Joining datasets

Abhijit Dasgupta, PhD

Goals

- Learn how to join data sets (merging)

Data

This data set is taken from a breast cancer proteome database available here and modified for this exercise.

- Clinical data: data/BreastCancer_Clinical.xlsx
- Proteome data: data/BreastCancer_Expression.xlsx

> These data are available in the class data folder/link, and the expectation is that you save them to the `data` folder of your project.

Joins

Putting data sets together

- Quite often, data on individuals lie in different tables
    - Clinical, demographic and bioinformatic data
    - Drug, procedure, and payment data (think Medicare)
    - Personal health data across different healthcare entities

Joining data sets

The simplest case is when we just need to add more data to existing data

- New patients in study, with same protocol (add rows)
- Adding pathology, imaging data for existing patients (add columns)

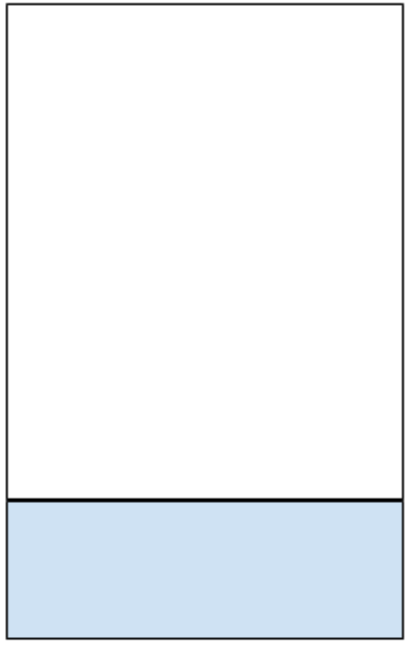
Joining data sets

`cbind(x,y)`

Add columns

Data sets have same subjects/observations, but new variables

`rbind(x,y)`



Add rows

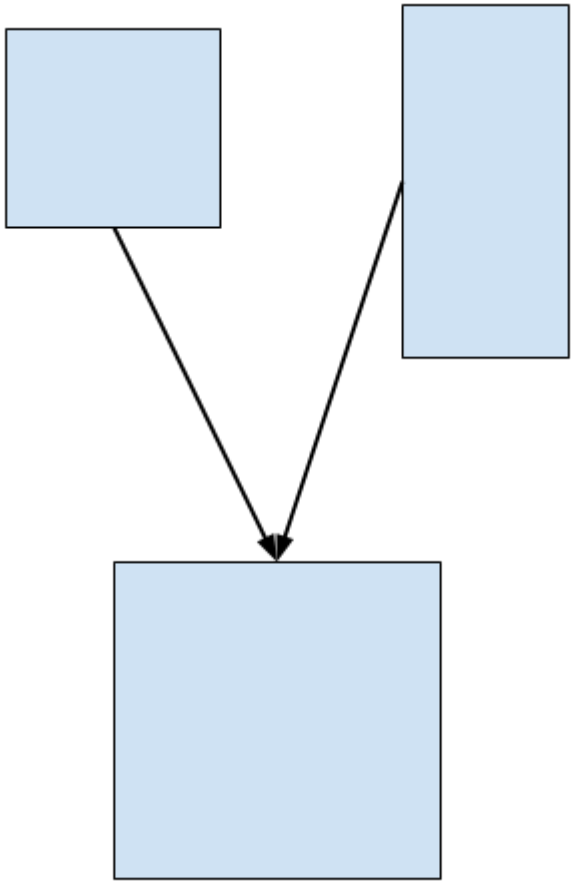
Data sets have same variables, but new subjects

Joining data sets

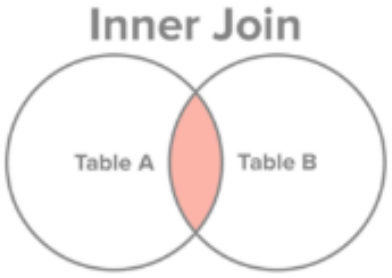
We will talk about more general ways of joining two datasets

We will assume:

