Data Table Joins

Code

Instructor - Soumya Mukherjee

Content Credit- Dr. Matthew Beckman and Olivia Beck

July 18, 2023

Agenda

- what do “join” operations do and why should we care?
- Overview of join operations with examples

Sidenote

- Sometimes you might run into an error that looks similar to the following error message

```
Error in exists(cacheKey, where = .rs.WorkingDataEnv, inherits = FALSE)
```

- If you encounter such an error, Just go to Session >> New Session (Be sure to save your progress first) and open a new session. The error should not come up anymore.

Combining data from different sources

Glyph-ready data often combines data from different sources.

- Perhaps they come from different experiments or institutions.
- Often they were collected with different objectives than yours.
- Perhaps they are completely different types of data.

Example: Medicare data (All of these are in dcData package)

How might you combine data from several of these tables to answer an interesting research question?

- `MedicareProviders` : Name and location
- `DirectRecoveryGroups` : Descriptions of standardized medical procedures

- `MedicareCharges` : Charges and payments for different DRGs by different providers
- `ZipDemography` : Population and age structure in each ZIP code.
- A glimpse (pun intended) into the dataframes:

Hide

```
glimpse(MedicareProviders)
```

```
Rows: 3,337
Columns: 7
$ idProvider      <int> 10001, 10005, 10006, 10007, 10008,…
$ nameProvider    <chr> "SOUTHEAST ALABAMA MEDICAL CENTER"…
$ addressProvider <chr> "1108 ROSS CLARK CIRCLE", "2505 U …
$ cityProvider    <fct> DOTHAN, BOAZ, FLORENCE, OPP, LUVER…
$ stateProvider   <fct> AL, AL, AL, AL, AL, AL, AL, AL, AL…
$ zipProvider     <int> 36301, 35957, 35631, 36467, 36049,…
$ referralRegion  <fct> AL - Dothan, AL - Birmingham, AL -…
```

Hide

```
glimpse(DirectRecoveryGroups)
```

```
Rows: 100
Columns: 2
$ drg           <chr> "039", "057", "064", "065", "066", "…
$ drgDefinition <chr> " EXTRACRANIAL PROCEDURES W/O CC/MCC…
```

Hide

```
glimpse(MedicareCharges)
```

```
Rows: 163,065
Columns: 5
$ drg             <chr> "039", "057", "064", "065", "066",…
$ idProvider      <int> 10001, 10001, 10001, 10001, 10001,…
$ totalDischarges <int> 91, 38, 84, 169, 33, 37, 13, 27, 3…
$ aveCharges      <dbl> 32963.08, 20312.79, 38820.39, 2734…
$ avePayments     <dbl> 5777.242, 4894.763, 10260.214, 654…
```

Hide

```
glimpse(ZipDemography)
```

```
Rows: 42,741
Columns: 44
$ Totalpopulation                                        <dbl> …
$ Male                                                   <dbl> …
$ Female                                                 <dbl> …
$ MedianAge                                              <dbl> …
$ Under5years                                            <dbl> …
$ X18yearsandover                                        <dbl> …
$ X65yearsandover                                        <dbl> …
$ Onerace                                                <dbl> …
$ White                                                  <dbl> …
$ BlackorAfricanAmerican                                 <dbl> …
$ AmericanIndianandAlaskaNative                          <dbl> …
$ Asian                                                  <dbl> …
$ NativeHawaiianandOtherPacificIslander                  <dbl> …
$ Someotherrace                                          <dbl> …
$ Twoormoreraces                                         <dbl> …
$ HispanicorLatinoofanyrace                              <dbl> …
$ AverageHouseholdSize                                   <dbl> …
$ Averagefamilysize                                      <dbl> …
$ Totalhousingunits                                      <dbl> …
$ Occupiedhousingunits                                   <dbl> …
$ Owneroccupiedhousingunits                              <dbl> …
$ Renteroccupiedhousingunits                             <dbl> …
$ Vacanthousingunits                                     <dbl> …
$ Population25yearsandover                               <dbl> …
$ Highschoolgraduateorhigher                             <dbl> …
$ Bachelorsdegreeorhigher                                <dbl> …
$ Civilianveterans                                       <dbl> …
$ Disabilitystatuspopulation21to64years                  <dbl> …
$ Foreignborn                                            <dbl> …
$ Nowmarriedpopulation15yearsandover                     <dbl> …
$ SpeakalanguageotherthanEnglishathome5yearsandover      <dbl> …
$ Inlaborforcepopulation16yearsandover                   <dbl> …
$ Meantraveltimetoworkinminutespopulation16yearsandolder <dbl> …
$ Medianhouseholdincomedollars                           <dbl> …
$ Medianfamilyincomedollars                              <dbl> …
$ Percapitaincomedollars                                 <dbl> …
```

```
$ Familiesbelowpovertylevel                 <dbl> …
$ Individualsbelowpovertylevel              <dbl> …
$ Singlefamilyowneroccupiedhomes            <dbl> …
$ Medianvaluedollars                        <dbl> …
$ Medianofselectedmonthlyownercosts         <dbl> …
$ WithaMortgage                             <dbl> …
$ Notmortgaged                              <dbl> …
$ ZIP                                       <chr> …
```

Relational databases

Storing data in separate tables can be beneficial even when the data are coming from the same source:

- Organizations can (and do) restrict access to tables with sensitive information, while permitting wider access with non-sensitive data tables
  - HIPPA & patient identifiers in healthcare industry
  - FERPA & student records at educational institutions
- There is no “one size fits all” glyph-ready format. Often the kinds of analysis that will be done are not specifically anticipated when data are collected.
- Glyph-ready data often contains **redundancies**.

