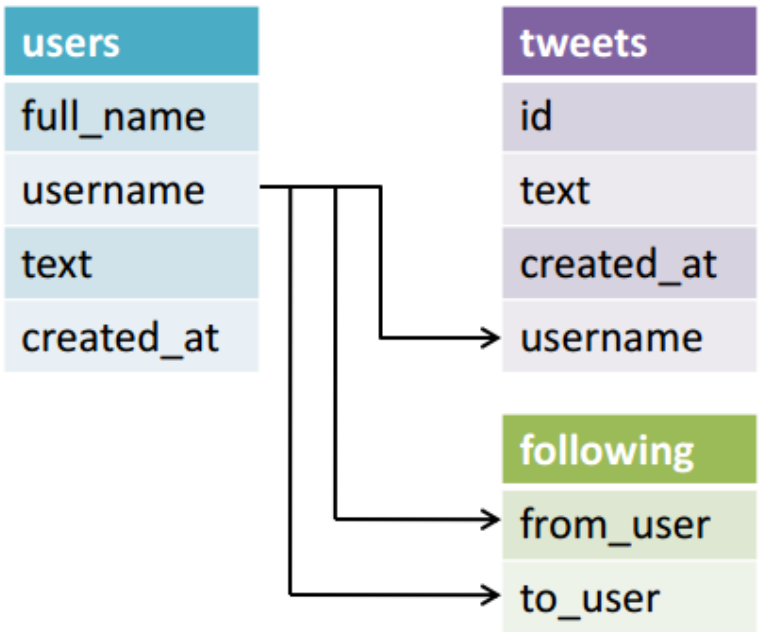
Subsection 2

Relational Data in R

Relational Data with dplyr

- It often happens that the data being analyzed exists in multiple tables of data
- The difficulty arises when trying to combine these disparate tables into a single, cohesive table on which analyses can be conducted



Relational Data with dplyr [CONT'D]

- A very common place for these various data tables is in a relational database, e.g., an SQL database (structured query language)
- Although R can connect directly and query from a variety of relational databases, this is beyond the scope of this course
- Using functions from the dplyr package, data frames can be joined to conduct additionally complex analyses
- There are three types of operations on relational data
  - mutating joins
  - filtering joins
  - set operations

n.b. Due to relational nature of the data examined in this section, the terms data frame/tibble and table will be used interchangeably

Mutating Joins

- A mutating join combines variables from two tables based on a matched **key** value

E.g. To join the following two tables based on the key `x1`, there are a number of possibilities

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

- Retain rows of both tables **only** for rows with matching keys
- Join matching rows from table b to a by matching key
- Join matching rows from table a to b by matching key
- Retain all values in all rows by matching key

Mutating Joins [CONT'D]

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.

Sample Customer/Transaction Tibbles for Example Joins

```
> (custTbl <- read_csv("mutatingJoins_ex01_table_01.csv"))
# A tibble: 6  4
  cust_id first_name   last_name gender
    <int>      <chr>       <chr>  <chr>
1  149201     Starla Godilington      F
2  136127     Barris     Stivani      M
3  389922    Rubetta   Schermick      F
4  836399     Elnore       Upham      F
5   83025     Merrel      Braney      M
6  763354     Fosser         Ian      M


> (transactionTbl <- read_csv("mutatingJoins_ex01_table_02.csv"))
# A tibble: 6  5
     id creditCardNum creditCardTypes transactionAmt cust_id
  <int>         <dbl>           <chr>          <chr>   <int>
1     1  3.575026e+15             jcb         $3.82  149201
2     2  3.553533e+15             jcb         $7.90  136127
3     3  3.050191e+13         blanche         $9.18  389922
4     4  4.912087e+12            visa         $2.91  836399
5     5  3.560840e+15             jcb         $9.94   83025
6     6  4.565765e+15            visa         $6.22   39021
```

- How much did each customer spend?

