

murders

	state	startdate	enddate	pollster	grade	samplesize
1	U.S.	2016-11-03	2016-11-06	ABC News/Washington Post	A+	2220
2	U.S.	2016-11-01	2016-11-07	Google Consumer Surveys	B	26574
3	U.S.	2016-11-02	2016-11-06	Ipsos	A-	2195
4	U.S.	2016-11-04	2016-11-07	YouGov	B	3677
5	U.S.	2016-11-03	2016-11-06	Gravis Marketing	B-	16639

polls_us_election_2016

Joins

- The join functions in the ***dplyr*** package make sure to tables are combined so that the corresponding rows are matched

Joins

- The join functions in the ***dplyr*** package make sure to tables are combined so that the corresponding rows are matched
- Does anyone here know SQL? If so, this is going to be very familiar

Joins

- The join functions in the ***dplyr*** package make sure to tables are combined so that the corresponding rows are matched
- Does anyone here know SQL? If so, this is going to be very familiar
- If not, no problem!

Joins

- The join functions in the ***dplyr*** package make sure to tables are combined so that the corresponding rows are matched
- Does anyone here know SQL? If so, this is going to be very familiar.
- If not, no problem!
- The general idea is that tables should joined/matched by one or more columns

Joins

- The join functions in the ***dplyr*** package make sure to tables are combined so that the corresponding rows are matched
- Does anyone here know SQL? If so, this is going to be very familiar.
- If not, no problem!
- The general idea is that tables should joined/matched by one or more columns
- Let's look back at the data:

	state	abb	region	population	total
1	Alabama	AL	South	4779736	135
2	Alaska	AK	West	710231	19
3	Arizona	AZ	West	6392017	232
4	Arkansas	AR	South	2915918	93
5	California	CA	West	37253956	1257

murders

	state	startdate	enddate	pollster	grade	samplesize
1	U.S.	2016-11-03	2016-11-06	ABC News/Washington Post	A+	2220
2	U.S.	2016-11-01	2016-11-07	Google Consumer Surveys	B	26574
3	U.S.	2016-11-02	2016-11-06	Ipsos	A-	2195
4	U.S.	2016-11-04	2016-11-07	YouGov	B	3677
5	U.S.	2016-11-03	2016-11-06	Gravis Marketing	B-	16639

polls_us_election_2016

Joins

- For simplicity of exposition, consider the following two tables

```
tab_1 <- slice(murders, 1:6) %>%
  select(state, population)

tab_2 <- results_us_election_2016 %>%
  filter(state %in% c("Alabama", "Alaska", "Arizona",
                      "California", "Connecticut", "Delaware")) %>%
  select(state, electoral_votes) %>%
  rename(ev = electoral_votes)
```

Joins

- For simplicity of exposition, consider the following two tables

```
tab_1 <- slice(murders, 1:6) %>%
  select(state, population)

tab_2 <- results_us_election_2016 %>%
  filter(state %in% c("Alabama", "Alaska", "Arizona",
                      "California", "Connecticut", "Delaware")) %>%
  select(state, electoral_votes) %>%
  rename(ev = electoral_votes)
```

- What's in `tab_1` and `tab_2`, respectively, any guesses?

Joins

- For simplicity of exposition, consider the following two tables

```
tab_1 <- slice(murders, 1:6) %>%
  select(state, population)

tab_2 <- results_us_election_2016 %>%
  filter(state %in% c("Alabama", "Alaska", "Arizona",
                      "California", "Connecticut", "Delaware")) %>%
  select(state, electoral_votes) %>%
  rename(ev = electoral_votes)
```

- What's in `tab_1` and `tab_2`, respectively, any guesses?
- `tab_1`: We take the first 6 observations in ***murders*** and select the variables *state* and *population*

Joins

- For simplicity of exposition, consider the following two tables

```
tab_1 <- slice(murders, 1:6) %>%
  select(state, population)

tab_2 <- results_us_election_2016 %>%
  filter(state %in% c("Alabama", "Alaska", "Arizona",
                      "California", "Connecticut", "Delaware")) %>%
  select(state, electoral_votes) %>%
  rename(ev = electoral_votes)
```

- What's in `tab_1` and `tab_2`, respectively, any guesses?
- `tab_1`: We take the first 6 observations in ***murders*** and select the variables *state* and *population*
- `tab_2`: We subset the ***results_us_election_2016*** and only consider the observations associated with: Alabama, Alaska, Arizona, California, Connecticut, and Delaware. Then, we select the *state* and *electoral_votes* variables. Finally, we rename the *electoral_votes* variable to *ev*.