Strategy: Join related tables *as needed* rather than smash all available data into one big table.

Example of a redundency in Glyph ready data:

Hide

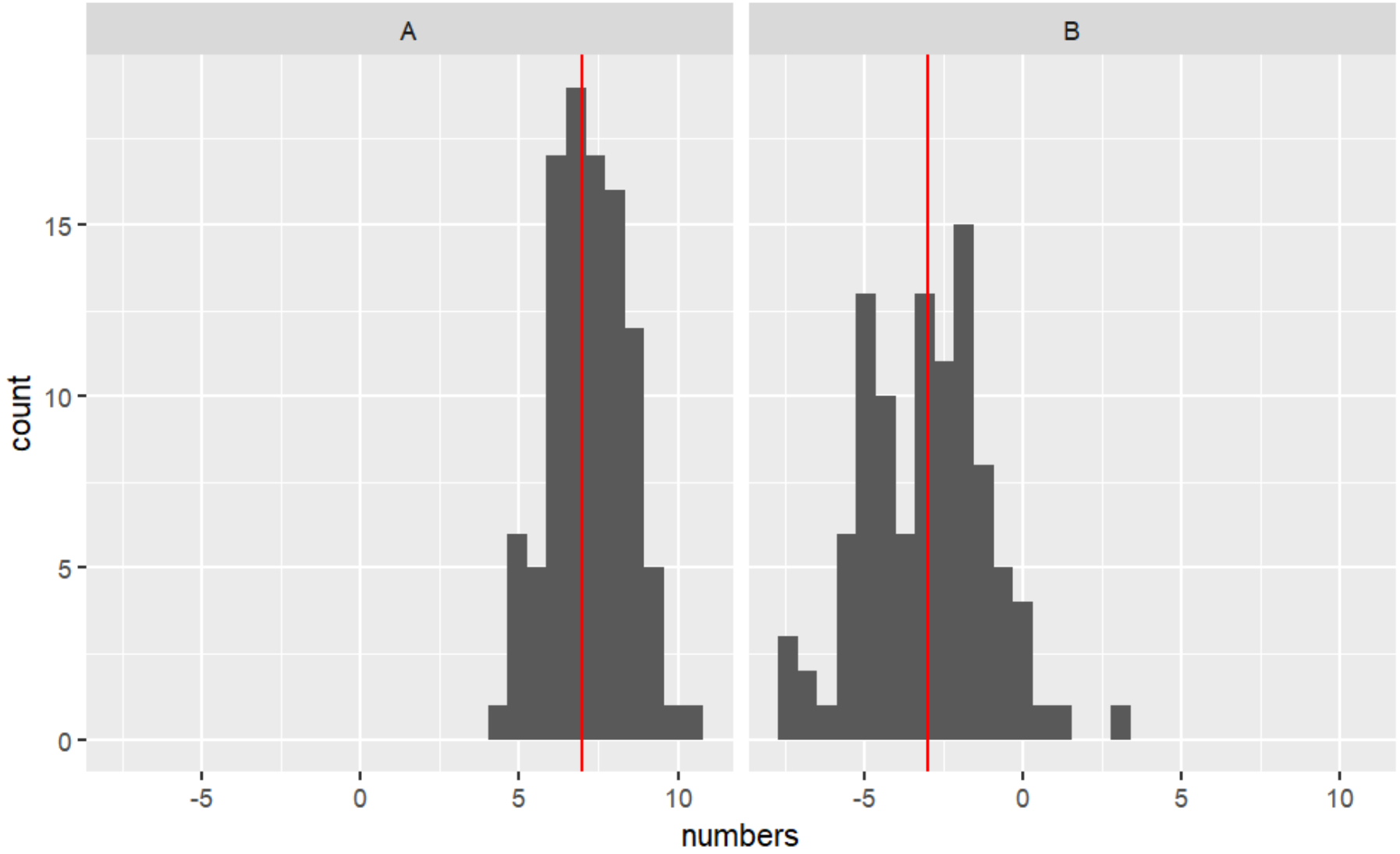
```
dat.A <- rnorm(n = 100, mean = 7, sd = 1)
dat.B <- rnorm(n = 100, mean = -3, sd = 2)

GlyphReady <- data.frame(letters = c(rep("A", 100), rep("B", 100)),
                  numbers = c(dat.A, dat.B),
                  center = c(rep(7, 100), rep (-3, 100)))

GlyphReady %>%
  sample_n(10)
```

letters <chr>	numbers <dbl>	center <dbl>
A	8.078518	7
A	6.253608	7
A	7.415362	7
A	7.410704	7
A	6.833520	7
B	-3.041263	-3
B	-2.112497	-3
B	-4.691075	-3
B	-2.827370	-3
A	8.488075	7