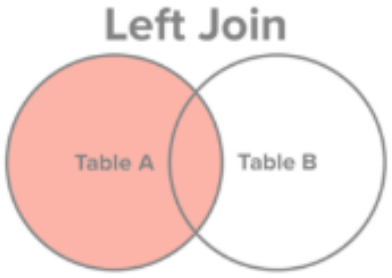
1. We have two rectangular data sets (so `data.frame` or `tibble`)
2. There is at least one variable (column) in common, even if they have different names
    - Patient ID number
    - SSN (Social Security number)
    - Identifiable information



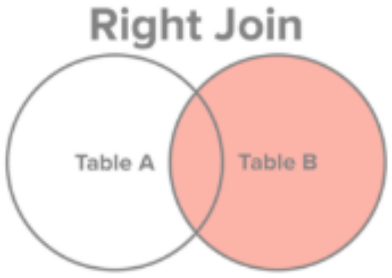
Joining data sets



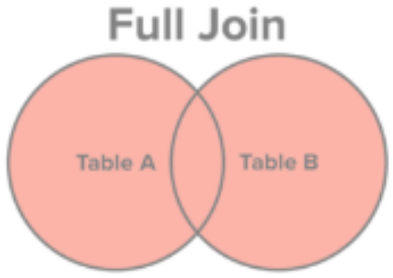
Select all records from Table A and Table B, where the join condition is met.



Select all records from Table A, along with records from Table B for which the join condition is met (if at all).



Select all records from Table B, along with records from Table A for which the join condition is met (if at all).



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

inner_join	left_join	right_join	outer_join

The "join condition" are the common variables in the two datasets, i.e. rows are selected if the values of the common variables in the left dataset matches the values of the common variables in the right dataset

These functions are available in the **dplyr** package.

A breast cancer example

```
library(readxl)
clinical <- read_excel('data/BreastCancer_Clinical.xlsx',
                       .name_repair = 'universal')
proteome <- read_excel('data/BreastCancer_Expression.xlsx',
                       .name_repair = 'universal')
```

```
clinical
```

```
# A tibble: 105 x 30
   Complete.TCGA.ID Gender Age.at.Initial.… ER.Stat
   <chr>            <chr>              <dbl> <chr>
 1 TCGA-A2-A0T2     FEMALE                66 Negativ
 2 TCGA-A2-A0CM     FEMALE                40 Negativ
 3 TCGA-BH-A18V     FEMALE                48 Negativ
 4 TCGA-BH-A18Q     FEMALE                56 Negativ
 5 TCGA-BH-A0E0     FEMALE                38 Negativ
 6 TCGA-A7-A0CE     FEMALE                57 Negativ
 7 TCGA-D8-A142     FEMALE                74 Negativ
 8 TCGA-A2-A0D0     FEMALE                60 Negativ
 9 TCGA-AO-A0J6     FEMALE                61 Negativ
10 TCGA-A2-A0YM     FEMALE                67 Negativ
# … with 95 more rows, and 26 more variables:
#   PR.Status <chr>, HER2.Final.Status <chr>, Tumor
#   Tumor..T1.Coded <chr>, Node <chr>, Node.Coded <
#   Metastasis <chr>, Metastasis.Coded <chr>,
#   AJCC.Stage <chr>, Converted.Stage <chr>,
```

```
proteome
```

```
# A tibble: 83 x 11
   TCGA_ID NP_958782 NP_958785 NP_958786 NP_000436
   <chr>       <dbl>     <dbl>     <dbl>     <dbl>
 1 TCGA-A…     1.10      1.11      1.11      1.11
 2 TCGA-C…     2.61      2.65      2.65      2.65
 3 TCGA-A…    -0.660    -0.649    -0.654    -0.632
 4 TCGA-B…     0.195     0.215     0.215     0.205
 5 TCGA-C…    -0.494    -0.504    -0.501    -0.510
 6 TCGA-C…     2.77      2.78      2.78      2.80
 7 TCGA-E…     0.863     0.870     0.870     0.866
 8 TCGA-C…     1.41      1.41      1.41      1.41
 9 TCGA-A…     1.19      1.19      1.19      1.19
10 TCGA-A…     1.10      1.10      1.10      1.10
# … with 73 more rows, and 6 more variables:
#   NP_958781 <dbl>, NP_958780 <dbl>, NP_958783 <db
#   NP_958784 <dbl>, NP_112598 <dbl>, NP_001611 <db
```