Joins

`tab_1`

	state	population
1	Alabama	4779736
2	Alaska	710231
3	Arizona	6392017
4	Arkansas	2915918
5	California	37253956
6	Colorado	5029196

`tab_2`

	state	ev
1	California	55
2	Arizona	11
3	Alabama	9
4	Connecticut	7
5	Alaska	3
6	Delaware	3

`left_join`

- Suppose we want a table like `tab_1` with the electoral votes column from `tab_2`
- We can use `left_join` for this
- Syntax:

```
left_join(first table, second table, by)
```

- `first table`: Table on the left
- `second table`: Table on the right
- `by`: Columns to match observations

left_join

```
left_join(tab_1, tab_2, by = "state")
```

	state	population	ev
1	Alabama	4779736	9
2	Alaska	710231	3
3	Arizona	6392017	11
4	Arkansas	2915918	NA
5	California	37253956	55
6	Colorado	5029196	NA

left_join

```
left_join(tab_1, tab_2, by = "state")
```

	state	population	ev
1	Alabama	4779736	9
2	Alaska	710231	3
3	Arizona	6392017	11
4	Arkansas	2915918	*NA*
5	California	37253956	55
6	Colorado	5029196	*NA*

- Notice the NA values in the ***ev*** column. Any thoughts on this?

left_join

```
left_join(tab_1, tab_2, by = "state")
```

	state	population	ev
1	Alabama	4779736	9
2	Alaska	710231	3
3	Arizona	6392017	11
4	Arkansas	2915918	*NA*
5	California	37253956	55
6	Colorado	5029196	*NA*

- Notice the NA values in the ***ev*** column. Any thoughts on this?
- The reason is that Arkansas and Colorado are not in `tab_2`

left_join

```
left_join(tab_1, tab_2, by = "state")
```

	state	population	ev
1	Alabama	4779736	9
2	Alaska	710231	3
3	Arizona	6392017	11
4	Arkansas	2915918	NA
5	California	37253956	55
6	Colorado	5029196	NA

- Notice the NA values in the ***ev*** column. Any thoughts on this?
- The reason is that Arkansas and Colorado are not in `tab_2`
- Let us explore this example a bit further

left_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

left_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

```
left_join(tab_1, tab_2, by = “state”)
```

left_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

```
left_join(tab_1, tab_2, by = “state”)
```

state	population	ev
Alabama	4,779,736	55
Alaska	710,231	11
Arizona	6,392,017	9
Arkansas	2,915,918	NA
California	37,253,956	3
Colorado	5,029,196	NA

`right_join`

- Suppose now that we want to a table like `tab_2` with the population column from `tab_1`
- We can use right_`join` for this
- Syntax:

```
right_join(first table, second table, by)
```

- `first table`: Table on the left
- `second table`: Table on the right
- `by`: Columns to match observations

right_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

right_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

```
right_join(tab_1, tab_2, by = “state”)
```

right_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

```
right_join(tab_1, tab_2, by = “state”)
```

state	population	ev
California	37,253,956	55
Arizona	6,392,017	11
Alabama	4,779,736	9
Connecticut	NA	7
Alaska	710,231	3
Delaware	NA	3

`inner_join`

- Notice the `NAs` produced when we used `left_join` or `right_join`

`inner_join`

- Notice the `NAs` produced when we used `left_join` or `right_join`
- This is because some observations in `tab_1` are not in `tab_2` and viceversa

`inner_join`

- Notice the `NAs` produced when we used `left_join` or `right_join`
- This is because some observations in `tab_1` are not in `tab_2` and viceversa
- If you want to keep only observations that appear in both tables, you can use `inner_join`