1-10 of 10 rows

Hide

```
GlyphReady %>%
  ggplot(aes(x = numbers)) +
  geom_histogram()+
  geom_vline(aes(xintercept = center), col = "red") +
  facet_wrap( ~ letters)
```



This glyph ready table is not an efficient use of storage. The column `center` stores redundant information. It would be more efficient to store 2 tables like this:

Hide

```
dat1 <-  data.frame(letters = c(rep("A", 100), rep("B", 100)),
                    numbers = c(dat.A, dat.B))
dat2 <- data.frame(letters = c("A", "B"),
                     center = c(7, -3))

dat1 %>%
  sample_n(10)
```

letters <chr>	numbers <dbl>
B	-4.41865559
B	-5.43879571
B	-2.08498857
A	9.26947114
A	7.97996816
A	7.04690029
B	0.05071876
A	10.18955824
A	5.46363565
B	-5.17593410

1-10 of 10 rows

Hide

```
dat2
```

letters <chr>	center <dbl>
A	7
B	-3

2 rows

So how do we go from `dat1` and `dat2` to `Glyphform` ? Joins!

Example: Average class size

Goal: Figure out the average class size *seen* by each student (…think for a moment about how that's different from *average class size*)

Hide

```
# read data file from local working directory
Grades <- read_csv("grades.csv")
```

```
Rows: 5902 Columns: 3── Column specification ────────────────────────────────
Delimiter: ","
chr (3): studentID, grade, lionpathCourseID
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

studentID <chr>	grade <chr>	lionpathCourseID <chr>
S31836	A	session2265
S32427	A-	session2532
S31668	A	session2136
S31923	B	session2641

4 rows

Hide

```
# read data from my local working directory
Courses <- read_csv("coursesUpdated.csv")
```

```
Rows: 1712 Columns: 6── Column specification ────────────────────────────────────
Delimiter: ","
chr (4): lionpathCourseID, dept, sem, instructorID
dbl (2): level, enrollment
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

lionpathCourseID <chr>	dept <chr>	level <dbl>	sem <chr>	enrollment <dbl>	instructorID <chr>
session3366	VBSC	200	FA2004	29	inst362
session3095	CAS	200	SP2004	18	inst142
session2533	PHYS	200	FA2002	63	inst177

3 rows

Key:

- `studentID` : unique student id–e.g., 9-XXXX-XXXX at Penn State
- `grade` : letter grade earned in a course
- `lionpathCourseID` : unique identifier for each course section
- `dept` : academic department of course
- `level` : academic level of course
- `sem` : semester of course offering
- `enrollment` : students enrolled
- `instructorID` : unique instructor id

Basic joins

x

A	B	C
a	t	1
b	u	2
c	v	3

+

y

A	B	D
a	t	3
b	u	2
d	w	1

Left & Right Tables (RStudio Data Transformation Cheat Sheet)
A *join* is a data verb that combines two tables called the **left** and the **right** tables.

- Joins establish a correspondence — i.e. match — between each case in the left table and zero or more cases in the right table.
- Some common joins we'll discuss simply differ in
    - how multiple matches should be handled
    - missing matches should be handled

Example: Average class size

Goal: Figure out the average class size seen by each student.

- `enrollment` comes from `Courses` table.
- Student (`studentID`) comes from `Grades` .
- `lionpathCourseID` is in both tables.

Hide

```
set.seed(101)  # this just makes my "random" sample the same each time

Grades %>%
  left_join(Courses) %>%
  sample_n(size = 4)
```

```
Joining with `by = join_by(lionpathCourseID)`Warning: Detected an unexpected many-to-many relationship between `x` and `y`.
```

studentID <chr>	grade <chr>	lionpathCourseID <chr>	dept <chr>	level <dbl>	sem <chr>	enrollment <dbl>	instructorID <chr>
S31827	B+	session2256	LARCH	100	FA2001	18	inst250
S32403	B	session2174	FIN	100	SP2002	50	inst190
S31659	B+	session3477	WFS	200	FA2004	39	inst354
S31911	A	session3497	DS	300	FA2004	38	inst408

4 rows

Once `Courses` and `Grades` are joined, it's straightforward to find the average enrollment seen by each student.

Hide

```
AveClassEachStudent <-
  Grades %>%
  left_join(Courses) %>%
  group_by(studentID) %>%
  summarise(ave_enroll_seen = mean(enrollment, na.rm = TRUE))
```

```
Joining with `by = join_by(lionpathCourseID)`Warning: Detected an unexpected many-to-many relationship between `x` and `y`.
```

studentID <chr>	ave_enroll_seen <dbl>
S32508	40.44444
S32235	35.30769
S32133	29.20000
S32127	69.66667
S31920	37.12500
S32247	42.33333

6 rows

Compared with average course enrollment:

Hide

```
# average enrollment
Courses %>%
  summarise(avg_course_enroll = mean(enrollment))
```

avg_course_enroll <dbl>
33.68925

1 row

Establishing a match between cases

A match between a case in the *left* table and a case in the *right* table is made based on the values in pairs of corresponding variables.