Sample Customer/Transaction Tibbles for Example Joins

```
  cust_id first_name   last_name gender
    <int>      <chr>       <chr>  <chr>
1  149201     Starla Godilington      F
2  136127     Barris     Stivani      M
3  389922    Rubetta   Schermick      F
4  836399     Elnore       Upham      F
5   83025     Merrel      Braney      M
6  763354     Fosser         Ian      M

     id creditCardNum creditCardTypes transactionAmt cust_id
  <int>         <dbl>           <chr>          <chr>   <int>
1     1  3.575026e+15             jcb          $3.82  149201
2     2  3.553533e+15             jcb          $7.90  136127
3     3  3.050191e+13         blanche          $9.18  389922
4     4  4.912087e+12            visa          $2.91  836399
5     5  3.560840e+15             jcb          $9.94   83025
6     6  4.565765e+15            visa          $6.22   39021
```

- `cust_id` 763354 exists in `custTbl` but does not have any transactions recorded in `transactionTbl`
- `id == 6` in `transactionTbl` indicates a transaction by `cust_id` 39021, but `cust_id` 39021 does not exist in `custTbl`

dplyr::inner_join()

- Inner joins join **only** the rows of two data frames that share a matched key

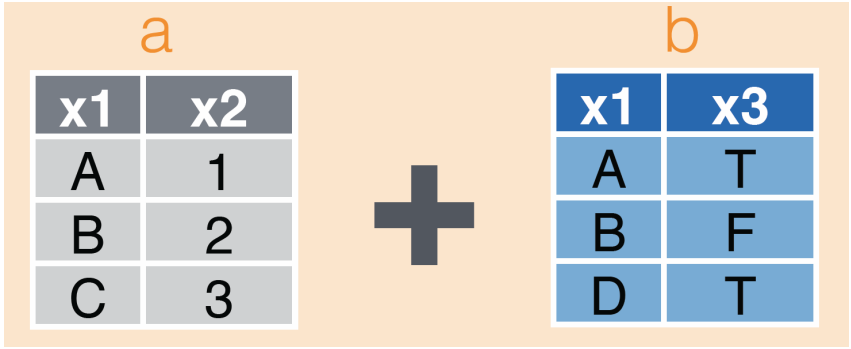
```
inner_join(custTbl, transactionTbl, by = "cust_id") %>% select(-id, -creditCardNum)

# A tibble: 5  6
  cust_id first_name   last_name gender creditCardTypes transactionAmt
    <int>      <chr>       <chr>  <chr>           <chr>          <chr>
1  149201     Starla Godilington      F             jcb          $3.82
2  136127     Barris     Stivani      M             jcb          $7.90
3  389922    Rubetta   Schermick      F         blanche          $9.18
4  836399     Elnore       Upham      F            visa          $2.91
5   83025     Merrel      Braney      M             jcb          $9.94
```

- The same result could have been obtained without using a join, but it would have been far more lengthy and cumbersome

n.b. Tibbles were joined on `cust_id` key, therefore the record in `custTbl` with `cust_id == 763354` has been omitted from the join, as well as the record in `custTbl` which does not have any transactions associated with `cust_id == 763354`

Outer Joins

Whereas **inner** joins keep rows that have key matches in both tables being joined, **outer** joins keep rows that exist in one or the other (or both) tables being joined.



Outer joins come in three variants (assume the key is `x1`):

- A **left** join would keep all rows in `a`
- A **right** join would keep all rows in `b`
- A **full** join would keep all rows in both `a` and `b`

dplyr::left_join()

```
left_join(custTbl, transactionTbl, by = "cust_id") %>% select(-id, -creditCardNum)

# A tibble: 6  6
  cust_id first_name   last_name gender creditCardTypes transactionAmt
    <int>      <chr>       <chr>  <chr>           <chr>          <chr>
1  149201      Starla Godilington      F             jcb          $3.82
2  136127      Barris     Stivani      M             jcb          $7.90
3  389922     Rubetta   Schermick      F         blanche          $9.18
4  836399      Elnore       Upham      F            visa          $2.91
5   83025      Merrel      Braney      M             jcb          $9.94
6  763354      Fosser         Ian      M            <NA>           <NA>
```