`inner_join`

- Notice the NAs produced when we used `left_join` or `right_join`
- This is because some observations in `tab_1` are not in `tab_2` and viceversa
- If you want to keep only observations that appear in both tables, you can use `inner_join`
- Syntax:

```
inner_join(first table, second table, by)
```

- `first table`: Table on the left
- `second table`: Table on the right
- `by`: Columns to match observations

inner_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

inner_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

```
inner_join(tab_1, tab_2, by = “state”)
```

state	population	ev
California	37,253,956	55
Arizona	6,392,017	11
Alabama	4,779,736	9
Alaska	710,231	3

`full_join`

- If, contrary to `inner_join`, you want to keep all the rows in both tables and fill the missing values with `NA`s, you can use `full_join`

full_join

- If, contrary to `inner_join`, you want to keep all the rows in both tables and fill the missing values with `NA`s, you can use `full_join`
- Syntax:

```
full_join(first table, second table, by)
```

- `first table`: Table on the left
- `second table`: Table on the right
- `by`: Columns to match observations

full_join

tab_1

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
Arkansas	2,915,918
California	37,253,956
Colorado	5,029,196

Direction of the join

tab_2

state	ev
California	55
Arizona	11
Alabama	9
Connecticut	7
Alaska	3
Delaware	3

full_join

```
full_join(tab_1, tab_2, by = "state")
```

state	population	ev
Alabama	4,779,736	9
Alaska	710,231	3
Arizona	6,392,017	11
Arkansas	2,915,918	NA
California	37,253,956	55
Colorado	5,029,196	NA
Connecticut	NA	7
Delaware	NA	3

Filtering Joins

- Sometimes, instead of joining two tables, we want to know which observations in one table are present or not in another.

Filtering Joins

- Sometimes, instead of joining two tables, we want to know which observations in one table are present or not in another.
- For this we can use `semi_join` and `anti_join`

Filtering Joins

- Sometimes, instead of joining two tables, we want to know which observations in one table are present or not in another.
- For this we can use `semi_join` and `anti_join`
- `semi_join` let us keep the observations from the first table that also appear in the second

Filtering Joins

- Sometimes, instead of joining two tables, we want to know which observations in one table are present or not in another.
- For this we can use `semi_join` and `anti_join`
- `semi_join` let us keep the observations from the first table that also appear in the second
- `anti_join` let us keep the observations from the first table that does not appear in the second

Filtering Joins

- Sometimes, instead of joining two tables, we want to know which observations in one table are present or not in another.
- For this we can use `semi_join` and `anti_join`
- `semi_join` let us keep the observations from the first table that also appear in the second
- `anti_join` let us keep the observations from the first table that does not appear in the second
- The syntax for both functions is the same the ones before

Filtering Joins

- Sometimes, instead of joining two tables, we want to know which observations in one table are present or not in another.
- For this we can use `semi_join` and `anti_join`
- `semi_join` let us keep the observations from the first table that also appear in the second
- `anti_join` let us keep the observations from the first table that does not appear in the second
- The syntax for both functions is the same the ones before
- Let us see an example

Filtering Joins

```
semi_join(tab_1, tab_2, by = “state”)
```

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
California	37,253,956

Filtering Joins

```
semi_join(tab_1, tab_2, by = “state”)
```

state	population
Alabama	4,779,736
Alaska	710,231
Arizona	6,392,017
California	37,253,956

```
anti_join(tab_1, tab_2, by = “state”)
```

state	population
Arkansas	2,915,918
Delaware	5,029,196

Summary

Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::left_join(a, b, by = "x1")

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::right_join(a, b, by = "x1")

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::inner_join(a, b, by = "x1")

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::full_join(a, b, by = "x1")

Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::semi_join(a, b, by = "x1")

All rows in a that have a match in b.

x1	x2
C	3

dplyr::anti_join(a, b, by = "x1")

All rows in a that do not have a match in b.

https://github.com/rstudio/cheatsheets

10 minute break

Binding rows and columns

- We just went through an extensive list of join functions

Binding rows and columns

- We just went through an extensive list of join functions
- All of these are characterized by a set of columns that are used to match the tables of interest