- **You** specify which pairs to use.
- A pair is a variable from the left table and a variable from the right table.
- Fine to specify several variables for matching
    - e.g., `by = c("A" = "var1", "B" = "var2")`
    - matches `A` & `B` from left table to `var1` and `var2` from right table
- Cases must have *exactly equal* values in the left variable and right variable for a match to be made.
    - A & B and A & B are a match
    - A & B and A & C are not a match
    - A & B and B & A are not a match
- If you **don't** specify the variables directly
    - The default value of `by =` is all variables with the same names in both tables
    - This is **not reliable** in general unless you've checked

Example:

```
Warning: Detected an unexpected many-to-many relationship between `x` and `y`.
```

studentID <chr>	grade <chr>	lionpathCourseID <chr>	dept <chr>	level <dbl>	sem <chr>	enrollment <dbl>	instructorID <chr>
S31185	D+	session1784	MATH	100	FA1991	33	inst265
S31185	B+	session1785	KINES	100	FA1991	78	inst458
S31185	A-	session1791	JST	100	FA1993	33	inst223
S31185	B+	session1792	JST	300	FA1993	30	inst235

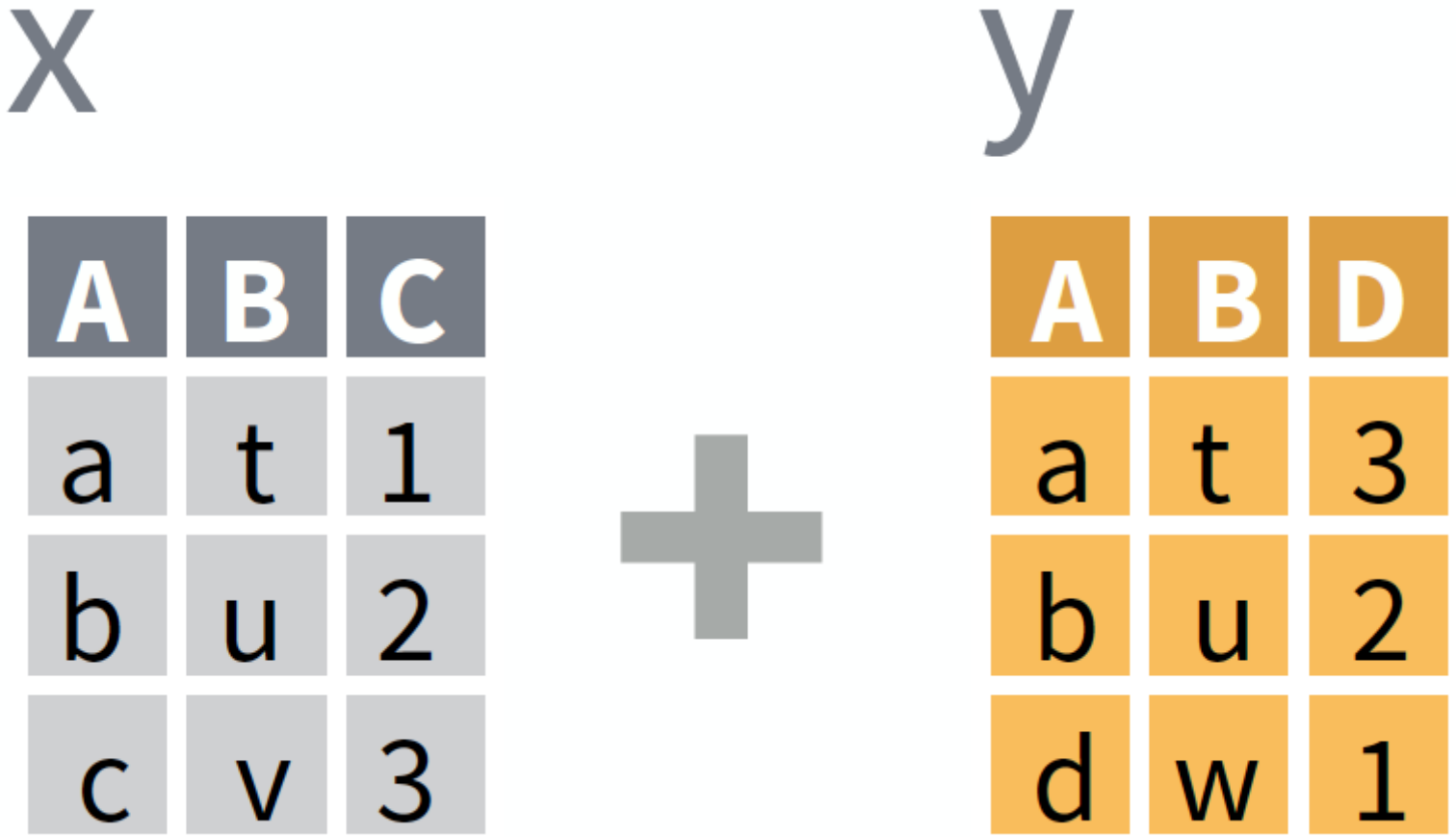
4 rows

Basic join types

Note: Images adapted from RStudio Data Transformation Cheat Sheet https://rstudio.com/resources/cheatsheets/ (https://rstudio.com/resources/cheatsheets/)

- What to do when there is **no match** between a left case and any right case?
- What to do when there are **multiple matching cases** in the right table for a case in the left table?

Different kinds of join have different answers to these questions.