A breast cancer example

```
library(readxl)
clinical <- read_excel('data/BreastCancer_Clinical.xlsx',
                       .name_repair = 'universal')
proteome <- read_excel('data/BreastCancer_Expression.xlsx',
                       .name_repair = 'universal')
```

```
clinical[,1:2]
```

```
# A tibble: 105 x 2
   Complete.TCGA.ID Gender
   <chr>            <chr>
 1 TCGA-A2-A0T2     FEMALE
 2 TCGA-A2-A0CM     FEMALE
 3 TCGA-BH-A18V     FEMALE
 4 TCGA-BH-A18Q     FEMALE
 5 TCGA-BH-A0E0     FEMALE
 6 TCGA-A7-A0CE     FEMALE
 7 TCGA-D8-A142     FEMALE
 8 TCGA-A2-A0D0     FEMALE
 9 TCGA-AO-A0J6     FEMALE
10 TCGA-A2-A0YM     FEMALE
# … with 95 more rows
```

```
proteome[,1:2]
```

```
# A tibble: 83 x 2
   TCGA_ID      NP_958782
   <chr>            <dbl>
 1 TCGA-AO-A12D     1.10
 2 TCGA-C8-A131     2.61
 3 TCGA-AO-A12B    -0.660
 4 TCGA-BH-A18Q     0.195
 5 TCGA-C8-A130    -0.494
 6 TCGA-C8-A138     2.77
 7 TCGA-E2-A154     0.863
 8 TCGA-C8-A12L     1.41
 9 TCGA-A2-A0EX     1.19
10 TCGA-AO-A12D     1.10
# … with 73 more rows
```

We see that both have the same ID variable, but with different names and different orders

A breast cancer example

Let's make sure that the ID's are truly IDs, i.e. each row has a unique value

```
length(unique(clinical$Complete.TCGA.ID)) == nrow(clinical)
```

```
  [1] TRUE
```

```
length(unique(proteome$TCGA_ID)) == nrow(proteome)
```

```
  [1] FALSE
```

Data example

For convenience we'll keep the first instance for each ID in the `proteome` data

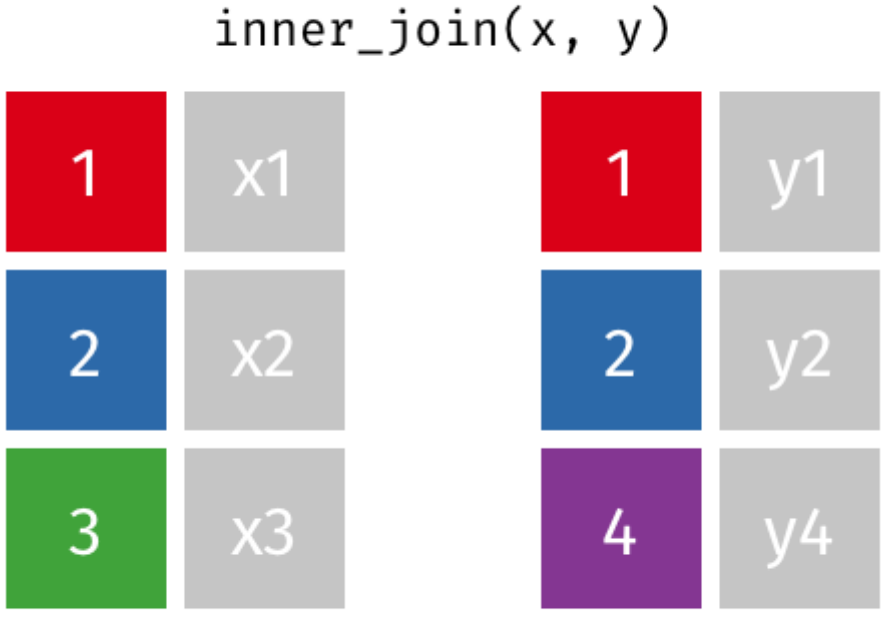
```
proteome <- proteome %>% filter(!duplicated(TCGA_ID))
```