Binding rows and columns

- We just went through an extensive list of join functions
- All of these are characterized by a set of columns that are used to match the tables of interest
- Another way in which datasets are combined is by *binding* them

Binding rows and columns

- We just went through an extensive list of join functions
- All of these are characterized by a set of columns that are used to match the tables of interest
- Another way in which datasets are combined is by *binding* them
- For example, the functions `bind_cols` and `bind_rows` bind two objects as columns and rows, respectively

Binding rows and columns

- We just went through an extensive list of join functions
- All of these are characterized by a set of columns that are used to match the tables of interest
- Another way in which datasets are combined is by *binding* them
- For example, the functions `bind_cols` and `bind_rows` bind two objects as columns and rows, respectively
- Let us see an example

Binding rows and columns

- Consider the ***starwars*** dataset that is available in the ***dplyr*** package

```
head(starwars)
```

name	height	mass	hair_color	skin_color	eye_color	birth_year	sex	gender	homeworld	species
Luke Skywalker	172	77	blond	fair	blue	19.0	male	masculine	Tatooine	Human
C-3PO	167	75	*NA*	gold	yellow	112.0	none	masculine	Tatooine	Droid
R2-D2	96	32	*NA*	white, blue	red	33.0	none	masculine	Naboo	Droid
Darth Vader	202	136	none	white	yellow	41.9	male	masculine	Tatooine	Human
Leia Organa	150	49	brown	light	brown	19.0	female	feminine	Alderaan	Human
Owen Lars	178	120	brown, grey	light	blue	52.0	male	masculine	Tatooine	Human

Binding rows and columns

- Consider the ***starwars*** dataset that is available in the ***dplyr*** package

```
head(starwars)
```

name	height	mass	hair_color	skin_color	eye_color	birth_year	sex	gender	homeworld	species
Luke Skywalker	172	77	blond	fair	blue	19.0	male	masculine	Tatooine	Human
C-3PO	167	75	*NA*	gold	yellow	112.0	none	masculine	Tatooine	Droid
R2-D2	96	32	*NA*	white, blue	red	33.0	none	masculine	Naboo	Droid
Darth Vader	202	136	none	white	yellow	41.9	male	masculine	Tatooine	Human
Leia Organa	150	49	brown	light	brown	19.0	female	feminine	Alderaan	Human
Owen Lars	178	120	brown, grey	light	blue	52.0	male	masculine	Tatooine	Human

- For ease of exposition, let us consider only a few observations and variables

```
one <- starwars[1:3, 1:5]
two <- starwars[4:6, 1:5]
```

Binding rows and columns

`one`

	name	height	mass	hair_color	skin_color
1	Luke Skywalker	172	77	blond	fair
2	C-3PO	167	75	*NA*	gold
3	R2-D2	96	32	*NA*	white, blue

`two`

	name	height	mass	hair_color	skin_color
1	Darth Vader	202	136	none	white
2	Leia Organa	150	49	brown	light
3	Owen Lars	178	120	brown, grey	light

Binding rows and columns

one

	name	height	mass	hair_color	skin_color
1	Luke Skywalker	172	77	blond	fair
2	C-3PO	167	75	NA	gold
3	R2-D2	96	32	NA	white, blue

two

	name	height	mass	hair_color	skin_color
1	Darth Vader	202	136	none	white
2	Leia Organa	150	49	brown	light
3	Owen Lars	178	120	brown, grey	light

- Binding by rows:

```
bind_rows(one, two)
```

	name	height	mass	hair_color	skin_color
1	Luke Skywalker	172	77	blond	fair
2	C-3PO	167	75	NA	gold
3	R2-D2	96	32	NA	white, blue
4	Darth Vader	202	136	none	white
5	Leia Organa	150	49	brown	light
6	Owen Lars	178	120	brown, grey	light

Binding rows and columns

one

	name	height	mass	hair_color	skin_color
1	Luke Skywalker	172	77	blond	fair
2	C-3PO	167	75	*NA*	gold
3	R2-D2	96	32	*NA*	white, blue

two

	name	height	mass	hair_color	skin_color
1	Darth Vader	202	136	none	white
2	Leia Organa	150	49	brown	light
3	Owen Lars	178	120	brown, grey	light