Left Table (X) & Right Table (Y) for illustration

Basic join types

Note: Images adapted from RStudio Data Transformation Cheat Sheet https://rstudio.com/resources/cheatsheets/ (https://rstudio.com/resources/cheatsheets/)

x

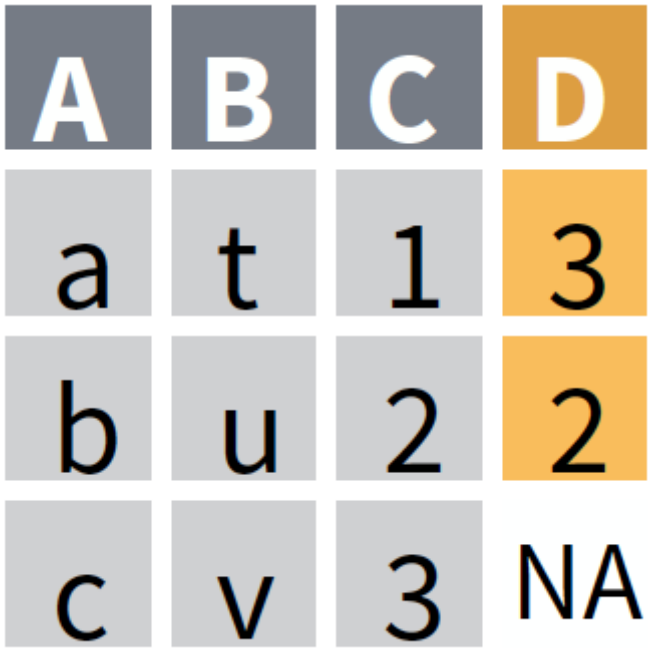
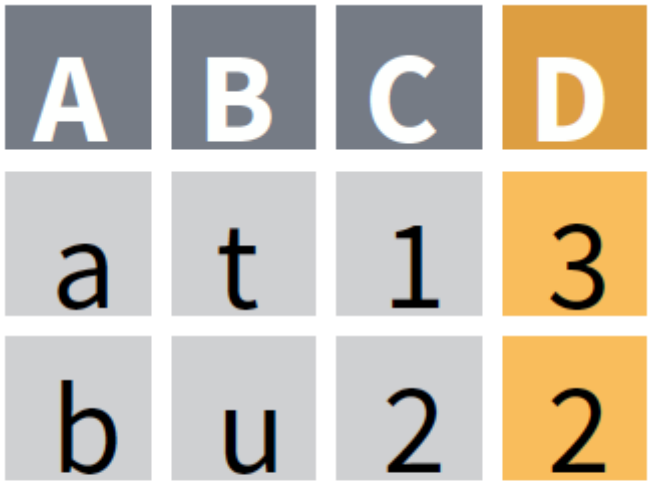
A	B	C
a	t	1
b	u	2
c	v	3

+

y

A	B	D
a	t	3
b	u	2
d	w	1

- `left_join()` : joins matching rows from the *right* table to the *left* table
  - i.e. we keep ALL information from the left table, and add information from the right table
- `inner_join()` : only retain rows for which a match exists
  - i.e. we keep only infomrmation that has a link ID on BOTH tables

Left Join	Inner Join
	
result: all LEFT table cases	*result:* only cases matched in BOTH tables
`X %>% left_join(Y)`	`X %>% inner_join(Y)`
`left_join(X, Y)`	`inner_join(X, Y)`

IF no right cases match the left case…

- `left_join()` : Keep the left case and fill in the new variables (from the right table) with `NA`
  - i.e. keep every unique combination in both tables
- `inner_join()` : Discard the left case.
  - i.e. get rid of everything that does not have a match in both tables