> `duplicated` = TRUE if a previous row contains the same value

```
length(unique(proteome$TCGA_ID)) == nrow(proteome)
```

```
[1] TRUE
```

Inner join



- Keep only rows that have common ids between the two data, and add columns
- The joined data will have no more rows than either data, but more columns than each

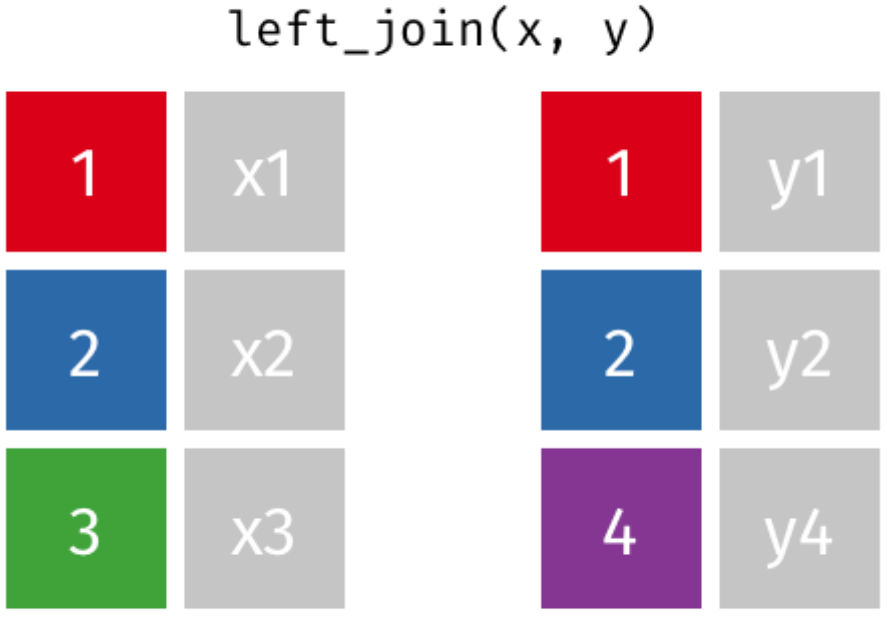
Inner join

```
common_rows <- inner_join(clinical[,1:6], proteome,
                          by=c('Complete.TCGA.ID'='TCGA_ID'))
```

```
# A tibble: 77 x 16
   Complete.TCGA.ID Gender Age.at.Initial.… ER.Status
   <chr>            <chr>             <dbl> <chr>
 1 TCGA-A2-A0CM     FEMALE               40 Negative
 2 TCGA-BH-A18Q     FEMALE               56 Negative
 3 TCGA-A7-A0CE     FEMALE               57 Negative
 4 TCGA-D8-A142     FEMALE               74 Negative
 5 TCGA-AO-A0J6     FEMALE               61 Negative
 6 TCGA-A2-A0YM     FEMALE               67 Negative
 7 TCGA-A2-A0D2     FEMALE               45 Negative
 8 TCGA-A2-A0SX     FEMALE               48 Negative
 9 TCGA-AO-A0JL     FEMALE               59 Negative
10 TCGA-AO-A12F     FEMALE               36 Negative
# … with 67 more rows, and 12 more variables:
#   PR.Status <chr>, HER2.Final.Status <chr>,
#   NP_958782 <dbl>, NP_958785 <dbl>, NP_958786 <dbl>,
#   NP_000436 <dbl>, NP_958781 <dbl>, NP_958780 <dbl>,
#   NP_958783 <dbl>, NP_958784 <dbl>, NP_112598 <dbl>,
#   NP_001611 <dbl>
```

Note that we have all the columns from both datasets, but only the common set of IDs from the two datasets