- Binding by columns:

```
bind_cols(one, two)
```

name...1	height...2	mass...3	hair_color...4	skin_color...5	name...6	height...7	mass...8	hair_color...9	skin_color...10
Luke Skywalker	172	77	blond	fair	Darth Vader	202	136	none	white
C-3PO	167	75	*NA*	gold	Leia Organa	150	49	brown	light
R2-D2	96	32	*NA*	white, blue	Owen Lars	178	120	brown, grey	light

Set Operators

- Lastly, another set of commands useful for combining datasets are *set operators*

Set Operators

- Lastly, another set of commands useful for combining datasets are *set operators*
- Specifically, we will look at the functions: `intersect`, `union`, `setdiff`, and `setequal`

Set Operators

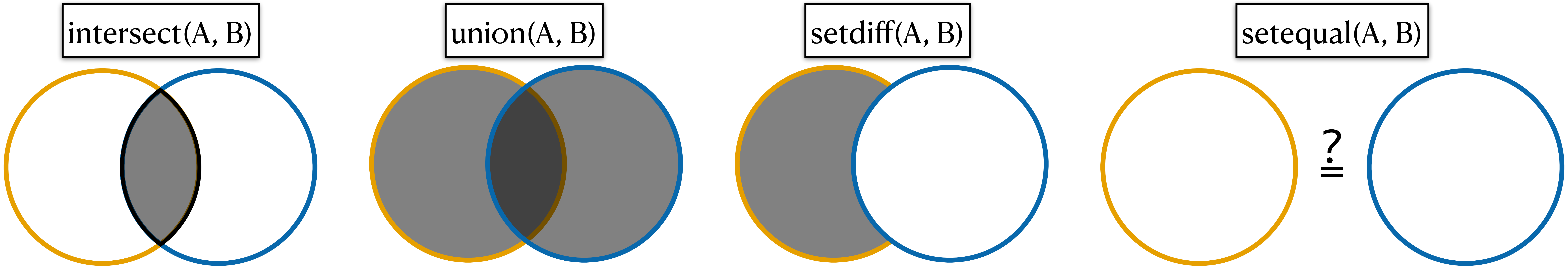
- Lastly, another set of commands useful for combining datasets are *set operators*
- Specifically, we will look at the functions: `intersect`, `union`, `setdiff`, and `setequal`
- In mathematics, a set is a collection of distinct elements

Set Operators

- Lastly, another set of commands useful for combining datasets are *set operators*
- Specifically, we will look at the functions: `intersect`, `union`, `setdiff`, and `setequal`
- In mathematics, a set is a collection of distinct elements
- Two sets are equal if and only if they have precisely the same elements

Set Operators

- Lastly, another set of commands useful for combining datasets are *set operators*
- Specifically, we will look at the functions: `intersect`, `union`, `setdiff`, and `setequal`
- In mathematics, a set is a collection of distinct elements
- Two sets are equal if and only if they have precisely the same elements
- Consider the following schematics where `A` and `B` are two vectors in R



Set Operators

- Here is a concrete example:

```
a <- 1:5
b <- 4:9

union(a, b)
[1] 1 2 3 4 5 6 7 8 9

intersect(a, b)
[1] 4 5

setdiff(a, b)
[1] 1 2 3

setequal(a, b)
[1] FALSE
```

References

1. Introduction to Data Science: Data analysis and prediction algorithms with R by Rafael A. Irizarry, Chapter 22. https://rafalab.github.io/dsbook/
2. R for Data Science by Grolemund & Wickham, Chapter 13. https://r4ds.had.co.nz/index.html

Referencias en español:

1. Introducción a la Ciencia de Datos: Análisis de datos y algoritmos de predicción con R por Rafael A. Irizarry, Capítulo 22. https://rafalab.github.io/dslibro/
2. R para Ciencia de Datos por Grolemund & Wickham, Capítulo 13. https://es.r4ds.hadley.nz

Your turn!

[Click here for the class website]