IF multiple right cases match the left case…

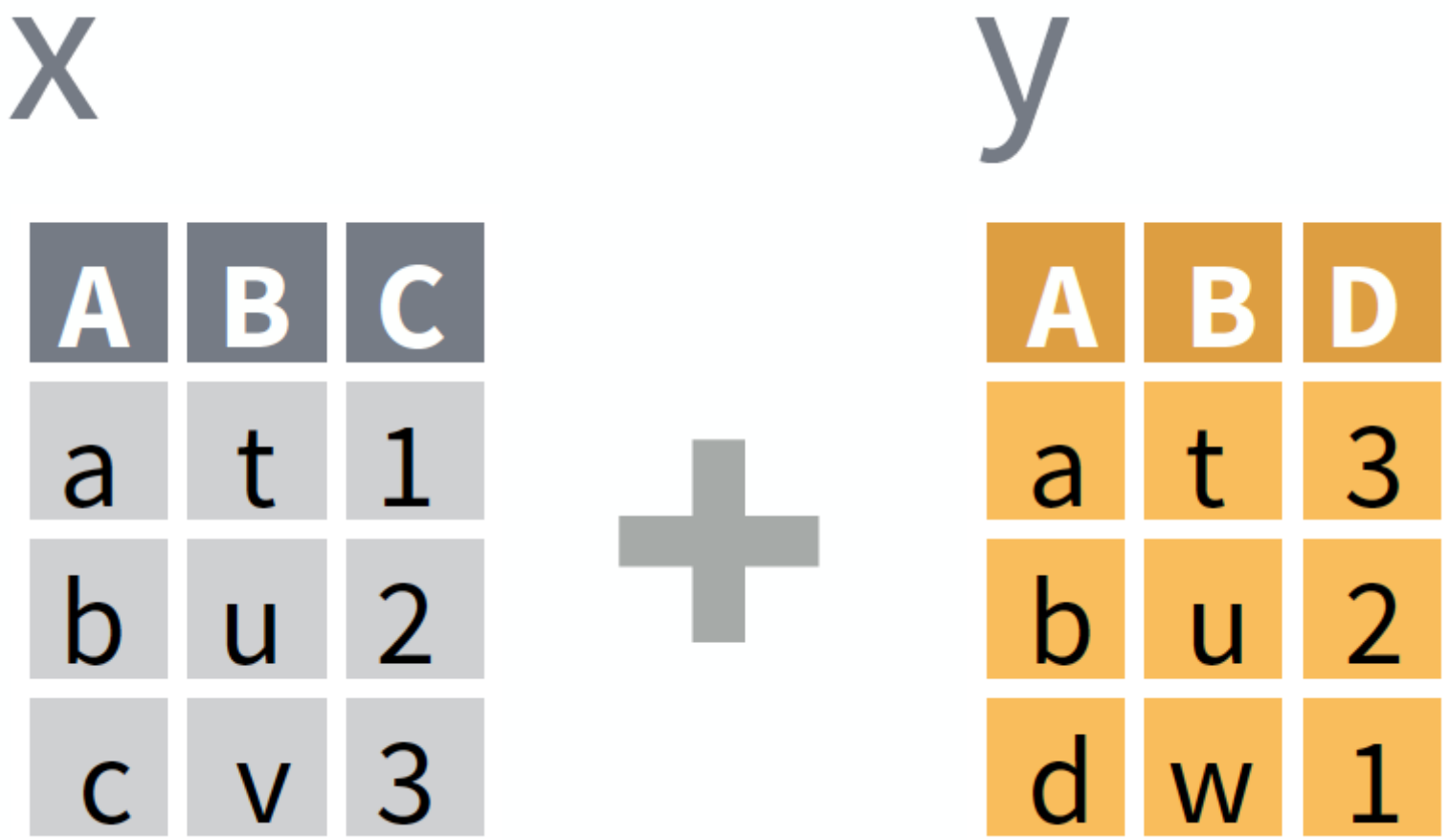
`left_join()` and `inner_join()` do the same thing:

- `left_join()` : Keep **all combinations**.
- `inner_join()` : Keep **all combinations**.

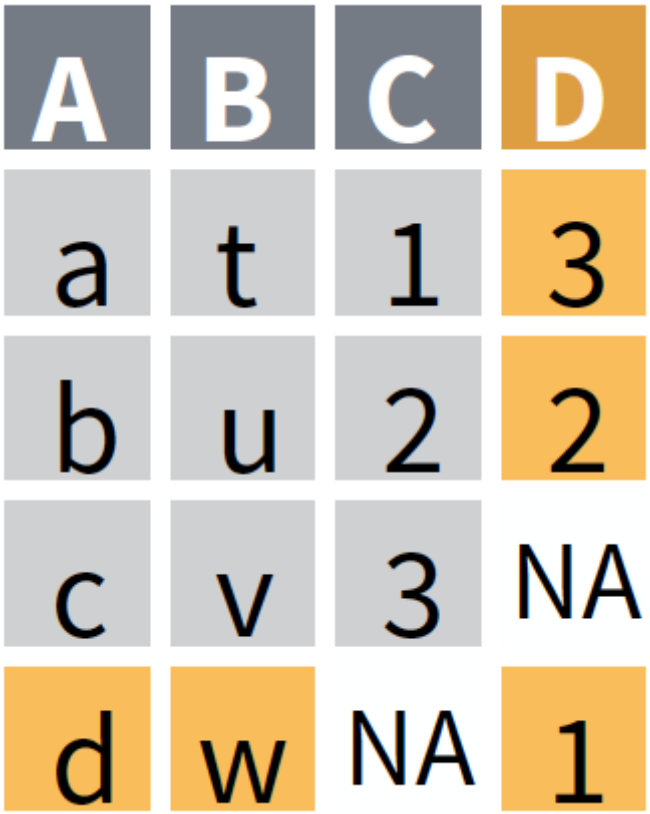
Other useful joins that sometimes come up:

Note: Images adapted from RStudio Data Transformation Cheat Sheet https://rstudio.com/resources/cheatsheets/ (https://rstudio.com/resources/cheatsheets/)

- `full_join()` Keep left case as well as unmatched right cases.
- `right_join(X, Y)` is the same as `left_join(Y, X)`
- Filtering or exploratory joins (See RStudio Cheat Sheet)
    - `semi_join()` Show left cases with a match in the right table (e.g., what WILL be joined)
    - `anti_join()` Discard left cases with a match in the right table (e.g., what will NOT be joined)


x
A B C
a t 1
b u 2
c v 3
y
A B D
a t 3
b u 2
d w 1

Full Join



result: ALL cases, EITHER table

```
X %>% full_join(Y)
```

```
full_join(X, Y)
```