Left join



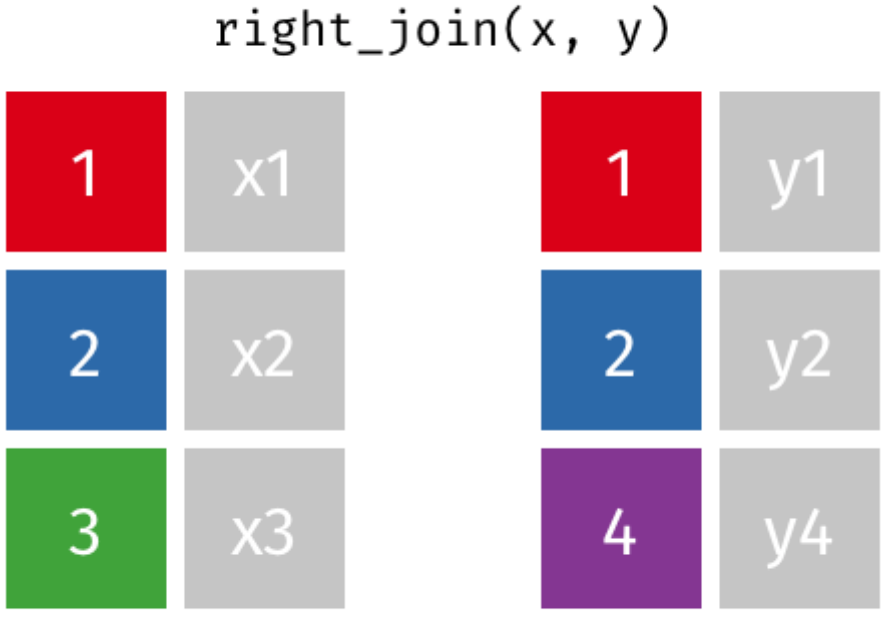
- Keep all rows of left data, add columns from right data only for rows with matching IDs
- If a row in left data has no corresponding row in the right data, the corresponding entries in the joined data are replaced by NA
- Joined data has same number of rows as left data, but more columns.

Left join

```
left_rows <- left_join(clinical[,1:6], proteome, by=c('Complete.TCGA.ID'='TCGA_ID'))
```

```
# A tibble: 105 x 16
  Complete.TCGA.ID Gender
  <chr>            <chr>
1 TCGA-A2-A0T2     FEMALE
2 TCGA-A2-A0CM     FEMALE
3 TCGA-BH-A18V     FEMALE
  Age.at.Initial.Pathologic.Diagnosis ER.Status PR.Status
                                <dbl> <chr>     <chr>
1                                  66 Negative  Negative
2                                  40 Negative  Negative
3                                  48 Negative  Negative
  HER2.Final.Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative             NA        NA        NA        NA
2 Negative              0.683     0.694     0.698     0.687
3 Negative             NA        NA        NA        NA
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1    NA        NA        NA        NA        NA
2     0.687     0.698     0.698     0.698     -2.65
3    NA        NA        NA        NA        NA
  NP_001611
      <dbl>
1    NA
2    -0.984
3    NA
```