- All rows in `custTbl` have been retained
- Where there were no matches for `cust_id` from `custTbl` to `transactionTbl`, `<NA>`s were inserted
- Where there were no matches for `cust_id` from `transactionTbl` to `custTbl`, rows were omitted

dplyr::right_join()

```
right_join(custTbl, transactionTbl, by = "cust_id") %>% select(-id, -creditCardNum)
```

```
# A tibble: 6  6
  cust_id first_name   last_name gender creditCardTypes transactionAmt
    <int>      <chr>       <chr>  <chr>           <chr>          <chr>
1  149201      Starla Godilington      F             jcb          $3.82
2  136127      Barris     Stivani      M             jcb          $7.90
3  389922     Rubetta   Schermick      F         blanche          $9.18
4  836399      Elnore       Upham      F            visa          $2.91
5   83025      Merrel      Braney      M             jcb          $9.94
6   39021        <NA>        <NA>   <NA>            visa          $6.22
```

- All rows in `transactionTbl` have been retained
- Where there were no matches for `cust_id` from `transactionTbl` to `custTbl`, `<NA>`s were inserted
- Where there were no matches for `cust_id` from `custTbl` to `transactionTbl`, rows were omitted

left_join() vs. right_join()

The following code snippets generate identical output, by rearranging the order of the arguments passed to the *_join's

```
> left_join(transactionTbl, custTbl, by = "cust_id") %>%
    select(-id, -creditCardNum)
# A tibble: 6  6
  creditCardTypes transactionAmt cust_id first_name   last_name gender
            <chr>          <chr>   <int>      <chr>       <chr>  <chr>
1             jcb          $3.82  149201     Starla Godilington      F
2             jcb          $7.90  136127     Barris     Stivani      M
3         blanche          $9.18  389922    Rubetta   Schermick      F
4            visa          $2.91  836399     Elnore       Upham      F
5             jcb          $9.94   83025     Merrel      Braney      M
6            visa          $6.22   39021       <NA>        <NA>   <NA>

> right_join(custTbl, transactionTbl, by = "cust_id") %>%
    select(-id, -creditCardNum)
# A tibble: 6  6
  cust_id first_name   last_name gender creditCardTypes transactionAmt
    <int>      <chr>       <chr>  <chr>           <chr>          <chr>
1  149201     Starla Godilington      F             jcb          $3.82
2  136127     Barris     Stivani      M             jcb          $7.90
3  389922    Rubetta   Schermick      F         blanche          $9.18
4  836399     Elnore       Upham      F            visa          $2.91
5   83025     Merrel      Braney      M             jcb          $9.94
6   39021       <NA>        <NA>   <NA>            visa          $6.22
```

dplyr::full_join()

```
full_join(custTbl, transactionTbl, by = "cust_id") %>% select(-id, -creditCardNum)
```

```
# A tibble: 7  6
  cust_id first_name   last_name gender creditCardTypes transactionAmt
    <int>      <chr>       <chr>  <chr>           <chr>          <chr>
1  149201     Starla Godilington      F             jcb          $3.82
2  136127     Barris     Stivani      M             jcb          $7.90
3  389922    Rubetta   Schermick      F         blanche          $9.18
4  836399     Elnore       Upham      F            visa          $2.91
5   83025     Merrel      Braney      M             jcb          $9.94
6  763354     Fosser         Ian      M            <NA>           <NA>
7   39021       <NA>        <NA>   <NA>            visa          $6.22
```

- A full join retains all rows from both tables, inserting <NA>s where there were no cust_id matches

Filtering Joins and Set Operations

dplyr offers other functions that may help in manipulating relational data

- `semi_join(a, b)` **keeps** only rows in `a` that match with `b` (but variables from `b` are not included)
- `anti_join(a, b)` **omits** only rows in `a` that match with `b` (but variables from `b` are not included)
- `intersect(a, b)` returns rows that are in **both** `a` and `b`
- `union(a, b)` returns rows that are in `a` or `b` **or both**
- `setdiff(a, b)` returns rows that are in `a` but not in `b`

LAB

LAB: `dplyr` and `*_joins`

First we download the `JSON` data set `dplyr_joins_lab.json` and explore it.

This is nested JSON data, which requires a little more code to flatten into a workable data frame.

We want to explore, among other things, the classes of purchases made by customer, so we will import join `classes.csv` to our JSON data we might get more descriptive data on purchases.

We can now explore our data and answer some interesting questions.

The categories, ordered by frequency of purchase, arranged in descending order and excluding `NA`s are

Class	Frequency
cleaningSupplies	748
petSupplies	736