Example: Grade-point averages

Here are three data tables relating student grades in courses at their college

```
Rows: 5902 Columns: 3── Column specification ────────────────────────────────────
Delimiter: ","
chr (3): studentID, grade, lionpathCourseID
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

studentID <chr>	grade <chr>	lionpathCourseID <chr>
S31185	D+	session1784
S31185	B+	session1785
S31185	A-	session1791

3 rows

```
Rows: 1712 Columns: 6── Column specification ────────────────────────────────────
Delimiter: ","
chr (4): lionpathCourseID, dept, sem, instructorID
dbl (2): level, enrollment
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

lionpathCourseID <chr>	dept <chr>	level <dbl>	sem <chr>	enrollment <dbl>	instructorID <chr>
session2686	DS	100	SP2003	263	inst129
session2532	GEOSC	200	FA2002	252	inst202
session2628	GEOSC	200	SP2003	228	inst201

3 rows

```
Rows: 15 Columns: 2── Column specification ────────────────────────────────────────
Delimiter: ","
chr (1): grade
dbl (1): gradepoint
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

grade <chr>	gradepoint <dbl>
A	4.00
A-	3.66
B+	3.33

3 rows

Exercise: Which to Join?

For each of these, say what tables you would need to join and what the corresponding variables will be.

1. How many students in each department?
2. What fraction of grades are below B+ in each department?
3. What's the grade-point average (GPA) for each student?
4. Grade-point average for each department or instructor

Solutions:

1. How many students in each department?

Hide

```
Grades %>%
  left_join(Courses,by = c("lionpathCourseID" = "lionpathCourseID" ))%>%
  group_by(dept)%>%
  summarise(Totalstud = n_distinct(studentID))
```

```
Warning: Detected an unexpected many-to-many relationship between `x` and `y`.
```

dept <chr>	Totalstud <int>
ASTRO	2
BBH	23
BMB	50
CAS	136
CHEM	174
DS	254
DSM	155
EARTH	9
ENGL	76
FIN	137

1-10 of 39 rows Previous **1** 2 3 4 Next

2. What fraction of grades are below B+ in each department?

Hide

```
left_join(Grades, Courses, by = c("lionpathCourseID" = "lionpathCourseID")) %>%
  mutate(BelowB_plus = if_else(grade %in% c("B+", "A-", "A"), 0, 1)) %>%
  group_by(dept) %>%
  summarise(frac_below_B_plus = sum(BelowB_plus) / n() )
```

```
Warning: Detected an unexpected many-to-many relationship between `x` and `y`.
```

dept <chr>	frac_below_B_plus <dbl>
ASTRO	0.0000000
BBH	0.4444444
BMB	0.4626866
CAS	0.2109705
CHEM	0.3679245
DS	0.4297352
DSM	0.2341270
EARTH	0.2500000
ENGL	0.2343750
FIN	0.5298013

3. What's the grade-point average (GPA) for each student?

Hide

```
left_join(Grades, GradePoint) %>%
  group_by(studentID) %>%
  summarise(GPA = mean(gradepoint, na.rm=TRUE))
```

```
Joining with `by = join_by(grade)`
```

studentID <chr>	GPA <dbl>
S31185	2.412500
S31188	3.018125

studentID <chr>	GPA <dbl>
S31191	3.212143
S31194	3.359167
S31197	3.356154
S31200	2.186429
S31203	3.819231
S31206	2.458462
S31209	3.130000
S31212	3.664375

4. Grade-point average for each department

Hide

```
Grad_dept <- left_join(Grades, Courses)
```

```
Joining with `by = join_by(lionpathCourseID)`Warning: Detected an unexpected many-to-many relationship between `x` and `y`.
```

Hide

```
left_join(Grad_dept, GradePoint) %>%
  group_by(dept) %>%
  summarise(avgGPA = mean(gradepoint, na.rm = TRUE))
```

```
Joining with `by = join_by(grade)`
```

dept <chr>	avgGPA <dbl>
ASTRO	3.495000
BBH	3.234762
BMB	3.242787
CAS	3.524026
CHEM	3.496415
DS	3.314192
DSM	3.559016
EARTH	3.637500
ENGL	3.480583
FIN	3.271765

Hide