Right join

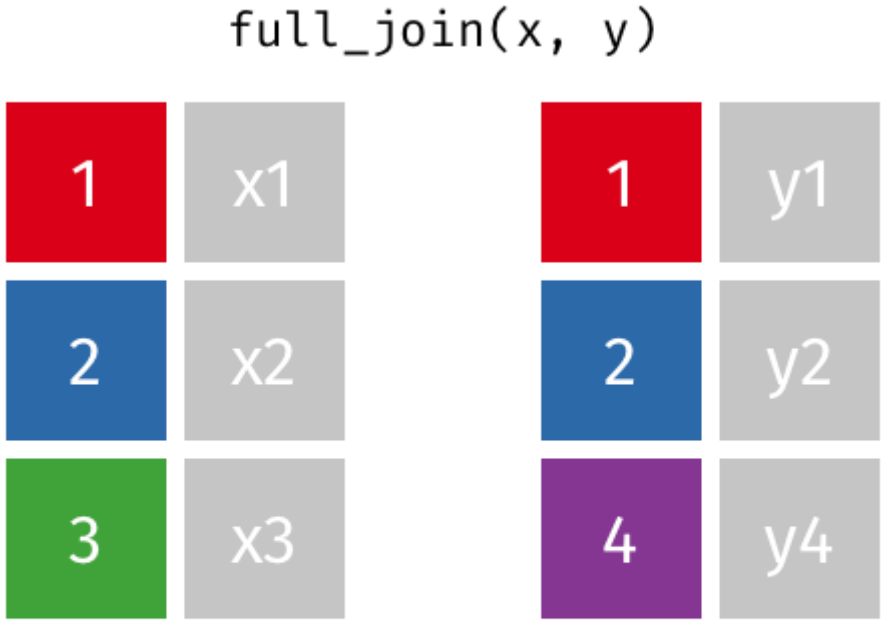


- Keep all the rows of the *right* data, add corresponding rows of left data *on the left*
- Once again, if there are rows of right data that do not have corresponding rows in left data, the entries are filled with NA
- The joined data has the same number of rows as the right data, but more columns (attached to its left). The order of the columns is the columns of the left data followed by the columns of the right data

Right join

```
right_rows <- right_join(clinical[,1:6], proteome, by=c('Complete.TCGA.ID'='TCGA_ID'))
```

```
# A tibble: 80 x 16
  Complete.TCGA.ID Gender
  <chr>            <chr>
1 TCGA-A2-A0CM     FEMALE
2 TCGA-BH-A18Q     FEMALE
3 TCGA-A7-A0CE     FEMALE
  Age.at.Initial.Pathologic.Diagnosis ER.Status PR.Status
                                <dbl> <chr>     <chr>
1                                  40 Negative  Negative
2                                  56 Negative  Negative
3                                  57 Negative  Negative
  HER2.Final.Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative              0.683     0.694     0.698     0.687
2 Negative              0.195     0.215     0.215     0.205
3 Negative             -1.12     -1.12     -1.12     -1.13
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     0.687     0.698     0.698     0.698     -2.65
2     0.215     0.215     0.215     0.215     -1.04
3    -1.13     -1.12     -1.12     -1.12       2.24
  NP_001611
      <dbl>
1    -0.984
2    -0.517
3    -2.58
```



This is the *kitchen sink* join

- All rows of the left and right data are included
- Non-corresponding entries are filled with NA
- The joined data set has at least as many rows as the larger of the two data, and more columns than either data.

Outer/Full Join

```
full_rows <- full_join(clinical[,1:6], proteome, by=c('Complete.TCGA.ID'='TCGA_ID'))
```

```
# A tibble: 108 x 16
  Complete.TCGA.ID Gender
  <chr>            <chr>
1 TCGA-A2-A0T2     FEMALE
2 TCGA-A2-A0CM     FEMALE
3 TCGA-BH-A18V     FEMALE
  Age.at.Initial.Pathologic.Diagnosis ER.Status PR.Status
                                <dbl> <chr>     <chr>
1                                  66 Negative  Negative
2                                  40 Negative  Negative
3                                  48 Negative  Negative
  HER2.Final.Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative             NA        NA        NA        NA
2 Negative              0.683     0.694     0.698     0.687
3 Negative             NA        NA        NA        NA
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1    NA        NA        NA        NA        NA
2     0.687     0.698     0.698     0.698     -2.65
3    NA        NA        NA        NA        NA
  NP_001611
      <dbl>
1    NA
2    -0.984
3    NA
```

Joins

In each of `inner_join`, `left_join`, `right_join` and `full_join`, the number of columns always increases

There are also two joins where the number of columns don't increase. They aren't really "joins" in that sense, but really fancy filters on a dataset

```
tbl <- tribble(~Join,~Use,~Description,
               "semi_join", "semi_join(A,B)", "Keep rows in A where ID matches some ID value in B",
               'anti_join', 'anti_join(A,B)', 'Keep rows in A where ID does NOT match any ID value in B')
knitr::kable(tbl, format='html')
```