```
#equivalently
# We can do this because we know there are not missing values in GradePoint
left_join(Grades, GradePoint) %>%
  left_join(Courses) %>%
  group_by(dept) %>%
  summarise(avgGPA = mean(gradepoint, na.rm = TRUE))
```

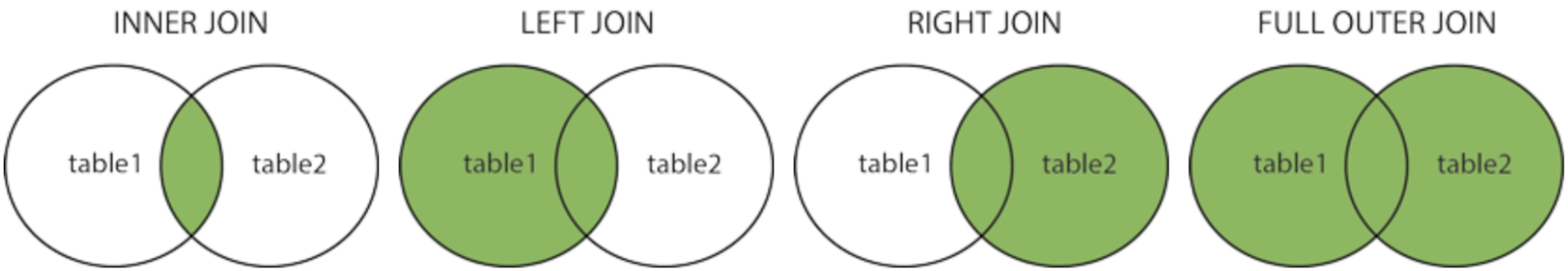
```
Joining with `by = join_by(grade)`Joining with `by = join_by(lionpathCourseID)`Warning: Detected an unexpected many-to-many
relationship between `x` and `y`.
```

dept <chr>	avgGPA <dbl>
ASTRO	3.495000
BBH	3.234762
BMB	3.242787
CAS	3.524026
CHEM	3.496415
DS	3.314192
DSM	3.559016
EARTH	3.637500
ENGL	3.480583
FIN	3.271765

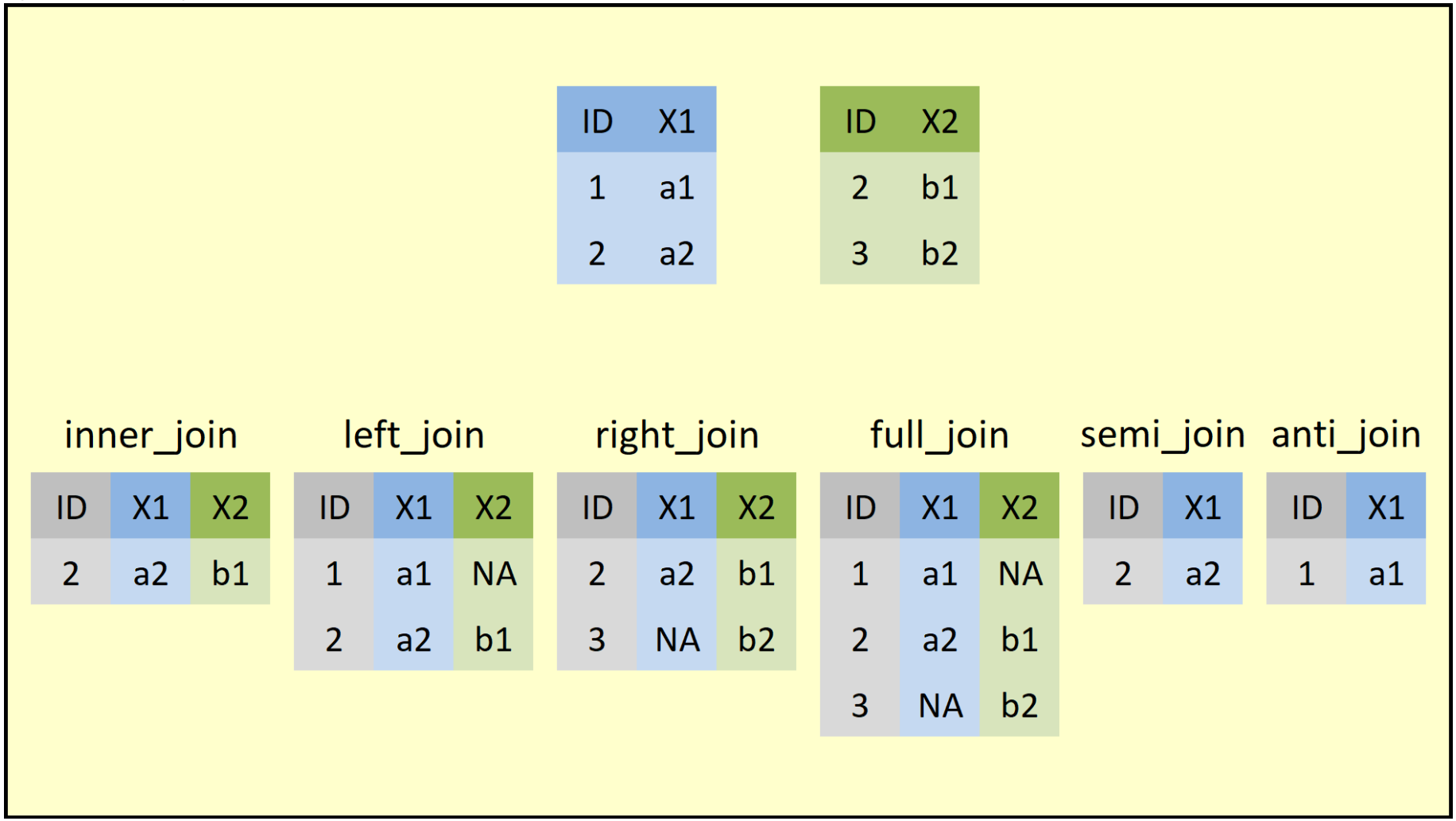
Other resources

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



A website I find very helpful: https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti (https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti).



Assignments

- Reading Quiz Chapter 08 due 9:59 am, Wednesday, July 19
- Reading Quiz Chapters 10 and 11 due 9:59am, Friday, July 21
- Activity: PopularNames due 9:59am, Friday, July 21