Join	Use	Description
semi_join	semi_join(A,B)	Keep rows in A where ID matches some ID value in B
anti_join	anti_join(A,B)	Keep rows in A where ID does NOT match any ID value in B

These just filter the rows of `A` without adding any columns of `B`. These can be faster than `dplyr::filter` when dealing with large data sets

Putting it in a pipe

```
final_data <- clinical %>%
  inner_join(proteome, by=c("Complete.TCGA.ID"="TCGA_ID")) %>%
  filter(Gender =='FEMALE') %>%
  select(Complete.TCGA.ID, Age.at.Initial.Pathologic.Diagnosis, ER.Status,
         starts_with("NP")) # grabs all the protein data
```

```
  # A tibble: 75 x 13
    Complete.TCGA.ID Age.at.Initial.Pathologic.Diagnosis
    <chr>                                          <dbl>
  1 TCGA-A2-A0CM                                      40
  2 TCGA-BH-A18Q                                      56
  3 TCGA-A7-A0CE                                      57
    ER.Status NP_958782 NP_958785 NP_958786 NP_000436
    <chr>         <dbl>     <dbl>     <dbl>     <dbl>
  1 Negative      0.683     0.694     0.698     0.687
  2 Negative      0.195     0.215     0.215     0.205
  3 Negative     -1.12     -1.12     -1.12     -1.13
    NP_958781 NP_958780 NP_958783 NP_958784 NP_112598
        <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
  1     0.687     0.698     0.698     0.698     -2.65
  2     0.215     0.215     0.215     0.215     -1.04
  3    -1.13     -1.12     -1.12     -1.12       2.24
    NP_001611
        <dbl>
  1    -0.984
  2    -0.517
  3    -2.58
  # … with 72 more rows
```

Some notes

- Joins are very much in the spirit of using SQL in databases
- In SAS, if you use `MERGE` in the `DATA` step to create merged variables, you need to sort the data by the common variables
    - This is a very expensive operation computationally
    - In SAS, you can avoid this by using `PROC SQL`
    - In R, this sorting is not necessary
- Learning to join data sets efficiently is one of the coolest skills of a data scientist, and makes life infinitely easier

Example code: Joining many datasets together

Requirement: Pull together over 200 datasets of variant alleles and expressions (1 per subject/cell line)

```
library(dplyr)

fnames <- dir('~/Desktop/Sreya', full.names = TRUE) # Grab and store the paths to the individual files
ids <- stringr::str_extract(fnames, '[:alnum:]+') # The file names have the subject ids in them
                                                  # as first bit of the string

## Data ingestion
data_corpus <- purrr::map(fnames, read_delim, delim='\t') # Creates a list of raw datasets

## Data munging
for (i in 1:length(data_corpus)){
  data_corpus[[i]] <- data_corpus[[i]] %>% # Note [[]] since I'm manipulating lists
    select(`Variant Allele`, HF) %>% # Keep only allele name and expression
    set_names("variant_allele", ids[i]) %>% # change column names to `variant_allele` and subject ID
    mutate(variant_allele = str_trim(variant_allele)) # Getting rid of extra spaces
}

## Data joining
data_merged <- Reduce(full_join, data_corpus) # Here is the join. This works since
                                              # all the data sets have only `variant_allele` in common
```

> We haven't seen two functions here: `purrr::map` and `Reduce`. I won't go into details here, but see the short version on next slide. Also notice that the number of files to be joined is never specified in the code. This could work for any number of files

Example code: Joining many datasets together

- The `map` function acts on a list (first argument) and applies a function (2nd argument) to each element, storing the result in a list the same size as the first argument. You could replace the map function with a for loop, but map is provably more efficient computationally. It is worth thinking about map like a for loop, though. Nice tutorial

- `Reduce` is a very powerful function that is one of the functional programming functions in R, i.e., it is a function that acts on functions. It takes as inputs a function (in our case, `full_join`), and a list (in our case, data_corpus). The input function should take two arguments of the same type, as `full_join` does, and Reduce goes through the list, applying the function to the first two elements of the list, then to the result and the 3rd element, then to the result and 4th element,and so on.